Broadview®
www.broadview.com.cn

· Java多线程编程实战系列 ·

本书用Java诠释多线程编程的“三十六计”——多线程设计模式。
每个设计模式的讲解都附有实战案例及源码解析，从理论到实战经验，
全面呈现常用多线程设计模式的来龙去脉。

Java 多线程编程实战指南

（设计模式篇）

黄文海 / 著

中国工信出版集团
電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

Java 多线程编程实战指南

（设计模式篇）

黄文海 / 著

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

内 容 简 介

随着 CPU 多核时代的到来，多线程编程在充分利用计算资源、提高软件服务质量方面扮演了越来越重要的角色。而解决多线程编程中频繁出现的普遍问题可以借鉴设计模式所提供的现成解决方案。然而，多线程编程相关的设计模式书籍多采用 C++作为描述语言，且书中所举的例子多与应用开发人员的实际工作相去甚远。本书采用 Java（JDK1.6）语言和 UML 为描述语言，并结合作者多年工作经历的相关实战案例，介绍了多线程环境下常用设计模式的来龙去脉：各个设计模式是什么样的及其典型的实际应用场景、实际应用时需要注意的事项以及各个模式的可复用代码实现。

本书适合有一定 Java 多线程编程基础、经验的读者。

图书在版编目（CIP）数据

Java 多线程编程实战指南：设计模式篇/黄文海著. 北京：电子工业出版社，2015.10
（Java 多线程编程实战系列）
ISBN 978-7-121-27006-2

Ⅰ. ①J… Ⅱ. ①黄… Ⅲ. ①JAVA 语言－程序设计－指南 Ⅳ. ①TP312-62

中国版本图书馆 CIP 数据核字(2015)第 195631 号

责任编辑：付　睿
印　　刷：中国电影出版社印刷厂
装　　订：中国电影出版社印刷厂
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：787×980　1/16　印张：17.5　字数：381 千字
版　　次：2015 年 10 月第 1 版
印　　次：2015 年 10 月第 1 次印刷
印　　数：3000 册　定价：59.00 元

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

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

服务热线：（010）88258888。

推荐序

欣闻文海兄弟的《Java 多线程编程实战指南》一书即将出版，心里感到非常激动和兴奋。与文海相识于 2014 年，某一天 InfoQ 中文站的运营编辑给我转发了一封读者投稿的邮件，标题是《Java 多线程编程模式实战指南之 Active Object 模式》。读完了稿件后立刻决定发布到 InfoQ 中文站上，因为这篇文章无论从内容选取、技术方向还是文字水平都是我见过的上乘之作。文章发布后也收到了很多读者的反馈，而该文章的作者正是文海。此后，文海又在 InfoQ 中文站上连载了多篇关于 Java 多线程设计模式相关的文章，均获得了不错的读者评价。

本书正是文海多年来工作经验的总结之作。众所周知，目前 Java 并发领域的经典好书大部分都是外版作品。不过值得欣喜的是，近一两年来，也有一些不错的国内开发者开始编写这个领域的图书，口碑也相当不错。文海的这部著作针对 Java 并发编程但又不局限于这个领域，它将 Java 多线程编程与设计模式这两大主题有机地结合到了一起。实际上，目前市场上虽然既有关于 Java 多线程编程的图书，也有关于设计模式的图书，但这两类图书内容之间却难以产生交集。介绍 Java 多线程的图书会专门讲解多线程编程的方方面面，而介绍设计模式的图书一般又会以经典的 23 种设计模式为蓝本，同时辅以一些简单的代码示例进行解读，难以让读者真正领会设计模式在实际开发中所起的作用。这本《Java 多线程编程实战指南》正是这两个领域的集大成者，它不仅深入透彻地分析了 Java 多线程编程的方方面面，还将其与设计模式有机地结合到了一起，形成了主动对象模式、两阶段终止模式、生产者/消费者模式、流水线模式、线程池模式等对实际项目开发会起到积极指导作用的诸多模式。可以这么说，本书不仅会向大家介绍 Java 多线程开发的难点与重点，还会探讨在某些场景下该使用哪种模式，这样做会给项目带来什么好处。从这个意义上来说，本书是 Java 多线程开发与设计模式理论的集大成者，相信会给广大的 Java 开发者带来切实的帮助。

目前已经是多核普及的时代，程序员也一定要编写面向多核的代码。虽然传统的 SSH（特指 Struts+Spring+Hibernate）依然还在发挥着重要的作用，但不得不说的是，作为一名有追求的

Java 开发者，眼光不应该局限于此。每一名有理想的 Java 开发者都应该系统学习有关多线程编程的知识，这不仅涉及程序语言与库的学习，还需要了解现代硬件体系架构（如 CPU、缓存、内存等），同时辅以恰当的设计模式，这样才能在未来游刃有余、得心应手。

虽然本人已经出版过多本技术图书，但为别人的书写序还是第一次。因此，在写这篇序之前我通读了该书的全部章节。事实也印证了之前的猜想，文海的这本书绝对是他本人的心血结晶之作，诸多的实际经验相信会给你带来不一样的感受。诚然，目前 Java 开发相关的技术图书已然汗牛充栋，但我相信，这本《Java 多线程编程实战指南》应该是每一个对代码有追求、对模式有见地的读者书架上不可或缺的一本书。

InfoQ 中文站 Java 主编：张龙

2015 年 9 月 14 日于北京

前言

随着现代 CPU 的生产工艺从提升 CPU 主频频率转向多核化，即在一块芯片上集成多个 CPU 内核（Core），以往那种靠 CPU 自身处理能力的提升所带来的软件计算性能提升的“免费午餐”不复存在。在此背景下，多线程编程在充分利用计算资源、提高软件服务质量方面扮演了越来越重要的角色。然而，多线程编程并非一个简单地使用多个线程进行编程的数量问题，其又有自身的问题。好比俗话说“一个和尚打水喝，两个和尚挑水喝，三个和尚没水喝”，简单地使用多个线程进行编程可能导致更加糟糕的计算效率。

设计模式相当于软件开发领域的“三十六计”，它为特定背景下反复出现的问题提供了一般性解决方案。多线程相关的设计模式为我们恰当地使用多线程进行编程并达到提升软件服务质量这一目的提供了指引和参考。当然，设计模式不是菜谱。即便是菜谱，我们也不能指望照着菜谱做就能做出一道美味可口的菜肴，但我们又不能因此而否认菜谱存在的价值。

可惜的是，国外与多线程编程相关的设计模式书籍多数采用 C++作为描述语言，且书中所举的例子又多与应用开发人员的实际工作经历相去甚远。本书作为国内第一本多线程编程相关设计模式的原创书籍，希望能够为 Java 开发者普及多线程相关的设计模式开一个头。

本书采用 Java（JDK1.6）语言和 UML（Unified Modeling Language）为描述语言，并结合作者多年工作经历的相关实战案例，介绍了多线程环境下常用设计模式的来龙去脉：各个设计模式是什么样的及其典型的实际应用场景、实际应用时需要注意的相关事项以及各个模式的可复用代码实现。

本书第 1 章对多线程编程基础进行了回顾，虽然该章讲的是基础，但重点仍然是强调“实战”。所谓“温故而知新”，有一定多线程编程基础、经验的读者也不妨快速阅读一下本章，说不定有新的收获。

本书第 3 章到第 14 章逐一详细讲解了多线程编程相关的 12 个常用设计模式。针对每个设计

模式，相应章节会从以下几个方面进行详细讲解。

模式简介。这部分简要介绍了相应设计模式的由来及核心思想，以便读者能够快速地对相应设计模式有个初步认识。

模式的架构。这部分会从静态（类及类与类之间的结构关系）和动态（类与类之间的交互）两个角度对相应设计模式进行详细讲解。模式架构分别使用 UML 类图（Class Diagram）和序列图（Sequence Diagram）对模式的静态和动态两个方面进行描述。

实战案例解析。在相应设计模式架构的基础上，本部分会给出相关的实战案例并对其进行解析。不同于教科书式的范例，实战案例强调的是“实战”这一背景。因此实战案例解析中，我们会先提出实际案例中我们面临的实际问题，并在此基础上结合相应设计模式讲解相应设计模式是如何解决这些问题的。实战案例解析中我们会给出相关的 Java 代码，并讲解这些代码与相应设计模式的架构间的对应关系，以便读者进一步理解相应设计模式。为了便于读者进行实验，本书给出的实战案例代码都力求做到可运行。实战案例解析有助于读者进一步理解相应的设计模式，并体验相应设计模式的应用场景。建议读者在阅读这部分时先关注重点，即实战案例中我们要解决哪些问题，相应设计模式又是如何解决这些问题的，以及实战案例的代码与相应设计模式的架构间的对应关系。而代码中其与设计模式非强相关的细节则可以稍后关注。

模式的评价与实现考量。这部分会对相应设计模式在实现和应用过程中需要注意的一些事项、问题进行讲解，并讨论应用相应设计模式所带来的好处及缺点。该节也会讨论相应设计模式的典型应用场景。

可复用实现代码。这部分给出相应设计模式的可复用实现代码。编写设计模式的可复用代码有助于读者进一步理解相应设计模式及其在实现和应用过程中需要注意的相关事项和问题，也便于读者在实际工作中应用相应设计模式。

Java 标准库实例。考虑到 Java 标准库的 API 设计过程中已经应用了许多设计模式，本书尽可能地给出相应设计模式在 Java API 中的应用情况。

相关模式。设计模式不是孤立存在的，一个具体的设计模式往往和其他设计模式之间存在某些联系。这部分会描述相应设计模式与其他设计模式之间存在的关系。这当中可能涉及 GOF 的设计模式，这类设计模式并不在本书的讨论范围之内。有需要的读者，请自行参考相关书籍。

本书的源码可以从 http://github.com/Viscent/javamtp 下载或博文视点官网 http://www.broadview.com.cn 相关图书页面下载。

目录

第1章

Java多线程编程实战基础

温故而知新

——《论语·为政》

有一定的多线程编程基础和工作经验的读者，也不妨继续往下看，看后或许会有新的发现。这一章的内容并非纯粹的理论“基础”，它更加强调“实战”。

1.1 无处不在的线程

进程（Process）代表运行中的程序。一个运行的Java程序就是一个进程。从操作系统的角度来看，线程（Thread）是进程中可独立执行的子任务。一个进程可以包含多个线程，同一个进程中的线程共享该进程所申请到的资源，如内存空间和文件句柄等。从JVM的角度来看，线程是进程中的一个组件（Component），它可以看作执行Java代码的最小单位。Java程序中任何一段代码总是执行在某个确定的线程中的。JVM启动的时候会创建一个main线程，该线程负责执行Java程序的入口方法（main方法）。如清单1-1所示的代码展示了Java程序中的代码总是由某个确定的线程运行的。

清单1-1. Java代码的执行线程

```
public class JavaThreadAnywhere {

    public static void main(String[] args) {
        System.out.println("The main method was executed by thread:"
            + Thread.currentThread().getName());
        Helper helper = new Helper("Java Thread Anywhere");
        helper.run();
    }

    static class Helper implements Runnable {
        private final String message;
```

```
        public Helper(String message) {
            this.message = message;
        }

        private void doSomething(String message) {
            System.out.println("The doSomething method was executed by thread:"
                + Thread.currentThread().getName());
            System.out.println("Do something with " + message);
        }

        @Override
        public void run() {
            doSomething(message);

        }
    }

}
```

如清单 1-1 所示的程序运行时输出如下：

```
The main method was executed by thread:main
The doSomething method was executed by thread:main
Do something with Java Thread Anywhere
```

从上面的输出可以看出，类 JavaThreadAnywhere 的 main 方法以及类 Helper 的 doSomething 方法都是由 main 方法负责执行的。

在多线程编程中，弄清楚一段代码具体是由哪个（或者哪种）线程去负责执行的这点很重要，这关系到性能问题、线程安全问题等。本书的后续章节会体现这点。

Java 中的线程可以分为守护线程（Daemon Thread）和用户线程（User Thread）。用户线程会阻止 JVM 的正常停止[1]，即 JVM 正常停止前应用程序中的所有用户线程必须先停止完毕；否则 JVM 无法停止。而守护线程则不会影响 JVM 的正常停止，即应用程序中有守护线程在运行也不影响 JVM 的正常停止。因此，守护线程通常用于执行一些重要性不是很高的任务，例如用于监视其他线程的运行情况。

1.2 线程的创建与运行

在 Java 语言中，一个线程就是一个 java.lang.Thread 类的实例。因此，在 Java 语言中创建一个线程就是创建一个 Thread 类的实例，当然这离不开内存的分配。创建一个 Thread 实例与创建其他类的实例所不同的是，JVM 会为一个 Thread 实例分配两个调用栈（Call Stack）所需的内存空间。这两个调用栈一个用于跟踪 Java 代码间的调用关系，另一个用于跟踪 Java

1 即通过 System.exit 调用停止 JVM，而非强行停止 JVM（如在 Linux 系统下使用 kill 命令停止 Java 进程）。