TURING 图灵程序设计丛书

Packt

Mastering Concurrency Programming with Java 9, Second Edition

精通Java并发编程

（第2版）

[西] 哈维尔 · 费尔南德斯 · 冈萨雷斯 著　唐富年 译

利用Java并发API强大功能，实现并发应用程序不费吹灰之力

中国工信出版集团

人民邮电出版社
POSTS & TELECOM PRESS

TURING 图灵程序设计丛书

Mastering Concurrency Programming
with Java 9, Second Edition

精通Java并发编程

（第2版）

[西] 哈维尔 · 费尔南德斯 · 冈萨雷斯 著
唐富年 译

人民邮电出版社
北 京

图书在版编目（CIP）数据

精通Java并发编程 : 第2版 / (西) 哈维尔·费尔南德斯·冈萨雷斯著 ; 唐富年译. -- 北京 : 人民邮电出版社, 2018.10
(图灵程序设计丛书)
ISBN 978-7-115-49166-4

Ⅰ. ①精… Ⅱ. ①哈… ②唐… Ⅲ. ①JAVA语言－程序设计 Ⅳ. ①TP312.8

中国版本图书馆CIP数据核字(2018)第190745号

内 容 提 要

Java 提供了一套非常强大的并发 API，可以轻松实现任何类型的并发应用程序。本书讲述 Java 并发 API 最重要的元素，包括执行器框架、Phaser 类、Fork/Join 框架、流 API、并发数据结构、同步机制，并展示如何在实际开发中使用它们。此外，本书还介绍了设计并发应用程序的方法论、设计模式、实现良好并发应用程序的提示和技巧、测试并发应用程序的工具和方法，以及如何使用面向 Java 虚拟机的其他编程语言实现并发应用程序。

本书适合 Java 开发人员阅读。

◆ 著　[西] 哈维尔·费尔南德斯·冈萨雷斯
译　唐富年
责任编辑　岳新欣
责任印制　周昇亮
◆ 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
三河市君旺印务有限公司印刷
◆ 开本：800×1000　1/16
印张：21.5
字数：508千字　2018年10月第1版
印数：1－3 000册　2018年10月河北第1次印刷
著作权合同登记号　图字：01-2017-8611号

定价：89.00元

读者服务热线：(010)51095186转600　印装质量热线：(010)81055316
反盗版热线：(010)81055315
广告经营许可证：京东工商广登字 20170147 号

译 者 序

Java是一门非常强大的编程语言，特色突出，性能卓越，几乎在你说得出名称的所有计算平台上，都或多或少会浮现出Java的影子。当初Sun公司在推出Java之际就将其作为一种开放式的编程语言，这无疑为Java注入了永久的生命力，也绝对是一个足以对人类社会进步产生重大影响的伟大决定。

Java并发API显然只是Java提供的一部分功能。然而到现在，在历经多次修改和丰富后，它已经强大到每个程序员都应予以高度重视的程度。在Java的每个版本中，并发API提供给程序员的功能都在增加。本书是近年来不可多得的一本专门介绍Java并发编程的图书，对于致力于Java大型程序设计、并行计算、分布式计算和大数据分析处理等方向的科研人员和工程人员来说，它值得一读。可以说本书是从并发处理的视角来探讨Java编程，也可以说是从Java的视角探讨并发处理。要阅读本书，需要预先了解Java语言的一些基础知识，需要有一些基本的Java程序设计经验，最好还了解一些并行计算或者数据处理的相关技术。译者不建议Java语言的初学者直接学习本书。

当2016年本书第1版（《精通Java 8并发编程》）刚出版时，图灵公司就敏锐洞察到本书的价值，并立即开始组织翻译工作。遗憾的是，此后不久官方就宣布了Java 9即将发布的消息。2017年，Java 9正式发布后，作者迅速推出了针对Java 9并发编程的第2版图书，因此原书第1版虽然已经翻译完成，但是最终没能跟读者见面。

在第2版中，作者修订了原书第1版的若干错误，更换了部分演示代码，增删了部分章节，使全书内容更具系统性，同时也增加和融入了Java 9的一些新特性。该书的主要特点如下。

- 第一，脉络清晰，内容全面。从执行器框架到流API，从并发数据结构到同步机制，从程序设计到调试测试，基本上所有与并发程序设计相关的内容都有所涉及。全书主线明晰，阅读起来比较轻松。
- 第二，语言通俗，举例充分。教科书式的语言相对较少，原理通俗易懂，实例简洁明了。几乎针对每个重要的知识点都提供了足够的代码示例，使得学习和练习都很方便。
- 第三，面向应用，便于上手。作者的视角并不是停留在并发编程本身，而是在于如何使用并发编程解决实际问题以及提高处理效能。读者不需要深陷于原理本身，宜结合实际各取所需，而且书中的示例也都很实用。

从第1版到第2版，本书的翻译过程冗长而艰苦，同时也让译者获益良多，但是限于译者水平，译文之中难免会出现一些错漏之处，敬请读者海涵。图灵公司的多位编辑在本书的翻译过程中给予了指导，为本书耗费了大量心血，在此一并表示感谢。

致 Nuria、Paula 和 Pelayo，感谢你们无限的关爱和耐心。

前　言

目前，计算机系统（以及其他相关系统，如平板电脑、智能手机等）可以让你同时执行多项任务。这是因为它们拥有并发的操作系统，能够同时控制多项任务。使用你最喜欢的编程语言中的并发 API，还能实现一个可以同时执行多项任务（读取文件、显示消息、读取网络上的数据）的应用程序。Java 提供了一套非常强大的并发 API，让你不费吹灰之力就可以实现任何类型的并发应用程序。在 Java 的每个版本中，该并发 API 提供给程序员的功能都有所增加。从 Java 8 开始，已经包含了流 API 以及一些便于实现并发应用程序的新方法和类。本书讲述了 Java 并发 API 最重要的元素，展示了如何在实际开发中使用它们。这些元素如下所示。

- 执行器框架，用于控制大量任务的执行。
- `Phaser` 类，用于执行可划分为多个阶段的任务。
- Fork/Join 框架，用于执行采用分治法解决问题的任务。
- 流 API，用于处理大型数据源，包括新的反应流。
- 并发数据结构，用于在并发应用程序中存储数据。
- 同步机制，用于组织并发任务。

此外，Java 并发 API 还包含更多内容，包括设计并发应用程序的方法论、设计模式、实现良好并发应用程序的提示和技巧、测试并发应用程序的工具和方法，以及采用其他面向 Java 虚拟机的语言（例如 Clojure、Groovy 和 Scala）实现并发应用程序的方法。

本书内容

第 1 章，“第一步：并发设计原理”。这一章将介绍并发应用程序的设计原理。你还将了解到并发应用程序可能出现的问题，以及设计并发应用程序的方法论，同时还会学到一些设计模式、提示和技巧。

第 2 章，“使用基本元素：`Thread` 和 `Runnable`”。这一章将解释如何采用 Java 语言中最基本的元素（`Runnable` 接口和 `Thread` 类）来实现并发应用程序。有了这些元素，你可以创建一个可与实际执行线程并行执行的新执行线程。

第 3 章，“管理大量线程：执行器”。这一章将介绍执行器框架的基本原理。该框架让你能够使用大量的线程，而无须创建或管理它们。你将实现 *k*-最近邻算法和一个基本的客户端/服务器应用程序。

第 4 章，“充分利用执行器”。这一章将探讨执行器的一些高级特性，包括为了在一段延迟之后或

每隔一定时间执行任务而进行的任务撤销和调度。你将实现一个高级客户端/服务器应用程序和一个新闻阅读器。

第 5 章，“从任务获取数据：Callable 接口与 Future 接口”。这一章将介绍如何在执行器中处理采用 Callable 与 Future 接口返回结果的任务。你将实现一个最佳匹配算法以及一个构建倒排索引的应用程序。

第 6 章，“运行分为多阶段的任务：Phaser 类”。这一章将介绍如何使用 Phaser 类来并发执行那些可分为多个阶段的任务。你将实现关键字抽取算法和遗传算法。

第 7 章，“优化分治解决方案：Fork/Join 框架”。这一章将介绍如何使用一种特殊的执行器，该执行器针对可以使用分治法解决的问题进行了优化，这就是 Fork/Join 框架及其工作窃取（work-stealing）算法。你将实现 k-means 聚类算法、数据筛选算法以及归并排序算法。

第 8 章，“使用并行流处理大规模数据集：MapReduce 模型”。这一章将介绍如何采用流来处理大规模数据集。你将学习如何使用流 API 和更多的流函数来实现 MapReduce 应用程序。你将实现一个数值汇总算法和一个信息检索工具。

第 9 章，“使用并行流处理大规模数据集：MapCollect 模型”。这一章将探讨如何使用流 API 中的 collect() 方法对数据流执行可变约简（mutable reduction）操作，将其转换为一种不同的数据结构，包括在 Collectors 类中预定义的一些收集器。你将实现一个无须建立索引就能够搜索数据的工具、一个推荐系统，以及计算社交网络中两个人的共同联系人列表的算法。

第 10 章，“异步流处理：反应流”。这一章将解释如何使用反应流来实现并发应用程序，而反应流则为带有非阻塞回压的异步流处理定义了标准。这种流的基本原理在官方网站的 Reactive Streams 介绍页面上有明确说明，而 Java 9 为其实现提供了必要的基础接口。

第 11 章，“探究并发数据结构和同步工具”。这一章将介绍如何使用最重要的并发数据结构（可用于并发应用程序而不会导致数据竞争条件的数据结构），以及 Java 并发 API 中用于组织任务执行的所有同步机制。

第 12 章，“测试与监视并发应用程序”。这一章将介绍如何获得 Java 并发 API 元素（线程、锁、执行器等）的状态信息。你还将学习如何使用 JConsole 应用程序来监视并发应用程序，以及如何使用 MultithreadedTC 库和 Java Pathfinder 应用程序来测试并发应用程序。

第 13 章，“JVM 中的并发处理：Clojure、带有 Gpars 库的 Groovy 以及 Scala”。这一章将介绍如何使用面向 Java 虚拟机的其他编程语言来实现并发应用程序。你将学习如何使用 Clojure、Scala 以及带有 Gpars 库的 Groovy 等编程语言所提供的并发元素。

阅读前提

要学习本书，你需要拥有 Java 编程语言的初中级知识，最好还对并发概念有基本的了解。

本书读者

如果你是了解并发编程基本原理的 Java 开发人员，同时又想成为 Java 并发 API 的专家型用户，

以便开发出能够充分利用计算机全部硬件资源的最优化应用程序，那么本书就非常适合你。

排版约定

在本书中，你会发现多种文本样式，用于区分不同种类的信息。下面是一些文本样式的例子，以及对这些样式含义的说明。

正文中的代码、数据库表名、用户输入等都采用如下样式："`modify()`方法并不是原子的，而`Account`类也不是线程安全的。"

代码段的样式如下。

```
public void task2() {
  section2_1();
  commonObject2.notify();
  commonObject1.wait();
  section2_2();
}
```

新术语和重点强调的内容都以黑体字表示。

此图标表示警告或重要说明。

此图标表示提示和技巧。

读者反馈

我们时刻欢迎你的反馈意见。这可以让我们了解你对本书的看法——喜欢什么或不喜欢什么。你的反馈对我们很重要，它可以帮助我们设计出真正让你受益良多的图书。请将一般性反馈意见直接发送至 feedback@packtpub.com，并在邮件的主题中注明书名。如果你对于某一主题有所专长，而且也有兴趣撰写或者参与编写一本书，请访问 www.packtpub.com/authors 查看我们的作者指南。

客户支持

现在，你已经是尊贵的 Packt 图书所有者了，我们通过以下方式使你的购买物有所值。

下载示例代码

你可以登录你的账户从 http://www.packtpub.com 下载本书的示例代码文件。如果你从其他地方购买了本书，可以访问 http://www.packtpub.com/support 并进行注册，我们会通过电子邮件将相关文件直接发送给你。可以通过以下步骤来下载代码文件。

(1) 使用你的电子邮件地址和密码登录网站或注册。

(2) 将鼠标放在顶部的 SUPPORT 选项卡上。

(3) 点击 Code Downloads & Errata 选项。

(4) 在 Search 框中输入书名。

(5) 选择要下载代码文件的那本书。

(6) 从下拉菜单中选择购书途径。

(7) 点击 Code Download 按钮。

下载文件之后，请确保使用下述工具的最新版本来解压或提取文件夹。

- WinRAR / 7-Zip（Windows）。
- Zipeg / iZip / UnRarX（Mac）。
- 7-Zip / PeaZip（Linux）。

GitHub 网站上也提供了本书配套的代码，可通过网址 https://github.com/PacktPublishing/Mastering-Concurrency-Programming-with-Java-9-Second-Edition 下载。通过网址 https://github.com/PacktPublishing/ 还可以获得我们各类图书和视频的配套代码。请检出它们供你使用吧！

勘误

尽管为确保内容的准确性，我们已经很谨慎，但是错误仍然在所难免。如果你在书中发现了任何文字或代码错误，请告知我们，我们将不胜感激。这样可以使其他读者免受同样的困惑，并能帮助我们改进本书的后续版本。如果你发现了任何错误，请访问 http://www.packtpub.com/submit-errata 将错误告知我们。[①]你需要在该页面上选定这本书，然后点击 Errata Submission Form 链接，输入勘误的详细内容。当勘误通过验证后，内容将被接受，而且该勘误信息将上传到我们的网站，或者添加到该书下面 Errata 部分的已有勘误表列表当中。要查看以前提交的勘误，可以访问 https://www.packtpub.com/books/content/support，在搜索栏中输入本书的名称。相关信息将出现在 Errata 部分中。

盗版问题

对所有媒体来说，互联网盗版都是一个长期存在的问题。在 Packt 公司，我们对自己的版权和许可证的保护非常严格。如果你在互联网上遇到以任何形式非法复制我们作品的行为，请立刻向我们提供具体地址或网站名称，以帮助我们采取补救措施。请通过 copyright@packtpub.com 联系我们，并且附上可疑盗版资料的链接。感谢你帮助我们保护作者，使我们能够带给你更有价值的内容。

其他问题

如果你对本书的任何方面还存有疑问，可以通过 questions@packtpub.com 邮箱联系我们，我们将尽力解决。

---

① 针对本书中文版的勘误，请到 http://www.ituring.com.cn/book/2018 查看和提交。——编者注

电子书

扫描如下二维码，即可购买本书电子版。

目　录

第 1 章　第一步：并发设计原理 …… 1
1.1　基本的并发概念 …… 1
1.1.1　并发与并行 …… 1
1.1.2　同步 …… 2
1.1.3　不可变对象 …… 2
1.1.4　原子操作和原子变量 …… 3
1.1.5　共享内存与消息传递 …… 3
1.2　并发应用程序中可能出现的问题 …… 3
1.2.1　数据竞争 …… 3
1.2.2　死锁 …… 4
1.2.3　活锁 …… 4
1.2.4　资源不足 …… 4
1.2.5　优先权反转 …… 5
1.3　设计并发算法的方法论 …… 5
1.3.1　起点：算法的一个串行版本 …… 5
1.3.2　第 1 步：分析 …… 5
1.3.3　第 2 步：设计 …… 5
1.3.4　第 3 步：实现 …… 6
1.3.5　第 4 步：测试 …… 6
1.3.6　第 5 步：调整 …… 6
1.3.7　结论 …… 7
1.4　Java 并发 API …… 8
1.4.1　基本并发类 …… 8
1.4.2　同步机制 …… 8
1.4.3　执行器 …… 9
1.4.4　Fork/Join 框架 …… 9
1.4.5　并行流 …… 9
1.4.6　并发数据结构 …… 9
1.5　并发设计模式 …… 10
1.5.1　信号模式 …… 10
1.5.2　会合模式 …… 11
1.5.3　互斥模式 …… 11
1.5.4　多元复用模式 …… 12
1.5.5　栅栏模式 …… 12
1.5.6　双重检查锁定模式 …… 12
1.5.7　读–写锁模式 …… 13
1.5.8　线程池模式 …… 14
1.5.9　线程局部存储模式 …… 14
1.6　设计并发算法的提示和技巧 …… 14
1.6.1　正确识别独立任务 …… 14
1.6.2　在尽可能高的层面上实施并发处理 …… 15
1.6.3　考虑伸缩性 …… 15
1.6.4　使用线程安全 API …… 15
1.6.5　绝不要假定执行顺序 …… 16
1.6.6　在静态和共享场合尽可能使用局部线程变量 …… 16
1.6.7　寻找更易于并行处理的算法版本 …… 17
1.6.8　尽可能使用不可变对象 …… 17
1.6.9　通过对锁排序来避免死锁 …… 17
1.6.10　使用原子变量代替同步 …… 18
1.6.11　占有锁的时间尽可能短 …… 19
1.6.12　谨慎使用延迟初始化 …… 19
1.6.13　避免在临界段中使用阻塞操作 …… 19
1.7　小结 …… 20

第 2 章　使用基本元素：Thread 和 Runnable …… 21
2.1　Java 中的线程 …… 21
2.1.1　Java 中的线程：特征和状态 …… 22
2.1.2　Thread 类和 Runnable 接口 …… 23
2.2　第一个例子：矩阵乘法 …… 24
2.2.1　公共类 …… 24
2.2.2　串行版本 …… 25

2.2.3 并行版本……25
2.3 第二个例子：文件搜索……32
2.3.1 公共类……32
2.3.2 串行版本……32
2.3.3 并发版本……33
2.3.4 对比解决方案……37
2.4 小结……38

第 3 章 管理大量线程：执行器……39
3.1 执行器简介……39
3.1.1 执行器的基本特征……39
3.1.2 执行器框架的基本组件……40
3.2 第一个例子：k-最近邻算法……40
3.2.1 k-最近邻算法：串行版本……41
3.2.2 k-最近邻算法：细粒度并发版本……42
3.2.3 k-最近邻算法：粗粒度并发版本……45
3.2.4 对比解决方案……46
3.3 第二个例子：客户端/服务器环境下的并发处理……48
3.3.1 客户端/服务器：串行版……48
3.3.2 客户端/服务器：并行版本……51
3.3.3 额外的并发服务器组件……54
3.3.4 对比两种解决方案……59
3.3.5 其他重要方法……61
3.4 小结……62

第 4 章 充分利用执行器……63
4.1 执行器的高级特性……63
4.1.1 任务的撤销……63
4.1.2 任务执行调度……64
4.1.3 重载执行器方法……64
4.1.4 更改一些初始化参数……64
4.2 第一个例子：高级服务器应用程序……65
4.2.1 ServerExecutor 类……65
4.2.2 命令类……70
4.2.3 服务器部件……72
4.2.4 客户端部件……78
4.3 第二个例子：执行周期性任务……79
4.3.1 公共部件……79
4.3.2 基础阅读器……81
4.3.3 高级阅读器……84
4.4 有关执行器的其他信息……87
4.5 小结……87

第 5 章 从任务获取数据：Callable 接口与 Future 接口……88
5.1 Callable 接口和 Future 接口简介……88
5.1.1 Callable 接口……88
5.1.2 Future 接口……89
5.2 第一个例子：单词最佳匹配算法……89
5.2.1 公共类……90
5.2.2 最佳匹配算法：串行版本……91
5.2.3 最佳匹配算法：第一个并发版本……92
5.2.4 最佳匹配算法：第二个并发版本……95
5.2.5 单词存在算法：串行版本……96
5.2.6 单词存在算法：并行版本……98
5.2.7 对比解决方案……100
5.3 第二个例子：为文档集创建倒排索引……102
5.3.1 公共类……103
5.3.2 串行版本……104
5.3.3 第一个并发版本：每个文档一个任务……105
5.3.4 第二个并发版本：每个任务多个文档……109
5.3.5 对比解决方案……112
5.3.6 其他相关方法……113
5.4 小结……113

第 6 章 运行分为多阶段的任务：Phaser 类……115
6.1 Phaser 类简介……115
6.1.1 参与者的注册与注销……116
6.1.2 同步阶段变更……116
6.1.3 其他功能……116
6.2 第一个例子：关键字抽取算法……117
6.2.1 公共类……118
6.2.2 串行版本……121
6.2.3 并发版本……123
6.2.4 对比两种解决方案……128
6.3 第二个例子：遗传算法……129

6.3.1 公共类……130
6.3.2 串行版本……132
6.3.3 并发版本……134
6.3.4 对比两种解决方案……139
6.4 小结……141

第 7 章 优化分治解决方案：Fork/Join 框架……142

7.1 Fork/Join 框架简介……142
7.1.1 Fork/Join 框架的基本特征……143
7.1.2 Fork/Join 框架的局限性……143
7.1.3 Fork/Join 框架的组件……144
7.2 第一个例子：k-means 聚类算法……144
7.2.1 公共类……145
7.2.2 串行版本……149
7.2.3 并发版本……151
7.2.4 对比解决方案……155
7.3 第二个例子：数据筛选算法……157
7.3.1 公共特性……157
7.3.2 串行版……157
7.3.3 并发版本……159
7.3.4 对比两个版本……165
7.4 第三个例子：归并排序算法……166
7.4.1 共享类……166
7.4.2 串行版本……167
7.4.3 并发版本……169
7.4.4 对比两个版本……172
7.5 Fork/Join 框架的其他方法……172
7.6 小结……173

第 8 章 使用并行流处理大规模数据集：MapReduce 模型……174

8.1 流的简介……174
8.1.1 流的基本特征……174
8.1.2 流的组成部分……175
8.1.3 MapReduce 与 MapCollect……177
8.2 第一个例子：数值综合分析应用程序…178
8.2.1 并发版本……178
8.2.2 串行版本……185
8.2.3 对比两个版本……186
8.3 第二个例子：信息检索工具……186
8.3.1 约简操作简介……187
8.3.2 第一种方式：全文档查询……188
8.3.3 第二种方式：约简的文档查询……191
8.3.4 第三种方式：生成一个含有结果的 HTML 文件……191
8.3.5 第四种方式：预先载入倒排索引……194
8.3.6 第五种方式：使用我们的执行器……195
8.3.7 从倒排索引获取数据：ConcurrentData 类……196
8.3.8 获取文件中的单词数……196
8.3.9 获取文件的平均 tfxidf 值……196
8.3.10 获取索引中的最大 tfxidf 值和最小 tfxidf 值……197
8.3.11 ConcurrentMain 类……198
8.3.12 串行版……199
8.3.13 对比两种解决方案……199
8.4 小结……202

第 9 章 使用并行流处理大规模数据集：MapCollect 模型……203

9.1 使用流收集数据……203
9.2 第一个例子：无索引条件下的数据搜索……205
9.2.1 基本类……205
9.2.2 第一种方式：基本搜索……207
9.2.3 第二种方式：高级搜索……209
9.2.4 本例的串行实现……211
9.2.5 对比实现方案……211
9.3 第二个例子：推荐系统……212
9.3.1 公共类……212
9.3.2 推荐系统：主类……213
9.3.3 ConcurrentLoader-Accumulator 类……215
9.3.4 串行版……216
9.3.5 对比两个版本……216
9.4 第三个例子：社交网络中的共同联系人……217
9.4.1 基本类……218
9.4.2 并发版本……219
9.4.3 串行版本……223
9.4.4 对比两个版本……223
9.5 小结……224

第 10 章 异步流处理：反应流 ····· 225
10.1 Java 反应流简介 ····· 225
10.1.1 Flow.Publisher 接口 ····· 226
10.1.2 Flow.Subscriber 接口 ····· 226
10.1.3 Flow.Subscription 接口 ··· 226
10.1.4 SubmissionPublisher 类 ··· 226
10.2 第一个例子：面向事件通知的集中式系统 ····· 227
10.2.1 Event 类 ····· 227
10.2.2 Producer 类 ····· 227
10.2.3 Consumer 类 ····· 228
10.2.4 Main 类 ····· 230
10.3 第二个例子：新闻系统 ····· 231
10.3.1 News 类 ····· 232
10.3.2 发布者相关的类 ····· 232
10.3.3 Consumer 类 ····· 235
10.3.4 Main 类 ····· 236
10.4 小结 ····· 238

第 11 章 探究并发数据结构和同步工具 ····· 240
11.1 并发数据结构 ····· 240
11.1.1 阻塞型数据结构和非阻塞型数据结构 ····· 241
11.1.2 并发数据结构 ····· 241
11.1.3 使用新特性 ····· 244
11.1.4 原子变量 ····· 251
11.1.5 变量句柄 ····· 252
11.2 同步机制 ····· 254
11.2.1 CommonTask 类 ····· 255
11.2.2 Lock 接口 ····· 255
11.2.3 Semaphore 类 ····· 256
11.2.4 CountDownLatch 类 ····· 258
11.2.5 CyclicBarrier 类 ····· 259
11.2.6 CompletableFuture 类 ····· 261
11.3 小结 ····· 268

第 12 章 测试与监视并发应用程序 ····· 269
12.1 监视并发对象 ····· 269
12.1.1 监视线程 ····· 269
12.1.2 监视锁 ····· 270
12.1.3 监视执行器 ····· 272
12.1.4 监视 Fork/Join 框架 ····· 273
12.1.5 监视 Phaser ····· 274
12.1.6 监视流 API ····· 275
12.2 监视并发应用程序 ····· 276
12.2.1 Overview 选项卡 ····· 278
12.2.2 Memory 选项卡 ····· 279
12.2.3 Threads 选项卡 ····· 280
12.2.4 Classes 选项卡 ····· 280
12.2.5 VM Summary 选项卡 ····· 281
12.2.6 MBeans 选项卡 ····· 283
12.2.7 About 选项卡 ····· 284
12.3 测试并发应用程序 ····· 284
12.3.1 使用 MultithreadedTC 测试并发应用程序 ····· 285
12.3.2 使用 Java Pathfinder 测试并发应用程序 ····· 288
12.4 小结 ····· 293

第 13 章 JVM 中的并发处理：Clojure、带有 GPars 库的 Groovy 以及 Scala ····· 294
13.1 Clojure 的并发处理 ····· 294
13.1.1 使用 Java 元素 ····· 295
13.1.2 引用类型 ····· 295
13.1.3 Ref 对象 ····· 298
13.1.4 Delay ····· 299
13.1.5 Future ····· 300
13.1.6 Promise ····· 301
13.2 Groovy 及其 GPars 库的并发处理 ····· 302
13.3 软件事务性内存 ····· 302
13.3.1 使用 Java 元素 ····· 302
13.3.2 数据并行处理 ····· 303
13.3.3 Fork/Join 处理 ····· 307
13.3.4 Actor ····· 308
13.3.5 Agent ····· 315
13.3.6 Dataflow ····· 316
13.4 Scala 的并发处理 ····· 322
13.4.1 Scala 中的 Future 对象 ····· 322
13.4.2 Promise ····· 328
13.5 小结 ····· 329

第1章 第一步：并发设计原理

计算机系统的用户总是希望自己的系统具有更好的性能。他们想要获得质量更高的视频、更好的视频游戏和更快的网络速度。几年前，提高处理器的速度可以为用户提供更好的性能。但是如今，处理器的速度并没有加快。取而代之的是，处理器增加了更多核心，这样操作系统就可以同时执行多个任务。这就是所谓的**并发处理**。并发编程涵盖了在一台计算机上同时运行多个任务或进程所需的所有工具和技术，以及任务或进程之间为消除数据丢失或不一致而进行的通信和同步。本章将探讨如下主题。

- 基本的并发概念。
- 并发应用程序中可能出现的问题。
- 设计并发算法的方法论。
- Java 并发 API。
- 并发设计模式。
- 设计并发算法的提示和技巧。

1.1 基本的并发概念

首先介绍一下并发的基本概念。要理解本书其余的内容，必须先理解这些概念。

1.1.1 并发与并行

并发和并行是非常相似的概念，不同的作者会给这两个概念下不同的定义。关于并发，最被人们认可的定义是，在单个处理器上采用单核执行多个任务即为并发。在这种情况下，操作系统的任务调度程序会很快从一个任务切换到另一个任务，因此看起来所有任务都是同时运行的。对于并行来说也有同样的定义：同一时间在不同的计算机、处理器或处理器核心上同时运行多个任务，就是所谓的“并行”。

另一个关于并发的定义是，在系统上同时运行多个任务（不同的任务）就是并发。而另一个关于并行的定义是：同时在某个数据集的不同部分之上运行同一任务的不同实例就是并行。

关于并行的最后一个定义是，系统中同时运行了多个任务。关于并发的最后一个定义是，一种解释程序员将任务和它们对共享资源的访问同步的不同技术和机制的方法。

正如你看到的，这两个概念非常相似，而且这种相似性随着多核处理器的发展也在不断增强。

1.1.2　同步

在并发中，我们可以将**同步**定义为一种协调两个或更多任务以获得预期结果的机制。同步方式有两种。

- **控制同步**：例如，当一个任务的开始依赖于另一个任务的结束时，第二个任务不能在第一个任务完成之前开始。
- **数据访问同步**：当两个或更多任务访问共享变量时，在任意时间里，只有一个任务可以访问该变量。

与同步密切相关的一个概念是**临界段**。临界段是一段代码，由于它可以访问共享资源，因此在任何给定时间内，只能够被一个任务执行。**互斥**是用来保证这一要求的机制，而且可以采用不同的方式来实现。

请记住，同步可以帮助你在完成并发任务的同时避免一些错误（本章稍后将详述），但是它也为你的算法引入了一些开销。你必须非常仔细地计算任务的数量，这些任务可以独立执行，而无须并行算法中的互通信。这就涉及并发算法的**粒度**。如果算法有着**粗粒度**（低互通信的大型任务），同步方面的开销就会较低。然而，也许你不会用到系统所有的核心。如果算法有着**细粒度**（高互通信的小型任务），同步方面的开销就会很高，而且该算法的吞吐量可能不会很好。

并发系统中有不同的同步机制。从理论角度来看，最流行的机制如下。

- **信号量**（semaphore）：一种用于控制对一个或多个单位资源进行访问的机制。它有一个用于存放可用资源数量的变量，并且可以采用两种原子操作来管理该变量的值。**互斥**（mutex，mutual exclusion 的简写形式）是一种特殊类型的信号量，它只能取两个值（即**资源空闲**和**资源忙**），而且只有将互斥设置为**忙**的那个进程才可以释放它。互斥可以通过保护临界段来帮助你避免出现竞争条件。
- **监视器**：一种在共享资源之上实现互斥的机制。它有一个互斥、一个条件变量、两种操作（等待条件和通报条件）。一旦你通报了该条件，在等待它的任务中只有一个会继续执行。

在本章中，你将要学习的与同步相关的最后一个概念是**线程安全**。如果共享数据的所有用户都受到同步机制的保护，那么代码（或方法、对象）就是**线程安全**的。数据的非阻塞的 CAS（compare-and-swap，比较和交换）原语是不可变的，这样就可以在并发应用程序中使用该代码而不会出任何问题。

1.1.3　不可变对象

不可变对象是一种非常特殊的对象。在其初始化后，不能修改其可视状态（其属性值）。如果想修改一个不可变对象，那么你就必须创建一个新的对象。

不可变对象的主要优点在于它是线程安全的。你可以在并发应用程序中使用它而不会出现任何问题。

不可变对象的一个例子就是 Java 中的 `String` 类。当你给一个 `String` 对象赋新值时，会创建一个新的 `String` 对象。

1.1.4 原子操作和原子变量

与应用程序的其他任务相比，**原子操作**是一种发生在瞬间的操作。在并发应用程序中，可以通过一个临界段来实现原子操作，以便对整个操作采用同步机制。

原子变量是一种通过原子操作来设置和获取其值的变量。可以使用某种同步机制来实现一个原子变量，或者也可以使用 CAS 以无锁方式来实现一个原子变量，而这种方式并不需要任何同步机制。

1.1.5 共享内存与消息传递

任务可以通过两种不同的方法来相互通信。第一种方法是**共享内存**，通常用于在同一台计算机上运行多任务的情况。任务在读取和写入值的时候使用相同的内存区域。为了避免出现问题，对该共享内存的访问必须在一个由同步机制保护的临界段内完成。

另一种同步机制是**消息传递**，通常用于在不同计算机上运行多任务的情形。当一个任务需要与另一个任务通信时，它会发送一个遵循预定义协议的消息。如果发送方保持阻塞并等待响应，那么该通信就是同步的；如果发送方在发送消息后继续执行自己的流程，那么该通信就是异步的。

1.2 并发应用程序中可能出现的问题

编写并发应用程序并不是一件容易的工作。如果不能正确使用同步机制，应用程序中的任务就会出现各种问题。本节将介绍一些此类问题。

1.2.1 数据竞争

如果有两个或者多个任务在临界段之外对一个共享变量进行写入操作，也就是说没有使用任何同步机制，那么应用程序可能存在**数据竞争**（也叫作**竞争条件**）。

在这些情况下，应用程序的最终结果可能取决于任务的执行顺序。请看下面的例子。

```
package com.packt.java.concurrency;

public class Account {

  private float balance;

  public void modify (float difference) {

    float value=this.balance;
    this.balance=value+difference;
  }

}
```

假设有两个不同的任务执行了同一个 `Account` 对象中的 `modify()` 方法。由于任务中语句的执行顺序不同，最终结果也会有所不同。假设初始余额为 1000，而且两个任务都调用了 `modify()` 方法并采用 1000 作为参数。最终的结果应该是 3000，但是如果两个任务都在同一时间执行了第一条语句，

然后又在同一时间执行了第二条语句，那么最终的结果将是2000。正如你看到的，`modify()`方法不是原子的，而`Account`类也不是线程安全的。

1.2.2　死锁

当两个（或多个）任务正在等待必须由另一线程释放的某个共享资源，而该线程又正在等待必须由前述任务之一释放的另一共享资源时，并发应用程序就出现了**死锁**。当系统中同时出现如下四种条件时，就会导致这种情形。我们将其称为Coffman条件。

- **互斥**：死锁中涉及的资源必须是不可共享的。一次只有一个任务可以使用该资源。
- **占有并等待条件**：一个任务在占有某一互斥的资源时又请求另一互斥的资源。当它在等待时，不会释放任何资源。
- **不可剥夺**：资源只能被那些持有它们的任务释放。
- **循环等待**：任务1正等待任务2所占有的资源，而任务2又正在等待任务3所占有的资源，以此类推，最终任务n又在等待由任务1所占有的资源，这样就出现了循环等待。

有一些机制可以用来避免死锁。

- **忽略它们**：这是最常用的机制。你可以假设自己的系统绝不会出现死锁，而如果发生死锁，结果就是你可以停止应用程序并且重新执行它。
- **检测**：系统中有一项专门分析系统状态的任务，可以检测是否发生了死锁。如果它检测到了死锁，可以采取一些措施来修复该问题，例如，结束某个任务或者强制释放某一资源。
- **预防**：如果你想防止系统出现死锁，就必须预防Coffman条件中的一条或多条出现。
- **规避**：如果你可以在某一任务执行之前得到该任务所使用资源的相关信息，那么死锁是可以规避的。当一个任务要开始执行时，你可以对系统中空闲的资源和任务所需的资源进行分析，这样就可以判断任务是否能够开始执行。

1.2.3　活锁

如果系统中有两个任务，它们总是因对方的行为而改变自己的状态，那么就出现了**活锁**。最终结果是它们陷入了状态变更的循环而无法继续向下执行。

例如，有两个任务：任务1和任务2，它们都需要用到两个资源：资源1和资源2。假设任务1对资源1加了一个锁，而任务2对资源2加了一个锁。当它们无法访问所需的资源时，就会释放自己的资源并且重新开始循环。这种情况可以无限地持续下去，所以这两个任务都不会结束自己的执行过程。

1.2.4　资源不足

当某个任务在系统中无法获取维持其继续执行所需的资源时，就会出现**资源不足**。当有多个任务在等待某一资源且该资源被释放时，系统需要选择下一个可以使用该资源的任务。如果你的系统中没有设计良好的算法，那么系统中有些线程很可能要为获取该资源而等待很长时间。

要解决这一问题就要确保**公平原则**。所有等待某一资源的任务必须在某一给定时间之内占有该资