HZ BOOKS
华章IT

技术畅销书全新升级，案例式讲解，逐一分析和验证每个技术点，通俗易懂

涵盖多线程编程的核心库、方法、原理，解决高并发环境下的业务瓶颈

Java
核心技术
系列

Java 多线程编程核心技术

（第2版）

Java Multi-thread Programming

（Second Edition）

高洪岩 著

机械工业出版社
China Machine Press

Java·核心技术系列

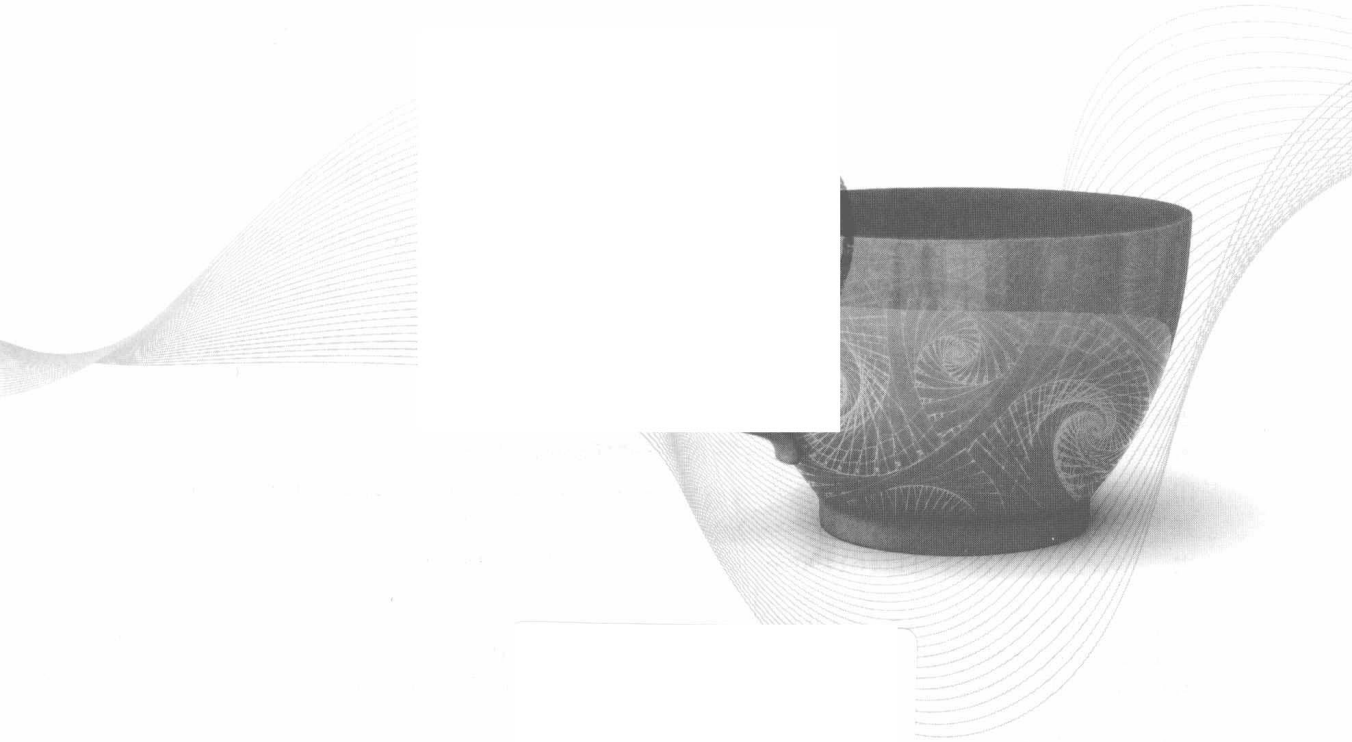
Java 多线程编程核心技术

（第2版）

Java Multi-thread Programming

（Second Edition）

高洪岩 著



机械工业出版社
China Machine Press

图书在版编目（CIP）数据

Java 多线程编程核心技术 / 高洪岩著 . —2 版 . —北京：机械工业出版社，2019.1
（Java 核心技术系列）

ISBN 978-7-111-61490-6

I. J…　II. 高…　III. JAVA 语言 - 程序设计　IV. TP312.8

中国版本图书馆 CIP 数据核字（2018）第 274886 号

Java 多线程编程核心技术（第 2 版）

出版发行：机械工业出版社（北京市西城区百万庄大街 22 号　邮政编码：100037）
责任编辑：赵亮宇　　责任校对：殷　虹
印　　刷：北京市荣盛彩色印刷有限公司　　版　　次：2019 年 5 月第 2 版第 1 次印刷
开　　本：186mm × 240mm　1/16　　印　　张：28.25
书　　号：ISBN 978-7-111-61490-6　　定　　价：99.00 元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换
客服热线：（010）88379426　88361066　　投稿热线：（010）88379604
购书热线：（010）68326294　　读者信箱：hzit@hzbook.com

Preface 前　言

本书是国内首本整本系统、完整地介绍 Java 多线程技术的书籍，作为笔者，我要感谢大家的支持与厚爱。

本书第 1 版在出版后获得了广大 Java 程序员与学习者的关注，技术论坛、博客、公众号等平台大量涌现出针对 Java 多线程技术的讨论与分享。能为国内 IT 知识的建设贡献微薄之力是让我最欣慰的。

有些读者在第一时间就根据书中的知识总结了学习笔记，并在博客中进行分享，笔者非常赞赏这种传播知识的精神。知识就要分享，知识就要传播，这样才能共同进步。

第 2 版与第 1 版的区别

本书第 1 版上市后收到了大量的读者反馈，我对每一个建议都细心地进行整理，力求在第 2 版中得以完善。

第 2 版在第 1 版的基础上着重加强了 8 点更新：

1）大量知识点重排，更有利于阅读与理解；

2）更新了读者提出的共性问题并进行集中讲解；

3）丰富 Thread.java 类 API 的案例，使其更具有实用性；

4）对线程的信息进行监控实时采样；

5）强化了 volatile 语义、多线程核心 synchronized 的案例；

6）力求知识点连贯，方便深度学习与理解，增加原子与线程安全的内容；

7）深入浅出地介绍代码重排特性；

8）细化工具类 ThrealLocal 和 InheritableThreadLocal 的源代码分析与原理。

由于篇幅有限，有关线程池的知识请参考笔者的另一本书——《 Java 并发编程：核心

方法与框架》，那本书中有针对 Java 并发编程技术的讲解。在向分布式领域进军时还需要用到 NIO 和 Socket 技术，故推荐笔者的拙作《 NIO 与 Socket 编程技术指南》，希望可以给读者带来一些帮助。

本书秉承大道至简的主导思想，只介绍 Java 多线程开发中最值得关注的内容，希望抛砖引玉，以个人的一些想法和见解，为读者拓展出更深入、更全面的思路。

本书特色

在撰写本书的过程中，我尽量少用“啰唆”的文字，全部以 Demo 式案例来讲解技术点的实现，使读者看到代码及运行结果后就可以知道项目要解决的是什么问题，类似于网络中博客的风格，让读者用最短的时间学习知识点，明白知识点如何应用，以及在使用时要避免什么，使读者能够快速学习知识并解决问题。

读者对象

- Java 程序员；
- 系统架构师；
- Java 多线程开发者；
- Java 并发开发者；
- 大数据开发者；
- 其他对多线程技术感兴趣的人员。

如何阅读本书

本书本着实用、易懂的学习原则，利用 7 章来介绍 Java 多线程相关的技术。

第 1 章讲解了 Java 多线程的基础，包括 Thread 类的核心 API 的使用。

第 2 章讲解了在多线程中对并发访问的控制，主要是 synchronized 的使用。由于此关键字在使用上非常灵活，所以该章用很多案例来说明它的使用，为读者学习同步知识打好坚实的基础。

第 3 章讲解了线程之间的通信与交互细节。该章主要介绍 wait()、notifyAll() 和 notify() 方法的使用，使线程间能够互相通信，合作完成任务。该章还介绍了 ThreadLocal 类的使用。学习完该章，读者就能在 Thread 多线程中进行数据的传递了。

第 4 章讲解了 Lock 对象。因为 synchronized 关键字使用起来比较麻烦，所以 Java 5 提

供了 Lock 对象，更好地实现了并发访问时的同步处理，包括读写锁等。

第 5 章讲解了 Timer 定时器类，其内部原理是使用多线程技术。定时器在执行计划任务时是很重要的，在进行 Android 开发时也会深入使用。

第 6 章讲解的单例模式虽然很简单，但如果遇到多线程将会变得非常麻烦。如何在多线程中解决这么棘手的问题呢？本章会全面给出解决方案。

第 7 章对前面章节遗漏的技术空白点进行补充，通过案例使多线程的知识体系更加完整，尽量做到不出现技术空白点。

交流和支持

由于笔者水平有限，加上编写时间仓促，书中难免会出现一些疏漏或者不准确的地方，恳请读者批评指正，期待能够得到你们的真挚反馈，在技术之路上互勉共进。

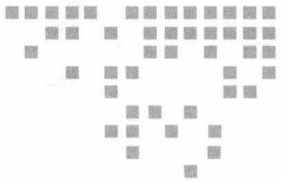
联系笔者的邮箱是 279377921@qq.com。

致谢

在本书出版的过程中，感谢公司领导和同事的大力支持，感谢家人给予我充足的时间来撰写稿件，感谢出生 3 个多月的儿子高晟京，看到你，我有了更多动力，最后感谢在此稿件上耗费大量精力的高婧雅编辑与她的同事们，是你们的鼓励和帮助，引导我顺利完成了本书。

高洪岩

Contents 目　录



第 1 章 Chapter 1

Java 多线程技能

作为本书的第 1 章，重点是让读者快速进入 Java 多线程的学习，所以本章主要介绍 Thread 类的核心方法。Thread 类的核心方法较多，读者应该着重掌握如下技术点：

- 线程的启动；
- 如何使线程暂停；
- 如何使线程停止；
- 线程的优先级；
- 线程安全相关的问题。

以上内容也是本章学习的重点与思路，掌握这些内容是进入 Java 多线程学习的必经之路。

1.1 进程和多线程概述

本书主要介绍在 Java 语言中使用的多线程技术，但讲到多线程技术时不得不提及“进程”这个概念，“百度百科”对“进程”的解释如图 1-1 所示。

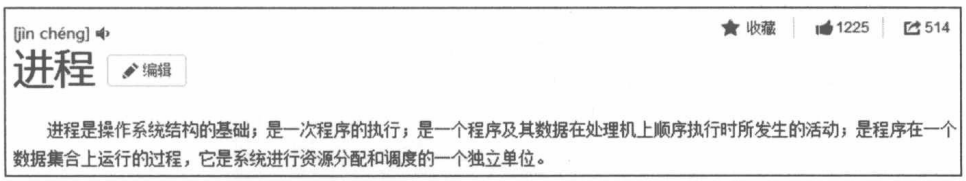


图 1-1 进程的定义

初看这段文字十分抽象，难以理解，那么再来看如图 1-2 所示的内容。

图 1-2 Windows 7 系统中的进程列表

难道一个正在操作系统中运行的 exe 程序可以理解成一个“进程”？没错！

通过查看“Windows 任务管理器”窗口中的列表，完全可以将运行在内存中的 exe 文件理解成进程——进程是受操作系统管理的基本运行单元。

程序是指令序列，这些指令可以让 CPU 完成指定的任务。*.java 程序经编译后形成 *.class 文件，在 Windows 中启动一个 JVM 虚拟机相当于创建了一个进程，在虚拟机中加载 class 文件并运行，在 class 文件中通过执行创建新线程的代码来执行具体的任务。创建测试用的代码如下：

```
public class Test1 {
    public static void main(String[] args) {
        try {
            Thread.sleep(Integer.MAX_VALUE);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

在没有运行这个类之前，任务管理器中以“j”开头的进程列表如图 1-3 所示。

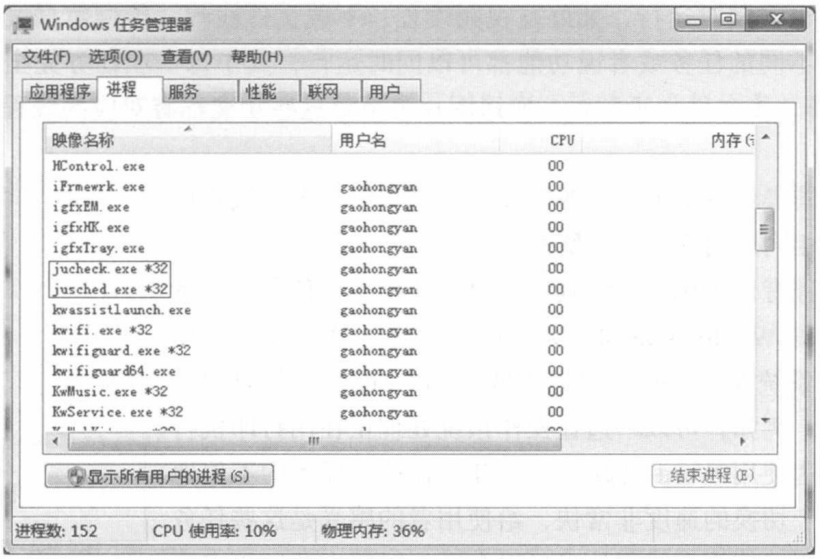


图 1-3　任务管理器中以“j”开头的进程

Test1 类重复运行 3 次后的进程列表如图 1-4 所示。可以看到，在任务管理器中创建了 3 个 javaw.exe 进程，说明每执行一次 main() 方法就创建一个进程，其本质上就是 JVM 虚拟机进程。

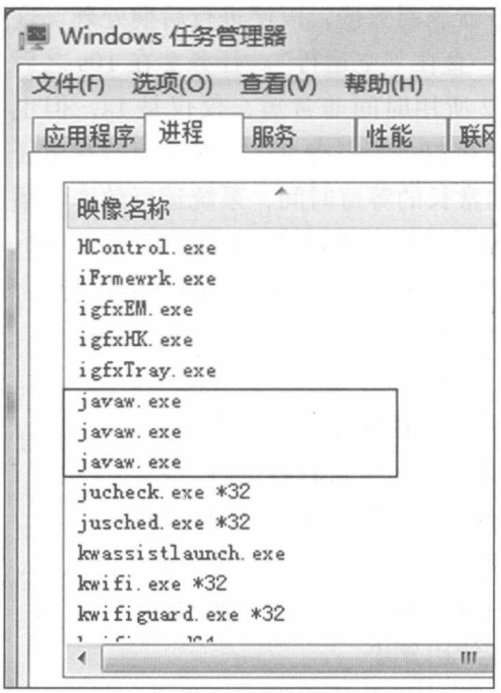


图 1-4　创建了 3 个 javaw.exe 进程

那什么是线程呢？线程可以理解为在进程中独立运行的子任务，例如，QQ.exe 运行时，

很多的子任务也在同时运行，如好友视频线程、下载文件线程、传输数据线程、发送表情线程等，这些不同的任务或者说功能都可以同时运行，其中每一项任务完全可以理解成是“线程”在工作，传文件、听音乐、发送图片表情等这些功能都有对应的线程在后台默默地运行。

进程负责向操作系统申请资源。在一个进程中，多个线程可以共享进程中相同的内存或文件资源。

使用多线程有什么优点呢？其实如果有使用“多任务操作系统”的经验，如 Windows 系列，大家应该都有这样的体会：使用多任务操作系统 Windows，可以大幅利用 CPU 的空闲时间来处理其他任务，例如，可以一边让操作系统处理正在用打印机打印的数据，一边使用 Word 编辑文档。CPU 在这些任务中不停地进行切换，由于切换的速度非常快，给使用者的感受是这些任务在同时运行，所以使用多线程技术可以在同一时间内执行更多不同的任务。

为了更加有效地理解多线程的优势，下面先来看如图 1-5 所示的单任务运行环境。

图 1-5 单任务运行环境

在图 1-5 中，任务 1 和任务 2 是两个完全独立、不相关的任务。任务 1 在等待远程服务器返回数据，以便进行后期处理，这时 CPU 一直呈等待状态，一直在“空运行”。任务 2 在 10s 之后被运行，虽然执行完任务 2 所用时间非常短，仅仅是 1s，但也必须等任务 1 运行结束后才可以运行任务 2，本程序运行在单任务环境中，所以任务 2 有非常长的等待时间，系统运行效率大幅降低。单任务的特点就是排队执行，即同步，就像在 cmd 中输入一条命令后，必须等待这条命令执行完才可以执行下一条命令。在同一时间只能执行一个任务，CPU 利用率大幅降低，这就是单任务运行环境的缺点。

多任务运行环境如图 1-6 所示。

在图 1-6 中，CPU 完全可以在任务 1 和任务 2 之间来回切换，使任务 2 不必等到 10s 之后再运行，系统和 CPU 的运行效率大大提升，这就是为什么要使用多线程技术、为什么要学习多线程。多任务的特点是在同一时间可以执行多个任务，这也是多线程技术的优点。使用多线程也就是在使用异步。

在通常情况下，单任务和多任务的实现与操作系统有关。例如，在一台计算机上使用同一个 CPU，安装 DOS 磁盘操作系统只能实现单任务运行环境，而

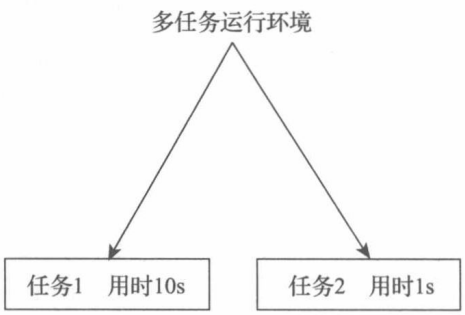


图 1-6 多任务运行环境

安装 Windows 操作系统则可以实现多任务运行环境。

在什么场景下使用多线程技术？笔者总结了两点。

1）阻塞。一旦系统中出现了阻塞现象，则可以根据实际情况来使用多线程技术提高运行效率。

2）依赖。业务分为两个执行过程，分别是 A 和 B。当 A 业务发生阻塞情况时，B 业务的执行不依赖 A 业务的执行结果，这时可以使用多线程技术来提高运行效率；如果 B 业务的执行依赖 A 业务的执行结果，则可以不使用多线程技术，按顺序进行业务的执行。

在实际的开发应用中，不要为了使用多线程而使用多线程，要根据实际场景决定。

注意　多线程是异步的，所以千万不要把 Eclipse 代码的顺序当作线程执行的顺序，线程被调用的时机是随机的。

1.2　使用多线程

想学习一个技术就要“接近”它，所以本节首先通过一个示例来接触一下线程。

一个进程正在运行时至少会有一个线程在运行，这种情况在 Java 中也是存在的，这些线程在后台默默地执行，例如，调用 public static void main() 方法的线程就是这样的，而且它由 JVM 创建。

创建示例项目 callMainMethodMainThread，并创建 Test.java 类，代码如下：

```
package test;

public class Test {

    public static void main(String[] args) {
        System.out.println(Thread.currentThread().getName());
    }

}
```

程序运行结果如图 1-7 所示。

在控制台输出的 main 其实就是一个名称为 main 的线程在执行 main() 方法中的代码。另外，需要说明一下，在控制台输出的 main 和 main 方法没有任何关系，它们仅仅是名字相同而已。

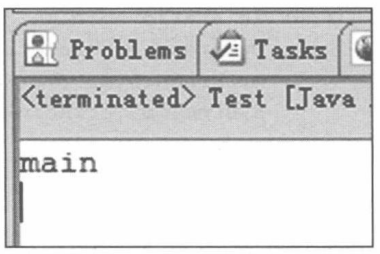


图 1-7　程序运行结果（主线程 main 出现）

1.2.1　继承 Thread 类

Java 的 JDK 开发包已经自带了对多线程技术的支持，通过它可以方便地进行多线程编程。实现多线程编程主要有两种方式：一种是继承

Thread 类，另一种是实现 Runnable 接口。

在学习如何创建新的线程前，先来看看 Thread 类的声明结构：

```
public class Thread implements Runnable
```

从上面的源代码中可以发现，Thread 类实现了 Runnable 接口，它们之间具有多态关系，多态结构的示例代码如下：

```
Runnable run1 = new Thread();
Runnable run2 = new MyThread();
Thread t1 = new MyThread();
```

其实使用继承 Thread 类的方式创建新线程时，最大的局限是不支持多继承，因为 Java 语言的特点是单根继承，所以为了支持多继承，完全可以实现 Runnable 接口，即一边实现一边继承，但这两种方式创建线程的功能是一样的，没有本质的区别。

本节主要介绍第一种方式。创建名称为 t1 的 Java 项目，创建一个自定义的线程类 MyThread.java，此类继承自 Thread，并且重写 run() 方法。在 run() 方法中添加线程要执行的任务代码如下：

```
package com.mythread.www;

public class MyThread extends Thread {
@Override
    public void run() {
        super.run();
        System.out.println("MyThread");
    }
}
```

运行类代码如下：

```
package test;

import com.mythread.www.MyThread;

public class Run {

    public static void main(String[] args) {
        MyThread mythread = new MyThread();
        mythread.start();//耗时大
        System.out.println("运行结束! ");//耗时小
    }

}
```

上面代码使用 start() 方法来启动一个线程，线程启动后会自动调用线程对象中的 run() 方法，run() 方法里面的代码就是线程对象要执行的任务，是线程执行任务的入口。

程序运行结果如图 1-8 所示。