阿里系和1号店资深技术专家撰写，Java并发编程领域的扛鼎之作，内容在InfoQ等社群得到高度认可

从JDK源码、JVM、CPU等多角度全面剖析与讲解Java并发编程的框架、原理和核心技术

Java
并发编程的艺术

The Art of Java Concurrency Programming

方腾飞 魏鹏 程晓明 著

机械工业出版社
China Machine Press

前言

为什么要写这本书

记得第一次写并发编程的文章时还是在2012年，当时花了几个星期的时间写了一篇文章《深入分析volatile的实现原理》，准备在自己的博客中发表。在同事建法的建议下，怀着试一试的心态投向了InfoQ，庆幸的是半小时后得到InfoQ主编采纳的回复，高兴之情无以言表。这也是我第一次在专业媒体上发表文章，而后在InfoQ编辑张龙的不断鼓励和支持下，我陆续在InfoQ发表了几篇与并发编程相关的文章，于是便形成了“聊聊并发”专栏。在这个专栏的写作过程中，我得到快速的成长和非常多的帮助，在此非常感谢InfoQ的编辑们。2013年，华章的福川兄找到我，问有没有兴趣写一本书，当时觉得自己资历尚浅，婉言拒绝了。后来和福川兄一直保持联系，最后允许我花两年的时间来完成本书，所以答应了下来。由于并发编程领域的技术点非常多且深，所以陆续又邀请了同事魏鹏和朋友晓明一起参与到本书的编写当中。

写本书的过程也是对自己研究和掌握的技术点进行整理的过程，希望本书能帮助读者快速掌握并发编程技术。

本书一共11章，由三名作者共同编写完成，其中第3章和第10章节由程晓明编写，第4章和第5章由魏鹏编写，其他7章由方腾飞编写。

本书特色

本书结合JDK的源码介绍了Java并发框架、线程池的实现原理，帮助读者做到知其所以然。

本书对原理的剖析不仅仅局限于Java层面，而是深入到JVM，甚至CPU层面来进行讲解，帮助读者从更底层看并发技术。

本书结合线上应用，给出了一些并发编程实战技巧，以及线上处理并发问题的步骤和思路。

读者对象

·Java开发工程师

·架构师

·并发编程爱好者

·开设相关课程的大专院校师生

如何阅读本书

阅读本书之前，你必须有一定的Java基础和开发经验，最好还有一定的并发编程基础。如果你是一名并发编程初学者，建议按照顺序阅读本书，并按照书中的例子进行编码和实战。如果你有一定的并发编程经验，可以把本书当做一个手册，直接看需要学习的章节。以下是各章节的基本介绍。

第1章介绍Java并发编程的挑战，向读者说明进入并发编程的世界可能会遇到哪些问题，以及如何解决。

第2章介绍Java并发编程的底层实现原理，介绍在CPU和JVM这个层面是如何帮助Java实现并发编程的。

第3章介绍深入介绍了Java的内存模型。Java线程之间的通信对程序员完全透明，内存可见性问题很容易困扰Java程序员，本章试图揭开Java内存模型的神秘面纱。

第4章从介绍多线程技术带来的好处开始，讲述了如何启动和终止线程以及线程的状态，

详细阐述了多线程之间进行通信的基本方式和等待/通知经典范式。

第5章介绍Java并发包中与锁相关的API和组件，以及这些API和组件的使用方式与实现细节。

第6章介绍了Java中的大部分并发容器，并深入剖析其实现原理，让读者领略大师的设计技巧。

第7章介绍了Java中的原子操作类，并给出一些实例。

第8章介绍了Java中提供的并发工具类，这是并发编程中的瑞士军刀。

第9章介绍了Java中的线程池实现原理和使用建议。

第10章介绍了Executor框架的整体结构和成员组件。

第11章介绍几个并发编程的实战，以及排查并发编程造成问题的方法。

勘误和支持

由于笔者的水平有限，编写时间仓促，书中难免会出现一些错误或者不准确的地方，恳请读者批评指正。为此，特意创建一个在线支持与应急方案的站点http://ifeve.com/book/。你可以将书中的错误发布在勘误表页面中，同时如果你遇到任何问题，也可以访问Q&A页面，我将尽量在线上为读者提供最满意的解答。书中的全部源文件除可以从华章网站[1]下载外，还可以从并发编程网站[2]下载，我也会将相应的功能更新及时发布出来。如果你有更多的宝贵意见，也欢迎发送邮件至邮箱tengfei@ifeve.com，期待能够得到你的真挚反馈。

致谢

感谢机械工业出版社华章公司的杨福川、高婧雅、孙海亮，在这一年多的时间中始终支持

我的写作，你们的鼓励和帮助引导我顺利完成全部书稿。

感谢方正电子的刘老师，是他带我进入了面向对象的世界。

感谢我的主管朱老板，他在工作和生活上给予我很多的帮助和支持，还经常激励我完成本书编写。

最后感谢我的爸妈、岳父母和老婆，感谢你们的支持，并时时刻刻为我灌输信心和力量！

谨以此书献给我的儿子方熙皓，希望他能健康成长，以及众多热爱并发编程的朋友们，希望你们能快乐工作，认真生活！

方腾飞

[1] 参见华章网站www.hzbook.com。——编辑注

[2] http://ifeve.com。——编者注

第1章　并发编程的挑战

并发编程的目的是为了让程序运行得更快，但是，并不是启动更多的线程就能让程序最大限度地并发执行。在进行并发编程时，如果希望通过多线程执行任务让程序运行得更快，会面临非常多的挑战，比如上下文切换的问题、死锁的问题，以及受限于硬件和软件的资源限制问题，本章会介绍几种并发编程的挑战以及解决方案。

1.1 上下文切换

即使是单核处理器也支持多线程执行代码，CPU通过给每个线程分配CPU时间片来实现这个机制。时间片是CPU分配给各个线程的时间，因为时间片非常短，所以CPU通过不停地切换线程执行，让我们感觉多个线程是同时执行的，时间片一般是几十毫秒（ms）。

CPU通过时间片分配算法来循环执行任务，当前任务执行一个时间片后会切换到下一个任务。但是，在切换前会保存上一个任务的状态，以便下次切换回这个任务时，可以再加载这个任务的状态。所以任务从保存到再加载的过程就是一次上下文切换。

这就像我们同时读两本书，当我们在读一本英文的技术书时，发现某个单词不认识，于是便打开中英文字典，但是在放下英文技术书之前，大脑必须先记住这本书读到了多少页的第多少行，等查完单词之后，能够继续读这本书。这样的切换是会影响读书效率的，同样上下文切换也会影响多线程的执行速度。

1.1.1 多线程一定快吗

下面的代码演示串行和并发执行并累加操作的时间，请分析：下面的代码并发执行一定比串行执行快吗？

```
public class ConcurrencyTest {
        private static final long count = 10000l;
        public static void main(String[] args) throws InterruptedException {
                concurrency();
                serial();
        }
        private static void concurrency() throws InterruptedException {
                long start = System.currentTimeMillis();
                Thread thread = new Thread(new Runnable() {
                        @Override
                        public void run() {
                                int a = 0;
                                for (long i = 0; i < count; i++) {
                                        a += 5;
                                }
                        }
                });
                thread.start();
                int b = 0;
                for (long i = 0; i < count; i++) {
                        b--;
                }
                long time = System.currentTimeMillis() - start;
                thread.join();
                System.out.println("concurrency :" + time+"ms,b="+b);
        }
        private static void serial() {
                long start = System.currentTimeMillis();
                int a = 0;
                for (long i = 0; i < count; i++) {
                        a += 5;
                }
                int b = 0;
                for (long i = 0; i < count; i++) {
                        b--;
                }
                long time = System.currentTimeMillis() - start;
                System.out.println("serial:" + time+"ms,b="+b+",a="+a);
        }
}
```

上述问题的答案是“不一定”，测试结果如表1-1所示。

表1-1 测试结果

循环次数	串行执行耗时 /ms	并发执行耗时	并发比串行快多少
1 亿	130	77	约 1 倍
1 千万	18	9	约 1 倍
1 百万	5	5	差不多
10 万	4	3	慢
1 万	0	1	慢

从表1-1可以发现，当并发执行累加操作不超过百万次时，速度会比串行执行累加操作要慢。那么，为什么并发执行的速度会比串行慢呢？这是因为线程有创建和上下文切换的开销。