Apress®

Java Threads and
the Concurrency Utilities

Java线程与并发编程实践

[美] Jeff Friesen 著
鄢倩 译

- Java线程API和并发工具的实用指南

中国工信出版集团
人民邮电出版社
POSTS & TELECOM PRESS

Java Threads and
the Concurrency Utilities

Java线程与并发编程实践

[美] Jeff Friesen 著
鄢倩 译

人民邮电出版社
北京

图书在版编目（CIP）数据

Java线程与并发编程实践 / （美）弗里森（Jeff Friesen）著 ; 鄢倩译. -- 北京 : 人民邮电出版社, 2017.2
ISBN 978-7-115-44036-5

Ⅰ. ①J… Ⅱ. ①弗… ②鄢… Ⅲ. ①JAVA语言－程序设计 Ⅳ. ①TP312.8

中国版本图书馆CIP数据核字(2016)第314919号

版权声明

◆ 著　[美] Jeff Friesen
译　鄢　倩
责任编辑　陈冀康
责任印制　焦志炜

◆ 人民邮电出版社出版发行　北京市丰台区成寿寺路 11 号
邮编　100164　电子邮件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
北京艺辉印刷有限公司印刷

◆ 开本：800×1000　1/16
印张：17
字数：208 千字　2017 年 2 月第 1 版
印数：1 – 3 000 册　2017 年 2 月北京第 1 次印刷
著作权合同登记号　图字：01-2016-5921 号

定价：55.00 元

读者服务热线：(010)81055410　印装质量热线：(010)81055316
反盗版热线：(010)81055315

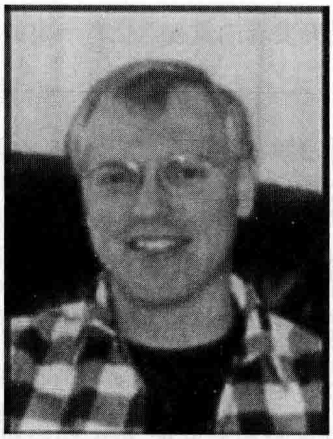
内容提要

Java 线程和并发工具是应用开发中的重要部分，备受开发者的重视，也有一定的学习难度。

本书是针对 Java 8 中的线程特性和并发工具的快速学习和实践指南。全书共 8 章，分别介绍了 Thread 类和 Runnable 接口、同步、等待和通知、线程组、定时器框架、并发工具、同步器、锁框架，以及高级并发工具等方面的主题。每章的末尾都以练习题的方式，帮助读者巩固所学的知识。附录 A 给出了所有练习题的解答，附录 B 给出了一个基于 Swing 线程的教程。

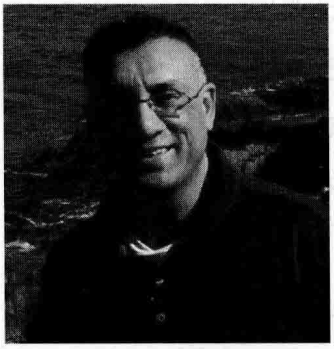
本书适合有一定基础的 Java 程序员阅读学习，尤其适合想要掌握 Java 线程和并发工具的读者阅读参考。

作者简介



Jeff Friesen 既是一名自由职业讲师，也是重点关注 Java 的软件开发者。除了是《Learn Java for Android Development》一书的作者以及《Android Recipes》一书的合著者，Jeff 还给 JavaWorld（JavaWorld.com）、informIT（InformIT.com）、Java.net 以及 DevSource（Devsource.com）写了大量关于 Java 及其他技术的文章。你可以通过他的个人网站 TutorTutor.ca 联系他。

技术审阅者简介



Sumit Pal 已经有超过 22 年的软件行业从业经验，在初创和企业级公司中担任过各类角色。他是大数据、可视化以及数据科学方面的咨询师、软件架构师，是大数据的热衷者，并且构建了端到端数据驱动分析系统。

在职业生涯的 22 年中，Sumit 陆续工作于 Microsoft（SQL Server 开发小组）、Oracle（OLAP 开发小组）以及 Verizon（大数据分析小组）公司。现在，他致力于提升各类客户的数据架构、大数据解决方案，并着手使用 Spark、Scala、Java 以及 Python 编码。他在构建从中间层、数据层到使用大数据以及 NoSQL 数据库完成可视化分析应用等可扩展系统领域有着广泛的经验。Sumit 对数据库中心、数据仓库、维度建模以及和 Java、Python，SQL 相关的数据科学方面有深刻的理解。

Sumit 拥有计算机科学的学士和硕士学位。

致谢

我要感谢在编写这本书的过程中帮助过我的所有人。尤其感谢邀请我写这本书的Steve Anglin以及写作过程中指导我的Mark Powers。

前言

线程和并发工具并非尤物，但是它们是正式应用的重要部分。本书会向你介绍 Java 8 Update 60 中线程特性以及并发工具的大部分内容。

第 1 章介绍了类 Thread 和接口 Runnable。你会学习如何创建 Thread 以及 Runnable 对象，获取和设置线程状态、启动线程、中断线程，将一条线程插入另外一条线程以及触发线程睡眠。

第 2 章关注同步。学习后你会解决一些问题，如没有同步就无法解决的竞态条件。你也能学到如何创建同步方法、块，以及如何使用忽略互斥访问的轻量级同步。

第 3 章探索等待和通知的重要话题。我们首先概览了类 Object 中支持这类概念的 API，并且通过生产者、消费者应用程序来演示这一 API，即一条线程产生条目而另一条线程消费之。

第 4 章介绍了之前 3 章没有覆盖的 3 个概念。首先，你会学习到线程组，可能想象不到它那么有用。其次，你会探索线程本地变量。最后，会学习到定时器框架，它简化了线程的定时任务。

前 4 章覆盖了低级的线程方法。第 5 章通过介绍并发工具切换至高级方法，它在简化多线程应用程序的同时也改善了性能。这一章之后还会介绍 executor、callable 以及 future。

第 6 章关注同步器（高级的同步构造）。你会学到倒计时门闩（countdown latch，一条或多条线程一直等在“门口”，直到另一条线程打开了这扇门，此时其他的线程才能继续执行）、同步屏障、交换器、信号量以及 Phaser。

第 7 章介绍了锁框架，它提供了大量针对锁及条件等待的接口和类，这种方式不同于对象原生的基于锁的同步以及 Object 的等待/通知机制。同时，它也做了一些诸如锁轮训方面的改善。

最后，第 8 章介绍了额外的并发工具，这部分内容在第 5 章到第 7 章没有涵盖。这章特别介绍了并发集合、原子变量、Fork/Join 框架以及 completion service。

每章都以各式各样的练习结尾，旨在帮助你掌握这些内容。除了问答题和判断题之外，你也经常遇到编程练习。附录 A 提供了这些问题的答案。

附录 B 提供了一个基于 Swing 线程的教程。你会学到 Swing 的单线程编程模型和大量 API，这些 API 避免了在图形上下文中使用额外线程导致的问题。你也会探索一个幻灯片应用程序，并且用这种好玩的方式结束本书。

注意： 我在某些例子中简洁地使用了 Java 8 中的 lambda 表达式，但是没有为此提供教程。你需要从其他地方获取这些知识。

注意： 你可以通过浏览器访问 www.apress.com/9781484216996，然后单击 Source Code 选项卡和紧跟其后的 Download Now 链接，来下载本书的源代码。

感谢你购买本书。我希望它对你在理解线程和并发工具上有所帮助。

——Jeff Friesen（2015 年 10 月）

目　录

第 1 部分　线程 API

第2部分 并发工具类

第3部分 附录

第 1 部分
线程 API

第 1 章

Thread 和 Runnable

Java 程序是通过线程执行的，线程在程序中具有独立的执行路径。当多条线程执行时，它们彼此之间的路径可以不同。举个例子，一条线程可能在执行 `switch` 语句的某个 `case` 分支，另一条线程很可能在执行其他 `case` 分支。

每个 Java 应用程序都有一个执行 `main()`函数的默认主线程。应用程序也可以创建线程在后台操作时间密集型任务，以确保对用户的响应。这些封装了代码执行序列的线程对象就被称为 runnable。

Java 虚拟机给每条线程分配独立的 JVM 栈空间以免彼此干扰。独立的栈使得线程可以追踪它们自己下一条要执行的指令，这些指令会依线程不同而有所区别。栈空间也为每条线程单独准备了一份方法参数、局部变量以及返回值的拷贝。

Java 主要是通过 `java.lang.Thread` 类以及 `java.lang.Runnable` 接口实现线程机制的。本章将介绍这些类型。

1.1 Thread 和 Runnable 简介

`Thread` 类为底层操作系统的线程体系架构提供一套统一接口（操作系统通常负责创建和管理线程）。单个的操作系统线程和一个 `Thread` 对象关联。

Runnable 接口为关联 Thread 对象的线程提供执行代码。这些代码放在 Runnable 的 void run()方法中，这个方法虽然不接收任何参数且没有返回值，但是有可能抛出异常，这点我们会在第 4 章讨论。

1.1.1 创建 Thread 和 Runnable 对象

除了默认主线程，线程都是通过创建合适的 Thread 和 Runnable 对象进入应用程序的。Thread 类声明了几个构造方法来初始化 Thread 对象。其中有几个构造方法需要接收 Runnable 对象作为参数。

我们有两种方式创建 Runnable 对象。第一种方式是创建一个实现了 Runnable 接口的匿名类，如下：

```
Runnable r = new Runnable()
             {
                 @Override
                 public void run()
                 {
                     // perform some work
                     System.out.println("Hello from thread");
                 }
             };
```

在 Java 8 之前，这是唯一一种创建 runnable 的方式。不过，Java 8 引入 lambda 表达式更为快捷地创建 runnable：

```
Runnable r = () -> System.out.println("Hello from thread");
```

lambda 确实比匿名类更简洁。我会在本章及随后的章节继续使用这些语言特性。

注意：一个 lambda 表达式是一个被传递到构造函数或者普通方法以供后续执行的匿名函数。和 Runnable 类似，lambda 表达式以 *functional interfaces*（声明单个抽象方法的接口）的形式工作。

创建 Runnable 对象之后，你可以把它传递到 Thread 类接收 Runnable 作为参数的构造函数中。举个例子，Thread(Runnable runnable)方法用指定的 runnable 初始化了一个新的 Thread 对象。下面的代码片段示范了这一做法：

```
Thread t = new Thread(r);
```

少数构造函数不会接收 Runnable 作为参数。例如，Thread()构造方法就不会接收它来初始化线程。你必须继承 Thread 类继而重写它的 run()方法（Thread 类实现了 Runnable 接口）并提供运行代码。如下面的代码片段所示：

```
class MyThread extends Thread
{
   @Override
   public void run()
   {
        // perform some work
        System.out.println("Hello from thread");
   }
}
// ...
MyThread mt = new MyThread();
```

1.1.2　获取和设置线程状态

一个 Thread 对象关联着一条线程的状态。这个状态由线程名称、线程存活的标识、线程的执行状态（是否正在执行？）、线程的优先级以及线程是否

为守护线程等标识构成。

1. 获取和设置线程的名称

每个 Thread 对象都会被赋予一个名称，这样有利于调试。这个名称如果不是显式指定的，那么默认会以一个 Thread-作为前缀。你可以通过调用 Thread 的 String getName()方法来获取这个名称。若要设置名称，则得把名称传递给一个合适的构造函数，如 Thread(Runnable r, String name)，或者调用 Thread 的 void setName(String name)方法。如下面的代码片段所示：

```
Thread t1 = new Thread(r, "thread t1");
System.out.println(t1.getName()); // Output: thread t1
Thread t2 = new Thread(r);
t2.setName("thread t2");
System.out.println(t2.getName()); // Output: thread t2
```

注意：Thread 的 long getId()方法会返回一个长整型的唯一名称。这个数字在线程的生命周期内不会改变。

2. 获取一条线程的存活状态

你可通过调用 Thread 的 boolean isAlive()方法来判断一条线程的存活状态。当线程是活的，该方法返回 true，反之，返回 false。一条线程的寿命仅仅起始于它真正在 start()方法（后面会讨论）中被启动起来，而结束于它刚刚离开 run()方法，此时线程死亡。下面的代码片段打印了一条新创建的线程的存活状态：

```
Thread t = new Thread(r);
System.out.println(t.isAlive()); // Output: false
```

3. 获取一条线程的执行状态

线程的执行状态由Thread.State枚举常量标识：

- NEW：该状态下线程还没有开始执行。
- RUNNABLE：该状态下线程正在JVM中执行。
- BLOCKED：该状态下线程被阻塞并等待一个监听锁（我会在第2章中介绍监听锁）。
- WAITING：该状态下线程无限期地等待另外一条线程执行特定的操作。
- TIMED_WAITING：该状态下线程在特定的时间内等待另外一条线程执行某种操作。
- TERMINATED：该状态下线程已经退出。

Thread通过提供Thread.State getState()方法来让应用程序判断线程的当前状态。示例如下：

```
Thread t = new Thread(r);
System.out.println(t.getState()); // Output: NEW
```

4. 获取和设置线程的优先级

当计算机有足够的处理器或处理内核，操作系统就会为每个处理器或核心分配单独的线程，这些线程可以同时执行。一旦计算机没有足够的处理器或核心的时候，多条线程只能轮转着使用共享的处理器和核心了。

注意：可以调用java.lang.Runtime类的int availableProcessors()方法，以确定JVM上可用的处理器或处理器核心的数量。方法的返回值可能会在JVM执行时发生变化，但是不会小于1。
