Broadview
www.broadview.com.cn

高等院校软件专业方向 系列教材

Java SE
程序设计高级教程

青岛东合信息技术有限公司 青岛海尔软件有限公司 编著

電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

Java SE
程序设计高级教程

青岛东合信息技术有限公司　青岛海尔软件有限公司　编著

電子工業出版社
Publishing House of Electronics Industry
北京•BEIJING

内 容 简 介

本书在理解 Java 面向对象编程思想的基础上，深入讲解了 Java 的高级应用。全书共有 9 章，分别介绍了线程、网络编程、Swing 图形界面（包括两章）、事件处理、JDBC、RMI、国际化和正则表达式。书中涉及 Java GUI 设计及事件处理技巧，通过多线程实现 Java 多任务处理，通过 Socket 编程体验 Java 对网络编程的支持，通过 JDBC 实现 Java 访问数据库及 JDBC 的规范和技巧，系统地介绍 Java 远程调用 RMI 技术及安全策略，介绍正则表达式的原理及 Java 对正则表达式的支持等内容。

本书重点突出、偏重应用，结合理论篇的实例和实践篇对贯穿案例的讲解、剖析及实现，使读者能迅速理解并掌握知识，全面提高动手能力。

本书适应面广，可作为本科计算机科学与技术、软件外包专业、高职高专计算机软件、计算机网络、计算机信息管理、电子商务和经济管理等专业的程序设计课程的教材。

图书在版编目（CIP）数据

Java SE 程序设计高级教程 / 青岛东合信息技术有限公司，青岛海尔软件有限公司编著.
— 北京：电子工业出版社，2010.8
ISBN 978-7-121-11268-3

Ⅰ. ①J… Ⅱ. ①青… ②青… Ⅲ. ①JAVA 语言—程序设计—高等学校—教材 Ⅳ. ①TP312

中国版本图书馆 CIP 数据核字(2010)第 128285 号

责任编辑：张月萍
印　　刷：北京天宇星印刷厂
装　　订：三河市鹏成印业有限公司
出版发行：电子工业出版社
　　北京市海淀区万寿路 173 信箱　　邮编：100036
开　　本：787×1092　1/16　　印张：21　　字数：537.6 千字
印　　次：2010 年 8 月第 1 次印刷
定　　价：46.00 元

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

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

服务热线：（010）88258888。

前　言

随着 IT 产业的迅猛发展，企业对应用型人才的需求越来越大。“全面贴近企业需求，无缝打造专业实用人才”是目前高校计算机专业教育的革新方向。

该系列教材是面向高等院校软件专业方向的标准化教材。本教材充分结合软件企业的用人需求，经过了充分的调研和论证，并参照多所高校一线专家的意见，具有系统性、实用性等特点。旨在使读者在系统掌握软件开发知识的同时，着重培养其综合应用能力和解决问题的能力。

该系列教材具有如下几个特色。

1. 以应用型人才为导向来培养学生

强调实践：本系列教材以应用型软件及外包人才为培养目标，在原有体制教育的基础上对课程进行了改革，强化“应用型”技术的学习。使学生在经过系统、完整的学习后能够达到如下要求：

- 具备软件开发工作所需的理论知识和操作技能，能熟练进行编码工作，并掌握软件开发过程的规范；
- 具备一定的项目经验，包括代码的调试、文档编写和软件测试等内容；
- 相当于一年的软件开发经验。

2. 以实用技能为核心来组织教学

二八原则：遵循企业生产过程中的“二八原则”，即企业生产过程中 80%的时间在使用 20%的核心技术，强调核心教学，即学生在学校用 80%的学习时间来掌握企业中所用到的核心技术，从而保证对企业常用技术的掌握。教材内容精而专，同时配以知识拓展和拓展练习，以满足不同层次的教学和学习需求。

3. 以新颖的教材架构来引导学习

自成体系：本系列教材采用的教材架构打破了传统的以知识为标准编写教材的方法，采用“全真案例”和“任务驱动”的组织模式。

目　录

理论篇 1

第 1 章　线程 2
1.1　线程基础 4
1.1.1　线程概述 4
1.1.2　Java 线程模型 4
1.2　线程使用 5
1.2.1　创建线程 5
1.2.2　线程状态 8
1.2.3　线程优先级 12
1.2.4　线程组 14
1.3　多线程 15
1.3.1　多线程概述 15
1.3.2　线程同步 16
1.3.3　线程通信 19
1.3.4　死锁 22
小结 24
练习 25

第 2 章　网络编程 26
2.1　网络基础 28
2.1.1　网络类型 28
2.1.2　TCP/IP 协议 28
2.1.3　IP 地址 30
2.1.4　端口 31
2.1.5　域名与 DNS 31
2.2　网络 API 32
2.2.1　InetAddress 类 32
2.2.2　URL 类 34
2.2.3　URLConnection 类 36
2.3　基于 TCP 的网络编程 37
2.3.1　Socket 类 38
2.3.2　ServerSocket 类 39
2.3.3　C/S 实例 40
2.3.4　多线程 Socket 通信 43
小结 45

练习 45

第 3 章 Swing 图形界面（1） 47

3.1 Swing 概述 49
3.1.1 Swing 简介 49
3.1.2 Swing 的结构 49
3.2 容器 50
3.2.1 顶层容器 50
3.2.2 中间容器 52
3.3 布局 53
3.3.1 FlowLayout 53
3.3.2 BorderLayout 55
3.3.3 GridLayout 56
3.3.4 CardLayout 58
3.3.5 NULL 布局 60
3.4 Swing 常用组件 61
3.4.1 按钮 61
3.4.2 标签 62
3.4.3 图标 62
3.4.4 文本组件 63
3.4.5 复选框 64
3.4.6 单选按钮 65
3.4.7 列表框 66
3.4.8 组合框 67
3.5 Swing 组件示例 67
3.5.1 登录界面 67
3.5.2 注册界面 68
小结 71
练习 72

第 4 章 事件处理 73

4.1 事件概述 75
4.1.1 Java 事件处理机制 75
4.1.2 事件处理要点 75
4.1.3 Java 事件体系结构 76
4.2 事件处理 76
4.2.1 事件类 76
4.2.2 监听接口 77
4.3 事件示例 78
4.3.1 行为事件处理示例 78
4.3.2 选项事件处理示例 80
4.3.3 键盘事件处理 82
4.3.4 鼠标事件处理 84
4.4 适配器 85
4.5 多事件处理 87

小结 90
练习 90

第 5 章 Swing 图形界面（2） 92
5.1 菜单 94
5.1.1 菜单栏（JMenuBar） 94
5.1.2 菜单（JMenu） 94
5.1.3 菜单项（JMenuItem） 94
5.1.4 菜单示例 95
5.1.5 弹出式菜单 97
5.2 工具栏 99
5.3 对话框 100
5.3.1 标准对话框 100
5.3.2 对话框 104
5.3.3 文件对话框 107
5.3.4 颜色对话框 111
5.4 JTable 类 114
5.4.1 表格 114
5.4.2 表格模型 115
5.4.3 表格列模型 115
5.4.4 表格选择模式 116
5.5 JTree 类 119
5.5.1 树 119
5.5.2 树模型 120
5.5.3 树节点 120
5.5.4 树事件 121
小结 124
练习 124

第 6 章 JDBC 125
6.1 JDBC 基础 127
6.1.1 JDBC 概述 127
6.1.2 JDBC 结构 127
6.1.3 JDBC 类型 128
6.1.4 JDBC 与 ODBC 129
6.2 访问数据库 130
6.2.1 连接数据库 131
6.2.2 连接实例 134
6.3 操作数据库 136
6.3.1 Statement 接口 136
6.3.2 PreparedStatement 接口 140
6.3.3 CallableStatement 接口 142
6.4 集元数据 145
6.4.1 DatabaseMetaData 接口 145
6.4.2 ResultSetMetaData 接口 147

6.5 事务操作 148
6.5.1 事务 148
6.5.2 保存点 150
小结 151
练习 152

第 7 章 RMI 153

7.1 RMI 概述 155
7.1.1 分布式对象 155
7.1.2 RMI 156
7.1.3 RMI 机制原理 157
7.2 开发 RMI 158
7.2.1 定义远程接口 158
7.2.2 实现远程接口 159
7.2.3 编写服务器类 160
7.2.4 编写客户端 161
7.2.5 部署运行 162
7.2.6 注意事项 164
小结 164
练习 165

第 8 章 国际化 167

8.1 国际化和本地化 169
8.1.1 国际化概述 169
8.1.2 Locale 类 170
8.2 格式化处理 173
8.2.1 数字格式化 173
8.2.2 货币格式化 175
8.2.3 日期格式化 175
8.3 资源包 179
8.3.1 ListResourceBundle 180
8.3.2 PropertyResourceBundle 182
8.4 消息格式化 183
8.5 字符集 186
小结 187
练习 188

第 9 章 正则表达式 189

9.1 正则表达式 191
9.1.1 正则表达式概述 191
9.1.2 模式 191
9.1.3 常用正则表达式 195
9.2 在 Java 中应用正则表达式 196
9.2.1 Pattern 类 197
9.2.2 Matcher 类 198
9.2.3 应用实例 201

小结 205
练习 206

实践篇 207

实践 1 线程 208
实践指导 208
实践 1.G.1 208
实践 1.G.2 211
实践 1.G.3 213
知识拓展 217
1. 定时器 217
2. ThreadLocal 218
拓展练习 220
练习 1.E.1 220
练习 1.E.2 220

实践 2 网络编程 221
实践指导 221
实践 2.G.1 221
实践 2.G.2 224
实践 2.G.3 227
知识拓展 230
1. 基于 UDP 的网络编程 230
2. 基于 UDP 的组播通信 233
拓展练习 236
练习 2.E.1 236

实践 3 Swing 图形界面（1） 237
实践指导 237
实践 3.G.1 237
实践 3.G.2 238
实践 3.G.3 242
实践 3.G.4 244
知识拓展 246
1. JSplitPane 246
2. JScrollPane 247
拓展练习 249
练习 3.E.1 249
练习 3.E.2 249

实践 4 事件处理 250
实践指导 250
实践 4.G.1 250
实践 4.G.2 253

实践 4.G.3 258
知识拓展 266
1. AdjustmentListener 266
2. 人物眼球转动 268
拓展练习 269
练习 4.E.1 269
练习 4.E.2 269

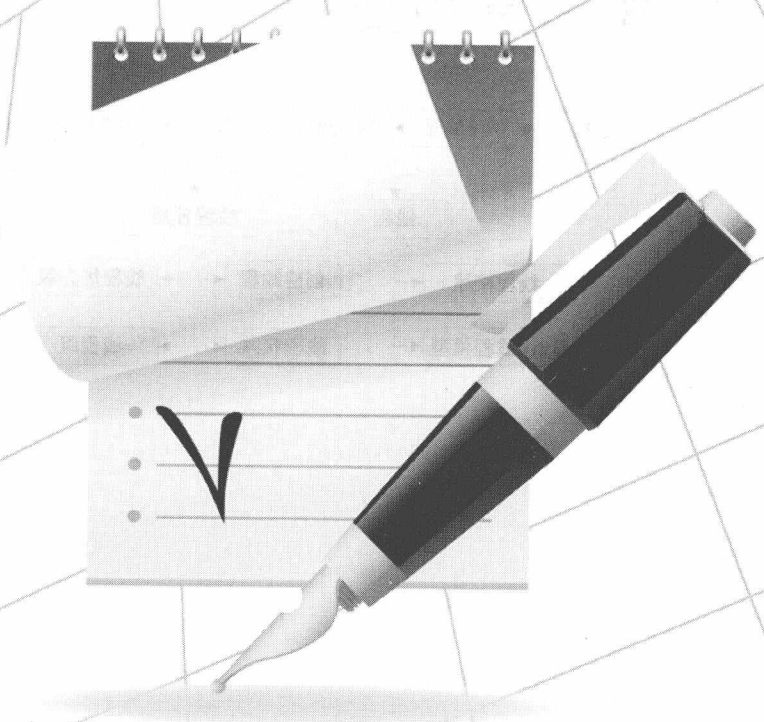
实践 5 Swing 图形界面（2） 270
实践指导 270
实践 5.G.1 270
实践 5.G.2 273
实践 5.G.3 276
实践 5.G.4 281
知识拓展 291
1. 2D 绘图 291
拓展练习 293
练习 5.E.1 293

实践 6 JDBC 294
实践指导 294
实践 6.G.1 294
实践 6.G.2 296
知识拓展 301
1. 可滚动和可更新的结果集 301
2. 操作 BLOB/CLOB 数据 304
3. 批处理 307
4. 使用 JDBC 连接不同的数据库 309
拓展练习 310
练习 6.E.1 310
练习 6.E.2 310

实践 7 RMI 311
实践指导 311
实践 7.G.1 311
知识拓展 317
1. CORBA 技术 317
2. SOAP 319
拓展练习 320
练习 7.E.1 320
练习 7.E.2 320

附录 A 正则表达式元字符 321

理论篇

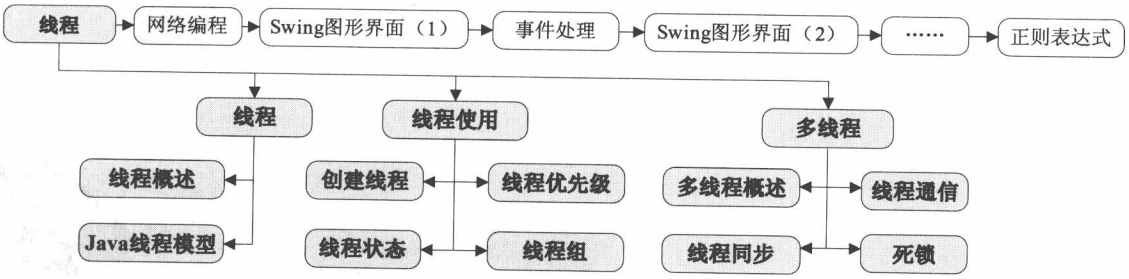


第1章 线 程

本章目标

- 理解线程的基本概念
- 理解 Java 的线程模型
- 掌握 Java 线程的状态和状态转换
- 掌握线程的创建和使用
- 掌握线程优先级的使用
- 掌握线程组的使用
- 理解多线程的概念
- 掌握 Java 的多线程实现
- 掌握线程的同步技巧
- 掌握线程的通信方式
- 理解死锁的概念

学习导航



任务描述

【描述 1.D.1】

通过 Thread 类获取程序的主线程。

【描述 1.D.2】

通过创建 Thread 类的子类，演示线程的创建。

【描述 1.D.3】

通过实现 Runnable 接口，演示线程的创建。

【描述 1.D.4】

演示线程的创建、运行和停止三个状态之间的切换。

【描述 1.D.5】

利用线程的 sleep 状态实现周期性打印信息。

【描述 1.D.6】

利用线程的 interrupt 方法实现线程中断。

【描述 1.D.7】

演示线程优先级的设置及使用。

【描述 1.D.8】

演示线程组的创建和使用。

【描述 1.D.9】

一个简单的多线程例子，能根据用户提供的字符，打印其后连续的 30 个字符。

【描述 1.D.10】

通过多线程，演示不使用同步机制可能出现的情况。

【描述 1.D.11】

通过生产/消费模型，演示线程通信机制的应用。

【描述 1.D.12】

使用资源竞争模型，演示死锁的产生。

1.1 线程基础

线程（Thread）是独立于其他线程运行的程序执行单元。在 Java 体系中，线程在多任务处理中起着举足轻重的作用。

1.1.1 线程概述

线程（轻量级程序）类似于一个程序，也有开始、执行和结束。它是运行在程序内部的一个比进程还要小的单元。使用线程的主要原因在于可以在一个程序中同时运行多个任务。每个 Java 程序都至少有一个线程——主线程。当一个 Java 程序启动时，JVM 会创建主线程，并在该线程中调用程序的 main()方法。

多线程就是同时有多个线程在执行。在多 CPU 的计算机中，多线程的实现是真正的物理上的同时执行，而对于单 CPU 的计算机而言，实现的只是逻辑上的同时执行。在每个时刻，真正执行的只有一个线程，由操作系统进行线程管理调度，但由于 CPU 的速度很快，让人感到像是多个线程在同时执行。

进程是指一种“自包容”的运行程序，有自己的地址空间；线程是进程内部单一的一个顺序控制流。基于进程的特点是允许计算机同时运行两个或更多的程序。基于线程的多任务处理环境中，线程是最小的处理单位。多线程程序在更低的层次中引入多任务处理。

多进程与多线程是多任务的两种类型。多线程与多进程的主要区别在于，线程是一个进程中一段独立的控制流，一个进程可以拥有若干个线程。在多进程设计中各个进程之间的数据块是相互独立的，一般彼此不影响，要通过信号、管道等进行交流。而在多线程设计中，各个线程不一定独立，同一任务中的各个线程共享程序段、数据段等资源。

多线程比多进程更方便于共享资源，而 Java 提供的同步机制解决了线程之间的数据完整性问题，使得多线程设计更易发挥作用。在 Java 程序设计中，动画设计及多媒体应用都会广泛地使用到多线程。

引入线程的优点是：

- 充分利用 CPU 资源。
- 简化编程模型。
- 简化异步事件处理。
- 使 GUI 更有效率。
- 节约成本。

1.1.2 Java 线程模型

Java 的线程模型是面向对象的。在 Java 中建立线程有两种方法：一种是继承 Thread 类；另一种是实现 Runnable 接口，并通过 Thread 和实现 Runnable 的类来建立线程。

Java 通过 Thread 类将线程所必需的功能都封装了起来。要想建立一个线程，必须要有一

个线程执行函数，这个线程执行函数对应 Thread 类的 run()方法；Thread 类还有一个 start()方法，这个方法负责启动线程，当调用 start()方法后，如果线程启动成功，将自动调用 Thread 类的 run()方法。因此，任何继承 Thread 的 Java 类都可以通过 Thread 类的 start()方法来建立线程。如果想运行自己的线程执行函数，那就要重写 Thread 类的 run()方法。

Java 线程模型中还提供了一个标识某个 Java 类是否可作为线程类的接口——Runnable，该接口只有一个抽象方法 run()，也就是 Java 线程模型的线程执行函数。

这两种方法从本质上说是一致的，即都是通过 Thread 类来建立线程，并运行 run()方法的。但它们的区别在于：由于 Java 不支持多继承，因此，这个线程类如果继承了 Thread，就不能再继承其他的类了，而通过实现 Runnable 接口的方法来建立线程，这样的线程类可以在必要的时候继承和业务有关的类，形成清晰的数据模型。

1.2　线程使用

在 Java 中创建线程有几种方法。每个 Java 程序至少包含一个线程：主线程。其他线程都是通过 Thread 构造器或实例化继承类 Thread 的类来创建的。

下述代码用于实现任务描述 1.D.1，通过 Thread 类获取程序的主线程。

【描述 1.D.1】 MainThread.java

```
class MainThread {
    public static void main(String args[]) {
        // 调用 Thread 类的 currentThread()方法获取当前线程
        Thread t = Thread.currentThread();
        System.out.println("主线程是: " + t);
    }
}
```

1.2.1　创建线程

在 Java 中创建线程有两种方法：使用 Thread 类和使用 Runnable 接口。在使用 Runnable 接口时需要建立一个 Thread 实例。因此，无论是通过 Thread 类还是 Runnable 接口建立线程，都必须建立 Thread 类或它的子类的实例。

1. Thread

Java 线程可以通过直接实例化 Thread 对象或实例化继承 Thread 的对象来创建其他线程。Thread 子类需要重写 run 方法，并在 run 方法内部定义线程的功能语句，其常用方法及使用说明见表 1-1。

表 1-1　Thread 类的方法列表

方法名	功能说明
Thread()	构造默认的线程对象
Thread(Runnable target)	使用传递的 Runnable 构造线程对象
Thread(Runnable target,String name)	使用传递的 Runnable 构造名为 name 的线程对象
Thread(ThreadGroup group,Runnable target, String name)	使用传递的 Runnable 在 group 线程组内构造名为 name 的线程对象
final String getName()	返回线程的名称
final boolean isAlive()	如果线程是激活的，则返回 true
final void setName(String name)	将线程的名称设置为由 name 指定的名称
set\getPriority()	设置得到线程优先级
final void join()	等待线程结束
static void sleep(long millis)	用于将线程挂起一段时间，单位毫秒
void start()	调用 run( )方法启动线程，开始线程的执行
void stop()	停止线程，已经不建议使用
void interrput()	中断线程
static int activeCount()	返回激活的线程数
static void yield()	使正在执行的线程临时暂停，并允许其他线程执行

下述代码用于实现任务描述 1.D.2，通过创建 Thread 类的子类，演示线程的创建。

【描述 1.D.2】 ThreadDemo.java

```
class Thread1 extends Thread {
    public void run() {
        // 获取当前线程的名字
        System.out.println(Thread.currentThread().getName());
    }
}
class Thread2 extends Thread {
    public Thread2(String name){
        super(name);
    }
    public void run() {
        // 获取当前线程的名字
        System.out.println(Thread.currentThread().getName());
    }
}
public class ThreadDemo {
    public static void main(String[] args) {
        Thread1 thread1 = new Thread1();
        // 构造名为 thread2 的线程对象
        Thread2 thread2 = new Thread2("thread2");
```

```
        thread1.start();
        thread2.start();
        // 获取主线程的名字
        System.out.println("["+Thread.currentThread().getName()+"]");
    }
}
```

可能的执行结果：

```
Thread-0
[main]
thread2
```

上述代码定义了两个线程 Thread1 和 Thread2，在创建 thread1 对象时并未指定线程名，因此，所输出的线程名是系统的默认值。而对于输出结果，不仅不同机器之间的结果可能不同，而且在同一机器上多次运行同一程序也可能生成不同结果。另外，线程的执行次序是不定的，除非使用同步机制以强制按特定的顺序执行。

注意　Thread 类的 start 方法不能多次调用，如不能调用两次 thread1.start()方法，否则会抛出一个 IllegalThreadStateException 异常。

2. Runnable

创建线程的另一种方式是实现 Runnable 接口。实现 Runnable 接口的类必须使用 Thread 类的实例才能创建线程。通过 Runnable 接口创建线程的步骤如下：

01　实例化实现 Runnable 接口的类。

02　建立一个 Thread 对象，并将第一步实例化后的对象作为参数传入 Thread 类的构造方法。

03　通过 Thread 类的 start()方法建立线程。

下述代码用于实现任务描述 1.D.3，通过实现 Runnable 接口，演示线程的创建。

【描述 1.D.3】 RunnableDemo.java

```
class Thread3 implements Runnable {
    // 重写 run 方法
    public void run() {
        // 获取当前线程的名字
        System.out.println(Thread.currentThread().getName());
        for (int i=0;i<30;i++){
            System.out.print("A");
        }
    }
}
```