HZ BOOKS
华章科技

The Pragmatic Programmers

Java并发编程领域的里程碑之作，资深Java技术专家、并发编程专家、敏捷开发专家和Jolt大奖得主撰写，Amazon五星畅销书

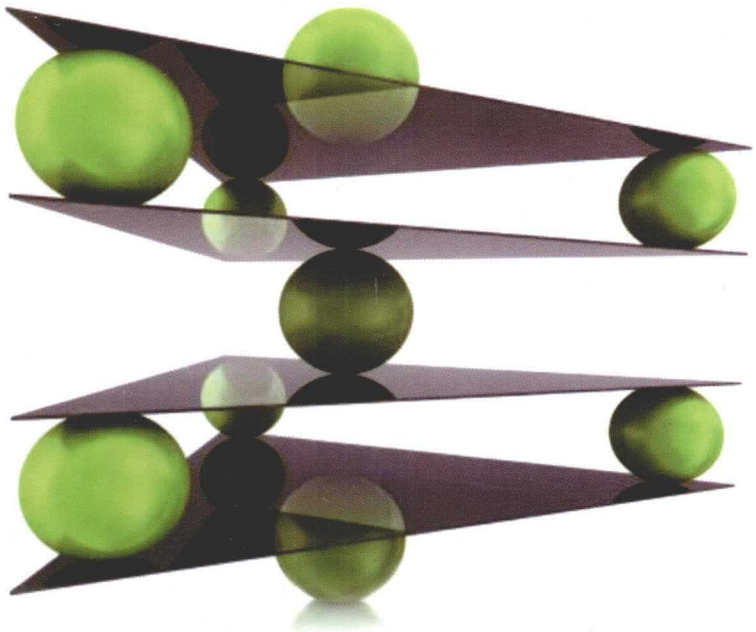
系统深入地讲解在JVM平台上如何利用JDK同步模型、软件事务内存模型和基于角色的并发模型进行并发编程，列举丰富示例，包含大量编程技巧和最佳实践

华章程序员书库

Programming Concurrency on the JVM
Mastering Synchronization, STM, and Actors

Java虚拟机并发编程

（美）Venkat Subramaniam 著
薛笛 译

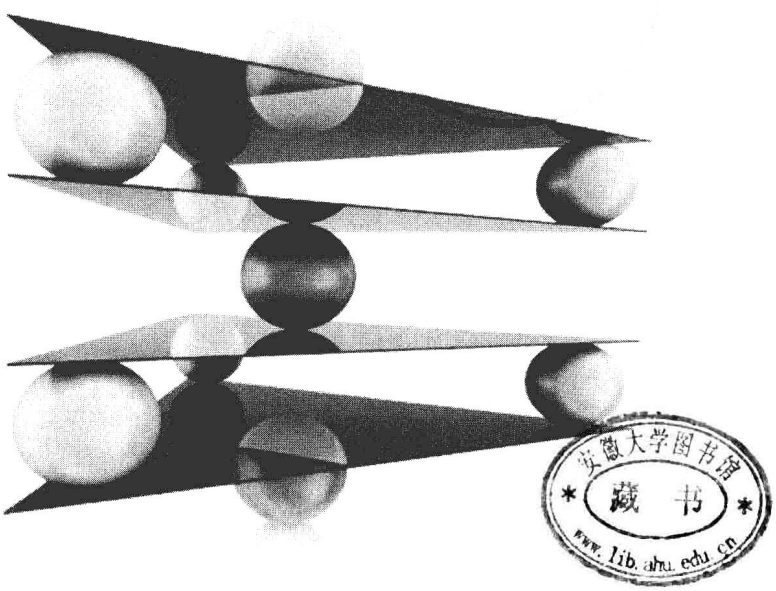


机械工业出版社
China Machine Press

Programming Concurrency on the JVM
Mastering Synchronization, STM, and Actors

Java虚拟机并发编程

（美）Venkat Subramaniam 著
薛笛 译



机械工业出版社
China Machine Press

图书在版编目（CIP）数据

Java 虚拟机并发编程 /（美）苏布拉马尼亚姆（Subramaniam, V.）著；薛笛译 . —北京：机械工业出版社，2013.4
（华章程序员书库）

书名原文：Programming Concurrency on the JVM：Mastering Synchronization, STM, and Actors

ISBN 978-7-111-41893-1

Ⅰ. J… Ⅱ. ①苏… ②薛… Ⅲ. JAVA 语言 – 程序设计 Ⅳ. TP312

中国版本图书馆 CIP 数据核字（2013）第 055404 号

本书版权登记号：图字：01-2011-7873

本书是 Java 并发编程领域的里程碑之作，由资深 Java 技术专家、并发编程专家、敏捷开发专家和 Jolt 大奖得主撰写，Amazon 五星级畅销书。它系统深入地讲解在 JVM 平台上如何利用 JDK 同步模型、软件事务内存模型和基于角色的并发模型更好地进行并发编程。全书以示例驱动，通俗易懂，包含大量编程技巧、注意事项和最佳实践。要重点强调的是，本书并不仅仅只适合于 Java 语言的并发编程，它还适用于 Clojure、Groovy、JRuby 和 Scala 等所有运行在 JVM 平台上的编程语言。

本书共 10 章，分为五个部分。第一部分：并发策略，阐释了影响并发性的因素、如何有效实现并发，以及并发的设计方法等；第二部分：现代 Java/JDK 并发，讨论了现代 Java API 的线程安全和效率，以及如何处理已有应用程序中的现实问题和重构遗留代码时的原则；第三部分：软件事务内存，深入讨论了 STM 并就如何在各种主要的 JVM 语言里使用 STM 给出了指导意见；第四部分：基于角色的并发，详细讲解了如何在基于角色的模型下消除并发问题以及如何在自己的首选语言中使用角色模型；第五部分：后记，回顾了本书讨论的解决方案并总结了并发编程中的注意事项和最佳实践。

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：秦 健

北京京师印务有限公司印刷

2013 年 5 月第 1 版第 1 次印刷

186mm × 240mm • 14.5 印张

标准书号：ISBN 978-7-111-41893-1

定 价：59.00 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）88378991 88361066 投稿热线：（010）88379604

购书热线：（010）68326294 88379649 68995259 读者信箱：hzjsj@hzbook.com

译 者 序

对于Java并发编程领域而言，JDK5的发布绝对具有里程碑式的意义。由该领域大师级人物Doug Lea亲自操刀的新并发API以及重新实现的并发容器使得开发人员摆脱synchronized、notify()、wait()这些原始的同步原语，为并发应用的开发提供了巨大的便利和性能提升，使Java并发编程前进到新的阶段。时过境迁，这几年来，虽然没有再出现类似于新并发API这样惊世骇俗的产品，但随着Java平台的不断演进，尤其在JVM开放了对动态语言的支持之后，一些新的语言及其背后所蕴含的设计方法和编程模型也被引入JVM，而这些新的设计方法和编程模型也深刻地影响着JVM并发编程领域的发展。

但是，出于对新语言的学习门槛、稳定性、性能、系统切换和维护成本等方面的顾虑，互联网企业，尤其是国内的大多数企业对这些新语言、新技术都持比较保守的态度，所以在国内我们很难接触到由这些新语言所引领的JVM并发编程的最新成果。当我们打开满是灰尘的并发编程工具箱，发现里面还是2007年起就一直在用的并发API，虽然用多了也感觉很顺手，但这些真的已经足够了吗？

本书为我们推开了一扇通往并发编程新领域的大门。作者以演讲般娓娓道来的方式告诉我们，其实我们一直都被共享可变性设计的缺陷折磨着却不自知。而我们仿佛是《黑客帝国》电影里生活的人们一样，在服下那粒可以看透真相的小药丸之前，总以为世界就应该是自己所熟悉的样子。再回到JVM并发编程领域，正是由于新的并发API太过成功，以致很多人已经形成了以新并发API为基础、以共享可变性设计为核心的固有套路和思维定式，认为这种方式所带来的饥饿、死锁、竞争条件等问题是并发编程固有的复杂性，从而隐忍和压抑着心中为解决这些问题而遭受的痛苦。从这个角度来说，作者很像《黑客帝国》中引领救世主Neo走出Matrix虚幻世界的导师墨菲斯，而本书就是那粒小药丸，吃了之后就可以看到JVM并发编程领域中不一样的风光，同时会感叹：哇哦，原来还有这么多新玩法！

相对于其他讲述并发编程的书籍，本书的亮点在于：

- **理论叙述深入浅出。**由于作者是一位出色的演讲者，所以本书的理论部分也大量运用了例证、类比等叙述手段，使得原本枯燥、生涩的理论叙述变得生动鲜活，从而更易于读者理解和接受。
- **实践内容极具实用性。**本书的内容涵盖了当前流行且成熟的多种JVM并发编程解决方案，包括Java7 Fork-Join API、STM、基于角色的模型，并且在每叙述完一种新模型之后都会有独立的一章专门用于展示前章所述模型在Clojure、Groovy、Java、JRuby和Scala中的用法和注意事项，有助于使用不同语言的读者都能找到相应的解决方案。虽然Java才是本书绝大多数国内读者的工作语言，但是上述几种语言对

于本书所述的并发模型的支持方面可谓各胜擅长，有很多在 Java 中实现起来语法很繁琐的模型在其他语言中可能几行代码就可以完成。所以，了解一下其他语言简洁优雅的实现方式，也不失为开阔眼界的好素材，以后再有类似需求的时候也可以多一种选择。

这里我要感谢本书的策划编辑杨福川，是他让我有机会参与本书的翻译工作，他的认可和鼓励为我完成本书的翻译提供了强大的助力。我还要感谢本书的责任编辑，他严谨、细致的专业态度让我受益匪浅。最后，我要感谢我的老婆璐璐，译书的这段时间里，我经常在电脑前敲键盘而无暇陪伴，感谢老婆的理解与支持。

本书所讲述的内容并不艰涩，领域也属我比较擅长的范畴，但想要将这位爱举例子的作者那很口语化、有点絮叨并且语法不那么严谨的英文转换成流畅且符合国人阅读习惯的文字着实不易。所以，虽然我很希望能够在人生第一部译作中力求完美，呈现给大家一部“很好读”的书，但终因时间和精力有限，译稿可能存在一些疏漏和翻译生硬之处，恳请各位读者批评指正。

前　言

除了咖啡因，我想没有什么能比写出一段执行速度飞快的代码更能令程序员们兴奋了。然而我们如何才能满足这种对计算速度的渴求呢？诚然，摩尔定律可以帮我们解决部分问题，但多核处理器才代表了未来真正的发展方向。为了能够充分发挥多核处理器的优势，我们每个程序员都应该熟悉并掌握并发编程的技能。

在一个并发程序中，两个或多个动作一般会同时发生。例如，一个并发程序可能需要一边下载多份文件、另一边还要执行一些计算任务以及数据库更新操作。在 Java 中，我们通常会使用线程来实现并发程序。虽然 Java 虚拟机在其设计之初就已经在语言层面支持了多线程，但并发编程的方法仍然在不断地进化和完善当中，而这正是我们在本书想要与你分享的内容。

并发编程的难点在于，我们既希望享受并发编程带来的好处，同时又不能让同步问题引火烧身。在 JVM 上的语言中，启动多个线程是很容易的，但它们彼此之间的执行顺序却是不可预知的。所以很快我们就将投身到一场为了协调线程并使其可以一致地处理数据而进行的战斗中去。

假设我们想要从 A 点出发快速抵达 B 点，根据旅行时间有多重要、有哪些可选的交通工具、预算是多少等因素，我们可以选择步行、坐巴士、自驾、坐和谐号或坐飞机飞过去。类似，为了提高 Java 代码的执行速度，我们同样也有很多选择。在 JVM 平台上，我们用得最多的一般是下面这三种并发模型：

- 称为“同步并受罪”的模型
- 软件事务内存（Software Transactional Memory，STM）模型
- 基于角色（actor-based）的并发模型

之所以将大家所熟知的 JDK 同步模型说成是“同步并受罪”，是因为如果我们忘记对共享可变状态进行同步或在错误的层级对其进行同步，所得到的结果将是不可预测的。如果足够幸运的话，我们还能在开发过程中捕获该问题；而如果我们没注意到这些问题，产品发布之后就会暴露出各种古怪且难以定位和重现的问题。而届时我们也无法从这些天杀的代码里找到任何编译错误、警告乃至任何简单的线索。

那些没有对共享可变状态的同步访问进行有效管理的程序本质上都是有问题的，但 Java 编译器却对此无动于衷。用纯 Java 对可变性进行编程就仿佛在和整天挑你毛病的丈母娘一起工作那样难受。相信你应该已经体会到这种痛苦了。

当我们撰写并发程序的时候，有如下三种方法可以帮助我们有效避免遇到并发所带来的问题：

- 在合适的地方进行正确的同步。

- 不共享状态。
- 不改变状态。

在使用现代 JDK 的并发 API 的时候，我们不得不花费大量精力来保证在合适的层级进行准确的同步。而 STM 则可以对用户隐藏同步操作并极大地减少出错几率。另一方面，基于角色的模型可以帮助我们避免使用共享状态，而避开共享状态正是我们赢得这场并发战役的秘密武器。

在本书中，我们将采用示例驱动的方法来学习上述三种模型以及如何利用这三种模型来更好地实现并发。

本书的目标读者

本书主要面向那些有一定经验的、正在使用诸如 Java、Clojure、Groovy、JRuby 和 Scala 这几种语言并且有兴趣进一步学习如何在 JVM 上管理和利用并发能力的 Java 程序员。

如果你是 Java 新手，那么本书将不会对你学习 Java 的基础功能有任何帮助。市面上有很多介绍 Java 编程基础的优秀教程可供选择，你可以先从这些书开始学起。

如果你已经在 JVM 平台上积累了相当的编程经验，但发觉还需要一些有助于进一步深入了解并发编程的资料，那么本书就是为你准备的。

如果你只对 Java 和 JDK 提供的解决方案，即 Java 线程和并发库感兴趣，那么我推荐你阅读两本非常优秀的专著，即 Brian Goetz 的《Java Concurrency in Practice》[Goe06] 和 Doug Lea 的《Concurrent Programming in Java》[Lea00]。这两本书在 Java 内存模型以及如何确保线程安全和一致性方面提供了非常丰富的信息。

本书旨在教会你如何使用 JDK 提供的方案来解决某些实际的并发问题，当然其中也会包含一些额外的技巧和方法。此外，你还将学到一些有助于更方便地实现隔离可变性的第三方 Java 库的知识。最后，你还将学到一些通过消除显式锁来降低复杂性和出错概率的类库等方面的内容。

本书意在帮助你学习一些目前已经比较成熟的工具和方法，以便你可以更方便地从中找出最适合解决你当前所面临的并发问题的方案。

本书的主要内容

本书将帮助你学习三种不同的并发解决方案，即现代 Java JDK 并发模型、软件事务内存（STM）和基于角色的并发模型。

本书主要分为 5 部分，即并发策略、现代 Java/JDK 并发模型、软件事务内存、基于角色的并发模型和后记。

在第 1 章中，我们将讨论是什么使得并发编程如此有用以及并发编程为何如此难以掌握。除此之外，该章还将对上面提到的三种并发模型进行简单介绍。

在深入研究上述三种并发解决方案之前，我们将在第 2 章先介绍一些影响程序并发性和加速效果的关键因素，并讨论如何实现有效并发的相关策略。

正如将在第 3 章中讨论的那样，我们采用的设计方法将产生两种截然不同的效果，即要么使我们在并发的海洋里乘风破浪尽情遨游，要么会让我们淹没其中连个水花都看不见。

自 Java 引入以来，其并发 API 的演进和增强一直非常迅速。所以第 4 章我们将讨论如何利用现代 Java API 来解决线程安全和并发性能问题。

鉴于我们十分渴望避免使用共享可变状态，所以在第 5 章我们将探讨一些解决现存应用程序实际问题的方法以及一些在重构历史遗留代码时所需要注意的问题。

第 6 章我们将深入探讨 STM，并学习如何使用 STM 来解决应用程序（特别那些读多写少的应用程序）中绝大部分的并发问题。

第 7 章我们将学习如何在几个不同的 JVM 语言中使用 STM。

第 8 章我们将学习到，如果我们依托隔离可变性进行设计，那么基于角色的模型就能够帮助我们彻底解决所有并发问题。

同样，如果你对几种不同的 JVM 语言都有兴趣，那么你将会在第 9 章学到如何在你偏爱的语言中使用基于角色的模型。

最后，我们将在第 10 章回顾本书前面所讨论过的所有解决方案并将其总结为一些简明扼要的知识点来帮助你更好地理解和实践。

并发还是并行

在业界，这两个术语其实并没有很明显差别，不同的人对于二者区别的阐述也都是各执一词（请不要就此问题并发地对他人提问……抑或我应该说不要并行提问）。

我们姑且把二者概念上的争论放到一边，先考虑一些更实际的问题。假设我们有一个运行在单核 CPU 机器上的多线程应用程序，在公司升级硬件的时候把该程序重新部署到了一台多核 CPU 的机器上。但其实无论底层硬件的部署方式如何变化，我们的代码本质上还是运行在一个独立的 JVM 进程上的，所以我们在开发过程中所要考虑问题也都基本相同，即如何创建和管理线程？如何保证数据完整性？如何处理锁和同步？以及我们创建的线程是否在合适的时间跨越内存栅栏？……

所以不管我们称之为并发也好还是并行也罢，能够真正解决问题才是保证程序正确、高效运行的关键。而这正是本书所关注的重点。

多语言程序员如何使用并发功能

Java 这个词在今天已经代表一个平台多过一门编程语言，而带有丰富类库的 JVM 已经逐渐演变成一个非常强大的平台。但与此同时，Java 语言却显得愈发老态龙钟。近些年来，在 JVM 平台上忽然涌现出了一批有趣且强大的语言，如 Clojure、JRuby、Grovvy 和 Scala。

在这些现代 JVM 语言中，部分语言如 Clojure、JRuby 和 Groovy 都是动态类型的，而其他如 Clojure 和 Scala 则深受函数式编程的影响。但这些语言都有一个共同的特点，那就是语法简洁且表达能力强。虽然学习这些语言的语法、范式或区别有一定的门槛，但学成之后我们可以用比 Java 更少的代码来实现相同的功能。更令人兴奋的是，我们可以将这些

JVM 语言的代码与 Java 代码混用，这样我们就成了名副其实的多语言程序员，请参阅附录 2 中 Neal Ford 有关“多语言程序员”的论述。

在本书中，我们将学习如何使用 java.util.concurrent API，以及如何通过 Akka 和 GPars 来使用 STM 和基于角色的模型。此外，我们还将学习如何在 Clojure、Java、JRuby、Groovy 和 Scala 中进行并发编程。如果你已经或打算使用其中任何一种语言，那么本书将会为你介绍在使用这些语言进行并发编程时所需要注意的各种相关事项。

示例和性能测评

虽然本书的大部分示例都是用 Java 实现的，但是你也会看到很多用 Clojure、Groovy、JRuby 和 Scala 实现的示例代码。在这些示例当中，我已尽量消除各语言之间的语法差异并尽可能避开语言专属的习惯用法，其目的主要是让大多数习惯使用 Java 的程序员更容易阅读和理解用其他语言实现的代码示例。

下面是本书所使用的语言和类库的版本汇总：

- Akka 1.1.3（http://akka.io/downloads）
- Clojure 1.2.1（http://clojure.org/downloads）
- Groovy 1.8（http://groovy.codehaus.org/Download）
- GPars 0.12（http://gpars.codehaus.org）
- Java SE 1.6（http://www.java.com/en/download）
- JRuby 1.6.2（http://jruby.org/download）
- Scala 2.9.0.1（http://www.scala-lang.org/downloads）

在对不同版本的示例代码的性能进行比较的时候，我会确保比较测试都在相同机器上完成。对于其中大部分示例，我采用的运行环境是搭载 2.8GHz 英特尔双核处理器、4GB 内存并预装了 Mac OS X 10.6.6 和 Java 1.6 update 24 的 MacBook Pro。而其他部分示例我用的则是搭载 8 核 Sunfire 2.33GHz 处理器、8GB 内存和预装 64 位 Windows XP 和 Java 1.6 的电脑。

除非另有说明，否则所有示例都是运行在服务器模式下的“Java HotSpot ™ 64-Bit Server VM”JVM 上。

所有示例全都可以在上面提到过的 Mac 和 Windows 机器上编译通过。

在代码示例的清单中，由于篇幅所限，我并没有列出 import 语句（以及 package 语句）。当你打算亲自运行代码示例的时候，如果你不确定某个类属于哪个 package 的话请别担心，因为我已经将完整的代码放到了网站上，你可以通过本书的网站 http://pragprog.com/titles/vspcom 下载全部示例的源代码。

致谢

有很多人为我完成本书的写作提供了巨大的帮助。这些年来，如果没有那些我所了解和敬重的伟大思想与我的个人思考之间的碰撞所激发出来的灵感，那么本书可能仍然只是

我脑海当中的一个念头而已。

首先我想感谢那些阅读了本书草稿并提供宝贵意见的审稿人，是他们使这本书变得更好。但是你在本书中所发现的任何错误则完全都是我的过失。

感 谢 Brian Goetz（@Brian Goetz）、Alex Miller（@puredanger） 和 Jonas Bonér（@jboner）几位审稿人，他们所提出的精辟评论使我受益匪浅。本书几乎每一页都被 Al cherer（@al_scherer）和 Scott Leberknight（@sleberknight）用鹰眼进行了彻底的检查和修订。非常感谢这些先生们。

这里我要特别感谢 Raju Gandhi（@looselytyped）、Ramamurthy Gopalakrishnan、Paul King（@paulk_asert）、Kurt Landrus（@koctya）、Ted Neward（@tedneward）、Chris Richardson（@crichardson）、Andreas Rueger、Nathaniel Schutta（@ntschutta）、Ken Sipe（@kensipe）以及 Matt Stine（@mstine），感谢他们花费大量宝贵时间帮我校对并给予我鼓励。感谢来自 Halloway（@stuarthalloway）的中肯评论，我已经根据他的一些建议对本书进行了改进。

我在各类 NFJS 会议上所进行的关于并发问题的演讲构成了本书的基本框架内容。所以在此感谢 NFJS（@nofluff）主管 Jay Zimmerman 给我在会议上进行演讲的机会，同时还要感谢与会的各位演讲嘉宾和参与者朋友们，与他们的交流和讨论使我深受启发。

在此要特别要感谢 Dave Briccetti（@dcbriccetti）、Frederik De Bleser（@enigmeta）、Andrei Dolganov、Rabea Gransberger、Alex Gout、Simon Sparks、Brian Tarbox、Michael Uren、Dale Visser 和 Tasos Zervos，以及那些花时间阅读本书 beta 版并在本书论坛中给予反馈的程序员。此外，Rabea Gransberger 那些见解深刻的评论、勘误以及观察评论使我受益匪浅。

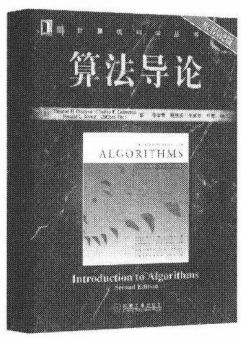
感谢那些我在本书所用到的以及在 JVM 上进行并发应用程序开发所依赖的那些语言和类库的开发者和参与者。

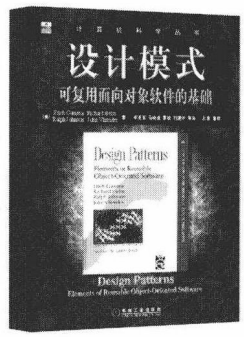
撰写本书的一个额外收获是使我越来越了解审阅本书第 1 版初稿的策划编辑 Steve Peter。他的幽默以及对细节的关注对本书的制作过程起到了极大的帮助。谢谢 Steve。很荣幸与本书的责任编辑 Brian P. Hogan（@bphogan）共同合作。他上手很快，给了我很多鼓励，同时还在许多需要改进的领域提出了很多建设性的意见和建议。谢谢 Brian。

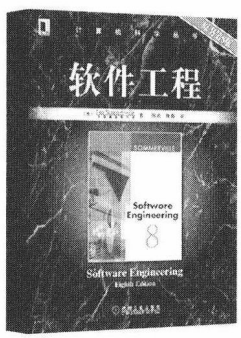
我要感谢整个 Pragmatic Bookshelf 团队一路以来的辛苦付出和给予我的鼓励。感谢 Kim Wimpsett、Susannah Pfalzer（@spfalzer）、Andy Hunt（@pragmaticandy）和 Dave Thomas（@pragdave）的帮助和指导，是他们使得成书过程变得如此有趣。

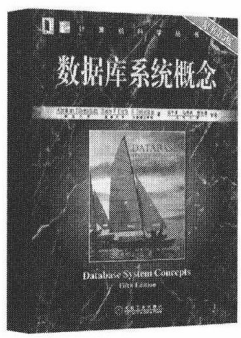
如果没有我妻子的支持，所有这一切都不可能完成，谢谢 Kavitha，感谢她为我付出的耐心与牺牲。此外，我从儿子 karthik 和 Krupa 那里也得到了很多鼓励，谢谢小伙子们，感谢他们总是好奇地问我是否已经把书写完了。最后，希望正在阅读本书的程序员能够将本书所学的知识更好地应用到未来工作中。谢谢！

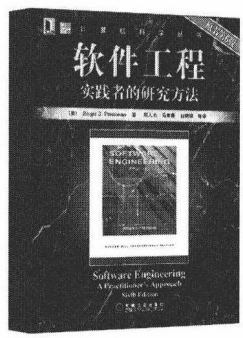
推荐阅读











算法导论（原书第2版）

2006、2007 CSDN、《程序员》杂志评选的十大IT好书之一　算法中的经典权威之作

编译原理（原书第2版）

编译领域无可替代的经典著作，被广大计算机专业人士誉为“龙书”

设计模式：可复用面向对象软件的基础

经典教材　权威之作

软件工程（原书第8版）

最受欢迎的软件工程指南

数据库系统概念（原书第5版）

数据库系统方面的经典教材，被美誉为“帆船书”

软件工程：实践者研究方法（原书第6版）

全球上百所大学和学院采用　最受欢迎的软件工程指南

推荐阅读

设计模式之禅

作者：秦小波 ISBN：978-7-111-29544-0 定价：69.00元

禅宗曰："教外别传，不立文字"，禅的境界本不该用文字来描述，言语也道不明白，但为了传道，悟道者仍要藉言语来说明。

何为禅？一种境界，一种体验，一种精神领域的最高修为。何为设计模式？对面向对象思想的深刻理解，对软件设计方法和编码经验的完美总结。

简单之美：软件开发实践者的思考

作者：倪健 ISBN：978-7-111-30103-5 定价：55.00元

多年以来，不管是从事一线的软件开发工作，还是从事管理工作，作者一直在思考这样一个问题：业界有这么多知识财富，可是在实践中真正能够被吸收和应用的却很少，这些知识财富的价值是毋庸置疑的，软件开发人员的热情和渴求也是毋庸置疑的，可问题究竟出在哪里呢？作者最后得出的结论是：这个问题要归结于思想和文化。

Java加密与解密的艺术

作者：梁栋 ISBN：978-7-111-29762-8 定价：69.00元

在如今这个信息化时代，数据是一切应用的核心和基础，有数据存在的地方就会有安全隐患，而密码学则是解决所有安全问题的银弹。

目　录

第一部分　并发策略

第二部分　现代 Java/JDK 并发模型

第三部分　软件事务内存

第四部分 基于角色的并发模型

第五部分 后记

第①章

并发的威力与风险

假设你已经跟老板立下军令状，说你会把新引进的那款强大的多核处理器变成一匹飞驰的骏马，拉着你们开发的应用程序呼啸奔驰。同时，你非常愿意立即着手把这股力量释放出来，并以此开发出响应更快、用户体验更好的应用程序来击败竞争对手。然而这些美好的愿望都被你同事的求助声打断了，他又碰到了一个很麻烦的同步问题。

绝大多数开发人员都对并发又爱又恨。

诚然并发编程是很困难的，但比起能获得的收益而言，所承受的折磨还是值得的。我们以可承受的代价所换来的这种处理能力是父辈们曾经梦寐以求的。只要能充分发挥出多核处理器的多任务并发执行能力，我们就可以创造出更棒的应用程序来。通过提前预判用户行为，我们就能够写出用户体验更好的应用程序。十几年前拖慢程序运行的那些功能现如今已经相当普及，我们不得不采用并发编程的实现方式才能使它们运行得更加流畅。

在本章中，我们将快速回顾一下并发编程技术的来龙去脉，并讨论在实践过程中可能遇到的一些风险。在本章末尾，我们将简单展望一下本书后面将会介绍的那些令人兴奋的并发编程技术。

1.1 线程：程序的执行流程

正如我们所熟知的那样，线程可以看成是进程中的一个执行流程。当我们运行一个程序的时候，其所属进程中至少存在一个执行线程。我们可以通过创建线程的方式来启动新的执行流程以达到并发执行多个任务的效果。此外，我们所使用的类库和框架可能也会根据其需要在后台启动额外的线程。

当多个线程在同一个应用程序或 JVM 实例下运行的时候，实际意味着此时有多个任务或操作在发运行。我们所说的并发程序通常就是指那些使用了多线程或多个并发执行流程的应用程序。

在一个只搭载了单核处理器的系统中，并发任务通常指多工（multiplexed）或多任务（multitasked）执行方式。也就是说，该单核处理器将会不断地在多个执行流程中进行上下文切换，但任意时刻有且只有一个线程（即执行流程）能够被执行。而在一个搭载了多核处理器的系统中，在任意时刻可以有多个执行流程（线程）被执行。其中，可以并发执行的线程数取决于处理器的可用内核数，而应用程序的并发线程数则取决于与该进程相关联的处理器内核数。

1.2 并发的威力

我们对并发编程如此感兴趣主要是基于如下两个原因：即并发编程可以帮助我们提高