HZ BOOKS
华章科技

PEARSON

华章专业开发者丛书

本书曾获Jolt大奖提名
JavaOne大会最畅销图书
了解Java并发编程必读佳作

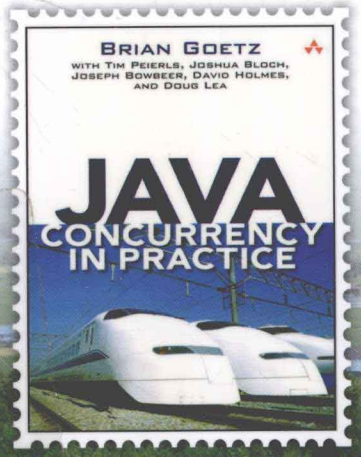
Java

并发编程实战

Java Concurrency in Practice

(美) Brian Goetz
Tim Peierls
Joshua Bloch
Joseph Bowbeer
David Holmes
Doug Lea
著

童云兰 等译



机械工业出版社
China Machine Press

华章专业开发者丛书

Java

并发编程实战

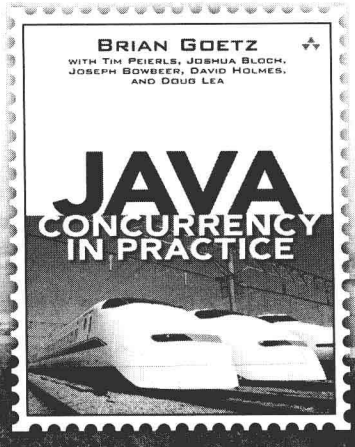
Java Concurrency in Practice

Brian Goetz
Tim Peierls
Joshua Bloch
Joseph Bowbeer
David Holmes
Doug Lea

（美）

著

童云兰 等译



机械工业出版社
China Machine Press

本书深入浅出地介绍了Java线程和并发，是一本完美的Java并发参考手册。书中从并发性和线程安全性的基本概念出发，介绍了如何使用类库提供的基本并发构建块，用于避免并发危险、构造线程安全的类及验证线程安全的规则，如何将小的线程安全类组合成更大的线程安全类，如何利用线程来提高并发应用程序的吞吐量，如何识别可并行执行的任务，如何提高单线程子系统的响应性，如何确保并发程序执行预期任务，如何提高并发代码的性能和可伸缩性等内容，最后介绍了一些高级主题，如显式锁、原子变量、非阻塞算法以及如何开发自定义的同步工具类。

本书适合Java程序开发人员阅读。

本书版权登记号：图字：01-2011-1513

图书在版编目（CIP）数据

Java并发编程实战 /（美）盖茨（Goetz, B.）等著；童云兰等译 . —北京：机械工业出版社，2012.2
（华章专业开发者丛书）

书名原文：Java Concurrency in Practice

ISBN 978-7-111-37004-8

I. J… II. ①盖… ②童… III. JAVA语言－程序设计 IV. TP312

中国版本图书馆CIP数据核字（2011）第281977号

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：关 敏

北京市荣盛彩色印刷有限公司印刷

2012年2月第1版第1次印刷

186mm×240mm • 19.5印张

标准书号：ISBN 978-7-111-37004-8

定价：69.00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

购书热线：（010）68326294；88376949；68995259

投稿热线：（010）88379604

读者信箱：hzjsj@hzbook.com

对本书的赞誉

“我曾有幸在一个伟大的团队中工作，参与设计和实现在 Java 5.0 和 Java 6 等平台中新增的并发功能。现在，仍然是这个团队，将透彻地讲解这些新功能，以及关于并发的一般性概念。并发已不再只是高级用户谈论的话题，每一位 Java 开发人员都应该阅读这本书。”

——Martin Buchholz，Sun 公司的 JDK 并发大师

“在过去 30 多年时间里，计算机性能一直遵循着摩尔定律，但从现在开始，它将遵循 Amdahl 定律。编写能高效利用多处理器的代码非常具有挑战性。 在这本书中介绍的一些概念和技术，对于在当前（以及未来的）系统上编写安全的和可伸缩的代码来说都是非常有用的。”

——Doron Rajwan，Intel 公司研究人员

“如果你正在编写、设计、调试、维护以及分析多线程的 Java 程序，那么本书正是你所需要的。如果你曾对某个方法进行过同步，但却不理解其中的原因，那么你以及你的用户都有必要从头至尾仔细地读一读这本书。”

——Ted Neward，《Effective Enterprise Java》的作者

“Brian 非常清晰地阐述了并发的一些基本问题与复杂性。对于使用线程并关注程序执行性能的开发人员来说，这是一本必读的书。”

——Kirk Pepperdine，JavaPerformanceTuning.com 网站 CTO

“本书深入浅出地介绍了一些复杂的编程主题，是一本完美的 Java 并发参考手册。书中的每一页都包含了程序员日常需要应对的问题（以及相应的解决方案）。随着摩尔定律的发展趋势由提高处理器核的速度转向增加处理器核的数量，如何有效地利用并发性已变得越来越重要，本书正好介绍了这些方面的内容。”

——Cliff Click 博士，Azul Systems 公司高级软件工程师

“我对并发有着浓厚的兴趣，并且与大多数程序员相比，我或许写过更多存在线程死锁的代码，也在同步上犯了更多的错误。在介绍 Java 线程和并发等主题的众多书籍中，Brian 的这本书最具可读性，它通过循序渐进的方式将一些复杂的主题阐述得很清楚。我将本书推荐给 Java Specialists' Newsletter 的所有读者，因为它不仅有趣，而且很有用，它介绍了当前 Java 开发人员正在面对的许多问题。”

——Heinz Kabutz 博士，Java Specialists' Newsletter 的维护者

“我一直努力想使一些简单的问题变得更简单，然而本书已经简化了一个复杂但却关键的主

题：并发。这本书采用了创新的讲解方法、简单明了的风格，它注定会成为一本非常重要的书。

——Bruce Tate，《Beyond Java》的作者

这本书为 Java 开发人员在线程编程领域提供了不可多得的知识。我在读这本书时受到了极大的启发，部分原因在于它详细地介绍了 Java 中并发领域的 API，但更重要的却在于这本书以一种透彻并且易懂的方式来介绍复杂的并发知识，这是其他书籍很难媲美的。

——Bill Venners，《Inside the Java Virtual Machine》的作者

译 者 序

并发编程是 Java 语言的重要特性之一，在 Java 平台上提供了许多基本的并发功能来辅助开发多线程应用程序。然而，这些相对底层的并发功能与上层应用程序的并发语义之间并不存在一种简单而直观的映射关系。因此，如何在 Java 并发应用程序中正确且高效地使用这些功能就成了 Java 开发人员的关注重点。

本书正是为了解决这个问题而写的。书中采用循序渐进的讲解方式，从并发编程的基本理论入手，逐步介绍了在设计 Java 并发程序时各种重要的设计原则、设计模式以及思维模式，同时辅以丰富的示例代码作为对照和补充，使得开发人员能够更快地领悟 Java 并发编程的要领，围绕着 Java 平台的基础并发功能快速地构建大规模的并发应用程序。

全书内容由浅入深，共分为四个部分。第一部分介绍了 Java 并发编程的基础理论，包括线程安全性与状态对象的基础知识，如何构造线程安全的类并将多个小型的线程安全类构建成更大型的线程安全类，以及 Java 平台库中的一些基础并发模块；第二部分介绍了并发应用程序的构造理论，包括应用程序中并行语义的分解及其与逻辑任务的映射，任务的取消与关闭等行为的实现，以及 Java 线程池中的一些高级功能，此外还介绍了如何提高 GUI 应用程序的响应性；第三部分介绍了并发编程的性能调优，包括如何避免活跃性问题，如何提高并发代码的性能和可伸缩性以获得理想的性能，以及在测试并发代码正确性和性能时的一些实用技术；第四部分介绍了 Java 并发编程中的一些高级主题，包括显式锁、原子变量、非阻塞算法以及如何开发自定义的同步工具类等。

本书的特点在于注重阐述并发技术背后的理论知识，对于每种技术的介绍不仅使读者能做到“知其然”，更能做到“知其所以然”。对于希望深入研究和探索 Java 并发编程的读者来说，本书是非常合适的。

参与本书翻译工作的还有李杨、吴汉平、徐光景、童胜汉、陈军、胡凯、刘红、张玮、陈红、李斌、李勇涛、王海涛、周云波、彭敏才、张世锋、朱介秋、宗敬、李静、叶锦、高波、熊莉、程凤、陈娟、胡世娟、董敏、谢路阳、冯卓、李志勇、胡欢、王进等。由于译者的时间和水平有限，翻译中的疏漏和错误在所难免，还望读者和同行不吝指正。

童云兰

2011 年 11 月于武汉

前　言

在写作本书时，对于中端桌面系统来说，多核处理器正变得越来越便宜。无独有偶，许多开发团队也注意到，在他们的项目中出现了越来越多与线程有关的错误报告。在 NetBeans 开发者网站上的最近一次公告中，一位核心维护人员注意到，为了修复与线程相关的问题，在某个类中竟然打了 14 次补丁。Dion Almaer，这位 TheServerSide 网站的前编辑，最近（在经过一番痛苦的调试过程并最终发现了一个与线程有关的错误之后）在其博客上写道，在大多数 Java 程序中充满了各种并发错误，使得程序只有在“偶然的情况下”才能正常工作。

确实，在开发、测试以及调试多线程程序时存在着巨大的困难，因为并发性错误通常并不会以某种确定的方式显现出来。当这些错误出现时，通常是在最糟糕的时刻，例如在正式产品中，或者在高负载的情况下。

当开发 Java 并发程序时，所要面对的挑战之一就是：平台提供的各种并发功能与开发人员在程序中需要的并发语义并不匹配。在 Java 语言中提供了一些底层机制，例如同步和条件等待，但在使用这些机制来实现应用级的协议与策略时必须始终保持一致。如果没有这些策略，那么在编写程序时，虽然程序看似能顺利地编译和运行，但却总会出现各种奇怪的问题。许多介绍并发的其他书籍更侧重于介绍一些底层机制和 API，而在设计级的策略和模式上叙述的不多。

Java 5.0 在 Java 并发应用程序的开发方面进展巨大，它不仅提供了一些新的高层组件，还补充了一些底层机制，从而使得无论是新手级开发人员还是专家级开发人员都能够更容易地构建并发应用程序。本书的作者都是 JCP 专家组的主要成员，也正是该专家组编写了这些新功能。本书不仅描述了这些新功能的行为和特性，还介绍了它们的底层设计模式和促使它们被添加到平台库中的应用场景。

我们的目标是向读者介绍一些设计规则和思维模式，从而使读者能够更容易也更乐意去构建正确的以及高性能的 Java 并发类和应用程序。

我们希望你能享受本书的阅读过程。

Brian Goetz
Williston，VT
2006 年 3 月

如何使用本书

为了解决在 Java 底层机制与设计级策略之间的不匹配问题，我们给出了一组简化的并发程序编写规则。专家看到这些规则会说：“嗯，这并不是完整的规则集。即使类 C 违背了规则 R，它仍然是线程安全的。”虽然在违背一些规则的情况下仍有可能编写出正确的并发程序，但这需

要对 Java 内存模型的底层细节有着深入的理解，而我们希望开发人员无须掌握这些细节就能编写出正确的并发程序。只要始终遵循这组简单的规则，就能编写出正确的并且可维护的并发程序。

我们假设读者对 Java 的基本并发机制已经有了一定程度的了解。本书并非是对并发的入门介绍——要了解这方面的内容，请参考其他书籍中有关线程的内容，例如《The Java Programming Language》（Arnold 等，2005）。此外，本书也不是介绍并发的百科全书——要了解这方面的内容，请参考《Concurrent Programming in Java》（Lea，2000）。事实上，本书提供了各种实用的设计规则，用于帮助开发人员创建安全的和高性能的并发类。在本书中相应的地方引用了以下书籍中的相关章节：《The Java Programming Language》、《Concurrent Programming in Java》、《The Java Language Specification》（Gosling 等，2005）以及《Effective Java》（Bloch，2001），并分别使用 [JPL n.m]、[CPJ n.m]、[JLS n.m] 和 [EJ Item n] 来表示它们。

在进行简要的介绍（第 1 章）之后，本书共分为四个部分：

基础知识。第一部分（第 2 章～第 5 章）重点介绍了并发性和线程安全性的基本概念，以及如何使用类库提供的基本并发构建块来构建线程安全类。在第一部分给出了一个清单，其中总结了这一部分中介绍的最重要的规则。

第 2 章与第 3 章构成了本书的基础。在这两章中给出了几乎所有用于避免并发危险、构造线程安全的类以及验证线程安全的规则。如果读者重“实践”而轻“理论”，那么可能会直接跳到第二部分，但在开始编写任何并发代码之前，一定要回来读一读这两章！

第 4 章介绍了如何将一些小的线程安全类组合成更大的线程安全类。第 5 章介绍了在平台库中提供的一些基础的并发构建模块，包括线程安全的容器类和同步工具类。

结构化并发应用程序。第二部分（第 6 章～第 9 章）介绍了如何利用线程来提高并发应用程序的吞吐量或响应性。第 6 章介绍了如何识别可并行执行的任务，以及如何在任务执行框架中执行它们。第 7 章介绍了如何使任务和线程在执行完正常工作之前提前结束。在健壮的并发应用程序与看似能正常工作的应用程序之间存在的重要差异之一就是，如何实现取消以及关闭等操作。第 8 章介绍了任务执行框架中的一些更高级特性。第 9 章介绍了如何提高单线程子系统的响应性。

活跃性、性能与测试。第三部分（第 10 章～第 12 章）介绍了如何确保并发程序执行预期的任务，以及如何获得理想的性能。第 10 章介绍了如何避免一些使程序无法执行下去的活跃性故障。第 11 章介绍了如何提高并发代码的性能和可伸缩性。第 12 章介绍了在测试并发代码的正确性和性能时可以采用的一些技术。

高级主题。第四部分（第 13 章～第 16 章）介绍了资深开发人员可能感兴趣的一些主题，包括：显式锁、原子变量、非阻塞算法以及如何开发自定义的同步工具类。

代码示例

虽然书中很多一般性的概念同样适用于 Java 5.0 之前的版本以及一些非 Java 的运行环境，但其中大多数示例代码（以及关于 Java 内存模型的所有描述）是基于 Java 5.0 或更高版本的，而且某些代码示例中还使用了 Java 6 的一些新增功能。

我们对书中的代码示例已经进行了压缩，以便减少代码量并重点突出与内容相关的部分。在本书的网站 http://www.javaconcurrencyinpractice.com 上提供了完整的代码示例、辅助示例以及勘误表。

代码示例可分为三类："好的"示例、"一般的"示例和"糟糕的"示例。"好的"示例是应该被效仿的技术。"糟糕的"示例是一定不能效仿的技术，而且还会用一个"Mr. Yuk"的图标[⊖]来表示该示例中的代码是"有害的"（参见程序清单 1）。"一般的"示例给出的技术并不一定是错的，但却是脆弱的、有风险的或是性能较差的，并且会用一个"Mr.Could Be Happier"图标来表示，如程序清单 2 所示。

程序清单 1　糟糕的链表排序方式（不要这样做）

```
public <T extends Comparable<? super T>> void sort(List<T> list) {
// 永远不要返回错误的答案!
    System.exit(0);
}
```

有些读者会质疑这些"糟糕的"示例在本书中的作用，毕竟，在一本书中应该给出如何做正确的事，而不是错误的事。这些"糟糕的"示例有两个目的，它们揭示了一些常见的缺陷，但更重要的是它们示范了如何分析程序的线程安全性，而要实现这个目的，最佳的方式就是观察线程安全性是如何被破坏的。

程序清单 2　非最优方式的链表排序

```
public <T extends Comparable<? super T>> void sort(List<T> list) {
    for (int i=0; i<1000000; i++)
        doNothing();
    Collections.sort(list);
}
```

致谢

本书诞生于 Java Community Process JSR 166 为 Java 5.0 开发 java.util.concurrent 包的过程中。还有许多人参与到 JSR 166 中，特别感谢 Martin Buchholz 将全部的工作融入到 JDK 中，并感谢 concurrency-interest 邮件列表中的所有读者对 API 草案提出的建议和反馈。

有不少来自各方面的人员都提出了建议和帮助，使得本书的内容得到了极大的充实。感谢 Dion Almaer、Tracy Bialik、Cindy Bloch、Martin Buchholz、Paul Christmann、Cliff Click、Stuart Halloway、David Hovemeyer、Jason Hunter、Michael Hunter、Jeremy Hylton、Heinz Kabutz、Robert Kuhar、Ramnivas Laddad、Jared Levy、Nicole Lewis、Victor Luchangco、Jeremy Manson、Paul Martin、Berna Massingill、Michael Maurer、Ted Neward、Kirk Pepperdine、Bill Pugh、Sam Pullara、Russ Rufer、Bill Scherer、Jeffrey Siegal、Bruce Tate、Gil Tene、Paul Tyma，以及硅谷模

⊖ Mr. Yuk 是匹兹堡儿童医院的注册商标，本书获得了该商标的授权，因此可以在本书中使用。

式小组的所有成员，他们通过各种技术交流为本书提供了指导建议，使得本书更加完善。

特别感谢 Cliff Biffle、Barry Hayes、Dawid Kurzyniec、Angelika Langer、Doron Rajwan 和 Bill Venners，他们非常仔细地审阅了本书的全稿，指出了代码示例中的错误，并提出了大量的改进建议。

感谢 Katrina Avery 的编辑工作，以及 Rosemary Simpson 在非常短的时间里完成了索引生成工作。感谢 Ami Dewar 绘制的插图。

感谢 Addison-Wesley 的全体成员，他们使本书得以最终问世。Ann Sellers 启动了编写本书的项目，Greg Doench 监督并帮助本书有条不紊地完成，Elizabeth Ryan 负责本书的出版过程。

此外还要感谢许许多多的软件工程师，他们开发了本书得以依赖的各种软件，这些软件包括 TEX、LATEX、Adobe Acrobat、pic、grap、Adobe Illustrator、Perl、Apache Ant、IntelliJ IDEA、GNU emacs、Subversion、TortoiseSVN，当然，还有 Java 平台及其类库。

目　录

第一部分　基础知识

第三部分 活跃性、性能与测试

第四部分 高级主题

第①章

简　介

编写正确的程序很难，而编写正确的并发程序则难上加难。与串行程序相比，在并发程序中存在更多容易出错的地方。那么，为什么还要编写并发程序？线程是 Java 语言中不可或缺的重要功能，它们能使复杂的异步代码变得更简单，从而极大地简化了复杂系统的开发。此外，要想充分发挥多处理器系统的强大计算能力，最简单的方式就是使用线程。随着处理器数量的持续增长，如何高效地使用并发正变得越来越重要。

1.1　并发简史

在早期的计算机中不包含操作系统，它们从头到尾只执行一个程序，并且这个程序能访问计算机中的所有资源。在这种裸机环境中，不仅很难编写和运行程序，而且每次只能运行一个程序，这对于昂贵并且稀有的计算机资源来说也是一种浪费。

操作系统的出现使得计算机每次能运行多个程序，并且不同的程序都在单独的进程中运行：操作系统为各个独立执行的进程分配各种资源，包括内存，文件句柄以及安全证书等。如果需要的话，在不同的进程之间可以通过一些粗粒度的通信机制来交换数据，包括：套接字、信号处理器、共享内存、信号量以及文件等。

之所以在计算机中加入操作系统来实现多个程序的同时执行，主要是基于以下原因：

资源利用率。在某些情况下，程序必须等待某个外部操作执行完成，例如输入操作或输出操作等，而在等待时程序无法执行其他任何工作。因此，如果在等待的同时可以运行另一个程序，那么无疑将提高资源的利用率。

公平性。不同的用户和程序对于计算机上的资源有着同等的使用权。一种高效的运行方式是通过粗粒度的时间分片（Time Slicing）使这些用户和程序能共享计算机资源，而不是由一个程序从头运行到尾，然后再启动下一个程序。

便利性。通常来说，在计算多个任务时，应该编写多个程序，每个程序执行一个任务并在必要时相互通信，这比只编写一个程序来计算所有任务更容易实现。

在早期的分时系统中，每个进程相当于一台虚拟的冯·诺依曼计算机，它拥有存储指令和数据的内存空间，根据机器语言的语义以串行方式执行指令，并通过一组 I/O 指令与外部设备通信。对每条被执行的指令，都有相应的“下一条指令”，程序中的控制流是按照指令集的规则来确定的。当前，几乎所有的主流编程语言都遵循这种串行编程模型，并且在这些语言的规范中也都清晰地定义了在某个动作完成之后需要执行的“下一个动作”。

串行编程模型的优势在于其直观性和简单性，因为它模仿了人类的工作方式：每次只做一

件事情，做完之后再做另一件。例如，首先起床，穿上睡衣，然后下楼，喝早茶。在编程语言中，这些现实世界中的动作可以被进一步抽象为一组粒度更细的动作。例如，喝早茶的动作可以被进一步细化为：打开橱柜，挑选喜欢的茶叶，将一些茶叶倒入杯中，看看茶壶中是否有足够的水，如果没有的话加些水，将茶壶放到火炉上，点燃火炉，然后等水烧开等等。在最后一步等水烧开的过程中包含了一定程度的异步性。当正在烧水时，你可以干等着，也可以做些其他事情，例如开始烤面包（这是另一个异步任务）或者看报纸，同时留意茶壶水是否烧开。茶壶和面包机的生产商都很清楚：用户通常会采用异步方式来使用他们的产品，因此当这些机器完成任务时都会发出声音提示。但凡做事高效的人，总能在串行性与异步性之间找到合理的平衡，对于程序来说同样如此。

这些促使进程出现的因素（资源利用率、公平性以及便利性等）同样也促使着线程的出现。线程允许在同一个进程中同时存在多个程序控制流。线程会共享进程范围内的资源，例如内存句柄和文件句柄，但每个线程都有各自的程序计数器（Program Counter）、栈以及局部变量等。线程还提供了一种直观的分解模式来充分利用多处理器系统中的硬件并行性，而在同一个程序中的多个线程也可以被同时调度到多个 CPU 上运行。

线程也被称为轻量级进程。在大多数现代操作系统中，都是以线程为基本的调度单位，而不是进程。如果没有明确的协同机制，那么线程将彼此独立执行。由于同一个进程中的所有线程都将共享进程的内存地址空间，因此这些线程都能访问相同的变量并在同一个堆上分配对象，这就需要实现一种比在进程间共享数据粒度更细的数据共享机制。如果没有明确的同步机制来协同对共享数据的访问，那么当一个线程正在使用某个变量时，另一个线程可能同时访问这个变量，这将造成不可预测的结果。

1.2 线程的优势

如果使用得当，线程可以有效地降低程序的开发和维护等成本，同时提升复杂应用程序的性能。线程能够将大部分的异步工作流转换成串行工作流，因此能更好地模拟人类的工作方式和交互方式。此外，线程还可以降低代码的复杂度，使代码更容易编写、阅读和维护。

在 GUI（Graphic User Interface，图形用户界面）应用程序中，线程可以提高用户界面的响应灵敏度，而在服务器应用程序中，可以提升资源利用率以及系统吞吐率。线程还可以简化 JVM 的实现，垃圾收集器通常在一个或多个专门的线程中运行。在许多重要的 Java 应用程序中，都在一定程度上用到了线程。

1.2.1 发挥多处理器的强大能力

过去，多处理器系统是非常昂贵和稀少的，通常只有在大型数据中心和科学计算设备中才会使用多处理器系统。但现在，多处理器系统正日益普及，并且价格也在不断地降低，即使在低端服务器和中端桌面系统中，通常也会采用多个处理器。这种趋势还将进一步加快，因为通过提高时钟频率来提升性能已变得越来越困难，处理器生产厂商都开始转而在单个芯片上放置多个处理器核。所有的主流芯片制造商都开始了这种转变，而我们也已经看到了在一些机器上