软件开发

羊皮卷

DUANTIAN KAIFA YANGPIJUAN

程序员是怎样炼成的？在持续激励中磨砺，在反复练习中成长，Java疯狂特训内容，尽在本书中……

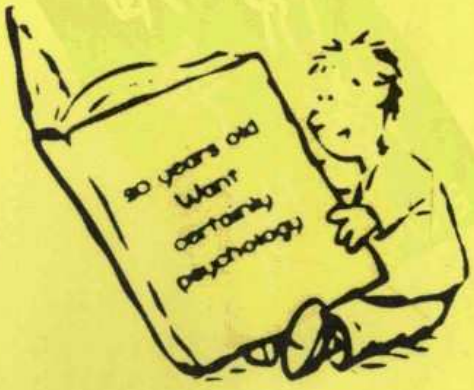
学通Java的24堂课

李钟尉　李伟 ◎ 等编著

120集大型多媒体教学视频

377个中小实例训练　580余段源码分析

206个应用模块精解　6大项目案例展示



DVD大型多媒体光盘

120集教学视频：120集（358段）多媒体教学视频，听程序员现场讲解

377个中小实例：夯实必备知识，强化基本功训练

580余源码分析：寻找编程感觉，培养编程思想

206个应用模块：激发学习兴趣，突出开发实战

6大项目案例：体验项目开发过程，积累项目开发经验

清华大学出版社

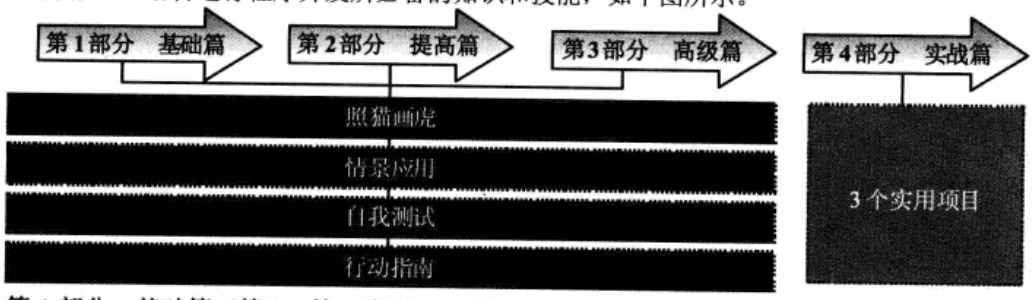
前　言

Preface

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。Java 自从面世以来一直以易学易用、功能强大的特点得到广泛的应用，其强大的跨平台特性使 Java 程序可以运行在任何一个系统平台上，甚至是手持电话、商务助理等电子产品，都可以运行 Java 程序，真正做到“一次编写，到处运行”。Java 可以编写桌面应用程序、Web 应用程序、分布式系统和嵌入式系统应用程序等，这使得 Java 成为应用范围最广阔的开发语言。

本书内容

学、练、用到精通只需 24 堂课。本书从初中级用户的角度进行科学合理的设计，通过 24 堂课全面地讲述了使用 Java 语言进行程序开发所必备的知识和技能，如下图所示。



第 1 部分　基础篇（第 1～第 7 堂课）：讲述了 Java 概述、使用 IDE 集成开发工具、Java 语言基础知识、流程控制语句、数组应用、面向对象编程、字符串处理。这些都是程序语言的基础，学通了这些内容后，再学其他篇相信一定会很简单。

第 2 部分　提高篇（第 8～第 14 堂课）：讲述了使用 Visual Basic 6.0 进行应用程序开发的各种常用技术，包括类的继承与多态特性、其他类特性与异常处理、Swing 编程基础、多线程编程、事件处理的应用、常用工具类、常用集合类。通过本篇的学习，读者能够开发小型应用程序，掌握面向对象的高级特性、学会任务多线程处理和使用集合对象保存数据等。

第 3 部分　高级篇（第 15～第 21 堂课）：讲述了数据库编程应用、输入输出流与文件、网络程序设计、表格控件的应用、树控件的应用、其他高级控件、图形绘制技术。通过本篇的学习，读者可以实现文件操作、图形图像处理、开发小型网络程序和数据库应用程序。

第 4 部分　实战篇（第 22～第 24 堂课）：讲述了进销存管理系统、企业内部通讯管理系统和图书馆管理系统 3 个完整的项目实例设计全过程，积累项目开发经验。

本书特点

- **配备120集（358段）多媒体教学视频讲解。**

  本书 DVD 光盘提供了覆盖全书的语音视频讲解，读者可以通过视频快速、直观、轻松地学习。

- **每一堂课都结合“照猫画虎”、“情景应用”。**
  为了增强读者的动手能力，并激发学习兴趣，本书提供了“照猫画虎”和“情景应用”栏目，根据实例模仿着去做是学习编程的最快方式。
- **每一堂课都结合励志故事和“行动指南”，时刻进行激励和鼓舞。**
  我们认为学习中尤其是最初的一段时间非常有必要不断地对学习者进行激励和鼓舞，让他们坚持下来是至关重要的，因此在书中不间断地用一些励志故事和行动指南鼓舞其信心。
- **光盘提供了本书的所有代码，即使只有一行。**
  本书光盘不仅提供了所有实例的源程序，还提供了书中所有示例的源代码，哪怕只有一行。读者可以直接复制，以提高学习效率。
- **本书所有习题和实战都给出了答案，部分习题还有解析，读者可以对照查阅。**

读者对象

- ☑ 有志于软件开发的初学者
- ☑ 准备从事软件开发的求职者
- ☑ 初中级程序开发人员
- ☑ 高等院校计算机相关专业的老师和学生
- ☑ 参与毕业设计的学生
- ☑ 程序测试及维护人员

本书作者

本书由明日科技组织编写，参加编写的程序员有李钟尉、李伟、王国辉、赛奎春、潘凯华、刘欣、李慧、陈丹丹、高春艳、孙秀梅、杨丽、刘玲玲、朱晓、刘燕、陈英、李鑫、李贺、肖鑫、张丽娜、沈博、刘冠男、曹飞飞、李丽、聂喜婷、王明昭、张英豪、王小科、赵会东、白伟明等。

由于水平有限，书中疏漏和不足之处在所难免，恳请广大读者朋友批评指正。

技术支持与服务

有关本书的问题，读者可以通过如下方式与我们沟通：

- 登录本技术网站：www.mingribook.com，查阅相关问题或者留言。
- 通过邮箱：tmoonbook@sina.com 或 th_press@263.net。
- 加入 QQ：200958607。

最后，感谢您选择本书，希望本书能成为您编程路上的领航者。

祝读书快乐！

特别提醒：

亲爱的读者朋友，由于近期纸张价格和印制成本大幅上涨，为不增加读者朋友的负担，又不减少书的内容，本书的最后几章内容不得不放在配书光盘中（参见目录），由此给您带来了不便，在此深表歉意。

编　者

目　录

Contents

第 1 部分　基础篇

第 1 堂课　Java 概述 3

视频讲解：31 分钟

1.1　Java 简介 4

1.1.1　Java 发展历史 4

1.1.2　Java 的几个版本 5

1.1.3　Java 应用领域 6

1.1.4　Java 项目成功案例 6

1.1.5　怎样学好 Java 8

1.2　环境搭建 8

1.2.1　JDK 下载 8

1.2.2　JDK 安装 10

1.2.3　配置环境变量 11

1.3　编写第一个 Java 程序 14

1.3.1　在记事本中输入代码 14

1.3.2　保存代码为源代码文件 15

1.3.3　编译并运行 Java 文件 16

1.4　照猫画虎——基本功训练 16

1.4.1　基本功训练 1——更换 JAVA_HOME 环境变量 16

1.4.2　基本功训练 2——提取 Path 变量值创建新环境变量 17

1.4.3　基本功训练 3——输出字符表情 17

1.4.4　基本功训练 4——输出“*”字符组成的三角形 18

1.5　情景应用——拓展与实践 19

1.5.1　情景应用 1——计算两个整数的和 19

1.5.2　情景应用 2——输出当前日期 19

1.5.3　情景应用 3——在控制台接收用户输入数字 20

1.5.4　情景应用 4——计算用户输入数字的乘积 21

1.6　自我测试 22

1.7　行动指南 23

1.8　成功可以复制——“盖茨第二” 23

第 2 堂课　使用 IDE 集成开发工具 25

视频讲解：128 分钟

2.1　Eclipse 简介 26

2.2　Eclipse 下载与安装 26

2.2.1　Eclipse 分类 26

2.2.2　下载 Eclipse 26

2.2.3　安装中文语言包 28

2.2.4　启动 Eclipse 29

2.3　安装界面设计器 30

2.4　熟悉 Eclipse 32

2.4.1　Eclipse 工作台 32

2.4.2　透视图与视图 33

2.4.3　包资源管理器视图 33

2.4.4　编辑器视图 33

2.4.5　控制台 34

2.4.6　大纲视图 34

2.4.7　问题视图 34

2.4.8　透视图工具栏 35

2.4.9　视图工具栏 35

2.4.10　快速视图工具栏 35

2.5　如何开发 Java 程序 35

2.5.1　新建 Java 项目 36

2.5.2　新建 Java 类 36

2.5.3　编写 Java 代码 37

2.5.4　运行 Java 程序 39

2.6　如何开发桌面应用程序 39

2.6.1　新建 Swing GUI 窗体 40

2.6.2 向窗体中添加 GUI 控件 41
2.6.3 实现控件的事件监听器 43
2.6.4 运行 GUI 程序 43
2.7 如何进行程序调试 43
2.7.1 设置断点 44
2.7.2 以调试方式运行 Java 程序 44
2.7.3 程序调试 44
2.8 了解项目类库 45
2.9 照猫画虎——基本功训练 46
2.9.1 基本功训练 1——添加 JAR 文件到构建路径 46
2.9.2 基本功训练 2——添加依赖项目到构建路径 46
2.9.3 基本功训练 3——创建用户自定义类库 47
2.9.4 基本功训练 4——为项目添加类库 48
2.9.5 基本功训练 5——为项目添加类文件夹 49
2.10 情景应用——拓展与实践 50
2.10.1 情景应用 1——为窗体添加文本框控件 50
2.10.2 情景应用 2——接收用户在控制台的输入 50
2.10.3 情景应用 3——编写一个信息提示窗体 51
2.10.4 情景应用 4——控制窗体位置 52
2.10.5 情景应用 5——编写 UI 界面的加法计算器 54
2.11 自我测试 55
2.12 行动指南 56
2.13 成功可以复制——迅雷创始人邹胜龙 57

第 3 堂课 Java 语言基础知识 59
视频讲解：178 分钟
3.1 Java 主类结构 60
3.1.1 定义类包 60
3.1.2 定义类 60
3.1.3 导入 API 类库 60
3.1.4 定义变量 61
3.1.5 定义主方法 61
3.1.6 运行主类 61
3.2 代码注释 61
3.2.1 单行注释 61
3.2.2 多行注释 61
3.2.3 JavaDoc 文档注释 62
3.3 变量与常量 62
3.3.1 Java 保留字 62
3.3.2 定义变量 63
3.3.3 定义常量 64
3.3.4 常量与变量的有效范围 64
3.4 数据类型 65
3.4.1 计算机常用数制 66
3.4.2 整数类型 67
3.4.3 浮点类型 69
3.4.4 字符类型 69
3.4.5 布尔类型 71
3.5 数据类型转换 71
3.5.1 自动类型转换 71
3.5.2 强制类型转换 73
3.6 运算符 74
3.6.1 赋值运算符 74
3.6.2 算术运算符 75
3.6.3 自增和自减运算符 76
3.6.4 比较运算符 76
3.6.5 逻辑运算符 77
3.6.6 位运算符 78
3.6.7 三元运算符 80
3.6.8 运算符优先级 81
3.7 编码规范 81
3.8 照猫画虎——基本功训练 82
3.8.1 基本功训练 1——合理设定变量类型 82
3.8.2 基本功训练 2——变量自增运算 83
3.8.3 基本功训练 3——类型转换 84
3.8.4 基本功训练 4——运算符的应用 84
3.8.5 基本功训练 5——求最大值 85
3.8.6 基本功训练 6——位移实现数字乘以 2 的 8 次幂 86
3.9 情景应用——拓展与实践 86
3.9.1 情景应用 1——输出长方形面积 86
3.9.2 情景应用 2——输出字节 bit 位的值 87
3.9.3 情景应用 3——判断奇偶数 88

3.9.4 情景应用 4——简单数据加密 89
3.9.5 情景应用 5——判断闰年 89
3.9.6 情景应用 6——异或运算实现变量值交换 90
3.10 自我测试 91
3.11 行动指南 92
3.12 成功可以复制——知识改变命运、科技改变生活 93

第 4 堂课 流程控制语句 95
视频讲解：113 分钟
4.1 复合语句 96
4.2 条件语句 96
4.2.1 if 条件判断语句 97
4.2.2 if 语句的嵌套 100
4.2.3 switch 分支语句 101
4.2.4 if 语句和 switch 语句的区别 105
4.3 循环控制 105
4.3.1 for 循环 105
4.3.2 while 循环 107
4.3.3 do…while 循环 109
4.3.4 print 与 println 方法 110
4.3.5 循环嵌套 111
4.4 循环的跳转 112
4.4.1 continue 跳转语句 112
4.4.2 break 跳转语句 114
4.5 照猫画虎——基本功训练 116
4.5.1 基本功训练 1——计算阶乘 116
4.5.2 基本功训练 2——计算素数 116
4.5.3 基本功训练 3——遍历 21 世纪的闰年 117
4.5.4 基本功训练 4——输出空心的菱形 118
4.5.5 基本功训练 5——循环的生命力 119
4.6 情景应用——拓展与实践 120
4.6.1 情景应用 1——把用户输入的字母变成大写 120
4.6.2 情景应用 2——应用 switch 语句实现生肖查询 121
4.6.3 情景应用 3——验证用户登录信息 122
4.6.4 情景应用 4——判断用户输入月份的季节 123
4.6.5 情景应用 5——汉字与区位码的转换 124
4.7 自我测试 126
4.8 行动指南 128
4.9 成功可以复制——中国通信设备行业的领跑者任正非 129

第 5 堂课 数组应用 131
视频讲解：90 分钟
5.1 一维数组的创建与应用 132
5.1.1 创建一维数组 132
5.1.2 初始化一维数组 133
5.1.3 遍历一维数组 133
5.2 二维数组的创建与应用 134
5.2.1 创建二维数组 134
5.2.2 初始化二维数组 135
5.2.3 遍历二维数组 135
5.3 多维数组的创建 136
5.3.1 数组的创建 136
5.3.2 初始化数组 137
5.3.3 遍历三维数组 137
5.4 照猫画虎——基本功训练 139
5.4.1 基本功训练 1——填充替换数组元素 139
5.4.2 基本功训练 2——对数组进行排序 140
5.4.3 基本功训练 3——复制数组 141
5.4.4 基本功训练 4——Foreach 循环遍历数组 142
5.4.5 基本功训练 5——找出数组中最大值 143
5.5 情景应用——拓展与实践 144
5.5.1 情景应用 1——使用冒泡排序 144
5.5.2 情景应用 2——使用直接选择排序法 146
5.5.3 情景应用 3——使用快速排序法 147
5.5.4 情景应用 4——反转数组中元素的顺序 149
5.5.5 情景应用 5——利用数组随机抽取幸运观众 150
5.6 自我测试 151

5.7 行动指南 152
5.8 成功可以复制——软件业的华人教父王嘉廉 153

第6堂课 面向对象编程 155
视频讲解：154分钟
6.1 类的定义 156
6.2 包 156
6.2.1 类包简介 156
6.2.2 定义类的包 156
6.2.3 类包的导入 158
6.2.4 静态导入 158
6.3 成员变量与成员方法 159
6.3.1 成员变量 159
6.3.2 成员方法 160
6.3.3 方法的返回值 160
6.4 对象的创建与使用 162
6.4.1 创建类的对象 162
6.4.2 访问对象的属性 162
6.4.3 执行对象的行为 162
6.4.4 对象的销毁 164
6.5 类成员与类方法 164
6.5.1 类变量 164
6.5.2 类方法 165
6.6 照猫画虎——基本功训练 167
6.6.1 基本功训练1——用构造方法完成初始化 167
6.6.2 基本功训练2——用静态代码块初始化 170
6.6.3 基本功训练3——this关键字的用途 171
6.6.4 基本功训练4——方法参数传递 172
6.6.5 基本功训练5——方法的重载 174
6.7 情景应用——拓展与实践 177
6.7.1 情景应用1——整数进制转换器 177
6.7.2 情景应用2——获取数字的取值范围 178
6.7.3 情景应用3——ASCII编码查看器 179
6.7.4 情景应用4——构造方法初始化员工对象 180
6.7.5 情景应用5——将字符串转换成整数 182
6.8 自我测试 183
6.9 行动指南 185
6.10 成功可以复制——创造互联网搜索时代 186

第7堂课 字符串处理 189
视频讲解：116分钟
7.1 String类 190
7.1.1 声明字符串 190
7.1.2 构建字符串对象 190
7.2 连接字符串 191
7.2.1 连接多个字符串 191
7.2.2 连接其他数据类型 191
7.3 获取字符串信息 192
7.3.1 获取字符串长度 192
7.3.2 字符串查找 193
7.3.3 获取指定索引位置的字符 194
7.4 格式化字符串 194
7.4.1 日期和时间字符串格式化 194
7.4.2 常规类型格式化 197
7.5 字符串生成器 198
7.6 使用正则表达式 200
7.7 照猫画虎——基本功训练 202
7.7.1 基本功训练1——获取子字符串 202
7.7.2 基本功训练2——字符串替换 203
7.7.3 基本功训练3——判断字符串的开始与结尾 204
7.7.4 基本功训练4——字母大小写转换 205
7.7.5 基本功训练5——字符串分割 206
7.8 情景应用——拓展与实践 207
7.8.1 情景应用1——字符串数组排序 207
7.8.2 情景应用2——格式化当前日期 208
7.8.3 情景应用3——正则表达式验证IP地址 210
7.8.4 情景应用4——正则表达式鉴别非法电话号码 211
7.8.5 情景应用5——获取字符串中汉字的个数 212

7.9　自我测试 213
7.10　行动指南 214
7.11　成功可以复制——IT 风云人物
鲍岳桥 215

第 2 部分　提高篇

第 8 堂课　类的继承与多态特性 219
视频讲解：172 分钟
8.1　类的继承 220
8.2　方法的重写与 Super 关键字 222
8.3　修饰符 224
8.3.1　public 修饰符 224
8.3.2　private 修饰符 224
8.3.3　protected 修饰符 225
8.3.4　默认权限修饰符 226
8.4　封装 227
8.4.1　把属性隐藏 227
8.4.2　定义设置器 227
8.4.3　定义访问器 228
8.4.4　使用 Eclipse 完成封装 228
8.5　Object 类 230
8.5.1　equals 方法 230
8.5.2　toString 方法 231
8.6　照猫画虎——基本功训练 232
8.6.1　基本功训练 1——多态数组 232
8.6.2　基本功训练 2——参数多态 234
8.6.3　基本功训练 3——抽象类实现几何面积计算 238
8.6.4　基本功训练 4——通过接口实现多态 241
8.6.5　基本功训练 5——对象的转型与类型判断 243
8.7　情景应用——拓展与实践 246
8.7.1　情景应用 1——单例模式的应用 246
8.7.2　情景应用 2——简单的汽车销售商场 247
8.7.3　情景应用 3——重新定义对象的等式判断 248
8.7.4　情景应用 4——重新定义对象的字符串表现方式 250
8.7.5　情景应用 5——重新计算对象的哈希码 ... 251
8.8　自我测试 252
8.9　行动指南 254
8.10　成功可以复制——中国第一程序员
求伯君 255
第 9 堂课　其他类特性与异常处理 257
视频讲解：70 分钟
9.1　final 特性 258
9.1.1　final 变量 258
9.1.2　final 方法 258
9.1.3　final 类 259
9.2　内部类 259
9.2.1　成员内部类 259
9.2.2　获取外部类的引用 261
9.2.3　局部内部类 262
9.2.4　匿名内部类 263
9.2.5　静态内部类 263
9.3　异常处理 264
9.3.1　什么是异常 264
9.3.2　捕获异常 264
9.4　抛出异常 266
9.4.1　向上抛出异常 266
9.4.2　使用 throw 关键字抛出异常 267
9.5　运行时异常 268
9.6　异常的使用原则 269
9.7　照猫画虎——基本功训练 269
9.7.1　基本功训练 1——使用普通内部类修改窗体背景色 269
9.7.2　基本功训练 2——使用局部内部类实现闹钟 270
9.7.3　基本功训练 3——使用匿名内部类实现图片浏览器 271
9.7.4　基本功训练 4——演示类未发现异常 272
9.7.5　基本功训练 5——自定义异常类 273

9.8 情景应用——拓展与实践 274
9.8.1 情景应用 1——实现文字的跑马灯效果 ... 274
9.8.2 情景应用 2——组装电脑 275
9.8.3 情景应用 3——跟随鼠标移动的文字 277
9.8.4 情景应用 4——鼠标经过时改变文本颜色 277
9.8.5 情景应用 5——在鼠标上显示当前时间 ... 278
9.9 自我测试 279
9.10 行动指南 281
9.11 成功可以复制——80 后新贵、泡泡网 CEO 李想 282

第 10 堂课 Swing 编程基础 283
视频讲解：153 分钟
10.1 JFrame 窗体控件 284
10.2 常用布局管理器 285
10.2.1 绝对布局 285
10.2.2 流式布局管理器 286
10.2.3 边界布局管理器 287
10.2.4 网格布局管理器 288
10.2.5 网格组布局管理器 289
10.3 常用面板 294
10.3.1 JPanel 面板 294
10.3.2 JScrollPane 面板 295
10.4 标签控件与图标 296
10.4.1 标签的使用 296
10.4.2 图标的使用 296
10.5 按钮控件 297
10.5.1 普通按钮控件 297
10.5.2 单选按钮控件 298
10.5.3 复选框控件 299
10.6 文本控件 300
10.6.1 文本框控件 300
10.6.2 密码框控件 301
10.6.3 文本域控件 302
10.7 列表控件 304
10.7.1 下拉列表框控件 304
10.7.2 列表框控件 305
10.8 进度条 306
10.9 照猫画虎——基本功训练 307
10.9.1 基本功训练 1——设置窗体背景颜色为淡蓝色 307
10.9.2 基本功训练 2——修改密码框的回显字符 308
10.9.3 基本功训练 3——查看图标的信息 309
10.9.4 基本功训练 4——修改列表项选择模式 309
10.9.5 基本功训练 5——监听列表项单击事件 310
10.10 情景应用——拓展与实践 311
10.10.1 情景应用 1——设置窗体的背景图片 ... 311
10.10.2 情景应用 2——震动效果的提示信息 ... 312
10.10.3 情景应用 3——实现计算器程序界面 ... 313
10.10.4 情景应用 4——实现用户注册界面 314
10.10.5 情景应用 5——测试密码是否正确 315
10.11 自我测试 316
10.12 行动指南 317
10.13 成功可以复制——“杀毒王”王江民 317

第 11 堂课 多线程编程 319
视频讲解：87 分钟
11.1 线程基础 320
11.1.1 线程与进程的异同 320
11.1.2 创建无返回值线程 320
11.1.3 线程生命周期详解 322
11.1.4 线程各种属性介绍 323
11.2 线程控制 324
11.2.1 线程的休眠 324
11.2.2 线程的插队 325
11.2.3 线程的停止 326
11.3 线程同步 327
11.3.1 需要同步的原因 327
11.3.2 同步方法的使用 328
11.3.3 使用同步块减负 329
11.3.4 使用特殊成员变量 330
11.4 线程应用 331
11.4.1 EventQueue 与线程分配 331

11.4.2 SwingWorker 类的使用 333
11.5 照猫画虎——基本功训练 333
11.5.1 基本功训练 1——查看线程的运行状态 333
11.5.2 基本功训练 2——查看线程的名称和 ID 334
11.5.3 基本功训练 3——查看线程优先级属性 334
11.5.4 基本功训练 4——使用重入锁实现同步 335
11.5.5 基本功训练 5——原子变量与线程同步 336
11.6 情景应用——拓展与实践 337
11.6.1 情景应用 1——水平移动的图标 337
11.6.2 情景应用 2——多彩的霓虹灯 338
11.6.3 情景应用 3——电子广告牌 339
11.6.4 情景应用 4——逐渐上升的气球 340
11.6.5 情景应用 5——动态绘制圆周 341
11.7 自我测试 341
11.8 行动指南 343
11.9 成功可以复制——初中站长的创业故事 344

第 12 堂课 事件处理的应用 345
视频讲解：77 分钟
12.1 什么是 GUI 事件 346
12.1.1 为程序添加交互能力 346
12.1.2 GUI 事件类 347
12.1.3 事件监听器 349
12.1.4 适配器 349
12.2 窗体事件 350
12.3 掌握鼠标操作 351
12.3.1 鼠标事件监听器 351
12.3.2 鼠标事件处理 352
12.4 掌握键盘控制 353
12.4.1 键盘事件监听器 353
12.4.2 键盘事件处理 354
12.5 照猫画虎——基本功训练 356
12.5.1 基本功训练 1——查看窗体的生命周期 356
12.5.2 基本功训练 2——修改窗体中的鼠标样式 358
12.5.3 基本功训练 3——使用鼠标改变窗体的背景色 359
12.5.4 基本功训练 4——显示程序运行时间 359
12.5.5 基本功训练 5——随机更换窗体背景图片 360
12.6 情景应用——拓展与实践 361
12.6.1 情景应用 1——简易配对游戏 361
12.6.2 情景应用 2——模拟相机拍摄 363
12.6.3 情景应用 3——打地鼠游戏 365
12.6.4 情景应用 4——鼠标经过时放大的按钮 367
12.6.5 情景应用 5——迟到的登录按钮 367
12.7 自我测试 369
12.8 行动指南 370
12.9 成功可以复制——缔造华人的硅谷传奇杨致远 371

第 13 堂课 常用工具类 373
视频讲解：93 分钟
13.1 日期时间类 374
13.1.1 创建 Date 类的对象 374
13.1.2 比较 Date 对象 374
13.1.3 更改 Date 对象 376
13.1.4 日期的格式化输出 376
13.2 数学运算 379
13.2.1 Math 类 379
13.2.2 Math 类的数学方法 379
13.3 随机数 383
13.3.1 通过 Math 类生成随机数 383
13.3.2 使用 Random 类生成随机数 385
13.4 数字格式化类 386
13.4.1 DecimalFormat 类 386
13.4.2 数字的格式化输出 387
13.5 照猫画虎——基本功训练 389

13.5.1 基本功训练 1——输出 30° 的三角函数值 389
13.5.2 基本功训练 2——输出 0.5 的反三角函数值 389
13.5.3 基本功训练 3——角度和弧度的转换 390
13.5.4 基本功训练 4——高精度整数运算 390
13.5.5 基本功训练 5——高精度浮点运算 391
13.6 情景应用——拓展与实践 392
13.6.1 情景应用 1——简单的数字时钟 392
13.6.2 情景应用 2——简单的模拟时钟 393
13.6.3 情景应用 3——简单的公历万年历 395
13.6.4 情景应用 4——七星彩号码生成器 396
13.6.5 情景应用 5——大乐透号码生成器 397
13.7 自我测试 398
13.8 行动指南 399
13.9 成功可以复制——征途巨人史玉柱的第一桶金 400

第 14 堂课 常用集合类 403
视频讲解：102 分钟
14.1 集合类概述 404
14.1.1 什么是集合类 404
14.1.2 Java 集合类的继承关系 404
14.2 List 集合 404
14.2.1 List 接口 404
14.2.2 ArrayList 类 407
14.3 Set 集合 408
14.3.1 Set 接口 408
14.3.2 HashSet 类 409
14.4 Map 集合 410
14.4.1 Map 接口 410
14.4.2 HashMap 类 411
14.4.3 TreeMap 类 413
14.5 照猫画虎——基本功训练 415
14.5.1 基本功训练 1——使用普通 for 循环遍历 ArrayList 415
14.5.2 基本功训练 2——使用 Foreach 循环遍历 ArrayList 416
14.5.3 基本功训练 3——使用 Iterator 遍历 ArrayList 416
14.5.4 基本功训练 4——使用 ListIterator 逆序遍历 ArrayList 417
14.5.5 基本功训练 5——使用 TreeSet 进行排序 418
14.6 情景应用——拓展与实践 419
14.6.1 情景应用 1——用动态数组保存学生姓名 419
14.6.2 情景应用 2——用 List 集合传递学生信息 420
14.6.3 情景应用 3——使用栈逆序输出字符串 420
14.6.4 情景应用 4——制作电子词典 421
14.6.5 情景应用 5——制作手机电话簿 422
14.7 自我测试 423
14.8 行动指南 424
14.9 成功可以复制——使计算机成为生活的必需品 425

第 3 部分 高级篇

第 15 堂课 数据库编程应用 429
视频讲解：122 分钟
15.1 JDBC 技术 430
15.1.1 数据库概述 430
15.1.2 JDBC-ODBC 技术介绍 431
15.1.3 JDBC 技术 432
15.2 JDBC 中常用的类和接口 432
15.2.1 DriverManager 类 432
15.2.2 Connection 接口 432
15.2.3 Statement 接口 433
15.2.4 PreparedStatement 接口 434
15.2.5 ResultSet 接口 434

15.3 数据库连接 435
15.3.1 加载数据库驱动 435
15.3.2 创建数据库连接 436
15.3.3 向数据库发送 SQL 语句 438
15.3.4 获取查询结果集 438
15.3.5 关闭连接 439
15.4 照猫画虎——基本功训练 439
15.4.1 基本功训练 1——在数据库中创建临时表格 439
15.4.2 基本功训练 2——在表格中增加数据 441
15.4.3 基本功训练 3——在表格中删除数据 442
15.4.4 基本功训练 4——在表格中修改数据 444
15.4.5 基本功训练 5——查询表格中的数据 445
15.5 情景应用——拓展与实践 447
15.5.1 情景应用 1——查询商品销售的前 3 名 447
15.5.2 情景应用 2——查询所有王姓同学信息 450
15.5.3 情景应用 3——查询男生和女生的人数 450
15.5.4 情景应用 4——用户重名校验 451
15.5.5 情景应用 5——用户登录系统 452
15.6 自我测试 453
15.7 行动指南 454
15.8 成功可以复制——图文世界的缔造者 455

第 16 堂课 输入输出流与文件 457
视频讲解：118 分钟
16.1 文件 458
16.1.1 File 类 458
16.1.2 常用方法 458
16.2 字节输入输出流 460
16.2.1 标准输入流 460
16.2.2 标准输出流 462
16.2.3 文件输入流 462
16.2.4 文件输出流 463
16.2.5 缓存输入流 464
16.2.6 缓存输出流 465
16.2.7 数据输入流 466
16.2.8 数据输出流 467
16.3 字符输入输出流 468
16.3.1 Reader 类 468
16.3.2 Writer 类 469
16.3.3 FileReader 类 470
16.3.4 FileWriter 类 471
16.3.5 Scanner 类 472
16.3.6 PrintWriter 类 473
16.4 StringBuilder 类 474
16.4.1 StringBuilder 与 StringBuffer 475
16.4.2 常用方法 476
16.5 照猫画虎——基本功训练 476
16.5.1 基本功训练 1——复制单个文件 476
16.5.2 基本功训练 2——删除指定文件 478
16.5.3 基本功训练 3——复制文件夹 478
16.5.4 基本功训练 4——删除含有文件的文件夹 479
16.5.5 基本功训练 5——压缩指定文件 480
16.6 情景应用——拓展与实践 482
16.6.1 情景应用 1——删除文件夹中所有.tmp 临时文件 482
16.6.2 情景应用 2——合并多个 txt 文件 483
16.6.3 情景应用 3——查看文件夹中所有隐藏文件 484
16.6.4 情景应用 4——简单键盘记录器 485
16.6.5 情景应用 5——记录用户的登录时间 486
16.7 自我测试 487
16.8 行动指南 489
16.9 成功可以复制——世界第一人称射击游戏之父 489

第 17 堂课 网络程序设计 491
视频讲解：108 分钟
17.1 网络基础 492
17.1.1 TCP/IP 协议 492
17.1.2 TCP 与 UDP 协议 492
17.1.3 端口和套接字 493

17.2 InetAddress 地址类 494
17.3 TCP 通信 496
17.3.1 ServerSocket 类 496
17.3.2 Socket 类 497
17.4 UDP 通信 500
17.4.1 DatagramSocket 类 500
17.4.2 DatagramPacket 类 501
17.5 照猫画虎——基本功训练 503
17.5.1 基本功训练 1——获取主机 IP 地址 503
17.5.2 基本功训练 2——设置连接超时 504
17.5.3 基本功训练 3——创建 URL 对象 505
17.5.4 基本功训练 4——提取 URL 协议名称 506
17.5.5 基本功训练 5——获取 URL 的端口 507
17.6 情景应用——拓展与实践 508
17.6.1 情景应用 1——客户端一对一通信 508
17.6.2 情景应用 2——编写简单聊天室程序 510
17.6.3 情景应用 3——使用 Socket 传输图片 512
17.7 自我测试 514
17.8 行动指南 514
17.9 成功可以复制——IT“大王”王志东 515

第 18 堂课 表格控件的应用 517
视频讲解：210 分钟
18.1 创建和定制表格 518
18.1.1 创建表格 518
18.1.2 定制表格 520
18.2 维护表格模型 524
18.2.1 创建表格模型 524
18.2.2 设置表格模型 524
18.2.3 维护模型对象 525
18.3 创建行标题栏 529
18.4 模型的事件监听与处理 533
18.5 照猫画虎——基本功训练 537
18.5.1 基本功训练 1——表头与列的高度设置 537
18.5.2 基本功训练 2——调整表格各列的宽度 540
18.5.3 基本功训练 3——设置表格的选择模式 541
18.5.4 基本功训练 4——为表头增添提示信息 542
18.5.5 基本功训练 5——单元格的粗粒度排序 544
18.6 情景应用——拓展与实践 545
18.6.1 情景应用 1——删除指定的图书信息 545
18.6.2 情景应用 2——分页显示 100 内的平方数 546
18.6.3 情景应用 3——五光十色的单元格 548
18.6.4 情景应用 4——实现表格的栅栏效果 550
18.6.5 情景应用 5——根据颜色的分量进行排序 551
18.7 自我测试 552
18.8 行动指南 553
18.9 成功可以复制——微型博客 Twitter 创始人 554

第 19 堂课 树控件的应用 555
视频讲解：141 分钟
19.1 创建树控件 556
19.2 维护树模型 558
19.2.1 创建模型对象 558
19.2.2 设置树控件的模型 560
19.2.3 维护树的模型 562
19.3 照猫画虎——基本功训练 566
19.3.1 基本功训练 1——树控件常用遍历方式 566
19.3.2 基本功训练 2——监听节点的选择事件 567
19.3.3 基本功训练 3——监听节点的展开事件 568
19.3.4 基本功训练 4——设置树控件选择模式 570
19.3.5 基本功训练 5——查看节点的各种状态 572
19.4 情景应用——拓展与实践 573

19.4.1 情景应用 1——以树结构显示文件夹下的文件 573
19.4.2 情景应用 2——增加图书节点 574
19.4.3 情景应用 3——删除图书节点 575
19.4.4 情景应用 4——查找图书节点 576
19.4.5 情景应用 5——为树节点增加提示信息 577
19.5 自我测试 578
19.6 行动指南 579
19.7 成功可以复制——因特网的点火人 580
第 20 堂课 其他高级控件 581
视频讲解：326 分钟
20.1 文件选择对话框 582
20.1.1 创建文件选择对话框 582
20.1.2 添加文件过滤器 583
20.2 工具栏 584
20.2.1 创建工具栏 584
20.2.2 添加工具按钮 585
20.3 菜单栏与弹出菜单 586
20.3.1 创建菜单栏 586
20.3.2 添加菜单和菜单项 587
20.3.3 创建弹出菜单 589
20.4 创建分割面板 591
20.4.1 创建分割面板 591
20.4.2 使用分割面板 592
20.5 创建并使用选项卡面板 594
20.5.1 创建选项卡面板 594
20.5.2 使用选项卡面板 595
20.6 桌面面板和内部窗体 596
20.6.1 创建桌面面板 596
20.6.2 创建内部窗体 598
20.7 使用系统托盘 600
20.7.1 获取系统托盘 600
20.7.2 为系统托盘添加图标 601
20.7.3 添加弹出菜单 601
20.8 照猫画虎——基本功训练 603
20.8.1 基本功训练 1——使子窗体最大化显示 603
20.8.2 基本功训练 2——模仿记事本的菜单栏 604
20.8.3 基本功训练 3——在工具栏中增加图标 605
20.8.4 基本功训练 4——为选项卡增加快捷键 606
20.8.5 基本功训练 5——为选项卡增加图标 607
20.9 情景应用——拓展与实践 608
20.9.1 情景应用 1——支持图片预览的文件选择对话框 608
20.9.2 情景应用 2——记录选项卡的访问状态 610
20.9.3 情景应用 3——简单的图片浏览器 611
20.9.4 情景应用 4——浏览文本文件内容 613
20.9.5 情景应用 5——在弹出菜单中显示图片 616
20.10 自我测试 617
20.11 行动指南 618
20.12 成功可以复制——暴雪公司的领航者 618
第 21 堂课 图形绘制技术 621
视频讲解：74 分钟
21.1 Graphics 简介 622
21.1.1 Graphics 622
21.1.2 Graphics2D 622
21.1.3 获得控件的 Graphics2D 对象 622
21.2 设置颜色与笔画 623
21.2.1 设置颜色 623
21.2.2 笔画属性 624
21.3 如何绘制文本 625
21.3.1 设置字体 625
21.3.2 绘制文本 626
21.4 在控件上绘图 628
21.5 绘制图形 628
21.6 照猫画虎——基本功训练 630
21.6.1 基本功训练 1——绘制图像 630

21.6.2 基本功训练 2——缩放图像 631
21.6.3 基本功训练 3——旋转图像 634
21.6.4 基本功训练 4——倾斜图像 635
21.6.5 基本功训练 5——翻转图像 636
21.7 情景应用——拓展与实践 637
21.7.1 情景应用 1——绘制直方图 637
21.7.2 情景应用 2——倾斜效果的文字 638
21.7.3 情景应用 3——纹理填充特效 639
21.7.4 情景应用 4——简单的校验图片 640
21.7.5 情景应用 5——显示彩色字符串 641
21.8 自我测试 642
21.9 行动指南 643
21.10 成功可以复制——3D 王国的开创者 643

第 4 部分 实战篇

第 22 堂课 进销存管理系统 647
视频讲解：162 分钟
22.1 项目描述 648
22.2 系统分析 648
22.2.1 需求分析 648
22.2.2 可行性分析 649
22.3 系统设计 650
22.3.1 系统目标 650
22.3.2 系统功能结构 650
22.3.3 系统业务流程图 651
22.3.4 系统编码规范 651
22.4 系统运行环境 653
22.5 数据库与数据表设计 654
22.5.1 数据库分析 654
22.5.2 创建数据库 654
22.5.3 创建数据表 655
22.6 创建项目 659
22.7 系统文件夹组织结构 659
22.8 公共类设计 660
22.8.1 SaveUserInfo 公共类 660
22.8.2 OnlyNumber 公共类 661
22.8.3 DaoConn 公共类 661
22.9 系统登录模块设计 662
22.9.1 设计系统登录窗体 662
22.9.2 “登录”按钮的事件处理 662
22.9.3 “密码”文本框的回车事件 663
22.10 主窗体模块设计 663
22.10.1 创建系统主窗体 664
22.10.2 设计菜单栏 664
22.10.3 设计工具栏 664
22.10.4 设计导航按钮 665
22.10.5 设计状态栏 665
22.10.6 设计显示背景图片的面板 665
22.11 供应商基本信息管理模块设计 667
22.11.1 设计供应商基本信息管理窗体 668
22.11.2 添加供应商信息 669
22.11.3 修改供应商信息 670
22.11.4 删除供应商信息 673
22.12 订购商品信息管理模块设计 675
22.12.1 设计订购商品信息管理窗体 675
22.12.2 订购商品信息 676
22.13 采购入库管理模块设计 678
22.13.1 设计采购入库管理窗体 679
22.13.2 采购入库管理 679
22.14 入库信息查询模块设计 682
22.14.1 设计入库信息查询窗体 683
22.14.2 入库信息查询 683
22.15 入库退货管理模块设计 686
22.15.1 设计入库退货管理窗体 687
22.15.2 设计录入商品入库退货信息窗体 688
22.15.3 设计保存输入信息的类 688
22.16 库存盘点窗体模块设计 692
22.16.1 设计库存盘点窗体 693
22.16.2 库存盘点 693

22.17　库存调拨模块设计 694
22.17.1　设计库存调拨窗体 695
22.17.2　添加商品调拨信息 696
22.17.3　查询商品调拨信息 697
22.18　销售订单管理模块设计 699
22.18.1　设计销售订单管理窗体 699
22.18.2　销售订单管理 700
22.19　销售出库管理模块设计 703
22.19.1　设计销售出库管理窗体 703
22.19.2　销售出库管理 704
22.20　销售退货管理模块设计 709
22.20.1　设计销售退货管理窗体 710
22.20.2　设计录入商品销售退货信息窗体 710
22.20.3　设计保存输入信息的类 711
22.21　添加用户信息模块设计 716
22.21.1　设计添加用户信息窗体 716
22.21.2　添加用户信息 717
22.22　用户设置窗体模块设计 718
22.22.1　设计用户设置窗体 719
22.22.2　修改用户密码 719
22.22.3　设置用户权限 720
22.22.4　删除用户信息 722
22.23　经手人管理模块设计 723
22.23.1　设计经手人管理窗体 723
22.23.2　添加经手人 724
22.23.3　设置经手人 725
22.24　开发常见问题与解决 726
22.24.1　不显示系统登录窗体 726
22.24.2　运行程序时找不到数据库驱动 726
22.24.3　运行程序时用户名或密码不正确 727
22.24.4　运行程序时找不到数据库 727
22.25　总结 728

特别提醒：

亲爱的读者朋友，由于近期纸张价格和印制成本大幅上涨，为不增加读者朋友的负担，又不减少书的内容，本书以下章节内容不得不放在配书光盘中，由此给您带来了不便，在此深表歉意。

第 23 堂课　企业内部通讯管理系统 729
视频讲解：10 分钟
（本章内容在配书光盘中）
23.1　开发背景 730
23.2　系统设计 730
23.2.1　系统目标 730
23.2.2　系统功能结构 730
23.2.3　数据库设计 730
23.2.4　文件夹组织结构 731
23.3　主窗体设计 732
23.4　公共模块设计 732
23.5　系统托盘模块设计 733
23.6　系统操作模块设计 733
23.7　用户管理模块设计 734
23.8　通讯模块设计 735
23.9　总结 736
第 24 堂课　图书馆管理系统 737
视频讲解：19 分钟
（本章内容在配书光盘中）
24.1　系统分析 738
24.2　系统设计 738
24.2.1　系统目标 738
24.2.2　系统功能结构 738
24.2.3　系统业务流程图 739
24.3　系统运行环境 739
24.4　数据库与数据表设计 740
24.4.1　数据库分析 740
24.4.2　数据表设计 740
24.5　系统文件夹组织结构 742
24.6　公共类设计 743
24.7　登录模块设计 743
24.8　系统主窗体设计 743
24.9　新书订购模块设计 744
24.10　图书验收模块设计 745
24.11　图书信息添加模块设计 745
24.12　图书信息修改模块设计 746
24.13　图书借阅模块设计 746

24.14 图书归还模块设计 747
24.15 图书查询模块设计 747
24.16 总结 748

附录 A Java 程序编码规范 749
（本章内容在配书光盘中）
A.1 命名规则 749
A.1.1 数据类型简写规则 749
A.1.2 变量及常量名称命名规则 749
A.1.3 类命名规则 749
A.1.4 成员变量命名规则 750
A.1.5 方法命名规则 750
A.1.6 接口命名规则 750
A.2 代码版式 750
A.2.1 统一代码缩进 750
A.2.2 合理使用空行 751
A.2.3 代码换行 751
A.2.4 空格的使用 751
A.3 代码注释 751
A.3.1 注释的目的 751
A.3.2 代码注释规范 751
A.4 数据库编程命名规范 752
A.4.1 数据库命名规范 752
A.4.2 数据表命名规范 752
A.4.3 字段命名规范 752
A.4.4 视图命名规范 753
A.4.5 存储过程命名规范 753
A.4.6 触发器命名规范 753

基础篇

- 第 1 堂课　Java 概述
- 第 2 堂课　使用 IDE 集成开发工具
- 第 3 堂课　Java 语言基础知识
- 第 4 堂课　流程控制语句
- 第 5 堂课　数组应用
- 第 6 堂课　面向对象编程
- 第 7 堂课　字符串处理

第 1 堂课

Java 概述

（视频讲解：31 分钟）

Java 语言是类型安全的、可以编写跨平台面向对象的程序设计语言。开发人员可以使用 Java 语言开发应用程序、企业网站、手机和其他电子设备的应用程序，其应用范围非常广泛。本堂课将向读者介绍什么是 Java 语言、它有几个不同的版本、如何学好 Java 语言，还介绍如何搭建 Java 环境和 Java 的学习与帮助资源。配合后面章节的技术讲解，使读者细致地学习 Java 的具体内容，最后达到完全掌握 Java 语言的目的。

学习摘要：

- Java 语言的发展历史
- Java 的不同版本
- Java 的应用领域
- 学习 Java 语言的方法
- 搭建 Java 环境
- Java 程序的编写步骤

1.1 Java 简介

Java 是 Sun Microsystems 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。自从面世以来一直以易学易用、功能强大的特点得到广泛的应用。其强大的跨平台特性使 Java 程序可以运行在任何一个系统平台上，甚至是手持电话、商务助理等电子产品，真正做到“一次编写，到处运行”。Java 语言可以编写桌面应用程序、Web 应用程序及分布式系统和嵌入式系统应用程序等，这使得 Java 成为应用范围最广阔的开发语言。本节将介绍 Java 语言的作用以及如何学好 Java 语言。

1.1.1 Java 发展历史

1990 年 12 月，Sun 公司成立了一个名为（Green Team）绿色项目的小组，James Gosing——Java 之父也是这个小组的成员，这个小组的主要目标是开发一种能够在消费性电子产品上运行的分布式的系统架构，以掌握这种应用的趋势。

1992 年 9 月 3 日（Green Team）绿色项目小组展示了 Start7 手持设备，它类似于今天的 PDA，但是功能要比 PDA 还强大，5 寸的彩色 LCD 显示屏、Pcmcia 接口并且支持无限通讯。

这个设备中的程序是使用 OAK 程序设计语言实现的，OAK 名字的由来是因为 James Gosing 的办公室窗外有一棵橡树，是根据这棵橡树而命名的。但是在注册商标时却发现这个名字已经被别的公司占用了，后来更名为 Java。

1993 年全世界第一个 Internet 网页浏览器 Mosaic 诞生了，James Gosing 认为 Internet 与 Java 的特性不谋而合，便使用 Java 以优异的功能在 Internet 平台上编写出高交互性的网页程序，这就是 Java Applet。如图 1.1～图 1.6 所示是一些 JDK 附带的 Applet 的 DEMO 程序，读者可以在 JDK 安装文件夹中找到更多更有趣的 DEMO。在那个年代，还没有其他的程序设计语言能够做到这一点。

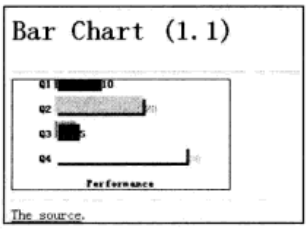


图 1.1 条形报表

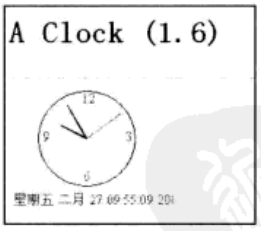


图 1.2 网页时钟程序

图 1.3 网页上的画图工具

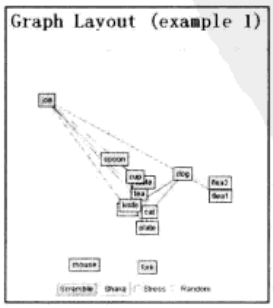


图 1.4 曲线布局

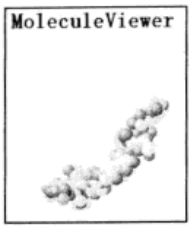


图 1.5　3D 的分子演示（可旋转）

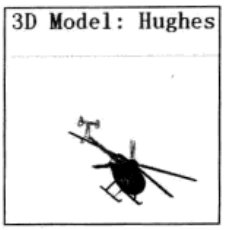


图 1.6　3D 的直升机模型（可旋转）

1995 年 OAK 被命名为 Java 编程语言，并于当年 5 月 23 日正式对外发布 JDK（Java SE Development Kits）1.0 版本，Java 语言在这一时刻正式成立了。它是一种极富创造力的面向对象的程序设计语言，同时也是一种跨平台的程序设计语言。用 Java 语言编写的程序，可以运行在任何平台和设备上，如跨越 IBM 个人电脑、Windows 视窗系统、MAC 苹果系统、手机应用程序、各种电子设备等系统和硬件平台。真正实现“一次编写，到处运行”，它非常适合于企业网络和 Internet 环境，并且已成为 Internet 中最有影响力、最受欢迎的编程语言之一。

在 2009 年 Oracle 公司正式收购了 Sun 成为 Java 的新家园，Oracle 致力于完善 Java 的功能，使它更加强大。

1.1.2　Java 的几个版本

自从 Sun 推出 Java 以来，就力图使之无所不能，所以 Java 发展到现在，按应用范围分为 3 个版本。这 3 个版本分别是 JavaSE、JavaEE 和 JavaME，这也就是 Sun ONE（Open Net Environment）体系，下面分别介绍这 3 个 Java 版本。

1．JavaSE

JavaSE 就是 Java 的标准版，主要用于桌面应用程序的开发，同时也是 Java 的基础，它包含 Java 语言基础、JDBC 数据库操作、I/O 输入输出、网络通讯、多线程等技术。JavaSE 的结构如图 1.7 所示。

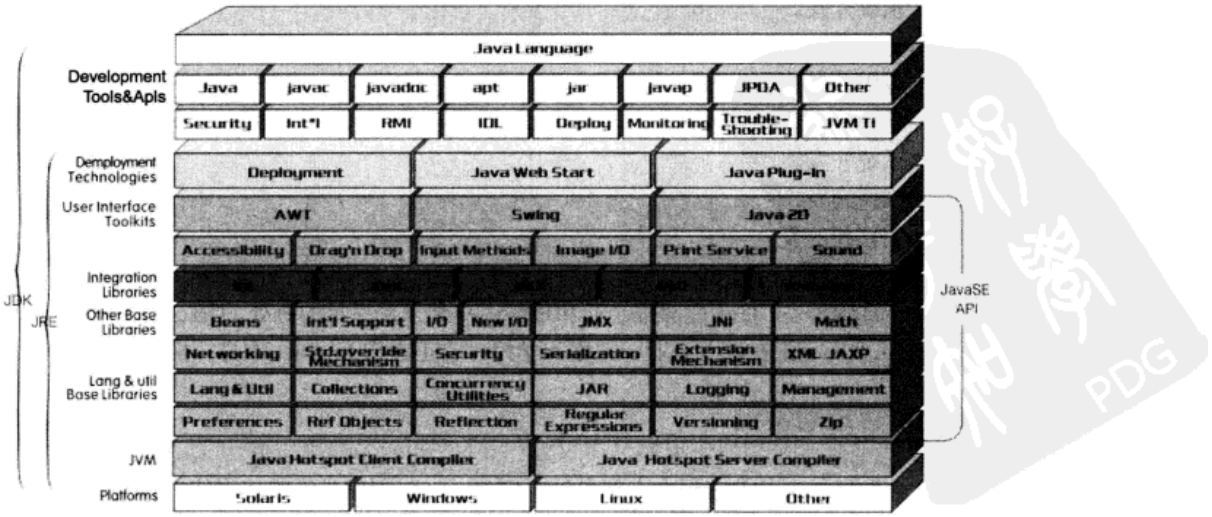


图 1.7　JavaSE 的结构图

2. JavaEE

JavaEE 是 Java 的企业版，主要用于开发企业级分布式的网络程序，如电子商务网站和 ERP 系统，其核心为 EJB。JavaEE 的 API 结构如图 1.8 所示。

3. JavaME

JavaME 主要应用于嵌入式系统开发，如掌上电脑、手机等移动通讯电子设备，现在大部分手机厂商所生产的手机都支持 Java 技术，几乎很少有手机不支持 Java，JavaME 的结构如图 1.9 所示。

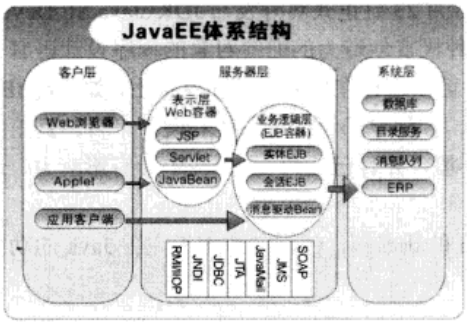


图 1.8　JavaEE 的结构图

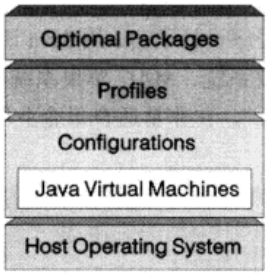


图 1.9　JavaME 的结构图

1.1.3　Java 应用领域

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。这是因为 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、手机和其他大量设备上运行。全世界的公司都已经发现了 Java 数不胜数的用途，它拥有无可比拟的能力，而且节省的时间和费用十分可观。Java 应用领域主要包括：

- ☑ 桌面应用系统开发。
- ☑ 嵌入式系统开发。
- ☑ 电子商务应用。
- ☑ 企业级应用开发。
- ☑ 交互式系统开发。
- ☑ 多媒体系统开发。
- ☑ 分布式系统开发。
- ☑ Web 应用系统开发。

Java 无处不在，它可应用于任何地方、任何领域，并且已拥有几百万个用户，其发展速度要快于在它之前的任何一种计算机语言。Java 能够给企业和最终用户带来数不尽的好处。Oracle 公司董事长兼首席执行官 Larru Ellison 说：“Java 正在进入企业、家庭和学校。它正在像 Internet 本身一样成为普遍存在的技术”。

1.1.4　Java 项目成功案例

仔细观察就会发现，Java 就在每个人的身边。例如，Java 程序员经常使用的 Java 开发工具有 Eclipse、NetBeans、JBuilder 等，RSSOwl、Limewire、Azureus、CyberDuck、OpenOffice 等优秀软件也都是使用 Java 编写的。各个手机厂商也都为自己的产品提供了 Java 技术的支持，各种手机上的 Java 程序和 Java 游戏已经

数不胜数。

Java 语言的项目有很多，它们被应用于实际生活中的各行各业，如清华大学的本科招生网、金网在线、中国债券信息网，其中涉及安全类和银行类的网站较多，包括中国邮政储蓄银行、中国债券信息网、中国农业银行网站、中国建设银行网站、中国工商银行网站、中国光大银行网站、交通银行网站等，它们的网站都使用了 Java 的网站开发技术，部分首页页面效果分别如图 1.10～图 1.15 所示。

图 1.10　清华大学的本科招生网

图 1.11　金网在线网站

图 1.12　中国邮政储蓄银行网站

图 1.13　中国债券信息网

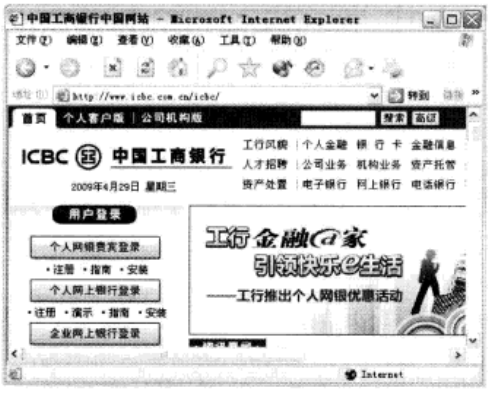


图 1.14　中国工商银行网站

图 1.15　中国光大银行网站

1.1.5 怎样学好 Java

如何学好 Java 语言，这是所有初学者共同面对的问题，其实，每种语言的学习方法都大同小异，需要注意的主要有以下几点：

- ☑ 明确自己的学习目标和大的方向，选择并锁定一门语言，按照自己的学习方向努力学习和认真研究。
- ☑ 初学者不要看太多的书，先找本基础书系统地学习，很多程序开发人员工作了很久也只熟悉部分基础而已，没有系统地学习 Java 语言。
- ☑ 了解设计模式，开发程序必须编写程序代码，这些代码必须具有高度的可读性，这样编写的程序才有调试、维护和升级的价值，学习一些设计模式，就能够更好地把握项目的整体结构。
- ☑ 不要死记语法，在刚接触一门语言，特别是 Java 语言时，掌握好基本语法，并大概了解一些功能就可以了。借助开发工具（如 Eclipse 或 NetBeans）的代码辅助功能，完成代码的录入，这样可以快速地进入学习状态。
- ☑ 多实践，多思考，多请教。只是读懂书本中的内容和技术是不行的，必须动手编写程序代码，并运行程序、分析运行结构，让大脑对学习内容有个整体的认识和肯定，用自己的方式去思考问题，编写代码来提高编程思想，平时多请教老师、经理、同事，和其他人多沟通技术问题，提高自己的技术和见识。
- ☑ 不要急躁，遇到技术问题，必须冷静对待，不要让自己的大脑思绪混乱，保持清醒的头脑才能分析和解决各种问题。可以尝试听歌、散步等活动放松自己。
- ☑ 遇到问题，首先尝试自己解决，这样可以提高自己的程序调试能力，并对常见问题有一定的了解，明白出错的原因，甚至举一反三，解决其他相关联的错误问题。
- ☑ 多查阅资料，可以经常到 Internet 上搜索相关资料或者解决问题的办法，网络上已经摘录了很多人遇到的问题和不同的解决办法，分析这些解决问题的方法，找出最好、最适合自己的方法。
- ☑ 多阅读别人的源代码，不但要看懂别人的程序代码，还要分析编程者的编程思想和设计模式，并融为己用。

1.2 环 境 搭 建

在学习 Java 语言之前首先要了解并搭建好它所需要的开发环境 JDK（Java SE Development Kits），它是 Java 标准版本 JavaSE 的开发工具包。本节以 Windows 平台为例，介绍它的下载、安装与环境搭建。

1.2.1 JDK 下载

JDK 可以在 Sun 公司的官方网站（http://java.sun.com）中下载。下面以目前最新版本的 JDK 6 为例介绍下载 JDK 的方法，具体下载步骤如下：

（1）打开浏览器。

（2）在浏览器的地址栏中输入网址“http://java.oracle.com”，打开 Java 的相关网页，如图 1.16 所示，右侧是下载排行，单击页面中的 Java SE 超链接可以进入下载页面。

（3）在 JavaSE 的下载页面包含很多下载链接，其中包括 JDK 与 JRE 的下载超链接，如图 1.17 所示。现在要准备进行 Java 程序开发，这就需要使用 JDK 包含的源码、编译、运行、调试等工具，所以要单击

Download JDK 按钮下载 JDK 安装文件。

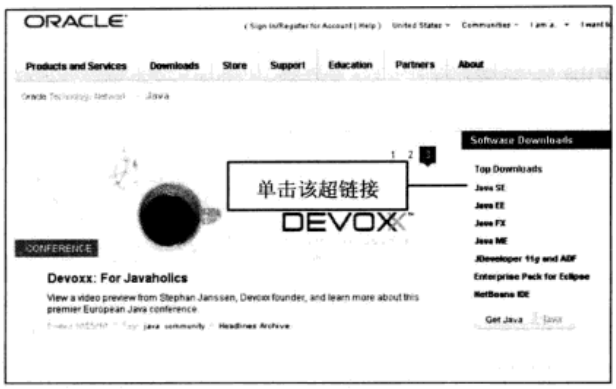


图 1.16　JavaSE 资源页面

图 1.17　JavaSE 下载页面

（4）在 JDK 的下载页面需要选择平台的类型为 Windows，如图 1.18 所示。网站会要求输入登录该网站的用户名与密码，如果没有在该网站注册的用户名与密码，可以单击其下方的 Continue 按钮跳过这个步骤。

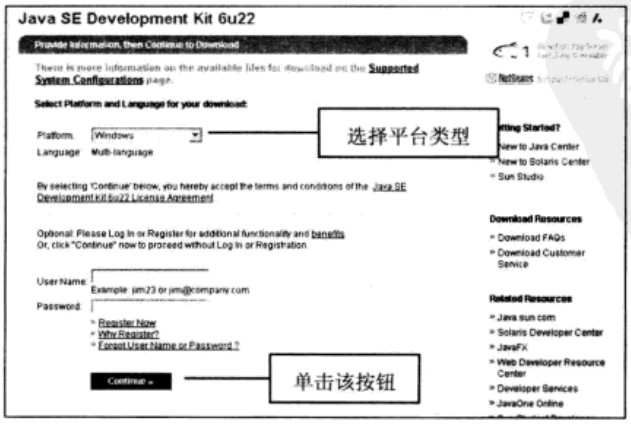


图 1.18　JDK 下载页面

（5）在选择指定的平台之后，会进入下载页面，页面中会显示针对该平台的 JDK 下载文件的超链接。

如图 1.19 所示，单击文件的超链接可以下载该文件。

图 1.19　登录页面

1.2.2　JDK 安装

下载 Windows 平台的 JDK 安装文件为 jdk-6u22-windows-i586.exe，其安装步骤如下：

（1）运行安装文件会启动安装向导的“设置”界面，如图 1.20 所示，单击“下一步”按钮。

（2）在自定义安装对话框中可以选择安装的功能组件，通过单击“更改”按钮可以更改 JDK 的安装路径。这里采用默认值，如图 1.21 所示，单击“下一步”按钮。

图 1.20　“设置”界面

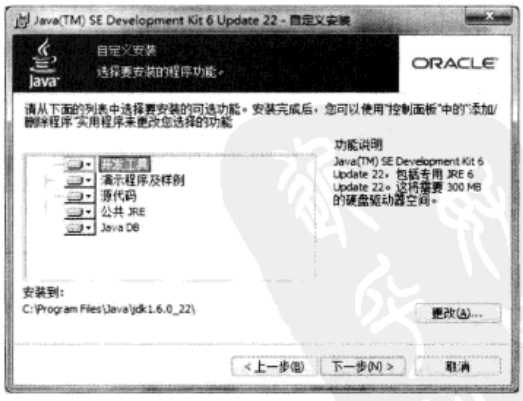


图 1.21　JDK 自定义安装对话框

（3）JRE 是 Java 运行环境的单词缩写，它体积小，只用于 Java 程序的运行命令，不包含开发相关命令。在安装过程中，如果计算机中没有安装 JRE 环境，安装向导会弹出 JRE 的安装对话框。在该对话框中单击“更改”按钮，可以更改 JRE 的安装路径。这里采用默认设置，如图 1.22 所示，单击“下一步”按钮，安装向导会继续完成安装过程。

图 1.22　选择目标文件夹

说明：JRE 全称 Java SE Runtime Environment，它是 Java 的运行环境，主要负责 Java 程序的运行，而 JDK 包含了 Java 程序开发所需要的编译、调试等工具，另外还包含了 JDK 的源代码。对于 Java 客户端，只需要安装 JRE 即可。

（4）安装结束后，弹出“完成”对话框，如图 1.23 所示，单击“完成”按钮，会弹出产品注册的网页，如果用户不想注册，可以关闭浏览器，JDK 与 JRE 的安装到此结束。

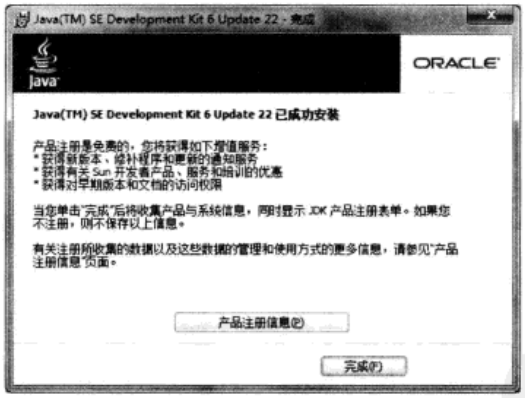


图 1.23　安装完成

1.2.3　配置环境变量

Java 开发环境安装完毕之后，需要设置系统的环境变量，但这不是必需的步骤。

（1）配置环境变量的原因有以下两点：

① 在不使用 IDE 集成开发工具的情况下，如果设置系统环境变量，会打通命令通道，在任何位置输入 Java 的编译指令或调试指令都可以执行，否则必须到 JDK 安装位置才能执行相关命令。

② 如果自定义配置某些软件或服务器环境，则需要 Java 的系统环境变量。

（2）不配置环境变量的原因主要有以下两点：

① 如果使用 NetBeans、Eclipse 等 IDE 集成开发工具进行 Java 程序开发，IDE 开发工具会自行检测 JDK 或 JRE 的位置，或者在 IDE 开发工具中指定 JDK 位置，所以不需要环境变量。但开发工具集成的其他软件

除外。

② 如果是给客户安装 JRE 环境，或者是普通用户自己安装 JRE（Java 运行环境），可以只下载 JRE 安装文件，不需要设置环境变量，这个运行环境相对于.NET 框架要小得多。

在 JDK 程序开发环境中学习 Java 语言开发，必须要了解如何配置环境变量。下面详细介绍在 Windows 系统中配置环境变量的具体步骤，并同时针对读者在配置过程中遇到的常见错误进行详细说明。

在 Windows 7 系统中，配置环境变量的步骤如下：

（1）在“计算机”图标上单击鼠标右键，在弹出的快捷菜单中选择“属性”命令，在打开的对话框中选择“高级系统设置”选项，然后单击“环境变量”按钮，将弹出“环境变量”对话框。“环境变量”对话框中分“用户变量”和“系统变量”两部分，如图 1.24 所示。计算机基础好的读者可能会比较了解，“用户变量”的设置是针对当前操作用户的，而“系统变量”是针对当前系统设置的，也就是所有用户共享系统环境变量。

（2）单击“系统变量”栏中的“新建”按钮，创建新的系统变量。要注意是系统变量，这样权限会大一些，避免更换用户再重新设置环境变量。在图 1.25 所示的“新建系统变量”对话框中，在“变量名”文本框中输入变量名“JAVA_HOME”，在“变量值”文本框中输入 JDK 在本机上的安装路径，然后单击“确定”按钮完成新建。

图 1.24　环境变量对话框

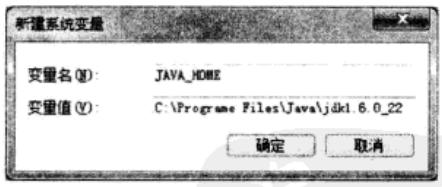


图 1.25　“新建系统变量”对话框

常见错误

① 在输入变量名“JAVA_HOME”时，注意中间是下划线符号“_”，而不是减号“--”，在按下减号键之前，要确认已经按下了 Shift 键，另外要切换到英文输入法进行输入。

② 在输入变量值时，必须输入 JDK 在读者本机上的安装路径，不要直接把图中的路径输入到自己的“变量值”文本框中，如果下载的 JDK 版本与书中的不同，那么安装文件夹的名称会有所变化，这个值与实际的安装路径不符，自然会出错。建议读者使用复制粘贴的方法进行输入，具体步骤如下：

在输入变量值之前，先打开我的电脑，然后依次打开 JDK 的安装文件夹，在地址栏中选择并复制 JDK 文件夹路径，如图 1.26 所示。关闭 JDK 文件夹，回到“新建系统变量”对话框，在“变量值”文本框中单击鼠标右键，在弹出的快捷菜单中选择“粘贴”命令，如图 1.27 所示。

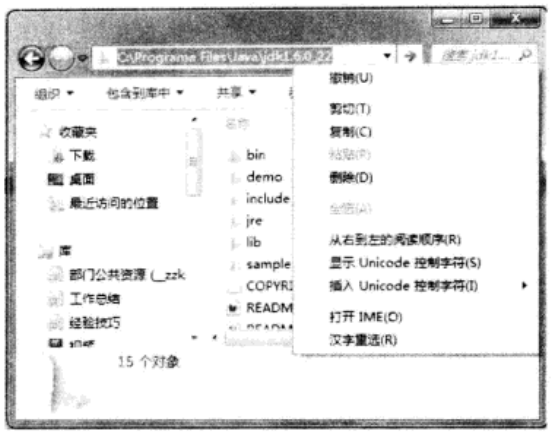


图 1.26　JDK 安装文件夹

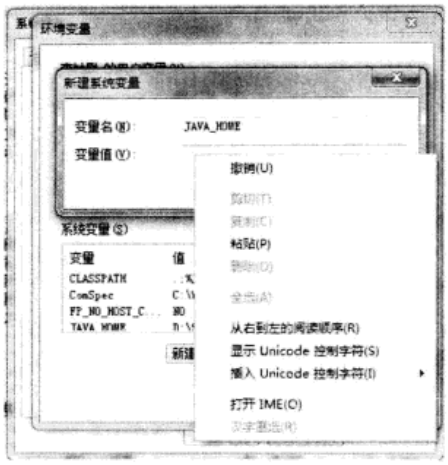


图 1.27　粘贴 JDK 位置

（3）在“环境变量”对话框的“系统变量”栏中拖动滚动条，找到 Path 变量，单击“编辑”按钮或者双击该变量对其进行修改，这时会弹出“编辑系统变量”对话框。对话框中的“变量名”和“变量值”文本框已经被系统预设，不需要修改这些内容。将光标定位到“变量值”文本框中，按下键盘上的 Home 键，在原有变量值最前面添加“.;%JAVA_HOME%\bin;”字符串，如图 1.28 所示。注意最后的分号“;”不能丢掉，而且分号是英文输入法的，它用于分割不同的变量值。单击“确定”按钮完成 Path 环境变量的设置。

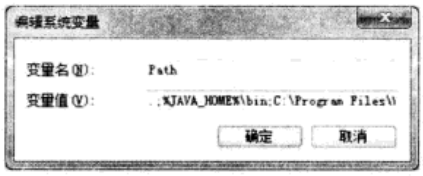


图 1.28　编辑 Path 系统环境变量

常见错误

① 在“变量值”文本框中输入字符串时，注意要在英文状态下进行输入，系统不能识别中文的分号“；”和百分号“%”。

② 输入的变量名称 JAVA_HOME 必须和之前设置的 JAVA_HOME 变量统一。

③ 尤其注意不能将 Path 变量的原有内容全部删除，然后重新输入 Java 的内容。这样 Java 环境变量是正常了，但会使系统中其他软件丧失 Path 功能。

说明： Path 是系统变量，它的作用类似于管道，可以连接变量中多个命令所在的文件夹路径，这样所有路径中的内容在任何位置就都可以使用了。如 system32 文件夹是系统预设在 Path 变量中的，这个文件夹中的所有命令都可以在任意位置执行，如果在 Path 变量中删除 system32 文件夹的路径，那么 ipconfig 等系统命令就不能在任意位置运行。

（4）JDK 安装成功之后必须确认环境配置是否正确。在 Windows 系统中测试 JDK 环境需要选择“开始”/“运行”命令，然后在“运行”对话框中输入“cmd”并单击“确定”按钮启动控制台。在控制台中输入“Javac”命令，按 Enter 键，将输出 JDK 编译命令的帮助信息，如图 1.29 所示，这说明 JDK 环境搭建成功。

图 1.29　在 Windows 下测试 JDK 后的结果

1.3　编写第一个 Java 程序

编写 Java 应用程序，可以使用任何一个文本编辑器来编写程序的源代码，然后使用 JDK 搭配的工具进行编译和运行。当然，现在流行的开发工具可以自动完成 Java 程序的编译和运行，甚至还带有代码辅助功能，可以提供完整的语法代码。但是大型的 IDE 开发工具需要的系统资源较大，在开发一个简单的程序时，还是原始的文本编辑器速度较快。另外，还有一批偏爱编写代码的程序人员，一直使用文本编辑器进行程序开发，这样的程序编写方法造就出的程序开发人员，往往是基础较强、技术比较扎实的程序编写人员。本节将介绍如何使用文本编辑器开发一个简单的 Java 程序，它在屏幕（或者说控制台）上输出 HelloJava,I like you.信息。

1.3.1　在记事本中输入代码

在文本编辑器中编写代码。使用文本编辑器编写 Java 程序代码的过程和平时编写文本文件是一样的，只要注意 Java 语法格式和编码规则即可。选择“开始”/“所有程序”/“附件”/“记事本”命令，在记事本中输入代码，如图 1.30 所示。

常见错误如下：

（1）在输入代码时，使用了中文的符号。例如，中括号“{}”和分号“;”，Java 编译器只识别英文符号，如果读者在输入程序代码时使用中文符号，会导致编译错误。

例如，将 System.out.println()方法所在代码行的最后一个分号替换为中文的“；”，在执行 javac HelloJava.java 命令编译源代码文件时，会出现如图 1.31 所示的错误。

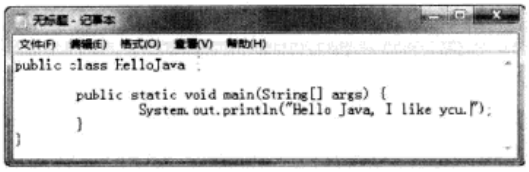


图 1.30　在文本编辑器中编写代码

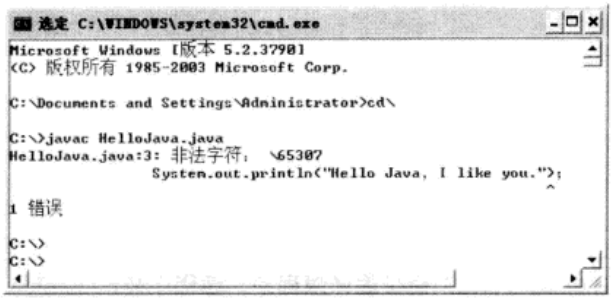


图 1.31　使用中文分号导致的错误

（2）程序代码字母的大小写。Java 对大小写字母是敏感的，也就是 class 不等于 Class，它们是两个词。如果把代码中的关键字或者类、对象变量等字母大小写混淆，将出现多种错误。

1.3.2　保存代码为源代码文件

在编写 HelloJava 程序代码的记事本窗口中选择“文件”/“保存”命令，在“另存为”对话框中首先选择文件的存储位置，这里选择“本地磁盘（C:）”，然后设置保存文件的类型为“所有文件”，即类型为*.*。这样对话框就不会自动为文件名添加.txt 后缀了，最后在“文件名”文本框中输入文件的名称“HelloJava.java”，如图 1.32 所示。

常见错误如下：

（1）在输入文件名时没有添加.java 后缀，或者没有设置“保存类型”为“所有文件”，导致文件最终保存为 HelloJava.java.txt，致使后缀名称错误。

（2）Java 源代码文件的名称必须和定义的类或接口名称一致，然后添加.java 后缀。本实例使用 class 关键字定义了名称为 HelloJava 的类，所以保存该文件时名称必须为 HelloJava.java，而且要区分大小写。如果把文件名称保存为 hellojava.java，即全部小写（与类名不匹配），然后执行编译该文件的指令会出现如图 1.33 所示的错误提示。

图 1.32　保存记事本编写的 Java 源代码文件

图 1.33　文件名与类名不匹配时的错误

1.3.3 编译并运行 Java 文件

Java 源程序需要编译成.class 字节码才能够运行，这需要使用 JDK 的 javac.exe 命令，输入命令时后缀.exe 可以省略。

选择“开始”/“运行”命令，在“运行”对话框中输入“cmd”，单击“确定”按钮，启动控制台。在控制台中输入“cd\”命令将当前位置切换到 C 盘根目录（HelloJava.java 文件的保存路径），然后输入“javac HelloJava.java”命令编译源程序。源程序被正确编译后，会在相同的位置生成相应的.class 文件，这就是编译后的 Java 字节码文件，它是可以运行的。

注意： 输入“javac HelloJava.java”命令时，要注意 javac 和 HelloJava.java 之间有一个空格字符。如果没有输入这个空格字符，将导致命令出错，无法执行。

在控制台中输入“java HelloJava”命令将执行编译后的 HelloJava.class 字节码文件。编译与运行 Java 程序的步骤及运行结果如图 1.34 所示。

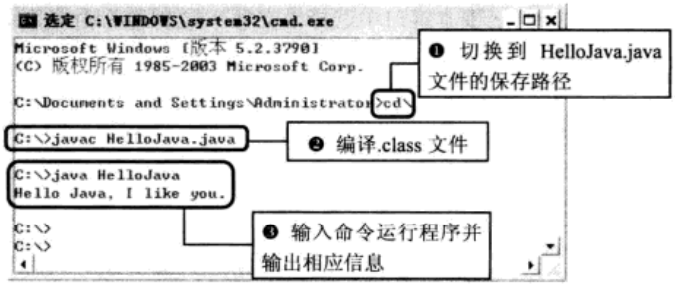


图 1.34 编译与运行 Java 程序的步骤及运行结果

说明： 输入“java HelloJava”命令运行 Java 程序时，运行的是经过编译的 java.class 文件，但是不需要输入.class 后缀名称。

注意： 使用“javac ××.java”命令编译 Java 源程序时，如果没有错误提示信息，说明编译成功，也许会有一些警告信息，但是编译能够通过。如果出现 Exception 类的异常错误信息，那么说明源程序的代码有问题，无法完成编译过程，这时可以根据相应的 Exception 异常判断错误原因和代码位置，来解决代码错误。有关异常类的知识，将在本书的其他章节介绍。

1.4 照猫画虎——基本功训练

1.4.1 基本功训练 1——更换 JAVA_HOME 环境变量

视频讲解：光盘\mr\lx\01\更换 JAVA_HOME 环境变量.exe

在配置环境变量时，介绍过的 JAVA_HOME 环境变量保存 JDK 的安装路径，并且在 Path 环境变量中引用。如果使用另外一个环境变量的名称是否会达到相同的效果，下面来具体测试一下。

实现过程如下：

（1）在“计算机”图标上单击鼠标右键，在弹出的快捷菜单中选择“属性”命令，在弹出的对话框中

选择“高级系统设置”选项，然后单击“环境变量”按钮，将弹出“环境变量”对话框。

（2）在“环境变量”对话框下方找到 JAVA_HOME 变量，双击该变量进行编辑。

（3）修改 JAVA_HOME 变量的名称为 JAVA_DIR，单击“确定”按钮。

（4）修改 Path 环境变量的值，把其中的 JAVA_HOME 替换为 JAVA_DIR，单击“确定”按钮。

（5）使用 javac 命令测试新环境变量，读者会发现该命令依然起作用。

照猫画虎：把 Path 环境变量中原有的 JAVA_HOME 替换为其变量值，也就是 JDK 安装位置的路径，正确替换完以后，Java 环境依然可用。（25 分）

1.4.2　基本功训练 2——提取 Path 变量值创建新环境变量

视频讲解：光盘\mr\lx\01\提取 Path 变量值创建新环境变量.exe

Path 是系统中重要的环境变量，如果它被破坏会导致系统部分命令不可用，或致使某些软件不能运行。如果把该变量值备份到另一个环境变量中，就可以在 Path 环境变量遭到破坏时对其进行恢复。实现过程如下：

（1）在“计算机”图标上单击鼠标右键，在弹出的快捷菜单中选择“属性”命令，在弹出的对话框中选择“高级系统设置”选项，然后单击“环境变量”按钮，将弹出“环境变量”对话框。

（2）在“环境变量”对话框下方找到 Path 变量，双击该变量进行编辑，复制完整的变量值。

（3）新创建一个环境变量 PathBackup，然后将复制的 Path 值粘贴到新的环境变量中，这样就给 Path 环境变量值做了备份。

照猫画虎：编辑 Path 环境变量，在它的变量值中引用备份变量。（25 分）

1.4.3　基本功训练 3——输出字符表情

视频讲解：光盘\mr\lx\01\输出字符表情.exe

实例位置：光盘\mr\ym\1.01\src

很多技术类图书中入门级别的实例或者第一个实例基本都是输出“Hello World”之类的一行文本，其实读者可以把实例扩展，输出自己感兴趣的内容，这样不但增加了学习兴趣，还巩固了入门基础。下面利用输出语句实现一个字符表情的输出，实现过程如下：

（1）打开记事本，在记事本中输入如下程序代码：

```
public class PrintCharFace {
    public static void main(String[] args) {
        // 输出字符组成的小猪表情
        System.out.println("    ┌(^^^^)┐ ");
        System.out.println("   {/ o   o /}");
        System.out.println("    ( (oo) )");
        System.out.println("     ^^ ^^");
    }
}
```

（2）保存源代码文件并执行编译命令。

（3）运行编译后的.class 字节码文件，将在屏幕中输出一个字符组成的表情。

实例运行效果如图 1.35 所示。

图 1.35　实例运行结果

照猫画虎：找到更复杂的字符表情，尝试使用 Java 程序控制输出它们。（25 分）（实例位置：光盘\mr\ym\1.01\zmhh）

1.4.4　基本功训练 4——输出“*”字符组成的三角形

视频讲解：光盘\mr\lx\01\输出“*”字符组成的三角形.exe

实例位置：光盘\mr\ym\1.02\src

在记事本中编写源码，实现输出由字符组成的图形，如三角形、正方形等，多做这类练习可以帮助读者巩固编码、编译与运行的步骤。下面通过实例完成三角形的输出，实现过程如下：

（1）打开记事本，在记事本中输入如下程序代码：

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

（2）保存源代码文件并执行编译命令。

（3）运行编译后的.class 字节码文件，将在屏幕中输出一个字符组成的表情。

实例运行结果如图 1.36 所示。

图 1.36　实例运行结果

照猫画虎：编程输出由“*”符号组成的圆形，圆形要求是空心的。（25 分）（实例位置：光盘\mr\ym\1.02\zmhh）

照猫画虎栏目分数统计：

照猫画虎题目	1	2	3	4	总分数	
分数						

1.5　情景应用——拓展与实践

1.5.1　情景应用 1——计算两个整数的和

视频讲解：光盘\mr\lx\01\计算两个整数的和.exe

实例位置：光盘\mr\ym\1.03\src

输出语句不但可以输出字母和各种符号，还可以输出各种类型的数字以及它们之间的运算结果。在输出语句中可以直接输入运算公式，例如，把 200+432 作为输出语句的内容，将会输出它们的计算结果，实例代码如下：

```
System.out.println(200 + 432);
```

程序运行效果如图 1.37 所示。

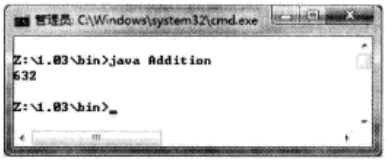


图 1.37　实例输出结果

DIY：尝试输出多个数字组成的加减法组合。（25 分）（实例位置：光盘\mr\ym\1.03\diy）

1.5.2　情景应用 2——输出当前日期

视频讲解：光盘\mr\lx\01\输出当前日期.exe

实例位置：光盘\mr\ym\1.04\src

程序能够输出的信息应该是广泛的，不受约束的。这样才能使程序更加灵活，在字符、数字和各种运算结果之外，用户常用的还有日期信息，也就是年月日，甚至还包括时间。下面将编码实现日期信息的输出，实例代码如下：

```
import java.util.Date;
public class PrintDate {
    public static void main(String[] args) {
        Date date = new Date();                          // 创建日期对象
        String str = date.toLocaleString();              // 获取日期的字符串格式
        System.out.println("当前日期：" + str);           // 输出日期字符串
    }
}
```

实例运行结果如图 1.38 所示。

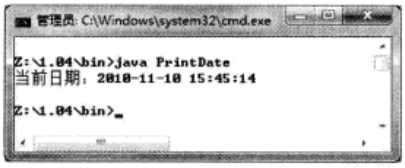


图 1.38　实例输出结果

你问我答：代码中的 Date 是什么意思？

实例运行结果将输出当前日期与时间。实例中使用了日期类型的变量，时间是 Java 中 Date 类的实例对象，知识相对复杂一些，但是这里将给读者介绍如何创建一个时间对象，并获取它的日期字符串。创建日期对象的关键代码如下：

```
Date date = new Date();                          // 创建日期对象
String str = date.toLocaleString();              // 获取日期的字符串格式
```

代码中变量 str 保存的就是有日期对象的 toLocaleString()方法返回的日期字符串。在实现这个步骤之前，必须在源代码文件的顶端执行 import java.util.Date 语句导入这个类。

DIY：尝试输出带有日期时间的文字信息“我在北京时间：XXXX 完成了第一堂课的学习”，XXXX 代表程序运行时的时间。（25 分）（实例位置：光盘\mr\ym\1.04\diy）

1.5.3 情景应用 3——在控制台接收用户输入数字

视频讲解：光盘\mr\lx\01\在控制台接收用户输入数字.exe

实例位置：光盘\mr\ym\1.05\src

System 类除了 out 和 err 两个输出流之外，还有 in 输入流的实例对象作为类成员，它可以接收用户的输入。下面通过这个输入流从控制台接收用户输入的数字与字符串，实例运行结果如图 1.39 所示，实现过程如下。

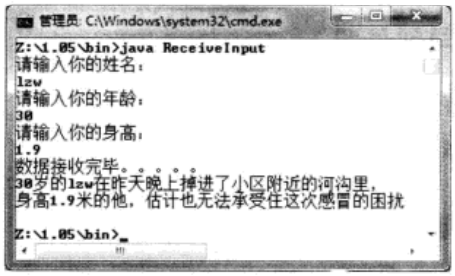


图 1.39 实例运行结果

创建 Receiveinput 类，在该类的主方法中创建 Scanner 扫描器来封装 System 类的 in 输入流，然后提示用户输入姓名、年龄和身高信息，并输出由这些信息组成的说明信息，实例代码如下：

```
import java.util.Scanner;
public class ReceiveInput {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);              // 创建输入扫描器
        System.out.println("请输入你的姓名：");
        String name = scan.nextLine();                      // 获取一行文本
        System.out.println("请输入你的年龄：");
        int age = scan.nextInt();                           // 获取整数输入
        System.out.println("请输入你的身高：");
        double stature = scan.nextDouble();                 // 获取 double 类型输入
        System.out.println("数据接收完毕。。。。。");
        System.out.println(age + "岁的" + name + "在昨天晚上掉进了小区附近的河沟里，\n" + "身高"
```

```
                + stature + "米的他，估计也无法承受住这次感冒的困扰");
        }
    }
```

你问我答：代码中的 Scanner 是什么意思？

程序中用到了 System 类的输入流也即类变量 in，它可以接收用户的输入信息，并且是标准的输入流实例对象。另外 Scanner 类是 Java 的扫描器类，它可以从输入流中读取指定类型的数据或字符串。本实例使用 Scanner 类封装了输入流对象，并使用 nextLine()、nextInt()、nextDouble()方法从输入流中获取用户输入的整行文本字符串、整数数据和 double 类型数据。

说明：在使用 Scanner 类时，必须在源代码顶端导入这个类，导入语句为 import java.util.Scanner;

DIY：把用户输入文本与当前日期连接为字符串输出。（25 分）（实例位置：光盘\mr\ym\1.05\diy）

1.5.4　情景应用 4——计算用户输入数字的乘积

视频讲解：光盘\mr\lx\01\计算用户输入数字的乘积.exe

实例位置：光盘\mr\ym\1.06\src

简单的数字运算只能体现计算机的速度与高效，程序需要与用户形成互动才有它存在的意义。例如，从控制台接收用户输入的数字，然后使用这些数字进行运算，就形成了与用户的互动，而且可以根据用户需求来指定要运算的数字，运行结果如图 1.40 所示，实现过程如下。

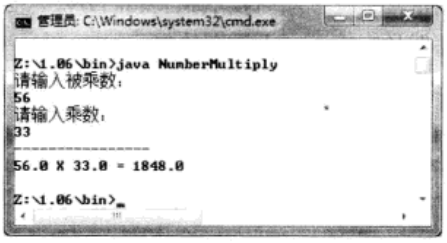


图 1.40　实例运行结果

创建 NumberMultiply 类，在该类的主方法中创建 Scanner 扫描器来封装 System 类的 in 输入流，然后提示用户输入参与运算的被乘数与乘数，并输出两个数字的乘积，实例代码如下：

```
import java.util.Scanner;
public class NumberMultiply {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);              // 创建输入扫描器
        System.out.println("请输入被乘数：");
        double num1 = scan.nextDouble();                    // 接收用户输入数字
        System.out.println("请输入乘数：");
        double num2 = scan.nextDouble();                    // 接收用户输入数字
        System.out.println("----------");
        // 输出计算结果
        System.out.println(num1 + " x " + num2 + " = " + num1 * num2);
    }
}
```

DIY：扩展本实例，使它可以计算 3 个用户输入的数字的乘积。（25 分）（实例位置：光盘\mr\ym\1.06\diy）

情景应用 DIY 栏目分数统计：

DIY 题目	1	2	3	4	总分数	
分数						

1.6 自 我 测 试

一、选择题（每题 10 分，5 道题）

1．下面哪个名称不属于 Java 版本？（　　）

A．JavaSE　　B．JavaEE　　C．JavaME　　D．JavaScript

2．当编译和运行下列代码时会发生什么？

```
public class Test {
        public static void main() {
            System.out.println("Hello World!");
        }
}
```

选择一个正确的答案：（　　）

A．不能正常编译，因为 main()方法定义错误

B．可以正常编译，但不能正常运行，因为找不到 main()方法

C．可以正常编译和运行，但是在控制台上没有输出

D．可以正常编译和运行，在控制台上显示“Hello World!”

3．Java 源代码的扩展名是（　　）。

A．.txt　　B．.exe　　C．.bat　　D．.java

4．Java 的编译命令是（　　）。

A．java　　B．javac　　C．javaw　　D．javaws

5．JDK 是 Java 的（　　）环境。

A．运行环境　　B．开发环境　　C．安装文件　　D．开发工具

二、填空题（侧重实用技能，每题 10 分，5 道题）

1．Java 的运行命令是_____。

2．JRE 是 Java 的_____环境。

3．Java 源文件经过编译后会生成字节码文件，字节码文件的扩展名是_____。

4．Java 开发环境要配置的两个系统环境变量是_____和_____。

5．Windows 系统配置系统环境变量时要使用_____符号分割多个变量值。

测试分数统计：

类别	第 1 题	第 2 题	第 3 题	第 4 题	第 5 题
选择题分数					
填空题分数					
总分数					

1.7　行 动 指 南

开始日期：_____年____月____日　　结束日期：_____年____月____日

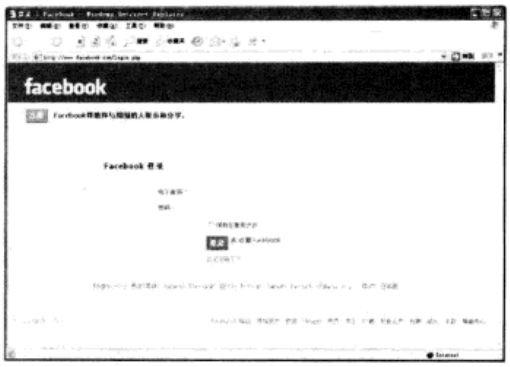
序号	内　容	行 动 指 南	
1	照猫画虎栏目 分数（　　）	分数>75 分	优秀，基本功掌握得不错，加油！
		75 分>分数>50 分	及格，知识掌握得不牢，重新做一遍照猫画虎。
		分数<50 分	请使用光盘中提供的“实战能力测试系统”进行提高训练。
	情景应用栏目 分数（　　）	分数>75 分	优秀，综合应用能力很强。
		75 分>分数>50 分	及格，综合应用能力需提高，再练一遍情景应用。
		分数<50 分	请使用光盘中提供的“实战能力测试系统”进行提高训练。
	自我测试栏目 分数（　　）	分数>75 分	优秀，有成为编程高手的潜质。
		分数<75 分	请使用光盘中提供的“实战能力测试系统”进行提高训练。
	综合评价	反复训练后，以上各项分数都在 75 分以上，方可进入下一堂课学习。	
2	编程能力培养：本栏目提供了一些实践题目，对于培养编程能力很有效，要做好。	（1）做一个最简单的计算器，接收用户输入的两个数字，然后计算加、减、乘、除各种运算的结果。	
		（2）编程输出一个“*”组成的空心正方形。	
		（3）接收用户输入的整数，然后输出该数字的平方。	
		（4）编程输出自己的姓名、性别、年龄、存款。	
3	编程习惯培养：本堂课培养读者在学习开发中记笔记的习惯，把开发中遇到的问题、总结的经验记录下来。		
4	创新能力培养：看看身边有没有可以用编程解决的问题，记录到右边的表格中。根据学习进度，尝试编写解决这些问题的小程序。		

1.8　成功可以复制——“盖茨第二”

1984 年 5 月 14 日，马克·扎克伯格出生在纽约，他很小就展现出了计算机天赋，6 年级就开始编程。对于学习编程的经历，马克·扎克伯格说：“我买了一些书，基本是自己在胡乱摸索，也向很多人请教过。我对电脑硬件不感兴趣，我只是对电脑程序很感兴趣。” 高中时，他与好友达杰罗为 MP3 播放器编写了一个智能插件软件，并放在互联网上供免费下载。很快，包括美国在线（AOL）和微软在内的各大公司纷纷向他抛来了橄榄枝。但是扎克伯格却拒绝了年薪 95 万美元的工作机会，而选择去哈佛大学上学。

身为常春藤名校的学生，马克·扎克伯格曾闹出了一起著名的黑客事件。在新生第一年即将结束的时候，马克发现哈佛大学没有制作学生年鉴的惯例，于是向学校理事会提议制作一个在线版本，但学校总是

用资料难以收集等借口来推托。在正式跨入二年级的前一天，他侵入了哈佛大学的学生信息登记系统，获得了所有在校本科生的资料，并迅速架设了名为 Facemash 的网站。这个网站随机展示两个学生的照片，让来访者判定哪一个更“漂亮”。在短短 4 个小时内，访问量达到 450 人次，22000 张照片被点击。校方发现该网站后当即掐断了马克宿舍的网络连接。在遭受了校方和哈佛校报的轮番训斥后，马克向全校学生进行了公开道歉。但他并不觉得自己做错了什么，“我始终认为，信息就应该得以利用。”



2004 年 2 月 4 日，社交网站 Facebook 在大学二年级学生马克·扎克伯格的手中诞生了。当时它还只是一个仅面向哈佛大学在校生的小型网络，随后在同学达斯汀·莫斯可维茨的协助下，将 Facebook 推广至美国的其他大学。2004 年年底，Facebook 的注册人数已突破一百万，马克·扎克伯格干脆从哈佛大学退学，全职营运网站。

在社交型网络发展的早期，Facebook 依靠数笔投资得以顺利运营。2005 年，雅虎曾正式开出了 10 亿美元的天价并购 Facebook，但被当时年仅 22 岁的马克一口回绝：“我们既不打算卖掉公司，近期内也不打算 IPO（初次公开募股）。”马克始终坚持公司的目标是寻求长期发展，不必分心兼顾别的事情。

2007 年年底，仅仅用了不到 4 年的时间，Facebook 已成为美国最火爆的大学生社交网站，网站全球排名第 8 位。目前注册用户超过了 4 亿，同时在线人数超过了 1 亿。

经典语录

最大的风险就是不去承担任何风险。在这个瞬息万变的世界里，不变即意味着失败。改变对于用户——尤其是网络服务用户而言——总是颠覆性的。

深度评价

机会就是别人不知道他知道了，别人不明白他明白了，别人犹豫或不做而他做了。当别人知道了，明白了，想要做时，他已经成功了！机会就是这样，总是偏爱少数人，因为大部分人都有一种惰性，喜欢跟风，人云亦云。

第 2 堂课

使用 IDE 集成开发工具

（视频讲解：128 分钟）

学好程序的基本控制结构是结构化程序设计的基础，它可以使程序结构清晰、易读性强，并且易于查错和排错。它包括顺序结构、选择结构和循环结构。本堂课详细地介绍了这 3 种控制结构，讲解过程中为了便于读者理解结合了大量的实例。

学习摘要：

- 了解 Eclipse 项目
- 掌握 Eclipse 的下载与安装
- 掌握 Eclipse 的汉化与插件安装
- 熟悉 Eclipse 界面
- 使用 Eclipse 开发 Java 程序
- 使用 Eclipse 开发桌面程序
- 使用 Eclipse 调试程序

2.1 Eclipse 简介

Eclipse 是目前最流行的 Java 集成开发工具之一，它是基于 Java 语言编写，开放源代码的、可扩展的（Integrated Development Environment，IDE）开发工具，由 IBM 公司投资 4000 万美元开发并捐赠给开源社区。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，这也是其区别于其他开发工具的特征之一。学习完本堂课之后，读者将对 Eclipse 有一个初步的了解，为后面的深入学习作个铺垫。

2.2 Eclipse 下载与安装

2.2.1 Eclipse 分类

Eclipse 目前划分为几个不同的安装包版本，其中包括 Java 环境安装包、JEE 环境安装包、C/C++语言安装包、RCP/Plugin 插件安装包、Modeling 建模环境安装包、（JEE BIRT）Reporting 企业级报表安装包和 Classic 标准安装包等，如图 2.1 所示，另外 Eclipse 还支持 PHP、JavaScript 等其他语言的版本。

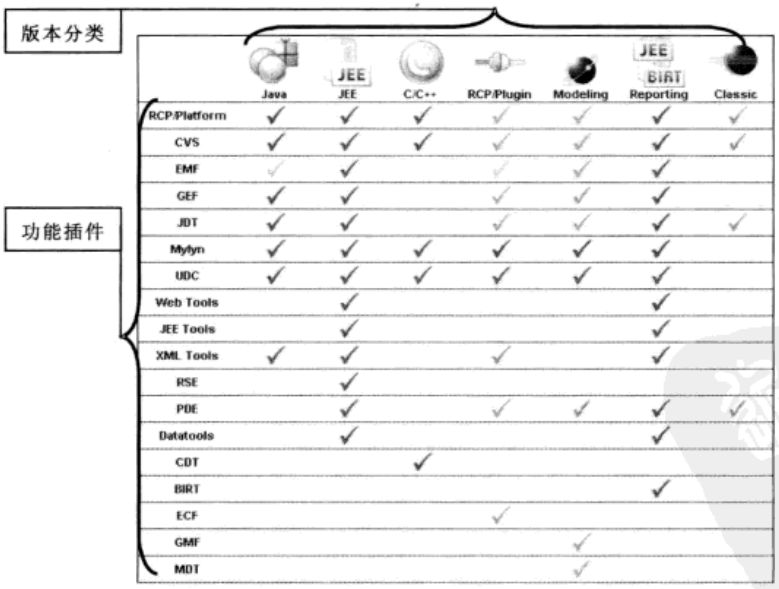


图 2.1 Eclipse 安装包分类

2.2.2 下载 Eclipse

在浏览器地址栏中输入 Eclipse 的下载网址“http://www.eclipse.org/downloads”进入官方网站，可以从网站下载 Eclipse 的最新版本。在撰写本书时，Eclipse 的最新版本是 Eclipse 3.6.1，具体下载步骤如下：

（1）打开 Eclipse 下载网站，单击页面中的 Eclipse IDE for Java EE Developers 超链接，可以进入该版

本 Eclipse 的下载页面，如图 2.2 所示。

图 2.2　Eclipse 主页

（2）在 Eclipse 的下载页面中包含很多不同系统平台的下载超链接，如 Windows、Mac OS X、Linux 等，如图 2.3 所示。单击 Windows 32-bit 超链接进入镜像下载页面，如果读者要下载其他系统平台的 Eclipse，可以单击相应平台的超链接。

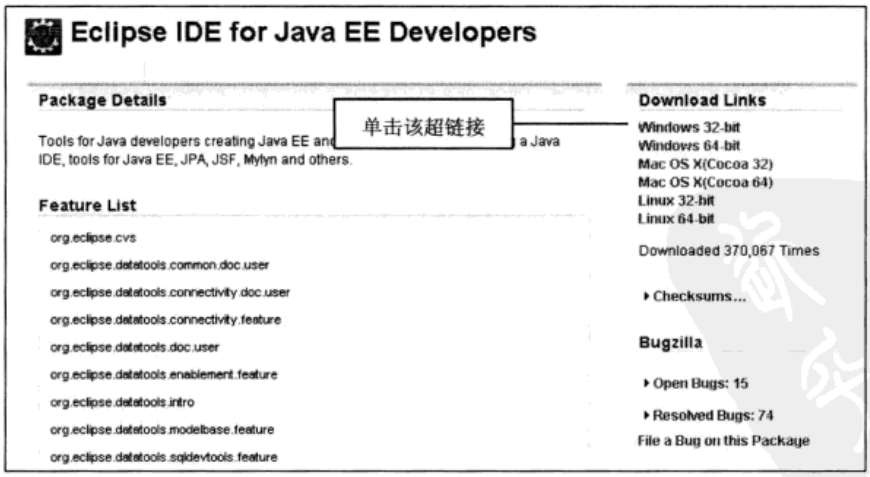


图 2.3　Eclipse 的下载页面

（3）在 Eclipse 镜像下载网页中，服务器已经为用户提供了根据 IP 地址判断的、最佳下载镜像的超链接，同时服务器也自动选择了适合用户操作系统的 Eclipse 版本。这里选择的是中国上海的镜像站点，网页提供的是 Win32 版本的 Eclipse 下载文件，如图 2.4 所示，单击镜像站点的超链接下载 Eclipse。

下载的 Eclipse 安装包是一个 ZIP 压缩文件，把它直接解压缩到 D 盘或其他磁盘中，运行文件夹中的 Eclipse.exe 文件就可以启动 Eclipse 了，但前提是要求计算机中安装了 Java 环境。

郑州大学图书馆

图 2.4　Eclipse 的压缩包对比页面

2.2.3　安装中文语言包

下载的 Eclipse 默认是英文版本的，对于不同国别的用户来说，还是使用自己国家的语言最流畅。为方便读者的学习，本节将向读者介绍如何获取和安装 Eclipse 的中文语言包，同时也可以下载其他国家的语言包。

Eclipse 3.3 之后的版本没有提供相应的多国语言包，但是 Eclipse Babel Project 项目可以实现新版本 Eclipse 的国际化，它可以为 Eclipse 提供多语言包的安装，包括简体中文、繁体中文、日文、韩文等。下面介绍安装中文语言包的步骤。

（1）在浏览器地址栏中输入 Babel 项目的网址“http://babel.eclipse.org/”，按 Enter 键打开项目首页，如图 2.5 所示，单击 Download a language pack in one of many 超链接。

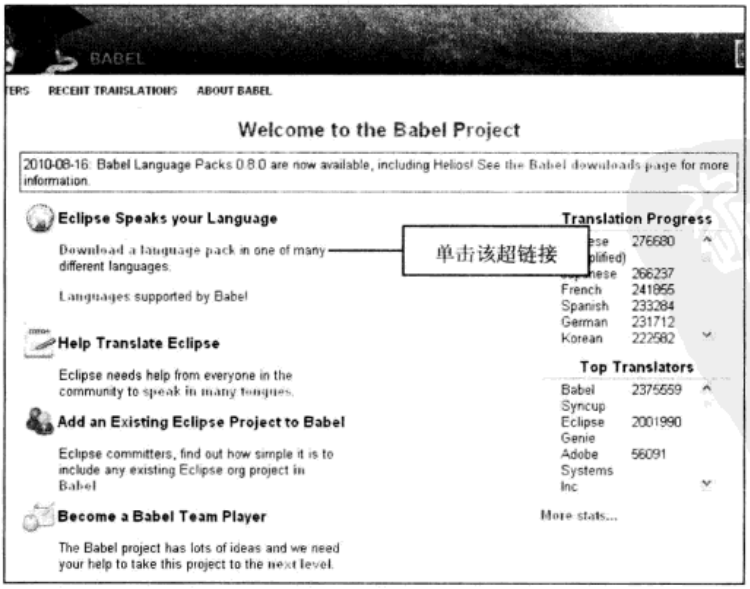


图 2.5　Babel 项目的首页面

（2）在 Babel 项目的下载页面中包含有 Eclipse 的语言包更新站点信息。截止到本书完稿时，该网站提

供的 Eclipse 最高版本的语言包是 Helios，也就是 Eclipse 3.6，如图 2.6 所示。单击 Helios 超链接将进入对应版本的语言包页面。

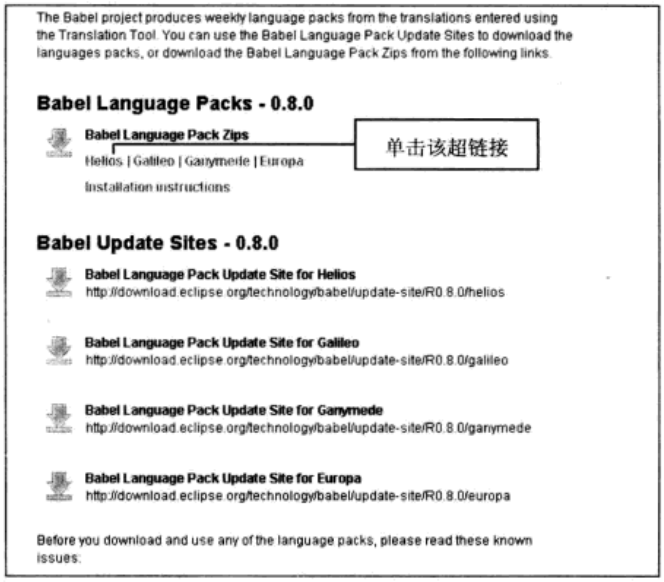


图 2.6　Babel 项目的下载页面

（3）在 Babel Language Packs for Helios 页面的 Go to 标题上，单击 Chinese（Simplified）超链接，如图 2.7 所示。它下面包含了简体中文语言包，这些中文语言包适用于不同的 Eclipse 插件，其中 BabelLanguage Pack-eclipse-zh_3.6.0.v…zip 文件是 Eclipse 的核心语言包。下载后，将语言包解压缩并覆盖 Eclipse 相应的文件夹即可附加中文语言包。

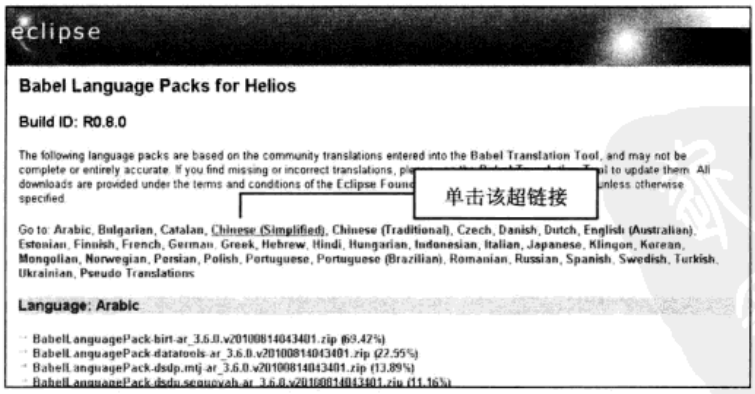


图 2.7　语言包下载页面

2.2.4　启动 Eclipse

现在已经配置好 Eclipse 的多国语言包，可以启动 Eclipse 了。在 Eclipse 的安装文件夹下运行 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。通过单击“浏览”按钮来选择本书光盘中提供的 ym 文

件夹的位置，它是一个 Eclipse 空间，包含了图书所有实例项目和工作空间的配置，如图 2.8 所示，单击“确定”按钮，将启动 Eclipse。

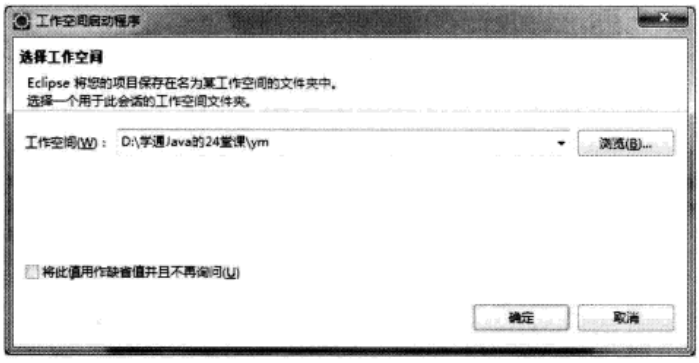


图 2.8　设置工作空间

在以后的操作中，读者可以指定不同的文件夹作为新的工作空间，在那里开发自己的项目。

说明： 光盘是只读的，所以需要把光盘中的 ym 文件夹复制到电脑中。例如，复制到磁盘 D，并去掉只读属性，由于光盘是只读的，读者在使用时需要将代码复制到本地磁盘中才能进行修改。

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”复选框设置默认工作空间，这样 Eclipse 启动时将不会再询问工作空间的设置。

Eclipse 首次启动时，会显示欢迎界面，在欢迎界面中包含了 Eclipse 的概述、新增内容、样本、教程以及工作台的相关按钮，如图 2.9 所示，这个界面可以定制不同的样式布局。

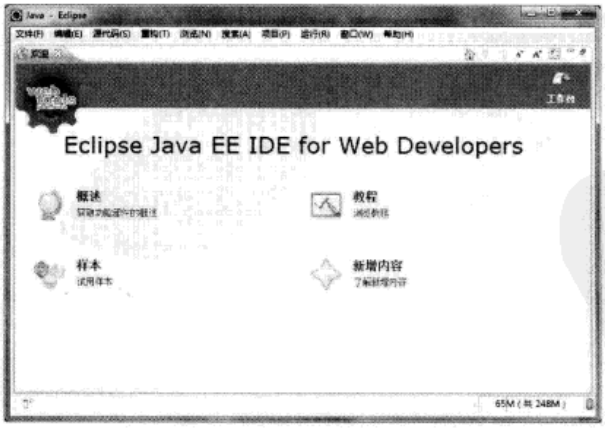


图 2.9　Eclipse 的欢迎界面

2.3　安装界面设计器

Eclipse 没有提供设计应用程序界面的功能，但是目前已经有很多实现界面设计器的 Eclipse 插件。其中 WindowBuilder 的功能强大，使用方便。下面将以 WindowBuilder 为例介绍其安装步骤。

（1）首先启动 Eclipse，选择菜单栏中的“帮助”/Install new software 命令，将弹出 Install 对话框，在该对话框中单击右上角的 Add 按钮，弹出 Add Repository 对话框，如图 2.10 所示。在 Name 文本框中输入新插件站点的名称，也可以自定义，如 babel；在 Location 文本框中输入新插件的网址，本书使用的是 Eclipse 3.6，所以在这里输入该版本对应的插件网址：“http://dl.google.com/eclipse/inst/d2wbpro/latest/3.6”。这个网址可以从 http://code.google.com/intl/zh-CN/javadevtools/download-wbpro.html 页面获取。

（2）在 Install 对话框中将会解析出刚刚添加的插件网址中包含的多个插件，如图 2.11 所示，在界面中选中 Swing Designer 复选框，单击“下一步”按钮。

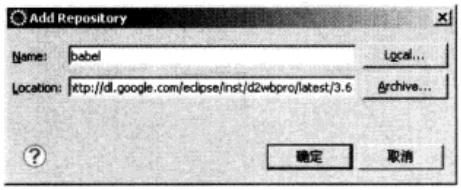


图 2.10　Add Repository 对话框

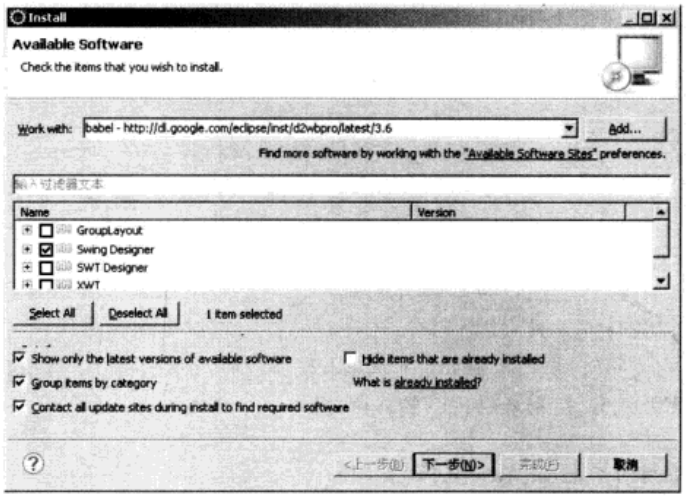


图 2.11　Install 对话框

（3）安装向导会显示要安装插件的名称、版本和 ID 等信息，如图 2.12 所示，单击“下一步”按钮。

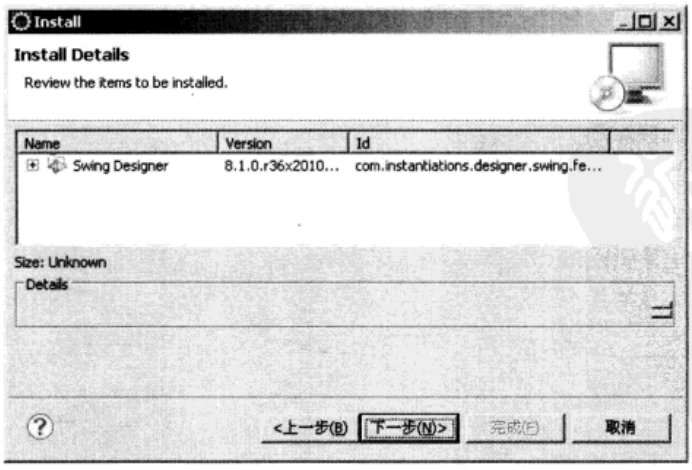


图 2.12　显示插件信息

（4）在正式安装时，会提示用户查看并接受许可协议才能继续安装，如图 2.13 所示，选中 I accept the terms of the license agreement 单选按钮，然后单击“完成”按钮完成安装。

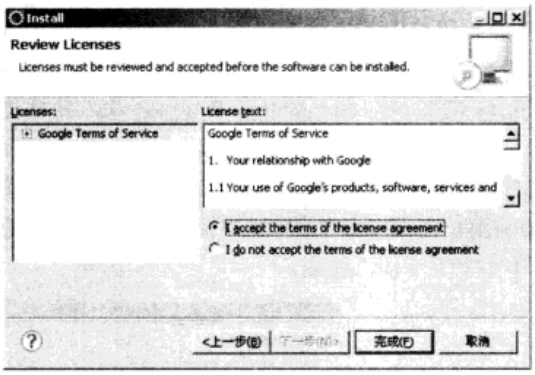


图 2.13　接受许可协议

2.4　熟悉 Eclipse

虽然使用记事本和 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，包括安装、配置与启动以及菜单栏、工具栏和各种视图的作用等。

2.4.1　Eclipse 工作台

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 2.14 所示。在接下来的章节中将介绍 Eclipse 的菜单栏与工具栏，以及透视图、视图，并介绍常用视图。

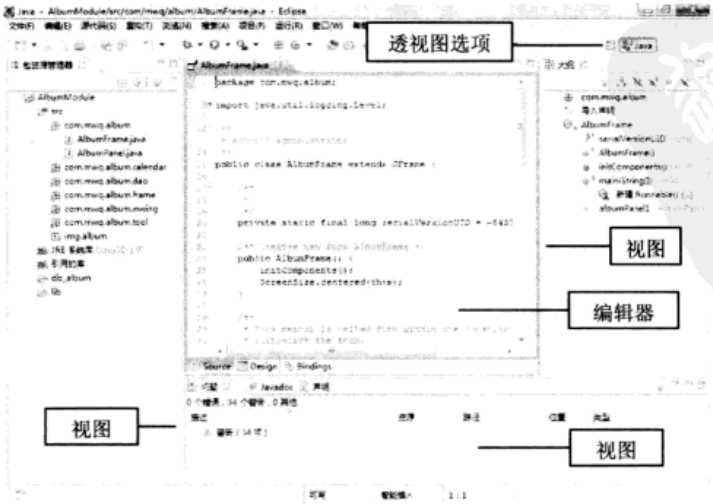


图 2.14　缩放后的 Eclipse 工作台

2.4.2　透视图与视图

透视图和视图是 Eclipse 中的概念名词，本节将分别介绍透视图和视图的概念以及它们在 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字。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

视图多用于提供信息的层次结构、显示活动编辑器的属性等，例如，“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”视图可以提供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栏和工具栏，并且可以通过拖动随意改变布局位置。

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“以折离”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。

2.4.3　包资源管理器视图

该视图用于提供项目结构中的 Java 元素，包括包、类、类库的引用等，但最主要的用途是提供项目中的源代码文件，包资源管理器视图的界面如图 2.15 所示。

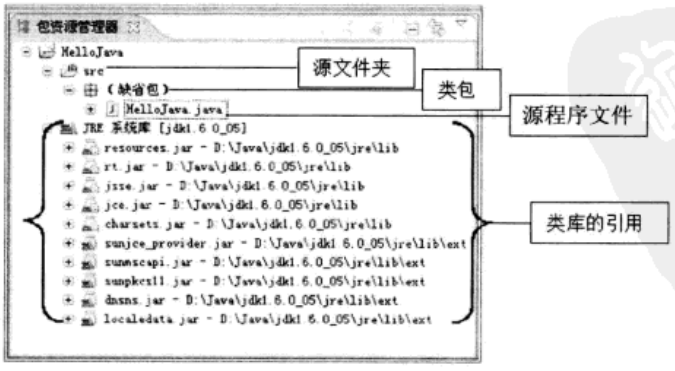


图 2.15　拆分后的包资源管理器视图

2.4.4　编辑器视图

编辑器是工作台上的一个主要的可视实体，在任何一个透视图中，都会包含一个编辑器区域，该区域

可以包含多个编辑器。Java 编辑器打开 HelloJava.java 类文件的界面如图 2.16 所示。

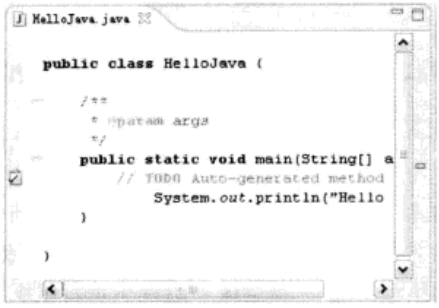


图 2.16　编辑器区域的 Java 编辑器

2.4.5　控制台

控制台视图用于显示程序运行时的输出结果和（Runtime Exception）运行时异常信息，在学习程序设计之前，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 2.17 所示。

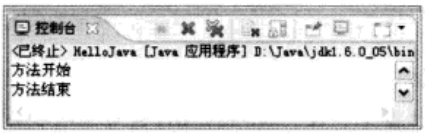


图 2.17　控制台视图

2.4.6　大纲视图

该视图显示当前在编辑器区域中打开的结构化文件的大纲，并列示结构化元素。大纲视图的内容是特定于编辑器的。如图 2.18 所示是针对 Java 源文件的大纲视图，它的结构化元素有类、变量和方法，工具栏的内容也是特定于编辑器的。

图 2.18　大纲视图

2.4.7　问题视图

当在工作台中使用资源时，各种构建器可以将问题、错误或警告自动记录在“问题”视图中。例如，当保存包含语法错误的 Java 源文件时，那些错误将记录在问题视图中。双击错误或警告的图标时，相关资

源的编辑器将自动打开文件并转到相应的代码行，如图 2.19 所示。

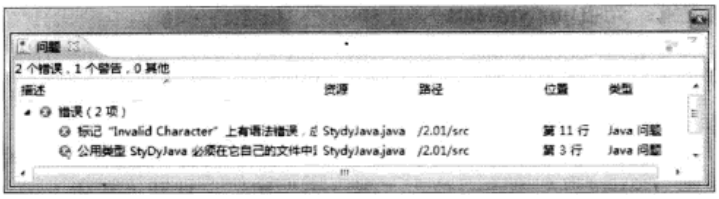


图 2.19　问题视图

2.4.8　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和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 2.20 所示。

图 2.20　透视图工具栏

2.4.9　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（有关视图的概念已在 2.4.2 节讲解过），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，它的工具栏界面如图 2.21 所示。

图 2.21　控制台视图的标题栏和工具栏

2.4.10　快速视图工具栏

快速视图工具栏包含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的形式隐藏，当单击指定的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 2.22 所示。

图 2.22　Eclipse 中的工具栏

2.5　如何开发 Java 程序

现在读者对 Eclipse 工具应该有个大体的认识了，本节将通过实例介绍如何使用 Eclipse 完成 StudyJava 程序的编写和运行。

例 2.01 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 项目用于编写和管理 Java 程序。**（实例位置：光盘\mr\ym\2.01）**

2.5.1 新建 Java 项目

（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，打开“新建 Java 项目”对话框。

（2）在“新建 Java 项目”对话框的“项目名”文本框中输入“2.01”。在“项目布局”栏中选中“为源文件和类文件创建单独的文件夹”单选按钮，如图 2.23 所示，单击“完成”按钮。

图 2.23 “新建 Java 项目”对话框

2.5.2 新建 Java 类

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 的主菜单中，选择“文件”/“新建”/“类”命令，打开“新建 Java 类”向导对话框，如图 2.24 所示。

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，通常向导会自动输入内容，没有特殊情况，不需要修改。

（2）在“包”文本框中输入类文件的包名，这里暂时默认为空，不输入任何内容，这样就会使用 Java 工程的默认色。

（3）在“名称”文本框中输入新建类的名称，这里输入“StudyJava”。

（4）选中 public static void main(String[] args)复选框，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。

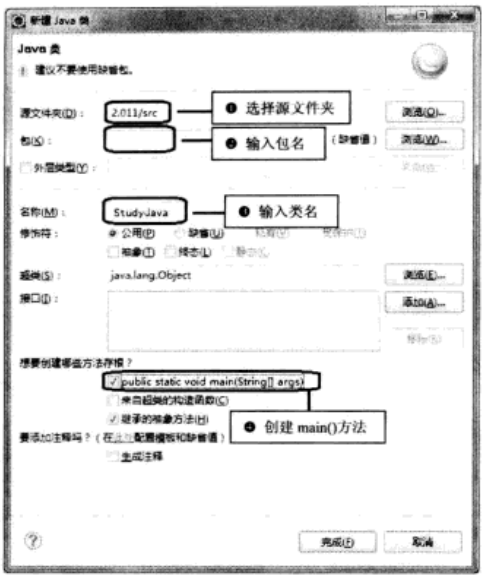


图 2.24　“新建 Java 类”向导对话框

2.5.3　编写 Java 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。本节将介绍如何使用 Java 编辑器和它的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，然后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 2.25 所示。

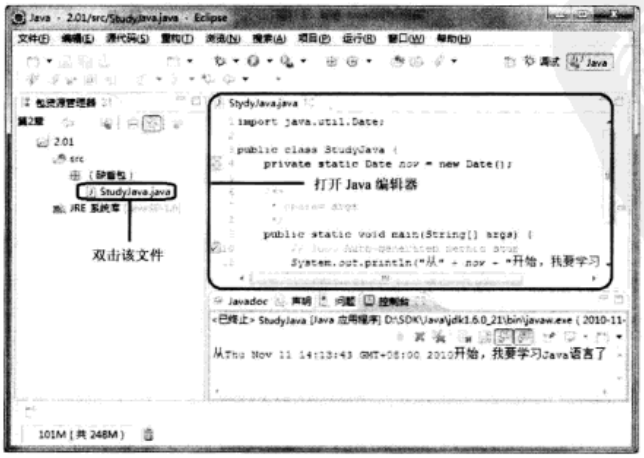


图 2.25　Java 编辑器界面

从图 2.25 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包括：

- ☑ 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 StudyJava 类的步骤如下：

在使用向导创建 StudyJava 类之后，向导会自动构建 StudyJava 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 2.26 所示。

```
1  import java.util.Date;
2
3  public class StudyJava {
4      private static Date now = new Date();
5
6      /**
7       * @param args
8       */
9      public static void main(String[] args) {
10         // TODO Auto-generated method stub
11         System.out.println("从" + now + "开始，我要学习Java语言了");
12     }
13
14 }
```

图 2.26　程序代码

在程序代码中，第 4 行和第 11 行是需要由程序开发人员编写的，其他代码都是自动生成的。

首先来看一下第 4 行代码，它包括 private、static 两个关键字和 Date 类，如果在记事本中手动输入可能不会花费多长时间，但是无法避免会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，另外还需要手动添加第 4 行引包的代码。

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，最后一个 Date 是一个类的名称，应该使用 Alt+/打开代码辅助菜单选择 Java.utility 包中的 Date 类，如图 2.27 所示。注意选择的 Date 类是 java.util 包中的，不要错选为 java.sql 包中的 Date 类，然后编辑器会自动在第一行添加引包语句 import java.util.Date。

其次是第 11 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，完成关键语法的录入，如图 2.28 所示。

图 2.27　使用代码辅助

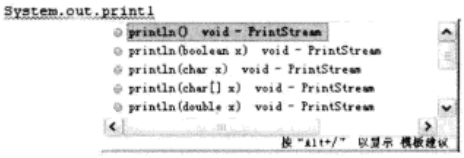


图 2.28　代码辅助菜单

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

技巧： 将光标移动到 Java 编辑器的错误代码位置，按 Ctrl+1 快捷键可以激活代码修正菜单，可以在菜单中选择合适的修正方法。

2.5.4　运行 Java 程序

StudyJava 类包含 main()主方法，它是一个可以运行的主类，在 Eclipse 中运行 StudyJava 程序，可以在“包资源管理器”视图中右击 StudyJava 文件，在弹出的快捷菜单中选择“运行方式”/“ Java 应用程序”命令，程序运行结果如图 2.29 所示。

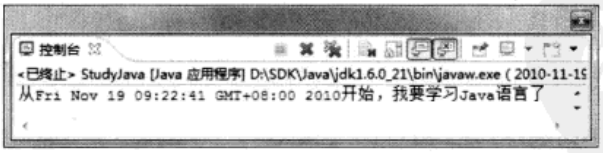


图 2.29　StudyJava 程序在控制台的输出结果

说明： 控制台中输出的日期字符串不是本地的日期格式，在以后的章节中会介绍如何格式化日期格式，到时候，读者可以尝试重新编写该示例，使用本地日期格式输出现在的字符串。

2.6　如何开发桌面应用程序

桌面应用程序又叫 GUI 应用程序，它运行于本地计算机，不需要服务器。Java 中使用 Swing 技术实现桌面应用程序的编写，所有窗体、菜单、对话框、按钮以及其他控件都是使用代码创建的，并且依靠代码

设置控件的属性和编写事件监听器。WindowBuilder 插件包含了界面设计功能，它可以使用可视化的 GUI 设计器进行程序界面设计，该设计器会自动生成相应的代码。

例 2.02 本节将带领读者完成一个简单的登录窗体界面的设计，介绍在 WindowBuilder 插件中如何创建 GUI 应用程序。**（实例位置：光盘\mr\ym\2.02）**

2.6.1 新建 Swing GUI 窗体

（1）首先创建一个 Java 项目，在项目的 src 文件夹上单击鼠标右键，在弹出的快捷菜单中选择“新建”/“其他”命令。

（2）在“新建”向导对话框中，展开 WindowBuilder/Swing Designer 节点，选择 JFrame 子节点，如图 2.30 所示，单击“下一步”按钮。

图 2.30 “新建”向导对话框

（3）在打开的 New JFrame 对话框的“名称”文本框中输入窗体类的名称“LoginFrame”，如图 2.31 所示，单击“完成”按钮。

图 2.31 New JFrame 对话框

（4）创建 LoginFrame 窗体后，会自动显示该文件的 Java 代码，选择编辑器底部的 Design 选项卡切换到设计器模式，效果如图 2.32 所示，可以选择 GUI 控件到设计器中设计窗体界面。

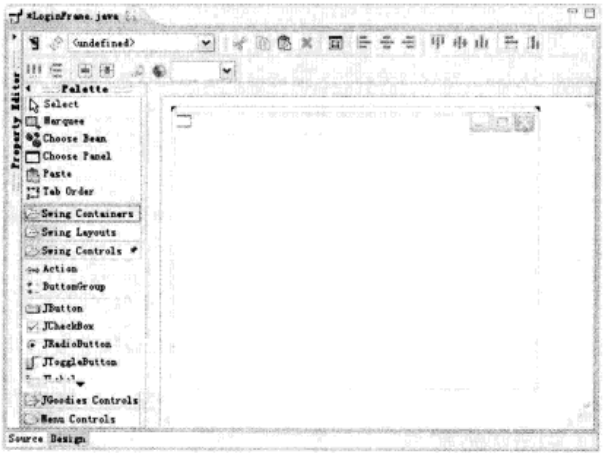


图 2.32　窗体设计界面

2.6.2　向窗体中添加 GUI 控件

（1）在设计器的窗体内部单击鼠标右键，在弹出的快捷菜单中选择 Set Layout/Absolute Layout 命令，这样就设置了窗体容器的布局管理器为绝对布局，因为现在还没有讲解布局管理器的知识，所以暂时使用绝对布局，即指定控件的 x、y 坐标位置和宽度、高度属性，这些都可以使用设计器完成。

（2）Palette 是调色板的意思，这里称为控件面板，它包含 Swing 控件、容器、菜单等内容。展开 Components 节点，这里包含了 Swing 的控件。选择 JLabel 控件，然后单击窗体设计器放置控件并调整位置，如图 2.33 所示。

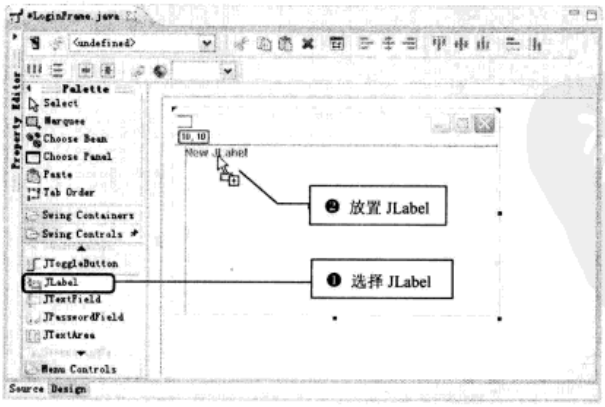


图 2.33　添加 JLabel 标签控件

（3）在 Properties 视图中，双击 text 属性值，设置属性值为“用户名”。该属性用于设置控件的文本内容，如图 2.34 所示。

（4）再添加一个 JTextField 文本框控件，调整控件的位置和宽度，如图 2.35 所示。

图 2.34　设置 text 属性

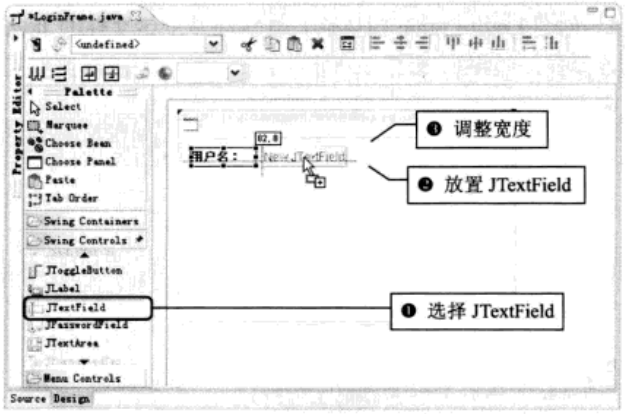


图 2.35　控件到设计界面的拖曳

（5）在设计器中选择刚添加的 JTextField 文本框控件，在 Properties 视图中单击 Variable 属性，设置控件的变量名称为 userNameC，如图 2.36 所示。

（6）参照步骤（1）和步骤（2）放置一个“密码”标签控件。

（7）从控件面板中选择 JPasswordField 密码框控件，将其添加到窗体中，并调整位置和大小，如图 2.37 所示。

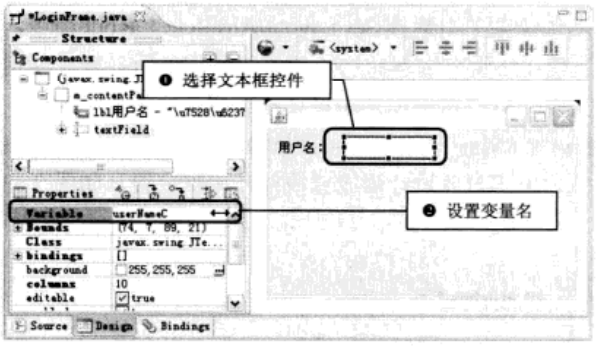


图 2.36　控件属性修改

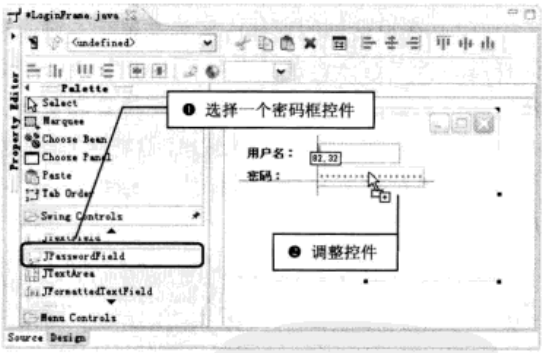


图 2.37　向设计器中添加密码框控件

（8）在设计器中选择刚添加的 JPasswordField 密码框控件，参照步骤（5）设置密码框控件的变量名为 passwordC。

（9）参照 Label 标签控件的添加步骤分别添加两个 JButton 按钮控件，并设置控件的 text 属性为“登录”与“退出”，然后调整窗体和所有控件的大小与位置，如图 2.38 所示。单击设计器工具栏中的按钮预览界面设计效果，如图 2.39 所示。

图 2.38　界面设计效果

图 2.39　界面预览效果

2.6.3　实现控件的事件监听器

事件监听器用于处理 GUI 界面发生的各种用户操作事件，如鼠标单击、按钮被按下等事件都由事件监听器负责监听并处理，下面介绍如何为“登录”按钮添加事件监听器。

（1）在设计器中双击“登录”按钮，设计器将转换到源代码编辑器界面，并添加了处理“登录”按钮动作事件的监听器代码，关键代码如下：

```
button.addActionListener(new ActionListener() {
    public void actionPerformed(final ActionEvent e) {
        // 这里输入代码
    }
});
```

（2）编写事件的监听器代码，输出窗体输入的用户名与密码，关键代码如下：

```
button.addActionListener(new ActionListener() {
    public void actionPerformed(final ActionEvent e) {
        // 这里输入代码
        System.out.println("单击了登录按钮");
        System.out.println("输入的用户名是：" + userNameC.getText());
        System.out.println("输入的密码是：" + passwordC.getText());
    }
});
```

2.6.4　运行 GUI 程序

（1）在“包资源管理器”视图中，展开 src/“缺省包”节点，选择 LoginFrame.java 文件，如图 2.40 所示。

（2）在 Eclipse 的主工具栏中单击 按钮运行，也可以在 LoginFrame.java 文件上单击鼠标右键，在弹出的快捷菜单中选择“运行方式”/“ Java 应用程序”命令，程序运行界面如图 2.41 所示。

（3）输入用户名和密码，单击“登录”按钮之后，在 Eclipse 的控制台会输出登录信息，如图 2.42 所示。

图 2.40　包资源管理器视图

图 2.41　运行效果

图 2.42　程序执行效果

2.7　如何进行程序调试

用户在程序开发过程中会不断地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（到处都是 System.out.println()方法）。

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。

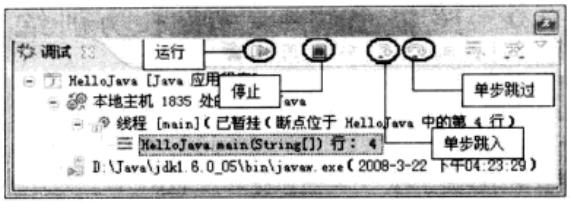


图 2.45　调试视图

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

☑　单步跳入

在调试视图的工具栏中单击按钮或按 F5 键，将执行该操作，跳入调用方法或者对象的内部单步执行程序并暂挂线程。

2.8　了解项目类库

Java 语言开发的项目应用都需要使用类库文件，这些类库就是由所谓的多个 JAR 文件组成的，JAR 文件是 Java 的打包文件，如 Spring、Hibernate、Struts 等是目前流行的框架技术，如果开发的项目使用了这三者中任何一个框架，都要把该框架中的 JAR 文件添加到项目的类库中，在 Eclipse 中称为构建路径，因为 Eclipse 在构建项目时需要这些类库。

为 Eclipse 构建路径添加类库的方法有很多种，在开始讲解这些不同的方法之前，先打开某个项目的“Java 构建路径”首选项设置界面，其步骤如下：

在需要添加类库到构建路径的项目名称上单击鼠标右键，在弹出的快捷菜单中选择“构建路径”/“配置构建路径”命令。这时将弹出该项目属性的“Java 构建路径”界面，选择“库”选项卡，如图 2.46 所示，这里显示的就是当前项目所引用的类库。

图 2.46　“Java 构建路径”界面

在这个首选项界面中的“库”选项卡可以通过界面右侧提供的按钮以各种方式为该项目添加需要的 JAR 类库。

2.9 照猫画虎——基本功训练

2.9.1 基本功训练 1——添加 JAR 文件到构建路径

视频讲解：光盘\mr\lx\02\添加 JAR 文件到构建路径.exe

最简单的涉及数据库操作的项目也要添加相应数据库的 JAR 类库，数据库驱动类和其他框架或组件的类库都是由单个或多个 JAR 文件组成的，所以需要把每个 JAR 文件添加到当前项目的构建路径中。

实现过程如下：

（1）在当前项目中新建一个文件夹，如 lib，并把需要用到的 JAR 文件都复制到该文件夹下。

（2）打开项目的“Java 构建路径”首选项设置界面，在“库”选项卡界面中单击右侧的“添加 JAR”按钮，弹出如图 2.47 所示的对话框，读者从项目中选择刚刚建立的 lib 文件夹中的所有 JAR 文件，单击“确定”按钮。

图 2.47 选择工作空间中的 JAR 文件

说明：这种添加类库的方式优点在于，所有 JAR 文件都是在项目文件夹中的，当把项目文件夹复制到另一个工作环境中时，所有 JAR 文件都和项目在一起。

照猫画虎：把 JAR 文件放在其他项目的文件夹中，在当前项目中添加其他项目中的 JAR 文件到构建路径中。（20 分）

2.9.2 基本功训练 2——添加依赖项目到构建路径

视频讲解：光盘\mr\lx\02\添加依赖项目到构建路径.exe

除了为构建路径添加 JAR 文件、库和类文件夹之外，还可以使当前项目依赖于其他项目，也就是说使用其他项目中定义的所有类，而不用管那个项目是如何添加类库的，下面介绍项目依赖关系的创建。

实现过程如下：

（1）在项目的“Java 构建路径”首选项设置界面中，选择“项目”选项卡，这时首选项界面如图 2.48 所示，单击右侧的“添加”按钮。

（2）打开如图 2.49 所示的“选择必需的项目”对话框，在其中选择一个或多个依赖的项目，单击“确

定”按钮，这样就添加了其他项目到当前项目的构建路径中。

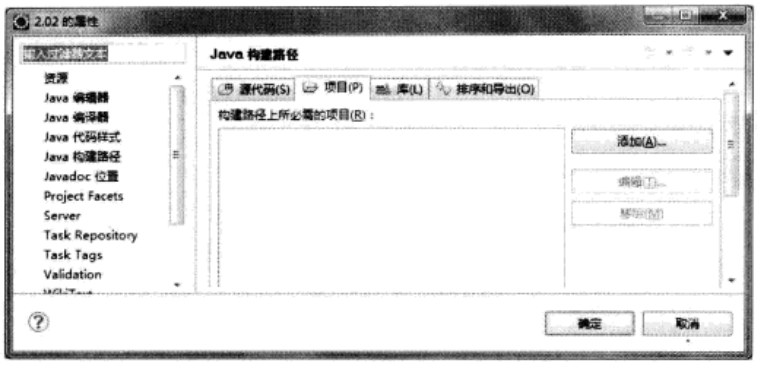


图 2.48　构建路径“项目”选项卡

图 2.49　“选择必需的项目”对话框

说明：这种方法可以使用依赖项目中编写的所有类。

照猫画虎：把多个项目作为当前项目的依赖项目。（20 分）

2.9.3　基本功训练 3——创建用户自定义类库

视频讲解：光盘\mr\lx\02\创建用户自定义类库.exe

像数据库、驱动类库这样只有少量 JAR 文件的，可以使用添加多个 JAR 文件的方法，但是像 Hibernate 那些包含大量 JAR 文件的框架，或者是同时使用多个框架的项目，需要添加大量的 JAR 文件到 Eclipse 的项目构建路径中，所有 JAR 文件都会显示在项目的“引用的库”子节点中，这样既不方便管理，也不方便阅读分析。

解决该问题的办法是创建用户库，可以将相同分类的所有 JAR 文件创建为一个指定名称的用户库，用户库是用户可以自定义的类库，在 Eclipse 工作空间中可以创建很多用户库，方便各种类库的分类。下面以 MySQL 数据库驱动类为例，创建名称为 MySQL_Driver 的自定义用户库。

实现过程如下：

（1）在 Eclipse 主菜单中选择“窗口”/“首选项”命令，在弹出的“首选项”对话框中，展开 Java/“构建路径”/“用户库”节点，将打开有关用户库管理的首选项设置界面，如图 2.50 所示。

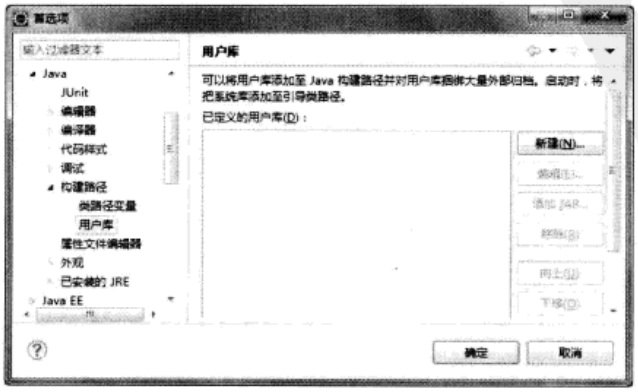


图 2.50　有关用户库首选项设置界面

（2）在用户库首选项设置界面右侧单击“新建”按钮，在弹出的“新建用户库”对话框中输入新的用户库的名称，如 MySQL_Driver，单击“确定”按钮。

（3）现在新建的用户库已经显示在用户库首选项设置界面中，选择新建的用户库，单击右侧的“添加 JAR”按钮，在弹出的对话框中选择需要的所有 JAR 文件，为用户库添加 JAR 文件后的界面如图 2.51 所示。单击“确定”按钮，MySQL_Driver 用户库就建立完成了。

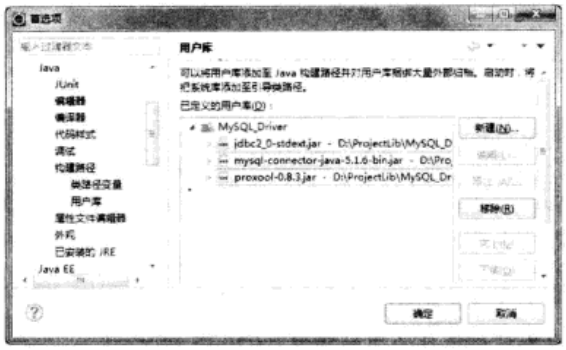


图 2.51　添加 JAR 文件后的用户库

说明：只有选择了用户库名称节点时，“添加 JAR”按钮才是可用的；如果选择的是用户库中的 JAR 文件节点，那么该按钮会变成灰色，即禁用状态。

照猫画虎：把多个项目作为当前项目的依赖项目。（20 分）

2.9.4　基本功训练 4——为项目添加类库

视频讲解：光盘\mr\lx\02\为项目添加类库.exe

添加的 JAR 文件过多会导致类库混乱不易管理，在学习了创建用户库以后，可以使用添加库功能把指定的库添加到构建路径中。这里所说的库包括自定义的用户库和各种插件带的库，例如，MyEclipse 就自带了很多框架的类库。下面介绍如何向构建路径中添加刚刚建立的用户库 MySQL_Driver。

实现过程如下：

（1）在项目的“Java 构建路径”首选项设置界面中单击“添加库”按钮，将出现如图 2.52 所示的对话框。在该对话框中包含了很多库的分类，例如，“JRE 系统库”是 Java 运行时的类库，是系统中安装的 JDK 或 JRE 包含的，排在最后的就是刚刚介绍过的“用户库”，这里选择它并单击“下一步”按钮。

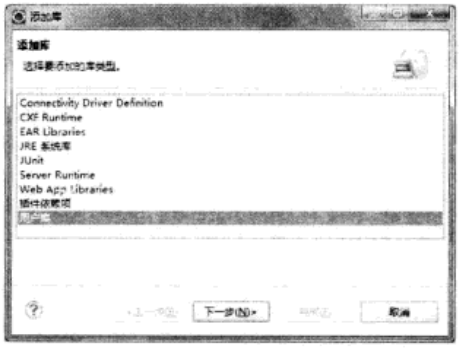


图 2.52　“添加库”对话框

（2）打开如图 2.53 所示的对话框，在其中选择项目需要的类库，这里选中刚刚建立的 MySQL_Driver 复选框，单击“完成”按钮。

图 2.53　选择需要的类库

说明：这种方法的优点在于，类库分类明确，方便管理与分析。但是缺点在于，类库管理的 JAR 文件在本地磁盘中，当单纯地复制项目文件夹到其他计算机中，或者将类库关联的 JAR 文件移动到别的文件夹中时，那么指定的用户库会失去作用，项目会出现错误，这时用户需要重新指定自定义用户库的 JAR 文件位置，或使用其他用户库。

照猫画虎：把 JRE 系统库从项目中移除。（20 分）

2.9.5　基本功训练 5——为项目添加类文件夹

视频讲解：光盘\mr\lx\02\为项目添加类文件夹.exe

Eclipse 中的 Java 项目包含 src 和 bin 文件夹，其中 src 文件夹中存放的是 Java 源代码文件，而 bin 文件夹用于存放编译后的 class 字节码文件。JAR 文件就是由这些 class 字节码文件压缩而成的，所以这些 class 文件应该能够代替 JAR 文件添加到构建路径中。

class 文件虽然数量比 JAR 文件多，而且比较零散，但是它和 JAR 文件添加到构建路径中的功效是一样的，其实现过程如下：

在项目的“Java 构建路径”首选项设置界面中单击“添加类文件夹”按钮，将出现如图 2.54 所示的“选择类文件夹”对话框，在该对话框中选择保存已经编译的 class 字节码文件的文件夹，单击“确定”按钮。

图 2.54　“选择类文件夹”对话框

照猫画虎：使用添加外部类文件夹的方式，引用磁盘 D 中保存的类文件夹到构建路径中。（20 分）

照猫画虎栏目分数统计：

照猫画虎题目	1	2	3	4	5	总分数	
分数							

2.10 情景应用——拓展与实践

2.10.1 情景应用 1——为窗体添加文本框控件

视频讲解：光盘\mr\lx\02\为窗体添加文本框控件.exe

实例位置：光盘\mr\ym\2.03\src

文本框是窗体界面中接收用户输入的控件，用户可以在其中输入任何字符，然后程序可以获取用户输入的内容进行运算或业务处理。现在通过 WindowBuilder 插件实现文本框到窗体界面的添加，实例运行结果如图 2.55 所示。

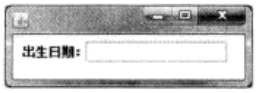


图 2.55 实例运行结果

实现过程如下：

（1）新建 Java 项目，在该项目中创建窗体类。

（2）设置窗体布局管理器为 Absolute Layout，也就是绝对定位的布局方式。

（3）在控件面板中选择JLabel标签控件，并将其添加到窗体左上角，如图 2.56 所示。

（4）设置标签控件的 text 属性为“出生日期:”。

（5）在控件面板中选择JTextField控件，然后添加到窗体中，并调整文本框的宽度，如图 2.57 所示。

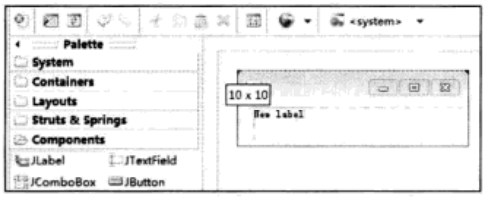


图 2.56 添加 JLabel 标签控件

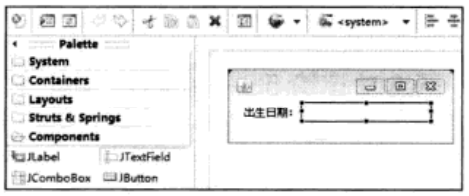


图 2.57 添加文本框控件到窗体中的效果

DIY：为程序界面添加控件，组成用户可以输入姓名和出生日期的界面。（20 分）（实例位置：光盘\mr\ym\2.03\diy）

2.10.2 情景应用 2——接收用户在控制台的输入

视频讲解：光盘\mr\lx\02\接收用户在控制台的输入.exe

实例位置：光盘\mr\ym\2.04\src

程序的主要职责之一就是与用户交互，如果为学习过程中的实例也添加与用户的交互能力，不但可以提高程序的趣味性，也可以增强程序的灵活处理能力。下面介绍的实例可以接收用户的输入，然后输出针对该输入内容的提示信息，实例运行结果如图 2.58 所示。

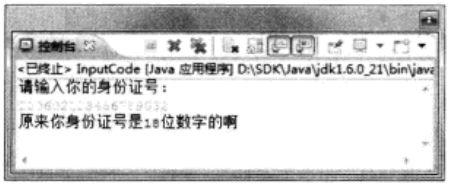


图 2.58 实例运行结果

实现过程如下：

（1）在 Eclipse 中创建 Java 项目，在项目中创建 InputCode 类。

（2）在 InputCode 类的主方法中编写代码接收用户输入的字符串，然后输出该字符串的长度，实例代码如下：

```
import java.util.Scanner;
public class InputCode {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);     // 创建输入流扫描器
        System.out.println("请输入你的身份证号：");     // 提示用户输入
        String line = scanner.nextLine();             // 获取用户输入的一行文本
                                                      // 打印对输入文本的描述
        System.out.println("原来你身份证号是" + line.length() + "位数字的啊");
    }
}
```

你问我答：如何接收用户输入的字符串并计算其长度？

（1）接收用户输入的字符串

Java 内置的 System 类不仅包含输出功能的 out 成员变量，它还接收数据的 in 成员变量，它们都是数据流，目前读者只需要知道它们用于输入输出即可。获取输入流的数据可以使用 Scanner 扫描器类和其方法，创建该类实例对象的关键代码如下：

```
Scanner scanner = new Scanner(System.in);
```

获取用户输入的一行文本字符串，需要调用扫描器的 nextLine()方法，其语法格式如下：

```
public String nextLine()
```

参数说明：该方法会返回一个字符串，该字符串就是用户输入的一行文本。例如，本实例中获取用户输入的文本并保存为字符串变量的关键代码如下：

```
String line = scanner.nextLine();
```

（2）计算字符串长度

字符串有一个长度属性，它是只读的，可以通过字符串提供的方法访问该属性来获取字符串的长度，其语法格式如下：

```
public int length()
```

参数说明：该方法会返回一个整数类型的值，这个值就是字符串的长度。例如，本实例是在输出语句中获取字符串的长度，并输出到控制台的关键代码如下：

```
System.out.println("原来你身份证号是" + line.length() + "位数字的啊");
```

DIY：编程接收用户输入的姓名，并输出针对该姓名的问候信息。（20 分）（实例位置：光盘\mr\ym\2.04\diy）

2.10.3　情景应用 3——编写一个信息提示窗体

视频讲解：光盘\mr\lx\02\编写一个信息提示窗体.exe

实例位置：光盘\mr\ym\2.05\src

信息提示对话框可以作为向用户说明程序当前情况与问题的工具。Java 语言的 JOptionPane 类已经提供了多种类型的信息提示，这些类型包括普通消息、警告、错误、疑问等。本实例实现一个 GUI 界面的程序，

如图 2.59 所示，用户可以在文本框中输入提款金额。程序根据用户输入，弹出对话框显示针对该输入数据的提示信息，如图 2.60 所示。

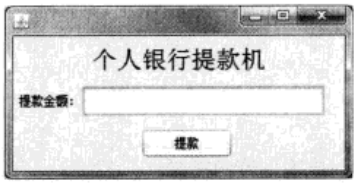


图 2.59　实例运行界面

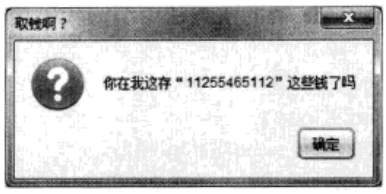


图 2.60　信息提示对话框

实现过程如下：

（1）在 Eclipse 中创建 Java 项目，在项目中创建窗体类 MessageDialog。

（2）设置窗体布局管理器为 Absolute Layout，也就是绝对定位的布局方式。

（3）为 MessageDialog 窗体添加 JLabel 控件、JTextField 文本框控件和 JButton 按钮控件，并设置它们的 text 属性，让这些控件组成如图 2.59 所示的窗体界面。

（4）编写“提款”按钮的事件处理。当用户单击该按钮时弹出如图 2.60 所示的信息提示对话框，该按钮事件处理的关键代码如下：

```
String text = textField.getText();                    // 获取文本框输入
// 显示疑问类型的信息提示对话框
JOptionPane.showMessageDialog(this, "你在我这存"" + text + ""这些钱了吗", "取钱啊？",
        JOptionPane.QUESTION_MESSAGE);
```

你问我答：如何显示信息提示对话框？

本实例也使用到了 JOptionPane 类的 showMessageDialog()方法来实现信息提示对话框的显示，该方法的语法格式如下：

```
public static void showMessageDialog(Component parentComponent,
                                     Object message,
                                     int messageType,)
                              throws HeadlessException
```

参数说明：parentComponent 是对话框的父窗体，本实例中使用 this 引用当前窗体类作为其父窗体；message 是对话框显示的信息字符串；messageType 是对话框类型，这个类型值确定对话框的信息图标样式。

DIY：从控制台接收用户输入的问候信息，并使用信息提示对话框显示。（20 分）（实例位置：光盘\mr\ym\2.05\diy）

2.10.4　情景应用 4——控制窗体位置

视频讲解：光盘\mr\lx\02\控制窗体位置.exe

实例位置：光盘\mr\ym\2.06\src

窗体位置的设置应该更加灵活地显示在屏幕上的任意位置，本实例将接收用户指定的屏幕坐标，来控制窗体位置的显示，实现窗体位置自定义。在如图 2.61 所示的两个文本框中分别输入窗体距离屏幕左边距与上边距的位置，并单击“设置”按钮就会改变窗体的位置。

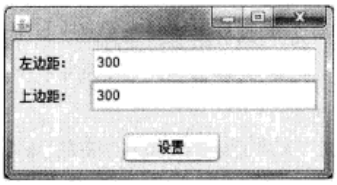


图 2.61　设置窗体在屏幕中的位置

实现过程如下：

（1）在 Eclipse 中创建 Java 项目，在项目中创建窗体类 SetLocation。

（2）设置窗体布局管理器为 Absolute Layout，也就是绝对定位的布局方式。

（3）在 SetLocation 窗体中添加两个标签控件、两个文本框控件和一个按钮控件。

（4）编写“设置”按钮的事件处理方法，在方法中获取用户在文本框中输入的数值字符串，然后把这个字符串转换为整数类型，并依次设置窗体的位置，关键代码如下：

```
protected void do_button_actionPerformed(ActionEvent e) {
    Object value = leftField.getValue();          // 获取左边距坐标文本
    Object value2 = topField.getValue();          // 获取上边距坐标文本
    if (value == null || value2 == null)
        return;
    int left = ((Number) value).intValue();       // 提取左边距坐标值
    int top = ((Number) value2).intValue();       // 提取上边距坐标值
    setLocation(left, top);                       // 用左边距和上边距坐标值设置窗体位置
}
```

你问我答：如何获取用户输入的整数和设置窗体位置？

（1）获取文本框输入值

文本框控件用于接收用户输入的字符串，通过文本框控件的 getText()方法可以获取这个字符串，其语法格式如下：

```
public String getText()
```

参数说明：这个方法将用户输入的字符串作为返回值，String 就是返回值的类型。

（2）字符串转换为整数

文本框只能接收用户输入的字符串，而程序要运算的是整数，所以必须把字符串 String 转换为 Integer 类的实例对象，这需要使用 Integer 类的 valueOf()方法，其语法格式如下：

```
public static Integer valueOf(String s)
```

参数说明：s 是字符串参数，它必须由数字组成，不能包含非数字的其他字符，方法的返回值是 Integer 类的实例对象。

Integer 类的实例对象包含很多可以调用的方法，其中的 intValue()方法可用于提取整数类型的数值，其语法格式如下：

```
public int intValue()
```

参数说明：该方法的返回值是 int 类型的数值，它可以参与数学运算。例如，本实例中把字符串转换为 Integer 类的实例对象，并获取整数值使用的代码如下：

```
Integer int1 = Integer.valueOf(value);
Integer int2 = Integer.valueOf(value2);
```

```
int left = int1.intValue();                    // 提取左边距坐标值
int top = int2.intValue();                     // 提取上边距坐标值
```

（3）设置窗体位置

本实例设置窗体在屏幕中的位置时，主要通过 JFrame 类的 setLocation()方法实现，该方法将窗体设置到新位置。通过 x 和 y 参数来指定新位置的左上角，其语法格式如下：

```
public void setLocation(int x, int y)
```

参数说明：x 是窗体水平坐标的新位置，y 是窗体垂直坐标的新位置。

DIY：编写一个固定窗体位置的程序界面，为窗体添加两个文本框控件来接收用户输入的整数，然后在按钮的事件处理方法中使用信息提示对话框输出它们的和。（20 分）（实例位置：光盘\mr\ym\2.06\diy）

2.10.5 情景应用 5——编写 UI 界面的加法计算器

视频讲解：光盘\mr\lx\02\编写 UI 界面的加法计算器.exe

实例位置：光盘\mr\ym\2.07\src

使用界面设计器，设计一个类似 Windows 系统的计算器程序界面，效果如图 2.62 所示，设计时不考虑菜单栏。

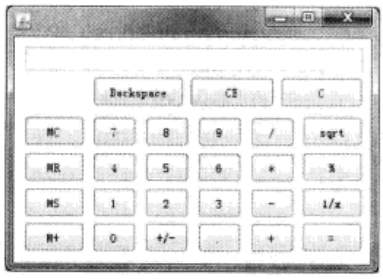


图 2.62 运行结果

实现过程如下：

（1）在项目中创建主窗体类 Calculator。

（2）设置窗体布局管理器为 Absolute Layout，也就是绝对定位的布局方式。

（3）在窗体顶部添加一个JTextField文本框组件，调整文本框与窗体宽度对应，如图 2.63 所示。

（4）在窗体设计界面的右侧，添加“sqrt”、“%”、“1/x”、“=”4 个按钮，如图 2.64 所示。设置“=”按钮的 foreground（前景色）属性为 RED，设置其他按钮的该属性为 BLUE。

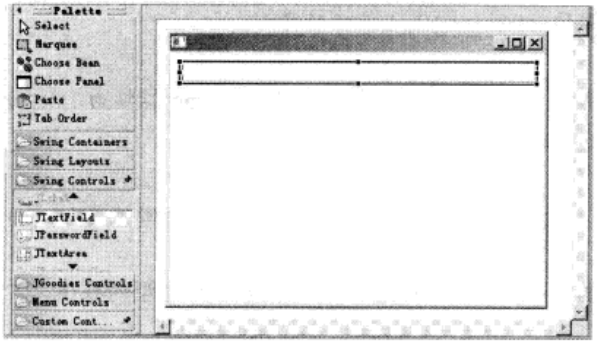


图 2.63 添加文本框

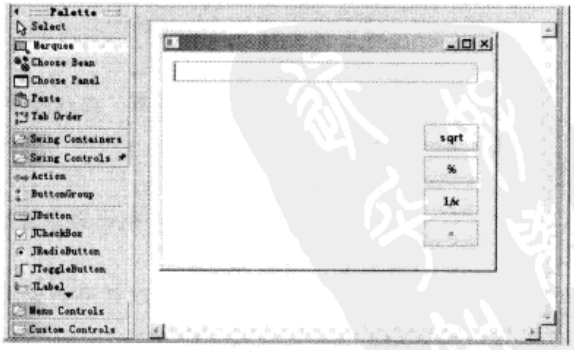


图 2.64 设置 foreground 属性

（5）通过Marquee选项可以实现鼠标划动选择多个按钮或其他组件，然后通过复制、粘贴命令可以直接添加相同的一组按钮到设计器中，如图 2.65 所示。

（6）把复制的按钮进行调整和设置，把它们定义成运算符号的按钮，如图 2.66 所示。

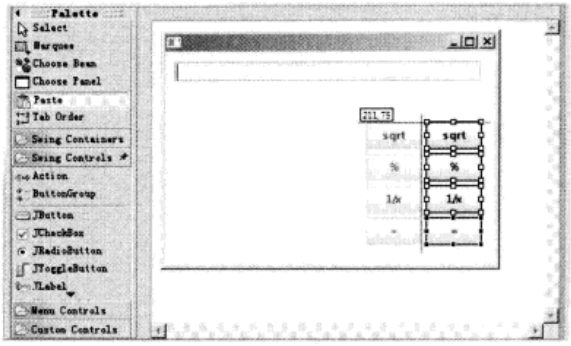


图 2.65　添加组按钮

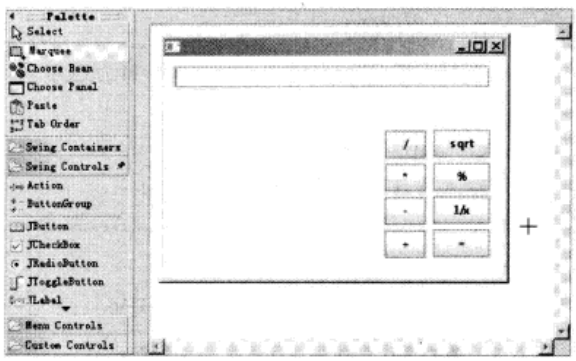


图 2.66　调整和设置复制的按钮

（7）继续添加其他按钮，如果个别按钮因文字太长而显示不全，可以尝试调整该按钮的 Margin 属性，把左右边界调整一下，如图 2.67 所示。

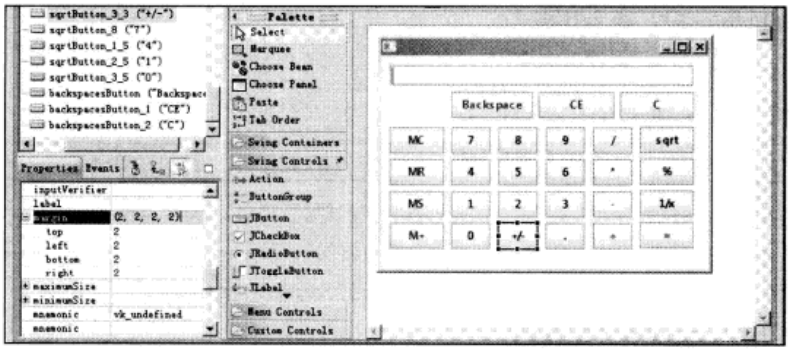


图 2.67　添加并修改新按钮

DIY：修改程序，为“=”按钮编写事件监听方法，当用户单击该按钮时，通过信息提示对话框显示“程序正在完善”的信息。（20 分）（实例位置：光盘\mr\ym\2.07\diy）

情景应用 DIY 栏目分数统计：

DIY 题目	1	2	3	4	5	总分数	
分数							

2.11　自 我 测 试

一、选择题（每题 10 分，5 道题）

1．下面哪个图标是 Eclipse 开发工具的图标？（　　）

A．　　B．　　C．

2．Eclipse 界面中哪个视图用于显示程序运行时输出的调试信息和（Runtime Exception）运行时异常信息？（　　）

A．包资源管理器视图　　B．　控制台视图

C．大纲视图　　D．问题视图

3．下面网址中，哪个是 Eclipse 的官方网址。（　　）

A．http://www.eclipse.org　　B．http://www.eclipse.com

C．http://www.eclipse.net　　D．http://www.eclipse.cn

4．输入文字使用哪个控件（　　）

A．JLabel　　B．JTextField　　C．JButton

5．Eclipse 是由哪个公司开发的。（　　）

A．Sun　　B．IBM　　C．微软　　D．谷歌

二、填空题（每题 10 分，5 道题）

1．Eclipse 的_____插件可以实现界面设计器功能。

2．Eclipse 的_____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。

3．_____能够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。

4．第一次启动 Eclipse 时，要为其指定_____文件夹。

5．在以_____方式运行程序时，断点才起作用。

测试分数统计：

类别	第 1 题	第 2 题	第 3 题	第 4 题	第 5 题
选择题分数					
填空题分数					
总分数					

2.12 行 动 指 南

开始日期：_____年___月___日　　结束日期：_____年___月___日

序号	内　容	行 动 指 南	
1	照猫画虎栏目 分数（　　）	分数>75 分	优秀，基本功掌握得不错，加油！
		75 分>分数>50 分	及格，知识掌握得不牢，重新做一遍照猫画虎。
		分数<50 分	请使用光盘中提供的“实战能力测试系统”进行提高训练。
	情景应用栏目 分数（　　）	分数>75 分	优秀，综合应用能力很强。
		75 分>分数>50 分	及格，综合应用能力需提高，再练一遍情景应用。
		分数<50 分	请使用光盘中提供的“实战能力测试系统”进行提高训练。
	自我测试栏目 分数（　　）	分数>75 分	优秀，有成为编程高手的潜质
		分数<75 分	请使用光盘中提供的“实战能力测试系统”进行提高训练。
	综合评价	反复训练后，以上各项分数都在 75 分以上，方可进入下一堂课学习。	
2	编程能力培养：本栏目提供了一些实践题目，对于培养编程能力很有效，要做好。	（1）到 Eclipse 网站查询最新的 Eclipse 版本并更新自己的 Eclipse。	
		（2）关闭 Eclipse 中所有视图，然后在透视图图标上单击鼠标右键，找到恢复视图布局的菜单项并恢复视图布局。	
		（3）在控制台输出字符组成的方向箭头，箭头的方向任意。	
		（4）在程序中使用信息提示对话框显示问候信息。	

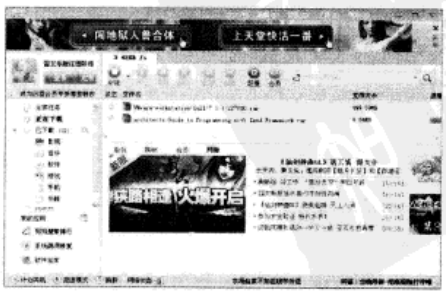
续表

序号	内　容	行 动 指 南
3	编程习惯培养：本堂课培养读者在学习开发中记笔记的习惯，把开发中遇到的问题、总结的经验记录下来。	
4	创新能力培养：看看身边有没有可以用编程解决的问题，记录到右边的表格中。根据学习进度，尝试编写解决这些问题的小程序。	

2.13　成功可以复制——迅雷创始人邹胜龙

“迅雷”于 2002 年年底由邹胜龙及程浩先生始创于美国硅谷。2003 年 1 月底，邹胜龙回国发展，在深圳市创办三代科技开发有限公司。2005 年 5 月，公司正式更名为深圳市迅雷网络技术有限公司，就是下载速度可达到迅雷不及掩耳之势。

“迅雷”立足于为全球互联网提供最好的多媒体下载服务。经过艰苦创业，“迅雷”在大中华区域以领先的技术和诚信的服务，赢得了广大用户的深深喜爱和许多合作伙伴的认同与支持。公司旗舰产品——迅雷，已经成为中国互联网最流行的应用服务软件之一。作为中国最大的下载服务提供商，迅雷每天服务来自几十个国家、超过数千万次的下载。现在迅雷每天为全球互联网传送 3200000GB 的资源，并以其下载软件的迅捷高速，每天为人类节省 1500 年时间。在本土，迅雷的市场覆盖率达 85%；迄今，全球已有 1.88 亿网民体验过迅雷提供的服务。随着中国互联网宽带的普及，迅雷凭借“简单、高速”的下载体验，已经成为高速下载的代名词。此外，“迅雷”也获得了晨光科技和 IDGVC 等数家知名风险投资企业的认同和合资。2007 年 1 月，迅雷宣布第三次融资成功，本次融资的领衔投资是联创策源（Ceyuan Ventures），参与投资的有晨光（Morningside Ventures）、IDGVC、Fidelity Asia Ventures，战略投资是 Google（谷歌）。这些投资合作伙伴除了给“迅雷”带来更加雄厚的资金实力外，也给“迅雷”带来了更丰富的行业资源和国际化公司运行。



迅雷界面

经典语录

迅雷的团队是一支学习能力和执行能力都很强的队伍，他们是中国互联网最优秀的团队之一。

深度评价

在网易科技采访邹胜龙时说：“您已经入选超过 5 万网友选出的互联网领袖扑克牌，并且进入了前十名，方便谈一下您的感受吗？”邹胜龙说：“非常荣幸！不过，我个人感觉受之有愧！其实，大家对我的厚爱主要是来自于对于迅雷的认识，而迅雷主要是由我的合伙人程浩和我们的团队一起建设的。相比之下，迅雷还非常年轻，还有很长的道路要走下去。”通过邹先生的这一番话可以了解到，任何一个成功的项目都不是一个人完成的，想要在 IT 领域中获得成功，就要做好团队合作的准备，凝聚大家的力量，发挥大家的聪明才智，才能创造财富。

第 3 堂课

Java 语言基础知识

（视频讲解：178 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，这种错误的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。本堂课对 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堂课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。

学习摘要：

- Java 主类结构
- 基本数据类型
- 常量与变量
- 各种运算符的使用
- 数据类型的转换
- 代码注释与编码规范

3.1 Java 主类结构

Java 语言是面向对象的程序设计语言，Java 程序的基本组成单元就是类，类体中又可包括属性与方法两部分。每一个 Java 本地应用程序都必须包含一个 main()方法，含有 main()方法的类称之为主类。下面通过程序来介绍 Java 主类结构。

例 3.01 创建包 myclass.struct，创建类 First。在类体中输入如图 3.1 所示的代码，实现在控制台上输出“让我看看，主类的结构。”文本内容。（**实例位置：光盘\mr\ym\3.01**）

```
1  package myclass.struct;                          // 定义类包
2
3  public class First {                             // 声明类
4      private static String s1 = "让我看看，";      // 成员变量
5
6      public static void main(String[] args) {     // 主方法
7          String s2 = "主类的结构。";               // 局部变量
8          System.out.print(s1);                    // 输出变量s1
9          System.out.println(s2);                  // 输出变量s2
10     }
11 }
```

图 3.1 实例代码

运行结果如图 3.2 所示。

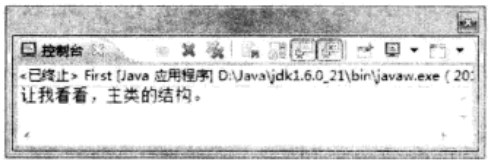


图 3.2 实例运行结果

3.1.1 定义类包

实例 3.01 中第 1 行语句 package myclass.struct;用于声明 First 类所在的包。package 为声明包的关键字。使用包可以分类存储不同的 Java 类，使代码更便于管理，本书将在第 6 堂课对包进行讲解。

3.1.2 定义类

一个 Java 应用程序是由若干个类组成的。例 3.01 中第 3 行代码就声明了一个 Java 类，它是本堂课的第一个类，所以取名为 First。

3.1.3 导入 API 类库

Java 类必须执行导入才能够被当前类使用，在 Java 语言中可以通过 import 关键字导入相关的类。

注意：Java 语言是严格区分大小写的。例如，关键字 class 不等于 Class，同理 First 与 first 是两个不同的 Java 类。

3.1.4　定义变量

通常将类的属性称为类的全局变量（或成员变量），将方法中的属性称之为局部变量。全局变量声明在类体当中，局部变量声明在方法体当中。全局变量和局部变量都有各自的应用范围。例 3.01 代码中第 4 行代码定义的 s1 是成员变量，而第 7 行代码定义的 s2 是局部变量。

3.1.5　定义主方法

main()方法是类体中的主方法。该方法从“{”号开始，至“}”号结束，中间的是方法体。public、static、void 分别是 main 方法的权限修饰符、静态修饰符和返回值类型声明，在 Java 程序中的 main()方法必须声明为 public static void。String arg[]是一个字符串类型的数组，它是 main()方法的参数。main()方法是程序开始执行的位置。

说明：main()方法是程序入口方法，它有固定的格式，方法的参数也固定为一个字符串数组。现在，读者只要记住主方法的格式就可以。在本书的基础章节，所有实例都比较简单，所以基本上都是在 main()方法中运行的。

3.1.6　运行主类

Java 类需要编译成以.class 为后缀的二进制字节码文件才能够运行，在 Eclipse 开发工具中，编译步骤是自动的，可以直接运行主类程序。方法是：在“包资源管理器”视图中找到要运行的类文件（该类必须包含 main()方法），在该文件上单击鼠标右键，在弹出的快捷菜单中选择“运行方式”/“ Java 应用程序”命令。

3.2　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。在 Java 源程序文件的任意位置都可添加注释语句。Java 编译器并不对注释中的文字进行编译，代码中的所有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。

3.2.1　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：

```
// 注释内容
```

例如，以下代码为声明的 int 型变量添加注释。

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

3.2.2　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。注释当中的内容可以换行。

语法：

```
/*
注释内容 1
注释内容 2
...
*/
```

3.2.3 JavaDoc 文档注释

“/**”与“*/”是文档注释的起始和结束标记，它们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 文档的内容。文档注释的格式与多行注释的格式相同，对于初学者来说仅了解文档注释即可。例如，为 First 类的主方法添加文档注释的关键代码如下：

```
/**
 * 这是 First 类的主方法，它将输出成员变量 s1 和局部变量 s2 的连接字符串。
 * @param args 主方法的入口参数，用于接收命令行参数
 */
public static void main(String[] args) {                    // 主方法
    String s2 = "主类的结构。";                               // 局部变量
    System.out.print(s1);                                    // 输出变量 s1
    System.out.println(s2);                                  // 输出变量 s2
}
```

添加了文档注释的类成员，会在 Eclipse 的 JavaDoc 视图中显示相应的说明。另外，在编辑器中把鼠标停留在该方法或引用该方法的位置上，也会出现悬浮提示信息，如图 3.3 所示。

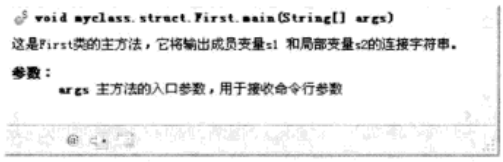


图 3.3 类成员文档注释信息

技巧：在 Java 源文件编辑器中，选择某个成员方法或成员变量，然后按 Shift+Alt+J 快捷键，Eclipse 会自动添加 JavaDoc 文档注释结构，如果是方法，还会自动添加参数名称。

3.3 变量与常量

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。变量与常量的命名都必须使用合法的标识符。本节将向读者介绍标识符与关键字以及变量与常量的命名。

3.3.1 Java 保留字

1. 标识符

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

Java 语言规定标识符由任意顺序的字母、下划线（_）、美元符号（$）和数字组成，并且第一个字符不能是数字，不能是 Java 中的保留关键字。

下面是合法标识符：

```
name
user_age
$page
```

下面是非法标识符：

```
4word                    // 第一个字符不能是数字
int                      // 不能使用 Java 中的保留关键字
User name                // 字符不能有空格
```

在 Java 语言中，标识符中的字母是严格区分大小写的，如 good 和 Good 是不同的两个标识符。Java 语言使用 unicode 标准字符集，最多可以标识 65535 个字符。因此，Java 语言中的字母不仅包括通常的拉丁文字 a、b、c 等，还包括汉字、日文以及其他许多语言中的文字。

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 中的关键字

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

3.3.2　定义变量

变量可以理解为日常生活中的一个容器，例如，箱子可以装衣服、金银首饰、图书、旅行用品等，关键时刻还可以藏人，总之变量的内容是可以改变的。变量的使用是程序设计中一个十分重要的环节。

为什么要定义变量呢？简单地说，就是要告诉编译器（compiler）这个变量是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给它以及它能存放什么样的数据。在程序运行过程中，空间内的值是变化的，所以称为变量。为了便于操作，给这个空间取个名字，称为变量名，内存空间内的值就是变量值。变量的命名必须是合法的标识符。在声明变量时可以不赋值，也可以直接赋给初值。

例如：

```
int age;                 // 声明没有赋值的 int 型变量
double price = '35.96';  // 声明 double 型变量并赋值
```

当编写以上程序代码时，究竟会产生什么样的作用呢？要了解这个问题，就需要对变量的内存配置有一定的认识。笔者用图解的方式，将上例程序代码在内存中的状况表现出来，如图 3.4 所示。

由图 3.4 可知，系统的内存可大略地分为 3 个区域，即系统区（OS）、程序区（Program）、数据区（Data）。当程序执行时，程序代码会加载到内存的程序区中，数据暂时存储在数据区中。对于变量的命名并不是任

意的，应遵循以下几条规则：

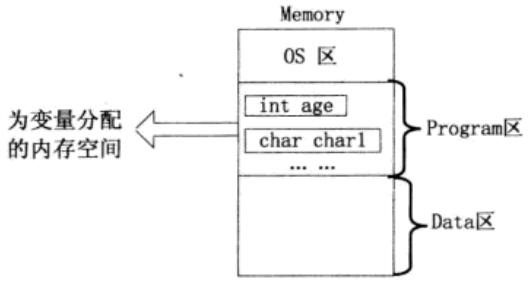


图 3.4　变量占用的内存空间

- ☑ 变量名必须是一个有效的标识符。
- ☑ 变量名不可以使用 Java 中的关键字。
- ☑ 变量名不能重复，在其作用范围内必须是唯一的。
- ☑ 应选择有意义的单词作为变量名，做到见其名知其意。

📖 **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。**

3.3.3　定义常量

常量是不能被改变的量，它和变量的性质完全相反。Java 使用关键字 final 来声明常量，并且常量在声明之后只能够初始化一次。

语法：

final 数据类型 常量名称[=值]

参数：常量名通常使用大写字母，这也符合 Java 编码规范，但这并不是必需的。声明常量时，完全可以和变量名一样使用小写字母，但那样容易造成混淆，降低代码的可读性，所以才有规范要求常量名全部大写，并使用下划线“_”字符分割多个单词。

例如：

```
final double PI=3.1415926;                    // π值
final String ITEM_NAME="进销存管理系统";       // 项目名称
final int COMMAND_CLOSE=908;                  // 命名编码
```

全局常量（或成员常量）被声明之后，必须马上初始化，或者在类的所有构造方法中进行初始化，否则编译器会报错。

3.3.4　常量与变量的有效范围

变量被声明与初始化之后，只是暂存在内存中，当变量不再被程序使用时，将被销毁并释放所占用的内存空间，也就是说变量有它的生命周期。每个变量的生命长短和它的有效范围息息相关，所谓有效范围就是变量在程序中起作用的区域，若超出该区域则无法访问该变量。变量根据有效范围的不同分为全局变量和局部变量。

☑ 全局变量

全局变量又称为成员变量，定义在类体中，它的有效范围是整个类的代码段，也就是在类体中的任何

位置都可以使用该变量。

例如：

```
public class Temp {
    int num=0;
    float price=3.50f;
    public void method1(){          // 方法 1 访问变量 num 和 price
        num=1;
        price=12.55f;
    }
    public void method2(){          // 方法 2 也可以访问变量 num 和 price
        num=num*10;
        price=price*0.5f;
    }
}
```

全局变量 num 和 price 在 method1()方法中被重新赋值，在 method2()方法中也同样可以被访问。

☑ 局部变量

在方法体中定义的变量就是一种局部变量，局部变量只在当前代码块（即花括号内）中有效。局部变量的生命周期取决于声明位置的代码块，超出这个代码块的范围就不能再使用代码块内的局部变量。

对方法来说，方法内声明的变量、包括参数变量都属于局部变量。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，局部变量也将会被销毁，然后 Java 的垃圾回收机制会在某一时刻清理该内存空间。

局部变量可与全局变量的名字相同，此时全局变量将被隐藏，但是可以使用 this.作为前缀来访问全局变量。

例 3.02　在项目中创建类 Val，分别定义名称相同的局部变量与全局变量，当名称相同时成员变量将被隐藏。代码如下：（**实例位置：光盘\mr\ym\3.02**）

```
public class Val {                                          // 新建类
    static String words = "全局变量";                        // 定义全局变量 words
    public static void main(String[] args) {                // 主方法
        String words = "局部变量";                           // 定义局部变量 words
        System.out.println("words 变量现在是：" + words);     // 将 words 的值输出
    }
}
```

运行结果如图 3.5 所示。

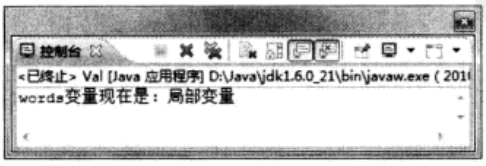


图 3.5　实例运行结果

3.4　数 据 类 型

Java 的数据类型分为基本数据类型和对象引用类型。在介绍面向对象知识之前，先来介绍基本数据类型，在 Java 中有 8 种基本数据类型来存储数值、字符和布尔值，如图 3.6 所示。

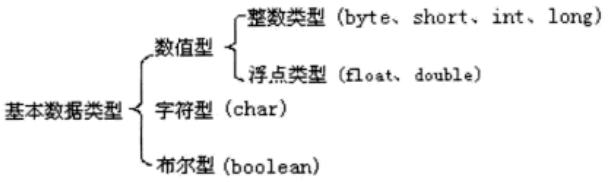


图 3.6　基本数据类型

3.4.1　计算机常用数制

现在的计算机功能丰富、性能优越，可以听歌、看电影、处理图像、进行 3D 游戏等。但它终究是一个电子产品，这些丰富的资源存储在计算机中只能使用 0 和 1 组成的数字表示，它们对应着电子晶体管的“关”和“开”状态，这就是所谓的二进制。例如，数字 128 在计算机中的二进制表现形式为 10000000。一个非常简单的数字存储到计算机中就变得非常繁琐、难以计算，所以又出现了八进制与十六进制，另外，在现实社会中，使用最多的是十进制，如重量、价格、年龄都是使用十进制表示的。而程序需要人来编写，所以，为迎合人类的计数习惯，几乎所有高级程序设计语言中的数值都以十进制作为默认数制，本节将介绍几种常用的数制。

1．十进制

十进制就是“逢十进一”，这是我们最熟悉的进制了。当我们去商场购物时，对货币的计算都要使用十进制进行计算，9 元钱加上 1 元钱就是 10 元钱，这就是“逢十进一”，进位后就成了 10。

2．二进制

所谓二进制就是“逢二进一”，它的字面量只包含 0 和 1。在二进制的预算法则中，1+1 不再等于 2，而是等于 10，二进制的 10 与十进制的 2 是相等的。

3．八进制

八进制的最大单个数字是 7，7 再加 1 就是 10，它和十进制的 8 是相等的。八进制的字面量包括 0、1、2、3、4、5、6、7。

4．十六进制

十六进制主要用于二进制的简化表达方式，十六进制的 1 位数字可以转换成 4 位二进制数字。超出 9 的数字在十六进制中以字母表示，其字面量包括 0、1、2、3、4、5、6、7、8、9、A、B、C、D、E、F，其中 A～F 分别代表十进制的 10～15。

常用进制的字面量对比如表 3.2 所示。

表 3.2　进制转换表

十六进制	二　进　制	十　进　制	八　进　制
0	0000	0	0
1	0001	1	1
2	0010	2	2
3	0011	3	3

续表

十六进制	二　进　制	十　进　制	八　进　制
4	0100	4	4
5	0101	5	5
6	0110	6	6
7	0111	7	7
8	1000	8	10
9	1001	9	11
A	1010	10	12
B	1011	11	13
C	1100	12	14
D	1101	13	15
E	1110	14	16
F	1111	15	17

技巧：可以使用 Windows 系统的计算器方便地转换不同进制之间的数值。选择“开始”/“所有程序”/“附件”/“计算机”命令启动它，并在计算机的菜单栏中选择“查看”/“程序员”命令，然后可以输入要转换的指定进制的数值，再单击要转换的目标数制即可。如图 3.7 所示是十六进制的 F1B0 转换成二进制的结果。

图 3.7　利用计算机转换数制

3.4.2　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，也可以是负数。整型常量在 Java 程序中有 3 种表示形式，分别为十进制、八进制和十六进制。

十进制是 Java 语言默认的数制，所以使用正常的表示形式，例如：

120、0、-127、340

注意：不能以 0 作为十进制数的开头（0 除外）。

八进制的表现形式是以最高位的数字 0 做区分，如果最高位不是 0，那么 Java 将把它解释为十进制，

否则解释为八进制，例如：

八进制：0123	0400	0256
十进制：83	256	174

注意：八进制必须以 0 开头。

十六进制的表现形式是在数值最高位添加 0x 标识，其中 x 字符不区分大小写，例如：

十六进制：	0x25	0X63	0xDF1
十进制：	37	99	3569

注意：十六进制必须以 0X 或 0x 开头。

整型变量根据它在内存中所占大小的不同，可分为 byte、short、int 和 long 4 种类型，它们具有不同的取值范围，如表 3.3 所示。

表 3.3 整型数据类型

数据类型	存储位数	取值范围
byte	8 位	-2^7 至 2^7-1 即-128～127
short	16 位	-2^{15} 至 $2^{15}-1$ 即-32768～32767
int	32 位	-2^{31} 至 $2^{31}-1$ 即-2147483648～2147483647
long	64 位	-2^{63} 至 $2^{63}-1$ 即-9223372036854775808～9223372036854775807

在为这 4 种整数类型变量赋值时，要注意变量能够接受的最大值与最小值，否则会出现错误。Java 默认的整数类型是 int，如果要标记整数是 long 类型，必须在数字末尾添加字母 L，如 long num = 2147483650L。

说明：在数字末尾添加 L，标识 long 类型整数的情况一般都发生在整数大于 int 最大取值范围时，因为 Java 默认的整数类型是 int。

例 3.03 在项目中创建类 Integer，在主方法中创建不同数值型变量，并将这些变量相加，将和输出。代码如下：（**实例位置：光盘\mr\ym\3.03**）

```
public class TestInteger {                                  // 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 = 4678945187338777L;                    // 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("结果为："+result);               // 将以上变量相加的结果输出
    }
}
```

程序运行结果如图 3.8 所示。

图 3.8 实例运行结果

3.4.3　浮点类型

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 3.4 所示。

表 3.4　浮点型数据类型

数 据 类 型	内存空间（8 位等于 1 字节）	取 值 范 围
float	32 位	1.4E-45～3.4028235E38
double	64 位	4.9E-324～1.7976931348623157E308

Java 语言默认使用 double 双精度类型的浮点数，就像 int 是默认的整数类型一样。如果将实数赋值给 float 类型的变量，则需要在实数末尾添加 F 字母作为后缀，否则被认为是 double 类型数值。

例如：

```
double sum=3799.39;              // 双精度是默认浮点类型
double length=156.34D;           // 双精度可以添加 D 标识
float price=34.50F;              // 单精度需要添加 F 标识
```

可以使用后缀 d 或 D 来明确表明这是一个 double 类型数据，但加不加 d 没有硬性规定，可以加也可以不加。而声明 float 型变量时如果不加 F，编译器会认为是 double 类型而出错。

3.4.4　字符类型

（1）char 型

字符类型（char）用于存储单个字符，占用 16 位（两个字节）的内存空间。字符类型的字面量以单引号表示，如‘s’表示一个字符。而“s”则表示一个字符串，虽然其只有一个字符，但由于使用双引号，所以它仍然表示字符串，而不是字符。

使用 char 关键字可定义字符变量，例如：

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

字符类型与整数类型可以互相赋值，也就是可以把整数赋值给字符变量，也可以把字符变量赋值给整数类型变量。由于字符 a 在 unicode 表中的编码是 97，因此允许将上面的语句写成：

```
char x = 97;
```

同 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 表中相应编码的字符，必须使用 char 型显示转化。

例 3.04　在项目中创建类 CharAndInt，在主方法中编写代码，演示字符类型与整数类型的通用赋值方式。关键代码如下：（**实例位置：光盘\mr\ym\3.04**）

```
public class CharAndInt {                                          // 定义类 Example
    public static void main(String[] args) {                       // 主方法
        int word = 'd', word2 = '@';                               // 定义 int 型变量
        char p = 39033, p2 = 30446;                                // 定义 char 型变量
        System.out.println("d 在 unicode 表中的编码是：" + word);
        System.out.println("@在 unicode 表中的编码是：" +  word2);
        System.out.println("unicode 表中编码 63045 是字符：" +  p);
```

```
        System.out.println("unicode 表中编码 55213 是字符：" +  p2);
    }
}
```

运行结果如图 3.9 所示。

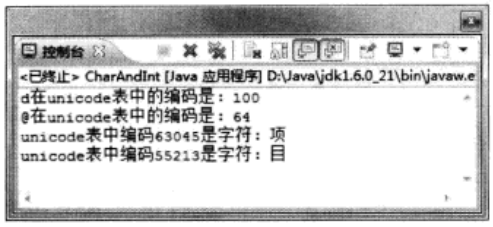


图 3.9　实例运行结果

（2）转义字符

转义字符是一种特殊的字符变量。转义字符以反斜线“\”开头，后跟一个或多个字符。转义字符具有特定的含义，不同于字符原有的意义，故称“转义”。例如，“\n”就是一个转义字符，意思是“回车换行”。Java 中转义字符如表 3.5 所示。

表 3.5　转义字符表

转 义 字 符	含　义
\ddd	1～3 位八进制数所表示的 unicode 字符，如\102 是字符 B
\uXXXX	4 位十六进制数所表示的 unicode 字符，如\u0042 是字符 B
\’	单引号字符
\\	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

将转义字符赋值给字符变量时，与字符常量值一样需要使用单引号。

例 3.05　在类的主方法中使用转义字符完成控制台字符串的输出，主方法的关键代码如下：（**实例位置：光盘\mr\ym\3.05**）

```
public class Temp {
    public static void main(String[] args) {
        System.out.println("\102");                    // 八进制编码的字符
        System.out.println("\u0042");                  // 十六进制编码的字符
        System.out.println("苹果\t 橘子\t 香蕉");       // 使用制表符分割
        System.out.println("\\字符是反斜杠");           // 输出反斜杠
        System.out.println("\n 换行字符");              // 输出回车
    }
}
```

实例运行结果如图 3.10 所示。

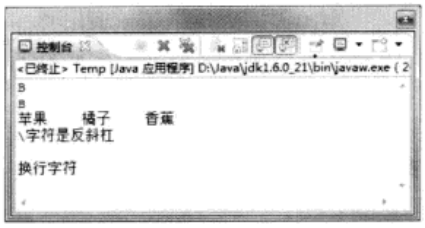


图 3.10　实例运行结果

3.4.5　布尔类型

布尔类型又称逻辑类型，只有两个值 true 和 false，分别代表布尔逻辑中的“真”和“假”，布尔值不能与整数类型进行转换。布尔类型通常被用在流程控制语句中作为判断条件。

通过关键字 boolean 定义布尔类型变量，例如：

```
boolean isAgree;                    // 定义布尔型变量 b
boolean isStart;                    // 定义布尔型变量 b1、b2
```

3.5　数据类型转换

数据类型转换就是把数据的类型转变为另一种类型。例如，可以将 long 类型数据 457 转换为一个 int 整数类型。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，不用做显示的类型转换（即自动类型转换）；反之则必须做强制类型转换。

3.5.1　自动类型转换

Java 的 8 个基本数据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，然后再进行运算。数据类型可以根据占用存储空间的大小分为高低级别，占用空间越小，级别越低，占用空间越大，级别越高。自动类型转换遵循低级到高级转换的规则，如图 3.11 所示。从左到右的数据类型不断升高，箭头指明了自动类型转换的规则。

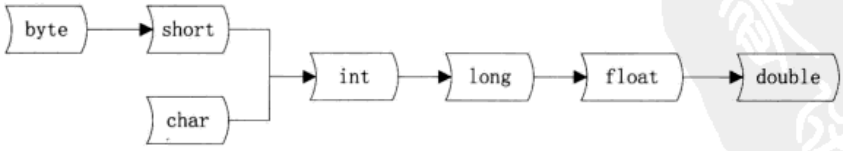


图 3.11　数据类型级别与自动转换规则

基本数据类型的自动类型转换会发生在以下几种情况。

☑　变量赋值

当为变量赋值的数据类型与变量类型不一致，并且赋值的数据类型级别低于变量类型的级别时，自动数据类型转换会将赋值数据自动转换为变量的类型。

例如：

```
byte byteNum=23;                    // 声明 byte 类型变量
int intNum=byteNum;                 // 把 byte 型数据赋值给 int 型变量
```

☑ 方法调用

把一个数值传递给方法的参数，并且这个数值的数据类型低于方法的参数变量的数据类型。

例如：

```
public static void say(int num){
    System.out.println("这是 int 类型数值："+num);
}
public static void main(String[] args) {
    byte byteNum=23;
    say(byteNum);                          // 把 byte 数值传递给方法的 int 参数
}
```

☑ 一元运算

在一元运算中（除了++和--运算符），如果操作数的类型低于 int 类型（即 byte、short 和 char 类型），则操作数会自动转换为 int 类型。

☑ 二元运算符

在二元运算中，所有低级的数据类型都会转换为运算中级别最高的数据类型。也就是说，如果有一个操作数是 double 类型，其他操作数都会自动转换为 double 类型。如果运算中最高的数据类型是 float，其他操作数都会转换为 float 类型。如果最高的数据类型为 long，那么所有操作数都会自动转换为 long 类型。如果最高的数据类型为 int，那么所有操作数都会自动转换成 int 类型，至于 byte、short、char 类型会自动转换为 int 或运算中级别最高的数据类型。

例 3.06 在项目中创建类 Conversion，在主方法中创建不同数值型的变量，实现各变量的自动类型转换。代码如下：**（实例位置：光盘\mr\ym\3.06）**

```
public class Conversion {
    public static void main(String[] args) {
        byte b1=34;
        short s1=123;
        char c1='B';
        int i1=40;
        long l1=900L;
        float f1=8.11F;
        double d1=3.1415926;
        System.out.println("double/byte+char->double\t"+(d1/b1+c1));
        System.out.println("long+short->long\t\t"+(l1+s1));
        System.out.println("char+byte+short->int\t\t"+(c1+b1+s1));
        System.out.println("float+b1/s1->float\t\t"+(f1+b1/s1));
        System.out.println("int+double->double\t\t"+(i1+d1));
    }
}
```

运行结果如图 3.12 所示。

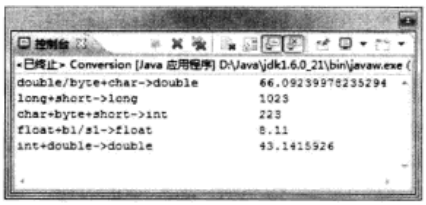


图 3.12 实例运行结果

3.5.2　强制类型转换

当把高级别（即占用存储空间较大）的数据类型赋值给低级别（即占用存储空间较小）的数据类型时，必须进行强制类型转换，其语法格式如下：

(数据类型)(表达式)

类型是要转换成的目标类型，如 int、short、double 等。表达式是要转换成的目标数据，它可以是任何基本类型的算术运算表达式。

例如：

```
long num=900L;                    // 900
short snum=(short)num;            // 900
```

long 类型的数值 900 没有超过 short 类型的取值范围，所以 short 类型的变量 snum 完全容纳了强制类型转换后的数值。

基本数据类型的强制类型转换需要注意数据完整性的问题，当把高级的数据类型强制转换为低级的数据类型时，如果数值超出低级数据类型的取值范围，数值将被截取，导致数据丢失、不完整。

例如：

```
short snum=516;                   // 516
byte bnum=(byte)snum;             // 4
```

snum 是 short 类型，数据长度是 16 位，而 bnum 是 byte 类型，长度为 8 位。当 snum 强制转换为 byte 类型时，数值 516 超出了 byte 类型的取值范围，这时只能从 short 类型截取低 8 位的数据赋值给 byte 类型的 bnum 变量，如图 3.13 所示。

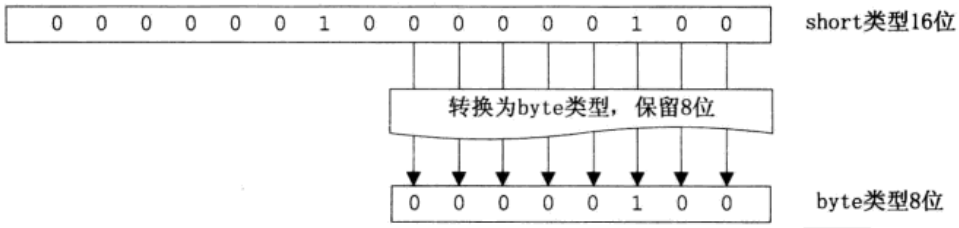


图 3.13　short 转换为 byte 时必须截取数据

这样就造成了数据丢失，所以在使用强制数据类型转换时，一定要倍加小心。

注意：boolean 布尔型的数值不能被牵制类型转换为其他数据类型，反之亦然。

例 3.07　将 long 类型数值强制类型转换为其他数据类型，实例代码如下：（**实例位置：光盘\mr\ym\3.07**）

```
public class PConversion {
    public static void main(String[] args) {
        long num=33333;
        System.out.println("原数是："+num);
        System.out.println("转成 int："+(int)num);
        System.out.println("转成 short："+(short)num);
        System.out.println("转成 byte："+(byte)num);
        System.out.println("转成 float："+(float)num);
        System.out.println("转成 char："+(char)num);
    }
}
```

实例运行结果如图 3.14 所示。

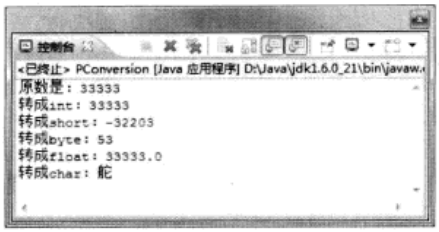


图 3.14　实例运行结果

3.6　运　算　符

代表不同运算操作的符号称为运算符，执行运算的数据称为操作数。操作数与操作符按一定语法形式组成有意义的符号序列称为表达式。例如，128+64 是一个表达式，其中变量名是最简单的表达式，它的值就是表达式的结果。表达式的结果也可以作为其他表达式的操作数，形成更加复杂的表达式。

3.6.1　赋值运算符

赋值运算符是以符号“=”表示，它主要用于修改变量的内容，其语法格式如下：

变量 = 表达式;

其功能是将“=”右侧的表达式的值赋值给左侧的变量，例如：

```
int num = 10-5+4;
```

该赋值运算将“=”右侧表达式 10-5+4 的结果 9 赋值给变量 num。如果在“=”前面加上其他运算符，就构成了复合赋值运算符。例如，num += 1 等价于 num = num + 1，复合赋值运算符集成了计算与赋值功能。

例如，使用赋值运算符为变量赋值，实例代码如下：

```
int a = 10;                          // 声明 int 型变量 a
int b = 5;                           // 声明 int 型变量 b
int c = a+b                          // 将变量 a 与 b 进行运算后的结果赋值给 c
b += a                               // 把变量 b 与 a 进行加法运算，然后结果赋值给 b
```

遵循赋值运算符的运算规则，可知变量 c 的赋值将先计算 a+b 的值，结果为 15，然后将 15 赋值给变量 c，因此 c=15。

由于赋值运算符“=”处理时会先取得右方表达式处理后的结果，因此一个表达式中若含有两个以上的“=”运算符时，会从最右方“=”开始处理。

例 3.08　在项目中创建类 Eval，在主方法中定义变量，使用赋值运算符为变量赋值。实例代码如下：（**实例位置：光盘\mr\ym\3.08**）

```
public class Eval {                                  // 创建类
    public static void main(String[] args) {         // 主方法
        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 的和赋值给变量 b，然后再赋值给变量 c
        a += 1;                                      // 将变量 a 的值加 1，然后再赋值给变量 a
        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
```

```
        System.out.println("a 值为：" + a);                    // 将变量 a 的值输出
    }
}
```

运行结果如图 3.15 所示。

图 3.15　实例运行结果

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 都得到同样的值 5，但是在程序开发中不建议使用这种赋值语法。

3.6.2　算术运算符

Java 中的算术运算符主要有+（加号）、--（减号）、*（乘号）、/（除号）、%（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 3.6 所示。

表 3.6　Java 中的算术运算符

运　算　符	说　　明	实　　例	结　　果
+	加或正值	10+50	60
-	减或负值	100-60	40
*	乘	4*10	40
/	除	10/2	5
%	取余数	12%5	2

其中“+”和“-”运算符还可以作为数据的正负符号，如+5、-7。

注意：进行除法运算时，要记住 0 不可以做除数，如 int a = 5/0;系统会抛出异常信息 java.lang.ArithmeticException:/by zero。

下面通过一个实例来介绍使用算术运算符的方法。

例 3.09　项目中创建类 Arith，在主方法中定义变量，使用算术运算符将变量的计算结果输出。实例代码如下：（**实例位置：光盘\mr\ym\3.09**）

```
public class Arith {                                          // 创建类
    public static void main(String[] args) {                  // 主方法
        float number1 = 75.14f;                               // 声明 float 型变量并赋值
        int number2 = -15;                                    // 声明 int 型变量并赋值
        System.out.println("和为：" + (number1 + number2));     // 将变量相加之和输出
        System.out.println("差为：" + (number2 - number1));     // 将变量相减之差输出
        System.out.println("积为：" + (number1 * number2));     // 将变量相乘的积输出
        System.out.println("商为：" + (number1 / number2));     // 将变量相除的商输出
    }
}
```

运行结果如图 3.16 所示。

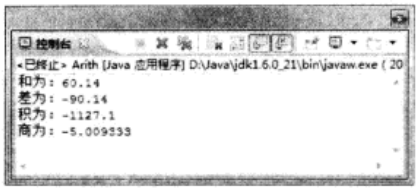


图 3.16　实例运行结果

3.6.3　自增和自减运算符

自增、自减运算符是一元算术运算符，可以作为前缀，也可以作为后缀使用。作用是使变量的值增 1 或减 1。放在变量前面的自增、自减运算符，如++x，会先将变量 x 的值加 1，然后再引用该变量的值参与运算。放在操作符后面的自增、自减运算符，如 x++，是先引用变量的值参与表达式的运算，然后再将该变量加 1。

例 3.10　输出自增与自减操作符作为前缀与后缀的表达式结果，仔细分析实例运行结果。实例代码如下：（**实例位置：光盘\mr\ym\3.10**）

```
public class Autoincrement {
    public static void main(String[] args) {
        int num = 10;
        System.out.println("num = " +          num);
        System.out.println("num++ = " +        num++);       // 使用 num 之后，再使 num 的值加 1
        System.out.println("num-- = " +        num--);       // 使用 num 之后，再使 num 的值减 1
        System.out.println("++num = " +        ++num);       // 使用 num 之前，先使 num 的值加 1
        System.out.println("--num = " +        --num);       // 使用 num 之前，先使 num 的值减 1
    }
}
```

运行结果如图 3.17 所示。

图 3.17　实例运行结果

粗略地分析，++num 与 num++的作用都相当于 num = num+1。假设 num = 4，则 b = ++num；先将 num 的值加 1，然后赋给 b，此时 num 值为 5，b 值为 5。b = num++；先将 num 的值赋给 b，再将 num 的值变为 5，此时 num 值为 5，b 值为 4。

3.6.4　比较运算符

比较运算符也称关系运算符，属于二元运算符，其运算结果是 boolean 布尔类型。当运算符对应的两个操作数关系成立时，运算结果是 true，否则是 false。所有比较运算符通常用在条件语句中作为判断的依据，

比较运算符共有 6 个，如表 3.7 所示。

表 3.7　比较运算符

运　算　符	使 用 格 式	说　　明	举　　例
>	x > y	如果 x 大于 y，结果为 true，否则为 false	10>7 结果为 true
<	x < y	如果 x 小于 y，结果为 true，否则为 false	10<7 结果为 false
==	x == y	如果 x 等于 y，结果为 true，否则为 false	10==7 结果为 false
>=	x >= y	如果 x 大于或者等于 y，结果为 true，否则为 false	10>=7 结果为 true
<=	x <= y	如果 x 小于或者等于 y，结果为 true，否则为 false	10<=10 结果为 true
!=	x != y	如果 x 不等于 y，结果为 true，否则为 false	10!=7 结果为 true

注意：相等运算符“==”和赋值运算符“=”要严格区分，不要混淆。如果在条件判断语句中把“==”写成“=”，会导致错误发生，这是初学者最容易犯错的地方。可以把常量放置在“==”的左面，右面放变量或其他，如 if（3 == x）是一些程序开发人员的惯用写法，这样做，当“==”错写为“=”时编译器会报告错误。

例 3.11　在项目中创建类 Compare，在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。实例代码如下：（**实例位置：光盘\mr\ym\3.11**）

```
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 的返回值为：" + (number1 == number2));
        System.out.println("4 != 5 的返回值为：" + (number1 != number2));
        System.out.println("4 < 5  的返回值为：" + (number1 < number2));
        System.out.println("4 > 5  的返回值为：" + (number1 > number2));
        System.out.println("4 >= 5 的返回值为：" + (number1 >= number2));
        System.out.println("4 <= 5 的返回值为：" + (number1 <= number2));
    }
}
```

运行结果如图 3.18 所示。

图 3.18　实例运行结果

3.6.5　逻辑运算符

和比较运算符相比，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或布尔值进行条件判断。而比较运算符只能进行一个简单的条件判断，如 x>y。

Java 中的逻辑运算符包括逻辑与“&&”、逻辑或“||”、逻辑非“!”，逻辑运算符的操作数必须是 boolean 型表达式。在逻辑运算符中，除了“!”是一元运算符之外，其他都是二元运算符，表 3.8 按照逻辑运算符的优先级给出了它们的用法和含义。

表 3.8 逻辑运算符

运 算 符	用 法	含 义	结 合 方 向
!（逻辑非）	! op1	op1 为 true，则结果为 false，否则为 true	右到左
&&（逻辑与）	op1 && op2	op1 和 op2 都为 true，则结果为 true	左到右
||（逻辑或））	op1 || op2	op1 和 op2 都为 false，则结果为 false	左到右

参与逻辑运算的对象必须是布尔类型的表达式或变量，经逻辑运算后的结果也是 boolean 类型的数据，使用逻辑运算符进行逻辑运算如表 3.9 所示。

表 3.9 使用逻辑运算符进行逻辑运算

x	y	x && y	x || y	! x
true	true	true	true	false
true	false	false	true	false
false	false	false	false	true
false	true	false	true	true

例 3.12 输出与、或、非逻辑关系的运算结果，实例代码如下：**（实例位置：光盘\mr\ym\3.12）**

```
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 result = ((a > b) && (a != b));               // 逻辑与&&运算
        boolean result2 = ((a > b) || (a != b));              // 逻辑或||运算
        System.out.println("(2 > 5) && (2 != 5) = "+result);  // 变量 result 输出
        System.out.println("(2 > 5) || (2 != 5) = "+result2); // 变量 result2 输出
        System.out.println("!(5 > 2) = " +!(5 > 2));          // 逻辑非！运算
    }
}
```

运行结果如图 3.19 所示。

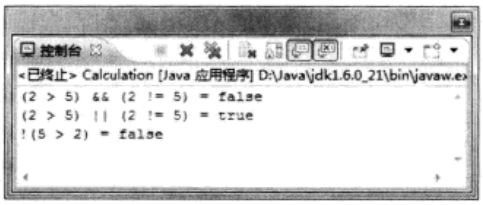


图 3.19 实例运行结果

3.6.6 位运算符

位运算符用于处理整型和字符型的操作数。位运算是完全针对二进制位（bit）单位的操作，Java 语言提供的位运算符如表 3.10 所示。

表 3.10　位运算符

运　算　符	用　　法	含　　义	运 算 分 类
～（非）	～op1	对 op1 操作数按位取反	按位运算
&（与）	op1 & op2	op1 和 op2 按位相与	
|（或）	op1 | op2	op1 和 op2 按位相或	
^（异或）	op1 ^ op2	op1 和 op2 按位异或	
<<（左移位）	op1 << op2	op1 按位左移 op2 次	移位运算符
>>（右移位）	op1 >> op2	op2 按位右移 op2 次	
>>>（无符号右移位）	op1 >>> op2	op2 不带符号按位右移 op2 次	

📖 **说明：Java 的位运算符中，非运算“～”是一元运算符，除此之外的都是二元运算符。**

1. “按位与”运算

“按位与”运算的运算符为“&”，“&”是二元运算符。“按位与”运算的运算法则是：如果两个整型数据 a、b 对应位都是 1，则该位结果才是 1，否则为 0，即

0&0=0　　0&1=0　　1&0=0　　1&1=1

例如，153 的二进制是 10011001，46 的二进制是 00101110，这两个数按位与运算的方法如下，括号中的是十进制数。

```
    10011001 (153)
&   00101110 (46)
---------------------
    00001000 (8)
```

2. “按位或”运算

“按位或”运算的运算符为“|”，它是二元运算符。“按位或”运算的运算法则是：如果两个操作数对应位都是 0，则该位的结果才是 0，否则为 1，即

0|0=0　　0|1=1　　1|0=1　　1|1=1

例如，153 的二进制是 10011001，46 的二进制是 00101110，这两个数按位进行或运算的方法如下，括号中的是十进制数。

```
    10011001 (153)
|   00101110 (46)
---------------------
    10111111 (191)
```

3. “按位取反”运算

“按位取反”运算也称“按位非”运算，运算符为“～”，它为一元运算符。它的运算法则是：操作数二进制中的 1 修改为 0，0 修改为 1，即

～0=1　　～1=0

例如，153 的二进制是 10011001，把它按位取反的结果为 01100110。

4. “按位异或”运算

“按位异或”运算的运算符为“^”，它为二元运算符。“按位异或”运算的运算法则是：当两个操作数的二进制对应位相同（同时为 0 或同时为 1）时，结果为 0，否则为 1，即

```
0^0=0        0^1=1        1^0=1        1^1=0
```

例如，153 的二进制是 10011001，46 的二进制是 00101110，这两个数按位进行异或运算的方法如下，括号中的是十进制数。

```
    10011001 (153)
^   00101110 (46)
--------------------------------
    10110111 (183)
```

5. 位移运算

除了上面介绍的按位运算符外，还可以对数据按二进制位进行移位操作，Java 中的移位运算符有以下 3 种：

☑ <<左移。

☑ >>右移。

☑ >>>无符号右移。

左移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。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，如图 3.20 所示。

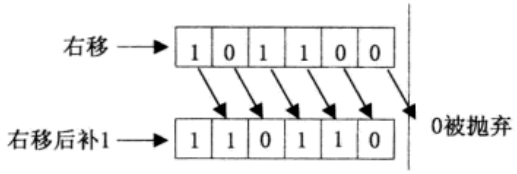


图 3.20 右移

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

注意：移位运算符适用的数据类型有 byte、short、char、int、long。

技巧：移位能实现整数除以或乘以 2 的 n 次方的效果，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。

3.6.7 三元运算符

三元运算符使用格式为：

条件式? 值 1:值 2

三元运算符的运算规则为：若条件式的值为 true，则整个表达式取值 1，否则取值 2。例如：

```
int x = 1;
int y = 2;
int num = x < y ? 10 : 20;
```

如上例所示，当表达式“x < y”的运算结果返回 true 时，则变量 num 取值 10，当表达式“x < y”返回 false 时，则变量 num 取值 20，上例的结果为 num 等于 10。

例 3.13 使用三元运算符求两个数中的最小数。实例代码如下：**（实例位置：光盘\mr\ym\3.13）**

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

```
        int x=32;                    // 声明变量 x
        int y=23;                    // 声明变量 y
        int min=x<y?x:y;             // 声明变量 min 并赋值 x 与 y 的最小数
        System.out.println("两数中最小的值为："+min);
    }
}
```

运行结果如图 3.21 所示。

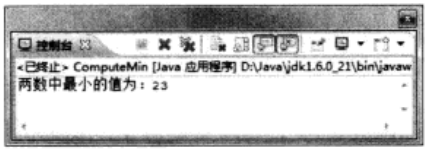


图 3.21　实例运行结果

3.6.8　运算符优先级

Java 中运算符的优先级决定了表达式中运算执行的先后顺序。如果两个运算符有相同的优先级，那么左边的表达式要比右边的表达式先被处理，表 3.11 显示了在 Java 中众多运算符的特定的优先级。

表 3.11　运算符的优先级别

优　先　级	描　　述	运　算　符
高	括号	()
●	正负号	+ -
↓	一元运算符	++ -- !
↓	乘除	* / %
↓	加减	+ -
↓	移位运算	<< >> >>>
↓	比较大小	< > >= <=
↓	比较是否相等	== !=
↓	按位与运算	&
↓	按位异或运算	^
↓	按位或运算	|
↓	逻辑与运算	&&
↓	逻辑或运算	||
↓	三元运算符	? :
低	赋值运算符	= += -= *= /= %=

技巧：在编写程序时尽量使用括号()运算符号来限定运算次序，以免产生错误的运算顺序。

3.7　编 码 规 范

编码规范对于程序开发人员很重要，遵循编码规范所编写的程序代码可读性高，有维护价值。如果编

写的程序代码别人看不懂，甚至没有人愿意看，那么所编写的程序代码就一文不值，而且再过一段时间，该程序的编写人员也会忘记程序的业务逻辑，无法有效地维护程序代码。

读者在学习开发的过程中一定要养成良好的编码规范，对于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，本节对编码规则做了简单总结，供读者学习。

☑ 每条语句要单独占一行，一条命令要以分号结束。

注意：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“；”，这时编译器会报出 illegal character（非法字符）这样的错误信息。

☑ 在声明变量时，尽量使每个变量的声明单独占一行，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，对于局部变量应在声明的同时对其进行初始化。

☑ 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[])
```

而多余的空格没有任何意义，为了便于理解和阅读，应控制好空格的数量。

☑ 变量的命名要尽量使用有意义的名称，变量的第一个单词小写，第二个以后的单词要求首字母大写，如 productPrice、age、name、userName。

☑ 类名和接口的命名不但要求使用有意义的单词，而且所有单词的首字母都要大写，如 ProductFactory 和 UserInfo。

☑ 常量的所有字母全部大写，多个单词使用下划线“_”分割，如 PRODUCT_UNIT。

☑ 适当的缩进代码，使代码对齐。如果使用 Eclipse 开发工具，按下 Shift+Ctrl+F 快捷键将自动格式化程序代码。

☑ 为了方便日后的维护，不要使用技术性很高、难懂易混淆判断的语句。由于不允许程序的开发与维护是同一个人，所以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，这样有助于阅读者很快地了解代码结构。

3.8 照猫画虎——基本功训练

3.8.1 基本功训练 1——合理设定变量类型

视频讲解：光盘\mr\lx\03\合理设定变量类型.exe

实例位置：光盘\mr\ym\3.14\src

变量是 Java 程序开发中的重要数据单元，必须熟练掌握其用途与定义方法。不同用途的变量要使用不同的数据类型严格区分，以方便其参与标准的运算。下面请读者完成相关训练，在主方法中分别定义代表商品优良等级、商品单价、销售总数、是否有库存的变量，并在控制台中输出变量内容，实例代码如下：

```
public class PrintVariable {
    public static void main(String[] args) {
        char level = '优';                    // 商品等级
        float price = 34.55f;                 // 商品单价
        int amount = 38900;                   // 销售总数
        boolean haveProduct = true;           // 是否有库存
        System.out.println("我公司商品品质为：" + level);
```

```
            System.out.println("其单价为：" + price);
            System.out.println("目前为止已经销售：" + amount + "套");
            System.out.println("现在是否有库存：" + haveProduct);
        }
    }
```

实例运行结果如图 3.22 所示。

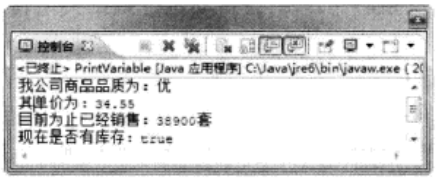


图 3.22　实例运行结果

照猫画虎：在实例中添加一个目前销售总额的变量，注意变量类型的合理性。（15 分）（实例位置：光盘\mr\ym\3.14\zmhh）

3.8.2　基本功训练 2——变量自增运算

视频讲解：光盘\mr\lx\03\变量自增运算.exe

实例位置：光盘\mr\ym\3.15\src

对于 Java 中数据的各种运算，以数字运算的使用最为广泛。有意思的是，整数的自增运算符可以让变量的值累加 1，这和其他运算有所区别，读者应该熟练掌握。下面请读者完成相关训练，定义一个整数变量，然后通过变量的后置自增运算为引用该变量的语句提供数值，实例代码如下：

```
public class NumberIncrease {
    public static void main(String[] args) {
        int count = 1;                                    // 自增次数
        System.out.println("第" + count + "次自增");
        count++;                                          // 变量自增
        System.out.println("第" + count++ + "次自增");
        System.out.println("第" + count++ + "次自增");
        System.out.println("第" + count++ + "次自增");
    }
}
```

实例运行结果如图 3.23 所示。

图 3.23　实例运行结果

照猫画虎：尝试改变实例，使用前置递增运算符，查看程序运算结果的改变。（15 分）（实例位置：光盘\mr\ym\3.15\zmhh）

3.8.3 基本功训练 3——类型转换

视频讲解：光盘\mr\lx\03\类型转换.exe

实例位置：光盘\mr\ym\3.16\src

将变量赋值给不同类型的其他变量时，Java 会自动进行数据类型转换。另外，程序开发人员也可以把变量强制类型转换为另一个类型。下面请读者完成相关训练，创建 int、float、double 等常用基本类型的变量并简单地实现这些变量的数据类型转换，实例代码如下：

```
public class TypeConvertion {
    public static void main(String[] args) {
        int intNum = 4;
        float floatNum = 9.5F;
        floatNum /= intNum;                    // 自动类型转换为 float
        System.out.println("floatNum = " + floatNum);
        double numX = 4.88;
        double numY = 78.83;
        int numZ = (int) numX + (int) numY;    // 强制类型转换 double 为 int
        System.out.println("numZ = " + numZ);
    }
}
```

运行结果如图 3.24 所示。

图 3.24 实例运行结果

照猫画虎：尝试把 double 类型转换为 long 类型数据并参加求和运算。（15 分）（实例位置：光盘\mr\ym\3.16\zmhh）

3.8.4 基本功训练 4——运算符的应用

视频讲解：光盘\mr\lx\03\运算符的应用.exe

实例位置：光盘\mr\ym\3.17\src

数据运算是程序开发的必要环节，而运算的体现方式需要通过运算符来表达，所以熟练掌握各种运算符是基础知识的重点。下面请读者完成相关训练，声明 int 类型和 double 类型的两个变量，然后在控制台中输出这两个变量的不同运算表达式的结果，关键代码如下：

```
int intNum = 1;                                // 声明 int 类型变量
double doubleNum = 1.0;                        // 声明 double 类型变量
intNum = 46 / 9;                               // 计算 int 类型公式
System.out.println("intNum = " + intNum);
intNum = 46 % 9 + 4 * 4 - 2;                   // 计算 int 类型公式
System.out.println("intNum = " + intNum);
intNum = 1;
intNum %= 3 / intNum + 3;                      // 计算 int 类型公式
```

```
System.out.println("intNum = " + intNum);
doubleNum = 4 + doubleNum * doubleNum + 4;        // 计算 double 类型公式
System.out.println("doubleNum = " + doubleNum);
doubleNum += 1.6 * 3 + (++intNum);                // 计算 double 类型公式
System.out.println("doubleNum = " + doubleNum);
intNum = 1;
doubleNum -= 1.6 * 3 + intNum++;                  // 计算 double 类型公式
System.out.println("doubleNum = " + doubleNum);
```

运行结果如图 3.25 所示。

图 3.25　实例运行结果

照猫画虎：尝试编程计算 10 以内所有奇数的和以及乘积，并输出到控制台。（15 分）（实例位置：光盘\mr\ym\3.17\zmhh）

3.8.5　基本功训练 5——求最大值

视频讲解：光盘\mr\lx\03\求最大值.exe

实例位置：光盘\mr\ym\3.18\src

三元运算符是 Java 语言中一个非常特殊的运算符，它可以通过条件表达式获取不同的运算结果。下面使用三元运算符判断并输出两个数值中最大的一个，实例代码如下：

```
public class PringMaxNum {
    public static void main(String[] args) {
        int num1 = 34;                                  // 整数 1
        int num2 = 89;                                  // 整数 2
        int maxNum = num1 > num2 ? num1 : num2;         // 求最大值
        System.out.println("数字 1："+num1);
        System.out.println("数字 2："+num2);
        System.out.println("两数中最大者是："+maxNum);
    }
}
```

实例运行结果如图 3.26 所示。

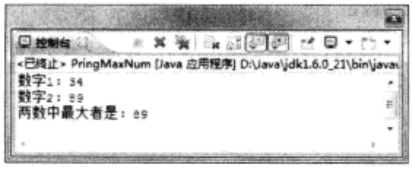


图 3.26　实例运行结果

照猫画虎：尝试使用三元运算符判断两个数字是否相等，运算结果是字符串，将其输出到控制台中。（20

分）（实例位置：光盘\mr\ym\3.18\zmhh）

3.8.6 基本功训练 6——位移实现数字乘以 2 的 8 次幂

视频讲解：光盘\mr\lx\03\位移实现数字乘以 2 的 8 次幂.exe

实例位置：光盘\mr\ym\3.19\src

Java 的位运算有操作二进制数据的能力，其中控制二进制数位左移与右移操作是相对复杂且高级的，它在一些特殊运算中起关键作用。下面通过位移操作实现特殊运算，定义一个整数变量，然后通过左移操作实现该变量乘以 2 的 8 次幂操作，实例代码如下：

```
public class ANumber {
    public static void main(String[] args) {
        int number = 16;                                          // 定义整数变量
        int result = number << 8;                                 // 位移运算
        System.out.println(number + "乘以 2 的 8 次幂等于=" + result);    // 输出运算结果
    }
}
```

实例运行结果如图 3.27 所示。

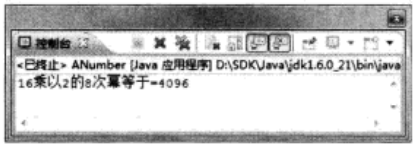


图 3.27 实例运行结果

照猫画虎：通过位运算实现数字除以 2 的 4 次幂。（20 分）（实例位置：光盘\mr\ym\3.19\zmhh）

照猫画虎栏目分数统计：

照猫画虎题目	1	2	3	4	5	6	总分数	
分数								

3.9 情景应用——拓展与实践

3.9.1 情景应用 1——输出长方形面积

视频讲解：光盘\mr\lx\03\输出长方形面积.exe

实例位置：光盘\mr\ym\3.20\src

实现一个求长方形面积的程序，其运行结果如图 3.27 所示。实现该程序时，主要用到了 Java 基础知识中的运算符，实现求长方形面积的公式为：

长方形面积 = 长×宽

实例运行结果如图 3.28 所示。

图 3.28　实例运行结果

实例代码如下：

```
public class PrintRectangleArea {
    public static void main(String[] args) {
        double width = 38.9;                                    // 宽度
        double height = 27.2;                                   // 高度
        double area = width * height;                           // 面积
        System.out.println("宽：" + width);
        System.out.println("高：" + height);
        System.out.println("长方形面积：" + area + "平方");
    }
}
```

DIY：编程计算圆形面积。要求：把 π 值定义为常量 PI 参与运算。（15 分）（实例位置：光盘\mr\ym\3.20\diy）

3.9.2　情景应用 2——输出字节 bit 位的值

视频讲解：光盘\mr\lx\03\输出字节 bit 位的值.exe

实例位置：光盘\mr\ym\3.21\src

一个字节由 8 位（bit）组成，位是最小单位，字节的位从低到高依次排序，索引下标是 0～7，如图 3.29 所示。

下面的实例通过位移运算计算二进制每个数位的值，并输出到控制台中，实例运行结果如图 3.30 所示。

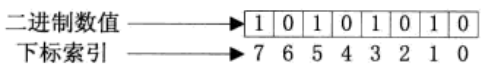


图 3.29　位的排列

图 3.30　实例运行结果

实例代码如下：

```
public class PrintBitValue {
    public static void main(String[] args) {
        byte value = 1;             // 创建一个 byte 字节类型的变量
        int bit1=value;
        int bit2=value<<1;          // 通过位移运算获取每个二进制位的值
        int bit3=value<<2;
        int bit4=value<<3;
```

```
        int bit5=value<<4;
        int bit6=value<<5;
        int bit7=value<<6;
        int bit8=value<<7;
        // 输出每个二进制索引的值
        System.out.println("索引为 0 的值是："+bit1);
        System.out.println("索引为 1 的值是："+bit2);
        System.out.println("索引为 2 的值是："+bit3);
        System.out.println("索引为 3 的值是："+bit4);
        System.out.println("索引为 4 的值是："+bit5);
        System.out.println("索引为 5 的值是："+bit6);
        System.out.println("索引为 6 的值是："+bit7);
        System.out.println("索引为 7 的值是："+bit8);
    }
}
```

DIY：二进制位右移。提示：把一个字节的 8 个二进制位全部设置为 1，即 0xFF，然后通过右移操作计算每次右移 1 位的结果，并输出到控制台。（15 分）（实例位置：光盘\mr\ym\3.21\diy）

3.9.3 情景应用 3——判断奇偶数

视频讲解：光盘\mr\lx\03\判断奇偶数.exe

实例位置：光盘\mr\ym\3.22\src

三元运算符是 if…else…条件语句的简写格式，它可以完成简单的条件判断，下面利用这个三元运算符实现奇偶数的判断，程序运行结果如图 3.31 所示。程序接收用户输入的整数数字，然后判断是奇数还是偶数，并将判断结果输出到控制台。

图 3.31 实例运行结果

实例代码如下：

```
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) ? "您输入的数字是：偶数" : "这个数字是：奇数";
        System.out.println(check);
    }
}
```

DIY：判断数字是否为 5 的倍数。提示：能被 5 整除的数的余数都是 0。（20 分）（实例位置：光盘\mr\ym\3.22\diy）

3.9.4　情景应用 4——简单数据加密

视频讲解：光盘\mr\lx\03\简单数据加密.exe

实例位置：光盘\mr\ym\3.23\src

通过位运算的异或运算符“^”把字符串与一个指定的值进行异或运算，从而改变字符串每个字符的值，这样就可以得到一个加密后的字符串，如图 3.32 所示。当把加密后的字符串作为程序输入内容时，异或运算会把加密后的字符串还原为原有字符串的值，如图 3.33 所示。

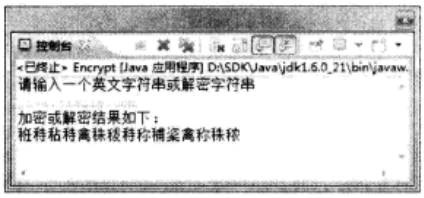


图 3.32　加密效果

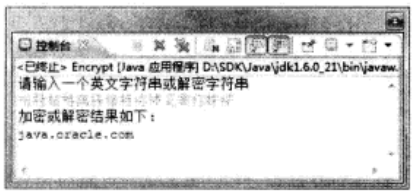


图 3.33　解密效果

实现过程如下：

```
import java.util.Scanner;
public class Encrypt {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        System.out.println("请输入一个英文字符串或解密字符串");
        String password = scan.nextLine();                    // 获取用户输入
        char[] array = password.toCharArray();                // 获取字符数组
        for (int i = 0; i < array.length; i++) {              // 遍历字符数组
            array[i] = (char) (array[i] ^ 31123);             // 对每个数组元素进行异或运算
        }
        System.out.println("加密或解密结果如下：\n" + new String(array));  // 输出密钥
    }
}
```

你问我答：toCharArray()方法有什么用途？

程序中使用了字符串的一个 toCharArray()方法，它可以把字符串拆分成字符组成的数组，而数组就是一组数据的有序集合。这在之后的章节会进行讲解，现在读者只要有所了解即可，应把注意力放到异或运算中。

DIY：信息加密。要求：把自己要传递给同学或同事的文本信息使用异或运算加密，对方接收到加密后的文本并得知你的密钥（即参与异或运算的数字）就可以看到实际内容。（20 分）（实例位置：光盘\mr\ym\3.23\diy）

3.9.5　情景应用 5——判断闰年

视频讲解：光盘\mr\lx\03\判断闰年.exe

实例位置：光盘\mr\ym\3.24\src

为了弥补因人为历法规定造成的年度天数与地球实际公转周期的时间差，设立了 366 天的闰年，闰年

的二月份有 29 天。下面使用程序完成闰年的判断，判断的年份由用户输入，程序运行结果如图 3.34 所示。

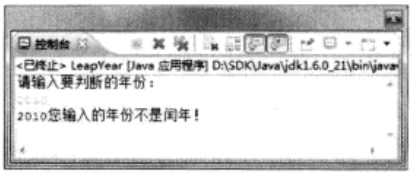


图 3.34　实例运行结果

满足两种条件的整数可以称为闰年，第一，能被 4 整除但不能被 100 整除，第二，能被 400 整除。该公式用 Java 语法实现的格式如下：

```
year % 4 == 0 && year % 100 != 0 || year % 400 == 0
```

实现过程如下：

（1）创建 Java 项目，在项目中创建 LeapYear 类。

（2）在 LeapYear 类的主方法中编程接收用户在控制台输入的数字作为年份，然后判断这个年份是否属于闰年并将其输出到控制台，实例代码如下：

```
import java.util.Scanner;
public class LeapYear {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        System.out.println("请输入要判断的年份：");
        long year = scan.nextLong();                                    // 接收用户输入的年份
        if (year % 4 == 0 && year % 100 != 0 || year % 400 == 0) {      // 是闰年
            System.out.print(year + "您输入的年份是闰年！ ");
        } else {                                                        // 不是闰年
            System.out.print(year + "您输入的年份不是闰年！ ");
        }
    }
}
```

DIY：计算 2 月份天数。要求：从控制台接收用户输入的年份，然后通过公式计算此年份中 2 月的天数。（15 分）（实例位置：光盘\mr\ym\3.24\diy）

3.9.6　情景应用 6——异或运算实现变量值交换

视频讲解：光盘\mr\lx\03\异或运算实现变量值交换.exe

实例位置：光盘\mr\ym\3.25\src

变量的互换常见于数组排序算法中，当两个数组元素需要交互时，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，那么可以使用更高效的方法，下面演示如何使用异或运算实现两个整数类型变量的高效互换，程序运行结果如图 3.35 所示。

实现过程如下：

```
public class VariableExchange {
    public static void main(String[] args) {
        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);
    }
}
```

图 3.35　实例运行结果

DIY：字符串交换。要求：把两个字符串类型的变量值进行交换，并输出交换后的变量值，验证数据是否正确。（15 分）（实例位置：光盘\mr\ym\3.25\diy）

情景应用 DIY 栏目分数统计：

DIY 题目	1	2	3	4	5	6	总分数	
分数								

3.10　自 我 测 试

一、选择题（每题 10 分，5 道题）

1．下列标识符中，哪些（个）是非法的？（　　）

A．here　　B．_there　　C．this　　D．that　　E．24lectures

2．下列转义字符中，哪些（个）是非法的？（　　）

A．\n　　B．\r　　C．\t　　D．\b　　E．\’

F．\q　　G．\s　　H．\”　　I．\\

3．下列整数表达式中，哪些（个）是非法的？

A．030　　B．24　　C．0X18　　D．ABC

4．当编译和运行下列代码时会发生什么？（　　）

```
public class Test {
    public static void main(String[] args) {
        byte bitNum = 888;
        System.out.println("num=" + bitNum);
    }
}
```

A．编译错误　　B．运行错误　　C．输出 888　　D．输出 125

5．分析下列代码，程序会输出什么结果？（　　）

```
public class Test {
    int num = 45;
    public static void main(String[] args) {
        int num = 88;
        System.out.println("num=" + num);
    }
}
```

A．45　　B．88　　C．num=45　　D．num=88

二、填空题（每题 10 分，5 道题）

1．基本数据类型 char 使用了多少位来表示？_____

2．Java 中所有成员变量和局部变量都会被默认初始化吗？_____

3．下列代码的运行结果是_____。

```
public class Test {
    public static void main(String[] args) {
        int number = 10;
        System.out.println(number++);
    }
}
```

4．%运算符用于何种运算？_____

5．&&操作符可以用于哪种基本数据类型？_____

测试分数统计：

类别	第 1 题	第 2 题	第 3 题	第 4 题	第 5 题
选择题分数					
填空题分数					
总分数					

3.11 行 动 指 南

开始日期：_____年____月____日　　结束日期：_____年____月____日

序号	内　容	行 动 指 南	
1	照猫画虎栏目	分数>75 分	优秀，可以继续下一章学习。
		75 分>分数>50 分	及格，重新做一遍照猫画虎。
	分数（　　）	分数<50 分	请使用光盘中提供的“实战能力测试系统”进行提高训练。
	情景应用栏目	分数>75 分	优秀，逻辑思维能力很不错。
		75 分>分数>50 分	及格，再练一遍情景应用。
	分数（　　）	分数<50 分	请使用光盘中提供的“实战能力测试系统”进行提高训练。
	自我测试栏目	分数>75 分	优秀，逻辑思维能力很不错。
	分数（　　）	分数<75 分	请使用光盘中提供的“实战能力测试系统”进行提高训练。
	综合评价	反复训练后，以上各项分数都在 75 分以上，方可进入下一堂课学习。	

续表

序号	内　容	行 动 指 南
2	编程能力培养：本栏目提供了一些实践题目，对于培养编程能力很有效，要做好。	（1）编程计算长方形的周长。
		（2）使用打印语句输出‘4’、‘k’、‘L’、‘&’字符的 Unicode 编码。然后再分别输出 Unicode 编码为 26126、26085、31185、25216、26446、38047、23561 的字符组成的字符串。
		（3）编写 Java 程序，完成下述算术表达式。$\frac{5}{3(ab+51)}-9(a+bc)$
		（4）在程序中定义一个整型变量，在输出语句中分别使用变量的前缀自增与后缀自增表达式作为参数，查看输出结果并分析运算过程。
3	编程习惯培养：本堂课培养读者在学习开发中记笔记的习惯，把开发中遇到的问题、总结的经验记录下来。	
4	创新能力培养：看看身边有没有可以用编程解决的问题，记录到右边的表格中。根据学习进度，尝试编写解决这些问题的小程序。	

3.12　成功可以复制——知识改变命运、科技改变生活

19 岁的李彦宏离开山西阳泉到梦想中的北大就读信息管理专业，23 岁远渡重洋赴美国布法罗纽约州立大学主攻计算机，正是北大的信息管理专业让他深谙搜索内涵，正是美国的计算机学业让他掌握了计算机工具。

在搜索引擎发展初期，李彦宏作为全球最早的研究者之一，最先创建了 ESP 技术，并将它成功地应用于 Infoseek 的搜索引擎中，另外，图像搜索引擎是他的另一项极具应用价值的技术创新。

在李彦宏 31 岁时，创建中国最大的搜索引擎公司——百度网络技术有限公司。知识改变了命运！百度公司创始人、CEO 李彦宏十分相信这句话。

在美国的 8 年人生历程中，李彦宏先后担任了道·琼斯公司高级顾问、《华尔街日报》网络版实时金融信息系统设计者、国际知名互联网企业——Infoseek 资深工程师等职务。他为道·琼斯公司设计的实时金融系统，迄今仍被广泛地应用于华尔街各大公司的网站。

1999 年年底，李彦宏携 120 万美金的风险投资回国与好友徐勇先生共同创建百度网络技术有限公司，并在短短 6 个月的时间内完成目前中国最大、最好的中文搜索引擎的开发工作。

2005 年 8 月，百度在美国纳斯达克成功上市，成为全球资本市场最受关注的上市公司之一。目前，百度也是全球跨国公司最多寻求合作的中国公司，随着百度日本公司的成立，百度加快了走向国际化的步伐。

经典语录

“活的搜索，改变生活。”

深度评价

从李彦宏的求学、工作及创业经历中给我们这样的启示，知识可以改变命运，知识可以改变生活。作为程序设计人员，要多学习，刻苦钻研，掌握更多的知识和技能，然后把学到的知识和技能应用到实际工作和生活中，从而去改变我们的工作和生活。

第 4 堂课

流程控制语句

（视频讲解：113 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。本堂课将向读者介绍 Java 语言中的流程控制语句。

学习摘要：

- Java 语言的复合语句
- if 条件语句
- switch 多条件分支语句
- if 语句的灵活运用
- while 循环语句的使用
- do…while 循环语句的使用
- while 与 do…while 语句的区别
- for 循环语句的使用

4.1 复合语句

在前面的学习中已经接触到了复合语句（也称为代码块）。它使用花括号“{}”把多条语句括起来，作为一条复合语句。花括号内可以是一条语句和多条语句，甚至没有语句都可以。而且在任何可以使用单一语句的地方都可以使用复合语句，并且在复合语句中还可以嵌套复合语句。

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。

例 4.01 在主方法中定义复合语句块，其中包含另一个复合语句块。**（实例位置：光盘\mr\ym\4.01）**

代码如下：

```
public class Compound {
    public static void main(String args[]) {
        int x = 20;
        {
            int y = 40;
            System.out.println(y);
            int z = 245;
            boolean b;
            {
                b = y > z;
                System.out.println(b);
            }
        }
        String word = "hello java";
        System.out.println(word);
    }
}
```

复合语句 // 复合语句

运行结果如图 4.1 所示。

图 4.1 实例运行结果

在使用复合语句时要注意，复合语句中定义的变量是局部变量，复合语句就是该变量的作用域。在该作用域中，某个变量被创建并能够被使用。如果在作用域外使用该变量，就会发生错误。例如，在本实例中如果在复合语句外使用变量 z、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

条件语句可根据不同的判断条件执行不同的语句，判断条件就是关系表达式，如 age>30 要求 30 岁以上的用户。条件语句包括 if 条件语句与 switch 多分支语句，本节将向读者介绍这两种条件语句的用法。

4.2.1　if 条件判断语句

if 条件语句是一个重要的编程语句，它告诉程序在某个条件成立的情况下执行某段程序，如果条件不成立，就执行另外的语句，这里说的语句通常为复合语句。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

使用简单的 if 条件语句，可判断是否执行紧跟在条件之后的复合语句。关键字 if 之后是作为条件判断的“关系表达式”，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 条件后的语句。

语法：

```
if(关系表达式){
    复合语句
}
```

关系表达式：必要参数，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与常量或者是逻辑运算符的表达式。

复合语句：可选参数，当关系表达式的值为 true 时执行语句。如“复合语句”中仅有一条语句，则可以使用单一语句。

例如：

```
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“{ }”改用单一语句并无语法错误，但为了增强程序的可读性最好使用带“{ }”的复合语句。

如果条件语句后的复合语句省略，需要在 if 语句末尾添加分号“;”，这时 if 语句不做任何操作。

例如，省略了 if 语句中的复合语句。

```
if (387 > 322 && 10 != 10);
```

另外，还可以使用空的复合语句。

```
if (387 > 322 && 10 != 10) {
}
```

简单的 if 条件语句的执行过程如图 4.2 所示。

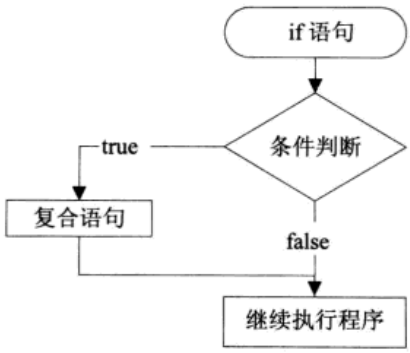


图 4.2　简单 if 条件语句的执行过程

下面以实例说明简单 if 语句的执行过程。

例 4.02 如果商品单价低于 50，就购买该商品。**（实例位置：光盘\mr\ym\4.02）**

代码如下：

```
public class Getif {                    // 创建类
    public static void main(String args[]) {
        double price = 10.50;           // 声明单价 price
        if (price < 50) {               // 如果单价小于 50
            System.out.println("购买该商品");
        }
        System.out.println("选购完毕");
    }
}
```

运行结果如图 4.3 所示。

图 4.3 实例运行结果

如果把实例中变量 price 的值改为 110.50，那么程序就不会输出“购买该商品”字符串，而直接输出“选购完毕”字符串。

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性地做出处理。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：

```
if ( 表达式 ) {
  复合语句 1
} else {
  复合语句 2
}
```

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 if 语句后的复合语句 1；如果表达式的值为 false，则执行 else 后面的复合语句 2。if…else 语句的执行过程如图 4.4 所示。

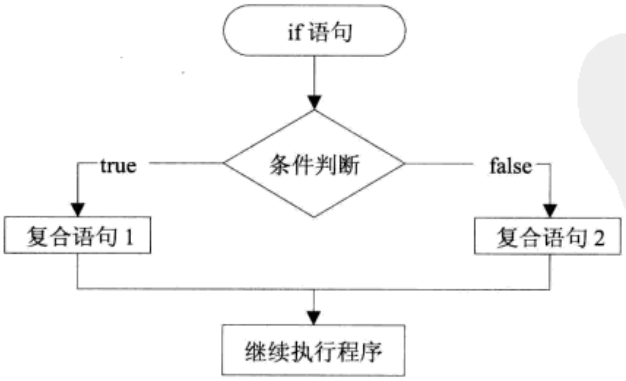


图 4.4 if…else 语句的执行过程

当复合语句省略时，代码格式与简单的 if 语句规定相同。下面以实例说明 if…else 语句的应用。

例 4.03 根据商品单价，判断是否购买。**（实例位置：光盘\mr\ym\4.03）**

代码如下：

```
public class Getifelse {                          // 创建类
    public static void main(String args[]) {
        double price = 110.50;                    // 声明单价 price
        if (price < 50) {                         // 如果单价小于 50
            System.out.println("购买该商品");
        } else {                                  // 否则
            System.out.println("太贵了，我不想买这东西。");
        }
        System.out.println("选购完毕");
    }
}
```

运行结果如图 4.5 所示。

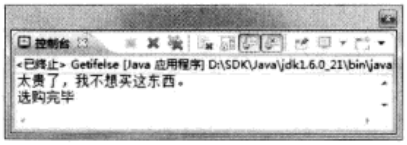


图 4.5　实例运行结果

技巧：对于 if…else 语句，可以使用三元运算符对语句进行简化，代码如下：

```
if(a > 0)
b = a;
else
b = -a;
```

可以简写成：

```
b = a > 0?a:-a;
```

上面代码为求绝对值的语句，如果 a > 0，就把 a 的值赋值给变量 b，否则将-a 赋值给变量 b。使用三元运算符的好处是可以简洁代码，并且有一个返回值。

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；如果满足另一种条件，则执行另一种处理”。

语法：

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

条件表达式 1 至条件表达式 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。

复合语句：当条件表达式 1 的值为 true 时，执行复合语句 1，当条件表达式 2 的值为 true 时，执行复合

语句 2，依次类推，if…else if 多分支语句的执行过程如图 4.6 所示。

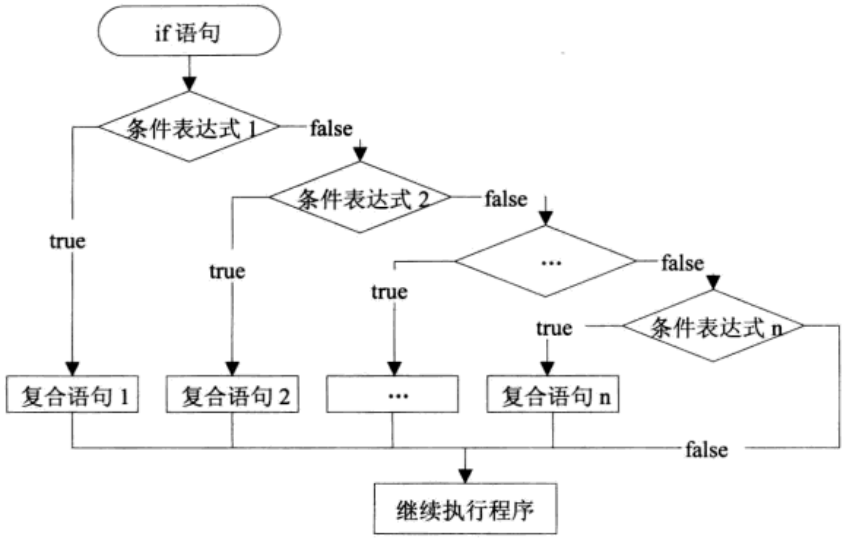


图 4.6　if…else if 多分支语句流程图

例 4.04　根据商品价格以及是否打折，做出多种判断。**（实例位置：光盘\mr\ym\4.04）**

代码如下：

```
public class GetTerm {                          // 创建类
    public static void main(String args[]) {
        boolean isRebate = true;                // 是否打折
        double price = 110.50;                  // 声明单价 price
        if (price < 50) {                       // 如果单价小于 50
            System.out.println("购买该商品");
        } else if (isRebate) {                  // 否则，如果打折
            System.out.println("既然打折就勉强买下吧");
        } else if (price > 100) {               // 否则，如果单价大于 100
            System.out.println("太贵了，我不想买这东西。");
        }
        System.out.println("选购完毕");
    }
}
```

运行结果如图 4.7 所示。

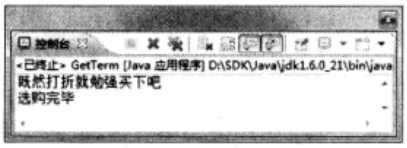


图 4.7　实例运行结果

4.2.2　if 语句的嵌套

if 语句除了多分支判断格式外，还可以嵌套使用。从本堂课前面的讲解读者应该明白，if 条件判断语句

执行的都是复合语句，笔者建议这样做，复合语句使用花括号“{ }”定义多条语句，那么也可以在其中再插入新的 if 语句，这样就形成了 if 语句的嵌套。

语法：

```
if( 条件表达式 1 ){
    …其他语句
    if( 条件表达式 2 ){
        复合语句
    }
}
```

其中使用条件表达式 2 的 if 语句嵌套在使用条件表达式 1 的 if 语句中，if 语句的嵌套使用可以实现更加复杂的逻辑判断，它很接近我们的思维方式。

例 4.05　根据商品价格以及是否打折、打折多少做出多种判断。**（实例位置：光盘\mr\ym\4.05）**

代码如下：

```
public class NestedIF {
    public static void main(String args[]) {
        boolean isRebate = true;                    // 打折状态变量
        double rebate = 8;                          // 折扣为 8 折
        double price = 110.50;                      // 声明单价 price
        if (price < 50) {                           // 如果单价小于 50
            System.out.println("购买该商品");
        } else if (isRebate) {                      // 否则，如果打折
            System.out.println("打折的话我再想想");
            if (rebate <= 7.5) {                    // 嵌套 if 语句判断折扣，如果低于 7.5 折
                System.out.println("折扣我很满意，买下了");
            } else {                                // 否则，折扣不满意
                System.out.println("打" + rebate + "折后的价格我也不想买");
            }
        } else if (price > 100) {                   // 否则，如果单价大于 100
            System.out.println("太贵了，我不想买这东西。");
        }
        System.out.println("选购完毕");
    }
}
```

运行结果如图 4.8 所示。

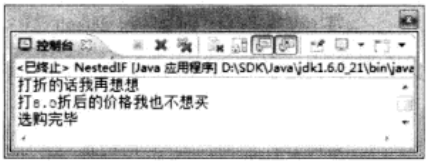


图 4.8　实例运行结果

4.2.3　switch 分支语句

if 语句和它的嵌套使用虽然能够处理复杂的多路分支判断，实现程序流程的控制，但是使用起来不太方便，编写的代码可读性不高。如果程序需要的只是使用分支语句根据不同的变量值执行不同的操作，那么，Java 提供的 switch 语句能够解决上述问题，它更加适合这种应用，而且可读性非常好。

switch 语句根据一个表达式的结果，执行特定的操作。

语法：

```
switch(表达式)
{
    case 常量 1:
        语句 1
        [break;]
    case 常量 2:
        语句 2
        [break;]
    ...
    case 常量 n:
        语句 n
        [break;]
    default:
        默认处理语句;
        [break;]
}
```

表达式：必要参数，它的结果用于判断执行的 case 子句。

case 常量：必要参数，它是分支的入口，用于执行特定于常量的操作。

语句：可以是多条语句，但不是复合语句，因为不需要使用花括号“{ }”，它是分支的业务处理代码。

break：可选参数，一般放在 case 分支语句的最后，在完成操作后，用于跳出 switch 语句，继续执行程序的其他流程。

default：可选参数，它是 switch 语句的默认分支入口，如果表达式的值和所有 case 子句的常量都不匹配，那么将执行 default 分支下的语句。

switch 分支语句的流程如图 4.9 所示。

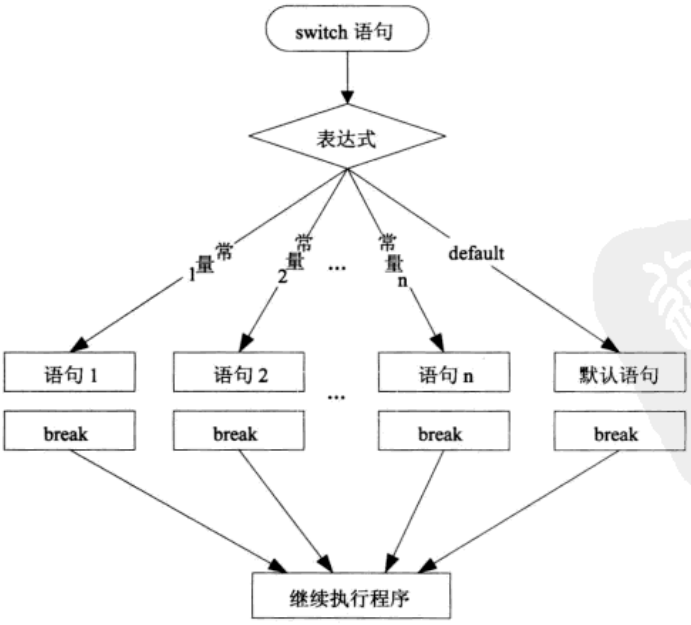


图 4.9　switch 语句流程图

switch 语句要点如下：

- ☑ 语句中表达式类型的级别必须低于 int 整数类型，即表达式的值只能是 byte、short、char、int 类型，不能使用浮点类型或 long 长整型。
- ☑ case 子句中的常量必须和表达式的类型兼容，而且每个 case 子句的常量值都不能相同。
- ☑ 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。
- ☑ 如果 case 语句中没有 break 语句，将继续执行下一个 case 中的语句，直到遇到 break 语句为止。
- ☑ 若没有一个 case 子句的常量与表达式的值相同，则执行 default 分支的语句。default 语句为可选的，如果它不存在，而且 switch 语句中的表达式的值不与任何 case 的常量值相同，switch 不做任何的处理。
- ☑ break 语句用来跳出 switch 语句，也可以说结束 switch 语句，因为 case 子句只是不同分支的标识，用来匹配表达式的值，是分支的入口。当 case 与表达式匹配之后，就开始执行分支的语句，而不再比较之后的任何 case 子句的常量。所以，每个 case 分支的最后，需要使用 break 语句来中断 switch 语句，防止继续执行其他的 case 分支语句或者 default 语句。
- ☑ switch 语句的所有功能都可以使用 if…else 语句实现。switch 语句比 if…else 语句要简练，但是更加复杂的条件分支和单独的条件判断还是要依赖 if 语句去实现。
- ☑ 如果多个相连的 case 分支省略 break 语句可以实现多个条件执行同一业务处理的效果。例如：

```
switch (5+26) {
    case 20:
    case 30:
    case 31:
        System.out.println("我喜欢这个数字");
        break;
    case 100:
        System.out.println("Hi，我更喜欢这个");
        break;
}
```

当表达式的值是 20、30、31 中的任何一个时，都会执行“System.out.println("我喜欢这个数字");”语句。

例 4.06　判断顾客对商品折扣的反应情况，分别定义折扣为 1～2 折、3～4 折、5～6 折、7～8 折和其他折扣时输出的内容。**（实例位置：光盘\mr\ym\4.06）**

代码如下：

```
public class GetSwitch {
    public static void main(String args[]) {
        float rebate = 8; // 声明折扣变量
        switch ((int) rebate) { // 使用 int 类型的表达式
            case 1:
            case 2:
                System.out.println("疯了吗？这个折扣肯定是骗人的");
                break;
            case 3:
            case 4:
                System.out.println("是不是商场要倒闭了");
                break;
            case 5:
            case 6:
```

```
                System.out.println("太好了，这种情况应该去抢购");
                break;
            case 7:
            case 8:
                System.out.println("这应该是促销的价格");
                break;
            default: // 其他折扣
                System.out.println("这个价格和其他商店差不多");
        }
    }
}
```

运行结果如图 4.10 所示。

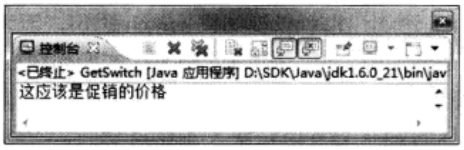


图 4.10　实例运行结果

注意：case 常量表达式的值可以是字符，但一定不可以是字符串，注意“ ”和‘ ’的区别。例如，下面的代码是非法的：

```
switch (66) {
    case 'A':
        break;
    case 'B':
        break;
    case 'C':
}
```

例 4.07　使用 switch 语句判断指定月份属于一年中的哪个季度。如果是 12 月、1 月、2 月，就属于冬季；如果是 3 月、4 月、5 月，就属于春季；如果是 6 月、7 月、8 月，就属于夏季；如果是 9 月、10 月和 11 月，就属于秋季，再出现其他数值就输出错误信息。**（实例位置：光盘\mr\ym\4.07）**

代码如下：

```
public class MonthAtSeason {
    public static void main(String[] args) {
        int month = 4; // 声明月份变量
        switch (month) {
            case 12:
            case 1:
            case 2:
                System.out.println(month + "月份是冬季");
                break;
            case 3:
            case 4:
            case 5:
                System.out.println(month + "月份是春季");
                break;
```

```
            case 6:
            case 7:
            case 8:
                System.out.println(month + "月份是夏季");
                break;
            case 9:
            case 10:
            case 11:
                System.out.println(month + "月份是秋季");
                break;
            default:
                System.out.println(month + "不是合法的月份数值");
                break;
        }
    }
}
```

运行结果如图 4.11 所示。

图 4.11　实例运行结果

4.2.4　if 语句和 switch 语句的区别

if 语句和 switch 语句可以从使用的效率上来进行区别，也可以从实用性的角度来区分。如果从使用的效率上进行区分，在对同一个变量的不同值作条件判断时，使用 switch 语句的效率相对更高一些，判断的分支越多越明显。如果从语句的实用性角度去区分，那 switch 语句肯定不如 if 条件语句。if 条件语句是应用最广泛和最实用的语句。

技巧： *在程序开发的过程中，具体如何来使用 if 和 switch 语句需根据实际情况而定，尽量做到物尽其用，不要因为 switch 语句的效率高就一味地使用，也不要因为 if 语句常用就不应用 switch 语句。要根据实际情况，具体问题具体分析，使用最合适的条件语句。*

4.3　循 环 控 制

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 for 循环语句、while 循环语句和 do…while 循环语句，下面分别对这 3 种循环语句进行介绍。

4.3.1　for 循环

for 循环是 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复合语句，直到循环条件成立而终止循环体。下面将对 for 循环做详细的介绍。

语法：

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

表达式 1：该表达式用于初始化循环控制变量，它只在循环开始时执行一次。

表达式 2：循环条件表达式，它的类型为 boolean，如果表达式的值为 true，则执行一次循环体。

表达式 3：该表达式用于改变循环体变量的值。

for 循环语句的流程如图 4.12 所示。

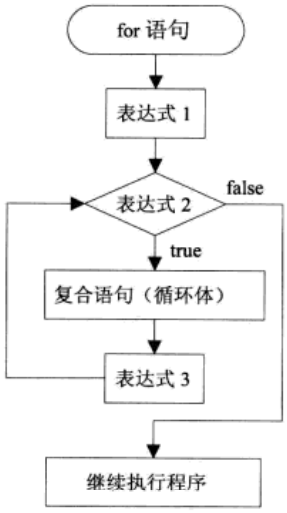


图 4.12　for 循环的流程图

for 循环的执行过程如下。

第一步：执行表达式 1，完成某一变量的初始化工作，如 int i=0。

第二步：判断表达式 2 的值，如 i<100。若表达式 2 的值为 true，则执行一次循环体，然后执行第三步；否则，退出循环体结束 for 循环。

第三步：执行表达式 3，这部分通常执行循环控制变量的修改。

第四步：跳转到第二步继续执行循环。

例 4.08　使用 for 循环计算 100 以内（包括 100）所有偶数之和。**（实例位置：光盘\mr\ym\4.08）**

代码如下：

```
public class Circulate {
    public static void main(String args[]) {
        int sum = 0;                          // 声明总和变量
        for (int i = 0; i <= 100; i += 2) {   // 定义 for 循环
            sum += i;                         // 累加 100 以内的偶数
        }
        System.out.println("100 以内的所有偶数之和为：" + sum);
    }
}
```

运行结果如图 4.13 所示。

例 4.09　使用 for 循环输出 14 和 98 的所有公约数。**（实例位置：光盘\mr\ym\4.09）**

代码如下：

```
public class CommonDivisor {
    public static void main(String[] args) {
        int num1 = 14;
        int num2 = 98;
        for (int i = 1; i <= num1 && i <= num2; i++) {
            if (num1 % i == 0 && num2 % i == 0)
                System.out.println(i + "\t 是 14 和 98 的公约数");
        }
    }
}
```

运行结果如图 4.14 所示。

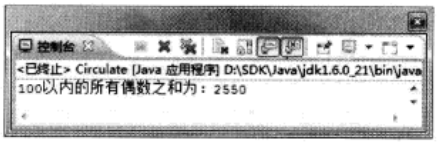


图 4.13　实例运行结果

图 4.14　实例运行结果

技巧：在编程时，有时会遇到使用 for 循环的特殊语法格式来实现无限循环，语法格式如下。

```
for(;;) {
    循环体
}
```

对于这种无限循环，可以通过 break 语句跳出循环，例如下面的代码：

```
for(;;) {
    if(x <20)
    break;
    x++;
}
```

4.3.2　while 循环

while 语句是“当型”循环语句，也就是当条件表达式成立时，执行循环体中的内容。其语法格式为：

```
while(条件表达式){
    循环体
}
```

条件表达式：这是用于控制循环的条件，如 count<100。

循环体：这是条件表达式成立的情况下执行的程序代码。

while 循环语句的流程如图 4.15 所示。

当条件表达式的返回值为真时，则执行循环体中的语句，当执行完循环体中的语句后，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。

例 4.10　通过 while 循环将整数 1～100 相加，并将结果输出。**（实例位置：光盘\mr\ym\4.10）**

代码如下：

```
public class GetSum {
    public static void main(String args[]) {
        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 <= 100) {
            sum += x++;                         // 当变量满足条件表达式时，则执行循环体语句
        }
        System.out.println("sum = " + sum);     // 将变量 sum 输出
    }
}
```

实例中 while 循环体中的 sum += x++;语句等价于 sum = sum+x;和 x = x+1;两条语句，实例运行结果如图 4.16 所示。

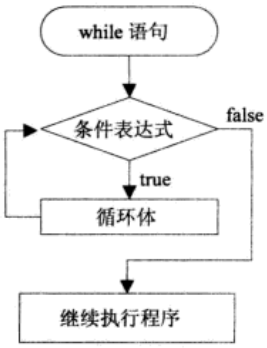


图 4.15　while 语句的执行过程

图 4.16　实例运行结果

例 4.11　使用 while 循环求出 100 以内所有奇数的和。**（实例位置：光盘\mr\ym\4.11）**

代码如下：

```
public class OddNumber {
    public static void main(String[] args) {
        int num = 1;                            // 定义循环控制变量
        int sum = 0;                            // 定义总和变量
        while (num <= 100) {
            sum += num;                         // 等价于 sum=sum+num;
            num += 2;                           // 等价于 num=num+2;
        }
        System.out.println("100 以内所有奇数的和是：" + sum);
    }
}
```

运行的结果如图 4.17 所示。

图 4.17　实例运行结果

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“;”，如下面的代码：

```
while(x = = 5);                          // 分号结束了 while 语句
System.out.println("x 的值为 5");          // 该语句不是循环体
```

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，另外，该情况同样有可能发生在其他控制语句中，如 for 语句和 if 语句。在程序开发时一定要留意这个问题。

4.3.3　do…while 循环

do…while 循环语句与 while 循环语句类似。它们之间的差别是：while 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：

```
do {
    循环体
} while(条件表达式);
```

与 while 语句的一个明显的区别是，do…while 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 循环语句的流程如图 4.18 所示。

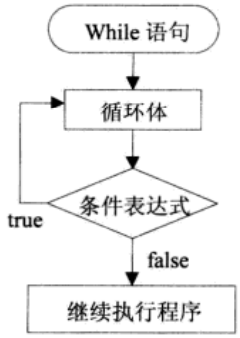


图 4.18　do…while 循环语句的执行过程

do…while 语句的执行步骤如下。

第一步：执行循环体中的语句。

第二步：判断条件表达式的值，如果该值为 true，就跳转到第一步；否则退出循环体，继续执行程序的其他代码。

例 4.12　使用 do…while 循环求 100 以内所有奇数的和。**（实例位置：光盘\mr\ym\4.12）**

代码如下：

```
public class OddNumberForDoWhile {
    public static void main(String[] args) {
        int num = 1;            // 定义循环控制变量
        int sum = 0;            // 定义总和变量
        do {
            sum += num;         // 等价于 sum=sum+num;
            num += 2;           // 等价于 num=num+2;
```

```
        } while (num <= 100);
        System.out.println("100 以内所有奇数的和是：" + sum);
    }
}
```

运行结果如图 4.19 所示。

例 4.13 分别使用 while 和 do…while 语句定义循环，在 num 变量小于 50 的情况下输出循环语句名称，num 变量的初值为 50 或大于 50 的整数。请读者根据本实例牢记两个循环的区别。**（实例位置：光盘\mr\ym\4.13）**

代码如下：

```
public class WhilePK {
    public static void main(String[] args) {
        // while 循环
        int num1 = 50;
        while (num1 < 50) {
            System.out.println("While 循环被执行了");
            num1++;
        }
        // do…while 循环
        int num2 = 50;
        do {
            System.out.println("Do…While 循环被执行了");
        } while (num2 < 50);
    }
}
```

运行结果如图 4.20 所示。

图 4.19 实例运行结果

图 4.20 实例运行结果

实例中的 while 循环首先判断条件表达式，当表达式的结果为 false 时不会执行循环体。而 do…while 循环会先执行一半循环体，然后判断条件表达式，所以无论 num2 变量是多少，do…while 循环都会输出信息。

4.3.4 print 与 println 方法

在本节之前的所有实例中，都是使用 System.out.println()方法输出字符串到控制台的。println()方法在输出字符串时，会自动在字符串的末尾添加“\n”换行字符，所以使用该方法输出的字符串和下一次调用该方法时输出的内容不会在同一行。

print()方法同样用于输出字符串到控制台，该方法输出字符串后不会自动换行，即多次调用该方法输出的字符串永远不会换行，它们会输出在同一行里，除非在字符串中添加\n 或\r 字符。而 println()方法在输出字符串后，会自动添加一个换行符，这是两个方法的区别。

例如：

```
System.out.print("Java");
System.out.print("编程词典");
```

```
System.out.print("的");
System.out.print("网站是：");
System.out.print("www.bccd.com");
```

上述代码在控制台中只输出一行字符串：“Java 编程词典的网站是：www.bccd.com”，而不是多行文本。这两个方法将在 4.3.5 节嵌套循环中多次组合使用。

4.3.5　循环嵌套

在一个循环语句的循环体中，又包含另一个完整的循环语句，称为嵌套循环。3 种不同的循环语句之间可以相互嵌套，嵌套循环可以是多层的，即被嵌套的循环语句还可以再嵌套其他循环，形成 3 层循环或多层循环。但是笔者不建议使用超过 3 层的嵌套循环，因为那会使程序更加难以阅读和理解。

例 4.14　使用双层的 for 循环在控制台中输出格式化的九九乘法表，要使用“\t”制表符使输出更加工整。**（实例位置：光盘\mr\ym\4.14）**

代码如下：

```
public class MultiplicationTable {
    public static void main(String[] args) {
        for (int i = 1; i <= 9; i++) {                  // 第一层循环
            for (int j = 1; j <= i; j++) {              // 第二次循环
                String str = j + "*" + i + "=" + j * i;
                System.out.print(str + "\t");    // 使用制表符格式化输出
            }
            System.out.println();
        }
    }
}
```

这个实例使用了两层 for 循环，第一层 for 循环控制行数，变量 i 在控制循环的同时也用作乘法表的被乘数。第二层循环控制乘法表的列，循环条件是循环控制变量 j 小于等于第一层循环控制变量 i，这样乘法表的列数就受到限制，形成了梯形表格。

运行结果如图 4.21 所示。

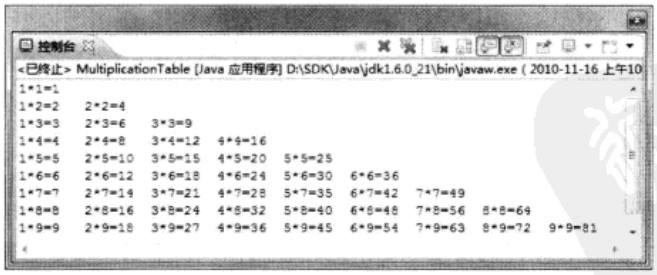


图 4.21　实例运行结果

例 4.15　在控制台中输出金字塔，这个金字塔是由左右对称的数字组成的。**（实例位置：光盘\mr\ym\4.15）**

代码如下：

```
public class Pyramid {
    public static void main(String[] args) {
        int row = 13;                                   // 声明行数
        for (int j = 1; j <= 13; j++) {                 // 第一层循环，控制每一行
```

```
            for (int i = row; i >= j; i--) {          // 嵌套的第一个循环，控制每行左侧空格数量
                System.out.print("    ");             // 输出 3 个空格
            }
            for (int i = j; i >= 1; i--) {            // 嵌套的第二个循环，控制每行左半部输出
                System.out.print(i < 10 ? "  " + i : " " + i);
            }
            for (int i = 2; i <= j; i++) {            // 嵌套的第三个循环，控制每行右半部输出
                System.out.print(i < 10 ? "  " + i : " " + i);
            }
            System.out.println();                     // 控制第一层循环的换行
        }
    }
}
```

实例中使用了两层 for 循环，第一层 for 循环控制金字塔的行，第二层由 3 个 for 循环分别控制每行的前导空格和金字塔的左右两半部分的输出。其中第一个内嵌 for 循环的控制变量使用倒序方式循环，并在循环体中输出空格字符，这样行数越少，输出的空格字符越多，就像金字塔层数越高面积就越小，直到顶层形成了塔尖。第二个内嵌 for 循环输出每行的左半部分数字，它也使用倒序的循环控制，由左至右输出从大到小的数字，其中“i < 10 ? " " + i : " " + i”语句使用三元运算符把小于 10 的数字左侧补两个空格，大于 10 的数字补 1 个空格，这样使输出结果更加工整一些。第三个内嵌的 for 循环以类似的方式在每行的右半部输出从小到大排序的数字。

运行结果如图 4.22 所示。

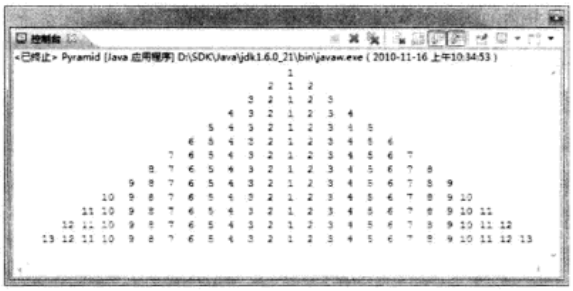


图 4.22　实例运行结果

4.4　循环的跳转

使用循环就像工厂的流水线作业，可以循环生成商品，提高效率。程序中使用循环的目的也是为了提高效率和代码的可读性，但是程序开发需要更复杂的业务流程，一个工作很好的循环体，在某些条件下可能需要终止循环，或者在某些条件下需要放弃本次循环。Java 语言定义的 break 语句和 continue 语句用于循环的跳转，它们可以实现上述功能。

4.4.1　continue 跳转语句

continue 语句只能应用在 for、while 和 do…while 循环语句中，用于结束本次循环直接跳过循环体剩余的语句，进行下一次循环。该语句语法如下：

continue 标号;

参数：标号是可选参数，标号由合法标识符和“:”字符组成，如“for1:”。

continue 语句有两种使用情况，即不带标号跳转和带标号跳转。

☑ 不带标号跳转

如果执行不带标号的 continue 语句，将结束本次循环，跳过循环体中剩余的没有被执行的语句，根据循环的条件判断是否执行下一次循环，或者结束循环。如果是 for 循环，还要先执行表达式 3，改变循环控制变量的值。

例 4.16 实现输出 100 以内能被 7 整除的所有数，使用 continue 语句过滤其他的数字。**（实例位置：光盘\mr\ym\4.16）**

代码如下：

```
public class ContinueNull {
    public static void main(String[] args) {
        System.out.println("100 以内能被 7 整除的数有：");
        for (int i = 1; i < 100; i++) {
            if (i % 7 != 0) {                    // 当 i 的值不能被 7 整除时
                continue;                        // 结束本次循环，放弃不能被 7 整除的数
            }
            System.out.print(i + "、");          // 输出 i 的值
        }
    }
}
```

实例中在 for 循环中加入了 if 判断语句，如果循环控制变量 i 不能被 7 整除，就执行 continue 语句放弃本次循环；否则输出变量 i，这时变量 i 肯定能被 7 整除。运行结果如图 4.23 所示。

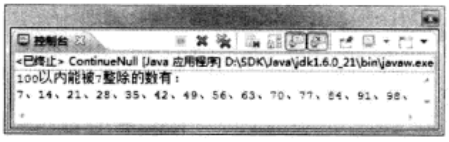


图 4.23　实例运行结果

☑ 带标号跳转

这种情况下，continue 语句不是跳过当前循环语句的一次循环，而是跳过标号指定的循环语句的一次循环，然后判断指定标号的循环条件，以决定是否执行标号指定的循环。

标号定义在语句之前，由一个合法的标识符和“:”字符组成。

语法：

标号:程序代码

continue 语句跳转的标号用于指定循环语句，所以定义在循环语句之前。带标号的 continue 跳转语句经常用在多层循环中。

例 4.17 使用带标号的 for 循环和带标号的 continue 语句输出九九乘法表。**（实例位置：光盘\mr\ym\4.17）**

代码如下：

```
public class MultiplicationTable {
    public static void main(String[] args) {
        outer: for (int i = 1; i <= 9; i++) {          // 第一层循环
            for (int j = 1; j <= 9; j++) {             // 第二层循环
                if (j > i) {                           // 如果 j>i
                    System.out.println();              // 换行
                    continue outer;                    // 跳过循环放弃不需要的乘数
                }
                String str = j + "*" + i + "=" + j * i;
```

```
                System.out.print(str + "\t"); // 使用制表符格式化输出
            }
            System.out.println();
        }
    }
}
```

实例 4.17 中在内层 for 循环中加入了 if 判断语句，如果循环控制变量 j 大于外层循环控制变量 i，那这个数字对于乘法表的当前行就没有用了，所以输出一个空行并执行 continue 语句跳出到 outer 标识的 for 循环，这时内层循环语句就结束或终止了，转去执行 outer 标识的外层循环。

运行结果如图 4.24 所示。

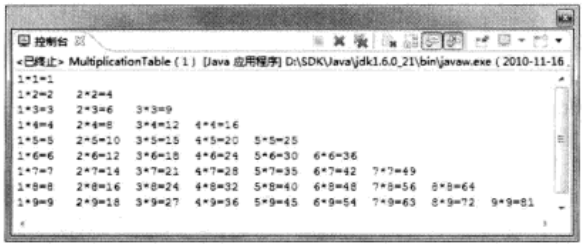


图 4.24　实例运行结果

如果把内层循环的 if 语句注释掉，如下所示：

```
//if (j > i) {                          // 如果 j>i
//    System.out.println();             // 换行
//    continue outer;                   // 跳过循环放弃不需要的乘数
//}
```

然后再运行程序，将会输出如图 4.25 所示的结果。

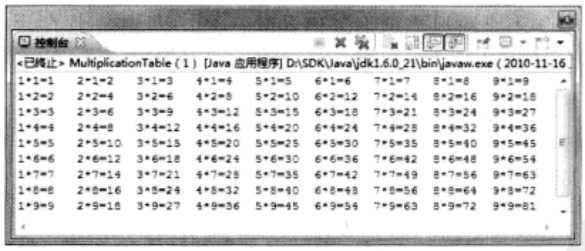


图 4.25　注释 if 语句后实例的运行结果

4.4.2　break 跳转语句

正常来说，退出循环的方式应该由循环条件来判断和控制，而 break 语句可以应用在 for、while 和 do…while 循环语句中，来实现强行退出循环。这也给循环增加了一些灵活性，也就是说，在循环条件为 true 的情况下，也可以根据需要使用 break 语句强行退出循环，该语句语法如下：

```
break 标号;
```

和 continue 语句一样，break 语句也有两种使用情况，即不带标号和带标号。

☑　不带标号

不带标号的 break 语句可以终止 break 语句所在的循环。

例 4.18　使用 while 循环和 break 语句输出 1 至 100 所有整数的和。**（实例位置：光盘\mr\ym\4.18）**

代码如下：

```
public class GetSum {
    public static void main(String[] args) {
        int sum = 0;                          // 声明总和变量
        int control = 1;                      // 声明控制变量
        while (true) {                        // 无限的 while 循环
            sum += control++;                 // 累加总和并步进控制变量
            if (control > 100) {              // 如果控制变量大于 100
                // 输出总和
                System.out.println("1 至 100 所有整数的和是：" + sum);
                break;                        // 终止循环
            }
        }
    }
}
```

实例中使用无限循环的 while 循环不停地累加整数的总和，并且 control 控制变量自增，当控制变量大于 100 时，说明 100 已经被累加到 sum 总和变量中了，这时循环的任务已经完成了，应该输出总和变量的数值，并执行 break 语句退出循环。

运行结果如图 4.26 所示。

☑　带标号

带标号的 break 语句可以终止标号指定的循环，常用于跳出多层循环的情况。

例 4.19　当九九乘法表的积大于 16 时，终止外层循环。**（实例位置：光盘\mr\ym\4.19）**

代码如下：

```
public class BreakTab {
    public static void main(String[] args) {
        loop: for (int i = 1; i <= 9; i++) {
            for (int j = 1; j <= i; j++) {
                if (i * j > 16) {             // 如果乘积大于 16
                    break loop;               // 终止外层循环
                }
                System.out.print(i + "*" + j + "=" + i * j + "\t");
            }
            System.out.print("\r\n");        // 输出一个回车换行符
        }
    }
}
```

运行结果如图 4.27 所示。

图 4.26　实例运行结果

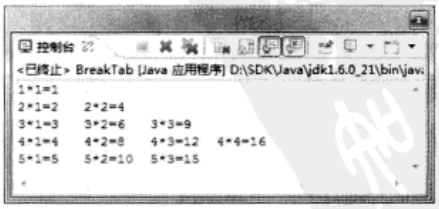


图 4.27　实例运行结果

说明：break 语句和 continue 语句的区别在于，break 语句将终止整个循环语句，而 continue 语句只结束本次循环。

4.5 照猫画虎——基本功训练

4.5.1 基本功训练 1——计算阶乘

视频讲解：光盘\mr\lx\04\计算阶乘.exe

实例位置：光盘\mr\ym\4.20\src

利用 for 循环计算数字 n 的阶乘，实现计算阶乘的公式为“n!= n*(n-1)*(n-2)… *2 *1”，实例代码如下：

```
public class Factorial {
    public static void main(String[] args) {
        int n = 16;
        long result = 1;
        if ((n < 0) || (n > 17)) {
            System.out.println("n 的取值范围是 0 至 17，大于 17 会超出 long 类型范围");
        } else if (n == 0) {
            System.out.println("0 的阶乘等于 1");
        } else {
            for (int i = n; i > 0; i--) {
                result *= i;
            }
            System.out.println(n + "的阶乘等于：" + result);
        }
    }
}
```

可以修改变量 n 的值计算不同数字的阶乘。由于大于 17 的数值的阶乘超过了 long 类型的取值范围，所以程序屏蔽了大于 17 的数字。另外，数学上没有负数的阶乘，所以程序要求 n 必须大于等于 0。0 的阶乘为 1，所以程序直接利用 if 判断输出 0！等于 1，其他数字将使用 for 循环计算结果。

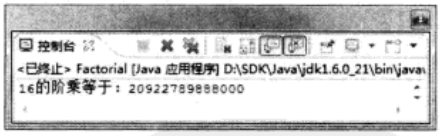


图 4.28 实例运行结果

运行结果如图 4.28 所示。

照猫画虎：接收用户输入的数字，然后计算该数字的阶乘。（20 分）（实例位置：光盘\mr\ym\4.20\zmhh）

4.5.2 基本功训练 2——计算素数

视频讲解：光盘\mr\lx\04\计算素数.exe

实例位置：光盘\mr\ym\4.21\src

素数是大于 1 的整数，并且其正因子只有 1 和它本身。编程求素数，这是常见的练习题，下面通过实例计算 40 个素数并分别显示在 4 行文本中，每行显示 10 个素数，实例代码如下：

```
public class PrimeNumber {
    public static void main(String[] args) {
        final int NUMBER_FOR_PRIMES = 40;                    // 声明素数的数量
```

```
        final int NUMBER_FOR_EVERY_LINE = 10;          // 声明每行输出多少素数
        int count = 0;                                 // 素数计数器
        int number = 2;                                // 用于判断素数的数字
        while (count < NUMBER_FOR_PRIMES) {
            boolean isPrime = true;                    // 是否是素数的状态变量
            for (int i = 2; i <= number / 2; i++) {
                if (number % i == 0) {
                    isPrime = false;
                    break;
                }
            }
            if (isPrime) {                             // 如果是素数
                count++;                               // 累加素数计数器
                System.out.print(number + "\t");       // 输出素数，不换行
                if (count % NUMBER_FOR_EVERY_LINE == 0) {      // 如果已输出 10 个素数
                    System.out.println();              // 换行
                }
            }
            number++;                                  // 自增用于判断的数字
        }
    }
}
```

实例首先定义了两个常量 NUMBER_FOR_PRIMES 和 NUMBER_FOR_EVERY_LINE，它们分别代表输出素数的数量 40 和每行输出素数的数量 10，然后声明了 count 变量作为素数计数器，还有 number 变量作为用于判断素数的数字，这两个变量在程序执行过程中都会被累加，count 变量在找到新的素数时累加，number 变量在判断一个数是否为素数后累加，这样下次判断的就是一个新数。

while 循环在素数达到 40 个之前不会被终止，它内嵌的 for 循环将判断 number 变量的当前数值是否为素数，如果是素数就使 count 变量加 1，然后不换行输出该素数，当 count 能被 10 整除时准备换行。number 变量在 while 循环体的最后是无条件自增的。

实例运行结果如图 4.29 所示。

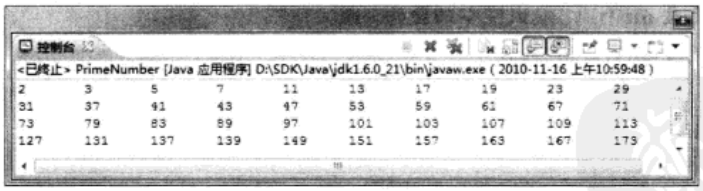


图 4.29　实例运行结果

照猫画虎：编程计算 100 以内 9 的倍数，并把它们输出到控制台。（20 分）（实例位置：光盘\mr\ym\4.21\zmhh）

4.5.3　基本功训练 3——遍历 21 世纪的闰年

视频讲解：光盘\mr\lx\04\遍历 21 世纪的闰年.exe

实例位置：光盘\mr\ym\4.22\src

创建 SearchLeapYear 类，在该类的主方法中定义一个遍历 21 世纪所有年份的循环，在循环中判断这些

年份是否为闰年，并在控制台中把所有闰年年份输出，每行只输出 5 个闰年，实例代码如下：

```
public class SearchLeapYear {
    public static void main(String[] args) {
        int count=0;                                                    // 声明计数器
        for (int year = 2000; year <= 2100; year++) {                   // 遍历 21 世纪年份
            if (year % 4 == 0 && year % 100 != 0 || year % 400 == 0) {  // 是闰年
                System.out.print(year + "是闰年！ \t");                  // 输出闰年
                count++;                                                // 计数器累加
                if(count>4){                                            // 每隔 5 个年份换行
                    System.out.println();
                    count=0;
                }
            }
        }
    }
}
```

实例运行结果如图 4.30 所示。

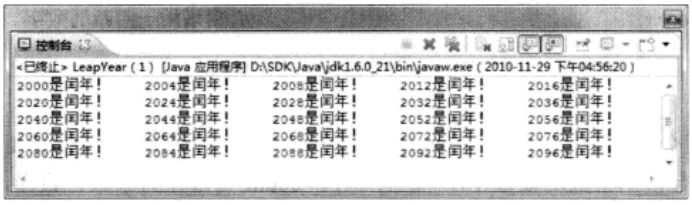


图 4.30　实例运行结果

照猫画虎：修改程序，输出所有非闰年的年份，每行输出 5 个年份。（20 分）（实例位置：光盘\mr\ym\4.22\zmhh）

4.5.4　基本功训练 4——输出空心的菱形

视频讲解：光盘\mr\lx\04\输出空心的菱形.exe

实例位置：光盘\mr\ym\4.23\src

创建 Diamond 类，在该类的主方法中完成 10 行的空心菱形输出，实现方法是使用两个双层 for 循环分别输出菱形的上半部分与下半部分，实例代码如下：

```
public class Diamond {
    public static void main(String[] args) {
        int size = 10;
        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("");
        }
    }
}
```

实例运行结果如图 4.31 所示。

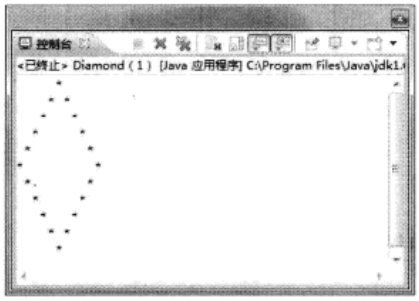


图 4.31　实例运行结果

照猫画虎：修改程序，输出一个等腰三角形，同样要求空心效果。（20 分）（实例位置：光盘\mr\ym\4.23\zmhh）

4.5.5　基本功训练 5——循环的生命力

视频讲解：光盘\mr\lx\04\循环的生命力.exe

实例位置：光盘\mr\ym\4.24\src

循环是常用的开发模式，它可以简化业务处理，提高代码编写与程序运行效率，但是循环中的控制算法要掌握好，否则容易造成死循环从而导致程序崩溃。例如，下面所演示的实例是由于循环控制变量超出取值范围而导致的程序崩溃。

创建 CycUtmost 类，在该类的主方法中创建 int 整数类型的变量 end，使其等于整数类型的最大值，然后将该值减去 50 作为开始循环的起始点，条件是循环控制变量小于等于 end 变量，在循环体中累加循环计数器，最后循环结束时显示这个计数器，实例代码如下：

```
public class CycUtmost {
    public static void main(String[] args) {
        int end=Integer.MAX_VALUE;          // 定义循环终止数
        int start=end-50;                   // 定义循环起始数
        int count=0;                        // 定义循环计数器
        for (int i = start; i <= end; i++) {  // 执行循环
            count++;                        // 循环计数
        }
        // 输出循环计数器
        System.out.println("本次循环次数为："+count);
    }
}
```

读者可能会认为这个程序会循环至少 50 次，然后把计数器的值输出，但实际上这个程序的运行会导致死循环的效果，因为控制条件是检测循环控制变量是否小于或等于整数类型的最大值，当整数类型达到其最大值再累加 1 时会得到整数类型的最小值，所以它永远不可能大于 end 变量，这样就导致了程序的死循环，所以在程序开发时要注意控制变量的取值范围。

照猫画虎：修改程序，把 for 循环中循环控制变量的类型定义为 long 或更高的数据类型，使循环可以正常结束。（20 分）（实例位置：光盘\mr\ym\4.24\zmhh）

照猫画虎栏目分数统计：

照猫画虎题目	1	2	3	4	5	总分数	
分数							

4.6 情景应用——拓展与实践

4.6.1 情景应用 1——把用户输入的字母变成大写

视频讲解：光盘\mr\lx\04\把用户输入的字母变成大写.exe

实例位置：光盘\mr\ym\4.25\src

编程接收用户输入的小写字母，然后把该字母的编码值减去 32，使之变为大写字母。在由小写字母到大写字母的转换过程中，要判断用户输入的内容是否为小写字母，对于除小写字母外的其他字符不做任何处理，输出提示信息即可，实例运行结果如图 4.32 所示。

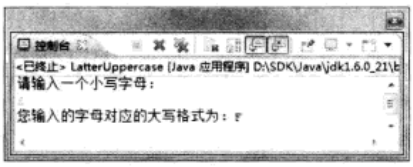


图 4.32 实例运行结果

实现过程如下：

（1）创建 Java 项目，在项目中创建 LatterUppercase 类。

（2）在 LatterUppercase 类的主方法中编写代码接收用户输入，并转换输入字母为大写格式，实例代码

如下：

```
import java.io.IOException;
public class LatterUppercase {
    public static void main(String[] args) throws IOException {
        System.out.println("请输入一个小写字母：");
        int read = System.in.read();                    // 读取用户输入
        if (read >= 97 && read <= 122) {                // 限制输入范围
            char ch = (char) read;                      // 把输入内容转换为字符类型
            ch-=32;                                     // 转换为大写格式
            System.out.println("您输入的字母对应的大写格式为："+ch);
        }else{
            System.out.println("输入的不是小写英文字母。");
        }
    }
}
```

DIY：把程序反过来，接收用户输入的大写字母，然后转换为小写并输出。（15 分）（实例位置：光盘\mr\ym\4.25\diy）

4.6.2　情景应用 2——应用 switch 语句实现生肖查询

视频讲解：光盘\mr\lx\04\应用 switch 语句实现生肖查询.exe

实例位置：光盘\mr\ym\4.26\src

编程接收用户输入的年份数字，输入内容必须是整数，然后通过该数字与 12 求余数的结果判断生肖，并把判断结果输出到控制台，实例运行结果如图 4.33 所示。

图 4.33　实例运行结果

实现过程如下：

（1）创建 Java 项目，在项目中创建 SwitchYear 类。

（2）在 SwitchYear 类的主方法中编写代码接收用户输入的整数，并把该整数作为年份进行运算，获取该年份所对应的生肖，然后输出到控制台，实例代码如下：

```
import java.util.Scanner;
public class SwitchYear {
    public static void main(String args[]) {
        System.out.println("请输入您的出生年份：");
        Scanner scan = new Scanner(System.in);  // 输入扫描器
        int year = scan.nextInt();              // 获取输入年份
                                                // 通过 switch 语句判断生肖
        switch (year % 12) {                    // 年份与 12 求余
            case 4:                             // 余数为 4
                System.out.println("您的生肖为：鼠");
                break;
            case 5:                             // 余数为 5
                System.out.println("您的生肖为：牛");
                break;
            case 6:                             // 余数为 6
                System.out.println("您的生肖为：虎");
```

```
                break;
            case 7:                                    // 余数为 7
                System.out.println("您的生肖为：兔");
                break;
            case 8:                                    // 余数为 8
                System.out.println("您的生肖为：龙");
                break;
            case 9:                                    // 余数为 9
                System.out.println("您的生肖为：蛇");
                break;
            case 10:                                   // 余数为 10
                System.out.println("您的生肖为：马");
                break;
            case 11:                                   // 余数为 11
                System.out.println("您的生肖为：羊");
                break;
            case 0:                                    // 余数为 0
                System.out.println("您的生肖为：猴");
                break;
            case 1:                                    // 余数为 1
                System.out.println("您的生肖为：鸡");
                break;
            case 2:                                    // 余数为 2
                System.out.println("您的生肖为：狗");
                break;
            case 3:                                    // 余数为 3
                System.out.println("您的生肖为：猪");
        }
    }
}
```

DIY：根据用户输入的出生日期，判断其星座。（15 分）（实例位置：光盘\mr\ym\4.26\diy）

4.6.3 情景应用 3——验证用户登录信息

视频讲解：光盘\mr\lx\04\验证用户登录信息.exe

实例位置：光盘\mr\ym\4.27\src

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，则允许用户登录，否则将用户拒之门外。本实例通过 if…else 进行多条件判断实现了登录信息验证，程序运行结果如图 4.34 所示。

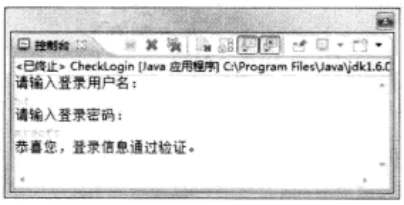


图 4.34 输入合法登录信息的效果

实现过程如下：

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，实例代码如下：

```
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")) {                             // 判断用户名的合法性
            System.out.println("用户名非法。");
        } else if (!password.equals("mrsoft")) {                  // 判断密码的合法性
            System.out.println("登录密码错误。");
        } else {                                                  // 通过以上两个条件判断，则默认通过登录验证
            System.out.println("恭喜您，登录信息通过验证。");
        }
    }
}
```

你问我答：如何判断字符相同？

字符串不同于基本数据类型，不应该使用“==”符号判断是否相等。字符串在 Java 语言中是对象类型，而判断对象是否相同要使用 equals()方法，其语法如下：

```
public boolean equals(Object anObject)
```

参数说明：anObject 是与当前字符串进行比较的其他字符串。例如，本实例使用了 equals()方法判断用户名与密码的合法性。

```
username.equals("mr")
password.equals("mrsoft")
```

DIY：编程接收用户输入的个人姓名与出生年份，使年满 18 岁的用户可以登录。（15 分）（实例位置：光盘\mr\ym\4.27\diy）

4.6.4　情景应用 4——判断用户输入月份的季节

视频讲解：光盘\mr\lx\04\判断用户输入月份的季节.exe

实例位置：光盘\mr\ym\4.28\src

根据用户输入的月份来判断季节，这是一个最典型的实践 switch 语法的例子。通过这个例子，可以完全掌握 switch 语法的用法与技巧，程序运行结果如图 4.35 所示。

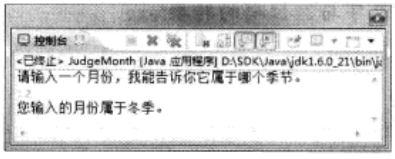


图 4.35　程序运行结果

实现过程如下：

创建 JudgeMonth 类，在该类的主方法中创建扫描器接收用户输入的月份数字，然后判断该月份属于哪个季节并输出到控制台，对于非法月份也要给出提示，实例代码如下：

```
import java.util.Scanner;
public class JudgeMonth {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);          // 创建扫描器
                                                        // 提示用户输入月份
        System.out.println("请输入一个月份，我能告诉你它属于哪个季节。");
        int month = scan.nextInt();                     // 接收用户输入
        switch (month) {                                // 判断月份属于哪个季节
            case 12:
            case 1:
            case 2:
                System.out.print("您输入的月份属于冬季。");
                break;
            case 3:
            case 4:
            case 5:
                System.out.print("您输入的月份属于春季");
                break;
            case 6:
            case 7:
            case 8:
                System.out.print("您输入的月份属于夏季");
                break;
            case 9:
            case 10:
            case 11:
                System.out.print("您输入的月份属于秋季");
                break;
            default:
                System.out.print("你那有" + month + "月份吗？");
        }
    }
}
```

DIY：把程序反过来，用户输入季度，如“春季”，然后程序输出对应的月份。（15 分）（实例位置：光盘\mr\ym\4.28\diy）

4.6.5 情景应用 5——汉字与区位码的转换

视频讲解：光盘\mr\lx\04\汉字与区位码的转换.exe

实例位置：光盘\mr\ym\4.29\src

区位码是一个四位的十进制数，每个区位码都对应着唯一的汉字，区位码的前两位叫做区码，后两位叫做位码。考生在填写高考信息表时会用到汉字区位码，例如，在报考志愿表中需要填写汉字区位码，在程序设计中可以方便地将汉字转换为区位码，实例运行效果如图 4.36 所示。

图 4.36　汉字与区位码的转换

实现过程如下：

（1）在项目中创建窗体类 ChineseToCode，在窗体中添加两个文本框和一个“转换为区位码”按钮。

（2）编写“转换为区位码”按钮的事件处理方法，在该方法中获取用户输入的汉字，提取这个汉字对应的两个字节的编码，然后经过运算获取区位码，这两个区位编码经过 formatNumber()方法的补零之后连接成区位编码字符串显示在文本框中，关键代码如下：

```
protected void do_button_actionPerformed(ActionEvent e) {
    String text = textField.getText();                    // 获取用户输入
    if (text.length() > 2) {                              // 禁止输入多个汉字
        JOptionPane.showMessageDialog(null, "不要输入过多汉字");
        return;
    }
    byte[] codeBit = text.getBytes();                     // 获取汉字的字节数组
    if (codeBit.length < 2) {                             // 禁止非汉字的区码获取
        JOptionPane.showMessageDialog(null, "您输入的好像不是汉字");
        return;
    }
    codeBit[0] -= 160;                                    // 提取字节对应的区码
    codeBit[1] -= 160;
                                                          // 组合最终区码编号
    String code = formatNumber(codeBit[0]) + formatNumber(codeBit[1]);
    resultField.setText(code);                            // 在文本框中显示汉字的区码
}
```

（3）编写 formatNumber()方法，该方法必须为数字参数补零，并把结果字符串返回给方法的调用者，关键代码如下：

```
private String formatNumber(int num) {
    String format = String.format("%02d", num);           // 数字的补零格式
    return format;                                        // 返回格式化后的字符串
}
```

你问我答：如何获取字符串的二进制字节和格式化字符串？

（1）获取二进制字节

得到汉字区位码的过程十分简单，首先通过字符串的 getBytes()方法获取字符串的二进制字节数组，然后将数组中每个二进制值分别减 160 后转换为字符串，连接两个字符串就组成了汉字区位码，获取字符串二进制数组的方法声明如下：

```
public byte[] getBytes()
```

访问数组中的每个元素可以使用下标来指定，如 array[0]是访问数组中的第一个元素。

（2）格式化数字字符串

获取汉字的二进制字节编码以后，可以把它们组合在一起，但是个别编码不足十位会出现低于两位数字的情况，这样组合后的区位码会缺位，必须把不足两位的编码补零，这就需要一些格式化的知识，

而字符串就提供了 format()方法来完成格式化操作，格式化数字字符串的声明如下：

```
public static String format(String format, Object... args)
```

参数说明：format 是一个格式化字符串，它用于指定格式化的模板。args 是被格式化的字符串。如本实例中使用该方法把数字组成的字符串通过格式化保留两位数字，而不足两位的数字会通过补零保证数字位数，代码如下：

```
String format = String.format("%02d", num);                    // 数字的补零格式
```

DIY：把程序改编为控制台接收用户输入的程序，也就是去掉窗体界面。（15 分）（实例位置：光盘\mr\ym\4.29\diy）

情景应用 DIY 栏目分数统计：

DIY 题目	1	2	3	4	5	总分数	
分数							

4.7 自 我 测 试

一、选择题（每题 10 分，5 道题）

1．当编译和运行下列代码时会发生什么？（　　）

```
public class Test {
    public static void main(String[] args) {
        char myChar = 'c';
        switch (myChar) {
            default:
            case 'a':
                System.out.println("a");
                break;
            case 'b':
                System.out.println("b");
                break;
        }
    }
}
```

A．switch 块非法，因为只能使用 int 类型作为参数

B．switch 块合法

C．switch 块非法，因为 default 必须放在块的结尾处

D．运行程序后，控制台无输出

E．运行程序后，控制台输出 a

2．当编译和运行下列代码时会输出什么？（　　）

```
public class Test {
    public static void main(String[] args) {
        int i = 5;
        do {
            System.out.println(i);
```

```
        } while (--i > 5);
        System.out.println("Finished");
    }
}
```

A．6　　B．5　　C．4　　D．Finished

3．当编译和运行下列代码后输出的结果是 Second，则 x 的范围是？（　　）

```
public class Test {
    public static void main(String[] args) {
        int x = (          );
        if (x > 0)
            System.out.println("First");
        else if (x > -3)
            System.out.println("Second");
        else
            System.out.println("Third");
    }
}
```

A．x > 0　　B．x <= 0　　C．x < -3　　D．x <=0 且 x > -3

4．当编译和运行下列代码时会发生什么？（　　）

```
public class Test {
    public static void main(String[] args) {
        boolean flag = false;
        if (flag = true) {
            System.out.println("true");
        } else {
            System.out.println("false");
        }
    }
}
```

A．编译错误　　B．运行错误　　C．输出 true　　D．输出 false

5．当编译和运行下列代码时会发生什么？（　　）

```
public class Test {
    public static void main(String[] args) {
        int total = 0;
        for (int i = 0; i > total; i++) {
            System.out.println(i);
        }
        System.out.println(total);
    }
}
```

A．编译错误　　B．运行错误　　C．输出 0　　D．输出 total

二、填空题（每题 10 分，5 道题）

1．能够终止循环的是_____语句。

2．能够跳过本次循环的语句是_____。

3．for 循环语句中规定了_____个表达式。

4．完成九九乘法表格的输出，需要使用_____层循环语句的嵌套。

5．循环语句中的循环体又称为_____语句。

测试分数统计：

类别	第 1 题	第 2 题	第 3 题	第 4 题	第 5 题
选择题分数					
填空题分数					
总分数					

4.8 行 动 指 南

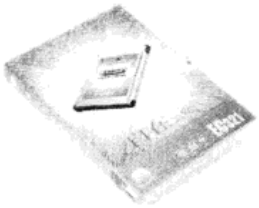
开始日期：_____年___月___日　　结束日期：_____年___月___日

序号	内　容	行 动 指 南	
1	照猫画虎栏目 分数（　　）	分数>75 分	优秀，基本功掌握得不错，加油！
		75 分>分数>50 分	及格，知识掌握得不牢，重新做一遍照猫画虎。
		分数<50 分	请使用光盘中提供的“实战能力测试系统”进行提高训练。
	情景应用栏目 分数（　　）	分数>75 分	优秀，综合应用能力很强。
		75 分>分数>50 分	及格，综合应用能力需提高，再练一遍情景应用。
		分数<50 分	请使用光盘中提供的“实战能力测试系统”进行提高训练。
	自我测试栏目 分数（　　）	分数>75 分	优秀，有成为编程高手的潜质。
		分数<75 分	请使用光盘中提供的“实战能力测试系统”进行提高训练。
	综合评价	反复训练后，以上各项分数都在 75 分以上，方可进入下一堂课学习。	
2	编程能力培养：本栏目提供了一些实践题目，对于培养编程能力很有效，要做好。	（1）接收用户输入的一个角度值，然后通过运行程序输出对应的弧度值。	
		（2）获取从当前年份开始，之后的 10 个闰年的年份，并将其输出到控制台。	
		（3）让用户输入单个字母抽取问卷，程序使用 switch 语句定义 24 个字母组成的条件分支，每个分支对应一个问卷的题目。	
		（4）定义一个长度为 20 的整数数组，数组元素使用随机数初始化，然后通过 for 循环遍历数组，过滤掉所有小于 10 的数组元素，输出大于等于 10 的所有数组元素。	
3	编程习惯培养：本堂课培养读者在学习开发中记笔记的习惯，把开发中遇到的问题、总结的经验记录下来。		
4	创新能力培养：看看身边有没有可以用编程解决的问题，记录到右边的表格中。根据学习进度，尝试编写解决这些问题的小程序。		

4.9　成功可以复制——中国通信设备行业的领跑者任正非

任正非祖籍浙江浦江县，出生在一个贫苦山区的小村庄，任正非的爷爷是一个做火腿的大师傅，任正非父亲的兄弟姊妹都没有读过书。由于爷爷的良心发现，也由于父亲的执著要求，父亲才读了书。任母虽然只有高中文化程度，但是受丈夫影响，通过自修，当上了中学教员。虽然是农村，却是一个知识分子家庭。家庭背景是任正非一生第一个决定性因素。中国的知识分子对知识的重视和追求，可谓“贫贱不能移”。在三年自然灾害时期，父母仍然坚持从牙缝里挤出粮食来让孩子读书。任正非凭借其才智和能力，以及对知识的追求，进入了一个科技密集型行业。在文化大革命时，任正非没有放弃学业，还自学了高等数学、逻辑、哲学和三门外语。任正非的知识渊博、见解独到，在他的讲话中体现为旁征博引、一针见血。后来任正非参军了，作为通讯兵的他被抽调到一个飞机制造厂，参与一项代号为 011 的军事通讯系统工程。当时中央军委提出要重视高科技的作用。任正非上进好学，有多项发明创造，两次填补了国家空白。因技术方面的多次突破，被选为军方代表到北京参加全国科学大会，时年 33 岁。

任正非从部队转业后选择了一家电子公司，而在创办华为时选择了高科技含量的电信行业。1992 年，任正非孤注一掷投入 C&C08 机的研发，虽然是形势所逼，但也可以看出他对技术的重视。当时身处房地产热和股票热的核心地带，任正非不仅不为所动，而且对于股票和泡沫深恶痛绝。其实在他内心更多的是对知识、技术和真才实学的尊重。



华为产品

2006 年美国《新闻周刊》说，尽管创立者任正非一直保持低调，但华为已经与电讯业的几个国际巨头北方电讯、朗讯科技、阿尔卡特、思科系统站在同一水平线展开竞争，而且华为常常可以从它们中赢得更多的网络运营业务。

如今，华为在全球设立了包括印度、美国、瑞典、俄罗斯以及中国的北京、上海、南京等多个研究所，100000 名员工中的 48%从事研发工作，截至 2008 年 6 月，华为已累计申请专利超过 29666 件，已连续数年成为中国申请专利最多的单位。华为在全球建立了 100 多个分支机构，营销及服务网络遍及全球，为客户提供了快速、优质的服务。

经典语录

世界上一切资源都可能枯竭，只有一种资源可以生生不息，那就是文化。

深度评价

对于人生而言，贫穷是老师，他可以教会人如何去生存。贫穷有压力，可以使你的脊梁比别人更硬，坦然吃苦、不屈不挠。在我国许许多多的民营企业家身上都可以找到贫穷的影子，而他们吃苦耐劳、坚韧不拔的品质，往往就是创业的精神所在，因此有人将贫穷当作是成功者的“财富”。

第 5 堂课

数组应用

（视频讲解：90 分钟）

数组是最为常见的一种数据结构。数组中的元素拥有相同的类型，数组即是用一个标识符将这些元素封装到一起的数据序列，这个数据序列可以是基本类型的，也可以是对象类型的。可以用一个统一的数组名和下标来访问数组中的单个指定元素。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。本堂课将向读者介绍有关数组的知识。

学习摘要：

- 一维数组的创建和使用
- 二维数组的创建和使用
- 遍历数组
- Foreach 循环遍历数组
- 填充替换数组中的元素
- 对数组进行排序
- 复制数组

5.1 一维数组的创建与应用

数组是具有相同数据类型的一组数据的集合。例如，球类集合：足球、篮球、羽毛球等；电器集合：电视机、洗衣机、电风扇等。在程序设计中，可以将这些集合称为数组。数组中的每个元素都具有相同的数据类型。在 Java 中可以将数组看作是一个对象，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更有效地管理和处理数据。可根据数组的维数将数组分为一维数组、二维数组…

说明：对象是一种复合数据，在之后的课程中会讲解面向对象编程，会详细地讲解类与对象。

5.1.1 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，必须先定义数组变量所属的类型，即声明数组，语法如下：

```
数组类型 数组名[];
```

或

```
数组类型[] 数组名;
```

参数：数组类型用于决定数组元素的数据类型。它可以是 Java 中任意的数据类型，包括基本数据类型和对象类型。数组名必须是一个合法的标识符。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个一维数组。例如，声明一维数组的关键代码如下：

```
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类型，要想真正使用数组还要为它分配内存空间。在为数组分配内存空间时必须指明数组的长度，为数组分配内存空间的格式如下：

```
数组名字 = new 数组类型[数组长度];
```

参数：数组名字是数组变量的名称。数组长度用于指定数组中成员的个数，即数组的大小。

通过上面的语法可知使用 new 关键字来分配数组时，必须指定数组的类型和数组长度，即数组的大小。例如：

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

以上代码表示要创建长度为 5 的整型数组，并且将创建的数组对象赋给引用变量 arr，即引用变量 arr 引用这个数组，如图 5.1 所示。

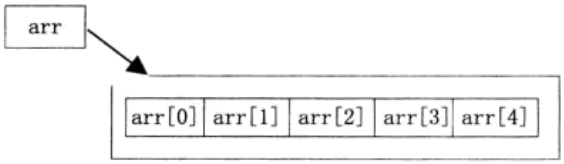


图 5.1 一维数组的内存分配示意图

在图 5.1 中，arr 为数组名称，方括号“[]”中的值为数组的下标。数组通过下标来区分数组中不同的元素。数组的下标是从 0 开始的。由于创建的数组 arr 中有 5 个元素，因此数组中元素的下标为 0～4。

📖 **说明：使用 new 关键字为数组分配内存时，整型数组中的各个元素的初始化值都为 0。**

也可以在声明数组的同时为数组分配内存，这种创建数组的方式是将数组的声明和内存的分配在一起执行，语法如下：

数组元素类型 数组名 = new 数组元素类型[数组元素的个数];

例如，创建数组 month，并指定数组长度是 12 的代码如下：

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

5.1.2 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为数组中每个元素赋值，数组的初始化有两种形式。第一种方式为：

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

第二种数组初始化方式为：

```
int arr[] =new int[4];
arr[0] = 34;
arr[1] = 23;
arr[2] = 12;
arr[3] = 6;
```

数组的初始化就是在花括号内用逗号分开的表达式列表，逗号分开了数组元素的值，系统会根据数据类型自动为数组分配一定的内存空间。

5.1.3 遍历一维数组

在 Java 集合中一维数组是常见的一种数据结构，下面的实例是使用一维数组将 1～12 月各月的天数输出。

例 5.01 在项目中创建类 GetDay，在主方法中创建 int 型数组，并实现将各月的天数输出。代码如下：

（实例位置：光盘\mr\ym\5.01）

```
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.println((i + 1) + "月有" + day[i] + "天");             // 输出的信息
        }
    }
}
```

运行结果如图 5.2 所示。

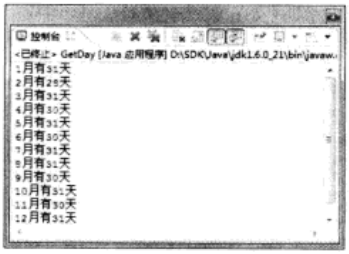


图 5.2　实例运行结果

5.2 二维数组的创建与应用

如果一维数组中的每个元素保存的都是一个数组对象，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。

5.2.1 创建二维数组

二维数组可以看作是特殊的一维数组，因此二维数组的创建同一维数组类似。

声明二维数组的语法为：

数组元素类型 数组名字[][];

或

数组元素类型[][] 数组名字;

例如：

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

二维数组可以表示一个平面表格的数据，第一维数组的下标可以看作表格的行，第二维数组的下标可以看作表格的列，如图 5.3 所示。

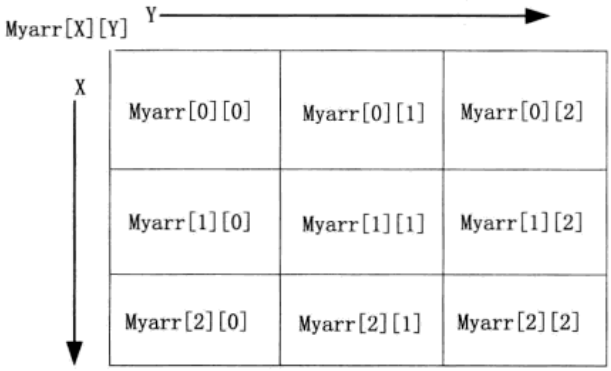


图 5.3 二维数组

同一维数组一样，二维数组在声明时也没有分配内存空间，同样要使用关键字 new 来分配内存，然后才可以访问每个元素。

对于高维数组，有两种为数组分配内存的方式。

（1）直接为二维数组分配内存空间，例如：

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

上述代码创建了二维数组 a，二维数组 a 中包括两个长度为 4 的一维数组，内存分配如图 5.4 所示。

（2）可分别为每一维数组分配内存，例如：

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

第二种方式同第一种方式实现的功能相同。使用这种方式为二维数组分配内存时，首先指定第一维数组的内存，然后单独地给第二维数组分配内存。通过第二种方式为二维数组分配内存，内存分配如图 5.5 所示。

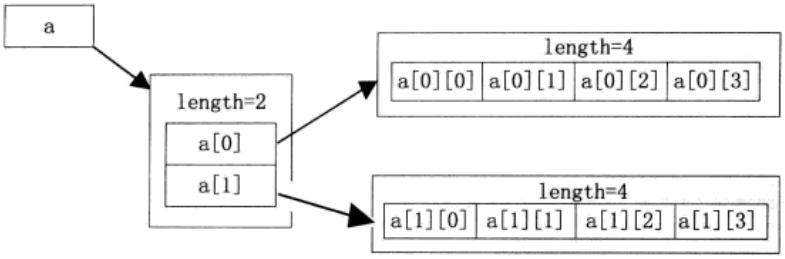


图 5.4　二维数组的第一种内存分配方式

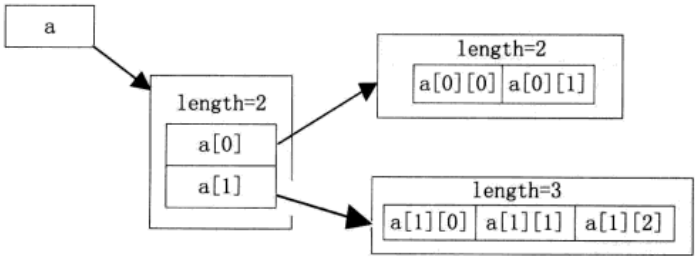


图 5.5　二维数组的第二种内存分配方式

5.2.2　初始化二维数组

二维数组的初始化同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，不同的是每个一维数组的元素使用大括号定义新的一维数组，即一维数组的每个元素又是一个新的一维数组，语法如下：

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

参数说明如下。

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。

例如，初始化二维数组，实例代码如下：

```
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 开始的。如上面的代码中 myarr[1][1]的值为 10。

int 型二维数组是以 int myarr[][]来定义的，所以可以直接给 myarr[x][y]赋值。例如，给 myarr[1]的第二个元素赋值的代码如下：

```
myarr[1][1] = 20;
```

5.2.3　遍历二维数组

二维数组在实际应用中非常广泛。下面的实例就是使用二维数组输出一个 3 行 4 列，且所有元素值都

为字符“*”的矩阵。

📖 **说明：对于整型二维数组，创建成功之后系统会给数组中每个元素的初始化值 0。**

例 5.02 在主方法中编写代码，使用两层循环实现输出一个 3 行 4 列的矩阵。代码如下：**（实例位置：光盘\mr\ym\5.02）**

```
public class Matrix {                                  // 创建类
    public static void main(String[] args) {           // 主方法
        char a[][] = new char[3][4];                    // 定义二维数组
        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
                a[i][j]='*';                            // 初始化数组内容
                System.out.print((char)a[i][j]);        // 将数组中的元素输出
            }
            System.out.println();                       // 输出空格
        }
    }
}
```

实例运行结果如图 5.6 所示。

图 5.6 实例运行结果

5.3 多维数组的创建

所谓多维数组就是二维和大于二维的数组，在 Java 中并不直接支持多维数组，包括二维数组。多维数组的声明是使用一维数组的嵌套声明实现的。就像二维数组，实际上是一个一维数组的每个元素又被声明为一维数组而构成的二维数组，可以说二维数组是特殊的一维数组。虽然二维数组也是多维数组，但是实际开发中也有使用三维数组的情况，本节将以三维数组为例，向读者介绍多维数组。

5.3.1 数组的创建

多维数组的创建可以参考二维数组，例如，二维数组由两个“[]”符号定义，那么三维数组或多维数组就由 3 个或多个“[]”声明，语法如下：

类型 数组名 [][][];

其中 3 个“[]”分别对应第一维数组长度、第二维数组长度和第三维数组长度，例如：

```
int array[2][3][3];
```

二维数组可以看作平面的表格，而三维数组能够表示立体的数据，如图 5.7 所示。

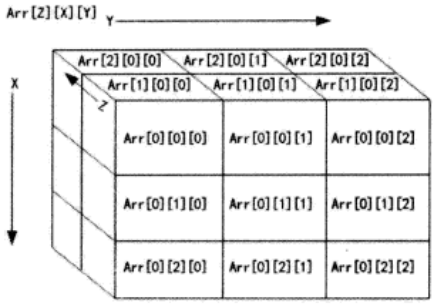


图 5.7　三维数组示意图

5.3.2　初始化数组

和一维、二维数组的初始化方法类似，同样可以使用大括号完成三维数组的初始化，不同的是要使用 3 层花括号“{}”，第一层定义第一维数组，第二层定义第二维数组，第三层定义第三维数组，语法如下：

```
type arrayName[][][] = {                    // 三维数组
    {                                       // 二维数组
        {value1,value2…valueN },            // 一维数组
        {value1,value2…valueN }             // 一维数组
    }
    {
        {value1,value2…valueN },
        {value1,value2…valueN }
    }
    …
};
```

参数说明如下。

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。

例如：

```
int array[][][] = {
        { { 1, 2, 3 }, { 4, 5, 6 } },
        { { 7, 8, 9 }, { 10, 11, 12 } },
        { { 13, 14, 15 }, { 16, 17, 18 } }
    };
```

或者：

```
int array[][][] = new int[][][]{
        { { 1, 2, 3 }, { 4, 5, 6 } },
        { { 7, 8, 9 }, { 10, 11, 12 } },
        { { 13, 14, 15 }, { 16, 17, 18 } }
    };
```

5.3.3　遍历三维数组

三维数组要使用 3 层循环来遍历。遍历时，可以使用 array[z][x][y]的格式指定数组下标来引用数组元素，

数组下标从 0 开始。如“array[1][0][0] = 97;”语句，把 array[1][0][0]即原有内容 7 替换为 97。

例 5.03 使用 3 层 for 循环遍历输出三维数组内容。代码如下：**（实例位置：光盘\mr\ym\5.03）**

```
public class Ransack {
    public static void main(String[] args) {
        int array[][][] = new int[][][]{                          // 创建并初始化数组
                {{ 1, 2, 3 }, { 4, 5, 6 }},
                {{ 7, 8, 9 }, { 10, 11, 12 }},
                {{ 13, 14, 15 }, { 16, 17, 18 }}
            };
        array[1][0][0] = 97;                                      // 改变指定数组元素
        for (int i = 0; i < array.length; i++) {                  // 遍历数组
            for (int j = 0; j < array[0].length; j++) {
                for (int k = 0; k < array[0][0].length; k++) {
                    System.out.print(array[i][j][k] + "\t");
                }
                System.out.println();                             // 输出一维数组后换行
            }
        }
    }
}
```

实例在遍历数组的最内层以循环控制变量 k 控制的循环中输出数组每个元素的内容，它的上层由循环控制变量 j 控制的循环在循环体的最后输出换行，使输出内容更加工整。

其中每个循环体都使用了 length()方法，该方法用于获取数组的长度，array.length()方法判断第一维数组的长度，array[0].length 方法判断下标为 0 的第二维数组的长度，因为本实例中使用的数组是长短相同的规则数组，所以第一个二维数组也就是其他同等级二维数组的长度，那么 array[0][0].length 方法就可想而知了，实例运行结果如图 5.8 所示。

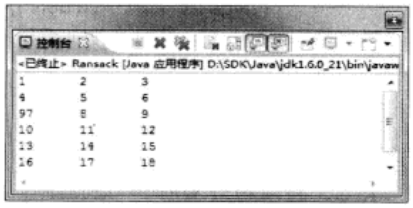


图 5.8 实例运行结果

读者可以尝试在每层循环的循环体开始或结束位置添加输出语句，输出读者能辨识的标识，然后多次运行本实例，有助于加强读者对多维数组的结构和遍历原理的理解。

例如，在第一层循环体开始位置添加如下语句：

```
System.out.println("三维数组第" + (i + 1) + "个元素是一个" + array[0].length
                    + "维数组，其内容如下：");
```

那么输出内容如图 5.9 所示。

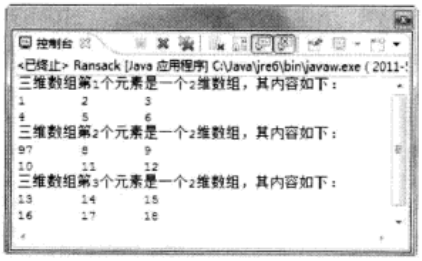


图 5.9 添加代码后的实例运行结果

5.4　照猫画虎——基本功训练

5.4.1　基本功训练 1——填充替换数组元素

视频讲解：光盘\mr\lx\05\填充替换数组元素.exe

实例位置：光盘\mr\ym\5.04\src

数组中的元素定义完成后，可通过 Arrays 类的静态方法 fill()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。下面以 int 型数组为例介绍两种常用的 fill()方法，实例代码如下：

```
import java.util.Arrays;
public class Fill {
    public static void main(String[] args) {
        int arr[] = new int[] { 8, 8, 8, 8, 8, 8, 8 };                    // 定义并初始化 int 型数组 arr
        System.out.println("填充前的数组内容如下：");
        for (int i = 0; i < arr.length; i++) {                            // 循环遍历数组中的元素
            System.out.print(arr[i]+"\t");                                // 将数组中的每个元素输出
        }
        System.out.println();
        Arrays.fill(arr, 1, 4, 7);                                        // 使用 fill 方法对数组进行初始化
        System.out.println("填充下标 1 至 4 所有元素后的数组内容如下：");
        for (int i = 0; i < arr.length; i++) {                            // 循环遍历数组中的元素
            System.out.print(arr[i]+"\t");                                // 将数组中的每个元素输出
        }
    }
}
```

运行结果如图 5.10 所示。

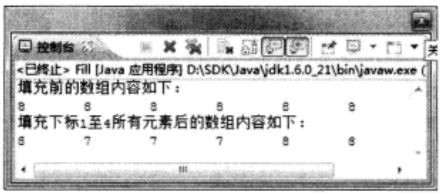


图 5.10　实例运行结果

你问我答：如何填充数组？

数组的填充就是对数组中的所有或部分元素赋值，可以使用各种循环语句在遍历数组的同时进行赋值。另外，Java 提供了 Arrays 类来完成各种常用的数组操作，它提供的众多静态方法中包含了填充数组的以下方法，下面介绍本实例使用过的方法。

（1）填充整个数组

该方法可将指定值分配给数组的每个元素，方法声明如下：

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

参数说明：array 是要进行元素填充的数组，value 是填充到数组中的值。

（2）填充部分数组

该方法将值分配给数组中指定范围的每个元素。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。）方法声明如下：

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

参数说明：array 是要进行填充的数组，fromIndex 用于指定填充的第一个元素的索引（包括），toIndex 用于指定填充的最后一个元素的索引（不包括），value 是要填充到数组中的值。例如，本实例填充部分数组元素的代码如下：

```
int arr[] = new int[] { 8, 8, 8, 8, 8, 8, 8 };
Arrays.fill(arr, 1, 4, 7);
```

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 ArrayIndexOutOfBounds Exception（数组越界异常，关于异常的知识将在后面的章节讲解）异常。

照猫画虎：使用 fill()方法对数组中所有元素赋值为数字 24。（20 分）（实例位置：光盘\mr\ym\5.04\zmhh）

5.4.2 基本功训练 2——对数组进行排序

视频讲解：光盘\mr\lx\05\对数组进行排序.exe

实例位置：光盘\mr\ym\5.05\src

在项目中创建类 Taxis，在主方法中创建一维数组，并将数组排序后在控制台输出，实例代码如下：

```
import java.util.Arrays;                                    // 导入 java.util.Arrays 类
public class Taxis {
    public static void main(String[] args) {
        int arr[] = new int[] { 6, 3, 5, 4, 2, 1 };         // 声明数组
        System.out.println("声明的数组原内容如下：");
        for (int i = 0; i < arr.length; i++) {              // 循环遍历排序后的数组
            System.out.print(arr[i]+"\t");                  // 将排序后数组中的各个元素输出
        }
        System.out.println("\n 排序后的数组内容如下：");
        Arrays.sort(arr);                                   // 将数组进行排序
        for (int i = 0; i < arr.length; i++) {              // 循环遍历排序后的数组
            System.out.print(arr[i]+"\t");                  // 将排序后数组中的各个元素输出
        }
    }
}
```

实例运行结果如图 5.11 所示。

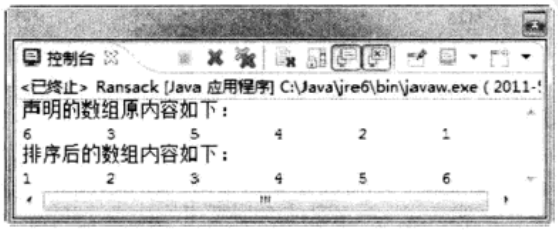


图 5.11 实例运行结果

你问我答：如何利用 Arrays 类实现数组排序？

通过 Arrays 类的静态方法 sort()可实现对数组排序，sort()方法提供了多种重载形式，可对任意类型数组进行升序排序，方法声明如下：

```
Arrays.sort(array)
```

参数说明：array 是要进行排序的数组名称。例如，本实例填充部分数组元素的代码如下：

```
int arr[] = new int[] { 6, 3, 5, 4, 2, 1 };
Arrays.sort(arr);
```

照猫画虎：编程实现对字符类型数组的排序，并输出排序结果。（20 分）（实例位置：光盘\mr\ym\5.05\zmhh）

5.4.3　基本功训练 3——复制数组

视频讲解：光盘\mr\lx\05\复制数组.exe

实例位置：光盘\mr\ym\5.06\src

在主方法中创建一维数组，实现将此数组复制为长度是 5 的新数组，并将新数组输出，实例代码如下：

```
import java.util.Arrays;                                    // 导入 java.util.Arrays 类
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("原数组内容如下：");
        for (int i = 0; i < arr.length; i++) {              // 循环变量复制后的新数组
            System.out.print(arr[i]+"\t");                  // 将新数组输出
        }
        int newarr[] = Arrays.copyOf(arr, 5);               // 复制数组 arr
        System.out.println("\n 复制的新数组内容如下：");
        for (int i = 0; i < newarr.length; i++) {           // 循环变量复制后的新数组
            System.out.print(newarr[i]+"\t");               // 将新数组输出
        }
    }
}
```

实例运行结果如图 5.12 所示。

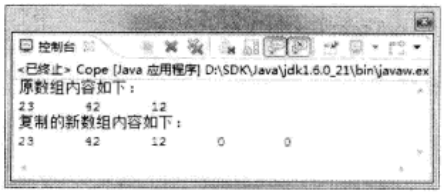


图 5.12　实例运行结果

你问我答：如何利用 Arrays 类实现数组复制？

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()方法将指定数组的指定长度复制到一个新数组中。这两个方法的具体语法介绍如下。

（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，方法声明如下：

copyOf(arr,int newlength)

参数说明：arr 是要进行复制的数组，newlength 是 int 型常量，用于指定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，String 类型数组则使用 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。

（2）copyOfRange()方法

该方法同样提供了多种重载形式，方法声明如下：

copyOfRange(arr,int formIndex,int toIndex)

参数说明：arr 是要进行复制的数组对象。formIndex 是开始复制数组的索引位置，它必须在 0 和整个数组的长度之间取值，新数组将包括索引是 formIndex 的元素。toIndex 是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。

照猫画虎：将数组中索引位置是 0～3 之间的元素复制到新数组中。（20 分）（实例位置：光盘\mr\ym\5.06\zmhh）

5.4.4 基本功训练 4——Foreach 循环遍历数组

视频讲解：光盘\mr\lx\05\Foreach 循环遍历数组.exe

实例位置：光盘\mr\ym\5.07\src

在项目中创建主类，在类的主方法中使用 3 层 Foreach 循环遍历输出三维数组内容，实例代码如下：

```
public class ForEachRansack {
    public static void main(String[] args) {
        int array[][][] = new int[][][]{                    // 创建并初始化数组
                {{1, 2, 3}, {4, 5, 6}},
                {{7, 8, 9}, {10, 11, 12}},
                {{13, 14, 15}, {16, 17, 18}}
            };
        for (int[][] is : array) {                          // 遍历数组
            for (int[] is2 : is) {
                for (int i : is2) {
                    System.out.print(i + "\t");
                }
                System.out.println();                       // 输出一维数组后换行
            }
        }
    }
}
```

array 是三维数组，其中的元素都是二维数组，所以第一个 Foreach 的变量是 int[][]类型，而二维数组中的元素是一维数组，所以第二个 Foreach 的变量是 int[]。

运行结果如图 5.13 所示。

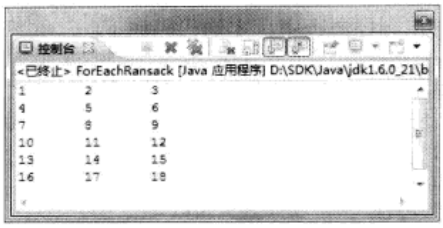


图 5.13　实例运行结果

技巧：在 Eclipse 中输入“fore”，然后按 Alt+/快捷键，在弹出的菜单中选择 Foreach 系列的模板命令，如选择遍历数组的模板，会自动添加 Foreach 语法格式。

你问我答：Foreach 是 for 循环吗？

Foreach 是 for 循环的简化格式，主要用于仅执行遍历功能的循环。例如，遍历数组与任何 Java 中的数据集合对象，在这里讲解 Foreach，主要是因为在讲解 for 循环时还没有介绍数组或集合对象，语法如下：

```
for(类型 var:集合){
    循环体
}
```

参数说明如下。

类型：是定义变量的类型。

var：是定义的遍历集合的变量。

集合：是被遍历的集合对象或数组。

例如：

```
int[] array={12,43,43,22,13,58};          // 定义数组
for (int i : array) {                     // Foreach 变量数组
    System.out.println(i);                // 输出内容
}
```

在循环体执行的过程中，var 变量将被赋值为集合或数组中的每个元素，和 for 语句的标准格式相比，Foreach 更加简便，更利于遍历集合对象与数组，但是缺少了循环条件，而且默认的循环条件是把指定的集合或数组从头循环到尾。虽然 Foreach 遍历数组或集合对象比较简便，但是在需要元素索引的情况下还是需要使用 for 语句的标准格式。

照猫画虎：使用 Foreach 遍历字符类型的数组，把数组中所有元素输出到控制台。（20 分）（实例位置：光盘\mr\ym\5.07\zmhh）

5.4.5　基本功训练 5——找出数组中最大值

视频讲解：光盘\mr\lx\05\找出数组中最大值.exe

实例位置：光盘\mr\ym\5.08\src

一维数组常用于保存线性数据，如数据库中的单行数据就可以使用一维数组保存。另外，数组也需要查询排序等算法的支持，以扩展程序处理数据的灵活性。本实例将定义整数类型的一维数组，然后通过循

环找出数组中的最大元素值，并输出到控制台中，实例代码如下：

```
public class SearchMaxNumber {
    public static void main(String[] args) {
        int[] numArray = { 45, 226, 484, 12, 1, 968, 1, 91, 8, 4515, 6848, 6 };
        int max = numArray[0];              // 创建最大数变量
        for (int j = 0; j < numArray.length; j++) {
            if (numArray[j] > max) {        // 提取最大整数
                max = numArray[j];
            }
        }
        System.out.println("数组 numArray 中的最大元素值是："+max);
    }
}
```

实例运行结果如图 5.14 所示。

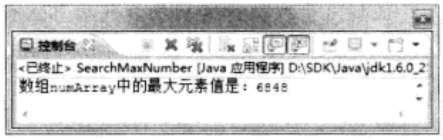


图 5.14　实例运行结果

照猫画虎：修改程序，找出数组中最小的值。（20 分）（实例位置：光盘\mr\ym\5.08\zmhh）

照猫画虎栏目分数统计：

照猫画虎题目	1	2	3	4	5	总分数	
分数							

5.5　情景应用——拓展与实践

5.5.1　情景应用 1——使用冒泡排序

视频讲解：光盘\mr\lx\05\使用冒泡排序.exe

实例位置：光盘\mr\ym\5.09\src

冒泡排序是数组排序算法中比较简单也比较常用的，它的排序方法类似于冒泡动作，所以起名为冒泡排序算法。另外，这也是学校考试和公司面试的常见考题。对于学习程序算法来说，冒泡排序是一个很好的入门题目。下面通过实例实现冒泡算法的应用，实例运行结果如图 5.15 所示。

图 5.15　实例运行结果

实现过程如下：

（1）创建项目，并在项目中创建 BubbleSort 类。

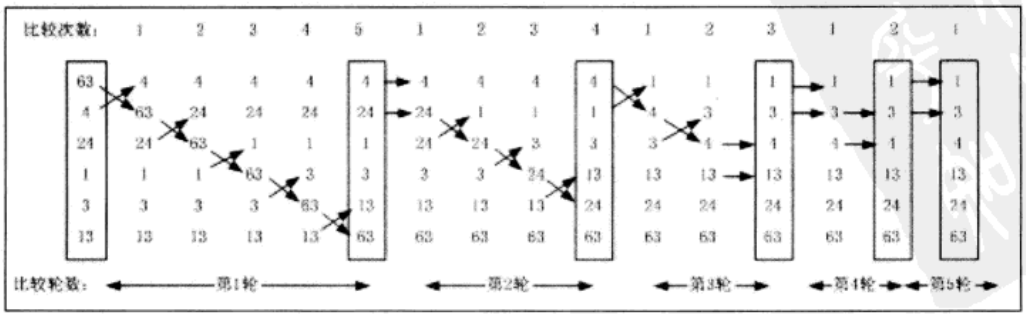
（2）在 BubbleSort 类的主方法中创建一个整数类型的数组，然后通过冒泡算法完成数组从小到大顺序的排序，并把数组中原有内容与排序后的内容输出到控制台，实例代码如下：

```
public class BubbleSort {
    public static void main(String[] args) {
        int[] array = new int[] { 5, 1, 2, 8, 4, 6, 9, 7, 3, 0 };
        int temp;                                                    // 临时变量
        System.out.println("原有数组内容：");
        printArray(array);
        // 从小到大排序
        for (int i = 1; i < array.length; i++) {                     // 排序数组的范围（整个数组）
            for (int j = 0; j < array.length - i; j++) {             // 比较相邻元素，较大的数往后冒泡
                if (array[j] > array[j + 1]) {
                    temp = array[j];                                 // 交换相邻的两个数
                    array[j] = array[j + 1];
                    array[j + 1] = temp;
                }
            }
        }
        System.out.println("从小到大排序后的结果：");
        printArray(array);                                           // 输出冒泡排序后的数组内容
    }
    /**
     * 遍历数组的方法，该方法输出数组中所有内容
     * @param array 要遍历的数组
     */
    public static void printArray(int[] array) {
        for (int i : array) {
            System.out.print(i + " ");
        }
        System.out.println("\n");
    }
}
```

你问我答：冒泡排序的实现原理是什么？

本实例实现时主要用到了冒泡排序算法，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，把较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。

冒泡算法在双层循环中实现，其中外层循环控制排序轮数，要排序数组长度-1 次。而内层循环主要是用于对比临近元素的大小，以确定是否交换位置，对比和交换次数依排序轮数而减少。例如，一个拥有 6 个元素的数组，在排序过程中每一次循环的排序过程和结果如下：



第一轮外层循环时把最大的元素值 63 移动到了最后面(相应的比 63 小的元素向前移动,类似气泡上升),第二轮外层循环不再对比最后一个元素值 63,因为它已经确认为最大(不需要上升),应该放在最后,需要对比和移动的是其他剩余元素,这次将元素 24 移动到了 63 的前一个位置。其他循环将依此类推,继续完成排序任务。

DIY:把程序反过来,实现数组从大到小顺序的排序,并输出排序结果到控制台。(20 分)(实例位置:光盘\mr\ym\5.09\diy)

5.5.2 情景应用 2——使用直接选择排序法

视频讲解:光盘\mr\lx\05\使用直接选择排序法.exe

实例位置:光盘\mr\ym\5.10\src

直接选择排序法是选择排序的一种,该算法的交换次数比冒泡排序少很多,所以选择排序比冒泡排序快。下面的实例中使用直接选择排序法对数组进行排序,它也是数组常见的应用,实例运行结果如图 5.16 所示。

图 5.16 实例运行结果

实现过程如下:

(1)创建项目,并在该项目中创建 SelectSort 类。

(2)在 SelectSort 类的主方法中创建一个整数类型的数组,然后通过直接选择排序法完成数组的正序排列,并把数组中原有内容与排序后的内容输出到控制台,实例代码如下:

```
package com.lzw;
public class SelectSort {
    public static void main(String[] args) {
        int[] array = new int[] { 5, 1, 2, 8, 4, 6, 9, 7, 3, 0 };
        int temp;                                          // 临时变量
        System.out.println("原有数组内容:");
        printArray(array);                                 // 输出原有数组
        int index;                                         // 索引变量
        for (int i = 1; i < array.length; i++) {           // 正序排列
            index = 0;
            for (int j = 1; j <= array.length - i; j++) {
                if (array[j] > array[index]) {
                    index = j;
                }
            }
            // 交换在 array.length-i 和 index 两个位置上的数组元素
            temp = array[array.length - i];
            array[array.length - i] = array[index];
            array[index] = temp;
        }
        System.out.println("正序排列数组内容:");
        printArray(array);                                 // 输出排序后的数组
    }
    /**
     * 遍历数组的方法,该方法输出数组中所有内容
```

```
     * @param array 要遍历的数组
     */
    public static void printArray(int[] array) {
        for (int i : array) {
            System.out.print(i + " ");
        }
        System.out.println("\n");
    }
}
```

你问我答：直接选择排序的实现原理是什么？

直接选择排序的基本思想是将指定排序位置与其他数组元素分别对比，如果满足条件就交换元素值，注意这里区别冒泡排序不是交换相邻元素，而是把满足条件的元素与指定的排序位置交换（如从最后一个元素开始排序），这样，排序好的位置逐渐扩大，最后整个数组都成为已排序好的格式。

这就好比有一个小学生，从包含 1～10 数字的乱序数字堆中分别选择合适的数字，组成一个从 1 到 10 的排序，而这个学生首先从数字堆中选出 1，放在第一位，然后选出 2（注意这时数字堆中已经没有 1 了），放在第二位，依此类推，直到找到数字 9 并放到 8 的后面，最后剩下 10，就不用选择了，直接放到最后就可以了。

这样的话，如果与冒泡排序相比，直接选择排序的交换次数要少很多，所以速度会快些。

每一遍从待排序的数据元素中选出最小（或最大）的一个元素，顺序放在已排好序的数列的最后，直到全部待排序的数据元素排完。

举例：

初始数组资源【63　4　24　1　3　15】
第一遍排序后【15　4　24　1　3】　63
第二遍排序后【1　4　3　1】　24　63
第三遍排序后【1　4　3】　15　24　63
第四遍排序后【1　3】　4　15　24　63
第五遍排序后【1】　3　4　15　24　63

DIY：使用直接选择排序算法对 char 类型的数组进行排序。（20 分）（实例位置：光盘\mr\ym\5.10\diy）

5.5.3　情景应用 3——使用快速排序法

视频讲解：光盘\mr\lx\05\使用快速排序法.exe

实例位置：光盘\mr\ym\5.11\src

快速排序（QuickSort）是对气泡排序的一种改进，其排序速度相对较快。本实例演示如何使用快速排序法对一维数组进行排序，实例中首先定义了一个乱序的字符数组，然后分别把原有数组内容与排序好的数组内容输出到控制台中，实例运行结果如图 5.17 所示。

图 5.17　使用快速排序法对一维数组进行排序

实现过程如下：

（1）在 Eclipse 中创建一个 Java 项目，在项目中创 QuickSort 类。

（2）在 QuickSort 类中编写遍历数组的 printArray()方法，用该方法实现数组内容的输出，避免多处出现输出数组内容的代码，关键代码如下：

```
public static void printArray(char[] array) {
    for (char i : array) {                          // 遍历数组
        System.out.print(i + " ");                  // 输出数组内容
    }
    System.out.println("\n");                       // 输出换行符
}
```

（3）编写快速排序方法 quickSort()，在这个方法中实现数组的快速排序，即对一个字符数组内容进行排序，关键代码如下：

```
private static void quickSort(char sortarray[], int lowIndex, int highIndex) {
    int lo = lowIndex;                              // 记录最小索引
    int hi = highIndex;                             // 记录最大索引
    int mid;                                        // 记录分界点元素
    if (highIndex > lowIndex) {
        mid = sortarray[(lowIndex + highIndex) / 2];    // 确定中间分界点元素值
        while (lo <= hi) {
            while ((lo < highIndex) && (sortarray[lo] < mid))
                ++lo;                               // 确定不大于分界元素值的最小索引
            while ((hi > lowIndex) && (sortarray[hi] > mid))
                --hi;                               // 确定大于分界元素值的最大索引
            if (lo <= hi) {                         // 如果最小与最大索引没有重叠
                char temp = sortarray[lo];          // 交换数组元素
                sortarray[lo] = sortarray[hi];
                sortarray[hi] = temp;               // 交换两个索引的元素
                ++lo;                               // 递增最小索引
                --hi;                               // 递减最大索引
            }
        }
        if (lowIndex < hi)                          // 递归排序没有未分解元素
            quickSort(sortarray, lowIndex, hi);
        if (lo < highIndex)                         // 递归排序没有未分解元素
            quickSort(sortarray, lo, highIndex);
    }
}
```

（4）最后，要完成实例的主方法，在主方法中定义 char 类型的数组，并初始化数组内容为乱序的字符序列，然后调用定义的 quickSort()方法将数组排序，最后输出排序结果，关键代码如下：

```
public static void main(String[] args) {
    char[] array = { 'a', 'b', 'f', 'e', 'g', 'd', 'h', 'c', 'j', 'i' };
    System.out.println("原有数组内容：");
    printArray(array);                              // 输出原有数组
    quickSort(array, 0, array.length - 1);
    System.out.println("正序排列数组内容：");
    printArray(array);                              // 输出排序后的数组
}
```

你问我答：如何获取数组长度？

数组拥有一个唯一的属性“长度”，它记录了数组本身元素的数量。获取该属性值的语法如下：

数组.length

参数：数组是任意一个数组变量。

例如，本实例中调用 quickSort()方法获取 array 数组长度的代码如下：

```
quickSort(array, 0, array.length - 1);
```

注意：数组的长度可以在创建时指定，但是指定之后就不能再修改了，否则会出现编译错误。

DIY：尝试使用快速排序算法为整数数组排序，但输出时把整数转换为字符类型。（20 分）（实例位置：光盘\mr\ym\5.11\diy）

5.5.4　情景应用 4——反转数组中元素的顺序

视频讲解：光盘\mr\lx\05\反转数组中元素的顺序.exe

实例位置：光盘\mr\ym\5.12\src

顾名思义，反转数组就是以相反的顺序把原有数组的内容重新排序。反转排序算法在程序开发中也经常用到。下面通过实例来介绍反转排序的具体用法，实例运行结果如图 5.18 所示。

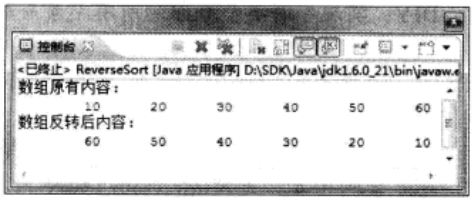


图 5.18　实例运行结果

实现过程如下：

（1）在项目中创建 ReverseSort 类，这个类的代码将作为反转排序的一个演示。

（2）在该类中编写 showArray()方法，该方法将负责显示数组内容，代码如下：

```
public static void showArray(int[] array) {
    for (int i : array) {                    // Foreach 格式遍历数组
        System.out.print("\t" + i);          // 输出每个数组元素值
    }
    System.out.println();
}
```

（3）编写 ReverseSort 类的主方法，在主方法中定义一个整数类型的数组，为演示数组反转效果，该数组的所有数组元素使用正序排列，然后通过反转数组算法重新排列所有元素，关键代码如下：

```
public static void main(String[] args) {
                                             // 创建一个数组
    int[] array = { 10, 20, 30, 40, 50, 60 };
                                             // 调用排序对象的方法将数组反转
    System.out.println("数组原有内容：");
    showArray(array);                        // 输出排序前的数组值
    int temp;
    int len = array.length;
    for (int i = 0; i < len / 2; i++) {
        temp = array[i];
        array[i] = array[len - 1 - i];
        array[len - 1 - i] = temp;
    }
    System.out.println("数组反转后内容：");
```

```
        showArray(array);                    // 输出排序后的数组值
    }
```

你问我答：反转算法的原理是什么？

反转实例比较简单，也很好理解，其原理就是，把数组中最后一个元素与第一个元素替换，倒数第二个元素与第二个元素替换，依此类推，直到把所有数组元素反转替换。

反转排序是对数组两边的元素进行替换，所以只需要循环数组长度的半数。例如，数组长度为 7，那么 for 循环只需要循环 7/2 也就是 3 次。

举例：

初始数组资源	【10	20	30	40	50	60】
第一遍排序后	60	【20	30	40	50】	10
第二遍排序后	60	50	【30	40】	20	10
第三遍排序后	60	50	40	15	20	10

DIY：实现字符类型数组的所有元素反转。（20 分）（实例位置：光盘\mr\ym\5.12\diy）

5.5.5 情景应用 5——利用数组随机抽取幸运观众

视频讲解：光盘\mr\lx\05\利用数组随机抽取幸运观众.exe

实例位置：光盘\mr\ym\5.13\src

在电视节目中，经常看到随机抽取幸运观众。如果观众抽取的范围较小，可以让程序使用数组实现，这样效率很高。实现的方法为：首先将所有观众姓名生成数组，然后获得数组元素的总数量，在数组元素中随机抽取元素的下标索引，根据抽取的下标获得数组的元素作为幸运观众，实例运行效果如图 5.19 所示。

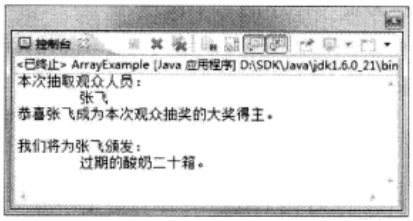


图 5.19 实例运行结果

实现过程如下：

（1）创建 Java 项目，在项目中创建 ArrayExample 类。

（2）在 ArrayExample 类的主方法中定义字符串数组，然后生成随机的数组下标索引，通过这个随机的索引值读取数组中的某个数组元素作为中奖人员名称，然后输出中奖信息，实例代码如下：

```
public class ArrayExample {
    public static void main(String[] args) {
        // 定义人员数组
        String[] personnelArray = { "赵云", "张飞", "吕布", "貂蝉", "关羽", "诸葛亮", "刘备" };
        double randomNum = (double) Math.random();                    // 生成随机数
        int index = (int) (randomNum * personnelArray.length);        // 把随机数转换为数组索引
        // 定义中奖信息
        String info = "本次抽取观众人员：\n\t" + personnelArray[index] + "\n 恭喜"
                + personnelArray[index] + "成为本次观众抽奖的大奖得主。" + "\n\n 我们将为"
                + personnelArray[index] + "颁发：\n\t 过期的酸奶二十箱。";
        System.out.println(info);                                     // 输出中奖信息}
    }
}
```

你问我答：如何生成随机数？

抽奖当然是随机抽取的，这就需要用到随机数，Java 在 Math 类中提供了静态方法 random()，可以生成 0～1 之间的 double 类型随机数，生成随机数的方法如下：

```
public static double random()
```

由于该方法生成的是 0～1 之间的小数，而数组下标是整数且又要根据数组长度来生成随机数，所以要把生成的随机数与数组长度相乘，就像本实例中的算法那样，关键代码如下：

```
double randomNum = (double) Math.random();
int index = (int) (randomNum * personnelArray.length);
```

这段代码把随机数与数组长度的乘积转换为整数类型作为随机数组下标索引。

DIY：生成一个由随机数初始化的数组，并使用冒泡排序算法对其进行排序。（20 分）（实例位置：光盘\mr\ ym\5.13\diy）

情景应用 DIY 栏目分数统计：

DIY 题目	1	2	3	4	5	总分数	
分数							

5.6 自 我 测 试

一、选择题（每题 10 分，5 道题）

1．当编译和运行下列代码时会发生什么？（　　）

```
public class Test {
    public static void main(String[] args) {
        char[] array = {'a',,'b'};
        System.out.println(array[0]);
    }
}
```

A．编译错误　　B．运行错误　　C．输出 a

2．当编译和运行下列代码时会发生什么？（　　）

```
public class Test {
    public static void main(String[] args) {
        long size = 10;
        int[] array = new int[size];
        size = 20;
        System.out.println(array.length);
    }
}
```

A．编译错误　　B．运行错误　　C．输出 10　　D．输出 20

3．下列定义二维数组的语句中，哪些是正确的？（　　）

A．double[][] array = new double[6][6];　　B．double array[][] = new double[6][6];

C．double[][] array = new double[][6];　　D．double array[][] = new double[][6];

E．double[][] array = new double[6][];　　F．double array[][] = new double[6][];

4．当编译和运行下列代码时会发生什么？（　　）

```
public class Test {
    public static void main(String[] args) {
        int[] array = new int[5];
        System.out.println(array[5]);
    }
}
```

A．编译错误　　B．运行错误　　C．输出 0　　D．输出 null

5．当编译和运行下列代码时会发生什么？（　　）

```
public class Test {
    public static void main(String[] args) {
        int[] array = new int[10];
        array.length = 15;
        System.out.println(array.length);
    }
}
```

A．编译错误　　B．运行错误　　C．输出 10　　D．输出 15

二、填空题（侧重实用技能，每题 10 分，5 道题）

1．Java 中的数组在使用 new 操作符创建时是否被初始化？_____

2．如果需要获得数组中可以保存的元素数量，可以使用数组的什么属性？_____

3．main()方法中的参数是一个数组，这个数组是什么类型的数组？_____

4．多维数组是由一维数组组合而成，这种说法是否正确？_____

5．遍历数组的 Foreach 是所有循环语句的简写格式吗？_____

测试分数统计：

类别	第 1 题	第 2 题	第 3 题	第 4 题	第 5 题
选择题分数					
填空题分数					
总分数					

5.7 行 动 指 南

开始日期：_____年____月____日　　结束日期：_____年____月____日

序号	内　容	行 动 指 南	
1	照猫画虎栏目	分数>75 分	优秀，基本功掌握得不错，加油！
		75 分>分数>50 分	及格，知识掌握得不牢，重新做一遍照猫画虎。
	分数（　　）	分数<50 分	请使用光盘中提供的“实战能力测试系统”进行提高训练。
	情景应用栏目	分数>75 分	优秀，综合应用能力很强。
		75 分>分数>50 分	及格，综合应用能力需提高，再练一遍情景应用。
	分数（　　）	分数< 50 分	请使用光盘中提供的“实战能力测试系统”进行提高训练。

续表

序号	内　容	行 动 指 南	
1	自我测试栏目 分数（　　）	分数>75 分	优秀，有成为编程高手的潜质。
		分数<75 分	请使用光盘中提供的“实战能力测试系统”进行提高训练。
	综合评价	反复训练后，以上各项分数都在 75 分以上，方可进入下一堂课学习。	
2	编程能力培养：本栏目提供了一些实践题目，对于培养编程能力很有效，要做好。	（1）在控制台接收用户输入的 10 个数字组成一个数组，然后在程序中把数组倒序排列，并输出到控制台。	
		（2）使用二维数组定义一个课程表，并输出到控制台。提示：根据每天课程多少，第二维数组的长度可能不相同。	
		（3）创建一个长度为 10 的数组，然后判断数组中最大值与最小值所在的索引位置，截取它们之间的数组元素组成新的数组。	
		（4）创建一个包含任意元素值的数组，在该数组长度范围内生成随机的索引值，把该索引值对应的数组元素输出到控制台，让用户输入对应的索引值，并判断正确性，然后输出判断结果。	
3	编程习惯培养：本堂课培养读者在学习开发中记笔记的习惯，把开发中遇到的问题、总结的经验记录下来。		
4	创新能力培养：看看身边有没有可以用编程解决的问题，记录到右边的表格中。根据学习进度，尝试编写解决这些问题的小程序。		

5.8　成功可以复制——软件业的华人教父王嘉廉

王嘉廉（Charles Wang）1944 年出生于上海法租界内，父亲曾是最高法院的法官。1952 年，8 岁的王嘉廉随父母移民美国，先是进入布鲁科技高中就读，后在市立大学皇后学院毕业。那时生活异常艰辛，王嘉廉常做一些被人瞧不起的活，经常连 32 美分一顿的午饭都吃不上。许多年后，回忆起早年的痛苦，他总是轻描淡写地带过，只是常说自己比比尔·盖茨经历的磨难多。“我知道饥饿的滋味，而他却不知道。”

虽然与盖茨将近 1000 亿美元的个人财富相比，王嘉廉的 10 多亿个人资产不过是盖茨的一个零头，但人们还是常常把他们放在一起，他们都是自己公司的中心和灵魂，都是凭借铁腕式的战略攻城略地，成为软件业头两号厂商。微软在 PC 领域一统天下，CA 的企业管理软件在大型机和客户机/服务器领域也是领袖群伦。

王嘉廉在皇后大学毕业后就结束了学习生涯。毕业后，王嘉廉进入哥伦比亚大学电子研究实验室做了一位程序员新手，结果一下子就爱上了编程。他在好几家公司从事软件工作，其中包括标准数据公司。在那里，他和大学的伙伴 Russ Artzt 编写并出售用于 IBM 大型机的系统软件。两人经常拜访客户，倾听企业

信息系统管理员反映的各种问题，从而看到了商机，萌生了创业的念头。1976 年，他终于可以将想法付诸实践。一家瑞士的 CA 国际公司，正在寻求一家美国公司，代理销售 CA-EARL 大型机软件。后来 CA 找到了标准数据，双方成立了一个合资公司。几个月后，标准数据准备放弃软件业务。王嘉廉、Artzt 和另外两位朋友 Sedino、Habermass 就成立了 CA 国际公司，作为瑞士公司的子公司。在曼哈顿麦迪逊大街的一间办公室里，他们开始推销 CA-SORT。当时条件十分艰难，他们是用服务换回了上机机室和办公室。Sedino 为房东担任招待员，Artzt 和 Habermass 负责产品开发，王嘉廉则是销售和市场部的光杆司令，负责销售业务。开始没有任何业绩，王嘉廉就为大楼的休息厅铺地毯、在走廊中挂镜子，以抵消房租。这家位于纽约的软件公司从一开始可以说一文不名，但 1980 年 4 月，他们却以 280 万美元买下了瑞士的母公司，从此 CA 完全属于他们。也就是从这时开始，CA 实施了一系列的购并举措，以实现高速增长。

CA 产品

经典语录

我们并不比其他公司的人更聪明，但不同的是，我们节省了许多相互指责的时间。从错误中学到教训，不断向上成长。

深度评价

尽管人们常说计算机行业是年轻人拼搏的战场，但也有宝刀不老者，55 岁的王嘉廉就是例证。他领导的 CA 在国际软件业是知名的公司，他本人也因为是华人却在白人领域中闯出一番天地而更具传奇色彩。王嘉廉勤奋、刻苦、善于把握时机、勇往直前的精神，永远值得每一个渴望成功的人学习。

第 6 堂课

面向对象编程

（视频讲解：154 分钟）

最早的程序开发使用的是结构化程序设计语言，但是随着时间的流逝，软件的规模逐渐扩大，使用结构化语言出现各种弊端，导致无休止地拖延开发周期，产品的质量也不尽如人意。这一切都体现了结构化语言已经不再适合当前的软件开发。现在程序设计者们将另一种开发思想引入程序中，那就是面向对象的开发思想。

面向对象最关键的两个词汇是类与对象，实质上可以将类看作是对象的抽象，它定义了对象所具有的属性与方法。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，因此掌握类与对象是学习 Java 语言的基础，并且可以使程序员更好、更快捷地掌握 Java 编程思想与编程方式。本堂课详细地介绍了类的定义和对象的使用。

学习摘要：

- 如何定义类
- Java 包的使用
- 成员变量与成员方法
- 对象的创建与使用
- 类的静态成员变量和静态方法
- 类的构造方法

6.1 类 的 定 义

类是用来定义一组对象共同具有的状态和行为的模板。而对象是现实世界中个体或事物的抽象表示，并且封装了它们的属性和行为。例如，一个员工可以表示为一个对象，它有姓名、性别、年龄等属性，同时也具有吃、喝、走、跑、说话等行为。这些属性和行为都是“人类”所具有的，也就是说类声明了这些共同的特性，在使用对象（也就是类的实例）之前，必须定义该对象的类，知道了对象的类型，才能够访问它的属性和行为。

在 Java 语言中，对象的行为被定义成类的方法，对象的属性被定义为类成员变量。所以一个类包括对象的属性和行为，它由 class 关键字声明，其语法格式如下：

```
class claassName{
    成员变量
    ……
    成员方法
    ……
}
```

其中 className 是要定义的类的名称，在类体中可以定义多个成员变量和成员方法，在之前的实例中，经常使用的是 main()方法，这就是一种特殊的成员方法，因为它作为程序执行的入口方法用于执行程序。

6.2 包

一个 Java 项目会定义多个类，类多了，就会涉及重名的问题，也就是命名冲突。那么 JDK API 中提供成千上万具有各种功能的类是如何管理的呢？Java 中提供了一种管理类文件的机制，就是包（package）。

6.2.1 类包简介

Java 中每个接口或类都来自不同的包，不论是 Java API 中的类与接口还是自定义的类与接口都需要隶属某一个包，这个包相当于一个文件夹，包含了一些类和接口的文件。如果没有包的存在，管理程序中的类也是一件非常麻烦的事情。当程序只有一个类组成时（就像之前章节的所有实例），那么这并不会给程序带来什么影响，但是随着程序代码的增多，免不了会出现类同名的问题，而且多个类放在一个文件夹中也不好分类管理。

例如，在程序中定义一个用户登录的 Login 类，而因业务需要，还要定义一个管理员登录的 Login 类，这两个类所实现的业务完全不同，于是问题就产生了，编译器不会允许存在同名的类文件。解决这类问题的办法是将这两个类放置在不同的包中。

6.2.2 定义类的包

在 Eclipse 中创建包的步骤如下：

（1）在项目的 src 节点上单击鼠标右键，在弹出的快捷菜单中选择“新建”/“包”命令。

（2）在“新建 Java 包”对话框的“名称”文本框中输入新建的包名，如输入“com.lzw”，如图 6.1 所

示，单击“完成”按钮。

图 6.1　“新建 Java 包”对话框

另外，在 Eclipse 中创建类时，可以同时输入该类分配的包名，向导会自动创建相应的包。

在 Java 中包被设计为与文件系统结构相对应，例如，一个包名为 com.lzw，那么该包中的类位于 com 文件夹的 lzw 子文件夹中。没有定义包的类会被归纳到默认包中。在实际开发中，应该为所有类分配包名，这样会养成良好的编程习惯与代码风格。在类中定义包名的语法格式如下：

package 包名

参数：包名是要创建的 Java 包的名称。

在类中指定包名时必须将 package 语句放置在程序的第一行，它必须是文件中的第一行非注释代码。当使用 package 关键字为类分配包名之后，包名将会成为类名中的一部分，这时，类的全名就是包名加上类名。例如，在使用位于 com.animal 包中的 Dog.java 类时，在其他包使用该类的情况下，需要使用形如 com.animal.Dog 这样的全名。

注意：Java 的命名规则要求，包全部使用小写字母。

对于类名的冲突问题，也许有的读者会产生疑问，那么如此多的包也许会产生包名冲突现象，这个可能性是比较大的。为了避免这样的问题，在 Java 中定义包名时通常使用创建者的 Internet 域名的反序，由于 Internet 域名是独一无二的，所以包名自然不会发生冲突问题，而且在团队工作中往往是先定义好包的结构，才开始程序的开发。

例 6.01　创建包名为 com.lzw 的 Math.java 类，在主方法中输出字符串信息，该类并非 java.lang 包的 Math 类。实例代码如下：**（实例位置：光盘\mr\ym\6.01）**

```
package com.lzw;                                              // 指定包名
public class Math {
    public static void main(String[] args) {
        System.out.println(Math.class.getName());             // 输出类的全称
        System.out.println(java.lang.Math.class.getName());   // 输出另一个 Math 类的全称
        System.out.println("我不是 java.lang.Math 类，而是 com.lzw.Math 类");
    }
}
```

运行结果如图 6.2 所示。

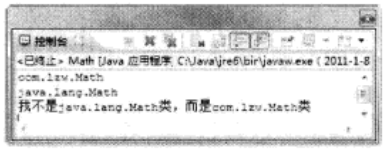


图 6.2　在程序中指定包名

📖 **说明：包名是类名的一部分，它们组合在一起就是这个类的完整名称。如果在一个类中同时引用了两个同名的类名，那么就要使用该类的完整名称，也就是指定其包名，就像本实例中那样。**

在本实例中，程序的第一行指定包名为 com.lzw，在 com.lzw 包中定义 Math 类，而 Java 也在 java.lang 包中定义了 Math 类，这两个类的名称相同，但是它们却没有冲突，因为它们处在不同的包中。

6.2.3 类包的导入

如果其他类中需要使用刚刚定义的 Math 类，那么如何告知编译器当前应该使用的是哪一个包中的 Math 类，是 java.lang.Math 类还是 com.lzw.Math 类。

java.lang 包是 java 默认导入的包，也就是不需要任何操作，java.lang 包中的所有类都可以直接使用。例如，Math.abs()将调用 java.lang 包中的 Math 类的 abs()方法，但是如果使用这种方式是无法调用 com.lzw.Math 类的方法的，除非当前类也在 com.lang 包中。如果在其他包中，就必须使用类的全名 com.lzw.Math，另外还可以使用 Java 的 import 语句导入指定的类包的单个或所有类，其语法格式如下：

```
import 包名.类名;
```

参数：包名是要导入到当前类的包的名称，类名是要导入的包名中包含的类名。例如，导入 com.lzw 包中的 Math 类的语法如下：

```
import com.lzw.Math;
```

执行这个语句之后，再次使用 Math 就不是 java.lang 包的 Math 类了，而是导入的 com.lzw.Math 类。

在使用 import 语句时，包名可以使用通配符“*”代表所有类，例如：

```
import com.lzw.*;
```

这样就在当前类导入了 com.lzw 包中的所有的类。读者回忆一下，在讲解数组时，使用过 Arrays 类，它是 java.util 包的类，相关的实例代码都使用 import java.util.Arrays 语句导入了 Arrays 类。

📢 **注意：当 import 导入了一个包中所有的类时，并不会导入相应子包中的类，如果用到这个包中的子类，需要再次对子包作单独导入。**

6.2.4 静态导入

import 关键字除了导入包之外，还可以导入静态成员，这是在 JDK 5.0 以上版本提供的新功能——“静态导入”，导入静态成员可以使程序员更为方便。import 导入静态成员的语法格式如下：

```
import static 静态成员
```

参数：静态成员是某个类的静态成员变量。例如，经常使用 System 类的 out 静态成员的 println()方法，如果能够直接使用 out 静态成员会更加方便，下面举例演示其实现方法。

例 6.02 在类中静态导入 System 类的 out 静态变量，然后以更简便的方式执行 println()方法。实例代码如下：**（实例位置：光盘\mr\ym\6.02）**

```
package com.lzw;
import static java.lang.System.out;                          // 导入静态成员变量
public class ImportTest {
    public static void main(String[] args) {
        out.println("静态导入 System 类的 out 静态成员变量");      // 直接使用静态成员
        out.println("可以在当前类直接使用导入的静态成员变量");
    }
}
```

运行结果如图 6.3 所示。

图 6.3　使用 import 关键字导入静态成员

在本实例中使用 import static 导入了 java.lang.System 类中的 out 静态成员变量，这时就可以在程序中直接引用这个静态成员变量，就好像它是在本类中定义的一样。

注意：import static 静态导入语句只能导入类的静态成员。

6.3　成员变量与成员方法

定义类之后要在类体中定义各种成员变量和成员方法来表示状态与动作行为，如名字、年龄以及走、跑、打架、打印、唱歌等，本节将向读者介绍成员变量和成员方法。

6.3.1　成员变量

成员变量是在类体中定义的变量，即全局变量，成员变量用于定义对象的状态。例如，Person 类有 name、age、sex 等成员变量分别表示姓名、年龄、性别等状态，具体代码如下：

```
public class Person {
    public String name;          // 姓名
    public int age;              // 年龄
    public String sex;           // 性别
}
```

成员变量必须直接定义在类体中，如果定义在其他方法或代码块中，就会成为局部变量，而不是成员变量。

成员变量是给对象使用的，每个对象被创建以后，都会有属于自己的属性，即成员变量。通过修改这些属性，从而改变对象的某个状态。如图 6.4 所示，不同的实例对象设置了不同的属性。

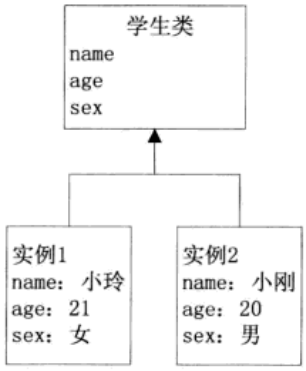


图 6.4　学生类的属性

6.3.2 成员方法

成员方法是对象行为的描述。每个方法都是由复合语句描述相应的行为，定义成员方法的语法格式如下：

```
[修饰符] 返回值类型 方法名([形参表]){
    …// 方法体
    return 返回值;
}
```

参数说明如下。

修饰符：可以是 public、private、protected、static 和 final 等。修饰符是可选的语法部分，有关权限修饰符将在第 9 堂课讲解。

返回值类型：如果方法需要返回值，必须在这里声明方法的返回值类型，可以是基本数据类型（int、short、double、boolean 等），也可以是对象类型（如数组、字符串等）。

方法名：这是必需的方法定义部分，程序将通过该名称调用这个方法。

形参表：这是可选部分，说明方法被调用时应该向它传递的数据。

形参表由 1 到多个形式参数构成，当方法有多个形式参数时，参数之间使用逗号“,”分割；如果方法没有参数，可以省略“形参表”部分。

例如：

```
public void saveBookInfo(String bookName, String publishing, String writer) {
    // 方法体
}
```

这段代码定义了 saveBookInfo()方法实现图书信息的保存，最简单的图书信息也要有书名、出版社、作者等信息，所以该方法的形参表定义了 bookName、publishing 和 writer 参数。

📖 **说明：成员变量和成员方法统称为类成员。**

6.3.3 方法的返回值

一个成员方法的参数可以是对象，也可以是基本数据类型的变量，同时成员方法存在有返回值和无返回任何值的情况。例如，计算应收款的 compute()方法，至少要返回应收款的金额，如图 6.5 所示。

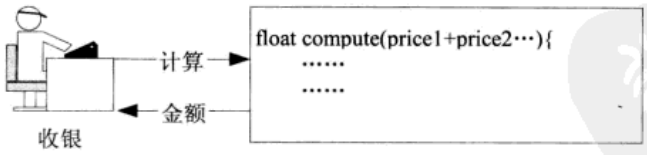


图 6.5 compute()方法返回 float 值

而 print()打印方法可以没有返回值，直接执行打印指令，控制打印机输出，如图 6.6 所示。当然也可以返回 boolean 值，反映是否打印完毕的状态。

声明方法时必须指定返回值类型，如果一个方法没有返回值，必须使用 void 关键字作为返回值类型。例如：

```
public void print() {
    // 方法体
}
```

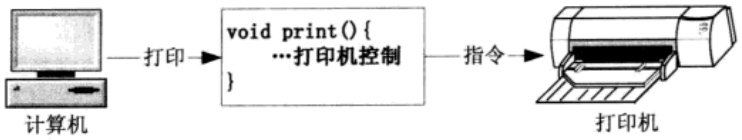


图 6.6　print()方法无返回值

如果声明方法时指定了方法的返回值类型，就必须在方法体中使用 return 语句返回相应类型的值，使用这个关键字后，方法的执行将被终止，语法如下：

return 返回值;

参数：返回值的类型必须和声明方法时的返回值类型兼容。

例如：

```
public float compute(){
    // 方法体
    return 35.50F;
}
```

计算金额的方法在声明时是 float 类型的返回值，那么 return 语句就必须返回 float 兼容类型的数值。学习基本数据类型的自动类型转换后，我们知道低于 float 级别的基本数据类型会自动转换为 float 类型。所以如果 return 语句返回 int、short 等类型的返回值，会自动转换为 float 类型，也就是说，这些基本数据类型和 float 类型是兼容的。

例如：

```
public float compute() {
    char sum = 'A';                    // 声明 char 类型的变量
    return sum;                        // 作为返回值，float 兼容 char 类型
}
```

这段代码在声明返回值类型为 float 的 compute()方法中，使用 return 语句返回了 char 类型的值。根据自动类型转换的原理，float 是兼容 char 类型的，所以这是可行的。但是高于 float 级别的数据类型必须强制类型转换为 float 类型，才能作为方法的返回值。

例如：

```
public float compute(){
    double sum=35.50;          // 声明 double 变量
    return (float) sum;        // 作为返回值，必须强制类型转换为 float
}
```

声明的 double 类型变量 sum 存储了最后经过计算的金额，如果把它作为方法的返回值，由于高于 float 类型，所以必须强制类型转换，否则编译时无法通过。

另外，在没有返回值，也就是返回值类型被声明为 void 的方法中，也可以使用不带返回值的 return 语句，它可以结束该方法的执行。

例如：

```
public void print(int pageNum) {
    if (pageNum > 10) {                    // 如果打印页数大于 10
        return;                            // 结束该方法
    }
    else {                                 // 否则
        System.out.println("开始打印……");    // 开始打印
    }
}
```

这个方法定义了 pageNum 参数作为打印的页数，在方法体中判断这个打印页数是否大于 10 页，如果大于，就调用空的 return 语句返回到方法的调用位置，也就是结束该方法，否则执行打印操作。

注意：如果方法声明了返回值类型，那么 return 语句就必须指定返回值，而不能使用空的 return 语句，只有声明为 void 类型，即没有返回值的方法，才能使用空的 return 语句作为方法结束语句。

6.4 对象的创建与使用

在学习了类的定义、成员变量与成员方法之后，必须创建该类的实例对象，才能够访问成员变量与成员方法。本节将介绍创建对象、访问成员变量和方法以及销毁对象。

6.4.1 创建类的对象

一个汽车的设计图纸永远不能用来驾驶出行，必须用它生产出真正的汽车才能使用。这个汽车的设计图纸可以比作之前设计的类，而真正的汽车（如红色轿车）可以比作是对象，只有对象才能被程序使用，而且同一类型的对象还有不同的属性，如颜色、高度、速度等，另外，对象还可以执行类设计的方法，即对象的行为，如开车、换档、加油、转向和刹车等。

创建类的实例对象需要使用 new 语句，声明并创建对象的语法格式如下：

```
类名 对象名=new 类构造方法();
```

参数：类名是要创建实例对象的类的名称，对象名可以看作是一个变量，这个变量名就是创建的实例对象的名称，构造方法是类创建对象时必须执行的方法，用于构造一个新的对象并初始化对象属性。

例如：

```
String name = new String("一个新的字符串对象");
```

这句代码声明了字符串变量 name，并且使用 new 语句创建了字符串对象，String 是一个特殊的类，它可以用双引号的字面量创建对象，也可以使用 new 语句创建，但是自己写的类只能使用 new 语句创建。

6.4.2 访问对象的属性

创建了对象以后，可以访问对象的属性，即成员变量。访问对象属性需要使用“.”操作符，语法如下：

```
对象名.属性
```

在对象名后面使用“.”操作符连接对象的属性，即类的成员变量，就可以引用相应的属性值，或者为某个属性赋值。

例如：

```
String carColor = whiteCar.color;
whiteCar.color = "白色";
```

第一条语句把 whiteCar 对象的颜色赋值给了 carColor 变量，第二条语句把 whiteCar 对象的 color 颜色属性赋值为“白色”。

6.4.3 执行对象的行为

对象的行为也就是类的成员方法，通常说调用或执行对象的某个方法。它的语法格式和访问对象的属性类似：

对象名.成员方法名();

调用成员方法时，如果该方法定义了参数，就必须在括号内提供相同类型的参数值，而且顺序要和定义该方法参数时的顺序相同。

例如：

```
whiteCar.gear(3);                    // 调用 gear()方法并传递参数值 3
whiteCar.drive();                    // 调用 drive()方法，无参数值传递
```

这段代码分别调用了 gear()方法和 drive()方法，并且 gear()方法还传递了 int 类型的参数值 3。该方法会根据这个参数值做相应的业务处理。

例 6.03　定义一个 Car 类作为汽车类，并为它声明颜色、速度、档位等成员变量，还有开车、换档等成员方法，然后创建该类的实例对象，并修改颜色属性、执行换档与开车方法。**（实例位置：光盘\mr\ym\6.03）**

（1）首先创建 Car 类，注意这个类不是主类，所以使用 Eclipse 开发工具创建的话不要选择创建 main()方法，该类的关键代码如下：

```
package com.lzw;                          // 定义类所在的包
/**
 * ……省略部分注释代码
 * 这是 Car 汽车类
 * @author Li Zhong Wei
 */
public class Car {
    public String color = "黑色";          // 颜色
    public int speed = 1;                 // 速度
    public int level;                     // 档位
    /**
     * 换档方法
     * @param num 档位参数
     */
    public void gear(int num) {
        level = num;                      // 换档
        speed = 10 * level * level;       // 变速
    }
    /**
     * 开车方法
     */
    public void drive() {
        System.out.println("我是一辆" + color + "的轿车。");
        System.out.println("现在以时速" + speed + "公里每小时的速度行驶中。");
        System.out.println("车档位是：" + level + "档。");
    }
}
```

（2）编写该实例的主类 Program。在使用 Eclipse 创建该类的向导中，选择创建 main()方法，因为要通过它执行程序，调用 Car 类做实例演示，该类的关键代码如下：

```
package com.lzw;                                        // 定义类所在的包
/**
 * ……省略部分注释代码
 * 这是主类
 * @author Li Zhong Wei
 */
public class Program {
```

```
    public static void main(String[] args) {
        Car whiteCar = new Car();                    // 创建 Car 类的实例对象
        whiteCar.color = "白色";                      // 修改该实例中 color 属性
        whiteCar.gear(3);                            // 调用 gear()方法修改档位和车速
        whiteCar.drive();                            // 执行开车方法
    }
}
```

运行结果如图 6.7 所示。

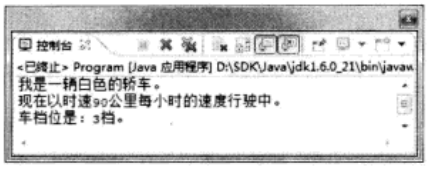


图 6.7　实例运行结果

6.4.4　对象的销毁

大多数面向对象的编程语言，要求显式地销毁某个对象，这点就没有 Java 做得好，Java 语言提供了垃圾回收机制，对于不再使用的对象会自动销毁，也可以在程序中显式地为某个对象赋值 null 值，使对象不再被使用。

例如：

```
whiteCar = null;
```

垃圾回收器会找到并销毁它，释放该对象所占用的资源。

6.5　类成员与类方法

成员变量与成员方法是每个对象的属性与行为的抽象，如果把它们定义成静态的，就会在内存中为它们分配固定的内存空间，同时它们也会成为类的成员，而非对象的成员，本节将向读者介绍类变量与类方法。

6.5.1　类变量

类变量也称为静态成员变量，它的语法在成员变量的基础上添加了 static 关键字，使成员变量成为静态的。静态的成员变量不是分配给每个实例变量的，而是属于类的变量，它在内存中是唯一的，可以直接使用“类名.成员变量”的格式去访问，它在内存中的位置是固定的，是该类的所有实例对象所共享的存储单元。

例如，中国人是一个类，其中 country 属性代表国家，把这个属性声明为静态的成员变量，并赋值为“中国”，它就是静态成员变量，即类的变量。具体代码如下：

```
public class Chinese {
    public static String country="中国";          // 国家
    public String name;                           // 姓名
    public int age;                               // 年龄
    public String sex;                            // 性别
}
```

所有该类的实例都会共享该变量，并且只占用一份内存空间，如图 6.8 所示。

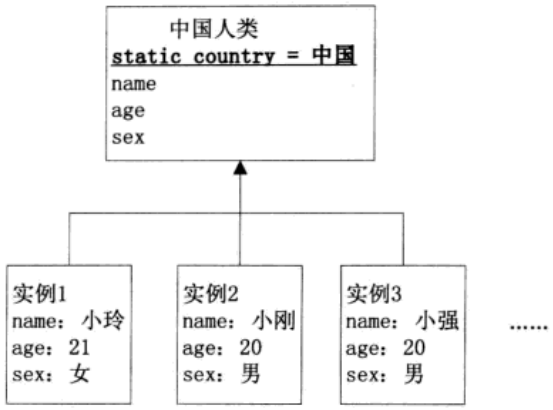


图 6.8　类的变量

如果修改了类变量 country 的值，例如：

```
Chinese.country = "中华人民共和国";
```

那么该类所有实例对象的 country 属性都会改变。如果使用成员变量，那么每个中国人的实例对象都会分配一个 country 属性，有多少个中国人的实例对象，就会创建多少份 country 属性的内存空间。

注意：声明为静态的变量必须是成员变量，而不能是局部变量，在方法体内或任何代码块中声明的变量不能使用 static 关键字。

6.5.2　类方法

成员变量与成员方法必须创建对象，通过对象才能访问。但有时也希望能够不必创建对象直接调用类的某个方法，也就是类方法。

类方法由 static 关键字定义，它也是静态成员方法，例如：

```
public static void print(String mess){
    // 方法体
}
```

静态成员方法与静态成员变量一样，都可以使用类名直接访问，同时也可以使用类的实例对象进行访问。

例 6.04　定义一个 Car 类作为汽车类，并为它声明产地、使用年限等成员变量，还有成员方法 printYield Info()，它用于输出本款轿车的出厂信息。然后创建该类的 3 个实例对象，并在修改产地静态成员变量前后，分别输出每个实例对象的产地信息。**（实例位置：光盘\mr\ym\6.04）**

（1）首先创建 Car 类，注意这个类不是主类，所以使用 Eclipse 开发工具创建的话不要选择创建 main() 方法，该类的关键代码如下：

```
package com.lzw;                          // 定义类所在的包
/**
 * ……省略部分注释代码
 * 这是 Car 汽车类
 * @author Li Zhong Wei
 */
public class Car {
```

```
    public static String yieldAddress = "长春"; // 产地
    public static int life = 15;                    // 使用年限
    /**
     * 打印出厂信息
     */
    public static void printYieldInfo() {
        System.out.println("本款轿车产自：" + yieldAddress);
        System.out.println("使用年限为：" + life + "年");
    }
}
```

（2）编写该实例的主类 Program。在使用 Eclipse 创建该类的向导中，选择创建 main()方法，因为要通过该方法执行程序，调用 Car 类做实例演示，该类的关键代码如下：

```
package com.lzw;                                    // 定义类所在的包
/**
 * ……省略部分注释代码
 * 这是主类
 * @author Li Zhong Wei
 */
public class Program {
    public static void main(String[] args) {
        Car newCar1 = new Car();                    // 创建 Car 类的实例对象
        Car newCar2 = new Car();                    // 创建 Car 类的实例对象
        Car newCar3 = new Car();                    // 创建 Car 类的实例对象
        Car.printYieldInfo();                       // 输出这款轿车的出厂信息
        System.out.println("========================");
        System.out.println("newCart1 的产地是："+newCar1.yieldAddress);
        System.out.println("newCart2 的产地是："+newCar2.yieldAddress);
        System.out.println("newCart3 的产地是："+newCar3.yieldAddress);
        newCar1.yieldAddress = "吉林省长春市";
        System.out.println("========================");
        System.out.println("修改类的 yieldAddress 变量之后");
        System.out.println("newCart1 的产地是："+newCar1.yieldAddress);
        System.out.println("newCart2 的产地是："+newCar2.yieldAddress);
        System.out.println("newCart3 的产地是："+newCar3.yieldAddress);
        System.out.println("========================");
    }
}
```

运行结果如图 6.9 所示。

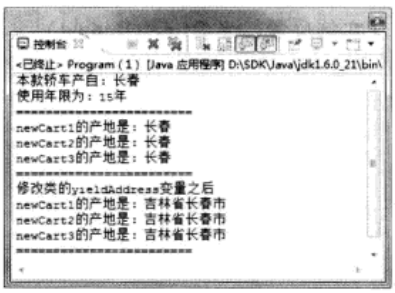


图 6.9　实例运行结果

注意：静态成员属于类的成员，在使用方式上应该使用类名做前缀访问静态成员，而不是使用对象名。上面的实例为了演示静态成员才使用了对象名，在 Eclipse 的 Java 编辑器中会提示警告信息。另外，静态成员变量和静态成员方法不能够被非静态方法访问，反之，非静态方法可以访问静态成员。

说明：现在读者应该明白 main()主方法为什么要声明为静态了，它必须能被 Java 虚拟机直接访问，而不用创建对象。

6.6　照猫画虎——基本功训练

6.6.1　基本功训练 1——用构造方法完成初始化

视频讲解：光盘\mr\lx\06\用构造方法完成初始化.exe

实例位置：光盘\mr\ym\6.05\src

定义 Student 类，在类中定义姓名、年龄和性别 3 个成员变量，再定义 3 个不同参数的构造方法，其中有一个是无参数的默认的构造方法，然后定义 sayMyInfo()方法输出自己的信息，实例代码如下：

```
package com.lzw;                                    // 声明类所在的包
/**
 * ……省略部分注释
 *  @author Li Zhong Wei
 */
public class Student {
    public String name;                             // 姓名
    public String sex;                              // 性别
    public int age;                                 // 年龄
    /**
     * 默认构造方法
     */
    public Student() {
        name = "未登记";
        sex = "未登记生";
        age = 0;
    }
    /**
     * 带姓名参数的构造方法
     * @param name 学生的姓名
     */
    public Student(String name) {
        this.name = name;                           // this 关键字防止参数 name 覆盖成员变量
    }
    /**
     * 初始化所有成员变量的构造方法
     * @param name 学生的姓名
     * @param sex 学生的性别
     * @param age 学生的年龄
```

```
     */
    public Student(String name, String sex, int age) {
        this(name);                                          // 调用另一个构造方法完成部分初始化
        this.sex = sex;
        this.age = age;
    }
    /**
     * 输出学生自己的信息的方法
     */
    public void sayMyInfo() {
        System.out.println("我的名字是：" + name);
        System.out.println("我是：" + sex + "生");
        System.out.println("我" + age + "岁了");
        System.out.println("========================");
    }
    public static void main(String[] args) {
        Student student1 = new Student();
        Student student2 = new Student("张三");
        Student student3 = new Student("李四", "男", 14);
        student1.sayMyInfo();
        student2.sayMyInfo();
        student3.sayMyInfo();
    }
}
```

运行结果如图 6.10 所示。

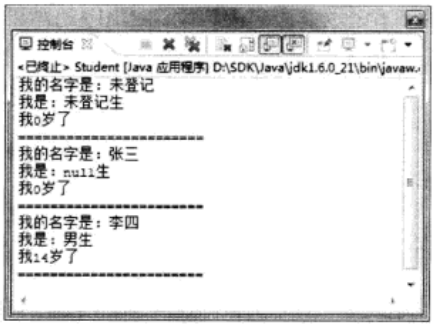


图 6.10　实例运行结果

本实例定义了两个参数列表不同的构造方法，还有一个无参数的默认构造方法。实例中分别使用这 3 个构造方法创建了对象，然后调用了每个对象的 sayMyInfo()方法输出自己的信息。从运行结果可以看出，不同的构造方法完成了不同程度的初始化，在实际程序开发中，可以利用构造方法完成更加灵活的初始化操作。如加载数据库驱动类、根据构造参数连接数据库等。

你问我答：什么是构造方法？

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，其功能主要是完成对象的初始化，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：

（1）声明构造方法没有返回值类型。

（2）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()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形式如下：

```
public 类名([参数列表]){
    //…构造方法体
}
```

参数说明如下。

public：构造方法修饰符。

类名：构造方法的名称，它必须和类名相同。

参数列表：可选参数，可以定义多个参数为构造方法传参，也可以不定义参数，使用没有任何参数的默认构造方法。

说明：如果类中没有明确定义任何构造方法，则编译器会自动创建一个不带参数的默认构造方法。

在构造方法中可以完成变量赋值、连接数据库等初始化工作，这样当实例化一个本类的对象时，相应的初始化工作就自动完成了。

和成员方法一样，构造方法也可以重载，定义参数列表不同的多个构造方法可以创建并初始化不同的对象。例如，Student 类的构造方法可以根据传递的姓名、性别、年龄等不同参数的值，来创建不同的学生对象，这些参数在程序中可能来源于现实世界，类似招生人员登记学生信息时，招生人员就是构造方法，每个学生就是一个对象，而它们的信息就是构造方法的参数，如图 6.11 所示。甚至可以为 Student 类添加指纹、血型等属性来进一步区别每个学生对象，但是那基本上超出了程序的业务范围。所以，在定义类时，要选择合适的属性和行为进行定义。

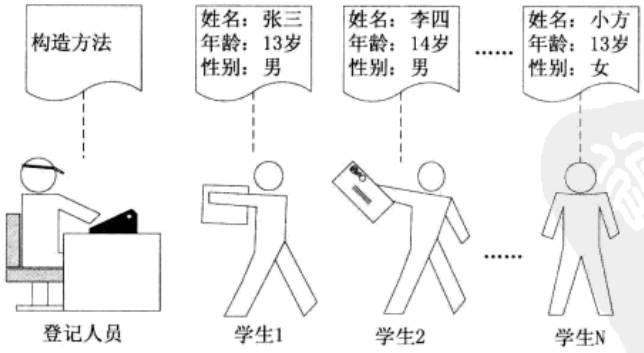


图 6.11　构造方法

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。

照猫画虎：修改实例，在每个构造方法中调用 sayMyInfo()方法，实现创建对象时自动输出信息。（20分）（实例位置：光盘\mr\ym\6.05\zmhh）

6.6.2 基本功训练 2——用静态代码块初始化

视频讲解：光盘\mr\lx\06\用静态代码块初始化.exe

实例位置：光盘\mr\ym\6.06\src

定义 StaticCode 类，在类中定义静态成员变量 count，该变量是对象计数器，用于记录该类创建了多少个对象。另外，在该类中定义静态代码块，初始化 count 变量的同时输出提示信息。再定义一个 StaticDemo 类，该类的静态代码块输出提示信息，然后在主方法中创建 3 个 StaticCode 类的实例对象，在控制台查看 StaticCode 类的静态代码块被执行几次。

（1）创建 StaticCode 类，关键代码如下：

```
package com.lzw;
/**
 * 类注释信息与实例相同
 * @author Li Zhong Wei
 */
public class StaticCode {
    static int count = 0;   // 对象计数器
    static {                // 静态代码块
        count = 0;
        System.out.println("StaticCode 类被加载，类属性初始化完毕");
    }
    /**
     * 默认的构造方法
     */
    public StaticCode() {
        count++;
        System.out.println("我是 StaticCode 类的第" + count + "个对象");
    }
}
```

（2）创建 StaticDemo 类，该类用于程序演示，关键代码如下：

```
package com.lzw;
public class StaticDemo {
    static {
        System.out.println("StaticDemo 类被加载，我是主类");
    }
    public static void main(String[] args) {
        System.out.println("主方法被执行");
        StaticCode codeBlock1 = new StaticCode();
        StaticCode codeBlock2 = new StaticCode();
        StaticCode codeBlock3 = new StaticCode();
    }
}
```

运行结果如图 6.12 所示。

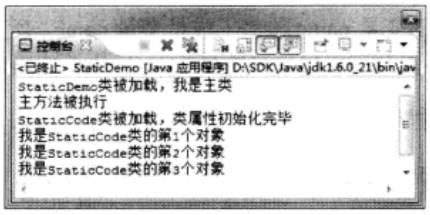


图 6.12　实例运行结果

你问我答：什么是静态代码块？

一个类可以创建不包含在任何方法体中的静态代码块，这个代码块是使用 static 关键字定义的一个复合语句，静态代码块在类被加载时，执行且仅执行一次，多用于类变量的初始化工作，其语法格式如下：

```
static {
    // 程序代码
}
```

本实例中 StaticDemo 类的静态代码块，在没有创建对象的情况下，由于虚拟机（Java 运行环境）加载了该类，执行 main()方法，所以静态代码块被执行。同时 main()方法中虽然定义了 3 个 StaticCode 类的实例对象，但是静态代码块仅执行了一次，尽管构造方法被执行了 3 次。

照猫画虎：把主方法中的代码移动到静态代码块中执行。（20 分）（实例位置：光盘\mr\ym\6.06\zmhh）

6.6.3　基本功训练 3——this 关键字的用途

视频讲解：光盘\mr\lx\06\this 关键字的用途.exe

实例位置：光盘\mr\ym\6.07\src

本实例用于演示 this 关键字的用途，创建了 Var 类，在类中定义了成员变量 name，然后在 sayName()方法中再定义局部变量 name，并输出 name 变量值。而在主方法中创建 Var 类的实例对象并调用 sayName()方法，看程序输出时是引用的成员变量还是局部变量，实例代码如下：

```
public class Var {
    String name="我的名字是 Java，我是 Var 类的成员变量";        // 成员变量
    public void sayName() {
        String name="我是 Java，我是 sayName()方法的局部变量"; // 局部变量
        System.out.println(name);                              // 在方法体中局部变量覆盖了成员变量
    }
    public static void main(String[] args) {
        Var var=new Var();                                     // 创建 Var 类实例对象
        var.sayName();                                         // 调用实例对象的 sayName()方法
    }
}
```

运行结果如图 6.13 所示。

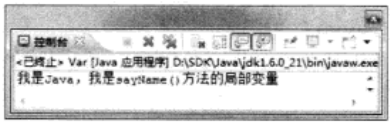


图 6.13　局部变量覆盖了成员变量

本实例中 sayName()方法的局部变量 name 覆盖了成员变量 name，所以在输出变量值时，输出的是局部变量而不是成员变量。如果在输出语句中引用成员变量 name，就必须添加 this 关键字。

你问我答：this 关键字有什么用？

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，也就是说，方法体中的局部变量覆盖了成员变量。这时可以使用 this 关键字来引用成员变量，this 关键字代表当前类的实例对象，它可以应用在类中定义的所有成员变量与成员方法，该关键字的语法如下：

this.成员

参数说明：成员可以是成员变量，也可以是成员方法。

注意：不能在静态上下文和静态代码块中使用 this 关键字，编译器不允许这样做。

照猫画虎：修改 sayName()方法中的输出语句，通过 this.name 输出成员变量的值。(20 分)(实例位置：光盘\mr\ym\6.07\zmhh)

6.6.4 基本功训练 4——方法参数传递

视频讲解：光盘\mr\lx\06\方法参数传递.exe

实例位置：光盘\mr\ym\6.08\src

在定义方法时，可以为方法指定参数列表，方法根据参数列表与方法的调用者进行数据交互，参数列表中的参数类型可以是 int、short、double 等基本数据类型，也可以是对象类型，但是这两种参数类型的值传递方式并不一样。如果传递的参数是基本数据类型，该方法会直接获得指定类型的数值；如果传递的参数是对象类型，那么，该方法获得的只是一个对象地址的引用。通过这个引用可以访问和操作对象属性与行为，但是对象的实体并没有传递给该方法，它还在内存的某个区域，在这种情况下，当方法改变对象的属性时，会直接改变对象实体的数据，而基本类型的数据是值传递的，改变它不会影响方法调用者传递的变量。下面以实例演示为方法传递对象参数和传递基本数据类型参数。

实现过程如下：

（1）编写 ObjectArg 类，在该类中定义几个成员变量，这个类的对象将用于演示方法传参，代码如下：

```
public class ObjectArg {
    public String name="";    // 姓名
    public int age=0;         // 年龄
    public String sex="";     // 性别
}
```

（2）编写 ArgDemo 类，分别定义测试基本数据类型传参和对象类型传参的方法，并输出它们修改参数的结果，代码如下：

```
public class ArgDemo {
    static int intArg = 10;                                    // int 基础类型的测试变量
    static ObjectArg objectArg = new ObjectArg();              // 对象类型的测试变量
    public static void main(String[] args) {
        changeIntArg(intArg);                                  // 测试 int 基本类型传参
        System.out.println("传递给方法的 int 类型参数："+intArg);
        System.out.println("intArg=" + intArg);                // 输出传递给方法的变量值
        objectArg.age = 29;                                    // 初始化对象数据
```

```
        objectArg.name = "李经理";
        objectArg.sex = "男";
        changeOjbectArg(objectArg);                    // 测试对象传参
        System.out.println("传递给方法的对象：");
        printObjInfo(objectArg);                       // 输出传递给方法的对象信息
    }
    /**
     * 测试基本数据类型传参的方法
     * @param arg int 类型参数
     */
    public static void changeIntArg(int arg) {
        System.out.println("==========实参未改变============= ");
        System.out.println("changeIntArg()方法开始执行");
        System.out.println("int 类型参数 arg=" + arg);
        System.out.println("将参数值除以 2");
        arg = arg / 2;                                 // 改变参数值
        System.out.println("arg=" + arg);
    }
    /**
     * 测试对象类型传参的方法
     * @param arg  对象参数
     */
    public static void changeOjbectArg(ObjectArg arg){
        System.out.println("========实参被改变============== ");
        System.out.println("changeOjbectArg()方法开始执行");
        System.out.println("输出对象参数：");
        printObjInfo(arg);                             // 输出对象参数信息
        arg.name="王经理";                              // 修改对象参数信息
        arg.sex="女";
        arg.age=20;
        System.out.println("修改后的对象参数：");
        printObjInfo(arg);                             // 输出修改后的对象参数信息
    }
    /**
     * 输出对象信息的方法
     * @param arg  对象参数
     */
    private static void printObjInfo(ObjectArg arg) {
        System.out.println("\t 名字："+arg.name);
        System.out.println("\t 性别："+arg.sex);
        System.out.println("\t 年龄："+arg.age);
    }
}
```

实例运行结果如图 6.14 所示。

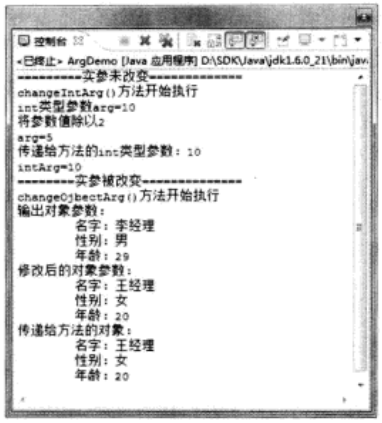


图 6.14　实例运行结果

照猫画虎：修改实例，为 ObjectArg 类定义 printObjInfo()方法，在 ArgDemo 类的 changeObjectArg()方法中替换原有的输出代码。（20 分）（实例位置：光盘\mr\ym\6.08\zmhh）

6.6.5　基本功训练 5——方法的重载

视频讲解：光盘\mr\lx\06\方法的重载.exe

实例位置：光盘\mr\ym\6.09\src

方法的重载就是在同一个类里定义了多个名称相同，但是参数类型和数量以及顺序不同的方法。为了更好地理解重载，这里将编写一个方法重载的实例。

实现过程如下：

（1）在项目中创建 OverLoadTest 类，在类中定义一个成员变量 sum，关键代码如下：

```
package com.lzw;                                    // 定义类所在的包
/**
 * ……省略部分注释代码
 * @author Li Zhong Wei
 */
public class OverLoadTest {
    static int sum = 0;                             // 成员变量 sum 用于累加计数
```

（2）编写 add()方法的第一个重载形式，它接收两个 int 类型的参数，并把两个参数相加后返回它们的和，方法的关键代码如下：

```
    /**
     * 把两个 int 类型整数相加的方法
     * @param numA 第一个 int 整数
     * @param numB 第二个 int 整数
     * @return 两数之和，类型为 int
     */
    public static int add(int numA, int numB) {
        return numA + numB;
    }
```

（3）编写第二个重载格式的方法，方法的两个参数是 double 类型的。同样，在方法体中计算它们的和，然后把结果作为方法的返回值，关键代码如下：

```
/**
 * 把两个 double 类型实数相加的方法
 * @param numA 第一个 double 实数
 * @param numB 第二个 double 实数
 * @return 两数之和，类型为 double
 */
public static double add(double numA, double numB) {
    return numA + numB;
}
```

（4）编写第三个格式的方法，这个方法包含一个 int 类型的参数，在方法体中参数值将被累加到 sum 成员变量中，并把 sum 的值作为方法的返回值，关键代码如下：

```
/**
 * 累加 int 类型整数的方法，该方法将参数值累加到 sum 成员变量中
 * @param numA int 整型参数
 * @return
 */
public static int add(int numA) {
    sum += numA;
    return sum;
}
```

（5）编写第四个重载的 add()方法，这个方法定义一个 int 类型的参数和一个 double 类型的参数，其中 int 类型的参数是方法的第一个参数，方法体把两个参数值的和作为返回值，关键代码如下：

```
/**
 * 把 int 类型整数和 double 类型实数相加的方法
 * @param numA int 整数
 * @param numB double 实数
 * @return 整数和实数的和，类型为 double
 */
public static double add(int numA, double numB) {
    return numA + numB;
}
```

（6）编写第五个重载的 add()方法，在这个方法的声明中分别定义了 double 类型和 int 类型的两个参数，其中 double 类型的参数作为第一个方法参数。在方法体中完成两个参数和的计算，并把计算结果作为方法的返回值，关键代码如下：

```
/**
 * 把 double 类型实数和 int 类型整数相加的方法
 * @param numA double 实数
 * @param numB int 整数
 * @return 整数和实数的和，类型为 double
 */
public static double add(double numA, int numB) {
    return numA + numB;
}
```

（7）在主方法中分别调用这些重载方法，并输出它们的返回值。这些方法的调用将会根据参数类型与顺序被自动定位，不会发生混淆，关键代码如下：

```
public static void main(String args[]) {
    System.out.println("调用 add(int, int)方法：" + add(1, 2));
    System.out.println("调用 add(double, double)方法：" + add(2.1, 3.3));
    System.out.println("调用 add(int)方法：" + add(1));
```

```
        System.out.println("调用 add(int, double)方法：" + add(4, 5.89));
        System.out.println("调用 add(double, int)方法：" + add(5.89, 4));
    }
}
```

在 Eclipse 中运行本实例，运行结果如图 6.15 所示。

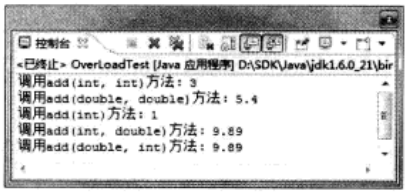


图 6.15　方法的重载

在本实例中，分别定义了 5 个方法。在这 5 个方法中，前两个方法的参数类型不同，所以这两个方法构成了重载关系。前两个方法与第三个方法比较时，第三个方法的参数个数少于前两个方法，所以这 3 个方法也是重载方法。最后两个方法比较时，发现除了参数的出现顺序不同之外，其他都相同，这样可以根据这个区别将两个方法构成重载关系。

本实例所有重载方法都在 main()方法中调用，如果读者在 main()方法中无法区分调用的是哪个重载方法，可以借助 Eclipse 来识别。细心的读者会发现，本实例的每个方法都添加了 JavaDoc 注释，而在第 3 堂课就介绍过：“添加了文档注释的成员，会在 Eclipse 的 JavaDoc 视图中显示相应的说明，另外在编辑器中，把鼠标停留在该方法或引用该方法的位置上，也会出现悬浮提示信息”。

既然每个方法的 JavaDoc 注释都不同，读者可以将鼠标放在 main()方法调用不同 add()方法的位置上，查看该方法的说明信息。如图 6.16 所示，是鼠标放在 main()方法第一条语句中 add()方法上的提示信息，根据这个提示，读者找到匹配的方法就可以了。

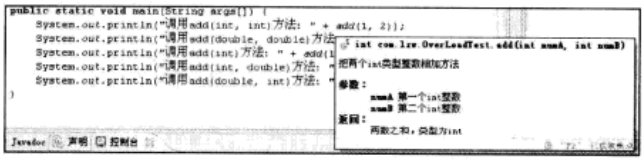


图 6.16　在 Eclipse 编辑器中显示方法的 JavaDoc 注释信息

你问我答：如何判断方法重构？

编译器利用方法名、参数类型和参数的个数以及参数的声明顺序来确定方法是否唯一。方法的重载使得方法以统一的名称被管理，使程序代码更具有条理，图 6.17 表明了可以构成重载的条件。

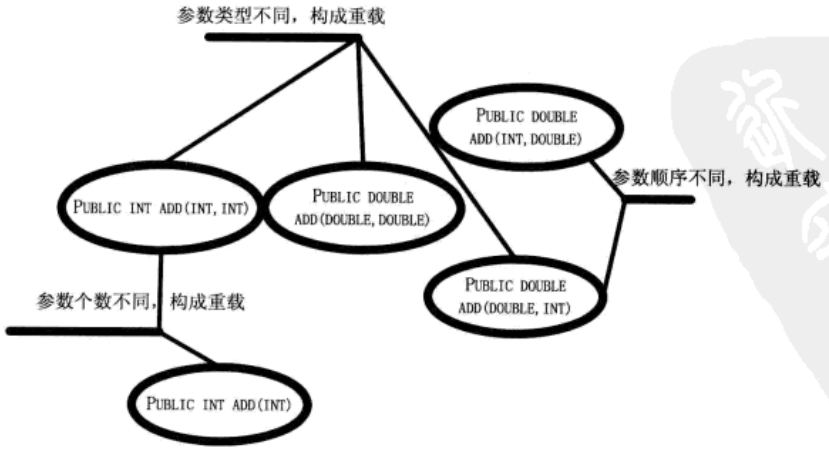


图 6.17　构成方法重载的条件

注意：虽然在方法重载中可以使两个方法的返回值不同，但只有返回值不同并不是区分方法重载的依据，必须通过参数的顺序、个数以及参数的类型来验证两个方法的重载。

照猫画虎：定义两个同名的重载方法，分别接收字符串与字符数组作为参数，在方法体中实现字符串的输出。（20 分）（实例位置：光盘\mr\ym\6.09\zmhh）

照猫画虎栏目分数统计：

照猫画虎题目	1	2	3	4	5	总分数	
分数							

6.7　情景应用——拓展与实践

6.7.1　情景应用 1——整数进制转换器

视频讲解：光盘\mr\lx\06\整数进制转换器.exe

实例位置：光盘\mr\ym\6.10\src

由于计算机的特殊结构，其内部使用二进制数据。为了方便记忆与阅读，又定义了八进制和十六进制格式来表示数据。一个八进制数位可以表示三位二进制数，一个十六进制数位可以表示四位二进制数。而对于普通人而言，使用十进制更加容易阅读。本实例将通过包装类的 Integer 类实现一个简单的进制转换器，完成十进制到其他不同进制的转换，实例的运行结果如图 6.18 所示。

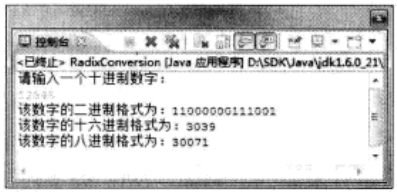


图 6.18　实例运行结果

注意：需要先在文本框中输入数字才能进行转换，本实例并未校验空输入的情况。

实现过程如下：

（1）创建 Java 项目，在项目中创建 RadixConversion 类。

（2）在 RadixConversion 类的主方法中创建输入扫描器，并接收用户输入的一个十进制整数，然后通过 Integer 类的静态方法把整数转换为其他进制的数字，关键代码如下：

```
import java.util.Scanner;
public class RadixConversion {
    public static void main(String[] args) {
        System.out.println("请输入一个十进制数字：");
        Scanner scan = new Scanner(System.in);                    // 创建输入扫描器
        int num = scan.nextInt();                                 // 读取用户输入的整数
        String value = Integer.toBinaryString(num);               // 把整数转换为二进制格式的字符串
        System.out.println("该数字的二进制格式为：" + value);
        value = Integer.toHexString(num);                         // 把整数转换为十六进制格式的字符串
        System.out.println("该数字的十六进制格式为：" + value);
        value = Integer.toOctalString(num);                       // 把整数转换为八进制格式的字符串
        System.out.println("该数字的八进制格式为：" + value);
    }
}
```

你问我答：Integer 类有什么用？

Integer 类设计的初衷是为了在基本类型 int 和引用类型之间建立一座桥梁。然而，类库的设计者发现，可以将很多有用的方法也放在该类中。本实例中使用到其定义的进制转换方法来实现进制转换，使用到的方法如表 6.1 所示。

表 6.1　Integer 类常用方法

方 法 名	作　用
toBinaryString(int i)	返回指定数字 i 的二进制表示形式
toOctalString(int i)	返回指定数字 i 的八进制表示形式
toHexString(int i)	返回指定数字 i 的十六进制表示形式
parseInt(String s)	解析字符串为 int 类型的值，字符串必须是仅由数字组成的

DIY：把用户在控制台输入的由数字组成的字符串转换为整数类型。（20 分）（实例位置：光盘\mr\ym\6.10\diy）

6.7.2　情景应用 2——获取数字的取值范围

视频讲解：光盘\mr\lx\06\获取数字的取值范围.exe

实例位置：光盘\mr\ym\6.11\src

Java 是一种强类型语言，每当定义变量时，都需要先指明其类型。对于数字基本类型而言，其取值范围是有限制的。例如，byte 类型的范围是从-128～127，如果将 128 赋值给一个 byte 类型的数据就会报错。本实例将通过各种数据的包装类获取指定类型数据的最大值和最小值，并将其输出到控制台中，实例的运行结果如图 6.19 所示。

图 6.19　各种数据类型的取值范围

实现过程如下：

（1）创建 Java 项目，在项目中创建 NumberLimitation 类。

（2）编写 NumberLimitation 类的主方法，在该类中定义输出各种基本数据类型取值范围的方法，这些方法可以通过各种包装类的属性直接获取类型的取值范围，然后在主方法中调用这些方法，关键代码如下：

```
public class NumberLimitation {
    public static void main(String[] args) {
        NumberLimitation nl = new NumberLimitation();
        nl.do_bytePrint();                  // 输出字节类型的取值范围
        nl.do_doublePrint();                // 输出双精度类型的取值范围
        nl.do_floatPrint();                 // 输出单精度类型的取值范围
        nl.do_intPrint();                   // 输出整数类型的取值范围
        nl.do_longPrint();                  // 输出长整数类型的取值范围
        nl.do_shortPrint();                 // 输出短整数类型的取值范围
    }
    protected void do_bytePrint() {         // 输出字节类型取值范围的方法
        System.out.println("byte 类型的最大值：" + Byte.MAX_VALUE);
```

```
        System.out.println("byte 类型的最小值：" + Byte.MIN_VALUE);
    }
    protected void do_shortPrint() {                    // 输出短整型类型的取值范围
        System.out.println("short 类型的最大值：" + Short.MAX_VALUE);
        System.out.println("short 类型的最小值：" + Short.MIN_VALUE);
    }
    protected void do_intPrint() {                      // 输出整数类型的取值范围
        System.out.println("int 类型的最大值：" + Integer.MAX_VALUE);
        System.out.println("int 类型的最小值：" + Integer.MIN_VALUE);
    }
    protected void do_longPrint() {                     // 输出长整数类型的取值范围
        System.out.println("long 类型的最大值：" + Long.MAX_VALUE);
        System.out.println("long 类型的最小值：" + Long.MIN_VALUE);
    }
    protected void do_floatPrint() {                    // 输出单精度类型的取值范围
        System.out.println("float 类型的最大值：" + Float.MAX_VALUE);
        System.out.println("float 类型的最小值：" + Float.MIN_VALUE);
    }
    protected void do_doublePrint() {                   // 输出双精度类型的取值范围
        System.out.println("double 类型的最大值：" + Double.MAX_VALUE);
        System.out.println("double 类型的最小值：" + Double.MIN_VALUE);
    }
}
```

你问我答：什么是包装类？

为了方便基本类型和对象（引用）类型之间的转换，Java 为每种基本类型都提供了对应的包装类。现说明如下：byte 的包装类是 Byte、short 的包装类是 Short、int 的包装类是 Integer、long 的包装类是 Long、float 的包装类是 Float、double 的包装类是 Double、boolean 的包装类是 Boolean、char 的包装类是 Character。在各个包装类中，都定义了一些常用的成员变量和方法。对于数字基本类型的包装类而言，其 MAX_VALUE 成员变量表示该类型所能取得的最大值、MIN_VALUE 成员变量表示该类型所能取得的最小值。以 Byte 为例，下面的代码可以获得 byte 类型的最大值：

```
Byte.MAX_VALUE
```

对于其他类型而言，代码是类似的。

DIY：输出 char 字符类型的取值范围，并输出到控制台中。（15 分）（实例位置：光盘\mr\ym\6.11\diy）

6.7.3 情景应用 3——ASCII 编码查看器

视频讲解：光盘\mr\lx\06\ASCII 编码查看器.exe

实例位置：光盘\mr\ym\6.12\src

ASCII 是 American Standard Code Information Interchange 的缩写，它是基于拉丁字母的一套电脑编码系统，主要用于显示英语字符，是目前世界上最通用的单字节编码。基本的 ASCII 编码包括了 128 个字符。本实例将编写一个 ASCII 编码查看器，可以将字符转换成数字，也可以反向转换，实例的运行效果如图 6.20 所示。

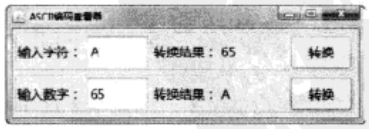


图 6.20　实例运行效果

说明：如果输入了多个字符，则在转换时只取第一个字符转换成数字。本实例未对数字范围进行校验。

实现过程如下：

（1）编写类 ASCIIViewer，该类继承了 JFrame。在窗体中主要包含了两个文本框和两个“转换”按钮，文本框用来获得用户的输入，“转换”按钮用来完成转换功能，并在标签上显示。

（2）编写方法 do_toNumberButton_actionPerformed()，用来监听单击第一个“转换”按钮事件。在该方法中，将用户输入的字符转换成数字，关键代码如下：

```
protected void do_toNumberButton_actionPerformed(ActionEvent e) {
    String ascii = asciiTextField.getText();                    // 获取输入字符
    int i = Character.codePointAt(ascii, 0);                    // 把字符转换为编码
    label3.setText("" + i);                                     // 将编码值显示到窗体标签控件中
}
```

（3）编写方法 do_toASCIIButton_actionPerformed()，用来监听单击第二个“转换”按钮事件。在该方法中，将用户输入的数字转换成字符，关键代码如下：

```
protected void do_toASCIIButton_actionPerformed(ActionEvent e) {
    String number = numberTextField.getText();                  // 获取输入数字
    char[] a = Character.toChars(Integer.parseInt(number));     // 把指定的编码转换为字符
    label6.setText(new String(a));                              // 显示转换后的字符到窗体标签控件中
}
```

你问我答：字符与编码如何转换？

Character 类是 char 类型的包装类，该类除了能将 char 类型转换成对象（引用）类型外，还包括大量处理字符编码的方法。下面介绍本实例用到的相关方法。

（1）获取字符编码

Character 类的 codePointAt()方法可以获得字符的编码，该方法的声明如下：

```
public static int codePointAt(char[] a,int index)
```

参数说明：a 是一个 char 类型的数组，index 是要转换的 char 数组元素的索引值。

（2）把字符编码转换为字符数组

使用 Character 类的 toChars()方法可以将指定的代码点转换成 UTF-16 编码的 char 数组，该方法的声明如下：

```
public static char[] toChars(int codePoint)
```

参数说明：codePoint 是一个 Unicode 代码点，也可以看作是字符编码值。

说明：这里的方法是针对 Unicode 编码的，它支持的字符编码要比 ASCII 多，而且起始的字符编码与 ASCII 是相同的，所以使用 Unicode 编码可以更好地完成本实例。

DIY：Unicode 编码值的范围是 1～65535，在这个范围内生成一个由随机数字确定的编码，然后在控制台中输出这个编码对应的字符。（20 分）（实例位置：光盘\mr\ym\6.12\diy）

6.7.4 情景应用 4——构造方法初始化员工对象

视频讲解：光盘\mr\lx\06\构造方法初始化员工对象.exe

实例位置：光盘\mr\ym\6.13\src

Java 程序的各种功能是通过对象调用相关方法完成的，因此必须先获得对象，并完成对象的初始化。

使用构造方法来获得对象是一种非常常用的方式，构造方法也支持重载，本实例将演示使用不同的构造方法来获得对象，实例的运行结果如图 6.21 所示。

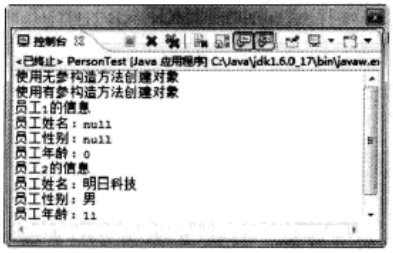


图 6.21　实例运行结果

实现过程如下：

（1）编写类 Person，该类定义了 3 个成员变量，即 name 代表姓名、gender 代表性别、age 代表年龄；两个构造方法，即无参数的使用默认值初始化成员变量、有参数的使用给定值初始化成员变量。3 个 get 方法分别用来获得姓名、性别和年龄，代码如下：

```
public class Person {
    private String name;                    // 定义姓名
    private String gender;                  // 定义性别
    private int age;                        // 定义年龄
    public Person() {                       // 定义没有参数的构造方法
        System.out.println("使用无参构造方法创建对象");
    }
    public Person(String name, String gender, int age) {  // 利用构造方法初始化成员变量
        this.name = name;
        this.gender = gender;
        this.age = age;
        System.out.println("使用有参构造方法创建对象");
    }
    public String getName() {               // 获得姓名
        return name;
    }
    public String getGender() {             // 获得性别
        return gender;
    }
    public int getAge() {                   // 获得年龄
        return age;
    }
}
```

（2）编写类 PersonTest，该类用来测试 Person 类。在该类的 main()方法中，创建了两个 Person 对象并输出了其属性，代码如下：

```
public class PersonTest {
    public static void main(String[] args) {
        Person person1 = new Person();                          // 创建对象
        Person person2 = new Person("明日科技", "男", 11);       // 创建对象
        System.out.println("员工 1 的信息");
        System.out.println("员工姓名：" + person1.getName());    // 输出姓名
        System.out.println("员工性别：" + person1.getGender());  // 输出性别
```

```
        System.out.println("员工年龄：" + person1.getAge());        // 输出年龄
        System.out.println("员工 2 的信息");
        System.out.println("员工姓名：" + person2.getName());       // 输出姓名
        System.out.println("员工性别：" + person2.getGender());     // 输出性别
        System.out.println("员工年龄：" + person2.getAge());        // 输出年龄
    }
}
```

说明：由于员工 1 的信息使用的是默认初始化值，因此字符串类型的成员变量初始化为 null 值，整型成员变量初始化为 0。

DIY：自定义初始化员工对象的构造方法，创建 5 个员工对象组成的数组，并使用冒泡排序算法根据年龄从小到大排序，然后在控制台输出。（25 分）（实例位置：光盘\mr\ym\6.13\diy）

6.7.5 情景应用 5——将字符串转换成整数

视频讲解：光盘\mr\lx\06\将字符串转换成整数.exe

实例位置：光盘\mr\ym\6.14\src

在 Swing 程序中，用户输入的信息通常是使用 getText()方法获得的，该方法的返回值是 String 类型。如果用户输入的是一串数字，而程序又需要使用这些数字进行运算，则可以将字符串转换成整型或浮点型。本实例利用用户输入的整数来计算其平方数，实例的运行效果如图 6.22 所示。

图 6.22 实例运行效果（转换前如左图，转换后如右图）

实现过程如下：

（1）编写类 IntegerConversion，该类继承了 JFrame。在窗体中包含了一个文本框来获得用户的输入、一个“转换”按钮用于转换用户的输入为整数并计算其平方、显示结果和提示信息的标签。

（2）编写方法 do_button_actionPerformed()，用来监听单击“转换”按钮事件。在该方法中，将用户的输入转换成整数并计算其平方数，然后将结果显示在标签中，关键代码如下：

```
protected void do_button_actionPerformed(ActionEvent e) {
    String input = textField.getText();          // 获得用户的输入文本
    int number = Integer.parseInt(input);        // 将文本转换成整数
    label3.setText(number * number + "");        // 计算平方数并显示结果
}
```

你问我答：如何解析字符串为 int 类型？

Integer 类是基本类型中 int 类型的包装类，它提供的静态方法 parseInt()可以将字符串转换成 int 类型，该方法的语法格式如下：

```
public static int parseInt(String str) throws NumberFormatException
```

参数说明：str 是要转换的字符串，如果不能成功转换则会抛出 NumberFormatException 异常。异常就是一个错误提示。要转换的字符串 str 必须完全是由数字组成。

DIY：创建一个包含 5 个文本框的窗体，接收这些文本框控件输入的数字，并把它们组成一个整数数组，然后输出它们排序后的结果。（20 分）（实例位置：光盘\mr\ym\6.14\diy）

情景应用 DIY 栏目分数统计：

DIY 题目	1	2	3	4	5	总分数	
分数							

6.8 自 我 测 试

一、选择题（每题 10 分，5 道题）

1．Java 的源代码文件可以分成 3 部分，即包声明、导入声明和类定义。虽然它们不一定不是必需的，但是如果这几部分同时存在，则需要遵守一定的顺序。以下选项哪个是正确的？（　　）

A．包声明、导入声明和类定义　　B．包声明、类定义和导入声明

C．导入声明、包声明和类定义　　D．导入声明、类定义和包声明

E．类定义、包声明和导入声明　　F．类定义、导入声明和包声明

2．当编译和运行下列代码时会发生什么？（　　）

```
package com.lzw;
private class First{
    private int age;
    public static void main(String[] args) {}
}
```

A．编译错误：不能使用 private 修饰成员变量　　B．编译错误：成员变量未初始化

C．编译错误：main()方法的签名不对　　D．编译错误：不能使用 private 修饰外部类

3．当编译和运行同一文件夹下的如下两个类时会发生什么？（　　）

```
//文件名：P1.java
package com.lzw;
public class P1 {
    public void say() {
        System.out.println("Hello World!");
    }
}

//文件名：P2.java
public class P2 {
    public void say() {
        P1 p1 = new P1();
        p1.say();
    }
}
```

A．两个文件都能正常编译，第二个文件也输出 Hello World！

B．两个文件都不能正常编译

C．两个文件都能正常编译，但第二个文件在运行时发生错误

D．第一个文件能正常编译，第二个文件不能正常编译

4．当编译和运行下列代码时会发生什么？（　　）

```
public class Test {
    public static void main(String[] args) {
        String x = new String("12");
        String y = new String("12");
        if (x.equals(y))
            System.out.println("Equal");
        else
            System.out.println("Not Equal");
    }
}
```

A．编译错误　　B．运行错误　　C．输出 Equal　　D．输出 Not Equal

5．当编译和运行下列代码时会发生什么？（　　）

```
public class Test {
    private String name = "MRSoft";
    public static void main(String[] args) {
        System.out.println(name);
    }
}
```

A．编译错误　　B．运行错误　　C．输出 MRSoft　　D．输出 null

二、填空题（侧重实用技能，每题 10 分，5 道题）

1．下列代码的运行结果是_____。

```
public class Test {
    public Test() {
        System.out.println("Hello");
    }
    public Test(byte b) {
        System.out.println("byte");
    }
    public Test(int i) {
        System.out.println("int");
    }
    public static void main(String[] args) {
        Test test = new Test(12);
    }
}
```

2．为构造方法编写一条语句，它可以调用当前类的无参数构造方法，即_____。

3．编译和运行下列代码，输出的是_____。

```
public class Test {
    private String text;
    public void Test() {
        text = "MRSoft";
    }
    public static void main(String[] args) {
```

```
        Test test = new Test();
        System.out.println(test.text);
    }
}
```

4．编译和运行下列代码，输出的是_____。

```
public class Base {
    public void say(int number) {
        System.out.println("int：" + number);
    }
    public void say(long number) {
        System.out.println("long：" + number);
    }
    public static void main(String[] args) {
        Base obj = new Base();
        obj.say(12);
    }
}
```

5．以下代码能否正确运行？_____

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

测试分数统计：

类别	第 1 题	第 2 题	第 3 题	第 4 题	第 5 题
选择题分数					
填空题分数					
总分数					

6.9　行 动 指 南

开始日期：_____年____月____日　　结束日期：_____年____月____日

序号	内　　容	行 动 指 南	
1	照猫画虎栏目	分数>75 分	优秀，基本功掌握得不错，加油！
		75 分>分数>50 分	及格，知识掌握得不牢，重新做一遍照猫画虎。
	分数（　　）	分数<50 分	请使用光盘中提供的“实战能力测试系统”进行提高训练。
	情景应用栏目	分数>75 分	优秀，综合应用能力很强。
		75 分>分数>50 分	及格，综合应用能力需提高，再练一遍情景应用。
	分数（　　）	分数< 50 分	请使用光盘中提供的“实战能力测试系统”进行提高训练。

续表

序号	内　容	行 动 指 南	
1	自我测试栏目 分数（　　）	分数>75 分	优秀，有成为编程高手的潜质。
		分数<75 分	请使用光盘中提供的“实战能力测试系统”进行提高训练。
	综合评价	反复训练后，以上各项分数都在 75 分以上，方可进入下一堂课学习。	
2	编程能力培养：本栏目提供了一些实践题目，对于培养编程能力很有效，要做好。	（1）把身边某一个事物的所有属性与行为抽象出来，定义一个类，尽量全面。	
		（2）创建一个学生类，它包括众多的成员变量，然后再创建一个以姓名、性别和年龄为属性并初始化学生对象的构造方法。	
		（3）通过用户在控制台输入的姓名、性别与年龄等信息初始化学生对象。提示：这些属性的类别不能单一地使用 String，要使用 int、char 等与属性对应的类型。	
		（4）在控制台接收用户输入的姓名，然后输出该姓名的每个字所对应的 Unicode 编码。	
3	编程习惯培养：本堂课培养读者在学习开发中记笔记的习惯，把开发中遇到的问题、总结的经验记录下来。		
4	创新能力培养：看看身边有没有可以用编程解决的问题，记录到右边的表格中。根据学习进度，尝试编写解决这些问题的小程序。		

6.10　成功可以复制——创造互联网搜索时代

谢尔盖·布林出生在前苏联的一个犹太人家庭。谢尔盖·布林在小学一年级就向老师提交计算机打印输出的设计方案。五岁时跟随父母一起移民美国。中学毕业后，布林进入马里兰大学攻读数学专业，随后进入斯坦福大学读研，在这里他遇到了拉里·佩奇，当时两人都是斯坦福大学的硕士，佩奇 24 岁，布林 23 岁。两人在一次校友会上结识，他们都有很强的主见，意见分歧时互不相让，不管讨论什么问题几乎都会争吵起来。但最终两人却在解决计算机学最大挑战“搜索引擎”的问题时找到了共同点。

1996 年初，佩奇和布林开始合作研究一个名为“BackRub”的搜索引擎，到 1998 年上半年逐步完善这项技术后，两人开始为这项技术寻找合作伙伴。他们找到雅虎的创始人之一戴维·菲洛。菲洛认为他们的技术确实很可靠，但建议他们自己建立一个搜索引擎公司发展业务，发展起来后再考虑合作。吃了无数个闭门羹之后佩奇和布林决定自己创业，但他们手中仅有的一点现金都因购买大量的数据盘和储存器作研究而花光了。他们的一位教师，也是 SUN 微系统的创始人之一安迪·别赫托希姆是个很有远见的人，在看完他们的演示后，立马开了张 10 万美元的支票帮助成立 Google 公司。

Google 搜索引擎源于拉里·佩奇和谢尔盖·布林在斯坦福大学读书时所做的一个研究项目。更确切地说，他们最开始是在佩奇简陋的宿舍搞研究。1998 年 9 月 7 日，在加利福尼亚州的曼罗帕克，Google 公司

在一个朋友租给佩奇和布林办公的车库里成立，在当时看来已经不错了，有一台洗衣机，还有热水器。他们最终在小车库搞出了大工程。佩奇说，那时他们俩到处借钱，教授、亲戚、朋友，只要是想得到的人都去借了，筹得 100 万美元作为最初投资。

1999 年 2 月他们搬到了新的办公室，虽然条件仍然简陋，但比车库好一点，一张乒乓球台就作为正式的会议场所，8 名员工在办公室里都转不过身，一个人要出门所有人都得起身挪开凳子才能腾出地方。

在这样艰苦的环境下，Google 公司一步步发展。当 Google 的竞争对手致力于成为门户网址、投入搜索服务的比重不大之时，布林反其道而行之，尽力完善其搜索引擎。Google 的使用率越来越高，每天的搜索量由 6 年前的 1 万次增至目前的 3 亿次。很多广告商要求在 Google 网页刊登广告。同时，雅虎、宝洁、美国能源部等许多大公司和政府机构也纷纷使用 Google 的搜索技术，Google 按照搜索施放数来收取授权使用费。Google 公司不仅创出奇特的搜索引擎技术，更在上市时以拍卖的方式进行定价，完成“对华尔街的清洗”，两个新的亿万富翁就这样诞生了。

经典语录

Google 取得的成功源于其创建者布林和佩奇的想象力，同样也源于他们的天赋。

深度评价

佩奇和布林的成功告诉我们：要成功，就要勇于创新！而要创新就需要一定的灵感，这灵感不是天生的，而是来自长期的积累和全身心的投入。没有积累就不会有创新。所以我们要掌握新技术，要善于学习，更要善于创新！

第 7 堂课

字符串处理

（视频讲解：116 分钟）

字符串是 Java 程序中经常处理的对象，如果字符串运用得不好，就会影响到程序运行的效率。在 Java 中，字符串作为 String 类的实例来处理。以对象的方式处理字符串，将使字符串更加灵活、方便。了解字符串上可用的操作，可以节省程序编写与维护的时间。

本堂课从创建字符串开始向读者介绍字符串本身的特性以及字符串上可用的几个操作等。

学习摘要：

- 字符串的创建方式
- 字符串的连接方式
- 获取字符串信息的方式
- 字符串的格式化
- 字符串生成器的用法
- 正则表达式

7.1 String 类

前面的章节中介绍了 char 类型，它只能表示单个字符，而由多个字符连接而成的称为字符串。在 Java 语言中字符串作为对象来处理，可以通过 java.lang 包中的 String 类来创建字符串对象。

7.1.1 声明字符串

在 Java 语言中字符串必须包含在一对" "（双引号）之内。字符串是由许多个字符连接而成的。如"23.23"、"ABCDE"、"你好"。这些都是字符串常量，字符串常量是系统能够显示的任何文字信息，甚至是单个字符。

注意：在 Java 中由双引号包围的都是字符串，不能作为其他数据类型来使用，如“1+2”的输出结果永远也不会是 3。

可以通过如下语法格式来声明字符串变量：

```
String str = [null]
```

参数：String 用于指定该变量为字符串类型。str 可以是任意有效的标识符，表示字符串变量的名称。null 是可选参数，如果省略 null，表示 str 变量是未初始化的状态，否则表示声明的字符串的值就等于 null。例如，声明字符串变量 str 可以使用如下代码：

```
String str;
```

说明：声明的字符串变量在使用前必须经过初始化才能使用，否则编译器会报出“变量为被初始化错误”。

7.1.2 构建字符串对象

在 Java 语言中将字符串作为对象来管理，因此可以像创建其他类对象一样来创建字符串对象。创建对象要使用类的构造方法，下面介绍 String 类的常用构造方法。

☑ 用字符数组构建字符串

方法声明：

```
public String(char arr[])
```

参数说明：字符数组 arr 包含要创建的字符串的所有字符。例如，用一个字符数组 a 创建 String 对象使用如下代码：

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

等价于

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

☑ 提取字符数组部分元素构建字符串

方法声明：

```
public String(char arr[], int offset, int length)
```

参数说明：arr 是包含一些字符的数组，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。例如，下列代码将提取字符数组 a 中的一部分创建一个字符串对象。

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

等价于

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

7.2　连接字符串

对于已声明的字符串，可以对其进行相应的操作。连接字符串就是字符操作中较简单的一种。可以对多个字符串进行连接，也可使字符串与其他数据类型进行连接。

7.2.1　连接多个字符串

使用“+”运算符可完成对多个字符串连接的功能。“+”运算符可以连接多个运算符并产生一个 String 对象。

例 7.01　在项目中创建类 Join，在主方法中创建 String 型变量，将字符变量连接的结果输出。代码如下：**（实例位置：光盘\mr\ym\7.01）**

```
public class Join {                                  // 创建类
    public static void main(String args[]) {         // 主方法
        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
        System.out.println(s);                       // 将 s 输出
    }
}
```

运行结果如图 7.1 所示。

图 7.1　实例运行结果

技巧：Java 中一个相连的字符串不能分开在两行中写。例如：

```
System.out.println("I like
Java")
```

这种写法是错误的，如果一个字符串太长，为了便于阅读，可以将这个字符串分在两行上书写。此时就可以使用“+”将两个字符串连起来，之后在加号处换行，因此上面的语句可以修改为：

```
System.out.println("I like" +
"Java");
```

7.2.2　连接其他数据类型

字符串也可同其他数据类型进行连接。如果将字符串同一些数据类型进行连接，会将这些数据直接转换成字符串。

例 7.02　在项目中创建类 Link，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。代码如下：**（实例位置：光盘\mr\ym\7.02）**

```
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
        // 将字符串与整型、浮点型变量相连，并将结果输出
        System.out.println("我每天花费" + booktime + "小时看书；"
                + practice + "小时上机练习");
    }
}
```

运行结果如图 7.2 所示。

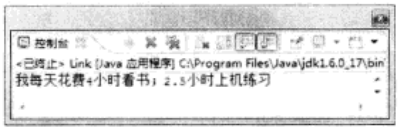


图 7.2　实例运行结果

本实例实现的是将字符串常量与整型变量 booktime 和浮点型变量 practice 相连后的结果输出，在这里 booktime 和 practice 都不是字符串，当它们与字符串相连时会自动将其转换成字符串形式，然后参与连接。

注意：只要“+”运算符的一个操作数是字符串，编译器就会将另一个操作数转换成字符串形式，所以应谨慎地将其他数据类型与字符串相连，以免得到出乎意料的结果。

如果将上例中的输出语句修改为：

```
System.out.println("我每天花费"+booktime+"小时看书;"+(practice+booktime)+
                "小时上机练习");
```

则实例修改后的运行结果如图 7.3 所示。

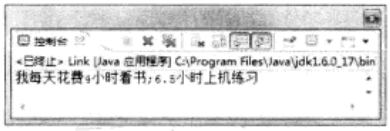


图 7.3　输出语句修改后的运行结果

为什么会这样呢？这是由于运算符是有优先级的，圆括号的优先级最高，所以先被执行，然后再将结果与字符串相连。

7.3　获取字符串信息

字符串作为对象，可通过相应的方法获取字符串的有效信息，如获取某字符串的长度、某个索引位置的字符等。本节将向读者介绍几种获取字符串相关信息的方法。

7.3.1　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，方法声明如下：

```
public String length();
```

例如，下面的代码可以获取字符串长度。字符串 str 的长度将赋值给 int 型变量 size，此时变量 size 的值为 15，这表示 length()方法返回的字符串的长度包括字符串中的空格。

```
String str = "We are students";
int size = str.length();
```

7.3.2　字符串查找

String 类提供了两种查找字符串的方法，即 indexOf()与 lastIndexOf()方法。这两种方法都允许在字符串中搜索指定条件的字符或字符串。indexOf()方法返回的是搜索的字符或字符串首次出现的位置，lastIndexOf()方法返回的是搜索的字符或字符串最后一次出现的位置。

☑　indexOf(String s)

该方法用于返回参数字符串 s 在指定字符串中首次出现的索引位置。当调用字符串的 indexOf()方法时，会从当前字符串的开始位置搜索 s 的位置；如果没有检索到字符串 s，该方法返回值是-1。方法声明如下：

```
public int indexOf(String substr)
```

参数说明：substr 是要搜索的子字符串。例如，查找字符 a 在字符串 str 中的索引位置可以使用如下代码：

```
String str = "We are students";
int size = str.indexOf("a");              //变量 size 的值是 3
```

理解字符串的索引位置，要对字符串的下标有所了解。在计算机中 String 对象是用数组表示的。字符串的下标是 0 至 length()-1。例如，上述代码中字符串 str 的下标如图 7.4 所示。

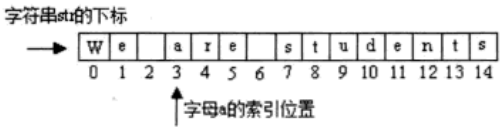


图 7.4　字符串 str 下标

☑　lastIndexOf(String str)

该方法用于返回参数字符串 str 在指定字符串中最后一次出现的索引位置。当调用字符串的 lastIndexOf()方法时，会从当前字符串的开始位置检索参数字符串 str，并将最后一次出现 str 的索引位置返回。如果没有检索到字符串 str，该方法返回-1，方法声明如下：

```
public int lastIndexOf(String substr)
```

参数说明：substr 是要搜索的子字符串。

📖 **说明：如果 lastIndexOf()方法中的参数是空字符串""(注意没有空格)，则返回的结果与调用该字符串的 length()方法的返回结果相同。下面的程序就可说明这个问题。**

例 7.03　在项目中创建类 Text，在主方法中创建 String 对象，使用 lastIndexOf()方法查看字符串 str 中空字符串的位置，然后输出字符串的长度，看它们是否相同。代码如下：**（实例位置：光盘\mr\ym\7.03）**

```
public class Text {                                    // 创建类
    public static void main(String args[]) {          // 主方法
        String str = "We are students";               // 定义字符串 str
        // 将空字符串在 str 中的索引位置赋值给变量 size
        int size = str.lastIndexOf("");
        // 将变量 size 输出
        System.out.println("空字符在字符串 str 中的索引位置是：" + size);
        // 将字符串 str 的长度输出
        System.out.println("字符串 str 的长度是：" + str.length());
    }
}
```

运行结果如图 7.5 所示。

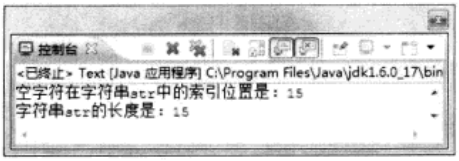


图 7.5　实例运行结果

7.3.3　获取指定索引位置的字符

使用 charAt()方法可将指定索引位置的字符返回，方法声明如下：

```
public char charAt(int index)
```

参数说明：index 用于指定要返回字符的下标。

例 7.04　在项目中创建类 Ref，在主方法中创建 String 对象，使用 charAt()方法查看字符串 str 中索引位置是 5 的字符。代码如下：**（实例位置：光盘\mr\ym\7.04）**

```
public class Ref {                                    // 创建类
    public static void main(String args[]) {          // 主方法
        String str = "hello word";                    // 定义字符串 str
        char mychar = str.charAt(6);                  // 将字符串 str 中索引位置是 5 的字符返回
        System.out.println("字符串 str 中索引位置是 5 的字符为：" + mychar);     // 输出信息
    }
}
```

运行结果如图 7.6 所示。

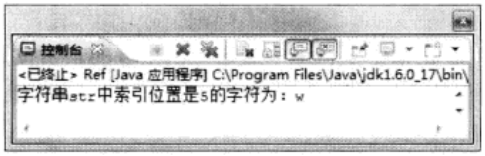


图 7.6　实例运行结果

7.4　格式化字符串

String 类的静态方法 format()用于创建格式化的字符串。该方法使用指定的格式字符串和参数返回一个经过格式化的字符串，格式化后的新字符串使用本地默认的语言环境。本节将介绍字符串格式化的使用方法。format()方法的方法声明如下：

```
public static String format(String format, Object... args)
```

参数说明：format 是格式字符串，args 是格式字符串中由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，可以只提供一个参数。

7.4.1　日期和时间字符串格式化

在应用程序设计中，经常需要显示时间和日期。如果想输出满意的日期时间格式，一般需要编写大量的代码并经过各种算法才能实现。format()方法通过给定特殊转换符作为参数来实现对日期和时间的格式化。

1．日期格式化

如果使用字符串的 format()方法格式化日期和时间，可以得到日期字符串的完整格式或者部分字段。例如，获取单独的年、月或日信息，下面的代码将返回一个月中的某一天。

```
Date date = new Date();                          // 创建 Date 对象 date
String str = String.format("%te", date);         // 通过 format()方法对 date 进行格式化
```

上述代码中变量 str 的值是当前日期中的日字段，例如，今天是 15 号，则 str 的值为 15；“%te”是转换符。常用的日期格式化转换符如表 7.1 所示。

表 7.1　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）	2
%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 7.05　在项目中创建类 Eval，实现将当前日期信息以 4 位年份、月份全称、2 位日期形式输出。代码如下：**（实例位置：光盘\mr\ym\7.05）**

```
import java.util.Date;                                   // 导入 java.util.Date 类
public class Eval {                                      // 新建类
    public static void main(String[] args) {             // 主方法
        Date date = new Date();                          // 创建 Date 对象 date
        String year = String.format("%tY", date);        // 将 date 进行格式化
        String month = String.format("%tB", date);
        String day = String.format("%td", date);
        System.out.println("今年是：" + year + "年");     // 输出信息
        System.out.println("现在是：" + month);
        System.out.println("今天是：" + day + "号");
    }
}
```

运行结果如图 7.7 所示。

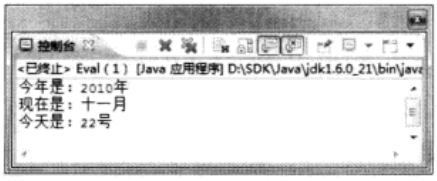


图 7.7　实例运行结果

2. 时间格式化

使用 format()方法不但可完成日期的格式化，也可实现时间的格式化。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，格式化时间的转换符如表 7.2 所示。

表 7.2 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 7.06 在项目中创建类 GetDate，实现将当前时间信息以 2 位小时、2 位分钟数、2 位秒数的形式输出。代码如下：**（实例位置：光盘\mr\ym\7.06）**

```
import java.util.Date;                                  // 导入 java.util.Date 类
public class GetDate {                                  // 新建类
    public static void main(String[] args) {            // 主方法
        Date date = new Date();                         // 创建 Date 对象 date
        String hour = String.format("%tH", date);       // 将 date 进行格式化
        String minute = String.format("%tM", date);
        String second = String.format("%tS", date);
        // 输出的信息
        System.out.println("现在是：" + hour + "时" + minute + "分"
                + second + "秒");
    }
}
```

运行结果如图 7.8 所示。

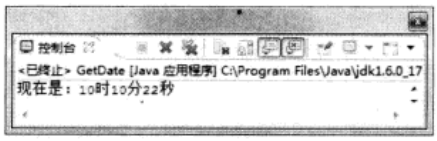


图 7.8 实例运行结果

3. 格式化常见的日期时间组合

格式化日期与时间的转换符定义了各种日期时间组合的格式，其中最常用的日期和时间的组合格式如

表 7.3 所示。

表 7.3　常见的日期和时间的组合格式

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 7.07　在项目中创建类 DateAndTime，在主方法中实现将当前日期时间的全部信息以及指定格式的日期输出。代码如下：**（实例位置：光盘\mr\ym\7.07）**

```
import java.util.Date;                                  // 导入 java.util.Date 类
public class DateAndTime {                              // 创建类
    public static void main(String[] args) {            // 主方法
        Date date = new Date();                         // 创建 Date 对象 date
        String time = String.format("%tc", date);       // 将 date 格式化
        String form = String.format("%tF", date);
        // 将格式化后的日期时间输出
        System.out.println("全部的时间信息是：" + time);
        System.out.println("年-月-日格式：" + form);
    }
}
```

运行结果如图 7.9 所示。

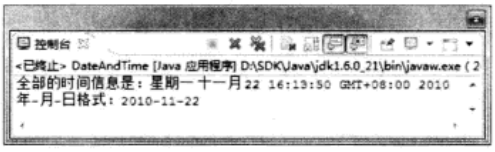


图 7.9　实例运行结果

7.4.2　常规类型格式化

常规类型的格式化可应用于任何参数类型，可通过如表 7.4 所示的转换符来实现。

表 7.4　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

续表

转 换 符	说 明	示 例
%e	结果被格式化为用计算机科学计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 7.08 在项目中创建类 General，在主方法中实现不同数据类型到字符串的转换。代码如下：**（实例位置：光盘\mr\ym\7.08）**

```
public class General {                                  // 新建类
    public static void main(String[] args) {            // 主方法
        String str = 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);         // 将结果以十进制格式显示
        System.out.println("400 的一半是：" + str);      // 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 7.10 所示。

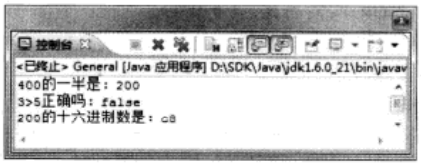


图 7.10 实例运行结果

7.5 字符串生成器

对于创建成功的字符串对象，它的长度是固定的，内容不能被改变。虽然使用“+”可以达到附加新字符或字符串的目的，但“+”会产生一个新的 String 实例，会在内存中创建新的字符串对象。如果重复地对字符串进行连接以获取符合条件的新字符串，将会极大地增加系统开销。而 Java 6 新增了可变的字符序列 StringBuilder 类，它大大提高了频繁增加与修改字符串的效率，通过下面简单的测试就可以知道。

例 7.09 在项目中创建类 Jerque，在主方法中编写如下代码，验证字符串操作和字符串生成器操作的效率。**（实例位置：光盘\mr\ym\7.09）**

```
public class Jerque {                                   // 新建类
    public static void main(String[] args) {            // 主方法
        String str = "";                                // 创建空字符串
        // 定义对字符串执行操作的起始时间
        long starTime = System.currentTimeMillis();
        for (int i = 0; i < 10000; i++) {               // 利用 for 循环执行 10000 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;                                    // 计算对字符串执行操作的时间
        System.out.println("String 消耗时间：" + time + "毫秒");              // 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 (int j = 0; j < 10000; 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 + "毫秒");      // 将操作时间输出
    }
}
```

运行结果如图 7.11 所示。

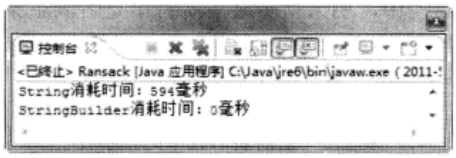


图 7.11　实例运行结果

通过实例 7.9 可以看出执行的时间差距很大。如果在程序中频繁地附加字符，建议使用 StringBuilder。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。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。若要使用 StringBuilder 最后输出字符串结果，可使用 toString()方法。利用 StringBuilder 类中的方法可动态地执行添加、删除和插入等字符串的编辑操作，该类的常用方法如下。

☑ append()方法

该方法用于向字符串生成器中追加内容。通过该方法的多个重载形式，可实现接收任何类型的数据，如 int、boolean、char、String、double 或者另一个字符串生成器等，方法声明如下：

```
public StringBuilder append(int content)
```

参数说明：content 表示要追加到字符串生成器中的内容。可以是任何类型的数据或其他对象，因为该方法有多种重载形式，每个形式接收不同数据类型的参数。

☑ insert(int offset , arg)方法

该方法用于向字符串生成器中的指定位置插入数据内容。通过该方法的不同重载形式，可实现向字符串生成器中插入 int、float、char 和 boolean 等基本数据类型或其他对象，方法声明如下：

```
public StringBuilder insert(int offset, str)
```

参数说明：offset 表示字符串生成器的位置。该参数必须大于等于 0，且小于等于此序列的长度。str 是将插入至字符串生成器的数据。该数据可以是任何的数据类型或其他对象。例如，下面的代码可以向字符串生成器中指定的位置添加字符。

```
StringBuilder bf = new StringBuilder("hello");          // 创建字符生成器
bf.insert(5, "word");                                   // 添加至字符生成器的内容
System.out.println(bf.toString());                      // 此时输出信息为 helloword
```

☑ delete(int start, int end)方法

移除此序列的子字符串中的字符。该子字符串从指定的 start 处开始，一直到索引 end-1 处的字符，如果不存在这种字符，则一直到序列尾部。如果 start 等于 end，则不发生任何更改，方法声明如下：

```
public StringBuilder delete(int start, int end)
```

参数说明：start 是将要删除的字符串的起点位置，end 是将要删除的字符串的终点位置。例如，删除指定位置的子字符串可以使用如下代码：

```
StringBuilder bf = new StringBuilder("StringBuilder");      // 创建字符串生成器
bf.delete(5, 10);                                           // 删除的子字符串
System.out.println(bf.toString());                          // 此时输出的信息为 Strinder
```

📖 **说明：想要了解更多的 StringBuilder 类方法，可查询 java.lang.StringBuilder 的 API 说明。**

7.6 使用正则表达式

正则表达式通常被用于判断语句中，用来检查某一字符串是否满足某一格式。正则表达式是含有一些具有特殊意义字符的字符串，这些特殊字符称为正则表达式的元字符。例如，“\\d”表示数字 0～9 中任何一个，“\\d”就是元字符。正则表达式中元字符及其意义如表 7.5 所示。

表 7.5 正则表达式中的元字符

元 字 符	正则表达式中的写法	意 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：正则表达式中的“.”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“\”。**

在正则表达式中可以使用方括号将若干个字符括起来表示一个元字符，该元字符可代表方括号中的任何一个字符。例如，reg = "[abc]4"，字符串"a4"、"b4"、"c4"都是和正则表达式匹配的字符串。方括号中的元字符还可以为其他格式，如：

[^456]：代表 4、5、6 之外的任何字符。

[a-r]：代表 a～r 的任何一个。

[a-zA-Z]：可表示任意一个英文字母。

[a-e[g-z]]：代表 a～e 或 g～z 中的任何一个字母（并运算）。

[a-o&&[def]]：代表字母 d、e、f（交运算）。

[a-d&&[^bc]]：代表字母 a、d（差运算）。

在正则表达式中允许使用限定修饰符来限定元字符出现的次数。例如，“A*”代表 A 可在字符串中出现 0 次或多次，限定修饰符的用法如表 7.6 所示。

表 7.6　限定修饰符

限定修饰符	意　义	示　例
?	0 次或 1 次	A?
*	0 次或多次	A*
+	一次或多次	A+
{n}	正好出现 n 次	A{2}
{n,}	至少出现 n 次	A{3,}
{n,m}	出现 n 次至 m 次	A{2,6}

例 7.10　在项目中创建类 Judge，在主方法中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。代码如下：**（实例位置：光盘\mr\ym\7.10）**

```
public class Judge {
    public static void main(String[] args) {
        // 定义要匹配 E-mail 地址的正则表达式
        String regex = "\\w+@\\w+(\\.\\w{2,3})*\\.\\w{2,3}";
        String str1 = "aaa@";                    // 定义要进行验证的字符串
        String str2 = "aaaaa";
        String str3 = "mingrisoft@mingrisoft.com";
        if (str1.matches(regex)) {               // 判断字符串变量是否与正则表达式匹配
            System.out.println(str1 + "是一个合法的 E_Mail 地址格式");
        }
        if (str2.matches(regex)) {
            System.out.println(str2 + "是一个合法的 E_Mail 地址格式");
        }
        if (str3.matches(regex)) {
            System.out.println(str3 + "是一个合法的 E_Mail 地址格式");
        }
    }
}
```

实例运行结果如图 7.12 所示。

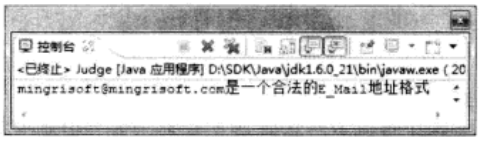


图 7.12　实例运行结果

正则表达式分析如下：

通常情况下 E-mail 的格式为 X@X.com.cn。字符 X 表示任意一个或多个字符，@为 E-mail 地址中的特有符号，符号@后还有一个或多个字符，之后是字符.com，也可能后面还有类似.cn 的标记。由于 E-mail 地址的这些特点，因此可以书写正则表达式\\w+@\\w+(\\.\\w+)*\\.\\w+来匹配 E-mail 地址。字符集\\w 匹配任意字符，符号“+”表示字符可以出现一次或多次，表达式(\\.\\w+)*表示形如.com 格式的字符串可以出现 0 次或多次。而最后的表达式\\.\\w+用于匹配 E-mail 地址中的结尾字符，如.com。

7.7 照猫画虎——基本功训练

7.7.1 基本功训练 1——获取子字符串

视频讲解：光盘\mr\lx\07\获取子字符串.exe

实例位置：光盘\mr\ym\7.11\src

在项目中创建类 Subs，在主方法中创建 String 对象，实现使用 substring()方法对字符串进行截取，并将截取后形成的新串输出。实例代码如下：

```
public class Subs {                                    // 创建类
    public static void main(String args[]) {           // 主方法
        String str = "hello word";                     // 定义的字符串
        String substr = str.substring(0, 3);           // 对字符串进行截取
        System.out.println("原字符串：" + str);
        System.out.println("截取下标 0 至 3 的子字符串为：");
        System.out.println("\t" + substr);             // 输出截取后的字符串
    }
}
```

运行结果如图 7.13 所示。

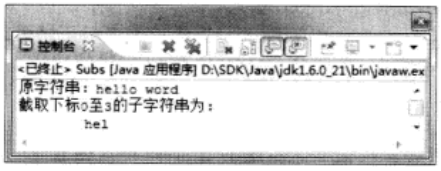


图 7.13 实例运行结果

你问我答：如何截取字符串？

通过 String 类的 substring()方法可对字符串进行截取。这些方法的共同点就是都利用字符串的下标进行截取，应明确字符串下标是从 0 开始的。

substring()方法被两种不同的方法重载，来满足不同的需要。

（1）substring(int beginIndex)

该方法返回的是从指定的索引位置开始截取直到该字符串结尾的子串。

public String substring(int beginIndex)

参数说明：beginIndex 用于指定从某一索引处开始截取字符串。例如，下面的代码将实现截取子字符串：

```
String str = "Hello Word";                    // 定义字符串 str
```

```
String substr = str.substring(3);          // 获取字符串，此时 substr 值为 lo Word
```

使用 substring(beginIndex)截取字符串的示意图如图 7.14 所示。

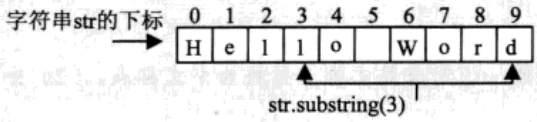


图 7.14　substring(3)方法的截取示意图

注意：在字符串中空格占用一个索引位置。

（2）substring(int beginIndex, int endIndex)

该方法返回的是从字符串某一索引位置开始截取至某一索引位置结束的子串，方法声明如下：

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

参数说明：beginIndex 是开始截取子字符串的索引位置，endIndex 是子字符串在整个字符串中的结束位置。

照猫画虎：把 Math.PI 转换为字符串，并保留 5 位小数截取为新字符串。（20 分）（实例位置：光盘\mr\ym\7.11\zmhh）

7.7.2　基本功训练 2——字符串替换

视频讲解：光盘\mr\lx\07\字符串替换.exe

实例位置：光盘\mr\ym\7.12\src

在项目中创建类 NewStr，在主方法中创建 String 型变量，将字符变量中的小写字母 j 替换成大写字母 J，并在控制台输出结果，实例代码如下：

```
public class NewStr {                                  // 创建类
    public static void main(String args[]) {           // 主方法
        String str = "java project";                   // 定义字符串 str
        // 字符串 str 中的字符“j”替换成“J”后返回的新字符串 newstr
        String newstr = str.replace("j", "J");
        System.out.println(newstr);                    // 将字符串 newstr 输出
    }
}
```

运行结果如图 7.15 所示。

图 7.15　实例运行结果

你问我答：replace()方法有什么作用？

replace()方法可实现将指定的字符或字符串替换成新的字符或字符串，方法声明如下：

```
public String replace(CharSequence target, CharSequence replacement)
```

参数说明：target 是准备替换的字符或字符串，replacement 用于替换原来字符串的内容。replace()方法

返回的结果是一个新的字符串。如果 target 没有出现在字符串序列中，则将原字符串返回。

说明：如果要替换的字符序列 target 在原字符串中重复出现多次，replace()方法会将所有的 target 全部替换成新的字符序列 replacement，并且严格区分字母的大小写。

照猫画虎：编程把用户输入的所有英文标点替换为中文标点。（20 分）（实例位置：光盘\mr\ym\7.12\zmhh）

7.7.3 基本功训练 3——判断字符串的开始与结尾

视频讲解：光盘\mr\lx\07\判断字符串的开始与结尾.exe

实例位置：光盘\mr\ym\7.13\src

在项目中创建类 StartOrEnd，在主方法中创建 String 型变量，并判断变量开始和结尾的字符串，实例代码如下：

```
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 b = 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？" + b);
        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 7.16 所示。

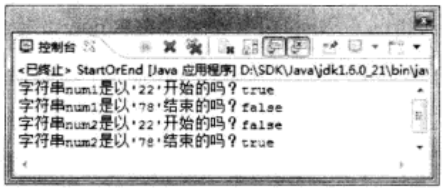


图 7.16 实例运行结果

你问我答：如何检测字符串的前缀与后缀？

startsWith()方法与 endsWith()方法分别用于判断字符串是否以指定的内容开始或结束，也就是判断前缀与后缀。这两个方法的返回值都为 boolean 类型。

（1）判断字符串前缀

startsWith()方法用于判断当前字符串对象的前缀是否是参数指定的子字符串，方法声明如下：

```
public boolean startsWith(String prefix)
```

参数说明：prefix 是判断前缀的字符。

（2）判断字符串后缀

endsWith()方法用于判断当前字符串是否是以给定的子字符串结束，方法声明如下：

```
public boolean endsWith(String suffix)
```

参数说明：suffix 是判断后缀的字符串。

照猫画虎：编程接收用户在控制台输入的网址，如果不是以 http 做前缀就提示用户格式错误信息。（20 分）（实例位置：光盘\mr\ym\7.13\zmhh）

7.7.4　基本功训练 4——字母大小写转换

视频讲解：光盘\mr\lx\07\字母大小写转换.exe

实例位置：光盘\mr\ym\7.14\src

在项目中创建类 UpAndLower，在主方法中创建 String 型变量，实现字符变量的大小写转换，并将转换后的结果输出，实例代码如下：

```
public class UpAndLower {                          // 创建类
    public static void main(String args[]) {       // 主方法
        String str = new String("abc DEF");        // 创建的字符串 str
        String newstr = str.toLowerCase();         // 使用 toLowerCase()方法实行小写转换
        String newstr2 = str.toUpperCase();        // 使用 toUpperCase()方法实行大写转换
        System.out.println(newstr);                // 将转换后的结果输出
        System.out.println(newstr2);
    }
}
```

运行结果如图 7.17 所示。

图 7.17　实例运行结果

你问我答：如何对字符串大小写进行转换？

字符串的 toLowerCase()方法可以将字符串内的所有字符从大写字母改写为小写字母，而 toUpperCase()方法可以将字符串中的小写字母改写为大写字母。

（1）toLowerCase()方法

该方法将字符串中所有字母转换为小写。如果字符串中没有应该被转换的字符，则将原字符串返回；否则将返回一个新的字符串，将原字符串中每个该进行小写转换的字符都转换成相应的小写字符，转换后字符串与原字符长度相同，该方法的方法声明如下：

```
public String toLowerCase()
```

（2）toUpperCase()方法

该方法的功能与 toLowerCase()方法完全相反，用于转换字符串中所有小写字母为大写字母，该方法的方法声明如下：

```
public String toUpperCase()
```

例如，本实例中就应用到这两个方法来完成大小写字母的转换，关键代码如下：

```
String newstr = str.toLowerCase(); // 使用 toLowerCase()方法实行小写转换
String newstr2 = str.toUpperCase();// 使用 toUpperCase()方法实行大写转换
```

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

照猫画虎：编程接收用户在控制台输入的 E-mail，把输入的内容全部修改为大写格式并将其输出到控制台。（20 分）（实例位置：光盘\mr\ym\7.14\zmhh）

7.7.5 基本功训练 5——字符串分割

视频讲解：光盘\mr\lx\07\字符串分割.exe

实例位置：光盘\mr\ym\7.15\src

在项目中创建类 Division，在主方法中创建 String 型变量，并将字符变量进行分割，将分割后的结果输出，实例代码如下：

```
public class Division {                                    // 创建类
    public static void main(String args[]) {               // 主方法
        String str = new String("abc,def,ghi,gkl");        // 定义的字符串 str
        // 使用 split()方法对字符串进行拆分，返回字符串数组
        String[] newstr = str.split(",");
        System.out.println("==============第 1 次分割============");
        for (int i = 0; i < newstr.length; i++) {          // 使用 for 循环遍历字符数组
            System.out.println(newstr[i]);                 // 输出信息
        }
        // 对字符串进行拆分，并限定拆分次数，返回字符数组
        String[] newstr2 = str.split(",", 2);
        System.out.println("==============第 2 次分割============");
        for (int j = 0; j < newstr2.length; j++) {         // 循环遍历字符数组
            System.out.println(newstr2[j]);                // 输出信息
        }
    }
}
```

运行结果如图 7.18 所示。

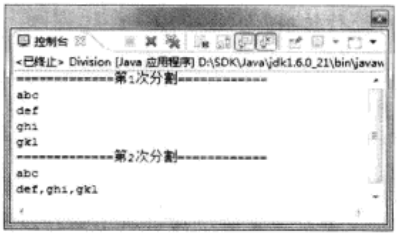


图 7.18 实例运行结果

你问我答：如何将字符串分割为数组？

通过 split()方法可以使用指定的分割符或正则表达式对字符串进行分割，并将分割后的结果存放在字符串数组中。split()方法提供了以下两种重载形式。

（1）split(String sign)

该方法可根据给定的分割符对字符串进行拆分，方法声明如下：

public String[] split(String regex)

参数说明：sign 为分割字符串的分割符，也可以使用正则表达式。

📖 **说明**：如果想定义多个分割符，可使用符号“|”连接。例如，“,|=”表示分割符分别为“,”和“=”。

（2）split(String sign,int limit)

该方法可根据给定的分割符对字符串进行拆分，并限定拆分的次数，方法声明如下：

public String[] split(String regex, int limit)

参数说明：sign 是分割字符串的分割符，也可以使用正则表达式，limit 是限制的分割次数。

照猫画虎：编程接收用户在控制台输入的以分号分割的员工名称，然后在主方法中把该字符串解析成字符串数组，每个数组元素都是一个员工名字。（20 分）（实例位置：光盘\mr\ym\7.15\zmhh）

照猫画虎栏目分数统计：

照猫画虎题目	1	2	3	4	5	总分数	
分数							

7.8　情景应用——拓展与实践

7.8.1　情景应用 1——字符串数组排序

视频讲解：光盘\mr\lx\07\字符串数组排序.exe

实例位置：光盘\mr\ym\7.16\src

用户名称也就是登录系统和网站所使用的名称，也称登录名称。一般情况下用户名都要求使用英文、数字与符号组成，如 li_zhongwei。这些用户名一般是根据用户注册的先后来排序的，这样不利于管理员的查找，本实例实现对用户名字符串进行排序，实例运行结果如图 7.19 所示。

图 7.19　实例升序排列的结果

实现过程如下：

（1）在项目中创建窗体类 UserSort，在窗体界面中添加一个 JList 列表框控件和“升序”、“降序”、“关闭”3 个按钮。

（2）编写处理“升序”和“降序”按钮事件的方法，该方法将使用冒泡算法完成用户名数组的排序，

并使用排序好的数组更新界面中 JList 控件的内容，关键代码如下：

```
protected void do_button_actionPerformed(final ActionEvent e) {
    for (int i = names.length; --i >= 0;) {                          // 遍历数组
        for (int j = 0; j < i; j++) {                                // 遍历并排序所有未排序元素
            boolean compare = names[j].compareToIgnoreCase(names[j + 1]) > 0;
            if (compare && e.getSource() == ascButton ||
                    !compare && e.getSource() == descButton) {       // 条件判断
                String temp = names[j];                              // 数组元素交换
                names[j] = names[j + 1];
                names[j + 1] = temp;
            }
        }
    }
    sourceList.setListData(names);                                   // 把数组设置到 JList 空间中
}
```

你问我答：如何实现字符串比较？

用户名是以字符串形式保存的，无论在数据库中还是在数据集合中。所以本实例使用了对比字符串的方法来实现字符串的对比，下面介绍本实例使用的关键方法。

（1）compareTo()方法

该方法将按字典顺序比较两个字符串。该比较基于字符串中各个字符的 Unicode 值。如果按字典顺序此 String 对象位于参数字符串之前，则比较结果为一个负整数；如果按字典顺序此 String 对象位于参数字符串之后，则比较结果为一个正整数。如果这两个字符串相等，则结果为 0；compareTo 只在方法 equals(Object) 返回 true 时才返回 0，compareTo()方法声明如下：

```
public int compareTo(String anotherString)
```

参数说明：anotherString 是要比较的 String 字符串对象。

（2）compareToIgnoreCase()方法

同上一个方法执行的功能相同，该方法也用于对比两个字符串，但是不再严格区分字母的大小写，该方法的方法声明如下：

```
public int compareToIgnoreCase(String str)
```

参数说明：str 是要比较的 String 字符串对象。

DIY：尝试不使用 compareToIgnoreCase()方法完成忽略字母大小写的用户名数组升序排列。（20 分）
（实例位置：光盘\mr\ym\7.16\diy）

7.8.2 情景应用 2——格式化当前日期

视频讲解：光盘\mr\lx\07\格式化当前日期.exe

实例位置：光盘\mr\ym\7.17\src

日期字符串的格式因语言环境而不同，国际化的程序必须考虑程序在不同语言环境中的应用。所以提供一个格式化类就非常必要，Java 的 java.text 包中提供了 DateFormat 类，本实例通过该类实现了几个不同语言环境的日期格式输出，实例运行结果如图 7.20 所示。

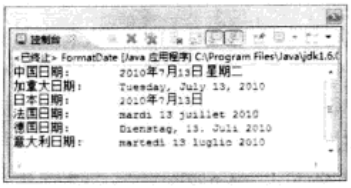


图 7.20　实例运行结果

实现过程如下：

在项目中创建 DateFormat 类，在该类的主方法中创建一个 Date 类的日期对象，再分别创建各语言环境的日期格式器对象，并输出这些格式器对象对该日期进行格式化后的字符串信息，程序关键代码如下：

```
import java.text.DateFormat;
import java.util.Date;
import java.util.Locale;
public class FormatDate {
    public static void main(String[] args) {
        Date date = new Date();
        DateFormat formater = DateFormat.getDateInstance(DateFormat.FULL,
                Locale.CHINA);
        // 中国日期
        String string = formater.format(date);
        System.out.println("中国日期：\t"+string);
        // 加拿大日期
        formater = DateFormat.getDateInstance(DateFormat.FULL,
                Locale.CANADA);
        System.out.println("加拿大日期：\t"+formater.format(date));
        // 日本日期
        formater = DateFormat.getDateInstance(DateFormat.FULL,
                Locale.JAPAN);
        System.out.println("日本日期：\t"+formater.format(date));
        // 法国日期
        formater = DateFormat.getDateInstance(DateFormat.FULL,
                Locale.FRANCE);
        System.out.println("法国日期：\t"+formater.format(date));
        // 德国日期
        formater = DateFormat.getDateInstance(DateFormat.FULL,
                Locale.GERMAN);
        System.out.println("德国日期：\t"+formater.format(date));
        // 意大利日期
        formater = DateFormat.getDateInstance(DateFormat.FULL,
                Locale.ITALIAN);
        System.out.println("意大利日期：\t"+formater.format(date));
    }
}
```

你问我答： DateFormat 如何格式化日期？

DateFormat 类可以实现对日期进行格式化，它位于 java.text 包，是一个抽象类，不能被实例化，但是它提供了一些静态方法来获取内部的实现类的实例对象。下面介绍本实例是如何获取 DateFormat 类的对象和如何进行格式化的。

（1）获取日期格式器

getDateInstance()方法可以获取指定样式和语言环境的日期格式器对象，方法声明如下：

```
public static final DateFormat getDateInstance(int style,Locale aLocale)
```

参数说明：style 用于指定格式器对象对日期使用的格式化样式，可选值有 SHORT（使用数字）、LONG（比较长的描述）和 FULL（完整格式）。aLocale 是格式器使用的语言环境对象。

（2）日期格式化

format()方法将一个日期对象格式化为指定格式的字符串，方法声明如下：

```
public final String format(Date date)
```

参数说明：date 是日期类的实例对象。例如，实例中格式化中国日期时使用了如下代码，包括获取日期格式器与执行日期格式化步骤。

```
DateFormat formater = DateFormat.getDateInstance(DateFormat.FULL, Locale.CHINA);
// 中国日期
String string = formater.format(date);
System.out.println("中国日期：\t"+string);
```

DIY：把实例中的输出格式都修改为较短的日期格式。（20 分）（实例位置：光盘\mr\ym\7.17\diy）

7.8.3 情景应用 3——正则表达式验证 IP 地址

视频讲解：光盘\mr\lx\07\正则表达式验证 IP 地址.exe

实例位置：光盘\mr\ym\7.18\src

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是要经过 DNS 服务器转换为 IP 地址才能找到服务器的。在很多网络程序中要求设置服务器 IP 地址或者输入对方连接 IP 地址，IP 地址的错误输入将使程序无法运行。本实例实现对 IP 地址的验证功能，实例运行结果如图 7.21 所示，把该功能加载到网络程序中，可以避免用户输入错误的 IP 地址。

图 7.21 实例运行结果

实现过程如下：

（1）在项目中创建窗体类 CheckIPAddress，在该窗体中添加一个输入 IP 地址的文本框和一个“验证”按钮。

（2）编写“验证”按钮的事件处理方法，该方法将获取用户输入的内容，然后调用 matches()方法对其进行判断，然后在对话框中输出结果，关键代码如下：

```
protected void do_button_actionPerformed(ActionEvent e) {
    String text = ipField.getText();                    // 获取用户输入
    String info = ipCheck(text);                        // 对输入文本进行 IP 验证
    JOptionPane.showMessageDialog(null, info);          // 用对话框输出验证结果
}
```

（3）编写验证 IP 地址的 matches()方法，该方法利用正则表达式对输入字符串进行验证，并返回验证结果，关键代码如下：

```
public String ipCheck(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)$";
        // 判断 IP 地址是否与正则表达式匹配
        if(text.matches(regex)){
            // 返回判断信息
            return text + "\n 是一个合法的 IP 地址！";
        }else{
            // 返回判断信息
            return text + "\n 不是一个合法的 IP 地址！";
        }
    }
    // 返回判断信息
    return "请输入要验证的 IP 地址！";
}
```

你问我答：matches()方法的作用是什么？

实例的关键点在于 IP 地址格式与数字范围的验证，用户在输入 IP 地址时，程序可以获取的只有字符串类型，所以本实例利用字符串的灵活性与正则表达式搭配进行 IP 格式与范围的验证。matches()方法是 String 字符串类的方法，用于判断字符串与指定的正则表达式是否匹配，该方法的方法声明如下：

public boolean matches(String regex)

参数说明：regex 是正则表达式，它将与字符串进行匹配。

DIY：编程接收用户在控制台的输入，使用正则表达式判断输入的内容是否全部为整数数字。（20 分）（实例位置：光盘\mr\ym\7.18\diy）

7.8.4　情景应用 4——正则表达式鉴别非法电话号码

视频讲解：光盘\mr\lx\07\正则表达式鉴别非法电话号码.exe

实例位置：光盘\mr\ym\7.19\src

程序经常需要用户输入用户信息或联系方式，其中有一些数据的格式是固定的，程序处理逻辑也是按照这个格式来实现的，但是由于用户输入的是字符串，其灵活性较大，容易输入错误格式的数据。例如，用户联系信息的电话号码就是固定格式的数据，本实例将演示如何利用正则表达式来确定输入的电话号码格式是否匹配，实例运行结果如图 7.22 所示，在程序中加入该模块可以禁止用户输入错误的电话号码。

图 7.22　输入正确电话格式的运行结果

实现过程如下：

（1）在项目中新建窗体类 CheckPhoneNum，在该窗体中添加 3 个文本框用于输入姓名、年龄与电话号码，再添加一个“验证”按钮。

（2）编写“验证”按钮的事件处理方法，该方法获取用户在文本框中输入的电话号码字符串，然后调用 check()方法进行验证，并把验证结果使用对话框输出，关键代码如下：

```
protected void do_button_actionPerformed(ActionEvent e) {
    String text = phoneNumField.getText();              // 获取用户输入
    String info = check(text);                          // 对输入文本进行 IP 验证
    JOptionPane.showMessageDialog(null, info);          // 用对话框输出验证结果
}
```

（3）编写 check()方法，该方法用于验证指定的字符串与正确的电话号码格式是否匹配，该方法首先判断字符串是否为空，然后再通过正则表达式对字符串进行验证，并将验证结果作为方法的返回值，关键代码如下：

```
public String check(String text){
    if(text == null || text.isEmpty()){
        return "请输入电话号码！";
    }
    // 定义正则表达式
    String regex = "^\\d{3}-?\\d{8}|\\d{4}-?\\d{8}$";
    // 判断输入数据是否为电话号码
    if(text.matches(regex)){
        return text + "\n 是一个合法的电话号码！";
    }else{
        return text + "\n 不是一个合法的电话号码！";
    }
}
```

DIY：编程判断用户输入的 E-mail 地址是否合法。（20 分）（实例位置：光盘\mr\ym\7.19\diy）

7.8.5 情景应用 5——获取字符串中汉字的个数

视频讲解：光盘\mr\lx\07\获取字符串中汉字的个数.exe

实例位置：光盘\mr\ym\7.20\src

字符串中可以包括数字、字母、汉字或其他字符。使用 Character 类的 isDigit()方法可以判断字符串中的某个字符是否为数字，使用 Character 类的 isLetter()方法可以判断字符串中的某个字符是否为字母。本实例将介绍一种方法用来判断字符串中的某个字符是否为汉字，通过此方法可以计算字符串中汉字的数量，实例运行效果如图 7.23 所示。

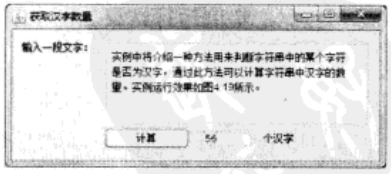


图 7.23 获取字符串中汉字的个数

实现过程如下：

（1）在项目中创建窗体类 ChineseAmount，在窗体中添加接收用户输入的 JTextArea 文本框控件、显示汉字数量的文本框控件和计算汉字数量的“计算”按钮。

（2）编写“计算”按钮的事件处理方法，在该方法中获取用户输入的字符串，然后遍历字符串中的每一个字符，使用正则表达式判断字符是否属于汉字，然后根据结果对汉字进行计数，最后把计数结果显示到界面文本框中，关键代码如下：

```
protected void do_button_actionPerformed(ActionEvent e) {
    String text = chineseArea.getText();  // 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++) {          // 遍历字符串中每一个字符
        // 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
        }
    }
    numField.setText(amount + "");                     // 在文本框中显示汉字数量
}
```

DIY：修改程序，输出文本中非汉字的所有字符的数量。（20 分）（实例位置：光盘\mr\ym\7.20\diy）

情景应用 DIY 栏目分数统计：

DIY 题目	1	2	3	4	5	总分数	
分数							

7.9　自 我 测 试

一、选择题（每题 10 分，5 道题）

1．获取字符串长度应该使用下列哪个方法？（　　）

A．length　　B．length()　　C．codePointAt()　　D．chatAt()

2．当编译和运行下列代码时会发生什么？（　　）

```
public class Test {
    public static void main(String[] args) {
        System.out.println("" + 2 + 3);
        System.out.println(2 + "" + 3);
        System.out.println(2 + 3 + "");
    }
}
```

A．编译错误　　B．运行错误　　C．输出 23、23 和 5　　D．输出 5、5 和 23

3．如果 String text = “MingRiSoft”;，则以下哪些语句是合法的？（　　）

A．text >> = 3;　　B．text[3] = ‘a’　　C．int length = text.length();　　D．text+12345;

4．当编译和运行下列代码时会发生什么？（　　）

```
public class Test {
    public static void main(String[] args) {
        String text = "ABCD";
        text += "E";
        text.replace("C", "F");
        System.out.println(text);
    }
}
```

A．编译错误　　B．输出 ABCEDF　　C．输出 ABFDE　　D．输出 ABCDE

5．在编译和运行下列代码时会发生什么？（　　）

```
public class Ref {
    public static void main(String args[]) {
        String str = "123456789";
        char mychar = str.charAt(9);
        System.out.println(mychar);
    }
}
```

A．编译错误　　B．输出 9　　C．输出 123456789　　D．运行时出现错误

二、填空题（每题 10 分，5 道题）

1．使用 String 类的_____方法可以让字符串 ABCdef 和字符串 abcDEF 比较的返回值是 true。

2．以下代码的运行结果是_____。

```
public class Test {
    public static void main(String[] args) {
        String text1 = "MingRi";
        String text2 = " Soft";
        System.out.println(text1 + text2.replace("Ri", "ri"));
    }
}
```

3．\\D*这个正则表达式的作用是_____。

4．StringBuilder 类的 append()方法的作用是_____。

5．String 类的_____方法可以判断字符串与指定的表达式是否匹配。

测试分数统计：

类别	第 1 题	第 2 题	第 3 题	第 4 题	第 5 题
选择题分数					
填空题分数					
总分数					

7.10 行 动 指 南

开始日期：_____年____月____日　　结束日期：_____年____月____日

序号	内　容	行 动 指 南	
1	照猫画虎栏目	分数>75 分	优秀，基本功掌握得不错，加油！
		75 分>分数>50 分	及格，知识掌握得不牢，重新做一遍照猫画虎。
	分数（　　）	分数<50 分	请使用光盘中提供的“实战能力测试系统”进行提高训练。
	情景应用栏目	分数>75 分	优秀，综合应用能力很强。
		75 分>分数>50 分	及格，综合应用能力需提高，再练一遍情景应用。
	分数（　　）	分数<50 分	请使用光盘中提供的“实战能力测试系统”进行提高训练。
	自我测试栏目	分数>75 分	优秀，有成为编程高手的潜质。
	分数（　　）	分数<75 分	请使用光盘中提供的“实战能力测试系统”进行提高训练。
	综合评价	反复训练后，以上各项分数都在 75 分以上，方可进入下一堂课学习。	

续表

序号	内　容	行 动 指 南
2	编程能力培养：本栏目提供了一些实践题目，对于培养编程能力很有效，要做好。	（1）在控制台接收用户输入，然后告诉用户输入了多少个字符。
		（2）在程序中定义字符串变量，并输出该变量的内容，然后让用户输入查找字符，程序经过运算后输出该字符在字符串中首次出现的位置。
		（3）把当前日期格式化为“2010-12-05”格式，并输出到控制台。
		（4）把数字组成的字符串分割为数组，以零或多个空格作为分隔符。提示：需要结合正则表达式。
3	编程习惯培养：本堂课培养读者在学习开发中记笔记的习惯，把开发中遇到的问题、总结的经验记录下来。	
4	创新能力培养：看看身边有没有可以用编程解决的问题，记录到右边的表格中。根据学习进度，尝试编写解决这些问题的小程序。	

7.11　成功可以复制——IT 风云人物鲍岳桥

出生于 1967 年的鲍岳桥在杭州大学数学系读书的时候就开始迷恋计算机。那时学校里学生上机的机会很少，鲍岳桥想方设法和机房看门的教师搞好关系，终于弄到了一份机房管理员的差事，这样鲍岳桥基本上所有的时间都可以上机了，大四时，他几乎天天泡在机房，这一段时间的“恶补”为他将来做软件打下了基础。

1989 年，鲍岳桥大学毕业分到了杭州一个橡胶厂。就是在这个橡胶厂，鲍岳桥开发出了 FOXBASE 反编译软件、普通码中文输入系统和 PTDOS，开始在国内软件编程领域崭露头角。

1998 年初，鲍岳桥离开北京电脑公司，与简晶等人组建了北京联众电脑技术有限公司。北京西北郊外马连洼，鲍岳桥、简晶、王建华三人挤在只有两个房间的联众公司里，一边写程序，一边交流写程序的心得。当时只有王建华的 Windows 编程经验多一些，鲍岳桥和简晶的工作就是学习，边学习边工作。为了提高效率，简晶将家搬到了办公室附近，鲍岳桥和王建华一人弄一个木兰小摩托，跑来跑去，风尘仆仆。工作是从 1998 年的大年初二开始的，联众的框架设计用了将近两个月的时间，完全基于 NT 平台。鲍岳桥称，这个框架从一开始就考虑得很完善，之后的几次升级基本没有再做改动。接下来，王建

华负责服务器端编程，鲍岳桥负责“游戏大厅”的开发，简晶负责具体游戏的设计。1998 年 6 月 4 日，联众游戏开通。没人知道这个网站，一个来玩的人都没有。三个人就发挥自身力量四处找网友，拉他们过来看看。在种种努力下联众的人气越来越旺。联众成立 3 年，同时在线人数每年都以 9 倍的速度向前滚动：1999 年初 1000 人同时在线，2000 年初 9000 人同时在线，2001 年初达到了 8 万人。注册用户方面，1999 年初是 3 万人，2000 年初是 70 万人，2001 年初是 700 万人。鲍岳桥称，棋牌类网站，联众全球第一。

经典语录

我个人感觉我不是一个很聪明的人，但我是一个非常投入的人，我做一件事时整天头脑里就想着这一件事。我始终相信就算你笨一点，只要你投入的精力比别人多很多，就会比别人做得好。

深度评价

不害怕吃苦、投入地去做是鲍岳桥成功的原因，这也正是我们编程人员应该具备的品质。勤能补拙是良训，哪怕我们天资不如人，只要我们热爱编程，坚持去做，就会取得应有的硕果。

提高篇

- 第 8 堂课　类的继承与多态特性
- 第 9 堂课　其他类特性与异常处理
- 第 10 堂课　Swing 编程基础
- 第 11 堂课　多线程编程
- 第 12 堂课　事件处理的应用
- 第 13 堂课　常用工具类
- 第 14 堂课　常用集合类

第 8 堂课

类的继承与多态特性

（视频讲解：172 分钟）

继承、多态和封装是 Java 面向对象的三大特征，它们是面向对象程序开发的重要环节，如果在程序中使用得当，这三大特征能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完好的类，减少重复代码的编写。多态机制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 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。正因为使用了这些机制，才使 Java 语言更具有生命力。

学习摘要：

- 类的继承
- 方法的重写
- 修饰符
- 封装的使用
- Object 类
- instanceof 操作符的使用
- 多态机制
- 抽象类与接口

8.1 类的继承

继承在面向对象开发思想中是一个非常重要的概念，通过继承，可以使用以前定义的类的成员方法和成员变量，经过简单的程序编码就可以构建功能强大的新类。这样节约了很多编程时间，同时减少了出错的机会，也可以提高软件的可维护性、可扩展性。本堂课将详细讲解类的继承。

继承可以理解为现实世界中的“是一种（is-a）”关系，例如，轿车是一种汽车，而汽车又是一种交通工具，如图 8.1 所示。

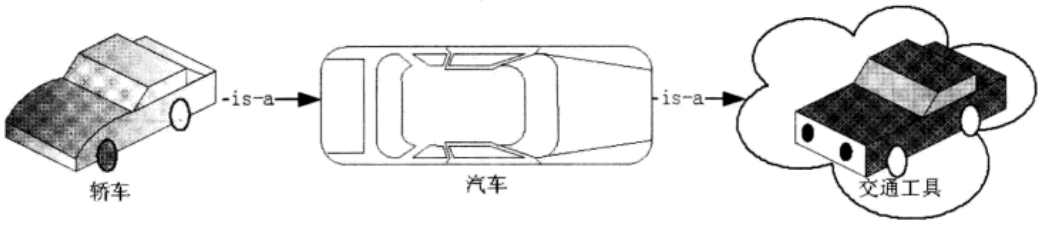


图 8.1 is-a 关系

应用 is-a 关系可以通过已知事物去认识未知事物，从而提高认知新事物的效率。Java 程序设计语言同样可以通过继承机制在现有类的基础上去定义一个新的类，并在原有类的基础上添加新的方法或修改原有方法，从而提高编程效率、减少错误几率。

实现继承是通过 extends 关键字在声明类时指定其父类，其声明格式如下：

[修饰符] class 类名 extends 父类名

参数：修饰符是可选参数，它是权限修饰符，用于限定类的访问级别，extends 父类名用于指定继承的父类，其中父类又可以称为超类，它可以是任何可继承的类。例如：

```
class SaloonCar extends Car{
    ……类体
}
```

对于这段代码，应该理解为 SaloonCar 类继承了 Car 类，所以 Car 是 SaloonCar 类的直接父类（或直接超类），而 SaloonCar 是 Car 类的直接子类。

例 8.01 在项目中分别创建 Car 类和 SaloonCar 类，在 Car 类中编写汽车类的部分变量与方法。SaloonCar 类使用 extends 关键字继承 Car 类，不用编写类体代码，也就是一个空类。然后编写 Program 类，在该类的主方法中创建 SaloonCar 类的实例对象，注意不是 Car 类而是空的 SaloonCar 类。然后设置该实例对象的属性，并执行相应的方法，看空的 SaloonCar 类是否继承并拥有父类的成员。**（实例位置：光盘\mr\ym\8.01）**

（1）编写 Car 类，它将作为 SaloonCar 类的父类，也就是被继承的一方，代码如下：

```
public class Car {
    public String color = "黑色";                    // 颜色
    public int speed = 1;                            // 速度
    public int level;                                // 档位
    public void gear(int num) {
        level = num;                                 // 换档
        speed = 10 * level * level;                  // 变速
    }
    public void drive() {
```

```
        System.out.println("我是一辆" + color + "的轿车。");
        System.out.println("现在以时速" + speed + "公里每小时的速度行驶中。");
        System.out.println("车档位是：" + level + "档。");
    }
}
```

（2）使用 Eclipse“新建 Java 类”向导，创建子类。在这个向导中指定包、类名的同时，还可以指定继承的超类（也就是父类）和实现的接口，如图 8.2 所示。而继承的超类正是子类 SaloonCar 要实现的，所以使用向导可以直接生成继承的代码。

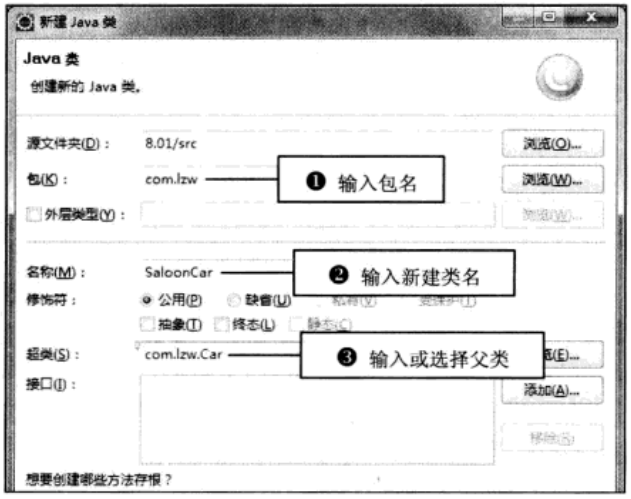


图 8.2　新建类向导窗口部分截图

向导生成的 SaloonCar 类代码如下，本例不做任何修改，因为要证明空的子类是否能继承父类的成员变量与成员方法。

```
package com.lzw;

public class SaloonCar extends Car {

}
```

建议读者使用代码输入方式对该步骤做几次练习，以巩固继承的编写格式。

（3）编写程序的 Program 类，它是本实例的主类，在该类的主方法中创建 SaloonCar 类的实例对象，注意这里创建的不是父类 Car 的实例对象。然后设置该对象的属性，并执行相应的方法，而这些都是父类中定义的。运行程序，查看子类是否继承了这些成员变量与成员方法，代码如下：

```
public class Program {
    public static void main(String[] args) {
        SaloonCar whiteCar = new SaloonCar();                    // 创建 Car 类的实例对象
        whiteCar.color = "红色";                                  // 修改该实例中的 color 属性
        whiteCar.gear(4);                                        // 调用 gear()方法修改档位和车速
        whiteCar.drive();                                        // 执行开车方法
    }
}
```

实例运行结果如图 8.3 所示。

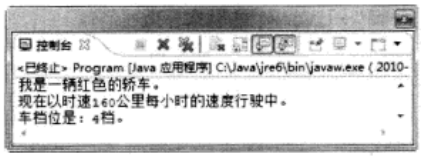


图 8.3　实例运行结果

从实例的运行结果可以看出，SaloonCar 类虽然没有在类体中定义任何成员变量与成员方法，但是它还是从父类继承了它应该拥有的成员。

8.2　方法的重写与 Super 关键字

继承某个父类而生成新的子类，不但拥有父类的变量与方法，还可以为子类添加新的成员变量和成员方法，增强父类的功能，也就是所谓的扩展。甚至还可以在子类中为父类的某个方法定义多个重载方法，增加该类的灵活性。但是有时父类的方法并不能完全适应于子类，或者子类需要有不同的行为。例如，父类的 drive()方法是输出开车的描述信息，而子类有可能要添加更多的功能，或者要丰富 drive()方法，在开车时要检测油箱的油量是否充足，如果油箱是空的，那么是无法使用 drive()的。

在这种情况下，子类可以重写父类的某个方法，或者说是覆盖父类的某个方法，只要在子类中定义与父类相同的方法即可。但是要注意方法的声明一定要和父类的方法声明一样，或者干脆把父类某方法的声明代码直接复制过来，去掉方法体，重新编写业务代码。

例如，在子类 SaloonCar 中重写父类的 drive()方法，代码如下：

```
public void drive() {// 重写父类的方法
    ……// 省略部分代码
    System.out.println("车子没有油了，无法驾驶。");
}
```

另外，还可以使用 super 关键字引用父类的方法，然后再添加新的业务代码。

例如：

```
public void drive() {                                    // 重写父类的方法
    if (oil > 0) {                                       // 如果油箱不是空的
        super.drive();                                   // 执行父类的 drive()方法
    } else {                                             // 否则，油箱是空的
        System.out.println("车子没有油了，无法驾驶。");
    }
}
```

这段代码重写了父类的 drive()方法，但是在 oil 变量大于 0 时，又需要执行父类的 drive()方法，所以使用了 super 关键字去访问父类的 drive()方法，这样子类就不必关心父类的 drive()方法是如何实现的。这段代码等价于：

```
public void drive() {                                                    // 重写父类的方法
    if (oil > 0) {                                                       // 如果油箱不是空的
        System.out.println("我是一辆" + color + "的轿车。");
        System.out.println("现在以时速" + speed + "公里每小时的速度行驶中。");
        System.out.println("车档位是：" + level + "档。");
    } else {                                                             // 否则，油箱是空的
```

```
            System.out.println("车子没有油了，无法驾驶。");
        }
    }
```

例 8.02　修改实例 8.01，在空的 SaloonCar 类中重写父类的 drive()方法，然后再次运行程序。**（实例位置：光盘\mr\ym\8.02）**

（1）在 Java 编辑器中，把光标定位在类体中，单击鼠标右键，在弹出的快捷菜单中选择“源代码”/“覆盖/实现方法”命令，如图 8.4 所示。

（2）在弹出的“覆盖/实现方法”对话框中列出了父类的方法，选择要重写的父类方法，然后选择一个插入点，并选中“生成方法注释”复选框，如图 8.5 所示，单击“确定”按钮。

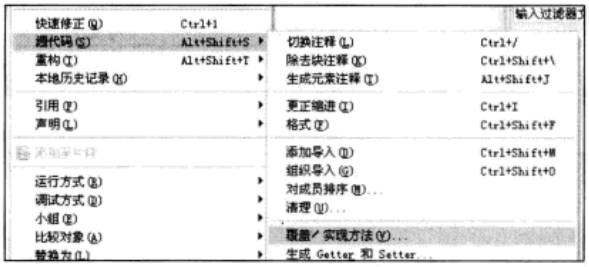


图 8.4　编辑器中的弹出菜单

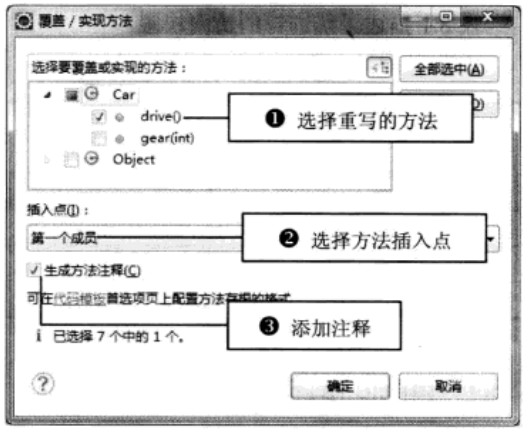


图 8.5　“覆盖/实现方法”对话框

（3）重写父类的 drive()方法，并在 SaloonCar 类中添加成员变量 oil，它在 drive()方法中用于判断油量多少，代码如下：

```
public class SaloonCar extends Car {
    public int oil = 16;
    @Override
    public void drive() {                                    // 重写父类的方法
        if (oil > 0) {                                       // 如果油箱不是空的
            super.drive();                                   // 执行父类的 drive()方法
            if (oil < 20) {                                  // 如果油箱不是空的，并且油量低于 20%
                System.out.println("油量不足 20%，请注意加油。");
            }
        } else {                                             // 否则，油箱是空的
            System.out.println("车子没有油了，无法驾驶。");
        }
    }
}
```

还可以在子类的构造方法中使用 super 关键字执行父类的构造方法，其语法格式如下：

```
super([构造参数列表])
```

参数：构造参数列表是可选参数，根据调用构造方法的形参声明来填充。例如：

```
public SaloonCar() {                    // 子类构造方法
    super();                            // 父类构造方法
}
```

Car 类没有定义默认的构造方法，编译器会自动为它创建一个隐藏的无参数的默认构造方法，这段代码在子类的构造方法中重写了这个构造方法。

8.3 修 饰 符

Java 中定义了 private、public、protected 和默认的权限修饰符，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规则。另外，还可以辅助 static 和 final 关键字定义特殊规则。本节将介绍这 4 种权限修饰符和它们的访问权限。

表 8.1 描述了修饰符的访问权限。

表 8.1 Java 语言中的修饰符

类的位置	private	protected	默认修饰符	public
本类	可见	可见	可见	可见
同包其他类	不可见	可见	可见	可见
其他包的类	不可见	不可见	不可见	可见
同包的子类	不可见	可见	可见	可见
其他包的子类	不可见	可见	不可见	可见

8.3.1 public 修饰符

public 是公有类型的权限修饰符，也就是说，使用 public 修饰的类、成员变量和成员方法，其他类都可以访问，包括任意包中的任意类以及子类。

例如，之前章节的所有实例都使用到了 main()主方法，该方法的权限修饰符就使用了 public，其声明代码如下：

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

该方法必须声明为 public 权限，因为这个方法必须能够被虚拟机访问以执行程序。

之前的实例中所有成员变量和成员方法都是使用 public 修饰符定义的，这里不再以新的实例做介绍。

8.3.2 private 修饰符

private 是私有权限修饰符，只有本类，也就是定义 private 私有成员的类能够访问，对于其他方式的访问，它都会拒绝，即“不要碰我”。

如果一个类的成员变量或成员方法被修饰为 private，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。

例 8.03 定义一个 Superclass 类，在该类中定义 private 修饰的私有方法 print()。然后创建子类与其他类进行访问，并把鼠标停留在有错误的代码位置，查看 Eclipse 的 Java 编译器提示的错误信息。**（实例位置：光盘\mr\ym\8.03）**

（1）首先创建 Superclass 类，代码如下：

```
public class Superclass {
    private void print() {
```

```
        System.out.println("我是 Superclass 类的 print 方法");
    }
}
```

（2）再创建 SubClass 子类，在该类中访问超类的 print()方法，Eclipse 的 Java 编辑器提示错误，如图 8.6 所示。

（3）再创建 OtherClass 类，在该类中创建 Superclass 类的实例对象，并访问该对象的 print()方法，Eclipse 的 Java 编辑器提示错误，如图 8.7 所示。

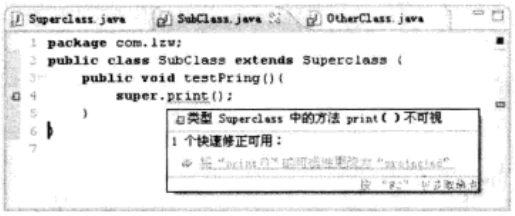


图 8.6　子类访问超类 private 成员

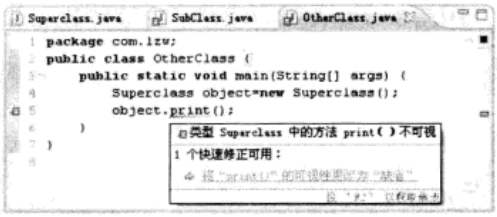


图 8.7　其他类访问类中 private 成员

8.3.3　protected 修饰符

protected 是保护级别的权限修饰符，它保护成员不会被其他包或非子类访问。也就是说 protected 修饰的成员，只能被子类（可以不是直接子类，即间接继承的子类也可以）和同一个包中定义的其他类访问。

例 8.04　定义一个 Superclass 类，在该类中定义 protected 修饰的方法 print()，然后创建子类与其他类进行访问，并把鼠标停留在有错误的代码位置，查看 Eclipse 的 Java 编译器提示的错误信息，或者查看正确的运行结果。**（实例位置：光盘\mr\ym\8.04）**

（1）首先创建 Superclass 类，代码如下：

```
public class Superclass {
    protected void print() {
        System.out.println("我是 Superclass 类的 print 方法，我使用 protected 修饰符");
    }
}
```

（2）再创建 SubClass 子类，它继承 Superclass 类，在该类中访问超类的 print()方法，代码如下：

```
public class SubClass extends Superclass {
    public static void main(String[] args) {
        SubClass obj = new SubClass();
        System.out.println("子类 SubClass 调用父类 Superclass 的 print 方法：");
        obj.print();
    }
}
```

实例运行结果如图 8.8 所示。

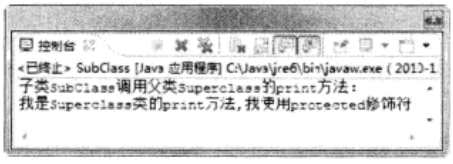


图 8.8　实例运行结果

（3）再创建 OtherClass 类，在该类中创建 Superclass 类的实例对象，并访问该对象的 print()方法，代码如下：

```
public class OtherClass {
    public static void main(String[] args) {
        Superclass object = new Superclass();
        System.out.println("同一个包的 OtherClass 类调用 Superclass 类的 print 方法：");
        object.print();
    }
}
```

实例运行结果如图 8.9 所示。

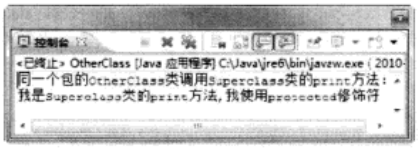


图 8.9　实例运行结果

（4）在其他包中创建一个 OtherPackageClass 类，在该类中创建 Superclass 类的实例对象，并访问该对象的 print()方法，Eclipse 的 Java 编辑器提示错误，如图 8.10 所示。

图 8.10　其他包的类访问 Superclass 类成员的错误信息

8.3.4　默认权限修饰符

当不添加任何权限修饰符时，编译器会使用默认的修饰符，该修饰符的权限级别与 protected 类似，不同之处在于，在其他包定义的子类无法访问父类默认权限修饰的成员。

例 8.05　定义一个 Superclass 类，在该类中定义没有修饰符的方法 print()。然后在其他包中创建该类的子类，在子类中访问父类的 print()方法，并把鼠标停留在有错误的代码位置，查看 Eclipse 的 Java 编译器提示的错误信息，或者查看运行结果。**（实例位置：光盘\mr\ym\8.05）**

（1）首先创建 Superclass 类，注意包的位置是 com.lw，代码如下：

```
package com.lw;

public class Superclass {
    void print() {
        System.out.println("我是 Superclass 类的 print 方法，我使用 protected 修饰符");
    }
}
```

（2）再创建 SubClass 子类，它继承 Superclass 类，并且定义在与父类不同的包中。在该类中访问超类的 print()方法，Eclipse 的 Java 编辑器提示错误，如图 8.11 所示。

图 8.11　不同包的子类不能访问父类默认权限修饰的成员

8.4　封　　装

Java 是面向对象的编程语言，而继承、多态与封装是面向对象的三大特性，本节将要介绍的封装是指把对象的属性隐藏在对象的内部，不允许外部直接访问和修改。例如，之前的实例就直接使用“whiteCar.color = "红色";”之类的代码直接修改 color 属性的值，应该避免这种情况，而使用该类定义的其他方法来实现内部属性的访问与修改。Java 规范中的设置器与访问器就是实现封装的标准方法，它们用于获取和设置对象的属性值。

8.4.1　把属性隐藏

要实现封装，第一步就是设置类的成员变量（即对象的属性）使用 private 修饰符，使其他类无法直接访问该成员变量，以防止外部直接访问和修改，例如：

```
public class Car {
    private String color = "黑色";          // 颜色
    private boolean running;                // 行驶状态
}
```

8.4.2　定义设置器

把成员变量设置为 private 私有权限之后，其他类就不能访问了，必须通过本类定义的设置器方法来设置和修改该成员变量的值，设置器方法的命名一般以 set 作为前缀，属性名作为后缀，setColor()方法从名称上就能够理解，它用于设置 color 属性值，例如：

```
public void setColor(String color) {
    this.color = color;                     // 把参数 color 赋值给成员变量 color
}
```

public 是方法的权限修饰符，对于设置器来说，它需要暴露给其他类，使其可以访问，所以使用 public 修饰符。参数列表定义了 String 类型的 color 参数，在方法体中为区别参数 color 与成员变量 color，使用了 this 关键字引用成员变量，并赋值为 color 参数的值，也可以使用与成员变量不同名称的参数，上述代码为了演示重名情况而使用了 this 关键字。

以后设置对象的 color 属性就可以使用 setColor()方法完成，例如：

```
Car whiteCar = new Car();                   // 创建 Car 类的实例对象
whiteCar.setColor("红色");                  // 修改该实例中 color 属性
```

📖 **说明**：如果不按规范做的话，可以定义任何方法实现属性的设置与修改，但是建议大家采用统一的规范编程，使用设置器实现属性的修改，增加程序的可读性。

8.4.3 定义访问器

和设置器一样，因为成员变量被设置为 private 私有权限，相对于其他类，就隐藏了这个成员变量，所以要使用访问器方法读取对象的属性值，访问器以 get 或 is 作为方法名称的前缀，以属性名作为后缀。例如，color 属性的访问器方法代码如下：

```
public String getColor() {
    return color;                    // 返回 color 属性值
}
```

对于 boolean 类型的属性，应该使用 is 前缀定义访问器方法，如 running 属性访问器方法的代码如下：

```
public boolean isRunning() {
    return running;                  // 返回 running 属性值
}
```

注意：要注意 is 和 get 作为前缀的访问器的用法，它们分别用于布尔类型和其他类型属性的访问。

8.4.4 使用 Eclipse 完成封装

Eclipse 拥有最强的代码辅助功能，使用向导可以自动为选择的成员变量添加设置器与访问器（或者是 get/set 方法）。

例 8.06 定义 Car 类，在该类中定义字符串 color 和布尔类型的 running 两个成员变量，并设置它们的访问器与设置器方法。**（实例位置：光盘\mr\ym\8.06）**

（1）在该类的编辑器中单击鼠标右键，在弹出的快捷菜单中选择“源代码”/“生成 Getter 和 Setter”命令，或者按 Shift+Alt+S/R 快捷键。这时将弹出“生成 Getter 和 Setter”对话框，该对话框就是生成访问器与设置器的向导。在该对话框中首先需要选择要创建的访问器与设置器，也可以单击右侧的“全部选中”按钮选择生成所有成员变量的访问器与设置器方法，然后需要选择生成方法的插入点和修饰符，最后选择是否添加方法注释，如图 8.12 所示。

图 8.12 访问器与设置器生成向导

如果没有特殊要求，使用默认值生成访问器与设置器即可，笔者只单击“全部选中”按钮和“确定”按钮就完成了访问器与设置器代码的生成。生成的程序代码如下所示，用户可以完善程序注释，或修改代码。

```
public class Car {
    private String color = "黑色";
    private boolean running;
    public void setColor(String color) {
        this.color = color;
    }
    public boolean isRunning() {
        return running;
    }
    public void setRunning(boolean running) {
        this.running = running;
    }
    public String getColor() {
        return color;
    }
}
```

可以不修改英文注释，但是本书为方便初学者，对生成的代码添加中文注释，最终修改完成的程序代码如下：

```
public class Car {
    private String color = "黑色";                          // 颜色
    private boolean running;                                // 行驶状态
    public void setColor(String color) {
        this.color = color;                                 // 把参数 color 赋值给成员变量 color
    }
    public String getColor() {
        return color;                                       // 返回 color 属性值
    }
    public boolean isRunning() {
        return running;                                     // 返回 running 属性值
    }
    public void setRunning(boolean running) {
        this.running = running;                             // 设置成员变量 running
    }
}
```

（2）编写实例的主类，演示访问器与设置器的使用，代码如下：

```
public class Program {
    public static void main(String[] args) {
        Car whiteCar = new Car();                           // 创建 Car 类的实例对象
        whiteCar.setColor("红色");                          // 修改该实例中 color 属性
        whiteCar.setRunning(true);                          // 设置 car 对象的行驶状态
        System.out.println("color 属性：" + whiteCar.getColor());
        System.out.println("running 属性：" + whiteCar.isRunning());
    }
}
```

实例运行结果如图 8.13 所示。

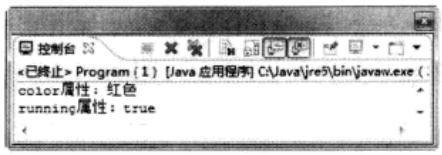


图 8.13　实例运行结果

8.5　Object 类

在 Java 中，所有的类都直接或间接继承了 java.lang.Object 类。Object 类是比较特殊的类，它是所有类的父类，也就是说 Java 语言中任何一个类都是 Object 类的子类。当创建一个类时，如果没有使用 extends 关键字继承指定的类，那么编译器总是默认直接继承 Object 类，如果指定继承其他类，那么也会间接继承 Object 类。

例如，String、Integer 等类都是继承于 Object 类，除此之外自定义的类也都继承于 Object 类，由于所有类都是 Object 子类，所以在定义类时，省略了 extends Object 关键字，图 8.14 中描述了这一原则。

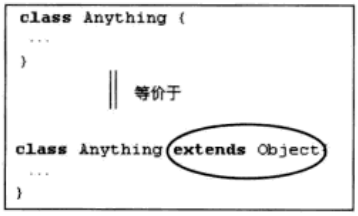


图 8.14　定义类时可以省略 extends Object 关键字

在 Object 类中主要包括 clone()、finalize()、equals()、toString()等方法，其中常用的两个方法为 equals()和 toString()。由于所有的类都是 Object 类的子类，所以任何类都可以重写 Object 类中的方法。

注意：Object 类中的 getClass()、notify()、notifyAll()、wait()等方法不能被重写，因为这些方法被定义为 final 类型。

8.5.1　equals 方法

该方法是在 Object 类中定义的，用于判断两个对象是否相同，大部分子类都重写了该方法，用于比较指定类型的对象，如 String、Integer 等都重写了该方法。如果 Object 的子类没有重写 equals()方法，该方法将默认使用“==”运算符判断两个对象。

在 Object 类中 equals()方法的原始定义代码如下：

```
public boolean equals(Object obj) {
    return (this == obj);
}
```

例 8.07　创建自定义的类 TempClass，用于测试，再创建 TestEquals 类，在该方法中创建 String 类的两个实例对象 s1 和 s2，再创建 TempClass 类的两个实例对象 v1 和 v2，分别输出 s1 与 s2、v1 与 v2 的 equals()方法调用结果。**（实例位置：光盘\mr\ym\8.07）**

（1）创建 TempClass 类，它是一个空类，以默认方式继承 Object，而且没有重写 equals 方法，具体代

码如下：

```
package com.lw;

public class TempClass {

}
```

（2）创建 TestEquals 类，该类拥有主方法，代码如下：

```
public class TestEquals {
    public static void main(String[] args) {
        String s1 = "123";                                          // 实例化两个对象，内容相同
        String s2 = "123";
        System.out.println("s1 equals s2 :" + s1.equals(s2));       // 使用 equals()方法调用
        TempClass v1 = new TempClass();                             // 实例化两个对象
        TempClass v2 = new TempClass();
        System.out.println("v1 equals v2 :" + v1.equals(v2));       // 使用 equals()方法比较 v1 与 v2 对象
    }
}
```

在 Eclipse 中运行本实例，运行结果如图 8.15 所示。

图 8.15　使用 equals()方法比较两个对象

从本实例的运行结果可以看出，在自定义的类中使用 equals()方法进行比较时，将返回 false，因为 equals()方法默认实现的是使用“==”运算符比较两个对象的引用地址，而不是比较对象的内容，所以要想真正做到比较两个对象的内容，需要在自定义类中重写 equals()方法。

8.5.2　toString 方法

Object 类中 toString()方法的功能是将一个对象转换为字符串形式，当一个对象参与字符串的“+”连接符操作时，也会调用该方法把对象转换成字符串，然后再和另一个字符串操作数连接。toString()方法会返回一个 String 实例，在实际应用中通常重写 toString()方法，为对象提供一个特定的输出模式。当这个类转换为字符串或与字符串连接时，将自动调用重写的 toString()方法。

例 8.08　在项目中创建 ObjectInstance 类，在类中重写 Object 类的 toString()方法，并在主方法中输出该类的实例对象，代码如下：**（实例位置：光盘\mr\ym\8.08）**

```
public class ObjectInstance {
    public String toString() {                                      // 重写 toString()方法
        return "大家好，我是 ObjectInstance 类的实例对象。";
    }
    public static void main(String[] args) {
        ObjectInstance object = new ObjectInstance();
        System.out.println("输出 ObjectInstance 实例对象：\n" + object);      // 打印本类对象
    }
}
```

在 Eclipse 中运行本实例，运行结果如图 8.16 所示。

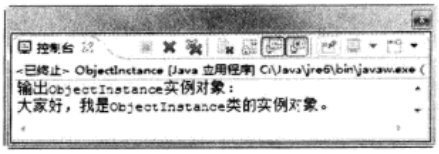


图 8.16　在类中重写 toString()方法

在本实例中重写父类 Object 类的 toString()方法，定义一段输出字符串，当用户打印 ObjectInstance 类对象时，将自动调用 toString()方法。

8.6　照猫画虎——基本功训练

8.6.1　基本功训练 1——多态数组

视频讲解：光盘\mr\lx\08\多态数组.exe

实例位置：光盘\mr\ym\8.09\src

多态数组就是数组使用父类声明，而数组中的元素都是子类的实例对象，它们有父类同名的方法，但是方法实现却可以不一样。也许通过实例来描述会使读者更清楚一些。

创建 Quadrangle 四边形类，作为其他类的父类。再创建 Square 正方形类、Rectangle 长方形类和 Parallelogram 平行四边形类，这些类都继承 Quadrangle 类并重写 draw()方法。使用 Quadrangle 类，也就是父类，定义一个四边形的数组。数组的元素分别是每个子类的实例对象，在 Foreach 循环中遍历数组，调用每个数组元素的 draw()方法。实现过程如下：

（1）创建 Quadrangle 四边形类，为该类定义 draw()方法，该类将作为其他类的父类，实例代码如下：

```
/**
 * 这是一个四边形类，它在本实例中作为正方形、长方形与平行四边形类的父类
 * @author Li Zhong Wei
 */
public class Quadrangle {
    /**
     * 绘图方法
     */
    public void draw(){
        System.out.println("绘制自身图形");
    }
}
```

（2）创建 Square 正方形类，该类将继承 Quadrangle 四边形类，并重写 draw()方法实现自己的绘制能力，实例代码如下：

```
/**
 * 这是一个正方形类，它继承 Quadrangle 四边形类。
 * @author Li Zhong Wei
 */
public class Square extends Quadrangle {
    /**
     * 正方形的绘图方法。
     */
```

```
    @Override
    public void draw() {
        System.out.println(" ------------------");
        System.out.println("|                  |");
        System.out.println("|     正方形      |");
        System.out.println("|                  |");
        System.out.println("|                  |");
        System.out.println(" ------------------");
    }
}
```

（3）创建 Rectangle 长方形类，该类将继承 Quadrangle 四边形类，并重写 draw()方法实现自己的绘制能力，输出一个长方形，实例代码如下：

```
public class Rectangle extends Quadrangle {
    /**
     * 长方形的绘图方法。
     */
    @Override
    public void draw() {
        System.out.println(" --------------------------");
        System.out.println("|                          |");
        System.out.println("|        长方形           |");
        System.out.println("|                          |");
        System.out.println(" --------------------------");
    }
}
```

（4）创建 Parallelogram 平行四边形类，该类将继承 Quadrangle 四边形类，并重写 draw()方法实现自己的绘制能力，输出一个平行四边形，实例代码如下：

```
/**
 * 这是一个平行四边形类，它继承 Quadrangle 四边形类。
 * @author Li Zhong Wei
 */
public class Parallelogram extends Quadrangle {
    /**
     * 平行四边形的绘图方法。
     */
    @Override
    public void draw() {
        System.out.println("          ----------------------");
        System.out.println("      /                      /");
        System.out.println("    /          平行        /");
        System.out.println("  /          四边形      /");
        System.out.println(" /                        /");
        System.out.println(" ----------------------");
    }
}
```

（5）创建本实例的主类 Demo，在该类的主方法中演示多态数组的定义和使用，实例代码如下：

```
/**
 * 这是本实例的主类，它将演示 instanceof 的用法
 * @author Li Zhong Wei
 */
```

```
public class Demo {
    public static void main(String args[]) {
        Quadrangle[]  quadrangles= new Quadrangle[3];    // 创建四边形数组
        quadrangles[0]=new Square();                     // 初始化第一个元素为正方形对象
        quadrangles[1]=new Rectangle();                  // 初始化第二个元素为长方形对象
        quadrangles[2]=new Parallelogram();              // 初始化第三个元素为平行四边形对象
        for (Quadrangle obj : quadrangles) {             // 遍历数组每个元素
            obj.draw();                                  // 执行每个元素的 draw()方法
        }
    }
}
```

程序运行结果如图 8.17 所示。

图 8.17　实例运行结果

你问我答：什么是多态？

多态是 Java 面向对象语言的三大特性之一，多态是指程序中同一种操作在不同环境中有不同的语义解释。要了解多态，首先要熟悉类的继承、向上转型、方法的重写与重载等知识点。向上转型可以声明父类的引用变量，但是赋值为子类的实例对象，也就是对象变量的声明与赋值可以不是一个类型。例如，把子类对象赋值给父类的对象变量：

```
Quadrangle obj = new Square();                    // 声明父类对象，初始化为子类
```

声明的变量 obj 是 Quadrangle 类型的，而赋值时却是 Square 类型的实例变量。通过使用父类类型的引用，可以定义多态数组或多态方法等。

照猫画虎：创建一个多态动物数组，然后调用数组所有元素的 bellow()方法，让动物们发出吼叫声。（20 分）（实例位置：光盘\mr\ym\8.09\zmhh）

8.6.2　基本功训练 2——参数多态

视频讲解：光盘\mr\lx\08\参数多态.exe

实例位置：光盘\mr\ym\8.10\src

多态是 Java 面向对象的一个重要特性，为了提高读者的学习兴趣，下面以一个笑话为题讲解一个多态的实例。主人养了 3 只爱喝酒的小老鼠，它们分别是白鼠、灰鼠和花鼠，有一天，主人买来了两瓶好酒。这酒很贵，主人没有喝过，想让 3 只小老鼠先试试酒劲有多大。可是这 3 只小老鼠喝酒后的反应都不一样，

通过下面的实例阐述这些小老鼠喝酒以后分别是什么反应。

分别创建主人类、鼠类和 3 个试酒的鼠类，使用 Java 的多态性实现主人给 3 只老鼠喂酒的动作，并分别输出每只老鼠喝酒后的反应。另外，3 个试酒的鼠类都继承 Mouse 鼠类，如图 8.18 所示。

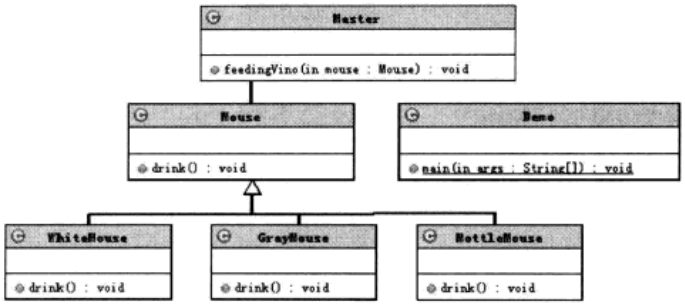


图 8.18　本实例类图

实现过程如下：

（1）创建 Mouse 鼠类，并定义 drink()饮酒方法，代码如下：

```
package com.mouse;
/**
 * Mouse 是鼠类，该类定义了 drink 方法
 * @author Li Zhong Wei
 */
public class Mouse {
    /**
     * 鼠类的饮酒方法
     */
    public void drink() {
        System.out.println("动物类的饮酒方法");
    }
}
```

（2）继承 Mouse 类创建小白鼠类 WhiteMouse，并重写父类的 drink()方法，实现小白鼠喝酒后的状态输出，代码如下：

```
package com.mouse;
/**
 * 小白鼠类
 * @author Li Zhong Wei
 */
public class WhiteMouse extends Mouse {
    /**
     * 小白鼠的饮酒方法，重写的父类方法
     */
    @Override
    public void drink() {
        System.out.println("小白鼠把酒一口喝下。");
        System.out.println("反应：");
        System.out.println("\t 睡着了，主人对这酒很满意。");
        System.out.println("================================");
    }
}
```

（3）继承 Mouse 类创建小灰鼠类 GrayMouse，并重写父类的 drink()方法，在该方法中实现小灰鼠喝酒后的状态反应信息，代码如下：

```
package com.mouse;
/**
 * 小灰鼠类
 * @author Li Zhong Wei
 */
public class GrayMouse extends Mouse {
    /**
     * 小灰鼠的饮酒方法，重写的父类方法
     */
    @Override
    public void drink() {
        System.out.println("小灰鼠运动一下，把酒一口喝下。");
        System.out.println("反应：");
        System.out.println("\t 脸色微红，不一会就开始跳舞。");
        System.out.println("主人感觉这酒还有点浪漫情调");
        System.out.println("================================");
    }
}
```

（4）继承 Mouse 类创建小花鼠类 MottleMouse，同样重写父类的 drink()方法，在方法体中输出小花鼠喝酒后的状态，实例代码如下：

```
package com.mouse;
/**
 * 小花鼠类
 * @author Li Zhong Wei
 */
public class MottleMouse extends Mouse {
    /**
     * 小花鼠的饮酒方法，重写的父类方法
     */
    @Override
    public void drink() {
        System.out.println("小花鼠看了看，把酒一口喝下。");
        System.out.println("反应：");
        System.out.print("\t 没什么反应，主人对这酒很失望，这么贵的酒\n，白花钱了。");
        System.out.print("……又过了一会儿，小花鼠找了一块板砖，\n");
        System.out.print("眼睛布满血丝，晃晃悠悠地走到主人面前，说道：猫呢\n");
        System.out.print("，它再欺负我们，看我不拍死丫的。\n");
        System.out.println("\t 主人目瞪口呆。");
        System.out.println("================================");
    }
}
```

（5）创建主人类 Master。它将用不同的老鼠试酒。

```
package com.lzw;                                    // 声明主人类所在包
import com.mouse.Mouse;                             // 导入老鼠类
/**
 * 这是主人类，它将用不同的老鼠试酒
 * @author Li Zhong Wei
```

```
 */
public class Master {
    /**
     * 试酒的方法，该方法接收 Mouse 类的参数，并调用 Mouse 类的 drink()方法试酒
     * @param mouse 试酒的老鼠
     */
    public void feedingVino(Mouse mouse) {
        System.out.println("主人倒满一小杯酒……");
        mouse.drink();                          // 使老鼠饮酒
    }
}
```

（6）创建 Demo 类，它是本实例的主类，用于执行程序。在主方法中分别创建 3 个鼠类的实例对象和主人类的实例对象，然后调用主人对象的 feedingVino()方法，分别以 3 个鼠类的实例对象作为方法的参数，具体代码如下：

```
import com.lzw.Master;                                  // 导入主人类
import com.mouse.GrayMouse;                             // 导入灰鼠类
import com.mouse.MottleMouse;                           // 导入花鼠类
import com.mouse.WhiteMouse;                            // 导入白鼠类
/**
 * 这是本实例的主类，用于演示 Java 的多态性，本实例主要演示参数的多态
 * @author Li Zhong Wei
 */
public class Demo {
    public static void main(String[] args) {
        WhiteMouse wMouse=new WhiteMouse();             // 创建小白鼠对象
        GrayMouse gMouse=new GrayMouse();               // 创建小灰鼠对象
        MottleMouse mMouse=new MottleMouse();           // 创建小花鼠对象
        Master master=new Master();                     // 创建主人对象
        System.out.println("试酒开始。\n==============================");
        master.feedingVino(wMouse);                     // 给小白鼠喂酒
        master.feedingVino(gMouse);                     // 给小灰鼠喂酒
        master.feedingVino(mMouse);                     // 给小花鼠喂酒
    }
}
```

实例运行结果如图 8.19 所示。

图 8.19　实例运行结果

从实例运行结果可以看出，feedingVino()方法传递的参数是多态的，不同的参数会有不同的表现。3 只试酒的小老鼠说明了这一点。

你问我答：方法如何实现多态？

方法可以通过参数实现多态，例如，定义一个 print()方法，该方法定义 Quadrangle 类型的参数，但是在调用 print()方法时可以传递子类对象的实参。

例如：

```
public void print(Quadrangle graph) {
    ……// 程序代码
}
```

另外，方法的返回类型也可以多态，例如，定义一个 getQuadrangle()方法，方法的返回值类型为 Quadrangle 类。那么方法体的 return 语句可以返回该类的任何子类。

例如：

```
public Quadrangle getQuadrangle() {
    Square graph = new Square();
    ……// 程序代码
    return graph;                          // 返回任何 Quadrangle 类的子类
}
```

照猫画虎：创建一个 Cat 猫类，在 Cat 类中定义 look()方法，通过该方法的参数多态实现输出不同老鼠看到猫的反应。（20 分）（实例位置：光盘\mr\ym\8.10\zmhh）

8.6.3 基本功训练 3——抽象类实现几何面积计算

视频讲解：光盘\mr\lx\08\抽象类实现几何面积计算.exe

实例位置：光盘\mr\ym\8.11\src

对于每个几何图形而言，都有一些共同的属性，如名字、面积等，而其计算面积的方法却各不相同。为了简化开发，可以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，本实例将演示抽象类与抽象方法的使用。

实现过程如下：

（1）编写类 Shape，该类是一个抽象类，在该类中定义两个方法，即 getName()方法和 getArea()方法。getName()方法使用反射机制获得类名称，getArea()方法是一个抽象方法，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）编写类 Circle，该类继承自 Shape 并实现了其抽象方法 getArea()。在该类的构造方法中，获得了圆形的半径，以此在 getArea()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）编写类 Rectangle，该类继承自 Shape 并实现了其抽象方法 getArea()。在该类的构造方法中，获得了矩形的长和宽，以此在 getArea()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）编写类 Test 用来进行测试，在该类中创建了 Circle 对象和 Rectangle 对象，并分别输出图形的名称和面积，实例代码如下：

```
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());
    }
}
```

实例运行结果如图 8.20 所示。

图 8.20　实例运行结果

你问我答：如何使用抽象类？

在解决实际问题时，一般将父类定义为抽象类，需要使用这个父类进行继承与多态处理。根据继承和多态原理，继承树中越是在上方的类越抽象，例如，鸽子类继承鸟类，鸟类继承动物类等。在多态机制中，并不需要将父类初始化对象，需要的只是子类对象，所以在 Java 语言中，设置抽象类不可以实

例化对象，图形类不能抽象出任何一种具体图形，但它的子类却可以。

抽象类的语法格式如下：

```
public abstract class Test {
    abstract void testAbstract();                    // 定义抽象方法
}
```

参数说明：abstract 为定义抽象类关键字。

使用 abstract 关键字定义的类称为抽象类，而使用这个关键字定义的方法称为抽象方法，抽象方法没有方法体，这个方法本身没有任何意义，除非它被重写，而承载这个抽象方法的抽象类就必须被继承，实际上抽象类除了被继承之外没有任何意义。

反过来讲，如果声明一个抽象的方法，就必须将承载这个抽象方法的类定义为抽象，不可能在非抽象类中定义抽象方法。换句话说，只要类中有一个抽象方法，此类就必须标记为抽象类。

抽象类被继承后必须重写抽象类中所有的抽象方法，并定义方法体，否则编译无法通过，图 8.21 说明了继承抽象类的关系图。

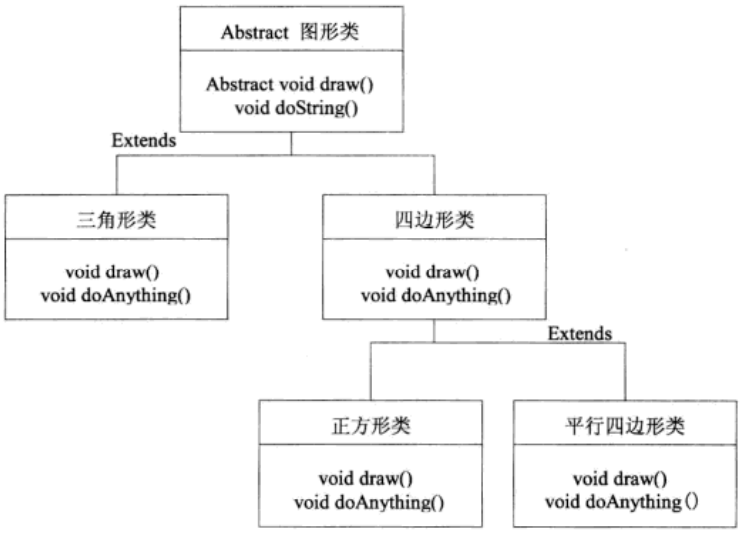


图 8.21　抽象类继承关系图

从图 8.21 中可以看到，继承抽象类的所有子类必须将抽象类中的抽象方法进行重写。这样在多态机制中可以将父类修改为抽象类，将 draw()方法设置为抽象方法，然后每个子类都重写这个方法。但这又会出现刚探讨多态时讨论的问题，程序中会有太多冗余的代码，同时这样的父类局限性很大，也许某个不需要 draw()方法的子类也不得不重写 draw()方法。

如果将 draw()方法放置在另外一个类中，让那些需要 draw()方法的类继承该类，而不需要 draw()方法的类继承图形类。可是所有的子类都需要图形类，因为这些类是从图形类中被导出的，同时某些类还需要 draw()方法，而 Java 语言规定，类不能同时继承多个父类，也就是 Java 只支持单继承。面临这种问题，接口的概念就出现了。

照猫画虎：修改“基本功训练 2——参数多态”中老鼠尝酒的实例，把老鼠类（即所有老鼠的父类 Mouse）修改为抽象类。（20 分）（实例位置：光盘\mr\ym\8.11\zmhh）

8.6.4　基本功训练 4——通过接口实现多态

视频讲解：光盘\mr\lx\08\通过接口实现多态.exe

实例位置：光盘\mr\ym\8.12\src

本实例将通过接口来完成多态数组的应用。实例中创建了 DrawTest 接口，并创建该接口的两个实现类 Parallelogram 平行四边形类与 Square 正方形类，然后在主方法中分别调用这两个类的 draw()方法。

实现过程如下：

（1）创建 DrawTest 接口，定义接口方法 draw()，所有实现该接口的类必须实现该方法，代码如下：

```
/**
 * 带有 draw 方法的接口
 * @author Li Zhong Wei
 */
public interface DrawTest {      // 定义接口
    public void draw();          // 定义接口方法
}
```

（2）定义 Parallelogram 平行四边形类，该类实现了 DrawTest 接口和其中的方法，代码如下：

```
/**
 * 平行四边形类，它实现了 DrawTest 接口
 * @author Li Zhong Wei
 */
public class Parallelogram implements DrawTest {
    public void draw() { // 由于该类实现了接口，所以需要覆盖 draw()方法
        System.out.println("平行四边形.draw()");
    }
}
```

（3）定义 Square 正方形类，它实现了 DrawTest 接口和其中的方法，代码如下：

```
/**
 * 正方形类，它实现 DrawTest 接口
 * @author Li Zhong Wei
 */
public class Square implements DrawTest {
    public void draw() {
        System.out.println("正方形.draw()");
    }
}
```

（4）编写 Demo 类，在该类的主方法中创建 DrawTest 接口类型的数组，它包含两个数组元素，分别是 Parallelogram 类与 Square 类的实例对象，然后在遍历该数组时执行每个数组元素的 draw()方法，代码如下：

```
/**
 * 实例的主类，创建接口类型的多态数组，并调用每个元素的 draw()方法
 * @author Li Zhong Wei
 */
public class Demo { // 定义四边形类
    public static void main(String[] args) {
        // 接口也可以进行向上转型操作
        DrawTest[] graphs = { new Square(), new Parallelogram() };
        for (int i = 0; i < graphs.length; i++) {
            graphs[i].draw(); // 调用 draw()方法
```

```
        }
    }
}
```

运行结果如图 8.22 所示。

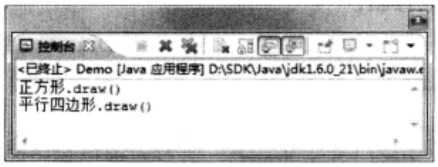


图 8.22　多态与接口结合

在本实例中，平行四边形类与四边形类实现了 DrawTest 接口，所以必须实现接口中的方法，在调用每个对象的 draw()方法时，首先将平行四边形类对象与正方形类对象向上转型为 DrawTest 接口形式，这里也许大多读者会有疑问，接口是否可以向上转型，其实在 Java 中无论是将一个类向上转型为父类对象，还是向上转型为该类实现接口都可以，但是转型后只能调用接口或父类中定义的方法。

你问我答：如何定义与实现接口？

接口是抽象类的延伸，可以将它看作是纯粹的抽象类，接口中的所有方法都没有方法体。接口使用 interface 关键字进行定义，它的语法格式如下：

```
public interface drawTest{
    void draw();                    // 接口内的方法，省略 abstract 关键字
}
```

参数说明如下。

interface：定义接口的关键字。

drawTest：接口名称。

一个类实现一个接口可以使用 implements 关键字，例如，下面的代码定义了 Parallelogram 类并实现了 DrawTest 接口：

```
public class Parallelogram implements drawTest{
    …//
}
```

注意：在接口中定义的方法必须被定义为 public 或 abstract 修饰符，其他修饰权限不被 Java 编译器所认可，如果省略方法声明时的修饰符，那么它的默认修饰符也是 public。

说明：在接口中定义的任何成员变量默认都是 static 和 final，也就是接口的常量。

在 Java 中不允许多重继承，但使用接口就可以解决这类问题，因为一个类可以同时实现多个接口，可以将所有需要继承的接口放置在 implements 关键字后使用逗号隔开，但这可能给一个类带来庞大的代码量，因为当继承一个接口时就需要实现接口中所有的方法，语法格式如下：

```
class 类名 extends 父类 implements 接口 1,接口 2,…,接口 n
```

另外，接口之间也可以继承。例如，下面的代码在定义一个接口时使该接口继承另外一个接口：

```
interface intf1{}
interface intf2 extends intf1{}
```

照猫画虎：编写一个 Apple 苹果类，让该类实现 java.io 包中的 Serializable 接口。（20 分）（实例位置：光盘\mr\ym\8.12\zmhh）

8.6.5　基本功训练 5——对象的转型与类型判断

视频讲解：光盘\mr\lx\08\对象的转型与类型判断.exe

实例位置：光盘\mr\ym\8.13\src

在项目中创建 Parallelogram 类和 Quadrangle、Square、Anything 3 个类，其中 Parallelogram 类和 Square 类继承 Quadrangle 类，在 Demo 类的主方法中分别创建这些类的对象，然后使用 instanceof 操作符判断它们的类型，并输出结果。

实现过程如下：

（1）编写 Quadrangle 四边形类，该类是一个空类，没有定义任何方法，代码如下：

```
/**
 * 这是一个四边形类，它在本实例中作为正方形与平行四边形类的父类
 * @author Li Zhong Wei
 */
public class Quadrangle {
}
```

（2）编写 Square 类，它继承 Quadrangle 类，同样不需要定义成员变量与成员方法，因为本实例的重点在于对象的转型与类型判断，所以不需要调用任何方法，代码如下：

```
/**
 * 这是一个正方形类，它继承 Quadrangle 四边形类
 * @author Li Zhong Wei
 */
public class Square extends Quadrangle {
}
```

（3）继承 Quadrangle 类编写 Parallelogram 类，代码如下：

```
/**
 * 这是一个平行四边形类，它继承 Quadrangle 四边形类
 * @author Li Zhong Wei
 */
public class Parallelogram extends Quadrangle {
}
```

（4）再编写一个空类 Anything，代码如下：

```
/**
 * 这是实例测试用的类，它可以是任何类，但是没有继承四边形和它的子类
 * @author Li Zhong Wei
 */
public class Anything {
}
```

（5）编写 Demo 类，这是实例的主类，在该类的主方法中分别创建实例中其他类的实例对象，然后调用该类定义的 verdict()方法判断每个对象的类型。verdict()方法在定义时的参数是 Object 类型，Object 类是所有类的父类，传递给该方法的所有对象将会向上转型为 Object 类，然后在方法体中输出该对象是由哪个类创建的，具体代码如下：

```
/**
 * 这是本实例的主类，它将演示 instanceof 的用法
 * @author Li Zhong Wei
 */
public class Demo {
    public static void main(String args[]) {
        Quadrangle quaObj = new Quadrangle();                    // 创建四边形对象
        Square squareObj = new Square();                         // 创建正方形对象
        Parallelogram parObj = new Parallelogram();              // 创建平行四边形对象
        Anything any=new Anything();                             // 创建其他类
        System.out.println("判断 Quadrangle 类的对象 quaObj：");
        verdict(quaObj);
        System.out.println("判断 Square 类的对象 squareObj：");
        verdict(squareObj);
        System.out.println("判断 Parallelogram 类的对象 parObj：");
        verdict(parObj);
        System.out.println("判断 Anything 类的对象 any：");
        verdict(any);
    }
    /**
     * 判断对象类型的方法
     * @param quaObj 判断的对象
     */
    private static void verdict(Object quaObj) {
        if (quaObj instanceof Quadrangle)
            System.out.println("\t 该对象是 Quadrangle 类的实例对象。");
        if(quaObj instanceof Square)
            System.out.println("\t 该对象是 Square 类的实例对象");
        if(quaObj instanceof Parallelogram){
            System.out.println("\t 该对象是 Parallelogram 类的实例对象");
        }
        if(quaObj instanceof Anything){
            System.out.println("\t 该对象是 Anything 类的实例对象");
        }
        System.out.println("=======================================");
    }
}
```

实例运行结果如图 8.23 所示。

图 8.23　实例运行结果

你问我答：什么是对象转型以及如何判断对象的类型？

（1）判断对象的类型

在执行类型转型时，如果把对象转换成错误的类型，就会发生 ClassCastException 异常，所以在执行类型转型之前需要先判断对象的类型，防止程序出现错误。Java 使用 instanceof 操作符来完成对象类型的判断。使用 instanceof 操作符可以判断对象是不是指定的类或子类的实例对象，或者接口的实现类对象，instanceof 关键字的用法如下：

```
obj instanceof Class
```

参数说明：obj 是一个实例对象，Class 指某个类或接口。

使用 instanceof 操作符时表达式的返回值为布尔值，如果返回值为 true，说明 obj 对象是 Class 类或子类的实例对象，否则返回 false 值。

（2）对象类型转换

从实例运行结果可以看出，一个对象既是创建该对象的类的实例，也是父类的实例。例如，实例中的平行四边形和正方形的输出结果，除了是本身类的对象之外，还是父类四边形的对象。这就是说，可以将它们强制类型转换成父类的类型，例如：

```
(squareObj)squareObj;                          // 强制类型转换为父类
```

或者把子类对象赋值给父类的对象变量，例如：

```
Quadrangle obj = new Square();                 // 声明父类对象，初始化为子类
```

这种子类对象赋值给父类引用的形式称为向上转型，它将导致声明的对象只能调用父类的方法。因为父类中没有定义子类的新方法，它只包含定义父类时的方法。

任何对象使用 instanceof 关键字和 Object 类进行判断，返回结果一定是 true，因为 Java 语言中所有的类都直接或间接地继承了 Object 类。也就是说任何对象都可以赋值给 Object 类型的变量。可以使用 Object 类型的变量来传递未知类型的对象，就像本实例中的 verdict()方法。

但是 Object 类型的对象没有定义的业务方法，要使用该对象的类的方法，可以在判断对象的类型之后，使用强制类型转换把对象转换为原来的类型，这样就可以调用需要的方法了。这种父类对象转换为子类对象的形式称为向下转型，例如：

```
if(quaObj instanceof Square) {
    Square newObj = (Square) quaObj;
    newObj.……                                 // 调用 Square 类的方法
}
```

注意：向下转型一定要先确定对象的类型是否和要转换的类型有继承关系，否则会出现错误。

照猫画虎：修改实例中主类的 verdict()方法，为该方法添加一个 Object 类的对象判断，看所有对象是否都是 Object 类的子类。（20 分）（实例位置：光盘\mr\ym\8.13\zmhh）

照猫画虎栏目分数统计：

照猫画虎题目	1	2	3	4	5	总分数	
分数							

8.7 情景应用——拓展与实践

8.7.1 情景应用 1——单例模式的应用

视频讲解：光盘\mr\lx\08\单例模式的应用.exe

实例位置：光盘\mr\ym\8.14\src

在中国历史上，有个很特殊的职业，通常其从业者有且仅有一人，那就是皇帝。作为大臣，是需要经常上朝参拜皇帝的。当叩首完毕后，发现还是上次那个人，心中暗喜：自己的饭碗还没丢！本实例使用单例模式来保证皇帝的唯一性，实例的运行效果如图 8.24 所示。

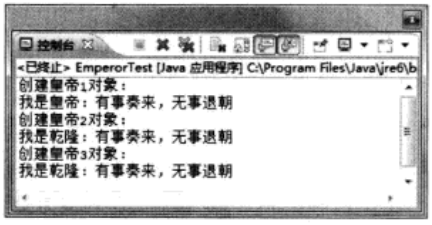


图 8.24 实例运行效果

实现过程如下：

（1）编写类 Emperor，该类的构造方法是私有的。它提供了一个 getInstance()方法来获得该类的实例，再定义一个 sayInfo()方法输出皇帝的名字，代码如下：

```
public class Emperor {
    private static Emperor emperor = null;
    private String name = "我是皇帝";
    private Emperor() {                                    // 把构造方法私有化
    }
    public static Emperor getInstance() {                  // 定义获取实例的方法
        if (emperor == null) {                             // 确保只创建一个实例对象
            emperor = new Emperor();
        }
        return emperor;
    }
    public void setName(String name) {                     // 修改 name 成员变量的方法
        this.name = name;
    }
    public void sayInfo() {                                // 输出信息的方法
        System.out.println(name + "：有事奏来，无事退朝");
    }
}
```

注意：对于多线程程序，有时并不能保证实例的唯一性，这就需要使用同步。

（2）编写类 EmperorTest，该类用来测试 Emperor 类。在该类的 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 EmperorTest {
    public static void main(String[] args) {
        System.out.println("创建皇帝 1 对象：");
        Emperor emperor1 = Emperor.getInstance();     // 获取实例对象
        emperor1.sayInfo();                           // 输出信息
        System.out.println("创建皇帝 2 对象：");
        Emperor emperor2 = Emperor.getInstance();     // 获取实例对象
        emperor2.setName("我是乾隆");                  // 修改 name 成员变量
        emperor2.sayInfo();                           // 输出信息
        System.out.println("创建皇帝 3 对象：");
        Emperor emperor3 = Emperor.getInstance();     // 获取实例对象
        emperor3.sayInfo();                           // 输出信息
    }
}
```

DIY：把象棋棋子“马”定义成一个 Horse，限制其实例对象数量为两个。（20 分）（实例位置：光盘\mr\ym\8.14\diy）

8.7.2　情景应用 2——简单的汽车销售商场

视频讲解：光盘\mr\lx\08\简单的汽车销售商场.exe

实例位置：光盘\mr\ym\8.15\src

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。本实例将实现一个简单的汽车销售商场，用来演示多态的用法，实例的运行效果如图 8.25 所示。

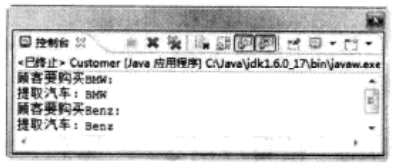


图 8.25　实例运行效果

实现过程如下：

（1）编写类 Car，该类是一个抽象类，在其中定义了一个抽象方法 getInfo()，代码如下：

```
public abstract class Car {
    public abstract String getInfo(); // 用来描述汽车的信息
}
```

（2）编写类 BMW，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）编写类 Benz，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）编写类 CarFactory，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）编写类 Customer，它是本实例的主类，在类的主方法中，根据用户的需要提取不同的汽车，实例代码如下：

```
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
    }
}
```

DIY：修改实例，为 CarFactory 类的 getCar()方法添加一个限制，每种汽车只销售一辆。（20 分）（实例位置：光盘\mr\ym\8.15\diy）

8.7.3 情景应用 3——重新定义对象的等式判断

视频讲解：光盘\mr\lx\08\重新定义对象的等式判断.exe

实例位置：光盘\mr\ym\8.16\src

由于生命的复杂性，寻找两只完全相同的宠物是不可能的。在 Java 语言中，解决这样的问题却简单得多。可以通过比较感兴趣的属性来判断两个对象是否相同。本实例将创建 3 个宠物猫对象，通过比较它们的名字、年龄、重量和颜色属性来看它们是否相同，实例的运行效果如图 8.26 所示。

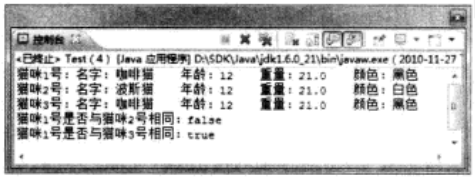


图 8.26 实例运行效果

实现过程如下：

（1）编写类 Cat，在该类中定义了 4 个域，即 name 表示名字、age 表示年龄、weight 表示重量、color 表示颜色。重写 equals()方法来比较对象的属性是否相同。代码如下：

```
public class Cat {
    private String name;
```

```
    private int age;
    private double weight;
    private String color;
    public Cat(String name, int age, double weight, String color) {
        this.name = name;
        this.age = age;
        this.weight = weight;
        this.color = color;
    }
    @Override
    public boolean equals(Object obj) {
        if (this == obj) {
            return true;
        }
        if (obj == null) {
            return false;
        }
        if (getClass() != obj.getClass()) {
            return false;
        }
        Cat cat = (Cat) obj;
        return name.equals(cat.name) && (age == cat.age)
                && (weight == cat.weight) && (color.equals(cat.color));
    }
    public String getInfo() {
        StringBuilder sb = new StringBuilder();
        sb.append("名字：" + name + "      ");
        sb.append("年龄：" + age + "      ");
        sb.append("重量：" + weight + "      ");
        sb.append("颜色：" + color + "      ");
        return sb.toString();
    }
}
```

（2）编写 Demo 类进行测试，在该类的 main()方法中创建 3 个猫咪对象，并分别为其初始化，然后输出猫咪对象和比较的结果，代码如下：

```
public class Demo {
    public static void main(String[] args) {
        Cat cat1 = new Cat("咖啡猫", 12, 21, "黑色");          // 创建第 1 个猫咪对象
        Cat cat2 = new Cat("波斯猫", 12, 21, "白色");          // 创建第 2 个猫咪对象
        Cat cat3 = new Cat("咖啡猫", 12, 21, "黑色");          // 创建第 3 个猫咪对象
        System.out.println("猫咪 1 号：" + cat1.getInfo());    // 输出每个猫咪对象的属性
        System.out.println("猫咪 2 号：" + cat2.getInfo());
        System.out.println("猫咪 3 号：" + cat3.getInfo());
        System.out.println("猫咪 1 号是否与猫咪 2 号相同：" + cat1.equals(cat2));
        System.out.println("猫咪 1 号是否与猫咪 3 号相同：" + cat1.equals(cat3));
    }
}
```

DIY：修改实例，为猫咪类添加一个体重属性，然后把该属性添加到 equals()方法的条件判断中。（20 分）（实例位置：光盘\mr\ym\8.16\diy）

8.7.4 情景应用 4——重新定义对象的字符串表现方式

视频讲解：光盘\mr\lx\08\重新定义对象的字符串表现方式.exe

实例位置：光盘\mr\ym\8.17\src

在实际编程中，可能需要使用字符串来表示对象，如在控制台输出对象。此时，虚拟机会自动调用toString()方法，然而该方法的默认返回值并没有多大用处，通常需要重写该方法。本实例将演示如何重写该方法，实例的运行效果如图 8.27 所示。

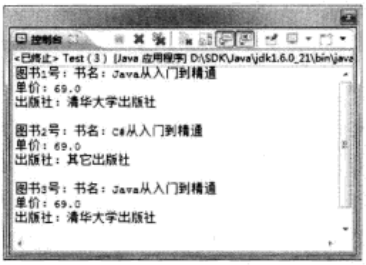


图 8.27 实例运行效果

实现过程如下：

（1）编写 Book 类，在该类中定义了 3 个成员变量，分别是 name 表示书名、price 表示单价、publisher 表示出版社。重写 toString()方法来输出对象的信息，实例代码如下：

```
public class Book {
    private String name;        // 书名
    private double price;       // 单价
    private String publisher;   // 出版社

    public Book(String name, double price, String publisher) {
        this.name = name;
        this.price = price;
        this.publisher = publisher;
    }

    @Override
    public String toString() {
        StringBuilder sb = new StringBuilder();
        sb.append("书名：" + name + "\n");
        sb.append("单价：" + price + "\n");
        sb.append("出版社：" + publisher + "\n");
        return sb.toString();
    }
}
```

（2）编写 Test 类进行测试，在该类的 main()方法中创建 3 个图书对象，并分别为其初始化。然后输出图书对象，代码如下：

```
public class Test {
    public static void main(String[] args) {
        Book cat1 = new Book("Java 从入门到精通", 69.00, "清华大学出版社");
```

```
        Book cat2 = new Book("C#从入门到精通", 69.00, "其他出版社");
        Book cat3 = new Book("Java 从入门到精通", 69.00, "清华大学出版社");
        System.out.println("图书 1 号：" + cat1);
        System.out.println("图书 2 号：" + cat2);
        System.out.println("图书 3 号：" + cat3);
    }
}
```

DIY：修改 Book 类的 toString()方法，把图书信息定义在一行文本中。（20 分）（实例位置：光盘\mr\ym\8.17\diy）

8.7.5　情景应用 5——重新计算对象的哈希码

视频讲解：光盘\mr\lx\08\重新计算对象的哈希码.exe

实例位置：光盘\mr\ym\8.18\src

Java 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，直接查找键即可，而不用遍历整个堆来查找对象。本实例将重新定义不同对象的散列值（即哈希码），实例的运行效果如图 8.28 所示。

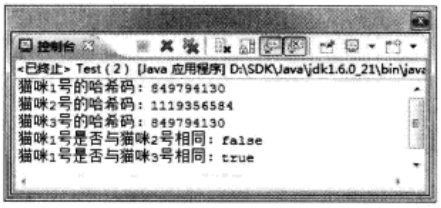


图 8.28　实例运行效果

实现过程如下：

（1）编写类 Cat，在该类中定义了 4 个域，分别是 name 表示名字、age 表示年龄、weight 表示重量、color 表示颜色。重写 equals()方法来比较对象的属性是否相同，重写 hashCode()方法来计算对象的哈希码，实例代码如下：

```
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）编写 Test 类进行测试，在该类的 main()方法中创建 3 只猫咪，并分别为其初始化。然后输出猫咪的哈希码和比较的结果，实例代码如下：

```
public class Test {
    public static void main(String[] args) {
        Cat cat1 = new Cat("Java", 12, 21, Color.BLACK);      // 创建第 1 个猫咪对象
        Cat cat2 = new Cat("C++", 12, 21, Color.WHITE);       // 创建第 2 个猫咪对象
        Cat cat3 = new Cat("Java", 12, 21, Color.BLACK);      // 创建第 3 个猫咪对象
        System.out.println("猫咪 1 号的哈希码：" + cat1.hashCode());
        System.out.println("猫咪 2 号的哈希码：" + cat2.hashCode());
        System.out.println("猫咪 3 号的哈希码：" + cat3.hashCode());
        System.out.println("猫咪 1 号是否与猫咪 2 号哈希码相同：" + (cat1.hashCode()==cat2.hashCode()));
        System.out.println("猫咪 1 号是否与猫咪 3 号哈希码相同：" + (cat1.hashCode()==cat3.hashCode()));
    }
}
```

你问我答：hashCode()方法的作用是什么？

Java 中任何一个类都是 Object 类的直接或间接子类。如果类没有超类，则它默认继承自 Object 类。在 Object 类中，实现了很多有用的方法。hashCode()方法的作用是计算对象的哈希码，支持此方法是为了提高哈希表（如 java.util.Hashtable 提供的哈希表）的性能，该方法的声明如下：

```
public int hashCode()
```

Object 类在实现该方法时，返回值是对象的存储地址，这与 equals()方法的实现相关。因此，当重写 equals()方法后，一定要重写 hashCode()方法。

DIY：为本实例的 Cat 猫咪类重写 equals()方法，并修改主类的 main()方法中判断猫咪相同条件的代码，使用 equals()方法进行对比。（20 分）（实例位置：光盘\mr\ym\8.18\diy）

情景应用 DIY 栏目分数统计：

DIY 题目	1	2	3	4	5	总分数	
分数							

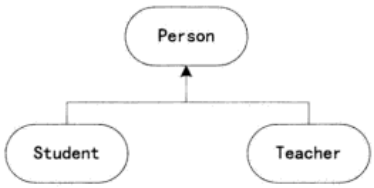
8.8 自 我 测 试

一、选择题（每题 10 分，5 道题）

1．下列哪些定义抽象类的语句是合法的？（　　）

A.
```
public class Test {
    abstract void enjoy();
}
```
B.
```
public abstract Test {
    abstract void enjoy();
}
```
C.
```
public abstract class Test {
    abstract void enjoy();
}
```
D.
```
public class abstract Test {
    abstract void enjoy();
}
```

2．对于 Person、Student 和 Teacher 类，存在如图所示的继承关系：



如果有个方法需要同时支持这 3 种类型的参数，则可以将参数的类型设置成？（　　）

A．Person　　B．Student　　C．Teacher　　D．Object

3．当编译和运行下列代码时会发生什么？（　　）

```
public class Base {
    private void say(int number) {
        System.out.println("Base：" + number);
    }
}
public class Over extends Base {
    public void say(int number) {
        System.out.println("Over：" + number);
    }

    public static void main(String[] args) {
        Over over = new Over();
        over.say(12);
    }
}
```

A．编译错误，因为 Base 类中的 say 方法是 private 的

B．运行错误，因为 Base 类中的 say 方法是 private 的

C．输出 Base：12

D．输出 Over：12

4．下列关于抽象方法的定义哪些（个）是正确的？（　　）

A．public abstract add();　　B．public abstract add(){};

C．public abstract void add();　　D．public abstract void add(){}

5．定义接口的关键字是？（　　）

A．int　　B．interface　　C．abstract　　D．import

二、填空题（每题 10 分，5 道题）

1．哪个关键字如果修饰了方法就必须修饰该方法所在的类？_____

2．toString()方法是在哪个类中定义的？_____

3．当重写 equals()方法之后通常还需要重写_____方法。

4．编写一行代码，使用 String 类的 substring()方法从字符串 phenylalanine 中获得字符串 alanine。

_____

5．编码判断对象属于哪个类，应该使用哪个关键字？_____

测试分数统计：

类别	第 1 题	第 2 题	第 3 题	第 4 题	第 5 题
选择题分数					
填空题分数					
总分数					

8.9 行 动 指 南

开始日期：_____年____月____日　　结束日期：_____年____月____日

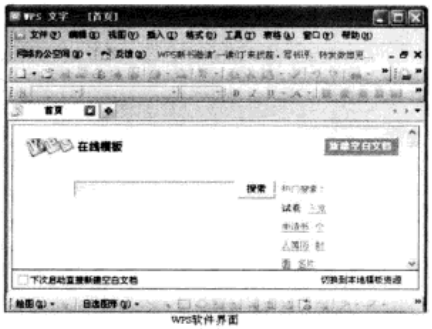
序号	内　容	行 动 指 南	
1	照猫画虎栏目 分数（　　）	分数>75 分	优秀，基本功掌握得不错，加油！
		75 分>分数>50 分	及格，知识掌握得不牢，重新做一遍照猫画虎。
		分数<50 分	请使用光盘中提供的“实战能力测试系统”进行提高训练。
	情景应用栏目 分数（　　）	分数>75 分	优秀，综合应用能力很强。
		75 分>分数>50 分	及格，综合应用能力需提高，再练一遍情景应用。
		分数<50 分	请使用光盘中提供的“实战能力测试系统”进行提高训练。
	自我测试栏目 分数（　　）	分数>75 分	优秀，有成为编程高手的潜质。
		分数<75 分	请使用光盘中提供的“实战能力测试系统”进行提高训练。
	综合评价	反复训练后，以上各项分数都在 75 分以上，方可进入下一堂课学习。	
2	编程实践能力培养：本栏目提供了一些实践题目，对于培养编程能力很有效，要做好。	（1）自定义学生类，并重写 equals()方法，然后判断两个学生对象是否相同。	
		（2）继承抽象的动物类，并完成两个子类的创建。	
		（3）定义接口，并完成两个实现该接口的类。	
		（4）自定义打印方法，方法的参数要通过接口实现多态。提示：参数使用自定义的图书接口，然后传递实现该接口的子类的对象。	
3	编程习惯培养：本堂课培养读者在学习开发中记笔记的习惯，把开发中遇到的问题、总结的经验记录下来。		
4	创新能力培养：看看身边有没有可以用编程解决的问题，记录到右边的表格中。根据学习进度，尝试编写解决这些问题的小程序。		

8.10　成功可以复制——中国第一程序员求伯君

求伯君，1964 年 11 月 26 日出生于浙江新昌县。1980 年，求伯君考入国防科技大学系统工程与数学系学习，因系统工程需要大量使用电脑，求伯君也比同龄人有了更早接触电脑的机会，这也成为求伯君的第一次人生机遇。1983 年，求伯君利用业余时间为国防科技大学图书馆开发了图书馆借还书管理系统，这是求伯君写的第一个能用的程序，为此求伯君获得了 40 多元钱的奖励，从此，求伯君与程序结下了不解之缘。

1986 年，求伯君改进编写了一个支持多种打印机的西山文字打印驱动程序，以 2000 元的价格卖给了四通，求伯君也成为四通的一员。西山文字打印驱动程序在四通定价为 500 元一套，卖了好几百套。

1988 年，求伯君离开四通公司，加入香港金山公司，负责开发用于 DOS 系统的中文文字处理软件——WPS。为了加快开发速度，求伯君把自己关在深圳的一个房间里，只要是醒着，就不停地写。什么时候困了，就睡一会儿，饿了就吃方便面。在这样的一年零四个月中，求伯君生了三次病，第一次肝炎，第二次肝炎复发，第三次又复发，每次住院一个月到两个月。第二次肝炎复发正是软件开发最紧要的关头，求伯君把电脑搬到病房里继续写。求伯君在病房中，写下了有十几万行代码的 WPS 软件。WPS 一进入市场就凭着良好的口碑迅速占领市场，当年卖了 3 万多套，销售额达 6000 多万元。


WPS软件界面

1994 年，求伯君在珠海成立珠海金山电脑公司，继续从事办公软件开发。开发的第一个软件产品——盘古组件（里面有 WPS、电子表和字典），遇到了 Word 的挑战，最终以失败告终。1996 年金山陷入苦难时期，运营资金紧张、员工信心不足。在危难时刻，求伯君毅然卖掉了别墅，凭着对 WPS 的深厚感情克服了重重困难与障碍，1997 年推出 WPS97。WPS97 推出仅两个多月，就销出了 13000 套，金山也借此迈过最艰难的时刻。

如今的金山软件，已占据通用软件全国第一，2009 年总营业收入达 10.22 亿元。

经典语录

如果从开始就想着怎样赚钱，我也不会有今天。事业和金钱无关。当你全身心投入开发的时候，不给你钱你也要干。开发时，根本没有心思考虑报酬。只有先成就了业，才有资格谈报酬。

深度评价

求伯君的成功告诉我们，热爱编程的程序员们要耐得住寂寞，坚定信念，用自己的热情全身心地投入才能创造成功的事业。

第 9 堂课

其他类特性与异常处理

（视频讲解：70 分钟）

在面向对象的程序设计中，类和对象是重点内容。其中类的特性有很多，除了具有普通特性之外，它还具有 final 特性、内部类特性等。本堂课将介绍这些特性，另外还会介绍 Java 异常处理技术。异常是程序执行过程中出现的各种错误，Java 语言将各种错误定义为异常，并提供捕获、处理异常的方法，避免程序因错误而导致瘫痪与崩溃。

学习摘要：

- final 方法与 final 类的定义
- 各种内部类的定义
- 异常对象的分类
- 捕获和抛出异常
- 异常处理的使用原则

9.1 final 特性

final 关键字可以用来修饰类、成员变量和方法等，它通常表示不可变化。例如，使用 final 修饰的成员变量，一旦赋值之后就不能再更改；使用 final 修饰的类，是不能够继承的。本节将讲述该关键字的用法。

9.1.1 final 变量

final 关键字可用于变量声明，一旦声明的 final 变量被赋值，就不可以再修改该变量的值。通常由 final 定义的变量为常量，例如，在类中定义的 PI 值，可以使用如下语句：

```
final double PI=3.1415926;
```

当在程序中用到 PI 这个常量时，它的值就是 3.1415926。如果在程序中再次对定义为 final 的常量赋值，编译器报错。

final 除了可以修饰基本类型的常量之外，还可以修饰引用类型。由于数组变量也可以被看作一个对象引用，所以 final 可修饰数组。一旦一个对象引用被修饰为 final，它只能恒定指向一个对象，无法将其改变指向另一个对象。一个既是 static 又是 final 的成员变量只占据一段不能改变的存储空间。

可以将方法的参数定义为 final 类型，这预示着无法在方法体中更改参数值或更改参数所指向的对象引用。

在程序中 final 数据可以出现的位置如图 9.1 所示，图中清晰地表明了在程序中哪些位置可以定义 final 数据。

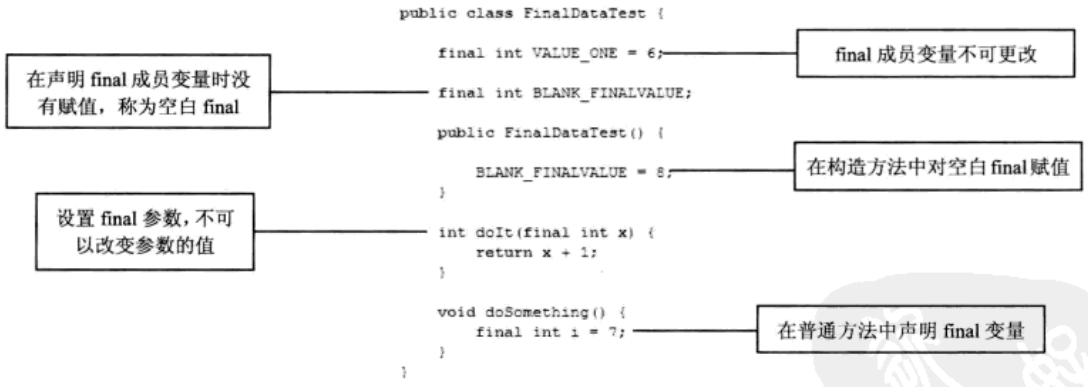


图 9.1 程序中可以定义为 final 的数据

9.1.2 final 方法

首先读者应该了解定义为 final 的方法不能被重写，将方法定义为 final 类型可以防止任何子类修改该方法的定义与实现方式，同时定义为 final 的方法执行效率要高于非 final 方法。在修饰权限中曾经提到过 private 修饰符，如果一个父类中某个方法被设置为 private 修饰符，子类将无法访问该方法，自然无法覆盖该方法，所以一个定义为 private 的方法隐式被指定为 final 类型，这样无须将一个定义为 private 的方法再定义为 final 类型。例如：

```
private final void test(){
    …//someSentence
}
```

9.1.3　final 类

定义为 final 的类不能被继承。如果希望一个类不允许任何类继承，并且不允许别人对这个类有任何改动，可以将这个类设为 final 形式。final 类的语法格式如下：

```
public final 类名{
    // 类体
}
```

将某个类设为 final 形式，类中的所有方法都被隐式设置为 final 形式，但是 final 类中的成员变量可以被定义为 final 或非 final 形式。

例 9.01　在项目中创建 FinalClass 类，在类中定义 doit()方法和变量 numb，在 main()方法中操作变量 numb 自增。实例的代码如下：（**实例位置：光盘\mr\ym\9.01**）

```
public final class FinalClass {
    int numb = 3;                                          // 定义成员变量
    public void doit() {                                   // 定义成员方法
    }
    public static void main(String args[]) {
        FinalClass finalObj = new FinalClass();            // 创建对象
        finalObj.numb++;                                   // 改变 numb 成员变量的值
        System.out.println("numb = " + finalObj.numb);
    }
}
```

实例运行结果可以证明，final 类的成员变量是可以不声明为 final 的，可以通过对象改变该变量的值，但是如果试图继承 final 类型的类或重写该类中的方法是不被允许的。如图 9.2 所示，在修正提示中也只能去掉 FinalClass 类的 final 修饰符才能继承。

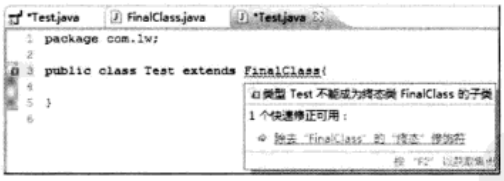


图 9.2　继承 Final 类

9.2　内　部　类

在类中定义的类称为内部类，内部类可分为成员内部类、局部内部类以及匿名类等。本节将向读者讲解内部类的使用。

9.2.1　成员内部类

在一个类的内部直接定义的类，称为成员内部类，其语法格式如下：

```
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，哪怕这些类成员被修饰为 private，图 9.3 充分说明了内部类的使用。尽管成员变量 numb 以及成员方法 testMethod()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私有成员。

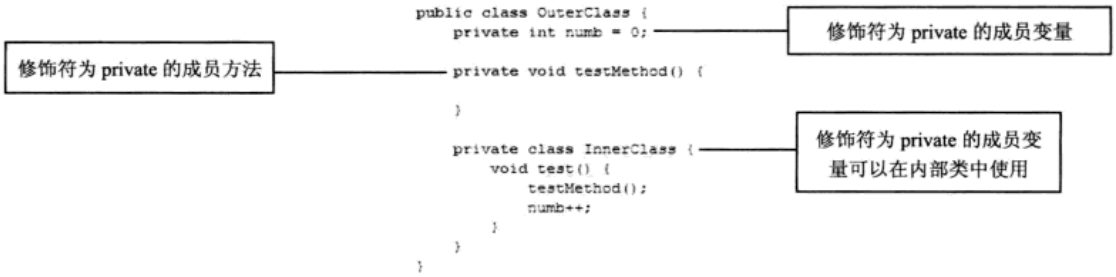


图 9.3　内部类可以使用外部类的成员

内部类的实例一定要绑定在外部类的实例上。如果从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，下面来看一个实例。

例 9.02　在项目中创建 OuterClass 类，在类中定义 innerClass 内部类和 doit()方法。在 main()方法中创建 OuterClass 类的实例对象并调用 doit()方法。实例的代码如下：（**实例位置：光盘\mr\ym\9.02**）

```
public class OuterClass {
    innerClass in = new innerClass();                    // 在外部类中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的非静态方法中实现
        OuterClass.innerClass in = out.doit();
    }
}
```

实例的外部类创建内部类实例时，与其他类创建对象相同。内部类可以访问它外部类的成员，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，实例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 9.4 说明了内部类 innerClass 对象与外部类 OuterClass 对象的关系。

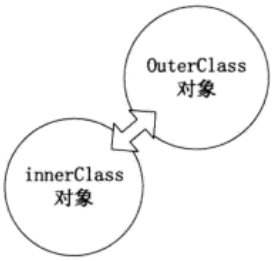


图 9.4　内部类对象与外部类对象的关系

从图 9.4 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，则需要使用外部类，内部类的形式指定该对象的类型。

在实例的 main()方法中，如果不使用 doit()方法返回内部类对象引用，可以直接使用实例化内部类对象，但是在 main()方法中或者其他类中实例化本实例的内部类对象时，必须在 new 操作符之前提供一个外部类的引用，例如，在 main()方法中实例化一个内部类对象，代码如下：

```
public static void main(String args[]) {
    OuterClass out = new OuterClass();
    OuterClass.innerClass in = out.doit();
    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，否则不能创建内部类对象。

另外，非内部类不能被声明为 private 或 protected 访问类型。如图 9.5 所示，把 OuterClass 外部类声明为 private 修饰符时，会出现错误提示。

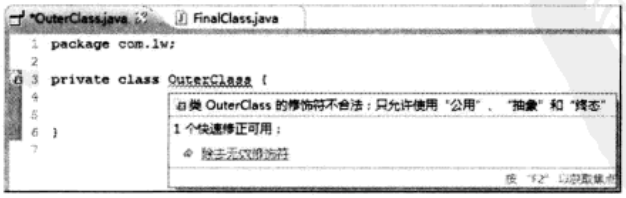


图 9.5　外部类不能声明为 private 与 protected

9.2.2　获取外部类的引用

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 9.03 在项目中创建 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。实例的代码如下：（**实例位置：光盘\mr\ym\9.03**）

```
public class TheSameName {
    private int x;                                          // 外部类的成员变量 x
    class Inner {
        private int x;                                      // 内部类的成员变量 x
        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 关键字即可获取外部类的一个引用。

9.2.3 局部内部类

在类的局部位置定义的类称为局部内部类，如在方法中定义的类。

例 9.04 在 doit()方法体中定义 InnerClass 类。（**实例位置：光盘\mr\ym\9.04**）

（1）定义 OutInterface 接口，代码如下：

```
public interface OutInterface {
}
```

（2）定义 OuterClass 类，代码如下：

```
public class OuterClass {
    public OutInterface doit(final String x) {              // doit()方法参数为 final 类型
        class innerClass implements OutInterface {          // 在 doit()方法中定义一个内部类
            innerClass(String s) {                          // 定义内部类的构造方法
                s = x;                                      // 接收参数
                System.out.println(s);                      // 输出参数
            }
        }
        return new InnerClass("doit");                      // 返回内部类对象
    }
}
```

在上述代码中可以看到，笔者将内部类定义在 doit()方法内部，但是有一点值得注意，内部类 innerClass 是 doit()方法的一部分，并非 OutClass 类中一部分，所以在 doit()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。

还有一个细节，就是将 doit()方法的参数设置为 final 类型，如果需要在方法体中使用局部变量，该局部变量需要被设置为 final 类型，换句话说，在方法中定义的内部类只能访问方法中 final 类型的局部变量，这是因为在方法中定义的局部变量相当于一个常量，它的生命周期超出方法运行的生命周期，由于该局部变量被设置为 final，所以不能在内部类中改变该局部变量的值。

9.2.4　匿名内部类

在创建内部类时，如果并没有指定内部类的名称，则称其为匿名内部类，比较常见的应用是 Swing 中的事件监听器。这里可以对 OuterClass 进行修改，例如，在 return 语句中编写返回值为一个匿名内部类，代码如下：

```
public class OuterClass {
    public OutInterface doit() {                                    // 定义 doit()方法
        return new OutInterface() {                                 // 声明匿名内部类
            private int i = 0;
            public int getValue() {
                return i;
            }
        };
    }
}
```

笔者将 doit()方法修改得有一些莫名其妙，但这种写法确实被 Java 编译器所认可，在 doit()方法内部首先返回一个 OutInterface 的引用，然后在 return 语句中插入一个定义内部类的代码。由于这个类没有名称，所以在这里该内部类被称为匿名内部类。实质上这种内部类的宗旨就是创建一个实现 OutInterface 接口的匿名类的对象。

匿名类的所有实现代码都需要在大括号之间进行编写，匿名类的语法格式如下：

```
new ClassName(){
…//内部类体
};
```

参数说明如下。

ClassName：类的名称。

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 OutInterface 实现类的对象。在匿名内部类定义结束后，需要加分号标识。这个分号并不是代表定义内部类结束的标识，而是代表创建 OutInterface 实现类对象表达式的标识。

📖 **说明：匿名内部类编译以后，会产生以“外部类名$序号”为名称的 class 文件，序号是 1～n 开始排列，分别代表 1～n 个匿名内部类，而非匿名的内部类会在“$”字符后面连接内部类的名称。**

9.2.5　静态内部类

使用 static 关键字修饰的内部类称为静态内部类。静态内部类的一个最大特点就是不可以使用外部类的非静态成员。

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，它会具有更多的限制。

静态内部类具有以下两个特点：

☑　如果创建静态内部类的对象，不需要外部类对象的引用。

☑　不能从静态内部类的对象中访问非静态外部类的对象。

例如，定义一个静态内部类 Inner，可以使用如下代码：

```
public class StaticInnerClass {
    int numbX = 100;
    static class Inner {
        void doitInner() {
            System.out.println("外部类" + numbX);
        }
    }
}
```

上述代码在内部类的 doitInner()方法中调用成员变量 numbX，由于 Inner 被修饰为 static 形式，而成员变量 numbX 却是非 static 类型的，所以在 doitInner()方法中不能调用 numbX 变量。

9.3 异 常 处 理

9.3.1 什么是异常

在程序中，错误可能产生于程序员没有预料到的各种情况，也可能是因为超出了程序员控制之外的环境因素，如用户的坏数据、试图打开一个根本不存在的文件等。在 Java 中，这种在程序运行时可能出现的一些错误称为异常。异常是一个在程序执行期间发生的事件，它中断了正在执行的程序的正常指令过程。

例 9.05 在项目中创建类 Baulk，在 main()方法中定义 int 型变量，将 0 作为除数赋值给该变量。实例代码如下：（**实例位置：光盘\mr\ym\9.05**）

```
public class Baulk {                                    // 创建类 Baulk
    public static void main(String[] args) {            // main()方法
        int result = 3 / 0;                              // 定义 int 型变量并赋值
        System.out.println(result);                      // 将变量输出
    }
}
```

实例运行效果如图 9.6 所示。

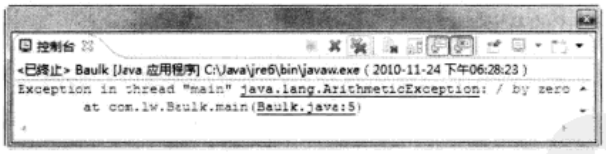


图 9.6 实例运行效果

上面程序运行的结果发生了算术异常 ArithmeticException。（根据给出的错误提示，可知发生错误是因为在算术表达式“3/0”中 0 作为除数出现）系统不再执行下去，提前结束。这种情况就是所说的异常。

Java 语言是一门面向对象的编程语言，因此，异常在 Java 语言中也是对象。当在某一方法中发生异常错误时，这个方法创建一个对象，并且把它传递给运行时的系统，这个对象就是异常对象。

9.3.2 捕获异常

Java 语言可以捕获程序中可能出现的异常，这主要由 try…catch 语句完成。Java 异常处理的结构为：

```
try {
    // 程序代码块
```

```
} catch (Exceptiontype1 e) {
    // 对 Exceptiontype1 异常的处理
} catch (Exceptiontrpe2 e) {
    // 对 Exceptiontype2 异常的处理
} finally {
    // 程序块
}
```

其中 try 语句块存放的是可能发生异常的 Java 语句，catch 程序块在 try 语句块之后，用来处理被捕获的异常。另外，finally 语句块是异常处理结构的最后执行部分。不管 try 块中代码如何退出，都将执行 finally 块。finally 是可选部分。

例 9.06　在项目中创建类 Take，在 main()方法中使用 try…catch 语句将可能出现的异常语句进行异常处理。实例的代码如下：（**实例位置：光盘\mr\ym\9.06**）

```
public class Take {                                          // 创建类
    public static void main(String[] args) {
        try {                                                // try 语句中包含可能出现异常的程序代码
            String str = "lili";                             // 定义字符串变量
            System.out.println(str + "年龄是：");             // 输出的信息
            int age = Integer.parseInt("20L");               // 数据类型转换
            System.out.println(age);
        } catch (Exception e) {                              // catch 语句块用来获取异常信息
            e.printStackTrace();                             // 输出异常性质
        }
        System.out.println("程序结束");                       // 输出信息
    }
}
```

实例运行效果如图 9.7 所示。

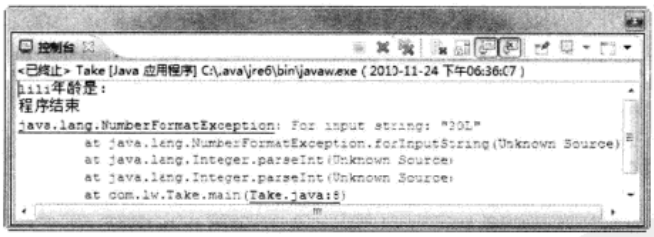


图 9.7　实例运行效果

从运行结果可以看出，程序仍然输出最后的“程序结束”信息，没有因为异常而终止。在实例 9.06 中将可能出现异常的代码用 try…catch 语句进行处理，当 try 代码块中的语句发生异常时，程序就会调转到 catch 代码块中执行，执行完 catch 代码块中的程序代码后，继续执行 catch 代码块后的其他代码，不会执行 try 代码块中发生异常语句后面的代码。由此可以得出，Java 的异常处理是结构化的，不会因为一个异常影响整个程序的执行。

catch 关键字后面括号中的参数类型 Exception 就是 try 代码块传递给 catch 代码块的异常类型，catch 代码块中语句“e.printStackTrace();”用于输出异常的堆栈跟踪信息。通常异常对象用的 3 个方法如下。

- ☑ getMessage()函数：获取异常信息。
- ☑ toString()函数：将异常对象转换为字符串，其中包括异常类型与错误信息。
- ☑ printStackTrace()函数：指出异常的类型、性质、栈层次及出现在程序中的位置。

注意：有时为了简单会忽略掉 catch 语句后的代码。这样 try…catch 语句就成了一种摆设。这样，一旦程序在运行过程中出现了异常，导致最终运行结果与期望的不一致。错误发生的原因就很难找了。要养成良好的编程习惯，最好在 catch 代码块中有处理异常的代码。

完整的异常处理语句一般都包含 finally 语句。不管程序中有无异常发生，也不管之间的 try…catch 是否顺利执行完毕，都会执行 finally 语句。

在以下 3 种特殊情况下，finally 块不会被执行。

- ☑ 在 finally 语句块中发生了异常。
- ☑ 在前面的代码中用 System.exit()退出了程序。
- ☑ 程序所在的线程死亡。

9.4 抛 出 异 常

若某个方法可能会发生异常，但不想在当前方法中处理这个异常，那么可以使用 throws 关键字将异常向上抛出，由调用者处理异常，另外 throw 关键字可以在方法中抛出异常。

9.4.1 向上抛出异常

throws 关键字可用于方法的声明中，它可以指明方法中可能会出现的异常。这样，在方法的内部就不需要处理这些异常，而交给该方法的使用者来处理。对于多个异常使用逗号分隔。

例 9.07 在项目中创建类 Shoot，在该类中创建方法 pop()，在该方法中抛出 NegativeArraySizeException 异常，在 main()方法中调用该方法，并实现异常处理。实例的代码如下：**（实例位置：光盘\mr\ym\9.07）**

```
public class Shoot {
    static void pop() throws NegativeArraySizeException {  // 定义方法并抛出 NegativeArraySizeException 异常
        int[] arr = new int[-3];                            // 创建非法数组
    }
    public static void main(String[] args) {                // main()方法
        try {                                               // try 语句处理异常信息
            pop();                                          // 调用 pop()方法
        } catch (NegativeArraySizeException e) {
            e.printStackTrace();
            System.out.println("pop()方法抛出的异常");  // 输出异常信息
        }
    }
}
```

实例运行效果如图 9.8 所示。

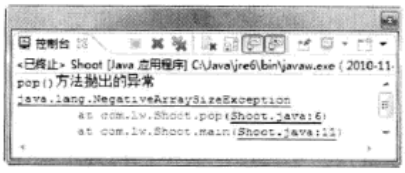


图 9.8 实例运行效果

使用 throws 关键字将异常抛给上一级方法调用者后，如果不想处理该异常，可以继续向上抛出，但最终要有能够处理该异常的代码。

📖 **说明：** 如果是 Error、RuntimeException 或它们的子类，那么可以不使用 throws 关键字来声明要抛出的异常，编译仍能顺利通过，但在运行时会被系统抛出。

9.4.2　使用 throw 关键字抛出异常

throw 关键字通常用于方法体中，它用于有意的异常抛出，例如，如果文件不存在，就抛出指定的异常。程序在执行到 throw 语句时立即终止，后面的语句都不执行，除非调用者捕获并处理了该异常。

throw 通常用来抛出用户自定义的异常，而自定义异常都是通过继承 Exception 类或其子类实现的，该类是所有可捕获异常的父类。

下面示例中介绍了如何创建自定义的异常以及如何抛出异常。

例 9.08　在项目中创建类 MyException 继承 Exception，创建自定义异常类；再创建 Captor 类，为该类中的 quotient()方法定义两个 int 型参数，如果参数 y 为负数，则会抛出 MyException 异常；最后在 main()方法中捕捉异常。（**实例位置：光盘\mr\ym\9.08**）

（1）编写 MyException 自定义异常类，代码如下：

```
public class MyException extends Exception {                    // 创建自定义异常类
    String message;                                             // 定义 String 类型变量
    public MyException(String ErrorMessagr) {                   // 父类方法
        message = ErrorMessagr;
    }
    public String getMessage() {                                // 覆盖 getMessage()方法
        return message;
    }
}
```

（2）编写 Captor 类，代码如下：

```
public class Captor {                                           // 创建类
    static int quotient(int x, int y) throws MyException {      // 定义方法抛出异常
        if (y < 0) {                                            // 判断参数是否小于 0
            throw new MyException("除数不能是负数");             // 异常信息
        }
        return x / y;                                           // 返回值
    }
    public static void main(String args[]) {                   // main()方法
        try {                                                   // try 语句包含可能发生异常的语句
            int result = quotient(3, -1);                       // 调用方法 quotient()
        } catch (MyException e) {                               // 处理自定义异常
            e.printStackTrace();
            System.out.println(e.getMessage());                 // 输出异常信息
        } catch (ArithmeticException e) {                       // 处理 ArithmeticException 异常
            System.out.println("除数不能为 0");                  // 输出提示信息
        } catch (Exception e) {                                 // 处理其他异常
            System.out.println("程序发生了其他的异常");          // 输出提示信息
        }
    }
}
```

实例运行效果如图 9.9 所示。

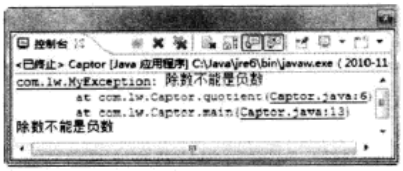


图 9.9 实例运行效果

在上面的实例中使用多个 catch 语句来捕捉异常。如果调用 quotient(3,-1)，将发生 MyException 异常，程序跳转到 catch (MyException e)代码块中执行；如果调用 quotient(5,0)，会发生 ArithmeticException 异常，程序跳转到 catch (ArithmeticException e)代码块中执行；如还有其他异常发生，将使用 catch (Exception e)捕捉异常。由于 Exception 是所有异常类的父类，如果将 catch (Exception e)代码块放在其他两个代码块的前面，后面的代码块将永远得不到执行，就没有什么意义了，所以 catch 语句的顺序不可调换。

注意：在 try…catch 语句中，多个 catch 的顺序一定要遵循子类在上父类在下的规则。

9.5 运行时异常

RuntimeException 异常是程序运行过程中产生的异常。所有异常类都是 Throwable 类的子类。Throwable 类派生了两个子类，分别是 Exception 和 Error 类。Error 类及其子类用来描述 Java 运行系统中的内部错误以及资源耗尽的错误，这类错误比较严重。Exception 类称为非致命性类，可以通过捕捉异常使程序继续执行。Exception 类根据错误发生的原因又分为 RuntimeException 异常和 RuntimeException 之外的异常，如图 9.10 所示。

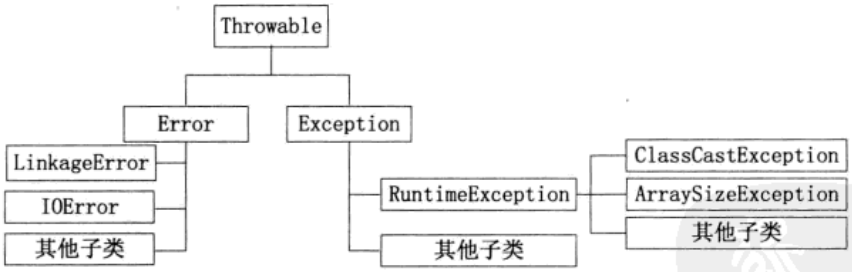


图 9.10 Java 异常类结构图

Java 中提供了常见的 RuntimeException 异常，这些异常可通过 try…catch 语句捕获，如表 9.1 所示。

表 9.1 RuntimeException 异常的种类

RuntimeException 异常的种类	说　明
NullPointerException	空指针异常类
ArrayIndexOutOfBoundsException	数组下标越界异常
ArithmeticException	算术异常类
ArrayStoreException	数组中包含不兼容的值抛出的异常
IllegalArgumentException	非法参数异常

续表

RuntimeException 异常的种类	说　明
SecurityException	安全性异常
NegativeArraySizeException	数组长度为负异常

9.6　异常的使用原则

Java 中的异常强制用户去考虑程序的健壮性和安全性，异常处理不应用控制程序流程。异常处理的主要作用是捕获程序在运行时发生的异常并进行相应的处理。

异常处理可遵循以下几条原则：

☑ 在当前方法声明中使用 try…catch 语句捕获异常。

☑ 一个方法被重写时，重写它的方法必须抛出相同的异常或异常的子类。

☑ 如果父类中方法抛出多个异常，那么子类重写该方法必须抛出那些异常的一个子集，不能抛出新异常。

9.7　照猫画虎——基本功训练

9.7.1　基本功训练 1——使用普通内部类修改窗体背景色

视频讲解：光盘\mr\lx\09\使用普通内部类修改窗体背景色.exe

实例位置：光盘\mr\ym\9.09\src

Swing 中窗体默认的背景色是白色，虽然可以使用外观来更改背景色，例如，使用 Nimbus 外观时窗体背景色是灰色，但是不同的程序对于窗体颜色可能有不同的需求。本实例将使用普通内部类完成修改窗体背景色的操作，关键代码如下：

```
private class ColorAction implements ActionListener {
    private Color background;
    public ColorAction(Color background) {
        this.background = background;                    // 获得要设置的颜色
    }
    @Override
    public void actionPerformed(ActionEvent e) {
        panel.setBackground(background);                 // 设置窗体背景色
    }
}
```

实例运行效果如图 9.11 所示。

图 9.11　实例运行效果

照猫画虎：使用普通内部类实现每隔 1 秒钟随机更换窗体背景色的功能。提示：使用多线程技术，当线程休眠 1 秒钟后更改窗体的颜色。（20 分）（实例位置：光盘\mr\ym\9.09\zmhh）

9.7.2 基本功训练 2——使用局部内部类实现闹钟

视频讲解：光盘\mr\lx\09\使用局部内部类实现闹钟.exe

实例位置：光盘\mr\ym\9.10\src

在日常生活中，闹钟是非常常见的，它可以帮助人们更好地安排时间而不会错过一些重要事件。本实例将使用局部内部类实现闹钟功能，它可以每隔 1 秒钟在控制台上输出一次当前的时间。如果用户确定退出程序，则控制台上会停止输出，关键代码如下：

```
public class AlarmClock {
    private int delay;                                                    // 表示延迟时间
    public AlarmClock(int delay) {                                        // 使用构造方法初始化各个域
        this.delay = delay;
    }
    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
            }
        }
        new Timer(delay, new Printer()).start();                          // 创建 Timer 对象并启动
    }
    public static void main(String[] args) {
        AlarmClock clock = new AlarmClock(1000);                          // 创建 AlarmClock 对象
        clock.start();                                                    // 启动 start()方法
        JOptionPane.showMessageDialog(null, "是否退出？");
        System.exit(0);                                                   // 退出程序
    }
}
```

实例运行效果如图 9.12 所示。

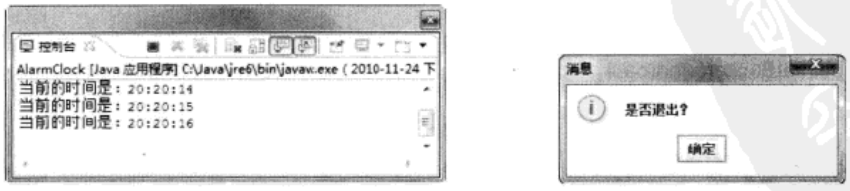


图 9.12 实例运行效果（控制台如左图，窗体如右图）

说明：程序启动后每秒钟都会在控制台输出时间，单击窗体中的“确定”按钮，退出程序。

照猫画虎：在本实例的基础上增加每秒钟发出提示声音的功能。提示：可以使用 Toolkit 类中提供的方法来发出声音。（20 分）（实例位置：光盘\mr\ym\9.10\zmhh）

9.7.3　基本功训练 3——使用匿名内部类实现图片浏览器

视频讲解：光盘\mr\lx\09\使用匿名内部类实现图片浏览器.exe

实例位置：光盘\mr\ym\9.11\src

本实例将完成一个简单的图片浏览工具，用户通过单击不同的按钮来切换在标签中显示的图片。这需要对按钮单击事件进行监听。本实例中是使用匿名内部类来完成的，这就需要对不同的按钮完成不同的事件处理代码，对于本实例略显繁琐，关键代码如下：

```
public ImageViewer() {
    setTitle("\u56FE\u7247\u6D4F\u89C8\u5668");
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setBounds(100, 100, 300, 240);
    contentPane = new JPanel();
    contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
    contentPane.setLayout(new BorderLayout(0, 0));
    setContentPane(contentPane);
    JPanel panel = new JPanel();
    contentPane.add(panel, BorderLayout.CENTER);
    panel.setLayout(new BorderLayout(0, 0));
    JPanel buttonPanel = new JPanel();
    buttonPanel.setLayout(new GridLayout(2, 3, 5, 5));
    panel.add(buttonPanel, BorderLayout.NORTH);
    JButton button1 = new JButton("\u56FE\u72471");          // 创建按钮
    button1.addActionListener(new ActionListener() {         // 为按钮增加监听器，完成替换标签图片的功能
        @Override
        public void actionPerformed(ActionEvent e) {
            label.setIcon(new ImageIcon("src/images/1.png"));
        }
    });
    buttonPanel.add(button1);                                // 应用按钮
    JButton button2 = new JButton("\u56FE\u72472");          // 创建按钮
    button2.addActionListener(new ActionListener() {         // 为按钮增加监听器，完成替换标签图片的功能
        @Override
        public void actionPerformed(ActionEvent e) {
            label.setIcon(new ImageIcon("src/images/2.png"));
        }
    });
    buttonPanel.add(button2);                                // 应用按钮
    JButton button3 = new JButton("\u56FE\u72473");          // 创建按钮
    button3.addActionListener(new ActionListener() {         // 为按钮增加监听器，完成替换标签图片的功能
        @Override
        public void actionPerformed(ActionEvent e) {
            label.setIcon(new ImageIcon("src/images/3.png"));
        }
    });
    buttonPanel.add(button3);                                // 应用按钮
    JButton button4 = new JButton("\u56FE\u72474");          // 创建按钮
    button4.addActionListener(new ActionListener() {         // 为按钮增加监听器，完成替换标签图片的功能
        @Override
```

```
            public void actionPerformed(ActionEvent e) {
                label.setIcon(new ImageIcon("src/images/4.png"));
            }
        });
        buttonPanel.add(button4);                                    // 应用按钮
        JButton button5 = new JButton("\u56FE\u72475");              // 创建按钮
        button5.addActionListener(new ActionListener() {             // 为按钮增加监听器，完成替换标签图片的功能
            @Override
            public void actionPerformed(ActionEvent e) {
                label.setIcon(new ImageIcon("src/images/5.png"));
            }
        });
        buttonPanel.add(button5);                                    // 应用按钮
        JButton button6 = new JButton("\u56FE\u72476");              // 创建按钮
        button6.addActionListener(new ActionListener() {             // 为按钮增加监听器，完成替换标签图片的功能
            @Override
            public void actionPerformed(ActionEvent e) {
                label.setIcon(new ImageIcon("src/images/6.png"));
            }
        });
        buttonPanel.add(button6);                                    // 应用按钮
        label = new JLabel("");
        label.setHorizontalAlignment(SwingConstants.CENTER);
        panel.add(label, BorderLayout.CENTER);
    }
```

实例运行效果如图 9.13 所示。

图 9.13　实例运行效果

照猫画虎：使用数组来简化本实例的代码，并用普通内部类完成显示图片的功能。（20 分）（实例位置：光盘\mr\ym\9.11\zmhh）

9.7.4　基本功训练 4——演示类未发现异常

视频讲解：光盘\mr\lx\09\演示类未发现异常.exe

实例位置：光盘\mr\ym\9.12\src

类未发现异常（ClassNotFoundException）用于表示使用了 Class 类的 forName()方法、ClassLoader 类的 findSystemClass()方法或 ClassLoader 类中的 loadClass()方法加载类，但是并没有找到这个类的情况。通常是发生在没有添加数据库驱动时。本实例将加载 MySQL 数据库驱动程序，代码如下：

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

```
        try {
            Class.forName("com.mysql.jdbc.Driver");                    // 加载 MySQL 驱动程序
        } catch (ClassNotFoundException e) {                           // 捕获异常
            e.printStackTrace();                                       // 打印堆栈信息
        }
    }
}
```

实例运行效果如图 9.14 所示。

图 9.14　实例运行效果

照猫画虎：修改 catch 语句的内容，提示用户数据库的驱动并不存在。（20 分）（实例位置：光盘\mr\ym\9.12\zmhh）

9.7.5　基本功训练 5——自定义异常类

视频讲解：光盘\mr\lx\09\自定义异常类.exe

实例位置：光盘\mr\ym\9.13\src

虽然 Java SE API 的设计者预先定义了很多异常类型，但是有些特殊情况还是没有考虑到，此时可以考虑自定义异常类。本实例将演示自定义异常类的用法。

实现过程如下：

（1）定义除零异常，它继承了算术异常类，并提供了两个构造方法，代码如下：

```
public class DivideZeroException extends ArithmeticException {      // 自定义异常类
    private static final long serialVersionUID = 1563874058117161205L;
    public DivideZeroException() {
    }                                                                // 实现默认构造方法
    public DivideZeroException(String msg) {
        super(msg);
    }                                                                // 实现有输出信息的构造方法
}
```

（2）编写测试类，输出数组元素与整数除法的结果。对于除零运算，使用了自定义异常，代码如下：

```
public class Test {
    public static void main(String[] args) {
        int[] array = new int[5];                                    // 定义长度为 5 的数组
        Arrays.fill(array, 5);                                       // 将数组中的元素赋值为 5
        for (int i = 4; i > -1; i--) {                               // 遍历整个数组
            if (i == 0) {                                            // 如果除零
                throw new DivideZeroException("发生除零异常");       // 如果除零就抛出有异常信息的构造方法
            }                                                        // 如果不是除零就输出结果
```

```
            System.out.println("array[" + i + "] / " + i + " = " + array[i] / i);
        }
    }
}
```

实例运行效果如图 9.15 所示。

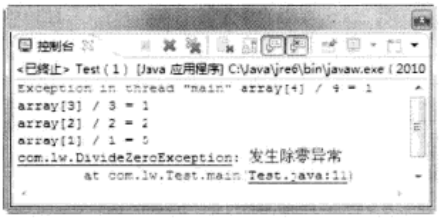


图 9.15　实例运行效果

照猫画虎：自定义对负数求平方根异常。（20 分）（实例位置：光盘\mr\ym\9.13\zmhh）

照猫画虎栏目分数统计：

照猫画虎题目	1	2	3	4	5	总分数	
分数							

9.8　情景应用——拓展与实践

9.8.1　情景应用 1——实现文字的跑马灯效果

视频讲解：光盘\mr\lx\09\实现文字的跑马灯效果.exe

实例位置：光盘\mr\ym\9.14\src

在程序运行过程中，会遇到需要实时显示信息的情况，此时可以考虑使用跑马灯效果来显示。它是除了对话框外，另一种显示信息的方式。本实例将完成这个特效，在窗体上循环显示“明日科技”4 个字，实例的运行效果如图 9.16 所示。

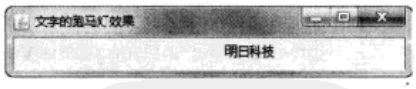


图 9.16　实例运行效果

实现过程如下：

（1）在项目中新建窗体类 DynamicTextLabel，在窗体中添加标签控件。

（2）使用普通内部类完成改变标签位置的操作，实例的完整代码如下：

```
public class DynamicTextLabel extends JFrame {
    private static final long serialVersionUID = -2035256617544621708L;
    private JPanel contentPane;
    private JLabel label;
    public static void main(String[] args) {
        try {
            UIManager.setLookAndFeel("com.sun.java.swing.plaf.nimbus.NimbusLookAndFeel");
        } catch (Throwable e) {
```

```
            e.printStackTrace();
        }
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                try {
                    DynamicTextLabel frame = new DynamicTextLabel();
                    frame.setVisible(true);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }
    public DynamicTextLabel() {
        setTitle("文字的跑马灯效果");                                    // 设置窗体的标题
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);                  // 设置关闭窗体时的动作
        setBounds(100, 100, 400, 70);                                    // 设置窗体的位置和大小
        contentPane = new JPanel();
        contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));              // 设置面板的边框
        setContentPane(contentPane);
        contentPane.setLayout(new FlowLayout(FlowLayout.LEFT, 5, 5));    // 设置面板的布局
        label = new JLabel("明日科技");                                  // 创建标签
        contentPane.add(label);
        new Thread(new DynamicThread()).start();                         // 启动新线程
    }
    private class DynamicThread implements Runnable {                    // 使用内部类完成标签移动操作
        @Override
        public void run() {
            while (true) {
                for (int i = 0; i < 400; i++) {
                    try {
                        Thread.sleep(10);                                // 线程休眠 0.01 秒
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                    label.setLocation(i, 5);                             // 移动标签
                }
            }
        }
    }
}
```

DIY：在本实例的基础上实现随机修改文本颜色的功能。（20 分）（实例位置：光盘\mr\ym\9.14\diy）

9.8.2　情景应用 2——组装电脑

视频讲解：光盘\mr\lx\09\组装电脑.exe

实例位置：光盘\mr\ym\9.15\src

对于电脑而言，是由不同的硬件构成的，如 CPU、内存、硬盘等。本实例将使用普通内部类完成组装

电脑的任务，对于不同的硬件使用不同的类来表示，实例的运行效果如图 9.17 所示。

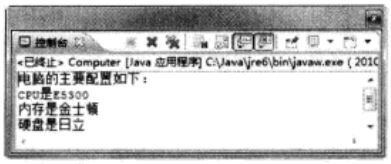


图 9.17　实例运行效果

实现过程如下：

（1）创建项目，并在项目中新建类 Computer。

（2）在 Computer 类中，内部创建类 CPU、Memory 和 HardDisc 分别表示电脑的 CPU、内存和硬盘。对于不同的内部类，均重写了其 toString()方法，这样可以方便地输出内部类的对象。在 main()方法中，首先创建了一个外部类对象，然后使用其完成内部类对象的创建，实例代码如下：

```
public class Computer {
    private class CPU {                                    // 创建内部类表示电脑的 CPU
        @Override
        public String toString() {
            return "CPU 是 E5300";
        }
    }
    private class Memory {                                 // 创建内部类表示电脑的内存
        @Override
        public String toString() {
            return "内存是金士顿";
        }
    }
    private class HardDisc {                               // 创建内部类表示电脑的硬盘
        @Override
        public String toString() {
            return "硬盘是日立";
        }
    }
    public static void main(String[] args) {
        System.out.println("电脑的主要配置如下：");
        Computer computer = new Computer();
        System.out.println(computer.new CPU());
        System.out.println(computer.new Memory());
        System.out.println(computer.new HardDisc());
    }
}
```

你问我答：computer.new CPU()代码的作用是什么？

本代码并非是 Java API 中提供的方法，它用于创建一个内部类对象。对于非静态内部类而言，需要使用外部类的对象才能在静态方法中创建，这有些类似于使用类的对象来引用类中非静态变量或方法。

DIY：在本实例的基础上增加显卡内部类并输出其配置。（20 分）（实例位置：光盘\mr\ym\9.15\diy）

9.8.3　情景应用 3——跟随鼠标移动的文字

视频讲解：光盘\mr\lx\09\跟随鼠标移动的文字.exe

实例位置：光盘\mr\ym\9.16\src

在浏览网页的过程中，经常遇到使用 JavaScript 完成的精美特效，这些特效使用 Swing 的事件监听机制也能够实现。本实例将完成跟随鼠标移动的文字特效，这是通过监听鼠标移动事件而定位标签位置实现的，实例的运行效果如图 9.18 所示。

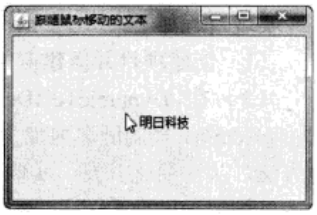


图 9.18　实例运行效果

实现过程如下：

（1）创建项目并新建窗体类 DynamicTextDemo。

（2）在 DynamicTextDemo 类的构造方法中增加鼠标移动事件监听。它首先将标签的文本设置成“明日科技”，然后获得了鼠标的坐标，根据这个坐标来不断改变标签的位置。这样就实现了跟随鼠标移动的文字特效，实例的关键代码如下：

```
public DynamicTextDemo() {
    setTitle("跟随鼠标移动的文本");                                    // 设置窗体标题
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);                    // 设置窗体关闭时的动作
    setBounds(100, 100, 300, 200);                                     // 设置窗体的大小和位置
    contentPane = new JPanel();                                        // 创建面板
    contentPane.addMouseMotionListener(new MouseMotionAdapter() {      // 监听鼠标移动事件
            @Override
            public void mouseMoved(MouseEvent e) {
                label.setText("明日科技");                             // 为标签设置文本信息
                Point mousePoint = e.getPoint();                       // 获得鼠标的位置
                label.setLocation(mousePoint.x + 15, mousePoint.y);    // 更新标签的位置
            }
        });
    contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));               // 设置边框
    setContentPane(contentPane);
    contentPane.setLayout(null);                                       // 使用绝对布局
    label = new JLabel("");                                            // 创建标签
    label.setBounds(6, 6, 55, 18);                                     // 设置标签大小和位置
    contentPane.add(label);                                            // 应用标签
}
```

你问我答：mouseMoved()方法有什么作用？

该方法可用于完成鼠标移动事件的处理。方法的参数是 MouseEvent 类型的对象，它可以很方便地获得鼠标的位置坐标等信息。

DIY：在本实例的基础上修改文本的颜色，让其每隔 0.1 秒随机变化成另一种颜色。（20 分）（实例位置：光盘\mr\ym\9.16\diy）

9.8.4　情景应用 4——鼠标经过时改变文本颜色

视频讲解：光盘\mr\lx\09\鼠标经过时改变文本颜色.exe

实例位置：光盘\mr\ym\9.17\src

对于 Swing 中的标签控件，可以增加鼠标进入标签和移出标签事件的监听，通过在两个方法中修改标

签文本的颜色就可以实现一些好玩的特效。本实例将鼠标进入标签后文本设置成红色，而移出标签后将文本设置成黑色，实例的运行效果如图 9.19 所示。

图 9.19　实例运行效果（鼠标进入标签如左图，鼠标移出标签如右图）

实现过程如下：

（1）创建项目并新建窗体类 DynamicTextDemo。

（2）在 DynamicTextDemo 类的构造方法中增加鼠标进入和移出标签事件监听，这是通过创建 MouseAdapter 类的匿名内部类完成的。使用 MouseAdapter 类可以避免实现 MouseListener 等接口规定的其他方法，从而简化开发，实例的关键代码如下：

```
public DynamicTextDemo() {
    setTitle("鼠标经过时改变文本颜色");                                    // 设置窗体标题
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);                      // 设置窗体关闭时的动作
    setBounds(100, 100, 300, 70);                                        // 设置窗体的大小和位置
    contentPane = new JPanel();                                          // 创建面板
    contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));                  // 设置边框
    setContentPane(contentPane);
    contentPane.setLayout(new FlowLayout(FlowLayout.CENTER, 5, 5));      // 使用流式布局
    label = new JLabel("明日科技");                                        // 创建标签
    label.addMouseListener(new MouseAdapter() {
        @Override
        public void mouseEntered(MouseEvent e) {                         // 监听鼠标进入标签事件
            label.setForeground(Color.RED);                              // 将标签上的文本设置成红色
        }
        @Override
        public void mouseExited(MouseEvent e) {                          // 监听鼠标离开标签事件
            label.setForeground(Color.BLACK);                            // 将标签上的文本设置成黑色
        }
    });
    contentPane.add(label);
}
```

DIY：在本实例的基础上增加文本颜色随机变换的功能，让每次鼠标进入到标签控件时文本颜色都不同。（20 分）（实例位置：光盘\mr\ym\9.17\diy）

9.8.5　情景应用 5——在鼠标上显示当前时间

视频讲解：光盘\mr\lx\09\在鼠标上显示当前时间.exe

实例位置：光盘\mr\ym\9.18\src

除了让一些广告信息随鼠标移动，对于用户而言显示当前时间更为有用。本实例将完成在鼠标上显示当前时间的功能，时间的格式是“小时:分钟:秒钟”。对于格式化时间样式，可以使用 SimpleDateFormat 类，实例的运行效果如图 9.20 所示。

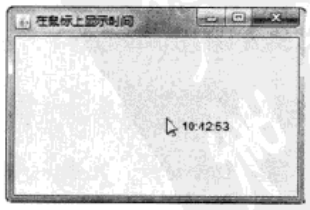


图 9.20　实例运行效果

实现过程如下：

（1）创建项目并新建窗体类 DynamicTextDemo。

（2）在 DynamicTextDemo 类的构造方法中增加鼠标移动事件监听。使用 SimpleDateFormat 类来辅助格式化时间。实例中使用的样式是“hh:mm:ss”，字符串中 h 表示小时，m 表示分钟，s 表示秒钟。将标签在鼠标右侧 15 像素显示，可以避免标签和鼠标指针重叠，实例的关键代码如下：

```
public DynamicTextDemo() {
    setTitle("在鼠标上显示时间");                                          // 设置窗体标题
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);                        // 设置窗体关闭时的动作
    setBounds(100, 100, 300, 200);                                         // 设置窗体的大小和位置
    contentPane = new JPanel();                                            // 创建面板
    contentPane.addMouseMotionListener(new MouseMotionAdapter() {          // 监听鼠标移动事件
            @Override
            public void mouseMoved(MouseEvent e) {
                SimpleDateFormat format = new SimpleDateFormat("hh:mm:ss"); // 设置时间的样式
                label.setText(format.format(new Date()));                  // 为标签设置文本信息
                Point mousePoint = e.getPoint();                           // 获得鼠标的位置
                label.setLocation(mousePoint.x + 15, mousePoint.y);        // 更新标签的位置
            }
        });
    contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));                    // 设置边框
    setContentPane(contentPane);
    contentPane.setLayout(null);                                           // 使用绝对布局
    label = new JLabel("");                                                // 创建标签
    label.setBounds(6, 6, 55, 18);                                         // 设置标签大小和位置
    contentPane.add(label);                                                // 应用标签
}
```

DIY：在本实例的基础上修改显示的文本样式为“年月日”。（20 分）（实例位置：光盘\mr\ym\9.18\diy）

情景应用 DIY 栏目分数统计：

DIY 题目	1	2	3	4	5	总分数	
分数							

9.9　自 我 测 试

一、选择题（每题 10 分，5 道题）

1．如果不希望类 Key 被继承，则如何定义该类？（　　）

A．public class Key{}　　B．public abstract class Key{}　　C．public final class Key{}

D．public native class Key{}　　E．public void class Key{}

2．当编译和运行下列代码时会发生什么？（　　）

```
public class Test {
    private final int x;
    public static void main(String[] args) {
        System.out.println(x);
    }
}
```

A．编译错误　B．运行错误　　C．输出 0　　D．输出 10

3．当发生 IOException 时，下列代码的输出结果是？（　　）

```
public class Test {
    public static void main(String[] args) {
        int number;
        try {
            number = System.in.read();
            System.out.println("Location 1");
        } catch (IOException e) {
            System.out.println("Location 2");
        } finally {
            System.out.println("Location 3");
        }
        System.out.println("Location 4");
    }
}
```

A．Location1　　B．Location2　　C．Location3　　D．Location4

4．当编译和运行下列代码时会发生什么？（　　）

```
public class Test {
    public static void main(String[] args) {
        try {
            return;
        } catch (Exception e) {
            System.out.println("Exception");
        } finally {
            System.out.println("Finally");
        }
    }
}
```

A．编译错误　　B．运行错误　　C．输出 Exception 和 Finally　　D．输出 Finally

5．当编译和运行下列代码时会发生什么？（　　）

```
public class A {
    private int x;
    public class B {
        public B() {
            System.out.println(x);
        }
    }
    public static void main(String[] args) {
        new B();
    }
}
```

A．编译失败，内部类尝试访问外部类变量

B．编译失败，内部类不能是 public 修饰的

C．编译失败，内部类不是静态的，因此需要先获得外部类对象才能访问

D．运行成功

二、填空题（每题 10 分，5 道题）

1．如果需要在声明方法时抛出异常，则应该使用的关键字是_____。

2．如果使用 final 修饰类，则该类还可以被继承吗？_____

3．对于内部类而言，可以使用 static 关键字修饰吗？_____

4．判断：对于匿名内部类，如果需要使用外部类或者方法中的变量，则该变量必须是静态的。_____

5．对于非静态内部类，可以直接在 main()方法中创建其对象吗？_____

测试分数统计：

类别	第 1 题	第 2 题	第 3 题	第 4 题	第 5 题
选择题分数					
填空题分数					
总分数					

9.10　行 动 指 南

开始日期：_____年____月____日　　结束日期：_____年____月____日

序号	内　容	行 动 指 南	
1	照猫画虎栏目 分数（　　）	分数>75 分	优秀，基本功掌握得不错，加油！
		75 分>分数>50 分	及格，知识掌握得不牢，重新做一遍照猫画虎。
		分数<50 分	请使用光盘中提供的“实战能力测试系统”进行提高训练。
	情景应用栏目 分数（　　）	分数>75 分	优秀，综合应用能力很强。
		75 分>分数>50 分	及格，综合应用能力需提高，再练一遍情景应用。
		分数<50 分	请使用光盘中提供的“实战能力测试系统”进行提高训练。
	自我测试栏目 分数（　　）	分数>75 分	优秀，有成为编程高手的潜质。
		分数<75 分	请使用光盘中提供的“实战能力测试系统”进行提高训练。
	综合评价	反复训练后，以上各项分数都在 75 分以上，方可进入下一堂课学习。	
2	编程实践能力培养：本栏目提供了一些实践题目，对于培养编程能力很有效，要做好。	（1）实现跟随鼠标跳动的文字。	
		（2）从天而降的文字效果。	
		（3）三角函数计算器。	
		（4）单击后隐藏的按钮控件。	
3	编程习惯培养：本堂课培养读者在学习开发中记笔记的习惯，把开发中遇到的问题、总结的经验记录下来。		
4	创新能力培养：看看身边有没有可以用编程解决的问题，记录到右边的表格中。根据学习进度，尝试编写解决这些问题的小程序。		

9.11 成功可以复制——80 后新贵、泡泡网 CEO 李想

1981 年，李想出生在河北石家庄。小时候，李想不太喜欢课堂，他总喜欢在实践中快学快用。需要什么才学什么，学了什么就马上用起来。上初中时，老师的一句激励“学习不好不要紧，但一定要做个优秀的人”成为他在课堂上所学到的最有价值的东西。高三时，李想迷上了个人网站，他把所有的时间都用在计算机上，像许多电脑迷一样，他也建了一个个人网站，“一开始是自己做着玩，但我这个人喜欢争强好胜，别人做得好，我就要比别人做得更好。”他把自己喜欢的电脑硬件产品都放在网上，有很多人上网和他交流，慢慢地就有了访问量，几个月后访问量达到 1 万人次/天。这时候，广告商就找上门来。当时李想的网站每个月有 6000、7000 元的广告收入，这对一个学生来说，简直太奢侈了。

但好景不长。1999 年下半年互联网泡沫破灭，李想的广告一个都没了。虽然遭遇挫折，但李想并不气馁，因为做网站是一个可以让他全力以赴的事情，也是他特别喜欢的事情。高中毕业后，李想没有选择继续读书，而是自己创业。2000 年，李想和一个朋友创办了 PCPOP（电脑泡泡）网站，初始投资就是做网站淘到的第一桶金，将近 10 万元。新网站很快就有了访问量，但却见不着效益，因为在石家庄没有收入机会，李想决定移师北京。

2001 年年底，李想到了北京，开始租了一间民房，半年后，网站访问量每天有 3～5 万人，广告商又找上门来，而且开出了更高的广告价格。2002 年，他们搬到写字楼，2003 年他们的收入达到 200 万，2004 年超过了 1 千万。李想是如何实现爆发式增长的呢？一方面，PCPOP 网站的内容基本上都是原创的，公司身居中关村，每天都能关注市场的最新动态，另外厂商也会把最新的产品拿来给他们测评，他们的专业性特征明显，而且更新速度非常快；另一方面，这些年市场消费行为也在发生变化。原先顾客要买一件商品，可能到电脑市场转上好几天，而现在更多的用户可以先在网上找到想要的东西，谁的网站专业性强、更新速度快，自然就会吸引更多的眼球。

因为兴趣和爱好，李想在 24 岁就成就了自己的互联网传奇。2010 年 4 月，李想成为搜狐网的首席策划师，我们期待着他更大的成功！

经典语录

做事要坚定，我当初如果不放弃考大学，也许今天我只是一名普通的打工者；做事要认真，如果做一件事情比别人多付出 5%的努力，就有可能拿到比别人多 200%的回报。

深度评价

在专业 IT 网站前五名中，除了李想的 PCPOP 网站是白手起家做起来的，其他都是靠几千万美金的投资“砸”出来的。李想的成功，得益于他做事认真、全力以赴的态度；得益于他“比别人多付出 5%的努力”的精神；得益于他“学习不好不要紧，但一定要做个优秀的人”的人生信条。

第10堂课

Swing 编程基础

（视频讲解：153 分钟）

Swing 是在早期的 AWT 基础上建立的。AWT 中很多控件都是重量级的，即和底层操作系统的关联很大。这样的缺点在于不同系统上创建的控件外观可能并不相同，而且不同的平台上运行 AWT 程序可能有不同的 bug。这有违 Java 的“一次编写，到处运行”口号。Swing 中大多数控件是轻量级控件，它们可以在不同的系统上显示统一的外观。当然这个外观肯定是和具体的系统有些差异。因此，Swing 针对不同的系统还提供了不同的外观供用户选择，这样就可以让程序与系统统一了。

学习摘要：

- 窗体控件的应用
- 使用布局管理器
- 文本控件
- 标签和按钮
- 列表控件
- 进度条

10.1 JFrame 窗体控件

Swing 中的控件都需要放置到窗体中才能显示和使用，常见的窗体有 JFrame、JDialog 和 JApplet。本节重点讲述 JFrame 的使用，该类位于 javax.swing 包中。在实际开发中，通常继承该类进行编程。

例 10.01 下面通过一个简单的实例演示如何创建窗体。实例的代码如下：**（实例位置：光盘\mr\ym\10.01）**

```
public class JFrameDemo extends JFrame {
    private static final long serialVersionUID = -8501438415481947988L;
    public static void main(String[] args) {
        try {                                                          // 修改窗体的外观为 Nimbus
            UIManager.setLookAndFeel("com.sun.java.swing.plaf.nimbus.NimbusLookAndFeel");
        } catch (Throwable e) {
            e.printStackTrace();
        }
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                try {
                    JFrameDemo frame = new JFrameDemo();               // 创建窗体对象
                    frame.setVisible(true);                            // 将窗体设置成可见
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }
    public JFrameDemo() {
        setTitle("最简单的窗体");                                       // 设置窗体的标题
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);                // 设置窗体退出时操作
        setBounds(100, 100, 250, 100);                                 // 设置窗体位置和大小
    }
}
```

实例运行效果如图 10.1 所示。

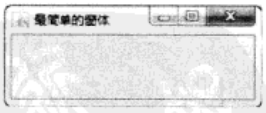


图 10.1 实例运行效果

在创建窗体时有以下要点必须注意：

- ☑ 窗体默认是不可见的，需要使用 setVisible()方法将其设置成可见。
- ☑ 窗体默认是没有大小的，需要使用 setBounds()方法设置显示位置和大小。
- ☑ 窗体默认是不能关闭的，需要使用 setDefaultCloseOperation()方法修改这个属性。

对于一个窗体而言，可以采用如下方式进行简单美化：

- ☑ 修改默认的外观，使用 setLookAndFeel()方法将窗体设置成需要的外观，此时会影响到容器中所有控件的默认外观。
- ☑ 为窗体增加标题，使用 setTitle()方法可以将指定的字符串设置成窗体标题。
- ☑ 修改窗体的图标，刚刚创建的窗体左上角使用的是 Java 图标，可以使用 setIconImages()方法将其修改成自己喜欢的图标。

说明：刚创建的窗体能够支持最大化和最小化，这两个功能不需要编程完成。

注意：使用 EventQueue 来启动程序是因为 Swing 不是线程安全的，这样可以避免线程相关问题。

当单击“关闭”按钮时，可以设置窗体如何响应这个操作。例 10.01 中将这个操作设置成关闭窗体。还有其他可以选择的操作，它们都定义在 WindowConstants 接口中，总结如表 10.1 所示。

表 10.1　WindowConstants 接口中定义的常量

常　量	说　明
DO_NOTHING_ON_CLOSE	该操作什么都不做，也不关闭窗体
HIDE_ON_CLOSE	这是窗体对象的默认关闭方式，它只将窗体隐藏，再次显示窗体时，恢复原样
DISPOSE_ON_CLOSE	销毁窗体对象，并关闭窗体
EXIT_ON_CLOSE	退出程序，并关闭窗体

10.2　常用布局管理器

在 Swing 中增加控件时，需要考虑控件的位置和大小。如果不使用布局管理器，则需要先在纸上画好各个控件的位置和计算控件间的距离，再向窗体中添加，这样虽然能够灵活地控制控件的位置，实现却非常麻烦。为了加快开发速度，Java 提供了一些布局管理器，它们可以将控件进行统一的管理，这样程序员就不需要考虑控件是否会重叠等问题。本节将向读者介绍 Swing 中的常用布局管理器，包括流式布局、边界布局、网格布局等。

10.2.1　绝对布局

Swing 中窗体具有默认布局管理器，如 JFrame 使用的是边界布局。虽然布局管理器能够简化程序的开发，但是为了获得最大的灵活性，可以使用绝对布局，即不使用任何布局管理器，使用该布局的步骤如下：

（1）使用 Container.setLayout(null)方式取消布局管理器。

（2）使用 Component.setBounds()方法来设置每个控件的大小与位置。

例 10.02　下面通过一个简单的实例演示如何使用绝对布局。实例的关键代码如下：**（实例位置：光盘\mr\ym\10.02）**

```
public AbsoluteLayoutDemo() {
    setTitle("绝对布局");                                          // 设置窗体的标题
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);               // 设置窗体退出时操作
    setBounds(100, 100, 250, 100);                                // 设置窗体位置和大小
    contentPane = new JPanel();                                   // 创建内容面板
    contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));           // 设置面板的边框
    setContentPane(contentPane);                                  // 应用内容面板
    contentPane.setLayout(null);                                  // 设置内容面板的布局管理器为 null
    JButton button1 = new JButton("按钮 1");                       // 创建按钮
    button1.setBounds(6, 6, 90, 30);                              // 设置按钮的大小和位置
    contentPane.add(button1);                                     // 在面板上增加按钮
    JButton button2 = new JButton("按钮 2");                       // 创建按钮
    button2.setBounds(138, 26, 90, 30);                           // 设置按钮的大小和位置
```

```
        contentPane.add(button2);                                    // 在面板上增加按钮
    }
```

实例运行效果如图 10.2 所示。

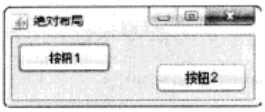


图 10.2　实例运行效果

setBounds()方法的定义如下：

```
public void setBounds(int x, int y, int width, int height)
```

其中 4 个参数的说明如表 10.2 所示。

表 10.2　setBounds()方法参数说明

参 数 名	说　明
x	控件的 x 坐标
y	控件的 y 坐标
width	控件的新宽度
height	控件的新高度

10.2.2　流式布局管理器

流式布局是面板的默认布局，它将控件按从左至右或从右至左的方向排列，这非常类似于段落中的文本行。流的方向取决于容器的 componentOrientation 属性，它可能是以下两个值中的一个。

☑ ComponentOrientation.LEFT_TO_RIGHT：顺序是从左向右（默认）。

☑ ComponentOrientation.RIGHT_TO_LEFT：顺序是从右向左。

流布局一般用来安排面板中的按钮或者标签和文本框。它使得按钮呈水平放置，直到同一条线上再也没有适合的按钮。线的对齐方式由 align 属性确定，可能的值为 LEFT、RIGHT、CENTER、LEADING 和 TRAILING，这些值的说明如表 10.3 所示。

表 10.3　流式布局对齐参数说明

常　量	说　明
LEFT	每一行中控件都是左对齐
RIGHT	每一行中控件都是右对齐
CENTER	每一行中控件都是居中对齐
LEADING	每一行中控件都是与容器开始边对齐
TRAILING	每一行中控件都是与容器结束边对齐

例 10.03　下面通过一个简单的实例演示如何使用流式布局。实例的关键代码如下：**（实例位置：光盘\mr\ym\10.03）**

```
public FlowLayoutDemo() {
    setTitle("流式布局");                                            // 设置窗体的标题
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);                  // 设置窗体退出时操作
    setBounds(100, 100, 300, 150);                                   // 设置窗体位置和大小
```

```
        contentPane = new JPanel();                                    // 创建内容面板
        contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));            // 设置面板的边框
        setContentPane(contentPane);                                   // 应用内容面板
        contentPane.setLayout(new FlowLayout(FlowLayout.CENTER, 5, 5));
                                                                       // 设置内容面板的布局管理器为流式布局
        JButton button1 = new JButton("按钮 1");                       // 创建按钮
        contentPane.add(button1);                                      // 在面板上增加按钮
        JButton button2 = new JButton("按钮 2");                       // 创建按钮
        contentPane.add(button2);                                      // 在面板上增加按钮
        JButton button3 = new JButton("按钮 3");                       // 创建按钮
        contentPane.add(button3);                                      // 在面板上增加按钮
        JButton button4 = new JButton("按钮 4");                       // 创建按钮
        contentPane.add(button4);                                      // 在面板上增加按钮
        JButton button5 = new JButton("按钮 5");                       // 创建按钮
        contentPane.add(button5);                                      // 在面板上增加按钮
        JButton button6 = new JButton("按钮 6");                       // 创建按钮
        contentPane.add(button6);                                      // 在面板上增加按钮
        JButton button7 = new JButton("按钮 7");                       // 创建按钮
        contentPane.add(button7);                                      // 在面板上增加按钮
        JButton button8 = new JButton("按钮 8");                       // 创建按钮
        contentPane.add(button8);                                      // 在面板上增加按钮
        JButton button9 = new JButton("按钮 9");                       // 创建按钮
        contentPane.add(button9);                                      // 在面板上增加按钮
}
```

实例运行效果如图 10.3 所示。

图 10.3　实例运行效果（将左图的宽度增加后变成右图）

说明：从图中可以看到流式布局中按钮是依次排列的，并且每行的按钮都是居中显示。

10.2.3　边界布局管理器

边界布局是窗体的默认布局，它将窗体分成了 5 部分，即东、西、南、北和中。通过使用不同的参数将控件放置在不同的位置，这些参数的说明如表 10.4 所示。

表 10.4　边界布局位置参数说明

常　量	说　明
EAST	将控件放置在窗体的东部
WEST	将控件放置在窗体的西部
SOUTH	将控件放置在窗体的南部
NORTH	将控件放置在窗体的北部
CENTER	将控件放置在窗体的中部

例 10.04 下面通过一个简单的实例演示如何使用边界布局。实例的关键代码如下：**（实例位置：光盘\mr\ym\10.04）**

```
public BorderLayoutDemo() {
    setTitle("边界布局");                                    // 设置窗体的标题
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);          // 设置窗体退出时操作
    setBounds(100, 100, 450, 300);                           // 设置窗体位置和大小
    contentPane = new JPanel();                              // 创建内容面板
    contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));      // 设置面板的边框
    contentPane.setLayout(new BorderLayout(0, 0));           // 设置内容面板的布局管理器为边界布局
    setContentPane(contentPane);                             // 应用内容面板
    JButton westButton = new JButton("西部");                 // 创建按钮
    contentPane.add(westButton, BorderLayout.WEST);          // 在面板上增加按钮
    JButton southButton = new JButton("南部");                // 创建按钮
    contentPane.add(southButton, BorderLayout.SOUTH);        // 在面板上增加按钮
    JButton eastButton = new JButton("东部") ;                // 创建按钮
    contentPane.add(eastButton, BorderLayout.EAST);          // 在面板上增加按钮
    JButton northButton = new JButton("北部");                // 创建按钮
    contentPane.add(northButton, BorderLayout.NORTH);        // 在面板上增加按钮
    JButton centreButton = new JButton("中部");               // 创建按钮
    contentPane.add(centreButton, BorderLayout.CENTER);      // 在面板上增加按钮
}
```

实例运行效果如图 10.4 所示。

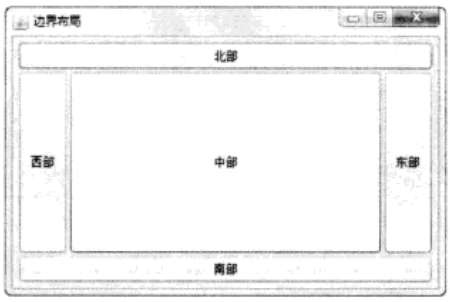


图 10.4 实例运行效果

10.2.4 网格布局管理器

网格布局将窗体划分成一个表格，可以在创建网格布局管理器时指定表格的行和列。默认情况下，是按照从左到右、从上到下组织增加到窗体中的控件的，而且每一个控件都会填充整个单元格，即改变窗体的大小时控件也会随之改变大小。

例 10.05 下面通过一个简单的实例演示如何使用网格布局。实例的关键代码如下：**（实例位置：光盘\mr\ym\10.05）**

```
public GridLayoutDemo() {
    setTitle("边界布局");                                    // 设置窗体的标题
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);          // 设置窗体退出时操作
    setBounds(100, 100, 450, 300);                           // 设置窗体位置和大小
    contentPane = new JPanel();                              // 创建内容面板
    contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));      // 设置面板的边框
```

```
        setContentPane(contentPane);                                    // 应用内容面板
        contentPane.setLayout(new GridLayout(3, 3, 5, 5));              // 使用 3 行 3 列的网格布局
        JButton button1 = new JButton("按钮 1");                        // 创建按钮
        contentPane.add(button1);                                       // 在面板上增加按钮
        JButton button2 = new JButton("按钮 2");                        // 创建按钮
        contentPane.add(button2);                                       // 在面板上增加按钮
        JButton button3 = new JButton("按钮 3");                        // 创建按钮
        contentPane.add(button3);                                       // 在面板上增加按钮
        JButton button4 = new JButton("按钮 4");                        // 创建按钮
        contentPane.add(button4);                                       // 在面板上增加按钮
        JButton button5 = new JButton("按钮 5");                        // 创建按钮
        contentPane.add(button5);                                       // 在面板上增加按钮
        JButton button6 = new JButton("按钮 6");                        // 创建按钮
        contentPane.add(button6);                                       // 在面板上增加按钮
        JButton button7 = new JButton("按钮 7");                        // 创建按钮
        contentPane.add(button7);                                       // 在面板上增加按钮
        JButton button8 = new JButton("按钮 8");                        // 创建按钮
        contentPane.add(button8);                                       // 在面板上增加按钮
        JButton button9 = new JButton("按钮 9");                        // 创建按钮
        contentPane.add(button9);                                       // 在面板上增加按钮
    }
```

实例运行效果如图 10.5 所示。

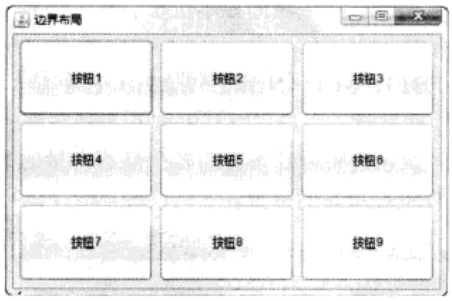


图 10.5　实例运行效果

10.2.5　网格组布局管理器

由 GridBagLayout 实现的布局管理器称为网格组布局管理器，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，每个控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 类管理的窗体中添加控件时，需要为每个控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控件的布局信息，例如，控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 实现的矩形网格的绘制方向由窗体决定，如果窗体的方向是从左到右，则位于矩形网格左上角的单元格的列索引为 0，此时控件左上角的点为起始点；如果窗体的方向是从右到左，则位于矩形网格右上角的单元格的列索引为 0，此时控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，以及在使用过程中的注意事项。

☑ gridx 和 gridy 属性

这两个属性用来设置控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，所以它表示的是控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，所以它表示的是控件起始点所在行的索引，如图 10.6 所示。

☑ gridwidth 和 gridheight 属性

这两个属性用来设置控件占用网格组的行数和列数。属性 gridwidth 为控件占用网格组的列数，也可以理解为以单元格为单位控件的宽度；属性 gridheight 为控件占用网格组的行数，也可以理解为以单元格为单位控件的高度，如图 10.7 所示。

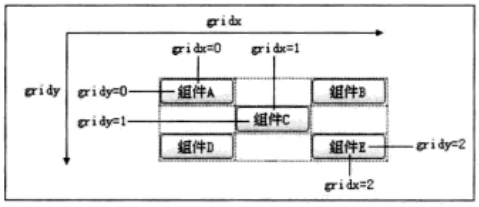


图 10.6 gridx 和 gridy 属性

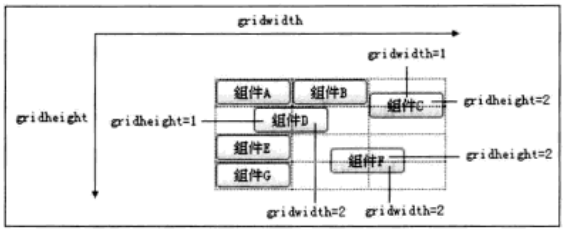


图 10.7 gridwidth 和 gridheight 属性

☑ anchor 属性

属性 anchor 用来设置控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 10.8 所示。代表这 9 个方位的单词也是该类中的静态常量，用户可以利用这 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 10.9 所示为将控件设置为各个静态常量的效果，图中的黑点为将相应控件的 anchor 属性设置为 CENTER 时控件中心所在的位置。

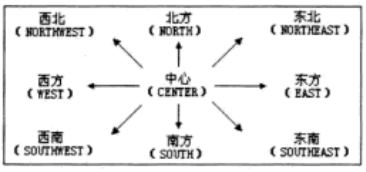


图 10.8 从方向上划分的 9 个方位

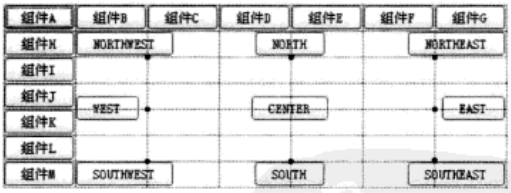


图 10.9 anchor 属性

说明：图 10.9 中控件 A～M 是辅助控件，起到占位作用。

☑ fill 属性

属性 fill 用来设置控件的填充方式。当单元格显示区域的面积大于控件面积时，或者一个控件占用多个单元格时，显示控件可能不必占用所有显示区域，在这种情况下可以通过 fill 属性设置控件的填充方式。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，即不调整控件大小至填满显示区域；如果将该属性设置为静态常量 HORIZONTAL，表示只调整控件水平方向的大小（即控件宽度）至填满显示区域；如果将该属性设置为静态常量 VERTICAL，表示只调整控件垂直方向的大小（即控件高度）至填满显示区域；如果将该属性设置为静态常量 BOTH，则表示同时调整控件的宽度和高度至填满显示

区域，具体效果如图 10.10 所示。

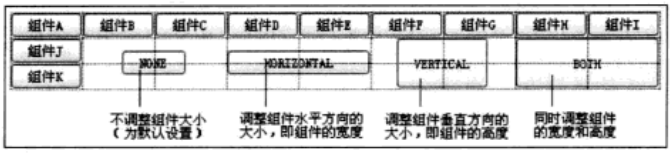


图 10.10　fill 属性

☑　insets 属性

属性 insets 用来设置控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 个入口参数依次为控件上方、左侧、下方和右侧的最小距离，单位为像素，如图 10.11 所示，默认为没有距离。

☑　ipadx 和 ipady 属性

这两个属性用来修改控件的首选大小。属性 ipadx 用来修改控件的宽度，属性 ipady 用来修改控件的高度。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 10.12 左图所示；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 10.12 右图所示。

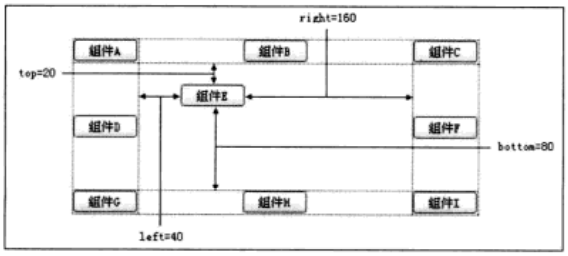


图 10.11　insets 属性

图 10.12　ipadx 和 ipady 属性（正数如左图，负数如右图）

☑　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 10.13 所示的效果，控件全部聚集在窗体的中央，在控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。

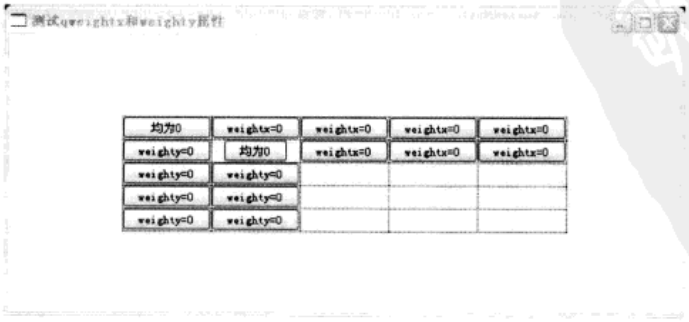


图 10.13　未设置属性 weightx 和 weighty

这两个属性的默认值均为 0，表示不分布窗体的额外空间。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 10.14 所示。

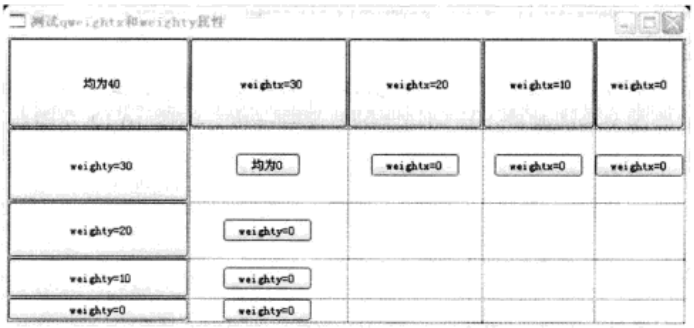


图 10.14　设置了属性 weightx 和 weighty

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，如图 10.14 所示。

例 10.06　下面通过一个简单的实例演示如何使用网格组布局。实例的关键代码如下：**（实例位置：光盘\mr\ym\10.06）**

```
public GridBagLayoutDemo() {
    setTitle("网格组布局");                                          // 设置窗体的标题
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);                  // 设置窗体退出时操作
    setBounds(100, 100, 450, 200);                                   // 设置窗体位置和大小
    contentPane = new JPanel();                                      // 创建内容面板
    contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));              // 设置面板的边框
    setContentPane(contentPane);                                     // 应用内容面板
    GridBagLayout gbl_contentPane = new GridBagLayout();             // 创建网格组布局
    contentPane.setLayout(gbl_contentPane);                          // 使用网格组布局
    JButton button1 = new JButton("A");                              // 创建按钮
    GridBagConstraints gbc_button1 = new GridBagConstraints();       // 创建网格组布局约束条件
    gbc_button1.insets = new Insets(0, 0, 5, 5);                     // 设置控件的空白
    gbc_button1.fill = GridBagConstraints.HORIZONTAL;                // 设置填充方式
    gbc_button1.weightx = 10.0;                                      // 第一列的分布方式为 10%
    gbc_button1.gridx = 0;                                           // 起始点为第 1 列
    gbc_button1.gridy = 0;                                           // 起始点为第 1 行
    contentPane.add(button1, gbc_button1);                           // 增加按钮及其约束条件
    JButton button2 = new JButton("B");                              // 创建按钮
```

```
GridBagConstraints gbc_button2 = new GridBagConstraints();          // 创建网格组布局约束条件
gbc_button2.insets = new Insets(0, 5, 5, 5);                        // 设置控件的空白
gbc_button2.fill = GridBagConstraints.HORIZONTAL;                   // 设置填充方式
gbc_button2.weightx = 20.0;                                         // 第一列的分布方式为 20%
gbc_button2.gridx = 1;                                              // 起始点为第 2 列
gbc_button2.gridy = 0;                                              // 起始点为第 1 行
contentPane.add(button2, gbc_button2);                              // 增加按钮及其约束条件
JButton button3 = new JButton("C");                                 // 创建按钮
GridBagConstraints gbc_button3 = new GridBagConstraints();          // 创建网格组布局约束条件
gbc_button3.gridheight = 2;                                         // 占用 2 列
gbc_button3.fill = GridBagConstraints.BOTH;                         // 设置填充方式
gbc_button3.weightx = 30.0;                                         // 第一列的分布方式为 30%
gbc_button3.insets = new Insets(0, 0, 5, 5);                        // 设置控件的空白
gbc_button3.gridx = 2;                                              // 起始点为第 3 列
gbc_button3.gridy = 0;                                              // 起始点为第 1 行
contentPane.add(button3, gbc_button3);                              // 增加按钮及其约束条件
JButton button4 = new JButton("D");                                 // 创建按钮
GridBagConstraints gbc_button4 = new GridBagConstraints();          // 创建网格组布局约束条件
gbc_button4.weightx = 40.0;                                         // 第一列的分布方式为 40%
gbc_button4.fill = GridBagConstraints.BOTH;                         // 设置填充方式
gbc_button4.gridheight = 4;                                         // 占用 4 列
gbc_button4.insets = new Insets(0, 5, 0, 0);                        // 设置控件的空白
gbc_button4.gridx = 3;                                              // 起始点为第 4 列
gbc_button4.gridy = 0;                                              // 起始点为第 1 行
contentPane.add(button4, gbc_button4);                              // 增加按钮及其约束条件
JButton button5 = new JButton("E");                                 // 创建按钮
GridBagConstraints gbc_button5 = new GridBagConstraints();          // 创建网格组布局约束条件
gbc_button5.fill = GridBagConstraints.HORIZONTAL;                   // 设置填充方式
gbc_button5.gridwidth = 2;                                          // 占用 2 行
gbc_button5.insets = new Insets(5, 0, 5, 5);                        // 设置控件的空白
gbc_button5.gridx = 0;                                              // 起始点为第 1 列
gbc_button5.gridy = 1;                                              // 起始点为第 2 行
contentPane.add(button5, gbc_button5);                              // 增加按钮及其约束条件
JButton button6 = new JButton("F");                                 // 创建按钮
GridBagConstraints gbc_button6 = new GridBagConstraints();          // 创建网格组布局约束条件
gbc_button6.fill = GridBagConstraints.HORIZONTAL;                   // 设置填充方式
gbc_button6.insets = new Insets(0, 0, 5, 5);                        // 设置控件的空白
gbc_button6.gridx = 0;                                              // 起始点为第 1 列
gbc_button6.gridy = 2;                                              // 起始点为第 3 行
contentPane.add(button6, gbc_button6);                              // 增加按钮及其约束条件
JButton button7 = new JButton("G");                                 // 创建按钮
GridBagConstraints gbc_button7 = new GridBagConstraints();          // 创建网格组布局约束条件
gbc_button7.fill = GridBagConstraints.BOTH;                         // 设置填充方式
gbc_button7.gridheight = 2;                                         // 占用 2 列
gbc_button7.gridwidth = 2;                                          // 占用 2 行
gbc_button7.insets = new Insets(5, 5, 0, 5);                        // 设置控件的空白
gbc_button7.gridx = 1;                                              // 起始点为第 2 列
gbc_button7.gridy = 2;                                              // 起始点为第 3 行
contentPane.add(button7, gbc_button7);                              // 增加按钮及其约束条件
JButton button8 = new JButton("H");                                 // 创建按钮
```

```
    GridBagConstraints gbc_button8 = new GridBagConstraints();          // 创建网格组布局约束条件
    gbc_button8.fill = GridBagConstraints.HORIZONTAL;                   // 设置填充方式
    gbc_button8.insets = new Insets(5, 0, 0, 0);                        // 设置控件的空白
    gbc_button8.gridx = 0;                                              // 起始点为第 1 列
    gbc_button8.gridy = 3;                                              // 起始点为第 4 行
    contentPane.add(button8, gbc_button8);                              // 增加按钮及其约束条件
}
```

实例运行效果如图 10.15 所示。

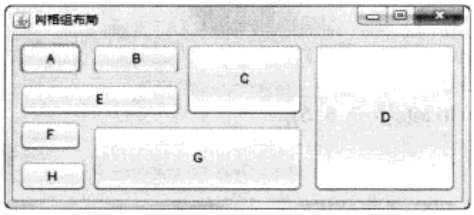


图 10.15　实例运行效果

10.3　常 用 面 板

面板也是一个 Swing 容器，它可以作为容器容纳其他控件，但它也必须添加到其他容器中才能发挥作用，如添加到窗体控件容器。Swing 中常用的面板包括 JPanel 面板以及 JScrollPane 面板。

10.3.1　JPanel 面板

JPanel 面板可以聚集一些控件进行布局，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。JPanel 控件作为容器类，除了可以添加控件之外，也可以设置布局管理器，利用它可以对面板中的控件进行布局。

例 10.07　下面通过一个简单的实例演示如何使用面板。实例的关键代码如下：**（实例位置：光盘\mr\ym\10.07）**

```
public JPanelDemo() {
    setTitle("面板使用");                                               // 设置窗体的标题
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);                     // 设置窗体退出时操作
    setBounds(100, 100, 250, 200);                                      // 设置窗体位置和大小
    contentPane = new JPanel();                                         // 创建内容面板
    contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));                 // 设置面板的边框
    setContentPane(contentPane);                                        // 应用内容面板
    contentPane.setLayout(new GridLayout(3, 3, 5, 5));                  // 使用 3 行 3 列的网格布局
    JPanel panel1 = new JPanel();                                       // 创建面板
    contentPane.add(panel1);                                            // 应用面板
    JButton button1 = new JButton("A");                                 // 创建按钮
    panel1.add(button1);                                                // 应用按钮
    JButton button2 = new JButton("B");                                 // 创建按钮
    panel1.add(button2);                                                // 应用按钮
    JButton button3 = new JButton("C");                                 // 创建按钮
    panel1.add(button3);                                                // 应用按钮
```

```
        JPanel panel2 = new JPanel();                          // 创建面板
        contentPane.add(panel2);                               // 应用面板
        JButton button4 = new JButton("D");                    // 创建按钮
        panel2.add(button4);                                   // 应用按钮
        JButton button5 = new JButton("E");                    // 创建按钮
        panel2.add(button5);                                   // 应用按钮
        JButton button6 = new JButton("F");                    // 创建按钮
        panel2.add(button6);                                   // 应用按钮
        JPanel panel3 = new JPanel();                          // 创建面板
        contentPane.add(panel3);                               // 应用面板
        JButton button7 = new JButton("G");                    // 创建按钮
        panel3.add(button7);                                   // 应用按钮
        JButton button8 = new JButton("H");                    // 创建按钮
        panel3.add(button8);                                   // 应用按钮
        JButton button9 = new JButton("I");                    // 创建按钮
        panel3.add(button9);                                   // 应用按钮
    }
```

实例运行效果如图 10.16 所示。

图 10.16　实例运行效果

📖 **说明：面板默认使用流式布局。**

10.3.2　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的内容，这时可以使用 JScrollPane 面板。JScrollPane 面板是带滚动条的面板，也是一种容器，但是常用于布置单个控件，并且不可以使用布局管理器。如果需要在 JScrollPane 面板中放置多个控件，需要将多个控件放置到 JPanel 面板上，然后将 JPanel 面板作为一个整体控件添加到 JScrollPane 控件上。

例 10.08　下面通过一个简单的实例演示如何使用滚动面板。实例的关键代码如下：**（实例位置：光盘\mr\ym\10.08）**

```
    public JScrollPaneDemo() {
        setTitle("滚动面板使用");                                  // 设置窗体的标题
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);          // 设置窗体退出时操作
        setBounds(100, 100, 250, 200);                           // 设置窗体位置和大小
        contentPane = new JPanel();                              // 创建内容面板
        contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));      // 设置面板的边框
        contentPane.setLayout(new BorderLayout(0, 0));           // 设置内容面板为边界布局
        setContentPane(contentPane);                             // 应用内容面板
        JScrollPane scrollPane = new JScrollPane();              // 创建滚动面板
        contentPane.add(scrollPane, BorderLayout.CENTER);        // 将滚动面板放置在内容面板中央
```

```
        JTextArea textArea = new JTextArea();                    // 创建文本域
        scrollPane.setViewportView(textArea);                    // 在滚动面板中显示文本域
    }
```

实例运行效果如图 10.17 所示。

图 10.17　实例运行效果

从本实例的运行结果可以看到，在窗体中创建一个带滚动条的编辑器，首先初始化一个文本域控件，在 Swing 中文本域控件为 JTextArea 类，并在初始化时设置文本域控件的大小大于窗体大小。如果读者对文本域控件的概念有些困惑，笔者将在后续章节中对其进行讲解。当创建带滚动条的面板时，将文本域控件传递给滚动面板类的构造方法，最后将带滚动条的文本域控件放置在窗体容器中即可。

10.4　标签控件与图标

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控件的目的，所以标签在 Swing 中是比较常用的控件。在本节中将探讨 Swing 标签的用法。

10.4.1　标签的使用

标签由 JLabel 类定义，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，例如，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表 10.5 所示。

表 10.5　JLabel 类构造方法说明

方 法 名	说　明
JLabel()	创建一个不带图标和文本的 JLabel 对象
JLabel(Icon icon)	创建带图标的 JLabel 对象
JLabel(Icon icon,int aligment)	创建带图标的 JLabel 对象，并设置水平对齐方式
JLabel(String text,int aligment)	创建带文本的 JLabel 对象，并设置水平对齐方式
JLabel(String text,Icon icon,int aligment)	创建带文本、图标的 JLabel 对象，并设置标签内容的水平对齐方式

10.4.2　图标的使用

Swing 中的图标可以放置在按钮、标签等控件上，用于描述控件的用途，图标可以由 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绘制。有关绘图的相关技术将在后面章节介绍。

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图标，利用 javax.swing.ImageIcon 类可以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 类的常用构造方法如表 10.6 所示。

表 10.6　ImageIcon 类构造方法说明

方　法　名	说　　明
ImageIcon()	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)方法来指定图片
ImageIcon(Image image)	直接从图片源创建图标
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
ImageIcon(URL url)	利用位于计算机网络上的图像文件 URL 地址创建图标

例 10.09　下面通过一个简单的实例演示如何创建包括图标的标签。实例的关键代码如下：**（实例位置：光盘\mr\ym\10.09）**

```
public JLabelDemo() {
    setTitle("包含图标的标签");                                          // 设置窗体的标题
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);                    // 设置窗体退出时操作
    setBounds(100, 100, 250, 100);                                     // 设置窗体位置和大小
    contentPane = new JPanel();                                        // 创建内容面板
    contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));                // 设置面板的边框
    contentPane.setLayout(new BorderLayout(0, 0));                     // 设置内容面板为边界布局
    setContentPane(contentPane);                                       // 应用内容面板
    JLabel label = new JLabel("吉林省明日科技有限公司");                  // 创建包含指定文本的标签
    label.setIcon(new ImageIcon(JLabelDemo.class.getResource("/images/icon.jpg")));  // 为标签增加图标
    contentPane.add(label, BorderLayout.CENTER);                       // 在面板上显示标签
}
```

实例运行效果如图 10.18 所示。

图 10.18　实例运行效果

10.5　按 钮 控 件

按钮在 Swing 控件中是较为常见的控件，它用于触发特定动作。Swing 中提供很多种按钮，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，在本节中将着重讲解这些按钮的使用。

10.5.1　普通按钮控件

JButton 是 Swing 的按钮控件类，它和 JLabel 控件都是常用的控件类。可以使用 JButton 创建普通样式的按钮，如“确定”、“取消”按钮等，JButton 类的常用构造方法如表 10.7 所示。

表 10.7　JButton 类构造方法说明

方 法 名	说　明
JButton()	创建一个按钮，它没有指定按钮文本，也没有按钮图标
JButton(Icon icon)	创建一个带图标的按钮
JButton(String text)	创建一个带文本的按钮
JButton(String text, Icon icon)	创建一个带初始文本和图标的按钮

例 10.10　下面通过一个简单的实例演示如何创建包括图标的按钮。实例的关键代码如下：**（实例位置：光盘\mr\ym\10.10）**

```
public JButtonDemo() {
    setTitle("按钮使用");                                          // 设置窗体的标题
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);                // 设置窗体退出时操作
    setBounds(100, 100, 250, 100);                                 // 设置窗体位置和大小
    contentPane = new JPanel();                                    // 创建内容面板
    contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));            // 设置面板的边框
    setContentPane(contentPane);                                   // 应用内容面板
    contentPane.setLayout(new FlowLayout(FlowLayout.CENTER, 5, 5)); // 设置内容面板为流式布局
    JButton modifyButton = new JButton("修改");                    // 创建按钮
    modifyButton.setIcon(new ImageIcon(JButtonDemo.class.getResource("/images/modify.gif")));
                                                                   // 为按钮增加图标
    contentPane.add(modifyButton);                                 // 在面板上增加按钮
    JButton deleteButton = new JButton("删除");                    // 创建按钮
    deleteButton.setIcon(new ImageIcon(JButtonDemo.class.getResource("/images/delete.gif")));
                                                                   // 为按钮增加图标
    contentPane.add(deleteButton);                                 // 在面板上增加按钮
}
```

实例运行效果如图 10.19 所示。

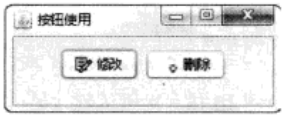


图 10.19　实例运行效果

10.5.2　单选按钮控件

JRadioButton 也是一种按钮，它用于创建界面中的单选按钮。单选按钮有选中与未选中状态，另外它也有和 JButton 类似的构造方法，如表 10.8 所示。

表 10.8　JRadioButton 类构造方法说明

方 法 名	说　明
JRadioButton ()	创建一个单选按钮，它没有指定按钮文本，也没有按钮图标
JRadioButton (Icon icon)	创建一个带图标的单选按钮
JRadioButton (String text)	创建一个带文本的单选按钮
JRadioButton (String text, Icon icon)	创建一个带初始文本和图标的单选按钮
JRadioButton (String text, boolean selected)	在创建单选按钮时指定文本与选中状态

单选按钮的组合使用，需要使用 ButtonGroup 类将按钮分组，该类使用默认构造方法创建，通过 add() 方法将按钮添加到按钮组中。组合起来的多个单选按钮添加到按钮组以后，同一时间只能有一个单选按钮被选中，即选择一个单选按钮之后，另外一个已经处于被选择状态的单选按钮将取消选择状态。如果没有指定按钮组，那么多个单选按钮之间互不影响，可以同时选中多个单选按钮，这和复选框就没有多大区别了。

例 10.11　下面通过一个简单的实例演示如何使用单选按钮。实例的关键代码如下：**（实例位置：光盘\mr\ym\10.11）**

```
public JRadioButtonDemo() {
    setTitle("单选按钮使用");                                          // 设置窗体的标题
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);                    // 设置窗体退出时操作
    setBounds(100, 100, 250, 100);                                     // 设置窗体位置和大小
    contentPane = new JPanel();                                        // 创建内容面板
    contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));                // 设置面板的边框
    setContentPane(contentPane);                                       // 应用内容面板
    contentPane.setLayout(new FlowLayout(FlowLayout.CENTER, 5, 5));    // 设置内容面板为流式布局
    JRadioButton radioButton1 = new JRadioButton("Java");              // 创建单选按钮
    contentPane.add(radioButton1);                                     // 应用单选按钮
    JRadioButton radioButton2 = new JRadioButton("PHP");               // 创建单选按钮
    contentPane.add(radioButton2);                                     // 应用单选按钮
    JRadioButton radioButton3 = new JRadioButton("C++");               // 创建单选按钮
    contentPane.add(radioButton3);                                     // 应用单选按钮
    ButtonGroup group = new ButtonGroup();                             // 创建单选按钮组
    group.add(radioButton1);
                                                                       // 将 radioButton1 增加到单选按钮组中
    group.add(radioButton2);
                                                                       // 将 radioButton2 增加到单选按钮组中
    group.add(radioButton3);
                                                                       // 将 radioButton3 增加到单选按钮组中
}
```

实例运行效果如图 10.20 所示。

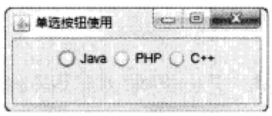


图 10.20　实例运行效果

10.5.3　复选框控件

复选框在 Swing 控件中使用也非常广泛，每一个复选框都提供“选中”与“未选中”两种状态，它与单选按钮唯一不同的是选择状态的图标一个是圆形，一个是方形，其他的使用方法完全一样。另外，JCheckBox 控件不需要设置按钮组，否则同一时间只能选中一个复选框，那就成单选按钮了。

例 10.12　下面通过一个简单的实例演示如何使用复选框。实例的关键代码如下：**（实例位置：光盘\mr\ym\10.12）**

```
public JCheckBoxDemo() {
    setTitle("复选框使用");                                            // 设置窗体的标题
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);                    // 设置窗体退出时操作
    setBounds(100, 100, 250, 100);                                     // 设置窗体位置和大小
```

```
        contentPane = new JPanel();                                          // 创建内容面板
        contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));                  // 设置面板的边框
        setContentPane(contentPane);                                         // 应用内容面板
        contentPane.setLayout(new FlowLayout(FlowLayout.CENTER, 5, 5));      // 设置内容面板为流式布局
        JCheckBox checkBox1 = new JCheckBox("Java");                         // 创建复选框
        contentPane.add(checkBox1);                                          // 应用复选框
        JCheckBox checkBox2 = new JCheckBox("PHP");                          // 创建复选框
        contentPane.add(checkBox2);                                          // 应用复选框
        JCheckBox checkBox3 = new JCheckBox("C++");                          // 创建复选框
        contentPane.add(checkBox3);                                          // 应用复选框
    }
```

实例运行效果如图 10.21 所示。

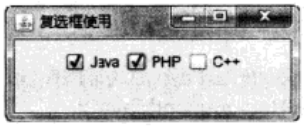


图 10.21　实例运行效果

10.6　文 本 控 件

一个应用程序所有输入功能的界面几乎都是由文本控件接收用户输入的。例如，姓名、住址、电话号码等信息需要使用文本框接收用户输入，用户密码当然要使用密码框控件接收，对于简历中的自我描述、大段或多行的文本录入工作都需要使用文本域控件。本节将对其进行介绍。

10.6.1　文本框控件

文本框用来显示或编辑一个单行文本，在 Swing 中通过 javax.swing.JTextField 类创建文本框控件对象，常用的构造方法如表 10.9 所示。

表 10.9　JTextField 类构造方法说明

方 法 名	说　明
JTextField()	构造一个默认的文本框控件
JTextField(String text)	构造一个包含指定字符串内容的文本框控件
JTextField(int columns)	构造一个具有指定列数的空的文本框控件
JTextField(String text, int columns)	构造一个具有指定列数并包含指定文本内容的文本框控件

在上述构造方法中可以看出，定义 JTextField 控件很简单，可以在初始化文本框时设置文本框的默认文字、文本框的首选列数等。其中首选列数 columns 用于定制控件在布局管理器中的首选大小，如果将文本框使用绝对布局或者布局在限制控件大小的布局管理器中，如 GridLayout，Columns 将不起作用。

例 10.13　下面通过一个简单的实例演示如何使用文本框。实例的关键代码如下：**（实例位置：光盘\mr\ym\10.13）**

```
    public JTextFieldDemo() {
        setTitle("文本框使用");                                              // 设置窗体的标题
```

```
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);          // 设置窗体退出时操作
        setBounds(100, 100, 250, 200);                           // 设置窗体位置和大小
        contentPane = new JPanel();                              // 创建内容面板
        contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));      // 设置面板的边框
        setContentPane(contentPane);                             // 应用内容面板
        contentPane.setLayout(new GridLayout(3, 1, 5, 5));       // 使用 3 行 1 列的网格布局
        JPanel panel1 = new JPanel();                            // 创建面板
        contentPane.add(panel1);                                 // 应用面板
        JLabel label1 = new JLabel("姓名：");                    // 创建标签
        panel1.add(label1);                                      // 增加标签
        textField1 = new JTextField();                           // 创建文本框
        panel1.add(textField1);                                  // 增加文本框
        textField1.setColumns(10);                               // 将文本框列数设置为 10
        JPanel panel2 = new JPanel();                            // 创建面板
        contentPane.add(panel2);                                 // 应用面板
        JLabel label2 = new JLabel("年龄：");                    // 创建标签
        panel2.add(label2);                                      // 增加标签
        textField2 = new JTextField();                           // 创建文本框
        panel2.add(textField2);                                  // 增加文本框
        textField2.setColumns(10);                               // 将文本框列数设置为 10
        JPanel panel3 = new JPanel();                            // 创建面板
        contentPane.add(panel3);                                 // 应用面板
        JLabel label3 = new JLabel("住址：");                    // 创建标签
        panel3.add(label3);                                      // 应用标签
        textField3 = new JTextField();                           // 创建文本框
        panel3.add(textField3);                                  // 增加文本框
        textField3.setColumns(10);                               // 将文本框列数设置为 10
}
```

实例运行效果如图 10.22 所示。

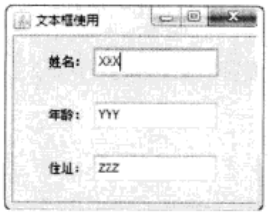


图 10.22　实例运行效果

10.6.2　密码框控件

密码框与文本框的定义与用法基本相同，唯一不同的是密码框控件将用户输入的字符串以某种字符显示，如“*”字符，这样可以隐藏用户输入的密码，防止被其他人看见。密码框是通过 javax.swing.JPasswordField 类来创建的，JPasswordField 类的构造方法与 JTextField 类的构造方法非常相似，该类常用的构造方法如表 10.10 所示。

表 10.10　JPasswordField 类构造方法说明

方 法 名	说　明
JPasswordField ()	构造一个默认的密码框控件
JPasswordField (String text)	构造一个包含指定字符串内容的密码框控件
JPasswordField (int columns)	构造一个具有指定列数的空的密码框控件
JPasswordField (String text, int columns)	构造一个具有指定列数并包含指定文本内容的密码框控件

例 10.14　下面通过一个简单的实例演示如何使用密码框。实例的关键代码如下：**（实例位置：光盘\mr\ym\10.14）**

```
public JPasswordFieldDemo() {
    setTitle("密码框使用");                                    // 设置窗体的标题
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);          // 设置窗体退出时操作
    setBounds(100, 100, 250, 100);                           // 设置窗体位置和大小
    contentPane = new JPanel();                              // 创建内容面板
    contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));      // 设置面板的边框
    contentPane.setLayout(new BorderLayout(0, 0));           // 设置内容面板为边界布局
    setContentPane(contentPane);                             // 应用内容面板
    JLabel label = new JLabel("密码：");                      // 创建标签
    contentPane.add(label, BorderLayout.WEST);               // 将标签增加到面板西部
    passwordField = new JPasswordField();                    // 创建密码框
    contentPane.add(passwordField, BorderLayout.CENTER);     // 将密码框增加到面板中央
}
```

实例运行效果如图 10.23 所示。

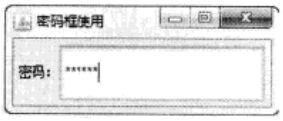


图 10.23　实例运行效果

技巧：使用 setEchoChar()方法可以改变密码框的回显字符。

10.6.3　文本域控件

文本域控件的创建也非常简单，它在程序中用于接受用户的多行文字输入，如记事本的文本编辑部分就是一个文本域控件，如图 10.24 所示。

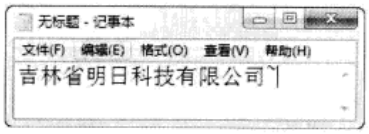


图 10.24　记事本文件

Swing 中的文本区域控件是 JTextArea 类的实例对象，该类常用的构造方法如表 10.11 所示。

表 10.11　JTextArea 类构造方法说明

方 法 名	说　明
JTextArea()	构造一个新的文本域控件，不包含任何文本内容
JTextArea(String text)	构建一个包含指定文本内容的文本域控件
JTextArea(int rows,int columns)	构建一个指定列数并包含指定文本内容的文本域控件

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行数与列数。

例 10.15　下面通过一个简单的实例演示如何使用文本域。实例的关键代码如下：**（实例位置：光盘\mr\ym\10.15）**

```
public JTextAreaDemo() {
    setTitle("文本域使用");                                          // 设置窗体的标题
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);                  // 设置窗体退出时操作
    setBounds(100, 100, 250, 200);                                   // 设置窗体位置和大小
    contentPane = new JPanel();                                      // 创建内容面板
    contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));              // 设置面板的边框
    contentPane.setLayout(new BorderLayout(0, 0));                   // 设置内容面板为边界布局
    setContentPane(contentPane);                                     // 应用内容面板
    JPanel panel1 = new JPanel();                                    // 创建面板
    FlowLayout fl_panel1 = (FlowLayout) panel1.getLayout();          // 获得面板的布局管理器
    fl_panel1.setAlignment(FlowLayout.LEFT);                         // 将布局管理器设置成左侧排列
    contentPane.add(panel1, BorderLayout.NORTH);                     // 应用面板
    JLabel label = new JLabel("请输入您的评语：");                    // 创建标签
    panel1.add(label);                                               // 应用标签
    JPanel panel2 = new JPanel();                                    // 创建面板
    contentPane.add(panel2, BorderLayout.SOUTH);                     // 应用面板
    JButton button1 = new JButton("提交");                           // 创建按钮
    panel2.add(button1);                                             // 应用按钮
    JButton button2 = new JButton("关闭");                           // 创建按钮
    panel2.add(button2);                                             // 应用按钮
    JScrollPane scrollPane = new JScrollPane();                      // 创建滚动面板
    contentPane.add(scrollPane, BorderLayout.CENTER);                // 应用面板
    JTextArea textArea = new JTextArea();                            // 创建文本域
    scrollPane.setViewportView(textArea);                            // 应用文本域
}
```

实例运行效果如图 10.25 所示。

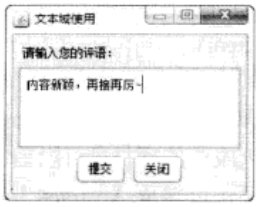


图 10.25　实例运行效果

在 JTextArea 类中存在一个 setLineWrap()方法，该方法用于设置文本域是否可以自动换行，如果将该方法的参数设置为 true，文本域将自动换行，相反则不自动换行，文本内容会溢出程序界面，也可以使用滚动

面板管理文本域，在文本溢出程序界面，即文本长度超出文本域控件长度时，会出现滚动条。

10.7 列 表 控 件

Swing 中提供两种列表控件，分别为下拉列表框 JComboBox 与列表框 JList，下拉列表框与列表框都带有一系列项目的控件，同时用户可以选择需要的项目。列表框比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，当用户使用时其中的项目才会显现出来。在本节中将详细讲解列表框与下拉列表框的应用。

10.7.1 下拉列表框控件

1. JComboBox 类

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图标，当用户单击该图标时，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 类来创建，它是 javax.swing.JComponent 类的子类，常用构造方法如表 10.12 所示。

表 10.12 JComboBox 类构造方法说明

方 法 名	说 明
JComboBox()	创建具有默认数据模型的空的 JComboBox
JComboBox(ComboBoxModel aModel)	创建一个 JComboBox，其列表项目取自指定的 ComboBoxModel
JComboBox(Object[] items)	创建包含指定数组中元素的 JComboBox
JComboBox(Vector<?> items)	创建包含指定 Vector 向量对象中元素的 JComboBox

在初始化下拉列表框时，可以选择同时指定下拉列表的项目内容，也可以在程序中使用其他方法设置下拉列表的内容，下拉列表中的内容可以被封装在 ComboBoxModel 类型、数组或 Vector 类的实例对象中。

2. JComboBox 模型

在开发程序中，一般将下拉列表中的项目封装为 ComboBoxModel 类型的情况比较多，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，但是必须实现如下两个方法：

- ☑ public void setSelectedItem(Object item)。
- ☑ public Object getSelectedItem()。

其中 setSelectedItem()方法是设置下拉列表的选择项，getSelectedItem()方法用于返回下拉列表中选择的项，有了这两个方法，就可以轻松对下拉列表中的项目进行操作。

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 3 个操作下拉列表的重要方法。

- ☑ getSize()：返回列表的长度。
- ☑ getElementAt(int index)：返回指定索引处的值。
- ☑ addItem(Object anObject)：为下拉组合框添加选项，anObject 可以是字符串。

例 10.16　下面通过一个简单的实例演示如何使用下拉列表框。实例的关键代码如下：**（实例位置：光盘\mr\ym\10.16）**

```
public JComboBoxDemo() {
    setTitle("下拉列表框使用");                                          // 设置窗体的标题
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);                      // 设置窗体退出时操作
    setBounds(100, 100, 250, 100);                                       // 设置窗体位置和大小
    contentPane = new JPanel();                                          // 创建内容面板
    contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));                  // 设置面板的边框
    setContentPane(contentPane);                                         // 应用内容面板
    contentPane.setLayout(new FlowLayout(FlowLayout.CENTER, 5, 5));      // 设置布局为流式布局
    JLabel label = new JLabel("证件类型：");                              // 创建标签
    contentPane.add(label);                                              // 应用标签
    JComboBox comboBox = new JComboBox();                                // 创建组合框
    comboBox.addItem("身份证");                                          // 为组合框增加选项
    comboBox.addItem("驾驶证");                                          // 为组合框增加选项
    comboBox.addItem("军官证");                                          // 为组合框增加选项
    contentPane.add(comboBox);                                           // 应用组合框
}
```

实例运行效果如图 10.26 所示。

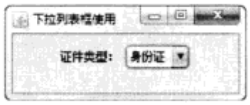


图 10.26　实例运行效果

10.7.2　列表框控件

列表框与下拉列表框的区别不仅仅表现在外观上，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，如果需要列表框具有滚动效果，可以将列表框放入滚动面板中。当用户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，可以连续选择两个选项之间的所有项目，也可以按住 Ctrl 键选择零散的个别项目，如图 10.27 所示。

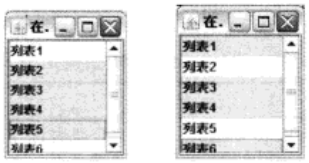


图 10.27　Shift 多选效果（左）与 Ctrl 单选效果（右）

Swing 中使用 JList 类对象来表示列表，表 10.13 列举了几个常用的构造方法。

表 10.13　JList 类构造方法说明

方　法　名	说　　明
JList()	构造一个具有空的、只读模型的 JList
JList(ListModel dataModel)	根据指定的非 null 模型对象构造一个显示元素的 JList
JList(Object[] listData)	构造一个 JList，使其显示指定数组中的元素
JList(Vector<?> listData)	构造一个 JList，使其显示指定 Vector 向量对象中的元素

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()方法对列表框元素进行设置，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 对象和 ListModel 模型。

例 10.17 下面通过一个简单的实例演示如何使用列表框。实例的关键代码如下：**（实例位置：光盘\mr\ym\10.17）**

```
public JListDemo() {
    setTitle("列表框使用");                                          // 设置窗体的标题
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);                  // 设置窗体退出时操作
    setBounds(100, 100, 250, 130);                                   // 设置窗体位置和大小
    contentPane = new JPanel();                                      // 创建内容面板
    contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));              // 设置面板的边框
    setContentPane(contentPane);                                     // 应用内容面板
    contentPane.setLayout(new FlowLayout(FlowLayout.CENTER, 5, 5));  // 设置布局为流式布局
    JLabel label = new JLabel("证件类型：");                          // 创建标签
    contentPane.add(label);                                          // 应用标签
    JList list = new JList(new String[] { "身份证", "驾驶证", "军官证" });  // 创建列表
    contentPane.add(list);                                           // 应用列表
}
```

实例运行效果如图 10.28 所示。

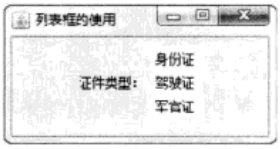


图 10.28 实例运行效果

10.8 进 度 条

利用 JProgressBar 类可以实现一个进度条，它只不过是一个矩形控件，通过填充它的部分或全部来指示一个任务的执行情况，创建进度条的常用构造方法如表 10.14 所示。

表 10.14 JProgressBar 类构造方法说明

方 法 名	说 明
JProgressBar()	创建一个显示边框但不带进度字符串，最小值 0、最大值 100 的水平进度条
JProgressBar(int orient)	创建具有指定 SwingConstants.VERTICAL 垂直或 SwingConstants.HORIZONTAL 水平方向的进度条
JProgressBar(int min, int max)	创建具有指定最小值和最大值的水平进度条
JProgressBar(int orient, int min, int max)	创建使用指定方向、最小值和最大值的进度条

例 10.18 下面通过一个简单的实例演示如何使用进度条。实例的关键代码如下：**（实例位置：光盘\mr\ym\10.18）**

```
public JProgressBarDemo() {
    setTitle("进度条使用");                                          // 设置窗体的标题
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);                  // 设置窗体退出时操作
    setBounds(100, 100, 250, 100);                                   // 设置窗体位置和大小
```

```
        contentPane = new JPanel();                                          // 创建内容面板
        contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));                  // 设置面板的边框
        setContentPane(contentPane);                                         // 应用内容面板
        contentPane.setLayout(new FlowLayout(FlowLayout.CENTER, 5, 5));      // 设置布局为流式布局
        final JProgressBar progressBar = new JProgressBar();                 // 创建进度条
        progressBar.setStringPainted(true);                                  // 让进度条上能够显示文本信息
        new Thread() {
            public void run() {
                for (int i = 0; i <= 100; i++) {                             // 通过循环更新任务完成比例
                    try {
                        Thread.sleep(100);                                   // 让当前线程休眠 0.1 秒
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                    progressBar.setValue(i);                                 // 设置进度条数值
                }
                progressBar.setString("升级完成！");                          // 设置提示信息
            };
        }.start();                                                           // 启动新线程
        contentPane.add(progressBar);                                        // 应用进度条
    }
```

实例运行效果如图 10.29 所示。

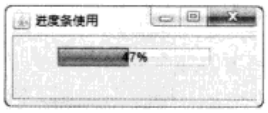


图 10.29　实例运行效果

技巧：为了让进度条有更好的响应效果，可以将其放置在一个新线程中运行。

10.9　照猫画虎——基本功训练

10.9.1　基本功训练 1——设置窗体背景颜色为淡蓝色

视频讲解：光盘\mr\lx\10\设置窗体背景颜色为淡蓝色.exe

实例位置：光盘\mr\ym\10.19\src

Swing 中窗体默认是白色的，虽然可以通过修改外观将背景色改成灰色（使用 NimbusLookAndFeel），还可以通过 setBackground()方法使窗体的颜色更加美观。本实例将窗体的背景色设置成淡蓝色。

实现过程如下：

（1）在 Eclipse 中选择“文件”/“新建”/“其他”命令，在弹出的对话框中选择 Swing Designer 文件夹下的 JFrame 窗体（如图 10.30 所示），单击“下一步”按钮，将该文件命名为 BackgroundColorDemo。

（2）打开源代码，在构造方法中增加设置面板背景色的语句并设置窗体的标题和布局，代码如下：

```
public BackgroundColorDemo() {
    setTitle("设置窗体背景为淡蓝色");                                  // 设置窗体的标题
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);                  // 设置窗体退出时操作
    setBounds(100, 100, 300, 100);                                   // 设置窗体位置和大小
```

```
        contentPane = new JPanel();                                    // 创建内容面板
        contentPane.setBackground(new Color(102, 204, 255));           // 设置窗体的背景色
        contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));            // 设置面板的边框
        contentPane.setLayout(new BorderLayout(0, 0));                 // 使用边界布局
        setContentPane(contentPane);                                   // 应用内容面板
    }
```

实例运行效果如图 10.31 所示。

图 10.30　文件类型选择向导

图 10.31　实例运行效果

你问我答：setBackground()方法的作用是什么？

该方法可以用来给面板设置新的颜色，该方法的声明如下：

```
public void setBackground(Color bg)
```

参数说明：bg 是要设置的颜色对象。

照猫画虎：使用 WindowBuilder 提供的控件属性修改工具将窗体颜色修改成粉色。提示：选择需要修改的控件之后，在左下角会有该控件的属性选择工具。（20 分）（实例位置：光盘\mr\ym\10.19\zmhh）

10.9.2　基本功训练 2——修改密码框的回显字符

视频讲解：光盘\mr\lx\10\修改密码框的回显字符.exe

实例位置：光盘\mr\ym\10.20\src

Swing 中密码框默认的回显字符是“*”，可以根据个人的需求对其进行修改。本实例将其修改成“$”，关键代码如下：

```
passwordField2.setEchoChar('$');
```

实例的运行效果如图 10.32 所示。

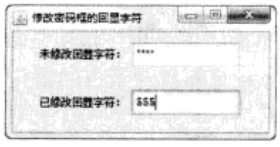


图 10.32　实例运行效果

照猫画虎：当焦点离开密码框时显示用户输入的密码，回显字符使用“·”。（20 分）（实例位置：光盘\mr\ym\10.20\zmhh）

10.9.3　基本功训练 3——查看图标的信息

视频讲解：光盘\mr\lx\10\查看图标的信息.exe

实例位置：光盘\mr\ym\10.21\src

对于标签控件而言，可以用来同时显示图标和文本信息。本实例将在标签的左侧显示图片，右侧显示该图片的描述信息。对于显示图片，可以使用 setIcon()方法完成。关键代码如下：

```
public ImageInfoDemo() {
    setTitle("查看图标信息");                                          // 设置窗体的标题
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);                    // 设置窗体退出时操作
    setBounds(100, 100, 300, 150);                                     // 设置窗体位置和大小
    contentPane = new JPanel();                                        // 创建内容面板
    contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));                // 设置面板的边框
    setContentPane(contentPane);                                       // 应用内容面板
    contentPane.setLayout(new FlowLayout(FlowLayout.CENTER, 5, 5));    // 设置内容面板为流式布局
    JLabel label = new JLabel("");                                     // 创建标签
    ImageIcon icon = new ImageIcon("src/images/1.png");                // 创建图标
    label.setIcon(icon);                                               // 为标签指定图标
    StringBuilder sb = new StringBuilder();                            // 创建 StringBuilder 对象保存图标信息
    sb.append("图标宽度：" + icon.getIconWidth() + ", ");               // 保存图标宽度
    sb.append("图标高度：" + icon.getIconHeight());                     // 保存图标高度
    label.setText(sb.toString());                                      // 为标签指定文本
    contentPane.add(label);                                            // 应用标签
}
```

实例的运行效果如图 10.33 所示。

图 10.33　实例运行效果

照猫画虎：在标签中使用上下结构显示图标及其描述。（20 分）（实例位置：光盘\mr\ym\10.21\zmhh）

10.9.4　基本功训练 4——修改列表项选择模式

视频讲解：光盘\mr\lx\10\修改列表项选择模式.exe

实例位置：光盘\mr\ym\10.22\src

在默认情况下，列表项支持单选和多选，可以通过修改 setSelectionMode()方法的参数限制用户的选择。本实例将限制用户只能使用单选，关键代码如下：

```
public SelectionModeDemo() {
    setTitle("单项选择列表");                                          // 设置窗体的标题
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);                    // 设置窗体退出时操作
```

```
        setBounds(100, 100, 300, 200);                                  // 设置窗体位置和大小
        contentPane = new JPanel();                                     // 创建内容面板
        contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));             // 设置面板的边框
        contentPane.setLayout(new BorderLayout(0, 0));                  // 设置内容面板为边界布局
        setContentPane(contentPane);                                    // 应用内容面板
        JScrollPane scrollPane = new JScrollPane();                     // 创建滚动面板
        contentPane.add(scrollPane, BorderLayout.CENTER);               // 将面板增加到边界布局中央
        JList list = new JList();                                       // 创建列表
        list.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);     // 设置列表的选择模式为单选
        scrollPane.setViewportView(list);                               // 在滚动面板中显示列表
        String[] listData = new String[12];                             // 创建一个含有 12 个元素的数组
        for (int i = 0; i < listData.length; i++) {
            listData[i] = "Java 编程词典真好用~";                        // 为数组中各个元素赋值
        }
        list.setListData(listData);                                     // 为列表设置数据
    }
```

实例的运行效果如图 10.34 所示。

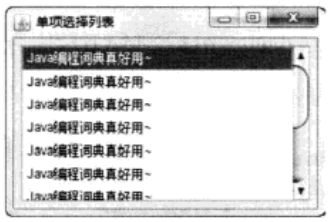


图 10.34　实例运行效果

你问我答：ListSelectionModel 接口的作用是什么？

ListSelectionModel 接口定义了列表支持的选择模式，其详细说明如表 10.15 所示。

表 10.15　列表选择模式常量

常　量　名	说　　明
MULTIPLE_INTERVAL_SELECTION	一次选择一个或多个连续的索引范围
SINGLE_INTERVAL_SELECTION	一次选择一个连续的索引范围
SINGLE_SELECTION	一次选择一个列表索引

照猫画虎：将列表的选择模式设置成一次选择一个或多个连续的索引范围。（20 分）（实例位置：光盘\mr\ym\10.22\zmhh）

10.9.5　基本功训练 5——监听列表项单击事件

视频讲解：光盘\mr\lx\10\监听列表项单击事件.exe

实例位置：光盘\mr\ym\10.23\src

列表控件提供了很多选项供用户选择，通常需要根据用户选择的列表项完成不同的操作。本实例实现了监听列表项的功能。

实现过程如下：

（1）在 Eclipse 中创建 JFrame 窗体，设置其名称为 SelectEventDemo。

（2）在窗体中增加 JList 控件并为其指定数据。

（3）为 JList 控件增加选择事件监听，它是通过 do_list_valueChanged()方法完成的，该方法的代码如下：

```
protected void do_list_valueChanged(ListSelectionEvent e) {
    label.setText("感谢您购买：" + list.getSelectedValue());          // 将用户选择的列表项显示在标签中
}
```

实例的运行效果如图 10.35 所示。

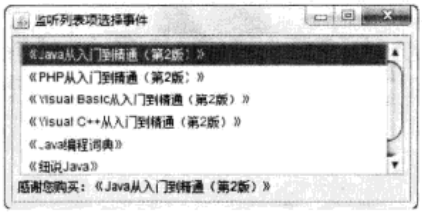


图 10.35　实例运行效果

你问我答：getSelectedValue()方法的作用是什么？

该方法返回最小的选择单元索引的值；只选择了列表中单个选项时，返回所选值。选择了多项时，返回最小的选择索引的值。如果什么也没有选择，则返回 null。

照猫画虎：使用对话框弹出用户选择的列表项。（20 分）（实例位置：光盘\mr\ym\10.23\zmhh）

照猫画虎栏目分数统计：

照猫画虎题目	1	2	3	4	5	总分数	
分数							

10.10　情景应用——拓展与实践

10.10.1　情景应用 1——设置窗体的背景图片

视频讲解：光盘\mr\lx\10\设置窗体的背景图片.exe

实例位置：光盘\mr\ym\10.24\src

除了为窗体设置其他的背景色，还可以通过设置背景图片来美化窗体，这样效果会更好。本实例将演示如何为窗体设置背景图片，实现的思路是在窗体上增加一个面板，然后重写面板的 paint()方法，使用 Graphics 类的 drawImage()方法在面板上绘制图片，实例的运行效果如图 10.36 所示。

实现过程如下：

（1）在项目中新建窗体类 BackgroundDemo，在窗体中添加面板控件。

（2）重写面板控件的 paint()方法来为窗体设置背景图片。首先使用图标来获得要绘制的图片，然后使用 drawImage()方法完成绘制，关键代码如下：

```
contentPane = new JPanel() {
    private static final long serialVersionUID = -1588458291133087637L;
    @Override
```

```
    public void paint(Graphics g) {
        ImageIcon icon = new ImageIcon("src/images/background.jpg");      // 获得图标
        Image image = icon.getImage();                                    // 获得图片
        g.drawImage(image, 0, 0, null);                                   // 在窗体上绘制图片
    }
};
```

图 10.36　实例运行效果

你问我答：drawImage()方法的作用是什么？

drawImage()方法是在 Graphics 类中定义的，该方法有很多重载形式。本实例使用了最简单的绘制图像的功能，该方法的方法声明如下：

```
public abstract boolean drawImage(Image img,int x,int y,ImageObserver observer)
```

方法中的参数说明如表 10.16 所示。

表 10.16　drawImage()方法参数说明

参 数 名	说　明
img	需要绘制的图像
x	图像顶点的 x 坐标
y	图像顶点的 y 坐标
observer	图像的观察者

DIY：在设置好背景图片的窗体中增加按钮控件，查看运行效果。（20 分）（实例位置：光盘\mr\ym\10.24\diy）

10.10.2　情景应用 2——震动效果的提示信息

视频讲解：光盘\mr\lx\10\震动效果的提示信息.exe

实例位置：光盘\mr\ym\10.25\src

QQ 是目前主流的网络聊天工具，它提供了发送抖动窗体的功能，该功能可以提示对方。本实例将模仿这个功能对 Swing 中的窗体实现震动效果，其思想是在一定的时间内不断地改变窗体左上角的坐标。如果时间间隔很短，则窗体看起来就像在震动，实例的运行效果如图 10.37 所示。

图 10.37　实例运行效果

实现过程如下：

（1）在项目中新建窗体类 ShockDemo。

（2）使用 do_this_windowActivated()方法来完成对窗体激活事件的监听，它新建了一个线程。在重写 run()方法时，让窗体每隔 0.05 秒就随机改变一下左上角的坐标，并循环执行 20 次，关键代码如下：

```
protected void do_this_windowActivated(WindowEvent e) {
    new Thread() {
        @Override
        public void run() {
            for (int i = 0; i < 20; i++) {                          // 使用 for 循环让窗体震动 20 次
                int newPoint = (int) (100 + Math.pow(-1, i) * 10);  // 根据循环次数计算新点的位置
                setLocation(newPoint, newPoint);                    // 设置窗体的显示位置
                try {
                    Thread.sleep(50);                               // 线程休眠 0.05 秒来实现动态效果
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        }
    }.start();                                                      // 启动新线程
}
```

DIY：修改窗体的震动方式，让顶点做圆周运动。（20 分）（实例位置：光盘\mr\ym\10.25\diy）

10.10.3　情景应用 3——实现计算器程序界面

视频讲解：光盘\mr\lx\10\实现计算器程序界面.exe

实例位置：光盘\mr\ym\10.26\src

Swing 对于控件的摆放提供了布局管理器来简化开发，常见的布局管理器有边界布局、网格布局等。本实例使用两种布局管理器来完成常见的计算器界面的设计。计算器界面可以分成两部分，即文本框和键盘区。对于键盘区，使用网格布局是非常方便的，实例的运行效果如图 10.38 所示：

图 10.38　实例运行效果

实现过程如下：

（1）在项目中新建窗体类 CalculatorDemo。

（2）本实例的界面可以分成两部分：对于文本框而言，将其放置在边界布局的北部；对于按钮而言，使用的是网格布局。实例的关键代码如下：

```
public CalculatorDemo() {
    setTitle("模拟计算器");                                          // 设置窗体的标题
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);                 // 设置窗体退出时操作
    setBounds(100, 100, 250, 200);                                  // 设置窗体位置和大小
    contentPane = new JPanel();                                     // 创建内容面板
    contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));             // 设置面板的边框
    contentPane.setLayout(new BorderLayout(0, 0));                  // 设置内容面板为边界布局
    setContentPane(contentPane);                                    // 应用内容面板
    JPanel panel1 = new JPanel();                                   // 新建面板用于保存文本框
    contentPane.add(panel1, BorderLayout.NORTH);                    // 将面板放置在边界布局的北部
    textField = new JTextField();                                   // 新建文本框
```

```
        textField.setHorizontalAlignment(SwingConstants.RIGHT);          // 文本框中的文本使用右对齐
        panel1.add(textField);                                           // 将文本框增加到面板中
        textField.setColumns(18);                                        // 设置文本框的列数是 18
        JPanel panel2 = new JPanel();                                    // 新建面板用于保存按钮
        contentPane.add(panel2, BorderLayout.CENTER);                    // 将面板放置在边界布局的中央
        panel2.setLayout(new GridLayout(4, 4, 5, 5));                    // 面板使用网格 4×4 布局
        JButton button01 = new JButton("7");                             // 新建按钮
        panel2.add(button01);                                            // 应用按钮
        JButton button02 = new JButton("8");                             // 新建按钮
        panel2.add(button02);                                            // 应用按钮
        JButton button03 = new JButton("9");                             // 新建按钮
        panel2.add(button03);                                            // 应用按钮
        JButton button04 = new JButton("+");                             // 新建按钮
        panel2.add(button04);                                            // 应用按钮
        JButton button05 = new JButton("4");                             // 新建按钮
        panel2.add(button05);                                            // 应用按钮
        JButton button06 = new JButton("5");                             // 新建按钮
        panel2.add(button06);                                            // 应用按钮
        JButton button07 = new JButton("6");                             // 新建按钮
        panel2.add(button07);                                            // 应用按钮
        JButton button08 = new JButton("-");                             // 新建按钮
        panel2.add(button08);                                            // 应用按钮
        JButton button09 = new JButton("3");                             // 新建按钮
        panel2.add(button09);                                            // 应用按钮
        JButton button10 = new JButton("2");                             // 新建按钮
        panel2.add(button10);                                            // 应用按钮
        JButton button11 = new JButton("1");                             // 新建按钮
        panel2.add(button11);                                            // 应用按钮
        JButton button12 = new JButton("*");                             // 新建按钮
        panel2.add(button12);                                            // 应用按钮
        JButton button13 = new JButton("0");                             // 新建按钮
        panel2.add(button13);                                            // 应用按钮
        JButton button14 = new JButton(".");                             // 新建按钮
        panel2.add(button14);                                            // 应用按钮
        JButton button15 = new JButton("=");                             // 新建按钮
        panel2.add(button15);                                            // 应用按钮
        JButton button16 = new JButton("/");                             // 新建按钮
        panel2.add(button16);                                            // 应用按钮
    }
```

DIY：实现手机界面。（20 分）（实例位置：光盘\mr\ym\10.26\diy）

10.10.4 情景应用 4——实现用户注册界面

视频讲解：光盘\mr\lx\10\实现用户注册界面.exe

实例位置：光盘\mr\ym\10.27\src

对于一个最简单的用户注册系统，仅需要用户提供用户名和密码。对于密码，还需要用户输入两次防止用户误操作。如果两次的密码相同并且用户名不存在，则可以完成新用户的注册。本实例完成这个简单的注册界面，实例的运行效果如图 10.39 所示。

图 10.39　实例运行效果

实现过程如下：

（1）在项目中新建窗体类 RegistDemo，在窗体中添加标签、文本框等控件。

（2）编写 do_button_actionPerformed()方法完成对用户单击“提交”按钮事件的监听，在该方法中校验用户两次输入的密码是否相同，关键代码如下：

```
protected void do_button_actionPerformed(ActionEvent e) {
    char[] password1 = passwordField1.getPassword();                    // 获得密码框 1 中的密码
    char[] password2 = passwordField2.getPassword();                    // 获得密码框 2 中的密码
    boolean equals = Arrays.equals(password1, password2);               // 判断两个密码框中密码是否相同
    if (equals) {                                                       // 如果密码相同则提示注册成功
        JOptionPane.showMessageDialog(this, "用户注册成功！", null,JOptionPane.INFORMATION_MESSAGE);
    } else {                                                            // 如果密码不同则提示密码不同
        JOptionPane.showMessageDialog(this, "两次密码不同！", null,JOptionPane.WARNING_MESSAGE);
    }
}
```

DIY：在本实例的基础上增加一个文本框获得用户的 E-mail 信息，并对 E-mail 中是否含有@进行判断。（20 分）（实例位置：光盘\mr\ym\10.27\diy）

10.10.5　情景应用 5——测试密码是否正确

视频讲解：光盘\mr\lx\10\测试密码是否正确.exe

实例位置：光盘\mr\ym\10.28\src

对于登录界面而言，至少需要用户输入用户名和密码。如果用户名和密码与数据库或其他加密文件中的相同，则可以正常登录，否则需要提示用户登录失败。出于高安全性的需求，还可以记录用户登录失败的次数，如果超过指定次数，则在一定时间内不能进行登录。本实例完成简单的登录校验，实例的运行效果如图 10.40 所示。

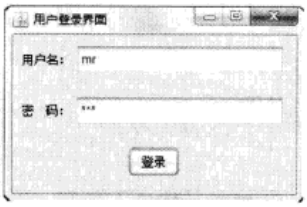


图 10.40　实例运行效果

实现过程如下：

（1）在项目中新建窗体类 LoginDemo，在窗体中添加标签、文本框等控件。

（2）使用 do_button_actionPerformed()方法完成对用户单击“登录”按钮事件的监听，如果用户输入的用户名为 mr，密码为 mrsoft，则登录成功，关键代码如下：

```
protected void do_button_actionPerformed(ActionEvent e) {
    String username = textField.getText();                                    // 获得用户输入的用户名
    String password = new String(passwordField.getPassword());                // 获得用户输入的密码
    if (username.equals("mr") && password.equals("mrsoft")) {                 // 如果用户名为 mr，密码为 mrsoft
        JOptionPane.showMessageDialog(this, "登录成功！", null,JOptionPane.INFORMATION_MESSAGE);
    } else {                                                                  // 提示登录成功
        JOptionPane.showMessageDialog(this, "登录失败！", null,JOptionPane.WARNING_MESSAGE);
    }                                                                         // 否则提示登录失败
}
```

DIY：在本实例的基础上增加一个计数器，提示用户还有几次可以输入错误的机会。（20 分）（实例位置：光盘\mr\ym\10.28\diy）

情景应用 DIY 栏目分数统计：

DIY 题目	1	2	3	4	5	总分数	
分数							

10.11 自 我 测 试

一、选择题（每题 10 分，5 道题）

1．下列哪些（个）是 Swing 中的窗体？（　　）

A．JFrame　　B．JDialog　　C．JApplet　　D．JLabel

2．下列哪些（个）是 Swing 中的布局管理器？（　　）

A．AbsoluteLayout　B．FlowLayout　　C．BorderLayout　D．GridLayout

3．JScrollPane 控件通常和哪（些）个控件组合使用？（　　）

A．JLabel　　B．JButton　　C．JTextField　　D．JTextArea

4．下列哪些（个）方法可以用来给 JComboBox 控件增加选项？（　　）

A．addItem()　　B．ComboBoxModel

C．Object[]　　D．Vector<?>

5．下列哪些（个）方法可以用来给 JList 控件增加选项？（　　）

A．addItem()　　B．ListModel　　C．Object[]　　D．Vector<?>

二、填空题（每题 10 分，5 道题）

1．Swing 中如果设置窗体为可见，需要使用哪个方法？_____

2．Swing 中窗体左上角坐标是？_____

3．Java 中，如果需要禁止 JButton 控件响应鼠标点击事件，可以使用哪个方法？_____

4．可以使用哪个方法来修改密码框控件的回显字符样式？_____

5．如果需要让进度条上显示文本信息，则需要使用哪个方法？_____

测试分数统计：

类别	第 1 题	第 2 题	第 3 题	第 4 题	第 5 题
选择题分数					
填空题分数					
总分数					

10.12　行 动 指 南

开始日期：_____年____月____日　　结束日期：_____年____月____日

序号	内　容	行 动 指 南	
1	照猫画虎栏目	分数>75 分	优秀，基本功掌握得不错，加油！
		75 分>分数>50 分	及格，知识掌握得不牢，重新做一遍照猫画虎。
	分数（　　）	分数<50 分	请使用光盘中提供的“实战能力测试系统”进行提高训练。
	情景应用栏目	分数>75 分	优秀，综合应用能力很强。
		75 分>分数>50 分	及格，综合应用能力需提高，再练一遍情景应用。
	分数（　　）	分数<50 分	请使用光盘中提供的“实战能力测试系统”进行提高训练。
	自我测试栏目	分数>75 分	优秀，有成为编程高手的潜质。
	分数（　　）	分数<75 分	请使用光盘中提供的“实战能力测试系统”进行提高训练。
	综合评价	反复训练后，以上各项分数都在 75 分以上，方可进入下一堂课学习。	
2	编程实践能力培养：本栏目提供了一些实践题目，对于培养编程能力很有效，要做好。	（1）淡入淡出的窗体。	
		（2）仿 QQ 隐藏窗体。	
		（3）制作圆形布局管理器。	
		（4）实现自动排序的列表。	
3	编程习惯培养：本堂课培养读者在学习开发中记笔记的习惯，把开发中遇到的问题、总结的经验记录下来。		
4	创新能力培养：看看身边有没有可以用编程解决的问题，记录到右边的表格中。根据学习进度，尝试编写解决这些问题的小程序。		

10.13　成功可以复制——“杀毒王”王江民

王江民，1951 年出生于上海，3 岁因患小儿麻痹后遗症而腿部残疾。初中毕业后，他回到老家山东烟

台，从一名街道工厂的学徒工干起，刻苦自学，成长为拥有各种创造发明 20 多项的机械和光电类专家。作为所在国营企业的技术骨干，先后被授予“全国新长征突击手标兵”、“全国青年自学成才标兵”称号。

改变王江民人生的阶段是 1987 年，那时王江民是所在国营企业的技术骨干。当时的国内计算机发展还处于预热期，但经常与高科技打交道的王江民认定计算机必将伴随高科技的发展改变人类社会的生活，如果不学，很可能会跟不上科技发展的步伐。“于是先买书，晚上熬夜，每天一点点地看”，开始了一个崭新领域的理论与实践探索之路。王江民说：“我 38 岁开始学计算机，没有感觉我老了，没有感觉我不行，只感到我的英语基础不好。再说，计算机是实践性非常强的学科。我搞计算机是用计算机，不是学计算机。”

机遇偏爱有准备肯钻研的人，1989 年，王江民花 1000 多元自己买了一台中华学习机，第二年又买了一台 8088PC 机。王江民首先学的是 Basic 语言。当时，王江民的孩子正上小学一年级，王江民在辅导孩子功课时突发奇想，能否编一程序代替家长辅导。没过多久，3 个月前尚对软件领域一片空白的王江民就编好了一套数学语文教学软件，试过后发现效果奇佳，拿到电脑报参加了一个评奖，结果被誉为“教育软件第一”，并被第一个在全国隆重推广。王江民从一开始就是在用计算机，而不是在学计算机。

软件获奖激发了王江民对软件业的极大兴趣，使他从此把精力转到了软件领域。在开发工具软件时，王江民发现一些或明或暗的病毒。王江民先是用 Debug 手工杀病毒，接着是写一段程序杀一种病毒。王江民有一个很好的习惯，就是杀一种病毒就在报刊上发表一篇文章，公布这段杀病毒的程序。后来，王江民觉得这些各自独立的杀病毒程序用起来很麻烦，就把 6 个杀不同病毒的程序集成到了一起，命名为 KV6，后来发展到 KV8、KV12、KV18、KV20。随着时间的推移，KV 系列杀毒软件在国内市场中一路高奏凯歌，先后囊括 16 项大奖，成为杀毒软件行业中无可置疑的领头羊。依靠 KV 系列软件，王江民也成为中关村的亿万富翁，并跻身“中国 IT 富豪榜 50 强”，成为新世纪（002280）“知识英雄”的典范，创造了中国软件行业的奇迹。

经典语录

“其实我最喜欢做的还是开发软件，去拯救那些被计算机病毒侵害的计算机。”

“我 38 岁开始学计算机，没有感觉我老了，没有感觉我不行，只感到我的英语基础不好。再说，计算机是实践性非常强的学科。我搞计算机是用计算机，不是学计算机。”

深度评价

王江民的成功启示我们，每一个想追求成功和传奇的人都会在他身上看到自己的希望和信心，因为王江民各方面的起点都非常之低，低到在外人看来凭着王江民的外在条件，他根本就没有任何成功的可能性。后来者们只要少一些浮躁，多一些热情和专注，多一些勤奋和执著，就一定能发现机会，并抓住机遇，书一段属于你自己的编程传奇。

第11堂课

多线程编程

（视频讲解：87分钟）

在前面的章节中，编写的程序都是单线程程序，因此程序只能一次做一件事情。例如，打开一个很大的文档，如果只看前面的一些内容就想关闭该文档，使用单线程程序是不能实现的。它必须完成当前的任务，打开全部文档，才能继续下一个任务，关闭文档。如果使用多线程编程，就可以让程序具有更好的互动性。对于单核计算机而言，多线程实际上是通过将 CPU 的使用划分成不同的时间片来实现的。目前，多核计算机越来越普及，使用多线程能够更好地利用它们。

学习摘要：

- 线程与进程的异同
- 线程的生命周期
- 线程的控制
- 线程的属性
- 实现线程同步的常见方式
- Swing 中的线程

11.1 线 程 基 础

线程是 Java 语言的关键技术之一，它在 Swing 中也扮演着重要的角色。本节将向读者介绍什么是线程以及如何创建并控制线程。

11.1.1 线程与进程的异同

现在的操作系统都支持多任务，如边听歌边下载电影，这通常是由不同的进程完成的。由于在操作系统上创建进程的开销非常大，因此提出了线程的概念，它相当于“轻量级”的进程。关于进程与线程之间的关系，可以参考图 11.1。

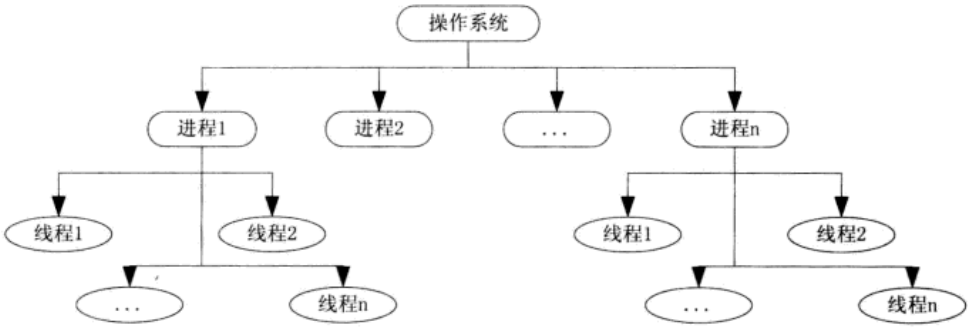


图 11.1 进程与线程的关系

在图 11.1 中，操作系统可以包括 n 个进程，这里 n 是由具体的系统所决定的，并不能是无穷大。对于一个进程而言，如进程 1，又可以包括 n 个线程，同样，这里 n 也是有数量限制的。

你问我答：线程和进程的本质区别是什么？

每个进程都拥有属于自己的一套变量，而在同一进程中创建的线程共享这些变量。因此，对于变量的修改通常需要使用同步。

Java 语言从诞生之日起就支持多线程编程。在 Object 类中定义了线程等待与唤醒方法，synchronized 关键字可以用来创建同步块或同步方法。随着 Java SE 5.0 版的发布，又新增了很多类来简化多线程开发。

注意：在使用 Java 语言编写多线程程序时，运行结果与操作系统密切相关，即使在同一台机器上，多次运行同一个程序，结果也可能不同。

11.1.2 创建无返回值线程

通常有两种方式用于创建无返回值线程，即实现 Runnable 接口和继承 Thread 类，下面分别对其进行讲解，并对这两者进行简单的对比。

1. 实现 Runnable 接口

Runnable 接口位于 java.lang 包中，这个接口非常简单，仅定义了一个 run()方法。通常将希望在新线程

中运行的代码写在这个方法中，方法声明如下：

```
public abstract void run();
```

例 11.01　下面通过一个简单的实例演示如何通过实现 Runnable 接口来创建新线程。实例的代码如下：

（实例位置：光盘\mr\ym\11.01）

```
public class RunnableDemo implements Runnable {
    @Override
    public void run() {                                    // 使用新线程运行 for 循环中的代码
        for (int i = 0; i < 10; i++) {
            System.out.println("新线程输出：" + i);          // 在控制台输出 10 条测试语句
        }
    }
    public static void main(String[] args) {
        Runnable runnable = new RunnableDemo();            // 创建 Runnable 接口的对象
        Thread thread = new Thread(runnable);              // 利用 Runnable 接口的对象创建 Thread 类对象
        thread.start();                                    // 调用 Thread 类的 start()方法来启动新线程
        for (int i = 0; i < 10; i++) {                     // 在主线程中运行 for 循环中的代码
            System.out.println("主线程输出：" + i);          // 在控制台输出 10 条测试语句
        }
    }
}
```

实例的运行效果如图 11.2 所示。

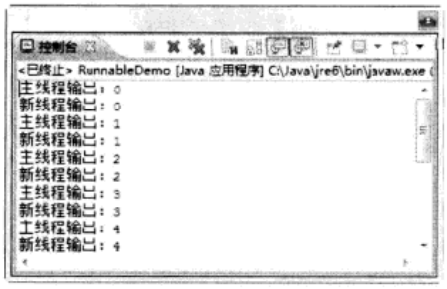


图 11.2　实例运行效果

说明：图 11.2 截取得并不完整，本实例的重点在于演示多线程程序的运行效果，读者可以看到 for 循环语句交替输出的效果。

下面对实现 Runnable 接口创建新线程的步骤总结如下：

（1）新建一个类，该类实现了 Runnable 接口并重写了 run()方法。run()方法中包含要在新线程中运行的代码。

（2）使用刚刚编写的类创建 Runnable 接口类型的对象。

（3）使用 Thread 类的构造方法创建 Thread 类的对象。

（4）调用 Thread 类的 start()方法来运行新线程。

注意：不要使用 Thread 类的 run()方法，它不会创建新的线程。

2．继承 Thread 类

Thread 类位于 java.lang 包，它包含了很多与线程密切相关的方法，该类还实现了 Runnable 接口。使用

Thread 类创建新线程时，也需要重写 run()方法。

例 11.02 下面通过一个简单的实例演示如何通过继承 Thread 类来创建新线程。实例的代码如下：**（实例位置：光盘\mr\ym\11.02）**

```
public class ThreadDemo extends Thread {
    @Override
    public void run() {                                    // 使用新线程运行 for 循环中的代码
        for (int i = 0; i < 10; i++) {
            System.out.println("新线程输出：" + i);          // 在控制台输出 10 条测试语句
        }
    }

    public static void main(String[] args) {
        ThreadDemo thread = new ThreadDemo();              // 创建 Thread 类对象
        thread.start();                                    // 调用 Thread 类的 start()方法来启动新线程
        for (int i = 0; i < 10; i++) {                     // 在主线程中运行 for 循环中的代码
            System.out.println("主线程输出：" + i);          // 在控制台输出 10 条测试语句
        }
    }
}
```

实例的运行效果如图 11.3 所示。

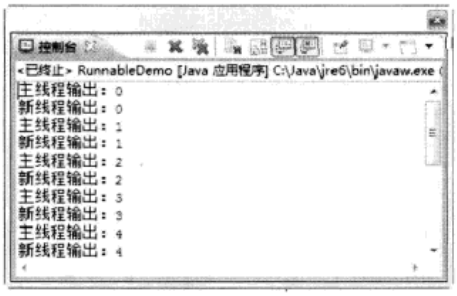


图 11.3 实例运行效果

说明：图 11.3 截取得并不完整，本实例的重点在于演示多线程程序的运行效果，读者可以看到 for 循环语句交替输出的效果。

3．两种方式的对比

通常推荐使用第一种方式，即实现 Runnable 接口来创建新线程，理由如下：

- ☑ Java 仅支持单继承，一旦一个类继承了其他类，则不能继承 Thread 类。
- ☑ 创建大量的 Thread 类对象，开销较大。
- ☑ Java SE 5.0 版中新增的很多简化线程开发的类是使用 Runnable 接口的。

11.1.3 线程生命周期详解

线程是具有生命周期的，生命周期是由不同的状态组成的。对于不同的状态，能够执行的操作不同。例如，对于已经处于 TERMINATED 状态的线程，是不能够再让其运行的。在 Thread 类的内部，定义了一个枚举 State，其中包含了线程的全部状态，其说明如表 11.1 所示。

表 11.1　Thread.State 枚举常量说明

枚 举 常 量	含　义
NEW	新建状态
RUNNABLE	运行（可运行）状态
BLOCKED	阻塞状态
TIMED_WAITING	休眠状态
WAITING	等待状态
TERMINATED	终止状态

以上状态的关系如图 11.4 所示。

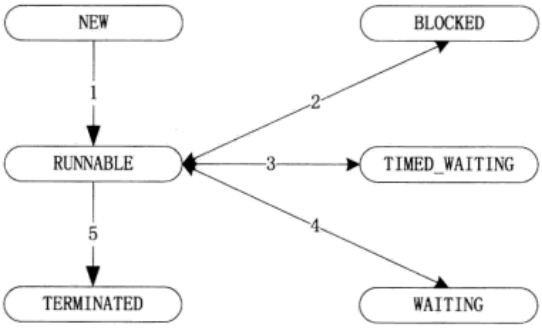


图 11.4　线程的生命周期

图中各个数字的含义如下。

- ☑ 1：线程由 NEW 状态变成 RUNNABLE 状态，这可以通过调用 start()方法实现。
- ☑ 2：线程在 RUNNABLE 状态和 BLOCKED 状态之间进行转换，这可以通过得到和等待内置锁实现。
- ☑ 3：线程在 RUNNABLE 状态和 TIMED_WAITING 状态之间进行转换，这可以通过调用 sleep()等方法实现。
- ☑ 4：线程在 RUNNABLE 状态和 WAITING 状态之间进行转换，这可以通过调用 wait()等方法实现。
- ☑ 5：线程由 RUNNABLE 状态变成 TERMINATED 状态，这可能是由于 run()方法运行结束或出现异常。

📖 **说明：当使用 new 操作符创建 Thread 对象时，线程处于 NEW 状态。**

11.1.4　线程各种属性介绍

1．线程优先级属性

Java 中所有线程都具有优先级属性，默认的优先级与创建该线程的线程优先级相同。可以使用 Thread 类的 getPriority()方法获得线程优先级，使用 setPriority()方法设置线程优先级。

2．线程名称属性

每个线程在创建时可以为其指定名称，如果未指定，则由系统进行指定。对于主线程，名称是 main。对于其他线程，默认名称样式是“Thread-数字”。可以使用 Thread 类的 getName()方法获得线程名称、setName()

方法设置线程名称。

注意：不同的线程可以有相同的名称，不能使用线程名称来区分线程。

3. 线程 ID 属性

每个线程在创建时就会被分配一个 ID 属性，可以使用 Thread 类的 getId()方法获得线程 ID。Thread 类并没有提供修改 ID 的方法。

注意：不同的线程不能有相同的 ID，系统是使用 ID 值来区分线程的。

4. 守护线程

守护线程的作用是为其他线程提供服务，如果其他线程运行完毕，即虚拟机中只剩下守护线程时，虚拟机会退出。

注意：不要让守护线程去访问文件等资源，它们可能在任何时候终止而资源却没有释放。

11.2 线 程 控 制

读者已经了解到线程对象是 Thread 类的子类，或者说是 Runnable 接口的实现类的实例对象。线程创建以后需要进行启动、休眠、停止等控制，本节将对这些线程控制的方法进行介绍。

11.2.1 线程的休眠

Thread 类的 sleep()方法可以让线程进入休眠状态。在使用该方法时，需要指明线程休眠的时间，在时间到达后，线程会自动唤醒，该方法有以下两种形式。

☑ 精度到毫秒，方法声明如下：

```
public static void sleep(long millis)throws InterruptedException
```

参数说明：millis 是以毫秒为单位的休眠时间。

☑ 精度到纳秒，方法声明如下：

```
public static void sleep(long millis,int nanos)throws InterruptedException
```

参数说明：millis 是以毫秒为单位的休眠时间，nanos 是以纳秒为单位的休眠时间。

说明：这两个方法都受系统计时器和调度程序精度和准确性的影响，因此时间可能并不十分精确。

注意：这两个方法都会抛出 InterruptedException 异常，需要对其进行捕获或抛出。

说明：1 秒= 10^3 毫秒= 10^6 微秒= 10^9 纳秒。

例 11.03 下面通过一个简单的实例演示如何使用 sleep()方法休眠线程。实例的代码如下：**（实例位置：光盘\mr\ym\11.03）**

```
public class SleepDemo {
    public static void main(String[] args) {
        long beforeTime = System.currentTimeMillis();          // 获得系统当前时间
        try {
            Thread.sleep(1000);                                // 线程休眠 1 秒钟
```

```
            } catch (InterruptedException e) {
                e.printStackTrace();                                    // 输出异常信息
            }
            long afterTime = System.currentTimeMillis();                // 获得系统当前时间
            System.out.print("线程休眠时间是：");
            System.out.println(afterTime - beforeTime + "毫秒");        // 获得线程休眠时间
        }
    }
```

实例的运行效果如图 11.5 所示。

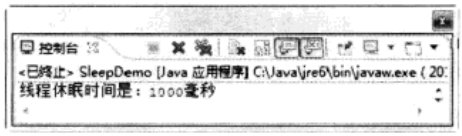


图 11.5　实例运行效果

11.2.2　线程的插队

通常情况下，线程调度器是根据线程优先级来分配资源的。对于高优先级的线程，其获得资源的几率也很高。如果想控制线程的执行顺序，例如，让某个线程执行到一半时让另一个线程优先运行完毕，再运行前一个线程，可以使用 Thread 类提供的 join()方法，该方法有 3 种形式。

☑　无时间参数，方法说明如下：

```
public final void join()throws InterruptedException
```

☑　精度到毫秒，方法说明如下：

```
public final void join(long millis)throws InterruptedException
```

参数说明：millis 是以毫秒为单位的休眠时间。

☑　精度到纳秒，方法说明如下：

```
public final void join(long millis,int nanos)throws InterruptedException
```

参数说明：millis 是以毫秒为单位的休眠时间，nanos 是以纳秒为单位的休眠时间。

注意：这两个方法都会抛出 InterruptedException 异常，需要对其进行捕获或抛出。

例 11.04　下面通过一个简单的实例演示如何使用 join()方法进行线程插队。实例的代码如下：**（实例位置：光盘\mr\ym\11.04）**

```
public class JoinDemo implements Runnable {
    @Override
    public void run() {                                         // 使用新线程运行 for 循环中的代码
        for (int i = 0; i < 5; i++) {
            System.out.println("航空邮件" + i);                  // 在控制台输出 10 条测试语句
        }
    }
    public static void main(String[] args) {
        Thread thread = new Thread(new JoinDemo());             // 利用 Runnable 接口的对象创建 Thread 类对象
        thread.start();                                         // 调用 Thread 类的 start()方法来启动新线程
        try {
            thread.join();                                      // 让新线程插队运行
        } catch (InterruptedException e) {
```

```
            e.printStackTrace();                                    // 输出异常信息
        }
        for (int i = 0; i < 5; i++) {                               // 在主线程中运行 for 循环中的代码
            System.out.println("普通邮件" + i);                      // 在控制台输出 10 条测试语句
        }
    }
}
```

实例的运行效果如图 11.6 所示。

图 11.6　实例运行效果

说明： 读者可以发现，所有航空邮件优先输出，然后才是普通邮件，这就是 join()方法的作用。要注释掉该方法再输出一下结果，就会发现不同。

11.2.3　线程的停止

线程通常是在 run()方法运行完毕后停止运行的，此外还可以使用布尔值来让线程提前停止。虽然 Thread 类也提供了 stop()方法来停止线程，但是由于该方法固有的不安全性，不推荐使用。

例 11.05　下面通过一个简单的实例演示如何使用布尔值停止线程。实例的代码如下：**（实例位置：光盘\mr\ym\11.05）**

```
public class ThreadTerminateDemo implements Runnable {
    @Override
    public void run() {
        int number = 0;                                             // 创建一个 int 类型的变量 number
        while (true) {                                              // 使用死循环进行输出
            System.out.print(number++ + " ");                       // number 的值每次加 1
            if (number % 15 == 0) {                                 // 如果 number 是 15 的倍数则换行
                System.out.println();
            }
            if (number == 100) {                                    // 如果 number 到达 100 则退出死循环
                break;
            }
        }
    }
    public static void main(String[] args) {
        Thread thread = new Thread(new ThreadTerminateDemo());      // 新建线程
        thread.start();                                             // 运行线程
    }
}
```

实例的运行效果如图 11.7 所示。

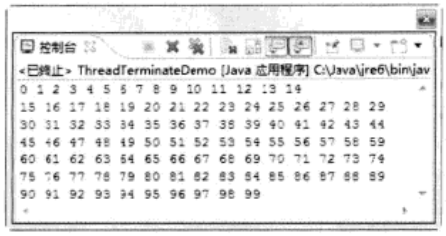


图 11.7　实例运行效果

技巧：也可以在 while 循环判断处使用一个布尔变量，当到达某一条件时，将这个变量修改成 false，就能退出 while 循环，这样代码更加灵活。

11.3　线 程 同 步

在单线程程序中，每次只能做一件事情，后面的事情也需要等前面的事情完成后才可以进行，如果使用多线程程序，虽然能够实现多处理，但是会发生两个以上的线程抢占资源的问题，如两个人同时使用一个麦克风等。在多线程编程中，必须防止这些资源访问的冲突。Java 提供线程同步的机制防止资源访问的冲突。

11.3.1　需要同步的原因

在编写多线程程序时，会遇到两个或两个以上线程同时修改一个共享数据的情形。此时，如果不使用同步，就会发生错误。

例 11.06　下面通过一个简单的实例演示如果不使用同步，两个线程同时修改一个共享数据可能引发的问题。实例的代码如下：**（实例位置：光盘\mr\ym\11.06）**

```
public class RaceConditionDemo implements Runnable {
    private int number = 0;                                          // 创建一个 int 类型变量进行测试
    @Override
    public void run() {
        for (int i = 0; i < 5; i++) {
            try {
                Thread.sleep(100);                                   // 线程休眠 0.1 秒增加出错几率
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            // 输出当前线程的名称和 number 的值，每次循环之后 number 的值都会加 1
            System.out.println(Thread.currentThread().getName() + ": "+ number++);
        }
    }
    public static void main(String[] args) {
        RaceConditionDemo run = new RaceConditionDemo();             // 获得 RaceConditionDemo 对象
        Thread thread1 = new Thread(run);                            // 创建线程 1
        Thread thread2 = new Thread(run);                            // 创建线程 2
        thread1.start();                                             // 运行线程 1
```

```
            thread2.start();                                    // 运行线程 2
        }
    }
```

实例的运行效果如图 11.8 所示。

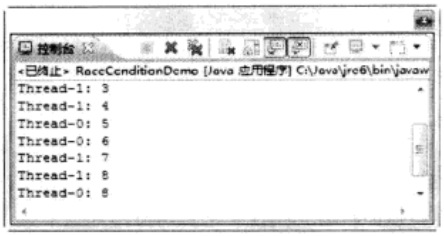


图 11.8　实例运行效果

注意：图 11.8 中最后两个结果都是 8，即最后一个自增运算并没有成功执行。

当修改 number 的值时，首先读取 number 的值，然后计算 number+1 的值，最后保存 number 的值。如果这 3 步中间又插入了其他线程来读取数据，则会发生问题，如图 11.9 所示。

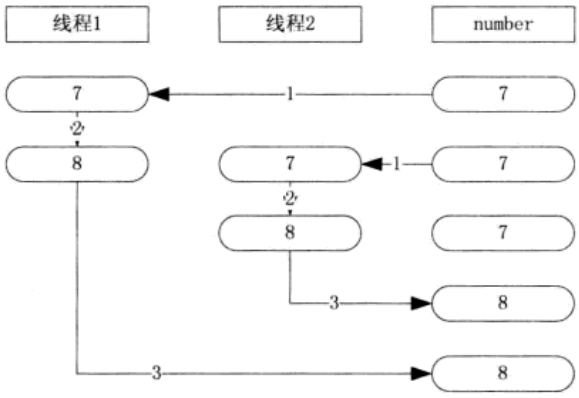


图 11.9　同时修改 number 的结果

说明：图中 number 的值选择的是 7，这样就可以用来解释例 11.06 的运行结果。

图中数字 1 表示读取 number 的值，数字 2 表示 number 的值加 1，数字 3 表示保存 number 的值。

11.3.2　同步方法的使用

Java 中提供了很多种实现同步的方式，本节将讲述最简单的同步方法。所谓同步方法就是使用 synchronized 关键字修饰的方法。如果某个对象调用了同步方法，其他对象在该方法运行完毕前是不能再进入这个方法的。这样就可以避免因为同时修改数据而引发的问题。

例 11.07　下面通过一个简单的实例演示同步方法的使用。实例的代码如下：**（实例位置：光盘\mr\ym\11.07）**

```
public class SynchronizedMethodDemo implements Runnable {
    private int number = 0;                                     // 创建一个 int 类型变量进行测试
    @Override
```

```
    public synchronized void run() {
        for (int i = 0; i < 5; i++) {
            try {
                Thread.sleep(100);                                    // 线程休眠 0.1 秒增加出错几率
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            // 输出当前线程的名称和 number 的值，每次循环之后 number 的值都会加 1
            System.out.println(Thread.currentThread().getName() + ": " + number++);
        }
    }
    public static void main(String[] args) {
        SynchronizedMethodDemo run = new SynchronizedMethodDemo(); // 获得 SynchronizedMethodDemo 对象
        Thread thread1 = new Thread(run);                          // 创建线程 1
        Thread thread2 = new Thread(run);                          // 创建线程 2
        thread1.start();                                           // 运行线程 1
        thread2.start();                                           // 运行线程 2
    }
}
```

实例的运行效果如图 11.10 所示。

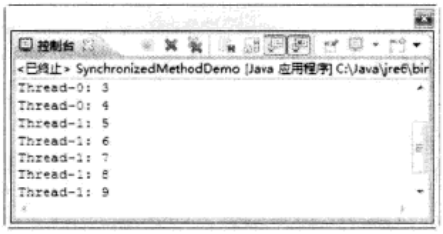


图 11.10　实例运行效果

说明：在使用 synchronized 关键字修饰 run()方法后，在一个线程运行完毕前，其他的线程并没有进入该方法。

11.3.3　使用同步块减负

虽然同步方法很简单，但同步的开销是很大的，因此应该尽量减少同步的代码。此时可以考虑使用同步块来减负。

例 11.08　下面通过一个简单的实例演示同步块的使用。实例的代码如下：**（实例位置：光盘\mr\ym\11.08）**

```
public class SynchronizedBlockDemo implements Runnable {
    private int number = 0;                                        // 创建一个 int 类型变量进行测试
    @Override
    public void run() {
        for (int i = 0; i < 5; i++) {
            try {
                Thread.sleep(100);                                 // 线程休眠 0.1 秒增加出错几率
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
```

```
            synchronized (this) {
                // 输出当前线程的名称和 number 的值，每次循环之后 number 的值都会加 1
                System.out.println(Thread.currentThread().getName() + ": " + number++);
            }
        }
    }
    public static void main(String[] args) {
        SynchronizedBlockDemo run = new SynchronizedBlockDemo(); // 获得 SynchronizedBlockDemo 对象
        Thread thread1 = new Thread(run);                        // 创建线程 1
        Thread thread2 = new Thread(run);                        // 创建线程 2
        thread1.start();                                         // 运行线程 1
        thread2.start();                                         // 运行线程 2
    }
}
```

实例的运行效果如图 11.11 所示。

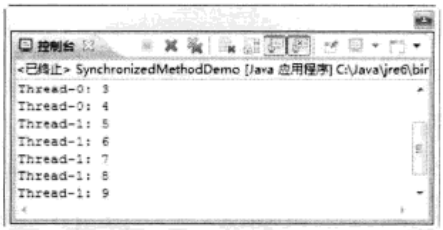


图 11.11　实例运行效果

11.3.4　使用特殊成员变量

如果仅仅因为一两个成员变量的修改而使用同步，开销还是很大。此时可以使用 volatile 关键字来修饰成员变量。它的作用在于告诉虚拟机该成员变量可能已经被修改了，需要进行确认才能使用，这样就避免了同步的需求。

例 11.09　下面通过一个简单的实例演示 volatile 关键字的使用。实例的代码如下：**（实例位置：光盘\mr\ym\11.09）**

```
public class VolatileDemo implements Runnable {
    private volatile int number = 0;                         // 创建一个 int 类型变量进行测试
    @Override
    public void run() {
        for (int i = 0; i < 5; i++) {
            try {
                Thread.sleep(100);                           // 线程休眠 0.1 秒增加出错几率
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            // 输出当前线程的名称和 number 的值，每次循环之后 number 的值都会加 1
            System.out.println(Thread.currentThread().getName() + ": " + number++);
        }
    }
    public static void main(String[] args) {
        VolatileDemo run = new VolatileDemo();               // 获得 VolatileDemo 对象
```

```
        Thread thread1 = new Thread(run);                    // 创建线程 1
        Thread thread2 = new Thread(run);                    // 创建线程 2
        thread1.start();                                     // 运行线程 1
        thread2.start();                                     // 运行线程 2
    }
}
```

实例的运行效果如图 11.12 所示。

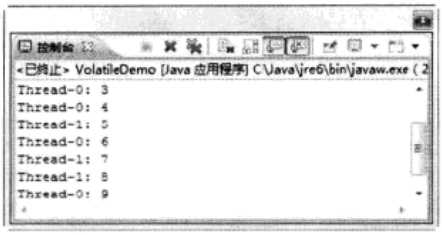


图 11.12　实例运行效果

11.4　线 程 应 用

使用多线程编程的原因之一是提高程序的响应性能，这在 Swing 中尤为重要。对于用户而言，更希望程序能够时刻响应自己的各种操作。本节将重点讲述线程在 Swing 中的应用。

注意：Swing 不是线程安全的。如果希望在多个线程中操作同一控件，很可能让程序崩溃。

11.4.1　EventQueue 与线程分配

EventQueue 是一个与平台无关的类，它是 Swing 的一个重要组成部分，负责 AWTEvent 类及其子类的分发。该类首先将所有事件放置在一个队列中，然后使用 dispatchEvent()方法来依次执行各个事件。

Swing 中的事件如果耗时过长，就会让程序暂时死掉，这是因为使用 dispatchEvent()方法来依次执行各个事件。为了将 Swing 程序在事件分配线程中运行，需要使用 invokeLater()方法，该方法的声明如下：

```
public static void invokeLater(Runnable runnable)
```

参数说明：runnable 是 Runnable 的对象，其 run 方法应该在 EventQueue 上同步执行。

例 11.10　下面通过一个简单的实例演示 EventQueue 类的使用，它可以在窗体上输出一个随机数。实例的代码如下：**（实例位置：光盘\mr\ym\11.10）**

```
public class EventQueueDemo extends JFrame {
    private static final long serialVersionUID = 5852413451918464763L;
    private JPanel contentPane;
    private JLabel label;
    public static void main(String[] args) {
        try {
            UIManager.setLookAndFeel("com.sun.java.swing.plaf.nimbus.NimbusLookAndFeel");
                                                                    // 更改 Swing 的外观
        } catch (Throwable e) {
            e.printStackTrace();
        }
```

```
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                try {
                    EventQueueDemo frame = new EventQueueDemo(); // 在 EventQueue 上创建窗体对象
                    frame.setVisible(true);                      // 将窗体对象设置才可见
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }
    public EventQueueDemo() {
        setTitle("EventQueue\u4F7F\u7528");                      // 设置窗体的标题
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);          // 设置窗体退出时属性
        setBounds(100, 100, 300, 150);                           // 设置窗体大小和位置
        contentPane = new JPanel();                              // 创建内容面板
        contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));      // 将内容面板边框设置为空
        setContentPane(contentPane);                             // 在窗体中增加内容面板
        contentPane.setLayout(new GridLayout(2, 1, 5, 5));
                                                    // 将内容面板的布局修改成两行一列的网格布局
        JPanel labelPanel = new JPanel();                        // 创建标签面板用于保存标签
        contentPane.add(labelPanel);                             // 在内容面板上增加标签面板
        labelPanel.setLayout(new BorderLayout(0, 0));            // 设置标签面板的布局为边框布局
        label = new JLabel("\u968F\u673A\u6570\uFF1A");          // 创建用于显示随机数的标签
        label.setHorizontalAlignment(SwingConstants.CENTER);     // 将标签文本居中显示
        labelPanel.add(label);                                   // 在标签面板上增加标签
        JPanel buttonPanel = new JPanel();                       // 创建按钮面板
        contentPane.add(buttonPanel);                            // 在内容面板上增加按钮面板
        JButton button = new JButton("\u751F\u6210\u968F\u673A\u6570"); // 创建按钮
        button.addActionListener(new ActionListener() {          // 为单击按钮事件增加监听器
            public void actionPerformed(ActionEvent e) {
                do_button_actionPerformed(e);
            }
        });
        buttonPanel.add(button);                                 // 在按钮面板上增加按钮
    }
    protected void do_button_actionPerformed(ActionEvent e) {
        int number = new Random().nextInt();                     // 获得一个随机整数
        label.setText("\u968F\u673A\u6570\uFF1A" + number);      // 将随机数显示在标签上
    }
}
```

实例的运行效果如图 11.13 所示。

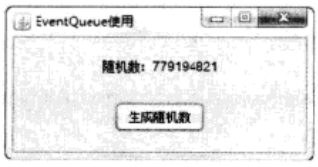


图 11.13　实例运行效果

11.4.2　SwingWorker 类的使用

对于 Swing 中的耗时任务，可以使用 Java SE 6.0 版中提供的 SwingWorker 类来实现。该类是 Swing 中专门用于执行耗时任务的抽象类。

例 11.11　下面通过一个简单的实例演示 SwingWorker 类的使用，它可以在 1000 个随机数中选择最大的并在窗体上显示。实例的关键代码如下：**（实例位置：光盘\mr\ym\11.11）**

```
private class RandomNumber extends SwingWorker<Void, Integer> {
    @Override
    protected Void doInBackground() throws Exception {
        TreeSet<Integer> set = new TreeSet<Integer>();          // 创建一个 TreeSet 类型对象
        for (int i = 0; i < 1000; i++) {
            set.add(new Random().nextInt());                    // 在 TreeSet 中存入 1000 个随机整数
        }
        label.setText("最大随机数是：" + set.last());           // 在标签上显示最大的整数
        return null;
    }
}
```

实例的运行效果如图 11.14 所示。

图 11.14　实例运行效果

说明：为了节约空间，例 11.11 中仅列出了 RandomNumber 类的代码，其他代码读者可以参考光盘中的源文件。

11.5　照猫画虎——基本功训练

11.5.1　基本功训练 1——查看线程的运行状态

视频讲解：光盘\mr\lx\11\查看线程的运行状态.exe

实例位置：光盘\mr\ym\11.12\src

对于线程而言，不同的状态能够执行的操作是不同的。本实例将使用 Thread 类中定义的 getState()方法查看新建线程的状态，代码如下：

```
public class ThreadStateDemo {
    public static void main(String[] args) {
        Thread thread = new Thread();          // 创建一个新线程
        System.out.println("新建线程的运行状态：" + thread.getState());
    }
}
```

实例的运行效果如图 11.15 所示。

图 11.15　实例运行效果

你问我答：getState()方法的作用是什么？

该方法是用来帮助用户获得当前线程的状态。Java 中线程的状态如表 11.1 所示。

照猫画虎：将线程启动并查看线程的运行状态。（20 分）（实例位置：光盘\mr\ym\11.12\zmhh）

11.5.2　基本功训练 2——查看线程的名称和 ID

视频讲解：光盘\mr\lx\11\查看线程的名称和 ID.exe

实例位置：光盘\mr\ym\11.13\src

在创建新线程时，用户可以指定线程的名称并查看线程的名称和 ID 值。不同线程之间是使用 ID 值进行区分的，因此不能修改线程的 ID 值。本实例将演示如何查看线程的名称和 ID 值，代码如下：

```
public class ThreadProperties {
    public static void main(String[] args) {
        Thread thread = Thread.currentThread();                    // 获得当前线程
        System.out.println("获得当前线程 ID：" + thread.getId());
        System.out.println("获得当前线程名称：" + thread.getName());
    }
}
```

实例的运行效果如图 11.16 所示。

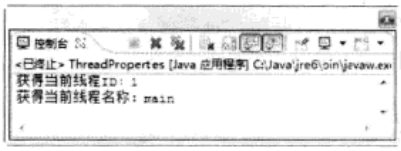


图 11.16　实例运行效果

照猫画虎：新建一个线程并查看该线程的名称和 ID，然后使用 setName()方法修改线程名称，最后再次查看线程名称和 ID。（20 分）（实例位置：光盘\mr\ym\11.13\zmhh）

11.5.3　基本功训练 3——查看线程优先级属性

视频讲解：光盘\mr\lx\11\查看线程优先级属性.exe

实例位置：光盘\mr\ym\11.14\src

线程的优先级对于线程而言是非常重要的属性，它往往涉及如何分配资源。本实例将演示如何查看线程的优先级，代码如下：

```
public class ThreadPriorityDemo {
    public static void main(String[] args) {
        Thread thread = new Thread();                                  // 新建线程
        System.out.println("新建线程的优先级：" + thread.getPriority());
        System.out.println("将新建线程的优先级修改成 10");
```

```
        thread.setPriority(10);                                    // 修改线程的优先级为 10
        System.out.println("新建线程的优先级：" + thread.getPriority());
    }
}
```

实例的运行效果如图 11.17 所示。

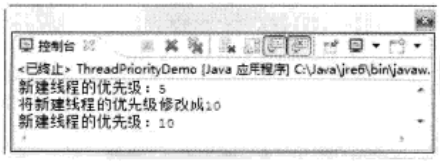


图 11.17　实例运行效果

照猫画虎：获得和修改主线程的优先级。（20 分）（实例位置：光盘\mr\ym\11.14\zmhh）

11.5.4　基本功训练 4——使用重入锁实现同步

视频讲解：光盘\mr\lx\11\使用重入锁实现同步.exe

实例位置：光盘\mr\ym\11.15\src

在 Java SE 5.0 版中，新增了很多高级的线程同步工具，重入锁就是其中之一。本实例将演示如何使用它来实现同步，代码如下：

```
public class ReentrantLockDemo implements Runnable {
    private int number = 0;                                        // 创建一个 int 类型变量进行测试
    private Lock lock = new ReentrantLock();                       // 创建重入锁对象
    @Override
    public void run() {
        lock.lock();                                               // 打开锁
        try {
            for (int i = 0; i < 5; i++) {
                try {
                    Thread.sleep(100);                             // 线程休眠 0.1 秒增加出错几率
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
                // 输出当前线程的名称和 number 的值，每次循环之后 number 的值都会加 1
                System.out.println(Thread.currentThread().getName() + ": " + number++);
            }
        } finally {
            lock.unlock();                                         // 释放锁
        }
    }
    public static void main(String[] args) {
        ReentrantLockDemo run = new ReentrantLockDemo();           // 获得 ReentrantLockDemo 对象
        Thread thread1 = new Thread(run);                          // 创建线程 1
        Thread thread2 = new Thread(run);                          // 创建线程 2
        thread1.start();                                           // 运行线程 1
        thread2.start();                                           // 运行线程 2
    }
}
```

实例的运行效果如图 11.18 所示。

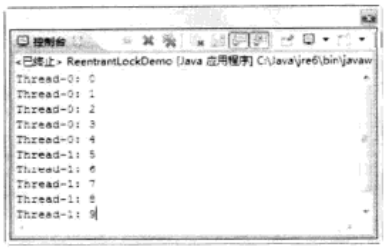


图 11.18　实例运行效果

你问我答：如何使用 ReentrantLock 类？

ReentrantLock 类位于 java.util.concurrent 包中，是一个可重入的互斥锁。ReentrantLock 具有与使用 synchronized 方法和语句所访问的隐式监视器锁相同的一些基本行为和语义，但功能更强大。该类中常用的方法如表 11.2 所示。

表 11.2　ReentrantLock 类中常用方法

方 法 名	说　明
ReentrantLock()	创建一个 ReentrantLock 实例
lock()	获得锁
unlock()	释放锁

注意：在使用重入锁时，需要使用 finally 块来执行释放锁的代码。

照猫画虎：模拟银行的存钱系统，让两个人同时向一个账户多次存钱，使用重入锁来避免并发问题。（20 分）（实例位置：光盘\mr\ym\11.15\zmhh）

11.5.5　基本功训练 5——原子变量与线程同步

视频讲解：光盘\mr\lx\11\原子变量与线程同步.exe

实例位置：光盘\mr\ym\11.16\src

在 Java SE 5.0 版中，新增了很多高级的线程同步工具，原子变量就是其中之一。本实例将演示如何使用它来实现同步，代码如下：

```
public class AtomicIntegerDemo implements Runnable {
    private AtomicInteger number = new AtomicInteger(0);      // 创建一个 AtomicInteger 类型变量进行测试
    @Override
    public void run() {
        for (int i = 0; i < 5; i++) {
            try {
                Thread.sleep(100);                            // 线程休眠 0.1 秒增加出错几率
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            // 输出当前线程的名称和 number 的值，每次循环之后 number 的值都会加 1
            System.out.println(Thread.currentThread().getName() + ": " + number.getAndIncrement());
        }
```

```
    }
    public static void main(String[] args) {
        AtomicIntegerDemo run = new AtomicIntegerDemo();      // 获得 AtomicIntegerDemo 对象
        Thread thread1 = new Thread(run);                     // 创建线程 1
        Thread thread2 = new Thread(run);                     // 创建线程 2
        thread1.start();                                      // 运行线程 1
        thread2.start();                                      // 运行线程 2
    }
}
```

实例的运行效果如图 11.19 所示。

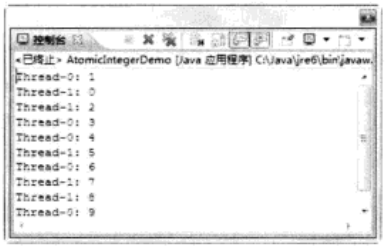


图 11.19　实例运行效果

你问我答：如何使用 AtomicInteger 类？

AtomicInteger 类位于 java.util.concurrent 包中，该类是用来表示支持原子操作的整数。该类中常用的方法如表 11.3 所示。

表 11.3　AtomicInteger 类中常用方法

方　法　名	说　　明
AtomicInteger(int initialValue)	创建值为 initialValue 的整型原子变量
getAndIncrement()	以原子方式将当前值加 1

照猫画虎：模拟银行的存钱系统，让两个人同时向一个账户多次存钱，使用原子变量来避免并发问题。（20 分）（实例位置：光盘\mr\ym\11.16\zmhh）

照猫画虎栏目分数统计：

照猫画虎题目	1	2	3	4	5	总分数	
分数							

11.6　情景应用——拓展与实践

11.6.1　情景应用 1——水平移动的图标

视频讲解：光盘\mr\lx\11\水平移动的图标.exe

实例位置：光盘\mr\ym\11.17\src

对于应用程序而言，如果能够增加一些动画效果是非常有益的。本实例将完成一个非常简单的动画，

让一个图片做水平移动，其实现思路是使用标签来显示图片，然后不断地修改标签的显示位置来改变图片的显示位置，实例的运行效果如图 11.20 所示。

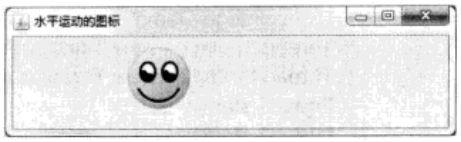


图 11.20　实例运行效果

实现过程如下：

（1）在项目中新建窗体类 DynamicIconDemo，在窗体中添加标签控件。

（2）编写内部类 IconMotion，它实现了 Runnable 接口。在 run()方法中，每隔 0.01 秒让标签水平向右移动 1 像素，垂直坐标不变。使用 for 循环让该操作完成 340 次，这样图片刚好从窗体的左侧移动到右侧。关键代码如下：

```
private class IconMotion implements Runnable {
    @Override
    public void run() {
        for (int i = 0; i < 340; i++) {
            try {
                Thread.sleep(10);                          // 当前线程休眠 0.01 秒
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            label.setBounds(10 + i, 10, 72, 72);           // 让标签在每次循环后都向右移动 1 个像素
        }
    }
}
```

DIY：实现垂直运动的图标。提示：修改标签位置时，让纵坐标不断地变化。（20 分）（实例位置：光盘\mr\ym\11.17\diy）

11.6.2　情景应用 2——多彩的霓虹灯

视频讲解：光盘\mr\lx\11\多彩的霓虹灯.exe

实例位置：光盘\mr\ym\11.18\src

本实例将完成一个多彩的霓虹灯，它可以不断地变换颜色。其实现思路是在窗体上增加 7 个标签，每隔 1 秒钟为这些标签依次更换颜色，这样就实现了动态变换颜色的效果。这些颜色是使用数组预先定义好，然后使用随机数来获取。实例的运行效果如图 11.21 所示。

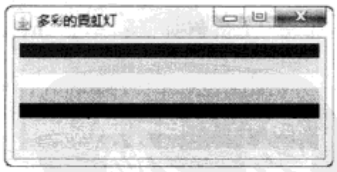


图 11.21　实例运行效果

实现过程如下：

（1）在项目中新建窗体类 NeonLight，在窗体中添加标签控件。

（2）编写内部类 ColorVariance，它实现了 Runnable 接口。在 run()方法中，使用死循环不断地更新标签的背景色。每次更新线程要休眠 1 秒钟，这样看起来就是动态更新的效果。每次使用一个 0～6 的随机数获得数组中的颜色，用于依次更新各个标签的背景色。关键代码如下：

```
private class ColorVariance implements Runnable {
    @Override
    public void run() {
        while (true) {
            try {
                Thread.sleep(1000);                        // 线程休眠 1 秒钟
            } catch (InterruptedException e) {
```

```
                e.printStackTrace();
            }
            for (int i = 0; i < label.length; i++) {                    // 从上至下依次改变标签的背景色
                label[i].setBackground(color[new Random().nextInt(7)]);
            }
        }
    }
}
```

DIY：模仿本实例，实现颜色不断变化的文字。（20 分）（实例位置：光盘\mr\ym\11.18\diy）

11.6.3　情景应用 3——电子广告牌

视频讲解：光盘\mr\lx\11\电子广告牌.exe

实例位置：光盘\mr\ym\11.19\src

在现代大都市中，电子广告牌随处可见。本实例也将完成一个，用于定期宣传公司新出版的图书。其实现思想是每隔 1 秒钟设置一下标签上的文本信息，实例的运行效果如图 11.22 所示。

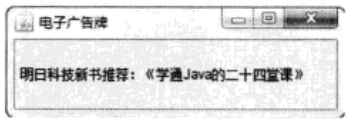


图 11.22　实例运行效果

实现过程如下：

（1）在项目中新建窗体类 BillBoard，在窗体中添加标签控件。

（2）编写内部类 Content，它实现了 Runnable 接口。在 run()方法中，使用死循环不断地更新标签的文本。每次更新线程要休眠 1 秒钟，这样看起来就是动态更新的效果。每次使用一个 0～4 的随机数获得数组中的字符串，用于更新标签上的文本。关键代码如下：

```
private class Content implements Runnable {
    @Override
    public void run() {
        String[] title = new String[5];                                 // 使用字符串数组保存广告内容
        title[0] = "《学通 Java 的二十四堂课》";
        title[1] = "《学通 C 语言的二十四堂课》";
        title[2] = "《学通 VC 的二十四堂课》";
        title[3] = "《学通 VB 的二十四堂课》";
        title[4] = "《学通 PHP 的二十四堂课》";
        while (true) {
            try {
                Thread.sleep(1000);                                     // 线程休眠 1 秒钟
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            label.setText("明日科技新书推荐："+ title[new Random().nextInt(title.length)]);
                                                                        // 随机更换标签中的文本信息
        }
    }
}
```

DIY：通过给标签增加图标来美化广告牌。提示：可以使用标签的 setIcon()方法增加图标。（20 分）（实例位置：光盘\mr\ym\11.19\diy）

11.6.4 情景应用 4——逐渐上升的气球

视频讲解：光盘\mr\lx\11\逐渐上升的气球.exe

实例位置：光盘\mr\ym\11.20\src

本实例将完成一个简单的动画效果——逐渐上升的气球，将气球图片放到一个标签中，通过不断地修改标签的位置，就可以让气球看起来是动态上升的，实例的运行效果如图 11.23 所示。

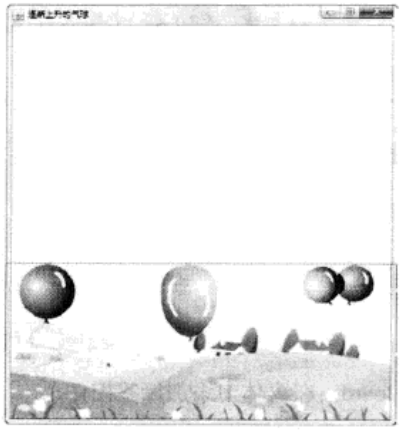


图 11.23 实例运行效果

实现过程如下：

（1）在项目中新建窗体类 FlyingBalloon，在窗体中添加标签控件。

（2）编写内部类 Fly，它实现了 Runnable 接口。在 run()方法中，使用死循环不断地更新标签的位置。每次更新线程要休眠 0.01 秒钟，这样看起来就是动态更新的效果。由于标签移动的速度是相同的，所有气球的移动速度也是相同的。关键代码如下：

```
private class Fly implements Runnable {
    @Override
    public void run() {
        for (int i = 0; i < 500; i++) {
            try {
                Thread.sleep(10);                          // 线程休眠 0.01 秒
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            balloonLabel1.setLocation(10, 500 - i);        // 改变第 1 个气球的位置
            balloonLabel2.setLocation(210, 500 - i);       // 改变第 2 个气球的位置
            balloonLabel3.setLocation(410, 500 - i);       // 改变第 3 个气球的位置
        }
    }
}
```

DIY：修改气球的上升速度，使 3 个气球的上升速度比满足 1:2:3。（20 分）（实例位置：光盘\mr\ym\11.20\diy）

11.6.5　情景应用 5——动态绘制圆周

视频讲解：光盘\mr\lx\11\动态绘制圆周.exe

实例位置：光盘\mr\ym\11.21\src

本实例使用多线程技术来实现一个绘制圆周的动画。其核心思想是在窗体上确定一个圆心，以 100 像素为半径做圆。将圆周分成 360 份，然后不断地使用直线连接圆周上的点，这样就完成了动态绘制圆周的效果。实例的运行效果如图 11.24 所示。

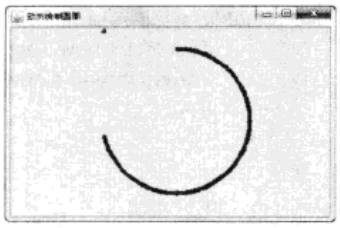


图 11.24　实例运行效果

实现过程如下：

（1）在项目中新建窗体类 CircleDrawing。

（2）编写内部类 Draw，它实现了 Runnable 接口。在 run()方法中，使用 for 循环不断连接圆周上相邻的两个点。每次更新线程要休眠 0.01 秒钟，这样看起来就是动态更新的效果。关键代码如下：

```
private class Draw implements Runnable {
    @Override
    public void run() {
        for (int i = 0; i < 360; i++) {
            try {
                Thread.sleep(10);                                          // 线程休眠 0.01 秒
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            int startX = (int) (225 + 100 * Math.sin(Math.PI / 180 * i));          // 计算起点横坐标
            int startY = (int) (130 - 100 * Math.cos(Math.PI / 180 * i));          // 计算起点纵坐标
            int endX = (int) (225 + 100 * Math.sin(Math.PI / 180 * (i + 1)));      // 计算终点横坐标
            int endY = (int) (130 - 100 * Math.cos(Math.PI / 180 * (i + 1)));      // 计算终点纵坐标
            Point start = new Point(startX, startY);                               // 获得起点坐标
            Point end = new Point(endX, endY);                                     // 获得终点坐标
            drawLine(start, end);                                                  // 绘制直线
        }
    }
}
```

DIY：使用两个线程从圆周上两个相对的点顺时针绘制圆周。（20 分）（实例位置：光盘\mr\ym\11.21\diy）

情景应用 DIY 栏目分数统计：

DIY 题目	1	2	3	4	5	总分数	
分数							

11.7　自 我 测 试

一、选择题（每题 10 分，5 道题）

1．当线程调用 wait()方法处于等待状态时，哪些（个）方法可以让其恢复运行？（　　）

A．join()方法　　B．run()方法　　C．notify()方法　　D．notifyAll()方法

2．对于如下代码，在 A 处使用哪些（个）语句可以在新线程中输出 Hello World？（　　）

```
public class Test implements Runnable {
    public void run() {
        System.out.println("Hello World");
    }
    public static void main(String[] args) {
        Test test = new Test();
        Thread thread = new Thread(test);
        // A
    }
}
```

A．test.run();　　B．thread.run();　　C．thread.start();　　D．System.out.println("Hello World");

3．当编译和运行下列代码时会发生什么？（　　）

```
public class Test extends Thread {
    public void start() {
        for (int i = 0; i < 10; i++) {
            System.out.print(i + " ");
        }
    }
    public static void main(String[] args) {
        Test test = new Test();
        test.run();
    }
}
```

A．编译错误，因为 Test 类中没有重写 run()方法

B．运行错误，因为 Test 类中没有重写 run()方法

C．正常运行，输出 0 1 2 3 4 5 6 7 8 9

D．正常运行，输出为空

4．下列关于线程的哪些（个）说法是正确的？（　）

A．创建新线程只能通过继承 Thread 类

B．线程的名称可以用来区别线程

C．当仅有守护线程存在时，程序会退出执行

D．使用同步机制可以避免不同线程对同一数据修改时可能发生的数据损毁

5．如果下列代码能够正常运行，关于 RunHandler 类的哪些（个）说法是正确的？（　）

```
public class Test {
    public static void main(String[] args) {
        Thread thread = new Thread(new RunHandler());
    }
}
```

A．RunHandler 类必须实现 Runnable 接口

B．RunHandler 类必须继承 Thread 类

C．RunHandler 类必须编写访问权限修饰符为 public、返回值为 void、参数为空的 run()方法

D．RunHandler 类必须编写 init()方法

二、填空题（每题 10 分，5 道题）

1．使用 Thread 类的哪个方法可以启动新线程？_____

2．在 Runnable 接口中定义的方法是？_____

3．编写同步方法时需要使用哪个关键字？_____

4．使用 Thread 类的哪个方法可以让线程进入休眠状态并在一定的时间后自动唤醒？_____

5．查看线程名称的方法是？_____

测试分数统计：

类别	第 1 题	第 2 题	第 3 题	第 4 题	第 5 题
选择题分数					
填空题分数					
总分数					

11.8　行 动 指 南

开始日期：_____年____月____日　　结束日期：_____年____月____日

序号	内　容	行 动 指 南	
1	照猫画虎栏目	分数>75 分	优秀，基本功掌握得不错，加油！
		75 分>分数>50 分	及格，知识掌握得不牢，重新做一遍照猫画虎。
	分数（　　）	分数<50 分	请使用光盘中提供的“实战能力测试系统”进行提高训练。
	情景应用栏目	分数>75 分	优秀，综合应用能力很强。
		75 分>分数>50 分	及格，综合应用能力需提高，再练一遍情景应用。
	分数（　　）	分数<50 分	请使用光盘中提供的“实战能力测试系统”进行提高训练。
	自我测试栏目	分数>75 分	优秀，有成为编程高手的潜质。
	分数（　　）	分数<75 分	请使用光盘中提供的“实战能力测试系统”进行提高训练。
	综合评价	反复训练后，以上各项分数都在 75 分以上，方可进入下一堂课学习。	
2	编程实践能力培养：本栏目提供了一些实践题目，对于培养编程能力很有效，要做好。	（1）查看 Java 虚拟机中的全部线程。	
		（2）编写一个死锁程序。	
		（3）编程模拟在线交易系统。	
		（4）创建一个线程池并对比线程池和线程的性能差异	
3	编程习惯培养：本堂课培养读者在学习开发中记笔记的习惯，把开发中遇到的问题、总结的经验记录下来。		
4	创新能力培养：看看身边有没有可以用编程解决的问题，记录到右边的表格中。根据学习进度，尝试编写解决这些问题的小程序。		

11.9 成功可以复制——初中站长的创业故事

1979 年，李兴平出生于广东兴宁县一个农民家庭。1999 年，网络在中国逐渐普及，兴宁县也开始有了网吧，李兴平开始迷上了上网。初中毕业后，在当地网吧找了一份网吧管理员的工作。因为要帮人攒电脑赚钱，他需要用网络查询配件报价之类的信息，但很快发觉在网上找资料非常困难，当时的中文网站不仅内容不够丰富、数量有限，而且要把那些用英文字母表示的网址一个个记下来，并不是一件容易的事情。他利用平时学到的网页制作知识设计了一个个人网页，把他认为好的网站搜集在一起，并和它们建立链接。后来，李兴平的电脑知识越来越丰富，也能设计出更复杂的网页，但是他发现其实静态页面简单又快捷，对于个人网站，静态页比动态页更实用。

为了让网页方便使用，他把当时中国排名前 5000 位的站点进行分门别类，做了一个“网址大全”式的导航网页，并赋予一个“其貌不扬”的名字“Hao123”。因为网站一直只有李兴平一个人在管理，他的工作量也很大，一天大概有 6～7 个小时是在检查、更新链接。但链接太多，不可能每天都全部检查，因此他经常担心某些网站含有非法链接，压力和工作量都很大。功夫不负有心人，因简单、易用、精确，Hao123 网站逐渐被广大网民所认识，并被大部分普通网民设为首页。2003 年，Hao123 的网址导航站点已经占领了中国网吧 50%以上的首页，这简陋得让精英们不屑一顾的 Hao123.com，也慢慢成为全球流量最大的中文网站之一。

为了让网站能有些收入，他开始做一些点击广告以及网站联盟等。随着流量增大，网站有了固定收入。最初的广告费一个月只有几百元，后来逐渐增加到 1000 元，又到 4000 元。到 2001 年，1 天竟能达到几万元。2003 年后，每个月的收入稳定在 80 万元左右。

现在，Hao123 已成为中国网民上网的第一门户和互联网网址导航的第一品牌。李兴平也成为中国网络界当之无愧的传奇人物！

经典语录

我天天泡在网吧里，我知道他们喜欢哪样的网站，讨厌哪些东西。我首先是一个互联网的热爱者，其次才是一个商人。

深度评价

李兴平的成功也许会让那些怀揣宏大的“商业计划书”以及数千万美元的回国创业的海龟们愤愤不平，凭什么这样一个既不懂技术，又不懂管理，不会讲洋文更没有强大资金背景支持的“土八路”赢得了这场战争？Hao123 的成功真正说明，技术并不是最重要的，重要的是你是否能“幸福着用户的幸福，痛苦着用户的痛苦”，重要的是你能否对所做的事业极其无限热爱和超级专注。

第12堂课

事件处理的应用

（视频讲解：77分钟）

Swing设计的GUI控件非常灵活，完全掌握后，可以设计出任何样式的GUI界面。但是不能和用户交互的界面是没有任何作用的，GUI程序依靠事件处理机制来和用户交互，用户在GUI界面上所做的任何操作都是事件，如键盘输入、选择某个单选项，就连移动鼠标等操作都会被GUI当作事件捕获，并处理该事件。本堂课将介绍应用程序编程常用的事件处理方法，让读者设计的Swing程序拥有交互能力。

学习摘要：

- GUI事件的简介
- 事件监听器的使用
- 事件适配器原理
- 窗体事件处理的方法
- 鼠标事件处理的方法
- 键盘事件处理的方法

12.1 什么是 GUI 事件

GUI 事件处理机制是建立交互式应用程序的关键技术，其中事件是用户在程序界面上的各种操作，如鼠标单击、双击、移动、按下键盘上的键、抬起、关闭窗体、输入文本等。而各种事件的业务处理由相应的监听器来完成，例如，用户在窗体上单击鼠标按钮，将会有指定的鼠标事件监听器来捕获这个操作，并做出相应的业务处理。

12.1.1 为程序添加交互能力

为了使读者能够更好地了解 Swing 的事件处理机制，先来做一个简单的实例，从该实例中了解事件监听器是如何处理用户操作的。

例 12.01 下面通过一个简单的实例演示如何监听文本框输入事件，代码如下：**（实例位置：光盘\mr\ym\12.01）**

```
public class EventDemo extends JFrame {
    private static final long serialVersionUID = 3944769904218849897L;
    private JPanel contentPane;
    private JTextField textField1;
    private JTextField textField2;
    private JLabel label1;
    private JLabel label2;
    public static void main(String[] args) {
        try {                                                          // 设置程序的外观为 Nimbus
            UIManager.setLookAndFeel("com.sun.java.swing.plaf.nimbus.NimbusLookAndFeel");
        } catch (Throwable e) {
            e.printStackTrace();
        }
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                try {
                    EventDemo frame = new EventDemo();                 // 创建窗体
                    frame.setVisible(true);                            // 设置窗体可见
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }
    public EventDemo() {
        setTitle("监听文本框输入事件");                                  // 设置窗体的标题
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);                // 设置窗体退出时操作
        setBounds(100, 100, 450, 150);                                 // 设置窗体位置和大小
        contentPane = new JPanel();                                    // 创建内容面板
        contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));            // 设置面板的边框
        setContentPane(contentPane);                                   // 应用内容面板
        contentPane.setLayout(new GridLayout(2, 1, 5, 5));             // 为内容面板使用网格布局
```

```
        JPanel panel1 = new JPanel();                                    // 新建面板用来组织标签和文本框
        contentPane.add(panel1);                                         // 应用面板
        label1 = new JLabel("第一个文本框：");                            // 新建标签
        panel1.add(label1);                                              // 应用标签
        textField1 = new JTextField();                                   // 新建文本框
        textField1.addKeyListener(new KeyAdapter() {                     // 为文本框增加按键释放事件的监听
                @Override
                public void keyReleased(KeyEvent e) {
                    do_textField1_keyReleased(e);
                }
            });
        panel1.add(textField1);                                          // 应用文本框
        textField1.setColumns(18);                                       // 将文本框设置成 18 列
        JPanel panel2 = new JPanel();                                    // 新建面板用来组织标签和文本框
        contentPane.add(panel2);                                         // 应用面板
        label2 = new JLabel("第二个文本框：");                            // 新建标签
        panel2.add(label2);                                              // 应用标签
        textField2 = new JTextField();                                   // 新建文本框
        panel2.add(textField2);                                          // 应用文本框
        textField2.setColumns(18);                                       // 将文本框设置成 18 列
    }
    protected void do_textField1_keyReleased(KeyEvent e) {
        String text = textField1.getText();                              // 获得第一个文本框中输入的文本
        textField2.setText(text);                                        // 将文本在第二个文本框中显示
    }
}
```

实例运行效果如图 12.1 所示。

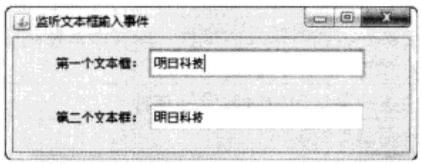


图 12.1　实例运行效果

实例中的界面布局与控件创建已经在第 10 章讲解过了，这里要讲解的重点是如何为控件添加事件监听器。每个控件都是 java.awt.Component 类的子类，它们都继承了 Component 类的方法，其中 addKeyListener() 方法用于向控件注册按键事件的监听器，当用户在控件上按下键盘上的按键时，将产生 KeyEvent 类型的事件对象，这个事件对象将被注册到控件上的监听器捕获，并由监听器指定的方法来处理。

12.1.2　GUI 事件类

所谓事件，就是程序运行过程中可能发生的各种情况，如窗体最小化、窗体最大化、关闭窗体、调整窗体大小、按下键盘上的键、释放键盘上的键以及单击鼠标按钮等，这些都属于事件（Event），事件在 Java 语言中也是一种对象，它们都是 java.awt.AwtEvent 类的直接子类或间接子类，AWTEvent 类的层次结构如图 12.2 所示。

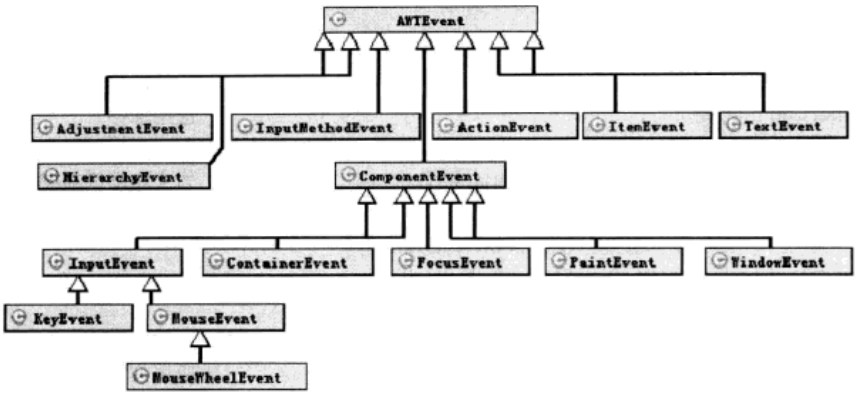


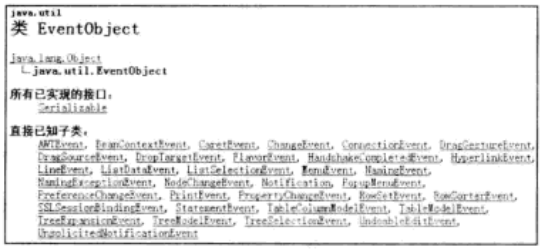
图 12.2　AWTEvent 事件类的层次结构图

常用 AWT 事件类及说明如表 12.1 所示。

表 12.1　AWT 常用事件类及说明

事 件 类	说　明
ActionEvent	动作事件，在用户单击按钮、选择菜单和列表等操作时产生该事件
AdjustmentEvent	调整事件，在滚动条变化时产生该事件
ComponentEvent	控件事件，在控件隐藏、显示、移动等操作中产生
ContainerEvent	容器事件，在容器发生变化时产生该事件，如向容器中添加控件、从容器中移除控件等
FocusEvent	焦点事件，当控件获得焦点与失去焦点时，触发该事件
HierarchyEvent	层次事件，当控件级别改变时触发的事件
InputMethodEvent	输入法事件，输入法改变时触发的事件
ItemEvent	选项事件，复选框控件、复选框菜单项、列表、下拉选择框等控件的数据改变时触发的事件
KeyEvent	按键事件，按下键盘上按键或释放键盘上按键时触发的事件
MouseEvent	鼠标事件，当改变鼠标状态时触发的事件，如单击鼠标按键、鼠标按键的按下与抬起状态、鼠标进入或离开控件范围等
MouseWheelEvent	鼠标滚轮事件，当滚动鼠标滚轮时触发的事件
PaintEvent	绘图事件，发生绘图事件时触发
TextEvent	文本事件，当文本框和文本域控件的内容发生改变时触发该事件
WindowEvent	窗口事件，当窗体被打开、关闭、最大化、最小化、还原和激活时触发该事件

AWTEvent 类继承于 EventObject 类，同时继承该类的还有很多事件类型，读者可参考 API 帮助文档中有关 EventObject 类的介绍，如图 12.3 所示。


java.util
类 EventObject

java.lang.Object
└ java.util.EventObject

所有已实现的接口：
Serializable

直接已知子类：
AWTEvent, BeanContextEvent, CaretEvent, ChangeEvent, ConnectionEvent, DragGestureEvent, DragSourceEvent, DropTargetEvent, FlavorEvent, HandshakeCompletedEvent, HyperlinkEvent, LineEvent, ListDataEvent, ListSelectionEvent, MenuEvent, NamingEvent, NamingExceptionEvent, NodeChangeEvent, Notification, PopupMenuEvent, PreferenceChangeEvent, PrintEvent, PropertyChangeEvent, RowSetEvent, RowSorterEvent, SQLSessionBindingEvent, StatementEvent, TableColumnModelEvent, TableModelEvent, TreeExpansionEvent, TreeModelEvent, TreeSelectionEvent, UndoableEditEvent, UnpublishedNotificationEvent

图 12.3　EventObject 类的有关介绍

12.1.3　事件监听器

事件监听器用于监听指定的事件类型，它们是 Swing 定义的不同的接口的实现对象。例如，KeyListener 接口是键盘按键事件 KeyEvent 的事件监听器接口，必须通过该接口编写自己的监听器实现类来处理事件，监听器类接收到事件之后，将委托指定的方法来执行事件处理，其流程如图 12.4 所示。

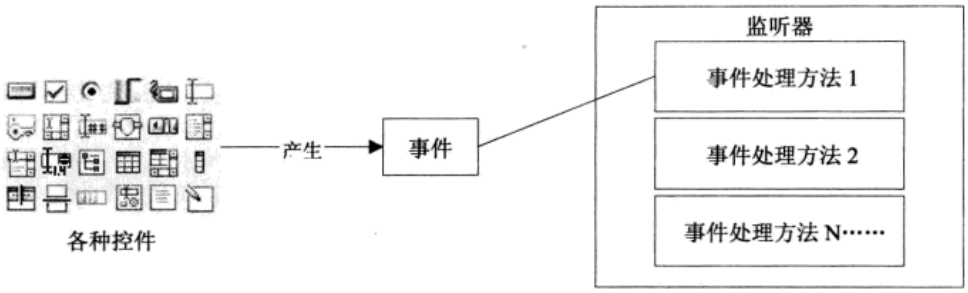


图 12.4　监听器类的工作流程

从该流程图中可以看出，各种类型的控件产生不同的事件对象，这些事件对象由指定的监听器捕获，并调用指定事件类型的处理方法来处理事件。

有的事件监听器接口中定义了多个方法，在实现这样的监听器接口时就必须实现接口中的所有方法，例如，KeyListener 接口的定义代码如下：

```
public interface KeyListener extends EventListener {
    public void keyTyped(KeyEvent e);
    public void keyPressed(KeyEvent e);
    public void keyReleased(KeyEvent e);
}
```

直接使用接口需要实现接口中的全部方法，如果仅对其中某个方法感兴趣，则需要添加其他方法的空实现。解决该问题最好的办法就是使用适配器。

12.1.4　适配器

适配器是事件监听接口的实现类，适配器实现了相应的事件监听器接口，并为接口中的所有方法定义了空的实现方法，也就是声明了接口中定义的所有方法，但是方法体是空的。该类在定义自己的事件监听器时，可以继承相应的适配器类，然后重写需要的事件方法就可以了。

另外，由于适配器中提供的都是监听器接口定义方法的空实现，此时实例化适配器的实例对象没有任何意义，所以适配器被定义为 abstract 抽象类，但是其中的事件方法都不是抽象的。这样可以有选择地重写指定方法。例如，按键事件监听器的适配器为 KeyAdapter 类，该类的定义代码如下：

```
public abstract class KeyAdapter implements KeyListener {
    public void keyTyped(KeyEvent e) {}
    public void keyPressed(KeyEvent e) {}
    public void keyReleased(KeyEvent e) {}
}
```

这里每个方法的声明格式都带有空的花括号{}，即方法体为空。在重写某个方法时，要在方法体中编写指定的事件处理代码。在接下来的章节中会讲解各种常用的事件处理方法。

12.2 窗 体 事 件

程序开发离不开窗体，所有控件都要放在窗体中呈现给用户。那么，对应于窗体的事件处理是必不可少的，窗体在系统中有不同的状态，例如，打开窗体时处于激活状态，当选择其他窗体时，该窗体又处于未激活状态。当单击窗体的最小化按钮、关闭按钮时将会触发相应的事件方法，下面将以实例说明窗体事件监听器是如何使用的。

例 12.02 创建一个窗体，并添加窗体事件监听器，当窗体关闭时显示提示信息。这是通过 do_this_windowClosing()方法实现的，关键代码如下：**（实例位置：光盘\mr\ym\12.02）**

```
public FrameEventDemo() {
    addWindowListener(new WindowAdapter() {                    // 为窗体关闭中事件增加监听
        @Override
        public void windowClosing(WindowEvent e) {
            do_this_windowClosing(e);
        }
    });
    setTitle("关闭窗体提示");                                   // 设置窗体的标题
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);            // 设置窗体退出时操作
    setBounds(100, 100, 450, 300);                             // 设置窗体位置和大小
    contentPane = new JPanel();                                // 创建内容面板
    contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));        // 设置面板的边框
    contentPane.setLayout(new BorderLayout(0, 0));             // 为内容面板使用边界布局
    setContentPane(contentPane);                               // 应用内容面板
}

protected void do_this_windowClosing(WindowEvent e) {
    JOptionPane.showMessageDialog(this, "确认关闭？", null, JOptionPane.CLOSED_OPTION);
}
```

实例运行效果如图 12.5 所示。

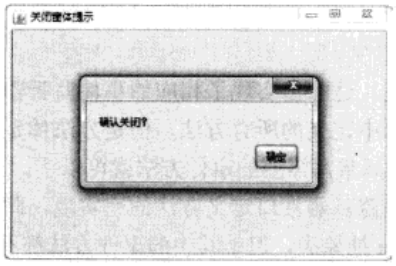


图 12.5 实例运行效果

下面的实例将窗体关闭方式设置为 DO_NOTHING_ON_CLOSE，这样窗体就不会对用户单击窗体关闭按钮做出响应，此时可以在实例中通过窗体关闭事件执行 System 类的 exit()方法退出当前 Java 程序，达到关闭程序的目的。

例 12.03 创建一个窗体，设置窗体的关闭方式为 DO_NOTHING_ON_CLOSE，然后在窗体关闭事件中退出程序。关键代码如下：**（实例位置：光盘\mr\ym\12.03）**

```
protected void do_this_windowClosing(WindowEvent e) {
    System.out.println("你单击了窗体的关闭按钮。");
    System.out.println("在这个事件处理方法中可以再干点别的。");
    System.out.println("例如，关闭数据库连接等。");
    System.out.println("好了，不说了，Bye");
    System.exit(0);
}
```

实例运行效果如图 12.6 所示。

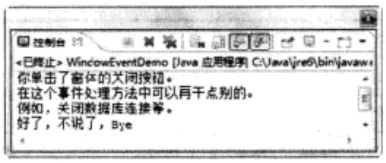


图 12.6　实例运行效果

注意：执行 System 类的 exit()方法退出当前 Java 程序和使用 EXIT_ON_CLOSE 方式通过关闭按钮关闭窗体是不一样的，后者只是关闭窗体，如果当前窗体是程序的一个模块，那么不会影响到程序的继续执行，而前者将退出整个程序。

12.3　掌握鼠标操作

鼠标操作是图形操作系统最常用的操作之一，用户通过鼠标的移动、单击、拖动等操作可以在程序中实现各种交互效果，如拖动窗体、单击按钮、选择文本框等。本节主要介绍鼠标各种事件的处理方法。

12.3.1　鼠标事件监听器

鼠标事件监听器由 MouseListener 接口和 MouseMotionListener 接口定义，它们分别定义了捕获不同的鼠标操作的方法。其中 MouseListener 接口中定义的 5 个方法及说明如表 12.2 所示。

表 12.2　MouseListener 监听器方法及说明

方　法	说　明
mouseClicked (MouseEvent e)	处理鼠标单击事件的方法
mouseEntered (MouseEvent e)	鼠标进入控件区域时执行的方法
mouseExited (MouseEvent e)	鼠标离开控件区域时执行的方法
mousePressed (MouseEvent e)	按下鼠标按键时执行的方法
mouseReleased (MouseEvent e)	释放鼠标按键时执行的方法

MouseListener 接口中的方法基本能够满足大多数应用程序的事件处理，MouseMotionListener 接口中定义了两个有关鼠标移动与鼠标拖动事件的处理方法，这两个方法及说明如表 12.3 所示。

表 12.3　MouseMotionListener 监听器方法及说明

方　法	说　明
mouseMoved (MouseEvent e)	处理鼠标移动事件的方法
mouseDragged (MouseEvent e)	处理鼠标拖动事件的方法

12.3.2 鼠标事件处理

两个鼠标事件监听器中的方法都定义了 MouseEvent 类型的形参，MouseEvent 类是鼠标事件类，是被监听器捕获的用户操作所生成的事件对象，该实例对象包含了许多鼠标事件发生时的参数信息，如鼠标的坐标位置、鼠标的按键等。这些属性可以通过相应的访问器方法获取，鼠标事件的常用方法如表 12.4 所示。

表 12.4 MouseEvent 类常用方法及说明

方 法	说 明
getButton()	返回更改了状态的鼠标按键（如果有）
getClickCount()	返回与此事件关联的鼠标单击次数
getLocationOnScreen()	返回鼠标相对于屏幕的绝对 x, y 坐标
getPoint()	返回事件相对于源控件的 x, y 坐标
translatePoint(int x, int y)	通过将事件坐标加上指定的 x（水平）和 y（垂直）偏移量，将事件的坐标平移到新位置

下面以实例分别演示这两个接口中事件处理方法的执行情况，读者可以在实例中做与鼠标有关的各种操作，充分了解自己开发的程序能够处理哪些与鼠标有关的操作事件。

例 12.04 创建一个窗体，为窗体添加 MouseListener 事件监听器，当用户在窗体上操作鼠标时，程序能够在文本框中输出各种鼠标的操作信息。实例的关键代码如下：**（实例位置：光盘\mr\ym\12.04）**

```
protected void do_panel_mouseClicked(MouseEvent e) {
    label.setText("发生鼠标单击事件");
}
protected void do_panel_mouseEntered(MouseEvent e) {
    label.setText("发生鼠标进入事件");
}
protected void do_panel_mouseExited(MouseEvent e) {
    label.setText("发生鼠标离开事件");
}
protected void do_panel_mousePressed(MouseEvent e) {
    label.setText("发生鼠标按下事件");
}
protected void do_panel_mouseReleased(MouseEvent e) {
    label.setText("发生鼠标释放事件");
}
```

实例运行效果如图 12.7 所示。

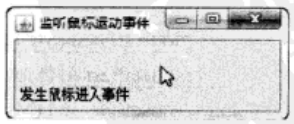


图 12.7 实例运行效果

例 12.05 创建一个窗体，为该窗体添加 MouseMotionListener 事件监听器，当用户在窗体上移动和拖动文本框控件时，程序能够显示鼠标位置或移动文本框位置。实例的关键代码如下：**（实例位置：光盘\mr\ym\12.05）**

```
protected void do_panel_mouseDragged(MouseEvent e) {
    Point point = e.getPoint();                          // 获得鼠标的坐标
    label.setText("鼠标位置：x = " + point.x + ",y = " + point.y);   // 在标签上显示鼠标的坐标
}
```

实例运行效果如图 12.8 所示。

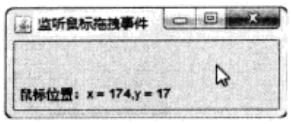


图 12.8　实例运行效果

12.4　掌握键盘控制

键盘控制与鼠标控制都是应用程序不可或缺的输入控制，其中键盘控制包括获取用户的按键、接收用户输入的文本等操作。本节将介绍 Java 程序如何处理键盘操作。

12.4.1　键盘事件监听器

键盘事件监听器由 KeyListener 接口定义，编写键盘事件监听器必须实现该接口中的 3 个方法，这 3 个方法分别处理不同的按键事件，这 3 个方法及其说明如表 12.5 所示。

表 12.5　KeyListener 事件监听器方法及说明

方　法	说　明
keyPressed (KeyEvent e)	按下某个按键时调用此方法
keyReleased (KeyEvent e)	释放某个按键时调用此方法
keyTyped (KeyEvent e)	输入某个按键时调用此方法

对应于键盘事件监听器的适配器是 KeyAdapter 类，实际运用可以继承该类，然后重写需要的单个方法。如果需要处理所有键盘事件的监听，那么干脆实现 KeyListener 接口，总之需要根据程序的需求选择实现 KeyListener 接口或继承 KeyAdapter 抽象类。

例 12.06　创建一个窗体，为该窗体添加 KeyListener 事件监听器，当用户在窗体按下键盘按键时，控制台输出用户的按键动作。实例的关键代码如下：**（实例位置：光盘\mr\ym\12.06）**

```
protected void do_this_keyPressed(KeyEvent e) {
    label.setText("按下" + e.getKeyChar() + "键");
}
protected void do_this_keyReleased(KeyEvent e) {
    label.setText("释放" + e.getKeyChar() + "键");
}
protected void do_this_keyTyped(KeyEvent e) {
    label.setText("输入" + e.getKeyChar() + "键");
}
```

实例运行效果如图 12.9 所示。

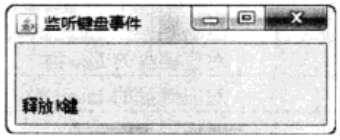


图 12.9　实例运行效果

12.4.2 键盘事件处理

键盘事件监听器中的每个事件处理方法都接收 KeyEvent 类型的按键事件对象，这个按键事件是用户在控件上的按键动作促使程序产生的，键盘事件监听器将捕获该事件，并根据不同的按键动作委托不同的事件处理方法去处理该事件，按键事件对象保存了用户按键时的相关属性，如按键的代码、按键对应的字符、按键的名称等，这些属性可以通过相应的属性设置器与属性获取器方法实现，如表 12.6 所示。

表 12.6 KeyEvent 事件监听器方法及说明

方 法	返回值	说 明
getKeyChar()	char	返回与此事件中的键关联的字符
getKeyCode()	int	返回与此事件中的键关联的整数 keyCode
getKeyModifiersText(int modifiers)	String	返回描述修改键的 String，如 Shift 或 Shift+Ctrl
getKeyText(int keyCode)	String	返回描述 keyCode 的 String，如 HOME、F1 或 A
isActionKey()	boolean	返回此事件中的键是否为“动作”键
setKeyChar(char keyChar)	void	设置 keyCode 值，以表示某个逻辑字符
setKeyCode(int keyCode)	void	设置 keyCode 值，以表示某个物理键

除了按键事件常用的方法之外，KeyEvent 类还定义了按键常量，这些常量分别对应着按键的键值，常量名称以“VK_”为前缀。例如，VK_A 常量代表按键 A，它的键值是 65，常用的常量如表 12.7 所示。

表 12.7 KeyEvent 事件类对应按键的常量及说明

常 量	说 明
VK_A～VK_Z	对应键盘的 A～Z 按键
VK_0～VK_9	对应键盘的 0～9 按键
VK_F1～VK_F24	对应键盘的 F1～F24 按键，普通键盘支持到 F12
VK_NUMPAD0～VK_NUMPAD9	对应小键盘的 0～9 按键
VK_ALT	对应键盘的 Alt 键
VK_BACK_SLASH	对应键盘的反斜杠“\”键
VK_CAPS_LOCK	对应键盘的 CapsLock 键，控制键盘上的大写状态指示灯
VK_CLOSE_BRACKET	对应键盘的右方括号“]”键
VK_COMMA	对应键盘的逗号“，”键
VK_CONTEXT_MENU	对应键盘的 Microsoft Windows 上下文菜单键，用于弹出菜单
VK_CONTROL	对应键盘的 Ctrl 键
VK_DELETE	对应键盘的 Delete 键
VK_END	对应键盘的 End 键
VK_ENTER	对应键盘的 Enter 键
VK_EQUALS	对应键盘的“=”键
VK_ESCAPE	对应键盘的 Esc 键
VK_INSERT	对应键盘的 Insert 键
VK_LEFT	对应键盘的左方向键
VK_MINUS	对应键盘的减号“-”键
VK_NUM_LOCK	对应键盘的 Num Lock 键

续表

常　量	说　明
VK_OPEN_BRACKET	对应键盘的左方括号“[”键
VK_PAGE_DOWN	对应键盘的向下翻页 Page Down 键
VK_PAGE_UP	对应键盘的向上翻页 Page Up 键
VK_PAUSE	对应键盘的 Pause Break 键
VK_PERIOD	对应键盘的句点“.”键
VK_RIGHT	对应键盘的右方向键
VK_SCROLL_LOCK	对应键盘的 Scroll Lock 键
VK_SEMICOLON	对应键盘的分号“;”键
VK_SHIFT	对应键盘的 Shift 键
VK_SLASH	对应键盘的正斜杠“/”键
VK_SPACE	对应键盘的空格键
VK_UNDEFINED	此值用于指出 keyCode 是未知的
VK_WINDOWS	对应键盘的 Windows 键

例 12.07　创建一个窗体，在窗体上使用文本框和标签控件组成算术加法运算题，运算的数字是随机生成的，当用户输入运算结果后按 Enter 键完成一道题的运算。实例的关键代码如下：**（实例位置：光盘\mr\ym\12.07）**

```
protected void do_this_windowActivated(WindowEvent e) {
    int random1 = new Random().nextInt(100);                    // 获得 0～99 的随机数
    int random2 = new Random().nextInt(100);                    // 获得 0～99 的随机数
    sum = random1 + random2;                                    // 计算两个随机数的和
    label.setText(random1 + " + " + random2 + " = ");           // 设置标签
}
protected void do_textField_keyPressed(KeyEvent e) {
    if (e.getKeyChar() == KeyEvent.VK_ENTER) {                  // 如果用户输入了 Enter 键
        String text = textField.getText();                      // 获得用于输入的文本
        if (text.isEmpty()) {                                   // 如果用户输入的是空字符串，则进行提示
            JOptionPane.showMessageDialog(this, "请输入您的答案！", null, JOptionPane.INFORMATION_
MESSAGE);
            return;
        }
        int result = -1;
        try {
            result = Integer.parseInt(text);                    // 将字符串转换成整数
        } catch (Exception e1) {
            JOptionPane.showMessageDialog(this, "您输入的答案格式不正确！", null, JOptionPane.WARNING_
MESSAGE);
            return;
        }
        if (result == sum) {                                    // 如果结果正确
            JOptionPane.showMessageDialog(this, "恭喜您回答正确！", null, JOptionPane.INFORMATION_
MESSAGE);
            return;
        } else {
            JOptionPane.showMessageDialog(this, "您输入的答案不正确！", null, JOptionPane.WARNING_
```

```
MESSAGE);
        }
    }
}
```

实例运行效果如图 12.10 所示。

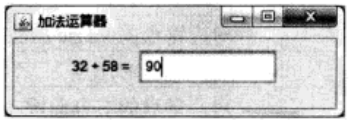


图 12.10　实例运行效果

说明：实例中应用 Integer 类的 parseInt()方法将整数字符串转换成 int 基本类型的整数，该方法要求被转换的字符串对象中只包含整数，否则会抛出 NumberFormatException 类的异常对象，说明数字格式异常。本实例中捕获该异常之后，在标签控件上以文字提示用户重新输入运算结果。

12.5　照猫画虎——基本功训练

12.5.1　基本功训练 1——查看窗体的生命周期

视频讲解：光盘\mr\lx\12\查看窗体的生命周期.exe

实例位置：光盘\mr\ym\12.08\src

在编写 Swing 程序时，需要考虑窗体的生命周期。对于不同的生命周期的窗体，可能需要执行不同的操作。例如，Windows 自带的记事本，如果在其中输入了文字而且用户没有保存，则在关闭时就会提示用户是否保存文本。本实例将编程监听窗体各个不同的生命周期，并在控制台上进行输出，关键代码如下：

```
public WindowEventDemo() {
    addWindowListener(new WindowAdapter() {
        @Override
        public void windowActivated(WindowEvent e) {                    // 为窗体激活事件增加监听
            do_this_windowActivated(e);
        }
        @Override
        public void windowClosed(WindowEvent e) {                       // 为窗体关闭事件增加监听
            do_this_windowClosed(e);
        }
        @Override
        public void windowClosing(WindowEvent e) {                      // 为窗体关闭中事件增加监听
            do_this_windowClosing(e);
        }
        @Override
        public void windowDeactivated(WindowEvent e) {                  // 为窗体停用事件增加监听
            do_this_windowDeactivated(e);
        }
        @Override
        public void windowDeiconified(WindowEvent e) {                  // 为窗体去图标化事件增加监听
            do_this_windowDeiconified(e);
        }
```

```
        @Override
        public void windowIconified(WindowEvent e) {                    // 为窗体图标化事件增加监听
            do_this_windowIconified(e);
        }
        @Override
        public void windowOpened(WindowEvent e) {                       // 为窗体打开事件增加监听
            do_this_windowOpened(e);
        }
    });
    setTitle("监听窗体事件");                                            // 设置窗体的标题
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);                     // 设置窗体退出时操作
    setBounds(100, 100, 450, 300);                                      // 设置窗体位置和大小
    contentPane = new JPanel();                                         // 创建内容面板
    contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));                 // 设置面板的边框
    contentPane.setLayout(new BorderLayout(0, 0));                      // 为内容面板使用边界布局
    setContentPane(contentPane);                                        // 应用内容面板
}
protected void do_this_windowActivated(WindowEvent e) {
    System.out.println("发生窗体激活事件");
}
protected void do_this_windowClosed(WindowEvent e) {
    System.out.println("发生窗体关闭事件");
}
protected void do_this_windowClosing(WindowEvent e) {
    System.out.println("发生窗体关闭中事件");
}
protected void do_this_windowDeactivated(WindowEvent e) {
    System.out.println("发生窗体停用事件");
}
protected void do_this_windowDeiconified(WindowEvent e) {
    System.out.println("发生窗体去图标化事件");
}
protected void do_this_windowIconified(WindowEvent e) {
    System.out.println("发生窗体图标化事件");
}
protected void do_this_windowOpened(WindowEvent e) {
    System.out.println("发生窗体打开事件");
}
```

实例运行效果如图 12.11 所示。

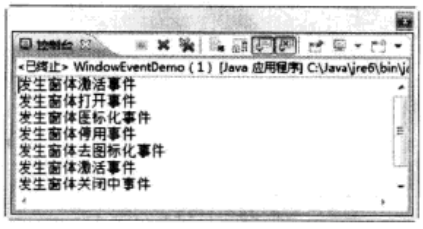


图 12.11　实例运行效果

照猫画虎：监听窗体激活事件，询问用户是否打开上次的文件。（20 分）（实例位置：光盘\mr\ym\12.08\zmhh）

12.5.2 基本功训练 2——修改窗体中的鼠标样式

视频讲解：光盘\mr\lx\12\修改窗体中的鼠标样式.exe

实例位置：光盘\mr\ym\12.09\src

Swing 中的鼠标样式是可以修改的，这和 Windows 系统类似。本实例将演示 4 种常见的样式，包括手型、十字型等，关键代码如下：

```
public CursorDemo() {
    setTitle("显示不同样式的鼠标");                                          // 设置窗体的标题
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);                          // 设置窗体退出时操作
    setBounds(100, 100, 300, 200);                                           // 设置窗体位置和大小
    contentPane = new JPanel();                                              // 创建内容面板
    contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));                      // 设置面板的边框
    setContentPane(contentPane);                                             // 应用内容面板
    contentPane.setLayout(new GridLayout(2, 2, 5, 5));                       // 为内容面板使用网格布局
    JLabel label1 = new JLabel("");                                          // 新建标签
    label1.setCursor(Cursor.getPredefinedCursor(Cursor.CROSSHAIR_CURSOR));   // 更改鼠标在该标签上的样式
    label1.setText("CROSSHAIR_CURSOR");                                      // 显示鼠标的样式
    contentPane.add(label1);                                                 // 应用标签
    JLabel label2 = new JLabel("");                                          // 新建标签
    label2.setCursor(Cursor.getPredefinedCursor(Cursor.TEXT_CURSOR));        // 更改鼠标在该标签上的样式
    label2.setText("TEXT_CURSOR");                                           // 显示鼠标的样式
    contentPane.add(label2);                                                 // 应用标签
    JLabel label3 = new JLabel("");                                          // 新建标签
    label3.setCursor(Cursor.getPredefinedCursor(Cursor.WAIT_CURSOR));        // 更改鼠标在该标签上的样式
    label3.setText("WAIT_CURSOR");                                           // 显示鼠标的样式
    contentPane.add(label3);                                                 // 应用标签
    JLabel label4 = new JLabel("");                                          // 新建标签
    label4.setCursor(Cursor.getPredefinedCursor(Cursor.HAND_CURSOR));        // 更改鼠标在该标签上的样式
    label4.setText("HAND_CURSOR");                                           // 显示鼠标的样式
    contentPane.add(label4);                                                 // 应用标签
}
```

实例运行效果如图 12.12 所示。

图 12.12 实例运行效果

你问我答：Cursor.TEXT_CURSOR 的含义是什么？

Cursor.TEXT_CURSOR 表示鼠标的形状是文本型，读者可以参考在记事本中输入文本时鼠标的形状。Java 的 Cursor 类中还提供了其他类型的鼠标样式，如 DEFAULT_CURSOR 表示默认样式的鼠标等。读者可以参考该类的 API 文档。

照猫画虎：实现自定义鼠标样式的功能。（20 分）（实例位置：光盘\mr\ym\12.09\zmhh）

12.5.3　基本功训练 3——使用鼠标改变窗体的背景色

视频讲解：光盘\mr\lx\12\使用鼠标改变窗体的背景色.exe

实例位置：光盘\mr\ym\12.10\src

使用 Swing 中的事件监听器可以监听用户单击鼠标的事件。本实例将根据用户是否单击鼠标来决定是否更换窗体的背景色，关键代码如下：

```
protected void do_this_mouseClicked(MouseEvent e) {
    int random = new Random().nextInt(color.length);                    // 获得一个随机数
    contentPane.setBackground(color[random]);                           // 更换面板的背景色
}
```

实例运行效果如图 12.13 所示。

图 12.13　实例运行效果

照猫画虎：模仿本实例实现定时更换窗体背景颜色的功能。（20 分）（实例位置：光盘\mr\ym\12.10\zmhh）

12.5.4　基本功训练 4——显示程序运行时间

视频讲解：光盘\mr\lx\12\显示程序运行时间.exe

实例位置：光盘\mr\ym\12.11\src

由于 Swing 中的各种控件都需要放置在窗体中，因此可以使用窗体来记录程序的运行时间。本实例将实现这个功能，关键代码如下：

```
protected void do_this_windowActivated(WindowEvent e) {
    final long time = System.currentTimeMillis();                       // 记录窗体打开的初始时间
    Timer timer = new Timer(1000, new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            long during = System.currentTimeMillis() - time;            // 获得窗体激活时间
            setTitle("窗体已经运行了" + (during / 1000) + "秒");          // 更新窗体的标题
        }
    });
    timer.start();                                                      // 启动 Timer 控件
}
```

实例运行效果如图 12.14 所示。

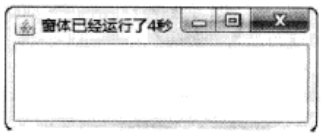


图 12.14　实例运行效果

你问我答：Timer 类的作用是什么？

Java SE API 中一共提供了 3 个 Timer 类，本实例使用的是 javax.swing 包中的 Timer 类。它可以让程序每隔固定时间完成一个操作，其构造方法如下：

```
public Timer(int delay,ActionListener listener)
```

参数说明：delay 是初始延迟和动作事件间延迟的毫秒数。listener 是需要定期执行的操作，可以为空。

照猫画虎：在本实例基础上实现时间的进位功能，例如，让 61 秒显示成 1 分 1 秒。（20 分）（实例位置：光盘\mr\ym\12.11\zmhh）

12.5.5 基本功训练 5——随机更换窗体背景图片

视频讲解：光盘\mr\lx\12\随机更换窗体背景图片.exe

实例位置：光盘\mr\ym\12.12\src

现在许多软件都提供了更换主界面背景图片的功能，此功能可以提高软件的人性化，使软件更加赏心悦目。本实例将实现随机更换主界面的功能，关键代码如下：

```
protected void do_this_windowActivated(WindowEvent e) {
    int random = new Random().nextInt(6) + 1;                                    // 获得一个 1～6 的随机数
    ImageIcon icon = new ImageIcon("src/images/photo" + random + ".jpg");        // 使用随机数获得图标
    setSize(icon.getIconWidth(), icon.getIconHeight() + 30);                     // 将窗体的大小设置成图标的大小
    label.setIcon(icon);                                                         // 为标签应用图标
}
```

实例运行效果如图 12.15 所示。

图 12.15 实例运行效果

你问我答：nextInt(6)方法的作用是什么？

Java SE API 中提供了生成随机数的方法。通常使用 Random 类，因为其功能较为强大。本实例首先创建了一个 Random 类的匿名对象，然后使用 nextInt()方法生成一个随机的整数，参数 6 表示了该随机数的范围，本实例中是 0～5。

照猫画虎：模仿本实例实现按图片的编号顺序依次更换窗体背景图片的功能。（20 分）（实例位置：光盘\mr\ym\12.12\zmhh）

照猫画虎栏目分数统计：

照猫画虎题目	1	2	3	4	5	总分数	
分数							

12.6　情景应用——拓展与实践

12.6.1　情景应用 1——简易配对游戏

视频讲解：光盘\mr\lx\12\简易配对游戏.exe

实例位置：光盘\mr\ym\12.13\src

对于一些简单的小游戏，仅使用鼠标就可以完成。本实例编写一个简易配对游戏，在面板中显示了 5 幅图片和 5 个标签，当把图片和标签上的文本对应起来时，游戏结束。用到的技术主要是鼠标各种事件的监听和处理，实例的运行效果如图 12.16 所示。

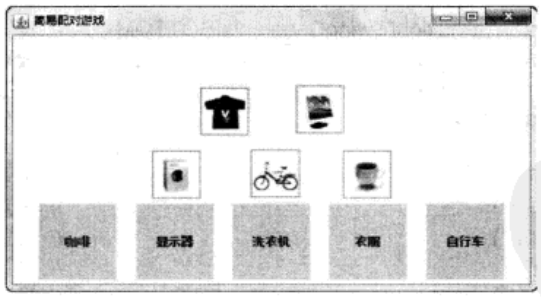


图 12.16　实例运行效果

通过将图片放置到对应的标签上就可以完成游戏，实现过程如下：

（1）创建项目并新建名为 Matching 的窗体。

（2）在窗体中增加图片和标签。其中各个图片先放到标签中，再放置到一个面板中并将该面板应用到玻璃窗格中，这样可以方便地移动图片而不会对窗体中的其他控件产生不良影响。对于标签需要指定标签的大小和背景色，方便玩家放置图片，关键代码如下：

```
public Matching() {
    super();
    getContentPane().setLayout(new BorderLayout());          // 将内容面板设置成边界布局
    setBounds(100, 100, 550, 300);                           // 设置窗体位置和大小
    setTitle("简易配对游戏");                                 // 设置窗体的标题
```

```
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);                    // 设置窗体退出时操作
        final JPanel imagePanel = new JPanel();                            // 创建面板来保存图片
        imagePanel.setLayout(null);                                        // 将面板设置成绝对布局
        imagePanel.setOpaque(false);                                       // 将面板设置成可见
        setGlassPane(imagePanel);                                          // 将面板应用到玻璃窗格
        getGlassPane().setVisible(true);                                   // 设置玻璃窗格可见
        ImageIcon icon[] = new ImageIcon[5];                               // 创建图标数组
        icon[0] = new ImageIcon(getClass().getResource("kafei.png"));      // 为图标数组指定图片
        icon[1] = new ImageIcon(getClass().getResource("xianshiqi.png"));  // 为图标数组指定图片
        icon[2] = new ImageIcon(getClass().getResource("xiyiji.png"));     // 为图标数组指定图片
        icon[3] = new ImageIcon(getClass().getResource("yifu.png"));       // 为图标数组指定图片
        icon[4] = new ImageIcon(getClass().getResource("zixingche.png"));  // 为图标数组指定图片
        final JPanel bottomPanel = new JPanel();                           // 新建面板保存标签
        bottomPanel.setLayout(new FlowLayout(FlowLayout.CENTER, 20, 5));// 将面板设置成流式布局
        getContentPane().add(bottomPanel, BorderLayout.SOUTH);             // 将面板放置在内容面板南部
        for (int i = 0; i < 5; i++) {
            img[i] = new JLabel(icon[i]);                                  // 创建图像标签
            img[i].setSize(50, 50);                                        // 设置标签大小
            img[i].setBorder(new LineBorder(Color.GRAY));                  // 设置线性边框
            int x = (int) (Math.random() * (getWidth() - 50));             // 生成随机坐标
            int y = (int) (Math.random() * (getHeight() - 150));           // 生成随机坐标
            img[i].setLocation(x, y);                                      // 设置随机坐标
            img[i].addMouseListener(this);                                 // 为每个图像标签添加鼠标事件监听器
            img[i].addMouseMotionListener(this);
            imagePanel.add(img[i]);                                        // 添加图像标签到图像面板
            targets[i] = new JLabel();                                     // 创建匹配位置标签
            targets[i].setOpaque(true);                                    // 使标签不透明，以设置背景色
            targets[i].setBackground(Color.ORANGE);                        // 设置标签背景色
            targets[i].setHorizontalTextPosition(SwingConstants.CENTER);   // 设置文本与图像水平居中
            targets[i].setVerticalTextPosition(SwingConstants.BOTTOM);     // 设置文本显示在图像下方
            targets[i].setPreferredSize(new Dimension(80, 80));            // 设置标签首选大小
            targets[i].setHorizontalAlignment(SwingConstants.CENTER);      // 文字居中对齐
            bottomPanel.add(targets[i]);                                   // 添加标签到底部面板
        }
        targets[0].setText("咖啡");                                        // 设置匹配位置的文本
        targets[1].setText("显示器");                                      // 设置匹配位置的文本
        targets[2].setText("洗衣机");                                      // 设置匹配位置的文本
        targets[3].setText("衣服");                                        // 设置匹配位置的文本
        targets[4].setText("自行车");                                      // 设置匹配位置的文本
    }
```

（3）监听鼠标按下和释放事件。当按下鼠标时要保存鼠标的位置。当释放鼠标时，需要检查图片是否放置到了正确的位置。如果全部图片都放置到了正确的位置，则隐藏玻璃窗格并用标签来显示图片，关键代码如下：

```
    public void mousePressed(MouseEvent e) {
        pressPoint = e.getPoint();                                         // 保存拖放图片标签时的起始坐标
    }
    public void mouseReleased(MouseEvent e) {
        if (check()) {                                                     // 如果配对正确
            getGlassPane().setVisible(false);
            for (int i = 0; i < 5; i++) {                                  // 遍历所有匹配位置的标签
                targets[i].setText("配对成功");                            // 设置正确提示
```

```
                targets[i].setIcon(img[i].getIcon());                    // 设置匹配的图标
            }
        }
    }
```

（4）监听鼠标拖拽事件，该事件可以让显示图片的标签随着鼠标的拖拽而不断地改变位置，关键代码如下：

```
public void mouseDragged(MouseEvent e) {
    JLabel source = (JLabel) e.getSource();                              // 获取事件源控件
    Point imgPoint = source.getLocation();                               // 获取控件坐标
    Point point = e.getPoint();                                          // 获取鼠标坐标
    source.setLocation(imgPoint.x + point.x - pressPoint.x, imgPoint.y + point.y - pressPoint.y); // 设置控件新坐标
}
```

（5）判断图片的位置，如果图片配对成功，则将对应的标签背景色设置成绿色；如果配对失败，则不做任何修改。关键代码如下：

```
private boolean check() {
    boolean result = true;
    for (int i = 0; i < 5; i++) {
        Point location = img[i].getLocationOnScreen();                   // 获取每个图像标签的位置
        Point seat = targets[i].getLocationOnScreen();                   // 获取每个对应位置的坐标
        targets[i].setBackground(Color.GREEN);                           // 设置匹配后的颜色
        // 如果配对错误
        if (location.x < seat.x || location.y < seat.y|| location.x > seat.x + 80 || location.y > seat.y + 80) {
            targets[i].setBackground(Color.ORANGE);                      // 回复对应位置的颜色
            result = false;                                              // 检测结果为 false
        }
    }
    return result;                                                       // 返回检测结果
}
```

DIY：在本实例基础上，对图片增加提示功能。当鼠标在图片上放置一定时间后会显示提示文本。（20 分）（实例位置：光盘\mr\ym\12.13\diy）

12.6.2　情景应用 2——模拟相机拍摄

视频讲解：光盘\mr\lx\12\模拟相机拍摄.exe

实例位置：光盘\mr\ym\12.14\src

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。本实例将实现类似的功能，用户可以通过单击方向键来完成移动镜头的操作，如单击→可以让镜头向右移动，实例的运行效果如图 12.17 所示。

实现过程如下：

（1）在 Eclipse 中创建 JFrame 窗体，将其命名为 KeyMoveBackground。

（2）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，关键代码如下：

```
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() {                                 // 为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
}
```

图 12.17　实例运行效果

（3）监听用户单击键盘上方向键事件，通过移动背景图片的标签来表现移动镜头的效果，每次移动 3 个像素，如单击→键可以让保存背景图片的标签向左移动 3 像素，关键代码如下：

```
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()))         // 在不超出屏幕的情况下
                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)                                       // 在不超出屏幕的情况下
                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()))          // 在不超出屏幕的情况下
                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) {                                        // 在不超出屏幕的情况下
                label.setLocation(location.x, location.y + step);        // 向下移动标签
            }
            break;
        default:
            break;
    }
}
```

DIY：在本实例基础上，监听用户按下 Enter 键事件，实现图片的标记功能。（20 分）（实例位置：光盘\mr\ym\12.14\diy）

12.6.3　情景应用 3——打地鼠游戏

视频讲解：光盘\mr\lx\12\打地鼠游戏.exe

实例位置：光盘\mr\ym\12.15\src

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。本实例将完成这个游戏，该游戏的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，实例的运行效果如图 12.18 所示。

图 12.18　实例运行效果

实现过程如下：

（1）在 Eclipse 中创建 JFrame 窗体，将其命名为 Shrewmouse。

（2）在窗体中增加 7 个标签，其中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，关键代码如下：

```
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
        ImageIcon img = new ImageIcon(getClass().getResource("background.jpg")); // 初始化背景图片对象
        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
            mouses[i].setSize(imgMouse.getIconWidth(), imgMouse.getIconHeight());// 设置标签与地鼠图片相同大小
            mouses[i].addMouseListener(new MouseAdapter() {                  // 为标签添加鼠标事件监听适配器
                    @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);                                     // 设置标签的位置
        mouses[1].setLocation(333, 250);                                     // 设置标签的位置
        mouses[2].setLocation(388, 296);                                     // 设置标签的位置
        mouses[3].setLocation(362, 364);                                     // 设置标签的位置
        mouses[4].setLocation(189, 353);                                     // 设置标签的位置
        mouses[5].setLocation(240, 409);                                     // 设置标签的位置
        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
    }
```

（3）编写多线程的核心方法，它可以每隔 1 秒钟随机让一个没有显示图片的标签显示地鼠图片，关键代码如下：

```
    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();
            }
        }
    }
```

DIY：在本实例基础上，记录玩家打地鼠的次数，在玩家关闭窗体时显示。（20 分）（实例位置：光盘\mr\ym\12.15\diy）

12.6.4　情景应用 4——鼠标经过时放大的按钮

视频讲解：光盘\mr\lx\12\鼠标经过时放大的按钮.exe

实例位置：光盘\mr\ym\12.16\src

Swing 支持对鼠标事件的监听，这其中就包括当鼠标进入到一个控件时。通过修改按钮的大小和按钮上字体的大小就可以实现一个鼠标经过时放大的按钮，实例的运行效果如图 12.19 所示。

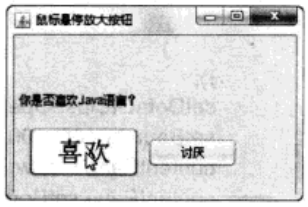


图 12.19　实例运行效果

实现过程如下：

（1）在 Eclipse 中创建 JFrame 窗体，将其命名为 ZoomOutButton。

（2）在窗体中增加按钮和标签，并监听标签上鼠标进入和离开按钮控件事件。对于进入事件，完成将按钮进行放大并将其中文本的字体大小增加 15；对于离开事件，完成将按钮的大小和文本字体复原，关键代码如下：

```
protected void do_button1_mouseEntered(MouseEvent e) {
    rect1 = button1.getBounds();                                                    // 获得按钮边界
    button1.setBounds(rect1.x - 10, rect1.y - 10, rect1.width + 20, rect1.height + 20);   // 设置按钮边界
    font1 = button1.getFont();                                                      // 获得按钮字体
    button1.setFont(new Font(font1.getName(), font1.getStyle(), font1.getSize() + 15));   // 设置按钮字体
}
protected void do_button1_mouseExited(MouseEvent e) {
    button1.setBounds(rect1);                                                       // 设置按钮边界
    button1.setFont(font1);                                                         // 设置按钮字体
}
protected void do_button2_mouseEntered(MouseEvent e) {
    rect2 = button2.getBounds();                                                    // 获得按钮边界
    button2.setBounds(rect2.x - 10, rect2.y - 10, rect2.width + 20, rect2.height + 20);   // 设置按钮边界
    font2 = button1.getFont();                                                      // 获得按钮字体
    button2.setFont(new Font(font2.getName(), font2.getStyle(), font2.getSize() + 15));   // 设置按钮字体
}
protected void do_button2_mouseExited(MouseEvent e) {
    button2.setBounds(rect2);                                                       // 设置按钮边界
    button2.setFont(font2)      ;                                                   // 设置按钮字体
}
```

DIY：实现鼠标悬停时缩小的按钮。（20 分）（实例位置：光盘\mr\ym\12.16\diy）

12.6.5　情景应用 5——迟到的登录按钮

视频讲解：光盘\mr\lx\12\迟到的登录按钮.exe

实例位置：光盘\mr\ym\12.17\src

对于 Swing 中的控件，可以通过修改其显示的位置来实现一些动态效果。如果恰当地控制移动的时间，就可以完成一个迟到的登录按钮。本实例将完成这个功能，实例的运行效果如图 12.20 所示。

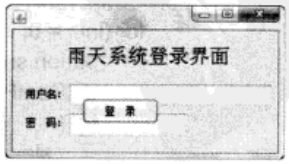


图 12.20　实例运行效果

实现过程如下：

（1）在 Eclipse 中创建 JFrame 窗体，将其命名为 LoginFrame。

（2）在窗体中增加按钮、标签和文本框。为了方便按钮动态显示，窗体使用的是绝对布局。因此不同的控件都需要指定其大小和位置，关键代码如下：

```
public LoginFrame() {
    addWindowListener(new WindowAdapter() {                    // 监听窗体激活事件
        @Override
        public void windowActivated(WindowEvent e) {
            do_this_windowActivated(e);
        }
    });
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);            // 设置窗体退出时操作
    setBounds(100, 100, 329, 185);                             // 设置窗体位置和大小
    contentPane = new JPanel();                                // 创建内容面板
    contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));        // 设置面板的边框
    setContentPane(contentPane);                               // 应用内容面板
    contentPane.setLayout(null);                               // 将内容面板的布局设置成空布局
    JLabel label1 = new JLabel("雨天系统登录界面");              // 创建标签控件
    label1.setHorizontalAlignment(SwingConstants.CENTER);      // 标签文本居中对齐
    label1.setFont(new Font("SansSerif", Font.PLAIN, 24));     // 设置标签控件字体
    label1.setBounds(6, 6, 309, 51);                           // 设置标签的位置和大小
    contentPane.add(label1);                                   // 在面板中增加标签
    JLabel label2 = new JLabel("用户名：");                     // 创建标签控件
    label2.setBounds(16, 69, 55, 18);                          // 设置标签的位置和大小
    contentPane.add(label2);                                   // 在面板中增加标签
    JLabel label3 = new JLabel("密　码：");                     // 创建标签控件
    label3.setBounds(16, 103, 55, 18);                         // 设置标签的位置和大小
    contentPane.add(label3);                                   // 在面板中增加标签
    textField = new JTextField();                              // 创建文本框
    textField.setBounds(65, 63, 242, 30);                      // 设置文本框的位置和大小
    contentPane.add(textField);                                // 在面板中增加文本框
    textField.setColumns(10);                                  // 设置文本框列数
    passwordField = new JPasswordField();                      // 创建密码框
    passwordField.setBounds(65, 99, 143, 30);                  // 设置密码框的位置和大小
    contentPane.add(passwordField);                            // 在面板中增加密码框
    button = new JButton("登　录");                             // 创建登录按钮但没有定位
    contentPane.add(button);                                   // 在面板中增加按钮
}
```

（3）编写方法 do_this_windowActivated()，该方法用来监听窗体激活事件。在该方法中，新建了一个线程来完成按钮的动态移动功能，每次移动之后线程休眠 0.001 秒，关键代码如下：

```
protected void do_this_windowActivated(WindowEvent e) {        // 创建激活事件处理方法
    new Thread() {                                             // 创建匿名线程
        @Override
        public void run() {
            for (int i = 0; i < 217; i++) {                    // 循环控制按钮的移动
                button.setBounds(i, i > 99 ? 99 : i, 90, 30);  // 移动按钮
                getRootPane().setComponentZOrder(button, 0);   // 把按钮置顶显示
                try {
                    sleep(1);                                  // 线程休眠
                } catch (InterruptedException e) {
                    e.printStackTrace();
```

```
                }
            }
        }
    }.start();                                                    // 启动线程
}
```

DIY：将本实例中的控件采用水平移动的方式同时出现。（20 分）（实例位置：光盘\mr\ym\12.17\diy）

情景应用 DIY 栏目分数统计：

DIY 题目	1	2	3	4	5	总分数	
分数							

12.7　自 我 测 试

一、选择题（每题 10 分，5 道题）

1．KeyListener 和 KeyAdapter 之间的关系是？（　　）

A．继承　　B．实现　　C．内部类　　D．无任何关系

2．通过如下代码为窗体激活事件增加监听器。当窗体激活后（　　）。

```
addWindowListener(new WindowAdapter() {
    public void windowActivated(WindowEvent e) {
        System.out.println("窗体激活");
    }
});
```

A．在控制台上输出窗体激活　　B．在控制台上无输出

C．编译错误　　D．运行错误

3．通过如下代码为窗体激活事件增加监听器。当窗体激活后（　　）。

```
addWindowListener(new WindowAdapter() {
    public void WindowActivated(WindowEvent e) {
        System.out.println("窗体激活");
    }
});
```

A．在控制台上输出窗体激活　　B．在控制台上无输出

C．编译错误　　D．运行错误

4．下列哪些（个）方法是在 MouseMotionListener 接口中定义的？（　　）

A．mousePressed()　　B．mouseReleased()　　C．mouseDragged()　　D．mousemoved()

5．使用 MouseEvent 类中的哪个方法可以获得鼠标相对于原控件的坐标？（　　）

A．getPoint()　　B．getPosition()　　C．getLocation()　　D．getXY()

二、填空题（每题 10 分，5 道题）

1．所有事件监听器方法的返回值是？_____

2．WindowListener 接口中定义了几个方法？_____

3．KeyEvent 类中定义了很多类常量和键盘上的按键对应。其中哪个常量和 Enter 键对应？_____

4．Swing 中的鼠标支持各种不同的形状，它们是定义在哪个类中的？_____

5．Swing 中事件的公共父类是？_____

测试分数统计：

类别	第 1 题	第 2 题	第 3 题	第 4 题	第 5 题
选择题分数					
填空题分数					
总分数					

12.8 行 动 指 南

开始日期：_____年____月____日　　结束日期：_____年____月____日

序号	内　容	行 动 指 南	
1	照猫画虎栏目 分数（　　）	分数>75 分	优秀，基本功掌握得不错，加油！
		75 分>分数>50 分	及格，知识掌握得不牢，重新做一遍照猫画虎。
		分数<50 分	请使用光盘中提供的“实战能力测试系统”进行提高训练。
	情景应用栏目 分数（　　）	分数>75 分	优秀，综合应用能力很强。
		75 分>分数>50 分	及格，综合应用能力需提高，再练一遍情景应用。
		分数<50 分	请使用光盘中提供的“实战能力测试系统”进行提高训练。
	自我测试栏目 分数（　　）	分数>75 分	优秀，有成为编程高手的潜质。
		分数<75 分	请使用光盘中提供的“实战能力测试系统”进行提高训练。
	综合评价	反复训练后，以上各项分数都在 75 分以上，方可进入下一堂课学习。	
2	编程实践能力培养：本栏目提供了一些实践题目，对于培养编程能力很有效，要做好。	（1）实现简单的绘图工具。	
		（2）编写键盘记录器。	
		（3）编写打字速度计算工具。	
		（4）编写拼图游戏。	
3	编程习惯培养：本堂课培养读者在学习开发中记笔记的习惯，把开发中遇到的问题、总结的经验记录下来。		
4	创新能力培养：看看身边有没有可以用编程解决的问题，记录到右边的表格中。根据学习进度，尝试编写解决这些问题的小程序。		

12.9　成功可以复制——缔造华人的硅谷传奇杨致远

1994 年 4 月，就读于斯坦福大学的杨致远与费洛为了完成论文，在网络上寻找资料。在这个过程中，他们也收集很多自己感兴趣的站点并将其加入到书签，以便查找。可是当所收集的站点越来越多时，管理和查找就成了大麻烦。于是他们把这些书签按类别整理好，编制成软件《Jerry's Guide to the World Wide Web》（杰瑞全球咨讯网指南），放到网络上让其他冲浪的人享用，结果大受欢迎与拥护。

随着《指南》越来越受欢迎与关注，许多网友纷纷进入斯坦福大学电机系的工作站使用这套软件，使校方大感困扰，抱怨这项发明影响了学校电脑的正常运作。但是杨致远与费洛仍然积极为此努力，并推出特色栏目 Cool Links 和 Hard to Believe，并将网站改名为 Jerry and David's Guide to the World Wide Web。但是由于条件限制，网站的数据和搜索引擎分别放在他们的个人计算机中。

在没日没夜、满目狼藉的工作室中，1994 年夏天的一个晚上，杨致远与费洛为他们的网站取了一个很特殊的名字——Yahoo!，于是神奇的 Yahoo！网站诞生了，当时是半夜 2 点。由于 Yahoo！检索系统实在方便，前景被普遍看好，广告收入也相当乐观。结果，Yahoo！一上市就一鸣惊人，风头大出。

1996 年 4 月 12 日，Yahoo！股票首次上市，每股价为$13，而开盘就达到$24.50，并持续飙升至$43。一下子给 Yahoo！公司带来将近 8.48 亿美金的市场值，创造了纳斯达克记录。1998 年《福布斯》杂志推出高科技百名富翁，杨致远以 10 亿美元的财富跃居第 16 位，超过了冠群 CEO 王嘉廉，成为高科技中的华人首富。1999 年，杨致远的纸面财富更是达到 75 亿美元。

Yahoo!的成功，与杨致远清晰快捷的思维、不知疲倦地为事业打拼是分不开的，在当今 IT 业，有 3 位没有上完大学就跑出去开公司、最后变成著名的亿万富翁的人，他们就是比尔·盖茨、迈克·戴尔和杨致远。

经典语录

如果只是为了成功和金钱去创业，能接受失败吗？不能。怎样才能接受失败？是因为能坚持，对所做事情的热爱，一种固执的“笨”。在创业中，过程始终比终点更为重要。

深度评价

“我不怕输，即使我失败，我也有重新来过的勇气。”杨致远这种不怕输的劲头也衬托出这个传奇人物的精髓所在。引用杨致远的一句话：机会是偶然的，但取得成功并非如此，你把一种嗜好变成了一个事业，这需要巨大的努力。

第13堂课

常用工具类

（视频讲解：93分钟）

程序开发经常需要一些算法，如提取当前系统日期、生成不重复的随机数字、对数字和日期的格式化输出、一些数学函数的运算等，这些都是程序开发中不可避免要使用到的技术。为了提高读者的程序开发能力，为使读者能够独立开发项目，本堂课分别介绍了上述相关算法和技术。

学习摘要：

- 日期时间对象
- 日期与时间的格式化
- 各种数学函数方法的调用
- 两种生成随机数的方法
- 数字的格式化输出

13.1 日期时间类

Date 类用于表示日期时间，它位于 java.util 包中，该类在设计之初没有考虑到国际化，并且现在有很多方法都被标记为过时状态(从 Java 的 API 规范文档中可以看到)，但它还是程序开发不可缺少的常用类之一。本节将介绍 Date 类的创建和常用方法，以及如何格式化为指定格式的字符串。这里不会介绍那些过时的方法，读者学到的将是最新的 API。

13.1.1 创建 Date 类的对象

Date 类最简单的构造方法就是默认的无参数的 Date()构造方法，该方法使用系统中当前的日期和时间创建初始化 Date 类的实例对象，例如：

```
Date now = new Date();
```

Date 类的另一个构造方法是 Date(long date)，该方法接收一个 long 类型的整数来初始化 Date 对象，这个 long 类型的整数是标准基准时间（称为“历元（epoch）”，即 1970 年 1 月 1 日 00:00:00）开始的毫秒数。很多和日期时间有关的类都能够转换为 long 类型整数。System 类的 currentTimeMillis()方法可以获取系统当前时间距历元的毫秒数，例如：

```
long timeMillis = System.currentTimeMillis();
Date date=new Date(timeMillis);
```

13.1.2 比较 Date 对象

Date 类创建的对象代表日期和时间，它们涉及最多的操作就是比较。例如，比较两个人的生日，哪个较早、哪个晚一些，或者两人的生日完全相等。

1. after()方法

该方法用于测试此日期对象是否在指定日期之后，该方法声明如下：

```
public boolean after(Date when)
```

参数说明：when 是要比较的日期对象。当且仅当此 Date 对象表示的时间比 when 表示的时间晚时，才返回 true；否则返回 false。

例 13.01 创建两个不同时间的 Date 对象，判断两个对象表示的时间，谁在前谁在后。实例代码如下：

（实例位置：光盘\mr\ym\13.01）

```
public class DateDemo {
    public static void main(String[] args) {
        Date now = new Date();                                    // 创建当前时间对象
        long tMillis = System.currentTimeMillis() + 5000;         // 当前时间增加 5 秒
        Date otherDate = new Date(tMillis);                       // 创建累加 5 秒后的时间
        if (otherDate.after(now))
            System.out.println("otherDate 对象表示的时间在 now 对象表示的时间之后");
        else
            System.out.println("otherDate 对象表示的时间在 now 对象表示的时间之前");
    }
}
```

实例的运行效果如图 13.1 所示。

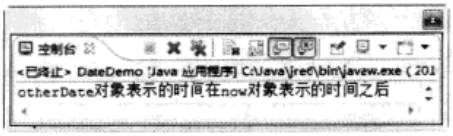


图 13.1　实例运行效果

2．before 方法

该方法用于测试此日期对象是否在指定日期之前，它和 after()方法正好相反，该方法声明如下：

```
public boolean before(Date when)
```

参数说明：when 是要比较的日期对象。当且仅当此 Date 对象表示的时间比 when 表示的时间早时，才返回 true；否则返回 false。例如：

```
boolean result = otherDate.before(now);
```

这段代码的运行结果是，将为 result 变量赋值两个日期对象执行 before()方法的结果。如果把代码放到 after()方法的实例中，那么 result 的变量值为 false。

3．compareTo 方法

该方法用于比较两个日期对象的顺序，该方法常用于多个 Date 对象的排序，该方法声明如下：

```
public int compareTo(Date anotherDate)
```

参数说明：anotherDate 是要比较的其他日期对象。

如果参数 anotherDate 表示的时间等于当前 Date 对象表示的时间，该方法返回值为 0；如果当前 Date 对象表示的时间在 anotherDate 参数表示的时间之前，则返回小于 0 的值；如果当前 Date 对象在 anotherDate 参数表示的时间之后，则返回大于 0 的值。

例 13.02　创建两个不同时间的 Date 对象，比较两个对象，分别输出两个对象的日期是相等、大于还是小于关系。实例代码如下：**（实例位置：光盘\mr\ym\13.02）**

```
public class DateDemo {
    public static void main(String[] args) {
        Date now = new Date();                                          // 创建当前时间对象
        long tMillis = System.currentTimeMillis() - 5000;               // 当前时间减少 5 秒
        Date otherDate = new Date(tMillis);                             // 创建累加 5 秒后的时间
        int compare = otherDate.compareTo(now);
        switch (compare) {
            case 0:
                System.out.println("两个日期对象表示的时间相等");
                break;
            case 1:
                System.out.println("otherDate 对象表示的时间大于 new 对象表示的时间");
                break;
            case -1:
                System.out.println("otherDate 对象表示的时间小于 new 对象表示的时间");
                break;
            default:
                System.out.println(compare);
                break;
        }
    }
}
```

实例的运行效果如图 13.2 所示。

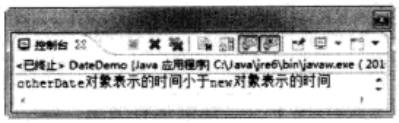


图 13.2　实例运行效果

13.1.3　更改 Date 对象

Date 类的大多数方法已经不推荐使用，其中 getTime()与 setTime()方法被保留了下来，它们分别用于设置和获取 Date 对象的毫秒数值。

☑　getTime 方法

该方法返回自 1970 年 1 月 1 日 00:00:00 GMT 以来的 Date 对象，此 Date 对象表示的毫秒数，该方法声明如下：

```
public long getTime()
```

☑　setTime()方法

该方法用于设置 Date 对象，以表示 1970 年 1 月 1 日 00:00:00 GMT 以后 time 毫秒的时间点，该方法声明如下：

```
public void setTime(long time)
```

参数说明：time 是毫秒数。

13.1.4　日期的格式化输出

在应用程序设计中，经常需要显示时间和日期。String 类的静态方法 format()通过特殊转义符作为参数可以实现对日期和时间的格式化，该方法用于创建格式化的字符串，它有以下两种重载形式。

方法声明：

```
format(String format,Object…args)
```

参数说明：format 是格式字符串。args 是格式字符串中由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。该方法使用指定的格式字符串和参数返回一个格式化字符串。格式化后的新字符串使用本地默认的语言环境。

方法声明：

```
format(Local local,String format,Object…args)
```

参数说明：local 是格式化过程中要应用的语言环境。如果为 null，则不进行本地化。format 是格式字符串。args 是格式字符串中由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。

1．日期格式化

对于日期的格式化，可以使用转义符，把 Date 类的实例对象格式化。

例如，返回年月日中的日，实例代码如下：

```
Date date = new Date();                          // 创建 Date 对象 date
String day = String.format("%te", date);         // 通过 format()方法对 date 进行格式化
```

上述代码变量 day 的值是当前日期中的天数，如今天是 15 号，则 day 的值为 15。“%te”是转换符，常用的日期和时间的格式转换符如表 13.1 所示。

表 13.1　常见的日期时间转换符

转　换　符	说　　明	示　　例
%te	一个月中的某一天（1～31）	2
%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 13.03　在项目中创建类 Eval，实现将当前日期信息以 4 位年份、月份全称、2 位日期形式输出。实例代码如下：**（实例位置：光盘\mr\ym\13.03）**

```
public class Eval {                                          // 新建类
    public static void main(String[] args) {                // 主方法
        Date date = new Date();                              // 创建 Date 对象 date
        String year = String.format("%tY", date);            // 将 date 进行格式化
        String month = String.format("%tB", date);           // 将 date 进行格式化
        String day = String.format("%td", date);             // 将 date 进行格式化
        System.out.println("今年是：" + year + "年");          // 输出信息
        System.out.println("现在是：" + month);               // 输出信息
        System.out.println("今天是：" + day + "号");          // 输出信息
    }
}
```

实例的运行效果如图 13.3 所示。

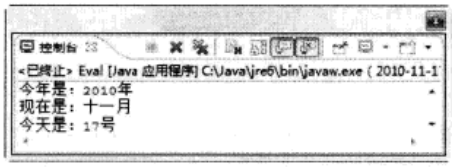


图 13.3　实例运行效果

2．时间格式化

使用 format()方法不但可以完成日期的格式化，也可以实现时间的格式化。时间格式化转换符比日期转换符种类更多、更精确，它可以将时间格式化成时、分、秒、毫秒。格式化时间的转换符如表 13.2 所示。

表 13.2　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

续表

转 换 符	说 明	示 例
%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 13.04 在项目中创建类 GetDate，实现将当前时间信息以 2 位小时、2 位分钟数、2 位秒数形式输出。实例代码如下：**（实例位置：光盘\mr\ym\13.04）**

```
public class GetDate {                                                    // 新建类
    public static void main(String[] args) {                              // 主方法
        Date date = new Date();                                           // 创建 Date 对象 date
        String hour = String.format("%tH", date);                         // 将 date 进行格式化
        String minute = String.format("%tM", date);                       // 将 date 进行格式化
        String second = String.format("%tS", date);                       // 将 date 进行格式化
        System.out.println("现在是：" + hour + "时" + minute + "分" + second + "秒");  // 输出的信息
    }
}
```

实例的运行效果如图 13.4 所示。

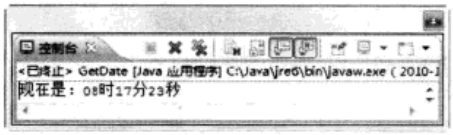


图 13.4 实例运行效果

3. 格式化常见的日期时间组合

格式化日期与时间的转换符定义了各种格式日期字符串的方式，其中最常用的日期和时间的组合格式如表 13.3 所示。

表 13.3 常见的日期和时间的组合格式

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 13.05　在项目中创建类 DateAndTime，在 main()方法中实现将当前日期时间的全部信息以及指定格式的日期输出。实例代码如下：**（实例位置：光盘\mr\ym\13.05）**

```
public class DateAndTime {                                          // 创建类
    public static void main(String[] args) {                       // 主方法
        Date date = new Date();                                     // 创建 Date 对象 date
        String time = String.format("%tc", date);                   // 将 date 格式化
        String form = String.format("%tF", date);                   // 将 date 格式化
        System.out.println("全部的时间信息是：" + time);              // 将格式化后的日期时间输出
        System.out.println("年-月-日格式：" + form);
    }
}
```

实例的运行效果如图 13.5 所示。

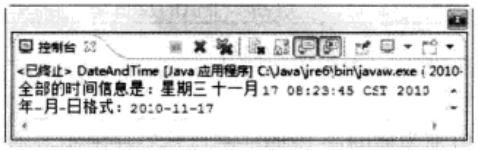


图 13.5　实例运行效果

13.2　数 学 运 算

在 Java 语言中提供了一个执行数学基本运算的 Math 类，该类包括常见且有用的数学运算方法，如三角函数方法、指数函数方法、对数函数方法、平方根函数方法等一些常用数学函数，除此之外该类还提供了一些常用的数学常量，如 PI、E 等，本节将介绍 Math 类以及其中一些常用的函数方法。

13.2.1　Math 类

Math 类包含了所有关于数学运算的函数方法，这些方法都是静态的，所以每个方法只要使用“Math.数学方法”就可以调用，使用起来比较简单。

在 Math 类中提供了众多数学函数方法，主要包括三角函数方法、指数函数方法、取整函数方法和取最大值、最小值以及平均值函数方法，这些方法都被定义为 static 形式，所以在程序中应用比较简便，可以使用如下语法进行调用：

```
Math.数学方法
```

在 Math 类中除了函数方法之外还存在一些常用数学常量，如 PI、E 等，这些数学常量作为 Math 类的成员变量出现，调用起来也很简单，可以使用如下语法进行调用：

```
Math.PI
Math.E
```

13.2.2　Math 类的数学方法

Math 类中的常用数学运算方法较多，大致可以将这些数学函数方法分为四大类别，分别为三角函数方法、指数函数方法、取整函数方法以及取最大值、最小值和绝对值的函数方法。

1. 三角函数方法

在 Math 类中包含的三角函数方法如表 13.4 所示。

表 13.4 Math 类中包含的三角函数方法及说明

方　法	说　明	返回值类型
sin(double a)	返回角的三角正弦	double
cos(double a)	返回角的三角余弦	double
tan(double a)	返回角的三角正切	double
asin(double a)	返回一个值的反正弦	double
acos(double a)	返回一个值的反余弦	double
atan(double a)	返回一个值的反正切	double
toRadians(double angdeg)	将角度转换为弧度	double
toDegrees(double angrad)	将弧度转换为角度	double

以上每个方法的参数和返回值都为 double 型的，将这些方法中参数的值设置为 double 型是有一定道理的，参数以弧度代替角度来实现，其中 1 度等于 π/180 弧度，所以 180 度可以使用 π 弧度来表示。除了可以获取角的正弦、余弦、正切、反正弦、反余弦、反正切之外，Math 类还提供了角度和弧度相互转换的 toRadians()和 toDegrees()方法，但值得注意的是，角度与弧度的互换通常是不精确的。

例 13.06 在项目中创建 TrigonometricFunction 类，在类的 main()方法中调用 Math 类提供的这种数学运算方法，并输出运算结果。实例代码如下：**（实例位置：光盘\mr\ym\13.06）**

```
public class TrigonometricFunction {
    public static void main(String[] args) {
        System.out.println("90 度的正弦值：" + Math.sin(Math.PI / 2));          // 取 90 度的正弦
        System.out.println("0 度的余弦值：" + Math.cos(0));                     // 取 0 度的余弦
        System.out.println("60 度的正切值：" + Math.tan(Math.PI / 3));          // 取 60 度的正切
        System.out.println("2 的平方根与 2 商的反正切值：" + Math.asin(Math.sqrt(2) / 2));
                                                                               // 取 2 的平方根与 2 商的反正弦
        System.out.println("2 的平方根与 2 商的反余切值：" + Math.acos(Math.sqrt(2) / 2));
                                                                               // 取 2 的平方根与 2 商的反余弦
        System.out.println("1 的反正切值：" + Math.atan(1));                    // 取 1 的反正切
        System.out.println("120 度的弧度值：" + Math.toRadians(120.0));         // 取 120 度的弧度值
        System.out.println("π/2 的角度值：" + Math.toDegrees(Math.PI / 2));     // 取 π/2 的角度
    }
}
```

实例的运行效果如图 13.6 所示。

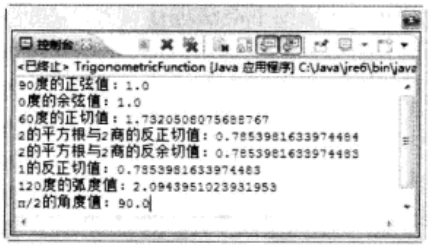


图 13.6 实例运行效果

从图 13.6 所示的运行结果中可以看出，90 度的正弦值为 1，0 度的余弦值为 1，60 度的正切值与 Math.sqrt(3) 的值应该是一致的，也就是取 3 的平方根。在结果中可以看到第 3 行和第 4 行的值是基本相同的，这个值换算后正是 45 度，也就是获取的 Math.sqrt(2)/2 反正弦、反余弦值与 1 的反正切值都是 45 度。最后两行打印语句实现的是角度和弧度的转换，其中 Math.toRadians(120.0) 语句是获取 120 度的弧度值，而 Math.toDegrees(Math.PI/2) 语句获取 π/2 的角度。读者可以将这些具体的值使用 π 的形式表示出来，与上述结果应该是基本一致的，这些结果不能做到十分精确，因为 π 本身也是一个近似值。

2．指数函数方法

在 Math 类中存在与指数相关的函数方法如表 13.5 所示。

表 13.5　Math 类中存在与指数相关的函数方法及说明

方　法	说　明	返回值类型
public static double exp(double a)	获取 e 的 a 次方，即取 e^a	double
public static double log(double a)	取自然对数	double
public static double log10(double a)	取底数为 10 的对数	double
public static double sqrt(double a)	取 a 的平方根，其中 a 的值不能为负值	double
public static double cbrt(double a)	取 a 的立方根	double
public static double pow(double a,double b)	取 a 的 b 次方	double

指数运算包括求平方根、取对数以及求 n 次方的运算，为了使读者更好地理解这些运算函数方法的用法，请读者看下面的实例。

例 13.07　在项目中创建 ExponentFunction 类，在类的 main()方法中调用 Math 类中的方法实现指数函数的运算，并输出运算结果。实例代码如下：**（实例位置：光盘\mr\ym\13.07）**

```
public class ExponentFunction {
    public static void main(String[] args) {
        System.out.println("e 的平方值：" + Math.exp(2));                    // 取 e 的 2 次方
        System.out.println("以 e 为底 2 的对数值：" + Math.log(2));          // 取以 e 为底 2 的对数
        System.out.println("以 10 为底 2 的对数值：" + Math.log10(2));       // 取以 10 为底 2 的对数
        System.out.println("4 的平方根值：" + Math.sqrt(4));                 // 取 4 的平方根
        System.out.println("8 的立方根值：" + Math.cbrt(8));                 // 取 8 的立方根
        System.out.println("2 的 2 次方值：" + Math.pow(2, 2));              // 取 2 的 2 次方
    }
}
```

实例的运行效果如图 13.7 所示。

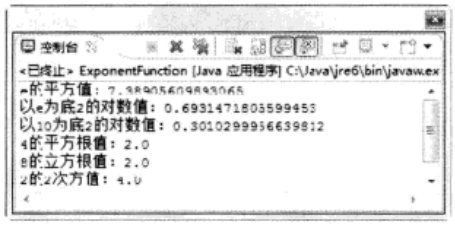


图 13.7　实例运行效果

从本实例可以看到，使用 Math 类中的方法比较简单，直接使用 Math 类名调用相应的方法即可。

3. 取整函数方法

在具体的问题中，取整操作使用也很普遍，所以 Java 在 Math 类中添加了数字取整方法，在 Math 类中关键的取整方法及说明如表 13.6 所示。

表 13.6 Math 类中关键的取整方法及说明

方　法	说　明	返回值类型
ceil(double a)	返回大于等于参数的最小整数	double
floor(double a)	返回小于等于参数的最大整数	double
rint(double a)	返回与参数最接近的整数，如果两个同为整数都同样接近，则结果取偶数	double
round(float a)	将参数加上 1/2 后返回与参数最近的整数	int
round(double a)	将参数加上 1/2 后返回与参数最近的整数，然后强制转换为长整型	long

在坐标轴上表示上述方法的布局如图 13.8 所示。

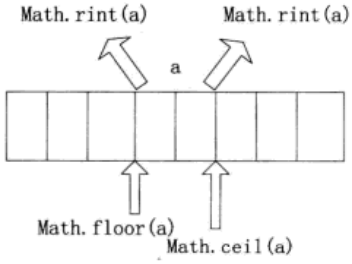


图 13.8 在坐标轴上表示方法的布局

例 13.08 在项目中创建 IntFunction 类，在该类的 main()方法中调用 Math 类中的方法实现取整函数的运算，并输出运算结果。实例代码如下：**（实例位置：光盘\mr\ym\13.08）**

```
public class IntFunction {
    public static void main(String[] args) {
        System.out.println("使用 ceil()方法取整：" + Math.ceil(5.2));      // 返回第一个大于等于参数的整数
        System.out.println("使用 floor()方法取整：" + Math.floor(2.5));    // 返回第一个小于等于参数的整数
        System.out.println("使用 rint()方法取整：" + Math.rint(2.7));      // 返回与参数最接近的整数
        System.out.println("使用 rint()方法取整：" + Math.rint(2.5));      // 返回与参数最接近的整数
        System.out.println("使用 round()方法取整：" + Math.round(3.4f)); // 将参数加上 0.5 后返回最接近的整数
        // 将参数加上 0.5 后返回最接近的整数，并将结果强制转换为长整型
        System.out.println("使用 round()方法取整：" + Math.round(2.5));
    }
}
```

实例的运行效果如图 13.9 所示。

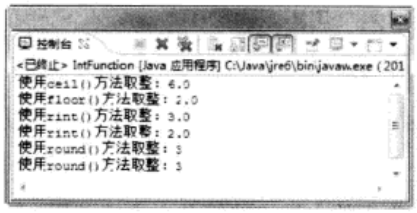


图 13.9 实例运行效果

4．取最大值、最小值以及绝对值函数方法

在程序中最常用的方法就是取最大值、最小值以及绝对值等，在 Math 类中包括的这些操作方法如表 13.7 所示。

表 13.7　Math 类中关键方法及说明

方　法	说　明	返回值类型
max(double a,double b)	取 a 与 b 之间的最大值	double
min(int a,int b)	取 a 与 b 之间的最小值，参数为整型	int
min(long a,long b)	取 a 与 b 之间的最小值，参数为长整型	long
min(float a,float b)	取 a 与 b 之间的最小值，参数为浮点型	float
min(double a,double b)	取 a 与 b 之间的最小值，参数为双精度型	double
abs(int a)	返回整型参数的绝对值	int
abs(long a)	返回长整型参数的绝对值	long
abs(float a)	返回浮点型参数的绝对值	float
abs(double a)	返回双精度型的绝对值	double

例 13.09　在项目中创建 AnyFunction 类，在该类的 main()方法中调用 Math 类中的方法实现求两数的最大值、最小值和取绝对值运算，并输出运算结果。实例代码如下：**（实例位置：光盘\mr\ym\13.09）**

```
public class AnyFunction {
    public static void main(String[] args) {
        System.out.println("4 和 8 较大者:" + Math.max(4, 8));          // 取两个参数的最大值
        System.out.println("4.4 和 4 较小者：" + Math.min(4.4, 4));     // 取两个参数的最小值
        System.out.println("-7 的绝对值：" + Math.abs(-7));             // 取参数的绝对值
    }
}
```

实例的运行效果如图 13.10 所示。

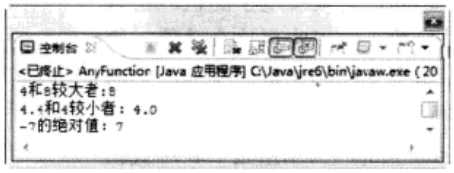


图 13.10　实例运行效果

13.3　随　机　数

在实际开发中随机数的使用是很普遍的，所以在程序中产生随机数操作很重要。在 Java 中主要提供两种方式产生随机数，分别为 Math 类的 random()方法和 Random 类提供的产生各种数据类型随机数的方法，本节将讲述这两种产生随机数的方式。

13.3.1　通过 Math 类生成随机数

在 Math 类中存在一个 random()方法，该方法用于产生随机数字，这个方法默认生成大于等于 0.0 小于

1.0 的 double 型随机数，即 0.00<= Math.random() < 1.0，虽然 Math.random()方法只可以产生 0～1 之间的 double 型数字，其实只要在 Math.random()语句上稍加处理，就可以通过这个方法产生任意范围的随机数，如图 13.11 所示。

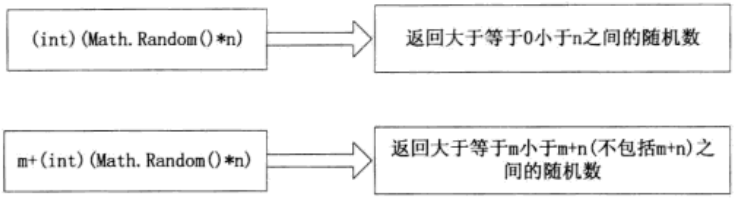


图 13.11　产生随机数示意图

例 13.10　在项目中创建 MathRondom 类，在类中编写 GetEvenNum()方法产生两数之间的随机数，并在 main()方法中输出这个随机数。实例代码如下：**（实例位置：光盘\mr\ym\13.10）**

```
public class MathRondom {
    public static int GetEvenNum(double num1, double num2) {          // 定义产生偶数的方法
        int s = (int) num1 + (int) (Math.random() * (num2 - num1));    // 产生 num1～num2 之间的随机数
        if (s % 2 == 0) {                                              // 判断随机数是否为偶数
            return s;                                                  // 返回
        } else
                                                                       // 如果是奇数
            return s + 1;                                              // 将结果加 1 后返回
    }
    public static void main(String[] args) {
        System.out.println("任意一个 2～32 之间的偶数：" + GetEvenNum(2, 32)); // 调用产生随机数的方法
    }
}
```

实例的运行效果如图 13.12 所示。

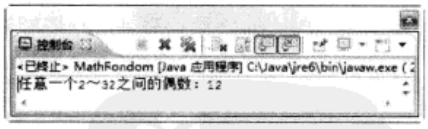


图 13.12　实例运行效果

本实例在每次运行时结果都不相同，这就实现了随机产生数据的效果，并且每次产生的值都是偶数，为了实现这个功能，笔者定义了一个 GetEvenNum()方法，这个方法的参数分别为产生随机数字的上限与下限，m+(随机数*n)这个公式可以获取 m～m+n 之间的随机数，所以 2+(int)(Math.random()*(32-2));这个表达式就可以求出 2～32 之间的随机数。当获取到这个区间的随机数以后需要判断这个数字是否为偶数，对该数字做对 2 取余的操作即可判断这个数字是否为偶数，如果该数字为奇数，将该奇数加 1 也可以返回偶数。

使用 Math 类的 random()方法也可以随机生成字符，可以使用如下代码生成字符 a～z 之间的字符。

```
(char)('a'+Math.random()*('z'-'a'+1));
```

通过上述表达式可以求出更多的随机字符，如 A～Z 之间的随机字符，进而可以推理到求任意两个字符之间的随机字符，可以使用如下语句表示：

```
(char)(cha1+Math.random()*(cha2-cha1+1));
```

在这里可以将这个表达式设计为一个方法，参数设置为随机产生字符的上限与下限，为了说明这个方法，请读者看下面的实例。

例 13.11　在项目中创建 MathRandomChar 类，在类中编写 GetRandomChar()方法产生随机字符，并在 main()方法中输出该字符。实例代码如下：**（实例位置：光盘\mr\ym\13.11）**

```
public class MathRandomChar {
    public static char GetRandomChar(char cha1, char cha2) {          // 定义获取任意字符之间的随机字符
        return (char) (cha1 + Math.random() * (cha2 - cha1 + 1));
    }
    public static void main(String[] args) {
        System.out.println("任意小写字符：" + GetRandomChar('a', 'z')); // 获取 a～z 之间的随机字符
        System.out.println("任意大写字符：" + GetRandomChar('A', 'Z'));// 获取 A～Z 之间的随机字符
        System.out.println("0 到 9 任意数字字符：" + GetRandomChar('0', '9')); // 获取 0～9 之间的随机字符
    }
}
```

实例的运行效果如图 13.13 所示。

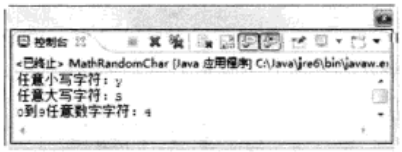


图 13.13　实例运行效果

注意：random()方法返回的值实际上是伪随机数，它通过复杂的运算得到一系列的数，该方法是通过当前时间作为随机数生成器的参数，所以每次执行程序都会产生不同的随机数。

13.3.2　使用 Random 类生成随机数

除了使用 Math 类中的 random()方法可以获取随机数之外，在 Java 中还提供了一种可以获取随机数的方式，那就是 java.util.Random 类，该类可以通过实例化一个 Random 对象创建一个随机数生成器。

方法声明：

```
public Random();
```

以这种形式实例化对象时，Java 编译器以系统当前时间作为随机数生成器的种子，因为每时每刻的时间不可能相同，所以产生的随机数也不同，但是如果运行速度太快，也会产生两次运行结果相同的随机数。

同时也可以在实例化 Random 类对象时，设置随机数生成器的种子。

方法声明：

```
public Random(seedValue);
```

参数说明：seedValue 是随机数生成器的种子。

在 Random 类中提供获取各种数据类型随机数的方法，表 13.8 中列举了几个常用的方法。

表 13.8　Random 类中关键方法及说明

方　法	说　明	返回值类型
nextInt()	返回一个随机整数	int
nextInt(int n)	返回大于等于 0 小于 n 的随机整数	int
nextLong()	返回一个随机长整型值	long
nextBoolean()	返回一个随机布尔型值	boolean
nextFloat()	返回一个随机浮点型值	float
nextDouble()	返回一个随机双精度型值	double
nextGaussian()	返回一个概率密度为高斯分布的双精度值	double

例 13.12 在项目中创建 RandomDemo 类，在类的 main()方法中创建 Random 类的对象，使用该对象生成各种类型的随机数，并输出结果。实例的代码如下：**（实例位置：光盘\mr\ym\13.12）**

```
public class RandomDemo {
    public static void main(String[] args) {
        Random r = new Random();                                    // 实例化一个 Random 类
        System.out.println("随机产生一个整数:" + r.nextInt());
        System.out.println("随机产生一个大于等于 0 小于 10 的整数：" + r.nextInt(10));
        System.out.println("随机产生一个布尔型的值：" + r.nextBoolean());
        System.out.println("随机产生一个双精度型的值：" + r.nextDouble());
        System.out.println("随机产生一个浮点型的值：" + r.nextFloat());
        System.out.println("随机产生一个概率密度为高斯分布的双精度值：" + r.nextGaussian());
    }
}
```

实例的运行效果如图 13.14 所示。

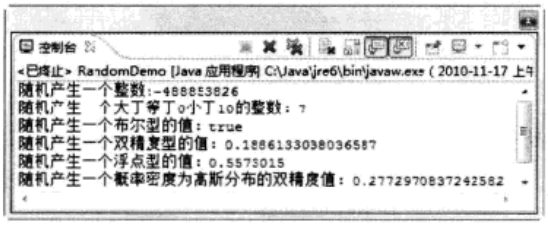


图 13.14 实例运行效果

13.4 数字格式化类

数字的格式化在解决实际问题时使用非常普遍，如表示某超市商品价格需要保留两位有效数字等问题。Java 主要对浮点型数据进行数字格式化操作，其中浮点型数据包括 double 型和 float 型数据，在 Java 中使用 java.text.DecimalFormat 类格式化数字，在本节中将着重讲解 DecimalFormat 类。

13.4.1 DecimalFormat 类

在 Java 中没有格式化的数据应遵循以下原则：

☑ 如果数据绝对值大于 0.001 且小于 10000000，Java 将以常规小数形式表示。

☑ 如果数据绝对值小于 0.001 或者大于 10000000，使用科学计数法表示。

由于上述输出格式不能满足解决实际问题的要求，所以通常将结果格式化为指定形式进行输出。在 Java 中可以使用 DecimalFormat 类进行格式化操作。

DecimalFormat 是 NumberFormat 的一个子类，该类用于格式化十进制数字，它可以将一些数字格式化为整数、浮点数、科学计数法、百分数等。通过使用该类可以为要输出的数字加上单位或控制数字的精度。一般情况下在实例化 DecimalFormat 对象时可以传递数字格式，也可以通过 DecimalFormat 类中的 applyPattern()方法来实现数字格式化。

当格式化数字时，在 DecimalFormat 类中使用一些特殊字符构成一个格式化模板，使数字按照一定的特殊字符规则进行匹配，表 13.9 中列举了定义格式化模板的特殊字符及其含义。

表 13.9　DecimalFormat 类中的特殊字符及其含义

字　符	含　义
0	代表阿拉伯数字，使用特殊字符“0”表示数字的一位阿拉伯数字。如果该位不存在数字，则显示 0
#	代表阿拉伯数字，使用特殊字符“#”表示数字的一位阿拉伯数字。如果该位存在数字，则显示字符；如果该位不存在数字，则不显示
.	小数分隔符或货币小数分隔符
-	负号
,	分组分隔符
E	分隔科学计数法中的尾数和指数
%	本符号放置在数字的前缀或后缀，将数字乘以 100 显示为百分数
\u2030	本符号放置在数字的前缀或后缀，将数字乘以 1000 显示为千分数
¤ \u00A4	本符号放置在数字的前缀或后缀，作为货币记号
'	本符号为单引号，当上述特殊字符出现在数字中时，应为特殊符号添加单引号，系统会将此符号视为普通符号处理

13.4.2　数字的格式化输出

本节以实例说明数字格式化的使用。

例 13.13　在项目中创建 DecimalFormatSimpleDemo 类，在该类中分别定义 SimgleFormat()方法和 UseApplyPatternMethodFormat()方法实现两种格式化数字的方式。实例代码如下：**（实例位置：光盘\mr\ym\13.13）**

```
import java.text.DecimalFormat;
public class DecimalFormatSimpleDemo {
    static public void SimgleFormat(String pattern, double value) {  // 使用实例化对象时设置格式化模式
        DecimalFormat myFormat = new DecimalFormat(pattern);  // 实例化 DecimalFormat 对象
        String output = myFormat.format(value);               // 将数字进行格式化
        System.out.println(value + "\t" + pattern + "\t" + output);
    }
    // 使用 applyPattern()方法对数字进行格式化
    static public void UseApplyPatternMethodFormat(String pattern, double value) {
        DecimalFormat myFormat = new DecimalFormat();         // 实例化 DecimalFormat 对象
        myFormat.applyPattern(pattern);                       // 调用 applyPatten()方法设置格式化模板
        System.out.println(value + " \t\t" + pattern + "\t\t" + myFormat.format(value));
    }
    public static void main(String[] args) {
        SimgleFormat("###,###.###", 123456.789);              // 调用静态 SimgleFormat()方法
        SimgleFormat("00000000.###kg", 123456.789);           // 在数字后加上单位
        // 按照格式模板格式化数字，不存在的位以 0 显示
        SimgleFormat("000000.000", 123.78);
        // 调用静态 UseApplyPatternMethodFormat()方法
        UseApplyPatternMethodFormat("#.###%", 0.789);         // 将数字转化为百分数形式
        UseApplyPatternMethodFormat("###.##", 123456.789);    // 将小数点后格式化为两位
        UseApplyPatternMethodFormat("0.00\u2030", 0.789);     // 将数字转化为千分数形式
    }
}
```

实例的运行效果如图 13.15 所示。

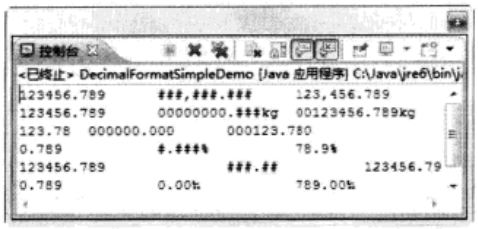


图 13.15　实例运行效果

从实例 13.13 中可以看到，代码的第一行使用 import 关键字将 java.text.DecimalFormat 这个类包含进来，这是首先告知系统，下面的代码将使用到 DecimalFormat 类，然后笔者定义两个格式化数字的方法，这两个方法的参数个数都为两个，分别代表数字格式化的模板字符串和具体需要格式化的数字，虽然这两个方法都可以实现格式化数字的操作，但使用的方式不同，SimgleFormat()方法是在实例化 DecimalFormat 对象时为构造方法传参设置数字格式化模板，而 UseApplyPatternMethodFormat()方法是在实例化 DecimalFormat 对象后调用 applyPattern()方法设置数字格式化模板。

最后在主方法中根据不同形式模板格式化数字。在结果中可以看到，对以“0”特殊字符构成的模板进行格式化，当数字的某位不存在时，将补位显示 0；而对以“#”特殊字符构成的模板进行格式化操作时，格式化后的数字位数与数字本身的位数一致。

在 DecimalFormat 类中除了可以设置格式化模式来格式化数字之外，还可以使用一些特殊的方法对数字进行格式化设置，例如：

```
DecimalFormat myFormat=new DecimalFormat();          // 实例化 DecimalFormat 类对象
myFormat.setGroupingSize(2);                         // 设置将数字分组的大小
myFormat.setGroupingUsed(false);                     // 设置是否支持分组
```

在上述代码中 setGroupingSize()方法是设置格式化数字的分组大小，而 setGroupingUsed()方法设置是否可以对数字进行分组操作。为了使读者更好地理解这两个方法的使用，请看下面的实例。

例 13.14　在项目中，创建 DecimalMethod 类，在该类的 main()方法中调用 setGroupingSize()与 setGroupingUsed()方法实现数字的分组。实例代码如下：**（实例位置：光盘\mr\ym\13.14）**

```
public class DecimalMethod {
    public static void main(String[] args) {
        DecimalFormat myFormat = new DecimalFormat();
        myFormat.setGroupingSize(2);                          // 设置将数字分组为 2
        String output = myFormat.format(123456.789);
        System.out.println("将数字以每两个数字分组 " + output);
        myFormat.setGroupingUsed(false);                      // 设置不允许数字进行分组
        String output2 = myFormat.format(123456.789);
        System.out.println("不允许数字分组 " + output2);
    }
}
```

实例的运行效果如图 13.16 所示。

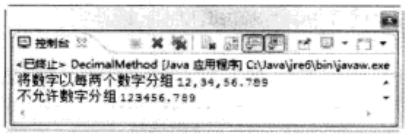


图 13.16　实例运行效果

13.5　照猫画虎——基本功训练

13.5.1　基本功训练 1——输出 30° 的三角函数值

视频讲解：光盘\mr\lx\13\输出 30° 的三角函数值.exe

实例位置：光盘\mr\ym\13.15\src

三角函数在数学中有着非常广泛的应用，为此 Java 提供了 Math 来完成正弦值、正切值和余弦值的计算。对于其他三角函数值，可以通过这几个值计算获得。例如，需要计算余切值，只需要将正切值取倒数即可。本实例将使用其方法来计算 30° 的常见三角函数值，代码如下：

```
public class MathDemo {
    public static void main(String[] args) {
        double angle = Math.toRadians(30);// 将角度值转换成弧度值
        System.out.println("30°的正弦值：" + Math.sin(angle));
        System.out.println("30°的余弦值：" + Math.cos(angle));
        System.out.println("30°的正切值：" + Math.tan(angle));
    }
}
```

实例的运行效果如图 13.17 所示。

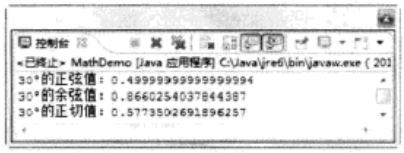


图 13.17　实例运行效果

注意：三角函数的参数要使用弧度值，因此需要使用 toRadians()方法将 30° 转换成对应的弧度值。

照猫画虎：在控制台上输出 60° 的三角函数值。（20 分）（实例位置：光盘\mr\ym\13.15\zmhh）

13.5.2　基本功训练 2——输出 0.5 的反三角函数值

视频讲解：光盘\mr\lx\13\输出 0.5 的反三角函数值.exe

实例位置：光盘\mr\ym\13.16\src

在 java.lang.Math 类中，提供了很多与数学计算相关的方法。本实例将使用其中的方法来计算 0.5 的常见反三角函数值，代码如下：

```
public class MathDemo {
    public static void main(String[] args) {
        double value = 0.5;
        System.out.println("0.5 的反正弦值：" + Math.asin(value));
        System.out.println("0.5 的反余弦值：" + Math.acos(value));
        System.out.println("0.5 的反正切值：" + Math.atan(value));
    }
}
```

实例的运行效果如图 13.18 所示。

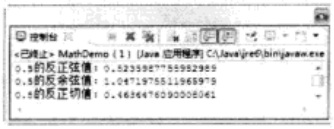


图 13.18　实例运行效果

提示：反三角函数的返回值是弧度制的。

照猫画虎：在控制台上输出 1 的反三角函数值。（20 分）（实例位置：光盘\mr\ym\13.16\zmhh）

13.5.3　基本功训练 3——角度和弧度的转换

视频讲解：光盘\mr\lx\13\角度和弧度的转换.exe

实例位置：光盘\mr\ym\13.17\src

Math 中三角函数的参数都是弧度制的。如果指定的参数是角度，就涉及如何将角度转换成弧度。对于反三角函数，其返回值是弧度制的，如果需要角度，则涉及如何将弧度转换成角度。这两种情况在 Math 类中都提供了相应的方法来支持。本实例将演示这两个方法的使用，代码如下：

```
public class MathDemo {
    public static void main(String[] args) {
        System.out.println("弧度值 0.5 转换成角度值：" + Math.toDegrees(0.5));
        System.out.println("角度值 30 转换成弧度值：" + Math.toRadians(30));
    }
}
```

实例的运行效果如图 13.19 所示。

图 13.19　实例运行效果

你问我答：toDegrees()和 toRadians()方法的作用各是什么？

这两个方法是 Math 类提供的将角度转换成弧度和将弧度转换成角度的方法。

方法声明：

```
public static double toDegrees(double angrad)
```

参数说明：angrad 是需要转换的弧度。

方法声明：

```
public static double toRadians(double angdeg)
```

参数说明：angdeg 是需要转换的角度。

照猫画虎：在控制台上以角度制输出 0.5 的反正弦值。（20 分）（实例位置：光盘\mr\ym\13.17\zmhh）

13.5.4　基本功训练 4——高精度整数运算

视频讲解：光盘\mr\lx\13\高精度整数运算.exe

实例位置：光盘\mr\ym\13.18\src

通过观察前面实例的运算结果，可以发现 Java 虚拟机的运算并不精确。例如，30° 的正弦值应该是 0.5，

而 sin()方法的返回值是 0.49。这对于需要高精度的场合显然不理想，为此 Java 提供了 BigInteger 类，该类可以完成任意精度的整数运算。本实例将使用其完成基本的四则运算，代码如下：

```
public class BigIntegerDemo {
    public static void main(String[] args) {
        BigInteger number1 = new BigInteger("5");                    // 创建 BigInteger 对象
        BigInteger number2 = new BigInteger("10");                   // 创建 BigInteger 对象
        System.out.println("5 + 10 = " + number1.add(number2));
        System.out.println("5 - 10 = " + number1.subtract(number2));
        System.out.println("5 * 10 = " + number1.multiply(number2));
        System.out.println("5 / 10 = " + number1.divide(number2));
    }
}
```

实例的运行效果如图 13.20 所示。

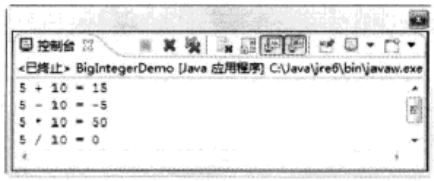


图 13.20　实例运行效果

你问我答：add()、subtract()、multiply()和 divide()方法的作用各是什么？

它们是在 BigInteger 类中定义的，可以用来完成基本的加、减、乘和除法运算。这些方法及说明如表 13.10 所示。

表 13.10　BigInteger 类中的基本方法及说明

方　法　名	说　　明
add(BigInteger val)	计算 this+val 的结果
subtract(BigInteger val)	计算 this−val 的结果
multiply(BigInteger val)	计算 this*val 的结果
divide(BigInteger val)	计算 this/val 的结果

照猫画虎：使用 BigInteger 类计算 200 的阶乘。（20 分）（实例位置：光盘\mr\ym\13.18\zmhh）

13.5.5　基本功训练 5——高精度浮点运算

视频讲解：光盘\mr\lx\13\高精度浮点运算.exe

实例位置：光盘\mr\ym\13.19\src

通过观察运算的结果，可以发现 Java 虚拟机的运算并不精确。例如，30° 的正弦值应该是 0.5，而 sin()方法的返回值是 0.49。这对于需要高精度的场合显然不理想，为此 Java 提供了 BigDecimal 类，它可以完成任意精度的浮点运算。本实例将使用其完成基本的四则运算。代码如下：

```
public class BigDecimalDemo {
    public static void main(String[] args) {
        BigDecimal number1 = new BigDecimal("5");                    // 创建 BigDecimal 对象
        BigDecimal number2 = new BigDecimal("10");                   // 创建 BigDecimal 对象
```

```
        System.out.println("5 + 10 = " + number1.add(number2));
        System.out.println("5 - 10 = " + number1.subtract(number2));
        System.out.println("5 * 10 = " + number1.multiply(number2));
        System.out.println("5 / 10 = " + number1.divide(number2));
    }
}
```

实例的运行效果如图 13.21 所示。

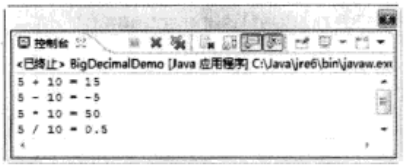


图 13.21　实例运行效果

你问我答： add()、subtract()、multiply()和 divide()方法的作用各是什么？

它们是在 BigDecimal 类中定义的，可以用来完成基本的加、减、乘和除法运算。这些方法及说明如表 13.11 所示。

表 13.11　BigDecimal 类中的基本方法及说明

方 法 名	说　明
add(BigDecimal val)	计算 this+val 的结果
subtract(BigDecimal val)	计算 this-val 的结果
multiply(BigDecimal val)	计算 this*val 的结果
divide(BigDecimal val)	计算 this/val 的结果

照猫画虎：使用 BigDecimal 类计算 1+1／2!+1／3!…1／20!的值。（20 分）（实例位置：光盘\mr\ym\13.19\zmhh）

照猫画虎栏目分数统计：

照猫画虎题目	1	2	3	4	5	总分数	
分数							

13.6　情景应用——拓展与实践

13.6.1　情景应用 1——简单的数字时钟

视频讲解：光盘\mr\lx\13\简单的数字时钟.exe

实例位置：光盘\mr\ym\13.20\src

在日常生活中，数字时钟随处可见。它比模拟时钟更加容易阅读，在一些重要的场合，如田径赛场，数字时钟是不可或缺的工具之一。本实例将使用 Java 语言编写一个简单的数字时钟，它可以按“小时：分钟：秒钟”的样式在标签中显示当前的时间，实例的运行效果如图 13.22 所示。

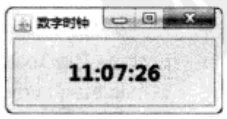


图 13.22　实例运行效果

实现过程如下：

（1）编写 format()方法，用来将数字格式化成长度为 2 的字符串，关键代码如下：

```
private String format(int number) {
    return number < 10 ? "0" + number : "" + number;          // 如果数字小于 10 就在其前面加 0 补齐
}
```

（2）编写 getTime()方法，用来获取虚拟机的当前时间，然后以一个指定的字符串形式返回，关键代码如下：

```
private String getTime() {
    Calendar calendar = new GregorianCalendar();
    int hour = calendar.get(Calendar.HOUR_OF_DAY);                    // 获得当前小时
    int minute = calendar.get(Calendar.MINUTE);                       // 获得当前分钟
    int second = calendar.get(Calendar.SECOND);                       // 获得当前秒
    return format(hour) + ":" + format(minute) + ":" + format(second);    // 返回格式化的字符串
}
```

（3）编写 do_this_windowActivated()方法，用来完成对窗体激活事件的响应。在该方法中，每隔 1 秒钟更新一次标签上的文本，内容是当前的时间。将这段代码放到死循环中，从而让标签不断地更新。需要特别注意的是，这些代码是在一个新的线程中运行的，关键代码如下：

```
protected void do_this_windowActivated(WindowEvent e) {
    new Thread() {
        public void run() {
            while (true) {                                            // 让时钟一直处于更新状态
                label.setText(getTime());                             // 更新时钟
                try {
                    Thread.sleep(1000);                               // 休眠 1 秒钟
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        };
    }.start();
}
```

你问我答：calendar.get(Calendar.HOUR_OF_DAY);的作用是什么？

Calendar 类是 Java 中定义的日期类。Calendar.HOUR_OF_DAY 的作用是表示当前所在的小时。使用 get()方法即可获得这个小时数。Calendar.MINUTE 和 Calendar.SECOND 的含义类似，分别表示分钟和秒钟。

DIY：在本实例的基础上增加显示日期的功能。（20 分）（实例位置：光盘\mr\ym\13.20\diy）

13.6.2　情景应用 2——简单的模拟时钟

视频讲解：光盘\mr\lx\13\简单的模拟时钟.exe

实例位置：光盘\mr\ym\13.21\src

在日常生活中，模拟时钟随处可见。本实例将使用 Java 语言编写一个简单的模拟时钟，它比数字时钟更加美观。实例的运行效果如图 13.23 所示。

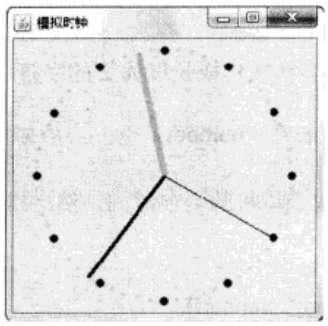


图 13.23　实例运行效果

实现过程如下：

（1）新建项目并创建窗体类 SimulatedClock。

（2）重写 JFrame 类中的 paint()方法，使用该方法在窗体中绘制刻度、表盘、指针等，其中指针的位置要依据当前的时间来绘制，关键代码如下：

```
public void paint(Graphics g) {
    super.paint(g);                                          // 调用父类的 paint()方法，这样能在画图时保存外观
    Rectangle rectangle = getBounds();                       // 获得控件的区域
    Insets insets = getInsets();                             // 获得控件的边框
    int radius = 120;                                        // 设置圆的半径是 120px
    int x = (rectangle.width - 2 * radius - insets.left - insets.right) / 2 + insets.left;
    int y = (rectangle.height - 2 * radius - insets.top - insets.bottom) / 2 + insets.top;
    Point2D.Double center = new Point2D.Double(x + radius, y + radius);// 获得圆心坐标
    Point2D.Double[] scales = new Point2D.Double[60];  // 用 60 个点保存表盘的刻度
    double angle = Math.PI / 30;                             // 表盘上两个点之间的夹角是 PI/30
    for (int i = 0; i < scales.length; i++) {                // 获得所有刻度的坐标
        scales[i] = new Point2D.Double();                    // 初始化点对象
        // 利用三角函数计算点的坐标
        scales[i].setLocation(x + radius + radius * Math.sin(angle * i), y + radius - radius * Math.cos(angle * i));
    }
    for (int i = 0; i < scales.length; i++) {                // 画所有刻度
        if (i % 5 == 0) {
                                                 // 如果序号是 5 则画成大点，这些点相当于石英钟上的数字
            g.setColor(Color.RED);
            g.fillOval((int) scales[i].x - 4, (int) scales[i].y - 4, 8, 8);
        } else {
                                                 // 如果序号不是 5 则画成小点，这些点相当于石英钟上的小刻度
            g.setColor(Color.CYAN);
            g.fillOval((int) scales[i].x - 2, (int) scales[i].y - 2, 4, 4);
        }
    }
    Calendar calendar = new GregorianCalendar();             // 创建日期对象
    int hour = calendar.get(Calendar.HOUR);                  // 获得当前小时数
    int minute = calendar.get(Calendar.MINUTE);              // 获得当前分钟数
    int second = calendar.get(Calendar.SECOND);              // 获得当前秒数
    Graphics2D g2d = (Graphics2D) g;
    g2d.setColor(Color.red);                                 // 将颜色设置成红色
    g2d.draw(new Line2D.Double(center, scales[second]));                    // 绘制秒针
```

```
        BasicStroke bs = new BasicStroke(3f, BasicStroke.CAP_ROUND, BasicStroke.JOIN_MITER);
        g2d.setStroke(bs);
        g2d.setColor(Color.blue);                                          // 将颜色设置成蓝色
        g2d.draw(new Line2D.Double(center, scales[minute]));               // 绘制分针
        bs = new BasicStroke(6f, BasicStroke.CAP_BUTT, BasicStroke.JOIN_MITER);
        g2d.setStroke(bs);
        g2d.setColor(Color.green);                                         // 将颜色设置成绿色
        g2d.draw(new Line2D.Double(center, scales[hour * 5 + minute / 12]));  // 绘制时针
    }
```

（3）编写 do_this_windowActivated()方法，用来完成对窗体激活事件的响应。它每隔 1 秒钟就重新绘制一次表盘。将这部分代码放置在一个死循环中就可以实现指针的不断运动。需要注意的是，这些代码要放置在一个新线程中执行，关键代码如下：

```
protected void do_this_windowActivated(WindowEvent e) {
    new Thread() {
        public void run() {
            while (true) {
                repaint();                                  // 绘制时钟
                try {
                    Thread.sleep(1000);                     // 线程休眠 1 秒钟
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        };
    }.start();
}
```

DIY：在本实例的基础上增加一个圆形的表盘。（20 分）（实例位置：光盘\mr\ym\13.21\diy）

13.6.3　情景应用 3——简单的公历万年历

视频讲解：光盘\mr\lx\13\简单的公历万年历.exe

实例位置：光盘\mr\ym\13.22\src

在日常生活中，日历是非常常见的。本实例将使用 Java 语言编写一个简单的公历万年历。在程序运行后会显示当前的日期，通过单击“上个月”、“下个月”按钮可以更新标签上的月份和表格中当前月的天数，实例的运行效果如图 13.24 所示。

图 13.24　实例运行效果

实现过程如下：

（1）新建项目并创建窗体类 PermanentCalendar。

（2）编写方法 updateLabel()，用来获得根据月份增量计算标签上应该显示的时间字符串，关键代码如下：

```
private String updateLabel(int increment) {
    calendar.add(Calendar.MONTH, increment);                       // 将当前月份增加 increment 月
    SimpleDateFormat formatter = new SimpleDateFormat("yyyy 年 MM 月");  // 设置字符串格式
    return formatter.format(calendar.getTime());                   // 获得指定格式的字符串
}
```

（3）编写 updateTable()方法，用来将当前月份的全部天增加到表格中，通过计算确定应该选择哪个单元格作为当前的天。对于表格的表头，可以使用 DateFormatSymbols 类提供的方法快速获取，关键代码如下：

```
private void updateTable(Calendar calendar) {
    String[] weeks = new DateFormatSymbols().getShortWeekdays();// 获得表示星期的字符串数组
    String[] realWeeks = new String[7];                          // 新建一个数组来保存截取后的字符串
    for (int i = 1; i < weeks.length; i++) {
                                                  // weeks 数组第一个元素是空字符串，因此从 1 开始循环
        realWeeks[i - 1] = weeks[i].substring(2, 3);             // 获得字符串的最后一个字符
    }
    int today = calendar.get(Calendar.DATE);                     // 获得当前日期
    int monthDays = calendar.getActualMaximum(Calendar.DAY_OF_MONTH);    // 获得当前月的天数
    calendar.set(Calendar.DAY_OF_MONTH, 1);                      // 将时间设置为本月第一天
    int weekday = calendar.get(Calendar.DAY_OF_WEEK);            // 获得本月第一天是星期几
    int firstDay = calendar.getFirstDayOfWeek();                 // 获得当前地区星期的起始日
    int whiteDay = weekday - firstDay;                           // 这个月第一个星期有几天被上个月占用
    Object[][] days = new Object[6][7];                          // 新建一个二维数组来保存当前月的各天
    for (int i = 1; i <= monthDays; i++) {
                                                  // 遍历当前月的所有天并将其添加到二维数组中
        days[(i - 1 + whiteDay) / 7][(i - 1 + whiteDay) % 7] = i;
    }// 数组的第一维表示一个月中各个星期，第二维表示一个星期中各个天
    DefaultTableModel model = (DefaultTableModel) table.getModel(); // 获得当前表格的模型
    model.setDataVector(days, realWeeks);                        // 给表格模型设置表头和表体
    table.setModel(model);                                       // 更新表格模型
    table.setRowSelectionInterval(0, (today - 1 + whiteDay) / 7);    // 设置选择的行
    table.setColumnSelectionInterval(0, (today - 1 + whiteDay) % 7); // 设置选择的列
}
```

DIY：在本实例的基础上为标签增加提示信息，显示当前的日期。（20 分）（实例位置：光盘\mr\ym\13.22\diy）

13.6.4 情景应用 4——七星彩号码生成器

视频讲解：光盘\mr\lx\13\七星彩号码生成器.exe

实例位置：光盘\mr\ym\13.23\src

七星彩是中国体彩推出的一种彩票，因为 500 万的巨奖而得到众多彩迷的追捧。在投注的机器上提供了机选的功能，该功能可以根据彩迷指定的注数生成随机号码，方便彩迷购买。本实例将实现一个七星彩号码生成器，实例的运行效果如图 13.25 所示。

图 13.25 实例运行效果

实现过程如下：

（1）创建项目并新建窗体类 SevenStar，并在其中添加文本框、按钮等控件。

（2）编写 do_button_actionPerformed()方法，完成对用户单击“生成号码”事件的响应。在该方法中，根据用户在文本框中输入的整数来确定生成号码的个数。对于位数不足 7 位的号码，统一使用前置 0 补齐。需要注意的是，本实例并未校验用户输入的是否是合法的数字。关键代码如下：

```
protected void do_button_actionPerformed(ActionEvent e) {
    int times = Integer.parseInt(textField.getText());                    // 获得用户输入需要生成的中奖号码个数
    StringBuilder sb = new StringBuilder();                               // 利用 StringBuilder 类保存彩票中奖号码
    for (int i = 0; i < times; i++) {
        int number = new Random().nextInt((int) Math.pow(10, 7)); // 生成随机数
        String luckNumber = "" + number;
        while (luckNumber.length() < 7) {
            luckNumber = "0" + luckNumber;                                // 如果随机数长度不够 7 位，则用 0 补齐
        }
        sb.append(luckNumber + "\n");
    }
    textArea.setText(sb.toString());                                      // 显示生成的中奖号码
}
```

DIY：模仿本实例编写排列三号码生成器。（20 分）（实例位置：光盘\mr\ym\13.23\diy）

13.6.5　情景应用 5——大乐透号码生成器

视频讲解：光盘\mr\lx\13\大乐透号码生成器.exe

实例位置：光盘\mr\ym\13.24\src

超级大乐透是中国体彩推出的一种彩票，因为 800 万的巨奖而得到众多彩迷的追捧。在投注的机器上提供了机选的功能，它可以根据彩迷指定的注数生成随机号码，方便彩迷购买。本实例将实现一个超级大乐透号码生成器，实例的运行效果如图 13.26 所示。

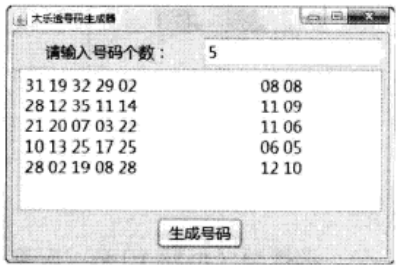


图 13.26　实例运行效果

实现过程如下：

（1）创建项目并新建窗体类 SuperFun，并在其中添加文本框、按钮等控件。

（2）编写 do_button_actionPerformed()方法，完成对用户单击“生成号码”事件的响应。在该方法中，根据用户在文本框中输入的整数来确定生成号码的个数。需要注意的是本实例并未校验用户输入的是否是合法的数字。关键代码如下：

```
protected void do_button_actionPerformed(ActionEvent e) {
    int times = Integer.parseInt(textField.getText());                    // 获得用户输入需要生成的中奖号码个数
    // 省略提示购买数量太多的代码
    StringBuilder sb = new StringBuilder();
    for (int i = 0; i < times; i++) {
        for (int j = 0; j < 5; j++) {                                     // 在 1～35 中随机选择 5 个数字
            List<Integer> list = new ArrayList<Integer>();
            for (int k = 1; k < 36; k++) {
                list.add(k);                                              // 将 1～35 添加到列表中
            }
            int number = list.get(new Random().nextInt(list.size())); // 随机选择一个数字
            String luckNumber = number < 10 ? "0" + number : "" + number;  // 格式化数字
            sb.append(luckNumber + " ");                                  // 向 sb 中增加数字
            list.remove(new Integer(number));                             // 删除选择的数字，这样就避免了重复
```

```
        }
        sb.append("\t\t");
        for (int j = 0; j < 2; j++) {                          // 在 1～12 中随机选择两个数字
            List<Integer> list = new ArrayList<Integer>();
            for (int k = 1; k < 13; k++) {
                list.add(k);                                   // 将 1～12 添加到列表中
            }
            int number = list.get(new Random().nextInt(list.size()));
            String luckNumber = number < 10 ? "0" + number : "" + number;  // 格式化数字
            sb.append(luckNumber + " ");                       // 向 sb 中增加数字
            list.remove(new Integer(number));                  // 删除选择的数字，这样就避免了重复
        }
        sb.append("\n");
    }
    textArea.setText(sb.toString());
}
```

DIY：模仿本实例编写双色球号码生成器。（20 分）（实例位置：光盘\mr\ym\13.24\diy）

情景应用 DIY 栏目分数统计：

DIY 题目	1	2	3	4	5	总分数	
分数							

13.7 自 我 测 试

一、选择题（每题 10 分，5 道题）

1．下列哪些方法是在 Random 类中定义的？（　　）

A．nextByte()　　B．nextShort()　　C．nextInt()　　D．nextLong()

2．如果下面代码的输出结果是 90，则需要使用 Math 类中的哪个方法？（　　）

```
public class Test {
    public static void mainI(String[] args) {
        double number = 90.7;
        System.out.println(method(number));
    }
}
```

A．Math.ceil(a)　　B．Math.floor(a)　　C．Math.max(a)　　D．Math.min(a)

3．当编译和运行下列代码时会发生什么？（　　）

```
public class Test {
    public static void main(String[] args) {
        Math math = new Math();
        System.out.println(math.max(10, 20));
    }
}
```

A．编译错误　　B．运行错误　　C．输出 10　　D．输出 20

4．为了计算余弦值，可以使用 Math 类的哪个方法？（　　）

A．cos()　　B．cosine()　　C．getCos()　　D．getCosine();

5．BigInteger 类位于哪个包中？（　　）

A．java.lang　　B．java.util　　C．java.math　　D．javax.math

二、填空题（每题 10 分，5 道题）

1．以下代码的运行结果是？_____

```
public class Test {
    public static void main(String[] args) {
        Integer number = new Integer(10);
        System.out.println(number + 12);
    }
}
```

2．Math 类是否支持继承？_____

3．java.lang 包中提供了 Math 类，其中有个生成随机数的方法，这个方法生成随机数的范围是？_____

4．对应 DecimalFormat 类而言，#号的含义是？_____

5．下列代码的运行结果是？_____

```
public class Test {
    public static void main(String[] args) {
        System.out.format("%tY", new Date());
    }
}
```

测试分数统计：

类别	第 1 题	第 2 题	第 3 题	第 4 题	第 5 题
选择题分数					
填空题分数					
总分数					

13.8　行 动 指 南

开始日期：_____年___月___日　　结束日期：_____年___月___日

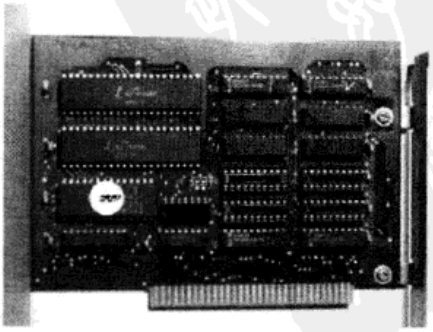
序号	内　容	行 动 指 南	
1	照猫画虎栏目	分数>75 分	优秀，基本功掌握得不错，加油！
		75 分>分数>50 分	及格，知识掌握得不牢，重新做一遍照猫画虎。
	分数（　　）	分数<50 分	请使用光盘中提供的“实战能力测试系统”进行提高训练。
	情景应用栏目	分数>75 分	优秀，综合应用能力很强。
		75 分>分数>50 分	及格，综合应用能力需提高，再练一遍情景应用。
	分数（　　）	分数<50 分	请使用光盘中提供的“实战能力测试系统”进行提高训练。
	自我测试栏目	分数>75 分	优秀，有成为编程高手的潜质。
	分数（　　）	分数<75 分	请使用光盘中提供的“实战能力测试系统”进行提高训练。
	综合评价	反复训练后，以上各项分数都在 75 分以上，方可进入下一堂课学习。	

续表

序号	内　容	行 动 指 南
2	编程实践能力培养：本栏目提供了一些实践题目，对于培养编程能力很有效，要做好。	（1）编写世界时钟，它可以根据指定的国家修改时间。
		（2）编写生日提示工具，它可以显示距离某人过生日还有几天。
		（3）编写农历万年历。
		（4）编写电子时钟。
3	编程习惯培养：本堂课培养读者在学习开发中记笔记的习惯，把开发中遇到的问题、总结的经验记录下来。	
4	创新能力培养：看看身边有没有可以用编程解决的问题，记录到右边的表格中。根据学习进度，尝试编写解决这些问题的小程序。	

13.9　成功可以复制——征途巨人史玉柱的第一桶金

1980 年，史玉柱以优异的成绩考入浙江大学数学系。大学期间，史玉柱坚持每天长跑 18 公里，以此磨练意志和精神，这也为他日后做事百折不挠打下了坚实基础。1984 年毕业，史玉柱被分配到了安徽省统计局农村抽样调查队。业余时间，史玉柱编写了一个程序，过去二三十个人忙活一年的活，一两天就干完了。搞得很多人从此没事干。干得起兴，史玉柱又编了一个分析软件，能分析出不同层次收入的农民可能会买什么东西、消费特征是什么，史玉柱的软件不断完善，以致得到了国家统计局的认可，要求全国各地的农村抽样调查都用史玉柱的软件。史玉柱因此得到了一二十元奖金和一个进步奖。相比当时每月 54 元的工资，史玉柱挺知足。



巨人汉卡图片

1988 年，史玉柱提交了辞职报告，在家潜心编写汉字文字处理软件。1989 年 7 月，史玉柱怀揣独立开发的汉卡软件和“M-6401 桌面排版印刷系统”软盘，南下深圳。当时，除了 4000 元钱，史玉柱一无所有。为了买到当时深圳最便宜的电脑（8500 元），他以加价 1000 元为条件，向电脑商获得推迟付款半个月的“优惠”，赊账得到了平生第一台电脑。为了推广产品，他用同样赊账的办法在《计算机世界》连续做了 3 期 1/4 版的广告。《计算机世界》给史玉柱的付款期限只有 15 天，可一直到广告见报后的第 12 天，史玉柱分文未进。就在关键时刻，第 13 天出现了转机：他一下子收到三张邮局汇款单，总金额 1.582 万元！先人一步的思维方式，让史玉柱迎来了最初的成功。两个月后，他账户上的金额竟达到了 10 万元。他再把钱投入广告中，扩大影

响，4 个月后，仅靠卖 M-6401 产品就回款 100 万元，半年之后回款 400 万元。

1991 年 4 月，史玉柱带着汉卡软件和 100 多名员工来到珠海，注册成立珠海巨人新技术公司。但是刚刚把企业做大的史玉柱感受到了市场的巨大压力，其 M-6402 系列产品受到了来自香港金山电脑的强烈冲击。为了迅速打开市场，史玉柱又做了一次大胆的豪赌——向全国各地的电脑销售商发出邀请，只要订购 10 块巨人汉卡，史玉柱为他们报销路费，让他们前来珠海参加巨人汉卡的全国订货会，这种大胆的营销方式，快速建成了巨人汉卡的营销网络。1991 年，巨人汉卡的销量一跃成为全国同类产品之首，公司获纯利 1000 多万元。1992 年，巨人集团的资本超过 1 亿元，迎来第一个事业高峰，史玉柱也收获了人生的第一桶金。

经典语录

不要认为自己初中水平怎么样，初中水平跟博士后没啥区别。只要能干就行，我一直是这个观点，不在乎学历，只要能干能做出贡献就行。

深度评价

从巨人汉卡到巨人大厦，从脑白金到黄金搭档，史玉柱是具有传奇色彩的创业者之一。他曾经在 5 年时间内跻身财富榜第 8 位；也曾一夜之间负债 2.5 亿；而如今他又是一个著名的东山再起者，保健巨鳄、网游新锐，身价数百亿的企业家。史玉柱的成功，突显出“执著与毅力”的魅力与价值。事业的跌宕起伏、世间的是非议论，唯有敢与苦难作伴的人，才能从跌倒的阴影中爬起来，迈向成功。

第14堂课

常用集合类

（视频讲解：102 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 类的图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。本堂课将向读者介绍 Java 中的各种集合类。

学习摘要：

- 集合类的概念
- List 集合存储对象的特点
- 使用 List 集合保存对象
- Set 集合存储对象的特点
- 使用 Set 集合保存对象
- Map 集合存储对象的特点
- 使用 Map 集合保存对象

14.1 集合类概述

Java 中的集合对象就像是一个容器，用来存放 Java 类的对象。有些容器内部存放的东西容易放入和取出，就像水桶中存放的水，很容易倒入和取出；为了方便存取，可将容器中的对象进行有序的排放，就像衣柜中的衣服，可以有序地排放，以便快速地查找。Java 中的集合类也是如此，有些方便存放和取出，而有些则方便查找。

14.1.1 什么是集合类

java.util 包中提供了一些集合类，这些集合类又被称为容器。提到容器很容易想到数组，集合类与数组不同之处是，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，集合用来存放对象的引用。

14.1.2 Java 集合类的继承关系

常用的集合有 List 集合、Set 集合和 Map 集合。List 与 Set 继承了 Collection 接口，各接口还提供了不同的实现类。集合类的继承关系如图 14.1 所示。

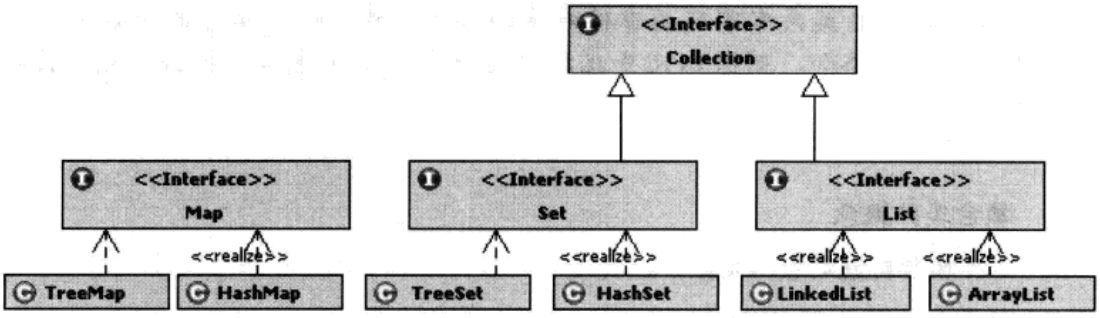


图 14.1 集合类的继承关系

14.2 List 集合

List 集合包括 List 接口和 List 接口的所有实现类。List 集合中的元素允许重复，各元素的顺序就是对象插入的顺序。类似 Java 数组，用户可通过索引（元素在集合中的位置）来访问集合中的元素。

14.2.1 List 接口

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 14.1 所示。

表 14.1　List 接口的常用方法

方　法	返 回 值	功 能 描 述
add(int index,Object obj)	void	用来向集合中指定的索引位置添加对象，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
addAll(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()	ListIterator	用来获得一个包含所有对象的 ListIterator 列表迭代器

从表 14.1 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。

要使用 List 集合，通常情况下要将对象声明为 List 类型，之后通过 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 和 LinkedList。例如：

```
List<String> list = new ArrayList<String>();
List<String> list2 = new LinkedList<String>();
```

要想获取集合中的元素，必须要遍历集合对象，可以通过 for 循环语句遍历集合对象。例如：

```
for(int i = 0;i<list.size();i++){
    System.out.println("i "+list.get(i));
}
```

另一种遍历集合对象的方式是，创建集合对象的迭代器，通过遍历迭代器来获取集合中的元素。例如：

```
Iterator<String> fristIterator = list.iterator();
while (fristIterator.hasNext()) {                    // 遍历集合中的元素
    System.out.print(fristIterator.next()+" ");
}
```

在 List 集合的众多方法中，add(int index,Object obj)方法与 set(int index,E element)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 14.01　创建集合对象，并向该集合中添加元素，通过 set()方法修改集合中的元素，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。实例代码如下：**（实例位置：光盘\mr\ym\14.01）**

```
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()) {                            // 遍历集合中的元素
```

```
            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()) {                             // 循环获取集合中的元素
            System.out.print(it.next() + " ");
        }
    }
}
```

实例的运行效果如图 14.2 所示。

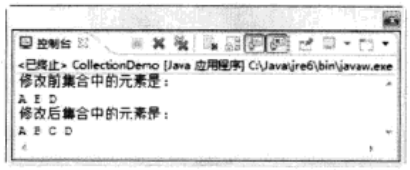


图 14.2　实例运行效果

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，可以使用方法 indexOf()；要想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 14.02　创建 List 集合对象，通过 add()方法向该集合中添加元素，并将对象 apple 和 b 在集合中第一次与最后一次出现的索引位置输出。实例代码如下：**（实例位置：光盘\mr\ym\14.02）**

```
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(list.indexOf(apple));               // 搜索元素在集合中第一次出现的索引位置
        System.out.println(list.lastIndexOf(apple));           // 搜索元素在集合中最后一次出现的索引位置
        System.out.println(list.indexOf(b));
        System.out.println(list.lastIndexOf(b));
    }
}
```

实例的运行效果如图 14.3 所示。

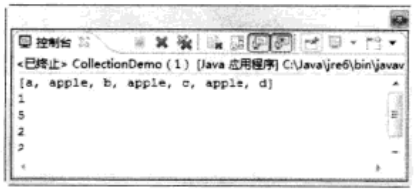


图 14.3　实例运行效果

14.2.2　ArrayList 类

ArrayList 类实现了长度可变的数组的功能，其中可以包括任意类型的元素，如 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，造成这种缺点的原因是向指定索引位置插入对象时，会将指定索引位置及之后的所有对象相应地向后移动一位，如图 14.4 所示。

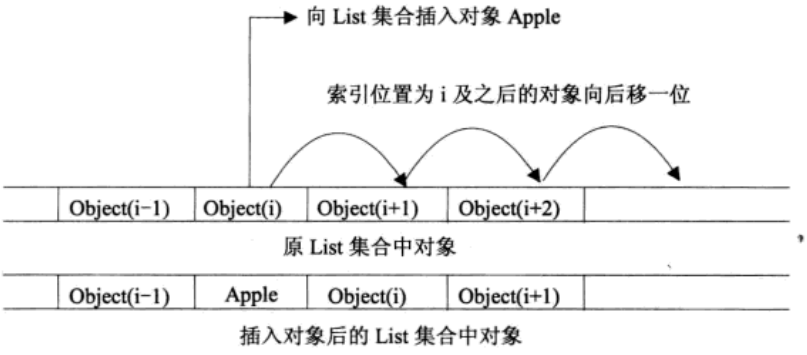


图 14.4　向 List 集合中插入对象

由于 List 集合的结构特点，便于对集合进行快速的随机访问，经常需要根据集合对象的索引位置访问集合，使用 ArrayList 实现集合的效率较高。

例 14.03　创建集合对象，并循环向集合中添加对象，获取小于集合长度的随机数，然后将集合中该数所在的索引位置的对象移出集合，并将集合中索引位置是 10 的对象输出。实例代码如下：**（实例位置：光盘\mr\ym\14.03）**

```
public class CollectionDemo {
    public static void main(String[] args) {
        List<Integer> list = new ArrayList<Integer>();                    // 创建集合对象
        for (int i = (int) 'a'; i <= (int) 'z'; i++) {
            list.add(i);
        }
        int i = (int) (Math.random() * (list.size() - 1));                // 获得小于集合长度的随机数
        System.out.println("随机获取的数为：" + i);
        System.out.println("随机获取集合中的元素：" + list.get(i));
        list.remove(i);                                                   // 将指定索引位置的元素移除集合
        System.out.println("索引位置是'10'的集合中的对象是：" + list.get(10));
    }
}
```

实例的运行效果如图 14.5 所示。

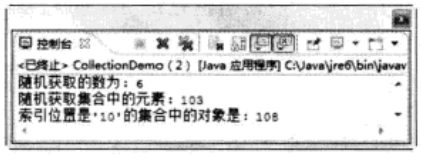


图 14.5　实例运行效果

14.3 Set 集合

Set 集合中的对象不按特定的方式排序，只是简单地把对象加入集合中，但 Set 集合中不能包含重复的对象。

14.3.1 Set 接口

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 接口的常用方法如表 14.2 所示。

表 14.2 Set 接口的常用方法

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	将参数集合中的所有元素添加到此 Set 集合的尾部
remove(Object obj)	boolean	将指定的参数对象移出集合
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	移除 Set 集合中包含在指定 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

要使用 Set 集合，通常情况下要将对象声明为 Set 类型，之后通过 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。例如：

```
Set<String> collSet = new HashSet<String>();
Set<String> collSet2 = new TreeSet<String>();
```

要获取 Set 集合对象，可以通过 iterator()对象获取 Iterator 实例，再通过遍历迭代器来获取集合中的对象。例如：

```
Iterator<String> it = set.iterator();
while(it.hasNext()){
    System.out.println("iiii "+it.next());
}
```

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 集合中来除掉重复值。

例 14.04 创建集合对象，并循环向该集合中添加对象，获取小于集合长度的随机数，然后将集合中该数所在的索引位置的对象移出集合，并将集合中索引位置是 10 的对象输出。实例代码如下：**（实例位置：光盘\mr\ym\14.04）**

```
public class CollectionDemo {
    public static void main(String[] args) {
        List<String> list = new ArrayList<String>();                    // 创建 List 集合对象
        list.add("a");                                                   // 向集合中添加元素
```

```
        list.add("b");
        list.add("c");
        list.add("a");
        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.println(it.next());
        }
    }
}
```

实例的运行效果如图 14.6 所示。

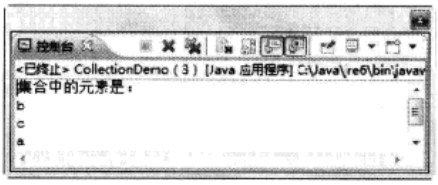


图 14.6　实例运行效果

14.3.2　HashSet 类

HashSet 类实现了 Set 接口，由哈希表（实际上是一个 HashMap 实例）支持。它不保证 Set 的迭代顺序，特别是它不保证该顺序恒久不变。此类允许使用 null 元素。添加到由 HashSet 类实现的 Set 集合中的对象，需要重新实现 equals()方法，从而保证插入集合中对象标识的唯一性。

由于 Set 集合中的对象是无序的，如下面的实例，遍历 Set 集合的结果与插入 Set 集合的顺序并不相同。

例 14.05　遍历输出 HashSet 中的全部元素。实现过程如下：**（实例位置：光盘\mr\ym\14.05）**

（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，并向该集合中添加对象，然后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", 220181));                      // 向集合中添加对象
        hashSet.add(new People("王同学", 220186));
        hashSet.add(new People("李同学", 220193));
        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());
        }
    }
}
```

实例的运行效果如图 14.7 所示。

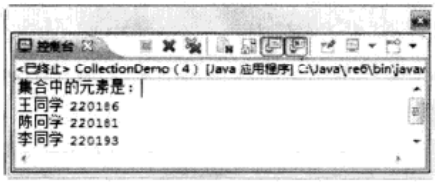


图 14.7　实例运行效果

14.4　Map 集合

Map 没有继承 Collection 接口，其提供的是 key 到 value 的映射。Map 中不能包含相同的 key 值，每个 key 只能映射一个 value。key 还决定了存储对象在映射中的存储位置，但不是 key 对象本身决定的，而是通过一种称为散列技术进行的处理，该技术可产生一个散列码的整数值，散列码通常用作一个偏移量，该偏移量对应分配给映射的内存区域的起始位置，从而来确定存储对象在映射中的存储位置。Map 集合包括 Map 接口和 Map 接口的所有实现类。

14.4.1　Map 接口

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()和 size()等，除此之外，还包括如表 14.3 所示的常用方法。

表 14.3　Map 接口的常用方法

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
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

要使用 Map 集合，通常情况下要将对象声明为 Map 类型，然后通过 Map 接口的实现类来进行实例化，Map 接口的实现类常用的有 HashMap 和 TreeMap 类。例如：

```
Map map = new HashMap<String,String>();
Map collmap2 = new TreeMap<String,String> ();
```

由于 Map 集合中的元素是通过 key 和 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 14.06　遍历 Map 集合中的 value 集合。实例代码如下：**（实例位置：光盘\mr\ym\14.06）**

```
public class CollectionDemo {
    public static void main(String[] args) {
        Map<String, String> map = new HashMap<String, String>();      // 创建 Map 集合
        map.put("01", "apple");                                        // 向集合中添加对象
        map.put("02", "pear");
        map.put("03", "orange");
        Collection<String> list = (Collection<String>) map.values();  // 获取 Map 集合中的所有 value 集合
        Iterator<String> it = list.iterator();                         // 创建集合迭代器
        System.out.println("Map 集合中的对象有：");
        while (it.hasNext()) {                                         // 循环遍历集合
            System.out.println(it.next());
        }
    }
}
```

实例的运行效果如图 14.8 所示。

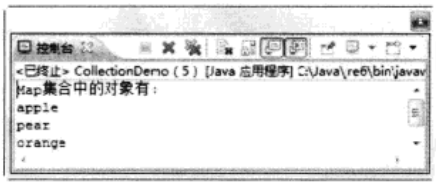


图 14.8　实例运行效果

14.4.2　HashMap 类

HashMap 是基于哈希表的 Map 接口实现。HashMap 通过哈希码对其内部的映射关系进行快速查找。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。此类不保证映射的顺序，特别是它不保证该顺序恒久不变。

例 14.07 创建 Map 集合，通过 HashMap 类实例化该 Map 集合。使用 Map 集合的 get()方法，获取指定键值的集合对象。实现过程如下：**（实例位置：光盘\mr\ym\14.07）**

（1）创建一个作为键对象的类 PK_people，在该类中重写 Object 类的 equals()方法和 hashCode()方法。重写这两个方法的目的是，通过对象的内容进行比较，具体代码如下：

```
public class PK_people {
    private String prefix;                                  // 主键前缀
    private int number;                                     // 主键编号
    public String getPrefix() {                             // 属性的 getXXX()方法
        return prefix;
    }
    public void setPrefix(String prefix) {
        this.prefix = prefix;
    }
    public int getNumber() {
        return number;
    }
    public void setNumber(int number) {
        this.number = number;
    }
    public String getPk() {
        return this.prefix + "_" + this.number;
    }
    public void setPk(String pk) {
        int i = pk.indexOf("_");
        this.prefix = pk.substring(0, i);
        this.number = new Integer(pk.substring(i));
    }
    public int hashCode() {                                 // 重写 hashCode()方法
        return number + prefix.hashCode();
    }
    public boolean equals(Object obj) {                     // 重写 equals()方法
        if (obj == null)                                    // 是否为 null
            return false;
        if (getClass() != obj.getClass())                   // 是否为同一类型的实例
            return false;
        if (this == obj)                                    // 是否为同一个实例
            return true;
        final PK_people other = (PK_people) obj;
        if (this.hashCode() != other.hashCode())            // 判断哈希码是否相等
            return false;
        return true;
    }
}
```

（2）新建 People 类，该类用于添加集合中的 value 值，它包含有 String 与 PK_people 类型对象，并包含有各属性的 setXXX()与 getXXX()方法，代码如下：

```
public class People {
    private String name;
    private PK_people number;
    public People(PK_people number, String name) {          // 构造方法
```

```
        this.number = number;
        this.name = name;
    }
    public String getName() {                                    // 属性对应的 getXXX()方法
        return name;
    }
    public void setName(String name) {                           // 属性对应的 setXXX()方法
        this.name = name;
    }
    public PK_people getNumber() {
        return number;
    }
    public void setNumber(PK_people number) {
        this.number = number;
    }
}
```

（3）新建测试方法，该方法首先新建一个 Map 集合，并向 Map 集合中添加一个映射关系；然后新建一个内容完全相同的键对象，并以该键对象作为参数，通过 get()方法获取相应的值对象，代码如下：

```
public class CollectionDemo {
    public static void main(String[] args) {
        Map<PK_people, People> map = new HashMap<PK_people, People>();
        PK_people pk_person = new PK_people();                   // 新建键对象
        pk_person.setPrefix("tsoft");
        pk_person.setNumber(10);
        map.put(pk_person, new People(pk_person, "陈小姐"));      // 初始化集合
        PK_people pk_person2 = new PK_people(); // 新建键对象，内容与上面键对象的内容完全相同
        pk_person2.setPrefix("tsoft");
        pk_person2.setNumber(10);
        People person2 = map.get(pk_person2);                    // 获得指定键对象映射的值对象
        if (person2 == null)                                     // 如果未得到相应的对象
            System.out.println("该键对象不存在！");
        else
            System.out.println("该对象的 key 值是：" + person2.getNumber().getNumber());
        System.out.println("该对象的 value 值是：" + person2.getName());
    }

}
```

实例的运行效果如图 14.9 所示。

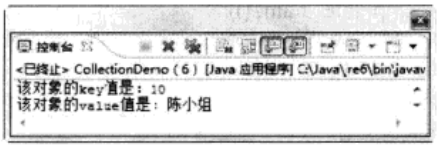


图 14.9　实例运行效果

14.4.3　TreeMap 类

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；否则建议使用 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。

TreeMap 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。

例 14.08 通过 HashMap 类实例化 Map 集合，并遍历该 Map 集合。然后创建 TreeMap 类实例，实现将集合中的元素顺序输出。实现过程如下：**（实例位置：光盘\mr\ym\14.08）**

（1）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;
    }
    public String getE_id() {
        return e_id;
    }
    public void setE_id(String e_id) {
        this.e_id = e_id;
    }
    public String getE_name() {
        return e_name;
    }
    public void setE_name(String e_name) {
        this.e_name = e_name;
    }
}
```

（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，从运行结果可以看出两者的不同，具体代码如下：

```
public class MapDemo {
    public static void main(String[] args) {                          // 主方法
        Map<String, String> map = new HashMap<String, String>();      // 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<String> set = map.keySet();                                // 获取 Map 集合中的 key 对象集合
        Iterator<String> 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<String, String> treemap = new TreeMap<String, String>();  // 创建 TreeMap 集合对象
        treemap.putAll(map);                                              // 向集合中添加对象
        Iterator<String> 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 = (String) map.get(str);                          // 获取集合中的所有 values 值
            System.out.println(str + " " + name);
        }
    }
}
```

实例的运行效果如图 14.10 所示。

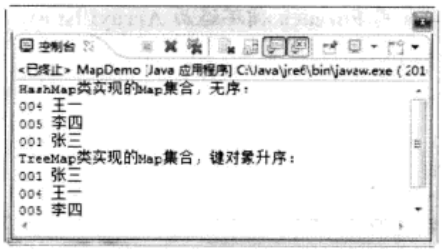


图 14.10　实例运行效果

14.5　照猫画虎——基本功训练

14.5.1　基本功训练 1——使用普通 for 循环遍历 ArrayList

视频讲解：光盘\mr\lx\14\使用普通 for 循环遍历 ArrayList.exe

实例位置：光盘\mr\ym\14.09\src

在使用集合类时，不仅要关心容器是如何保存元素的，而且要关心如何取出元素。本实例将使用普通 for 循环遍历 ArrayList，从中取出所有序号为奇数的元素。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);                       // 输出列表中的全部元素
        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)+"   ");
        }
    }
}
```

实例的运行效果如图 14.11 所示。

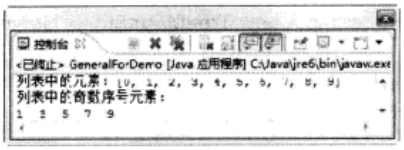


图 14.11 实例运行效果

照猫画虎：使用普通 for 循环输出列表中序号为偶数的元素。（20 分）（实例位置：光盘\mr\ym\14.09\zmhh）

14.5.2 基本功训练 2——使用 Foreach 循环遍历 ArrayList

视频讲解：光盘\mr\lx\14\使用 Foreach 循环遍历 ArrayList.exe

实例位置：光盘\mr\ym\14.10\src

在 Java SE 5.0 版中，新增了 Foreach 循环，它也可以遍历集合，而且代码更加简单明了。本实例将使用其遍历 ArrayList。代码如下：

```
public class EnhanceForDemo {
    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 (Integer i : list) {                                          // 输出列表中的全部元素
            System.out.print(i + " ");
        }
    }
}
```

实例的运行效果如图 14.12 所示。

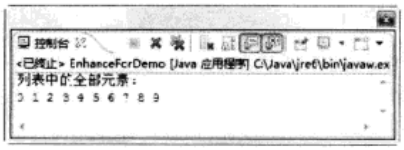


图 14.12 实例运行效果

技巧：使用 Foreach 循环并不能控制遍历的位置，此时可以考虑使用 continue 的跳转语句来控制元素的获得。

照猫画虎：使用 Foreach 循环输出列表中序号为偶数的元素。（20 分）（实例位置：光盘\mr\ym\14.10\zmhh）

14.5.3 基本功训练 3——使用 Iterator 遍历 ArrayList

视频讲解：光盘\mr\lx\14\使用 Iterator 遍历 ArrayList.exe

实例位置：光盘\mr\ym\14.11\src

在正文中，演示了如何使用 Iterator 和 while 循环组合遍历集合。本实例将使用 Iterator 和 for 循环组合

遍历集合。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()+" ");
        }
    }
}
```

实例的运行效果如图 14.13 所示。

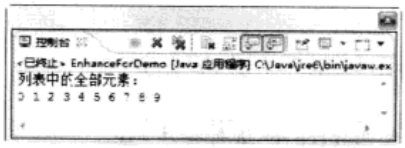


图 14.13　实例运行效果

照猫画虎：使用 Iterator 和 for 循环输出列表中序号为偶数的元素。（20 分）（实例位置：光盘\mr\ym\14.11\zmhh）

14.5.4　基本功训练 4——使用 ListIterator 逆序遍历 ArrayList

视频讲解：光盘\mr\lx\14\使用 ListIterator 逆序遍历 ArrayList.exe

实例位置：光盘\mr\ym\14.12\src

对于列表而言，除了 Iterator，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。本实例将使用其逆序遍历 ArrayList。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() + " ");
        }
    }
}
```

实例的运行效果如图 14.14 所示。

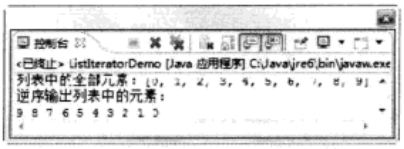


图 14.14　实例运行效果

你问我答：hasPrevious()和 previous()方法的作用各是什么？

ListIterator 接口继承了 Iterator 接口并且对其进行了加强。hasPrevious()方法用于判断当前游标所处的位置前是否还有一个元素，previous()方法用于将游标移动到前一个元素。它们和 hasNext()、next()方法是相当的。

照猫画虎：使用 ListIterator 和 for 循环逆序输出列表中序号为奇数的元素。（20 分）（实例位置：光盘\mr\ym\14.12\zmhh）

14.5.5　基本功训练 5——使用 TreeSet 进行排序

视频讲解：光盘\mr\lx\14\使用 TreeSet 进行排序.exe

实例位置：光盘\mr\ym\14.13\src

在正文中演示了 TreeSet 类的最简单用法，本实例将使用其完成逆序排序的功能。

实现过程如下：

（1）通过实现 Comparator 接口来定义排序方式，代码如下：

```
public class Descending implements Comparator<Integer> {
    @Override
    public int compare(Integer o1, Integer o2) {// 定义排序方式
        return o2 - o1;
    }
}
```

（2）使用指定的排序方式创建 TreeSet 类型对象，代码如下：

```
public class TreeSetDemo {
    public static void main(String[] args) {
        Set<Integer> set = new TreeSet<Integer>(new Descending());
        for (int i = 0; i < 10; i++) {                          // 向集合中添加递增数据
            set.add(i);
        }
        System.out.println("集合中元素的逆序排列：");
        System.out.println(set);
    }
}
```

实例的运行效果如图 14.15 所示。

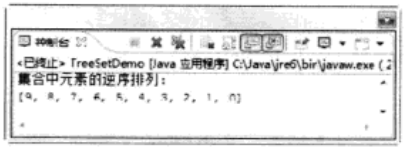


图 14.15　实例运行效果

你问我答：如何定义排序方式？

通过实现 Comparator 接口即可定义排序方式。该接口使用了泛型，可以简化编程。实现该接口时需要重写 compare()方法。该方法有两个参数，如果第一个参数大于第二个并且希望升序排序，则返回一个正数；如果两者相等，则返回 0；如果第一个参数小于第二个并且希望升序排序，则返回一个负数。

照猫画虎：使用 TreeSet 对字符串实现逆序排序。（20 分）（实例位置：光盘\mr\ym\14.13\zmhh）

照猫画虎栏目分数统计：

照猫画虎题目	1	2	3	4	5	总分数	
分数							

14.6　情景应用——拓展与实践

14.6.1　情景应用 1——用动态数组保存学生姓名

视频讲解：光盘\mr\lx\14\用动态数组保存学生姓名.exe

实例位置：光盘\mr\ym\14.14\src

集合类中的 ArrayList 由于具有自由扩展容量的能力，因此是 Java 数组很好的替代品。本实例将使用其保存学生姓名，用户新输入的姓名和已经输入的姓名将统一在一个列表控件中显示。实例的运行效果如图 14.16 所示。

图 14.16　实例运行效果

实现过程如下：

（1）在项目中新建窗体类 StudentList。在窗体中添加文本框控件、列表框控件和“保存”按钮。

（2）编写“保存”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，最后更新列表框控件模型。关键代码如下：

```
protected void do_button_actionPerformed(ActionEvent e) {
    String name = textField.getText();                          // 获得用户输入的文本信息
    names.add(name);                                            // 在列表框中增加文本
    list.setModel(new AbstractListModel() {
        private static final long serialVersionUID = -8677773874352259166L;
        @Override
        public int getSize() {                                  // 设置数据模型的长度
            return names.size();
        }
        @Override
        public Object getElementAt(int index) {                // 设置数据模型的索引
            return names.get(index);
        }
    });
}
```

DIY：在本实例的基础上实现删除学生信息的方法。（20 分）（实例位置：光盘\mr\ym\14.14\diy）

14.6.2 情景应用 2——用 List 集合传递学生信息

视频讲解：光盘\mr\lx\14\用 List 集合传递学生信息.exe

实例位置：光盘\mr\ym\14.15\src

使用集合类的一个好处是传递信息非常方便。假如需要传递 100 条记录，分开传递可能需要 100 次，如果使用集合类仅需要一次。本实例将演示如何使用列表传递学生信息，实例的运行效果如图 14.17 所示。

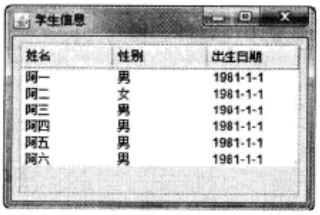


图 14.17 实例运行效果

实现过程如下：

（1）在项目中新建窗体类 StudentsInfo。在窗体中添加滚动面板，在该面板中将放置表格控件。

（2）编写 do_this_windowActivated()方法完成窗体激活事件监听。在该方法中使用列表保存学生信息，然后使用 for 循环将各个信息增加到表格模型中，最后更新表格模型。关键代码如下：

```
protected void do_this_windowActivated(WindowEvent e) {
    List<String> list = new ArrayList<String>();
    list.add("阿一,男,1981-1-1");                                    // 添加数据到集合对象
    list.add("阿二,女,1981-1-1");                                    // 添加数据到集合对象
    list.add("阿三,男,1981-1-1");                                    // 添加数据到集合对象
    list.add("阿四,男,1981-1-1");                                    // 添加数据到集合对象
    list.add("阿五,男,1981-1-1");                                    // 添加数据到集合对象
    list.add("阿六,男,1981-1-1");                                    // 添加数据到集合对象
    DefaultTableModel model = (DefaultTableModel) table.getModel();  // 获得表格模型
    model.setColumnIdentifiers(new Object[] { "姓名", "性别", "出生日期" });  // 为表格设置表头
    for (String info : list) {                                       // 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
    }
    table.setModel(model);                                           // 更新表格模型
}
```

DIY：模仿本实例，使用 Map 来传递信息并在表格中显示。（20 分）（实例位置：光盘\mr\ym\14.15\diy）

14.6.3 情景应用 3——使用栈逆序输出字符串

视频讲解：光盘\mr\lx\14\使用栈逆序输出字符串.exe

实例位置：光盘\mr\ym\14.16\src

栈是一种先入后出的数据结构，它具有非常重要的应用。本实例将使用该结构实现字符串的逆序输出，实例的运行效果如图 14.18 所示。

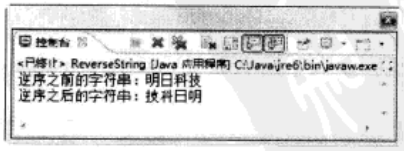


图 14.18 实例运行效果

实现过程如下：

（1）新建项目并创建 ReverseString 类。

（2）在 ReverseString 类的主方法中创建 Stack 类对象。将

字符串“明日科技”按字符的顺序放入到栈中，然后使用 pop()方法将各个字符从栈中弹出，由于进入和弹出的顺序刚好相反，因此输出的字符串是逆序的。代码如下：

```
public class ReverseString {
    public static void main(String[] args) {
        Stack<Character> stack = new Stack<Character>();          // 创建栈对象
        String text = "明日科技";                                  // 创建需要逆序的字符串
        char[] array = text.toCharArray();                        // 将字符串转换成 char 类型的数组
        for (char c : array) {                                    // 将数组中的元素入栈
            stack.push(c);
        }
        System.out.println("逆序之前的字符串：" + text);
        System.out.print("逆序之后的字符串：");
        while (!stack.isEmpty()) {                                // 将字符出栈
            System.out.print(stack.pop());
        }
    }
}
```

你问我答：如何使用 push()和 pop()方法？

Stack 类是 Java API 提供的工具类，它可以实现栈的功能。push()方法提供了入栈操作，pop()方法提供了出栈操作。

注意：使用 pop()方法时元素被弹出了栈。

DIY：使用栈模拟盘子的使用过程。（20 分）（实例位置：光盘\mr\ym\14.16\diy）

14.6.4　情景应用 4——制作电子词典

视频讲解：光盘\mr\lx\14\制作电子词典.exe

实例位置：光盘\mr\ym\14.17\src

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。本实例将制作一个电子词典，实例的运行效果如图 14.19 所示。

图 14.19　实例运行效果

实现过程如下：

（1）在项目中新建窗体类 DictionaryDemo。在窗体中添加标签等控件。

（2）编写 do_this_windowActivated()方法完成窗体激活事件监听。通过该方法向 Map 中增加数据，关键代码如下：

```
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。在该方法中完成对单词的查询操作，并根据不同的情况进行提示，关键代码如下：

```
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
    }
}
```

DIY：在本实例的基础上实现增加新单词的功能。（20 分）（实例位置：光盘\mr\ym\14.17\diy）

14.6.5 情景应用 5——制作手机电话簿

视频讲解：光盘\mr\lx\14\制作手机电话簿.exe

实例位置：光盘\mr\ym\14.18\src

为了便于保存联系方式，手机都提供有电话簿功能。它也是一种映射关系，因此可以使用 Map 来实现。本实例将制作一个手机电话簿，实例的运行效果如图 14.20 所示。

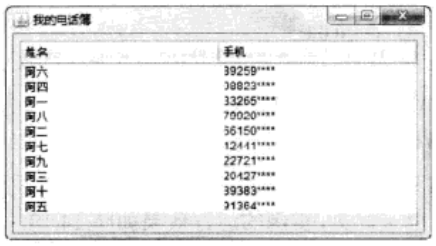


图 14.20 实例运行效果

实现过程如下：

（1）在项目中新建窗体类 Phonebook。在窗体中添加表格控件。

（2）编写 do_this_windowActivated()方法完成窗体激活事件监听。通过该方法向表格模型中增加数据，最后更新表格模型。关键代码如下：

```
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
    }
```

你问我答：keyset()方法有什么作用？

对于映射而言，可以使用该方法获得所有的键，它们组成了一个 Set。再利用 Iterator 提供的方法遍历 Set 就能获得所有的键值对。

DIY：使用列表实现电话簿的功能。（20 分）（实例位置：光盘\mr\ym\14.18\diy）

情景应用 DIY 栏目分数统计：

DIY 题目	1	2	3	4	5	总分数	
分数							

14.7　自 我 测 试

一、选择题（每题 10 分，5 道题）

1．下列哪些方法是在 Collection 接口中定义的？（　　）

A．iterator()　　B．isEmpty()　　C．toArray()　　D．setText()

2．如果希望数据有序存储并且便于修改，可以使用哪种 Collection 接口的实现类？（　　）

A．LinkedList　　B．ArrayList　　C．TreeMap　　D．HashSet

3．如果希望数据有序存储并且便于查询，可以使用哪种 Collection 接口的实现类？（　　）

A．LinkedList　　B．ArrayList　　C．TreeMap　　D．HashSet

4．如果希望遍历 Set 中的全部元素，可以使用哪种方式？（　　）

A．普通 for 循环　　B．Foreach 循环　　C．Iterator　　D．SetIterator

5．下列代码的运行结果是？（　　）

```
public class Test {
    public static void main(String[] args) {
        List<Integer> list = new ArrayList<Integer>();
        Iterator<Integer> it = list.iterator();
        System.out.println(it.next());
    }
}
```

A．0　　B．抛出异常　　C．编译错误　　D．运行错误

二、填空题（每题 10 分，5 道题）

1．哪个接口的实现类不能够保存相同的对象？_____

2．HashMap 是否实现了 Collection 接口？_____

3．使用 Map 接口中定义的哪个方法可以获得全部键组成的 Set_____

4．如果指定的键在 Map 中并没有与其对应的值，则 get()方法的返回值是_____。

5．实现元素的先进后出，可以使用栈结构。在 Java API 中与之相对应的类是_____。

测试分数统计：

类别	第 1 题	第 2 题	第 3 题	第 4 题	第 5 题
选择题分数					
填空题分数					
总分数					

14.8 行 动 指 南

开始日期：_____年____月____日　　结束日期：_____年____月____日

序号	内　容	行 动 指 南	
1	照猫画虎栏目 分数（　　）	分数>75 分	优秀，基本功掌握得不错，加油！
		75 分>分数>50 分	及格，知识掌握得不牢，重新做一遍照猫画虎。
		分数<50 分	请使用光盘中提供的“实战能力测试系统”进行提高训练。
	情景应用栏目 分数（　　）	分数>75 分	优秀，综合应用能力很强。
		75 分>分数>50 分	及格，综合应用能力需提高，再练一遍情景应用。
		分数<50 分	请使用光盘中提供的“实战能力测试系统”进行提高训练。
	自我测试栏目 分数（　　）	分数>75 分	优秀，有成为编程高手的潜质。
		分数<75 分	请使用光盘中提供的“实战能力测试系统”进行提高训练。
	综合评价	反复训练后，以上各项分数都在 75 分以上，方可进入下一堂课学习。	
2	编程实践能力培养：本栏目提供了一些实践题目，对于培养编程能力很有效，要做好。	（1）实现省市级联选择框。	
		（2）获得升序排列无重复元素的随机数集合。	
		（3）编程实现二分法查找数据。	
		（4）编程实现冒泡法查找数据。	
3	编程习惯培养：本堂课培养读者在学习开发中记笔记的习惯，把开发中遇到的问题、总结的经验记录下来。		
4	创新能力培养：看看身边有没有可以用编程解决的问题，记录到右边的表格中。根据学习进度，尝试编写解决这些问题的小程序。		

14.9　成功可以复制——使计算机成为生活的必需品

比尔·盖茨出生于 1955 年 10 月 28 日，与两个姐姐一起在美国西雅图长大。他们的父亲是西雅图的一名律师，母亲是学校的一名教师。比尔从小就精力过人，早在婴儿时期自己就能让摇篮晃动起来，从小就极爱思考，一迷上某事便能全身心投入。外祖母经常和小比尔玩游戏，尤其是一些智力游戏，在外祖母的帮助与指导下，比尔成了兴趣广泛、废寝忘食的读者——读书成了他打发精力的好方式。随着儿子年龄的增长，家庭中的环境已无法满足比尔·盖茨天赋的进一步发挥。小比尔有时还会责备母亲智力不足呢！于是，父母把目光投向社会，积极为比尔寻找属于他的空间。小学毕业后，父母在征求比尔·盖茨的意见后，送他进了湖滨中学。在湖滨中学，比尔痴迷上令他此后倾注毕生精力的计算机。

比尔·盖茨在湖滨中学读书时，常按自己的兴趣爱好来安排学习。比尔·盖茨在喜欢的课程上下功夫，学得非常棒，如数学和阅读方面。每次父母看到比尔拿回来的成绩单，尽管他们知道比尔在一些课程上会学得更好，但他们并没有拉下脸来责备比尔·盖茨。

中学毕业后，比尔·盖茨很想到哈佛大学去读书，这也正是父母们最大的心愿。比尔·盖茨的父母是非常开明和民主的，他们并没有要求孩子们完成自己喜欢的事。经过冷静的思考之后，父母放弃了让儿子当律师的想法，让比尔·盖茨在大学领域里自由地发展。这一点帮了比尔·盖茨的大忙。

1975 年，年仅 19 岁的盖茨就预言："我们意识到软件时代到来了，并且对于芯片的长期潜能我们有足够的洞察力，这意味着什么？我现在不去抓住机会反而去完成我的哈佛学业，软件工业绝对不会原地踏步等着我。"所以比尔·盖茨决定离开哈佛大学，放弃锦绣的学业，与别人一起创办计算机公司。为了征得父母的同意，比尔与父母多次交谈，平静地表达了自己的想法。了解儿子秉性和志向的父母又能说什么呢？或许儿子的天赋与计算机事业是最佳的切合点吧，比尔·盖茨便毅然离开了令亿万学子向往的哈佛大学，开始在软件领域大展鸿图。

微软总部

如今，如果你的办公桌上有一台个人电脑，里面几乎都装有微软的操作系统。比尔·盖茨使个人计算机成了日常生活用品，并因而改变了每一个现代人的工作、生活乃至交往方式。

因此有人说，比尔·盖茨对软件的贡献，就像爱迪生发明了灯泡。

经典语录

不要让这个世界的复杂性阻碍你前进，要成为一个行动主义者。将解决人类的不平等视为己任，它将成为你生命中最重要的经历之一。

深度评价

比尔·盖茨在 19 岁时曾经预言："我们意识到软件时代到来了，并且对于芯片的长期潜能我们有足够的洞察力，这意味着什么？我现在不去抓住机会反而去完成我的哈佛学业，软件工业绝对不会原地踏步等着我。"他的经历告诉我们：一个人想要成功，就要学会在机遇从头顶上飞过时跳起来抓住它。这样逮到机遇的机会就会增大。一旦做出决定就不要拖延。任何事情想到就去做，立即行动！

高级篇

- 第 15 堂课　数据库编程应用
- 第 16 堂课　输入输出流与文件
- 第 17 堂课　网络程序设计
- 第 18 堂课　表格控件的应用
- 第 19 堂课　树控件的应用
- 第 20 堂课　其他高级控件
- 第 21 堂课　图形绘制技术

第15堂课

数据库编程应用

（视频讲解：122分钟）

数据库系统是由数据库、数据库管理系统和应用系统、数据库管理员构成的。数据库管理系统简称DBMS，是数据库系统的关键组成部分，包括数据库定义、数据查询、数据维护等。而JDBC技术是连接数据库与应用程序的纽带。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。本堂课将向读者介绍Java语言中数据库操作部分。

学习摘要：

- 数据库的概念
- 数据库的种类及功能
- JDBC中常用的类和接口
- 数据库操作的步骤
- 利用JDBC技术顺序查询数据的方法
- 利用JDBC技术实现模糊查询
- 利用JDBC技术实现对数据的添加、修改、删除

15.1 JDBC 技术

程序设计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。

15.1.1 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时重新输入它们。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。
- ☑ 数据的一致性和可维护性，可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：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 15.1 所示。

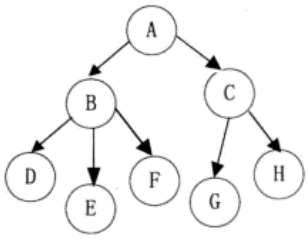


图 15.1 层次型数据库

☑ 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，但编写应用程序时程序员必须熟悉数据库的逻辑结构。网状型数据库如图 15.2 所示。

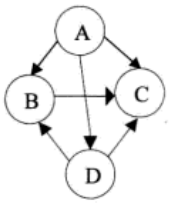


图 15.2　网状型数据库

☑ 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，关系模型是由一系列表格组成的。

15.1.2　JDBC-ODBC 技术介绍

JDBC 是一种用于执行 SQL 语句的 Java API（Application Programming Interface，应用程序设计接口）。它是连接数据库和 Java 应用程序的一个纽带。JDBC-ODBC 桥是一个 JDBC 驱动程序，完成了从 JDBC 操作到 ODBC 操作之间的转换工作。由于 ODBC 技术被广泛地使用，因此 Java 利用 JDBC-ODBC 桥可访问几乎所有的数据库。下面介绍一种通过 JDBC-ODBC 桥连接数据库的方法，通过这种方法无须手动配置 ODBC 数据源，而是采用默认的 ODBC 数据源。具体步骤如下：

（1）加载 JDBC-ODBC 桥的驱动程序。代码如下：

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

Class 类是 java.lang 包中的一个类，通过该类的静态方法 forName()可加载 sun.jdbc.odbc 包中的 JdbcOdbcDriver 类，来建立 JDBC-ODBC 桥连接器。

（2）使用 java.sql 包中的 Connection 接口，并通过 DriverManager 类的静态方法 getConnection()创建连接对象。格式如下：

```
Connection conn = DriverManager.getConnection("jdbc:odbc:数据源名字" , "user name" , "password");
```

📖 **说明：数据源名称必须是配置成功的，否则不会连接成功。**

数据源必须给出一个简短的描述名。假设没有设置 user name 和 password，要与数据源 tom 交换数据，建立 Connection 对象的代码如下：

```
Connection conn = DriverManager.getConnection("jdbc.odbc:tom","","");
```

（3）向数据库发送 SQL 语句。使用 Statement 接口声明一个 SQL 语句对象，并通过刚才创建的连接数据库对象 conn 的 createStatement()方法创建这个 SQL 对象。代码如下：

```
Statement sql = conn.createStatement();
```

JDBC-ODBC 桥作为连接数据库的过渡性技术，已经不被 Java 技术广泛应用，现在广泛应用的是 JDBC 技术。但并不表示 JDBC-ODBC 桥技术已经被淘汰。JDBC-ODBC 桥作为 sun.jdbc.odbc 包与 JDK 一起自动安装，不需要特殊配置。

15.1.3 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，需要依赖于数据库厂商提供的 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，可以完全专著于业务逻辑的开发。
- ☑ 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。

15.2 JDBC 中常用的类和接口

JDBC 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。本节将向读者介绍 JDBC 技术中常用的类和接口，这些类和接口都位于 java.sql 包中。

15.2.1 DriverManager 类

DriverManager 类用来管理数据库中的所有驱动程序，它是 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 15.1 所示。

表 15.1 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、用户名和密码，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

15.2.2 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 15.3 所示。

图 15.3 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 15.2 所示。

表 15.2　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

15.2.3　Statement 接口

Statement 实例用于在已经建立连接的基础上向数据库发送 SQL 语句。Statement 接口用来执行静态的 SQL 语句。例如，执行 INSERT、UPDATE 和 DELETE 语句，可调用该接口的 executeUpdate()方法；执行 SELECT 语句，可调用该接口的 executeQuery()方法。

Statement 实例可以通过 Connection 实例的 createStatement()方法获取。例如，获取 Statement 实例，代码如下：

```
Connection conn = DriverManager.getConnection("url", "userName", "password");
Statement statement = conn.createStatement();
```

Statement 接口的常用方法如表 15.3 所示。

表 15.3　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()	执行给定的 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

15.2.4 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。可以通过 Connection 类的 preparedStatement()方法获取 PreparedStatement 对象。例如：

```
Connection conn = DriverManager.getConnection("url", "userName", "passWord");
PreparedStatement preps = conn.prepareStatement(sql);
```

参数 sql 为待执行的 SQL 语句。PreparedStatement 接口的常用方法如表 15.4 所示。

表 15.4 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)	将参数的 pIndex 位置设置为给定的 byte 型参数值 bt
setDouble(int pIndex,double dou)	将参数的 pIndex 位置设置为给定的 double 型参数值 dou
setInt(int pIndex,int x)	将参数的 pIndex 位置设置为给定的 int 型参数值 x
setObject(int pIndex,Object o)	将参数的 pIndex 位置设置为给定的 Object 型参数值 o
setString(int pIndex,String str)	将参数的 pIndex 位置设置为给定的 String 型参数值 str

15.2.5 ResultSet 接口

ResultSet 接口类似于一张数据表，用来暂时存放数据库查询操作所获得的结果集。ResultSet 实例具有指向当前数据行的指针。指针开始的位置在查询结果集第一条记录的前面。在获取查询结果集时，可通过 next()方法将指针向下移。如果存在下一行，该方法返回 true，否则返回 false。

ResultSet 接口提供了从当前行检索不同类型值的 getXXX()方法，通过该方法的不同重载形式，可实现分别通过列的索引编号和列的名称检索列值。此外，该接口提供了一组更新方法 updateXXX()，可通过列的索引编号和列的名称，更新当前行的指定列。但是该方法并未将操作同步到数据库中，需要执行 updateRow()或 insertRow()方法完成同步操作。该接口的常用方法如表 15.5 所示。

表 15.5 ResultSet 接口的常用方法

方　法	说　明
getInt()	以 int 形式获取此 ResultSet 对象的当前行中的指定列值。如列值是 NULL，则返回值是 0
getFloat()	以 float 形式获取此 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 Data 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

续表

方　法	说　明
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

15.3　数据库连接

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。

15.3.1　加载数据库驱动

1．数据库驱动类

怎么来理解数据库驱动呢？就如有的机器安装摄像头时，需要安装摄像头的驱动；有的机器安装 U 盘就需要安装相应的 U 盘驱动，这样机器才能识别相应的设备。数据库的驱动类似于摄像头的驱动或者 U 盘的驱动，当机器上安装了相应的数据库后，需要安装相应的数据库驱动才能使机器识别这种数据库（如 MySQL、SQL Server）。

Java 程序怎样去调用数据库呢？Sun 公司提供了 JDBC 技术，用于与数据库建立联系，但需要注意的是只提供了接口，由数据库提供商实现这些接口，就是所谓的数据库驱动。

由于不同的数据库厂商实现 JDBC 接口不同，因此就产生了不同的数据库驱动包。数据库驱动包中包含了一些类，它们负责与数据库建立连接，把一些 SQL 语句传到数据库中去。例如，Java 程序实现与 SQL Server 2000 数据库建立连接，需要在程序中加载驱动包 msbase.jar、mssqlserver.jar、msutil.jar 或 jtds.jar，与 MySQL 数据库建立连接需要在程序中加载驱动包 mysql-connectot-java.jar 等。

2．加载数据库驱动类

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()。

forName()方法的语法如下：

```
Class.forName(String driverManager)
```

forName()方法参数指定要加载的数据库驱动，加载成功后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。

例如，加载 MySQL 数据库驱动，代码如下：

```
try {
    Class.forName("com.mysql.jdbc.Driver"); //加载数据库驱动
} catch (ClassNotFoundException e) {
    e.printStackTrace();
}
```

15.3.2 创建数据库连接

在进行数据库操作时，只需要第一次访问数据库时加载数据库驱动。然后每次访问数据库时，创建一个 Connection 对象，之后执行操作数据库的 SQL 语句。可通过 DriverManager 类的 getConnection()方法创建 Connection 实例。该方法的声明如下：

```
public static Connection getConnection(String url,String user,String password)throws SQLException
```

参数说明：url 表示指定数据库的 URL，user 表示指定数据库的用户名，password 表示指定数据库的密码。

1. MySQL 数据库的连接

在当前比较流行的数据库中，MySQL 数据库是一个完全开放源代码的数据库软件。它具有功能强、使用简便、管理方便、运行速度快、安全可靠性强等优点，同时具有客户机/服务器体系结构的分布式数据库管理系统，是完全网络化的跨平台关系型数据库系统。MySQL 数据库也是 Java 程序开发者常用的数据库之一。要建立与 MySQL 数据库的连接，首先要在程序中加载 MySQL 数据库的驱动，可以到 MySQL 的官方网站上下载，网址为 http://www.mysql.org。

例 15.01 创建类 GetConn，在该类中实现与 MySQL 数据库建立连接，连接的数据库名称为 db_database15。实例代码如下：**（实例位置：光盘\mr\ym\15.01）**

```
public class GetConn {
    public Connection conn = null;                                          // 创建 Connection 对象
    public Connection getConnection() {                                     // 获取数据库连接方法
        try {
            Class.forName("com.mysql.jdbc.Driver");                         // 加载数据库驱动
            String url = "jdbc:mysql://localhost:3306/db_database15";       // 指定连接数据库的 URL
            String user = "root";                                           // 指定连接数据库的用户名
            String passWord = "111";                                        // 指定连接数据库的密码
            conn = DriverManager.getConnection(url, user, passWord);
            if (conn != null) {                                             // 如果 Connection 实例不为空
                System.out.println("数据库连接成功");                        // 提示信息
            }
        } catch (Exception e) {
            e.printStackTrace();
        }                                                                   // 异常处理
        return conn;                                                        // 返回 Connection 对象
    }
    public static void main(String[] args) {                                // 程序主方法
        GetConn getConn = new GetConn();                                    // 创建 GetConn 对象
        getConn.getConnection();                                            // 调用连接数据库方法
    }
}
```

实例运行效果如图 15.4 所示。

图 15.4　实例运行效果

注意：在使用本程序前，需要向项目中导入 MySQL 数据库的驱动。

2．连接 SQL Server 2005 数据库

连接 SQL Server 2005 数据库应用的驱动程序为 sqljdbc.jar，可以到 Microsoft 的官方网站上下载，地址为 http://www.microsoft.com。

例 15.02　创建类 GetConn，在该类中实现与 SQL Server 2005 数据库建立连接，连接的数据库名称为 db_database15。实例代码如下：**（实例位置：光盘\mr\ym\15.02）**

```
public class GetConn {
    Connection con;
    public Connection getConnection() {                          // 建立返回值为 Connection 的方法
        try {
            Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");
            con = DriverManager.getConnection("jdbc:sqlserver://localhost:1433;
                                                    databaseName=db_database15", "sa", "|||");
            if (con != null) {
                System.out.println("数据库连接成功");
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        return con;                                              // 按方法要求返回 Connection 对象
    }
    public static void main(String[] args) {
        GetConn getConn = new GetConn();                         // 创建本类对象
        getConn.getConnection();                                 // 调用数据库连接方法
    }
}
```

实例运行效果如图 15.5 所示。

图 15.5　实例运行效果

注意：在使用本程序前，需要向项目中导入 SQL Server 2005 数据库的驱动。

3．其他常见数据库的驱动和 URL

JDBC 的一个很大的优势就是截断了应用程序和底层数据库之间的联系。通过修改数据库的驱动和 URL 信息及其相关的驱动文件，就可以更换数据库。各种常见数据库的驱动和 URL 如表 15.6 所示。

表 15.6 常见数据库的驱动和 URL 信息

数 据 库 名	驱 动	URL
Oracle Thin	oracle.jdbc.driver.OracleDriver	jdbc:oracle:thin:@<HOST>:<PORT>:<SID>
PostgreSQL (v6.5 and earlier)	postgresql.Driver	jdbc:postgresql://<HOST>:<PORT>/<DB>
PostgreSQL (v7.0 and later)	org.postgresql.Driver	jdbc:postgresql://<HOST>:<PORT>/<DB>
Sybase (jConnect 4.2 and earlier)	com.sybase.jdbc.SybDriver	jdbc:sybase:Tds:<HOST>:<PORT>
Sybase (jConnect 5.2)	com.sybase.jdbc2.jdbc.SybDriver	jdbc:sybase:Tds:<HOST>:<PORT>

📖 说明：在使用其他数据库时也需要去其官方网站下载数据库驱动程序。

15.3.3 向数据库发送 SQL 语句

建立数据库连接的目的是与数据库进行通信，实现方式为执行 SQL 语句，但 Connection 实例并不能执行 SQL 语句。此时需要通过 Connection 接口的 createStatement()方法获取 Statement 对象。

例如，创建 Statement 对象 state，代码如下：

```
try {
    Statement state = con.createStatement();
} catch (SQLException e) {
    e.printStackTrace();
}
```

15.3.4 获取查询结果集

Statement 接口的 executeUpdate()方法或 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。只有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。

例如，获取查询结果集，代码如下：

```
ResultSet res = state.executeQuery("select * from tb_emp");
```

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 循环中使用 next()方法迭代结果集。

例如，循环遍历查询结果集，代码如下：

```
while (rest.next()) {                    //rest 为 ResultSet 对象
    String name = rest.getString("name");
    String person = rest.getString("person");
    String date = rest.getString("date");
}
```

ResultSet 对象的 getXXX()方法可获取查询结果集中的数据，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 rest.getString("name")，name 则表示表的列名。如果此时 name 列为查询结果集中的第 2 列，也可通过代码“rest.getString(2)”得到查询结果。图 15.6 表示结果集 rest 的结果。

图 15.6　实例运行效果

15.3.5　关闭连接

在进行数据库访问时，Connection、Statement 和 ResultSet 实例都会占用一定的系统资源。因此在每次访问数据库后，及时地释放这些对象占用的资源是一个很好的编程习惯。Connection、Statement 和 ResultSet 实例都提供了 close()方法，用于释放对象占用的数据库和 JDBC 资源。

例如，关闭 ResultSet 对象、Statement 对象和 Connection 实例，代码如下：

```
resultSet.close();
statement.close();
connection.close();
```

如果通过 DriverManager 类的 getConnection()方法获取 Connection 实例，那么通过关闭 Connection 实例就可同时关闭 Statement 实例与 ResultSet 实例。当采用数据库连接池时，close()方法并没有释放 Connection 实例，而是将其放入连接池中，又可被其他连接调用。此时，如果没有调用 Statement 与 ResultSet 实例的 close()方法，它们在 Connection 中会越来越多，虽然 JVM（Java 虚拟机）会定时地清理缓存，但当数据库连接达到一定数量时，清理不够及时，就会严重影响数据库和计算机的运行速度。

15.4　照猫画虎——基本功训练

15.4.1　基本功训练 1——在数据库中创建临时表格

视频讲解：光盘\mr\lx\15\在数据库中创建临时表格.exe

实例位置：光盘\mr\ym\15.03\src

对于关系型数据库而言，最重要的莫过于表格。本实例将演示如何使用 MySQL 数据库来创建表格。

实现过程如下：

（1）创建 DBConfig 接口，用来保存 JDBC 参数，代码如下：

```
public interface DBConfig {
    String DRIVER = "com.mysql.jdbc.Driver";                    // 声明数据库的驱动
    String URL = "jdbc:mysql://localhost:3306/db_database15";   // 声明数据库的 URL
    String USERNAME = "root";                                   // 声明数据库用户名
    String PASSWORD = "111";                                    // 声明数据库中 root 用户对应的密码
}
```

（2）编写 DBHelper 类，它完成了在数据库中创建表格的功能。表格包括 id、name、gender 等列。代码如下：

```
public class DBHelper implements DBConfig {
    public static void main(String[] args) {
        try {
```

```
            Class.forName(DRIVER);                                          // 加载数据库驱动程序
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
        Connection conn = null;
        try {
            conn = DriverManager.getConnection(URL, USERNAME, PASSWORD); // 获得 Connection 对象
        } catch (SQLException e) {
            e.printStackTrace();
        }
        Statement stat = null;
        try {
            stat = conn.createStatement();                                  // 获得 Statement 对象
        } catch (SQLException e) {
            e.printStackTrace();
        }
        String sql = "create table student( id int primary key auto_increment,  name varchar(20),gender
char(2), age int, "
                + "department varchar(50), email varchar(50));";            // 创建要执行的 SQL 语句
        try {
            stat.executeUpdate(sql);                                        // 执行 SQL 语句
        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            try {
                stat.close();                                               // 关闭 Statement
            } catch (SQLException e) {
                e.printStackTrace();
            }
            try {
                conn.close();                                               // 关闭 Connection
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
}
```

运行本实例前后，查询数据库中的表格，效果如图 15.7 所示。

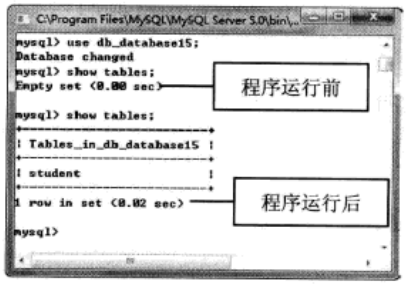


图 15.7　实例运行效果

说明：运行前数据库中并没有表格，运行后新建了表格 student。

技巧：使用 show tables;命令可以查看当前使用的数据库中的全部表格。

你问我答：executeUpdate()方法有什么作用？

在获得 Statement 对象后，可以使用 executeUpdate()方法完成表格的创建、修改和删除等操作。此外，还可以使用该命令完成数据的插入、修改和删除操作。

照猫画虎：将本实例中创建的表格删除。（20 分）（实例位置：光盘\mr\ym\15.03\zmhh）

15.4.2　基本功训练 2——在表格中增加数据

视频讲解：光盘\mr\lx\15\在表格中增加数据.exe

实例位置：光盘\mr\ym\15.04\src

在创建完表格之后，可以使用 JDBC 向表格中增加数据。本实例将演示这个功能。

实现过程如下：

（1）创建 DBConfig 接口，用来保存 JDBC 参数，代码如下：

```
public interface DBConfig {
    String DRIVER = "com.mysql.jdbc.Driver";                          // 声明数据库的驱动
    String URL = "jdbc:mysql://localhost:3306/db_database15";         // 声明数据库的 URL
    String USERNAME = "root";                                         // 声明数据库用户名
    String PASSWORD = "111";                                          // 声明数据库中 root 用户对应的密码
}
```

（2）编写 DBHelper 类，它完成了在 student 中增加数据的功能。表格包括 id、name、gender 等列。代码如下：

```
public class DBHelper implements DBConfig {
    public static void main(String[] args) {
        try {
            Class.forName(DRIVER);                                    // 加载数据库驱动程序
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
        Connection conn = null;
        try {
            conn = DriverManager.getConnection(URL, USERNAME, PASSWORD); // 获得 Connection 对象
        } catch (SQLException e) {
            e.printStackTrace();
        }
        Statement stat = null;
        try {
            stat = conn.createStatement();                            // 获得 Statement 对象
        } catch (SQLException e) {
            e.printStackTrace();
        }
        String sql = "insert into student(name,gender,age,department,email) values ('小明','男',20,'生物
','sl***@163.com');";
```

```
                                                                    // 创建要执行的 SQL 语句
        try {
            stat.executeUpdate(sql);                                // 执行 SQL 语句
        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            try {
                stat.close();                                       // 关闭 Statement
            } catch (SQLException e) {
                e.printStackTrace();
            }
            try {
                conn.close();                                       // 关闭 Connection
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
}
```

运行本实例前后，查询数据库中的表格，效果如图 15.8 所示。

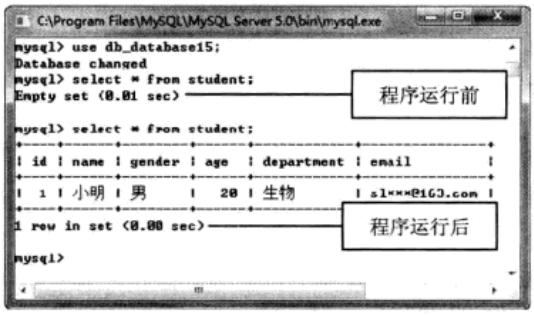


图 15.8　实例运行效果

说明：程序运行前，student 表中没有数据，程序运行后表中增加了一条数据记录。

照猫画虎：使用 Swing 编写图形界面完成向表格中插入数据的功能。（20 分）（实例位置：光盘\mr\ym\15.04\zmhh）

15.4.3　基本功训练 3——在表格中删除数据

视频讲解：光盘\mr\lx\15\在表格中删除数据.exe

实例位置：光盘\mr\ym\15.05\src

对于表格中不需要的数据，可以使用删除命令将其删除。本实例将使用 JDBC 删除 student 表中的全部数据。

实现过程如下：

（1）创建 DBConfig 接口，用来保存 JDBC 参数，代码如下：

```
public interface DBConfig {
    String DRIVER = "com.mysql.jdbc.Driver";                        // 声明数据库的驱动
```

```
    String URL = "jdbc:mysql://localhost:3306/db_database15";          // 声明数据库的 URL
    String USERNAME = "root";                                          // 声明数据库用户名
    String PASSWORD = "111";                                           // 声明数据库中 root 用户对应的密码
}
```

（2）编写 DBHelper 类，它完成了在 student 中删除数据的功能。表格包括 id、name、gender 等列。代码如下：

```
public class DBHelper implements DBConfig {
    public static void main(String[] args) {
        try {
            Class.forName(DRIVER);                                     // 加载数据库驱动程序
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
        Connection conn = null;
        try {
            conn = DriverManager.getConnection(URL, USERNAME, PASSWORD); // 获得 Connection 对象
        } catch (SQLException e) {
            e.printStackTrace();
        }
        Statement stat = null;
        try {
            stat = conn.createStatement();                             // 获得 Statement 对象
        } catch (SQLException e) {
            e.printStackTrace();
        }
        String sql = "delete from student;";                           // 创建要执行的 SQL 语句
        try {
            stat.executeUpdate(sql);                                   // 执行 SQL 语句
        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            try {
                stat.close();                                          // 关闭 Statement
            } catch (SQLException e) {
                e.printStackTrace();
            }
            try {
                conn.close();                                          // 关闭 Connection
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
}
```

运行本实例前后，查询数据库中的表格，效果如图 15.9 所示。

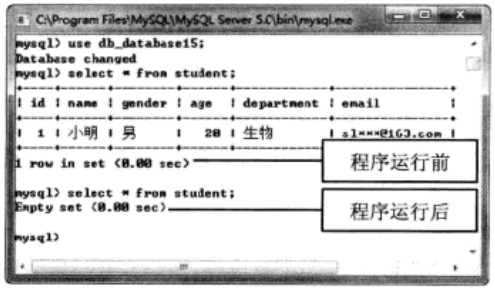


图 15.9　实例运行效果

说明：程序运行前表中有一条记录，程序运行后表中没有记录。

照猫画虎：删除表格中年龄为 20 的全部记录。（20 分）（实例位置：光盘\mr\ym\15.05\zmhh）

15.4.4　基本功训练 4——在表格中修改数据

视频讲解：光盘\mr\lx\15\在表格中修改数据.exe

实例位置：光盘\mr\ym\15.06\src

在录入数据时，难免会发生各种各样的错误，此时可以考虑使用 SQL 语句来修改错误。本实例将修改 student 表中的学院信息为土木。

实现过程如下：

（1）创建 DBConfig 接口，用来保存 JDBC 参数，代码如下：

```
public interface DBConfig {
    String DRIVER = "com.mysql.jdbc.Driver";                    // 声明数据库的驱动
    String URL = "jdbc:mysql://localhost:3306/db_database15";   // 声明数据库的 URL
    String USERNAME = "root";                                   // 声明数据库用户名
    String PASSWORD = "111";                                    // 声明数据库中 root 用户对应的密码
}
```

（2）编写 DBHelper 类，它完成了在 student 中修改数据的功能。表格包括 id、name、gender 等列。代码如下：

```
public class DBHelper implements DBConfig {
    public static void main(String[] args) {
        try {
            Class.forName(DRIVER);                                  // 加载数据库驱动程序
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
        Connection conn = null;
        try {
            conn = DriverManager.getConnection(URL, USERNAME, PASSWORD); // 获得 Connection 对象
        } catch (SQLException e) {
            e.printStackTrace();
        }
        Statement stat = null;
        try {
            stat = conn.createStatement();                          // 获得 Statement 对象
```

```
            } catch (SQLException e) {
                e.printStackTrace();
            }
            String sql = "update student set department = '土木' where department = '生物' ;";
                                                                    // 创建要执行的 SQL 语句
            try {
                stat.executeUpdate(sql);                            // 执行 SQL 语句
            } catch (SQLException e) {
                e.printStackTrace();
            } finally {
                try {
                    stat.close();                                   // 关闭 Statement
                } catch (SQLException e) {
                    e.printStackTrace();
                }
                try {
                    conn.close();                                   // 关闭 Connection
                } catch (SQLException e) {
                    e.printStackTrace();
                }
            }
        }
    }
```

运行本实例前后，查询数据库中的表格，效果如图 15.10 所示。

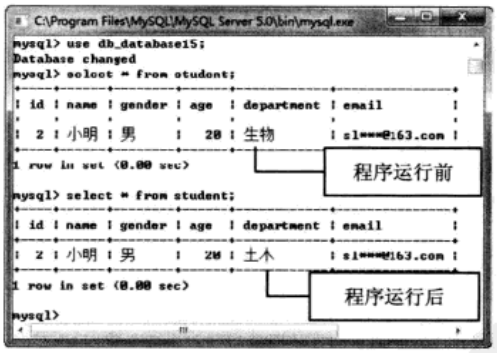


图 15.10　实例运行效果

说明：程序运行前记录中 department 是生物，运行后 department 是土木。

照猫画虎：将表格中年龄为 20 的数据统一改为 22。（20 分）（实例位置：光盘\mr\ym\15.06\zmhh）

15.4.5　基本功训练 5——查询表格中的数据

视频讲解：光盘\mr\lx\15\查询表格中的数据.exe

实例位置：光盘\mr\ym\15.07\src

本实例将使用 JDBC 从 student 表中查询出所有数据并在控制台上显示。

实现过程如下：

（1）创建 DBConfig 接口，用来保存 JDBC 参数，代码如下：

```
public interface DBConfig {
    String DRIVER = "com.mysql.jdbc.Driver";                          // 声明数据库的驱动
    String URL = "jdbc:mysql://localhost:3306/db_database15";         // 声明数据库的 URL
    String USERNAME = "root";                                         // 声明数据库用户名
    String PASSWORD = "111";                                          // 声明数据库中 root 用户对应的密码
}
```

（2）编写 DBHelper 类，它完成了在 student 中查询数据的功能。表格包括 id、name、gender 等列。代码如下：

```
public class DBHelper implements DBConfig {
    public static void main(String[] args) {
        try {
            Class.forName(DRIVER);                                    // 加载数据库驱动程序
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
        Connection conn = null;
        try {
            conn = DriverManager.getConnection(URL, USERNAME, PASSWORD); // 获得 Connection 对象
        } catch (SQLException e) {
            e.printStackTrace();
        }
        Statement stat = null;
        try {
            stat = conn.createStatement();                            // 获得 Statement 对象
        } catch (SQLException e) {
            e.printStackTrace();
        }
        String sql = "select * from student; ;";                      // 创建要执行的 SQL 语句

        ResultSet rs = null;
        try {
            rs = stat.executeQuery(sql);                              // 使用 SQL 语句查询
            StringBuilder sb = new StringBuilder();                   // 创建 StringBuilder 对象
            while (rs.next()) {
                sb.delete(0, sb.capacity());                          // 将对象中的数据清空
                sb.append("序号：" + rs.getInt("id") + " ");           // 在 StringBuilder 中增加数据
                sb.append("姓名：" + rs.getString("name") + " ");      // 在 StringBuilder 中增加数据
                sb.append("性别：" + rs.getString("gender") + "\n");   // 在 StringBuilder 中增加数据
                sb.append("年龄：" + rs.getInt("age") + " ");          // 在 StringBuilder 中增加数据
                sb.append("学院：" + rs.getString("department") + " "); // 在 StringBuilder 中增加数据
                sb.append("Email：" + rs.getString("email"));          // 在 StringBuilder 中增加数据
                System.out.println(sb.toString());                    // 在控制台上输出数据
            }
        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            try {
                rs.close();                                           // 关闭 ResultSet
            } catch (SQLException e) {
```

```
                e.printStackTrace();
            }
            try {
                stat.close();                                    // 关闭 Statement
            } catch (SQLException e) {
                e.printStackTrace();
            }
            try {
                conn.close();                                    // 关闭 Connection
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
}
```

实例运行效果如图 15.11 所示。

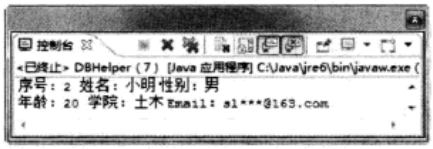


图 15.11　实例运行效果

照猫画虎：在表格控件中显示所有学生的信息。（20 分）（实例位置：光盘\mr\ym\15.07\zmhh）

照猫画虎栏目分数统计：

照猫画虎题目	1	2	3	4	5	总分数	
分数							

15.5　情景应用——拓展与实践

15.5.1　情景应用 1——查询商品销售的前 3 名

视频讲解：光盘\mr\lx\15\查询商品销售的前 3 名.exe

实例位置：光盘\mr\ym\15.08\src

使用数据库来保存数据具有其他方式无法比拟的优越性，如使用相关数据库提供的查询操作，可以很容易地查询某一字段的前几条记录。本实例将使用 MySQL 数据库完成查询商品销售额前 3 名的员工，实例的运行效果如图 15.12 所示。

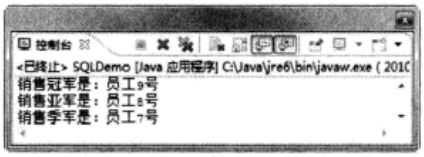


图 15.12　实例运行效果

实现过程如下：

（1）创建 DBConfig 接口，用来保存 JDBC 参数，这些参数包括数据库的驱动、URL、用户名和密码等信息。代码如下：

```
public interface DBConfig {
    String DRIVER = "com.mysql.jdbc.Driver";                    // 声明数据库的驱动
    String URL = "jdbc:mysql://localhost:3306/db_database15";   // 声明数据库的 URL
    String USERNAME = "root";                                   // 声明数据库用户名
    String PASSWORD = "111";                                    // 声明数据库中 root 用户对应的密码
}
```

（2）编写 DBHelper 类，它是一个工具类，实现了 DBConfig 接口，这样可以方便地使用数据库的驱动等信息。在该类中主要提供了用于实现数据增删改的 update()方法和用于实现数据查询的 query()方法。为了便于释放资源，提供了 close()方法。代码如下：

```
public class DBHelper implements DBConfig {
    private static Connection conn;                             // 创建 Connection 类型引用
    private static Statement stat;                              // 创建 Statement 类型引用
    private static ResultSet rs;                                // 创建 ResultSet 类型引用
    public static Connection getConnection() {                  // 获得 Connection 类型对象
        try {
            Class.forName(DRIVER);
            conn = DriverManager.getConnection(URL, USERNAME, PASSWORD);
            return conn;
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return null;
    }
    public static Statement openStatement() {                   // 获得 Statement 类型对象
        try {
            stat = getConnection().createStatement();
            return stat;
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return null;
    }
    public static int update(String sql) {                      // 完成数据的更新操作
        try {
            return openStatement().executeUpdate(sql);
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return -1;
    }
    public static ResultSet query(String sql) {                 // 完成数据的查询操作
        try {
            rs = openStatement().executeQuery(sql);
            return rs;
```

```
            } catch (SQLException e) {
                e.printStackTrace();
            }
            return null;
        }
        public static void close() {                                    // 释放资源
            if (rs != null) {
                try {
                    rs.close();
                } catch (SQLException e) {
                    e.printStackTrace();
                }
            }
            if (stat != null) {
                try {
                    stat.close();
                } catch (SQLException e) {
                    e.printStackTrace();
                }
            }
            if (conn != null) {
                try {
                    conn.close();
                } catch (SQLException e) {
                    e.printStackTrace();
                }
            }
        }
    }
```

（3）编写类 SQLDemo，它使用刚刚编写的工具类 DBHelper 完成查询销售额前 3 名员工信息的操作，并在控制台上输出他们的名字。由于返回的是 ResultSet 类型的对象，因此需要使用 next()方法来移动游标，从而获得数据记录。代码如下：

```
public class SQLDemo {
    public static void main(String[] args) throws SQLException {
        String sql = "select * from achievement order by salecount desc limit 0, 3;"; // 查询前 3 条记录
        ResultSet rs = DBHelper.query(sql);                                  // 获得 ResultSet
        rs.next();                                                           // 移动游标
        System.out.println("销售冠军是：" + rs.getString("name"));
                                                                             // 输出第一条记录中的名字信息
        rs.next();// 移动游标
        System.out.println("销售亚军是：" + rs.getString("name"));
                                                                             // 输出第二条记录中的名字信息
        rs.next();// 移动游标
        System.out.println("销售季军是：" + rs.getString("name"));
                                                                             // 输出第三条记录中的名字信息
    }
}
```

DIY：获得销售最差的 3 名员工的信息。（20 分）（实例位置：光盘\mr\ym\15.08\diy）

15.5.2 情景应用 2——查询所有王姓同学信息

视频讲解：光盘\mr\lx\15\查询所有王姓同学信息.exe

实例位置：光盘\mr\ym\15.09\src

在使用数据库时，还可以利用标准 SQL 语句的很多特性。例如，模糊查询可以根据用户提供的部分信息查询出满足条件的数据。它可以分成两类：一类是查询结果有指定的长度，另一类是查询结果有任意的长度。本实例将使用模糊查询技术查询数据库中所有王姓同学的信息，并在控制台上输出名字和成绩。实例的运行效果如图 15.13 所示。

图 15.13 实例运行效果

实现过程如下：

（1）创建 DBConfig 接口，用来保存 JDBC 参数。代码请读者参考 15.5.1 小节。

（2）编写 DBHelper 类，用于完成数据的增加、查询等功能。代码请读者参考 15.5.1 小节。

（3）编写 SQLDemo，用于查询王姓同学的信息，获得的是 ResultSet 类型的对象。通过 while 循环和 next()方法，可以完成输出所有数据的操作，代码如下：

```
public class SQLDemo {
    public static void main(String[] args) throws SQLException {
        String sql = "select * from studentinfo where name like '王%';";
        ResultSet rs = DBHelper.query(sql);                          // 获得 ResultSet
        while (rs.next()) {                                          // 输出所有王姓同学成绩
            System.out.println(rs.getString("name") + "的成绩是" + rs.getInt("score"));
        }
    }
}
```

DIY：查询数据库中名字长度为 2 的李姓同学的信息。（20 分）（实例位置：光盘\mr\ym\15.09\diy）

15.5.3 情景应用 3——查询男生和女生的人数

视频讲解：光盘\mr\lx\15\查询男生和女生的人数.exe

实例位置：光盘\mr\ym\15.10\src

在使用数据库时，还可以利用标准 SQL 语句的很多特性，如分组查询。该技术可以将数据进行分组，然后确定每组中记录的条数，如男生和女生各有多少人、成绩在 90 分以上的有多少人等。这些在实际生活中是非常常用的。本实例将使用其查询男生和女生的人数，实例的运行效果如图 15.14 所示。

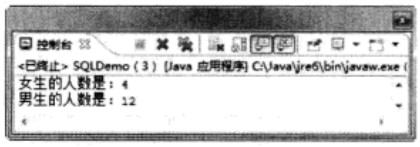


图 15.14 实例运行效果

实现过程如下：

（1）创建 DBConfig 接口，用来保存 JDBC 参数。代码请读者参考 15.5.1 小节。

（2）编写 DBHelper 类，用来完成数据的增加、查询等功能。代码请读者参考 15.5.1 小节。

（3）编写 SQLDemo，用于查询男生和女生的人数，获得的是 ResultSet 类型的对象。通过 while 循环和 next()方法，可以完成输出所有数据的操作，代码如下：

```
public class SQLDemo {
    public static void main(String[] args) throws SQLException {
        String sql = "select gender, count(gender) from studentinfo group by gender;";
        ResultSet rs = DBHelper.query(sql);
        while (rs.next()) {
            System.out.println(rs.getString(1) + "生的人数是：" + rs.getInt(2));
        }
    }
}
```

DIY：统计各个分数的学生人数。（20 分）（实例位置：光盘\mr\ym\15.10\diy）

15.5.4　情景应用 4——用户重名校验

视频讲解：光盘\mr\lx\15\用户重名校验.exe

实例位置：光盘\mr\ym\15.11\src

在注册用户时，通常需要有一个唯一标识该用户的信息，即数据库中的一个键值。通常可以选择用户名，但是如果两个用户申请注册同一个用户名，则需要提示后一个用户该名字已经注册了。本实例将实现用户重名校验功能，实例的运行效果如图 15.15 所示。

图 15.15　实例运行效果

实现过程如下：

（1）创建 DBConfig 接口，用来保存 JDBC 参数。代码请读者参考 15.5.1 小节。

（2）编写 DBHelper 类，用来完成数据的增加、查询等功能。代码请读者参考 15.5.1 小节。

（3）继承 JFrame 类编写 RegistDemo 类，并在其上增加标签、按钮、密码框等控件。

（4）完成对“验证”按钮的监听事件，它是通过 do_button1_actionPerformed()方法实现的。在该方法中，首先获得用户输入的用户名，然后使用其作为条件在数据库中进行查询，如果数据库中存在这个用户名，则提示用户“用户名冲突”，否则可以注册。关键代码如下：

```
protected void do_button1_actionPerformed(ActionEvent e) {
    String name = textField.getText();                          // 获得用户输入的用户名
    if (name.isEmpty()) {                                       // 如果用户名为空则进行提示
        JOptionPane.showMessageDialog(this, "请输入用户名！", null, JOptionPane.INFORMATION_ MESSAGE);
        return;
    }
    String sql = "select * from account where name = '" + name + "';";
    ResultSet rs = DBHelper.query(sql);                         // 查询表格中包含该用户名的记录
    try {
        if (!rs.next()) {
            JOptionPane.showMessageDialog(this, "还可以注册！", null, JOptionPane.INFORMATION_ MESSAGE);
            return;
        } else {
```

```
                JOptionPane.showMessageDialog(this, "用户名冲突！", null, JOptionPane.INFORMATION_ MESSAGE);
                return;
            }
        } catch (SQLException e1) {
            e1.printStackTrace();
        }
    }
```

DIY：在本实例的基础上完成增加用户的功能。（20 分）（实例位置：光盘\mr\ym\15.11\diy）

15.5.5 情景应用 5——用户登录系统

视频讲解：光盘\mr\lx\15\用户登录系统.exe

实例位置：光盘\mr\ym\15.12\src

出于安全性的考虑，软件往往需要使用用户名和密码进行登录。最常见的就是各种不同的数据库和 Linux 系统。一般而言，不同的用户在软件中能够完成的操作是不同的，即具有权限的差别。本实例将完成一个简单的登录系统，用户要输出用户名和密码，然后在数据库中进行查询，如果匹配则提示登录成功。实例的运行效果如图 15.16 所示。

图 15.16 实例运行效果

实现过程如下：

（1）创建 DBConfig 接口，用来保存 JDBC 参数。代码请读者参考 15.5.1 小节。

（2）编写 DBHelper 类，用来完成数据的增加、查询等功能。代码请读者参考 15.5.1 小节。

（3）继承 JFrame 类编写 LoginDemo 类，并在其上增加标签、按钮、密码框等控件。

（4）完成对“登录”按钮的监听事件，它是通过 do_button1_actionPerformed()方法完成的。首先对用户输入的信息进行简单的校验，如果输入为空，则进行提示。然后根据用户名在数据库中查询密码，如果两者都成功匹配，则提示用户登录成功。关键代码如下：

```
protected void do_button1_actionPerformed(ActionEvent e) {
    String username = textField.getText();                          // 获得用户名
    char[] password = passwordField.getPassword();                  // 获得密码
    if ((username.isEmpty()) || (password.length == 0)) {           // 验证用户名或密码是否为空
        JOptionPane.showMessageDialog(this, "用户名或密码为空！", null, JOptionPane.WARNING_ MESSAGE);
        return;
    }
    String sql = "select password from account where name = '" + username + "';";
    ResultSet rs = DBHelper.query(sql);
    try {
        if (!rs.next()) {
            JOptionPane.showMessageDialog(this, "用户名错误！", null, JOptionPane.WARNING_ MESSAGE);
            return;
        }
    } catch (SQLException e1) {
        e1.printStackTrace();
    }
    try {
```

```
            String password1 = rs.getString(1);                          // 获得指定用户名的密码
            if (password1.equals(new String(password))) {                // 验证密码是否正确
                JOptionPane.showMessageDialog(this, "登录成功！", null, JOptionPane.INFORMATION_ MESSAGE);
                return;
            } else {
                JOptionPane.showMessageDialog(this, "密码错误！", null, JOptionPane.WARNING_MESSAGE);
                return;
            }
        } catch (SQLException e1) {
            e1.printStackTrace();
        }
    }
```

DIY：在本实例的基础上增加注册用户的功能。注册的信息包括用户名和密码。用户需要输入两次密码以防止用户无操作。（20 分）（实例位置：光盘\mr\ym\15.12\diy）

情景应用 DIY 栏目分数统计：

DIY 题目	1	2	3	4	5	总分数	
分数							

15.6　自 我 测 试

一、选择题（每题 10 分，5 道题）

1．常见的数据库包括以下哪些类型？（　　）

A．层次型　　B．网状型　　C．关系型　　D．面向对象型

2．使用下列哪些（个）语句可以加载 MySQL 数据库驱动？（　　）

A．Class.forName(“com.mysql.jdbc.Driver”);

B．Class.loadDriver(“com.mysql.jdbc.Driver”);

C．Class.forName(“Com.mysql.jdbc.Driver”);

D．Class.loadDriver(“Com.mysql.jdbc.Driver”);

3．下面代码中，会抛出哪些（个）类型的异常？（　　）

```
public class Test {
    private static final String DRIVER = "com.mysql.jdbc.Driver";
    private static final String URL = "jdbc:mysql://localhost:3306/db_database15";
    private static final String USERNAME = "root";
    private static final String PASSWORD = "111";
    public static void main(String[] args) {
        Class.forName(DRIVER);                                           // 加载数据库驱动程序
        Connection conn = DriverManager.getConnection(URL, USERNAME, PASSWORD);
                                                                         // 获得 Connection 对象
    }
}
```

A．ClassNotFoundException　　B．SQLException

C．JDBCException　　D．MySQLException

4．使用 JDBC 查询数据库时，如果结果集中没有数据，则会发生的情况是（　　）。

A．获得的 ResultSet 对象为 null

B．获得的 ResultSet 对象如果调用 next()方法会抛出异常

C．获得的 ResultSet 对象如果调用 next()方法会返回 false

D．获得的 ResultSet 对象如果调用 getRow()方法会返回-1

5．下列哪些（个）方法是在 ResultSet 中定义的？（　　）

A．getByte()　　B．getBytes()　　C．getChar()　　D．getChars()

二、填空题（每题 10 分，5 道题）

1．如果 MySQL 数据库的 URL 是 jdbc:mysql://localhost:3306/db_database15，则该数据库的默认端口号是_____。

2．如果数据库的 URL 是 jdbc:mysql://localhost:3306/db_database15，则当前使用的是哪个数据库？____

3．JDBC 相关类和接口都位于哪个包中？_____

4．如果要关闭连接，则可以使用 Collection 接口中的哪个方法？_____

5．java.sql.ResultSet 是类还是接口？_____

测试分数统计：

类别	第 1 题	第 2 题	第 3 题	第 4 题	第 5 题
选择题分数					
填空题分数					
总分数					

15.7 行 动 指 南

开始日期：_____年____月____日　　结束日期：_____年____月____日

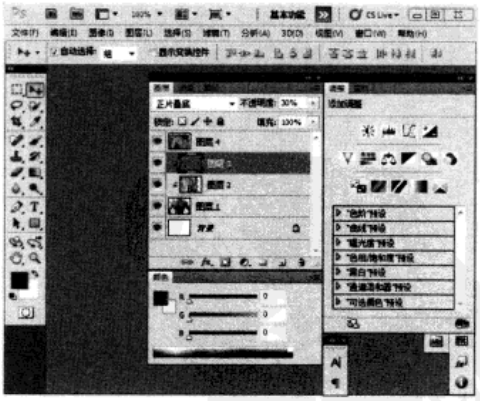
序号	内　容	行 动 指 南	
1	照猫画虎栏目 分数（　　）	分数>75 分	优秀，基本功掌握得不错，加油！
		75 分>分数>50 分	及格，知识掌握得不牢，重新做一遍照猫画虎。
		分数<50 分	请使用光盘中提供的“实战能力测试系统”进行提高训练。
	情景应用栏目 分数（　　）	分数>75 分	优秀，综合应用能力很强。
		75 分>分数>50 分	及格，综合应用能力需提高，再练一遍情景应用。
		分数<50 分	请使用光盘中提供的“实战能力测试系统”进行提高训练。
	自我测试栏目 分数（　　）	分数>75 分	优秀，有成为编程高手的潜质。
		分数<75 分	请使用光盘中提供的“实战能力测试系统”进行提高训练。
	综合评价	反复训练后，以上各项分数都在 75 分以上，方可进入下一堂课学习。	
2	编程实践能力培养：本栏目提供了一些实践题目，对于培养编程能力很有效，要做好。	（1）使用数据库制作电子词典。	
		（2）使用数据库完成图书销售系统。	
		（3）使用数据库记录系统的登录信息。	
		（4）使用数据库完成学生管理系统。	

续表

序号	内　容	行 动 指 南
3	编程习惯培养：本堂课培养读者在学习开发中记笔记的习惯，把开发中遇到的问题、总结的经验记录下来。	
4	创新能力培养：看看身边有没有可以用编程解决的问题，记录到右边的表格中。根据学习进度，尝试编写解决这些问题的小程序。	

15.8　成功可以复制——图文世界的缔造者

1940 年，约翰·沃洛克（John Warnock）出生于美国犹他州的盐湖城。小时候，他并没有显示出任何过人的才华，直到读中学九年级时，代数考试仍不及格。有一次，学校组织智商测试，测试主持人询问他今后打算干什么工作？他想了想回答道："或许，我想成为一名工程师。"主持者毫不客气地打断他的话："测试结果表明，在工程学领域，你成功的概率几乎是零。"幸运的是，他的数学老师帮助他在数学中找到了乐趣。之后他的数学成绩稳步上升，终于如愿地考上犹他大学，并先后取得了数学学士学位、数学硕士学位、机电工程博士学位。



1965 年约翰·沃洛克在犹他大学获得数学硕士学位，并开始在犹他大学计算中心工作。一次，一个正在研究图形程序的学生在开发中遇到了困难，他随意询问了正在工作的约翰·沃洛克，希望能找到点灵感。没有想到，约翰·沃洛克提出了一种前所未有的解决方法，该方法简单明了，但大学里所有的软件高手都没想到。这个小插曲改变了约翰·沃洛克的生活，他变成了学校里的软件高手，这也促使他把研究课题转到了图形处理领域。

约翰·沃洛克和同事开发了一种定义三维数据库规则的程序——JAM 语言，在 JAM 的基础上，他又与同事开发出一种打印机标准 Interpress。约翰·沃洛克发现，Interpress 是一件　很有发展潜力的产品，如果进行推广，将改变整个打印和出版业。他们想将 Interpress 发展成产品，但在其工作的施乐公司是不可能的，于是他们决定自己创业。

1982 年，约翰·沃洛克和 Chuck Geschke 共同创办了 Adobe 公司。如今，他已成为这个市值几百亿美元的软件企业的领袖，作为桌面出版、电子文档和图形处理软件市场先驱和领导者的 Adobe 公司，不但拥有 Photoshop、PageMaker、Acrobat Reader 软件，PDF 格式和 PostScript 语言也已是世界范围的标准。2005

年，美国《财富》杂志把 Adobe 公司评为硅谷最成功的两家公司之一。

经典语录

“我想对孩子们说，除了我之外，爱因斯坦 9 年级数学考试也不及格，牛顿也是几何考试不及格。所以，考试不能说明发明者或成功者今后的前途，学生们完全可以怀疑别人告诉他们的事情。”

深度评价

“和对市场的影响程度相比，开发伟大产品带来的乐趣和满足比任何其他东西给我的动力都大。”

第16堂课

输入输出流与文件

（视频讲解：118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 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。本堂课将向读者介绍 Java 的 I/O 处理。

学习摘要：

- File 类操作磁盘文件
- 字节输入输出流的使用
- 字符输入输出流的使用
- 字符串生成器的用法

16.1 文　　件

在变量、数组、对象中存储数据是暂时的，程序结束后它们就会丢失。为了更永久地保存程序创建的数据，可以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这些数据。Java 的 I/O 技术可以将数据保存到文本文件中，以达到永久保存的目的。

16.1.1 File 类

java.io 包中封装了用于输入输出的类。File 类在 io 包中唯一代表磁盘文件本身，该类定义了一些与平台无关的方法来操作文件。通过 File 实例可以获取文件本身的一些信息，如文件名、文件长度、文件读写权限等。通过 File 类的一些方法，还可实现文件的创建、删除、重命名等操作。

可以通过以下 3 种方法创建一个 File 对象。

☑ 指定文件的路径名。该方法声明如下：

```
new File(String pathName)
```

参数说明：pathName 是包含有文件名称的路径名。

☑ 通过给定的父路径和子路径字符串创建 File 对象。该方法声明如下：

```
new File(String parent,String child)
```

参数说明：parent 是父路径字符串，如“D://”、“D://txt”。child 是子路径字符串，如“word.txt”。

注意：通过第二种方法创建的 File 对象，指定的父路径必须存在，否则将会报出 IOException 异常。

☑ 使用一个 File 对象与字符串创建 File 对象。该方法声明如下：

```
new File(File parent,String child)
```

参数说明：parent 是父抽象路径名，child 是子路径名字符串。

16.1.2 常用方法

File 类提供了很多操作文件的方法，常用的方法如表 16.1 所示。

表 16.1 File 类的常用方法

方　法	返 回 值	说　明
getName()	String	获取文件的名称
canRead()	boolean	判断文件是否是可读的
canWrite()	boolean	判断文件是否可被写入
exits()	boolean	判断文件是否存在
length()	long	获取文件的长度（以字节为单位）
getAbsolutePath()	String	获取文件的绝对路径
getParent()	String	获取文件的父路径
isFile()	boolean	判断文件是否存在
isDirectory()	boolean	判断文件是否是一个目录
isHidden()	boolean	判断文件是否是隐藏文件
lastModified()	Long	获取文件最后修改时间

续表

返 回 值	方　法	说　明
listRoots()	File[]	列出可用的文件系统根
mkdir()	boolean	创建此抽象路径名指定的目录
renameTo()	boolean	重新命名此抽象路径名表示的文件

例 16.01　获取 E 盘 text 文件夹下 word.txt 文件的文件名称、父路径、长度等信息。实例代码如下：**（实例位置：光盘\mr\ym\16.01）**

```
public class FileDemo {
    public static void main(String[] args) {
        File file = new File("E://text//word.txt");                    // 创建文件对象
        if (file.exists()) {                                           // 判断该文件是否存在
            String name = file.getName();                              // 获取文件名称
            String parent = file.getParent();                          // 获取文件父路径
            long leng = file.length();                                 // 获取文件长度
            boolean bool = file.canWrite();                            // 判断该文件是否可改写
            System.out.println("文件名称为：" + name);                  // 输出信息
            System.out.println("文件目录为：" + parent);
            System.out.println("文件大小为：" + leng + " bytes");
            System.out.println("是否为可改写文件：" + bool);
        }
    }
}
```

实例运行效果如图 16.1 所示。

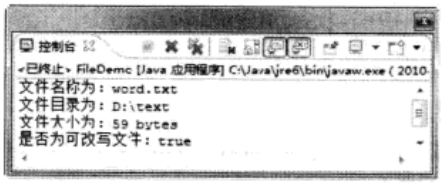


图 16.1　实例运行效果

通过 File 类，不仅可以获取指定文件的各属性信息，还可以实现文件的重命名、获取指定目录下的所有文件等。

例 16.02　获取 E 盘 text 文件夹下所有文件的完整路径名。实例代码如下：**（实例位置：光盘\mr\ym\16.02）**

```
public class FileDemo {
    public static void main(String[] args) {
        File dir = new File("E://text");                               // 根据指定的文件路径创建文件对象
        if (dir.isDirectory()) {                                       // 如果该文件对象指定的是一个目录
            File[] files = dir.listFiles();                            // 获取该目录下的抽象路径名数组
            for (int i = 0; i < files.length; i++) {                   // 循环遍历该数组
                File file = files[i];                                  // 获取数组中的元素
                System.out.println("第" + (i + 1) + "个文件的名称是：" + file.getAbsolutePath());
            }
        }
    }
}
```

实例运行效果如图 16.2 所示。

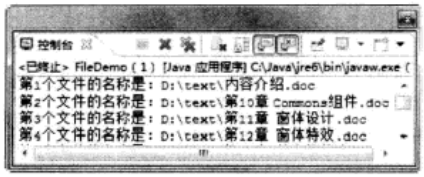


图 16.2 实例运行效果

16.2 字节输入输出流

流是一组有序的数据序列，根据操作的类型，可将其分为输入流和输出流两种。I/O 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。

程序通过输入流可以读取源中的数据，如图 16.3 所示。

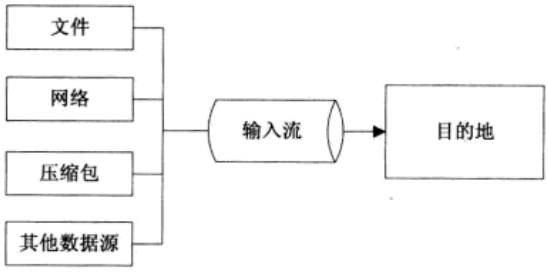


图 16.3 输入模式

程序通过向输出流中写数据把信息传递到目的地。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。输出模式如图 16.4 所示。

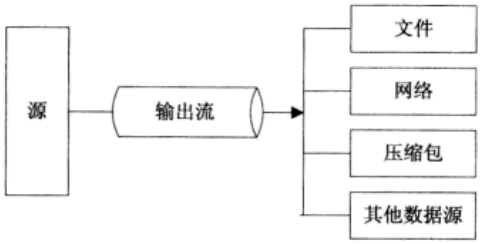


图 16.4 输出模式

字节流用于处理二进制数据的读取和写入，以字节为单位，InputStream 类和 OutputStream 类是字节流的抽象类，它们定义了数据流读取和写入的基本方法。

16.2.1 标准输入流

InputStream 类是字节输入流的抽象类，是所有字节输入流的父类。该类定义了操作输入流的各种方法，

如表 16.2 所示。

表 16.2　InputStream 类的常用方法

方　法	返 回 值	说　明
available()	int	返回当前输入流的数据读取方法可以读取的有效字节数量
close()	void	关闭当前输入流，并释放任何与之关联的系统资源
mark(int readlimit)	void	在当前输入流中做标记位置。当调用 reset()方法时将返回到该位置，从标记位置开始，到再读入 readlimit 个字符为止
markSupported()	boolean	测试当前输入流是否支持 mark()和 reset()方法。只要其中一个不支持，则返回 false
read()	int	从当前输入流中读取一个字节。若已到达流结尾，则返回-1
read(byte[] bytes,int off,int len)	int	从当前输入流读取一定的 byte 数据，并存放在数组中。然后返回读取的 byte 数据的数量。若到达流结尾，则返回-1
reset()	void	将当前输入流重新定位到最后一次调用 mark()方法时的位置

下面通过实例介绍如何使用 InputStream 从控制台获取用户输入的数据信息。

例 16.03　从控制台获得用户输入并在控制台上输出。实例代码如下：**（实例位置：光盘\mr\ym\16.03）**

```
public class Echo {
    public static void main(String[] args) {
        InputStream is = System.in;                                    // 定义 InputStream 对象
        byte[] bytes = new byte[1024];
        try {
            System.out.println("请输入字符串：");                        // 提示用户输入字符串
            is.read(bytes);                                            // 将输入的字符串保存在数组中
            String input = new String(bytes);                          // 将数组中的内容转换成字符串
            System.out.println("您输入的字符串：" + input.trim());        // 输出字符串
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            try {
                is.close();                                            // 释放资源
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

实例运行效果如图 16.5 所示。

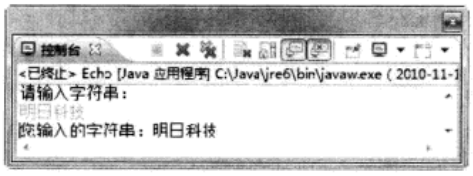


图 16.5　实例运行效果

16.2.2 标准输出流

OutputStream 类是字节输出流的抽象类，该类定义了所有输出流的操作方法，如表 16.3 所示。

表 16.3 OutputStream 类的常用方法

方 法	返 回 值	说 明
close()	void	将输出流关闭，并释放与当前输出流有关的系统资源
flush()	void	刷新输出流，并强制写入所有缓冲的字节数据
write(int ib)	void	将指定的字节写入此输出流
write(byte[] by)	void	将 by.length 个字节从指定的 byte 数组写入此输出流
write(byte[] by,int off,int len)	void	将指定 by 数组中从偏移量 off 开始的 len 个字节写入此输出流

例 16.04 创建 OutputStream 实例 out，并将其赋值为 System.out，标准输出流。通过 write()方法向流中写数据。实例代码如下：**（实例位置：光盘\mr\ym\16.04）**

```
public class OutputStreamDemo {
    public static void main(String[] args) {
        OutputStream out = System.out;                                    // 实例化 OutputStream 类
        try {                                                             // 创建 byte 数组
            byte[] bs = "本实例使用 OutputStream 输出流，在控制台输出字符串\n".getBytes();
            out.write(bs);                                                // 向流中写数据
            bs = "输出内容：\n".getBytes();
            out.write(bs);
            bs = "          《Java 从基础到项目实战》学好 Java 的必备书".getBytes();
            out.write(bs);
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            try {
                out.close();                                              // 关闭流
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

实例运行效果如图 16.6 所示。

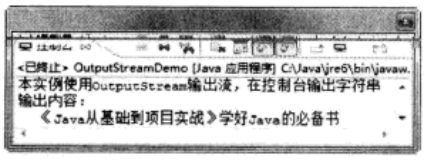


图 16.6 实例运行效果

16.2.3 文件输入流

在程序运行期间，大部分数据都是在内存中进行操作，当程序关闭时，这些数据将完全消失。对于需

要永久保存的数据，如日志信息、程序的配置信息等，可以使用文件输入输出流，与指定的文件建立关联。然后将需要的数据输出到文件中，以达到永久保存的目的。

FileInputStream 文件字节输入流，可以从指定路径的文件中读取字节数据。该类继承自 InputStream 类，并实现了读取输入流的各种方法。可以通过以下两种方式创建 FileInputStream 实例。

☑ 使用 File 作为方法参数。该方法声明如下：

```
new FileInputStream(File file)
```

参数说明：file 是 File 类型对象。

☑ 使用文件路径作为方法参数。该方法声明如下：

```
new FileInputStream(String path)
```

参数说明：path 是文件的绝对路径或相对路径。

下面的实例通过 FileInputStream 类的 read()方法，读取指定磁盘中的文件内容。

例 16.05 本实例实现读取 E 盘 text 文件夹下的 word.txt 文件，并将读取结果输出到控制台上。实例代码如下：**（实例位置：光盘\mr\ym\16.05）**

```
public class FileInputStreamDemo {
    public static void main(String[] args) {
        File file = new File("E:\\text\\word.txt");                    // 创建文件对象
        try {
            FileInputStream fis = new FileInputStream(file);           // 创建 FileInputStream 类对象
            int lenght;
            byte by[] = new byte[1024];                                // 创建 byte 对象
            while ((lenght = fis.read(by)) != -1) {                    // 循环读取文件中的数据
                String str = new String(by, 0, lenght);               // 根据读取信息创建字符串对象
                System.out.println("从文件中读取出的数据是：  " + str);   // 输出信息
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

实例运行效果如图 16.7 所示。

图 16.7　实例运行效果

16.2.4　文件输出流

与 FileInputStream 类对应，FileOutputStream 类实现向文件中写数据。该类继承自 OutputStream 类，并实现了输出数据的各种方法。创建文件字节输出流常用的构造方法如下：

☑ 使用 File 类型的文件对象，创建与文件关联的 FileOutputStream 类对象。该方法声明如下：

```
new FileOutputStream(File file)
```

参数说明：file 是 File 文件类型对象。

☑ 根据指定的文件路径和文件名称创建 FileOutputStream 实例。该方法声明如下：

```
new FileOutputStream(String path)
```

参数说明：path 指定文件的路径和文件名称。

下面通过实例介绍 FileOutputStream 类的应用。

例 16.06 使用 FileOutputStream 类实现向指定文件中写文件的功能。实例代码如下：**（实例位置：光盘\mr\ym\16.06）**

```
public class FileOutputStreamDemo {
    public static void main(String[] args) {
        try {
            File file = new File("E://text//writeFile.txt");                    // 创建文件对象
            if (!file.exists()) {                                               // 如果该文件不存在
                file.createNewFile();                                           // 新建文件
            }
            FileOutputStream fos = new FileOutputStream(file);                  // 创建 FileOutputStream 实例
            byte[] bytes = "Java 编程词典软件，学 Java 必备。".getBytes();       // 创建字节数组
            fos.write(bytes);                                                   // 向文件中写数据
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

打开 E 盘 text 文件夹中的 writeFile.txt 文件，效果如图 16.8 所示。

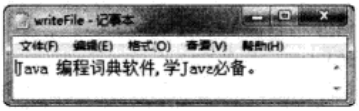


图 16.8 实例运行效果

16.2.5 缓存输入流

缓存流可以说是 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。缓存区的大小，取决于其所在的操作系统、可用的内存空间以及机器配置。BufferedInputStream 对象位于 InputStream 对象之前。图 16.9 描述了 BufferedInputStream 类读取文件的过程。

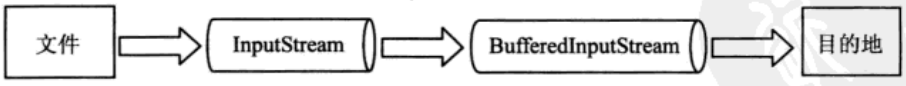


图 16.9 BufferedInputStream 类读取文件的过程

BufferedInputStream 类常用的构造方法如下：

☑ 以 InputStream 类为对象创建 BufferedInputStream 实例。该方法声明如下：

```
new BufferedInputStream(InputStream in)
```

参数说明：in 是输入流对象。

☑ 创建指定缓冲区大小的 BufferedInputStream。

```
new bufferedInputStream(InputStream ins,int size)
```

参数说明：ins 是底层输入流，size 用来指定缓冲区的大小。

有了缓存输入流，可以在流上执行 skip()、mark()和 reset()方法。下面的实例将通过 BufferedInputStream 类实现读取指定文件的前 5 个字节。

例 16.07　使用 BufferedInputStream 类实现循环读取指定磁盘文件的前 5 个字节。实例代码如下：**（实例位置：光盘\mr\ym\16.07）**

```
public class BufferedInputStreamDemo {
    public static void main(String[] args) {
        try {
            File file = new File("E://text//writeFile.txt");                    // 创建文件对象
            FileInputStream fin = new FileInputStream(file);                    // 创建 FileInputStream 对象
            BufferedInputStream bip = new BufferedInputStream(fin);             // 创建 BufferedInputStream 对象
            int count = 0;
            bip.mark(50);                                                       // 在输入流中定义标记位置
            for (int i = 0; i < 10; i++) {                                      // 在循环中读取文件内容
                count++;
                int read = bip.read();                                          // 读取文件内容
                if (count % 5 == 0)
                    bip.reset();
                                                                    // 将流定位到最后一次调用 mark()方法时的位置
                System.out.print((char) read);                                  // 将读取的内容输出
            }
            bip.close();                                                        // 关闭流
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

E 盘 text 文件夹下的 writeFile.txt 文件中数据如图 16.10 所示。

实例运行效果如图 16.11 所示。

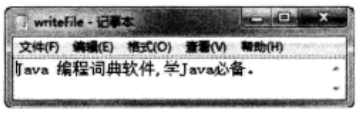


图 16.10　writeFile.txt 文件中数据

图 16.11　实例运行效果

16.2.6　缓存输出流

通过设置 BufferedOutputStream 缓存字节输出流，应用程序可以将各个字节写入底层输出流中。使用 BufferedOutputStream 输出信息和使用 OutputStream 输出信息完全相同，只不过 BufferedOutputStream 类有一个 flush()方法可以将缓存区中的数据强制输出完。该类的构造方法如下：

☑　创建一个新的缓存输出流，以将数据写入指定的底层输出流。该方法声明如下：

```
new BufferedOutputStream(OutputStream in)
```

参数说明：in 是输入流对象。

☑　创建一个新的缓存输出流，以将具有指定缓存区大小的数据，写入指定的底层输出流。该方法声明如下：

```
new BufferedOutputStream(OutputStream out,int size)
```

参数说明：out 是底层输出流，size 是缓冲区大小。

例 16.08 使用 BufferedOutputStream 类向 F 盘的 Getif.txt 文件中写数据。实例代码如下：**（实例位置：光盘\mr\ym\16.08）**

```
public class BufferedOutputStreamDemo {
    public static void main(String[] args) {
        try {
            File file = new File("F://Getif.txt");                          // 创建文件对象
            if (!file.exists()) {                                           // 如果该对象不存在
                file.createNewFile();                                       // 新建文件
            }
            FileOutputStream fos = new FileOutputStream(file);              // 创建 FileOutputStream 对象
            BufferedOutputStream bus = new BufferedOutputStream(fos);       // 创建 BufferedOutputStream 对象
            byte[] by = "香蕉   ".getBytes();                               // 创建字节数组
            bus.write(by);                                                  // 向流中写数据
            byte[] by2 = "苹果   ".getBytes();
            bus.write(by2);
            byte[] by3 = "橘子   ".getBytes();
            bus.write(by3);
            bus.flush();                                                    // 刷新缓存输入流
            bus.close();                                                    // 关闭流
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

运行程序，将定义的 byte 数组写入到 F 盘的 Getif.txt 文件中。写入数据后的 Getif.txt 文件如图 16.12 所示。

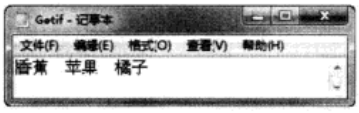


图 16.12 实例运行效果

16.2.7 数据输入流

数据输入流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，当读取数值时，不必再关心这个数值应当是多少个字节。该类的构造方法如下：

```
new DataInputStream(InputStream in)
```

参数说明：in 是输入流对象。

DataInputStream 类的常用方法如表 16.4 所示。

表 16.4 DataInputStream 类的常用方法

方 法	返 回 值	说 明
readBoolean()	boolean	读取一个输入布尔值。该方法适用于读取用接口 DataOutput 的 writeBoolean()方法写入的字节
readByte()	byte	读取一个字节。该方法适用于读取用接口 DataOutput 的 writeByte()方法写入的字节

续表

方　法	返 回 值	说　明
readChar()	char	读取一个字符。该方法适用于读取用接口 DataOutput 的 writeChar()方法写入的字节
readInt()	int	从文件中读取一个 int 值。该方法适用于读取用接口 DataOutput 的 writeInt()方法写入的字节
readFloat()	float	从文件中读取一个单精度浮点值。该方法适用于读取用接口 DataOutput 的 writeFloat()方法写入的字节
readUTF()	String	读取一个 UTF 字符串，可以使用 DataOutput 接口的 writeUTF()方法写入适合此方法读取的数据

16.2.8　数据输出流

数据输出流（DataOutputStream）允许应用程序以适当方式将基本 Java 数据类型写入输出流中，然后应用程序可以使用数据输入流将数据读入。DataOutputStream 类的构造方法如下：

```
new DataOutputStream(OutputStream out);
```

参数说明：out 是输出流对象。

DataOutputStream 类的常用方法如表 16.5 所示。

表 16.5　DataOutputStream 类的常用方法

方　法	返 回 值	说　明
writeBoolean(boolean bool)	void	把一个布尔值作为单字节值写入
writeBytes (String str)	void	将字符串按字节顺序写入到基础输出流中
writeChars(String str)	void	将字符串按字符顺序写入到基础输出流中
writeUTF(String str)	void	写入一个 UTF 字符串
writeInt(int v)	void	写入一个 int 型数据
writeFloat(float f)	void	写入一个 float 型数据

DataOutputStream 类提供了 3 种写入字符串的方法，分别为 writeBytes()、writeChars()和 writeUTF()。由于 Java 中的字符是 unicode 编码，是双字节的，writeBytes 只是将字符串中每一个字符的低字符内容写入目标设备中；而 writeChars 将字符串中每一个字符的两个字节的内容都写入到目标设备中。writeUTF 方法使用修改过的 UTF-8 编码方式向底层输出流中写入字符串。

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中读取一个字符串，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，那么就无法知道读取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，所以只有通过 readUTF()方法才能准确地读回写入字符串。

例 16.09　通过 DataOutputStream 类的 writeUTF()、writeChars()和 writeBytes()方法向指定的磁盘文件中写数据，并通过 DataInputStream 类的 readUTF()方法将写入的数据输出到控制台上。实例代码如下：**（实例位置：光盘\mr\ym\16.09）**

```
public class DataStreamDemo {
    public static void main(String[] args) {
        try {
            File file = new File("E:\\word.txt");                    // 创建文件对象
```

```
            if (!file.exists()) {                                    // 如果该文件不存在
                file.createNewFile();                                // 新建文件
            }
            FileOutputStream fs = new FileOutputStream(file);        // 创建 FileOutputStream 对象
            DataOutputStream ds = new DataOutputStream(fs);          // 创建 DataOutputStream 对象
            ds.writeUTF("使用 writeUFT()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.");
            fs.close();                                              // 将流关闭
            FileInputStream fis = new FileInputStream("E:\\word.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
            fis.close();
        } catch (Exception e) {
            e.printStackTrace();                                     // 输出异常信息
        }
    }
}
```

运行程序，向 word.txt 文件中写入的数据如图 16.13 所示。在控制台上的输出如图 16.14 所示。

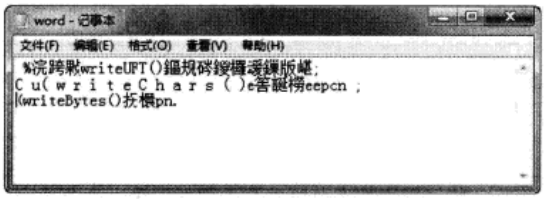


图 16.13　向文件中写入的数据

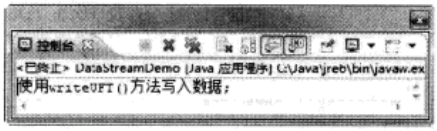


图 16.14　实例运行效果

16.3　字符输入输出流

使用字节流 InputStream 与 OutputStream 向文件中写入数据与读取数据，有一点不足，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。使用字符输入输出流可以避免这种现象。

字节流与字符流的区别是，字节流以字节为单位传送数据，可以是任何类型的数据，如文本、音频、视频、图片等；而字符流以字符为单位传送数据，只能传送文本类型的数据。

16.3.1　Reader 类

字符流用于处理字符数据的读取和写入，它以字符为单位。Reader 类与 Writer 类是字符的抽象类，它们定义了字符流读取和写入的基本方法，各个子类会依其特点实现或覆盖这些方法。

说明：计算机中每个汉字的显示占用两个字节，而在 Java 语言中，字符类型的数据也是双字节的，所以 Java 中的字符类型可以容纳汉字信息。

Reader 类定义了操作字符输入流的各种方法，该类的常用方法如表 16.6 所示。

表 16.6　Reader 类的常用方法

方　法	返 回 值	说　明
close()	void	关闭并释放该流占用的所有资源
mark(int readLimit)	void	标记流中的当前位置
read()	int	从流中读取单个字符
read(char[] cbuf)	int	读取一些字符到字符数组内，返回所读入的字符的数量。若已达到流结果，则返回-1
read(char[] cbuf,int off,int len)	int	读取一些字符到 cbuf 数组，下标从 off 开始，长度为 len，返回所读入的字符的数量。若已到达流结果，则返回-1
reset()	void	将当前输入流重新定位到最后一次调用 mark()方法时的位置

例 16.10　创建 Reader 的子类 InputStreamReader 实例，并将其初始化为控制台的字符输入流。然后通过 read()方法将读取的数据输出。实例代码如下：**（实例位置：光盘\mr\ym\16.10）**

```
public class InputStreamReaderDemo {
    public static void main(String[] args) {
        // 创建 Reader 子类 InputStreamReader 实例
        InputStreamReader rin = new InputStreamReader(System.in);
        try {
            char[] cs = new char[100];                                  // 创建字符数组
            System.out.println("请输入字符串：");                        // 提示用户输入字符串
            rin.read(cs);                                               // 向流中读取数据
            String input = new String(cs);                              // 根据读取内容创建字符串
            System.out.println("您输入的字符串：" + input.trim());       // 在控制台上输出内容
            rin.close();                                                // 关闭流
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

实例运行效果如图 16.15 所示。

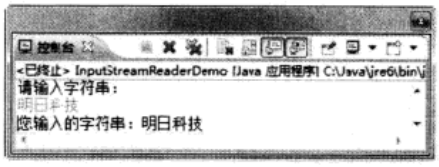


图 16.15　实例运行效果

16.3.2　Writer 类

与 Reader 类对应，Writer 类用于解决字符数据的输出流，该类是所有字符输出流的抽象类，子类必须实现的方法仅有 write(char cbuf,int off,int len)、flush()和 close()。但多数子类重写此类定义的一些方法，以达到更高的效率。Writer 类的常用方法如表 16.7 所示。

表 16.7 Writer 类的常用方法

方　法	返 回 值	说　明
close()	void	关闭当前输出流，并释放与当前输出流有关的系统资源
flush()	void	刷新当前输出流，并强制写入所有缓冲区的字节数据
write(char[] cbuf)	void	将字符数组中的数据写入到字符输出流
write(char cbuf,int off,int len)	void	将字符数组从下标 off 开始向输出流写入 len 长度的数据
write(String str)	void	向输出流写入一个字符串数据
write(String str,int off,int len)	void	向输出流写入一个从 off 开始、长度为 len 的字符数据

下面的实例通过 Writer 的子类实现使用字符流输出数据。

例 16.11 创建 Writer 的子类 PrintWriter 实例 out，并将其初始化为控制台的字符输出流。然后通过 write() 方法向流中写入字符数组，并在控制台上输出。实例代码如下：**（实例位置：光盘\mr\ym\16.11）**

```
public class PrintWriterDemo {
    public static void main(String[] args) {
        try {
            Writer out = new PrintWriter(System.out);                          // 创建 PrintWriter 实例
            char[] cs = "本实例使用字符输出流，在控制台输出字符串".toCharArray();   // 创建字符数组
            out.write(cs);                                                     // 向流中写数据
            out.close();                                                       // 将流关闭
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

实例运行效果如图 16.16 所示。

图 16.16 实例运行效果

16.3.3 FileReader 类

FileReader 类继承自 Reader 类，并实现了读取字符输入流的各种方法，可以从指定路径中读取字符数据。该类的构造方法如下：

☑ 在给定从中读取数据的 File 对象情况下，创建一个 FileReader 实例。该方法声明如下：

```
new FileReader(File file);
```

参数说明：file 是要读入数据的文件对象。

☑ 根据给定的文件名创建 FileReader 实例。该方法声明如下：

```
new FileReader(String fileName)
```

参数说明：fileName 表示与系统有关的文件名。

下面通过实例介绍使用 FileReader 类读取指定磁盘文件中的数据。

例 16.12 使用 FileReader 类实现读取 word.txt 文件中的数据，并将其在控制台上输出。实例代码如下：

（实例位置：光盘\mr\ym\16.12）

```
public class FileReaderDemo {
    public static void main(String[] args) {
        File file = new File("E://word.txt");                    // 创建文件对象
        try {
            FileReader fReader = new FileReader(file);           // 创建 FileReader 实例
            int length;
            while ((length = fReader.read()) != -1) {            // 循环读取指定文件中的内容
                System.out.print((char) length);
            }
            fReader.close();                                     // 关闭流
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

实例运行效果如图 16.17 所示。

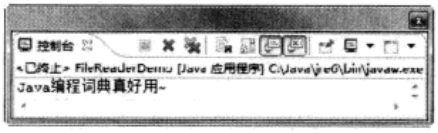


图 16.17　实例运行效果

16.3.4　FileWriter 类

FileWriter 类实现了文件字符输出流，该类继承自 Writer 类。它实现了写入字符文件的便捷性，文件字符输出流关联指定路径的文件，数据通过文件字符输出流以字符为单位输出并保存到文件中。可通过以下两种语法格式创建 FileWriter 对象。

☑　使用指定的 File 类型参数，创建与该文件关联的 FileWriter 实例。该方法声明如下：

```
new FileWriter(File file)
```

参数说明：file 是要写入数据的文件对象。

☑　根据指定的文件名称和路径，创建关联该文件的 FileWriter 类实例。该方法声明如下：

```
new FileWriter(String path)
```

参数说明：path 表示与系统有关的文件名。

下面通过实例介绍使用 FileReader 与 FileWriter 类实现文件的读取与写入。

例 16.13　使用字符输入、输出流，将文字信息加密后保存到指定文件，再读取。实例代码如下：**（实例位置：光盘\mr\ym\16.13）**

```
public class FileWriterDemo {
    public static void main(String[] args) {
        char chars[] = "登录密码是：mrsoft".toCharArray();          // 定义写入文件的字符数组
        int n = 0, m = 0;
        File file = new File("E://word.txt");                    // 写入信息的文件
        for (int i = 0; i < chars.length; i++) {                 // 将信息加密后写入文件
            chars[i] = (char) (chars[i] ^ 'A');
        }
```

```
        try {
            FileWriter fileWriter = new FileWriter(file);                    // 创建 FileWriter 对象
            fileWriter.write(chars, 0, chars.length);                        // 向文件中写数据
            fileWriter.close();                                              // 将流关闭
            FileReader fileReader = new FileReader(file);                    // 创建 FileReader 对象
            char tom[] = new char[1024];
            while ((n = fileReader.read(tom)) != -1) {                       // 循环读取指定文件中的内容
                String str = new String(tom, 0, n);                          // 创建字符串对象
                m = n;
                System.out.println("密文：" + str);                          // 将加密后的文件输出
            }
            fileReader.close();                                              // 将流关闭
            for (int j = 0; j < m; j++) {
                tom[j] = (char) (tom[j] ^ 'A');                              // 获取解密后的数据
            }
            String ss = new String(tom, 0, m);
            System.out.println("明文：" + ss);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

运行程序，将加密后的文字写入到 E 盘的 word.txt 文件中，如图 16.18 所示；然后将密文和明文输出，如图 16.19 所示。

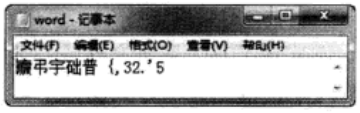


图 16.18　加密的文件信息

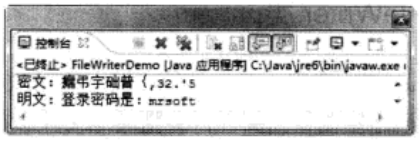


图 16.19　实例运行结果

从该实例的编译和运行结果看，使用字符输入输出流与字节输入输出流实现文件读取和写入的结果没有什么区别。使用 FileWriter 类向文件中写入字符串，不用将字符串转换为字节数组。相对于 FileOutputStream 来说，简化了编程工作。

16.3.5　Scanner 类

Scanner 类实现了使用正则表达式来解析基本类型和字符串的简单文本扫描器，该类位于 java.util 包中。Scanner 使用分隔符模式将其输入分解为标记，该类有多种构造方法。

☑　创建一个从指定文件中扫描的 Scanner 对象。该方法声明如下：

```
public Scanner(File source)
```

参数说明：source 是要扫描的文件对象。

☑　创建扫描指定字符串的 Scanner 对象。该方法声明如下：

```
public Scanner(String source)
```

参数说明：source 是要扫描的字符串。

☑　创建对指定输入流扫描的 Scanner 对象。该方法声明如下：

```
public Scanner(InputStream source)
```

参数说明：source 是要扫描的输入流。

Scanner 提供了不同的方法可将得到的标记转换为不同类型的值，如表 16.8 所示。

表 16.8　Scanner 类的常用方法

方　法	返 回 值	说　明
findInLine()	String	在忽略分隔符的情况下查找下一个从指定字符串构造的模式
nextInt()	int	扫描下一个 int 值
nextShort()	short	扫描下一个 short 值
nextFloat()	float	扫描下一个 float 值

下面通过实例介绍 Scanner 实例的应用。

例 16.14　使用正则表达式解析字符，并将满足条件的字符输出。实例代码如下：**（实例位置：光盘\mr\ym\16.14）**

```
public class ScannerDemo {
    public static void main(String[] args) {
        String input = "1 and 2 and yellow and blue and";                    // 定义要进行扫描的字符串
        Scanner scanner = new Scanner(input);                                // 创建 Scanner 实例
        scanner.findInLine("(\\d+) and (\\d+) and (\\w+) and (\\w+)");       // 查找指定字符串
        MatchResult result = scanner.match();                                // 执行扫描操作，返回匹配结果
        for (int i = 1; i <= result.groupCount(); i++)
            // 循环遍历扫描后的结果
            System.out.println(result.group(i));                             // 将扫描后的结果输出
        scanner.close();                                                     // 关闭扫描器，释放资源
    }
}
```

实例运行效果如图 16.20 所示。

图 16.20　实例运行效果

16.3.6　PrintWriter 类

在前面的程序中大量地用到了“System.out.println”语法输出信息。其中，System.out 是 PrintStream 类的一个实例对象。PrintStream 类位于 java.io 包中，该类提供了一系列的 print()和 pritln()方法，用于信息的输出。io 包中还提供了一个与 PrintStream 对应的 PrintWriter 类，PrintWriter 即使遇到换行符（‘\n’）也不会自动清空缓存区，只有在设置了 autoflush 模式的情况下使用 println()方法后才会自动清空缓存区。

PrintWriter 类的构造方法如下：

☑　使用指定文件创建不具有自动行刷新的 PrintWriter 实例。该方法声明如下：

```
public PrintWriter(File file)
```

参数说明：file 是作为此 writer 目标使用的文件。

☑ 创建具有指定文件名称且不带自动行刷新的 PrintWriter 实例。该方法声明如下：

public PrintWriter(String fileName)

参数说明：fileName 是用作此 writer 目标的文件名称。

☑ 根据 OutputStream 实例创建不带自动行刷新的 PrintWriter 实例。该方法声明如下：

public PrintWriter(OutputStream out)

参数说明：out 是输出流。

该类的常用方法如表 16.9 所示。

表 16.9 PrintWriter 类的常用方法

方 法	返 回 值	说 明
append(char c)	PrintWriter	将指定字符添加到此 writer 中，此方法等同于 out.write()
flush()	void	刷新该流的缓冲
print()	void	该方法提供了多种重载形式，用于打印字符
println()	void	该方法提供了多种重载形式，用于打印字符。该方法的行为如同先调用 print()方法，再调用 println()方法

Windows 的文本换行是“\r\n”，而在 Linux 操作系统下的文本换行是“\n”。如果希望程序生成平台相关的文本换行，而不是在各种平台下都用“\n”作为文本换行，就应该使用 PrintWriter 的 println()方法。该方法可以根据不同的操作系统而生成相应的换行符。

16.4 StringBuilder 类

对于创建成功的字符串对象，它的长度是固定的，内容不能被改变和编译。虽然使用符号“+”可以达到附加新字符或字符串的目的，但“+”会产生一个新的 String 实例，会在内存中创建新的字符串对象。如果重复地对字符串进行修改，将极大地增加系统开销。而 J2SE 5.0 新增了可变的字符序列 StringBuilder 类，会大大提高频繁做字符串增加的效率。通过下面简单的测试就可以知道其差距。

例 16.15 在项目中创建类 Jerque，在 main()方法中编写如下代码，验证字符串操作和字符串生成器操作的效率。实例代码如下：**（实例位置：光盘\mr\ym\16.15）**

```
public class StringBuilderDemo {
    public static void main(String[] args) {                    // 主方法
        String str = "";                                         // 创建空字符串
        long starTime = System.currentTimeMillis();              // 定义对字符串执行操作的起始时间
        for (int i = 0; i < 10000; i++) {                        // 利用 for 循环执行 10000 次操作
            str = str + i;                                       // 循环追加字符串
        }
        long endTime = System.currentTimeMillis();               // 获取系统时间
        long time = endTime - starTime;                          // 计算对字符串执行操作的时间
        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 (int j = 0; j < 10000; j++) {                        // 利用 for 循环进行操作
            builder.append(j);                                   // 循环追加字符
        }
        endTime = System.currentTimeMillis();                    // 获取操作后的时间
```

```
        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
    }
}
```

实例运行效果如图 16.21 所示。

通过本实例可以看出，执行的时间差距很大。如果在程序中频繁地附加字符串，建议使用 StringBuilder。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。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。若要使 StringBuilder 最后输出字符串结果，可使用 toString()方法。

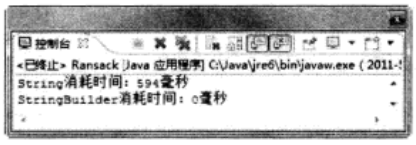


图 16.21　实例运行效果

16.4.1　StringBuilder 与 StringBuffer

java.lang 包中定义了 StringBuffer 类，该类是线程安全的可变字符序列，是一个类似于 String 的字符串缓冲区，但不能修改。StringBuffer 类是线程安全的，可以在必要时对字符串更新方法进行同步。

StringBuffer 主要操作 append()和 insert()方法，可重载这些方法，以接收任意类型的数据。每个方法都能有效地将给定的数据转换成字符串，然后将字符串的字符追加或插入到字符串缓冲区中。

StringBuilder 是 JDK 1.5 新增的类，是 StringBuffer 的一个简易替换，如果字符串缓冲区被单个线程使用，建议优先采用 StringBuilder 类。因为在大多数情况下，StringBuilder 类要比 StringBuffer 类运行速度快。StringBuilder 与 StringBuffer 类提供的外部方法是一样的。如果需要同步，则建议使用 StringBuffer。

例 16.16　使用 StringBuffer 类将字符串“Java 编程词典”倒置输出。实例代码如下：**（实例位置：光盘\mr\ym\16.16）**

```
public class StringBuilderDemo {
    public static void main(String[] args) {
        StringBuilder sb = new StringBuilder("Java 编程词典");    // 使用指定字符串创建 StringBuilder 对象
        System.out.println("逆序前对象：" + sb);                  // 输出对象
        System.out.println("逆序后对象：" + sb.reverse());        // 逆序输出对象
    }

}
```

实例运行效果如图 16.22 所示。

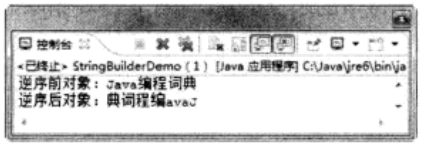


图 16.22　实例运行效果

例 16.17　使用 StringBuffer 类的 insert()方法实现向字符串中追加数据。实例代码如下：**（实例位置：光盘\mr\ym\16.17）**

```
public class StringBuilderDemo {
    public static void main(String[] args) {
        String str = new String("小明买了一本《Java 从基础到项目实战》");    // 定义字符串对象
        StringBuffer strBuffer = new StringBuffer(str);                    // 创建 StringBuffer 对象
```

```
        strBuffer.insert(0, "张");                                    // 向字符串中追加数据
        System.out.println(strBuffer.toString());                     // 将处理后的字符串输出
    }
}
```

实例运行效果如图 16.23 所示。

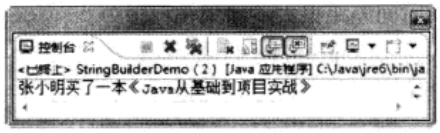


图 16.23　实例运行效果

16.4.2　常用方法

StringBuilder 与 StringBuffer 类提供的外部方法相同，都可动态地执行添加、删除、插入等字符串的编辑操作。该类的常用方法介绍如下。

☑　append()方法

该方法用于向字符串生成器中追加内容，通过该方法的多个重载形式，可实现接收任何类型的数据，如 int、boolean、char、String、double 或另一个字符串生成器等。append()方法的声明如下：

```
append(content)
```

参数说明：content 表示要追加到字符串生成器中的内容。可以是任何类型的数据或其他对象。

☑　intsert(int offset , arg)方法

该方法用于向字符串生成器中的指定位置插入数据内容。通过该方法的不同重载形式，可实现向字符串生成器中插入 int、float、char、boolean 等基本数据类型或其他对象。intsert(int offset,arg)方法的声明如下：

```
insert(int offset,arg)
```

参数说明：offset 是字符串生成器的位置。该参数必须大于等于 0，且小于等于此序列的长度。arg 是将插入至字符串生成器的位置。该参数可以是任何的数据类型或其他对象。

☑　delete(int start , int end)方法

移除此序列的子字符串中的字符。该子字符串从指定的 start 处开始，一直到索引 end-1 处的字符，如果不存在这种字符，则一直到序列尾部。如果 start 等于 end，则不发生任何更改。delete(int start , int end)方法的声明如下：

```
delete(int start , int end)
```

参数说明：start 是将要删除的字符串的起点位置，end 是将要删除的字符串的终点位置。

16.5　照猫画虎——基本功训练

16.5.1　基本功训练 1——复制单个文件

视频讲解：光盘\mr\lx\16\复制单个文件.exe

实例位置：光盘\mr\ym\16.18\src

对于操作系统而言，都提供了复制文件的功能。本实例将使用 Java 语言将文件从一个文件夹复制到另一个文件夹中。实例代码如下：

```
public class FileCopyDemo {
    public static void main(String[] args) {
        File source = new File("d:/test1/Java 编程词典.txt");          // 创建文件，作为要复制的文件
        File target = new File("d:/test2/Java 编程词典.txt");          // 创建文件，作为复制后的文件
        if (!target.exists()) {                                        // 如果复制后的文件不存在
            try {
                target.createNewFile();                                // 创建文件
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
        FileInputStream fis = null;                                    // 创建文件输入流对象
        FileOutputStream fos = null;                                   // 创建文件输出流对象
        try {
            fis = new FileInputStream(source);                         // 实例化文件输入流对象
            fos = new FileOutputStream(target);                        // 实例化文件输出流对象
            byte[] bytes = new byte[1024];                             // 创建字节数组
            int length;
            while ((length = fis.read(bytes)) != -1) {
                fos.write(bytes, 0, length);                           // 将源文件内容写入新文件
            }
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            if (fis != null) {
                try {
                    fis.close();                                       // 释放资源
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
            if (fos != null) {
                try {
                    fos.close();                                       // 释放资源
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }
    }
}
```

打开 test2 文件夹中新复制的文件，效果如图 16.24 所示。

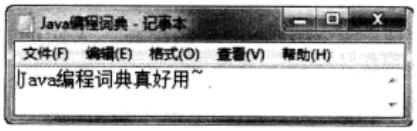


图 16.24　实例运行效果

注意：在使用本实例时，需要在 D 盘创建两个文件夹 test1 和 test2，并在 test1 中创建一个名为 Java 编程词典的文本文件。

照猫画虎：模仿本实例复制一个文件夹下的多个文件。（20 分）（实例位置：光盘\mr\ym\16.18\zmhh）

16.5.2 基本功训练 2——删除指定文件

视频讲解：光盘\mr\lx\16\删除指定文件.exe

实例位置：光盘\mr\ym\16.19\src

对于操作系统而言，都提供了删除文件的功能。本实例将使用 Java 语言将指定文件删除。实例代码如下：

```
public class FileDeleteDemo {
    public static void main(String[] args) {
        File file = new File("d:/test1/Java 编程词典.txt");                    // 指定要删除的文件
        if (file.delete()) {                                                   // 如果文件存在就将其删除
            System.out.println("文件" + file.getName() + "删除成功！");
        } else {
            System.out.println("文件" + file.getName() + "不存在！");
        }
    }
}
```

实例运行效果如图 16.25 所示。

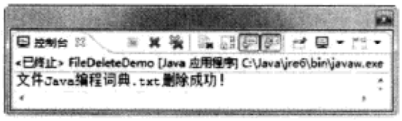


图 16.25 实例运行效果

你问我答：delete()方法有什么作用？

File 类中定义的 delete()方法用于删除文件和空文件夹，该方法的返回值是一个布尔值。如果文件成功删除则返回 true，否则返回 false。

注意：在使用本实例时，需要在 D 盘创建文件夹 test1。

照猫画虎：模仿本实例删除一个文件夹下的多个文件。（20 分）（实例位置：光盘\mr\ym\16.19\zmhh）

16.5.3 基本功训练 3——复制文件夹

视频讲解：光盘\mr\lx\16\复制文件夹.exe

实例位置：光盘\mr\ym\16.20\src

使用文件夹可以将文件分组管理，从而方便复制和删除。本实例将演示如何复制文件夹，实例代码如下：

```
public class DirectoryCopyDemo {
    public static void main(String[] args) throws IOException {
        File sources = new File("d:/test1");                                   // 源文件夹
```

```
            File targets = new File("d:/test2");                                    // 目标文件夹
            if (!targets.exists()) {                                                // 如果目标文件夹不存在
                targets.mkdir();                                                    // 创建文件夹
            }
            File[] source = sources.listFiles();                                    // 获得源文件夹中所有文件
            FileInputStream fis = null;
            FileOutputStream fos = null;
            for (File file : source) {
                fis = new FileInputStream(file);                                    // 创建输出流
                File target = new File(targets.getAbsolutePath() + File.separator + file.getName());
                if (!target.exists()) {
                    target.createNewFile();                                         // 创建新文件
                }
                fos = new FileOutputStream(target);
                byte[] bytes = new byte[1024];                                      // 创建字节数组
                int length;
                while ((length = fis.read(bytes)) != -1) {
                    fos.write(bytes, 0, length);                                    // 将源文件内容写入新文件
                }
                System.out.println(file.getName() + " 复制完毕！");
            }
        }
    }
```

实例运行效果如图 16.26 所示。

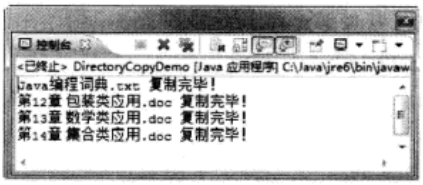


图 16.26　实例运行效果

你问我答：mkdir()方法有什么作用？

File 类中定义的 mkdir()方法用于创建指定的文件夹，该方法的返回值是一个布尔值。如果文件夹成功创建则返回 true，否则返回 false。

照猫画虎：在本实例的基础上增加一个计数器统计复制的文件个数。（20 分）（实例位置：光盘\mr\ym\16.20\zmhh）

16.5.4　基本功训练 4——删除含有文件的文件夹

视频讲解：光盘\mr\lx\16\删除含有文件的文件夹.exe

实例位置：光盘\mr\ym\16.21\src

虽然 File 类中提供了 delete()方法，但它只能删除空文件夹和文件。本实例将使用其删除含有文件的文件夹，思路是先删除文件夹中的全部文件，再删除文件夹。实例代码如下：

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

```
        File directory = new File("D:/test1");                    // 获得要删除的文件夹
        File[] files = directory.listFiles();                     // 获得文件夹中的全部文件
        for (File file : files) {                                 // 删除文件夹中的全部文件
            file.delete();
            System.out.println(file.getName() + "被删除！");
        }
        directory.delete();                                       // 删除空文件夹
    }
}
```

实例运行效果如图 16.27 所示。

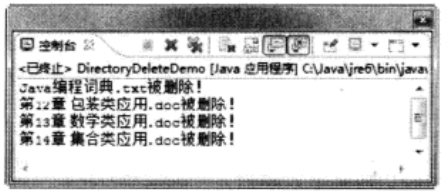


图 16.27　实例运行效果

照猫画虎：在本实例的基础上增加一个计数器统计删除的文件个数。（20 分）（实例位置：光盘\mr\ym\16.21\zmhh）

16.5.5　基本功训练 5——压缩指定文件

视频讲解：光盘\mr\lx\16\压缩指定文件.exe

实例位置：光盘\mr\ym\16.22\src

在 Java API 中提供了创建压缩文件的功能，本实例将使用其实现压缩指定文件的功能，实例代码如下：

```
public class FileZipDemo {
    public static void main(String[] args) {
        File source = new File("d:/password.txt");                // 获得需要压缩的文件
        File target = new File("d:/数据库密码.zip");               // 创建压缩后的文件
        FileInputStream fis = null;
        FileOutputStream fos = null;
        ZipOutputStream zos = null;
        try {
            fis = new FileInputStream(source);                    // 创建文件输入流
            fos = new FileOutputStream(target);                   // 创建文件输出流
            zos = new ZipOutputStream(fos);                       // 创建压缩输出流
            byte[] buffer = new byte[1024];
            ZipEntry entry = new ZipEntry(source.getName());      // 利用文件的名字创建 ZipEntry 对象
            zos.putNextEntry(entry);                              // 在压缩文件中添加一个 ZipEntry 对象
            int read = 0;
            while ((read = fis.read(buffer)) != -1) {
                zos.write(buffer, 0, read);                       // 将输入写入到压缩文件
            }
            zos.closeEntry();                                     // 关闭 ZipEntry
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
```

```
            e.printStackTrace();
        } finally {
            if (zos != null) {
                try {
                    zos.close();                                    // 关闭压缩输出流
                    System.out.println("文件压缩完毕！");
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
            if (fos != null) {
                try {
                    fos.close();                                    // 关闭文件输出流
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
            if (fis != null) {
                try {
                    fis.close();                                    // 关闭文件输入流
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }
    }
}
```

实例运行效果如图 16.28 所示。

图 16.28　实例运行效果

你问我答：如何压缩文件？

首先确定要压缩的文件和压缩后的文件，并利用其创建文件输入流和文件输出流。再利用文件输出流创建压缩输出流。对于每个文件，需要创建一个 ZipEntry 对象。通常使用文件名作为其构造方法的参数。然后将文件写入到 ZipEntry 对象中。最后关闭该对象和输入流、输出流对象。

注意：关闭流的顺序要和打开的顺序相反。

注意：使用 Java 提供的压缩工具压缩中文文件会出现乱码的情况。

照猫画虎：模仿本实例压缩文件夹中的全部文件。（20 分）（实例位置：光盘\mr\ym\16.22\zmhh）

照猫画虎栏目分数统计：

照猫画虎题目	1	2	3	4	5	总分数	
分数							

16.6 情景应用——拓展与实践

16.6.1 情景应用 1——删除文件夹中所有.tmp 临时文件

视频讲解：光盘\mr\lx\16\删除文件夹中所有.tmp 临时文件.exe

实例位置：光盘\mr\ym\16.23\src

对于 Windows 操作系统而言，运行中会产生很多后缀名为.tmp 的临时文件，它们会占用大量的磁盘空间并影响系统的运行速度。本实例将删除文件夹中全部.tmp 文件，其思路是遍历文件夹中所有文件，如果文件以.tmp 结尾，则删除。实例运行效果如图 16.29 所示。

图 16.29 实例运行效果

实现过程如下：

（1）在项目中新建类 TempFilesDelete。

（2）在该类的 main()方法中，首先确定需要遍历的文件夹，然后使用 listFiles()方法获得文件夹中所有临时文件。此处使用了一个 FileFilter 接口的匿名实现类。通过重写 accept()方法可以确定接受哪些文件。对于文件类型的判断，在 Windows 系统中可以使用文件的扩展名，实例的完整代码如下：

```
public class TempFilesDelete {
    public static void main(String[] args) {
        File rootFile = new File("d:/temp");                    // 获得要删除.tmp 文件的文件夹
        File[] files = rootFile.listFiles(new FileFilter() {    // 获取文件夹中所有.tmp 文件
                    @Override
                    public boolean accept(File pathname) {
                        if (pathname.getName().endsWith(".tmp")) {
                            return true;
                        } else {
                            return false;
                        }
                    }
                });
        for (File file : files) {
            file.delete();                                      // 删除文件
            System.out.println(file.getName() + "被删除！ ");
        }
    }
}
```

你问我答：accept()方法有什么作用？

accept()方法是在 FileFilter 接口中定义的，该方法可以用来过滤文件。该方法声明如下：

boolean accept(File pathname)

参数说明：pathname 是要测试的文件。通常以文件的后缀名、创建时间等条件来过滤文件。

注意：在 D 盘上创建 temp 文件夹并在其中放置任意个以.tmp 为后缀的文件。

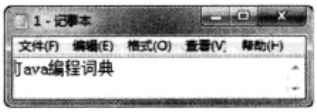
DIY：模仿本实例删除文件夹中所有的 txt 文件。（20 分）（实例位置：光盘\mr\ym\16.23\diy）

16.6.2　情景应用 2——合并多个 txt 文件

视频讲解：光盘\mr\lx\16\合并多个 txt 文件.exe

实例位置：光盘\mr\ym\16.24\src

对于文件的读写操作，可以使用流技术。此外，还可以用它来完成文件的合并操作。本实例将合并多个 txt 文件，其思路是依次为各个文件创建输入流，然后统一将它们写入到一个文件中。需要合并的文本文件如图 16.30 所示，合并完成后的文本文件如图 16.31 所示。



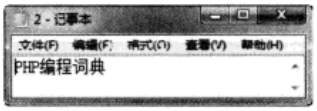


图 16.30　需要合并的文本文件

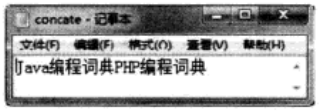


图 16.31　合并后的文件内容

实现过程如下：

（1）在项目中新建类 FileConcatenation。

（2）在 FileConcatenation 类的 main()方法中，先确定要合并的文件所在的文件夹，然后遍历该文件夹中所有 txt 文件。使用文件输出流将所有文件写入到一个新的文件中，使用 for 循环来简化操作。对于期间出现的各个异常，使用 try…catch 进行处理，最后释放资源。实例的完整代码如下：

```
public class FileConcatenation {
    public static void main(String[] args) {
        File rootFile = new File("d:/test");                    // 获得要合并 txt 文件的文件夹
        File[] files = rootFile.listFiles(new FileFilter() {    // 获取文件夹中所有 txt 文件
                    @Override
                    public boolean accept(File pathname) {
                        if (pathname.getName().endsWith(".txt")) {
                            return true;
                        } else {
                            return false;
                        }
                    }
                });
        File concatedFile = new File("d:/concate.txt");         // 声明合成后的文件
        FileInputStream fis = null;
        FileOutputStream fos = null;
        BufferedInputStream bis = null;
        try {
            fos = new FileOutputStream(concatedFile);           // 创建文件输出流对象
            for (File file : files) {                           // 遍历所有要合成的文件
                fis = new FileInputStream(file);                // 创建文件输入流
                bis = new BufferedInputStream(fis);             // 利用文件输入流创建缓冲输入流
                int length = 0;
                while ((length = bis.read()) != -1) {
                    fos.write(length);                          // 将输入流写入新文件
                }
            }
```

```
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            if (bis != null) {
                try {
                    bis.close();                                    // 关闭缓冲输入流
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
            if (fis != null) {
                try {
                    fis.close();                                    // 关闭文件输入流
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
            if (fos != null) {
                try {
                    fos.close();                                    // 关闭文件输出流
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }
    }
}
```

说明：在运行本实例前要先在 D 盘创建一个名为 test 的文件夹，并在其中放置任意个文本文件。实例运行后会在 D 盘生成一个名为 concate 的文本文件。

DIY：模仿本实例合并文件夹中所有 mp3 文件。（20 分）（实例位置：光盘\mr\ym\16.24\diy）

16.6.3 情景应用 3——查看文件夹中所有隐藏文件

视频讲解：光盘\mr\lx\16\查看文件夹中所有隐藏文件.exe

实例位置：光盘\mr\ym\16.25\src

对于操作系统中的文件，通常具有很多属性，如是否是隐藏文件、是否可读写等。对于不希望其他用户看到的文件，可以将其设置成隐藏状态。对于系统文件，可以将其设置成只读状态，防止用户误写入。本实例将查看文件夹中所有隐藏文件，实例的运行效果如图 16.32 所示。

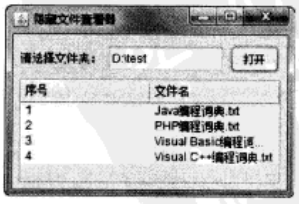


图 16.32 实例运行效果

实现过程如下：

（1）在 Eclipse 中创建 JFrame 窗体，名称为 HiddenFilesViewer。

（2）在窗体中增加标签、表格等控件。

（3）为“打开”按钮增加事件监听器，它是通过 do_button_ actionPerformed()方法完成的。在该方法中，

创建了文件选择器来让用户选择要查看的文件夹，通过为 listFiles()方法增加文件过滤器来去掉非隐藏文件。最后使用一个表格来显示所有隐藏文件。关键代码如下：

```
protected void do_button_actionPerformed(ActionEvent e) {
    JFileChooser chooser = new JFileChooser();                          // 创建文件选择器
    chooser.setFileSelectionMode(JFileChooser.DIRECTORIES_ONLY);// 设置文件选择器只能选择文件夹
    int result = chooser.showOpenDialog(this);                          // 保存文件选择器的操作结果
    if (result == JFileChooser.APPROVE_OPTION) {
        File directory = chooser.getSelectedFile();                     // 获得用户选择的文件夹
        textField.setText(directory.getAbsolutePath());                 // 在文本框中显示文件夹所在路径
        File[] hiddenFiles = directory.listFiles(new FileFilter() {     // 过滤掉非隐藏文件
                    @Override
                    public boolean accept(File pathname) {
                        if (pathname.isHidden()) {
                            return true;
                        } else {
                            return false;
                        }
                    }
                });
        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 count = 1;
        for (File hiddenFile : hiddenFiles) {
            model.addRow(new Object[] { count++, hiddenFile.getName() }); // 增加行
        }
        table.setModel(model);                                          // 更新表格模型
    }
}
```

DIY：模仿本实例查看文件夹中所有可读文件。（20 分）（实例位置：光盘\mr\ym\16.25\diy）

16.6.4　情景应用 4——简单键盘记录器

视频讲解：光盘\mr\lx\16\简单键盘记录器.exe

实例位置：光盘\mr\ym\16.26\src

对于木马、病毒等，通常可以监视用户在键盘上的操作，如输入用户名和密码，从而盗取账号。本实例将完成一个类似的功能，它主要用于记录用户名和密码，并且明文显示在文本文件中。实例的运行效果如图 16.33 所示。记录器生成的文本文件内容如图 16.34 所示。

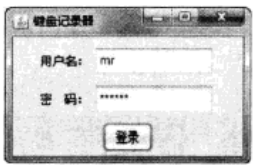


图 16.33　实例运行效果

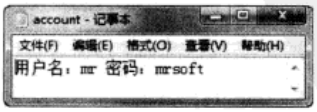


图 16.34　键盘记录结果

实现过程如下：

（1）在 Eclipse 中创建 JFrame 窗体，名称为 KeyBoardListener。

（2）在窗体中增加标签、按钮等控件。

（3）为“登录”按钮增加事件监听器，它是通过 do_button_actionPerformed()方法完成的。在该方法中，首先获得用户输入的用户名和密码，如果两者有一个为空，则提示用户进行输入。然后使用文件输出流将其写入到本地文本文件中。关键代码如下：

```
protected void do_button_actionPerformed(ActionEvent e) {
    String username = textField.getText();                          // 获得用户名
    String password = new String(passwordField.getPassword());      // 获得密码
    if ((username.isEmpty()) || (password.isEmpty())) {             // 判断用户名或密码是否为空
        JOptionPane.showMessageDialog(this, "用户名或密码为空！", null, JOptionPane.WARNING_MESSAGE);
        return;
    }
    File file = new File("d:/account.txt");                         // 创建文件保存账号信息
    String content = "用户名：" + username + " 密码：" + password;    // 获得要写入的字符串
    FileOutputStream fos = null;
    try {
        fos = new FileOutputStream(file);                           // 创建文件输出流对象
        fos.write(content.getBytes());                              // 将字符串写入到文件中
    } catch (FileNotFoundException e1) {
        e1.printStackTrace();
    } catch (IOException e1) {
        e1.printStackTrace();
    } finally {
        try {
            fos.close();                                            // 关闭文件输出流
        } catch (IOException e1) {
            e1.printStackTrace();
        }
    }
}
```

DIY：在本实例的基础上将记录文件设置成隐藏的。（20 分）（实例位置：光盘\mr\ym\16.26\diy）

16.6.5 情景应用 5——记录用户的登录时间

视频讲解：光盘\mr\lx\16\记录用户的登录时间.exe

实例位置：光盘\mr\ym\16.27\src

出于安全性的考虑，需要记录用户登录的详细信息。这样如果系统发生问题，可以方便地找到责任人员。本实例将使用一个文本文件来保存用户名和登录系统的时间。实例的运行效果如图 16.35 所示，记录器生成的文本文件内容如图 16.36 所示。

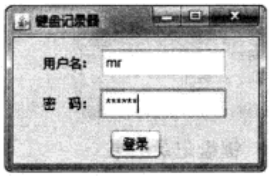


图 16.35 实例运行效果

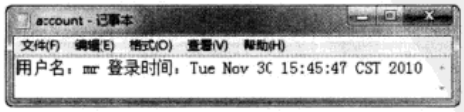


图 16.36 键盘记录结果

实现过程如下：

（1）在 Eclipse 中创建 JFrame 窗体，名称为 KeyBoardListener。

（2）在窗体中增加标签、按钮等控件。

（3）为“登录”按钮增加事件监听器，它是通过 do_button_actionPerformed()方法完成的。在该方法中，首先获得用户输入的用户名和密码，如果两者有一个为空，则提示用户进行输入。然后使用文件输出流将用户名和当前时间写入到本地文本文件中。该方法的代码如下：

```
protected void do_button_actionPerformed(ActionEvent e) {
    String username = textField.getText();                                  // 获得用户名
    String password = new String(passwordField.getPassword());              // 获得密码
    if ((username.isEmpty()) || (password.isEmpty())) {                     // 判断用户名或密码是否为空
        JOptionPane.showMessageDialog(this, "用户名或密码为空！", null, JOptionPane.WARNING_MESSAGE);
        return;
    }
    File file = new File("d:/account.txt");                                 // 创建文件保存账号信息
    String content = "用户名：" + username + " 登录时间：" + new Date();     // 获得要写入的字符串
    FileOutputStream fos = null;
    try {
        fos = new FileOutputStream(file);                                   // 创建文件输出流对象
        fos.write(content.getBytes());                                      // 将字符串写入到文件中
    } catch (FileNotFoundException e1) {
        e1.printStackTrace();
    } catch (IOException e1) {
        e1.printStackTrace();
    } finally {
        try {
            fos.close();                                                    // 关闭文件输出流
        } catch (IOException e1) {
            e1.printStackTrace();
        }
    }

}
```

DIY：在本实例的基础上将记录文件中的日期样式修改成“年月日小时分秒”。（20 分）（实例位置：光盘\mr\ym\16.27\diy）

情景应用 DIY 栏目分数统计：

DIY 题目	1	2	3	4	5	总分数	
分数							

16.7　自 我 测 试

一、选择题（每题 10 分，5 道题）

1．下列代码的运行结果是？（　　）

```
public class Test {
    public static void main(String[] args) {
        File file = new File("d:\mrsoft.txt");
        try {
            file.createNewFile();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

A．编译错误　　B．运行错误

C．在 D 盘上创建 mrsoft.txt 文件　　D．在 D 盘上创建 mrsoft 文件

2．使用 File 类中哪个方法可以判断文件是否是隐藏的？（　　）

A．isHidden()　　B．canHidden()　　C．isHiddened()　　D．canHiddened()

3．如果要删除 StringBuilder 中的全部字符，可以使用哪个方法？（　　）

A．delete()方法　　B．empty()方法　　C．clear()方法　　D．reset()方法

4．下列代码的运行结果是？（　　）

```
public class Test {
    public static void main(String[] args) {
        StringBuilder sb = new StringBuilder();
        System.out.println(sb.capacity());
        System.out.println(sb.length());
    }
}
```

A．在控制台上输出 16 和 0　　B．在控制台上输出 12 和 0

C．在控制台上输出 10 和 0　　D．在控制台上输出 8 和 0

5．如果使用 Java 自带的类实现创建 ZIP 格式压缩文件的功能，至少需要使用下面哪些(个)类？（　　）

A．ZipInputStream　　B．ZipEntry　　C．ZipOutputStream　　D．StringBuffer

二、填空题（每题 10 分，5 道题）

1．File 类中是否有隐藏文件的方法？_____

2．File 类的 delete()方法能否直接删除非空文件夹？_____

3．如果需要将文件命名统一成一个样式，如“明日科技 01”、“明日科技 02”等，则可以使用 File 类中的哪个方法实现重命名功能？_____

4．如果需要线程安全的可变字符序列，则可以使用哪个类？_____

5．使用 File 类中的 delete()方法来删除文件，如果该文件并不存在，则方法的返回值是？_____

测试分数统计：

类别	第 1 题	第 2 题	第 3 题	第 4 题	第 5 题
选择题分数					
填空题分数					
总分数					

16.8　行 动 指 南

开始日期：_____年___月___日　　结束日期：_____年___月___日

序号	内　容	行 动 指 南	
1	照猫画虎栏目	分数>75 分	优秀，基本功掌握得不错，加油！
		75 分>分数>50 分	及格，知识掌握得不牢，重新做一遍照猫画虎。
	分数（　　）	分数<50 分	请使用光盘中提供的“实战能力测试系统”进行提高训练。
	情景应用栏目	分数>75 分	优秀，综合应用能力很强。
		75 分>分数>50 分	及格，综合应用能力需提高，再练一遍情景应用。
	分数（　　）	分数<50 分	请使用光盘中提供的“实战能力测试系统”进行提高训练。
	自我测试栏目	分数>75 分	优秀，有成为编程高手的潜质。
	分数（　　）	分数<75 分	请使用光盘中提供的“实战能力测试系统”进行提高训练。
	综合评价	反复训练后，以上各项分数都在 75 分以上，方可进入下一堂课学习。	
2	编程实践能力培养：本栏目提供了一些实践题目，对于培养编程能力很有效，要做好。	（1）删除磁盘中所有.tmp 临时文件。	
		（2）快速移动文件。	
		（3）将图片文件写入到数据库中。	
		（4）解压缩 ZIP 文件。	
3	编程习惯培养：本堂课培养读者在学习开发中记笔记的习惯，把开发中遇到的问题、总结的经验记录下来。		
4	创新能力培养：看看身边有没有可以用编程解决的问题，记录到右边的表格中。根据学习进度，尝试编写解决这些问题的小程序。		

16.9　成功可以复制——世界第一人称射击游戏之父

出生于德克萨斯州的约翰·卡马克（John Carmack）与许多电脑天才一样，从小就对电脑和程序设计充满狂热，七年级时的卡马克对《创世纪》和《巫术》等游戏痴迷不已，开始制作基于 AppleII 的小游戏。由于对编程的热爱，卡马克中途放弃了在密苏里大学就读计算机科学专业的机会，开始了自己艰辛的创业之路。为了获取 Softdisk 公司的程序员职位，他在雪地中徒步走了三英里才得以赶到招聘地点。在这里，约翰·卡马克结识了约翰·罗梅洛（John Romero《文明》系列游戏的缔造者）、汤姆·豪尔（Tom Hall）等意气相投的好友，随后他们一同返回德克萨斯州建立了自己的游戏公司——ID Software。

约翰·卡马克是一个天才的程序设计师，仅仅依靠自学和钻研就掌握了高深的程序设计技巧，他甚至相信可以用编程完成一切。在当今这个 3D 技术飞速发展的时代，没有哪一家公司可以像 ID Software 那样

引领技术潮流，也没有哪一家公司可以让硬件制造商俯首称臣，这都要归功于约翰·卡马克开发的 3D 加速技术。《雷神之锤》刚刚问世的时候，3D 加速卡在人们眼里还只是一个可笑的空想而已，只有约翰·卡马克对 3D 技术的威力深信不疑，他为《雷神之锤》制作了一个专门在 Verite 显卡上运行的特别版本，画面看上去非常漂亮，可惜的是 Verite 显卡未能在市场上站稳脚跟。随后卡马克又采用 OpenGL 标准为《雷神之锤》制作了一个新的版本，使所有具备 3D 加速能力的显卡都能以更快的速度、更高的分辨率渲染出更华丽的图像。一时间，所有的电脑用户都争相购买这款游戏，人们甚至为了能玩上 3D 游戏而去购买昂贵的 PC，这给 ID Software 和约翰·卡马克带来了上亿美元的商业利润。直到今天，显卡生产商在研发新产品之前还会先同约翰·卡马克商量一下，以确保他们的硬件可以完美地支持 ID Software 出品的游戏，并确保硬件性能符合时代的潮流。

现在，约翰·卡马克仍然担当着 ID Software 的游戏软件首席程序员，出自他手的电脑游戏已达数十部之多。但卡马克并不满足于现状，他的最新目标是有关火箭技术的研究项目，并为此特意成立了 Armadillo 航空宇宙技术研究公司，为着又一个梦想继续努力。

亲自开发的游戏有德军总部 3D（Wolfenstein 3D）、毁灭战士（Doom）和雷神之锤（Quake）等十多部经典游戏。

使用其游戏引擎开发的游戏有反恐精英和荣誉勋章等。

经典语录

在信息时代，客观障碍已经不复存在，所谓障碍都是主观上的。如果你想动手开发什么，不需要几百万美元的资金，只需要在冰箱里面放满比萨和可乐，再有一台便宜的计算机，和为之献身的决心。我们在地板上睡过，我们从河水中趟过。

深度评价

如果你问比尔·盖茨，“你最欣赏的程序员有哪些？”在他的回答中，肯定会出现 John Carmack 的名字。没错，John Carmack 在程序员中的崇拜程度，甚至到了无以为加的地步，微软在很多问题上都非常尊重他的意见。正是因为对游戏和编程的热爱和孜孜不倦的追求，才使得高中毕业的 John Carmack 成为享誉世界的著名程序员，并开创了游戏领域的最大传奇。

第17堂课

网络程序设计

（视频讲解：108 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的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。本堂课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。

学习摘要：

- TCP 与 UDP 网络协议
- 端口和套接字
- InetAddress 类的应用
- ServerSocket 类的应用
- Socket 类的应用
- DatagramPacket 类的应用
- DatagramSocket 类的应用

17.1 网 络 基 础

为了实现两台计算机的通信，必须要有一个网络线路连接两台计算机。由服务器提供信息，客户机请求信息，网络连接服务器与客户机实现两者相互通信，如图 17.1 所示。

图 17.1 服务器、客户机和网络

通常人们所说的“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，也可以由同一区域里面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）。我们熟悉的因特网（Internet），则是由无数的 LAN 和 WAN 组成的。

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，就可以编写出高质量的网络通信程序。

17.1.1 TCP/IP 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。

Internet 网络采用的协议是 TCP/IP 协议。它的全称是 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，每一个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 32 位地址长度，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 128 位地址长度，这就是 Ipv6。据估算，Ipv6 实际上在整个地球的每平方米面积上可分配 1000 多个地址。在 Ipv6 的设计过程中解决了地址短缺问题。

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 17.2 所示。

图 17.2 TCP/IP 层次结构

17.1.2 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，分别是传输控制协议

（Transmission Control Protocol，TCP）和用户数据报协议（User Datagram Protocol，UDP）。

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 可以保证从一端数据送至连接的另一端时，数据能够确实送达，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，也就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，可能会出现一个混乱的 HTML 文件或一些无效的信息。

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 协议在认证上的额外耗费，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，也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。

17.1.3　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 17.3 所示。

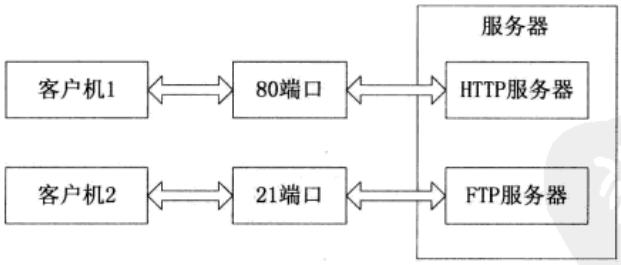


图 17.3　端口

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，来避免端口号与另一个应用或系统服务所用。

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 17.4 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。

TCP 网络程序设计是指利用 Socket 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 17.5 所示。

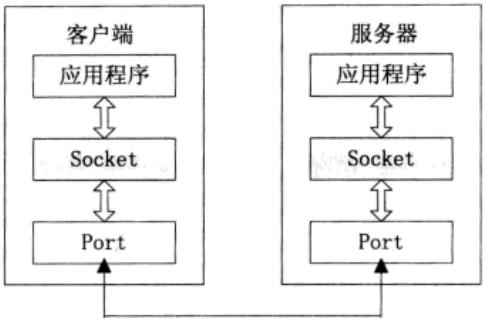


图 17.4　套接字

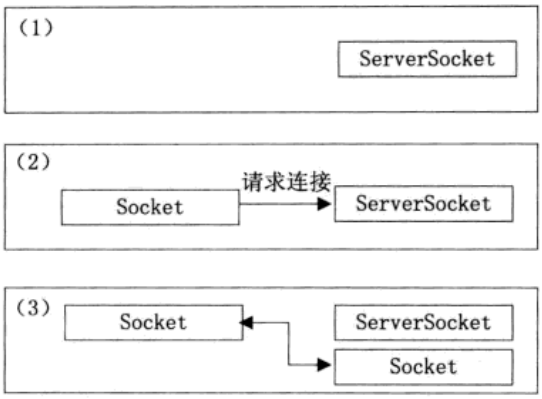


图 17.5　服务器套接字

（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）服务器接受客户的连接请求，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。

17.2　InetAddress 地址类

Internet 上的主机有两种方式表示地址，分别为域名和 IP 地址。java.net 包中的 InetAddress 类对象包含有一个 Internet 主机地址的域名和 IP 地址。该类的常用方法如表 17.1 所示。

表 17.1　InetAddress 类的常用方法

方　法	返 回 值	功 能 描 述
getByName(String host)	InetAddress	在给定主机名的情况下确定主机的 IP 地址
getHostAddress()	String	返回 IP 地址字符串
getHostName()	String	获取此 IP 地址的主机名
get(int index)	Object	用于获取指定索引位置的对象
getLocalHost()	InetAddress	返回本地主机的 InetAddress 对象
toString()	String	将此 IP 地址转换为 String

InetAddress 类提供的 getByName()方法，可以将一个域名或 IP 地址传递给该方法的参数，来获取一个 InetAddress 对象，该对象含有主机地址的域名和 IP 地址。该对象用如下格式表示其包含的信息：

www.mingrisoft.com/192.168.1.249

1．获取局域网中的主机地址

通过该方法可以获取 Internet 或局域网上的主机地址，下面的实例是在局域网中获取名称为 lzw 的电脑的主机名和 IP 地址。由于无法连接名称指定的主机时 getByName()方法会抛出 UnknownHostException 异常，

因此在调用该方法时，应该捕获该异常。

例 17.01　获取局域网中的主机域名和 IP 地址。实例代码如下：**（实例位置：光盘\mr\ym\17.01）**

```
public static void main(String[] args) {
    try {
        InetAddress intAddress = InetAddress
                .getByName("lzw");
        // 获取 InetAddress 实例
        System.out.println(intAddress.toString()); // 将该对象的主机名称和主机地址输出
    } catch (UnknownHostException e) {
        e.printStackTrace();
    }
}
```

上述程序中 lzw 是笔者的计算机名称，可以把它替换为局域网中其他计算机的名称或 Internet 中的网址。实例运行结果如图 17.6 所示。

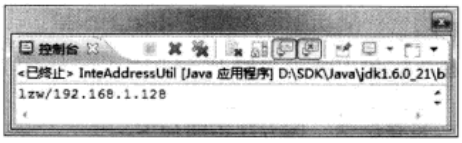


图 17.6　本实例的运行结果

2．获取本地主机地址

根据 InetAddress 类的静态方法 getLocalHost()，可以获取含有本地主机域名和 IP 地址的 InetAddress 对象。下面的例子分别显示了本机名和 IP 地址。

例 17.02　获取本机名和 IP 地址。实例代码如下：**（实例位置：光盘\mr\ym\17.02）**

```
public static void main(String[] args) {
    InetAddress ip;                                    // 创建 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();           // 获取本机名
        String localip = ip.getHostAddress();          // 获取本机 IP 地址
        System.out.println("本机名：" + localname);     // 将本机名输出
        System.out.println("本机 IP 地址：" + localip); // 将本机 IP 输出
    } catch (UnknownHostException e) {
        e.printStackTrace();                           // 输出异常信息
    }
}
```

运行程序，结果如图 17.7 所示。

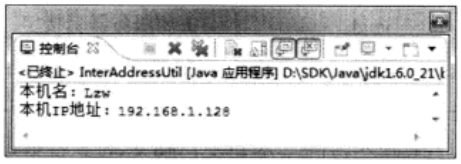


图 17.7　本实例的运行结果

17.3 TCP 通信

TCP 是一种网络传输控制协议，它是面向连接的、可靠的、基于字节流的一种数据传输协议。在 Java 中通过 Socket 来实现 TCP 协议。

17.3.1 ServerSocket 类

ServerSocket 用来表示服务器套接字。服务器套接字通过指定的端口来等待连接的套接字，它的主要功能是等待来自网络上的连接“请求”。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。队列的默认大小是 50。

ServerSocket 类的构造方法如下：

☑ 创建非绑定服务器的套接字。该方法声明如下：

```
public ServerSocket();
```

☑ 创建绑定特定端口的服务器套接字。该方法声明如下：

```
public ServerSocket(int port)
```

参数说明：port 用于指定绑定的端口号。

☑ 指定队列的最大长度，创建绑定到指定本地端口号的套接字。该方法声明如下：

```
public ServerSocket(int port,int backlog)
```

参数说明：port 用于指定绑定的端口号，backlog 指定队列的最大长度。在创建 ServerSocket 时，可能会抛出 IOException 异常，所以要进行异常捕获。例如，创建 ServerSocket 实例的代码如下：

```
try {
    ServerSocket serverSocket = new ServerSocket(1880);
} catch (IOException e) {
    e.printStackTrace();
}
```

ServerSocket 类的常用方法如表 17.2 所示。

表 17.2 ServerSocket 类的常用方法

方 法	返 回 值	说 明
accept()	Socket	等待客户机的连接。若连接，则创建一个套接字
isBound()	boolean	判断 ServerSocket 的绑定状态
getInetAddress()	InetAddress	返回此服务器套接字的本地地址
isClosed()	boolean	返回服务器套接字的关闭状态
close()	void	关闭服务器套接字
bind(SocketAddress endpoint)	void	将 ServerSocket 绑定到特定地址（IP 地址和端口号）
getLocalPort()	int	返回服务器套接字等待的端口号

调用 ServerSocket 类的 accept()方法会返回一个和客户端 Socket 对象相连接的 Socket 对象，服务器端的 Socket 对象使用 getOutputStream()方法获得的输出流将指向客户端 Socket 对象使用 getInputStream()方法获得

的那个输入流；同样，服务器端的 Socket 对象使用 getInputStream()方法获得的输入流将指向客户端 Socket 对象使用 getOutputStream()方法获得的那个输出流。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，同样如此，如图 17.8 所示。

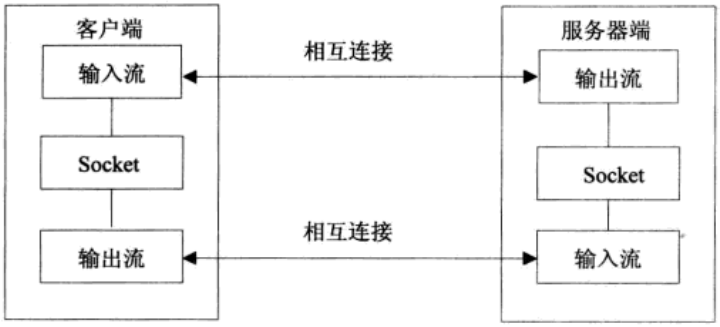


图 17.8　套接字连接示意图

注意：accept()方法会堵塞当前线程的继续执行，直到接收到客户的呼叫。

下面的实例成功创建服务器端套接字后，如果没有客户机请求连接，则不会输出“与客户机建立连接”信息。

例 17.03　创建服务器端套接字。实例代码如下：**（实例位置：光盘\mr\ym\17.03）**

```
public static void main(String[] args) {
    try {
        ServerSocket serverSocket = new ServerSocket(1880);   // 创建服务器端套接字
        System.out.println("成功创建服务器套接字");
        while (true) {
            System.out.println("等待客户机的连接");
            Socket socket = serverSocket.accept();              // 等待与客户机的连接
            System.out.println("与客户机建立连接");
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

本实例运行结果如图 17.9 所示。该程序需要单击■按钮手动关闭。

图 17.9　本实例的运行结果

17.3.2　Socket 类

Socket 类是套接字类。使用 Socket 时，需要指定待连接服务器的 IP 地址和端口号。客户机创建了 Socket 对象以后，会向指定的 IP 地址及端口尝试连接。服务器套接字会创建新的套接字，与客户端套接字建立连

接。服务器端套接字与客户端套接字成功连接后，则可以获取套接字的输入输出流，进行数据交换。

Socket 类常用的构造方法介绍如下。

☑ 创建到服务器连接的套接字。该方法声明如下：

```
public Socket(String host,int port)
```

参数说明：host 用于指定服务器的 IP 地址，port 是连接服务器的端口号。在创建 Socket 时，可能发生 IOException 与 UnknownHostException 异常，所以创建套接字时需要捕获异常。例如，创建 Socket 对象的关键代码如下：

```
try {
    Socket scoket = new Socket("192.168.1.128",8080);
} catch (UnknownHostException e) {
    e.printStackTrace();
} catch (IOException e) {
    e.printStackTrace();
}
```

☑ 通过 InetAddress 类创建 Socket 类对象。该方法声明如下：

```
public Socket(InetAddress address,int port)
```

参数说明：address 参数是一个计算机地址 InetAddress 类的对象，port 是连接服务器的端口号。例如，下面的代码就使用这个构造方法创建了 Socket 类的对象：

```
try {
    InetAddress address = InetAddress.getByName("192.168.1.128");
    Socket scoket = new Socket(address,1433);
} catch (UnknownHostException e) {
    e.printStackTrace();
} catch (IOException e) {
    e.printStackTrace();
}
```

该类的常用方法如表 17.3 所示。

表 17.3 Socket 类的常用方法

方 法	返 回 值	说 明
getInetAddress()	InetAddress	获取套接字连接的地址
getPort()	int	获取此套接字连接的端口
getLocalAddress()	InetAddress	获取套接字绑定的本地地址
getLocalPort()	int	获取套接字绑定的本地端口
close()	void	关闭套接字连接
getInputStream()	InputStream	获取套接字的输入流
getOutputStream()	OutputStream	获取套接字的输出流

当套接字建立成功后，可以想象一条通信的“线路”已经建立起来，套接字可以使用 getInputStream() 方法获得一个输入流，然后用输入流读取线路信息，还可以通过套接字的 getOutputStream()方法获取一个输出流对象，将信息写入“线路”中。

例 17.04 客户端向服务器发送信息，服务器读取客户端发送的信息，并将读取的数据写入到输出流中。实例代码如下：**（实例位置：光盘\mr\ym\17.04）**

（1）服务器端程序。

```
public class Server {
    public static void main(String[] args) {
        ServerSocket serverSocket = null;   // 创建服务器端套接字
        Socket clientSocket = null;         // 创建客户端套接字
        String str = null;
        DataOutputStream out = null;        // 创建 DataOutputStream 类对象
        DataInputStream in = null;          // 创建 DataInputStream 类对象
        try {
            serverSocket = new ServerSocket(4331);                          // 实例化 ServerSocket 对象
            clientSocket = serverSocket.accept();                           // 接收客户的套接字连接呼叫
            in = new DataInputStream(clientSocket.getInputStream());        // 实例化 DataInputStream 对象
            out = new DataOutputStream(clientSocket.getOutputStream()); // 实例化 DataOutputStream 对象
            while (true) {
                str = in.readUTF();         // 读取客户放入连接中的信息
                out.writeUTF("hello，我是服务器");                           // 通过输出流向线路中写信息
                out.writeUTF(str);
                System.out.println("服务器收到：" + str);
                Thread.sleep(1000);         // 线程休眠
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

（2）客户端程序。

```
public class SocketUtil {
    public static void main(String[] args) {
        String str = null;
        Socket clientSocket;                // 创建客户端套接字
        DataInputStream in = null;          // 创建 DataInputStream 对象
        DataOutputStream out = null;        // 创建 DataOutputStream 对象
        try {
            clientSocket = new Socket("127.0.0.1", 4331);                   // 实例化 Socket 对象
            in = new DataInputStream(clientSocket.getInputStream());        // 实例化 DataInputStream 对象
            // 实例化 DataOutputStream 对象
out = new DataOutputStream(clientSocket.getOutputStream());
            out.writeUTF("你好！！");       // 写数据
            while (true) {
                str = in.readUTF();         // 读取流中数据
                out.writeUTF(((int) (Math.random() * 10) + 1) + "");        // 向流中写入0～10之间的随机数
                System.out.println("客户端收到：" + str);                    // 输出信息
                Thread.sleep(1000);         // 线程休眠
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

📖 **说明：IP 地址 127.0.0.1 代表的是本地 IP。**

若客户机要连接服务器，则服务器必须首先做出动作。因此，运行程序时，一定要先启动服务器端程序，再运行客户端程序。运行程序，客户端向服务器发送信息“你好!!”，然后每隔 1000 毫秒，客户端向服务器发送一个 0～10 之间的随机数。服务器会将客户发送的数据返回给客户端。在控制台上输出的信息如图 17.10 所示。

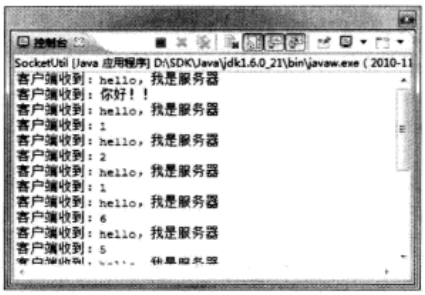


图 17.10　本实例的运行结果

17.4　UDP 通信

用户数据报（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，但有时需要较快的传输信息，允许容忍小的错误，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。

下面是 UDP 程序的步骤。

发送数据包：

（1）使用 DatagramSocket()创建一个数据包套接字。

（2）使用 DatagramPacket(byte[] buf , int offset , int length , InetAddress address , int port)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)创建数据包套接字，并将其绑定到指定的端口。

（2）使用 DatagramPacket(byte[] buf , int length)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。

17.4.1　DatagramSocket 类

DatagramSocket 类可以建立 UDP 程序的套接字，数据报套接字是包投递服务的发送或接收点。每个在

数据报套接字上发送或接收的包都是单独编址和路由的。该类常用的构造方法介绍如下。

☑　创建绑定到本地主机上任何可用的端口。该方法声明如下：

```
public DatagramSocket();
```

☑　创建绑定到本地主机上的指定端口的套接字。该方法声明如下：

```
public DatagramSocket(int port)
```

参数说明：port 参数是创建套接字时指定的端口号。

☑　创建绑定的指定本地地址。

该方法适用于多块网卡和多个 IP 地址。该方法声明如下：

```
public DatagramSocket(int port , InetAddress address)
```

参数说明：port 是要使用的本地端口，address 是要绑定的本地地址。

在接收程序时，必须指定一个端口号，不要让系统随机产生。可以使用第二个构造函数。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。

可以通过该类的 send()方法发送数据报包。例如：

```
DatagramSocket dataSocket = new DatagramSocket();
dataSocket.send(packet);                    // 参数 packet 为 DatagramPacket 实例
```

可以通过该类的 receive()方法接收数据报包。例如：

```
DatagramSocket clientSocket = new DatagramSocket();
clientSocket.receive(recvPacket);           // 参数 recvPacket 为 DatagramPacket 实例
```

17.4.2　DatagramPacket 类

该类表示数据报包，数据报包用来实现无连接包投递服务。每条报文仅根据该包中包含的信息，从一台机器路由到另一台机器。从一台机器发送到另一台机器的多个包可能选择不同的路由，也可能按不同的顺序到达，不对包投递做出保证。DatagramPacket 类的构造方法介绍如下。

☑　指定接收长度，创建 DatagramPacket 实例。该方法声明如下：

```
new DatagramPacket(byte[] buf,int length);
```

参数说明：buf 是保存传入数据包的缓冲区，length 是要读取的字节数。

☑　指定了数据包的内存空间和大小，同时指定了数据包的目标地址和端口。该方法声明如下：

```
new DatagramPacket(byte[] buf,int length,InetAddress address,int port)
```

参数说明：buf 是数据报包数据，length 是数据包的长度，address 是目的地址，port 指定目的端口号。

例 17.05　服务器端接收数据，再发送数据，客户端实现发送数据，再接收数据。实例代码如下：**（实例位置：光盘\mr\ym\17.05）**

（1）服务器端程序。

```
public static void main(String[] args) {
    DatagramSocket serverSocket;
    try {
        serverSocket = new DatagramSocket(6000);          // 创建 DatagramSocket 对象
        byte[] recvBuf = new byte[100];                   // 创建字节数组
        DatagramPacket recvPacket = new DatagramPacket(recvBuf,
                recvBuf.length);                          // 创建 DatagramPacket 对象
        serverSocket.receive(recvPacket);                 // 接收数据包
        String recvStr = new String(recvPacket.getData(), 0, recvPacket
```

```
                .getLength());
        System.out.println("Hello!" + recvStr);
        int port = recvPacket.getPort();                    // 获取接收端口
        InetAddress address = recvPacket.getAddress();      // 创建 InetAddress 实例
        String sendStr = "服务器发送的信息";
        byte[] sendBuf;
        sendBuf = sendStr.getBytes();
        DatagramPacket sendPacket = new DatagramPacket(sendBuf,
                sendBuf.length, address, port);             // 创建数据包
        serverSocket.send(sendPacket);                      // 发送数据包
        serverSocket.close();                               // 关闭 DatagramSocket 对象
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

（2）客户端程序。

```
public static void main(String[] args) {
    DatagramSocket clientSocket;
    try {
        clientSocket = new DatagramSocket();                // 创建套接字对象
        String sendStr = "你好，我是客户机";
        byte[] sendBuf;
        sendBuf = sendStr.getBytes();
        InetAddress addr = InetAddress.getByName("127.0.0.1");
        int port = 6000;                                    // 指定端口
        DatagramPacket sendPacket = new DatagramPacket(sendBuf,
                sendBuf.length, addr, port);                // 创建数据包
        clientSocket.send(sendPacket);                      // 发送数据包
        byte[] recvBuf = new byte[100];
        DatagramPacket recvPacket = new DatagramPacket(recvBuf,
                recvBuf.length);                            // 创建数据包对象
        clientSocket.receive(recvPacket);                   // 接收信息
        String recvStr = new String(recvPacket.getData(), 0, recvPacket
                .getLength());
        System.out.println("收到：" + recvStr);
        clientSocket.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

运行程序，结果如图 17.11 所示。

图 17.11　本实例的运行结果

17.5　照猫画虎——基本功训练

17.5.1　基本功训练 1——获取主机 IP 地址

视频讲解：光盘\mr\lx\17\获取主机 IP 地址.exe

实例位置：光盘\mr\ym\17.06\src

在进行网络编程时，由于网络通信的需要，经常需要获取本地主机的 IP 地址，本实例将演示如何通过 Java 程序获得本地主机的 IP 地址。

实现过程如下：

（1）新建一个项目。

（2）在项目中创建一个继承自 JFrame 类的窗体类 GetLocalHostIpFrame。

（3）将 GetLocalHostIpFrame 窗体的内容面板设置为绝对布局，然后在面板上放置两个 JLabel 标签和一个 JTextField 文本框，文本框的名称为 tf_ip，再添加两个 JButton 命令按钮，分别用于获得本地主机的 IP 地址和退出系统。

（4）在“获取 IP 地址”按钮的事件中，实现获得本地主机的 IP 地址，并显示在文本框中，该按钮的代码如下：

```
button.addActionListener(new ActionListener() {
    public void actionPerformed(final ActionEvent e) {
        try {
            InetAddress inetAddr = InetAddress.getLocalHost();    // 创建本地主机的 InetAddress 对象
            String ip = inetAddr.getHostAddress();                // 获得本地主机的 IP 地址
            tf_ip.setText(ip);                                    // 在文本框中显示 IP 地址
        } catch (UnknownHostException e1) {
            e1.printStackTrace();
        }
    }
});
```

运行程序，单击窗体上的“获取 IP 地址”按钮，将在文本框中显示本地主机的 IP 地址，效果如图 17.12 所示。

图 17.12　获取本地主机的 IP 地址

照猫画虎：编程接收用户在控制台输入的计算机名称，然后解析出它的 IP 地址并输出。（20 分）（实例位置：光盘\mr\ym\17.06\zmhh）

17.5.2 基本功训练 2——设置连接超时

视频讲解：光盘\mr\lx\17\设置连接超时.exe

实例位置：光盘\mr\ym\17.07\src

在进行网络编程时，由于网络连接非常消耗资源，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，则进行其他的处理。下面通过实例实现连接超时的提示。

实现过程如下：

（1）新建一个项目。

（2）在项目中创建一个继承自 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。

（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，该方法的代码如下：

```
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();
    }
}
```

运行程序，等待 10 秒钟后，将弹出消息框提示连接超时，效果如图 17.13 所示。

图 17.13 设置等待连接的超时时间

你问我答：如何设置连接超时？

本实例主要是通过 ServerSocket 类的实例调用 setSoTimeout()方法设置连接的等待超时时间，从而实现设置等待连接的超时时间。方法声明如下：

public void setSoTimeout(int timeout) throws SocketException

参数说明：timeout 是以毫秒为单位指定等待连接的超时时间。使用该方法需要处理 SocketException 异常。

照猫画虎：修改程序的连接超时时间为 24 小时。（20 分）（实例位置：光盘\mr\ym\17.07\zmhh）

17.5.3　基本功训练 3——创建 URL 对象

视频讲解：光盘\mr\lx\17\创建 URL 对象.exe

实例位置：光盘\mr\ym\17.08\src

在 Java 语言中万物皆对象，甚至常用的网址也是对象，网址就是在浏览网页时区分不同网站的地址。例如，http://www.eclipse.org 是 Eclipse 的官方网站的地址，简称官网。在 Java 语言中使用 URL 类抽象了网址的所有属性与行为，它可以根据一个网址来创建对象。下面演示将网址封装为 Java 对象。

实现过程如下：

（1）创建 Java 项目，在项目中创建一个窗体类 URLFrame。

（2）在窗体的左侧、中间和右侧分别添加标签控件、文本框控件和按钮控件。其中按钮控件的 text 属性设置为“验证”。

（3）编写“验证”按钮的事件处理方法，把用户在文本框中输入的网址字符串作为 URL 类的构造方法的参数来创建对象，如果创建过程中出现程序异常，就提示错误信息。关键代码如下：

```
button.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        String addText = textField.getText();          // 获取文本框输入
        try {
            URL url=new URL(addText);                  // 创建 URL 类的对象
            JOptionPane.showMessageDialog(URLFrame.this, "URL 对象创建成功\n"+url);
        } catch (MalformedURLException e1) {           // 发生程序异常
            JOptionPane.showMessageDialog(URLFrame.this, "URL 对象创建失败\n"+e1.getMessage());
        }
    }
});
```

实例运行效果如图 17.14 所示。

图 17.14　实例运行效果

你问我答：URL 类有什么作用？

URL 类位于 java.net 包中，它代表一个统一资源定位符，是指向互联网资源的指针。资源可以是简单的文件或目录，也可以是更为复杂的网址。下面介绍创建 URL 对象的构造方法。该方法声明如下：

public URL(String spec) throws MalformedURLException

参数说明：spec 是将作为 URL 解析的字符串。

照猫画虎：把程序修改成控制台输入与输出的 URL 完整路径验证实例。（20 分）（实例位置：光盘\mr\ym\17.08\zmhh）

17.5.4 基本功训练 4——提取 URL 协议名称

视频讲解：光盘\mr\lx\17\提取 URL 协议名称.exe

实例位置：光盘\mr\ym\17.09\src

URL 是统一资源定位符，它包含很多类别，如 http、https、ftp、file 和 jar 等。本实例将提取一个 URL 中所规定的协议名称。实例代码如下：

```
import java.net.MalformedURLException;
import java.net.URL;
import java.util.Scanner;
public class URLDemo {
    public static void main(String[] args) {
        try {
            Scanner scan = new Scanner(System.in);    // 创建输入流扫描器
            System.out.println("请输入一个完整的网址：");
            String line = scan.nextLine();                    // 获取用户输入文本
            URL url = new URL(line);                          // 创建 URL 对象
            System.out.println("这个网址的主机名称是：" + url.getHost());              // 获取主机名称
            System.out.println("这个网址的 URL 协议名称是：" + url.getProtocol());     // 获取协议名称
        } catch (MalformedURLException e) {
            System.out.println("输入的是非法网址");      // 提示错误信息
        }
    }
}
```

实例运行效果如图 17.15 所示。

图 17.15 实例运行效果

你问我答：如何获取 URL 协议名称？

URL 类提供了多个方法来获取其属性，其中包括主机名称、协议名称等。本实例的核心在于解析出 URL 中的协议名称，所以要调用 getProtocol()方法。该方法的声明如下：

```
public String getProtocol()
```

例如，本实例中获取 URL 协议名称的关键代码如下：

```
URL url = new URL(line);                                          // 创建 URL 对象
System.out.println("这个网址的 URL 协议名称是：" + url.getProtocol());   // 获取协议名称
```

照猫画虎：尝试由 URL 的协议名称与主机名称属性的连接组成原有 URL 的字符串。（20 分）（实例位置：光盘\mr\ym\17.09\zmhh）

17.5.5　基本功训练 5——获取 URL 的端口

视频讲解：光盘\mr\lx\17\获取 URL 的端口.exe

实例位置：光盘\mr\ym\17.10\src

编程从控制台中接收用户输入的一个完整的 URL 网址，然后通过程序进行判断，并获取该网址使用的端口号。实例代码如下：

```
import java.net.MalformedURLException;
import java.net.URL;
import java.util.Scanner;
public class URLPort {
    public static void main(String[] args) {
        try {
            Scanner scan = new Scanner(System.in);                    // 创建输入流扫描器
            System.out.println("请输入一个完整的网址：");
            String line = scan.nextLine();                            // 获取用户输入文本
            URL url = new URL(line);                                  // 创建 URL 对象
            int port = url.getPort();
            if (port == -1)
                port = url.getDefaultPort();
            System.out.println("这个网址使用的端口号是：" + port);      // 输出端口号
        } catch (MalformedURLException e) {
            System.out.println("输入的是非法网址");                    // 提示错误信息
        }
    }
}
```

实例运行结果如图 17.16 所示。

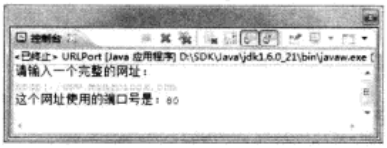


图 17.16　实例运行结果

你问我答：如何确定 URL 的端口号？

URL 类提供了两种获取端口号的方法，它们可以获取当前使用的端口号与默认的端口号，下面分别介绍这两种方法。

（1）获取当前端口号的方法

URL 可以指定不同的端口号，如 http://www.mingribook.com:8080 中的 8080 就是端口号。URL 类的 getPort()方法可以获取这个端口号。该方法声明如下：

```
public int getPort()
```

（2）获取默认端口号

当 getPort()方法返回值为-1 时，说明 URL 没有指定端口号，而是使用默认端口号，这时可以通过 getDefault

Port()方法获取。该方法声明如下：

```
public int getDefaultPort()
```

照猫画虎：编程接收用户输入的完整 URL，然后输出它的主机名与端口号（包括默认端口号）。（20 分）（实例位置：光盘\mr\ym\7.10\zmhh）

照猫画虎栏目分数统计：

照猫画虎题目	1	2	3	4	5	总分数	
分数							

17.6 情景应用——拓展与实践

17.6.1 情景应用 1——客户端一对一通信

视频讲解：光盘\mr\lx\17\客户端一对一通信.exe

实例位置：光盘\mr\ym\17.11\src

在使用套接字进行网络编程时，有时需要在不同的客户端之间进行通信，其中有一种通信方式就是一个客户端与另一个指定的客户端进行通信。本实例讲解了如何实现一个客户端与另一个指定的客户端进行通信，运行程序，服务器启动后，启动 3 个客户端程序，并分别以 aaa、bbb 和 ccc 进行登录，然后在左侧的用户列表中选择接收信息的用户，输入聊天信息即可发送到目标用户，效果如图 17.17 和图 17.18 所示。

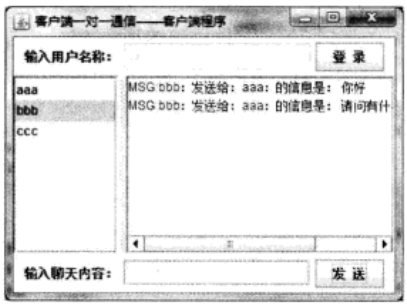


图 17.17 客户端 aaa 接收到 bbb 的信息

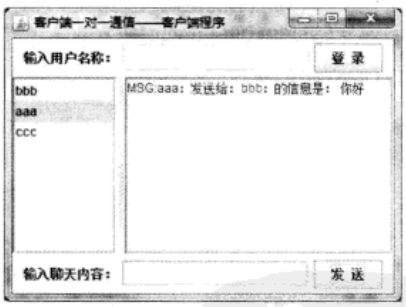


图 17.18 客户端 bbb 接收到 aaa 的信息

实现过程如下：

（1）新建一个项目。

（2）在项目中创建一个继承自 JFrame 类的服务器窗体类 ClientOneToOne_ServerFrame 和一个客户端窗体类 ClientOneToOne_ClientFrame。

（3）在服务器窗体类 ClientOneToOne_ServerFrame 中定义 ServerThread 线程类，在该线程类的 run() 方法中有一部分代码用于为客户端添加用户列表，其关键代码如下：

```
if (info.startsWith("用户：")) {                    // 添加登录用户到客户端列表
    key = info.substring(3, info.length());        // 获得用户名并作为键使用
    map.put(key, socket);                          // 添加键值对
    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
            Socket s = map.get(receiveKey);                // 获得与该键对应的套接字对象
            PrintWriter out = new PrintWriter(s
                    .getOutputStream(), true);             // 创建输出流对象
            Iterator<String> keyIt1 = set.iterator();      // 获得所有键的迭代器
            while (keyIt1.hasNext()) {
                String receiveKey1 = keyIt1.next();        // 获得键，用于向客户端添加用户列表
                out.println(receiveKey1);                  // 发送信息
                out.flush();                               // 刷新输出缓冲区
            }
        }

    } else {                                               // 转发接收到的消息
        // 这里省略了转发接收消息的代码
    }
```

（4）在服务器窗体类 ClientOneToOne_ServerFrame 的线程类 ServerThread 中，在 run()方法中有一部分代码用于转发客户端发送的消息，其关键代码如下：

```
    if (info.startsWith("用户：")) {                       // 添加登录用户到客户端列表
        // 这里省略了向客户端添加用户列表的代码
    } else {                                               // 转发接收的消息
        key = info.substring(info.indexOf("：发送给：") + 5, info
                .indexOf("：的信息是："));                 // 获得接收方的 key 值，即接收方的用户名
        String sendUser = info.substring(0, info
                .indexOf("：发送给："));                   // 获得发送方的 key 值，即发送方的用户名
        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
            if (key.equals(receiveKey)
                    && !sendUser.equals(receiveKey)) {     // 如果是发送方，但不是用户本身
                Socket s = map.get(receiveKey);            // 获得与该键对应的套接字对象
                PrintWriter out = new PrintWriter(s
                        .getOutputStream(), true);         // 创建输出流对象

                out.println("MSG:"+info);                  // 发送信息
                out.flush();                               // 刷新输出缓冲区
            }
        }
    }
```

（5）在客户端窗体类 ClientOneToOne_ClientFrame 中定义 ClientThread 线程类，用于对接收到服务器的信息进行处理，如果是登录用户就添加到用户列表中，如果是消息就追加到文本域中。该线程类的 run()方法实现该功能的关键代码如下：

```
    if (!info.startsWith("MSG:")) {
        boolean itemFlag = false;                  // 标记是否为列表框添加列表项，为 true 不添加，为 false 添加
        for (int i = 0; i < model.getSize(); i++) {
            if (info.equals((String) model.getElementAt(i))) {
                itemFlag = true;                   // 标记为 true，表示是用户
            }
```

```
        }
        if (itemFlag) {
            model.addElement(info);                    // 添加列表项
        } else {
            itemFlag = false;                          // 标记设置为 false
        }
    } else {
        ta_info.append(info + "\n");                   // 在文本域中显示信息
        if (info.equals("88")) {
            break;                                     // 结束线程
        }
    }
```

DIY：编写一个使用 Socket 实现简单通信的程序。（35 分）（实例位置：光盘\mr\ym\17.11\diy）

17.6.2 情景应用 2——编写简单聊天室程序

视频讲解：光盘\mr\lx\17\编写简单聊天室程序.exe

实例位置：光盘\mr\ym\17.12\src

在使用套接字进行网络编程时，有时需要在不同的客户端之间进行通信，其中有一种通信方式就是一个客户端与其他的多个客户端进行通信，这在聊天室中称为核心通信方式。本实例以聊天室为例讲解如何实现一个客户端与其他多个客户端进行通信，运行程序，服务器启动后，启动两个客户端程序，然后通过第一个客户端发送信息，则其他所有的客户端都会收到服务器发送的信息，效果如图 17.19 和图 17.20 所示。

图 17.19 第二个客户端接收到第一个客户端的信息

图 17.20 第三个客户端接收到第一个客户端的信息

实现过程如下：

（1）新建一个 Java 项目。

（2）在项目中创建 Server 服务器类，在该类的主方法中创建 ServerSocket 对象，并在循环中接收用户的 Socket 连接。关键代码如下：

```
static ArrayList<Socket> clients = new ArrayList<Socket>();     // Socket 集合
public static void main(String[] args) {
    try {
        ServerSocket server = new ServerSocket(8888);
        System.out.println("服务器已经启动.....");
        while (true) {
            Socket socket = server.accept();                    // 接收客户端连接
            clients.add(socket);                                // 把客户端保存到集合中
```

```
                transmitMessage(socket);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```

（3）在 Server 服务器类中编写 transmitMessage()方法，该方法用于接收客户端的输入，并转发给所有客户端。关键代码如下：

```
private static void transmitMessage(final Socket socket) throws IOException {
    new Thread() {
        private Scanner scan;
        public void run() {
            try {
                InputStream is = socket.getInputStream();    // 获取客户端 Socket 输入流
                scan = new Scanner(is);                      // 创建输入流扫描器
                // 发送连接成功信息
                socket.getOutputStream().write("连接成功...\n".getBytes());
            } catch (IOException e1) {
                e1.printStackTrace();
            }
            while (true) {
                String text = scan.nextLine() + "\n";        // 接收客户发送的信息
                for (Socket client : clients) {
                    try {
                        OutputStream out = client.getOutputStream();// 获取每个客户端的输出流
                        out.write(text.getBytes());          // 向每个客户端转发信息
                        out.flush();
                    } catch (IOException e) {
                    }
                }
            }
        }
    }.start();
}
```

（4）编写客户端窗体类的代码，在构造方法中创建 Socket 对象连接服务器，并执行 receiveMessage()方法接收服务器发送的信息。关键代码如下：

```
socket = new Socket("127.0.0.1", 8888);  // 创建连接服务器的 Socket
receiveMessage();                        // 启动接收信息的线程
```

（5）编写 receiveMessage()方法，该方法用于接收用户发送的信息，并把该信息显示到窗体的文本域控件中。关键代码如下：

```
public void receiveMessage() {                          // 接收信息的方法
    new Thread() {                                      // 创建线程

        public void run() {
            try {
                is = socket.getInputStream();
                Scanner scan = new Scanner(is);         // 创建输入流扫描器
                while (socket.isConnected()) {
```

```
                    String text = scan.nextLine() + "\n"; // 读取一行信息
                    textArea.append(text);          // 将读取的信息显示到文本域控件
                    sleep(100);                     // 线程休眠
                }
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }.start();                                      // 线程启动
}
```

（6）为“发送”按钮编写事件监听器实现信息的发送，事件监听器代码中首先获取 Socket 的输出流，然后把用户昵称与发送文本连接在一起向服务器发送。关键代码如下：

```
button.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        try {
            out = socket.getOutputStream();                 // 获取 Socket 输出流
            String name = nameField.getText() + ": ";       // 获取昵称
            String text = name + textField.getText() + "\n"; // 为信息添加结束符
            out.write(text.getBytes());                     // 发送信息到服务器端
            out.flush();
            textField.setText("");                          // 清空文本框内容
        } catch (IOException e1) {
            e1.printStackTrace();
        }
    }
});
```

DIY：修改本实例，在昵称后添加客户的 IP 地址。（35 分）（实例位置：光盘\mr\ym\17.12\diy）

17.6.3 情景应用 3——使用 Socket 传输图片

视频讲解：光盘\mr\lx\17\使用 Socket 传输图片.exe

实例位置：光盘\mr\ym\17.13\src

在使用套接字进行网络编程时，有时需要通过 Socket 传输图片，本实例讲解了如何通过套接字传输图片，运行程序，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，效果如图 17.21 和图 17.22 所示。也可以在客户端选择图片，单击“发送”按钮，向服务器端发送图片。

图 17.21 在服务器端选择图片并发送

图 17.22 在客户端显示接收到的图片

实现过程如下：

（1）新建一个项目。

（2）在项目中创建一个继承自 JFrame 类的服务器窗体类 ServerSocketFrame 和一个客户端窗体类 ClientSocketFrame，并完成服务器和客户端窗体界面的设计。

（3）在服务器窗体类 ServerSocketFrame 中，定义 getServer()方法用于创建服务器端套接字、监听客户端程序，以及创建向客户端发送信息的输出流对象和创建用于接收客户端发送信息的输入流对象，该方法的关键代码如下：

```
server = new ServerSocket(1978);                                  // 实例化 Socket 对象
while (true) {                                                    // 如果套接字是连接状态
    socket = server.accept();                                     // 实例化 Socket 对象
    out = new DataOutputStream(socket.getOutputStream());         // 获得输出流对象
    in = new DataInputStream(socket.getInputStream());            // 获得输入流对象
    getClientInfo();                                              // 调用 getClientInfo()方法
}
```

（4）在服务器窗体类 ServerSocketFrame 中，定义 getClientInfo()方法用于接收客户端发送的图片，该方法的关键代码如下：

```
long lengths = in.readLong();                                     // 读取图片文件的长度
byte[] bt = new byte[(int) lengths];                              // 创建字节数组
for (int i = 0; i < bt.length; i++) {
    bt[i] = in.readByte();                                        // 读取字节信息并存储到字节数组
}
receiveImg = new ImageIcon(bt).getImage();                        // 创建图像对象
receiveImagePanel.repaint();                                      // 重新绘制图像
```

（5）在客户端窗体类 ClientSocketFrame 中，“发送”按钮的事件用于向服务器传输图片。“发送”按钮的事件代码如下：

```
DataInputStream inStream = null;                                  // 定义数据输入流对象
if (imgFile != null) {
    lengths = imgFile.length();                                   // 获得选择图片的大小
    inStream = new DataInputStream(new FileInputStream(imgFile)); // 创建输入流对象
} else {
    JOptionPane.showMessageDialog(null, "还没有选择图片文件。");
    return;
}
out.writeLong(lengths);                                           // 将文件的大小写入输出流
byte[] bt = new byte[(int) lengths];                              // 创建字节数组
int len = -1;
while ((len = inStream.read(bt)) != -1) {                         // 将图片文件读取到字节数组
    out.write(bt);                                                // 将字节数组写入输出流
}
```

DIY：编程实现使用 Socket 传送音频文件。（30 分）（实例位置：光盘\mr\ym\17.13\diy）

情景应用 DIY 栏目分数统计：

DIY 题目	1	2	3	总分数	
分数					

17.7 自 我 测 试

一、选择题（每题 10 分，5 道题）

1．端口号的取值范围是？（　　）

A．0～32768　　B．0～65535　　C．0～256　　D．任何整数

2．InetAddress 类属于哪个 Java 包？（　　）

A．java.util　　B．java.net　　C．java.text　　D．javax.net

3．Java 的套接字服务器类是？（　　）

A．ServerSocket　　B．SocketServer　　C．Socket　　D．Server;

4．TCP 与 UDP 哪个传输数据比较安全？（　　）

A．TCP　　B．两个都安全　　C．UDP　　D．两个都不安全

5．获取 URL 默认端口号的方法是？（　　）

A．getDefaultPort()　　B．getPort()　　C．toString()　　D．getHostPort()

二、填空题（侧重实用技能，每题 10 分，5 道题）

1．创建 Socket 对象时，要捕获的异常包括_____和_____。

2．创建 URL 类的对象时，构造方法可能报出什么异常？_____。

3．ServerSocket 类获取客户端连接的方法是_____。

4．设置 Socket 连接超时的方法是_____。

5．URL 类的_____方法可以获取协议名称。

测试分数统计：

类别	第 1 题	第 2 题	第 3 题	第 4 题	第 5 题
选择题分数					
填空题分数					
总分数					

17.8 行 动 指 南

开始日期：_____年____月____日　　结束日期：_____年____月____日

序号	内　容	行 动 指 南	
1	照猫画虎栏目	分数>75 分	优秀，基本功掌握得不错，加油！
		75 分>分数>50 分	及格，知识掌握得不牢，重新做一遍照猫画虎。
	分数（　　）	分数<50 分	请使用光盘中提供的“实战能力测试系统”进行提高训练。
	情景应用栏目	分数>75 分	优秀，综合应用能力很强。
		75 分>分数>50 分	及格，综合应用能力需提高，再练一遍情景应用。
	分数（　　）	分数<50 分	请使用光盘中提供的“实战能力测试系统”进行提高训练。

续表

序号	内　容	行 动 指 南	
1	自我测试栏目 分数（　　）	分数>75 分	优秀，有成为编程高手的潜质。
		分数<75 分	请使用光盘中提供的“实战能力测试系统”进行提高训练。
	综合评价	反复训练后，以上各项分数都在 75 分以上，方可进入下一堂课学习。	
2	编程能力培养：本栏目提供了一些实践题目，对于培养编程能力很有效，要做好。	（1）编程获取百度的 IP 地址。	
		（2）创建网络程序，要求在客户端通过窗体接收用户输入的个人信息，然后在服务器端把这些信息保存到数据库。	
		（3）编写一个文件网络传输程序，要求在服务器端传输指定的文件（不需要 GUI 界面），然后客户端在窗体中以进度条形式显示文件接收进度，当文件接收完毕后显示文件选择对话框，让用户选择保存路径。	
		（4）实现网络遥控程序（提示：客户端在窗体界面中单击指定的按钮，服务器端执行相应的命令，可以使用提示对话框显示执行结果）。	
3	编程习惯培养：本堂课培养读者在学习开发中记笔记的习惯，把开发中遇到的问题、总结的经验记录下来。		
4	创新能力培养：看看身边有没有可以用编程解决的问题，记录到右边的表格中。根据学习进度，尝试编写解决这些问题的小程序。		

17.9　成功可以复制——IT“大王”王志东

1967 年，王志东出生在东莞虎门镇的一个教师家庭，王志东的童年在半饥半饱的状态下度过，清苦的家况成为他日后学习和创业的初始动力。在大学的时候王志东对计算机产生了浓厚的兴趣，大三时，在中关村，王志东给皮包公司做过推销，也做过汉化、二次开发、系统集成等技术活，为以后的软件开发奠定了基础。1991 年 6 月，王志东独立完成了当时国内第一个实用化 Windows 3.0 中文环境——BDWin 3.0，成为北大方正 1991 年的七大成果之一。但是当时王志东所在的公司并没有推广这个项目，这使王志东很伤心，并且辞职。但是他没有放弃，并于 1992 年 4 月成立了自己的公司——“新天地电子信息技术研究所”。想要按照硅谷的方式进行发展，实现自己的硅谷梦。1992 年 5 月，王志东独立研制并推出全球第一套实用 Windows 3.1 中文平台“中文之星（Chinese Star 1.1）”。“中文之星”一经推出即在国内迅速普及，加速了我国的电脑应用，创造了很好的社会效益和经济效益。但由于同合伙人意见不合，王志东黯然离开新天地，他的硅谷梦也化为泡影。王志东深感挫折，但是他的才能没有被埋没，1993 年 12 月，王志东出任新组建的“四通利方信息技术有限公司”总经理，时年 26 岁。主持开发的 RichWin 系列软件于 1995 年下半年推出，成为

RichWin

当时世界上水平最高的外挂式中文平台系统软件，迅速占领市场。1998 年 12 月，四通利方宣布成功并购位于“硅谷”的华渊生活资讯网，成立新浪网，王志东出任总裁，1 年后兼任 CEO。2001 年 6 月，由于意见分歧王志东被解除新浪网的首席执行官和董事职务。但是王志东平静地看待这一切，于当年年底，他创立了北京点击科技有限公司，重新走上创业之路。

在王志东亲自带领下，2002 年年底，点击科技推出了国内第一个协同应用平台——竞开协同应用平台（GENEKING），接着又推出了基于“竞开协同应用平台”的第一个产品——“竞开协同之星”，紧接着又推出一系列竞开系列软件。竞开（Geneking）系列协同软件被计世资讯、赛迪顾问等多家权威机构评为“中国协同软件市场领导型产品”和“中国协同软件市场第一品牌”，并被中国软件行业协会评为“优秀软件产品”。

经典语录

创业的过程就是实现梦想的过程，所以我是追着我自己的梦而走。

深度评价

王志东的成功之路有过辉煌到过低谷，他追逐着自己的梦想一路走来，创造了一段传奇人生。正如他所说“年轻人就应该有梦想有追求”，失败了没有关系，因为“青春正如一张白纸”。

第18堂课

表格控件的应用

（视频讲解：210 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。本堂课将学习 Swing 表格的使用方法，在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实例。

学习摘要：

- Swing 表格的创建方法
- 定制和操纵表格的常用方法
- 维护表格模型的常用方法
- JTable 和 DefaultTableModel 类的主要功能
- 提供行标题栏表格的开发思路
- Swing 表格的设计思路
- 模型的事件监听和处理

18.1 创建和定制表格

表格是最常用的数据统计组件之一，是由多行和多列组成的二维表形式。在 Swing 中由 JTable 类实现表格，本节将学习利用 JTable 类创建和定制表格。

18.1.1 创建表格

javax.swing.JTable 类创建的对象是一个表格，可以使用以下两种方法创建表格。

1. 使用列名数组和数据数组创建表格

使用 JTable 类的如下构造方法可以根据指定的列名数组和数据数组创建表格。该方法声明如下：

```
JTable(Object[][] rowData, Object[] columnNames)
```

参数说明：rowData 是封装表格数据的数组，columnNames 是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，实现表格的显示。例如：

```
final JScrollPane scrollPane = new JScrollPane();
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
table = new JTable(tableValues, columnNames);           // 创建表格
scrollPane.setViewportView(table);                      // 把表格添加到滚动面板
```

这段代码创建了一个滚动面板，又创建了表格的列名数组和数据数组，然后使用列名数组和数据数组创建了表格，并把表格添加到滚动面板上。

📖 **说明：上面代码只是说明了如何创建表格、如何将表格放到滚动面板中，并没有把滚动面板放到其他容器中。**

请读者看下面的代码，在这个实例中演示了使用列名数组和数据数组创建表格的方法。

例 18.01 本例利用构造方法 JTable(Object[][] rowData, Object[] columnNames)创建一个表格，并将表格添加到滚动面板中。实例的关键代码如下：**（实例位置：光盘\mr\ym\18.01）**

```
public JTableDemo() {
    setTitle("使用数组创建表格");                                    // 设置窗体的标题
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);                 // 设置窗体退出时操作
    setBounds(100, 100, 250, 150);                                  // 设置窗体位置和大小
    contentPane = new JPanel();                                     // 创建内容面板
    contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));             // 设置面板的边框
    contentPane.setLayout(new BorderLayout(0, 0));                  // 使用边界布局
    setContentPane(contentPane);                                    // 应用内容面板
    JScrollPane scrollPane = new JScrollPane();                     // 创建滚动面板
    contentPane.add(scrollPane, BorderLayout.CENTER);               // 应用滚动面板
    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
    table = new JTable(tableValues, columnNames);                   // 创建表格
```

```
    scrollPane.setViewportView(table);
}
```

实例运行效果如图 18.1 所示。

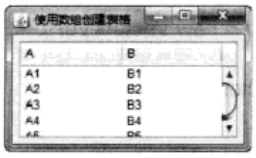


图 18.1　实例运行效果

2．使用标题数组和数据数组创建表格

在 JTable 类中还提供了如下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。该方法声明如下：

```
JTable(Vector rowData, Vector columnNames)
```

参数说明：rowData 是封装表格数据的向量，columnNames 是封装表格列名的向量。

例如：

```
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);                                  // 添加表格行向量
}
JTable table = new JTable(tableValueV, columnNameV);        // 创建指定表格列名和表格数据的表格
final JScrollPane scrollPane = new JScrollPane();           // 创建滚动面板
scrollPane.setViewportView(table);                          // 把表格添加到滚动面板中
```

这段代码创建了一个滚动面板，又创建了表格的列名向量和数据向量，然后使用列名向量和数据向量创建了表格，并把表格添加到滚动面板中。

📖 **说明：上面代码只是说明了如何使用向量创建表格、如何将表格放到滚动面板中，并没有把滚动面板放到其他容器中。**

请读者看下面的代码，在这个实例中演示了使用列名向量和数据向量创建表格的方法。

例 18.02　本例利用构造方法 JTable(Vector rowData, Vector columnNames)创建一个表格，并将表格添加到滚动面板中。实例的关键代码如下：**（实例位置：光盘\mr\ym\18.02）**

```
public JTableDemo() {
    setTitle("使用向量创建表格");                                  // 设置窗体的标题
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);             // 设置窗体退出时操作
    setBounds(100, 100, 250, 150);                              // 设置窗体位置和大小
    contentPane = new JPanel();                                 // 创建内容面板
    contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));         // 设置面板的边框
    contentPane.setLayout(new BorderLayout(0, 0));              // 使用边界布局
    setContentPane(contentPane);                                // 应用内容面板
    JScrollPane scrollPane = new JScrollPane();                 // 创建滚动面板
```

```
    contentPane.add(scrollPane, BorderLayout.CENTER);                    // 应用滚动面板
    Vector<String> columnNameV = new Vector<String>();                   // 定义表格列名向量
    columnNameV.add("A");                                                // 添加列名
    columnNameV.add("B");                                                // 添加列名
    Vector<Vector<String>> tableValueV = new Vector<Vector<String>>();   // 定义表格数据向量
    for (int row = 1; row < 6; row++) {
        Vector<String> rowV = new Vector<String>();                      // 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
    }
    table = new JTable(tableValueV, columnNameV);                        // 创建指定列名和数据的表格
    scrollPane.setViewportView(table);
}
```

实例运行效果如图 18.2 所示。

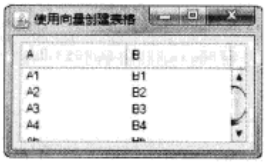


图 18.2　实例运行效果

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，如果不需要提供该功能，可以通过重写 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。例如：

```
public class MTable extends JTable {                                    // 实现自己的表格类
    // 重写表格类的 isCellEditable(int row,int column)方法
    public boolean isCellEditable(int row, int column) {                // 表格不可编辑
        return false;                                                   // 返回 false
    }
}
```

这段代码创建了一个表格类的子类，并重写了表格类的 isCellEditable(int row,int column)方法，将其返回值设置为 false，表示表格单元格不可编辑。

如果表格只有几列，通常不需要表格列的可重新排列功能；如果表格的列较多，就可以把一些重要的列移动到表格前几列的位置，这样能保证对表格中重要信息的浏览。表格列的移动可以通过 javax.swing.table.JTableHeader 类的 setReorderingAllowed(boolean reorderingAllowed)方法来实现，该方法设置表格是否支持重新排列功能，设置为 false 表示不支持重新排列功能，设置为 true 表示支持重新排列功能。通过 JTable 类的 getTableHeader()方法可以获得 JTableHeader 类的实例。例如：

```
final JScrollPane scrollPane = new JScrollPane();                       // 创建滚动面板
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
table = new JTable(tableValues, columnNames);                           // 创建表格
scrollPane.setViewportView(table);                                      // 把表格添加到滚动面板
JTableHeader tableHeader = table.getTableHeader();                      // 获得表格头的实例
tableHeader.setReorderingAllowed(false);                                // 设置列不支持重新排列
```

这段代码创建了一个滚动面板，又创建了表格的列名数组和数据数组，使用列名数组和数据数组创建了表格，并把表格添加到滚动面板上，然后获得 JTableHeader 类的实例，最后一行代码设置表格不支持列的

重新排列功能。

默认情况下单元格中的内容靠左侧显示，如果需要单元格中的内容居中显示，可以通过重写 JTable 类的 getDefaultRenderer(Class<?> columnClass)方法实现。例如：

```
public class MTable extends JTable {
    // 重写 JTable 类的 getDefautRenderer 方法
    public TableCellRenderer getDefaultRenderer(Class<?> columnClass) {
        // 获得 DefaultTableCellRenderer 对象
        DefaultTableCellRenderer cr = (DefaultTableCellRenderer) super.getDefaultRenderer(columnClass);
        // 设置单元格内容居中
        cr.setHorizontalAlignment(DefaultTableCellRenderer.CENTER);
        return cr; // 返回 DefaultTableCellRenderer 对象
    }
}
```

这段代码创建了一个表格类的子类，并重写了表格类的 getDefaultRenderer(Class<?> columnClass)方法，通过 javax.swing.table.DefaultTableCellRenderer 类的 setHorizontalAlignment(int alignment)方法将表格单元格的水平对齐方式设置为 DefaultTableCellRenderer.CENTER 字段，表示单元格内容居中显示。

DefaultTableCellRenderer 类的 setHorizontalAlignment(int alignment)方法常用的字段如表 18.1 所示。

表 18.1　用于设置表格单元格水平对齐方式的字段

字　段	说　明
DefaultTableCellRenderer.LEFT	表格单元格内容左对齐显示
DefaultTableCellRenderer.CENTER	表格单元格内容居中显示
DefaultTableCellRenderer.RIGHT	表格单元格内容右对齐显示

JTable 类创建的对象是一个表格，该类还提供了一些用来定义表格的常用方法。JTable 类中用来定义表格的常用方法如表 18.2 所示。

表 18.2　JTable 类中用来定义表格的常用方法

字　段	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

在利用表 18.2 中的 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 18.3 中列出的 javax.swing.ListSelectionModel 接口的字段中选择。

表 18.3　ListSelectionModel 接口中用于设置选择模式的字段

字　段	常　量　值	说　明
SINGLE_SELECTION	0	只允许选择一行
SINGLE_INTERVAL_SELECTION	1	允许连续选择多行
MULTIPLE_INTERVAL_SELECTION	2	可以随意选择多行

在利用表 18.2 中的 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 18.4 列出的 JTable 类的字段中选择。

📖 **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 18.4 JTable 类中用来设置自动调整模式的字段

字　段	常 量 值	说　明
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
AUTO_RESIZE_NEXT_COLUMN	1	只调整其下一列的宽度
AUTO_RESIZE_SUBSEQUENT_COLUMNS	2	按比例调整其后所有列的宽度，为默认设置
AUTO_RESIZE_LAST_COLUMN	3	只调整最后一列的宽度
AUTO_RESIZE_ALL_COLUMNS	4	按比例调整表格所有列的宽度

📖 **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。

请读者看下面的代码，在这个实例中演示了定制表格的方法。

例 18.03 本例利用本节所讲的全部知识对表格进行了定制。实例代码如下：**（实例位置：光盘\mr\ym\18.03）**

```
public class DefinedTableDemo extends JFrame {
    private static final long serialVersionUID = 2261131570553903421L;
    private JTable table;
    public static void main(String args[]) {
        DefinedTableDemo frame = new DefinedTableDemo();                    // 创建窗体
        frame.setVisible(true);                                             // 显示窗体
    }
    public DefinedTableDemo() {
        setTitle("定制表格窗体");                                            // 设置窗体标题
        setBounds(100, 100, 250, 200);                                      // 设置窗体的位置和大小
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);                     // 设置窗体的默认关闭模式
        final JScrollPane scrollPane = new JScrollPane();                   // 创建滚动面板
        getContentPane().add(scrollPane, BorderLayout.CENTER);              // 在窗体中央添加滚动面板
        // 创建表格列名数组
        Vector<String> columnNames = new Vector<String>();
        columnNames.add("A");
        columnNames.add("B");
        columnNames.add("C");
        columnNames.add("D");
        Vector<Vector<String>> rowValues = new Vector<Vector<String>>();
        for (int i = 1; i < 6; i++) {
            Vector<String> rowValue = new Vector<String>();
            rowValue.add("A" + i);
            rowValue.add("B" + i);
            rowValue.add("C" + i);
            rowValue.add("D" + i);
            rowValues.add(rowValue);
```

```
        }

        table = new MTable(rowValues, columnNames);
        // 关闭表格列的自动调整功能，这样就能产生水平滚动条了，否则只能产生垂直滚动条
        table.setAutoResizeMode(JTable.AUTO_RESIZE_OFF);
        // 选择模式设置为可以连续选择多行
        table.setSelectionMode(ListSelectionModel.SINGLE_INTERVAL_SELECTION);
        table.setSelectionBackground(Color.YELLOW);                    // 被选择行的背景色为黄色
        table.setSelectionForeground(Color.RED);                       // 被选择行的前景色为红色
        table.setRowHeight(30);                                        // 表格的行高为 30 像素
        scrollPane.setViewportView(table);                             // 把表格添加到滚动面板中
    }
    private class MTable extends JTable {
        private static final long serialVersionUID = 2213676529544788264L;
        public MTable(Vector<Vector<String>> rowData, Vector<String> columnNames) {
            super(rowData, columnNames);                               // 调用父类的构造方法
        }

        // 重写 JTable 类的 getTableHeader()方法
        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();
            hr.setHorizontalAlignment(DefaultTableCellRenderer.CENTER); // 设置列名居中显示
            return tableHeader;
        }

        // 重写 JTable 类的 getDefaultRenderer(Class<?> columnClass)方法
        public TableCellRenderer getDefaultRenderer(Class<?> columnClass) {    // 定义单元格
            DefaultTableCellRenderer cr = (DefaultTableCellRenderer) super.getDefaultRenderer(columnClass);
            cr.setHorizontalAlignment(DefaultTableCellRenderer.CENTER);        // 设置单元格内容居中
            return cr;
        }
        // 重写 JTable 类的 isCellEditable(int row, int column)方法
        public boolean isCellEditable(int row, int column) {                   // 表格不可编辑
            return false;
        }
    }
}
```

实例运行效果如图 18.3 所示。

图 18.3　实例运行效果

18.2 维护表格模型

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 创建表格模型

接口 javax.swing.table.TableModel 定义了一个表格模型，该模型提供了操作表格的方法，抽象类 javax.swing.table.AbstractTableModel 实现了 TableModel 接口的大部分方法，只有如下 3 个方法没有实现。

- ☑ public int getRowCount();。
- ☑ public int getColumnCount();。
- ☑ public Object getValueAt(int rowIndex, int columnIndex);。

通过继承 AbstractTableModel 类实现上面 3 个方法可以创建自己的表格模型类。javax.swing.table.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 类中用来定义表格的常用方法

构 造 方 法	说 明
DefaultTableModel()	创建一个零行零列的表格模型
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

18.2.2 设置表格模型

表格模型创建完成后，再通过 JTable 类的构造方法 JTable(TableModel dm)创建表格，并把表格模型设置为该构造方法的参数，这样就实现了利用表格模型创建表格。例如：

```
String[] columnNames = { "A", "B" };                                    // 定义模型的列名数组
String[][] values = { { "A1", "B1" }, { "A2", "B2" }, { "A3", "B3" }, { "A4", "B4" }, { "A5", "B5" } }; // 定义模型的数据数组
// 按照数组 values 指定的数据和数组 columnNames 指定的列名创建一个表格模型
DefaultTableModel tableModel = new DefaultTableModel(values, columnNames);
JTable table = new JTable(tableModel);                                  // 利用表格模型创建表格
```

这段代码首先定义了表格模型的列名数组和数据数组，并创建了表格模型，最后把表格模型设置为表格构造方法的参数创建表格。

从 JDK 1.6 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，javax.swing.table.TableRowSorter 类是 Swing 提供的一个排序器类，可以为表格设置排序器。例如：

```
String[] columnNames = { "A", "B" };                                    // 定义模型的列名数组
String[][] values = { { "A1", "B1" }, { "A2", "B2" }, { "A3", "B3" }, { "A4", "B4" }, { "A5", "B5" } }; // 定义模型的数据数组
// 按照数组 values 指定的数据和数组 columnNames 指定的列名创建一个表格模型
DefaultTableModel tableModel = new DefaultTableModel(values, columnNames);
```

```
JTable table = new JTable(tableModel);                                    // 创建表格
table.setRowSorter(new TableRowSorter(tableModel));                       // 设置排序器
```

请读者看下面的代码，在这个实例中演示了使用表格模型创建表格和使用表格排序器排序表格的方法。

例 18.04　利用表格模型创建一个表格，并使用表格排序器对表格进行排序。实例的关键代码如下：**（实例位置：光盘\mr\ym\18.04）**

```
public JTableDemo() {
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setBounds(100, 100, 450, 300);
    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);
    String[] columnNames = { "A", "B" };                                  // 定义表格模型的列名数组
    String[][] values = { { "A1", "B1" }, { "A2", "B2" }, { "A3", "B3" }, { "A4", "B4" }, { "A5", "B5" } };
                                                                          // 定义表格模型的数据数组
    // 按照数组 values 指定的数据和数组 columnNames 指定的列名创建一个表格模型
    DefaultTableModel tableModel = new DefaultTableModel(values, columnNames);
    table = new JTable(tableModel);                                       // 创建表格
    table.setRowSorter(new TableRowSorter<DefaultTableModel>(tableModel));   // 设置表格排序器
    scrollPane.setViewportView(table);
}
```

实例运行效果如图 18.4 所示。

单击 A 列的表头，效果如图 18.5 所示。

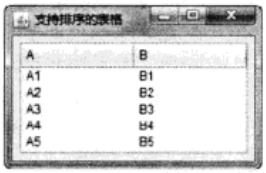


图 18.4　实例运行效果

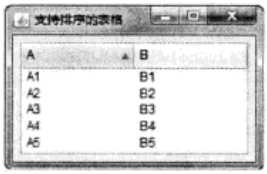


图 18.5　实例运行效果

18.2.3　维护模型对象

在使用表格时，经常需要对表格中的内容进行维护，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，它的两个重载方法介绍如下。

☑　以数组作为参数。该方法声明如下：

```
public void addRow(Object[] rowData)
```

参数说明：rowData 是要添加的行数据。

☑　以向量作为参数。该方法声明如下：

```
public void addRow(Vector rowData)
```

参数说明：rowData 是要添加的行数据。

例如：

```
String[] columnNames = { "A", "B" };                          // 定义表格模型的列名数组
String[][] values = { { "A1", "B1" }, { "A2", "B2" }, { "A3", "B3" } };   // 定义表格模型的数据数组
// 按照数组 values 指定的数据和数组 columnNames 指定的列名创建一个表格模型
DefaultTableModel tableModel = new DefaultTableModel(values, columnNames);
JTable table = new JTable(tableModel);                        // 利用表格模型创建表格
String[] rowValue = { "A4", "B4" };                           // 定义表格模型的行值数组
tableModel.addRow(rowValue);                                  // 为表格模型添加行值数组
```

这段代码首先定义了表格模型的列名数组和数据数组，并创建了表格模型，然后把表格模型设置为表格构造方法的参数创建表格，接着创建一个行值数组，并把行值数组追加到表格模型的末尾。

（2）添加到表格模型的指定位置，可以通过 insertRow()方法完成，它的两个重载方法介绍如下。

☑ 以数组作为参数。该方法声明如下：

```
public void insertRow(int row,Object[] rowData)
```

参数说明：row 是要插入的位置，rowData 是要添加的行数据。

☑ 以向量作为参数。该方法声明如下：

```
public void insertRow(int row,Vector rowData)
```

参数说明：row 是要插入的位置，rowData 是要添加的行数据。

例如：

```
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 < 4; 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);                                    // 将行向量添加到表格数据向量中
}
// 按照向量 tableValueV 指定的数据和向量 columnNameV 指定的列名创建一个表格模型
DefaultTableModel tableModel = new DefaultTableModel(tableValueV, columnNameV);
JTable table = new JTable(tableModel);                        // 通过表格模型创建表格
Vector rowValueVector = new Vector();                         // 定义行值向量
rowValueVector.add("A4");                                     // 向行值向量添加数据
rowValueVector.add("B4");                                     // 向行值向量添加数据
tableModel.insertRow(1, rowValueVector);                      // 向表格模型行索引为 1 的行添加行值向量
```

这段代码首先定义了表格模型的列名向量和数据向量，并创建了表格模型，然后把表格模型设置为表格构造方法的参数创建表格，接着创建一个行值向量，并把行值向量添加到表格模型行索引为 1 的行，即第 2 行。

注意：表格模型的行索引和列索引都是从零开始的，因此行索引值是 1 的行是表格模型中的第 2 行，当用 insertRow 方法向表格模型的某一行添加行数据后，表格模型中原来该行和该行下面的其他行会自动下移一行。

如果需要修改表格模型中某一单元格的数据，可以通过 DefaultTableModel 类的 setValueAt(Object aValue,

int row, int column)方法来完成，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 DefaultTableModel 类的 getValueAt(int row, int column)方法获得表格模型中指定单元格的值，其中 row 为单元格所在行的索引，column 为单元格所在列的索引；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 DefaultTableModel 类的 removeRow(int row)方法来完成，其中 row 为欲删除行的索引。例如：

```
String[] columnNames = { "A", "B" };                                    // 定义表格模型的列名数组
String[][] values = { { "A1", "B1" }, { "A2", "B2" },    { "A3", "B3" } };   // 定义表格模型的数据数组
// 按照数组 values 指定的数据和数组 columnNames 指定的列名创建一个表格模型
DefaultTableModel tableModel=new DefaultTableModel(values,columnNames);
JTable table=new JTable(tableModel);                                    // 利用表格模型创建表格
tableModel.setValue("A10",1,0);                                         // 修改第二行第一列的值为 A10
String s=(String)tableModel.getValue(1,0);                              // 获得表格模型第二行第一列的值
System.out.println(s);                                                  // 输出获得的表格模型第二行第一列的值
tableModel.remove(1);                                                   // 删除表格模型第二行的数据
```

这段代码首先定义了表格模型的列名数组和数据数组，并创建了表格模型，然后把表格模型设置为表格构造方法的参数创建表格，接着修改表格模型第二行第一列的值为 A10，再把获得的表格模型第二行第一列的值在控制台上输出，最后删除表格模型第二行的数据，删除该行数据后，下面的其他行会自动上移一行。

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。

请读者看下面的代码，在这个实例中演示了表格模型的维护方法。

例 18.05　利用表格模型创建一个表格，并使用表格排序器对表格进行排序。实例代码如下：**（实例位置：光盘\mr\ym\18.05）**

```
public class EditTableModelFrame extends JFrame {
    private static final long serialVersionUID = -6462158531496637367L;
    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[]) {
        try {
            UIManager.setLookAndFeel("com.sun.java.swing.plaf.nimbus.NimbusLookAndFeel");
        } catch (Throwable e) {
            e.printStackTrace();
        }
        EditTableModelFrame frame = new EditTableModelFrame();
        frame.setVisible(true);
    }
    public EditTableModelFrame() {
        super();
        setTitle("维护表格模型");
        setBounds(100, 100, 400, 200);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);                 // 设置窗体的默认关闭模式
        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<DefaultTableModel>(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());                              // 将获得的值赋值给文本框
        bTextField.setText(ob.toString());                              // 将获得的值赋值给文本框
    }
});
scrollPane.setViewportView(table);
final JPanel panel = new JPanel();
getContentPane().add(panel, BorderLayout.SOUTH);                        // 把面板添加到窗体下面
panel.add(new JLabel("A："));
aTextField = new JTextField("A4", 5);
panel.add(aTextField);
panel.add(new JLabel("B："));
bTextField = new JTextField("B4", 5);
panel.add(bTextField);
final JButton addButton = new JButton("添加");
addButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        String[] rowValues = { aTextField.getText(), bTextField.getText() };// 创建表格行数组
        tableModel.addRow(rowValues);                                   // 向表格模型中添加一行
        int rowCount = table.getRowCount() + 1;                         // 把表格的总行数加 1
        aTextField.setText("A" + rowCount);          // 文本框设置值为 A 连接总行数加 1 的值
        bTextField.setText("B" + rowCount);          // 文本框设置值为 B 连接总行数加 1 的值
    }
});
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
            // 修改表格模型当中的指定行
            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);
    }
}
```

实例运行效果如图 18.6 所示。

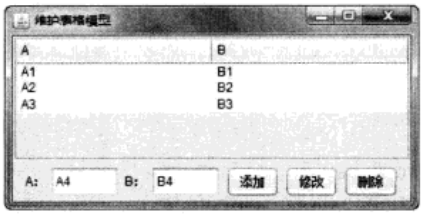


图 18.6　实例运行效果

18.3　创建行标题栏

通过 JTable 类创建的表格的列标题栏是永远可见的，即使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 18.7 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 18.8 所示的效果，即不知道每一行的具体销售日期，但是针对表格的列则不会出现这样的问题。

图 18.7　表格左侧的一列为表格的基本数据

图 18.8　移动滚动条查看未显示出的列数据

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，使其永远可见，就解决了上面的问题，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。

请读者看下面的代码，在这个实例中讲解了带行标题栏表格的创建方法。

例 18.06　本例实现了一个提供行标题栏的表格，运行本例后，移动水平滚动条，表格最左侧的列仍然可见。实现本例的基本步骤如下：**（实例位置：光盘\mr\ym\18.06）**

（1）创建 MFixedColumnTable 类，该类继承了 JPanel 类，并声明 3 个属性，具体代码如下：

```
public class MFixedColumnTable extends JPanel {
    private Vector<String> columnNameV;                          // 表格的列名数组
    private Vector<Vector<Object>> tableValueV;                  // 表格的数据数组
    private int fixedColumn = 1;                                 // 固定列的数量
}
```

（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 AbstractTableModel 类的 3 个抽象方法外，还需要重构 getColumnName(int columnIndex)方法，具体代码如下：

```
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）创建用于右侧移动列表格的模型类 FloatingColumnTableModel，该类继承了 AbstractTableModel 类，并且为 MFixedColumnTable 类的内部类。FloatingColumnTableModel 类除了需要实现 AbstractTableModel 类的 3 个抽象方法外，还需要重构 getColumnName(int columnIndex)方法，具体代码如下：

```
private class FloatingColumnTableModel extends AbstractTableModel {
    public int getColumnCount() {                                // 返回可移动列的数量
        return columnNameV.size() - fixedColumn;                 // 返回去掉固定列后的数量
    }
    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
        return tableValueV.get(rowIndex).get(columnIndex + fixedColumn); // 为列索引加上固定列的数量
    }
    public String getColumnName(int columnIndex) {               // 返回指定列的名称
        return columnNameV.get(columnIndex + fixedColumn);       // 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。

（4）在 MFixedColumnTable 类中再声明如下 4 个属性：

```
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移动列表格中的对应行；同样，当选中右侧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，具体代码如下：

```
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;                   // 为成员变量赋值
    }
    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
            floatingColumnTable.setRowSelectionInterval(row, row);// 同时选中右侧可移动列表格中的相应行
        } 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
            fixedColumnTable.setRowSelectionInterval(row, row); // 同时选中左侧固定列表格中的相应行
        }
    }
}
```

注意：这里实现的事件监听器要求两个表格必须均是单选模式的，即一次只允许选中一行。

（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、移动列表格和滚动面板，具体代码如下：

```
public MFixedColumnTable(Vector columnNameV, Vector 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());
    JViewport viewport = new JViewport();
    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);
    scrollPane.setViewportView(floatingColumnTable);
```

```
        add(scrollPane, BorderLayout.CENTER);
    }
```

（7）创建 RowTitleTableFrame 类，该类继承自 JFrame 类成为窗体，编写测试带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央。该类的完整代码如下：

```
public class RowTitleTableFrame extends JFrame {
    private static final long serialVersionUID = -8272741793330300492L;
    public RowTitleTableFrame() {
        super();
        setTitle("带行标题栏的表格");                                    // 设置窗体的标题
        setBounds(100, 100, 400, 230);                                   // 设置窗体的位置和大小
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);                  // 设置窗体的默认关闭模式
        Vector<String> columnNameV = new Vector<String>();               // 创建列名向量
        columnNameV.add("日期");
        for (int i = 1; i < 21; i++) {
            columnNameV.add("商品" + i);
        }
        Vector<Vector<Object>> tableValueV = new Vector<Vector<Object>>(); // 创建数据向量
        for (int row = 1; row < 31; row++) {
            Vector<Object> rowV = new Vector<Object>();                  // 创建行向量
            rowV.add(row);
            for (int col = 0; col < 20; col++) {
                rowV.add((int) (Math.random() * 1000));                  // 向行向量添加随机整数
            }
            tableValueV.add(rowV);                                       // 把行向量添加到数据向量
        }
        // 创建面板，在该面板中实现带行标题栏的表格
        final MFixedColumnTable panel = new MFixedColumnTable(columnNameV, tableValueV, 1);
        getContentPane().add(panel, BorderLayout.CENTER);                // 把面板添加到窗体中央

    }
    public static void main(String[] args) {
        RowTitleTableFrame frame = new RowTitleTableFrame();             // 创建窗体
        frame.setVisible(true);                                          // 显示窗体
    }
}
```

运行本例后将得到如图 18.9 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 18.10 所示的效果，表格最左侧的“日期”列仍然可见。

图 18.9　实例运行效果

图 18.10　移动水平流动条后的效果

18.4　模型的事件监听与处理

在学习模型的事件监听与处理之前，先回顾一下前面章节的内容，由于接口 TableModel 定义了一个表格模型，该模型提供了操作表格的方法，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。该方法声明分别如下：

```
public int getRowCount();
public int getColumnCount();
public Object getValueAt(int rowIndex, int columnIndex);
```

参数说明：rowIndex 是元素的行坐标，columnIndex 是元素的列坐标。

因此，通过继承 AbstractTableModel 类实现上面 3 个方法可以创建自己的表格模型类。Swing 提供的 DefaultTableModel 类继承了 AbstractTableModel 类，并实现了上面 3 个方法的表格模型类。

无论是自定义的继承自抽象类 AbstractTableModel 的表格模型类，还是 Swing 提供的 DefaultTableModel 类，都可以使用如下方法为表格模型添加表格模型事件监听器，并进行事件处理操作。该方法声明如下：

```
public void addTableModelListener(TableModelListener l)
```

参数说明：l 是表格模型监听器。

该方法每当数据模型发生变化时，如对表格模型的单元格进行了更新、为表格模型添加了数据行、从表格模型中删除了数据行等操作，就会触发表格模型的事件监听器，并执行如下方法，该方法声明如下：

```
public void tableChanged(TableModelEvent e)
```

参数说明：e 是表格模型事件。

该方法有一个 TableModelEvent 类型的入口参数。TableModelEvent 类提供的方法可以对表格模型事件进行处理。TableModelEvent 类的方法如表 18.6 所示。

表 18.6　TableModelEvent 类的方法

方　法	说　明
getColumn()	返回表格模型中事件的列
getFirstRow()	返回表格模型中第一个被更改的行
getLastRow()	返回表格模型中最后一个被更改的行
getType()	返回事件类型，该类型为 INSERT、UPDATE 和 DELETE 之一

TableModelEvent 类的 getType()方法可以判断事件的类型，通过 TableModelEvent 类的字段可以实现对事件类型的判断。TableModelEvent 类的常用字段如表 18.7 所示。

表 18.7　TableModelEvent 类的常用字段

字　段	说　明
TableModelEvent.DELETE	表示对表格模型中的行或列进行了删除操作
TableModelEvent.INSERT	表示在表格模型中添加了新行或新列
TableModelEvent.UPDATE	表示对表格模型中的现有数据进行了修改操作

例如：

```
String[] columnNames = { "A", "B" };                          // 定义表格模型的列名数组
String[][] tableValues = { { "A1", "B1" }, { "A2", "B2" }, { "A3", "B3" } };   // 定义表格模型的数据数组
```

```
// 按照数组 tableValues 指定的数据和数组 columnNames 指定的列名创建一个表格模型
DefaultTableModel tableModel = new DefaultTableModel(tableValues, columnNames);
tableModel.addTableModelListener(new TableModelListener() {      // 通过匿名类添加表格模型监听器
    public void tableChanged(TableModelEvent e) {                // 表格模型变化时对模型事件进行处理
        if (e.getType() == TableModelEvent.INSERT) {             // 是否在表格模型中添加了新行或新列
            System.out.println("数据模型发生了变化");              // 如果添加了新行或新列就输出信息
        }
    }
});
```

这段代码定义了表格模型的列名数组和数据数组，并创建了表格模型，然后为表格模型添加表格模型事件监听器。如果把下面两行语句放在按钮的动作事件或其他组件的事件中，当单击按钮时，就向表格模型的末尾追加一行数据，导致表格模型数据发生变化，因而就会触发表格模型事件监听器实现事件处理操作，由于事件类型是向表格模型中添加了新行，因此在控制台上输出信息。

```
String[] rowValue = { "A4", "B4" };                  // 定义表格模型的行值数组
tableModel.addRow(rowValue);                         // 为表格模型添加行值数组
```

除了数据模型发生变化会触发表格模型事件监听器以外，还可以通过 AbstractTableModel 类提供的方法来触发表格模型事件监听器。AbstractTableModel 类用于触发表格模型事件监听器的常用方法如表 18.8 所示。

表 18.8　触发表格模型事件监听器的常用方法

方　法	说　明
firstTableCellUpdated(int row,int column)	通知所有监听器，已更新[row, column]处的单元格值
firstTableRowsDeleted(int firstRow,int lastRow)	通知所有侦听器，已删除范围在[firstRow, lastRow]的行
firstTableRowsInserted(int firstRow,int lastRow)	通知所有侦听器，已插入范围在[firstRow, lastRow]的行
firstTableRowsUpdated(int firstRow,int lastRow)	通知所有侦听器，已更新范围在[firstRow, lastRow]的行

例如：

```
String[] columnNames = { "A", "B" };                                    // 定义表格模型的列名数组
String[][] tableValues = { { "A1", "B1" }, { "A2", "B2" }, { "A3", "B3" } };   // 定义表格模型的数据数组
// 按照数组 tableValues 指定的数据和数组 columnNames 指定的列名创建一个表格模型
DefaultTableModel tableModel = new DefaultTableModel(tableValues, columnNames);
tableModel.addTableModelListener(new TableModelListener() {             // 添加模型监听器
        public void tableChanged(TableModelEvent e) {                   // 表格模型变化时执行此方法
            System.out.println("数据模型发生了变化");                     // 在控制台上输出信息
        }
    });
```

这段代码定义了表格模型的列名数组和数据数组，并创建了表格模型，然后为表格模型添加表格模型事件监听器。如果把下面任何一条语句放在按钮的动作事件或其他组件的事件中，当单击按钮时，就会触发表格模型事件监听器对事件进行处理，并在控制台上输出信息。

```
tableModel.fireTableCellUpdated(0, 1);      // 通知所有监听器行索引为 0、列索引为 1 的单元格被修改了
tableModel.fireTableRowsInserted(1, 1);     // 通知所有监听器在行索引为 1 的位置添加了数据行
tableModel.fireTableRowsDeleted(1, 1);      // 通知所有监听器行索引从 1 到 1 的行被删除了
tableModel.fireTableRowsUpdated(1, 2);      // 通知所有监听器行索引从 1 到 2 的行进行了修改
```

这段代码中的任何一条都能触发表格模型事件监听器，如果只希望执行某种事件类型的操作，则以条件语句和事件类型进行处理。例如：

```
String[] columnNames = { "A", "B" };                                    // 定义表格模型的列名数组
String[][] tableValues = { { "A1", "B1" }, { "A2", "B2" }, { "A3", "B3" } };   // 定义表格模型的数据数组
```

```
// 按照数组 tableValues 指定的数据和数组 columnNames 指定的列名创建一个表格模型
DefaultTableModel tableModel = new DefaultTableModel(tableValues, columnNames);
tableModel.addTableModelListener(new TableModelListener() {          // 通过匿名类添加表格模型监听器
        public void tableChanged(TableModelEvent e) {                // 表格模型变化时执行此方法
            if (e.getType() == TableModelEvent.INSERT) {             // 是否添加了新行或新列
                System.out.println("数据模型发生了变化");              // 输出信息
            }
        }
    });
```

这段代码定义了表格模型的列名数组和数据数组，并创建了表格模型，然后为表格模型添加表格模型事件监听器。如果把下面的语句分别放在按钮的动作事件中，当单击按钮时，就会触发表格模型事件监听器对事件进行处理，但是只有下面的第一条语句放在按钮中，单击按钮才能在控制台上输出信息，这是由于在上面的代码中用条件语句对事件类型进行了判断。

```
tableModel.fireTableRowsInserted(1, 1);        // 通知所有监听器在行索引为 1 的位置添加了数据行
tableModel.fireTableCellUpdated(0, 1);         // 通知所有监听器行索引为 0、列索引为 1 的单元格被修改了
tableModel.fireTableRowsDeleted(1, 1);         // 通知所有监听器行索引从 1 到 1 的行被删除了
tableModel.fireTableRowsUpdated(1, 2);         // 通知所有监听器行索引从 1 到 2 的行进行了修改
```

注意：当使用 AbstractTableModel 类的方法触发表格模型事件监听器时，并不是真的对表格模型进行了添加、修改和删除等操作，只是通过这些方法来通知表格模型的所有监听器，并对事件进行响应处理，在事件响应代码中可以对事件类型进行判断，使各事件类型分别实现不同的功能。

请读者看下面的代码，在这个实例中演示了表格模型事件监听器的使用。

例 18.07　本例演示了表格模型事件监听器的使用。实例的关键代码如下：**（实例位置：光盘\mr\ym\18.07）**

```
public class TableModelEventFrame extends JFrame {
    private static final long serialVersionUID = -1439361565492730724L;
    private DefaultTableModel tableModel;                                    // 定义表格模型对象
    private JTable table;                                                    // 定义表格对象
    public TableModelEventFrame() {
        super();
        setTitle("表格模型事件");                                             // 设置窗体标题
        setBounds(100, 100, 300, 160);                                       // 设置窗体的位置和大小
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);                      // 设置窗体的默认关闭模式
        final JScrollPane scrollPane = new JScrollPane();                    // 创建滚动面板
        getContentPane().add(scrollPane, BorderLayout.CENTER);               // 在窗体中央添加滚动面板
        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
        tableModel.addTableModelListener(new TableModelListener() {          // 为表格添加模型事件监听器
                public void tableChanged(TableModelEvent e) {                // 表格内容改变执行模型事件
                                                                             // 判断是否在表格模型中添加了新行或新列
            if (e.getType() == TableModelEvent.INSERT) {
                                                                             // 如果添加了新行或新列，则输出信息
                System.out.println("你单击的是“添加”按钮");
            }
                                                                             // 判断是否对表格模型中的行或列进行了编辑修改
            if (e.getType() == TableModelEvent.UPDATE) {
```

```
                                        // 如果对表格模型中的行或列进行了编辑修改，则输出信息
                    System.out.println("你单击的是“编辑”按钮或“修改”按钮");
                }
                                        // 判断是否删除了表格模型中的行或列
                if (e.getType() == TableModelEvent.DELETE) {
                                        // 如果删除了表格模型中的行或列，则输出信息
                    System.out.println("你单击的是“删除”按钮");
                }
            }
        });
table = new JTable(tableModel);                                     // 创建指定表格模型的表格
table.setRowSorter(new TableRowSorter<DefaultTableModel>(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
scrollPane.setViewportView(table);                                  // 将表格添加到滚动面板视图
final JPanel panel = new JPanel();                                  // 创建面板
getContentPane().add(panel, BorderLayout.SOUTH);                    // 把面板添加到窗体下面
final JButton editButton = new JButton("编辑");                     // 创建“编辑”按钮
editButton.addActionListener(new ActionListener() {                 // 为“编辑”按钮添加动作监听器
            public void actionPerformed(ActionEvent e) {            // 单击按钮执行动作事件
                int selectedRow = table.getSelectedRow();           // 获得被选中行的索引
                int selectedColumn = table.getSelectedColumn();     // 获得选中列的索引
                if (selectedRow != -1 && selectedColumn != -1) {    // 判断是否存在被选中行
                                        // 通知所有监听器对选择的单元格进行了编辑操作
                    tableModel.fireTableCellUpdated(selectedRow, selectedColumn);
                }
            }
        });
panel.add(editButton);                                              // 把“编辑”按钮添加到面板上
final JButton addButton = new JButton("添加");                      // 创建“添加”按钮
addButton.addActionListener(new ActionListener() {                  // 为“添加”按钮添加动作监听器
            public void actionPerformed(ActionEvent e) {            // 单击按钮执行动作事件
                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
                                        // 通知所有监听器在选择行的索引位置添加了数据行
                    tableModel.fireTableRowsInserted(selectedRow, selectedRow);
                }
            }
        });
panel.add(addButton);                                               // 把“添加”按钮添加到面板上
final JButton updButton = new JButton("修改");                      // 创建“修改”按钮
updButton.addActionListener(new ActionListener() {                  // 为“修改”按钮添加动作监听器
            public void actionPerformed(ActionEvent e) {            // 单击按钮执行动作事件
                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
                                        // 通知所有监听器修改了选择行索引位置的数据行
                    tableModel.fireTableRowsUpdated(selectedRow, selectedRow);
                }
            }
        });
```

```
        panel.add(updButton);                                          // 把“修改”按钮添加到面板上
        final JButton delButton = new JButton("删除");                  // 创建“删除”按钮
        delButton.addActionListener(new ActionListener() {             // 为“删除”按钮添加动作监听器
                    public void actionPerformed(ActionEvent e) {       // 单击按钮执行动作事件
                        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
                                                    // 通知所有监听器删除了选择行索引位置的数据行
                            tableModel.fireTableRowsDeleted(selectedRow, selectedRow);
                        }
                    }
                });
        panel.add(delButton);                                          // 把“删除”按钮添加到面板上
    }
    public static void main(String[] args) {
        try {
            UIManager.setLookAndFeel("com.sun.java.swing.plaf.nimbus.NimbusLookAndFeel");
        } catch (Throwable e) {
            e.printStackTrace();
        }
        TableModelEventFrame frame = new TableModelEventFrame(); // 创建窗体
        frame.setVisible(true);                                  // 显示窗体
    }
}
```

实例运行效果如图 18.11 所示。

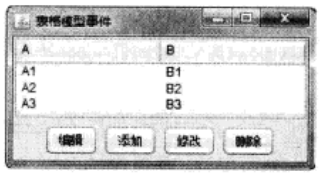


图 18.11　实例运行效果

18.5　照猫画虎——基本功训练

18.5.1　基本功训练 1——表头与列的高度设置

视频讲解：光盘\mr\lx\18\表头与列的高度设置.exe

实例位置：光盘\mr\ym\18.08\src

Swing 中的表格在创建之后大小就固定了。如果想放大表格中文字的字体，则会有一部分文字显示不了。本实例将演示如何设置表头和列的高度，实例的代码如下：

```
public class ResizeTableDemo extends JFrame {
    private static final long serialVersionUID = 6589584750853602612L;
    private JPanel contentPane;
    private JTable table1;
    private JTable table2;
    public static void main(String[] args) {
```

```
        try {                                                    // 修改 Swing 的外观
            UIManager.setLookAndFeel("com.sun.java.swing.plaf.nimbus.NimbusLookAndFeel");
        } catch (Throwable e) {
            e.printStackTrace();
        }
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                try {
                    ResizeTableDemo frame = new ResizeTableDemo();   // 创建窗体对象
                    frame.setVisible(true);                          // 将窗体设置成可见
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }
    public ResizeTableDemo() {
        addWindowListener(new WindowAdapter() {                      // 为窗体激活事件增加监听器
            @Override
            public void windowActivated(WindowEvent e) {
                do_this_windowActivated(e);
            }
        });
        setTitle("设置表格宽度与高度");                               // 设置窗体的标题
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);              // 设置窗体关闭时的属性
        setBounds(100, 100, 450, 250);                               // 设置窗体的位置和大小
        contentPane = new JPanel();                                  // 创建内容面板
        contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));          // 为内容面板设置边框
        setContentPane(contentPane);                                 // 应用内容面板
        contentPane.setLayout(new GridLayout(1, 2, 5, 5));           // 将内容面板设置成网格布局
        JPanel panel1 = new JPanel();                                // 创建面板
        contentPane.add(panel1);                                     // 应用面板
        panel1.setLayout(new BorderLayout(0, 0));                    // 将面板设置成边界布局
        JLabel label1 = new JLabel("默认的表格");                     // 创建包含指定内容的标签
        label1.setHorizontalAlignment(SwingConstants.CENTER);        // 将标签文本设置成居中显示
        label1.setFont(new Font("微软雅黑", Font.PLAIN, 16));         // 设置标签文本的字体
        panel1.add(label1, BorderLayout.NORTH);                      // 应用标签
        JScrollPane scrollPane1 = new JScrollPane();                 // 创建滚动面板
        panel1.add(scrollPane1, BorderLayout.CENTER);                // 应用滚动面板
        table1 = new JTable();                                       // 创建表格
        scrollPane1.setViewportView(table1);                         // 应用表格
        JPanel panel2 = new JPanel();                                // 创建面板
        contentPane.add(panel2);                                     // 应用面板
        panel2.setLayout(new BorderLayout(0, 0));                    // 将面板设置成边界布局
        JLabel label2 = new JLabel("自定义表格");                     // 创建包含指定内容的标签
        label2.setHorizontalAlignment(SwingConstants.CENTER);        // 将标签文本设置成居中显示
        label2.setFont(new Font("微软雅黑", Font.PLAIN, 16));         // 设置标签文本的字体
        panel2.add(label2, BorderLayout.NORTH);                      // 应用标签
```

```
        JScrollPane scrollPane2 = new JScrollPane();                          // 创建滚动面板
        panel2.add(scrollPane2, BorderLayout.CENTER);                         // 应用滚动面板
        table2 = new JTable();                                                // 创建表格
        table2.setFont(new Font("微软雅黑", Font.PLAIN, 14));                  // 设置表格的字体
        table2.setRowHeight(35);                                              // 设置行高
        JTableHeader header = table2.getTableHeader();                        // 获得表格的表头对象
        header.setFont(new Font("微软雅黑", Font.PLAIN, 16));                  // 设置表头的字体
        header.setPreferredSize(new Dimension(header.getWidth(), 40));        // 设置表头的高度
        scrollPane2.setViewportView(table2);                                  // 应用表格
    }
    protected void do_this_windowActivated(WindowEvent e) {
        DefaultTableModel model = new DefaultTableModel();                    // 创建表格模型
        model.setRowCount(0);                                                 // 清空表格模型
        model.setColumnIdentifiers(new Object[] { "排名", "语言" });           // 设置表头内容
        model.addRow(new Object[] { "1", "Java" });                           // 增加数据行
        model.addRow(new Object[] { "2", "C" });                              // 增加数据行
        model.addRow(new Object[] { "3", "C#" });                             // 增加数据行
        table1.setModel(model);                                               // 应用表格模型
        table2.setModel(model);                                               // 应用表格模型
    }
}
```

实例运行效果如图 18.12 所示。

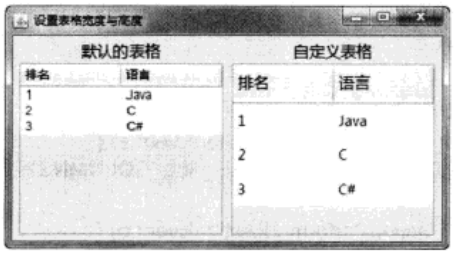


图 18.12　实例运行效果

你问我答：setRowHeight()方法和 setPreferredSize()方法各有什么作用？

setRowHeight()方法用于设置表格所有行的高度，但是不包括表头。setPreferredSize()方法用于设置控件的首选大小。这两个方法的声明分别如下：

public void setRowHeight(int rowHeight)

参数说明：rowHeight 是新的行高。

public void setPreferredSize(Dimension preferredSize)

参数说明：preferredSize 是新的首选大小。

技巧：setRowHeight()方法还有一个重载形式，可以根据指定的行来设置该行的高度。

照猫画虎：设置表格中的奇数行高度比偶数行大 10 像素，表头不变。（20 分）（实例位置：光盘\mr\ym\18.08\zmhh）

18.5.2 基本功训练 2——调整表格各列的宽度

视频讲解：光盘\mr\lx\18\调整表格各列的宽度.exe

实例位置：光盘\mr\ym\18.09\src

在用户调整表格的列时，其他列的宽度也会随之改变。但是，JTable 也提供了其他的调整方式，如仅改变最后一列的宽度等。本实例将演示调整表格各列宽度的方式。

实现过程如下：

（1）在项目中新建窗体类 TableColumnWidthDemo，并增加单选按钮组和表格控件。

（2）监听窗体激活事件，并增加表格中的数据，关键代码如下：

```
protected void do_this_windowActivated(WindowEvent e) {
    DefaultTableModel tableModel = (DefaultTableModel) table.getModel();                    // 获得表格模型
    tableModel.setRowCount(0);                                                              // 清空表格中的数据
    tableModel.setColumnIdentifiers(new Object[] { "书名", "出版社", "出版时间", "丛书类别", "定价" }); // 设置表头
    tableModel.addRow(new Object[] { "Java 从入门到精通（第 2 版）", "清华大学出版社", "2010-07-01", "软件
工程师入门丛书", "59.8 元" });                                                              // 增加表格中的数据
    tableModel.addRow(new Object[] { "PHP 从入门到精通（第 2 版）", "清华大学出版社", "2010-07-01", "软件
工程师入门丛书", "69.8 元" });
    tableModel.addRow(new Object[] { "Visual Basic 从入门到精通（第 2 版）", "清华大学出版社", "2010-07-01",
"软件工程师入门丛书", "69.8 元" });
    tableModel.addRow(new Object[] { "Visual C++从入门到精通（第 2 版）", "清华大学出版社", "2010-07-01", "
软件工程师入门丛书", "69.8 元" });
    table.setModel(tableModel);                                                             // 更新表格模型
}
```

（3）监听用户在单选按钮组中的选择事件，关键代码如下：

```
protected void do_radioButton1_actionPerformed(ActionEvent e) {
    table.setAutoResizeMode(JTable.AUTO_RESIZE_ALL_COLUMNS);      // 修改表格宽度的调整方式
}
protected void do_radioButton2_actionPerformed(ActionEvent e) {
    table.setAutoResizeMode(JTable.AUTO_RESIZE_LAST_COLUMN);      // 修改表格宽度的调整方式
}
protected void do_radioButton3_actionPerformed(ActionEvent e) {
    table.setAutoResizeMode(JTable.AUTO_RESIZE_NEXT_COLUMN);      // 修改表格宽度的调整方式
}
protected void do_radioButton4_actionPerformed(ActionEvent e) {
    table.setAutoResizeMode(JTable.AUTO_RESIZE_OFF);              // 修改表格宽度的调整方式
}
```

实例运行效果如图 18.13 所示。

图 18.13 实例运行效果

◈ 你问我答：setAutoResizeMode()方法有哪些参数？

JTable 类提供了 5 种方式来调整表格列的方式，其说明如表 18.9 所示。

表 18.9　表格常用调整模式说明

字　段	说　明
AUTO_RESIZE_ALL_COLUMNS	在所有的调整大小操作中，按比例调整所有的列
AUTO_RESIZE_LAST_COLUMN	所有的调整大小操作中，只对最后一列进行调整
AUTO_RESIZE_NEXT_COLUMN	在 UI 中调整了一个列时，对其下一列进行相反方向的调整
AUTO_RESIZE_OFF	不自动调整列的宽度，使用滚动条
AUTO_RESIZE_SUBSEQUENT_COLUMNS	在 UI 调整中，更改后续列以保持总宽度不变。此为默认行为

照猫画虎：使用组合框完成模式选择功能。（20 分）（实例位置：光盘\mr\ym\18.09\zmhh）

18.5.3　基本功训练 3——设置表格的选择模式

视频讲解：光盘\mr\lx\18\设置表格的选择模式.exe

实例位置：光盘\mr\ym\18.10\src

对于表格而言，可以有多种选择模式，如选择一行、多选、一列、多列等。本实例将演示如何设置表格的选择模式。

实现过程如下：

（1）在项目中新建窗体类 TableSelectionModeDemo，并增加单选按钮组和表格控件。

（2）监听窗体激活事件，并增加表格中的数据，关键代码如下：

```
protected void do_this_windowActivated(WindowEvent e) {
    DefaultTableModel tableModel = (DefaultTableModel) table.getModel();                    // 获得表格模型
    tableModel.setRowCount(0);                                                              // 清空表格中的数据
    tableModel.setColumnIdentifiers(new Object[] { "书名", "出版社", "出版时间", "丛书类别", "定价" }); // 增加数据
    tableModel.addRow(new Object[] { "Java 从入门到精通（第 2 版）", "清华大学出版社", "2010-07-01", "软件
工程师入门丛书", "59.8 元" });
    tableModel.addRow(new Object[] { "PHP 从入门到精通（第 2 版）", "清华大学出版社", "2010-07-01", "软件
工程师入门丛书", "69.8 元" });
    tableModel.addRow(new Object[] { "Visual Basic 从入门到精通（第 2 版）", "清华大学出版社", "2010-07-01",
"软件工程师入门丛书", "69.8 元" });
    tableModel.addRow(new Object[] { "Visual C++从入门到精通（第 2 版）", "清华大学出版社", "2010-07-01", "
软件工程师入门丛书", "69.8 元" });
    table.setModel(tableModel);                                                             // 应用表格模型
}
```

（3）监听用户在单选按钮组中的选择事件，关键代码如下：

```
protected void do_radioButton1_actionPerformed(ActionEvent e) {
    table.getSelectionModel().setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
                                                                                            // 修改表格的选择模式
}
protected void do_radioButton2_actionPerformed(ActionEvent e) {
    table.getSelectionModel().setSelectionMode(ListSelectionModel.SINGLE_INTERVAL_SELECTION);
                                                                                            // 修改表格的选择模式
```

```
}
protected void do_radioButton3_actionPerformed(ActionEvent e) {
    table.getSelectionModel().setSelectionMode(ListSelectionModel.MULTIPLE_INTERVAL_SELECTION);
                                                                    // 修改表格的选择模式
}
```

实例运行效果如图 18.14 所示。

图 18.14 实例运行效果

你问我答：setSelectionMode()方法有哪些参数？

JTable 类提供了 3 种方式来调整表格的选择模式，其说明如表 18.10 所示。

表 18.10 表格常用选择模式说明

字 段	说 明
MULTIPLE_INTERVAL_SELECTION	一次选择一个或多个连续的索引范围
SINGLE_INTERVAL_SELECTION	一次选择一个连续的索引范围
SINGLE_SELECTION	一次选择一个列表索引

照猫画虎：增加一个复选框来禁用和启用列选择。（20 分）（实例位置：光盘\mr\ym\18.10\zmhh）

18.5.4 基本功训练 4——为表头增添提示信息

视频讲解：光盘\mr\lx\18\为表头增添提示信息.exe

实例位置：光盘\mr\ym\18.11\src

为了节约窗体空间，表头通常使用缩写形式。为了方便用户使用，可以对其增加提示信息。本实例将完成这个功能。

实现过程如下：

（1）在项目中新建窗体类 TableSelectionModeDemo，并增加单选按钮组和表格控件。

（2）继承 JTableHeader 类编写 ToolTipHeader，它实现了根据用户选择的列编号返回提示信息的功能，代码如下：

```
private class ToolTipHeader extends JTableHeader {
    private static final long serialVersionUID = 6694115973725345619L;
    private String[] toolTips;
    public ToolTipHeader(TableColumnModel model) {
        super(model);
    }
    public void setToolTips(String[] toolTips) {
        this.toolTips = toolTips;
```

```
    }
    @Override
    public String getToolTipText(MouseEvent event) {
        int index = columnAtPoint(event.getPoint());                    // 获得鼠标在表头上的位置
        if (index != -1) {
            return toolTips[index];                                     // 返回提示信息
        } else {
            return "";
        }
    }
}
```

（3）在构造方法中增加了表格并替换了表头，关键代码如下：

```
public TableHeadTip() {
    setTitle("为表头增加提示信息");
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setBounds(100, 100, 450, 200);
    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);
    DefaultTableModel tableModel = (DefaultTableModel) table.getModel();                    // 获得表模型
    tableModel.setColumnIdentifiers(new Object[] { "Title", "Press", "Date", "Category", "Price" });// 设置表头内容
    tableModel.addRow(new Object[] { "Java 从入门到精通（第 2 版）", "清华大学出版社", "2010-07-01", "软件
工程师入门丛书", "59.8 元" });                                                              // 增加行
    tableModel.addRow(new Object[] { "PHP 从入门到精通（第 2 版）", "清华大学出版社", "2010-07-01", "软件
工程师入门丛书", "69.8 元" });
    tableModel.addRow(new Object[] { "Visual Basic 从入门到精通（第 2 版）", "清华大学出版社", "2010-07-01",
"软件工程师入门丛书", "69.8 元" });
    tableModel.addRow(new Object[] { "Visual C++从入门到精通（第 2 版）", "清华大学出版社", "2010-07-01", "
软件工程师入门丛书", "69.8 元" });
    table.setModel(tableModel);                                                             // 设置表格模型
    String[] tips = { "书名", "出版社", "出版时间", "丛书类别", "定价" };                          // 创建提示信息
    ToolTipHeader header = new ToolTipHeader(table.getColumnModel());                       // 创建新表头
    header.setToolTips(tips);                                                               // 设置提示信息
    table.setTableHeader(header);                                                           // 设置新表头
}
```

实例运行效果如图 18.15 所示。

图 18.15　实例运行效果

照猫画虎：修改提示信息的样式，字体使用微软雅黑，大小为 6，颜色为红色。（20 分）（实例位置：光盘\mr\ym\18.11\zmhh）

18.5.5 基本功训练 5——单元格的粗粒度排序

视频讲解：光盘\mr\lx\18\单元格的粗粒度排序.exe

实例位置：光盘\mr\ym\18.12\src

在使用表格时，通常需要其提供排序功能。这样可以根据不同的需要选择不同的列。本实例将完成这个功能。

实现过程如下：

（1）在项目中新建窗体类 SortedTableDemo，并增加表格控件。

（2）监听窗体激活事件，完成向表中增加数据的功能，并使表格支持排序，关键代码如下：

```
protected void do_this_windowActivated(WindowEvent e) {
    DefaultTableModel tableModel = (DefaultTableModel) table.getModel();
    // 获得表格模型
    tableModel.setRowCount(0);                                          // 清空表格中的数据
    tableModel.setColumnIdentifiers(new Object[] { "书名", "出版社", "出版时间", "丛书类别", "定价" });
    // 设置表头
    tableModel.addRow(new Object[] { "Java 从入门到精通（第 2 版）", "清华大学出版社", "2010-07-01", "软件
工程师入门丛书", "59.8 元" });
    // 增加列
    tableModel.addRow(new Object[] { "PHP 从入门到精通（第 2 版）", "清华大学出版社", "2010-07-01", "软件
工程师入门丛书", "69.8 元" });
    tableModel.addRow(new Object[] { "Visual Basic 从入门到精通（第 2 版）", "清华大学出版社", "2010-07-01",
"软件工程师入门丛书", "69.8 元" });
    tableModel.addRow(new Object[] { "Visual C++从入门到精通（第 2 版）", "清华大学出版社", "2010-07-01",
"软件工程师入门丛书", "69.8 元" });
    table.setModel(tableModel);                                         // 应用表格模型
    table.setAutoCreateRowSorter(true);                                 // 支持排序方式
}
```

实例运行效果如图 18.16 所示。

图 18.16 实例运行效果

你问我答：setAutoCreateRowSorter()方法有什么作用？

表格在默认情况下是不支持排序的，使用该方法可以让表格支持粗粒度排序，即根据列来排序。

照猫画虎：不使用监听器完成本实例的功能。提示：将增加数据等代码放置在构造方法中。（20 分）（实例位置：光盘\mr\ym\18.12\zmhh）

照猫画虎栏目分数统计：

照猫画虎题目	1	2	3	4	5	总分数	
分数							

18.6　情景应用——拓展与实践

18.6.1　情景应用 1——删除指定的图书信息

视频讲解：光盘\mr\lx\18\删除指定的图书信息.exe

实例位置：光盘\mr\ym\18.13\src

Swing 中表格的数据可以根据需求动态变化，如对于表格中不需要的数据，可以将其删除。本实例将演示如何删除用户选择的行，实例的运行效果如图 18.17 所示。

图 18.17　实例运行效果

实现过程如下：

（1）在项目中新建窗体类 RowDeleteDemo，并增加表格和按钮控件。

（2）监听窗体激活事件，完成向表中增加数据的功能。这里使用了表格模型对象，它可以很方便地处理表格中的数据，增加的数据主要是明日科技公司出版的图书。关键代码如下：

```
protected void do_this_windowActivated(WindowEvent e) {
    DefaultTableModel tableModel = (DefaultTableModel) table.getModel();
    // 获得表格模型
    tableModel.setRowCount(0);// 清空表格中的数据
    tableModel.setColumnIdentifiers(new Object[] { "书名", "出版社", "出版时间", "丛书类别", "定价" });
    // 设置表头
    tableModel.addRow(new Object[] { "Java 从入门到精通（第 2 版）", "清华大学出版社", "2010-07-01", "软件
工程师入门丛书", "59.8 元" });
    // 增加列
    tableModel.addRow(new Object[] { "PHP 从入门到精通（第 2 版）", "清华大学出版社", "2010-07-01", "软件
工程师入门丛书", "69.8 元" });
    tableModel.addRow(new Object[] { "Visual Basic 从入门到精通（第 2 版）", "清华大学出版社", "2010-07-01",
"软件工程师入门丛书", "69.8 元" });
    tableModel.addRow(new Object[] { "Visual C++从入门到精通（第 2 版）", "清华大学出版社", "2010-07-01",
"软件工程师入门丛书", "69.8 元" });
    table.setModel(tableModel);                                    // 应用表格模型
}
```

（3）监听用户单击“删除行”按钮事件，这是通过 do_button_actionPerformed()方法完成的。在该方法中，首先获得了表格模型，然后获得用户选择的行，如果用户未选择任何行，则提示用户进行选择。然后根据选择的行来删除数据。关键代码如下：

```
protected void do_button_actionPerformed(ActionEvent e) {
    DefaultTableModel model = (DefaultTableModel) table.getModel();    // 获得表格模型
    int index = table.getSelectedRow();                                // 获得用户选择的行
```

```
    if (index == -1) {                                          // 如果用户没有选择任何行，则进行提示
        JOptionPane.showMessageDialog(this, "请选择要删除的行", "", JOptionPane.WARNING_MESSAGE);
        return;
    }
    model.removeRow(table.getSelectedRow());                    // 从表格模型中删除用户选择的行
    table.setModel(model);                                      // 更新表格模型
}
```

DIY：删除用户选择的连续多行数据。（20 分）（实例位置：光盘\mr\ym\18.13\diy）

18.6.2 情景应用 2——分页显示 100 内的平方数

视频讲解：光盘\mr\lx\18\分页显示 100 内的平方数.exe

实例位置：光盘\mr\ym\18.14\src

对于数据量非常大的表格，每次加载、显示全部数据是非常耗时的，此时可以考虑使用分页技术来加载部分数据。当用户需要浏览下一页时，可以再获得下一页中的数据并在表格中进行显示。本实例将使用分页技术显示 100 内的平方数，实例的运行效果如图 18.18 所示。

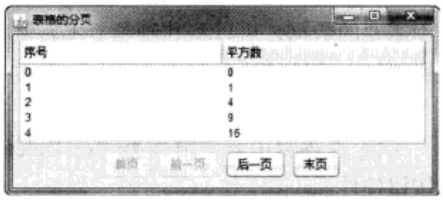


图 18.18 实例运行效果

实现过程如下：

（1）在项目中新建窗体类 PageTableDemo，并增加表格和按钮控件。

（2）监听窗体激活事件，这是通过 do_this_windowActivated()方法完成的。在该方法中，首先向表格模型中增加全部要显示的数据，然后根据分页的结果来计算当前页需要显示的数据，最后在表格中进行显示。同时，对按钮是否可用进行控制，如已经处于第一页时“前一页”按钮就应该被禁用。关键代码如下：

```
protected void do_this_windowActivated(WindowEvent e) {
    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[] { "序号", "平方数" });          // 定义表头
    for (int i = 0; i < 100; i++) {
        defaultModel.addRow(new Object[] { i, i * i });                           // 向表格模型中增加数据
    }
    maxPageNumber = (int) Math.ceil(defaultModel.getRowCount() / pageSize);       // 计算总页数
    DefaultTableModel newModel = new DefaultTableModel();                         // 创建新的表格模型
    newModel.setColumnIdentifiers(new Object[] { "序号", "平方数" });              // 定义表头
    for (int i = 0; i < pageSize; i++) {
        newModel.addRow((Vector) defaultModel.getDataVector().elementAt(i));      // 根据页面大小来获得数据
    }
    table.setModel(newModel);                                                     // 设置表格模型
    firstPageButton.setEnabled(false);                                            // 禁用“首页”按钮
    latePageButton.setEnabled(false);                                             // 禁用“前一页”按钮
    nextPageButton.setEnabled(true);                                              // 启用“后一页”按钮
    lastPageButton.setEnabled(true);                                              // 启用“末页”按钮
}
```

（3）监听用户单击“首页”按钮事件，它是通过 do_firstPageButton_actionPerformed()方法实现的。在该方法中计算了首页应该显示的数据，更新表格数据并对相关的按钮进行了启用和禁用。关键代码如下：

```
protected void do_firstPageButton_actionPerformed(ActionEvent e) {
    currentPageNumber = 1;                                          // 将当前页码设置成 1
    Vector dataVector = defaultModel.getDataVector();               // 获得原表格模型中的数据
    DefaultTableModel newModel = new DefaultTableModel();           // 创建新的表格模型
    newModel.setColumnIdentifiers(new Object[] { "序号", "平方数" }); // 定义表头
    for (int i = 0; i < pageSize; i++) {
        newModel.addRow((Vector) dataVector.elementAt(i));          // 根据页面大小来获得数据
    }
    table.setModel(newModel);                                       // 设置表格模型
    firstPageButton.setEnabled(false);                              // 禁用“首页”按钮
    latePageButton.setEnabled(false);                               // 禁用“前一页”按钮
    nextPageButton.setEnabled(true);                                // 启用“后一页”按钮
    lastPageButton.setEnabled(true);                                // 启用“末页”按钮
}
```

（4）监听用户单击“前一页”按钮事件，它是通过 do_latePageButton_actionPerformed()方法实现的。在该方法中计算了前一页应该显示的数据，更新表格数据并对相关的按钮进行了启用和禁用。关键代码如下：

```
protected void do_latePageButton_actionPerformed(ActionEvent e) {
    currentPageNumber--;                                            // 将当前页面减 1
    Vector dataVector = defaultModel.getDataVector();               // 获得原表格模型中的数据
    DefaultTableModel newModel = new DefaultTableModel();           // 创建新的表格模型
    newModel.setColumnIdentifiers(new Object[] { "序号", "平方数" }); // 定义表头
    for (int i = 0; i < pageSize; i++) {
                                                                    // 根据页面大小来获得数据
        newModel.addRow((Vector) dataVector.elementAt((int) (pageSize * (currentPageNumber - 1) + i)));
    }
    table.setModel(newModel);                                       // 设置表格模型
    if (currentPageNumber == 1) {
        firstPageButton.setEnabled(false);                          // 禁用“首页”按钮
        latePageButton.setEnabled(false);                           // 禁用“前一页”按钮
    }
    nextPageButton.setEnabled(true);                                // 启用“后一页”按钮
    lastPageButton.setEnabled(true);                                // 启用“末页”按钮
}
```

（5）监听用户单击“后一页”按钮事件，它是通过 do_nextPageButton_actionPerformed()方法实现的。在该方法中计算了下一页应该显示的数据，更新表格数据并对相关的按钮进行了启用和禁用。关键代码如下：

```
protected void do_nextPageButton_actionPerformed(ActionEvent e) {
    currentPageNumber++;                                            // 将当前页面加 1
    Vector dataVector = defaultModel.getDataVector();               // 获得原表格模型中的数据
    DefaultTableModel newModel = new DefaultTableModel();           // 创建新的表格模型
    newModel.setColumnIdentifiers(new Object[] { "序号", "平方数" }); // 定义表头
    if (currentPageNumber == maxPageNumber) {
        int lastPageSize = (int) (defaultModel.getRowCount() - pageSize * (maxPageNumber - 1));
        for (int i = 0; i < lastPageSize; i++) {
                                                                    // 根据页面大小来获得数据
            newModel.addRow((Vector) dataVector.elementAt((int) (pageSize * (maxPageNumber - 1) + i)));
        }
        nextPageButton.setEnabled(false);                           // 禁用“后一页”按钮
```

```
            lastPageButton.setEnabled(false);                                    // 禁用“末页”按钮
        } else {
            for (int i = 0; i < pageSize; i++) {
                                                                                 // 根据页面大小来获得数据
                newModel.addRow((Vector) dataVector.elementAt((int) (pageSize * (currentPageNumber - 1) +
i)));
            }
        }
        table.setModel(newModel);                                                // 设置表格模型
        firstPageButton.setEnabled(true);                                        // 启用“首页”按钮
        latePageButton.setEnabled(true);                                         // 启用“前一页”按钮
    }
```

（6）监听用户单击“末页”按钮事件，它是通过 do_lastPageButton_actionPerformed()方法实现的。在该方法中计算了末页应该显示的数据，更新表格数据并对相关的按钮进行了启用和禁用。关键代码如下：

```
    protected void do_lastPageButton_actionPerformed(ActionEvent e) {
        currentPageNumber = maxPageNumber;                                       // 将当前页面设置为末页
        Vector dataVector = defaultModel.getDataVector();                        // 获得原表格模型中的数据
        DefaultTableModel newModel = new DefaultTableModel();                    // 创建新的表格模型
        newModel.setColumnIdentifiers(new Object[] { "序号", "平方数" });          // 定义表头
        int lastPageSize = (int) (defaultModel.getRowCount() - pageSize * (maxPageNumber - 1));
        if (lastPageSize == 5) {
            for (int i = 0; i < pageSize; i++) {
                                                                                 // 根据页面大小来获得数据
                newModel.addRow((Vector) dataVector.elementAt((int) (pageSize * (maxPageNumber - 1) + i)));
            }
        } else {
            for (int i = 0; i < lastPageSize; i++) {
                                                                                 // 根据页面大小来获得数据
                newModel.addRow((Vector) dataVector.elementAt((int) (pageSize * (maxPageNumber - 1) + i)));
            }
        }
        table.setModel(newModel);                                                // 设置表格模型
        firstPageButton.setEnabled(true);                                        // 启用“首页”按钮
        latePageButton.setEnabled(true);                                         // 启用“前一页”按钮
        nextPageButton.setEnabled(false);                                        // 禁用“后一页”按钮
        lastPageButton.setEnabled(false);                                        // 禁用“末页”按钮
    }
```

DIY：将每页显示的数据由 5 行变成 6 行。（20 分）（实例位置：光盘\mr\ym\18.14\diy）

18.6.3 情景应用 3——五光十色的单元格

视频讲解：光盘\mr\lx\18\五光十色的单元格.exe

实例位置：光盘\mr\ym\18.15\src

Swing 中的表格背景色默认是白色的，为了让程序更加美观，可以自定义单元格的背景色。本实例使用渲染器来为不同的单元格生成不同颜色，颜色是使用随机数生成的。实例的运行效果如图 18.19 所示。

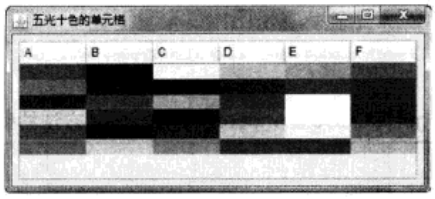


图 18.19　实例运行效果

实现过程如下：

（1）在项目中新建窗体类 ColorfulTableCellDemo，并增加表格控件。

（2）创建内部类 ColorTableCellRenderer，它实现了 TableCellRenderer 接口，用于对单元格样式进行设置。本实例在 getTableCellRendererComponent()方法中，将单元格的背景色设置成随机生成的颜色，关键代码如下：

```
private class ColorTableCellRenderer extends JPanel implements TableCellRenderer {
    private static final long serialVersionUID = 8932176536826008653L;
    @Override
    public Component getTableCellRendererComponent(JTable table, Object value, boolean isSelected,
boolean hasFocus, int row, int column) {
        int r = new Random().nextInt(256);                    // 获得一个随机数来定义颜色
        int g = new Random().nextInt(256);                    // 获得一个随机数来定义颜色
        int b = new Random().nextInt(256);                    // 获得一个随机数来定义颜色
        setBackground(new Color(r, g, b));                    // 为单元格设置背景色
        return this;
    }
}
```

（3）在构造方法中创建表格，将其设置成 6 行 6 列，并应用创建的渲染器，关键代码如下：

```
public ColorfulTableCellDemo() {
    setTitle("五光十色的单元格");
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setBounds(100, 100, 450, 200);
    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();
    DefaultTableModel model = (DefaultTableModel) table.getModel();        // 获得表格模型
    model.setColumnCount(6);                                               // 设置表格列数为 6
    model.setRowCount(6);                                                  // 设置表格行数为 6
    table.setModel(model);                                                 // 应用表格模型
    table.setDefaultRenderer(Object.class, new ColorTableCellRenderer());  // 应用渲染器
}
```

你问我答：getTableCellRendererComponent()方法有什么作用？

该方法用于绘制单元格。该方法声明如下：

```
Component getTableCellRendererComponent(JTable table,Object value,boolean isSelected,boolean hasFocus,
```

int row,int column)
参数说明如表 18.11 所示。

表 18.11 getTableCellRendererComponent 方法的参数及说明

参　数	说　明
table	要求渲染器渲染的 JTable，可以为 null
value	要呈现的单元格的值
isSelected	是否使用选中样式的高亮显示来呈现该单元格
hasFocus	单元格是否具有焦点
row	要渲染的单元格的行索引
column	要渲染的单元格的列索引

DIY：将颜色修改成单元格位置的线性函数。红色分量与单元格横坐标成正比，绿色分量与单元格纵坐标成正比，蓝色分量是红色与绿色的和。（20 分）（实例位置：光盘\mr\ym\18.15\diy）

18.6.4 情景应用 4——实现表格的栅栏效果

视频讲解：光盘\mr\lx\18\实现表格的栅栏效果.exe
实例位置：光盘\mr\ym\18.16\src

对于长时间观看表格的用户，如果每行的背景色是相同的，则很容易造成视觉疲劳。为了对用户提供更好的体验，通常使用栅栏效果来显示单元格，即对奇数行和偶数行使用不同的前景色和背景色。本实例将完成这个功能，实例的运行效果如图 18.20 所示。

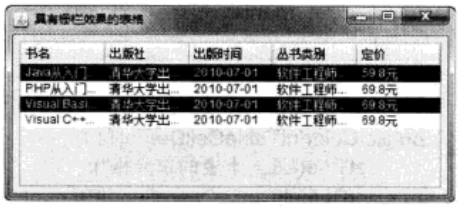


图 18.20 实例运行效果

实现过程如下：

（1）在项目中新建窗体类 FenseTableDemo，并增加表格控件。

（2）创建内部类 FenseRenderer，完成渲染单元格的功能。它将奇数行的前景色设置成白色，背景色设置成蓝色。对于偶数行正好相反。关键代码如下：

```
private class FenseRenderer implements TableCellRenderer {
    @Override
    public Component getTableCellRendererComponent(JTable table, Object value, boolean isSelected,
boolean hasFocus, int row, int column) {
                                                        // 将渲染器转换成标签
        JLabel renderer = (JLabel) new DefaultTableCellRenderer().getTableCellRendererComponent(table,
value, isSelected, hasFocus, row, column);
        if (row % 2 == 0) {                             // 设置标签的前景色和背景色
            renderer.setForeground(Color.WHITE);
            renderer.setBackground(Color.BLUE);
        } else {
            renderer.setForeground(Color.BLUE);
            renderer.setBackground(Color.WHITE);
        }
```

```
        return renderer;
    }
```

DIY：将单元格行号为奇数的换成白色背景，偶数的换成蓝色背景。（20 分）（实例位置：光盘\mr\ym\18.16\diy）

18.6.5　情景应用 5——根据颜色的分量进行排序

视频讲解：光盘\mr\lx\18\根据颜色的分量进行排序.exe

实例位置：光盘\mr\ym\18.17\src

除了根据列的整体进行排序外，还可以指定更详细的排序方式。本实例将根据红、绿、蓝三原色在具体颜色中值的大小来排序。先按红色升序排列，红色值相同则比较绿色，依此类推。实例的运行效果如图 18.21 所示。

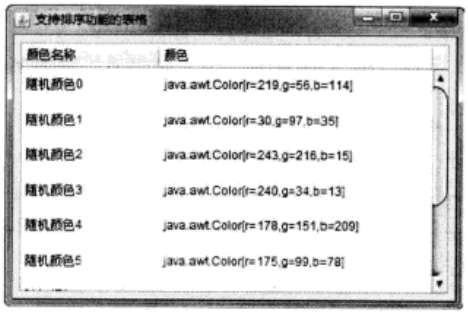


图 18.21　实例运行效果

实现过程如下：

（1）在项目中新建窗体类 SortedTable，并增加表格控件。

（2）监听窗体激活事件，使用随机数生成颜色并将其增加到表格中，然后增加排序功能。排序依据是红、绿、蓝 3 种颜色的分量大小，关键代码如下：

```
protected void do_this_windowActivated(WindowEvent e) {
    DefaultTableModel model = (DefaultTableModel) table.getModel();          // 获得表格模型
    model.setRowCount(0);                                                    // 清空表格中的数据
    model.setColumnIdentifiers(new Object[] { "颜色名称", "颜色" });          // 设置表头
    for (int i = 0; i < 10; i++) {
        int r = new Random().nextInt(256);
        int g = new Random().nextInt(256);
        int b = new Random().nextInt(256);
        model.addRow(new Object[] { "随机颜色" + i, new Color(r, g, b) });    // 增加数据
    }
    TableRowSorter<TableModel> sorter = new TableRowSorter<TableModel>(model);
    sorter.setComparator(1, new Comparator<Color>() {                        // 为第二列设置排序器
            @Override
            public int compare(Color o1, Color o2) {                         // 设置排序方式
                int r = o1.getRed() - o2.getRed();
                int g = o1.getGreen() - o2.getGreen();
                int b = o1.getBlue() - o2.getBlue();
                if (r != 0) {                                                // 首先按红色值排序
                    return r;
                } else if (g != 0) {                                         // 其次按绿色值排序
                    return g;
                } else {
                    return b;                                                // 最后按蓝色值排序
                }
```

```
            }
        });
    table.setRowSorter(sorter);                                    // 为表格增加排序器
    table.getColumnModel().getColumn(0).setPreferredWidth(150);    // 设置第一列的宽度
    table.getColumnModel().getColumn(1).setPreferredWidth(300);    // 设置第二列的宽度
}
```

你问我答：setComparator()方法有什么作用？

该方法可以为指定的列增加排序方式。该方法声明如下：

```
public void setComparator(int column,Comparator<?> comparator)
```

参数说明：column 是增加排序方式的列，comparator 是增加排序的方式。

DIY：修改排序器，让其按蓝色、绿色和红色顺序升序排序。（20 分）（实例位置：光盘\mr\ym\18.17\diy）

情景应用 DIY 栏目分数统计：

DIY 题目	1	2	3	4	5	总分数	
分数							

18.7 自 我 测 试

一、选择题（每题 10 分，5 道题）

1．下列哪些方法是 JTable 类的常用构造方法？（　　）

A．Table()　　B．JTable(int numRows, int numColumns)

C．JTable(Object[][] rowData, Object[] columnNames)　　D．JTable(TableModel dm)

2．JTable 类中哪个方法可以获得该对象的表格模型？（　　）

A．getTableModel()　　B．getDataModel()　　C．getCellModel　　D．getModel

3．如果需要获得表格第二行第一列的元素，可以使用下面哪条语句？（　　）

A．table.getValueAt(2,1);　　B．table.getValueAt(2,0);

C．table.getValueAt(1,1);　　D．table.getValueAt(1,0);

4．如果需要将表格第二行的高度设置成 100，可以使用下面哪条语句？（　　）

A．table.setRowHeight(1, 100);　　B．table.setRowHeight(2, 100);

C．table.setRowHeight(1, 99);　　D．table.setRowHeight(2, 99);

5．使用 DefaultTableModel 中的哪些方法可以给表格增加行数据？（　　）

A．addRow(Object[] rowData)　　B．addRow(Vector rowData)

C．addRows(Object[] rowData)　　D．addRows(Vector rowData)

二、填空题（每题 10 分，5 道题）

1．在实际显示的单元格中，第一行第一列在表格模型中的坐标是_____。

2．DefaultTableModel 中是否提供了删除多行的方法？_____

3．如果已经存在表格模型 model，想将其应用在 JTable 对象 table 上，则可以使用哪条语句？_____

4．JTable 类中哪个方法可以获得用户选择的行号？_____

5．JTable 类中哪个方法可以获得用户选择的行数？_____

测试分数统计：

类别	第 1 题	第 2 题	第 3 题	第 4 题	第 5 题
选择题分数					
填空题分数					
总分数					

18.8　行 动 指 南

开始日期：_____年____月____日　　结束日期：_____年____月____日

序号	内　容	行 动 指 南	
1	照猫画虎栏目 分数（　　）	分数>75 分	优秀，基本功掌握得不错，加油！
		75 分>分数>50 分	及格，知识掌握得不牢，重新做一遍照猫画虎。
		分数<50 分	请使用光盘中提供的“实战能力测试系统”进行提高训练。
	情景应用栏目 分数（　　）	分数>75 分	优秀，综合应用能力很强。
		75 分>分数>50 分	及格，综合应用能力需提高，再练一遍情景应用。
		分数<50 分	请使用光盘中提供的“实战能力测试系统”进行提高训练。
	自我测试栏目 分数（　　）	分数>75 分	优秀，有成为编程高手的潜质。
		分数<75 分	请使用光盘中提供的“实战能力测试系统”进行提高训练。
	综合评价	反复训练后，以上各项分数都在 75 分以上，方可进入下一堂课学习。	
2	编程实践能力培养：本栏目提供了一些实践题目，对于培养编程能力很有效，要做好。	（1）为表格增加插入行的功能。	
		（2）为表格增加查找指定行数据的功能。	
		（3）为表格增加修改指定行数据的功能。	
		（4）使用 JTable 控件显示数据库中的表格。	
3	编程习惯培养：本堂课培养读者在学习开发中记笔记的习惯，把开发中遇到的问题、总结的经验记录下来。		
4	创新能力培养：看看身边有没有可以用编程解决的问题，记录到右边的表格中。根据学习进度，尝试编写解决这些问题的小程序。		

18.9 成功可以复制——微型博客 Twitter 创始人

Twitter，这个发布于 2006 年的微型博客网站，目前已成为互联网时代最新最热的明星。对很多人来说，Twitter 的迅速崛起实在令人意外，对 Twitter 的创始人埃文·威廉姆斯（Evan Williams）来说，同样颇为感慨。

要知道，他当初推出 Twitter，主要是为了方便交流和共享。

埃文·威廉姆斯，1970 年 8 月 20 日出生在内布拉斯加州的一个农场，高中毕业后考入了内布拉斯加州大学，中途认为“上大学是浪费时间”而退学。1994 年，埃文·威廉姆斯创办了一家互联网公司。虽然埃文·威廉姆斯当时并不懂互联网，但他认为互联网最终将非常重要。通过自学，埃文·威廉姆斯知道了如何创建网站，并给一些企业做一些项目。但因没有管理经验，企业没经营多久就倒闭了。回忆过去，埃文·威廉姆斯说：“当时我赔了很多钱，包括我父亲的投资。公司因为拖欠国税局的税款而关门，当时我的很多员工都要疯掉了。”

Twitter 首页

在第一次创业失败后，埃文·威廉姆斯并没有放弃，他决定去硅谷闯荡。威廉姆斯进入一家媒体公司，一开始做销售，后来开始编写电脑程序，很快，他就开始承接硅谷巨头英特尔公司和惠普公司的业务。但他厌倦了公司的环境，决定辞职继续创业。

1999 年，埃文·威廉姆斯与人合伙成立了 Pyra 实验室，主要制作管理软件。在这期间，为了方便交流，埃文·威廉姆斯偶然发明了博客。后来，博客变成了公司的主业，博客网站（Blogger.com）也随之上线。前两年大火特火的博客一词，就是埃文·威廉姆斯发明的。2001 年，《财富》杂志曾将“博客网”评选为最佳创新网站。2000 年，互联网泡沫破灭，埃文·威廉姆斯和他的公司也未能幸免。他别无选择，只有裁掉公司所有员工，最后只剩下他一人独自苦撑。不过，“博客网”不仅改变了网络世界，也改变了埃文·威廉姆斯的人生。2003 年，谷歌公司决定收购“博客网”，濒临绝境的埃文·威廉姆斯把公司卖给了谷歌，这笔交易让威廉姆斯捞到了他的第一桶金。直到今天，埃文·威廉姆斯仍然认为，“博客网”是他最大的创造。

2005 年，埃文·威廉姆斯成立了一家播客公司，又“偶然”地创造了 Twitter，谱写了新的互联网奇迹。

经典语录

企业家分为两种类型，一种是把赚钱当作目标，另一种是为了创造新事物，创造从未有过的产品或服务。我属于第二种，并热衷于此。这是世界上出现新事物的源泉，而我非常幸运能够创造出一个改变世界的产品。

深度评价

从埃文·威廉姆斯身上，我们学到了三点经验：一是积极地思考，创新地工作，想办法让工作更高效、更便捷、更快速，或许你就发现了机会；二是不断尝试才能创造价值，坚持才能收获成功；三是专注于为用户创造最大的价值，而不是其他（如金钱等）。

第19堂课

树控件的应用

（视频讲解：141 分钟）

树状结构是一种常用的信息表现形式之一，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，本堂课将深入学习该类的使用方法，以及一些相关类的使用方法，在讲解过程中为了便于读者理解结合了大量的实例。

学习摘要：

- 创建树的方法
- 维护树模型的方法
- 定制树的基本方法
- 处理选中节点事件的方法
- 处理展开节点事件的方法

19.1 创建树控件

树状结构是一种常用的信息表现形式之一，它可以直观地显示出一组信息的层次结构，可以使用 Swing 中的 javax.swing.JTree 类来创建树，该类的常用构造方法如表 19.1 所示。

表 19.1 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

树组件本身不能显示滚动条，一般都与滚动面板一起使用。例如：

```
final JScrollPane scrollPane = new JScrollPane();          // 创建滚动面板
JTree tree = new JTree();                                  // 创建默认的树
scrollPane.setViewportView(tree);                          // 把树放到滚动面板中
```

这段代码首先创建了一个滚动面板，然后创建了一个默认的树，并把树添加到滚动面板的视图中，如果把滚动面板放到窗体上，将在窗体上显示如图 19.1 所示的树。由于把树放到滚动面板中，所以当窗体中不能完全显示树的所有节点时，将会出现滚动条。

图 19.1 使用默认方式创建树

说明：上面代码只是说明了如何使用创建默认树的构造方法 JTree()创建树，使用树节点创建树和使用树模型创建树将在本堂课陆续讲解。

javax.swing.tree.DefaultMutableTreeNode 类实现了 TreeNode 接口，用来创建树的节点，一个节点只能有一个父节点，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，如果某个节点不需要有子节点，可以将其设置为不允许有子节点，该类的常用构造方法如表 19.2 所示。

表 19.2 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，没有父节点的节点称为根节点。可以通过根节点利用 JTree 类的构造方法 JTree(TreeNode root)直接创建树。例如：

```
final JScrollPane scrollPane = new JScrollPane();                                    // 创建滚动面板
DefaultMutableTreeNode rootNode = new DefaultMutableTreeNode("根节点");              // 创建根节点
DefaultMutableTreeNode childNode1 = new DefaultMutableTreeNode("一级子节点 1");     // 创建一级子节点
DefaultMutableTreeNode childNode11 = new DefaultMutableTreeNode("二级子节点 11");// 创建二级子节点
DefaultMutableTreeNode childNode12 = new DefaultMutableTreeNode("二级子节点 12");// 创建二级子节点
childNode1.add(childNode11);                                                         // 为一级子节点添加二级子节点
childNode1.add(childNode12);                                                         // 为一级子节点添加二级子节点
DefaultMutableTreeNode childNode2 = new DefaultMutableTreeNode("一级子节点 2");     // 创建一级子节点
DefaultMutableTreeNode childNode21 = new DefaultMutableTreeNode("二级子节点 21");// 创建二级子节点
DefaultMutableTreeNode childNode22 = new DefaultMutableTreeNode("二级子节点 22");// 创建二级子节点
childNode2.add(childNode21);                                                         // 为一级子节点添加二级子节点
childNode2.add(childNode22);                                                         // 为一级子节点添加二级子节点
rootNode.add(childNode1);                                                            // 为根节点添加一级子节点
rootNode.add(childNode2);                                                            // 为根节点添加一级子节点
tree = new JTree(rootNode);                                                          // 使用根节点创建树
scrollPane.setViewportView(tree);                                                    // 把树放到滚动面板中
```

这段代码首先创建了一个滚动面板，然后创建了根节点、一级子节点和二级子节点，并把二级子节点添加到一级子节点上，使其成为一级子节点的子节点，接着又把一级子节点添加到根节点上，使其成为根节点的子节点，最后通过根节点创建了一个树，并把树添加到滚动面板的视图中，如果把滚动面板放到窗体上，将在窗体上显示如图 19.2 所示的树。

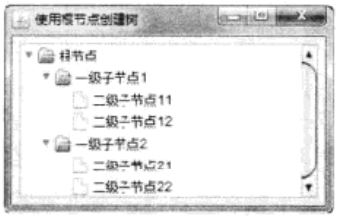


图 19.2　使用根节点创建树

请读者看下面的代码，在这个实例中演示了创建树的方法。

例 19.01　本例利用一个根节点创建了一个树，该树有一个根节点、3 个一级子节点和 4 个二级子节点。实例的关键代码如下：**（实例位置：光盘\mr\ym\19.01）**

```
public TreeCreationDemo() {
    setTitle("为树控件增加节点");
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setBounds(100, 100, 400, 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);
    DefaultMutableTreeNode rootNode = new DefaultMutableTreeNode("根节点");          // 创建根节点
    DefaultMutableTreeNode childNode1 = new DefaultMutableTreeNode("一级子节点 1");// 创建一级子节点
    DefaultMutableTreeNode childNode11 = new DefaultMutableTreeNode("二级子节点 11");// 创建二级子节点
    DefaultMutableTreeNode childNode12 = new DefaultMutableTreeNode("二级子节点 12");// 创建二级子节点
    childNode1.add(childNode11);                                                     // 为一级子节点添加二级子节点
    childNode1.add(childNode12);                                                     // 为一级子节点添加二级子节点
    DefaultMutableTreeNode childNode2 = new DefaultMutableTreeNode("一级子节点 2");  // 创建一级子节点
    DefaultMutableTreeNode childNode21 = new DefaultMutableTreeNode("二级子节点 21");// 创建二级子节点
    DefaultMutableTreeNode childNode22 = new DefaultMutableTreeNode("二级子节点 22");// 创建二级子节点
    childNode2.add(childNode21);                                                     // 为一级子节点添加二级子节点
    childNode2.add(childNode22);                                                     // 为一级子节点添加二级子节点
    DefaultMutableTreeNode childNode3 = new DefaultMutableTreeNode("一级子节点 3");  // 创建一级子节点
```

```
        rootNode.add(childNode1);                                    // 为根节点添加一级子节点
        rootNode.add(childNode2);                                    // 为根节点添加一级子节点
        rootNode.add(childNode3);                                    // 为根节点添加一级子节点
        JTree tree = new JTree(rootNode);                            // 使用根节点创建树
        scrollPane.setViewportView(tree);
    }
```

实例的运行效果如图 19.3 所示。

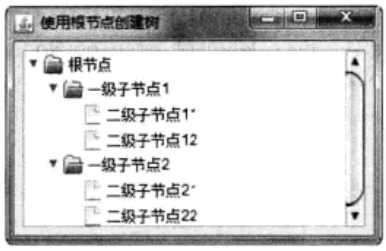


图 19.3　实例运行效果

19.2　维护树模型

除了使用 JTree 类和根节点创建树以外，还可以使用 JTree 类和树模型创建树，javax.swing.tree.TreeModel 接口定义了一个树模型，该模型提供了操作树的方法。在创建树时，可以先创建一个树模型 TreeModel，然后再把树模型作为参数传递给树的构造方法 JTree(TreeModel newModel)创建树。

19.2.1　创建模型对象

javax.swing.tree.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。

☑　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。该方法声明如下：

```
public DefaultTreeModel(TreeNode root)
```

参数说明：root 是作为树的根的 TreeNode 对象。

☑　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。该方法声明如下：

```
public DefaultTreeModel(TreeNode root, boolean asksAllowsChildren)
```

参数说明：root 是作为树的根的 TreeNode 对象。asksAllowsChildren 是一个布尔值，如果任何节点都可以有子节点，则为 false；如果询问每个节点看是否有子节点，则为 true。

由 DefaultTreeModel 类实现的树模型判断节点是否为叶子节点有以下两种方式。

第一种方式：如果节点不存在子节点则为叶子节点。例如：

```
DefaultMutableTreeNode rootNode = new DefaultMutableTreeNode("部门");          // 创建根节点
DefaultMutableTreeNode childNode1 = new DefaultMutableTreeNode("开发部");      // 创建一级子节点
DefaultMutableTreeNode childNode11 = new DefaultMutableTreeNode("开发部经理");// 创建二级子节点
DefaultMutableTreeNode childNode12 = new DefaultMutableTreeNode("开发部成员");// 创建二级子节点
childNode1.add(childNode11);                                                   // 为一级子节点添加二级子节点
childNode1.add(childNode12);                                                   // 为一级子节点添加二级子节点
DefaultMutableTreeNode childNode2 = new DefaultMutableTreeNode("基础部");      // 创建一级子节点
```

```
rootNode.add(childNode1);                                          // 为根节点添加一级子节点
rootNode.add(childNode2);                                          // 为根节点添加一级子节点
DefaultTreeModel treeModel = new DefaultTreeModel(rootNode);       // 使用根节点创建树模型
```

这段代码首先创建了根节点、一级子节点和二级子节点，并把二级子节点添加到一级子节点上，使其成为一级子节点的子节点，接着又把一级子节点添加到根节点上，使其成为根节点的子节点，最后通过根节点创建了一个采用默认方式判断节点是否为叶子节点的树模型，如果通过此树模型创建一个树，把树添加到滚动面板的视图中，再把滚动面板放到窗体上，将在窗体上显示如图 19.4 所示的树。

图 19.4　使用默认方式创建树

说明：上面代码中的二级子节点“开发部经理”和“开发部成员”以及一级子节点“基础部”都将显示没有子节点的叶子节点图标。

第二种方式：根据节点是否允许有子节点，只要不允许有子节点，则该节点就是叶子节点，如果允许有子节点，则不管该节点是否有子节点，都不是叶子节点。将第二个构造方法的参数 asksAllowsChildren 设置为 true，表示允许有子节点，设置为 false 表示不允许有子节点。例如：

```
DefaultMutableTreeNode rootNode = new DefaultMutableTreeNode("公司");                  // 创建根节点
DefaultMutableTreeNode childNode1 = new DefaultMutableTreeNode("设计部");              // 创建一级子节点
DefaultMutableTreeNode childNode11 = new DefaultMutableTreeNode("设计部经理");         // 创建二级子节点
DefaultMutableTreeNode childNode12 = new DefaultMutableTreeNode("设计部成员");         // 创建二级子节点
childNode1.add(childNode11);                                       // 为一级子节点添加二级子节点
childNode1.add(childNode12);                                       // 为一级子节点添加二级子节点
DefaultMutableTreeNode childNode2 = new DefaultMutableTreeNode("财务部");              // 创建一级子节点
rootNode.add(childNode1);                                          // 为根节点添加一级子节点
rootNode.add(childNode2);                                          // 为根节点添加一级子节点
DefaultTreeModel treeModel = new DefaultTreeModel(rootNode, true);// 使用根节点创建允许有子节点的树模型
```

这段代码首先创建了根节点、一级子节点和二级子节点，并把二级子节点添加到一级子节点上，使其成为一级子节点的子节点，接着又把一级子节点添加到根节点上，使其成为根节点的子节点，最后通过根节点创建了一个不允许有子节点的树模型，如果通过此树模型创建一个树，把树添加到滚动面板的视图中，再把滚动面板放到窗体上，将在窗体上显示如图 19.5 所示的树。

图 19.5　使用非默认方式创建树

📖 **说明：上面代码中的二级子节点“设计部经理”和“设计部成员”以及一级子节点“财务部”虽然都没有子节点，但却都将显示允许有子节点的非叶子节点图标。**

19.2.2 设置树控件的模型

树模型创建完成后，再通过 JTree 类的构造方法 JTree(TreeModel newModel)创建树，并把树模型设置为该构造方法的参数，这样就实现了利用树模型创建树。例如：

```
final JScrollPane scrollPane = new JScrollPane();                                    // 创建滚动面板
DefaultMutableTreeNode rootNode = new DefaultMutableTreeNode("公司");                 // 创建根节点
DefaultMutableTreeNode childNode1 = new DefaultMutableTreeNode("开发部");             // 创建一级子节点
DefaultMutableTreeNode childNode11 = new DefaultMutableTreeNode("开发部经理");        // 创建二级子节点
DefaultMutableTreeNode childNode12 = new DefaultMutableTreeNode("开发部成员");        // 创建二级子节点
childNode1.add(childNode11);                                                         // 为一级子节点添加二级子节点
childNode1.add(childNode12);                                                         // 为一级子节点添加二级子节点
DefaultMutableTreeNode childNode2 = new DefaultMutableTreeNode("设计部");             // 创建一级子节点
DefaultMutableTreeNode childNode21 = new DefaultMutableTreeNode("设计部经理");        // 创建二级子节点
DefaultMutableTreeNode childNode22 = new DefaultMutableTreeNode("设计部成员");        // 创建二级子节点
childNode2.add(childNode21);                                                         // 为一级子节点添加二级子节点
childNode2.add(childNode22);                                                         // 为一级子节点添加二级子节点
DefaultMutableTreeNode childNode3 = new DefaultMutableTreeNode("基础部");             // 创建一级子节点
rootNode.add(childNode1);                                                            // 为根节点添加一级子节点
rootNode.add(childNode2);                                                            // 为根节点添加一级子节点
rootNode.add(childNode3);                                                            // 为根节点添加一级子节点
DefaultTreeModel treeModel = new DefaultTreeModel(rootNode);                         // 使用根节点创建树模型
JTree tree = new JTree(treeModel);                                                   // 使用树模型创建树
scrollPane.setViewportView(tree);                                                    // 把树放到滚动面板的视图中
```

这段代码首先创建了一个滚动面板，然后创建了根节点、一级子节点和二级子节点，并把二级子节点添加到一级子节点上，使其成为一级子节点的子节点，接着又把一级子节点添加到根节点上，使其成为根节点的子节点，最后通过根节点创建了一个树模型，并且采用默认方式判断节点是否为叶子节点的树模型创建了一个树，把树添加到滚动面板的视图中，如果把滚动面板放到窗体上，将在窗体上显示如图 19.6 所示的树。

图 19.6 设置树模型

请读者看下面的代码，在这个实例中演示了使用默认方式的树模型创建树的方法。

例 19.02 本例利用根节点通过树模型创建了一个树，采用的是使用默认方式判断节点是否为叶子节点树模型。实例的关键代码如下：**（实例位置：光盘\mr\ym\19.02）**

```
public TreeModelDemo() {
    setTitle("使用树模型创建树");
    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);
    JScrollPane scrollPane = new JScrollPane();
```

```
        contentPane.add(scrollPane, BorderLayout.CENTER);
        DefaultMutableTreeNode rootNode = new DefaultMutableTreeNode("图书");          // 创建根节点
        DefaultMutableTreeNode childNode1 = new DefaultMutableTreeNode("编程类图书");// 创建一级子节点
        DefaultMutableTreeNode childNode11 = new DefaultMutableTreeNode("Java 类图书");// 创建二级子节点
        DefaultMutableTreeNode childNode12 = new DefaultMutableTreeNode("VC 类图书");  // 创建二级子节点
        DefaultMutableTreeNode childNode13 = new DefaultMutableTreeNode("C#类图书");   // 创建二级子节点
        childNode1.add(childNode11);                                                   // 为一级子节点添加二级子节点
        childNode1.add(childNode12);                                                   // 为一级子节点添加二级子节点
        childNode1.add(childNode13);                                                   // 为一级子节点添加二级子节点
        DefaultMutableTreeNode childNode2 = new DefaultMutableTreeNode("图像类图书");  // 创建一级子节点
        DefaultMutableTreeNode childNode21 = new DefaultMutableTreeNode("Photoshop 类图书"); // 创建二级子节点
        DefaultMutableTreeNode childNode22 = new DefaultMutableTreeNode("CAD 类图书");     // 创建二级子节点
        DefaultMutableTreeNode childNode23 = new DefaultMutableTreeNode("3d Max 类图书");  // 创建二级子节点
        childNode2.add(childNode21);                                                   // 为一级子节点添加二级子节点
        childNode2.add(childNode22);                                                   // 为一级子节点添加二级子节点
        childNode2.add(childNode23);                                                   // 为一级子节点添加二级子节点
        rootNode.add(childNode1)                                                       // 为根节点添加一级子节点
        rootNode.add(childNode2);                                                      // 为根节点添加一级子节点
        DefaultTreeModel treeModel = new DefaultTreeModel(rootNode);                  // 使用根节点创建树模型
        JTree tree = new JTree(treeModel);
        scrollPane.setViewportView(tree);
    }
```

实例的运行效果如图 19.7 所示。

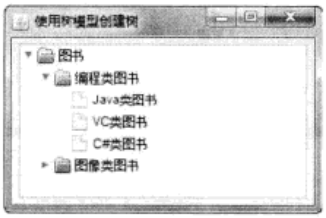


图 19.7　实例运行效果

例 19.03　本例利用根节点通过树模型创建了一个树，采用的是使用默认方式判断节点是否为叶子节点树模型。实例的关键代码如下：**（实例位置：光盘\mr\ym\19.03）**

```
    public TreeModelDemo() {
        setTitle("使用树模型创建树");
        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);
        JScrollPane scrollPane = new JScrollPane();
        contentPane.add(scrollPane, BorderLayout.CENTER);
        DefaultMutableTreeNode rootNode = new DefaultMutableTreeNode("中国");          // 创建根节点
        DefaultMutableTreeNode childNode1 = new DefaultMutableTreeNode("吉林省");      // 创建一级子节点
        DefaultMutableTreeNode childNode11 = new DefaultMutableTreeNode("长春市");     // 创建二级子节点
        DefaultMutableTreeNode childNode12 = new DefaultMutableTreeNode("吉林市", false);
                                                                                       // 创建不允许有子节点的二级子节点
```

```
        DefaultMutableTreeNode childNode13 = new DefaultMutableTreeNode("四平市", true);
                                                                    // 创建允许有子节点的二级子节点
        childNode1.add(childNode11);                                // 为一级子节点添加二级子节点
        childNode1.add(childNode12);                                // 为一级子节点添加二级子节点
        childNode1.add(childNode13);                                // 为一级子节点添加二级子节点
        DefaultMutableTreeNode childNode2 = new DefaultMutableTreeNode("辽宁省");     // 创建一级子节点
        DefaultMutableTreeNode childNode21 = new DefaultMutableTreeNode("沈阳市");    // 创建二级子节点
        DefaultMutableTreeNode childNode22 = new DefaultMutableTreeNode("铁岭市");    // 创建二级子节点
        childNode2.add(childNode21);                                // 为一级子节点添加二级子节点
        childNode2.add(childNode22);                                // 为一级子节点添加二级子节点
        rootNode.add(childNode1);                                   // 为根节点添加一级子节点
        rootNode.add(childNode2);                                   // 为根节点添加一级子节点
        DefaultTreeModel treeModel = new DefaultTreeModel(rootNode);    // 使用根节点创建树模型
        JTree tree = new JTree(treeModel);
        scrollPane.setViewportView(tree);
    }
```

运行本例，将得到如图 19.8 所示的窗体，窗体中的树是采用非默认方式判断节点的，这个树中名称为“吉林市”的节点不允许有子节点，即该节点为叶子节点，而“长春市”节点、“四平市”节点、“沈阳市”节点和“铁岭市”节点虽然没有子节点，但这 4 个节点却允许有子节点。

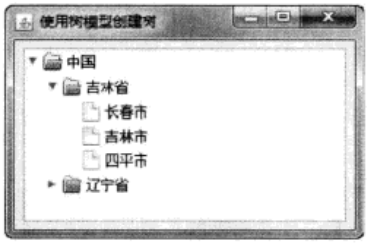


图 19.8　实例运行效果

说明： 上面代码在使用 DefaultTreeModel 类的构造方法 DefaultTreeModel(TreeNode root,boolean asksAllowsChildren)创建树模型时，把第二个入口参数指定为 true 表示允许有子节点，因此“长春市”、“四平市”、“沈阳市”和“铁岭市”4 个节点虽然没有子节点，但显示的却是允许有子节点的图标。在创建二级子节点“吉林市”时，使用了 DefaultMutableTreeNode 类的构造方法 DefaultMutableTreeNode(Object userObject,boolean allowsChildren)，把第二个入口参数指定为 false，表示不允许有子节点，因此显示的是不允许有子节点的叶子节点图标。

19.2.3　维护树的模型

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：

```
public void insertNodeInto(MutableTreeNode newChild,MutableTreeNode parent,int index)
```

参数说明：newChild 是新插入的节点，parent 是插入节点的父节点，index 是插入的位置。

例如，假设要为节点 parentNode 添加一个子节点 treeNode，当前在父节点 parentNode 中已经存在 6 个子节点，现在要将该节点添加到其所有子节点之后，若父节点 parentNode 为树模型 treeModel 中的一个节点，则向父节点 parentNode 的所有子节点的最后添加子节点 treeNode 的典型代码如下：

```
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。

例如，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。

请读者看下面的代码，在这个实例中演示了如何维护树模型。

例 19.04　本例通过维护树模型，实现了一个对中国各省份以及各省份中城市的维护操作。实例代码如下：**（实例位置：光盘\mr\ym\19.04）**

```
public class EditableTreeModeDemo extends JFrame {
    private static final long serialVersionUID = 546598023500487736L;
    private JPanel contentPane;
    private JTextField textField;
    private JTree tree;
    private DefaultTreeModel treeModel;
    public static void main(String[] args) {
        try {
            UIManager.setLookAndFeel("com.sun.java.swing.plaf.nimbus.NimbusLookAndFeel");
        } catch (Throwable e) {
            e.printStackTrace();
        }
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                try {
                    EditableTreeModeDemo frame = new EditableTreeModeDemo();
                    frame.setVisible(true);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }
    public EditableTreeModeDemo() {
```

```
        setTitle("维护树结构");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setBounds(100, 100, 350, 200);
        contentPane = new JPanel();
        contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
        contentPane.setLayout(new BorderLayout(0, 0));
        setContentPane(contentPane);
        JPanel panel = new JPanel();
        contentPane.add(panel, BorderLayout.SOUTH);
        textField = new JTextField();
        panel.add(textField);
        textField.setColumns(10);
        JButton button1 = new JButton("增加");
        button1.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                do_button1_actionPerformed(e);
            }
        });
        panel.add(button1);
        JButton button2 = new JButton("修改");
        button2.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                do_button2_actionPerformed(e);
            }
        });
        panel.add(button2);
        JButton button3 = new JButton("删除");
        button3.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                do_button3_actionPerformed(e);
            }
        });
        panel.add(button3);
        JScrollPane scrollPane = new JScrollPane();
        contentPane.add(scrollPane, BorderLayout.CENTER);
        DefaultMutableTreeNode rootNode = new DefaultMutableTreeNode("中国");      // 创建根节点
        DefaultMutableTreeNode childNode1 = new DefaultMutableTreeNode("黑龙江省"); // 创建一级子节点
    DefaultMutableTreeNode childNode11 = new DefaultMutableTreeNode("哈尔滨市"); // 创建二级子节点
    DefaultMutableTreeNode childNode12 = new DefaultMutableTreeNode("齐齐哈尔市"); // 创建二级子节点
     DefaultMutableTreeNode childNode13 = new DefaultMutableTreeNode("大庆市"); // 创建二级子节点
     childNode1.add(childNode11);                                              // 为一级子节点添加二级子节点
     childNode1.add(childNode12);                                              // 为一级子节点添加二级子节点
     childNode1.add(childNode13);                                              // 为一级子节点添加二级子节点
     DefaultMutableTreeNode childNode2 = new DefaultMutableTreeNode("辽宁省");// 创建一级子节点
     DefaultMutableTreeNode childNode21 = new DefaultMutableTreeNode("沈阳市"); // 创建二级子节点
    DefaultMutableTreeNode childNode22 = new DefaultMutableTreeNode("鞍山市"); // 创建二级子节点
     childNode2.add(childNode21);                                              // 为一级子节点添加二级子节点
     childNode2.add(childNode22);                                              // 为一级子节点添加二级子节点
     rootNode.add(childNode1);                                                 // 为根节点添加一级子节点
```

```
            rootNode.add(childNode2);                                    // 为根节点添加一级子节点
            treeModel = new DefaultTreeModel(rootNode);
            tree = new JTree(treeModel);
            scrollPane.setViewportView(tree);
        }
        protected void do_button1_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
                                                                         // 获得选中的父节点
            DefaultMutableTreeNode parentNode = (DefaultMutableTreeNode) selectionPath.getLastPathComponent();
            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
        }
        protected void do_button2_actionPerformed(ActionEvent e) {
            TreePath selectionPath = tree.getSelectionPath();            // 获得选中的欲修改节点的路径
                                                                         // 获得选中的欲修改节点
            DefaultMutableTreeNode node = (DefaultMutableTreeNode) selectionPath.getLastPathComponent();
            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
        }
        protected void do_button3_actionPerformed(ActionEvent e) {
                                                                         // 获得选中的欲删除节点
            DefaultMutableTreeNode node = (DefaultMutableTreeNode) tree.getLastSelectedPathComponent();
            // 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
                tree.setSelectionPath(new TreePath(nextSelectedNode.getPath())); // 选中节点
            }
        }
}
```

实例的运行效果如图 19.9 所示。

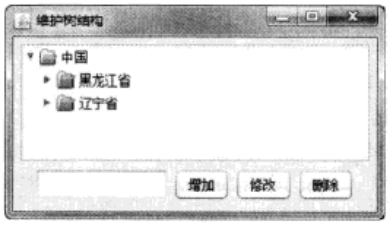


图 19.9　实例运行效果

注意：读者在运行本程序时需要先选择一个节点才能进行添加、删除等操作。

19.3 照猫画虎——基本功训练

19.3.1 基本功训练 1——树控件常用遍历方式

视频讲解：光盘\mr\lx\19\树控件常用遍历方式.exe

实例位置：光盘\mr\ym\19.05\src

对于列表和表格控件，可以使用索引进行遍历。对于树控件，根据遍历的算法不同，会得到不同的元素顺序。通常的遍历方式有深度优先遍历和广度优先遍历。本实例将演示如何实现广度优先遍历。

实现过程如下：

（1）创建窗体类 TreeTraverseDemo，并在其上增加树控件、按钮控件等。

（2）监听窗体激活事件，使用其完成向树中增加数据的功能，关键代码如下：

```
protected void do_this_windowActivated(WindowEvent e) {
    root = new DefaultMutableTreeNode("根节点");                                    // 创建根节点
    DefaultMutableTreeNode parent1 = new DefaultMutableTreeNode("父节点 1"); // 创建子节点
    parent1.add(new DefaultMutableTreeNode("子节点 1"));                  // 为“父节点 1”增加“子节点 1”
    parent1.add(new DefaultMutableTreeNode("子节点 2"));                  // 为“父节点 1”增加“子节点 2”
    root.add(parent1);                                                    // 将 parent1 设置为 root 的子节点
    DefaultMutableTreeNode parent2 = new DefaultMutableTreeNode("父节点 2"); // 创建子节点
    parent2.add(new DefaultMutableTreeNode("子节点 3"));                  // 为“父节点 2”增加“子节点 1”
    parent2.add(new DefaultMutableTreeNode("子节点 4"));                  // 为“父节点 2”增加“子节点 1”
    root.add(parent2);                                                    // 将 parent2 设置为 root 的子节点
    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 getNodesInfor()方法，用于根据不同的枚举数据更新文本域内容，关键代码如下：

```
private void getNodesInfor(Enumeration enums) {
    textArea.setText("");                                                 // 清空文本域
    StringBuilder sb = new StringBuilder();                               // 利用 StringBuilder 保存节点信息
    while (enums.hasMoreElements()) {
        sb.append((DefaultMutableTreeNode) enums.nextElement());          // 增加节点信息
        sb.append("\n");                                                  // 增加换行符
    }
    textArea.setText(sb.toString());                                      // 更新文本域信息
}
```

（4）编写方法监听用户单击按钮事件，分别使用两种不同的方式获得树节点枚举。关键代码如下：

```
protected void do_breadthFirstButton_actionPerformed(ActionEvent e) {
    Enumeration enums = root.breadthFirstEnumeration();          // 获得包含广度优先遍历结果的枚举类型
    getNodesInfor(enums);                                        // 显示枚举类型中的数据
}
```

实例的运行效果如图 19.10 所示。

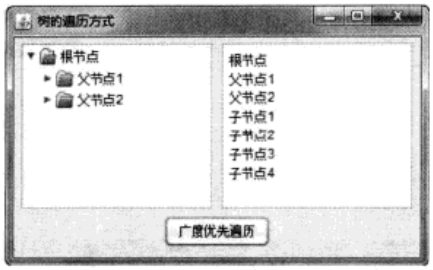


图 19.10　实例运行效果

你问我答：breadthFirstEnumeration()和 depthFirstEnumeration()方法各有什么作用？

breadthFirstEnumeration()方法可以创建并返回一个枚举，该枚举按广度优先的顺序遍历以此节点为根的子树。

depthFirstEnumeration()方法可以创建并返回一个枚举，该枚举按深度优先的顺序遍历以此节点为根的子树。

照猫画虎：模仿本实例实现深度优先遍历。（20 分）（实例位置：光盘\mr\ym\19.05\zmhh）

19.3.2　基本功训练 2——监听节点的选择事件

视频讲解：光盘\mr\lx\19\监听节点的选择事件.exe

实例位置：光盘\mr\ym\19.06\src

树控件不仅仅能用来显示元素，还可以完成各种操作。本实例将根据用户选择的不同节点来更新文本域中的信息。

实现过程如下：

（1）创建窗体类 NodeSelectionDemo，在窗体中增加树控件和文本域控件。

（2）监听窗体激活事件，使用其完成对树控件增加节点的操作，关键代码如下：

```
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）为树控件增加节点选择事件，在用户选择节点的同时更新文本域中的文本，关键代码如下：

```
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）》";
    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);                              // 将文本域设置成 text1
        } else {
            textArea.setText(text2);                              // 将文本域设置成 text2
        }
    }
```

实例的运行效果如图 19.11 所示。

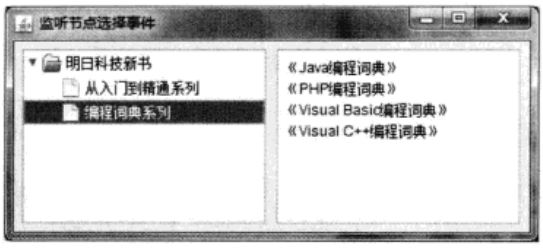


图 19.11　实例运行效果

你问我答：getSelectionPath()方法有什么作用？

该方法用于获得首选节点的路径，此时用户只能选择一个节点。利用选择的路径就可以获得用户选择的元素，从而完成其他操作。

照猫画虎：使用对话框提示用户选择的不同节点。（20 分）（实例位置：光盘\mr\ym\19.06\zmhh）

19.3.3　基本功训练 3——监听节点的展开事件

视频讲解：光盘\mr\lx\19\监听节点的展开事件.exe

实例位置：光盘\mr\ym\19.07\src

除了可以监听节点选择事件，还可以监听节点的展开事件，这是树控件特有的功能。本实例将演示其用法，关键代码如下：

```
public class TreeExpansionDemo extends JFrame {
    private static final long serialVersionUID = -7371422527758123527L;
    private JPanel contentPane;
    private JTextArea textArea;
    public static void main(String[] args) {
        try {
            UIManager.setLookAndFeel("com.sun.java.swing.plaf.nimbus.NimbusLookAndFeel");
        } catch (Throwable e) {
            e.printStackTrace();
        }
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                try {
                    TreeExpansionDemo frame = new TreeExpansionDemo();
                    frame.setVisible(true);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
```

```
        });
    }
    public TreeExpansionDemo() {
        setTitle("\u76D1\u542C\u8282\u70B9\u5C55\u5F00\u4E8B\u4EF6");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setBounds(100, 100, 450, 300);
        contentPane = new JPanel();
        contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
        setContentPane(contentPane);
        contentPane.setLayout(new GridLayout(1, 2, 5, 5));
        JScrollPane scrollPane1 = new JScrollPane();
        contentPane.add(scrollPane1);
        DefaultMutableTreeNode rootNode = new DefaultMutableTreeNode("学科");      // 创建根节点
        DefaultMutableTreeNode childNode1 = new DefaultMutableTreeNode("数学");      // 创建一级子节点
        DefaultMutableTreeNode childNode11 = new DefaultMutableTreeNode("小学数学"); // 创建二级子节点
        DefaultMutableTreeNode childNode12 = new DefaultMutableTreeNode("初中数学"); // 创建二级子节点
        DefaultMutableTreeNode childNode13 = new DefaultMutableTreeNode("高中数学"); // 创建二级子节点
        childNode1.add(childNode11);                                              // 为一级子节点添加二级子节点
        childNode1.add(childNode12);                                              // 为一级子节点添加二级子节点
        childNode1.add(childNode13);                                              // 为一级子节点添加二级子节点
        DefaultMutableTreeNode childNode2 = new DefaultMutableTreeNode("物理");   // 创建一级子节点
        DefaultMutableTreeNode childNode21 = new DefaultMutableTreeNode("初中物理"); // 创建二级子节点
        DefaultMutableTreeNode childNode22 = new DefaultMutableTreeNode("高中物理"); // 创建二级子节点
        DefaultMutableTreeNode childNode23 = new DefaultMutableTreeNode("大学物理"); // 创建二级子节点
        childNode2.add(childNode21);                                              // 为一级子节点添加二级子节点
        childNode2.add(childNode22);                                              // 为一级子节点添加二级子节点
        childNode2.add(childNode23);                                              // 为一级子节点添加二级子节点
        rootNode.add(childNode1);                                                 // 为根节点添加一级子节点
        rootNode.add(childNode2);                                                 // 为根节点添加一级子节点
        DefaultTreeModel model = new DefaultTreeModel(rootNode);                  // 根据根节点创建默认的树模型
        JTree tree = new JTree(model);
        tree.addTreeExpansionListener(new TreeExpansionListener() {
            public void treeCollapsed(TreeExpansionEvent event) {
            }
            public void treeExpanded(TreeExpansionEvent event) {
                do_tree_treeExpanded(event);
            }
        });
        scrollPane1.setViewportView(tree);
        JScrollPane scrollPane2 = new JScrollPane();
        contentPane.add(scrollPane2);
        textArea = new JTextArea();
        scrollPane2.setViewportView(textArea);
    }
    protected void do_tree_treeExpanded(TreeExpansionEvent event) {
        TreePath path = event.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
        textArea.append("节点" + node + "已经被展开！ \n");                        // 在文本域中追加节点信息
    }
}
```

实例的运行效果如图 19.12 所示。

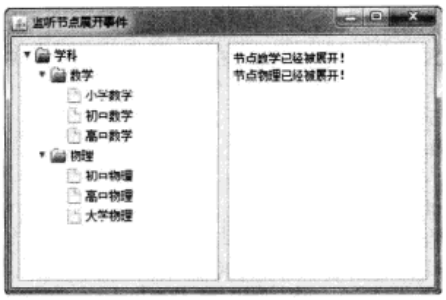


图 19.12　实例运行效果

你问我答：treeExpanded()方法有什么作用？

该方法是在 TreeExpansionListener 类中定义的，它可以完成监听用户展开树节点的动作。在该类中还定义了一个 treeCollapsed()方法，它可以完成监听用户折叠树节点的动作。

照猫画虎：模仿本实例，完成监听用户折叠节点的操作。（20 分）（实例位置：光盘\mr\ym\19.07\zmhh）

19.3.4　基本功训练 4——设置树控件选择模式

视频讲解：光盘\mr\lx\19\设置树控件选择模式.exe

实例位置：光盘\mr\ym\19.08\src

对于树控件而言，也可以像表格一样设置选择模式，如只能选择一行、连续多行或任意多行。本实例将演示如何使用这些选择模式。

实现过程如下：

（1）创建窗体类 NodeSelectionDemo，并增加树控件和组合框控件。

（2）在构造方法中为树控件和组合框控件增加数据，关键代码如下：

```
public NodeSelectionDemo() {
    setTitle("树节点的选择模式");
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setBounds(100, 100, 400, 300);
    contentPane = new JPanel();
    contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
    contentPane.setLayout(new BorderLayout(0, 0));
    setContentPane(contentPane);
    JPanel panel = new JPanel();
    contentPane.add(panel, BorderLayout.NORTH);
    comboBox = new JComboBox();
    comboBox.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            do_comboBox_actionPerformed(e);
        }
    });
    comboBox.addItem("CONTIGUOUS_TREE_SELECTION");                    // 为组合框增加数据
    comboBox.addItem("DISCONTIGUOUS_TREE_SELECTION");
    comboBox.addItem("SINGLE_TREE_SELECTION");
```

```
        panel.add(comboBox);
        JScrollPane scrollPane = new JScrollPane();
        contentPane.add(scrollPane, BorderLayout.CENTER);
        tree = new JTree();
        DefaultMutableTreeNode root = new DefaultMutableTreeNode("明日科技新书");      // 设置根节点
        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
        scrollPane.setViewportView(tree);
    }
```

（3）为组合框增加选择事件监听器，关键代码如下：

```
protected void do_comboBox_actionPerformed(ActionEvent e) {
    if (count++ == 0) {                                                       // 第一次调用该方法时直接退出
        return;
    }
    Map<String, Integer> columnModel = new HashMap<String, Integer>();    // 使用映射保存关系
    columnModel.put("CONTIGUOUS_TREE_SELECTION",
TreeSelectionModel.CONTIGUOUS_TREE_SELECTION);
    columnModel.put("DISCONTIGUOUS_TREE_SELECTION",
TreeSelectionModel.DISCONTIGUOUS_TREE_SELECTION);
    columnModel.put("SINGLE_TREE_SELECTION", TreeSelectionModel.SINGLE_TREE_SELECTION);
    String text = (String) comboBox.getSelectedItem();                        // 获得用户选择的字符串
    tree.getSelectionModel().setSelectionMode(columnModel.get(text));         // 设置树控件选择模式
}
```

实例的运行效果如图 19.13 所示。

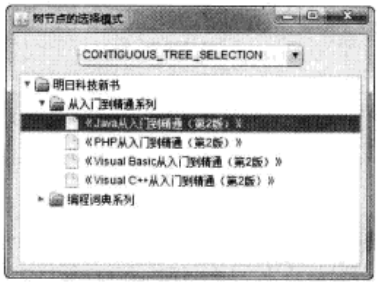


图 19.13　实例运行效果

你问我答：setSelectionMode()方法有什么作用？

该方法可以为树控件设置选择模式，树控件支持的选择模式有 3 种，如表 19.3 所示。

表 19.3　TreeSelectionModel 类定义的常量

常　量　名	作　　用
CONTIGUOUS_TREE_SELECTION	选择只能是连续的
DISCONTIGUOUS_TREE_SELECTION	选择可以包含任何数量的项，这些项不必是连续的
SINGLE_TREE_SELECTION	一次只能选择一个路径

照猫画虎：模仿本实例，使用单选按钮组完成修改选择模式的操作。（20 分）（实例位置：光盘\mr\ym\19.08\zmhh）

19.3.5　基本功训练 5——查看节点的各种状态

视频讲解：光盘\mr\lx\19\查看节点的各种状态.exe

实例位置：光盘\mr\ym\19.09\src

对于树控件而言，不同的节点具有不同的属性，如是否是叶子节点、节点的子节点个数等。本实例将演示如何查看这些属性。

实现过程如下：

（1）创建窗体类 NodeStateViewer，并增加树控件和文本域控件。

（2）在窗体的构造方法中为树控件增加数据和节点选择事件监听。在 do_tree_valueChanged()方法中，完成更改文本域内容的操作，该方法的关键代码如下：

```
protected void do_tree_valueChanged(TreeSelectionEvent e) {
    TreePath path = tree.getSelectionPath();                          // 获得用户选择的路径
    if (path == null) {
        return;
    }
    DefaultMutableTreeNode node = (DefaultMutableTreeNode) path.getLastPathComponent();
                                                                      // 获得用户选择的节点
    StringBuilder sb = new StringBuilder();                           // 使用 StringBuilder 保存字符串
    sb.append("该节点的子节点个数：" + node.getChildCount() + "\n");
    sb.append("该节点在树中的层次：" + node.getLevel() + "\n");
    sb.append("该节点后代的叶子数：" + node.getLeafCount() + "\n");
    textArea.setText(sb.toString());                                  // 更新文本域
}
```

实例的运行效果如图 19.14 所示。

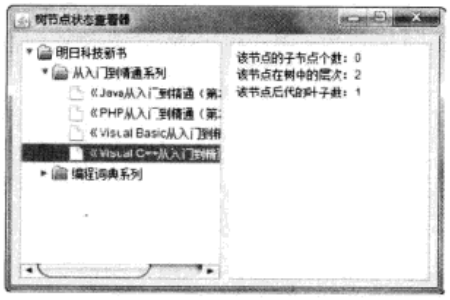


图 19.14　实例运行效果

你问我答：getChildCount()方法有什么作用？

该方法用于获得树中节点具有子节点的个数。如果没有子节点，即该节点是叶子节点，则返回值是零。

照猫画虎：在本实例的基础上增加判断节点是否是根节点和叶子节点的操作。（20 分）（实例位置：光盘\mr\ym\19.09\zmhh）

照猫画虎栏目分数统计：

照猫画虎题目	1	2	3	4	5	总分数	
分数							

19.4　情景应用——拓展与实践

19.4.1　情景应用 1——以树结构显示文件夹下的文件

视频讲解：光盘\mr\lx\19\以树结构显示文件夹下的文件.exe

实例位置：光盘\mr\ym\19.10\src

使用树结构可以很清晰地表示一种层次关系，即根节点的等级最高，叶子节点的等级最低。本实例将使用其显示文件夹下的全部文件。用户需要先选择一个文件夹，然后以这个文件夹作为根节点、文件夹中的文件作为子节点创建树。实例的运行效果如图 19.15 所示。

图 19.15　实例运行效果

实现过程如下：

（1）创建窗体 FileViewer，在窗体中增加按钮、树等控件。

（2）监听用户单击“选择文件夹”按钮事件，这是通过 do_button_actionPerformed()方法实现的。在该方法中，首先创建了一个文件选择器，然后根据用户的选择获得文件夹对象，将这个文件夹作为根节点。使用 listFiles()方法获得文件夹中所有的文件并将其作为子节点。关键代码如下：

```
protected void do_button_actionPerformed(ActionEvent e) {
    JFileChooser chooser = new JFileChooser();                          // 创建文件选择器
    chooser.setFileSelectionMode(JFileChooser.DIRECTORIES_ONLY);// 设置文件选择器仅能选择文件夹
    chooser.setMultiSelectionEnabled(false);                            // 设置文件选择器不支持多重选择
    int result = chooser.showOpenDialog(this);                          // 保存文件选择器选择状态
```

```
    if (result == JFileChooser.APPROVE_OPTION) {
        File directory = chooser.getSelectedFile();                          // 获得用户选择的文件夹
        File[] files = directory.listFiles();                                // 获得文件夹中的全部文件
        // 使用用户选择的文件夹创建根节点
        DefaultMutableTreeNode root = new DefaultMutableTreeNode(directory.getName());
        for (File file : files) {
            if (file.isFile()) {
                root.add(new DefaultMutableTreeNode(file.getName()));        // 向根节点中增加节点
            }
        }
        tree.setModel(new DefaultTreeModel(root));                           // 更新树模型
    }
}
```

DIY：模仿本实例显示文件夹中的所有文本文件。（20 分）（实例位置：光盘\mr\ym\19.10\diy）

19.4.2 情景应用 2——增加图书节点

视频讲解：光盘\mr\lx\19\增加图书节点.exe

实例位置：光盘\mr\ym\19.11\src

Swing 中树控件的使用是非常灵活的，用户可以方便地修改控件的结构。本实例将演示如何为树增加节点，用户需要选择一个节点，单击“增加节点”按钮后，可以在同级增加一个新节点。实例的运行效果如图 19.16 所示。

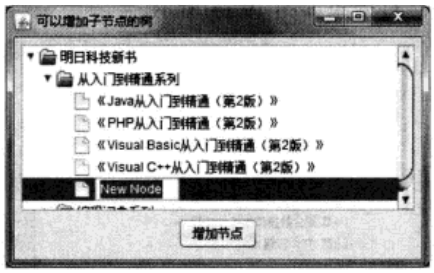


图 19.16 实例运行效果

实现过程如下：

（1）创建窗体 NodeAdditionDemo，在窗体中增加按钮、树等控件。

（2）监听用户单击“增加节点”按钮事件，这是通过 do_button_actionPerformed()方法完成的。在该方法中，首先获得用户选择的节点，如果用户没有选择，则直接退出方法。然后获得树的模型，为其增加新的节点。最后重写绘制这个树结构。关键代码如下：

```
protected void do_button_actionPerformed(ActionEvent e) {
    DefaultMutableTreeNode selectNode = (DefaultMutableTreeNode) tree.getLastSelectedPathComponent();
                                                                         // 获得用户选择的节点
    if (selectNode == null) {                                            // 如果用户没有选择节点则返回
        return;
    }
    DefaultTreeModel model = (DefaultTreeModel) tree.getModel();         // 获得当前树的模型
    DefaultMutableTreeNode newNode = new DefaultMutableTreeNode("New Node");  // 新建节点
```

```
    model.insertNodeInto(newNode, selectNode, selectNode.getChildCount()); // 增加节点
    TreeNode[] nodes = model.getPathToRoot(newNode);          // 向上构建节点的父节点一直到根节点
    TreePath path = new TreePath(nodes);                                  // 创建 TreePath 对象
    tree.scrollPathToVisible(path); // 确保路径中所有的路径控件均展开（最后一个路径控件除外）并滚动
    tree.setSelectionPath(path);                                          // 选择指定路径标识的节点
    tree.startEditingAtPath(path);                                        // 设置新建的节点处于可编辑状态
    tree.repaint();                                                       // 重新绘制树
}
```

你问我答：setSelectionPath()方法有什么作用？

该方法可用于选择指定的节点，方法的参数是该节点所在的路径。

DIY：在本实例的基础上进行增强，如果用户未选择任何节点，则提示用户选择。（20 分）（实例位置：光盘\mr\ym\19.11\diy）

19.4.3　情景应用 3——删除图书节点

视频讲解：光盘\mr\lx\19\删除图书节点.exe

实例位置：光盘\mr\ym\19.12\src

除了可以在创建树时指定树的节点，还可以在程序运行后动态进行修改。本实例将演示如何为树删除节点，用户需要先在树控件上选择要删除的节点，单击“删除节点”按钮来完成删除操作。需要注意的是，删除的节点是不能够恢复的。实例的运行效果如图 19.17 所示。

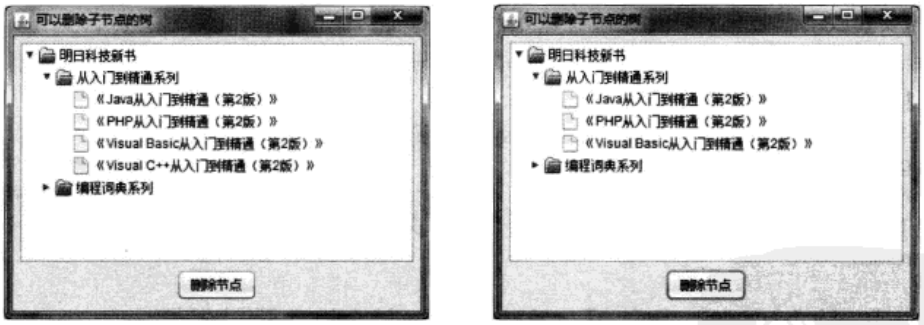


图 19.17　实例运行效果（删除前如左图，删除后如右图）

实现过程如下：

（1）创建窗体 NodeAdditionDemo，在窗体中增加按钮、树等控件。

（2）监听用户单击“删除节点”按钮事件，这是通过 do_button_actionPerformed()方法完成的。在该方法中，首先获得用户选择的节点，如果用户没有选择，则直接退出方法。然后获得树的模型，删除选择的节点。最后重写绘制这个树结构。关键代码如下：

```
protected void do_button_actionPerformed(ActionEvent e) {
    DefaultMutableTreeNode selectNode = (DefaultMutableTreeNode) tree.getLastSelectedPathComponent();
                                                          // 获得用户选择的节点
    if ((selectNode == null) || (selectNode.isRoot())) {     // 如果没有选择节点或选择了根节点则返回
        return;
    }
    DefaultTreeModel model = (DefaultTreeModel) tree.getModel(); // 获得当前树的模型
```

```
        model.removeNodeFromParent(selectNode);                    // 删除节点
        tree.repaint();                                            // 重新绘制树
    }
```

DIY：在本实例的基础上进行增强，如果用户未选择任何节点，则提示用户选择。（20 分）（实例位置：光盘\mr\ym\19.12\diy）

19.4.4 情景应用 4——查找图书节点

视频讲解：光盘\mr\lx\19\查找图书节点.exe

实例位置：光盘\mr\ym\19.13\src

树结构的特点决定了在其中进行人工查找节点是非常麻烦的。为了方便用户使用，通常需要程序员提供查找功能。本实例将演示如何查找树中的指定节点，用户需要输入完整的节点才能进行查询。在查询完成后，会将匹配的那个节点选中。实例的运行效果如图 19.18 所示。

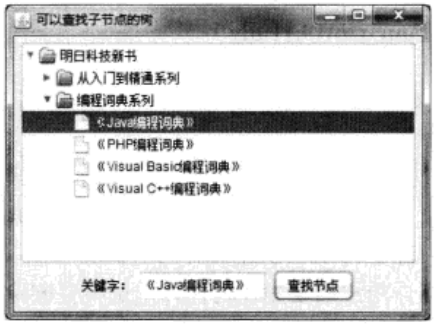


图 19.18 实例运行效果

实现过程如下：

（1）创建窗体 NodeSearchDemo，在窗体中增加按钮、树等控件。

（2）监听用户单击“查找节点”按钮事件，这是通过 do_button_actionPerformed()方法完成的。在该方法中，首先获得用户输入的要查找的节点信息，如果没有输入，则进行提示。然后使用工具方法获得全部树节点，如果其中存在与用户输入的字符串匹配的节点，就将其选中，否则提示未找到。关键代码如下：

```
protected void do_button_actionPerformed(ActionEvent e) {
    String key = textField.getText();                              // 获得用户输入的关键字
    if ((key == null) || (key.isEmpty())) {                        // 如果关键字为空，则提示用户重新输入
        JOptionPane.showMessageDialog(this, "请输入关键字", "", JOptionPane.WARNING_MESSAGE);
        return;
    }
    DefaultTreeModel model = (DefaultTreeModel) tree.getModel(); // 获得树模型
    DefaultMutableTreeNode targetNode = null;
    Enumeration enums = root.breadthFirstEnumeration();            // 获得树的全部节点
    while (enums.hasMoreElements()) {                              // 遍历全部节点进行查找
        DefaultMutableTreeNode tempNode = (DefaultMutableTreeNode) enums.nextElement();
        if ((tempNode.toString()).equals(key)) {
            targetNode = tempNode;
        }
    }
    if (targetNode == null) {                                      // 如果没有找到则进行提示
        JOptionPane.showMessageDialog(this, "未找到需要的结果", "", JOptionPane.WARNING_MESSAGE);
        return;
    } else {                                                       // 如果找到了，则将该节点设置成选择状态
        TreeNode[] nodes = model.getPathToRoot(targetNode);
        TreePath path = new TreePath(nodes);
        tree.scrollPathToVisible(path);
```

```
            tree.setSelectionPath(path);
        }
    }
```

DIY：在本实例的基础上进行增强，实现简单的模糊查询功能。（20 分）（实例位置：光盘\mr\ym\19.13\diy）

19.4.5　情景应用 5——为树节点增加提示信息

视频讲解：光盘\mr\lx\19\为树节点增加提示信息.exe

实例位置：光盘\mr\ym\19.14\src

由于用户屏幕大小有限，对于专业的应用程序，每个像素的浪费都是非常可惜的。对于树结构而言，其本身就很复杂，各个节点的层次关系如果全部展开会非常占用空间。此时可以考虑对节点采用一种简写的方式，然后为每个节点提供提示信息，从而节约空间。本实例将演示其用法，实例的运行效果如图 19.19 所示。

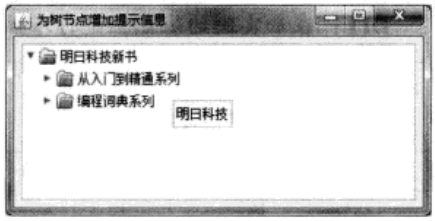


图 19.19　实例运行效果

实现过程如下：

（1）创建窗体 NodeTipDemo，在窗体中增加树控件。

（2）创建 ToolTipNode 类，它继承了 TreeCellRenderer。重写 getTreeCellRendererComponent()方法，在该方法中对于不同的节点为其指定不同的提示信息，这些信息是使用 Map 来保存的。当创建该类对象时，需要指定 Map 类型参数。关键代码如下：

```
private 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(map.get(value));                        // 设置提示信息
        return renderer;
    }
}
```

你问我答：getTreeCellRendererComponent()方法有什么作用？

该方法用于修改节点的样式。方法声明如下：

Component getTreeCellRendererComponent(JTree tree,Object value,boolean selected,boolean expanded, boolean leaf,int row,boolean hasFocus)

参数说明如表 19.4 所示。

表 19.4 getTreeCellRendererComponent()方法的参数说明

参 数 名	作 用
tree	需要渲染的树
value	树的当前节点
selected	节点是否处于被选择状态
expanded	节点是否处于展开状态
leaf	节点是否是叶子
row	节点的索引
hasFocus	节点是否具有焦点

DIY：在本实例的基础上进行增强，修改提示信息的字体为微软雅黑，大小是 16，颜色是红色。（20 分）（实例位置：光盘\mr\ym\19.14\diy）

情景应用 DIY 栏目分数统计：

DIY 题目	1	2	3	4	5	总分数	
分数							

19.5 自 我 测 试

一、选择题（每题 10 分，5 道题）

1．下列哪些是 JTree 类的构造方法？（ ）

A．Tree() B．JTree(TreeModel newModel) C．JTree(TreeNode root) D．JTree(List list)

2．树控件中节点选择模式包括？（ ）

A．CONTIGUOUS_TREE_SELECTION B．DISCONTIGUOUS_TREE_SELECTION

C．SINGLE_TREE_SELECTION D．DISINGLE_TREE_SELECTION

3．TreeModel 是？（ ）

A．接口 B．抽象类 C．内部类 D．其他

4．DefaultMutableTreeNode 类支持的树的常见遍历方式有？（ ）

A．深度优先遍历 B．广度优先遍历 C．长度优先遍历 D．宽度优先遍历

5．树节点的展开和折叠事件定义在哪个接口中？（ ）

A．TreeExpansionListener B．TreeExpandedListener

C．TreeCollapsedListener D．TreeListener

二、填空题（每题 10 分，5 道题）

1．如果需要查看某个节点的子节点个数，则可以使用 DefaultTreeModel 类中的哪个方法？_____

2．DefaultMutableTreeNode 类可以既作为根节点又作为子节点吗？_____

3．DefaultMutableTreeNode 类是线程安全的吗？_____

4．DefaultMutableTreeNode 类提供的深度优先遍历方法名是？_____

5．如果需要自定义树节点，则可以使用 TreeCellRenderer 接口中定义的哪个方法（写成方法名即可）？_____

测试分数统计：

类别	第 1 题	第 2 题	第 3 题	第 4 题	第 5 题
选择题分数					
填空题分数					
总分数					

19.6　行 动 指 南

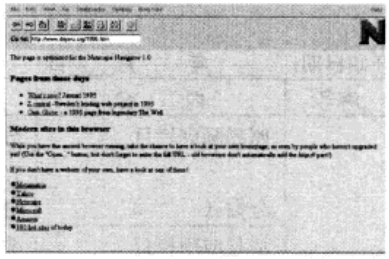
开始日期：_____年____月____日　　结束日期：_____年____月____日

序号	内　容	行 动 指 南	
1	照猫画虎栏目 分数（　　）	分数>75 分	优秀，基本功掌握得不错，加油！
		75 分>分数>50 分	及格，知识掌握得不牢，重新做一遍照猫画虎。
		分数<50 分	请使用光盘中提供的“实战能力测试系统”进行提高训练。
	情景应用栏目 分数（　　）	分数>75 分	优秀，综合应用能力很强。
		75 分>分数>50 分	及格，综合应用能力需提高，再练一遍情景应用。
		分数<50 分	请使用光盘中提供的“实战能力测试系统”进行提高训练。
	自我测试栏目 分数（　　）	分数>75 分	优秀，有成为编程高手的潜质。
		分数<75 分	请使用光盘中提供的“实战能力测试系统”进行提高训练。
	综合评价	反复训练后，以上各项分数都在 75 分以上，方可进入下一堂课学习。	
2	编程实践能力培养：本栏目提供了一些实践题目，对于培养编程能力很有效，要做好。	（1）自定义节点的外观。	
		（2）实现节点的双击修改功能。	
		（3）自定义树节点的图标。	
		（4）查看磁盘中的全部文件。	
3	编程习惯培养：本堂课培养读者在学习开发中记笔记的习惯，把开发中遇到的问题、总结的经验记录下来。		
4	创新能力培养：看看身边有没有可以用编程解决的问题，记录到右边的表格中。根据学习进度，尝试编写解决这些问题的小程序。		

19.7 成功可以复制——因特网的点火人

安德森出生在威斯康星州一个小镇的普通家庭，9 岁开始接触计算机，在图书馆自学 Basic 语言。当升入六年级时，安德森已经创造出一个替他做数学家庭作业的虚拟计算器。七年级时，他编出一个在自家计算机上玩的游戏程序。1993 年，刚刚大学毕业的安德森还没有找到更好的工作，于是就和几个志同道合的朋友一起写 Internet 浏览软件。早期的万维网简陋，只有文本，没有图像、声音和色彩，只是一个基于文本之上，仅仅通过原始界面才能进入的网络。安德森在使用万维网颇感不便的同时，敏锐地察觉到开发一个带有图形、易于客户使用的网页浏览器的潜在市场。经过不断探索和坚持，1994 年 4 月，安德森带领他的工作组开发成功“Navigator”（领航员）浏览器，随后该浏览器的销售在 Internet 上如“凯歌般地行进”，一下就占据了 80%以上的份额。

1995 年 8 月 9 日，互联网时代到来了，成立还不到 16 个月、从未赢利过的 Netscape 公司在纽约上市。投资银行事先估计每股仅能卖 14 美元左右，然而开盘后股价一路飙升，最高时竟达到让全美经纪人都目瞪口呆的 71 美元，2 个小时内 500 万股被抢购一空，这家创始资金只有 400 万美元的小公司一夜之间便成为 20 亿美元的巨人。华尔街日报评论说，通用动力公司花了 43 年才使市值达到 27 亿美元，而网景只花了 1 分钟。年仅 24 岁的安德森也仿佛神话般地从一文不名到拥有 1.74 亿美元身价的“Internet 富翁”。1997 年 7 月的美国《旗帜》周刊更把安德森们称为“无限制资本家”，他们正从根本上重塑着美国社会，推动着从工业经济向信息经济的过渡。



Navigator 1.0 浏览器

经典语录

在互联网的光芒下，操作系统只不过是一套“漏洞重重的设备驱动器”。

深度评价

马克·安德森的成功启示我们，作为编程开发人员，要善于从生活、工作中发现机会，并坚持把一个小机会做到完美，就可以成就大事业。编程知识不在于多，而在于把学到的知识应用于实践，改善我们工作、学习中的低效率或不便。

第20堂课

其他高级控件

（视频讲解：326 分钟）

Swing 提供了一些高级控件，如文件选择对话框、工具栏、菜单、分割面板、选项卡面板、桌面面板、内部窗体和系统托盘，通过对这些控件的使用，可以设计出更加人性化的界面，以提高应用程序的灵活性。

学习摘要：

- 使用文件选择对话框
- 文件过滤器的使用
- 工具栏的使用方法
- 菜单的使用方法
- 弹出式菜单的使用方法
- 为菜单项设置图标的方法
- 如何使用分割面板
- 选项卡面板的使用
- 桌面面板和内部窗体
- 系统托盘的使用

20.1 文件选择对话框

在进行应用程序开发时，经常需要选择文件，如选择照片文件、从指定的文件中读入数据、选择指定类型的文件等，这些操作可以通过 javax.swing.JFileChooser 类来实现，本节将详细讲解文件选择对话框的创建以及如何为文件选择对话框添加过滤器。

20.1.1 创建文件选择对话框

JFileChooser 类为用户选择文件提供了一种简单的机制。通过该类创建的对象是一个文件选择对话框，可以方便地选择文件。该类常用的构造方法如表 20.1 所示。

表 20.1 JFileChooser 类的常用构造方法及说明

构 造 方 法	说 明
JFileChooser()	构造一个指向用户默认目录的文件选择对话框
JFileChooser(File f)	使用给定的 File 作为路径来构造一个文件选择对话框
JFileChooser(String dir)	构造一个使用给定路径的文件选择对话框

JFileChooser 类还提供了两个比较重要的字段 APPROVE_OPTION 和 CANCEL_OPTION，分别用于判断用户在文件选择对话框中是单击了“打开”按钮还是单击了“取消”按钮，如果单击的是“打开”按钮，表示用户选择了文件，可以对文件进行操作；如果单击的是“取消”按钮，表示用户没有选择文件。

JFileChooser 类的实例调用 showOpenDialog(Component parent)方法，可以打开一个文件选择对话框，该方法返回一个整数值，用于判断用户是否选择了文件，如果选择了文件就可以通过 getSelectFile()方法获得所选文件的 File 对象，通过 File 对象就可以获得文件的相关属性，如文件名、路径、文件大小等信息。例如：

```
JFileChooser fileChooser = new JFileChooser();                          // 创建对话框
int returnValue = fileChooser.showOpenDialog(getContentPane());         // 打开对话框
if (returnValue == JFileCHooser.APPROVE_OPTION) {                       // 判断条件
    File file = fileChooser.getSelectedFile();                          // 获得文件对象
    String name = file.getName();                                       // 获得文件名
    String path = file.getAbsolutePath();                               // 获得绝对路径
    long len = file.length();                                           // 获得文件的大小
}
```

这段代码使用 JFileChooser 类创建了文件选择对话框的实例 fileChooser，并调用 showOpenDialog()方法打开文件选择对话框，然后用字段 APPROVE_OPTION 判断用户是否选择了文件，如果选择了文件，就可以获得文件的 File 对象，进而获得所选文件的文件名、文件的绝对路径和文件的大小。

请读者看下面的代码，在这个实例中演示了文件选择对话框的使用。

例 20.01 在项目中创建 JFileChooserDemo 类，该类继承自 JFrame 类成为窗体类。在该类中创建标签控件、文本框控件和按钮控件，并将它们添加到窗体界面中，当用户单击“选择文件”按钮时，将打开文件选择对话框，从对话框中选择文件，单击“打开”按钮，将在文本框中显示所选文件的绝对路径。实例的关键代码如下：**（实例位置：光盘\mr\ym\20.01）**

```
protected void do_button_actionPerformed(ActionEvent e) {
    JFileChooser chooser = new JFileChooser();                          // 创建文件选择器
```

```
        chooser.setFileSelectionMode(JFileChooser.FILES_ONLY);          // 设置仅能选择文件
        chooser.setMultiSelectionEnabled(false);                        // 禁用多重选择
        int result = chooser.showOpenDialog(this);                      // 获得文件选择器状态
        if (result == JFileChooser.APPROVE_OPTION) {
            File file = chooser.getSelectedFile();                      // 获得用户选择的文件
            textField.setText(file.getAbsolutePath());                  // 更新文本框中内容
        }
    }
```

实例的运行效果如图 20.1 所示，单击“选择文件”按钮，效果如图 20.2 所示。

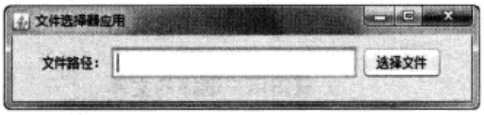


图 20.1　实例运行效果

图 20.2　单击“选择文件”按钮后的效果

20.1.2　添加文件过滤器

JFileChooser 类提供了 setFileFilter(FileFilter filter)方法，通过该方法可以为文件选择对话框设置过滤器，使其只显示指定类型的文件，该方法的参数类型 javax.swing.filechooser.FileFilter 是一个抽象类，实现该类可以对文件进行过滤。该类中的方法如表 20.2 所示。

表 20.2　FileFilter 类的方法及说明

方　法	说　明
public abstract boolean accept(File file)	用于对文件进行过滤
public abstract String getDescription()	返回文件选择对话框中文件类型的描述信息

类 javax.swing.filechooser.FileNameExtensionFilter 实现了 FileFilter 类，FileNameExtensionFilter 类只有一个构造方法，通过该类的实例可以实现文件的过滤操作。该方法声明如下：

```
public FileNameExtensionFilter(String description,String... extensions)
```

参数说明：description 是过滤器的文本描述，可以为 null。extensions 是接受的文件扩展名。

例如：

```
JFileChooser fileChooser = new JFileChooser();
    // 创建对话框
FileFilter filter = new FileNameExtensionFilter("图像文件(*.bmp;*.gif;*.jpg;*.jpeg) ", "bmp", "gif", "jpg", "jpeg");
    // 创建文件过滤器
fileChooser.setFileFilter(filter);                                      // 设置文件过滤器
int returnValue = fileChooser.showOpenDialog(getContentPane());        // 显示对话框
```

这段代码使用 JFileChooser 类创建了文件选择对话框的实例 fileChooser，使用 FileFilter 类的实现类 FileNameExtensionFilter 创建了文件过滤器 filter，并用 JFileChooser 类的 setFileFilter(FileFilter filter)方法为

文件选择对话框的实例 fileChooser 设置了文件过滤器，使打开的文件选择对话框只显示 bmp、gif、jpg 和 jpeg 类型的文件。

请读者看下面的代码，在这个实例中演示了如何使用文件过滤器在文件选择对话框中显示指定类型的文件。

例 20.02 在项目中创建 FileFilterFrame 类，该类继承自 JFrame 类成为窗体类。在该类中创建标签控件、文本框控件和按钮控件，并将它们添加到窗体界面中，当用户单击“选择文件”按钮时，将打开文件选择对话框，并且在文件选择对话框中只显示 bmp、gif、jpg 和 jpeg 类型的图像文件，在对话框中选择图像文件，单击“打开”按钮，将在文本框中显示所选图像文件的名称和大小。实例的关键代码如下：**（实例位置：光盘\mr\ym\20.02）**

```
protected void do_button_actionPerformed(ActionEvent e) {
    JFileChooser chooser = new JFileChooser();                          // 创建文件选择器
    FileFilter filter = new FileNameExtensionFilter("图像文件(*.bmp;*.gif;*.jpg;*.jpeg)", "bmp", "gif", "jpg", "jpeg");
    chooser.setFileFilter(filter);                                      // 为文件选择对话框设置文件过滤器
    chooser.setFileSelectionMode(JFileChooser.FILES_ONLY);              // 设置仅能选择文件
    chooser.setMultiSelectionEnabled(false);                            // 禁用多重选择
    int result = chooser.showOpenDialog(this);                          // 获得文件选择器状态
    if (result == JFileChooser.APPROVE_OPTION) {
        File file = chooser.getSelectedFile();                          // 获得用户选择的文件
        textField.setText(file.getAbsolutePath());                      // 更新文本框中内容
    }
}
```

实例的运行效果如图 20.3 所示，单击“选择文件”按钮，效果如图 20.4 所示。

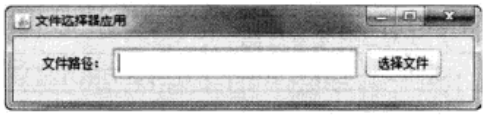


图 20.3 实例运行效果

图 20.4 单击“选择文件”按钮后的效果

20.2 工 具 栏

工具栏是我们所熟悉的，在很多应用程序或软件中都提供了工具栏，并把一些常用命令和操作放到工具栏上，如果能够熟练使用工具栏，可以大大提高工作效率，Swing 提供的 javax.swing.JToolBar 类创建的对象就是一个工具栏，本节将详细讲解工具栏的创建以及如何向工具栏中添加工具按钮。

20.2.1 创建工具栏

JToolBar 类创建的对象是一个工具栏，工具栏放置在窗体默认布局的顶部位置，可以用鼠标将其拖动到

窗体的左侧、右侧或底部位置，也可以通过鼠标将工具栏拖到窗体以外的区域，使其成为浮动工具栏。该类常用的构造方法如表 20.3 所示。

表 20.3　JToolBar 类的常用构造方法及说明

构 造 方 法	说　明
JToolBar()	构造一个默认位置为水平方向的工具栏
JToolBar(String name)	构造一个默认位置为水平方向、在浮动状态时具有指定标题的工具栏

例如：

```
JToolBar toolBar1 = new JToolBar();                    // 创建默认位置为水平方向的工具栏
JToolBar toolBar2 = new JToolBar("文件操作");          // 创建水平方向且有标题的工具栏
```

这段代码使用 JToolBar 类创建了两个工具栏，第一行代码创建的是默认位置为水平方向且没有标题的工具栏，第二行代码创建的是默认位置为水平方向且标题为“文件操作”的工具栏。

20.2.2　添加工具按钮

JToolBar 类也提供了操作工具栏的方法，可以实现工具栏的设置，如添加工具按钮、添加分隔符等。该类的常用方法如表 20.4 所示。

表 20.4　JToolBar 类的常用方法及说明

方　法	说　明
Component add(Component c)	在工具栏的末尾添加参数指定的控件，如按钮、复选框、单选按钮等
void addSeparator()	将默认大小的分隔符添加到工具栏的末尾
void addSeparator(Dimension size)	将指定大小的分隔符添加到工具栏的末尾

例如：

```
JToolBar toolBar1 = new JToolBar();                    // 创建默认位置为水平方向的工具栏
JButton btn_open = new JButton("打开");                // 创建“打开”按钮
JButton btn_exit = new JButton("退出");                // 创建“退出”按钮
toolBar1.add(btn_open);                                // 把“打开”按钮添加到工具栏末尾
toolBar1.addSeparator();                               // 在工具栏的末尾添加默认大小的分隔符
toolBar1.add(btn_exit);                                // 把“退出”按钮添加到工具栏末尾
```

这段代码创建了一个工具栏和两个按钮“打开”和“退出”，然后把“打开”按钮添加到工具栏上，又在“打开”按钮后方添加一个默认大小的分隔符，最后在工具栏的末尾添加一个“退出”按钮。

本节将以实例说明工具栏的创建、向工具栏添加工具按钮以及为工具按钮添加事件监听器。

例 20.03　在项目中创建 JToolBarDemo 类，该类继承自 JFrame 类成为窗体类。在该类中创建工具栏并向工具栏添加工具按钮，然后为工具按钮添加事件监听器实现事件响应操作。实例的关键代码如下：**（实例位置：光盘\mr\ym\20.03）**

```
public JToolBarDemo() {
    setTitle("创建工具栏");                                  // 设置窗体标题
    setBounds(100, 100, 240, 200);                          // 设置窗体位置和大小
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);         // 设置窗体默认关闭方式
    JToolBar toolBar = new JToolBar("文件操作");             // 创建有标题的工具栏对象
    final JButton btn_new = new JButton("新建");             // 创建工具按钮
    final JButton btn_open = new JButton("打开");            // 创建工具按钮
```

```
    final JButton btn_save = new JButton("保存");                    // 创建工具按钮
    final JButton btn_exit = new JButton("退出");                    // 创建工具按钮
    btn_exit.addActionListener(new ActionListener() {                // 为“退出”按钮添加动作监听器
        public void actionPerformed(final ActionEvent arg0) {        // 响应动作事件
            System.exit(0);                                          // 退出系统
        }
    });
    toolBar.add(btn_new);                                            // 在工具栏上添加工具按钮
    toolBar.add(btn_open);                                           // 在工具栏上添加工具按钮
    toolBar.add(btn_save);                                           // 在工具栏上添加工具按钮
    toolBar.addSeparator();                                          // 在工具栏上添加分隔线
    toolBar.add(btn_exit);                                           // 在工具栏上添加工具按钮
    getContentPane().add(toolBar, BorderLayout.NORTH);               // 把工具栏添加到窗体的顶部位置
}
```

实例的运行效果如图 20.5 所示。

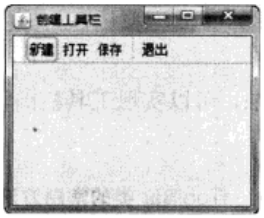


图 20.5　实例运行效果

注意：单击工具栏上的“新建”、“打开”和“保存”按钮时没有任何反应，是由于在程序中没有给这 3 个按钮添加事件监听器，而只给“退出”按钮添加了事件监听器和实现功能的程序代码，所以单击“退出”按钮可以退出系统。

20.3　菜单栏与弹出菜单

菜单栏和弹出菜单都是菜单，在应用程序中，菜单是必不可少的界面元素，几乎每一个应用程序都需要菜单，而单击菜单中的菜单项就可以执行操作命令，简单易用、方便快捷。

20.3.1　创建菜单栏

使用类javax.swing.JMenuBar创建的对象是一个菜单栏，菜单栏只有一个无参数的构造方法JMenuBar()，用于构造一个菜单栏。例如：

```
JMenuBar menuBar = new JMenuBar();                    // 创建一个菜单栏
```

这段代码使用 JMenuBar 类创建了一个菜单栏。

JMenuBar 类用于创建菜单栏，创建了菜单栏就可以调用该类的 add(JMenu menu)方法把菜单添加到菜单栏上。例如：

```
JMenu menu = new JMenu("文件");                       // 创建一个“文件”菜单
JMenuBar menuBar = new JMenuBar();                    // 创建一个菜单栏
menuBar.add(menu);                                    // 把菜单添加到菜单栏上
```

这段代码首先创建了一个菜单，然后创建了一个菜单栏，最后使用 JMenuBar 类的 add 方法把菜单添加

到菜单栏上。

📖 **说明：这里使用的菜单类 JMenu 将在 20.3.2 小节进行讲解。**

20.3.2　添加菜单和菜单项

1．菜单

菜单是图形化用户界面的重要组成部分，用户通过菜单可以方便地与应用程序进行交互，类 javax.swing.JMenu 创建的对象是一个菜单。该类的常用构造方法如表 20.5 所示。

表 20.5　JMenu 类的常用构造方法

构 造 方 法	说　明
JMenu()	构造一个没有初始文本的新菜单
JMenu(String s)	构造一个初始文本为指定字符串的新菜单

例如：

```
JMenuBar menuBar = new JMenuBar();                    // 创建菜单栏
JMenu mn_file = new JMenu("文件");                    // 创建“文件”菜单
JMenu mn_edit = new JMenu("编辑");                    // 创建“编辑”菜单
menuBar.add(mn_file);                                 // 把“文件”菜单添加到菜单栏上
menuBar.add(mn_edit);                                 // 把“编辑”菜单添加到菜单栏上
```

这段代码创建了一个菜单栏和两个菜单“文件”和“编辑”，然后把“文件”菜单和“编辑”菜单添加到菜单栏上。

类 JMenu 创建的对象是一个菜单，菜单用于添加菜单项和子菜单，实现对菜单项的分类管理。该类的常用方法如表 20.6 所示。

表 20.6　JMenu 类的常用方法

方　法	说　明
add(JMenuItem menuItem)	将某个菜单项追加到此菜单的末尾
insert(JMenuItem menuItem,int pos)	在给定位置插入指定的菜单项
addSeparator()	将新分隔符追加到菜单的末尾
insertSeparator(int pos)	在指定的位置插入分隔符

2．菜单项

类 javax.swing.JMenuItem 创建的对象是一个菜单项，当用户单击某个菜单项时，将触发该菜单项的动作事件，进而完成该菜单项的功能。该类的常用构造方法如表 20.7 所示。

表 20.7　JMenuItem 类的常用构造方法

构 造 方 法	说　明
JMenuItem()	创建不带文本和图标的菜单项
JmenuItem(String text)	创建带有指定文本的菜单项
JmenuItem(Icon icon)	创建带有指定图标的菜单项
JMenuItem(String text,Icon icon)	创建带有指定文本和图标的菜单项

使用 JMenuItem 类的 setIcon(Icon icon)方法，可以为菜单项添加图标。例如：

```
JMenuBar menuBar = new JMenuBar();                          // 创建菜单栏
JMenu mn_file = new JMenu("文件");                          // 创建“文件”菜单
JMenuItem mi_new = new JMenuItem("新建");                   // 创建“新建”菜单项
JMenuItem mi_exit = new JMenuItem("退出");                  // 创建“退出”菜单项
URL url = this.getClass().getResource("exit.png");          // 获取图片所在的 URL
Icon icon = new ImageIcon(url);                             // 实例化 Icon 对象
mi_exit.setIcon(icon);                                      // 为“退出”菜单项添加图标
mn_file.add(mi_new);                                        // 添加“新建”菜单项到“文件”菜单
mn_file.add(mi_exit);                                       // 添加“退出”菜单项到“文件”菜单
menuBar.add(mn_file);                                       // 把“文件”菜单添加到菜单栏上
```

这段代码创建了一个菜单栏和一个“文件”菜单，然后创建两个菜单项“新建”和“退出”，并创建了图标文件的 URL，然后创建了图标对象，并为“退出”菜单项添加了图标，再把“新建”和“退出”两个菜单项添加到“文件”菜单上，最后把“文件”菜单添加到菜单栏上。

3．菜单项事件

在应用程序中单击某个菜单项，通常会执行某个操作，这需要为菜单项添加事件监听器，菜单项通常使用动作事件监听器来捕获动作事件，进而实现相应的业务逻辑。为菜单项添加动作事件监听器可以使用 addActionListener(ActionListener actionListener)方法来实现。例如：

```
JMenuBar menuBar = new JMenuBar();                          // 创建菜单栏
JMenu mn_file = new JMenu("文件");                          // 创建“文件”菜单
JMenuItem mi_exit = new JMenuItem("退出");                  // 创建“退出”菜单项
mi_exit.addActionListener(new ActionListener() {            // 添加事件监听器
    public void actionPerformed(final ActionEvent arg0) {
        System.exit(0);                                     // 退出系统
    }
});
mn_file.add(mi_exit);                                       // 把“退出”菜单项添加到“文件”菜单上
menuBar.add(mn_file);                                       // 把“文件”菜单添加到菜单栏上
```

这段代码创建了一个菜单栏和一个“文件”菜单，然后创建一个菜单项“退出”，并为“退出”菜单项添加了动作事件监听器，作用是运行程序时单击“退出”菜单项将退出系统，再把“退出”菜单项添加到“文件”菜单上，最后把“文件”菜单添加到菜单栏上。

本节将以实例说明菜单栏、菜单、菜单项以及如何为菜单项添加图标和事件监听器。

例 20.04 在项目中创建 JMenuDemo 类，该类继承自 JFrame 类成为窗体类。在该类中创建菜单栏，并向菜单栏添加“文件”和“编辑”两个菜单，然后添加菜单项，并为“退出”菜单项添加动作事件监听器实现动作事件的响应操作。实例的关键代码如下：**（实例位置：光盘\mr\ym\20.04）**

```
public JMenuDemo() {
    setTitle("创建菜单");
    setBounds(100, 100, 240, 160);                          // 设置窗体的位置和大小
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);         // 设置窗体默认关闭方式
    JMenuBar menuBar = new JMenuBar();                      // 创建菜单栏
    JMenu mn_file = new JMenu("文件");                      // 创建“文件”菜单
    JMenu mn_edit = new JMenu("编辑");                      // 创建“编辑”菜单
    JMenuItem mi_new = new JMenuItem("新建");               // 创建“新建”菜单项
    JMenuItem mi_open = new JMenuItem("打开");              // 创建“打开”菜单项
    JMenuItem mi_save = new JMenuItem("保存");              // 创建“保存”菜单项
```

```
        JMenuItem mi_exit = new JMenuItem("退出");                    // 创建“退出”菜单项
        mi_exit.setIcon(new ImageIcon("src/images/exit.png"));        // 为“退出”菜单项添加图标
        mi_exit.addActionListener(new ActionListener() {              // 为“退出”菜单项添加动作事件监听器
            public void actionPerformed(final ActionEvent arg0) {
                System.exit(0);                                       // 退出系统
            }
        });
        JMenuItem mi_copy = new JMenuItem("复制");                    // 创建“复制”菜单项
        JMenuItem mi_paste = new JMenuItem("粘贴");                   // 创建“粘贴”菜单项
        mn_file.add(mi_new);                                          // 把“新建”菜单项添加到“文件”菜单上
        mn_file.add(mi_open);                                         // 把“打开”菜单项添加到“文件”菜单上
        mn_file.add(mi_save);                                         // 把“保存”菜单项添加到“文件”菜单上
        mn_file.addSeparator();                                       // 在“保存”和“退出”菜单项间加分隔符
        mn_file.add(mi_exit);                                         // 把“退出”菜单项添加到“文件”菜单上
        mn_edit.add(mi_copy);                                         // 把“复制”菜单项添加到“编辑”菜单上
        mn_edit.add(mi_paste);                                        // 把“粘贴”菜单项添加到“编辑”菜单上
        menuBar.add(mn_file);                                         // 把“文件”菜单添加到菜单栏上
        menuBar.add(mn_edit);                                         // 把“编辑”菜单添加到菜单栏上
        setJMenuBar(menuBar);                                         // 把菜单栏添加到窗体上
    }
```

实例的运行效果如图 20.6 所示。

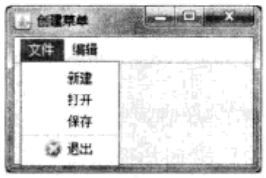


图 20.6　实例运行效果

注意：单击“文件”菜单中的“新建”、“打开”和“保存”3 个菜单项与“编辑”菜单中的“复制”和“粘贴”菜单项时没有任何反应，是由于在程序中没有给这 5 个菜单项添加事件监听器，而只给“退出”菜单项添加了事件监听器，所以单击“退出”按钮可以退出系统。再有就是这里只给“退出”菜单项添加了图标，其他菜单项的添加方法与此相同，读者可以自行添加。

20.3.3　创建弹出菜单

使用类 javax.swing.JPopupMenu 创建的对象是弹出菜单，弹出菜单是一种特殊形式的菜单，并不固定在窗体中的某个位置，而是由鼠标指针来决定弹出菜单出现的位置。该类的构造方法如表 20.8 所示。

表 20.8　JPopupMenu 类的构造方法

构 造 方 法	说　明
JPopupMenu( )	构造一个新的弹出菜单
JPopupMenu(String label)	构造一个有标题的弹出菜单

例如：

```
JPopupMenu popupMenu1 = new JPopupMenu();                 // 创建一个弹出菜单
JPopupMenu popupMenu2 = new JPopupMenu("操作");           // 创建一个标题为“操作”的弹出菜单
```

这段代码使用 JPopupMenu 类创建了两个弹出菜单，第一行创建的是没有标题的弹出菜单，第二行创建的是标题为“操作”的弹出菜单。

类 JPopupMenu 创建的对象是一个弹出菜单，该类提供了一些方法，可以完成弹出菜单的常用操作，如添加菜单项、显示弹出菜单等。该类的常用方法如表 20.9 所示。

表 20.9 JPopupMenu 类的常用方法及说明

方 法	说 明
add(JMenuItem menuItem)	将指定菜单项添加到此弹出菜单的末尾
addSeparator()	将新分隔符添加到菜单的末尾
isPopupTrigger(MouseEvent e)	如果弹出菜单的当前控件将鼠标事件视为弹出菜单的触发器，则返回 true
show(Component parent,int x,int y)	在控件调用者坐标空间中的位置(x,y)处显示弹出菜单

使用类 JPopupMenu 创建了弹出菜单后，就可以使用 JPopupMenu 类的方法为弹出菜单添加菜单项、分隔符等操作了。例如：

```
final JPopupMenu popupMenu = new JPopupMenu();                    // 创建一个弹出菜单
JMenuItem mi_new = new JMenuItem("新建");                          // 创建“新建”菜单项
JMenuItem mi_open = new JMenuItem("打开");                         // 创建“打开”菜单项
JMenuItem mi_exit = new JMenuItem("退出");                         // 创建“退出”菜单项
popupMenu.add(mi_new);                                            // 把“新建”菜单项添加到弹出菜单
popupMenu.add(mi_open);                                           // 把“打开”菜单项添加到弹出菜单
popupMenu.addSeparator();                                         // 添加分隔符
popupMenu.add(mi_exit);                                           // 把“退出”菜单项添加到弹出菜单
```

这段代码首先创建了一个弹出菜单，并在弹出菜单上添加了菜单项和分隔符。由此可见，弹出菜单的创建方法与普通菜单很相似，只需要把菜单项添加到弹出菜单上即可。

本节将以实例说明弹出菜单的创建、向弹出菜单添加菜单项以及通过控件的鼠标事件显示弹出菜单。

例 20.05 在项目中创建 JPopupMenuDemo 类，该类继承自 JFrame 类成为窗体类。在该类中创建一个弹出菜单，并为弹出菜单添加菜单项，然后创建标签控件，为标签控件添加鼠标事件监听器，使其能够响应鼠标的键释放事件以显示弹出菜单。实例的关键代码如下：**（实例位置：光盘\mr\ym\20.05）**

```
public JPopupMenuDemo() {
    setTitle("创建弹出菜单");
    setBounds(100, 100, 240, 160);                                // 设置窗体位置和大小
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);               // 设置窗体默认关闭方式
    final JPopupMenu popupMenu = new JPopupMenu();                // 创建一个弹出菜单
    JMenuItem mi_new = new JMenuItem("新建");                      // 创建“新建”菜单项
    JMenuItem mi_open = new JMenuItem("打开");                     // 创建“打开”菜单项
    JMenuItem mi_exit = new JMenuItem("退出");                     // 创建“退出”菜单项
    popupMenu.add(mi_new);                                        // 把“新建”菜单项添加到弹出菜单
    popupMenu.add(mi_open);                                       // 把“打开”菜单项添加到弹出菜单
    popupMenu.addSeparator();                                     // 添加分隔符
    popupMenu.add(mi_exit);                                       // 把“退出”菜单项添加到弹出菜单
    final JLabel label = new JLabel();
    label.addMouseListener(new MouseAdapter() {                   // 为标签添加鼠标事件监听器
        public void mouseReleased(final MouseEvent e) {           // 执行标签的鼠标事件
            // 如果弹出菜单的当前控件标签将鼠标事件视为弹出菜单的触发器
            if (e.isPopupTrigger()) {
```

```
                                                    // 在标签控件的鼠标右键单击处显示弹出菜单
                popupMenu.show(e.getComponent(), e.getX(), e.getY());
            }
        }
    });
    label.setOpaque(true);                                  // 设置标签为不透明
    label.setBackground(Color.PINK);                        // 设置标签的背景颜色
    getContentPane().add(label);                            // 把标签添加到窗体上
}
```

实例的运行效果如图 20.7 所示。

图 20.7　实例运行效果

注意： 本实例运行后，单击弹出菜单中的“新建”、“打开”和“退出”3 个菜单项时没有任何反应，是因为在程序中没有给这 3 个菜单项添加事件监听器。由于弹出菜单的菜单项监听器与普通菜单中的添加方法相同，所以这里只实现了显示弹出菜单的操作。再有就是在显示弹出菜单时如果不加上语句 e.isPopupTrigger()进行判断，则在标签上单击鼠标左键也将显示弹出菜单。

20.4　创建分割面板

分割面板可以将其所在的区域分割成两部分，这两部分之间有一个分隔条，可以通过调整分隔条的位置来改变这两部分的相对大小，用户可以根据需要进行调整，并且分割条两侧的每一部分中还可以再放分割面板，实现分割面板的嵌套使用。分割面板既可以在水平方向上进行分割，也可以在垂直方向上进行分割，程序员可以根据需要选择是在水平方向上分割，还是在垂直方向上进行分割。

20.4.1　创建分割面板

1．分割面板的构造方法

使用 javax.swing.JSplitPane 类创建的对象是一个分割面板。该类常用的构造方法如表 20.10 所示。

表 20.10　JSplitPane 类的常用构造方法

构 造 方 法	说　明
JSplitPane()	创建一个在水平方向进行分割、无连续布局、为控件使用两个按钮的分割面板
JsplitPane(int orient)	创建一个在指定方向进行分割且无连续布局的分割面板
JSplitPane(int orient, Component leftC, Component rightC)	创建一个在指定方向进行分割、不连续重绘的指定控件的分割面板

2. 表示分割方向的字段

在 JSplitPane 类的构造方法中有一个参数 orient，该参数是用于指定分割面板分割方向的字段。JSplitPane 类的分割方向字段如表 20.11 所示。

表 20.11 JSplitPane 类的分割方向字段

字 段	说 明
JSplitPane.HORIZONCAL_SPLIT	表示在水平方向上对分割面板进行分割
JSplitPane.VERTICAL_SPLIT	表示在垂直方向上对分割面板进行分割

例如：

```
// 创建一个在水平方向进行分割、无连续布局、为控件使用两个按钮的分割面板
final JSplitPane splitPane1 = new JSplitPane();
// 创建一个在垂直方向进行分割且无连续布局的分割面板
final JSplitPane splitPane2 = new JSplitPane(JSplitPane.VERTICAL_SPLIT);
```

这段代码使用 JSplitPane 类创建了两个分割面板，第一行创建的是一个在水平方向进行分割、无连续布局、为控件使用两个按钮的分割面板，第二行创建的是一个在垂直方向进行分割且无连续布局的分割面板。

注意：上面代码中的第二条语句在 JSplitPane 类的构造方法中使用了字段 JSplitPane.VERTICAL_SPLIT，表示创建的是一个在重直方向进行分割的分割面板。

20.4.2 使用分割面板

类 JSplitPane 创建的对象是一个分割面板，JSplitPane 类提供了一些方法，可以完成分割面板的常用操作。该类的常用方法如表 20.12 所示。

表 20.12 JSplitPane 类的常用方法

方 法	说 明
getBottomComponent()	返回分隔条下面或者右边的控件
getDividerSize()	返回分隔条的大小
getLeftComponent()	返回分隔条左边或者上面的控件
getRightComponent()	返回分隔条右边或者下面的控件
getTopComponent()	返回分隔条上面或者左边的控件
remove(Component component)	移除分割面板中的子控件 component
setBottomComponent(Component comp)	将控件设置到分隔条的下面或者右边
setLeftComponent(Component comp)	将控件设置到分隔条的左边或者上面
setRightComponent(Component comp)	将控件设置到分隔条的右边或者下面
setTopComponent(Component comp)	将控件设置到分隔条的上面或者左边
setDividerLocation(int location)	设置分隔条的位置
setDividerSize(int newSize)	设置分隔条的大小
setOneTouchExpandable(boolean newValue)	参数设置为 true 时，可以使分割面板在分隔条上提供一个 UI 小部件来快速展开/折叠分隔条
setOrientation(int orient)	设置分隔方向，设置为 JSplitPane.HORIZONCAL_SPLIT 表示在水平方向上对分割面板进行分割，设置为 JSplitPane.VERTICAL_SPLIT 表示在垂直方向上对分割面板进行分割

例如：

```
final JSplitPane splitPane1 = new JSplitPane();                    // 创建水平分割面板
splitPane1.setOrientation(JSplitPane.VERTICAL_SPLIT);              // 设置为在垂直方向进行分割
JLabel labelTop = new JLabel("上部标签");                          // 创建标签
JLabel labelBottom = new JLabel("底部标签");                       // 创建标签
splitPane1.setTopComponent(labelTop);                              // 在分割面板上部放标签
splitPane1.setBottomComponent(labelBottom);                        // 在分割面板底部放标签
splitPane1.setOneTouchExpandable(true);                            // 设置分割面板显示 UI 小部件
```

这段代码首先创建一个在水平方向进行分割、无连续布局、为控件使用两个按钮的分割面板，然后把分割面板设置为在垂直方向进行分割的分割面板，接着创建了两个标签，并把这两个标签分别放到垂直分割面板的上部和底部，最后把分割面板设置为显示 UI 小部件。

本节将以实例说明分割面板的创建、向分割面板添加控件和使分隔条显示 UI 小部件。

例 20.06　在项目中创建 SplitPaneFrame 类，该类继承自 JFrame 类成为窗体类。在该类中创建一个水平分割面板和一个垂直分割面板，并在水平分割面板的左侧和垂直分割面板的上面和下面各放一个标签，然后把垂直分割面板放到水平分割面板的右侧，设置水平分割面板的分隔条显示 UI 小部件。实例的关键代码如下：**（实例位置：光盘\mr\ym\20.06）**

```
public SplitPaneFrame() {
    setTitle("创建分割面板");
    setBounds(100, 100, 200, 160);                                 // 设置窗体位置和大小
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);                // 设置窗体默认关闭方式
    final JSplitPane hSplitPane = new JSplitPane();                // 创建在水平方向分割的分割面板
    JLabel labelLeft = new JLabel("左侧");                         // 创建标签
    // 创建一个在垂直方向进行分割的分割面板
    final JSplitPane vSplitPane = new JSplitPane(JSplitPane.VERTICAL_SPLIT);
    JLabel labelTop = new JLabel("上面");                          // 创建标签
    JLabel labelBottom = new JLabel("下面");                       // 创建标签
    vSplitPane.setTopComponent(labelTop);                          // 在垂直分割面板的上面放置标签
    vSplitPane.setBottomComponent(labelBottom);                    // 在垂直分割面板的下面放置标签
    vSplitPane.setDividerLocation(50);                             // 设置垂直分割面板的分割条距上边为 50 像素
    hSplitPane.setLeftComponent(labelLeft);                        // 在水平分割面板的左侧放置标签
    hSplitPane.setRightComponent(vSplitPane);                      // 在水平分割面板的右侧放置垂直分割面板
    hSplitPane.setDividerLocation(50);                             // 设置水平分割面板分割条距左边为 50 像素
    hSplitPane.setOneTouchExpandable(true);                        // 设置水平分割面板的分隔条显示 UI 小部件
    getContentPane().add(hSplitPane);                              // 把水平分割面板添加到窗体上
}
```

运行本实例，效果如图 20.8 所示，在窗体上有一个水平分割面板，在水平分割面板的右侧有一个垂直分割面板，并且在水平分割面板的分隔条上有 UI 小部件，单击 UI 小部件上的箭头可以快速展开或折叠分隔条，垂直分割面板的分隔条上没有 UI 小部件。

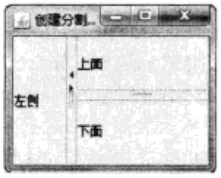


图 20.8　实例运行效果

20.5 创建并使用选项卡面板

选项卡面板可以方便地在窗体上放置多个标签页，每个标签页相当于一个与此选项卡大小相同的容器，可以在每个标签页的容器中放置一个或多个控件，使每个标签页实现不同的界面效果。

20.5.1 创建选项卡面板

1. 选项卡面板的构造方法

使用 javax.swing.JTabbedPane 类创建的对象是一个选项卡面板，它允许用户通过选择具有给定标题和/或图标的选项卡，在一组控件之间进行切换。该类的构造方法如表 20.13 所示。

表 20.13 JTabbedPane 类的构造方法

构 造 方 法	说 明
JTabbedPane()	创建一个在上面显示标签，布局方式为自动换行的空选项卡
JTabbedPane(int tabPlace)	创建一个在指定位置显示标签，布局方式为自动换行的空选项卡
JTabbedPane(int tabPlace, int tabLayout)	创建一个在指定位置显示标签和具有指定布局策略的空选项卡

2. 选项卡面板的字段

在 JTabbedPane 类的构造方法中的参数 tabPlace 是用于指定选项卡面板标签显示位置的字段，参数 tabLayout 是指定选项卡面板标签布局的字段。选项卡面板的常用字段如表 20.14 所示。

表 20.14 JTabbedPane 类的常用字段

字 段	说 明
JTabbedPane.TOP	表示在选项卡的上边位置显示标签
JTabbedPane.BOTTOM	表示在选项卡的下边位置显示标签
JTabbedPane.LEFT	表示在选项卡的左侧位置显示标签
JTabbedPane.RIGHT	表示在选项卡的右侧位置显示标签
JTabbedPane.WRAP_TAB_LAYOUT	当选项卡的一行或一列不能显示所有标签时，标签会自动换行或换列
JTabbedPane.SCROLL_TAB_LAYOUT	当选项卡的一行或一列标签显示不全时，标签行或列会出现 UI 调节控件

例如：

```
// 创建一个在上面显示标签，布局方式为自动换行的空选项卡
JTabbedPane tabbedPane1 = new JTabbedPane();
// 创建一个在左侧显示标签，布局方式为自动换行的空选项卡
JTabbedPane tabbedPane2 = new JTabbedPane(JTabbedPane.LEFT);
// 创建一个在左侧显示标签，布局方式为自动换行的空选项卡
JTabbedPane tabbedPane3 = new JTabbedPane(JTabbedPane.LEFT, JTabbedPane.WRAP_TAB_LAYOUT);
// 创建一个在右侧显示标签，布局方式为带 UI 调节控件的选项卡
JTabbedPane tabbedPane4 = new JTabbedPane(JTabbedPane.RIGHT, JTabbedPane.SCROLL_TAB_LAYOUT);
// 创建一个在下面显示标签，布局方式为带 UI 调节控件的选项卡
JTabbedPane tabbedPane = new JTabbedPane(JTabbedPane.BOTTOM, JTabbedPane.SCROLL_TAB_LAYOUT);
```

这段代码使用 JTabbedPane 类的 3 个构造方法创建了 5 个选项卡面板，并且在创建选项卡面板时还使用

JTabbedPane 类的字段指定了选项卡标签的位置和布局方式。

20.5.2　使用选项卡面板

类 JTabbedPane 创建的对象是一个选项卡面板，JTabbedPane 类提供了一些方法，可以完成选项卡面板的常用操作。该类的常用方法如表 20.15 所示。

表 20.15　JTabbedPane 类的常用方法

方　法	说　明
addTab(String title,Component component)	添加一个标题为 title 且没有图标的选项卡
addTab(String title,Icon icon,Component component)	添加一个标题为 title，图标为 icon 的选项卡
addTab(String title,Icon icon,Component component, String tip)	添加一个标题为 title，图标为 icon，工具提示文本为 tip 的选项卡
getSelectedComponent()	返回选项卡面板中当前选择的选项卡
getSelectedIndex()	返回当前选择的选项卡的索引
getTabComponentAt(int index)	返回索引值为 index 位置上的选项卡
getTabCount()	返回选项卡面板的选项卡数
getTitleAt(int index)	返回索引值为 index 位置的选项卡标题
getToolTipTextAt(int index)	返回索引值为 index 位置的选项卡工具提示文本
insertTab(String title,Icon icon,Component component, String tip,int index)	在索引值为 index 位置处插入一个标题为 title、图标为 icon、工具提示文本为 tip 的选项卡
setIconAt(int index,Icon icon)	将索引值为 index 位置的图标设置为 icon
setSelectedComponent(Component c)	设置此选项卡面板的已选控件
setSelectedIndex(int index)	设置索引值为 index 位置处的选项卡为当前选项卡
setTitleAt(int index,String title)	将索引值为 index 位置的选项卡标题设置为 title
setToolTipTextAt(int index,String toolTipText)	将索引值为 index 位置的工具提示文本设置为 toolTipText

本节将以实例说明选项卡面板的使用。

例 20.07　在项目中创建 TabbedPaneFrame 类，该类继承自 JFrame 类成为窗体类。在该类中创建一个选项卡面板，并在选项卡面板中添加控件，然后为选项卡设置标签图标和工具提示文本。实例的关键代码如下：**（实例位置：光盘\mr\ym\20.07）**

```
public TabbedPaneFrame() {
    super();
    setTitle("创建选项卡面板");                                    // 设置窗体标题
    setBounds(100, 100, 220, 200);                                 // 设置窗体位置和大小
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);                // 设置窗体默认关闭方式
    // 创建一个在上面显示标签，布局方式为自动换行的空选项卡
    final JTabbedPane tabbedPane = new JTabbedPane();
    final JButton button = new JButton("按钮");                    // 创建按钮控件
    Icon icon1 = new ImageIcon("src/images/111.png");              // 实例化 Icon 对象
    // 添加第一个选项卡，其标题为“选项卡一”，标题图标为 111.png
    // 添加的控件是按钮，工具提示文本为“这里使用了按钮”
    tabbedPane.addTab("选项卡一", icon1, button, "这里使用了按钮");
    final JCheckBox checkBox = new JCheckBox("复选框");             // 创建复选框控件
    // 添加第二个选项卡，在其上放置复选框控件，该选项卡无标签图标和工具提示文本
```

```
        tabbedPane.addTab("选项卡二", null, checkBox, null);
        Icon icon2 = new ImageIcon("src/images/222.png");                    // 实例化 Icon 对象
        // 为第二个选项卡添加标签图标
        tabbedPane.setIconAt(1, icon2);
        textArea = new JTextArea("文本域");                                  // 创建文本域控件
        // 添加第三个选项卡，在其上放置文本域控件，该选项卡无标签图标和工具提示文本
        tabbedPane.addTab("选项卡三", null, textArea, null);
        // 为第三个选项卡添加工具提示文本
        tabbedPane.setToolTipTextAt(2, "这里使用了文本域");
        final JRadioButton radioButton = new JRadioButton("单选按钮");       // 创建单选按钮控件
        // 添加第四个选项卡，在其上放置单选按钮控件，该选项卡无标签图标和工具提示文本
        tabbedPane.addTab("选项卡四", null, radioButton, null);
        getContentPane().add(tabbedPane, BorderLayout.CENTER);               // 把选项卡放置到窗体上
    }
```

运行本实例，结果如图 20.9 所示。“选项卡一”和“选项卡二”标题中含有图标，“选项卡三”和“选项卡四”标题中没有图标。当把鼠标移到标签“选项卡一”和“选项卡三”上时，稍停片刻，会显示提示文本。单击不同的选项卡标签可以看到不同的界面效果。

图 20.9　实例运行效果

20.6　桌面面板和内部窗体

桌面面板和内部窗体是用于创建多文档界面的，多文档界面的特点是存在一个外部窗体（也称为主窗体），在外部窗体中可以有多个内部窗体，内部窗体支持拖动、关闭、图标化、调整大小、标题显示和菜单栏等功能。当外部窗体最小化时，内部窗体会被隐藏；当拖动内部窗体时，不能拖出外部窗体。多文档界面的创建方法是首先创建桌面面板，然后把桌面面板放到 JFrame 窗体内容面板默认布局的中央，再创建内部窗体，并把内部窗体放到桌面面板容器中。

20.6.1　创建桌面面板

类 javax.swing.JDesktopPane 是多文档界面或虚拟桌面的容器，该类创建的对象是一个桌面面板，JDesktopPane 类只有一个无入口参数的构造方法 JDesktopPane()，用于构造一个桌面面板容器。例如：

```
JDesktopPane desktopPane =new JDesktopPane ( );// 创建一个桌面面板
```

这段代码使用 JDesktopPane 类创建了一个桌面面板。

类 JDesktopPane 创建的对象是一个桌面面板，该类提供的方法可以方便地操作桌面面板中的内部窗体和控件。该类的常用方法如表 20.16 所示。

表 20.16　JDesktopPane 类的常用方法

方　法	说　明
JInternalFrame[] getAllFrames()	返回桌面面板中当前显示的所有内部窗体
JInternalFrame getSelectedFrame()	返回此桌面面板中当前活动的内部窗体，如果当前没有活动的内部窗体，则返回 null
void remove(int index)	从此桌面面板中移除指定索引的控件
void removeAll()	从此桌面面板中移除所有控件
JInternalFrame selectFrame(boolean b)	选择此桌面面板中的下一个内部窗体
void setDragMode(int dragMode)	设置桌面面板使用的“拖动样式”
void setSelectedFrame(JInternalFrame f)	设置此桌面面板中当前活动的内部窗体

说明：本节只对桌面面板进行讲解，有关内部窗体的内容将在 20.6.2 节进行讲解。

类 JDesktopPane 创建的对象是一个桌面面板，使用该类的 setDragMode 方法可以设置内部窗体的拖动模式。该类设置拖动模式的字段如表 20.17 所示。

表 20.17　JDesktopPane 类设置拖动模式的字段

字　段	说　明
JDesktopPane.LIVE_DRAG_MODE	表示正在被拖动项的所有内容都出现在桌面面板内部，是默认设置
JDesktopPane.OUTLINE_DRAG_MODE	表示只是正在被拖动项的轮廓出现在桌面面板内部

例如：

```
final JDesktopPane desktopPane = new JDesktopPane();                    // 创建一个桌面面板
desktopPane.setDragMode(JDesktopPane.OUTLINE_DRAG_MODE);                // 拖动时只显示轮廓
```

这段代码使用 JDesktopPane 类创建了一个桌面面板，然后设置桌面面板的拖动模式，使其内部窗体被拖动时只显示轮廓。

JDesktopPane 类用于创建桌面面板，是内部窗体的容器，为了使多文档界面美观，可以为桌面面板添加背景图片，方法是创建一个带背景图片的标签，并把标签放到桌面面板容器的最底层。

例如：

```
final JDesktopPane desktopPane = new JDesktopPane();                    // 创建一个桌面面板
final JLabel imgLabel = new JLabel();                                   // 创建一个标签
URL url = this.getClass().getResource("picture.jpg");                   // 获取图片所在的 URL
Icon icon = new ImageIcon(url);                                         // 实例化 Icon 对象
imgLabel.setIcon(icon);                                                 // 为标签指定图片
imgLabel.setBounds(0, 0, icon.getIconWidth(), icon.getIconHeight());    // 设置标签的位置和大小
desktopPane.add(imgLabel, new Integer(Integer.MIN_VALUE));              // 把标签放到桌面面板的最底层
```

这段代码创建了一个桌面面板和一个标签，紧接着又创建了一个图片的 URL，并通过图片的 URL 创建了图片对象，然后为标签指定了图片对象，并指定标签的位置和大小，最后把标签放到桌面面板的最底层作为背景显示。

注意：上面代码最后一条语句中的 new Integer(Integer.MIN_VALUE)是用于指定把标签放在桌面面板最底层的关键代码，这样标签就可以显示在所有内部窗体的后方，而不会遮挡住内部窗体。

20.6.2 创建内部窗体

内部窗体通常与桌面面板一起使用，这样可以方便内部窗体的操作，类 javax.swing.JInternalFrame 创建的对象是一个内部窗体。该类的构造方法如表 20.18 所示。

表 20.18 JInternalFrame 类的构造方法

构 造 方 法	说 明
JInternalFrame()	创建不可调整大小、不可关闭、不可最大化、不可图标化、没有标题的内部窗体
JInternalFrame(String title)	创建不可调整大小、不可关闭、不可最大化、不可图标化、具有指定标题的内部窗体
JInternalFrame(String title,boolean resizable)	创建不可关闭的、不可最大化的、不可图标化的，以及具有指定标题和可调整大小的内部窗体
JInternalFrame(String title,boolean resizable,boolean closeable)	创建不可最大化的、不可图标化的，以及具有指定标题、可调整大小和可关闭的内部窗体
JInternalFrame(String title,boolean resizable,boolean closeable, boolean maximizable)	创建不可图标化的，但具有指定标题、可调整大小、可关闭和可最大化的内部窗体
JInternalFrame(String title,boolean resizable,boolean closeable, boolean maximizable,boolean iconifiable)	创建具有指定标题、可调整、可关闭、可最大化和可图标化的内部窗体

说明：表 20.18 中所列出的 JInternalFrame 类的 6 个构造方法中，最后一个较为常用，本书所使用的就是最后一个有 5 个入口参数的构造方法，当然也可以使用其他的构造方法，然后再用 JInternalFrame 类的方法进行设置也能实现相同的功能。

类 JInternalFrame 创建的对象是一个内部窗体，该类提供的方法可以实现对内部窗体的常用设置。该类的常用方法如表 20.19 所示。

表 20.19 JInternalFrame 类的常用方法

方 法	说 明
void dispose()	使此内部窗体不可见、取消选定并关闭它
String getTitle()	返回内部窗体的标题
boolean isClosed()	返回此内部窗体当前是否已关闭
boolean isIcon()	返回内部窗体当前是否已图标化
boolean isSelected()	返回内部窗体当前是否为选定的或处于激活状态的窗体
void setFrameIcon(Icon icon)	设置要在此内部窗体的标题栏中显示的图像
void setMaximizable(boolean b)	设置内部窗体是否提供最大化按钮
void setJMenuBar(JMenuBar m)	设置内部窗体的菜单栏
void setResizable(Boolean b)	设置是否可以调整内部窗体的大小
void setSelected(boolean selected)	设置是否激活此内部窗体
void setTitle(String title)	设置内部窗体的标题
void show()	如果内部窗体不可见，则将该内部窗体置于前端，使其可见并尝试选定它
void toBack()	将此内部窗体发送至后台
void toFrond	将此内部窗体置于前端

本节以实例说明桌面面板和内部窗体的结合使用。

例 20.08　在项目中创建 UseInternalFrame 类，该类继承自 JFrame 类成为窗体类。在该类中创建一个桌面面板、一个显示背景图片的标签和 3 个内部窗体。实例的关键代码如下：**（实例位置：光盘\mr\ym\20.08）**

```
public UseInternalFrame() {
    super();                                                    // 使用父类的构造方法
    setTitle("创建内部窗体");                                    // 设置窗体标题
    setBounds(100, 100, 420, 240);                              // 设置窗体位置和大小
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);             // 设置窗体默认关闭方式
    final JDesktopPane desktopPane = new JDesktopPane();        // 创建桌面面板
    getContentPane().add(desktopPane, BorderLayout.CENTER);     // 把桌面面板添加到窗体中央
    // 设置内部窗体的拖动模式，拖动时只显示外部轮廓
    desktopPane.setDragMode(JDesktopPane.OUTLINE_DRAG_MODE);
    final JLabel imgLabel = new JLabel();                       // 创建一个标签
    Icon icon1 = new ImageIcon("src/images/picture.jpg");       // 实例化 Icon 对象
    imgLabel.setIcon(icon1);                                    // 为标签指定图片
    // 设置标签的位置和大小
    imgLabel.setBounds(0, 0, icon1.getIconWidth(), icon1.getIconHeight());
    // 把标签放到桌面面板的最底层
    desktopPane.add(imgLabel, new Integer(Integer.MIN_VALUE));
    // 创建标题为“内部窗体一”、可调整大小、可关闭、可最大化和可图标化的内部窗体
    final JInternalFrame internalFrame1 = new JInternalFrame("内部窗体一", true, true, true, true);
    internalFrame1.setVisible(true);                            // 显示内部窗体
    internalFrame1.setBounds(10, 10, 169, 129);                 // 设置内部窗体的位置和大小
    desktopPane.add(internalFrame1);                            // 把内部窗体添加到内容面板
    // 创建标题为“内部窗体二”、可调整大小、可关闭、可最大化和可图标化的内部窗体
    final JInternalFrame internalFrame2 = new JInternalFrame("内部窗体二", true, true, true, true);
    internalFrame2.setVisible(true);                            // 显示内部窗体
    internalFrame2.setBounds(130, 40, 169, 129);                // 设置内部窗体的位置和大小
    desktopPane.add(internalFrame2);                            // 把内部窗体添加到内容面板
    // 创建标题为“内部窗体三”、可调整大小、可关闭、可最大化和可图标化的内部窗体
    final JInternalFrame internalFrame3 = new JInternalFrame("内部窗体三", true, true, true, true);
    internalFrame3.setVisible(true);                            // 显示内部窗体
    internalFrame3.setBounds(243, 67, 169, 129);                // 设置内部窗体的位置和大小
    desktopPane.add(internalFrame3);                            // 把内部窗体添加到内容面板
    Icon icon2 = new ImageIcon("src/images/titleicon.png");     // 实例化 Icon 对象
    // 获得桌面面板中当前显示的所有内部窗体
    JInternalFrame[] internalFrames = desktopPane.getAllFrames();
    for (int i = 0; i < internalFrames.length; i++) {
        internalFrames[i].setFrameIcon(icon2);                  // 设置每个内部窗体的标题图标
    }
    internalFrame1.toBack();                                    // 将此内部窗体一发送至后台
    internalFrame3.toFront();                                   // 将此内部窗体三置于前端
}
```

运行本实例，结果如图 20.10 所示。在外部窗体内部有 3 个内部窗体，可以最大化、图标化和关闭内部窗体，用鼠标拖动任何一个内部窗体，都不会被拖出外部窗体，并且拖动时只显示其外部轮廓。

图 20.10　实例运行效果

说明：本实例在拖动内部窗体时只显示外部轮廓，这是由于把桌面面板的拖动模式设置为字段 JdesktopPane.OUTLINE_DRAG_MODE 的原因，如果希望拖动时看到内部窗体的全部内容，则把桌面面板的拖动模式设置为 JDesktopPane.LIVE_DRAG_MODE 即可。

20.7　使用系统托盘

系统托盘是个特殊的区域，能够驻留程序，被运行在桌面上的所有应用程序共享，通常显示在桌面的底部，如 Microsoft Windows 的系统托盘就显示在任务栏的状态区，在那里可以随时访问正在运行的程序。

20.7.1　获取系统托盘

java.awt.SystemTray 类表示桌面的系统托盘，该类提供了一些方法可以实现托盘操作。该类的常用方法如表 20.20 所示。

表 20.20　SystemTray 类的常用方法

方　法	说　明
void add(TrayIcon trayIcon)	将托盘图标添加到系统托盘
static SystemTray getSystemTray()	获取表示桌面托盘区的系统托盘实例
TrayIcon[] getTrayIcons()	返回由此应用程序添加到托盘中的所有图标的数组
Dimension getTrayIconSize()	返回托盘图标在系统托盘中占用的空间大小（以像素为单位）
static boolean isSupported()	返回当前平台是否支持系统托盘
void remove(TrayIcon trayIcon)	从系统托盘中移除指定的托盘图标

说明：本节只对系统托盘进行讲解，有关托盘图标的内容将在 20.7.2 节进行讲解。

在某些平台上，可能不存在或不支持系统托盘，因此在使用系统托盘时，可以先使用 SystemTray 类的 isSupported()方法判断当前系统是否支持系统托盘，如果支持系统托盘，就可以使用 SystemTray 类的 getSystemTray()方法获得系统托盘。例如：

```
if (SystemTray.isSupported()) {                          // 判断是否支持系统托盘
    SystemTray systemTray = SystemTray.getSystemTray();  // 获得系统托盘对象
}
```

这段代码首先判断当前系统是否支持系统托盘，如果支持系统托盘就获得系统托盘对象。

20.7.2　为系统托盘添加图标

通过 20.7.1 节的学习，已经可以获得系统托盘了，但是还不能显示托盘图标，因此还需要为系统托盘添加托盘图标，系统托盘可以包含一个或多个托盘图标，可以使用 SystemTray 类的 add(TrayIcon icon)方法将托盘图标添加到系统托盘，当不再需要托盘时，可以使用 SystemTray 类的 remove(TrayIcon icon)方法将托盘图标移除。

java.awt.TrayIcon 类创建的对象是一个托盘图标。该类的构造方法如表 20.21 所示。

表 20.21　TrayIcon 类的构造方法

构 造 方 法	说　明
TrayIcon(Image image)	创建带有指定图像的托盘图标
TrayIcon(Image image,String tooltip)	创建带有指定图像和工具提示文本的托盘图标
TrayIcon(Image image,String tooltip, PopupMenu popup)	创建带指定图像、工具提示文本和弹出菜单的托盘图标

TrayIcon 类创建的对象是托盘图标，该类提供了一些方法可以方便地操作托盘图标。该类的常用方法如表 20.22 所示。

表 20.22　TrayIcon 类的常用方法

方　法	说　明
addMouseListener(MouseListener listener)	添加鼠标监听器，以接收发自此托盘图标的鼠标事件
setImage(Image image)	设置托盘图标的图像
setPopupMenu(PopupMenu popupMenu)	设置托盘图标的弹出菜单
setToolTip(String tooltip)	设置托盘图标的工具提示文本

例如：

```
if (SystemTray.isSupported()) {                          // 判断是否支持系统托盘
    URL url = this.getClass().getResource("exit.png");   // 获取图片所在的 URL
    ImageIcon icon = new ImageIcon(url);                 // 实例化图像对象
    Image image = icon.getImage();                       // 获得 Image 对象
    TrayIcon trayIcon = new TrayIcon(image);             // 创建托盘图标
    trayIcon.setToolTip("系统托盘");                      // 添加工具提示文本
    SystemTray systemTray = SystemTray.getSystemTray();  // 获得系统托盘对象
    try {
        systemTray.add(trayIcon);                        // 添加托盘图标
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

这段代码首先判断当前系统是否支持系统托盘，如果支持系统托盘，就获取图片的 URL 并创建图像对象，然后创建托盘图标，紧接着为托盘图标设置工具提示文本，最后获得系统托盘对象，并将托盘图标添加到系统托盘。

20.7.3　添加弹出菜单

通过 20.7.2 节的学习，已经可以获得系统托盘和显示托盘图标了，本节将学习为托盘图标添加弹出菜

单，方法有两种，下面分别介绍。

第一种方法的声明如下：

```
public TrayIcon(Image image,String tooltip,PopupMenu popup)
```

参数说明：image 是要使用的 Image。tooltip 是将用作工具提示文本的字符串；如果值为 null，则不显示工具提示。popup 是将用于托盘图标的弹出菜单的菜单；如果值为 null，则不显示弹出菜单。

通过此构造方法创建托盘图标对象时，给第 3 个入口参数传递一个弹出菜单即可。例如：

```
PopupMenu popupMenu = new PopupMenu();                          // 创建弹出菜单
MenuItem exit = new MenuItem("退出");                           // 创建菜单项
popupMenu.add(exit);                                            // 为弹出菜单添加菜单项
URL url = this.getClass().getResource("exit.png");              // 获取图片所在的 URL
ImageIcon icon = new ImageIcon(url);                            // 实例化图像对象
Image image = icon.getImage();                                  // 获得 Image 对象
TrayIcon trayIcon = new TrayIcon(image, "系统托盘", exit);      // 创建托盘图标
```

这段代码首先创建弹出菜单和菜单项，并把菜单项添加到弹出菜单上，然后获得图像的 URL 并创建图像对象，再获得图像的 Image 对象，最后创建带图像、工具提示文本和弹出菜单的托盘图标。

第二种方法的声明如下：

```
public void setPopupMenu(PopupMenu popup)
```

参数说明：popup 是一个 PopupMenu 类型的对象，如果该参数为 null，则移除所有弹出菜单。

例如：

```
PopupMenu popupMenu = new PopupMenu();                          // 创建弹出菜单
MenuItem exit = new MenuItem("退出");                           // 创建菜单项
popupMenu.add(exit);                                            // 为弹出菜单添加菜单项
URL url = this.getClass().getResource("exit.png");              // 获取图片所在的 URL
ImageIcon icon = new ImageIcon(url);                            // 实例化图像对象
Image image = icon.getImage();                                  // 获得 Image 对象
TrayIcon trayIcon = new TrayIcon(image);                        // 创建托盘图标
trayIcon.setPopupMenu(popupMenu);                               // 添加弹出菜单
```

这段代码首先创建弹出菜单和菜单项，并把菜单项添加到弹出菜单上，然后获得图像的 URL 并创建图像对象，再获得图像的 Image 对象，创建带图像的托盘图标，最后为托盘图标设置弹出菜单。

本节以实例说明系统托盘的使用。

例 20.09 在项目中创建 SystemTrayFrame 类，该类继承自 JFrame 类成为窗体类。在该类中获取系统托盘，并为系统托盘添加带工具提示文本和弹出菜单的托盘图标，通过弹出菜单可以退出系统，双击托盘图标可以显示应用程序窗体。实例的关键代码如下：**（实例位置：光盘\mr\ym\20.09）**

```
public SystemTrayFrame() {
    setTitle("使用系统托盘");                                   // 设置窗体标题
    setBounds(100, 100, 260, 200);                              // 设置窗体位置和大小
    setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);          // 设置窗体默认关闭方式
    if (SystemTray.isSupported()) {                             // 判断是否支持系统托盘
        ImageIcon icon = new ImageIcon("src/images/exit.png");  // 实例化图像对象
        Image image = icon.getImage();                          // 获得 Image 对象
        TrayIcon trayIcon = new TrayIcon(image);                // 创建托盘图标
        trayIcon.addMouseListener(new MouseAdapter() {          // 为托盘添加鼠标适配器
            public void mouseClicked(MouseEvent e) {            // 鼠标事件
                if (e.getClickCount() == 2) {                   // 判断是否双击了鼠标
                    showFrame();                                // 调用方法显示窗体
                }
```

```
            }
        });
        trayIcon.setToolTip("系统托盘");                          // 添加工具提示文本
        PopupMenu popupMenu = new PopupMenu();                    // 创建弹出菜单
        MenuItem exit = new MenuItem("退出");                     // 创建菜单项
        exit.addActionListener(new ActionListener() {             // 添加事件监听器
            public void actionPerformed(final ActionEvent arg0) {
                System.exit(0);                                   // 退出系统
            }
        });
        popupMenu.add(exit);                                      // 为弹出菜单添加菜单项
        trayIcon.setPopupMenu(popupMenu);                         // 为托盘图标添加弹出菜单
        SystemTray systemTray = SystemTray.getSystemTray();       // 获得系统托盘对象
        try {
            systemTray.add(trayIcon);                             // 为系统托盘添加托盘图标
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
public void showFrame() {
    this.setVisible(true);                                        // 显示窗体
}
```

运行本实例，在托盘图标上单击鼠标右键，将显示弹出菜单，结果如图 20.11 所示。双击托盘图标将显示应用程序窗体。

图 20.11　实例运行效果

20.8　照猫画虎——基本功训练

20.8.1　基本功训练 1——使子窗体最大化显示

视频讲解：光盘\mr\lx\20\使子窗体最大化显示.exe

实例位置：光盘\mr\ym\20.10\src

在使用内部窗体时，有时需要将窗体最大化，本实例将使用按钮来完成这个功能。

实现过程如下：

（1）在项目中创建一个窗体类，并将其命名为 MaxInternalFrame，在窗体中添加桌面面板和一个“最大化”按钮。

（2）监听单击“最大化”按钮事件，完成将子窗体最大化操作，关键代码如下：

```
protected void do_button_actionPerformed(ActionEvent e) {
    JInternalFrame jif = new JInternalFrame("子窗体", true, true, true, true);   // 创建内部子窗体
    jif.setSize(200, 200);                                                       // 设置窗体大小
    desktopPane.add(jif);                                                        // 添加子窗体到桌面面板
    jif.setVisible(true);                                                        // 显示子窗体
```

```
        try {
            jif.setMaximum(true);                                    // 设置内部子窗体最大化状态
        } catch (PropertyVetoException e1) {
            e1.printStackTrace();
        }
        desktopPane.setComponentZOrder(button, 0);                   // 把按钮置顶
    }
```

实例的运行效果如图 20.12 所示。

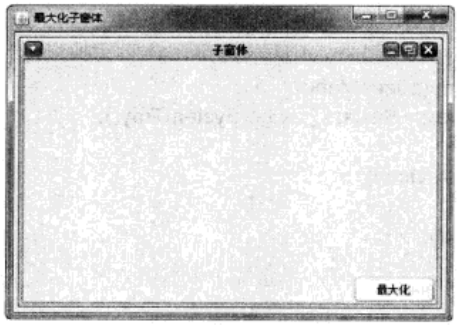


图 20.12　实例运行效果

你问我答：setMaximum()方法有什么作用？

该方法用于将子窗体最大化。该方法声明如下：

```
public void setMaximum(boolean b)throws PropertyVetoException
```

参数说明：b 是一个布尔值，用于设置是否最大化。

照猫画虎：模仿该实例，使用按钮关闭子窗体。（20 分）（实例位置：光盘\mr\ym\20.10\zmhh）

20.8.2　基本功训练 2——模仿记事本的菜单栏

视频讲解：光盘\mr\lx\20\模仿记事本的菜单栏.exe

实例位置：光盘\mr\ym\20.11\src

对于 Windows 系统而言，提供了记事本功能。本实例将模仿其菜单栏，关键代码如下：

```
public Notepad() {
    // 省略设置框架属性的代码
    JMenuBar menuBar = new JMenuBar();                                   // 创建菜单栏
    setJMenuBar(menuBar);                                                // 在框架中增加菜单栏
    JMenu fileMenu = new JMenu("\u6587\u4EF6(F)");                       // 创建名为“文件”的菜单
    menuBar.add(fileMenu);                                               // 将菜单添加到菜单栏中
    JMenuItem newMenuItem = new JMenuItem("\u65B0\u5EFA(N)");            // 创建新的菜单项
    fileMenu.add(newMenuItem);                                           // 将菜单项添加到菜单中
    JMenuItem openMenuItem = new JMenuItem("\u6253\u5F00(O)...");        // 创建新的菜单项
    fileMenu.add(openMenuItem);                                          // 将菜单项添加到菜单中
    JMenuItem saveMenuItem = new JMenuItem("\u4FDD\u5B58(S)");           // 创建新的菜单项
    fileMenu.add(saveMenuItem);                                          // 将菜单项添加到菜单中
    JMenuItem saveAsMenuItem = new JMenuItem("\u53E6\u5B58\u4E3A(A)..."); // 创建新的菜单项
    fileMenu.add(saveAsMenuItem);                                        // 将菜单项添加到菜单中
```

```
        JSeparator separator1 = new JSeparator();                          // 创建分隔符
        fileMenu.add(separator1);                                          // 将分隔符添加到菜单中
        JMenuItem pageSetMenuItem = new JMenuItem("\u9875\u9762\u8BBE\u7F6E(U)...");
        fileMenu.add(pageSetMenuItem);                                     // 将分隔符添加到菜单中
        JMenuItem printMenuItem = new JMenuItem("\u6253\u5370(P)...");     // 创建新的菜单项
        fileMenu.add(printMenuItem);                                       // 将菜单项添加到菜单中
        JSeparator separator2 = new JSeparator();                          // 创建分隔符
        fileMenu.add(separator2);                                          // 将分隔符添加到菜单中
        JMenuItem exitMenuItem = new JMenuItem("\u9000\u51FA(X)");         // 创建新的菜单项
        fileMenu.add(exitMenuItem);                                        // 将菜单项添加到菜单中
        // 省略其他菜单和文本域代码
    }
```

实例的运行效果如图 20.13 所示。

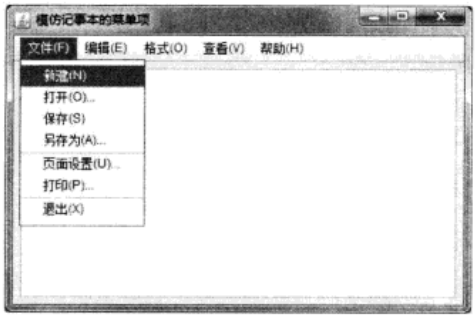


图 20.13　实例运行效果

照猫画虎：在本实例的基础上，将菜单项的字体设置成宋体，大小是 16。（20 分）（实例位置：光盘\mr\ym\20.11\zmhh）

20.8.3　基本功训练 3——在工具栏中增加图标

视频讲解：光盘\mr\lx\20\在工具栏中增加图标.exe

实例位置：光盘\mr\ym\20.12\src

对于工具栏而言，使用图标可以让其更加美观。本实例将完成一个由图标组成的工具栏，关键代码如下：

```
public IconToolBarDemo() {
    setTitle("\u5DE5\u5177\u680F\u7684\u5E94\u7528");
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setBounds(100, 100, 300, 230);
    contentPane = new JPanel();
    contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
    contentPane.setLayout(new BorderLayout(0, 0));
    setContentPane(contentPane);
    JToolBar toolBar = new JToolBar();                                     // 创建工具栏
    contentPane.add(toolBar, BorderLayout.NORTH);                          // 设置工具栏的位置
    JButton cutButton = new JButton("");                                   // 新建一个按钮
    cutButton.setToolTipText("\u526A\u5207");                              // 为按扭增加提示信息
    cutButton.setIcon(new ImageIcon("src/images/cut.png"));                // 为按钮设置图标
```

```
        toolBar.add(cutButton);                                          // 将按钮增加到工具栏上
        JButton findButton = new JButton("");                            // 新建一个按钮
        findButton.setToolTipText("\u67E5\u627E");                       // 为按扭增加提示信息
        findButton.setIcon(new ImageIcon("src/images/find.png"));        // 为按钮设置图标
        toolBar.add(findButton);                                         // 将按钮增加到工具栏上
        JButton openButton = new JButton("");                            // 新建一个按钮
        openButton.setToolTipText("\u6253\u5F00");                       // 为按扭增加提示信息
        openButton.setIcon(new ImageIcon("src/images/open.png"));        // 为按钮设置图标
        toolBar.add(openButton);                                         // 将按钮增加到工具栏上
        JButton saveButton = new JButton("");                            // 新建一个按钮
        saveButton.setToolTipText("\u4FDD\u5B58");                       // 为按扭增加提示信息
        saveButton.setIcon(new ImageIcon("src/images/save.png"));        // 为按钮设置图标
        toolBar.add(saveButton);                                         // 将按钮增加到工具栏上

    }
```

实例的运行效果如图 20.14 所示。

图 20.14　实例运行效果

照猫画虎：在本实例的基础上，将提示信息修改成在按钮上显示。（20 分）（实例位置：光盘\mr\ym\20.12\zmhh）

20.8.4　基本功训练 4——为选项卡增加快捷键

视频讲解：光盘\mr\lx\20\为选项卡增加快捷键.exe

实例位置：光盘\mr\ym\20.13\src

使用选项卡可以方便地扩展面板的空间。对于用户而言，使用鼠标来操作选项卡显然没有键盘方便，本实例将演示如何为选项卡增加快捷键，从而方便在不同选项卡之间切换。关键代码如下：

```
public MnemonicTabbedCardDemo() {
    setTitle("为选项卡面板增加快捷键");
    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);
    tabbedPane = new JTabbedPane(JTabbedPane.TOP);                       // 创建选项卡面板
    tabbedPane.setBorder(new EtchedBorder(EtchedBorder.LOWERED, null, null));   // 增加边框
    contentPane.add(tabbedPane, BorderLayout.CENTER);                    // 使用面板
    JLabel JavaLabel = new JLabel("Java 编程词典真好用！ ");              // 创建标签
    tabbedPane.addTab("Java", null, JavaLabel, null);                    // 设置选项卡标题
```

```
    tabbedPane.setMnemonicAt(0, KeyEvent.VK_J);                    // 为第一个选项卡增加快捷键“J”
    JLabel phpLabel = new JLabel("PHP 编程词典真好用！ ");          // 创建标签
    tabbedPane.addTab("PHP", null, phpLabel, null);                // 设置选项卡标题
    tabbedPane.setMnemonicAt(1, KeyEvent.VK_P);                    // 为第二个选项卡增加快捷键“P”
    JLabel VBLabel = new JLabel("VB 编程词典真好用！ ");            // 创建标签
    tabbedPane.addTab("Visual Basic", null, VBLabel, null);        // 设置选项卡标题
    tabbedPane.setMnemonicAt(2, KeyEvent.VK_B);                    // 为第三个选项卡增加快捷键“B”
    JLabel VCLabel = new JLabel("VC 编程词典真好用！ ");            // 创建标签
    tabbedPane.addTab("Visual C++", null, VCLabel, null);          // 设置选项卡标题
    tabbedPane.setMnemonicAt(3, KeyEvent.VK_C);                    // 为第四个选项卡增加快捷键“C”
}
```

实例的运行效果如图 20.15 所示。

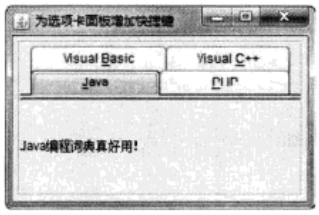


图 20.15　实例运行效果

你问我答：setMnemonicAt()方法有什么作用？

该方法可以为不同的选项卡指定不同的快捷键。该方法声明如下：

```
public void setMnemonicAt(int tabIndex,int mnemonic)
```

参数说明：tabIndex 是助记符引用的选项卡的索引，mnemonic 是表示助记符的键代码。

照猫画虎：在本实例的基础上，增加一个标题为 Ruby 的选项卡，并设置助记符为 R。（20 分）（实例位置：光盘\mr\ym\20.13\zmhh）

20.8.5　基本功训练 5——为选项卡增加图标

视频讲解：光盘\mr\lx\20\为选项卡增加图标.exe

实例位置：光盘\mr\ym\20.14\src

对于选项卡标题，可以使用图标进行美化。本实例将演示如何实现这个功能，关键代码如下：

```
public IconTabbedPaneDemo() {
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setTitle("\u5305\u542B 图标\u7684\u9009\u9879\u5361");
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setBounds(100, 100, 300, 200);
    contentPane = new JPanel();
    contentPane.setBorder(new EtchedBorder(EtchedBorder.LOWERED, null, null));
    contentPane.setLayout(new BorderLayout(0, 0));
    setContentPane(contentPane);
    JTabbedPane tabbedPane = new JTabbedPane(JTabbedPane.TOP);
    contentPane.add(tabbedPane, BorderLayout.CENTER);
    JLabel imageLabel1 = new JLabel("图标 1");                      // 创建标签
    tabbedPane.addTab("", null, imageLabel1, null);                // 设置选项卡标题
```

```
        tabbedPane.setIconAt(0, new ImageIcon("src/images/1.png"));          // 为第一个选项卡使用图片 1
        JLabel imageLabel2 = new JLabel("图标 2");                            // 创建标签
        tabbedPane.addTab("", null, imageLabel2, null);                       // 设置选项卡标题
        tabbedPane.setIconAt(1, new ImageIcon("src/images/2.png"));          // 为第二个选项卡使用图片 2
        JLabel imageLabel3 = new JLabel("图标 3");                            // 创建标签
        tabbedPane.addTab("", null, imageLabel3, null);                       // 设置选项卡标题
        tabbedPane.setIconAt(2, new ImageIcon("src/images/3.png"));          // 为第三个选项卡使用图片 3
        JLabel imageLabel4 = new JLabel("图标 4");                            // 创建标签
        tabbedPane.addTab("", null, imageLabel4, null);                       // 设置选项卡标题
        tabbedPane.setIconAt(3, new ImageIcon("src/images/4.png"));          // 为第四个选项卡使用图片 4
    }
```

实例的运行效果如图 20.16 所示。

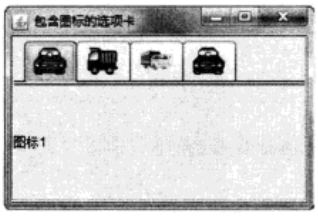


图 20.16　实例运行效果

照猫画虎：在本实例的基础上，为选项卡标题增加文字说明，即同时显示图片和文本信息。从左至右依次是“图标 1”、“图标 2”、“图标 3”和“图标 4”。（20 分）（实例位置：光盘\mr\ym\20.14\zmhh）

照猫画虎栏目分数统计：

照猫画虎题目	1	2	3	4	5	总分数	
分数							

20.9　情景应用——拓展与实践

20.9.1　情景应用 1——支持图片预览的文件选择对话框

视频讲解：光盘\mr\lx\20\支持图片预览的文件选择对话框.exe

实例位置：光盘\mr\ym\20.15\src

Swing 提供了 JFileChooser 控件，它可以实现选择文件或文件夹的操作，但它并不能支持对图片的预览功能。对于文件选择器而言，如果能够提供图片的预览功能，是非常方便用户使用的。本实例将完成这个功能，实例的运行效果如图 20.17 所示。

实现过程如下：

（1）新建项目并创建窗体类 ImagePreviewDialog。

（2）创建 PaintPanel 类，该类继承了 JPanel。通过重写 paintComponent()方法来完成在该面板上显示图片的功能。在创建该类的对象后，需要为其传递一个 Image 类型的参数，这样才能进行图片的绘制。在面板上绘制图片，可以使用 Graphics 类中定义的 drawImage()方法。关键代码如下：

```
private class PaintPanel extends JPanel {
    private static final long serialVersionUID = 35108003450998472 87L;
    private Image image;
    public PaintPanel() {
        super();
        setOpaque(false);
        setLayout(null);
    }
    public void setImage(Image image) {
        this.image = image;
    }
    @Override
    protected void paintComponent(Graphics g) {                    // 重新绘制控件外观
        if (image != null) {
            g.drawImage(image, 0, 0, getWidth(), getHeight(), this);   // 绘制图片与控件大小相同
        }
        super.paintComponent(g);                                   // 执行超类方法
    }
}
```

图 20.17　实例运行效果

（3）在 ImagePreviewDialog 类的构造方法中，创建了文件选择器对象和刚刚自定义的 PaintPanel 类对象。使用 setAccessory()方法把面板设置为文件选择器控件。此外，对于文件选择器属性改变事件增加了监听器，关键代码如下：

```
public ImagePreviewDialog() {
    setTitle("支持文件预览的对话框");
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setBounds(100, 100, 450, 300);
    contentPane = new JPanel();
    contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
    contentPane.setLayout(new BorderLayout(0, 0));
    setContentPane(contentPane);
    JFileChooser fileChooser = new JFileChooser();                 // 创建文件选择器
    contentPane.add(fileChooser, BorderLayout.CENTER);             // 添加到窗体
    paint = new PaintPanel();                                      // 创建图片预览面板
    paint.setBorder(new BevelBorder(BevelBorder.LOWERED, null, null, null, null));// 设置面板的边框
    paint.setPreferredSize(new Dimension(150, 300));               // 设置预览面板的大小
    fileChooser.setAccessory(paint);                               // 把面板设置为文件选择器控件
    fileChooser.addPropertyChangeListener(new PropertyChangeListener() { // 添加选择器的属性事件监听器
        public void propertyChange(PropertyChangeEvent arg0) {
```

```
                do_this_propertyChange(arg0);
            }
        });
    }
```

（4）编写文件选择器的属性改变事件处理方法，当改变选定文件时，这个方法会把图片文件加载到程序中，并设置图片预览面板的属性进行显示。关键代码如下：

```
protected void do_this_propertyChange(PropertyChangeEvent e) {
    // 处理改变选定文件的属性事件
    if (JFileChooser.SELECTED_FILE_CHANGED_PROPERTY == e.getPropertyName()) {
        File picfile = (File) e.getNewValue();                          // 获取选定的文件
        if (picfile != null && picfile.isFile()) {
            try {
                Image image = getToolkit().getImage(picfile.toURI().toURL());   // 从文件加载图片
                paint.setImage(image);                                  // 设置预览面板的图片
                paint.repaint();                                        // 刷新预览面板的界面
            } catch (MalformedURLException e1) {
                e1.printStackTrace();
            }
        }
    }
}
```

你问我答：setAccessory()方法有什么作用？

该方法用于设置文件选择器的 Accessory 控件属性，这个属性可以把一个控件作为文件选择器的辅助控件。

DIY：增加文件选择过滤功能，使其仅支持 jpg、png 和 gif 格式的文件。（20 分）（实例位置：光盘\mr\ym\20.15\diy）

20.9.2 情景应用 2——记录选项卡的访问状态

视频讲解：光盘\mr\lx\20\记录选项卡的访问状态.exe

实例位置：光盘\mr\ym\20.16\src

对于选项卡而言，可以为其标题设置图标。如果访问过的选择卡标题中的图标与未访问过的不同，就相当于完成了记录访问状态的功能。这有些类似网页中的超链接，每次单击之后文本的颜色会发生变化。本实例将完成这个功能，实例的运行效果如图 20.18 所示。

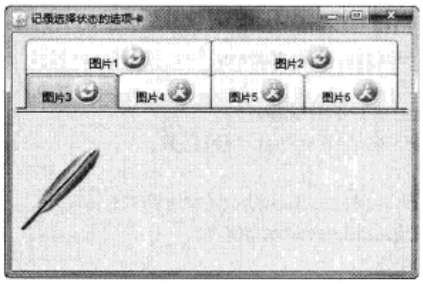


图 20.18 实例运行效果

实现过程如下：

（1）创建窗体类 VisitedTabbedPane，并增加选项卡面板控件。

（2）在该类的构造方法中，增加了选项卡面板。对于该面板，增加了变化事件监听器。然后使用一个标签数组来保存选项卡的内容，使用数组来简化标签的图片设置。最后将这些标签增加到选项卡中。对于不同的选项卡标题，其图标进行了统一设置。关键代码如下：

```
public VisitedTabbedPane() {
    setTitle("记录选择状态的选项卡");
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setBounds(100, 100, 450, 300);
    contentPane = new JPanel();
    contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
    contentPane.setLayout(new BorderLayout(0, 0));
    setContentPane(contentPane);
    tabbedPane = new JTabbedPane(JTabbedPane.TOP);
    tabbedPane.addChangeListener(new ChangeListener() {
        public void stateChanged(ChangeEvent e) {
            do_tabbedPane_stateChanged(e);
        }
    });
    contentPane.add(tabbedPane, BorderLayout.CENTER);
    JLabel[] imageLabels = new JLabel[6];
    for (int i = 0; i < imageLabels.length; i++) {
        imageLabels[i] = new JLabel();                                    // 创建标签
        imageLabels[i].setIcon(new ImageIcon("src/images/" + (i + 1) + ".png"));   // 为标签设置图标
        tabbedPane.addTab("图片" + i, null, imageLabels[i], null);        // 设置选项卡面板标题
        tabbedPane.setIconAt(i, new ImageIcon("src/images/no.png"));      // 设置选项卡标题图片 no
    }
}
```

（3）监听选项卡面板选择事件，它获得了用户选择的选项卡并重新设置了图片。关键代码如下：

```
protected void do_tabbedPane_stateChanged(ChangeEvent e) {
    if (tabbedPane.getSelectedComponent() != null) {
        int index = tabbedPane.getSelectedIndex();                        // 获得用户选择的选项卡
        tabbedPane.setIconAt(index, new ImageIcon("src/images/yes.png")); // 设置新图标
    }
}
```

DIY：在本实例的基础上，禁用访问过的选项卡面板。（20 分）（实例位置：光盘\mr\ym\20.16\diy）

20.9.3　情景应用 3——简单的图片浏览器

视频讲解：光盘\mr\lx\20\简单的图片浏览器.exe

实例位置：光盘\mr\ym\20.17\src

使用 Swing 中的标签控件可以很方便地显示图片，本实例将使用其制作一个图片浏览器。用户通过单击“上一张”、“下一张”按钮来完成图片的切换。为了贴近实战，如果当前显示的图片是第一张图片，则“上一张”按钮会被禁用。同理，如果当前显示的图片是最后一张图片，则“下一张”按钮会被禁用。实例的运行效果如图 20.19 所示。

图 20.19　实例运行效果

实现过程如下：

（1）创建窗体类 ImageViewer，在窗体中增加标签和按钮控件。

（2）对“上一张”按钮增加事件监听器，这是通过 do_button1_actionPerformed()方法实现的。它完成了更换标签上显示的图标的功能，同时根据图片在数组中的位置来决定是否需要禁用按钮。核心代码如下：

```
protected void do_button1_actionPerformed(ActionEvent e) {
    button2.setEnabled(true);                                    // 启用按钮
    ImageIcon icon = (ImageIcon) label.getIcon();                // 获得标签上显示的图片
    for (int i = 0; i < icons.length; i++) {
        if (icons[i].equals(icon)) {                             // 查找图片的索引
            if (i == 0) {
                button1.setEnabled(false);                       // 禁用按钮
                return;
            }
            label.setIcon(icons[(i - 1) % icons.length]);        // 更新图片
            if (i == 1) {
                button1.setEnabled(false);                       // 禁用按钮
            }
        }
    }
}
```

（3）对“下一张”按钮增加事件监听器，这是通过 do_button2_actionPerformed()方法实现的。它完成了更换标签上显示的图标的功能，同时根据图片在数组中的位置来决定是否需要禁用按钮。核心代码如下：

```
protected void do_button2_actionPerformed(ActionEvent e) {
    button1.setEnabled(true);                                    // 启用按钮
    ImageIcon icon = (ImageIcon) label.getIcon();                // 获得标签上显示的图片
    for (int i = 0; i < icons.length; i++) {                     // 查找图片的索引
        if (icons[i].equals(icon)) {
            if (i == icons.length - 1) {
                button2.setEnabled(false);                       // 禁用按钮
                return;
            }
            label.setIcon(icons[(i + 1) % icons.length]);        // 更新图片
            if (i == icons.length - 2) {
                button2.setEnabled(false);                       // 禁用按钮
            }
        }
    }
}
```

DIY：在本实例的基础上增加显示图片信息的功能。（20 分）（实例位置：光盘\mr\ym\20.17\diy）

20.9.4　情景应用 4——浏览文本文件内容

视频讲解：光盘\mr\lx\20\浏览文本文件内容.exe

实例位置：光盘\mr\ym\20.18\src

本实例将完成一个很实用的小工具，它可以打开文本文件，并输出文本文件的位置、内容。其中用到了菜单、工具条、选项卡面板等控件，是 Swing 高级控件的一个组合应用。实例的运行效果如图 20.20 所示。

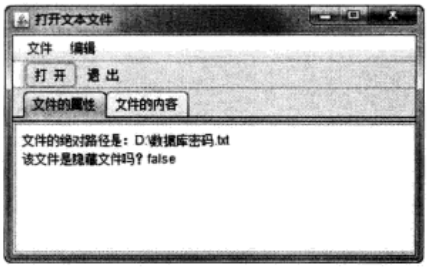


图 20.20　实例运行效果

实现过程如下：

（1）创建窗体类 TextFileOpener，在窗体中增加菜单、工具条、选项卡面板等控件。

（2）在该类的构造方法中，增加了菜单等控件。对于“打开”和“退出”按钮，分别增加了事件监听器。对于“打开”按钮，可以完成文件的选择和显示文件信息、内容等操作。对于“退出”按钮，可以方便地退出程序。实例的完整代码如下：

```
public class TextFileOpener extends JFrame {
    private static final long serialVersionUID = -9077023825514749548L;
    private JTextArea ta_showText;                                    // 定义显示文件属性的文本域
    private JTextArea ta_showProperty;                                // 定义显示文件内容的文本域
    public static void main(String[] args) {
        try {
            UIManager.setLookAndFeel("com.sun.java.swing.plaf.nimbus.NimbusLookAndFeel");
        } catch (Throwable e) {
            e.printStackTrace();
        }
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                try {
                    TextFileOpener frame = new TextFileOpener();
                    frame.setVisible(true);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }
    public TextFileOpener() {
        setTitle("打开文本文件");                                      // 设置窗体标题
        setBounds(100, 100, 400, 250);                                // 设置窗体位置和大小
```

```
setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);// 设置窗体默认关闭方式
final JMenuBar menuBar = new JMenuBar();                    // 创建菜单栏
setJMenuBar(menuBar);                                       // 把菜单栏放到窗体上
final JMenu mn_file = new JMenu();                          // 创建“文件”菜单
mn_file.setText("文件");                                    // 为“文件”菜单设置标题
menuBar.add(mn_file);                                       // 把“文件”菜单添加到菜单栏上
final JMenuItem mi_open = new JMenuItem();                  // 创建“打开”菜单项
mi_open.addActionListener(new ActionListener() {            // 为“打开”菜单项添加监听器
    public void actionPerformed(final ActionEvent arg0) {
        openTextFile();                                     // 调用方法，操作文件
    }
});
mi_open.setText("打开");                                    // 设置“打开”菜单项的标题
mn_file.add(mi_open);                                       // 把“打开”菜单项添加到“文件”菜单
mn_file.addSeparator();                                     // 添加菜单分隔符
final JMenuItem mi_exit = new JMenuItem();                  // 创建“退出”菜单项
mi_exit.addActionListener(new ActionListener() {            // 为“退出”菜单项添加监听器
    public void actionPerformed(final ActionEvent arg0) {
        System.exit(0);                                     // 退出系统
    }
});
mi_exit.setText("退出");                                    // 设置“退出”菜单项的标题
mn_file.add(mi_exit);                                       // 把“退出”菜单项添加到“文件”菜单
final JMenu mn_edit = new JMenu();                          // 创建“编辑”菜单
mn_edit.setText("编辑");                                    // 为“编辑”菜单设置标题
menuBar.add(mn_edit);                                       // 把“编辑”菜单添加到菜单栏上
final JMenuItem mi_copy = new JMenuItem();                  // 创建“复制”菜单项
mi_copy.setText("复制");                                    // 设置“复制”菜单项的标题
mn_edit.add(mi_copy);                                       // 把“复制”菜单项添加到“编辑”菜单
final JMenuItem mi_cut = new JMenuItem();                   // 创建“剪切”菜单项
mi_cut.setText("剪切");                                     // 设置“剪切”菜单项的标题
mn_edit.add(mi_cut);                                        // 把“剪切”菜单项添加到“编辑”菜单
final JMenuItem mi_paste = new JMenuItem();                 // 创建“粘贴”菜单项
mi_paste.setText("粘贴");                                   // 设置“粘贴”菜单项的标题
mn_edit.add(mi_paste);                                      // 把“粘贴”菜单项添加到“编辑”菜单
final JToolBar toolBar = new JToolBar();                    // 创建工具栏
getContentPane().add(toolBar, BorderLayout.NORTH);          // 把工具栏放到窗体上方
final JButton btn_open = new JButton();                     // 创建工具按钮
btn_open.addActionListener(new ActionListener() {           // 添加动作监听器
    public void actionPerformed(final ActionEvent arg0) {
        openTextFile();                                     // 调用方法，操作文件
    }
});
btn_open.setText("  打  开  ");                             // 设置工具按钮的标题
toolBar.add(btn_open);                                      // 把工具按钮添加到工具栏上
final JButton btn_exit = new JButton();                     // 创建工具按钮
btn_exit.addActionListener(new ActionListener() {           // 添加动作监听器
    public void actionPerformed(final ActionEvent arg0) {
        System.exit(0);                                     // 退出系统
```

```
            }
        });
        btn_exit.setText("  退  出  ");                          // 设置工具按钮的标题
        toolBar.add(btn_exit);                                   // 把工具按钮添加到工具栏上
        final JTabbedPane tabbedPane = new JTabbedPane();        // 创建选项卡面板
        getContentPane().add(tabbedPane, BorderLayout.CENTER);   // 把选项卡面板放到窗体中央
        final JScrollPane scrollPane1 = new JScrollPane();       // 创建滚动面板
        // 把滚动面板放到选项卡的第一个选项页
        tabbedPane.addTab("文件的属性", null, scrollPane1, null);
        ta_showProperty = new JTextArea();                       // 创建文本域
        // 把文本域添加到滚动面板的视图中
        scrollPane1.setViewportView(ta_showProperty);
        final JScrollPane scrollPane2 = new JScrollPane();       // 创建滚动面板
        // 把滚动面板放到选项卡的第二个选项页
        tabbedPane.addTab("文件的内容", null, scrollPane2, null);
        ta_showText = new JTextArea();                           // 创建文本域
        // 把文本域添加到滚动面板的视图中
        scrollPane2.setViewportView(ta_showText);
    }
    public void openTextFile() {
        JFileChooser fileChooser = new JFileChooser();           // 创建文件选择对话框
        int returnValue = fileChooser.showOpenDialog(getContentPane());  // 打开文件选择对话框
        if (returnValue == JFileChooser.APPROVE_OPTION) {        // 判断用户是否选择了文件
            File file = fileChooser.getSelectedFile();           // 获得文件对象
            ta_showProperty.append("文件的绝对路径是：" + file.getAbsolutePath() + "\n");
                                                                 // 获得文件的绝对路径
            ta_showProperty.append("该文件是隐藏文件吗？" + file.isHidden() + "\n");  // 是否为隐藏文件
            FileReader reader;                                   // 声明字符流
            BufferedReader in;                                   // 声明字符缓冲流
            try {
                reader = new FileReader(file);                   // 创建字符流
                in = new BufferedReader(reader);                 // 创建字符缓冲流
                String info = in.readLine();                     // 从文件中读取一行信息
                while (info != null) {                           // 判断是否读到内容
                    ta_showText.append(info + "\n");             // 把读到的信息追加到文本域中
                    info = in.readLine();                        // 继续读下一行信息
                }
                in.close();                                      // 关闭字符缓冲流
                reader.close();                                  // 关闭字符流
            } catch (Exception ex) {
                ex.printStackTrace();                            // 输出栈踪迹
            }
        }
    }
}
```

DIY：在本实例的基础上增加过滤器，使用户仅能选择文本文件。（20 分）（实例位置：光盘\mr\ym\20.18\diy）

20.9.5 情景应用 5——在弹出菜单中显示图片

视频讲解：光盘\mr\lx\20\在弹出菜单中显示图片.exe

实例位置：光盘\mr\ym\20.19\src

为了让弹出菜单更加美观，可以使用图片替代文本。例如，使用一个剪刀图片代表剪切操作、使用一个放大镜图片代表查找操作等。本实例将这些常见的图片应用到了弹出菜单中，从而代替了枯燥乏味的文本说明。实例的运行效果如图 20.21 所示。

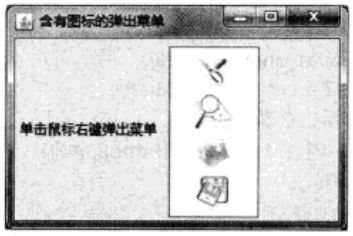


图 20.21 实例运行效果

实现过程如下：

（1）在项目中新建窗体类 PopupMenuDemo。

（2）在该类的构造方法中增加菜单项。对于每个菜单项并没有指定文本信息，而是设置了图标。关键代码如下：

```
public PopupMenuDemo() {
    setTitle("含有图标的弹出菜单");
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setBounds(100, 100, 300, 200);
    contentPane = new JPanel();
    contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
    JPopupMenu popupMenu = new JPopupMenu();                    // 创建弹出式菜单
    contentPane.setComponentPopupMenu(popupMenu);               // 为面板增加弹出式菜单
    JMenuItem cut = new JMenuItem();                            // 创建新菜单项
    cut.setIcon(new ImageIcon("src/images/cut.png"));           // 为菜单项设置标签
    popupMenu.add(cut);                                         // 增加菜单项
    JMenuItem find = new JMenuItem();                           // 创建新菜单项
    find.setIcon(new ImageIcon("src/images/find.png"));         // 为菜单项设置标签
    popupMenu.add(find);                                        // 增加菜单项
    JMenuItem open = new JMenuItem();                           // 创建新菜单项
    open.setIcon(new ImageIcon("src/images/open.png"));         // 为菜单项设置标签
    popupMenu.add(open);                                        // 增加菜单项
    JMenuItem save = new JMenuItem();                           // 创建新菜单项
    save.setIcon(new ImageIcon("src/images/save.png"));         // 为菜单项设置标签
    popupMenu.add(save);                                        // 增加菜单项
    contentPane.setLayout(new BorderLayout(0, 0));
    setContentPane(contentPane);
    JLabel label = new JLabel("单击鼠标右键弹出菜单");
```

```
        contentPane.add(label, BorderLayout.CENTER);
    }
```

DIY：在本实例的基础上，为弹出菜单中的图片增加说明信息，从上至下依次是“剪切”、“查找”、“打开”和“保存”。（20 分）（实例位置：光盘\mr\ym\20.19\diy）

情景应用 DIY 栏目分数统计：

DIY 题目	1	2	3	4	5	总分数	
分数							

20.10　自 我 测 试

一、选择题（每题 10 分，5 道题）

1．下列代码的作用是？（　　）

```
JFileChooser chooser = new JFileChooser();
chooser.setFileSelectionMode(JFileChooser.FILES_ONLY);
```

A．创建了一个只能选择文件的文件选择器　　B．创建了一个只能选择文件夹的文件选择器

C．编译错误　　D．运行错误

2．使用 setOrientation()方法可以为工具条设置方向，该方法的参数可以是？（　　）

A．HORIZONTAL　　B．VERTICAL　　C．LEFT　　D．RIGHT

3．对于 JSplitPane 而言，可以使用 setOrientation()方法分隔方向，该方法的参数可以是？（　　）

A．HORIZONTAL　　B．VERTICAL　　C．LEFT　　D．RIGHT

4．如果要为选项卡面板 tab 的第二个选项卡增加快捷键 R，则可以使用下列哪条语句？（　　）

A．tab. setMnemonicAt(1, KeyEvent.VK_R)　　B．tab. setMnemonicAt(3, KeyEvent.VK_C)

C．tab. setMnemonicAt(2, KeyEvent.VK_R)　　D．tab. setMnemonicAt(4, KeyEvent.VK_C)

5．如果需要最大化子窗体，则可以使用 JInternalFrame 类中的哪个方法？（　　）

A．setMax (boolean b)　　B．setMin (boolean b)

C．setMaximum(boolean b)　　D．setMinimum(boolean b)

二、填空题（每题 10 分，5 道题）

1．如果需要限制文件选择器不能够同时选择多个文件，则可以使用 JFileChooser 类中的哪个方法？___

2．对于文件选择器 chooser，如果希望它从 D 盘开始选择，则可以使用哪行代码？_____

3．如果要对工具条增加分隔符，则可以使用哪个方法？_____

4．使用 javax.swing 包中哪个类可以创建一个菜单栏？_____

5．如果需要关闭子窗体，则可以使用 JInternalFrame 类中的哪个方法？_____

测试分数统计：

类别	第 1 题	第 2 题	第 3 题	第 4 题	第 5 题
选择题分数					
填空题分数					
总分数					

20.11 行 动 指 南

开始日期：_____年____月____日　　结束日期：_____年____月____日

序号	内　容	行 动 指 南	
1	照猫画虎栏目 分数（　　）	分数>75 分	优秀，基本功掌握得不错，加油！
		75 分>分数>50 分	及格，知识掌握得不牢，重新做一遍照猫画虎。
		分数<50 分	请使用光盘中提供的“实战能力测试系统”进行提高训练。
	情景应用栏目 分数（　　）	分数>75 分	优秀，综合应用能力很强。
		75 分>分数>50 分	及格，综合应用能力需提高，再练一遍情景应用。
		分数<50 分	请使用光盘中提供的“实战能力测试系统”进行提高训练。
	自我测试栏目 分数（　　）	分数>75 分	优秀，有成为编程高手的潜质。
		分数<75 分	请使用光盘中提供的“实战能力测试系统”进行提高训练。
	综合评价	反复训练后，以上各项分数都在 75 分以上，方可进入下一堂课学习。	
2	编程实践能力培养：本栏目提供了一些实践题目，对于培养编程能力很有效，要做好。	（1）制作电子相册。	
		（2）制作简单的记事本。	
		（3）为记事本设置系统托盘图标。	
		（4）为记事本增加定时保存功能。	
3	编程习惯培养：本堂课培养读者在学习开发中记笔记的习惯，把开发中遇到的问题、总结的经验记录下来。		
4	创新能力培养：看看身边有没有可以用编程解决的问题，记录到右边的表格中。根据学习进度，尝试编写解决这些问题的小程序。		

20.12 成功可以复制——暴雪公司的领航者

暴雪公司 CEO 迈克·莫汉（Mike Morhaime）是一位相当具有传奇色彩的人物，他从只有 3 个办公桌的办公室起家，经过多年的打拼将暴雪发展成为一家全球知名的电脑游戏及电视游戏软件公司。他在暴雪的 20 年中，成功开发了魔兽争霸系列、星际争霸系列、黑破坏神系列以及《魔兽世界》游戏。魔兽争霸及星际争霸均被多项知名电子竞技比赛列为主要比赛项目，而单单是《魔兽世界》游戏，每年就可以为暴雪公司带来高达将近 10 亿美元的收入。

迈克·莫汉在小学的时候就和哥哥妹妹们一起买了名为 Bally Professional Arcade 的游戏机，而他对于其上游戏的制作颇感兴趣。他在上学的时候就一直在研究如何让那些程序很好地运行。

作为美国加州大学电子工程系的学生，迈克·莫汉有更多的机会接触到游戏制作与开发，这也为他未

来创业奠定了基础。1991 年，大学毕业的迈克·莫汉（Mike Morhaime）和合作者共同创建了暴雪游戏开发公司。

一个大型游戏的开发，往往需要一年甚至更长的时间，投入与回报的周期比较长。初期，迈克·莫汉为了支付员工的薪水，经常需要从个人信用卡提取现金，依赖个人的借款来支撑公司。在那段时间里，他们的压力相当大，既要调动雇员的最大积极性发挥其才智，又要承受只出不入“耕耘期”的阵痛。当时，从资金流动的视角看，他们几乎一无所有。

1995 年对于迈克·莫汉来说是真正具有意义的一年，他们连续推出了《魔兽争霸 2》、《星际争霸》、《黑破坏神 2》等大作，受到了市场的追捧和热销。2007 年《魔兽世界：燃烧的远征》在发售后 24 小时内卖出了 240 万套，《魔兽世界》全球注册用户更是达到 1000 万。《魔兽世界》的成功使得迈克·莫汉成为享誉世界的游戏设计大师，也使他创办的暴雪公司一跃成为世界最顶级的游戏开发商，创造了游戏产业的一个神话。

经典语录

优秀的游戏是我们的事业。

深度评价

借用 John Carmack 的话，更容易了解迈克·莫汉的成功：游戏的程序就是一行行的代码，需要你的灵魂才能赋予它以生命，有了生命的游戏才能带给大家快乐，也才能给你所想要的，你的游戏是你的热情，是你的汗水，是你的喜怒哀乐，是你的梦想，你的游戏，其实就是你自己。我从未度过没有编程的一天，这就是我的全部。

第21堂课

图形绘制技术

（视频讲解：74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。它们是程序开发不可缺少的技术，使用这些技术可以为程序提供数据统计、图表分析等功能，并可以提高程序的交互能力。本堂课将向读者介绍绘图技术的基本知识以及图像处理技术。

学习摘要：

- Java 的绘图类
- 控件绘图上下文的获取
- 绘图颜色与笔画的设置
- 绘图文字属性的设置
- 绘制文字和图像
- 控件重绘的方法

21.1 Graphics 简介

绘图是高级程序设计中非常重要的技术，如应用程序需要绘制闪屏图片、绘制程序背景、绘制控件外观，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，由于实际使用的绘图上下文基本都是 Graphics2D 类，所以本节如果没有特殊说明的话统称为 Graphics。

21.1.1 Graphics

Graphics 类是所有图形上下文的抽象父类，它允许应用程序在控件、图片对象以及闪屏图像上进行绘制。Graphics 类封装了 Java 支持的基本绘图操作所需的状态信息，这些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，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。

21.1.2 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，所以本节主要介绍如何使用 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。*

21.1.3 获得控件的 Graphics2D 对象

Graphics 绘图上下文在 Java 语言中负责控件、图像与图形的绘制，单独创建的 Graphics 上下文对象没有任何意义，它必须从绘制目的地获取 Graphics 对象。

项目开发时，GUI 设计会根据不同的业务内容出现各种需求，这些需求可能是现有控件无法满足的，因此需要在控件上绘制文本、图像等操作改变控件。Swing 的控件对象本身不提供这些操作的方法，但是它提供了获取控件绘图上下文对象的 getGraphics()方法。该方法声明如下：

```
public Graphics getGraphics()
```

为控件创建一个图形上下文。如果控件当前是不可显示的，则此方法返回 null。也就是说，执行 setVisible()方法或者默认就是显示状态的控件调用该方法会获取绘图上下文对象，否则会取 null 空值。

Swing 的控件基本上都是使用 Graphics2D 类的实例对象进行界面绘制的，它拥有更加丰富的属性和绘图功能。但是 getGraphics()方法获取的却是 Graphics 类的实例对象，为了使用 Graphics2D 的丰富功能，需要把获取的绘图上下文对象强制类型转换为 Graphics2D 类型。

例 21.01 获取窗体的 Graphics2D 对象，并在标签上输出绘图上下文中的字体、前景色、背景色属性。

实例的关键代码如下：**（实例位置：光盘\mr\ym\21.01）**

```
public Graphics2DDemo() {
    setTitle("Graphics2D 简单功能演示");                          // 设置窗体的标题
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);              // 设置窗体退出时操作
    setBounds(100, 100, 450, 100);                               // 设置窗体位置和大小
    contentPane = new JPanel();                                  // 创建内容面板
    contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));          // 设置面板的边框
    setContentPane(contentPane);                                 // 应用内容面板
    contentPane.setLayout(new GridLayout(3, 1, 5, 5));           // 设置内容面板为网格布局
    JLabel label1 = new JLabel("");                              // 创建标签
    contentPane.add(label1);                                     // 使用标签
    JLabel label2 = new JLabel("");                              // 创建标签
    contentPane.add(label2);                                     // 使用标签
    JLabel label3 = new JLabel("");                              // 创建标签
    contentPane.add(label3);                                     // 使用标签
    setVisible(true);                                            // 设置窗体可见
    Graphics2D graphics = (Graphics2D) getGraphics();            // 获得 Graphics2D 对象
    label1.setText("绘图前景色：" + graphics.getColor());          // 在标签中显示绘图前景色
    label2.setText("绘图背景色：" + graphics.getBackground());     // 在标签中显示绘图背景色
    label3.setText("上下文字体：" + graphics.getFont());           // 在标签中显示上下文字体
}
```

实例的运行效果如图 21.1 所示。

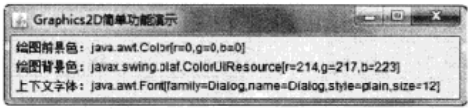


图 21.1　实例运行效果

注意：需要先将窗体设置成可见的，才能使用 getGraphics()方法，否则会出现空指针异常。

21.2　设置颜色与笔画

颜色与笔画是绘图的重要属性，这决定了使用什么颜色绘制图形，并指定笔画的粗细和样式等。本节将介绍这两个绘图属性。

21.2.1　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的支持与否，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 对象的常用构造方法介绍如下。

☑　使用十六进制整数指定红、绿、蓝三原色。该方法声明如下：

```
public Color(int rgb)
```

参数说明：rgb 为 6 位十六进制整数，前两位表示红色分量，中间两位表示绿色，最后两位表示蓝色，范围都是 00～FF。

📖 说明：需要使用 0X 或 0x 来表示十六进制数。

☑ 使用十进制整数分别指定红、绿、蓝三原色。该方法声明如下：

```
public Color(int r,int g,int b)
```

参数说明：r 表示红色，g 表示绿色，b 表示蓝色，范围都是 0～255。

Color 类定义了常用色彩的常量值，如表 21.1 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的颜色对象。

表 21.1 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 LIGHT_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。该方法的声明如下：

```
void setColor(Color c)
```

参数说明：c 是一个颜色对象。

📢 注意：设置绘图颜色以后，再进行绘图或者绘制文本，它们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，需要再次调用 setColor()方法设置其他颜色。

21.2.2 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，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风格等样式。该方法的声明如下：

```
void setStroke(Stroke s)
```

参数说明：s 是 Stroke 接口的实现类。

BasicStroke 类实现了 Stroke 接口，用于绘制图形的轮廓。该类提供了很多的构造方法，现选取最复杂的进行讲解。该方法声明如下：

```
public BasicStroke(float width,int cap,int join,float miterlimit,float[] dash,float dash_phase)
```

方法中的参数说明如表 21.2 所示。

表 21.2　BasicStroke 类的构造方法参数

参 数 名	说　明
width	BasicStroke 的宽度。此宽度必须大于或等于 0.0f。如果将宽度设置为 0.0f，则将笔画呈现为可用于目标设备和抗锯齿提示设置的最细线条
cap	BasicStroke 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。miterlimit 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 个常量对线端点的装饰效果如图 21.2 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 21.3 所示。

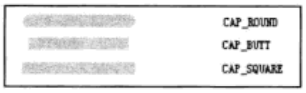


图 21.2　cap 参数对线端点的装饰效果

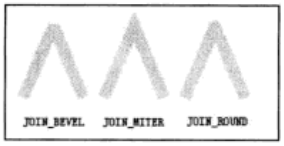


图 21.3　join 参数修饰线段交汇的效果

21.3　如何绘制文本

Java 绘图类可以绘制文本内容，在绘制文本之前可以设置使用的字体、字符大小等。本节将介绍如何设置文本的字体以及如何绘制文本。

21.3.1　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。该方法声明如下：

```
public Font(String name,int style,int size)
```

参数说明：name 表示字体的名称，style 表示字体的样式，size 表示字体的大小。

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 21.4 所示。

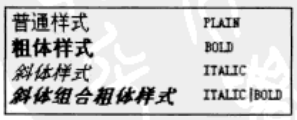


图 21.4　字体样式

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。该方法声明如下：

```
setFont(Font font)
```

参数说明：font 是 Font 类的字体对象。

21.3.2 绘制文本

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，从而实现绘制文字的功能。该方法的声明如下：

```
drawString(String str, int x, int y);或 drawString(String str, float x, float y)
```

参数说明：str 是要绘制的文本字符串，x 是绘制字符串的水平起始位置，y 是绘制字符串的垂直起始位置。

这两个方法唯一不同的就是使用的 x 和 y 参数的类型不同，前者是 int 类型参数，而后者是 float 类型参数。方法中的 x 和 y 参数指的是字符串左下角的坐标，而不是左上角的，如图 21.5 所示。

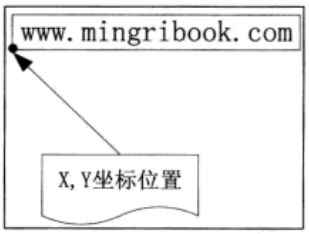


图 21.5 绘制文本的坐标点

这也许和其他语言绘制文本的方式不太一样，绘制的字符串的高度就是单个文字的高度，可以通过字体大小来确定，但是绘制的字符串由多个字符组成，其中还有字符间距问题，导致这个字符串的宽度计算比较复杂。

Font 类的 getStringBounds()方法可以获取字符串需要占据的边界大小。该方法的声明如下：

```
public Rectangle2D getStringBounds(String str, FontRenderContext frc)
```

参数说明：str 是需要计算边界大小的字符串，frc 是指定的 FontRenderContext。

该方法返回 Rectangle2D 类的 2D 矩形对象，该对象中的宽度与高度就是字符串在绘图上下文中实际占据的边界大小。

FontRenderContext 类是正确测量文本所需的信息容器。因为将轮廓映射到像素的规则不同，而且应用程序提供的呈现提示不同，所以文本的测量也有所不同。可以从 Graphics 绘图上下文中获取该类的实例对象。

例 21.02 在窗体上绘制字符串“明日科技”和该字符串的矩形边界。实例代码如下：**（实例位置：光盘\mr\ym\21.02）**

```
public class DrawStringDemo extends JFrame {
    private static final long serialVersionUID = -4363922318457227027L;
    private JPanel contentPane;
    public static void main(String[] args) {
        try {
            UIManager.setLookAndFeel("com.sun.java.swing.plaf.nimbus.NimbusLookAndFeel");
        } catch (Throwable e) {
            e.printStackTrace();
        }
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                try {
```

```
                    DrawStringDemo frame = new DrawStringDemo();        // 创建窗体对象
                    frame.setVisible(true);                             // 将窗体设置成可见
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }
    public DrawStringDemo() {
        addWindowListener(new WindowAdapter() {                         // 增加窗体激活事件监听器
            @Override
            public void windowActivated(WindowEvent e) {
                do_this_windowActivated(e);
            }
        });
        setTitle("绘制字符串");                                          // 设置窗体的标题
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);                 // 设置窗体退出时操作
        setBounds(100, 100, 250, 100);                                  // 设置窗体位置和大小
        contentPane = new JPanel();                                     // 创建内容面板
        contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));             // 设置面板的边框
        contentPane.setLayout(new BorderLayout(0, 0));                  // 设置内容面板为边框布局
        setContentPane(contentPane);                                    // 应用内容面板
    }
    protected void do_this_windowActivated(WindowEvent e) {
        Graphics2D graphics = (Graphics2D) getGraphics();               // 获得 Graphics2D 对象
        String str = "明日科技";                                         // 声明绘制的字符串
        graphics.drawString(str, 50, 50);                               // 在指定位置绘制文本
        FontRenderContext frc1 = graphics.getFontRenderContext();       // 获取字体渲染信息
        Rectangle2D sb1 = graphics.getFont().getStringBounds(str, frc1); // 获取字符串的边界
        graphics.drawRect(50, 50, (int) sb1.getWidth(), (int) sb1.getHeight()); // 使用字符串的坐标绘制矩形边界
        graphics.drawString(str, 180, 50);                              // 在指定位置绘制文本
        FontRenderContext frc2 = graphics.getFontRenderContext();       // 获取字体渲染信息
        Rectangle2D sb2 = graphics.getFont().getStringBounds(str, frc2); // 获取字符串的边界
        int width = (int) sb2.getWidth();                               // 获取字符串边界的宽度
        int height = (int) sb2.getHeight();                             // 获取字符串边界的高度
        graphics.drawRect(180, 50 - height, width, height);             // 使用字符串的坐标绘制矩形边界

    }
}
```

实例的运行效果如图 21.6 所示。

从实例的运行结果中可以看出，左半部在绘制矩形边界时，与字符串使用了相同的起始坐标，但是它和字符串没有重合在一起，这进一步说明了字符串的起始坐标是从左下角开始的，而其他绘图方法（如绘制矩形边界的方法）是从左上角开始绘制文本的，这点需要读者注意。

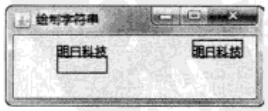


图 21.6　实例运行效果

窗体右半部的矩形边界用起始坐标的纵向坐标减去字符串的高度达到与字符串重合的效果。

21.4 在控件上绘图

前面的章节都是使用控件的 getGraphics()方法获取绘图上下文，然后在事件中利用这个绘图上下文完成绘制任务。这种方式存在一个弊端，就是每当改变窗体大小和其他状态时，在窗体中绘制的内容就不见了，这是因为控件有自己的外观绘图机制，该机制会调用控件中的 paint()方法执行外观绘制。所以继承指定控件编写自己的控件类，并重写 paint()方法就可以在控件上绘制各种内容了，在窗体状态改变需要重新绘制外观时，会自动调用 paint()方法。

例 21.03 在窗体上绘制艺术字“明日科技”。实例的关键代码如下：**（实例位置：光盘\mr\ym\21.03）**

```
public void paint(Graphics g) {
    super.paint(g);                                                          // 调用父类构造方法
    Graphics2D g2 = (Graphics2D) g;                                          // 获得 Graphics2D 对象
    String str = "明日科技！";                                                // 设置要显示的字符串
    Font font = new Font("隶书", Font.BOLD | Font.ITALIC, 40);               // 创建新字体
    g2.setFont(font);                                                        // 应用字体
    for (int i = 0; i < str.length(); i++) {
        g2.setColor(Color.GRAY);                                             // 设置前景色
        g2.drawString(str.charAt(i) + "", 50 + i * font.getSize(), 100 + i * font.getSize()); // 在指定位置绘制文本
        g2.drawString(str.charAt(i) + "", 370 - i * font.getSize(), 100 + i * font.getSize()); // 在指定位置绘制文本
    }
    for (int i = 0; i < str.length(); i++) {
        g2.setColor(Color.MAGENTA);                                          // 设置背景色
        g2.drawString(str.charAt(i) + "", 40 + i * font.getSize(), 90 + i * font.getSize());  // 在指定位置绘制文本
        g2.drawString(str.charAt(i) + "", 360 - i * font.getSize(), 90 + i * font.getSize());  // 在指定位置绘制文本
    }
}
```

实例的运行效果如图 21.7 所示。

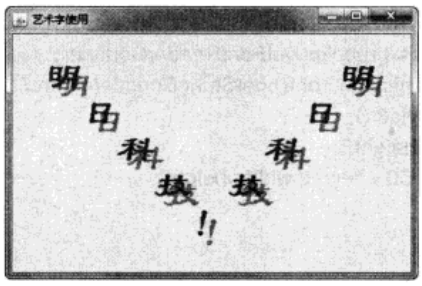


图 21.7 实例运行效果

21.5 绘 制 图 形

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，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。Graphics 类常用的图形绘制方法如表 21.3 所示。

表 21.3　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);	
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	fillRoundRect(10, 10, 50, 30,10,10);	

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，推荐使用绘图类。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。方法声明如下：

draw(Shape form)或 fill(Shape form)

参数说明：form 是实现了 Shape 接口的对象。

java.awt.geom 包中提供了一些常用的图形类，它们都实现了 Shape 接口，它们包括：

- ☑ 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，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 21.04 在窗体上绘制 5 个圆圈。各个圆圈的位置模仿奥运会的标志。实例的关键代码如下：**（实例位置：光盘\mr\ym\21.04）**

```
public void paint(Graphics g) {
    super.paint(g);
    g.drawOval(50, 50, 100, 100);          // 绘制第 1 个圆形
    g.drawOval(130, 50, 100, 100);         // 绘制第 2 个圆形
    g.drawOval(210, 50, 100, 100);         // 绘制第 3 个圆形
    g.drawOval(100, 120, 100, 100);        // 绘制第 4 个圆形
    g.drawOval(180, 120, 100, 100);        // 绘制第 5 个圆形
}
```

实例的运行效果如图 21.8 所示。

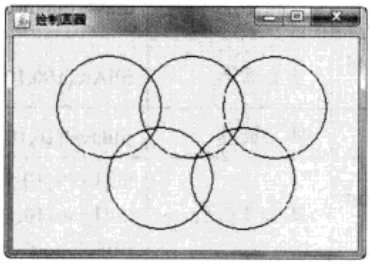


图 21.8 实例运行效果

21.6 照猫画虎——基本功训练

21.6.1 基本功训练 1——绘制图像

视频讲解：光盘\mr\lx\21\绘制图像.exe

实例位置：光盘\mr\ym\21.05\src

除了可以使用标签来显示图片外，还可以使用 Graphics 类的 drawImage()方法绘制图像。其优势是可以将绘制的图像作为窗体的背景，并在其上增加其他控件。本实例将演示如何在窗体上绘制图像。

实现过程如下：

（1）在 Eclipse 中创建 JFrame 窗体，名称为 DrawImageDemo。

（2）重写 JLabel 类的 paint()方法，完成后的代码如下：

```
public void paint(Graphics g) {
    ImageIcon icon = new ImageIcon("src/images/image.jpg");          // 获得图标
    Image image = icon.getImage();                                   // 获得图像
    int height = icon.getIconHeight();                               // 获得图像高度
```

```
    int width = icon.getIconWidth();                    // 获得图像宽度
    setSize(width, height+30);                          // 设置窗体的大小刚好容纳整个图像
    g.drawImage(image, 0, 30, null);                    // 绘制图像
}
```

实例的运行效果如图 21.9 所示。

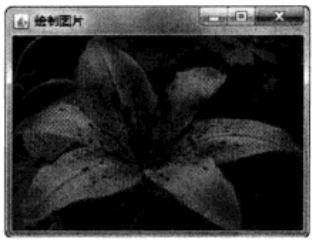


图 21.9　实例运行效果

技巧：窗体的坐标是包括顶部标题栏的，其宽度是 30 像素。因此将图像的显示位置下移 30 像素就可以让图片上部完全显示。

你问我答：如何使用 drawImage()方法绘制图像？

为了绘制图像，首先要获得图像，即获得 Image 类型的对象。使用 ImageIcon 类的 getImage()方法可以获得该对象。其次，需要使用 Graphics 类的 drawImage()方法将图像绘制到窗体上。该方法的声明如下：

```
public abstract boolean drawImage(Image img,int x,int y,ImageObserver observer)
```

该方法中的参数说明如表 21.4 所示。

表 21.4　drawImage()方法的参数说明

参 数 名	说　明
img	需要绘制的图像
x	图像顶点的 x 坐标
y	图像顶点的 y 坐标
observer	图像的观察者

照猫画虎：模仿该实例，给标签绘制图片。提示：通过重写 paint()方法实现。（20 分）（实例位置：光盘\mr\ym\21.05\zmhh）

21.6.2　基本功训练 2——缩放图像

视频讲解：光盘\mr\lx\21\缩放图像.exe

实例位置：光盘\mr\ym\21.06\src

Java 的绘图技术中提供了对图片的缩放功能，这对于用户使用是非常方便的。本实例将演示如何缩放图像。

实现过程如下：

（1）在 Eclipse 中创建 JFrame 窗体，名称为 ZoomImageDemo。

（2）重写 JLabel 类的 paint()方法，完成后的代码如下：

```
public class ZoomImageDemo extends JFrame {
    private static final long serialVersionUID = -5978743533997979745L;
    private JPanel contentPane;
    private JLabel label;
    public static void main(String[] args) {
        try {
            UIManager.setLookAndFeel("com.sun.java.swing.plaf.nimbus.NimbusLookAndFeel");
        } catch (Throwable e) {
            e.printStackTrace();
        }
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                try {
                    ZoomImageDemo frame = new ZoomImageDemo();
                    frame.setVisible(true);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }
    public ZoomImageDemo() {
        setTitle("缩放图像");                                          // 设置窗体的标题
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);               // 设置窗体退出时操作
        setBounds(100, 100, 450, 300);                                // 设置窗体的位置和大小
        contentPane = new JPanel();                                   // 创建内容面板
        contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));           // 设置面板的边框
        contentPane.setLayout(new BorderLayout(0, 0));                // 设置内容面板为边界布局
        setContentPane(contentPane);                                  // 应用内容面板
        JPanel panel = new JPanel();                                  // 创建面板
        contentPane.add(panel, BorderLayout.SOUTH);                   // 将新面板放置在内容面板南面
        final JSlider slider = new JSlider();                         // 创建滑块
        slider.setSnapToTicks(true);                                  // 设置滑块自动靠近最近的刻度值
        slider.setMajorTickSpacing(25);                               // 将主刻度间隔设置成 25
        slider.setPaintLabels(true);                                  // 显示滑块标签
        slider.setPaintTicks(true);                                   // 显示滑块刻度值
        slider.setMinimum(0);                                         // 设置滑块的最小值为 0
        slider.setMaximum(100);                                       // 设置滑块的最大值为 100
        Dictionary<Integer, Component> sliderLabel = new Hashtable<Integer, Component>();
                                                                      // 创建滑块标签集合
        sliderLabel.put(0, new JLabel("0%"));                         // 为 0 刻度增加标签 0%
        sliderLabel.put(25, new JLabel("25%"));                       // 为 25 刻度增加标签 25%
        sliderLabel.put(50, new JLabel("50%"));                       // 为 50 刻度增加标签 50%
        sliderLabel.put(75, new JLabel("75%"));                       // 为 75 刻度增加标签 75%
        sliderLabel.put(100, new JLabel("100%"));                     // 为 100 刻度增加标签 100%
        slider.setLabelTable(sliderLabel);                            // 应用滑块标签
        slider.addChangeListener(new ChangeListener() {               // 为滑块增加事件监听器
            public void stateChanged(ChangeEvent e) {
                do_slider_stateChanged(e);
            }
```

```
        });
        panel.setLayout(new BorderLayout(0, 0));                    // 设置新面板的布局为边界布局
        panel.add(slider);                                          // 在新面板上增加滑块
        label = new JLabel("") {
            private static final long serialVersionUID = 2310809088590940991L;
            @Override
            public void paint(Graphics g) {
                ImageIcon icon = new ImageIcon("src/images/image.jpg");  // 获得图标
                Image image = icon.getImage();                      // 获得图像
                g.clearRect(0, 0, this.getWidth(), this.getHeight());  // 清除绘图上下文的内容
                int value = slider.getValue();                      // 获得滑块值
                int newHeight = icon.getIconHeight() * value / 100;  // 计算新的图像高度
                int newWidth = icon.getIconWidth() * value / 100;   // 计算新的图像宽度
                g.drawImage(image, 0, 0, newWidth, newHeight, null);  // 绘制图像
            }
        };
        contentPane.add(label, BorderLayout.CENTER);                // 在内容面板中央增加标签
    }
    protected void do_slider_stateChanged(ChangeEvent e) {
        label.repaint();                                            // 滑动滑块时重新绘制标签
    }
}
```

实例的运行效果如图 21.10 所示。

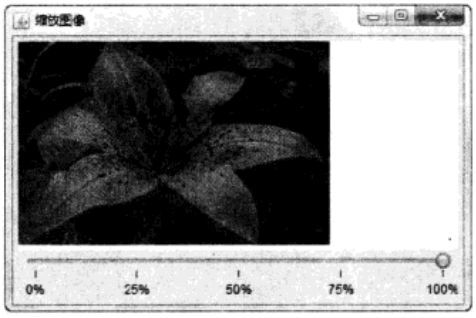


图 21.10　实例运行效果

你问我答：如何使用 drawImage()方法缩放图像？

为了缩放图像，首先要绘制图像。其次，需要使用 Graphics 类的 drawImage()方法修改图片的宽度和高度，以此实现缩放功能。该方法的声明如下：

```
public abstract boolean drawImage(Image img,int x,int y,int width,int height,ImageObserver observer)
```

该方法中的参数说明如表 21.5 所示。

表 21.5　drawImage()方法的参数说明

参 数 名	说　明
img	需要绘制的图像
x	图像顶点的 x 坐标
y	图像顶点的 y 坐标

续表

参 数 名	说 明
width	图像的宽度
height	图像的高度
observer	图像的观察者

照猫画虎：模仿该实例，使用按钮实现动态缩放图片。（20 分）（实例位置：光盘\mr\ym\21.06\zmhh）

21.6.3 基本功训练 3——旋转图像

视频讲解：光盘\mr\lx\21\旋转图像.exe

实例位置：光盘\mr\ym\21.07\src

在使用动画效果显示图片时，会遇到旋转图片的情况。Java 对此提供了很好的支持，只输入需要转动的角度即可。本实例将演示如何旋转图像。

实现过程如下：

（1）在 Eclipse 中创建 JFrame 窗体，名称为 RotateImageDemo。

（2）重写 JLabel 类的 paint()方法，完成后的代码如下：

```
label = new JLabel("") {
    private static final long serialVersionUID = 8885169965910681721L;
    @Override
    public void paint(Graphics g) {
        ImageIcon icon = new ImageIcon("src/images/image.jpg");          // 获得图标
        Image image = icon.getImage();                                   // 获得图像
        Graphics2D g2 = (Graphics2D) g;                                  // 获得 Graphics2D 对象
        int value = slider.getValue();                                   // 获得滑块值
        g2.rotate(Math.toRadians(value));                                // 将图片旋转 value 度
        g2.drawImage(image, 0, 0, null);                                 // 绘制图像

    }
};
```

实例的运行效果如图 21.11 所示。

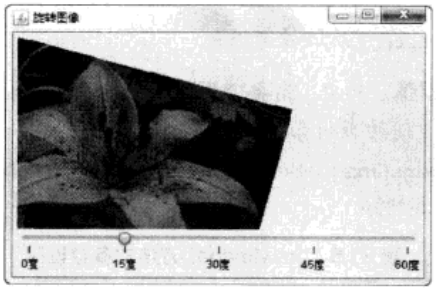


图 21.11 实例运行效果

你问我答：rotate()方法有什么作用？

使用 Graphics2D 类的 rotate()方法可以实现图片的旋转。该方法的声明如下：

public abstract void rotate(double theta)
参数说明：theta 是顺时针方向旋转的角度。

照猫画虎：模仿该实例，使用按钮实现旋转图片。（20 分）（实例位置：光盘\mr\ym\21.07\zmhh）

21.6.4　基本功训练 4——倾斜图像

视频讲解：光盘\mr\lx\21\倾斜图像.exe
实例位置：光盘\mr\ym\21.08\src

除了对图像进行缩放、旋转，还可以对其进行倾斜，如将一个矩形的图像变成平行四边形显示。本实例将倾斜显示图像。

实现过程如下：

（1）在 Eclipse 中创建 JFrame 窗体，名称为 ShearImageDemo。

（2）重写 JLabel 类的 paint()方法，完成后的代码如下：

```
label = new JLabel("") {
    private static final long serialVersionUID = 8885169965910681721L;
    @Override
    public void paint(Graphics g) {
        ImageIcon icon = new ImageIcon("src/images/image.jpg");      // 获得图标
        Image image = icon.getImage();                               // 获得图像
        Graphics2D g2 = (Graphics2D) g;                              // 获得 Graphics2D 对象
        int value = slider.getValue();                               // 获得滑块值
        g2.shear((double) value / 10, 0);                            // 将图片向水平方向倾斜
        g2.drawImage(image, 0, 0, null);                             // 绘制图像
    }
};
```

实例的运行效果如图 21.12 所示。

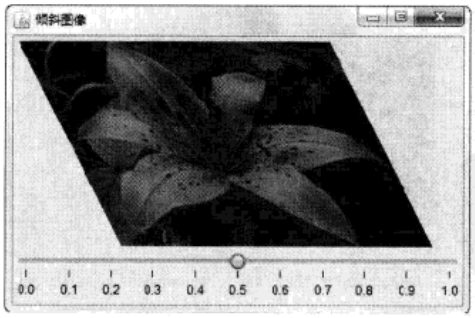


图 21.12　实例运行效果

你问我答：shear()方法有什么作用？

使用 Graphics2D 类的 shear()方法可以实现图片的旋转。该方法的声明如下：

public abstract void shear(double shx,double shy)
参数说明：shx 是在正 X 轴方向移动坐标的乘数，它可以作为其 Y 坐标的函数；shy 是在正 Y 轴方向移动坐标的乘数，它可以作为其 X 坐标的函数。

照猫画虎：模仿该实例，实现图像的纵向倾斜。（20 分）（实例位置：光盘\mr\ym\21.08\zmhh）

21.6.5 基本功训练 5——翻转图像

视频讲解：光盘\mr\lx\21\翻转图像.exe

实例位置：光盘\mr\ym\21.09\src

在处理图片的过程中，会使用到翻转图像的功能。Java 对此也提供了支持，本实例将演示如何实现对图像的水平翻转的功能。

实现过程如下：

（1）在 Eclipse 中创建 JFrame 窗体，名称为 ReverseImageDemo。

（2）重写 JLabel 类的 paint()方法，完成后的代码如下：

```
label = new JLabel("") {
    private static final long serialVersionUID = 8885169965910681721L;
    @Override
    public void paint(Graphics g) {
        ImageIcon icon = new ImageIcon("src/images/image.jpg");          // 获得图标
        Image image = icon.getImage();                                   // 获得图像
        if (reverse) {
            g.drawImage(image, 0, 0, icon.getIconWidth(),icon.getIconHeight(), icon.getIconWidth(), 0, 0,
                    icon.getIconHeight(), null);                         // 绘制图像
        } else {
            g.drawImage(image, 0, 0, null);                              // 绘制图像
        }

    }
};
```

实例的运行效果如图 21.13 所示。

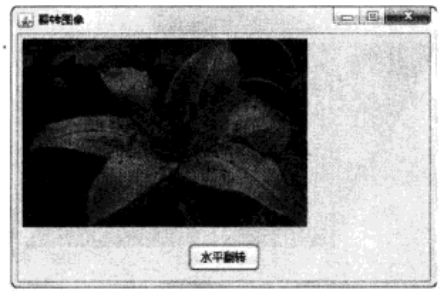


图 21.13 实例运行效果

你问我答： 如何使用 drawImage()方法实现图片翻转？

使用 Graphics 类的 drawImage ()方法可以实现图片的翻转。它是通过置换图片不同顶点的坐标实现的。该方法的声明如下：

```
public abstract boolean drawImage(Image img,int dx1,int dy1,int dx2,int dy2,int sx1,int sy1,int sx2,int
sy2,ImageObserver observer)
```

该方法中的参数说明如表 21.6 所示。

表 21.6　drawImage()方法的参数说明

参 数 名	说　明
img	需要绘制的图像
dx1	目标矩形第一个角的 x 坐标
dy1	目标矩形第一个角的 y 坐标
dx2	目标矩形第二个角的 x 坐标
dy2	目标矩形第二个角的 y 坐标
sx1	源矩形第一个角的 x 坐标
sy1	源矩形第一个角的 y 坐标
sx2	源矩形第二个角的 x 坐标
sy2	源矩形第二个角的 y 坐标
observer	当缩放并转换了更多图像时要通知的对象

说明：第一个角是矩形的左上角，第二个角是矩形的右下角。

照猫画虎：模仿该实例，实现图像的垂直翻转。（20 分）（实例位置：光盘\mr\ym\21.09\zmhh）

照猫画虎栏目分数统计：

照猫画虎题目	1	2	3	4	5	总分数	
分数							

21.7　情景应用——拓展与实践

21.7.1　情景应用 1——绘制直方图

视频讲解：光盘\mr\lx\21\绘制直方图.exe

实例位置：光盘\mr\ym\21.10\src

在使用 Excel 等具有统计功能的软件时，可以为选择的一组数据绘制直方图。这样可以对数据进行简单的处理，如可以方便地比较一个季度中各个产品的销量。使用 Java 语言也能实现这个功能。本实例将在一个窗体中绘制一个直方图，来表示两种产品的销量。实例的运行效果如图 21.14 所示。

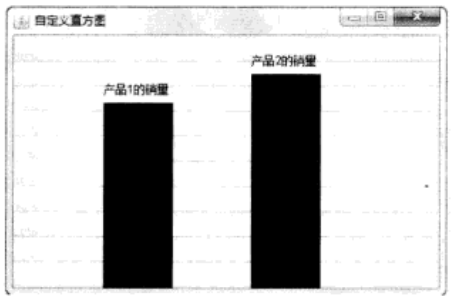


图 21.14　实例运行效果

实现过程如下：

（1）在 Eclipse 中新建项目，并创建窗体类 HistogramDemo。

（2）重写 HistogramDemo 类继承的 paint()方法，在该方法中完成绘制直方图的操作。它可以分成两部分：一个是直方图的背景，显示销售比例；另一个是矩形块，显示产品的销量。完成后的代码如下：

```
public void paint(Graphics g) {
    g.setColor(Color.WHITE);                                          // 将画笔颜色设置成白色
    g.fillRect(0, 0, getWidth(), getHeight());                        // 使用白色填充整个面板
    Insets inset = getInsets();                                       // 获得可用的绘图区域
    g.setColor(Color.LIGHT_GRAY);                                     // 将画笔颜色设置成浅灰色
    int usefulWidth = getWidth() - inset.right;                       // 可用宽度
    int usefulHeight = getHeight() - inset.top - inset.bottom;        // 可用高度
    int lineHeight = usefulHeight / 10;                               // 计算百分比线间距离
    for (int i = 1; i < 11; i++) {
        Point start = new Point(inset.left, inset.top + lineHeight * i);   // 计算百分比线起点
        Point end = new Point(usefulWidth, inset.top + i * lineHeight);    // 计算百分比线终点
        g.drawString(100 - i * 10 + "%", start.x, start.y);           // 绘制字符串
        g.drawLine(start.x, start.y, end.x, end.y);                   // 绘制百分比线
    }
    g.setColor(Color.BLUE);                                           // 将画笔颜色设置成蓝色
    g.drawString("产品 1 的销量", 100, 90);                            // 绘制字符串
    g.drawRect(100, 100, 70, usefulHeight - 70);                      // 绘制矩形
    g.fillRect(100, 100, 70, usefulHeight - 70);                      // 为矩形填充颜色
    g.drawString("产品 2 的销量", 250, 60);                            // 绘制字符串
    g.drawRect(250, 70, 70, usefulHeight - 40);                       // 绘制矩形
    g.fillRect(250, 70, 70, usefulHeight - 40);                       // 为矩形填充颜色
}
```

DIY：在本实例的基础上，让直方图的高度可以随机改变。（20 分）（实例位置：光盘\mr\ym\21.10\diy）

21.7.2 情景应用 2——倾斜效果的文字

视频讲解：光盘\mr\lx\21\倾斜效果的文字.exe

实例位置：光盘\mr\ym\21.11\src

在使用 Word 软件时，其中内置的艺术字功能对于文档的美化非常有用。本实例将使用 Java 语言来实现文字的倾斜效果。其实现思路非常简单，在绘制完文字之后调整画布的方向即可。实例的运行效果如图 21.15 所示。

图 21.15 实例运行效果

实现过程如下：

（1）在 Eclipse 中新建项目，并创建窗体类 ShearTextDemo。

（2）使用一个面板来显示文本信息，这是通过重写 JPanel 类的 paint()方法完成的。在该方法中，首先将方法参数转型为 Graphics2D 类型，它提供了倾斜画布的 shear()方法，最后完成文本信息的绘制。完成后的代码如下：

```
JPanel panel = new JPanel() {
    private static final long serialVersionUID = -910922917238887026L;
    @Override
    public void paint(Graphics g) {
        Graphics2D g2 = (Graphics2D) g;                    // 获得 Graphics2D 对象
        g2.setFont(new Font("华文行楷", Font.BOLD, 72));    // 设置字体
        g2.shear(0.1, -0.4);                               // 倾斜画布
        g2.drawString("明日科技", 30, 200);                 // 绘制文本
    }
};
```

DIY：修改文本为“编程词典”，字体为“微软雅黑”，加粗显示，大小是 72。倾斜方向由左上至右下。（20 分）（实例位置：光盘\mr\ym\21.11\diy）

21.7.3　情景应用 3——纹理填充特效

视频讲解：光盘\mr\lx\21\纹理填充特效.exe

实例位置：光盘\mr\ym\21.12\src

在 Word 软件中，可以对图片实现纹理填充特效。本实例将使用 Java 语言来实现纹理填充效果，它使用一个简单的自定义图形来填充整个窗体。这个简单的图形是一个九宫格，只对对角线上的 5 个正方形进行颜色填充。实例的运行效果如图 21.16 所示。

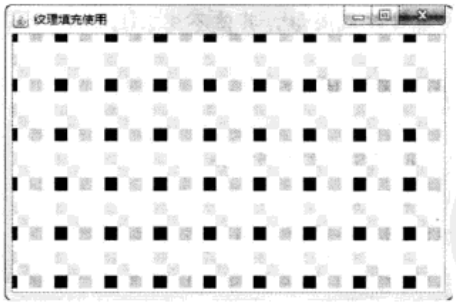


图 21.16　实例运行效果

实现过程如下：

（1）在 Eclipse 中新建项目，并创建窗体类 TextureFillDemo。

（2）重写该类的 paint()方法，完成九宫格中 5 个正方形的绘制，然后将这个大正方形重复地填充到一个窗体中。完成后的代码如下：

```
public void paint(Graphics g) {
    BufferedImage image = new BufferedImage(200, 200,BufferedImage.TYPE_INT_RGB); // 创建缓冲图像对象
    Graphics2D g2 = image.createGraphics();                                        // 获得缓冲图像的 Graphics2D 对象
    g2.setColor(Color.WHITE);                                                      // 将画笔设置成白色
```

```
        g2.fillRect(0, 0, 200, 200);                                    // 使用白色填充整个缓冲图像
        g2.setColor(Color.ORANGE);                                      // 将画笔设置成橙色
        g2.fillRect(0, 0, 50, 50);                                      // 绘制大小为 50 的正方形
        g2.setColor(Color.YELLOW);                                      // 将画笔设置成黄色
        g2.fillRect(100, 0, 50, 50);                                    // 绘制大小为 50 的正方形
        g2.setColor(Color.GREEN);                                       // 将画笔设置成绿色
        g2.fillRect(100, 100, 50, 50);                                  // 绘制大小为 50 的正方形
        g2.setColor(Color.BLUE);                                        // 将画笔设置成蓝色
        g2.fillRect(0, 100, 50, 50);                                    // 绘制大小为 50 的正方形
        g2.setColor(Color.CYAN);                                        // 将画笔设置成青色
        g2.fillRect(50, 50, 50, 50);                                    // 绘制大小为 50 的正方形
        Rectangle2D rect = new Rectangle2D.Double(0, 0, 50, 50);        // 创建矩形对象
        TexturePaint textPaint = new TexturePaint(image, rect);         // 创建纹理填充对象
        Graphics2D graphics2 = (Graphics2D) g;                          // 转换 paint()方法的绘图上下文对象
        graphics2.setPaint(textPaint);                                  // 设置纹理填充对象
        graphics2.fillRect(0, 0, getWidth(), getHeight());              // 使用指定的纹理对象填充窗体
    }
```

你问我答：如何使用 Java 语言实现纹理填充？

（1）创建纹理填充对象。在 Java 中使用 TexturePaint 类可以创建一个纹理填充对象。该类的构造方法如下：

```
public TexturePaint(BufferedImage txtr,Rectangle2D anchor)
```

参数说明：txtr 是具有用于绘制的纹理的 BufferedImage 对象。anchor 是用户空间中用于定位和复制纹理的 Rectangle2D。

（2）应用纹理填充对象。使用 Graphics2D 类的 setPaint()方法可以应用纹理对象。该方法的声明如下：

```
public abstract void setPaint(Paint paint)
```

参数说明：paint 是呈现过程中用于生成颜色的 Paint 对象或者为 null。

DIY：将本实例中的正方形图案修改成圆形，其他不变。（20 分）（实例位置：光盘\mr\ym\21.12\diy）

21.7.4 情景应用 4——简单的校验图片

视频讲解：光盘\mr\lx\21\简单的校验图片.exe

实例位置：光盘\mr\ym\21.13\src

在网页上注册用户时，通常要填写一个表单。为了防止恶意注册，程序会要求用户填写一个校验信息，通常是一个随机生成的图片。本实例将使用 Java 的绘图技术生成一个简单的校验图片，实例的运行效果如图 21.17 所示。

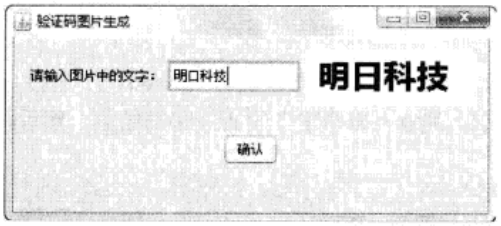


图 21.17 实例运行效果

实现过程如下：

（1）在 Eclipse 中新建项目，并创建窗体类 CAPTCHA。在窗体中增加一个标签、文本框、按钮等控件。

（2）监听窗体激活事件，这是通过 do_this_windowActivated()方法完成的。在该方法中绘制一幅宽度为 200、高度为 100 的图片，图片中文本内容为“明日科技”。然后使用标签来显示绘制的图片。关键代码如下：

```
protected void do_this_windowActivated(WindowEvent e) {
    Image image = createImage(200, 100);                    // 创建新图片并指定其大小
    Graphics2D g = (Graphics2D) image.getGraphics();         // 获得图片的 Graphics 对象并转换其类型
    g.setColor(Color.WHITE);                                 // 设置画笔为白色
    g.fillRect(0, 0, 200, 100);                              // 将图片的背景填充成白色
    g.setColor(Color.BLACK);                                 // 设置画笔为黑色
    g.setFont(new Font("微软雅黑", Font.BOLD, 30));          // 设置文本的字体
    g.drawString("明日科技", 30, 60);                        // 绘制文本信息
    label.setIcon(new ImageIcon(image));                     // 在标签上显示绘制的图片
}
```

（3）监听单击“确认”按钮事件，这是通过 do_button_actionPerformed()方法完成的。在该方法中，对用户在文本框中输入的信息进行校验，并使用对话框显示校验的结果。关键代码如下：

```
protected void do_button_actionPerformed(ActionEvent e) {
    String text = textField.getText();// 获得用户输入的文本
    if(text.equals("明日科技")){// 判断用户输入的文本与“明日科技”是否相同
        JOptionPane.showMessageDialog(this, "验证码输入正确", null, JOptionPane.INFORMATION_MESSAGE);
    }else {
        JOptionPane.showMessageDialog(this, "验证码输入错误", null, JOptionPane.WARNING_MESSAGE);
    }
}
```

DIY：生成字符串长度为 5 的验证码，内容在大小写字母中随机选取。（20 分）（实例位置：光盘\mr\ym\21.13\diy）

21.7.5　情景应用 5——显示彩色字符串

视频讲解：光盘\mr\lx\21\显示彩色字符串.exe

实例位置：光盘\mr\ym\21.14\src

在设计应用程序时，不仅要考虑如何实现功能，还要考虑如何让程序美观。通常可以使用不同的色彩搭配，让程序更加美观。本实例将绘制一个包含字符串的图片，并且每个字符串的颜色都不相同。模仿 Word 中艺术字的效果。实例的运行结果如图 21.18 所示。

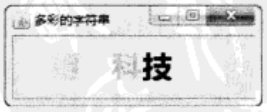


图 21.18　实例运行效果

实现过程如下：

（1）在 Eclipse 中新建项目并创建窗体类 ColorfulTextDemo。

（2）在窗体中增加一个标签并监听窗体激活事件，这是通过 do_this_windowActivated()方法完成的。该方法使用 Graphics 类中的 drawString()方法来绘制字符串。关键代码如下：

```
protected void do_this_windowActivated(WindowEvent e) {
    Image image = createImage(200, 100);                    // 创建新图片并指定其大小
    Graphics2D g = (Graphics2D) image.getGraphics();         // 获得图片的 Graphics 对象并转换其类型
```

```
        g.setColor(Color.WHITE);                                    // 设置画笔为白色
        g.fillRect(0, 0, 200, 100);                                 // 将图片的背景填充成白色
        g.setFont(new Font("微软雅黑", Font.BOLD, 30));              // 设置文本的字体
        String text = "明日科技";                                    // 创建要绘制的字符串
        Color[] colors = { Color.ORANGE, Color.YELLOW, Color.GREEN, Color.BLUE,Color.CYAN }; // 创建颜色数组
        for (int i = 0; i < text.length(); i++) {
            g.setColor(colors[i % 5]);                              // 选择颜色
            g.drawString(text.charAt(i) + "", 30 + i * 30, 60);     // 绘制文本信息
        }
        label.setIcon(new ImageIcon(image));                        // 在标签上显示绘制的图片
    }
```

DIY：生成字符串长度为 5 的多彩字符串，颜色依次为橙色、黄色、绿色、蓝色和青色。内容在大小写字母中随机选取。（20 分）（实例位置：光盘\mr\ym\21.14\diy）

情景应用 DIY 栏目分数统计：

DIY 题目	1	2	3	4	5	总分数	
分数							

21.8 自 我 测 试

一、选择题（每题 10 分，5 道题）

1．Graphics 类和 Graphics2D 类之间的关系是？（　　）

A．Graphics 类是 Graphics2D 类的子类　　B．Graphics 类是 Graphics2D 类的父类

C．Graphics 类是 Graphics2D 类的内部类　　D．Graphics 类是 Graphics2D 类的外部类

2．在创建颜色对象时，使用了如下代码：

```
Color color = new Color(1,2,3);
```

其中的数字 1、2、3 分别表示什么？（　　）

A．1 表示红色分量　B．2 表示绿色分量　　C．3 表示蓝色分量　　D．3 表示黄色分量

3．下列哪些（个）不是 Color 类中预定义的颜色？（　　）

A．RED　　B．ORANGE　　C．PURPLE　　D．GREEN

4．rotate()方法可用于旋转图像，它位于哪个类中？（　　）

A．Graphics 类　　B．Graphics2D 类　　C．Graphics3D 类　　D．Image 类

5．以下哪些类不是 Shape 接口的实现类？（　　）

A．Arc2D　　B．Circle2D　　C．Line2D　　D．Point2D

二、填空题（每题 10 分，5 道题）

1．如果要新建一个字体对象，字体名称是“微软雅黑”，样式是粗体，大小是 15，则代码是？_____

2．使用 Graphics 类的哪个方法可以绘制字符串？（只给出方法名称即可，不需要写参数）_____

3．使用 Graphics 类的哪个方法可以绘制图像？（只给出方法名称即可，不需要写参数）_____

4．使用 Graphics 类的 drawOval()方法可以绘制圆形吗？_____

5．可以使图像倾斜的方法 shear()位于哪个类中？_____

测试分数统计：

类别	第 1 题	第 2 题	第 3 题	第 4 题	第 5 题
选择题分数					
填空题分数					
总分数					

21.9　行 动 指 南

开始日期：_____年____月____日　　结束日期：_____年____月____日

序号	内　容	行 动 指 南	
1	照猫画虎栏目 分数（　　）	分数>75 分	优秀，基本功掌握得不错，加油！
		75 分>分数>50 分	及格，知识掌握得不牢，重新做一遍照猫画虎。
		分数<50 分	请使用光盘中提供的“实战能力测试系统”进行提高训练。
	情景应用栏目 分数（　　）	分数>75 分	优秀，综合应用能力很强。
		75 分>分数>50 分	及格，综合应用能力需提高，再练一遍情景应用。
		分数<50 分	请使用光盘中提供的“实战能力测试系统”进行提高训练。
	自我测试栏目 分数（　　）	分数>75 分	优秀，有成为编程高手的潜质。
		分数<75 分	请使用光盘中提供的“实战能力测试系统”进行提高训练。
	综合评价	反复训练后，以上各项分数都在 75 分以上，方可进入下一堂课学习。	
2	编程实践能力培养：本栏目提供了一些实践题目，对于培养编程能力很有效，要做好。	（1）绘制饼形图。	
		（2）绘制三次曲线。	
		（3）绘制奥运图标。	
		（4）实现图片的水面倒影效果。	
3	编程习惯培养：本堂课培养读者在学习开发中记笔记的习惯，把开发中遇到的问题、总结的经验记录下来。		
4	创新能力培养：看看身边有没有可以用编程解决的问题，记录到右边的表格中。根据学习进度，尝试编写解决这些问题的小程序。		

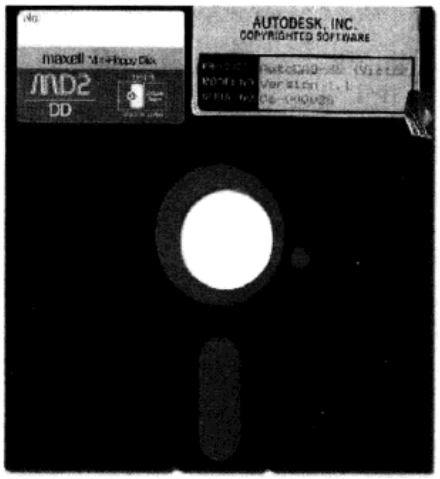
21.10　成功可以复制——3D 王国的开创者

约翰·沃克开始是一个程序员，在生产 UNIVAC（世界上第一台大型商用计算机）的公司里担任系统

程序设计师。那时还没有 PC 和互联网，开发和使用价格昂贵的大型计算机只是少数人的专利。在工作之余，Walker 改良了当时非常受欢迎的 Animal 游戏，改良后的游戏立刻引起热烈回响，很多人都向他索取游戏磁盘，这促使他想出了“游戏自我复制”的游戏传播方式。1975 年 1 月，Walker 写了一个自我复制的程序 Pervade，它可以在游戏启动时自动检查目录，然后自我复制到任何一个尚未有拷贝的目录中，或者覆盖已有的旧版游戏，可以说，Pervade 程序算是病毒的前身。

20 世纪 70 年代，随着微型计算机的出现，有一小部分人意识到这将会成为计算机的主流。怎样才能抓住这个机遇呢？1977 年，约翰·沃克和朋友成立了 Marinchip 公司，试图开发微型计算机硬件系统。但因 IBM 率先推出 PC 机（与其类似的微型计算机硬件），再加上运营资金不足，约翰·沃克放弃了走硬件发展的道路。

面对变化的市场，Walker 决定调整战略，转为向市面上流行的 PC 机操作系统开发应用软件。在尝试了 Basic 编辑器、C 编辑器及数据库软件后，Walker 把目光锁定在绘图程序上。当时，市场上只有适用于大中型计算机的 CAD 绘图软件，且价格昂贵，而微型机上还缺少成熟的 CAD 软件。由于中小企业买不起大型计算机，只能依靠手工绘制工程图样，劳动强度大，工作效率低。



AutoCAD 1.1软件磁盘

1982 年 1 月，约翰·沃克和朋友合伙成立了 Autodesk 公司，购买了别人开发的 Interact 软件，重新改写并丰富其功能。当年 12 月，推出用于 S-100 和 Z-80 计算机的 AutoCAD 1.0 版。从此，AutoCAD 一发而不可收，陆续推出了适用于各种操作系统的版本。Autodesk 公司资产也在短短的 7 年内从最初的不足 6 万美元增长到 6 亿美元，这一切全依赖于约翰·沃克敏锐的洞察力。现在，AutoCAD 已成为世界排名第一的通用工程绘图软件。

越来越重的管理任务让喜欢平静生活的约翰·沃克有些痛苦，1989 年，约翰·沃克交出管理权，之后他离开美国定居瑞士。不过，他现在还保留着编程的习惯，与硅谷最早的黑客群体保持着联系。

在风起云涌的 PC 时代，约翰·沃克成就了自己的传奇。

经典语录

企业的成功是建立在技术卓越的产品上的，如果我们不能成为技术和性能的领导者，我们将无法在市场上生存。

深度评价

Autodesk 公司资产在短短的 7 年内从最初的不足 6 万美元增长到 6 亿美元，这一切全依赖于约翰·沃克敏锐的洞察力，把握住了市场的先机。善于识别与把握先机是极为重要的。互联网时代要想谋求发展，必须要有极强的发现新兴事物、发现现有事物发展方向的个人能力，否则只能跟在别人之后，很难有大的发展。

实战篇

- 第 22 堂课　进销存管理系统
- 第 23 堂课　企业内部通讯管理系统
- 第 24 堂课　图书馆管理系统

第22堂课

进销存管理系统

（视频讲解：162分钟）

进销存管理系统是促进企业发展的重要组成部分，是商业企业经营管理中的核心环节，也是一个企业能否取得效益的关键，如果能够做到合理采购、及时销售、库存量最小、减少积压，那么企业就能取得最佳效益。在现代社会中，大多数企事业单位，特别是中小型企业，实现信息化管理是首要的任务。只有实现信息化管理，才能提高工作效率和企业的管理水平。市场经济快速多变，竞争激烈，企业采用信息化管理进货、库存、销售等诸多环节也已成为趋势及必然。

本堂课将使用Java Swing技术和MySQL数据库开发跨平台的简易通企业进销存管理系统，该系统是典型的MIS（管理信息系统），主要包括创建并维护后台数据库和前端应用程序的开发两个方面。

学习摘要：

- 使用JDBC技术操作MySQL数据库
- 使用Swing技术布局界面
- 使用Swing菜单栏与工具栏
- 使用Swing设计状态栏
- 实现程序数据的自动计算
- 实现多条件查询和模糊查询
- 限制用户的使用权限

22.1 项 目 描 述

企业进销存管理系统可以进行用户名和密码的判断，只有输入正确的用户名和密码才能进入系统，进入系统后，用户可以通过菜单、工具栏和导航按钮实现灵活的人机交互操作。

进入本系统后，用户首先要进行基本信息的录入，如录入供应商基本信息、商品基本信息、客户基本信息和员工基本信息，但是在录入商品基本信息前，必须先录入供应商基本信息，录入完基本信息后，还可以对这些基本信息进行修改、删除和查询等操作。

本系统可以进行入库采购，实现采购签单操作；经过采购签单的商品可以进行入库操作，同时更新库存量，并可以对入库商品进行查询操作，没有经过采购签单的商品不能进行入库操作；对于入库的商品还可以进行退货操作，并同步更新库存量，且能进行退货查询。

本系统还可以进行销售签单操作，只有经过销售签单操作客户才能拿到销售订单，然后根据销售订单进行出库操作，同步更新库存量，并可以对销售出库的商品进行查询操作，没有销售订单商品不能出库；对于销售出库的商品也可以进行退货操作，并同步更新库存量，且能进行退货查询。

本系统还可以进行库存盘点、库存调拨和库存查询等操作，并且系统管理员还可以对用户的权限进行设置，可以把用户设置为管理员或一般用户，设置为管理员权限的用户可以修改密码和查看用户信息，而一般用户只能修改密码而不能查看用户信息。具有系统管理员权限和管理员权限的用户还可以进行添加用户和进行经手人设置等操作，而一般用户不能添加用户，也不能进行经手人设置。

22.2 系 统 分 析

本节将对简易通企业进销存管理系统进行系统的分析，其中包括需求分析、可行性分析。另外，在项目计划书中详细地描述了开发背景、团队和开发环境。

22.2.1 需求分析

简易通企业进销存管理系统的主要工作，是对企业的采购、进货、销售和库存以信息化的方式进行管理。它可以最大限度地减少各个环节中可能出现的错误，能够有效减少盲目采购、降低采购成本、合理控制库存、减少资金占用，并提高市场灵敏度，使企业能够合理安排进销存的每个关键步骤，并能够提升企业市场竞争力。针对这些问题，经销商对产品的进销存合理化提出了更高的要求。概括地讲，用户对进销存系统的需求具有普遍性。

通过实际调查，要求简易通企业进销存管理系统具有以下功能：

- ☑ 界面设计美观大方、方便、快捷、操作灵活。
- ☑ 可以通过导航按钮快速实现人机交互操作。
- ☑ 实现强大的进销存管理，包括基本信息、采购、进货、销售和库存管理。
- ☑ 能够在不同的操作系统下运行，不局限于特定的平台。
- ☑ 提供库存盘点功能。
- ☑ 提供库存调拨功能。
- ☑ 提供了多条件查询和模糊查询。

☑ 提供技术支持的联系方式，可以连接到技术网站。

22.2.2 可行性分析

根据《GB8567－88 计算机软件产品开发文件编制指南》中可行性分析的要求，制定可行性研究报告如下。

1. 引言

☑ 编写目的

为了给软件开发企业的决策层提供是否进行项目实施的参考依据，现以文件的形式分析项目的风险、项目需要的投资与效益。

☑ 背景

XXX 科技有限公司是一家以商品推广为主的商业性企业，为了更好地管理进货、销售和库存，现需要委托其他公司开发一个企业进销存管理系统，项目名称为“简易通企业进销存管理系统”。

2. 可行性研究的前提

（1）要求

☑ 附加进货-退货和销售退货功能以增加管理的灵活性。

☑ 系统的功能要符合本企业的实际情况。

☑ 管理内容较多，涉及窗口容易混乱，应提供窗口集合操作和菜单管理。

☑ 支持管理用户的功能，提高系统安全性。

（2）目标

简易通企业进销存管理系统的主要目标是提供强大的进销存管理功能，减少盲目采购、降低采购成本、合理控制库存、减少资金占用，并提高市场灵敏度。

（3）评价尺度

项目需要在两个月内交付用户使用，系统分析人员需要 3 天内到位，用户需要 7 天时间确认需求分析文档，去除其中可能出现的问题，如用户可能临时有事，占用 5 天时间确认需求分析。那么程序开发人员需要在 50 天的时间内进行系统设计、程序编码、系统测试、程序调试和系统打包部署工作，其间，还包括了员工每周的休息时间。

3. 投资及效益分析

☑ 支出

根据预算，公司计划投入 8 个人，为此需要支付 9 万元的工资及各种福利待遇；项目的安装、调试以及用户培训、员工出差等费用支出需要 2.5 万元；在项目后期维护阶段预计需要投入 2 万元的资金，累计项目投入需要 13.5 万元资金。

☑ 收益

客户提供项目开发资金 30 万元，对于项目后期进行的改动，采取协商的原则，根据改动规模额外提供资金。因此，从投资与收益的效益比上看，公司大致可以获得 16.5 万元的利润。

项目完成后，会给公司提供资源储备，包括技术、经验的积累。

4. 结论

根据上面的分析，在技术上不会存在问题，因此项目延期的可能性很小；在效益上，公司投入 8 个人、

两个月的时间获利 16.5 万元，比较可观；另外，公司还可以储备项目开发的经验和资源。因此，认为该项目可以开发。

22.3 系 统 设 计

22.3.1 系统目标

根据企业对进销存管理的要求，制定简易通企业进销存管理系统应达到如下目标。

- ☑ 灵活的人机交互界面，操作简单方便、界面简洁美观。
- ☑ 可以通过菜单、工具栏或导航按钮灵活操作。
- ☑ 对进货和销售提供相应的退货管理功能。
- ☑ 实现各种查询，如单一条件查询或多条件查询等。
- ☑ 可以随时修改系统口令。
- ☑ 可以设置用户的使用权限。
- ☑ 系统最大限度地实现了易安装性、易维护性和易操作性。
- ☑ 系统运行稳定、安全可靠。

22.3.2 系统功能结构

简易通企业进销存管理系统的功能结构如图 22.1 所示。

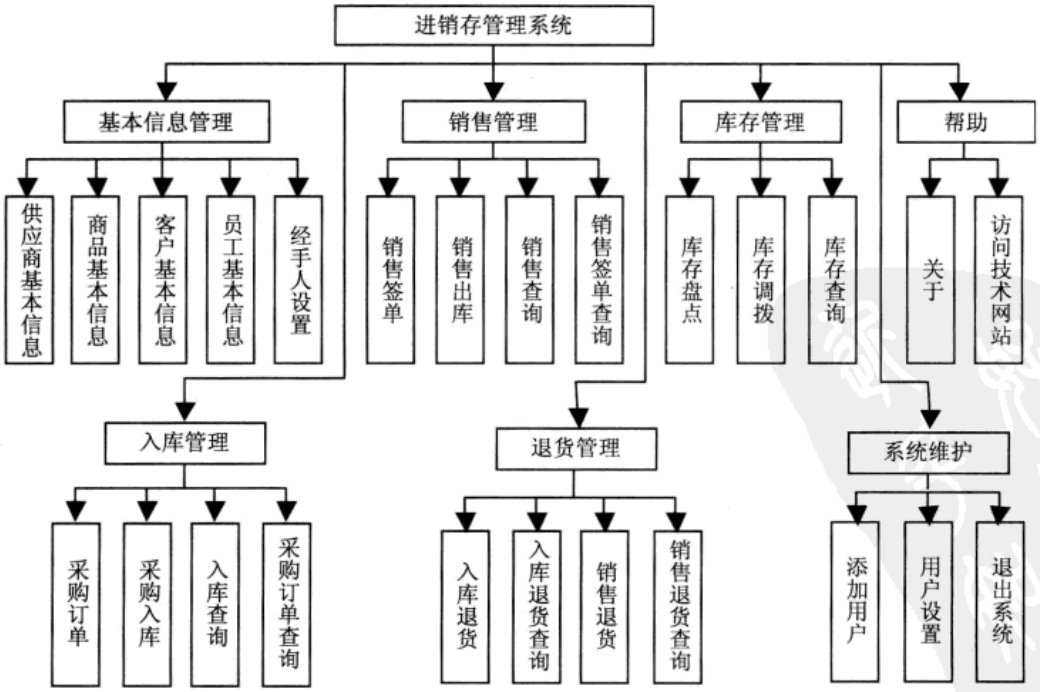


图 22.1 简易通企业进销存管理系统的功能结构图

22.3.3　系统业务流程图

简易通企业进销存管理系统的业务流程图如图 22.2 所示。

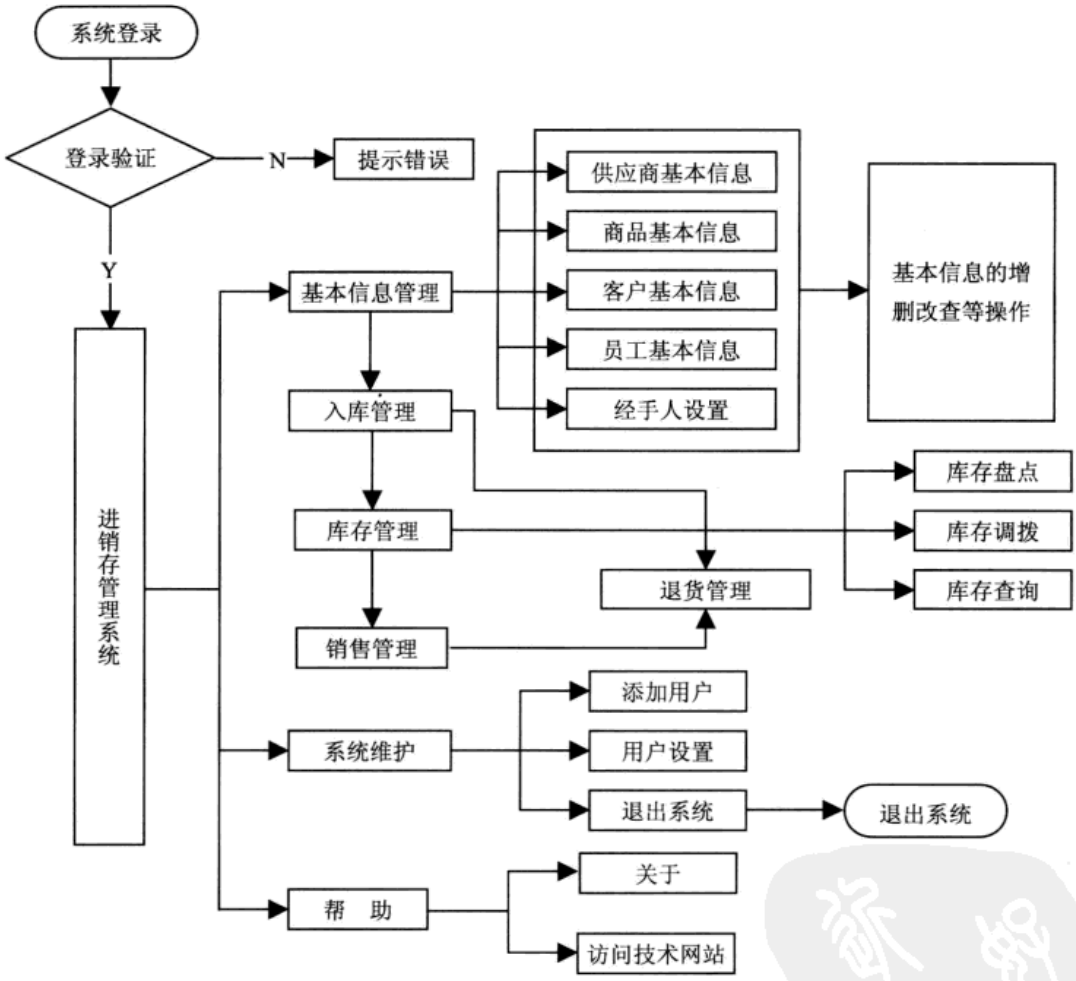


图 22.2　简易通企业进销存管理系统的业务流程图

22.3.4　系统编码规范

开发企业应用程序需要以团队合作来完成，每个人负责不同的业务模块，为了使程序的结构与代码风格统一标准化，增加代码可读性，需要在编码之前制定一套统一的编码规范，下面介绍简易通企业进销存管理系统开发中的编码规范。

1．数据库命名规范

☑　数据库

数据库命名以字母“db_”为前缀（小写），如简易通企业进销存管理系统的数据库名为“jxc”加前缀

“db_”构成数据库名 db_jxc，如表 22.1 所示。

表 22.1 数据库命名

数据库名称	描　述
db_jxc	企业进销存管理系统数据库

☑ 数据表

数据表以字母“tb_”（小写）做前缀，后面加数据表相关缩写做后缀。下面将举例进行说明，如表 22.2 所示。

表 22.2 数据表命名

数据表名称	描　述
tb_gysinfo	供应商信息表
tb_spinfo	商品信息表
tb_jsr	经手人表

☑ 字段

字段采用名称缩写形式命名，没有前缀和后缀限制。下面将举例进行说明，如表 22.3 所示。

表 22.3 字段命名

字 段 名 称	描　述
gysqc	供应商全称
dz	地址

☑ 视图

视图命名以“v_”（小写）做前缀，后面加表示该视图作用的相关英文单词或缩写做后缀。下面将举例进行说明，如表 22.4 所示。

表 22.4 视图命名

视 图 名 称	描　述
v_ruku	入库信息视图
v_sell	销售信息视图

2. 业务编码命名规范

☑ 供应商信息编号

供应商信息的 ID 编号以字符串“gys”做前缀，加上 4 位数字的编号做后缀，编号数字从 1001 开始。例如 gys1004。

☑ 客户信息编号

客户信息的 ID 编号以字符串“kh”为前缀，加上 4 位数字的编号做后缀，编号数字从 1001 开始。例如 kh1002。

☑ 商品信息编号

商品信息的 ID 编号使用字符串“sp”做前缀，加上 4 位数字的编号做后缀，编号数字从 1001 开始。例

如 sp2045。

☑ 员工信息编号

员工信息的 ID 编号使用字符串“yg”做前缀，加上 4 位数字的编号做后缀，编号数字从 1001 开始。例如 yg1005。

☑ 销售签单票号

销售签单票号的 ID 编号命名规则以“XSDD”字符串做前缀，加上销售签单的签单日期和 3 位数字做后缀，编号数字从 001 开始。例如 XSDD20081205001。

☑ 销售票号

销售票号的 ID 编号命名规则以“XS”字符串做前缀，加上商品的销售日期和 3 位数字做后缀，编号数字从 001 开始。例如 XS20071109003。

☑ 销售退货票号

销售退货票号的 ID 编号命名规则以“TH”字符串做前缀，加上商品的销售退货日期和 3 位数字做后缀，编号数字从 001 开始。例如 TH20071109003。

☑ 入库订单票号

入库订单票号的 ID 编号命名规则以“RKDD”字符串做前缀，加上商品的入库订购日期和 3 位数字做后缀，编号数字从 001 开始。例如 RKDD20071109003。

☑ 入库票号

入库票号的 ID 编号命名规则以“RK”字符串做前缀，加上商品的入库日期和 3 位数字做后缀，编号数字从 001 开始。例如 RK20071109003。

☑ 入库退货票号

入库退货票号的 ID 编号命名规则以“TH”字符串做前缀，加上商品的入库退货日期和 3 位数字做后缀，编号数字从 001 开始。例如 TH20071109003。

22.4　系统运行环境

1. 本系统的硬件开发环境

☑ 显示器：液晶 17 寸。
☑ 分辨率：1024×768。
☑ CPU：P4 3.2GHz。
☑ 内存：1024MB 以上。
☑ 硬盘空间：80GB。
☑ 光驱：DVD16X。
☑ 鼠标：光电鼠标。
☑ 键盘：防水键盘。

2. 本系统的软件开发环境

☑ 操作系统：Windows 2003（SP1）。
☑ 数据库：MySQL 5.0。
☑ JDK 环境：Java SE Development KIT(JDK) Version 6。

- ☑ 开发工具：Eclipse 3.4.2。
- ☑ 数据库管理软件：MySQL Administrator 1.2.13。
- ☑ 运行平台：Windows、Linux 各个版本、MAC 等任何平台。
- ☑ 运行环境：Java SE Runtime Environment (JRE) Version 6。
- ☑ 分辨率：最佳效果 1024 像素×768 像素。

22.5 数据库与数据表设计

简易通企业进销存管理系统是典型的（MIS）管理信息系统，数据库是其重要组成部分。该系统的数据库设计是根据需求分析以及系统功能结构定制的。

22.5.1 数据库分析

简易通企业进销存管理系统需要使用数据库存储和管理进销存过程中的所有信息。MySQL Server 数据库系统是目前使用最多的数据库系统，其安全、易用、性能优越、安装和操作简便等特性是受欢迎的主要因素。考虑到进销存数据量的庞大和安全性的保障，本系统决定采用 MySQL Server 5.0 以上版本的数据库系统作为进销存管理的后台数据库，数据库名称为 db_jxc。其中包含了 20 个数据表、8 个视图。

22.5.2 创建数据库

数据库是企业进销存管理系统的数据集合，是系统开发的首要环节。数据库结构设计得好坏直接影响着系统的效率和性能，为了方便数据库的创建，本系统使用 MySQL Administrator 来创建数据库。本系统的数据库名称为 db_jxc，创建完成后的效果如图 22.3 所示。

在 MySQL Administrator 中创建 db_jxc 数据库的具体步骤如下：

（1）选择“开始”/“所有程序”/MySQL/MySQL Administrator 命令，如图 22.4 所示。

（2）在打开的 MySQL Administrator 1.2.8 对话框中如果存在连接，则可以单击下拉按钮，从下拉列表中选择连接，然后添加服务器、端口号、用户名和密码，如图 22.5 所示。单击 OK 按钮，即可进入 MySQL Administrator 窗口。

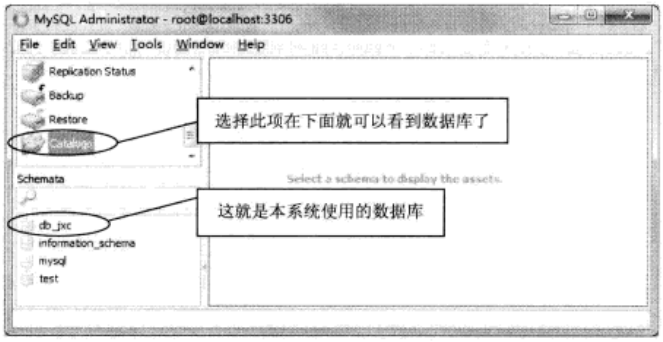


图 22.3 MySQL Administrator 窗口

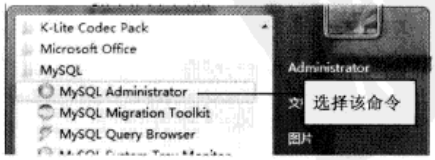


图 22.4 选择 MySQL Administrator 命令

（3）在打开的 MySQL Administrator 1.2.8 对话框中如果不存在连接，则单击□按钮，将打开 Options

对话框，首先选择 Connections 选项，然后单击 New Connection 按钮，在 Connection Parameters 选项卡中输入连接参数，如连接名、用户名、密码、服务器、端口号和计划名（即要连接的数据库名称），最后单击 Apply 按钮，如图 22.6 所示。

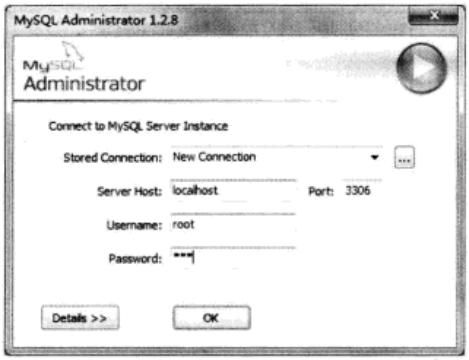


图 22.5　MySQL Administrator 1.2.8 对话框

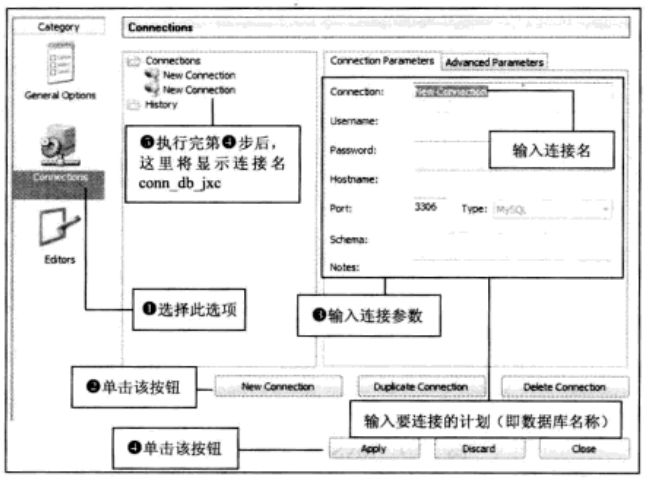


图 22.6　Options 对话框

（4）在打开的 MySQL Administrator 窗体中选择 Catalogst 选项，然后在 Schemata 下面列表区的空白处单击鼠标右键，在弹出的快捷菜单中选择 Create New Schema 命令，如图 22.7 所示。

（5）在 Create new Schema 对话框中输入计划的名称（即数据库名称）“db_jxc”，如图 22.8 所示。单击 OK 按钮，完成数据库的创建。

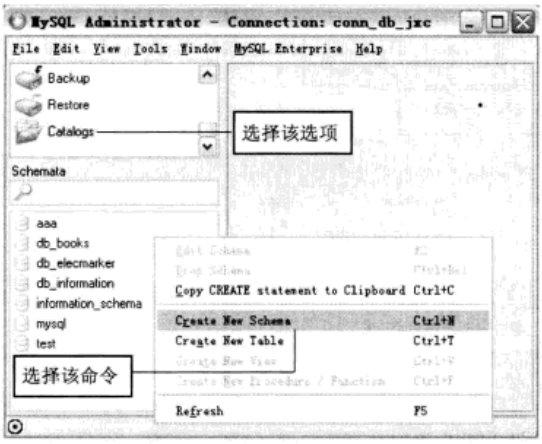


图 22.7　MySQL Administrator 窗口

图 22.8　Create new Schema 对话框

22.5.3　创建数据表

在已经创建的 db_jxc 数据库中创建 20 个数据表，创建完成后的数据表如图 22.9 所示。

下面以 user_tab 数据表为例介绍数据表的创建过程。

（1）在新建的计划 db_jxc 节点上右击鼠标，在弹出的快捷菜单中选择 Create New Table 命令，如图 22.10

所示。

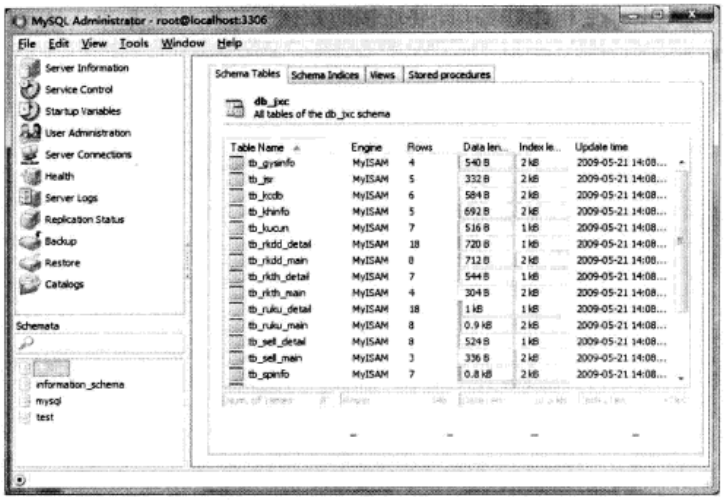


图 22.9　MySQL Administrator 窗口

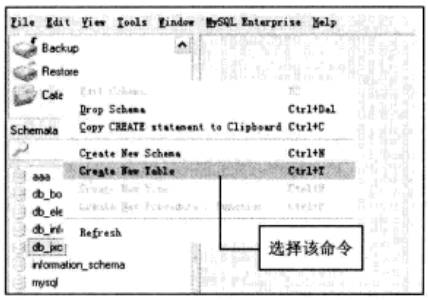


图 22.10　选择 Create New Table 命令

（2）在打开的 MySQL Table Editor 窗口中，输入要创建的表名、字段名称、字段的数据类型，然后设置是否创建主键、是否允许空值、选择编码，如图 22.11 所示。

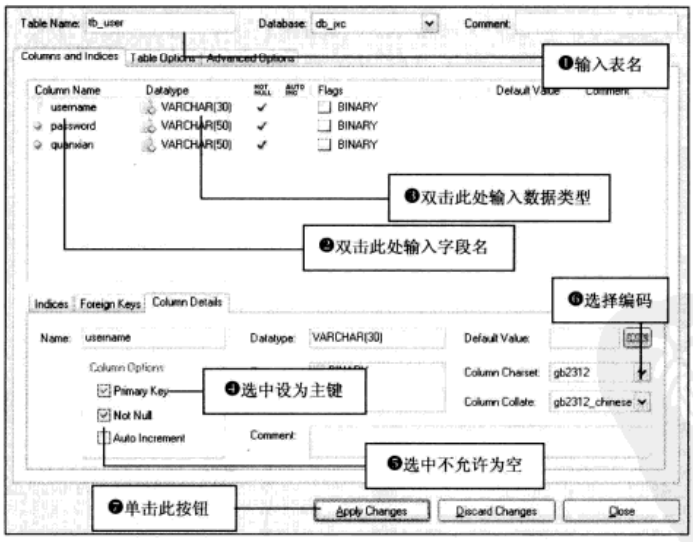


图 22.11　MySQL Table Editor 窗口

（3）在图 22.11 所示的 MySQL Table Editor 窗口中设置完表的相关内容后，单击 Apply Changes 按钮，将弹出 Confirm Table Edit 对话框，如图 22.12 所示。单击 Execute 按钮，完成表 user_tab 的创建。

由于篇幅有限，其他数据表的创建过程这里就不再介绍，相信读者能够举一反三，结合下面给出的数据表结构，创建其他的数据表也不成问题。

☑　供应商信息表

供应商信息表的名称为 tb_gysinfo，它主要用于存储供应商详细信息，其结构如表 22.5 所示。

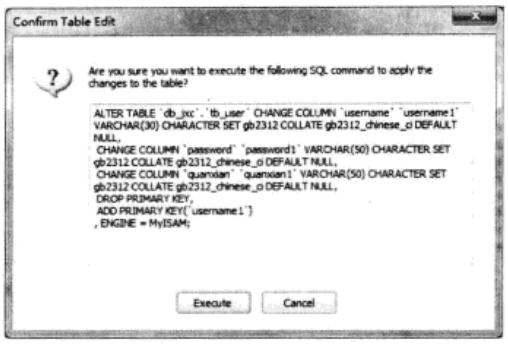


图 22.12　Confirm Table Edit 对话框

表 22.5　tb_gysinfo 供应商信息表结构

字 段 名 称	数 据 类 型	字 段 大 小	是 否 主 键	说　明
id	varchar	50	主键	供应商编号
gysqc	varchar	90		供应商名称
jc	varchar	50		供应商简称
address	varchar	100		供应商地址
yb	varchar	50		邮政编码
dh	varchar	50		电话
cz	varchar	50		传真
lxr	varchar	50		联系人
lxrdh	varchar	50		联系人电话
khyh	varchar	50		开户银行
yhzh	varchar	50		银行账号
dzxx	varchar	50		电子信箱

☑　商品信息表

商品信息表的名称为 tb_spinfo，它主要用于存储商品详细信息，其结构如表 22.6 所示。

表 22.6　tb_spinfo 商品信息表结构

字 段 名 称	数 据 类 型	字 段 大 小	是 否 主 键	说　明
id	varchar	50	主键	商品编号
spmc	varchar	50		商品名称
jc	varchar	50		商品简称
cd	varchar	50		产地
dw	varchar	50		商品计量单位
gg	varchar	50		商品规格
bz	varchar	50		包装
ph	varchar	50		批号
pzwh	varchar	50		批准文号
gysid	varchar	50		供应商编号
gysqc	varchar	90		供应商全称
memo	varchar	100		备注

☑ 入库主表

入库主表的名称为 tb_ruku_main，它主要用于存储入库单据信息，其结构如表 22.7 所示。

表 22.7 tb_ruku_main 入库主表结构

字段名称	数据类型	字段大小	是否主键	说明
rkid	varchar	30	主键	入库票号
pzs	integer			品种数量
je	decimal(19,4)			总计金额
ysjl	varchar	50		验收结论
gysqc	varchar	90		供应商全称
gyslxr	varchar	50		供应商联系人电话
rkrq	datetime			入库日期
dgrq	datetime			订购日期
dgczy	varchar	30		订购时的操作员
dgjsr	varchar	30		订购时的经手人
jsfs	varchar	10		结算方式
rkjsr	varchar	30		入库时的经手人
rkczy	varchar	30		入库时的操作员
rkddid	varchar	30		入库订单票号

☑ 入库明细表

入库明细表的名称为 tb_ruku_detail，它主要用于存储入库详细信息，其结构如表 22.8 所示。

表 22.8 tb_ruku_detail 入库明细表结构

字段名称	数据类型	字段大小	是否主键	说明
rkid	varchar	30		入库票号
spid	varchar	50		商品编号
dj	decimal(19,4)			单价
sl	integer			数量
rkddid	varchar	50		入库订单票号

☑ 销售主表

销售主表的名称为 tb_sell_main，它主要用于存储销售单据信息，其结构如表 22.9 所示。

表 22.9 tb_sell_main 销售主表结构

字段名称	数据类型	字段大小	是否主键	说明
xsid	varchar	30	主键	销售票号
pzs	integer	8		销售品种数
je	decimal(19,4)	8		总计金额
ysjl	varchar	50		验收结论
khqc	varchar	100		客户名称
xsrq	datetime			销售日期
qdrq	datetime			签单日期

续表

字 段 名 称	数 据 类 型	字 段 大 小	是 否 主 键	说　明
qdczy	varchar	30		销售签单时的操作员
qdjsr	varchar	30		销售签单时的经手人
jsfs	varchar	10		结算方式
xsczy	varchar	30		销售时的操作员
xsjsr	varchar	30		销售时的经手人
xsqdid	varchar	30		销售签单票号

☑ 销售明细表

销售明细表的名称为 tb_sell_detail，它主要用于存储销售详细信息，其结构如表 22.10 所示。

表 22.10　tb_sell_detail 销售明细表结构

字 段 名 称	数 据 类 型	字 段 大 小	是 否 主 键	说　明
xsid	varchar	30		销售票号
spid	varchar	50		商品编号
spmc	varchar	100		商品名称
dj	decimal(19,4)			销售单价
sl	integer			销售数量
xsqdid	varchar	30		销售签单票号

📖 **说明：** 由于篇幅有限，这里只列举了重要的数据表的结构，其他的数据表结构可参见光盘中的数据库文件。

22.6　创 建 项 目

简易通企业进销存管理系统是使用 Eclipse 3.6 开发的，在 Eclipse 3.6 开发环境中创建简易通企业进销存管理系统项目的具体步骤如下：

（1）启动 Eclipse 3.6，在 Eclipse 3.6 工作台中选择“文件”/“新建”/“Java 项目”命令，打开“新建 Java 项目”对话框。

（2）在“新建 Java 项目”对话框的“项目名”文本框中输入新建项目的名称，这里输入的是“简易通企业进销存管理系统”，其他选项都使用默认设置，单击“完成”按钮，完成项目的创建。

22.7　系统文件夹组织结构

为了方便对简易通企业进销存管理系统进行管理和维护，在编写项目代码之前，需要定制好项目的系统文件夹组织结构。例如，不同的 Java 包存放不同的窗体、公共类、工具类或者图片资源等，这样可以规范系统的整体架构。创建完系统中可能用到的文件夹或者 Java 包之后，在开发时，只需将所创建的类文件或资源文件保存到相应的文件夹中即可。简易通企业进销存管理系统的文件夹组织结构图如图 22.13 所示。

图 22.13　文件夹组织结构图

22.8　公共类设计

公共类是代码重用的一种形式，它将各个功能模块经常调用的方法提取到共用的 Java 类中，例如，访问数据库的 DaoConn 类容纳了建立数据库连接的方法，其他功能模块都可以使用该类建立与数据库连接和对数据库进行各种操作。这样不但实现了项目代码的重用，还提高了程序的性能和代码的可读性。本节将向读者介绍简易通企业进销存管理系统中的公共类设计。

22.8.1　SaveUserInfo 公共类

SaveUserInfo 公共类放在 zzk.zhuoyue.savestate 包中，用于保存用户的登录信息，userName 属性用于存储登录用户名，password 属性用于存储登录用户的密码，quanXian 属性用于存储登录用户的权限。SaveUserInfo 公共类的程序代码如下：

代码位置：简易通企业进销存管理系统\src\zzk\zhuoyue\savestate\SaveUserInfo.java

```
package zzk.zhuoyue.savestate;                                    // 声明包
public class SaveUserInfo {
    private static String userName="";                            // 用户名
    private static String password="";                            // 密码
    private static String quanXian="";                            // 用户权限
    public static String getPassword() {                          // 获得密码
        return password;
    }
    public static void setPassword(String password) {             // 设置密码
        SaveUserInfo.password = password;
    }
    public static String getQuanXian() {                          // 获得用户权限
        return quanXian;
    }
    public static void setQuanXian(String quanXian) {             // 设置用户权限
        SaveUserInfo.quanXian = quanXian;
    }
```

```
    public static String getUserName() {                    // 获得用户名
        return userName;
    }
    public static void setUserName(String userName) {       // 设置用户名
        SaveUserInfo.userName = userName;
    }
}
```

注意：代码中相应的 getXXX()方法和 setXXX()方法被声明为 public 公有的 static 静态的方法，这样在其他类中就可以通过类名直接访问这些方法，对登录信息进行操作，而相应的属性则被声明为 private 私有属性，这样就实现了属性的封装。

22.8.2　OnlyNumber 公共类

OnlyNumber 公共类放在 zzk.zhuoyue.onlynumber 包中，用于监听用户输入字符，当遇到非法字符时终止输入，这样可以在很大程序上减少输入的错误，方便用户操作。OnlyNumber 公共类的程序代码如下：

代码位置：简易通企业进销存管理系统\src\zzk\zhuoyue\onlynumber\OnlyNumber.java

```
package zzk.zhuoyue.onlynumber;
import java.awt.event.*;
public class OnlyNumber extends KeyAdapter {
    public void keyTyped(KeyEvent e) {                      // 重写 keyTyped()方法
        String key="-0123456789.*";                         // 声明字符串并赋初值
        if(key.indexOf(e.getKeyChar())<0){                  // 判断输入的字符是否在字符串 key 中
            e.consume();                                    // 销毁本次输入的字符
        }
    }
}
```

22.8.3　DaoConn 公共类

DaoConn 公共类放在 zzk.zhuoyue.dao 包中，用于指定数据库驱动，连接数据库名称、用户名称和密码以及获得数据连接对象；为其他功能模块操作数据库提供方便。DaoConn 公共类的程序代码如下：

代码位置：简易通企业进销存管理系统\src\zzk\zhuoyue\dao\DaoConn.java

```
package zzk.zhuoyue.dao;
import java.sql.*;
public class DaoConn {
public Connection getConn() throws Exception{
Class.forName("com.mysql.jdbc.Driver");                     // 定义数据库驱动类的名称
Connection conn=null;                                       // 声明数据库的连接对象
String url="jdbc:mysql://localhost:3306/db_jxc";            // 定义访问数据库的 URL
String user="root";                                         // 定义访问数据库的用户名
String password="111";                                      // 定义访问数据库的密码
conn=DriverManager.getConnection(url, user, password);      // 实现数据库的连接
return conn;
    }
}
```

22.9 系统登录模块设计

本模块使用的数据表：tb_user

系统登录模块主要用于对进入简易通企业进销存管理系统的用户进行安全性检查，以防止非法用户进入该系统。只有输入正确的用户名 mr 和密码 mrsoft，单击“登录”按钮才能进入系统。系统登录模块的界面效果如图 22.14 所示。

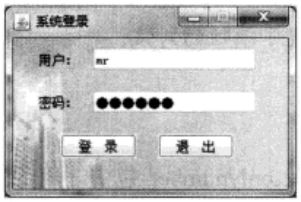


图 22.14 系统登录

22.9.1 设计系统登录窗体

在 zzk.zhuoyue.daohang 包中创建 LoginFrame 类，该类继承自 JFrame 类，使其成为一个窗体，设置窗体的标题为“系统登录”，在窗体内容面板的中央放置一个自定义的用于显示背景的内部类面板，设置面板的布局为 null，在面板上布置各种组件，实现系统登录窗体的设计。系统登录窗体用到的主要组件如表 22.11 所示。

表 22.11 系统登录窗体用到的主要组件

组 件 类 型	组件 ID	主要属性设置	说 明
JTextField	txf_username	无	登录用户名文本框
JPasswordField	pwd_password	无	登录密码文本框
JButton	btn_enter	Text 属性设置为“登录”	“登录”按钮
	btn_exit	Text 属性设置为“退出”	“退出”按钮

22.9.2 “登录”按钮的事件处理

“登录”按钮用于执行用户名和密码的验证工作，如果验证用户名和密码有效，则启动系统，否则禁止进入系统，并提示“用户名或密码不正确!!!”。将“登录”按钮添加到显示背景的内部类中，“登录”按钮的程序代码如下：

代码位置：简易通企业进销存管理系统\src\zzk\zhuoyue\daohang\LoginFrame.java

```
private void btn_enterActionPerformed(java.awt.event.ActionEvent evt) {
    Connection conn = null;                                    // 声明数据库的连接对象
    Statement st = null;                                       // 声明 Statement 对象
    ResultSet rs = null;;                                      // 声明结果集对象
try {
    conn = daoConn.getConn();                                  // 建立数据库的连接
    st = conn.createStatement();                               // 创建 Statement 对象
```

```
        String user = txf_username.getText().trim();                    // 获得用户名
        String pwd = new String(pwd_password.getPassword()).trim();     // 获得密码
        String sql = "select * from tb_user where username='" + user
                + "' and password='" + pwd + "'";                       // 查询语句
        rs = st.executeQuery(sql);                                      // 执行查询语句
        if (rs.next() && rs.getString("username") != null) {            // 用户名和密码正确
            String quanxian = rs.getString("quanxian");                 // 获得用户权限
            SaveUserInfo.setUserName(user);                             // 保存用户名到公共类
            SaveUserInfo.setPassword(pwd);                              // 保存密码到公共类
            SaveUserInfo.setQuanXian(quanxian);                         // 保存用户权限到公共类
            MainDaohangFrame.launch();                                  // 启动主窗体
            hideLoginFrame();                                           // 隐藏登录窗体
        } else {
            JOptionPane.showMessageDialog(this, "用户名或密码不正确！！！");
            txf_username.setText("");                                   // 清空文本框
            pwd_password.setText("");                                   // 清空密码框
            txf_username.requestFocus();                                // 使用文本框获得输入焦点
        }
    }catch (Exception ex) {
        JOptionPane.showMessageDialog(this, ex.getMessage(), "数据库异常",
                JOptionPane.INFORMATION_MESSAGE);                       // 显示消息框
    }finally {
        try {
            conn.close();                                               // 关闭数据库连接
        }
        catch (Exception e) {

        }
    }
}
```

22.9.3　“密码”文本框的回车事件

在系统登录窗体的“密码”文本框中添加了动作事件监听器，当在“密码”文本框中按 Enter 键时，将调用“登录”按钮的动作事件，也就是说在“密码”文本框中输入密码后，按 Enter 键将实现与“登录”按钮相同的功能。“密码”文本框的程序代码如下：

代码位置：简易通企业进销存管理系统\src\zzk\zhuoyue\daohang\LoginFrame.java

```
private void pwd_passwordActionPerformed(java.awt.event.ActionEvent evt) {
    btn_enterActionPerformed(evt);                          // 调用“登录”按钮的动作事件
}
```

注意：由于篇幅有限，这里只给出了主要组件的事件处理过程，其他组件的事件处理过程和方法没有给出，可以参看光盘中的源程序。

22.10　主窗体模块设计

主窗体是人机交互的界面，用户通过主窗体中的菜单、工具栏和按钮等组件进行管理操作。主窗体由

“菜单栏”、“工具栏”、窗体中心处的“导航按钮”以及“状态栏”组成，如图 22.15 所示。

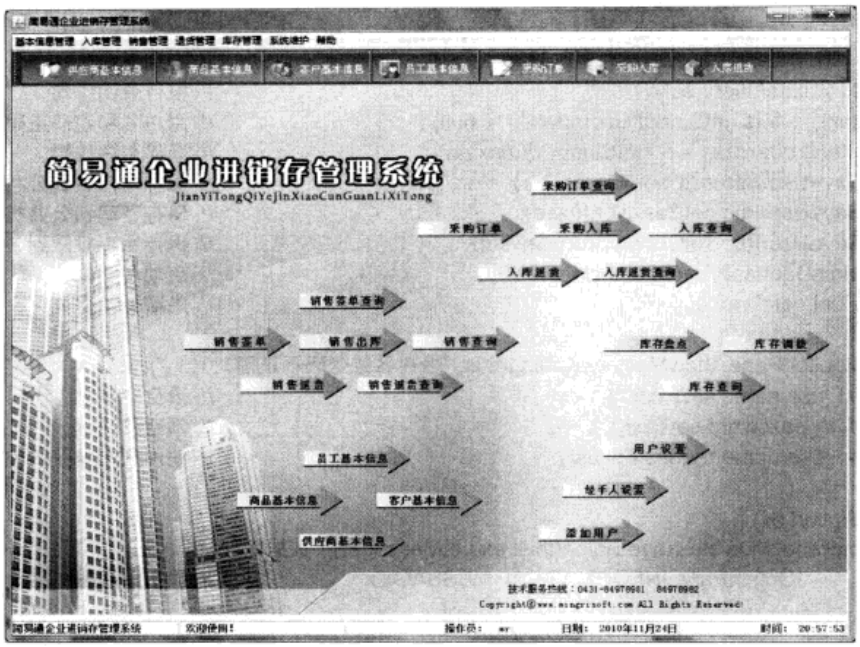


图 22.15　系统主窗体

22.10.1　创建系统主窗体

在 zzk.zhuoyue.daohang 包中创建 MainFrame 类，该类继承自 JFrame 类，使其成为一个窗体，设置窗体的标题为“简易通企业进销存管理系统”，实现系统主窗体的创建。

22.10.2　设计菜单栏

（1）在组件面板上单击 Menu Bar 组件，将其添加到窗体的上面，为窗体添加菜单栏 jMenuBar1。

（2）在组件面板上单击 Cascaded Menu 组件，将其添加到菜单栏 jMenuBar1 上，为菜单栏 jMenuBar1 添加菜单，把菜单标题设置为“基本信息管理”，名称设置为 mn_jbxxgl，完成菜单的添加。

（3）在组件面板上单击 Menu Item 组件，将其添加到“基本信息管理”菜单上，为“基本信息管理”菜单添加菜单项，把菜单项的标题设置为“供应商基本信息”，名称设置为 menu_gysxx，为菜单项添加图标，单击 Properties 选项卡中的 icon 属性。然后单击[...]按钮，在打开的 select image 对话框中为菜单项选择图标文件，单击“确定”按钮，完成图标的添加。至此，完成“供应商基本信息”菜单项的添加，如果需要还可以添加菜单项的分割线组件 Separator ，其添加方法与菜单项相同，这里不做介绍。

（4）要添加更多的菜单，可以重复步骤（2），要在菜单上添加更多的菜单项，可以重复步骤（3），这里就不做介绍了。

22.10.3　设计工具栏

（1）在组件面板上单击 JToolBar 组件，将其添加到窗体内容面板的上部，为窗体添加工具栏 jToolBar1。

（2）在组件面板上单击 JButton 组件，将其添加到工具栏 jToolBar1 上，为工具栏 jToolBar1 添加工具按钮，设置按钮标题为空，把按钮名称设置为 tool_gysxx，单击 Properties 选项卡中的 icon 属性。然后单击[...]按钮，在打开的 select image 对话框中为工具按钮选择图标文件，单击“确定”按钮，完成图标的添加。接下来为工具按钮添加鼠标指向按钮的图标，单击 Properties 选项卡中的 rollovericon 属性，然后单击[...]按钮，在打开的 select image 对话框中为工具按钮选择鼠标指向按钮的图标文件，单击“确定”按钮，完成转滚图标的添加，至此完成工具按钮的添加，用同样的方法继续添加其他工具栏按钮，即可完成工具栏的设计，这里不再一一介绍。

22.10.4　设计导航按钮

在组件面板上单击 JPanel 组件，将其放在窗体内容面板的中心位置，设置其名称为 jPanel1，并设置布局为 null，然后在 null 布局的面板容器 jPanel1 上放置若干按钮组件 JButton，并放到适合的位置，选择某个按钮，单击 Properties 选项卡中的 icon 属性。然后单击[...]按钮，在打开的 select image 对话框中为导航按钮选择图标文件，单击“确定”按钮，完成图标的添加。接下来为导航按钮添加鼠标指向按钮的图标，单击 Properties 选项卡中的 rollovericon 属性，然后单击[...]按钮，在打开的 select image 对话框中为导航按钮选择鼠标指向按钮的图标文件，单击“确定”按钮，完成转滚图标的添加，用同样的方法为其他导航按钮添加图标和转滚图标，完成导航按钮的设计。

22.10.5　设计状态栏

Swing 组件中没有“状态栏”，因此需要手动创建，“状态栏”的创建方法如下：

（1）在窗体内容面板的底部位置放置一个面板容器 JPanel，其名称为 jPanel2，将面板容器 jPanel2 设置为“网格式”布局，然后在其上添加 5 个标签组件 JLabel。

（2）在面板容器 jPanel2 中选择标签 jLabel1，在 Properties 页设置 border 属性为 Etched Border，单击 border 属性前方的加号“+”将其展开，设置 etchType 为 raised，完成该标签组件的设置。

（3）其他标签的设置可以重复第（2）步。设置完成的状态栏结果如图 22.16 所示。

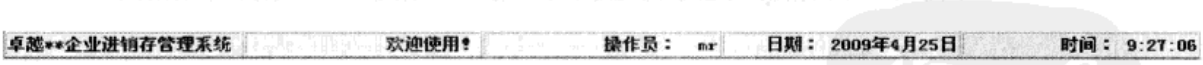


图 22.16　状态栏

22.10.6　设计显示背景图片的面板

为了使主窗体运行后能显示背景图片，创建一个显示背景图片的面板，创建过程如下：

（1）在 zzk.zhuoyue.daohang 包上单击鼠标右键，在弹出的快捷菜单中选择“新建”/“类”命令，打开“新建 Java 类”对话框。

（2）在“新建 Java 类”对话框的“名称”文本框中输入所要创建的面板名称，这里输入的是“PanelBack”，其他使用默认值，单击“完成”按钮，完成类的创建。

（3）修改类 PanelBack 的代码，使其继承自类 JPanel，成为面板。面板 PanelBack 用于显示图片，在主窗体中使用该面板的实例，使主窗体显示背景图片，该面板的程序代码如下：

代码位置：简易通企业进销存管理系统\src\zzk\zhuoyue\daohang\PanelBack.java

```
package zzk.zhuoyue.daohang;
import java.awt.*;
import java.net.URL;
import javax.swing.ImageIcon;
public class PanelBack extends javax.swing.JPanel {
Image image;                                                    // 定义图像对象
int w;
int h;
public PanelBack(int w,int h) {
URL url=PanelBack.class.getResource("/res/背景.jpg");           // 得到图像的 URL
image=new ImageIcon(url).getImage();                            // 创建图像对象
initComponents();
this.w=w;
this.h=h;
}
// 这里省略了 w 和 h 的 set 和 get 方法
protected void paintComponent(Graphics g) {                     // 重写 paintComponent()方法
super.paintComponent(g);
Graphics2D g2=(Graphics2D)g;                                    // 强制转换为 Graphics2D
g2.drawImage(image,0,0,w,h, this);                              // 绘制图像
}
private void initComponents() {
setLayout(new java.awt.BorderLayout());                         // 设置面板布局
}
}
```

初始化状态栏的方法，在构造方法中调用该方法，为状态栏上的标签设置显示的信息，其程序代码如下：

代码位置：简易通企业进销存管理系统\src\zzk\zhuoyue\daohang\MainFrame.java

```
private void init(){
jLabel1.setText("简易通企业进销存管理系统");
jLabel2.setText("欢迎使用！ ");
jLabel3.setText("操作员：   "+SaveUserInfo.getUserName());          // 获得登录用户名
DateFormat df=DateFormat.getDateInstance(DateFormat.LONG);          // 定义长日期格式对象
String dateString=df.format(new Date());                            // 获得格式化的日期字符串
jLabel4.setText("日期：  "+dateString);
df=DateFormat.getTimeInstance(DateFormat.MEDIUM);                   // 实例化时间格式对象
dateString=df.format(new Date());                                   // 获得格式化的时间字符串
jLabel5.setText("时间：  "+dateString);
}
```

该方法首先获得面板 jPanel1 上的组件数组，然后把自定义面板 pnl_back 放在面板 jPanel1 上，把 jPanel1 上的所有按钮都移除，并添加到自定义面板 pnl_back 上，按钮的位置和大小保持不变，其程序代码如下：

代码位置：简易通企业进销存管理系统\src\zzk\zhuoyue\daohang\MainFrame.java

```
private void ComponentPosition(){
Component[] component=jPanel1.getComponents();                  // 得到 jPanel1 上的组件数组
pnl_back.setLayout(null);                                       // 设置背景面板为空布局
jPanel1.add(pnl_back,new Integer(Integer.MAX_VALUE));           // 把背景面板添加到 jPanel1 上
sizeAndPosition();                                              // 调用方法，设置背景面板的大小和位置
for (int i=0;i<component.length;i++){
jPanel1.remove(component[i]);                                   // 移除 jPanel1 上的所有按钮
((JButton)component[i]).setOpaque(false);
```

```
((JButton)component[i]).setContentAreaFilled(false);
((JButton)component[i]).setBorder(null);
// 把 jPanel1 上移除的按钮添加到背景面板上
pnl_back.add(component[i]);
// 指定按钮在背景面板上的位置和大小
component[i].setBounds(component[i].getX(),component[i].getY(),
component[i].getWidth(),component[i].getHeight());
}
pnl_back.validate();                                    // 使背景面板上的组件有效
}
```

在显示主窗体时，使用该方法可以使主窗体从完全透明过渡到完全不透明状态，实现使主窗体渐渐显示的效果，其程序代码如下：

代码位置：简易通企业进销存管理系统\src\zzk\zhuoyue\daohang\MainFrame.java

```
public static void launch() {
    java.awt.EventQueue.invokeLater(new Runnable() {
        public void run() {
            final MainDaohangFrame mainFrame=new MainDaohangFrame();    // 实例化主窗体
            try {
                AWTUtilities.setWindowOpacity(mainFrame, 0.0f);         // 使窗体完全透明
                mainFrame.setVisible(true);
                SwingUtilities.invokeLater(new Runnable() {
                    public void run() {
                        for (float i = 0.01f; i < 1.0f; i = i + 0. 2f) {
                            AWTUtilities.setWindowOpacity(mainFrame, i);    // 调整窗体的透明度
                            try {
                                Thread.sleep(40);                           // 休眠 40 毫秒
                            } catch (InterruptedException e) {
                                e.printStackTrace();
                            }
                        }
                        AWTUtilities.setWindowOpacity(mainFrame, 1);        // 使窗体完全不透明
                    }
                });
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    });
}
```

注意：由于篇幅有限，这里只给出了背景面板类和部分方法的程序代码，其他程序代码可以参看光盘中的源程序。

22.11　供应商基本信息管理模块设计

本模块使用的数据表：tb_gysinfo

“供应商基本信息管理”模块负责添加、修改、删除和查询供应商信息，其中“添加供应商信息”选

项卡用于添加供应商的基本信息，“修改供应商信息”选项卡用于修改指定的供应商信息，“删除供应商信息”选项卡用于删除指定的供应商信息和查询供应商的信息。“供应商基本信息管理”窗体界面如图 22.17 所示。

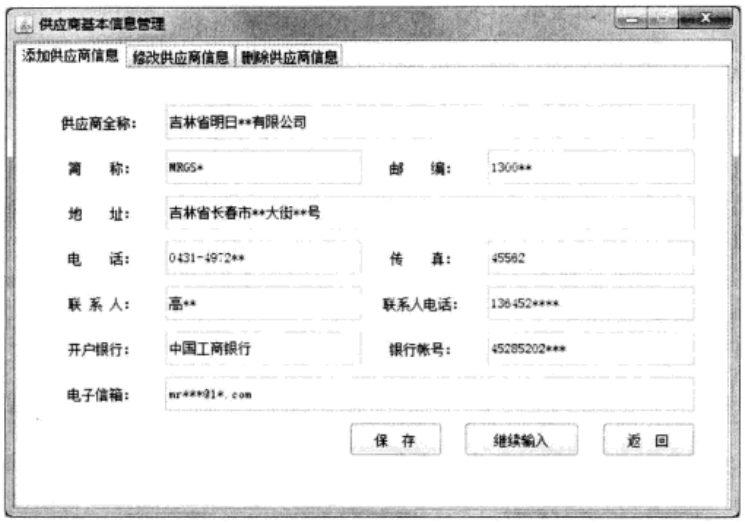


图 22.17 “供应商基本信息管理”窗体界面

22.11.1 设计供应商基本信息管理窗体

在项目“简易通企业进销存管理系统”中新建一个包 zzk.zhuoyue.jbxxframe，然后在该包中建立一个名称为 GysInfoFrame 的 JFrame 窗体，设置窗体布局为“边框式”布局，并在“边框式”布局的窗体中心放一个选项卡面板，在选项卡面板上放置 3 个 null 布局的面板对应选项卡面板上的 3 个选项，其中第一个面板上放置 11 个标签、11 个文本框和 3 个按钮，第二个面板上放置 12 个标签、11 个文本框、3 个按钮和一个组合框，第 3 个面板上放置一个标签、一个组合框、一个滚动面板、一个表和 3 个按钮。该窗体中所用到的部分关键组件如表 22.12 所示。

表 22.12 供应商基本信息管理窗体用到的部分关键组件

控 件 类 型	控件 ID	主要属性设置	用　途
JTextField	txf_gysqc	无	供应商全称
	txf_jiancheng	无	简称
	txf_dizhi	无	地址
	txf_youbian	无	邮编
	txf_dianhua	无	电话
	txf_chuanzhen	无	传真
	txf_lianxiren	无	联系人
	txf_lianxidianhua	无	联系人电话
	txf_kaihuhang	无	开户银行
	txf_yinhangzhanghao	无	银行账号
	txf_email	无	电子信箱

续表

控 件 类 型	控件 ID	主要属性设置	用　途
JComboBox	cmb_xiugaigysqc	设置 Editable 属性为 true	显示所有不重复的供应商全称
JButton	btn_baocun	设置 text 属性为“保存”	保存按钮
	btn_chongshu	设置 text 属性为“继续输入”	重新输入按钮
JTable	tb_shanchu	设置 autoResizeMode 属性为 Off	显示供应商信息的表格

22.11.2　添加供应商信息

在“供应商基本信息管理”窗体界面中选择“添加供应商信息”选项卡，可以直接输入供应商基本信息，输入完成后单击“保存”按钮，可以保存本次输入的供应商信息，同时“保存”按钮变为灰色不可用状态。单击“继续输入”按钮，“保存”按钮变为可用状态，并可以输入新的供应商信息。单击“返回”按钮将退出供应商基本信息管理窗体，返回应用程序主窗体。

“添加供应商信息”选项卡中的“保存”按钮用于保存所添加的供应商信息，当用户输入完供应商的相关信息后，单击“保存”按钮，可以保存该供应商的信息。“保存”按钮的事件处理过程代码如下：

代码位置：简易通企业进销存管理系统\src\zzk\zhuoyue\jbxxframe\GysInfoFrame.java

```
private void btn_baocunActionPerformed(java.awt.event.ActionEvent evt) {
    try {
        // 获得供应商信息
        String gysid=null;
        String gysqc=txf_gysqc.getText().trim();
        …这里省略了其他获得信息的代码
        String email=txf_email.getText().trim();
        // 判断几个重要的供应商信息是否输入，如果没输入则提示
        if (gysqc.equals("") || jiancheng.equals("") || dizhi.equals("") ||
                youbian.equals("") || lianxiren.equals("") || lianxidianhua.equals("")){
            JOptionPane.showMessageDialog(this,"信息不全，请输入。");
            return;
        }
        conn=daoConn.getConn();                                    // 实例化数据库连接对象
        st=conn.createStatement();                                 // 实例化 Statement 对象
        String sql="select Max(id) as id from tb_gysinfo ";        // 查询最大的供应商 id
        rs=st.executeQuery(sql);                                   // 执行 SQL 语句
        // 自动生成供应商 id
        if (rs.next() && rs.getString("id")!=null) {               // 存在最大的供应商 id
            String id=rs.getString("id");                          // 获得最大的供应商 id
            String subgysid=id.substring(3);                       // 截到最大的供应商 id 的后 4 位
            // 对后 4 位加 1 生成新的供应商 id
            gysid="gys"+String.valueOf(Integer.parseInt(subgysid)+1);
    }else {                                                        // 不存在最大的供应商 id
         gysid="gys1001";                                          // 供应商 id 后 4 位为 1001
}
// 定义保存供应商信息的 SQL 语句
    String addsql="insert into tb_gysinfo values('"+gysid+"','"+
    gysqc+"','"+jiancheng+"','"+dizhi+"','"+youbian+"','"+
    dianhua+"','"+chuanzhen+"','"+lianxiren+"','"+lianxidianhua+"','"+
     kaihuhang+"','"+yinhangzhanghao+"','"+email+"')";
```

```
        st.executeUpdate(addsql);                                    // 执行 SQL 语句
        JOptionPane.showMessageDialog(this,"保存成功！！！");// 提示保存成功
        btn_baocun.setEnabled(false);                                // 使按钮变为灰色不可用状态
    }catch(Exception ex) {
        ex.printStackTrace();
    }
    }
```

注意：由于篇幅有限，这里只给出了“保存”按钮的事件处理过程，其他组件的事件处理过程和方法没有给出，可以参看光盘中的源程序。

22.11.3 修改供应商信息

在“供应商基本信息管理”窗体界面中选择“修改供应商信息”选项卡，可以对供应商信息进行修改，用户可以从组合框中选择或者手动输入供应商全称，然后单击“查询”按钮即可显示该供应商信息，这时可以对该供应商信息进行修改。在相应的文本框中输入新的修改信息，单击“修改”按钮完成修改操作。“修改供应商信息”选项卡界面如图 22.18 所示。

图 22.18 “修改供应商信息”选项卡界面

“修改供应商信息”选项卡中的“修改”按钮用于修改供应商信息，当用户从组合框中选择或输入供应商全称时，单击“查询”按钮，将在文本框中显示该供应商的相关信息，对不满意的内容可以在文本框中进行修改，然后单击“修改”按钮，完成修改操作。“修改”按钮的事件处理过程代码如下：

代码位置：简易通企业进销存管理系统\src\zzk\zhuoyue\jbxxframe\GysInfoFrame.java

```
private void btn_xiugaiActionPerformed(java.awt.event.ActionEvent evt) {
    if (JOptionPane.showConfirmDialog(this,
"确实要修改该供应商信息吗？\t\n 单击“确定”按钮将修改该信息。","确认对话框",
JOptionPane.OK_CANCEL_OPTION)==JOptionPane.OK_OPTION) {
        updateGysInfo(gongyingshangid);                              // 调用修改供应商信息的方法
        addGysqcToComboBox(cmb_xiugaigysqc);                         // 在组合框中添加供应商全称
        addGysqcToComboBox(cmb_shanchugysqc);
        if (cmb_xiugaigysqc.getItemCount()>0){
            String gysqc=cmb_xiugaigysqc.getSelectedItem().toString();
            queryGysInfoToTextFields(gysqc);                         // 调用方法在文本框中显示供应信息
            // 调用在“删除供应商信息”选项卡中显示全部供应商信息的方法
```

```
                showGysInfoToTable("全部");
            }
        }
    }
```

使用该方法可以根据供应商 id 修改供应商信息，当要修改供应商信息时，只需要调用该方法，并把要修改的供应商 id 作为实参传递给该方法，即可完成供应商信息的修改，其程序代码如下：

代码位置：简易通企业进销存管理系统\src\zzk\zhuoyue\jbxxframe\GysInfoFrame.java

```
public void updateGysInfo(String gysid){
    try{
        // 获得控件的值
        String gysqc=txf_xiugaigysqc.getText().trim();
        String jiancheng=txf_xiugaijiancheng.getText().trim();
        String dizhi=txf_xiugaidizhi.getText().trim();
        String youbian=txf_xiugaiyoubian.getText().trim();
        String dianhua=txf_xiugaidianhua.getText().trim();
        String chuanzhen=txf_xiugaichuanzhen.getText().trim();
        String lianxiren=txf_xiugailianxiren.getText().trim();
        String lianxidianhua=txf_xiugailianxidianhua.getText().trim();
        String kaihuhang=txf_xiugaikaihuhang.getText().trim();
        String yinhangzhanghao=txf_xiugaiyinhangzhanghao.getText().trim();
        String email=txf_xiugaiemail.getText().trim();
        conn=daoConn.getConn();
        // 定义带参数的 SQL 语句，每个问号是一个参数
        String sql="update tb_gysinfo set gysqc=?,jc=?,dz=?,yb=?,dh=?,cz=?,lxr=?,
        lxrdh=?,khyh=?,yhzh=?,dzxx=? where id=?";
        ps=conn.prepareStatement(sql);                          // 实例化 PreparedStatement 对象
        // 为 12 个参数顺序地传递实参值
        ps.setString(1,gysqc);
        ps.setString(2,jiancheng);
        ps.setString(3,dizhi);
        ps.setString(4,youbian);
        ps.setString(5,dianhua);
        ps.setString(6,chuanzhen);
        ps.setString(7,lianxiren);
        ps.setString(8,lianxidianhua);
        ps.setString(9,kaihuhang);
        ps.setString(10,yinhangzhanghao);
        ps.setString(11,email);
        ps.setString(12,gysid);
        ps.executeUpdate();                                     // 执行 SQL 语句
        JOptionPane.showMessageDialog(this,"修改成功！！！");
    }catch(Exception ex){
        JOptionPane.showMessageDialog(this,"数据库异常！！！\t\n"+ex.getMessage());
    }
}
```

使用该方法可以为指定的组合框添加供应商全称，当组合框中需要列出供应商全称时，可以调用该方法，并把组合框的名称作为实参传递给该方法，完成向组合框中添加供应商全称的操作，其程序代码如下：

代码位置：简易通企业进销存管理系统\src\zzk\zhuoyue\jbxxframe\GysInfoFrame.java

```
public void addGysqcToComboBox(JComboBox comboBox){
    try
```

```
    {
        comboBox.removeAllItems();                                    // 移除组合框中的所有列表项
        conn=daoConn.getConn();
        st=conn.createStatement();
        String sql="select distinct gysqc from tb_gysinfo";
        rs=st.executeQuery(sql);
        while (rs.next() && rs.getString(1)!=null)
            comboBox.addItem(rs.getString(1));                        // 为组合框添加列表项
    }catch(Exception ex){
        ex.printStackTrace();
    }
}
```

根据供应商全称在文本框中显示相应供应商信息的方法，为了使其他文本框中显示的信息能与所选择的供应商对应，可以在选择供应商全称后，把供应商全称作为实参传递给该方法，其程序代码如下：

代码位置：简易通企业进销存管理系统\src\zzk\zhuoyue\jbxxframe\GysInfoFrame.java

```
public void queryGysInfoToTextFields(String gysqc){
    try
    {
        conn=daoConn.getConn();
        st=conn.createStatement();
        String sql="select * from tb_gysinfo where gysqc='"+gysqc+"'";
        rs=st.executeQuery(sql);
        if (rs.next() && rs.getString("id")!=null){
            gongyingshangid=rs.getString("id");                       // 为全局变量赋值
            txf_xiugaigysqc.setText(rs.getString("gysqc"));           // 为文本框指定文本
            txf_xiugaijiancheng.setText(rs.getString("jc"));
            txf_xiugaidizhi.setText(rs.getString("dz"));
            txf_xiugaiyoubian.setText(rs.getString("yb"));
            txf_xiugaidianhua.setText(rs.getString("dh"));
            txf_xiugaichuanzhen.setText(rs.getString("cz"));
            txf_xiugailianxiren.setText(rs.getString("lxr"));
            txf_xiugailianxidianhua.setText(rs.getString("lxrdh"));
            txf_xiugaikaihuhang.setText(rs.getString("khyh"));
            txf_xiugaiyinhangzhanghao.setText(rs.getString("yhzh"));
            txf_xiugaiemail.setText(rs.getString("dzxx"));
        }
    }catch(Exception ex){
        ex.printStackTrace();
    }
}
```

根据供应商全称在表中显示供应商信息的方法，为了使表中显示的信息能与所选择的供应商对应，可以在选择供应商全称后，把供应商全称作为实参传递给该方法，其程序代码如下：

代码位置：简易通企业进销存管理系统\src\zzk\zhuoyue\jbxxframe\GysInfoFrame.java

```
public void showGysInfoToTable(String gysqc){
    try{
        String sql=null;
        gongyingshangquancheng=gysqc;                                 // 为全局变量赋值
        if (gysqc.equals("全部")){
            sql="select * from gysinfo_tab";
        }else{
```

```
            sql="select * from tb_gysinfo where gysqc='"+gysqc+"'";
        }
        conn=daoConn.getConn();
        // 定义可以滚动的 Statement 对象
        st=conn.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
        ResultSet.CONCUR_READ_ONLY);
        rs=st.executeQuery(sql);
        // 定义表头数组
        String[] columnNames={"ID","供应商全称","简称","地址","邮编",
        "电话","传真","联系人","联系人电话","开户银行","银行账号","电子信箱"};
        int currentRow=0;
        if (rs.last() && rs.getString("gysqc")!=null)                    // 将光标定位到结果集的最后一行
            totalRows=rs.getRow();                                       // 获得当前行的行号，即总记录数
        String[][] tableValues=new String[totalRows][12];                // 定义在表中显示内容的数组
        rs.beforeFirst();                                                // 将光标定位到结果集的第一行之前
        while (rs.next() && rs.getString("gysqc")!=null){
            for (int i=0;i<12;i++){
                int column=i+1;
                tableValues[currentRow][i]=rs.getString(column);
            }
            currentRow++;
        }
        // 实例化表格模型
        tableModel=new DefaultTableModel(tableValues,columnNames);
        tb_shanchu.setModel(tableModel);                                 // 为表指定表格模型
    }catch(Exception e){
        e.printStackTrace();
    }
}
```

22.11.4　删除供应商信息

在“供应商基本信息管理”窗体界面中选择“删除供应商信息”选项卡，可以对供应商信息进行查询和删除，用户可以从组合框中选择或者手动输入供应商全称，然后单击“查询”按钮即可在表中显示该供应商的信息，单击“全部显示”按钮，可以在表中显示所有供应商的信息。如果要删除供应商信息，可以在表中选择要删除的供应商对应的行，单击“删除”按钮即可删除该供应商信息。“删除供应商信息”选项卡界面如图 22.19 所示。

图 22.19　“删除供应商信息”选项卡界面

“删除供应商信息”选项卡用于查询和删除供应商信息，在组合框中选择或输入供应商全称，单击“查询”按钮查询指定条件的供应商信息，单击“全部显示”按钮，则显示所有供应商信息；从表中选择某供应商所在的行，单击“删除”按钮，删除该供应商信息。“删除”按钮的事件处理过程代码如下：

代码位置：简易通企业进销存管理系统\src\zzk\zhuoyue\jbxxframe\GysInfoFrame.java

```
private void btn_shanchuActionPerformed(java.awt.event.ActionEvent evt) {
    Object object=null;
    int selectedRow=tb_shanchu.getSelectedRow();                    // 得到表格中选择行的行号
    if (totalRows>1 && selectedRow==-1) {                           // 表中总行数大于 1，且没有选择任何行
        JOptionPane.showMessageDialog(this,"请选择要删除的行！！！");
        return;
    }else if (totalRows==1){                                        // 表中总行数等于 1
        object=tableModel.getValueAt(0,0);
    }else if (totalRows<1 || tb_shanchu.getRowCount()==0){          // 表中没有数据
        JOptionPane.showMessageDialog(this,"没有可以删除的数据！！！");
        return;
    }else{
        object=tableModel.getValueAt(selectedRow,0);
    }
    if (object!=null){
        String gysid=object.toString();
        if (JOptionPane.showConfirmDialog(this,"确实要删除该供应商信息吗？\t\n"+
                "单击“确定”按钮将永久删除，不能恢复。\t\n","确认对话框",
                JOptionPane.OK_CANCEL_OPTION)==JOptionPane.OK_OPTION){
            deleteGysInfo(gysid);                                   // 调用删除供应商信息的方法
            addGysqcToComboBox(cmb_xiugaigysqc);                    // 在“修改”按钮中已提供该方法
            addGysqcToComboBox(cmb_shanchugysqc);
            if (cmb_xiugaigysqc.getItemCount()>0){
                String gysqc=cmb_xiugaigysqc.getSelectedItem().toString();
                queryGysInfoToTextFields(gysqc);                    // 在“修改”按钮中已提供该方法
                showGysInfoToTable(gongyingshangquancheng);         // 在“修改”按钮中已提供该方法
            }
        }
    }else{
        JOptionPane.showMessageDialog(this,"请选择有数据的行！！！");
    }
}
```

当要删除供应商信息时，只需要使用删除供应商的方法，并把要删除的供应商 id 作为实参传递给该方法即可，其程序代码如下：

代码位置：简易通企业进销存管理系统\src\zzk\zhuoyue\jbxxframe\GysInfoFrame.java

```
private void deleteGysInfo(String gysid){
    try{
        conn=daoConn.getConn();
        String sql="delete from tb_gysinfo where id=?";
        ps=conn.prepareStatement(sql);
        ps.setString(1,gysid);
        ps.executeUpdate();
```

```
            JOptionPane.showMessageDialog(this,"删除成功！！！");
            addGysqcToComboBox(cmb_xiugaigysqc);                    // 在“修改”按钮中已提供该方法
            addGysqcToComboBox(cmb_shanchugysqc);
            Object obj=cmb_xiugaigysqc.getSelectedItem();
            String gysqc="";
            if (obj!=null){
                gysqc=cmb_xiugaigysqc.getSelectedItem().toString();
            }
            queryGysInfoToTextFields(gysqc);                        // 在“修改”按钮中已提供该方法
            showGysInfoToTable(gongyingshangquancheng);             // 在“修改”按钮中已提供该方法
        }catch(Exception ex){
            JOptionPane.showMessageDialog(this,"数据库异常！！！\t\n"+ex.getMessage());
        }
    }
```

22.12　订购商品信息管理模块设计

本模块使用的数据表：tb_rkdd_main、tb_rkdd_detail、tb_spinfo、tb_gysinfo 和 tb_jsr

“订购商品信息”管理模块负责商品入库前的订购操作，只有订购了的商品才能入库，否则不能入库。单击“添加”按钮，可以初始化组件，添加新记录；单击“删除”按钮，可以删除表格中选择的记录；单击“订购”按钮，可以保存本次订购信息。“订购商品信息”管理窗体如图 22.20 所示。

图 22.20　“订购商品信息”管理窗体

22.12.1　设计订购商品信息管理窗体

在项目“简易通企业进销存管理系统”中新建一个包 zzk.zhuoyue.dinggouruku，然后在该包中建立一个名称为 CaigouDingdanFrame 的 JFrame 窗体，设置窗体布局为“边框式”布局，并在“边框式”布局的窗体中心放一个 null 布局的面板，在面板上放置 11 个标签、8 个文本框、4 个按钮、4 个组合框、一个滚动面板和一个表。该窗体中所用到的关键组件如表 22.13 所示。

表 22.13　订购商品信息窗体用到的关键组件

控 件 类 型	控件 ID	主要属性设置	用　　途
JTextField	txf_rkddid	设置 editable 属性为 false	入库订单 ID
	txf_gyslxr	无	供应商联系人
	txf_dgrq	设置 editable 属性为 false	订购日期
	txf_czy	设置 editable 属性为 false	操作员
	txf_pzs	设置 editable 属性为 false	品种数量
	txf_hpzs	设置 editable 属性为 false	货品总数
	txf_hjje	设置 editable 属性为 false	合计金额
	txf_ysjl	无	验收结论
JComboBox	cmb_gysqc	无	显示所有不重复的供应商全称
	cmb_jkfs	设置 model 属性为“现金，支票”	显示结款方式
	cmb_jsr	无	显示所有已启用的经手人
	cmb_spmc	无	显示所有不重复的商品名称
JButton	btn_tianjia	设置 text 属性为“添加”	添加新记录
	btn_dinggou	设置 text 属性为“订购”	保存订购信息
	btn_shanchu	设置 text 属性为“删除”	删除记录
	btn_fanhui	设置 text 属性为“返回”	返回主界面
JTable	tb_shanchu	设置 autoResizeMode 属性为 Off	显示本次订购商品信息的表格

22.12.2　订购商品信息

在“订购商品信息”窗体界面中单击“添加”按钮，可以初始化组件，同时在表中插入一空行，此时用户需要先选择供应商，然后在表格中新插入行的商品名称单元格内单击鼠标，将显示一个组合框，这时用户可以从组合框中选择商品名称，选择完商品名称后就会在新插入的行中显示该商品的相关信息，用户只需在表格中输入单价和数量，按 Enter 键即可自动计算相关数据，并显示在对应的文本框中。如果本订单还有其他信息需要添加，可以单击“添加”按钮重复上面的过程继续输入。当表中某行信息输入有误时，可以选择该行，单击“删除”按钮即可删除。全部信息输入完成后，选择经手人并输入验收结论，单击“订购”按钮，可以保存订单信息。单击“返回”按钮，退出“订购商品信息”窗体，返回应用程序主窗体。“订购商品信息”窗体中“订购”按钮的事件处理过程代码如下：

代码位置：简易通企业进销存管理系统\src\zzk\zhuoyue\dinggouruku\CaigouDingdanFrame.java

```
private void btn_dinggouActionPerformed(java.awt.event.ActionEvent evt) {
    String jsr=cmb_jsr.getSelectedItem().toString();                    // 获得经手人
    String ysjl=txf_ysjl.getText();                                     // 获得验收结论
    if (showDataToCompoment()){
        if (jsr.equals("请选择经手人")){
            JOptionPane.showMessageDialog(this,"请选择经手人！！！"); // 显示消息对话框
        }else if (ysjl==null || ysjl.equals("")){
            JOptionPane.showMessageDialog(this,"请输入验收结论！！！");
            txf_ysjl.requestFocus();
        }else{
            if (txf_rkddid.getText()==null || txf_rkddid.getText().trim().equals("") ||
```

```
        txf_dgrq.getText()==null || txf_dgrq.getText().trim().equals("")){
    JOptionPane.showMessageDialog(this,
      "请单击“添加”按钮，输入相关信息后，再订购。");
}else{
    try{
        String sql="insert into tb_rkdd_detail values(?,?,?,?)";
        conn=daoConn.getConn();
        conn.setAutoCommit(false);
        ps=conn.prepareStatement(sql);
        String rkddid=txf_rkddid.getText();
        int totalRows=tb_spinfo.getRowCount();
        int hpsl=0;                                          // 存放每种货品数量的变量
        int dj=0;                                            // 存放单价的变量
        String spbh=null;
        for (int i=0;i<totalRows;i++){
            spbh=(String)tb_spinfo.getValueAt(i,1);
            try{
                dj=Integer.valueOf(tb_spinfo.getValueAt(i,6).toString());
                hpsl=Integer.valueOf(tb_spinfo.getValueAt(i,7).toString());
            }catch(Exception ex){
                JOptionPane.showMessageDialog(this,
                  "商品单价或数量无效，请输入数字，并回车确认。");
                return;
            }
            if (spbh!=null && hpsl>0 && dj>0){
                // 为 SQL 语句的参数赋值
                ps.setString(1,rkddid);
                ps.setString(2,spbh);
                ps.setFloat(3,dj);
                ps.setInt(4,hpsl);
                ps.executeUpdate();
                ps.addBatch();                               // 批量添加参数
            }else{
                JOptionPane.showMessageDialog(this,
                  "商品数量输入有误，请重新输入，并回车确认。");
                return;
            }
            spbh=null;
            hpsl=0;
            dj=0;
        }
        conn.commit();
        conn.setAutoCommit(true);
        sql="insert into tb_rkdd_main values(?,?,?,?,?,?,?,?,?,?)";
        ps=conn.prepareStatement(sql);
        ps.setString(1,txf_rkddid.getText());                // 给SQL语句的第一个参数赋值
        ps.setInt(2,Integer.valueOf(txf_pzs.getText()));     // 给SQL语句的第二个参数赋值
        ps.setInt(3,Integer.valueOf(txf_hjje.getText()));
        ps.setString(4,txf_ysjl.getText());
        ps.setString(5,cmb_gysqc.getSelectedItem().toString());
        ps.setString(6,txf_gyslxr.getText());
```

```
                    ps.setString(7,txf_dgrq.getText());
                    ps.setString(8,txf_czy.getText());
                    ps.setString(9,cmb_jsr.getSelectedItem().toString());
                    ps.setString(10,cmb_jkfs.getSelectedItem().toString());
                    ps.executeUpdate();
                    JOptionPane.showMessageDialog(this,"订单添加成功。");
                    showSpInfoToTable();                    // 调用该方法清除表中内容，只显示标题
                    tbRows=0;
                    cmb_spmc.removeAllItems();              // 移除组合框中的所有列表项
                    cmb_gysqc.setSelectedItem("请选择供应商全称");
                    txf_gyslxr.setText(null);               // 清空文本框
                    cmb_jsr.setSelectedItem("请选择经手人");  // 设置组合框的默认选项
                    txf_pzs.setText(null);
                    txf_hpzs.setText(null);
                    txf_hjje.setText(null);
                    txf_ysjl.setText(null);
                    txf_dgrq.setText(null);
                    txf_rkddid.setText(null);
                    btn_dinggou.setEnabled(false);
                    enterCounts=1;
                }catch(Exception ex){
                    ex.printStackTrace();
                }
            }
        }
    }
}
```

当用户需要清除表中内容时，可以调用清除表中内容的方法把表中的内容清除，使其只显示表的标题，其程序代码如下：

代码位置：简易通企业进销存管理系统\src\zzk\zhuoyue\dinggouruku\CaigouDingdanFrame.java

```
public void showSpInfoToTable() {
    String[] columnNames = {"商品名称", "商品编号", "产地", "单位", "规格",
"包装","单价","数量", "批号", "批准文号"};
    String[][] tableValues = new String[0][10];             // 建立数组
    tableModel = new DefaultTableModel(tableValues, columnNames);
    tb_spinfo.setModel(tableModel);
    TableColumn column=tb_spinfo.getColumn("商品名称");      // 获得商品名称列的 TableColumn
    column.setPreferredWidth(120) ;                         // 设置表格中商品名称列的列宽
}
```

注意：由于篇幅有限，这里只给出了“订购”按钮的事件处理过程和该事件处理过程所用到的方法 showSpInfoToTable()，其他方法和组件的事件处理过程没有给出，可以参看光盘中的源程序。

22.13 采购入库管理模块设计

本模块使用的数据表和视图：tb_rkdd_main、tb_rkdd_detail、tb_ruku_main、tb_ruku_detail、tb_spinfo、tb_kucun、tb_jsr 和 v_rkdd

“采购入库管理”模块负责商品订购信息的入库操作，以采购订单票号为依据进行入库，没有进行采

购签单的商品不能入库。“采购入库管理”窗体界面如图 22.21 所示。

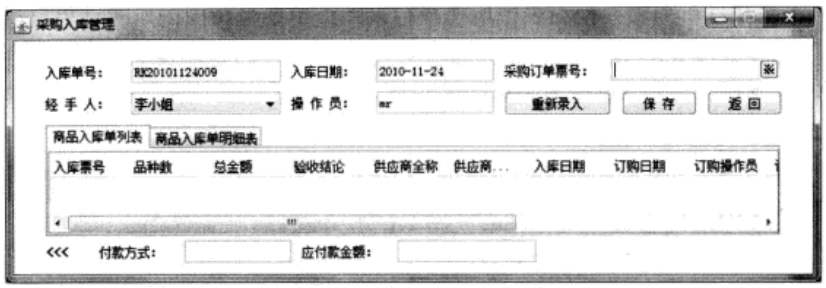


图 22.21　“采购入库管理”窗体界面

22.13.1　设计采购入库管理窗体

在项目“简易通企业进销存管理系统”的包 zzk.zhuoyue.dinggouruku 中建立一个名称为 CaigouRukuFrame 的 JFrame 窗体，设置窗体布局为“边框式”布局，并在“边框式”布局的窗体中心放一个 null 布局的面板，在面板上放置 8 个标签、7 个文本框、4 个按钮、一个组合框、4 个滚动面板、4 个表和一个选项卡面板。该窗体中所用到的关键组件如表 22.14 所示。

表 22.14　采购入库管理窗体用到的关键组件

控 件 类 型	控件 ID	主要属性设置	用　途
JTextField	txf_rkid	设置 editable 属性为 false	入库单号
	txf_rkrq	设置 editable 属性为 false	入库日期
	txf_rkddid	无	采购订单票号
	txf_rkczy	设置 editable 属性为 false	操作员
	txf_fkfs	设置 editable 属性为 false	付款方式
	txf_fkje	设置 editable 属性为 false	应付款金额
	txf_tbrkddid	无	随表格移动的文本框
JButton	btn_quanbudingdan	设置 text 属性为“※”	在表中显示所有订单商品信息
	btn_chongxinluru	设置 text 属性为“重新录入”	重新输入数据
	btn_baocun	设置 text 属性为“保存”	保存入库信息
	btn_fanhui	设置 text 属性为“返回”	返回主界面
JComboBox	cmb_jsr	无	显示所有启用的经手人
JLabel	lbl_zkyss	设置 text 属性为“<<<”	展开或收缩窗体
JTable	tb_main	设置 autoResizeMode 属性为 Off	显示商品入库单列表
	tb_showrukuinfo	设置 autoResizeMode 属性为 Off	显示商品入库信息
	tb_showdingdan	设置 autoResizeMode 属性为 Off	显示商品订单列表
	tb_detail	设置 autoResizeMode 属性为 Off	显示商品入库单明细表

22.13.2　采购入库管理

在“采购入库管理”窗体界面中应先选择经手人，然后单击“采购订单票号”右侧的※按钮，将显示

没有入库的订单列表，已经入库的将不再显示，这样可以避免重复输入同一个订单。单击选择要入库的订单信息所在的行，将隐藏订单列表并把与表中所选择行采购订单票号相同的所有订单信息添加到对应的表格中。单击“<<<”标签展开窗体，查看本次输入的订单详细信息，同时标签上显示的内容变为“>>>”，如图 22.22 所示。单击“>>>”标签，又恢复为原来的状态。单击“保存”按钮，可以保存本次入库信息。单击“重新录入”按钮，重新输入数据。

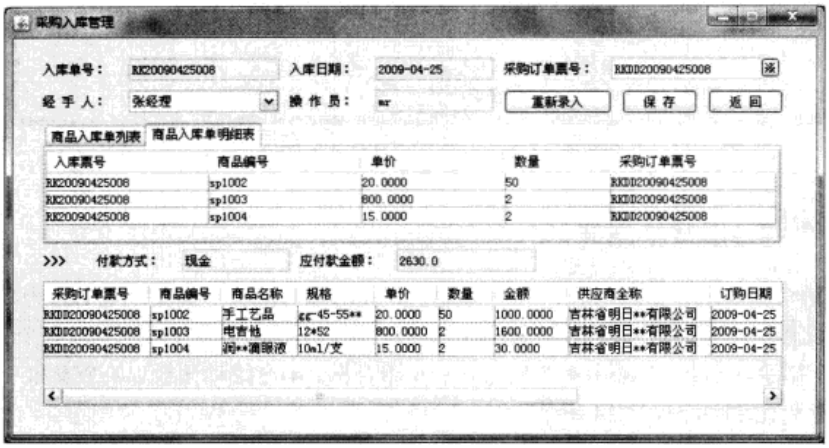


图 22.22 “采购入库管理”窗体界面

“采购入库管理”窗体中的“保存”按钮用于完成采购订单的入库操作，当用户选择完订单后，单击“保存”按钮，可以保存订单列表和明细表，同时库存增加。“保存”按钮的事件处理过程代码如下：

代码位置：简易通企业进销存管理系统\src\zzk\zhuoyue\dinggouruku\CaigouRukuFrame.java

```
private void btn_baocunActionPerformed(java.awt.event.ActionEvent evt) {
    if (tb_main.getRowCount()<=0){
        JOptionPane.showMessageDialog(this,"没有要保存的数据记录。");
        return;
    }
    if (cmb_jsr.getSelectedItem().toString().equals("请选择经手人")){
        JOptionPane.showMessageDialog(this,
          "请先选择经手人，然后单击采购订单票号右侧的按钮进行选择输入。");
        return;
    }
    try{
        con=daoConn.getConn();                                    // 获得数据库连接
        conn=daoConn.getConn();
        connec=daoConn.getConn();
        connect=daoConn.getConn();
        con.setAutoCommit(false);                                 // 取消自动提交
        conn.setAutoCommit(false);
        connec.setAutoCommit(false);
        connect.setAutoCommit(false);
        String sqldetail="insert into tb_ruku_detail values(?,?,?,?,?)";   // 保存入库明细信息
        ps=conn.prepareStatement(sqldetail);
        int tbDetailTotalRows=tb_detail.getModel().getRowCount();
        int tbDetailTotalColumns=tb_detail.getModel().getColumnCount();
```

```
String[] spbh=new String[tbDetailTotalRows];                    // 声明保存入库明细表中商品编号的数组
float[] spdj=new float[tbDetailTotalRows];                      // 声明保存入库明细表中商品单价的数组
int[] spsl=new int[tbDetailTotalRows];                          // 声明保存入库明细表中商品数量的数组
for (int r=0;r<tbDetailTotalRows;r++){
    for (int c=0;c<tbDetailTotalColumns;c++){
        if (c==3){
            ps.setInt(c+1,Integer.parseInt(tb_detail.getModel().getValueAt(r,c).toString()));
        }else if (c==2){
            ps.setFloat(c+1,Float.parseFloat(tb_detail.getModel().getValueAt(r,c).toString()));
        }else{
            ps.setString(c+1,tb_detail.getModel().getValueAt(r,c).toString());
        }
    }
    spbh[r]=tb_detail.getModel().getValueAt(r,1).toString();
    spdj[r]=Float.parseFloat(tb_detail.getModel().getValueAt(r,2).toString());
    spsl[r]=Integer.parseInt(tb_detail.getModel().getValueAt(r,3).toString());
    ps.executeUpdate();
    ps.addBatch();
}
// 定义保存入库列表的 SQL 语句
String sqlmain="insert into tb_ruku_main values(?,?,?,?,?,?,?,?,?,?,?,?,?,?)";
ps=con.prepareStatement(sqlmain);
int tbMailTotalRows=tb_main.getModel().getRowCount();
int tbMailTotalColumns=tb_main.getModel().getColumnCount();
for (int r=0;r<tbMailTotalRows;r++){
    for (int c=0;c<tbMailTotalColumns;c++){
        if (c==1){
            ps.setInt(c+1,Integer.parseInt(tb_main.getModel().getValueAt(r,c).toString()));
        }else if (c==2){
            ps.setFloat(c+1,Float.parseFloat(tb_main.getModel().getValueAt(r,c).toString()));
        }else{
            ps.setString(c+1,tb_main.getModel().getValueAt(r,c).toString());
        }
    }
    ps.executeUpdate();
    ps.addBatch();
}
// 实现入库和库存同步操作
st=connec.createStatement();
smt=connect.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
      ResultSet.CONCUR_UPDATABLE);
String[] sqlspinfo=new String[spbh.length];
String[] sqlkcinfo=new String[spbh.length];
String spsql="select id,spmc,jc,cd,gg,bz,dw from tb_spinfo where id='";
String kcsql="select dj,sl from tb_kucun where id='";
for (int i=0;i<spbh.length;i++){
    sqlspinfo[i]=spsql+spbh[i]+"'";
    sqlkcinfo[i]=kcsql+spbh[i]+"'";
    rs=st.executeQuery(sqlspinfo[i]);
```

```
                rst=smt.executeQuery(sqlkcinfo[i]);
                if (rst.next()){                          // 如果库存表中有匹配的记录，则修改库存表中的内容
                    if (rs.next()){
                        float dj=rst.getFloat("dj");
                        int sl=rst.getInt("sl");
                        int zsl=sl+spsl[i];
                        smt.executeUpdate("update kucun_tab set dj="+
                          spdj[i]+",sl="+zsl+" where id='"+spbh[i]+"'");
                    }
                }else{                                    // 如果库存表中没有匹配的记录，则向库存表中添加记录
                    if (rs.next()){
                        smt.executeUpdate("insert into tb_kucun values('"+rs.getString("id")+"','"+
                        rs.getString("spmc")+"','"+rs.getString("jc")+"','"+rs.getString("cd")+"','"+
                        rs.getString("gg")+"','"+rs.getString("bz")+"','"+rs.getString("dw")+"','"+
                        spdj[i]+","+spsl[i]+")");
                    }
                }
            }
            conn.commit();
            con.commit();
            connec.commit();
            connect.commit();
            btn_baocun.setEnabled(false);
            JOptionPane.showMessageDialog(this,"保存成功！！！");
        }catch(Exception ex){
            try{
                JOptionPane.showMessageDialog(this,"保存失败！！！");
                con.rollback();
                conn.rollback();
                connec.rollback();
                connect.rollback();
            }catch(Exception e){}
        }
    }
```

注意： 由于篇幅有限，这里只给出了“保存”按钮的事件处理过程，其他方法和组件的事件处理过程没有给出，可以参看光盘中的源程序。

22.14 入库信息查询模块设计

本模块使用的视图：v_ruku

“入库信息查询”模块负责查询商品的入库信息，可以根据选中的复选框，指定查询方式是单一条件查询还是多条件查询。通过两个组合框可以选择不同的字段进行查询，也可以指定是精确查询还是模糊查询。单击“查询”按钮可以查询指定条件的入库信息，单击“全部显示”按钮又可以显示所有入库商品的信息。“入库信息查询”窗体界面如图 22.23 所示。

图 22.23　“入库信息查询”窗体界面

22.14.1　设计入库信息查询窗体

在项目“简易通企业进销存管理系统”的 zzk.zhuoyue.dinggouruku 包中建立一个名称为 RukuChaxunFrame 的 JFrame 窗体，设置窗体布局为 null，在窗体上放一个滚动面板、一个表和一个面板，在面板上放置 3 个复选框、3 个文本框、3 个按钮和两个组合框。该窗体中所用到的关键组件如表 22.15 所示。

表 22.15　入库信息查询窗体用到的关键组件

控 件 类 型	控件 ID	主要属性设置	用　途
JTextField	txf_ziduanzhi	无	字段值
	txf_qishiriqi	无	起始日期
	txf_zongzhiriqi	无	终止日期
JComboBox	cmb_ziduanming	无	字段的名称
	cmb_fuhao	无	使用的符号
JButton	btn_chaxun	设置 text 属性为“查询”	按条件查询
	btn_quanbuxianshi	设置 text 属性为“全部显示”	显示所有入库商品的信息
	btn_fanhui	设置 text 属性为“返回”	返回主界面
JCheckBox	chk_xuanzetiaojian	设置 selected 属性为 true	按组合框中选择的字段查询
	chk_qishiriqi	无	按起始日期查询
	chk_zongzhiriqi	无	按终止日期查询
JTable	tb_showrukuinfo	设置 autoResizeMode 属性为 Off	显示入库商品信息

22.14.2　入库信息查询

在“入库信息查询”窗体界面中至少要选中一个复选框，如果选中了某个复选框并且指定该复选框的条件，则单击“查询”按钮，即可显示满足条件的信息。如果同时选中了多个复选框，并且为每个复选框都指定了条件，单击“查询”按钮，将显示同时满足多个条件的信息。“查询”按钮的事件处理过程代码如下：

代码位置：简易通企业进销存管理系统\src\zzk\zhuoyue\dinggouruku\RukuChaxunFrame.java

```
private void btn_chaxunActionPerformed(java.awt.event.ActionEvent evt) {
    String fieldName=cmb_ziduanming.getSelectedItem().toString();
    String sign=cmb_fuhao.getSelectedItem().toString();
```

```
        String value=txf_ziduanzhi.getText().trim();
        String startDate=txf_qishiriqi.getText().trim();
        String endDate=txf_zongzhiriqi.getText().trim();
        String sql="select * from v_ruku ";
        if (fieldName.equals("入库票号"))
            fieldName="rkid";                                    // if 语句后只有一条语句可以省略大括号
        if (fieldName.equals("商品名称")){
            fieldName="spmc";
        }
        if (fieldName.equals("产地")){
            fieldName="cd";
        }
        if (fieldName.equals("规格")){
            fieldName="gg";
        }
        if (fieldName.equals("供应商全称")){
            fieldName="gysqc";
        }
    // 根据选中的复选框生成满足相应条件的 SQL 语句
        if (chk_xuanzetiaojian.isSelected() && chk_qishiriqi.isSelected() && chk_zongzhiriqi.isSelected()){
            if (!value.equals("")   && !startDate.equals("") && !endDate.equals("")){
                if (sign.equals("等于")){
                    sql=sql+"where "+fieldName+"='"+value+"' and rkrq >='"+
    startDate+"' and rkrq<='"+endDate+"'";
                }else{
                    sql=sql+"where "+fieldName+" like '%"+value+"%' and rkrq >='"+
    startDate+"' and rkrq <='"+endDate+"'";
                }
            }else{
                JOptionPane.showMessageDialog(this,"输入的信息不全，请输入。");
                return;
            }
        }else if (chk_xuanzetiaojian.isSelected() && chk_qishiriqi.isSelected() &&
    !chk_zongzhiriqi.isSelected()){
            if (!value.equals("")   && !startDate.equals("")){
                if (sign.equals("等于")){
                    sql=sql+"where "+fieldName+"='"+value+"' and rkrq ='"+startDate+"'";
                }else{
                    sql=sql+"where "+fieldName+" like '%"+value+"%' and rkrq ='"+startDate+"'";
                }
            }else{
                JOptionPane.showMessageDialog(this,"输入的信息不全，请输入。");
                return;
            }
        }else if (chk_xuanzetiaojian.isSelected() &&
    !chk_qishiriqi.isSelected() && chk_zongzhiriqi.isSelected()){
            if (!value.equals("")   &&   !endDate.equals("")){
                if (sign.equals("等于")){
```

```
                sql=sql+"where "+fieldName+"='"+value+"' and rkrq ='"+endDate+"'";
            }else{
                sql=sql+"where "+fieldName+" like '%"+value+"%' and rkrq ='"+endDate+"'";
            }
        }else{
            JOptionPane.showMessageDialog(this,"输入的信息不全，请输入。");
            return;
        }
    }else if (chk_xuanzetiaojian.isSelected() &&
!chk_qishiriqi.isSelected() && !chk_zongzhiriqi.isSelected()){
        if (!value.equals("")){
            if (sign.equals("等于")){
                sql=sql+"where "+fieldName+"='"+value+"'";
            }else{
                sql=sql+"where "+fieldName+" like '%"+value+"%'";
            }
        }else{
            JOptionPane.showMessageDialog(this,"输入的信息不全，请输入。");
            return;
        }
    }else if (!chk_xuanzetiaojian.isSelected() &&
chk_qishiriqi.isSelected() && chk_zongzhiriqi.isSelected()){
        if (!startDate.equals("") && !endDate.equals("")){
            sql=sql+"where rkrq >='"+startDate+"' and rkrq<='"+endDate+"'";
        }else{
            JOptionPane.showMessageDialog(this,"输入的信息不全，请输入。");
            return;
        }
    }else if (!chk_xuanzetiaojian.isSelected() &&
chk_qishiriqi.isSelected() && !chk_zongzhiriqi.isSelected()){
        if (!startDate.equals("")){
            sql=sql+"where rkrq ='"+startDate+"'";
        }else{
            JOptionPane.showMessageDialog(this,"输入的信息不全，请输入。");
            return;
        }
    }else if (!chk_xuanzetiaojian.isSelected() &&
!chk_qishiriqi.isSelected() && chk_zongzhiriqi.isSelected()){
        if (!endDate.equals("")){
            sql=sql+"where rkrq ='"+endDate+"'";
        }else{
            JOptionPane.showMessageDialog(this,"输入的信息不全，请输入。");
            return;
        }
    }else{
        JOptionPane.showMessageDialog(this,"请指定查询条件！！！");
        return;
    }
```

```
    showDataToTable(sql);                                    // 调用方法查询信息
}
```

在表格中显示查询结果的方法，当要查询商品入库信息时，只需要把满足条件的 SQL 语句作为实参传递给该方法，就会在表格中显示出相应的商品入库信息，其程序代码如下：

代码位置：简易通企业进销存管理系统\src\zzk\zhuoyue\dinggouruku\RukuChaxunFrame.java

```
private void showDataToTable(String sql){
    String[] columnNames = {"入库票号","商品编号","商品名称", "产地", "规格", "单价", "数量","金额",
"结算方式","入库日期", "批号", "批准文号","供应商全称","验收结论","备注"};
    String[][] tableValues = getData(sql);
    tableModel = new DefaultTableModel(tableValues, columnNames);
    tb_showrukuinfo.setModel(tableModel);
    TableColumn column=tb_showrukuinfo.getColumn("入库票号");      // 得到入库票号列的 TableColumn
    column.setPreferredWidth(110) ;                               // 设置表格中入库票号列的列宽
    column=tb_showrukuinfo.getColumn("产地");
    column.setPreferredWidth(100) ;
}
```

注意： 由于篇幅有限，这里只给出了“查询”按钮的事件处理过程和该事件处理过程所用到的方法 showDataToTable(String sql)，其他方法和组件的事件处理过程没有给出，可以参看光盘中的源程序。

22.15 入库退货管理模块设计

本模块使用的数据表：tb_rkth_main、tb_jsr、tb_gysinfo、tb_spinfo、tb_kucun 和 tb_rkth_detail

“入库退货管理”模块负责入库商品的退货操作，退货时要先选择经手人、结款方式和供应商，然后单击“录入数据”按钮，打开“录入商品入库退货信息”窗体输入退货信息。“入库退货管理”窗体界面如图 22.24 所示。

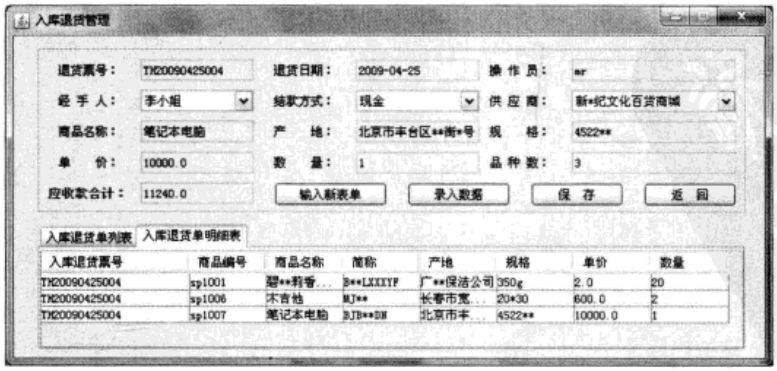


图 22.24 “入库退货管理”窗体界面

“录入商品入库退货信息”窗体用于入库退货信息的输入操作，该窗体的 3 个组合框中显示的是“入库退货管理”窗体中所选供应商的商品信息。在任一组合框中选择商品信息后按 Enter 键，其他组件会自动显示相关内容，用户只需输入单价和数量即可，在“数量”文本框中按 Enter 键或单击“确定”按钮后，如果输入的数量超出库存会有相应提示，否则直接添加到“入库退货管理”窗体中。“录入商品入库退货信息”

窗体界面如图 22.25 所示。

图 22.25　“录入商品入库退货信息”窗体界面

22.15.1　设计入库退货管理窗体

在项目“简易通企业进销存管理系统”的 zzk.zhuoyue.dinggouruku 包中建立一个名称为 RukuTuihuoFrame 的 JFrame 窗体，设置窗体布局为空布局，并在窗体上放一个空布局的面板，在面板上放置 13 个标签、10 个文本框、4 个按钮和 3 个组合框，再在窗体上放一个选项卡面板，并在选项卡面板上放两个面板，每个面板上各放一个滚动面板和一个表。该窗体中所用到的关键组件如表 22.16 所示。

表 22.16　入库退货管理窗体用到的关键组件

控 件 类 型	控件 ID	主要属性设置	用　途
JTextField	txf_thph	设置 editable 属性为 false	退货票号
	txf_thrq	设置 editable 属性为 false	退货日期
	txf_spmc	设置 editable 属性为 false	商品名称
	txf_czy	设置 editable 属性为 false	操作员
	txf_chandi	设置 editable 属性为 false	产地
	txf_guige	设置 editable 属性为 false	规格
	txf_danjia	设置 editable 属性为 false	单价
	txf_shuliang	设置 editable 属性为 false	数量
	txf_pzs	设置 editable 属性为 false	品种数
	txf_yskhj	设置 editable 属性为 false	应收款合计
JComboBox	cmb_gysqc	无	显示有入库记录的供应商全称
	cmb_jsfs	无	显示结款方式
	cmb_jsr	无	显示所有启用的经手人
JButton	btn_shuruxinbiaodan	设置 text 属性为“输入新表单”	重新输入新的表单
	btn_lurushuju	设置 text 属性为“录入数据”	打开“录入商品入库退货信息”窗体
	btn_baocun	设置 text 属性为“保存”	保存商品入库退货信息
	btn_fanhui	设置 text 属性为“返回”	返回主界面
JTable	tb_main	设置 autoResizeMode 属性为 Off	显示商品入库退货单列表
	tb_detail	设置 autoResizeMode 属性为 Off	显示商品入库退货单明细表

22.15.2 设计录入商品入库退货信息窗体

在项目“简易通企业进销存管理系统”的 zzk.zhuoyue.dinggouruku 包中建立一个 JFrame 窗体，其名称为 InputRukuTuihuoFrame，设置窗体布局为空布局，并在窗体上放两个空布局的面板，在第一个面板上放置 7 个标签、4 个文本框和 3 个组合框，在第二个面板上放置两个按钮。该窗体中所用到的关键组件如表 22.17 所示。

表 22.17 录入商品入库退货信息窗体用到的关键组件

控 件 类 型	控件 ID	主要属性设置	用 途
JTextField	txf_gg	设置 editable 属性为 false	规格
	txf_cd	设置 editable 属性为 false	产地
	txf_dj	无	单价
	txf_sl	无	数量
JComboBox	cmb_spbh	无	显示商品编号
	cmb_spmc	无	显示商品名称
	cmb_jc	无	显示商品的简称
JButton	btn_ok	设置 text 属性为“确定”	确认本次输入
	btn_cancel	设置 text 属性为“取消”	取消本次输入

22.15.3 设计保存输入信息的类

在项目“简易通企业进销存管理系统”的 zzk.zhuoyue.dinggouruku 包中创建一个类 SaveRukuTuihuoInfo，用于保存“录入商品入库退货信息”窗体中输入的数据信息，并在“入库退货管理”窗体中使用。该类的主要程序代码如下：

代码位置：简易通企业进销存管理系统\src\zzk\zhuoyue\dinggouruku\SaveRukuTuihuoInfo.java

```
package zzk.zhuoyue.dinggouruku;
public class SaveRukuTuihuoInfo {
    private String spbh;                                    // 定义商品编号
    private String spmc;                                    // 定义商品名称
    private String jc;                                      // 定义简称
    private String cd;                                      // 定义产地
    private String gg;                                      // 定义规格
    private float dj;                                       // 定义单价
    private int sl;                                         // 定义数量
    // 保存按钮状态，如果单击“确定”按钮，则把 button 的值设置为 true，否则设置为 false
    private boolean button;
    // 这里省略了 is、set 和 get 方法
}
```

“录入商品入库退货信息”窗体中的“确定”按钮，用于判断用户输入的退货信息是否齐全以及是否超出库存，销毁本窗体并把输入的退货信息传递到“入库退货管理”窗体中。“确定”按钮的事件处理过程代码如下：

代码位置：简易通企业进销存管理系统\src\zzk\zhuoyue\dinggouruku\InputRukuTuihuoFrame.java

```
private void btn_okActionPerformed(java.awt.event.ActionEvent evt) {
    if (txf_dj.getText().length()<=0){
        JOptionPane.showMessageDialog(this,"请输入单价。");
        txf_dj.requestFocus();
        return;
    }
    if (txf_sl.getText().length()<=0){
        JOptionPane.showMessageDialog(this,"请输入数量。");
        txf_sl.requestFocus();
        return;
    }
    int srsl=Integer.parseInt(txf_sl.getText());
    int kcsl=checkKucunShuLiang();                                    // 调用方法，获得库存数量
    if (kcsl>0 && kcsl<srsl){
        JOptionPane.showMessageDialog(this,"库存不足，请输入小于等于"+kcsl+"的数量值。");
        return;
    }else if (kcsl<=0){
        JOptionPane.showMessageDialog(this,"库存数量为零，不能退货。");
        return;
    }
    Vector v=RukuTuihuoFrame.getSpid();                               // 获得入库退货管理窗体中的商品编号
    // 把本次输入的退货信息传递到入库退货管理窗体中
    RukuTuihuoFrame.setSaveRukuTuihuo(saveRukuInfo(true));            // 这里用到的方法可以参见光盘
    if (v==null){
        v=new Vector();
    }
    v.add(cmb_spbh.getSelectedItem().toString());                     // 在向量中添加商品编号
    RukuTuihuoFrame.setSpid(v);                                       // 传递向量到入库退货管理窗体
    Vector vec=RukuTuihuoFrame.getSpje();                             // 获得入库退货管理窗体中的金额向量
    if (vec==null){
        vec=new Vector();
    }
    float je=Float.parseFloat(txf_dj.getText().trim());               // 单价
    int sl=Integer.parseInt(txf_sl.getText().trim());                 // 数量
vec.add(je*sl);                                                       // 计算应收款合计并添加到向量中
// 保存应收款合计到 RukuTuihuoFrame 窗体
    RukuTuihuoFrame.setSpje(vec);
    this.dispose();                                                   // 销毁该窗体
}
```

“入库退货管理”窗体处于激活状态时的事件处理过程，用于在表中显示输入的数据，由于在“录入商品入库退货信息”窗体中单击“确定”按钮将销毁窗体，此时“入库退货管理”窗体处于激活状态，因此可以响应窗体激活事件。“入库退货管理”窗体处于激活状态时的事件处理过程代码如下：

代码位置：简易通企业进销存管理系统\src\zzk\zhuoyue\dinggouruku\RukuTuihuoFrame.java

```
private void formWindowActivated(java.awt.event.WindowEvent evt) {
    if (RukuTuihuoFrame.getSaveRukuTuihuo()!=null &&
RukuTuihuoFrame.getSaveRukuTuihuo().isButton()){
        int pzs=0;                                                    // 商品的品种数量
        if (getSpid().size()>0){
```

```
            pzs=getSpid().size();
            txf_pzs.setText(String.valueOf(pzs));
        }
        float hjje=0f;                                          // 商品的合计金额
        if (getSpje().size()>0){
            for (int i=0;i<getSpje().size();i++){
                hjje=hjje+Float.parseFloat(getSpje().get(i).toString());
            }
        }
        txf_yskhj.setText(String.valueOf(hjje));                // 为文本框赋值为应收款合计
        // 定义从控件获得信息的变量
        String rkthph=txf_thph.getText().trim();
        String spbh=RukuTuihuoFrame.getSaveRukuTuihuo().getSpbh();
        String spmc=RukuTuihuoFrame.getSaveRukuTuihuo().getSpmc();
        String jc=RukuTuihuoFrame.getSaveRukuTuihuo().getJc();
        String cd=RukuTuihuoFrame.getSaveRukuTuihuo().getCd();
        String gg=RukuTuihuoFrame.getSaveRukuTuihuo().getGg();
        String dj=String.valueOf(RukuTuihuoFrame.getSaveRukuTuihuo().getDj());
        String sl=String.valueOf(RukuTuihuoFrame.getSaveRukuTuihuo().getSl());
        txf_spmc.setText(spmc);                                 // 为文本框赋值
        txf_chandi.setText(cd);
        txf_guige.setText(gg);
        txf_danjia.setText(dj);
        txf_shuliang.setText(sl);
        String gysqc=cmb_gysqc.getSelectedItem().toString();
        String thrq=txf_thrq.getText();
        String czy=txf_czy.getText();
        String jsr=cmb_jsr.getSelectedItem().toString();
        String jsfs=cmb_jsfs.getSelectedItem().toString();
        if(rowCount==1){                                        // rowCount 为全局变量，初值为 1
            String[] valueMain={rkthph,String.valueOf(pzs),String.valueOf(hjje),gysqc,thrq,czy,jsr,jsfs};
            tableModelMain.addRow(valueMain);                   // 在退货单列表中添加新行
        }else{
            tableModelMain.setValueAt(String.valueOf(pzs), 0, 1);   // 修改退货单列表中的品种数
            tableModelMain.setValueAt(String.valueOf(hjje), 0, 2);  // 修改退货单列表中的金额
        }
        String[] valueDetail={rkthph,spbh,spmc,jc,cd,gg,String.valueOf(dj),String.valueOf(sl)};
        tableModelDetail.addRow(valueDetail);                   // 在退货单明细表中添加新行
        RukuTuihuoFrame.getSaveRukuTuihuo().setButton(false);
        rowCount++;
    }
}
```

“入库退货管理”窗体中的“保存”按钮可以保存入库退货信息列表和明细表，同时更新库存，减少库存量。“保存”按钮的事件处理过程代码如下：

代码位置：简易通企业进销存管理系统\src\zzk\zhuoyue\dinggouruku\RukuTuihuoFrame.java

```
private void btn_baocunActionPerformed(java.awt.event.ActionEvent evt) {
    if (tb_main.getRowCount()<=0){
        JOptionPane.showMessageDialog(this,
"没有要保存的数据记录。\r\n 请单击“录入数据”按钮输入数据。");
```

```
        return;
    }
    try{
        con=daoConn.getConn();                                  // 建立数据库连接对象
        conn=daoConn.getConn();
        connect=daoConn.getConn();
        con.setAutoCommit(false);                               // 取消自动提交
        conn.setAutoCommit(false);
        connect.setAutoCommit(false);
        // 保存入库退货明细信息
        String sqldetail="insert into tb_rkth_detail values(?,?,?,?,?,?,?,?)";
        ps=conn.prepareStatement(sqldetail);
        int tbDetailTotalRows=tableModelDetail.getRowCount();
        int tbDetailTotalColumns=tableModelDetail.getColumnCount();
        String[] spbh=new String[tbDetailTotalRows];            // 保存入库退货明细表中商品编号的数组
        float[] spdj=new float[tbDetailTotalRows];              // 保存入库退货明细表中商品单价的数组
        int[] spsl=new int[tbDetailTotalRows];                  // 保存入库退货明细表中商品数量的数组
        // 为参数赋值
        for (int r=0;r<tbDetailTotalRows;r++){
            for (int c=0;c<tbDetailTotalColumns;c++){
                if (c==7){
                    ps.setInt(c+1,Integer.parseInt(tableModelDetail.getValueAt(r,c).toString()));
                }else if (c==6){
                    ps.setFloat(c+1,Float.parseFloat(tableModelDetail.getValueAt(r,c).toString()));
                }else{
                    ps.setString(c+1,tableModelDetail.getValueAt(r,c).toString());
                }
            }
            // 下面 3 条语句分别给商品编号、单价和数量 3 个数组赋值
            spbh[r]=tableModelDetail.getValueAt(r,1).toString();
            spdj[r]=Float.parseFloat(tableModelDetail.getValueAt(r,6).toString());
            spsl[r]=Integer.parseInt(tableModelDetail.getValueAt(r,7).toString());
            ps.executeUpdate();                                 // 执行 SQL 语句
            ps.addBatch();                                      // 批量添加
        }
        // 保存入库退货列表
        String sqlmain="insert into tb_rkth_main values(?,?,?,?,?,?,?,?)";
        ps=con.prepareStatement(sqlmain);
        int tbMainTotalColumns=tableModelMain.getColumnCount();
        // 为 SQL 语句的参数赋值
        for (int c=0;c<tbMainTotalColumns;c++){
            if (c==1){
                ps.setInt(c+1,Integer.parseInt(tableModelMain.getValueAt(0,c).toString()));
            }else if (c==2){
                ps.setFloat(c+1,Float.parseFloat(tableModelMain.getValueAt(0,c).toString()));
            }else{
                ps.setString(c+1,tableModelMain.getValueAt(0,c).toString());
            }
        }
```

```
            ps.executeUpdate();                                    // 执行 SQL 语句
            ps.addBatch();                                         // 批量添加
            // 实现入库退货和库存同步操作
            smt=connect.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
    ResultSet.CONCUR_UPDATABLE);
            String[] sqlkcinfo=new String[spbh.length];            // 定义数组
            String kcsql="select dj,sl from tb_kucun where id='";
            for (int i=0;i<spbh.length;i++){
                sqlkcinfo[i]=kcsql+spbh[i]+"'";
                rst=smt.executeQuery(sqlkcinfo[i]);
                if (rst.next()){
                    int sl=rst.getInt("sl");                       // 库存中的原数量
                    int zsl=sl-spsl[i];                            // 更新库存的新数量
                    smt.executeUpdate("update tb_kucun set sl="+zsl+" where id='"+spbh[i]+"'");
                }
            }
            conn.commit();                                         // 提交数据，并将其保存到数据库中
            con.commit();
            connect.commit();
            JOptionPane.showMessageDialog(this,"保存成功！！！");
        }catch(Exception ex){
            try{
                ex.printStackTrace();
                JOptionPane.showMessageDialog(this,"保存失败！！！");
                con.rollback();                                    // 回滚数据，取消保存
                conn.rollback();
                connect.rollback();
            }catch(Exception e){
            }
        }
        init();                                                    // 调用方法初始化退货票号和退货日期
        rowCount=1;                                                // 保存行数的成员变量，初值为 1
        if (RukuTuihuoFrame.getSaveRukuTuihuo()!=null){
            RukuTuihuoFrame.getSaveRukuTuihuo().setButton(false);
        }
        …这里省略了非关键代码
    }
```

注意：由于篇幅有限，这里只给出了一个保存退货信息的类 SaveRukuTuihuoInfo 和部分按钮的事件处理过程，其他组件的事件处理过程和方法的代码没有给出，可以参看光盘中的源程序。

22.16 库存盘点窗体模块设计

本模块使用的数据表：tb_kucun

“库存盘点窗体”模块负责库存商品的盘点操作，当用户在表格的盘点数量单元格中输入盘点数后按 Enter 键，系统会自动计算损溢数量和损溢金额。“库存盘点窗体”界面如图 22.26 所示。

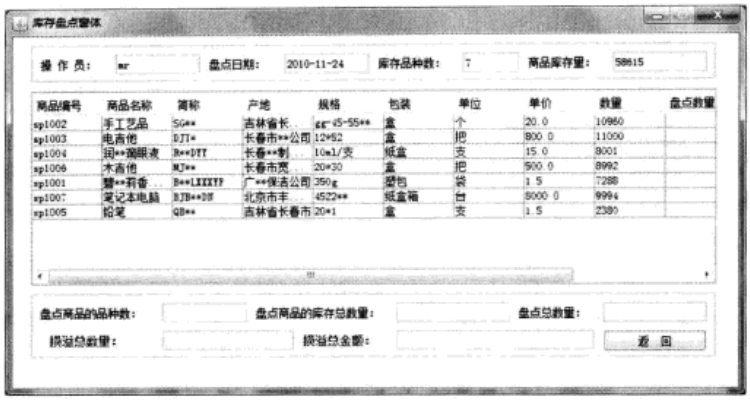


图 22.26　“库存盘点窗体”界面

22.16.1　设计库存盘点窗体

在项目“简易通企业进销存管理系统”中建立一个名为 zzk.zhuoyue.kucun 的包，在该包中建立一个名称为 KucunPandianFrame 的 JFrame 窗体，设置窗体布局为空布局，并在窗体上放两个空布局的面板和一个滚动面板，在第一个面板上放置 4 个标签和 4 个文本框，在第二个面板上放置 5 个标签、5 个文本框和一个按钮，在滚动面板中放置一个表。该窗体中所用到的关键组件如表 22.18 所示。

表 22.18　库存盘点窗体用到的关键组件

控 件 类 型	控件 ID	主要属性设置	用　途
JTextField	txf_czy	设置 editable 属性为 false	操作员
	txf_pdrq	设置 editable 属性为 false	盘点日期
	txf_pzs	设置 editable 属性为 false	库存中的货品种数
	txf_spzsl	设置 editable 属性为 false	库存中所有商品的总数量
	txf_pdpzs	设置 editable 属性为 false	盘点商品的品种数
	txf_pdkcsl	设置 editable 属性为 false	盘点商品的库存总数量
	txf_pdzsl	设置 editable 属性为 false	盘点商品的总数量
	txf_syzsl	设置 editable 属性为 false	损溢商品的总数量
	txf_syzje	设置 editable 属性为 false	损溢商品的总金额
JTable	tb_showspinfo	设置 autoResizeMode 属性为 Off	显示库存商品信息
JButton	btn_fanhui	设置 text 属性为“返回”	返回主窗体

22.16.2　库存盘点

在“库存盘点窗体”的某种商品所在行的盘点数量单元格中输入盘点数后按 Enter 键，系统会自动计算其损溢数量和损溢金额，当盘点数量大于库存数量时，损溢数量和损溢金额显示负值；当盘点数量小于库存数量时，损溢数量和损溢金额显示正值；当盘点数量等于库存数量时，损溢数量和损溢金额显示为 0。“库存盘点窗体”中，表格的键盘释放事件程序代码如下：

代码位置：简易通企业进销存管理系统\src\zzk\zhuoyue\kucun\KucunPandianFrame.java

```
private void tb_showspinfoKeyReleased(java.awt.event.KeyEvent evt) {
    if (evt.getKeyCode()==KeyEvent.VK_ENTER){
```

```
            try{
                int totalRows=tb_showspinfo.getRowCount();
                int totalPdkcsl=0;                                    // 定义盘点的商品库存总数量
                int totalPdpzs=0;                                     // 定义盘点品种数
                int totalPdsl=0;                                      // 定义盘点总数量
                int totalSysl=0;                                      // 定义损溢总数量
                float totalSyje=0;                                    // 定义损溢总金额
                for (int row=0;row<totalRows;row++){
                    Object pdslObject=tableModel.getValueAt(row,9);   // 获得表模型中输入的盘点数量
                    Object slObject=tableModel.getValueAt(row,8);     // 获得表模型中的库存数量
                    Object djObject=tableModel.getValueAt(row,7);     // 获得表模型中的商品单价
                    int pdsl=0;
                    int sl=0;
                    float dj=0f;
                    if (pdslObject!=null && pdslObject.toString().trim().length()>0 &&
                        slObject!=null && slObject.toString().trim().length()>0   &&
                        djObject!=null && djObject.toString().trim().length()>0){
                        pdsl=Integer.parseInt(pdslObject.toString());     // 将盘点数量转换为整数
                        sl=Integer.parseInt(slObject.toString());         // 将库存数量转换为整数
                        dj=Float.parseFloat(djObject.toString());         // 将商品单价转换为单精度型
                        totalPdkcsl=totalPdkcsl+sl;                       // 计算所有进行盘点商品的库存总量
                        totalPdpzs++;                                     // 计算进行盘点的商品的品种数
                        totalPdsl=totalPdsl+pdsl;                         // 计算所有进行盘点商品的盘点总量
                        totalSysl=totalSysl+(sl-pdsl);                    // 计算损溢总数
                        totalSyje=totalSyje+((sl-pdsl)*dj);               // 计算损溢总金额
                        txf_pdkcsl.setText(String.valueOf(totalPdkcsl));
                        txf_pdpzs.setText(String.valueOf(totalPdpzs));
                        txf_pdzsl.setText(String.valueOf(totalPdsl));
                        txf_syzsl.setText(String.valueOf(totalSysl));
                        txf_syzje.setText(String.valueOf(totalSyje));
                        // 在表中显示损溢数量和损溢金额
                        tableModel.setValueAt(String.valueOf(sl-pdsl), row,10);
                        tableModel.setValueAt(String.valueOf(dj*(sl-pdsl)), row,11);
                    }
                }
            }catch(Exception ex){
                JOptionPane.showMessageDialog(this,"您输入的盘点数量不是有效的数字。");
            }
        }
    }
```

注意：由于篇幅有限，这里只给出了表格的键盘释放事件处理过程，其他方法和组件的事件处理过程没有给出，可以参看光盘中的源程序。

22.17 库存调拨模块设计

本模块使用的数据表：tb_kucun 和 tb_kcdb

“库存调拨窗体”模块包含两个选项卡，“添加商品调拨信息”选项卡用于库存商品的调拨操作，“查询商品调拨信息”选项卡用于查询商品的调拨信息。“库存调拨窗体”界面如图 22.27 所示。

图 22.27　“库存调拨窗体”界面

22.17.1　设计库存调拨窗体

在项目“简易通企业进销存管理系统”的 zzk.zhuoyue.kucun 包中建立一个名称为 KucunDiaoboFrame 的 JFrame 窗体，设置窗体布局为“边框式”布局，并在窗体中心放一个选项卡面板，在选项卡面板上放两个空布局的面板，在第一个面板上放置 11 个标签、12 个文本框（其中有一个在商品编号文本框的后方）、4 个按钮和一个滚动面板，在滚动面板中放置一个表。在第二个面板上放置一个标签、一个文本框、一个组合框、3 个按钮和一个滚动面板，在滚动面板中放置一个表。该窗体中所用到的关键组件如表 22.19 所示。

表 22.19　库存调拨窗体用到的关键组件

控 件 类 型	控件 ID	主要属性设置	用　途
JTextField	txf_czy	设置 editable 属性为 false	操作员
	txf_dbrq	设置 editable 属性为 false	调拨日期
	txf_dbph	设置 editable 属性为 false	调拨票号
	txf_spbh	设置 editable 属性为 false	商品编号
	txf_kucun	无	随着表中可编辑单元格移动
	txf_spmc	设置 editable 属性为 false	商品名称
	txf_jc	设置 editable 属性为 false	简称
	txf_dw	设置 editable 属性为 false	单位
	txf_cd	设置 editable 属性为 false	产地
	txf_gg	设置 editable 属性为 false	规格
	txf_dbsl	无	调拨数量
	txf_memo	无	调拨原因
	txf_fieldValue	无	查询内容
JTable	tb_kucun	设置 autoResizeMode 属性为 Off	显示库存商品信息
	tb_diaobo	设置 autoResizeMode 属性为 Off	显示商品调拨信息
JComboBox	cmb_fieldName	无	查询字段
JButton	btn_save	设置 text 属性为“保存”	保存商品调拨信息
	btn_reInput	设置 text 属性为“重输”	重新输入调拨信息
	btn_showData	设置 text 属性为“...”	在表中显示数据供用户选择
	btn_query	设置 text 属性为“查询”	查询商品调拨信息
	btn_showAll	设置 text 属性为“全部”	显示所有调拨信息
	btn_csReturn	设置 text 属性为“返回”	返回主窗体

22.17.2 添加商品调拨信息

在“添加商品调拨信息”选项卡中单击…按钮，可以打开商品库存信息列表，如图 22.28 所示。双击需要调拨的商品所在的行，就可以把该商品的信息添加到相应的组件中。如图 22.29 所示，用户只需输入调拨数量和调拨说明，单击“保存”按钮即可保存调拨信息。

图 22.28 “添加商品调拨信息”选项卡

图 22.29 保存调拨信息

“添加商品调拨信息”选项卡用于添加商品调拨信息，单击“商品编号”文本框后的按钮，可以打开商品信息列表。然后双击需要调拨的商品信息，自动添加到相应的组件中，输入数量和说明，单击“保存”按钮即可保存调拨信息。“保存”按钮的事件处理过程代码如下：

代码位置：简易通企业进销存管理系统\src\zzk\zhuoyue\kucun\KucunDiaoboFrame.java

```
private void btn_saveActionPerformed(java.awt.event.ActionEvent evt) {
    try{
        String dbid=txf_dbph.getText().trim();                    // 定义调拨票号
        …这里省略了其他获得信息的方法
        if (dbid.equals("") || spid.equals("") || spmc.equals("") || memo.equals("") || dbrq.equals("")){
            JOptionPane.showMessageDialog(this,"信息不全，请输入重要信息。");
            return;
        }
        int dbsl=0;                                               // 定义调拨数量
        try{
            dbsl=Integer.parseInt(txf_dbsl.getText().trim());     // 把调拨数量转换为整数
        }catch(Exception e){
            JOptionPane.showMessageDialog(this,"调拨数量必须为数字，且必须输入。");
            return;
        }
        // 获得库存数量
        Object obj=tb_kucun.getModel().getValueAt(tb_kucun.getSelectedRow(),8);
        if (obj!=null && dbsl>Integer.parseInt(obj.toString())){
            JOptionPane.showMessageDialog(this,"库存不足，该商品库存数量为"+obj.toString()+"。");
            return;
        }
        int kcsl=0;                                               // 定义库存数量
        if (obj!=null){
            kcsl=Integer.parseInt(obj.toString());
        }
        conn=daoConn.getConn();
        con=daoConn.getConn();
```

```
            conn.setAutoCommit(false);                                    // 取消自动提交
            con.setAutoCommit(false);
            // 定义更新库存的 SQL 语句
            String sq="update tb_kucun set sl="+(kcsl-dbsl)+" where id='"+spid+"'";
            stm=con.createStatement();
            stm.executeUpdate(sq);                                        // 执行 SQL 语句
            // 定义添加调拨信息的 SQL 语句
            String sql="insert into tb_kcdb values(?,?,?,?,?,?,?,?,?,?,?)";
            ps=conn.prepareStatement(sql);
            ps.setString(1,dbid);
            ps.setString(2,spid);
            ps.setString(3,spmc);
            ps.setString(4,jc);
            ps.setString(5,cd);
            ps.setString(6,dw);
            ps.setString(7,gg);
            ps.setInt(8,dbsl);
            ps.setString(9,dbrq);
            ps.setString(10,memo);
            ps.setString(11,czy);
            ps.executeUpdate();
            conn.commit();                                                // 提交数据
            con.commit();
            btn_save.setEnabled(false);                                   // 设置“保存”按钮不可用
            String dbsql="select * from tb_kcdb";
            queryDiaobo(dbsql);                                           // 查询商品调拨信息同步更新
            JOptionPane.showMessageDialog(this,"保存成功。");
        }catch(Exception ex){
            try{
                conn.rollback();                                          // 回滚数据
                con.rollback();
                JOptionPane.showMessageDialog(this,"保存失败。\r\n"+ex.getMessage());
            }catch(Exception e){
                e.printStackTrace();
            }
        }
    }
```

注意：由于篇幅有限，这里只给出了“保存”按钮的事件处理过程，其他方法和组件的事件处理过程没有给出，可以参看光盘中的源程序。

22.17.3　查询商品调拨信息

在“查询商品调拨信息”选项卡的组合框中选择查询字段，然后在其右侧的文本框中输入查询内容，单击“查询”按钮可以查看调拨商品的信息；单击“全部”按钮，可以查看所有调拨商品的信息，如图 22.30 所示。

“查询商品调拨信息”选项卡用于对商品的调拨信息进行查询，从组合框中选择字段后，在文本框中输入查询内容，单击“查询”按钮，可以查询指定条件的调拨信息；单击“全部”按钮，可以显示全部调拨商品的信息。“查询”按钮的事件处理过程代码如下：

图 22.30 “查询商品调拨信息”选项卡

代码位置：简易通企业进销存管理系统\src\zzk\zhuoyue\kucun\KucunDiaoboFrame.java

```
private void btn_queryActionPerformed(java.awt.event.ActionEvent evt) {
    String dbsql="select * from tb_kcdb";                          // 定义 SQL 语句
    String fieldName=cmb_fieldName.getSelectedItem().toString();
    String fieldValue=txf_fieldValue.getText().trim();
    if (fieldName.equals("商品名称")){
        fieldName="spmc";                                          // 获得商品名称字段
    }else if (fieldName.equals("简称")){
        fieldName="jc";                                            // 获得简称字段
    }else{
        fieldName="dbrq";                                          // 获得调拨日期字段
    }
    if (fieldValue.equals("")){
        JOptionPane.showMessageDialog(this,"请输入要查询的内容。");
        txf_fieldValue.requestFocus();
        return;
    }
    dbsql=dbsql+" where "+fieldName+"='"+fieldValue+"'";           // 获得 SQL 语句
    queryDiaobo(dbsql);                                            // 调用过程
}
```

定义一个方法 queryDiaobo(String sql)，调用该方法需要传递一个查询语句，用于显示商品调拨信息，其程序代码如下：

代码位置：简易通企业进销存管理系统\src\zzk\zhuoyue\kucun\KucunDiaoboFrame.java

```
private void queryDiaobo(String sql){
    String[] dbtitles={"调拨票号","商品编号","商品名称","简称","产地",
"单位","规格","调拨数量","调拨日期","调拨原因","操作员"};
    DefaultTableModel dbmodel=showDataToTable(dbtitles,sql);
    tb_diaobo.setModel(dbmodel);
    TableColumn column=tb_diaobo.getColumn("调拨票号");            // 获得调拨票号表格列
    column.setPreferredWidth(120) ;                                // 设置列宽
}
```

注意：由于篇幅有限，这里只给出了“查询”按钮的事件处理过程和该事件处理过程所用到的方法 queryDiaobo(String sql)，其他方法和组件的事件处理过程没有给出，可以参看光盘中的源程序。

22.18　销售订单管理模块设计

本模块使用的数据表：tb_xsqd_main、tb_xsqd_detail、tb_spinfo、tb_kucun、tb_khinfo 和 tb_jsr

“销售订单管理”模块负责商品销售前的订购操作，有销售订单的商品才能销售出库，否则不能销售出库。“销售订单管理”窗体界面如图 22.31 所示。

图 22.31　“销售订单管理”窗体界面

22.18.1　设计销售订单管理窗体

在项目“简易通企业进销存管理系统”中新建一个 zzk.zhuoyue.xiaoshouchuku 包，然后在该包中建立一个名称为 XiaoshouQiandanFrame 的 JFrame 窗体，设置窗体布局为空布局，在窗体上放两个空布局的面板、两个滚动面板和一个文本框，在第一个面板上放置 9 个标签、6 个文本框和 3 个组合框，在第二个面板上放 5 个按钮，在两个滚动面板中各放一个表。该窗体中所用到的关键组件如表 22.20 所示。

表 22.20　销售订单管理窗体用到的关键组件

控 件 类 型	控件 ID	主要属性设置	用　　途
JTextField	txf_xsddid	设置 editable 属性为 false	销售订单 ID
	txf_czy	设置 editable 属性为 false	操作员
	txf_xsqdrq	设置 editable 属性为 false	销售签单日期
	txf_ysjl	无	验收结论
	txf_pzs	设置 editable 属性为 false	品种数量
	txf_hjje	设置 editable 属性为 false	合计金额
	txf_tablecell	无	随表中的当前单元格移动
JComboBox	cmb_jsr	无	显示所有不重复的经手人
	cmb_jsfs	设置 model 属性为“现金，支票”	显示结款方式
	cmb_kh	无	显示所有不重复的客户
JButton	btn_newxiaoshoudan	设置 text 属性为“输入新销售单”	重新输入新销售单
	btn_newjilu	设置 text 属性为“添加新记录”	输入新记录
	btn_baocun	设置 text 属性为“保存销售单”	保存销售单
	btn_shanchu	设置 text 属性为“删除记录”	删除记录
	btn_fanhui	设置 text 属性为“返回”	返回主界面
JTable	tb_newjilu	设置 autoResizeMode 属性为 Off	显示本销售单中添加的记录
	tb_xuanzejilu	设置 autoResizeMode 属性为 Off	显示可以销售的商品信息

22.18.2 销售订单管理

在“销售订单管理”窗体界面中单击“输入新销售单”按钮，可以初始化组件，选择经手人、客户并输入验收结论后，单击“添加新记录”按钮，就会显示一个含有可以销售的商品信息表，双击表中需要销售的商品所在的行添加商品信息，输入单价和数量后按 Enter 键，完成本记录的添加。要继续添加记录，可再次单击“添加新记录”按钮，重复上面的操作即可。选择已添加记录所在的行，单击“删除记录”按钮，可以删除所选记录；单击“保存销售单”按钮，可以保存本销售单信息。

方法 initTable()可以使文本框随着表格中的活动单元格移动，双击表中数据行所在的单元格，可以使文本框获得焦点，该方法的程序代码如下：

代码位置：简易通企业进销存管理系统\src\zzk\zhuoyue\xiaoshouchuku\XiaoshouQiandanFrame.java

```
private void initTable(JTable table,JTextField tf) {
    // 获得表的 TableColumn 对象
    TableColumn column=table.getColumnModel().getColumn(0);
    DefaultCellEditor editor = new DefaultCellEditor(tf);          // 设置表的单元格编辑器为文本框 tf
    editor.setClickCountToStart(2);                                // 在单元格中单击两次启动编辑器
    column.setCellEditor(editor);                                  // 为列设置编辑器
    // 通过循环为表的所有列都设置编辑器
    for (int i=1;i<11;i++){
        column = table.getColumnModel().getColumn(i);              // 获得表中指定的 TableColumn 对象
        editor = new DefaultCellEditor(tf);                        // 设置表的单元格编辑器为文本框 tf
        editor.setClickCountToStart(2);                            // 在单元格中单击两次启动编辑器
        column.setCellEditor(editor);                              // 为列设置编辑器
    }
}
```

文本框 txf_tablecell 的获得焦点事件过程，用于向表中添加新销售订单数据。文本框 txf_tablecell 获得焦点的事件处理过程代码如下：

代码位置：简易通企业进销存管理系统\src\zzk\zhuoyue\xiaoshouchuku\XiaoshouQiandanFrame.java

```
private void txf_tablecellFocusGained(java.awt.event.FocusEvent evt) {
    pzs++;                                                         // 存放品种数的成员变量加 1
    txf_pzs.setText(String.valueOf(pzs));
    String[] rowData=new String[13];
    int row=tb_xuanzejilu.getSelectedRow();
    if (v==null){
        v=new Vector();                                            // 用于存储商品编号的成员变量
    }
    v.add(tb_xuanzejilu.getModel().getValueAt(row, 0).toString());  // 在向量中存储商品编号
    for (int c=0;c<11;c++){
        if (c>=6){
            rowData[c+2]=tb_xuanzejilu.getModel().getValueAt(row, c).toString();
        }else{
            rowData[c]=tb_xuanzejilu.getModel().getValueAt(row, c).toString();
        }
    }
    tableModel.addRow(rowData);
```

```
        jScrollPane2.setVisible(false);
        jScrollPane1.setVisible(true);
        btn_newjilu.setVisible(true);
        this.setSize(700,406);
        TableCellEditor cellEditor = tb_xuanzejilu.getCellEditor();
        if (cellEditor != null){
            cellEditor.stopCellEditing();                                    // 停止表格中单元格的编辑
        }
        int currentRow=0;
        if (tb_newjilu.getRowCount()>0){
            currentRow=tb_newjilu.getRowCount()-1;
            tb_newjilu.setFocusable(false);
            tb_newjilu.setFocusable(true);
            tb_newjilu.editCellAt(currentRow,6);
            ((DefaultCellEditor)(tb_newjilu.getCellEditor(currentRow,0))).getComponent().requestFocus();
        }
    }
```

“销售订单管理”窗体中“保存销售单”按钮的事件过程，可以把表中添加的所有订单数据保存到数据库中。“保存销售单”按钮的事件处理过程代码如下：

代码位置：简易通企业进销存管理系统\src\zzk\zhuoyue\xiaoshouchuku\XiaoshouQiandanFrame.java

```
    private void btn_baocunActionPerformed(java.awt.event.ActionEvent evt) {
        int totalRows=tb_newjilu.getRowCount();
        if (totalRows>0){
            Object objdj=tb_newjilu.getModel().getValueAt(totalRows-1,6);
            Object objsl=tb_newjilu.getModel().getValueAt(totalRows-1,7);
            if (objdj==null || objsl==null || objdj.toString().length()<=0 || objsl.toString().length()<=0){
                JOptionPane.showMessageDialog(this,
"单价和数量必须输入。\r\n 输入完成后按回车键确认。");
                return;
            }
        }else{
            JOptionPane.showMessageDialog(this,"没有可以保存的数据，请录入数据后再保存。");
            return;
        }
        String xsqdph=txf_xsddid.getText();                                  // 销售订单票号
        int pzsl=Integer.parseInt(txf_pzs.getText());                        // 品种数
        float je=Float.parseFloat(txf_hjje.getText());                       // 合计金额
        String ysjl=txf_ysjl.getText().trim();                               // 验收结论
        String kh=cmb_kh.getSelectedItem().toString().trim();                // 客户全称
        String dgrq=txf_xsqdrq.getText();                                    // 订购日期
        String czy=txf_czy.getText();                                        // 操作员
        String jsr=cmb_jsr.getSelectedItem().toString().trim();              // 经手人
        String jsfs=cmb_jsfs.getSelectedItem().toString();                   // 结款方式
        // 下面是保存数据的代码
        try{
            con=daoConn.getConn();
            conn=daoConn.getConn();
            con.setAutoCommit(false);                                        // 取消自动提交
```

```
            conn.setAutoCommit(false);
            String sqlmain="insert into tb_xsqd_main values(?,?,?,?,?,?,?,?,?)";
            ps=con.prepareStatement(sqlmain);
            ps.setString(1,xsqdph);                                    // 传递参数
            ps.setInt(2,pzsl);
            ps.setFloat(3,je);
            ps.setString(4,ysjl);
            ps.setString(5,kh);
            ps.setString(6,dgrq);
            ps.setString(7,czy);
            ps.setString(8,jsr);
            ps.setString(9,jsfs);
            ps.executeUpdate();                                        // 执行 SQL 语句
            String sqldetail="insert into tb_xsqd_detail values(?,?,?,?,?)";
            pst=conn.prepareStatement(sqldetail);
            for (int row=0;row<totalRows;row++){
                pst.setString(1,xsqdph);
                pst.setString(2,(String)tb_newjilu.getModel().getValueAt(row,0));
                pst.setString(3,(String)tb_newjilu.getModel().getValueAt(row,1));
                pst.setFloat(4,Float.parseFloat((String)tb_newjilu.getModel().getValueAt(row,6)));
                pst.setInt(5,Integer.parseInt((String)tb_newjilu.getModel().getValueAt(row,7)));
                pst.executeUpdate();
                pst.addBatch();
            }
            con.commit();                                              // 提交数据，并将其保存到数据库
            conn.commit();
            JOptionPane.showMessageDialog(this,"保存成功！！！");
            pzs=0;
            txf_hjje.setText("0");
            v.removeAllElements();                                     // 移除向量中的所有元素
            txf_ysjl.setText(null);
            txf_pzs.setText("0");
            cmb_jsr.setSelectedItem("请选择经手人");
            cmb_kh.setSelectedItem("请选择客户");
            init();
            showData();                                                // 调用方法初始化表中数据
        }catch(Exception ex){
            try{
                con.rollback();                                        // 回滚数据，取消保存
                conn.rollback();
            }catch(Exception e){
            }
            ex.printStackTrace();
        }
    }
```

注意：由于篇幅有限，这里只给出了主要组件的事件处理过程和部分所需要的方法，其他方法和组件的事件处理过程没有给出，可以参看光盘中的源程序。

22.19　销售出库管理模块设计

本模块使用的数据表和视图：tb_xsqd_main、tb_xsqd_detail、tb_sell_main、tb_sell_detail、tb_spinfo、tb_kucun、tb_jsr 和 v_xsdd

“销售出库管理”模块负责商品的销售操作，客户必须先进行销售签单，有销售订单的商品才能出库，否则不能出库。“销售出库管理”窗体界面如图 22.32 所示。

图 22.32　“销售出库管理”窗体界面

22.19.1　设计销售出库管理窗体

在项目“简易通企业进销存管理系统”的包 zzk.zhuoyue.xiaoshouchuku 中建立一个名称为 XiaoshouChukuFrame 的 JFrame 窗体，设置窗体布局为“边框式”布局，并在“边框式”布局的窗体中心放一个 null 布局的面板，在面板上放置 8 个标签、8 个文本框、4 个按钮、一个组合框、4 个滚动面板、4 个表和一个选项卡面板。该窗体中所用到的关键组件如表 22.21 所示。

表 22.21　销售出库管理窗体用到的关键组件

控 件 类 型	控件 ID	主要属性设置	用　途
JTextField	txf_xsid	设置 editable 属性为 false	销售单号
	txf_xsrq	设置 editable 属性为 false	销售日期
	txf_xsddid	无	销售订单票号
	txf_xsczy	设置 editable 属性为 false	操作员
	txf_kh	设置 editable 属性为 false	客户全称
	txf_fkfs	设置 editable 属性为 false	付款方式
	txf_fkje	设置 editable 属性为 false	应收款金额
	txf_tbxsqdid	无	随表格移动的文本框
JComboBox	cmb_jsr	无	显示所有启用的经手人
JButton	btn_chongxinluru	设置 text 属性为“重新录入”	重新输入数据
	btn_baocun	设置 text 属性为“保存”	保存入库信息
	btn_quanbuxiaoshoudan	设置 text 属性为“※”	显示全部销售订单商品信息
	btn_fanhui	设置 text 属性为“返回”	返回主界面
JLabel	lbl_zkyss	设置 text 属性为“<<<”	展开或收缩窗体

续表

控 件 类 型	控件 ID	主要属性设置	用　　途
JTable	tb_main	设置 autoResizeMode 属性为 Off	显示商品销售单列表
	tb_detail	设置 autoResizeMode 属性为 Off	显示商品销售单明细表
	tb_showxiaoshoudan	设置 autoResizeMode 属性为 Off	显示商品销售订单列表
	tb_view	设置 autoResizeMode 属性为 Off	显示商品销售信息

22.19.2 销售出库管理

在“销售出库管理”窗体界面中应先选择经手人，然后单击“销售签单票号”右侧的按钮，将显示没有出库的销售订单列表，已经销售的订单将不再显示，这样可以避免重复输入同一个订单。单击选择要销售的订单信息所在的行，将隐藏订单列表并把与选择行销售签单票号相同的所有订单信息添加到对应的表中。单击“<<<”标签展开窗体，查看本次输入的订单详细信息，同时标签上显示的内容变为“>>>”，单击“>>>”标签，又返回原来的状态。单击“保存”按钮，可以保存本次销售信息。单击“重新录入”按钮，重新输入数据。单击“返回”按钮，退出“采购入库管理”窗体，返回应用程序主窗体。

声明一个通用的在表中显示数据的方法，返回值是表中要显示的内容，该方法中有 4 个参数，第一个参数是数据库中的数据表名或视图名，第二个参数是数据表或视图中的字段列表，第 3 个参数是数据表或视图中作为条件的字段，最后一个参数是与条件字段对应的条件值。该方法的程序代码如下：

代码位置：简易通企业进销存管理系统\src\zzk\zhuoyue\xiaoshouchuku\XiaoshouChukuFrame.java

```
public String[][] showDataToTable(String table,String[] fieldList,String conditionField,String condition){
    try{
        conn=daoConn.getConn();
        st=conn.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
ResultSet.CONCUR_READ_ONLY);
        String sql="select ";                    // 定义 SQL 语句并用循环与传递的实参构成新的 SQL 语句
        for (int i=0;i<fieldList.length;i++){
            if (i<fieldList.length-1){
                sql=sql+fieldList[i]+",";
            }else{
                sql=sql+fieldList[i]+" from "+table;
            }
        }
        if (!conditionField.trim().equals("") && conditionField.indexOf("in")>0){
            sql=sql+" where "+conditionField;                          // 形成带条件的 SQL 语句
        }else if (!conditionField.trim().equals("")){
            sql=sql+" where "+conditionField+"='"+condition+"'";       // 形成带条件的 SQL 语句
        }
        rs=st.executeQuery(sql);
        md=rs.getMetaData();                                           // 获得 ResultSetMetaData 对象
        String[][] returnData=null;
        if (rs.last()){
            returnData=new String[rs.getRow()][md.getColumnCount()]; // 创建数组
        }else{
            returnData=new String[0][md.getColumnCount()];             // 创建数组
        }
```

```
            rs.beforeFirst();
            int currentR=0;                                          // 获得表中当前操作的行号
            // 利用循环动态为数组赋值
            while (rs.next()){
                for (int column=0;column<md.getColumnCount();column++){
                    if (md.getColumnTypeName(column+1).trim().equals("numeric")){
                        returnData[currentR][column]=String.valueOf(rs.getFloat(column+1));
                    }else if (md.getColumnTypeName(column+1).trim().equals("int")){
                        returnData[currentR][column]=String.valueOf(rs.getInt(column+1));
                    }else{
                        returnData[currentR][column]=rs.getString(column+1);
                    }
                }
                currentR++;
            }
            return returnData;
        }catch(Exception ex){
            ex.printStackTrace();
            return null;
        }
    }
```

声明为表设置表格模型的方法，该方法有 3 个参数，第一个参数是要显示数据的表名，第二个参数是表的列标题数组，第 3 个参数是表中显示的值数组。该方法的程序代码如下：

代码位置：简易通企业进销存管理系统\src\zzk\zhuoyue\xiaoshouchuku\XiaoshouChukuFrame.java

```
public void showSpInfoToTable(JTable table,String[] columnNames,String[][] tableValues) {
    tableModel = new DefaultTableModel(tableValues, columnNames);        // 创建表格模型对象
    if (tableModel.getRowCount()<=0){
        jScrollPane3.setVisible(false);
        txf_xsddid.requestFocus();                                       // 使文本框获得焦点
        return;
    }
    table.setModel(tableModel);                                          // 给表设置表格模型
}
```

“销售出库管理”窗体中的“保存”按钮，用于保存商品的销售信息，用户选择完销售订单后，单击“保存”按钮，可以保存销售信息。“保存”按钮的事件处理过程代码如下：

代码位置：简易通企业进销存管理系统\src\zzk\zhuoyue\xiaoshouchuku\XiaoshouChukuFrame.java

```
private void btn_baocunActionPerformed(java.awt.event.ActionEvent evt) {
    if (tb_main.getRowCount()<=0){
        JOptionPane.showMessageDialog(this,"没有要保存的数据记录。");
        return;
    }
    try{
        con=daoConn.getConn();                                           // 建立数据库连接
        conn=daoConn.getConn();
        connec=daoConn.getConn();
        connect=daoConn.getConn();
        con.setAutoCommit(false);                                        // 取消数据库的自动提交
        conn.setAutoCommit(false);
```

```
        connec.setAutoCommit(false);
        connect.setAutoCommit(false);
        // 保存销售明细信息
        String sqldetail="insert into tb_sell_detail values(?,?,?,?,?,?)";
        ps=conn.prepareStatement(sqldetail);
        int tb4TotalRows=tb_detail.getModel().getRowCount();
        int tb4TotalColumns=tb_detail.getModel().getColumnCount();
        String[] spbh=new String[tb4TotalRows];                    // 声明保存销售明细表中商品编号的数组
        float[] spdj=new float[tb4TotalRows];                      // 声明保存销售明细表中商品单价的数组
        int[] spsl=new int[tb4TotalRows];                          // 声明保存销售明细表中商品数量的数组
        // 用循环为 SQL 语句的参数赋值
        for (int r=0;r<tb4TotalRows;r++){
            for (int c=0;c<tb4TotalColumns;c++){
                if (c==4){
                    ps.setInt(c+1,Integer.parseInt(tb_detail.getModel().getValueAt(r,c).toString()));
                }else if (c==3){
                    ps.setFloat(c+1,Float.parseFloat(tb_detail.getModel().getValueAt(r,c).toString()));
            }else{
                ps.setString(c+1,tb_detail.getModel().getValueAt(r,c).toString());
            }
        }
        spbh[r]=tb_detail.getModel().getValueAt(r,1).toString();                    // 为编号数组赋值
        spdj[r]=Float.parseFloat(tb_detail.getModel().getValueAt(r,3).toString());  // 为单价数组赋值
        spsl[r]=Integer.parseInt(tb_detail.getModel().getValueAt(r,4).toString());  // 为数量数组赋值
        ps.executeUpdate();
        ps.addBatch();
    }
    // 保存销售列表
    String sqlmain="insert into tb_sell_main values(?,?,?,?,?,?,?,?,?,?,?,?,?)";
    ps=con.prepareStatement(sqlmain);
    int tb1TotalRows=tb_main.getModel().getRowCount();                  // 获得表格模型中数据的总行数
    int tb1TotalColumns=tb_main.getModel().getColumnCount();            // 获得表格模型中数据的总列数
    for (int r=0;r<tb1TotalRows;r++){
        for (int c=0;c<tb1TotalColumns;c++){
            if (c==1){
                ps.setInt(c+1,Integer.parseInt(tb_main.getModel().getValueAt(r,c).toString()));
                }else if (c==2){
                    ps.setFloat(c+1,Float.parseFloat(tb_main.getModel().getValueAt(r,c).toString()));
                }else{
                    ps.setString(c+1,tb_main.getModel().getValueAt(r,c).toString());
                }
            }
            ps.executeUpdate();                                         // 执行 SQL 语句
            ps.addBatch();                                              // 批量添加
        }
        // 实现销售和库存同步操作
        st=connec.createStatement();
        smt=connect.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
ResultSet.CONCUR_UPDATABLE);
```

```
            String[] sqlspinfo=new String[spbh.length];
            String[] sqlkcinfo=new String[spbh.length];
            String spsql="select id,spmc,jc,cd,gg,bz,dw from tb_spinfo where id='";
            String kcsql="select spmc,dj,sl from tb_kucun where id='";
            for (int i=0;i<spbh.length;i++){
                sqlspinfo[i]=spsql+spbh[i]+"'";                    // 在数组中存放生成的商品信息 SQL 语句
                sqlkcinfo[i]=kcsql+spbh[i]+"'";                    // 在数组中存放生成的库存信息 SQL 语句
                rs=st.executeQuery(sqlspinfo[i]);
                rst=smt.executeQuery(sqlkcinfo[i]);
                if (rst.next()){
                    // 如果库存表中有匹配的记录，则修改库存表中的内容
                    if (rs.next()){
                        int sl=rst.getInt("sl");                   // 获得库存表中的数量
                        int zsl=sl-spsl[i];                        // 更改库存的新数量
                        // 更新库存量
                        smt.executeUpdate("update tb_kucun set sl="+zsl+" where id='"+spbh[i]+"'");
                    }
                }else{
                    if (rs.next()){
                        JOptionPane.showMessageDialog(this,
"库存中没有"+rs.getString("spmc")+"这种商品信息，请核实后重新输入。");
                    }
                    con.rollback();                                // 回滚数据，取消保存
                    conn.rollback();
                    connec.rollback();
                    connect.rollback();
                    return;
                }
            }
            conn.commit();                                         // 提交数据，并将其保存到数据库
            con.commit();
            connec.commit();
            connect.commit();
            btn_baocun.setEnabled(false);
            JOptionPane.showMessageDialog(this,"保存成功！！！");
        }catch(Exception ex){
            try{
                JOptionPane.showMessageDialog(this,"保存失败！！！");
                con.rollback();                                    // 回滚数据，取消保存
                conn.rollback();
                connec.rollback();
                connect.rollback();
            }catch(Exception e){
            }
        }
    }
```

“销售出库管理”窗体中“<<<”标签的事件处理过程，用于展开和收缩窗体，其程序代码如下：

代码位置：简易通企业进销存管理系统\src\zzk\zhuoyue\xiaoshouchuku\XiaoshouChukuFrame.java

```
private void jLabel6MouseClicked(java.awt.event.MouseEvent evt) {
```

```
    if (jLabel6.getText().equals("<<<")){
        jLabel6.setText(">>>");                          // 改变标签的标题为“>>>”
        jScrollPane2.setLocation(30,330);                // 设置滚动面板的位置
        jScrollPane2.setVisible(true);                   // 显示滚动面板
        this.setSize(755,550);                           // 窗体的高度变大，展开窗体
    }else{
        jLabel6.setText("<<<");                          // 改变标签标题为“<<<”
        jScrollPane2.setLocation(30,330);
        jScrollPane2.setVisible(false);                  // 隐藏滚动面板
        this.setSize(755,360);                           // 窗体的高度变小，收缩窗体
    }
}
```

“销售出库管理”窗体中销售签单票号右侧的※按钮，用于显示销售订单列表，单击该按钮将在表中显示销售订单列表，双击销售订单列表中需要销售的订单，系统会自动为组件添加销售信息。※按钮的事件处理过程代码如下：

代码位置：简易通企业进销存管理系统\src\zzk\zhuoyue\xiaoshouchuku\XiaoshouChukuFrame.java

```
private void btn_quanbuxiaoshoudanActionPerformed(java.awt.event.ActionEvent evt) {
    if (cmb_jsr.getSelectedItem().toString().equals("请选择经手人")){
        JOptionPane.showMessageDialog(this,"请先选择经手人。");
        return;
    }
    txf_xsddid.requestFocus();
    String[] fieldList={"xsqdid","spid","spmc","gg","dj","sl","je","khqc","qdrq","czy","jsr","jsfs"};
    String[][] tableValues=showDataToTable("v_xsdd",fieldList,
            "xsqdid not in(select xsqdid from tb_sell_main)","");          // 只显示没有入库的订单信息
    String[] columnNames={"销售签单票号","商品编号","商品名称","规格","单价",
    "数量","金额","客户全称","订购日期","操作员","经手人","结款方式"};
    showSpInfoToTable(tb_showxiaoshoudan,columnNames,tableValues);
    if (tb_showxiaoshoudan.getRowCount()>0){
        TableColumn column=tb_showxiaoshoudan.getColumn("销售签单票号");
        column.setPreferredWidth(120) ;
        jScrollPane3.setBounds(30,40,690,210);
        jScrollPane3.setVisible(true);
        txf_xsddid.requestFocus();
    }else{
        JOptionPane.showMessageDialog(this,"必须先签销售单然后才能进行销售。");
    }
}
```

“销售出库管理”窗体中的销售订单表格，用于显示销售订单列表，双击销售订单表中需要销售的订单，系统会自动为文本框添加销售信息，并使文本框 txf_tbxsqdid 获得焦点。销售订单表格的鼠标单击事件处理过程代码如下：

代码位置：简易通企业进销存管理系统\src\zzk\zhuoyue\xiaoshouchuku\XiaoshouChukuFrame.java

```
private void tb_showxiaoshoudanMouseClicked(java.awt.event.MouseEvent evt) {
    txf_xsddid.setText(null);
    currentRow=tb_showxiaoshoudan.getSelectedRow();          // 获得表中选择行的行号
    currentColumn=tb_showxiaoshoudan.getSelectedColumn();    // 获得表中选择列的列号
    txtTextFieldPosition(currentRow,currentColumn);          // 设计文本框的显示位置
    txf_tbxsqdid.setText((String)tb_showxiaoshoudan.getValueAt(currentRow,currentColumn));
```

```
    txf_xsddid.setText((String)tb_showxiaoshoudan.getValueAt(currentRow,0));
    txf_kh.setText((String)tb_showxiaoshoudan.getValueAt(currentRow,7));
    txf_fkfs.setText((String)tb_showxiaoshoudan.getValueAt(currentRow,11));
    txf_tbxsqdid.setVisible(true);                          // 使文本框显示
    txf_tbxsqdid.requestFocus();                            // 使文本框获得焦点
}
```

“销售出库管理”窗体中文本框 txf_tbxsqdid 的事件处理过程，用于隐藏销售订单表，在销售单列表和销售单明细表中显示销售信息。文本框 txf_tbxsqdid 的获得焦点的事件处理过程代码如下：

代码位置：简易通企业进销存管理系统\src\zzk\zhuoyue\xiaoshouchuku\XiaoshouChukuFrame.java

```
private void txf_tbxsqdidFocusGained(java.awt.event.FocusEvent evt) {
    String xsddid=txf_xsddid.getText().trim();              // 获得销售签单票号
    showDataToTable(xsddid);                                // 在表中显示指定销售签单票号的数据
    jScrollPane3.setVisible(false);                         // 隐藏滚动面板
    txf_tbxsqdid.setVisible(false);                         // 隐藏文本框
    jTabbedPane1.setSelectedIndex(1);                       // 显示滚动面板中索引为 1 的页
}
```

“销售出库管理”窗体中“重新录入”按钮的事件处理过程，用于初始化组件。“重新录入”按钮的事件处理过程代码如下：

代码位置：简易通企业进销存管理系统\src\zzk\zhuoyue\xiaoshouchuku\XiaoshouChukuFrame.java

```
private void btn_chongxinluruActionPerformed(java.awt.event.ActionEvent evt) {
    chushihua();                                            // 调用方法，初始化操作
    btn_baocun.setEnabled(true);                            // 使“保存”按钮可用
    deleteTableRow(tb_detail);                              // 删除表中的所有数据行
    deleteTableRow(tb_main);                                // 删除表中的所有数据行
    deleteTableRow(tb_view);                                // 删除表中的所有数据行
}
```

注意：由于篇幅有限，这里只给出了主要组件的事件处理过程和部分方法，其他方法和组件的事件处理过程没有给出，可以参看光盘中的源程序。

22.20　销售退货管理模块设计

本模块使用的数据表和视图：tb_xsth_main、tb_xsth_detail、tb_sell_main、tb_sell_detail、tb_jsr、tb_khinfo、tb_spinfo、tb_kucun、v_sell_xsth_sl 和 v_xsth_sl

“销售退货管理”模块负责已经销售商品的退货操作，退货时要先选择经手人、结款方式和客户名称，然后单击“录入数据”按钮，打开“录入商品销售退货信息”窗体，输入退货信息。“销售退货管理”窗体界面如图 22.33 所示。

“录入商品销售退货信息”窗体用于销售退货信息的输入操作，该窗体的 3 个组合框中显示的是“销售退货管理”窗体中所选客户名称的商品销售信息。在任一组合框中选择商品信息后按 Enter 键，其他组件会自动显示相关内容，用户只需选择销售票号并输入单价和数量即可，单击“确定”按钮后，如果输入的数量超出销售的数量会有相应提示，否则直接添加到“销售退货管理”窗体中。“录入商品销售退货信息”窗体如图 22.34 所示。

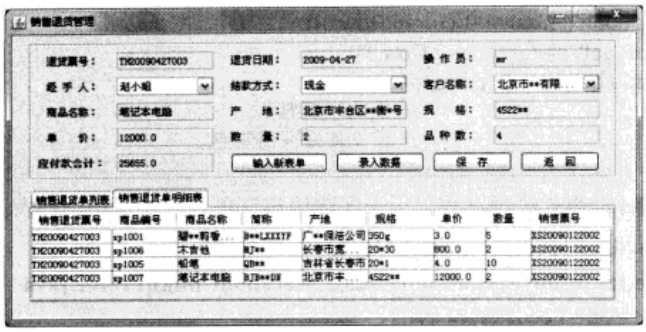


图 22.33 “销售退货管理”窗体界面

图 22.34 “录入商品销售退货信息”窗体

22.20.1 设计销售退货管理窗体

在项目“简易通企业进销存管理系统”的 zzk.zhuoyue.xiaoshouchuku 包中建立一个名称为 XiaoshouTuihuoFrame 的 JFrame 窗体，设置窗体布局为空布局，并在窗体上放一个空布局的面板，在面板上放置 13 个标签、10 个文本框、4 个按钮、3 个组合框，再在窗体上放一个选项卡面板，并在选项卡面板上放两个面板，每个面板上各放一个滚动面板和一个表。该窗体中所用到的关键组件如表 22.22 所示。

表 22.22 销售退货管理窗体用到的关键组件

控 件 类 型	控件 ID	主要属性设置	用 途
JTextField	txf_thph	设置 editable 属性为 false	退货票号
	txf_thrq	设置 editable 属性为 false	退货日期
	txf_spmc	设置 editable 属性为 false	商品名称
	txf_czy	设置 editable 属性为 false	操作员
	txf_chandi	设置 editable 属性为 false	产地
	txf_guige	设置 editable 属性为 false	规格
	txf_danjia	设置 editable 属性为 false	单价
	txf_shuliang	设置 editable 属性为 false	数量
	txf_pzs	设置 editable 属性为 false	品种数
	txf_yfkhj	设置 editable 属性为 false	应付款合计
JComboBox	cmb_khqc	无	显示有销售记录的客户全称
	cmb_jsfs	无	显示结款方式
	cmb_jsr	无	显示所有启用的经手人
JButton	btn_shuruxinbiaodan	设置 text 属性为“输入新表单”	重新输入新的表单
	btn_lurushuju	设置 text 属性为“录入数据”	打开“录入商品销售退货信息”窗体
	btn_baocun	设置 text 属性为“保存”	保存商品销售退货信息
	btn_fanhui	设置 text 属性为“返回”	返回主界面
JTable	tb_main	设置 autoResizeMode 属性为 Off	显示商品销售退货单列表
	tb_detail	设置 autoResizeMode 属性为 Off	显示商品销售退货单明细表

22.20.2 设计录入商品销售退货信息窗体

在项目“简易通企业进销存管理系统”的 zzk.zhuoyue.xiaoshouchuku 包中建立一个名称为 InputXiaoshou

TuihuoFrame 的 JFrame 窗体，设置窗体布局为空布局，并在窗体上放两个空布局的面板，在第一个面板上放置 8 个标签、4 个文本框和 4 个组合框，在第二个面板上放置两个按钮。该窗体中所用到的关键组件如表 22.23 所示。

表 22.23　录入商品销售退货信息窗体用到的关键组件

控 件 类 型	控件 ID	主要属性设置	用　途
JTextField	txf_gg	设置 editable 属性为 false	规格
	txf_cd	设置 editable 属性为 false	产地
	txf_dj	无	单价
	txf_sl	无	数量
JComboBox	cmb_spbh	无	显示商品编号
	cmb_spmc	无	显示商品名称
	cmb_xsid	无	显示所选客户的销售票号
	cmb_jc	无	显示商品的简称
JButton	btn_ok	设置 text 属性为“确定”	确认本次输入
	btn_cancel	设置 text 属性为“取消”	取消本次输入

22.20.3　设计保存输入信息的类

在项目“简易通企业进销存管理系统”的 zzk.zhuoyue.xiaoshouchuku 包中建立一个名称为 SaveXiaoshouTuihuoInfo 的类，用于保存“录入商品销售退货信息”窗体中输入的数据信息，并在“销售退货管理”窗体中使用。该类的主要程序代码如下：

代码位置：简易通企业进销存管理系统\src\zzk\zhuoyue\xiaoshouchuku\SaveXiaoshouTuihuoInfo.java

```
package zzk.zhuoyue.xiaoshouchuku;
public class SaveXiaoshouTuihuoInfo {
    private String spbh;                                    // 定义商品编号
    private String spmc;                                    // 定义商品名称
    private String jc;                                      // 定义简称
    private String cd;                                      // 定义产地
    private String gg;                                      // 定义规格
    private float dj;                                       // 定义单价
    private int sl;                                         // 定义数量
    private String xsid;                                    // 定义销售票号
    // 保存按钮状态，如果单击“确定”按钮，则把 button 的值设置为 true，否则设置为 false
    private boolean button;
    // 这里省略了 is、set 和 get 方法
}
```

“录入商品销售退货信息”窗体中的“确定”按钮，用于判断用户输入的退货信息是否齐全以及是否超出销售的数量，销毁本窗体并把输入的退货信息传递到“销售退货管理”窗体中。“确定”按钮的事件处理过程代码如下：

代码位置：简易通企业进销存管理系统\src\zzk\zhuoyue\xiaoshouchuku\InputXiaoshouTuihuoFrame.java

```
private void btn_okActionPerformed(java.awt.event.ActionEvent evt) {
    if (txf_gg.getText()==null || txf_gg.getText().length()<=0){
        JOptionPane.showMessageDialog(this,"请先选择商品编号或商品名称或简称并按回车键。");
        cmb_spbh.requestFocus();
```

```
            return;
        }
        if (cmb_xsid.getSelectedItem().toString().equals("请选择销售票号")){
            JOptionPane.showMessageDialog(this,"请选择销售票号！");
            return;
        }
        if (txf_dj.getText().length()<=0){
            JOptionPane.showMessageDialog(this,"请输入单价。");
            txf_dj.requestFocus();
            return;
        }
        if (txf_sl.getText().length()<=0){
            JOptionPane.showMessageDialog(this,"请输入数量。");
            txf_sl.requestFocus();
            return;
        }
        float srdj=0f;                                          // 存放输入的单价
        int srsl=0;                                             // 存放输入的数量
        try{
            srdj=Float.parseFloat(txf_dj.getText());            // 获得输入的单价值
            srsl=Integer.parseInt(txf_sl.getText());            // 获得输入的数量值
        }catch(Exception e){
            JOptionPane.showMessageDialog(this,"请在单价和数量单元格中输入数字。");
            return;
        }
        if (srdj<=0){
            JOptionPane.showMessageDialog(this,"请输入退货单价值。");
            return;
    }
        if (srsl<=0){
            JOptionPane.showMessageDialog(this,"请输入退货数量值。");
            return;
    }
        if (getTotalQuantity()<=0){
            JOptionPane.showMessageDialog(this,"您所购买的该销售票号的商品已经全部退完，不能再退。");
            return;
        }
        if (srsl-getTotalQuantity()>0){
            JOptionPane.showMessageDialog(this,
"您的退货数量已经超出购买的数量。\r\n 您的最大退货量是"+getTotalQuantity()+"。");
            return;
        }
        Vector v=XiaoshouTuihuoFrame.getSpid();                 // 获得商品编号的向量
        // 传递输入的信息到销售退货管理窗体
        XiaoshouTuihuoFrame.setXiaoshouTuihuo(saveXiaoshouInfo(true));
        if (v==null){
            v=new Vector();
        }
        v.add(cmb_spbh.getSelectedItem().toString());           // 在向量中添加本次输入的商品编号
        XiaoshouTuihuoFrame.setSpid(v);                         // 将向量传递到销售退货管理窗体
```

```
    Vector vec=XiaoshouTuihuoFrame.getSpje();                    // 获得商品金额的向量
    if (vec==null){
        vec=new Vector();
    }
    float dj=Float.parseFloat(txf_dj.getText().trim());          // 获得单价值
    int sl=Integer.parseInt(txf_sl.getText().trim());            // 获得数量值
    vec.add(dj*sl);                                              // 在向量中添加金额
    XiaoshouTuihuoFrame.setSpje(vec);                            // 将向量传递到销售退货管理窗体
    this.dispose();                                              // 销毁窗体
}
```

“销售退货管理”窗体处于激活状态时的事件处理过程，用于在表中显示输入的数据，由于在“录入商品销售退货信息”窗体中单击“确定”按钮将销毁窗体，此时“销售退货管理”窗体处于激活状态，因此可以响应窗体激活事件。“销售退货管理”窗体处于激活状态时的事件处理过程代码如下：

代码位置：简易通企业进销存管理系统\src\zzk\zhuoyue\xiaoshouchuku\XiaoshouTuihuoFrame.java

```
private void formWindowActivated(java.awt.event.WindowEvent evt) {
    if (XiaoshouTuihuoFrame.getXiaoshouTuihuo()!=null &&
XiaoshouTuihuoFrame.getXiaoshouTuihuo().isButton()){
        int pzs=0;                                               // 定义品种数
        if (getSpid().size()>0){
            pzs=getSpid().size();                                // 通过向量获得品种数
            txf_pzs.setText(String.valueOf(pzs));                // 在文本框中显示品种数
        }
        float hjje=0f;                                           // 定义合计金额
        if (getSpje().size()>0){
            for (int i=0;i<getSpje().size();i++){
                hjje=hjje+Float.parseFloat(getSpje().get(i).toString());   // 计算合计金额
            }
        }
        txf_yfkhj.setText(String.valueOf(hjje));
        // 获得组件的内容
        String xsthph=txf_thph.getText().trim();
        String spbh=XiaoshouTuihuoFrame.getXiaoshouTuihuo().getSpbh();
        String spmc=XiaoshouTuihuoFrame.getXiaoshouTuihuo().getSpmc();
        String jc=XiaoshouTuihuoFrame.getXiaoshouTuihuo().getJc();
        String cd=XiaoshouTuihuoFrame.getXiaoshouTuihuo().getCd();
        String gg=XiaoshouTuihuoFrame.getXiaoshouTuihuo().getGg();
        String dj=String.valueOf(XiaoshouTuihuoFrame.getXiaoshouTuihuo().getDj());
        String sl=String.valueOf(XiaoshouTuihuoFrame.getXiaoshouTuihuo().getSl());
        txf_spmc.setText(spmc);                                  // 为文本框赋值
        txf_chandi.setText(cd);
        txf_guige.setText(gg);
        txf_danjia.setText(dj);
        txf_shuliang.setText(sl);
        String khqc=cmb_khqc.getSelectedItem().toString();
        String thrq=txf_thrq.getText();
        String czy=txf_czy.getText();
        String jsr=cmb_jsr.getSelectedItem().toString();
        String jsfs=cmb_jsfs.getSelectedItem().toString();
        String xsid=XiaoshouTuihuoFrame.getXiaoshouTuihuo().getXsid();
```

```
            // 在表中添加行
            if(rowCount==1){
                // 获取数据的数组
                String[] valueMain={xsthph,String.valueOf(pzs),String.valueOf(hjje),khqc,thrq,czy,jsr,jsfs};
                tableModelMain.addRow(valueMain);                    // 在表中添加数据行
            }else{
                tableModelMain.setValueAt(String.valueOf(pzs), 0, 1);        // 修改表中的品种数
                tableModelMain.setValueAt(String.valueOf(hjje), 0, 2);       // 修改表中的合计金额
            }
            // 获得退货明细的数组
            String[] valueDetail={xsthph,spbh,spmc,jc,cd,gg,String.valueOf(dj),String.valueOf(sl),xsid};
            tableModelDetail.addRow(valueDetail);                    // 在表中添加数据行
            XiaoshouTuihuoFrame.getXiaoshouTuihuo().setButton(false);
            rowCount++;
        }
    }
```

“销售退货管理”窗体中的“保存”按钮可以保存销售退货信息列表和明细表，同时更新库存，增加库存量。“保存”按钮的事件处理过程代码如下：

代码位置：简易通企业进销存管理系统\src\zzk\zhuoyue\xiaoshouchuku\XiaoshouTuihuoFrame.java

```
private void btn_baocunActionPerformed(java.awt.event.ActionEvent evt) {
    if (tb_main.getRowCount()<=0){
        JOptionPane.showMessageDialog(this,
"没有要保存的数据记录。\r\n 请单击“录入数据”按钮输入数据。");
        return;
    }
    try{
        con=daoConn.getConn();                          // 获得数据库连接
        conn=daoConn.getConn();
        connect=daoConn.getConn();
        con.setAutoCommit(false);                       // 取消数据的自动提交
        conn.setAutoCommit(false);
        connect.setAutoCommit(false);
        // 保存销售退货明细信息
        String sqldetail="insert into tb_xsth_detail values(?,?,?,?,?,?,?,?,?)";
        ps=conn.prepareStatement(sqldetail);
        // 获得表模型的总行数
        int tbDetailTotalRows=tableModelDetail.getRowCount();
        // 获得表模型的总列数
        int tbDetailTotalColumns=tableModelDetail.getColumnCount();
        String[] spbh=new String[tbDetailTotalRows];     // 声明保存销售退货明细表中商品编号的数组
        float[] spdj=new float[tbDetailTotalRows];       // 声明保存销售退货明细表中商品单价的数组
        int[] spsl=new int[tbDetailTotalRows];           // 声明保存销售退货明细表中商品数量的数组
        // 为 SQL 语句中的参数赋值
        for (int r=0;r<tbDetailTotalRows;r++){
            for (int c=0;c<tbDetailTotalColumns;c++){
                if (c==7){
                    ps.setInt(c+1,Integer.parseInt(tableModelDetail.getValueAt(r,c).toString()));
                }else if (c==6){
                    ps.setFloat(c+1,Float.parseFloat(tableModelDetail.getValueAt(r,c).toString()));
```

```
                }else{
                    ps.setString(c+1,tableModelDetail.getValueAt(r,c).toString());
                }
            }
            // 把退货商品的编号赋值给数组
            spbh[r]=tableModelDetail.getValueAt(r,1).toString();
            // 把退货商品的单价赋值给数组
            spdj[r]=Float.parseFloat(tableModelDetail.getValueAt(r,6).toString());
            // 把退货商品的数量赋值给数组
            spsl[r]=Integer.parseInt(tableModelDetail.getValueAt(r,7).toString());
            ps.executeUpdate();                                    // 执行 SQL 语句
            ps.addBatch();                                         // 批量添加
        }
        // 保存销售退货列表
        String sqlmain="insert into tb_xsth_main values(?,?,?,?,?,?,?,?)";
        ps=con.prepareStatement(sqlmain);
        // 获得表模型的总列数
        int tbMainTotalColumns=tableModelMain.getColumnCount();
        // 为 SQL 语句的参数赋值
        for (int c=0;c<tbMainTotalColumns;c++){
            if (c==1){
                ps.setInt(c+1,Integer.parseInt(tableModelMain.getValueAt(0,c).toString()));
            }else if (c==2){
                ps.setFloat(c+1,Float.parseFloat(tableModelMain.getValueAt(0,c).toString()));
            }else{
                ps.setString(c+1,tableModelMain.getValueAt(0,c).toString());
            }
        }
        ps.executeUpdate();                                        // 执行 SQL 语句
        ps.addBatch();                                             // 批量添加
        // 实现销售退货和库存同步操作
        smt=connect.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
ResultSet.CONCUR_UPDATABLE);
        String[] sqlkcinfo=new String[spbh.length];                // 存放 SQL 语句的数组
        String kcsql="select dj,sl from tb_kucun where id='";
        // 对每种退货商品更新库存量
        for (int i=0;i<spbh.length;i++){
            sqlkcinfo[i]=kcsql+spbh[i]+"'";                        // 为数组赋值 SQL 语句
            rst=smt.executeQuery(sqlkcinfo[i]);
            if (rst.next()){
                int sl=rst.getInt("sl");                           // 库存表中的原有数量
                int zsl=sl+spsl[i];                                // 用于修改库存的新数量
                smt.executeUpdate("update tb_kucun set sl="+zsl+" where id='"+spbh[i]+"'");
            }
        }
        conn.commit();                                             // 提交数据，更新数据库
        con.commit();
        connect.commit();
        JOptionPane.showMessageDialog(this,"保存成功！！！");
    }catch(Exception ex){
```

```
            try{
                JOptionPane.showMessageDialog(this,"保存失败！！！");
                con.rollback();                                    // 回滚数据，取消数据库更新操作
                conn.rollback();
                connect.rollback();
            }catch(Exception e){
            }
        }
        init();                                                    // 调用方法
        rowCount=1;                                                // 初始化成员变量
        if (XiaoshouTuihuoFrame.getXiaoshouTuihuo()!=null){
            XiaoshouTuihuoFrame.getXiaoshouTuihuo().setButton(false);
        }
        …这里省略了非关键代码
    }
```

注意：由于篇幅有限，这里只给出了主要组件的事件处理过程，其他方法和组件的事件处理过程没有给出，可以参看光盘中的源程序。

22.21 添加用户信息模块设计

本模块使用的数据表：tb_user

“添加用户信息”模块用于添加新用户，当登录用户是系统管理员或管理员时，可以添加用户；当登录用户是一般用户时，不能打开“添加用户信息”窗体。“添加用户信息”窗体界面如图 22.35 所示。

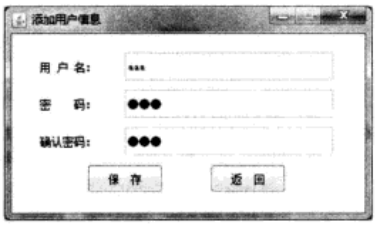


图 22.35 “添加用户信息”窗体界面

22.21.1 设计添加用户信息窗体

在项目“简易通企业进销存管理系统”中建立一个名为 zzk.zhuoyue.userframe 的包，在该包中建立一个名称为 AddUserInfoFrame 的 JFrame 窗体，设置窗体布局为空布局，在窗体上放置 3 个标签、一个文本框、两个密码框和两个按钮。该窗体中所用到的关键组件如表 22.24 所示。

表 22.24 添加用户信息窗体用到的关键组件

控 件 类 型	控件 ID	主要属性设置	用 途
JTextField	txf_username	无	用户名称
JPasswordField	pwd_password	无	密码
	pwd_okpassword	无	确认密码

续表

控 件 类 型	控件 ID	主要属性设置	用　途
JButton	btn_baocun	设置 text 属性为“保存”	保存用户
	btn_fanhui	设置 text 属性为“返回”	返回主窗体

22.21.2　添加用户信息

在“添加用户信息”窗体中，当用户以系统管理员或管理员的身份进入系统时，可以添加用户，输入用户名、密码和确认密码，单击“保存”按钮将添加一个新的一般用户。当用户以一般用户的身份登录系统时，不能打开“添加用户信息”窗体。“保存”按钮的事件处理过程代码如下：

代码位置：简易通企业进销存管理系统\src\zzk\zhuoyue\userframe\AddUserInfoFrame.java

```
private void btn_baocunActionPerformed(java.awt.event.ActionEvent evt) {
    String username=txf_username.getText().trim();
    String pwd=new String(pwd_password.getPassword());
    String okpwd=new String(pwd_okpassword.getPassword());
    if (!okUserAndPassword(username,pwd,okpwd)){                    // 对用户名和密码进行判断
        return;
    }
    saveUserInfo(username,pwd);                                     // 调用方法，保存用户信息
}
```

调用判断用户和密码的方法可以判断用户名和密码是否为空，以及密码和确认密码是否一致，其程序代码如下：

代码位置：简易通企业进销存管理系统\src\zzk\zhuoyue\userframe\AddUserInfoFrame.java

```
private boolean okUserAndPassword(String user,String pwd,String okpwd){
    if (user==null || user.equals("")){
        JOptionPane.showMessageDialog(this,"用户名不能为空，请输入用户名。");
        txf_username.requestFocus();
        return false;
    }else{
        if (pwd==null || pwd.equals("")){
            JOptionPane.showMessageDialog(this,"密码不能为空，请输入密码。");
            pwd_password.requestFocus();
            return false;
        }else if (okpwd==null || okpwd.equals("")){
            JOptionPane.showMessageDialog(this,"确认密码不能为空，请输入确认密码。");
            pwd_okpassword.requestFocus();
            return false;
        }else{
            if (!pwd.equals(okpwd)){
                JOptionPane.showMessageDialog(this,"两次输入的密码不一致，请重新输入。");
                pwd_password.setText(null);
                pwd_okpassword.setText(null);
                pwd_password.requestFocus();
                return false;
            }else{
                return true;
```

```
            }
        }
    }
}
```

输入用户名、密码和确认密码后，单击“保存”按钮将调用保存用户信息的方法保存新用户信息，其程序代码如下：

代码位置：简易通企业进销存管理系统\src\zzk\zhuoyue\userframe\AddUserInfoFrame.java

```
private void saveUserInfo(String username,String pwd){
    try{
        String sql="insert into tb_user   values(?,?,?)";
        conn=daoConn.getConn();
        ps=conn.prepareStatement(sql);
        ps.setString(1,username);
        ps.setString(2,pwd);
        ps.setString(3,"一般用户");
        ps.executeUpdate();
        JOptionPane.showMessageDialog(this,"新用户添加成功。");
    }catch(Exception ex){
        JOptionPane.showMessageDialog(this,"数据库中已经存在该用户。\t\n 请换个用户名。");
    }
    txf_username.setText(null);
    pwd_password.setText(null);
    pwd_okpassword.setText(null);
    txf_username.requestFocus();
}
```

注意：由于篇幅有限，这里只给出了主要组件的事件处理过程和方法，其他组件的事件处理过程没有给出，可以参看光盘中的源程序。

22.22 用户设置窗体模块设计

本模块使用的数据表：tb_user

“用户设置窗体”模块用于设置权限、删除用户、查看用户和修改登录用户密码等操作，当登录用户是一般用户时，只能修改其密码不能进行其他操作；当登录用户是管理员时，除了可以修改其密码，还可以查看用户信息；当登录用户是系统管理员时，可以进行所有操作。“用户设置窗体”界面如图 22.36 所示。

图 22.36 “用户设置窗体”界面

22.22.1　设计用户设置窗体

在项目“简易通企业进销存管理系统”的 zzk.zhuoyue.userframe 包中建立一个名称为 EditUserInfoFrame 的 JFrame 窗体，设置窗体布局为“边框式”布局，在窗体中心放一个选项卡面板，在选项卡面板中放 3 个面板，在第一个面板上放置 3 个标签、一个文本框、两个密码框和两个按钮。在第二个面板上放置两个标签、两个组合框和两个按钮。在第 3 个面板上放置一个标签、一个组合框、4 个按钮和一个滚动面板，在滚动面板上放一个表。该窗体中所用到的关键组件如表 22.25 所示。

表 22.25　用户设置窗体用到的关键组件

控 件 类 型	控件 ID	主要属性设置	用　途
JTable	tb_showuserinfo	设置 autoResizeMode 属性为 Off	显示用户信息
JComboBox	cmb_username	无	用户名列表
	cmb_quanxian	无	权限列表
JPasswordField	pwd_oldpassword	无	用户名称
	pwd_newpassword	无	密码
	pwd_oknewpassword	无	确认密码
JButton	btn_xiugai	设置 text 属性为“修改”	修改用户密码
	btn_queding	设置 text 属性为“确定”	设置用户权限
	btn_scchaxun	设置 text 属性为“查询”	查询用户信息
	btn_scquanbuxianshi	设置 text 属性为“全部显示”	显示所有用户信息
	btn_scshanchu	设置 text 属性为“删除”	删除用户
	btn_xgfanhui	设置 text 属性为“返回”	返回主窗体

22.22.2　修改用户密码

在“修改密码”选项卡中，可以修改登录用户的密码，在“原密码”文本框中输入原密码，在“新密码”文本框中输入新密码，在“确认新密码”文本框中再次输入新密码，单击“修改”按钮可以修改用户密码。“修改”按钮的事件处理过程代码如下：

代码位置：简易通企业进销存管理系统\src\zzk\zhuoyue\userframe\EditUserInfoFrame.java

```
private void btn_xiugaiActionPerformed(java.awt.event.ActionEvent evt) {
    String username=SaveUserInfo.getUserName();
    String oldpwd=new String(pwd_oldpassword.getPassword());
    String newpwd=new String(pwd_newpassword.getPassword());
    String oknewpwd=new String(pwd_oknewpassword.getPassword());
    if (newpwd==null || newpwd.equals("")){
        JOptionPane.showMessageDialog(this,"原密码不能为空！！！");
    }else if (oldpwd.equals(SaveUserInfo.getPassword())){
        if (newpwd==null || newpwd.equals("")){
            JOptionPane.showMessageDialog(this,"新密码不能为空！！！");
        }else if (oknewpwd==null || oknewpwd.equals("")){
            JOptionPane.showMessageDialog(this,"确认新密码不能为空！！！");
        }else if (!newpwd.equals(oknewpwd)){
            JOptionPane.showMessageDialog(this,"新密码和确认新密码不一致！！！");
```

```
            }else{
                try{
                    if (JOptionPane.showConfirmDialog(this, "确实要修改密码吗？\t\n 单击“确定”按钮将修改。",
"确认对话框", JOptionPane.OK_CANCEL_OPTION) == JOptionPane.OK_OPTION) {
                        String sql="update tb_user set password=? where username=?";
                        conn=daoConn.getConn();
                        ps=conn.prepareStatement(sql);
                        ps.setString(1,newpwd);
                        ps.setString(2,username);
                        ps.executeUpdate();
                        JOptionPane.showMessageDialog(this,"密码修改成功！！！");
                        SaveUserInfo.setPassword(newpwd);                    // 存储新密码信息
                    }
                }catch(Exception ex){
                    JOptionPane.showMessageDialog(this,"数据库异常。\t\n"+ex.getMessage());
                }
            }
        }else{
            JOptionPane.showMessageDialog(this,"原密码不正确！！！");
        }
        pwd_oldpassword.setText(null);                                       // 清空密码框中的内容
        pwd_newpassword.setText(null);
        pwd_oknewpassword.setText(null);
        pwd_oldpassword.requestFocus();
    }
```

注意：由于篇幅有限，这里只给出了“修改”按钮的事件处理过程，其他组件的事件处理过程和方法没有给出，可以参看光盘中的源程序。

22.22.3 设置用户权限

在“设置权限”选项卡中，以系统管理员身份登录的用户，可以修改管理员和一般用户的权限，在“用户名”组合框中选择用户，在“权限”组合框中选择要修改成何种权限（即选择管理员还是一般用户），单击“确定”按钮完成权限的修改，如图 22.37 所示。以管理员身份登录的用户，可以查看用户信息，但是不能修改用户权限。以一般用户身份登录的用户，不能选择“设置权限”选项卡。

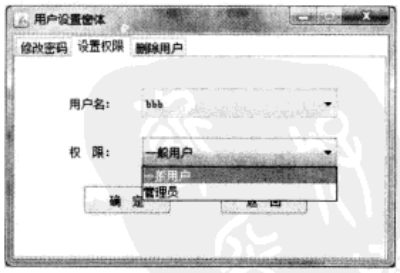


图 22.37 “设置权限”选项卡

选项卡面板的选项卡状态改变时系统首先判断登录用户的权限，系统管理员和管理员可以进入“设置权限”选项卡和“删除用户”选项卡，而一般用户不能进入并提示。选项卡状态改变事件的程序代码如下：

代码位置：简易通企业进销存管理系统\src\zzk\zhuoyue\userframe\EditUserInfoFrame.java

```
private void jTabbedPane1StateChanged(javax.swing.event.ChangeEvent evt) {
    // 选择“设置权限”选项卡
    if (jTabbedPane1.getTitleAt(jTabbedPane1.getSelectedIndex()).equals("设置权限")){
        if (SaveUserInfo.getQuanXian().equals("系统管理员")){
            // 在组合框中添加用户
```

```
            addUserToComboBox(cmb_username,"系统管理员");
                showQuanxian();                                    // 显示与组合框中用户对应的权限
            }else if (SaveUserInfo.getQuanXian().equals("管理员")){
                addUserToComboBox(cmb_username,"管理员");
                showQuanxian();
            }else{
                JOptionPane.showMessageDialog(this,"您不具有该权限，您只能修改密码！！！");
                jTabbedPane1.setSelectedIndex(0);                  // 设置活动选项卡索引为 0
            }
        }
        // 选择“删除用户”选项卡
        if (jTabbedPane1.getTitleAt(jTabbedPane1.getSelectedIndex()).equals("删除用户")){
            if (SaveUserInfo.getQuanXian().equals("系统管理员")){
                showUsername();                                    // 向删除页组合框中添加用户名
                showUserInfoToTable("全部");                       // 在表中显示用户名
                addUserToComboBox(cmb_username,"系统管理员");      // 向修改页组合框中添加用户名
            }else if (SaveUserInfo.getQuanXian().equals("管理员")){
                showUsername();
                showUserInfoToTable("全部");
                addUserToComboBox(cmb_username,"管理员");
            }else{
                JOptionPane.showMessageDialog(this,"您不具有该权限，您只能修改密码！！！");
                jTabbedPane1.setSelectedIndex(0);                  // 设置活动选项卡索引为 0
            }
        }
    }
```

在“设置权限”选项卡中，如果登录用户是系统管理员，可以修改用户权限，而管理员则不能修改用户权限。要修改权限，系统管理员可以从组合框中选择要修改权限的用户和权限，单击“确定”按钮完成修改操作。“确定”按钮的事件处理过程代码如下：

代码位置：简易通企业进销存管理系统\src\zzk\zhuoyue\userframe\EditUserInfoFrame.java

```
    private void btn_quedingActionPerformed(java.awt.event.ActionEvent evt) {
        try{
            if (SaveUserInfo.getQuanXian().equals("系统管理员")){
                String username=cmb_username.getSelectedItem().toString();
                String quanxian=cmb_quanxian.getSelectedItem().toString();
                String sql="update tb_user set quanxian=? where username=?";
                conn=daoConn.getConn();
                ps=conn.prepareStatement(sql);
                ps.setString(1,quanxian);
                ps.setString(2,username);
                ps.executeUpdate();
                JOptionPane.showMessageDialog(this,"权限设置成功！！！");
            }else if (SaveUserInfo.getQuanXian().equals("管理员")){
                JOptionPane.showMessageDialog(this,"管理员只能查看用户信息，不能设置权限！！！");
            }
        }catch(Exception ex){
            ex.printStackTrace();
        }
    }
```

注意：由于篇幅有限，这里只给出了主要组件的事件处理过程，其他组件的事件处理过程和方法没有给出，可以参看光盘中的源程序。

22.22.4 删除用户信息

在“删除用户”选项卡中，以系统管理员身份登录的用户，可以查看用户信息，还可以删除用户，在表中选择要删除的用户所在的行，单击“删除”按钮可删除选择的用户，如图 22.38 所示。以管理员身份登录的用户，可以查看用户信息，但是不能删除用户。以一般用户身份登录的用户，不能选择“删除用户”选项卡。

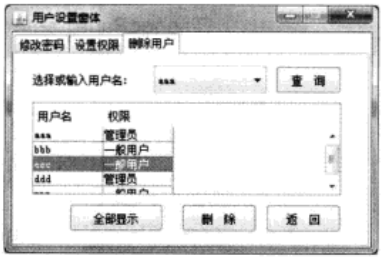


图 22.38 “删除用户”选项卡

在“删除用户”选项卡中，系统管理员可以删除和查看用户信息，而管理员只能查看用户信息但不能删除用户信息。从表中选择要删除的用户，单击“删除”按钮完成删除操作。“删除”按钮的事件处理过程代码如下：

代码位置：简易通企业进销存管理系统\src\zzk\zhuoyue\userframe\EditUserInfoFrame.java

```
private void btn_scshanchuActionPerformed(java.awt.event.ActionEvent evt) {
    try{
        if (SaveUserInfo.getQuanXian().equals("系统管理员")){
            int currentRow=tb_showuserinfo.getSelectedRow();          // 得到选择行的索引值
            String username="";
            Object object=tableModel.getValueAt(currentRow,0);       // 获得选择行的用户名对象
            if (object!=null){
                username=object.toString();                           // 把用户名对象转换成字符串
            }
            String sql="delete from tb_user where username=?";        // 定义删除用户的 SQL 语句
            conn=daoConn.getConn();
            ps=conn.prepareStatement(sql);
            ps.setString(1,username);                                 // 向 SQL 语句传递参数
            if (ps.executeUpdate()>0){
                JOptionPane.showMessageDialog(this,"删除成功！！！");
// 向“删除用户”选项卡的组合框中添加用户名
                showUsername();
                showUserInfoToTable("全部");                          // 在表中显示所有用户信息
            }
        }else if (SaveUserInfo.getQuanXian().equals("管理员")){
            JOptionPane.showMessageDialog(this,
"管理员只具有查看用户信息的权限，不能删除用户！！！");
        }
    }catch(Exception ex){
        ex.printStackTrace();
    }
}
```

注意：由于篇幅有限，这里只给出了“删除”按钮的事件处理过程，其他组件的事件处理过程和方法没有给出，可以参看光盘中的源程序。

22.23　经手人管理模块设计

本模块使用的数据表：tb_yginfo 和 tb_jsr

“经手人管理”模块用于添加经手人、启用或禁用经手人以及删除经手人等操作。“经手人管理”窗体界面如图 22.39 所示。

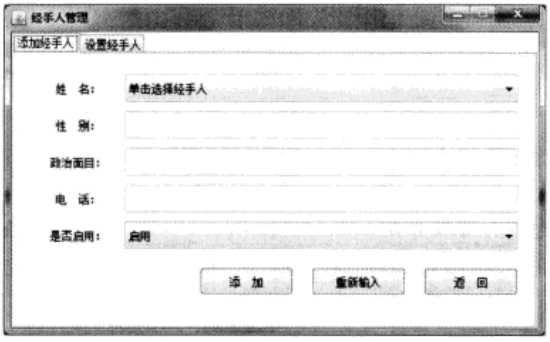


图 22.39　“经手人管理”窗体界面

22.23.1　设计经手人管理窗体

在项目“简易通企业进销存管理系统”的 zzk.zhuoyue.jbxxframe 包中建立一个名称为 JsrInfoFrame 的 JFrame 窗体，设置窗体布局为“边框式”布局，在窗体中心放一个选项卡面板，在选项卡面板中放两个面板，在第一个面板上放置 5 个标签、3 个文本框、两个组合框、3 个按钮和一个滚动面板，在滚动面板上放一个表。在第二个面板上放置 6 个标签、6 个文本框、3 个按钮和一个滚动面板，在滚动面板上放一个表。该窗体中所用到的关键组件如表 22.26 所示。

表 22.26　经手人管理窗体用到的关键组件

控 件 类 型	控件 ID	主要属性设置	用　途
JTextField	txf_xingbie	无	显示员工的性别
	txf_zhengzhimianmu	无	显示员工的政治面貌
	txf_dianhua	无	显示员工的电话
	txf_setId	设置 enabled 属性为 false	显示经手人 ID
	txf_setXingming	设置 enabled 属性为 false	显示经手人姓名
	txf_setXingbie	设置 enabled 属性为 false	显示经手人性别
	txf_setZhengzhimianmu	设置 enabled 属性为 false	显示经手人政治面貌
	txf_setDianhua	设置 enabled 属性为 false	显示经手人电话
	txf_setShifouqiyong	设置 enabled 属性为 false	显示启用或禁用
JTable	tb_yuangong	设置 autoResizeMode 属性为 Off	显示员工信息
	tb_showdata	设置 autoResizeMode 属性为 Off	显示经手人信息
JComboBox	cmb_xingming	无	员工姓名列表
	cmb_shifouqiyong	无	启用状态列表

续表

控 件 类 型	控件 ID	主要属性设置	用　途
JButton	btn_tianjia	设置 text 属性为“添加”	添加经手人
	btn_chongshu	设置 text 属性为“重新输入”	重新输入经手人
	btn_jinyongqiyong	设置 text 属性为“启用/禁用”	禁用或启用经手人
	btn_shanchu	设置 text 属性为“删除”	删除经手人信息
	btn_fanhui	设置 text 属性为“返回”	返回主窗体

22.23.2　添加经手人

在“添加经手人”选项卡中，可以添加经手人信息，在“姓名”右侧的组合框中选择员工姓名后，将显示该员工信息的表，单击表中该员工所在的行，系统自动添加员工信息到相应文本框中，选择是否启用，单击“添加”按钮保存经手人信息。“添加”按钮的程序代码如下：

代码位置：简易通企业进销存管理系统\src\zzk\zhuoyue\jbxxframe\JsrInfoFrame.java

```
private void btn_tianjiaActionPerformed(java.awt.event.ActionEvent evt) {
    try{
        String id=jsrid;
        String xingming=cmb_xingming.getSelectedItem().toString().trim();
        String xingbie=txf_xingbie.getText().trim();
        String zhengzhimianmu=txf_zhengzhimianmu.getText().trim();
        String dianhua=txf_dianhua.getText().trim();
        String shifouqiyong=cmb_shifouqiyong.getSelectedItem().toString().trim();
        String sql="insert into tb_jsr values(?,?,?,?,?,?)";
        if (xingming.equals("单击选择经手人")){
            JOptionPane.showMessageDialog(this,"请选择经手人。");
            return;
        }
        conn=daoConn.getConn();
        ps=conn.prepareStatement(sql);
        ps.setString(1,id);
        ps.setString(2,xingming);
        ps.setString(3,xingbie);
        ps.setString(4,zhengzhimianmu);
        ps.setString(5,dianhua);
        ps.setString(6,shifouqiyong);
        if (ps.executeUpdate()>0){
            addJsrToComboBox(cmb_xingming);                    // 调用方法，在组合框中添加经手人
            cmb_xingming.setEnabled(false);
            btn_tianjia.setEnabled(false);
            showJsrInfoToTable();                              // 调用方法，在表格中显示经手人信息
            JOptionPane.showMessageDialog(this,"保存成功！！！");
        }else{
            JOptionPane.showMessageDialog(this,"保存失败！！！");
        }
    }catch(Exception e){
```

```
            JOptionPane.showMessageDialog(this,"保存失败！！！\t\n"+
"数据库中已经存在该员工信息。");
            return;
        }
    }
```

注意：由于篇幅有限，这里只给出了“添加”按钮的事件处理过程，其他组件的事件处理过程和方法没有给出，可以参看光盘中的源程序。

22.23.3　设置经手人

在“设置经手人”选项卡中，可以启用或禁用经手人，还可以删除经手人，在表中选择要设置的经手人所在的行，单击“启用/禁用”按钮，可以启用或禁用经手人，并在表和对应的文本框中显示设置结果，如图 22.40 所示。从表中选择要删除的经手人所在的行，单击“删除”按钮，可以删除该经手人信息。

图 22.40　“设置经手人”选项卡

“设置经手人”选项卡中的“启用/禁用”按钮，可以把经手人设置为启用或禁用。“启用/禁用”按钮的程序代码如下：

代码位置：简易通企业进销存管理系统\src\zzk\zhuoyue\jbxxframe\JsrInfoFrame.java

```
private void btn_jinyongqiyongActionPerformed(java.awt.event.ActionEvent evt) {
    whetherOption(flag);                    // flag 是一个 boolean 的成员变量，用于判断启用或禁用状态
    showJsrInfoToTable();                   // 调用方法，在表格中显示经手人信息
}
```

用于启用或禁用经手人的方法根据布尔型的标识变量确定启用或禁用经手人，当标识变量为 false 时，把经手人设置为启用；当标识变量为 true 时，把经手人设置为禁用，该方法的程序代码如下：

代码位置：简易通企业进销存管理系统\src\zzk\zhuoyue\jbxxframe\JsrInfoFrame.java

```
private void whetherOption(boolean tf){
    try{
        String shifouqiyong="";                          // 定义变量存放启用状态
        if (tf==false){
            shifouqiyong="启用";
        }else
        {
            shifouqiyong="禁用";
        }
        conn=daoConn.getConn();                          // 获得数据库连接
```

```
            st=conn.createStatement();
            // 定义修改经手人启用状态的 SQL 语句
            String sql="update tb_jsr set sfky='"+shifouqiyong+"' where id ='"+jsrid+"'";
            st.executeUpdate(sql);                                    // 执行 SQL 语句
            txf_setShifouqiyong.setText(shifouqiyong);                // 在文本框中显示启用状态
        }catch(Exception ex){
            ex.printStackTrace();
        }
    }
```

注意：由于篇幅有限，这里只给出了“启用/禁用”按钮的事件处理过程和用于启用或禁用经手人的方法，其他组件的事件处理过程和方法没有给出，可以参看光盘中的源程序。

22.24 开发常见问题与解决

22.24.1 不显示系统登录窗体

（1）问题描述：在编写程序的过程中，需要运行程序进行测试，但是在运行时不显示系统登录窗体，在控制台输出如图 22.41 所示的异常信息。

```
Exception in thread "main" java.lang.NullPointerException
        at javax.swing.ImageIcon.<init>(Unknown Source)
        at zzk.zhuoyue.daohang.LoginFrame$PanelBack.<init>(LoginFrame.java:90)
        at zzk.zhuoyue.daohang.LoginFrame.<init>(LoginFrame.java:31)
        at zzk.zhuoyue.main.Main.main(Main.java:29)
```

图 22.41 异常信息

（2）发生问题的原因：在用 ImageIcon 创建图像时，图片文件的 URL 不正确，因而发生了空指针异常。

（3）问题的解决方法：指定图片的正确的 URL，然后再用 ImageIcon 创建图像。创建图像的正确代码如下：

```
URL url = PanelBack.class.getResource("/res/登录.jpg");                    // 获得图片的 URL
Image image = new ImageIcon(url).getImage();                               // 通过图片的 URL 创建图像对象
```

注意：代码中的图片文件“登录.jpg”所在的文件夹 res 必须放在项目 src 文件夹中，否则上面代码的第一行不能写成“/res/登录.jpg”。

22.24.2 运行程序时找不到数据库驱动

（1）问题描述：在编写程序的过程中，需要运行程序进行测试，但是在运行时能打开系统登录窗体，当输入用户名“mr”和密码“mrsoft”后，单击“登录”按钮，弹出数据库异常对话框，提示“com.mysql.jdbc.Driver”错误信息，如图 22.42 所示。

（2）发生问题的原因：没有把数据库驱动构建到类库路径，因而导致运行时找不到数据库驱动。

（3）问题的解决方法：在简易通企业进销存管理系统上单击鼠标右键，在弹出的快捷菜单中选择“属性”命令，在打开的“简易通企业进销存管理系统的属性”对话框的左侧列表中选择“Java 构建路径”选项，在右侧选择“库”选项卡，单击“添加 Jar”按钮，将打开“选择 Jar”对话框，选择 MySQL 的数据库

驱动，单击“确定”按钮，再单击“简易通企业进销存管理系统的属性”对话框中的“确定”按钮，完成把数据库驱动构建到类库路径的操作。

图 22.42　数据库异常对话框

22.24.3　运行程序时用户名或密码不正确

（1）问题描述：在编写程序的过程中，需要运行程序进行测试，但是在运行时能打开系统登录窗体，当输入用户名“mr”和密码“mrsoft”后，单击“登录”按钮，弹出“数据库异常”对话框，如图 22.43 所示。

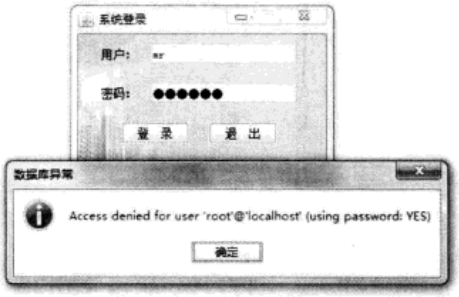


图 22.43　“数据库异常”对话框

（2）发生问题的原因：在公共类中连接数据库时，连接数据库服务器的用户名或密码不正确。

（3）问题的解决方法：在连接数据库的公共类中，把连接数据库的用户名改为正确的用户 root 和读者当前计算机系统中 MySQL 数据库的正确密码。

22.24.4　运行程序时找不到数据库

（1）问题描述：在编写程序的过程中，需要运行程序进行测试，但是在运行时能打开系统登录窗体，当输入用户名“mr”和密码“mrsoft”后，单击“登录”按钮，弹出“数据库异常”对话框，如图 22.44 所示。

图 22.44　“数据库异常”对话框

（2）发生问题的原因：在公共类中连接数据库时，数据库名书写不正确，导致应用程序找不到数据库。

（3）问题的解决方法：在连接数据库的公共类中，把数据库名改为本系统正确的数据库名 db_jxc 即可解决。

22.25 总　　结

本堂课重点介绍了简易通企业进销存管理系统的分析、设计、运行环境、数据库设计以及关键模块的开发过程、项目的运行和打包，还对程序开发中常见的问题与解决方法进行了讲解。通过对本堂课的学习，读者应该能够掌握应用程序的开发过程和 Swing 各种常用组件的使用，熟练使用菜单栏和工具栏，灵活应用表格组件，掌握 JDBC 技术和项目的开发、打包与发行过程。