实战Java高并发程序设计

作者：葛一鸣/郭超/

目录

图书在版编目（CIP）数据

实战Java高并发程序设计 / 葛一鸣，郭超编著．——北京：电子工业出版社，2015.11

ISBN 978-7-121-27304-9

I．①实… II．①葛…②郭… III．①JAVA语言—程序设计 IV．①TP312

中国版本图书馆CIP数据核字（2015）第231507号

责任编辑：董 英

印 刷：

装 订：

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：22

字 数：493千字

版 次：2015年11月第1版

印 次：2015年11月第1次印刷

印 数：3000册

定 价：69.00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。

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。

服务热线：（010）88258888。

内容简介

在过去单核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，并行程序开发就显得尤为重要。

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中进行并行程序设计的基本方法。第二，进一步详细介绍JDK中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，详细讨论有关“锁”的优化和提高并行程序性能级别的方法和思路。第四，介绍并行的基本设计模式及Java 8对并行程序的支持和改进。第五，介绍高并发框架Akka的使用方法。最后，详细介绍并行程序的调试方法。

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。

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架构成为了一种必然的技术趋势。所以，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。可以看到，目前服务端CPU的核心数已经轻松超越10核心，而Java显然已经成为当下最流行的服务端编程语言，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8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和误用。

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虚拟机层面对并行程序的优化支持。此外，还花费一定篇幅介绍了有关无锁的计算。

第5章介绍了并行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中为并行计算做的新的改进，包括并行流、CompletableFuture、StampedLock和LongAdder。

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。

本书特色

本书的主要特点如下。

1. 结构清晰。本书一共8章，总体上循序渐进，逐步提升。每一章都各自有鲜明的侧重点，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。同时也不忘记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，但要通读本书并取得良好的学习效果，要求读者需要具备基本的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的理解的研发人员

本书的约定

本书在叙述过程中，有如下约定：

- 本书中所述的JDK 1.5、JDK 1.6、JDK 1.7、JDK 1.8分别等同于JDK 5、JDK 6、JDK 7、JDK 8。
- 如无特殊说明，本书的程序、示例均在JDK 1.7环境中运行。

联系作者

本书的写作过程远比我想象得更艰辛，为了让全书能够更清楚、更正确地表达和论述，我经历了很多个不眠之夜，即使现在回想起来，我也忍不住会打个寒战。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397196583，一起探讨学习中的困难、分享学习的经验，我期待与大家一起交流、共同进步。同时，也希望大家可以关注我的博客http://www.uucode.net/。

感谢

这本书能够面世，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，并可以安心工作。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，才能让本书逐步完善。最后，感谢我的母亲30年如一日对我的体贴和关心。

参与本书编写的还有安继宏、白慧、薛淑英、蒋玺、曹静、马玉杰、陈明明、张丽萍、任娜娜、李清艺、荆海霞、赵全利、孙迪，在此一并感谢！

葛一鸣

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，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

到底我们该如何选择呢？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。他在1991年首先在网络上发布了Linux源码，从此一发而不可收。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。尽管Linux核心团队与BitMover公司进行了协商，但是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。

如果大家正在使用Git，我相信你们一定会被Git的魅力所折服，如果还没有了解过Git，那么我强烈建议你去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.

需要有多么奇葩的想象力才能想象出并行计算的用武之地？

并行计算只有在图像处理和服务端编程2个领域可以使用，并且它在这2个领域确实有着大量广泛的使用。但是在其他任何地方，并行计算毫无建树！

So the whole argument that people should parallelize their code is fundamentally flawed. It rests on incorrect assumptions. It's a fad that has been going on too long.

因此，人们在争论是否应该将他们的代码并行化是一个本质上的错误。这完全就基于一个错误的假设。“并行”是一个早该结束的时髦用语。

看了这段较为完整的表述，大家应该对Linus的观点有所感触，我对此也表示赞同。与串行程序不同，并行程序的设计和实现异常复杂，不仅仅体现在程序的功能分离上，多线程间的协调性、乱序性都会成为程序正确执行的障碍。只要你稍不留神，就会失之毫厘，谬以千里！混乱的程序难以阅读、难以理解，更难以调试。所谓并行，也就是把简单问题复杂化的典型。因此，只有“疯子”才会叫嚣并行就是未来（the crazies talking about scaling to hundreds of cores are just that - crazy）。

但是，Linus也提出了两个特例，那就是图像处理和服务端程序是可以、也需要使用并行技术的。仔细想想，为什么图像处理和服务端程序是特例呢？

和用户终端程序不同，图像处理往往拥有极大的计算量。一张1024×768像素的图片，包含多达78万6千多个像素。即使将所有的像素遍历一遍，也得花不少时间。更何况，图像处理涉及大量的矩阵计算。矩阵的规模和数量都会非常大。面对如此密集的计算，很有可能超过单核CPU的计算能力，所以自然需要引入多核计算了。

而服务端程序与一般的用户终端程序相比，一方面，服务端程序需要承受很重的用户访问压力。根据淘宝的数据，它在“双十一”一天，支付宝核心数据库集群处理了41亿个事务，执行285亿次SQL，生成15TB日志，访问1931亿次内存数据块，13亿个物理读。如此密集的访问，恐怕任何一台单机都难以胜任，因此，并行程序也就自然成了唯一的出路。另一方面，服务端程序往往会比用户终端程序拥有更复杂的业务模型。面对复杂业务模型，并行程序会比串行程序更容易适应业务需求，更容易模拟我们的现实世界。毕竟，我们的世界本质上是并行的。比如，当你开开心心去上学的时候，妈妈可能在家里忙着家务，爸爸在外打工赚钱，一家人其乐融融。如果有一天，你需要使用你的计算机来模拟这个场景，你会怎么做呢？如果你就在一个线程里，既做了你自己，又做了妈妈，又做了爸爸，显然这不是一种好的解决方案。但如果你使用三个线程，分别模拟这三个人，一切看起来又是那么自然，而且容易被人理解。

再举一个专业点的例子，比如基础平台Java虚拟机，虚拟机除了要执行main函数主线程外，还需要做JIT编译，需要做垃圾回收。无论是main函数、JIT编译还是垃圾回收，在虚拟机内部都实现为单独的一个线程。是什么使得虚拟机的研发人员这么做呢？显然，这是因为建模的需要。因为这里的每一个任务都是相对独立的。我们不应该将没有关联的业务代码拼凑在一起，分离为不同的线程更容易理解和维护。因此，使用并行也不完全出自性能的考虑，而有时候，我们会很自然地那么做。

1.1.2 可怕的现实：摩尔定律的失效

摩尔定律是由英特尔创始人之一戈登·摩尔提出来的。其内容为：集成电路上可容纳的电晶体（晶体管）数目，约每隔24个月便会增加一倍；经常被引用的“18个月”，是由英特尔首席执行官大卫·豪斯所说：预计18个月会将芯片的性能提高一倍（即更多的晶体管使其更快）。

说得直白点，就是每18个月到24个月，我们的计算机性能就能翻一番。

反过来说，就是每过18个月到24个月，你在未来用一半的价钱就能买到和现在性能相同的计算设备了。这听起来是一件多么激动人心的事情呀！

但是，摩尔定律并不是一种自然法则或者物理定律，它只是基于人为观测数据后，对未来的预测。按照这种速度，我们的计算能力将会按照指数速度增长，用不了多久，我们的计算能力就能超越“上帝”了！畅想未来，基于强劲的超级计算机，我们甚至可以模拟整个宇宙。

摩尔定律的有效性已经超过半个世纪了，然而，在2004年，Intel宣布将4GHz芯片的发布时间推迟到2005年，在2004年秋季，Intel宣布彻底取消4GHz计划（图1.2）。

图1.2 Intel CEO Barret单膝下跪对取消4GHz感到抱歉

是什么迫使世界顶级的科技巨头放弃4GHz的研发呢？显然，就目前的硅电路而言，很有可能已经走到了头。我们的制造工艺已经到了纳米了。1纳米是10-9米，也就是10亿分之一米。这已经是一个相当小的数字了。就目前的科技水平而言，如果无法在物质分子层面以下进行工作，那么也许4GHz的芯片就已经接近了理论极限。因为即使一个水分子，它的直径也有0.4纳米。再往下发展就显得有些困难。当然，如果我们使用完全不同的计算理论或者芯片生产工艺，也许会有本质的突破，但目前还没有看到这种技术被大规模使用的可能。

因此，摩尔定律在CPU的计算性能上可能已经失效。虽然，现在Intel已经研制出了4GHz芯片，但可以看到，在近10年的发展中，CPU主频的提升已经明显遇到了一些暂时不可逾越的瓶颈。

1.1.3 柳暗花明：不断地前进

虽然CPU的性能已经几近止步，长达半个世纪的摩尔定律轰然倒地。但是这依然没有阻挡科学家和工程师们带领我们不断向前的脚步。

从2005年开始，我们已经不再追求单核的计算速度，而着迷于研究如何将多个独立的计算单元整合到单独的CPU中，也就是我们所说的多核CPU。短短十几年的发展，家用型CPU，比如Intel i7就可以拥有4核心，甚至8核心。而专业服务器则通常可以配有几个独立的CPU，每一个CPU都拥有多达8个甚至更多的内核。从整体上看，专业服务器的内核总数甚至可以达到几百个。

非常令人激动，摩尔定律在另外一个侧面又生效了。根据这个定律，我们可以预测，每过18到24个月，CPU的核心数就会翻一番。用不了多久，拥有几十甚至上百CPU内核的芯片就能进入千家万户。

顶级计算机科学家唐纳德·尔文·克努斯（Donald Ervin Knuth），如此评价这种情况：在我看来，这种现象（并发）或多或少是由于硬件设计者已经无计可施了导致的，他们将摩尔定律失效的责任推脱给软件开发者。

唐纳德（图1.3）是著名计算机巨著《计算机程序设计艺术》的作者。《美国科学家》杂志曾将该书与爱因斯坦的《相对论》，狄拉克的《量子力学》和理查·费曼的《量子电动力学》等书并列为20世纪最重要的12本物理科学类专论书之一。

图1.3 唐纳德院士

1.1.4 光明或是黑暗

根据唐纳德的观点，摩尔定律本应该由硬件开发人员维持。但是，很不幸，硬件工程师似乎已经无计可施了。为了继续