内 容 简 介

《Java 从入门到精通（实例版）》从初学者角度出发，通过通俗易懂的语言、丰富多彩的实例，详细介绍了进行 Java 应用程序开发应该掌握的各方面技术。全书共 21 章，包括初识 Java，熟悉 Eclipse 开发工具，Java 语言基础，流程控制，数组，字符串，类和对象，接口、继承与多态，类的高级特性，Java 集合类，异常处理，输入/输出，Swing 程序设计，高级事件处理，多线程，网络通信，JDBC 操作数据库，Swing 高级组件，高级布局管理器，AWT 绘图技术，酒店管理系统等。所有知识都结合具体实例进行介绍，涉及的程序代码给出了详细的注释，可以使读者轻松领会 Java 应用程序开发的精髓，快速提高开发技能。另外，本书除了纸质内容之外，配书光盘中还给出了海量开发资源库，主要内容如下：

- ☑ 语音视频讲解：总时长 14 小时，共 94 段
- ☑ 实例资源库：732 个实例及源码详细分析
- ☑ 模块资源库：15 个经典模块开发过程完整展现
- ☑ 项目案例资源库：15 个企业项目开发过程完整展现
- ☑ 测试题库系统：616 道能力测试题目
- ☑ 面试资源库：369 个企业面试真题
- ☑ PPT 电子教案

本书适合作为软件开发入门者的自学用书，也适合作为高等院校相关专业的教学参考书，也可供开发人员查阅、参考。

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 从入门到精通（实例版）/明日科技编著. —北京：清华大学出版社，2012.9
（软件开发视频大讲堂）

ISBN 978-7-302-28754-4

I. ①J…　II. ①明…　III. ①Java 语言-程序设计　IV. ①TP312

中国版本图书馆 CIP 数据核字（2012）第 089424 号

责任编辑：赵洛育
封面设计：刘洪利
版式设计：文森时代
责任校对：柴　燕
责任印制：何　芊

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn，http://www.wqbook.com
地　　址：北京清华大学学研大厦 A 座　　**邮　　编**：100084
社 总 机：010-62770175　　**邮　　购**：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质 量 反 馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 刷 者：清华大学印刷厂
装 订 者：北京市密云县京文制本装订厂
经　　销：全国新华书店
开　　本：203mm×260mm　　**印　张**：36.75　　**字　　数**：986 千字
（附海量开发资源库 DVD1 张）
版　　次：2012 年 9 月第 1 版　　**印　　次**：2012 年 9 月第1 次印刷
印　　数：1～6000
定　　价：69.80 元

产品编号：046158-01

如何使用 Java 开发资源库

在学习《Java 从入门到精通（实例版）》一书时，配合随书光盘提供了“Java 开发资源库”系统，可以帮助读者快速提升编程水平和解决实际问题的能力。《Java 从入门到精通（实例版）》和 Java 开发资源库配合学习流程如图 1 所示。

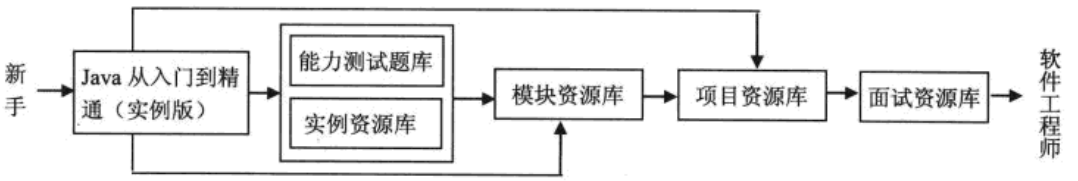


图 1　从入门到精通与开发资源库配合学习流程图

打开光盘的“Java 开发资源库”文件夹，运行 Java 开发资源库.exe 程序，即可进入“Java 开发资源库”系统，主界面如图 2 所示。

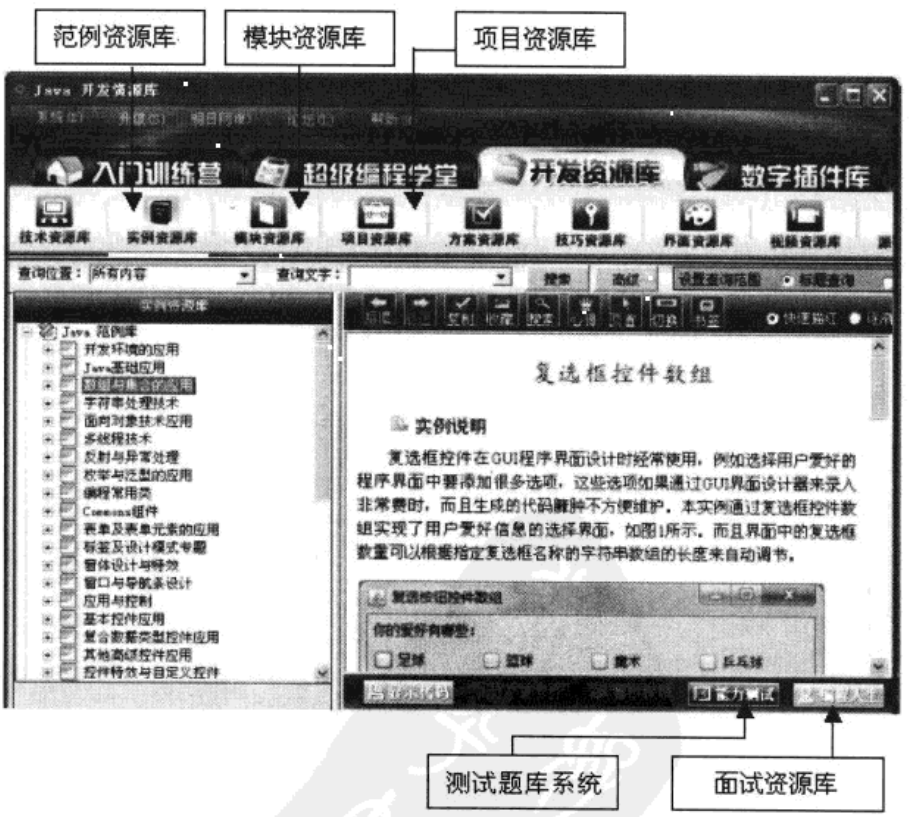


图 2　Java 开发资源库主界面

在学习《Java 从入门到精通（实例版）》某一章节时，可以配合实例资源库的相应章节，利用实例资源库提供的大量热点实例和关键实例巩固所学编程技能，提高编程兴趣和自信心。也可以配合能力测试题库的对应章节进行测试，检验学习成果，具体流程如图 3 所示。

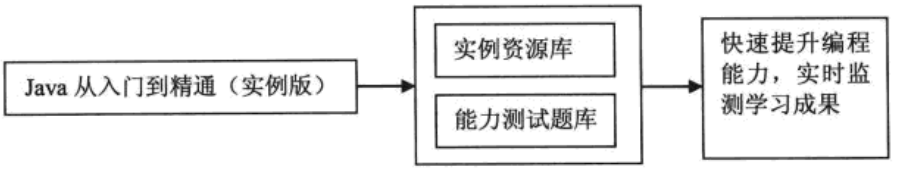


图 3　使用实例资源库和能力测试题库

对于数学逻辑能力和英语基础较为薄弱的读者，或者想了解个人数学逻辑思维能力和编程英语基础的用户，本书提供了数学及逻辑思维能力测试和编程英语能力测试供练习和测试，如图 4 所示。

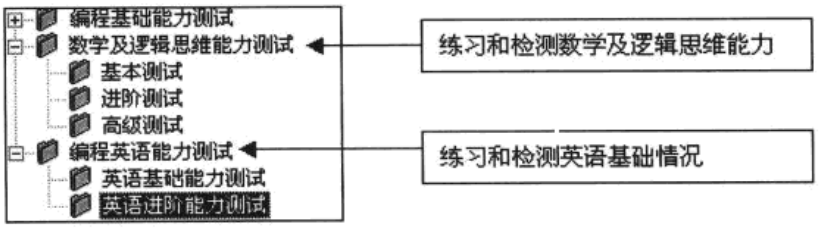


图 4　数学及逻辑思维能力测试和编程英语能力测试目录

当《Java 从入门到精通（实例版）》学习完成时，可以配合模块资源库和项目资源库的 30 个模块和项目，全面提升个人综合编程技能和解决实际开发问题的能力，为成为 Java 软件开发工程师打下坚实基础。具体模块和项目目录如图 5 所示。

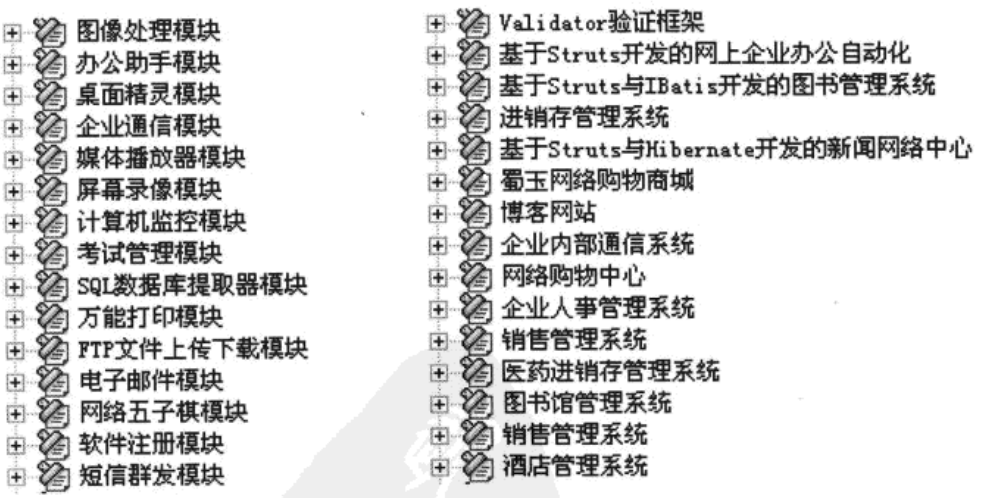


图 5　模块资源库和项目资源库目录

万事俱备，该到软件开发的主战场上接受洗礼了。面试资源库提供了大量国内外软件企业的常见面试真题，同时还提供了程序员职业规划、程序员面试技巧、企业面试真题汇编和虚拟面试系统等精彩内容，是程序员求职面试的绝佳指南。面试资源库的具体内容如图 6 所示。

图 6 面试资源库具体内容

如果您在使用本书开发资源库时遇到问题，可加我们的 QQ：4006751066（可容纳 10 万人），我们将竭诚为您服务。

前　言

Preface

丛书说明：“软件开发视频大讲堂”（第1版）于2008年8月出版以来，因其编写细腻，易学实用，配备全程视频等，在软件开发类图书市场上产生了很大反响，绝大部分品种在全国软件开发零售图书排行榜中名列前茅，2009年多个品种被评为“全国优秀畅销书”。

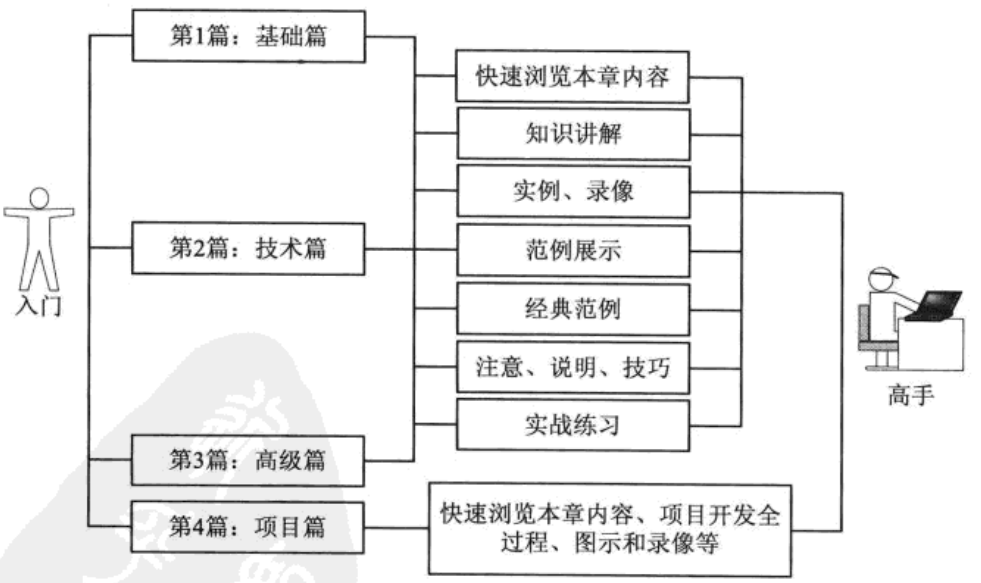
“软件开发视频大讲堂”丛书（第2版）于2010年8月出版，自出版至今，绝大部分品种在全国软件开发类零售图书排行榜中，依然持续名列前茅。丛书迄今累计已销售近40万册，被百余所高校计算机相关专业、软件学院选为教学参考书，在众多的软件开发类图书中成为一支最耀眼的品牌。

“软件开发视频大讲堂”丛书（第3版）在前两版的基础上，增删了品种，修正了疏漏，重新录制了视频，提供了从入门学习，到实例应用，到模块开发，到项目开发，到能力测试，直到面试等各个阶段的海量开发资源库。为了方便教学，还提供了教学课件PPT。

Java是Sun公司推出的能够跨越多平台的、可移植性最高的一种面向对象的编程语言，也是目前最先进、特征最丰富、功能最强大的计算机语言。利用Java可以编写桌面应用程序、Web应用程序、分布式系统应用程序、嵌入式系统应用程序等，从而使其成为应用最广泛的开发语言。

本书内容

本书提供了从入门到编程高手所必备的各类知识，共4篇，大体结构如下图所示。



第 1 篇：基础篇。本篇通过介绍初识 Java、熟悉 Eclipse 开发工具、Java 语言基础、流程控制、数组、字符串、类和对象等内容，并结合大量的图示、实例、视频等使读者快速掌握 Java 语言，为以后编程奠定坚实的基础。

第 2 篇：技术篇。本篇介绍了接口、继承与多态，类的高级特性，Java 集合类，异常处理，输入/输出，Swing 程序设计等内容。学习完本篇内容，能够开发一些小型应用程序。

第 3 篇：高级篇。本篇介绍高级事件处理、多线程、网络通信、JDBC 操作数据库、Swing 高级组件、高级布局管理器、AWT 绘制技术等内容。学习完本篇内容，能够开发高级的桌面应用程序、多媒体程序和打印程序等。

第 4 篇：项目篇。本篇通过开发一个大型、完整的酒店管理系统，运用软件工程的设计思想，让读者学习如何进行软件项目的实战开发。书中按照编写项目计划书→系统设计→数据库设计→创建项目→实现项目→运行项目→项目打包部署→解决开发常见问题的流程进行介绍，带领读者一步步亲身体验开发项目的全过程。

本书特点

- **由浅入深，循序渐进。**本书以初中级程序员为对象，先从 Java 语言基础学起，再学习 Java 的核心技术，然后学习 Swing 的高级应用，最后学习开发一个完整项目。讲解过程中步骤详尽、版式新颖，在操作的内容图片上以❶❷❸……的编号+内容的方式进行标注，让读者在阅读时一目了然，从而快速掌握书中内容。
- **语音视频，讲解详尽。**书中每一章节均提供声图并茂的语音视频教学录像，读者可以根据书中提供的录像位置在光盘中找到。这些录像能够引导初学者快速入门，感受编程的快乐和成就感，增强进一步学习的信心，从而快速成为编程高手。
- **实例典型，轻松易学。**通过例子学习是最好的学习方式，本书通过一个知识点、一个例子、一个结果、一段评析、一个综合应用的模式，透彻详尽地讲述了实际开发中所需的各类知识。另外，为了便于读者阅读程序代码，快速学习编程技能，书中几乎每行代码都提供了注释。
- **精彩栏目，贴心提醒。**本书根据需要在各章安排了很多“注意”、“说明”和“技巧”等小栏目，让读者可以在学习过程中更轻松地理解相关知识点及概念，更快地掌握个别技术的应用技巧。
- **应用实践，随时练习。**书中几乎每章都提供了“实战练习”，以让读者能够通过对问题的解答重新回顾、熟悉所学知识，举一反三，为进一步学习做好充分的准备。

读者对象

- ☑ 初学编程的自学者
- ☑ 大中专院校的老师和学生
- ☑ 毕业设计的学生
- ☑ 程序测试及维护人员
- ☑ 编程爱好者
- ☑ 相关培训机构的老师和学员
- ☑ 初中级程序开发人员
- ☑ 参加实习的“菜鸟”程序员

读者服务

为了方便解决本书疑难问题，读者朋友可加我们的 QQ：4006751066（可容纳 10 万人），也可以登录 www.mingribook.com 留言，我们将竭诚为您服务。

致读者

本书由 Java 程序开发团队组织编写，主要编写人员有陈丹丹、王国辉、李伟、李银龙、刘欣、刘玲玲、顾彦玲、杨丽、寇长梅、王小科、曹飞飞、朱晓、李慧、潘凯华、李继业、赵会东、赵永发、刘淇、王双、赵旭阳、陈媛、陈英、刘莉莉、管媛辉、高春艳、张金辉、邹淑芳、高悦、高茹、王敬洁、李贺、李浩然、郭锐、郭铁、郝洪斌、张世辉、李严、苗春义、刘清怀、张领等。在编写过程中，我们以科学、严谨的态度，力求精益求精，但错误、疏漏之处在所难免，敬请广大读者批评指正。

感谢您购买本书，希望本书能成为您编程路上的领航者。

“零门槛”编程，一切皆有可能。祝读书快乐！

编　者

目 录

Contents

第1篇 基 础 篇

第1章 初识 Java 3

视频讲解：20 分钟

1.1 Java 语言概述 4

1.1.1 Java 语言的历史 4

1.1.2 Java 语言的现状 4

1.1.3 Java 语言的面向对象特性 4

1.2 JDK 的下载和安装 6

1.2.1 JDK 下载 6

1.2.2 JDK 安装 7

1.2.3 配置 JDK 9

1.2.4 测试开发环境 11

1.3 第一个 Java 程序 11

1.4 本章小结 14

第2章 熟悉 Eclipse 开发工具 15

视频讲解：22 分钟

2.1 熟悉 Eclipse 16

2.1.1 Eclipse 简介 16

2.1.2 Eclipse 的下载与安装 16

2.1.3 Eclipse 汉化 17

2.1.4 Eclipse 工作台 21

2.1.5 透视图与视图 22

2.1.6 菜单栏 23

2.1.7 工具栏 27

2.1.8 “包资源管理器”视图 28

2.1.9 “控制台”视图 28

2.2 使用 Eclipse 28

2.2.1 创建 Java 项目 29

2.2.2 创建 Java 类文件 29

2.2.3 使用编辑器编写程序代码 30

2.2.4 运行 Java 程序 32

2.3 程序调试 32

2.4 本章小结 33

第3章 Java 语言基础 35

视频讲解：58 分钟

3.1 Java 程序的基本结构 36

3.2 标识符和关键字 37

3.2.1 标识符 37

3.2.2 关键字 38

3.3 基本数据类型 38

3.3.1 整数类型 39

3.3.2 浮点类型 40

3.3.3 字符类型 42

3.3.4 布尔类型 43

3.4 变量与常量 43

3.4.1 声明变量 44

3.4.2 声明常量 45

3.4.3 变量的有效范围 45

3.5 运算符 47

3.5.1 赋值运算符 47

3.5.2 算术运算符 48

3.5.3 自增和自减运算符 49

3.5.4 比较运算符 49

3.5.5 逻辑运算符 50

3.5.6 位运算符 51

3.5.7 三元运算符 53

3.5.8 运算符优先级 54

3.5.9 范例 1：不用其他变量实现两变量互换 55

3.5.10 范例 2：判断数字的奇偶性 56

3.6 类型转换 56
3.6.1 自动类型转换 56
3.6.2 强制类型转换 57
3.6.3 范例 3：类型转换实战 58
3.7 代码注释和编码规范 59
3.7.1 代码注释 59
3.7.2 编码规范 60
3.8 经典范例 61
3.8.1 经典范例 1：判断某一年是否是闰年 61
3.8.2 经典范例 2：求球形的体积 61
3.9 本章小结 62
3.10 实战练习 62

第 4 章 流程控制 63
视频讲解：31 分钟
4.1 复合语句 64
4.2 条件语句 65
4.2.1 if 条件语句 65
4.2.2 switch 多分支语句 69
4.2.3 范例 1：验证登录信息的合法性 71
4.2.4 范例 2：为新员工分配部门 71
4.3 循环语句 73
4.3.1 while 循环语句 73
4.3.2 do…while 循环语句 74
4.3.3 for 循环语句 75
4.3.4 范例 3：使用 while 循环遍历数组 77
4.3.5 范例 4：使用 for 循环输出九九乘法表 77
4.4 跳转语句 78
4.4.1 break 语句 78
4.4.2 continue 语句 79
4.4.3 return 语句 80
4.4.4 范例 5：终止循环体 80
4.4.5 范例 6：循环体的过滤器 81
4.5 经典范例 82
4.5.1 经典范例 1：使用 for 循环输出空心的菱形 82
4.5.2 经典范例 2：使用 for 循环输出杨辉三角 84
4.6 本章小结 84
4.7 实战练习 85

第 5 章 数组 87
视频讲解：31 分钟
5.1 数组概述 88
5.2 一维数组 88
5.2.1 创建一维数组 89
5.2.2 初始化一维数组 90
5.2.3 范例 1：求一维数组各元素的和 90
5.2.4 范例 2：获取一维数组的最小值 91
5.3 多维数组 92
5.3.1 二维数组 92
5.3.2 三维数组 94
5.3.3 范例 3：对矩阵进行转置运算 95
5.3.4 范例 4：求方阵的迹 96
5.4 数组的基本操作 96
5.4.1 遍历数组 97
5.4.2 填充替换数组元素 98
5.4.3 对数组进行排序 99
5.4.4 复制数组 100
5.4.5 范例 5：对比一维、二维数组所占内存 102
5.4.6 范例 6：使用直接插入排序法排序 103
5.5 经典范例 104
5.5.1 经典范例 1：使用冒泡排序法排序 104
5.5.2 经典范例 2：输出九宫格 105
5.6 本章小结 105
5.7 实战练习 106

第 6 章 字符串 107
视频讲解：48 分钟
6.1 创建字符串 108
6.2 字符串操作 108
6.2.1 字符串连接 108
6.2.2 获取字符串信息 110
6.2.3 去除字符串中的空格 112
6.2.4 字符串替换 114
6.2.5 判断字符串 115
6.2.6 字母大小写转换 117
6.2.7 字符串分割 118
6.2.8 范例 1：根据指定分割符把字符串分行 119
6.2.9 范例 2：判断字符串是否是数字格式 120

6.3 格式化字符串 121
6.3.1 格式化方法 121
6.3.2 日期格式化 122
6.3.3 时间格式化 123
6.3.4 日期时间组合格式化 124
6.3.5 常规类型格式化 125
6.3.6 范例 3：将金额格式化成大写 126
6.3.7 范例 4：将数字格式化成货币格式 128
6.4 正则表达式 128
6.4.1 判断是否符合正则表达式的方法 128
6.4.2 正则表达式的元字符 129
6.4.3 正则表达式的限定符 130
6.4.4 方括号中元字符的含义 131
6.4.5 范例 5：验证 IP 地址的合法性 132
6.4.6 范例 6：计算文章中汉字的个数 133
6.5 字符串生成器 134
6.5.1 StringBuilder 类 134
6.5.2 StringBuilder 类的常用方法 135
6.5.3 范例 7：将汉字和字母转换成 Unicode 码 138
6.5.4 范例 8：去掉字符串中重复的字符 138
6.6 经典范例 139
6.6.1 经典范例 1：实现字符串加密 139
6.6.2 经典范例 2：验证字符串是否回文 140
6.7 本章小结 141
6.8 实战练习 141

第 7 章 类和对象 143
视频讲解：46 分钟
7.1 面向对象概述 144
7.1.1 什么是对象 144
7.1.2 什么是类 145
7.1.3 面向对象的特点 146
7.2 类 148
7.2.1 类的构造方法 148
7.2.2 类的主方法 149
7.2.3 成员变量 150
7.2.4 成员方法 151
7.2.5 局部变量 152
7.2.6 局部变量的有效范围 152
7.2.7 静态变量、常量和方法 153
7.2.8 权限修饰符 155
7.2.9 this 关键字 156
7.2.10 范例 1：自定义图书类 157
7.2.11 范例 2：温度单位转换工具 158
7.3 对象 158
7.3.1 对象的创建 159
7.3.2 访问对象的属性和行为 160
7.3.3 对象的引用 162
7.3.4 对象的比较 162
7.3.5 对象的销毁 163
7.3.6 范例 3：统计图书销量 164
7.3.7 范例 4：重新计算对象的哈希码 165
7.4 经典范例 166
7.4.1 经典范例 1：汉诺塔问题求解 166
7.4.2 经典范例 2：单例模式的应用 167
7.5 本章小结 168
7.6 实战练习 168

第 2 篇　技　术　篇

第 8 章 接口、继承与多态 171
视频讲解：18 分钟
8.1 接口的使用 172
8.1.1 接口的定义 172
8.1.2 接口的实现 172
8.1.3 范例 1：图片的不同格式保存 173
8.1.4 范例 2：为汽车增加 GPS 定位功能 174
8.2 类的继承 175
8.2.1 继承的实现 175
8.2.2 继承中的重写 176
8.2.3 使用 super 关键字 177
8.2.4 范例 3：经理与员工的差异 179

8.2.5 范例 4：重写父类中的方法.....180
8.3 多态 181
8.3.1 什么是多态.....181
8.3.2 范例 5：计算几何图形的面积.....183
8.3.3 范例 6：简单的汽车销售商场.....184
8.4 经典范例 185
8.4.1 经典范例 1：使用 Comparable 接口自定义排序.....185
8.4.2 经典范例 2：动态设置类的私有域.....187
8.5 本章小结 188
8.6 实战练习 188

第 9 章 类的高级特性 189
视频讲解：34 分钟
9.1 抽象类 190
9.2 内部类 192
9.2.1 成员内部类.....192
9.2.2 局部内部类.....195
9.2.3 匿名内部类.....195
9.2.4 静态内部类.....197
9.2.5 内部类的继承.....198
9.2.6 范例 1：局部内部类设置闹钟.....198
9.2.7 范例 2：静态内部类求极值.....199
9.3 Class 类与 Java 反射..... 200
9.3.1 访问构造方法.....201
9.3.2 访问成员变量.....204
9.3.3 访问方法.....206
9.3.4 范例 3：运用反射查看类的成员.....209
9.3.5 范例 4：动态调用类中方法.....210
9.4 经典范例 211
9.4.1 经典范例 1：利用反射重写 toString()方法.....211
9.4.2 经典范例 2：普通内部类的简单应用.....212
9.5 本章小结 213
9.6 实战练习 213

第 10 章 Java 集合类..... 215
视频讲解：45 分钟
10.1 集合类概述 216
10.2 集合类接口的常用方法 216
10.2.1 List 接口的常用方法.....216
10.2.2 Set 接口的常用方法.....218
10.2.3 Map 接口的常用方法.....219
10.2.4 范例 1：用 List 集合传递学生信息.....220
10.2.5 范例 2：Map 集合二级联动.....222
10.3 集合类接口的实现类 223
10.3.1 List 接口的实现类.....223
10.3.2 Set 接口的实现类.....224
10.3.3 Map 接口的实现类.....226
10.3.4 范例 3：for 循环遍历 ArrayList.....227
10.3.5 范例 4：用动态数组保存学生姓名.....228
10.4 迭代器 229
10.4.1 迭代器的创建和使用.....229
10.4.2 范例 5：Iterator 遍历 ArrayList.....230
10.4.3 范例 6：ListIterator 遍历 ArrayList.....231
10.5 经典范例 232
10.5.1 经典范例 1：制作电子词典.....232
10.5.2 经典范例 2：制作手机电话本.....234
10.6 本章小结 235
10.7 实战练习 235

第 11 章 异常处理..... 237
视频讲解：49 分钟
11.1 异常概述 238
11.2 异常的分类 238
11.2.1 可控式异常.....238
11.2.2 运行时异常.....240
11.2.3 范例 1：算术异常.....241
11.2.4 范例 2：数组下标越界异常.....242
11.3 获取异常信息 243
11.4 处理异常 244
11.4.1 使用 try…catch 处理异常.....244
11.4.2 使用 try…catch…finally 处理异常.....245
11.4.3 使用 try…finally 处理异常.....246
11.5 抛出异常 248
11.5.1 使用 throws 声明抛出异常.....248
11.5.2 使用 throw 语句抛出异常.....250
11.5.3 范例 3：方法中抛出异常.....251
11.5.4 范例 4：方法上抛出异常.....251
11.6 自定义异常 252
11.6.1 创建自定义异常类.....252

11.6.2 使用自定义异常类 253
11.7 异常的使用原则 254
11.8 经典范例 254
11.8.1 经典范例 1：捕获单个异常 254
11.8.2 经典范例 2：数据库操作异常 255
11.9 本章小结 256
11.10 实战练习 256

第 12 章 输入/输出 257
视频讲解：30 分钟
12.1 流概述 258
12.2 输入/输出流 258
12.2.1 输入流 258
12.2.2 输出流 260
12.2.3 范例 1：显示指定类型的文件 261
12.2.4 范例 2：查找替换文本文件内容 262
12.3 File 类 263
12.3.1 文件的创建与删除 263
12.3.2 获取文件信息 264
12.3.3 范例 3：文件批量重命名 265
12.3.4 范例 4：快速批量移动文件 266
12.4 文件输入/输出流 267
12.4.1 FileInputStream 与 FileOutputStream 类 267
12.4.2 FileReader 类和 FileWriter 类 268
12.4.3 范例 5：窗体动态加载磁盘文件 270
12.4.4 范例 6：删除文件夹中所有文件 272
12.5 带缓存的输入/输出流 272
12.5.1 BufferedInputStream 类与 BufferedOutputStream 类 272
12.5.2 BufferedReader 类与 BufferedWriter 类 273
12.5.3 范例 7：读取属性文件单个属性值 275
12.5.4 范例 8：合并多个 TXT 文件 276
12.6 数据输入/输出流 277
12.7 ZIP 压缩输入/输出流 278
12.7.1 压缩文件 278
12.7.2 解压缩 ZIP 文件 280
12.7.3 范例 9：压缩包解压到指定文件夹 281
12.7.4 范例 10：压缩所有子文件夹 283
12.8 经典范例 284
12.8.1 经典范例 1：简单的投票软件 284
12.8.2 经典范例 2：电子通讯录 284
12.9 本章小结 285
12.10 实战练习 286

第 13 章 Swing 程序设计 287
视频讲解：51 分钟
13.1 Swing 概述 288
13.2 Swing 常用窗体 288
13.2.1 JFrame 框架窗体 288
13.2.2 JDialog 窗体 290
13.2.3 范例 1：设置窗体大小 291
13.2.4 范例 2：禁止改变窗体的大小 292
13.3 标签组件与图标 292
13.3.1 标签的使用 292
13.3.2 图标的使用 293
13.3.3 范例 3：为图片添加说明 295
13.4 常用布局管理器 296
13.4.1 绝对布局 297
13.4.2 流布局管理器 297
13.4.3 边界布局管理器 299
13.4.4 网格布局管理器 300
13.5 常用面板 301
13.5.1 JPanel 面板 301
13.5.2 JScrollPane 面板 302
13.6 按钮组件 303
13.6.1 提交按钮组件 303
13.6.2 单选按钮组件 304
13.6.3 复选框组件 305
13.7 列表组件 306
13.7.1 下拉列表框组件 306
13.7.2 列表框组件 308
13.8 文本组件 310
13.8.1 文本框组件 310
13.8.2 密码框组件 311
13.8.3 文本域组件 311
13.8.4 范例 4：给文本域设置背景图片 312
13.8.5 范例 5：给文本区设置背景图片 313
13.9 经典范例 313

13.9.1 经典范例 1：设置窗体标题栏图标.....313
13.9.2 经典范例 2：随机更换窗体背景.....314
13.10 本章小结 315
13.11 实战练习 315

第 3 篇 高 级 篇

第 14 章 高级事件处理 319
视频讲解：30 分钟
14.1 键盘事件 320
14.2 鼠标事件 322
14.3 窗体事件 324
14.3.1 捕获窗体焦点变化事件.....324
14.3.2 捕获窗体状态变化事件.....325
14.3.3 捕获其他窗体事件.....326
14.4 选项事件 328
14.5 表格模型事件 329
14.6 经典范例 332
14.6.1 经典范例 1：模拟相机拍摄.....332
14.6.2 经典范例 2：打地鼠游戏.....333
14.7 本章小结 335
14.8 实战练习 335

第 15 章 多线程 337
视频讲解：41 分钟
15.1 线程简介 338
15.2 实现线程的两种方式 338
15.2.1 继承 Thread 类 339
15.2.2 实现 Runnable 接口 340
15.2.3 范例 1：查看线程的运行状态.....342
15.2.4 范例 2：查看 JVM 中的线程名.....344
15.3 线程的生命周期 345
15.4 操作线程的方法 346
15.4.1 线程的休眠.....347
15.4.2 线程的加入.....348
15.4.3 线程的中断.....350
15.4.4 线程的礼让.....351
15.4.5 范例 3：查看和修改线程优先级.....351
15.4.6 范例 4：休眠当前线程.....353
15.5 线程的优先级 354
15.6 线程同步 356
15.6.1 线程安全.....356
15.6.2 线程同步机制.....357
15.7 线程间的通信 359
15.8 经典范例 361
15.8.1 经典范例 1：查看 JVM 中的线程名.....361
15.8.2 经典范例 2：查看和修改线程名称.....362
15.9 本章小结 364
15.10 实战练习 364

第 16 章 网络通信 365
视频讲解：33 分钟
16.1 网络程序设计基础 366
16.1.1 局域网与因特网.....366
16.1.2 网络协议.....366
16.1.3 端口和套接字.....367
16.1.4 范例 1：获得内网的所有 IP 地址.....368
16.1.5 范例 2：解析网页中的内容.....369
16.2 TCP 程序设计基础 370
16.2.1 InetAddress 类.....371
16.2.2 ServerSocket 类.....372
16.2.3 TCP 网络程序.....373
16.2.4 范例 3：设置等待连接的超时时间.....376
16.2.5 范例 4：获得 Socket 信息.....377
16.3 UDP 程序设计基础 378
16.3.1 DatagramPacket 类.....379
16.3.2 DatagramSocket 类.....379
16.3.3 UDP 网络程序.....379
16.4 经典范例 383
16.4.1 经典范例 1：聊天室服务器端.....383
16.4.2 经典范例 2：聊天室客户端.....385
16.5 本章小结 387
16.6 实战练习 387

第 17 章　JDBC 操作数据库 389
视频讲解：1 小时 25 分钟
17.1　JDBC 概述 390
17.1.1　数据库概述 390
17.1.2　JDBC 技术 391
17.2　JDBC 的常用类和接口 391
17.2.1　DriverManager 类 391
17.2.2　Connection 接口 392
17.2.3　Statement 接口 393
17.2.4　PreparedStatement 接口 393
17.2.5　ResultSet 接口 393
17.3　通过 JDBC 操作数据库 394
17.3.1　加载数据库驱动 394
17.3.2　建立连接 395
17.3.3　向数据库添加数据 396
17.3.4　获得查询结果集 397
17.3.5　更改数据库中的数据 398
17.3.6　删除数据库中的数据 399
17.3.7　模糊查询 400
17.3.8　范例 1：将复选框内容存到数据库 401
17.3.9　范例 2：查询三年二班的学生信息 403
17.4　经典范例 404
17.4.1　经典范例 1：管理学生信息 404
17.4.2　经典范例 2：权限管理 410
17.5　本章小结 415
17.6　实战练习 415

第 18 章　Swing 高级组件 417
视频讲解：48 分钟
18.1　利用 JTable 类直接创建表格 418
18.1.1　创建表格 418
18.1.2　定制表格 420
18.1.3　操纵表格 423
18.1.4　范例 1：列表元素与提示信息 425
18.1.5　范例 2：监听列表单击事件 426
18.2　表格模型与表格 427
18.2.1　利用表格模型创建表格 427
18.2.2　维护表格模型 429
18.2.3　范例 3：实现自动排序列表 431
18.2.4　范例 4：可以预览字体的列表 432
18.3　提供行标题栏的表格 433
18.3.1　提供行标题栏的表格 433
18.3.2　范例 5：提供行标题栏的表格 433
18.4　Swing 树组件 437
18.4.1　简单的树 437
18.4.2　处理选中节点事件 438
18.4.3　遍历树节点 440
18.4.4　定制树 442
18.4.5　维护树模型 444
18.4.6　处理展开节点事件 446
18.4.7　范例 6：为树节点增加提示信息 448
18.4.8　范例 7：双击编辑树节点功能 449
18.5　经典范例 450
18.5.1　经典范例 1：编写中国省市信息树 450
18.5.2　经典范例 2：监听节点的选择事件 451
18.6　本章小结 452
18.7　实战练习 452

第 19 章　高级布局管理器 453
视频讲解：39 分钟
19.1　箱式布局管理器 454
19.2　卡片布局管理器 456
19.3　网格组布局管理器 458
19.4　弹簧布局管理器 464
19.4.1　使用弹簧布局管理器 464
19.4.2　使用弹簧和支柱 467
19.4.3　利用弹簧控制组件大小 468
19.5　经典范例 469
19.5.1　经典范例 1：制作圆形布局管理器 469
19.5.2　经典范例 2：制作阶梯布局管理器 470
19.6　本章小结 471
19.7　实战练习 471

第 20 章　AWT 绘图技术 473
视频讲解：33 分钟
20.1　绘制图形 474
20.1.1　Graphics 474
20.1.2　Graphics2D 476
20.1.3　范例 1：绘制指定角度的填充扇形 478
20.1.4　范例 2：绘制多边形 478
20.2　绘图颜色与笔画属性 479

光盘“开发资源库”目录

第 1 大部分　实例资源库

（732 个完整实例分析，光盘路径：开发资源库/实例资源库）

开发环境的应用

- 下载 JDK 开发工具包
- 把 JDK 工具包安装到指定磁盘
- 设置 JDK 的环境变量
- 验证 Java 开发环境
- 下载并安装 JRE 执行环境
- 编程输出星号组成的等腰三角形
- 下载最新的 Eclispe
- 为最新的 Eclipse 安装中文语言包
- 活用 Eclipse 的工作空间
- 在 Eclispe 项目中编程输出字符表情
- 为 Eclipse 添加新的 JDK 环境
- 设置 Eclipse 中文 API 提示信息
- 为项目添加类库
- 使当前项目依赖另一个项目
- 安装界面设计器
- 设计 Windows 系统的运行对话框界面
- 设计计算器程序界面
- 设计进销存管理系统的关于界面
- JDK 1.5 的安装与配置
- Tomcat 5.5 的安装
- 配置 Windows 2000+SQL Server 2000+Tomcat 运行
- 配置 Windows 2000+Oracle +Tomcat 运行环境
- 配置 Windows 2000+ Access +Tomcat 运行环境
- 配置 Windows 2000+MySQL+Tomcat 运行环境
- 配置 Windows XP 2003+SQL Server 2000+ Tomcat 运行环境
- 在 Linux 下安装 JDK 1.5
- 在 Linux 下配置 Tomcat 服务器
- 配置 Linux+MySQL+ Tomcat 运行环境
- 配置 Windows+Resin 运行环境
- 配置 Linux+Resin 运行
- 安装与配置 WebLogic 服务器
- WebLogic 中 SQL Server 2000 的 JDBC 连接池配置
- 应用 Dreamweaver 开发 JSP 程序
- 配置 Windows 2000+MySQL+Tomcat 运行环境
- 应用 MyEclipse 开发 JSP 程序
- 应用 NetBeans 开发 JSP 程序

Java 基础应用

- 基本语法
- 运算符
- 条件语句
- 循环控制

数组与集合的应用

- 数组演练
- 数组操作
- 数组排序与查询
- 常用集合的使用

字符串处理技术

- 格式化字符串
- 辨别字符串
- 操作字符串

面向对象技术应用

- Java 中类的定义
- 修饰符的使用
- 包装类的使用
- 面向对象的特征
- Object 类的应用

- 克隆与序列化
- 接口和内部类

多线程技术

- 线程的基础
- 线程的同步
- 线程的进阶

反射与异常处理

- 反射的基础
- 反射的进阶
- 常见的未检查型异常
- 常见的已检查型异常
- 处理异常

枚举与泛型的应用

- 枚举使用的简介
- 泛型使用的简介

编程常用类

- Calendar 类的使用
- SimpleDateFormat 与 TimeZone 类的使用
- System 类的使用
- Math 类的使用
- 其他常用类的使用

Commons 组件

- Commons Lang 组件简介
- Commons Math 组件简介
- Commons IO 组件简介
- Commons BeanUtils 组件简介
- 其他 Commons 组件简介

表单及表单元素的应用

- 获取文本框编辑框隐藏域的值
- 获取下拉列表菜单的值
- 获取复选框的值
- 获取单选按钮的值
- 把数据库中的记录显示到下拉列表中
- 将数组中的数据添加到下拉列表中
- 级联菜单
- 修改数据时下拉列表的默认值为数据库中原数据信息
- 可以输入文字的下拉列表
- 根据下拉列表的值显示不同控件
- 根据数据表结构自动生成数据录入页面
- 投票信息一次性设置
- 自动计算金额
- 设置文本框的只读属性
- 让您的密码域更安全
- 限制多行文本域输入的字符个数
- 不提交表单获取单选按钮的值
- 选中单选按钮后显示其他表单元素
- 防止表单在网站外部提交
- 同一个页中的多表单提交

标签及设计模式专题

- 利用〈c：forEach〉循环标签实现数据显示
- 利用 EL 表达式语言实现页面逻辑处理简单化
- 自定义文件下载标签
- 自定义图片浏览标签
- 自定义数据查询标签
- 应用 HQL 检索方式查询数据
- 应用 QBC 检索方式查询数据
- 应用一对一关联实现级联添加数据
- 应用一对多关联实现级联操作
- 使用本地 SQL 检索
- DispathAction 类实现用户查询
- LookupDispatchAction 类实现用户管理
- 利用 Token 令牌机制处理用户重复提交
- 利用 Validator 验证框架处理用户登录
- 解决用户提交的中文乱码
- 利用动态 FormBean 实现对用户的操作
- Struts 与 Hibernate 结合实现数据添加和查询
- 在 Spring 中的表单控制器中实现验证处理
- 利用表单控制器实现数据添加操作
- 利用 Spring 中的多方法控制器实现数据查询和删除操作
- 通过 Spring+Hibernate 框架实现大批量数据添加

窗体设计与特效

- 设置窗体位置
- 设置窗体大小
- 设置窗体的标题栏
- 设置窗体的背景
- 窗体形状及应用
- 对话框
- MDI 窗体的使用

让窗体更有活力
窗体与控件外观

窗口与导航条设计

打开新窗口显示广告信息
自动关闭的广告窗口
弹出窗口居中显示
打开新窗口显示详细信息
弹出窗口的 Cookie 控制
为弹出的窗口加入关闭按钮
关闭弹出窗口时刷新父窗口
关闭 IE 主窗口时，不弹出询问对话框
弹出网页模式对话框
弹出全屏显示的网页（模式）对话框
网页拾色器
日期选择器
全屏显示无边框有滚动条窗口
应用 JavaScript 实现指定尺寸的无边框窗口
应用 CSS+DIV 实现无边框窗口
带图标的文字导航条
Flash 导航条
图片按钮导航条
导航条的动画效果
不用图片实现质感导航条
二级导航菜单
半透明背景的下拉菜单
弹出式下拉菜单
展开式导航条
收缩式导航菜单
树状导航菜单

应用与控制

将表单数据输出到 Word
将查询结果输出到 Word
通过 ODBC 访问 Excel
利用 Java Excel 访问 Excel
在 JSP 页面通过按钮打开新的 Excel 文件
将查询结果导出到 Excel
导出到 Access 数据库中
导出到 Excel 数据库中
在 JSP 中压缩 ZIP 文件
在 JSP 中解压缩 ZIP 文件
利用 Spring 生成 Excel 工作表
利用 Spring 生成 PDF 文件

基本控件应用

顶层容器的应用
布局管理器应用
输入控件的应用
选择控件的应用
菜单控件的应用
其他技术的应用

复合数据类型控件应用

列表的简单应用
列表的高级应用
表格的简单应用
表格的高级应用
树控件简单应用
树控件高级应用

其他高级控件应用

JTextPane 控件的应用
JEditorPane 控件的应用
其他文本控件的应用
进度指示器的应用
控件组织器的应用

控件特效与自定义控件

控件边框效果
控件渲染让界面 UI 更灵活
让控件活起来
自定义控件

操作办公文档

操作 Word
操作 Excel
操作 PDF

文件、文件夹与系统

单表单元素上传文件到数据库
多表单元素上传文件到数据库
上传文件到服务器
限制文件大小的文件上传
遍历指定目录下的所有文件
获取驱动器信息
遍历指定驱动器
访问类路径上的资源文件

获取文件信息
查看文件是否存在
重命名文件
对文件夹创建、删除的操作
使用 Java 的输入输出流从文本文件中读取注册服务条款
使用 Java 的输入输出流实现永久计数器
通过文本文件向数据库传递数据
读取属性文件
复制文件夹
文件下载
使用 JSP 生成 XML 文档
使用 DOM 读取 XML 文件
使用 SAX 读取 XML 文件
修改文件属性
显示指定类型的文件
以树结构显示文件路径
查找替换文本文件内容
支持图片预览的文件选择对话框
设置 Windows 的文件属性
文件批量重命名
快速批量移动文件
删除磁盘所有.tmp 临时文件
提取数据库内容到文件
提取文本文件的内容到 MySQL 数据库
将图片文件保存到 SQL Server 数据库
显示数据库中的图片信息
提取技术网站数据到文件夹
读取文件路径到数据库
在数据库中建立磁盘文件索引
窗体动态加载磁盘文件
删除文件夹中所有文件
创建磁盘索引文件
快速全盘查找文件
获取磁盘所有文本文件
网络文件夹备份
键盘录入内容保存到文本文件
将数组写入到文件中并逆序输出
利用 StringBuffer 避免文件的多次写入
合并多个 txt 文件
实现文件简单加密与解密
对大文件实现分割处理
将分割后的文件重新合并
读取属性文件的单个属性值
向属性文件中添加信息
在复制文件时使用进度条
从 XML 文件中读取数据
读取 Jar 文件属性
电子通讯录
批量复制指定扩展名的文件
计数器小程序
将某文件夹中的文件进行分类存储
利用 StreamTokenizer 统计文件的字符数
在指定目录下搜索文件
序列化与反序列化对象
文件锁定
投票统计
压缩所有文本文件
压缩包解压到指定文件夹
压缩所有子文件夹
深层文件夹压缩包的释放
解决压缩包中文乱码
Apache 实现文件解压缩
把窗体压缩成 ZIP 文件
解压缩 Java 对象
文件压缩为 RAR 文档
解压缩 RAR 压缩包
文件分卷压缩
为 RAR 压缩包添加注释
获取压缩包详细文件列表
从 RAR 压缩包中删除文件
在压缩文件中查找字符串
重命名 RAR 压缩包中的文件
创建自解压 RAR 压缩包
设置 RAR 压缩包密码
以压缩格式传输网络数据
压缩远程文件夹
压缩存储网页
JSP 批量文件上传
Struts 的文件上传

Spring 的文件上传
从 FTP 下载文件
文件列表维护
文件在线压缩与解压缩
判断远程文件是否存在
通过文本文件实现网站计数器
JSP 生成 XML 文件

数据库的操作

通过 JDBC-ODBC 桥连接 SQL Server 数据库
通过 JDBC 连接 SQL Server 数据库
通过 Tomcat 连接池连接 SQL Server 数据库
通过 WebLogic 连接池连接 SQL Server 数据库
应用 Hibernate 连接 SQL Server 数据库
通过 JDBC-ODBC 桥连接 Access 数据库
应用 Hibernate 连接 Access 数据库
通过 JDBC 连接 MySQL 数据库
通过 Tomcat 连接池连接 MySQL 数据库
应用 Hibernate 连接 MySQL 数据库
通过 JDBC 连接 Oracle 数据库
应用 Hibernate 连接
利用 SQL 语句实现分页
利用结果集进行分页
转到指定页的分页
具有页码跳转功能的分页
分栏显示
分类、分栏显示
对超长文本数据进行分页显示
单条数据录入
批量数据插入
插入用户登录日志信息
更新指定记录
批量更新
商品价格调整
修改密码
找回密码
动态创建 SQL Server 数据库
动态创建 SQL Server 数据表和字段
动态创建 MySQL 数据库
列举 SQL Server 数据库中的数据表
列举 MySQL 数据库中的数据表
查看数据表结构
在线维护投票数据库
通过 JDBC 获取插入记录的自动编号
通过 Hibernate 获取插入记录的自动编号
在线删除指定的一个数据表
在线删除多个指定的数据表
在线删除指定数据表中的指定索引
清空指定数据表中的所有数据
快速清空指定数据表中的所有记录
批量清空数据表中的数据
生成 SQL 数据库脚本
恢复 SQL 数据库脚本
删除指定记录
批量删除数据
删除数据前给予提示
获取从数据库里删除的记录数
生成有规律的编号
生成无规律的编号
SQL Server 数据备份
SQL Server 数据恢复
动态附加数据库
应用 JDBC 事务
Hibernate 中应用事务
对数据进行降序查询
对数据进行多条件排序查询
对统计结果进行排序
查询 SQLServer 数据表中前 3 条数据
查询 SQLServer 数据库后 3 条数据
查询 MySQL 数据库中的前 3 条数据
查询 MySQL 数据库中后 3 条数据
按照字母顺序对留学生表进行排序
按姓氏笔画排序
将汉字按音序排序
按列的编号排序
从表中随机返回记录
使用 GROUP BY 函数实现对数据的分组统计
使用 GROUP BY 函数实现多表分组统计
利用 SUM 函数实现数据汇总
利用 AVG 函数实现计算平均值
利用 MIN 函数求数据表中的最小数据

利用 MAX 函数求数据表中的最大值
利用 COUNT 函数求销售额大于某值的图书种类
查询编程词典 6 月的销售量
查询与张静同一天入司的员工信息
使用 IN 谓词查询某几个时间的数据
日期查询中避免千年虫问题
在查询结果中不显示重复记录
使用 NOT 查询不满足条件的记录
使用 between 进行区间查询
列出销量表中重复记录和记录条数
使用关系运算符查询某一时间段数据
计算两个日期之间的月份数
格式化金额
在查询语句中过滤掉字符串中的空格
通过 JDBC-ODBC 桥连接 SQL Server2000
通过 JDBC 连接 SQLServer2000 数据库
JDBC 连接 SQLServer2005 数据库
JDBC 技术连接 Oracle 数据库
JDBC 连接 JavaDB 数据库
列举 SQLServer 数据库下的数据表
列举 MySQL 数据库下的数据表
动态维护投票数据库
MySQL 数据备份
MySQL 数据恢复
获取 SQLServer 数据表字段的描述信息
将员工信息添加到数据表
添加数据时使用数据验证
在插入数据时过滤掉危险字符
将用户选择爱好以字符串形式保存到数据库
将数据从一张表复制到另张表
使用 UNION ALL 语句批量插入数据
在删除数据时给出提示信息
将数据表清空
字符串大小写转换
使用 JDBC 连接常用数据库
JDBC 实现简单的搜索引擎
使用 JDBC 操作数据库
使用 JDBC 查看数据表结构
用户密码管理
使用 JDBC 批量处理数据
在 JDBC 中使用存储过程
使用 Tomcat 连接池
使用 dbcp 连接池
使用 c3p0 连接池
MySQL 数据库的备份和恢复
MySQL 数据库分页
SQL Server 数据库备份与恢复
SQL Server 数据库附加与分离
SQL Server 数据库分页
Hibernate 实现分页
将数据表导出到 XML 文件中
从 XML 文件导入数据表
在 MySQL 数据库增加新用户权限
使用事务提高数据库操作的安全性
在数据库中添加或读取文件数据
利用触发器记录系统登录日志
查询数值型数据
查询字符串
查询日期型数据
查询逻辑型数据
查询非空数据
查询文本框中指定的字符串
查询下拉列表框中指定的数值数据
查询下拉列表框中的日期数据
将表单元素中的内容作为字段、运算符和内容进行查询
利用变量查询字符串
利用变量查询数值型数据
查询前 5 名数据
查询后 5 名数据
取出数据统计结果前 3 名数据
查询指定 SQL Server 数据库中的日期型数据
查询指定 Access 数据库中的日期型数据
查询指定时间段的数据
按月查询数据
查询大于指定条件的数据
查询时不显示重复记录
NOT 与谓词进行组合条件的查询
列出数据中的重复记录和记录条数
单列数据分组统计

- 多列数据分组统计
- 多表分组统计
- 利用聚集函数 SUM 对学生成绩进行汇总
- 利用聚集函数 AVG 求某班学生的平均成绩
- 利用聚集函数 MIN 求销售额最少的商品
- 利用聚集函数 MAX 求月销售额完成最多的员工
- 利用聚集函数 COUNT 求日销售额大于某值的图书种类数
- 利用 FROM 子句进行多表查询
- 使用表的别名
- 合并多个结果集
- 简单的嵌套查询
- 复杂的嵌套查询
- 用子查询作派生的表
- 用子查询作表达式
- 用子查询关联数据
- 多表联合查询
- 对联合查询后的结果进行排序
- 条件联合查询
- 简单内连接查询
- 复杂内连接查询
- 自连接
- LEFT OUTER JOIN 查询
- RIGHT OUTER JOIN 查询
- 使用外连接进行多表联合查询
- 利用 IN 谓词限定查询范围
- 用 IN 查询表中的记录信息
- 由 IN 引入的关联子查询
- 静态交叉表
- 动态交叉表
- 对查询结果进行格式化（四舍五入）
- 在查询中使用字符串函数
- 在查询中使用日期函数
- 利用 HAVING 语句过滤分组数据
- 复杂条件查询

数据库的高级应用

- 创建视图
- 使用视图过滤不想要的数据
- 使用视图与计算数据
- 使用视图重新格式化检索出来的数据
- 获取数据库中的全部用户视图
- 修改视图
- 删除视图
- 视图的应用
- 创建存储过程
- 调用存储过程实现用户身份验证
- 应用储存过程添加数据
- 调用加密存储过程
- 获取数据库中所有存储过程
- 修改存储过程
- 删除存储过程
- 应用存储过程实现数据分页
- 创建触发器
- 应用触发器自动插入回复记录
- 应用触发器添加日志信息
- 在删除成绩表时将学生表中数据删除
- 在程序中调用 UPDATE 触发器
- 获取数据库中的触发器名称
- 创建带有触发条件的触发器
- 使用批处理删除数据
- 使用批处理提升部门员工工资
- 将教师表中数据全部添加到选课表
- 在批处理中使用事务

实用的 JavaScript 函数

- 小写金额转换为大写金额
- 处理字符串中的空格
- 验证输入的日期格式是否正确
- 检查表单元素是否为空
- 验证 E-mail 是否正确
- 通过正则表达式验证电话号码
- 验证输入的字符串是否为汉字
- 验证身份证号码
- 客户端验证用户名和密码
- 验证网址是否合法
- 验证数量和金额
- 限制输入字符串的长度
- 显示长日期格式的系统日期
- 倒计时
- 实时显示系统时间

- 特殊日期提示

数据查询

- 使用子查询
- 嵌套查询
- 连接查询
- 函数查询

Servlet 技术

- 将 HTML 元素嵌入到 Servlet
- 在 Servlet 中实现页面转发的操作
- 在 Servlet 中获取当前页的绝对路径
- 在 Servlet 中对 Cookie 的操作
- 利用 JavaBean 由 Servlet 向 JSP 传递数据
- 在 Servlet 中使用 JDBC- ODBC 桥访问数据库
- 在 Servlet 中使用 JDBC 访问数据库
- 使用 Servlet 访问数据库连接池
- 使用过滤器验证用户身份
- 使用过滤器进行网站流量统计
- 使用过滤器对响应页面中的敏感字符进行过滤
- 使用监听器查看在线用户
- 利用监听器使服务器端免登录
- 过滤非法文字
- 编码过滤器解决中文问题
- 过滤器验证用户
- 过滤器分析流量
- 使用过滤器禁止浏览器缓存页面
- 监听在线用户
- 监听器实现免登录

JavaBean 技术

- 连接数据库的方法
- 数据查询的方法
- 带参数的数据查询
- 数据增加的方法
- 数据修改的方法
- 数据删除的方法
- 数据库分页的方法
- 对结果集进行分页的方法
- 关闭数据库的方法
- 数据库事务处理的方法
- 调用数据库存储过程的方法
- 附加数据库的方法
- 备份数据库的方法
- 还原数据库的方法
- 自动获得汉字的拼音简码
- 转换输入文本中的回车和空格
- 小写金额转换为大写金额
- 判断字符串是否以指定字符开头
- 计算字符串的实际长度
- 字符串截取
- 字符串转换成数组
- 检查字符是否有英文字母
- 小写字母转换为大写字母
- 大写字母转换为小写字母
- 把数组转换成字符串
- 将整型数据格式化为指定长度的字符串
- 把一个长数字分位显示
- 过滤输入字符串中的危险符号
- 判断是否为当前时间的方法
- 判断用户输入的是否是数字的方法
- 对输入数据中的 HTML 字符进行转换的方法
- 过滤字符串中的空格与 null 值的方法
- 对 SQL 语句中输入的空值进行处理的方法
- 将整型值转换为字符型的方法
- 判断用户输入的是否为有效 id 值的方法
- 获取年份的方法
- 获取月份的方法
- 获取日的方法
- 显示指定格式的日期的方法
- 显示指定格式的时间的方法
- 显示完整日期时间的方法
- 对字符串进行 GBK 编码
- 对字符串进行 ISO-8859-1 编码
- 随机产生指定位数的验证码
- 生成指定位数的随机字符串
- 用户登录模块
- 带验证码的用户登录模块
- 带识别状态的用户登录模块
- 输出提示页面的方法
- 输出分页导航的方法
- 版权信息生成的方法
- 生成柱形图

- 生成折线图
- 生成饼状图
- 实现进度条
- 弹出提示对话框并重定向网页
- 打开指定大小的新窗口并居中显示

AJAX 技术

- Ajax 无刷新分页
- Ajax 实现聊天室
- Ajax 验证用户名是否被注册
- Ajax 刷新 DIV 内容
- Ajax 级联选择框
- Ajax 实现带进度条的文件上传

在线统计

- 通过 Application 对象实现网站计数器
- 网站图形计数器
- 记录用户 IP 地址的计数器
- 只对新用户计数的计数器
- 统计用户在某一页停留的时间
- 统计用户在站点停留的时间
- 判断用户是否在线
- 实时统计在线人数
- 统计日访问量
- 利用柱形图统计分析网站访问量

网络通信

- 发送电子邮件
- 发送 HTML 格式邮件
- 带附件的邮件发送程序
- 邮件群发
- Spring 利用 Web Service 发送手机短信
- 利用短信猫发送手机短信
- 实现邮件发送
- 实现邮件接收
- 发送带附件的邮件
- 接收带附件的邮件
- 显示 POP3 未读邮件和已读邮件
- IP 地址转换成整数
- 获取本地天气预报

信息提取与图表分析

- 远程获取其他网页信息
- 网站访问量显示图表
- 投票结果显示图表
- 利用折线图分析多种商品的价格走势
- 利用折线图分析某种商品的价格走势
- 年销售额及利润图表分析

行业应用

- 禁止重复投票的在线投票系统
- 每个 IP 一个月只能投票一次的投票系统
- 一般用户注册
- 带检测用户名的用户注册
- 分步用户注册
- 论坛
- 购物车
- Application 形式的聊天室
- 聊天室（私聊）
- 数据库形式的聊天室（私聊）
- 简易万年历
- 带有备忘录的万年历

网站策略与安全

- 通过邮箱激活注册用户
- 越过表单限制漏洞
- 文件上传漏洞
- 防止 SQL 注入式攻击
- 获取客户端信息
- 防止网站文件盗链下载
- 禁止网页刷新
- 禁止复制和另存网页内容
- 防止页面重复提交
- 获取指定网页源代码并盗取数据
- 隐藏 JSP 网址扩展名
- 数据加密
- MD5 加密

安全技术

- 确定对方的 IP 地址
- 获取客户端 TCP / IP 端口的方法
- 替换输入字符串中的危险字符
- 禁止用户输入危险字符
- 用户安全登录
- 带验证码的用户登录模块

- 防止用户直接输入地址访问 JSP 文件
- 禁止复制网页内容
- 禁止网页被另存为
- 屏蔽 IE 主菜单
- 屏蔽键盘相关事件
- 屏蔽鼠标右键
- 对登录密码进行加密
- 字符串加密

实用工具

- 在线查询 IP 地理位置
- 手机号码归属地查询
- 工行在线支付
- 支付宝的在线支付
- 快钱在线支付
- 在线文本编辑器
- 网页拾色器
- 在线验证 18 位身份证
- 在线汉字转拼音
- 在线万年历
- 进制转换工具

高级应用开发

- 自动选择语言跳转
- JSP 防刷计数器
- 用 JSP 操作 XML 实现留言板
- 网站支持 RSS 订阅
- JSP 系统流量分析
- 用 JSP 生成 WEB 静态网页

图形图像与多媒体

- 别致的图形计数器
- 预览并上传图片
- 分页浏览图像
- 生成中文验证码
- 生成关键字验证码
- 生成隐藏的验证码
- 生成带背景和雪花的验证码
- 生成带有干扰线的验证码
- 为图像添加文字水印和图片水印
- 制作相册幻灯片
- 在线连续播放音乐
- 在线播放 FLV 视频
- 同步显示 LRC 歌词
- JSP 生成条形码
- 通过下拉列表框选择头像
- 从网页对话框中选择头像
- 通过滑动鼠标放大或缩小图片
- 随机显示图片
- 幻灯片式图片播放
- 浮动广告
- 插入 Flash 动画
- 插入背景透明的 Flash
- 在线播放 MP3 歌曲列表
- MP3 文件下载
- 自制视频播放器
- 在线影片欣赏

报表打印

- 利用 JavaScript 调用 IE 自身的打印功能实现打印
- 利用 WebBrowser 打印
- 将页面中的客户列表导出到 Word 并打印
- 利用 Word 自动打印指定格式的会议记录
- 利用 Excel 打印工资报表
- 将 Web 页面中的数据导出到 Excel 并自动打印
- 打印库存明细表
- 打印库存盘点报表
- 打印库存汇总报表
- 打印指定条件的库存报表
- 打印汇款单
- 打印信封
- 利用柱形图分析报表
- 利用饼形图分析报表
- 利用折线图分析报表
- 利用区域图分析报表
- 导出报表到 Excel 表格
- 导出报表为 PDF 文档
- 实现打印报表功能
- 实现打印预览功能
- 用 JSP 实现 Word 打印
- JSP+CSS 打印简单的数据报表
- JSP 套打印快递单

JSP 生成便于打印的网页
将数据库中的数据写入到 Excel
将数据库中的数据写入到 Word
……

第 2 大部分　模块资源库

（15 个经典模块，光盘路径：开发资源库/模块资源库）

模块 1　程序打包
- 概述
  - jar 件文
  - MANIFEST 文件
- 使用命令实现程序打包
  - 完成单个文件打包
  - 完成打包多个文件
- 在 Eclipse 中实现程序打包
- 常见问题与解决

模块 2　数码照片管理模块
- 模块概述
  - 设计思路
  - 模块架构
- 关键技术
  - 捕获树的选中节点事件
  - 捕获树的展开节点事件
  - 浏览方式切换技术
  - 随意选取照片技术
  - 图片缩放与内存溢出
  - 工具提示回行显示技术
- 效果预览
- 实现对相册树的维护
- 实现添加照片的功能
- 实现修改照片信息的功能
- 实现删除照片的功能
- 实现全屏查看照片功能
- 实现浏览方式的切换
- 实现查找照片功能
- 实现图片播放器
- 保存选中图片到指定路径

模块 3　FTP 文件管理模块
- FTP 文件管理模块概述
  - 模块简介
  - 功能结构
  - 业务流程
  - 程序预览
- 关键技术
  - 架设 FTP 服务器
  - 登录 FTP 服务器
  - 浏览本地资源
  - 浏览 FTP 服务器资源
  - FTP 文件上传与下载
  - 向 FTP 服务器发送命令
  - 获取文件在本系统的显示图标
  - 任务队列
- 实现 FTP 站点管理功能
  - 装载属性文件
  - 装载 FTP 站点信息
  - 编写站点维护对话框
  - 维护 FTP 站点
- 实现登录面板
- 实现本地资源管理
  - 呈现本地资源
  - 本地资源的控制面板
- 实现 FTP 资源管理
  - 呈现本地资源
  - FTP 服务器资源的控制面板
- 实现队列管理
  - 任务队列
  - 本地队列文件上传
  - FTP 队列文件下载

模块 4　电子地图
- 模块概述
  - 设计思路

- 模块架构
- 效果预览
- **关键技术**
  - Java DB 数据库技术
  - 万年历选择框技术
  - 滑块组件使用技术
  - 列表组件使用技术
  - 维护树模型技术
- **实现地图处理器类**
- **实现用来绘制地图的标签组件**
  - 绘制地图显示区的大地图
  - 绘制鹰眼漫游区的小地图
- **实现操作地图功能**
  - 实现缩放地图功能
  - 实现移动地图功能
- **实现维护标记功能**
  - 实现弹出菜单功能
  - 实现对标记的维护
  - 实现查看标记信息功能
- **实现 搜索标记功能**
  - 实现常用搜索功能
  - 实现高级搜索功能
  - 描红并居中显示选中标记

模块 5　网络五子棋游戏

- **五子棋模块概述**
  - 模块简介
  - 程序预览
- **关键技术**
  - 实现透明的登录界面
  - 监控网络连接状态
  - 绑定属性的 JavaBean
  - 在棋盘中绘制棋子
  - 实现动态调整棋盘大小
  - 游戏悔棋
  - 游戏回放
- **实现登录界面**
- **编写游戏主窗体**
- **编写下棋面板**
- **编写棋盘面板**
- **实现游戏规则算法**
- **编写棋盘模型**
- **编写联机通讯类**

模块 6　远程协助模块

- **远程协助模块介绍**
  - 模块简介
  - 功能结构
  - 程序预览
- **关键技术**
  - 截取屏幕图像
  - 控制计算机的输入
  - 在网络中发送和接收图片
  - RMI 实现远程控制
  - 自定义组件显示远程屏幕
- **联系人管理**
  - 添加联系人
  - 修改联系人
  - 删除联系人
  - 显示联系人列表
  - 处理联系人选择事件
- **创建网络服务器**
- **编写远程连接面板**
- **启动 RMI 远程方法服务**
- **实现远程监控界面**
- **实现系统托盘**

模块 7　软件注册模块

- **软件注册模块概述**
  - 模块概述
  - 功能结构
  - 程序预览
- **关键技术**
  - 读取客户端 MAC 地址
  - Java 操作注册表
  - 避免用户修改系统时间
  - 右键单击弹出菜单
  - 一次性粘贴注册码
  - 获取两个时间的相隔天数
  - ini 文件的读写
  - RSA 加密解密算法

- 软件注册导航窗体的实现
  - 窗体概述
  - 窗体界面设计
  - 软件试用部分的实现
- 软件注册窗体的实现
  - 窗体概述
  - 窗体界面设计
  - 验证注册码
  - 限制注册用户使用时间
  - 根据注册机器的硬件信息保证软件使用唯一性
- 注册机的实现
  - 窗体概述
  - 窗体界面设计
  - 生成注册码

模块 8　多媒体播放器模块

- 模块概述
  - 模块概述
  - 功能结构
  - 程序预览
- 关键技术
  - 安装 jmf-2_1_1e-windows-i586.exe
  - JMF 播放视频文件
  - 设置窗体外观感觉
  - 窗体全屏显示
  - 友情链接
- 实现播放媒体文件
  - 选择本地媒体文件
  - 实现媒体播放
- 实现播放控制
- 播放列表维护
  - 添加列表数据
  - 实现列表“上移”、“下移”
  - 实现列表元素重命名
  - 实现删除列表内容
  - 实现播放列表中元素全部删除
  - 通过双击列表选择播放文件
- 播放控制
- 创建最近播放列表
  - 编写数据库操作方法
  - 动态添加菜单项
- 实现自动检索系统中媒体文件
  - 创建选择文件夹对话框
  - 获取媒体文件集合
  - 将媒体文件添加到播放列表

模块 9　决策分析模块

- 模块概述
  - 设计思路
  - 模块架构
  - 效果预览
- 数据接口
  - 接口设计
  - 测试数据
- 关键技术
  - 支持固定列表格技术
  - 使用 JFreeChart 绘制统计图技术
  - 使用 JavaExcel 生成 Excel 文件
  - 使用 IText 生成 PDF 格式的文件
  - 多线程与进度条的使用
- 实现过程
  - 实现动态控制表格的固定列数量
  - 实现组件间的可用性控制
  - 生成统计图与使用进度条
  - 保存统计图到指定路径
  - 导出报表到 Excel 表格
  - 利用报表和统计图生成 PDF 文件

模块 10　网站留言簿

- 概述与开发环境
  - 概述
  - 开发环境
- 实例运行结果
- 设计与分析
  - 系统分析
  - 系统流程
  - 文件夹及文件架构
  - Hibernate 配置文件及类的分布
- 技术要点
  - 获取留言及回复信息
  - 获取系统日期和时间

保存留言信息时自动插入回复记录
数据表结构
创建 Hibernate 配置文件
创建实体类及映射文件----创建实体类
创建实体类及映射文件----创建映射文件
业务处理逻辑类----对留言及回复信息操作的 Topic 类 1
业务处理逻辑类-----对留言及回复信息操作的 Topic 类 2
业务处理逻辑类----对管理员操作的 Manager 类
创建公共类----ChStr 类和 GetTime 类
创建公共类----style 样式文件和 check 类
添加留言信息
显示留言信息
回复留言
删除留言
用户登录页面
公共页
调试、发布与运行
调试
发布与运行

模块 11　桌面精灵
模块概述
关键技术
实现滚动字幕
实现支持农历的万年历
实现维护记录功能
实现搜索记录功能

模块 12　短信发送模块
短信发送模块概述
关键技术
数据库设计
设置并连接短信猫
读取短信
发送短信
发信箱的实现
联系人管理

模块 13　数据分页
概述与开发环境
实例运行结果
设计与分析
技术要点
开发过程
发布与运行

模块 14　电子阅读器模块
电子阅读模块概述
关键技术
实现主窗体
PDF 文档读取的实现
缩位图的实现
书签的实现
全屏显示 PDF 文档

模块 15　复杂条件查询
……

第 3 大部分　项目资源库

（15 个企业开发项目，光盘路径：开发资源库/项目资源库）

项目 1　Validator 验证框架
……

项目 2　网上办公自动化系统
概述与开发环境
概述
开发环境
需求分析
项目规划

- 系统功能结构图
- **数据库设计**
  - 数据表概要说明
  - 数据表的结构
- **前期准备**
  - 配置应用 Struts 结构文件及数据库连接文件
  - 配置 web.xml 文件
  - 配置 Struts 标签库文件
  - 编写 Struts 框架配置文件
- **网站总体设计**
  - 系统架构设计
  - 类的分布架构设计
  - 文件架构设计
  - 网站首页运行效果图
  - 删除发文子模块
- **公共类的编写**
  - 数据库的连接及操作方法类：DB
  - 数据表信息类：Content
  - 分页类：Page
  - 类型转换类：Change
  - 检查用户权限类：CheckUserAble
  - 解决 Struts 中的中文乱码的类：FormToChinese
  - 检查用户是否已经在线的公共类
- **登录模块**
  - 创建登录的页面：index.jsp
  - 配置 struts-config.xml 文件
  - 创建 LogonForm 类
  - 创建 LogonAction 类
- **自定义标签的开发**
  - 如何开发及使用自定义标签
  - 本例中 office 自定义标签的开发过程
- **收/发文管理模块**
  - 建立发文子模块
  - 浏览发文子模块
- **会议管理模块**
  - 查看会议记录子模块
  - 添加会议记录子模块
  - 浏览会议的详细内容
  - 删除会议子模块
- **公告管理模块**
  - 修改公告子模块
  - 浏览公告信息
  - 查看公告详细信息
  - 添加公告信息
- **人力资源管理模块**
  - 浏览员工信息模块中的查询功能
  - 个人信息子模块
  - 查看员工详细信息
  - 修改员工信息
  - 修改个人信息
  - 添加员工信息
- **文档管理**
  - 浏览文件详细内容
  - 删除文件子模块
  - 文件上传子模块
  - 文件下载子模块
  - 浏览文件信息
- **资产管理模块**
  - 资产管理模块
  - 浏览办公用品信息
  - 更新办公用品信息
  - 添加办公用品信息
  - 浏览车辆信息
  - 浏览车辆详细信息
  - 添加车辆信息
  - 修改车辆信息
  - 资产管理模块公共类
- **内部邮件管理模块**
  - 查看收件箱
  - 查看发件箱
  - 查看邮件详细信息
  - 添加邮件信息
  - 内部邮件管理模块公共类
- **意见箱模块**
  - 查看意见箱
  - 查看意见详细信息
  - 发送新的建议
  - 意见箱模块公共类

- 退出模块
- 疑难解答
- 公共模块

项目 3　图书管理系统

- 概述与开发环境
  - 概述
  - 开发环境
- 需求分析
- 系统设计
  - 项目规划
  - 系统功能结构图
- 数据库设计
  - 数据表概要说明
  - 数据表关系概要说明
  - 主要数据表的结构
- 网站总体设计
  - 系统架构设计
  - 类的分布架构设计
- 公共类的编写
  - 自动转码类
  - 对系统时间操作类
  - 取得自动编号操作类
  - 自动转码类
- IBatis 设计模式的介绍
  - IBatis 设计模式的组成
  - 构建 IBatis 设计模式的 SQL Map 配置代码
- 配置 Struts
- ActionForm 类的编写及配置
  - ActionForm 类的编写及配置
  - 管理员信息 ActionForm 类的代码
  - 图书信息操作的 ActionForm 类
  - 图书借阅信息操作的 ActionForm 类
- 对数据表操作持久类的编写
  - IBatis 设计模式的 SQL Map 映射文件
  - IBatis 基本组件
  - 管理员信息表的操作
  - 图书信息表的操作
  - 图书借阅信息表的操作
- 图书管理系统总体架构
  - 图书管理系统文件架构设计
  - 图书管理系统首页设计
- 管理员功能模块
  - 模块功能介绍
  - 创建管理员的 Action 实现类
  - 系统登录设计
  - PaginatedList 类实现分页显示管理员列表
  - 添加管理员信息
  - 设置管理员权限
  - 删除管理员信息
  - 管理员查询
  - 修改密码
- 图书管理功能模块
  - 图书管理功能模块
  - 创建图书的 Action 实现类
  - 查看图书信息列表
  - 添加图书信息
  - 删除图书信息
  - 查询图书详细信息
  - 修改图书信息
  - 库存进书管理
  - 查询库存量不足的图书
  - 库存查询
- 图书借还管理功能模块
  - 图书借还功能模块
  - 创建图书借阅的 Action 实现类
  - 图书借阅
  - 图书归还
  - 图书续借
  - 借阅查询
- 图书设置模块
  - 图书存放位置管理
  - 图书类别管理
  - 读者类别管理
  - 读者信息管理
  - 图书设置模块公共类
- 图书销售模块

图书销售管理
图书销售查询
图书销售模块公共类
图表分析模块
图书销售图表分析
图书借阅图表分析
图表分析模块公共类
公共模块
疑难问题分析与解决
映射对数据表操作的 XML 文件的路径错误
映射属性的错误操作

项目 4　进销存管理系统
开发背景和系统分析
开发背景
需求分析
可行性分析
编写项目计划书
系统设计
系统目标
系统功能结构
业务逻辑编码规则
系统流程图
构建开发环境
系统预览
文件夹组织结构
数据库设计
数据库分析
进销存管理系统的 E-R 图
使用 PowerDesigner 建模
创建数据库
主窗体设计
创建主窗体
创建导航面板
公共模块设计
编写 Dao 公共类
编写 Item 类
基础信息模块设计
基础信息模块概述
基础信息模块技术分析
供应商添加实现过程
供应商修改与删除实现过程
单元测试
进货管理模块设计
进货管理模块概述
进货管理模块技术分析
进货单实现过程
查询统计模块设计
查询统计模块概述
查询统计模块技术分析
销售查询实现过程
库存管理模块设计
库存管理模块概述
库存管理模块技术分析
价格调整实现过程
单元测试
系统打包发布
开发技巧与难点分析
使用 PowerDesigner 逆向生成数据库 E-R 图

项目 5　基于 Struts 与 Hibernate 开发的新闻网络中心
……

项目 6　网上购物商城
概述与系统分析
总体设计
项目规划
系统功能结构图
系统设计
设计目标
开发及运行环境
逻辑结构设计
技术准备
MVC 概述
Struts 概述
在 MyEclipse 中配置应用 Struts 结构文件
系统架构设计

- 系统文件夹架构图
- 文件架构设计
- **JavaBean 的设计**
  - JavaBean 的设计
  - 数据库连接的 JavaBean 的编写
  - 设置系统中使用的 SQL 语句的 JavaBean
  - 解决 Struts 中文乱码问题
  - 检查用户是否已经在线的公共类
- **会员管理模块**
  - 会员登录
  - 用户注册
  - 找回密码
- **网站主页设计**
  - 网站主页设计
  - 网站首页面导航信息版块
  - 网站首页面左部信息版块
  - 网站首页面右部信息版块
  - 网站首页面版权信息版块
- **会员资料修改模块**
- **购物车模块**
  - 添加购物车
  - 查看购物车
  - 生成订单
  - 清空购物车
- **商品销售排行模块**
  - 商品销售排行榜
  - 分页显示特价商品
- **网站后台主要功能模块设计**
  - 网站后台首页设计
  - 后台管理员身份验证模块
  - 商品设置模块
  - 订单设置模块
- **退出模块**
- **疑难问题分析**
  - 中文乱码问题的处理
  - 关闭网站后 session 没有被注销

项目 7　博客系统

- **概述与系统分析**
  - 总体设计
    - 项目规划
    - 系统功能结构图
  - 系统设计
    - 设计目标
    - 开发及运行环境
    - 逻辑结构设计
  - 技术准备
    - Hibernate 框架概述
    - Hibernate 配置文件
    - 创建持久化类
    - Hibernate 映射文件
  - 系统构架设计
    - 系统文件夹架构图
    - 文件夹架构设计
  - 公共类设计
    - 获得当前系统时间类
    - 字符处理类的编写
    - 将字符串转化成字符数组类
    - Hibernate 的初始化与 Session 管理类的编写
  - 网站前台主要功能设计
    - 网站首页页面设计
    - 网站计数功能实现
    - 网络日历功能
    - 博主信息显示模块
    - 浏览博主发表文章模块
    - 添加评论模块
  - 网站后台主要功能模块设计
    - 网站后台主要功能模块设计
    - 后台首页设计
    - 个人相片设置模块
    - 博主设置模块
    - 博客文章管理模块
  - 疑难问题分析
    - Hibernate 的映射类型
    - 如何使用 Hibernate 声明事务边界
  - 程序调试与错误处理

项目 8　企业内部通信系统

- 开发背景和系统分析
  - 开发背景
  - 需求分析
  - 可行性分析
  - 编写项目计划书
- 系统设计
  - 系统目标
  - 系统功能结构
  - 数据库设计
  - 系统预览
  - 文件夹组织结构
- 主窗体设计
  - 创建主窗体
  - 记录窗体位置
- 公共模块设计
  - 数据库操作类
  - 系统工具类
- 系统托盘模块设计
  - 系统托盘模块概述
  - 系统托盘模块技术分析
  - 系统托盘模块实现过程
- 系统工具模块设计
  - 系统工具模块概述
  - 系统工具模块技术分析
  - 系统工具模块实现过程
- 用户管理模块设计
  - 用户管理模块概述
  - 用户管理模块技术分析
  - 用户管理模块实现过程
  - 单元测试
- 通信模块设计
  - 通信模块概述
  - 通信模块技术分析
  - 通信模块实现过程
- 开发技巧和 JDK 6 新增的系统托盘
  - 开发技巧与难点分析
  - 使用 JDK 6 新增的系统托盘

项目 9　网络购物中心

……

项目 10　企业人事管理系统

- 开发背景和系统分析
  - 开发背景
  - 系统分析
- 系统设计
  - 系统目标
  - 系统功能结构
  - 系统预览
  - 业务流程图
  - 文件夹结构设计
- 数据库设计
  - 数据库分析
  - 数据库概念设计
  - 数据库逻辑结构设计
- 主窗体设计
  - 导航栏的设计
  - 工具栏的设计
- 公共模块设计
  - 编写 Hibernate 配置文件
  - 编写 Hibernate 持久化类和映射文件
  - 编写通过 Hibernate 操作持久化对象的常用方法
  - 创建用于特殊效果的部门树对话框
  - 创建通过部门树选取员工的面板和对话框
- 人事管理模块设计
  - 人事管理模块功能概述
  - 人事管理模块技术分析
  - 人事管理模块实现过程
  - 单元测试
- 待遇管理模块设计
  - 待遇管理模块功能概述
  - 待遇管理模块技术分析
  - 待遇管理模块实现过程
- 系统维护模块设计
  - 系统维护模块功能概述
  - 系统维护模块技术分析
  - 系统维护模块实现过程

单元测试
开发技巧与难点分析
Hibernate 关联关系的建立方法
建立一对一关联
建立一对多关联

项目 11　销售管理系统
……

项目 12　医药进销存管理系统
概述与系统分析
总体设计
系统设计
技术准备
系统总体架构设计
系统公共类设计
Java 实体类及 Hibernate 映射文件的设计
系统功能模块设计
系统主界面设计
设计药品基本情况模块
设计客户基本情况模块
设计供应商基本情况模块
设计药品采购模块
设计药品销售模块
设计库存盘点模块
设计销售退货模块
设计客户回款模块
设计基本信息查询模块
设计入库明细查询模块
设计销售明细查询模块
设计回款信息查询模块
设计增加用户功能模块
设计用户维护功能模块
系统提示模块设计
疑难问题解析
系统常见错误处理
……

项目 15　酒店管理系统
概述与系统分析
总体设计
系统设计
技术准备
系统架构设计
数据持久层设计
主窗体的格局设计
开台签单功能的设计与实现
自动结账功能的设计与实现
销售统计功能的设计与实现
人员管理功能的设计与实现
系统维护功能的设计与实现
系统安全功能的设计与实现
疑难问题分析

第 4 大部分　能力测试资源库

（616 道能力测试题目，光盘路径：开发资源库/能力测试）

第 1 部分　Java 编程基础能力测试
……

第 2 部分　数学及逻辑思维能力测试
基本测试
进阶测试
高级测试

第 3 部分　编程英语能力测试
英语基础能力测试
英语进阶能力测试

第 5 大部分　面试系统资源库

（369 项面试真题，光盘路径：开发资源库/编程人生）

第 1 部分 Java 程序员职业规划

- 你了解程序员吗
- 程序员自我定位

第 2 部分　Java 程序员面试技巧

- 面试的三种方式
- 如何应对企业面试
- 英语面试
- 电话面试
- 智力测试

第 3 部分　Java 常见面试题

- java 语法面试题
- 字符串与数组面试题
- 面向对象试题
- java 异常面试题
- 多线程试题
- 集合类试题
- 数据库相关试题
- 网络与数据流的试题
- 数据结构与算法试题
- 软件工程与设计模式

第 4 部分　Java 企业面试真题汇编

- 企业面试真题汇编（一）
- 企业面试真题汇编（二）
- 企业面试真题汇编（三）
- 企业面试真题汇编（四）

第 5 部分　Java 虚拟面试系统

基础篇

- 第 1 章　初识 Java
- 第 2 章　熟悉 Eclipse 开发工具
- 第 3 章　Java 语言基础
- 第 4 章　流程控制
- 第 5 章　数组
- 第 6 章　字符串
- 第 7 章　类和对象

本篇通过介绍初识 Java、熟悉 Eclipse 开发工具、Java 语言基础、流程控制、数组、字符串、类和对象等内容，并结合大量的图示、实例、视频等使读者快速掌握 Java 语言，为以后编程奠定坚实的基础。

第 1 章

初识 Java

（视频讲解：20 分钟）

Java 是一种可以编写跨平台的、面向对象的程序设计语言。本章将向读者简单介绍 Java 语言和相关的特性、不同的版本以及学好 Java 语言的方法等。初识 Java 所需要掌握的内容，其主要目的是让读者对 Java 语言有一个整体的了解，然后再慢慢地学习具体内容，最后达到完全掌握 Java 语言的目的。

通过阅读本章，您可以：

- 了解 Java 语言的历史
- 了解 Java 语言的现状
- 了解 Java 语言的面向对象特性
- 掌握不同平台的 JDK 环境搭建
- 掌握 Java 程序的编写方法

1.1 Java 语言概述

视频讲解：光盘\TM\lx\1\Java 语言概述.exe

1.1.1 Java 语言的历史

Java 语言是 Sun 公司于 1990 年开发的，当时 Green 项目小组的研究人员正在致力于为未来的智能设备开发一种新的编程语言。由于该小组的成员 James Gosling 对 C++语言在执行过程中的表现非常不满，就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（Oak 就是 Java 语言的前身），这个名字源于 Gosling 办公室窗外的一棵橡树（Oak）。此时的 Oak 已经具备安全性、网络通信、面向对象、多线程等特性，是一款相当优秀的程序语言。后来，在注册 Oak 商标时，发现它已经被另一家公司注册，所以不得不改名。要取什么名字呢？工程师们边喝咖啡边讨论着，看看手上的咖啡，联想到印度尼西亚有一个盛产咖啡的岛屿（中文名叫爪哇），于是将其改名为 Java。

随着 Internet 的迅速发展，Web 应用日益广泛，Java 语言也得到了迅速发展。1994 年，Gosling 用 Java 语言开发了一个实时性较高、可靠、安全、有交互功能的新型 Web 浏览器，它不依赖于任何硬件平台和软件平台。该浏览器被命名为 HotJava，并于 1995 年在业界发表，引起了巨大的轰动，Java 语言的地位随之得到了肯定。1995 年 5 月 23 日，JDK（Java Development Kit）1.0a2 版本正式对外发布。此后，Java 语言的发展异常迅速。2009 年 4 月 20 日，Sun 公司被 Oracle 公司收购。

1.1.2 Java 语言的现状

借助 Java，程序开发人员可以自由地使用现有的硬件和软件系统平台。由于 Java 是独立于平台的，它还可以应用于计算机之外的领域。Java 程序可以在便携式计算机、电视、电话、手机和其他设备上运行。Java 的用途数不胜数，它拥有无可比拟的能力，使用它所节省的时间和费用十分可观。

如果仔细观察就会发现，Java 就在我们身边。使用 Java 语言编写的常见开源软件包括 NetBeans 和 Eclipse 集成开发环境、JBoss 和 GlassFish 应用服务器；商业软件包括永中 Office、合金战士 Chrome、Websphere 和 Oracle Database 11g。此外，各手机厂商都为自己的产品提供了 Java 技术的支持，手机上的 Java 程序和游戏已经不胜枚举。

为了满足不同开发人员的需求，Java 开发分成了以下 3 个方向。

- ☑ Java SE：主要用于桌面程序的开发。它是学习 Java EE 和 Java ME 的基础，也是本书的重点内容。
- ☑ Java EE：主要用于网页程序的开发。随着互联网的发展，越来越多的企业使用 Java 语言来开发自己的官方网站，其中不乏一些世界 500 强。
- ☑ Java ME：主要用于嵌入式系统程序的开发。

1.1.3 Java 语言的面向对象特性

Java 语言的作者们编写了具有广泛影响的 Java 白皮书，里面详细介绍了他们的设计目标以及实现

成果。此外，还用简短的篇幅介绍了 Java 语言的特性。下面将对其进行简要介绍。

1. 简单

Java 语言的语法简单明了，容易掌握，而且是纯面向对象的语言。Java 语言的简单性主要体现在以下几点：

- ☑ 语法规则和 C++类似。从某种意义上讲，Java 语言是由 C 和 C++语言转变而来，所以 C 程序设计人员可以很容易地掌握 Java 语言的语法。
- ☑ Java 语言对 C++进行了简化和提高。例如，Java 使用接口取代了多重继承，并取消了指针，因为指针和多重继承通常使程序变得复杂。Java 语言还通过实现垃圾自动收集，大大简化了程序设计人员的资源释放管理工作。
- ☑ Java 提供了丰富的类库和 API 文档以及第三方开发包，另外还有大量的基于 Java 的开源项目，现在 JDK 也开放源代码了，读者可以通过分析项目的源代码，提高自己的编程水平。

2. 面向对象

Java 语言本身是一种面向对象的程序设计语言。Java 提倡万物皆对象，语法中不能在类外面定义单独的变量和方法，也就是说，Java 语言最外部的数据类型是对象，所有的元素都要通过类和对象来访问。

3. 分布性

Java 的分布性包括操作分布和数据分布，其中操作分布是指在多个不同的主机上布置相关操作，而数据分布是将数据分别存放在多个不同的主机上，这些主机是网络中的不同成员。Java 可以凭借 URL 对象访问网络对象，访问方式与访问本地系统相同。

4. 可移植性

Java 程序具有与体系结构无关的特性，从而使 Java 程序可以方便地移植到网络的不同计算机中。同时，Java 的类库中也实现了针对不同平台的接口，使这些类库也可以移植。

5. 解释性

运行 Java 程序需要解释器。任何移植了 Java 解释器的计算机或其他设备都可以用 Java 字节码进行解释执行。字节码独立于平台，它本身携带了许多编译时信息，使得连接过程更加简单，开发过程也就更加迅速，更具探索性。

6. 安全性

Java 语言删除了类似 C 语言中的指针和内存释放等语法，从而有效地避免了非法操作内存。Java 程序代码要经过代码校验、指针校验等很多的测试步骤才能够运行，所以未经允许的 Java 程序不可能出现损害系统平台的行为，而且使用 Java 可以编写防病毒和防修改的系统。

7. 健壮性

Java 的设计目标之一是编写多方面可靠的应用程序，Java 将检查程序在编译和运行时的错误，以及消除错误。类型检查能帮助用户检查出许多在开发早期出现的错误。同时，很多集成开发工具 IDE（如 Eclipse、NetBeans）的出现使编译和运行 Java 程序更加容易。

8. 多线程

多线程机制能够使应用程序在同一时间并行执行多项任务，而且相应的同步机制可以保证不同线程能够正确地共享数据。使用多线程，可以带来更好的交互能力和实时行为。

9. 高性能

Java 编译后的字节码是在解释器中运行的，所以它的速度比多数交互式应用程序提高了很多。另外，字节码可以在程序运行时被翻译成特定平台的机器指令，从而进一步提高运行速度。

10. 动态

Java 在很多方面比 C 和 C++更能够适应发展的环境，可以动态调整库中方法和变量的增加，而客户端却不需要作任何更改。在 Java 中动态调整是非常简单、直接的。

1.2 JDK 的下载和安装

视频讲解：光盘\TM\lx\1\JDK 的下载和安装.exe

开发 Java 程序必须安装 JDK（Java Development Kit）开发环境，它包含演示程序和样例、Java 公共 API 类的源代码、Java 运行环境、编译调试等工具。本节将介绍如何下载和安装 JDK。

1.2.1 JDK 下载

由于 Sun 公司已经被 Oracle 收购，因此 JDK 可以在 Oracle 公司的官方网站（http://www.oracle.com/index.html）下载。下面以目前最新版本 JDK 6 Update 29 为例介绍下载 JDK 的方法，具体步骤如下：

（1）打开浏览器，进入 Oracle 官方主页，如图 1.1 所示。

图 1.1 Oracle 官方主页

（2）选择 Downloads 选项卡，选择 Java for Developers，在跳转的页面中滚动到如图 1.2 所示位置，单击 JDK 下方的 Download 按钮。

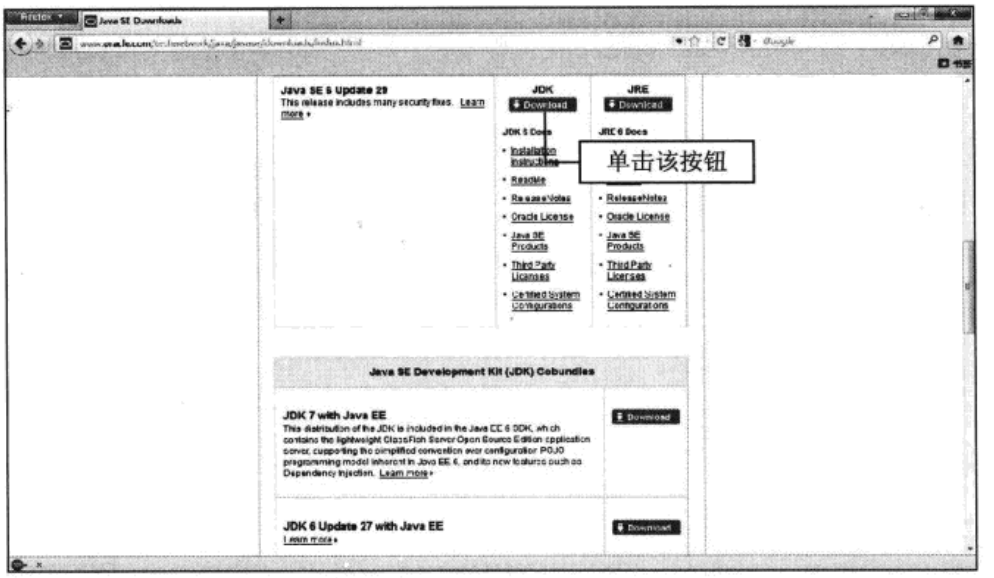


图 1.2　Java 开发资源下载页面

（3）在新页面中，同意协议并根据计算机硬件和系统选择适当的版本进行下载，如图 1.3 所示。

图 1.3　JDK 下载页面

1.2.2　JDK 安装

下载完适合自己系统的 JDK 版本后，就可以进行安装了。这里以 Windows 系统为例，讲解 JDK

的安装过程，具体步骤如下：

（1）双击运行刚刚下载完毕的 JDK 程序，弹出如图 1.4 所示的 JDK 安装向导窗口，单击“下一步”按钮。

（2）弹出如图 1.5 所示的界面，单击“更改”按钮，将安装位置修改为 C:\Java\jdk1.6.0_29\，如图 1.6 所示。

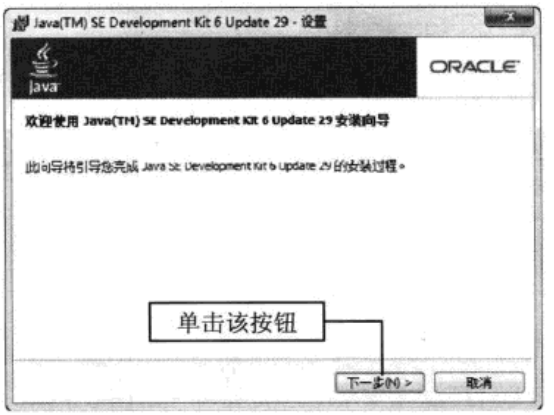


图 1.4　JDK 安装向导窗口

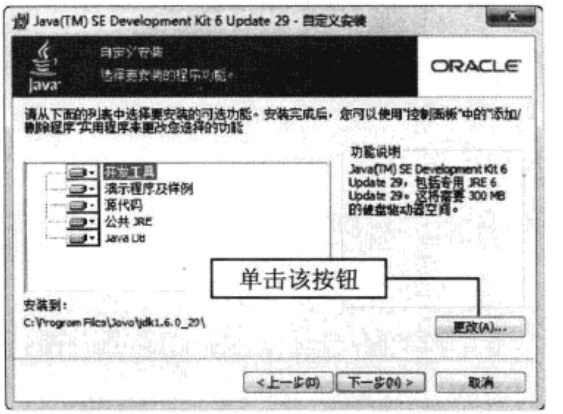


图 1.5　JDK 安装功能及位置选择窗口

注意　由于在 Windows 系统中软件默认安装到 Program Files 文件夹中，这个路径中包含了一个空格，通常建议将 JDK 安装到没有空格的路径中。

（3）单击“下一步”按钮，开始安装，如图 1.7 所示。

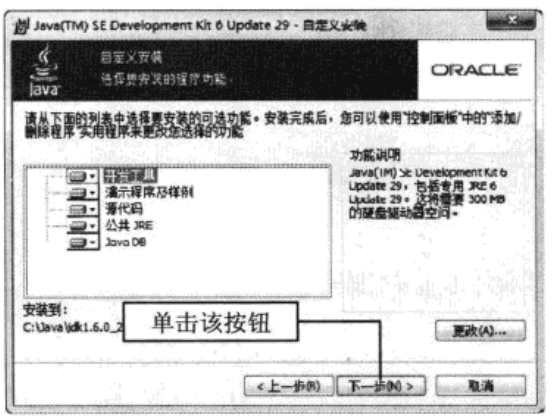


图 1.6　更改安装位置

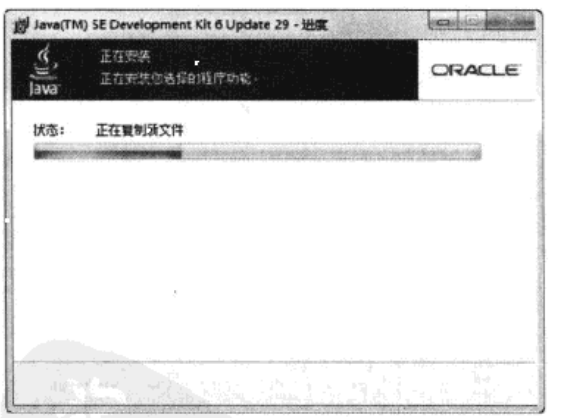


图 1.7　JDK 开始安装

注意

在安装 JDK 时，不要同时运行其他的安装程序，以免出现冲突。

（4）弹出如图 1.8 所示的界面，单击“更改”按钮，将安装路径修改为 C:\Java\jre6\，如图 1.9 所示。

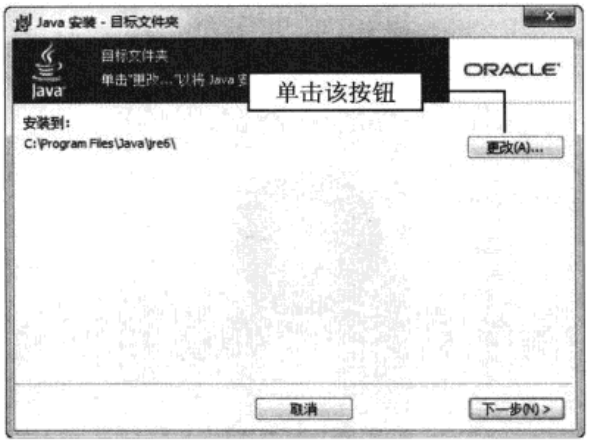


图 1.8　JRE 安装路径选择

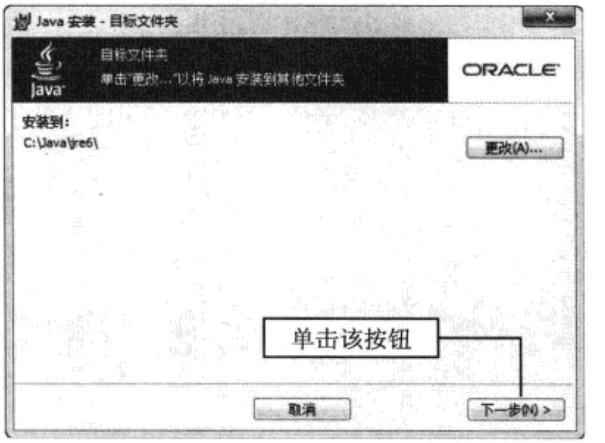


图 1.9　更改 JRE 安装路径

（5）单击“下一步”按钮进行安装，如图 1.10 所示。

（6）图 1.11 显示的是 JDK 安装完成窗口。

图 1.10　JRE 安装进度

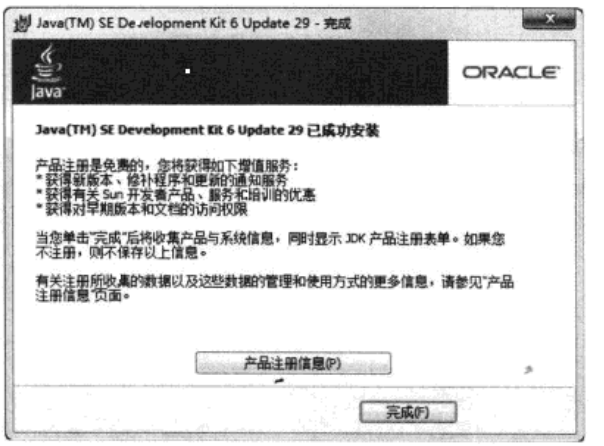


图 1.11　JDK 安装完成窗口

1.2.3　配置 JDK

在安装完 JDK 之后，需要对环境变量进行配置，具体步骤如下：

说明

如果使用集成开发工具，如 Eclipse、NetBeans 等，可以省略此步骤。

（1）在 Windows 7 系统中，同时按住 Win 键和 Pause 键打开“系统属性”窗口，如图 1.12 所示。选择“高级系统设置”选项。

（2）弹出如图 1.13 所示的对话框，单击“环境变量”按钮。

图 1.12　系统基本信息窗口

（3）弹出如图 1.14 所示的“环境变量”对话框，单击“新建”按钮，新建系统变量。

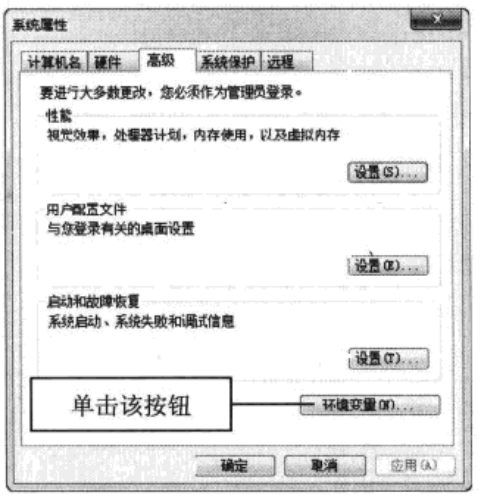


图 1.13　“系统属性”对话框

图 1.14　“环境变量”对话框

（4）弹出“新建系统变量”对话框，在“变量名”文本框中输入“JAVA_HOME”，在“变量值”文本框中输入 JDK 的安装路径“C:\Java\jdk1.7.0_24\”，如图 1.15 所示。单击“确定”按钮，完成环境变量 JAVA_HOME 的配置。

（5）在系统变量中查找 Path 变量，如果不存在，则新建系统变量 Path；否则选中该变量，单击“编辑”按钮，打开“编辑系统变量”对话框，如图 1.16 所示。在“变量值”文本框的起始位置添加“%JAVA_HOME%\bin;”。

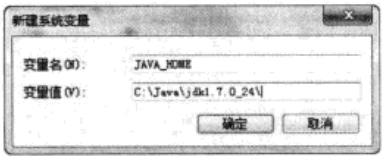


图 1.15　“新建系统变量”对话框

图 1.16　“编辑系统变量”对话框

说明　在 Windows 系统中，环境变量需要使用英文的分号进行分隔；在 Linux 系统中，环境变量需要使用英文的冒号进行分隔。要注意全角和半角的区别。

1.2.4　测试开发环境

JDK 程序的安装和配置完成后，需要测试 JDK 是否能够在计算机上运行。具体测试步骤如下：

同时按住 Win 键和 R 键，打开“运行”窗口。输入“cmd”并单击“运行”按钮，将进入到 DOS 环境中。在命令提示符后面直接输入“javac”，按 Enter 键，系统会输出 javac 的帮助信息，如图 1.17 所示。这说明已经成功配置了 JDK，否则需要仔细检查上面步骤的配置是否正确。

图 1.17　测试 JDK 安装的 DOS 窗口

1.3　第一个 Java 程序

视频讲解：光盘\TM\lx\1\第一个 Java 程序.exe

在完成 JDK 的下载和安装后，就可以开始编写 Java 程序了。下面将编写第一个 Java 程序，其用途是在 DOS 控制台上显示“我能学好 Java 语言！”。通过本节内容，读者可以学习开发 Java 程序的流程。

1．编写源代码

使用 Java 编程的第一步是编写源代码，这里需要使用到文本编辑器。目前有各种各样的文本编辑

器，如“记事本”工具、Office 等。这里使用 Windows 系统自带的“记事本”工具。

选择“开始”/“所有程序”/“附件”/“记事本”命令，然后输入如下内容：

```
public class MyApp {
    public static void main(String[] args) {
        System.out.println("我能学好 Java 语言！");
    }
}
```

按 Ctrl+S 键保存源代码，弹出如图 1.18 所示的窗口。

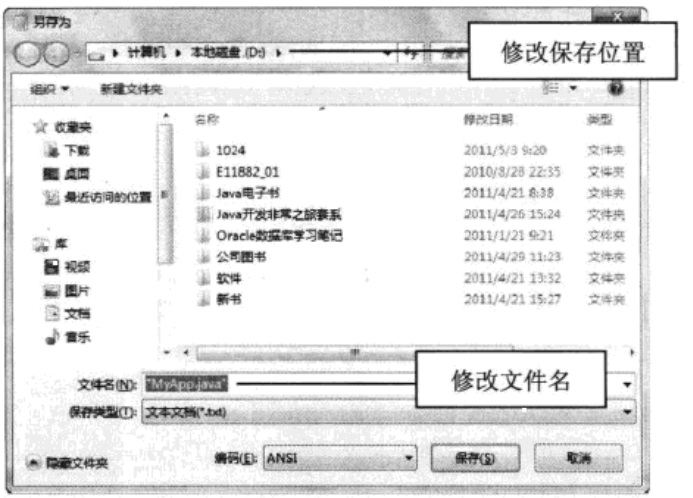


图 1.18　保存窗口

将文件保存到 D 盘，文件名使用 MyApp.java。

注意　默认“记事本”软件会为文件增加扩展名.txt，而 Java 程序的源代码扩展名要使用.java。所以在修改文件名时需要使用双引号将文件名括起来，这样就不会增加新的扩展名。

说明　文件名需要与源代码中公共类的名字相同，这里公共类是 MyApp，因此文件名也是 MyApp。

注意　Java 语言是区分大小写的，因此文件名必须是 MyApp，而不能是 Myapp，或者其他。

2. 编译源代码

选择“开始”/“所有程序”/“附件”/“命令提示符”命令，将路径切换到 D 盘，并使用 dir 命令查看 D 盘中的文件，其运行效果如图 1.19 所示。

接着运行 javac MyApp.java 命令，然后使用 dir 命令查看 D 盘中的文件，其运行效果如图 1.20 所示。

图 1.19　编译前的 D 盘文件列表

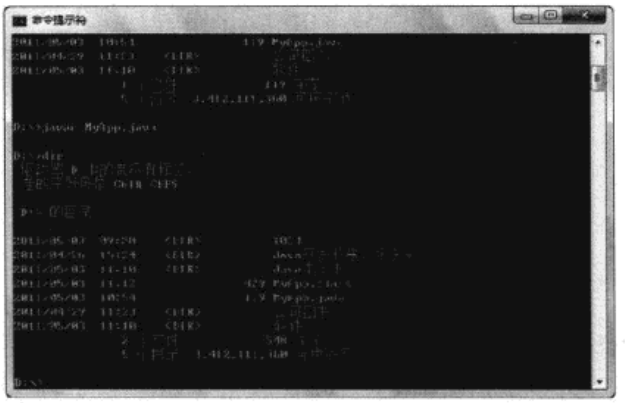


图 1.20　编译后的 D 盘文件列表

读者可以发现，在 D 盘中出现了一个名为 MyApp、类型为 class 的文件。

说明

javac 是 JDK 中提供的编译源代码的命令，由于已经配置了环境变量，因此可以直接使用。

注意

javac 命令和 MyApp.java 之间存在一个空格。

3. 运行 class 文件

输入“java MyApp”来运行 class 文件，其运行效果如图 1.21 所示。

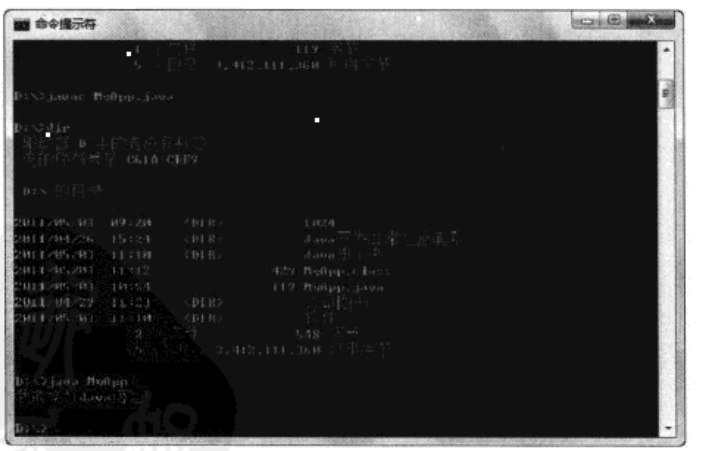


图 1.21　运行 class 文件后的窗体

读者可以看到，在 DOS 控制台上输出了“我能学好 Java 语言！”。

注意 使用 java 命令时，不需要输入文件的扩展名。而使用 javac 命令时，需要输入文件的扩展名。

1.4 本章小结

本章首先简单介绍了 Java 语言的历史、现状及各种特性，接着介绍了 JDK 的安装和配置，这是进行 Java 程序开发的第一步，也是本章的重点。最后介绍了如何开发第一个 Java 应用程序，读者可以从中学习到 Java 开发的一般流程是编写源代码、编译源代码和运行 class 文件。

第 2 章

熟悉 Eclipse 开发工具

（视频讲解：22 分钟）

学习 Java 语言程序设计必须选择一个功能强大、使用简单、能够辅助程序设计的 IDE 集成开发工具，而 Eclipse 是目前最流行的 Java 语言开发工具，它以强大的代码辅助功能，帮助程序开发人员自动完成语法、补全文字、代码修正、API 提示等编码工作，节省了大量的程序开发时间。本章将简要介绍 Eclipse 开发工具，使读者能够初步了解 Eclipse 并使用它完成程序设计工作。

通过阅读本章，您可以：

- 初步了解 Eclipse
- 掌握 Eclipse 的下载和安装
- 掌握 Eclipse 的汉化
- 了解 Eclipse 界面
- 掌握 Eclipse 的使用
- 掌握调试 Eclipse 程序

PDG

2.1 熟悉 Eclipse

视频讲解：光盘\TM\lx\2\熟悉 Eclipse.exe

虽然使用“记事本”工具和 JDK 编译工具已经可以编写 Java 程序，但是在项目开发过程中必须使用大型的 IDE 集成开发工具来编写 Java 程序，这样可以避免编码错误、方便管理项目结构，而且使用 IDE 集成开发工具的代码辅助功能可以快速地录入程序代码。本节将介绍 Eclipse 开发工具，包括它的安装、配置与启动、菜单栏、工具栏和各种视图的作用等。

2.1.1 Eclipse 简介

Eclipse 是由 IBM 公司投资 4000 万美元开发的 IDE 集成开发工具。它是目前最流行的 Java 集成开发工具之一，基于 Java 语言编写，并且是开放源代码的、可扩展的（Integrated Development Environment，IDE）开发工具。另外，IBM 公司捐出 Eclipse 源代码，组建了 Eclipse 联盟，由该联盟负责这种工具的后续开发。Eclipse 为编程人员提供了一流的 Java 程序开发环境。它的平台体系结构是在插件概念的基础上构建的，插件是 Eclipse 平台最具特色的特征之一，也是其区别于其他开发工具的特征之一。学习完本章之后，读者将对 Eclipse 有一个初步的了解，为后面的深入学习做铺垫。

2.1.2 Eclipse 的下载与安装

（1）打开浏览器，进入 Eclipse 官方主页，地址是 http://www.eclipse.org，如图 2.1 所示。

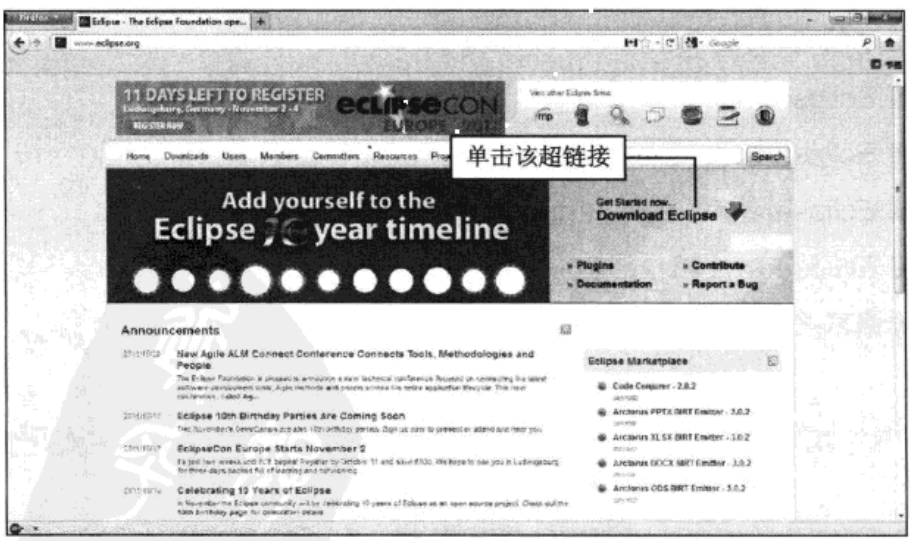


图 2.1 Eclipse 官方主页

（2）单击 Download Eclipse 超链接，在弹出的界面中根据读者的操作系统不同，在 Eclipse IDE for Java Developers 标签右侧选择适当的版本，如图 2.2 所示。

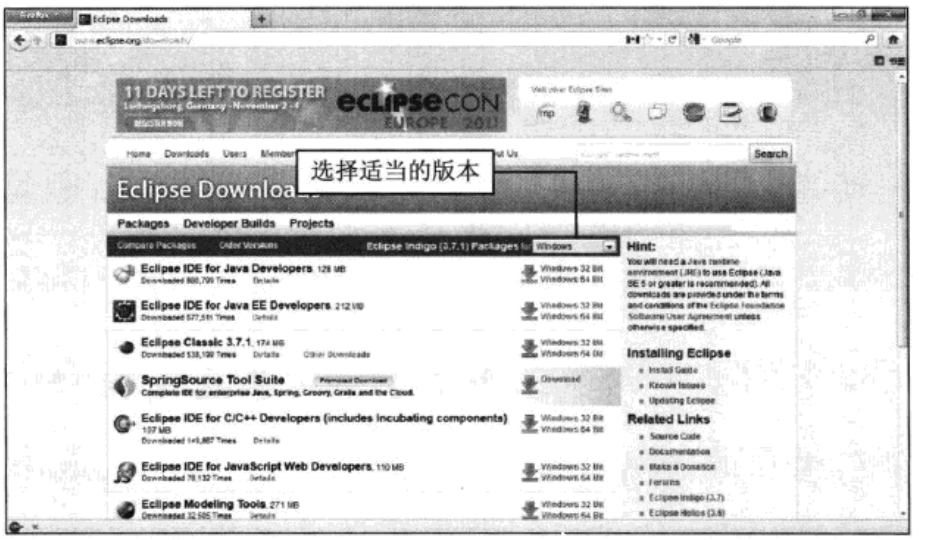


图 2.2　Eclipse 版本选择页面

（3）弹出如图 2.3 所示的界面，单击[China] Beijing Institute of Technology (http)超链接进行下载。

图 2.3　Eclipse 下载页面

（4）将下载完成的压缩包进行解压缩，就完成了 Eclipse 的安装。

2.1.3　Eclipse 汉化

为了方便不熟悉英语的用户使用 Eclipse，下面讲解如何对 Eclipse 进行汉化。

（1）打开浏览器，进入 Eclipse Babel 官方主页，地址是 http://www.eclipse.org/babel/，如图 2.4 所示。

图 2.4　Eclipse Babel 官方主页

（2）单击页面左侧的 Downloads 超链接，弹出如图 2.5 所示的界面，复制 Babel Language Pack Update Site for Indigo 下方的 http://download.eclipse.org/technology/babel/update-site/R0.9.0/indigo 超链接。

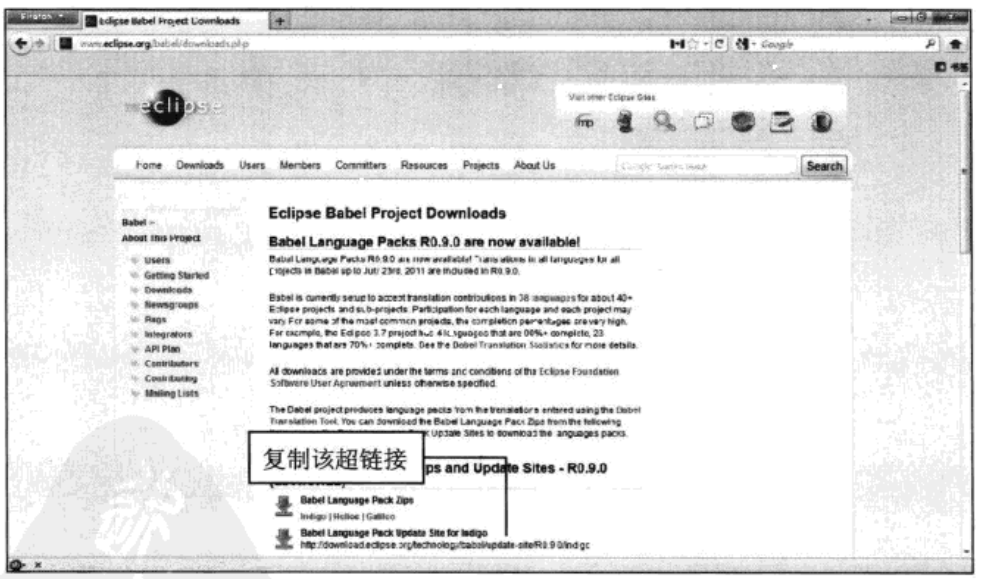


图 2.5　Eclipse Babel 下载页面

（3）启动 Eclipse，选择 Help 菜单，如图 2.6 所示。

（4）选择 Install New Software 命令，弹出如图 2.7 所示的对话框。

（5）单击 Add 按钮，显示 Add Repository（增加仓库）对话框，如图 2.8 所示。在 Name 文本框中输入“Eclipse Babel”，在 Location 文本框中输入刚刚复制的网址，单击 OK 按钮联网查找可用软件包。

图 2.6　Eclipse 的 Help 菜单

图 2.7　“安装新插件”对话框

（6）弹出如图 2.9 所示的界面，选择 Babel Language Packs for eclipse 下的 Babel Language Pack for eclipse in Chinese(Simplified)，单击 Next 按钮。

图 2.8　“增加仓库”对话框

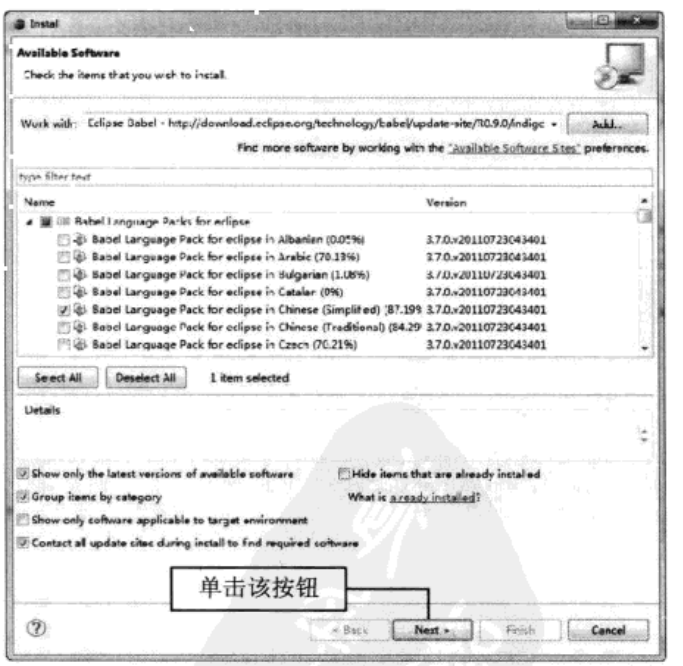


图 2.9　“安装新插件”对话框

（7）弹出如图 2.10 所示的界面，其中显示了插件的详细信息，包括插件名称、版本号和 ID。单

击 Next 按钮继续。

图 2.10　选择安装插件的详细信息

（8）弹出如图 2.11 所示的安装协议界面，选中 I accept the terms of the license agreement 单选按钮同意协议，然后单击 Next 按钮开始安装。

（9）弹出如图 2.12 所示的窗口，显示了插件的下载进度。

图 2.11　选择安装插件的详细信息

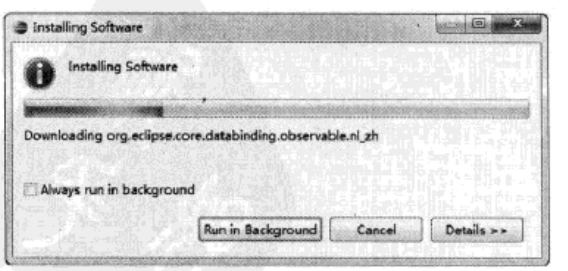


图 2.12　插件的下载进度

（10）在下载完成后会询问是否安装该插件，如图 2.13 所示。

（11）单击 OK 按钮确认安装，在安装完成后会要求重新启动 Eclipse，如图 2.14 所示。单击 Restart Now 按钮完成安装。

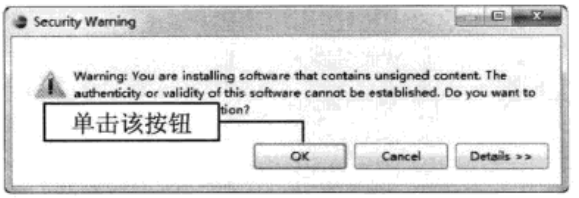


图 2.13　询问是否安装该插件

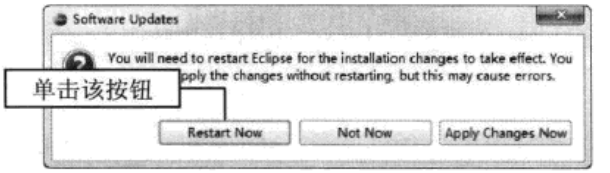


图 2.14　询问是否重启 Eclipse

2.1.4　Eclipse 工作台

现在已经配置好 Eclipse 的多国语言包，可以启动 Eclipse 了。在 Eclipse 的安装文件夹中运行 eclipse.exe 文件，Eclipse 在启动时会弹出“工作空间启动程序”对话框，该对话框用于设置 Eclipse 的工作空间（工作空间用于保存 Eclipse 建立的程序项目和相关设置）。本书的开发环境统一设置工作空间为 Eclipse 安装位置的 workspace 文件夹。在“工作空间启动程序”对话框的“工作空间”下拉列表框中输入“.\workspace”，如图 2.15 所示，单击“确定”按钮，将启动 Eclipse。

注意　在每次启动 Eclipse 时，Eclipse 都会出现设置工作空间的对话框，通过选中“将此值用作缺省值并且不再询问”复选框可以设置默认工作空间，这样 Eclipse 启动时就不会再询问工作空间的设置了。

Eclipse 首次启动时会显示欢迎界面，其中包含了 Eclipse 的概述、新增内容、样本、教程以及工作台的相关按钮，如图 2.16 所示。

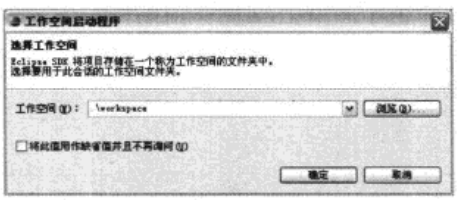


图 2.15　设置工作空间

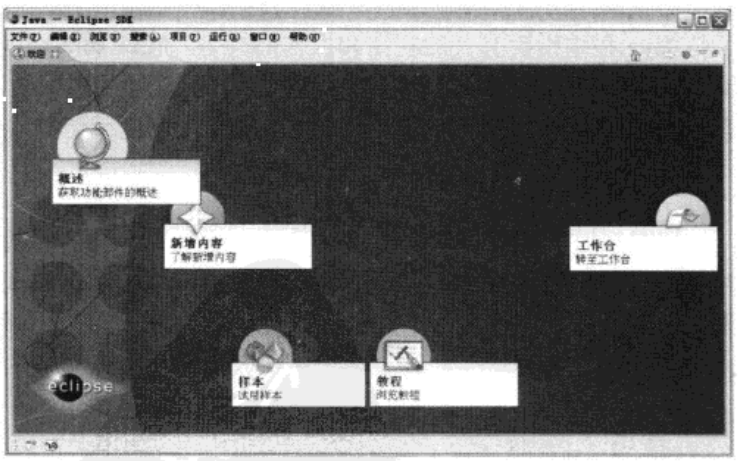


图 2.16　Eclipse 的欢迎界面

在 Eclipse 的欢迎界面中，单击“工作台”按钮或者关闭欢迎视图界面，将显示 Eclipse 的工作台。

它是程序人员开发程序的主要场所。Eclipse 还可以将各种插件无缝地集成到工作台中，也可以在工作台中开发各种插件。Eclipse 工作台主要包括标题栏、菜单栏、工具栏、编辑器、透视图和相关的视图等，如图 2.17 所示。下面将介绍 Eclipse 的菜单栏与工具栏，以及什么是透视图、视图，并介绍常用视图。

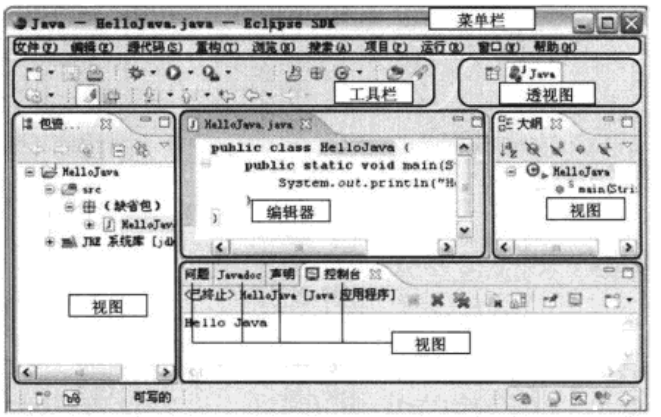


图 2.17　缩放后的 Eclipse 工作台

2.1.5　透视图与视图

透视图和视图是 Eclipse 中的概念，下面将分别介绍什么是透视图、什么是视图，以及它们在 Eclipse 中的作用。

1．透视图

透视图是 Eclipse 工作台提供的附加组织层，也就是说，透视图起到一个组织的作用，它实现多个视图的布局和可用操作的集合，并为这个集合定义一个名称。例如，Eclipse 提供的 Java 透视图就组织了与 Java 程序设计有关的视图和操作的集合，而 **调试** 透视图负责组织与程序调试有关的视图和操作集。在 Eclipse 的 Java 开发环境中提供了几种常用的透视图，如 Java 透视图、资源透视图、调试透视图、小组同步透视图等。不同的透视图之间可以进行切换，但是同一时刻只能使用一个透视图。

2．视图

视图多用于浏览信息的层次结构、显示活动编辑器的属性，如“控制台”视图用于显示程序运行时的输出信息和异常错误，而“包资源管理器”视图可以浏览项目的文件组织结构。视图可以单独出现，也可以与其他视图以选项卡样式叠加在一起，它们可以有自己独立的菜单和工具栏，并且可以通过拖曳随意改变布局位置。

技巧　在视图标题上单击鼠标右键，在弹出的快捷菜单中选择“已拆离”命令可以将视图从 Eclipse 工作台界面中分离，以单独窗口存在，同样的操作也可以将视图再次合并到工作台界面中。本书大部分视图都是从 Eclipse 中分离后才截取的界面效果。

2.1.6　菜单栏

Eclipse 的菜单栏包含了 Eclipse 的基本命令，在使用不同的编辑器时，还会动态地添加有关该编辑器的菜单。基本的菜单栏中除了常用的“文件”、“编辑”、“窗口”和“帮助”等菜单以外，还提供了一些功能菜单，如“源代码”和“重构”等，如图 2.18 所示。

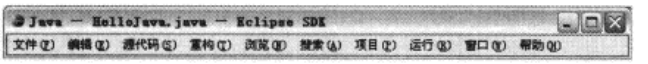


图 2.18　Eclipse 的菜单栏

每个菜单中都包含不同的命令，这些命令用于完成最终的操作，如文件的打开与保存、代码格式化、程序的运行与分步调试等。每个菜单所包含的命令如图 2.19 所示。

菜单中的命令虽多，但并不是所有命令都经常使用。本节将介绍几个最常用的菜单及其命令，至于其他不常用的菜单，读者可以在日后程序开发过程中慢慢掌握。

图 2.19　Eclipse 菜单命令

1. “文件”菜单

“文件”菜单中包含“新建”、“保存”、“关闭”、“打印”、“切换工作空间”、“属性”等命令。菜单中包含的内容虽多，但也有常用和不常用的。在此将“文件”菜单中的常用命令都集中列在表 2.1 中，这样阅读和查询更方便。

表 2.1 “文件”菜单中的常用命令

图　标	命　令	说　明	快捷键
	新建	创建新项目、元素或资源	Shift+Alt+N
	打开文件	打开已经存在的文件	
	关闭	关闭当前编辑器	Ctrl+W
	全部关闭	关闭所有编辑器	Shift+Ctrl+W
	保存	保存当前编辑器的内容	Ctrl+S
	刷新	刷新所选元素的内容	F5
	切换工作空间	切换工作空间到其他位置。这将导致 Eclipse 重启	
	导入	打开“导入向导”对话框	
	导出	打开“导出向导”对话框	
	属性	打开所选元素的属性对话框	Alt+Enter

2. “编辑”菜单

“编辑”菜单用于辅助程序代码设计工作，除常用的“剪切”、“复制”、“粘贴”命令之外，还提供了“快速修正”、“将选择范围扩展到”和“内容辅助”等高级命令。常用命令如表 2.2 所示。

表 2.2 “编辑”菜单中的常用命令

图　标	命　令	说　明	快捷键
	将选择范围扩展到	将选择编辑内容的范围扩大到外层元素、下一个元素、上一个元素或者恢复上一次选择的元素	
	查找/替换	搜索编辑器中的内容片段，并根据需要替换为新的内容	Ctrl+F
	查找下一个	搜索当前所选内容下一次出现的地方	Ctrl+K
	查找上一个	搜索当前所选内容上一次出现的地方	Shift+Ctrl+K
	添加书签	当前光标所在行添加书签	
	添加任务	当前光标所在行添加任务	
	灵活插入方式	切换插入方式。当禁用灵活插入方式时，将禁用自动缩进、添加右方括号等辅助功能	Shift+Ctrl+Insert
	内容辅助	在当前光标位置打开内容辅助对话框	
	文字补全	补全当前编辑器中正在输入的文字	Ctrl+Alt+/
	快速修正	如果光标位于问题代码附近，则打开一个解决方案对话框	Ctrl+1

3. “源代码”菜单

“源代码”菜单中包含的命令都是和代码编写相关的一些命令，主要用于辅助编程。常用命令如表 2.3 所示。

表 2.3　“源代码”菜单中的常用命令

图　标	命　令	说　明	快 捷 键
	切换注释	注释或取消注释当前选择的所有行	Ctrl+/
	添加块注释	在当前选择的多行代码周围添加块注释	Shift+Ctrl+/
	除去块注释	从当前选择的多行代码中除去块注释	Shift+Ctrl+\
	更正缩进	更正当前选择的代码行的缩进	Ctrl+I
	格式	使用代码格式化程序来格式化当前 Java 代码	Shift+Ctrl+F
	组织导入	导入当前类所使用的类包	Shift+Ctrl+O
	覆盖/实现方法	使用向导覆盖父类或实现接口中的方法	
	生成 Getter 和 Setter	使用向导创建成员变量的 getXXX()/setXXX()方法	
	生成 hashCode()和 equals()	打开“生成 hashCode()和 equals()”对话框	
	使用字段生成构造函数	添加构造函数，这些构造函数初始化当前选择的类型的字段，可用于类型、字段或类型中的文本选择	
	从超类中生成构造函数	对于当前所选择的类型，按照超类中的定义来添加构造函数	
	包围方式	使用代码模板包围所选语句	Shift+Alt+Z
	外部化字符串	打开“将字符串外部化”向导，此向导允许通过使用语句访问属性文件来替换代码中的所有字符串	

4. “重构”菜单

“重构”菜单是 Eclipse 最关键的菜单，主要包括对项目重构的相关命令，是读者最应该掌握的。常用命令如表 2.4 所示。

表 2.4　“重构”菜单中的常用命令

图　标	命　令	说　明	快 捷 键
	重命名	重命名所选择的 Java 元素	Shift+Alt+R
	移动	移动所选择的 Java 元素	Shift+Alt+V
	抽取方法	创建一个包含当前所选择的语句或表达式的新方法，并相关地引用	Shift+Alt+M
	抽取局部变量	创建为当前所选择的表达式指定的新变量，并将选择替换为对新变量的引用	Shift+Alt+L
	抽取常量	从所选表达式创建静态终态字段并替换字段引用，并且可以选择重写同一表达式的其他出现位置	
	内联	直接插入局部变量、方法或常量	Shift+Alt+I
	将匿名类转换为嵌套类	将匿名内部类转换为成员类	
	将成员类型转换为顶级	为所选成员类型创建新的 Java 编译单元，并根据需要更新所有引用	
	将局部变量转换为字段	如果该变量是在创建时初始化的，则此操作将把初始化移至新字段的声明或类的构造函数	
	抽取超类	从一组同类型中抽取公共超类	
	抽取接口	根据当前类的方法创建接口，并使该类实现这个接口	
	包括字段	将对变量的所有引用替换为 getXXX()/setXXX()方法	
	历史记录	浏览工作空间重构历史记录，并提供用于从重构历史记录中删除重构的选项	

5. “项目”菜单

“项目”菜单主要用于管理 Eclipse 中的项目，可以实现项目的打开与关闭、自动构建等操作。常用命令如表 2.5 所示。

表 2.5 “项目”菜单中的常用命令

图　标	命　令	说　明	快 捷 键
	打开项目	打开选择的已关闭的项目	
	关闭项目	关闭当前所选择的项目	
	全部构建	在工作空间中构建所有项目	Ctrl+B
	构建项目	构建当前所选择的项目	
	构建工作集	构建当前工作集中所有项目	
	清理	选择要清理的项目	
	自动构建	选择此项，在保存所有已修改的文件时都将自动重建它们	
	生成 Javadoc	对当前选择的项目打开“生成 Javadoc”向导	
	属性	查看选择项目的属性	

6. “运行”菜单

“运行”菜单中列出了与程序运行和调试相关的各种操作，同时还能保存运行和调试的记录。常用命令如表 2.6 所示。

表 2.6 “运行”菜单中的常用命令

图　标	命　令	说　明	快 捷 键
	运行上次启动	快速重复最近的程序启动	Ctrl+F11
	调试上次启动	快速重复最近的程序调试	F11
	运行历史记录	显示以运行方式启动的启动配置的最近历史记录的子菜单	
	运行方式	显示当前程序所支持的运行方式	
	运行	打开“启动配置”对话框来管理运行方式启动配置	
	调试历史记录	显示最近程序调试的历史记录	
	调试方式	显示当前程序所支持的调试方式	
	调试	打开“调试配置”对话框来管理调试配置	
	切换行断点	添加或除去 Java 编辑器中当前行的断点	Shift+Ctrl+B
	切换观察点	添加或除去当前 Java 变量的观察点	
	跳过所有断点	跳过已经设定的所有断点	
	除去所有断点	从“断点”视图中去除所有断点	
	添加 Java 异常断点	添加异常断点	

7. “窗口”菜单

“窗口”菜单主要用于显示、隐藏或处理 Eclipse 中的各种视图、透视图和操作。常用命令如表 2.7 所示。

表 2.7　“窗口”菜单中的常用命令

图　标	命　令	说　明	快 捷 键
·	打开透视图	在 Eclipse 中打开新的透视图	
	显示视图	在当前透视图中显示指定的视图	
	将透视图另存为	将当前透视图布局保存为一个新的透视图	
	复位透视图	将当前透视图的布局复位到上次保存的状态	
	关闭透视图	关闭当前透视图	
	关闭所有透视图	关闭 Eclipse 中所有打开的透视图	
	工作集	有关工作集的管理操作	
	首选项	打开 Eclipse 的首选项对话框，对 Eclipse 进行系统的设置	

2.1.7　工具栏

Eclipse 的工具栏位于菜单栏的下方，这与大多数软件的布局格式相同。工具栏中的按钮都是菜单命令对应的快捷图标，在打开不同的编辑器时还会动态地添加与编辑器相关的新工具栏按钮。另外，除了菜单栏下面的工具栏称为主工具栏之外，Eclipse 中还有视图工具栏、透视图工具栏和快速视图工具栏等多种工具栏。

1．主工具栏

主工具栏就是位于 Eclipse 菜单栏下方的工具栏，其内容将根据不同的透视图和不同类型的编辑器显示相关工具按钮，如图 2.20 所示

图 2.20　Eclipse 主工具栏

2．视图工具栏

Eclipse 界面中包含多种视图，这些视图各有不同的用途，根据视图的功能需求会在视图的标题栏位置添加相应的视图工具栏。例如，“控制台”视图用于输出程序运行中的输出结果和运行时异常信息，其工具栏如图 2.21 所示。

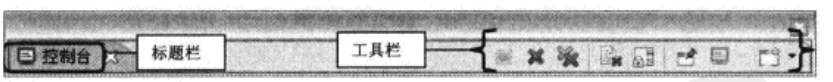


图 2.21　“控制台”视图的标题栏和工具栏

3．透视图工具栏

透视图工具栏主要包括切换已经打开的不同透视图的缩略按钮、选择打开其他视图的按钮。在相应的工具按钮上单击鼠标右键会弹出透视图的管理菜单，实现透视图的定制、关闭、复位、布局位置、是否显示文本等操作，如图 2.22 所示。

4. 快速视图工具栏

快速视图工具栏包含了一组快速视图的图标按钮，该工具栏中的视图会以图标按钮形式隐藏，当单击指定工具按钮时显示指定的视图，而该视图失去焦点时会自动隐藏到快速工具栏。默认情况下，快速视图工具栏出现在工作台的左下角，如图 2.23 所示。

图 2.22　透视图工具栏

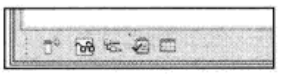


图 2.23　Eclipse 中的快速视图工具栏

2.1.8　“包资源管理器”视图

该视图用于浏览项目结构中的 Java 元素，包括包、类、类库的引用等，但最主要的用途还是操作项目中的源代码文件。“包资源管理器”视图的界面如图 2.24 所示。

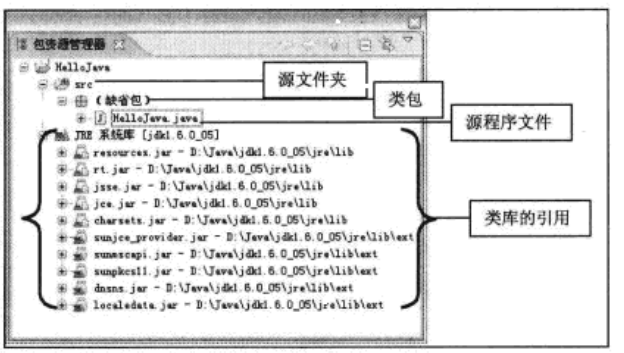


图 2.24　拆分后的“包资源管理器”视图

2.1.9　“控制台”视图

“控制台”视图用于显示程序运行时的输出结果和运行时的异常信息（Runtime Exception），在学习 Swing 程序设计之前，必须使用控制台实现与程序的交互，例如为方便某个方法的调试，在方法执行前后分别输出“方法开始”和“方法结束”信息。“控制台”视图的界面如图 2.25 所示。

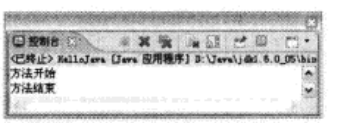


图 2.25　拆分后的“控制台”视图

2.2　使用 Eclipse

视频讲解：光盘\TM\lx\2\使用 Eclipse.exe

现在读者对 Eclipse 工具应该有个大体的认识了，本节将介绍如何使用 Eclipse 完成 HelloJava 程序

的编写和运行。

2.2.1　创建 Java 项目

在 Eclipse 中编写程序，必须先创建项目。Eclipse 中有很多种项目，其中 Java 项目用于管理和编写 Java 程序。创建该项目的步骤如下：

（1）选择“文件”/“新建”/“项目”命令，打开“新建项目”对话框。该对话框包含创建项目的向导，在向导中选择“Java 项目”节点，单击“下一步”按钮。

（2）在弹出的“新建 Java 项目”对话框的“项目名”文本框中输入“HelloJava”，在“项目布局”栏中选中“创建单独的源文件夹和输出文件夹”单选按钮，如图 2.26 所示，单击“完成”按钮。

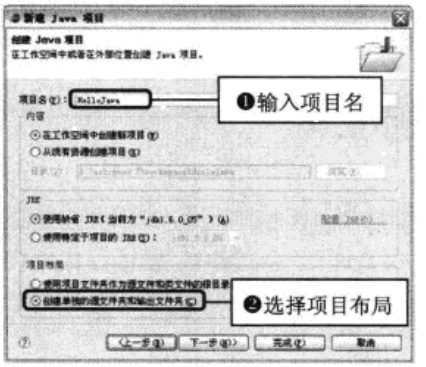


图 2.26　“新建 Java 项目”对话框

2.2.2　创建 Java 类文件

创建 Java 类文件时，自动打开 Java 编辑器。创建 Java 类文件可以通过“新建 Java 类”向导来完成。在 Eclipse 菜单栏中选择“文件”/“新建”/“类”命令，打开“新建 Java 类”向导对话框，如图 2.27 所示。

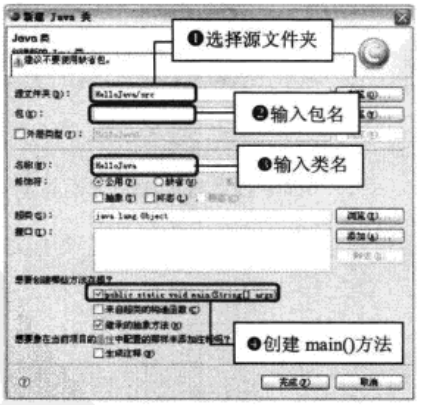


图 2.27　“新建 Java 类”向导对话框

使用该向导对话框创建 Java 类的步骤如下：

（1）在“源文件夹”文本框中输入项目源程序文件夹的位置。通常向导会自动填充该文本框，没有特殊情况，不需要修改。

（2）在“包”文本框中输入类文件想使用的包名，这里暂时默认为空，不输入任何信息，这样就会使用 Java 工程的“缺省包”。

（3）在“名称”文本框中输入新建类的名称，如 HelloJava。

注意 虽然 HelloJava 类名与 Java 项目同名，但是它们分别代表类文件和 Java 项目，读者要注意区分它们的含义。

（4）选中 public static void main(String[] args)复选框，向导在创建类文件时，会自动为该类添加 main()方法，使该类成为可以运行的主类。

2.2.3 使用编辑器编写程序代码

编辑器总是位于 Eclipse 工作台的中间区域，该区域可以重叠放置多个编辑器。编辑器的类型可以不同，但是它们的主要功能都是完成 Java 程序、XML 配置等代码编写或可视化设计工作。本节将向读者介绍如何使用 Java 编辑器和其代码辅助功能快速编写 Java 程序。

1. 打开 Java 编辑器

在使用向导创建 Java 类文件之后，会自动打开 Java 编辑器编辑新创建的 Java 类文件。除此之外，打开 Java 编辑器最常用的方法是在“包资源管理器”视图中双击 Java 源文件或者右击 Java 源文件，在弹出的快捷菜单中选择“打开方式”/“Java 编辑器”命令。Java 编辑器的界面如图 2.28 所示。

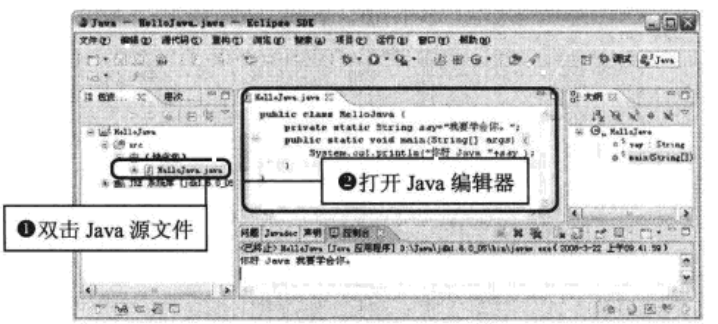


图 2.28 Java 编辑器的界面

从图 2.28 中可以看到，Java 编辑器以各种样式和不同的颜色突出显示 Java 语法。这些突出显示的语法如下：

- ☑ 程序代码注释
- ☑ Javadoc 注释
- ☑ Java 关键字

☑ 变量

☑ 字符串

技巧 在 Java 编辑器左侧单击鼠标右键，在弹出的快捷菜单中选择“显示行号”命令，可以开启 Java 编辑器显示行号的功能。

2. 编写 Java 代码

Eclipse 的强大之处并不在于编辑器能突出显示 Java 语法，而是它强大的代码辅助功能。在编写 Java 程序代码时，可以使用 Ctrl+Alt+/快捷键自动补全 Java 关键字，也可以使用 Alt+/快捷键启动 Eclipse 代码辅助菜单。使用 Eclipse 的代码辅助功能编写 HelloJava 类的步骤如下：

在使用向导创建 HelloJava 类之后，向导会自动构建 HelloJava 类结构的部分代码，并且已经建立了 main()方法，程序开发人员需要做的就是将代码补全，为程序添加相应的业务逻辑。本程序的完整代码如图 2.29 所示。

在 HelloJava 程序代码中，第 1、3、5、6 行是由向导创建的，完成这个程序只要编写第 2 行和第 4 行代码即可。

首先来看一下第 2 行代码。它包括 private、static、String 3 个关键字，这 3 个关键字如果在“记事本”中手动输入可能不会费多长时间，但是无法避免出现输入错误的情况，例如将 private 关键字输入为 privat，缺少了字母 e，这个错误可能在程序编译时才会被发现。这只是简单的关键字，如果是名称更长、更复杂的关键字，就更容易出现错误。在 Eclipse 的 Java 编辑器中可以输入关键字的部分字母，然后使用 Ctrl+Alt+/快捷键自动补全 Java 关键字，如图 2.30 所示。

```
1 public class HelloJava {
2     private static String say="我要学会你。";
3     public static void main(String[] args) {
4         System.out.println("你好 Java "+say );
5     }
6 }
```

图 2.29　HelloJava 程序代码

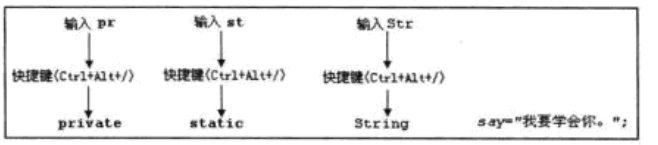


图 2.30　使用快捷键补全关键字

其次是第 4 行的程序代码。它使用 System.out.println()方法输出文字信息到控制台，这是程序开发时最常使用的方法之一。当输入“.”操作符时，编辑器会自动弹出代码辅助菜单，也可以在输入部分文字之后使用 Alt+/快捷键调出代码辅助菜单，完成关键语法的输入，如图 2.31 所示。

注意 System.out.println()方法在 Java 编辑器中可以通过输入“syso”和按 Alt+/键完成快速输入。

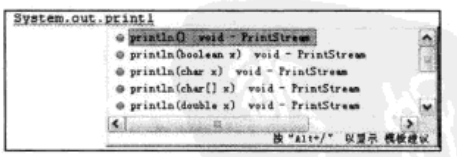


图 2.31　代码辅助菜单

技巧 将光标移动到 Java 编辑器的错误代码位置，按 Ctrl+1 键可以激活代码修正菜单，进而选择菜单中合适的修正方法。

2.2.4 运行 Java 程序

HelloJava 类包含 main()主方法，它是一个可以运行的主类。在 Eclipse 中运行 HelloJava 程序，可以在“包资源管理器”视图中右击 HelloJava 文件，在弹出的快捷菜单中选择“运行方式”/“Java 应用程序”命令。运行结果如图 2.32 所示。

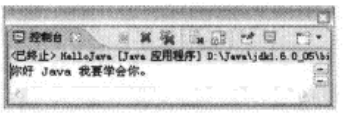


图 2.32 HelloJava 程序在控制台的输出结果

2.3 程序调试

视频讲解：光盘\TM\lx\2\程序调试.exe

读者在程序开发过程中会不断体会到程序调试的重要性。为验证 Java 单元的运行状况，经常在某个方法调用的开始和结束位置分别使用 System.out.println()方法输出状态信息，根据这些信息判断程序执行状况，这是非常古老的程序调试方法，而且经常导致程序代码混乱（导出都是 System.out.println()方法）。

本节将简单介绍 Eclipse 内置的 Java 调试器的使用方法，使用该调试器可以设置程序的断点、实现程序单步执行、在调试过程中查看变量和表达式的值等调试操作，这样可以避免在程序中编写大量的 System.out.println()方法输出调试信息。

使用 Eclipse 的 Java 调试器需要设置程序断点，然后使用单步调试分别执行程序代码的每一行。

1. 断点

设置断点是程序调试中必不可少的有效手段，Java 调试器每次遇到程序断点时都会将当前线程挂起，即暂停当前程序的运行。

可以在 Java 编辑器中显示代码行号的位置双击添加或删除当前行的断点，或者在当前行号的位置单击鼠标右键，在弹出的快捷菜单中选择“切换断点”命令实现断点的添加与删除，如图 2.33 所示。

2. 以调试方式运行 Java 程序

在 Eclipse 中调试 HelloJava 程序，可以在“包资源管理器”视图中右击 HelloJava 文件，在弹出的快捷菜单中选择“调试方式”/“Java 应用程序”命令。图 2.33 中在第 4 行代码设置了断点，调试器将在该断点处挂起当前线程，使程序暂停，如图 2.34 所示。

图 2.33　Java 编辑器中的断点

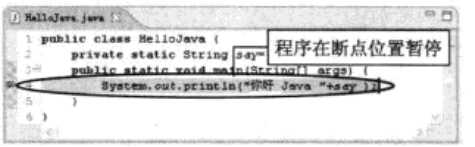


图 2.34　程序执行到断点后暂停

3. 程序调试

程序执行到断点被暂停后，可以通过“调试”视图工具栏上的按钮，执行相应的调试操作，如运行、停止等。“调试”视图如图 2.35 所示。

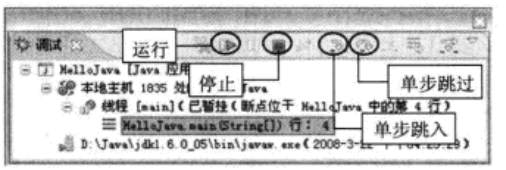


图 2.35　“调试”视图

（1）单步跳过。在“调试”视图的工具栏中单击按钮或按 F6 键，将执行单步跳过操作，即运行单独的一行程序代码，但是不进入调用方法的内部，然后跳到下一个可执行点并暂挂线程。

说明

不停地执行单步跳过操作，会每次一行执行程序代码，直到程序结束或等待用户操作。

（2）单步跳入。在“调试”视图的工具栏中单击按钮或按 F5 键，执行该操作将跳入调用方法或者对象的内部单步执行程序并暂挂线程。

2.4　本 章 小 结

本章主要介绍了 Eclipse 开发工具，包括它的配置与启动、菜单栏、工具栏，以及常用的“包资源管理器”和“控制台”视图；另外还介绍了 Eclipse 的使用方法，包括创建 Java 项目、创建 Java 类、编写程序代码和运行程序；最后还讲解了 Java 程序调试器的简单使用方法。

第 3 章

Java 语言基础

（视频讲解：58 分钟）

很多人认为学习 Java 语言之前必须要学习 C++语言，其实并非如此，这种错误认识是因为很多人在学习 Java 语言之前都学过 C++语言，事实上 Java 语言要比 C++语言更容易掌握。要掌握并熟练应用 Java 语言，就需要对 Java 语言的基础进行充分的了解。本章对 Java 语言基础进行了比较详细的介绍，对于初学者来说应该对本章的各个小节进行详细的阅读、思考，才能达到事半功倍的效果。

通过阅读本章，您可以：

- 了解 Java 程序的基本结构
- 了解 Java 中的标识符和关键字
- 了解 Java 语言中的基本数据类型
- 理解 Java 语言中的常量与变量
- 掌握 Java 语言中运算符的使用
- 理解 Java 语言中数据类型的转换
- 了解 Java 语言中的代码注释与编码规范

3.1 Java 程序的基本结构

视频讲解：光盘\TM\lx\3\Java 程序的基本结构.exe

要学习 Java 程序，首先应该了解程序的基本结构，这样有利于更进一步学习 Java 语言。一个 Java 程序的基本结构大体可以分为包、类、main()主方法、标识符、关键字、语句和注释等，如图 3.1 所示。

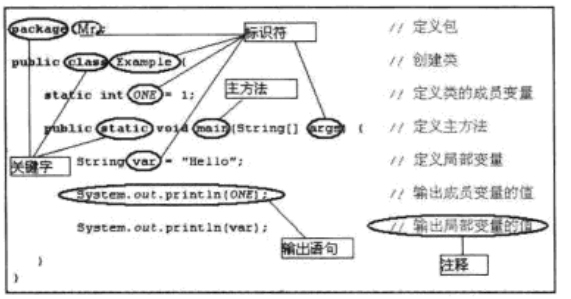


图 3.1 Java 程序的基本结构

第 1 条语句“package Mr;”定义了 Java 程序中类所在的包是 Mr，其中 Mr 是一个标识符，由程序员自己定义；package 是定义包的关键字，在 Java 程序中要定义包就必须使用 package 关键字。

说明 在 Java 语言中，标识符和关键字是区分大小写的，否则将导致程序无法正常执行。例如，如果将关键字 package 写成 Package，程序就会出错而无法正常执行，并且每条语句必须以分号结尾。

第 2 条语句“public class Example”是创建类的语句，其中 public 是 Java 程序的关键字，这里用于修饰类；class 是用于创建类的关键字，class 后的 Example 就是所创建类的名称，该类名由程序员自己定义。

第 3 条语句“static int ONE = 1;”定义了类的成员变量，其中 static 是 Java 程序的关键字，这里用于修饰成员变量；int 也是一个关键字，是 Java 程序中的整数数据类型，用于定义常量和变量，这里定义的是类的成员变量；ONE 是一个标识符，是该类的成员变量，其名称由程序员自己定义，该成员变量的数据类型是整数类型。

第 4 条语句“public static void main(String[] args)”是类的主方法，是 Java 应用程序的入口点，Java 程序是从该方法开始执行的，其中 main 是主方法的名称，程序员不可以更改；public 是 Java 程序的关键字，这里用于修饰主方法；static 是 Java 程序的关键字，这里用于修饰主方法；void 也是一个关键字，用于修饰方法，表示方法没有返回值；String 也是一个类，用于创建字符串对象，这里用于修饰主方法的形参，在 String 后跟上一对方括号“[”和“]”表示主方法的形参是一个字符串数组；args 是一个标识符，是主方法的形参数组，其数据类型是 String 类型。

说明 在 Java 语言中，主方法的写法是固定的，除了主方法的形参“String[] args”可以修改为“String args[]”以外，不可以改变第 4 条语句中的任何部分，否则 Java 程序都将无法运行。

第 5 条语句“String var = "Hello";”在主方法内定义了一个局部变量，其中 String 是一个类，用于

创建字符串对象，这里创建了一个局部变量；var 是局部变量的名称，是程序自己定义的一个标识符，"Hello"是局部变量 var 的值，是一个字符串常量；等号“=”是赋值运算符，用于将等号右边的字符串常量赋值给等号左边的变量 var，这样变量 var 的值就是 Hello 了。

说明　在 Java 程序中，除了字符串中常用的标点符号以外，代码中的所有标点符号必须是半角的或在英文输入法下输入（如逗号、分号、双引号等），否则程序会出错，这也是初学者容易犯的错误。

第 6 条语句“System.out.println(ONE);”是一个输出语句，可以在命令行或控制台输出信息，其中“System.out.println();”是输出语句的固定写法，其中 System 是一个系统类的名称，其第一个字母必须大写，out 是 System 类提供的一个标准输出流，println()是标准输出流 out 提供的方法，用于输出信息；println()方法内部的 ONE 是要输出的内容，这里的 ONE 是类的一个成员变量，其值是 1，所以执行该语句将输出 1。

最后一条语句“System.out.println(var);”是一个输出语句，其含义同第 6 条语句，这里的 var 是主方法内定义的一个局部变量，其值是"Hello"，所以执行该语句将输出 Hello。

3.2　标识符和关键字

视频讲解：光盘\TM\lx\3\标识符和关键字.exe

标识符是 Java 程序重要的组成部分，在程序中必须使用；关键字是 Java 中一些具有特殊意义的字符，不可以使用关键字对标识符进行命名。

3.2.1　标识符

标识符是 Java 程序中必须使用的，但也不是随便使用的，有一定的规则。本节将介绍什么是标识符和标识符的命名规则。

1．何为标识符

标识符可以简单地理解为一个名字，用来标识类名、变量名、方法名、数组名、文件名的有效字符序列。

【例 3.1】　定义变量 i 并赋值为 100。

```
int i = 100;
System.out.println(i);                    //在控制台输出信息
```

变量名 i 就是标识符，由程序员所起，但并不是随便取的，也有一定的规则。下面将介绍标识符的命名规则。

2．标识符的命名规则

标识符就是一个名字，对于所要表示的内容，用什么名字并不重要，但要通过标识符看出所写内

容即可。就好比人的名字，你叫什么名字并不重要，在别人叫这个名字时知道是你就可以了。标识符虽然可以任意取名，但是也要遵循以下规则：

- ☑ Java 语言的标识符由字母、数字、下划线和美元符号组成，第一个字符不能为数字。非法的标识符如 7word、5fox；合法的标识符如 tb_user、_u88。
- ☑ Java 语言使用 Unicode 标准字符集，最多可以识别 65535 个字符。因此，Java 语言中的字母可以是 Unicode 字符集中的任何字符，包括拉丁字母、汉字、日文和其他许多语言中的字符。合法的标识符如价格、User。
- ☑ 标识符不能是 Java 的关键字和保留字。非法的标识符如 this、goto。
- ☑ 在 Java 语言中标识符是区分大小写的，如果两个标识符的字母相同但是大小写不同，就是不同的标识符。good 和 Good 就是两个不同的标识符。

说明 在程序开发中，虽然可以使用汉字、日文等作为标识符，但为了避免出现错误，尽量不要使用，最好连下划线和数字也不要使用，而只用英文进行命名，且首字母用大写字母书写。

3.2.2 关键字

关键字是 Java 语言中已经被赋予特定意义的一些单词，不可以把这些字作为标识符来使用。Java 中的关键字如表 3.1 所示。

表 3.1 Java 关键字

int	public	this	finally	boolean	abstract
continue	float	long	short	throw	throws
return	break	for	static	new	interface
if	double	default	byte	do	case
strictfp	package	super	void	try	switch
else	catch	implements	private	final	class
extends	volatile	while	synchronized	instanceof	char
protecte	importd	transient	implements	dafaule	

说明 在命名标识符时，虽然 const 和 goto 不是 Java 的关键字，但也不可以使用，这两个词可能会在以后的升级版本中得以使用。

3.3 基本数据类型

视频讲解：光盘\TM\lx\3\基本数据类型.exe

在 Java 中有 8 种基本数据类型，其中 6 种是数值类型，另外两种分别是字符类型和布尔类型，而

6 种数值类型中有 4 种是整数类型，另外两种是浮点类型，如图 3.2 所示。

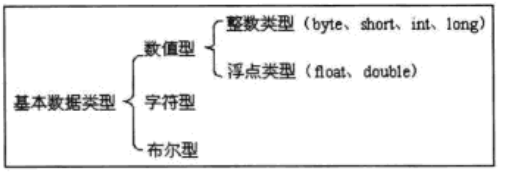


图 3.2　Java 基本数据类型

3.3.1　整数类型

整数类型用来存储整数数值，即没有小数部分的数值，可以是正数、负数，也可以是零。根据所占内存的大小不同，可以分为 byte、short、int 和 long 4 种类型。它们所占的内存和取值范围如表 3.2 所示。

表 3.2　整数类型

数 据 类 型	内存空间（8 位等于 1 字节）	取 值 范 围
byte	8 位（1 个字节）	-128～127
short	16 位（2 个字节）	-32768～32767
int	32 位（4 个字节）	-2147483648～2147483647
long	64 位（8 个字节）	-9223372036854775808～9223372036854775807

1．byte 型

使用 byte 关键字来定义 byte 型变量，可以一次定义多个变量并对其进行赋值，也可以不进行赋值。byte 型是整型中所分配的内存空间最少的，只分配 1 个字节；取值范围也是最小的，只在-128～127 之间，在使用时一定要注意，以免数据溢出产生错误。

【例 3.2】　定义 byte 型变量。

```
byte x = 48,y = -108,z;                    //定义 byte 型变量 x、y、z，并赋初值给 x、y
```

2．short 型

short 型即短整型，使用 short 关键字来定义 short 型变量，可以一次定义多个变量并对其进行赋值，也可以不进行赋值。系统给 short 型分配 2 个字节的内存，取值范围也比 byte 型大了很多，在-32768～32767 之间，虽然取值范围变大，但还是要注意数据溢出。

3．int 型

int 型即整型，使用 int 关键字来定义 int 型变量，可以一次定义多个变量并对其进行赋值，也可以不进行赋值。int 型变量取值范围很大，在-2147483648～2147483647 之间，足够一般情况下使用，所以是整型变量中应用最广泛的。

【例 3.3】　定义 int 型变量。

```
int x = 450,y = -462,z;                    //定义 int 型变量 x、y、z，并赋初值给 x、y
```

4．long 型

long 型即长整型，使用 long 关键字来定义 long 型变量，可以一次定义多个变量并对其进行赋值，也可以不进行赋值。而在对 long 型变量赋值时结尾必须加上“L”或者“l”，否则将不被认为是 long 型。当数值过大，超出 int 型范围时就使用 long 型，系统分配给 long 型变量 8 个字节，取值范围则更大，在-9223372036854775808～9223372036854775807 之间。

【例 3.4】 定义 long 型变量。

```
long x = 4556824L,y = -462447716l,z;    //定义 long 型变量 x、y、z，并赋初值给 x、y
```

说明

在定义 long 型变量时最好在结尾处加“L”，因为“l”非常容易和数字“1”弄混。

【例 3.5】 在项目中创建类 Number，在主方法中创建不同的整数类型变量，并将这些变量相加，将结果输出。（**实例位置：光盘\TM\sl\3\1**）

```
public class Number {                                          //创建类
    public static void main(String[] args) {                   //主方法
        byte mybyte = 124;                                     //声明 byte 型变量并赋值
        short myshort = 32564;                                 //声明 short 型变量并赋值
        int myint = 45784612;                                  //声明 int 型变量并赋值
        long mylong = 46789451L;                               //声明 long 型变量并赋值
        long result = mybyte+myshort+myint+mylong;             //获得各数相加后的结果
        System.out.println("四种类型相加的结果为："+result);    //将以上变量相加的结果输出
    }
}
```

运行结果如图 3.3 所示。

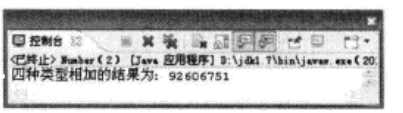


图 3.3 使用整数类型变量

上面的 4 种整数类型在 Java 程序中有 3 种表示形式，分别为十进制表示法、八进制表示法和十六进制表示法。

- ☑ 十进制表示法。十进制的表现形式大家都很熟悉，即逢十进一，每位上的数字最大是 9，如 120、0、-127 都是十进制数。
- ☑ 八进制表示法。八进制即逢八进一，每位上的数字最大是 7，且必须以 0 开头。例如，0123（转换成十进制数为 83）、-0123（转换成十进制数为-83）都是八进制数。
- ☑ 十六进制表示法。中国古代使用的就是十六进制数，所谓半斤八两就是如此，逢十六进一，每位上最大数字是 f（15），且必须以 0X 或 0x 开头。例如，0x25（转换成十进制数为 37）、0Xb01e（转换成十进制数为 45086）都是十六进制数。

3.3.2 浮点类型

浮点类型表示有小数部分的数字。在 Java 语言中，浮点类型分为单精度浮点类型（float）和双精

度浮点类型（double），它们具有不同的取值范围，如表 3.3 所示。

表 3.3　浮点类型

数据类型	内存空间（8 位等于 1 字节）	取值范围
float	32 位（4 个字节）	1.4E-45～3.4028235-E38
double	64 位（8 个字节）	4.9E-324～1.7976931348623157E-308

1．float 型

float 型即单精度浮点型，使用 float 关键字来定义 float 型变量，可以一次定义多个变量并对其进行赋值，也可以不进行赋值。在对 float 型进行赋值时在结尾必须添加“F”或者“f”，如果不加，系统自动将其定义为 double 型变量。float 型变量的取值范围在 1.4E-45 和 3.4028235E-38 之间。

【例 3.6】　定义 float 型变量。

```
float x = 12.521F,y = -5.264f,z;        //定义 float 型变量 x、y、z，并赋初值给 x、y
```

2．double 型

double 型即双精度浮点型，使用 double 关键字来定义 double 型变量，可以一次定义多个变量并对其进行赋值，也可以不进行赋值。在给 double 型赋值时，可以使用后缀“D”或“d”明确表明这是一个 double 类型数据，但加不加并没有硬性规定，可以加也可以不加。double 型变量的取值范围在 4.9E-324 和 1.7976931348623157E-308 之间。

【例 3.7】　定义 double 型变量。

```
double x = 12.521D,y = -5.264d,z=5.115,p;          //定义 double 型变量 x、y、z、p，并赋初值给 x、y、z
```

【例 3.8】　在项目中创建 SumNumber 类，在类的主方法中声明字符型变量，然后进行加法运算，并将运行结果输出。（**实例位置：光盘\TM\sl\3\2**）

```
public class SumNumber {                                          //创建类
    public static void main(String[] args) {                      //主方法
        float f1 = 13.23f;                                        //定义 float 型变量
        double d1 = 4562.12d;                                     //定义 double 型变量
        double d2 = 45678.1564;                                   //定义 double 型变量
        double result = f1+d1+d2;                                 //获得各数相加后的结果
        System.out.println("浮点型相加达到结果为："+result);       //将以上变量相加的结果输出
    }
}
```

运行结果如图 3.4 所示。

图 3.4　定义浮点类型

运行结果中输出了一个 16 位的数，这是因为 float 型变量是 8 位的，在转换成 16 位时自动在后面

补了一些数，这将在后面数据类型转换中详细介绍。

3.3.3 字符类型

char 型即字符类型，使用 char 关键字进行声明，用于存储单个字符，系统分配两个字节的内存空间。在定义字符型变量时，要用单引号括起来。例如，‘s’表示一个字符，且单引号中只能有一个字符，多了就不是字符类型了，而是字符串类型，需要用双引号进行声明。

【例 3.9】 声明 char 型变量。

```
char x = 'a';
```

由于字符 a 在 unicode 表中的排序位置是 97，因此允许将上面的语句写成：

```
char x = 97;
```

说明 与 C、C++语言一样，Java 语言也可以把字符作为整数对待。由于 Unicode 编码采用无符号编码，可以存储 65536 个字符（0x0000～0xffff），所以 Java 中的字符几乎可以处理所有国家的语言文字。若想得到一个 0～65536 之间的数所代表的 Unicode 表中相应位置上的字符，也必须使用 char 型显式转换。

【例 3.10】 在项目中创建类 Export，实现将 Unicode 表中某些位置上的字符以及一些字符在 Unicode 表中的位置在控制台上输出。

```
public class Export {                                          //定义类
    public static void main(String[] args) {                   //主方法
        int i = 'd';                                           //定义整型变量
        char c = 97;                                           //定义 char 型变量
        System.out.println("字符 d 的 Unicode 码是：" +i);
        System.out.println("Unicode 码 97 代表的字符是：" +c);
    }
}
```

运行结果如图 3.5 所示。

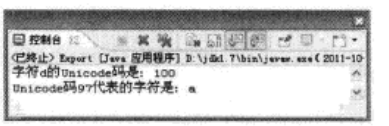


图 3.5 输出字符变量

在字符类型中有一种特殊的字符，以反斜线“\”开头，后跟一个或多个字符，具有特定的含义，不同于字符原有的意义，叫做转义字符。例如，“\n”就是一个转义字符，意思是“回车换行”。Java 中的转义字符如表 3.4 所示。

表 3.4 转义字符

转义字符	含义
\ddd	1～3 位八进制数据所表示的字符，如\456
\dxxxx	4 位十六进制所表示的字符，如\0052
\’	单引号字符

续表

转义字符	含　义
\\	反斜杠字符
\t	垂直制表符，将光标移到下一个制表符的位置
\r	回车
\n	换行
\b	退格
\f	换页

说明　转义字符本身也是字符，所以将转义字符赋值给字符变量时，与其他字符常量值一样需要使用单引号。

【例 3.11】　在项目中创建类 Gess，在类的主方法中使用转义字符定义字符型变量反斜杠和五角星，并在控制台上将转义字符输出。

```
public class Gess {                                          //定义类
    public static void main(String[] args) {                 //主方法
        char char1 = '\\';                                   //将转义字符“\\”赋值给变量 char1
        char char2 = '\u2605';                               //将转义字符“\u2605”赋值给变量 char2
        System.out.println("输出反斜杠："char1);              //输出结果\
        System.out.println("输出五角星："char2);              //输出结果★
    }
}
```

运行结果如图 3.6 所示。

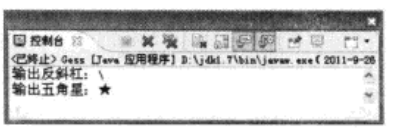


图 3.6　输出转义字符

3.3.4　布尔类型

布尔类型又称逻辑类型，只有 true 和 false 两个值，分别代表布尔逻辑中的“真”和“假”。使用 boolean 关键字声明布尔类型变量，通常被用在流程控制中作为判断条件。

【例 3.12】　声明 boolean 型变量。

```
boolean b1 = true,b2 = false,b3;          //定义布尔型变量 b1、b2、b3，并给 b1、b2 赋初值
```

3.4　变量与常量

视频讲解：光盘\TM\lx\3\变量与常量.exe

在程序执行过程中，其值不能改变的量称为常量，其值能被改变的量称为变量。变量与常量的声

明都必须使用合法的标识符，所有变量与常量只有在声明之后才能使用。

3.4.1 声明变量

在程序设计中，变量的使用是一个十分重要的环节。定义一个变量，就是要告诉编译器（compiler）这个变量属于哪一种数据类型，这样编译器才知道需要配置多少空间，以及能存放什么样的数据。变量都有一个变量名，变量名必须是合法的标识符，内存空间内的值就是变量值。在声明变量时可以不给予赋值，也可以直接赋给初值。

【例 3.13】 声明变量。

```
int age;                    //声明 int 型变量
char char1 = 'r';           //声明 char 型变量并赋值
```

以上程序代码的内存状态如图 3.7 所示。

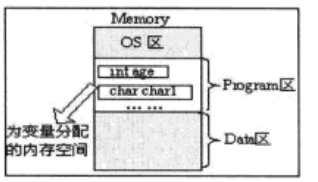


图 3.7 变量占用的内存空间

由图 3.7 可知，系统的内存可大略分为 3 个区域，即系统区（OS）、程序区（Program）和数据区（Data）。当程序执行时，程序代码会加载到内存中的程序区，数据暂时存储在数据区中。

变量虽然是由程序员所命名的，但是变量的命名并不是任意的，需要遵循一定的规则。Java 中变量的命名规则如下：

- ☑ 变量名必须是一个有效的标识符。变量名必须使用 Java 语言中合法的标识符，即以字母、数字和下划线组成，且首字母不能是数字，还不可以使用 Java 中的关键字。
- ☑ 变量名不能重复。如果两个变量具有同样的变量名，系统在对其进行使用时就不知道调用哪个变量，运行结果就会出现错误。
- ☑ 应选择有意义的单词作为变量名。在命名变量名时，最好能通过变量名看出变量的内容，这样既方便读者对程序的理解，增加可读性；又方便程序的维护，减轻程序维护人员的工作负担。

说明 在 Java 语言中允许使用汉字或其他语言文字作为变量名，如“int 年龄 =21”，在程序运行时并不出现什么错误，但建议读者尽量不要使用这些语言文字作为变量名。

【例 3.14】 在项目中创建类 Gess，声明整型变量 i 并赋给初值 10，之后再将 100 赋给 i，最后将 i 的值输出。

```
public class Gess {                                    //定义类
    public static void main(String[] args) {           //主方法
        int i = 10;                                    //定义变量 i，并赋给初值 10
        System.out.println("i 的初值是："+i);           //输出 i 的值
        i = 100;                                       //将 100 赋给 i
        System.out.println("i 现在的值是："+i);         //输出 i 的值
    }
}
```

运行结果如图 3.8 所示。

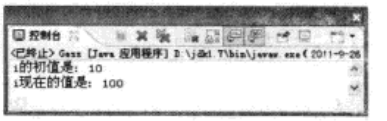


图 3.8　输出变量的值

3.4.2　声明常量

在程序运行过程中一直不会改变的量称为常量（constant），通常也被称为“final 变量”。常量在整个程序中只能被赋值一次。在为所有对象共享值时，常量是非常有用的。

在 Java 语言中声明一个常量，除了要指定数据类型外，还需要通过 final 关键字进行限定。声明常量的标准语法格式如下：

```
final 数据类型 常量名称[=值]
```

常量名通常使用大写字母，但这并不是必需的，很多 Java 程序员使用大写字母表示常量，常常是为了清楚地表明正在使用常量。

【例 3.15】　声明常量。

```
final double PI = 3.1415926F;          //声明 double 型常量 PI 并赋值
final boolean BOOL = true;             //声明 boolean 型常量 BOOL 并赋值
```

说明　当定义的常量如果属于“成员变量”，则必须在定义时就赋给初值，否则将会在编译时出现错误。“成员变量”将在 3.4.3 节中讲解。

【例 3.16】　在项目中创建类 Part，在类体中创建变量 age 与常量 PI。在主方法中分别给变量与常量赋值，通过输出信息可测试变量与常量的有效范围。（**实例位置：光盘\TM\sl\3\3**）

```
public class Part {                                          //新建类 Part
    static final double PI = 3.14;                           //声明常量 PI
    static int age = 23;                                     //声明 int 型变量 age 并进行赋值
    public static void main(String[] args) {                 //主方法
        final int number;                                    //声明 int 型常量 number
        number = 1235;                                       //对常量进行赋值
        age = 22;                                            //再次对变量进行赋值
        System.out.println("常量 PI 的值为：" + PI);          //将 PI 的值输出
        System.out.println("赋值后 number 的值为:" + number); //将 number 的值输出
        System.out.println("int 型变量 age 的值为：" + age);  //将 age 的值输出
    }
}
```

运行结果如图 3.9 所示。

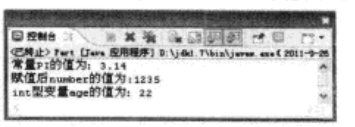


图 3.9　使用常量和变量

3.4.3　变量的有效范围

变量的有效范围是指程序代码能够访问该变量的区域，若超出变量所在区域访问变量则编译时会

出现错误。在程序中，一般会根据变量能够访问的区域将变量分为“成员变量”和“局部变量”。

1. 成员变量

在类体中定义的变量被称为成员变量，成员变量在整个类中都有效。类的成员变量又可分为静态变量和实例变量两种。

【例 3.17】 声明静态变量和实例变量。

```
class var {
    int x = 45;                                    //定义实例变量
    static int y = 90;                             //定义静态变量
}
```

其中，x 是实例变量，y 为静态变量（也称类变量）。如果成员变量的类型前面加上关键字 static，这样的成员变量称为静态变量。静态变量的有效范围可以跨类，甚至可达到整个应用程序之内。对于静态变量，除了能在定义它的类内存取，还能直接以“类名.静态变量”的方式在其他类内使用。

2. 局部变量

在类的方法体中定义的变量（方法内部定义，“{”与“}”之间的代码中声明的变量）称之为局部变量。局部变量只在当前代码块中有效，通俗地理解就是在其所定义的大括号内有效，出了这个大括号就没有效了，在其他类体中不能调用该变量。

局部变量的生命周期取决于方法，当方法被调用时，Java 虚拟机为方法中的局部变量分配内存空间，当该方法的调用结束后，则会释放方法中局部变量占用的内存空间，局部变量也随即销毁。

【例 3.18】 在项目中创建类 Val，分别定义名称相同的局部变量与成员变量，最后在控制台输出变量，观察输出的结果。

```
public class Val {                                 //新建类
    static int times = 3;                          //定义成员变量 times
    public static void main(String[] args) {       //主方法
        int times = 4;                             //定义局部变量 times
        System.out.println("times 的值为：" + times);   //将 times 的值输出
        System.out.println("times 的值为：" + Val.times);   //输出静态变量
    }
}
```

运行结果如图 3.10 所示。

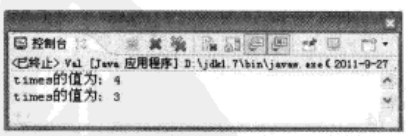


图 3.10 输出变量的值

说明 局部变量可与成员变量的名字相同，此时成员变量将被隐藏。即这个成员变量在此方法中暂时失效。如果想要调用成员变量，需要使用“类名.静态变量”调用。

成员变量和局部变量都有各自的有效范围，如图 3.11 所示。

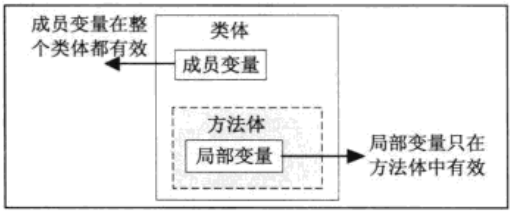


图 3.11　变量的有效范围

3.5 运　算　符

视频讲解：光盘\TM\lx\3\运算符.exe

运算符是一些特殊的符号，主要用于数学函数、一些类型的赋值语句和逻辑比较方面。Java 中提供了丰富的运算符，如赋值运算符、算术运算符、比较运算符等，本节将向读者介绍这些运算符。

3.5.1 赋值运算符

赋值运算符即“=”，是一个二元运算符（即对两个操作数进行处理），其功能是将右方操作数所含的值赋值给左方的操作数。语法格式如下：

```
变量类型 变量名 = 所赋的值;
```

左方必须是一个变量，而右边所赋的值可以是任何数值或表达式，包括变量（如 a、number）、常量（如 123、‘book’）或有效的表达式（如 45*12）。

【例 3.19】　使用赋值运算符为变量赋值。

```
int a = 10;                    //声明 int 型变量 a
int b = 5;                     //声明 int 型变量 b
int c = a+b+2;                 //将变量 a、b 和 2 进行运算后的结果赋值给 c
```

遵循赋值运算符的运算规则，可知系统将先计算 a+b+2 的值，结果为 17。然后将 17 赋值给变量 c，因此运算后“c=17”。

【例 3.20】　在项目中创建 Eval 类，在类的主方法中定义变量，使用赋值运算符为变量赋值。

```
int a, b, c;                   //声明 int 型变量 a、b、c
a = 15;                        //将 15 赋值给变量 a
c = b = a +4;                  //将 a+4 的值赋给变量 b、c
System.out.println("c 值为：" + c);   //将变量 c 的值输出
System.out.println("b 值为：" + b);   //将变量 b 的值输出
```

运行结果 b 和 c 的值都等于 19。

说明 在 Java 中可以把赋值运算符连在一起使用。例如：

```
x = y = z = 5;
```

在这个语句中，变量 x、y、z 得到同样的值 5。赋值运算符使用的顺序是先将 a+4 的值赋给 b，再将 b 的值赋给 c，从右向左运算。不过，在程序开发中不建议使用这种赋值语法。

3.5.2 算术运算符

Java 中的算术运算符主要有+（加号）、-（减号）、*（乘号）、/（除号）和%（求余），它们都是二元运算符。Java 中算术运算符的功能及使用方式如表 3.5 所示。

表 3.5 算术运算符

运算符	说明	实例	结果
+	加	12.45f+15	31.45
-	减	4.56-0.16	4.4
*	乘	5L*12.45f	62.25
/	除	7/2	3
%	取余数	1%2	2

其中“+”和“-”运算符还可以作为数据的正负符号，如+5、-7。

注意 除法运算时，要记住 0 不可以作除数。例如，int a = 5/0;系统会报出 ArithmeticException 的异常。

【例 3.21】 在项目中创建 Arith 类，在类的主方法中定义变量，使用算术运算符对变量进行计算，并在控制台将计算结果输出。（**实例位置：光盘\TM\sl\3\4**）

```
public class Arith {                                                    //创建类
    public static void main(String[] args) {                            //主方法
        float number1 = 45.56f;                                         //声明 float 型变量并赋值
        int number2 = 152;                                              //声明 int 型变量并赋值
        System.out.println("45.56f 和 152 的和为：" + number1 + number2);   //将变量相加之和输出
        System.out.println("45.56f 和 152 的差为：" + (number2 - number1));  //将变量相减之差输出
        System.out.println("45.56f 和 152 的积为：" + number1 * number2);   //将变量相乘的积输出
        System.out.println("45.56f 和 152 的商为：" + number1 / number2);   //将变量相除的商输出
    }
}
```

运行结果如图 3.12 所示。

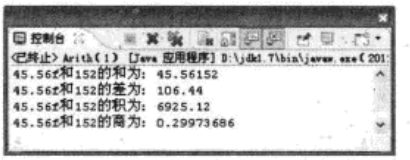


图 3.12　使用算术运算符运算

3.5.3　自增和自减运算符

自增、自减运算符是单目运算符，可以放在操作元之前，也可以放在操作元之后。操作元必须是一个整型或浮点型变量。放在操作元前面的自增、自减运算符，会先将变量的值加 1（减 1），然后再使该变量参与表达式的运算；放在操作元后面的自增、自减运算符，会先使变量参与表达式的运算，然后再将该变量加 1（减 1）。示例代码如下：

```
++a(--a)                    //表示在使用变量 a 之前，先使 a 的值加（减）1
a++(a--)                    //表示在使用变量 a 之后，使 a 的值加（减）1
```

【例 3.22】　粗略地分析++a 与 a++的作用都相当于 a = a+1，赋值 a = 4，计算 a++和++a 的值。

```
int a = 4;                  //声明变量 a，并赋值
b = ++a;                    //将 a 自加赋给 b
b = a++;                    //将 a 自加赋给 b
```

第 2 条语句是先将 a 自加，把 a 自加后的值赋给 b，自加后 a 的值是 5，b 的值也是 5；第 3 条语句是先将 a 的值赋给 b，然后再将 a 自加，自加后 a 的值是 5，而 b 的值是 4。

3.5.4　比较运算符

比较运算符属于二元运算符，用于程序中的变量和变量之间、变量和常量之间以及其他类型的信息之间的比较。比较运算符的运算结果是 boolean 型，当运算符对应的关系成立时，运算结果是 true，否则结果是 false。比较运算符通常用在条件语句中来作为判断的依据。比较运算符的种类和用法如表 3.6 所示。

表 3.6　比较运算符

运　算　符	作　　用	举　　例	操 作 数 据	结　　果
>	比较左方是否大于右方	'a'>'b'	整型、浮点型、字符型	false
<	比较左方是否小于右方	156 < 456	整型、浮点型、字符型	false
==	比较左方是否等于右方	'c'=='c'	基本数据类型、引用型	true
>=	比较左方是否大于等于右方	479>=426	整型、浮点型、字符型	true
<=	比较左方是否小于等于右方	12.45<=45.5	整型、浮点型、字符型	false
!=	比较左方是否不等于右方	'y'!= 't'	基本数据类型、引用型	true

【例 3.23】 在项目中创建 Compare 类，在类的主方法中创建整型变量，使用比较运算符对变量进行比较运算，将运算后的结果输出。（**实例位置：光盘\TM\sl\3\5**）

```
public class Compare {                                          //创建类
    public static void main(String[] args) {
        int number1 = 4;                                        //声明 int 型变量 number1
        int number2 = 5;                                        //声明 int 型变量 number2
        //依次将变量 number1 与变量 number2 的比较结果输出
        System.out.println("4>5 是否成立：" + (number1 > number2));
        System.out.println("4<5 是否成立：" + (number1 < number2));
        System.out.println("4==5 是否成立：" + (number1 == number2));
        System.out.println("4!=5 是否成立：" + (number1 != number2));
        System.out.println("4>=5 是否成立：" + (number1 >= number2));
        System.out.println("4<=5 是否成立：" + (number1 <= number2));
    }
}
```

运行结果如图 3.13 所示。

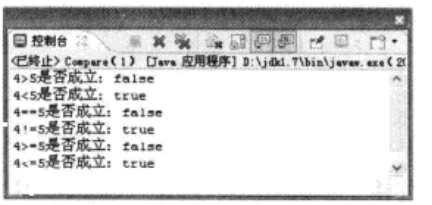


图 3.13 使用比较运算符比较

3.5.5 逻辑运算符

逻辑运算符包括&&（&）（逻辑与）、||（|）（逻辑或）和!（逻辑非），返回值为布尔类型的表达式，操作元也必须是 boolean 型数据。与比较运算符相比，逻辑运算符可以表示更加复杂的条件，如连接几个关系表达式进行判断。在逻辑运算符中，除了“!”是一元运算符之外，其余的都是二元运算符，其用法和含义如表 3.7 所示。

表 3.7 逻辑运算符

运 算 符	含 义	用 法	结 合 方 向
&&、&	逻辑与	op1&&op2	左到右
||、|	逻辑或	op1||op2	左到右
!	逻辑非	! op	右到左

用逻辑运算符进行逻辑判断时，不同的逻辑运算符与不同的操作元进行操作时，运行结果也不相同，运行结果如表 3.8 所示。

表 3.8 使用逻辑运算符进行逻辑运算

表达式 1	表达式 2	表达式 1&&表达式 2	表达式 1||表达式 2	! 表达式 1
true	true	true	true	false
true	false	false	true	false
false	false	false	false	true
false	true	false	true	true

说明　在 Java 中，逻辑运算符“&&”与“&”都表示“逻辑与”，那么它们之间的区别在哪里呢？从表 3.8 中可以看出，当两个表达式都为 true 时，逻辑与的结果才会是 true。使用逻辑运算符“&”会判断两个表达式；而逻辑运算符“&&”则是针对 boolean 类型进行判断，当第一个表达式为 false 时则不去判断第二个表达式，直接输出结果。使用“&&”可节省计算机判断的次数。通常将这种在逻辑表达式中从左端的表达式可推断出整个表达式的值称为“短路”，而那些始终执行逻辑运算符两边的表达式称为“非短路”。“&&”属于“短路”运算符，而“&”则属于“非短路”运算符。“||”和“|”也是如此。

【例 3.24】　在项目中创建 Calculation 类，在类的主方法中创建整型变量，使用逻辑运算符对变量进行运算，并将运算结果输出。

```
public class Calculation {                                          //创建类
    public static void main(String[] args) {
        int a = 2;                                                  //声明 int 型变量 a
        int b = 5;                                                  //声明 int 型变量 b
        boolean result1 = ((a > b) && (a != b));                    //声明布尔型变量
        boolean result2 = ((a > b) || (a != b));                    //声明布尔型变量
        System.out.println("(a > b) && (a != b)的值是："+result1);   //将变量 result1 输出
        System.out.println(" (a > b) || (a != b)的值是："+result2);   //将变量 result2 输出
    }
}
```

运行结果如图 3.14 所示。

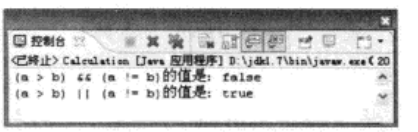


图 3.14　使用逻辑运算符判断

3.5.6　位运算符

位运算符用于处理整型和字符型的操作数，对其内存进行操作，数据在内存中以二进制的形式表示。例如，int 型变量 7 的二进制表示是 00000000 00000000 00000000 00000111，-8 的二进制表示是 11111111 11111111 11111111 11111000，最高位是符号位，0 表示正数，1 表示负数。Java 语言提供的位运算符如表 3.9 所示。

表 3.9　位运算符

运　算　符	含　　义	用　　法	运 算 分 类
~	按位取反	~op1	按位运算
&	按位与	op1 & op2	
|	按位或	op1 | op2	
^	按位异或	op1 ^ op2	
<<	左移	op1 << op2	移位运算符
>>	右移	op1 >> op2	
>>>	无符号右移	op1 >>> op2	

1. “按位与”运算

“按位与”运算的运算符为“&”，是双目运算符。其运算的法则是：如果两个操作数对应位都是 1，则结果位才是 1，否则为 0。如果两个操作数的精度不同，则结果的精度与精度高的操作数相同，如图 3.15 所示。

2. “按位或”运算

“按位或”运算的运算符为“|”，是双目运算符。其运算法则是：如果两个操作数对应位都是 0，则结果位才是 0，否则为 1。如果两个操作数的精度不同，则结果的精度与精度高的操作数相同，如图 3.16 所示。

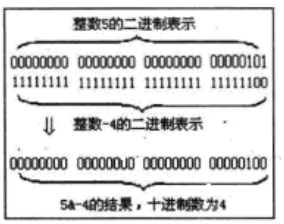


图 3.15　5&-4 的运算过程

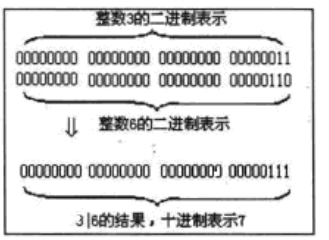


图 3.16　3|6 的运算过程

3. “按位非”运算

“按位非”运算也称“按位取反”运算，运算符为“～”，是单目运算符。其运算法则是：将操作数二进制中的 1 全部修改为 0，0 全部修改为 1，如图 3.17 所示。

4. “按位异或”运算

“按位异或”运算的运算符是“^”，是双目运算符。其运算法则是：当两个操作数的二进制表示相同（同时为 0 或同时为 1）时，结果为 0，否则为 1。若两个操作数的精度不同，则结果数的精度与精度高的操作数相同，如图 3.18 所示。

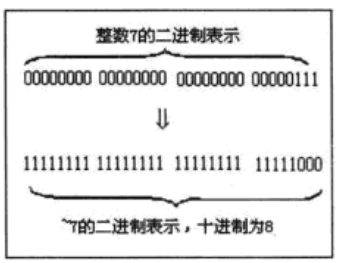


图 3.17　～7 的运算过程

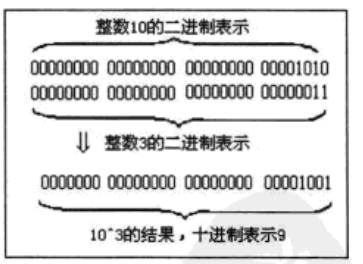


图 3.18　10^3 的运算过程

5. 移位运算符

Java 语言中的移位运算符有 3 种，其操作的数据类型只有 byte、short、char、int 和 long 5 种。

☑ 左移运算符“<<”。所谓左移运算符，就是将左边的操作数在内存中的二进制数据左移右边操作数指定的位数，左边移空的部分补 0。示例代码如下：

```
48 << 1;                    //将 48 的二进制数向左移 1 位
```

将 48 的二进制数向左移 1 位，移位后的结果是 96。

☑ 右移运算符“>>”。右移则复杂一些，当使用“>>”符号时，如果最高位是 0，左移空的位就填入 0；如果最高位是 1，右移空的位就填入 1，使用方法与左移类似。示例代码如下：

```
48 >> 1;                    //将 48 的二进制数向右移 1 位
```

将 48 的二进制数向右移 1 位，移位后的结果是 24。移位过程如图 3.19 所示。

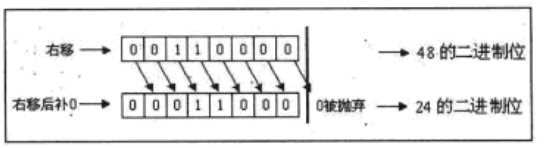


图 3.19　使用移位运算符移位

☑ 无符号右移运算符“>>>”。Java 还提供了无符号右移运算符“>>>”，不管最高位是 0 还是 1，左移空的高位都填入 0。

技巧　移位能让用户实现整数除以或乘以 2 的 n 次方的效果。例如，y<<2 与 y*4 的结果相同；y>>1 的结果与 y/2 的结果相同。总之，一个数左移 n 位，就是将这个数乘以 2 的 n 次方；一个数右移 n 位，就是将这个数除以 2 的 n 次方。

【例 3.25】　在项目中创建 Calculation 类，在类的主方法中创建整型变量，使用逻辑运算符对变量进行运算，将运算结果输出。

```
public class Calculation {                                  //声明类
    public static void main(String[] args) {                //主方法
        int i = 46;                                         //定义变量
        int j = 97;
        char c = 'a';
        System.out.println("46&97 的值是："+(i&j));          //输出 i 和 j 按位与的结果
        System.out.println("46^a 的值是："+(i^c));           //输出 i 和 j 按位异或的结果
        System.out.println("46>>1 的值是："+(i>>1));         //将 i 按位右移 1 位
    }
}
```

运行结果如图 3.20 所示。

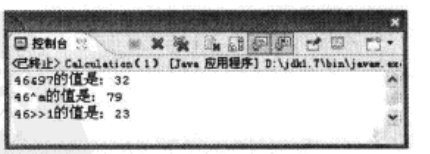


图 3.20　使用逻辑运算符判断

3.5.7　三元运算符

三元运算符是 Java 中唯一一个三目运算符，其操作元有 3 个，第一个是条件表达式，其余的是两个值，条件表达式成立时运算取第一个值，不成立时取第二个值。示例代码如下：

```
boolean b = 20 < 45 ? true : false;
```

三元运算符用于判断，等价的 if…else 语句如下：

```
boolean a;                    //声明 boolean 变量
if(20<45)                     //将 20<45 作为判断条件
    a = true;                 //条件成立将 true 赋值给 a
else
    a = false;                //条件不成立将 false 赋值给 a
```

当表达式“20<45”的运算结果返回真时，则 boolean 型变量 a 取值 true；当表达式“20<45”返回假时，则 boolean 型变量 a 取值 false，此例的结果是 true。

【例 3.26】 在项目中创建 Calculation 类，在类的主方法中创建整型变量，使用三目运算符，并将运算结果输出。

```
public class Calculation {                                  //声明类
    public static void main(String[] args) {                //主方法
        int i = 46;                                         //局部变量
        int j = 97;
        int z = i > j ? 100 : 200;                          //使用三目运算符
        System.out.println("i>j?100:200 的值是："+z);       //输出 z
    }
}
```

运行结果如图 3.21 所示。

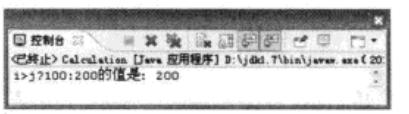


图 3.21 使用三目运算符

3.5.8 运算符优先级

Java 中的表达式就是使用运算符连接起来的符合 Java 规则的式子，运算符的优先级决定了表达式中运算执行的先后顺序。各运算符的优先级由高到低如图 3.22 所示。

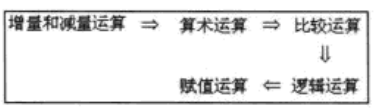


图 3.22 运算符的优先级

各运算符之间大致的优先级由图 3.22 列出，但如果两个符号属于同一运算符又怎么区分优先级呢？Java 中各符号的优先级如表 3.10 所示。

表 3.10 运算符的优先级

优 先 级	描 述	运 算 符
1	括号	( )
2	正负号	+、-
3	一元运算符	++、--、!
4	乘除	*、/、%
5	加减	+、-
6	移位运算	>>、>>>、<<
7	比较大小	<、>、>=、<=
8	比较是否相等	==、!=

续表

优　先　级	描　　述	运　算　符
9	按位与运算	&
10	按位异或运算	^
11	按位或运算	|
12	逻辑与运算	&&
13	逻辑或运算	||
14	三元运算符	? :
15	赋值运算符	=

说明　如果两个运算有相同的优先级，那么左边的表达式要比右边的表达式先被处理。在编写程序时尽量使用括号运算符来限定运算次序，以免产生错误的运算顺序。

3.5.9　范例 1：不用其他变量实现两变量互换

在对变量进行互换时，将创建一个临时变量来共同完成互换，临时变量的创建增加了系统资源的消耗。如果需要交换的是两个整数类型的变量，则可以使用更高效的方法，使用异或运算符进行互换，而不使用第三变量。运行结果如图 3.23 所示。（**实例位置：光盘\TM\sl\3\6**）

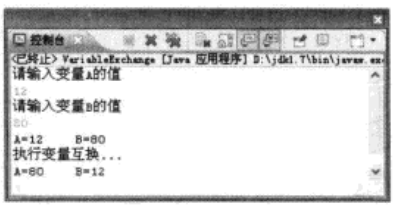


图 3.23　实现变量互换

在项目中创建 VariableExchange 类，在类中创建扫描器对象接收用户输入的两个变量值，然后通过位运算符中的异或运算符“^”实现两个变量的互换。代码如下：

```
import java.util.Scanner;
public class VariableExchange {                          //声明类
    public static void main(String[] args) {             //主方法
        Scanner scan = new Scanner(System.in);           //创建扫描器
        System.out.println("请输入变量 A 的值");
        long A = scan.nextLong();                        //接收第一个变量值
        System.out.println("请输入变量 B 的值");
        long B = scan.nextLong();                        //接收第二个变量值
        System.out.println("A=" + A + "\tB=" + B);
        System.out.println("执行变量互换...");
        A = A ^ B;                                       //执行变量互换
        B = B ^ A;
        A = A ^ B;
        System.out.println("A=" + A + "\tB=" + B);       //输出交换后的结果
    }
}
```

3.5.10 范例 2：判断数字的奇偶性

根据数字被 2 除的余数来判断一个数的奇偶性，如果余数是 0 该数是偶数，否则是奇数。本范例使用三元运算符进行判断。运行结果如图 3.24 所示。（**实例位置：光盘\TM\sl\3\7**）

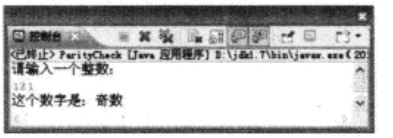


图 3.24 判断数字的奇偶性

在项目中创建 ParityCheck 类，在该类的主方法中创建扫描器对象接受用户输入的数字，通过三元运算符判断该数的奇偶性，并在控制台输出。代码如下：

```
import java.util.Scanner;
public class ParityCheck {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);                    //创建输入流扫描器
        System.out.println("请输入一个整数：");
        long number = scan.nextLong();                            //获取用户输入的整数
        String check = (number % 2 == 0) ? "这个数字是：偶数" : "这个数字是：奇数";
        System.out.println(check);
    }
}
```

3.6 类 型 转 换

类型转换是将变量从一种类型更改为另一种类型的过程。例如，可以将 String 类型数据 456 转换为一个 int 整型变量 456。Java 对数据类型的转换有严格的规定，数据从占用存储空间较小的类型转换为占用存储空间较大的数据类型时，则做自动类型转换（隐式类型转换）；反之则必须做强制类型转换（显式类型转换）。

3.6.1 自动类型转换

Java 中 8 种基本类型可以进行混合运算，不同类型的数据在运算过程中首先会自动转换为同一类型，再进行运算。数据类型根据占用存储空间的大小分为高低不同的级别，占用空间小的级别低，占用空间大的级别高，自动类型转换遵循低级到高级转换的规则。

【例 3.27】 定义 float 型变量 number1 并赋值 45，int 型变量 number2 并赋值 152，在控制台将 number1+number2 的值输出。

```
float number1 = 45f;                                    //声明 float 型变量并赋值
int number2 = 152;                                      //声明 int 型变量并赋值
System.out.println(number1 + number2);
```

系统先将 int 型变量转换成 float 变量之后相加，此时执行输出语句，在控制台上输出 197.0。

自动类型转换要遵循一定的规则，那么在运算时各类型间将怎么转换呢？各种情况下数据类型间转换的一般规则如表 3.11 所示。

表 3.11　隐式类型转换规则

操作数 1 的数据类型	操作数 2 的数据类型	转换后的数据类型
byte、short、char	int	int
byte、short、char、int、	long	long
byte、short、char、int、long	float	float
byte、short、char、int、long、float	double	double

【例 3.28】　在项目中创建 Conver 类，在类的主方法中创建不同数值型的变量进行运算，在控制台上将运行结果输出。（**实例位置：光盘\TM\sl\3\8**）

```
public class Conver{                                          //创建类
    public static void main(String[] args) {                  //主方法
        byte mybyte = 127;                                    //定义 byte 型变量 mybyte，并赋值 127
        int myint = 150;                                      //定义 int 型变量 myint，并赋值 150
        float myfloat = 452.12f;                              //定义 float 型变量 myfloat，并赋值
        char mychar = 10;                                     //定义 char 型变量 mychar，并赋值
        double mydouble = 45.46546;                           //定义 double 型变量，并赋值
        //将运算结果输出
        System.out.println("127 与 452.12 相加的和是：" + (mybyte + myfloat));
        System.out.println("127 与 150 相乘的积是：" + mybyte * myint);
        System.out.println("127 被 10 除的商是：" + mybyte / mychar);
        System.out.println("45.46546 和 10 相加的和是：" + (mydouble + mychar));
    }
}
```

运行结果如图 3.25 所示。

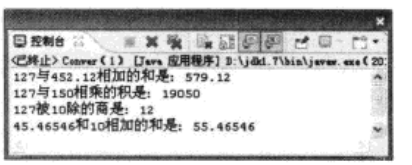


图 3.25　变量的自动类型转换

3.6.2　强制类型转换

当把高精度的变量的值赋给低精度的变量时，必须使用显式类型转换运算（又称强制类型转换）。语法格式如下：

```
(类型名)要转换的值
```

【例 3.29】　将不同的数据类型进行显式类型转换。

```
int a = (int)45.23;                    //此时输出 a 的值为 45
long y = (long)456.6F;                 //此时输出 y 的值为 456
int b = (int)'d';                      //此时输出 b 的值为 100
```

当把整数赋值给一个 byte、short、int、long 型变量时，不可超出这些变量的取值范围，否则就会发生数据溢出。示例代码如下：

```
Short s = 516;
byte b = (byte)s;
```

由于 byte 型变量的最大值是 127，516 已经超过了其取值范围，因此发生数据溢出。数据转换的过程如图 3.26 所示。

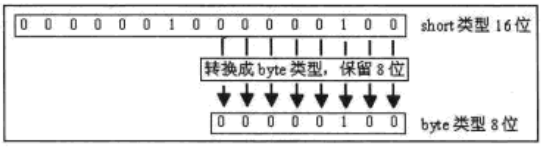


图 3.26　short 型转换成 byte 类型过程

此时就造成了数据丢失，所以在使用强制数据类型转换时，一定要加倍小心，不要超出变量的取值范围，否则就得不到想要的结果。

注意

boolean 型的值不能被转换为其他数据类型，反之亦然。

3.6.3　范例 3：类型转换实战

不管是自动类型转换还是强制类型转换都是程序设计中经常用到的，下面针对数据类型转换展开实战，运行结果如图 3.27 所示。（**实例位置：光盘\TM\sl\3\9**）

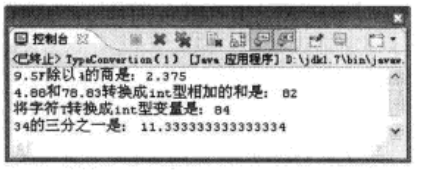


图 3.27　数据类型转换

在项目中创建 TypeConvertion 类，在类的主方法中定义不同数据类型的变量，然后在不同数据类型之间进行转换，并在控制台输出。代码如下：

```
public class TypeConvertion {                                    //声明类
    public static void main(String[]args){                       //主方法
        int intNum = 4;                                          //定义变量
        float floatNum = 9.5F;

        floatNum /= intNum;                                      //自动类型转换成 float 型
        System.out.println("9.5F 除以 4 的商是："+floatNum);

        double numX = 4.88;
        double numY = 78.83;
        int numZ = (int)numX+(int)numY;                          //将 double 型强制转换成 int 型
        System.out.println("4.88 和 78.83 转换成 int 型相加的和是：  "+numZ);

        char charVar = 'T';
        int intVar = (int)charVar;                               //将 char 型强制转换成 int 型
        System.out.println("将字符 T 转换成 int 型变量是："+intVar);
```

```
        int num1 = 34;
        double num2 = (double)num1/3;                    //将 int 型强制转换成 double 型
        System.out.println("34 的三分之一是：  "+num2);
    }
}
```

3.7　代码注释和编码规范

视频讲解：光盘\TM\lx\3\代码注释和编码规范.exe

在程序代码中适当地添加注释可以提高程序的可读性、可维护性；好的编码规范可以使程序更易阅读和理解。本节将向读者介绍 Java 中的几种代码注释以及应该注意的编码规范。

3.7.1　代码注释

通过在程序代码中添加注释可提高程序的可读性，注释中包含了程序的信息，可以帮助程序员更好地阅读和理解程序。在 Java 源程序文件的任意位置都可添加注释语句，注释中的文字 Java 编译器并不进行编译，所有代码中的注释文字并不对程序产生任何影响。Java 语言提供了 3 种添加注释的方法，分别为单行注释、多行注释和文档注释。

1．单行注释

“//”为单行注释标记，从符号“//”开始直到换行为止的所有内容均作为注释而被编译器忽略。语法格式如下：

```
//注释内容
```

声明 int 型变量 age，并用单行注释加以注释。示例代码如下：

```
int age ;            //定义 int 型变量用于保存年龄信息
```

2．多行注释

“/* */”为多行注释标记，符号“/*”与“*/”之间的所有内容均为注释内容。注释中的内容可以换行。语法格式如下：

```
/*
    注释内容 1
    注释内容 2
    ……
*/
```

在多行注释中可嵌套单行注释。示例代码如下：

```
/*
    程序名称：Hello word   //开发时间:2008-03-05
*/
```

但在多行注释中不可嵌套多行注释。非法代码如下：

```
/*
    程序名称：Hello word
    /*开发时间:2008-03-05
    作者：张先生
    */
*/
```

3. 文档注释

“/** */”为文档注释标记。符号“/**”与“*/”之间的内容均为文档注释内容。当文档注释出现在任何声明（如类的声明、类的成员变量的声明、类的成员方法的声明等）之前时，会被 Javadoc 文档工具读取作为 Javadoc 文档内容。文档注释的格式与多行注释的格式相同。对于初学者而言，文档注释并不是很重要，了解即可。语法格式如下：

```
/**
    *程序名称：Hello word
    *开发时间:2008-03-05
    *作者：张先生
*/
```

说明 一定要养成良好的编程风格。软件编码规范中提到“可读性第一，效率第二”，所以程序员必须要在程序中添加适量的注释来提高程序的可读性和可维护性。程序中注释要占程序代码总量的 20%～50%。

3.7.2 编码规范

在学习开发的过程中要养成良好的编码规范，因为规整的代码格式会给程序的开发与日后的维护提供很大方便。总结的编码规范如下：

- ☑ 每条语句要单独占一行。虽然 Java 语言中可以在一行当中写几条语句，但为了程序看起来更加规范，且便于维护，要养成每行只写一条语句的好的编码规范。
- ☑ 每条命令都要以分号结束。语句要以分号结尾，程序代码中的分号必须为英文状态下的，初学者经常会将“;”写成中文状态下的“；”，此时编译器会报出 illegal character（非法字符）这样的错误信息。
- ☑ 声明变量时要分行声明。即使是相同的数据类型也要将其放置在单独的一行上，这样有助于添加注释。
- ☑ Java 语句中多个空格看成一个。在 Java 代码中，关键字与关键字间如果有多个空格，这些空

格均被视作一个。例如：

```
public    static    void    main(String args[])  ⟹ 等价于  public static void main(String args[])
```

☑ 不要使用技术性很高、难懂、易混淆判断的语句。由于程序的开发与维护不能是同一个人，为了程序日后的维护方便，应尽量使用简单的技术完成程序需要的功能。

☑ 对于关键的方法要多加注释。多加注释会增加程序的可读性，有助于阅读者很快地了解代码结构。

3.8　经 典 范 例

3.8.1　经典范例 1：判断某一年是否是闰年

视频讲解：光盘\TM\lx\3\判断某一年是否是闰年.exe

为了弥补人类历法的年度天数和地球公转实际周期的时间差，设立了有 366 天的闰年，闰年的二月份有 29 天。闰年的判定规则是：如果该年能被 4 整除且不能被 100 整除或者能被 400 整除，则该年是闰年，否则不是。运行结果如图 3.28 所示。（**实例位置：光盘\TM\sl\3\10**）

图 3.28　判断是否是闰年

在项目中创建 LeapYear 类，在类的主方法中创建扫描器对象接收输入的年份，然后判断该年份是否是闰年，最后在控制台输出。代码如下：

```
import java.util.Scanner;                                          //声明包
public class LeapYear {                                            //声明类
    public static void main(String[] args) {                       //主方法
        Scanner scan = new Scanner(System.in);                     //扫描器
        System.out.println("请输入一个年份：");
        long year = scan.nextLong();                               //接收用户输入
        if (year % 4 == 0 && year % 100 != 0 || year % 400 == 0) {  //判断是否是闰年
            System.out.print(year + "年是闰年！");                  //输出是闰年
        } else {
            System.out.print(year + "年不是闰年！");                //输出不是闰年
        }
    }
}
```

3.8.2　经典范例 2：求球形的体积

视频讲解：光盘\TM\lx\3\求球形的体积.exe

在数学运算中，经常会计算球形的体积，这也是空间解析几何中必修的。通过计算机计算球形的

体积是很简单的，使用算术运算符即可实现。运行结果如图 3.29 所示。（**实例位置：光盘\TM\sl\3\11**）

在项目中创建 Volume 类，在类的主方法中创建扫描器对象接收输入的半径，然后根据球形体积公式计算球形的体积，并在控制台上输出。代码如下：

图 3.29　计算球形的体积

```
import java.util.Scanner;                                    //声明包
public class Volume {                                        //声明类
    public static void main(String[] args) {                 //主方法
        Scanner scan = new Scanner(System.in);               //扫描器
        System.out.println("请输入球形的半径：");
        double r = scan.nextDouble();                        //接收数据
        final double PI = 3.1415926;                         //定义常量 PI
        double volume = 4.0/3.0 * PI * r * r * r;            //求球形的体积
        System.out.println("球形的半径是：" + r);             //输出球形的半径
        System.out.println("圆周率是：" + PI);                //输出圆周率
        System.out.println("球形的体积是：" + volume);        //输出球形的体积
    }
}
```

3.9　本章小结

本章向读者介绍的是 Java 语言基础，其中需要读者重点掌握的是 Java 语言的基本数据类型、变量与常量以及运算符等三大知识点。初学者经常会将 String 类型认为是 Java 语言的基本数据类型，在此提醒读者 Java 语言的基本数据类型中并没有 String 类型。另外，对数据类型之间的转换也要有一定的了解。在使用变量时，需要读者注意的是变量的有效范围，否则在使用时会出现编译错误或浪费内存资源。此外，Java 中的各种运算符也是 Java 基础中的重点，正确使用这些运算符，才能得到预期的结果。

3.10　实战练习

1．编写 Java 程序，不使用乘号“*”，而只使用移位运算符计算出 21*16 的值。（**答案位置：光盘\TM\sl\3\12**）

2．编写 Java 程序，声明两个 int 型变量，运用三元运算符判断两个变量是否相等，若不相等，求出两个数中较大的。（**答案位置：光盘\TM\sl\3\13**）

3．编写 Java 程序，声明两个变量并赋值 38.9 和 27.2 作为矩形的长和宽，求出该矩形的面积。（**答案位置：光盘\TM\sl\3\14**）

第 4 章

流程控制

（视频讲解：31 分钟）

流程控制对于任何一门编程语言来说都是至关重要的，它提供了控制程序步骤的基本手段。如果没有流程控制语句，整个程序将按照线性的顺序来执行，不能根据用户的输入决定执行的序列。本章将向读者介绍 Java 语言中的流程控制语句。

通过阅读本章，您可以：

- 理解 Java 语言中复合语句的使用方法
- 掌握 if 条件语句的使用方法
- 了解 if 语句与 switch 语句间的区别
- 掌握 while 循环语句的使用方法
- 掌握 do…while 循环语句的使用方法
- 了解 while 语句与 do…while 语句的区别
- 掌握 for 语句的使用方法
- 了解跳转语句的使用

PDG

4.1 复合语句

视频讲解：光盘\TM\lx\4\复合语句.exe

同 C 语言或其他语言相同，Java 语言的复合语句是以整个块区为单位的语句，所以又称块语句。复合语句由开括号“{”开始，闭括号“}”结束。

在前面的学习中已经接触到了这种复合语句。例如在定义一个类或方法时，类体就是以“{ }”作为开始与结束的标记，方法体同样也是以“{ }”作为标记。对于复合语句中的每个语句都是从上到下地被执行。复合语句以整个块为单位，可以用在任何一个单独语句可以用到的地方，并且在复合语句中还可以嵌套复合语句。

【例 4.1】 在项目中创建 Compound 类，在主方法中定义复合语句块，其中包含另一复合语句块。**（实例位置：光盘\TM\sl\4\1）**

```
public class Compound {
    public static void main(String args[]) {
        {
            int y = 40;
            System.out.println("输出 y 的值："+y);
            int z = 245;
            boolean b;
            {
                b = y > z;
                System.out.println("y>z 成立吗："+b);   //复合语句
            }
        }
        String word = "hello java";
        System.out.println("输出字符串："+word);
    }
}
```

复合语句

运行结果如图 4.1 所示。

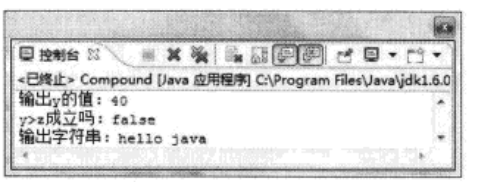


图 4.1 使用复合语句

在使用复合语句时要注意，复合语句为局部变量创建了一个作用域，该作用域为程序的一部分，在该作用域中某个变量被创建并能够被使用。如果在某个变量的作用域外使用该变量，则会发生错误，例如在本实例中如果在复合语句外使用变量 x、y、b 将会出现错误。而变量 x 可在整个方法体中使用。

4.2 条件语句

视频讲解：光盘\TM\lx\4\条件语句.exe

条件语句可根据不同的条件执行不同的语句。条件语句包括 if 条件语句与 switch 多分支语句。本节将向读者介绍条件语句的用法。

4.2.1 if 条件语句

if 条件语句是一个重要的编程语句，它用于告诉程序在某个条件成立的情况下执行某段程序，而在另一种情况下执行另外的语句。

使用 if 条件语句，可选择是否要执行紧跟在条件之后的那个语句。关键字 if 之后是作为条件的“布尔表达式”，如果该表达式返回的结果为 true，则执行其后的语句；若为 false，则不执行 if 条件之后的语句。if 条件语句可分为简单的 if 条件语句、if…else 语句和 if…else if 多分支语句。

1. 简单的 if 条件语句

语法格式如下：

```
if(布尔表达式) {
    语句序列
}
```

- ☑ 布尔表达式：必要参数，表示它最后返回的结果必须是一个布尔值。它可以是一个单纯的布尔变量或常量，或者使用关系或布尔运算符的表达式。
- ☑ 语句序列：可选参数。可以是一条或多条语句，当表达式的值为 true 时执行这些语句。如语句序列中仅有一条语句，则可以省略条件语句中的大括号。

【例 4.2】 语句序列中只有一条语句。

```
int a = 100;
    if(a == 100)
    System.out.print("a 的值是 100");
```

说明 虽然 if 和 else 语句后面的复合语句块只有一条语句，省略“{}”并无语法错误，但为了增强程序的可读性最好不要省略。

【例 4.3】 省略了 if 条件表达式中的语句序列。

```
boolean b = false;
if(b);
```

```
boolean b = false;
if(b){}
```

简单的 if 条件语句的执行过程如图 4.2 所示。

【例 4.4】 在项目中创建 Getif 类，在主方法中定义整型变量。使用条件语句判断两个变量的大小来决定输出结果。（**实例位置：光盘\TM\sl\4\2**）

```
public class Getif {                                    //创建类
    public static void main(String args[]) {            //主方法
        int x = 45;                                     //声明 int 型变量 x，并赋给初值
        int y = 12;                                     //声明 int 型变量 y，并赋给初值
        if (x > y) {                                    //判断 x 是否大于 y
            System.out.println("变量 x 大于变量 y");     //如果条件成立，输出的信息
        }
        if (x < y) {                                    //判断 x 是否小于 y
            System.out.println("变量 x 小于变量 y");     //如果条件成立，输出的信息
        }
    }
}
```

运行结果如图 4.3 所示。

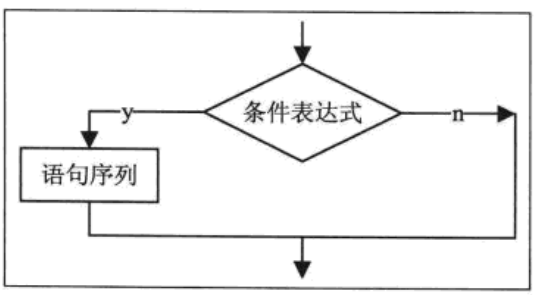


图 4.2 if 条件语句的执行过程

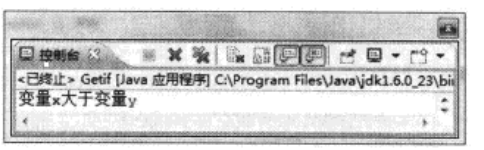


图 4.3 使用 if 语句判断大小

2. if…else 语句

if…else 语句是条件语句中最常用的一种形式，它会针对某种条件有选择地做出处理。通常表现为“如果满足某种条件，就进行某种处理，否则就进行另一种处理”。语法格式如下：

```
if(表达式) {
    若干语句
}
else {
    若干语句
}
```

if 后面()内的表达式的值必须是 boolean 型的。如果表达式的值为 true，则执行紧跟 if 语句的复合语句；如果表达式的值为 false，则执行 else 后面的复合语句。if…else 语句的执行过程如图 4.4 所示。

同简单的 if 条件语句一样，如果 if…else 语句的语句序列中只有一条语句（不包括注释），则可以省略该语句序列外面的大括号。有时为了编程的需要，else 或 if 后面的大括号里可以没有语句。

【例 4.5】　在项目中创建 Getifelse 类，在主方法中定义变量，并通过使用 if…else 语句判断变量的值来决定输出结果。（**实例位置：光盘\TM\sl\4\3**）

```
public class Getifelse {
    public static void main(String args[]) {                    //主方法
        int math = 95;                                          //声明 int 型局部变量，并赋给初值 95
        int english = 56;                                       //声明 int 型局部变量，并赋给初值 56
        if (math > 60) {                                        //使用 if 语句判断 math 是否大于 60
            System.out.println("数学及格了");                    //条件成立时输出信息
        } else {
            System.out.println("数学没有及格");                  //条件不成立输出的信息
        }
        if (english > 60) {                                     //判断英语成绩是否大于 60
            System.out.println("英语及格了");                    //条件成立输出的信息
        } else {
            System.out.println("英语没有及格");                  //条件不成立输出的信息
        }
    }
}
```

运行结果如图 4.5 所示。

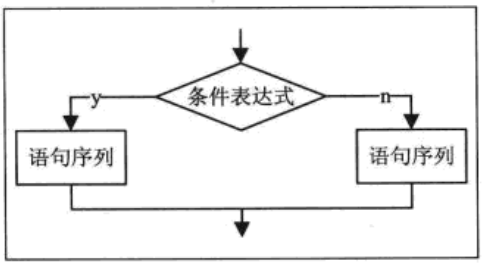


图 4.4　if…else 语句的执行过程

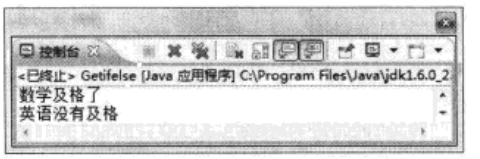


图 4.5　判断是否及格

3．if…else if 多分支语句

if…else if 多分支语句用于针对某一事件的多种情况进行处理。通常表现为“如果满足某种条件，就进行某种处理，否则，如果满足另一种则执行另一种处理”。语法格式如下：

```
if(条件表达式 1){
语句序列 1
}
else if(条件表达式 2) {
  语句序列 2
}
…
else if(条件表达式 n) {
    语句序列 n
}
```

☑　条件表达式 1～条件表达式 n：必要参数。可以由多个表达式组成，但最后返回的结果一定要

为 boolean 类型。

☑ 语句序列：可以是一条或多条语句，当条件表达式 1 的值为 true 时，执行语句序列 1；当条件表达式 2 的值为 true 时，执行语句序列 2，依此类推。当省略任意一组语句序列时，可以保留其外面的大括号，也可以将大括号替换为“;”。

if…else if 多分支语句的执行过程如图 4.6 所示。

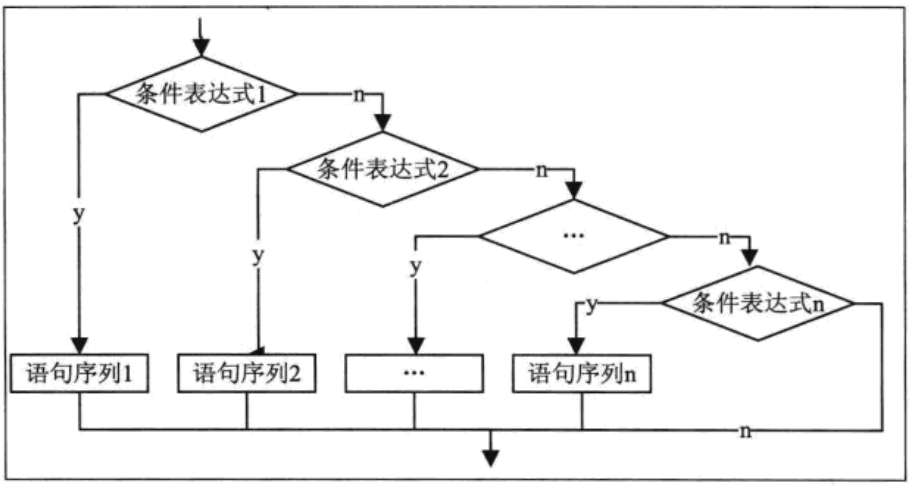


图 4.6　if…else if 多分支语句执行过程

【例 4.6】　在项目中创建 GetTerm 类，在主方法中定义变量 x，使用 if…else if 多分支语句通过判断 x 的值决定输出结果。（**实例位置：光盘\TM\sl\4\4**）

```
public class GetTerm {                                  //创建主类
    public static void main(String args[]) {            //主方法
        int x = 20;                                     //声明 int 型局部变量
        if (x > 30) {                                   //判断变量 x 是否大于 30
            System.out.println("a 的值大于 30");         //条件成立的输出信息
        } else if (x > 10) {                            //判断变量 x 是否大于 10
            System.out.println("a 的值大于 10，但小于 30"); //条件成立的输出信息
        } else if (x > 0) {                             //判断变量 x 是否大于 0
            System.out.println("a 的值大于 0，但小于 10");  //条件成立的输出信息
        } else {                                        //当以上条件都不成立时，执行的语句块
            System.out.println("a 的值小于 0");          //输出信息
        }
    }
}
```

运行结果如图 4.7 所示。

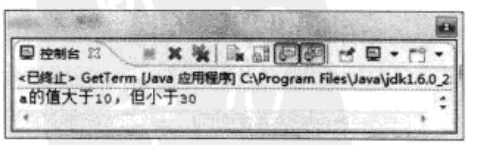


图 4.7　使用 if…else if 语句

在本实例中，由于变量 x 为 20，条件 x>30 为假，程序向下执行判断下面的条件；条件 x>10 为真，所以执行条件 x>10 后面的程序块中语句。输出“a 的值大于 10，但小于 30”，然后退出 if 语句。

注意

if语句只执行条件为真的命令语句，其他语句都不会执行。

4.2.2 switch 多分支语句

在编程中一个常见的问题就是检测一个变量是否符合某个条件，如果不匹配，再用另一个值来检测它，依此类推。当然，这种问题使用 if 条件语句也可以完成。

【例 4.7】 使用 if 语句检测变量是否符合某个条件。

```
if(grade == "A"){
      System.out.println("真棒");
      }
if(grade == "b"){
      System.out.println("做的不错");
      }
```

这个程序显得比较笨重，程序员需要测试不同的值来给出输出语句。在 Java 语言中，可以用 switch 语句将动作组织起来，就能以一个较简单明了的方式来实现“多选一”的选择。语法格式如下：

```
switch(表达式)
{
case 常量值 1;
      语句块 1
      [break;]
...
case 常量值 n;
      语句块 n
      [break;]
default;
      语句块 n+1;
      [break;]
}
```

switch 语句中表达式的值必须是整型或字符型，常量值 1～常量值 n 必须也是整型或字符型。switch 语句首先计算表达式的值，如果表达式的值和某个 case 后面的变量值相同，则执行该 case 语句后的若干个语句直到遇到 break 语句为止。此时如果该 case 语句中没有 break 语句，将继续执行后面 case 中的若干个语句，直到遇到 break 语句为止。若没有一个常量的值与表达式的值相同，则执行 default 后面的语句。default 语句为可选的，如果它不存在，而且 switch 语句中表达式的值不与任何 case 的常量值相同，switch 则不做任何处理。

注意

同一个 switch 语句，case 的常量值必须互不相同。

switch 语句的执行过程如图 4.8 所示。

【例 4.8】 在项目中创建 GetSwitch 类，在主方法中应用 switch 语句将周一～周三的英文单词打印出来。（**实例位置：光盘\TM\sl\4\5**）

```
public class GetSwitch {                                    //创建类
public static void main(String args[]) {                    //主方法
        System.out.println("今天是星期几：");
        int week = 2;                                       //定义 int 型变量 week
        switch (week) {                                     //指定 switch 语句的表达式为变量 week
        case 1:                                             //定义 case 语句中的常量为 1
            System.out.println("Monday");                   //输出信息
            break;
        case 2:                                             //定义 case 语句中的常量为 2
            System.out.println("Tuesday");
            break;
        case 3:                                             //定义 case 语句中的常量为 3
            System.out.println("Wednesday");
            break;
        default:                                            //default 语句
            System.out.println("Sorry,I don't Know");
        }
    }
}
```

运行结果如图 4.9 所示。

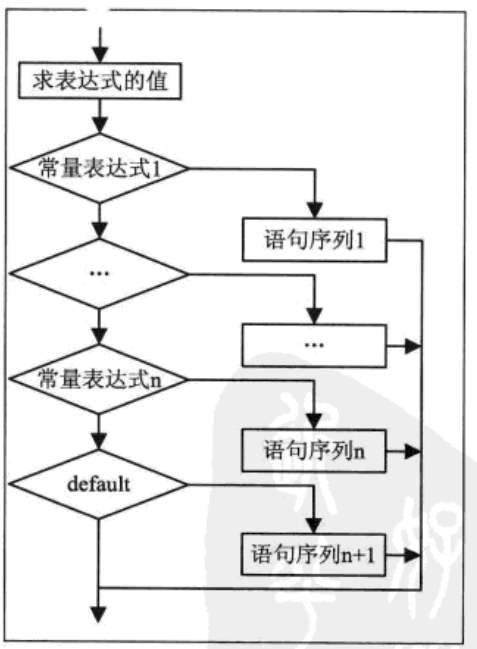


图 4.8 switch 语句的执行过程

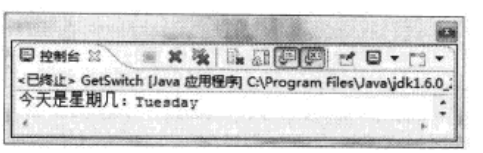


图 4.9 使用 switch 语句

注意　在 switch 语句中，case 语句后常量表达式的值可以为整数，但绝不可以是实数。例如下面的代码就是不合法的。

```
case 1.1;
```

常量表达式的值可以是字符，但一定不可以是字符串。例如下面的代码也是非法的。

```
case "ok";
```

4.2.3　范例 1：验证登录信息的合法性

大多系统登录模块都会接收用户通过键盘输入的登录信息，这些登录信息将会被登录模块验证，如果使用的是指定的用户名与密码，则允许程序登录，否则将用户拒之门外。本范例通过 if…else 语句进行多条件判断来实现登录信息的验证。运行结果如图 4.10 所示。（**实例位置：光盘\TM\sl\4\6**）

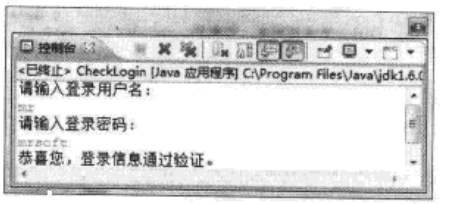


图 4.10　验证登录信息的合法性

在项目中创建 CheckLogin 类，在该类的主方法中接收用户输入的登录用户名与登录密码，然后通过 if 条件语句分别判断用户名与密码，并输出登录验证结果。代码如下：

```
import java.util.Scanner;
public class CheckLogin {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);                    //创建扫描器
        System.out.println("请输入登录用户名：");
        String username = scan.nextLine();                        //接收用户输入的登录名
        System.out.println("请输入登录密码：");
        String password = scan.nextLine();                        //接收用户输入的登录密码
        if (!username.equals("mr")) {                             //判断用户名合法性
            System.out.println("用户名非法。");
        } else if (!password.equals("mrsoft")) {                  //判断密码合法性
            System.out.println("登录密码错误。");
        } else {                                                  //通过两个条件判断则默认通过登录验证
            System.out.println("恭喜您，登录信息通过验证。");
        }
    }
}
```

4.2.4　范例 2：为新员工分配部门

本范例的关键技术在于 switch 多分支语句的使用，该语句只支持对常量的判断，而常量又只能是

Java 的基本数据类型，采用字符串的哈希码进行判断，也就是把 String 类的 hashCode()方法返回值作为 switch 语法的表达式，case 关键字之后跟随的是各种字符串常量的哈希码整数值。运行结果如图 4.11 所示。（**实例位置：光盘\TM\sl\4\7**）

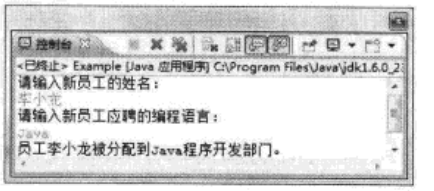


图 4.11　为新员工分配部门

在项目中创建 Example 类，在该类的主方法中创建标准输入流的扫描器，通过扫描器获取人事部门输入的姓名与应聘编程语言，然后根据每个语言对应的哈希码来判断分配部门。代码如下：

```
import java.util.Scanner;
public class Example {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        System.out.println("请输入新员工的姓名：");
        String name = scan.nextLine();                          //接收员工名称
        System.out.println("请输入新员工应聘的编程语言：");
        String language = scan.nextLine();                      //接收员工应聘的编程语言

        switch (language.hashCode()) {                          //根据编程语言确定员工分配的部门
            case 3254818:                                       //java 的哈希码
            case 2301506:                                       //Java 的哈希码
            case 2269730:                                       //JAVA 的哈希码
                System.out.println("员工"+name+"被分配到 Java 程序开发部门。");
                break;
            case 3104:                                          //c#的哈希码
            case 2112:                                          //C#的哈希码
                System.out.println("员工"+name+"被分配到 C#项目维护组。");
                break;
            case -709190099:                                    //Asp.net 的哈希码
            case 955463181:                                     //Asp.net 的哈希码
            case 9745901:                                       //Aap.net 的哈希码
                System.out.println("员工"+name+"被分配到 Asp.net 程序测试部门。");
                break;
            default:
                System.out.println("本公司不需要" + language + "语言的程序开发人员。");
        }
    }
}
```

在 switch 语法中每个 case 关键字可以作为一个条件分支，但是对于多个条件采取相同业务处理的情况，可以把多个 case 分支关联在一起，省略它们之间的 break 语句，而在最后一个相同的 case 分支中实现业务处理并执行 break 语句。

4.3 循环语句

视频讲解：光盘\TM\lx\4\循环语句.exe

循环语句就是在满足一定条件的情况下反复执行某一个操作。在 Java 中提供了 3 种常用的循环语句，分别是 while 循环语句、do…while 循环语句和 for 循环语句。下面分别对这 3 种循环语句进行介绍。

4.3.1 while 循环语句

while 循环语句也称为条件判断语句，它的循环方式为利用一个条件来控制是否要继续反复执行这个语句。语法格式如下：

```
while(条件表达式)
{
    执行语句
}
```

当条件表达式的返回值为真时，则执行“{}”中的语句，当执行完“{}”中的语句后，重新判断条件表达式的返回值，直到表达式返回的结果为假时，退出循环。while 循环语句的执行过程如图 4.12 所示。

【例 4.9】 在项目中创建 GetSum 类，在主方法中通过 while 循环将整数 1～10 相加。（**实例位置：光盘\TM\sl\4\8**）

```
public class GetSum{                                    //创建类
public static void main(String args[]) {                //主方法
        int x = 1;                                      //定义 int 型变量 x，并赋给初值
        int sum = 0;                                    //定义变量用于保存相加后的结果
        while (x <= 10) {
            sum = sum + x;                              //while 循环语句，当变量满足条件表达式时执行循环体语句
            x++;
        }
        System.out.println("1 到 10 的和是：sum = " + sum);      //将变量 sum 输出
    }
}
```

运行结果如图 4.13 所示。

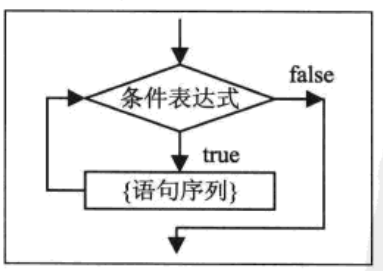


图 4.12 while 语句的执行过程

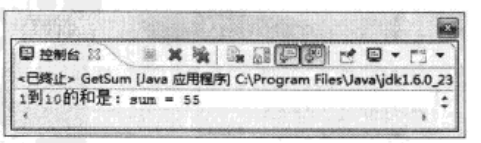


图 4.13 例 4.9 的运行结果

注意 初学者经常犯的一个错误就是在 while 表达式的括号后加“;”。例如：

```
while(x = = 5);
System.out.println("x 的值为 5");
```

这时程序会认为要执行一条空语句，而进入无限循环，Java 编译器又不会报错，可能会浪费很多时间去调试，应注意这个问题。

4.3.2 do…while 循环语句

do…while 循环语句与 while 循环语句类似，它们之间的区别是 while 循环语句为先判断条件是否成立再执行循环体，而 do…while 循环语句则先执行一次循环后，再判断条件是否成立。也就是说 do…while 循环语句中大括号中的程序段至少要被执行一次。语法格式如下：

```
do
{
    执行语句
}
    while(条件表达式);
```

与 while 语句的一个明显区别是 do…while 循环语句在结尾处多了一个分号（;）。根据 do…while 循环语句的语法特点总结出 do…while 循环语句的执行过程如图 4.14 所示。

【例 4.10】 在项目中创建 Cycle 类，在主方法中编写代码，通过本实例可看出 while 循环语句与 do…while 循环语句的区别。（**实例位置：光盘\TM\sl\4\9**）

```
public class Cycle {
    public static void main(String args[]){
        int a = 100;                                    //声明 int 型变量 a 并赋初值 100
        while(a == 60)                                  //指定进入循环体的条件
        {
            System.out.println("ok！ a==60");           //while 语句循环体
            a--;
        }
        int b = 100;                                    //声明 int 型变量 b 并赋初值 100
        do
        {
            System.out.println("ok！ b==100");          //do…while 语句循环体
            b--;
        }
        while(b == 60);                                 //指定循环结束条件
    }
}
```

运行结果如图 4.15 所示。

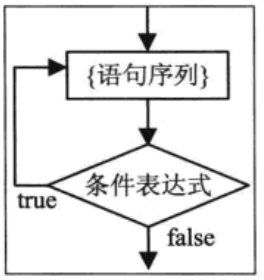


图 4.14　do…while 循环语句的执行过程

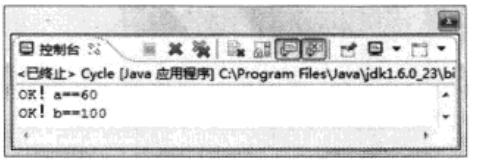


图 4.15　使用 do…while 循环语句

4.3.3　for 循环语句

for 循环语句是 Java 程序设计中最有用的循环语句之一。一个 for 循环可以用来重复执行某条语句，直到某个条件得到满足。在 Java 5 以后新增了 foreach 语法，本节将对这两种 for 循环形式进行详细的介绍。

1．for 语句

语法格式如下：

```
for(表达式 1;表达式 2;表达式 3)
{
    语句序列
}
```

☑　表达式 1：初始化表达式，负责完成变量的初始化。

☑　表达式 2：循环条件表达式，值为 boolean 型的表达式，指定循环条件。

☑　表达式 3：循环后操作表达式，负责修整变量，改变循环条件。

在执行 for 循环时，首先执行表达式 1，完成某一变量的初始化工作；下一步判断表达式 2 的值，若表达式 2 的值为 true，则进入循环体；在执行完循环体后紧接着计算表达式 3，这部分通常是增加或减少循环控制变量的一个表达式。这样一轮循环就结束了。第二轮循环从计算表达式 2 开始，若表达式 2 返回 true，则继续循环，否则跳出整个 for 语句。for 循环语句的执行过程如图 4.16 所示。

【例 4.11】　在项目中创建 Circulate 类，在主方法中使用 for 循环语句来计算 2～100 之间所有偶数之和。

```
public class Circulate{                                  //创建 Circulate 类
    public static void main(String args[]){              //主方法
        int sum = 0;                                     //声明变量，用于保存各数相加后的结果
        for(int i = 2;i<=100;i+=2){
            sum = sum+i;                                 //指定循环条件及循环体
        }
```

```
        System.out.println("2 到 100 之间的所有偶数之和为："+sum);        //将相加后的结果输出
        }
}
```

运行结果如图 4.17 所示。

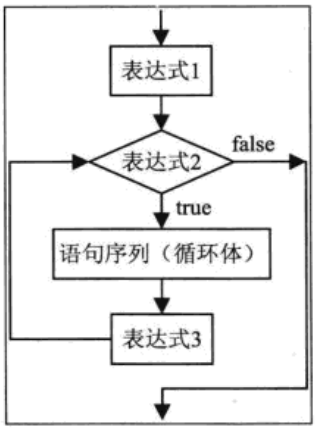


图 4.16　for 循环语句的执行过程

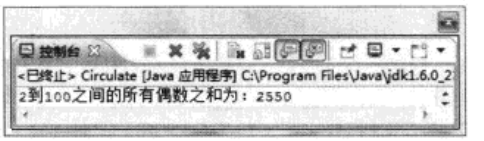


图 4.17　使用 for 循环语句计算偶数和

2．foreach 语句

foreach 语句是 for 语句的特殊简化版本，foreach 语句并不能完全取代 for 语句，然而任何 foreach 语句都可以改写为 for 语句版本。foreach 并不是一个关键字，习惯上将这种特殊的 for 语句格式称之为 foreach 语句。foreach 语句在遍历数组等方面为程序员提供了很大的方便（本书将在第 5 章对数组进行详细的介绍）。语法格式如下：

```
for(元素变量 x : 遍历对象 obj) {
    引用了 x 的 Java 语句;
}
```

foreach 语句中的元素变量 x，不必对其进行初始化。下面通过简单的例子来介绍 foreach 语句是怎样遍历一维数组的。

【例 4.12】　在项目中创建 Repetition 类，在主方法中定义一维数组，并用 foreach 语句遍历该数组。（**实例位置：光盘\TM\sl\4\10**）

```
public class Repetition {                                    //创建 Repetition 类
    public static void main(String args[]){                  //主方法
        int arr[] = {7, 10, 1};                              //声明一维数组
        System.out.println("一维数组中的元素分别为：");       //输出信息
        for (int x : arr) {                                  //foreach 语句
        System.out.println(x+"\t");
        }
    }
}
```

运行结果如图 4.18 所示。

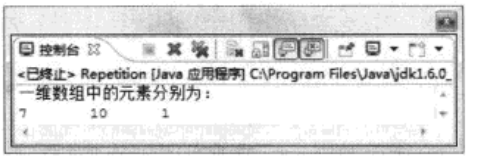


图 4.18　使用 foreach 语句遍历数组

4.3.4　范例 3：使用 while 循环遍历数组

本范例利用自增运算符结合 while 循环获取每个数组元素的值，然后把它们输出到控制台中。其中自增运算符控制索引变量的递增。运行结果如图 4.19 所示。（**实例位置：光盘\TM\sl\4\11**）

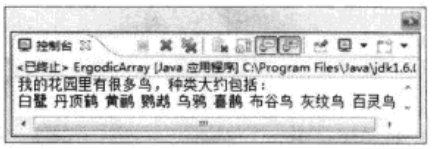


图 4.19　使用 while 循环遍历数组

创建 ErgodicArray 类，在该类的主方法中创建一个鸟类数组，然后创建一个索引变量，这个变量用于指定数组下标，随着该索引的递增，while 循环会逐步获取每个数组的元素并输出到控制台中。代码如下：

```
public class ErgodicArray {
    public static void main(String[] args) {
        String[] aves = new String[] { "白鹭", "丹顶鹤", "黄鹂", "鹦鹉", "乌鸦", "喜鹊",
                "布谷鸟", "灰纹鸟", "百灵鸟" };                                    //创建鸟类数组
        int index = 0;                                                            //创建索引变量
        System.out.println("我的花园里有很多鸟，种类大约包括：");
        while (index < aves.length) {                                             //遍历数组
            System.out.print(aves[index++]+"   ");                                //自增索引值
        }
    }
}
```

4.3.5　范例 4：使用 for 循环输出九九乘法表

Java 基本语法中的 for 循环非常灵活并且可以嵌套使用，其中双层 for 循环是程序开发中使用最频繁的，常用于操作表格数据，对于行数与列数相同的表格操作代码比较简单，但是类似九九乘法表就不好控制了，因为它的列数要与行数对应，可以说这个表格是个三角形，本范例通过双层循环输出了这个九九乘法表。运行结果如图 4.20 所示。（**实例位置：光盘\TM\sl\4\12**）

创建 MultiplicationTable 类，在该类的主方法中创建双层 for 循环。第一层 for 循环也称为外层循环，用于控制表格的行；第二层循环也称为内层循环，用于控制表格的列。这里第二层循环的控制变量非常重要，它的条件判断是列数要等于行数的最大值，然后输出内层与外层循环控制变量的乘积，这样就实现了九九乘法表。代码如下：

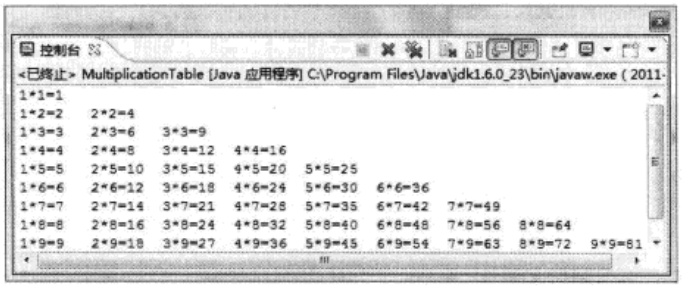


图 4.20　使用 for 循环输出乘法表

```
public class MultiplicationTable {
    public static void main(String[] args) {
        for(int i=1;i<=9;i++){                          //循环控制变量从 1 遍历到 9
            for(int j=1;j<=i;j++){                      //第二层循环控制变量与第一层最大索引相等
                System.out.print(j+"*"+i+"="+i*j+"\t");  //输出计算结果但不换行
            }
            System.out.println();                       //在外层循环中换行
        }
    }
}
```

4.4　跳 转 语 句

视频讲解：光盘\TM\lx\4\跳转语句.exe

Java 语言中提供了 3 种跳转语句，分别是 break 语句、continue 语句和 return 语句。下面对这 3 种跳转语句进行详细介绍。

4.4.1　break 语句

break 语句大家应该不会陌生，在介绍 switch 语句时已经应用过了。在 switch 语句中，break 语句用于中止下面 case 语句的比较。实际上，break 语句还可以应用在 for、while 和 do…while 循环语句中，用于强行退出循环，也就是忽略循环体中任何其他语句和循环条件的限制。

【例 4.13】 使用 for 循环语句计算 1～100 之间所有连续整数的和。（**实例位置：光盘\TM\sl\4\13**）

```
public class MultiplicationTable {
    public static void main(String[] args) {
    int sum=0;                                          //定义保存数据和的 int 变量
    String flag="从 1 到 100 之间连续整数的和是：";      //给出输出信息
    for(int i=1;i<=100;i++){                            //循环获取从 1～100 的数
        sum+=i;                                         //将各数相加
    }
```

```
    System.out.println(flag+sum);                          //输出相加后的结果
    }
}
```

运行结果如图 4.21 所示。

在上面的循环中添加通过 if 语句控制的 break 语句，具体代码如下：

```
public class MultiplicationTable {
    public static void main(String[] args) {
    int sum=0;
    String flag="从 1 到 100 之间连续整数的和是：";
    for(int i=1;i<=100;i++){                               //循环获取从 1～100 的数
        sum+=i;                                            //将各数进行相加
        if(sum>1000){                                      //如果 sum 大于 1000
          flag="从 1 到"+i+"之间连续整数的和是：";
          break;                                           //break 关键字退出循环
        }
    }
    System.out.println(flag+sum);
    }
}
```

运行结果如图 4.22 所示。

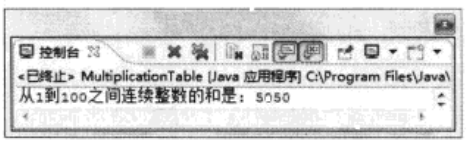


图 4.21　输出 1～100 的和

图 4.22　使用 break 语句

从上面的两段代码中可以看出，虽然 for 循环被设计为计算从 1～100 之间所有连续整数的和，但是由于当累加和大于 1000 时使用 break 语句中止了 for 循环语句，所以当循环结束时 i 的值并不等于 100，而是等于 45。需要说明的是，使用 break 语句只能退出当前循环。

4.4.2　continue 语句

continue 语句只能应用在 for、while 和 do…while 循环语句中，用于让程序直接跳过其后面的语句，进行下一次循环。

【例 4.14】　在项目中创建 ContinueDemo 类，在主方法中应用 while 循环语句和 continue 语句输出 10 以内的全部奇数。（**实例位置：光盘\TM\sl\4\14**）

```
public class ContinueDemo{
    public static void main(String[] args) {
        int i = 0;                                         //定义循环增量
        System.out.println("十以内的全部奇数是：");
        while (i < 10) {
```

```
            i++;                                    //累加 i 的值
            if (i % 2 == 0) {                       //当 i 的值能被 2 整除，表示该数不是奇数
                continue;                           //进行下一次循环
            }
            System.out.print(i + "  ");             //输出 i 的值
        }
    }
}
```

运行结果如图 4.23 所示。

当使用 continue 语句中止本次循环后，如果循环条件的结果为 false，则退出循环，否则继续下一次循环。

图 4.23　输出 10 以内的全部奇数

4.4.3　return 语句

return 语句可以从一个方法返回，并把控制权交给调用它的语句。语法格式如下：

```
return [表达式];
```

表达式：可选参数，表示要返回的值。它的数据类型必须与方法声明中的返回值类型一致，可以通过强制类型转换实现。

return 语句通常被放在被调用方法的最后，用于退出当前方法并返回一个值。当把单独的 return 语句放在一个方法的中间时，会产生 Unreachable code 编译错误。但是可以通过把 return 语句用 if 语句括起来的方法，将 return 语句放在一个方法中间，用来实现在程序未执行完方法中的全部语句时退出。

4.4.4　范例 5：终止循环体

循环用于复杂的业务处理，可以提高程序的性能和代码的可读性，但是循环中也有特殊情况，如由于某些原因需要立刻中断循环去执行下面的业务逻辑。运行结果如图 4.24 所示。（**实例位置：光盘\TM\sl\4\15**）

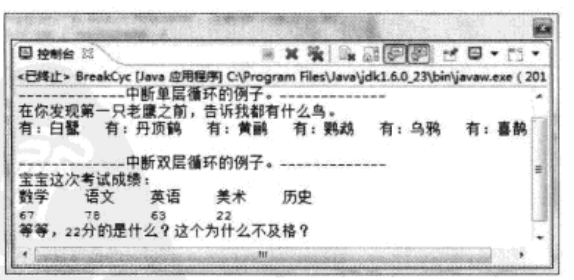


图 4.24　终止循环体

在项目中创建 BreakCyc 类，在该类的主方法中创建一个字符串数组，在使用 foreach 语句遍历时

判断如果发现数组中包含字符串“老鹰”则立刻中断循环。然后创建一个整数类型二维数组，使用双层 foreach 循环遍历，当发现第一个小于 60 的数组元素时，则立刻中断整个双层循环，而不是内层循环。代码如下：

```
public class BreakCyc {
    public static void main(String[] args) {
        System.out.println("\n------------中断单层循环的例子。------------");
        String[] array = new String[] { "白鹭", "丹顶鹤", "黄鹂", "鹦鹉", "乌鸦", "喜鹊",      //创建数组
                "老鹰", "布谷鸟", "老鹰", "灰纹鸟", "老鹰", "百灵鸟" };
        System.out.println("在你发现第一只老鹰之前，告诉我都有什么鸟。");
        for (String string : array) {                                                      //foreach 遍历数组
            if (string.equals("老鹰"))                                                      //如果遇到老鹰
                break;                                                                     //中断循环
            System.out.print("有：" + string+"          ");                                  //否则输出数组元素
        }

        System.out.println("\n\n------------中断双层循环的例子。------------");

        int[][] myScores = new int[][] { { 67, 78, 63, 22, 66 },                           //创建成绩数组
                { 55, 68, 78, 95, 44 }, { 95, 97, 92, 93, 81 } };
        System.out.println("宝宝这次考试成绩：\n 数学\t 语文\t 英语\t 美术\t 历史");
        No1: for (int[] is : myScores) {                                                   //遍历成绩表格
            for (int i : is) {
                System.out.print(i + "\t");                                                //输出成绩
                if (i < 60) {                                  //如果中途遇到不及格的，立刻中断所有输出
                    System.out.println("\n 等等，" + i + "分的是什么？这个为什么不及格？");
                    break No1;
                }
            }
            System.out.println();
        }
    }
}
```

4.4.5　范例 6：循环体的过滤器

循环体中可以通过 break 语句中断整个循环，这增加了循环的控制能力，但是对于特殊情况还是不够，例如某些条件下需要放弃部分循环处理，而不是整个循环体。Java 提供了 continue 语句来实现这一功能，continue 可以放弃本次循环体的剩余代码，不执行它们而开始下一轮的循环。本范例利用 continue 语句实现了循环体过滤器，可以过滤“老鹰”字符串，并做相应的处理，但是放弃 continue 语句之后的所有代码。运行结果如图 4.25 所示。（**实例位置：光盘\TM\sl\4\16**）

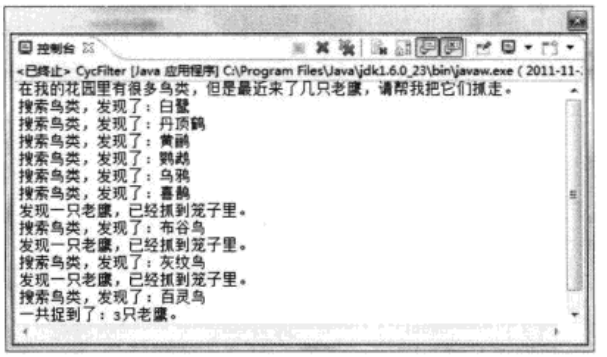


图 4.25　循环体的过滤器

在项目中创建 CycFilter 类，在该类的主方法中创建鸟类名称的字符串数组，其中包含多个“老鹰”字符串，然后通过 foreach 循环遍历该数组，在循环过程中如果遍历的数组元素是“老鹰”字符串，则输出发现老鹰的信息并过滤循环体之后的所有代码。代码如下：

```
public class CycFilter {
    public static void main(String[] args) {
        String[] array = new String[] { "白鹭", "丹顶鹤", "黄鹂", "鹦鹉", "乌鸦", "喜鹊",        //创建数组
                "老鹰", "布谷鸟", "老鹰", "灰纹鸟", "老鹰", "百灵鸟" };
        System.out.println("在我的花园里有很多鸟类，但是最近来了几只老鹰，请帮我把它们抓走。");
        int eagleCount = 0;
        for (String string : array) {                                          //foreach 遍历数组
            if (string.equals("老鹰")) {                                        //如果遇到老鹰
                System.out.println("发现一只老鹰，已经抓到笼子里。");
                eagleCount++;
                continue;                                                      //中断循环
            }
            System.out.println("搜索鸟类，发现了：" + string);                   //否则输出数组元素
        }
        System.out.println("一共捉到了：" + eagleCount + "只老鹰。");
    }
}
```

break 语句和 continue 语句都是对循环体的控制语句，它们不仅应用于 for 循环，在任何循环体中都可以使用这些语句，灵活使用可以让循环实现更加复杂的运算和业务处理。

4.5　经 典 范 例

4.5.1　经典范例 1：使用 for 循环输出空心的菱形

视频讲解：光盘\TM\lx\4\使用 for 循环输出空心的菱形.exe

输出空心的菱形图案，这在等级考试与公司面试时也出现过类似题目，本范例的目的在于熟练掌

握 for 循环的嵌套使用。运行结果如图 4.26 所示。（**实例位置：光盘\TM\sl\4\17**）

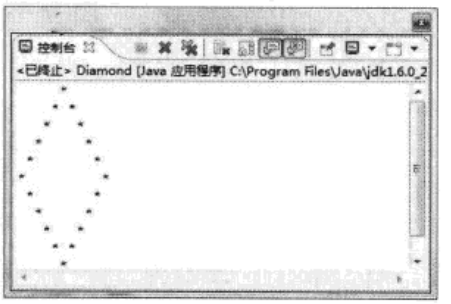


图 4.26　输出空心菱形

创建 Diamond 类，在该类的主方法中调用 printHollowRhombus()方法完成 10 行的空心菱形输出。其中 printHollowRhombus()方法是范例中自定义的，该方法使用两个双层 for 循环分别输出菱形的上半部分与下半部分。代码如下：

```
public class Diamond {
    public static void main(String[] args) {
        printHollowRhombus(10);
    }

    public static void printHollowRhombus(int size) {
        if (size % 2 == 0) {
            size++;                                          //计算菱形大小
        }
        for (int i = 0; i < size / 2 + 1; i++) {
            for (int j = size / 2 + 1; j > i + 1; j--) {
                System.out.print(" ");                       //输出左上角位置的空白
            }
            for (int j = 0; j < 2 * i + 1; j++) {
                if (j == 0 || j == 2 * i) {
                    System.out.print("*");                   //输出菱形上半部边缘
                } else {
                    System.out.print(" ");                   //输出菱形上半部空心
                }
            }
            System.out.println("");
        }
        for (int i = size / 2 + 1; i < size; i++) {
            for (int j = 0; j < i - size / 2; j++) {
                System.out.print(" ");                       //输出菱形左下角空白
            }
            for (int j = 0; j < 2 * size - 1 - 2 * i; j++) {
                if (j == 0 || j == 2 * (size - i - 1)) {
                    System.out.print("*");                   //输出菱形下半部边缘
                } else {
                    System.out.print(" ");                   //输出菱形下半部空心
                }
```

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

4.5.2 经典范例 2：使用 for 循环输出杨辉三角

视频讲解：光盘\TM\lx\4\使用 for 循环输出杨辉三角.exe

杨辉三角形由数字排列，可以把它看作一个数字表，其基本特性是两侧数值均为 1，其他位置的数值是其正上方的数值与左上角数值之和。本范例通过数组来实现这个杨辉三角形。运行结果如图 4.27 所示。（**实例位置：光盘\TM\sl\4\18**）

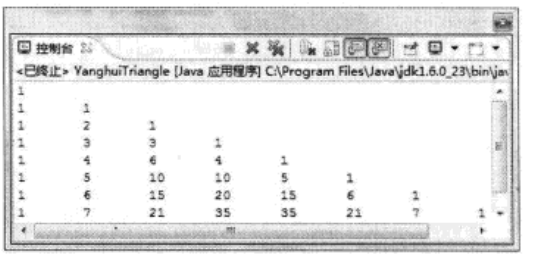


图 4.27 输出杨辉三角

创建 YanghuiTriangle 类，在该类的主方法中创建一个二维数组，并指定二维数组的第一维长度，这个数组用于存放杨辉三角形的数值表，通过双层 for 循环来实现第二维数组的长度，然后计算整个数组的每个元素的值。代码如下：

```
public class YanghuiTriangle {
    public static void main(String[] args) {
        int triangle[][]=new int[8][];                          //创建二维数组
        for (int i = 0; i < triangle.length; i++) {             //遍历二维数组的第一层
            triangle[i]=new int[i+1];                           //初始化第二层数组的大小
            for(int j=0;j<=triangle[i].length-1;j++){           //遍历第二层数组
                if(i==0||j==0||j==triangle[i].length-1){        //将两侧的数组元素赋值为 1
                    triangle[i][j]=1;
                }else{                                          //其他数值通过公式计算
                    triangle[i][j]=triangle[i-1][j]+triangle[i-1][j-1];
                }
                System.out.print(triangle[i][j]+"\t");          //输出数组元素
            }
            System.out.println();
        }
    }
}
```

4.6 本章小结

本章向读者介绍了流程控制语句（复合语句、条件语句和循环语句）。使用复合语句可以为变量定义一个有效区域。通过使用 if 与 switch 语句，可以基于布尔类型的测试，将一个程序分成不同的部分。

通过 while、do…while 循环语句和 for 循环语句，可以让程序的一部分重复地执行，直到满足某个终止循环的条件。通过本章的学习，读者应该学会在程序中灵活使用流程控制语句。

4.7 实 战 练 习

1. 编写 Java 程序，实现判断变量 x 是奇数还是偶数。（**答案位置：光盘\TM\sl\4\19**）
2. 编写 Java 程序，应用 for 循环打印菱形。（**答案位置：光盘\TM\sl\4\20**）
3. 编写 Java 程序，使用 while 循环语句计算 1+1/2!+1/3!…1/20!之和。（**答案位置：光盘\TM\sl\4\21**）

PDG

第 5 章

数组

（视频讲解：31 分钟）

数组是最为常见的一种数据结构，是相同类型的、用一个标识符封装到一起的基本类型数据序列或对象序列。可以用一个统一的数组名和下标来唯一确定数组中的元素。实质上数组是一个简单的线性序列，因此数组访问起来很快。本章将向读者介绍有关数组的知识。

通过阅读本章，您可以：

- 掌握一维数组创建和使用的方法
- 掌握多维数组创建和使用的方法
- 了解如何遍历数组
- 了解如何填充替换数组中的元素
- 了解如何对数组进行排序
- 了解如何复制数组

PDG

5.1 数组概述

视频讲解：光盘\TM\lx\5\数组概述.exe

数组是具有相同数据类型的一组数据的集合。当需要使用的变量很多，而且数据类型相同时，逐个声明就显得非常麻烦，这时可以声明一个数组，然后对数组进行操作，从而省去了不少操作。例如，球类的集合——足球、篮球、羽毛球等；电器集合——电视机、洗衣机、电风扇等，就可以分别定义在一个数组中。Java 语言中虽然基本数据类型不是对象，但是由基本数据类型组成的数组则是对象，在程序设计中引入数组可以更有效地管理和处理数据。

数组根据维数的不同分为一维数组、二维数组和多维数组，大家习惯地将一维看成直线、二维看成平面、三维看成立体空间，那再多维又该怎么理解呢？其实 Java 语言中数组的维数并不用如此理解，这样非但不容易理解，反而会有使用的不便。下面给大家介绍一种简单的理解方法，如图 5.1 所示。

通过图示就能很好理解了，通俗地讲就是一维数组的每个基本单元都是基本数据类型的数据；二维数组就是每个基本单元是一维数组的一维数组；依此类推，n 维数组的每个基本单元都是 n-1 维数组的 n-1 维数组。下面通过三维数组实例进一步加深理解，如图 5.2 所示。

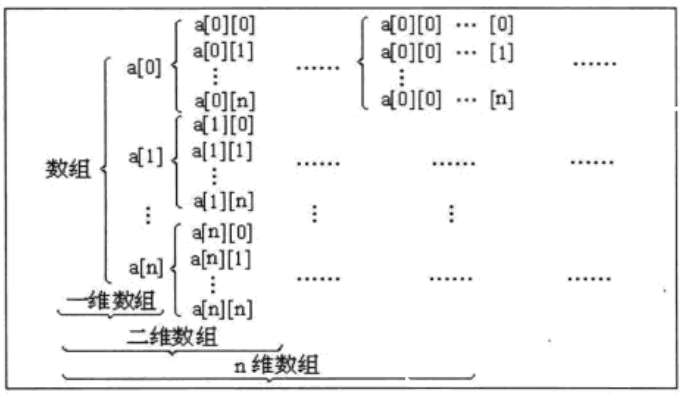


图 5.1 各维数组的理解图

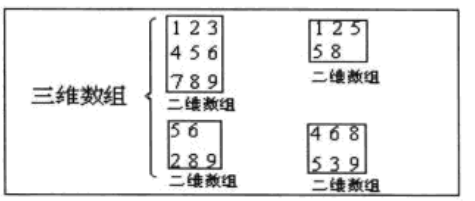


图 5.2 三维数组示意图

图 5.2 中三维数组的元素都是一个二维数组，再把每个二维数组看成一个整体，则此三维数组也是一个二维数组的结构。

5.2 一维数组

视频讲解：光盘\TM\lx\5\一维数组.exe

一维数组实质上是一组相同类型数据的集合，当需要在程序中处理一组数据或者传递一组数据时，可以应用这种类型的数组。本节将向读者介绍一维数组。

5.2.1　创建一维数组

数组作为对象允许使用 new 关键字进行内存分配。在使用数组之前，必须首先定义数组变量所属的类型，即声明数组。声明一维数组有两种形式，语法格式分别如下：

```
数组元素类型　数组名字[];
数组元素类型[]　数组名字;
```

- ☑ 数组元素类型：决定了数组的数据类型，它可以是 Java 中任意的数据类型，包括基本数据类型和非基本数据类型。
- ☑ 数组名字：为一个合法的标识符。
- ☑ 符号“[ ]”：指明该变量是一个数组类型变量，单个“[ ]”表示要创建的数组是一维数组。

【例 5.1】　声明一维数组。

```
int arr[];                //声明 int 型数组，数组中的每个元素都是 int 型数值
String []str;             //声明 String 数组，数组中的每个元素都是 String 型数值
```

声明数组后，还不能访问它的任何元素，因为声明数组仅仅是给出了数组名字和元素的数据类型，要想真正使用数组还要为其分配内存空间，且分配内存空间时必须指明数组的长度。分配内存空间的语法格式如下：

```
数组名字 = new 数组元素类型[数组元素的个数];
```

- ☑ 数组名字：已经声明的数组变量的名称。
- ☑ new：对数组分配空间的关键字。
- ☑ 数组元素的个数：指定数组中变量的个数，即数组的长度。

【例 5.2】　为数组分配内存。

```
arr = new int[5];
```

上述代码表示为已经创建好的数组 arr 分配长度为 5 的内存空间，即该数组可以有 5 个元素。应该注意的是，在内存空间中数组的下标是从 0 开始的。arr 的内存存储状态如图 5.3 所示。

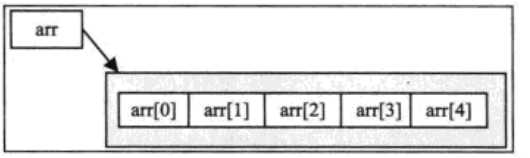


图 5.3　一维数组的内存模式

说明

使用 new 关键字为数组分配内存时，数组中各个元素的初始化值都为 0。

创建数组和分配内存不一定要分开执行，可以在创建数组时直接为变量赋值。语法格式如下：

```
数组元素类型 数组名[] = new 数组元素类型[数组元素的个数];
```

【例 5.3】 声明一维数组并分配内存。

```
int month[ ] = new int[12];
```

上述代码创建了一个一维数组 month，并指定了数组长度是 12。这种创建数组的方法也是 Java 程序编写过程中普遍的做法。

注意 无论用何种方法声明数组，前面的中括号中都不可以填写内容，否则将会在编译时出现错误。

5.2.2 初始化一维数组

数组可以与基本数据类型一样进行初始化操作，数组的初始化可分别初始化数组中每个元素。数组的初始化有两种形式。

【例 5.4】 两种方法初始化一维数组。

```
int arr[] = new int[]{1,2,3,5,25};          //第一种初始化方式
int arr2[] = {34,23,12,6};                  //第二种初始化方式
```

数组的初始化方式是：把数据类型包括在大括号之内，中间用逗号分开数组元素的值，系统自动为数组分配一定的空间。第一种初始化方式，创建 5 个元素的数组，其值依次为 1、2、3、5、25；第二种初始化方式，创建 4 个元素的数组，其值依次为 34、23、12、6。

说明 初始化数组时可以省略 new 运算符和数组的长度，编译器将根据初始值的数量来自动计算数组长度，并创建数组。

5.2.3 范例 1：求一维数组各元素的和

一维数组每个元素都有自己的值，使用 for 循环根据数组的下标，将数组的每个元素的值相加求和。运行结果如图 5.4 所示。（**实例位置：光盘\TM\sl\5\1**）

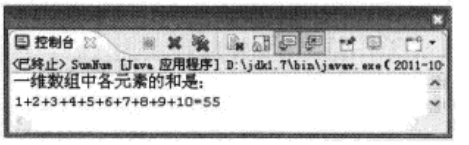


图 5.4 输出一维数组各元素的和

在项目中创建 SumNum 类，在类的主方法中使用数组存储 1～10 这 10 个整数，然后求出这 10 个数的和，并在控制台输出。代码如下：

```
public class SumNum {
    public static void main(String[] args) {
        int[] num = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };          //创建并初始化一维数组 num
```

```
        int sum = 0;                                        //进行累加求和的变量 sum
        System.out.println("一维数组中各元素的和是：");
        for (int i = 0; i < 10; i++) {                      //通过 for 循环遍历数组
            if (i == 9) {                                   //判断数组的下标是否为 9
                System.out.print(num[i] + "=");             //数组元素的下标是 9，输出等号
            } else {
                System.out.print(num[i] + "+");             //数组元素的下标不是 9，输出加号
            }
            sum = sum + num[i];                             //进行累加求和
        }
        System.out.println(sum);                            //输出和
    }
}
```

5.2.4　范例 2：获取一维数组的最小值

一维数组的每个元素可以不同，有大有小，通过 for 循环和 if 条件语句，再根据数组下标求出数组元素中的最小值。运行结果如图 5.5 所示。（**实例位置：光盘\TM\sl\5\2**）

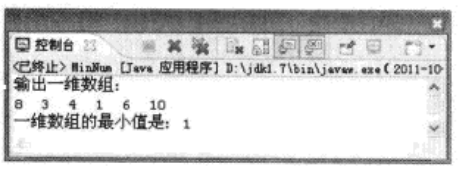


图 5.5　获取一维数组的最小值

在项目中创建 MinNum 类，在类的主方法中创建一维数组，求数组中值最小的元素，并在控制台输出。代码如下：

```
public class MinNum {
    public static void main(String[] args) {
        int[] num = { 8, 3, 4, 1, 6, 10 };                  //创建并初始化一维数组 num
        System.out.println("输出一维数组：");
        for (int i = 0; i < num.length; i++) {
            System.out.print(num[i] + "   ");               //输出一维数组
        }
        int min = num[0];
        for (int j = 0; j < num.length - 1; j++) {          //通过 for 循环遍历数组
            if (min > num[j + 1]) {
                min = num[j + 1];
            }
        }
        System.out.println("\n 一维数组的最小值是：" + min);   //输出数组的最小值
    }
}
```

5.3 多维数组

视频讲解：光盘\TM\lx\5\多维数组.exe

二维和多于二维的数组被统称为多维数组，其中二维数组在程序中经常会使用，三维数组也偶尔会使用，所以本节主要介绍二维数组和三维数组。

5.3.1 二维数组

如果一维数组中的各个元素仍然是一维数组，那么它就是一个二维数组。二维数组常用于表示表，表中的信息以行和列的形式组织，第一个下标代表元素所在的行，第二个下标代表元素所在的列。

1. 二维数组的创建

声明二维数组的方法有两种，语法格式分别如下：

```
数组元素类型 数组名字[][];
数组元素类型[][] 数组名字;
```

- ☑ 数组元素类型：决定了数组的数据类型，它可以是 Java 中任意的数据类型，包括基本数据类型和非基本数据类型。
- ☑ 数组名字：为一个合法的标识符。
- ☑ 符号“[]”：指明该变量是一个数组类型变量，两个“[]”表示要创建的数组是二维数组。

【例 5.5】 创建二维数组。

```
int myarr[][];
```

与一维数组一样，如果二维数组在声明时没有分配内存空间，同样也要使用关键字 new 来分配内存，然后才可以访问每个元素。

二维数组可以看成是由多个一维数组所组成，在给二维数组分配内存时，可以为这些一维数组同时分配相同的内存。第一个中括号中的数字是一维数组的个数，第二个中括号中的数字是这些一维数组的长度。

【例 5.6】 为二维数组每一维分配相同内存。

```
a = new int[2][4];
```

为二维数组分配内存，分配后二维数组 a 拥有两个长度是 4 的一维数组。分配的内存如图 5.6 所示。

给二维数组分配内存时，还可以对其每一个一维数组单独分配内存，且分配的内存可以并不相同，在第一个中括号中定义一维数组的个数，然后就利用一维数组分配内存的方法分配内存。

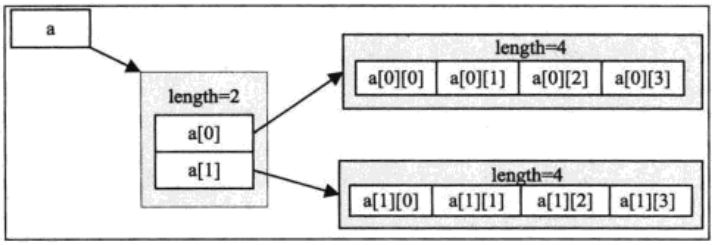


图 5.6　为二维数组分配内存

【例 5.7】　分别为每一维单独分配内存。

```
a = new int[2][];
a[0] = new int[2];
a[1] = new int[3];
```

为二维数组分配内存，先给二维数组分配两个一维数组，然后对第一个一维数组分配内存 2；第二个一维数组分配内存 3。分配的内存如图 5.7 所示。

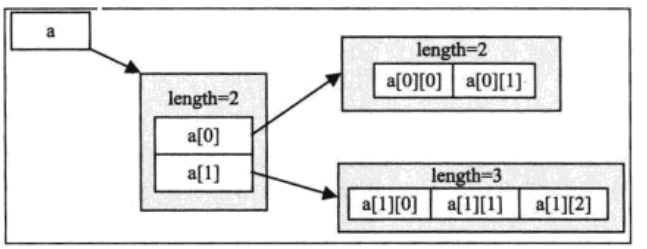


图 5.7　为二维数组分配内存

2．二维数组初始化

二维数组的初始化与一维数组初始化类似，同样可以使用大括号完成二维数组的初始化。语法格式如下：

```
type arrayname[][] = {value1,value2…valuen};
```

☑　type：数组数据类型。

☑　arrayname：数组名称，一个合法的标识符。

☑　value：数组中各元素的值。

【例 5.8】　初始化二维数组。

```
int myarr[][] = {{12,0},{45,10}};
```

初始化二维数组后，要明确数组的下标都是从 0 开始。如上面的代码中 myarr[1][1]的值为 10。int 型二维数组是以 int a[][]来定义的，所以可以直接给 a[x][y]赋值。如给 a[1]的第二个元素赋值：

```
a[1][1] = 20;
```

【例 5.9】　在项目中创建 Matrix 类，在主方法中编写代码实现输出一个 3 行 4 列且所有元素都是 0 的矩阵。

```
public class Matrix {                                          //创建类
    public static void main(String[] args) {                   //主方法
        int a[][] = new int[3][4];                             //定义二维数组
        System.out.println("输出 3 行 4 列的数组：");
        for (int i = 0; i < a.length; i++) {
            for (int j = 0; j < a[i].length; j++) {            //循环遍历数组中的每个元素
                System.out.print(a[i][j]+" ");                 //将数组中的元素输出
            }
            System.out.println();                              //输出空格
        }
    }
}
```

运行结果如图 5.8 所示。

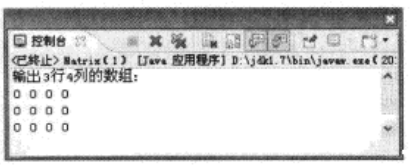


图 5.8　输出 3 行 4 列的矩阵

说明

对于整型二维数组，创建成功之后系统会赋给数组中每个元素初始化值 0。

5.3.2　三维数组

三维数组的声明与一、二维数组类似，一维使用一个中括号、二维使用两个中括号，依此类推，三维则使用 3 个中括号；初始化三维数组时，由 3 层大括号进行初始化；使用时也更麻烦一些，需要使用 3 层 for 循环。

【例 5.10】　在项目中创建 Ransack 类，在类的主方法中创建三维数组，并将三维数组在控制台输出。（**实例位置：光盘\TM\sl\5\3**）

```
public class Ransack {                                         //创建类
    public static void main(String[] args) {
            int arr[][][] = new int[][][] {                    //创建三维数组
            {{ 1, 2, 3 }, { 4, 5, 6 }},
            {{ 7, 8, 9 }, {10, 11, 12}},
            {{13, 14, 15}, {16, 17, 18}}
            };
        for (int i = 0; i < arr.length; i++) {                 //遍历数组
            System.out.println("三维数组的第"+(i+1)+"个元素是一个"+arr[0].length
                    +"维数组，内容如下：");
            for (int j = 0; j < arr[0].length; j++) {          //遍历数组
                for (int k = 0; k < arr[0][0].length; k++) {
```

```
                    System.out.print(arr[i][j][k] + "\t");                    //输出数组元素
                }
                System.out.println();
            }
        }
    }
}
```

运行结果如图 5.9 所示。

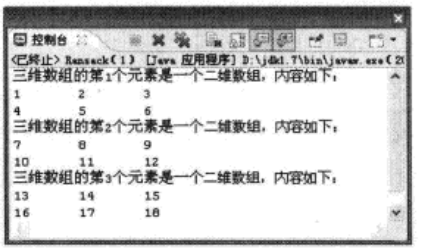


图 5.9　使用三维数组

三维数组的元素是二维数组，二维数组的元素是一维数组，每个元素使用 arr[x][y][z]的格式表示，如 arr[0][1][2]就代表数 6。

5.3.3　范例 3：对矩阵进行转置运算

所谓矩阵的转置就是将矩阵的行列互换，把 a[i][j]的值存储在元素 a[j][i]的位置，转换前是 m×n 维矩阵，转换后则是 n×m 维矩阵。运行结果如图 5.10 所示。（**实例位置：光盘\TM\sl\5\4**）

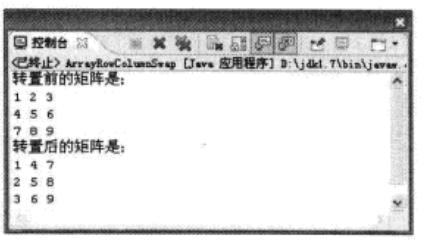


图 5.10　矩阵的转置

在项目中创建 ArrayRowColumnSwap 类，在类的主方法中声明二维矩阵，然后将此矩阵进行转置运算，并将结果在控制台输出。代码如下：

```
public class ArrayRowColumnSwap {                                    //创建类
    public static void main(String[] args) {
        int arr[][] = new int[][] { { 1, 2, 3 }, { 4, 5, 6 }, { 7, 8, 9 } };    //创建二维数组
        System.out.println("转置前的矩阵是：");
        printArray(arr);                                             //输出二维数组
        int arr2[][] = new int[arr.length][arr.length];
        for (int i = 0; i < arr.length; i++) {                       //调整数组行列数据
            for (int j = 0; j < arr[i].length; j++) {
                arr2[j][i] = arr[i][j];
            }
        }
        System.out.println("转置后的矩阵是：");
        printArray(arr2);                                            //输出二维数组
    }
```

```
    private static void printArray(int[][] arr) {
        for (int i = 0; i < arr.length; i++) {                    //遍历数组
            for (int j = 0; j < arr.length; j++) {
                System.out.print(arr[i][j] + " ");                //输出数组元素
            }
            System.out.println();
        }
    }
}
```

5.3.4 范例 4：求方阵的迹

方阵的迹，即方阵主对角线上所有元素的和，英文名是 Trace of the matrix。迹同时也是方阵的所有特征值之和，求法有很多。下面通过 Java 语言编写程序求方阵的迹。运行结果如图 5.11 所示。（**实例位置：光盘\TM\sl\5\5**）

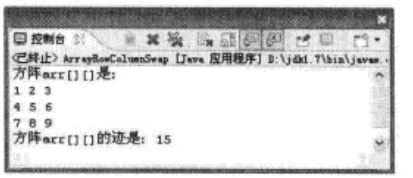


图 5.11 求方阵的迹

在项目中创建 Trace 类，在类的主方法中创建二维方阵，然后求出二维方阵的迹，并在控制台将结果输出。代码如下：

```
public class Trace {                                              //创建类
    public static void main(String[] args) {
        int arr[][] = new int[][] { { 1, 2, 3 }, { 4, 5, 6 }, { 7, 8, 9 } };   //创建二维数组
        int tr = 0;
        System.out.println("方阵 arr[][]是：");
        for (int i = 0; i < arr.length; i++) {                    //遍历数组
            for (int j = 0; j < arr.length; j++) {
                System.out.print(arr[i][j] + " ");                //输出数组元素
            }
            System.out.println();
        }
        for (int i = 0; i < arr.length; i++) {
            tr += arr[i][i];
        }
        System.out.println("方阵 arr[][]的迹是："+tr);
    }
}
```

5.4 数组的基本操作

视频讲解：光盘\TM\lx\5\数组的基本操作.exe

java.util 包的 Arrays 类包含用来操作数组（如排序和搜索）的各种方法。本节将向读者介绍数组的

基本操作。

5.4.1　遍历数组

遍历数组就是获取数组中的每个元素。通常遍历数组都是使用 for 循环来实现。

1．遍历一维数组

【例 5.11】　在项目中创建 GetDay 类，在主方法中创建 int 型数组，并实现将各月的天数输出。

```
public class GetDay {                                                    //创建类
    public static void main(String[] args) {                             //主方法
        int day[] = new int[] { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };   //创建并初始化一维数组
        for (int i = 0; i < 12; i++) {                                   //利用循环将信息输出
            System.out.print((i + 1) + "月有" + day[i] + "天" + "\t");   //输出信息
            if ((i + 1) % 3 == 0) {                                      //i+1 和 3 求余为零
                System.out.print('\n');                                  //输出回车
            }
        }
    }
}
```

运行结果如图 5.12 所示。

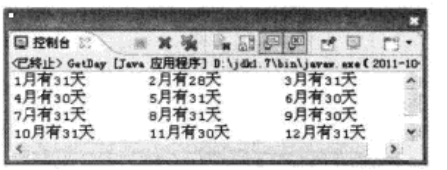


图 5.12　输出每月的天数

2．遍历二维数组

遍历二维数组要比遍历一维数组麻烦一些，需使用双层 for 循环，还要通过数组的 length 属性获得数组的长度。

【例 5.12】　在项目中创建 Trap 类，在类的主方法中编写代码，定义二维数组，实现将二维数组中的元素输出。

```
public class Trap {                                                      //创建类
    public static void main(String[] args) {                             //主方法
        int b[][] = new int[][] { { 1 }, { 2, 3 }, { 4, 5, 6 } };        //定义二维数组
        System.out.println("二维数组是：");
        for (int k = 0; k < b.length; k++) {
            for (int c = 0; c < b[k].length; c++) {                      //循环遍历二维数组中的每个元素
                System.out.print(b[k][c]+"   ");                         //将数组中的元素输出
            }
            System.out.println();                                        //换行
        }
    }
}
```

运行结果如图 5.13 所示。

在遍历数组时，使用 foreach 语句更简单，该语句并不是一个新的语法，而是 for 循环的简化格式。下面通过 foreach 语句遍历二维数组。

【例 5.13】　在项目中创建 Tautog 类，在主方法中定义二维数组，使用 foreach 语句遍历二维数组。

```
public class Tautog {                                          //创建类
    public static void main(String[] args) {                   //主方法
        int arr2[][] = { { 3, 4, 3 }, { 1, 2 } };              //定义二维数组
        System.out.println("二维数组中的元素是：");              //提示信息
        for (int x[] : arr2) {                                 //外层循环变量为一维数组
            for (int e : x) {                                  //循环遍历每一个数组元素
                System.out.print(e + "   ");                   //输出二维数组的元素
            }
            System.out.println();
        }
    }
}
```

运行结果如图 5.14 所示。

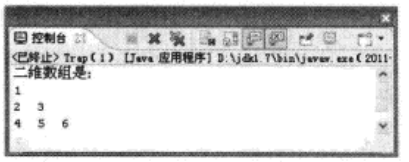


图 5.13　遍历二维数组

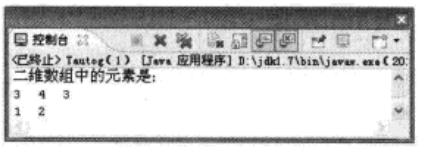


图 5.14　用 foreach 语句遍历二维数组

5.4.2　填充替换数组元素

数组中的元素定义完成后，可通过 Arrays 类的静态方法 fill()来对数组中的元素进行替换。该方法通过各种重载形式可完成任意类型的数组元素的替换。fill()方法有两种参数类型。下面以 int 型数组为例介绍 fill()方法的使用方法。

1．fill(int[] a, int value)方法

该方法可将指定的 int 值分配给 int 型数组的每个元素。语法格式如下：

```
fill(int[] a , int value)
```

☑　a：要进行元素替换的数组。

☑　value：要存储数组中所有元素的值。

☑　返回值：填充后的数组。

【例 5.14】　在项目中创建 Swap 类，在主方法中创建一维数组，并实现通过 fill()方法填充数组元素，最后将数组中的各个元素输出。

```
import java.util.Arrays;                                       //导入 java.util.Arrays 类
public class Swap {                                            //创建类
    public static void main(String[] args) {                   //主方法
        int arr[] = new int[5];                                //创建 int 型数组
        Arrays.fill(arr,8);                                    //使用同一个值对数组进行填充
        for(int i = 0;i<arr.length;i++){                       //循环遍历数组中的元素
```

```
            System.out.println("第"+(i+1)+"个元素是："+arr[i]);    //将数组中的元素依次输出
        }
    }
}
```

运行结果如图 5.15 所示。

图 5.15　数组的填充

2. fill(int[] a,int fromIndex,int toIndex,int value)方法

该方法将指定的 int 值分配给 int 型数组指定范围中的每个元素。填充的范围从索引 fromIndex（包括）一直到索引 toIndex（不包括）。如果 fromIndex = = toIndex，则填充范围为空。语法格式如下：

```
fill(int[] a,int fromIndex,int toIndex,int value)
```

- ☑ a：要进行填充的数组。
- ☑ fromIndex：要使用指定值填充的第一个元素的索引（包括）。
- ☑ toIndex：要使用指定值填充的最后一个元素的索引（不包括）。
- ☑ value：要存储数组所有元素的值。
- ☑ 返回值：替换元素后的数组。

注意　如果指定的索引位置大于或等于要进行填充的数组的长度，则会报出 ArrayIndexOutOfBoundsException（数组越界异常，关于异常的知识将在后面的章节讲解）异常。

【例 5.15】　在项目中创建 Displace 类，创建一维数组，并通过 fill()方法替换数组元素，最后将数组中的各个元素输出。（**实例位置：光盘\TM\sl\5\6**）

```
import java.util.Arrays;                                    //导入 java.util.Arrays 类
public class Displace {                                     //创建类
    public static void main(String[] args) {                //主方法
        int arr[] = new int[] { 45, 12, 2, 10,1};           //定义并初始化 int 型数组 arr
        Arrays.fill(arr, 1, 3, 8);                          //使用 fill()方法对数组进行初始化
        for (int i = 0; i < arr.length; i++) {              //循环遍历数组中元素
            System.out.println("第" + i + "个元素是：" + arr[i]);  //将数组中的每个元素输出
        }
    }
}
```

运行结果如图 5.16 所示。

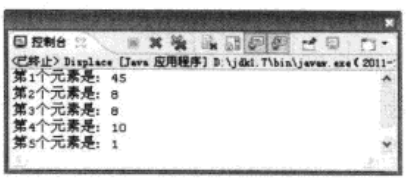


图 5.16　替换数组中元素

5.4.3　对数组进行排序

通过 Arrays 类的静态 sort()方法可实现对数组排序，sort()方法提供了许多种重载形式，可对任意类型数组进行升序排序。语法格式如下：

```
Arrays.sort(object)
```

☑ object：指进行排序的数组名称。

☑ 返回值：排序后的数组。

【例 5.16】 在项目中创建 Taxis 类，在主方法中创建一维数组，并将数组排序后输出。（**实例位置：光盘\TM\sl\5\7**）

```
import java.util.Arrays;                                    //导入 java.util.Arrays 类
public class Taxis {                                        //创建类
    public static void main(String[] args) {                //主方法
        int arr[] = new int[] { 23, 42, 12, 8,5,10 };       //声明数组
        System.out.println("原一维数组是：");
        for (int i = 0; i < arr.length; i++) {              //循环遍历数组
            System.out.print(arr[i]+"   ");                 //输出原来数组
        }
        Arrays.sort(arr);                                   //将数组进行排序
        System.out.println("\n 升序排列后的数组是：");
        for (int i = 0; i < arr.length; i++) {              //循环遍历排序后的数组
            System.out.print(arr[i]+"   ");                 //将排序后数组中的各个元素输出
        }
    }
}
```

运行结果如图 5.17 所示。

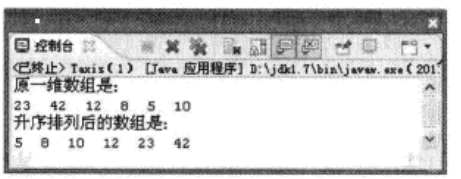


图 5.17 数组的排序

说明 Java 语言中的 String 类型数组的排序算法是根据字典编排顺序排序的，因此数字排在字母前面，大写字母排在小写字母前面。

5.4.4 复制数组

Arrarys 类的 copyOf()方法与 copyOfRange()方法可实现对数组的复制。copyOf()方法是复制数组至指定长度，copyOfRange()方法则将指定数组的指定长度复制到一个新数组中。

1. copyOf()方法

该方法提供了很多种重载形式，来满足不同类型数组的复制。语法格式如下：

```
copyOf(arr,int newlength)
```

☑ arr：要进行复制的数组。

☑ newlength：int 型常量，指复制后的新数组的长度。如果新数组的长度大于数组 arr 的长度，则用 0 填充（根据复制数组的类型来决定填充的值，整型数组用 0 填充，char 型数组则会使用 null 来填充）；如果复制后的数组长度小于数组 arr 的长度，则会从数组 arr 的第一个元素开始截取至满足新数组长度为止。

☑ 返回值：复制后的数组。

【例 5.17】　在项目中创建 Cope 类，在主方法中创建一维数组，实现将此数组复制得到一个长度为 5 的新数组，并将新数组输出。（**实例位置：光盘\TM\sl\5\8**）

```
import java.util.Arrays;
public class Cope {                                          //创建类
    public static void main(String[] args) {                 //主方法
        int arr[] = new int[] { 23, 42, 12 };                //定义数组
        System.out.println("复制后的数组是：");
        for (int i = 0; i < arr.length; i++) {               //循环遍历数组
            System.out.print(arr[i]+"   ");                  //将原来数组输出
        }
        int newarr[] = Arrays.copyOf(arr, 5);                //复制数组 arr
        System.out.println("\n 复制后的数组是：");
        for (int i = 0; i < newarr.length; i++) {            //循环变量复制后的新数组
            System.out.print(newarr[i]+"   ");               //将新数组输出
        }
    }
}
```

运行结果如图 5.18 所示。

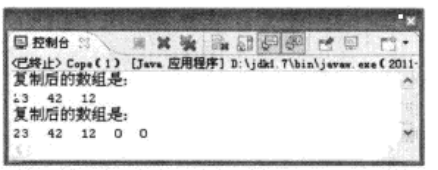


图 5.18　复制数组

2. copyOfRange()方法

该方法同样提供了多种重载形式。语法格式如下：

```
copyOfRange(arr,int formIndex,int toIndex)
```

☑ arr：要进行复制的数组对象。

☑ formIndex：指定开始复制数组的索引位置。formIndex 必须在 0 至整个数组的长度之间。新数组包括索引是 formIndex 的元素。

☑ toIndex：要复制范围的最后索引位置。可以大于数组 arr 的长度。新数组不包括索引是 toIndex 的元素。

☑ 返回值：复制指定位置后的数组。

【例 5.18】 在项目中创建 Repeat 类，在主方法中创建一维数组，并将数组中索引位置为 0～3 之间的元素复制到新数组中，最后将新数组输出。（**实例位置：光盘\TM\sl\5\9**）

```
import java.util.Arrays;
public class Repeat {                                          //创建类
    public static void main(String[] args) {                   //主方法
        int arr[] = new int[] { 23, 42, 12, 84, 10 };          //定义数组
        System.out.println("原来的数组是：");
        for (int i = 0; i < arr.length; i++) {
            System.out.print(arr[i]+"  ");                     //输出原来的数组
        }
        int newarr[] = Arrays.copyOfRange(arr, 0, 3);          //复制数组
        System.out.println("\n 将数组下标 0～3 复制到新数组中：");
        for (int i = 0; i < newarr.length; i++) {              //循环遍历复制后的新数组
            System.out.print(newarr[i]+"  ");                  //将新数组中的每个元素输出
        }
    }
}
```

运行结果如图 5.19 所示。

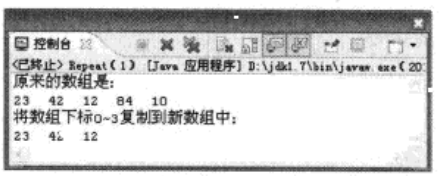


图 5.19 复制指定位置的数组

5.4.5 范例 5：对比一维、二维数组所占内存

很多程序员认为，存储相同的数据，一维数组和二维数组所占用的内存空间基本相同，这种认识是错误的，下面通过实例进行说明。运行结果如图 5.20 所示。（**实例位置：光盘\TM\sl\5\10**）

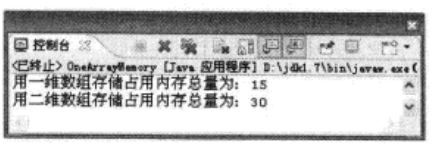


图 5.20 对比一、二维数组所占内存

在项目中创建 OneArrayMemory 类，在类的主方法中使用一维和二维数组存储相同的数据，在控制台输出它们所占的内存，并加以比较。代码如下：

```
public class OneArrayMemory {
    public static void main(String[] args) {
        int num1 = 1024 * 1024 * 2;                            //数组中元素的个数
        int[] arr1 = new int[num1];                            //创建 int 型一维数组
        for (int i = 0; i < arr1.length; i++) {
            arr1[i] = i;                                       //为数组元素赋值
        }
        //获得占用内存总数，并将单位转换为 MB
        long memory1 = Runtime.getRuntime().totalMemory() / 1024 / 1024;
```

```
        System.out.println("用一维数组存储占用内存总量为：" + memory1);
        int num2 = 1024 * 1024;                          //数组中元素的个数
        int[][] arr2 = new int[num2][2];                 //创建 int 型二维数组
        for (int i = 0; i < arr2.length; i++) {
            arr2[i][0] = i;                              //为数组元素赋值
            arr2[i][1] = i;                              //为数组元素赋值
        }
        //获得占用内存总数，并将单位转换为 MB
        long memory2 = Runtime.getRuntime().totalMemory() / 1024 / 1024;
        System.out.println("用二维数组存储占用内存总量为：" + memory2);
    }
}
```

5.4.6　范例 6：使用直接插入排序法排序

插入排序的基本思想是将要排序的值逐个插入到其所适合的位置，其后的元素都向后移一个位置。如 1、3 和 2 排序，先是 1，然后 3 插入到 1 后面，最后 2 插入到 1 后面，3 往后移一位。运行结果如图 5.21 所示。（**实例位置：光盘\TM\sl\5\11**）

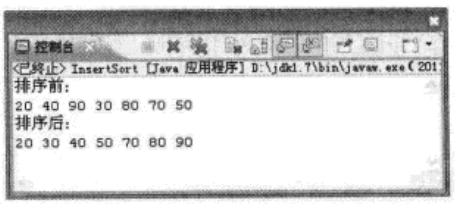


图 5.21　使用直接插入排序法排序

在项目中创建 InsertSort 类，在类的主方法中创建一维数组，然后将数组中的元素进行排序，并在控制台将结果输出。代码如下：

```
public class InsertSort {
    public static void main(String[] args) {
        int[] array=new int[]{20,40,90,30,80,70,50};
        System.out.println("排序前：");
        for (int i = 0; i < array.length; i++) {            //初始化数组元素
            System.out.print(array[i]+" ");                 //输出数组元素
        }
        int tmp;                                            //定义临时变量
        int j;
        for (int i = 1; i < array.length; i++) {
            tmp = array[i];                                 //保存临时变量
            for (j = i - 1; j >= 0 && array[j] > tmp; j--) {
                array[j + 1] = array[j];                    //数组元素交换
            }
            array[j + 1] = tmp;                             //在排序位置插入数据
        }
        System.out.println("\n 排序后：");
        for (int i = 0; i < array.length; i++) {            //初始化数组元素
            System.out.print(array[i]+" ");                 //输出数组元素
        }
    }
}
```

5.5 经典范例

5.5.1 经典范例 1：使用冒泡排序法排序

视频讲解：光盘\TM\lx\5\使用冒泡排序法排序.exe

冒泡排序的基本思想是对比相邻的元素值，如果满足条件就交换元素值，把较小的元素移动到数组前面，把大的元素移动到数组后面（也就是交换两个元素的位置），这样数组元素就像气泡一样从底部上升到顶部。运行结果如图 5.22 所示。**（实例位置：光盘\TM\sl\5\12）**

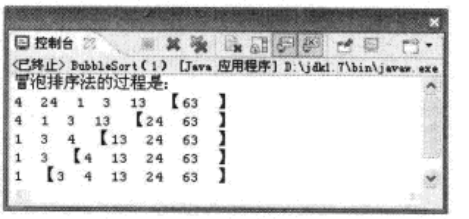


图 5.22 使用冒泡排序法排序

在项目中创建 BubbleSort 类，在类的主方法中创建一维数组，使用冒泡排序法进行排序，并将结果在控制台输出。代码如下：

```
public class BubbleSort {
    public static void main(String[] args) {
        int[] array=new int[]{63,4,24,1,3,13};                    //声明并初始化一维数组
        System.out.println("冒泡排序法的过程是：");
        for (int i = 1; i < array.length; i++) {
            for (int j = 0; j < array.length - i; j++) {          //比较相邻两个元素，较大的数往后冒泡
                if (array[j] > array[j + 1]) {
                int temp = array[j];                              //把第一个元素值保持到临时变量中
                array[j] = array[j + 1];                          //把第二个元素值保存到第一个元素单元中
                array[j + 1] = temp;                              //把第一个元素原值保存到第二个元素中
                }
                System.out.print(array[j] + "  ");                //把排序后的数组元素显示到文本域中
            }
            System.out.print("【");
            for (int j = array.length - i; j < array.length; j++) {
            System.out.print(array[j] + "  ");                   //把排序后的数组元素显示到文本域中
            }
            System.out.println("】");
        }
    }
}
```

说明

冒泡排序法的算法比较简单，排序的结果稳定，缺点是时间效率不太高。

5.5.2　经典范例 2：输出九宫格

视频讲解：光盘\TM\lx\5\输出九宫格.exe

九宫格是大家都喜欢玩的一种游戏，在一个三维方阵的 9 个元素中分别填入 1～9 中的 9 个数，使得每一行、列和对角线上 3 个数的和都等于 15。运行结果如图 5.23 所示。（**实例位置：光盘\TM\sl\5\13**）

图 5.23　输出九宫格

在项目中创建 NineTable 类，在类的主方法中创建三维方阵，按照九宫格的玩法填入 1～9，在控制台将结果输出。代码如下：

```
public class NineTable {
    public static void main(String[] args) {
        int arr[][] = new int[3][3];                    //创建一个三阶方阵
        int a = 2;                                      //第 3 行的行下标
        int b = 3/2;                                    //第 2 列的列下标
        for(int i=1;i<=9;i++){                          //给数组赋值
            arr[a++][b++] = i;
            if(i%3==0){                                 //如果 i 是 3 的倍数
                a = a-2;
                b = b-1;
            }else{                                      //如果 i 不是 3 的倍数
                a = a%3;
                b = b%3;
            }
        }
        System.out.println("输出九宫格：");              //遍历输出九宫格
        for(int i=0;i<3;i++){
            for(int j=0;j<3;j++){
                System.out.print(arr[i][j]+"  ");
            }
            System.out.print("\n");
        }
    }
}
```

5.6　本 章 小 结

本章向读者介绍的是数组的创建及使用方法。需要读者注意的是，数组的下标是从 0 开始，最后一个元素的下标总是“数组名.length-1”。本章的重点是遍历数组以及使用 Arrays 类中的各种方法对数组进行操作，如填充替换数组、复制数组等。此外 Arrays 类还提供了其他操作数组的方法，有兴趣的

读者可以查阅相关资料。

5.7 实战练习

1．编写 Java 程序，创建一维数组 arr[]，并将其遍历输出。（**答案位置：光盘\TM\sl\5\14**）

2．编写 Java 程序，创建一维数组 arr[]，将数组中最大的数输出。（**答案位置：光盘\TM\sl\5\15**）

3．编写 Java 程序，创建二维数组 arr[][]，将二维数组中所有元素的和输出。（**答案位置：光盘\TM\sl\5\16**）

第 6 章

字符串

（视频讲解：48 分钟）

字符串即 String 类，是 Java 中一个比较特殊的类，它不是 Java 的基本数据类型之一，却可以像基本数据类型一样使用，像声明和初始化等操作都是相同的，并且使用非常频繁，是程序经常处理的对象，如果处理不好就会影响程序运行的效率，所以学好 String 类的用法是很重要的。本章讲述了 String 类的创建方式等操作，都是 String 类学习的重点，供读者翻阅学习，一定要仔细阅读，并熟练掌握这些操作方法。

通过阅读本章，您可以：

- 掌握字符串的创建方式
- 掌握字符串的常用操作
- 掌握字符串的格式化
- 理解正则表达式
- 掌握字符串生成器的用法

PDG

6.1 创建字符串

视频讲解：光盘\TM\lx\6\创建字符串.exe

String 类即字符串类型，并不是 Java 的基本数据类型，但可以像基本数据类型一样使用，用双引号括起来进行声明。在 Java 中用 String 类的构造方法来创建字符串变量。常用的构造方法如下：

1. String()

一个 String 对象，可表示一个空字符序列。

【例 6.1】 使用 String()方法初始化一个新创建的 String 对象。

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

2. String(char a[])

用一个字符数组 a 创建 String 对象。

【例 6.2】 使用 String(char a[])方法创建 String 对象。

```
char a[] = {'g','o','o','d'};
String s = new String(a);
```

等价于 ⇒

```
String s = new String("good")
```

3. String(char a[], int offset, int length)

提取字符数组 a 中的一部分创建一个字符串对象。参数 offset 表示开始截取字符串的位置，length 表示截取字符串的长度。

【例 6.3】 使用 String(char a[], int offset, int length)方法创建 String 对象。

```
char a[]={'s','t','u','d','e','n','t'};
String s = new String(a,2,4);
```

等价于 ⇒

```
String s = new String("uden");
```

6.2 字符串操作

视频讲解：光盘\TM\lx\6\字符串操作.exe

对于已经声明的字符串，可以对其进行相应的操作，String 类中包含很多种方法，程序员可以使用这些方法对字符串进行操作以满足实际编程的需要。

6.2.1 字符串连接

字符串连接是对字符串的最基本的操作之一，主要有连接多个字符串和连接其他数据类型两种不

同的操作，下面分别介绍这两种操作。

1. 多个字符串连接

连接多个字符串时，在每两个连接的字符串之间用“+”相连，“+”就是字符串的连接符，连接之后生成一个新的字符串。

【例 6.4】 使用“+”运算符完成对多个字符串的连接。

```
String s1 = new String("hello");          //声明 String 对象 s1
String s2 = new String("word");           //声明 String 对象 s2
String s = s1 + " " + s2;                 //将对象 s1 和 s2 连接后的结果赋值给 s
```

将 s1、空格和 s2 进行相连，运行结果是“hello word”。

注意 Java 中的字符串不能分开在两行中写。如果一个字符串太长，需要将这个字符串分在两行上书写，此时可以使用“+”将两个字符串连起来，并在加号处换行。

2. 连接其他数据类型

在字符串和其他数据类型连接时，同样使用“+”连接符，连接之后的返回值是字符串。

【例 6.5】 在项目中创建 Link 类，在主方法中创建数值型变量，实现将字符串与整型、浮点型变量相连的结果输出。

```
public class Link {                                    //创建类
    public static void main(String args[]) {           //主方法
        int booktime = 4;                              //声明的 int 型变量 booktime
        float practice = 2.5f;                         //声明的 float 型变量 practice
        System.out.println("我每天花费"+booktime+"小时看书；"+practice+
                "小时上机练习");                        //字符串与整型、浮点型变量相连，将结果输出
    }
}
```

运行结果如图 6.1 所示。

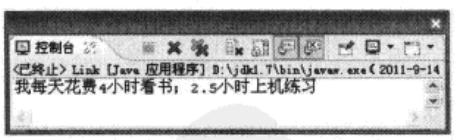


图 6.1 输出字符串连接后的运行结果

这里 booktime 和 practice 都不是字符串，当其他数据类型与字符串相连时会自动将其转换成字符串类型，然后进行连接。

注意 只要“+”运算符的一个操作类型是字符串，编译器就会将另外的操作类型转换成字符串形式，所以应谨慎地将其他数据类型与字符串相连，以免得到意想不到的结果。

6.2.2 获取字符串信息

字符串变量是一个对象，可以通过相应方法获取字符串的有效信息，如获取字符串的长度、获取指定字符的索引位置等。

1. 获取字符串长度

使用 String 类的 length()方法可获取声明的字符串对象的长度。语法格式如下：

```
str.length();
```

☑ str：指定的字符串对象。

☑ 返回值：返回指定字符串的长度。

【例 6.6】 使用 length()方法得出字符串的长度。

```
String s = "We are students";                          //声明字符串 s
System.out.println("字符串的长度是："+s.length());      //输出字符串长度
```

获取字符串“We are students”的长度，运行结果是“字符串的长度是：15”，很明显在计算字符串长度时字符串中的空格也被计算在内了，这点应该注意。

2. 获取指定字符的索引位置

String 类提供了 indexOf()和 lastIndexOf()方法来获取指定字符的索引位置，不同的是，indexOf()方法返回的是搜索的字符首次出现位置的索引，而 lastIndexOf()方法返回的是搜索的字符最后出现位置的索引。

（1）indexOf()方法

调用字符串的 indexOf()方法搜索 s 时，会从字符串的起始位置依次搜索，返回 s 首次出现位置的索引，如果没有检索到字符串 s，返回-1。语法格式如下：

```
str.indexOf(substr)
```

☑ str：任意字符串对象。

☑ substr：要搜索的字符。

☑ 返回值：返回字符序列中第一次出现该字符的索引。

【例 6.7】 在项目中创建 Select 类，在主方法中创建 String 对象，使用 indexOf()方法查找字符 s 在字符串 s 中的索引位置，并将结果输出。

```
public class Select {                                          //创建类
    public static void main(String[]args) {                    //主方法
        String s = "We are students";                          //声明字符串 s
        System.out.println("字符 s 在字符串 s 中的位置是：" +s.indexOf("s"));
    }
}
```

运行结果如图 6.2 所示。

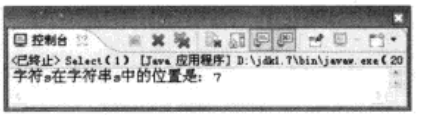


图 6.2　使用 indexOf()方法查找字符的位置

运行结果是 7，说明在计算机中 String 对象的索引是从 0 开始的，这点需要注意。

（2）lastIndexOf()方法

调用字符串的 lastIndexOf()方法搜索 s 时，会从字符串的起始位置依次搜索，返回 s 最后出现位置的索引，如果没有检索到字符串 s，返回-1。语法格式如下：

```
str. lastIndexOf(substr)
```

☑　str：任意字符串对象。

☑　substr：要搜索的字符串。

☑　返回值：返回字符序列中最后出现的该字符的索引。

【例 6.8】　使用 lastIndexOf()方法查找字符 s 在字符串 s 中的索引位置。

```
String s = "We are students";                                    //声明字符串 s
System.out.println("字符 s 在字符串 s 中的位置是：");
System.out.println(s. lastIndexOf ("s"));
```

运行的结果是“字符 s 在字符串 s 中的位置是：14”，表明此方法输出字符 s 在字符串 s 中最后一次出现的位置。

注意　如果 lastIndexOf()方法中的参数是空字符串" "，则返回的结果与调用该字符串的 length()方法的返回结果相同。

说明　图 6.3 分析了字符串 s 的下标存储状态。

图 6.3　字符串 s 下标示意图

3．获取指定索引位置的字符

使用 String 类的 charAt()方法可获取指定索引处的字符，返回字符的索引。语法格式如下：

```
str.charAt(int index)
```

☑　str：任意字符串。

☑　index：整型值，用于指定要返回字符的下标。

☑ 返回值：返回指定索引位置的字符。

【例 6.9】 使用 charAt()方法查找字符串“hello word”的索引位置是 6 的字符。

```
String s = "hello word";                                    //定义字符串 s
char mychar2 = s.charAt(6);                                 //将字符串 s 中索引位置是 6 的字符返回
System.out.println("字符串 s 中索引位置是 6 的字符为："+mychar2); //输出信息
```

运行结果是“字符串 s 中索引位置是 6 的字符为：w”。

6.2.3 去除字符串中的空格

去掉字符串中的空格有两种，一是去除字符串的前导、尾部空格，二是去除字符串中的所有空格，可以用不同的方法来实现。

1．去除字符串的前导、尾部空格

trim()方法用于去除字符串的前导、尾部空格。语法格式如下：

```
str.trim()
```

☑ str：任意的字符串对象。

☑ 返回值：去除前导和尾部空格后的字符串。

【例 6.10】 在项目中创建 Blak 类，在主方法中创建 String 对象，将字符变量原来的长度与去掉前导和尾部空格后的长度输出。

```
public class Blak {                                         //创建类
    public static void main(String args[]) {                //主方法
        String s1 = "     Java class     ";                 //声明字符串 s，前后各有几个空格
        String s2 = s1.trim();
        System.out.println("字符串原来的长度："+s1.length());   //将 s 原来的长度输出
        System.out.println("去除空格后的长度："+s2.length()); //输出 s 去掉前后空格后的结果
        System.out.println("去除空格后的字符串是："+s2);
    }
}
```

运行结果如图 6.4 所示。

结果显示此方法去除了字符串的前导空格和尾部空格，而中间的一个空格并没有去除。

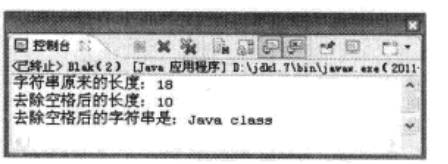


图 6.4 去除空格

2．去除字符串中的所有空格

去除字符串中所有空格可以用 StringTokenizer()和 replaceAll()方法来实现。

（1）StringTokenizer()方法

该方法利用分割和追加实现去除空格的目的。语法格式如下：

```
StringTokenizer(String str, String delim)
```

☑ str：要去除空格的字符串。

☑ delim：从字符串 str 中去除的字符串。

☑ 返回值：去掉所有空格后的字符串。

【例 6.11】 在项目中创建 StringTo 类，使用 StringTokenizer()方法去除字符串中所有空格，并将结果输出。（**实例位置：光盘\TM\sl\6\1**）

```
import java.util.StringTokenizer;                                   //包声明
public class StringTo {
    public static void main(String[] args) {
        String text = "   We are students   ";                      //定义包含空格的字符串
        System.out.println("原字符串是：");
         System.out.println(text);
        StringTokenizer st = new StringTokenizer(text," ");         //使用空格来分割字符串
        StringBuffer sb = new StringBuffer();                       //定义字符串缓存对象
        int i = 1;
        while(st.hasMoreTokens()){                                  //判断字符串中是否还有空格
            i++;
            sb.append(st.nextToken());                              //向字符串生成器中追加内容
        }
        System.out.println("去掉字符串中所有空格之后的字符串是：");
        System.out.println(sb.toString());
    }
}
```

运行结果如图 6.5 所示。

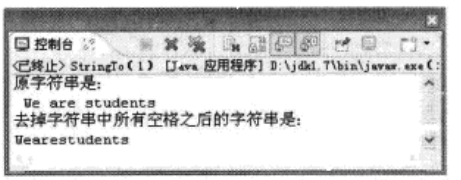


图 6.5 使用 StringTokenizer()方法去除所有空格

（2）replaceAll()方法

该方法是将空格" "用""替换，以达到去除字符串中空格的目的。语法格式如下：

```
str.replaceAll(String regex, String replacement)
```

☑ str：要去除空格的字符串。

☑ regex：str 中想要被替换的字符串。

☑ replacement：新字符串，用来替换每个 regex。

☑ 返回值：去掉所有空格后的字符串。

【例 6.12】 使用 replaceAll()方法去除字符串中的所有空格。

```
String s = " J a v a 编 程 词 典 ";                    //定义带有空格的字符串
s = s.replaceAll(" ", "");                              //去除字符串中的所有空格
System.out.println("去除空格后的字符串为："+s);          //将经过处理后的字符串输出
```

运行结果是“去除空格后的字符串为：Java 编程词典”。

6.2.4 字符串替换

字符串替换就是新字符串替换掉原字符串中指定位置的字符串，生成一个新的字符串，通过replace()和 replaceFirst()等方法可以实现。

1．replace()方法

该方法用于替换所有与指定字符串相匹配的字符串。语法格式如下：

```
str.replace(String regex, String replacement)
```

☑ regex：str 中想要被替换的字符串。

☑ replacement：新字符串，用来替换每个 regex。

☑ 返回值：返回替换后的字符串。

【例 6.13】 使用 replace()方法把字符串“bad bad study”中的 bad 全部替换成 good。

```
String s = "bad bad study";                          //定义字符串 s
String news = s.replace("bad", "good");              //将 s 中的 bad 替换成 good 并返回字符串
System.out.println("替换后的字符串是："+news);        //将字符串 news 输出
```

运行结果是“替换后的字符串是：good good study”，replace()方法把两个 bad 都替换成了 good。

2．replaceFirst()方法

该方法用于替换第一个出现的指定字符串，而后面出现的指定字符串则不会被替换。语法格式如下：

```
str.replaceFirst(String regex, String replacement)
```

☑ regex：str 中想要被替换的字符串。

☑ replacement：新字符串，用来替换第一个 regex。

☑ 返回值：返回替换后的字符串。

【例 6.14】 使用 replaceFirst()方法把字符串“bad bad study”中的 bad 替换成 good。

```
String s = "bad bad study";                          //用于测试的字符串
s = s.replaceFirst("bad","good");                    //替换字符串
System.out.println("替换后的字符串是："+s);          //输出替换后的结果
```

运行结果是“替换后的字符串是：good bad study”，replaceFirst()方法只把第一个 bad 替换成了 good，而第二个 bad 并没有被替换。

注意 要替换的字符 regex 的大小写有严格的要求，需要和原字符串保持一致，否则将不能成功替换。

6.2.5 判断字符串

判断字符串主要分判断字符串是否相等、判断字符串是否以指定的字符串开始和结尾两种，可以通过不同的方法实现。

1. 判断字符串是否相等

判断字符串是否相等有 equals()和 equalsIgnoreCase()两种方法，而又有各自的规则，使用不好就会出错，所以一定要区分好什么时候用哪种方法。

（1）equals()方法

使用 equals()方法对字符串进行比较时是严格区分大小写的，在此条件下，如果两个字符串仍具有相同的字符和长度，则返回 true，否则返回 false。语法格式如下：

```
str.equals(String otherstr)
```

☑ str：参与比较的一个字符串对象。

☑ otherstr：参加比较的另一个字符串对象。

☑ 返回值：返回 boolean 类型。

（2）equalsIgnoreCase()方法

使用 equalsIgnoreCase()方法是在忽略大小写的情况下比较两个字符串是否相等，如果两个字符串具有相同的字符和长度，则返回 true，否则返回 false。语法格式如下：

```
str.equalsIgnoreCase(String otherstr)
```

☑ str：参与比较的一个字符串。

☑ otherstr：参加比较的另一个字符串对象。

☑ 返回值：返回 boolean 类型。

【例 6.15】 在项目中创建 Opinion 类，在主方法中创建 String 型变量，用 3 种方法实现判断两个字符串是否相等，将结果输出，通过输出的结果可以看出“==”、equals()和 equalsIgnoreCase()方法的区别。（**实例位置：光盘\TM\sl\6\2**）

```
public class Opinion {                                    //创建类
    public static void main(String args[]) {              //主方法
        String s1 = new String("I am a student");         //声明字符串 s1
        String s2 = new String("I am a student");         //声明字符串 s2
        String s3 = new String("I AM A STUDENT");         //声明字符串 s3
        String s4 = s1;                                   //将 s1 的值赋给 s4

        boolean b1 = (s1 == s2);                          //用“==”比较 s1 和 s2
        boolean b2 = (s1 == s4);                          //用“==”比较 s1 和 s4
        boolean b3 = s1.equals(s2);                       //用 equals()方法比较 s1 和 s2
        boolean b4 = s1.equals(s3);                       //用 equals()方法比较 s1 和 s3
        boolean b5 = s1.equalsIgnoreCase(s2);             //用 equalsIgnoreCase()方法比较 s1 和 s2
        boolean b6 = s1.equalsIgnoreCase(s3);             //用 equalsIgnoreCase()方法比较 s1 和 s3
```

```
        System.out.println("s1 == s2:"+b1);                    //输出 b1
        System.out.println("s1 == s4:"+b2);                    //输出 b2
        System.out.println("s1 equals s2:"+b3);                //输出 b3
        System.out.println("s1 equals s3:"+b4);                //输出 b4
        System.out.println("s1 equalsIgnoreCase s2:"+b5);      //输出 b5
        System.out.println("s1 equalsIgnoreCase s3:"+b6);      //输出 b6
    }
}
```

运行结果如图 6.6 所示。

注意 比较运算符“==”比较的是内存位置，不适宜拿来比较字符串，但整型等可以用比较运算符比较；equals()方法比较的两个字符串内容必须完全一样；equalsIgnoreCase()方法在忽略大小写的情况下内容必须一样。实例中各变量在内存中的储存状态如图 6.7 所示。

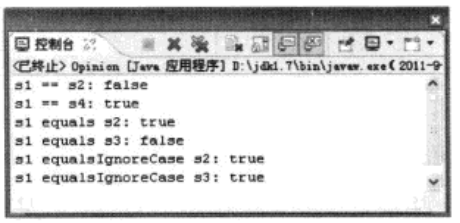


图 6.6　3 种方式判断字符串相等

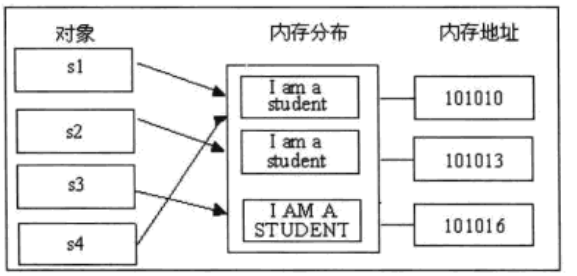


图 6.7　变量的内存结构

2. 判断字符串的开始与结尾

String 类的 startsWith()与 endsWith()方法分别用于判断字符串是否以指定的内容开始和结尾，它们的返回值都为 boolean 类型。

（1）startsWith()方法

该方法用于判断当前字符串对象是否以参数指定的字符串开始。语法格式如下：

```
str.startsWith(String prefix)
```

☑ prefix：指作为前缀的字符。

☑ 返回值：返回 boolean 类型。

（2）endsWith()方法

该方法用于判断当前字符串对象是否以参数指定的字符串结尾。语法格式如下：

```
str.endsWith(String suffix)
```

☑ suffix：指作为后缀的字符串。

☑ 返回值：返回 boolean 类型。

【例 6.16】 在项目中创建 StartOrEnd 类，在主方法中创建 String 型变量，判断变量是否以指定的字符串开始和结尾。（**实例位置：光盘\TM\sl\6\3**）

```
public class StartOrEnd {                                          //创建类
    public static void main(String args[]) {                       //主方法
        String num1 = "22045612";                                  //定义字符串 num1
        String num2 = "21304578";                                  //定义字符串 num2

        boolean b1 = num1.startsWith("22");                        //判断字符串 num1 是否以'22'开头
        boolean b2 = num1.endsWith("78");                          //判断字符串 num1 是否以'78'开头
        boolean b3 = num2.startsWith("22");                        //判断字符串 num2 是否以'22'开头
        boolean b4 = num2.endsWith("78");                          //判断字符串 num2 是否以'78'开头

        System.out.println("字符串 num1 是以'22'开始的吗？"+b1);
        System.out.println("字符串 num1 是以'78'结束的吗？"+b2);    //输出信息
        System.out.println("字符串 num2 是以'22'开始的吗？"+b3);
        System.out.println("字符串 num2 是以'78'结束的吗？"+b4);
    }
}
```

运行结果如图 6.8 所示。

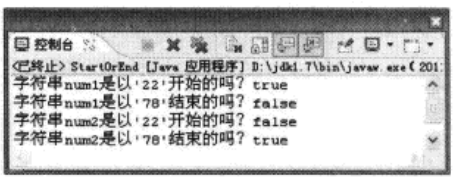


图 6.8 判断字符串的开始和结尾

6.2.6 字母大小写转换

String 类提供了字母大小写转换的方法，即 toLowerCase() 和 toUpperCase()方法。

1. toLowerCase()方法

该方法将字符串中的大写字母转换为小写字母，如果原来就是小写字母，则不改变，最后返回一个新的字符串，字符长度与原字符长度相同。语法格式如下：

```
str.toLowerCase()
```

- ☑ str：要进行转换的字符串。
- ☑ 返回值：全换成小写后的字符串。

2. toUpperCase()方法

该方法将字符串中的小写字母转换为大写字母，如果原来就是大写字母，则不改变，最后返回一个新的字符串，字符长度与原字符长度相同。语法格式如下：

```
str.toUpperCase()
```

- ☑ str：要进行转换的字符串。
- ☑ 返回值：全换成大写后的字符串。

说明

使用 toLowerCase()和 toUpperCase()方法进行大小写转换时，数字或非字符不受影响。

【例 6.17】 在项目中创建 UpAndLower 类，在主方法中创建 String 型变量，实现字符串的大小写转换，并将转换后的结果输出。

```
public class UpAndLower {                                    //创建类
    public static void main(String args[]) {                 //主方法
        String s = new String("abc DEF");                    //创建的字符串 s
        String news1 = s.toLowerCase();                      //使用 toLowerCase()方法进行小写转换
        String news2 = s.toUpperCase();                      //使用 toUpperCase()方法进行大写转换
        System.out.println("全部转换成小写字母后的字符串是："+news1);
        System.out.println("全部转换成大写字母后的字符串是："+news2);
    }
}
```

运行结果如图 6.9 所示。

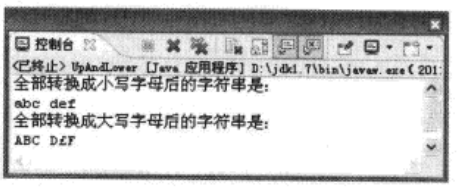


图 6.9 字母大小写转换

6.2.7 字符串分割

split()方法根据指定的分割符对字符串进行分割，并将分割后的结果存放在字符串数组中。split()方法提供了两种重载形式。

1. split(String sign)方法

该方法根据指定的分割符对字符串进行完全分割。语法格式如下：

```
str.split(String sign)
```

☑ sign：分割字符串的分割符，也可以使用正则表达式。

☑ 返回值：根据分隔符分割后的字符串。

【例 6.18】 在项目中创建 Division 类，在主方法中创建 String 型变量，将字符变量进行分割，并将分割后的结果输出。

```
public class Division {                                      //创建类
    public static void main(String args[]) {                 //主方法
        String s = new String("abc,def,ghi,gkl");            //定义的字符串 s
        String[] news = s.split(",");                        //使用 split()方法拆分字符串，返回字符串数组
        System.out.println("按分隔字符分割后的字符串是：");
        for (int i = 0; i < news.length; i++) {              //使用 for 循环遍历字符数组
        System.out.println(news[i]);                         //输出信息
        }
    }
}
```

运行结果如图 6.10 所示。

2. split(String sign,int limit)方法

该方法根据给定的分割符对字符串进行拆分，并限定拆分的份数。语法格式如下：

```
str.split(String sign,int limit)
```

☑ sign：分割字符串的分割符，也可以使用正则表达式。

☑ limit：限制的分割份数。

☑ 返回值：按照分隔符分割指定次数后的字符串。

【例 6.19】 在项目中创建 Division2 类，在主方法中创建 String 型变量，将字符变量进行分割，并将分割后的结果输出。

```
public class Division2 {                                    //创建类
    public static void main(String args[]) {                //主方法
        String s = new String("abc,def,ghi,gkl");           //定义的字符串 s
        String[] news2 = s.split(",",2);                    //拆分字符串并限定份数
        for(int j = 0;j<news2.length;j++){                  //循环遍历字符数组
        System.out.println("按分隔字符分割一次后的字符串是：");
        System.out.println(news2[j]);                       //输出信息
        }
    }
}
```

运行结果如图 6.11 所示。

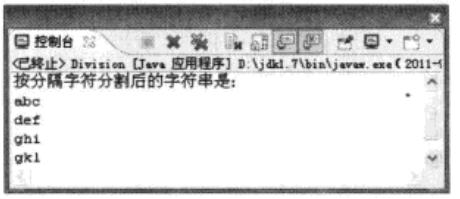


图 6.10 使用 split(String sign)方法分割字符串

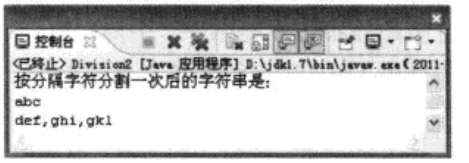


图 6.11 使用 split(String sign,int limit)方法分割字符串

说明 没有统一的对字符进行分割的符号。如果想定义多个分割符，可使用符号“|”。例如，“,|=”表示分割符分别为“,”和“=”。

6.2.8 范例 1：根据指定分割符把字符串分行

在很多时候需要对一篇文章按照指定分割符进行分行，按 Enter 键就比较麻烦了，尤其是文章较长时，这时候就需要编写程序进行分行，分行后字符串存储在字符串数组中。运行结果如图 6.12 所示。（实例位置：光盘\TM\sl\6\4）

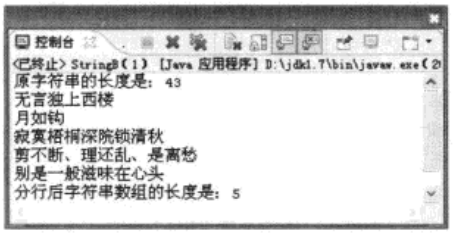


图 6.12　根据标点对字符串分行

在项目中创建 Division 类，使用 split(String sign)方法，根据逗号和句号进行分割，并输出字符串长度和字符数组维数。代码如下：

```
public class Division {                                          //创建类
    public static void main(String args[]) {                     //主方法
        String s =  "无言独上西楼，月如钩，寂寞梧桐深院锁清秋。" +
                    "剪不断、理还乱、是离愁，别是一般滋味在心头。";
        System.out.println("原字符串的长度是："+s.length());
        String[] news = s.split("，|。");                        //使用 split()方法拆分字符串
        for (int i = 0; i < news.length; i++) {                  //使用 for 循环遍历字符数组
        System.out.println(news[i]);                             //输出字符数组
        }
        System.out.println("分行后字符串数组的长度是："+news.length);  //输出字符数组长度
    }
}
```

6.2.9　范例 2：判断字符串是否是数字格式

字符串的内容可以是数字、字母等任何形式，下面通过程序来判断字符串 12312312 的内容是否是数字。运行结果如图 6.13 所示。（**实例位置：光盘\TM\sl\6\5**）

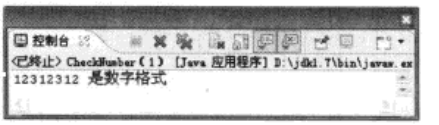


图 6.13　判断字符串是否是数字格式

在项目中创建 CheckNumber 类，将字符串拆分成字符数组，判断每个字符是否是数字，如果全是数字字符串就是数字格式，否则不是。代码如下：

```
public class CheckNumber {
    public static void main(String[] args) {
        String s = "12312312";                          //获取字符串
        if (CheckNumber.IsNumber(s)) {                  //调用方法判断，如果返回 true
            System.out.println(s + " 是数字格式");      //输出是数字格式
        } else {                                        //否则
            System.out.println(s + " 不是数字格式");
        }
    }
    public static boolean IsNumber(String str) {
        char[] c = str.toCharArray();                   //将字符串拆分并赋给字符数组
        for (int i = 0; i < c.length; i++) {            //遍历每一个字符
```

```
            if (Character.isDigit(c[i])) {                    //如果是数字，继续循环
            } else {                                          //否则
                return false;                                 //直接返回 false
            }
        }
        return true;                                          //都是数字则返回 true
    }
}
```

6.3 格式化字符串

视频讲解：光盘\TM\lx\6\格式化字符串.exe

格式化字符串是程序中经常用到的，主要包括日期格式化、时间格式化、日期/时间组合的格式化和常规类型格式化。

6.3.1 格式化方法

在应用程序设计中，经常需要显示时间和日期。format()方法通过给定的特殊转换符作为参数来实现对日期和时间的格式化。format()方法有以下两种重载形式。

1．format(String format,Object…args)方法

该方法使用指定的格式字符串和参数，格式化后的新字符串使用本地默认的语言环境，返回格式化后的字符串。语法格式如下：

```
str.format(String format,Object…args)
```

- ☑ format：格式化转换符。
- ☑ args：格式化字符串中格式说明符引用的参数。如果还有格式说明符以外的参数，则忽略这些额外的参数。此参数的数目是可变的，可以为0。
- ☑ 返回值：一个格式化的字符串。

2．format(Local l,String format,Object…args)方法

该方法使用指定的格式字符串和参数，格式化后的新字符串使用参数所规定的语言环境，返回格式化后的字符串。语法格式如下：

```
str.format(Local l,String format,Object…args)
```

- ☑ l：格式化过程中要应用的语言环境。如果l为null，则不进行本地化。
- ☑ format：格式化转换符。
- ☑ args：格式化字符串中格式说明符引用的参数。如果还有格式说明符以外的参数，则忽略这些

额外的参数。此参数的数目是可变的，可以为 0。

☑ 返回值：一个格式化字符串。

6.3.2 日期格式化

使用 format()方法对日期进行格式化时，会用到日期格式化转换符。常用的日期格式化转换符如表 6.1 所示。

表 6.1 常见的日期格式化转换符

转换符	说明	示例
%te	一个月中的某一天（1～31）	6
%tb	指定语言环境的月份简称	Feb（英文）、二月（中文）
%tB	指定语言环境的月份全称	February（英文）、二月（中文）
%tA	指定语言环境的星期几全称	Monday（英文）、星期一（中文）
%ta	指定语言环境的星期几简称	Mon（英文）、星期一（中文）
%tc	包括全部日期和时间信息	星期二 三月 25 13:37:22 CST 2008
%tY	4 位年份	2008
%tj	一年中的第几天（001～366）	085
%tm	月份	03
%td	一个月中的第几天（01～31）	02
%ty	2 位年份	08

【例 6.20】 在项目中创建 Example 类，在该类的主方法中创建一个 Date 日期类的实例对象 today，利用 format()方法格式化 today 日期对象。（**实例位置：光盘\TM\sl\6\6**）

```
import java.util.Date;                                          //包声明
import java.util.Locale;
public class Example {
    public static void main(String[] args) {
        Date today = new Date();
        String a = String.format(Locale.US, "%tb", today);     //格式化为月份的英文缩写
        String b = String.format(Locale.US, "%tB", today);     //格式化为月份的英文全写
        String c = String.format("%ta", today);                //格式化为星期简称
        String d = String.format("%tA", today);                //格式化为星期全称
        String e = String.format("%tY", today);                //格式化为 4 位的年份值
        String f = String.format("%ty", today);                //格式化为 2 位的年份值
        String g = String.format("%tm", today);                //格式化为 2 位的月份值
        String h = String.format("%td", today);                //格式化为 2 位的日期值
        String i = String.format("%te", today);                //格式化为 1 位的日期值

        System.out.println("格式化后的字符串为月份的英文缩写：" + a);
        System.out.println("格式化后的字符串为月份的英文全写：" + b);
        System.out.println("月格式化后的字符串为星期：" + c);
        System.out.println("格式化后的字符串为星期：" + d);
```

```
        System.out.println("格式化后的字符串为 4 位的年份值：" + e);
        System.out.println("格式化后的字符串为 2 位的年份值：" + f);
        System.out.println("格式化后的字符串为 2 位的月份值：" + g);
        System.out.println("格式化后的字符串为 2 位的日期值：" + h);
        System.out.println("格式化后的字符串为 1 位的日期值：" + i);
    }
}
```

运行结果如图 6.14 所示。

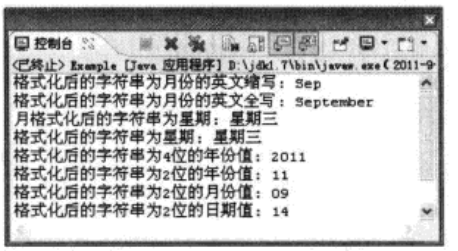


图 6.14　使用 format()方法格式化 today

6.3.3　时间格式化

使用 format()方法对时间进行格式化时，会用到时间格式化转换符。时间格式化转换符要比日期转换符更多、更精确，它可以将时间格式化为时、分、秒、毫秒。常用的时间格式化转换符如表 6.2 所示。

表 6.2　时间格式化转换符

转　换　符	说　　明	示　　例
%tH	2 位数字的 24 时制的小时（00～23）	14
%tI	2 位数字的 12 时制的小时（01～12）	05
%tk	2 位数字的 24 时制的小时（0～23）	5
%tl	2 位数字的 12 时制的小时（1～12）	10
%tM	2 位数字的分钟（00～59）	05
%tS	2 位数字的秒数（00～60）	12
%tL	3 位数字的毫秒数（000～999）	920
%tN	9 位数字的微秒数（000000000～999999999）	062000000
%tp	指定语言环境下上午或下午标记	下午（中文）、pm（英文）
%tz	相对于 GMT RFC 82 格式的数字时区偏移量	+0800
%tZ	时区缩写形式的字符串	CST
%ts	1970-01-01 00:00:00 至现在经过的秒数	1206426646
%tQ	1970-01-01 00:00:00 至现在经过的毫秒数	1206426737453

【例 6.21】　在项目中创建 GetTime 类，在主方法中创建 String 型变量，实现时间各种格式的转换，并在控制台输出结果。（**实例位置：光盘\TM\sl\6\7**）

```
import java.util.Date;
public class GetTime {
    public static void main(String[] args) {
        Date date=new Date();
        System.out.println("默认时间格式："+date);
        System.out.println("两位 24 小时制的小时："+String.format("%tH", date));
        System.out.println("两位 12 小时制的小时："+String.format("%tI", date));
        System.out.println("24 小时制的小时："+String.format("%tk", date));
        System.out.println("12 小时制的小时："+String.format("%tl", date));
        System.out.println("两位小时中的分钟："+String.format("%tM", date));
        System.out.println("两位分钟中的秒："+String.format("%tS", date));
        System.out.println("表示时区缩写形式的字符串："+String.format("%tZ", date));
        System.out.println("上午还是下午："+String.format("%tp", date));
    }
}
```

运行结果如图 6.15 所示。

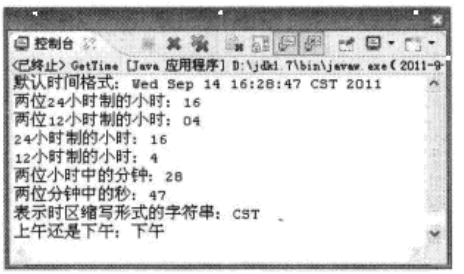


图 6.15　获取时间的各种格式的输出

6.3.4　日期时间组合格式化

因为日期与时间经常是同时出现的，所以格式化转换符还定义了各种日期和时间组合的格式。其中最常用的日期和时间的组合格式如表 6.3 所示。

表 6.3　常见的日期和时间组合的格式

转换符	说明	示例
%tF	“年-月-日”格式（4 位年份）	2008-03-25
%tD	“月/日/年”格式（2 位年份）	03/25/08
%tc	全部日期和时间信息	星期二 三月 25 15:20:00 CST 2008
%tr	“时:分:秒 PM（AM）”格式（12 时制）	03:22:06 下午
%tT	“时:分:秒”格式（24 时制）	15:23:50
%tR	“时:分”格式（24 时制）	15:25

【例 6.22】　在项目中创建 DateAndTime2 类，在主方法中创建 String 型变量，将日期和时间格式化后的结果输出。（**实例位置：光盘\TM\sl\6\8**）

```
import java.util.Date;
public class DateAndTime2 {
    public static void main(String[] args) {
        Date date=new Date();
        System.out.println("默认的日期时间格式："+date);
        System.out.println("24 小时制的时间，被格式化为小时和分钟："+String.format("%tR", date));
        System.out.println("24 小时制的时间，被格式化为小时、分钟和秒："+String.format("%tT", date));
        System.out.println("12 小时制的时间，被格式化为："+String.format("%tr", date));
        System.out.println("日期被格式化为："+String.format("%tD", date));
        System.out.println("ISO 8601 格式的完整日期，被格式化为："+String.format("%tF", date));
        System.out.println("日期和时间被格式化为："+String.format("%tc", date));
    }
}
```

运行结果如图 6.16 所示。

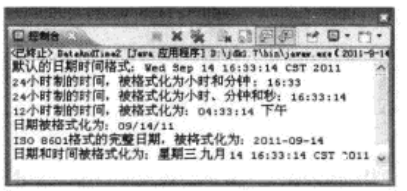


图 6.16 各种格式的输出

6.3.5 常规类型格式化

在程序设计过程中，经常需要对常规类型的数据进行格式化，如格式化为整数、格式化为科学计数表示等，在 Java 语言中可以使用常规类型的格式化转换符来实现。表 6.4 列出了常规类型的格式化转换符。

表 6.4 常规转换符

转 换 符	说 明	示 例
%b、%B	结果被格式化为布尔类型	true
%h、%H	结果被格式化为散列码	A05A5198
%s、%S	结果被格式化为字符串类型	"abcd"
%c、%C	结果被格式化为字符类型	'a'
%d	结果被格式化为十进制整数	40
%o	结果被格式化为八进制整数	11
%x、%X	结果被格式化为十六进制整数	4b1
%e	结果被格式化为用计算机科学记数法表示的十进制数	1.700000e+01
%a	结果被格式化为带有效位数和指数的十六进制浮点值	0X1.C000000000001P4
%n	结果为特定于平台的行分隔符	
%%	结果为字面值'%'	%

【例 6.23】 在项目中创建 General 类，在主方法中创建 String 型变量，实现不同数据类型到字

符串的转换。

```
public class General{                                          //新建类
    public static void main(String[] args) {                   //主方法
        String str1 = String.format("%d",400/2);               //将结果以十进制显示
        String str2 = String.format("%b",3 > 5);               //将结果以布尔形式显示
        String str3 = String.format("%x",200);                 //将结果以十六进制格式显示
        System.out.println("400 的一半是："+str1);              //输出格式化字符串
        System.out.println("3>5 正确吗："+str2);
        System.out.println("200 的十六进制数是："+str3);
    }
}
```

运行结果如图 6.17 所示。

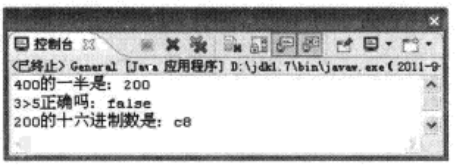


图 6.17　格式化常规类型

6.3.6　范例 3：将金额格式化成大写

在处理财务账款时，一般需要使用大写金额，如进行转账时，需要将转账金额写成大写的。对于这种情况，如果手动填写不仅麻烦，而且容易出错，所以常常需要通过程序控制自动进行转换。本范例实现了小写金额到大写金额的转换。运行结果如图 6.18 所示。（**实例位置：光盘\TM\sl\6\9**）

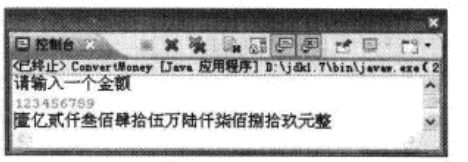


图 6.18　数字转换成大写

（1）在项目中创建 ConvertMoney 类，在该类的主方法中接收用户输入的金额，然后通过 convert()方法将这个金额转换成大写金额的字符串格式，并输出到控制台。代码如下：

```
public static void main(String[] args) {
    Scanner scan = new Scanner(System.in);                     //创建扫描器
    System.out.println("请输入一个金额");
    String convert = convert(scan.nextDouble());               //获取金额转换后的字符串
    System.out.println(convert);                               //输出转换结果
}
```

（2）编写金额转换方法 convert()，该方法在主方法中被调用，用于金额大写格式的转换。在该方法中创建 DecimalFormat 类的实例对象，通过这个格式器对象把金额数字格式化，值保留 3 位小数。然后分别调用 getInteger()与 getDecimal()方法转换整数与小数部分，并返回转换后的结果。代码如下：

```
public static String convert(double d) {
    DecimalFormat df = new DecimalFormat("#0.###");                    //实例化 DecimalFormat 对象
    String strNum = df.format(d);                                       //格式化 double 数字
    if (strNum.indexOf(".") != -1) {                                    //判断是否包含小数点
        String num = strNum.substring(0, strNum.indexOf("."));
        if (num.length() > 12) {                                        //整数部分大于 12 不能转换
            System.out.println("数字太大，不能完成转换！");
            return "";
        }
    }
    String point = "";                                                  //小数点
    if (strNum.indexOf(".") != -1) {
        point = "元";
    } else {
        point = "元整";
    }
    String result = getInteger(strNum) + point + getDecimal(strNum);    //转换结果
    if (result.startsWith("元")) {                                      //判断字符串是否以“元”结尾
        result = result.substring(1, result.length());                  //截取字符串
    }
    return result;                                                      //返回新的字符串
}
```

（3）编写 getInteger()方法，该方法用于转换数字整数部分的大写格式，在该方法中判断数字是否包含小数点，然后把数字转换为字符串并反转字符顺序，为每个数字添加对应的大写单位。代码如下：

```
public static String getInteger(String num) {
    if (num.indexOf(".") != -1) {                                       //判断是否包含小数点
        num = num.substring(0, num.indexOf("."));
    }
    num = new StringBuffer(num).reverse().toString();                   //反转字符串
    StringBuffer temp = new StringBuffer();                             //创建一个 StringBuffer 对象
    for (int i = 0; i < num.length(); i++) {                            //加入单位
        temp.append(STR_UNIT[i]);
        temp.append(STR_NUMBER[num.charAt(i) - 48]);
    }
    num = temp.reverse().toString();                                    //反转字符串
    num = numReplace(num, "零拾", "零");                                //替换字符串的字符
    num = numReplace(num, "零佰", "零");                                //替换字符串的字符
    num = numReplace(num, "零仟", "零");                                //替换字符串的字符
    num = numReplace(num, "零万", "万");                                //替换字符串的字符
    num = numReplace(num, "零亿", "亿");                                //替换字符串的字符
    num = numReplace(num, "零零", "零");                                //替换字符串的字符
    num = numReplace(num, "亿万", "亿");                                //替换字符串的字符
    if (num.lastIndexOf("零") == num.length() - 1) {                    //如果字符串以零结尾将其除去
        num = num.substring(0, num.length() - 1);
    }
    return num;
}
```

6.3.7 范例 4：将数字格式化成货币格式

数字可以标识货币、百分比、积分、电话号码等，就货币而言，在不同的国家和地区会以不同的格式来定义。本范例将接收用户输入的数字，将这个数字转换为不同国家或地区的货币格式，然后在控制台中输出这些货币格式。运行结果如图 6.19 所示。（**实例位置：光盘\TM\sl\6\10**）

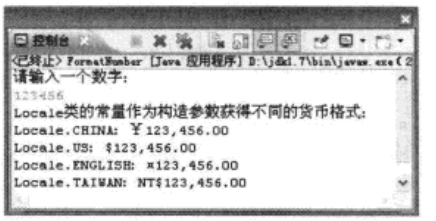


图 6.19 货币格式化

在项目中创建 FormatNumber 类，使用 format()方法将数字格式化成不同国家或地区的货币格式。代码如下：

```
import java.text.NumberFormat;
import java.util.Locale;
import java.util.Scanner;
public class FormatNumber {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);                                  //创建输入流扫描器
        System.out.println("请输入一个数字：");
        double number = scan.nextDouble();                                      //获取用户输入数字
        System.out.println("Locale 类的常量作为构造参数获得不同的货币格式：");
        NumberFormat format = NumberFormat.getCurrencyInstance(Locale.CHINA);   //创建格式化对象
        System.out.println("Locale.CHINA：" + format.format(number));           //输出格式化的货币
        format = NumberFormat.getCurrencyInstance(Locale.US);
        System.out.println("Locale.US：" + format.format(number));
        format = NumberFormat.getCurrencyInstance(Locale.ENGLISH);
        System.out.println("Locale.ENGLISH：" + format.format(number));
        format = NumberFormat.getCurrencyInstance(Locale.TAIWAN);
        System.out.println("Locale.TAIWAN：" + format.format(number));
    }
}
```

6.4 正则表达式

视频讲解：光盘\TM\lx\6\正则表达式.exe

在程序设计过程中，经常需要对输入的数据格式进行检查，这时就会用到正则表达式，匹配正则表达式则数据格式正确，否则格式错误。

6.4.1 判断是否符合正则表达式的方法

为了检查输入的数据是否满足某种格式，从 JDK 1.4 开始可以使用 String 类的 matches()方法进行

判断。语法格式如下：

```
boolean matches(String regex)
```

☑ regex：指定的正则表达式。

☑ 返回值：返回 boolean 类型。

该方法用于告知当前字符串是否匹配参数 regex 指定的正则表达式。返回值是 boolean 类型，如果当前字符串与正则表达式匹配，则该方法返回 true，否则返回 false。

6.4.2 正则表达式的元字符

正则表达式是由一些含有特殊意义的字符组成的字符串，这些含有特殊意义的字符称为元字符。表 6.5 列出了正则表达式的部分元字符。

表 6.5 正则表达式中的元字符

元 字 符	正则表达式中的写法	含 义
.	“.”	代表任意一个字符
\d	“\\d”	代表 0～9 的任何一个数字
\D	“\\D”	代表任何一个非数字字符
\s	“\\s”	代表空白字符，如'\t'、'\n'
\S	“\\S”	代表非空白字符
\w	“\\w”	代表可用作标识符的字符，但不包括“$”
\W	“\\W”	代表不可用于标识符的字符
\p{Lower}	\\p{Lower}	代表小写字母{a～z}
\p{Upper}	\\p{Upper}	代表大写字母{A～Z}
\p{ASCII}	\\p{ASCII}	ASCII 字符
\p{Alpha}	\\p{Alpha}	字母字符
\p{Digit}	\\p{Digit}	十进制数字，即[0～9]
\p{Alnum}	\\p{Alnum}	数字或字母字符
\p{Punct}	\\p{Punct}	标点符号：!"#$%&'()*+,-./:;<=>?@[\]^_`{|}～
\p{Graph}	\\p{Graph}	可见字符：[\p{Alnum}\p{Punct}]
\p{Print}	\\p{Print}	可打印字符：[\p{Graph}\x20]
\p{Blank}	\\p{Blank}	空格或制表符：[\t]
\p{Cntrl}	\\p{Cntrl}	控制字符：[\x00-\x1F\x7F]

说明 在正则表达式中，“.”代表任何一个字符，因此在正则表达式中如果想使用普通意义的点字符“.”，必须使用转义字符“\”。

【例 6.24】 在项目中创建 Decision 类，判断指定字符串是否合法，要求字符串由 7 个字符组成，并且第一位必须是大写字母，2～4 位必须是小写字母，后 3 位必须是数字字符，并在控制台输出结果。

（实例位置：光盘\TM\sl\6\11）

```
public class Decision {
    public static void main(String[] args) {
        String s1 = "Mrkj007";                                  //需要进行判断的字符串
        String s2 = "mrkj007";
                                                                //判断字符串变量 s 的值是否合法
        boolean bn1 = s1.matches("\\p{Upper}\\p{Lower}\\p{Lower}\\p{Lower}\\d\\d\\d");
        boolean bn2 = s2.matches("\\p{Upper}\\p{Lower}\\p{Lower}\\p{Lower}\\d\\d\\d");
        System.out.println(s1+"是合法的数据吗？"+bn1);          //在控制台输出结果
        System.out.println(s2+"是合法的数据吗？"+bn2);
    }
}
```

运行结果如图 6.20 所示。

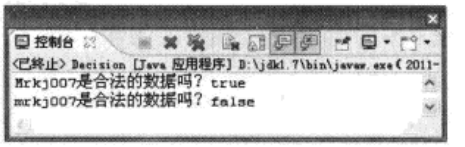


图 6.20　判断字符是否合法

本实例中为 matches()方法指定的正则表达式参数是 1 个\\p{Upper}、3 个\\p{Lower}和 3 个\\d，正则表达式中总计有 7 个字符，表示字符串必须由 7 个字符组成，第一位是大写字母，再三位是小写字母，最后三位是数字字符。由于本例中字符串变量 s1 的值符合该正则表达式，所以输出结果为 true，s2 的第一位是小写字母，不符合正则表达式，所以输出结果为 false。

6.4.3　正则表达式的限定符

在使用正则表达式时，如果需要某一类型的元字符多次输出，逐个输入就相当麻烦，这时可以使用正则表达式的限定元字符来重复次数。表 6.6 列出了常用限定符及其含义。

表 6.6　限定符

限定修饰符	意　义	示　例
?	0 次或 1 次	A?
*	0 次或多次	A*
+	0 次或多次	A+
{n}	正好出现 n 次	A{2}
{n,}	至少出现 n 次	A{3,}
{n,m}	出现 n～m 次	A{2,6}

像例 6.24 中的语句：

```
boolean bn2 = s2.matches("\\p{Upper}\\p{Lower}\\p{Lower}\\p{Lower}\\d\\d\\d");
```

就可以写成：

```
boolean bn=str.matches("\\p{Upper}\\p{Lower}{3}\\d{3}");
```

运行结果是相同的，但写起来就方便多了，尤其在重复次数更多的时候。

【例 6.25】 在项目中创建 Judge 类，在主方法中创建 String 对象，实现使用正则表达式来判断指定的变量是否是合法的 E-mail 地址。（**实例位置：光盘\TM\sl\6\12**）

```
public class Judge {                                          //新建类
    public static void main(String[] args) {                  //主方法
        String regex = "\\w{0,}\\@\\w{0,}\\.{1}\\w{0,}";      //正则表达式，定义邮箱格式
        String str1 = "aaa@";                                 //定义字符串 str1
        String str2 = "aaaa";                                 //定义字符串 str2
        String str3 = "aaaaa@111.com";                        //定义字符串 str3
          if(str1.matches(regex)){                            //判断字符串 str1 是否与正则表达式匹配
              System.out.println(str1+" 是一个 E_mail 地址格式"); //输出信息
          }
          if(str2.matches(regex)){                            //判断字符串 str2 是否与正则表达式匹配
              System.out.println(str2+" 是一个 E_mail 地址格式"); //输出信息
          }
          if(str3.matches(regex)){                            //判断字符串 str3 是否与正则表达式匹配
              System.out.println(str3+" 是一个 E_mail 地址格式"); //输出信息
          }
          else{                                               //如果与正则表达式都不匹配
              System.out.println("都不是 E_mail 地址格式");    //输出信息
          }
    }
}
```

运行结果如图 6.21 所示。

实例中返回与正则表达式匹配的字符串，是合法的 E-mail 地址，如果所有的字符串都和正则表达式不匹配，则返回“都不是 E-mail 地址格式”。

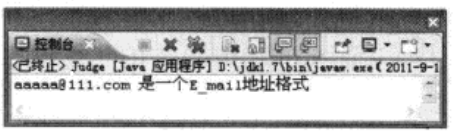


图 6.21 判断 E-mail 地址的合法性

6.4.4 方括号中元字符的含义

在正则表达式中还可以用方括号把多个字符括起来，方括号中各种正则表达式代表不同的含义。表 6.7 列出了方括号中元字符及其含义。

表 6.7 方括号中元字符的含义

字 符	含 义
[abc]	表示 a、b 或者 c
[^abc]	表示 a、b 和 c 之外的任何字符
[a-zA-Z]	a～z 或 A～Z 的任何字符
[a-d[m-p]]	a～d 或 m～p 的任何字符
[a-z&&[def]]	d、e 或者 f
[a-z&&[^bc]]	a～z 之间不含 b 和 c 的所有字符
[a-z&&[^m-p]]	a～z 之间不含 m～p 的所有字符

【例 6.26】 在项目中创建 CheckHandSetNum 类，在主方法中创建 String 对象，使用正则表达式对手机号码的合法性进行验证，在使用正则表达式时需要用到一些元字符和方括号中的元字符。（**实例位置：光盘\TM\sl\6\13**）

```
public class CheckHandSetNum {
    public static void main(String[] args) {
        String telephone1 = "11024588795";                    //定义要验证的号码
        String telephone2 = "15044268138";
        System.out.println(check(telephone1));
        System.out.println(check(telephone2));
    }
    public static String check(String handset) {
        String regex = "1[3,5,8]\\d{9}$";                     //定义正则表达式
        if (handset.matches(regex)) {                         //测试匹配结果
            return handset + "\n 是合法的手机号";             //提示合法手机号码
        } else {
            return handset + "\n 不是合法的手机号";           //提示非法手机号码
        }
    }
}
```

运行结果如图 6.22 所示。

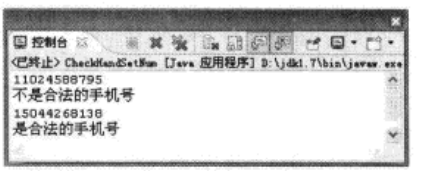


图 6.22 判断手机号的合法性

本实例中为 matches()方法指定的正则表达式前两位是 13、15 或 18，后面 9 位是 0～9 中的任意数，符合则是合法的手机号码，否则就是非法的手机号码，正则表达式最后的“$”是字符串的结尾匹配符。

6.4.5 范例 5：验证 IP 地址的合法性

IP 地址是网络上每台计算机的标识，在浏览器中输入的网址也要经过 DNS 服务器转换为 IP 地址才能找到服务器。对 IP 地址的验证要比对电话号码的验证复杂一些，因为 IP 地址的各数字之间使用“.”号进行连接，并且 3 种元字符都会用到。运行结果如图 6.23 所示。（**实例位置：光盘\TM\sl\6\14**）

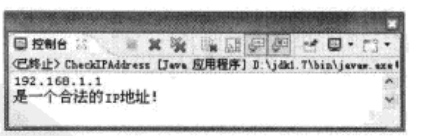


图 6.23 判断 IP 地址的合法性

在项目中创建 CheckIPAddress 类，判断输入的 IP 地址每一个字符是否匹配 IP 地址的正则表达式，每个字符都匹配是合法的 IP 地址，否则非法。代码如下：

```
public class CheckIPAddress {
    public static void main(String[] args) {
        String ip = "192.168.1.1";                            //创建判定的 IP
        System.out.println(matches(ip));
    }
```

```
    public static String matches(String text) {
        if (text != null && !text.isEmpty()) {                          //定义正则表达式
            String regex =  "^(1\\d{2}|2[0-4]\\d|25[0-5]|[1-9]\\d|[1-9])\\."
                          + "(1\\d{2}|2[0-4]\\d|25[0-5]|[1-9]\\d|\\d)\\."
                          + "(1\\d{2}|2[0-4]\\d|25[0-5]|[1-9]\\d|\\d)\\."
                          + "(1\\d{2}|2[0-4]\\d|25[0-5]|[1-9]\\d|\\d)$";
            if (text.matches(regex)) {                                  //判断 IP 地址是否匹配
                return text + "\n 是一个合法的 IP 地址！";
            } else {
                return text + "\n 不是一个合法的 IP 地址！";
            }
        }
        return "请输入要验证的 IP 地址！";
    }
}
```

本范例的运行结果为 192.168.1.1 是一个合法的 IP 地址，这段程序中的正则表达式比较长，表达式中的“^(1\\d{2}|2[0-4]\\d|25[0-5]|[1-9]\\d|[1-9])\\.”表示 IP 的前三位数字是 1 后面跟两个 0～9 的数字，或者 2 后面跟一个 0～4 的数字再跟一个 0～9 的数字，或者 25 开头跟一个 0～5 的数字，再或者以一个 1～9 的数字开头跟一个 0～9 的数字，再跟一个 1～9 的数字。只要满足其中的一个，前三位数就是合法的；后面的表达式与其类似。如果所有位数都匹配，则是一个合法的 IP 地址，若其中某一位不匹配，就不是合法的 IP 地址。

6.4.6 范例 6：计算文章中汉字的个数

计算一篇文章中的汉字格式通常是出版社的工作之一，可以使用正则表达式中表示汉字的正则表达式来限定条件，再将文章进行遍历，从而计算出汉字个数。运行结果如图 6.24 所示。（**实例位置：光盘\TM\sl\6\15**）

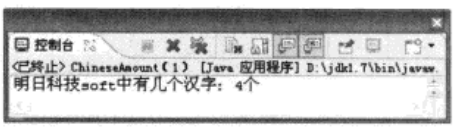


图 6.24 判断汉字个数

在项目中创建 ChineseAmount 类，判断字符串的每个字符是否匹配汉字的正则表达式，如果匹配，计数器加 1，最后计数器的值就是汉字个数。代码如下：

```
import java.util.regex.Pattern;
public class ChineseAmount {
    public static void main(String[] args) {
        String text = "明日科技 soft";                          //获取用户输入
        int amount = 0;                                         //创建汉字数量计数器
        for (int i = 0; i < text.length(); i++) {               //遍历字符串每一个字符
                                                                //使用正则表达式判断字符是否属于汉字编码
            boolean matches = Pattern.matches("^[\u4E00-\u9FA5]{0,}$", ""
                              + text.charAt(i));
            if (matches) {                                      //如果是汉字
                amount++;                                       //累加计数器
            }
```

```
        }
        System.out.println("明日科技 soft" + "中有几个汉字："+ amount + "个");
    }
}
```

6.5 字符串生成器

视频讲解：光盘\TM\lx\6\字符串生成器.exe

字符串生成器即 StringBuilder 类，是字符串一个重要的常用类，在程序中会经常用到，另外在程序运行时间上的节省也是很显著的。

6.5.1 StringBuilder 类

StringBuilder 类即字符串生成器，新创建的 StringBuilder 对象初始容量是 16 个字符，可以自行指定初始长度，也可以动态地执行添加、删除和插入等字符串的编辑操作，大大提高了频繁增加字符串的效率。如果附加的字符超过可容纳的长度，则 StringBuilder 对象将自动增加长度以容纳被附加的字符。

【例 6.27】 通过实例对比 String 与 StringBuilder 在对字符串做更新操作时所用时间的差别。（**实例位置：光盘\TM\sl\6\16**）

```
public class Jerque {                                          //新建类
    public static void main(String[] args) {                   //主方法
        String str = "";                                       //创建空字符串
        long starTime = System.currentTimeMillis();            //定义执行操作的起始时间
        for(long i = 0;i<1000000;i++){                         //for 循环执行 1000000 次操作
            str = str+i;                                       //循环追加字符串
        }
        long endTime = System.currentTimeMillis();             //定义对字符串操作后的时间
        long time = endTime - starTime;                        //计算执行操作的时间
        System.out.println("String 消耗时间："+time);          //将执行的时间输出
        StringBuilder builder = new StringBuilder("");         //创建字符串生成器
        starTime= System.currentTimeMillis();                  //定义操作执行前的时间
        for(long j = 0;j<1000000;j++){                         //利用 for 循环进行操作
            builder.append(j);                                 //循环追加字符
        }
        endTime = System.currentTimeMillis();                  //定义操作后的时间
        time = endTime - starTime;                             //追加操作执行的时间
        System.out.println("StringBuilder 消耗时间："+time);   //将操作时间输出
    }
}
```

运行结果如图 6.25 所示。

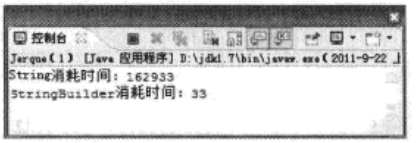


图 6.25 比较 String 和 StringBuilder 运行时间

通过这一实例可以看出执行时间上的巨大差别。如果在程序中频繁地附加字符串，建议使用 StringBuilder。

说明 StringBuilder 类是一个可变的字符序列，提供一个与 StringBuffer 类兼容的 API，但是并不保证与之同步。该类是 StringBuffer 类的一个简易替换，在字符串缓存被单个线程使用时要比 StringBuffer 类快。但是将 StringBuilder 的实例用于多个线程是不安全的，这时就要使用 StringBuffer 类来实现。

6.5.2 StringBuilder 类的常用方法

StringBuilder 类的构造方法有很多，主要是参数上的区别，这里主要介绍几种在编程中经常会用到的方法。

1. StringBuilder append(String str)方法

该方法将参数 str 指定的字符串追加到字符串生成器中。语法格式如下：

```
builder.append(String str);
```

- ☑ builder：声明的字符串生成器。
- ☑ str：追加到字符串生成器中的字符串。
- ☑ 返回值：返回字符串生成器的值，String 型。

【例 6.28】 使用 StringBuilder append(String str)方法将字符串追加到字符串生成器中。

```
String s = "good";                                    //声明字符串
StringBuilder builder=new StringBuilder();            //构造字符串生成器
builder.append(s);                                    //将 s 的值追加到字符串生成器
System.out.println("追加后 builder 的值是："+builder);
```

输出结果是“追加后 builder 的值是：good”，已经成功地将字符串 s 的值 good 追加到字符串生成器中。

说明 append()方法不仅可以实现字符串的追加，还可以实现任何类型的数据追加，如 int、boolean、char 等，甚至其他的字符串生成器也可以往里追加。

2. StringBuilder append(StringBuffer sb)方法

该方法将参数 sb 指定的字符串缓存 StringBuffer 的值追加到字符串生成器中。语法格式如下：

```
builder.append(StringBuffer sb);
```

☑ builder：声明的字符串生成器。

☑ sb：缓存中的值，追加到字符串生成器中。

☑ 返回值：返回将缓存的值追到字符串生成器后的值，String 型。

【例 6.29】 使用 StringBuilder append(StringBuffer sb)方法将字符串缓存中的值追加到字符串生成器中。

```
StringBuffer buffer=new StringBuffer("Sunday");          //创建初始内容为 Sunday 的字符串缓存
StringBuilder builder=new StringBuilder();               //构造字符串生成器
builder.append(buffer);                                  //向字符串生成器中追加字符串缓存
System.out.println("追加后 builder 的值是："+builder);
```

输出结果是“追加后 builder 的值是：Sunday”，显示已经成功地将字符串缓存 buffer 中的值 Sunday 追加到字符串生成器中。

3. StringBuilder insert(int offset, String str)方法

该方法将参数指定的字符串 str 添加到 offset 指定的位置。语法格式如下：

```
builder.insert(int offset, String str);
```

☑ builder：声明的字符串生成器。

☑ offset：指定字符串添加的下标索引位置。

☑ str：添加的字符串。

☑ 返回值：返回添加字符串后的字符串生成器的值，String 型。

【例 6.30】 在项目中创建 SubO 类，在主方法中构造一个字符串生成器 builder，初始值是字符串 s1 的值 int，将字符串 s2 的值 ser 追加到字符串生成器索引位置 2 处，并将 builder 输出。

```
public class SubO {
    public static void main(String[]args){
        String s1 = "int";                                  //创建字符串对象
        String s2 = "ser";                                  //创建字符串对象
        StringBuilder builder=new StringBuilder(s1);        //构造初始值为 int 的字符串生成器
        builder.insert(2,s2);                               //向索引值为 2 处添加字符串 ser
        System.out.println("追加后 builder 的值是："+builder);
    }
}
```

运行结果如图 6.26 所示。

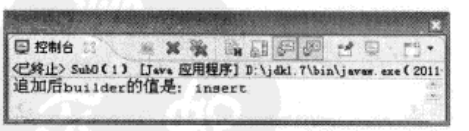


图 6.26 将字符串追加到指定索引处

输出结果显示已经成功地将字符串 ser 追加到字符串生成器 builder 中，现在 builder 的值是 insert。此外还能从结果中看出，字符串生成器中字符的索引位置是从 0 开始的，即 i 的索引位置是 0，n 的索引位置是 1，所以 ser 追加在 n 后面索引位置为 2 处，即原来 t 的位置，追加后 s、e、r 和 t 的索引位置分别是 2、3、4 和 5。

4. StringBuilder delete(int start, String end)方法

该方法移除字符串生成器中的子字符串，该子字符串从指定的 start 处开始，到指定的 end 处结束。语法格式如下：

```
builder.delete(int start, String end);
```

- ☑ builder：声明的字符串生成器。
- ☑ start：删除子字符串的起始索引位置。
- ☑ end：删除子字符串的结尾索引位置。
- ☑ 返回值：返回删除子字符串后的字符串生成器的值，String 型。

【例 6.31】 在项目中创建 SubD 类，在主方法中构造一个字符串生成器 builder，初始值是 String Builder，将指定的字符串生成器的子字符串从中删除，将结果输出。

```
public class SubD {
    public static void main(String[]args){
        StringBuilder builder = new StringBuilder("StringBuilderd");      //构建字符串生成器
        StringBuilder s1 = builder.delete(5, 5);                          //删除字符串
        System.out.println("删除指定字符串后：s1="+s1);                   //输出删除后的字符串
        StringBuilder s2 = builder.delete(5, 10);
        System.out.println("删除指定字符串后：s2="+s2);
    }
}
```

运行结果如图 6.27 所示。

从输出结果中可以看出两点：如果 start 和 end 的位置相同，字符串生成器的值不变，按原样输出；字符串生成器索引 start 的值被删掉，end 的值仍然保留，说明删除的子字符串包含开始索引不包含结尾索引。

图 6.27 删除字符串生成器的子字符串

5. String toString()方法

该方法返回字符串生成器的字符串表示，即将字符串生成器转换为字符串，转换后字符串生成器的值不变。语法格式如下：

```
str.toString();
```

- ☑ str：字符串生成器的值，转换字符串。
- ☑ 返回值：返回转换后的字符串。

【例 6.32】 使用 String toString()方法将字符生成器中的值转换为字符串。

```
StringBuilder builder = new StringBuilder("HelloWord");      //构建字符串生成器
String s = builder.toString();                               //将字符串生成器转换为字符串
System.out.println("s="+s);
System.out.println("builder="+builder);
```

输出结果是“s=HelloWord”和“builder=HelloWord”，已经成功地将字符串生成器中的值转换成了

字符串类型，并且知道转换之后字符串生成器中的值不变，仍然是 HelloWord。

说明 这里只列举出来部分常用的方法，想要了解更多 StringBuilder 类方法，可以查询 java.lang.StringBuilder 的 API 文档。

6.5.3 范例 7：将汉字和字母转换成 Unicode 码

Unicode 码是国际通用的一种计算机编码，各国语言文字、标点符号都可以用 Unicode 码表示，现在生活中很多地方都会用到 Unicode 码。下面的程序就是将字母、汉字等转换成 Unicode 码。运行结果如图 6.28 所示。（**实例位置：光盘\TM\sl\6\17**）

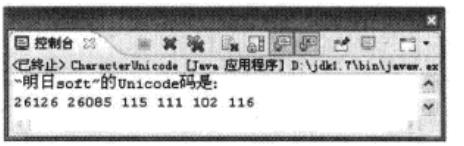


图 6.28 将字符转换成 Unicode 码

在项目中创建 CharacterUnicode 类，将字符串转换成字符数组，再将汉字和字母转换成 Unicode 码追加到字符串生成器中。代码如下：

```
public class CharacterUnicode{
    public static void main(String[] args) {
        String text = "明日 soft";                          //获取用户输入的字符串
        char[] charArray = text.toCharArray();              //获取字符串的字符数组
        StringBuilder builder = new StringBuilder();        //创建字符串生成器
        for (char c : charArray) {                          //遍历字符数组
            builder.append((int) c + " ");                  //转换成 Unicode 码并追加到 builder 中
        }
        System.out.println(""明日 soft"的 Unicode 码是：");
        System.out.println(builder.toString());             //结果输出到文本框
    }
}
```

6.5.4 范例 8：去掉字符串中重复的字符

在项目中创建 SBDelete 类，使用字符串生成器 StringBuilder 类的方法去掉字符串中重复的字符，下面借助冰心《繁星》中的一首小诗进行说明，运行结果如图 6.29 所示。（**实例位置：光盘\TM\sl\6\18**）

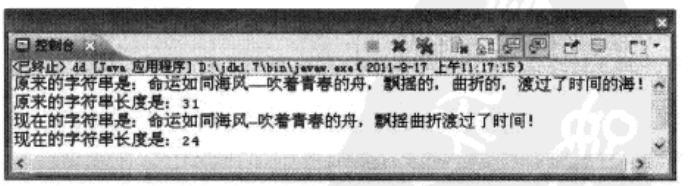


图 6.29 去掉字符串中重复的字符

在项目中创建 SBDelete 类，把每一个字符和其后面的进行比较，如果相同就利用 StringBuilder 类的 delete()方法将后面的删除。代码如下：

```
public class SBDelete {
    public static void main(String[]args){
        String s = "命运如同海风—吹着青春的舟，飘摇的，曲折的，渡过了时间的海！";
        StringBuilder sb = new StringBuilder(s);                    //创建字符串生成器并赋值
        System.out.println("原来的字符串是："+sb);                   //输出原字符串
        System.out.println("原来的字符串长度是："+sb.length());       //输出原字符串长度
        for (int i=0;i<sb.length();i++){
            for (int j=i+1;j<sb.length();j++){                      //每个字符和其后的进行比较
                if (sb.charAt(i)==sb.charAt(j)){                    //如果字符相同
                    sb.deleteCharAt(j);                             //将后面的删除
                }
            }
        }
        System.out.println("现在的字符串是："+sb);                   //输出新字符串
        System.out.println("现在的字符串长度是："+sb.length());       //输出新字符串长度
    }
}
```

6.6 经 典 范 例

6.6.1 经典范例 1：实现字符串加密

视频讲解：光盘\TM\lx\6\实现字符串加密.exe

为了保证安全，经常需要对字符串进行加密，可以有很多方法，最常用的就是获取字符串的字节数组，再对字节数组中的每个字节都进行操作，这时所得的字符串与原字符串就不相同，以此达到加密的效果，解密时再将加密字符串进行相反的运算，这样即可得到原字符串。运行结果如图 6.30 所示。（**实例位置：光盘\TM\sl\6\19**）

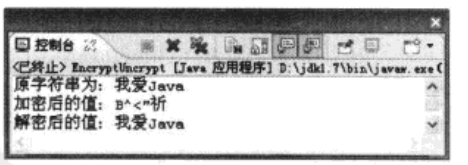


图 6.30 实现字符串加密

在项目中创建 EncryptUncrypt 类，将字符串转换成字节数组，再将每个字节和密文字符异或操作进行加密；做相反的运算就可以进行解密。代码如下：

```
public class EncryptUncrypt {
    public static String EAndU(String value, char secret) {     //对 value 加密，secret 密文字符
        byte[] bt = value.getBytes();                           //将需要加密的内容转换为字节数组
        for (int i = 0; i < bt.length; i++) {
            bt[i] = (byte) (bt[i] ^ (int) secret);              //通过异或运算进行加密
        }
```

```
        return new String(bt, 0, bt.length);                    //返回加密后的字符串
    }
    public static void main(String[] args) {
        String value = "我爱 Java";                              //需要加密的内容
        char secret = '祈';                                      //密文字符
        System.out.println("原字符串为：" + value);
        String encrypt = EncryptUncrypt.EAndU(value, secret);   //加密
        System.out.println("加密后的值：" + encrypt);
        String uncrypt = EncryptUncrypt.EAndU(encrypt, secret); //解密
        System.out.println("解密后的值：" + uncrypt);
    }
}
```

6.6.2 经典范例 2：验证字符串是否回文

视频讲解：光盘\TM\lx\6\验证字符串是否回文.exe

给定一个字符串，如果从前往后读和从后往前读完全一样就称为回文。创建一个字符串，内容是南朝梁简文帝萧纲的诗《咏雪》，通过程序验证这首诗的回文性。运行结果如图 6.31 所示。（**实例位置：光盘\TM\sl\6\20**）

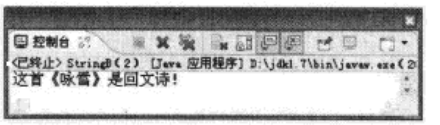


图 6.31 验证字符串是否回文

在项目中创建 StringB 类，使用 reverse()方法把字符串反转，然后比较两个字符串相同位置的字符是否相同，如果都相同就是回文。代码如下：

```
public class StringB {
    public static void main(String[] args) {
        String str1 = "这首《咏雪》";                            //获取字符串
        StringBuilder str2 = new StringBuilder(str1);           //将字符串添加到字符串生成器
        str2.reverse();                                         //字符串生成器中字符串反转
        int count = 0;                                          //定义变量
        for (int i = 0; i < str1.length(); i++) {               //两字符串同索引处字符比较
            if (str1.charAt(i) != str2.charAt(i)) {             //有一个不相等
                System.out.println("str1 不是回文诗！");        //输出不是回文诗
                break;                                          //跳出循环
            }
            if (str1.charAt(i) == str2.charAt(i)) {             //如果相等
                count++;                                        //count 自加
            }
        }
        if (count == str1.length()) {                           //count 和 str1 长度相等
            System.out.println("str1 是回文诗！");              //返回是回文诗
        }
    }
}
```

6.7 本章小结

Java 语言中的字符串处理技术是学习 Java 语言的关键。本章向读者介绍了字符串的创建、基本操作以及常用的字符串操作等基础知识，这些在实际编程中会经常遇到，因此应熟练地掌握。另外还介绍了格式化字符串、使用正则表达式以及字符串生成器等一些高级的字符串处理技术，这些也是字符串处理技术的重点，读者也应该熟练掌握。

6.8 实战练习

1．在项目中创建 Number 类，判断字符串“mingrikejijavabu”中字符‘i’出现了几次，并将结果输出。（**答案位置：光盘\TM\sl\6\21**）

2．前面介绍了去除字符串中所有空格的两种方法，试着不利用这两种方法，只利用 for 循环和 if 条件句实现去除空格。（**答案位置：光盘\TM\sl\6\22**）

3．在项目中创建 DateAndTime 类，将今天的全部信息格式化后输出，再将日期以“年-月-日”的格式输出。（**答案位置：光盘\TM\sl\6\23**）

PDG

第 7 章

类和对象

（视频讲解：46 分钟）

在 Java 语言中经常被提到的两个词汇是类与对象，实质上可以将类看作是对象的载体，它定义了对象所具有的功能。学习 Java 语言必须要掌握类与对象，这样可以从深层次去理解 Java 这种面向对象语言的开发理念，使程序员更好、更快地掌握 Java 编程思想与编程方式，因此掌握类与对象是学习 Java 语言的基础。本章详细介绍了类的各种方法以及对象，讲解过程中为了使初学者更容易入门，笔者列举了大量实例。

通过阅读本章，您可以：

- 了解面向对象编程思想
- 掌握如何定义类
- 掌握类的成员变量、成员方法
- 掌握权限修饰符
- 掌握局部变量以及作用范围
- 掌握 this 关键字
- 掌握如何创建对象
- 掌握使用对象获取对象的属性和行为
- 掌握对象的创建、比较和销毁

PDG

7.1 面向对象概述

视频讲解：光盘\TM\lx\7\面向对象概述.exe

在程序开发初期人们使用结构化开发语言，但是随着时间的流逝，软件的规模越来越庞大，结构化语言的弊端也逐渐暴露出来，开发周期被无休止地拖延，产品的质量也不尽如人意，人们终于发现结构化语言已经不再适合当前的软件开发。这时人们开始将另一种开发思想引入程序中，即面向对象的开发思想。面向对象思想是人类最自然的一种思考方式，它将所有预处理的问题抽象为对象，同时了解这些对象具有哪些相应的属性以及展示这些对象的行为，以解决这些对象面临的一些实际问题，这样就在程序开发中引入了面向对象设计的概念，面向对象设计实质上就是对现实世界的对象进行建模操作。

7.1.1 什么是对象

现实世界中，随处可见的一种事物就是对象，对象是事物存在的实体，如人类、书桌、计算机、高楼大厦等。人类解决问题的方式总是将复杂的事物简单化，于是就会思考这些对象都是由何种部分组成的。通常都会将对象划分为两部分，即动态部分与静态部分。静态部分，顾名思义，就是不能动的部分，这部分被称为“属性”，任何对象都会具备其自身属性，例如一个人，它包括高矮、胖瘦、性别、年龄等属性。然而具有这些属性的人会执行哪些动作也是一个值得探讨的部分，这个人可以哭泣、微笑、说话、行走，这些是这个人具备的行为（动态部分），人类通过探讨对象的属性和观察对象的行为了解对象。

在计算机的世界中，面向对象程序设计的思想要以对象来思考问题，首先要将现实世界的实体抽象为对象，然后考虑这个对象具备的属性和行为。例如，现在面临一只大雁要从北方飞往南方这样一个实际问题，试着以面向对象的思想来解决这一实际问题，步骤如下：

（1）首先可以从这一问题中抽象出对象，这里抽象出的对象为大雁。

（2）识别这个对象的属性。对象具备的属性都是静态属性，例如大雁有一对翅膀、黑色的羽毛等。这些属性如图 7.1 所示。

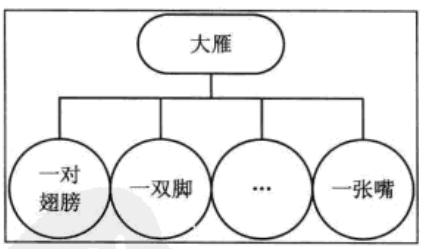


图 7.1 识别对象上的属性

（3）识别这个对象的动态行为，即这只大雁可以进行的动作，如飞行、觅食等，这些行为都是因为这个对象基于其属性而具有的动作。这些行为如图 7.2 所示。

（4）识别出这些对象的属性和行为后，这个对象就被定义完成，然后可以根据这只大雁具有的特性制定这只大雁要从北方飞向南方的具体方案以解决问题。实质上究其本质，所有的鸟类都具有以上的属性和行为，可以将这些属性和行为封装起来以描述鸟类。由此可见，类实质上就是封装对象属性和行为的载体，而对象则是类抽象出来的一个实例，两者之间的关系如图 7.3 所示。

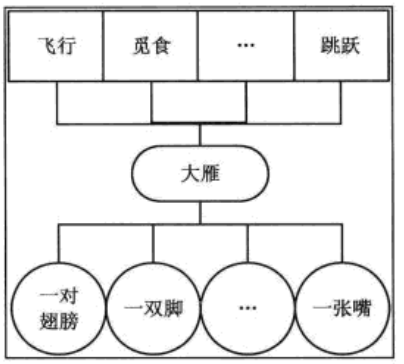


图 7.2 对象具有的属性和行为

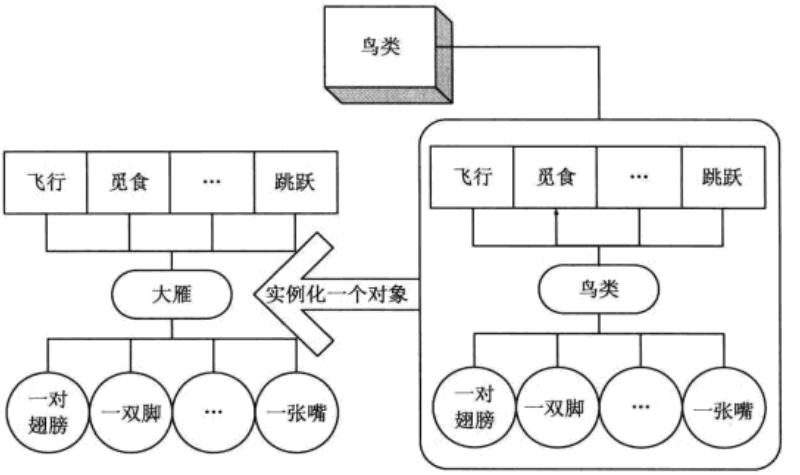


图 7.3 描述对象与类之间的关系

7.1.2 什么是类

不能将所谓的一个事物描述成一类事物，如一只鸟不能称为鸟类，如果需要对同一类事物统称，就不得不说明类这个概念。

类就是同一类事物的统称，如果将现实世界中的一个事物抽象成对象，类就是这类对象的统称，如鸟类、家禽类、人类等。类是构造对象时所依赖的规范，例如，一只鸟具有一对翅膀，而它可以通过这对翅膀飞行，而基本上所有的鸟都具有翅膀这个特性和飞行的技能，这样的具有相同特性和行为的一类事物就称为类，类的思想就是这样产生的。在图 7.3 中已经描述过类与对象之间的关系，对象就是符合某个类定义所产生出来的实例。更为恰当的描述是：类是世间事物的抽象称呼，而对象则是这个事物相对应的实体。如果面临实际问题，通常需要实例化类对象来解决。例如解决大雁南飞的问题，这里只能拿这只大雁来处理这个问题，不能拿大雁类或鸟类来解决。

类是封装对象的属性和行为的载体，反过来说具有相同属性和行为的一类实体被称为类。例如一个鸟类，鸟类封装了所有鸟的共同属性和应具有的行为，其结构如图 7.4 所示。

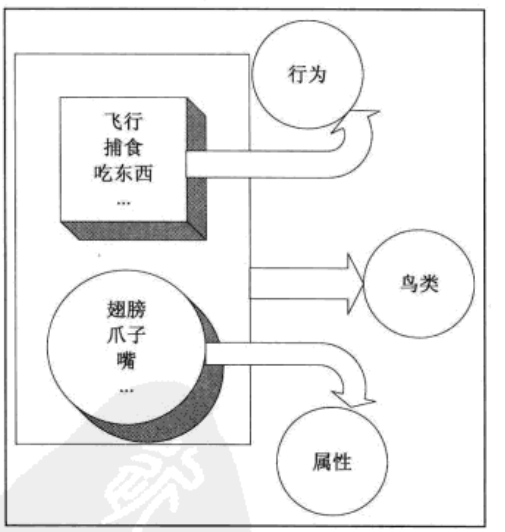


图 7.4 鸟类结构

定义完成鸟类之后，可以根据这个类抽象出一个实体对象，最后通过实体对象来解决相关一些实际问题。

在 Java 语言中，类中对象的行为是以方法的形式定义的，对象的属性是以成员变量的形式定义的，而类包括对象的属性和方法，有关类的具体实现会在后续章节中进行介绍。

7.1.3 面向对象的特点

面向对象程序设计具有以下几个特点。

1. 封装

封装是面向对象编程的核心思想，将对象的属性和行为封装起来，而将对象的属性和行为封装起来的载体就是类，类通常对客户隐藏其实现细节，这就是封装的思想。例如，用户使用计算机，只需要使用手指敲击键盘就可以实现一些功能，用户无须知道计算机内部是如何工作的，即使用户可能碰巧知道计算机的工作原理，但在使用计算机时并不完全依赖于计算机工作原理这些细节。

采用封装的思想保证了类内部数据结构的完整性，应用该类的用户不能轻易直接操纵此数据结构，而只能执行类允许公开的数据。这样避免了外部对内部数据的影响，提高程序的可维护性。

使用类实现封装特性如图 7.5 所示。

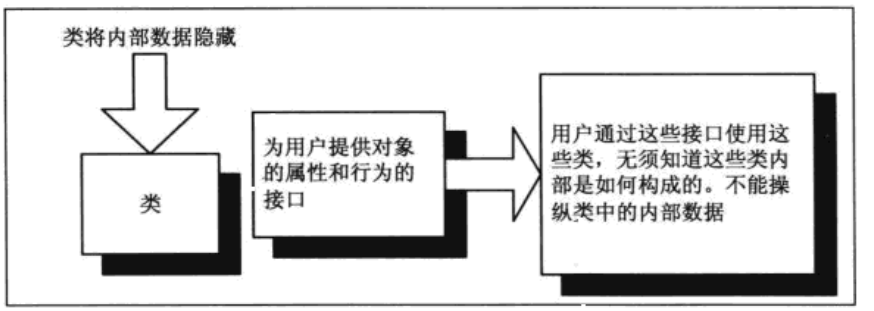


图 7.5 封装特性示意图

2. 继承

类与类之间同样具有关系，如一个百货公司类与销售员类相联系，类之间这种关系被称为关联。关联是描述两个类之间的一般二元关系，例如一个百货公司类与销售员类就是一个关联，再如学生类以及教师类也是一个关联。两个类之间的关系有很多种，继承是关联中的一种。

当处理一个问题时，可以将一些有用的类保留下来，当遇到同样问题时拿来复用。假如这时需要解决信鸽送信的问题，我们很自然就会想到图 7.4 所示的鸟类。由于鸽子属于鸟类，鸽子具有鸟类相同的属性和行为，便可以在创建信鸽类时将鸟类拿来复用，并且保留鸟类具有的属性和行为。不过，并不是所有的鸟都有送信的习惯，因此还需要再添加一些信鸽具有的独特属性以及行为。鸽子类保留了鸟类的属性和行为，这样就节省了定义鸟和鸽子共同具有的属性和行为的时间，这就是继承的基本思想。可见软件的代码使用继承思想可以缩短软件开发的时间，复用那些已经定义好的类可以提高系统性能，减少系统在使用过程中出现错误的几率。

继承性主要利用特定对象之间的共有属性。例如，平行四边形是四边形（正方形、矩形也都是四边形），平行四边形与四边形具有共同特性，就是拥有 4 个边，可以将平行四边形类看作四边形的延伸，平行四边形复用了四边形的属性和行为，同时添加了平行四边形独有的属性和行为，如平行四边形的

对边平行且相等。这里可以将平行四边形类看作是从四边形类中继承的。在Java语言中将类似于平行四边形的类称为子类，将类似于四边形的类称为父类或超类。值得注意的是，可以说平行四边形是特殊的四边形，但不能说四边形是平行四边形，也就是说子类的实例都是父类的实例，但不能说父类的实例是子类的实例，图形类之间的继承关系如图7.6所示。

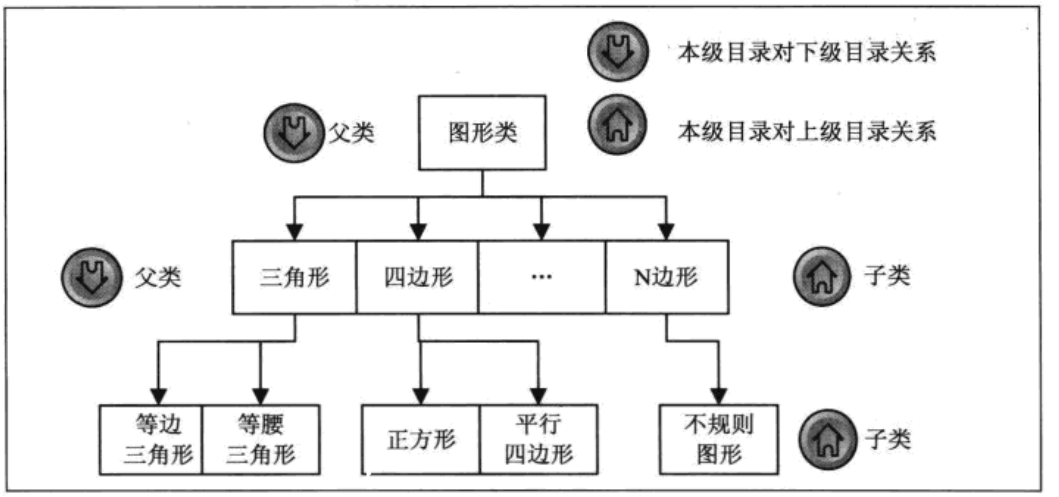


图7.6 图形类层次结构示意图

从图7.6中可以看出，继承关系可以使用树形关系来表示，父类与子类存在一种层次关系。一个类处于继承体系中，它既可以是其他类的父类，为其他类提供属性和行为，也可以是其他类的子类，继承父类的属性和方法，如三角形既是图形类的子类同时也是等边三角形的父类。

3. 多态

前面已介绍了继承，了解了父类和子类，其实将父类对象应用于子类的特征就是多态。依然以图形类来说明多态，每个图形都拥有绘制自己的能力，这个能力可以看作是该类具有的行为，如果将子类的对象统一看作是超类的实例对象，这样当绘制任何图形时，可以简单地调用父类也就是图形类绘制图形的方法即可绘制任何图形，这就是多态最基本的思想。

多态性允许以统一的风格编写程序，以处理种类繁多的已存在的类以及相关类。该统一风格可以由父类来实现，根据父类统一风格的处理，就可以实例化子类的对象。由于整个事件的处理都只依赖于父类的方法，所以日后只要维护和调整父类的方法即可，这样既降低了维护的难度，又节省了时间。

在提到多态的同时，不得不提到抽象类和接口，因为多态的实现并不依赖具体类，而是依赖于抽象类和接口。

再回到绘制图形的实例上来。作为所有图形的父类图形类，它具有绘制图形的能力，这个方法可以称为“绘制图形”，但如果要执行这个“绘制图形”的命令，没人知道应该画什么样的图形，并且如果要在图形类中抽象出一个图形对象，没有人能说清这个图形究竟是什么图形，所以使用“抽象”这个词汇来描述图形类比较恰当。在Java语言中称这样的类为抽象类，抽象类不能实例化对象。在多态的机制中，父类通常会被定义为抽象类，在抽象类中给出一个方法的标准，而不给出实现的具体流程。实质上这个方法也是抽象的，例如图形类中的“绘制图形”方法只提供一个可以绘制图形的标准，并

没有提供具体绘制图形的流程，因为没有人知道究竟需要绘制什么形状的图形。

在多态的机制中，比抽象类更为方便的方式是将抽象类定义为接口。由抽象方法组成的集合就是接口。接口的概念在现实中也极为常见，如从不同的五金商店买来螺丝和螺丝钉，螺丝很轻松地就可以拧在螺丝钉上，可能螺丝和螺丝钉的厂家不同，但这两个物品可以很轻易地组合在一起，这是因为生产螺丝和螺丝钉的厂家都遵循着一个标准，这个标准在 Java 语言中就是接口。依然拿“绘制图形”来说明，可以将“绘制图形”作为一个接口的抽象方法，然后使图形类实现这个接口，同时实现“绘制图形”这个抽象方法，当三角形类需要绘制时，就可以继承图形类，重写其中“绘制图形”方法，改写这个方法为“绘制三角形”，这样就可以通过这个标准绘制不同的图形。

7.2 类

视频讲解：光盘\TM\lx\7\类.exe

在 7.1.2 节中已经讲解过类是封装对象的属性和行为的载体，而在 Java 语言中对象的属性以成员变量的形式存在，而对象的方法以成员方法的形式存在。本节将介绍在 Java 语言中类是如何定义的。

7.2.1 类的构造方法

在类中除了成员方法之外，还存在一种特殊类型的方法，那就是构造方法。构造方法是一个与类同名的方法，对象的创建就是通过构造方法完成的，每当类实例化一个对象时，类都会自动调用构造方法。构造方法的特点如下：

- ☑ 构造方法没有返回值。
- ☑ 构造方法的名称要与本类的名称相同。

注意 在定义构造方法时，构造方法没有返回值，但这与普通没有返回值的方法不同，普通没有返回值的方法使用 public void methodEx()这种形式进行定义，但构造方法并不需要 void 关键字进行修饰。

构造方法的定义语法格式如下：

```
public book(){
    //...构造方法体
}
```

- ☑ public：构造方法修饰符。
- ☑ book：构造方法的名称。

在构造方法中可以为成员变量赋值，这样当实例化一个本类的对象时，相应的成员变量也将被初始化。

注意 如果在类中定义的构造方法都不是无参的构造方法，则编译器不会为类设置一个默认的无参构造方法，当试图调用无参构造方法实例化一个对象时，编译器会报错。所以只有在类中没有定义任何构造方法时，编译器才会在该类中自动创建一个不带参数的构造方法。

在上文中介绍过 this 关键字，了解了 this 可以调用类的成员变量和成员方法，事实上 this 还可以调用类中的构造方法。

【例 7.1】 在项目中创建 AnyThting 类，该类中使用 this 关键字调用构造方法。

```
public class AnyThting {
    public AnyThting(){                                    //定义无参构造方法
        this("this 调用有参构造方法");                      //使用 this 关键字调用有参构造方法
        System.out.println("无参构造方法");
    }
    public AnyThting(String name){                         //定义有参构造方法
        System.out.println("有参构造方法");
    }
}
```

在例 7.1 中可以看到定义了两个构造方法，在无参构造方法中可以使用 this 关键字调用有参的构造方法。但使用这种方式值得注意的是，只可以在无参构造方法中的第一句使用 this 关键字调用有参构造方法。

7.2.2 类的主方法

主方法是类的入口点，它定义了程序从何处开始；主方法提供对程序流向的控制，Java 编译器通过主方法来执行程序。主方法的语法格式如下：

```
public static void main(String[] args){
    //方法体
}
```

在主方法的定义中可以看到主方法具有以下特性：

- ☑ 主方法是静态的。要直接在主方法中调用其他方法，则该方法必须也是静态的。
- ☑ 主方法没有返回值。
- ☑ 主方法的形参为数组。

用 args[0]～args[n]分别代表程序的第一到第 n 个参数，可以使用 args.length 获取参数的个数。

【例 7.2】 在项目中创建 TestMain 类，在主方法中编写如下代码，并在 Eclipse 中设置程序参数。

```
public class TestMain {
    public static void main(String[] args) {               //定义主方法
        for(int i=0;i<args.length;i++){                    //根据参数个数做循环操作
```

```
            System.out.println(args[i]);                    //循环打印参数内容
        }
    }
}
```

运行结果如图 7.7 所示。

在 Eclipse 中设置程序参数的步骤如下：

（1）在 Eclipse 中选择菜单栏中的“运行”/“运行”命令，弹出“运行”对话框。

（2）选择“自变量”选项卡，在“程序自变量”文本框中输入相应的参数，每个参数间按 Enter 键隔开。具体设置如图 7.8 所示。

图 7.7　带参数的程序

图 7.8　Eclipse 中的“运行”对话框

7.2.3　成员变量

在 Java 语言中对象的属性称为成员变量，也可以称为属性。为了了解成员变量，首先定义一个图书类，成员变量对应于类对象的属性，在 Book 类中设置 id、name 和 category 3 个成员变量，分别对应于图书编号、图书名称和图书类别这 3 个图书属性。

【例 7.3】　在项目中创建 Book 类，并在该类中定义并使用成员变量。

```
public class Book{
    private int id;                                  //定义一个 int 型的成员变量
    private String name;                             //定义一个 String 型的成员变量
    private String category;                         //定义一个 String 型的成员变量
    public String getName(){                         //定义一个 getName()方法
        int id=0;                                    //局部变量
        setName("Java");                             //调用类中其他方法
        return id+this.name;                         //设置方法返回值
```

```
    }
    private void setName(String name){                    //定义一个 setName()方法
        this.name=name;                                   //将参数值赋予类中的成员变量
    }
    public Book getBook(){
        return this;                                      //返回 Book 类引用
    }
}
```

根据以上代码，读者可以看到在 Java 语言中使用 class 关键字来定义类，Book 是类的名称。同时在 Book 类中定义了 3 个成员变量，成员变量的类型可以设置为 Java 语言中合法的数据类型，其实成员变量就是普通的变量，可以为它设置初始值，也可以不为其设置，如果不设置初始值，则会有默认值。在 3 个成员变量前面读者应该注意到 private 关键字，它用来定义一个私有成员（关于权限修饰符的说明会在 7.2.8 节中进行介绍）。

7.2.4 成员方法

在 Java 语言中使用成员方法对应于类对象的行为。以 Book 类为例，它包含 getName()和 setName()两个方法，分别为获取图书名称和设置图书名称的方法。定义成员方法的语法格式如下：

```
权限修饰符 返回值类型 方法名(参数类型 参数名){
    ...//方法体
    return 返回值;
}
```

一个成员方法可以有参数，这个参数可以是对象也可以是基本数据类型的变量，同时成员方法有返回值和不返回任何值的选择，如果方法需要返回值可以在方法体中使用 return 关键字，使用这个关键字后，方法的执行将被终止。

注意

Java 语言中的成员方法无返回值可以使用 void 关键字表示。

成员方法的返回值可以是计算结果也可以是其他想要的数值、对象，返回值类型要与方法返回的值类型一致。

在成员方法中可以调用其他成员方法和类成员变量，例如在 getName()方法中就调用了 setName()方法将图书名称赋予一个值。同时在成员方法中可以定义一个变量，这个变量为局部变量（局部变量的内容会在 7.2.5 节中进行介绍）。

说明

如果一个方法中含有与成员变量同名的局部变量，则方法中对这个变量的访问以局部变量进行访问。例如变量 id，在 getName()方法中 id 的值为 0，而不是成员变量中 id 的值。

7.2.5 局部变量

在 7.2.4 节中已经讲述过成员方法，如果在成员方法内定义一个变量，那么这个变量被称为局部变量。

例如，在例 7.1 中定义的 Book 类中，getName()方法的 id 变量即为局部变量。实际上方法中的形参也可作为一个局部变量。例如在定义 setName(String name)方法时，String name 这个形参就被看作是局部变量。

局部变量是在方法被执行时创建，在方法执行结束时被销毁。局部变量在使用时必须进行赋值操作或被初始化，否则会出现编译错误。

【例 7.4】 在项目中创建一个类文件，在该类中定义 getName()方法并进行调用。

```
public String getName(){                    //定义一个 getName()方法
    int id=0;                               //局部变量
    setName("Java");                        //调用类中其他方法
    return id+this.name;                    //设置方法返回值
}
```

如果将 id 这个局部变量的初始值去掉，编译器将出现错误。

7.2.6 局部变量的有效范围

可以将局部变量的有效范围称为变量的作用域，局部变量的有效范围从该变量的声明开始到该变量的结束为止。局部变量的作用范围如图 7.9 所示。

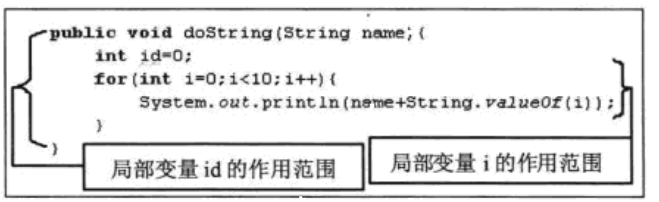


图 7.9 局部变量的作用范围

在相互不嵌套的作用域中可以同时声明两个名称、类型相同的局部变量，如图 7.10 所示。

图 7.10 在不同嵌套区域可以定义相同名称、类型的局部变量

但是在相互嵌套的区域中不可以这样声明，如果将局部变量 id 值在方法体的 for 循环中再次定义，

编译器将会报错，如图 7.11 所示。

```
public void doString(String name){
    int id=0;
    for(int i=0;i<10;i++){
        System.out.println(name+String.valueOf(i));
    }
    for(int i=0;i<3;i++){
        System.out.println(i);
        int id=7;
    }
}
```

在嵌套区域中定义局部变量 id 重复

图 7.11 在方法体的嵌套区域中不可以定义相同名称和类型的局部变量

注意 在作用范围外使用局部变量是一个常见的错误，因为在作用范围外没有声明局部变量的代码。

7.2.7 静态变量、常量和方法

在介绍静态变量、常量和方法之前首先需要介绍 static 关键字，因为由 static 修饰的变量、常量和方法被称作静态变量、常量和方法。

有时在处理问题时，会需要两个类在同一个内存区域共享一个数据。例如，在球类中使用 PI 这个常量，可能除了本类需要这个常量之外，在另外一个圆类中也需要使用这个常量。这时没有必要在两个类中同时创建 PI 这个常量，因为这样系统会将这两个不在同一个类中定义的常量分配到不同的内存空间中。为了解决这个问题，可以将这个常量设置为静态的。PI 常量在内存中被共享的布局如图 7.12 所示。

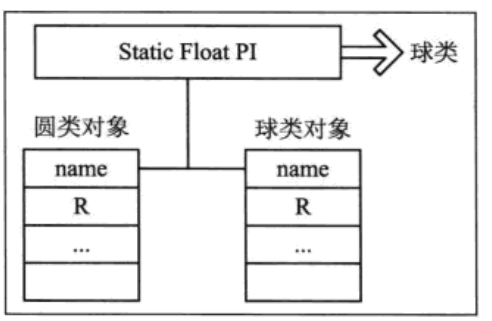


图 7.12 PI 常量在内存中被共享情况

被声明为 static 的变量、常量和方法被称为静态成员。静态成员是属于类所有的，区别于个别对象，可以在本类或其他类使用类名和“.”运算符调用静态成员。语法格式如下：

类名.静态类成员

【例 7.5】 在项目中创建 StaticTest 类，该类中的主方法调用静态成员并在控制台中输出。

```
public class StaticTest {
    static double PI=3.1415;                    //在类中定义静态常量
    static int id;                              //在类中定义静态变量
    public static void method1(){               //在类中定义静态方法
        //方法体
    }
    public void method2(){
    System.out.println(StaticTest.PI);          //调用静态常量
    System.out.println(StaticTest.id);          //调用静态变量
    StaticTest.method1();                       //调用静态方法
    }
}
```

在例 7.5 中设置了 3 个静态成员，分别为常量、变量和方法，然后在 method2()方法中分别调用这 3 个静态成员，直接使用“类名.静态成员”的形式进行调用即可。

注意 虽然静态成员也可以使用“对象.静态成员”的形式进行调用，但这样的形式通常不被鼓励使用，因为这样容易混淆静态成员和非静态成员。

静态数据与静态方法的作用通常是为了提供共享数据或方法，如数据计算公式等，以 static 声明并实现，这样当需要使用时，直接使用类名调用这些静态成员即可。尽管使用这种方式调用静态成员比较方便，但静态成员同样遵循着 public、private、protected 修饰符的约束。

【例 7.6】 在项目中创建 StaticTest 类，该类中的主方法调用静态成员并在控制台中输出。

```
public class StaticTest {
    static double PI=3.1415;                    //在类中定义静态常量
    static int id;                              //在类中定义静态变量
    public static void method1(){               //在类中定义静态方法
        //方法体
    }
    public void method2(){                      //在类中定义一个非静态方法
    System.out.println(StaticTest.PI);          //调用静态常量
    System.out.println(StaticTest.id);          //调用静态变量
    StaticTest.method1();                       //调用静态方法
    }
    public static StaticTest method3(){         //在类中定义一个静态方法
        method2();                              //调用非静态方法
        return this;                            //在 return 语句中使用 this 关键字
    }
}
```

读者也许会发现在 Eclipse 中输入上述代码后，编译器会发生错误，这是因为 method3()方法为一个静态方法，而在其方法体中调用了非静态方法和 this 关键字。在 Java 语言中对静态方法有以下两点规定：

- ☑ 在静态方法中不可以使用 this 关键字。
- ☑ 在静态方法中不可以直接调用非静态方法。

注意 在Java语言中规定不能将方法体内的局部变量声明为static。例如下述代码就是错误的：

```
public class example{
 public void method(){
      static int i=0;
 }
}
```

技巧 如果在执行类时，希望先执行类的初始化动作，可以使用static定义一个静态区域。例如：

```
public class example{
 static{
      //some
 }
}
```

当这段代码被执行时，首先执行static块中的程序，并且只会执行一次。

7.2.8 权限修饰符

Java语言中的权限修饰符主要包括private、public和protected，这些修饰符控制着对类和类的成员变量以及成员方法的访问。如果一个类的成员变量或成员方法被修饰为private，则该成员变量只能在本类中被使用，在子类中是不可见的，并且对其他包的类也是不可见的。如果将类的成员变量和成员方法的访问权限设置为public，则除了可以在本类使用这些数据之外，还可以在子类和其他包中的类中使用。如果一个类的访问权限被设置为private，这个类将隐藏其内的所有数据，以免用户直接访问它。如果需要使类中的数据被子类或其他包中的类使用，可以将这个类设置为public访问权限。如果一个类使用protected修饰符，那么只有本包内该类的子类或其他类可以访问此类中的成员变量和成员方法。

这么看来，public和protected修饰的类可以由子类访问，如果子类和父类不在同一包中，那么只有修饰符为public的类可以被子类进行访问。如果父类不允许通过继承产生的子类访问它的成员变量，那么必须使用private声明父类的这个成员变量。表7.1中描述了private、protected、public修饰符的修饰权限。

表7.1 Java语言中的修饰符

访问包位置	类修饰符		
	private	protected	public
本类	可见	可见	可见
同包其他类或子类	不可见	可见	可见
其他包的类或子类	不可见	不可见	可见

注意 当声明类时不使用 public、protected、private 修饰符设置类的权限，则这个类预设为包存取范围，即只有一个包中的类可以调用这个类的成员变量或成员方法。

【例 7.7】 在项目中的 com.wsy 包下创建 AnyClass 类，该类使用默认的访问权限。

```
package com.wsy;
class AnyClass {
    public void doString(){
        //...方法体
    }
}
```

在上述代码中，由于类的修饰符为默认修饰符，即只有一个包内的其他类和子类可以对该类进行访问，而 AnyClass 类中的 doString()方法却又被设置为 public 访问权限，即使这样，doString()方法的访问权限依然与 AnyClass 类的访问权限相同，因为 Java 语言规定，类的权限设定会约束类成员上的权限设定，所以上述代码等同于下面的代码。

【例 7.8】 本实例等同于例 7.7 的代码。

```
package com.wsy;
class AnyClass {
    void doString(){
        //...方法体
    }
}
```

7.2.9 this 关键字

【例 7.9】 在项目中创建一个类文件，该类中定义了 setName()方法，并将方法的参数值赋予类中的成员变量。

```
private void setName(String name){          //定义一个 setName()方法
    this.name=name;                         //将参数值赋予类中的成员变量
}
```

在上述代码中可以看到，成员变量与在 setName()方法中的形式参数的名称相同，都为 name，那么如何在类中区分使用的是哪一个变量？在 Java 语言中规定使用 this 关键字来指定。this 关键字被隐式地用于引用对象的成员变量和方法，如在上述代码中，this.name 指定的就是 Book 类中的 name 成员变量，而 this.name=name 语句中的第二个 name 则指定的是形参 name。实质上 setName()方法实现的功能就是将形参 name 的值赋予成员变量 name。

在这里读者明白了 this 可以调用成员变量和成员方法，但这并不是 Java 语言中的最常规调用方式——使用“对象.成员变量”或“对象.成员方法”进行调用。既然 this 关键字可以调用成员变量和成员方法，究竟 this 关键字与对象之间具有怎样的关系？事实上 this 引用就是对一个对象的引用。

在 setName()方法体中也可以使用 name=name 这种形式，可以在引用成员变量时省略 this 关键字，其结果与使用 this 关键字是一致的。既然使用 this 与没有使用 this 的效果相同，那么为什么要使用 this 关键字？其实 this 除了可以调用成员变量或成员方法之外，还可以作为方法的返回值。

【例 7.10】 在项目中创建一个类文件，该类中定义 Book 类型的方法，并通过 this 关键字进行返回。

```
public Book getBook(){
    return this;                                    //返回 Book 类引用
}
```

在 getBook()方法中，方法的返回值为 Book 类，所以方法体中使用 return this 这种形式将 Book 类的对象进行返回。

7.2.10 范例 1：自定义图书类

在使用 Java 语言进行开发时，时刻要以面向对象的思想考虑问题。面向对象的基础就是类，本范例将演示如何自定义类，它用于表示图书。运行结果如图 7.13 所示。（**实例位置：光盘\TM\sl\7\1**）

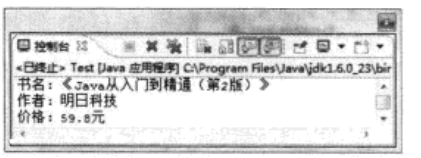


图 7.13 输出自定义的图书类对象

（1）在项目中创建 Book 类，在类中定义 3 个成员变量，分别表示书名、作者和价格，同时提供构造方法和成员方法来修改成员变量。代码如下：

```
public class Book {
    private String title;                                   //定义书名
    private String author;                                  //定义作者
    private double price;                                   //定义价格
    public Book(String title, String author, double price) {   //利用构造方法初始化域
        this.title = title;
        this.author = author;
        this.price = price;
    }
    public String getTitle() {                              //获得书名
        return title;
    }
    public String getAuthor() {                             //获得作者
        return author;
    }
    public double getPrice() {                              //获得价格
        return price;
    }
}
```

（2）在 com.mingrisoft 包中创建类文件，名称为 Test。在该类的 main()方法中，创建了一个 Book

对象并输出了其属性。代码如下：

```
public class Test {
    public static void main(String[] args) {
        Book book = new Book("《Java 从入门到精通（第 2 版）》", "明日科技", 59.8); //创建对象
        System.out.println("书名：" + book.getTitle());                          //输出书名
        System.out.println("作者：" + book.getAuthor());                         //输出作者
        System.out.println("价格：" + book.getPrice() + "元");                   //输出价格
    }
}
```

7.2.11 范例 2：温度单位转换工具

目前有两种广泛使用的温度单位，即摄氏度和华氏度。在标准大气压下，沸腾的水可以表示成 100 摄氏度和 212 华氏度。本范例将编写一个简单的温度单位转换工具，它可以将用户输入的摄氏度转换成华氏度，其运行结果如图 7.14 所示。（**实例位置：光盘\TM\sl\7\2**）

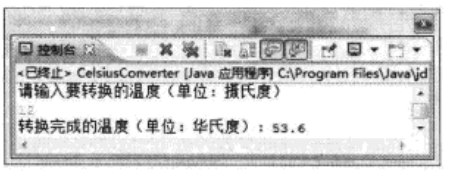


图 7.14 将输入的摄氏温度转换成华氏温度

在项目中创建 CelsiusConverter 类，在类中定义了两个方法，getFahrenheit()方法用于将摄氏温度转换成华氏温度，main()方法用于测试。代码如下：

```
public class CelsiusConverter {
    public double getFahrenheit(double celsius) {
        double fahrenheit = 1.8 * celsius + 32;                                  //计算华氏温度
        return fahrenheit;                                                       //返回华氏温度
    }
    public static void main(String[] args) {
        System.out.println("请输入要转换的温度（单位：摄氏度）");
        Scanner in = new Scanner(System.in);                                     //获得控制台输入
        double celsius = in.nextDouble();                                        //获得用户输入的摄氏温度
        CelsiusConverter converter = new CelsiusConverter();                     //创建类的对象
        double fahrenheit = converter.getFahrenheit(celsius);                    //转换温度为华氏度
        System.out.println("转换完成的温度（单位：华氏度）：" + fahrenheit);      //输出转换结果
    }
}
```

7.3 对 象

视频讲解：光盘\TM\lx\7\对象.exe

Java 是一门面向对象的程序设计语言，对象是由类抽象出来的，所有的问题都是通过对象来处理，

对象可以操作类的属性和方法解决相应的问题，所以了解对象的产生、操作和消亡对学习 Java 语言是十分必要的。本节就来讲解对象在 Java 语言中的应用。

7.3.1 对象的创建

在 7.1 节中曾经介绍过对象，对象可以认为是在一类事物中抽象出某一个特例，通过这个特例来处理这类事物出现的问题，在 Java 语言中通过 new 操作符来创建对象。以前曾经在讲解构造方法中介绍过每实例化一个对象就会自动调用一次构造方法，实质上这个过程就是创建对象的过程。准确地说，可以在 Java 语言中使用 new 操作符调用构造方法创建对象。语法格式如下：

```
Test test=new Test();
Test test=new Test("a");
```

其中参数说明如表 7.2 所示。

表 7.2 创建对象语法中的参数说明

设 置 值	描 述
Test	类名
test	创建 Test 类对象
new	创建对象操作符
"a"	构造方法的参数

test 对象被创建出来时，test 对象就是一个对象的引用，这个引用在内存中为对象分配了存储空间，可以在构造方法中初始化成员变量，当创建对象时，自动调用构造方法，也就是说在 Java 语言中初始化与创建是被捆绑在一起的。

每个对象都是相互独立的，在内存中占据独立的内存地址，并且每个对象都具有自己的生命周期，当一个对象的生命周期结束时，对象变成了垃圾，由 Java 虚拟机自带的垃圾回收机制处理，不能再被使用（对于垃圾回收机制的知识会在 7.3.5 小节中进行介绍）。

注意

在 Java 语言中对象和实例事实上可以通用。

下面来看一个创建对象的实例。

【例 7.11】 在项目中创建 CreateObject 类，在该类中创建对象并在主方法中创建对象。（**实例位置：光盘\TM\sl\7\3**）

```
public class CreateObject {
    public CreateObject(){                          //构造方法
        System.out.println("创建对象");
    }
    public static void main(String args[]){         //主方法
```

```
        new CreateObject();                                    //创建对象
    }
}
```

运行结果如图 7.15 所示。

在上述实例的主方法中使用 new 操作符创建对象，在创建对象的同时，自动调用构造方法中的代码。

图 7.15　创建对象

7.3.2　访问对象的属性和行为

当用户使用 new 操作符创建一个对象后，可以使用“对象.类成员”来获取对象的属性和行为。前文已经提到过，对象的属性和行为在类中是通过类成员变量和成员方法的形式来表示的，所以当对象获取类成员，也就相应地获取了对象的属性和行为。

【例 7.12】　在项目中创建 TransferProperty 类，该类中说明对象是如何调用类成员的。（**实例位置：光盘\TM\sl\7\4**）

```
public class TransferProperty {
    int i=47;                                                  //定义成员变量
    public void call(){                                        //定义成员方法
        System.out.println("调用 call()方法");
        for(i=0;i<3;i++){
            System.out.print(i+" ");
            if(i==2){
                System.out.println("\n");
            }
        }
    }
    public TransferProperty(){                                 //定义构造方法
    }
    public static void main(String[] args) {
        TransferProperty t1=new TransferProperty();            //创建一个对象
        TransferProperty t2=new TransferProperty();            //创建另一个对象
        t2.i=60;                                               //将类成员变量赋值为 60
        //使用第一个对象调用类成员变量
        System.out.println("第一个实例对象调用变量 i 的结果："+t1.i++);
        t1.call();                                             //使用第一个对象调用类成员方法
        //使用第二个对象调用类成员变量
        System.out.println("第二个实例对象调用变量 i 的结果："+t2.i);
        t2.call();                                             //使用第二个对象调用类成员方法
    }
}
```

运行结果如图 7.16 所示。

在上述代码的主方法中首先实例化一个对象，然后使用“.”操作符调用类的成员变量和成员方法。

但是在运行结果中可以看到，虽然使用两个对象调用同一个成员变量，结果却不相同，因为在打印这个成员变量的值之前将该值重新赋值为60，但在赋值时使用的是第二个对象t2调用成员变量，所以在第一个对象t1调用成员变量打印该值时仍然是成员变量的初始值，由此可见，两个对象的产生是相互独立的，改变了t2的i值，不会影响到t1的i值。在内存中这两个对象的布局如图7.17所示。

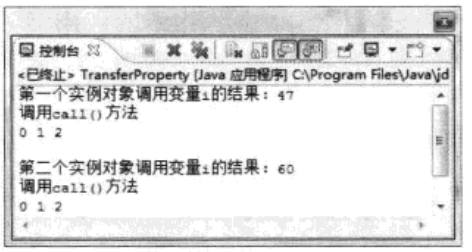


图7.16 使用对象调用类成员

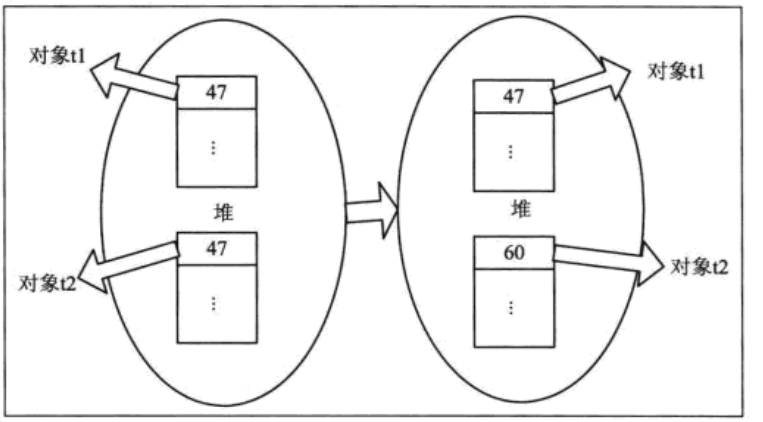


图7.17 内存中t1、t2两个对象的布局

如果希望成员变量不被其中任何一个对象改变，可以使用static关键字（前文曾经介绍过一个被声明为static的成员变量的值可以被本类或其他类的对象共享）。可以将上述代码改写为以下形式。

【例7.13】 在项目中创建CopyOfTransferProperty类，该类举例说明对象调用静态成员变量。（**实例位置：光盘\TM\sl\7\5**）

```
public class CopyOfTransferProperty {
    static int i=47;                                          //定义静态成员变量
    public void call(){                                       //定义成员方法
        System.out.println("调用 call()方法");
        for(i=0;i<3;i++){
            System.out.print(i+" ");
            if(i==2){
                System.out.println("\n");
            }
        }
    }
    public CopyOfTransferProperty(){                          //定义构造方法
    }
    public static void main(String[] args) {                  //定义主方法
        CopyOfTransferProperty t1=new CopyOfTransferProperty();  //创建一个对象
        CopyOfTransferProperty t2=new CopyOfTransferProperty();  //创建另一个对象
        t2.i=60;                                              //将类成员变量赋值为 60
        //使用第一个对象调用类成员变量
        System.out.println("第一个实例对象调用变量 i 的结果："+t1.i++);
        t1.call();                                            //使用第一个对象调用类成员方法
        //使用第二个对象调用类成员变量
        System.out.println("第二个实例对象调用变量 i 的结果："+t2.i);
```

```
        t2.call();                                          //使用第二个对象调用类成员方法
    }
}
```

运行结果如图 7.18 所示。

从上述运行结果中可以看到，由于使用“t2.i=60;”语句改变了静态成员变量的值，使用对象 t1 调用成员变量的值也为 60，这正是 i 值被定义为静态成员变量的功效，即使使用两个对象对同一个静态成员变量进行操作，依然可以改变静态成员变量的值，因为在内存中两个对象同时指向同一块内存区域。“t1.i++;”语句执行完毕后，i 值变为 3。当再次调用 call()方法时又被重新赋值为 0，做循环打印操作。

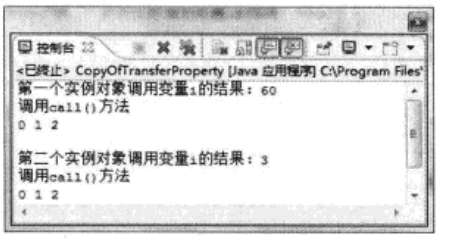


图 7.18　对象调用静态成员变量运行结果

7.3.3　对象的引用

在 Java 语言中尽管一切都可以看作对象，但真正操作标识符实质上是一个引用，那么引用究竟在 Java 语言中是如何体现的？来看下面的语法。

语法格式如下：

```
类名 对象引用名称
```

例如一个 Book 类的引用可以使用的代码如下：

```
Book book;
```

通常一个引用不一定需要有一个对象相关联，引用与对象相关联的语法如下：

```
Book book=new Book();
```

- ☑ Book：类名。
- ☑ book：对象。
- ☑ new：创建对象操作符。

注意　引用只是存放一个对象的内存地址，并非存放一个对象，严格说引用和对象是不同的，但是可以将这种区别忽略，如可以简单地说 book 是 Book 类的一个对象，而事实上应该说 book 是包含 Book 对象的一个引用。

7.3.4　对象的比较

在 Java 语言中有两种对象的比较方式，分别为“==”运算符与 equals()方法。实质上这两种方式有着本质区别，下面举例说明。

【例 7.14】 在项目中创建 Compare 类，该类说明了“==”运算符与 equals()方法的区别。（**实例位置：光盘\TM\sl\7\6**）

```
public class Compare {
    public static void main(String[] args) {
        String c1=new String("abc");                                    //创建两个 String 型对象引用
        String c2=new String("abc");
        String c3=c1;                                                   //将 c1 对象引用赋予 c3
        System.out.println("c2==c3 的运算结果为："+(c2==c3));            //使用“==”运算符比较 c2 与 c3
        System.out.println("c2.equals(c3)的运算结果为："+(c2.equals(c3))); //使用 equals()方法比较 c2 与 c3
    }
}
```

运行结果如图 7.19 所示。

从上述运行结果中可以看出，“==”运算符和 equals()方法比较的内容是不相同的。equals()方法是 String 类中的方法，它用于比较两个对象引用所指的内容是否相等；而“==”运算符比较的是两个对象引用的地址是否相等。由于 c1 与 c2 是两个不同的对象引用，两者在内存中的位置不同，而“String c3=c1;”语句将 c1 的引用赋给 c3，所以 c1 与 c3 这两个对象引用是相等的，也就是打印 c1==c3 这样的语句将返回 true 值。对象 c1、c2、c3 在内存中的布局如图 7.20 所示。

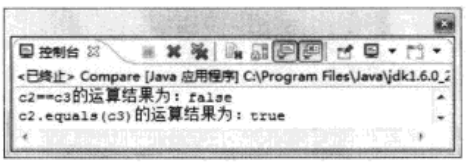


图 7.19 比较两个对象引用

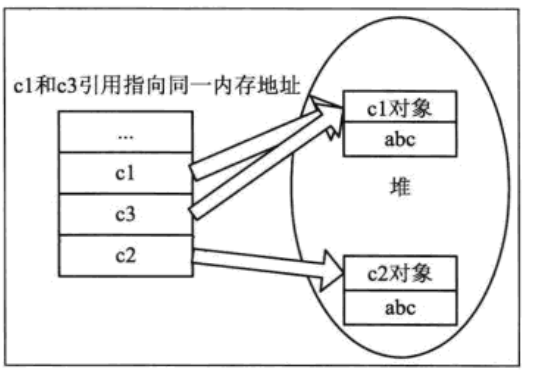


图 7.20 对象 c1、c2、c3 在内存中的布局

7.3.5 对象的销毁

每个对象都有生命周期，当对象的生命周期结束时，分配给该对象的内存地址将会被回收，在其他语言中需要手动回收废弃的对象，但是 Java 语言拥有一套完整的垃圾回收机制，用户不必担心废弃的对象占用内存，垃圾回收器将回收无用的占用内存的资源。

在谈到垃圾回收机制之前，读者首先需要了解何种对象会被 Java 虚拟机视为垃圾。主要包括以下两种情况：

（1）对象引用超过其作用范围，则这个对象将被视为垃圾，如图 7.21 所示。

（2）将对象赋值为 null，如图 7.22 所示。

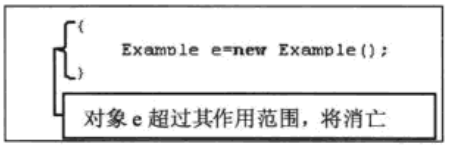


图 7.21　对象超过作用范围将消亡

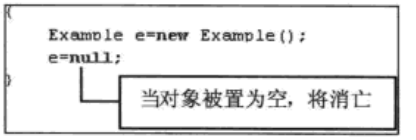


图 7.22　对象被置为 null 值时将消亡

虽然垃圾回收机制已经很完善，但垃圾回收器只能回收那些由 new 操作符创建的对象，如果某些对象不是通过 new 操作符在内存中获取一块内存区域，这种对象可能不被垃圾回收机制所识别，所以在 Java 语言中提供了一个 finalize()方法，这个方法是 Object 类的方法，它被声明为 protected，用户可以在自己的类中定义这个方法，如果用户在类中定义了 finalize()方法，在垃圾回收时首先调用该方法，并且在下一次垃圾回收动作发生时，才能真正回收对象占用的内存。

说明　垃圾回收或是 finalize()方法不保证一定会发生，如 Java 虚拟机面临内存损耗待尽的情形，它是不会执行垃圾回收的。

由于垃圾回收不受人为控制，具体执行时间也不确定，所以 finalize()方法也就无法执行，为此，Java 提供了 System.gc()方法强制启动垃圾回收器，这与给 120 打电话通知医院来救护的道理一样，告知垃圾回收器来清理。

7.3.6　范例 3：统计图书销量

在商品（类的实例）的销售过程中，需要对销量进行统计。本范例将在类的构造方法增加计数器来实现该功能，其运行结果如图 7.23 所示。（**实例位置：光盘\TM\sl\7\7**）

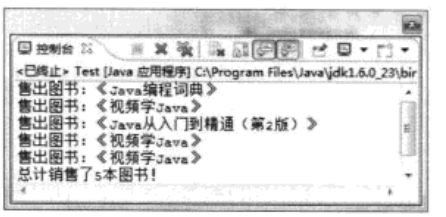


图 7.23　输出图书销售信息

（1）在项目中创建 Book 类，在类中定义了一个静态的成员变量用于保存实例化的次数。在构造方法中实现了计数器的功能，其代码如下：

```
public class Book {
    private static int counter = 0;                          //定义一个计数器
    public Book(String title) {
        System.out.println("售出图书：" + title);            //输出书名
        counter++;                                           //计数器加 1
    }
    public static int getCounter() {                         //获得计数器的结果
        return counter;
    }
}
```

（2）在项目中创建 Test 类，在类的 main()方法中创建 Book 类对象并输出创建对象的个数，代码如下：

```
public class Test {
    public static void main(String[] args) {
        //创建书名数组
        String[] titles = { "《Java 从入门到精通（第 2 版）》", "《Java 编程词典》", "《视频学 Java》" };
        for (int i = 0; i < 5; i++) {
            new Book(titles[new Random().nextInt(3)]);                    //利用书名数组创建 Book 对象
        }
        System.out.println("总计销售了" + Book.getCounter() + "本图书！");   //输出创建对象的个数
    }
}
```

7.3.7　范例 4：重新计算对象的哈希码

Java 语言中创建的对象是保存在堆中的，为了提高查找的速度而使用了散列查找。散列查找的基本思想是定义一个键来映射对象所在的内存地址。当需要查找对象时，直接查找键就可以了，这样就不用遍历整个堆来查找对象了。本范例将查看不同对象的散列值。运行结果如图 7.24 所示。（**实例位置：光盘\TM\sl\7\8**）

（1）在项目中创建 Cat 类，在类中定义 4 个成员变量分别表示猫咪的名字、年龄、重量和颜色，并提供构造方法来设置这些属性值。本类的重点内容在于重写 equals()方法和 hashCode()方法。重写 equals()方法可以比较两个对象是否相同，重写 hashCode()方法可以让相同的对象保存在相同的位置。代码如下：

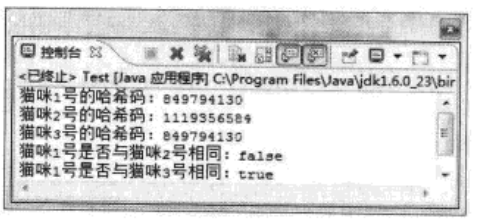


图 7.24　输出对象哈希码和比较结果

```
public class Cat {
    private String name;                                                  //表示猫咪的名字
    private int age;                                                      //表示猫咪的年龄
    private double weight;                                                //表示猫咪的重量
    private Color color;                                                  //表示猫咪的颜色
    public Cat(String name, int age, double weight, Color color) {        //初始化猫咪的属性
        this.name = name;
        this.age = age;
        this.weight = weight;
        this.color = color;
    }
    @Override
    public boolean equals(Object obj) {                                   //利用属性来判断猫咪是否相同
        if (this == obj) {                                                //如果两个猫咪是同一个对象则相同
            return true;
        }
        if (obj == null) {                                                //如果两个猫咪有一个为 null 则不同
            return false;
        }
```

```
        if (getClass() != obj.getClass()) {                          //如果两个猫咪的类型不同则不同
            return false;
        }
        Cat cat = (Cat) obj;
        return name.equals(cat.name) && (age == cat.age)
                && (weight == cat.weight) && (color.equals(cat.color));  //比较猫咪的属性
    }
    @Override
    public int hashCode() {                                          //重写 hashCode()方法
        return 7 * name.hashCode() + 11 * new Integer(age).hashCode() + 13
                * new Double(weight).hashCode() + 17 * color.hashCode();
    }
}
```

（2）在 com.mingrisoft 包中创建类文件，名称为 Test。在该类的 main()方法中创建了 3 只猫咪，并为其初始化，然后输出猫咪的哈希码和比较的结果。代码如下：

```
public class Test {
    public static void main(String[] args) {
        Cat cat1 = new Cat("Java", 12, 21, Color.BLACK);                  //创建猫咪 1 号
        Cat cat2 = new Cat("C++", 12, 21, Color.WHITE);                   //创建猫咪 2 号
        Cat cat3 = new Cat("Java", 12, 21, Color.BLACK);                  //创建猫咪 3 号
        System.out.println("猫咪 1 号的哈希码：" + cat1.hashCode());       //输出猫咪 1 号的哈希码
        System.out.println("猫咪 2 号的哈希码：" + cat2.hashCode());       //输出猫咪 2 号的哈希码
        System.out.println("猫咪 3 号的哈希码：" + cat3.hashCode());       //输出猫咪 3 号的哈希码
        System.out.println("猫咪 1 号是否与猫咪 2 号相同：" + cat1.equals(cat2));  //比较是否相同
        System.out.println("猫咪 1 号是否与猫咪 3 号相同：" + cat1.equals(cat3));  //比较是否相同
    }
}
```

7.4 经典范例

7.4.1 经典范例 1：汉诺塔问题求解

视频讲解：光盘\TM\lx\7\汉诺塔问题求解.exe

汉诺塔问题的描述如下：有 A、B 和 C 3 根柱子，在 A 上从下往上按照从小到大的顺序放着 64 个圆盘，以 B 为中介，把盘子全部移动到 C 上。移动过程中，要求任意盘子的下面要么没有盘子，要么只能有比它大的盘子。本范例将演示如何求解 3 阶汉诺塔问题。运行结果如图 7.25 所示。（**实例位置：光盘\TM\sl\7\9**）

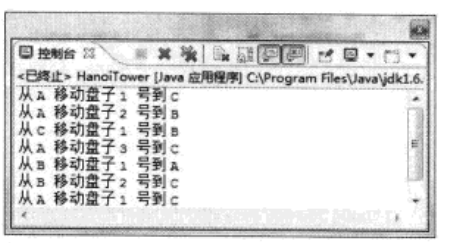


图 7.25 三阶汉诺塔解决步骤

在项目中创建 HanoiTower 类，在类中定义了一个 moveDish()方法，它使用递归算法完成汉诺塔问题的求解；一个 main()方法用于测试。代码如下：

```
public class HanoiTower {
    public static void moveDish(int level, char from, char inter, char to) {
        if (level == 1) {                                          //如果只有一个盘子就退出迭代
            System.out.println("从 " + from + " 移动盘子 1 号到 " + to);
        } else {                                                   //如果有大于一个盘子就继续迭代
            moveDish(level - 1, from, to, inter);
            System.out.println("从 " + from + " 移动盘子 " + level + " 号到 " + to);
            moveDish(level - 1, inter, from, to);
        }
    }
    public static void main(String[] args) {
        int nDisks = 3;                                            //设置汉诺塔为三阶
        moveDish(nDisks, 'A', 'B', 'C');                           //实现移动算法
    }
}
```

7.4.2 经典范例 2：单例模式的应用

视频讲解：光盘\TM\lx\7\单例模式的应用.exe

中国历史上的皇帝通常仅有一人。为了保障其唯一性，古人采用增加“防伪标识”的办法，如玉玺。更简单的方法是限制皇帝的创建。本范例使用单例模式来保证皇帝的唯一性。运行结果如图 7.26 所示。（**实例位置：光盘\TM\ sl\7\10**）

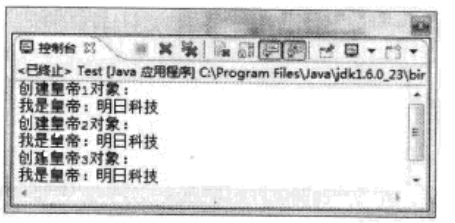


图 7.26 使用单例模式保证了皇帝的唯一性

（1）在项目中创建 Emperor 类，在类中将构造方法设置成私有，并提供了一个静态方法用于获得该类的实例。代码如下：

```
public class Emperor {
    private static Emperor emperor = null;                         //声明一个 Emperor 类的引用
    private Emperor() {                                            //将构造方法私有
    }
    public static Emperor getInstance() {                          //实例化引用
        if (emperor == null) {
            emperor = new Emperor();
        }
        return emperor;
    }
    public void getName() {                                        //使用普通方法输出皇帝的名字
        System.out.println("我是皇帝：明日科技");
    }
}
```

（2）在项目中创建 Test 类，在类的 main()方法中创建了 3 个 Emperor 对象并输出了其名字。代码如下：

```
public class Test {
    public static void main(String[] args) {
        System.out.println("创建皇帝 1 对象：");
        Emperor emperor1 = Emperor.getInstance();          //创建皇帝对象
        emperor1.getName();                                 //输出皇帝的名字
        System.out.println("创建皇帝 2 对象：");
        Emperor emperor2 = Emperor.getInstance();          //创建皇帝对象
        emperor2.getName();                                 //输出皇帝的名字
        System.out.println("创建皇帝 3 对象：");
        Emperor emperor3 = Emperor.getInstance();          //创建皇帝对象
        emperor3.getName();                                 //输出皇帝的名字
    }
}
```

7.5 本章小结

通过本章的学习，读者应该掌握面向对象的编程思想，这样对 Java 的学习十分有帮助，同时在此基础上读者可以编写类，定义类成员、构造方法、主方法以解决一些实际问题，类以及类成员可以使用权限修饰符进行修饰，读者应该了解这些修饰符的具体范围。由于在 Java 中通过对象来处理问题，所以对象的创建、比较、销毁的应用就显得非常重要。作为初学者应该反复揣摩这些基本概念和面向对象的编程思想，为以后 Java 语言的学习打下坚实的基础。

7.6 实战练习

1．尝试编写一个类，定义一个修饰权限为 private 的成员变量，定义两个成员方法，一个方法实现为此成员变量赋值，另一个成员方法获取这个成员变量的值，保证其他类继承该类时能获取该类的成员变量的值。（**答案位置：光盘\TM\sl\7\11**）

2．尝试编写一个矩形类，将长与宽作为矩形类的属性，在构造方法中将长、宽初始化，定义一个成员方法求此矩形的面积。（**答案位置：光盘\TM\sl\7\12**）

3．根据运行参数的个数决定循环打印变量 i 值的次数。（**答案位置：光盘\TM\sl\7\13**）

第2篇

技术篇

- 第 8 章　接口、继承与多态
- 第 9 章　类的高级特性
- 第 10 章　Java 集合类
- 第 11 章　异常处理
- 第 12 章　输入/输出
- 第 13 章　Swing 程序设计

本篇介绍了接口、继承与多态，类的高级特性，Java 集合类，异常处理，输入/输出，Swing 程序设计等。学习完这一部分后，能够开发一些小型应用程序。

PDG

PDG

第 8 章

接口、继承与多态

（视频讲解：18 分钟）

继承和多态是面向对象开发语言中非常重要的一个环节，如果在程序中使用得当，可以将整个程序的架构变得非常有弹性，同时可以减少代码的冗余性。继承机制的使用可以复用一些定义好的类，减少重复代码的编写。多态机制的使用可以动态调整对象的调用，降低对象之间的依存关系。同时为了优化继承与多态，一些类除了继承父类还使用接口的形式。Java 语言中的类可以同时实现多个接口，接口被用来建立类与类之间关联的标准。在 Java 中正因为使用这些机制使 Java 语言更具有生命力。

通过阅读本章，您可以：

- 掌握接口的使用
- 掌握类的继承
- 掌握 super 关键字的使用方法
- 了解什么是多态

PDG

8.1 接口的使用

视频讲解：光盘\TM\lx\8\接口的使用.exe

Java 语言只支持单重继承，不支持多继承，即一个类只能有一个父类。但是在实际应用中，又经常需要使用多继承来解决问题。为了解决该问题，Java 语言提供了接口来实现类的多重继承功能。

8.1.1 接口的定义

使用 interface 来定义一个接口。接口定义与类的定义类似，也是分为接口的声明和接口体，其中接口体由变量定义和方法定义两部分组成。定义接口的基本语法格式如下：

```
[修饰符] interface 接口名 [extends 父接口名列表]{
    [public] [static] [final] 变量;
    [public] [abstract] 方法;
}
```

定义接口的语法格式的参数说明如表 8.1 所示。

表 8.1 定义接口的语法格式的参数说明

参　数	说　明
修饰符	可选参数，用于指定接口的访问权限，可选值为 public。如果省略则使用默认的访问权限
接口名	必选参数，用于指定接口的名称，接口名必须是合法的 Java 标识符。一般情况下，要求首字母大写
extends 父接口名列表	可选参数，用于指定要定义的接口继承于哪个父接口。当使用 extends 关键字时，父接口名为必选参数
方法	接口中的方法只有定义而没有被实现

【例 8.1】 定义一个用于计算的接口，在该接口中定义一个常量 PI 和两个方法。

```
public interface ICalculate {
    final float PI=3.14159f;                    //定义用于表示圆周率的常量 PI
    float getArea(float r);                     //定义一个用于计算面积的方法 getArea()
    float getCircumference(float r);            //定义一个用于计算周长的方法 getCircumference()
}
```

注意

Java 的类文件一样，接口文件的文件名必须与接口名相同。

8.1.2 接口的实现

接口在定义后，就可以在类中实现该接口。在类中实现接口可以使用 implements 关键字，基本语

法格式如下：

```
[修饰符] class <类名> [extends  父类名] [implements  接口列表] {
}
```

实现接口的语法格式的参数说明如表 8.2 所示。

表 8.2　实现接口的语法格式的参数说明

参　　数	说　　明
修饰符	可选参数，用于指定类的访问权限，可选值为 public、abstract 和 final
类名	必选参数，用于指定类的名称，类名必须是合法的 Java 标识符。一般情况下，要求首字母大写
extends 父类名	可选参数，用于指定要定义的类继承于哪个父类。当使用 extends 关键字时，父类名为必选参数
implements 接口列表	可选参数，用于指定该类实现的是哪些接口。当使用 implements 关键字时，接口列表为必选参数。当接口列表中存在多个接口名时，各个接口名之间使用逗号分隔

在类中实现接口时，方法名、返回值类型、参数的个数及类型必须与接口中的完全一致，并且必须实现接口中的所有方法。

【例 8.2】　编写一个名称为 Cire 的类，该类实现例 8.1 中定义的接口 ICalculate。

```
public class Cire implements ICalculate {
    //定义计算圆面积的方法
    public float getArea(float r) {
        float area=PI*r*r;                    //计算圆面积并赋值给变量 area
        return area;                          //返回计算后的圆面积
    }
    //定义计算圆周长的方法
    public float getCircumference(float r) {
        float circumference=2*PI*r;           //计算圆周长并赋值给变量 circumference
        return circumference;                 //返回计算后的圆周长
    }
}
```

在类的继承中，只能做单重继承，而实现接口时，一次则可以实现多个接口，每个接口间使用逗号“,”分隔，这时就可能出现变量或方法名冲突的情况。解决该问题时，如果变量冲突，则需要明确指定变量的接口，可以通过“接口名.变量”实现。如果出现方法冲突，则只要实现一个方法即可。下面通过一个具体的实例详细介绍以上问题的解决方法。

8.1.3　范例 1：图片的不同格式保存

在使用图像处理软件处理图片后，需要选择一种格式进行保存。本范例运用接口对图片用不同格式进行保存。运行结果如图 8.1 所示。（**实例位置：光盘\TM\sl\8\1**）

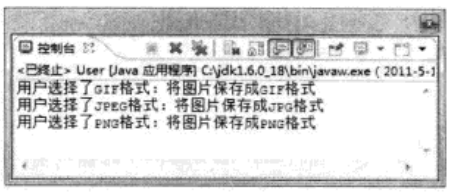


图 8.1　图片的不同格式保存

（1）编写 ImageSaver 接口，在该接口中定义 save()方法。代码如下：

```
public interface ImageSaver {
    void save();                                    //定义 save()方法
}
```

（2）在项目中创建 GIFSaver 类，该类实现了 ImageSaver 接口，在实现 save()方法时将图片保存为 GIF 格式。代码如下：

```
public class GIFSaver implements ImageSaver {
    @Override
    public void save() {                            //实现 save()方法
        System.out.println("将图片保存成 GIF 格式");
    }
}
```

8.1.4 范例 2：为汽车增加 GPS 定位功能

对于刚从工厂生产出来的商品，有些功能并不能完全满足用户的需要，因此，用户通常会对其进行一定的改装工作。本范例将为普通的汽车增加 GPS 定位功能。运行结果如图 8.2 所示。（**实例位置：光盘\TM\ sl\8\2**）

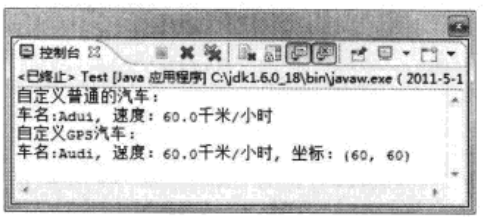


图 8.2 为汽车增加 GPS 定位功能

（1）在项目中创建 Car 类，在该类中首先定义两个属性，一个是 name（表示汽车的名字），另一个是 speed（表示汽车的速度），并为其提供了 getXXX()和 setXXX()方法，然后通过重写 toString()方法来方便输出 Car 对象。代码如下：

```
public class Car {
    private String name;                            //表示汽车的名称
    private double speed;                           //表示汽车的速度
    //省略 getXXX()和 setXXX()方法
    @Override
    public String toString() {                      //重写 toString()方法
        StringBuilder sb = new StringBuilder();
        sb.append("车名：" + name + ", ");
        sb.append("速度：" + speed + "千米/小时");
        return sb.toString();
    }
}
```

（2）在项目中编写接口 GPS，在该接口中定义了 getLocation()方法，用来确定汽车的位置。代码如下：

```
public interface GPS {
    Point getLocation();                            //提供定位功能
}
```

（3）在项目中编写 GPSCar 类，该类继承 Car 并实现 GPS 接口。在该类中首先实现 getLocation()方法，用于实现确定汽车位置的功能，然后重写 toString()方法方便输出 GPSCar 对象。代码如下：

```
public class GPSCar extends Car implements GPS {
    @Override
    public Point getLocation() {                                    //利用汽车的速度来确定汽车的位置
        Point point = new Point();
        point.setLocation(super.getSpeed(), super.getSpeed());
        return point;
    }
    @Override
    public String toString() {                                      //重写 toString()方法
        StringBuilder sb = new StringBuilder();
        sb.append(super.toString());
        sb.append(", 坐标：(" + getLocation().x + ", " + getLocation().y + ")");
        return sb.toString();
    }
}
```

8.2　类 的 继 承

视频讲解：光盘\TM\lx\8\类的继承.exe

继承一般是指晚辈从父辈那里继承财产，也可以说是子女拥有父母所给予他们的东西。在面向对象程序设计中，继承的含义与此类似，所不同的是，这里继承的实体是类而非人。也就是说继承是子类拥有父类的成员。下面将介绍在 Java 语言中，如何实现类的继承。

8.2.1　继承的实现

在 Java 语言中，继承通过 extends 关键字来实现。也就是用 extends 指明当前类是子类，并指明从哪个类继承而来。即在子类的声明中，通过使用 extends 关键字来显式地指明其父类。其基本的声明格式如下：

```
[修饰符] class 子类名 extends 父类名{
    类体
}
```

- ☑ 修饰符：可选参数，用于指定类的访问权限，可选值为 public、abstract 和 final。
- ☑ 子类名：必选参数，用于指定子类的名称，类名必须是合法的 Java 标识符。一般情况下，要求首字母大写。
- ☑ extends 父类名：必选参数，用于指定要定义的子类继承于哪个父类。

【例 8.3】 定义一个 Pigeon 类，该类继承于父类 Bird，即 Pigeon 类是 Bird 类的子类。（**实例位置：光盘\TM\sl\8\3**）

父类 Bird 的代码如下：

```
public class Bird {
    String color="白色";                    //颜色
    String skin="羽毛";                     //皮毛
}
```

子类 Pigeon 的代码如下：

```
public class Pigeon extends Bird {
    public static void main(String[] args) {
        Pigeon pigeon=new Pigeon();
        System.out.println(pigeon.color);   //输出成员变量 color
    }
}
```

运行结果如图 8.3 所示。

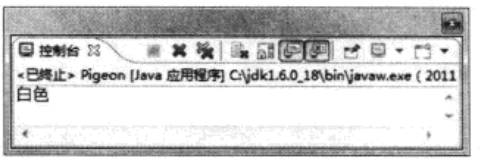


图 8.3 在子类中输出父类的成员变量

8.2.2 继承中的重写

重写是指父子类之间的关系，当子类继承父类中所有可能被子类访问的成员方法时，如果子类的方法名与父类的方法名相同，那么子类就不能继承父类的方法，此时，称为子类的方法重写了父类的方法。重写体现了子类补充或者改变父类方法的能力。通过重写，可以使一个方法在不同的子类中表现出不同的行为。

说明

重写也可以称为覆盖。

【例 8.4】 定义一个动物类 Animal 及它的子类，并在其子类中重写父类的相关方法。（**实例位置：光盘\TM\sl\8\4**）

（1）创建一个名称为 Animal 的类，在该类中声明一个成员方法 cry()。代码如下：

```
public class Animal {
    public Animal() {
    }
    public void cry() {
        System.out.println("动物发出叫声！");
    }
}
```

（2）创建一个 Animal 类的子类 Dog，在该类中重写了父类的成员方法 cry()。代码如下：

```
public class Dog extends Animal {
    public Dog() {
    }
    public void cry() {
        System.out.println("狗发出“汪汪……”声！");
    }
}
```

（3）创建一个 Animal 类的子类 Cat，在该类中重写了父类的成员方法 cry()。代码如下：

```
public class Cat extends Animal {
    public Cat() {
    }
    public void cry() {
        System.out.println("猫发出“喵喵……”声！");
    }
}
```

（4）创建一个 Animal 类的子类 Sheep，在该类中不定义任何方法。代码如下：

```
public class Sheep extends Animal {
}
```

（5）创建一个名称为 Zoo 的类，在该类的 main()方法中分别创建子类 Dog、Cat 和 Sheep 的对象并为该对象分配内存，然后分别调用各对象的 cry()方法。代码如下：

```
public class Zoo {
    public static void main(String[] args) {
        Dog dog=new Dog();                                    //创建 Dog 类的对象并为其分配内存
        System.out.println("执行 dog.cry();语句时的输出结果：");
        dog.cry();
        Cat cat=new Cat();                                    //创建 Cat 类的对象并为其分配内存
        System.out.println("执行 cat.cry();语句时的输出结果：");
        cat.cry();
        Sheep sheep=new Sheep();                              //创建 Sheep 类的对象并为其分配内存
        System.out.println("执行 sheep.cry();语句时的输出结果：");
        sheep.cry();
    }
}
```

运行结果如图 8.4 所示。

从上面的运行结果中可以看出，由于 Dog 类和 Cat 类都重写了父类的方法 cry()，所以执行的是子类中的 cry()方法，但是 Sheep 类没有重写父类的方法，所以执行的是父类中的 cry()方法。

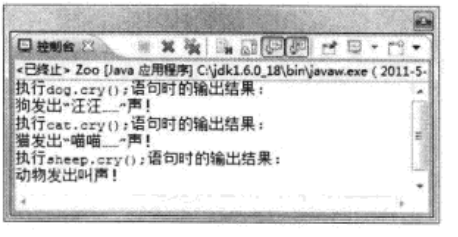


图 8.4　在子类中重写父类的相关方法

8.2.3　使用 super 关键字

子类可以继承父类的非私有成员变量和成员方法（不是以 private 关键字修饰的）作为自己的成员

变量和成员方法。但是，如果子类中声明的成员变量与父类的成员变量同名，则子类不能继承父类的成员变量，此时称子类的成员变量隐藏了父类的成员变量。如果子类中声明的成员方法与父类的成员方法同名，并且方法的返回值及参数个数和类型也相同，则子类不能继承父类的成员方法，此时称子类的成员方法重写了父类的成员方法。这时，如果想在子类中访问父类中被子类隐藏的成员方法或变量，就可以使用 super 关键字。super 关键字主要有以下两种用途。

1．调用父类的构造方法

子类可以调用由父类声明的构造方法。但是必须在子类的构造方法中使用 super 关键字来调用。语法格式如下：

```
super([参数列表]);
```

如果父类的构造方法中包括参数，则参数列表为必选项，用于指定父类构造方法的入口参数。

【例 8.5】 在子类中调用父类的构造方法。（**实例位置：光盘\TM\sl\8\5**）

（1）在项目中创建 Beast 类，在类中添加一个默认的构造方法和一个带参数的构造方法。代码如下：

```
public class Beast {
    String skin = "";                          //成员变量
    public Beast() {                           //默认构造方法
    }
    public Beast(String strSkin) {             //带参数的构造方法
        skin = strSkin;
    }
    public void move() {                       //成员方法
        System.out.println("跑");
    }
}
```

（2）如果想在子类 Tiger 中使用父类的带参数的构造方法，则需要在子类 Tiger 的构造方法中进行调用。代码如下：

```
public class Tiger extends Beast {
    public Tiger (){
        super("条纹");                         //使用父类的带参数的构造方法
    }
}
```

2．操作被隐藏的成员变量和被重写的成员方法

如果想在子类中操作父类中被隐藏的成员变量和被重写的成员方法，也可以使用 super 关键字。语法格式如下：

```
super.成员变量名
super.成员方法名([参数列表])
```

例如，在例 8.5 中，如果想在子类 Tiger 的方法中改变父类 Beast 的成员变量 skin 的值，可以使用

以下代码：

```
super.skin="条纹";
```

如果想在子类 Tiger 的方法中使用父类 Beast 的成员方法 move()，可以使用以下代码：

```
super.move();
```

8.2.4　范例 3：经理与员工的差异

对于在同一家公司工作的经理和员工而言，两者是有很多共同点的。例如每个月都要发工资，但是经理在完成目标任务后，还会获得奖金。此时，利用员工类来编写经理类就会少写很多代码，利用继承技术可以让经理类使用员工类中定义的属性和方法。本范例将通过继承演示经理与员工的差异。运行结果如图 8.5 所示。（**实例位置：光盘\TM\sl\8\6**）

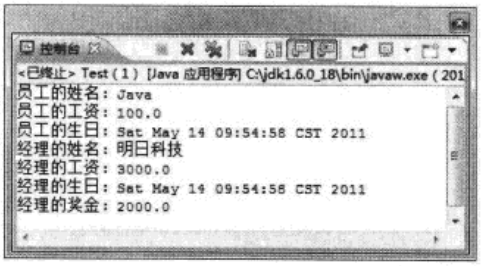


图 8.5　经理与员工的差异

（1）在项目中创建 Employee 类，在该类中定义 3 个属性，分别是 name（表示员工的姓名）、salary（表示员工的工资）和 birthday（表示员工的生日），并分别为它们定义了 getXXX()和 setXXX()方法。代码如下：

```
import java.util.Date;
public class Employee {
    private String name;                          //员工的姓名
    private double salary;                        //员工的工资
    private Date birthday;                        //员工的生日

    public String getName() {                     //获取员工的姓名
        return name;
    }
    public void setName(String name) {            //设置员工的姓名
        this.name = name;
    }

    public double getSalary() {                   //获取员工的工资
        return salary;
    }
    public void setSalary(double salary) {        //设置员工的工资
        this.salary = salary;
    }
    public Date getBirthday() {                   //获取员工的生日
        return birthday;
    }
    public void setBirthday(Date birthday) {      //设置员工的生日
```

```
        this.birthday = birthday;
    }
}
```

（2）在项目中创建一个名称为 Manager 的类，该类继承自 Employee。在该类中，定义了一个 bonus 域，表示经理的奖金，并为其设置了 getXXX()和 setXXX()方法。代码如下：

```
public class Manager extends Employee {
    private double bonus;                          //经理的奖金
    public double getBonus() {                     //获得经理的奖金
        return bonus;
    }
    public void setBonus(double bonus) {           //设置经理的奖金
        this.bonus = bonus;
    }
}
```

（3）在项目中再创建一个名称为 Test 的类，用于测试。在该类中分别创建 Employee 和 Manager 对象，并为其赋值，然后输出其属性。代码如下：

```
import java.util.Date;                             //导入 java.util.Date 类
public class Test {
    public static void main(String[] args) {
        Employee employee = new Employee();        //创建 Employee 对象并为其赋值
        employee.setName("Java");
        employee.setSalary(100);
        employee.setBirthday(new Date());
        Manager manager = new Manager();           //创建 Manager 对象并为其赋值
        manager.setName("明日科技");
        manager.setSalary(3000);
        manager.setBirthday(new Date());
        manager.setBonus(2000);
        //输出经理和员工的属性值
        System.out.println("员工的姓名：" + employee.getName());
        System.out.println("员工的工资：" + employee.getSalary());
        System.out.println("员工的生日：" + employee.getBirthday());
        System.out.println("经理的姓名：" + manager.getName());
        System.out.println("经理的工资：" + manager.getSalary());
        System.out.println("经理的生日：" + manager.getBirthday());
        System.out.println("经理的奖金：" + manager.getBonus());
    }
}
```

8.2.5 范例 4：重写父类中的方法

在继承了一个类之后，就可以使用父类中定义的方法，然而父类中的方法可能并不完全适用于子

类。此时，如果不想定义新的方法，则可以重写父类中的方法。本范例将演示如何重写父类中的方法。运行结果如图 8.6 所示。（**实例位置：光盘\TM\sl\8\7**）

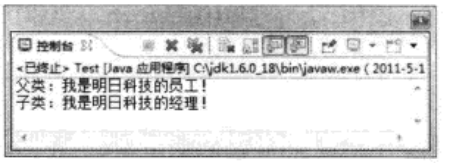


图 8.6　重写父类中的方法

（1）在项目中创建 Employee 类，在该类中添加 getInfo()方法，返回值为字符串“父类：我是明日科技的员工！”。代码如下：

```
public class Employee {
    public String getInfo() {                          //定义测试用的方法
        return "父类：我是明日科技的员工！";
    }
}
```

（2）在 com.mingrisoft 包中再创建一个名称为 Manager 的类，该类继承自 Employee。在该类中，重写 getInfo()方法。代码如下：

```
public class Manager extends Employee {
    @Override
    public String getInfo() {                          //重写测试用的方法
        return "子类：我是明日科技的经理！";
    }
}
```

8.3　多　　态

视频讲解：光盘\TM\lx\8\多态.exe

8.3.1　什么是多态

多态性是面向对象程序设计的重要部分。在 Java 语言中，通常使用方法的重载（Overloading）和重写（Overriding）实现类的多态性。其中，重写已经在前面介绍过，下面将对方法的重载进行介绍。

说明　重写之所以具有多态性，是因为父类的方法在子类中被重写，子类和父类的方法名称相同，但完成的功能却不一样，所以说，重写也具有多态性。

方法的重载是指在一个类中出现多个方法名相同，但参数个数或参数类型不同的方法，则称为方法的重载。Java 语言在执行具有重载关系的方法时，将根据调用参数的个数和类型区分具体执行的是哪个方法。下面将通过一个具体的实例进行说明。

【例 8.6】　定义一个名称为 Calculate 的类，在该类中定义两个名称为 getArea()的方法（参数个数不同）和两个名称为 draw()的方法（参数类型不同）。（**实例位置：光盘\TM\sl\8\8**）

```
public class Calculate {
    final float PI=3.14159f;                          //定义一个用于表示圆周率的常量 PI
    //求圆形的面积
    public float getArea(float r){                    //定义一个用于计算面积的方法 getArea()
        float area=PI*r*r;
        return area;
    }
    //求矩形的面积
    public float getArea(float l,float w){            //重载 getArea()方法
        float area=l*w;
        return area;
    }
    //画任意形状的图形
    public void draw(int num){                        //定义一个用于画图的方法 draw()
        System.out.println("画"+num+"个任意形状的图形");
    }
    //画指定形状的图形
    public void draw(String shape){                   //重载 draw()方法
        System.out.println("画一个"+shape);
    }
    public static void main(String[] args) {
        Calculate calculate=new Calculate();          //创建 Calculate 类的对象并为其分配内存
        float l=20;
        float w=30;
        float areaRectangle=calculate.getArea(l, w);
        System.out.println("求长为"+l+" 宽为"+w+"的矩形的面积是："+areaRectangle);
        float r=7;
        float areaCirc=calculate.getArea(r);
        System.out.println("求半径为"+r+"的圆的面积是："+areaCirc);
        int num=7;
        calculate.draw(num);
        calculate.draw("三角形");
    }
}
```

运行结果如图 8.7 所示。

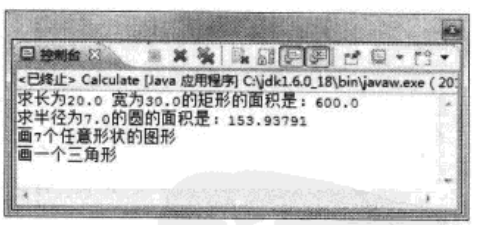


图 8.7　演示方法的重载

说明 重载的方法之间并不一定必须有联系，但是为了提高程序的可读性，一般只重载功能相似的方法。

注意

在进行方法的重载时，方法返回值的类型不能作为区分方法的标志。

8.3.2　范例 5：计算几何图形的面积

对于每个几何图形而言，都有一些共同的属性，如名字和面积等，而其计算面积的方法却各不相同。为了简化开发，本范例将定义一个超类来实现输出名字的方法，并使用抽象方法来计算面积。运行结果如图 8.8 所示。（**实例位置：光盘\TM\sl\8\9**）

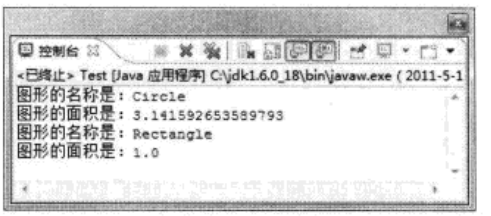


图 8.8　计算几何图形的面积

（1）在项目中创建一个抽象类，名称为 Shape。在该类中定义两个方法，一个是 getName()，用于使用反射机制获得类名称；另一个是抽象方法 getArea()，并未实现。代码如下：

```
public abstract class Shape {
    public String getName() {                          //获得图形的名称
        return this.getClass().getSimpleName();
    }
    public abstract double getArea();                  //获得图形的面积
}
```

（2）在项目中创建一个名称为 Circle 的类，该类继承自 Shape，并实现了抽象方法 getArea()。在该类的构造方法中，获得了圆形的半径，用于在 getArea()方法中计算面积。代码如下：

```
public class Circle extends Shape {
    private double radius;
    public Circle(double radius) {                     //获得圆形的半径
        this.radius = radius;
    }
    @Override
    public double getArea() {                          //计算圆形的面积
        return Math.PI * Math.pow(radius, 2);
    }
}
```

（3）在项目中创建一个名称为 Rectangle 的类，该类继承自 Shape，并实现了抽象方法 getArea()。在该类的构造方法中，获得了矩形的长和宽，用于在 getArea()方法中计算面积。代码如下：

```
public class Rectangle extends Shape {
    private double length;
    private double width;
    public Rectangle(double length, double width) {    //获得矩形的长和宽
```

```
        this.length = length;
        this.width = width;
    }
    @Override
    public double getArea() {                                    //计算矩形的面积
        return length * width;
    }
}
```

（4）在项目中创建一个名称为 Test 的类，用来进行测试，在该类中创建 Circle 和 Rectangle 对象，并分别输出图形的名称和面积。代码如下：

```
public class Test {
    public static void main(String[] args) {
        Circle circle = new Circle(1);                           //创建圆形对象并将半径设置成 1
        System.out.println("图形的名称是：" + circle.getName());
        System.out.println("图形的面积是：" + circle.getArea());
        Rectangle rectangle = new Rectangle(1, 1);               //创建矩形对象并将长和宽设置成 1
        System.out.println("图形的名称是：" + rectangle.getName());
        System.out.println("图形的面积是：" + rectangle.getArea());
    }
}
```

8.3.3 范例 6：简单的汽车销售商场

当顾客在商场购物时，卖家需要根据顾客的需求提取商品。对于汽车销售商场也是如此。用户需要先指定购买的车型，然后商家去提取该车型的汽车。本范例将实现一个简单的汽车销售商场，用来演示多态的用法。运行结果如图 8.9 所示。（**实例位置：光盘\TM\sl\8\10**）

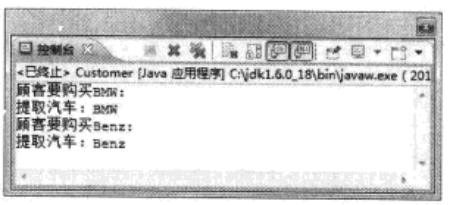


图 8.9 简单的汽车销售商场

（1）在项目中创建一个抽象类，名称为 Car，在该类中定义一个抽象方法 getInfo()。代码如下：

```
public abstract class Car {
    public abstract String getInfo();                            //用来描述汽车的信息
}
```

（2）在项目中创建一个名称为 BMW 的类，该类继承自 Car 并实现了其 getInfo()方法。代码如下：

```
public class BMW extends Car {
    @Override
    public String getInfo() {                                    //用来描述汽车的信息
        return "BMW";
    }
}
```

（3）在项目中创建一个名称为 Benz 的类，该类继承自 Car 并实现了其 getInfo()方法。代码如下：

```
public class Benz extends Car {
    @Override
    public String getInfo() {                                   //用来描述汽车的信息
        return "Benz";
    }
}
```

（4）在项目中创建一个名称为 CarFactory 的类，该类定义了一个静态方法 getCar()，它可以根据用户指定的车型来创建对象。代码如下：

```
public class CarFactory {
    public static Car getCar(String name) {
        if (name.equalsIgnoreCase("BMW")) {                     //如果需要 BMW 则创建 BMW 对象
            return new BMW();
        } else if (name.equalsIgnoreCase("Benz")) {             //如果需要 Benz 则创建 Benz 对象
            return new Benz();
        } else {                                                //暂时不能支持其他车型
            return null;
        }
    }
}
```

（5）在项目中创建一个名称为 Customer 的类，用来进行测试，在 main()方法中，根据用户的需要提取了不同的汽车。代码如下：

```
public class Customer {
    public static void main(String[] args) {
        System.out.println("顾客要购买 BMW：");
        Car bmw = CarFactory.getCar("BMW");                     //用户要购买 BMW
        System.out.println("提取汽车：" + bmw.getInfo());        //提取 BMW
        System.out.println("顾客要购买 Benz：");
        Car benz = CarFactory.getCar("Benz");                   //用户要购买 Benz
        System.out.println("提取汽车：" + benz.getInfo());       //提取 Benz
    }
}
```

8.4 经典范例

8.4.1 经典范例 1：使用 Comparable 接口自定义排序

视频讲解：光盘\TM\lx\8\使用 Comparable 接口自定义排序.exe

默认情况下，保存在 List 集合中的数组是不进行排序的，但可以通过使用 Comparable 接口自定义

排序规则来自动排序。本范例将介绍如何使用 Comparable 接口自定义排序规则并自动排序。运行结果如图 8.10 所示。（**实例位置：光盘\TM\sl\8\11**）

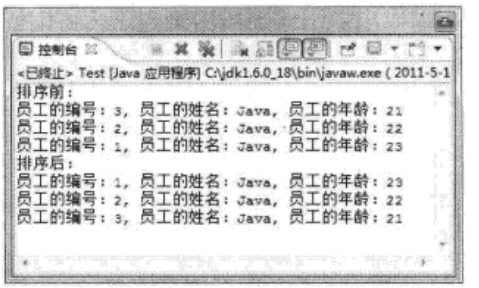


图 8.10　使用 Comparable 接口自定义排序

在项目中创建 Employee 类，在该类中首先定义 3 个属性，分别为 id（表示员工的编号）、name（表示员工的姓名）和 age（表示员工的年龄），然后在构造方法中初始化这 3 个属性，最后再实现接口中定义的 compareTo()方法，将对象按编号升序排列。代码如下：

```
public class Employee implements Comparable<Employee> {
    private int id;                                          //员工的编号
    private String name;                                     //员工的姓名
    private int age;                                         //员工的年龄
    public Employee(int id, String name, int age) {          //利用构造方法初始化各个域
        this.id = id;
        this.name = name;
        this.age = age;
    }
    @Override
    public int compareTo(Employee o) {                       //利用编号实现对象间的比较
        if (id > o.id) {
            return 1;
        } else if (id < o.id) {
            return -1;
        }
        return 0;
    }
    @Override
    public String toString() {                               //重写 toString()方法
        StringBuilder sb = new StringBuilder();
        sb.append("员工的编号：" + id + ", ");
        sb.append("员工的姓名：" + name + ", ");
        sb.append("员工的年龄：" + age);
        return sb.toString();
    }
}
```

8.4.2　经典范例 2：动态设置类的私有域

视频讲解：光盘\TM\lx\8\动态设置类的私有域.exe

为了保证面向对象的封装特性，通常会将域设置成私有的，然后提供对应的 getXXX()和 setXXX()方法。对于非内部类而言，只能使用 getXXX()和 setXXX()方法来操作该域。然而利用反射机制，就可以在运行时修改类的私有域。本范例将通过简单的 Student 类来演示反射的这种用法。运行结果如图 8.11 所示。（**实例位置：光盘\TM\sl\8\12**）

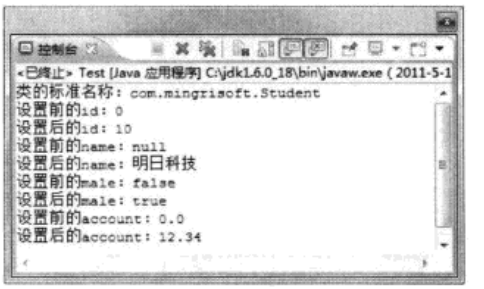


图 8.11　动态设置类的私有域

（1）在项目中创建 Student 类，在该类中定义 4 个域及其对应的 getXXX()和 setXXX()方法。这 4 个域分别是 id（表示学生的序号）、name（表示学生的姓名）、male（表示学生是否为男性）和 account（表示学生的账户余额）。代码如下：

```
public class Student {
    private int id;                                    //表示学生的序号
    private String name;                               //表示学生的姓名
    private boolean male;                              //表示学生的性别
    private double account;                            //表示学生的账户余额
    //省略了各个域的 getXXX()和 setXXX()方法
}
```

（2）在 com.mingrisoft 包中编写 Test 类进行测试。在 main()方法中，分别为不同的域设置不同的值，并输出初始值和新值作为对比。代码如下：

```
public class Test {
    public static void main(String[] args) {
        Student student = new Student();
        Class<?> clazz = student.getClass();                    //获得代表 student 对象的 Class 对象
        System.out.println("类的标准名称：" + clazz.getCanonicalName());
        try {
            Field id = clazz.getDeclaredField("id");
            System.out.println("设置前的 id：" + student.getId());
            id.setAccessible(true);
            id.setInt(student, 10);                             //设置 id 值为 10
            System.out.println("设置后的 id：" + student.getId());

            Field name = clazz.getDeclaredField("name");
            System.out.println("设置前的 name：" + student.getName());
            name.setAccessible(true);
            name.set(student, "明日科技");                      //设置 name 值为明日科技
            System.out.println("设置后的 name：" + student.getName());

            Field male = clazz.getDeclaredField("male");
```

```
            System.out.println("设置前的 male：" + student.isMale());
            male.setAccessible(true);
            male.setBoolean(student, true);                    //设置 male 值为 true
            System.out.println("设置后的 male：" + student.isMale());

            Field account = clazz.getDeclaredField("account");
            System.out.println("设置前的 account：" + student.getAccount());
            account.setAccessible(true);
            account.setDouble(student, 12.34);                 //设置 account 值为 12.34
            System.out.println("设置后的 account：" + student.getAccount());
        } catch (SecurityException e) {
            e.printStackTrace();
        } catch (NoSuchFieldException e) {
            e.printStackTrace();
        } catch (IllegalArgumentException e) {
            e.printStackTrace();
        } catch (IllegalAccessException e) {
            e.printStackTrace();
        }
    }
}
```

8.5 本章小结

通过本章的学习，读者可以了解继承与多态的机制，掌握重载、类型转换等技术，学会使用接口与抽象类，从而对继承和多态有一个比较深入的了解。尽管读者已经学习过本章，但笔者还是建议初学者仔细揣摩继承与多态机制，因为继承和多态本身是比较抽象的概念问题，深入理解需要一段时间，使用多态机制必须扩展自己的编程视野，应该将编程的着眼点放在类与类之间的共同特性以及关系上，这样将为软件开发带来更快的速度、更完善的代码组织架构、更好的扩展性和维护性。

8.6 实战练习

1．创建一个抽象类，验证它是否可以实例化对象。（**答案位置：光盘\TM\sl\8\13**）

2．尝试创建一个父类，在父类中创建两个方法，在子类中覆盖第二个方法，为子类创建一个对象，将它向上转型到基类并调用这个方法。（**答案位置：光盘\TM\sl\8\14**）

3．尝试创建一个父类和子类，分别创建构造方法，然后向父类和子类添加成员变量和方法，并总结构建子类对象时的顺序。（**答案位置：光盘\TM\sl\8\15**）

第 9 章

类的高级特性

（视频讲解：34 分钟）

类除了具有普通的特性之外，还具有一些高级特性，如包、内部类等。包在整个管理中起到了非常重要的作用，使用包可以有效地管理繁杂的类文件，解决类重名问题，当在类中配合包与权限修饰符使用时，可以控制其他人对类成员的访问。同时在 Java 语言中一个更为有效的隐藏实现细节的技巧是使用内部类，通过使用内部类机制可以向上转型为被内部类实现的公共接口。由于在类中可以定义多个内部类，所以实现接口的方式也不止一个，只要将内部类中的方法设置为类最小范围的修饰权限即可将内部类的实现细节有效地隐藏。

通过阅读本章，您可以：

- 掌握抽象类的使用
- 掌握内部类的分类
- 掌握成员内部类的使用
- 掌握局部内部类的使用
- 掌握匿名内部类的使用
- 掌握静态内部类的使用
- 掌握内部类的继承
- 掌握 Class 类与 Java 反射

PDG

9.1 抽 象 类

视频讲解：光盘\TM\lx\9\抽象类.exe

所谓抽象类就是只声明方法的存在而不去具体实现它的类。抽象类不能被实例化，也就是不能创建其对象。在定义抽象类时，要在 class 关键字前面加上 abstract 关键字。定义抽象类，语法格式如下：

```
abstract class 类名 {
    类体
}
```

【例 9.1】 定义一个名称为 Fruit 的抽象类。

```
abstract class Fruit {                          //定义抽象类
    public String color;                        //定义颜色成员变量
    //定义构造方法
    public Fruit() {
        color="绿色";                           //对变量 color 进行初始化
    }
}
```

在抽象类中创建的，没有实际意义的，必须要子类重写的方法称为抽象方法。抽象方法只有方法的声明，而没有方法的实现，用 abstract 关键字进行修饰。声明一个抽象方法的基本格式如下：

```
abstract <方法返回值类型> 方法名(参数列表);
```

- ☑ 方法返回值类型：必选参数，用于指定方法的返回值类型，如果该方法没有返回值，可以使用关键字 void 进行标识。方法返回值的类型可以是任何 Java 数据类型。
- ☑ 方法名：必选参数，用于指定抽象方法的名称，方法名必须是合法的 Java 标识符。
- ☑ 参数列表：可选参数，用于指定方法中所需的参数。当存在多个参数时，各参数之间应使用逗号分隔。方法的参数可以是任何 Java 数据类型。

例如，在例 9.1 中定义的抽象类中添加一个抽象方法，代码如下：

```
//定义抽象方法
public abstract void harvest();                 //收获的方法
```

注意

抽象方法不能使用 private 或 static 关键字进行修饰。

包含一个或多个抽象方法的类必须被声明为抽象类。因为抽象方法没有定义方法的实现部分，如果不声明为抽象类，这个类将可以生成对象，这时当用户调用抽象方法时，程序就不知道如何处理了。

【例 9.2】 本实例主要实现定义一个水果类 Fruit，该类为抽象类，并在该类中定义一个抽象方法，同时在其子类中通过重写的方法实现该抽象方法。（**实例位置：光盘\TM\sl\9\1**）

（1）使用 abstract 关键字创建抽象类 Fruit，在该类中定义相应的变量和方法。代码如下：

```
abstract class Fruit {                          //定义抽象类
    public String color;                        //定义颜色成员变量
    //定义构造方法
    public Fruit(){
        color="绿色";                           //对变量 color 进行初始化
    }
    //定义抽象方法
    public abstract void harvest();             //收获的方法
}
```

（2）创建 Fruit 类的子类 Apple，并实现 harvest()方法。代码如下：

```
public class Apple extends Fruit {
    @Override
    public void harvest() {
        System.out.println("苹果已经收获！");      //输出字符串“苹果已经收获！”
    }
}
```

（3）创建一个 Fruit 类的子类 Orange，并实现 harvest()方法。代码如下：

```
public class Orange extends Fruit {
    @Override
    public void harvest() {
        System.out.println("桔子已经收获！");      //输出字符串“桔子已经收获！”
    }
}
```

（4）创建一个包含 main()方法的公共类 Farm，在该类中执行 Fruit 类的两个子类的 harvest()方法。代码如下：

```
public class Farm {
    public static void main(String[] args) {
        System.out.println("调用 Apple 类的 harvest()方法的结果：");
        Apple apple=new Apple();                //声明 Apple 类的一个对象 apple，并为其分配内存
        apple.harvest();                        //调用 Apple 类的 harvest()方法
        System.out.println("调用 Orange 类的 harvest()方法的结果：");
        Orange orange=new Orange();             //声明 Orange 类的一个对象 orange，并为其分配内存
        orange.harvest();                       //调用 Orange 类的 harvest()方法
    }
}
```

运行结果如图 9.1 所示。

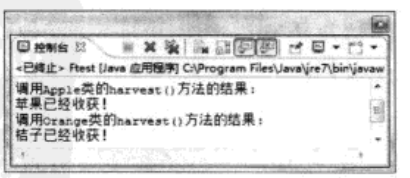


图 9.1　定义抽象类及抽象方法

9.2 内 部 类

视频讲解：光盘\TM\lx\9\内部类.exe

如果在一个类中再定义一个类，就将在类中再定义的那个类称为内部类。内部类可分为成员内部类、局部内部类以及匿名内部类。

9.2.1 成员内部类

1. 成员内部类简介

在一个类中使用内部类可以在内部类中直接存取其所在类的私有成员变量。本节首先介绍成员内部类。成员内部类的语法格式如下：

```
public class OuterClass{                          //外部类
    private class InnerClass{                     //内部类
        //...
    }
}
```

在内部类中可以随意使用外部类的成员方法以及成员变量，尽管这些类成员被修饰为 private。图 9.2 充分说明了内部类的使用，尽管成员变量 i 以及成员方法 g()都在外部类中被修饰为 private，但在内部类中可以直接使用外部类中的类成员。

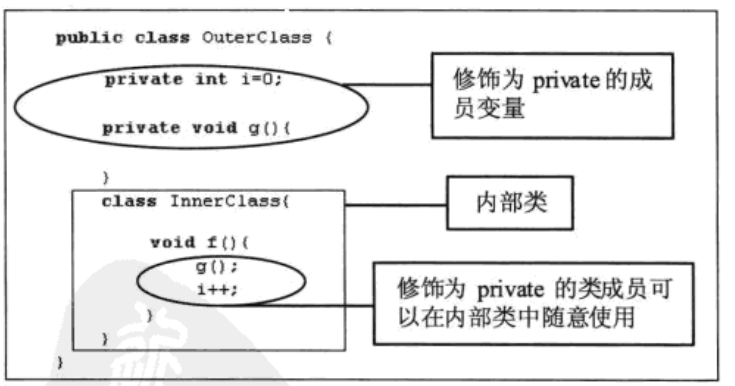


图 9.2 内部类可以使用外部类的成员

内部类的实例一定要绑定在外部类的实例上，如果在外部类中初始化一个内部类对象，那么内部类对象就会绑定在外部类对象上。内部类初始化方式与其他类初始化方式相同，都是使用 new 关键字。

【例 9.3】 创建 OuterClass 类，在类中定义 innerClass 内部类和 doit()方法，在主方法中创建 Outer Class 类的实例对象并调用 doit()方法。（**实例位置：光盘\TM\sl\9\2**）

```
public class OuterClass {
    innerClass in=new innerClass();                    //在外部类实例化内部类对象引用
    public void ouf(){
        in.inf();                                      //在外部类方法中调用内部类方法
    }
    class innerClass{
        innerClass(){                                  //内部类构造方法
        }
        public void inf(){                             //内部类成员方法
        }
        int y=0;                                       //定义内部类成员变量
    }
    public innerClass doit(){                          //外部类方法，返回值为内部类引用
        //y=4;                                         //外部类不可以直接访问内部类成员变量
        in.y=4;
        return new innerClass();                       //返回内部类引用
    }
    public static void main(String args[]){
        OuterClass out=new OuterClass();
        //内部类的对象实例化操作必须在外部类或外部类中的非静态方法中实现
        OuterClass.innerClass in=out.doit();
    }
}
```

例 9.3 中的外部类创建内部类实例时与其他类创建对象引用时相同。内部类可以访问它的外部类的成员，但内部类的成员只有在内部类的范围之内是可知的，不能被外部类使用。例如，在例 9.3 中如果将内部类的成员变量 y 再次赋值时将会出错，但是如果使用内部类对象引用调用成员变量 y 即可。图 9.3 说明了内部类 innerClass 对象与外部类 OuterClass 对象的关系。

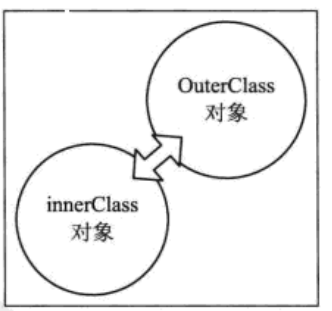


图 9.3　内部类对象与外部类对象的关系

从图 9.3 中可以看出，内部类对象与外部类对象关系非常紧密，内外可以交互使用彼此类中定义的变量。

注意　如果在外部类和非静态方法之外实例化内部类对象，需要使用“外部类.内部类”的形式指定该对象的类型。

在例 9.3 的主方法中如果不使用 doit()方法返回内部类对象引用，可以直接使用内部类实例化内部

类对象，但由于是在主方法中实例化内部类对象，必须在 new 操作符之前提供一个外部类的引用。

【例 9.4】 在主方法中实例化一个内部类对象。

```
public static void main(String args[]){
    OuterClass out=new OuterClass();
    OuterClass.innerClass in2=out.new innerClass();          //实例化内部类对象
}
```

在实例化内部类对象时，不能在 new 操作符之前使用外部类名称那种形式实例化内部类对象，而是应该使用外部类的对象来创建其内部类的对象。

注意 内部类对象会依赖于外部类对象，除非已经存在一个外部类对象，否则类中不会出现内部类对象。

2. 使用 this 关键字获取内部类与外部类的引用

如果在外部类中定义的成员变量与内部类的成员变量名称相同，可以使用 this 关键字。

【例 9.5】 创建 TheSameName 类，在类中定义成员变量 x，定义一个内部类 Inner，并在内部类中也创建 x 变量，在内部类的 doit()方法中分别操作两个 x 变量。（**实例位置：光盘\TM\sl\9\3**）

```
public class TheSameName {
    private int x;
    private class Inner{
        private int x=9;
        public void doit(int x){
            x++;                                    //调用的是形参 x
            this.x++;                               //调用内部类的变量 x
            TheSameName.this.x++;                   //调用外部类的变量 x
        }
    }
}
```

在类中如果内部类与外部类遇到成员变量重名的情况可以使用 this 关键字进行处理，例如在内部类中使用 this.x 语句可以调用内部类的成员变量 x，而使用 TheSameName.this.x 语句可以调用外部类的成员变量 x，即使用外部类名称后跟一个点操作符和 this 关键字便可获取外部类的一个引用。

读者应该明确一点，在内存中所有对象被放置在堆中，将方法以及方法中的形参或局部变量放置在栈中，如图 9.4 所示。在栈中的 doit()方法指向内部类的对象，而内部类的对象与外部类的

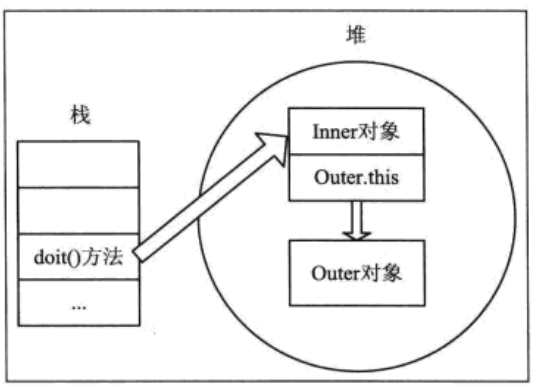


图 9.4 内部类对象与外部类对象在内存中的分布情况

对象是相互依赖的，Outer.this 对象指向外部类对象。

9.2.2　局部内部类

局部内部类是指在类的方法中定义的内部类，它的作用范围也是在这个方法体内。下面将通过一个具体的例子来说明如何定义局部内部类。

【例 9.6】　在外部类的 sell()方法中创建 Apple 局部内部类，然后创建该内部类的实例，并调用其定义的 price()方法输出单价信息。（**实例位置：光盘\TM\sl\9\4**）

```
public class SellOutClass {
    private String name;                                    //私有成员变量
    public SellOutClass() {                                 //构造方法
        name = "苹果";
    }
    public void sell(int price) {
        class Apple {                                       //局部内部类
            int innerPrice = 0;
            public Apple(int price) {                       //构造方法
                innerPrice = price;
            }
            public void price() {                           //方法
                System.out.println("现在开始销售" + name);
                System.out.println("单价为：" + innerPrice + "元");
            }
        }
        Apple apple = new Apple(price);                     //实例化 Apple 类的对象
        apple.price();                                      //调用局部内部类的方法
    }
    public static void main(String[] args) {
        SellOutClass sample = new SellOutClass();           //实例化 SellOutClass 类的对象
        sample.sell(100);                                   //调用 SellOutClass 类的 sell()方法
    }
}
```

运行结果如图 9.5 所示。

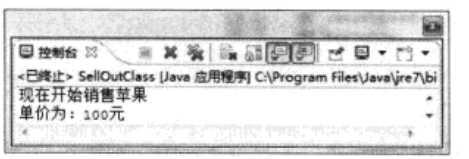


图 9.5　通过调用局部内部类输出单价信息

在上述代码中可以看到，笔者将内部类定义在 sell()方法内部。但是有一点值得注意，内部类 Apple 是 sell()方法的一部分，并非 SellOutClass 类的一部分，所以在 sell()方法的外部不能访问该内部类，但是该内部类可以访问当前代码块的常量以及此外部类的所有成员。

9.2.3　匿名内部类

在编写程序代码时，不一定要给内部类取一个名字，可以直接以对象名来代替。匿名内部类的所

有实现代码都需要在大括号之间进行编写。

说明 在图形化编程的事件监控器代码中，会大量使用匿名内部类，这样可以大大简化代码，并增强代码的可读性。

匿名内部类的语法格式如下：

```
return new A() {
    ...//内部类体
};
```

其中，A 表示对象名。

由于匿名内部类没有名称，所以匿名内部类使用默认构造方法来生成匿名内部类的对象。在匿名内部类定义结束后，需要加分号标识，这个分号并不代表定义内部类结束的标识，而代表创建匿名内部类的引用表达式的标识。

说明 匿名内部类编译以后，会产生以“外部类名$序号”为名称的.class 文件，序号以 1～n 排列，分别代表 1～n 个匿名内部类。

【例 9.7】 在 main()方法中编写匿名内部类去除字符串中的全部空格。（**实例位置：光盘\TM\sl\9\5**）

```
public class OutString {
    public static void main(String[] args) {
        final String sourceStr = "吉林省 明日 科技有限公司——编程 词典！";
        IStringDeal s = new IStringDeal() {                          //编写匿名内部类
            @Override
            public String filterBlankChar() {
                String convertStr = sourceStr;
                convertStr = convertStr.replaceAll(" ", "");         //替换全部空格
                return convertStr;                                   //返回转换后的字符串
            }
        };
        System.out.println("源字符串：" + sourceStr);                //输出源字符串
        System.out.println("转换后的字符串：" + s.filterBlankChar());  //输出转换后的字符串
    }
}
```

编写接口 IStringDeal，在该接口中声明一个方法 filterBlankChar()。代码如下：

```
public interface IStringDeal {
    public String filterBlankChar();          //声明过滤字符串中的空格的方法
}
```

运行结果如图 9.6 所示。

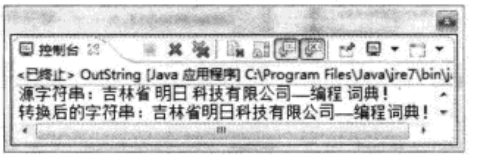


图 9.6　去除字符串中的全部空格

9.2.4　静态内部类

在内部类前添加修饰符 static，这个内部类就变为静态内部类。一个静态内部类中可以声明 static 成员，但是在非静态内部类中不可以声明静态成员。静态内部类有一个最大的特点，就是不可以使用外部类的非静态成员，所以静态内部类在程序开发中比较少见。

可以这样认为，普通的内部类对象隐式地在外部保存了一个引用，指向创建它的外部类对象，但如果内部类被定义为 static 时，它应该具有更多的限制。静态内部类具有以下两个特点：

☑　创建静态内部类的对象，不需要其外部类的对象。

☑　不能从静态内部类的对象中访问非静态外部类的对象。

【例 9.8】　定义一个静态内部类 StaticInnerClass。

```
public class StaticInnerClass {
    int x=100;
    static class Inner{
        void doitInner(){
            System.out.println("外部类"+x);                    //调用外部类的成员变量 x
        }
    }
}
```

在例 9.8 中，在内部类的 doitInner()方法中调用成员变量 x，由于 Inner 被修饰为 static 形式，而成员变量 x 却是非 static 类型的，所以在 doitInner()方法中不能调用 x 变量。

进行程序测试时，如果在每一个 Java 文件中都设置一个主方法，将出现很多额外代码，而程序本身并不需要这些主方法，为了解决这个问题，可以将主方法写入静态内部类中。

【例 9.9】　在静态内部类中定义主方法，并访问内部类中的方法。（**实例位置：光盘\TM\sl\9\6**）

```
public class StaticInnerClass {
    static int x=100;
    static class Inner{
        static void doitInner(){
            System.out.println("外部类的成员变量"+x);          //调用外部类的成员变量 x
        }
        public static void main(String args[]){               //定义主方法
            doitInner();                                      //访问内部类的方法
        }

    }
}
```

运行结果如图 9.7 所示。

如果编译例 9.9 中的类，将编译生成一个名称为 StaticInnerClass$Inner 的独立类和一个 StaticInnerClass 类，只要使用 java StaticInnerClass$Inner 就可以运行主方法中的内容，这样当测试完成需要将所有.class 文件打包时，只要删除 StaticInnerClass$Inner 独立类即可。

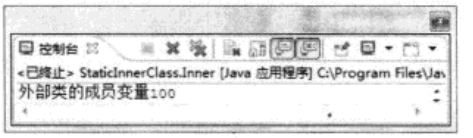


图 9.7　访问静态内部类中的方法

9.2.5　内部类的继承

内部类也和其他普通类一样可以被继承，但是继承内部类比继承普通类要复杂一些，需要设置专门的语法来完成。

【例 9.10】　创建 OutputInnerClass 类，使 OutputInnerClass 类继承 ClassA 类中的内部类 ClassB。

```
public class OutputInnerClass extends ClassA.ClassB {            //继承内部类 ClassB
    public OutputInnerClass(ClassA a) {
        a.super();
    }
}
class ClassA {
    class ClassB {
    }
}
```

在某个类继承内部类时，必须硬性给予这个类一个带参数的构造方法，并且该构造方法的参数为需要继承内部类的外部类的引用，同时在构造方法体中使用 a.super()语句，这样才为继承提供了必要的对象引用。

9.2.6　范例 1：局部内部类设置闹钟

日常生活中，闹钟的应用非常广泛，使用它可以更好地帮助人们安排时间。本范例将实现一个非常简单的闹钟。运行本范例，控制台会不断输出当前的时间，并且每隔一秒钟会发出提示音。用户可以单击“确定”按钮来退出程序。运行结果如图 9.8 所示。**（实例位置：光盘\TM\sl\9\7）**

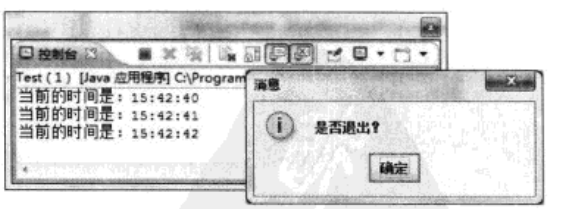


图 9.8　局部内部类设置闹钟

在项目中创建 AlarmClock 类，在该类中首先定义两个属性，一个是 delay，表示延迟的时间；另一个是 flag，表示是否需要发出提示声音。然后在 start()方法中使用 Timer 类来安排动作发出事件。代码如下：

```
public class AlarmClock {
    private int delay;                                           //表示延迟时间
```

```
    private boolean flag;                                          //表示是否要发出声音
    public AlarmClock(int delay, boolean flag) {                   //使用构造方法初始化各个域
        this.delay = delay;
        this.flag = flag;
    }
    public void start() {
        class Printer implements ActionListener {                  //定义内部类实现动作监听接口
            @Override
            public void actionPerformed(ActionEvent e) {
                SimpleDateFormat format = new SimpleDateFormat("k:m:s");//定义时间的格式
                String result = format.format(new Date());         //获得当前的时间
                System.out.println("当前的时间是：" + result);       //显示当前的时间
                if (flag) {                                        //根据 flag 来决定是否要发出声音
                    Toolkit.getDefaultToolkit().beep();
                }
            }
        }
        new Timer(delay, new Printer()).start();                   //创建 Timer 对象并启动
    }
}
```

9.2.7　范例 2：静态内部类求极值

当对元素进行排序时，需要明确各个元素如何比较大小。使用既定的比较方式，就可以求出一个数组中的最大值和最小值。本范例使用静态内部类来实现使用一次遍历求最大和最小值。运行结果如图 9.9 所示。（**实例位置：光盘\TM\sl\9\8**）

（1）在项目中创建 MaxMin 类，在该类中首先定义一个静态内部类 Result。然后在 Result 类中定义两个浮点型属性，一个是 max，表示最大值；另一个是 min，表示最小值。再使用构造方法为其初始化，并提供 getXXX()方法来获得这两个值。最后定义一个静态方法 getResult()，该方法的返回值是 Result 类型，这样就可以既保存最大值，又保存最小值了。代码如下：

图 9.9　求数组中的最大值和最小值

```
public class MaxMin {
    public static class Result {
        private double max;                                        //表示最大值
        private double min;                                        //表示最小值
        public Result(double max, double min) {                    //使用构造方法进行初始化
            this.max = max;
            this.min = min;
        }
        public double getMax() {                                   //获得最大值
            return max;
```

```
        }
        public double getMin() {                                    //获得最小值
            return min;
        }
    }
    public static Result getResult(double[] array) {
        double max = Double.MIN_VALUE;
        double min = Double.MAX_VALUE;
        for (double i : array) {                                    //遍历数组获得最大值和最小值
            if (i > max) {
                max = i;
            }
            if (i < min) {
                min = i;
            }
        }
        return new Result(max, min);                                //返回 Result 对象
    }
}
```

（2）在项目中创建 Test 类进行测试，该类的主方法中，使用随机数初始化了一个长度为 5 的数组，并求得该数组的最大值和最小值。代码如下：

```
public class Test {
    public static void main(String[] args) {
        double[] array = new double[5];
        for (int i = 0; i < array.length; i++) {                    //初始化数组
            array[i] = 100*Math.random();
        }
        System.out.println("源数组：");
        for (int i = 0; i < array.length; i++) {                    //显示数组中的各个元素
            System.out.println(array[i]);
        }
        System.out.println("最大值：" + MaxMin.getResult(array).getMax());    //显示最大值
        System.out.println("最小值：" + MaxMin.getResult(array).getMin());    //显示最小值
    }
}
```

9.3 Class 类与 Java 反射

视频讲解：光盘\TM\lx\9\Class 类与 Java 反射.exe

通过 Java 反射机制，可以在程序中访问已经装载到 JVM 中的 Java 对象的描述，实现访问、检测和修改描述 Java 对象本身信息的功能。Java 反射机制的功能十分强大，在 java.lang.reflect 包中提供了对该功能的支持。

众所周知，所有 Java 类均继承了 Object 类，在 Object 类中定义了一个 getClass()方法，该方法返

回一个类型为 Class 的对象。代码如下：

```
Class textFieldC = textField.getClass();          //textField 为 JTextField 类的对象
```

利用 Class 类的对象 textFieldC，可以访问用来返回该对象的 textField 对象的描述信息。主要的描述信息如表 9.1 所示。

表 9.1　通过反射可访问的主要描述信息

组成部分	访问方法	返回值类型	说明
包路径	getPackage()	Package 对象	获得该类的存放路径
类名称	getName()	String 对象	获得该类的名称
继承类	getSuperclass()	Class 对象	获得该类继承的类
实现接口	getInterfaces()	Class 型数组	获得该类实现的所有接口
构造方法	getConstructors()	Constructor 型数组	获得所有权限为 public 的构造方法
	getConstructor(Class<?>... parameterTypes)	Constructor 对象	获得权限为 public 的指定构造方法
	getDeclaredConstructors()	Constructor 型数组	获得所有构造方法，按声明顺序返回
	getDeclaredConstructor(Class<?>... parameterTypes)	Constructor 对象	获得指定构造方法
方法	getMethods()	Method 型数组	获得所有权限为 public 的方法
	getMethod(String name, Class<?>... parameter Types)	Method 对象	获得权限为 public 的指定方法
	getDeclaredMethods()	Method 型数组	获得所有方法，按声明顺序返回
	getDeclaredMethod(String name, Class<?>... parameterTypes)	Method 对象	获得指定方法
成员变量	getFields()	Field 型数组	获得所有权限为 public 的成员变量
	getField(String name)	Field 对象	获得权限为 public 的指定成员变量
	getDeclaredFields()	Field 型数组	获得所有成员变量，按声明顺序返回
	getDeclaredField(String name)	Field 对象	获得指定成员变量
内部类	getClasses()	Class 型数组	获得所有权限为 public 的内部类
	getDeclaredClasses()	Class 型数组	获得所有内部类
内部类的声明类	getDeclaringClass()	Class 对象	如果该类为内部类则返回它的成员类，否则返回 null

说明　在通过 getFields()和 getMethods()方法依次获得权限为 public 的成员变量和方法时，还包含从超类中继承到的成员变量和方法；而通过 getDeclaredFields()和 getDeclaredMethods()方法只是获得在本类中定义的所有成员变量和方法。

9.3.1　访问构造方法

在通过下列一组方法访问构造方法时，将返回 Constructor 类型的对象或数组。每个 Constructor 对象代表一个构造方法，利用 Constructor 对象可以操纵相应的构造方法。

- ☑ getConstructors()
- ☑ getConstructor(Class<?>... parameterTypes)
- ☑ getDeclaredConstructors()
- ☑ getDeclaredConstructor(Class<?>... parameterTypes)

如果是访问指定的构造方法，需要根据该构造方法的入口参数的类型来访问。例如访问一个入口参数类型依次为 String 和 int 型的构造方法，可以通过下面两种方式实现。

```
objectClass.getDeclaredConstructor(String.class, int.class);
objectClass.getDeclaredConstructor(new Class[] { String.class, int.class });
```

Constructor 类中提供的常用方法如表 9.2 所示。

表 9.2　Constructor 类中的常用方法

方　法	说　明
isVarArgs()	查看该构造方法是否允许带有可变数量的参数，如果允许则返回 true，否则返回 false
getParameterTypes()	按照声明顺序以 Class 数组的形式获得该构造方法的各个参数的类型
getExceptionTypes()	以 Class 数组的形式获得该构造方法可能抛出的异常类型
newInstance(Object... initargs)	通过该构造方法利用指定参数创建一个该类的对象，如果未设置参数则表示采用默认无参数的构造方法
setAccessible(boolean flag)	如果该构造方法的权限为 private，默认为不允许通过反射利用 newInstance(Object... initargs)方法创建对象。如果先执行该方法，并将入口参数设为 true，则允许创建
getModifiers()	获得可以解析出该构造方法所采用修饰符的整数

通过 java.lang.reflect.Modifier 类可以解析出 getModifiers()方法的返回值所表示的修饰符信息，在该类中提供了一系列用来解析的静态方法，既可以查看是否被指定的修饰符修饰，还可以以字符串的形式获得所有修饰符。该类常用的静态方法如表 9.3 所示。

表 9.3　Modifier 类中的常用静态方法

静 态 方 法	说　明
isPublic(int mod)	查看是否被 public 修饰符修饰，如果是则返回 true，否则返回 false
isProtected(int mod)	查看是否被 protected 修饰符修饰，如果是则返回 true，否则返回 false
isPrivate(int mod)	查看是否被 private 修饰符修饰，如果是则返回 true，否则返回 false
isStatic(int mod)	查看是否被 static 修饰符修饰，如果是则返回 true，否则返回 false
isFinal(int mod)	查看是否被 final 修饰符修饰，如果是则返回 true，否则返回 false
toString(int mod)	以字符串的形式返回所有修饰符

例如，判断 constructor 对象所代表的构造方法是否被 private 修饰，以及以字符串形式获得该构造方法的所有修饰符。代码如下：

```
int modifiers = constructor.getModifiers();
boolean isEmbellishByPrivate = Modifier.isPrivate(modifiers);
String embellishment = Modifier.toString(modifiers);
```

【例 9.11】　访问构造方法。（**实例位置：光盘\TM\sl\9\9**）

（1）在项目中创建一个 MoreConstructor 类，在该类中声明一个 String 型成员变量和 3 个 int 型成

员变量，并提供 3 个构造方法。代码如下：

```
public class MoreConstructor {
    String s;
    int i, i2, i3;
    private MoreConstructor() {                                    //不带参数的构造方法
    }
    protected MoreConstructor(String s, int i) {                   //带两个参数的构造方法
        this.s = s;
        this.i = i;
    }
    //带可变参数的构造方法
    public MoreConstructor(String... strings) throws NumberFormatException {
        if (0 < strings.length)
            i = Integer.valueOf(strings[0]);
        if (1 < strings.length)
            i2 = Integer.valueOf(strings[1]);
        if (2 < strings.length)
            i3 = Integer.valueOf(strings[2]);
    }
    public void print() {
        System.out.println("s=" + s);
        System.out.println("i=" + i);
        System.out.println("i2=" + i2);
        System.out.println("i3=" + i3);
    }
}
```

（2）编写测试类 AccessConstructor，在该类中通过反射访问 MoreConstructor 类中的所有构造方法，并将该构造方法是否允许带有可变数量的参数、入口参数类型和可能抛出的异常类型信息输出到控制台。代码如下：

```
Constructor[] declaredConstructors = exampleC.getDeclaredConstructors();    //获得所有构造方法
for (int i = 0; i < declaredConstructors.length; i++) {
    Constructor constructor = declaredConstructors[i];                      //遍历构造方法
    System.out.println("查看是否允许带有可变数量的参数：" + constructor.isVarArgs());
    System.out.println("该构造方法的入口参数类型依次为：");
    Class[] parameterTypes = constructor.getParameterTypes();               //获得所有参数类型
    for (int j = 0; j < parameterTypes.length; j++) {
        System.out.println(" " + parameterTypes[j]);
    }
    System.out.println("该构造方法可能抛出的异常类型为：");
    //获得所有可能抛出的异常信息类型
    Class[] exceptionTypes = constructor.getExceptionTypes();
    for (int j = 0; j < exceptionTypes.length; j++) {
        System.out.println(" " + exceptionTypes[j]);
    }
    MoreConstructor example2 = null;
    while (example2 == null) {
        //如果该成员变量的访问权限为 private，则抛出异常，即不允许访问
```

```
        try {
            if (i == 0)                                    //通过执行默认没有参数的构造方法创建对象
                example2 = (MoreConstructor) constructor.newInstance();
            else if (i == 1)                               //通过执行具有两个参数的构造方法创建对象
                example2 = (MoreConstructor) constructor.newInstance("7", 5);
            else {                                         //通过执行具有可变数量参数的构造方法创建对象
                Object[] parameters = new Object[] { new String[] { "100", "200", "300" } };
                example2 = (MoreConstructor) constructor.newInstance(parameters);
            }
        } catch (Exception e) {
            System.out.println("在创建对象时抛出异常，下面执行 setAccessible()方法");
            constructor.setAccessible(true);               //设置为允许访问
        }
    }
    example2.print();
    System.out.println();
}
```

运行本例，当通过反射访问构造方法 MoreConstructor()时，输出信息如图 9.10 所示。

当通过反射访问构造方法 MoreConstructor(String s, int i)时，输出信息如图 9.11 所示。

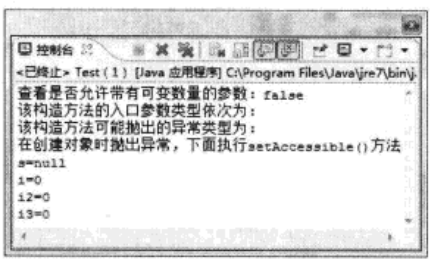


图 9.10　访问 MoreConstructor()输出的信息

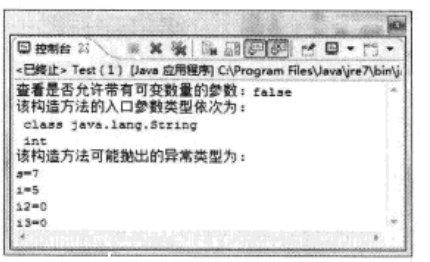


图 9.11　访问 MoreConstructor(String s, int i)输出的信息

当通过反射访问构造方法 MoreConstructor(String... strings)时，输出信息如图 9.12 所示。

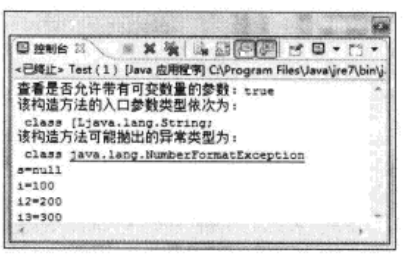


图 9.12　访问 MoreConstructor(String... strings)输出的信息

9.3.2　访问成员变量

在通过下列一组方法访问成员变量时，将返回 Field 类型的对象或数组。每个 Field 对象代表一个成员变量，利用 Field 对象可以操纵相应的成员变量。

☑　getFields()

- ☑ getField(String name)
- ☑ getDeclaredFields()
- ☑ getDeclaredField(String name)

如果是访问指定的成员变量，可以通过该成员变量的名称来访问。例如访问一个名称为 birthday 的成员变量，代码如下：

```
object. getDeclaredField("birthday");
```

Field 类中提供的常用方法如表 9.4 所示。

表 9.4　Field 类中的常用方法

方　法	说　明
getName()	获得该成员变量的名称
getType()	获得表示该成员变量类型的 Class 对象
get(Object obj)	获得指定对象 obj 中该成员变量的值，返回值为 Object 型
set(Object obj, Object value)	将指定对象 obj 中该成员变量的值设置为 value
getInt(Object obj)	获得指定对象 obj 中类型为 int 的该成员变量的值
setInt(Object obj, int i)	将指定对象 obj 中类型为 int 的该成员变量的值设置为 i
getFloat(Object obj)	获得指定对象 obj 中类型为 float 的该成员变量的值
setFloat(Object obj, float f)	将指定对象 obj 中类型为 float 的该成员变量的值设置为 f
getBoolean(Object obj)	获得指定对象 obj 中类型为 boolean 的该成员变量的值
setBoolean(Object obj, boolean z)	将指定对象 obj 中类型为 boolean 的该成员变量的值设置为 z
setAccessible(boolean flag)	如果该构造方法的权限为 private，默认为不允许通过反射利用 newInstance(Object... initargs)方法创建对象。如果先执行该方法，并将入口参数设为 true，则允许创建
getModifiers()	获得可以解析出该成员变量所采用修饰符的整数

【例 9.12】　访问成员变量。（**实例位置：光盘\TM\sl\9\10**）

（1）在项目中创建一个 MoreFields 类，在该类中依次声明一个 int、float、boolean 和 String 型的成员变量，并将它们设置为不同的访问权限。代码如下：

```
public class MoreFields {
    int i;
    public float f;
    protected boolean b;
    private String s;
}
```

（2）通过反射访问 MoreFields 类中的所有成员变量，将成员变量的名称和类型信息输出到控制台，并分别将各个成员变量在修改前后的值输出到控制台。代码如下：

```
Field[] declaredFields = exampleC.getDeclaredFields();                    //获得所有成员变量
for (int i = 0; i < declaredFields.length; i++) {
    Field field = declaredFields[i];                                      //遍历成员变量
    System.out.println("名称为：" + field.getName());                     //获得成员变量名称
    Class fieldType = field.getType();                                    //获得成员变量类型
    System.out.println("类型为：" + fieldType);
```

```
    boolean isTurn = true;
    while (isTurn) {
        try { //如果该成员变量的访问权限为 private，则抛出异常，即不允许访问
            isTurn = false;
            System.out.println("修改前的值为：" + field.get(example));          //获得成员变量值
            //判断成员变量的类型是否为 int 型
            if (fieldType.equals(int.class)) {
                System.out.println("利用方法 setInt()修改成员变量的值");
                field.setInt(example, 168);                                     //为 int 型成员变量赋值
             //判断成员变量的类型是否为 float 型
            } else if (fieldType.equals(float.class)) {
                System.out.println("利用方法 setFloat()修改成员变量的值");
                field.setFloat(example, 99.9F);                                 //为 float 型成员变量赋值
             //判断成员变量的类型是否为 boolean 型
            } else if (fieldType.equals(boolean.class)) {
                System.out.println("利用方法 setBoolean()修改成员变量的值");
                field.setBoolean(example, true);                                //为 boolean 型成员变量赋值
            } else {
                System.out.println("利用方法 set()修改成员变量的值");
                field.set(example, "mingri");                                   //可以各种类型变量赋值
            }
            System.out.println("修改后的值为：" + field.get(example));          //获得成员变量值
        } catch (Exception e) {
            System.out.println("在设置成员变量值时抛出异常，下面执行 setAccessible()方法！");
            field.setAccessible(true);                                          //设置为允许访问
            isTurn = true;
        }
    }
    System.out.println();
}
```

运行结果如图 9.13 所示。

图 9.13　通过反射访问成员变量

从该信息中可以看出，要访问权限为 private 的成员变量 s，需要执行 setAccessible()方法，并将入口参数设为 true，否则不允许访问。

9.3.3　访问方法

在通过下列一组方法访问方法时，将返回 Method 类型的对象或数组。每个 Method 对象代表一个方法，利用 Method 对象可以操纵相应的方法。

- ☑ getMethods()
- ☑ getMethod(String name, Class<?>... parameterTypes)
- ☑ getDeclaredMethods()
- ☑ getDeclaredMethod(String name, Class<?>... parameterTypes)

如果是访问指定的方法，需要根据该方法的名称和入口参数的类型来访问。例如访问一个名称为

print、入口参数类型依次为 String 和 int 型的方法，可以通过下面两种方式实现。

```
objectClass.getDeclaredMethod("print", String.class, int.class);
objectClass.getDeclaredMethod("print", new Class[] {String.class, int.class });
```

Method 类中提供的常用方法如表 9.5 所示。

表 9.5　Method 类中的常用方法

方　　法	说　　明
getName()	获得该方法的名称
getParameterTypes()	按照声明顺序以 Class 数组的形式获得该方法的各个参数的类型
getReturnType()	以 Class 对象的形式获得该方法的返回值的类型
getExceptionTypes()	以 Class 数组的形式获得该方法可能抛出的异常类型
invoke(Object obj, Object... args)	利用指定参数 args 执行指定对象 obj 中的该方法，返回值为 Object 型
isVarArgs()	查看该构造方法是否允许带有可变数量的参数，如果允许则返回 true，否则返回 false
getModifiers()	获得可以解析出该方法所采用修饰符的整数

【例 9.13】　访问方法。（**实例位置：光盘\TM\sl\9\11**）

（1）在项目中创建一个 MoreMethod 类，并编写 4 个典型的方法。代码如下：

```
public class MoreMethod {
    static void staticMethod() {
        System.out.println("执行 staticMethod()方法");
    }
    public int publicMethod(int i) {
        System.out.println("执行 publicMethod()方法");
        return i * 100;
    }
    protected int protectedMethod(String s, int i) throws NumberFormatException {
        System.out.println("执行 protectedMethod()方法");
        return Integer.valueOf(s) + i;
    }
    private String privateMethod(String... strings) {
        System.out.println("执行 privateMethod()方法");
        StringBuffer stringBuffer = new StringBuffer();
        for (int i = 0; i < strings.length; i++) {
            stringBuffer.append(strings[i]);
        }
        return stringBuffer.toString();
    }
}
```

（2）通过反射访问 MoreMethod 类中的所有方法，将各个方法的名称、入口参数类型、返回值类型等信息输出到控制台，并执行部分方法。代码如下：

```
Method[] declaredMethods = exampleC.getDeclaredMethods();                    //获得所有方法
for (int i = 0; i < declaredMethods.length; i++) {
    Method method = declaredMethods[i];                                      //遍历方法
    System.out.println("名称为：" + method.getName());                        //获得方法名称
```

```
    System.out.println("是否允许带有可变数量的参数：" + method.isVarArgs());
    System.out.println("入口参数类型依次为：");
    Class[] parameterTypes = method.getParameterTypes();                    //获得所有参数类型
    for (int j = 0; j < parameterTypes.length; j++) {
        System.out.println(" " + parameterTypes[j]);
    }
    System.out.println("返回值类型为：" + method.getReturnType());          //获得方法返回值类型
    System.out.println("可能抛出的异常类型有：");
    //获得方法可能抛出的所有异常类型
    Class[] exceptionTypes = method.getExceptionTypes();
    for (int j = 0; j < exceptionTypes.length; j++) {
        System.out.println(" " + exceptionTypes[j]);
    }
    boolean isTurn = true;
    while (isTurn) {
        //如果该方法的访问权限为 private，则抛出异常，即不允许访问
        try {
            isTurn = false;
            if (i == 0)
                method.invoke(example);                                     //执行没有入口参数的方法
            else if (i == 1)
                System.out.println("返回值为：" + method.invoke(example, 168));      //执行方法
            else if (i == 2)
                System.out.println("返回值为：" + method.invoke(example, "7", 5));   //执行方法
            else {
                Object[] parameters = new Object[] { new String[] { "M", "W", "Q" } };   //定义二维数组
                System.out.println("返回值为：" + method.invoke(example, parameters)); //执行方法
            }
        } catch (Exception e) {
            System.out.println("在执行方法时抛出异常，下面执行 setAccessible()方法！");
            method.setAccessible(true);                                     //设置为允许访问
            isTurn = true;
        }
    }
    System.out.println();
}
```

在反射中执行具有可变数量的参数的构造方法时，需要将入口参数定义成二维数组。运行本例，访问 staticMethod()方法，运行结果如图 9.14 所示。

访问 publicMethod()方法，运行结果如图 9.15 所示。

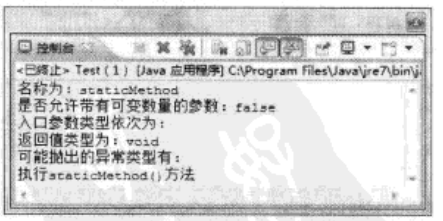


图 9.14　访问 staticMethod()方法输出的信息

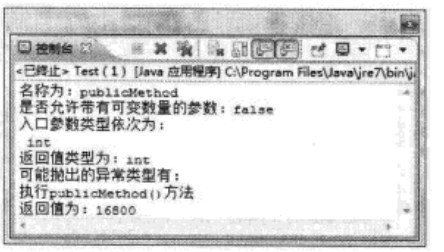


图 9.15　访问 publicMethod()方法输出的信息

访问 protectedMethod()方法，运行结果如图 9.16 所示。

访问 privateMethod()方法，运行结果如图 9.17 所示。

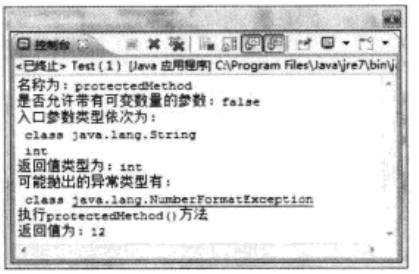


图 9.16　访问 protectedMethod()方法输出的信息

图 9.17　访问 privateMethod()方法输出的信息

9.3.4　范例 3：运用反射查看类的成员

通常类的声明包括常见修饰符（public、protected、private、abstract、static、final 和 strictfp 等）、类的名称、类的泛型参数、类的继承类（实现的接口）和类的注解等。本范例将演示如何用反射获得这些信息。运行结果如图 9.18 所示。（**实例位置：光盘\TM\sl\9\12**）

图 9.18　查看类的成员

在项目中创建 ClassDeclarationViewer 类，在类的主方法中输出与类声明相关的各个项。代码如下：

```
public class ClassDeclarationViewer {
    public static void main(String[] args) throws ClassNotFoundException {
        Class<?> clazz = Class.forName("java.util.ArrayList");                    //获得 ArrayList 类对象
        System.out.println("类的标准名称：" + clazz.getCanonicalName());
        System.out.println("类的修饰符：" + Modifier.toString(clazz.getModifiers()));
        //输出类的泛型参数
        TypeVariable<?>[] typeVariables = clazz.getTypeParameters();
        System.out.print("类的泛型参数：");
        if (typeVariables.length != 0) {
            for (TypeVariable<?> typeVariable : typeVariables) {
                System.out.println(typeVariable + "\t");
            }
        } else {
```

```
            System.out.println("空");
        }
        //输出类所实现的所有接口
        Type[] interfaces = clazz.getGenericInterfaces();
        System.out.println("类所实现的接口：");
        if (interfaces.length != 0) {
            for (Type type : interfaces) {
                System.out.println("\t" + type);
            }
        } else {
            System.out.println("\t" + "空");
        }
        //输出类的直接继承类，如果是继承自 Object 则返回空
        Type superClass = clazz.getGenericSuperclass();
        System.out.print("类的直接继承类：");
        if (superClass != null) {
            System.out.println(superClass);
        } else {
            System.out.println("空");
        }
        //输出类的所有注释信息，有些注释信息是不能用反射获得的
        Annotation[] annotations = clazz.getAnnotations();
        System.out.print("类的注解：");
        if (annotations.length != 0) {
            for (Annotation annotation : annotations) {
                System.out.println("\t" + annotation);
            }
        } else {
            System.out.println("空");
        }
    }
}
```

9.3.5 范例 4：动态调用类中方法

在 Java 语言中，调用类的方法有两种方式：对于静态方法可以直接使用类名调用，对于非静态方法必须使用类的对象调用。反射机制提供了比较另类的调用方式，可以根据需要指定要调用的方法，而不必在编程时确定。调用的方法不仅只限于 public 的，还可以是 private 的。本范例将使用反射机制调用 Math 类的静态方法 sin()和 String 类的非静态方法 equals()。运行结果如图 9.19 所示。（**实例位置：光盘\TM\sl\9\13**）

图 9.19 动态调用类中的方法

在项目中创建 MethodTest 类，在该类的主方法中分别调用

了 Math 类的静态方法 sin()和 String 类的非静态方法 equals()。代码如下：

```
public class MethodTest {
    public static void main(String[] args) {
        try {
            System.out.println("调用 Math 类的静态方法 sin()");
            Method sin = Math.class.getDeclaredMethod("sin", Double.TYPE);
            Double sin1 = (Double) sin.invoke(null, new Integer(1));
            System.out.println("1 的正弦值是：" + sin1);
            System.out.println("调用 String 类的非静态方法 equals()");
            Method equals = String.class.getDeclaredMethod("equals", Object.class);
            Boolean mrsoft = (Boolean) equals.invoke(new String("明日科技"), "明日科技");
            System.out.println("字符串是否是明日科技：" + mrsoft);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

9.4　经 典 范 例

9.4.1　经典范例 1：利用反射重写 toString()方法

视频讲解：光盘\TM\lx\9\利用反射重写 toString()方法.exe

为了输出对象方便，Object 类提供了 toString()方法。但是该方法的默认值是由类名和哈希码组成的，实用性并不强，通常需要重写该方法以提供更多的信息。本范例主要实现重写类的 toString()方法，用于在调用 toString()方法时，使用反射输出类的包、类的名字、类的公共构造方法、类的公共域和类的公共方法。另外，在重写该方法时，为其传递一个 Object 型的参数，这样可以避免多次重写 toString()方法。运行结果如图 9.20 所示。（**实例位置：光盘\TM\sl\9\14**）

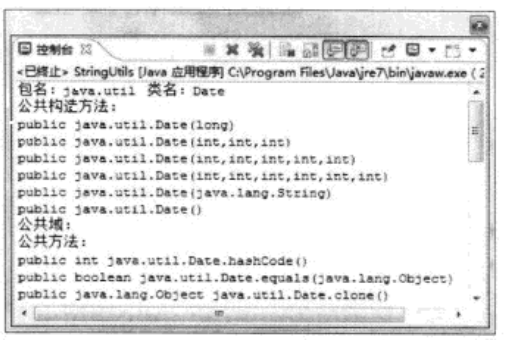


图 9.20　利用反射重写 toString()方法

在项目中创建 StringUtils 类，在该类中定义两个方法，一个是 toString()方法，用于输出类的公共方法和域等信息；另一个是 main()方法，用来进行测试。代码如下：

```
public class StringUtils {
    @SuppressWarnings("unchecked")
    public String toString(Object object) {
        Class clazz = object.getClass();                          //获得代表该类的 Class 对象
        StringBuilder sb = new StringBuilder();                   //利用 StringBuilder 来保存字符串
        Package packageName = clazz.getPackage();                 //获得类所在的包
        sb.append("包名：" + packageName.getName() + "\t");        //输出类所在的包
```

```
        String className = clazz.getSimpleName();                    //获得类的简单名称
        sb.append("类名：" + className + "\n");                      //输出类的简单名称
        sb.append("公共构造方法：\n");
        //获得所有代表构造方法的 Constructor 数组
        Constructor[] constructors = clazz.getDeclaredConstructors();
        for (Constructor constructor : constructors) {
            String modifier = Modifier.toString(constructor.getModifiers()); //获得修饰符
            if (modifier.contains("public")) {                       //查看修饰符是否含有 public
                sb.append(constructor.toGenericString() + "\n");
            }
        }
        sb.append("公共域：\n");
        Field[] fields = clazz.getDeclaredFields();                  //获得代表所有域的 Field[]数组
        for (Field field : fields) {
            String modifier = Modifier.toString(field.getModifiers());
            if (modifier.contains("public")) {                       //查看修饰符是否含有 public
                sb.append(field.toGenericString() + "\n");
            }
        }
        sb.append("公共方法：\n");
        Method[] methods = clazz.getDeclaredMethods();               //获得代表所有方法的 Method[]数组
        for (Method method : methods) {
            String modifier = Modifier.toString(method.getModifiers());
            if (modifier.contains("public")) {                       //查看修饰符是否含有 public
                sb.append(method.toGenericString() + "\n");
            }
        }
        return sb.toString();
    }
    public static void main(String[] args) {
        System.out.println(new StringUtils().toString(new java.util.Date()));
    }
}
```

9.4.2 经典范例 2：普通内部类的简单应用

视频讲解：光盘\TM\lx\9\普通内部类的简单应用.exe

在使用图形界面程序时，用户总是希望界面是丰富多彩的，这就要求程序员根据不同的情况为界面设置不同的颜色。本范例定义了 3 个按钮，用户通过单击不同的按钮，可以为面板设置不同的颜色。运行结果如图 9.21 所示。（**实例位置：光盘\TM\sl\9\15**）

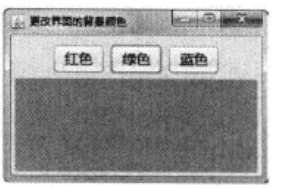


图 9.21 运行结果

（1）编写 ButtonTest 类，该类继承了 JFrame。在框架中包含了 3 个按钮，分别用来为面板设置不同的颜色。

（2）编写 ColorAction 类，该类继承自 ActionListener 接口。在该类的构造方法中，需要为其指定一种颜色，在 actionPerformed()方法中将面板设置成指定的颜色。代码如下：

```
private class ColorAction implements ActionListener {
    private Color background;
    public ColorAction(Color background) {
        this.background = background;
    }
    @Override
    public void actionPerformed(ActionEvent e) {
        panel.setBackground(background);

    }
}
```

9.5 本章小结

本章首先讲解了抽象类，之后又分别讲解了内部类中的成员内部类、局部内部类、匿名内部类和静态内部类，最后讲解了反射。通过学习本章，读者可以对内部类有一个更深层次的了解，并且能够在编写程序时熟练应用内部类。

9.6 实战练习

1．编写 Java 程序，创建一个类，并在类中创建一个成员内部类，通过成员内部类计算 1 到任意数的和的操作，并在外部类中进行测试。（**答案位置：光盘\TM\sl\9\16**）

2．编写 Java 程序，创建一个类，在该类中定义一个方法，并在方法内部创建一个局部内部类，通过局部内部类求任意两个数的乘积，并在外部类中进行测试。（**答案位置：光盘\TM\sl\9\17**）

3．编写 Java 程序，创建一个接口，在该接口中定义一个方法，然后创建一个类，在类中定义一个形参为接口类型的方法，并调用接口中的方法，在类的方法中创建一个实现接口的匿名内部类，用于实现接口中的方法。（**答案位置：光盘\TM\sl\9\18**）

第10章

Java 集合类

（视频讲解：45 分钟）

集合可以看作是一个容器，如红色的衣服可以看作是一个集合、所有 Java 类的书可以看作是一个集合。对于集合中的各个对象很容易将其存放到集合中，也很容易将其从集合中取出来，还可以将其按照一定的顺序进行摆放。Java 中提供了不同的集合类，这些类具有不同的存储对象的方式，并提供了相应的方法方便用户对集合进行遍历、添加、删除以及查找指定的对象。学习 Java 语言一定要学会使用集合。本章向读者介绍 Java 中的各种集合类。

通过阅读本章，您可以：

- 掌握集合类接口的常用方法
- 掌握集合类接口的实现类
- 掌握迭代器的创建和使用

10.1 集合类概述

视频讲解：光盘\TM\lx\10\集合类概述.exe

Java 语言的 java.util 包中提供了一些集合类，这些集合类又被称为容器。提到容器不难会想到数组，集合类与数组的不同之处是，数组的长度是固定的，集合的长度是可变的；数组用来存放基本类型的数据，集合用来存放对象的引用。常用的集合有 List 集合、Set 集合、Map 集合，其中 List 与 Set 实现了 Collection 接口。各接口还提供了不同的实现类。上述集合类的继承关系如图 10.1 所示。

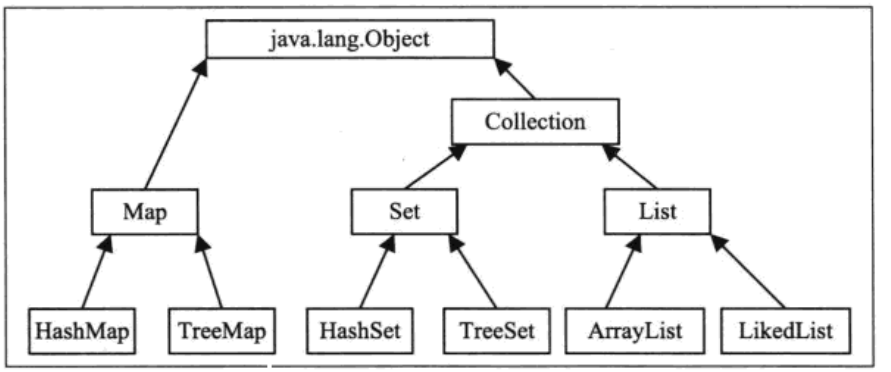


图 10.1 常用集合类的继承关系

10.2 集合类接口的常用方法

视频讲解：光盘\TM\lx\10\集合类接口的常用方法.exe

Collection 接口是层次结构中的根接口。构成 Collection 的单位，被称之为元素。Collection 接口通常不能直接使用，但该接口提供了添加和删除元素、管理数据的方法。由于 List 接口与 Set 接口都实现了 Collection 接口，因此这些方法对 List 集合与 Set 集合是通用的。

10.2.1 List 接口的常用方法

List 接口继承了 Collection 接口，因此包含 Collection 中的所有方法。又因为 List 是列表类型，所以 List 接口还提供了一些适合于自身的常用方法，如表 10.1 所示。

表 10.1 List 接口的常用方法

方　法	返 回 值	功 能 描 述
add(int index,Object obj)	void	用来向集合中的指定索引位置添加对象，集合的索引位置从 0 开始，其他对象的索引位置相对向后移一位

续表

方　法	返　回　值	功 能 描 述
allAll(int index,Collection coll)	boolean	向集合的指定索引位置添加指定的集合对象
remove(int index)	Object	用来移除集合中指定索引位置的对象
get(int index)	Object	用于获取指定索引位置的对象
indexOf(Object obj)	int	该方法返回列表中对象第一次出现的索引位置，如果集合中不包含该元素，则返回-1
lastIndexOf(Object obj)	int	该方法返回列表中对象最后一次出现的索引位置，如果集合中不包含该元素，则返回-1
subList(int formIndex,int toIndex)	List	获取从索引 formIndex 到 toIndex 之间的元素对象
set(int index,E element)	Object	用指定元素替换列表中指定位置的元素，返回以前在指定位置的元素
listIterator()	ListIteratot	用来获得一个包含所有对象的 ListIterator 列表迭代器

从表 10.1 中可以看出，List 接口中适合于自身的方法都与索引有关。由于 List 集合以线性方式存储对象，因此可以通过对象的索引来操作对象。

在 List 集合的众多方法中，add(int index,Object obj)方法与 set(int index,Object obj)方法不容易区分。通过下面的实例，可以看出这两个方法之间的差别。

【例 10.1】 创建集合对象，并向集合中添加元素，通过 set()方法修改集合中元素，再通过 add()方法向集合中添加元素，通过迭代器遍历集合元素。（**实例位置：光盘\TM\sl\10\1**）

```
public class CollectionDemo {
    public static void main(String[] args) {
        String a = "A", b = "B", c = "C", d = "D", e = "E";          //定义要插入集合的字符串对象
        List<String> list = new LinkedList<String>();                //创建 List 集合
        list.add(a);                                                 //向集合中添加元素
        list.add(e);
        list.add(d);
        Iterator<String> fristIterator = list.iterator();            //创建集合的迭代器
        System.out.println("修改前集合中的元素是：");                  //输出信息
        while (fristIterator.hasNext()) {                            //遍历集合中元素
            System.out.print(fristIterator.next() + " ");
        }
        list.set(1, b);                                              //将索引位置为 1 的对象修改为对象 b
        list.add(2, c);                                              //将对象 c 添加到索引位置为 2 的位置
        Iterator<String> it = list.iterator();                       //创建将集合对象修改之后的迭代器对象
        System.out.println();
        System.out.println("修改后集合中的元素是：");
        while (it.hasNext()) {                                       //循环获取集合中元素
            System.out.print(it.next() + " ");
        }
    }
}
```

运行结果如图 10.2 所示。

List 集合中可以包含重复的对象，若要获取重复对象第一次出现的索引位置可以使用 indexOf()方

法，想要获取重复对象最后一次出现的索引位置，可以使用 lastIndexOf()方法。使用 indexOf()与 lastIndexOf()方法时，如果指定的对象在 List 集合中只有一个，则通过这两个方法获得的索引位置是相同的。

【例 10.2】 创建 List 集合对象，通过 add()方法向集合中添加元素，并将对象 apple、b 在集合中第一次与最后一次出现的索引位置输出。（**实例位置：光盘\TM\sl\10\2**）

```
public class CollectionDemo {
    public static void main(String[] args) {
        String a = "a", b = "b", c = "c", d = "d", apple = "apple";      //要添加到集合中的对象
        List<String> list = new ArrayList<String>();                     //创建 List 集合对象
        list.add(a);                                                     //对象 a 的索引位置为 0
        list.add(apple);                                                 //对象 apple 的索引位置为 1
        list.add(b);                                                     //对象 b 的索引位置为 2
        list.add(apple);                                                 //对象 apple 的索引位置为 3
        list.add(c);                                                     //对象 c 的索引位置为 4
        list.add(apple);                                                 //对象 apple 的索引位置为 5
        list.add(d);                                                     //对象 d 的索引位置为 6
        System.out.println(list);                                        //输出列表中的全部元素
        System.out.println("apple 第一次出现的索引位置是："+list.indexOf(apple));
        System.out.println("apple 最后一次出现的索引位置是："+list.lastIndexOf(apple));
        System.out.println("b 第一次出现的索引位置是："+list.indexOf(b));
        System.out.println("b 最后一次出现的索引位置是："+list.lastIndexOf(b));
    }
}
```

运行结果如图 10.3 所示。

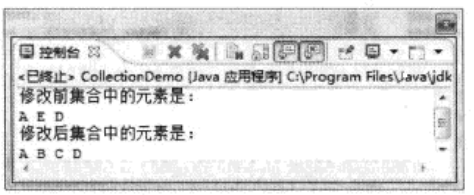


图 10.2 修改集合中的元素

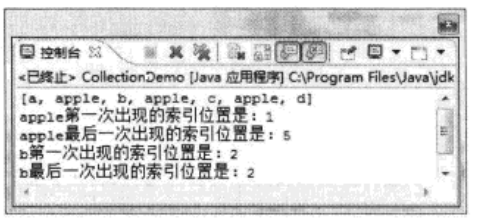


图 10.3 实例运行结果

10.2.2 Set 接口的常用方法

Set 集合由 Set 接口和 Set 接口的实现类组成。Set 接口继承了 Collection 接口，因此包含 Collection 接口的所有方法。Set 接口中的常用方法如表 10.2 所示。

表 10.2 Set 接口中的常用方法

方 法	返 回 值	功 能 描 述
add(Object obj)	boolean	如果此 Set 集合中尚未存在指定的元素，则添加此元素
addAll(Collection coll)	boolean	将参数集合中所有元素添加到此 Set 集合的尾部
remove(Object obj)	boolean	将指定的参数对象移除集合

续表

方　法	返　回　值	功 能 描 述
retainAll(Collection c)	boolean	只保存 Set 集合中包含在指定 Collection 集合中的内容
removeAll(Collection c)	boolean	在 Set 集合中移除包含在指定 Collection 中的元素
clear()	void	移除此 Set 中的所有元素
iterator()	Iterator	返回此 Set 中的元素上进行迭代的迭代器
size()	int	返回此 Set 集合中的所有元素数
isEmpty()	boolean	如果 Set 不包含元素，则返回 true

由于 Set 集合中不允许存在重复值，因此可以使用 Set 集合中的 addAll()方法，将 Collection 集合添加到 Set 集合中并除掉重复值。

【例 10.3】 创建一个 List 集合对象，并往 List 集合中添加元素。再创建一个 Set 集合，利用 addAll()方法把 List 集合对象存入到 Set 集合中并除掉重复值，最后打印 Set 集合中的元素。（**实例位置：光盘\TM\sl\10\3**）

```
public class CollectionDemo {
    public static void main(String[] args) {
        List<String> list = new ArrayList<String>();          //创建 List 集合对象
        list.add("apple");                                    //向集合中添加元素
        list.add("pear");
        list.add("banana");
        list.add("apple");
        Set<String> set = new HashSet<String>();              //创建 List 集合对象
        set.addAll(list);                                     //将 List 集合添加到 Set 集合中
        Iterator<String> it = set.iterator();                 //创建 Set 集合迭代器
        System.out.println("集合中的元素是：");
        while (it.hasNext()) {
          System.out.print(it.next()+"\t");
        }
    }
}
```

运行结果如图 10.4 所示。

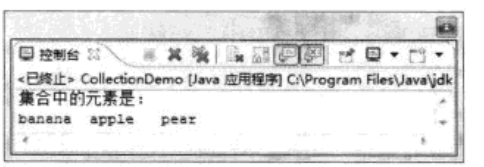


图 10.4　输出集合中的元素

10.2.3　Map 接口的常用方法

Map 接口提供了将键映射到值的对象。一个映射不能包含重复的键；每个键最多只能映射到一个值。Map 接口中同样提供了集合的常用方法，如 clear()、isEmpty()、size()等，除此之外还包括如表 10.3 所示的常用方法。

表 10.3　Map 接口中的常用方法

方　法	返　回　值	功 能 描 述
put(key k,value v)	Object	向集合中添加指定的 key 与 value 的映射关系
containskey(Object key)	boolean	如果此映射包含指定键的映射关系，则返回 true

续表

方　法	返 回 值	功 能 描 述
containsValue(Object value)	boolean	如果此映射将一个或多个键映射到指定值，则返回 true
get(Object key)	Object	如果存在指定的键对象，则返回该对象对应的值，否则返回 null
keySet()	Set	返回该集合中的所有键对象组成的 Set 集合
values()	Collection	返回该集合中所有值对象形成的 Collection 集合

由于 Map 集合中的元素是通过 key、value 进行存储的，要获取集合中指定的 key 值或 value 值，需要先通过相应的方法获取 key 集合或 value 集合，再遍历 key 集合或 value 集合获取指定值。

【例 10.4】 向一个 Map 集合中插入元素并根据 key 的值打印集合中的元素。（**实例位置：光盘\TM\sl\10\4**）

```
public class MapDemo {
    public static void main(String[] args) {
        Map<String, String> map = new HashMap<String, String>();          //创建 Map 集合
        map.put("1", "apple");                                             //向集合中添加对象
        map.put("2", "pear");
        map.put("3", "orange");
        for (int i = 1; i <= 3; i++) {
            System.out.println("第"+i+"元素是："+map.get("" + i + ""));    //输出对应位置的元素值
        }
    }
}
```

运行结果如图 10.5 所示。

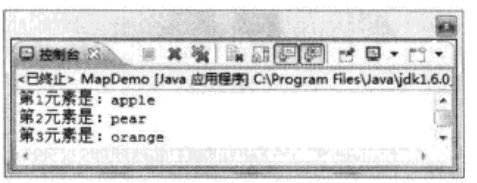


图 10.5　向 Map 集合中插入元素

10.2.4　范例 1：用 List 集合传递学生信息

集合在程序开发中经常用到，如在业务方法中将学生信息、商品信息等存储到集合中，然后作为方法的返回值返回给调用者，以此传递大量有序数据。本范例将使用 List 集合在方法之间传递学生信息。运行结果如图 10.6 所示。（**实例位置：光盘\TM\sl\10\5**）

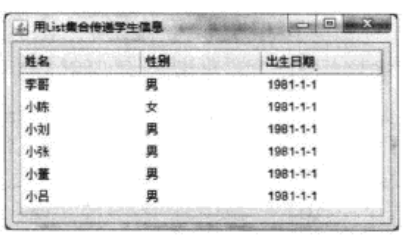


图 10.6　将学生信息添加到集合中

（1）在项目中新建窗体类 ClassInfo。在窗体中添加滚动面板，这个面板将放置表格控件。代码如下：

```
public ClassInfo() {
    setTitle("\u7528List\u96C6\u5408\u4F20\u9012\u5B66\u751F\u4FE1\u606F");
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setBounds(100, 100, 392, 223);
    contentPane = new JPanel();
    contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
    contentPane.setLayout(new BorderLayout(0, 0));
    setContentPane(contentPane);
    JScrollPane scrollPane = new JScrollPane();
    contentPane.add(scrollPane, BorderLayout.CENTER);
    scrollPane.setViewportView(getTable());
}
```

（2）编写 getTable()方法，在该方法中创建表格对象，设置表格的数据模型，然后调用 getStudents()方法获取保存学生信息的集合对象，在遍历该集合对象的同时把每个元素添加到表格模型的行，并显示到表格控件中。代码如下：

```
private JTable getTable() {
    if (table == null) {
    table = new JTable();                                          //创建表格控件
    table.setRowHeight(23);                                        //设置行高度
    String[] columns={"姓名","性别","出生日期"};                    //创建列名数组
    DefaultTableModel model=new DefaultTableModel(columns,0);      //创建表格模型
    table.setModel(model);                                         //设置表格模型
    List<String> students = getStudents();                         //调用方法传递 List 集合对象
    for (String info : students) {                                 //遍历学生集合对象
        String[] args = info.split(",");                           //把学生信息拆分为数组
        model.addRow(args);                                        //把学生信息添加到表格的行
        }
    }
    return table;
}
```

（3）编写 getStudents()方法，该方法将向调用者传递 List 集合对象，方法中为集合对象添加了多个元素，每个元素值都是一个学生信息，其中包括姓名、性别、出生日期。代码如下：

```
private List<String> getStudents() {
    List<String> list=new ArrayList<String>();                     //创建 List 集合对象
    list.add("李哥,男,1981-1-1");                                   //添加数据到集合对象
    list.add("小陈,女,1981-1-1");
    list.add("小刘,男,1981-1-1");
    list.add("小张,男,1981-1-1");
    list.add("小董,男,1981-1-1");
    list.add("小吕,男,1981-1-1");
    return list;
}
```

10.2.5 范例 2：Map 集合二级联动

Map 集合可以保存键值映射关系，这非常适合本范例所需要的数据结构，所有省份信息可以保存为 Map 集合的键，而每个键可以保存对应的城市信息，本范例就利用这个 Map 集合实现了省市级联选择框，当选择省份信息时，将改变城市下拉列表框对应的内容。运行结果如图 10.7 所示。（**实例位置：光盘\TM\sl\10\6**）

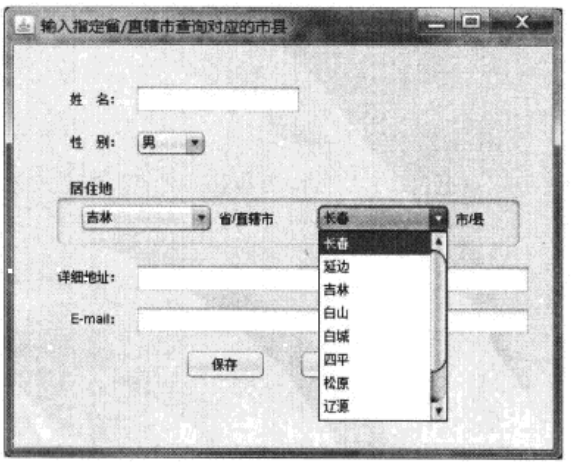


图 10.7 省市级联选择框

（1）在项目中新建窗体类 CityMap，在该类中创建并初始化 Map 集合对象，在该集合对象中保存各省市的关联信息。代码如下：

```
public class CityMap {
    public static Map<String,String[]> model=new LinkedHashMap();
    static{
        model.put("北京", new String[]{"北京"});
        model.put("上海", new String[]{"上海"});
        model.put("天津", new String[]{"天津"});
        model.put("重庆", new String[]{"重庆"});
        model.put("黑龙江", new String[]{"哈尔滨","齐齐哈尔","牡丹江","大庆","伊春","双鸭山","鹤岗","鸡西","佳
木斯","七台河","黑河","绥化","大兴安岭"});
        model.put("吉林", new String[]{"长春","延边","吉林","白山","白城","四平","松原","辽源","大安","通化"});
        model.put("辽宁", new String[]{"沈阳","大连","葫芦岛","旅顺","本溪","抚顺","铁岭","辽阳","营口","阜新","
朝阳","锦州","丹东","鞍山"});
        //省略类似代码
    }
}
```

（2）创建 MainFrame 主窗体类，在该类中添加 3 个文本框用于输入姓名、详细地址和 E-mail 信息，添加选择性别的下拉列表框，添加“保存”和“重置”按钮，最后添加两个核心控件，也就是选

择省份与选择城市的下拉列表框。

（3）编写 getProvince()方法，在该方法中获取 Map 集合的键映射，也就是省份信息的 Set 集合，然后将该集合转换为数组，并作为方法的返回值，这个方法将在省份下拉列表框的初始化代码中调用。代码如下：

```
public Object[] getProvince() {
    Map<String, String[]> map = CityMap.model;          //获取省份信息保存到 Map 中
    Set<String> set = map.keySet();                     //获取 Map 集合中的键，并以 Set 集合返回
    Object[] province = set.toArray();                  //转换为数组
    return province;                                    //返回获取的省份信息
}
```

（4）编写选择省份的下拉列表框的事件处理方法，该方法在省份下拉列表框改变选项时被调用，方法首先获取下拉列表框的选项值，然后把该值作为键并到 Map 集合中查找对应该键的值，返回结果是对应省份的所有城市名称组成的数组，最后用这个数组创建一个数据模型添加到城市下拉列表框控件中，以更新内容。代码如下：

```
private void itemChange() {
    String selectProvince = (String) comboBox.getSelectedItem();
    cityComboBox.removeAllItems();                          //清空市/县列表
    String[] arrCity = getCity(selectProvince);             //获取市/县
    cityComboBox.setModel(new DefaultComboBoxModel(arrCity));   //重新添加市/县列表的值
}
```

（5）编写 getCity()方法，该方法主要负责获取对应省份的城市数组，它将在省份下拉列表框控件的事件处理方法中被调用。代码如下：

```
public String[] getCity(String selectProvince) {
    Map<String, String[]> map = CityMap.model;          //获取省份信息保存到 Map 中
    String[] arrCity = map.get(selectProvince);         //获取指定键的值
    return arrCity;                                     //返回获取的市/县
}
```

10.3　集合类接口的实现类

视频讲解：光盘\TM\lx\10\集合类接口的实现类.exe

10.3.1　List 接口的实现类

要使用 List 集合，通常情况下需要声明为 List 类型，然后通过 List 接口的实现类来对集合进行实例化。List 接口的实现类常用的有 ArrayList 与 LinkedList。

1. ArrayList 类

该类实现了可变的数组，允许所有元素，包括 null。可以根据索引位置对集合进行快速的随机访问。缺点是向指定的索引位置插入对象或删除对象的速度较慢。语法格式如下：

```
List<String> list = new ArrayList<String>();
```

2. LinkedList 类

该类采用链表结构保存对象。这种结构的优点是便于向集合中插入和删除对象，经常需要向集合中插入、删除对象时，使用 LinkedList 类实现的 List 集合的效率较好；但对于随机访问集合中的对象，使用 LinkedList 类实现 List 集合的效率较慢。语法格式如下：

```
List<String> list2 = new LinkedList<String>();
```

使用 List 集合时通常声明为 List 类型，可通过不同的实现类来实例化集合。

【例 10.5】 分别通过 ArrayList、LinkedList 类实例化 List 集合。

```
List list = new ArrayList();
List list2 = new LinkedList();
```

【例 10.6】 在项目中创建 Gather 类，在主方法中创建集合对象，通过 Math 类的 random()方法随机获取集合中的某个元素，然后移除数组中索引位置为 2 的元素，最后遍历数组。（**实例位置：光盘\TM\sl\10\7**）

```
public class Gather {                                          //创建 Gather 类
    public static void main(String[] args) {                   //主方法
        List list = new ArrayList();                           //创建集合对象
        int i = (int) (Math.random() * (list.size()-1));       //获得 0～2 之间的随机数
        list.add("a");                                         //向集合中添加元素
        list.add("b");
        list.add("c");
        System.out.println("随机获取数组中的元素："+list.get(i));
        list.remove(2);                                        //将指定索引位置的元素从集合中移除
        System.out.println("将索引是'2'的元素从数组移除后，数组中的元素是：");
        for (int j = 0; j < list.size(); j++) {                //循环遍历集合
            System.out.print(list.get(j)+"   ");
        }
    }
}
```

运行结果如图 10.8 所示。

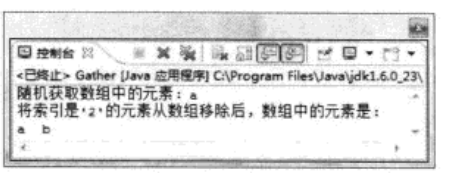


图 10.8 移除集合中索引为 2 的元素

10.3.2 Set 接口的实现类

要使用 Set 集合，通常情况下需要声明为 Set 类型，然后

通过 Set 接口的实现类来实例化。Set 接口的实现类常用的有 HashSet 和 TreeSet 类。语法格式如下：

```
Set<String> collSet = new HashSet<String>();
Set<String> collSet2 = new TreeSet<String>();
```

由于 Set 集合中的对象是无序的，遍历 Set 集合的结果与插入 Set 集合的顺序并不相同。

【例 10.7】　遍历输出 HashSet 中的全部元素。（**实例位置：光盘\TM\sl\10\8**）

（1）创建 People 对象，该对象中包含有 String 类型与 long 类型属性，并包含有属性的 setXXX()与 getXXX()方法。代码如下：

```
public class People {
    private String name;
    private long id_card;
    public People(String name, long id_card) {
        this.name = name;
        this.id_card = id_card;
    }
    public long getId_card() {
        return id_card;
    }
    public void setId_card(long id_card) {
        this.id_card = id_card;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
}
```

（2）在 main()方法中，创建 Set 集合对象，并向集合中添加对象，通过迭代器将集合中的元素输出。代码如下：

```
public class CollectionDemo {
    public static void main(String[] args) {
        Set<People> hashSet = new HashSet<People>();          //创建 Set 集合对象
        hashSet.add(new People("陈同学", 201101));            //向集合中添加对象
        hashSet.add(new People("王同学", 201102));
        hashSet.add(new People("李同学", 201103));
        Iterator<People> it = hashSet.iterator();             //创建集合迭代器
        System.out.println("集合中的元素是：");
        while (it.hasNext()) {                                //循环遍历迭代器
            People person = it.next();
            System.out.println(person.getName() + "   " + person.getId_card());
        }
    }
}
```

运行结果如图 10.9 所示。

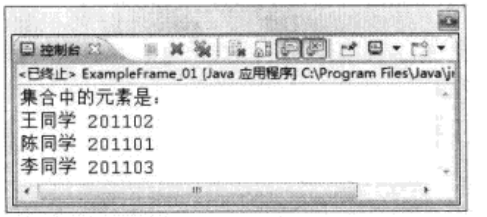


图 10.9　输出集合中所有的元素

10.3.3　Map 接口的实现类

Map 接口常用的实现类有 HashMap 和 TreeMap。通常建议使用 HashMap 实现类实现 Map 集合，因为由 HashMap 类实现的 Map 集合对于添加和删除映射关系效率更高。HashMap 是基于哈希表的 Map 接口的实现，HashMap 通过哈希码对其内部的映射关系进行快速查找；由 HashMap 类实现的 Map 集合对于添加或删除映射关系效率较高；而 TreeMap 中的映射关系存在一定的顺序，如果希望 Map 集合中的对象存在一定的顺序，应该使用 TreeMap 类实现 Map 集合。

1．HashMap 类

该类基于哈希表的 Map 接口的实现，此实现提供所有可选的映射操作，并允许使用 null 值和 null 键，但必须保证键的唯一性。HashMap 通过哈希码对其内部的映射关系进行快速查找。此类不保证映射的顺序，特别是不保证该顺序恒久不变。

2．TreeMap 类

该类不仅实现了 Map 接口，还实现了 java.util.SortedMap 接口，因此集合中的映射关系具有一定的顺序。但在添加、删除和定位映射关系上，TreeMap 类比 HashMap 类的性能差一些。由于 TreeMap 类实现的 Map 集合中的映射关系是根据键对象按照一定的顺序排列的，因此不允许键对象是 null。

可以通过 HashMap 类创建 Map 集合，当需要顺序输出时，再创建一个完成相同映射关系的 TreeMap 类实例。

【例 10.8】　通过 HashMap 类实例化 Map 集合，并遍历该 Map 集合，然后创建 TreeMap 实例实现将集合中的元素顺序输出。（**实例位置：光盘\TM\sl\10\9**）

（1）首先创建 Emp 类，代码如下：

```
public class Emp {
    private String e_id;
    private String e_name;
    public Emp( String e_id,String e_name) {
        this.e_id = e_id;
        this.e_name = e_name;
    }
    /**********************省略了属性的 setXXX()以及 getXXX()方法*******************************/
}
```

（2）创建一个用于测试的主类。首先新建一个 Map 集合，并添加集合对象，分别遍历由 HashMap 与 TreeMap 类实现的 Map 集合，观察两者的不同点。代码如下：

```
public class MapText {                                    //创建 MapText 类
    public static void main(String[] args) {              //主方法
```

```
        Map map = new HashMap();                                    //由 HashMap 实现的 Map 对象
        Emp emp = new Emp("001", "张三");
        Emp emp2 = new Emp("005", "李四");                          //创建 Emp 对象
        Emp emp3 = new Emp("004", "王一");
        map.put(emp.getE_id(), emp.getE_name());
        map.put(emp2.getE_id(), emp2.getE_name());                  //将对象添加到集合中
        map.put(emp3.getE_id(), emp3.getE_name());
        Set set = map.keySet();                                     //获取 Map 集合中的 key 对象集合
        Iterator it = set.iterator();
        System.out.println("HashMap 类实现的 Map 集合，无序：");
        while (it.hasNext()) {
            String str = (String) it.next();
            String name =(String) map.get(str);                     //遍历 Map 集合
            System.out.println(str + " "+name);
        }
        TreeMap treemap = new TreeMap();                            //创建 TreeMap 集合对象
        treemap.putAll(map);                                        //向集合添加对象
        Iterator iter = treemap.keySet().iterator();
        System.out.println("TreeMap 类实现的 Map 集合，键对象升序：");
        while (iter.hasNext()) {                                    //遍历 TreeMap 集合对象
            String str = (String) iter.next();                      //获取集合中的所有 key 对象
            String name =(Strirg) map.get(str);                     //获取集合中的所有 values 值
            System.out.println(str+" "+name);
        }
    }
}
```

运行结果如图 10.10 所示。

图 10.10　输出 Map 集合中的元素

10.3.4　范例 3：for 循环遍历 ArrayList

在使用集合类时，不仅要了解容器是如何保存元素的，而且要知道如何取出元素。本范例将使用普通 for 循环遍历 ArrayList，从中取出所有序号为奇数的元素。运行结果如图 10.11 所示。（**实例位置：光盘\TM\sl\10\10**）

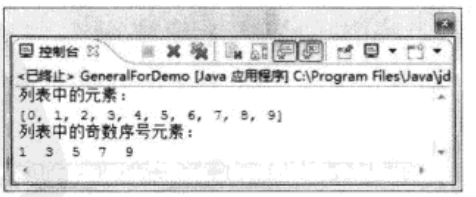


图 10.11　输出序号为奇数的元素

在项目中新建 GeneralForDemo 类，在类中创建一个 ArrayList 集合为其指定泛型为 Integer 类型，

并为其添加 10 个元素，利用 for 循环遍历 ArrayList 集合，输出列表中序号为奇数的元素。代码如下：

```
public class GeneralForDemo {
    public static void main(String[] args) {
        List<Integer> list = new ArrayList<Integer>();                    //创建列表
        for (int i = 0; i < 10; i++) {                                    //向列表中增加 10 个元素
            list.add(i);
        }
        System.out.println("列表中的元素：" + list);                      //输出列表中全部的元素
        System.out.println("列表中的奇数序号元素：");
        for (int i = 1; i < list.size(); i += 2) {                        //输出列表中序号为奇数的元素
            System.out.print(list.get(i)+"   ");
        }
    }
}
```

10.3.5 范例 4：用动态数组保存学生姓名

Java 语言中提供了各种数据集合类，这些类主要用于保存复杂结构的数据，其中 ArrayList 集合可以看作动态数组。它突破普通数组固定长度的限制，可以随时向数组中添加和移除元素，这将使数组更加灵活，如果要获取普通数组，还可以通过该类的 toArray()方法获得。本范例通过这个 ArrayList 集合类实现向程序动态添加与删除学生姓名的功能，其中所有数据都保存在 ArrayList 集合的实例对象中。运行结果如图 10.12 所示。（**实例位置：光盘\TM\sl\10\11**）

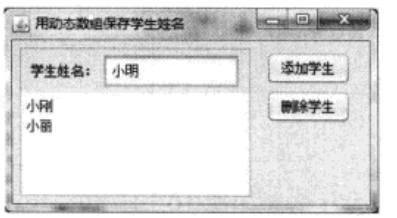


图 10.12　用动态数组保存学生姓名

（1）在项目中新建 DynamicArray 窗体类，在窗体中添加文本框控件、列表控件、“添加学生”按钮和“删除学生”按钮。编写“添加学生”按钮的事件处理方法，在该方法中获取用户在文本框中输入的字符串，然后将这个字符串添加到 ArrayList 集合中，再调用 replaceModel()方法把集合中的数据显示到窗体的列表控件中。代码如下：

```
protected void do_button_actionPerformed(ActionEvent e) {
    textField.requestFocusInWindow();
    textField.selectAll();                          //选择文本框中的文本准备下次输入
    String text = textField.getText();              //获取用户输入的姓名
    if (text.isEmpty())                             //过滤为输入姓名的情况
        return;
    arraylist.add(text);                            //把姓名添加到数组集合中
    replaceModel();                                 //把数组集合中的内容显示到界面列表控件中
}
```

（2）编写“删除学生”按钮的事件处理方法，在该方法中获取列表控件的当前选择项，然后从

ArrayList 集合中移除这个选项的值，最后调用 replaceModel()方法把集合中的数据显示到窗体的列表控件中。代码如下：

```
protected void do_button_1_actionPerformed(ActionEvent e) {
    Object value = list.getSelectedValue();                    //获取列表控件的选择项
    arraylist.remove(value);                                   //从数组集合中移除用户的选择项
    replaceModel();                                            //把数组集合中的内容显示到界面列表控件中
}
```

（3）编写 replaceModel()方法，在该方法中重新设置列表控件的模型，模型要读取 ArrayList 集合的元素并显示到列表控件中。代码如下：

```
private void replaceModel() {
    list.setModel(new AbstractListModel() {                    //为列表控件设置数据模型并显示数组集合中的数据
        @Override
        public int getSize() {                                 //获取数组大小
            return arraylist.size();
        }
        @Override
        public Object getElementAt(int index) {                //获取指定索引元素
            return arraylist.get(index);
        }
    });
}
```

10.4　迭　代　器

视频讲解：光盘\TM\lx\10\迭代器.exe

10.4.1　迭代器的创建和使用

利用 Iterator 接口创建迭代器，Iterator 接口位于 java.util 包下。Iterator 接口中有 3 个方法。

- ☑ hasNext()：如果仍有元素可以迭代，则返回 true。
- ☑ next()：返回迭代的下一个元素。
- ☑ remove()：从迭代器指向的 collection 中移除迭代器返回的最后一个元素（可选操作）。

ListIterator 接口继承了 Iterator 接口，下面通过一个简单的实例演示迭代器的用法。

【例 10.9】　创建 ListIteratorDemo 类，在 main()方法中创建了一个 ArrayList 对象，然后打印集合中的元素和位置等信息。（**实例位置：光盘\TM\sl\10\12**）

```
public class ListIteratorDemo {
    public static void main(String[] args) {
        //创建一个 ArrayList，其中能够保存的数据类型是 Integer
        ArrayList<Integer> array = new ArrayList<Integer>();
```

```
        //使用 Collections 类中的工具方法 addAll()向集合中增加元素
        Collections.addAll(array, 1, 2, 3, 4, 5, 6);
        System.out.println("集合中的元素：" + array);
        //使用无参数的方法获得 ListIterator 对象
        ListIterator<Integer> iterator = array.listIterator();
        //对于初始位置，判断是否具有下一个元素
        boolean hasNext = iterator.hasNext();
        System.out.println("集合是否具有下一个元素：" + hasNext);
        //对于初始位置，判断是否具有前一个元素
        boolean hasPrevious = iterator.hasPrevious();
        System.out.println("集合是否具有前一个元素：" + hasPrevious);
        int next = iterator.next();                                    //获得下一个元素
        System.out.println("获得集合的下一个元素：" + next);
        int nextIndex = iterator.nextIndex();                          //获得下一个元素的索引
        System.out.println("获得集合的下一个元素的索引：" + nextIndex);
        int previous = iterator.previous();                            //获得前一个元素
        System.out.println("获得集合的前一个元素：" + previous);
        int previousIndex = iterator.previousIndex();                  //获得前一个元素的索引
        System.out.println("获得集合的前一个元素的索引：" + previousIndex);
        iterator.add(7);                                               //向列表中增加元素 7
        System.out.println("向集合中增加元素 7 后的集合：" + array);
        iterator.next();                                               //获得下一个元素
        iterator.set(12);                                              //将获得的元素设置成 12
        System.out.println("将获得的下一个元素修改成 12 后的集合：" + array);
        iterator.remove();
        System.out.println("将获得的下一个元素删除后的集合：" + array);
    }
}
```

运行结果如图 10.13 所示。

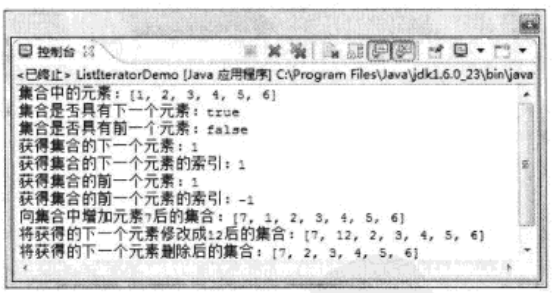


图 10.13　使用迭代器输入 ArrayList 中的元素

10.4.2　范例 5：Iterator 遍历 ArrayList

ArrayList 在以后的开发中会经常用到，本范例将练习使用 Iterator 和 for 循环组合遍历集合。运行结果如图 10.14 所示。（**实例位置：光盘\TM\sl\10\13**）

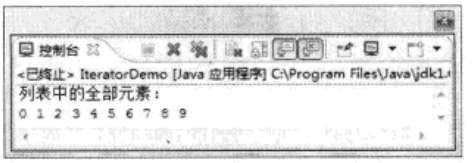


图 10.14　使用 Iterator 遍历 ArrayList

在项目中新建 IteratorDemo 类，在类中创建一个 ArrayList 集合为其指定泛型为 Integer 类型，并为其添加 10 个元素，利用迭代器遍历 ArrayList 集合，其循环条件为如果迭代器中仍有元素可以迭代则继续循环，如果没有则跳出循环。代码如下：

```
public class IteratorDemo {
    public static void main(String[] args) {
        List<Integer> list = new ArrayList<Integer>();          //创建列表
        for (int i = 0; i < 10; i++) {                          //向列表中增加 10 个元素
            list.add(i);
        }
        System.out.println("列表中的全部元素：");
        for(Iterator<Integer> it = list.iterator();it.hasNext();) {
            System.out.print(it.next()+" ");
        }
    }
}
```

10.4.3　范例 6：ListIterator 遍历 ArrayList

对于列表而言，除了 Iterator，Java 语言还提供了一个功能更加强大的 ListIterator，它可以实现逆序遍历列表中的元素。本范例将使用其逆序遍历 ArrayList。运行结果如图 10.15 所示。（**实例位置：光盘\TM\sl\10\14**）

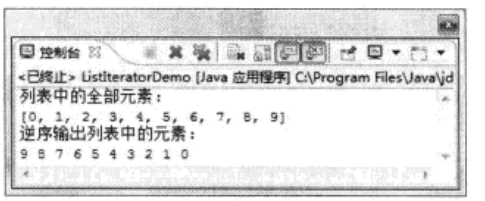


图 10.15　逆序遍历 ArrayList

在项目中新建 ListIteratorDemo 类，在类中创建一个 ArrayList 集合为其指定泛型为 Integer 类型，并为其添加 10 个元素，获得迭代器对象后利用 hasPrevious()方法逆序输出 ArrayList 集合中的元素。代码如下：

```
public class ListIteratorDemo {
    public static void main(String[] args) {
        List<Integer> list = new ArrayList<Integer>();          //创建列表
        for (int i = 0; i < 10; i++) {                          //向列表中增加 10 个元素
            list.add(i);
        }
        System.out.println("列表中的全部元素：" + list);
        System.out.println("逆序输出列表中的元素：");
```

```
        ListIterator<Integer> li = list.listIterator();                    //获得 ListIterator 对象
        for (li = list.listIterator(); li.hasNext();) {                    //将游标定位到列表结尾
            li.next();
        }
        for (; li.hasPrevious();) {                                        //逆序输出列表中的元素
            System.out.print(li.previous() + " ");
        }
    }
}
```

10.5 经典范例

10.5.1 经典范例 1：制作电子词典

视频讲解：光盘\TM\lx\i0\制作电子词典.exe

词典通常用于解释一个词的含义，这是一种映射关系，因此可以使用 Map 来实现。本范例将制作一个电子词典。运行结果如图 10.16 所示。**（实例位置：光盘\TM\sl\10\15）**

图 10.16 电子词典

（1）在项目中新建窗体类 DictionaryDemo，在窗体中添加标签等控件。代码如下：

```
public DictionaryDemo() {
    addWindowListener(new WindowAdapter() {
        @Override
        public void windowActivated(WindowEvent e) {
            do_this_windowActivated(e);
        }
    });
    setTitle("\u6211\u7684\u7535\u5B50\u8BCD\u5178");
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setBounds(100, 100, 300, 200);
    contentPane = new JPanel();
    contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
    contentPane.setLayout(new BorderLayout(0, 0));
    setContentPane(contentPane);

    JPanel panel1 = new JPanel();
    contentPane.add(panel1, BorderLayout.NORTH);

    JLabel label = new JLabel("\u8BF7\u8F93\u5165\u8981\u67E5\u8BE2\u7684\u5355\u8BCD\uFF1A");
    panel1.add(label);

    textField = new JTextField();
    panel1.add(textField);
```

```
        textField.setColumns(10);

        JPanel panel2 = new JPanel();
        contentPane.add(panel2, BorderLayout.SOUTH);

        JButton button = new JButton("\u67E5\u8BE2");
        button.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                do_button_actionPerformed(e);
            }
        });
        panel2.add(button);

        JScrollPane scrollPane = new JScrollPane();
        contentPane.add(scrollPane, BorderLayout.CENTER);

        textArea = new JTextArea();
        textArea.setEditable(false);
        scrollPane.setViewportView(textArea);
}
```

（2）编写 do_this_windowActivated()方法完成窗体激活事件监听，在该方法中向 Map 中增加数据。代码如下：

```
protected void do_this_windowActivated(WindowEvent e) {
    words = new HashMap<String, String>();
    words.put("apple", "苹果");
    words.put("banana", "香蕉");
    words.put("water", "水");
}
```

（3）编写 do_button_actionPerformed()方法，用来响应按钮单击事件。在该方法中完成了对单词的查询操作，并根据不同的情况进行了提示。代码如下：

```
protected void do_button_actionPerformed(ActionEvent e) {
    String text = textField.getText();                          //获得用户输入的单词
    if (text.isEmpty()) {                                       //如果用户输入为空则提示用户
        JOptionPane.showMessageDialog(this, "请输入要查询的单词！", null,
JOptionPane.WARNING_MESSAGE);
        return;
    }
    String meaning = words.get(text);                           //查询单词的含义
    if (meaning == null) {                                      //如果没有这个单词则提示用户
        JOptionPane.showMessageDialog(this, "要查询的单词不存在！", null,
JOptionPane.WARNING_MESSAGE);
        return;
    } else {
        textArea.setText(meaning);                              //显示单词的含义
    }
}
```

10.5.2 经典范例 2：制作手机电话本

视频讲解：光盘\TM\lx\10\制作手机电话本.exe

为了便于保存联系方式，手机都提供电话簿功能。它也是一种映射关系，因此可以使用 Map 来实现。本范例将制作一个手机电话簿。运行结果如图 10.17 所示。（**实例位置：光盘\TM\sl\10\16**）

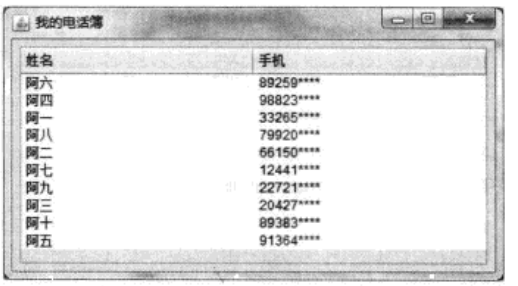


图 10.17 手机电话簿

（1）在项目中新建窗体类 Phonebook，在窗体中添加表格控件。代码如下：

```
public Phonebook() {
        addWindowListener(new WindowAdapter() {
            @Override
            public void windowActivated(WindowEvent e) {
                do_this_windowActivated(e);
            }
        });
        setTitle("\u6211\u7684\u7535\u8BDD\u7C3F");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setBounds(100, 100, 450, 250);
        contentPane = new JPanel();
        contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
        contentPane.setLayout(new BorderLayout(0, 0));
        setContentPane(contentPane);

        JScrollPane scrollPane = new JScrollPane();
        contentPane.add(scrollPane, BorderLayout.CENTER);

        table = new JTable();
        scrollPane.setViewportView(table);
}
```

（2）编写 do_this_windowActivated()方法完成窗体激活事件监听，在该方法中向表格模型中增加数据，最后更新表格模型。代码如下：

```
protected void do_this_windowActivated(WindowEvent e) {
    Map<String, String> directory = new HashMap<String, String>();          //创建集合
    directory.put("阿一", "33265****");                                      //向集合中增加元素
```

```
    directory.put("阿二", "66150****");                                    //向集合中增加元素
    directory.put("阿三", "20427****");                                    //向集合中增加元素
    directory.put("阿四", "98823****");                                    //向集合中增加元素
    directory.put("阿五", "91364****");                                    //向集合中增加元素
    directory.put("阿六", "89259****");                                    //向集合中增加元素
    directory.put("阿七", "12441****");                                    //向集合中增加元素
    directory.put("阿八", "79920****");                                    //向集合中增加元素
    directory.put("阿九", "22721****");                                    //向集合中增加元素
    directory.put("阿十", "89383****");                                    //向集合中增加元素
    DefaultTableModel model = (DefaultTableModel) table.getModel();       //获得表格模型
    model.setColumnIdentifiers(new Object[] { "姓名", "手机" });           //设置表头
    Set<String> names = directory.keySet();                               //获得键集合
    for (Iterator<String> it = names.iterator(); it.hasNext();) {
        String name = it.next();                                          //获得键
        model.addRow(new Object[] { name, directory.get(name) });         //向表格中增加元素
    }
    table.setModel(model);                                                //更新表格模型
}
```

10.6　本 章 小 结

本章向读者介绍了 Java 语言中常见的集合，包括 List 集合、Set 集合、Map 集合。对于每种集合的特点读者应该有所了解，重点掌握集合的遍历、添加对象、删除对象的方法。本章在介绍每种集合时都给出了典型实用的小例子，以帮助读者掌握集合类的常用方法。集合是 Java 语言中很重要的部分，通过本章的学习，读者应该学会使用集合类。

10.7　实 战 练 习

1．将 1～100 之间的所有正整数存放在一个 List 集合中，并将集合中索引位置是 10 的对象从集合中移除。（**答案位置：光盘\TM\sl\10\17**）

2．分别向 Set 集合以及 List 集合中添加“A”、“a”、“c”、“C”、“a”5 个元素，观察重复值“a”能否在 List 集合以及 Set 集合中成功添加。（**答案位置：光盘\TM\sl\10\18**）

3．创建 Map 集合，创建 Emp 对象，并将创建的 Emp 对象添加到集合中（Emp 对象的 id 作为 Map 集合的键），并将 id 为 005 的对象从集合中移除。（**答案位置：光盘\TM\sl\10\19**）

第11章

异常处理

（视频讲解：49分钟）

在程序中总是存在各种错误，使应用程序在运行时终止。为了在程序执行过程中发生错误时能正常运行，可以使用 Java 提供的异常处理机制捕获可能发生的异常，对异常进行处理并使程序能正常运行。

通过阅读本章，您可以：

- 了解异常的概念
- 了解异常的分类
- 掌握如何获取异常信息
- 掌握如何处理异常
- 了解如何抛出异常
- 了解自定义异常
- 了解异常的使用原则

PDG

11.1 异常概述

视频讲解：光盘\TM\lx\11\异常概述.exe

假设一辆轿车发生了故障，可能是某个零件发生了问题，也可能是没有油了。如果是由于零件问题，只需要更换零件就可以解决；如果是没有油了，只需要加满油就可以正常行驶了。程序中的异常与此类似，就对程序中可能发生异常的语句进行处理，使程序能够正常执行。

在程序开发过程中，可能存在各种错误，有些错误是可以避免的，而有些错误却是意想不到的，在 Java 中把这些可能发生的错误称为异常。图 11.1 说明了异常类的继承关系。

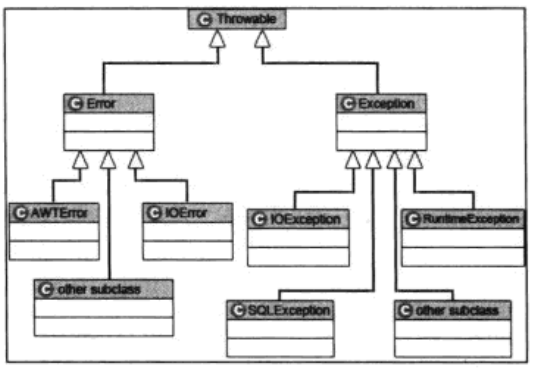


图 11.1 异常类的继承关系图

从图 11.1 中可以看出，Throwable 类是所有异常类的超类，该类的两个直接子类是 Error 和 Exception。其中，Error 及其子类用于指示合理的应用程序不应该试图捕获的严重问题，Exception 及其子类给出了合理的应用程序需要捕获的异常。

说明 图 11.1 中列出了异常类的继承关系，由于界面大小有限，而 Error 类和 Exception 类的子类又比较多，所以这里使用 other subclass 代表 Error 类和 Exception 类的其他子类。

11.2 异常的分类

视频讲解：光盘\TM\lx\11\异常的分类.exe

在 Java 中可以捕获的异常（即 Exception 类的子类）分为可控式异常和运行时异常两种类型。

11.2.1 可控式异常

在 Java 中把那些可以预知的错误，例如从文件中读取数据、对数据库进行操作等，在程序编译时就能对程序中可能存在的错误进行处理，并给出具体的错误信息，我们把这些错误称为可控式异常。表 11.1 中列出了常用的可控式异常及说明。

表 11.1　常用可控式异常及说明

异　　常	说　　明
IOException	当发生某种 I/O 异常时，抛出此异常
SQLException	提供关于数据库访问错误或其他错误信息的异常
ClassNotFoundException	类没有找到异常
NoSuchFieldException	类不包含指定名称的字段时产生的信号
NoSuchMethodException	无法找到某一特定方法时，抛出该异常

【例 11.1】　创建项目 01，在项目中创建 Example_01 类，在该类中加载一个不存在的类，观察发生的异常。（**实例位置：光盘\TM\sl\11\1**）

```
public class Example_01 {
    private int num=10;                                  //成员变量
    public int getNum() {                                //成员方法
        return num;                                      //返回成员变量的值
    }
    public void setNum(int num) {                        //成员方法
        this.num = num;                                  //设置成员变量的值
    }
    public Example_01(){                                 //类的构造方法
        try {
            Class.forName("com.mingrisoft.Test");        //装载 com.mingrisoft 包中的 Test 类
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
        System.out.println("测试。");                    //在控制台输出“测试。”
    }
    public static void main(String[] args) {
        Example_01 exam=new Example_01();                //创建类的实例
        exam.setNum(888);                                //调用 setNum()方法设置成员变量 num 的值为 888
        System.out.println(exam.getNum());               //调用 getNum()方法输出成员变量的值 888
    }
}
```

运行结果如图 11.2 所示。

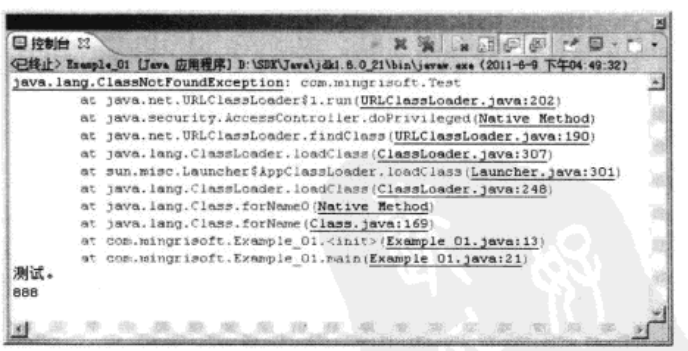


图 11.2　ClassNotFoundException 异常

说明 由于在构造方法中加载 com.mingrisoft 包中的 Test 类时，Java 的虚拟机没有找到 Test 类，从而引发了 ClassNotFoundException 异常，所以在控制台输出了 ClassNotFoundException 这样的异常信息，由于在执行“Class.forName("com.mingrisoft.Test");”语句时发生了异常，所以导致程序中的其他代码没有执行，所以在控制台并没有输出“测试。”和“888”。

11.2.2 运行时异常

在 Java 中有些错误是不能被编译器检测到的，例如，在进行除法运算时，除数为零；试图把一个不是由数字组成的字符串使用 Integer 类的 parseInt()方法转换为整数等，Java 的编译器是检测不到的，因而能够正常编译，但是在运行时就会发生异常，我们把这些异常称为运行时异常。表 11.2 列出了常用的运行时异常及说明。

表 11.2 常用的运行时异常及说明

方　法	说　明
IndexOutOfBoundsException	指示某集合或数组的索引值超出范围时抛出该异常
NullPointerException	当应用程序试图在需要对象的地方使用 null 时，抛出该异常
ArithmeticException	当出现异常的运算条件时，抛出此异常
IllegalArgumentException	抛出的异常表明向方法传递了一个不合法或不正确的参数
ClassCastException	当试图将对象强制转换为不是实例的子类时，抛出该异常

【例 11.2】 通过 Integer 类的实例创建 Object 对象 o，观察将 Object 对象 o 强制转换为字符串时发生的异常。

```
Object o = new Integer(0);                    //通过 Integer 类的实例创建 Object 对象 o
System.out.println((String)o);                //将 Object 对象 o 强制转换为字符串时出错
```

说明 上面代码首先创建一个 Object 对象 o，该对象是通过 Object 的子类 Integer 创建的，也就是说 Object 对象 o 是子类 Integer 的实例，因此当使用 String 类将对象 o 强制转换为字符串时出错了，因为对象 o 是 Object 的子类 Integer 创建的，而不是 String 类创建的，所以产生了错误，如果将对象 o 强制转换为 Integer 类型就不会出错，因为对象 o 是 Integer 类创建的。

【例 11.3】 在项目中创建 Example_02 类，在该类中创建一个数组，然后使用超出数组下标范围的值访问数组中的元素，观察发生的异常。（**实例位置：光盘\TM\sl\11\2**）

```
int[] number={ 100, 80, 50, 70, 20, 60};      //创建并初始化具有 6 个元素的数组
public void setNum(int index,int value){      //根据索引值 index 为数组元素赋值 value 的方法
    number[index]=value;                      //为数组中索引值 index 处的元素赋值 value
}
public int getNum(int index){                 //根据索引值 index 获得数组中对应元素值的方法
```

```
        return number[index];                          //获得数组中索引值 index 处的元素值
    }
    public static void main(String[] args) {
        Example_02 ex=new Example_02();                //创建类的实例
        //调用方法获得数组中索引值 0 处的元素值，即第一个元素的值 100
        int value=ex.getNum(0);
        System.out.println(value);                     //输出第一个元素的值 100
        //索引值 6 超出了数组下标的范围，因此将发生异常，导致程序终止，下面的代码将不会被执行
        value=ex.getNum(6);
        System.out.println(value);
        //获得数组中最后一个元素的值 60
        value=ex.getNum(5);
        System.out.println(value);                     //输出最后一个元素值 60
    }
```

运行结果如图 11.3 所示。

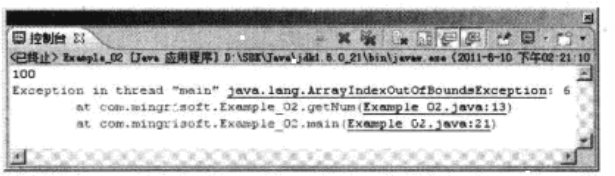


图 11.3　ArrayIndexOutOfBoundsException 异常

说明　从输出结果可以看出，程序发生了 ArrayIndexOutOfBoundsException 异常，该异常是 IndexOutOfBoundsException 异常的子异常。当程序执行到语句“value=ex.getNum(6);”时发生了异常，因为数组中有 6 个元素，数组下标的范围是从 0～5，所以用 6 作为数组下标的索引值发生了错误，因而导致下面的其他语句不会被执行，所以程序中只输出了数组中第一个元素的值 100，而没有输出最后一个元素的值 60。

11.2.3　范例 1：算术异常

算术异常即 ArithmeticException，是指整数被 0 除产生的异常。在 Java 语言中，如果一个整数被 0 除，那么将抛出 ArithmeticException，但是浮点数被 0 除，将不引发算术异常，这与数学中不同。本范例将演示出现算术异常的情况，并进行处理。运行结果如图 11.4 所示。（**实例位置：光盘\TM\sl\11\3**）

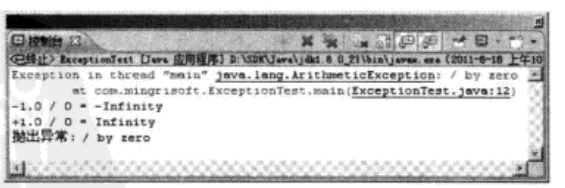


图 11.4　算术异常

（1）在 Eclipse 中创建项目 03，并在该项目中创建 com.mingrisoft 包。

（2）在 com.mingrisoft 包中创建类文件，名称为 ExceptionTest。在该类的主方法中，演示出现算术异常的情况。这里将第一条可能出现异常的语句应用 try…catch 语句捕获，并输出异常信息，第二条可能出现异常的语句不作处理。代码如下：

```
public class ExceptionTest {
    public static void main(String[] args) {
        System.out.println("-1.0 / 0 = " + (-1.0 / 0));          //演示负浮点数除 0
        System.out.println("+1.0 / 0 = " + (+1.0 / 0));          //演示正浮点数除 0
        try{
          System.out.println("-1 / 0 = " + (-1 / 0));            //演示负整数除 0
        }catch(Exception e){
          System.out.println("抛出异常："+e.getMessage());
        }
        System.out.println("+1 / 0 = " + (+1 / 0));              //演示正整数除 0
        System.out.println("输出结束。");
    }
}
```

说明 在 Java 的异常处理机制中，有一个默认处理异常的程序。当程序出现异常时，默认处理程序将显示一个描述异常的字符串，打印异常发生处的堆栈轨迹，并终止程序。

11.2.4 范例 2：数组下标越界异常

数组下标越界异常即 ArrayIndexOutOfBoundsException，当访问的数组元素的下标值大于数组的最大下标值时发生。也就是数组元素的下标值大于等于数组的长度时发生。本范例将演示出现数组下标越界异常（ArrayIndexOutOfBoundsException）的情况。运行结果如图 11.5 所示。（**实例位置：光盘\TM\sl\11\4**）

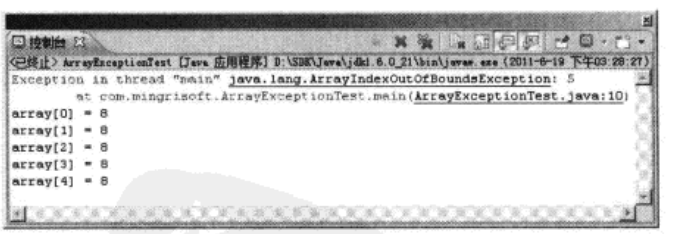


图 11.5 数组下标越界异常

（1）在 Eclipse 中创建项目 04，并在该项目中创建 com.mingrisoft 包。

（2）在 com.mingrisoft 包中创建类文件，名称为 ArrayExceptionTest。在该类的 main()方法中，首先声明一个长度为 5 的整型数组，并应用 Arrays 对象的 fill()方法，将数组中所有元素赋值为 8，然后应用 for 循环遍历输出所有数组元素。代码如下：

```
public class ArrayExceptionTest {
    public static void main(String[] args) {
        int[] array = new int[5];                    //声明一个长度为 5 的整型数组
        Arrays.fill(array, 8);                       //将新声明数组的所有元素赋值为 8
        for (int i = 0; i < 6; i++) {                //遍历输出所有数组元素
            System.out.println("array[" + i + "] = " + array[i]);
        }
    }
}
```

说明　如果要遍历数组中的全部元素，则推荐使用 foreach 循环，它可以避免数组的下标越界。如果要使用数组的下标，则需要记住数组的下标是从 0 开始计算的。如果需要使用数组的长度，则推荐使用 length 属性。另外使用 ArrayList 类也可以避免这些问题。

11.3　获取异常信息

视频讲解：光盘\TM\lx\11\获取异常信息.exe

获取异常信息就好比工厂里某个线路出现故障停电了，电工要从线路中找出故障的原因，就像程序中获取到了异常信息。

在 Java 中 java.lang.Throwable 类是所有异常类的超类，该类提供了获得异常信息的方法。表 11.3 中列出了获取异常信息的方法及说明。

表 11.3　获取异常信息的方法及说明

方　　法	说　　明
String getLocalizedMessage()	获得此 Throwable 的本地化描述
String getMessage()	获得此 Throwable 的详细消息字符串
void printStackTrace()	将此 Throwable 及其栈踪迹输出至标准错误流
String toString()	获得此 Throwable 的简短描述

【例 11.4】　在项目中创建 Example_03 类，在该类中使用表 11.3 中的方法输出进行除法运算时除数为 0 的异常信息。（**实例位置：光盘\TM\sl\11\5**）

```
public class Example_03 {
    public void printBugInfo(){                      //定义成员方法
        try{
            int x=100;                               //定义局部变量 x
            int y=0;                                 //定义局部变量 y
            int z=x / y;                             //计算 x 除以 y 的商
            //输出计算结果
            System.out.println(x+"除以"+y+"的商是："+z);
        }catch(Exception ex){
```

```
            ex.printStackTrace();                          //输出异常到标准错误流
            //使用 getMessage()方法输出异常信息
            System.out.println("getMessage 方法：      "+ex.getMessage());
            //使用 getLocalizedMessage()方法输出异常信息
            System.out.println("getLocalizedMessage 方法：      "+ex.getLocalizedMessage());
            //使用 toString()方法输出异常信息
            System.out.println("toString 方法：      "+ex.toString());
        }
    }
    public static void main(String[] args) {
        Example_03 ex=new Example_03();                    //创建类的实例
        ex.printBugInfo();                                 //调用方法
    }
}
```

运行结果如图 11.6 所示。

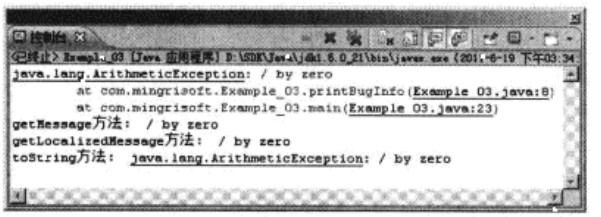


图 11.6　除数为 0 的异常信息

说明　输出结果是除数为 0 时发生的异常，其中前 3 行是 printStackTrace()方法输出的异常信息，第 4 行是 getMessage()方法输出的异常信息，第 5 行是 getLocalizedMessage()方法输出的异常信息，最后一行是 toString()方法输出的异常信息。在本实例的代码中使用了 try…catch 语句块来捕获程序中的异常信息，在 try 和 catch 之间两个大括号内是程序需要正常执行，但是却又有可能发生异常的代码，在 catch 后的两个大括号内是 try 和 catch 之间的代码发生错误时执行的代码，用于进行异常处理。

11.4　处 理 异 常

视频讲解：光盘\TM\lx\11\处理异常.exe

在 Java 语言中当程序发生异常时，可以使用 try…catch、try…catch…finally 或 try…finally 进行处理。接下来将对这 3 个语句块分别进行讲解。

11.4.1　使用 try…catch 处理异常

对于程序中可能发生异常的语句，可以将其添加到 try…catch 语句块中，这样当程序发生异常时，

就可以对其进行相应的处理。

try…catch 语句块的语法格式如下：

```
try{
    需要正常执行的语句
}catch(Exception ex){
    对异常进行处理的语句
}
```

- ☑ try 和 catch 是进行异常处理的关键字。
- ☑ try 和 catch 之间的两个大括号内是程序需要正常执行但又可能发生异常的语句。
- ☑ catch 后的两个小括号内是程序需要处理的异常类型。
- ☑ catch 后的两个大括号内是对程序发生的异常进行处理的语句。

11.4.2　使用 try…catch…finally 处理异常

对于程序中可能发生异常的语句，可以将其添加到 try…catch…finally 语句块中，这样当程序发生异常时，就可以对其进行相应的处理。

try…catch…finally 语句块的语法格式如下：

```
try{
    需要执行的语句
}catch(Exception ex){
    对异常进行处理的语句
}finally{
    一定会被处理的语句
}
```

- ☑ try、catch 和 finally 是进行异常处理的关键字。
- ☑ try 和 catch 之间的两个大括号内是程序需要正常执行但又可能发生异常的语句。
- ☑ catch 后的两个小括号内是程序需要处理的异常类型。
- ☑ catch 后的两个大括号内是对程序发生的异常进行处理的语句。
- ☑ finally 后的两个大括号内的语句，不管程序是否发生异常都要执行（也就是说程序执行完 try 和 catch 之间的语句或执行完 catch 后两个大括号内的语句都将执行 finally 后的语句），因此 finally 语句块通常用于执行垃圾回收、释放资源等操作。

技巧　在 Java 中进行异常处理时，应该尽量使用 finally 块进行资源回收，因为在 try…catch…finally 语句块中，不管程序是否发生异常，最终都会执行 finally 语句块，因此可以在 finally 块中添加释放资源的代码。

【例 11.5】　在项目中创建 IO 流，分配内存资源。使用完后，在 finally 中关闭 IO 流并释放内存资源。代码如下：（**实例位置：光盘\TM\sl\11\6**）

```
private FileInputStream in=null;                                    //声明 FileInputStream 对象 in
public void readInfo(){                                             //定义方法
    try{
        //创建 FileInputStream 对象 in
        in=new FileInputStream("src/com/mingrisoft/CloseIo.java");
        System.out.println("创建 IO 流，分配内存资源。");
    }catch(IOException io){
        io.printStackTrace();                                       //输出栈踪迹
        System.out.println("创建 IO 对象发生异常。");
    }finally{
        if (in!=null){
            try{
                in.close();                                         //关闭 FileInputStream 对象 in，释放资源
                System.out.println("关闭 IO 流，释放内存资源。");
            }catch(IOException ioe){
                ioe.printStackTrace();                              //输出栈踪迹
                System.out.println("关闭 IO 对象发生异常。");
            }
        }
    }
}
public static void main(String[] args) {
    CloseIo ex=new CloseIo();                                       //创建对象
    ex.readInfo();                                                  //调用 readInfo()方法
}
```

运行结果如图 11.7 所示。

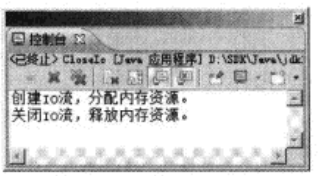


图 11.7　在控制台输出释放信息

说明 从输出结果可以看出，程序在 try 语句块中创建了 IO 对象，然后在 finally 语句块中关闭了 IO 对象，释放了内存资源。

11.4.3　使用 try…finally 处理异常

对于程序中可能发生异常的语句，可以将其添加到 try…finally 语句块中，这样当程序发生异常时，就可以在 finally 语句块中对其进行相应的处理，另外当程序没有发生异常时，执行完 try 和 finally 之间的语句后，也将执行 finally 语句块中的代码，因此可以在 finally 语句块中放置一些必须执行的代码，如释放内存资源的代码等。

try…finally 语句块的语法格式如下：

```
try{
    需要执行的语句
} finally{
    一定会被处理的语句
}
```

- ☑ try 和 finally 是进行异常处理的关键字。
- ☑ try 和 finally 之间的两个大括号内是程序需要正常执行但又可能发生异常的语句。
- ☑ finally 后两个大括号内的语句是不管程序是否发生异常最终都要执行的语句，因此 finally 语句块通常用于放置程序中必须执行的代码，如关闭数据库连接、关闭 IO 流等。

说明　在有 try…finally 语句块的程序中，只要程序执行了 try 语句块中的代码，不管 try 语句块是否发生异常，与该 try 语句块对应的 finally 语句块都一定会被执行，因此通常使用 finally 语句块进行资源释放。

【例 11.6】　在 07 项目中创建 Example_07 类，使用 try…finally 语句块对程序进行异常处理和资源释放。代码如下：（**实例位置：光盘\TM\sl\11\7**）

```
private FileReader read=null;                                   //声明 FileReader 对象 read
public void readFileInfo(){                                     //定义方法
    try{
        try {
            //创建 FileReader 对象 read
            read=new FileReader("src/com/mingrisoft/Example_07.java");
            System.out.println("找到指定的文件，创建 IO 对象成功！");
        } catch (FileNotFoundException e) {
            e.printStackTrace();                                //输出栈踪迹
        }
    }finally{
        if (read!=null){
            try{
                read.close();                                   //关闭 FileReader 对象 read，释放资源
                System.out.println("关闭 IO 对象！");
            }catch(IOException ioe){
                ioe.printStackTrace();                          //输出栈踪迹
                System.out.println("关闭 IO 对象发生异常。");
            }
        }
    }
}
public static void main(String[] args) {
    Example_07 ex=new Example_07();                             //创建对象
    ex.readFileInfo();                                          //调用 readFileInfo()方法
}
```

运行结果如图 11.8 所示。

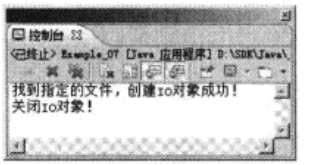


图 11.8　在控制台输出操作信息

说明 从输出结果可以看出，程序在 try 语句块中创建了 IO 对象并分配了内存资源，然后在 finally 语句块中关闭了 IO 对象并释放了内存资源。

11.5　抛 出 异 常

视频讲解：光盘\TM\lx\11\抛出异常.exe

对于程序中发生的异常，除了可以使用 try…catch 语句块处理之外，还可以使用 throws 声明或 throw 语句抛出异常。下面将分别进行讲解。

11.5.1　使用 throws 声明抛出异常

throws 通常用于方法声明，当方法中可能存在异常，却不想在方法中对异常进行处理时，就可以在声明方法时使用 throws 声明抛出的异常，然后在调用该方法的其他方法中对异常进行处理（如使用 try…catch 语句或使用 throws 声明抛出的异常）。

如果需要使用 throws 声明抛出多个异常，各异常之间要用逗号分隔。throws 声明抛出异常的语法格式如下：

```
数据类型　方法名(形参列表)　throws 异常类 1,异常类 2,……,异常类 n{
    方法体;
}
```

- ☑ 数据类型是基本数据类型或对象类型。
- ☑ 方法名是 Java 语言的合法标识符。
- ☑ throws 是抛出异常的关键字。
- ☑ 异常类是 Java 语言的异常类或自定义异常类。
- ☑ 方法体是该方法需要执行的语句。

【例 11.7】　使用 throws 抛出 Exception 异常。

```
public void showInfo() throws Exception{                    //抛出 Exception 异常
    FileInputStream in=new FileInputStream("C:/Record.txt");   //创建 IO 对象
}
```

说明 showInfo()方法使用 throws 抛出了 Exception 异常，这样在该方法中创建 IO 对象时就不会发生异常了，但是必须在调用该方法的其他方法中对 Exception 异常进行处理，否则调用该方法的其他方法会发生异常。

【例 11.8】 定义 methodName()方法，该方法调用了上面定义的 showInfo()方法，并对 showInfo()方法抛出的异常进行了处理。

```
void methodName() {
    try{
        showInfo();                          //使用 try…catch 语句块处理 showInfo()方法抛出的异常
    }catch(Exception ex){
        System.out.println(ex.getMessage() );
    }
}
```

说明 methodName()方法调用了 showInfo()方法，并对 showInfo()方法抛出的 Exception 异常进行了处理，否则该方法将出错。

【例 11.9】 在项目中创建 Example_08 类，在该类中创建一个使用 throws 抛出异常的 createFile()方法，然后创建一个 test()方法，在该方法中调用 createFile()方法，并进行异常处理。（**实例位置：光盘\TM\sl\11\8**）

```
public class Example_08 {
    private FileReader read=null;                //声明 FileReader 对象 read
    public void createFile() throws Exception{   //定义方法，使用 throws 抛出 Exception 异常
        //创建 FileReader 对象 read
        read=new FileReader("src/com/mingrisoft/Example_08.java");
        System.out.println("分配内存资源。");
    }
    public void test(){
        try{
            createFile();                    //调用 createFile()方法，使用 try…catch…finally 语句处理异常
        }catch(Exception ex){
            ex.printStackTrace();            //输出栈踪迹
            System.out.println("创建 IO 对象异常。");
        }finally{
            if (read!=null){
                try {
                    read.close();            //关闭 IO 流
                    System.out.println("释放内存资源。");
                } catch (IOException e) {
                    e.printStackTrace();     //输出栈踪迹
                    System.out.println("关闭 IO 对象异常。");
                }
            }
```

```
        }
    }
    public static void main(String[] args) {
        Example_08 ex=new Example_08();        //创建对象
        ex.test();                             //调用 test()方法
    }
}
```

运行结果如图 11.9 所示。

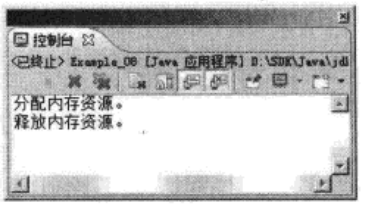


图 11.9　分配和释放内存资源提示

11.5.2　使用 throw 语句抛出异常

在通常情况下，程序发生错误时系统会自动抛出异常，而有时希望程序自行抛出异常，可以使用 throw 语句来实现。

throw 语句通常用在方法中，在程序中自行抛出异常，使用 throw 语句抛出的是异常类的实例，通常与 if 语句一起使用。

throw 语句的语法格式如下：

```
throw new Exception("对异常的说明");
```

☑　throw 是抛出异常的关键字。

☑　Exception 是异常类（通常使用自定义异常类）。

【例 11.10】　在项目中创建 Example_09 类，使用该类计算圆的面积。设定圆的半径不能小于 20，如果半径小于 20，则使用 throw 语句抛出异常，并给出提示信息。（**实例位置：光盘\TM\sl\11\9**）

```
public class Example_09 {
    final static double PI=3.14;                          //圆周率
    public void computeArea(double r) throws Exception{   //根据半径计算圆面积的方法
        if (r<=20.0){
            //使用 throw 语句抛出异常
throw new Exception("程序异常：\n 半径为 ："+r+"\n 半径不能小于 20。");
        }else{
            double circleArea=PI*r*r;                     //计算圆的面积
            System.out.println("半径是"+r+"的圆面积是："+circleArea);
        }
    }
    public static void main(String[] args) {
        Example_09 ex=new Example_09();                   //创建对象
        try {
            ex.computeArea(10);                           //调用方法
        } catch (Exception e) {
            System.out.println(e.getMessage());           //输出异常信息
        }
    }
}
```

运行结果如图 11.10 所示。

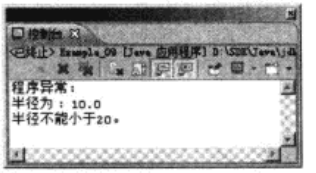


图 11.10　半径小于 20 抛出异常

说明 computeArea()方法根据圆的半径计算圆的面积，并且当圆的半径小于等于 0 时，使用 throw 语句抛出异常，由于该方法使用 throw 语句抛出了异常，所以必须在调用该方法时对其进行异常处理，本实例在主方法中使用 try…catch 语句对其进行了异常处理。

11.5.3　范例 3：方法中抛出异常

在项目开发中，通常是自顶向下进行的。在完成项目的整体设计后，需要对每个接口和类进行编写。如果一个类使用了其他类还没有实现的方法，则可以在实现其他类方法时让其抛出 UnsupportedOperationException，以便在以后进行修改完成。运行结果如图 11.11 所示。**（实例位置：光盘\TM\sl\11\10）**

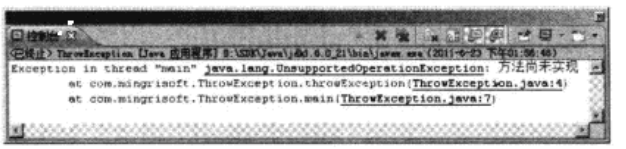


图 11.11　方法中抛出异常

（1）在 Eclipse 中创建项目 10，并在该项目中创建 com.mingrisoft 包。

（2）在 com.mingrisoft 包中编写 ThrowException 类，在该类中定义两个方法，一个是 throwException() 方法，用于抛出异常；另一个是 main()方法，用于进行测试。代码如下：

```
public class ThrowException {
    public static void throwException() {
        throw new UnsupportedOperationException("方法尚未实现");        //抛出异常
    }
    public static void main(String[] args) {
        ThrowException.throwException();                                //调用抛出异常的方法
    }
}
```

11.5.4　范例 4：方法上抛出异常

在方法的执行过程中，如果存在可能遇到引发问题的因素，则应该在定义方法时加以说明。例如

读取文件的方法可能遇到文件不存在的情况，此时需要在方法声明时抛出文件不存在异常。本范例将演示如何在方法上抛出异常。运行结果如图 11.12 所示。（**实例位置：光盘\TM\sl\11\11**）

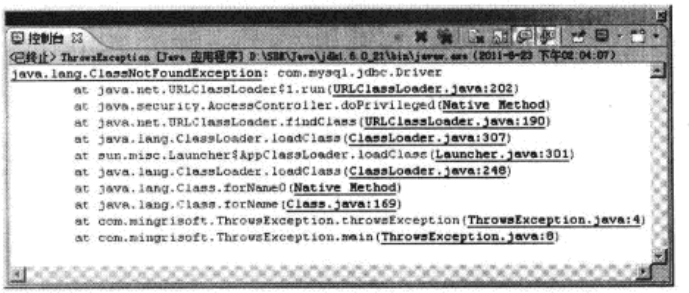


图 11.12　方法上抛出异常

（1）在 Eclipse 中创建项目 11，并在该项目中创建 com.mingrisoft 包。

（2）在 com.mingrisoft 包中编写 ThrowsException 类，并在该类中定义两个方法，一个是 throw Exception()方法，用于抛出异常；另一个是 main()方法，用于进行测试。代码如下：

```
public class ThrowsException {
    public static void throwsException() throws ClassNotFoundException {          //抛出异常
        Class.forName("com.mysql.jdbc.Driver");
    }

    public static void main(String[] args) {
        try {                                                                     //捕获异常
            ThrowsException.throwsException();                                    //调用抛出异常的方法
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
    }
}
```

11.6　自定义异常

视频讲解：光盘\TM\lx\11\自定义异常.exe

11.6.1　创建自定义异常类

创建自定义的异常类需要继承自 Exception 类，并提供含有一个 String 类型形参的构造方法，该形参就是异常的描述信息，可以通过 getMessage()方法获得。例如：

```
public class NewException extends Exception{
    public NewException(String s){
```

```
        super(s);
    }
}
```

说明 上面代码创建了一个自定义异常类 NewException，也就是说 NewException 是自定义异常类的名称，该类继承自 Exception 类，该类构造方法的形参 s 是需要传递的异常描述信息，该信息可以通过异常类的 getMessage()方法获得。

11.6.2 使用自定义异常类

创建完自定义异常类后，就可以在程序中使用了。使用自定义异常类可以通过 throw 语句抛出异常，接下来通过实例来说明自定义异常类的使用。

【例 11.11】 在项目中创建 Example_12 类，在该类中计算圆的面积，设定圆的半径不能小于 20，如果半径小于 20，则调用自定义异常类 NewException，处理程序所发生的异常。（**实例位置：光盘\TM\sl\11\12**）

自定义异常类 NewException 的代码如下：

```
public class NewException extends Exception {                    //创建自定义异常类
    public NewException(Double r) {                              //有一个 Double 类型形参的构造方法
        System.out.println("发生异常： 圆的半径不能小于 20");
        System.out.println("圆的半径为:"+r);
    }
}
```

Example_12 类的代码如下：

```
public static void showArea(double r) throws NewException {      //创建求圆面积的方法
    if (r <= 20) {
        throw new NewException(r);                               //抛出异常
    }
    double area = 3.14 * r * r;                                  //计算圆的面积
    System.out.println("圆的面积是：" + area);                    //输出圆的面积
}
public static void main(String[] args) {
    try {
        showArea(10);                                            //调用 showArea()方法，传递半径 10
    } catch (NewException ex) {
        System.out.println(ex);                                  //输出异常信息
    }
}
```

运行结果如图 11.13 所示。

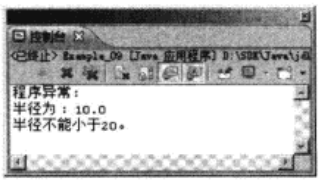


图 11.13　半径小于 20 抛出异常

11.7　异常的使用原则

视频讲解：光盘\TM\lx\11\异常的使用原则.exe

在程序中使用异常，可以捕获程序中的错误，但是异常的使用也要遵循一定的规则。下面是异常类的几项使用原则：

- ☑ 不要过多地使用异常，这样会增加系统的负担。
- ☑ 在方法中使用 try…catch 语句块捕获异常时，要对异常作出处理。
- ☑ try…catch 语句块的范围不要太大，这样不利于对异常的分析。
- ☑ 一个方法被覆盖时，覆盖它的方法必须抛出相同的异常或子异常。

11.8　经 典 范 例

前面的内容介绍了对异常的处理，下面给出几个在以后的实际开发中经常遇到的异常的例子。

11.8.1　经典范例 1：捕获单个异常

视频讲解：光盘\TM\lx\11\捕获单个异常.exe

当遇到异常时，除了可以将异常抛出，还可以将其捕获。抛出异常虽然简单，但是有时却不得不使用捕获来处理异常。如果程序遇到异常而没有捕获，则程序会直接退出。这在大多数情况下是不能被接受的，至少需要保存程序当前状态才能退出。本范例将演示如何捕获单个异常。运行结果如图 11.14 所示。（**实例位置：光盘\TM\sl\11\13**）

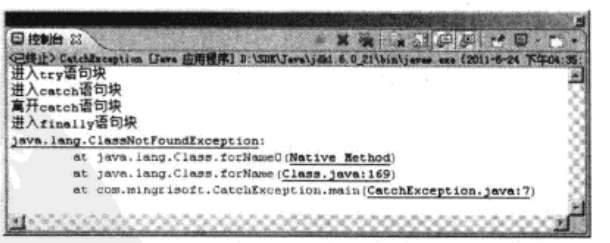


图 11.14　捕获单个异常

（1）在 Eclipse 中创建项目 14，并在该项目中创建 com.mingrisoft 包。

（2）在 com.mingrisoft 包中编写 CatchException 类，在该类的 main()方法中应用 try…catch…finally 语句捕获单个异常。代码如下：

```
public class CatchException {
    public static void main(String[] args) {
        try {                                                    //定义 try 语句块
            System.out.println("进入 try 语句块");
            @SuppressWarnings("unused")
            Class<?> clazz = Class.forName("");                  //得到一个空的 Class 对象
            System.out.println("离开 try 语句块");
        } catch (ClassNotFoundException e) {                     //定义 catch 语句块
            System.out.println("进入 catch 语句块");
            e.printStackTrace();
            System.out.println("离开 catch 语句块");
        } finally {                                              //定义 finally 语句块
            System.out.println("进入 finally 语句块");
        }
    }
}
```

说明　Java 中捕获异常是通过 try…catch…finally 语句来完成的。其中 try 语句块是必需的，catch 和 finally 语句块可以选择一个或者两个。try 语句块用来放置可能出现问题的语句，catch 语句块用来放置异常发生后执行的代码。finally 语句块用来放置无论是否发生异常都需要执行的代码。

11.8.2　经典范例 2：数据库操作异常

视频讲解：光盘\TM\lx\11\数据库操作异常.exe

数据库操作异常即 SQLException，通常发生在出现数据库访问错误时。本范例将演示出现数据库操作异常的情况。运行结果如图 11.15 所示。（**实例位置：光盘\TM\sl\11\14**）

图 11.15　数据库操作异常

（1）在 Eclipse 中创建项目 15，并在该项目中创建 com.mingrisoft 包。

（2）在 com.mingrisoft 包中编写 ExceptionTest 类，在该类的 main()方法中，编写数据库连接的代码，并且捕获可能抛出的异常。代码如下：

```
public class ExceptionTest {
    public static void main(String[] args) {
        String URL = "jdbc:mysql://localhost:3306/db_database";        //MySQL 数据库的 URL
        String DRIVER = "com.mysql.jdbc.Driver";                       //MySQL 数据库的驱动
        String USERNAME = "mr";                                        //数据库的用户名
        Connection connection = null;
        try {
            Class.forName(DRIVER);                                     //加载驱动
            //建立连接
            connection = DriverManager.getConnection(URL, USERNAME, "");
        } catch (SQLException e) {                                     //捕获 SQLException
            e.printStackTrace();
        } catch (ClassNotFoundException e) {                           //捕获 ClassNotFoundException
            e.printStackTrace();
        } finally {
            try {
                connection.close();                                    //释放资源
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
}
```

11.9 本章小结

本章向读者介绍的是 Java 中的异常处理机制。通过本章的学习，读者应了解异常的概念、几种常见的异常类，掌握异常处理技术，以及如何创建、激活用户自定义的异常处理器。Java 中的异常处理是通过 try…catch 语句来实现的，也可以使用 throws 语句向上抛出。建议读者不要将异常抛出，应该编写异常处理语句。对于异常处理的使用原则，读者也应该理解。

11.10 实战练习

1．编写一个异常类 MyException，再编写一个 Student 类，该类有一个产生异常的 speak(int m)方法。要求参数 m 的值大于 1000 时，方法抛出一个 MyException 对象。最后编写主类，在主方法中创建 Student 对象，让该对象调用 speak()方法。（**答案位置：光盘\TM\sl\11\15**）

2．创建 Number 类，通过类中的 count()方法可得到任意两个数相乘的结果，并在调用该方法的主方法中使用 try…catch 语句捕捉可能发生的异常。（**答案位置：光盘\TM\sl\11\16**）

3．创建 Computer 类，该类中有一个计算两个数的最大公约数的方法，如果向该方法传递负整数，该方法就会抛出自定义异常。（**答案位置：光盘\TM\sl\11\17**）

第12章

输入/输出

（视频讲解：30分钟）

在变量、数组和对象中存储数据是暂时的，程序结束后它们就会丢失。为了能够永久地保存程序创建的数据，需要将其保存在磁盘文件中，这样以后就可以在其他程序中使用它们。Java 的 I/O 技术可以将数据保存到文本文件、二进制文件甚至是 ZIP 压缩文件中，以达到永久性保存数据的要求。掌握 I/O 处理技术能够提高对数据的处理能力。本章将向读者介绍 Java 的 I/O 技术。

通过阅读本章，您可以：

- 了解 Java 中流的概念
- 了解 Java 中输入/输出流的分类
- 掌握文件输入/输出流的使用方法
- 掌握带缓存的输入/输出流的使用方法
- 理解 ZIP 压缩输入/输出流的应用

PDG

12.1 流 概 述

视频讲解：光盘\TM\lx\12\流概述.exe

流是一组有序的数据序列，根据操作的类型，可分为输入流和输出流两种。I/O（Input/Output）流提供了一条通道程序，可以使用这条通道把源中的字节序列送到目的地。虽然 I/O 流经常与磁盘文件存取有关，但是程序的源和目的地也可以是键盘、鼠标、内存或显示器窗口等。

Java 由数据流处理输入/输出模式，程序从指向源的输入流中读取源中的数据，如图 12.1 所示。源可以是文件、网络、压缩包或者其他数据源。

输出流的指向是数据要到达的目的地，程序通过向输出流中写入数据把信息传递到目的地，如图 12.2 所示。输出流的目标可以是文件、网络、压缩包、控制台和其他数据输出目标。

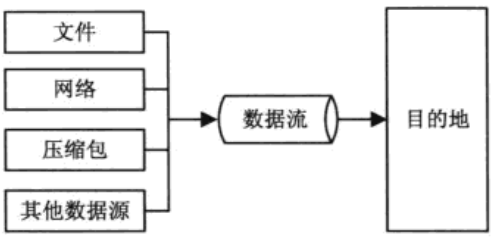


图 12.1 输入模式

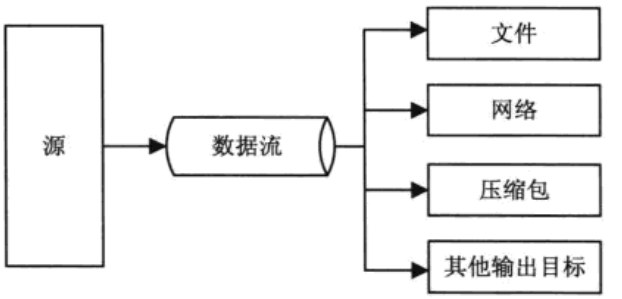


图 12.2 输出模式

12.2 输入/输出流

视频讲解：光盘\TM\lx\12\输入/输出流.exe

Java 语言定义了许多类专门负责各种方式的输入/输出，这些类都被放在 java.io 包中。其中，所有输入流类都是抽象类 InputStream（字节输入流）或抽象类 Reader（字符输入流）的子类；而所有输出流都是抽象类 OutputStream（字节输出流）或抽象类 Writer（字符输出流）的子类。

12.2.1 输入流

InputStream 类是字节输入流的抽象类，是所有字节输入流的父类。InputStream 类的具体层次结构如图 12.3 所示。

该类中所有方法遇到错误时都会引发 IOException 异常。下面是对该类中的一些方法的简要说明，如表 12.1 所示。

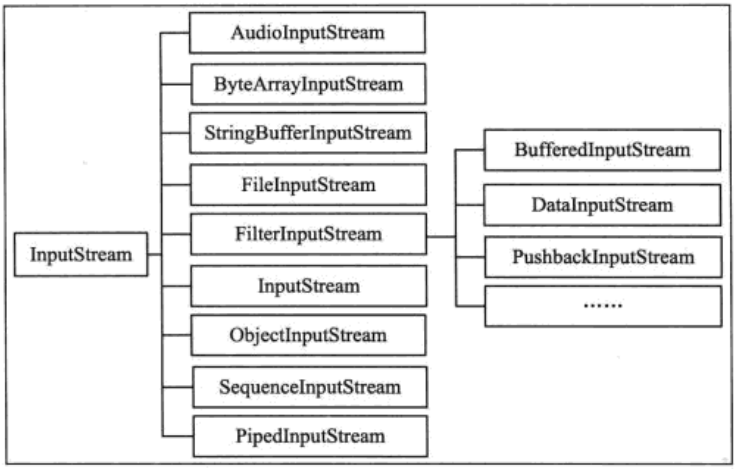


图 12.3　InputStream 类的层次结构

表 12.1　InputStream 类常用的方法

方　法	说　明
read()	从输入流中读取数据的下一个字节。返回 0～255 范围内的 int 字节值。如果因为已经到达流末尾而没有可用的字节，则返回值-1
read(byte[] b)	从输入流中读入一定长度的字节，并以整数的形式返回字节数
mark(int readlimit)	在输入流的当前位置放置一个标记，readlimit 参数告知此输入流在标记位置失效之前允许读取的字节数
reset()	将输入指针返回到当前所做的标记处
skip(long n)	跳过输入流上的 n 个字节并返回实际跳过的字节数
markSupported()	如果当前流支持 mark()/reset()操作就返回 true
close()	关闭此输入流并释放与该流关联的所有系统资源

注意　并不是所有的 InputStream 类的子类都支持 InputStream 中定义的所有方法，如 skip()、mark()、reset()等，这些方法只对某些子类有用。

Java 中的字符是 Unicode 编码，是双字节的。InputStream 是用来处理字节的，在处理字符文本时不是很方便。Java 为字符文本的输入提供了专门一套单独的类 Reader，但 Reader 类并不是 InputStream 类的替换者，只是在处理字符串时简化了编程。Reader 类是字符输入流的抽象类，所有字符输入流的实现都是它的子类。Reader 类的具体层次结构如图 12.4 所示。

Reader 类中方法与 InputStream 中方法类似，读者在需要时可查看 JDK 文档。

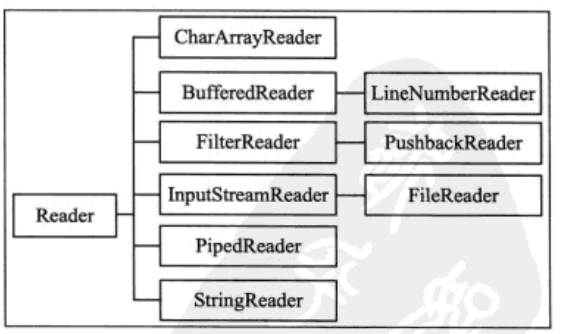


图 12.4　Reader 类的层次结构

12.2.2 输出流

OutputStream 类是字节输入流的抽象类，此抽象类表示输出字节流的所有类的超类。OutputStream 类的具体层次如图 12.5 所示。

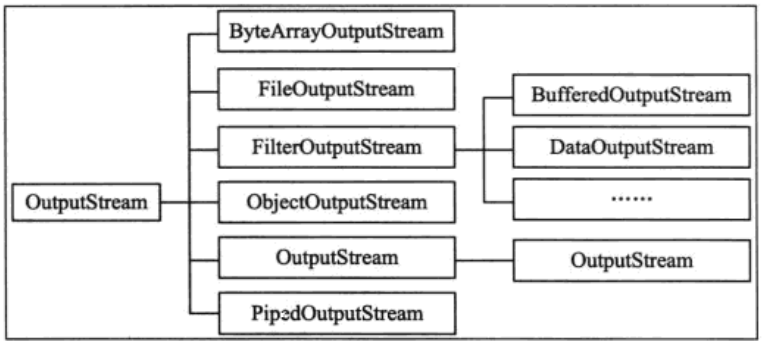


图 12.5 OutputStream 类的层次结构

OutputStream 类中的所有方法均返回 void，在遇到错误时会引发 IOException 异常。下面对 OutputStream 类中的方法作一简单的介绍，如表 12.2 所示。

表 12.2 OutputStream 类常用的方法

方　法	说　明
write(int b)	将指定的字节写入此输出流
write(byte[] b)	将 b.length 个字节从指定的 byte 数组写入此输出流
write(byte[] b,int off,int len)	将指定 byte 数组中从偏移量 off 开始的 len 个字节写入此输出流
flush()	彻底完成输出并清空缓存区
close()	关闭输出流

Writer 类是字符输出流的抽象类，所有字符输出类的实现都是它的子类。Writer 类的层次结构如图 12.6 所示。

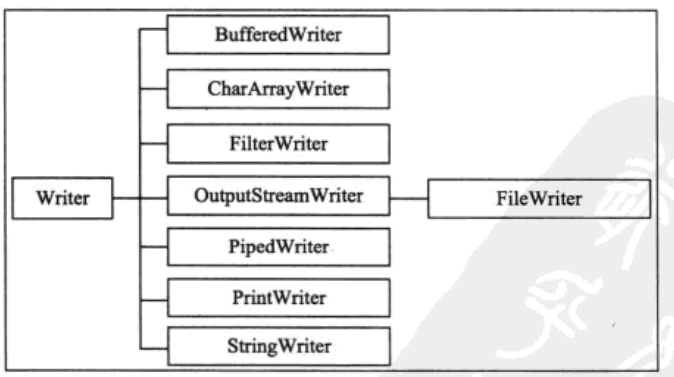


图 12.6 Writer 类的层次结构

12.2.3　范例 1：显示指定类型的文件

在 Windows 系统中，通过文件的扩展名来区别不同类型的文件。本范例将根据用户输入的扩展名，列出指定文件夹内该类型文件的文件名、文件大小和修改时间。运行结果如图 12.7 所示。（**实例位置：光盘\TM\sl\12\1**）

图 12.7　查看文件夹中指定类型文件的信息

在项目中创建 FilesList 类，该类继承了 JFrame 并增加了标签、文本框等控件。单击“选择文件夹”按钮，将判断用户输入的扩展名是否为空。如果不为空则根据该字符串过滤用户选择的文件夹中的文件，将符合条件的文件信息增加到表格模型中，最后更新表格。代码如下：

```
protected void do_button_actionPerformed(ActionEvent e) {
    final String fileType = textField.getText();                          //获得用户输入的文件类型
    if (fileType.isEmpty()) {                                             //提示用户输入的文件类型
        JOptionPane.showMessageDialog(this, "请输入文件类型！", "", JOptionPane.WARNING_MESSAGE);
        return;
    }
    JFileChooser chooser = new JFileChooser();                            //创建文件选择器
    chooser.setFileSelectionMode(JFileChooser.DIRECTORIES_ONLY);          //设置仅能选择文件夹
    chooser.setMultiSelectionEnabled(false);                              //禁止选择多个文件夹
    int result = chooser.showOpenDialog(this);                            //打开文件选择器
    if (result == JFileChooser.APPROVE_OPTION) {
        File[] listFiles = chooser.getSelectedFile().listFiles(new java.io.FileFilter() {
            @Override
            public boolean accept(File pathname) {
                if (pathname.getName().endsWith(fileType)) {
                    return true;
                } else {
                    return false;
                }
            }
        });                                                               //获得符合条件的文件
        DefaultTableModel model = (DefaultTableModel) table.getModel();   //获得默认表格模型
        //指定日期格式
        SimpleDateFormat format = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
        for (File file : listFiles) {
            String name = file.getName();                                 //获得文件名
            long size = file.length();                                    //获得文件大小
            String modifyDate = format.format(new Date(file.lastModified()));  //获得文件修改日期
            model.addRow(new String[] { name, "" + size, modifyDate });   //向表格中增加数据
        }
```

```
        table.setModel(model);                                    //更新表格模型
    }
}
```

12.2.4 范例 2：查找替换文本文件内容

目前主流的文本编辑工具都提供了替换功能，但是需要先打开文本文件。本范例可以直接替换用户选择的文本文件内容。运行结果如图 12.8 所示。（**实例位置：光盘\TM\sl\12\2**）

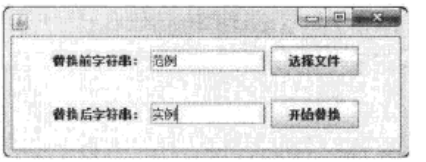


图 12.8 将文件中的“范例”替换成“实例”

在项目中创建 ReplaceTool 类，该类继承了 JFrame 并增加了标签、文本框和按钮控件。单击“开始替换”按钮，可以完成文本替换功能。其事件监听器的代码如下：

```
protected void do_replaceButton_actionPerformed(ActionEvent e) {
    String before = beforeTextField.getText();                    //获得替换前字符串
    //省略校验代码
    String after = afterTextField.getText();                      //获得替换后字符串
    //省略校验代码
    FileReader reader = null;                                     //创建文件读流
    FileWriter writer = null;                                     //创建文件写流
    StringBuilder sb = new StringBuilder();                       //使用 StringBuilder 对象保存文件内容
    int flag = 0;                                                 //声明文件读入标示
    char[] temp = new char[1024];                                 //使用字符数组读入文件
    try {
        reader = new FileReader(textFile);                        //使用选择的文件创建读流
        while ((flag = reader.read(temp)) != -1) {
            sb.append(temp);                                      //读入文件中的内容
        }
        String content = sb.toString().replace(before, after);    //替换字符串
        writer = new FileWriter(textFile);                        //创建文件写流
        writer.write(content);                                    //将替换后的字符串写入到文件
    } catch (FileNotFoundException e1) {
        e1.printStackTrace();
    } catch (IOException e1) {
        e1.printStackTrace();
    } finally {
    //省略释放资源代码
    }
    JOptionPane.showMessageDialog(this, "字符串替换成功！");
    return;
}
```

12.3　File 类

视频讲解：光盘\TM\lx\12\File 类.exe

File 类是 io 包中唯一代表磁盘文件本身的对象。File 类定义了一些与平台无关的方法来操作文件，可以通过调用 File 类中的方法，实现创建、删除、重命名文件等。File 类的对象主要用来获取文件本身的一些信息，如文件所在的目录、文件的长度、文件读写权限等。数据流可以将数据写入到文件中，而文件也是数据流最常用的数据媒体。

12.3.1　文件的创建与删除

可以使用 File 类创建一个文件对象，通常使用以下 3 种构造方法来创建文件对象。

（1）File(String pathname)

该构造方法通过将给定路径名字符串转换为抽象路径名来创建一个新 File 实例。语法格式如下：

```
new File(String pathname)
```

pathname：是指路径名称（包含文件名）。

```
File file = new File("d:/1.txt");
```

（2）File(String parent , String child)

该构造方法根据定义的父路径和子路径字符串（包含文件名）创建一个新的 File 对象。语法格式如下：

```
new File(String parent , String child)
```

- ☑ parent：父路径字符串。
- ☑ child：子路径字符串。

（3）File(File f , String child)

该构造方法根据 parent 抽象路径名和 child 路径名字符串创建一个新 File 实例。语法格式如下：

```
new File(File f , String child)
```

- ☑ f：父路径对象。
- ☑ child：子路径字符串。

注意　对于 Microsoft Windows 平台，包含盘符的路径名前缀由驱动器号和一个“:”组成。如果路名是绝对路径名，还可能后跟“\\”。

使用 File 类创建一个文件对象，语法格式如下：

```
File file = new File("D:/myword","word.txt");
```

如果 D:/myword 目录中没有名称为 word 的文件，File 类对象可通过调用 createNewFile()方法创建一个名称为 word.txt 的文件；如果 word.txt 文件存在，可以通过文件对象的 delete()方法将其删除，如例 12.1 所示。

【例 12.1】 在项目中创建 FileTest 类，在主方法中判断 D 盘的 mywork 文件夹是否存在 work.txt 文件，如果该文件存在将其删除，不存在则创建该文件。（**实例位置：光盘\TM\sl\12\3**）

```
import java.io.File;                                    //导入 java.io.File 类
public class FileTest {                                 //创建 FileTest 类
    public static void main(String[] args) {            //主方法
        File file = new File("D:/myword", "test.txt");  //创建文件对象
        if (file.exists()) {                            //如果该文件存在
            file.delete();                              //将文件删除
            System.out.println("文件已删除");            //输出的提示信息
        } else {                                        //如果文件不存在
            try {                                       //try 语句块捕捉可能出现的异常
                file.createNewFile();                   //创建该文件
                System.out.println("文件已创建");        //输出的提示信息
            } catch (Exception e) {                     //catch 处理该异常
                e.printStackTrace();                    //输出异常信息
            }
        }
    }
}
```

运行结果如图 12.9 所示。

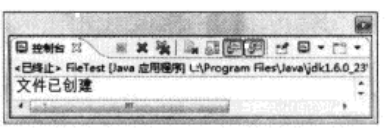


图 12.9 提示文件创建成功

注意

由于 D:/mywork 目录下并没有 work 文件，因此运行程序会创建 work.txt 文件。

12.3.2 获取文件信息

File 类提供了很多方法用于获取文件本身的一些信息，File 类的常用方法如表 12.3 所示。

表 12.3　File 类的常用方法

方　　法	说　　明
getName()	获取文件的名称
canRead()	判断文件是否是可读的
canWrite()	判断文件是否可被写入
exits()	判断文件是否存在
length()	获取文件的长度（以字节为单位）
getAbsolutePath()	获取文件的绝对路径
getParent()	获取文件的父路径
isFile()	判断文件是否存在
isDirectory()	判断文件是否是一个目录
isHidden()	判断文件是否是隐藏文件
lastModified()	获取文件最后修改时间

【例 12.2】　获取 D 盘中 mywork 文件夹下的 work.txt 文件的文件名、文件长度，并判断该文件是否是隐藏文件。（**实例位置：光盘\TM\sl\12\4**）

```
import java.io.File;                                          //导入 java.io.File 类
public class FileTest {                                       //创建类
    public static void main(String[] args) {
        File file = new File("D:/mywork", "work.txt");        //创建文件对象
        if(file.exists()){                                    //如果文件存在
        String name = file.getName();                         //获取文件名称
        long length = file.length();                          //获取文件长度
        boolean hidden = file.isHidden();                     //判断文件是否是隐藏文件
        System.out.println("文件名称："+name);                 //输出信息
        System.out.println("文件长度是："+length);
        System.out.println("该文件是隐藏文件吗？"+hidden);
        }
        else{                                                 //如果文件不存在
            System.out.println("该文件不存在");                //输出信息
        }
    }
}
```

运行结果如图 12.10 所示。

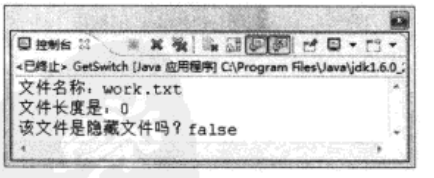


图 12.10　显示文件信息

12.3.3　范例 3：文件批量重命名

在 Windows 操作系统中，支持文件重命名但是不支持批量重命名。对于多个文件，如果要命名为希望的格式显得非常麻烦。本范例将开发一个文件批量重命名工具。运行结果如图 12.11 和图 12.12 所示。（**实例位置：光盘**

TM\sl\12\5）

图 12.11　替换前的预览效果

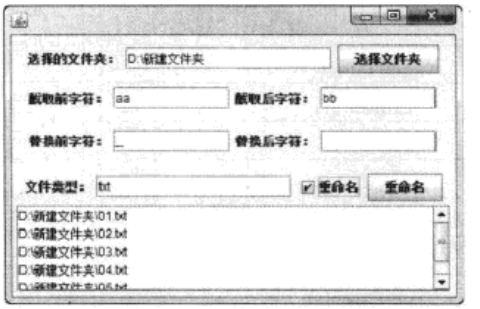


图 12.12　替换后的实际效果

在项目中创建 RenameTool 类，该类继承了 JFrame 并增加了标签、文本框、按钮和复选框等控件。选中复选框控件，单击“重命名”按钮，可以完成文件的重命名操作。事件监听器的代码如下：

```
protected void do_renameButton_actionPerformed(ActionEvent e) {
    //省略获得用户输入以及校验是否选择了非空文件夹代码
    textArea.setText("");                                          //清空文本域中数据
    for (File selectFile : selectFiles) {
        String fileName = selectFile.getName();                    //获得文件名
        //省略处理文件名字符串代码
        fileName = selectFile.getParent() + File.separator + fileName;   //获得修改后的文件名
        if (choiceCheckBox.isSelected()) {
            textArea.append(fileName + "\n\r");                    //在文本域中显示重命名的结果
            selectFile.renameTo(new File(fileName));               //重命名文件
        } else {
            textArea.append(fileName + "\n\r");                    //重命名文件
        }
    }
}
```

12.3.4　范例 4：快速批量移动文件

在 Windows 操作系统中，移动大量文件花费的时间非常可观。使用 Java 语言提供的文件重命名方式也可以实现文件移动功能，而且能够节约大量的时间。本范例将完成这个功能，其运行结果如图 12.13 所示。**（实例位置：光盘\TM\sl\12\6）**

在项目中创建 FileMoveTool 类，该类继承了 JFrame 并增加了标签、文本框、按钮和复选框等控件。在选择完源文件、目标文件夹后，单击“移动”按钮，可以完成文件的移动功能。事件监听器代码如下：

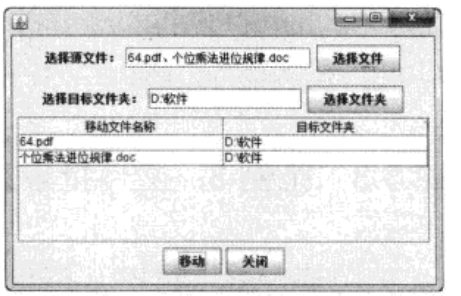


图 12.13　移动文件后的效果

```
protected void do_moveButton_actionPerformed(ActionEvent e) {
    //省略选择文件和文件夹的校验代码
    DefaultTableModel model = (DefaultTableModel) table.getModel();                    //获得表格模型
for (File selectFile : selectFiles) {
        //获得新文件名
        String fileName = targetDirectory.getAbsolutePath() + File.separator + selectFile.getName();
        selectFile.renameTo(new File(fileName));                                        //移动文件
        //向表格模型中增加数据
        model.addRow(new String[] { selectFile.getName(), targetDirectory.getAbsolutePath() });
    }
    table.setModel(model);                                                              //设置表格模型
}
```

12.4　文件输入/输出流

视频讲解：光盘\TM\lx\12\文件输入/输出流.exe

程序运行期间，大部分数据都是在内存中进行操作，当程序结束或关闭时，这些数据将消失。如果需要将数据永久保存，可使用文件输入/输出流与指定的文件建立连接，将需要的数据永久保存到文件中。本节将向读者介绍文件输入/输出流。

12.4.1　FileInputStream 与 FileOutputStream 类

FileInputStream 类与 FileOutputStream 类都是用来操作磁盘文件的。如果用户的文件读取需求比较简单，则可以使用 FileInputStream 类。该类继承自 InputStream 类。FileOutputStream 类与 FileInputStream 类对应，提供了基本的文件写入能力。FileOutputStream 类是 OutoputStream 类的子类。

FileInputStream 类常用的构造方法如下：

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

第一个构造方法使用给定的文件名 name 创建一个 FileInputStream 对象，第二个构造方法使用 File 对象创建 FileInputStream 对象。第一个构造方法比较简单，但第二个构造方法允许在把文件连接输入流之前对文件作进一步分析。

FileOutputStream 类有与 FileInputStream 类相同参数的构造方法，创建一个 FileOutputStream 对象时，可以指定不存在的文件名，但此文件不能是一个已被其他程序打开的文件。下面的实例就是使用 FileInputStream 与 FileOutputStream 类实现文件的读取与写入功能。

【例 12.3】　使用 FileOutputStream 类向文件 work 写入信息，然后通过 FileInputStream 类将 work 文件中的数据读取到控制台上。（**实例位置：光盘\TM\sl\12\7**）

```
import java.io.*;                                                    //导入 java.io 包
public class FileTest {                                              //创建类
```

```
    public static void main(String[] args) {                          //主方法
        File file = new File("D:/mywork", "work.txt");                //创建文件对象
        try {                                                         //捕捉异常
            FileOutputStream out = new FileOutputStream(file);        //创建 FileOutputStream 对象
            byte buy[] = "明日科技  Java 部".getBytes();               //创建 byte 型数组
            out.write(buy);                                           //将数组中信息写入到文件中
            out.close();                                              //将流关闭
        } catch (Exception e) {                                       //catch 语句处理异常信息
            e.printStackTrace();                                      //输出异常信息
        }
        try{
            FileInputStream in = new FileInputStream(file);           //创建 FileInputStream 类对象
            byte byt[] = new byte[1024];                              //创建 byte 数组
            int len = in.read(byt);                                   //从文件中读取信息
            System.out.println("文件中的信息是："+new String(byt,0,len));//将文件中信息输出
            in.close();                                               //关闭流
        }catch (Exception e) {
            e.printStackTrace();                                      //输出异常信息
        }
    }
}
```

运行结果如图 12.14 所示。

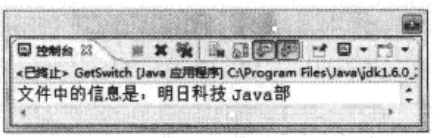


图 12.14　读取文件

注意　虽然 Java 语言在程序结束时自动关闭所有打开的流，但是当使用完流后，显式地关闭任何打开的流仍是一个好习惯。一个被打开的流有可能会用尽系统资源，这取决于平台和实现。如果没有将打开的流关闭，当另一个程序试图打开另一个流时，这些资源可能会得不到。

12.4.2　FileReader 类和 FileWriter 类

使用 FileOutputStream 类向文件中写入数据与使用 FileInputStream 类从文件中将内容读出来，存在一点不足，即这两个类都只提供了对字节或字节数组的读取方法。由于汉字在文件中占用两个字节，如果使用字节流，读取不好可能会出现乱码现象。此时采用字符流 FileReader 或 FileWriter 类即可避免这种现象。

FileReader、FileWriter 字符流对应了 FileInputStream、FileOutputStream 类。FileReader 流顺序地读取文件，只要不关闭流，每次调用 read()方法就顺序地读取源中其余的内容，直到源的末尾或流被关闭。

下面通过一个应用程序介绍 FileReader 与 FileWriter 类的用法。

【例 12.4】　本实例创建 Swing 窗体，单击窗体中的“写入文本”按钮实现将文本框中的数据写入到磁盘文件中；单击“读取文本”按钮，系统将磁盘文件中的信息显示在文本框中。（**实例位置：光盘\TM\sl\12\8**）

```
import javax.swing.*;                                                    //导入 javax.swing 包
/**********************省略了导入相应的类包*********************************/
public class Ftest extends JFrame{                                       //创建类，继承 JFrame 类
    private static final long serialVersionUID = 1L;
    private JPanel jContentPane = null;                                  //创建面板对象
    private JTextArea jTextArea = null;                                  //创建文本域对象
    private JPanel controlPanel = null;                                  //创建面板对象
    private JButton openButton = null;                                   //创建按钮对象
    private JButton closeButton = null;                                  //创建按钮对象
/**********************省略了对窗体进行布局代码*********************************/
    private JButton getOpenButton() {
        if (openButton == null) {
            openButton = new JButton();
            openButton.setText("写入文本");                              //修改按钮的提示信息
            openButton.addActionListener(new java.awt.event.ActionListener() {
                public void actionPerformed(java.awt.event.ActionEvent e) {   //按钮的单击事件
                    File file = new File("D:/work.txt");                 //创建文件对象
                    try {
                        FileWriter out = new FileWriter(file);           //创建 FileWriter 对象
                        String s = jTextArea.getText();                  //获取文本域中文本
                        out.write(s);                                    //将信息写入磁盘文件
                        out.close();                                     //将流关闭
                        } catch (Exception e1) {
                        e1.printStackTrace();
                    }
                }
            });
        }
        return openButton;
    }
    private JButton getCloseButton() {
        if (closeButton == null) {
            closeButton = new JButton();
            closeButton.setText("读取文本");                             //修改按钮的提示信息
            closeButton.addActionListener(new java.awt.event.ActionListener() {
                public void actionPerformed(java.awt.event.ActionEvent e) {   //按钮的单击事件
                    File file = new File("D:/work.txt");                 //创建文件对象
                    try {
                        FileReader in = new FileReader(file);            //创建 FileReader 对象
                    char byt[] = new char[1024];                         //创建 char 型数组
                    int len = in.read(byt);                              //将字节读入数组
                    jTextArea.setText(new String (byt,0,len));           //设置文本域的显示信息
                    in.close();                                          //关闭流
```

```
                    } catch (Exception e1) {
                        e1.printStackTrace();
                    }
                }
            });
        }
        return closeButton;
    }
    public Ftest() {
        super();
        initialize();
    }

    private void initialize() {
        this.setSize(300, 200);
        this.setContentPane(getJContentPane());
        this.setTitle("JFrame");
    }
    private JPanel getJContentPane() {
        if (jContentPane == null) {
            jContentPane = new JPanel();
            jContentPane.setLayout(new BorderLayout());
            jContentPane.add(getJTextArea(), BorderLayout.CENTER);
            jContentPane.add(getControlPanel(), BorderLayout.SOUTH);
        }
        return jContentPane;
    }
    public static void main(String[] args) {                    //主方法
        Ftest thisClass = new Ftest();                          //创建本类对象
        thisClass.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        thisClass.setVisible(true);                             //设置该窗体为显示状态
    }
}
```

运行结果如图 12.15 所示。

该程序设计了一个文本域和两个按钮，当单击“读取文本”按钮时，会根据路径找到磁盘文件，把文件中的数据信息显示到文本域中；当单击“写入文本”按钮时，会将用户在文本域中的输入信息写入到指定的磁盘文件中。

图 12.15　使用 FileReader 类和 FileWriter 类操作文件

12.4.3　范例 5：窗体动态加载磁盘文件

使用图形化操作系统时，打开一个文件夹会显示当前文件夹中的文件及子文件夹。本范例将完成

类似的功能。用户在选择一个文件夹后，可以在表格中动态加载该文件夹的内容，其运行结果如图 12.16 所示。（**实例位置：光盘\TM\sl\12\9**）

图 12.16　显示文件夹中的文件

在项目中创建 FileListFrame 类，该类继承了 JFrame 并增加了标签、文本框、按钮和表格等控件。在选择完文件夹后，将在表格中动态加载文件夹中的文件。事件监听器代码如下：

```
protected void do_chooseButton_actionPerformed(ActionEvent e) {
    JFileChooser chooser = new JFileChooser();                          //创建文件选择器
    chooser.setFileSelectionMode(JFileChooser.DIRECTORIES_ONLY);        //限制仅能选择文件夹
    chooser.setMultiSelectionEnabled(false);                            //禁止多重选择
    int result = chooser.showOpenDialog(this);                          //打开文件选择器
    if (result == JFileChooser.APPROVE_OPTION) {
        File selectDirectory = chooser.getSelectedFile();               //获得选择的文件夹
        chooseTextField.setText(selectDirectory.getAbsolutePath());     //显示文件夹位置
        final File[] files = selectDirectory.listFiles();               //获得文件夹中文件及子目录
        //获得表格模型
        final DefaultTableModel model = (DefaultTableModel) table.getModel();
        //创建时间格式化方式
        final SimpleDateFormat format = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
        new Thread() {
            public void run() {
                for (File file : files) {
                    if (file.isFile()) {                        //如果是文件则向表格模型中增加数据
                        model.addRow(new  Object[]  {  file.getName(),  file.length(),  format.format(new
Date(file.lastModified())) });
                        try {
                            Thread.sleep(1000);                 //当前线程休眠 1 秒钟实现动态加载
                        } catch (InterruptedException e) {
                            e.printStackTrace();
                        }
                    }
                }
            };
        }.start();
    }
}
```

12.4.4 范例 6：删除文件夹中所有文件

删除文件是操作系统提供的基本功能之一，本范例将使用 Java 语言实现这个功能，同时记录删除的文件路径。运行结果如图 12.17 所示。（**实例位置：光盘\TM\sl\12\10**）

图 12.17 删除指定文件夹中的文件

在项目中创建 FileDeletionFrame 类，该类继承了 JFrame 并增加了标签、文本框、按钮等控件。在选择完文件夹后，单击“删除”按钮，将删除该文件夹。这里使用 deleteFile()方法完成删除任务，代码如下：

```
private void deleteFile(File root, JTextArea textArea) {
    if (root.isFile()) {                                        //如果是文件则直接删除
        root.delete();
        textArea.append(root.getAbsolutePath() + "\n\r");       //将删除的文件记录到文本区中
    } else {
        File[] files = root.listFiles();                        //获得文件夹中的文件及其子文件夹
        for (File file : files) {
            if (file.isFile()) {                                //如果是文件则直接删除
                file.delete();
                textArea.append(file.getAbsolutePath() + "\n\r");   //将删除的文件记录到文本区中
            } else {                                            //如果是文件夹则进行迭代
                deleteFile(file, textArea);
            }
        }
        root.delete();                                          //删除空文件夹
    }
}
```

12.5 带缓存的输入/输出流

视频讲解：光盘\TM\lx\12\带缓存的输入/输出流.exe

缓存可以说是 I/O 的一种性能优化。缓存流为 I/O 流增加了内存缓存区。有了缓存区，使得在流上执行 skip()、mark()和 reset()方法都成为可能。

12.5.1 BufferedInputStream 类与 BufferedOutputStream 类

BufferedInputStream 类可以对任意的 InputStream 类进行带缓存区的包装以达到性能的优化。

BufferedInputStream 类有两个构造函数：

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

第一种形式的构造函数创建了一个带有 32 个字节的缓存流；第二种形式的构造函数按指定的大小来创建缓存区。一个最优的缓存区的大小，取决于它所在的操作系统、可用的内存空间以及机器配置。从构造函数可以看出，BufferedInputStream 对象位于 InputStream 类对象之前，图 12.18 描述了字节数据读取文件的过程。

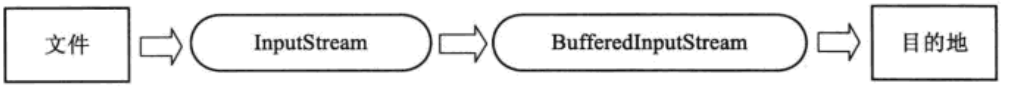


图 12.18　BufferedInputStream 读取文件过程

使用 BufferedOutputStream 输出信息和向 OutputStream 输入信息完全一样，只不过 BufferedOutputStream 有一个 flush()方法用来将缓存区的数据强制输出完。BufferedOutputStream 类也有两个构造方法：

- ☑ BufferedOutputStream(OutputStream in)
- ☑ BufferedOutputStream(OutputStream in , int size)

第一种构造函数创建一个 32 个字节的缓存区，第二种形式以指定的大小来创建缓存区。

注意　flush()方法就是用于即使缓存区没有满的情况下，也将缓存区的内容强制写入到外设，习惯上称这个过程为刷新。flush()方法只对使用缓存区的 OutputStream 类的子类有效。当调用 close()方法时，系统在关闭流之前，也会将缓存区中信息刷新到磁盘文件中。

12.5.2　BufferedReader 类与 BufferedWriter 类

BufferedReader 类与 BufferedWriter 类分别继承 Reader 类与 Writer 类。这两个类同样具有内部缓存机制，并可以行为单位进行输入/输出。

根据 BufferedReader 类的特点，总结出如图 12.19 所示的字符数据读取文件的过程。

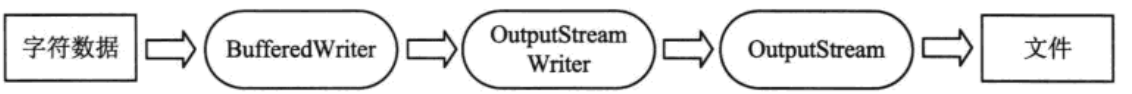


图 12.19　BufferedReader 类读取文件的过程

BufferedReader 类常用的方法如下。

- ☑ read()方法：读取单个字符。
- ☑ readLine()方法：读取一个文本行，并将其返回为字符串；若无数据可读，则返回 null。
- ☑ write(String s, int off, int len)方法：写入字符串的某一部分。
- ☑ flush()方法：刷新该流的缓存。

☑ newLine()方法：写入一个行分隔符。

在使用 BufferedWriter 类的 Write()方法时，数据并没有立刻被写入至输出流中，而是首先进入缓存区中。如果想立刻将缓存区中的数据写入输出流中，一定要调用 flush()方法。

【例 12.5】 本实例向指定的磁盘文件中写入数据，并通过 BufferedReader 类将文件中的信息分行显示。（**实例位置：光盘\TM\sl\12\11**）

```
import java.io.*;                                                    //导入 java.io 包
public class Student {                                               //创建类
    public static void main(String args[]) {                         //主方法
        String content[] = { "明日科技", "Java 部", "快速入门" };      //定义字符串数组
        File file = new File("D:/work.txt");                         //创建文件对象
        try {
            FileWriter fw = new FileWriter(file);                    //创建 FileWriter 类对象
            BufferedWriter bufw = new BufferedWriter(fw);            //创建 BufferedWriter 类对象
            for (int k = 0; k < content.length; k++) {               //循环遍历数组
                //将字符串数组中元素写入到磁盘文件中
                bufw.write(content[k]);
                //将数组中的单个元素以单行的形式写入文件
                bufw.newLine();
            }
            bufw.close();                                            //将 BufferedWriter 流关闭
            fw.close();                                              //将 FileWriter 流关闭
        } catch (Exception e) {                                      //处理异常
            e.printStackTrace();
        }
        try {
            FileReader fr = new FileReader(file);                    //创建 FileReader 类对象
            BufferedReader bufr = new BufferedReader(fr);            //创建 BufferedReader 类对象
            String s = null;                                         //创建字符串对象
            int i = 0;                                               //声明 int 型变量
            //如果文件的文本行数不为 null，则进入循环
            while ((s = bufr.readLine()) != null) {
                i++;                                                 //将变量做自增运算
                System.out.println("第" + i + "行：" + s);            //输出文件数据
            }
            bufr.close();                                            //将 FileReader 流关闭
            fr.close();                                              //将 FileReader 流关闭
        } catch (Exception e) {                                      //处理异常
            e.printStackTrace();
        }
    }
}
```

运行结果如图 12.20 所示。

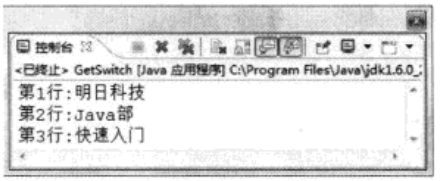


图 12.20　分行显示文件信息

12.5.3　范例 7：读取属性文件单个属性值

在编程中，有些信息经常需要修改，如 JDBC 属性，此时可以考虑将它们写入到属性文件中。本范例将完成属性文件的读入功能。运行结果如图 12.21 所示。（**实例位置：光盘\TM\sl\12\12**）

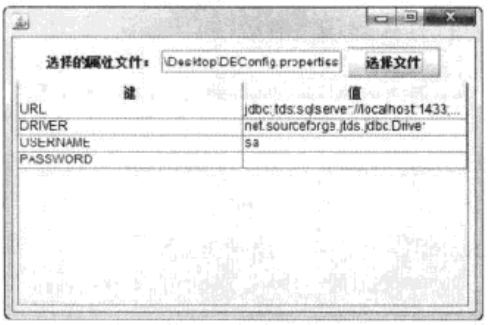


图 12.21　选择 DBConfig 属性文件

在项目中创建 PropertiesFileTool 类，该类继承了 JFrame 并增加了标签、文本框、按钮等控件。在选择完文件后，会在下面的表格中显示文件中保持的键和值。该按钮事件监听器代码如下：

```
protected void do_button_actionPerformed(ActionEvent e) {
    JFileChooser chooser = new JFileChooser();                          //创建文件选择器
    chooser.setFileFilter(new FileNameExtensionFilter("属性文件", "properties"));  //设置文件过滤器
    chooser.setFileSelectionMode(JFileChooser.FILES_ONLY);              //设置仅能选择文件
    chooser.setMultiSelectionEnabled(false);                            //禁止多个选择
    int result = chooser.showOpenDialog(this);
    if (result == JFileChooser.APPROVE_OPTION) {
        File selectFile = chooser.getSelectedFile();                    //获得选择的文件
        textField.setText(selectFile.getAbsolutePath());                //显示选择的文件路径
        Properties properties = new Properties();
        FileReader reader = null;
        DefaultTableModel model = (DefaultTableModel) table.getModel();  //获得表格模型
        try {
            reader = new FileReader(selectFile);
            properties.load(reader);                                    //加载属性文件
            Enumeration<?> keys = properties.propertyNames();           //获得属性文件的键枚举
            while (keys.hasMoreElements()) {
                String key = (String) keys.nextElement();               //获得键
```

```
                String value = properties.getProperty(key);                    //获得值
                model.addRow(new String[] { key, value });                     //向表格中增加记录
            }
        } catch (FileNotFoundException e1) {
            e1.printStackTrace();
        } catch (IOException e1) {
            e1.printStackTrace();
        } finally {
            //省略释放资源代码
        }
    }
}
```

12.5.4 范例 8：合并多个 TXT 文件

文本文件是操作系统常用的文件类型，由于没有各种复杂的格式，它可以方便地进行读写。本范例将合并多个文本文件。运行结果如图 12.22 所示。（**实例位置：光盘\TM\sl\12\13**）

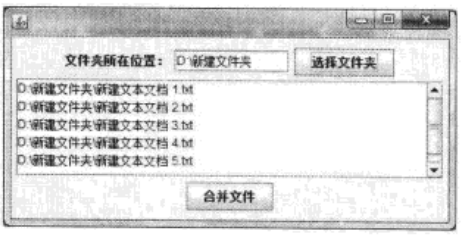


图 12.22 选择文件夹后的效果

在项目中创建 TextFileConcatenation 类，该类继承了 JFrame 并增加了标签、文本框、按钮等控件。在选择完文件夹后，单击“合并文件”按钮，可以将选择的文件夹中所有文本文件内容写入到 D 盘的 concatenation.txt 文件中。该按钮事件监听器代码如下：

```
protected void do_concatButton_actionPerformed(ActionEvent e) {
    //省略校验代码
    BufferedReader reader = null;
    FileWriter writer = null;
    try {
        writer = new FileWriter("d://concatenation.txt");                      //创建文件输出流
        for (File textFile : textFiles) {                                      //遍历用户选择的文本文件
            reader = new BufferedReader(new FileReader(textFile));             //创建缓冲输入流
            String line;
            while ((line = reader.readLine()) != null) {
                writer.write(line);                                            //将读入的数据写入到文件中
            }
        }
        JOptionPane.showMessageDialog(this, "文件合并成功！", "提示信息",
JOptionPane.INFORMATION_MESSAGE);
        return;
    } catch (IOException e1) {
        e1.printStackTrace();
    } finally {
```

```
        //省略释放资源代码
    }
}
```

12.6　数据输入/输出流

视频讲解：光盘\TM\lx\12\数据输入/输出流.exe

数据输入/输出流（DataInputStream 类与 DataOutputStream 类）允许应用程序以与机器无关的方式从底层输入流中读取基本 Java 数据类型。也就是说，当读取一个数据时，不必再关心这个数值应当是什么字节。

（1）DataInputStream 类与 DataOutputStream 类的构造方法如下。

- ☑ DataInputStream(InputStream in)：使用指定的基础 InputStream 创建一个 DataInputStream。
- ☑ DataOutputStream(OutputStream out)：创建一个新的数据输出流，将数据写入指定基础输出流。

（2）DataOutputStream 类提供了如下 3 种写入字符串的方法：

- ☑ writeBytes(String s)
- ☑ writeChars(String s)
- ☑ writeUTF(String s)

由于 Java 中的字符是 Unicode 编码，是双字节的，writeBytes 只是将字符串中的每一个字符的低字节内容写入目标设备中；而 writeChars 将字符串中的每一个字符的两个字节的内容都写到目标设备中；writeUTF 将字符串按照 UTF 编码后的字节长度写入目标设备，然后才是每一个字节的 UTF 编码。

DataInputStream 类只提供了一个 readUTF()方法返回字符串。这是因为要在一个连续的字节流读取一个字符串，如果没有特殊的标记作为一个字符串的结尾，并且事先也不知道这个字符串的长度，也就无法知道读取到什么位置才是这个字符串的结束。DataOutputStream 类中只有 writeUTF()方法向目标设备中写入字符串的长度，所以也只能准确地读回写入字符串。

【例 12.6】　分别通过 DataOutputStream 类的 writeUTF()、writeChars()、writeBytes()方法向指定的磁盘文件中写入数据，并通过 DataInputStream 类的 readUTF()方法将写入的数据输出到控制台上。（**实例位置：光盘\TM\sl\12\14**）

```
import java.io.*;                                                    //导入 java.io 包
public class Example_01 {                                            //创建类
    public static void main(String[] args) {                         //主方法
        try {
            FileOutputStream fs = new FileOutputStream("D:/word.txt");   //创建 FileOutputStream 对象
            DataOutputStream ds = new DataOutputStream(fs);              //创建 DataOutputStream 对象
            ds.writeUTF("使用 writeUTF()方法写入数据;");                  //写入磁盘文件数据
            ds.writeChars("使用 writeChars()方法写入数据;");
            ds.writeBytes("使用 writeBytes()方法写入数据.");
            ds.close();                                                  //将流关闭
            FileInputStream fis = new FileInputStream("D:/work.txt");    //创建 FileInputStream 对象
            DataInputStream dis = new DataInputStream(fis);              //创建 DataInputStream 对象
```

```
            System.out.print(dis.readUTF());                    //将文件数据输出
        } catch (Exception e) {
            e.printStackTrace();                                //输出异常信息
        }
    }
}
```

运行结果如图 12.23 所示。

使用“记事本”程序将 work.txt 打开，如图 12.24 所示。尽管在“记事本”程序中看不出 writeUTF()方法写入的字符串是“使用 writeUTF()方法写入数据”，但程序通过 readUTF()方法读回后显示在屏幕上的仍是“使用 writeUTF()方法写入数据”。但如果使用 writeChars()和 writeBytes()方法写入字符串后，想要读取回来就没那么幸运了，读者不妨编写程序尝试一下。

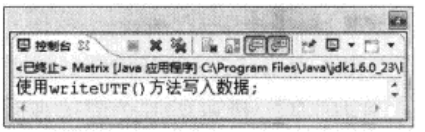


图 12.23　使用 writeUTF()方法写入数据

图 12.24　work.txt 文件内容

12.7　ZIP 压缩输入/输出流

视频讲解：光盘\TM\lx\12\ZIP 压缩输入/输出流.exe

使用 ZIP 压缩管理文件（ZIP archive），是一种十分典型的文件压缩形式，可以节省存储空间。关于 ZIP 压缩的 I/O 实现，在 Java 的内置类中，提供了非常好用的相关类，所以实际的实现方式非常容易。本节将向读者介绍使用 java.util.zip 包中的 ZipOutputStream 类与 ZipInputStream 类来实现文件的压缩/解压缩。如要从 ZIP 压缩管理文件内读取某个文件，要先找到对应该文件的“目录进入点”（从它可知该文件在 ZIP 文件内的位置），才能读取这个文件的内容。如果要将文件内容写至 ZIP 文件内，必须先写入对应于该文件的“目录进入点”，并且把要写入文件内容的位置移到此进入点所指的位置，然后再写入文件内容。

Java 实现了 I/O 数据流与网络数据流的单一接口，因此数据的压缩、网络传输和解压缩的实现比较容易。ZipEntry 类产生的对象，是用来代表一个 ZIP 压缩文件内的进入点（entry）。ZipInputStream 类用来读取 ZIP 压缩格式的文件，所支持的包括已压缩及未压缩的进入点（entry）。ZipOutputStream 类用来写出 ZIP 压缩格式的文件，而且所支持的包括已压缩及未压缩的进入点（entry）。下面介绍利用 ZipEntry、ZipInputStream 和 ZipOutputStream 3 个 Java 类实现 ZIP 数据压缩方式的编程方法。

12.7.1　压缩文件

利用 ZipOutputStream 类对象，可将文件压缩为.zip 文件。ZipOutputStream 类的构造函数如下：

```
ZipOutputStream(OutputStream out);
```

ZipOutputStream 类的常用方法如表 12.4 所示。

表 12.4　ZipOutputStream 类的常用方法

方　　法	说　　明
putNextEntry(ZipEntry e)	开始写一个新的 ZipEntry，并且将流内的位置移至此 entry 所指数据的开头
write(byte[] b , int off , int len)	将字节数组写入当前 ZIP 条目数据
finish()	完成写入 ZIP 输出流的内容，无须关闭它所配合的 OutputStream
setComment(String comment)	可设置此 ZIP 文件的注释文字

下面的实例为压缩 E 盘下的 mywork 文件夹，在 mywork 文件夹中有 work1.txt 文件和 work2.txt 文件，并将压缩后的 work.zip 文件保存在 E 盘根目录下。

【例 12.7】　创建 MyZip 类，在 zip()方法中实现使用 ZipOutputStream 类对文件进行压缩，在主方法中调用该方法。（**实例位置：光盘\TM\sl\12\15**）

```
import java.io.*;                                          //导入 java.io 包
import java.util.zip.*;                                    //导入 java.util.zip 包
public class MyZip {                                       //创建类
    private void zip(String zipFileName, File inputFile) throws Exception {
        ZipOutputStream out = new ZipOutputStream(new FileOutputStream(
                zipFileName));                             //创建 ZipOutputStream 类对象
        zip(out, inputFile, "");                           //调用方法
        System.out.println("压缩中…");                     //输出信息
        out.close();                                       //将流关闭
    }
    //方法重载
    private void zip(ZipOutputStream out, File f, String base) throws Exception {
        if (f.isDirectory()) {                             //测试此抽象路径名表示的文件是否是一个目录
            File[] fl = f.listFiles();                     //获取路径数组
            out.putNextEntry(new ZipEntry(base + "/"));    //写入此目录的 entry
            base = base.length() == 0 ? "" : base + "/";   //判断参数是否为空
            for (int i = 0; i < fl.length; i++) {          //循环遍历数组中文件
                zip(out, fl[i], base + fl[i]);
            }
        } else {
            out.putNextEntry(new ZipEntry(base));          //创建新的进入点
            FileInputStream in = new FileInputStream(f);   //创建 FileInputStream 对象
            int b;                                         //定义 int 型变量
            System.out.println(base);
            while ((b = in.read()) != -1) {                //如果没有到达流的尾部
                out.write(b);                              //将字节写入当前 ZIP 条目
            }
            in.close();                                    //关闭流
        }
    }
```

```
    public static void main(String[] temp) {                    //主方法
        MyZip book = new MyZip();                               //创建本例对象
        try {
            book.zip("E:/work.zip", new File("E:/mywork"));  //调用方法，参数为压缩后文件与要压缩文件
            System.out.println("压缩完成");                     //输出信息
        } catch (Exception ex) {
        }
    }
}
```

运行程序，可发现控制台的变化如图 12.25 所示。

从这个实例中可以看出，每一个 ZIP 文件中可能包含多个文件（本实例中包含了 work1.txt 和 work2.txt 两个文件）。使用 ZipOutputStream 类将文件写入目标 ZIP 文件时，必须先使用 ZipOutputStream 对象的 putNextEntry()方法，写入个别文件的 entry，将流内目前指到的位置移到该 entry 所指的开头位置。

执行完之后，在 E 盘的根目录下会产生 work.zip 压缩文件，将其打开后如图 12.26 所示。

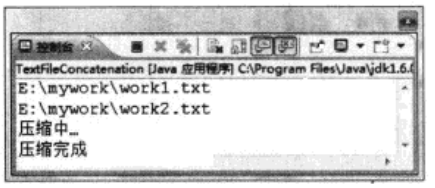


图 12.25　显示压缩信息

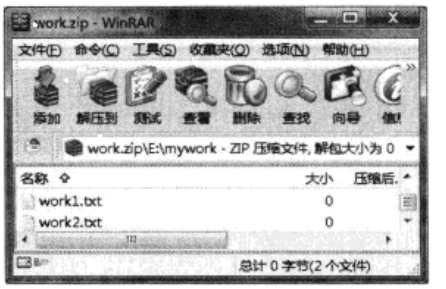


图 12.26　work.zip 文件

12.7.2　解压缩 ZIP 文件

ZipInputStream 类可读取 ZIP 压缩格式的文件，包括对已压缩和未压缩条目的支持（entry）。ZipInputStream 类的构造函数如下：

```
ZipInputStream(InputStream in)
```

ZipInputStream 类的常用方法如表 12.5 所示。

表 12.5　ZipInputStream 类的常用方法

方　法	说　明
read(byte[] b , int off , int len)	读取目标 b 数组内 off 偏移量的位置，长度是 len 字节
available()	判断是否已读完目前 entry 所指定的数据。已读完返回 0，否则返回 1
closeEntry()	关闭当前 ZIP 条目并定位流以读取下一个条目
skip(long n)	跳过当前 ZIP 条目中指定的字节数
getNextEntry()	读取下一个 ZipEntry，并将流内的位置移至该 entry 所指数据的开头
createZipEntry(String name)	以指定的 name 参数新建一个 ZipEntry 对象

下面的实例是将执行例 12.7 后生成的压缩文件 work.zip 进行解压，解压后的文件存储在 ZipEntry 类的 getName()方法获取的目录下。

【例 12.8】　创建 Decompressing 类，通过 ZipInputStream 类将例 12.7 生成的压缩文件解压到指定文件夹中。（**实例位置：光盘\TM\sl\12\16**）

```
import java.io.*;                                          //导入 java.io 包
import java.util.zip.*;                                    //导入 java.util.zip 包
public class Decompressing {                               //创建文件
    public static void main(String[] temp) {
        ZipInputStream zin;                                //创建 ZipInputStream 对象
        try {                                              //try 语句捕获可能发生的异常
            //实例化对象，指明要进行解压的文件
            zin = new ZipInputStream(new FileInputStream("E:/work.zip"));
            ZipEntry entry = zin.getNextEntry();           //获取下一个 ZipEntry
            while (((entry = zin.getNextEntry()) != null)
                    && !entry.isDirectory()) {             //如果 entry 不为空，并不在同一目录下
                File file = new File(entry.getName());     //获取文件目录
                if (!file.exists()) {                      //如果该文件不存在
                    file.createNewFile();                  //创建文件
                }
                zin.closeEntry();                          //关闭当前 entry
                System.out.println(entry.getName()+"解压成功");
            }
            zin.close();                                   //关闭流
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

运行结果如图 12.27 所示。

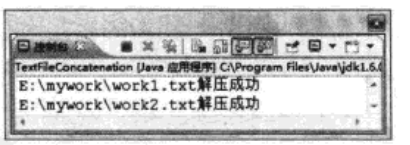


图 12.27　解压文件

程序执行完毕之后，E 盘的 mywork 文件夹中包含了 work1.txt 与 work2.txt 文件。本实例是通过此 ZipEntry 的 getName()方法，得知此文件的名称（含 path），借此来决定压缩之后的目录和文件名。使用 ZipInputStream 类来解压文件，必须先使用 ZipInputStream 类的 getNextEntry()方法来取得其内的第一个 ZipEntry。

12.7.3　范例 9：压缩包解压到指定文件夹

在获得一个以 ZIP 格式压缩的文件之后，需要将其进行解压缩，还原成压缩前的文件。本范例使用 Java 自带的压缩工具包来实现解压缩文件到指定文件夹的功能。运行结果如图 12.28 所示。（**实例位**

置：光盘\TM\sl\12\17）

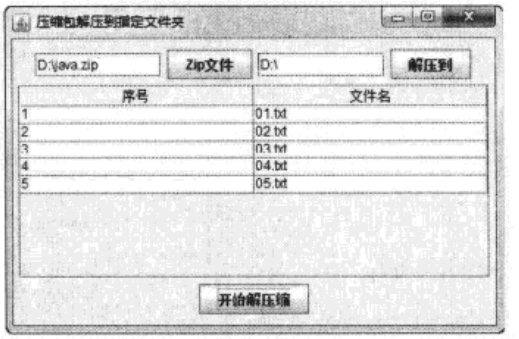


图 12.28　完成解压缩

在项目中创建 UnZipTextFileFrame 类，该类继承了 JFrame 并增加了标签、文本框、按钮等控件。选择 ZIP 文件和解压缩位置后，单击“开始解压缩”按钮完成解压缩操作，同时在表格中显示解压缩的文件。该按钮事件监听器代码如下：

```
protected void do_unzipButton_actionPerformed(ActionEvent arg0) {
    DefaultTableModel model = (DefaultTableModel) table.getModel();          //获得表格模型
    model.setColumnIdentifiers(new Object[] { "序号", "文件名" });            //设置表头
    int id = 1;                                                              //声明序号变量
    ZipFile zf = null;
    try {
        zf = new ZipFile(zipFile);                          //利用用户选择的 ZIP 文件创建 ZipFile 对象
        Enumeration e = zf.entries();                       //创建枚举变量
        while (e.hasMoreElements()) {                       //遍历枚举变量
            ZipEntry entry = (ZipEntry) e.nextElement();    //获得 ZipEntry 对象
            if (!entry.getName().endsWith(".txt")) {        //如果不是文本文件就不进行解压缩
                continue;
            }
            //利用用户选择的文件夹和 ZipEntry 对象名称创建解压后的文件
            File currentFile = new File(targetFile + File.separator + entry.getName());
            FileOutputStream out = new FileOutputStream(currentFile);
            InputStream in = zf.getInputStream(entry);      //利用获得的 ZipEntry 对象的输入流
            int buffer = 0;
            while ((buffer = in.read()) != -1) {            //将输入流写入到本地文件
                out.write(buffer);
            }
            model.addRow(new Object[] { id++, currentFile.getName() });      //增加一行表格数据
            in.close();                                                      //释放资源
            out.close();
        }
        table.setModel(model);                                               //更新表格
        JOptionPane.showMessageDialog(this, "解压缩完成");                   //提示用户解压缩完成
```

```
    } catch (ZipException e) {                          //捕获异常
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    } finally {
        //省略释放资源代码
    }
}
```

12.7.4　范例 10：压缩所有子文件夹

图 12.29　压缩完用户选择的文件夹

在压缩文件时，通常情况下一个文件夹都会有若干个子文件夹。此时该怎样处理呢？本范例向读者展示了如何压缩包含子文件夹的文件夹。运行结果如图 12.29 所示。**（实例位置：光盘\TM\sl\12\18）**

在项目中创建 ZipDirectoryFrame 类，该类继承了 JFrame 并增加了标签、文本框、按钮等控件。在选择需要压缩的文件夹之后，单击“开始压缩”按钮完成压缩任务。压缩任务的关键代码位于 zipFile()方法中，代码如下：

```
private void zipFile(List<String> path, File targetZipFile, String base) throws IOException {
    //根据给定的 targetZipFile 创建文件输出流对象
    FileOutputStream fos = new FileOutputStream(targetZipFile);
    ZipOutputStream zos = new ZipOutputStream(fos);                //利用文件输出流对象创建 Zip 输出流对象
    byte[] buffer = new byte[1024];
    for (String string : path) {                                   //遍历所有要压缩文件的路径
        File currentFile = new File(string);
        //利用要压缩文件的相对路径创建 ZipEntry 对象
        ZipEntry entry = new ZipEntry(string.substring(base.length() + 1, string.length()));
        FileInputStream fis = new FileInputStream(currentFile);
        zos.putNextEntry(entry);
        int read = 0;
        while ((read = fis.read(buffer)) != -1) {                  //将数据写入到 Zip 输出流中
            zos.write(buffer, 0, read);
        }
        zos.closeEntry();                                          //关闭 ZipEntry 对象
        fis.close();
    }
    zos.close();                                                   //释放资源
    fos.close();
}
```

12.8 经典范例

12.8.1 经典范例 1：简单的投票软件

视频讲解：光盘\TM\lx\12\简单的投票软件.exe

使用 I/O 技术能够将信息保存到本地文件中，这样就可以持久保存。本范例将完成一个简单的投票软件，其运行结果如图 12.30 所示。（**实例位置：光盘\TM\sl\12\19**）

图 12.30 显示投票结果

在项目中创建 Ballot 类，在该文件中定义 MyMin 类完成窗体上控件的增加。编写 Candidate 类完成票数的保存和获得。代码如下：

```
public int getBallot(String name) {
    File file = new File("D://count.txt");                          //创建文件对象
    FileReader fis;
    try {
        if (!file.exists())                                         //如果该文件不存在
            file.createNewFile();                                   //新建文件
        fis = new FileReader(file);
        BufferedReader bis = new BufferedReader(fis);               //创建 BufferedReader 对象
        String str[] = new String[3];
        String size;
        int i = 0;
        while ((size = bis.readLine()) != null) {                   //循环读取文件内容
            str[i] = size.trim();                                   //去除字符串中的空格
            if (str[i].startsWith(name)) {
                int length = str[i].indexOf(":");
                String sub = str[i].substring(length + 1, str[i].length());   //对字符串进行截取
                len = Integer.parseInt(sub);
                continue;
            }
            i++;
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
    return len;
}
```

12.8.2 经典范例 2：电子通讯录

视频讲解：光盘\TM\lx\12\电子通讯录.exe

在日常生活中，使用通讯录来保存姓名、电话、邮箱等信息。本范例将开发一个类似的程序，其

运行结果如图 12.31 和图 12.32 所示。（**实例位置：光盘\TM\sl\12\20**）

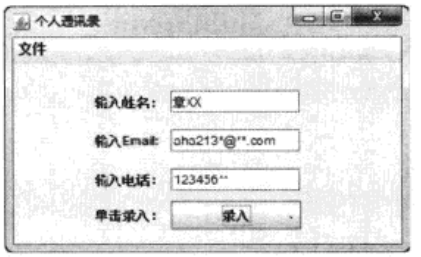


图 12.31　录入信息

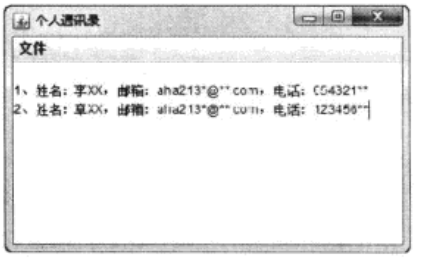


图 12.32　显示信息

在项目中创建 AddressList 类，该类继承了 JFrame 并增加了标签、文本框、按钮等控件。在输入完信息之后，单击“录入”按钮会将其保存到 D 盘中，选择“文件”/“显示”命令会显示已经录入的信息。代码如下：

```
private void kinbuttonActionPerformed(java.awt.event.ActionEvent evt) {
    try {
if(nametextField.getText().equals("")||(emailtextField.getText().equals(""))||(phonetextField.getText().equals"")))
{                                                                    //如果用户没有将信息输入完整
            JOptionPane.showMessageDialog(this, "请输入完整内容", "信息提示框", JOptionPane. WARNING_
MESSAGE);                                                            //给出提示信息
            return;                                                  //退出程序
        }
        if (!file.exists())                                          //如果文件不存在
            file.createNewFile();                                    //新建文件
        //创建 BufferedWriter 对象
        BufferedWriter out = new BufferedWriter(new OutputStreamWriter(new FileOutputStream(file, true)));
        out.write("姓名：" + nametextField.getText() + ",  ");       //向文件中写内容
        out.write("邮箱：" + emailtextField.getText() + ",  ");
        out.write("电话：" + phonetextField.getText());
        out.newLine();                                               //新建一行
        out.close();                                                 //关闭流
    } catch (Exception e1) {
        e1.printStackTrace();
    }
}
```

12.9　本章小结

本章向读者介绍了 Java 输入/输出流，Java 输入输出（I/O）机制提供了一套简单的标准化 API，以方便从不同的数据源读取和写入字符或字节数据。学习 Java 的 I/O 处理技术，必须了解 Java 的字节流和字符流。对于字节流和字符流扩展的子类也应熟练地掌握。这些子类所实现的数据流可以把数据输出到指定的设备终端，或者从指定的设备终端输入数据。另外，使用数据流来读取磁盘文件信息以及

使用数据流向磁盘文件写入信息，也是本章的重点，读者应该熟练掌握。

12.10 实战练习

1．编写程序，实现读取文件时出现一个表示读取进度的进度条。可使用 javax.swing 包提供的输入流类 ProgressMonitorInputStream。（**答案位置：光盘\TM\sl\12\21**）

2．编写程序，使用字符输入、输出流读取文件，将一段文字加密后存入文件，然后再读取，并将加密前与加密后的文件输出。（**答案位置：光盘\TM\sl\12\22**）

3．编写程序，实现当用户输入姓名和密码时，将每一个姓名和密码加在文件中。（**答案位置：光盘\TM\sl\12\23**）

第13章

Swing 程序设计

（视频讲解：51 分钟）

Swing 较早期版本中的 AWT 更为强大、性能更优良，Swing 中除了保留 AWT 中的几个重要的重量级组件之外，其他组件都为轻量级组件，这样使用 Swing 开发出的窗体风格会与当前运行平台上的窗体风格一致，同时程序员也可以在跨平台时指定窗体统一的风格与外观。Swing 的使用很复杂，本章主要讲解 Swing 中的基本要素，包括容器、组件和窗体布局。

通过阅读本章，您可以：

- 了解 Swing 组件
- 掌握使用常用窗体
- 掌握在标签上设置图标
- 掌握应用程序中的布局管理器
- 掌握常用面板
- 掌握按钮组件
- 掌握列表组件
- 掌握文本组件

PDG

13.1 Swing 概述

视频讲解：光盘\TM\lx\13\Swing 概述.exe

Swing 是 GUI（图形用户界面）开发工具包，在 AWT（抽象窗口工具包）的基础上使开发跨平台的 Java 应用程序界面成为可能，早期的 AWT 组件开发的图形用户界面，要依赖于本地系统，当把 AWT 组件开发的应用程序移植到其他平台的系统上运行时，不能保证其外观风格，因此 AWT 组件是依赖于本地系统平台的。

使用 Swing 开发的 Java 应用程序，其界面是不受本地系统平台限制的，也就是说 Swing 开发的 Java 应用程序移植到其他系统平台上时，其界面外观是不会改变的，因为 Swing 组件内部提供了相应的用户界面，而这些用户界面是纯 Java 语言编写的而不依赖于本地系统平台，所以 Swing 开发的应用程序可以方便地移植。

说明 虽然 Swing 提供的组件可以方便地开发 Java 应用程序，但是 Swing 并不能取代 AWT，在开发 Swing 程序时通常要借助于 AWT 的一些对象来共同完成应用程序的设计。

13.2 Swing 常用窗体

视频讲解：光盘\TM\lx\13\Swing 常用窗体.exe

Swing 窗体是 Swing 的一个组件，同时也是创建图形化用户界面的容器，可以将其他组件放置在窗体容器中，来完成应用程序界面的设计。

13.2.1 JFrame 框架窗体

JFrame 窗体是一个容器，它是 Swing 程序中各个组件的载体，可以将 JFrame 看作是承载这些 Swing 组件的容器。在开发应用程序时可以通过继承 java.swing.JFrame 类创建一个窗体，在这个窗体中添加组件，同时为组件设置事件。由于该窗体继承了 JFrame 类，所以它拥有一些最大化、最小化、关闭的按钮。

JFrame 在程序中的语法格式如下：

```
JFrame jf=new JFrame(title);
Container container=jf.getContentPane();
```

- ☑ jf：JFrame 类的对象。
- ☑ container：Container 类的对象，可以使用 JFrame 对象调用 getContentPane()方法获取。

读者大致应该有这样一个概念，Swing 组件的窗体通常与组件和容器相关，所以在 JFrame 对象创建完成后，需要调用 getContentPane()方法将窗体转换为容器，然后在容器中添加组件或设置布局管理器。通常这个容器用来包含和显示组件。如果需要将组件添加至容器，可以使用来自 Container 类的 add()方法进行设置。在容器中添加组件后，也可以使用 Container 类的 remove()方法将这些组件从容器中删除。

【例 13.1】　在项目中创建 Example1 类，该类集成 JFrame 类成为窗体类，在该类中创建标签组件，并添加到窗体界面中。（**实例位置：光盘\TM\sl\13\1**）

```
public class Example1 extends JFrame {                                  //定义一个类继承 JFrame 类
    public void CreateJFrame(String title) {                            //定义一个 CreateJFrame()方法
        JFrame jf = new JFrame(title);                                  //实例化一个 JFrame 对象
        Container container = jf.getContentPane();                      //获取一个容器
        JLabel jl = new JLabel("这是一个 JFrame 窗体");                  //创建一个 JLabel 标签
        jl.setHorizontalAlignment(SwingConstants.CENTER);               //使标签上的文字居中
        container.add(jl);                                              //将标签添加到容器中
        container.setBackground(Color.white);                           //设置容器的背景颜色
        jf.setVisible(true);                                            //使窗体可视
        jf.setSize(200, 150);                                           //设置窗体大小
        jf.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);     //设置窗体关闭方式
    }
    //在主方法中调用 CreateJFrame()方法
    public static void main(String args[]) {
        new Example1().CreateJFrame("创建一个 JFrame 窗体");
    }
}
```

运行结果如图 13.1 所示。

图 13.1　创建 JFrame 窗体

在例 13.1 中 Example1 类继承了 JFrame 类，在 CreateJFrame()方法中实例化 JFrame 对象。JFrame 类的常用构造方法包括 public JFrame()和 public JFrame(String title)两种形式。

JFrame 类中的两种构造方法分别为无参的构造方法与有参的构造方法，第一种形式的构造方法创建一个初始不可见、没有标题的新窗体；第二种形式的构造方法在实例化该 JFrame 对象时创建一个不可见但具有标题的窗体。可以使用 JFrame 对象调用 show()方法使窗体可见，但是这个方法早已经被新版 JDK 所弃用，而通常使用 setVisible(true)方法使窗体可见。

同时可以使用 setSize(int x,int y)方法设置窗体大小，其中 x 与 y 变量分别代表窗体的宽与高。

创建窗体后，需要给予窗体一个关闭方式，可以调用 setDefaultCloseOperation()方法关闭窗体。Java 为窗体关闭提供了几种方式，常用的有以下 4 种方式，这几种操作实质上是一个 int 类型的常量，被封装在 javax.swing.WindowConstants 接口中。

- ☑ DO_NOTHING_ON_CLOSE
- ☑ DISPOSE_ON_CLOSE
- ☑ HIDE_ON_CLOSE
- ☑ EXIT_ON_CLOSE

第一种窗体退出方式代表什么都不做就将窗体关闭；而第二种退出方式则代表任何注册监听程序

对象后会自动隐藏并释放窗体；第 3 种方式表示隐藏窗口的默认窗口关闭；而第 4 种退出方式表示退出应用程序默认窗口关闭。

13.2.2 JDialog 窗体

JDialog 窗体是 Swing 组件中的对话框，它继承了 AWT 组件中的 java.awt.Dialog 类。

JDialog 窗体的功能是从一个窗体中弹出另一个窗体，就像是在使用 IE 浏览器时弹出的确定对话框一样。JDialog 窗体实质上就是另一种类型的窗体，它与 JFrame 窗体类似，JDialog 窗体在使用时也需要调用 getContentPane()方法将窗体转换为容器，然后在容器中设置窗体的特性。

在应用程序中创建 JDialog 窗体需要实例化 JDialog 类，通常使用以下几个 JDialog 类的构造方法。

- ☑ public JDialog()：创建一个没有标题和父窗体的对话框。
- ☑ public JDialog(Frame f)：创建一个指定父窗体的对话框，但该窗体没有标题。
- ☑ public JDialog(Frame f,boolean model)：创建一个指定类型的对话框，并指定父窗体，但该窗体没有指定标题。
- ☑ public JDialog(Frame f,String title)：创建一个指定标题和父窗体的对话框。
- ☑ public JDialog(Frame f,String title,boolean model)：创建一个指定标题、窗体和模式的对话框。

下面来看一个实例，该实例主要实现单击 JFrame 窗体中的按钮后，弹出一个对话框窗体。

【例 13.2】 在项目中创建 MyJDialog 类，该类继承 JDialog 窗体，并在窗体中添加按钮，当用户单击该按钮后，将弹出一个对话框窗体。（**实例位置：光盘\TM\sl\13\2**）

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class MyJDialog extends JDialog{                          //创建新类继承 JDialog 类
    public MyJDialog(){
        //实例化一个 JDialog 类对象，指定对话框的父窗体、窗体标题和类型
        super(new MyFrame(),"第一个 JDialog 窗体",true);
        Container container=getContentPane();                    //创建一个容器
        container.add(new JLabel("这是一个对话框"));             //在容器中添加标签
        setSize(100,100);                                        //设置对话框窗体大小
    }
    public static void main(String args[]){
        new MyJDialog();                                         //实例化 MyJDialog 类对象
    }
}
class MyFrame extends JFrame{                                    //创建新类
    public MyFrame(){
        Container container=getContentPane();                    //创建一个容器
        container.setLayout(null);
        JLabel jl=new JLabel("这是一个 JFrame 窗体");            //在窗体中设置标签
        jl.setHorizontalAlignment(SwingConstants.CENTER);        //将标签的文字置于标签中间位置
        container.add(jl);
        JButton bl=new JButton("弹出对话框");                    //定义一个按钮
```

```
        bl.setBounds(10,10,100,21);
        bl.addActionListener(new ActionListener(){                    //为按钮添加鼠标单击事件
            public void actionPerformed(ActionEvent e){
                new MyJDialog().setVisible(true);                     //使 MyJDialog 窗体可见
            }
        });
        container.add(bl);                                            //将按钮添加到容器中
        ...//省略部分代码
    }
}
```

运行结果如图 13.2 所示。

图 13.2　弹出 JDialog 窗体

在本实例中，为了使对话框在父窗体弹出，定义了一个 JFrame 窗体，首先在该窗体中定义一个按钮，然后为此按钮添加一个鼠标单击监听事件（在这里使用了匿名内部类的形式，如果读者对这部分的代码实现有疑问，不妨回顾一下第 11 章该部分的内容，对于监听事件笔者会在后续章节中进行讲解，在这里读者只要知道这部分代码是待用户单击该按钮后实现的某种功能即可），这里使用 new MyJDialog().setVisible(true)语句使对话框窗体可见，这样就实现了当用户单击该按钮后将弹出对话框的功能。

在 MyJDialog 类中，由于它继承了 JDialog 类，所以可以在构造方法中使用 super 关键字调用 JDialog 构造方法。在这里使用了 public JDialog(Frame f,String title,boolean model)形式的构造方法，相应地设置了笔者自定义的 JFrame 窗体以及对话框的标题和窗体类型。

在本实例代码中可以看到，JDialog 窗体与 JFrame 窗体形式基本相同，甚至在设置窗体的特性时调用的方法名称都基本相同，如设置窗体大小、设置窗体关闭状态等。

13.2.3　范例 1：设置窗体大小

用户打开软件后，首先看到的就是软件窗体的大小，那么如何设置窗体的大小就成了摆在开发者面前的一个首要问题。本范例将告诉读者，如何使用 Java 语言实现限制窗体大小的功能。运行结果如图 13.3 所示。**（实例位置：光盘\TM\sl\13\3）**

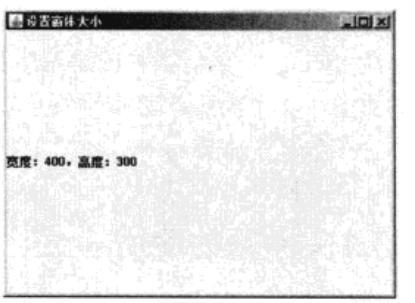


图 13.3　限制窗体大小

在项目中创建窗体类 ControlFormSize，在窗体中添加一个标签控件，编写程序代码，设置窗体的标题、默认关闭方式与窗体大小，然后在标签控件中显示对窗体设置的大小。代码如下：

```
public ControlFormSize() {
    setTitle("设置窗体大小");                                          //设置窗体标题
```

```
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);          //默认关闭方式
    setSize(400, 300);                                       //设置窗体大小
    contentPane = new JPanel();                              //创建内容面板
    contentPane.setLayout(new BorderLayout(0, 0));
    setContentPane(contentPane);                             //设置内容面板
    JLabel label = new JLabel("宽度：400，高度：300");        //创建标签控件
    contentPane.add(label, BorderLayout.CENTER);             //添加标签控件到窗体
}
```

13.2.4 范例 2：禁止改变窗体的大小

本范例主要实现禁止改变窗体大小的功能，运行本范例，默认可以通过鼠标拖曳的方式改变窗体大小，但是当用户单击“禁止改变窗体大小”按钮后，窗体将会以一种对话框的方式进行显示，这时就不可以再用鼠标拖曳的方式改变窗体的大小了。运行结果如图 13.4 所示。（**实例位置：光盘\TM\sl\13\4**）

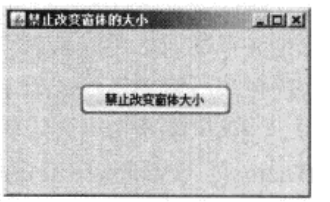


图 13.4 禁止改变窗体的大小

（1）在项目中新建窗体类 LimitChangeFormSize，在该窗体中添加一个按钮控件，用来执行禁止改变窗体大小功能。

（2）编写“禁止改变窗体大小”按钮的事件处理方法，在该方法中设置窗体的 resizable 属性为 false。代码如下：

```
protected void do_button_actionPerformed(ActionEvent e) {
    setResizable(false);                                     //禁止改变窗体大小
}
```

13.3 标签组件与图标

视频讲解：光盘\TM\lx\11\标签组件与图标.exe

在 Swing 中显示文本或提示信息的方法是使用标签，它支持文本字符串和图标。在应用程序的用户界面中，一个简短的文本标签可以使用户知道这些组件的目的，所以标签在 Swing 中是比较常用的组件。在本节中将探讨 Swing 标签的用法、如何创建标签，以及如何在标签上放置文本与图标。

13.3.1 标签的使用

标签由 JLabel 类定义，它的父类为 JComponent 类。

标签可以显示一行只读文本、一个图像或带图像的文本，它并不能产生任何类型的事件，只是简单地显示文本和图片，但是可以使用标签的特性指定标签上文本的对齐方式。

JLabel 类提供了多种构造方法，这样可以创建多种标签，如显示只有文本的标签、只有图标的标签或是包含文本与图标的标签。JLabel 类常用的几个构造方法如下。

- ☑ public JLabel()：创建一个不带图标和文本的 JLabel 对象。
- ☑ public JLabel(Icon icon)：创建一个带图标的 JLabel 对象。
- ☑ public JLabel(Icon icon,int aligment)：创建一个带图标的 JLabel 对象，并设置图标水平对齐方式。
- ☑ public JLabel(String text,int aligment)：创建一个带文本的 JLabel 对象，并设置文字水平对齐方式。
- ☑ public JLabel(String text,Icon icon,int aligment)：创建一个带文本、带图标的 JLabel 对象，并设置标签内容的水平对齐方式。

13.3.2　图标的使用

Swing 中的图标可以放置在按钮、标签等组件上，用于描述组件的用途。图标可以用 Java 支持的图片文件类型进行创建，也可以使用 java.awt.Graphics 类提供的功能方法来创建。

1．创建图标

在 Swing 中通过 Icon 接口来创建图标，可以在创建时给定图标的大小、颜色等特性。如果使用 Icon 接口，必须实现 Icon 接口中的以下 3 个方法：

- ☑ public int getIconHeight()
- ☑ public int getIconWidth()
- ☑ public void paintIcon(Component arg0, Graphics arg1, int arg2, int arg3)

getIconHeigth()与 getIconWidth()方法顾名思义是获取图表长与宽的方法，paintIcon()方法用于实现在指定坐标位置画图。

下面举一个实现 Icon 接口创建图标的例子。

【例 13.3】 在项目中创建实现 Icon 接口的 DrawIcon 类，该类实现自定义的图标类。（**实例位置：光盘\TM\sl\13\5**）

```
import java.awt.*;
import javax.swing.*;
public class DrawIcon implements Icon {                    //实现 Icon 接口
    private int width;                                     //声明图标的宽
    private int height;                                    //声明图标的长
    public int getIconHeight() {                           //实现 getIconHeight()方法
        return this.height;
    }
    public int getIconWidth() {                            //实现 getIconWidth()方法
        return this.width;
    }
    public DrawIcon(int width,int height){                 //定义构造方法
        this.width=width;
        this.height=height;
```

```
    }
    //实现 paintIcon()方法
    public void paintIcon(Component arg0, Graphics arg1, int x, int y) {
        arg1.fillOval(x, y, width, height);                    //绘制一个圆形
    }
    public static void main(String[] args) {
        DrawIcon icon=new DrawIcon(15,15);
        //创建一个标签，并设置标签上的文字在标签正中间
        JLabel j=new JLabel("测试",icon,SwingConstants.CENTER);
        JFrame jf=new JFrame();                                //创建一个 JFrame 窗口
        Container c=jf.getContentPane();
        ...//省略部分代码
    }
}
```

运行结果如图 13.5 所示。

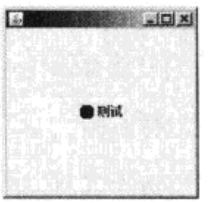


图 13.5　实现 Icon 接口创建图标

在本实例中，由于 DrawIcon 类继承了 Icon 接口，所以在该类中必须实现 Icon 接口中定义的所有方法，其中在实现 paintIcon()方法中使用 Graphics 类中的方法绘制一个圆形的图标，其余实现接口的方法为返回图标长与宽。在 DrawIcon 类的构造方法中设置了图标的长与宽，这样如果需要在窗体中使用图标，就可以使用如下代码创建图标。

```
DrawIcon icon=new DrawIcon(15,15);
```

在前文中提到过，一般情况下会将图标放置在按钮或标签上，在这里将图标放置在标签上，然后将标签添加到容器中，这样就实现了在窗体中使用图标的功能。

2．使用图片图标

Swing 中的图标除了可以绘制之外，还可以使用某个特定的图片创建。Swing 利用 javax.swing.ImageIcon 类根据现有图片创建图标，ImageIcon 类实现了 Icon 接口，同时 Java 支持多种图片格式。

ImageIcon 类有多个构造方法，其中几个常用的构造方法如下。

- ☑ public ImageIcon()：该构造方法创建一个通用的 ImageIcon 对象，当真正需要设置图片时再使用 ImageIcon 对象调用 setImage(Image image)方法来操作。
- ☑ public ImageIcon(Image image)：可以直接从图片源创建图标。
- ☑ public ImageIcon(Image image,Strign description)：除了可以从图片源创建图标之外，还可以为这个图标添加简短的描述，但这个描述不会在图标上显示，可以使用 getDescription()方法获取这个描述。
- ☑ public ImageIcon(URL url)：该构造方法利用位于计算机网络上的图像文件创建图标。

下面来看一个创建图片图标的实例。

【例 13.4】 在项目中创建实现 Icon 接口的 DrawIcon 类，该类实现自定义的图标类。（**实例位置：光盘\TM\sl\13\6**）

```
import java.awt.Container;
import java.net.URL;
import javax.swing.*;
public class MyImageIcon extends JFrame{
    public MyImageIcon() {
        Container container = getContentPane();
        JLabel jl = new JLabel("这是一个 JFrame 窗体",JLabel.CENTER);        //创建一个标签
        URL url=MyImageIcon.class.getResource("imageButton.jpg");          //获取图片所在的 URL
        Icon icon = new ImageIcon(url);                                     //实例化 Icon 对象
        jl.setIcon(icon);                                                   //为标签设置图片
        jl.setHorizontalAlignment(SwingConstants.CENTER);                   //设置文字放置在标签中间
        jl.setOpaque(true);                                                 //设置标签为不透明状态
        container.add(jl);                                                  //将标签添加到容器中
        setSize(250,100);                                                   //设置窗体大小
        setVisible(true);                                                   //使窗体可见
        setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);            //设置窗体关闭模式
    }
    public static void main(String args[]){
        new MyImageIcon();                                                  //实例化 MyImageIcon 对象
    }
}
```

运行结果如图 13.6 所示。

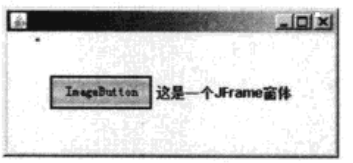


图 13.6　使用图片创建图标

注意　java.lang.Class 类中的 getResource()方法可以获取资源文件的 URL 路径。例 13.4 中该方法的参数是 imageButton.jpg，这个路径是相对于 MyImageIcon 类文件的，所以可将 imageButton.jpg 图片文件与 MyImageIcon 类文件放在同一个文件夹下。

在本实例中，首先使用 public JLabel(String text,int aligment)构造方法创建一个 JLabel 对象，然后调用 setIcon()方法为标签设置图标。当然读者也可以选择在初始化 JLabel 对象时为标签指定图标，这时需要获取一个 Icon 实例。

13.3.3　范例 3：为图片添加说明

使用图片的标签在窗体中添加一张图片，再使用文字标签为图片添加说明。运行结果如图 13.7 所示。（**实例位置：光盘\TM\sl\13\7**）

图 13.7　为图片添加说明

在项目中创建 UseLabelFrame 类，该类继承自 JFrame 类成为窗体类，然后分别创建显示文本和图片的标签组件。代码如下：

```
public UseLabelFrame() {
    super();                                                    //调用父类的构造方法
    setTitle("使用标签组件");                                    //设置窗体标题
    setBounds(100, 100, 330, 200);                              //设置窗体的显示位置和大小
    //设置窗体的默认关闭方式为关闭并退出应用程序
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    final JLabel label = new JLabel();                          //创建标签组件
    label.setText("这是一幅美丽的山水照片：");                    //设置标签的标题
    getContentPane().add(label, BorderLayout.NORTH);            //将标签添加到窗体的北部（即上部）位置
    URL url = UseLabelFrame.class.getResource("picture.jpg");   //获得图片的 URL
    Icon icon = new ImageIcon(url);                             //创建图标
    final JLabel label_1 = new JLabel();                        //创建标签组件
    label_1.setIcon(icon);                                      //设置标签的图标
    getContentPane().add(label_1, BorderLayout.CENTER);         //将标签添加到窗体的中部位置
}
public static void main(String[] args) {
    UseLabelFrame frame = new UseLabelFrame();                  //创建窗体对象
    frame.setVisible(true);                                     //显示窗体
}
```

13.4　常用布局管理器

视频讲解：光盘\TM\lx\13\常用布局管理器.exe

在 Swing 中，每个组件在容器中都有一个具体的位置和大小，在容器中摆放各种组件时很难判断其具体位置和大小。布局管理器提供 Swing 组件安排展示在容器中的方法，它提供了基本的布局功能，使用布局管理器较程序员直接在容器中控制 Swing 组件的位置和大小要方便得多，可以有效地处理整个窗体的布局。Swing 提供的常用布局管理器包括流布局管理器、边界布局管理器、网格布局管理器。本节将探讨 Swing 中的常用布局管理器。

13.4.1　绝对布局

在 Swing 中，除了使用布局管理器之外也可以使用绝对布局。绝对布局，顾名思义，就是硬性指定组件在容器中的位置和大小，可以使用绝对坐标的方式来指定组件的位置。

使用绝对布局的步骤如下：

（1）使用 Container.setLayout(null)方法取消布局管理器。

（2）使用 Component.setBounds()方法设置每个组件的大小与位置。

下面来看一个绝对布局的例子。

【例 13.5】　在项目中创建继承 JFrame 窗体组件的 AbsolutePosition 类，设置布局管理器为 null，即绝对定位的布局方式，并创建两个按钮组件，将按钮分别定位在不同的窗体位置上。（**实例位置：光盘\TM\sl\13\8**）

```
public class AbsolutePosition extends JFrame{
    public AbsolutePosition(){
        setTitle("本窗体使用绝对布局");                    //设置该窗体的标题
        setLayout(null);                                   //使该窗体取消布局管理器设置
        setBounds(0,0,200,150);                            //绝对定位窗体的位置与大小
        Container c=getContentPane();                      //创建容器对象
        JButton b1=new JButton("按钮 1");                  //创建按钮
        JButton b2=new JButton("按钮 2");                  //创建按钮
        b1.setBounds(10, 30, 80, 30);                      //设置按钮的位置与大小
        b2.setBounds(60, 70, 100, 20);
        c.add(b1);                                         //将按钮添加到容器中
        c.add(b2);
        setVisible(true);                                  //使窗体可见
        //设置窗体关闭方式
        setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
    }
    public static void main(String[] args) {
        new AbsolutePosition();
    }
}
```

运行结果如图 13.8 所示。

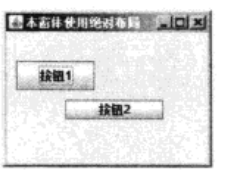


图 13.8　在应用程序中使用绝对布局

13.4.2　流布局管理器

流（FlowLayout）布局管理器是布局管理器中最基本的布局管理器。流布局管理器在整个容器中的布局正如其名，像“流”一样从左到右摆放组件，直到占据了这一行的所有空间，然后再向下移动一行。默认情况下，组件在每一行上都是居中排列的，但是通过设置也可以更改组件在每一行上的排列位置。

FlowLayout 类中具有以下常用的构造方法：

☑ public FlowLayout()

☑ public FlowLayout(int alignment)

☑ public FlowLayout(int alignment,int horizGap,int vertGap)

构造方法中的 alignment 参数表示使用流布局管理器后组件在每一行的具体摆放位置。它可以被赋予以下 3 个值之一：

☑ FlowLayout.LEFT=0

☑ FlowLayout.CENTER=1

☑ FlowLayout.RIGHT=2

上述 3 个值分别代表容器使用流布局管理器后组件在每一行中的摆放位置。例如，将 alignment 设置为 0 时，每一行的组件将被指定按照左对齐排列；而将 alignment 设置为 2 时，每一行的组件将被指定为按照右对齐排列。

在 public FlowLayout(int alignment,int horizGap,int vertGap)构造方法中还存在 horizGap 与 vertGap 两个参数，这两个参数分别以像素为单位指定组件之间的水平间隔与垂直间隔。

下面是一个流布局管理器的例子。在此例中，首先将容器的布局管理器设置为 FlowLayout，然后在窗体上摆放组件。

【例 13.6】 在项目中创建 FlowLayoutPosition 类，该类继承 JFrame 类成为窗体组件。设置该窗体的布局管理器为 FlowLayout 布局管理器的实例对象。（**实例位置：光盘\TM\sl\13\9**）

```
public class FlowLayoutPosition extends JFrame{
    public FlowLayoutPosition(){
        setTitle("本窗体使用流布局管理器");                    //设置窗体标题
        Container c=getContentPane();
        //设置窗体使用流布局管理器，使组件右对齐，并且设置组件之间的水平间隔与垂直间隔
        setLayout(new FlowLayout(2,10,10));
        for(int i=0;i<10;i++){                                 //在容器中循环添加 10 个按钮
            c.add(new JButton("button"+i));
        }
        setSize(300, 200);                                     //设置窗体大小
        setVisible(true);                                      //设置窗体可见
        //设置窗体关闭方式
        setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);
    }
    public static void main(String[] args) {
        new FlowLayoutPosition();
    }
}
```

运行结果如图 13.9 所示。

从本实例的运行结果中可以看到，如果改变整个窗体的大小，相应地其中组件的摆放位置也会发生变化，这正好验证了使用流布局管理器时组件从左到右摆放，当组件填满一行后，将自动换行，直到所有的组件都摆放在容器中为止。

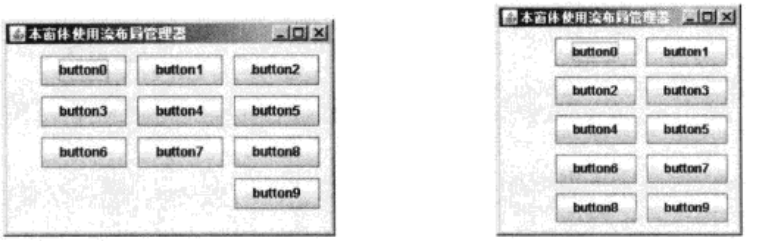


图 13.9　在应用程序中使用流布局管理器

13.4.3　边界布局管理器

在默认不指定窗体布局的情况下，Swing 组件的布局模式是边界（BorderLayout）布局管理器。例如在例 13.3 中，笔者在容器中只添加了一个标签组件，在运行结果中可以看到，这个标签被放置在窗体中间，并且整个组件占据了窗体的所有空间，实质上在这个容器中默认使用了边界布局管理器。

但是边界布局管理器的功能不止如此，还可以将容器划分为东、南、西、北、中 5 个区域，可以将组件加入到这 5 个区域中。容器调用 Container 类的 add()方法添加组件时可以设置此组件在边界布局管理器中的区域，区域的控制可以由 BorderLayout 类中的成员变量来决定，这些成员变量的具体含义如表 13.1 所示。

表 13.1　BorderLayout 类的主要成员变量

成 员 变 量	说　　明
BorderLayout.NORTH	在容器中添加组件时，组件置于顶端
BorderLayout.SOUTH	在容器中添加组件时，组件置于底端
BorderLayout.EAST	在容器中添加组件时，组件置于右端
BorderLayout.WEST	在容器中添加组件时，组件置于左端
BorderLayout.CENTER	在容器中添加组件时，组件置于中间开始填充，直到与其他组件边界连接

下面举一个在容器中设置边界布局管理器的例子，在本实例中分别在容器的东、南、西、北、中区域添加 5 个按钮。

【例 13.7】　在项目中创建 BorderLayoutPosition 类，该类继承 JFrame 类成为窗体组件，设置该窗体的布局管理器使用 BorderLayout 类的实例对象。（**实例位置：光盘\TM\sl\13\10**）

```
public class BorderLayoutPosition extends JFrame{
    String[]
border={BorderLayout.CENTER,BorderLayout.NORTH,BorderLayout.SOUTH,BorderLayout.WEST,BorderLayout.EAST};
                                                        //定义一个组件摆放位置的数组
    String[] buttonName={"center button","north button","south button","west button","east button"};
    public BorderLayoutPosition(){
        setTitle("这个窗体使用边界布局管理器");
        Container c=getContentPane();                   //定义一个容器
        setLayout(new BorderLayout());                  //设置容器为边界布局管理器
        for(int i=0;i<border.length;i++){
            c.add(border[i],new JButton(buttonName[i]));  //在容器中添加按钮，并设置按钮布局
        }
```

```
        setSize(350, 200);                                    //设置窗体大小
        setVisible(true);                                     //使窗体可视
        //设置窗体关闭方式
        setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);
    }
    public static void main(String[] args) {
        new BorderLayoutPosition();
    }
}
```

运行结果如图 13.10 所示。

在本实例中将布局以及组件名称分别放置在数组中，然后设置容器使用边界布局管理器，最后在循环中将按钮添加至容器中，并设置组件布局。add()方法提供在容器中添加组件的功能，并同时设置组件的摆放位置。

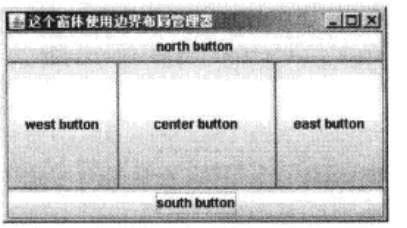


图 13.10　在应用程序中使用边界布局管理器

13.4.4　网格布局管理器

网格（GridLayout）布局管理器将容器划分为网格，所以组件可以按行和列进行排列。在网格布局管理器中，每一个组件的大小都相同，并且网格中的空格的个数由网格的行数和列数决定，例如一个两行两列的网格能产生 4 个大小相等的网格。组件从网格的左上角开始，按照从左到右、从上到下的顺序加入到网格中，而且每一个组件都会填满整个网格，改变窗体的大小，组件也会随之改变大小。

网格布局管理器主要有以下两个常用构造方法：

- ☑ public GridLayout(int rows,int columns)
- ☑ public GridLayout(int rows,int columns,int horizGap,int vertGap)

在上述构造方法中，rows 与 columns 参数代表网格的行数与列数，这两个参数只有一个参数可以为 0，代表一行或一列可以排列任意多个组件；参数 horizGap 与 vertGap 指定网格之间的间距，其中 horizGap 参数指定网格之间的水平间距，而 vertGap 参数指定网格之间的垂直间距。

下面来看一个在应用程序中使用网格布局管理器的例子。

【例 13.8】　在项目中创建 GridLayoutPosition 类，该类继承 JFrame 类成为窗体组件，设置该窗体使用 GridLayout 布局管理器。（**实例位置：光盘\TM\sl\13\11**）

```
public class GridLayoutPosition extends JFrame{
    public GridLayoutPosition(){
        Container c=getContentPane();
        setLayout(new GridLayout(7,3,5,5));          //设置容器使用网格布局管理器，设置 7 行 3 列的网格
        for(int i=0;i<20;i++){
            c.add(new JButton("button"+i));          //循环添加按钮
        }
        setSize(300, 300);
        setTitle("这是一个使用网格布局管理器的窗体");
        setVisible(true);
        setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
    }
```

```
    public static void main(String[] args) {
        new GridLayoutPosition();
    }
}
```

运行结果如图 13.11 所示。

图 13.11　在应用程序中使用网格布局管理器

13.5　常 用 面 板

视频讲解：光盘\TM\lx\13\常用面板.exe

面板也是一个 Swing 容器，它可以作为容器容纳其他组件，但它也必须被添加到其他容器中。Swing 中常用的面板包括 JPanel 面板和 JScrollPane 面板。

13.5.1　JPanel 面板

JPanel 面板可以聚集一些组件来布局。读者首先应该明确的是面板也是一种容器，因为它也继承自 java.awt.Container 类。

【例 13.9】　在项目中创建 JPanelTest 类，该类继承 JFrame 类成为窗体组件，在该类中创建 4 个 JPanel 面板组件，并将它们添加到窗体中。（**实例位置：光盘\TM\sl\13\12**）

```
public class JPanelTest extends JFrame{
    public JPanelTest(){
        Container c=getContentPane();
        c.setLayout(new GridLayout(2,1,10,10));                    //将整个容器设置为 2 行 1 列的网格布局
        //初始化一个面板，设置 1 行 3 列的网格布局
        JPanel p1=new JPanel(new GridLayout(1,3,10,10));
        JPanel p2=new JPanel(new GridLayout(1,2,10,10));
        JPanel p3=new JPanel(new GridLayout(1,2,10,10));
        JPanel p4=new JPanel(new GridLayout(2,1,10,10));
        p1.add(new JButton("1"));                                  //在面板中添加按钮
        ...//省略部分代码
```

```
        c.add(p1);                                          //在容器中添加面板
        c.add(p2);
        c.add(p3);
        c.add(p4);
        ...//省略部分代码
    }
    public static void main(String[] args) {
        new JPanelTest();
    }
}
```

运行结果如图 13.12 所示。

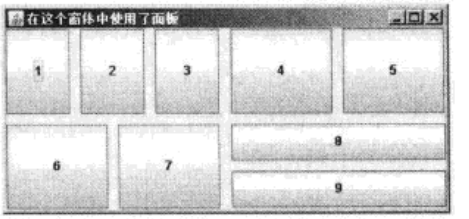


图 13.12　在应用程序中使用 JPanel 面板

13.5.2　JScrollPane 面板

在设置界面时，可能会遇到在一个较小的容器窗体中显示一个较大部分内容的情况，这时可以使用 JScrollPane 面板。JScrollPane 面板是带滚动条的面板，它也是一种容器，但是 JScrollPane 只能放置一个组件，并且不可以使用布局管理器。如果需要在 JScrollPane 面板中放置多个组件，需要将多个组件放置在 JPanel 面板上，然后将 JPanel 面板作为一个整体组件添加在 JScrollPane 面板上。

下面举一个 JScrollPane 面板的例子。

【例 13.10】　在项目中创建 JScrollPaneTest 类，该类继承 JFrame 类成为窗体组件，在该类中创建 4 个 JScrollPane 面板组件，并将它们添加到窗体中。（**实例位置：光盘\TM\sl\13\13**）

```
public class JScrollPaneTest extends JFrame{
    public JScrollPaneTest() {
        Container c=getContentPane();                       //创建容器
        JTextArea ta=new JTextArea(20,50);                  //创建文本区域组件
        ta.setText("带滚动条的文字编译器");
        JScrollPane sp=new JScrollPane(ta);                 //创建 JScrollPane 面板对象
        c.add(sp);                                          //将该面板添加到该容器中
        ...//省略部分代码
    }
    public static void main(String[] args) {
        new JScrollPaneTest();
    }
}
```

运行结果如图 13.13 所示。

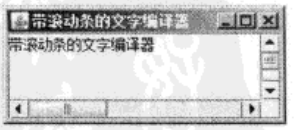


图 13.13　在应用程序中使用 JScroll Pane 面板

从本实例的运行结果中可以看到，在窗体中创建一个带滚动条的文字编译器，首先需要初始化编译器（在 Swing 中编译器类为 JTextArea 类），并在初始化时完成编译器的大小指定（如果读者对编译器的概念有些困惑，可以参见

后续章节）。当创建带滚动条的面板时，将编译器加入面板中，最后将带滚动条的编译器放置在容器中即可。

13.6 按钮组件

视频讲解：光盘\TM\lx\13\按钮组件.exe

按钮在 Swing 中是较为常见的组件，它用于触发特定动作。Swing 中提供了很多按钮，包括提交按钮、复选框、单选按钮等，这些按钮都是从 AbstractButton 类中继承而来的，本节将着重讲解这些按钮的使用。

13.6.1 提交按钮组件

Swing 中的提交按钮由 JButton 对象表示，其构造方法主要有如下几种形式：

- ☑ public JButton()
- ☑ public JButton(String text)
- ☑ public JButton(Icon icon)
- ☑ public JButton(String text,Icon icon)

通过使用上述构造方法，在 Swing 按钮上不仅能显示文本标签，还可以显示图标。上述构造方法中的第一个构造方法可以生成不带任何文本组件的对象和图标，可以使用相应方法为按钮设置指定的文本和图标；其他构造方法都在初始化时指定了按钮上显示的图标或文字。

来看一个例子，引例设置的窗体中指定了一个同时带文字与图标的按钮。

【例 13.11】 在项目中新建 JButtonTest 类，该类继承 JFrame 类成为窗体组件，在该窗体中创建按钮组件，并为按钮设置图标。（**实例位置：光盘\TM\sl\13\14**）

```
public class JButtonTest extends JFrame {
    public JButtonTest() {
        URL url = JButtonTest.class.getResource("imageButtoo.jpg");
        Icon icon = new ImageIcon(url);
        setLayout(new GridLayout(3, 2, 5, 5));                    //设置网格布局管理器
        Container c = getContentPane();                           //创建容器
        for (int i = 1; i <= 5; i++) {

            JButton J = new JButton("按钮" + i, icon);            //创建按钮，同时设置按钮文字与图标
            c.add(J);                                             //在容器中添加按钮
            if (i % 2 == 0) {
                J.setEnabled(false);                              //设置其中一些按钮不可用
            }
        }
        JButton jb = new JButton();                               //实例化一个没有文字与图片的按钮
        jb.setMaximumSize(new Dimension(90, 30));                 //设置按钮与图片相同大小
```

```
        jb.setIcon(icon);                                    //为按钮设置图标
        jb.setHideActionText(true);
        jb.setToolTipText("图片按钮");                        //设置按钮提示为文字
        jb.setBorderPainted(false);                          //设置按钮边界不显示
        c.add(jb);                                           //将按钮添加到容器中
        setTitle("创建带文字与图片的按钮");
        setSize(350, 150);
        setVisible(true);
        setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);
    }
    public static void main(String args[]) {
        new JButtonTest();
    }
}
```

运行结果如图 13.14 所示。

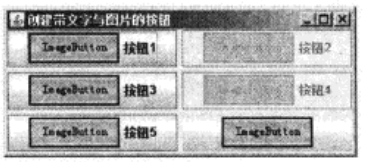


图 13.14　按钮组件的应用

说明 上述这些设置按钮属性的方法多来自于 JButton 的父类 AbstractButton 类，这里只是简单列举了几个常用的方法，读者如果有需要可以查询 Java API，使用自己需要的方法实现相应的功能。

13.6.2　单选按钮组件

在默认情况下，单选按钮（JRadioButton）显示一个圆形图标，并且通常在该图标旁放置一些说明性文字，而在应用程序中，一般将多个单选按钮放置在按钮组中，使这些单选按钮表现出某种功能，当用户选中某个单选按钮后，按钮组中其他按钮将被自动取消。单选按钮是 Swing 组件中 JRadioButton 类的对象，该类是 JToggleButton 的子类，而 JToggleButton 类又是 AbstractButton 类的子类，所以控制单选按钮的诸多方法都是 AbstractButton 类中的方法。

1．单选按钮

可以使用 JRadioButton 类中的构造方法创建单选按钮对象。JRadioButton 类的常用构造方法主要有以下几种形式：

- ☑ public JRadioButton()
- ☑ public JRadioButton(Icon icon)
- ☑ public JRadioButton(Icon icon,boolean selected)
- ☑ public JRadioButton(String text)
- ☑ public JRadioButton(String text,Icon icon)

☑　public JRadioButton(String text,Icon icon,boolean selected)

根据上述构造方法的形式，可以知道在初始化单选按钮时，同时设置单选按钮的图标、文字以及默认是否被选中等属性。

2．按钮组

在 Swing 中存在一个 ButtonGroup 类用于产生按钮组，如果希望将所有的单选按钮放置在按钮组中，需要实例化一个 JRadioButton 对象，并使用该对象调用 add()方法添加单选按钮。

【例 13.12】　在应用程序窗体中定义一个单选按钮组。

```
JRadioButton jr1 = new JRadioButton();
JRadioButton jr2 = new JRadioButton();
JRadioButton jr3 = new JRadioButton();
ButtonGroup group = new ButtonGroup();
group.add(jr1);
group.add(jr2);
group.add(jr3);
```

从中可以看到，实质上单选按钮与提交按钮的用法基本类似，只是实例化单选按钮对象后需要将其添加至按钮组中。

13.6.3　复选框组件

复选框（JCheckBox）在 Swing 组件中使用也非常广泛，它具有一个方块图标，外加一段描述性文字。与单选按钮唯一不同的是复选框可以进行多选设置，每一个复选框都提供“选中”与“不选中”两种状态。复选框由 JCheckBox 类的对象表示，它同样继承于 AbstractButton 类，所以复选框组件的属性设置也来源于 AbstractButton 类。

JCheckBox 的常用构造方法如下：

☑　public JCheckBox()

☑　public JCheckBox(Icon icon,boolean checked)

☑　public JCheckBox(String text,boolean checked)

复选框与其他按钮设置类似，除了可以在初始化时设置图标之外还可以设置复选框的文字与是否被选中。

下面来看一个实例，在这个实例中笔者将滚动面板与复选框结合使用。

【例 13.13】　在项目中创建 CheckBoxTest 类，该类继承 JFrame 类成为窗体组件，在类中设置窗体使用 BorderLayout 布局管理器，为窗体添加多个复选框对象。（**实例位置：光盘\TM\sl\13\15**）

```
public class CheckBoxTest extends JFrame{
    ...//省略非关键代码
    public CheckBoxTest(){
        ...//省略非关键代码
        c.setLayout(new BorderLayout());
        c.add(panel1, BorderLayout.NORTH);
```

```
        final JScrollPane scrollPane = new JScrollPane(jt);
        panel1.add(scrollPane);
        c.add(panel2, BorderLayout.SOUTH);
        panel2.add(jc1);
        jc1.addActionListener(new ActionListener(){
            public void actionPerformed(ActionEvent e){
                jt.append("复选框 1 被选中\n");
            }
        });
        ...//省略其他复选框监听事件
    }
...//省略主方法
}
```

运行结果如图 13.15 所示。

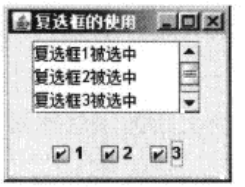


图 13.15　复选框的应用

13.7　列 表 组 件

视频讲解：光盘\TM\lx\13\列表组件.exe

Swing 中提供了下拉列表框与列表框两种列表组件。下拉列表框与列表框都是带有一系列项目的组件，用户可以从中选择需要的项目。列表框较下拉列表框更直观一些，它将所有的项目罗列在列表框中；但下拉列表框较列表框更为便捷和美观，它将所有的项目隐藏起来，当用户选用其中的项目时才会显现出来。

13.7.1　下拉列表框组件

1．JComboBox 类

初次使用 Swing 中的下拉列表框（JComboBox）时，会感觉到此类下拉列表框与 Windows 操作系统中的下拉列表框有一些相似，实质上两者并不完全相同，因为 Swing 中的下拉列表框不仅可以从中选择项目，同时也提供用户编辑项目中的内容。

下拉列表框是一个带条状的显示区，它具有下拉功能，在下拉列表框的右方存在一个倒三角形的按钮，当用户单击该按钮时，下拉列表框中的项目将会以列表形式显示出来。

Swing 中的下拉列表框使用 JComboBox 类对象来表示，它是 javax.swing.JComponent 类的子类。

它的常用构造方法如下：

- ☑ public JComboBox()
- ☑ public JComboBox(ComboBoxModel dataModel)
- ☑ public JComboBox(Object[] arrayData)
- ☑ public JComboBox(Vector vector)

在初始化下拉列表框时，可以同时指定下拉列表框的项目内容，也可以在程序中使用其他方法设置下拉列表框的内容，下拉列表框中的内容可以被封装在 ComboBoxModel 类型、数组或者 Vector 类型中。

2．JComboBox 模型

在开发程序中，将下拉列表框中的项目封装为 ComboBoxModel 的情况较多。ComboBoxModel 为接口，它代表一般模型，可以自定义一个类实现该接口，然后在初始化 JComboBox 对象时向上转型为 ComboBoxModel 接口类型，但是必须实现以下两个方法。

- ☑ public void setSelectedItem(Object item)
- ☑ public Object getSelectedItem()

其中，setSelectedItem()方法是设置下拉列表框的选中项，getSelectedItem()方法用于返回下拉列表框中选中项，有了这两个方法，就可以轻松地对下拉列表框中的项目进行操作。

自定义这个类除了实现该接口之外，还可以继承 AbstractListModel 类，在该类中也有两个操作下拉列表框的重要方法。

- ☑ getSize()：返回列表的长度。
- ☑ getElementAt(int index)：返回指定索引处的值。

【例 13.14】　在项目中创建 JComboBoxModelTest 类，使该类继承 JFrame 类成为窗体组件，在类中创建下拉列表框，并添加到窗体中。（**实例位置：光盘\TM\sl\13\16**）

```
public class JComboBoxModelTest extends JFrame{
    JComboBox jc=new JComboBox(new MyComboBox());
    JLabel jl=new JLabel("请选择证件:");
    public JComboBoxModelTest(){
        ...//省略非关键代码
        cp.setLayout(new FlowLayout());
        cp.add(jl);
        cp.add(jc);
    }
    ...//省略主方法
}
class MyComboBox extends AbstractListModel implements ComboBoxModel{
    String selecteditem=null;
    String[] test={"身份证","军人证","学生证","工作证"};
    public Object getElementAt(int index){                    //根据索引返回值
        return test[index];
    }
    public int getSize(){                                     //返回下拉列表框中项目的数目
```

```
        return test.length;
    }
    public void setSelectedItem(Object item){                    //设置下拉列表框项目
        selecteditem=(String)item;
    }
    public Object getSelectedItem(){                             //获取下拉列表框中的项目
        return selecteditem;
    }
    ...//省略非关键代码
}
```

运行结果如图 13.16 所示。

图 13.16　选择证件类型

13.7.2　列表框组件

列表框（JList）与下拉列表框的区别不仅表现在外观上，当激活下拉列表框时，会出现下拉列表框中的内容，但列表框只是在窗体上占据固定的大小，如果要使列表框具有滚动效果，可以将列表框放入滚动面板中。用户在选择列表框中的某一项时，按住 Shift 键并选择列表框中的其他项目，其他项目也将被选中；也可以按住 Ctrl 键并单击列表框中的项目，这样列表框中的项目处于非选择状态。

Swing 中使用 JList 类对象来表示列表框，常用的构造方法如下：

- ☑ public void JList()
- ☑ public void JList(Object[] listData)
- ☑ public void JList(Vector listData)
- ☑ public void JList(ListModel dataModel)

在上述构造方法中，存在一个没有参数的构造方法，可以在初始化列表框后使用 setListData()方法对列表框进行设置，同时也可以在初始化的过程中对列表框中的项目进行设置。设置的方式有 3 种类型，包括数组、Vector 类型和 ListModel 模型。

当使用数组作为构造方法的参数时，首先需要创建列表项目的数组，然后再利用构造方法来初始化列表框。

【例 13.15】　使用数组作为初始化列表框的参数。

```
String[] contents={"列表 1","列表 2","列表 3","列表 4"};
JList jl=new JList(contents);
```

如果使用上述构造方法中的第 3 个构造方法，将 Vector 类型的数据作为初始化 JList 组件的参数，通常可以使用如下代码。

【例 13.16】　使用 Vector 类型数据作为初始化列表框的参数。

```
Vector contents=new Vector();
JList jl=new JList(contents);
```

```
contents.add("列表 1");
contents.add("列表 2");
contents.add("列表 3");
contents.add("列表 4");
```

如果使用 ListModel 模型为参数，需要创建 ListModel 对象。ListModel 是 Swing 包中的一个接口，它提供了获取列表框属性的方法。但是在通常情况下，为了使用户不完全实现 ListModel 接口中的方法，通常自定义一个类继承实现该接口的抽象类 AbstractListModel。在这个类中提供了 getElementAt()与 getSize()方法，其中 getElementAt()方法代表根据项目的索引获取列表框中的值，而 getSize()方法用于获取列表框中项目的个数。

【例 13.17】 在项目中创建 JListTest 类，使该类继承 JFrame 类成为窗体组件，在该类中创建列表框，并添加到窗体中。（**实例位置：光盘\TM\sl\13\17**）

```
public class JListTest extends JFrame{
    public JListTest(){
        Container cp=getContentPane();
        cp.setLayout(null);
        JList jl=new JList(new MyListModel());
        JScrollPane js=new JScrollPane(jl);
        js.setBounds(10, 10, 100, 100);
        cp.add(js);
        ...//非关键代码省略
    }
    ...//主方法省略
}
class MyListModel extends AbstractListModel{                    //继承抽象类 AbstractListModel
    //设置列表框内容
    private String[] contents={"列表 1","列表 2","列表 3","列表 4","列表 5","列表 6"};
    public Object getElementAt(int x){                          //重写 getElementAt()方法
        if(x<contents.length)
            return contents[x++];
        else
            return null;
    }
    public int getSize() {                                      //重写 getSize()方法
        return contents.length;
    }
}
```

运行结果如图 13.17 所示。

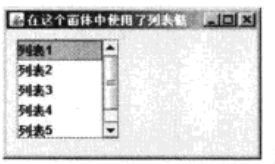


图 13.17　列表框的使用

13.8 文本组件

视频讲解：光盘\TM\lx\13\文本组件.exe

文本组件在现实项目开发中使用最为广泛，尤其文本框与密码框组件，通过文本组件可以很轻松地处理单行文字、多行文字、口令字段。

13.8.1 文本框组件

文本框（JTextField）用来显示或编辑一个单行文本，在 Swing 中通过 javax.swing.JTextField 类对象创建，该类继承了 javax.swing.text.JTextComponent 类。创建文本框常用的构造方法如下：

- ☑ public JTextField()
- ☑ public JTextField(String text)
- ☑ public JTextField(int fieldwidth)
- ☑ public JTextField(String text,int fieldwidth)
- ☑ public JTextField(Document docModel,String text,int fieldWidth)

在上述构造方法中可以看出，定义 JTextField 组件很简单，可以在初始化文本框时设置文本框的默认文字、文本框的长度等。

下面来看一个关于文本框的实例。

【例 13.18】 在项目中创建 JTextFieldTest 类，使该类继承 JFrame 类成为窗体组件，在该类中创建文本框和按钮组件，并添加到窗体中。（**实例位置：光盘\TM\sl\13\18**）

```
public class JTextFieldTest extends JFrame{
    public JTextFieldTest(){
        ...//省略非关键代码
        final JTextField jt=new JTextField("aaa",20);
        final JButton jb=new JButton("清除");
        ...//省略非关键代码
        jt.addActionListener(new ActionListener(){                    //为文本框添加事件
            public void actionPerformed(ActionEvent arg0) {
                jt.setText("触发事件");                               //设置文本框中的值
            }
        });
        jb.addActionListener(new ActionListener(){                    //为按钮添加事件
            public void actionPerformed(ActionEvent arg0) {
                jt.setText("");                                       //将文本框置空
                jt.requestFocus();                                    //焦点回到文本框
            }
        });
    }
    ...//省略主方法
}
```

在本实例的窗体中主要设置了一个文本框和一个按钮，然后分别为文本框和按钮设置事件，当用户将光标焦点落于文本框中并按 Enter 键时，文本框将进行 actionPerformed()方法中设置的操作。同时还为按钮添加了相应的事件，当用户单击“清除”按钮时，文本框中的字符串将被清除。运行结果如图 13.18 所示。

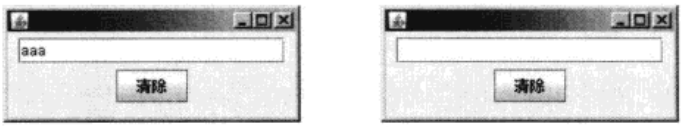


图 13.18　按钮控制文本框中的值

13.8.2　密码框组件

密码框（JPasswordField）与文本框的定义与用法类似，唯一不同的是密码使用户输入的字符串以某种符号进行加密。密码框对象是通过 javax.swing.JPasswordField 类来创建，JPasswordField 类的构造方法与 JTextField 类的构造方法类似。常用的构造方法如下：

- ☑ public JPasswordField()
- ☑ public JPasswordFiled(String text)
- ☑ public JPasswordField(int fieldwidth)
- ☑ public JPasswordField(String text,int fieldwidth)
- ☑ public JPasswordField(Document docModel,String text,int fieldWidth)

【例 13.19】　在程序中定义密码框。

```
JPasswordField jp=new JPasswordField();
jp.setEchoChar('#');                                    //设置回显字符
```

在 JPasswordField 类中提供了一个 setEchoChar()方法，可以改变密码框的回显字符。

13.8.3　文本域组件

Swing 中任何一个文本域（JTextArea）都是 JTextArea 类型的对象。JTextArea 常用的构造方法如下：

- ☑ public JTextArea()
- ☑ public JTextArea(String text)
- ☑ public JTextArea(int rows,int columns)
- ☑ public JTextArea(Document doc)
- ☑ public JTextArea(Document doc,String Text,int rows,int columns)

在上述构造方法中，可以在初始化文本域时提供默认文本以及文本域的长与宽。

【例 13.20】　在项目中创建 JTextAreaTest 类，使该类继承 JFrame 类成为窗体组件，在该类中创建文本框和按钮组件，并添加到窗体中。（**实例位置：光盘\TM\sl\13\19**）

```
import java.awt.Container;
import javax.swing.*;
```

```
public class JTextAreaTest extends JFrame{
    public JTextAreaTest(){
        ...//非关键性代码省略
        Container cp=getContentPane();
        JTextArea jt=new JTextArea("文本域",6,6);
        jt.setLineWrap(true);                                    //使文本域可以自动换行
        cp.add(jt);
    }
    ...//省略主方法
}
```

运行结果如图 13.19 所示。

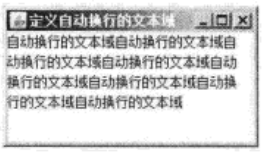


图 13.19　自动换行的文本域

13.8.4　范例 4：给文本域设置背景图片

在软件的美化过程中，比较常用的方式之一是使用背景图片。在 Swing 中，除了可以给框架增加背景图片外，还可以给文本域设置背景图片。运行结果如图 13.20 所示。（**实例位置：光盘\TM\sl\13\20**）

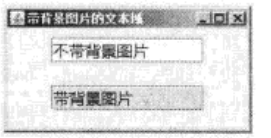


图 13.20　给文本域设置背景图片

（1）编写 BackgroundJTextFieldTest 类，该类继承了 JFrame。在框架中包括了两个文本域。

（2）编写 BackgroundJTextField 类，该类继承了 JTextField。在该类的构造方法中利用传递的 File 参数获得一个图像，以此来作为文本框的背景图片。在 paintComponent()方法中，将此背景图片绘制到文本域中。代码如下：

```
public class BackgroundJTextField extends JTextField {
    private static final long serialVersionUID = 5810044732894008630L;
    private TexturePaint paint;
    public BackgroundJTextField(File file) {
        super();
        try {
            BufferedImage image = ImageIO.read(file);            //获得缓冲图片
            Rectangle rectangle = new Rectangle(0, 0, image.getWidth(), image.getHeight());
            paint = new TexturePaint(image, rectangle);          //创建 TexturePaint 对象
            setOpaque(false);                                    //将文本域设置成透明的
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    protected void paintComponent(Graphics g) {
        Graphics2D g2 = (Graphics2D) g;                          //将 g 转型为 Graphics2D
        g2.setPaint(paint);                                      //设置新的颜色模式
        g.fillRect(0, 0, getWidth(), getHeight());               //让图片充满整个区域
```

```
        super.paintComponent(g);                                    //调用父类的同名方法
    }
}
```

13.8.5　范例 5：给文本区设置背景图片

在软件的美化过程中，比较常用的方式之一是使用背景图片。在 Swing 中，除了可以给框架增加背景图片外，还可以给文本区设置背景图片。运行结果如图 13.21 所示。（**实例位置：光盘\TM\sl\13\21**）

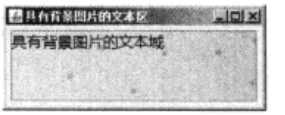


图 13.21　给文本区设置背景图片

（1）编写 BackgroundJTextAreaTest 类，该类继承 JFrame。在框架中包括一个自定义文本区。

（2）编写 BackgroundJTextArea 类，该类继承了 JTextArea。在其构造方法中利用给定的路径获得图片，并将文本区设置成透明的。在 paint()方法中，将图片绘制到文本区中。代码如下：

```
public class BackgroundJTextArea extends JTextArea {
    private static final long serialVersionUID = -4157782271632761973L;
    private Image image;
    public BackgroundJTextArea(String path) {
        ImageIcon imageIcon = new ImageIcon(path);              //获得图片图标
        image = imageIcon.getImage();                           //获得图片
        setOpaque(false);                                       //将文本区设置成透明的
    }
    public void paint(Graphics g) {
        g.drawImage(image, 0, 0, this);                         //绘制图片
        super.paint(g);
    }
}
```

13.9　经 典 范 例

前面学习了如何创建 Swing 界面，下面利用前面所学知识设计几个在实际应用中会遇到的例子。

13.9.1　经典范例 1：设置窗体标题栏图标

视频讲解：光盘\TM\lx\13\设置窗体标题栏图标.exe

窗体的标题栏图标也称为窗体的图标，当窗体显示时在左上角标题栏位置会显示这个图标与窗体标题信息，这用于区分不动窗体或标注窗体，本实例通过 Java 语言实现窗体图标的设置，单击任意一

个按钮，就会把窗体图标更换为按钮的图标。程序运行后，除窗体之外，在任务栏也会显示窗体的图标。运行结果如图 13.22 所示。（**实例位置：光盘\TM\sl\13\22**）

图 13.22　设置窗体标题栏图标

（1）在项目中新建窗体类 FrameIcon。在窗体中设置背景，并添加 4 个更改窗体图标的按钮。

（2）编写所有按钮的事件处理方法，在用户单击不同按钮时，该方法可以判断事件源并获取相对图标文件的 URL 路径，然后通过该路径创建图片对象设置给窗体的 iconImage 属性。代码如下：

```
protected void do_button_actionPerformed(ActionEvent e) {
    String resource = "";                                   //定义图标文件名称变量
    if (e.getSource() == button1)                           //确定用户单击的按钮
        resource = "icon1.png";                             //确定按钮对应的图标文件
    if (e.getSource() == button2)
        resource = "icon2.png";
    if (e.getSource() == button3)
        resource = "icon3.png";
    if (e.getSource() == button4)
        resource = "icon4.png";
    URL url = getClass().getResource(resource);             //获取图标文件路径
    setIconImage(Toolkit.getDefaultToolkit().getImage(url));  //设置窗体的图标
}
```

13.9.2　经典范例 2：随机更换窗体背景

视频讲解：光盘\TM\lx\13\随机更换窗体背景.exe

在一些管理软件中实现随机更换主界面背景可以增加软件的人性化程度，使用户心情愉悦。本范例实现随机更换主界面的功能，每次窗体恢复显示时主界面背景图片都会随机更换。运行结果如图 13.23 所示。（**实例位置：光盘\TM\sl\13\23**）

图 13.23　随机更换窗体背景

（1）在项目中新建窗体类 RandomBackgroundImage。在窗体中添加自定义的支持背景的面板控件。

（2）编写初始化背景图片数组的 initPhotoArray()方法，在该方法中创建图片数组，然后通过 for 循环初始化数组中的元素，每个元素赋值为一个图片对象。代码如下：

```
private void initPhotoArray() {
    images = new Image[6];                                          //初始化背景图片数组
    String photoPath = "";
    for (int i = 0; i < images.length; i++) {                       //遍历数组并初始化所有元素
        photoPath = "/com/img/photo" + (i + 1) + ".jpg";            //生成文件名
        images[i] = getToolkit()
                .getImage(getClass().getResource(photoPath));       //初始化数组元素
    }
}
```

（3）编写窗体激活事件的处理方法，当窗体刚刚被显示，或者从最小化恢复到显示状态时，都会触发这个窗体激活事件，在事件处理方法中，首先生成随机数，然后用这个随机数作为图片数组的下标索引获取一个图片设置给支持背景的面板控件，最后重新绘制窗体界面。代码如下：

```
private void initPhotoArray() {
    images = new Image[6];                                          //初始化背景图片数组
    String photoPath = "";
    for (int i = 0; i < images.length; i++) {                       //遍历数组并初始化所有元素
        photoPath = "/com/img/photo" + (i + 1) + ".jpg";            //生成文件名
        images[i] = getToolkit()
                .getImage(getClass().getResource(photoPath));       //初始化数组元素
    }
}
```

13.10　本 章 小 结

本章对 Swing 程序设计中常用的组件进行了讲解，包括常用窗体、容器的常用布局、常用面板组件的使用、菜单的创建以及工具栏的创建，学完本章读者应该可以实现应用程序界面的设计。

13.11　实 战 练 习

1．尝试开发一个窗体，如图 13.24 所示。（**答案位置：光盘\TM\sl\13\24**）

2．尝试创建一个窗体，选择合适的布局管理器，并在窗体中设置一个下拉列表框，初始状态下拉列表框中没有项目，设置一个按钮，为按钮设置动作事件监听器，当用户单击该按钮时，下拉列表框中相应添加数组中的内容。（**答案位置：光盘\TM\sl\13\25**）

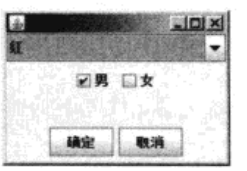


图 13.24　要开发的窗体

3．尝试开发一个登录窗体，包括用户名、密码以及提交按钮和重置按钮，当用户输入用户名 mr、密码 mrsoft 时，弹出登录成功提示对话框。（**答案位置：光盘\TM\sl\13\26**）

第3篇

高级篇

- 第 14 章　高级事件处理
- 第 15 章　多线程
- 第 16 章　网络通信
- 第 17 章　JDBC 操作数据库
- 第 18 章　Swing 高级组件
- 第 19 章　高级布局管理器
- 第 20 章　AWT 绘图技术

本篇介绍了高级事件处理、多线程、网络通信、JDBC 操作数据库、Swing 高级组件、高级布局管理器、AWT 绘图技术等内容。学习完本篇后，能够开发高级的桌面应用程序、多媒体程序和打印程序等。

PDG

第14章

高级事件处理

（视频讲解：30分钟）

本章将讲解一些常用高级事件的处理方法，包括键盘事件、鼠标事件、窗体事件、选项事件和表格模型事件，通过捕获这些事件并对其进行处理，可以更进一步控制程序的流程，保证每一步操作的合法性，实现一些更人性化的性能。例如，通过捕获键盘事件验证输入数据的合法性，通过捕获表格模型事件实现自动计算表格某一列的和等。

通过阅读本章，您可以：

- 学会处理键盘事件
- 学会处理鼠标事件
- 学会处理窗体焦点变化、状态变化等事件
- 学会处理选项事件
- 学会处理表格模型事件

PDG

14.1 键盘事件

视频讲解：光盘\TM\lx\14\键盘事件.exe

当向文本框中输入内容时，将发出键盘事件（KeyEvent）。KeyEvent 类负责捕获键盘事件，可以通过为组件添加实现了 KeyListener 接口的监听器类来处理相应的键盘事件。

KeyListener 接口共有 3 个抽象方法，分别在发生击键事件、按键被按下和释放时触发。KeyListener 接口的具体定义如下：

```
public interface KeyListener extends EventListener {
    public void keyTyped(KeyEvent e);                    //发生击键事件时触发
    public void keyPressed(KeyEvent e);                  //按键被按下时触发
    public void keyReleased(KeyEvent e);                 //按键被释放时触发
}
```

在每个抽象方法中均传入了 KeyEvent 类的对象，KeyEvent 类中比较常用的方法如表 14.1 所示。

表 14.1 KeyEvent 类中的常用方法

方　法	功能简介
getSource()	用来获得触发此次事件的组件对象，返回值为 Object 类型
getKeyChar()	用来获得与此事件中的键相关联的字符
getKeyCode()	用来获得与此事件中的键相关联的整数 keyCode
getKeyText(int keyCode)	用来获得描述 keyCode 的标签，如 A、F1 和 HOME 等
isActionKey()	用来查看此事件中的键是否为“动作”键
isControlDown()	用来查看 Ctrl 键在此次事件中是否被按下，当返回 true 时表示被按下
isAltDown()	用来查看 Alt 键在此次事件中是否被按下，当返回 true 时表示被按下
isShiftDown()	用来查看 Shift 键在此次事件中是否被按下，当返回 true 时表示被按下

技巧 在 KeyEvent 类中以 VK_开头的静态常量代表各个按键的 keyCode，可以通过这些静态常量判断事件中的按键，以及获得按键的标签。

【例 14.1】 演示捕获和处理键盘事件的方法，尤其是键盘事件监听器接口 KeyListener 中各个方法的使用方法。（实例位置：光盘\TM\sl\14\1）

```
final JLabel label = new JLabel();
label.setText("备注：");
getContentPane().add(label, BorderLayout.WEST);
final JScrollPane scrollPane = new JScrollPane();
getContentPane().add(scrollPane, BorderLayout.CENTER);
JTextArea textArea = new JTextArea();
textArea.addKeyListener(new KeyListener() {
    public void keyPressed(KeyEvent e) {                 //按键被按下时触发
```

```
            String keyText = KeyEvent.getKeyText(e.getKeyCode());      //获得描述 keyCode 的标签
            if (e.isActionKey()) {                                     //判断按下的是否为动作键
                System.out.println("您按下的是动作键 “" + keyText + "” ");
            } else {
                System.out.print("您按下的是非动作键 “" + keyText + "” ");
                int keyCode = e.getKeyCode();                          //获得与此事件中的键相关联的字符
                switch (keyCode) {
                case KeyEvent.VK_CONTROL:                              //判断按下的是否为 Ctrl 键
                    System.out.print("，Ctrl 键被按下");
                    break;
                case KeyEvent.VK_ALT:                                  //判断按下的是否为 Alt 键
                    System.out.print("，Alt 键被按下");
                    break;
                case KeyEvent.VK_SHIFT:                                //判断按下的是否为 Shift 键
                    System.out.print("，Shift 键被按下");
                    break;
                }
                System.out.println();
            }
        }
        public void keyTyped(KeyEvent e) {                             //发生击键事件时触发
            System.out.println("此次输入的是 “" + e.getKeyChar() + "” "); //获得输入的字符
        }
        public void keyReleased(KeyEvent e) {                          //按键被释放时触发
            String keyText = KeyEvent.getKeyText(e.getKeyCode());      //获得描述 keyCode 的标签
            System.out.println("您释放的是 “" + keyText + "” 键");
            System.out.println();
        }
    });
    textArea.setLineWrap(true);
    textArea.setRows(3);
    textArea.setColumns(15);
    scrollPane.setViewportView(textArea);
```

运行本实例，首先输入小写字母“m”，然后输入一个空格，接下来输入大写字母“M”，再按 Shift 键，最后按 F5 键，运行结果如图 14.1 所示。

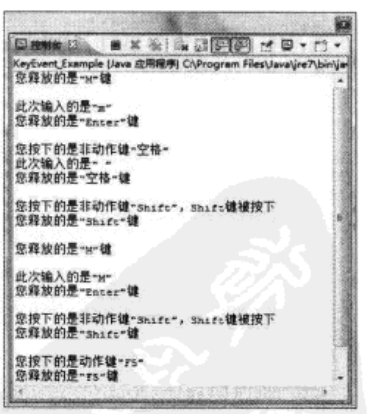


图 14.1　键盘事件

14.2 鼠标事件

视频讲解：光盘\TM\lx\14\鼠标事件.exe

所有组件都能发出鼠标事件（MouseEvent），MouseEvent 类负责捕获鼠标事件，可以通过为组件添加实现了 MouseListener 接口的监听器类来处理相应的鼠标事件。

MouseListener 接口共有 5 个抽象方法，分别在光标移入或移出组件时、鼠标按键被按下或释放时和发生单击事件时触发。所谓单击事件，就是按键被按下并释放。需要注意的是，如果按键是在移出组件之后才被释放，则不会触发单击事件。MouseListener 接口的具体定义如下：

```
public interface MouseListener extends EventListener {
    public void mouseEntered(MouseEvent e);          //光标移入组件时触发
    public void mousePressed(MouseEvent e);          //鼠标按键被按下时触发
    public void mouseReleased(MouseEvent e);         //鼠标按键被释放时触发
    public void mouseClicked(MouseEvent e);          //发生单击事件时触发
    public void mouseExited(MouseEvent e);           //光标移出组件时触发
}
```

在每个抽象方法中均传入了 MouseEvent 类的对象，MouseEvent 类中比较常用的方法如表 14.2 所示。

表 14.2 MouseEvent 类中的常用方法

方 法	功 能 简 介
getSource()	用来获得触发此次事件的组件对象，返回值为 Object 类型
getButton()	用来获得代表触发此次按下、释放或单击事件的按键的 int 型值
getClickCount()	用来获得单击按键的次数

当需要判断触发此次事件的按键时，可以通过表 14.3 中的静态常量判断由 getButton()方法返回的 int 型值代表的键。

表 14.3 MouseEvent 类中代表鼠标按键的静态常量

静 态 常 量	常 量 值	代 表 的 键
BUTTON1	1	代表鼠标左键
BUTTON2	2	代表鼠标滚轮
BUTTON3	3	代表鼠标右键

【例 14.2】 演示捕获和处理鼠标事件的方法，尤其是鼠标事件监听器接口 MouseListener 中各个方法的使用方法。（**实例位置：光盘\TM\sl\14\2**）

```
final JLabel label = new JLabel();
label.addMouseListener(new MouseListener() {
    public void mouseEntered(MouseEvent e) {          //光标移入组件时触发
        System.out.println("光标移入组件");
    }
```

```
    public void mousePressed(MouseEvent e) {                    //鼠标按键被按下时触发
        System.out.print("鼠标按键被按下，");
        int i = e.getButton();                                  //通过该值可以判断按下的是哪个键
        if (i == MouseEvent.BUTTON1)
            System.out.println("按下的是鼠标左键");
        if (i == MouseEvent.BUTTON2)
            System.out.println("按下的是鼠标滚轮");
        if (i == MouseEvent.BUTTON3)
            System.out.println("按下的是鼠标右键");
    }
    public void mouseReleased(MouseEvent e) {                   //鼠标按键被释放时触发
        System.out.print("鼠标按键被释放，");
        int i = e.getButton();                                  //通过该值可以判断释放的是哪个键
        if (i == MouseEvent.BUTTON1)
            System.out.println("释放的是鼠标左键");
        if (i == MouseEvent.BUTTON2)
            System.out.println("释放的是鼠标滚轮");
        if (i == MouseEvent.BUTTON3)
            System.out.println("释放的是鼠标右键");
    }
    public void mouseClicked(MouseEvent e) {                    //发生单击事件时触发
        System.out.print("单击了鼠标按键，");
        int i = e.getButton();                                  //通过该值可以判断单击的是哪个键
        if (i == MouseEvent.BUTTON1)
            System.out.print("单击的是鼠标左键，");
        if (i == MouseEvent.BUTTON2)
            System.out.print("单击的是鼠标滚轮，");
        if (i == MouseEvent.BUTTON3)
            System.out.print("单击的是鼠标右键，");
        int clickCount = e.getClickCount();
        System.out.println("单击次数为" + clickCount + "下");
    }
    public void mouseExited(MouseEvent e) {                     //光标移出组件时触发
        System.out.println("光标移出组件");
    }
});
```

运行本实例，首先将光标移入窗体，然后单击鼠标左键，接着双击鼠标右键，最后将光标移出窗体，运行结果如图 14.2 所示。

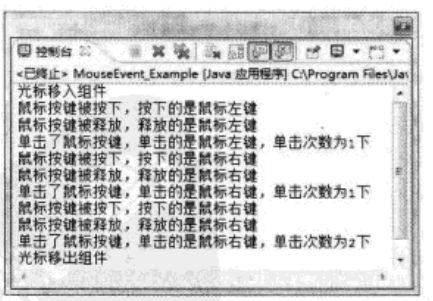


图 14.2　鼠标事件

注意

从图 14.2 中可以发现，当双击鼠标时，第一次单击鼠标将触发一次单击事件。

14.3 窗 体 事 件

视频讲解：光盘\TM\lx\14\窗体事件.exe

在捕获窗体事件（WindowEvent）时，可以通过 3 个事件监听器接口来实现，分别为 WindowFocusListener、WindowStateListener 和 WindowListener。本节将深入学习这 3 种事件监听器的使用方法，主要是各自捕获的事件类型和各个抽象方法的触发条件。

14.3.1 捕获窗体焦点变化事件

需要捕获窗体焦点发生变化的事件时，即窗体获得或失去焦点的事件时，可以通过实现了 WindowFocusListener 接口的事件监听器完成。WindowFocusListener 接口的具体定义如下：

```
public interface WindowFocusListener extends EventListener {
    public void windowGainedFocus(WindowEvent e);          //窗体获得焦点时触发
    public void windowLostFocus(WindowEvent e);            //窗体失去焦点时触发
}
```

通过捕获窗体获得或失去焦点的事件，可以进行一些相关操作，例如当窗体重新获得焦点时，令所有组件均恢复为默认设置。

【例 14.3】 演示捕获和处理窗体焦点变化事件的方法，尤其是窗体焦点事件监听器接口 WindowFocusListener 中各个方法的使用方法。（**实例位置：光盘\TM\sl\14\3**）

```
public class WindowFocusListener_Example extends JFrame {
    public static void main(String args[]) {
        WindowFocusListener_Example frame = new WindowFocusListener_Example();
        frame.setVisible(true);
    }
    public WindowFocusListener_Example() {
        super();
        addWindowFocusListener(new MyWindowFocusListener());     //为窗体添加焦点事件监听器
        setTitle("捕获窗体焦点事件");
        setBounds(100, 100, 500, 375);
        setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
    }
    private class MyWindowFocusListener implements WindowFocusListener {
        public void windowGainedFocus(WindowEvent e) {          //窗口获得焦点时触发
            System.out.println("窗口获得了焦点！");
```

```
        }
        public void windowLostFocus(WindowEvent e) {                    //窗口失去焦点时触发
            System.out.println("窗口失去了焦点！");
        }
    }
}
```

运行结果如图 14.3 所示。

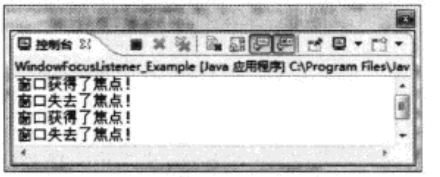


图 14.3　捕获窗体焦点事件监听器

14.3.2　捕获窗体状态变化事件

需要捕获窗体状态发生变化的事件时，即窗体由正常化变为图标化、由最大化变为正常化等事件时，可以通过实现 WindowStateListener 接口的事件监听器来完成。WindowStateListener 接口的具体定义如下：

```
public interface WindowStateListener extends EventListener {
    public void windowStateChanged(WindowEvent e);                    //窗体状态发生变化时触发
}
```

在抽象方法 windowStateChanged()中传入了 WindowEvent 类的对象。WindowEvent 类中有如下两个常用方法，用来获得窗体的状态，它们均返回一个代表窗体状态的 int 型值。

- ☑ getNewState()：用来获得窗体现在的状态。
- ☑ getOldState()：用来获得窗体以前的状态。

可以通过 Frame 类中的静态常量判断返回的 int 型值具体代表什么状态，这些静态常量如表 14.4 所示。

表 14.4　Frame 类中代表窗体状态的静态常量

静 态 常 量	常 量 值	代 表 的 键
NORMAL	0	代表窗体处于“正常化”状态
ICONIFIED	1	代表窗体处于“图标化”状态
MAXIMIZED_BOTH	6	代表窗体处于“最大化”状态

【例 14.4】 演示捕获和处理窗体状态变化事件的方法，尤其是窗体状态变化事件监听器接口 WindowStateListener 中各个方法的使用方法。（**实例位置：光盘\TM\sl\14\4**）

```
public class WindowStateListener_Example extends JFrame {
    public static void main(String args[]) {
        WindowStateListener_Example frame = new WindowStateListener_Example();
        frame.setVisible(true);
    }
    public WindowStateListener_Example() {
        super();
        addWindowStateListener(new MyWindowStateListener());  //为窗体添加状态事件监听器
        setTitle("捕获窗体状态事件");
        setBounds(100, 100, 500, 375);
```

```
        setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
    }
    private class MyWindowStateListener implements WindowStateListener {
        public void windowStateChanged(WindowEvent e) {
            int oldState = e.getOldState();                    //获得窗体以前的状态
            int newState = e.getNewState();                    //获得窗体现在的状态
            String from = "";                                  //标识窗体以前状态的中文字符串
            String to = "";                                    //标识窗体现在状态的中文字符串
            switch (oldState) {                                //判断窗体以前的状态
            case Frame.NORMAL:                                 //窗体处于正常化
                from = "正常化";
                break;
            case Frame.MAXIMIZED_BOTH:                         //窗体处于最大化
                from = "最大化";
                break;
            default:                                           //窗体处于图标化
                from = "图标化";
            }
            switch (newState) {                                //判断窗体现在的状态
            case Frame.NORMAL:                                 //窗体处于正常化
                to = "正常化";
                break;
            case Frame.MAXIMIZED_BOTH:                         //窗体处于最大化
                to = "最大化";
                break;
            default:                                           //窗体处于图标化
                to = "图标化";
            }
            System.out.println(from + "——>" + to);
        }
    }
}
```

运行本实例，首先将窗体图标化后再恢复正常化，然后将窗体最大化后再图标化，最后将窗体最大化后再恢复正常化，在控制台将得到如图 14.4 所示的信息。

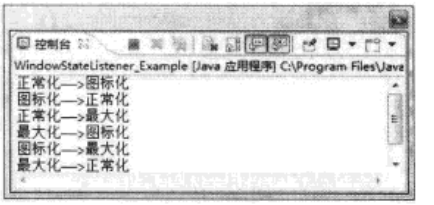


图 14.4　捕获窗体状态变化事件

14.3.3　捕获其他窗体事件

需要捕获其他与窗体有关的事件时，例如捕获窗体被打开、将要被关闭、已经被关闭等事件时，可以通过实现了 WindowListener 接口的事件监听器完成。WindowListener 接口的具体定义如下：

```
public interface WindowListener extends EventListener {
    public void windowActivated(WindowEvent e);                //窗体被激活时触发
    public void windowOpened(WindowEvent e);                   //窗体被打开时触发
    public void windowIconified(WindowEvent e);                //窗体被图标化时触发
```

```
    public void windowDeiconified(WindowEvent e);                    //窗体被非图标化时触发
    public void windowClosing(WindowEvent e);                        //窗体将要被关闭时触发
    public void windowDeactivated(WindowEvent e);                    //窗体不再处于激活状态时触发
    public void windowClosed(WindowEvent e);                         //窗体已经被关闭时触发
}
```

通过捕获窗体将要被关闭等事件，可以进行一些相关操作，例如当窗体将要被关闭时，询问是否保存未保存的设置等。

【例 14.5】　演示捕获和处理其他窗体事件的方法，尤其是事件监听器接口 WindowListener 中各个方法的使用方法。（实例位置：光盘\TM\sl\14\5）

```
public class WindowListener_Example extends JFrame {
    public static void main(String args[]) {
        WindowListener_Example frame = new WindowListener_Example();
        frame.setVisible(true);
    }
    public WindowListener_Example() {
        super();
        addWindowListener(new MyWindowListener());                   //为窗体添加其他事件监听器
        setTitle("捕获其他窗体事件");
        setBounds(100, 100, 500, 375);
        setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
    }
    private class MyWindowListener implements WindowListener {
        public void windowActivated(WindowEvent e) {                 //窗体被激活时触发
            System.out.println("窗口被激活！");
        }
        public void windowOpened(WindowEvent e) {                    //窗体被打开时触发
            System.out.println("窗口被打开！");
        }
        public void windowIconified(WindowEvent e) {                 //窗体被图标化时触发
            System.out.println("窗口被图标化！");
        }
        public void windowDeiconified(WindowEvent e) {               //窗体被非图标化时触发
            System.out.println("窗口被非图标化！");
        }
        public void windowClosing(WindowEvent e) {                   //窗体将要被关闭时触发
            System.out.println("窗口将要被关闭！");
        }
        public void windowDeactivated(WindowEvent e) {               //窗体不再处于激活状态时触发
            System.out.println("窗口不再处于激活状态！");
        }
        public void windowClosed(WindowEvent e) {                    //窗体已经被关闭时触发
            System.out.println("窗口已经被关闭！");
        }
    }
}
```

运行结果如图 14.5 所示。

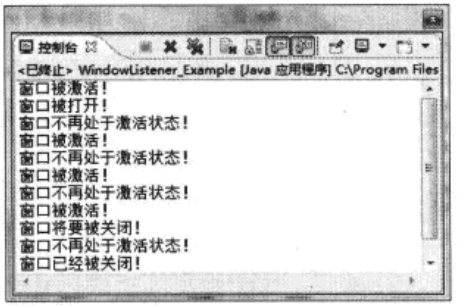


图 14.5　捕获其他窗体事件

14.4　选 项 事 件

视频讲解：光盘\TM\lx\14\选项事件.exe

当修改下拉菜单中的选中项时，将发出选项事件（ItemEvent）。ItemEvent 类负责捕获选项事件，可以通过为组件添加实现了 ItemListener 接口的监听器类来处理相应的选项事件。

ItemListener 接口只有一个抽象方法，在修改一次下拉菜单选中项的过程中，该方法将被触发两次，一次是由取消原来选中项的选中状态触发的，另一次是由选中新选项触发的。ItemListener 接口的具体定义如下：

```
public interface ItemListener extends EventListener {
    void itemStateChanged(ItemEvent e);
}
```

在抽象方法 itemStateChanged()中传入了 ItemEvent 类的对象。ItemEvent 类中有如下两个常用方法。

（1）getItem()

用来获得触发此次事件的选项，该方法的返回值为 Object 型。

（2）getStateChange()

用来获得此次事件的类型，即是由取消原来选中项的选中状态触发的，还是由选中新选项触发的。getStateChange()方法将返回一个 int 型值，可以通过 ItemEvent 类中的如下静态常量判断此次事件的具体类型。

- ☑ SELECTED：如果返回值等于该静态常量，说明此次事件是由选中新选项触发的。
- ☑ DESELECTED：如果返回值等于该静态常量，说明此次事件是由取消原来选中项的选中状态触发的。

通过捕获选项事件，可以进行一些相关操作，如同步处理其他下拉菜单的可选项。

【例 14.6】　演示捕获和处理选项事件的方法，尤其是事件监听器接口 ItemListener 中各个方法的使用方法。（**实例位置：光盘\TM\sl\14\6**）

```
JComboBox comboBox = new JComboBox();                    //创建一个下拉菜单
for (int i = 1; i < 6; i++) {                            //通过循环添加选项
    comboBox.addItem("选项" + i);
```

```
}
comboBox.addItemListener(new ItemListener() {                          //添加选项事件监听器
    public void itemStateChanged(ItemEvent e) {
        int stateChange = e.getStateChange();                          //获得事件类型
        String item = e.getItem().toString();                          //获得触发此次事件的选项
        if (stateChange == ItemEvent.SELECTED) {                       //查看是否由选中选项触发
            System.out.println("此次事件由　　选中　选项 “" + item + "” 触发！");
        } else if (stateChange == ItemEvent.DESELECTED) {              //查看是否由取消选中选项触发
            System.out.println("此次事件由　取消选中　选项 “" + item + "” 触发！");
        } else {                                                       //由其他原因触发
            System.out.println("此次事件由其他原因触发！");
        }
    }
});
```

运行结果如图 14.6 所示。

首先将选中项由“选项 1”改为“选项 2”；然后将选中项由“选项 2”改为“选项 2”；最后将选中项由“选项 2”改为“选项 5”，将得到如图 14.7 所示的信息。

图 14.6　运行结果

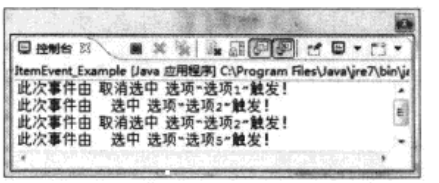


图 14.7　选项事件

注意　当选中项未发生变化时，并不会触发选项事件，例如在将选中项由“选项 2”改为“选项 2”时，在控制台并未输出信息。

14.5　表格模型事件

视频讲解：光盘\TM\lx\14\表格模型事件.exe

当向表格模型中添加行时，或者是修改或删除表格模型中的现有行时，将发出表格模型事件（TableModelEvent）。TableModelEvent 类负责捕获表格模型事件，可以通过为组件添加实现了 TableModelListener 接口的监听器类来处理相应的表格模型事件。

TableModelListener 接口只有一个抽象方法，当向表格模型中添加行时，或者是修改或删除表格模型中的现有行时，该方法将被触发。TableModelListener 接口的具体定义如下：

```
public interface TableModelListener extends java.util.EventListener {
    public void tableChanged(TableModelEvent e);
}
```

在抽象方法 tableChanged()中传入了 TableModelEvent 类的对象，TableModelEvent 类中比较常用的方法如表 14.5 所示。

表 14.5　TableModelEvent 类中的常用方法

方　法	功 能 简 介
getType()	获得此次事件的类型
getFirstRow()	获得触发此次事件的表格行的最小索引值
getLastRow()	获得触发此次事件的表格行的最大索引值
getColumn()	如果事件类型为 UPDATE，获得触发此次事件的表格列的索引值；否则将返回-1

getType()方法将返回一个 int 型值，可以通过 TableModelEvent 类中的如下静态常量判断此次事件的具体类型。

- ☑ INSERT：如果返回值等于该静态常量，说明此次事件是由插入行触发的。
- ☑ UPDATE：如果返回值等于该静态常量，说明此次事件是由修改行触发的。
- ☑ DELETE：如果返回值等于该静态常量，说明此次事件是由删除行触发的。

通过捕获表格模型事件，可以进行一些相关操作，如自动计算表格某一列的总和。

【例 14.7】　演示捕获和处理表格模型事件的方法，尤其是事件监听器接口 TableModelEvent 中各个方法的使用方法。（**实例位置：光盘\TM\sl\14\7**）

```
public class TableModelEvent_Example extends JFrame {
    private JTable table;                                   //声明一个表格对象
    private DefaultTableModel tableModel;                   //声明一个表格模型对象
    private JTextField aTextField;
    private JTextField bTextField;
    public static void main(String args[]) {
        TableModelEvent_Example frame = new TableModelEvent_Example();
        frame.setVisible(true);
    }
    public TableModelEvent_Example() {
        super();
        setTitle("表格模型事件示例");
        setBounds(100, 100, 500, 375);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        final JScrollPane scrollPane = new JScrollPane();
        getContentPane().add(scrollPane, BorderLayout.CENTER);
        String[] columnNames = { "A", "B" };
        String[][] rowValues = { { "A1", "B1" }, { "A2", "B2" }, { "A3", "B3" }, { "A4", "B4" } };
         //创建表格模型对象
        tableModel = new DefaultTableModel(rowValues, columnNames);
         //为表格模型添加事件监听器
        tableModel.addTableModelListener(new TableModelListener() {
            public void tableChanged(TableModelEvent e) {
                int type = e.getType();                     //获得事件的类型
                int row = e.getFirstRow() + 1;              //获得触发此次事件的表格行索引
                int column = e.getColumn() + 1;             //获得触发此次事件的表格列索引
                if (type == TableModelEvent.INSERT) {       //判断是否由插入行触发
```

```
                System.out.print("此次事件由 插入 行触发，");
                System.out.println("此次插入的是第 " + row + " 行！");
            } else if (type == TableModelEvent.UPDATE) {      //判断是否由修改行触发
                System.out.print("此次事件由 修改 行触发，");
                System.out.println("此次修改的是第 " + row + " 行第 " + column + " 列！");
            } else if (type == TableModelEvent.DELETE) {      //判断是否由删除行触发
                System.out.print("此次事件由 删除 行触发，");
                System.out.println("此次删除的是第 " + row + " 行！");
            } else {
                System.out.println("此次事件由 其他原因 触发！");
            }
        }
    });
    table = new JTable(tableModel);                           //利用表格模型对象创建表格对象
    scrollPane.setViewportView(table);
    final JPanel panel = new JPanel();
    getContentPane().add(panel, BorderLayout.SOUTH);
    final JLabel aLabel = new JLabel("A：");
    panel.add(aLabel);
    aTextField = new JTextField(15);
    panel.add(aTextField);
    final JLabel bLabel = new JLabel("B：");
    panel.add(bLabel);
    bTextField = new JTextField(15);
    panel.add(bTextField);
    final JButton addButton = new JButton("添加");
    addButton.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            String[] rowValues = { aTextField.getText(), aTextField.getText() };
            tableModel.addRow(rowValues);                     //向表格模型中添加一行
            aTextField.setText(null);
            bTextField.setText(null);
        }
    });
    panel.add(addButton);
    final JButton delButton = new JButton("删除");
    delButton.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            int[] selectedRows = table.getSelectedRows();     //获得表格中的选中行
            for (int row = 0; row < selectedRows.length; row++) {
                //从表格模型中移除表格中的选中行
                tableModel.removeRow(selectedRows[row] - row);
            }
        }
    });
    panel.add(delButton);
  }
}
```

运行结果如图 14.8 所示。

首先向表格中添加一行；然后依次双击值为 A2 和 B4 的单元格对其进行修改；最后选中表格的第 3 行和第 5 行，单击下面的“删除”按钮删除这两行，输出信息如图 14.9 所示。

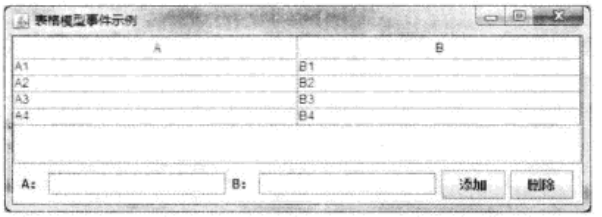


图 14.8　表格模型事件示例

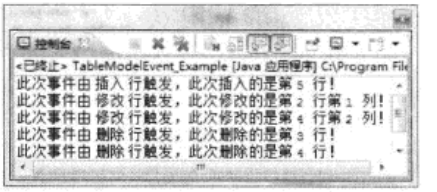


图 14.9　测试后控制台信息

14.6　经 典 范 例

14.6.1　经典范例 1：模拟相机拍摄

视频讲解：光盘\TM\lx\14\模拟相机拍摄.exe

在使用数码相机拍摄时，用户可以通过屏幕上提供的聚焦图标来选择拍摄的位置。本范例将实现类似功能，用户可以通过按方向键来完成移动镜头的操作，例如按→键可以让镜头向右移动。运行结果如图 14.10 所示。（**实例位置：光盘\TM\sl\14\8**）

图 14.10　运行结果

（1）在 Eclipse 中创建 JFrame 窗体，名为 KeyMove Background，对于图片，使用标签来保存。将保存背景图片的标签设置成窗体的背景，将保存镜头图片的标签应用到玻璃面板中。代码如下：

```
public KeyMoveBackground() {
    super();
    setResizable(false);                                          //禁止调整窗体大小
    getContentPane().setLayout(null);                             //设置空布局
    setTitle("方向键移动背景");                                    //设置窗体标题
    setBounds(100, 100, 500, 375);                                //设置窗体位置和大小
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);               //设置窗体退出时操作
    label = new JLabel();                                         //创建标签控件
    ImageIcon icon = new ImageIcon(getClass().getResource("background.jpg"));
    ImageIcon glassImg = new ImageIcon(getClass().getResource("glass.png"));
    label.setIcon(icon);                                          //设置标签使用背景图像
    label.setSize(icon.getIconWidth(), icon.getIconHeight());     //使标签与图像同步大小
    label.setLocation(-icon.getIconWidth() / 3, -icon.getIconHeight() / 3);   //设置标签默认位置
    addKeyListener(new KeyAdapter() {                             //窗体添加按键事件监听适配器
        public void keyPressed(final KeyEvent e) {
            do_label_keyPressed(e);                               //调用事件处理方法
        }
    });
```

```
    getContentPane().add(label);                                    //添加背景标签到窗体
    JLabel glassLabel = new JLabel(glassImg);                       //创建取景框标签
    JPanel glassPane = new JPanel(new BorderLayout());
    glassPane.add(glassLabel, BorderLayout.CENTER);                 //添加取景框标签到玻璃面板
    glassPane.setOpaque(false);                                     //使面板透明
    setGlassPane(glassPane);                                        //设置窗体使用玻璃面板
    getGlassPane().setVisible(true);                                //显示玻璃面板
}
```

（2）监听用户按键盘上方向键事件，通过移动背景图片标签来表现移动镜头的效果。每次移动 3 个像素，例如按→键可以让保存背景图片的标签向左移动 3 像素。代码如下：

```
protected void do_label_keyPressed(final KeyEvent e) {
    int code = e.getKeyCode();                                      //获取按键代码
    Point location = label.getLocation();                           //获取标签控件位置
    int step = 3;                                                   //移动速度
    switch (code) {
        case KeyEvent.VK_RIGHT:                                     //如果按键代码是右方向键
            if (location.x > (getWidth() - label.getWidth()))       //在不超出屏幕情况下
                label.setLocation(location.x - step, location.y);   //向左移动标签
            break;
        case KeyEvent.VK_LEFT                                       //如果按键代码是左方向键
            if (location.x < 0)                                     //在不超出屏幕情况下
                label.setLocation(location.x + step, location.y);   //向右移动标签
            break;
        case KeyEvent.VK_DOWN:                                      //如果按键代码是下方向键
            if (location.y > (getHeight() - label.getHeight()))     //在不超出屏幕情况下
                label.setLocation(location.x, location.y - step);   //向上移动标签
            break;
        case KeyEvent.VK_UP:                                        //如果按键代码是上方向键
            if (location.y < 0) {                                   //在不超出屏幕情况下
                label.setLocation(location.x, location.y + step);   //向下移动标签
            }
            break;
        default:
            break;
    }
}
```

14.6.2　经典范例 2：打地鼠游戏

视频讲解：光盘\TM\lx\14\打地鼠游戏.exe

打地鼠是一款非常简单易学的游戏，玩家使用鼠标不断地单击图片上的地鼠图标即可。本范例将完成这个游戏，其核心思想是监听用户单击鼠标事件，如果单击的位置有地鼠（一个包含图片的标签），则去掉标签上的图片。运行结果如图 14.11 所示。（**实例位置：光盘\TM\sl\14\9**）

图 14.11　运行结果

（1）在 Eclipse 中创建 JFrame 窗体，名为 Shrewmouse，在窗体中增加 7 个标签，一个用来显示游戏的背景图片，另外 6 个用来显示地鼠图片。监听标签上鼠标单击事件，如果该标签上有图标，则将其删除。代码如下：

```
public Shrewmouse() {
    super();
    setResizable(false);                                                    //禁止调整窗体大小
    getContentPane().setLayout(null);                                       //窗体不使用布局管理器
    setTitle("简易打地鼠游戏");                                              //设置窗体标题
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);                         //设置窗体关闭属性
    //初始化背景图片对象
    ImageIcon img = new ImageIcon(getClass().getResource("background.jpg"));
    imgMouse = new ImageIcon(getClass().getResource("mouse.png"));          //初始化地鼠图片对象
    mouses = new JLabel[6];                                                 //创建显示地鼠的标签数组
    for (int i = 0; i < 6; i++) {                                           //遍历数组
        mouses[i] = new JLabel();                                           //初始化每一个数组元素
        //设置标签与地鼠图片相同大小
        mouses[i].setSize(imgMouse.getIconWidth(), imgMouse.getIconHeight());
        //为标签添加鼠标事件监听适配器
        mouses[i].addMouseListener(new MouseAdapter() {
                @Override
                public void mouseClicked(MouseEvent e) {
                    Object source = e.getSource();                          //获取事件源，即地鼠标签
                    if (source instanceof JLabel) {                         //如果事件源是标签控件
                        JLabel mouse = (JLabel) source;                     //强制转换为 JLabel 标签
                        mouse.setIcon(null);                                //取消标签图标
                    }
                }
            });
        getContentPane().add(mouses[i]);                                    //添加显示地鼠的标签到窗体
    }
```

```
    mouses[0].setLocation(253, 300);                                    //设置每个标签的位置
    mouses[1].setLocation(333, 250);                                    //设置每个标签的位置
    mouses[2].setLocation(388, 296);                                    //设置每个标签的位置
    mouses[3].setLocation(362, 364);                                    //设置每个标签的位置
    mouses[4].setLocation(189, 353);                                    //设置每个标签的位置
    mouses[5].setLocation(240, 409);                                    //设置每个标签的位置
    final JLabel backLabel = new JLabel();                              //创建显示背景的标签
    backLabel.setBounds(0, 0, img.getIconWidth(), img.getIconHeight());  //设置标签与背景图片相同大小
    setBounds(100, 100, img.getIconWidth(), img.getIconHeight() + 30);  //设置窗体近似背景图片大小
    backLabel.setIcon(img);                                             //添加背景到标签
    getContentPane().add(backLabel);                                    //添加背景标签到窗体
}
```

（2）编写多线程的核心方法，它可以每隔一秒钟随机让一个没有显示图片的标签显示地鼠图片。代码如下：

```
public void run() {
    while (true) {                                                      //使用无限循环
        try {
            Thread.sleep(1000);                                         //使线程休眠 1 秒
            int index = (int) (Math.random() * 6);                      //生成随机的地鼠索引
            if (mouses[index].getIcon() == null)                        //如果地鼠标签没有设置图片
                mouses[index].setIcon(imgMouse);                        //为该标签添加地鼠图片
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

14.7　本 章 小 结

通过本章的学习，相信读者已经可以熟练地处理一些高级事件，包括键盘事件、鼠标事件、窗体事件、选项事件和表格模型事件。至此，已经掌握了 5 种事件的处理方法，通过配合使用这 5 种事件监听器，可以充分控制各种组件，例如通过为文本框组件同时添加焦点事件监听器和键盘事件监听器，可以有效地控制文本框中输入的内容。

14.8　实 战 练 习

1．设计一个通过捕获文本框的键盘事件实现只允许输入数字的文本框。（**答案位置：光盘\TM\sl\14\10**）

2．设计一个通过捕获表格模型事件实现自动计算表格某一数值列的和。（**答案位置：光盘\TM\sl\14\11**）

第15章

多线程

（视频讲解：41分钟）

如果一次只完成一件事情，很容易实现，但事实上世间很多事情都是同时进行的，所以在 Java 中为了模拟这种状态，引入了线程机制。简单地说，当程序同时完成多件事情时，就是所谓的多线程程序。多线程应用相当广泛，使用多线程可以创建窗口程序、网络程序等。本章会由浅入深地介绍多线程，除了介绍多线程的概念之外，还结合实例让读者了解如何使程序具有多线程功能。

通过阅读本章，您可以：

- 了解多线程在 Windows 操作系统中的执行模式
- 掌握实现线程的两种方式
- 掌握线程的状态
- 掌握使线程进入各种状态的方法
- 掌握线程的优先级
- 掌握线程安全
- 掌握线程同步机制
- 掌握线程间的通信

PDG

15.1 线程简介

视频讲解：光盘\TM\lx\15\线程简介.exe

世间万物会同时完成很多工作，如人体同时进行呼吸、血液循环、思考问题等活动，用户既可以使用计算机听歌，也可以使用它打印文件，而这些活动完全可以同时进行，这种思想放在 Java 中被称为并发，而将并发完成的每一件事情称为线程。

在人们的生活中，并发机制非常重要，但并不是所有的程序语言都支持线程。在以往的程序中，多以一个任务完成后再进行下一个项目的模式进行开发，这样下一个任务的开始必须等待前一个任务的结束。Java 语言提供并发机制，程序员可以在程序中执行多个线程，每一个线程完成一个功能，并与其他线程并发执行，这种机制被称为多线程。

多线程是非常复杂的机制，如果此时读者不能体会这句话的含义，可以理解为同时阅读 3 本书，首先阅读第 1 本书第 1 章，然后再阅读第 2 本书第 1 章，再阅读第 3 本书第 1 章，回过头再阅读第 1 本书第 2 章，依此类推，不用很长时间读者就可以体会多线程的复杂性。

既然多线程这样复杂，那么它在操作系统中是怎样工作的呢？其实 Java 中的多线程在每个操作系统中的运行方式也存在差异，在此着重说明多线程在 Windows 操作系统的运行模式。Windows 操作系统是多任务操作系统，它以进程为单位。一个进程是一个包含有自身地址的程序，每个独立执行的程序都称为进程，也就是正在执行的程序。系统可以分配给每个进程一段有限的使用 CPU 的时间（也可以称为 CPU 时间片），CPU 在这段时间中执行某个进程，然后下一个时间片又跳至另一个进程中去执行。由于 CPU 转换较快，所以使得每个进程好像是同时执行一样。

图 15.1 表明了 Windows 操作系统的执行模式。

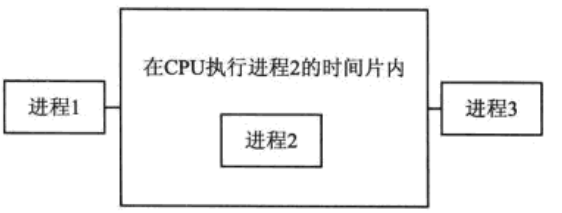


图 15.1　Windows 操作系统的执行模式

一个线程则是进程中的执行流程，一个进程中可以同时包括多个线程，每个线程也可以得到一小段程序的执行时间，这样一个进程就可以具有多个并发执行的线程。在单线程中，程序代码按调用顺序依次往下执行，如果需要一个进程同时完成多段代码的操作，就需要产生多线程。

15.2 实现线程的两种方式

视频讲解：光盘\TM\lx\15\实现线程的两种方式.exe

在 Java 中主要提供了两种方式实现线程，分别为继承 java.lang.Thread 类与实现 java.lang.Runnable 接口。本节将着重讲解这两种实现线程的方式。

15.2.1　继承 Thread 类

Thread 类是 java.lang 包中的一个类，从这个类中实例化的对象代表线程，程序员启动一个新线程需要建立 Thread 实例。Thread 类中常用的两个构造方法如下：

- ☑ public Thread(String threadName)
- ☑ public Thread()

其中第一个构造方法是创建一个名称为 threadName 的线程对象。

继承 Thread 类创建一个新的线程的语法格式如下：

```
public class ThreadTest extends Thread{
    //...
}
```

完成线程真正功能的代码放在类的 run()方法中，当一个类继承 Thread 类后，就可以在该类中覆盖 run()方法，将实现该线程功能的代码写入 run()方法中，然后同时调用 Thread 类中的 start()方法执行线程，也就是调用 run()方法。

Thread 对象需要一个任务来执行，任务是指线程在启动时执行的工作，该工作的功能代码被写在 run()方法中。这个 run()方法必须使用如下这种语法格式：

```
public void run(){
    //...
}
```

注意

如果 start()方法调用一个已经启动的线程，系统将抛出 IllegalThreadStateException 异常。

当执行一个线程程序时，就自动产生一个线程，主方法正是在这个线程上运行的。当不再启动其他线程时，该程序就为单线程程序，如本章以前的程序都是单线程程序。主方法线程启动由 Java 虚拟机负责，程序员负责启动自己的线程。

语法格式如下：

```
public static void main(String[] args) {
    new ThreadTest().start();
}
```

下面看一个继承 Thread 类的实例。

【例 15.1】　在项目中创建 ThreadTest 类，该类继承 Thread 类方法创建线程。（**实例位置：光盘\TM\sl\15\1**）

```
public class ThreadTest extends Thread{                    //指定类继承 Thread 类
    private int count=10;
    public void run(){                                     //重写 run()方法
```

```
        while(true){
            System.out.print(count+"  ");                    //打印 count 变量
            if(--count==0){                                  //使 count 变量自减，当自减为 0 时，退出循环
                return;
            }
        }
    }
    public static void main(String[] args) {
        new ThreadTest().start();
    }
}
```

运行结果如图 15.2 所示。

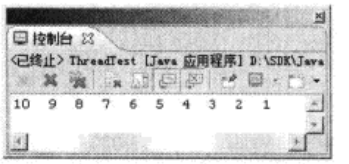


图 15.2　使用继承 Thread 类方法创建线程

在上述实例中，该类继承了 Thread 类，然后在该类中覆盖了 run()方法。通常在 run()方法中使用无限循环的形式使得线程一直运行下去，所以要指定一个跳出循环的条件，例如本实例中使用变量 count 递减为 0 作为跳出循环的条件。

在 main()方法中，使线程执行需要调用 Thread 类中的 start()方法，start()方法调用被覆盖的 run()方法，如果不调用 start()方法，线程永远都不会启动，在主方法没有调用 start()方法之前，Thread 对象只是一个实例，而不是一个真正的线程。

15.2.2　实现 Runnable 接口

到目前为止，线程都是通过扩展 Thread 类来创建的，如果程序员需要继承其他类（非 Thread 类）并使该程序可以使用线程，就需要使用 Runnable 接口。例如，一个扩展 JFrame 类的 GUI 程序不可能再继承 Thread 类，因为 Java 语言中不支持多继承，这时该类就需要实现 Runnable 接口使其具有使用线程的功能。

实现 Runnable 接口的语法格式如下：

```
public class Thread extends Object implements Runnable
```

说明　有兴趣的读者可以查询 API，从中可以惊奇地发现实质上 Thread 类就是实现了 Runnable 接口，其中的 run()方法正是对 Runnable 接口中的 run()方法的具体实现。

实现 Runnable 接口的程序会创建一个 Thread 对象，并将 Runnable 对象与 Thread 对象相关联。Thread 类中有如下两个构造方法：

- ☑ public Thread(Runnable r)
- ☑ public Thread(Runnable r,String name)

这两个构造方法的参数中都存在 Runnable 实例，使用以上构造方法就可以将 Runnable 实例与 Thread 实例相关联。

使用 Runnable 接口启动新的线程的步骤如下：

（1）建立 Runnable 对象。

（2）使用参数为 Runnable 对象的构造方法创建 Thread 实例。

（3）调用 start()方法启动线程。

通过 Runnable 接口创建线程时程序员首先需要编写一个实现 Runnable 接口的类，然后实例化该类的对象，这样就建立了 Runnable 对象；接下来使用相应的构造方法创建 Thread 实例；最后使用该实例调用 Thread 类中的 start()方法启动线程。图 15.3 表明了实现 Runnable 接口创建线程的流程。

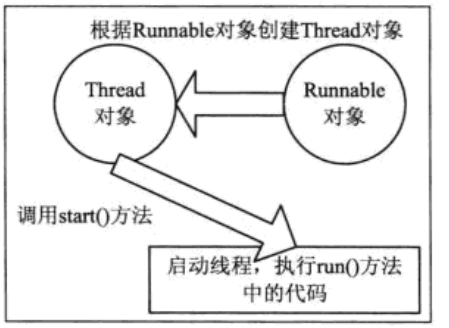


图 15.3　实现 Runnable 接口创建线程的流程

线程最引人注目的部分应该是与 Swing 相结合创建 GUI 程序，下面演示一个 GUI 程序，该程序实现了图标滚动的功能。

【例 15.2】　在项目中创建 SwingAndThread 类，该类继承了 JFrame 类，实现图标在标签上滚动的功能，其中使用了 Swing 与线程相结合的技术。（**实例位置：光盘\TM\sl\15\2**）

```
import java.awt.Container;
import java.net.URL;
import java.util.Random;
import javax.swing.*;
public class SwingAndThread extends JFrame{
    private JLabel jl=new JLabel();                                    //声明 JLabel 对象
    private static Thread t;                                           //声明线程对象
    private int count=0;                                               //声明计数变量
    private Container container = getContentPane();                    //声明容器
    public SwingAndThread(){
        setBounds(300, 200, 250, 100);                                 //绝对定位窗体大小与位置
        container.setLayout(null);                                     //使窗体不使用任何布局管理器
        //获取图片的 URL
        URL url= SwingAndThread.class.getResource("/com/mingrisoft/1.gif");
        Icon icon = new ImageIcon(url);                                //实例化一个 Icon
        jl.setIcon(icon);                                              //将图标放置在标签中
        jl.setHorizontalAlignment(SwingConstants.LEFT);                //设置图片在标签的最左方
        jl.setBounds(10, 10, 200, 50);                                 //设置标签的位置与大小
        jl.setOpaque(true);
        t=new Thread(new Runnable(){                       //定义匿名内部类，该类实现 Runnable 接口
            public void run(){                             //重写 run()方法
                while(count<=200){                         //设置循环条件
                    jl.setBounds(count, 10, 200, 50);      //将标签的横坐标用变量表示
                    try{
                        t.sleep(1000);                     //使线程休眠 1000 毫秒
                    }
                    catch(Exception e){
                        e.printStackTrace();
                    }
                    count+=4;                              //使横坐标每次增加 4
```

```
                    if(count==200){
                        count=10;                    //当图标到达标签的最右边时，使其回到标签最左边
                    }
                }
            }
        });
        t.start();                                   //启动线程
        container.add(jl);                           //将标签添加到容器中
        setVisible(true);                            //使窗体可视
        //设置窗体的关闭方式
        setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);
    }
    public static void main(String[] args) {
        new SwingAndThread();                        //实例化一个 SwingAndThread 对象
    }
}
```

运行结果如图 15.4 所示。

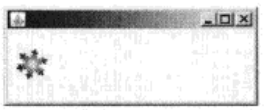
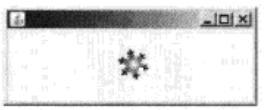



图 15.4　使图标在标签上滚动

在本实例中，为了使图标具有滚动功能，需要在类的构造方法中创建 Thread 实例。在创建该实例的同时需要 Runnable 对象作为 Thread 类构造方法的参数，然后使用内部类形式实现 run()方法。在 run()方法中主要循环图标的横坐标位置，当图标横坐标到了标签的最右方时，将图标的横坐标再置于图标滚动的初始位置。

注意　启动一个新的线程，不是直接调用 Thread 子类对象的 run()方法，而是调用 Thread 子类的 start()方法，Thread 类的 start()方法产生一个新的线程，该线程运行 Thread 子类的 run()方法。

15.2.3　范例 1：查看线程的运行状态

线程共有新建、运行（可运行）、阻塞、等待、计时等待和终止 6 种状态。当使用 new 操作符创建新线程时，线程处于新建状态。当调用 start()方法时，线程处于运行（可运行）状态。当线程需要获得对象的内置锁，而该锁正被其他线程拥有时，线程处于阻塞状态。当线程等待其他线程通知调度表可以运行时，该线程处于等待状态。对于一些含有时间参数的方法，如 Thread 类的 sleep()方法，可以使线程处于计时等待状态。当 run()方法运行完毕或出现异常时，线程处于终止状态。运行结果如图 15.5 所示。**（实例位置：光盘\TM\sl\15\3）**

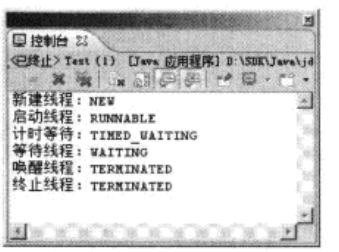


图 15.5　查看线程的运行状态

（1）编写 ThreadState 类，该类实现了 Runnable 接口。在该类中定义了 3 个方法：waitForASecond()方法用于将当前线程暂时等待 0.5 秒，waitForYears()方法用于将当前线程永久等待，notifyNow()方法用于通知等待状态的线程运行。run()方法中，运行了 waitForASecond()和 waitForYears()方法，代码如下：

```
public class ThreadState implements Runnable {
    public synchronized void waitForASecond() throws InterruptedException {
        wait(500);                          //使当前线程等待 0.5 秒或其他线程调用 notify()或 notifyAll()方法
    }
    public synchronized void waitForYears() throws InterruptedException {
        wait();                             //使当前线程永久等待，直到其他线程调用 notify()或 notifyAll()方法
    }
    public synchronized void notifyNow() throws InterruptedException {
        notify();                           //唤醒由调用 wait()方法进入等待状态的线程
    }
    public void run() {
        try {
            waitForASecond();               //在新线程中运行 waitForASecond()方法
            waitForYears();                 //在新线程中运行 waitForYears()方法
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

（2）编写 Test 类进行测试，在 main()方法中输出了线程的各种不同状态。代码如下：

```
public class Test {
    public static void main(String[] args) throws InterruptedException {
        ThreadState state = new ThreadState();                  //创建 state 对象
        Thread thread = new Thread(state);                      //利用 state 对象创建 Thread 对象
        System.out.println("新建线程：" + thread.getState());    //输出线程状态
        thread.start();                                         //调用 thread 对象的 start()方法，启动新线程
        System.out.println("启动线程：" + thread.getState());    //输出线程状态
        Thread.sleep(100);                  //当前线程休眠 0.1 秒，使新线程运行 waitForASecond()方法
        System.out.println("计时等待：" + thread.getState());    //输出线程状态
        Thread.sleep(1000);                 //当前线程休眠 1 秒，使新线程运行 waitForYears()方法
        System.out.println("等待线程：" + thread.getState());    //输出线程状态
        state.notifyNow();                                      //调用 state 的 notifyNow()方法
        System.out.println("唤醒线程：" + thread.getState());    //输出线程状态
        Thread.sleep(1000);                                     //当前线程休眠 1 秒，使新线程结束
        System.out.println("终止线程：" + thread.getState());    //输出线程状态
    }
}
```

15.2.4 范例 2：查看 JVM 中的线程名

在 Java 虚拟机（JVM）中，除了用户创建的线程，还有服务于用户线程的其他线程。它们根据用途被分配到不同的组中进行管理。本范例演示了如何查看 JVM 中线程的名称及其所在组的名称。运行结果如图 15.6 所示。（**实例位置：光盘\TM\sl\15\4**）

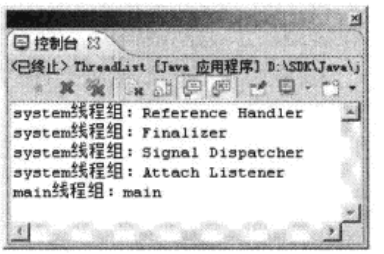


图 15.6 查看 JVM 的线程名

注意 读者可以在本范例的基础上增加想查看的信息，如线程的状态、线程的优先级、是否为守护线程等。

编写 ThreadList 类，在该类中包含了 4 个方法：getRootThreadGroups()方法用于获得根线程组，getThreads()方法用于获得指定线程组中所有线程的名称，getThreadGroups()方法用于获得线程组中所有子线程组，main()方法用于测试。代码如下：

```
public class ThreadList {
    private static ThreadGroup getRootThreadGroups() {                    //获得根线程组
        ThreadGroup rootGroup = Thread.currentThread().getThreadGroup();  //获得当前线程组
        while (true) {
            if (rootGroup.getParent() != null) {        //如果 getParent()方法的返回值非空则不是根线程组
                rootGroup = rootGroup.getParent();                        //获得父线程组
            } else {
                break;                                                    //如果到达根线程组则退出循环
            }
        }
        return rootGroup;                                                 //返回根线程组
    }
    public static List<String> getThreads(ThreadGroup group) {            //获得给定线程组中所有线程名
        List<String> threadList = new ArrayList<String>();                //创建保存线程名的列表
        Thread[] threads = new Thread[group.activeCount()];               //根据活动线程数创建线程数组
        int count = group.enumerate(threads, false);                      //复制线程到线程数组
        for (int i = 0; i < count; i++) {           //遍历线程数组将线程名及其所在组保存到列表中
            threadList.add(group.getName() + "线程组：" + threads[i].getName());
        }
```

```
        return threadList;                                          //返回列表
    }
    public static List<String> getThreadGroups(ThreadGroup group) {          //获得线程组中子线程组
        List<String> threadList = getThreads(group);            //获得给定线程组中线程名
        ThreadGroup[] groups = new ThreadGroup[group.activeGroupCount()];     //创建线程组数组
        int count = group.enumerate(groups, false);             //复制子线程组到线程组数据
        for (int i = 0; i < count; i++) {                                     //遍历所有子线程组
            threadList.addAll(getThreads(groups[i]));           //利用 getThreads()方法获得线程名列表
        }
        return threadList;                                                    //返回所有线程名
    }
    public static void main(String[] args) {
        for (String string : getThreadGroups(getRootThreadGroups())) {
            System.out.println(string);                         //遍历输出列表中的字符串
        }
    }
}
```

注意　读者在编写多线程程序时，时刻要注意线程的状态。不同状态下，线程能够执行的任务是不同的。

15.3　线程的生命周期

视频讲解：光盘\TM\lx\15\线程的生命周期.exe

线程具有生命周期，其中包含出生状态、就绪状态、运行状态、等待状态、休眠状态、阻塞状态和死亡状态 7 种状态。出生状态就是用户在创建线程时处于的状态，在用户使用该线程实例调用 start()方法之前线程都处于出生状态；当用户调用 start()方法后，线程处于就绪状态（又被称为可执行状态）；当线程得到系统资源后就进入运行状态。

一旦线程进入可执行状态，它会在就绪与运行状态下辗转，同时也有可能进入等待、休眠、阻塞或死亡状态。当处于运行状态下的线程调用 Thread 类中的 wait()方法时，该线程处于等待状态，进入等待状态的线程必须调用 Thread 类中的 notify()方法才能被唤醒，而 notifyAll()方法是将所有处于等待状态下的线程唤醒；当线程调用 Thread 类中的 sleep()方法时，则会进入休眠状态。如果一个线程在运行状态下发出输入/输出请求，该线程将进入阻塞状态，在其等待输入/输出结束时线程进入就绪状态，对于阻塞的线程来说，即使系统资源空闲，线程依然不能回到运行状态；当线程的 run()方法执行完毕时，线程进入死亡状态。图 15.7 描述了线程生命周期中的各种状态。

说明　使线程处于不同状态下的方法会在 15.4 节中进行介绍，在此读者了解线程的多个状态即可。

虽然多线程看起来像同时执行，但事实上在同一时间点上只有一个线程被执行，只是线程之间切换较快，所以才会使人产生线程是同时进行的假象。在 Windows 操作系统中，系统会为每个线程分配一小段 CPU 时间片，一旦 CPU 时间片结束就会将当前线程换为下一个线程，即使在该线程没有结束的情况下。

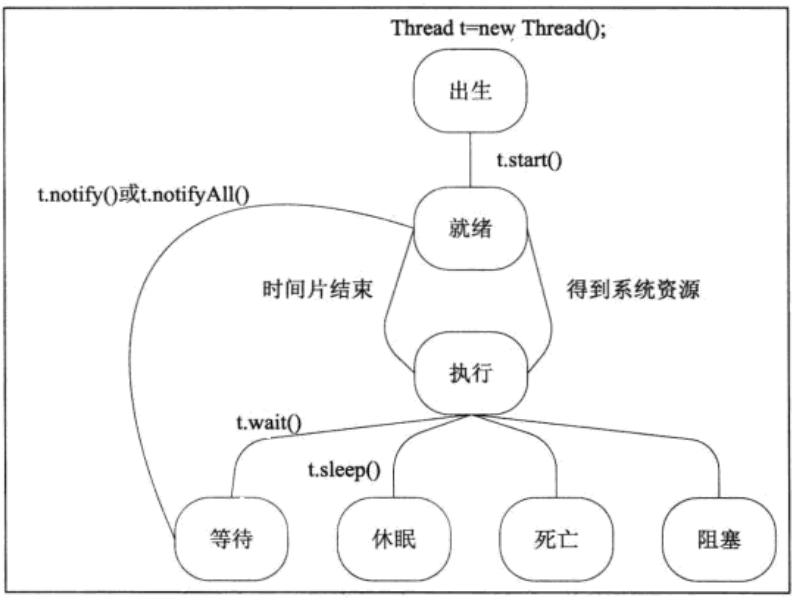


图 15.7　线程的生命周期状态图

根据图 15.7，可以总结出使线程处于就绪状态有以下几种可能。

- ☑ 调用 sleep()方法。
- ☑ 调用 wait()方法。
- ☑ 等待输入/输出完成。

当线程处于就绪状态后，可以通过以下几种方式使线程再次进入运行状态。

- ☑ 线程调用 notify()方法。
- ☑ 线程调用 notifyAll()方法。
- ☑ 线程调用 interrupt()方法。
- ☑ 线程的休眠时间结束。
- ☑ 输入/输出结束。

图 15.7 中描述了线程的生命周期状态，下面将着重讲解使线程处于各种状态的方法。

15.4　操作线程的方法

视频讲解：光盘\TM\lx\15\操作线程的方法.exe

操作线程有很多方法，这些方法可以使线程从某一种状态过渡到另一种状态。

15.4.1 线程的休眠

一种能控制线程行为的方法是调用 sleep()方法。sleep()方法需要一个参数用于指定该线程休眠的时间，该时间使用毫秒为单位。在前面的实例中已经演示过 sleep()方法，它通常是在 run()方法内的循环中被使用。

sleep()方法的语法格式如下：

```
try{
Thread.sleep(2000);
}catch(InterruptedException e){
    e.printStackTrace();
}
```

上述代码不会使线程在 2 秒之内进入就绪状态。由于 sleep()方法的执行有可能抛出 InterruptedException 异常，所以将 sleep()方法的调用放在 try…catch 块中。虽然使用了 sleep()方法的线程在一段时间内会醒来，但是并不能保证它醒来后进入运行状态，只能保证它进入就绪状态。

为了使读者更深入地了解线程的休眠方法，来看下面的实例。

【例 15.3】 在项目中创建 SleepMethodTest 类，该类继承了 JFrame 类，实现在窗体中自动画线段的功能，并且为线段设置颜色，颜色是随机产生的。（**实例位置：光盘\TM\sl\15\5**）

```
public class SleepMethodTest extends JFrame {
    private Thread t;
    //定义颜色数组
    private static Color[] color = { Color.BLACK, Color.BLUE, Color.CYAN,
            Color.GREEN, Color.ORANGE, Color.YELLOW, Color.RED, Color.PINK,
            Color.LIGHT_GRAY };
    private static final Random rand = new Random();                    //创建随机对象
    private static Color getC() {                                       //获取随机颜色值的方法
        return color[rand.nextInt(color.length)];
    }
    public SleepMethodTest() {
        t = new Thread(new Runnable() {                                 //创建匿名线程对象
                int x = 30;                                             //定义初始坐标
                int y = 50;

                public void run() {                                     //覆盖线程接口方法
                    while (true) {                                      //无限循环
                        try {
                            t.sleep(100);                               //线程休眠 0.1 秒
                        } catch (InterruptedException e) {
                            e.printStackTrace();
                        }
                        Graphics graphics = getGraphics();              //获取组件绘图上下文对象
                        graphics.setColor(getC());                      //设置绘图颜色
                        graphics.drawLine(x, y, 100, y++);              //绘制直线并递增垂直坐标
```

```
                    if (y >= 80) {
                        y = 50;
                    }
                }
            }
        }
    });
    t.start();                                          //启动线程
}
public static void main(String[] args) {
    init(new SleepMethodTest(), 100, 100);
}
public static void init(JFrame frame, int width, int height) {      //初始化程序界面的方法
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setSize(width, height);
    frame.setVisible(true);
}
}
```

运行结果如图 15.8 所示。

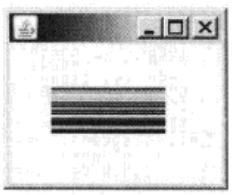


图 15.8　线程的休眠

在本实例中定义了 getC()方法，该方法用于随机产生 Color 类型的对象，并且在产生线程的匿名内部类中使用 getGraphics()方法获取 Graphics 对象，使用该对象调用 setColor()方法为图形设置颜色；调用 drawLine()方法绘制一条线段，同时线段会根据纵坐标的变化自动调整。

15.4.2　线程的加入

如果当前某程序为多线程程序，假如存在一个线程 A，现在需要插入线程 B，并要求线程 B 先执行完毕，然后再继续执行线程 A，此时可以使用 Thread 类中的 join()方法来完成。这就好比此时读者正在看电视，却突然有人上门收水费，读者必须付完水费后才能继续看电视。

当某个线程使用 join()方法加入到另一个线程时，另一个线程会等待该线程执行完毕再继续执行。

【例 15.4】　在项目中创建 JoinTest 类，该类继承了 JFrame 类。该实例包括两个进度条，进度条的进度由线程来控制，通过使用 join()方法使上面的进度条必须等待下面的进度条完成后才可以继续。**（实例位置：光盘\TM\sl\15\6）**

```
import java.awt.BorderLayout;
import javax.swing.JFrame;
import javax.swing.JProgressBar;
public class JoinTest extends JFrame{
    private Thread threadA;                                     //定义两个线程
    private Thread threadB;
    final JProgressBar progressBar = new JProgressBar();        //定义两个进度条组件
    final JProgressBar progressBar2 = new JProgressBar();
    int count=0;
    public static void main(String[] args) {
        init(new JoinTest(),100,100);
```

```
    }
    public JoinTest() {
        super();
        getContentPane().add(progressBar, BorderLayout.NORTH);          //将进度条设置在窗体最北面
        getContentPane().add(progressBar2, BorderLayout.SOUTH);         //将进度条设置在窗体最南面
        progressBar.setStringPainted(true);                             //设置进度条显示数字字符
        progressBar2.setStringPainted(true);
        //使用匿名内部类形式初始化 Thread 实例
        threadA=new Thread(new Runnable()
            int count=0;
            public void run(){                                          //重写 run()方法
                while(true){
                    progressBar.setValue(++count);                      //设置进度条的当前值
                    try{
                        threadA.sleep(100);                             //使线程 A 休眠 100 毫秒
                        threadB.join();                                 //使线程 B 调用 join()方法
                    }catch(Exception e){
                        e.printStackTrace();
                    }
                }
            }
        });
        threadA.start();                                                //启动线程 A
        threadB=new Thread(new Runnable(){
            int count=0;
            public void run(){
                while(true){
                    progressBar2.setValue(++count);                     //设置进度条的当前值
                    try{
                        threadB.sleep(100);                             //使线程 B 休眠 100 毫秒
                    }catch(Exception e){
                        e.printStackTrace();
                    }
                    if(count==100)                                      //当 count 变量增长为 100 时
                        break;                                          //跳出循环
                }
            }
        });
        threadB.start();                                                //启动线程 B
    }
    public static void init(JFrame frame,int width,int height){         //设置窗体各种属性方法
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(width, height);
        frame.setVisible(true);
    }
}
```

运行结果如图 15.9 所示。

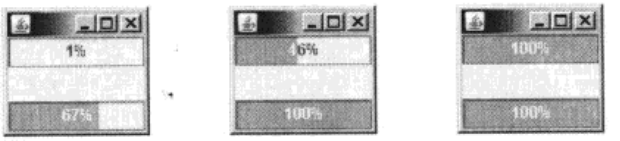


图 15.9　使用 join()方法控制进度条的滚动

在本实例中同时创建了两个线程，这两个线程分别负责进度条的滚动。在线程 A 的 run()方法中使线程 B 的对象调用 join()方法，因为 join()方法是使当前运行线程暂停，直到调用 join()方法的线程执行完毕后再执行，所以线程 A 等待线程 B 执行完毕后再开始执行，也就是下面的进度条滚动完毕后上面的进度条才开始滚动。

15.4.3　线程的中断

在以往时候会使用 stop()方法停止线程，但当前版本的 JDK 早已废除了 stop()方法，同时也不建议使用 stop()方法来停止一个线程的运行。现在提倡在 run()方法中使用无限循环的形式，然后使用一个布尔型标记控制循环的停止。

【例 15.5】　在项目中创建 InterruptedTest 类，该类继承了 Runnable 类，设置线程正确的停止方式。

```
public class InterruptedTest implements Runnable{
    private boolean isContinue=false;                    //设置一个标记变量，默认值为 false
    public void run(){                                   //重写 run()方法
        while(true){
            //...
            if(isContinue)                               //当 isContinue 变量为 true 时，停止线程
                break;
        }
    }
    public void setContinue(){                           //定义设置 isContinue 变量为 true 的方法
        this.isContinue=true;
    }
}
```

如果线程是因为使用了 sleep()或 wait()方法进入了就绪状态，这时可以使用 Thread 类中的 interrupt()方法使线程离开 run()方法，同时结束线程，但程序会抛出 InterruptedException 异常。

【例 15.6】　在项目中创建 InterruptedSwing 类，该类继承了 Runnable 类，创建一个进度条，在表示进度条的线程中使用 interrupted()方法。（**实例位置：光盘\TM\sl\15\7**）

```
public class InterruptedSwing extends JFrame{
    Thread thread;
    public static void main(String[] args) {
        init(new InterruptedSwing(),100,100);
    }
    public InterruptedSwing() {
        super();
        final JProgressBar progressBar = new JProgressBar();          //创建进度条
        getContentPane().add(progressBar, BorderLayout.NORTH);        //将进度条放置在窗体合适位置
```

```
        progressBar.setStringPainted(true);                          //设置进度条上显示的数字
        thread=new Thread(new Runnable(){
            int count=0;
            public void run(){
                while(true){
                    progressBar.setValue(++count);                   //设置进度条的当前值
                    try{
                        thread.sleep(1000);                          //使线程休眠 1000 毫秒
                    }catch(InterruptedException e){                  //捕捉 InterruptedException 异常
                        System.out.println("当前线程被中断");
                    }
                }
            }
        });
        thread.start();                                              //启动线程
        thread.interrupt();                                          //中断线程
    }
    public static void init(JFrame frame,int width,int height){
        ...//非关键代码省略
    }
}
```

运行结果如图 15.10 所示。

图 15.10　线程的中断

15.4.4　线程的礼让

Thread 类中提供了一种礼让方法，使用 yield()方法表示，它只是给当前正处于运行状态下的线程一个提醒，告知它可以将资源礼让给其他线程，但这仅仅是一种暗示，没有任何一种机制保证当前线程会将资源礼让。

yield()方法使具有同样优先级的线程有进入可执行状态的机会，当当前线程放弃执行权时会再度回到就绪状态。对于支持多任务的操作系统来说，不需要调用 yeild()方法，因为操作系统会为线程自动分配 CPU 时间片来执行。

15.4.5　范例 3：查看和修改线程优先级

Java 中每个线程都有优先级属性。默认情况下，新建线程的优先级与创建该线程的线程优先级相同。每当线程调度器选择要运行的线程时，通常选择优先级较高的线程。本范例演示如何查看和修改

线程的优先级。运行结果如图 15.11 所示。（**实例位置：光盘\TM\sl\15\8**）

图 15.11　修改线程优先级

注意　线程优先级是高度依赖于操作系统的，而且 Sun 对于不同的操作系统提供的虚拟机并不完全相同。

（1）编写 ThreadPriorityTest 类，该类继承了 JFrame。在框架中包含了一个表格，用来显示当前线程组中运行的线程。一个文本域用来获得用户输入的新线程优先级。“修改”按钮实现了修改优先级的功能，并更新了表格。

（2）编写 do_this_windowActivated()方法，用来监听窗体激活事件。在该方法中，使用当前线程所在线程组中的线程 ID、名称和优先级作为表格的数据。代码如下：

```
protected void do_this_windowActivated(WindowEvent e) {
    ThreadGroup group = Thread.currentThread().getThreadGroup();          //获得当前线程所在线程组
    Thread[] threads = new Thread[group.activeCount()];                   //使用数组保存活动状态的线程
    group.enumerate(threads);                                             //获得所有线程
    DefaultTableModel model = (DefaultTableModel) table.getModel();       //获得表格模型
    model.setRowCount(0);                                                 //清空表格模型中的数据
    //定义表头
    model.setColumnIdentifiers(new Object[] { "线程 ID", "线程名称", "优先级" });
    for (Thread thread : threads) {                                       //增加行数据
        model.addRow(new Object[] { thread.getId(), thread.getName(),
        thread.getPriority() });
    }
    table.setModel(model);                                                //更新表格模型
}
```

（3）编写 do_button_actionPerformed()方法，用来监听单击“修改”按钮事件。在该方法中，获得了用户输入的优先级和用户选择的行，根据用户输入的优先级修改了线程优先级。代码如下：

```
protected void do_button_actionPerformed(ActionEvent e) {
    String text = textField.getText();                                    //获得用户输入的优先级
    Integer priority = Integer.parseInt(text);                            //将优先级转换成 Integer 对象
    int selectedRow = table.getSelectedRow();                             //获得用户选择的行
    DefaultTableModel model = (DefaultTableModel) table.getModel();       //获得默认表格模型
    model.setValueAt(priority, selectedRow, 2);                           //更改表格中的数据
    repaint();                                                            //重新绘制各个控件
}
```

15.4.6　范例 4：休眠当前线程

在龟兔赛跑的寓言故事中，原本领先的兔子因为骄傲自满在中途休息而痛失了冠军。其寓意告诉人们为人要谦虚，不要骄傲。本范例将使用线程休眠技术来模拟龟兔赛跑的过程。运行结果如图 15.12 所示。**（实例位置：光盘\TM\sl\15\9）**

（1）编写 RaceFrame 类，该类继承了 JFrame。在框架中包含了两个文本域用来输出乌龟和兔子的比赛记录，“比赛开始”按钮用来开始比赛。

（2）编写内部类 Rabbit，该类实现了 Runnable 接口。在 run() 方法中，让兔子休眠了 1 秒钟，代码如下：

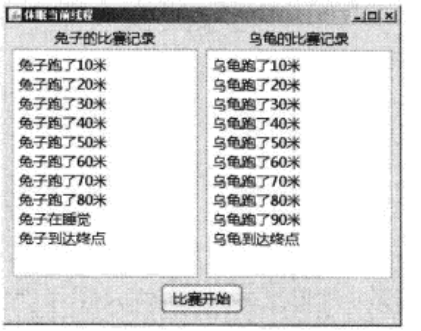


图 15.12　休眠当前线程

```
private class Rabbit implements Runnable {
    public void run() {
        for (int i = 1; i < 11; i++) {                          //循环 10 次模拟赛跑的过程
            String text = rabbitTextArea.getText();             //获得文本域中的信息
            try {
                Thread.sleep(1);                                //线程休眠 0.001 秒，模拟兔子在跑步
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            rabbitTextArea.setText(text + "兔子跑了" + i + "0 米\n");   //显示兔子的跑步距离
            if (i == 9) {
                rabbitTextArea.setText(text + "兔子在睡觉\n");      //当跑了 90 米时开始睡觉
                try {
                    Thread.sleep(10000);                        //线程休眠 10 秒，模拟兔子在睡觉
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
            if (i == 10) {
                try {
                    Thread.sleep(1);                            //线程休眠 0.001 秒，模拟兔子在跑步
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
                rabbitTextArea.setText(text + "兔子到达终点\n");    //显示兔子到达了终点
            }
        }
    }
}
```

说明

由于乌龟和兔子在比赛中的行为类似，因此不讲解乌龟的代码。

15.5 线程的优先级

视频讲解：光盘\TM\lx\15\线程的优先级.exe

每个线程都具有各自的优先级，线程的优先级可以在程序中表明该线程的重要性，如果有很多线程处于就绪状态，系统会根据优先级来决定首先使哪个线程进入运行状态。但这并不意味着低优先级的线程得不到运行，而只是它运行的几率比较小，如垃圾回收线程的优先级就较低。

Thread 类中包含的成员变量代表了线程的某些优先级，如 Thread.MIN_PRIORITY（常数 1）、Thread.MAX_PRIORITY（常数 2）、Thread.NORM_PRIORITY（常数 5）。其中每个线程的优先级都在 Thread.MIN_PRIORITY～Thread.MAX_PRIORITY 之间，在默认情况下其优先级都是 Thread.NORM_PRIORITY。每个新产生的线程都继承了父线程的优先级。

在多任务操作系统中，每个线程都会得到一小段 CPU 时间片运行，在时间结束时，将轮换另一个线程进入运行状态，这时系统会选择与当前线程优先级相同的线程予以运行。系统始终选择就绪状态下优先级较高的线程进入运行状态。图 15.13 表明了处于各个优先级状态下的线程的运行顺序。

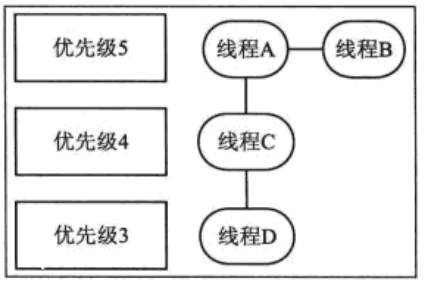


图 15.13 处于各个优先级状态下的线程的运行顺序

在图 15.13 中，优先级为 5 的线程 A 首先得到 CPU 时间片；当该时间结束后，轮换到与线程 A 相同优先级的线程 B；当线程 B 的运行时间结束后，会继续轮换到线程 A，直到线程 A 与线程 B 都执行完毕，才会轮换到线程 C；当线程 C 结束后，最后才会轮到线程 D。

线程的优先级可以使用 setPriority()方法调整，如果使用该方法设置的优先级不在 1～10 之内，将产生一个 IllegalArgumentException 异常。

下面的实例演示了图 15.13 描述的状况，依然以进度条为例说明。

【例 15.7】 在项目中创建 PriorityTest 类，该类继承了 Runnable 类。创建 4 个进度条，分别由 4 个线程来控制，并且为这 4 个线程设置不同的优先级。（**实例位置：光盘\TM\sl\15\10**）

```
public class PriorityTest extends JFrame{
    ...//非关键代码省略
    public PriorityTest() {
        ...//非关键代码省略
        threadA=new Thread(new MyThread(progressBar));        //分别实例化 4 个线程
        threadB=new Thread(new MyThread(progressBar2));
        threadC=new Thread(new MyThread(progressBar3));
```

```
        threadD=new Thread(new MyThread(progressBar4));
        setPriority("threadA", 5, threadA);
        setPriority("threadB", 5, threadB);
        setPriority("threadC", 4, threadC);
        setPriority("threadD", 3, threadD);
    }
    //定义设置线程的名称、优先级的方法
    public static void setPriority(String threadName,int priority,Thread t){
        t.setPriority(priority);                                    //设置线程的优先级
        t.setName(threadName);                                      //设置线程的名称
        t.start();                                                  //启动线程
    }
    public static void main(String[] args) {
        init(new PriorityTest(),100,100);
    }
    ...//非关键代码省略
    private final class MyThread implements Runnable {              //定义一个实现 Runnable 接口的类
        private final JProgressBar bar;
        int count=0;
        private MyThread(JProgressBar bar) {
            this.bar = bar;
        }
        public void run(){                                          //重写 run()方法
            while(true){
                bar.setValue(count+=10);                            //设置滚动条的值每次自增 10
                try{
                    Thread.sleep(1000);
                }catch(InterruptedException e){
                    System.out.println("当前线程被中断");
                }
            }
        }
    }
}
```

运行结果如图 15.14 所示。

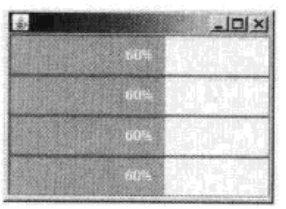


图 15.14　线程的优先级

在本实例中，定义了 4 个线程，这 4 个线程用于设置 4 个进度条的进度。笔者定义了 setPriority() 方法来设置每个线程的优先级和名称等。虽然在图 15.14 中看这 4 个进度条好像是在一起滚动，但如果仔细观察还是可以看到细微差别，可以看到第一个进度条总是最先变化，由于 threadA 线程和 threadB 线程优先级最高，所以系统首先处理这两个线程，然后是 threadC 和 threadD 这两个线程。

15.6 线程同步

视频讲解：光盘\TM\lx\15\线程同步.exe

在单线程程序中，每次只能做一件事情，后面的事情需要等待前面的事情完成后才可以进行；但是如果使用多线程程序，就会发生两个线程抢占资源的问题，例如两个人同时说话、两个人同时过同一个独木桥等。所以在多线程编程中，需要防止这些资源访问的冲突。Java 提供线程同步的机制来防止资源访问的冲突。

15.6.1 线程安全

实际开发中，使用多线程程序的情况很多，如银行排号系统、火车站售票系统等。这种多线程的程序通常会发生问题，以火车站售票系统为例，在代码中判断当前票数是否大于 0，如果大于 0 则执行将该票出售给乘客功能，但当两个线程同时访问这段代码时（假如这时只剩下一张票），第一个线程将票售出，与此同时第二个线程也已经执行完成判断是否有票的操作，并得出结论票数大于 0，于是它也执行售出操作，这样就会产生负数。所以在编写多线程程序时，应该考虑到线程安全问题。实质上线程安全问题来源于两个线程同时存取单一对象的数据。

【例 15.8】 在项目中创建 ThreadSafeTest 类，该类继承了 Runnable 类，主要实现模拟火车站售票系统。（**实例位置：光盘\TM\sl\15\11**）

```
public class ThreadSafeTest implements Runnable {
    int num=10;                                          //设置当前总票数
    public void run(){
        while(true){
            if(tnum>0){
                try{
                    Thread.sleep(100);
                }catch(Exception e){
                    e.printStackTrace();
                }
                System.out.println("车票还有 " + num--+" 张");
            }
        }
    }
    public static void main(String[] args) {
        ThreadSafeTest t=new ThreadSafeTest();           //实例化类对象
        Thread tA=new Thread(t);                         //以该类对象分别实例化 4 个线程
        Thread tB=new Thread(t);
        Thread tC=new Thread(t);
        Thread tD=new Thread(t);
        tA.start();                                      //分别启动线程
        tB.start();
```

```
        tC.start();
        tD.start();
    }
}
```

运行结果如图 15.15 所示。

从图 15.15 中可以看到，最后打印售出后剩下的票为负值，这样就出现了问题。这是由于同时创建了 4 个线程，这 4 个线程执行 run()方法，在 num 变量为 1 时，线程 1、线程 2、线程 3、线程 4 都对 num 变量有存储功能，当线程 1 执行 run()方法时，还没有来得及做递减操作，笔者就指定它调用 sleep()方法进入就绪状态，这时线程 2、线程 3、线程 4 都进入了 run()方法，发现 num 变量依然大于 0，但此时线程 1 休眠时间已到，将 num 变量值递减，同时线程 2、线程 3、线程 4 也都对 num 变量进行递减操作，从而就产生了负值。

图 15.15　资源共享冲突后出现的问题

15.6.2　线程同步机制

如何解决资源共享的问题？基本上所有解决多线程资源冲突问题都会采用给定时间只允许一个线程访问共享资源，这时就需要给共享资源上一道锁。这就好比一个人上洗手间，这个人进入洗手间后将门锁上，当他出来时再将锁打开，然后其他人才可以进入。

1．同步块

在 Java 中提供了同步机制，可以有效地防止资源冲突。同步机制使用 synchronized 关键字。

【例 15.9】　创建 CopyOf ThreadSafeTest 类，在该类中修改例 15.8 中的 run()方法，使对 num 操作的代码设置在同步块中。（**实例位置：光盘\TM\sl\15\12**）

```
public class CopyOfThreadSafeTest implements Runnable {
    int num=10;
        public void run(){
            while(true){
                synchronized (""){
                    if(num>0){
                        try{
                            Thread.sleep(1000);
                        }catch(Exception e){
                            e.printStackTrace();
                        }
                        System.out.println("tickets"+--num);
                    }
                }
            }
        }
    public static void main(String[] args) {
```

```
        CopyOfThreadSafeTest t=new CopyOfThreadSafeTest();
        Thread tA=new Thread(t);
        Thread tB=new Thread(t);
        Thread tC=new Thread(t);
        Thread tD=new Thread(t);
        tA.start();
        tB.start();
        tC.start();
        tD.start();
    }
}
```

运行结果如图 15.16 所示。

图 15.16　修改例 15.8 中的 run()方法

从图 15.16 中可以看出，打印到最后票数没有出现负数，这是因为将资源放置在了同步块中。这个同步块也被称为临界区，它使用 synchronized 关键字建立。其语法格式如下：

```
synchronized(Object){
    ...//
}
```

通常是将共享资源的操作放置在 synchronized 定义的区域内，这样当其他线程也获取到这个锁时，必须等待锁被释放时才能进入该区域。Object 为任意一个对象，每个对象都存在一个标志位，并具有两个值，分别为 0 和 1。一个线程运行到同步块时首先检查该对象的标志位，如果为 0 状态，表明此同步块中存在其他线程在运行。这时该线程处于就绪状态，直到处于同步块中的线程执行完同步块中的代码为止。这时该对象的标识位被设置为 1，该线程才能执行同步块中的代码，并将 Object 对象的标识位设置为 0，防止其他线程执行同步块中的代码。

2. 同步方法

同步方法就是在方法前面修饰 synchronized 关键字的方法，其语法格式如下：

```
synchronized void f(){}
```

当某个对象调用了同步方法时，该对象上的其他同步方法必须等待该同步方法执行完毕才能被执行。必须将每个能访问共享资源的方法修饰为 synchronized，否则就会出错。

修改例 15.9，将共享资源操作放置在一个同步方法中，如例 15.10 所示。

【例 15.10】　在项目中创建一个类文件，在该类中定义同步方法。

```
public synchronized void doit() {                    //定义同步方法
    if(num>0){
        try{
            Thread.sleep(10);
        }catch(Exception e){
            e.printStackTrace();
        }
        System.out.println("tickets"+--num);
    }
}
public void run(){
    while(true){
        doit();                                      //在 run()方法中调用该同步方法
    }
}
```

将共享资源的操作放置在同步方法中，运行结果与使用同步块的结果一致。

15.7　线程间的通信

视频讲解：光盘\TM\lx\15\线程间的通信.exe

在学习完如何避免线程产生冲突的问题后，下面将学习线程之间的通信。线程之间的通信使用 wait()、notify()以及 notifyAll()方法实现。

我们知道线程如果调用 wait()方法后可以使该线程从运行状态进入就绪状态，而 sleep()方法也达到这样一个效果，那么两者究竟有何区别？从同步的角度上来说，调用 sleep()方法的线程不释放锁，但调用 wait()方法的线程释放锁。

使用 wait()方法有以下两种形式：

- ☑ wait(time)
- ☑ wait()

第一种形式的 wait()方法与 sleep()方法的含义相同，都是指在此时间之内暂停；而第二种形式的 wait()方法会使线程永久无限地等待下去，需要使用 notify()或者 notifyAll()方法唤醒。

注意

wait()与 notify()、notifyAll()方法只能在同步块或同步方法中使用。

下面举一个进度条的例子，同时设计一个使进度条递增的线程和一个使进度条递减的线程，使用 wait()和 notify()方法控制。

【例 15.11】　在项目中创建 Commnicate 类，该类继承了 Runnable 类。创建一个进度条，使用线程的 wait()与 notify()方法控制进度条的进度。（**实例位置：光盘\TM\sl\15\13**）

```
public class Commnicate extends JFrame{
    ...//非关键性代码省略
    public static void main(String[] args) {
        init(new Commnicate(),150,100);
    }
    public Commnicate() {
        ...//非关键代码省略
        deValue();
        addValue();
        t1.start();
        t2.start();
    }
    public void addValue(){
        t1=new Thread(new Runnable(){                                   //实例化线程 t1
            public void run(){
                while(true){
                    if(count>=100){
                        System.out.println("进度条已满，递增线程等待");
                        break;
                    }
                    if (count == 0) {
                        progressBar.setValue(count += 100);
                        System.out.println("进度条的当前值为："+count);
                        synchronized (t2) {                             //定义同步块
                            System.out.println("进度条已有值，可以进行递减操作");
                            t2.notify();                                //在同步块中将线程 t2 唤醒
                        }
                    }
                    try{
                        Thread.currentThread().sleep(100);              //使当前线程休眠 100 毫秒
                    }catch(Exception e){
                        e.printStackTrace();
                    }
                }
            }
        });
    }
    public void deValue(){
        t2=new Thread(new Runnable(){                                   //定义线程 t2
            public void run(){
                while (true) {
                    if(count==0){
                        synchronized (this) {                           //定义同步块
                            try {
                                wait();                                 //使线程 t2 等待
                            } catch (Exception e) {
                                e.printStackTrace();
                            }
                        }
```

```
                    }
                    progressBar.setValue(--count);                    //将进度条的当前值递减
                    System.out.println("进度条的当前值为：" + count);
                    try {
                        Thread.currentThread().sleep(100);
                    } catch (Exception e) {
                        e.printStackTrace();
                    }
                }
            }
        });
    }
    ...//非关键代码省略
}
```

运行结果如图 15.17 所示。

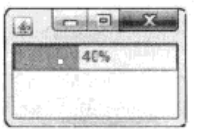


图 15.17　运行结果

15.8　经 典 范 例

15.8.1　经典范例 1：线程的插队运行

教学录像：光盘\MR\lx\15\线程的插队运行.exe

在编写多线程程序时，会遇到让一个线程优先于其他线程运行的情况。此时除了可以设置该线程的优先级高于其他线程外，更直接的方式是使用 Thread 类的 join()方法。本实例将演示该方法的实际效果。运行结果如图 15.18 所示。（**实例位置：光盘\TM\sl\15\14**）

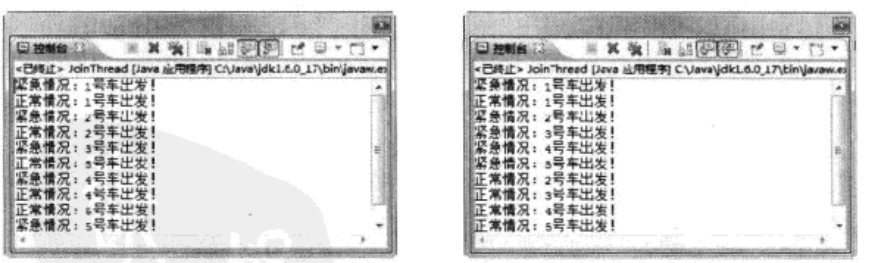


图 15.18　运行结果

（1）编写类 EmergencyThread，该类实现了 Runnable 接口。在 run()方法中，每隔 0.1 秒输出一条语句。代码如下：

```
public class EmergencyThread implements Runnable {
    @Override
    public void run() {
        for (int i = 1; i < 6; i++) {
            try {
                Thread.sleep(100);                          //当前线程休眠 0.1 秒实现动态更新
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            System.out.println("紧急情况：" + i + "号车出发！");    //紧急情况下车辆出发
        }
    }
}
```

（2）编写类 JoinThread 用来进行测试，在该类中使用 EmergencyThread 创建并运行新的线程。使用 join()方法让新线程优先于当前线程运行，代码如下：

```
public class JoinThread {
    public static void main(String[] args) {
        Thread thread = new Thread(new EmergencyThread());    //创建新线程
        thread.start();                                       //运行新线程
        for (int i = 1; i < 6; i++) {
            try {
                Thread.sleep(100);                            //当前线程休眠 0.1 秒实现动态更新
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            System.out.println("正常情况：" + i + "号车出发！");    //正常情况下车辆出发
            try {
                thread.join();                                //使用 join()方法让新创建的线程优先完成
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

15.8.2 经典范例 2：查看和修改线程名称

视频讲解：光盘\TM\lx\15\查看和修改线程名称.exe

Java 中所有线程都有一个默认的名称。对于用户自定义的线程，默认的名称格式是“Thread-数字”。当然，用户可以在创建新线程时指定线程的名称，也可以修改已有线程的名称。本范例将编写一个简单的 GUI 程序，实现定义和修改线程名称的功能。运行结果如图 15.19 所示。（**实例位置：光盘\TM\sl\15\15**）

图 15.19 运行结果

（1）编写 ThreadNameTest 类，该类继承了 JFrame。在框架中包含了一个表格，用来显示当前线程组中运行的线程。两个文本域用来获得用户输入的新线程名称和想修改成的名称。“新建线程”按钮用来实现使用指定或默认的名称创建线程的功能，“修改名称”按钮用来实现使用指定名称修改用户选择的线程名称的功能。

（2）编写 Forever 类，该类实现了 Runnable 接口，在 run()方法中使用了死循环。代码如下：

```
private class Forever implements Runnable {
    @Override
    public void run() {
        while (true) {                                  //死循环
        }
    }

}
```

（3）编写 do_this_windowActivated()方法，用来监听窗体激活事件。在该方法中，使用当前线程所在线程组中的线程 ID 和名称作为表格的数据。代码如下：

```
protected void do_this_windowActivated(WindowEvent e) {
    ThreadGroup group = Thread.currentThread().getThreadGroup();     //获得当前线程所在线程组
    Thread[] threads = new Thread[group.activeCount()];              //使用数组保存活动状态的线程
    group.enumerate(threads);                                        //获得所有线程
    DefaultTableModel model = (DefaultTableModel) table.getModel();  //获得表格模型
    model.setRowCount(0);                                            //清空表格模型中的数据
    model.setColumnIdentifiers(new Object[] { "线程 ID", "线程名称" });  //定义表头
    for (Thread thread : threads) {                                  //增加行数据
        model.addRow(new Object[] { thread.getId(), thread.getName() });
    }
    table.setModel(model);                                           //更新表格模型
}
```

（4）编写 do_button1_actionPerformed()方法，用来监听单击“新建线程”按钮事件。在该方法中，根据用户是否输入新线程的名称创建新线程，并更新表格中的数据。代码如下：

```
protected void do_button1_actionPerformed(ActionEvent e) {
    Object[] newThread = null;
    String name = textField1.getText();                      //获得用户输入的名称
    if (name.isEmpty()) {                                    //如果用户没有输入则使用默认名称创建新线程
        Thread thread = new Thread(new Forever());
        thread.start();
        newThread = new Object[] { thread.getId(), thread.getName() };
    } else {                                                 //如果用户有输入则使用指定名称创建新线程
        Thread thread = new Thread(new Forever(), name);
        thread.start();
        newThread = new Object[] { thread.getId(), name };
    }
    ((DefaultTableModel) table.getModel()).addRow(newThread);   //更新表格中的数据
}
```

（5）编写 do_button2_actionPerformed()方法，用来监听单击“修改名称”按钮事件。在该方法中，获得了用户输入的名称和用户选择的行，根据用户输入的名称修改了线程名称。代码如下：

```
protected void do_button2_actionPerformed(ActionEvent e) {
    int selectedRow = table.getSelectedRow();        //获得用户选择的行
    String newName = textField2.getText();           //获得用户输入的名称
    if ((selectedRow == -1) || newName.isEmpty()) { //如果没有选择或输入为空则直接退出该方法
        return;
    }
    DefaultTableModel model = (DefaultTableModel) table.getModel();//获得表格模型
    model.setValueAt(newName, selectedRow, 1);  //更改表格中的数据
    repaint();                                  //重新绘制各个控件
}
```

15.9 本章小结

本章讲述了线程，通过对其学习读者应该掌握线程与 Swing 技术相结合使用的方法。

学习多线程编程就像进入了一个全新的领域，它与以往的编程思想截然不同，随着大多数操作系统对多线程的支持，很多程序语言都已支持和扩展多线程，作为初学者应该积极转换编程思维，使自己的编程思想进入多线程编程的思维方式。多线程本身是一种非常复杂的机制，完全理解它也需要一段时间，并且需要深入地学习。本章将多线程与 Swing 技术联系在一起，列举了大量实例，使读者从实例中体会多线程机制，为读者掌握多线程的基础知识打下了坚实的基础，从而深刻理解其概念并编写出合理的多线程程序。

15.10 实战练习

1. 尝试定义一个继承 Thread 类的类，并覆盖 run()方法，在 run()方法中每隔 100 毫秒打印一句话。**（答案位置：光盘\TM\sl\15\16）**

2. 尝试开发一个窗体，在窗体中有两个按钮，一个是“开始”按钮，一个是“结束”按钮，当用户单击“开始”按钮时，在控制台中持续打印一段话，当用户单击“停止”按钮时，控制台结束打印。**（答案位置：光盘\TM\sl\15\17）**

3. 尝试开发一个窗体，在窗体中设计一个进度条，使进度条每次递增滚动。**（答案位置：光盘\TM\sl\15\18）**

第16章

网络通信

（视频讲解：33分钟）

Internet 提供了大量、多样的信息，很少有人接触过 Internet 后能拒绝它的诱惑。计算机网络实现了多个计算机互联系统，相互连接的计算机之间彼此能够进行数据交流。网络应用程序就是在已连接的不同计算机上运行的程序，这些程序相互之间可以交换数据。而编写网络应用程序，首先必须明确网络应用程序所要使用的网络协议，TCP/IP 协议是网络应用程序的首选。本章从介绍网络协议开始，向读者介绍 TCP 网络程序和 UDP 网络程序。

通过阅读本章，您可以：

- 了解常见的几种网络协议
- 理解端口和套接字
- 掌握 InetAddress 类的应用
- 掌握 ServerSocket 类的应用
- 学习编写 TCP 网络程序
- 掌握 DatagramPacket 类的应用
- 掌握 DatagramSocket 类的应用
- 学习编写 UDP 程序

PDG

16.1 网络程序设计基础

视频讲解：光盘\TM\lx\16\网络程序设计基础.exe

网络程序设计是指编写与其他计算机进行通信的程序。Java 已经将网络程序所需要的东西封装成不同的类。只要创建这些类的对象，使用相应的方法，即使设计人员不具备有关的网络知识，也可以编写出高质量的网络通信程序。

16.1.1 局域网与因特网

为了实现两台计算机的通信，必须要用一个网络线路连接两台计算机，如图 16.1 所示。

图 16.1 服务器、客户机和网络

服务器是指提供信息的计算机或程序，客户机是指请求信息的计算机或程序，而网络用于连接服务器与客户机，实现两者相互通信。但有时在某个网络中很难将服务器与客户机区分开。通常所说的“局域网”（Local Area Network，LAN），就是一群通过一定形式连接起来的计算机。它可以由两台计算机组成，也可以由同一区域内的上千台计算机组成。由 LAN 延伸到更大的范围，这样的网络称为“广域网”（Wide Area Network，WAN）。人们熟悉的因特网（Internet），则是由无数的 LAN 和 WAN 组成。

16.1.2 网络协议

网络协议规定了计算机之间连接的物理、机械（网线与网卡的连接规定）、电气（有效的电平范围）等特征以及计算机之间的相互寻址规则、数据发送冲突的解决、长的数据如何分段传送与接收等。就像不同的国家有不同的法律一样，目前网络协议也有多种，下面简单地介绍几个常用的网络协议。

1. IP 协议

IP 其实是 Internet Protocol 的简称，由此可知它就是一种“网络协议”。Internet 网络采用的协议是 TCP/IP 协议，其全称是 Transmission Control Protocol/Internet Protocol。Internet 依靠 TCP/IP 协议，在全球范围内实现不同硬件结构、不同操作系统、不同网络系统的互联。在 Internet 网上存在数以亿计的主机，每一台主机在网络上通过为其分配的 Internet 地址表示自己，这个地址就是 IP 地址。到目前为止 IP 地址用 4 个字节，也就是 32 位的二进制数来表示，称为 IPv4。为了便于使用，通常取用每个字节的十进制数，并且每个字节之间用圆点隔开来表示 IP 地址，如 192.168.1.1。现在人们正在试验使用 16 个字节来表示 IP 地址，这就是 IPv6，但 IPv6 还没有投入使用。

TCP/IP 模式是一种层次结构，共分为 4 层，分别为应用层、传输层、互联网层和主机到网络层。

各层实现特定的功能，提供特定的服务和访问接口，并具有相对的独立性，如图 16.2 所示。

2. TCP 与 UDP 协议

在 TCP/IP 协议栈中，有两个高级协议是网络应用程序编写者应该了解的，即“传输控制协议”（Transmission Control Protocol，TCP）与“用户数据报协议”（User Datagram Protocol，UDP）。

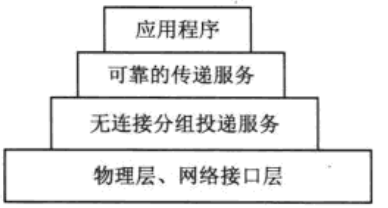


图 16.2　TCP/IP 层次结构

TCP 协议是一种以固接连线为基础的协议，它提供两台计算机间可靠的数据传送。TCP 可以保证数据从一端送至连接的另一端时能够确实送达，而且抵达的数据的排列顺序和送出时的顺序相同，因此 TCP 协议适合可靠性要求比较高的场合。就像拨打电话一样，必须先拨号给对方，等两端确定连接后，相互能听到对方的说话，也知道对方回应的是什么。

HTTP、FTP 和 Telnet 等都需要使用可靠的通信频道，例如 HTTP 从某个 URL 读取数据时，如果收到的数据顺序与发送时不相同，可能就会出现一个混乱的 HTML 文件或是一些无效的信息。

UDP 是无连接通信协议，不保证可靠数据的传输，但能够向若干个目标发送数据，接收发自若干个源的数据。UDP 是以独立发送数据包的方式进行。这种方式就像邮递员送信给收信人，可以寄出很多信给同一个人，而每一封信都是相对独立的，各封信送达的顺序并非重要，而收信人接收信件的顺序也不能保证与寄出信件的顺序相同。

UDP 协议适合于一些对数据准确性要求不高的场合，如网络聊天室、在线影片等。由于 TCP 协议在认证上存在额外耗费，有可能使传输速度减慢；UDP 协议可能会更适合这些对传输速度和时效要求非常高的网站，即使有一小部分数据包的遗失或传送顺序有所不同，并不会严重危害该项通信。

注意　一些防火墙和路由器会设置成不允许 UDP 数据包传输，因此若遇到 UDP 连接方面的问题，应先确定是否允许 UDP 协议。

16.1.3　端口和套接字

一般而言，一台计算机只有单一的连到网络的“物理连接”（Physical Connection），所有的数据都通过此连接对内、对外送达特定的计算机，这就是端口。网络程序设计中的端口（port）并非真实的物理存在，而是一个假想的连接装置。端口被规定为一个在 0～65535 之间的整数。HTTP 服务一般使用 80 端口，FTP 服务使用 21 端口。假如一台计算机提供了 HTTP、FTP 等多种服务，那么客户机通过不同的端口来确定连接到服务器的哪项服务上，如图 16.3 所示。

说明　通常 0～1023 之间的端口数用于一些知名的网络服务和应用，用户的普通网络应用程序应该使用 1024 以上的端口数，以避免端口号与另一个应用或系统服务所用端口冲突。

网络程序中套接字（Socket）用于将应用程序与端口连接起来。套接字是一个假想的连接装置，就

像插插头的设备“插座”，用于连接电器与电线，如图 16.4 所示。Java 将套接字抽象化为类，程序设计者只需创建 Socket 类对象，即可使用套接字。

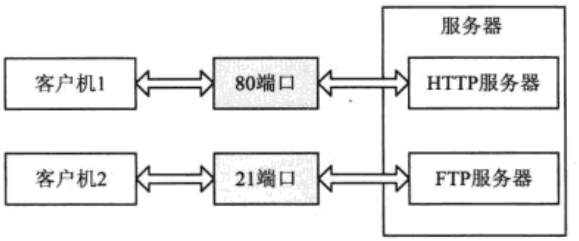


图 16.3　端口

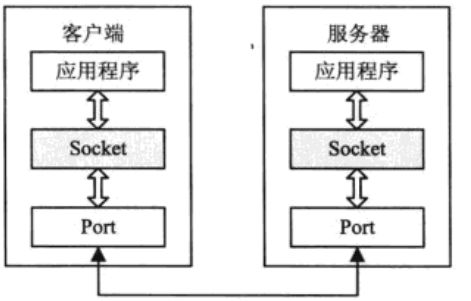


图 16.4　套接字

16.1.4　范例 1：获得内网的所有 IP 地址

在进行网络编程时，有时需要对局域网内的所有主机进行遍历，为此需要获得内网的所有 IP 地址。本范例将演示如何在 Java 应用程序中获得内网的所有 IP 地址。运行程序，单击“显示所有 IP”按钮，将在文本域中显示内网中所有主机的 IP 地址。运行结果如图 16.5 所示。（**实例位置：光盘\TM\sl\16\1**）

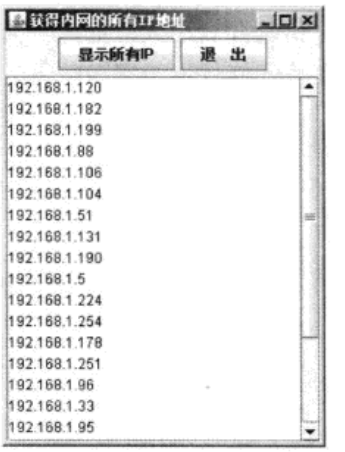


图 16.5　内网的所有 IP 地址

（1）在 com.mingrisoft 包中创建一个继承自 JFrame 类的窗体类 GainAllIpFrame。

（2）在 GainAllIpFrame 窗体的内容面板顶部位置放一个面板控件，并在该面板控件上放两个按钮，然后在内容面板的中部位置放一个滚动面板控件，并在滚动面板上放一个文本域控件，完成窗体界面的设置。

（3）在 GainAllIpFrame 窗体类中定义一个 gainAllIp()方法，用于获得内容中的所有 IP 地址，并追加到文本域中显示。

（4）在 GainAllIpFrame 窗体类中定义一个 PingIpThread 线程类，用于 Ping 指定的 IP 地址，并判断其是否为有效的内网 IP 地址，如果是就添加到集合对象中。代码如下：

```
public String ip;                                          //表示 IP 地址的成员变量
public PingIpThread(String ip) {                           //参数为需要判断的 IP 地址
    this.ip = ip;
}
public void run() {
    try {
        //获得所 Ping 的 IP 进程，-w 280 是等待每次回复的超时时间，-n 1 是要发送的回显请求数
        Process process = Runtime.getRuntime().exec(
                "ping " + ip + " -w 280 -n 1");
```

```
        InputStream is = process.getInputStream();                    //获得进程的输入流对象
        InputStreamReader isr = new InputStreamReader(is);            //创建 InputStreamReader 对象
        BufferedReader in = new BufferedReader(isr);                  //创建缓冲字符流对象
        String line = in.readLine();                                  //读取信息
        while (line != null) {
            if (line != null && !line.equals("")) {
                if (line.substring(0, 2).equals("来自")
                        || (line.length() > 10 && line.substring(0, 10)
                                .equals("Reply from"))) {             //判断是 Ping 通过的 IP 地址
                    pingMap.put(ip, "true");                          //向集合中添加 IP
                }
            }
            line = in.readLine();                                     //再读取信息
        }
    } catch (IOException e) {
    }
}
```

16.1.5　范例 2：解析网页中的内容

本范例演示了如何在 Java 应用程序中解析网页的内容。运行程序，在输入网址标签右侧的文本框中输入网址，单击“解析网页”按钮，将在文本域中显示解析的网页内容。运行结果如图 16.6 所示。**（实例位置：光盘\TM\sl\16\2）**

图 16.6　解析网页中的内容

（1）在 Eclipse 中新建项目 02，在项目中创建 com.mingrisoft 包。

（2）在 com.mingrisoft 包中创建一个继承 JFrame 类的 InternetContentFrame 窗体类。

（3）在 InternetContentFrame 窗体类中创建用于解析网页内容的 getURLConnection()方法，代码如下：

```
public Collection<String> getURLConnecnnon(String urlString){
    URL url = null;                                          //声明 URL
    URLConnection conn = null;                               //声明 URLConnection
    Collection<String> urlCollection = new ArrayList<String>();  //创建集合对象
    try{
        url = new URL(urlString);                            //创建 URL 对象
        conn = url.openConnection();                         //获得连接对象
        conn.connect();                                      //打开到 URL 引用资源的通信链接
        InputStream is = conn.getInputStream();              //获取流对象
        //转换为字符流
        InputStreamReader in = new InputStreamReader(is,"UTF-8");
        BufferedReader br = new BufferedReader(in);          //创建缓冲流对象
        String nextLine = br.readLine();                     //读取信息，解析网页
        while (nextLine !=null){
            urlCollection.add(nextLine);                     //解析网页的全部内容，添加到集合中
            nextLine = br.readLine();                        //读取信息，解析网页
        }
    }catch(Exception ex){
        ex.printStackTrace();
    }
    return urlCollection;
}
```

（4）在 InternetContentFrame 窗体类中为“解析网页”按钮添加事件代码，实现调用 getURLConnection()方法解析网页内容并显示在文本域中。代码如下：

```
button.addActionListener(new ActionListener() {
    public void actionPerformed(final ActionEvent e) {
        String address = tf_address.getText().trim();        //获得输入的网址
        Collection urlCollection = getURLCollection(address);  //调用方法，获得网页内容的集合对象
        Iterator it = urlCollection.iterator();              //获得集合的迭代器对象
        while(it.hasNext()){
            ta_content.append((String)it.next()+"\n");       //在文本域中显示解析的内容
        }
    }
});
```

16.2 TCP 程序设计基础

视频讲解：光盘\TM\lx\16\TCP 程序设计基础.exe

TCP 网络程序设计是指利用 Scoket 类编写通信程序。利用 TCP 协议进行通信的两个应用程序是有主次之分的，一个称为服务器程序，另一个称为客户机程序。两者的功能和编写方法大不一样。服务

器端与客户端的交互过程如图 16.7 所示。

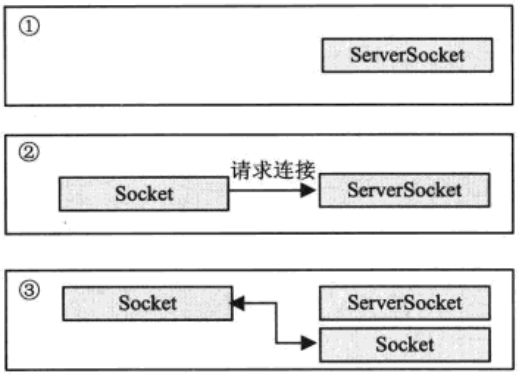


图 16.7　服务器端与客户端的交互

①——服务器程序创建一个 ServerSocket（服务器端套接字），调用 accept()方法等待客户机来连接。

②——客户端程序创建一个 Socket，请求与服务器建立连接。

③——服务器接收客户机的连接请求，同时创建一个新的 Socket 与客户建立连接。服务器继续等待新的请求。

16.2.1　InetAddress 类

java.net 包中的 InetAddress 类是与 IP 地址相关的类，利用该类可以获取 IP 地址、主机地址等信息。InetAddress 类的常用方法如表 16.1 所示。

表 16.1　InetAddress 类的常用方法

返　回　值	方　　法	说　　明
InetAddress	getByName(String host)	获取与 host 相对应的 InetAddress 对象
String	getHostAddress()	获取 InetAddress 对象所含的 IP 地址
String	getHostName()	获取此 IP 地址的主机名
InetAddress	getLocalHost()	返回本地主机的 InetAddress 对象

说明　这里只给出了 InetAddress 类的常用方法，该类还提供了许多其他方法，如果需要其他方法，可以查询 JDK 的 API 文档。

【例 16.1】　使用 InetAddress 类的 getHostName()和 getHostAddress()方法获得本地主机名、本机 IP 地址。（**实例位置：光盘\TM\sl\16\3**）

```
import java.net.InetAddress;
import java.net.UnknownHostException;
public class Address {                                    //创建类
    public static void main(String[] args) {
```

```
        InetAddress ip;                                          //声明 InetAddress 对象
        try {                                                    //try 语句块捕捉可能出现的异常
            ip = InetAddress.getLocalHost();                     //实例化 InetAddress 对象
            String localname = ip.getHostName();                 //获取本地主机名
            String localip = ip.getHostAddress();                //获取本地主机的 IP 地址
            System.out.println("本地主机名：" + localname);       //输出本地主机名
            System.out.println("本机 IP 地址：" + localip);       //输出本地主机的 IP 地址
        } catch (UnknownHostException e) {
            e.printStackTrace();                                 //输出异常信息
        }
    }
}
```

运行结果如图 16.8 所示。

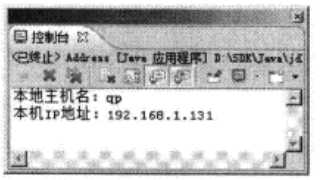


图 16.8　获得本地主机名和 IP 地址

注意 InetAddress 类的方法会抛出 UnknownHostException 异常，所以必须进行异常处理。这个异常在主机不存在或网络连接错误时发生。

16.2.2　ServerSocket 类

java.net 包中的 ServerSocket 类用于表示服务器套接字，其主要功能是等待来自网络上的“请求”，它可通过指定的端口来等待连接的套接字。服务器套接字一次可以与一个套接字连接。如果多台客户机同时提出连接请求，服务器套接字会将请求连接的客户机存入队列中，然后从中取出一个套接字，与服务器新建的套接字连接起来。若请求连接数大于最大容纳数，则多出的连接请求被拒绝。队列的默认大小为 50。

ServerSocket 类的构造方法都抛出 IOException 异常，分别有以下几种形式。

- ☑ ServerSocket()：创建非绑定服务器套接字。
- ☑ ServerSocket(int port)：创建绑定到特定端口的服务器套接字。
- ☑ ServerSocket(int port, int backlog)：利用指定的 backlog 创建服务器套接字并将其绑定到指定的本地端口号。
- ☑ ServerSocket(int port, int backlog, InetAddress bindAddress)：使用指定的端口、侦听 backlog 和要绑定到的本地 IP 地址创建服务器。这种情况适用于计算机上有多块网卡和多个 IP 的情况，可以明确规定 ServerSocket 在哪块网卡或 IP 地址上等待客户的连接请求。

ServerSocket 类的常用方法如表 16.2 所示。

表 16.2　ServerSocket 类的常用方法

返　回　值	方　　法	说　　明
Socket	accept()	等待客户机的连接。若连接，则创建一套接字
boolean	isBound()	判断 ServerSocket 的绑定状态
InetAddress	getInetAddress()	返回此服务器套接字的本地地址
boolean	isClosed()	返回服务器套接字的关闭状态
void	close()	关闭服务器套接字
void	bind(SocketAddress endpoint)	将 ServerSocket 绑定到特定地址（IP 地址和端口号）
int	getInetAddress()	返回服务器套接字等待的端口号

说明　这里只给出了 ServerSocket 类的常用方法，该类还提供了许多其他方法，在使用时可以查询 JDK 的 API 文档。

调用 ServerSocket 类的 accept()方法，将返回一个与客户端 Socket 对象相连接的 Socket 对象，服务器端的 Socket 对象使用 getOutputStream()方法获得的输出流对象，将指向客户端 Socket 对象使用 getInputStream()方法获得的输入流对象；同样，服务器端的 Socket 对象使用 getInputStream()方法获得的输入流对象，将指向客户端 Socket 对象使用 getOutputStream()方法获得的输出流对象。也就是说当服务器向输出流写入信息时，客户端通过相应的输入流就能读取。反之亦然。

注意　accept()方法会阻塞线程的继续执行，直到接收到客户的呼叫。如果没有客户呼叫服务器，那么下面的代码“System.out.println("连接中")”将不会执行。如果没有客户请求的情况下 accept()方法没有发生阻塞，则一定是程序出现了问题，这种问题通常是使用了一个还在被其他程序占用的端口号，ServerSocket 绑定没有成功。

```
yu = server.accept();
System.out.println("连接中");
```

16.2.3　TCP 网络程序

知道了 TCP 程序的工作过程，就可以编写 TCP 服务器程序了。在网络编程中如果只要求客户机向服务器发送消息，不用服务器向客户机发送消息，称为单向通信。客户机套接字和服务器套接字连接成功后，客户机通过输出流发送数据，而服务器会使用输入流接收数据。下面是简单的单向通信的实例。

【例 16.2】　本实例是一个 TCP 服务器端程序，在 getserver()方法中建立服务器套接字，调用 getClientMessage()方法获取客户端信息。（**实例位置：光盘\TM\sl\16\4**）

```
public class MyTcp {                                    //创建 MyTcp 类
    private BufferedReader reader;                      //创建 BufferedReader 对象
    private ServerSocket server;                        //创建 ServerSocket 对象
```

```
    private Socket socket;                                          //创建 Socket 对象 socket
    void getserver() {
        try {
            server = new ServerSocket(8998);                        //实例化 Socket 对象
            System.out.println("服务器套接字已经创建成功");          //输出信息
            while (true) {                                          //如果套接字是连接状态
                System.out.println("等待客户机的连接");              //输出信息
                socket = server.accept();                           //实例化 Socket 对象
                reader = new BufferedReader(new InputStreamReader(socket
                        .getInputStream()));                        //实例化 BufferedReader 对象
                getClientMessage();                                 //调用 getClientMessage()方法
            }
        } catch (Exception e) {
            e.printStackTrace();                                    //输出异常信息
        }
    }
    private void getClientMessage() {
        try {
            while (true) {                                          //如果套接字是连接状态
                System.out.println("客户机：" + reader.readLine());  //获得客户端信息
            }
        } catch (Exception e) {
            e.printStackTrace();                                    //输出异常信息
        }
        try {
            if (reader != null) {
                reader.close();                                     //关闭流
            }
            if (socket != null) {
                socket.close();                                     //关闭套接字
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    public static void main(String[] args) {                        //主方法
        MyTcp tcp = new MyTcp();                                    //创建本类对象
        tcp.getserver();                                            //调用方法
    }
}
```

运行结果如图 16.9 所示。

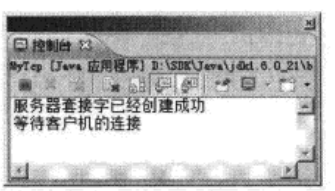


图 16.9　服务器端运行结果

注意　本实例是服务器端程序，当本实例运行后，如果在服务器没有停止运行的情况下再次运行本实例将发生异常，这是由于服务器程序使用的端口号已经被占用，所以在服务器程序没有停止的情况下再次运行就发生了异常。

从图 16.9 中可以看出，服务器端程序运行后，将输出提示信息，并等待客户机的连接。下面再来看一下客户端程序。

【例 16.3】　客户端程序，实现将用户在文本框中输入的信息发送至服务器端，并将文本框中输入的信息显示在客户端的文本域中。（**实例位置：光盘\TM\sl\16\5**）

```
public class MyClien extends JFrame {                                  //创建类继承 JFrame 类
    private PrintWriter writer;                                        //声明 PrintWriter 类对象
    Socket socket;                                                     //声明 Socket 对象
    private JTextArea ta = new JTextArea();                            //创建 JTextArea 对象
    private JTextField tf = new JTextField();                          //创建 JTextField 对象
    Container cc;                                                      //声明 Container 对象
    public MyClien(String title) {                                     //构造方法
        super(title);                                                  //调用父类的构造方法
        cc = this.getContentPane();                                    //实例化对象
        cc.add(ta, "North");                                           //将文本域放在窗体的上部
        cc.add(tf, "South");                                           //将文本框放在窗体的下部
        tf.addActionListener(new ActionListener() {                    //绑定事件
                public void actionPerformed(ActionEvent e) {
                    writer.println(tf.getText());                      //将文本框中信息写入流
                    ta.append(tf.getText() + '\n');                    //将文本框中信息显示在文本域中
                    tf.setText(" ");                                   //将文本框清空
                }
            });
    }
    private void connect() {                                           //连接套接字方法
        ta.append("尝试连接\n");                                       //文本域中提示信息
        try {                                                          //捕捉异常
            socket = new Socket("192.168.1.131", 8998);                //实例化 Socket 对象
            writer = new PrintWriter(socket.getOutputStream(), true);  //获得输出流对象
            ta.append("完成连接\n");                                   //文本域中提示信息
        } catch (Exception e) {
            e.printStackTrace();                                       //输出异常信息
        }
    }
    public static void main(String[] args) {                           //主方法
        MyClien clien = new MyClien("向服务器送数据");                 //创建本例对象
        clien.setBounds(300,260,340,220);                              //设置窗体位置和大小
        clien.setVisible(true);                                        //将窗体显示
        clien.connect();                                               //调用连接方法
    }
}
```

运行结果如图 16.10 所示。

注意 为了使本实例能够正常运行，必须先运行例 16.3 的服务器端程序，并且实例的端口号要与例 16.3 中服务器套接字的端口号一致，否则程序会发生错误。

说明 当一台机器上安装了多个网络应用程序时，很可能指定的端口号已被占用。有时可能还会遇到以前很好的网络程序突然运行不起来了，这种情况很可能也是由于端口被其他程序占用了。此时可以运行 netstat -help 来获得帮助，使用命令 netstat -an 来查看该程序所使用的端口，如图 16.11 所示。

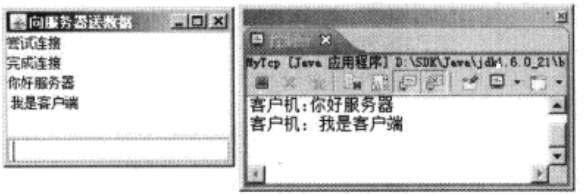


图 16.10　客户端的运行结果

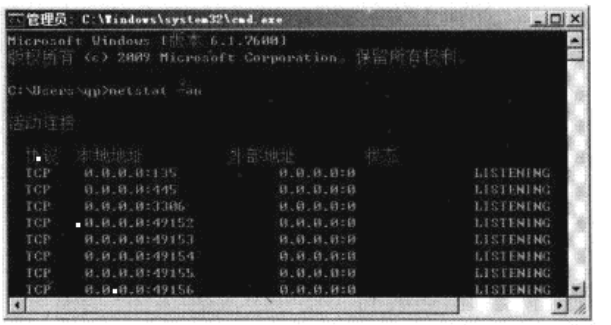


图 16.11　在命令行查看端口

16.2.4　范例 3：设置等待连接的超时时间

在进行网络编程时，由于进行网络连接是比较消耗资源的，因此，可以对连接的等待时间进行设置，如果在规定的时间没有进行连接，则进行其他处理。运行程序，等待 10 秒钟后，将弹出消息框提示连接超时。运行结果如图 16.12 所示。（**实例位置：光盘\TM\sl\16\6**）

（1）在 Eclipse 中新建项目 06，在项目中创建 com.mingrisoft 包。

（2）在 com.mingrisoft 包中创建一个继承自 JFrame 类的窗体类 ConnectionTimeoutSetFrame。

（3）在 ConnectionTimeoutSetFrame 窗体的内容面板上添加一个 JScrollPane 滚动面板控件，然后在滚动面板的视图上放置一个 JTextArea 文本域控件，文本域控件的名称为 ta_info，用于显示服务器等待客户端连接的相关信息，并显示等待连接超时的信息。

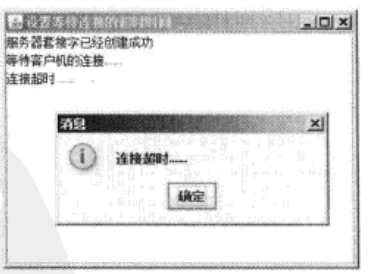


图 16.12　设置等待连接的超时时间

（4）实现创建服务器套接字和设置等待连接超时时间的代码定义在 getServer()方法中，代码如下：

```
public void getServer() {
    try {
        server = new ServerSocket(1978);                    //实例化 Socket 对象
```

```
        server.setSoTimeout(10000);                          //设置连接超时时间为 10 秒
        ta_info.append("服务器套接字已经创建成功\n");          //输出信息
        while (true) {                                       //如果套接字是连接状态
            ta_info.append("等待客户机的连接.....\n");        //输出信息
            server.accept();                                 //等待客户机连接
        }
    } catch (SocketTimeoutException e) {
        ta_info.append("连接超时.....");
        JOptionPane.showMessageDialog(null, "连接超时.....");
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

16.2.5　范例 4：获得 Socket 信息

本范例演示在网络通信程序中，如何获取远程服务器和客户机的 IP 地址与端口号。运行程序，结果如图 16.13 和图 16.14 所示，其中图 16.13 是服务器端程序，用于等待客户端的连接；图 16.14 是客户端程序，用于获得远程服务器和客户机的 IP 地址与端口号。（**实例位置：光盘\TM\sl\16\7**）

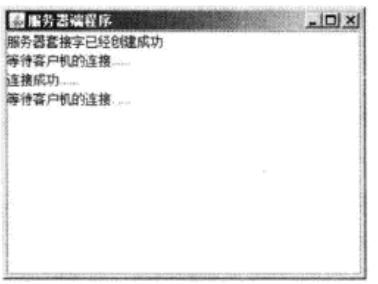


图 16.13　服务器端程序

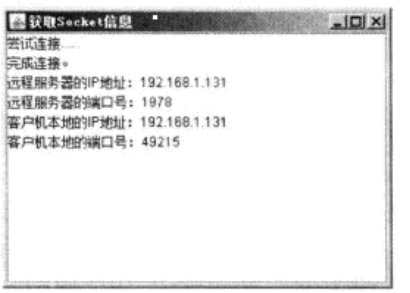


图 16.14　获取 Socket 信息

（1）在 Eclipse 中新建项目 07，在项目中创建 com.mingrisoft 包。

（2）在 com.mingrisoft 包中创建一个继承自 JFrame 类的服务器窗体类 ServerSocketFrame 和一个客户端窗体类 ClientSocketFrame。

（3）在服务器窗体类 ServerSocketFrame 的内容面板上添加一个 JScrollPane 滚动面板控件，然后在滚动面板的视图上放置一个 JTextArea 文本域控件。文本域控件的名称为 ta_info，用于显示客户端与服务器的连接信息。

（4）在客户端窗体类 ClientSocketFrame 的内容面板上添加一个 JScrollPane 滚动面板控件，然后在滚动面板的视图上放置一个 JTextArea 文本域控件。文本域控件的名称为 ta_info，用于显示远程服务器和客户机的 IP 地址与端口号。

（5）客户端窗体类 ClientSocketFrame 中，用于显示远程服务器和客户机的 IP 地址与端口号的代码定义在 connect()方法中，该方法的代码如下：

```
private void connect() {                                          //连接套接字方法
    ta.append("尝试连接.....\n");                                  //文本域中信息
    try {                                                         //捕捉异常
        socket = new Socket("192.168.1.131", 1978);               //实例化 Socket 对象
        ta.append("完成连接。\n");                                  //文本域中提示信息
        InetAddress netAddress = socket.getInetAddress();         //获得远程服务器的地址
        String netIp = netAddress.getHostAddress();               //获得远程服务器的 IP 地址
        int netPort = socket.getPort();                           //获得远程服务器的端口号
        InetAddress localAddress = socket.getLocalAddress();      //获得客户端的地址
        String localIp = localAddress.getHostAddress();           //获得客户端的 IP 地址
        int localPort = socket.getLocalPort();                    //获得客户端的端口号
        ta.append("远程服务器的 IP 地址：" + netIp + "\n");
        ta.append("远程服务器的端口号：" + netPort + "\n");
        ta.append("客户机本地的 IP 地址：" + localIp + "\n");
        ta.append("客户机本地的端口号：" + localPort + "\n");
    } catch (Exception e) {
        e.printStackTrace();                                      //输出异常信息
    }
}
```

16.3 UDP 程序设计基础

视频讲解：光盘\TM\lx\16\UDP 程序设计基础.exe

用户数据包协议（UDP）是网络信息传输的另一种形式。基于 UDP 的通信和基于 TCP 的通信不同，基于 UDP 的信息传递更快，但不提供可靠的保证。使用 UDP 传递数据时，用户无法知道数据能否正确地到达主机，也不能确定到达目的地的顺序是否和发送的顺序相同。虽然 UDP 是一种不可靠的协议，但有时需要较快地传输信息，能容忍小的错误，可以考虑使用 UDP。

基于 UDP 通信的基本模式如下：

☑ 将数据打包（称为数据包），然后将数据包发往目的地。

☑ 接收别人发来的数据包，然后查看数据包。

发送数据包：

（1）使用 DatagramSocket()方法创建一个数据包套接字。

（2）使用 DatagramPacket(byte[] buf, int offset, int length, InetAddress address, int port)方法创建要发送的数据包。

（3）使用 DatagramSocket 类的 send()方法发送数据包。

接收数据包：

（1）使用 DatagramSocket(int port)方法创建数据包套接字，绑定到指定的端口。

（2）使用 DatagramPacket(byte[] buf, int length)方法创建字节数组来接收数据包。

（3）使用 DatagramPacket 类的 receive()方法接收 UDP 包。

注意 使用 DatagramSocket 类的 receive()方法接收数据时，如果还没有可以接收的数据，在正常情况下 receive()方法将阻塞，一直等到网络上有数据到来才接收该数据并返回。如果网络上没有数据发送过来，receive()方法也没有阻塞，肯定是程序有问题，大多数是使用了一个被其他程序占用的端口号。

16.3.1　DatagramPacket 类

java.net 包的 DatagramPacket 类用来表示数据包。DatagramPacket 类的构造函数如下：

- ☑ DatagramPacket(byte[] buf, int length)
- ☑ DatagramPacket(byte[] buf, int length, InetAddress address, int port)

第一种构造函数创建 DatagramPacket 对象，指定了数据包的内存空间和大小。第二种构造函数不仅指定了数据包的内存空间和大小，而且指定了数据包的目标地址和端口。在发送数据时，必须指定接收方的 Socket 地址和端口号，因此使用第二种构造函数可创建发送数据的 DatagramPacket 对象。

16.3.2　DatagramSocket 类

java.net 包中的 DatagramSocket 类用于表示发送和接收数据包的套接字。该类的构造函数如下：

- ☑ DatagramSocket()
- ☑ DatagramSocket(int port)
- ☑ DatagramSocket(int port, InetAddress addr)

第一种构造函数创建 DatagramSocket 对象，构造数据包套接字并将其绑定到本地主机上任何可用的端口。使用第二种构造函数创建 DatagramSocket 对象，创建数据包套接字并将其绑定到本地主机上的指定端口。第三种构造函数创建 DatagramSocket 对象，创建数据包套接字，将其绑定到指定的本地地址。第三种构造函数适用于有多块网卡和多个 IP 的情况。

注意 在接收程序时，必须指定一个端口号，不要让系统随机产生。可以使用第二种构造函数。就像有个朋友要你给他写信，可他的地址不确定是不行的。在发送程序时，通常使用第一种构造函数，不指定端口号，这样系统就会为我们分配一个端口号。就像寄信一样，不需要到指定的邮局去寄。

16.3.3　UDP 网络程序

根据前面所讲的网络编程的基本知识，以及 UDP 网络编程的特点，下面创建一个广播数据包程序。广播数据包是一种较新的技术，类似于广播电台，广播电台需要在指定的波段和频率上广播信息，收听者也要将收音机调到指定的波段、频率才可以收听广播内容。

【例 16.4】 主机不断地重复播出节目预报，可以保证加入到同一组的主机随时可接收到广播信息。接收者将正在接收的信息放在一个文本域中，并将接收的全部信息放在另一个文本域中。（**实例位置：光盘\TM\sl\16\8**）

（1）广播主机程序不断地向外发出广播信息，代码如下：

```
public class Weather extends Thread {                                   //创建类，该类为多线程执行程序
    String weather = "节目预报：八点有大型晚会，请收听";                    //需要发出的广播信息
    int port = 9898;                                                    //定义端口
    InetAddress iaddress = null;                                        //创建 InetAddress 对象
    MulticastSocket socket = null;                                      //声明多点广播套接字
    Weather() {                                                         //构造方法
        try {
            iaddress = InetAddress.getByName("224.255.10.0");           //实例化 InetAddress，指定地址
            socket = new MulticastSocket(port);                         //实例化多点广播套接字
            socket.setTimeToLive(1);                                    //指定发送范围是本地网络
            socket.joinGroup(iaddress);                                 //加入广播组
        } catch (Exception e) {
            e.printStackTrace();                                        //输出异常信息
        }
    }
    public void run() {                                                 //run()方法
        while (true) {
            DatagramPacket packet = null;                               //声明 DatagramPacket 对象
            byte data[] = weather.getBytes();                           //声明字节数组
            packet = new DatagramPacket(data, data.length, iaddress, port); //将数据打包
            System.out.println(new String(data));                       //将广播信息输出
            try {
                socket.send(packet);                                    //发送数据
                sleep(3000);                                            //线程休眠
            } catch (Exception e) {
                e.printStackTrace();                                    //输出异常信息
            }
        }
    }
    public static void main(String[] args) {                            //主方法
        Weather w = new Weather();                                      //创建本类对象
        w.start();                                                      //启动线程
    }
}
```

运行结果如图 16.15 所示。

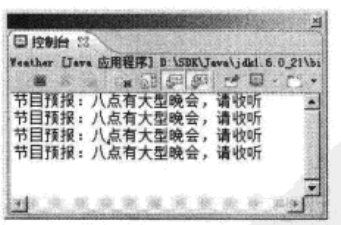


图 16.15 广播主机程序的运行结果

（2）接收广播程序。当单击“开始接收”按钮时，开始接收主机播出的信息；当单击“停止接收”按钮时，系统会停止接收广播主机播出的信息。代码如下：

```
public class Receive extends JFrame implements Runnable, ActionListener {
    int port;                                                          //定义 int 型变量
    InetAddress group = null;                                          //声明 InetAddress 对象
    MulticastSocket socket = null;                                     //创建多点广播套接字对象
    JButton ince = new JButton("开始接收");                            //创建按钮对象
    JButton stop = new JButton("停止接收");
    JTextArea inceAr = new JTextArea(10, 10);                          //显示接收广播的文本域
    JTextArea inced = new JTextArea(10, 10);
    Thread thread;                                                     //创建 Thread 对象
    boolean b = false;                                                 //创建 boolean 型变量
    public Receive() {                                                 //构造方法
        super("广播数据报");                                           //调用父类方法
        setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
        thread = new Thread(this);                                     //创建线程对象
        ince.addActionListener(this);                                  //绑定按钮 ince 的单击事件
        stop.addActionListener(this);                                  //绑定按钮 stop 的单击事件
        inceAr.setForeground(Color.blue);                              //指定文本域中文字颜色
        JPanel north = new JPanel();                                   //创建 JPanel 对象
        north.add(ince);                                               //将按钮添加到面板 north 上
        north.add(stop);
        add(north, BorderLayout.NORTH);                                //将 north 放置在窗体的上部
        JPanel center = new JPanel();                                  //创建面板对象 center
        center.setLayout(new GridLayout(1, 2));                        //设置面板布局
        center.add(inceAr);                                            //将文本域添加到面板上
        center.add(inced);
        add(center, BorderLayout.CENTER);                              //设置面板布局
        validate();                                                    //刷新
        port = 9898;                                                   //设置端口号
        try {
            group = InetAddress.getByName("224.255.10.0");             //指定接收地址
            socket = new MulticastSocket(port);                        //绑定多点广播套接字
            socket.joinGroup(group);                                   //加入广播组
        } catch (Exception e) {
            e.printStackTrace();                                       //输出异常信息
        }
        setBounds(100, 50, 360, 380);                                  //设置布局
        setVisible(true);                                              //将窗体设置为显示状态
    }
    public void run() {                                                //run()方法
        while (true) {
            byte data[] = new byte[1024];                              //创建 byte 数组
            DatagramPacket packet = null;                              //创建 DatagramPacket 对象
            packet = new DatagramPacket(data, data.length, group, port);  //待接收的数据包
            try {
                socket.receive(packet);                                //接收数据包
                String message = new String(packet.getData(), 0, packet
```

```
                    .getLength());                                  //获取数据包中内容
                inceAr.setText("正在接收的内容：\n" + message);      //将接收内容显示在文本域中
                inced.append(message + "\n");                       //每条信息为一行
            } catch (Exception e) {
                e.printStackTrace();                                //输出异常信息
            }
            if (b == true) {                                        //当变量等于 true 时，退出循环
                break;
            }
        }
    }
    public void actionPerformed(ActionEvent e) {                    //单击事件
        if (e.getSource() == ince) {                                //单击按钮 ince 触发的事件
            ince.setBackground(Color.red);                          //设置按钮颜色
            stop.setBackground(Color.yellow);
            if (!(thread.isAlive())) {                              //如线程不处于新建状态
                thread = new Thread(this);                          //实例化 Thread 对象
            }
            thread.start();                                         //启动线程
            b = false;                                              //设置变量值
        }
        if (e.getSource() == stop) {                                //单击按钮 stop 触发的事件
            ince.setBackground(Color.yellow);                       //设置按钮颜色
            stop.setBackground(Color.red);
            b = true;                                               //设置变量值
        }
    }
    public static void main(String[] args) {                        //主方法
        Receive rec = new Receive();                                //创建本类对象
        rec.setSize(460, 200);                                      //设置窗体大小
    }
}
```

运行结果如图 16.16 所示。

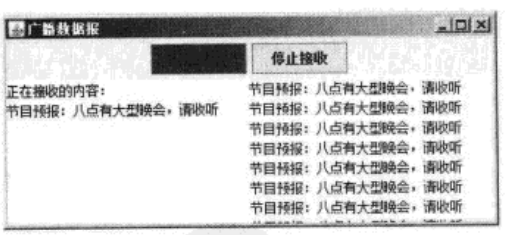


图 16.16　接收广播的运行结果

说明　要广播或接收广播的主机地址必须加入到一个组内，地址在 224.0.0.0～224.255.255.255 之间，这类地址并不代表某个特定主机的位置。加入到同一个组的主机可以在某个端口上广播信息，也可以在某个端口上接收信息。

16.4　经 典 范 例

网络通信在以后的开发中非常重要，现在做几个关于网络通信的例子。

16.4.1　经典范例 1：聊天室服务器端

视频讲解：光盘\TM\lx\16\聊天室服务器端.exe

本范例实现了聊天室服务器端的功能，运行程序，服务器端等待客户端的连接，并显示客户端的连接信息。运行结果如图 16.17 所示。（**实例位置：光盘\TM\sl\16\9**）

（1）在 Eclipse 中新建项目 09，在项目中创建 com.mingrisoft 包。

（2）在 com.mingrisoft 包中创建一个继承自 JFrame 类的服务器窗体类 ChatServerFrame，然后在窗体上添加一个滚动面板，并在滚动面板上添加一个文本域控件，完成窗体界面的设计。

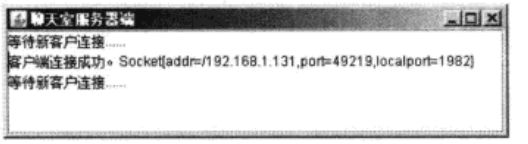


图 16.17　聊天室服务器端

（3）在服务器窗体类 ChatServerFrame 的成员声明区定义一个 Hashtable 对象，用于存储登录用户的用户名和套接字对象。代码如下：

```
//用于存储连接到服务器的用户名和客户端套接字对象
private Hashtable<String, Socket> map = new Hashtable<String, Socket>();
```

（4）在服务器窗体类 ChatServerFrame 中，定义 createSocket()方法，用于创建服务器套接字对象、获得连接到服务器的客户端套接字对象以及启动线程对象对客户端发送的信息进行处理。代码如下：

```
public void createSocket() {
    try {
        server = new ServerSocket(1982);                          //创建服务器套接字对象
        while (true) {
            ta_info.append("等待新客户连接.....\n");
            socket = server.accept();                             //获得套接字对象
            ta_info.append("客户端连接成功。" + socket + "\n");
            new ServerThread(socket).start();                     //创建并启动线程对象
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

（5）在服务器窗体类 ChatServerFrame 中定义内部线程类 ServerThread，用于对客户端的连接信息以及发送的信息进行处理和转发。代码如下：

```
class ServerThread extends Thread {
    //省略了部分代码
    public void run() {
        try {
            //获得输入流对象
            ObjectInputStream ins = new ObjectInputStream(socket.getInputStream());
            while (true) {
                //省略了部分代码
                if (v != null && v.size() > 0) {
                    for (int i = 0; i < v.size(); i++) {
                        String info = (String) v.get(i);                    //读取信息
                        String key = "";
                        if (info.startsWith("用户：")) {                     //添加登录用户到客户端列表
                            //省略了部分代码
                        } else if (info.startsWith("退出：")) {
                            //省略了部分代码
                        } else {                                            //转发接收的消息
                            //获得接收方的 key 值，即接收方的用户名
                            key = info.substring(info.indexOf("：发送给：") + 5,
                                    info.indexOf("：的信息是："));
                            //获得发送方的 key 值，即发送方的用户名
                            String sendUser = info.substring(0, info
                                    .indexOf("：发送给："));
                            Set<String> set = map.keySet();             //获得集合中所有键的 Set 视图
                            Iterator<String> keyIt = set.iterator();    //获得所有键的迭代器
                            while (keyIt.hasNext()) {
                                String receiveKey = keyIt.next();       //获得表示接收信息的键
                                //与接受用户相同，但不是发送用户
                                if (key.equals(receiveKey) && !sendUser.equals(receiveKey)) {
                                    Socket s = map.get(receiveKey); //获得与该键对应的套接字对象
                                                                    //创建输出流对象
                                    PrintWriter out = new PrintWriter(s.getOutputStream(), true);
                                    out.println("MSG:" + info);         //发送信息
                                    out.flush();                        //刷新输出缓冲区
                                }
                            }
                        }
                    }
                }
            }
        } catch (IOException e) {
            ta_info.append(socket + "已经退出。\n");
        }
    }
}
```

16.4.2　经典范例 2：聊天室客户端

视频讲解：光盘\TM\lx\16\聊天室客户端.exe

本范例实现了聊天室客户端，运行程序，用户登录服务器后，可以从用户列表中选择单个用户进行聊天，也可以选择多个用户进行聊天。运行结果如图 16.18 所示。（**实例位置：光盘\TM\sl\16\10**）

图 16.18　聊天室客户端

（1）在 Eclipse 中新建项目 10，在项目中创建 com.mingrisoft 包。

（2）在 com.mingrisoft 包中创建一个继承自 JFrame 类的客户端窗体类 ChatClientFrame，用于进行用户登录，发送聊天信息和显示聊天信息，在该类中完成窗体界面的设计。

（3）在客户端窗体类 ChatClientFrame 中定义 createClientSocket() 方法，用于创建套接字对象、创建输出流对象以及启动线程对象对服务器转发的信息进行处理。代码如下：

```
public void createClientSocket() {
    try {
        Socket socket = new Socket("192.163.1.131", 1982);          //创建套接字对象
        out = new ObjectOutputStream(socket.getOutputStream());     //创建输出流对象
        new ClientThread(socket).start();                           //创建并启动线程对象
    } catch (UnknownHostException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

（4）在客户端窗体类 ChatClientFrame 中，定义内部线程类 ClientThread，用于对服务器端转发的信息进行处理，并显示在相应的控件中。

（5）在客户端窗体类 ChatClientFrame 中，为“登录”按钮添加实现用户登录功能的代码。“登录”按钮的事件代码如下：

```
if (loginFlag) {                                                    //已登录标记为 true
    JOptionPane.showMessageDialog(null, "在同一窗口只能登录一次。");
    return;
}
String userName = tf_newUser.getText().trim();                      //获得登录用户名
Vector v = new Vector();                                            //定义向量，用于存储登录用户
v.add("用户：" + userName);                                          //添加登录用户
try {
    out.writeObject(v);                                             //将用户向量发送到服务器
    out.flush();                                                    //刷新输出缓冲区
} catch (IOException ex) {
```

```
    ex.printStackTrace();
}
tf_newUser.setEnabled(false);                                    //禁用用户文本框
loginFlag = true;                                                //将已登录标记设置为 true
```

（6）在客户端窗体类 ChatClientFrame 中定义发送聊天信息的 send()方法。代码如下：

```
private void send() {
    if (!loginFlag) {
        JOptionPane.showMessageDialog(null, "请先登录。");
        return;                                                  //如果用户没登录则返回
    }
    String sendUserName = tf_newUser.getText().trim();           //获得登录用户名
    String info = tf_send.getText();                             //获得输入的发送信息
    if (info.equals("")) {
        return;                                                  //如果没输入信息则返回，即不发送
    }
    Vector<String> v = new Vector<String>();                     //创建向量对象，用于存储发送的消息
    Object[] receiveUserNames = user_list.getSelectedValues();   //获得选择的用户数组
    if (receiveUserNames.length <= 0) {
        return;                                                  //如果没选择用户则返回
    }
    for (int i = 0; i < receiveUserNames.length; i++) {
        String msg = sendUserName + "：发送给：" + (String) receiveUserNames[i]
                + "：的信息是：  " + info;                        //定义发送的信息
        v.add(msg);                                              //将信息添加到向量
    }
    try {
        out.writeObject(v);                                      //将向量写入输出流，完成信息的发送
        out.flush();                                             //刷新输出缓冲区
    } catch (IOException e) {
        e.printStackTrace();
    }
    DateFormat df = DateFormat.getDateInstance();                //获得 DateFormat 实例
    String dateString = df.format(new Date());                   //格式化为日期
    df = DateFormat.getTimeInstance(DateFormat.MEDIUM);          //获得 DateFormat 实例
    String timeString = df.format(new Date());                   //格式化为时间
    String sendUser = tf_newUser.getText().trim();               //获得发送信息的用户
    String receiveInfo = tf_send.getText().trim();               //显示发送的信息
     //在文本域中显示信息
    ta_info.append("  "+sendUser + "      " +dateString+"  "+timeString+"\n   "+receiveInfo+"\n");
    tf_send.setText(null);                                       //清空文本框
    ta_info.setSelectionStart(ta_info.getText().length()-1);     //设置选择的起始位置
    ta_info.setSelectionEnd(ta_info.getText().length());         //设置选择的结束位置
    tf_send.requestFocus();                                      //使发送信息文本框获得焦点
}
```

16.5　本 章 小 结

本章向读者介绍了 Java 网络编程知识，对于网络协议等内容，程序设计人员都应该有所了解，有兴趣的读者还可以查阅其他资料来获取更详细的信息。TCP、UDP 网络编程的区别、java.net 包中提供的网络应用程序的常用类，以及这些类中的常用方法是本章的重点。通过本章的学习，读者应该自己尝试着编写一些网络程序来巩固所学知识。

16.6　实 战 练 习

1．在使用套接字进行网络编程时，有时需要通过 Socket 传递对象，做一个使用 Socket 传递对象的例子。（**答案位置：光盘\TM\sl\16\11**）

2．在使用套接字进行网络编程时，有时需要通过 Socket 传输图片，做一个通过套接字传输图片的例子，要求在服务器端选择图片，单击“发送”按钮，就会将图片发送到客户端。（**答案位置：光盘\TM\sl\16\12**）

3．在使用套接字进行网络编程时，使用 Socket 传递汉字有时会出现乱码，做一个例子防止出现汉字乱码的问题。（**答案位置：光盘\TM\sl\16\13**）

第17章

JDBC 操作数据库

（视频讲解：1 小时 25 分钟）

JDBC 是连接数据库和 Java 程序的桥梁，通过 JDBC API 可以方便地实现对各种主流数据库的操作。学习 Java 语言，必须学习 JDBC 技术，因为 JDBC 技术是在 Java 语言中被广泛使用的一种操作数据库的技术。每个应用程序的开发都是使用数据库保存数据。使用 JDBC 技术访问数据库可达到查找满足条件的记录，或者向数据库添加、修改、删除数据。本章向读者介绍 Java 语言的数据库操作部分。

通过阅读本章，您可以：

- 了解 JDBC 概念
- 掌握 JDBC 的常用类和接口
- 掌握如何通过 JDBC 操作数据库

17.1 JDBC 概述

视频讲解：光盘\TM\lx\17\JDBC 概述.exe

程序设计中离不开数据库的支持。每个程序的开发都是使用数据库保存数据，使用 JDBC 技术访问数据库可达到查询满足条件的记录，或者向数据库添加、修改、删除数据。

17.1.1 数据库概述

数据库是一种存储结构，它允许使用各种格式输入、处理和检索数据，不必在每次需要数据的时候重新输入它们。例如在需要某人的电话号码时，需要查看电话本，按照姓名来查阅。这个电话本就是一个数据库。

1. 数据库特点

数据库具有以下主要特点：

- ☑ 实现数据共享。数据共享包含所有用户可同时存取数据库中的数据，也包括用户可以用各种方式通过接口使用数据库，并提供数据共享。
- ☑ 减少数据的冗余度。同文件系统相比，由于数据库实现了数据共享，从而避免了用户各自建立应用文件。减少了大量重复数据，减少了数据冗余，维护了数据的一致性。
- ☑ 数据的独立性。数据的独立性包括数据库中数据库的逻辑结构和应用程序相互独立，也包括数据物理结构的变化不影响数据的逻辑结构。
- ☑ 数据实现集中控制。文件管理方式中，数据处于一种分散的状态，不同的用户或同一用户在不同处理中其文件之间毫无关系。利用数据库可对数据进行集中控制和管理，并通过数据模型表示各种数据的组织以及数据间的联系。
- ☑ 数据一致性和可维护性，以确保数据的安全性和可靠性。主要包括：安全性控制，以防止数据丢失、错误更新和越权使用；完整性控制，保证数据的正确性、有效性和相容性；并发控制，使在同一时间周期内，允许对数据实现多路存取，又能防止用户之间的不正常交互作用；故障的发现和恢复。

2. 数据库的种类及功能

数据库系统一般基于某种数据模型，可以分为层次型、网状型、面向对象型、关系型等。

- ☑ 层次型数据库：类似于树结构，是一组通过链接而相互联系在一起的记录。层次模型的特点是记录之间的联系是通过指针实现的。由于层次模型层次顺序严格而且复杂。这样导致对数据的各项操作都很困难。层次型数据库如图 17.1 所示。
- ☑ 网状型数据库：网络模型是使用网络结构表示实体类型、实体间联系的数据模型。网络模型容易实现多对多的联系。但需要编写应用程序时，程序员必须熟悉数据库的逻辑结构，如图 17.2 所示。

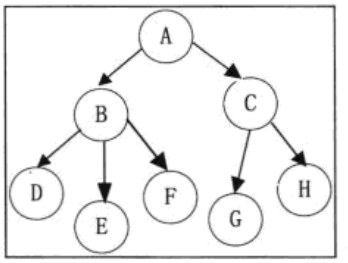


图 17.1　层次型数据库

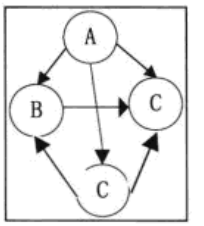


图 17.2　网状型数据库

- ☑ 面向对象型数据库：建立在面前对象模型基础上。
- ☑ 关系型数据库：是目前最流行的数据库，是基于关系模型建立的数据库，关系模型由一系列表格组成，后面章节将详细讲解。

17.1.2　JDBC 技术

JDBC 制定了统一的访问各类关系数据库的标准接口，为各个数据库厂商提供了标准接口的实现。在 JDBC 技术问世之前，各家数据库厂商执行各自的一套 API，使得开发人员访问数据库非常困难，特别是更换数据库时，需要修改大量的代码。JDBC 的发布获得了巨大的成功，很快就成为了 Java 访问数据库的标准。需要注意的是，JDBC 并不能直接访问数据库，需要依赖与数据库厂商提供的 JDBC 驱动程序。JDBC 技术具有以下优点：

- ☑ JDBC 与 ODBC 十分相似，便于软件开发人员理解。
- ☑ JDBC 使软件开发人员从复杂的驱动程序编写工作中解脱出来，可以完全专注于业务逻辑的开发。
- ☑ JDBC 支持多种关系型数据库，大大增加了软件的可移植性。
- ☑ JDBC API 是面向对象的，软件开发人员可以将常用的方法进行二次封装，从而提高代码的重用性。

17.2　JDBC 的常用类和接口

视频讲解：光盘\TM\lx\17\JDBC 的常用类和接口.exe

JDBC 是一种可用于执行 SQL 语句的 Java API（Application Programming Interface，应用程序设计接口），是连接数据库和 Java 应用程序的一个纽带。

在 Java 语言中提供了丰富的类和接口用于数据库编程，利用这些类和接口可以方便地进行数据访问和处理。本节向读者介绍一些常用的 JDBC 接口和类，这些类或接口都在 java.sql 包中。

17.2.1　DriverManager 类

DriverManager 类用来管理数据库中的所有驱动程序，是 JDBC 的管理层，作用于用户和驱动程序

之间，跟踪可用的驱动程序，并在数据库的驱动程序之间建立连接。此外，DriverManager 类也处理诸如驱动程序登录时间限制及登录和跟踪信息的显示等事务。DriverManager 类中的方法都是静态方法，所以在程序中无须对它进行实例化，直接通过类名就可以调用。DriverManager 类的常用方法如表 17.1 所示。

表 17.1　DriverManager 类的常用方法

方　法	说　明
getConnection(String url, String user, String password)	指定 3 个入口参数，依次是连接数据库的 URL、用户名、密码，来获取与数据库的连接
setLoginTimeout()	获取驱动程序试图登录到某一数据库时可以等待的最长时间，以秒为单位
println(String message)	将一条消息打印到当前 JDBC 日志流中

17.2.2　Connection 接口

Connection 接口代表与特定的数据库的连接。要对数据表中的数据进行操作，首先要获取数据库连接。Connection 实例就像在应用程序与数据库之间开通了一条渠道，如图 17.3 所示。

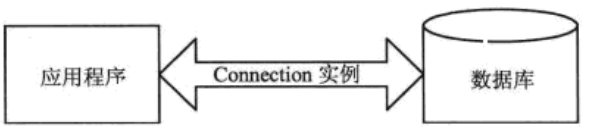


图 17.3　Connection 实例

可通过 DriverManager 类的 getConnection()方法获取 Connection 实例。Connection 接口的常用方法如表 17.2 所示。

表 17.2　Connection 接口的常用方法

方　法	说　明
createStatement()	创建 Statement 对象
createStatement(int resultSetType,int resultSetConcurrency)	创建一个 Statement 对象，该对象将生成具有给定类型、并发性和可保存性的 ResultSet 对象
prepareStatement()	创建预处理对象 PreparedStatement
isReadOnly()	查看当前 Connection 对象的读取模式是否是只读形式
setReadOnly()	设置当前 Connection 对象的读写模式，默认是非只读模式
commit()	使所有上一次提交、回滚后进行的更改成为持久更改，并释放此 Connection 对象当前持有的所有数据库锁
roolback()	取消在当前事务中进行的所有更改，并释放此 Connection 对象当前持有的所有数据库锁
close()	立即释放此 Connection 对象的数据库和 JDBC 资源，而不是等待它们被自动释放

17.2.3　Statement 接口

Statement 接口用于创建向数据库中传递 SQL 语句的对象，该接口提供了一些方法可以实现对数据库的常用操作。Statement 接口的常用方法如表 17.3 所示。

表 17.3　Statement 接口的常用方法

方　法	说　明
execute(String sql)	执行静态的 SELECT 语句，该语句可能返回多个结果集
executeQuery(String sql)	执行给定的 SQL 语句，该语句返回单个 ResultSet 对象
clearBatch()	清空此 Statement 对象的当前 SQL 命令列表
executeBatch()	将一批命令提交给数据库来执行，如果全部命令执行成功，则返回更新计数组成的数组。数组元素的排序与 SQL 语句的添加顺序对应
executeUpdate()	执行给定 SQL 语句，该语句可以为 INSERT、UPDATE 或 DELETE 语句
addBatch(String sql)	将给定的 SQL 命令添加到此 Statement 对象的当前命令列表中。如果驱动程序不支持批量处理将抛出异常
close()	释放 Statement 实例占用的数据库和 JDBC 资源

17.2.4　PreparedStatement 接口

PreparedStatement 接口继承 Statement，用于执行动态的 SQL 语句，通过 PreparedStatement 实例执行的 SQL 语句，将被预编译并保存到 PreparedStatement 实例中，从而可以反复地执行该 SQL 语句。PreparedStatement 接口的常用方法如表 17.4 所示。

表 17.4　PreparedStatement 接口的常用方法

方　法	说　明
execute()	在此 PreparedStatement 对象中执行 SQL 语句，该语句可以是任何类型的 SQL 语句
executeQuery()	在此 PreparedStatement 对象中执行 SQL 查询语句，返回结果为查询结果集 ResultSet 对象
executeUpdate()	在此 PreparedStatement 对象中执行 SQL 语句，该 SQL 语句必须是一个 INSERT、UPDATE、DELETE 语句，或者是没有返回值的 DDL 语句
setByte(int pIndex,byte bt)	将参数 pIndex 位置上设置为给定的 byte 型参数 bt
setDouble(int pIndex,double dou)	将参数 pIndex 位置上设置为给定的 double 型参数值 dou
setInt(int pIndex,int x)	将参数 pIndex 位置上设置为给定的 int 型参数值 x
setObject(int pIndex,Object o)	将参数 pIndex 位置上设置为给定的 Object 型参数值
setString(int pIndex,String str)	将参数 pIndex 位置上设置为给定的 String 型参数值

17.2.5　ResultSet 接口

ResultSet 接口类似于一个临时表，用来暂时存放数据库查询操作所获得的结果集。ResultSet 接口

的常用方法如表 17.5 所示。

表 17.5　ResultSet 接口的常用方法

方　法	说　明
getInt()	以 int 形式获取 ResultSet 对象的当前行的指定列值。如列值是 null，则返回值是 0
getFloat()	以 float 形式获取 ResultSet 对象的当前行的指定列值。如列值是 null，则返回值是 0
getDate()	以 Date 形式获取 ResultSet 对象的当前行的指定列值。如列值是 null，则返回值是 null
getBoolean()	以 boolean 形式获取 ResultSet 对象的当前行的指定列值。如列值是 null，则返回 null
getString()	以 String 形式获取 ResultSet 对象的当前行的指定列值。如列值是 null，则返回 null
getObject()	以 Object 形式获取 ResultSet 对象的当前行的指定列值。如列值是 null，则返回 null
next()	将指针向下移一行
updateInt()	用 int 值更新指定列
updateFloat()	用 float 值更新指定列
updateLong()	用指定的 long 值更新指定列
updateString()	用指定的 String 值更新指定列
updateObject()	用 Object 值更新指定列
updateNull()	将指定的列值修改为 null
updateDate()	用指定的 Date 值更新指定列
updateDouble()	用指定的 double 值更新指定列

17.3　通过 JDBC 操作数据库

视频讲解：光盘\TM\lx\17\通过 JDBC 操作数据库.exe

如果需要访问数据库，首先要加载数据库驱动，数据库驱动只需在第一次访问数据库时加载一次。然后在每次访问数据库时创建一个 Connection 实例，获取数据库连接，这样就可以执行操作数据库的 SQL 语句。最后在完成数据库操作时，释放与数据库的连接。

17.3.1　加载数据库驱动

将下载的数据库驱动文件添加到项目中后，首先需要加载数据库驱动程序，才能进行数据库操作。Java 加载数据库驱动的方法是调用 Class 类的静态方法 forName()。语法格式如下：

```
Class.forName(String driverManager)
```

☑　driverManager：所需类的完全限定名。

☑　返回值：返回与带有给定字符串名的类或接口相关联的 Class 对象。

forName()方法参数指定要加载的数据库驱动，加载成功，会将加载的驱动类注册给 DriverManager。如果加载失败，则会抛出 ClassNotFoundException 异常。

【例 17.1】　加载 MySQL 数据库的驱动。

```
try {
    Class.forName("com.mysql.jdbc.Driver");                    //加载数据库驱动
} catch (ClassNotFoundException e) {
    e.printStackTrace();
}
```

技巧　其实加载的驱动类是可以找到的，它们是以一定的规律保存在相应数据库驱动类中，以查找 MySQL 的驱动类为例。查找方法如图 17.4 所示。

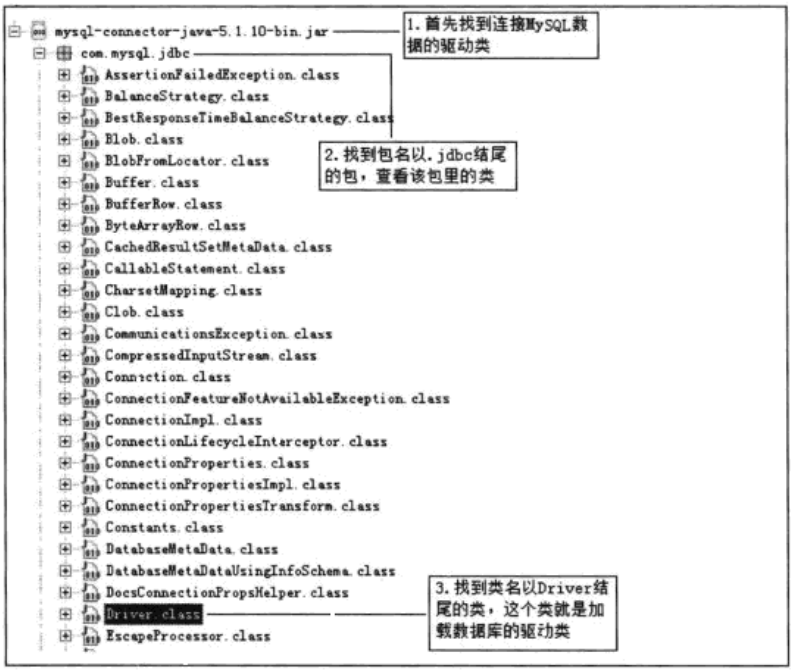


图 17.4　查找驱动连接类

17.3.2　建立连接

加载完数据库驱动即可建立数据库的连接，要连接数据库可以使用 DriverManager 类的静态方法 getConnection()来实现，并分别将数据库的 URL、数据库用户名和密码作为该方法的参数，即可建立到指定数据库的连接。

【例 17.2】　建立与 MySQL 数据库的连接。

```
Class.forName("com.mysql.jdbc.Driver");                    //加载数据库驱动
String url = "jdbc:mysql://localhost:3306/db_database17";  //声明数据库 db_database17 的 URL
String user = "root";                                      //数据库用户名
String password = "111";                                   //数据库密码
//建立数据库连接，获得连接对象 conn
Connection conn = DriverManager.getConnection(url, user, password);
```

【例 17.3】 建立与 SQL Server 2005 数据库的连接。

```
Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");          //加载数据库驱动
//声明数据库 db_database17 的 URL
String url = " jdbc:sqlserver://127.0.0.1:1433;DatabaseName=db_database17";
String user = "sa";                                                     //数据库用户名
String password = "";                                                   //数据库密码
Connection conn = DriverManager.getConnection(url, user, password);    //建立数据库连接，获得连接对象 conn
```

说明 对比例 17.2 和例 17.3，其中 url 的前半部分分别为“jdbc:mysql://”、“jdbc:sqlserver://”，是连接对应数据库的固定写法。接着 localhost 是本地计算机，可以换成 IP 地址 127.0.0.1，如果连接的是网络中其他计算机中的数据库，只需要将 localhost 改为那台计算机的 IP 地址即可。3306 和 1433 分别为 MySQL 和 SQL Server 2005 数据库的默认端口号。最后的 db_database17 是所要连接的数据库名。user 和 password 为对应数据库的用户名和密码。

技巧 由于在一个程序中经常需要对数据库进行操作，如果每次操作数据库都要建立数据库的连接，这样不但会出现代码重复，而且也极大地影响了编程效率，为此可以为数据库连接单独创建一个类，在该类中定义一个进行数据库连接的静态方法，当需要使用数据库连接时，可以通过类名调用该方法获得数据库的连接。

17.3.3 向数据库添加数据

在 Java 程序中，一旦建立了数据库连接，就可以使用 Connection 接口的 createStatement()方法来获得 Statement 对象，也可调用 prepareStatement()方法获得 PreparedStatement 对象，通过 executeUpdate()方法来执行 SQL 语句，就可以向数据库添加数据了。

【例 17.4】 使用 Statement 接口中的 executeUpdate()方法向 MySQL 数据库中插入一条数据。（**实例位置：光盘\TM\sl\17\1**）

```
Class.forName("com.mysql.jdbc.Driver");                                 //加载数据库驱动
String url = "jdbc:mysql://localhost:3306/db_database17";               //声明数据库 db_database17 的 URL
String user = "root";                                                   //数据库用户名
String password = "111";                                                //数据库密码
//建立数据库连接，获得连接对象 conn
Connection conn = DriverManager.getConnection(url, user, password);
//生成一条 SQL 语句
String sql = "insert into tb_users (username,password,sex,age) values('张三','111','男','22')";
Statement stmt = conn.createStatement();                                //创建一个 Statment 对象
stmt.executeUpdate(sql);                                                //执行 SQL 语句
conn.close();                                                           //关闭数据库连接对象
```

运行结果如图 17.5 所示。

【例 17.5】　使用 PreparedStatement 接口中的 executeUpdate()方法向 MySQL 数据库中插入一条数据。（**实例位置：光盘\TM\sl\17\2**）

```
Class.forName("com.mysql.jdbc.Driver");                                  //加载数据库驱动
String url = "jdbc:mysql://localhost:3306/db_database17";                //声明数据库 db_database17 的 URL
String user = "root";                                                    //数据库用户名
String password = "111";                                                 //数据库密码
//建立数据库连接，获得连接对象 conn
Connection conn = DriverManager.getConnection(url, user, password);
//生成一条 SQL 语句
String sql = "insert into tb_users (username,password,sex,age) values(?,?,?,?)";
PreparedStatement ps = conn.prepareStatement(sql);                       //创建一个 Statment 对象
ps.setString(1, "李四");                                                 //为 SQL 语句中第一个问号赋值
ps.setString(2, "aaa");                                                  //为 SQL 语句中第二个问号赋值
ps.setString(3, "男");                                                   //为 SQL 语句中第三个问号赋值
ps.setInt(4, 23);                                                        //为 SQL 语句中第四个问号赋值
ps.executeUpdate();                                                      //执行 SQL 语句
conn.close();                                                            //关闭数据库连接对象
```

运行结果如图 17.6 所示。

id	username	password	sex	age
1	张三	111	男	22

图 17.5　在数据库中查看插入的数据

id	username	password	sex	age
1	张三	111	男	22
2	李四	aaa	男	23

图 17.6　在数据库中查看插入的数据

17.3.4　获得查询结果集

Statement 接口的 executeUpdate()或 executeQuery()方法可以执行 SQL 语句，executeUpdate()方法用于执行数据的插入、修改或删除操作，返回影响数据库记录的条数。executeQuery()方法用于执行 SELECT 查询语句，将返回一个 ResultSet 型的结果集。通过遍历查询结果集的内容，才可获取 SQL 语句执行的查询结果。

ResultSet 对象具有指向当前数据行的光标。最初，光标被置于第一行之前，可以通过该对象的 next()方法将光标移动到下一行；如果 ResultSet 对象没有下一行，next()方法返回 false，所以可以在 while 循环使用 next()方法迭代结果集。

【例 17.6】　使用 Statement 接口中提供的 executeQuery()方法查询数据库中 tb_users 表中所有的数据，把查询到的结果赋值给 ResultSet 对象，通过操作 ResultSet 对象把数据打印到控制台上。（**实例位置：光盘\TM\sl\17\3**）

```
Class.forName("com.mysql.jdbc.Driver");                                  //加载数据库驱动
String url = "jdbc:mysql://localhost:3306/db_database17";                //声明数据库 db_database17 的 URL
String user = "root";                                                    //数据库用户名
String password = "111";                                                 //数据库密码
```

```
//建立数据库连接，获得连接对象 conn
Connection conn = DriverManager.getConnection(url, user, password);
Statement stmt=conn.createStatement();                    //创建一个 Statment 对象
String sql="select * from tb_users";                      //生成一条 SQL 语句
ResultSet rs=stmt.executeQuery(sql);                      //执行查询，把查询结果赋值给结果集对象
int id,age;                                               //声明两个变量分别表示 id 和年龄
String username,pwd,sex;                                  //声明 3 个变量分别表示用户名、密码和性别
System.out.println("id\t 用户名\t 密码\t 性别\t 年龄");   //其中的\t 相当于 8 个空格
while(rs.next(){                                          //遍历结果集
    id=rs.getInt("id");                                   //获得 id 值
    username=rs.getString(2);                             //获得用户名
    pwd=rs.getString("password");                         //获得密码
    sex=rs.getString(4);                                  //获得性别
    age=rs.getInt("age");                                 //获得年龄
    System.out.println(id+"\t"+username+"\t"+pwd+"\t"+sex+"\t"+age);
}
```

运行结果如图 17.7 所示。

说明 ResultSet 对象的 getXXX()方法可获取查询结果集中数据，由于 ResultSet 中保存的数据是表的形式，因此可通过使用 getXXX()方法指定列的序号与列的名称来获取数据。例如上述实例中的代码“rs.getString("username")”，username 表示表的列名。如果此时 username 列为查询结果集中的第 2 列，也可通过代码“rs.getString(2)”得到查询结果。图 17.8 表示了结果集 rs 的结果。

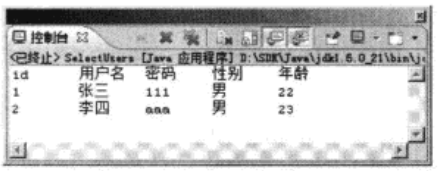


图 17.7　tb_users 表中所有的数据

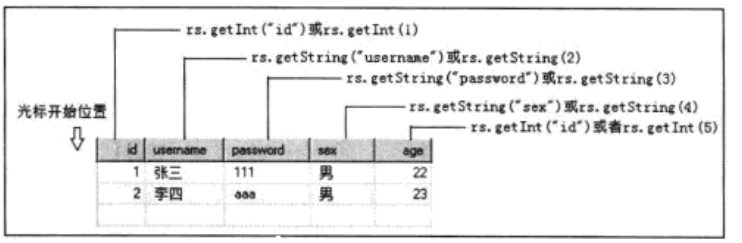


图 17.8　结果集 rs 的结果

17.3.5　更改数据库中的数据

修改数据是数据库操作中必不可少的一部分，使用 Statement 接口中的 executeUpdate()方法可以修改数据库表中的数据，也可以使用 PreparedStatement 接口中的 executeUpdate()方法对数据库中的表进行修改操作。

【例 17.7】　使用 Statement 接口中的 executeUpdate()方法修改数据库 tb_users 表中的数据，把 id 为 1 的用户年龄改为 20。（**实例位置：光盘\TM\sl\17\4**）

```
Class.forName("com.mysql.jdbc.Driver");                   //加载数据库驱动
String url = "jdbc:mysql://localhost:3306/db_database17";  //声明数据库 db_database17 的 URL
String user = "root";                                     //数据库用户名
```

```
String password = "111";                                    //数据库密码
//建立数据库连接，获得连接对象 conn
Connection conn = DriverManager.getConnection(url, user, password);
String sql="update tb_users set age=20 where id=1";         //生成一条 SQL 语句
Statement stmt = conn.createStatement();                    //创建一个 Statment 对象
stmt.executeUpdate(sql);                                    //执行 SQL 语句
System.out.println("修改数据成功！");
conn.close();                                               //关闭数据库连接对象
```

运行结果如图 17.9 所示。

修改数据是根据一定的条件进行修改，这个条件可以是固定的，也可以是一个范围。下面做一个根据范围修改的例子，同样使用 executeUpdate()方法，它的返回值是一个 int 类型的数据，表示受影响的行数，获得这个值就可以知道这次一共修改了几条数据。

【例 17.8】 使用 PreparedStatement 接口中的 executeUpdate()方法修改数据库 tb_users 表中的数据，把所有性别为男的用户密码改为 admin，修改成功后提示修改了几条数据。（**实例位置：光盘\TM\sl\17\5**）

```
Class.forName("com.mysql.jdbc.Driver");                     //加载数据库驱动
String url = "jdbc:mysql://localhost:3306/db_database17";   //声明数据库 db_database17 的 URL
String user = "root";                                       //数据库用户名
String password = "111";                                    //数据库密码
//建立数据库连接，获得连接对象 conn
Connection conn = DriverManager.getConnection(url, user, password);
String sql="update tb_users set password=? where sex=?";    //生成一条 SQL 语句
PreparedStatement ps = conn.prepareStatement(sql);          //创建 PreparedStatement 对象
ps.setString(1, "admin");                                   //为第一个问号赋值
ps.setString(2, "男");                                      //为第二个问号赋值
int count = ps.executeUpdate();                             //执行 SQL 语句
System.out.println("成功修改"+count+"条数据！");
conn.close();                                               //关闭数据库连接对象
```

运行结果如图 17.10 所示。

图 17.9　提示修改数据成功

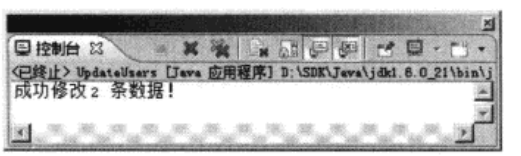


图 17.10　提示修改几条数据

17.3.6　删除数据库中的数据

删除数据表中数据同样是一个很常用的技术，使用 executeUpdate()方法执行用来做删除的 SQL 语句可以删除数据库表中的数据。

【例 17.9】 使用 Statement 接口中的 executeUpdate()方法，删除数据库 tb_users 表中 id 为 1 的用户信息。（**实例位置：光盘\TM\sl\17\6**）

```
Class.forName("com.mysql.jdbc.Driver");                          //加载数据库驱动
String url = "jdbc:mysql://localhost:3306/db_database17";        //声明数据库 db_database17 的 URL
String user = "root";                                            //数据库用户名
String password = "111";                                         //数据库密码
//建立数据库连接，获得连接对象 conn
Connection conn = DriverManager.getConnection(url, user, password);
String sql="delete from tb_users where id=1";                    //生成一条 SQL 语句
Statement stmt = conn.createStatement();                         //创建一个 Statement 对象
stmt.executeUpdate(sql);                                         //执行 SQL 语句
System.out.println("删除数据成功！");
conn.close();                                                    //关闭数据库连接对象
```

运行结果如图 17.11 所示。

图 17.11　删除数据成功提示

17.3.7　模糊查询

模糊查询是比较常见的一种查询方式。例如在订单表中，包含有订单的具体日期。如果要查询某年某月的订单信息，最好的方式就是使用模糊查询。进行模糊查询需要使用关键字 LIKE。在使用 LIKE 关键字进行模糊查询时，可以使用通配符“%”来代替 0 个或多个字符，使用下划线来代替一个字符。

【例 17.10】　查询数据库 tb_users 表中用户名带李的用户信息。（**实例位置：光盘\TM\sl\17\7**）

```
Class.forName("com.mysql.jdbc.Driver");                          //加载数据库驱动
String url = "jdbc:mysql://localhost:3306/db_database17";        //声明数据库 db_database17 的 URL
String user = "root";                                            //数据库用户名
String password = "111";                                         //数据库密码
//建立数据库连接，获得连接对象 conn
Connection conn = DriverManager.getConnection(url, user, password);
Statement stmt=conn.createStatement();
String sql="select * from tb_users where username like '%李%'";  //生成 SQL 语句
ResultSet rs=stmt.executeQuery(sql);                             //执行 SQL 语句
int id,age;
String username,pwd,sex;
System.out.println("id\t 用户名\t 密码\t 性别\t 年龄");
while(rs.next()){
    id=rs.getInt("id");
    username=rs.getString(2);
    pwd=rs.getString("password");
    sex=rs.getString(4);
    age=rs.getInt("age");
    System.out.println(id+"\t"+username+"\t"+pwd+"\t"+sex+"\t"+age);   //输出查询结果
}
```

运行结果如图 17.12 所示。

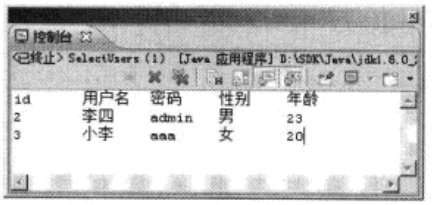


图 17.12　查询表中用户名带李的用户信息

17.3.8　范例 1：将复选框内容存到数据库

在用户注册模块中，要求用户添加个人爱好是十分常见的，通常情况下，系统会提供给用户以复选框的形式来选择。本范例要求读者实现将用户选择的爱好信息，以字符串的形式保存到数据库中。运行结果如图 17.13 所示。（**实例位置：光盘\TM\sl\17\8**）

图 17.13　保存爱好信息

（1）在项目中创建 ToStringFrame 类，该类继承 JFrame 类，实现窗体类。代码如下：

```
public ToStringFrame() {
    checkBox = new JCheckBox("旅游");                    //添加复选框
    checkBox.setBounds(119, 72, 63, 23);
    panel.add(checkBox);
    checkBox_1 = new JCheckBox("看书");                  //添加复选框
    checkBox_1.setBounds(184, 72, 63, 23);
    panel.add(checkBox_1);
    checkBox_2 = new JCheckBox("上网");                  //添加复选框
    checkBox_2.setBounds(119, 115, 63, 23);
    panel.add(checkBox_2);
    checkBox_3 = new JCheckBox("音乐");                  //添加复选框
    checkBox_3.setBounds(259, 72, 103, 23);
    panel.add(checkBox_3);
    checkBox_4 = new JCheckBox("羽毛球");                //添加复选框
    checkBox_4.setBounds(184, 115, 74, 23);
    panel.add(checkBox_4);
    checkBox_5 = new JCheckBox("乒乓球");                //添加复选框
    checkBox_5.setBounds(259, 115, 74, 23);
    panel.add(checkBox_5);
    JButton insertButton = new JButton("保存");          //添加“保存”按钮
```

```
    insertButton.addActionListener(new ActionListener() {      //调用“保存”按钮的单击事件
        public void actionPerformed(ActionEvent arg0) {
            do_button_actionPerformed(arg0);
        }
    });
}
```

（2）在项目中创建工具类 InsertString，在该类中定义获得数据库连接的方法 getConn()。同时该类中还定义了向用户表中添加数据的方法 insertUsers()，该方法需要两个参数分别表示用户名和爱好。代码如下：

```
public Connection getConn() {
    try {
        Class.forName("com.mysql.jdbc.Driver");                          //加载数据库驱动
    } catch (ClassNotFoundException e) {
        e.printStackTrace();
    }
    String url = "jdbc:mysql://localhost:3306/db_database17";            //连接数据库 URL
    String userName = "root";                                            //连接数据库的用户名
    String passWord = "111";                                             //连接数据库密码
    try {
        conn = DriverManager.getConnection(url, userName, passWord);     //获取数据库连接
        if (conn != null) {
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return conn;                                                         //返回 Connection 对象
}
public void insertUsers(String name ,String loves) {
    conn = getConn();                                                    //获取数据库连接
    try {
        PreparedStatement statement = conn
                //定义插入数据库的预处理语句
                .prepareStatement("insert into tb_student (sname,hobby) values(?,?)");
        statement.setString(1,name);
        statement.setString(2,loves);
        statement.executeUpdate();                                       //执行预处理语句
        conn.close();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

（3）在窗体类中编写“保存”按钮的单击事件。当用户单击“保存”按钮时获得复选框选中的信息，把获得的信息转换成字符串，调用 InsertString 类中的 insertUsers()方法把用户名和转换的字符串保存到数据库中。代码如下：

```
if (checkBox.isSelected()) {                    //判断复选框是否被选中
    buff.append(checkBox.getText() + "、");     //将指定的字符串追加到此字符序列
```

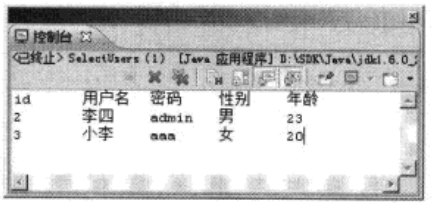


图 17.12　查询表中用户名带李的用户信息

17.3.8　范例 1：将复选框内容存到数据库

在用户注册模块中，要求用户添加个人爱好是十分常见的，通常情况下，系统会提供给用户以复选框的形式来选择。本范例要求读者实现将用户选择的爱好信息，以字符串的形式保存到数据库中。运行结果如图 17.13 所示。（**实例位置：光盘\TM\sl\17\8**）

图 17.13　保存爱好信息

（1）在项目中创建 ToStringFrame 类，该类继承 JFrame 类，实现窗体类。代码如下：

```
public ToStringFrame() {
    checkBox = new JCheckBox("旅游");                    //添加复选框
    checkBox.setBounds(119, 72, 63, 23);
    panel.add(checkBox);
    checkBox_1 = new JCheckBox("看书");                  //添加复选框
    checkBox_1.setBounds(184, 72, 63, 23);
    panel.add(checkBox_1);
    checkBox_2 = new JCheckBox("上网");                  //添加复选框
    checkBox_2.setBounds(119, 115, 63, 23);
    panel.add(checkBox_2);
    checkBox_3 = new JCheckBox("音乐");                  //添加复选框
    checkBox_3.setBounds(259, 72, 103, 23);
    panel.add(checkBox_3);
    checkBox_4 = new JCheckBox("羽毛球");                //添加复选框
    checkBox_4.setBounds(184, 115, 74, 23);
    panel.add(checkBox_4);
    checkBox_5 = new JCheckBox("乒乓球");                //添加复选框
    checkBox_5.setBounds(259, 115, 74, 23);
    panel.add(checkBox_5);
    JButton insertButton = new JButton("保存");          //添加“保存”按钮
```

```
    insertButton.addActionListener(new ActionListener() {      //调用“保存”按钮的单击事件
        public void actionPerformed(ActionEvent arg0) {
            do_button_actionPerformed(arg0);
        }
    });
}
```

（2）在项目中创建工具类 InsertString，在该类中定义获得数据库连接的方法 getConn()。同时该类中还定义了向用户表中添加数据的方法 insertUsers()，该方法需要两个参数分别表示用户名和爱好。代码如下：

```
public Connection getConn() {
    try {
        Class.forName("com.mysql.jdbc.Driver");                          //加载数据库驱动
    } catch (ClassNotFoundException e) {
        e.printStackTrace();
    }
    String url = "jdbc:mysql://localhost:3306/db_database17";            //连接数据库 URL
    String userName = "root";                                            //连接数据库的用户名
    String passWord = "111";                                             //连接数据库密码
    try {
        conn = DriverManager.getConnection(url, userName, passWord);     //获取数据库连接
        if (conn != null) {
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return conn;                                                         //返回 Connection 对象
}
public void insertUsers(String name ,String loves) {
    conn = getConn();                                                    //获取数据库连接
    try {
        PreparedStatement statement = conn
                //定义插入数据库的预处理语句
                .prepareStatement("insert into tb_student (sname,hobby) values(?,?)");
        statement.setString(1,name);
        statement.setString(2,loves);
        statement.executeUpdate();                                       //执行预处理语句
        conn.close();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

（3）在窗体类中编写“保存”按钮的单击事件。当用户单击“保存”按钮时获得复选框选中的信息，把获得的信息转换成字符串，调用 InsertString 类中的 insertUsers()方法把用户名和转换的字符串保存到数据库中。代码如下：

```
if (checkBox.isSelected()) {                    //判断复选框是否被选中
    buff.append(checkBox.getText() + "、");     //将指定的字符串追加到此字符序列
```

```
    }
    return conn;
}
```

（2）创建学生信息的实体类 Students，用来接收和传递学生的信息。代码如下：

```
private int id;                                    //学生学号
private String sname;                              //学生姓名
private String sex;                                //学生性别
private int age;                                   //学生年龄
private String hobby;                              //学生爱好
private String sclass;                             //学生班级
……                                                 //省略了 Getter()和 Setter()方法
```

（3）创建操作数据库的实现类 StuDao，在该类中定义了增加、删除、修改学生信息等方法。代码如下：

```
public int addusers(Students stu){                 //添加学生信息
    int i=0;
    DbTools db=new DbTools();
    Connection conn=db.getConn();                  //创建数据库连接对象
    try {
        //生成一条 SQL 语句
        String sql="insert into tb_student (sname,sex,age,hobby,sclass)values(?,?,?,?,?)";
        PreparedStatement ps=conn.prepareStatement(sql);   //创建 PreparedStatement 对象
        ps.setString(1, stu.getSname());           //为 SQL 中的？赋值
        ps.setString(2, stu.getSex());
        ps.setInt(3, stu.getAge());
        ps.setString(4, stu.getHobby());
        ps.setString(5, stu.getSclass());
        i=ps.executeUpdate();                      //执行添加方法
        ps.close();
        conn.close();                              //关闭连接
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return i;
}
public int updateusers(Students stu,int id){       //修改学生信息
    int i=0;
    DbTools db=new DbTools();
    Connection conn=db.getConn();                  //创建数据库连接对象
    try {
        //生成一条 SQL 语句
        String sql="update tb_student set sname=?,sex=?,age=?,hobby=?,sclass=? where id=?";
        PreparedStatement ps=conn.prepareStatement(sql);   //创建 PreparedStatement 对象
        ps.setString(1, stu.getSname());           //为 SQL 中的？赋值
        ps.setString(2, stu.getSex());
        ps.setInt(3, stu.getAge());
```

```
            ps.setString(4, stu.getHobby());
            ps.setString(5, stu.getSclass());
            ps.setInt(6, id);
            i=ps.executeUpdate();                          //执行修改方法
            ps.close();
            conn.close();                                  //关闭连接
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return i;
    }
    public List selectAll(){                               //查询所有学生信息
        String sql="select * from tb_student";             //生成一条 SQL 语句
        DbTools db=new DbTools();
        Connection conn=db.getConn();                      //创建数据库连接对象
        ResultSet rs=null;
        List<Students> ls=new ArrayList<Students>();       //创建 List 集合并规定其泛型
        try {
            Statement st=conn.createStatement();           //创建 Statement 对象
            //执行查询，把查询结果集赋值给 ResultSet 对象
            rs=st.executeQuery(sql);
            while(rs.next()){                              //遍历查询结果集
                Students stu=new Students();
                stu.setId(rs.getInt("id"));                //把查询到的结果赋值给 Students 对象
                stu.setSname(rs.getString("sname"));
                stu.setSex(rs.getString("sex"));
                stu.setAge(rs.getInt("age"));
                stu.setHobby(rs.getString("hobby"));
                stu.setSclass(rs.getString("sclass"));
                ls.add(stu);                               //把 Students 对象添加到集合中
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return ls;                                         //返回 List 对象
    }
    public Students getOne(int id){                        //查询一个学生信息
        Students stu=new Students();
        String sql="select * from tb_student where id="+id;   //生成一条 SQL 语句
        DbTools db=new DbTools();
        Connection conn=db.getConn();
        ResultSet rs=null;
        try {
            Statement st=conn.createStatement();
            rs=st.executeQuery(sql);
            while(rs.next()){                              //遍历查询结果集
                stu.setSname(rs.getString("sname"));       //把查询到的结果赋值给 Students 对象
                stu.setSex(rs.getString("sex"));
                stu.setAge(rs.getInt("age"));
```

```
                stu.setHobby(rs.getString("hobby"));
                stu.setSclass(rs.getString("sclass"));
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return stu;                                                    //返回 Students 对象
    }
    public void deleteStu(int id){                                     //删除学生信息
        String sql="delete from tb_student where id="+id;              //生成一条 SQL 语句
        DbTools db=new DbTools();
        Connection conn=db.getConn();                                  //创建数据库连接
        ResultSet rs=null;
        try {
            Statement st=conn.createStatement();                       //创建 Statement 对象
            st.execute(sql);                                           //执行删除语句

        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
```

说明　这里先把具体的实现方法写好，这样当用户创建出一个功能窗体时直接调用写好的方法，就可以实现这个功能。

（4）创建主窗体 MainFrame 类，当启动主窗体时查询数据库学生表中的所有数据，把这些数据以表格的形式显示到主窗体中。在主窗体中还添加了“添加”、“修改”、“删除”和“关闭”4 个功能按钮。代码如下：

```
public void selecttable() {                                            //显示学生信息的表格
    table = new JTable();
    defaultModel = (DefaultTableModel) table.getModel();               //获得表格模型
    defaultModel.setRowCount(0);                                       //清空表格模型中的数据
    defaultModel.setColumnIdentifiers(new Object[] { "学号", "姓名", "性别",
            "年龄", "班级" });                                         //定义表头
    table.getTableHeader().setReorderingAllowed(false);
    table.setModel(defaultModel);                                      //设置表格模型
    List list = dao.selectAll();
    for (int i = 0; i < list.size(); i++) {
        Students stu = (Students) list.get(i);
        defaultModel.addRow(new Object[] { stu.getId(), stu.getSname(),
                stu.getSex(), stu.getAge(), stu.getSclass() });        //为表格赋值
    }
    scrollPane.setViewportView(table);
}
```

（5）单击主窗体中的“添加”按钮会弹出添加学生信息窗体，用来添加学生信息。运行结果如

图 17.16 所示。代码如下：

图 17.16　添加学生信息

```
protected void do_insertButton_actionPerformed(ActionEvent arg0) {
    StuDao dao = new StuDao();
    Students stu = new Students();
    if (checkBox.isSelected()) {
        buff.append(checkBox.getText() + "、");
    }
    if (checkBox_1.isSelected()) {                          //判断复选框是否被选中
        buff.append(checkBox_1.getText() + "、");           //将指定的字符串追加到此字符序列
    }
    if (checkBox_2.isSelected()) {                          //判断复选框是否被选中
        buff.append(checkBox_2.getText() + "、");           //将指定的字符串追加到此字符序列
    }
    if (checkBox_3.isSelected()) {                          //判断复选框是否被选中
        buff.append(checkBox_3.getText() + "、");           //将指定的字符串追加到此字符序列
    }
    if (checkBox_4.isSelected()) {                          //判断复选框是否被选中
        buff.append(checkBox_4.getText() + "、");           //将指定的字符串追加到此字符序列
    }
    if (checkBox_5.isSelected()) {                          //判断复选框是否被选中
        buff.append(checkBox_5.getText() + "、");           //将指定的字符串追加到此字符序列
    }
    String strLoves = buff.toString();                      //获得序列中数据的字符串表示形式
    stu.setSname(sname.getText());                          //为姓名赋值
    stu.setSex(sexComboBox.getSelectedItem().toString());   //为性别赋值
    stu.setAge(Integer.parseInt(age.getText()));            //为年龄赋值
    stu.setSclass(sclass.getText());                        //为班级赋值
    stu.setHobby(strLoves);                                 //为爱好赋值
    if (!(sname.getText().equals(""))
            && (!sclass.getText().equals(""))
            && (!age.getText().equals(""))) {               //判断学生信息是否添加完整
        dao.addusers(stu);                                  //执行添加方法
        JOptionPane.showMessageDialog(getContentPane(), "数据添加成功！", "信息提示框",
                JOptionPane.WARNING_MESSAGE);               //添加成功提示
    } else {
```

```
        JOptionPane.showMessageDialog(getContentPane(), "请将信息添加完整！",
                "信息提示框", JOptionPane.WARNING_MESSAGE);
    }
}
```

（6）单击主窗体中的“修改”按钮会弹出修改学生信息窗体，用来修改学生信息，注意如果没有选中要修改的学生信息会弹出提示对话框。运行结果如图 17.17 所示。代码如下：

图 17.17　修改学生信息

```
Students stu=dao.getOne(id);                          //获得要修改的学生信息
sname.setText(stu.getSname());                        //为学生姓名赋值
age.setText(stu.getAge()+"");                         //为学生年龄赋值
sclass.setText(stu.getSclass());                      //为学生班级赋值
String[] hy=stu.getHobby().split("、");
if(hy.length>0){
    for(int i=0;i<hy.length;i++){                     //为爱好赋值
        if(hy[i].equals("旅游")){                     //判断所要修改学生的爱好
            checkBox.setSelected(true);               //设置学生爱好的复选框为选中
        }
        if(hy[i].equals("看书")){
            checkBox_1.setSelected(true);
        }
        if(hy[i].equals("上网")){
            checkBox_2.setSelected(true);
        }
        if(hy[i].equals("音乐")){
            checkBox_3.setSelected(true);
        }
        if(hy[i].equals("羽毛球")){
            checkBox_4.setSelected(true);
        }
        if(hy[i].equals("乒乓球")){
            checkBox_5.setSelected(true);
        }
    }
}
```

```
if("男".equals(stu.getSex())){                                    //为性别赋值
    sexComboBox.setSelectedIndex(0);
}else{
    sexComboBox.setSelectedIndex(1);
}
```

说明 修改学生信息的难点就在于当修改窗体加载时显示学生的信息，特别是显示复选框的内容。当修改完毕后单击“修改”按钮，调用 StuDao 类中的 updateusers()方法实现修改学生信息。

（7）单击“删除”按钮，执行删除操作。主要是获得学生的学号，根据学号调用 StuDao 类中的 deleteStu()方法，删除学生信息。运行结果如图 17.18 所示。

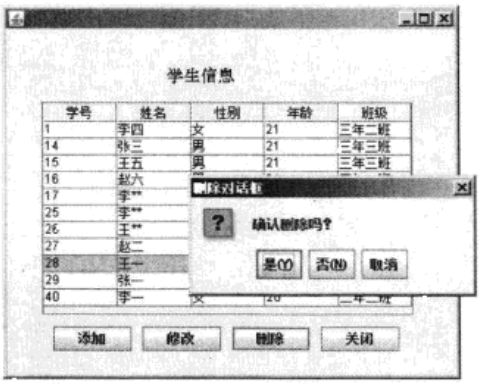


图 17.18　删除学生信息

17.4.2　经典范例 2：权限管理

视频讲解：光盘\TM\lx\17\权限管理.exe

在以后的开发中，权限管理会被经常用到，最常见的就是管理员，管理员有着比普通用户更高的权限。现在就做一个管理员登录的例子，当管理员登录成功后可以删除普通用户，而普通用户登录以后不能进行删除操作。（**实例位置：光盘\TM\sl\17\11**）

（1）首先创建操作数据库连接类 DbTools。代码如下：

```
Connection conn;
public Connection getConn(){
    try {
        Class.forName("com.mysql.jdbc.Driver");                    //加载数据库驱动
        String url="jdbc:mysql://localhost:3306/db_database17";    //声明数据库 db_database17 的 URL
        String user="root";                                        //数据库用户名
        String password="111";                                     //数据库密码
        conn=DriverManager.getConnection(url, user, password);     //建立数据库连接，获得连接对象 conn
    } catch (ClassNotFoundException e) {
        e.printStackTrace();
```

```
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return conn;
}
```

（2）创建用户信息的实体类 Users，用来接收和传递用户的信息。代码如下：

```
private int id;                                          //用户 id
private String username;                                 //用户名
private String password;                                 //用户密码
private String popedom;                                  //用户权限
```

（3）创建操作数据库的实现类 UserDao，在该类中定义了实现用户登录、判断权限、用户注册等方法。代码如下：

```
public List<Users> selectAll() {                         //查询用户信息
    String sql = "select * from tb_login";               //生成查询用户信息的 SQL 语句
    DbTools db = new DbTools();
    Connection conn = db.getConn();                      //创建 Connection 对象
    List<Users> ls=new ArrayList<Users>();               //创建一个集合
    try {
        Statement st = conn.createStatement();
        ResultSet rs = st.executeQuery(sql);             //执行查询语句
        while(rs.next()){                                //遍历结果集
            if(rs.getInt(4)!=1){                         //判断是否是管理员
                Users us = new Users();
                us.setId(rs.getInt(1));                  //为 id 赋值
                us.setUsername(rs.getString(2));         //为用户名赋值
                us.setPassword(rs.getString(3));         //为用户密码赋值
                us.setPopedom(rs.getString(4));          //为权限赋值
                ls.add(us);                              //把用户对象添加到集合中
            }
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return ls;
}
public int loginuser(String name, String pwd) {          //用户登录
    String sql = "select popedom from tb_login where username='" + name
            + "' and password='" + pwd+"'";              //生成 SQL 语句
    DbTools db = new DbTools();
    Connection conn = db.getConn();
    try {
        Statement st = conn.createStatement();
        ResultSet rs = st.executeQuery(sql);             //执行 SQL 语句
        if (rs.next()){                                  //判断结果集中是否有值
            if (rs.getInt(1) == 1) {                     //判断登录用户的权限值是否为 1
                return 1;                                //1 表示的权限为管理员
```

```
                } else {
                        return 2;                                   //2 表示的权限为普通用户
                }
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return 3;                                                   //3 表示无此用户
}
public void addUser(String name,String pwd){                        //用户注册
        //生成添加的 SQL 语句
        String sql="insert into tb_login (username,password)values('"+name+"','"+pwd+"')";
        DbTools db = new DbTools();
        Connection conn = db.getConn();
        try {
            Statement st = conn.createStatement();
            st.executeUpdate(sql);                                  //执行 SQL 语句
            conn.close();                                           //关闭 Connection 对象
        } catch (SQLException e) {
            e.printStackTrace();
        }
}
public void deleteUser(int id){                                     //删除用户
        String sql="delete from tb_login where id="+id;             //生成 SQL 语句
        DbTools db=new DbTools();
        Connection conn=db.getConn();
        ResultSet rs=null;
        try {
            Statement st=conn.createStatement();                    //执行删除语句
            st.execute(sql);
        } catch (SQLException e) {
            e.printStackTrace();
        }
}
```

（4）创建登录窗体类 RegistDemo，在该类中定义了“登录”和“注册”两个按钮，登录窗体运行结果如图 17.19 所示。代码如下：

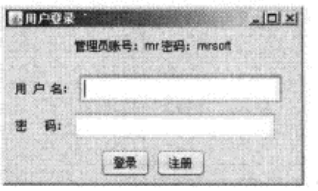


图 17.19　用户登录窗体

```
protected void do_button2_actionPerformed(ActionEvent e) {          //“登录”按钮事件
        UserDao dao=new UserDao();
        int i=dao.loginuser(uname.getText(), pwd1.getText());       //调用用户登录方法
```

```
    if(i==3){                                                    //判断是否有此用户
        JOptionPane.showMessageDialog(this, "登录失败", "",
                JOptionPane.WARNING_MESSAGE);
        return;
    }else{
        UserUtil u=new UserUtil();
        u.setPopedom(i);                                         //传递该用户的权限
        SelectFrame sf=new SelectFrame();
        sf.setVisible(true);                                     //将查看用户窗体设置为可见
        this.setVisible(false);                                  //将登录窗体隐藏
    }
}
protected void do_button3_actionPerformed(ActionEvent e) {       //“注册”按钮事件
    AddFrame add=new AddFrame();
    add.setVisible(true);                                        //将注册窗体设置为可见
    this.setVisible(false);                                      //将登录窗体隐藏
}
```

（5）单击登录窗体中的“注册”按钮，弹出注册窗体，在该窗体中实现用户注册功能。运行结果如图 17.20 所示。代码如下：

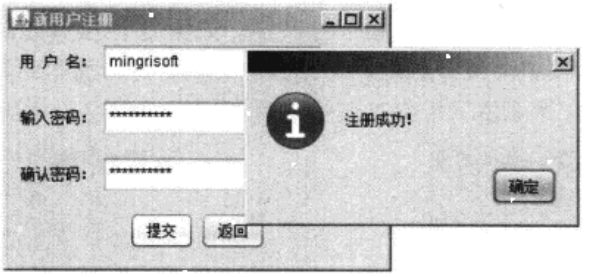


图 17.20　用户注册窗体

```
protected void do_button2_actionPerformed(ActionEvent e) {       //“提交”按钮事件
    //生成 SQL 语句
    String sql = "select * from tb_login where username = '" +textField.getText()+ "';";
    DbTools db=new DbTools();
    Connection conn=db.getConn();
    ResultSet rs= null;
    try {
        Statement st=conn.createStatement();
        rs=st.executeQuery(sql);                                 //执行 SQL 语句
        if (!rs.next()) {                                        //判断用户名是否可用
            if(pwd1.getText().equals(pwd2.getText())){           //判断两次密码是否一致
                UserDao dao=new UserDao();
                dao.addUser(textField.getText(), pwd1.getText());   //调用添加用户方法
                JOptionPane.showMessageDialog(this, "注册成功！", null, JOptionPane.INFORMATION_ MESSAGE);
                this.setVisible(false);                          //将注册窗体隐藏
                RegistDemo r=new RegistDemo();
                r.setVisible(true);                              //将登录窗体设置为可见
```

```
        }else{
            JOptionPane.showMessageDialog(this, "两次密码不一致！", null, JOptionPane.INFORMATION_
MESSAGE);                                                    //提示两次密码不一致
            return;
        }
        } else {                                             //提示用户名冲突
            JOptionPane.showMessageDialog(this, "用户名冲突！", null, JOptionPane.INFORMATION_
MESSAGE);
            return;
        }
    } catch (SQLException e1) {
        e1.printStackTrace();
    }
}
```

说明 注册的新用户权限都为普通权限，如果想让某普通用户的权限为管理员，需要在数据库中将该用户的 popedom 字段改为 1。1 为管理员，2 为普通用户。

（6）在登录窗体中输入用户名和密码后单击“登录”按钮，如果用户名和密码相匹配将登录到查看用户窗体。当加载该窗体时，判断登录用户的权限，如果是管理员登录会显示“删除”按钮，如图 17.21 所示，单击“删除”按钮会进行删除用户的操作。如果是普通用户登录则不显示“删除”按钮，如图 17.22 所示。查看用户窗体中显示了除管理员外所有用户的信息。代码如下：

图 17.21　管理员登录

图 17.22　普通用户登录

```
if(UserUtil.getNumber()==1){                                    //判断用户是否为管理员
JButton deleteButton = new JButton("删除");                     //创建“删除”按钮
deleteButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent arg0) {
        do_deleteButton_actionPerformed(arg0);                  //调用“删除”按钮事件
    }
});
protected void do_deleteButton_actionPerformed(ActionEvent arg0) {   //“删除”按钮单击事件
    int row = table.getSelectedRow();
    if (row == -1) {
        //如果用户没有选择任何行，则进行提示
```

```
            JOptionPane.showMessageDialog(this, "请选择要删除的用户", "",
                        JOptionPane.WARNING_MESSAGE);
            return;
        }
        if (row >= 0) {
            int n = JOptionPane.showConfirmDialog(getContentPane(), "确认删除吗？",
                        "删除对话框", JOptionPane.YES_NO_CANCEL_OPTION);
            if (n == JOptionPane.YES_OPTION) {                    //如果用户确认信息
                dao.deleteUser(Integer.parseInt(table.getValueAt(row, 0)
                        .toString()));                            //获取要删除的用户 id，执行删除方法
            }
            selecttable();
        }
}
```

17.5　本 章 小 结

本章结合实例对 JDBC 编程进行了讲解，包括 JDBC 的常用类和接口的使用，数据库驱动的加载，数据连接的建立，添加、修改、删除和查询数据库表中的数据，最后对 JDBC 的事务进行了讲解，通过本章的学习，读者应该能够独立完成数据库程序的设计。

17.6　实 战 练 习

1. 删除表格中年龄为 20 的全部记录。（**答案位置：光盘\TM\sl\17\12**）
2. 使用 Java 程序获得表结构。（**答案位置：光盘\TM\sl\17\13**）
3. 注册时验证用户名是否可以。（**答案位置：光盘\TM\sl\17\14**）
4. 查询男生和女生的人数。（**答案位置：光盘\TM\sl\17\15**）

第18章

Swing 高级组件

（视频讲解：48 分钟）

表格是最常用的数据统计形式之一，在日常生活中经常需要使用表格统计数据，如对销售数据的统计、日常开销的统计，以及生成员工待遇报表等。本章将学习 Swing 表格的使用方法，最后还讲解了提供行标题栏表格的实现思路和方法。在讲解过程中为了便于读者理解结合了大量的举例。

通过阅读本章，您可以：

- 学会创建 Swing 表格的方法
- 学会定义和操纵表格的常用方法
- 学会维护表格模型的常用方法
- 掌握 JTable 和 DefaultTableModel 类的主要功能
- 掌握提供行标题栏表格的开发思路
- 了解 Swing 表格的设计思路

PDG

18.1 利用 JTable 类直接创建表格

视频讲解：光盘\TM\lx\18\利用 JTable 类直接创建表格.exe

表格是最常用的数据统计形式之一，在 Swing 中由 JTable 类实现表格。本节将学习利用 JTable 类创建和定义表格，以及操纵表格的方法。

18.1.1 创建表格

在 JTable 类中除了默认的构造方法外，还提供了利用指定表格列名数组和表格数据数组创建表格的构造方法，语法格式如下：

```
JTable(Object[][] rowData, Object[] columnNames)
```

☑ rowData：封装表格数据的数组。

☑ columnNames：封装表格列名的数组。

在使用表格时，通常将其添加到滚动面板中，然后将滚动面板添加到相应的位置。

【例 18.1】 利用构造方法 JTable(Object[][] rowData, Object[] columnNames)创建一个表格，并将表格添加到滚动面板中。（**实例位置：光盘\TM\sl\18\1**）

```
public class ExampleFrame_01 extends JFrame {
    public static void main(String args[]) {
        ExampleFrame_01 frame = new ExampleFrame_01();
        frame.setVisible(true);
    }
    public ExampleFrame_01() {
        super();
        setTitle("创建可以滚动的表格");
        setBounds(100, 100, 240, 150);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        String[] columnNames = { "A", "B" };                          //定义表格列名数组
        String[][] tableValues = { { "A1", "B1" }, { "A2", "B2" },
                { "A3", "B3" }, { "A4", "B4" }, { "A5", "B5" } };     //定义表格数据数组
        JTable table = new JTable(tableValues, columnNames);          //创建指定列名和数据的表格
        JScrollPane scrollPane = new JScrollPane(table);              //创建显示表格的滚动面板
        getContentPane().add(scrollPane, BorderLayout.CENTER);        //将滚动面板添加到边界布局的中间
    }
}
```

运行结果如图 18.1 所示。

当调小窗体的高度时，将出现滚动条，如图 18.2 所示。

图 18.1　创建可以滚动的表格

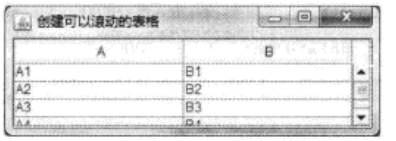


图 18.2　支持滚动条的表格

在 JTable 类中还提供了利用指定表格列名向量和表格数据向量创建表格的构造方法，语法格式如下：

```
JTable(Vector rowData, Vector columnNames)
```

☑　rowData：封装表格数据的向量。

☑　columnNames：封装表格列名的向量。

在使用表格时，有时并不需要使用滚动条，即在窗体中可以显示出整个表格，在这种情况下，也可以直接将表格添加到相应的容器中。

注意　如果是直接将表格添加到相应的容器中，则需要通过 JTable 类的 getTableHeader()方法获得 JTableHeader 类的对象，然后再将该对象添加到容器的相应位置，否则表格将没有列名。

【例 18.2】　利用构造方法 JTable(Vector rowData, Vector columnNames)创建一个表格，并将表格直接添加到容器中。（**实例位置：光盘\TM\sl\18\2**）

```
Vector columnNameV = new Vector();                          //定义表格列名向量
columnNameV.add("A");                                       //添加列名
columnNameV.add("B");                                       //添加列名
Vector tableValueV = new Vector();                          //定义表格数据向量
for (int row = 1; row < 6; row++) {
    Vector rowV = new Vector();                             //定义表格行向量
    rowV.add("A" + row);                                    //添加单元格数据
    rowV.add("B" + row);                                    //添加单元格数据
    tableValueV.add(rowV);                                  //将表格行向量添加到表格数据向量中
}
JTable table = new JTable(tableValueV, columnNameV);        //创建指定表格列名和表格数据的表格
getContentPane().add(table, BorderLayout.CENTER);           //将表格添加到边界布局的中间
JTableHeader tableHeader = table.getTableHeader();          //获得表格头对象
getContentPane().add(tableHeader, BorderLayout.NORTH);      //将表格头添加到边界布局的上方
```

运行结果如图 18.3 所示。

当调小窗体的高度时，仍不会出现滚动条，如图 18.4 所示。

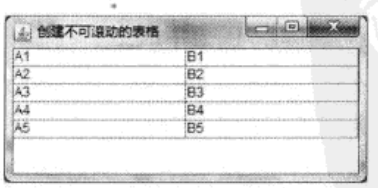


图 18.3　创建不可滚动的表格

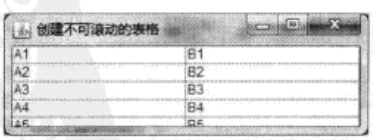


图 18.4　不支持滚动条的表格

18.1.2 定制表格

表格创建完成后，还需要对其进行一系列的定义，以便适合于具体的使用情况。默认情况下通过双击表格中的单元格就可以对其进行编辑，如图 18.5 所示。

如果不需要提供该功能，可以通过重构 JTable 类的 isCellEditable(int row, int column)方法来实现。默认情况下该方法返回 boolean 型值 true，表示指定单元格可编辑，如果返回 false 则表示不可编辑。

如果表格只有几列，通常不需要表格列的可重新排列功能。在创建不支持滚动条的表格时已经使用了 JTableHeader 类的对象，通过该类的 setReorderingAllowed(boolean reorderingAllowed)方法即可设置表格是否支持重新排列功能，设为 false 表示不支持重新排列功能，如图 18.6 所示。

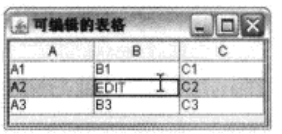


图 18.5　可编辑的表格

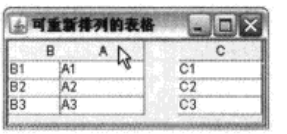


图 18.6　可重新排列的表格

默认情况下单元格中的内容靠左侧显示，如果需要令单元格中的内容居中显示，可以通过重构 JTable 类的 getDefaultRenderer(Class<?> columnClass)方法来实现。重构后的代码如下：

```
public TableCellRenderer getDefaultRenderer(Class<?> columnClass) {
    DefaultTableCellRenderer cr = (DefaultTableCellRenderer) super.getDefaultRenderer(columnClass);
    cr.setHorizontalAlignment(DefaultTableCellRenderer.CENTER);
    return cr;
}
```

运行结果如图 18.7 所示。

图 18.7　单元格内容居中显示

表 18.1 列出了 JTable 类中用来定义表格的常用方法。

表 18.1　JTable 类中用来定义表格的常用方法

方　法	说　明
setRowHeight(int rowHeight)	设置表格的行高，默认为 16 像素
setRowSelectionAllowed(boolean sa)	设置是否允许选中表格行，默认为允许选中，设为 false 表示不允许选中
setSelectionMode(int sm)	设置表格行的选择模式
setSelectionBackground(Color bc)	设置表格选中行的背景色
setSelectionForeground(Color fc)	设置表格选中行的前景色（通常情况下为文字的颜色）
setAutoResizeMode(int mode)	设置表格的自动调整模式

在利用 setSelectionMode(int sm)方法设置表格行的选择模式时，它的入口参数可以从表 18.2 列出的 ListSelectionModel 类的静态常量中选择。

表 18.2　ListSelectionModel 类中用来设置选择模式的静态常量

静 态 常 量	常 量 值	代表的选择模式
SINGLE_SELECTION	0	只允许选择一个，如图 18.8 所示
SINGLE_INTERVAL_SELECTION	1	允许连续选择多个，如图 18.9 所示
MULTIPLE_INTERVAL_SELECTION	2	可以随意选择多个，如图 18.10 所示

图 18.8　单选模式

图 18.9　连选模式

图 18.10　多选模式

在利用 setAutoResizeMode(int mode)方法设置表格的自动调整模式时，它的入口参数可以从表 18.3 列出的 JTable 类的静态常量中选择。

说明　所谓表格的自动调整模式，就是在调整表格某一列的宽度时，表格采用何种方式保持其总宽度不变。

表 18.3　JTable 类中用来设置自动调整模式的静态常量

静 态 常 量	常 量 值	代表的自动调整模式
AUTO_RESIZE_OFF	0	关闭自动调整功能，使用水平滚动条时的必要设置，如图 18.11 所示
AUTO_RESIZE_NEXT_COLUMN	1	只调整其下一列的宽度，如图 18.12 所示
AUTO_RESIZE_SUBSEQUENT_COLUMNS	2	按比例调整其后所有列的宽度，为默认设置，如图 18.13 所示
AUTO_RESIZE_LAST_COLUMN	3	只调整最后一列的宽度，如图 18.14 所示
AUTO_RESIZE_ALL_COLUMNS	4	按比例调整表格所有列的宽度，如图 18.15 所示

说明　当调整表格所在窗体的宽度时，如果关闭了表格的自动调整功能，表格的总宽度仍保持不变，如图 18.16 所示；如果开启了表格的自动调整功能，表格将按比例调整所有列的宽度至适合窗体的宽度，如图 18.17 所示。

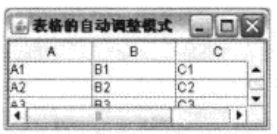


图 18.11　OFF

图 18.12　NEXT

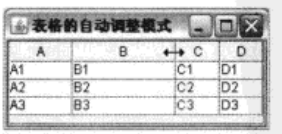


图 18.13　SUB

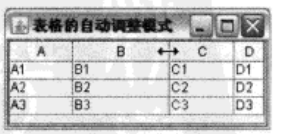


图 18.14　LAST

图 18.15　ALL

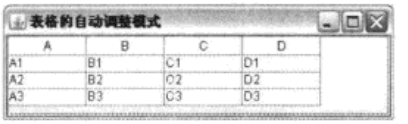


图 18.16　关闭的情况下调整窗体宽度

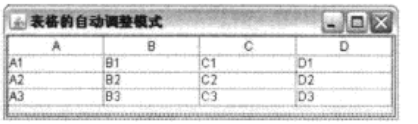


图 18.17　开启的情况下调整窗体宽度

【例 18.3】　利用本节所讲的全部知识对表格进行定义。（**实例位置：光盘\TM\sl\18\3**）

```
public class ExampleFrame_03 extends JFrame {
    public static void main(String args[]) {
        ExampleFrame_03 frame = new ExampleFrame_03();
        frame.setVisible(true);
    }
    public ExampleFrame_03() {
        super();
        setTitle("定义表格");
        setBounds(100, 100, 500, 375);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        final JScrollPane scrollPane = new JScrollPane();
        getContentPane().add(scrollPane, BorderLayout.CENTER);
        String[] columnNames = { "A", "B", "C", "D", "E", "F", "G" };
        Vector columnNameV = new Vector();
        for (int column = 0; column < columnNames.length; column++) {
            columnNameV.add(columnNames[column]);
        }
        Vector tableValueV = new Vector();
        for (int row = 1; row < 21; row++) {
            Vector rowV = new Vector();
            for (int column = 0; column < columnNames.length; column++) {
                rowV.add(columnNames[column] + row);
            }
            tableValueV.add(rowV);
        }
        JTable table = new MTable(tableValueV, columnNameV);
        table.setAutoResizeMode(JTable.AUTO_RESIZE_OFF);      //关闭表格列的自动调整功能
        table.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);          //选择模式为单选
        table.setSelectionBackground(Color.YELLOW);          //被选择行的背景色为黄色
        table.setSelectionForeground(Color.RED);             //被选择行的前景色（文字颜色）为红色
        table.setRowHeight(30);                              //表格的行高为 30 像素
        scrollPane.setViewportView(table);
    }
    private class MTable extends JTable {                    //实现自己的表格类
        public MTable(Vector rowData, Vector columnNames) {
            super(rowData, columnNames);
        }
        @Override
        public JTableHeader getTableHeader() {               //定义表格头
            JTableHeader tableHeader = super.getTableHeader();                //获得表格头对象
            tableHeader.setReorderingAllowed(false);         //设置表格列不可重排
            DefaultTableCellRenderer hr = (DefaultTableCellRenderer) tableHeader
                    .getDefaultRenderer();                   //获得表格头的单元格对象
```

```
            hr.setHorizontalAlignment(DefaultTableCellRenderer.CENTER);        //设置列名居中显示
            return tableHeader;
        }
        @Override
        public TableCellRenderer getDefaultRenderer(Class<?> columnClass) {      //定义单元格
            DefaultTableCellRenderer cr = (DefaultTableCellRenderer) super
                    .getDefaultRenderer(columnClass);      //获得表格的单元格对象
            cr.setHorizontalAlignment(DefaultTableCellRenderer.CENTER); //设置单元格内容居中显示
            return cr;
        }
        @Override
        public boolean isCellEditable(int row, int column) {      //表格不可编辑
            return false;
        }
    }
}
```

运行结果如图 18.18 所示。

选中表格的第 2 行，选中行的背景色为黄色，文字颜色为红色，并且所有单元格的内容均居中显示。

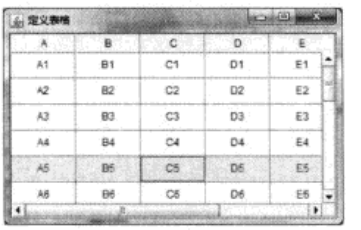


图 18.18　定义表格

18.1.3　操纵表格

在编写应用表格的程序时，经常需要获得表格的一些信息，如表格拥有的行数和列数。下面是 JTable 类中 3 个经常用来获得表格信息的方法。

- ☑ getRowCount()：获得表格拥有的行数，返回值为 int 型。
- ☑ getColumnCount()：获得表格拥有的列数，返回值为 int 型。
- ☑ getColumnName(int column)：获得位于指定索引位置的列的名称，返回值为 String 型。

表 18.4 中列出了经常用来操纵表格选中行的方法，包括设置、查看、统计、获取和取消选中行的方法。

表 18.4　JTable 类中经常用来操纵表格选中行的方法

方　法	说　明
setRowSelectionInterval(int from, int to)	选中行索引从 from 到 to 的所有行（包括索引为 from 和 to 的行）
addRowSelectionInterval(int from, int to)	将行索引从 from 到 to 的所有行追加为表格的选中行
isRowSelected(int row)	查看行索引为 row 的行是否被选中
selectAll()	选中表格中的所有行
clearSelection()	取消所有选中行的选择状态
getSelectedRowCount()	获得表格中被选中行的数量，返回值为 int 型，如果没有被选中的行则返回-1
getSelectedRow()	获得被选中行中最小的行索引值，返回值为 int 型，如果没有被选中的行则返回-1
getSelectedRows()	获得所有被选中行的索引值，返回值为 int 型数组

注意 由 JTable 类实现的表格的行索引和列索引均从 0 开始，即第 1 行的索引为 0，第 2 行的索引为 1 依此类推。

在 JTable 类中还提供了一个用来移动表格列位置的方法 moveColumn(int column, int targetColumn)，其中 column 为欲移动列的索引值，targetColumn 为目的列的索引值。移动表格列的具体执行方式如图 18.19 所示。

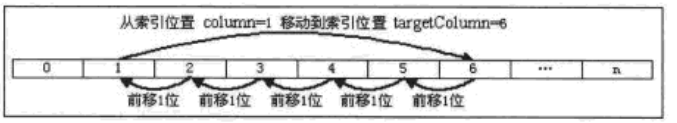


图 18.19 移动表格列的具体执行方式

【例 18.4】 操纵表格，使用本节讲到的所有方法的功能。（实例位置：光盘\TM\sl\18\4）

```
table = new JTable(tableValueV, columnNameV);
table.setRowSelectionInterval(1, 3);                    //设置选中行
table.addRowSelectionInterval(5, 5);                    //添加选中行
scrollPane.setViewportView(table);
JPanel buttonPanel = new JPanel();
getContentPane().add(buttonPanel, BorderLayout.SOUTH);
JButton selectAllButton = new JButton("全部选择");
selectAllButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        table.selectAll();                              //选中所有行
    }
});
buttonPanel.add(selectAllButton);
JButton clearSelectionButton = new JButton("取消选择");
clearSelectionButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        table.clearSelection();                         //取消所有选中行的选择状态
    }
});
buttonPanel.add(clearSelectionButton);
System.out.println("表格共有" + table.getRowCount() + "行" + table.getColumnCount() + "列");
System.out.println("共有" + table.getSelectedRowCount() + "行被选中");
System.out.println("第 3 行的选择状态为：" + table.isRowSelected(2));
System.out.println("第 5 行的选择状态为：" + table.isRowSelected(4));
System.out.println("被选中的第一行的索引是：" + table.getSelectedRow());
int[] selectedRows = table.getSelectedRows();           //获得所有被选中行的索引
System.out.print("所有被选中行的索引是：");
for (int row = 0; row < selectedRows.length; row++) {
    System.out.print(selectedRows[row] + "  ");
}
System.out.println();
System.out.println("列移动前第 2 列的名称是：" + table.getColumnName(1));
System.out.println("列移动前第 2 行第 2 列的值是：" + table.getValueAt(1, 1));
table.moveColumn(1, 5);                                 //将位于索引 1 的列移动到索引 5 处
```

```
System.out.println("列移动后第 2 列的名称是：" + table.getColumnName(1));
System.out.println("列移动后第 2 行第 2 列的值是：" + table.getValueAt(1, 1));
```

运行结果如图 18.20 所示。

其中表格的第 2、3、4 和 6 行被选中，并且列名为 B 的列从索引 1 处移动到了索引 5 处。单击“全部选择”按钮将选中表格的所有行，单击“取消选择”按钮将取消所有选中行的选择状态。运行结果如图 18.21 所示。

图 18.20　被选中指定行的表格

图 18.21　输出到控制台的信息

18.1.4　范例 1：列表元素与提示信息

有时出于节约空间等原因，列表项并不能很好地表明其含义，此时如果为列表项增加提示信息会很有用。然而默认的列表项提示信息方法有些复杂，本范例将自定义一个能显示提示信息的列表。运行结果如图 18.22 所示。（**实例位置：光盘\TM\sl\18\5**）

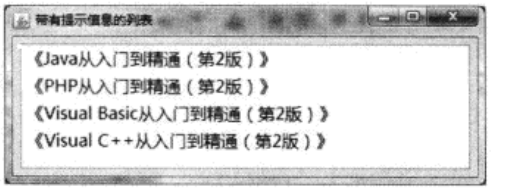


图 18.22　实例运行结果

（1）编写 ToolTipList 类，该类继承自 JList。该类定义了一个以二维数组为参数的构造方法并重写了 getToolTipText()方法。在重写的方法中，为不同的列表项指定了不同的提示信息。代码如下：

```
public class ToolTipList extends JList {
    private static final long serialVersionUID = -5334116242803068391L;//随机的序列化标识
    private Object[][] data;                                          //定义一个二维数组保存传递的参数
    public ToolTipList(Object[][] data) {
        this.data = data;
        Object[] listData = new Object[data.length];                  //定义一个一维数组保存列表项
        for (int i = 0; i < listData.length; i++) {
            listData[i] = data[i][0];                                 //获得列表项
        }
        setListData(listData);                                        //设置列表项
    }
    @Override
    public String getToolTipText(MouseEvent event) {
        int index = locationToIndex(event.getPoint());                //获得鼠标所在位置的列表项的索引
        if (index > -1) {
```

```
            //返回提示信息
            return "<html><font face=微软雅黑 size=16 color=red>" + data[index][1] + "</font></html>";
        } else {
            return super.getToolTipText(event);
        }
    }
}
```

（2）编写 do_this_windowActivated()方法，用来监听窗体激活事件。在该方法中，初始化了一个自定义列表对象并将其添加到滚动面板中。代码如下：

```
protected void do_this_windowActivated(WindowEvent e) {
    String[][] data = new String[4][2];                          //定义一个 4 行 2 列的二维数组
    data[0][0] = "《Java 从入门到精通（第 2 版）》";                //初始化数据
    data[0][1] = "清华大学出版社";                                  //初始化数据
    data[1][0] = "《PHP 从入门到精通（第 2 版）》";                 //初始化数据
    data[1][1] = "清华大学出版社";                                  //初始化数据
    data[2][0] = "《Visual Basic 从入门到精通（第 2 版）》";        //初始化数据
    data[2][1] = "清华大学出版社";                                  //初始化数据
    data[3][0] = "《Visual C++从入门到精通（第 2 版）》";           //初始化数据
    data[3][1] = "清华大学出版社";                                  //初始化数据
    JList list = new ToolTipList(data);                          //创建自定义列表
    list.setFont(new Font("微软雅黑", Font.PLAIN, 16));            //设置列表项的字体
    scrollPane.setViewportView(list);                            //将列表添加到滚动面板中
}
```

18.1.5 范例 2：监听列表单击事件

对于基本的 Swing 控件，当用户使用时通常会触发动作事件。列表采用了另一种不同的事件机制：当用户单击列表项时，会触发列表选择事件，可以对该事件进行监听以对用户的选择进行响应。本范例演示了如何实现此类监听。运行结果如图 18.23 所示。（**实例位置：光盘\TM\sl\18\6**）

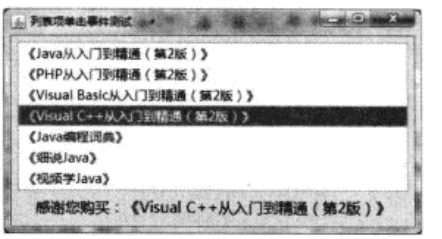


图 18.23 实例运行结果

（1）编写 do_this_windowActivated()方法，用来监听窗体激活事件。在该方法中，初始化了列表的数据。代码如下：

```
protected void do_this_windowActivated(WindowEvent e) {
    String[] listData = new String[7];
    listData[0] = "《Java 从入门到精通（第 2 版）》";                //初始化数据
    listData[1] = "《PHP 从入门到精通（第 2 版）》";                 //初始化数据
    listData[2] = "《Visual Basic 从入门到精通（第 2 版）》";        //初始化数据
    listData[3] = "《Visual C++从入门到精通（第 2 版）》";           //初始化数据
    listData[4] = "《Java 编程词典》";                               //初始化数据
    listData[5] = "《细说 Java》";                                   //初始化数据
    listData[6] = "《视频学 Java》";                                 //初始化数据
```

```
    list.setListData(listData);                                    //设置列表中的数据
}
```

（2）编写 do_list_valueChanged()方法，用来监听列表项选择事件。在该方法中，根据用户选择的列表项来更新标签的信息。代码如下：

```
protected void do_list_valueChanged(ListSelectionEvent e) {
    label.setText("感谢您购买：" + list.getSelectedValue());          //更新标签的信息
}
```

18.2　表格模型与表格

视频讲解：光盘\TM\lx\18\表格模型与表格.exe

用来创建表格的 JTable 类并不负责存储表格中的数据，而是由表格模型负责存储。当利用 JTable 类直接创建表格时，只是将数据封装到了默认的表格模型中。本节将学习表格模型的使用方法。

18.2.1　利用表格模型创建表格

接口 TableModel 定义了一个表格模型，抽象类 AbstractTableModel 实现了 TableModel 接口的大部分方法，只有如下 3 个抽象方法没有实现。

- ☑ public int getRowCount()
- ☑ public int getColumnCount()
- ☑ public Object getValueAt(int rowIndex, int columnIndex)

通过继承 AbstractTableModel 类实现上面 3 个抽象方法可以创建自己的表格模型类。DefaultTableModel 类是由 Swing 提供的继承了 AbstractTableModel 类并实现了上面 3 个抽象方法的表格模型类。DefaultTableModel 类提供的常用构造方法如表 18.5 所示。

表 18.5　DefaultTableModel 类提供的常用构造方法

构造方法	说　明
DefaultTableModel()	创建一个 0 行 0 列的表格模型
DefaultTableModel(int rowCount, int columnCount)	创建一个 rowCount 行 columnCount 列的表格模型
DefaultTableModel(Object[][] data, Object[] columnNames)	按照数组中指定的数据和列名创建一个表格模型
DefaultTableModel(Vector data, Vector columnNames)	按照向量中指定的数据和列名创建一个表格模型

表格模型创建完成后，通过 JTable 类的构造方法 JTable(TableModel dm)创建表格，就实现了利用表格模型创建表格。

从 JDK1.6 开始，提供了对表格进行排序的功能。通过 JTable 类的 setRowSorter(RowSorter<? extends TableModel> sorter)方法可以为表格设置排序器。TableRowSorter 类是由 Swing 提供的排序器类。为表格设置排序器的典型代码如下：

```
DefaultTableModel tableModel = new DefaultTableModel();          //创建表格模型
JTable table = new JTable(tableModel);                           //创建表格
table.setRowSorter(new TableRowSorter(tableModel));              //设置排序器
```

如果为表格设置了排序器，当单击表格的某一列头时，在该列名称的后面将出现▲标记，说明按该列升序排列表格中的所有行，如图 18.24 所示。

当再次单击该列头时，标记将变为▼形状，说明按该列降序排列表格中的所有行，如图 18.25 所示。

图 18.24　按升序排列

图 18.25　按降序排列

注意　在使用表格排序器时，通常要为其设置表格模型，否则将出现如图 18.26 所示的效果。一种方法是通过构造方法 TableRowSorter(TableModel model)创建排序器，另一种方法是通过 setModel (TableModel model)方法为排序器设置表格模型。

【例 18.5】　利用表格模型创建一个表格，并对表格使用表格排序器。（**实例位置：光盘\TM\sl\18\7**）

```
JScrollPane scrollPane = new JScrollPane();
getContentPane().add(scrollPane, BorderLayout.CENTER);
String[] columnNames = { "A", "B" };                                  //定义表格列名数组
String[][] tableValues = { { "A1", "B1" }, { "A2", "B2" }, { "A3", "B3" } };   //定义表格数据数组
//创建表格模型
DefaultTableModel tableModel = new DefaultTableModel(tableValues, columnNames);
JTable table = new JTable(tableModel);                                //创建指定表格模型的表格
table.setRowSorter(new TableRowSorter(tableModel));                   //设置表格的排序器
scrollPane.setViewportView(table);
```

运行结果如图 18.27 所示。

图 18.26　未设置表格模型

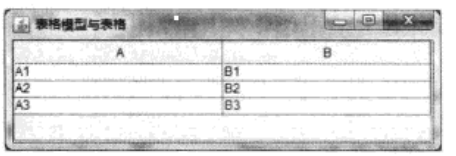


图 18.27　运行结果

单击名称为 B 列的列头后，运行结果如图 18.28 所示。

表格按 B 列升序排列；再次单击名称为 B 列的列头后，表格按 B 列降序排列，运行结果如图 18.29 所示。

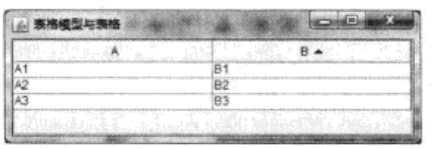


图 18.28　升序排列

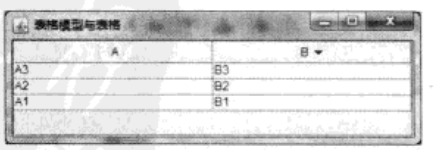


图 18.29　降序排列

18.2.2　维护表格模型

在使用表格时，经常需要对表格中的内容进行维护，例如向表格中添加新的数据行、修改表格中某一单元格的值、从表格中删除指定的数据行……这些操作均可以通过维护表格模型来完成。

在向表格模型中添加新的数据行时有两种情况：一种情况是添加到表格模型的尾部，另一种情况是插入到表格模型的指定索引位置。

（1）添加到表格模型的尾部，可以通过 addRow()方法完成。它的两个重载方法如下。

- ☑ addRow(Object[] rowData)：将由数组封装的数据添加到表格模型的尾部。
- ☑ addRow(Vector rowData)：将由向量封装的数据添加到表格模型的尾部。

（2）添加到表格模型的指定位置，可以通过 insertRow()方法完成。它的两个重载方法如下。

- ☑ insertRow(int row, Object[] rowData)：将由数组封装的数据添加到表格模型的指定索引位置。
- ☑ insertRow(int row, Vector rowData)：将由向量封装的数据添加到表格模型的指定索引位置。

如果需要修改表格模型中某一单元格的数据，可以通过方法 setValueAt(Object aValue, int row, int column)完成，其中 aValue 为单元格修改后的值，row 为单元格所在行的索引，column 为单元格所在列的索引；可以通过方法 getValueAt(int row, int column)获得指定单元格的值，该方法的返回值类型为 Object。

如果需要删除表格模型中某一行的数据，可以通过方法 removeRow(int row)来完成，其中 row 为欲删除行的索引。

注意　在删除表格模型中的数据时，每删除一行，其后所有行的索引值将相应地减 1，所以当连续删除多行时，需要注意对删除行索引的处理。

【例 18.6】　通过维护表格模型，实现了向表格中添加新的数据行、修改表格中某一单元格的值，以及从表格中删除指定的数据行。（**实例位置：光盘\TM\sl\18\8**）

```
public class ExampleFrame_06 extends JFrame {
    private DefaultTableModel tableModel;                    //定义表格模型对象
    private JTable table;                                    //定义表格对象
    private JTextField aTextField;
    private JTextField bTextField;
    public static void main(String args[]) {
        ExampleFrame_06 frame = new ExampleFrame_06();
        frame.setVisible(true);
    }
    public ExampleFrame_06() {
        super();
        setTitle("维护表格模型");
        setBounds(100, 100, 500, 375);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        final JScrollPane scrollPane = new JScrollPane();
        getContentPane().add(scrollPane, BorderLayout.CENTER);
```

```
String[] columnNames = { "A", "B" };                                    //定义表格列名数组
String[][] tableValues = { { "A1", "B1" }, { "A2", "B2" }, { "A3", "B3" } };   //定义表格数据数组
tableModel = new DefaultTableModel(tableValues, columnNames);           //创建表格模型
table = new JTable(tableModel);                                         //创建指定表格模型的表格
table.setRowSorter(new TableRowSorter(tableModel));                     //设置表格的排序器
table.setSelectionMode(ListSelectionModel.SINGLE_SELECTION); //设置表格的选择模式为单选
table.addMouseListener(new MouseAdapter() {                             //为表格添加鼠标事件监听器
    public void mouseClicked(MouseEvent e) {                            //发生了单击事件
        int selectedRow = table.getSelectedRow();                       //获得被选中行的索引
        Object oa = tableModel.getValueAt(selectedRow, 0);              //从表格模型中获得指定值
        Object ob = tableModel.getValueAt(selectedRow, 1);              //从表格模型中获得指定值
        aTextField.setText(oa.toString());                              //将值赋值给文本框
        bTextField.setText(ob.toString());                              //将值赋值给文本框
    }
});
scrollPane.setViewportView(table);
final JPanel panel = new JPanel();
getContentPane().add(panel, BorderLayout.SOUTH);
panel.add(new JLabel("A："));
aTextField = new JTextField("A4", 10);
panel.add(aTextField);
panel.add(new JLabel("B："));
bTextField = new JTextField("B4", 10);
panel.add(bTextField);
final JButton addButton = new JButton("添加");
addButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        //创建表格行数组
        String[] rowValues = { aTextField.getText(), bTextField.getText() };
        tableModel.addRow(rowValues);                                   //向表格模型中添加一行
        int rowCount = table.getRowCount() + 1;
        aTextField.setText("A" + rowCount);
        bTextField.setText("B" + rowCount);
    }
});
panel.add(addButton);
final JButton updButton = new JButton("修改");
updButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        int selectedRow = table.getSelectedRow();                       //获得被选中行的索引
        if (selectedRow != -1) {                                        //判断是否存在被选中行
            //修改表格模型中的指定值
            tableModel.setValueAt(aTextField.getText(), selectedRow, 0);
            tableModel.setValueAt(bTextField.getText(), selectedRow, 1);
        }
    }
});
panel.add(updButton);
final JButton delButton = new JButton("删除");
```

```
        delButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                int selectedRow = table.getSelectedRow();                    //获得被选中行的索引
                if (selectedRow != -1)                                       //判断是否存在被选中行
                    tableModel.removeRow(selectedRow);                       //从表格模型中删除指定行
            }
        });
        panel.add(delButton);
    }
}
```

运行结果如图 18.30 所示。

其中，A、B 文本框分别用来编辑 A、B 列的信息。单击“添加”按钮可以将编辑好的信息添加到表格中，选中表格的某一行后，在 A、B 文本框中将显示该行对应列的信息。重新编辑后单击“修改”按钮可以修改表格中的信息，单击“删除”按钮可以删除表格中被选中的行。

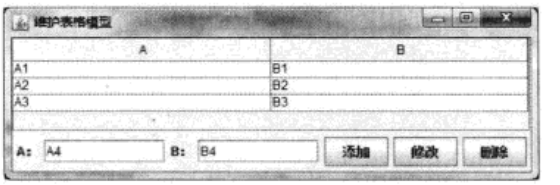


图 18.30　维护表格模型

18.2.3　范例 3：实现自动排序列表

为了用户选择列表项方便，通常会将列表项按某种顺序进行排序，如升序或降序。此时有两种实现方式：第一种是将排好顺序的数据添加到列表中；第二种是自定义能够排序的列表模型，每当向列表中增加元素时自动计算其位置。显然第二种更加灵活。本范例将自定义一个能排序的列表模型。运行结果如图 18.31 所示。（**实例位置：光盘\TM\sl\18\9**）

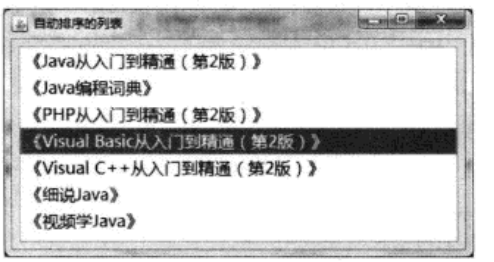


图 18.31　运行结果

（1）编写 SortedListModel 类，该类继承自 AbstractListModel。在该类中使用 TreeSet 来保存数据，因此不用实现排序细节。除了重写抽象方法，该类还增加了一个 add()方法，用来向 TreeSet 中增加数据。代码如下：

```
public class SortedListModel extends AbstractListModel {
    private static final long serialVersionUID = -8908769624938773296L;
    private TreeSet<Object> model = new TreeSet<Object>();
    @Override
    public Object getElementAt(int index) {                    //获得模型中指定索引的值
        return model.toArray()[index];
    }
    @Override
    public int getSize() {                                     //获得模型中元素的个数
        return model.size();
    }
    public void add(Object element) {                          //向 TreeSet 中增加元素
```

```
        if (model.add(element)) {
            fireContentsChanged(this, 0, getSize());
        }
    }
}
```

（2）编写 SortedListModelTest 类，该类继承自 JFrame。在该类的构造方法中，增加了一个列表，并设置其列表模型为支持排序的列表模型，代码如下：

```
public SortedListModelTest() {
    //省略框架属性设置和列表创建代码
    SortedListModel model = new SortedListModel();                    //创建可以排序的列表模型
    model.add("《Java 从入门到精通（第 2 版）》");                      //为列表模型增加元素
    model.add("《PHP 从入门到精通（第 2 版）》");                       //为列表模型增加元素
    model.add("《Visual Basic 从入门到精通（第 2 版）》");              //为列表模型增加元素
    model.add("《Visual C++从入门到精通（第 2 版）》");                 //为列表模型增加元素
    model.add("《Java 编程词典》");                                     //为列表模型增加元素
    model.add("《细说 Java》");                                         //为列表模型增加元素
    model.add("《视频学 Java》");                                       //为列表模型增加元素
    list.setModel(model);                                             //设置列表模型
}
```

18.2.4 范例 4：可以预览字体的列表

在 Word 2007 软件中，可以在选择字体时预览字体的样式，这对于用户来说无疑是非常友好的。本范例也将实现一个可以预览字体样式的列表。列表项的名字是字体名，列表项的样式就是该字体的样式。运行结果如图 18.32 所示。（**实例位置：光盘\TM\sl\18\10**）

图 18.32 运行结果

（1）编写 FontListCellRenderer 类，该类实现了 ListCellRenderer 接口。在该接口定义的方法中，实现了修改每个列表项字体的功能。代码如下：

```
public class FontListCellRenderer implements ListCellRenderer {
    @Override
    public Component getListCellRendererComponent(JList list, Object value, int index, boolean isSelected,
boolean cellHasFocus) {
        DefaultListCellRenderer defaultRenderer = new DefaultListCellRenderer();
        JLabel renderer = (JLabel) defaultRenderer.getListCellRendererComponent(list, value, index, isSelected,
cellHasFocus);                                          //获得 getListCellRendererComponent 的默认实现
        Font font = (Font) value;                       //获得列表项字体
        renderer.setFont(font);                         //设置标签的字体
        renderer.setText(font.getFontName());           //设置标签的文本
        return renderer;
    }
}
```

（2）编写 do_this_windowActivated()方法，用来监听窗体激活事件。在该方法中，为列表增加了

数据并且设置了新的渲染类，代码如下：

```
protected void do_this_windowActivated(WindowEvent e) {
    String[] fontNames = GraphicsEnvironment.getLocalGraphicsEnvironment().
getAvailableFontFamilyNames();                          //获得系统支持的全部字体
    DefaultListModel model = new DefaultListModel();    //创建表格模型
    for (String fontName : fontNames) {                 //遍历全部字体并将其添加到表格模型中
        model.addElement(new Font(fontName, Font.PLAIN, 24));
    }
    list.setModel(model);                               //设置表格模型
    ListCellRenderer renderer = new FontListCellRenderer();
    list.setCellRenderer(renderer);                     //设置列表中的渲染工具
}
```

18.3　提供行标题栏的表格

视频讲解：光盘\TM\lx\18\提供行标题栏的表格.exe

18.3.1　提供行标题栏的表格

通过 JTable 类创建的表格的列标题栏是永远可见的，即使是向下滚动了垂直滚动条，这就大大增强了表格的可读性。但是当不能显示出表格的所有列时，如果向右滚动水平滚动条则会导致表格左侧的部分列不可见，而通常情况下表格左侧的一列或几列为表格的基本数据，如图 18.33 所示。

如果通过移动滚动条查看未显示出的列数据，则会导致如图 18.34 所示的效果，即不知道每一行的具体销售日期，但是针对表格列则不会出现这样的问题。如果能够使表格左侧的一列或几列不随着水平滚动条滚动，也能够永远可见，就解决了上面的问题。

图 18.33　表格左侧的一列为表格的基本数据

图 18.34　移动滚动条查看未显示出的列数据

可以通过两个并列显示的表格实现这样的效果，其中左侧的表格用来显示永远可见的一列或几列，右侧的表格则用来显示其他的表格列。

18.3.2　范例 5：提供行标题栏的表格

本范例实现了一个提供行标题栏的表格，运行本范例后将得到如图 18.35 所示的窗体，在表格最左侧的“日期”列下方并没有滚动条，移动水平滚动条后将得到如图 18.36 所示的效果，表格最左侧的“日

期”列仍然可见。（**实例位置：光盘\TM\sl\18\11**）

图 18.35　提供行标题栏的表格

图 18.36　移动滚动条后的效果

（1）创建 MFixedColumnTable 类，该类继承了 JPanel 类，并声明 3 个属性。代码如下：

```
public class MFixedColumnTable extends JPanel {
    private Vector<String> columnNameV;                    //表格列名数组
    private Vector<Vector<Object>> tableValueV;            //表格数据数组
    private int fixedColumn = 1;                           //固定列数量
}
```

（2）创建用于左侧固定列表格的模型类 FixedColumnTableModel，该类继承了 AbstractTableModel 类，并且为 MFixedColumnTable 类的内部类。FixedColumnTableModel 类除了需要实现 Abstract TableModel 类的 3 个抽象方法外，还需要重构 getColumnName(int columnIndex)方法。代码如下：

```
private class FixedColumnTableModel extends AbstractTableModel {
    public int getColumnCount() {                          //返回固定列的数量
        return fixedColumn;
    }
    public int getRowCount() {                             //返回行数
        return tableValueV.size();
    }
    public Object getValueAt(int rowIndex, int columnIndex) {   //返回指定单元格的值
        return tableValueV.get(rowIndex).get(columnIndex);
    }
    public String getColumnName(int columnIndex) {         //返回指定列的名称
        return columnNameV.get(columnIndex);
    }
}
```

（3）创建用于右侧可移动列表格的模型类 FloatingColumnTableModel，该类继承了 AbstractTable Model 类，并且为 MFixedColumnTable 类的内部类。FixedColumnTableModel 类除了需要实现 Abstract TableModel 类的 3 个抽象方法外，还需要重构 getColumnName(int columnIndex)方法。代码如下：

```
private class FloatingColumnTableModel extends AbstractTableModel {
    public int getColumnCount() {                          //返回可移动列的数量
        return columnNameV.size() - fixedColumn;           //需要扣除固定列的数量
    }
    public int getRowCount() {                             //返回行数
```

```
        return tableValueV.size();
    }
    public Object getValueAt(int rowIndex, int columnIndex) {          //返回指定单元格的值
        //为列索引加上固定列的数量
        return tableValueV.get(rowIndex).get(columnIndex + fixedColumn);
    }
    public String getColumnName(int columnIndex) {                     //返回指定列的名称
        return columnNameV.get(columnIndex + fixedColumn);             //需要为列索引加上固定列的数量
    }
}
```

（4）在 MFixedColumnTable 类中再声明 4 个属性，代码如下：

```
private JTable fixedColumnTable;                                       //固定列表格对象
private FixedColumnTableModel fixedColumnTableModel;                   //固定列表格模型对象
private JTable floatingColumnTable;                                    //移动列表格对象
private FloatingColumnTableModel floatingColumnTableModel;             //移动列表格模型对象
```

（5）创建用于同步两个表格中被选中行的事件监听器类 MListSelectionListener，即当选中左侧固定列表格中的某一行时，监听器会同步选中右侧可移动列表格中的对应行；同样，当选中右侧可移动列表格中的某一行时，监听器会同步选中左侧固定列表格中的对应行。该类继承了 ListSelectionListener 类，并且为 MFixedColumnTable 类的内部类。代码如下：

```
private class MListSelectionListener implements ListSelectionListener {
    boolean isFixedColumnTable = true;                                 //默认由选中固定列表格中的行触发
    public MListSelectionListener(boolean isFixedColumnTable) {
        this.isFixedColumnTable = isFixedColumnTable;
    }
    public void valueChanged(ListSelectionEvent e) {
        if (isFixedColumnTable) {                                      //由选中固定列表格中的行触发
            int row = fixedColumnTable.getSelectedRow();               //获得固定列表格中的选中行
            //同时选中右侧可移动列表格中的相应行
            floatingColumnTable.setRowSelectionInterval(row, row);
        } else {                                                       //由选中可移动列表格中的行触发
            int row = floatingColumnTable.getSelectedRow();            //获得可移动列表格中的选中行
            //同时选中左侧固定列表格中的相应行
            fixedColumnTable.setRowSelectionInterval(row, row);
        }
    }
}
```

（6）编写 MFixedColumnTable 类的构造方法，需要传入 3 个参数，分别为表格列名数组、表格数据数组和固定列数量，之后就是创建固定列表格、可移动列表格和滚动面板。代码如下：

```
public MFixedColumnTable(Vector <string> columnNameV, Vector <Vector<object>> tableValueV, int fixedColumn) {
    super();
    setLayout(new BorderLayout());
    this.columnNameV = columnNameV;                                    //表格列名数组
```

```
    this.tableValueV = tableValueV;                                    //表格数据数组
    this.fixedColumn = fixedColumn;                                    //固定列数量
    //创建固定列表格
    fixedColumnTableModel = new FixedColumnTableModel();               //创建固定列表格模型对象
    fixedColumnTable = new JTable(fixedColumnTableModel);              //创建固定列表格对象
    ListSelectionModel fixed = fixedColumnTable.getSelectionModel();   //获得选择模型对象
    fixed.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);       //选择模式为单选
    fixed.addListSelectionListener(new MListSelectionListener(true));  //添加行被选中的事件监听器
    //创建可移动列表格
    floatingColumnTableModel = new FloatingColumnTableModel();         //创建可移动列表格模型对象
    floatingColumnTable = new JTable(floatingColumnTableModel);        //创建可移动列表格对象
    floatingColumnTable.setAutoResizeMode(JTable.AUTO_RESIZE_OFF);     //关闭表格的自动调整功能
    ListSelectionModel floating = floatingColumnTable.getSelectionModel();  //获得选择模型对象
    floating.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);    //选择模式为单选
    floating.addListSelectionListener(new MListSelectionListener(false));  //添加行被选中的事件监听器
    //创建滚动面板
    JScrollPane scrollPane = new JScrollPane();                        //创建一个滚动面板对象
    scrollPane.setCorner(JScrollPane.UPPER_LEFT_CORNER, fixedColumnTable
            .getTableHeader());                                        //将固定列表格头放到滚动面板的左上方
    JViewport viewport = new JViewport();                              //创建一个用来显示基础信息的视口对象
    viewport.setView(fixedColumnTable);                                //将固定列表格添加到视口中
    viewport.setPreferredSize(fixedColumnTable.getPreferredSize());    //设置视口的首选大小
    scrollPane.setRowHeaderView(viewport);                             //将视口添加到滚动面板的标题视口中
    scrollPane.setViewportView(floatingColumnTable);                   //将可移动列表格添加到默认视口
    add(scrollPane, BorderLayout.CENTER);
}
```

（7）创建 ExampleFrame_07 类，编写测试提供行标题栏表格的代码，首先封装表格列名数组和表格数据数组，然后创建 MFixedColumnTable 类的对象，最后将其添加到窗体中。代码如下：

```
Vector columnNameV = new Vector();
columnNameV.add("日期");
for (int i = 1; i < 21; i++) {
    columnNameV.add("商品" + i);
}
Vector tableValueV = new Vector();
for (int row = 1; row < 31; row++) {
    Vector rowV = new Vector();
    rowV.add(row);
    for (int col = 0; col < 20; col++) {
        rowV.add((int) (Math.random() * 1000));
    }
    tableValueV.add(rowV);
}
final MFixedColumnTable panel = new MFixedColumnTable(columnNameV, tableValueV, 1);
getContentPane().add(panel, BorderLayout.CENTER);
```

18.4　Swing 树组件

视频讲解：光盘\TM\lx\18\Swing 树组件.exe

18.4.1　简单的树

树状结构是一种常用的信息表现形式，它可以直观地显示出一组信息的层次结构。Swing 中的 JTree 类用来创建树，该类的常用构造方法如表 18.6 所示。

表 18.6　JTree 类的常用构造方法

构造方法	说　明
JTree()	创建一个默认的树
JTree(TreeNode root)	根据指定根节点创建树
JTree(TreeModel newModel)	根据指定树模型创建树

DefaultMutableTreeNode 类实现了 TreeNode 接口，用来创建树的节点。一个树只能有一个父节点，可以有 0 个或多个子节点，默认情况下每个节点都允许有子节点，如果需要可以设置为不允许。该类的常用构造方法如表 18.7 所示。

表 18.7　DefaultMutableTreeNode 类的常用构造方法

构造方法	说　明
DefaultMutableTreeNode()	创建一个默认的节点，默认情况下允许有子节点
DefaultMutableTreeNode(Object userObject)	创建一个具有指定标签的节点
DefaultMutableTreeNode(Object userObject, boolean allowsChildren)	创建一个具有指定标签的节点，并且指定是否允许有子节点

利用 DefaultMutableTreeNode 类的 add(MutableTreeNode newChild)方法可以为该节点添加子节点，该节点则称为父节点，没有父节点的节点则称为根节点。可以通过根节点利用构造方法 JTree(TreeNode root)直接创建树，也可以先创建一个树模型 TreeModel，然后再通过树模型利用构造方法 JTree(TreeModel newModel)创建树。

DefaultTreeModel 类实现了 TreeModel 接口，该类仅提供了如下两个构造方法，所以在利用该类创建树模型时，必须指定树的根节点。

- ☑ DefaultTreeModel(TreeNode root)：创建一个采用默认方式判断节点是否为叶子节点的树模型。
- ☑ DefaultTreeModel(TreeNode root, boolean asksAllowsChildren)：创建一个采用指定方式判断节点是否为叶子节点的树模型。

由 DefaultTreeModel 类实现的树模型判断节点是否为叶子节点有两种方式：默认方式为如果节点不存在子节点则为叶子节点，如图 18.37 所示；另一种方式则是根据节点是否允许有子节点，只要不允

许有子节点就是叶子节点，如果允许有子节点，即使并不包含任何子节点，也不是叶子节点，将入口参数 asksAllowsChildren 设置为 true 表示采用该种方式，如图 18.38 所示。

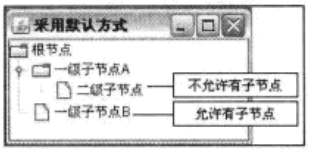


图 18.37　采用默认方式

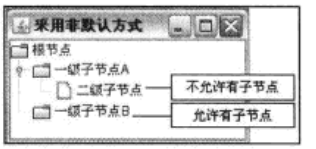


图 18.38　采用非默认方式

【例 18.7】　简单的树。（**实例位置：光盘\TM\sl\18\12**）

本例利用一个根节点创建了 3 个树，从左到右依次添加到窗体中，其中的一个是利用构造方法 JTree(TreeNode root)直接创建的；其他两个是通过树模型创建的，分别采用默认和非默认的方式判断节点是否为叶子节点。代码如下：

```
DefaultMutableTreeNode root = new DefaultMutableTreeNode("根节点");                    //创建根节点
DefaultMutableTreeNode nodeFirst = new DefaultMutableTreeNode("一级子节点 A");          //创建一级节点
root.add(nodeFirst);                                                    //将一级节点添加到根节点
//创建不允许有子节点的二级节点
DefaultMutableTreeNode nodeSecond = new DefaultMutableTreeNode("二级子节点", false);
nodeFirst.add(nodeSecond);                                              //将二级节点添加到一级节点
root.add(new DefaultMutableTreeNode("一级子节点 B"));                    //创建一级节点
JTree treeRoot = new JTree(root);                                       //利用根节点直接创建树
getContentPane().add(treeRoot, BorderLayout.WEST);
//利用根节点创建树模型，采用默认的判断方式
DefaultTreeModel treeModelDefault = new DefaultTreeModel(root);
JTree treeDefault = new JTree(treeModelDefault);                        //利用树模型创建树
getContentPane().add(treeDefault, BorderLayout.CENTER);
//利用根节点创建树模型，并采用非默认的判断方式
DefaultTreeModel treeModelPointed = new DefaultTreeModel(root, true);
JTree treePointed = new JTree(treeModelPointed);                        //利用树模型创建树
getContentPane().add(treePointed, BorderLayout.EAST);
```

运行结果如图 18.39 所示。

窗体左侧的树是直接创建的，中间的树是采用默认方式判断节点的，这两个树中名称为“一级子节点 B”的节点图标均为叶子节点图标；右侧的树是采用非默认方式判断节点的，该树中名称为“一级子节点 B”的节点图标均为非叶子节点图标。

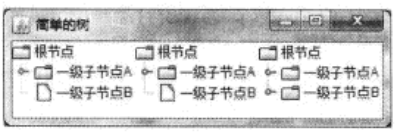


图 18.39　简单的树

18.4.2　处理选中节点事件

树的节点允许被选中和取消选中状态，通过捕获树节点的选择事件，可以处理相应的操作。树的选择模式有 3 种，通过 TreeSelectionModel 类的对象可以设置树的选择模式。可以通过 JTree 类的 getSelectionModel()方法获得 TreeSelectionModel 类的对象，然后通过 TreeSelectionModel 类的 setSelectionMode(int mode)方法设置选择模式。该方法的入口参数可以从表 18.8 中列出的该类的静态常

量中选择。

表 18.8　TreeSelectionModel 类中代表选择模式的静态常量

静 态 常 量	常 量 值	说　　明
SINGLE_TREE_SELECTION	1	只允许选中一个，如图 18.40 所示
CONTIGUOUS_TREE_SELECTION	2	允许连续选中多个，如图 18.41 所示
DISCONTIGUOUS_TREE_SELECTION	4	允许任意选中多个，如图 18.42 所示，为树的默认模式

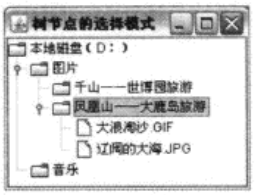


图 18.40　单选模式

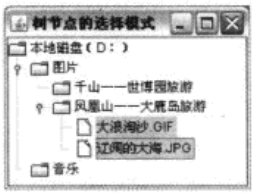


图 18.41　连选模式

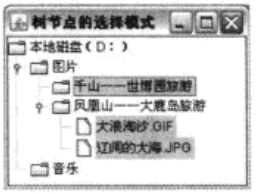


图 18.42　多选模式

当选中树节点和取消树节点的选中状态时，将发出 TreeSelectionEvent 事件，通过实现 TreeSelectionListener 接口可以捕获该事件，定义代码如下：

```
public interface TreeSelectionListener extends EventListener {
    void valueChanged(TreeSelectionEvent e);
}
```

当捕获发出的 TreeSelectionEvent 事件时，valueChanged(TreeSelectionEvent e)方法将被触发执行，此时通过 JTree 类的 getSelectionPaths()方法可以获得所有被选中节点的路径，该方法将返回一个 TreePath 类型的数组；通过 getSelectionPaths()方法将获得选中节点中索引值最小的节点的路径，即 TreePath 类的对象，也可以理解为选中节点中距离根节点最近的节点的路径。在获得选中节点的路径之前，可以通过 JTree 类的 isSelectionEmpty()方法查看是否存在被选中的节点，如果返回 false 表示存在被选中的节点，通过 getSelectionCount()方法可以获得被选中节点的数量。

TreePath 类表示树节点的路径，即通过该类可以获得子节点所属的父节点，以及父节点所属的上级节点，直到树的根节点。TreePath 类的常用方法如表 18.9 所示。

表 18.9　TreePath 类的常用方法

方　　法	说　　明
getPath()	以 Object 数组的形式返回该路径中所有节点的对象，在数组中的顺序按照从根节点到子节点的顺序
getLastPathComponent()	获得路径中最后一个节点的对象
getParentPath()	获得路径中除了最后一个节点的路径
pathByAddingChild(Object child)	获得向路径中添加指定节点后的路径
getPathCount()	获得路径中包含节点的数量
getPathComponent(int element)	获得路径中指定索引位置的节点对象

【例 18.8】　利用 TreeSelectionListener 监听器捕获选中树节点和取消选中树节点的事件，并将选

中节点的路径信息全部输出到控制台。（**实例位置：光盘\TM\sl\18\13**）

```
TreeSelectionModel treeSelectionModel = tree.getSelectionModel();          //获得树的选择模型
treeSelectionModel.setSelectionMode(
        TreeSelectionModel.CONTIGUOUS_TREE_SELECTION);                     //设置树的选择模式为连选
tree.addTreeSelectionListener(new TreeSelectionListener() {
    public void valueChanged(TreeSelectionEvent e) {
        if (!tree.isSelectionEmpty()) {                                    //查看是否存在被选中的节点
            TreePath[] selectionPaths = tree.getSelectionPaths();          //获得所有被选中节点的路径
            for (int i = 0; i < selectionPaths.length; i++) {
                TreePath treePath = selectionPaths[i];                     //获得被选中节点的路径
                //以 Object 数组的形式返回该路径中所有节点的对象
                Object[] path = treePath.getPath();
                for (int j = 0; j < path.length; j++) {
                    //获得节点
                    DefaultMutableTreeNode node = (DefaultMutableTreeNode) path[j];
                    System.out.print(node.getUserObject() + (j == (path.length - 1) ? "" : "-->"));
                }
                System.out.println();
            }
            System.out.println();
        }
    }
});
```

运行结果如图 18.43 所示。

首先展开树的所有节点，然后选中“千山——世博园旅游”节点，最后追加选中“凤凰山——大鹿岛旅游”节点，输出信息如图 18.44 所示。

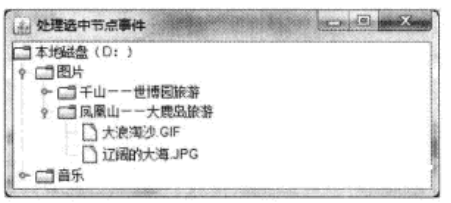


图 18.43　处理选中节点事件

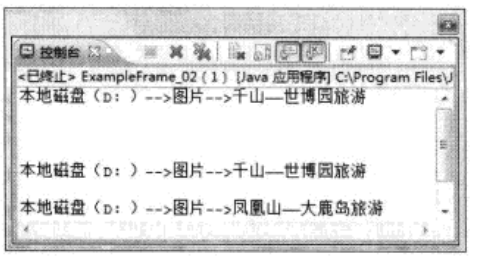


图 18.44　输出到控制台的信息

18.4.3　遍历树节点

有时需要对树进行遍历，也就是遍历树中的部分或全部节点，以便查找某一节点，或者是对树中的节点执行某一操作。DefaultMutableTreeNode 类提供了两组相对的遍历方式，下面进行详细介绍。

按前序遍历和按后序遍历是一组相对的遍历方式，按前序遍历树节点的顺序如图 18.45 所示，通过 preorderEnumeration()方法将返回按前序遍历的枚举对象；按后序遍历树节点的顺序如图 18.46 所示，通过 postorderEnumeration()方法将返回按后序遍历的枚举对象。

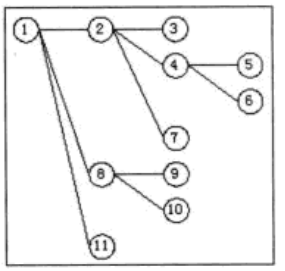


图 18.45　按前序遍历

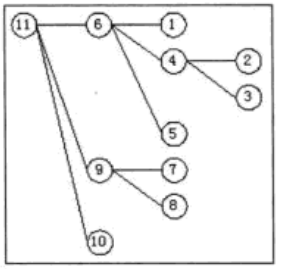


图 18.46　按后序遍历

以广度优先遍历和以深度优先遍历是一组相对的遍历方式。以广度优先遍历树节点的顺序如图 18.47 所示，通过 breadthFirstEnumeration()方法将返回以广度优先遍历的枚举对象；以深度优先遍历树节点的顺序如图 18.48 所示，通过 depthFirstEnumeration()方法将返回以深度优先遍历的枚举对象。

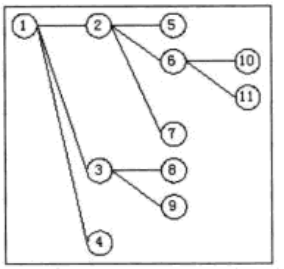


图 18.47　以广度优先遍历

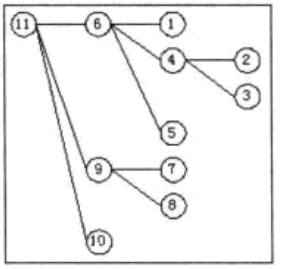


图 18.48　以深度优先遍历

说明　因为按后序遍历和以深度优先遍历这两种遍历方式的具体遍历方法相同，所以图 18.46 和图 18.48 是相同的，实际上 depthFirstEnumeration()方法只是调用了 postorderEnumeration()方法。

通过 DefaultMutableTreeNode 类的 children()方法，可以得到仅包含该节点的子节点的枚举对象，以便快速遍历节点的子节点。在 DefaultMutableTreeNode 类中还提供了一些常用方法，如表 18.10 所示。

表 18.10　DefaultMutableTreeNode 类的常用方法

方　法	说　明
getLevel()	获得该节点相对于根节点的级别值，如根节点的子节点的级别值为 1
getDepth()	获得以此节点为根节点的树的深度，如果该节点没有子节点则深度为 0
getParent()	获得该节点的父节点对象
getChildCount()	获得该节点拥有子节点的个数
getFirstChild()	获得该节点的第一个子节点对象
getSiblingCount()	获得该节点拥有兄弟节点的个数
getNextSibling()	获得该节点的后一个兄弟节点
getPreviousSibling()	获得该节点的前一个兄弟节点
getPath()	获得该节点的路径
isRoot()	判断该节点是否为根节点
isLeaf()	判断该节点是否为叶子节点

【例 18.9】 以按钮的方式提供了本节讲解的 5 种遍历方式，通过单击相应的按钮可以在控制台查看具体的遍历方式。（**实例位置：光盘\TM\sl\18\14**）

```
Enumeration enumeration;                                    //声明节点枚举对象
if (mode.equals("按前序遍历"))
    enumeration = root.preorderEnumeration();               //按前序遍历所有树节点
else if (mode.equals("按后序遍历"))
    enumeration = root.postorderEnumeration();              //按后序遍历所有树节点
else if (mode.equals("以广度优先遍历"))
    enumeration = root.breadthFirstEnumeration();           //以广度优先遍历所有树节点
else if (mode.equals("以深度优先遍历"))
    enumeration = root.depthFirstEnumeration();             //以深度优先遍历所有树节点
else
    enumeration = root.children();                          //遍历该节点的子节点
while (enumeration.hasMoreElements()) {                     //遍历节点枚举对象
    //获得节点
    DefaultMutableTreeNode node = (DefaultMutableTreeNode) enumeration.nextElement();
    for (int i = 0; i < node.getLevel(); i++) {             //根据节点级别输出占位符
        System.out.print("----");
    }
    System.out.println(node.getUserObject());               //输出节点标签
}
```

运行结果如图 18.49 所示。

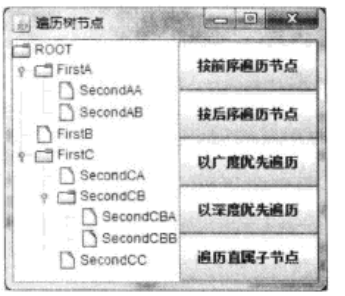


图 18.49 遍历树节点

18.4.4 定制树

在使用树时，经常需要对树进行一系列的设置，例如对节点图标的设置、对节点间连接线的设置，以及对树展开状况的设置等，以便满足实际需要的视觉效果。本小节将学习利用 JTree 类的一些方法定制树。

默认情况下显示树的根节点，但是不显示根节点手柄，如图18.50所示。有时并不希望显示树的根节点，此时可以调用 setRoot Visible(boolean rootVisible)方法，并将入口参数设为 false，效果如图 18.51 所示。如果希望在树的根节点前面也显示手柄，可以调用 setShowsRootHandles(boolean newValue)方法，并将入口参数设为 true，效果如图 18.52 所示。

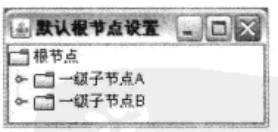


图 18.50 默认根节点设置

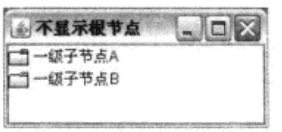


图 18.51 不显示根节点

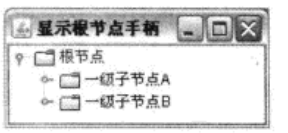


图 18.52 显示根节点手柄

默认情况下树节点的图标效果如图 18.53 所示，其中▭为非叶子节点的图标，▯为叶子节点的图标。通过 DefaultTreeCellRenderer 类的对象可以修改节点图标，通过 JTree 类的 getCellRenderer()方法可以

得到该对象。setLeafIcon(Icon newIcon)方法用来设置叶子节点图标，如图 18.54 所示为不采用叶子节点图标的效果；setClosedIcon(Icon newIcon)方法用来设置节点处于折叠状态时采用的图标，如图 18.55 所示为不采用节点折叠图标的效果；setOpenIcon(Icon newIcon)方法用来设置节点处于展开状态时采用的图标，如图 18.56 所示为不采用节点展开图标的效果。

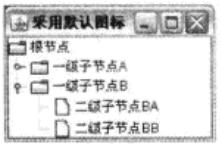


图 18.53　默认图标

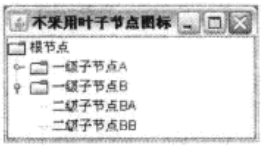


图 18.54　叶子节点图标

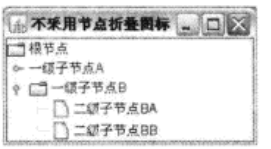


图 18.55　节点折叠图标

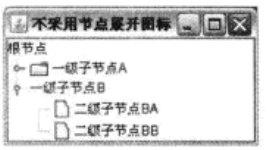


图 18.56　节点展开图标

默认情况下在树节点之间绘制连接线，效果如图 18.57 所示。通过 putClientProperty(Object key, Object value)方法可以设置连接线的绘制方式，此时需要将入口参数 key 设置为 JTree.lineStyle；将入口参数 value 设置为 None 表示不绘制节点间的连接线，效果如图 18.58 所示；设置为 Horizontal 表示绘制水平分栏线，绘制方式为仅在根节点和一级节点之间，或者是一级节点和一级节点之间，效果如图 18.59 所示；设置为 Angled 则表示绘制节点间的连接线，等效于默认设置。

图 18.57　默认节点连接线

图 18.58　取消节点连接线

图 18.59　绘制水平分栏线

默认情况下只有树的根节点是展开的，其他子节点均处于折叠状态，如果希望在初次运行时树的某一节点就处于展开状态，可以通过 expandPath(TreePath path)方法实现。在调用该方法时需要传入欲展开节点的路径。

【例 18.10】　利用树实现一个分层的导航栏，并且在初次运行时树的所有节点就处于展开状态。**（实例位置：光盘\TM\sl\18\15）**

```
tree.setRootVisible(false);                                    //不显示树的根节点
tree.setRowHeight(20);                                         //树节点的行高为 20 像素
tree.setFont(new Font("宋体", Font.BOLD, 14));                 //设置树节点的字体
tree.putClientProperty("JTree.lineStyle", "None");             //节点间不采用连接线
DefaultTreeCellRenderer treeCellRenderer = (DefaultTreeCellRenderer) tree
        .getCellRenderer();                                    //获得树节点的绘制对象
treeCellRenderer.setLeafIcon(null);                            //设置叶子节点不采用图标
treeCellRenderer.setClosedIcon(null);                          //设置节点折叠时不采用图标
treeCellRenderer.setOpenIcon(null);                            //设置节点展开时不采用图标
Enumeration enumeration = root.preorderEnumeration();          //按前序遍历所有树节点
while (enumeration.hasMoreElements()) {
    DefaultMutableTreeNode node = (DefaultMutableTreeNode) enumeration.nextElement();
    if (!node.isLeaf()) {                                      //判断是否为叶子节点
        TreePath path = new TreePath(node.getPath());          //创建该节点的路径
```

```
            tree.expandPath(path);                          //如果不是叶子节点则展开该节点
        }
    }
```

运行结果如图 18.60 所示。

图 18.60　定制树

18.4.5　维护树模型

在使用树时，有时需要提供对树的维护功能，包括向树中添加新节点，以及修改或删除树中的现有节点，这些操作需要通过树的模型类 DefaultTreeModel 来实现。下面就来介绍维护树模型的方法。

1．添加树节点

利用 DefaultTreeModel 类的 insertNodeInto()方法可以向树模型中添加新的节点。insertNodeInto()方法的具体定义如下：

```
insertNodeInto(MutableTreeNode newChild, MutableTreeNode parent, int index)
```

insertNodeInto()方法中各个入口参数的说明如表 18.11 所示。

表 18.11　insertNodeInto()方法中各个入口参数的说明

入口参数	说　明
newChild	新添加的节点对象
parent	新添加节点所属的父节点对象，该对象必须为树模型中的一个节点
index	新添加节点在其父节点中的索引位置，索引值从 0 开始

例如，假设要为节点 parentNode 添加一个子节点 treeNode，当前在父节点 parentNode（父节点 parentNode 为树模型 treeModel 中的一个节点）中已经存在 6 个子节点，现在要将该节点添加到所有子节点之后。代码如下：

```
treeModel.insertNodeInto(treeNode, parentNode, parentNode.getChildCount());
```

2．修改树节点

DefaultTreeModel 类的 nodeChanged(TreeNode node)方法用来通知树模型某节点已经被修改，否则如果修改的是节点的用户对象，修改信息将不会被同步到 GUI 界面。其中入口参数为被修改的节点对象。

例如，假设现在已经修改了树模型 treeModel 中的节点 treeNode（修改的是节点 treeNode 的用户对象），通知树模型 treeModel 其组成节点 treeNode 已经被修改的典型。代码如下：

```
treeModel.nodeChanged(treeNode);
```

3．删除树节点

DefaultTreeModel 类的 removeNodeFromParent(MutableTreeNode node)方法用来从树模型中删除指定节点 node。假设要从树模型 treeModel 中删除节点 treeNode，删除的典型代码如下：

```
treeModel.removeNodeFromParent(treeNode);
```

注意　树的根节点不允许删除，当试图删除根节点时，将抛出 java.lang.IllegalArgumentException: node does not have a parent.异常。

【例 18.11】　通过维护树模型，实现维护企业架构树。运行结果如图 18.25 所示。（**实例位置：光盘\TM\sl\18\16**）

```
final JButton addButton = new JButton("添加");
addButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        DefaultMutableTreeNode node = new DefaultMutableTreeNode(
                textField.getText());                                   //创建欲添加节点
        TreePath selectionPath = tree.getSelectionPath();               //获得选中的父节点路径
        DefaultMutableTreeNode parentNode = (DefaultMutableTreeNode) selectionPath
                .getLastPathComponent();                                //获得选中的父节点
        treeModel.insertNodeInto(node, parentNode, parentNode
                .getChildCount());                                      //插入节点到所有子节点之后
        TreePath path = selectionPath.pathByAddingChild(node);          //获得新添加节点的路径
        if (!tree.isVisible(path))
            tree.makeVisible(path);                                     //如果该节点不可见则令其可见
    }
});
panel.add(addButton);
final JButton updButton = new JButton("修改");
updButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        TreePath selectionPath = tree.getSelectionPath();               //获得选中的欲修改节点的路径
        DefaultMutableTreeNode node = (DefaultMutableTreeNode) selectionPath
                .getLastPathComponent();                                //获得选中的欲修改节点
        node.setUserObject(textField.getText());                        //修改节点的用户标签
        treeModel.nodeChanged(node);                                    //通知树模型该节点已经被修改
        tree.setSelectionPath(selectionPath);                           //选中被修改的节点
    }
});
panel.add(updButton);
final JButton delButton = new JButton("删除");
delButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        DefaultMutableTreeNode node = (DefaultMutableTreeNode) tree
                .getLastSelectedPathComponent();                        //获得选中的欲删除节点
        //查看欲删除的节点是否为根节点，根节点不允许删除
```

```
            if (!node.isRoot()) {
                DefaultMutableTreeNode nextSelectedNode = node
                        .getNextSibling();                          //获得下一个兄弟节点，以备选中
                if (nextSelectedNode == null)                       //查看是否存在兄弟节点
                    nextSelectedNode = (DefaultMutableTreeNode) node
                            .getParent();                           //如果不存在则选中其父节点
                treeModel.removeNodeFromParent(node);               //删除节点
                //选中节点
                tree.setSelectionPath(new TreePath(nextSelectedNode.getPath()));
            }
        }
    });
    panel.add(delButton);
```

运行结果如图 18.61 所示。

单击“添加”按钮后将向当前选中的节点中添加一个标签为“名称”文本框中内容的子节点；单击“修改”按钮后将把当前选中节点的标签修改为“名称”文本框中的内容；单击“删除”按钮后将删除当前选中的节点，根节点除外。

图 18.61　维护树模型

18.4.6　处理展开节点事件

有时需要捕获树节点被展开和折叠的事件，例如需要验证用户的权限，如果用户没有权限查看该节点包含的子节点，则不允许树节点展开。

当展开和折叠树节点时，将发出 TreeExpansionEvent 事件，通过实现 TreeWillExpandListener 接口，可以在树节点展开和折叠之前捕获该事件。TreeWillExpandListener 接口的具体定义如下：

```
public interface TreeWillExpandListener extends EventListener {
    public void treeWillExpand(TreeExpansionEvent event) throws ExpandVetoException;
    public void treeWillCollapse(TreeExpansionEvent event) throws ExpandVetoException;
}
```

如果此次事件是由将要展开节点发出的，treeWillExpand()方法将被触发；如果此次事件是由将要折叠节点发出的，treeWillCollapse()方法将被触发。

通过实现 TreeExpansionListener 接口，可以在树节点展开和折叠时捕获该事件。TreeExpansion Listener 接口的具体定义如下：

```
public interface TreeExpansionListener extends EventListener {
    public void treeExpanded(TreeExpansionEvent event);
    public void treeCollapsed(TreeExpansionEvent event);
}
```

如果此次事件是由展开节点发出的，treeExpanded()方法将被触发；如果此次事件是由将要折叠节点发出的，treeCollapsed()方法将被触发。

【例 18.12】　同时为树添加了 TreeWillExpandListener 和 TreeExpansionListener 监听器，目的是向控制台输出相应的提示信息，以展示这两个监听器的使用方法。（**实例位置：光盘\TM\sl\18\17**）

```
tree.addTreeWillExpandListener(new TreeWillExpandListener() {        //捕获树节点将要被展开或折叠的事件
    public void treeWillCollapse(TreeExpansionEvent e) {             //树节点将要被折叠时触发
        TreePath path = e.getPath();                                 //获得将要被折叠节点的路径
        DefaultMutableTreeNode node = (DefaultMutableTreeNode) path
                .getLastPathComponent();                             //获得将要被折叠的节点
        System.out.println("节点“" + node + "”将要被折叠！");
    }
    public void treeWillExpand(TreeExpansionEvent e) {               //树节点将要被展开时触发
        TreePath path = e.getPath();                                 //获得将要被展开节点的路径
        DefaultMutableTreeNode node = (DefaultMutableTreeNode) path
                .getLastPathComponent();                             //获得将要被展开的节点
        System.out.println("节点“" + node + "”将要被展开！");
    }
});
tree.addTreeExpansionListener(new TreeExpansionListener() {          //捕获树节点已经被展开或折叠的事件
    public void treeCollapsed(TreeExpansionEvent e) {                //树节点已经折叠时触发
        TreePath path = e.getPath();                                 //获得已经被折叠节点的路径
        DefaultMutableTreeNode node = (DefaultMutableTreeNode) path
                .getLastPathComponent();                             //获得已经被折叠的节点
        System.out.println("节点“" + node + "”已经被折叠！");
        System.out.println();
    }
    public void treeExpanded(TreeExpansionEvent e) {                 //树节点已经被展开时触发
        TreePath path = e.getPath();                                 //获得已经被展开节点的路径
        DefaultMutableTreeNode node = (DefaultMutableTreeNode) path
                .getLastPathComponent();                             //获得已经被展开的节点
        System.out.println("节点“" + node + "”已经被展开！");
        System.out.println();
    }
});
```

运行结果如图 18.62 所示。

首先展开“技术部”节点，然后展开“服务部”节点，最后再折叠“技术部”节点，输出信息如图 18.63 所示。

图 18.62　处理展开节点事件

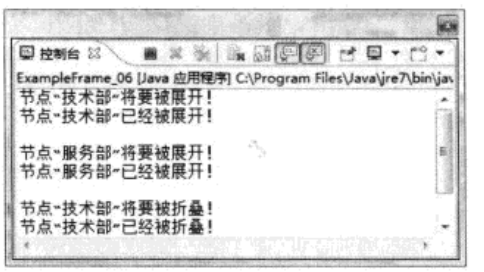


图 18.63　控制台的输出信息

18.4.7 范例 6：为树节点增加提示信息

在比较复杂的程序中，为了节约界面的空间，会将树控件的节点使用比较简略的文本说明信息。作为对节点的补充说明，可以为其增加提示信息。本范例将演示如何为节点增加自定义的提示信息。运行结果如图 18.64 所示。（**实例位置：光盘\TM\sl\18\18**）

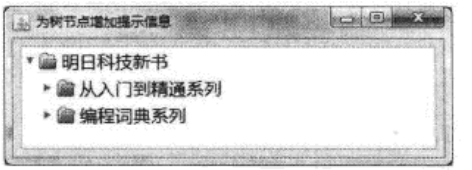


图 18.64 运行结果

（1）编写 ToolTipNode 类，该类实现了 TreeCellRenderer 接口。在构造方法中，使用 map 参数来初始化键值对。在 getTreeCellRendererComponent()方法中，为默认树节点渲染器设置了自定义的提示信息。代码如下：

```
public class ToolTipNode implements TreeCellRenderer {
    private static final long serialVersionUID = -1884123073630846839L;
    private DefaultTreeCellRenderer renderer = new DefaultTreeCellRenderer();
    private Map<DefaultMutableTreeNode, String> map;                    //保存键值对
    public ToolTipNode(Map<DefaultMutableTreeNode, String> map) {
        this.map = map;                                                 //初始化键值对
    }
    @Override
    public Component getTreeCellRendererComponent(JTree tree, Object value, boolean selected, boolean
expanded, boolean leaf, int row, boolean hasFocus) {
        //调用默认的 getTreeCellRendererComponent()方法
        renderer.getTreeCellRendererComponent(tree, value, selected, expanded, leaf, row, hasFocus);
        renderer.setToolTipText("<html><font face=微软雅黑 size=16 color=red>" + map.get(value) + "</font>
</html>");                                                              //设置提示信息
        return renderer;
    }
}
```

（2）编写 do_this_windowActivated()方法，用来监听窗体激活事件。在该方法中，为树增加了数据并使用了自定义的渲染器。代码如下：

```
protected void do_this_windowActivated(WindowEvent e) {
    DefaultMutableTreeNode root = new DefaultMutableTreeNode("明日科技新书");      //创建根节点
    DefaultMutableTreeNode parent1 = new DefaultMutableTreeNode("从入门到精通系列");
    parent1.add(new DefaultMutableTreeNode("《Java 从入门到精通（第 2 版）》"));
    parent1.add(new DefaultMutableTreeNode("《PHP 从入门到精通（第 2 版）》"));
    parent1.add(new DefaultMutableTreeNode("《Visual Basic 从入门到精通（第 2 版）》"));
    parent1.add(new DefaultMutableTreeNode("《Visual C++从入门到精通（第 2 版）》"));
    root.add(parent1);                                                  //增加子节点
    DefaultMutableTreeNode parent2 = new DefaultMutableTreeNode("编程词典系列");
    parent2.add(new DefaultMutableTreeNode("《Java 编程词典》"));
    parent2.add(new DefaultMutableTreeNode("《PHP 编程词典》"));
    parent2.add(new DefaultMutableTreeNode("《Visual Basic 编程词典》"));
    parent2.add(new DefaultMutableTreeNode("《Visual C++编程词典》"));
```

```
    root.add(parent2);                                                    //增加子节点
    DefaultTreeModel model = new DefaultTreeModel(root);                  //使用根节点创建默认树模型
    tree.setModel(model);                                                 //更新树模型
    ToolTipManager.sharedInstance().registerComponent(tree);              //为树注册提示信息管理器
    //利用映射保存提示信息
    Map<DefaultMutableTreeNode, String> map = new HashMap<DefaultMutableTreeNode, String>();
    map.put(root, "明日科技");                                            //增加提示信息
    map.put(parent1, "明日科技");                                         //增加提示信息
    map.put(parent2, "明日科技");                                         //增加提示信息
    tree.setCellRenderer(new ToolTipNode(map));                           //设置新的渲染器
}
```

18.4.8　范例 7：双击编辑树节点功能

有些时候，程序员定义的树节点并不能很好地满足用户的需要，此时用户会希望能够编辑现有的树节点。本范例通过设置节点编辑器来实现这个功能。运行结果如图 18.65 和图 18.66 所示。（**实例位置：光盘\TM\sl\18\19**）

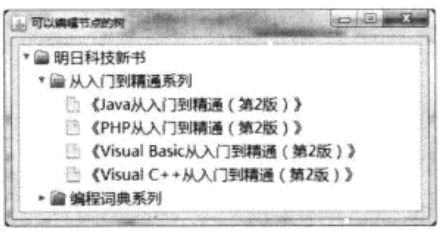


图 18.65　初始状态

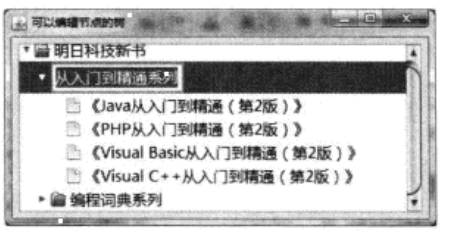


图 18.66　编辑状态

DefaultCellEditor 是表单元格和树单元格的默认编辑器。它实现了 TreeCellEditor 接口，因此可以修改树的节点。DefaultCellEditor 的构造方法支持将树的节点设置成使用复选框、组合框和文本域。本范例将其设置成了文本域，使用的构造方法声明如下：

```
public DefaultCellEditor(JTextField textField)
```

textField：一个 JTextField 对象。

编写 do_this_windowActivated()方法，用来监听窗体激活事件。在该方法中，初始化了树的各个节点并设置了节点编辑器。代码如下：

```
protected void do_this_windowActivated(WindowEvent e) {
    DefaultMutableTreeNode root = new DefaultMutableTreeNode("明日科技新书");      //创建根节点
    DefaultMutableTreeNode parent1 = new DefaultMutableTreeNode("从入门到精通系列");
    parent1.add(new DefaultMutableTreeNode("《Java 从入门到精通（第 2 版）》"));
    parent1.add(new DefaultMutableTreeNode("《PHP 从入门到精通（第 2 版）》"));
    parent1.add(new DefaultMutableTreeNode("《Visual Basic 从入门到精通（第 2 版）》"));
    parent1.add(new DefaultMutableTreeNode("《Visual C++从入门到精通（第 2 版）》"));
    root.add(parent1);                                                    //增加子节点
    DefaultMutableTreeNode parent2 = new DefaultMutableTreeNode("编程词典系列");
```

```
    parent2.add(new DefaultMutableTreeNode("《Java 编程词典》"));
    parent2.add(new DefaultMutableTreeNode("《PHP 编程词典》"));
    parent2.add(new DefaultMutableTreeNode("《Visual Basic 编程词典》"));
    parent2.add(new DefaultMutableTreeNode("《Visual C++编程词典》"));
    root.add(parent2);                                          //增加子节点
    DefaultTreeModel model = new DefaultTreeModel(root);        //创建树模型
    tree.setModel(model);                                       //设置树模型
    JTextField textField = new JTextField();                    //创建文本域对象
    textField.setFont(new Font("微软雅黑", Font.PLAIN, 16));    //为文本域设置字体
    TreeCellEditor editor = new DefaultCellEditor(textField);   //创建树编辑器
    tree.setEditable(true);                                     //设置树节点可编辑
    tree.setCellEditor(editor);                                 //使用树编辑器
}
```

18.5 经典范例

18.5.1 经典范例 1：编写中国省市信息树

视频讲解：光盘\TM\lx\18\编写中国省市信息树.exe

对于具有层次关系的结构，使用树控件描述是非常方便的，如文件夹及其子文件夹之间的关系、国家的行政结构关系等。本范例将使用树控件来表示中国的各个行政区域。使用 Swing 库中定义的工具类可以非常容易的实现。运行结果如图 18.67 所示。**（实例位置：光盘\TM\sl\18\20）**

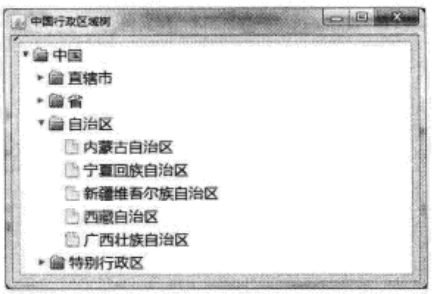


图 18.67 运行结果

（1）编写 ChinaGeographyTree 类，该类继承了 JFrame。在框架中包含了一颗树，在树中显示了中国的直辖市、省、自治区和特别行政区信息。

（2）编写 do_this_windowActivated()方法，用来监听窗体激活事件。在该方法中，为树控件增加节点信息。代码如下：

```
protected void do_this_windowActivated(WindowEvent e) {
    DefaultMutableTreeNode root = new DefaultMutableTreeNode("中国");     //创建根节点
    DefaultMutableTreeNode municipalities = new DefaultMutableTreeNode("直辖市");
    municipalities.add(new DefaultMutableTreeNode("北京"));          //为“直辖市”增加子节点“北京”
    municipalities.add(new DefaultMutableTreeNode("上海"));          //为“直辖市”增加子节点“上海”
    municipalities.add(new DefaultMutableTreeNode("天津"));          //为“直辖市”增加子节点“天津”
    municipalities.add(new DefaultMutableTreeNode("重庆"));          //为“直辖市”增加子节点“重庆”
    //省略其他节点的信息
    root.add(municipalities);                                        //为根节点增加“直辖市”节点
    root.add(province);                                              //为根节点增加“省”节点
    root.add(ARegion);                                               //为根节点增加“自治区”节点
```

```
    root.add(SARegion);                                          //为根节点增加“特别行政区”节点
    DefaultTreeModel model = new DefaultTreeModel(root);         //利用根节点创建树模型
    tree.setModel(model);                                        //为树设置新的树模型
}
```

18.5.2　经典范例 2：监听节点的选择事件

视频讲解：光盘\TM\lx\18\监听节点的选择事件.exe

通常情况下，树控件是与其他的控件一同使用的。当选择了树控件的一个节点后，会触发其他控件状态的改变事件。本范例将监听节点的选择事件，当用户选择不同的节点时，会在右侧的文本区中显示该节点的一些信息。运行结果如图 18.68 和图 18.69 所示。（**实例位置：光盘\TM\sl\18\21**）

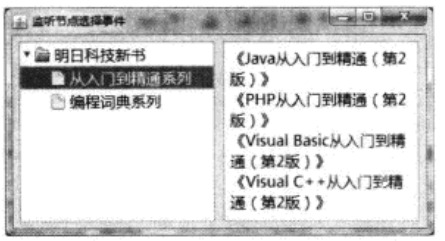


图 18.68　监听节点的选择事件（一）

图 18.69　监听节点的选择事件（二）

（1）编写 SelectedEventTest 类，该类继承了 JFrame。在框架中包含了一棵树和一个文本区，文本区用来响应用户选择树节点的事件。

（2）编写 do_this_windowActivated()方法，用来监听窗体激活事件。在该方法中，初始化了树中的节点信息。代码如下：

```
protected void do_this_windowActivated(WindowEvent e) {
    DefaultMutableTreeNode root = new DefaultMutableTreeNode("明日科技新书");          //创建根节点
    DefaultMutableTreeNode parent1 = new DefaultMutableTreeNode("从入门到精通系列");
    root.add(parent1);                                                                 //增加子节点
    DefaultMutableTreeNode parent2 = new DefaultMutableTreeNode("编程词典系列");
    root.add(parent2);                                                                 //增加子节点
    DefaultTreeModel model = new DefaultTreeModel(root);                 //使用根节点创建默认树模型
    tree.setModel(model);                                                              //更新树模型
}
```

（3）编写 do_tree_valueChanged()方法，用来监听用户选择不同的树节点事件。在该方法中，根据用户选择的节点更新了文本区内容。代码如下：

```
protected void do_tree_valueChanged(TreeSelectionEvent e) {
    TreePath path = tree.getSelectionPath();                    //获得用户选择的节点路径
    if(path==null) {                                            //如果没有选择节点则直接返回
        return;
    }
    DefaultMutableTreeNode node = (DefaultMutableTreeNode)path.getLastPathComponent();
```

```
    String text1 = "《Java 从入门到精通（第 2 版）》\n《PHP 从入门到精通（第 2 版）》\n《Visual Basic 从入门到精通（第 2 版）》\n《Visual C++从入门到精通（第 2 版）》";
    String text2 = "《Java 编程词典》\n《PHP 编程词典》\n《Visual Basic 编程词典》\n《Visual C++编程词典》";
    if (node.toString().equals("从入门到精通系列")) {                    //如果选择了“从入门到精通系列”节点
        textArea.setText(text1);                                         //将文本区设置成 text1
    } else {
        textArea.setText(text2);                                         //将文本区设置成 text2
    }
}
```

18.6 本章小结

通过本章的学习，相信读者已经可以熟练地使用 JTable 表格，包括通过各种方式创建表格、根据实际需要定制表格、通过编码操纵表格及维护表格模型，也已能熟练地使用 JTree 树，包括通过各种方式创建树、根据实际需要定制树、各种遍历树节点的方法、维护树模型的方法，以及处理选中节点事件和展开节点事件的方法。通过对 JTree 树的灵活使用，可以直观地显示出各种具有层次结构的信息。

18.7 实战练习

1. 利用 Swing 表格设计一个用来选择日期的对话框。（**答案位置：光盘\TM\sl\18\22**）
2. 设计一个以多列为行标题栏的例子。（**答案位置：光盘\TM\sl\18\23**）
3. 尝试开发一个支持树状结构的下拉菜单。（**答案位置：光盘\TM\sl\18\24**）

第19章

高级布局管理器

（视频讲解：39分钟）

Swing还提供了一些高级布局管理器，如箱式布局管理器、卡片布局管理器、网格组布局管理器以及弹簧布局管理器，通过对这些布局管理器的使用，可以设计出更好、更实用的程序界面。在讲解过程中为了便于读者理解，结合了大量的图例，还针对每个知识点进行了举例。

通过阅读本章，您可以：

- 学会箱式布局管理器的使用方法
- 学会卡片布局管理器的使用方法
- 学会网格组布局管理器的使用方法
- 学会弹簧布局管理器的使用方法
- 了解弹簧和支柱的使用方法
- 能够利用弹簧和支柱显示组件的大小：
- 学会利用各种布局管理器设计程序界面

PDG

19.1 箱式布局管理器

视频讲解：光盘\TM\lx\19\箱式布局管理器.exe

由 BoxLayout 类实现的布局管理器称为箱式（BoxLayout）布局管理器，用来管理一组水平或垂直排列的组件，如果是用来管理一组水平排列的组件，则称为水平箱，效果如图 19.1 所示。

如果是用来管理一组垂直排列的组件，则称为垂直箱，效果如图 19.2 所示。

图 19.1 水平箱

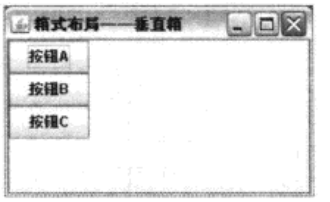


图 19.2 垂直箱

BoxLayout 类仅提供了一个构造方法 BoxLayout(Container target, int axis)，其入口参数 target 为要采用该布局方式的容器对象；入口参数 axis 为要采用的布局方式，如果将其设置为静态常量 X_AXIS，表示创建一个水平箱，组件将从左到右排列，设置为静态常量 Y_AXIS 则表示创建一个垂直箱，组件将从上到下排列。无论水平箱还是垂直箱，当将窗体调小至不能显示所有组件时，组件仍会排列在一行或一列，如图 19.3 所示，组件的排列顺序依据添加到容器中的先后顺序。

箱式布局在排列组件时，对于水平箱，会试图将所有组件的高度调整到其中最高组件的高度，如果某一组件不能满足这个要求，会根据该组件的垂直调整值（即 alignmentY 属性的值）进行调整。具体的调整方案如图 19.4 所示。

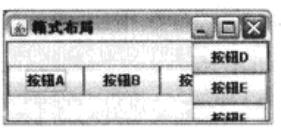


图 19.3 调小窗体效果

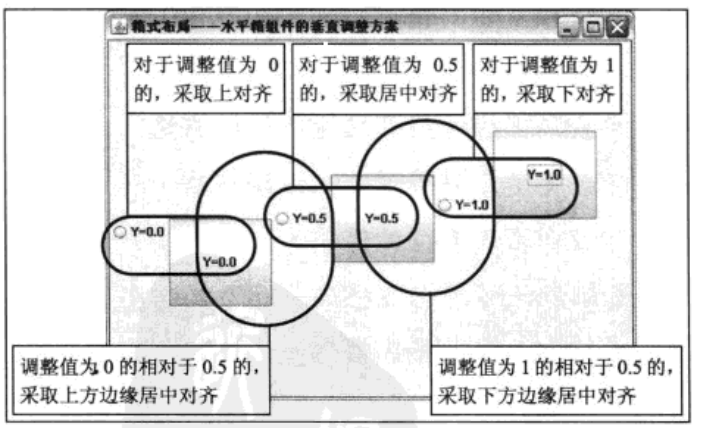


图 19.4 水平箱组件的垂直调整方案

对于垂直箱，会试图将所有组件的宽度调整到其中最宽组件的宽度，如果某一组件不能满足这个要求，会根据该组件的水平调整值（即 alignmentX 属性的值）进行调整。具体的调整方案如图 19.5 所示。

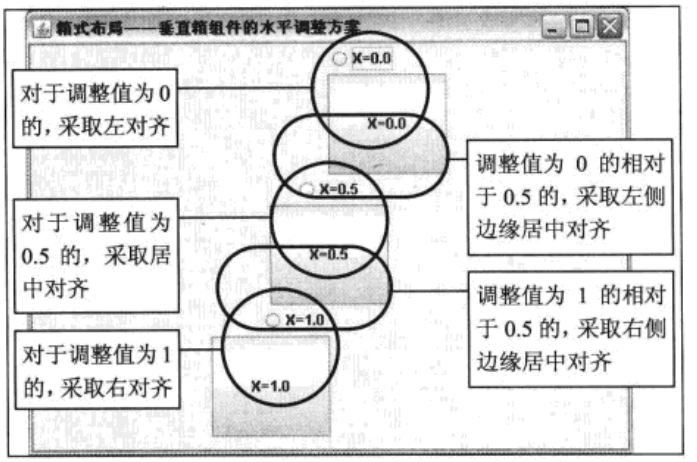


图 19.5　垂直箱组件的水平调整方案

默认情况下箱式布局管理器的组件之间没有间距，如果需要在组件之间设置间距，可以通过使用 Box 类提供的 6 个不可见组件实现，这些组件就是专门用来设置箱式布局的。

Box 类是以 BoxLayout 类为其布局管理器的轻量级容器，可以分别通过静态方法 createHorizontalBox()和 createVerticalBox()获得水平或垂直布局管理器的箱式容器。在该类中提供了两种类型的不可见的组件，分别为支柱（Strut）类型和胶水（Glue）类型，它们具体包含的组件如图 19.6 所示。

图 19.6　Box 类提供的不可见组件

支柱类型的组件具有指定的宽度、高度或大小，通常用来设置组件间的固定间距，例如使用水平支柱设置 3 个按钮间距后的效果如图 19.7 所示。

胶水类型的组件类似于一个弹簧，通常用来将组件平均分布到容器中，例如使用水平胶水设置 3 个按钮间距后的效果如图 19.8 所示。

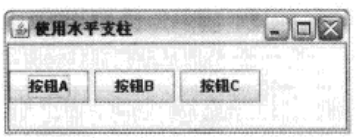


图 19.7　使用水平支柱设置效果

图 19.8　使用水平胶水设置效果

【例 19.1】　使用箱式布局管理器实现组件的右对齐和上对齐，以及控制组件之间的间距。（**实例位置：光盘\TM\sl\19\1**）

```
Box topicBox = Box.createHorizontalBox();                    //创建一个水平箱容器
getContentPane().add(topicBox, BorderLayout.NORTH);          //添加到窗体中
topicBox.add(Box.createHorizontalStrut(5));                  //添加一个 5 像素宽的水平支柱
JLabel topicLabel = new JLabel("主题：");                    //创建主题标签
```

```
topicBox.add(topicLabel);                          //添加到水平箱容器中
topicBox.add(Box.createHorizontalStrut(5));        //添加一个 5 像素宽的水平支柱
JTextField topicTextField = new JTextField(30);    //创建文本框
topicBox.add(topicTextField);                      //添加到水平箱容器中
Box box = Box.createVerticalBox();                 //创建一个垂直箱容器
getContentPane().add(box, BorderLayout.CENTER);    //添加到窗体中
box.add(Box.createVerticalStrut(5));               //添加一个 5 像素高的垂直支柱
Box contentBox = Box.createHorizontalBox();        //创建一个水平箱容器
contentBox.setAlignmentX(1);                       //设置组件的水平调整值，靠右侧对齐
box.add(contentBox);                               //添加到垂直箱容器中
contentBox.add(Box.createHorizontalStrut(5));      //添加一个 5 像素宽的水平支柱
JLabel contentLabel = new JLabel("内容：");        //创建一个标签组件
contentLabel.setAlignmentY(0);                     //设置组件的垂直调整值，靠上方对齐
contentBox.add(contentLabel);                      //添加到水平箱容器中
contentBox.add(Box.createHorizontalStrut(5));      //添加一个 5 像素宽的水平支柱
JScrollPane scrollPane = new JScrollPane();        //创建一个滚动面板
scrollPane.setAlignmentY(0);                       //设置组件的垂直调整值，靠上方对齐
contentBox.add(scrollPane);                        //添加到水平箱容器中
JTextArea contentTextArea = new JTextArea();       //创建一个文本区域
scrollPane.setViewportView(contentTextArea);       //添加到滚动面板中
box.add(Box.createVerticalStrut(5));               //添加一个 5 像素高的垂直支柱
JButton submitButton = new JButton("确定");        //创建一个按钮
submitButton.setAlignmentX(1);                     //设置组件的水平调整值，靠右侧对齐
box.add(submitButton);                             //添加到垂直箱容器中
```

运行结果如图 19.9 所示。

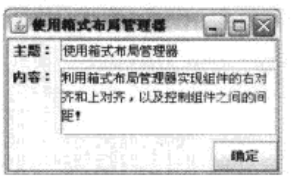


图 19.9　使用箱式布局管理器

窗体中的内容标签和内容文本区实现了上对齐，并且它们所在的水平箱容器和“确定”按钮实现了靠右侧对齐。

19.2　卡片布局管理器

视频讲解：光盘\TM\lx\19\卡片布局管理器.exe

由 CardLayout 类实现的布局管理器称为卡片（CardLayout）布局管理器，用来操纵其所管理容器中包含的容器或组件。每个直接添加到其所管理容器中的容器或组件为一个卡片，最先被添加的容器或组件被认为是第一个卡片，最后被添加的则为最后一个卡片，初次运行时将显示第一个卡片。

在 CardLayout 类中提供了 5 个用来显示卡片的方法，如表 19.1 所示，具体的使用方法如图 19.10 所示。

表 19.1　CardLayout 类中用来显示卡片的方法

方　　法	说　　明
first(Container parent)	显示第一个卡片
last(Container parent)	显示最后一个卡片
next(Container parent)	显示当前所显示卡片之后的卡片
previous(Container parent)	显示当前所显示卡片之前的卡片
show(Container parent, String name)	显示指定标签的卡片

说明　表 19.1 中的 5 个方法均需要传入一个 Container 型的对象，该对象为布局管理器对象所管理的容器对象。

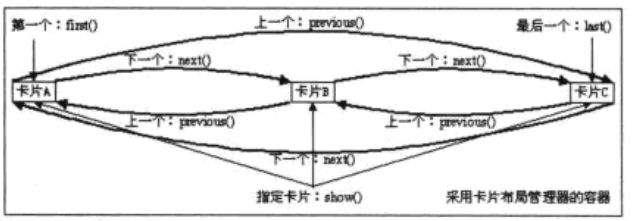


图 19.10　显示采用卡片布局管理器的容器中的卡片

注意　当利用 next()或 previous()方法显示卡片时，将循环显示所有卡片。例如，如果当前显示的是最后一个卡片，再调用 next()方法将显示第一个卡片；如果当前显示的是第一个卡片，再调用 previous()方法将显示最后一个卡片。

如果需要使用方法 show(Container parent, String name)显示卡片，则需要利用方法 add(Component comp, Object constraints)向其所管理的容器中添加组件，其中入口参数 parent 为卡片对象，name 为卡片的标签。

【例 19.2】　使用卡片布局管理器的方法，并利用 CardLayout 类中用来显示卡片的方法实现各种显示卡片的按钮。（**实例位置：光盘\TM\sl\19\2**）

```
public class ExampleFrame_02 extends JFrame {
    private JPanel cardPanel;                                   //采用卡片布局管理器的面板对象
    private CardLayout cardLayout;                              //卡片布局管理器对象
    public static void main(String args[]) {
        ExampleFrame_02 frame = new ExampleFrame_02();
        frame.setVisible(true);
    }
    public ExampleFrame_02() {
        super();
        setTitle("使用卡片布局管理器");
        setBounds(100, 100, 600, 400);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        cardLayout = new CardLayout();                          //创建一个卡片布局管理器对象
```

```
        cardPanel = new JPanel(cardLayout);                    //创建一个采用卡片布局管理器的面板对象
        getContentPane().add(cardPanel, BorderLayout.CENTER);   //添加到窗体中间
        String[] labelNames = { "卡片 A", "卡片 B", "卡片 C" };
        for (int i = 0; i < labelNames.length; i++) {
            final JLabel label = new JLabel(labelNames[i]);     //创建代表卡片的标签对象
            cardPanel.add(label, labelNames[i]);                //向采用卡片布局管理器的面板中添加卡片
        }
        final JPanel buttonPanel = new JPanel();                //创建一个按钮面板
        getContentPane().add(buttonPanel, BorderLayout.SOUTH);  //添加到窗体下方
        String[] buttonNames = { "第一个", "前一个", "卡片 A", "卡片 B", "卡片 C", "后一个", "最后一个" };
        for (int i = 0; i < buttonNames.length; i++) {
            final JButton button = new JButton(buttonNames[i]);
            button.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e) {
                    String buttonText = button.getText();
                    if (buttonText.equals("第一个"))
                        cardLayout.first(cardPanel);            //显示第一个卡片
                    else if (buttonText.equals("前一个"))
                        cardLayout.previous(cardPanel);         //显示上一个卡片
                    else if (buttonText.equals("卡片 A"))
                        cardLayout.show(cardPanel, "卡片 A"); //显示卡片 A
                    else if (buttonText.equals("卡片 B"))
                        cardLayout.show(cardPanel, "卡片 B"); //显示卡片 B
                    else if (buttonText.equals("卡片 C"))
                        cardLayout.show(cardPanel, "卡片 C"); //显示卡片 C
                    else if (buttonText.equals("后一个"))
                        cardLayout.next(cardPanel);             //显示下一个卡片
                    else
                        cardLayout.last(cardPanel);             //显示最后一个卡片
                }
            });
            buttonPanel.add(button);
        }
    }
}
```

运行结果如图 19.11 所示。

图 19.11　使用卡片布局管理器

19.3　网格组布局管理器

视频讲解：光盘\TM\lx\19\网格组布局管理器.exe

由 GridBagLayout 类实现的布局管理器称为网格组（GridBagLayout）布局管理器，它实现了一个

动态的矩形网格，这个矩形网格由无数个矩形单元格组成，每个组件可以占用一个或多个这样的单元格。所谓动态的矩形网格，就是可以根据实际需要随意增减矩形网格的行数和列数。

在向由 GridBagLayout 类管理的容器中添加组件时，需要为每个组件创建一个与之关联的 GridBagConstraints 类的对象，通过该类中的属性可以设置组件的布局信息，例如组件在网格组中位于第几行、第几列，以及需要占用几行几列等。

通过 GridBagLayout 类实现的矩形网格的绘制方向由容器决定，如果容器的方向是从左到右，则位于矩形网格左上角的单元格的列索引为 0，此时组件左上角的点为起始点；如果容器的方向是从右到左，则位于矩形网格右上角的单元格的列索引为 0，此时组件右上角的点为起始点。

下面将详细讲解 GridBagLayout 类中各个属性的功能和使用方法，以及在使用过程中需要注意的一些事项。

1．gridx 和 gridy 属性

这两个属性用来设置组件起始点所在单元格的索引值。需要注意的是，属性 gridx 设置的是 X 轴（即网格水平方向）的索引值，所以它表示的是组件起始点所在列的索引；属性 gridy 设置的是 Y 轴（即网格垂直方向）的索引值，所以它表示的是组件起始点所在行的索引，如图 19.12 所示。

2．gridwidth 和 gridheight 属性

这两个属性用来设置组件占用网格组的行数和列数。属性 gridwidth 为组件占用网格组的列数，也可以理解为以单元格为单位组件的宽度；属性 gridheight 为组件占用网格组的行数，也可以理解为以单元格为单位组件的高度，如图 19.13 所示。

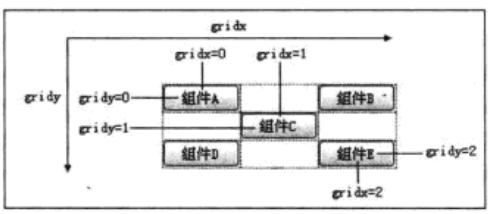


图 19.12　gridx 和 gridy 属性

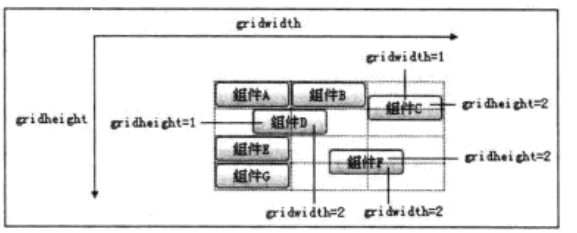


图 19.13　gridwidth 和 gridheight 属性

3．anchor 属性

属性 anchor 用来设置组件在其所在显示区域的显示位置。通常将显示区域从方向上划分为 9 个方位，分别为北方（NORTH）、东北（NORTHEAST）、东方（EAST）、东南（SOUTHEAST）、南方（SOUTH）、西南（SOUTHWEST）、西方（WEST）、西北（NORTHWEST）和中心（CENTER），如图 19.14 所示。

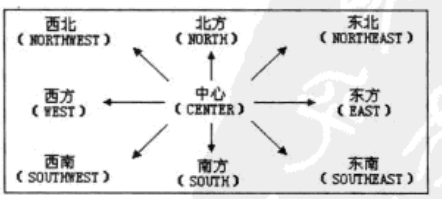


图 19.14　从方向上划分的 9 个方位

代表这 9 个方位的单词也是该类中的静态常量，可以利用这 9 个静态常量设置 anchor 属性，其中静态常量 CENTER 为默认位置。如图 19.15 所示为将组件设置为各个静态常量的效果，图中的黑点为将相应组件的 anchor 属性设置为 CENTER 时组件中心所在的位置。

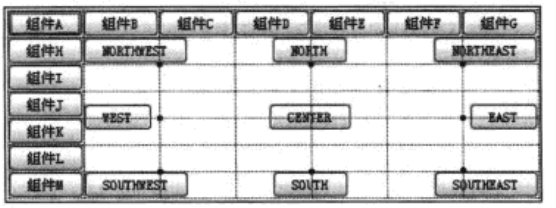


图 19.15　anchor 属性

说明

图 19.15 中的组件 A 至组件 M 是辅助组件，只起到占位作用。

4．fill 属性

属性 fill 用来设置组件的填充方式。当单元格显示区域的面积大于组件面积时，或者一个组件占用多个单元格时，显示组件可能不必占用所有显示区域，在这种情况下可以通过 fill 属性设置组件的填充方式。可以利用 4 个静态常量设置该属性，默认情况下是将该属性设置为静态常量 NONE，即不调整组件大小至填满显示区域；如果将该属性设置为静态常量 HORIZONTAL，表示只调整组件水平方向的大小（即组件宽度）至填满显示区域；如果将该属性设置为静态常量 VERTICAL，表示只调整组件垂直方向的大小（即组件高度）至填满显示区域；如果将该属性设置为静态常量 BOTH，则表示同时调整组件的宽度和高度至填满显示区域。具体效果如图 19.16 所示。

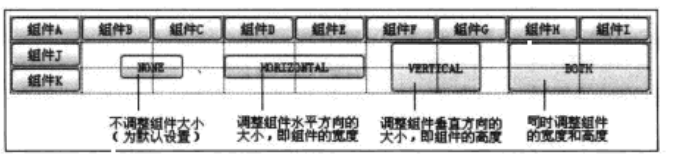


图 19.16　fill 属性

说明

图 19.16 中的组件 A 至组件 K 是辅助组件，只起到占位作用。

5．insets 属性

属性 insets 用来设置组件四周与单元格边缘之间的最小距离。该属性的类型为 Insets，Insets 类仅有一个构造方法 Insets(int top, int left, int bottom, int right)，它的 4 个入口参数依次为组件上方、左侧、下方和右侧的最小距离，单位为像素，如图 19.17 所示，默认为没有距离。

6．ipadx 和 ipady 属性

这两个属性用来修改组件的首选大小。属性 ipadx 用来修改组件的宽度，属性 ipady 用来修改组件

的高度。如果为正数，则在首选大小的基础上增加指定的宽度和高度，如图 19.18 所示。

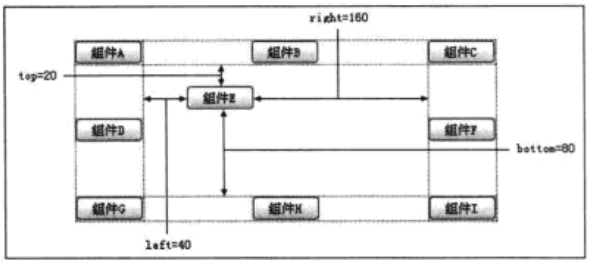


图 19.17　insets 属性

如果为负数，则在首选大小的基础上减小指定的宽度和高度，如图 19.19 所示。

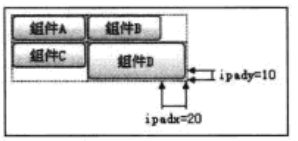


图 19.18　ipadx 和 ipady 属性为正数

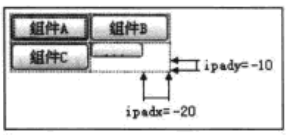


图 19.19　ipadx 和 ipady 属性为负数

7．weightx 和 weighty 属性

这两个属性用来设置网格组的每一行和每一列对额外空间的分布方式。在不设置属性 weightx 和 weighty（即采用默认设置）的情况下，当窗体调整到足够大时，将出现如图 19.20 所示的效果，组件全部聚集在窗体的中央，在组件四周出现了大片的额外空间。为了避免这种情况出现，可以通过这两个属性设置网格组的每一行和每一列对额外空间的分布方式。

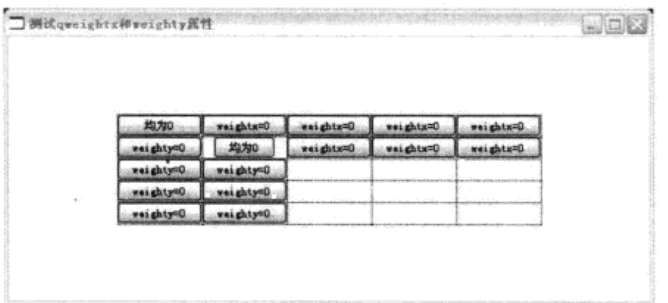


图 19.20　未设置属性 weightx 和 weighty

这两个属性的默认值均为 0，表示不分布容器的额外空间。属性 weightx 用来设置其所在列对额外空间的分布方式，如果在该列中设置了多个 weightx 属性，则取它们的最大值为该列的分布方式；属性 weighty 用来设置其所在行对额外空间的分布方式，如果在该行中设置了多个 weighty 属性，则取它们的最大值为该行的分布方式。

技巧　在设置网格组的每一行和每一列对额外空间的分布方式时，建议只设置第一行的 weightx 属性和第一列的 weighty 属性，这样会方便前期调试和后期维护。

网格组的行和列对额外空间的分布方式完全相同，下面以网格组的行为例详细讲解对额外空间的分布方式。网格组布局管理器首先计算出每一行的分布方式，即获取每一行的 weighty 属性的最大值；然后再计算每个最大值占所有最大值总和的百分比；最后将额外空间的相应百分比分配给对应行，如图 19.21 所示。

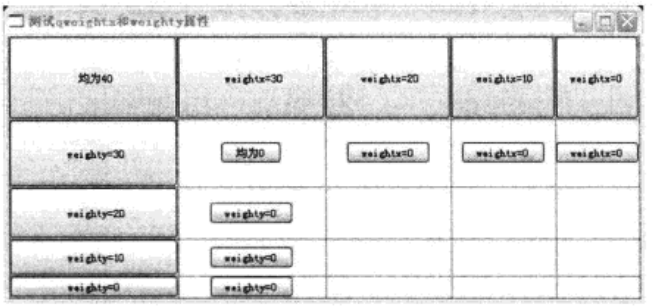


图 19.21　设置了属性 weightx 和 weighty

技巧　在设置网格组的每一行和每一列对额外空间的分布方式时，建议为各个属性按百分比取值。

【例 19.3】　使用网格组布局管理器管理组件，并且使用 GridBagConstraints 类中所有用来设置组件布局信息的属性。设计本例的出发点是对比各种设置的效果，所有注释内容均为用来对比的属性。（**实例位置：光盘\TM\sl\19\3**）

```
final JButton button = new JButton("A");
final GridBagConstraints gridBagConstraints = new GridBagConstraints();
gridBagConstraints.gridy = 0;                                   //起始点为第 1 行
gridBagConstraints.gridx = 0;                                   //起始点为第 1 列
gridBagConstraints.weightx = 10;                                //第 1 列的分布方式为 10%
gridBagConstraints.fill = GridBagConstraints.HORIZONTAL;
getContentPane().add(button, gridBagConstraints);
final JButton button_1 = new JButton("B");
final GridBagConstraints gridBagConstraints_1 = new GridBagConstraints();
gridBagConstraints_1.gridy = 0;
gridBagConstraints_1.gridx = 1;
gridBagConstraints_1.insets = new Insets(0, 5, 0, 0);           //设置组件左侧的最小距离
gridBagConstraints_1.weightx = 20;                              //第 1 列的分布方式为 20%
gridBagConstraints_1.fill = GridBagConstraints.HORIZONTAL;
getContentPane().add(button_1, gridBagConstraints_1);
final JButton button_2 = new JButton("C");
final GridBagConstraints gridBagConstraints_2 = new GridBagConstraints();
gridBagConstraints_2.gridy = 0;                                 //起始点为第 1 行
gridBagConstraints_2.gridx = 2;                                 //起始点为第 3 列
gridBagConstraints_2.gridheight = 2;                            //组件占用两行
gridBagConstraints_2.insets = new Insets(0, 5, 0, 0);
gridBagConstraints_2.weightx = 30;                              //第 1 列的分布方式为 30%
gridBagConstraints_2.fill = GridBagConstraints.BOTH;            //同时调整组件的宽度和高度
```

```
getContentPane().add(button_2, gridBagConstraints_2);
final JButton button_3 = new JButton("D");
final GridBagConstraints gridBagConstraints_3 = new GridBagConstraints();
gridBagConstraints_3.gridy = 0;
gridBagConstraints_3.gridx = 3;
gridBagConstraints_3.gridheight = 4;
gridBagConstraints_3.insets = new Insets(0, 5, 0, 5);          //设置组件左侧和右侧的最小距离
gridBagConstraints_3.weightx = 40;                             //第 1 列的分布方式为 40%
gridBagConstraints_3.fill = GridBagConstraints.BOTH;
getContentPane().add(button_3, gridBagConstraints_3);
final JButton button_4 = new JButton("E");
final GridBagConstraints gridBagConstraints_4 = new GridBagConstraints();
gridBagConstraints_4.gridy = 1;
gridBagConstraints_4.gridx = 0;
gridBagConstraints_4.gridwidth = 2;                            //组件占用两列
gridBagConstraints_4.insets = new Insets(5, 0, 0, 0);          //设置组件上方的最小距离
gridBagConstraints_4.fill = GridBagConstraints.HORIZONTAL;     //只调整组件的宽度
getContentPane().add(button_4, gridBagConstraints_4);
final JButton button_5 = new JButton("F");
final GridBagConstraints gridBagConstraints_5 = new GridBagConstraints();
gridBagConstraints_5.gridy = 2;                                //起始点为第 3 行
gridBagConstraints_5.gridx = 0;                                //起始点为第 1 列
gridBagConstraints_5.insets = new Insets(5, 0, 0, 0);
gridBagConstraints_5.fill = GridBagConstraints.HORIZONTAL;
getContentPane().add(button_5, gridBagConstraints_5);
final JButton button_6 = new JButton("G");
final GridBagConstraints gridBagConstraints_6 = new GridBagConstraints();
gridBagConstraints_6.gridy = 2;
gridBagConstraints_6.gridx = 1;
gridBagConstraints_6.gridwidth = 2;                            //组件占用两列
gridBagConstraints_6.gridheight = 2;                           //组件占用两行
gridBagConstraints_6.insets = new Insets(5, 5, 0, 0);
gridBagConstraints_6.fill = GridBagConstraints.BOTH;           //同时调整组件的宽度和高度
getContentPane().add(button_6, gridBagConstraints_6);
final JButton button_7 = new JButton("H");
final GridBagConstraints gridBagConstraints_7 = new GridBagConstraints();
gridBagConstraints_7.gridy = 3;
gridBagConstraints_7.gridx = 0;
gridBagConstraints_7.insets = new Insets(5, 0, 0, 0);
gridBagConstraints_7.fill = GridBagConstraints.HORIZONTAL;
getContentPane().add(button_7, gridBagConstraints_7);
```

运行结果如图 19.22 所示。

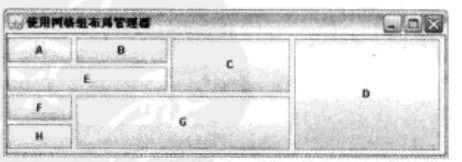


图 19.22　使用网格组布局管理器

组件 C 占用两行，填充方式为全部填充；组件 D 占用 4 行，填充方式为全部填充；组件 E 占用两列，填充方式为水平填充；组件 G 占用两行两列，填充方式为全部填充。

如果将组件 G 的填充方式改为垂直填充，并在水平方向上将组件的首选宽度增加 30 像素，将显示位置设为东方，修改后组件 G 的实现代码如下：

```
final JButton button_6 = new JButton("G");
final GridBagConstraints gridBagConstraints_6 = new GridBagConstraints();
gridBagConstraints_6.gridy = 2;
gridBagConstraints_6.gridx = 1;
gridBagConstraints_6.gridwidth = 2;                          //组件占用两列
gridBagConstraints_6.gridheight = 2;                         //组件占用两行
gridBagConstraints_6.insets = new Insets(5, 5, 0, 0);
gridBagConstraints_6.fill = GridBagConstraints.VERTICAL;     //只调整组件的高度
gridBagConstraints_6.ipadx = 30;                             //增加组件的首选宽度
gridBagConstraints_6.anchor = GridBagConstraints.EAST;       //显示在东方
getContentPane().add(button_6, gridBagConstraints_6);
```

运行结果如图 19.23 所示。

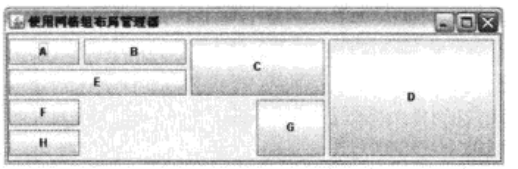


图 19.23　修改组件 G 设置后的效果

19.4　弹簧布局管理器

视频讲解：光盘\TM\lx\19\弹簧布局管理器.exe

由 SpringLayout 类实现的布局管理器称为弹簧（SpringLayout）布局管理器，利用该布局管理器管理组件，当改变窗体的大小时，能够在不改变组件间相对位置的前提下自动调整组件的大小，使组件依旧布满整个窗体，从而保证了窗体的整体效果。在使用该布局管理器时，需要与它的内部类 Constraints 以及 Spring 类配合使用，其中 Constraints 类用来管理组件的位置和大小，Spring 类用来创建弹簧和支架，这也是弹簧布局管理器的核心，即利用弹簧的可伸缩性动态控制组件的位置和大小。

19.4.1　使用弹簧布局管理器

弹簧布局管理器以容器和组件的边缘为操纵对象，通过为组件和容器边缘以及组件和组件边缘建立约束，实现对组件布局的管理，如图 19.24 所示。

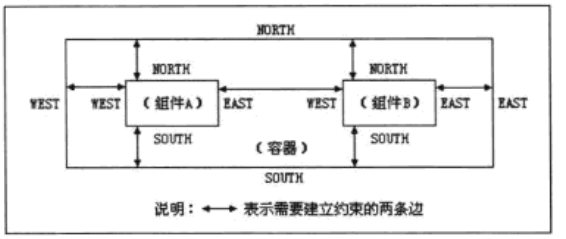


图 19.24　为容器和组件建立约束

通过方法 putConstraint(String e1, Component c1, int pad, String e2, Component c2)可以为各个边之间建立约束，该方法的入口参数说明如表 19.2 所示。

表 19.2　putConstraint()方法的入口参数说明

参　数	说　明
c1	需要参考的组件对象
e1	需要参考的组件对象具体需要参考的边
c2	被参考的组件对象（也可以是需要参考的组件对象所属的容器对象）
e2	被参考的组件对象具体被参考的边
pad	两条边之间的距离，即两个组件的间距

技巧　当 e2 在 e1 的北方或西方时，pad 应为正数；如果 e2 在 e1 的南方或东方时，pad 则应为负数；否则两条边将重叠。

其中入口参数 e1 和 e2 可以从该类提供的静态常量中选择，如表 19.3 所示，这些静态常量分别表示组件相应方向的边。

表 19.3　表示组件边缘的静态常量

静 态 常 量	常　量　值	说　明
EAST	East	表示组件东侧的边
SOUTH	South	表示组件南侧的边
WEST	West	表示组件西侧的边
NORTH	North	表示组件北侧的边

下面为图 19.24 中组件 A 的北侧边和其所属容器的北侧边建立约束，组件 A 的北侧边参考容器的北侧边，假设两条边之间的距离为 60 像素。代码如下：

```
springLayout.putConstraint(SpringLayout.NORTH, 组件 A 对象, 60, SpringLayout.NORTH, 容器对象);
```

接着为图 19.24 中组件 B 的西侧边和组件 A 的东侧边建立约束，假设两条边之间的距离为 60 像素。代码如下：

```
springLayout.putConstraint(SpringLayout.WEST, 组件 B 对象, 60, SpringLayout.EAST, 组件 A 对象);
```

最后为图 19.24 中组件 B 的东侧边和其所属容器的东侧边建立约束，假设两条边之间的距离为 60 像素。代码如下：

```
springLayout.putConstraint(SpringLayout.EAST, 组件 B 对象, -60, SpringLayout.EAST, 容器对象);
```

上面 3 个示例代码中，第一个演示了为组件边缘和容器边缘建立约束的方法；第二个演示了为组件边缘和组件边缘建立约束的方法；最后一个演示了当被参考的边缘在需要参考边缘的东侧时建立约束的方法，关键是需要将入口参数 pad 设为负数，否则将看不到组件 B 的东侧边缘。

说明 如果需要，也可以为图 19.24 中组件 A 的南侧边和其所属容器的北侧边建立约束；只要牢记同在水平或垂直方向上的任意两条边之间都可以建立约束即可。

【例 19.4】 使用弹簧布局管理器实现如图 19.25 所示的窗体，在调整窗体的大小后，组件仍会布满整个窗体，并且组件间的相对位置不会改变。（**实例位置：光盘\TM\sl\19\4**）

```
SpringLayout springLayout = new SpringLayout();                    //创建弹簧布局管理器对象
Container contentPane = getContentPane();                          //获得窗体容器对象
contentPane.setLayout(springLayout);                               //将窗体容器修改为采用弹簧布局管理器
JLabel topicLabel = new JLabel("主题：");
contentPane.add(topicLabel);
springLayout.putConstraint(SpringLayout.NORTH, topicLabel, 5,
        SpringLayout.NORTH, contentPane);                          //主题标签北侧——>容器北侧
springLayout.putConstraint(SpringLayout.WEST, topicLabel, 5,
        SpringLayout.WEST, contentPane);                           //主题标签西侧——>容器西侧
JTextField topicTextField = new JTextField();
contentPane.add(topicTextField);
springLayout.putConstraint(SpringLayout.NORTH, topicTextField, 5,
        SpringLayout.NORTH, contentPane);                          //主题文本框北侧——>容器北侧
springLayout.putConstraint(SpringLayout.WEST, topicTextField, 5,
        SpringLayout.EAST, topicLabel);                            //主题文本框西侧——>主题标签东侧
springLayout.putConstraint(SpringLayout.EAST, topicTextField, -5,
        SpringLayout.EAST, contentPane);                           //主题文本框东侧——>容器东侧
JLabel contentLabel = new JLabel("内容：");
contentPane.add(contentLabel);
springLayout.putConstraint(SpringLayout.NORTH, contentLabel, 5,
        SpringLayout.SOUTH, topicTextField);                       //内容标签北侧——>主题文本框南侧
springLayout.putConstraint(SpringLayout.WEST, contentLabel, 5,
        SpringLayout.WEST, contentPane);                           //内容标签西侧——>容器西侧
JScrollPane contentScrollPane = new JScrollPane();
contentScrollPane.setViewportView(new JTextArea());
contentPane.add(contentScrollPane);
springLayout.putConstraint(SpringLayout.NORTH, contentScrollPane, 5,
        SpringLayout.SOUTH, topicTextField);                       //滚动面板北侧——>主题文本框南侧
springLayout.putConstraint(SpringLayout.WEST, contentScrollPane, 5,
        SpringLayout.EAST, contentLabel);                          //滚动面板西侧——>内容标签东侧
springLayout.putConstraint(SpringLayout.EAST, contentScrollPane, -5,
        SpringLayout.EAST, contentPane);                           //滚动面板东侧——>容器东侧
```

```
JButton resetButton = new JButton("清空");
contentPane.add(resetButton);
springLayout.putConstraint(SpringLayout.SOUTH, resetButton, -5,
        SpringLayout.SOUTH, contentPane);                    //“清空”按钮南侧——>容器南侧
JButton submitButton = new JButton("确定");
contentPane.add(submitButton);
springLayout.putConstraint(SpringLayout.SOUTH, submitButton, -5,
        SpringLayout.SOUTH, contentPane);                    //“确定”按钮南侧——>容器南侧
springLayout.putConstraint(SpringLayout.EAST, submitButton, -5,
        SpringLayout.EAST, contentPane);                     //“确定”按钮东侧——>容器东侧
springLayout.putConstraint(SpringLayout.SOUTH, contentScrollPane, -5,
        SpringLayout.NORTH, submitButton);                   //滚动面板南侧——>“确定”按钮北侧
springLayout.putConstraint(SpringLayout.EAST, resetButton, -5,
        SpringLayout.WEST, submitButton);                    //“清空”按钮东侧——>“确定”按钮西侧
```

运行结果如图 19.25 所示。

调整大小后的窗体如图 19.26 所示。

图 19.25　利用弹簧布局管理器实现的窗体

图 19.26　调小后的窗体

19.4.2　使用弹簧和支柱

利用 Spring 类可以创建弹簧和支柱，但并不是通过构造方法实现，而是通过静态方法 constant()实现。该方法有如下两个重载方法。

- ☑ constant(int min, int pref, int max)：用来创建弹簧。
- ☑ constant(int pref)：用来创建支柱。

在利用方法 constant(int min, int pref, int max)创建弹簧时需要设置 3 个参数，分别为弹簧的最小值、首选值和最大值。最小值可以理解为弹簧被压缩到极限时的长度；首选值可以理解为弹簧自然放置情况下的长度；最大值可以理解为弹簧被拉伸到极限时的长度。如果这 3 个值相等，弹簧就没有了伸缩能力，也就变成了支柱，如果是创建支柱通常利用方法 constant(int pref)，因为它更方便。

弹簧还有一个参数就是实际值，可以理解为弹簧当前的长度，通过 getValue()和 setValue(int value)方法可以获得和设置弹簧当前的长度。

通过 width(Component c)和 height(Component c)方法可以快速获得一个弹簧，它们的入口参数均为组件对象，所得弹簧的最小值、首选值、最大值和实际值，分别为指定组件最小值（minimumSize）、首选值（preferredSize）、最大值（maximumSize）和实际值（size）的宽度或高度。

在 Spring 类中还提供了如下 4 个方法，均可用来根据指定条件获得一个新的弹簧。

- ☑ max(Spring s1, Spring s2)：所得弹簧的最小值、首选值和最大值，均为这两个弹簧相应值中相对较大的值。
- ☑ sum(Spring s1, Spring s2)：所得弹簧的最小值、首选值和最大值，均为这两个弹簧相应值的和。
- ☑ scale(Spring s, float factor)：所得弹簧的最小值、首选值和最大值，均为指定弹簧相应值的 factor 倍。
- ☑ minus(Spring s)：所得弹簧的最小值为指定弹簧最大值的负数，首选值为指定弹簧首选值的负数，最大值为指定弹簧最小值的负数。

如果需要使用弹簧建立约束，可以通过方法 putConstraint(String e1, Component c1, int pad, String e2, Component c2)的重载方法 putConstraint(String e1, Component c1, Spring s, String e2, Component c2)，实际上前者只是简单地调用了后者。

【例 19.5】 使用弹簧和支柱动态调整按钮间的距离，其中按钮间的距离为按钮与窗体间距离的 2 倍。（**实例位置：光盘\TM\sl\19\5**）

```
Spring vST = Spring.constant(20);                                   //创建一个支柱
Spring hSP = Spring.constant(20, 100, 500);                         //创建一个弹簧
JButton lButton = new JButton("按钮 L");
getContentPane().add(lButton);
springLayout.putConstraint(SpringLayout.NORTH, lButton, vST,
        SpringLayout.NORTH, getContentPane());                      //“按钮 L”北侧——>容器北侧
springLayout.putConstraint(SpringLayout.WEST, lButton, hSP,
        SpringLayout.WEST, getContentPane());                       //“按钮 L”西侧——>容器西侧
JButton rButton = new JButton("按钮 R");
getContentPane().add(rButton);
springLayout.putConstraint(SpringLayout.NORTH, rButton, 0,
        SpringLayout.NORTH, lButton);                               //“按钮 R”北侧——>“按钮 R”北侧
springLayout.putConstraint(SpringLayout.WEST, rButton, Spring.scale(hSP, 2),
        SpringLayout.EAST, lButton);                                //“按钮 R”西侧——>“按钮 L”东侧
springLayout.putConstraint(SpringLayout.EAST, getContentPane(), hSP,
        SpringLayout.EAST, rButton);                                //容器东侧——>“按钮 R”东侧
```

运行结果如图 19.27 所示。

可以随便调整窗体的宽度，如图 19.28 所示。

图 19.27 运行结果

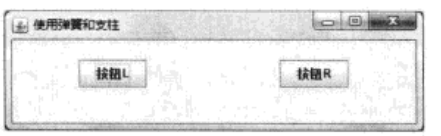


图 19.28 随便调整窗体宽度后的效果

19.4.3 利用弹簧控制组件大小

利用 SpringLayout 类的 getConstraints(Component c)方法可以得到 SpringLayout.Constraints 类的对

象，通过该类的 setWidth(Spring width)和 setHeight(Spring height)方法可以为组件的宽度和高度添加约束。

【例 19.6】　利用弹簧控制组件大小，其中按钮的宽度由弹簧控制，高度由支柱控制。（**实例位置：光盘\TM\sl\19\6**）

```
Spring widthSP = Spring.constant(60, 300, 600);                    //创建一个弹簧
Spring heightST = Spring.constant(60);                             //创建一个支柱
Constraints lButtonCons = springLayout.getConstraints(lButton);    //获得“按钮 L”的 Constraints 对象
lButtonCons.setWidth(widthSP);                                     //设置控制组件宽度的弹簧
lButtonCons.setHeight(heightST);                                   //设置控制组件高度的支柱
Constraints rButtonCons = springLayout.getConstraints(rButton);    //获得“按钮 R”的 Constraints 对象
rButtonCons.setWidth(widthSP);                                     //设置控制组件宽度的弹簧
rButtonCons.setHeight(heightST);                                   //设置控制组件高度的支柱
```

运行结果如图 19.29 所示。

可以随便调整窗体的宽度，如图 19.30 所示。

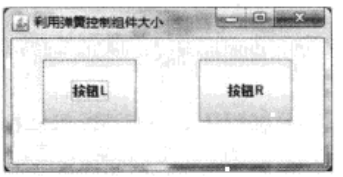


图 19.29　运行结果

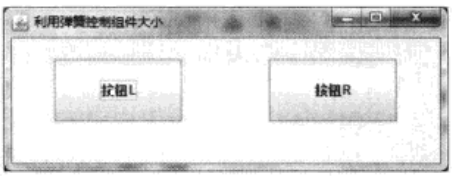


图 19.30　随便调整窗体宽度后的效果

19.5　经 典 范 例

19.5.1　经典范例 1：制作圆形布局管理器

视频讲解：光盘\TM\lx\19\制作圆形布局管理器.exe

尽管 Java 的 API 中实现了十余种布局管理器，但有时却不能完全满足不同用户的需求，此时可以考虑自己实现一个布局管理器。圆形是日常生活中最常见的几何形状之一，本范例演示如何实现圆形布局管理器。运行结果如图 19.31 所示。（**实例位置：光盘\TM\sl\19\7**）

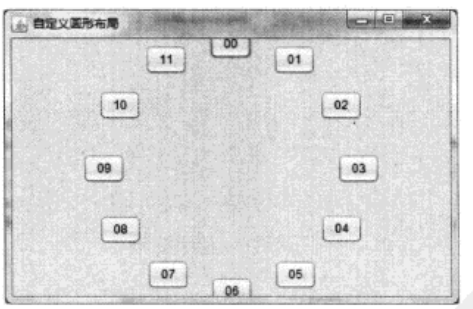


图 19.31　运行结果

编写 CircleLayout 类，该类继承了 LayoutManager。其中重点实现了 layoutContainer()方法，该方法用来实现如何放置容器中控件的功能。代码如下：

```
public void layoutContainer(Container parent) {
    double centerX = parent.getBounds().getCenterX();                    //获得容器中心的 X 坐标
    double centerY = parent.getBounds().getCenterY();                    //获得容器中心的 Y 坐标
    Insets insets = parent.getInsets();                                  //获得容器默认边框对象
    double horizon = centerX - insets.left;                              //获得水平可用长度的一半
    double vertical = centerY - insets.top;                              //获得垂直可用长度的一半
    double radius = horizon > vertical ? vertical : horizon;             //取小的为圆形半径
    int count = parent.getComponentCount();                              //获得容器中控件的个数
    for (int i = 0; i < count; i++) {                                    //依次设置所有可见控件的位置和大小
        Component component = parent.getComponent(i);
        if (component.isVisible()) {
            Dimension size = component.getPreferredSize();               //大小使用其最佳大小
            double angle = 2 * Math.PI * i / count;                      //获得角度的大小
            double x = centerX + radius * Math.sin(angle);               //获得圆周点的 X 坐标
            double y = centerY - radius * Math.cos(angle);               //获得圆周点的 Y 坐标
            //重新设置控件的位置和大小
            component.setBounds((int) x - size.width / 2, (int) y - size.height / 2, size.width, size.height);
        }
    }
}
```

19.5.2 经典范例 2：制作阶梯布局管理器

视频讲解：光盘\TM\lx\19\制作阶梯布局管理器.exe

尽管 Java 的 API 中实现了十余种布局管理器，有时却不能完全满足不同用户的需求。此时可以考虑自己实现一个布局管理器。阶梯是日常生活中最常见的几何形状之一。本范例演示如何实现阶梯布局管理器。运行结果如图 19.32 所示。（**实例位置：光盘\TM\sl\19\8**）

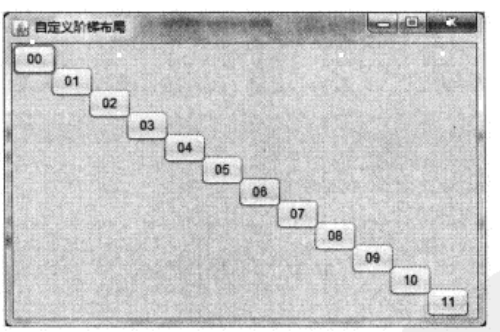


图 19.32　运行结果

编写 CircleLayout 类，该类继承了 LayoutManager。其中重点实现了 layoutContainer()方法，该方法用来实现如何放置容器中控件的功能。代码如下：

```
public void layoutContainer(Container parent) {
    Insets insets = parent.getInsets();                                    //获得容器默认边框对象
    int maxWidth = parent.getWidth() - (insets.left + insets.right);       //获得最大可用宽度
    int maxHeight = parent.getHeight() - (insets.top + insets.bottom);     //获得最大可用高度
    int count = parent.getComponentCount();                                //获得容器中控件的个数
    for (int i = 0; i < count; i++) {                                      //依次设置所有可见控件的位置和大小
        Component component = parent.getComponent(i);
        if (component.isVisible()) {
            Dimension size = component.getPreferredSize();                 //大小使用其最佳大小
            int x = maxWidth / count * i;                                  //将宽度分成 count 份根据 i 值调整 X 坐标
            int y = maxHeight / count * i;                                 //将高度分成 count 份根据 i 值调整 Y 坐标
            component.setBounds(x, y, size.width, size.height);            //重新设置控件的位置和大小
        }
    }
}
```

19.6 本章小结

通过本章的学习，相信读者已经可以熟练地使用一些高级布局管理器，包括箱式布局管理器、卡片布局管理器、网格组布局管理器和弹簧布局管理器。至此，已经掌握了 4 种布局管理器的使用方法，通过配合使用这 4 种布局管理器，针对每个程序界面都会有多种解决方案，在设计时可以充分考虑各种解决方案的优缺点，以便从中选择一种最适合的解决方案，开发出更美观、大方、实用的程序界面。

19.7 实战练习

1．利用本章学习的箱式、卡片和网格组布局管理器设计一个程序界面。（**答案位置：光盘\TM\sl\19\9**）

2．利用弹簧布局管理器设计一个程序界面。（**答案位置：光盘\TM\sl\19\10**）

第20章

AWT 绘图技术

（视频讲解：33 分钟）

要开发高级的应用程序，就必须适当掌握图像处理技术。它是程序开发不可缺少的技术，使用该技术可以为程序提供数据统计、图表分析等功能，提高程序的交互能力。本章将向读者介绍绘图技术的基本知识以及图像处理。

通过阅读本章，您可以：

- 了解 Java 绘制图形
- 了解 Java 绘图颜色与笔画属性
- 掌握 Java 绘制文本
- 掌握 Java 图片处理

PDG

20.1 绘制图形

视频讲解：光盘\TM\lx\20\绘制图形.exe

绘图是高级程序设计中非常重要的技术，例如应用程序需要绘制闪屏图片、绘制背景图片、绘制组件外观，Web 程序可以绘制统计图、绘制数据库存储的图片资源等。正所谓“一图胜千言”，使用图片能够更好地表达程序运行结果、进行细致的数据分析与保存等。本节将介绍 Java 语言程序设计的绘图类 Graphics 与 Griphics2D。

20.1.1 Graphics

Graphics 类是所有图形上下文的抽象基类，它允许应用程序在组件以及闭屏图像上进行绘制。Graphics 类封装了 Java 支持的基本绘图操作所需的状态信息，主要包括颜色、字体、画笔、文本、图像等。

Graphics 类提供了绘图常用的方法，利用这些方法可以实现直线、矩形、多边形、椭圆、圆弧等形状和文本、图片的绘制操作。另外，在执行这些操作之前，还可以使用相应的方法，设置绘图的颜色、字体等状态属性。

Java 可以分别使用 Graphics 和 Graphics2D 绘制图形，Graphics 类使用不同的方法实现不同图形的绘制，例如 drawLine()方法用于绘制直线、drawRect()方法用于绘制矩形、drawOval()方法用于绘制椭圆形等。

【例 20.1】 在项目中创建 DrawCircle 类，使该类继承 JFrame 类成为窗体组件，在类中创建继承 JPanel 类的 DrawPanel 内部类，并重写 paint()方法，实现绘制 5 个圆形组成的图案。（**实例位置：光盘\TM\sl\20\1**）

```
package com.lzw;
import java.awt.Graphics;
import javax.swing.JFrame;
import javax.swing.JPanel;
public class DrawCircle extends JFrame {
    private final int OVAL_WIDTH = 80;                          //圆形的宽
    private final int OVAL_HEIGHT = 80;                         //圆形的高
    public DrawCircle(){
        super();
        initialize();                                           //调用初始化方法
    }
    //初始化方法
    private void initialize() {
        this.setSize(300, 200);                                 //设置窗体大小
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);         //设置窗体关闭模式
        setContentPane(new DrawPanel());                        //设置窗体面板为绘图面板对象
        this.setTitle("绘图实例");                              //设置窗体标题
```

```
    }
    //主方法
    public static void main(String[] args) {
        new DrawCircle ().setVisible(true);                          //创建窗体
    }
    //创建绘图面板
    class DrawPanel extends JPanel {
        public void paint(Graphics g) {
            super.paint(g);
            g.drawOval(10, 10, OVAL_WIDTH, OVAL_HEIGHT);             //绘制第 1 个圆形
            g.drawOval(80, 10, OVAL_WIDTH, OVAL_HEIGHT);             //绘制第 2 个圆形
            g.drawOval(150, 10, OVAL_WIDTH, OVAL_HEIGHT);            //绘制第 3 个圆形
            g.drawOval(50, 70, OVAL_WIDTH, OVAL_HEIGHT);             //绘制第 4 个圆形
            g.drawOval(120, 70, OVAL_WIDTH, OVAL_HEIGHT);            //绘制第 5 个圆形
        }
    }
}
```

运行结果如图 20.1 所示。

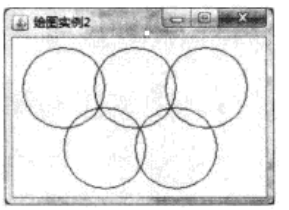


图 20.1　绘制圆形图的窗体

Graphics 类常用的图形绘制方法如表 20.1 所示。

表 20.1　Graphics 类常用的图形绘制方法

方　　法	说　　明	举　　例	绘 图 效 果
drawArc(int x, int y, int width, int height, int startAngle, int arcAngle)	弧形	drawArc(100,100,100,50,270,200);	
drawLine(int x1, int y1, int x2, int y2)	直线	drawLine(10,10,50,10); drawLine(30,10,30,40);	
drawOval(int x, int y, int width, int height)	椭圆	drawOval(10,10,50,30);	
drawPolygon(int[] xPoints, int[] yPoints, int nPoints)	多边形	int[] xs={10,50,10,50}; int[] ys={10,10,50,50}; drawPolygon(xs, ys, 4);	
drawPolyline(int[] xPoints, int[] yPoints, int nPoints)	多边线	int[] xs={10,50,10,50}; int[] ys={10,10,50,50}; drawPolyline(xs, ys, 4);	
drawRect(int x, int y, int width, int height)	矩形	drawRect(10,10,100,50);	
drawRoundRect(int x, int y, int width, int height, int arcWidth, int arcHeight)	圆角矩形	drawRoundRect(10, 10, 50, 30, 10, 10);	

续表

方　法	说　明	举　例	绘图效果
fillArc(int x, int y, int width, int height, int startAngle, int arcAngle)	实心弧形	fillArc(100,100,50,30,270,200);	
fillOval(int x, int y, int width, int height)	实心椭圆	fillOval(10,10,50,30);	
fillPolygon(int[] xPoints, int[] yPoints, int nPoints)	实心多边形	int[] xs={10,50,10,50}; int[] ys={10,10,50,50}; fillPolygon(xs, ys, 4);	
fillRect(int x, int y, int width, int height)	实心矩形	fillRect(10, 10, 50, 30);	
fillRoundRect(int x, int y, int width, int height, int arcWidth, int arcHeight)	实心圆角矩形	g.fillRoundRect(10, 10, 50, 30,10,10);	

20.1.2 Graphics2D

使用 Graphics 类可以完成简单的图形绘制任务，但是它所实现的功能非常有限，如无法改变线条的粗细、不能对图片使用旋转和模糊等过滤效果。

Graphics2D 继承 Graphics 类，实现了功能更加强大的绘图操作的集合。由于 Graphics2D 类是 Graphics 类的扩展，也是推荐使用的 Java 绘图类，所以本章主要介绍如何使用 Graphics2D 类实现 Java 绘图。

说明 Graphics2D 是推荐使用的绘图类，但是程序设计中提供的绘图对象大多是 Graphics 类的实例对象，这时应该使用强制类型转换将其转换为 Graphics2D 类型。例如：

```
public void paint(Graphics g) {
    Graphics2D g2 = (Graphics2D) g;                    //强制类型转换为 Graphics2D 类型
    g2.……
    }
}
```

Graphics2D 是继承 Graphics 类编写的，它包含了 Graphics 类的绘图方法并添加了更强的功能，是推荐使用的绘图类。Graphics2D 可以分别使用不同的类来表示不同的形状，如 Line2D、Rectangle2D 等。

要绘制指定形状的图形，需要先创建并初始化该图形类的对象，这些图形类必须是 Shape 接口的实现类，然后使用 Graphics2D 类的 draw()方法绘制该图形对象或者使用 fill()方法填充该图形对象。语法格式如下：

```
draw(Shape form)
```

或者

```
fill(Shape form)
```

其中，form 表示实现 Shape 接口的对象。

java.awt.geom 包中提供了如下一些常用的图形类，这些图形类都实现了 Shape 接口。

- ☑ Arc2D
- ☑ CubicCurve2D
- ☑ Ellipse2D
- ☑ Line2D
- ☑ Point2D
- ☑ QuadCurve2D
- ☑ Rectangle2D
- ☑ RoundRectangle2D

注意　各图形类都是抽象类型的。在不同图形类中有 Double 和 Float 两个实现类，这两个实现类以不同精度构建图形对象。为方便计算，在程序开发中经常使用 Double 类的实例对象进行图形绘制，但是如果程序中要使用成千上万个图形，则建议使用 Float 类的实例对象进行绘制，这样会节省内存空间。

【例 20.2】　在窗体的实现类中创建图形类的对象，然后使用 Graphics2D 类绘制和填充这些图形。（实例位置：光盘\TM\sl\20\2）

```
package com.lzw;
import java.awt.*;
import java.awt.geom.*;
import javax.swing.*;
public class DrawFrame extends JFrame {
    public DrawFrame() {
        super();
        initialize();                                          //调用初始化方法
    }
    //初始化方法
    private void initialize() {
        this.setSize(300, 200);                                //设置窗体大小
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);        //设置窗体关闭模式
        add(new CanvasPanel());                                //设置窗体面板为绘图面板对象
        this.setTitle("绘图实例 2");                           //设置窗体标题
    }
    public static void main(String[] args) {
        new DrawFrame().setVisible(true);
    }
    class CanvasPanel extends JPanel {
        public void paint(Graphics g) {
            super.paint(g);
            Graphics2D g2 = (Graphics2D) g;
            Shape[] shapes = new Shape[4];                     //声明图形数组
            shapes[0] = new Ellipse2D.Double(5, 5, 100, 100);  //创建圆形对象
            shapes[1] = new Rectangle2D.Double(110, 5, 100, 100);  //创建矩形对象
            shapes[2] = new Rectangle2D.Double(15, 15, 80, 80);    //创建圆形对象
```

```
            shapes[3] = new Ellipse2D.Double(120, 15, 80, 80);          //创建矩形对象
            for (Shape shape : shapes) {                                //遍历图形数组
                Rectangle2D bounds = shape.getBounds2D();
                if (bounds.getWidth() == 80)
                    g2.fill(shape);                                     //填充图形
                else
                    g2.draw(shape);                                     //绘制图形
            }
        }
    }
}
```

运行结果如图 20.2 所示。

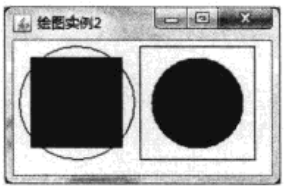


图 20.2　运行结果

20.1.3　范例 1：绘制指定角度的填充扇形

在 Java 中绘制指定角度的填充扇形，编写 Java 程序并运行，在窗体上将输出填充扇形。运行结果如图 20.3 所示。（**实例位置：光盘\TM\sl\20\3**）

（1）在项目中创建一个继承 JFrame 类的 DrawSectorFrame 窗体类。

（2）在 DrawSectorFrame 窗体类中创建内部面板类 DrawSectorPanel，并重写 JComponent 类的 paint()方法，在该方法中使用 Graphics 类的 fillAre()方法绘制填充扇形。

（3）将内部面板类 DrawSectorPanel 的实例添加到窗体类 DrawSectorFrame 的内容面板上，用于在窗体上显示绘制的填充扇形。代码如下：

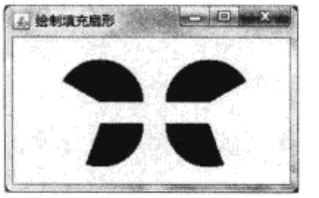


图 20.3　绘制指定角度的填充扇形

```
class DrawSectorPanel extends JPanel {                          //创建内部面板类
    public void paint(Graphics g) {                             //重写 paint()方法
        g.fillArc(40, 20, 80, 80, 0, 150);                      //绘制填充扇形
        g.fillArc(140, 20, 80, 80, 180, -150);                  //绘制填充扇形
        g.fillArc(40, 40, 80, 80, 0, -110);                     //绘制填充扇形
        g.fillArc(140, 40, 80, 80, 180, 110);                   //绘制填充扇形
    }
}
```

20.1.4　范例 2：绘制多边形

在 Java 中绘制多边形，编写 Java 程序并运行，在窗体上将输出多边形。运行结果如图 20.4 所示。（**实例位置：光盘\TM\sl\20\4**）

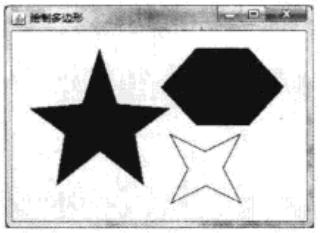


图 20.4　绘制多边形

（1）在项目中创建一个继承 JFrame 类的 DrawPolygonFrame 窗体类。

（2）在 DrawPolygonFrame 窗体类中创建内部面板类 DrawPolygonPanel，并重写 JComponent 类的 paint()方法，在该方法中使用 Graphics 类的 drawPolygon()和 fillPolygon()方法绘制多边形。

（3）将内部面板类 DrawPolygonPanel 的实例添加到窗体类 DrawPolygonFrame 的内容面板上，用于在窗体上显示绘制的多边形。代码如下：

```
class DrawPolygonPanel extends JPanel {                                  //创建内部面板类
    public void paint(Graphics g) {                                      //重写 paint()方法
        int[] x1 = { 100,120,180,140,150,100,50,60,20,80 };              //多边形的横坐标
        int[] y1 = { 20,85,90,120,180,140,180,120,90,85 };               //多边形的纵坐标
        int n1 = 10;                                                     //多边形的边数
        g.fillPolygon(x1, y1, n1);                                       //绘制多边形
        int[] x2 = { 210, 270, 310, 270, 210, 170 };                     //多边形的横坐标
        int[] y2 = { 20, 20, 65, 110, 110, 65 };                         //多边形的纵坐标
        int n2 = 6;                                                      //多边形的边数
        g.fillPolygon(x2, y2, n2);                                       //绘制实心多边形
        int[] x3 = { 180, 220, 260, 240, 260, 220, 180, 200 };           //多边形的横坐标
        int[] y3 = { 120, 140, 120, 160, 200, 180, 200, 160 };           //多边形的纵坐标
        int n3 = 8;                                                      //多边形的边数
        g.drawPolygon(x3, y3, n3);                                       //绘制多边形
    }
}
```

20.2　绘图颜色与笔画属性

视频讲解：光盘\TM\lx\20\绘图颜色与笔画属性.exe

Java 语言使用 Color 类封装颜色的各种属性，并对颜色进行管理。另外，在绘制图形时还可以指定线的粗细、虚线还是实线等笔画属性。

20.2.1　设置颜色

使用 Color 类可以创建任何颜色的对象，而不用担心不同平台对该颜色支持与否，因为 Java 以跨

平台和与硬件无关的方式支持颜色管理。创建 Color 对象的构造方法如下：

```
Color col = new Color(int r, int g, int b)
```

或者

```
Color col = new Color(int rgb)
```

- ☑ rgb：颜色值，该值是红、绿、蓝三原色的总和。
- ☑ r：该参数是三原色中红色的取值。
- ☑ g：该参数是三原色中绿色的取值。
- ☑ b：该参数是三原色中蓝色的取值。

Color 类定义了常用色彩的常量值，如表 20.2 所示。这些常量都是静态的 Color 对象，可以直接使用这些常量值定义颜色对象。

表 20.2　常用的 Color 常量

常　量　名	颜　色　值
Color BLACK	黑色
Color BLUE	蓝色
Color CYAN	青色
Color DARK_GRAY	深灰色
Color GRAY	灰色
Color GREEN	绿色
Color L IGHT_GRAY	浅灰色
Color MAGENTA	洋红色
Color ORANGE	桔黄色
Color PINK	粉红色
Color RED	红色
Color WHITE	白色
Color YELLOW	黄色

绘图类可以使用 setColor()方法设置当前颜色，语法格式如下：

```
setColor(Color color);
```

color：Color 对象，代表一个颜色值，如红色、黄色或者默认的黑色。

【例 20.3】　设置当前绘图颜色为红色。

```
public void paint(Graphics g) {
    super.paint(g);
    Graphics2D g2 = (Graphics2D) g;
    g.setColor(Color.RED);
    ……
}
```

说明 设置绘图颜色以后，再进行绘图或者绘制文本，都会采用该颜色作为前景色；如果想再绘制其他颜色的图形或文本，需要在此调用 setColor()方法设置其他颜色。

20.2.2　笔画属性

在默认情况下，Graphics 绘图类使用的笔画属性是粗细为 1 个像素的正方形，而 Java2D 的 Graphics2D 类可以调用 setStroke()方法设置笔画的属性，例如改变线条的粗细、使用实线还是虚线、定义线段端点的形状和风格等，语法格式如下：

```
setStroke(Stroke stroke)
```

stroke：Stroke 接口的实现类。

setStroke()方法必须接受一个 Stroke 接口的实现类作参数，java.awt 包中提供了 BasicStroke 类，它实现了 Stroke 接口，并且通过不同的构造方法创建笔画属性不同的对象，这些构造方法如下：

- ☑ BasicStroke()
- ☑ BasicStroke(float width)
- ☑ BasicStroke(float width, int cap, int join)
- ☑ BasicStroke(float width, int cap, int join, float miterlimit)
- ☑ BasicStroke(float width, int cap, int join, float miterlimit, float[] dash, float dash_phase)

这些构造方法中的参数说明如表 20.3 所示。

表 20.3　参数说明

参　数　名	说　　明
width	笔画宽度，此宽度必须大于或等于 0.0f。如果将宽度设置为 0.0f，则将笔画设置为当前设备的默认宽度
cap	线段端点的装饰
join	应用在路径线段交汇处的装饰
miterlimit	斜接处的剪裁限制。该参数值必须大于或等于 1.0f
dash	表示虚线模式的数组
dash_phase	开始虚线模式的偏移量

cap 参数可以使用 CAP_BUTT、CAP_ROUND 和 CAP_SQUARE 常量，这 3 个常量对线段端点的装饰效果如图 20.5 所示。

join 参数用于修饰线段交汇效果，可以使用 JOIN_BEVEL、JOIN_MITER 和 JOIN_ROUND 常量，这 3 个常量对线段交汇的修饰效果如图 20.6 所示。

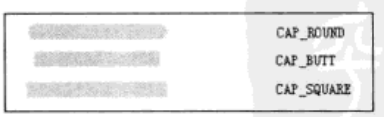


图 20.5　cap 参数对线段端点的装饰效果

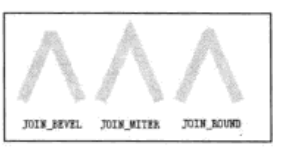


图 20.6　join 参数修饰线段交汇的效果

20.2.3 范例 3：为图形填充渐变色

在 Java 中绘制图形时，为图形填充渐变色，编写 Java 程序并运行，在窗体上将输出图形渐变色。运行结果如图 20.7 所示。（**实例位置：光盘\TM\sl\20\5**）

（1）在项目中创建一个继承 JFrame 类的 FillGradientFrame 窗体类。

（2）在 FillGradientFrame 窗体类中创建内部面板类 FillGradientPanel，并重写 JComponent 类的 paint()方法，在该方法中使用 Graphics2D 类的 setPaint()方法设置封装了渐变色的对象，该对象是通过 GradientPaint 类创建的。

（3）将内部面板类 FillGradientPanel 的实例添加到窗体类 FillGradientFrame 的内容面板上，用于在窗体上显示填充了渐变颜色后的图形。代码如下：

图 20.7 为图形填充渐变色的效果

```
class FillGradientPanel extends JPanel {                    //创建内部面板类
    public void paint(Graphics g) {                         //重写 paint()方法
        Graphics2D g2 = (Graphics2D) g;                     //获得 Graphics2D 对象
        //创建矩形对象
        Rectangle2D.Float rect = new Rectangle2D.Float(20, 20, 280, 140);
        //创建循环渐变的 GradientPaint 对象
        GradientPaint paint = new GradientPaint(20,20,Color.BLUE,100,80,Color.RED,true);
        g2.setPaint(paint);                                 //设置渐变
        g2.fill(rect);                                      //绘制矩形
    }
}
```

20.2.4 范例 4：设置笔画的粗细

在 Java 中绘制图形时，设置笔画的粗细，编写 Java 程序并运行，在窗体上将输出不同粗细的笔画。运行结果如图 20.8 所示。（**实例位置：光盘\TM\sl\20\6**）

（1）在项目中创建一个继承 JFrame 类的 StrokeWidthFrame 窗体类。

（2）在 StrokeWidthFrame 窗体类中创建内部面板类 ChangeStrokeWidthPanel，并重写 JComponent 类的 paint()方法，在该方法中使用 BasicStroke 类创建笔画对象，并使用 Graphics2D 类的 setStroke()方法设置笔画的粗细。

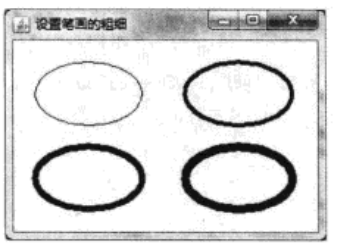


图 20.8 设置笔画粗细的效果

（3）将内部面板类 ChangeStrokeWidthPanel 的实例添加到窗体类 StrokeWidthFrame 的内容面板上，用于在窗体上显示设置笔画粗细后的图形。代码如下：

```
class ChangeStrokeWidthPanel extends JPanel {                          //创建内部面板类
    public void paint(Graphics g) {                                    //重写 paint()方法
        Graphics2D g2 = (Graphics2D)g;                                 //获得 Graphics2D 对象
        BasicStroke stroke = new BasicStroke(1);                       //创建宽度为 1 的笔画对象
        g2.setStroke(stroke);                                          //设置笔画对象
        Ellipse2D.Float ellipse = new Ellipse2D.Float(20,20,100,60);   //创建椭圆对象
        g2.draw(ellipse);                                              //绘制椭圆
        stroke = new BasicStroke(4);                                   //创建宽度为 4 的笔画对象
        g2.setStroke(stroke);                                          //设置笔画对象
        ellipse = new Ellipse2D.Float(160,20,100,60);                  //创建椭圆对象
        g2.draw(ellipse);                                              //绘制椭圆
        stroke = new BasicStroke(6);                                   //创建宽度为 6 的笔画对象
        g2.setStroke(stroke);                                          //设置笔画对象
        ellipse = new Ellipse2D.Float(20,100,100,60);                  //创建椭圆对象
        g2.draw(ellipse);                                              //绘制椭圆
        stroke = new BasicStroke(8);                                   //创建宽度为 8 的笔画对象
        g2.setStroke(stroke);                                          //设置笔画对象
        ellipse = new Ellipse2D.Float(160,100,100,60);                 //创建椭圆对象
        g2.draw(ellipse);                                              //绘制椭圆
    }
}
```

20.3　绘 制 文 本

视频讲解：光盘\TM\lx\20\绘制文本.exe

Java 绘图类也可以绘制文本内容，在绘制文本之前可以设置使用的字体、大小等。本节将介绍如何绘制文本以及设置文本的字体。

20.3.1　设置字体

Java 使用 Font 类封装了字体的大小、样式等属性，该类在 java.awt 包中定义，其构造方法可以指定字体的名称、大小和样式 3 个属性。语法格式如下：

```
Font(String name, int style, int size)
```

- ☑ name：字体的名称。
- ☑ style：字体的样式。
- ☑ size：字体的大小。

其中字体样式可以使用 Font 类的 PLAIN、BOLD 和 ITALIC 常量，这 3 个字体样式常量的效果如图 20.9 所示。

普通样式	PLAIN
粗体样式	BOLD
斜体样式	ITALIC
斜体组合斜体样式	ITALIC|BOLD

图 20.9　字体样式

设置绘图类的字体可以使用绘图类的 setFont()方法。设置字体以后在图形上下文中绘制的所有文字都使用该字体，除非再次设置其他字体。语法格式如下：

```
setFont(Font font)
```

font：Font 类的字体对象。

20.3.2　显示文字

Graphics2D 类提供了 drawString()方法，使用该方法可以实现图形上下文的文本绘制，从而实现在图片上显示文字的功能。语法格式如下：

```
drawString(String str, int x, int y);
```

或者

```
drawString(String str, float x, float y)
```

☑　str：要绘制的文本字符串。

☑　x：绘制字符串的水平起始位置。

☑　y：绘制字符串的垂直起始位置。

这两个方法唯一不同的就是方法使用的 x 和 y 参数的类型不同。

【例 20.4】　绘制一个矩形图，在矩形图的中间显示文本，文本的内容是当前时间。（**实例位置：光盘\TM\sl\20\7**）

```
package com.lzw;
import java.awt.*;
import java.awt.geom.Rectangle2D;
import java.util.Date;
import javax.swing.JFrame;
public class DrawString extends JFrame {
    private Shape rect;                                          //矩形对象
    private Font font;                                           //字体对象
    private Date date;                                           //当前日期对象
    public DrawString() {
        rect = new Rectangle2D.Double(10,10,200,80);
        font = new Font("宋体",Font.BOLD,16);
        date = new Date();
        this.setSize(230, 140);                                  //设置窗体大小
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);          //设置窗体关闭模式
```

```
        add(new CanvasPanel());                                    //设置窗体面板为绘图面板对象
        this.setTitle("绘图文本");                                  //设置窗体标题
    }
    public static void main(String[] args) {
        new DrawString().setVisible(true);
    }
    class CanvasPanel extends Canvas {
        public void paint(Graphics g) {
            super.paint(g);
            Graphics2D g2 = (Graphics2D) g;
            g2.setColor(Color.CYAN);                               //设置当前绘图颜色
            g2.fill(rect);                                         //填充矩形
            g2.setColor(Color.BLUE);                               //设置当前绘图颜色
            g2.setFont(font);                                      //设置字体
            g2.drawString("现在时间是", 20, 30);                    //绘制文本
            g2.drawString(String.format("%tr", date), 50, 60);     //绘制时间文本
        }
    }
}
```

运行结果如图 20.10 所示。

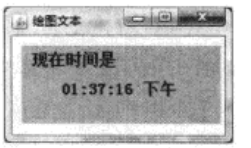


图 20.10　在窗体中绘制文本

20.3.3　范例 5：设置文本的字体

在 Java 中绘制文本时，如何设置文本的字体，其中包括字体名称、字体大小和字体样式。编写 Java 程序并运行，运行结果如图 20.11 所示。（**实例位置：光盘\TM\sl\20\8**）

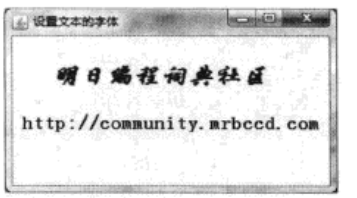


图 20.11　设置文本字体的效果

（1）在项目中创建一个继承 JFrame 类的 TextFontFrame 窗体类。

（2）在 TextFontFrame 窗体类中创建内部面板类 ChangeTextFontPanel，并重写 JComponent 类的 paint()方法，在该方法中使用 Font 类创建字体对象，并使用 Graphics 类的 setFont()方法设置文本的字体。

（3）将内部面板类 ChangeTextFontPanel 的实例添加到窗体类 TextFontFrame 的内容面板上，用于在窗体上显示指定字体后的文本。代码如下：

```
class ChangeTextFontPanel extends JPanel {                         //创建内部面板类
    public void paint(Graphics g) {                                //重写 paint()方法
        String value = "明日编程词典社区";
        int x = 40;                                                //文本位置的横坐标
        int y = 50;                                                //文本位置的纵坐标
        Font font = new Font("华文行楷", Font.BOLD + Font.ITALIC, 26);//创建字体对象
        g.setFont(font);                                           //设置字体
        g.drawString(value, x, y);                                 //绘制文本
```

```
        value = "http://community.mrbccd.com";
        x = 10;                                          //文本位置的横坐标
        y = 100;                                         //文本位置的纵坐标
        font = new Font("宋体", Font.BOLD, 20);          //创建字体对象
        g.setFont(font);                                 //设置字体
        g.drawString(value, x, y);                       //绘制文本
    }
}
```

20.3.4 范例 6：设置文本的图形和颜色

本范例演示在 Java 中绘制文本和图形时，如何设置文本和图形的颜色。运行结果如图 20.12 所示。（**实例位置：光盘\TM\sl\20\9**）

图 20.12 设置文本和图形颜色的效果

（1）在项目中创建一个继承 JFrame 类的 TextAndShapeColorFrame 窗体类。

（2）在 TextAndShapeColorFrame 窗体类中创建内部面板类 TextAndShapeColorPanel，并重写 JComponent 类的 paint()方法，在该方法中使用 Color 类创建颜色对象，并使用 Graphics 类的 setColor()方法设置文本和图形的颜色。

（3）将内部面板类 TextAndShapeColorPanel 的实例添加到窗体类 TextAndShapeColorFrame 的内容面板上，用于在窗体上显示设置颜色后的文本和图形。代码如下：

```
class TextAndShapeColorPanel extends JPanel {            //创建内部面板类
    public void paint(Graphics g) {                      //重写 paint()方法
        String value = "只要努力————";
        int x = 60;                                      //文本位置的横坐标
        int y = 60;                                      //文本位置的纵坐标
        Color color = new Color(255,0,0);                //创建颜色对象
        g.setColor(color);                               //设置颜色
        g.drawString(value, x, y);                       //绘制文本
        value = "一切皆有可能";
        x = 140;                                         //文本位置的横坐标
        y = 100;                                         //文本位置的纵坐标
        color = new Color(0,0,255);                      //创建颜色对象
        g.setColor(color);                               //设置颜色
        g.drawString(value, x, y);                       //绘制文本
        color = Color.ORANGE;                            //通过 Color 类的字段获得颜色对象
        g.setColor(color);                               //设置颜色
        g.drawRoundRect(40,30,200,100,40,30);            //绘制圆角矩形
        g.drawRoundRect(45,35,190,90,36,26);             //绘制圆角矩形
    }
}
```

20.4 图 片 处 理

视频讲解：光盘\TM\lx\20\图片处理.exe

20.4.1 绘制图片

绘图类不仅可以绘制图形和文本，还可以使用 drawImage()方法将图片资源显示到绘图上下文中，而且可以实现各种特效处理，如图片的缩放、翻转等。本节主要介绍如何显示图片，语法格式如下：

```
drawImage(Image img, int x, int y, ImageObserver observer)
```

该方法将 img 图片显示在 x、y 指定的位置上，方法中涉及的参数说明如表 20.4 所示。

表 20.4　参数说明

参 数 名	说　明
img	要显示的图片对象
x	水平位置
y	垂直位置
observer	要通知的图像观察者

该方法的使用与绘制文本的 drawString()方法类似，唯一不同的是 drawImage()方法需要指定通知的图像观察者。

【例 20.5】　在整个窗体中显示图片，图片的大小保持不变。（实例位置：光盘\TM\sl\20\10）

```
package com.lzw;
import java.awt.*;
import java.net.URL;
import javax.swing.JFrame;
public class DrawImage extends JFrame {
    Image img;
    public DrawImage() {
        URL imgUrl = DrawImage.class.getResource("img.jpg");      //获取图片资源的路径
        img=Toolkit.getDefaultToolkit().getImage(imgUrl);         //获取图片资源
        this.setSize(440, 300);                                   //设置窗体大小
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);           //设置窗体关闭模式
        add(new CanvasPanel());                                   //设置窗体面板为绘图面板对象
        this.setTitle("绘制图片");                                 //设置窗体标题
    }
    public static void main(String[] args) {
        new DrawImage().setVisible(true);
    }
    class CanvasPanel extends Canvas {
```

```
        public void paint(Graphics g) {
            super.paint(g);
            Graphics2D g2 = (Graphics2D) g;
            g2.drawImage(img, 0, 0, this);                    //显示图片
        }
    }
}
```

运行结果如图 20.13 所示。

图 20.13　显示图片的窗体

开发高级的桌面应用程序，必须掌握一些图像处理与动画制作的技术，例如在程序中显示统计图、销售趋势图、动态按钮等。本节将在 Java 绘图的基础上讲解图像处理技术。

20.4.2　放大与缩小

在讲解绘制图片时，使用了 drawImage()方法将图片以原始大小显示在窗体中，想要实现图片的放大与缩小则需要使用它的重载方法。语法格式如下：

```
drawImage(Image img, int x, int y, int width, int height, ImageObserver observer)
```

该方法将 img 图片显示在 x、y 指定的位置上，并指定图片的宽度和高度属性。方法中涉及的参数说明如表 20.5 所示。

表 20.5　参数说明

参　数　名	说　　明
img	要显示的图片对象
x	水平位置
y	垂直位置
width	图片的新宽度属性
height	图片的新高度属性
observer	要通知的图像观察者

【例 20.6】　在窗体中显示原始大小的图片，然后通过两个按钮的单击事件，分别显示该图片缩小与放大后的效果。（**实例位置：光盘\TM\sl\20\11**）

```
package com.lzw;
import java.awt.*;
import java.net.*;
import javax.swing.*;
public class ImageZoom extends JFrame {
    Image img;
    ...//省略部分成员变量
    public ImageZoom() {
        initialize();                                         //调用初始化方法
```

```
    }
    //界面初始化方法
    private void initialize() {
        URL imgUrl = ImageZoom.class.getResource("img.jpg");          //获取图片资源的路径
        img = Toolkit.getDefaultToolkit().getImage(imgUrl);           //获取图片资源
        canvas = new MyCanvas();
        this.setBounds(100, 100, 800, 600);                           //设置窗体大小和位置
        this.setContentPane(getContentPanel());                       //设置内容面板
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);               //设置窗体关闭模式
        this.setTitle("绘制图片");                                     //设置窗体标题
    }
    ...//省略布局方法的代码
    //获取滑块组件
    private JSlider getJSlider() {
        if (jSlider == null) {
            jSlider = new JSlider();                                  //创建滑块组件
            jSlider.setMaximum(1000);                                 //设置滑块最大取值
            jSlider.setValue(100);                                    //设置滑块最小取值
            jSlider.setMinimum(1);                                    //设置滑块当前值
            //添加滑块改变事件
            jSlider.addChangeListener(new javax.swing.event.ChangeListener() {
                public void stateChanged(javax.swing.event.ChangeEvent e) {
                    canvas.repaint();                                 //重新绘制画板内容
                }
            });
        }
        return jSlider;
    }
    //主方法
    public static void main(String[] args) {
        new ImageZoom().setVisible(true);
    }
    //画板类
    class MyCanvas extends Canvas {
        public void paint(Graphics g) {
            int newW = 0, newH = 0;
            imgWidth = img.getWidth(this);                            //获取图片宽度
            imgHeight = img.getHeight(this);                          //获取图片高度
            float value = jSlider.getValue();                         //滑块组件的取值
            newW = (int) (imgWidth * value / 100);                    //计算图片放大后的宽度
            newH = (int) (imgHeight * value / 100);                   //计算图片放大后的高度
            g.drawImage(img, 0, 0, newW, newH, this);                 //绘制指定大小的图片
        }
    }
}
```

运行结果如图 20.14 所示。

图 20.14　图像缩放效果

说明

repaint()方法将调用 paint()方法，实现组件或画板的重画功能，类似于界面刷新。

20.4.3　图片翻转

图像的翻转需要使用 drawImage()方法的另一个重载方法。语法格式如下：

```
drawImage(Image img, int dx1, int dy1, int dx2, int dy2, int sx1, int sy1, int sx2, int sy2, ImageObserver observer)
```

此方法总是用非缩放的图像来呈现缩放的矩形，并且动态地执行所需的缩放。此操作不使用缓存的缩放图像。执行图像从源到目标的缩放：源矩形的第一个坐标被映射到目标矩形的第一个坐标，第二个源坐标被映射到第二个目标坐标。按需要缩放和翻转子图像以保持这些映射关系。方法中涉及的参数说明如表 20.6 所示。

表 20.6　参数说明

参　数　名	说　　明
img	要绘制的指定图像
dx1	目标矩形第一个坐标的 x 位置
dy1	目标矩形第一个坐标的 y 位置
dx2	目标矩形第二个坐标的 x 位置
dy2	目标矩形第二个坐标的 y 位置
sx1	源矩形第一个坐标的 x 位置
sy1	源矩形第一个坐标的 y 位置
sx2	源矩形第二个坐标的 x 位置
sy2	源矩形第二个坐标的 y 位置
observer	要通知的图像观察者

【例 20.7】　在窗体界面中绘制图像的翻转效果。程序中 drawImage()方法使用的参数名称与语法中介绍的相同，MyCanvas 类只是在 paint()方法中按照参数顺序执行 drawImage()方法，图片的翻转由控制按钮变换参数值，然后执行 MyCanvas 类的 repaint()方法实现。（**实例位置：光盘\TM\sl\20\12**）

```
package com.lzw;
import java.awt.*;
import java.net.URL;
import javax.swing.*;
public class PartImage extends JFrame {
    private Image img;
    private int dx1,dy1,dx2,dy2;
    private int sx1,sy1,sx2,sy2;
    ...//省略部分代码
    private MyCanvas canvasPanel = null;
    public PartImage() {
        dx2=sx2=300;                                    //初始化图像大小
        dy2=sy2=200;
        initialize();                                   //调用初始化方法
    }
    ...//省略部分代码
    //获取“水平翻转”按钮
    private JButton getJButton() {
        if (jButton == null) {
            jButton = new JButton();
            jButton.setText("水平翻转");
            jButton.addActionListener(new java.awt.event.ActionListener() {
                public void actionPerformed(java.awt.event.ActionEvent e) {
                    sx1=Math.abs(sx1-300);              //改变源矩形两个坐标的 x 位置
                    sx2=Math.abs(sx2-300);
                    canvasPanel.repaint();
                }
            });
        }
        return jButton;
    }
    //获取“垂直翻转”按钮
    private JButton getJButton1() {
        if (jButton1 == null) {
            jButton1 = new JButton();
            jButton1.setText("垂直翻转");
            jButton1.addActionListener(new java.awt.event.ActionListener() {
                public void actionPerformed(java.awt.event.ActionEvent e) {
                    sy1=Math.abs(sy1-200);              //改变源矩形两个坐标的 y 位置
                    sy2=Math.abs(sy2-200);
                    canvasPanel.repaint();
                }
            });
        }
        return jButton1;
    }
    ...//省略部分代码
    //画板
    class MyCanvas extends JPanel {
        public void paint(Graphics g) {
```

```
            g.drawImage(img, dx1, dy1, dx2, dy2,sx1,sy1,sx2,sy2, this); //绘制指定大小的图片
        }
    }
}
```

运行结果如图 20.15 所示。

图 20.15　源图、水平翻转和垂直翻转的效果

20.4.4　图片旋转

图像的旋转需要调用 Graphics2D 类的 rotate()方法，该方法将根据指定的弧度旋转图像。语法格式如下：

```
rotate(double theta)
```

theta：旋转的弧度。

说明　该方法只接受旋转的弧度作参数，可以使用 Math 类的 toRadians()方法将角度转换为弧度。toRadians()方法接受角度值作参数，返回值是转换完毕的弧度值。

【例 20.8】　在主窗体中绘制 3 个旋转后的图像，每个图像旋转角度值为 5。（**实例位置：光盘\TM\sl\20\13**）

```
package com.lzw;
import java.awt.*;
import java.net.URL;
import javax.swing.*;
public class RotateImage extends JFrame {
    private Image img;
    private MyCanvas canvasPanel = null;
    public RotateImage() {
        initialize();                                          //调用初始化方法
    }
    //界面初始化方法
    private void initialize() {
        URL imgUrl = RotateImage.class.getResource("cow.jpg");  //获取图片资源的路径
        img = Toolkit.getDefaultToolkit().getImage(imgUrl);     //获取图片资源
```

```
        canvasPanel = new MyCanvas();
        this.setBounds(100, 100, 400, 350);                    //设置窗体大小和位置
        add(canvasPanel);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);        //设置窗体关闭模式
        this.setTitle("图片旋转");                              //设置窗体标题
    }
    //主方法
    public static void main(String[] args) {
        new RotateImage().setVisible(true);
    }
    //画板
    class MyCanvas extends JPanel {
        public void paint(Graphics g) {
            Graphics2D g2=(Graphics2D) g;
            g2.rotate(Math.toRadians(5));
            g2.drawImage(img, 70, 10, 300, 200, this);         //绘制指定大小的图片
            g2.rotate(Math.toRadians(5));
            g2.drawImage(img, 70, 10, 300, 200, this);         //绘制指定大小的图片
            g2.rotate(Math.toRadians(5));
            g2.drawImage(img, 70, 10, 300, 200, this);         //绘制指定大小的图片
        }
    }
}
```

运行结果如图 20.16 所示。

图 20.16　图像旋转效果

20.4.5　图片倾斜

可以使用 Graphics2D 类提供的 shear()方法设置绘图的倾斜方向，从而实现使图像倾斜的效果。语法格式如下：

```
shear(double shx, double shy)
```

- ☑ shx：水平方向的倾斜量。
- ☑ shy：垂直方向的倾斜量。

【例 20.9】 在窗体上绘制图像，使图像在水平方向实现倾斜效果。（实例位置：光盘\TM\sl\20\14）

```
package com.lzw;
import java.awt.*;
import java.net.URL;
import javax.swing.*;
public class TiltImage extends JFrame {
    private Image img;
    private MyCanvas canvasPanel = null;
    public TiltImage() {
        initialize();                                          //调用初始化方法
    }
    //界面初始化方法
    private void initialize() {
        URL imgUrl = TiltImage.class.getResource("cow.jpg");   //获取图片资源的路径
        img = Toolkit.getDefaultToolkit().getImage(imgUrl);    //获取图片资源
        canvasPanel = new MyCanvas();
        this.setBounds(100, 100, 400, 300);                    //设置窗体大小和位置
        add(canvasPanel);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);        //设置窗体关闭模式
        this.setTitle("图片倾斜");                              //设置窗体标题
    }
    //主方法
    public static void main(String[] args) {
        new TiltImage().setVisible(true);
    }
    //画板
    class MyCanvas extends JPanel {
        public void paint(Graphics g) {
            Graphics2D g2=(Graphics2D) g;
            g2.shear(0.3, 0);
            g2.drawImage(img, 0, 0, 300, 200, this);           //绘制指定大小的图片
        }
    }
}
```

运行结果如图 20.17 所示。

图 20.17 水平倾斜的图片效果

20.4.6　范例 7：图形的交运算

在 Java 中如何实现图形的交运算，即保留两个图形的交集。编写 Java 程序并运行，将在窗体上显示进行交运算后的图形。运行结果如图 20.18 所示。（**实例位置：光盘\TM\sl\20\15**）

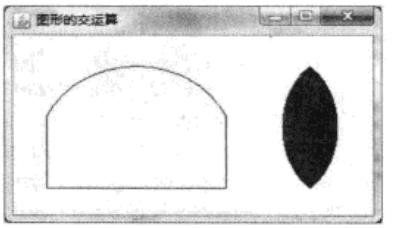


图 20.18　图形进行交运算的效果

（1）在项目中创建一个继承 JFrame 类的 IntersectOperationFrame 窗体类。

（2）在 IntersectOperationFrame 窗体类中创建内部面板类 IntersectOperationPanel，并重写 JComponent 类的 paint()方法，在该方法中实现图形的交运算。

（3）将内部面板类 IntersectOperationPanel 的实例添加到窗体类 IntersectOperationFrame 的内容面板上，用于在窗体上显示图形进行交运算后的效果。代码如下：

```
class IntersectOperationPanel extends JPanel {                          //创建内部面板类
    public void paint(Graphics g) {                                     //重写 paint()方法
        Graphics2D g2 = (Graphics2D)g;                                  //获得 Graphics2D 对象
        Rectangle2D.Float rect = new Rectangle2D.Float(30, 30, 160, 120);  //创建矩形对象
        Ellipse2D.Float ellipse = new Ellipse2D.Float(20, 30, 180, 180);   //创建圆对象
        Area area1 = new Area(rect);                                    //创建区域矩形
        Area area2 = new Area(ellipse);                                 //创建区域圆
        area1.intersect(area2);                                         //两个区域图形交运算
        g2.draw(area1);                                                 //绘制交运算的区域图形
        Ellipse2D.Float ellipse1 = new Ellipse2D.Float(190, 20, 100, 140);  //创建椭圆对象
        Ellipse2D.Float ellipse2 = new Ellipse2D.Float(240, 20, 100, 140);  //创建椭圆对象
        Area area3 = new Area(ellipse1);                                //创建区域椭圆
        Area area4 = new Area(ellipse2);                                //创建区域椭圆
        area3.intersect(area4);                                         //两个区域椭圆交运算
        g2.fill(area3);                                                 //绘制交运算的区域椭圆
    }
}
```

20.4.7　范例 8：图形的异或运算

在 Java 中如何实现图形的异或运算，即两个图形去除交集后剩下的部分。编写 Java 程序并运行，将在窗体上显示进行异或运算后的图形。运行结果如图 20.19 所示。（**实例位置：光盘\TM\sl\20\16**）

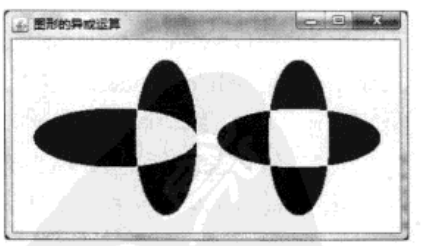


图 20.19　图形进行异或运算的效果

（1）在项目中创建一个继承 JFrame 类的 ExclusiveOrOperationFrame 窗体类。

（2）在 ExclusiveOrOperationFrame 窗体类中创建内部面板类 ExclusiveOrOperationPanel，并重写 JComponent 类的 paint()方法，在该方法中实现图形的异或运算。

（3）将内部面板类 ExclusiveOrOperationPanel 的实例添加到窗体类 ExclusiveOrOperationFrame 的内容面板上，用于在窗体上显示图形进行异或运算后的效果。代码如下：

```
class ExclusiveOrOperationPanel extends JPanel {                          //创建内部面板类
    public void paint(Graphics g) {                                       //重写 paint()方法
        Graphics2D g2 = (Graphics2D)g;                                    //获得 Graphics2D 对象
        Ellipse2D.Float ellipse1 = new Ellipse2D.Float(20, 70, 160, 60);  //创建椭圆对象
        Ellipse2D.Float ellipse2 = new Ellipse2D.Float(120, 20, 60, 160); //创建椭圆对象
        Area area1 = new Area(ellipse1);                                  //创建区域椭圆
        Area area2 = new Area(ellipse2);                                  //创建区域椭圆
        area1.exclusiveOr(area2);                                         //两个区域椭圆进行异或运算
        g2.fill(area1);                                                   //绘制异或运算后的区域椭圆
        Ellipse2D.Float ellipse3 = new Ellipse2D.Float(200, 70, 160, 60); //创建椭圆对象
        Ellipse2D.Float ellipse4 = new Ellipse2D.Float(250, 20, 60, 160); //创建椭圆对象
        Area area3 = new Area(ellipse3);                                  //创建区域椭圆
        Area area4 = new Area(ellipse4);                                  //创建区域椭圆
        area3.exclusiveOr(area4);                                         //两个区域椭圆进行异或运算
        g2.fill(area3);                                                   //绘制异或运算后的区域椭圆
    }
}
```

20.5 经典范例

20.5.1 经典范例 1：绘制花瓣

视频讲解：光盘\TM\lx\20\绘制花瓣.exe

本范例使用坐标轴平移和图形旋转等技术绘制花瓣。编写 Java 程序并运行，将在窗体上绘制花瓣。运行结果如图 20.20 所示。（**实例位置：光盘\TM\sl\20\17**）

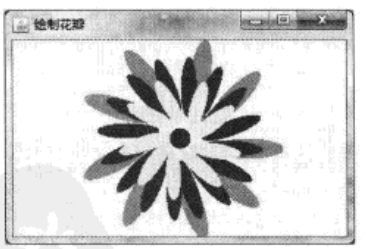


图 20.20 绘制花瓣

（1）在项目中创建一个继承 JFrame 类的 DrawFlowerFrame 窗体类。

（2）在 DrawFlowerFrame 窗体类中创建内部面板类 DrawFlowerPanel，并重写 JComponent 类的 paint()方法，在该方法中实现花瓣的绘制。

（3）将内部面板类 DrawFlowerPanel 的实例添加到窗体类 DrawFlowerFrame 的内容面板上，用于在窗体上显示绘制的花瓣。代码如下：

```
class DrawFlowerPanel extends JPanel {                                    //创建内部面板类
    public void paint(Graphics g) {                                       //重写 paint()方法
        Graphics2D g2 = (Graphics2D)g;                                    //获得 Graphics2D 对象
```

```
        //平移坐标轴
        g2.translate(drawFlowerPanel.getWidth() / 2, drawFlowerPanel.getHeight() / 2);
         //绘制绿色花瓣
        Ellipse2D.Float ellipse = new Ellipse2D.Float(30, 0, 70, 20);          //创建椭圆对象
        Color color = new Color(0,255,0);                                      //创建颜色对象
        g2.setColor(color);                                                    //指定颜色
        g2.fill(ellipse);                                                      //绘制椭圆
        int i=0;
        while (i<8){
            g2.rotate(30);                                                     //旋转画布
            g2.fill(ellipse);                                                  //绘制椭圆
            i++;
        }
        //绘制红色花瓣
        ellipse = new Ellipse2D.Float(20, 0, 60, 15);                          //创建椭圆对象
        color = new Color(255,0,0);                                            //创建颜色对象
        g2.setColor(color);                                                    //指定颜色
        g2.fill(ellipse);                                                      //绘制椭圆
        i=0;
        while (i<15){
            g2.rotate(75);                                                     //旋转画布
            g2.fill(ellipse);                                                  //绘制椭圆
            i++;
        }
        //绘制黄色花瓣
        ellipse = new Ellipse2D.Float(10, 0, 50, 15);                          //创建椭圆对象
        color = new Color(255,255,0);                                          //创建颜色对象
        g2.setColor(color);                                                    //指定颜色
        g2.fill(ellipse);                                                      //绘制椭圆
        i=0;
        while (i<8){
            g2.rotate(30);                                                     //旋转画布
            g2.fill(ellipse);                                                  //绘制椭圆
            i++;
        }
        //绘制红色中心点
        color = new Color(255, 0, 0);                                          //创建颜色对象
        g2.setColor(color);                                                    //指定颜色
        ellipse = new Ellipse2D.Float(-10, -10, 20, 20);                       //创建椭圆对象
        g2.fill(ellipse);                                                      //绘制椭圆
    }
}
```

20.5.2　经典范例 2：绘制艺术图案

视频讲解：光盘\TM\lx\20\绘制艺术图案.exe

本范例使用坐标轴平移、图形旋转和获得随机数等技术绘制艺术图案。编写 Java 程序并运行，将在窗体上绘制艺术图案。运行结果如图 20.21 所示。（**实例位置：光盘\TM\sl\20\18**）

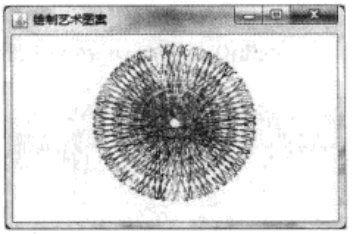


图 20.21　艺术图案

（1）在项目中创建一个继承 JFrame 类的 ArtDesignFrame 窗体类。

（2）在 ArtDesignFrame 窗体类中创建内部面板类 ArtDesignPanel，并重写 JComponent 类的 paint() 方法，在该方法中实现艺术图案的绘制。

（3）将内部面板类 ArtDesignPanel 的实例添加到窗体类 ArtDesignFrame 的内容面板上，用于在窗体上显示艺术图案。代码如下：

```
class ArtDesignPanel extends JPanel {                                  //创建内部面板类
    public void paint(Graphics g) {                                    //重写 paint()方法
        Graphics2D g2 = (Graphics2D)g;                                 //获得 Graphics2D 对象
        Ellipse2D.Float ellipse = new Ellipse2D.Float(-80, 5, 160, 10); //创建椭圆对象
        Random random = new Random();                                  //创建随机数对象
        g2.translate(160, 90);                                         //平移坐标轴
        int R = random.nextInt(256);                                   //随机产生颜色的 R 值
        int G = random.nextInt(256);                                   //随机产生颜色的 G 值
        int B = random.nextInt(256);                                   //随机产生颜色的 B 值
        Color color = new Color(R,G,B);                                //创建颜色对象
        g2.setColor(color);                                            //指定颜色
        g2.draw(ellipse);                                              //绘制椭圆
        int i=0;
        while (i<100){
            R = random.nextInt(256);                                   //随机产生颜色的 R 值
            G = random.nextInt(256);                                   //随机产生颜色的 G 值
            B = random.nextInt(256);                                   //随机产生颜色的 B 值
            color = new Color(R,G,B);                                  //创建新的颜色对象
            g2.setColor(color);                                        //指定颜色
            g2.rotate(10);                                             //旋转画布
            g2.draw(ellipse);                                          //绘制椭圆
            i++;
        }
    }
}
```

20.6 本章小结

本章主要讲解了 Java 的绘图技术，是 java.awt 包所提供的功能。其中，有关绘图和图像处理技术的介绍比较详细，主要包括基本图形绘制、设置绘图颜色与笔画属性、绘制文本、绘制图片以及图像

的缩放、翻转、倾斜、旋转等图像处理技术。通过对本章的学习，读者应该能够掌握基本绘图技术和图像处理技术，在日后程序开发时，可以使用本章讲解的知识编写统计图表等功能。

20.7　实 战 练 习

1．创建一个主窗体，在窗体上分别绘制矩形、三角形、圆形和椭圆形。（**答案位置：光盘\TM\sl\20\19**）

2．使用不同的颜色、不同的笔画属性绘制五环图形，并在五环图下显示年、月、日，文字要求使用宋体，大小为 14。（**答案位置：光盘\TM\sl\20\20**）

3．尝试利用综合线程技术编写动画程序。（**答案位置：光盘\TM\sl\20\21**）

第4篇

项目篇

⏭ 第21章 酒店管理系统

本篇通过开发一个完整的酒店管理系统，运用软件工程的设计思想，让读者学习如何进行 Java 项目的实践开发。书中按照编写项目设计思路→数据库设计→公共模块设计→主窗体设计→用户登录窗口设计→开台签单工作区设计→结账工作区设计→后台管理工作区设计的过程，带领读者一步一步亲身体验开发项目的全过程。

第21章

酒店管理系统

（视频讲解：1小时21分钟）

随着软件技术的不断发展，应用软件已经遍及社会的各行各业，大到厂矿校企，小到餐饮洗浴，并且正在以其独特的优势，服务于社会的各行各业。将应用软件应用于现代的餐饮业，解决了传统记账、统计、核算方式既费时、费力，又容易出错的问题。通过使用酒店管理系统，可以快速完成营业记账工作，并且可以轻松地对营业额进行统计、核算，原来既费时又费力的工作，现在只需要轻点几下鼠标和键盘，就可以轻松完成，既提高了工作效率，又节省了人力资源，为餐饮企业的快速发展创造了巨大空间。

通过阅读本章，您可以：

- 熟悉酒店管理系统的软件结构和业务流程
- 掌握根据用户输入的部分内容快速获取商品的方法
- 掌握人性化控制商品数量的方法
- 掌握系统自动结账的实现方法
- 掌握 Swing 中鼠标或键盘事件的使用方法
- 掌握年、月、日下拉列表框联动保证日期合法性的方法
- 掌握通过正则表达式验证用户输入数据合法性的方法
- 掌握系统时钟的实现方法

21.1 概　　述

随着人民生活水平的不断提高，餐饮业在服务行业中的地位越来越重要，如何从激烈的竞争中脱颖而出，已经成为每位餐饮经营者在思考的问题。

经过多年的发展，对餐饮企业的管理已经逐渐由简单的人工管理，逐步进入到规范、科学管理的阶段。众所周知，在科学管理的具体实现方法中，最有效的工具就是应用管理软件进行管理。

以往的人工操作管理中存在着许多问题，例如：

- ☑ 人工计算账单容易出现错误。
- ☑ 收银工作中容易发生账单丢失。
- ☑ 客人具体消费信息难以查询。
- ☑ 无法对以往营业数据进行查询。

21.2 系 统 分 析

视频讲解：光盘\TM\lx\21\系统分析.exe

随着餐饮行业的迅速发展，现有的人工管理方式已不能满足管理者的需求，广大餐饮业经营者已经意识到使用计算机应用软件的重要性，决定在餐饮企业的经营管理上引入计算机应用软件管理系统。

根据餐饮行业的特点和实际情况，酒店管理系统应以餐饮业务为基础，突出前台管理，重视营业数据分析等功能，从专业角度出发，努力为餐饮管理者提供科学、有效的管理模式和数据分析功能。

开台点菜功能是酒店管理系统的最常用功能，也是酒店管理系统的主要功能之一，所以需要将该功能设计得更加人性化和智能化。例如在确定添加菜品时，既可以通过菜品编号确定，又可以通过菜品助记码确定，并且默认添加菜品的数量为一个。

自动结账功能也是酒店管理系统的最常用功能，同样是酒店管理系统的主要功能之一，用户只需要选中结账的台号，系统就会自动为选中的台号计算消费金额，并且用户输入实收金额后，系统还会自动计算出需要找零的金额，这样既节省了系统操作员的精力，又避免了由于计算失误造成的损失。

报表功能是酒店管理系统不可缺少的一部分，因为酒店管理系统是一个记账式软件，如果一个记账式软件没有报表功能，就好比一个双目失明的狮子，已经失去了生存的能力，对于一个餐饮企业，日报表是不可缺少的。

再就是系统安全和系统维护功能，这是应用软件必不可少的一部分，用来保障软件的安全运行。

21.3 系 统 设 计

视频讲解：光盘\TM\lx\21\系统设计.exe

21.3.1 系统目标

依据餐饮行业的特点，本系统需要实现以下目标：

- ☑ 操作简单方便、界面简洁大方。
- ☑ 方便、快捷的开台点菜功能。
- ☑ 智能化定位菜品的功能。
- ☑ 快速查看开台点菜信息的功能。
- ☑ 自动结账功能。
- ☑ 按开台和商品实现的日结账功能。
- ☑ 按日消费额汇总统计实现的月结账功能。
- ☑ 按日营业额实现的年结账功能。
- ☑ 系统运行稳定、安全可靠。

21.3.2　系统功能结构

酒店管理系统的功能结构如图 21.1 所示。

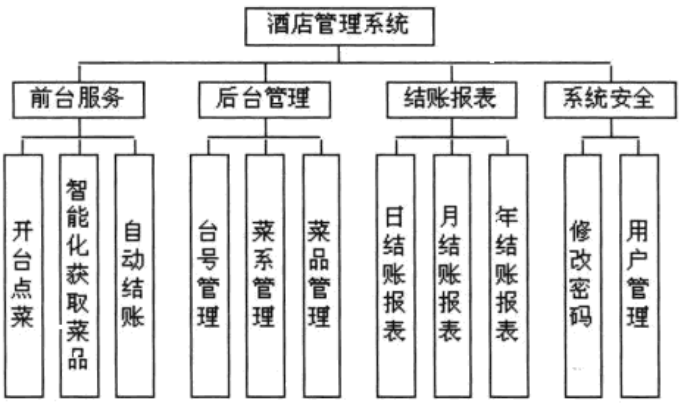


图 21.1　酒店管理系统功能结构

21.3.3　系统预览

酒店管理系统由多个程序界面组成，下面仅列出几个典型界面。

酒店管理系统的主窗体效果如图 21.2 所示，窗体的中间部分用来显示当前的开台及点菜信息，窗体的下方用来操作该系统，如开台点菜，自动结账，台号、菜系和菜品的维护，营业额报表等。

单击图 21.2 中右下方的“台号管理”按钮，将打开如图 21.3 所示的“台号管理”对话框，用来维护台号信息，包括台号及座位数。

单击图 21.2 中右下方的“菜品管理”按钮，将打开如图 21.4 所示的“菜品管理”对话框，用来维护菜品信息，包括名称、助记码、菜系、单位和单价，其中助记码用来在点菜时快速获取菜品信息（建议设置为菜品名称的首字母，如将菜品“雪蔓火焰山”的助记码设置为 xmhys）。

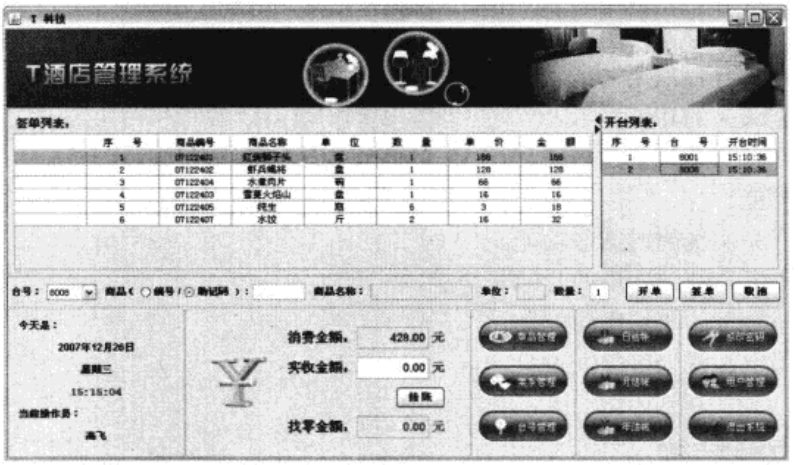


图 21.2　酒店管理系统主窗体效果

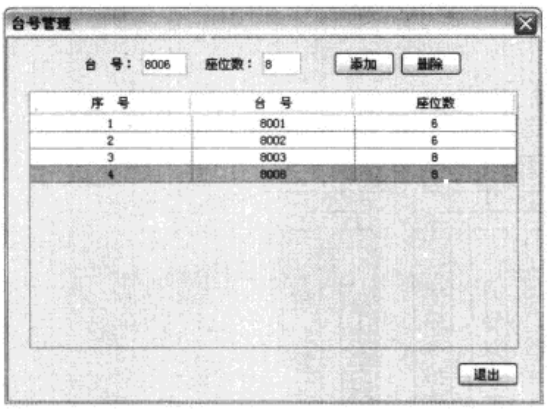


图 21.3　台号管理

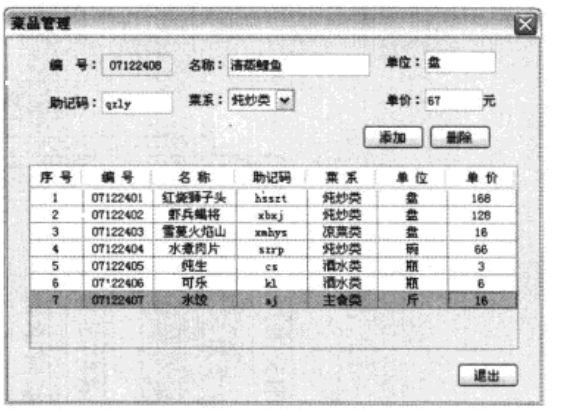


图 21.4　菜品管理

单击图 21.2 中右下方的“日结账”按钮，将打开如图 21.5 所示的“日结账”对话框，用来统计指定日期的销售情况，包括总日营业额和各个商品的日营业额。

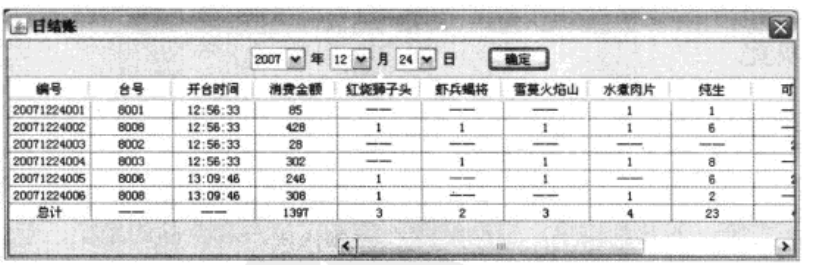


图 21.5　日结账窗体效果

21.3.4　业务流程图

酒店管理系统的业务流程图如图 21.6 所示。

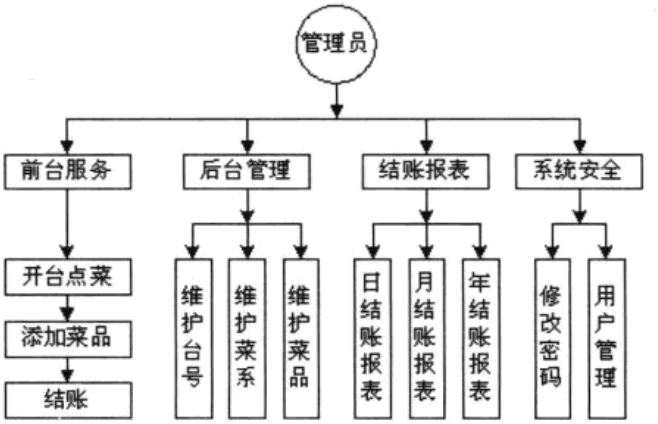


图 21.6　酒店管理系统的业务流程图

21.3.5　文件夹结构设计

每个项目都会有相应的文件夹组织结构。当项目中的窗体过多时，为了便于查找和使用，可以将窗体进行分类，放入不同的文件夹中，这样既便于前期开发，又便于后期维护。酒店管理系统文件夹组织结构如图 21.7 所示。

图 21.7　酒店管理系统文件夹结构

21.4 数据库设计

视频讲解：光盘\TM\lx\21\数据库设计.exe

在开发应用程序时，对数据库的操作是必不可少的，而一个数据库的设计优秀与否，将直接影响到软件的开发进度和性能，所以对数据库的设计就显得尤为重要。数据库的设计要根据程序的需求及其功能制定，如果在开发软件之前不能很好地设计数据库，在开发过程中将反复修改数据库，必将严重影响开发进度。

21.4.1 数据库分析

酒店管理系统的需求包括开台点菜功能、智能化获取菜品功能、自动结账功能、营业额报表功能等。在这些功能中主要涉及的数据表包括台号表、菜品表、消费单表；为了使系统更完善，还需要为菜品分类，即需要用到菜系表；为了实现菜品的日销售情况统计，还要建立一个消费项目表，用来记录消费单消费的菜品。

21.4.2 数据库概念设计

数据库设计是系统设计过程中的重要组成部分，它是通过管理系统的整体需求而制定的，数据库设计的好坏直接影响到系统的后期开发。下面对本系统中具有代表性的数据库设计进行详细说明。

餐台和菜系在本系统中是最简单的实体，在本系统中用来描述餐台信息的只有台号和座位数，而描述菜系的主要是名称。餐台信息表的 E-R 图如图 21.8 所示，菜系信息表的 E-R 图如图 21.9 所示。

图 21.8 餐台信息表 E-R 图

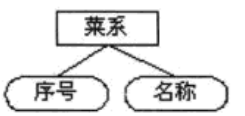


图 21.9 菜系信息表 E-R 图

在描述菜品实体时，加入了助记码，目的是为了实现智能化获取菜品功能，通过这一功能系统操作员可以快速地获取顾客所点的菜品信息。菜品信息表的 E-R 图如图 21.10 所示。

图 21.10 菜品信息表 E-R 图

消费单用来记录每次消费的相关信息，如消费时使用的餐台、消费时间、消费金额等。消费单信息表的 E-R 图如图 21.11 所示。

消费项目用来记录每个消费单消费的菜品，记录的主要信息有所属消费单、消费菜品、消费数量、

消费额。消费项目信息表的 E-R 图如图 21.12 所示。

图 21.11　消费单信息表 E-R 图

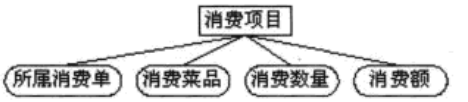


图 21.12　消费项目信息表 E-R 图

21.4.3　数据库逻辑结构设计

在数据库概念设计中已经分析了菜品、消费单、消费项目等主要的数据库实体对象，这些实体对象是数据表结构的基本模型，最终的数据模型都要实施到数据库中，形成整体的数据结构，可以使用 PowerDesigner 工具完成这个数据库的建模。具体的模型结构如图 21.13 所示。

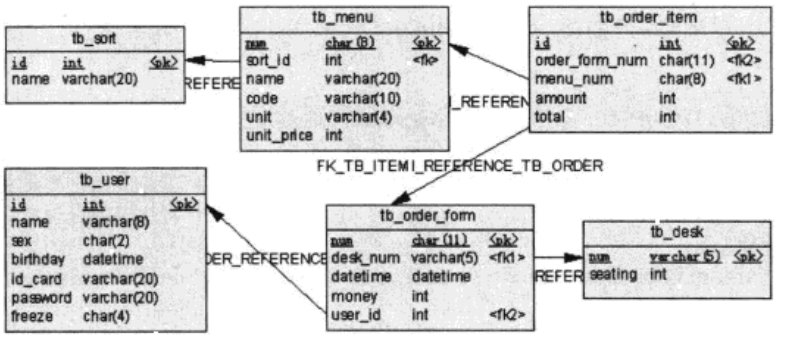


图 21.13　酒店管理系统数据库模型

21.4.4　视图设计

完成数据库建模后，还可以根据实际需要建立一些视图，通过对视图的应用，可以减少在程序中编写复杂的 SQL 语句。在开发酒店管理系统的日结账功能时，需要查询指定日期的所有消费单，然后根据消费单查询消费项目并关联查询项目名称，所以为表 tb_menu 和表 tb_order_item 建立了一个视图 v_order_item_and_menu，如图 21.14 所示。

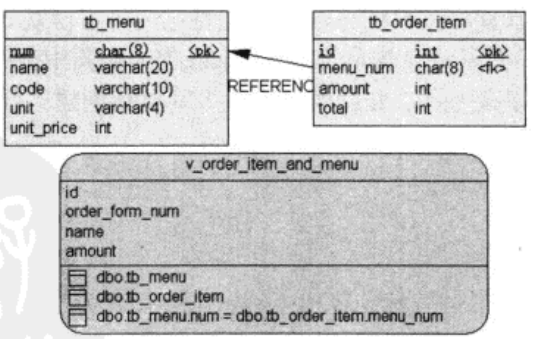


图 21.14　酒店管理系统用到的视图

21.5 公共模块设计

视频讲解：光盘\TM\lx\21\公共模块设计.exe

21.5.1 编写数据库连接类

数据库连接类负责加载数据库驱动程序，以及创建和关闭数据库连接，为了最大程度地应用每个已经创建的数据库连接，这里将其保存到了 ThreadLocal 类的对象中。

首先在数据库连接类中定义一些常量，包括连接数据库使用的驱动程序、连接数据库的路径、连接数据库使用的用户名和密码，并且定义一个 ThreadLocal 类的对象，用来保存已经创建的数据库连接。代码如下：

```
private static final String DRIVERCLASS = "com.microsoft.jdbc.sqlserver.SQLServerDriver";//数据库驱动路径
private static final String URL ="jdbc:microsoft:sqlserver://localhost:1433;DatabaseName=db_DrinkeryManage";
private static final String USERNAME = "sa";                    //连接数据库的用户名
private static final String PASSWORD = "";                      //连接数据库的密码
//创建保存连接的对象
private static final ThreadLocal<Connection> threadLocal = new ThreadLocal<Connection>();
```

然后编写用来加载数据库驱动程序的代码，通常情况下将其放到静态代码块中，这样做的好处是只在该类第一次被加载（即第一次被调用）时执行加载数据库驱动程序的动作，避免了反复加载数据库驱动程序，从而提高软件的性能。代码如下：

```
static {                                                        //通过静态方法加载数据库驱动
    try {
        Class.forName(DRIVERCLASS).newInstance();               //加载数据库驱动
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

最后编写用来创建和关闭数据库连接的方法，这里将这两个方法均定义为静态的，这样类名即可调用方法，方便使用。在这两个方法中首先从 ThreadLocal 类的对象中获得数据库连接，然后判断是否存在可用的数据库连接，如果存在则直接返回或关闭，否则重新创建。代码如下：

```
public static Connection getConnection() {                      //创建数据库连接的方法
    Connection conn = threadLocal.get();                        //从线程中获得数据库连接
    if (conn == null) {                                         //没有可用的数据库连接
        try {
            //创建新的数据库连接
            conn = DriverManager.getConnection(URL, USERNAME, PASSWORD);
            threadLocal.set(conn);                              //将数据库连接保存到线程中
        } catch (SQLException e) {
```

```
                e.printStackTrace();
            }
        }
        return conn;
    }
    public static boolean closeConnection() {                    //关闭数据库连接的方法
        boolean isClosed = true;                                 //默认关闭成功
        Connection conn = threadLocal.get();                     //从线程中获得数据库连接
        threadLocal.set(null);                                   //清空线程中的数据库连接
        if (conn != null) {                                      //数据库连接可用
            try {
                conn.close();                                    //关闭数据库连接
            } catch (SQLException e) {
                isClosed = false;                                //关闭失败
                e.printStackTrace();
            }
        }
        return isClosed;
    }
```

21.5.2　封装常用的操作数据库的方法

对数据库的操作包括查询、添加、修改和删除，其中查询是通过 executeQuery(String sql)方法执行 SQL 语句实现的，添加、修改和删除是通过 executeUpdate(String sql)方法执行 SQL 语句实现的。在本系统中共提供了 4 个用来执行查询的方法，分别用来查询多个记录、查询指定记录、查询多个记录的指定值和查询指定记录的指定值。由于篇幅有限，在这里只介绍用来查询多个记录的方法，以及用来添加、修改和删除记录的方法。

下面的方法用来查询多个记录。为了将检索结果直接应用于表格组件，这里全部用 Vector 向量对象封装查询结果，并且为代表每一行的向量添加了行序号。代码如下：

```
protected Vector selectSomeNote(String sql) {
    Vector<Vector<Object>> vector = new Vector<Vector<Object>>();      //创建结果集向量
    Connection conn = JDBC.getConnection();                            //获得数据库连接
    try {
        Statement stmt = conn.createStatement();                       //创建连接状态对象
        ResultSet rs = stmt.executeQuery(sql);                         //执行 SQL 语句获得查询结果
        int columnCount = rs.getMetaData().getColumnCount();           //获得查询数据表的列数
        int row = 1;                                                   //定义行序号
        while (rs.next()) {                                            //遍历结果集
            Vector<Object> rowV = new Vector<Object>();                //创建行向量
            rowV.add(new Integer(row++));                              //添加行序号
            for (int column = 1; column <= columnCount; column++) {
                rowV.add(rs.getObject(column));                        //添加列值
            }
            vector.add(rowV);                                          //将行向量添加到结果集向量中
        }
        rs.close();                                                    //关闭结果集对象
```

```
        stmt.close();                                          //关闭连接状态对象
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return vector;                                             //返回结果集向量
}
```

下面的方法用来添加、修改和删除记录。这里采用手动提交，目的是捕获持久化异常，并回退数据库，以保证数据的合法性。代码如下：

```
public boolean longHaul(String sql) {
    boolean isLongHaul = true;                                 //默认持久化成功
    Connection conn = JDBC.getConnection();                    //获得数据库连接
    try {
        conn.setAutoCommit(false);                             //设置为手动提交
        Statement stmt = conn.createStatement();               //创建连接状态对象
        stmt.executeUpdate(sql);                               //执行 SQL 语句
        stmt.close();                                          //关闭连接状态对象
        conn.commit();                                         //提交持久化
    } catch (SQLException e) {
        isLongHaul = false;                                    //持久化失败
        try {
            conn.rollback();                                   //回退
        } catch (SQLException e1) {
            e1.printStackTrace();
        }
        e.printStackTrace();
    }
    return isLongHaul;                                         //返回持久化结果
}
```

21.5.3 自定义表格组件

在使用表格时，通常情况下将表格列的值设置为居中显示，这样看起来更美观。由 JTable 组件实现的表格将一个表格分为了两个部分，一部分是表格头，即用来显示表格列名的部分；另一部分是除表格头之外的部分。表格的每一部分对应一个 DefaultTableCellRenderer 类的对象，即单元格对象，通过该对象可以设置所属表格部分包含单元格的相关属性，如设置单元格内容的显示位置。

通过 JTable 类的 getDefaultRenderer()方法可以得到单元格对象，这个单元格对象代表的是除表格头之外的部分包含的单元格，在 MTable 类中对该方法进行了简单的重构，即通过单元格对象设置单元格内容为居中显示。代码如下：

```
public TableCellRenderer getDefaultRenderer(Class<?> columnClass) {
    //获得除表格头之外部分的单元格对象
❶   DefaultTableCellRenderer tableRenderer = (DefaultTableCellRenderer)
super.getDefaultRenderer(columnClass);
```

```
    //设置单元格内容居中显示
❷   tableRenderer.setHorizontalAlignment(DefaultTableCellRenderer.CENTER);
    return tableRenderer;
}
```

☑ ❶ getDefaultRenderer(columnClass)：该方法用来获得表格的单元格对象。

☑ ❷ setHorizontalAlignment(int alignment)：该方法用来设置单元格内容的显示位置，可选的静态常量有 LEFT（靠左侧显示）、CENTER（居中显示）和 RIGHT（靠右侧显示）。

如果想设置表格头的相关绘制属性，首先要获得表格头对象，即 JTableHeader 类的对象，通过 JTableHeader 类的 setReorderingAllowed(boolean reorderingAllowed)方法可以设置表格列是否可以重排；通过 JTableHeader 类的 getDefaultRenderer()方法也可以得到单元格对象，这个单元格对象代表的是表格头包含的单元格，在 MTable 类中对该方法也进行了简单的重构，即通过单元格对象设置单元格内容（即列名）为居中显示。代码如下：

```
public JTableHeader getTableHeader() {
    //获得表格头对象
    JTableHeader tableHeader = super.getTableHeader();
❶   tableHeader.setReorderingAllowed(false);                    //设置表格列不可重排
    //获得表格头的单元格对象
❷   DefaultTableCellRenderer headerRenderer = (DefaultTableCellRenderer)
tableHeader.getDefaultRenderer();
    //设置单元格内容（即列名）居中显示
    headerRenderer.setHorizontalAlignment(DefaultTableCellRenderer.CENTER);
    return tableHeader;
}
```

☑ ❶ setReorderingAllowed(boolean reorderingAllowed)：该方法用来设置表格头是否可以重新排列，即表格的列是否可以移动，默认为 true，即默认可以移动。

☑ ❷ getDefaultRenderer()：该方法用来获得表格头（即显示列名的位置）的单元格对象。

JTable 类的 setRowSelectionInterval(int fromRow, int toRow)方法用来设置表格中选中的行，第一个参数为开始选中行的索引，第二个参数为结束选中行的索引。在本系统中表格的选择模式为单选，如果通过该方法设置，需要设置两个参数，所以在 MTable 类中实现了一个重载方法 setRowSelectionInterval (int row)，用来设置唯一被选中的行，该方法仍然需要调用前面的方法。代码如下：

```
public void setRowSelectionInterval(int row) {              //重载方法
    setRowSelectionInterval(row, row);
}
```

21.5.4　编写利用正则表达式验证数据合法性的方法

在获得用户的输入内容时，经常需要验证用户输入数据的合法性，例如对用户输入日期的验证。验证这一类数据时，较好的办法是利用正则表达式，所以本系统提供了一个可重用的利用正则表达式验证数据合法性的方法，每次使用时只需要传入验证规则和验证内容，返回值为 boolean 型，当返回 true

时表示验证通过，数据合法；当返回 false 时表示验证未通过，数据不合法。代码如下：

```
public static boolean execute(String rule, String content) {
    Pattern pattern = Pattern.compile(rule);                    //利用验证规则创建 Pattern 对象
    Matcher matcher = pattern.matcher(content);                 //利用验证内容获得 Matcher 对象
    return matcher.matches();                                   //返回验证结果
}
```

21.6 主窗体设计

视频讲解：光盘\TM\lx\21\主窗体设计.exe

本系统将主窗体划分为 6 个工作区，分别是开台签单工作区、自动结账工作区、后台管理工作区、结账报表工作区、系统安全工作区和系统提示区，酒店管理系统主窗体效果如图 21.15 所示。

图 21.15 酒店管理系统主窗体效果

在开台签单工作区中使用了分割面板，这样系统操作员可以根据实际需要，调整开台列表和签单列表的大小，并设置分割面板支持快速展开/折叠的分割条，效果如图 21.16 所示。既可以将光标移动到分割条的上方随意调整分割条的位置，又可以通过单击分割条上的◀或▶按钮将分割条移动到分割面板的最左侧或最右侧，单击另一个则使分割条恢复到原位置；不支持快速展开/折叠的分割条效果如图 21.17 所示。

图 21.16 支持快速展开/折叠的分割条

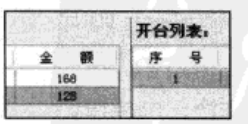


图 21.17 不支持快速展开/折叠的分割条

实现图 21.16 所示分割面板的关键代码如下：

```
final JSplitPane splitPane = new JSplitPane();                    //创建分割面板对象
❶    splitPane.setOrientation(JSplitPane.HORIZONTAL_SPLIT);       //设置为水平分割
❷    splitPane.setDividerLocation(755);                           //设置面板默认的分割位置
❸    splitPane.setDividerSize(10);                                //设置分割条的宽度
❹    splitPane.setOneTouchExpandable(true);                       //设置为支持快速展开/折叠分割条
splitPane.setBorder(new TitledBorder(null, "",
            TitledBorder.DEFAULT_JUSTIFICATION,
            TitledBorder.DEFAULT_POSITION, null, null));          //设置面板的边框
getContentPane().add(splitPane, BorderLayout.CENTER);             //将分割面板添加到上级容器中
final JPanel leftPanel = new JPanel();                            //创建放于分割面板左侧的普通面板对象
❺    splitPane.setLeftComponent(leftPanel);                       //将普通面板对象添加到分割面板的左侧
...//此处省略了部分代码
final JPanel rightPanel = new JPanel();                           //创建放于分割面板右侧的普通面板对象
❻    splitPane.setRightComponent(rightPanel);                     //将普通面板对象添加到分割面板的右侧
```

- ☑ ❶ setOrientation(int orientation)：该方法用于设置分割面板的分割方向，默认为在水平方向分割，即 HORIZONTAL_SPLIT；如果希望在垂直方向分割，则需要将参数设置为 VERTICAL_SPLIT。
- ☑ ❷ setDividerLocation(int location)：该方法用于设置分割条的显示位置，入口参数为针对分割面板的绝对位置；该方法拥有一个重载方法 setDividerLocation(double proportionalLocation)，用来以百分比的形式设置分割条的显示位置，proportionalLocation 为 0～1.0 的双精度浮点值。
- ☑ ❸ setDividerSize (int width)：该方法用于设置分割条的宽度，默认宽度为 5 像素。
- ☑ ❹ setOneTouchExpandable(boolean expandable)：该方法用于设置是否支持快速展开/折叠的分割条，默认为 false，即不支持。需要注意的是，有些外观可能不支持该功能。
- ☑ ❺ setLeftComponent(Component comp)：该方法用来将指定的组件设置到分割面板的左侧或上方，等同于方法 setTopComponent(Component comp)。
- ☑ ❻ setRightComponent(Component comp)：该方法用来将指定的组件设置到分割面板的右侧或下方，等同于方法 setBottomComponent (Component comp)。

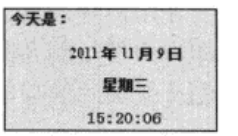


图 21.18　系统提示区的时钟

在系统提示区提供了时钟的功能，如图 21.18 所示。

系统提示区的时钟是显示到标签组件上的。对标签组件的具体设置代码如下：

```
timeLabel = new JLabel();                                         //创建用于显示时间的标签对象
timeLabel.setFont(new Font("宋体", Font.BOLD, 14));               //设置标签中的文字为宋体、粗体、14 号
timeLabel.setForeground(new Color(255, 0, 0));                    //设置标签中的文字为红色
timeLabel.setHorizontalAlignment(SwingConstants.CENTER);          //设置标签中的文字居中显示
clueOnPanel.add(timeLabel);                                       //将标签添加到上级容器中
new Time().start();                                               //开启线程
```

下面在 TipWizardFrame 类中创建一个内部类 Time，该类继承了线程类 Thread，并重构了 run()方法，每隔一秒修改一次用于显示时间的标签中的时间信息。代码如下：

```
class Time extends Thread {                                       //创建内部类
    public void run() {                                           //重构父类的方法
```

```
        while (true) {
            Date date = new Date();                                    //创建日期对象
            //获取当前时间，并显示到时间标签中
            timeLabel.setText(date.toString().substring(11, 19));
            try {
                Thread.sleep(1000);                                    //令线程休眠 1 秒
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

21.7 用户登录窗口设计

视频讲解：光盘\TM\lx\21\用户登录窗口设计.exe

用户登录窗口是每一个应用软件都不可缺少的部分，其主要功能是保证用户的数据安全；同时用户登录窗口也是用户看到的第一个系统界面，因此，一个设计优秀的用户登录窗口，将有效地提高用户对系统的第一印象。

用户登录窗口的设计优秀与否，主要包括以下几个方面：

- ☑ 美观大方。
- ☑ 简单易懂。
- ☑ 安全性高。
- ☑ 使用方便。

要使用户登录界面美观大方，就离不开对图片的使用，但是 JPanel 类并不支持绘制背景图片的功能，即将组件绘制到图片的上方，可以通过重写 JPanel 类的 paintComponent(Graphics g)方法，实现支持绘制背景图片的功能。代码如下：

```
public class MPanel extends JPanel {
    private ImageIcon imageIcon;                                   //声明一个图片对象
    public MPanel(URL imgUrl) {
        super();                                                   //继承父类的构造方法
        setLayout(new GridBagLayout());                            //将布局管理器修改为网格组布局
        imageIcon = new ImageIcon(imgUrl);                         //根据传入的 URL 创建 ImageIcon 对象
        setSize(imageIcon.getIconWidth(), imageIcon.getIconHeight());//设置面板与图片等大
    }
    protected void paintComponent(Graphics g) {                    //重写 JPanel 类的 paintComponent()方法
        super.paintComponent(g);                                   //调用 JPanel 类的 paintComponent()方法
        Image image = imageIcon.getImage();                        //通过 ImageIcon 对象获得 Image 对象
        g.drawImage(image, 0, 0, null);                            //绘制 Image 对象，即将图片绘制到面板中
    }
}
```

利用通过继承 JPanel 类得到的 MPanel 类，即可很方便地将图片设置为面板的背景图片。这样，在

设计用于用户登录界面的背景图片时，就可以将一些辅助信息设计到图片上了，系统为用户登录界面设计的背景图片如图 21.19 所示。

利用图 21.19 所示图片作为用户登录界面的背景图片，在绘制用户登录界面时，只需要利用代码绘制一个下拉列表框、一个密码框和 3 个按钮即可。绘制完成后的本系统登录界面效果如图 21.20 所示。

图 21.19　为用户登录界面设计的背景图片

图 21.20　酒店管理系统用户登录界面

首先创建一个用于用户登录界面的窗体，为窗体设置标题、大小等信息，并将一个支持背景图片功能的面板添加到窗体中，背景图片即图 21.19 所示的图片。代码如下：

```
public class LandFrame extends JFrame {
    private JPasswordField passwordField;                          //密码框
    private JComboBox usernameComboBox;                            //用户名下拉列表框
    public LandFrame() {
        //首先设置窗口的相关信息
        super();                                                   //调用父类的构造方法
        setTitle(" T 科技");                                       //设置窗口的标题
❶       setResizable(false);                                       //设置窗口不可以改变大小
❷       setAlwaysOnTop(true);                                      //设置窗口总在最前方
        setBounds(100, 100, 428, 292);                             //设置窗口的大小
❸       setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);            //设置当关闭窗口时执行的动作
        //下面将创建一个面板对象并添加到窗口的容器中
        final MPanel panel = new MPanel(this.getClass().getResource(
                "/img/land_background.jpg"));                      //创建一个面板对象
        panel.setLayout(new GridBagLayout());                      //设置面板的布局管理器为网格组布局
        getContentPane().add(panel, BorderLayout.CENTER);          //将面板添加到窗体中
        ...//此处省略了部分代码
    }
}
```

☑ ❶ setResizable(boolean resizable)：该方法用于设置窗口的大小是否可以调整，默认为 true，即在默认情况下可以调整；当设置为 false 时，窗口大小将不可以调整，并且“最大化”按钮将不可用。

☑ ❷ setAlwaysOnTop(Boolean alwaysOnTop)：该方法用于设置窗口是否永远在屏幕的最前方，默认为 false；当设置为 true 时，窗口将永远在屏幕的最前方。

☑ ❸ setDefaultCloseOperation(int operation)：该方法用于设置当用户关闭此窗体时执行的操作。必须为以下 4 个静态常量之一：DO_NOTHING_ON_CLOSE（常量值为 0）——不执行任何

操作；HIDE_ON_CLOSE（常量值为 1）——隐藏该窗体；DISPOSE_ON_CLOSE（常量值为 2）——释放该窗体；EXIT_ON_CLOSE（常量值为 3）——退出应用程序。默认值为静态常量 HIDE_ON_CLOSE。

下面将依次创建一个下拉列表框组件和一个密码框组件，并利用网格组布局管理器将它们添加到背景面板中，分别用于选择登录用户和输入登录密码。代码如下：

```
//创建并设置用户名下拉列表框
usernameComboBox = new JComboBox();                                    //创建用户名下拉列表框组件对象
usernameComboBox.setMaximumRowCount(5);                                //设置下拉列表框最多可显示的选项数
usernameComboBox.addItem("请选择");                                    //为下拉列表框添加提示项
usernameComboBox
        .addActionListener(new UsernameComboBoxActionListener());      //为下拉列表框添加事件监听器
final GridBagConstraints gridBagConstraints = new GridBagConstraints();  //创建网格组布局管理器对象
gridBagConstraints.anchor = GridBagConstraints.WEST;                   //设置为靠左侧显示
gridBagConstraints.gridy = 1;                                          //设置行索引为 1
gridBagConstraints.gridx = 2;                                          //设置列索引为 2
panel.add(usernameComboBox, gridBagConstraints);                       //将组件按指定的布局管理器添加到面板中
//创建并设置密码框
passwordField = new JPasswordField();                                  //创建密码框组件对象
passwordField.setColumns(20);                                          //设置密码框可显示的字符数
passwordField.setText("        ");                                     //设置密码框默认显示 6 个空格
passwordField.addFocusListener(new PasswordFieldFocusListener());      //为密码框添加焦点监听器
final GridBagConstraints gridBagConstraints_1 = new GridBagConstraints();  //创建网格组布局管理器对象
gridBagConstraints_1.insets = new Insets(5, 0, 0, 0);                  //设置组件外部上方的填充量为 5 像素
gridBagConstraints_1.anchor = GridBagConstraints.WEST;                 //设置为靠左侧显示
gridBagConstraints_1.gridy = 2;                                        //设置行索引为 2
gridBagConstraints_1.gridx = 2;                                        //设置列索引为 2
panel.add(passwordField, gridBagConstraints_1);                        //将组件按指定的布局管理器添加到面板中
```

下面将创建一个用来显示按钮的面板，该面板的背景必须为透明的，否则将遮盖背景图片。该面板采用的布局管理器为水平箱布局，并且其占用网格组布局模式的两列。代码如下：

```
//创建并设置一个用来添加 3 个按钮的面板
final JPanel buttonPanel = new JPanel();                               //创建一个用来添加按钮的面板
buttonPanel.setOpaque(false);                                          //设置面板的背景为透明
buttonPanel.setLayout(new BoxLayout(buttonPanel, BoxLayout.X_AXIS));   //设置面板采用水平箱布局
final GridBagConstraints gridBagConstraints_4 = new GridBagConstraints();  //创建网格组布局管理器对象
gridBagConstraints_4.insets = new Insets(10, 0, 0, 0);                 //设置组件外部上方的填充量为 10 像素
gridBagConstraints_4.gridwidth = 2;                                    //设置其占两列
gridBagConstraints_4.gridy = 3;                                        //设置行索引为 3
gridBagConstraints_4.gridx = 1;                                        //设置列索引为 1
panel.add(buttonPanel, gridBagConstraints_4);                          //将组件按指定的布局管理器添加到面板中
```

用来绘制“登录”、“重置”和“退出”按钮的代码基本相同，所以在这里只介绍用来绘制“登录”按钮的代码。因为按钮的显示内容均是通过图片实现的，所以在创建按钮对象时，建议设置为不绘制

按钮的边框，也不绘制按钮的内容区域，目的是使按钮变为透明的，避免在按钮图片的四周出现不希望的绘制效果；并且建议将按钮边框和标签之间的间隔设置为 0。还可以为不同状态的按钮设置不同效果的图片，如图 21.21 所示为默认情况下按钮的效果，图 21.22 中的“登录”按钮则展示了当光标位于按钮上方时按钮的效果，图 21.23 中的“登录”按钮则展示了当按钮被按下时按钮的效果。

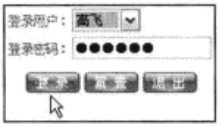


图 21.21　光标不在按钮上方

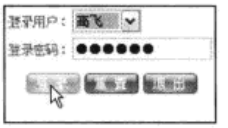


图 21.22　光标在按钮上方

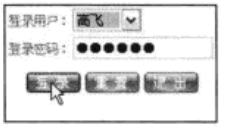


图 21.23　按钮被按下的瞬间

用来绘制“登录”按钮的具体代码如下：

```
final JButton landButton = new JButton();                          //创建“登录”按钮组件对象
❶    landButton.setMargin(new Insets(0, 0, 0, 0));                 //设置按钮边框和标签之间的间隔
❷    landButton.setContentAreaFilled(false);                       //设置不绘制按钮的内容区域
❸    landButton.setBorderPainted(false);                           //设置不绘制按钮的边框
//获得默认情况下“登录”按钮显示图片的 URL
URL landUrl = this.getClass().getResource("/img/land_submit.png");
❹    landButton.setIcon(new ImageIcon(landUrl));                   //设置默认情况下“登录”按钮显示的图片
//获得当鼠标经过“登录”按钮时显示图片的 URL
URL landOverUrl = this.getClass().getResource(
            "/img/land_submit_over.png");
❺    landButton.setRolloverIcon(new ImageIcon(landOverUrl));       //设置当鼠标经过“登录”按钮时显示的图片
URL landPressedUrl = this.getClass().getResource(
            "/img/land_submit_pressed.png");                       //获得当“登录”按钮被按下时显示图片的 URL
❻    landButton.setPressedIcon(new ImageIcon(landPressedUrl));     //设置当“登录”按钮被按下时显示的图片
landButton.addActionListener(new LandButtonActionListener());      //为“登录”按钮添加事件监听器
buttonPanel.add(landButton);                                       //将“登录”按钮添加到用来添加按钮的面板中
```

- ☑ ❶ setMargin(Insets m)：该方法用于设置按钮边框和标签之间的间隔。该方法接受 Insets 类的实例，Insets 类仅提供了一个构造方法 Insets(int top, int left, int bottom, int right)，4 个入口参数依次为按钮上方、左侧、下方和右侧的间距。
- ☑ ❷ setContentAreaFilled(boolean contentAreaFilled)：该方法用于设置是否绘制按钮的内容区域，默认为 true，即默认为绘制；如果为按钮设置了图片，则在图片的四周可能出现灰色边框，所以在这里设置为不绘制。
- ☑ ❸ setBorderPainted(boolean borderPainted)：该方法用于设置是否绘制按钮的边框，默认为 true，即默认为绘制；如果为按钮设置了图片，则建议设置为 false，即不绘制按钮边框。
- ☑ ❹ setIcon(Icon defaultIcon)：该方法用于设置按钮在默认情况下显示的图片。
- ☑ ❺ setRolloverIcon(Icon rolloverIcon)：该方法用于设置当光标在按钮上方时显示的图片。
- ☑ ❻ setPressedIcon(Icon pressedIcon)：该方法用于设置当按钮被按下时显示的图片。

当第一次使用本系统时，在数据库中将不存在系统管理员。在这种情况下，系统将提供一个默认用户，供用户登录后添加管理员，如图 21.24 所示。添加管理员后，将不再提供系统默认用户，如图 21.25 所示。

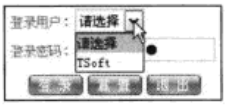


图 21.24　系统默认用户

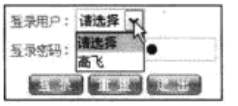


图 21.25　系统管理员

用户单击“登录”按钮后，系统将首先判断是否选择了登录用户，然后再判断是系统默认用户，还是系统管理员，最后验证登录密码，如果通过验证则登录成功，否则登录失败并弹出提示。实现“登录”按钮的具体代码如下：

```
class LandButtonActionListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        String username = usernameComboBox.getSelectedItem().toString();      //获得登录用户的名称
        if (username.equals("请选择")) {                                       //查看是否选择了登录用户
            JOptionPane.showMessageDialog(null, "请选择登录用户！", "友情提示",
                    JOptionPane.INFORMATION_MESSAGE);                         //弹出提示
            resetUsernameAndPassword();                                       //恢复登录用户和登录密码
        }
        char[] passwords = passwordField.getPassword();                       //获得登录用户的密码
         //将密码从 char 型数组转换成字符串
        String inputPassword = turnCharsToString(passwords);
        if (username.equals("TSoft")) {                                       //查看是否为默认用户登录
            if (inputPassword.equals("111")) {                                //查看密码是否为默认密码
                land(null);                                                   //登录成功
                String infos[] = { "请立刻单击“用户管理”按钮添加用户！",
                        "添加用户后需要重新登录，本系统才能正常使用！" };//组织提示信息
                JOptionPane.showMessageDialog(null, infos, "友情提示",
                        JOptionPane.INFORMATION_MESSAGE);                     //弹出提示
            } else {                                                          //密码错误
                //弹出提示
                JOptionPane.showMessageDialog(null, "默认用户 TSoft 的登录密码为 111！",
                        "友情提示", JOptionPane.INFORMATION_MESSAGE);
                passwordField.setText("111");                                 //将密码设置为默认密码
            }
        } else {
            if (inputPassword.length() == 0) {                                //用户未输入登录密码
                JOptionPane.showMessageDialog(null, "请输入登录密码！", "友情提示",
                        JOptionPane.INFORMATION_MESSAGE);                     //弹出提示
                resetUsernameAndPassword();                                   //恢复登录用户和登录密码
            }
            Vector user = Dao.getInstance().sUserByName(username);            //查询登录用户
            String password = user.get(5).toString();                         //获得登录用户的密码
            if (inputPassword.equals(password)) {                             //查看登录密码是否正确
                land(user);                                                   //登录成功
            } else {                                                          //登录密码错误
                //弹出提示
                JOptionPane.showMessageDialog(null, "登录密码错误，请确认后重新登录！",
                        "友情提示", JOptionPane.INFORMATION_MESSAGE);
                resetUsernameAndPassword();                                   //恢复登录用户和登录密码
            }
```

```
        }
    }
    private void resetUsernameAndPassword() {                    //恢复登录用户和登录密码
        usernameComboBox.setSelectedIndex(0);                    //恢复选中的登录用户为“请选择”项
        passwordField.setText("        ");                       //恢复密码框的默认值为 6 个空格
        return;                                                  //直接返回
    }
    private void land(Vector user) {                             //登录成功
        TipWizardFrame tipWizard = new TipWizardFrame(user);     //创建主窗体对象
        tipWizard.setVisible(true);                              //设置主窗体可见
        setVisible(false);                                       //设置登录窗口不可见
    }
}
```

21.8　开台签单工作区设计

视频讲解：光盘\TM\lx\21\开台签章工作区设计.exe

开台签单工作区是本系统最常用的工作区，所以需要将该工作区设计得更人性化和智能化。例如，在获取欲添加的菜品时，既可以通过菜品编号获得，又可以通过菜品助记码获得，并且默认菜品的数量为一个，因为这是最通用的。

21.8.1　开台签单工作区功能概述

开台签单工作区的主要功能有开台、点菜、加菜、签单、查看开台信息和签单信息，其效果如图 21.26 所示。

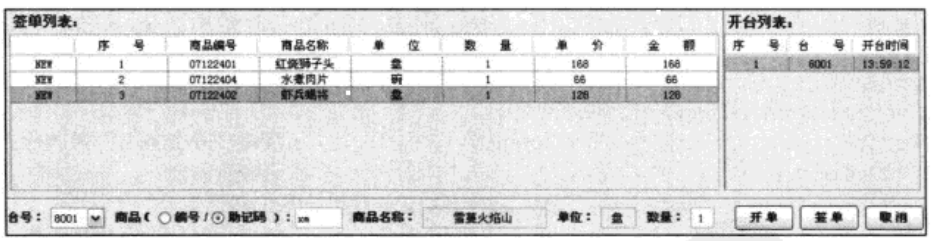


图 21.26　开台签单工作区效果

当顾客要求开台用餐时，首先在图 21.26 下方的“台号”下拉列表框中选择分配的餐台号，然后选择获取顾客点菜的方式，默认为通过助记码获取，也可以通过编号获取，这里假设以助记码获取。在输入商品助记码的过程中会在“商品名称”文本框中显示匹配商品的名称，并在“单位”文本框中显示该商品的销售单位。当在“商品名称”文本框中显示的为顾客所点的菜品时，如果点菜数量为 1，可以通过按 Enter 键将该菜品添加到签单列表中；如果需要修改数量，修改后则需要通过单击“开单”按钮添加到签单列表中。新点的菜品在表格的最前方会显示 NEW，在这种情况下可以选中后单击“取消”按钮取消该菜品。最后，在点菜结束后单击“签单”按钮，新点菜品前方的 NEW 会消失，在这种情

况下将不允许取消，至此点菜完成。

如果顾客在用餐的过程中要求添加菜品，既可以在“台号”下拉列表框中选择要求添加菜品的餐台号后添加，也可以在“开台列表”中选择要求添加菜品的餐台号，因为它与“台号”下拉列表框是联动的，即当在“台号”下拉列表框中选择餐台号后，如果在“开台列表”中存在该台号，对应的行也将被选中；如果更改“开台列表”中的选中行，在“台号”下拉列表框中也将更改为选中的餐台号。

21.8.2 开台签单工作区技术分析

在开发开台签单工作区时，为了使系统更人性化、智能化，要充分利用各种事件监听器。

例如为“台号”下拉列表框添加 ActionListener 监听器，用来捕获下拉列表框选项被改变的事件，目的是同步处理“开台列表”和“签单列表”中的信息；为“商品（编号/助记码）”文本框添加 KeyListener 监听器，用来捕获在该文本框中按键被释放的事件，目的是跟踪用户输入内容同步获取最接近的商品，尽量让用户输入最少的内容就能得到需要的商品；为“数量”文本框添加 FocusListener 监听器，用来捕获该文本框获得或失去焦点的事件，因为默认数量为 1，当获得焦点时将数量设置为空，用户在改变数量时就不用先删除默认数量 1 了，当失去焦点时，查看用户是否输入了数量，如果未输入，则恢复默认数量 1；为“开台列表”添加 MouseListener 监听器，用来捕获表格行被选中的事件，目的是同步处理“签单列表”和“台号”下拉列表框中的信息。

21.8.3 开台签单工作区实现过程

首先为“台号”下拉列表框添加事件监听器，用来处理开台或点菜的相关信息。如果选中的台号尚未开台（即新开台），则取消选择“开台列表”中的选中行，并清空“签单列表”中的所有行；如果选中的台号已经开台（即添加菜品），并且在“开台列表”中该台号未被选中，则选中“开台列表”中的该台号，并刷新“签单列表”中的菜品信息，即显示为当前选中台号所点的菜品。代码如下：

```
numComboBox.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
❶       int rowCount = rightTable.getRowCount();                    //获得“开台列表”中的行数，即已开台数
        if (rowCount > 0) {                                         //已经有开台
❷           String selectedDeskNum = numComboBox.getSelectedItem()
                    .toString();                                    //获得“台号”下拉列表框中选中的台号
            int needSelectedRow = -1;                               //默认选中的台号未开台
            //通过循环查看选中的台号是否已经开台
            opened: for (int row = 0; row < rowCount; row++) {
                //获得已开台的台号
❸               String openedDeskNum = rightTable.getValueAt(row, 1).toString();
                //查看选中的台号是否已经开台
                if (selectedDeskNum.equals(openedDeskNum)) {
                    needSelectedRow = row;                          //选中的台号已经开台
                    break opened;                                   //跳出循环
                }
```

```
            }
            if (needSelectedRow == -1) {                          //选中的台号尚未开台，即新开台
❹               rightTable.clearSelection();                      //取消选择“开台列表”中的选中行
                leftTableValueV.removeAllElements();              //清空“签单列表”中的所有行
❺               leftTableModel.setDataVector(leftTableValueV, leftTableColumnV); //刷新“签单列表”
            } else {                                              //选中的台号已经开台，即添加菜品
                //“台号”下拉列表框中选中的台号在“开台列表”中未被选中
                if (needSelectedRow != rightTable.getSelectedRow()) {
                    //在“开台列表”中选中该台号
                    rightTable.setRowSelectionInterval(needSelectedRow);
                    leftTableValueV.removeAllElements();      //清空“签单列表”中的所有行
                    //将选中台号的签单列表添加到“签单列表”中
                    leftTableValueV.addAll(menuOfDeskV.get(needSelectedRow));
                    //刷新“签单列表”
                    leftTableModel.setDataVector(leftTableValueV, leftTableColumnV);
                    leftTable.setRowSelectionInterval(0);     //选中“签单列表”中的第一行
                }
            }
        }
    }
});
```

- ☑ ❶ getRowCount()：该方法用于获取表格中拥有记录的行数，返回值为 int 型。
- ☑ ❷ getSelectedItem()：该方法用于获取下拉列表框中被选中的项目对象，返回值为 Object 型。
- ☑ ❸ getValueAt(int row, int column)：该方法用于获取指定单元格的值。需要注意的是，表格的行和列的索引均从 0 开始，返回值为 Object 型。
- ☑ ❹ clearSelection()：该方法用于取消选择当前表格中所有被选中的行。
- ☑ ❺ setDataVector(Vector dataVector, Vector columnIdentifiers)：该方法用于重新设置表格的列名和行数据。需要注意的是，第一个参数为用来封装表格行数据的向量，第二个参数为用来封装表格列名的向量。

然后开发智能获取点菜功能，通过为文本框添加键盘事件监听器实现。当按下的是 Enter 键时，等同于单击“开单”按钮，执行开单操作；如果按下的不是 Enter 键，则获取输入的内容，同时判断输入的是商品编号，还是商品助记码，并按指定条件查询所有符合条件的菜品，如果存在符合条件的菜品，则获取第一个符合条件的菜品，并显示菜品名称和单位，否则将菜品名称和单位设置为空。代码如下：

```
codeTextField.addKeyListener(new KeyAdapter() {
❶   public void keyReleased(KeyEvent e) {                         //通过键盘监听器实现智能获取菜品
❷       if (e.getKeyCode() == KeyEvent.VK_ENTER) {                //按 Enter 键
            makeOutAnInvoice();                                   //开单
        } else {
            String input = codeTextField.getText().trim();        //获得输入内容
            Vector vector = null;                                 //符合条件的菜品集合
            if (input.length() > 0) {                             //输入内容不为空
❸               if (codeRadioButton.isSelected()) {               //按助记码查询
                    vector = dao.sMenuByCode(input);              //查询符合条件的菜品
                    if (vector.size() > 0)                        //存在符合条件的菜品
```

```
                    vector = (Vector) vector.get(0);          //获得第一个符合条件的菜品
                else                                          //不存在符合条件的菜品
                    vector = null;
            } else {                                          //按编号查询
                vector = dao.sMenuById(input);                //查询符合条件的菜品
                if (vector.size() > 0)                        //存在符合条件的菜品
                    vector = (Vector) vector.get(0);          //获得第一个符合条件的菜品
                else                                          //不存在符合条件的菜品
                    vector = null;
            }
        }
        if (vector == null) {                                 //不存在符合条件的菜品
            nameTextField.setText(null);                      //设置“商品名称”文本框为空
            unitTextField.setText(null);                      //设置“单位”文本框为空
        } else {                                              //存在符合条件的菜品
            nameTextField.setText(vector.get(3).toString());  //设置为符合条件的菜品名称
            unitTextField.setText(vector.get(5).toString());  //设置为符合条件的菜品单位
        }
    }
}
});
```

☑ ❶ keyReleased()：当释放键盘中的按键后会触发该方法。

☑ ❷ getKeyCode()：该方法将返回一个 int 型值，该 int 型值代表触发此次事件的按键；静态常量 VK_ENTER 代表 Enter 键。

☑ ❸ isSelected()：该方法用来查看单选按钮是否处于选中状态，如果被选中则返回 true，否则返回 false。

下面的代码将实现一个智能化的用来填写数量的文本框，默认数量为 1，当文本框获得焦点时，自动将文本框设置为空；当文本框失去焦点时，将查看文本框是否输入了内容，如果未输入内容，则仍设置为默认数量 1。代码如下：

```
amountTextField = new JTextField();                           //创建“数量”文本框
amountTextField.addFocusListener(new FocusListener() {
❶   public void focusGained(FocusEvent e) {                   //当文本框获得焦点时执行
        amountTextField.setText(null);                        //设置“数量”文本框为空
    }
❷   public void focusLost(FocusEvent e) {                     //当文本框失去焦点时执行
        String amount = amountTextField.getText().trim();     //获得输入的数量
        if (amount.length() == 0)                             //未输入数量
            amountTextField.setText("1");                     //恢复为默认数量 1
    }
});
amountTextField.setText("1");                                 //默认数量为 1
```

☑ ❶ focusGained(FocusEvent e)：当被监听的组件对象获得焦点时将触发该方法。

☑ ❷ focusLost(FocusEvent e)：当被监听的组件对象失去焦点时将触发该方法。

如果在输入商品编号或助记码时按 Enter 键，或者单击“开单”按钮，将执行开台点菜操作。如果

是新开台点菜，则需要先处理开台信息，即在“开台列表”中添加新开台的信息，然后再处理点菜信息，即在“签单列表”中添加新点菜的信息；如果是为已开台添加菜品，则直接处理点菜信息。代码如下：

```
private void makeOutAnInvoice() {
    String deskNum = numComboBox.getSelectedItem().toString();        //获得台号
    String menuName = nameTextField.getText();                        //获得商品名称
    String menuAmount = amountTextField.getText();                    //获得数量
    if (deskNum.equals("请选择")) {                                    //验证是否已经选择台号
        JOptionPane.showMessageDialog(null, "请选择台号！", "友情提示",
                JOptionPane.INFORMATION_MESSAGE);
        return;
    }
    if (menuName.length() == 0) {                                     //验证是否已经确定商品
        JOptionPane.showMessageDialog(null, "请录入商品名称！", "友情提示",
                JOptionPane.INFORMATION_MESSAGE);
        return;
    }
    //验证数量是否有效，数量必须在 1～99 之间
    if (!Validate.execute("[1-9]{1}([0-9]{0,1})", menuAmount)) {
        String info[] = new String[] { "您输入的数量错误！", "数量必须在 1-99 之间！" };
        JOptionPane.showMessageDialog(null, info, "友情提示", JOptionPane.INFORMATION_MESSAGE);
        return;
    }
    //处理开台信息
    int rightSelectedRow = rightTable.getSelectedRow();               //获得被选中的台号
    int leftRowCount = 0;                                             //默认点菜数量为 0
    if (rightSelectedRow == -1) {                                     //没有被选中的台号，即新开台
        rightSelectedRow = rightTable.getRowCount();                  //被选中的台号为新开的台
        Vector deskV = new Vector();                                  //创建一个代表新开台的向量对象
        deskV.add(rightSelectedRow + 1);                              //添加开台序号
        deskV.add(deskNum);                                           //添加开台号
        deskV.add(Today.getTime());                                   //添加开台时间
        rightTableModel.addRow(deskV);                                //将开台信息添加到“开台列表”中
        rightTable.setRowSelectionInterval(rightSelectedRow);         //选中新开的台
        menuOfDeskV.add(new Vector());                                //添加一个对应的签单列表
    } else {                                                          //选中的台号已经开台，即添加菜品
        leftRowCount = leftTable.getRowCount();                       //获得已点菜的数量
    }
    //处理点菜信息
    Vector vector = dao.sMenuByName(menuName);                        //获得被点菜品
    int amount = Integer.valueOf(menuAmount);                         //将菜品数量转为 int 型
    int unitPrice = Integer.valueOf(vector.get(5).toString());        //将菜品单价转为 int 型
    int money = unitPrice * amount;                                   //计算菜品消费额
    Vector<Object> menuV = new Vector<Object>();                      //创建一个代表新点菜的向量对象
    menuV.add("NEW");                                                 //添加新点菜标记
    menuV.add(leftRowCount + 1);                                      //添加点菜序号
    menuV.add(vector.get(0));                                         //添加菜品编号
    menuV.add(menuName);                                              //添加菜品名称
```

```
    menuV.add(vector.get(4));                                    //添加菜品单位
    menuV.add(amount);                                           //添加菜品数量
    menuV.add(unitPrice);                                        //添加菜品单价
    menuV.add(money);                                            //添加菜品消费额
    leftTableModel.addRow(menuV);                                //将点菜信息添加到“签单列表”中
    leftTable.setRowSelectionInterval(leftRowCount);             //将新点菜设置为选中行
    menuOfDeskV.get(rightSelectedRow).add(menuV);                //将新点菜信息添加到对应的签单列表
}
```

在新添加菜品的前方会有一个 NEW 标记，确定点菜结束后单击“签单”按钮，将取消所有新添加菜品前方的 NEW 标记。在未取消 NEW 标记的情况下可以选中后单击“取消”按钮取消该菜品，如果该餐台只点了该菜品，取消该菜品后将同时取消该餐台的开台信息；如果该餐台已经点了其他菜品，并且取消的不是最后点的菜品，还需要修改所点菜品的序号。代码如下：

```
String NEW = leftTable.getValueAt(lSelectedRow, 0).toString();          //获得选中菜品的新点菜标记
if (NEW.equals("")) {                                                   //没有新点菜标记，不允许取消
    JOptionPane.showMessageDialog(null, "很抱歉，该商品已经不能取消！",
            "友情提示", JOptionPane.INFORMATION_MESSAGE);
    return;
} else {
    //获得“开台列表”中的选中行，即取消菜品的台号
    int rSelectedRow = rightTable.getSelectedRow();
    int i = JOptionPane.showConfirmDialog(null, "确定要取消“" + rightTable.getValueAt(rSelectedRow, 1)
            + "”中的商品“" + leftTable.getValueAt(lSelectedRow, 3) + "”？",
            "友情提示", JOptionPane.YES_NO_OPTION);                     //弹出提示信息确认是否取消
    if (i == 0) {                                                       //确认取消
        leftTableModel.removeRow(lSelectedRow);                         //从“签单列表”中取消菜品
        int rowCount = leftTable.getRowCount();                         //获得取消后的点菜数量
        if (rowCount == 0) {                                            //未点任何菜品
            rightTableModel.removeRow(rSelectedRow);                    //取消开台
            menuOfDeskV.remove(rSelectedRow);                           //移除签单列表
        } else {
            if (lSelectedRow == rowCount) {                             //取消菜品为最后一个
                lSelectedRow -= 1;                                      //设置最后一个菜品为选中的
            } else {                                                    //取消菜品不是最后一个
                Vector<Vector<Object>> menus = menuOfDeskV.get(rSelectedRow);
                for (int row = lSelectedRow; row < rowCount; row++) {   //修改点菜序号
                    leftTable.setValueAt(row + 1, row, 1);
                    menus.get(row).set(1, row + 1);
                }
            }
            leftTable.setRowSelectionInterval(lSelectedRow);            //设置选中行
        }
    }
}
```

当用户要求添加菜品时，可以在“台号”下拉列表框中选择要求添加菜品的台号，也可以在“开台列表”中选择要求添加菜品的台号，在这里选择后将同步选中“台号”下拉列表框中的相应台号。代码如下：

```
rightTable.addMouseListener(new MouseAdapter() {
❶   public void mouseClicked(MouseEvent e) {
        int rSelectedRow = rightTable.getSelectedRow();                    //获得“开台列表”中的选中行
        leftTableValueV.removeAllElements();                               //清空“签单列表”中的所有行
        //将选中台号的签单列表添加到“签单列表”中
        leftTableValueV.addAll(menuOfDeskV.get(rSelectedRow));
         //刷新“签单列表”
        leftTableModel.setDataVector(leftTableValueV, leftTableColumnV);
        leftTable.setRowSelectionInterval(0);                              //选中“签单列表”中的第一行
        //同步选中“台号”下拉菜单中的相应台号
❷       numComboBox.setSelectedItem(rightTable.getValueAt(rSelectedRow, 1));
    }
});
```

☑ ❶ mouseClicked(MouseEvent e)：当鼠标按键被按住并且在原地释放时触发该方法；假设为 JTable 表格添加该监听器，如果是在第一行上方按下鼠标按键，但是并不立即释放，而是等移动到其他行再释放被按下的鼠标按键，将不会触发该方法。

☑ ❷ setSelectedItem(Object selectedItem)：该方法用来设置下拉列表框中被选中的选项。

21.8.4　单元测试

在测试快速获取商品功能时，输入部分助记码后再输入一个空格，在“商品名称”文本框中仍然显示输入空格之前的商品名称，如图 21.27 所示。

图 21.27　输入空格后的效果

这其中有两个小错误，一个错误是输入空格后就不应该显示原来的商品名称，这是因为在获取输入内容时去掉了首尾空格。代码如下：

```
String input = codeTextField.getText().trim();                          //获取输入内容
```

另一个错误是该文本框就不应该允许输入空格，解决了这个错误，上面的小错误也就自然解决了。但是建议将去掉首尾空格的代码去掉，这样对提升软件性能还是有好处的。这也是一个良好的编码习惯，就是应该尽量避免编写没有必要的代码。

如果想令文本框不允许输入空格，可以通过重载 KeyListener 类的 keyTyped(KeyEvent e)方法实现，通过该方法的入口参数 e 的 getKeyChar()方法可以得到此次输入的字符，如果为空格则通过 consume()方法销毁此次事件。代码如下：

```
public void keyTyped(KeyEvent e) {
    if (e.getKeyChar() == ' ')                                          //判断用户输入的是否为空格
        e.consume();                                                    //如果是空格则销毁此次按键事件
}
```

同样，对“数量”文本框也可以采用这种方法控制用户输入的内容，并且可以控制输入数量的最大位数和不允许输入的第一位为 0。代码如下：

```
amountTextField.addKeyListener(new KeyAdapter() {
    public void keyTyped(KeyEvent e) {
        int length = amountTextField.getText().length();//获取当前数量的位数
        if (length < 2) {                                //位数小于两位
            //将允许输入的字符定义成字符串
            String num = (length == 0 ? "123456789" : "0123456789");
            if (num.indexOf(e.getKeyChar()) < 0)         //查看按键字符是否包含在允许输入的字符中
                e.consume();                             //如果不包含在允许输入的字符中则销毁此次按键事件
        } else {
            e.consume();                                 //如果不小于数量允许的最大位数则销毁此次按键事件
        }
    }
});
```

21.9 自动结账工作区设计

视频讲解：光盘\TM\lx\21\自动结账工作区设计.exe

21.9.1 自动结账工作区功能概述

自动结账工作区有两个主要功能，一个功能是自动计算当前选中餐台的消费金额，例如选中餐台8001，如图 21.28 所示，在自动结账工作区将显示该餐台的消费金额，如图 21.29 所示。

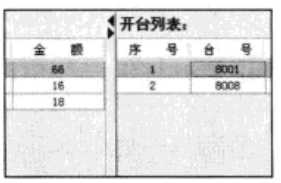


图 21.28 选中的台号为 8001

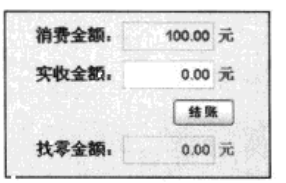


图 21.29 8001 的消费金额

另一个功能是在结账时自动计算找零金额。用户输入“实收金额”后单击“结账”按钮，系统将自动计算出需要找零的金额，并弹出一个结账完成的提示框，如图 21.30 所示。

图 21.30 结账提示

21.9.2 自动结账工作区技术分析

如果要实现自动计算当前选中餐台消费金额的功能，就要随时监控“签单列表”中内容的变化。例如添加或取消菜品，或者是“开台列表”中的选中行发生了改变，都将导致“签单列表”中的内容发生改变。如果希望随时监控表格中内容的变化，可通过为表格模型添加 TableModelListener 监听器实现，无论是向表格模型中添加行，还是从表格模型中移除行，以及修改表格中某一单元格的值，都将触发该事件。

21.9.3　自动结账工作区实现过程

首先实现自动计算当前选中餐台消费金额的功能，即为与“签单列表”对应的表格模型添加一个 TableModelListener 监听器，在监听器中通过循环重新计算该餐台的消费金额，并更新“消费金额”文本框。代码如下：

```
leftTableModel.addTableModelListener(new TableModelListener() {
    public void tableChanged(TableModelEvent e) {                    //通过表格模型监听器实现自动结账
        int rowCount = leftTable.getRowCount();                      //获得签单列表中的行数
        float expenditure = 0.0f;                                    //默认消费 0 元
        for (int row = 0; row < rowCount; row++) {                   //通过循环计算消费金额
            expenditure += Float.valueOf(leftTable.getValueAt(row, 7).toString());     //累加消费金额
        }
        expenditureTextField.setText(expenditure + "0");             //更新“消费金额”文本框
    }
});
```

然后实现结账功能。在结账之前首先要判断是否有未签单的菜品，如果有则弹出提示信息，如果没有则获得消费金额和实收金额；然后判断实收金额是否小于消费金额，如果小于同样弹出提示，否则进行结账操作，将消费信息持久化到数据库，并弹出结账完成的提示框；最后将“实收金额”文本框和“找零金额”文本框设置为默认值。代码如下：

```
int rowCount = leftTable.getRowCount();                                  //获得结账餐台的点菜数量
String NEW = leftTable.getValueAt(rowCount - 1, 0).toString();           //获得最后点菜的标记
//如果最后点菜被标记为 NEW，则弹出提示
if (NEW.equals("NEW")) {
    JOptionPane.showMessageDialog(null, "请先确定未签单商品的处理方式！",
            "友情提示", JOptionPane.INFORMATION_MESSAGE);
} else {
    float expenditure = Float.valueOf(expenditureTextField.getText());   //获得消费金额
    float realWages = Float.valueOf(realWagesTextField.getText());       //获得实收金额
    //如果实收金额小于消费金额，则弹出提示
    if (realWages < expenditure) {
        JOptionPane.showMessageDialog(null, "请输入实收金额！",
                "友情提示", JOptionPane.INFORMATION_MESSAGE);
        //令“实收金额”文本框获得焦点
❶       realWagesTextField.requestFocus();
    } else {
        changeTextField.setText(realWages – expenditure + "0");          //计算并设置“找零金额”文本框
        String[] values = {getNum(), rightTable.getValueAt(selectedRow, 1).toString(),
                Today.getDate() + " " + rightTable.getValueAt(selectedRow, 2),
                expenditureTextField.getText(), user.get(0).toString() };  //组织消费单信息
        dao.iOrderForm(values);                                          //持久化到数据库
        values[0] = dao.sOrderFormOfMaxId();                             //获得消费单编号
        //通过循环获得各个消费项目的信息
        for (int i = 0; i < rowCount; i++) {
            values[1] = leftTable.getValueAt(i, 2).toString();           //获得商品编号
```

```
            values[2] = leftTable.getValueAt(i, 5).toString();                    //获得商品数量
            values[3] = leftTable.getValueAt(i, 7).toString();                    //获得商品消费金额
            dao.iOrderItem(values);                                               //持久化到数据库
        }
        JOptionPane.showMessageDialog(null, rightTable
                .getValueAt(selectedRow, 1) + "结账完成！", "友情提示",
                JOptionPane.INFORMATION_MESSAGE);                                 //弹出结账完成提示
❷       rightTableModel.removeRow(selectedRow);                                   //从“开台列表”中取消开台
        leftTableValueV.removeAllElements();                                      //清空“签单列表”
        leftTableModel.setDataVector(leftTableValueV, leftTableColumnV);          //刷新“签单列表”
        realWagesTextField.setText("0.00");                                       //清空“实收金额”文本框
        changeTextField.setText("0.00");                                          //清空“找零金额”文本框
        menuOfDeskV.remove(selectedRow);              //从“签单列表”集合中移除已结账的签单列表
    }
}
```

☑ ❶ requestFocus()：该方法用来为调用的组件对象请求获得焦点。

☑ ❷ removeRow(int rowIndex)：该方法用来从表格中移除指定行索引所代表的行。

21.10 结账报表工作区设计

视频讲解：光盘\TM\lx\21\结账报表工作区设计.exe

21.10.1 结账报表工作区功能概述

本系统提供了 3 种方式的结账报表，分别是日结账报表、月结账报表和年结账报表，在结账报表工作区只提供了打开这 3 种结账报表功能的按钮，如图 21.31 所示。

日结账报表功能提供了对一日营业情况的统计，包括日开台数量、各个餐台的消费金额、菜品的消费情况、各个菜品的日销售情况，以及日营业额等，如图 21.32 所示。

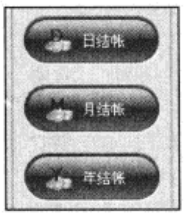


图 21.31 结账报表工作区

图 21.32 日结账报表

月结账报表功能提供了对一个月营业情况的统计，包括日开台总数、日总营业额、日开台的平均消费额、日开台的最大和最小消费额、当月的总开台数、月总营业额，以及一个月中的日平均营业额、一个月中开台的最大和最小消费额，如图 21.33 所示。

年结账报表功能提供了对一年营业情况的统计，包括一年中每天的营业额、每月的营业额、每月同一日期的总营业额，以及一年的营业额，如图 21.34 所示。

月结账

2007 年 12 月 确定

日期	开台总数	消费总额	平均消费额	最大消费额	最小消费额
24	6	1397	232	428	28
25	——	——	——	——	——
26	7	1328	189	428	100
27	——	——	——	——	——
28	——	——	——	——	——
29	——	——	——	——	——
30	——	——	——	——	——
31	——	——	——	——	——
总计	13	2725	209	428	28

图 21.33　月结账报表

年结账

2007 年 确定

日期	月	7 月	8 月	9 月	10 月	11 月	12 月	总计
24	—	——	——	——	——	——	1397	1397
25	—	——	——	——	——	——	——	——
26	—	——	——	——	——	——	1328	1328
27	—	——	——	——	——	——	——	——
28	—	——	——	——	——	——	——	——
29	—	——	——	——	——	——	——	——
30	—	——	——	——	——	——	——	——
31	—	——	——	——	——	——	——	——
总计	—	——	——	——	——	——	2725	2725

图 21.34　年结账报表

21.10.2　结账报表工作区技术分析

在实现结账报表功能时有以下两个技术要点：

☑　对日期有效性的控制

例如，在实现日结账功能时，无论用户修改了统计日期的年度或月份，都会影响到日下拉列表框中的可选项，包括大月（31 天）和小月（30 天）的变化，以及 2 月份在平年（28 天）和闰年（29 天）的变化，如果不能正确处理这些变化，将导致系统无法正常运行。其实在实现月结账报表和年结账报表时也会涉及这个问题，只是在系统界面上不会明显地体会到。解决该问题的大体思路是通过为年度和月份下拉列表框添加事件监听器，实现对日下拉列表框可选项的控制。

☑　对统计表格的控制

当系统界面不能显示出所有统计记录时，只需要将表格放到滚动面板中，这个方法对系统界面不能显示出统计记录的所有行很有效，因为在移动垂直滚动条时，表格的列名并不会随之滚动，即表格的列名是永远可见的；但是当系统界面不能显示出统计记录的所有列时，这个方法就不是很好了，因为在移动水平滚动条时，表格的所有列都会随之滚动，导致最左侧的一列或几列不可见，而表格最左侧的一列或者是几列通常情况下也希望是永远可见的，即不会随着滚动条的移动而滚动。解决该问题的大体思路是实现两个表格，一个表格用来显示最左侧希望永远可见的一列或几列；另一个表格用来显示其他列，然后将两个表格并列显示。

21.10.3　结账报表工作区实现过程

首先解决在实现结账报表功能时日期的有效性问题，需要定义一个数组，用来存放各个月份拥有

的天数，默认 2 月份为 28 天。为了方便使用，将月份与数组的索引一一对应，即不使用数组索引为 1 的位置。代码如下：

```
private int daysOfMonth[] = { 0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };
```

下面为年度下拉列表框添加事件监听器。首先获得选中的年度，并判断是平年还是闰年，以确定 2 月份的天数，即修改例程 30 中索引为 2 的值，如果为平年则修改为 28，为闰年则修改为 29；然后获得当前选中的月份，如果当前选中的为 2 月份，则继续获得日下拉列表框拥有可选项的数量。如果日下拉列表框拥有可选项的数量不等于例程 30 中索引为 2 的值，当日下拉列表框拥有可选项的数量为 28 时，则为日下拉列表框添加一个可选项 29，否则从日下拉列表框中移除可选项 29。代码如下：

```
yearComboBox.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        int year = (Integer) yearComboBox.getSelectedItem();      //获得选中的年度
        judgeLeapYear(year);                                      //判断是否为闰年，以确定 2 月份的天数
        int month = (Integer) monthComboBox.getSelectedItem();    //获得选中的月份
        if (month == 2) {                                         //如果选中的为 2 月
❶           int itemCount = dayComboBox.getItemCount();           //获得日下拉列表框当前的天数
            //如果日下拉列表框当前的天数不等于 2 月份的天数
            if (itemCount != daysOfMonth[2]) {
                if (itemCount == 28)                              //如果日下拉列表框当前的天数为 28 天
❷                   dayComboBox.addItem(29);                      //则添加为 29 天
                else                                              //否则日下拉列表框当前的天数则为 29 天
❸                   dayComboBox.removeItem(29);                   //则减少为 28 天
            }
        }
    }
});
```

☑ ❶ getItemCount()：该方法用来获得下拉列表框拥有选项的数量。

☑ ❷ addItem(Object item)：该方法用来向下拉列表框中添加指定选项。

☑ ❸ removeItem(Object item)：该方法用来从下拉列表框中移除指定选项。

下面为月份下拉列表框添加事件监听器。首先获得选中的月份，并获得日下拉列表框拥有可选项的数量。如果日下拉列表框拥有可选项的数量不等于当前选中月份拥有的天数，当日下拉列表框拥有可选项的数量大于当前选中月份拥有的天数时，则移除日下拉列表框中最大的可选项，并将日下拉列表框拥有可选项的数量减 1；否则将日下拉列表框拥有可选项的数量加 1，并添加到日下拉列表框的可选项中。代码如下：

```
monthComboBox.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        int month = (Integer) monthComboBox.getSelectedItem();    //获得选中的月份
        int itemCount = dayComboBox.getItemCount();               //获得日下拉列表框当前的天数
        //如果日下拉列表框当前的天数不等于选中月份的天数
        while (itemCount != daysOfMonth[month]) {
            if (itemCount > daysOfMonth[month]) {                 //如果大于选中月份的天数
                dayComboBox.removeItem(itemCount);                //则移除最后一个可选项
```

```
                itemCount--;                                    //并将日下拉列表框当前的天数减 1
            } else {                                            //否则小于选中月份的天数
                itemCount++;                                    //将日下拉列表框当前的天数加 1
                dayComboBox.addItem(itemCount);                 //并添加为可选项
            }
        }
    }
}
```

通过年度和月份下拉列表框的事件监听器对日下拉列表框可选项的控制，无论选择哪一年或哪一月，日下拉列表框提供的日期可选项都是一个有效的日期。

下面解决当系统界面不能显示出统计记录的所有列时，移动滚动条导致最左侧的一列或几列不可见的问题。首先通过实现抽象类 AbstractTableModel，编写一个用来创建固定列表格模型的类 FixedColumnTableModel。代码如下：

```
class FixedColumnTableModel extends AbstractTableModel {
    public int getColumnCount() {                               //返回固定列的数量
        return fixedColumn;
    }
    public int getRowCount() {                                  //返回行数
        return tableValueV.size();
    }
    public Object getValueAt(int rowIndex, int columnIndex) {   //返回指定单元格的值
        return tableValueV.get(rowIndex).get(columnIndex);
    }
    public String getColumnName(int columnIndex) {              //返回指定列的名称
        return tableColumnV.get(columnIndex);
    }
}
```

然后再通过实现抽象类 AbstractTableModel，编写一个用来创建移动列表格模型的类 FloatingColumnTableModel。代码如下：

```
class FloatingColumnTableModel extends AbstractTableModel {
    public int getColumnCount() {                               //返回移动列的数量
        return tableColumnV.size() - fixedColumn;               //需要扣除固定列的数量
    }
    public int getRowCount() {                                  //返回行数
        return tableValueV.size();
    }
    public Object getValueAt(int rowIndex, int columnIndex) {   //返回指定单元格的值
        //需要为列索引加上固定列的数量
        return tableValueV.get(rowIndex).get(columnIndex + fixedColumn);
    }
    public String getColumnName(int columnIndex) {              //返回指定列的名称
        return tableColumnV.get(columnIndex + fixedColumn);     //需要为列索引加上固定列的数量
    }
}
```

注意

在处理与表格列有关的信息时，均需要在表格总列数的基础上减去固定列的数量。

下面通过实现接口 ListSelectionListener，编写一个用来同步两个表格中的选中行的事件监听器类 MListSelectionListener，即当选中固定列表格中的某一行时，监听器会同时选中移动列表格中的对应行，同样，当选中移动列表格中的某一行时，监听器会同时选中固定列表格中的对应行。代码如下：

```
class MListSelectionListener implements ListSelectionListener {
    boolean isFixedColumnTable = true;                          //默认由选中固定列表格中的行触发
    public MListSelectionListener(boolean isFixedColumnTable) {
        this.isFixedColumnTable = isFixedColumnTable;
    }
    public void valueChanged(ListSelectionEvent e) {
        if (isFixedColumnTable) {                               //由选中固定列表格中的行触发
            //获得固定列表格中的选中行
            int selectedRow = fixedColumnTable.getSelectedRow();
            //同时选中移动列表格中的选中行
            floatingColumnTable.setRowSelectionInterval(selectedRow);
        } else {                                                //由选中移动列表格中的行触发
            //获得移动列表格中的选中行
            int selectedRow = floatingColumnTable.getSelectedRow();
            //同时选中固定列表格中的选中行
            fixedColumnTable.setRowSelectionInterval(selectedRow);
        }
    }
}
```

最后依次创建固定列表格和移动列表格，并通过这两个表格的选择模型对象，为两个表格添加事件监听器；再创建一个滚动面板对象，并将固定列表格的列头对象添加到滚动面板的左上方；最后创建一个视口对象，先将固定表格对象添加到视口对象中，并将视口的首选大小设置为固定列表格的首选大小，并依次将视口和移动列表格添加到滚动面板的标题视口和默认视口中。代码如下：

```
fixedColumnTableModel = new FixedColumnTableModel();                    //创建固定列表格模型对象
fixedColumnTable = new MTable(fixedColumnTableModel);                   //创建固定列表格对象
ListSelectionModel fixed = fixedColumnTable.getSelectionModel();        //获得选择模型对象
fixed.addListSelectionListener(new MListSelectionListener(true));       //添加行被选中的事件监听器
floatingColumnTableModel = new FloatingColumnTableModel();              //创建移动列表格模型对象
floatingColumnTable = new MTable(floatingColumnTableModel);             //创建移动列表格对象
ListSelectionModel floating = floatingColumnTable.getSelectionModel();  //获得选择模型对象
floating.addListSelectionListener(new MListSelectionListener(false));   //添加行被选中的事件监听器
JScrollPane scrollPane = new JScrollPane();                             //创建一个滚动面板对象
//将固定列表格头放到滚动面板的左上方
scrollPane.setCorner(JScrollPane.UPPER_LEFT_CORNER, fixedColumnTable.getTableHeader());
JViewport viewport = new JViewport();                           //创建一个用来显示基础信息的视口对象
viewport.setView(fixedColumnTable);                             //将固定列表格添加到视口中
//设置视口的首选大小为固定列表格的首选大小
```

```
viewport.setPreferredSize(fixedColumnTable.getPreferredSize());
scrollPane.setRowHeaderView(viewport);                    //将视口添加到滚动面板的标题视口中
scrollPane.setViewportView(floatingColumnTable);          //将移动列表格添加到默认视口
```

21.10.4　单元测试

在测试年结账报表功能时，当系统界面不能显示出统计记录的所有列时，移动滚动条导致最左侧的一列或几列不可见的问题的确解决了；但是当选中右侧移动列表格的多行时，在左侧固定列表格中只选中了一行，如图 21.35 所示；当选中右侧移动列表格的第 26～30 行时，在左侧固定列表格中只选中了第 26 行。

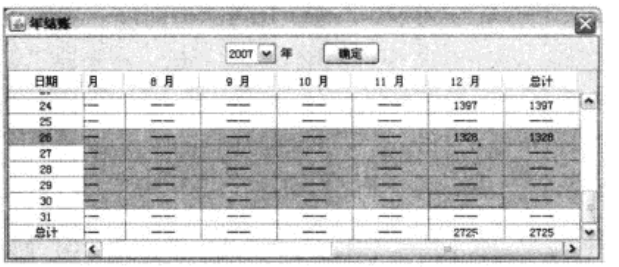


图 21.35　选中移动列表格的多行

同样，当选中左侧固定列表格的多行时，在右侧移动列表格中也只选中了一行，如图 21.36 所示，当选中左侧固定列表格的第 26～30 行时，在右侧移动列表格中只选中了第 26 行。

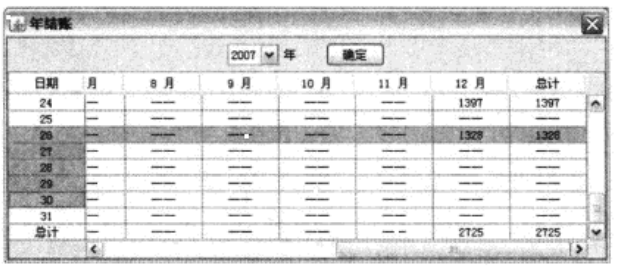


图 21.36　选中固定列表格的多行

图 21.35 和图 21.36 所示的两种情况并不是想要的，这是因为在实现事件监听器时，并没有同步选中关联表中的所有对应行，而只是选中了关联表中的第一个对应行，但表格却不是单选模式的，这样才导致出现这种情况。解决的办法是将表格设置为单选模式，因为这两个表格均是通过 FixedColumnTablePanel 类的内部类 MTable 实现的，所以只需要在内部类 MTable 中重构其父类 JTable 类的 getSelectionModel()方法，代码如下：

```
public ListSelectionModel getSelectionModel() {
    ListSelectionModel selectionModel = super.getSelectionModel();
    selectionModel.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
    return selectionModel;
}
```

21.11 后台管理工作区设计

视频讲解：光盘\TM\lx\21\后台管理工作区设计.exe

后台管理工作区用来维护软件正常运行需要的一些信息，如台号信息、菜系信息、菜品信息，只有填写了这些信息，才能进行开台，以至结账和生成报表。

21.11.1 后台管理工作区功能概述

在后台管理工作区提供了对台号、菜系和菜品信息的维护功能，在添加信息时，一是验证数据的合法性，例如在添加台号信息时，不小心将座位数输入为了 100，在单击“添加”按钮时将弹出座位数输入错误的提示，如图 21.37 所示；再就是查看新添加的信息是否已经存在，例如在添加菜系信息时，输入“炖炒类”后单击“添加”按钮，将弹出菜系已经存在的提示，因为添加同名的菜系是没有意义的，如图 21.38 所示。

图 21.37 餐台座位数输入错误

图 21.38 添加的菜系已经存在

在删除信息之前，通常情况下建议弹出一个确认提示框，以免由于疏忽错误删除信息，如图 21.39 所示。

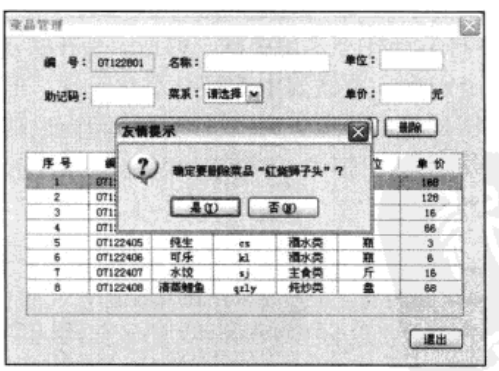


图 21.39 删除菜品之前弹出的确认提示框

21.11.2　后台管理工作区技术分析

在对用户输入的数据进行验证时，如果某个数据不符合要求，通常情况下希望对应的组件获得焦点。如果是对用户输入的数据进行逐个验证，这个问题就很好解决了；但是当对用户输入的数据进行批量验证时，就很难判断是哪个组件接收的数据不符合要求。这个问题可以通过 Java 的反射机制解决。通过 Java 反射可以轻松地实现对组件的内容进行批量验证，并且在接收数据不符合要求的情况下，可以直接令相应的组件获得焦点，另外通过为组件设置名称，还可以弹出一个很有针对性的提示。

例如，在实现添加菜品功能时，就是通过 Java 反射机制验证 4 个文本框是否为空，在未输入单价的情况下直接单击“添加”按钮，就会弹出一个很有针对性的提示，如图 21.40 所示，单击提示框中的“确定”按钮后，“单价”文本框将获得焦点，如图 21.41 所示。

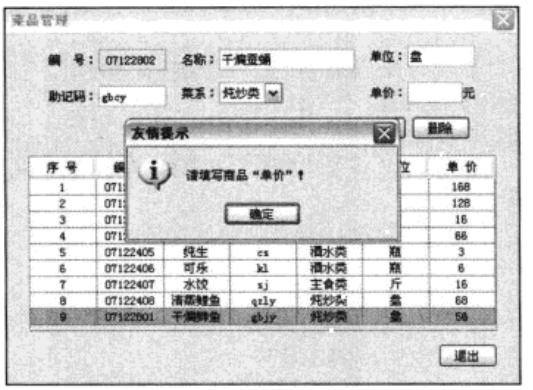


图 21.40　有针对性的提示内容

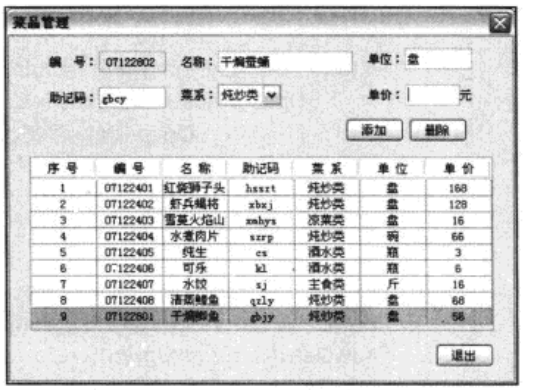


图 21.41　为空的“单价”文本框获得焦点

21.11.3　后台管理工作区实现过程

后台管理包括对台号、菜系和菜品的管理，下面将依次讲解这 3 个功能的实现过程，以及一些典型的技巧。

1. 实现台号管理功能

首先实现添加台号的功能。在执行添加台号操作时，首先要判断台号和座位数是否有效，台号最多为 5 个字符，座位数不能大于 99，并且座位号允许重复；然后创建一个向量对象，用来封装新添加台号的信息，并添加到表格中；最后将新添加的台号信息保存到数据库中。代码如下：

```
final JButton addButton = new JButton();                          //创建添加台号按钮对象
addButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        String num = numTextField.getText().trim();               //获取台号，并去掉首尾空格
        String seating = seatingTextField.getText().trim();       //获取座位数，并去掉首尾空格
        if (num.equals("") || seating.equals("")) {               //查看是否输入了台号和座位数
```

```
                JOptionPane.showMessageDialog(null, "请输入台号和座位数！", "友情提示",
                        JOptionPane.INFORMATION_MESSAGE);
                return;
            }
            if (num.length() > 5) {                                    //查看台号的长度是否超过 5 位
                JOptionPane.showMessageDialog(null, "台号最多只能为 5 个字符！", "友情提示",
                        JOptionPane.INFORMATION_MESSAGE);
                numTextField.requestFocus();                           //为“台号”文本框请求获得焦点
                return;
            }
            if (!Validate.execute("[1-9]{1}([0-9]{0,1})", seating)) {  //验证座位数是否在 1～99 之间
                String[] infos = { "座位数输入错误！", "座位数必须在 1～99 之间！" };
                JOptionPane.showMessageDialog(null, infos, “友情提示”,
JOptionPane.INFORMATION_MESSAGE);
                seatingTextField.requestFocus();                       //为“座位数”文本框请求获得焦点
                return;
            }
            if (dao.sDeskByNum(num) != null) {                         //查看该台号是否已经存在
                JOptionPane.showMessageDialog(null, "该台号已经存在！", "友情提示",
                        JOptionPane.INFORMATION_MESSAGE);
                numTextField.requestFocus();                           //为“台号”文本框请求获得焦点
                return;
            }
            int row = table.getRowCount();                             //获得当前拥有台号的个数
            Vector newDeskNumV = new Vector();                         //创建一个代表新台号的向量
            newDeskNumV.add(new Integer(row + 1));                     //添加序号
            newDeskNumV.add(num);                                      //添加台号
            newDeskNumV.add(seating);                                  //添加座位数
            tableModel.addRow(newDeskNumV);                            //将新台号信息添加到表格中
            table.setRowSelectionInterval(row, row);                   //设置新添加的台号为被选中
            numTextField.setText(null);                                //将“台号”文本框设置为空
            seatingTextField.setText(null);                            //将“座位数”文本框设置为空
            dao.iDesk(num, seating);                                   //将新添加的台号信息保存到数据库中
            JDBC.closeConnection();                                    //关闭数据库连接
        }
    });
    addButton.setText("添加");
```

然后实现删除台号的功能。在执行删除台号操作之前，首先要判断是否选中了要删除的台号；然后弹出提示确认是否真的删除，如果真的要删除，还要判断该餐台是否正在被使用；最后执行删除操作。代码如下：

```
final JButton delButton = new JButton();                               //创建删除台号按钮对象
delButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        int selectedRow = table.getSelectedRow();                      //获得选中的餐台
        if (selectedRow == -1) {                                       //未选中任何餐台
            JOptionPane.showMessageDialog(null, "请选择要删除的餐台！", "友情提示",
                    JOptionPane.INFORMATION_MESSAGE);
```

```
        } else {
            String deskNum = table.getValueAt(selectedRow, 1).toString(); //获得选中餐台的编号
            //查看该餐台是否正在被使用
            for (int row = 0; row < openedDeskTable.getRowCount(); row++) {
                if (deskNum.equals(openedDeskTable.getValueAt(row, 1))) {
                    JOptionPane.showMessageDialog(null, "该餐台正在使用，不能删除！", "友情提示",
                            JOptionPane.INFORMATION_MESSAGE);
                    return;                                  //该餐台正在被使用，不能删除，返回
                }
            }
            String infos[] = new String[] {                                  //组织确认信息
                    "确定要删除餐台：",
                    "      台　号：" + deskNum,
                    "      座位数：" + table.getValueAt(selectedRow, 2) .toString() };
            int i = JOptionPane.showConfirmDialog(null, infos, "友情提示",
                    JOptionPane.YES_NO_OPTION);                              //弹出确认提示
            if (i == 0) {                                                    //确认删除
                dao.dDeskByNum(deskNum);                                     //从数据库中删除
                tableModel.setDataVector(dao.sDesk(), columnNameV);          //刷新表格
                int rowCount = table.getRowCount();                          //获得删除后拥有的餐台数
                if (rowCount > 0) {                                          //还拥有餐台
                    if (selectedRow == rowCount)                             //删除的为最后一个餐台
                        selectedRow -= 1;                                    //将选中的餐台前移一行
                    //设置当前选中的餐台
                    table.setRowSelectionInterval(selectedRow, selectedRow);
                }
                JDBC.closeConnection();                                      //关闭数据库连接
            }
        }
    }
});
delButton.setText("删除");
```

2. 实现菜系管理功能

首先实现添加菜系的功能。在执行添加菜系操作时，首先要判断菜系名称的长度是否超出了允许的最大长度，并查看该菜系名称是否已经存在；然后创建一个向量对象，用来封装新添加菜系的信息，并添加到表格中；最后将新添加的菜系信息保存到数据库中。代码如下：

```
final JButton addButton = new JButton();                          //创建添加菜系名称按钮对象
addButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        String sortName = sortNameTextField.getText().trim();     //获得菜系名称，并去掉首尾空格
        if (sortName.equals("")) {                                //查看是否输入了菜系名称
            JOptionPane.showMessageDialog(null, "请输入菜系名称！", "友情提示",
                    JOptionPane.INFORMATION_MESSAGE);
            return;
        }
        //查看菜系名称的长度是否超过了 10 个汉字
```

```
        if (sortName.length() > 10) {
            JOptionPane.showMessageDialog(null, "菜系名称最多只能为 10 个汉字！",
                    "友情提示", JOptionPane.INFORMATION_MESSAGE);
            return;
        }
        if (dao.sSortByName(sortName).size() > 0) {                //查看该菜系名称是否已经存在
            JOptionPane.showMessageDialog(null, "该菜系已经存在！", "友情提示",
                    JOptionPane.INFORMATION_MESSAGE);
            return;
        }
        int row = tableModel.getRowCount();                        //获得当前拥有菜系名称的个数
        Vector newSortV = new Vector();                            //创建一个代表新菜系名称的向量
        newSortV.add(new Integer(row + 1));                        //添加序号
        newSortV.add(sortName);                                    //添加菜系名称
        tableModel.addRow(newSortV);                               //将新菜系名称信息添加到表格中
        table.setRowSelectionInterval(row, row);                   //设置新添加的菜系名称为选中的
        sortNameTextField.setText(null);                           //将“菜系名称”文本框设置为空
        dao.iSort(sortName);                                       //将新添加的菜系名称信息保存到数据库中
        JDBC.closeConnection();                                    //关闭数据库连接
    }
});
addButton.setText("添加");
```

然后实现删除菜系的功能。在执行删除菜系操作之前，首先要判断是否有菜系被选中；然后弹出提示确认是否真的删除；最后执行删除操作，如果删除的是表格中的最后一行，则选中删除后表格中的最后一行，如果删除的不是最后一行，则选中删除后同一位置的表格行。代码如下：

```
final JButton delButton = new JButton();                           //创建删除菜系名称按钮对象
delButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        int row = table.getSelectedRow();                          //获得选中的菜系
        String delSortName = (String) table.getValueAt(row, 1);    //获得选中的菜系名称
        int j = JOptionPane.showConfirmDialog(null, "确定要删除菜系“" + delSortName
                + "”？", "友情提示", JOptionPane.YES_NO_OPTION);//弹出确认提示
        if (j == 0) {                                              //确认删除
            tableModel.removeRow(row);                             //从表格中移除菜系信息
            int rowCount = table.getRowCount();                    //获得删除后拥有的菜系数
            if (rowCount > 0) {
                if (row < table.getRowCount()) {                   //删除的不是位于表格最后的菜系
                    for (int i = row; i < table.getRowCount(); i++) {
                        table.setValueAt(i + 1 + "", i, 0);        //修改位于删除菜系之后的序号
                    }
                    table.setRowSelectionInterval(row, row);  //设置上移到删除行索引的菜系为被选中
                } else {                                           //删除的是位于表格最后的菜系
                    table.setRowSelectionInterval(row - 1, row - 1); //设置当前位于表格最后的菜系被选中
                }
            }
            dao.dSortByName(delSortName);                          //从数据库中删除菜系
            JDBC.closeConnection();                                //关闭数据库连接
```

```
            }
        }
});
delButton.setText("删除");
```

3. 实现菜品管理功能

首先实现添加菜品的功能。在执行添加菜品操作时，首先通过 Java 反射机制验证 4 个文本框是否为空，如果为空则弹出要求填写信息的提示框，并且通过获取组件的标识名称，组织出有针对性的提示信息，还要为空的文本框请求获得焦点，之后再对这些信息进行具体的验证；然后创建一个向量对象，用来封装新添加菜品的信息，并添加到表格中；最后将新添加的菜品信息保存到数据库中。代码如下：

```
final JButton addButton = new JButton();
addButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        //通过 Java 反射获取 MenuDialog 类的所有属性
❶       Field[] fields = MenuDialog.class.getDeclaredFields();
        for (int i = 0; i < fields.length; i++) {
            Field field = fields[i];                                    //获得指定属性
❷           if (field.getType().equals(JTextField.class)) {             //只验证 JTextField 类型的属性
❸               field.setAccessible(true);                              //私有属性必须设置为 true 才允许访问
                JTextField textField = null;                            //声明一个 JTextField 类型的对象
                try {
                    //获得本类中的相应对象
❹                   textField = (JTextField) field.get(MenuDialog.this);
                } catch (Exception exception) {
                    exception.printStackTrace();
                }
                if (textField.getText().trim().equals("")) {            //文本框为空
                    JOptionPane.showMessageDialog(null, "请填写商品 “"
❺                       + textField.getName() + "” ！ ", "友情提示",
                        //弹出需要输入信息的提示
                        JOptionPane.INFORMATION_MESSAGE);
❻                   textField.requestFocus();                           //令文本框获得焦点
                    return;                                             //返回
                }
            }
        }
        if (sortComboBox.getSelectedIndex() == 0) {                     //单独验证下拉列表框
            JOptionPane.showMessageDialog(null, "请选择商品所属 “菜系” ！ ", "友情提示",
                    JOptionPane.INFORMATION_MESSAGE);
            return;
        }
        String menu[] = new String[6];                                  //创建一个数组，用来保存菜品信息
        menu[0] = numTextField.getText().trim();                        //获得菜品编号
        menu[1] = nameTextField.getText().trim();                       //获得菜品名称
        menu[2] = codeTextField.getText().trim();                       //获得菜品助记码
        menu[3] = sortComboBox.getSelectedItem().toString();            //获得菜品所属菜系
```

```
        menu[4] = unitTextField.getText().trim();                    //获得菜品单位
        menu[5] = unitPriceTextField.getText().trim();               //获得菜品单价
        if (menu[1].length() > 10) {
            JOptionPane.showMessageDialog(null, "菜品名称最多只能为 10 个汉字！",
                    "友情提示", JOptionPane.INFORMATION_MESSAGE);
            nameTextField.requestFocus();
            return;
        }
        if (menu[2].length() > 10) {
            JOptionPane.showMessageDialog(null, "助记码最多只能为 10 个字符！",
            "友情提示", JOptionPane.INFORMATION_MESSAGE);
            codeTextField.requestFocus();
            return;
        }
        if (menu[4].length() > 2) {
            JOptionPane.showMessageDialog(null, "单位最多只能为 2 个汉字！",
            "友情提示", JOptionPane.INFORMATION_MESSAGE);
            unitTextField.requestFocus();
            return;
        }
        if (!Validate.execute("[1-9]{1}[0-9]{0,3}", menu[5])) {
            String infos[] = { "单价输入错误！", "单价必须在 1～9999" };
            JOptionPane.showMessageDialog(null, infos, "友情提示", JOptionPane.INFORMATION_ MESSAGE);
            unitPriceTextField.requestFocus();
            return;
        }
        if (dao.sMenuByName(menu[1]) != null) {
            JOptionPane.showMessageDialog(null, "该菜品已经存在！", "友情提示",
                    JOptionPane.INFORMATION_MESSAGE);
            nameTextField.requestFocus();
            return;
        }
        int row = tableModel.getRowCount();                          //获得当前拥有的菜品数量
        Vector newMenuV = new Vector();
        newMenuV.add(row + 1);                                       //添加序号
        for (int i = 0; i < menu.length; i++) {
            newMenuV.add(menu[i]);                                   //添加菜品信息
        }
        tableModel.addRow(newMenuV);                                 //将新菜品添加到表格中
        table.setRowSelectionInterval(row, row);                     //选中新添加的菜品
        //获得所属菜系
        Vector sortVector = (Vector) dao.sSortByName(menu[3]).get(0);
        menu[3] = sortVector.get(1).toString();                      //设置菜系主键
        dao.iMenu(menu);                                             //将新菜品信息保存到数据库
        JDBC.closeConnection();                                      //关闭数据库连接
    }
});
addButton.setText("添加");
```

☑ ❶ getDeclaredFields()：该方法返回一个 Field 型数组，在数组中包含调用类的所有属性，包括公共、保护、默认（包）访问和私有字段，但不包括继承的字段。

☑ ❷ getType()：该方法返回一个 Class 对象，它标识了此属性的类型，如果只想对类型为 JTextField 的属性进行操作，可以通过代码 getType().equals(JTextField.class)查看属性的类型是否为 JTextField。

☑ ❸ setAccessible(boolean accessible)：在默认情况下通过 Java 反射是不允许访问私有属性的，如果通过该方法将 accessible 属性设置为 true，则允许访问私有属性。

☑ ❹ get(Object obj)：该方法用来返回指定对象上此 Field 表示的字段的值。如果该值为基本类型，将自动将其包装为封装类型。MenuDialog.this 代表获得本类中此 Field 表示的字段的值。

☑ ❺ getName()：该方法用来获得组件的名称，主要是为增强软件的人性化特点而设计的。例如这里就利用这个人性化的设计，很方便地弹出了准确的提示信息。

☑ ❻ requestFocus()：该方法用来为组件请求获得焦点，主要是为增强软件的人性化特点而设计的。例如这里就利用这个人性化的设计，很方便地令不符合条件的文本框获得焦点。

然后实现删除菜品的功能。在执行删除菜品操作之前，首先要判断是否存在被选中的菜品；然后弹出提示确认是否真的删除；最后，如果删除的不是表格的最后一行，还要修改其后未删除菜品的序号。代码如下：

```
final JButton delButton = new JButton();
delButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
❶       int row = table.getSelectedRow();                          //获得选中的菜品
        String delMenuName = table.getValueAt(row, 2).toString();
        String info = "确定要删除菜品 “" + delMenuName + "” ? ";
        int j = JOptionPane.showConfirmDialog(null, info, "友情提示",
                JOptionPane.YES_NO_OPTION);                        //弹出确认提示框
        if (j == 0) {                                              //确认删除
❷           tableModel.removeRow(row);                             //从表格中移除菜品信息
            int rowCount = table.getRowCount();                    //获得删除后拥有的菜品数
            if (rowCount > 0) {                                    //还拥有菜品
                if (row < table.getRowCount()) {                   //删除的不是位于表格最后的菜系
                    for (int i = row; i < table.getRowCount(); i++) {
                        table.setValueAt(i + 1 + "", i, 0);        //修改位于删除菜系之后的序号
                    }
                    //设置上移到删除行索引的菜系为被选中
                    table.setRowSelectionInterval(row, row);
                } else {
                    //设置当前位于表格最后的菜系被选中
                    table.setRowSelectionInterval(row - 1, row - 1);
                }
            }
            dao.dMenuByName(delMenuName);                          //从数据库中删除菜品
            JDBC.closeConnection();                                //关闭数据库连接
        }
    }
```

```
});
delButton.setText("删除");
```

☑ ❶ getSelectedRow()：该方法用来获得表格中选中行的索引，返回值为 int 型。如果没有选中行，则返回-1；如果有多行被选中，则返回所有选中行中索引值最小的索引。

☑ ❷ removeRow(int rowIndex)：该方法用来从表格中移除位于指定索引位置的行，表格的行索引从 0 开始。

21.11.4 单元测试

当测试菜品管理功能时，在不填写任何菜品信息的情况下直接单击“添加”按钮，弹出如图 21.42 所示的提示信息。

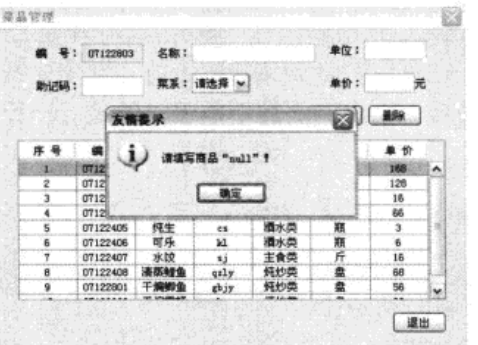


图 21.42 通过 Java 反射验证是否为空弹出的错误提示

这是因为没有为文本框组件设置标识名称，默认情况下文本框组件的标识名称为空，所以在通过文本框组件的 getName()方法获得文本框组件的标识名称时才是 null，解决的办法是通过 setName(String name)方法为文本框组件设置标识名称。代码如下：

```
nameTextField = new JTextField();                    //创建“名称”文本框对象
nameTextField.setName("名称");                       //为“名称”文本框设置标识名称
codeTextField = new JTextField();                    //创建“助记码”文本框对象
codeTextField.setName("助记码");                     //为“助记码”文本框设置标识名称
unitTextField = new JTextField();                    //创建“单位”文本框对象
unitTextField.setName("单位");                       //为“单位”文本框设置标识名称
unitPriceTextField = new JTextField();               //创建“单价”文本框对象
unitPriceTextField.setName("单价");                  //为“单价”文本框设置标识名称
```

21.12 开发技巧与难点分析

视频讲解：光盘\TM\lx\21\开发技巧与难点分析.exe

作为一个软件开发人员，在设计开发应用软件的过程中，要时刻从软件使用者的角度出发，力求

开发出一款功能实用、界面简单、操作人性化并且智能化的产品，只有这样的产品，才更容易被用户接受。

笔者在开发本系统的过程中，就是严格按照这些要求实施的。主要有以下几点：

1．通过输入少量内容就可以快速获取相关产品

通过 KeyListener 监听器，可以很方便地捕获各种键盘事件。KeyListener 监听器通过 3 个接口方法捕获 3 种不同类型的键盘事件：keyPressed(KeyEvent e)方法用来捕获键盘中的某个按键被按下的事件，当某个按键被按下时，将执行该方法；keyReleased(KeyEvent e)方法用来捕获键盘中的某个按键被释放的事件，当某个按键被释放时，将执行该方法；keyTyped(KeyEvent e)方法用来捕获键入键盘中的某个键的事件，当键入某个键时，将执行该方法。

“按下键”和“释放键”事件是低级别事件，它们依赖于平台和键盘布局。只要按下或释放按键就会生成这些事件，这些事件是获取不生成字符输入的键（如动作键、组合键等）的唯一方式。通过 KeyEvent 类的 getKeyCode()方法可以获取代表按下或释放键的虚拟键码，虚拟键码用于报告按下了键盘上的哪个键，而不是通过一个或多个按键组合所生成的字符（如 A 是由 Shift+A 生成的）。

“键入键”事件是高级别事件，通常情况下不依赖于平台或键盘布局。在输入 Unicode 字符时会生成此类事件，不生成 Unicode 字符的键是不会生成键入键事件的（如动作键、组合键等）。最简单的情况是按下单个键（如 A）将产生键入键事件，但经常是通过一系列按键（如 Shift+A）来产生字符，并且按下键事件和键入键事件的映射关系可能是多对一或多对多的。键释放时通常情况下不需要生成一个键入键事件，但是在某些情况下，只有释放某个键才会生成键入键事件（如在 Windows 中通过 Alt-Numpad 方法来输入 ASCII 序列）。

利用组件的 addKeyListener(KeyListener keyListener)方法可以将该监听器对象注册到组件中，这样在按下、释放或键入键生成键盘事件时，将调用监听器对象中的相关方法，并传递过来一个 KeyEvent 对象。

2．人性化控制商品数量 focusLost(FocusEvent e)

通过 FocusListener 监听器，可以很方便地捕获关于焦点的事件。FocusListener 监听器通过两个接口方法捕获两种不同类型的焦点事件：focusGained(FocusEvent e)方法用来捕获组件获得焦点的事件，当其监听的组件获得焦点时，将执行该方法；focusLost(FocusEvent e)方法用来捕获组件失去焦点的事件，当其监听的组件失去焦点时，将执行该方法。

焦点事件分为两个级别，分别是持久性的和暂时性的。如果焦点直接从一个组件移动到另一个组件，如通过调用 requestFocus()方法，或者用户通过 Tab 键遍历组件时，则为持久性焦点更改事件。如果是由于另一个操作间接引起的组件暂时失去焦点，如释放窗口或拖动滚动条，则为暂时性焦点更改事件。在这种情况下，一旦该操作结束，将自动恢复到原始焦点状态。对于释放窗口的情况，只要重新激活窗口就能恢复到原始焦点状态。持久性焦点事件和暂时性焦点事件通过 FOCUS_GAINED 和 FOCUS_LOST 区分，可以通过 isTemporary()方法判断事件的级别。

利用组件的 addFocusListener (FocusListener focusListener)方法可以将该监听器对象注册到组件中，这样在组件获得或失去焦点时，将调用监听器对象中的相关方法，并传递过来一个 FocusEvent 对象。

3．系统自动结账

通过 TableModelListener 监听器，可以很方便地捕获由表格模型发生变化产生的事件，包括向表格模型中添加行、从表格模型中移除行，以及修改某一单元格的值。TableModelListener 监听器只提供了

一个接口方法 tableChanged(TableModelEvent e)。通过 TableModelEvent 对象可以判断触发此次事件的具体原因，例如通过 getType()方法的返回值判断是由向表格模型中添加行触发的，还是由从表格模型中移除行或修改某一单元格的值触发的。当返回值为静态常量 INSERT 时，说明是由向表格模型中添加行触发的；当返回值为静态常量 UPDATE 时，说明是由从表格模型中移除行触发的；当返回值为静态常量 DELETE 时，则是由修改某一单元格的值触发的。

利用组件的 addTableModelListener(TableModelListener tableModelListener)方法可以将该监听器对象注册到组件中，这样在表格模型发生变化时，将调用监听器对象中的 tableChanged(TableModelEvent e)方法，并传递过来一个 TableModelEvent 对象。

21.13 使用 Visio 逆向生成数据库 E-R 图

视频讲解：光盘\TM\lx\21\使用 Visio 逆向生成数据库 E-R 图.exe

利用 Visio 也可以根据现有数据库生成数据库 E-R 图，具体步骤如下：

（1）在 Visio 中新建一个“数据库”/“数据库模型图”，如图 21.43 所示。

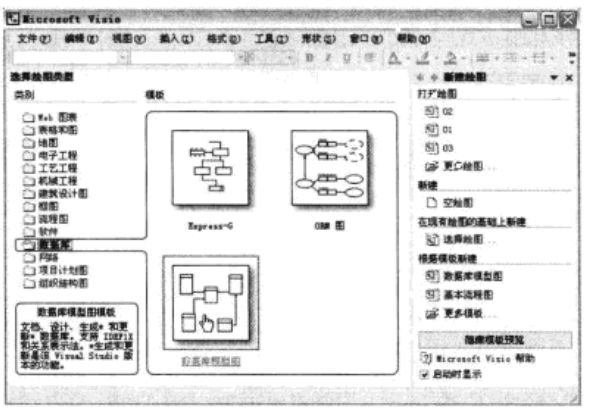


图 21.43 新建数据库模型图

（2）新建数据库模型图后，在菜单栏中将添加一个“数据库”菜单，选择“数据库”/“反向工程”命令，打开如图 21.44 所示的对话框。

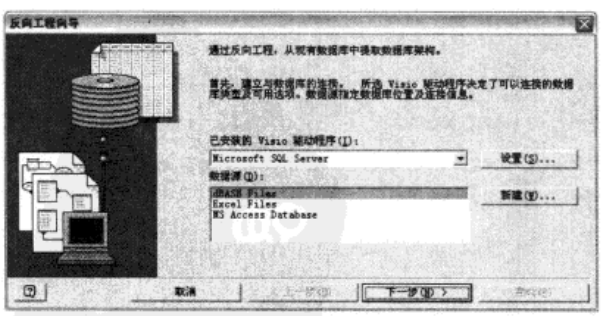


图 21.44 选择数据库连接

（3）图 21.44 所示的对话框用来选择数据库连接，如果没有相应的连接，则单击右侧的“新建”按钮建立相应的连接，这里是为本章数据库建立连接，如图 21.45 和图 21.46 所示。

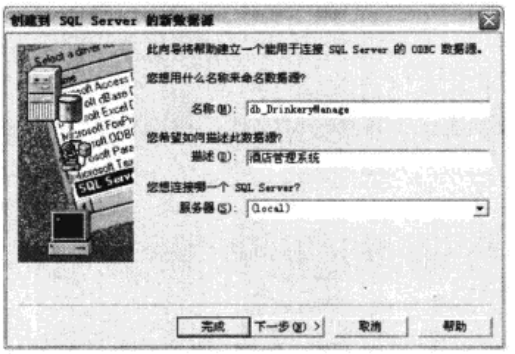


图 21.45　设置数据库连接信息

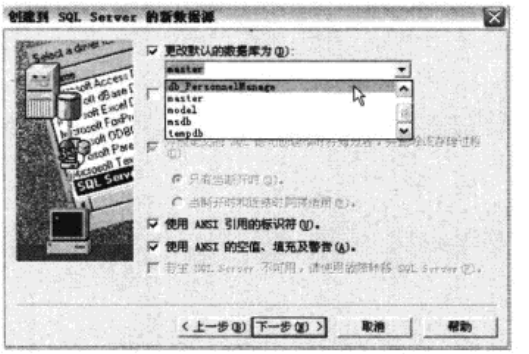


图 21.46　选择本章使用的数据库

（4）成功连接数据库后，将打开如图 21.47 所示的对话框，从中选择执行反向工程的表和视图，单击“下一步”按钮。

图 21.47　选择表和视图

（5）在打开的如图 21.48 所示的对话框中可以选择是否自动添加反向工程项目的形状到当前页，这里选中“是，将形状添加到当前页”单选按钮，然后单击“完成”按钮，生成的数据库 E-R 图如图 21.49 所示。

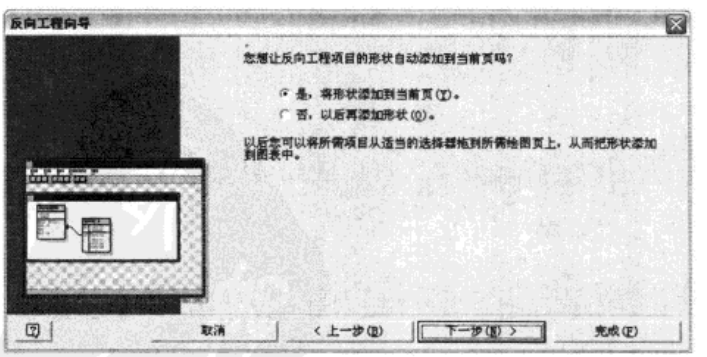


图 21.48　选择是否自动添加反向工程项目的形状到当前页

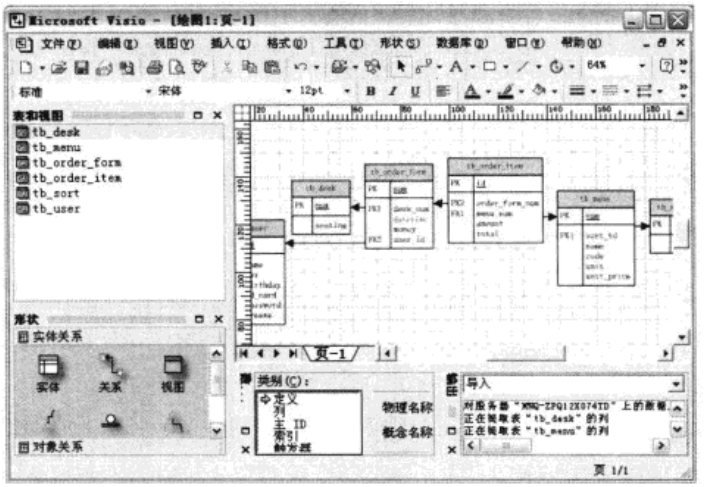


图 21.49　生成的数据库 E-R 图

21.14　本 章 小 结

本章通过一个典型的酒店管理系统，既为读者展示了酒店管理系统的业务流程，又为读者展示了酒店管理系统的基本开发思路和实施方法。通过对本章的学习，读者可以了解到 Java 应用程序的开发流程；酒店管理系统的业务流程和软件结构；快速获取商品的实现方法；人性化控制商品数量的实现方法；系统自动结账的实现方法；另外，在本系统中还提供了典型的结账报表功能。

希望读者学完本章后能够做一个仔细的总结，将本章中涉及的知识进行整理，真正成为自己的收获。