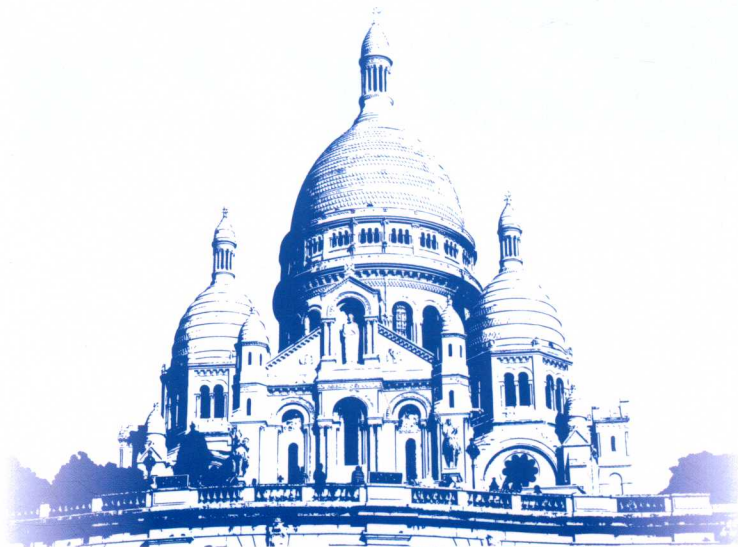



涵盖并行程序设计基础、思路、方法和实战
内容丰富，实例典型，实用性强

Broadview[®]
www.broadview.com.cn



实战 Java
高并发程序设计
(第2版)

葛一鸣 著

 中国工信出版集团

 电子工业出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

实战 Java 高并发程序设计

(第2版)

葛一鸣 著

電子工業出版社

Publishing House of Electronics Industry

内 容 简 介

在单核 CPU 时代，单任务在一个时间点只能执行单一程序，随着多核 CPU 的发展，并行程序开发变得尤为重要。

本书主要介绍基于 Java 的并行程序设计基础、思路、方法和实战。第一，立足于并发程序基础，详细介绍 Java 进行并行程序设计的基本方法。第二，进一步详细介绍了 JDK 对并行程序的强大支持，帮助读者快速、稳健地进行并行程序开发。第三，详细讨论了“锁”的优化和提高并行程序性能级别的方法和思路。第四，介绍了并行的基本设计模式，以及 Java 8/9/10 对并行程序的支持和改进。第五，介绍了高并发框架 Akka 的使用方法。第六，详细介绍了并行程序的调试方法。第七，分析 Jetty 代码并给出一些其在高并发优化方面的例子。

本书内容丰富，实例典型，实用性强，适合有一定 Java 基础的技术开发人员阅读。

未经许可，不得以任何方式复制或抄袭本书之部分或全部内容。
版权所有，侵权必究。

图书在版编目 (CIP) 数据

实战 Java 高并发程序设计 / 葛一鸣著. —2 版. —北京: 电子工业出版社, 2018.9
ISBN 978-7-121-35003-0

I. ①实… II. ①葛… 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 207736 号

策划编辑: 董 英

责任编辑: 汪达文

印 刷: 三河市良远印务有限公司

装 订: 三河市良远印务有限公司

出版发行: 电子工业出版社

北京市海淀区万寿路 173 信箱

邮编: 100036

开 本: 787×980 1/16 印张: 26

字数: 525 千字

版 次: 2015 年 10 月第 1 版

2018 年 9 月第 2 版

印 次: 2018 年 9 月第 1 次印刷

印 数: 3000 册 定价: 89.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：(010) 88254888, 88258888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

本书咨询联系方式：010-51260888-819, faq@phei.com.cn。

前 言

关于 Java 与并行

由于单核 CPU 的主频逐步逼近极限，多核 CPU 架构成了一种必然的技术趋势，因此多线程并行程序便显得越来越重要。并行计算的一个重要应用场景就是服务端编程。目前服务端 CPU 的核心数已经轻松超越 10 个，而 Java 显然已经成为当下最流行的服务端编程语言，且已经更新到 JDK 10，因此熟悉和了解基于 Java 的并行程序开发有着重要的实用价值。

本书的体系结构

本书立足于实际开发，又不缺乏理论介绍，力求通俗易懂、循序渐进。本书共分为 9 章。

第 1 章主要介绍了并行计算中相关的一些基本概念，树立读者对并行计算的基本认识，介绍了两个重要的并行性能评估定律，以及 Java 内存模型 JMM。

第 2 章介绍了 Java 并行程序开发的基础，包括 Java 中 Thread 的基本使用方法等，也详细介绍了并行程序容易引发的一些错误，以及容易出现的误用。

第 3 章介绍了 JDK 内部对并行程序开发的支持，主要介绍 juc (java.util.concurrent) 中一些工具的使用方法、各自的特点及它们的内部实现原理。

第 4 章介绍了在开发过程中可以进行的对锁的优化，也进一步简要描述了 Java 虚拟机层面对并程序优化支持。此外，还花费一定的篇幅介绍了无锁的计算。

第 5 章介绍了并程序设计中常见的一些设计模式，以及一些典型的并行算法和使用方法，其中包括重要的 Java NIO 和 AIO 的介绍。

第 6 章介绍了 Java 8/9/10 为并行计算做的改进，包括并行流、CompletableFuture、StampedLock、LongAdder，以及发布和订阅模式等。

第 7 章主要介绍了高并发框架 Akka 的基本使用方法，并使用 Akka 框架实现了一个简单的粒子群算法，模拟超高并发的场景。

第 8 章介绍了使用 Eclipse 进行多线程调试的方法，并演示了通过 Eclipse 进行多线程调试重现 ArrayList 的线程不安全问题。

第 9 章介绍了 Jetty，并分析了 Jetty 的一些关键代码，主要展示它在高并发优化中所做的一些努力，也为读者学会并深入理解高并发带来一些提示和思考。

本书特色

本书的主要特色如下。

1. **结构清晰**。本书一共 9 章，总体上循序渐进，逐步提升。每一章都有鲜明的侧重点，有利于读者快速抓住重点。
2. **理论结合实战**。本书注重实战，书中重要的知识点都安排了代码实例，帮助读者理解。同时也不忘对系统的内部实现原理进行深度剖析。
3. **通俗易懂**。本书尽量避免采用过于理论化的描述方式，简单的白话文风格贯穿全书，配图基本上为手工绘制，降低了理解难度，并尽量做到读者在阅读过程中少盲点、无盲点。

适合阅读人群

虽然本书力求通俗，但是要通读本书并取得良好的学习效果，要求读者具备基本的 Java

知识或者一定的编程经验。因此，本书适合以下读者。

- 拥有一定开发经验的 Java 平台开发人员（Java、Scala、JRuby 等）。
- 软件设计师、架构师。
- 系统调优人员。
- 有一定的 Java 编程基础并希望进一步加深对并行程序的理解的研发人员。

本书的约定和更新

本书在叙述过程中，有如下约定。

- 本书所述的 JDK 1.5、JDK 1.6、JDK 1.7、JDK 1.8、JDK 1.9、JDK 1.10 分别等同于 JDK 5、JDK 6、JDK 7、JDK 8、JDK 9、JDK 10。
- 如无特殊说明，本书的程序、示例均在 JDK 1.8 以上环境中运行。

相较前一版，本书的主要更新如下。

1. 第 3 章增加的内容如下。

- 使用 JMH 进行性能测试。
- CopyOnWriteArrayList ConcurrentLinkedQueue 性能测试。
- 使用 Guava 的 RateLimiter 限流。
- Guava 中对线程池的扩展。
- 介绍 JDK 中 ArrayBlockingQueue 的算法。

2. 第 5 章增加的内容如下。

- Guava 对 Future 模式的支持。

3. 第 6 章增加的内容如下。

- 支持 timeout 的 CompletableFuture。
- ConcurrentHashMap 在新版本 JDK 中的增强。
- 发布和订阅模式。

4. 增加第 9 章，主要介绍 Jetty 多线程优化。从细节入手分析 Jetty 在多线程并发方面做出的努力和优化，对实践应用具有极强的参考价值。

联系作者

本书的写作过程远比我想象得艰辛，为了让全书能够更清楚、更准确地表达和论述，我经历了很多个不眠之夜，即使现在回想起来，我也忍不住会打个冷战。由于写作水平的限制，书中难免会有不妥之处，望读者谅解。

为此，如果读者有任何疑问或者建议，非常欢迎大家加入 QQ 群 254693571，一起探讨学习中的困难、分享学习经验，我期待与大家一起交流、共同进步。同时，大家也可以关注我的博客 <http://www.uucode.net/>。欢迎大家去博文视点社区下载本书推荐的参考文献。

感谢

本书能够面世，是因为得到了众人的支持。首先，要感谢我的妻子，她始终不辞辛劳、毫无怨言地对我照顾有加，才让我得以腾出大量时间安心工作。其次，要感谢所有编辑为我一次又一次地审稿改错，批评指正，促进本书逐步完善。最后，感谢我的母亲三十年如一日对我的体贴和关心。

葛一鸣

读者服务

轻松注册成为博文视点社区用户（www.broadview.com.cn），您即可享受以下服务：

- **下载资源：**本书如提供示例代码及资源文件，均可在 [下载资源](#) 处下载。
- **提交勘误：**您对书中内容的修改意见可在 [提交勘误](#) 处提交，若被采纳，将获赠博文视点社区积分（在您购买电子书时，积分可用来抵扣相应金额）。
- **与作者交流：**在页面下方 [读者评论](#) 处留下您的疑问或观点，与作者和其他读者一同学习交流。

页面入口：<http://www.broadview.com.cn/35003>



目 录

第 1 章 走入并行世界	1
1.1 何去何从的并行计算	1
1.1.1 忘掉那该死的并行	2
1.1.2 可怕现实：摩尔定律的失效	4
1.1.3 柳暗花明：不断地前进	5
1.1.4 光明或是黑暗	6
1.2 你必须知道的几个概念	7
1.2.1 同步（Synchronous）和异步（Asynchronous）	7
1.2.2 并发（Concurrency）和并行（Parallelism）	8
1.2.3 临界区	9
1.2.4 阻塞（Blocking）和非阻塞（Non-Blocking）	9
1.2.5 死锁（Deadlock）、饥饿（Starvation）和活锁（Livelock）	10
1.3 并发级别	11
1.3.1 阻塞	11
1.3.2 无饥饿（Starvation-Free）	11
1.3.3 无障碍（Obstruction-Free）	12
1.3.4 无锁（Lock-Free）	13
1.3.5 无等待（Wait-Free）	13

1.4	有关并行的两个重要定律	14
1.4.1	Amdahl 定律	14
1.4.2	Gustafson 定律	16
1.4.3	是否相互矛盾	17
1.5	回到 Java: JMM	18
1.5.1	原子性 (Atomicity)	18
1.5.2	可见性 (Visibility)	20
1.5.3	有序性 (Ordering)	22
1.5.4	哪些指令不能重排: Happen-Before 规则	27
第 2 章	Java 并行程序基础	29
2.1	有关线程你必须知道的事	29
2.2	初始线程: 线程的基本操作	32
2.2.1	新建线程	32
2.2.2	终止线程	34
2.2.3	线程中断	38
2.2.4	等待 (wait) 和通知 (notify)	41
2.2.5	挂起 (suspend) 和继续执行 (resume) 线程	45
2.2.6	等待线程结束 (join) 和谦让 (yield)	49
2.3	volatile 与 Java 内存模型 (JMM)	50
2.4	分门别类的管理: 线程组	53
2.5	驻守后台: 守护线程 (Daemon)	54
2.6	先做重要的事: 线程优先级	56
2.7	线程安全的概念与关键字 synchronized	57
2.8	程序中的幽灵: 隐蔽的错误	61
2.8.1	无提示的错误案例	62
2.8.2	并发下的 ArrayList	63
2.8.3	并发下诡异的 HashMap	64
2.8.4	初学者常见的问题: 错误的加锁	67

第 3 章 JDK 并发包.....	71
3.1 多线程的团队协作：同步控制.....	71
3.1.1 关键字 synchronized 的功能扩展：重入锁.....	72
3.1.2 重入锁的好搭档：Condition.....	81
3.1.3 允许多个线程同时访问：信号量（Semaphore）.....	85
3.1.4 ReadWriteLock 读写锁.....	86
3.1.5 倒计时器：CountDownLatch.....	89
3.1.6 循环栅栏：CyclicBarrier.....	91
3.1.7 线程阻塞工具类：LockSupport.....	94
3.1.8 Guava 和 RateLimiter 限流.....	98
3.2 线程复用：线程池.....	101
3.2.1 什么是线程池.....	102
3.2.2 不要重复发明轮子：JDK 对线程池的支持.....	102
3.2.3 刨根究底：核心线程池的内部实现.....	108
3.2.4 超负载了怎么办：拒绝策略.....	112
3.2.5 自定义线程创建：ThreadFactory.....	115
3.2.6 我的应用我做主：扩展线程池.....	116
3.2.7 合理的选择：优化线程池线程数量.....	119
3.2.8 堆栈去哪里了：在线程池中寻找堆栈.....	120
3.2.9 分而治之：Fork/Join 框架.....	124
3.2.10 Guava 中对线程池的扩展.....	128
3.3 不要重复发明轮子：JDK 的并发容器.....	130
3.3.1 超好用的工具类：并发集合简介.....	130
3.3.2 线程安全的 HashMap.....	131
3.3.3 有关 List 的线程安全.....	132
3.3.4 高效读写的队列：深度剖析 ConcurrentLinkedQueue 类.....	132
3.3.5 高效读取：不变模式下的 CopyOnWriteArrayList 类.....	138
3.3.6 数据共享通道：BlockingQueue.....	139
3.3.7 随机数据结构：跳表（SkipList）.....	144
3.4 使用 JMH 进行性能测试.....	146
3.4.1 什么是 JMH.....	147

3.4.2	Hello JMH.....	147
3.4.3	JMH 的基本概念和配置.....	150
3.4.4	理解 JMH 中的 Mode.....	151
3.4.5	理解 JMH 中的 State.....	153
3.4.6	有关性能的一些思考.....	154
3.4.7	CopyOnWriteArrayList 类与 ConcurrentLinkedQueue 类.....	157
第 4 章	锁的优化及注意事项.....	161
4.1	有助于提高锁性能的几点建议.....	162
4.1.1	减少锁持有时间.....	162
4.1.2	减小锁粒度.....	163
4.1.3	用读写分离锁来替换独占锁.....	165
4.1.4	锁分离.....	165
4.1.5	锁粗化.....	168
4.2	Java 虚拟机对锁优化所做的努力.....	169
4.2.1	锁偏向.....	169
4.2.2	轻量级锁.....	169
4.2.3	自旋锁.....	170
4.2.4	锁消除.....	170
4.3	人手一支笔: ThreadLocal.....	171
4.3.1	ThreadLocal 的简单使用.....	171
4.3.2	ThreadLocal 的实现原理.....	173
4.3.3	对性能有何帮助.....	179
4.4	无锁.....	182
4.4.1	与众不同的并发策略: 比较交换.....	182
4.4.2	无锁的线程安全整数: AtomicInteger.....	183
4.4.3	Java 中的指针: Unsafe 类.....	185
4.4.4	无锁的对象引用: AtomicReference.....	187
4.4.5	带有时间戳的对象引用: AtomicStampedReference.....	190
4.4.6	数组也能无锁: AtomicIntegerArray.....	193
4.4.7	让普通变量也享受原子操作: AtomicIntegerFieldUpdater.....	194

4.4.8	挑战无锁算法：无锁的 Vector 实现	196
4.4.9	让线程之间互相帮助：细看 SynchronousQueue 的实现	201
4.5	有关死锁的问题	205
第 5 章	并行模式与算法	209
5.1	探讨单例模式	209
5.2	不变模式	213
5.3	生产者-消费者模式	215
5.4	高性能的生产者-消费者模式：无锁的实现	220
5.4.1	无锁的缓存框架：Disruptor	221
5.4.2	用 Disruptor 框架实现生产者-消费者模式的案例	222
5.4.3	提高消费者的响应时间：选择合适的策略	225
5.4.4	CPU Cache 的优化：解决伪共享问题	226
5.5	Future 模式	230
5.5.1	Future 模式的主要角色	232
5.5.2	Future 模式的简单实现	233
5.5.3	JDK 中的 Future 模式	236
5.5.4	Guava 对 Future 模式的支持	238
5.6	并行流水线	240
5.7	并行搜索	244
5.8	并行排序	246
5.8.1	分离数据相关性：奇偶交换排序	246
5.8.2	改进的插入排序：希尔排序	250
5.9	并行算法：矩阵乘法	254
5.10	准备好了再通知我：网络 NIO	258
5.10.1	基于 Socket 的服务端多线程模式	259
5.10.2	使用 NIO 进行网络编程	264
5.10.3	使用 NIO 来实现客户端	272
5.11	读完了再通知我：AIO	274
5.11.1	AIO EchoServer 的实现	275
5.11.2	AIO Echo 客户端的实现	277

第 6 章	Java 8/9/10 与并发	281
6.1	Java 8 的函数式编程简介	281
6.1.1	函数作为一等公民	282
6.1.2	无副作用	283
6.1.3	声明式的 (Declarative)	283
6.1.4	不变的对象	284
6.1.5	易于并行	284
6.1.6	更少的代码	284
6.2	函数式编程基础	285
6.2.1	FunctionalInterface 注释	285
6.2.2	接口默认方法	286
6.2.3	lambda 表达式	290
6.2.4	方法引用	291
6.3	一步一步走入函数式编程	293
6.4	并行流与并行排序	298
6.4.1	使用并行流过滤数据	298
6.4.2	从集合得到并行流	299
6.4.3	并行排序	299
6.5	增强的 Future: CompletableFuture	300
6.5.1	完成了就通知我	300
6.5.2	异步执行任务	301
6.5.3	流式调用	303
6.5.4	CompletableFuture 中的异常处理	303
6.5.5	组合多个 CompletableFuture	304
6.5.6	支持 timeout 的 CompletableFuture	306
6.6	读写锁的改进: StampedLock	306
6.6.1	StampedLock 使用示例	307
6.6.2	StampedLock 的小陷阱	308
6.6.3	有关 StampedLock 的实现思想	310
6.7	原子类的增强	313
6.7.1	更快的原子类: LongAdder	314

6.7.2	LongAdder 功能的增强版: LongAccumulator.....	320
6.8	ConcurrentHashMap 的增强.....	321
6.8.1	foreach 操作.....	321
6.8.2	reduce 操作.....	321
6.8.3	条件插入.....	322
6.8.4	search 操作.....	323
6.8.5	其他新方法.....	324
6.9	发布和订阅模式.....	324
6.9.1	简单的发布订阅例子.....	326
6.9.2	数据处理链.....	328
第 7 章	使用 Akka 构建高并发程序.....	331
7.1	新并发模型: Actor.....	332
7.2	Akka 之 Hello World.....	332
7.3	有关消息投递的一些说明.....	336
7.4	Actor 的生命周期.....	337
7.5	监督策略.....	341
7.6	选择 Actor.....	346
7.7	消息收件箱 (Inbox).....	346
7.8	消息路由.....	348
7.9	Actor 的内置状态转换.....	351
7.10	询问模式: Actor 中的 Future.....	354
7.11	多个 Actor 同时修改数据: Agent.....	356
7.12	像数据库一样操作内存数据: 软件事务内存.....	359
7.13	一个有趣的例子: 并发粒子群的实现.....	363
7.13.1	什么是粒子群算法.....	364
7.13.2	粒子群算法的计算过程.....	364
7.13.3	粒子群算法能做什么.....	366
7.13.4	使用 Akka 实现粒子群.....	367

第 8 章 并行程序调试	375
8.1 准备实验样本	375
8.2 正式起航	376
8.3 挂起整个虚拟机	379
8.4 调试进入 ArrayList 内部	380
第 9 章 多线程优化示例——Jetty 核心代码分析	385
9.1 Jetty 简介与架构	385
9.2 Jetty 服务器初始化	387
9.2.1 初始化线程池	387
9.2.2 初始化 ScheduledExecutorScheduler	389
9.2.3 初始化 ByteBufferPool	390
9.2.4 维护 ConnectionFactory	393
9.2.5 计算 ServerConnector 的线程数量	394
9.3 启动 Jetty 服务器	394
9.3.1 设置启动状态	394
9.3.2 注册 ShutdownMonitor	395
9.3.3 计算系统的线程数量	395
9.3.4 启动 QueuedThreadPool	396
9.3.5 启动 Connector	396
9.4 处理 HTTP 请求	399
9.4.1 Accept 成功	399
9.4.2 请求处理	401

1

第 1 章

走入并行世界

当你打开本书，也许你正试图将你的应用改造成并行模式运行，也许你只是单纯地对并行程序感兴趣。无论出于何种原因，你正对并行计算充满好奇、疑问和求知欲。如果是这样，那就对了，带着你的好奇和疑问，让我们一起遨游并行程序的世界，深入了解它们究竟是如何工作的吧！

不过首先，我想要公布一条令人沮丧的消息。就在大伙儿都认为并行计算必然成为未来的大趋势时，2014 年底，在 *Avoiding ping pong* 论坛上，伟大的 Linus Torvalds 提出了一个截然不同的观点，他说：“忘掉那该死的并行吧！”（原文：*Give it up. The whole "parallel computing is the future" is a bunch of crock.*）

1.1 何去何从的并行计算

我们到底该如何选择呢？本节的目的就是拨云见日。

1.1.1 忘掉那该死的并行

Linus Torvalds 是一个传奇式的人物 (图 1.1), 是他给出了 Linux 的原型, 并一直致力于推广和发展 Linux 系统。1991 年, 他首先在网络上发布了 Linux 源码, 从此 Linux 迅速崛起壮大, 成为目前使用最广泛的操作系统之一。

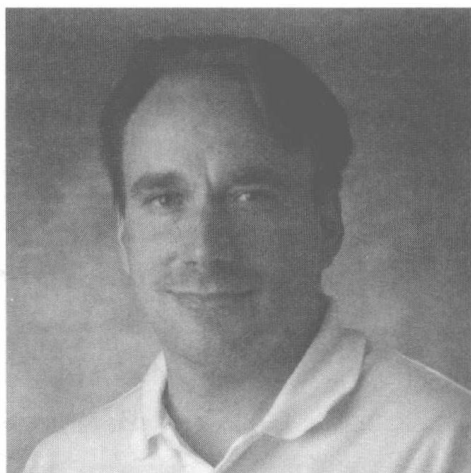


图 1.1 传奇的 Linus Torvalds

自 2002 年起, Linus 就决定使用 BitKeeper 作为 Linux 内核开发的版本控制工具, 以此来维护 Linux 的内核源码。BitKeeper 是一套分布式版本控制软件, 它是一套商用系统, 由 BitMover 公司开发。2005 年, BitKeeper 宣称发现 Linux 内核开发人员使用逆向工程来试图解析 BitKeeper 内部协议。因此, 决定向 Linus 收回 BitKeeper 授权。虽然 Linux 核心团队与 BitMover 公司进行了协商, 但是仍无法解决他们之间的分歧。因此, Linus 决定自行研发版本控制工具来代替 BitKeeper。于是, Git 诞生了。

如果你正在使用 Git, 那么我相信你一定会为 Git 的魅力所征服; 如果你还没有了解过 Git, 那么我强烈建议你关注一下这款优秀的产品。

而正是这位传奇人物, 给目前红红火火的并行计算泼了一大盆冷水。那么, 并行计算究竟应该何去何从呢?

在 Linus 的发言中这么说道:

Where the hell do you envision that those magical parallel algorithms would be used?

The only place where parallelism matters is in graphics or on the server side, where we already largely have it. Pushing it anywhere else is just pointless.