

## <<Java并发编程实战>>

### 图书基本信息

书名：<<Java并发编程实战>>

13位ISBN编号：9787111370048

10位ISBN编号：711137004X

出版时间：2012-2

出版时间：机械工业出版社华章公司

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

页数：312

译者：童云兰

版权说明：本站所提供下载的PDF图书仅提供预览和简介，请支持正版图书。

更多资源请访问：<http://www.tushu007.com>

## &lt;&lt;Java并发编程实战&gt;&gt;

## 前言

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，所要面临的挑战之一就是：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

## <<Java并发编程实战>>

### 内容概要

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。

## <<Java并发编程实战>>

### 作者简介

本书作者都是Java Community Process JSR 166专家组（并发工具）的主要成员，并在其他很多JCP专家组里任职。

Brian Goetz有20多年的软件咨询行业经验，并著有至少75篇关于Java开发的文章。

Tim Peierls是“现代多处理器”的典范，他在BoxPop.biz、唱片艺术和戏剧表演方面也颇有研究。

Joseph Bowbeer是一个Java ME专家，他对并发编程的兴趣始于Apollo计算机时代。

David Holmes是《The Java Programming Language》一书的合著者，任职于Sun公司。

Joshua Bloch是Google公司的首席Java架构师，《Effective Java》一书的作者，并参与著作了《Java Puzzlers》。

Doug Lea是《Concurrent Programming》一书的作者，纽约州立大学Oswego分校的计算机科学教授。

## &lt;&lt;Java并发编程实战&gt;&gt;

## 书籍目录

对本书的赞誉

译者序

前言

第1章 简介

1.1 并发简史

1.2 线程的优势

1.2.1 发挥多处理器的强大能力

1.2.2 建模的简单性

1.2.3 异步事件的简化处理

1.2.4 响应更灵敏的用户界面

1.3 线程带来的风险

1.3.1 安全性问题

1.3.2 活跃性问题

1.3.3 性能问题

1.4 线程无处不在

第一部分 基础知识

第2章 线程安全性

2.1 什么是线程安全性

2.2 原子性

2.2.1 竞态条件

2.2.2 示例：延迟初始化中的竞态条件

2.2.3 复合操作

2.3 加锁机制

2.3.1 内置锁

2.3.2 重入

2.4 用锁来保护状态

2.5 活跃性与性能

第3章 对象的共享

3.1 可见性

3.1.1 失效数据

3.1.2 非原子的64位操作

3.1.3 加锁与可见性

3.1.4 Volatile变量

3.2 发布与逸出

3.3 线程封闭

3.3.1 Ad-hoc线程封闭

3.3.2 栈封闭

3.3.3 ThreadLocal类

3.4 不变性

3.4.1 Final域

3.4.2 示例：使用Volatile类型来发布不可变对象

3.5 安全发布

3.5.1 不正确的发布：正确的对象被破坏

3.5.2 不可变对象与初始化安全性

3.5.3 安全发布的常用模式

## &lt;&lt;Java并发编程实战&gt;&gt;

3.5.4 事实不可变对象

3.5.5 可变对象

3.5.6 安全地共享对象

#### 第4章 对象的组合

4.1 设计线程安全的类

4.1.1 收集同步需求

4.1.2 依赖状态的操作

4.1.3 状态的所有权

4.2 实例封闭

4.2.1 Java监视器模式

4.2.2 示例：车辆追踪

4.3 线程安全性的委托

4.3.1 示例：基于委托的车辆追踪器

4.3.2 独立的状态变量

4.3.3 当委托失效时

4.3.4 发布底层的状态变量

4.3.5 示例：发布状态的车辆追踪器

4.4 在现有的线程安全类中添加功能

4.4.1 客户端加锁机制

4.4.2 组合

4.5 将同步策略文档化

#### 第5章 基础构建模块

5.1 同步容器类

5.1.1 同步容器类的问题

5.1.2 迭代器与Concurrent-ModificationException

5.1.3 隐藏迭代器

5.2 并发容器

5.2.1 ConcurrentHashMap

5.2.2 额外的原子Map操作

5.2.3 CopyOnWriteArrayList

5.3 阻塞队列和生产者-消费者模式

5.3.1 示例：桌面搜索

5.3.2 串行线程封闭

5.3.3 双端队列与工作窃取

5.4 阻塞方法与中断方法

5.5 同步工具类

5.5.1 闭锁

5.5.2 FutureTask

5.5.3 信号量

5.5.4 栅栏

5.6 构建高效且可伸缩的结果缓存

#### 第二部分 结构化并发应用程序

#### 第6章 任务执行

6.1 在线程中执行任务

6.1.1 串行地执行任务

6.1.2 显式地为任务创建线程

6.1.3 无限制创建线程的不足

## &lt;&lt;Java并发编程实战&gt;&gt;

## 6.2 Executor框架

6.2.1 示例：基于Executor的Web服务器

6.2.2 执行策略

6.2.3 线程池

6.2.4 Executor的生命周期

6.2.5 延迟任务与周期任务

## 6.3 找出可利用的并行性

6.3.1 示例：串行的页面渲染器

6.3.2 携带结果的任务Callable与Future

6.3.3 示例：使用Future实现页面渲染器

6.3.4 在异构任务并行化中存在的局限

6.3.5 CompletionService:Executor与BlockingQueue

6.3.6 示例：使用CompletionService实现页面渲染器

6.3.7 为任务设置时限

6.3.8 示例：旅行预定门户网站

## 第7章 取消与关闭

## 7.1 任务取消

7.1.1 中断

7.1.2 中断策略

7.1.3 响应中断

7.1.4 示例：计时运行

7.1.5 通过Future来实现取消

7.1.6 处理不可中断的阻塞

7.1.7 采用newTaskFor来封装非标准的取消

## 7.2 停止基于线程的服务

7.2.1 示例：日志服务

7.2.2 关闭ExecutorService

7.2.3 “毒丸”对象

7.2.4 示例：只执行一次的服务

7.2.5 shutdownNow的局限性

## 7.3 处理非正常的线程终止

## 7.4 JVM关闭

7.4.1 关闭钩子

7.4.2 守护线程

7.4.3 终结器

## 第8章 线程池的使用

## 8.1 在任务与执行策略之间的隐性耦合

8.1.1 线程饥饿死锁

8.1.2 运行时间较长的任务

## 8.2 设置线程池的大小

## 8.3 配置ThreadPoolExecutor

8.3.1 线程的创建与销毁

8.3.2 管理队列任务

8.3.3 饱和策略

8.3.4 线程工厂

8.3.5 在调用构造函数后再定制ThreadPoolExecutor

## 8.4 扩展 ThreadPoolExecutor

## &lt;&lt;Java并发编程实战&gt;&gt;

- 8.5 递归算法的并行化
- 第9章 图形用户界面应用程序
  - 9.1 为什么GUI是单线程的
    - 9.1.1 串行事件处理
    - 9.1.2 Swing中的线程封闭机制
  - 9.2 短时间的GUI任务
  - 9.3 长时间的GUI任务
    - 9.3.1 取消
    - 9.3.2 进度标识和完成标识
    - 9.3.3 SwingWorker
  - 9.4 共享数据模型
    - 9.4.1 线程安全的数据模型
    - 9.4.2 分解数据模型
  - 9.5 其他形式的单线程子系统
- 第三部分 活跃性、性能与测试
- 第10章 避免活跃性危险
  - 10.1 死锁
    - 10.1.1 锁顺序死锁
    - 10.1.2 动态的锁顺序死锁
    - 10.1.3 在协作对象之间发生的死锁
    - 10.1.4 开放调用
    - 10.1.5 资源死锁
  - 10.2 死锁的避免与诊断
    - 10.2.1 支持定时的锁
    - 10.2.2 通过线程转储信息来分析死锁
  - 10.3 其他活跃性危险
    - 10.3.1 饥饿
    - 10.3.2 糟糕的响应性
    - 10.3.3 活锁
- 第11章 性能与可伸缩性
  - 11.1 对性能的思考
    - 11.1.1 性能与可伸缩性
    - 11.1.2 评估各种性能权衡因素
  - 11.2 Amdahl定律
    - 11.2.1 示例：在各种框架中隐藏的串行部分
    - 11.2.2 Amdahl定律的应用
  - 11.3 线程引入的开销
    - 11.3.1 上下文切换
    - 11.3.2 内存同步
    - 11.3.3 阻塞
  - 11.4 减少锁的竞争
    - 11.4.1 缩小锁的范围（“快进快出”）
    - 11.4.2 减小锁的粒度
    - 11.4.3 锁分段
    - 11.4.4 避免热点域
    - 11.4.5 一些替代独占锁的方法
    - 11.4.6 监测CPU的利用率



## &lt;&lt;Java并发编程实战&gt;&gt;

- 11.4.7 向对象池说“不”
- 11.5 示例：比较Map的性能
- 11.6 减少上下文切换的开销
- 第12章 并发程序的测试
  - 12.1 正确性测试
    - 12.1.1 基本的单元测试
    - 12.1.2 对阻塞操作的测试
    - 12.1.3 安全性测试
    - 12.1.4 资源管理的测试
    - 12.1.5 使用回调
    - 12.1.6 产生更多的交替操作
  - 12.2 性能测试
    - 12.2.1 在PutTakeTest中增加计时功能
    - 12.2.2 多种算法的比较
    - 12.2.3 响应性衡量
  - 12.3 避免性能测试的陷阱
    - 12.3.1 垃圾回收
    - 12.3.2 动态编译
    - 12.3.3 对代码路径的不真实采样
    - 12.3.4 不真实的竞争程度
    - 12.3.5 无用代码的消除
  - 12.4 其他的测试方法
    - 12.4.1 代码审查
    - 12.4.2 静态分析工具
    - 12.4.3 面向方面的测试技术
    - 12.4.4 分析与监测工具
- 第四部分 高级主题
  - 第13章 显式锁
    - 13.1 Lock与 ReentrantLock
      - 13.1.1 轮询锁与定时锁
      - 13.1.2 可中断的锁获取操作
      - 13.1.3 非块结构的加锁
    - 13.2 性能考虑因素
    - 13.3 公平性
    - 13.4 在synchronized和ReentrantLock之间进行选择
    - 13.5 读-写锁
  - 第14章 构建自定义的同步工具
    - 14.1 状态依赖性的管理
      - 14.1.1 示例：将前提条件的失败传递给调用者
      - 14.1.2 示例：通过轮询与休眠来实现简单的阻塞
      - 14.1.3 条件队列
    - 14.2 使用条件队列
      - 14.2.1 条件谓词
      - 14.2.2 过早唤醒
      - 14.2.3 丢失的信号
      - 14.2.4 通知
      - 14.2.5 示例：阀门类

## &lt;&lt;Java并发编程实战&gt;&gt;

- 14.2.6 子类的安全问题
- 14.2.7 封装条件队列
- 14.2.8 入口协议与出口协议
- 14.3 显式的Condition对象
- 14.4 Synchronizer剖析
- 14.5 AbstractQueuedSynchronizer
- 14.6 java.util.concurrent同步器类中的 AQS
  - 14.6.1 ReentrantLock
  - 14.6.2 Semaphore与CountDownLatch
  - 14.6.3 FutureTask
  - 14.6.4 ReentrantReadWriteLock
- 第15章 原子变量与非阻塞同步机制
  - 15.1 锁的劣势
  - 15.2 硬件对并发的支持
    - 15.2.1 比较并交换
    - 15.2.2 非阻塞的计数器
    - 15.2.3 JVM对CAS的支持
  - 15.3 原子变量类
    - 15.3.1 原子变量是一种“更好的volatile”
    - 15.3.2 性能比较：锁与原子变量
  - 15.4 非阻塞算法
    - 15.4.1 非阻塞的栈
    - 15.4.2 非阻塞的链表
    - 15.4.3 原子的域更新器
    - 15.4.4 ABA问题
- 第16章 Java内存模型
  - 16.1 什么是内存模型，为什么需要它
    - 16.1.1 平台的内存模型
    - 16.1.2 重排序
    - 16.1.3 Java内存模型简介
    - 16.1.4 借助同步
  - 16.2 发布
    - 16.2.1 不安全的发布
    - 16.2.2 安全的发布
    - 16.2.3 安全初始化模式
    - 16.2.4 双重检查加锁
  - 16.3 初始化过程中的安全性
- 附录A 并发性标注
- 参考文献

## &lt;&lt;Java并发编程实战&gt;&gt;

## 章节摘录

版权页：插图：第一章 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。

<<Java并发编程实战>>

媒体关注与评论

## &lt;&lt;Java并发编程实战&gt;&gt;

## 编辑推荐

《Java并发编程实战》第16届Jolt大奖提名图书，JavaOne大会最畅销图书，了解Java并发编程必读佳作。

线程是Java平台的基础组成部分之一。

随着多核处理器逐渐成为主流，如何高效地使用并发已成为构建高性能应用程序的重要因素。

Java SE 5和Java 6在并发程序开发方面取得了巨大的进步，在其Java虚拟机中能支持一些高性能的并且具有高可伸缩性的并发类，此外还支持一组新的并发基础构建模块。

在《Java并发编程实战》中，这些新功能的编写者们不仅介绍了它们的工作原理和使用方式，还介绍了隐藏在这些功能背后的研究背景与设计模式。

然而，在开发、测试以及调试多线程的程序时仍然存在巨大的困难。

开发人员很容易编写出一些看似能正常工作，但在一些情况下仍然会失败的程序（包括在正式发布的产品中，以及在高负载环境中）。

《Java并发编程实战》不仅讲解了并发的理论基础，还介绍了各种实际的开发技术，这些知识对于构建可靠的、可伸缩的以及可维护的并发应用程序来说非常有用。

《Java并发编程实战》并不仅是简单地罗列出各种并发API以及机制，而是详细地介绍了许多设计原则、设计模式以及思维模式，这些内容使得开发人员更容易构建出正确的并且高性能的并发程序。

《Java并发编程实战》主要内容包括：并发性与线程安全性的基本概念，构建以及组合各种线程安全类的技术，使用java.util.concurrent包中的各种并发构建基础模块，性能优化中的注意事项，如何测试并发程序，以及一些高级主题，包括原子变量，无阻塞算法及JAVA内存模。

<<Java并发编程实战>>

版权说明

本站所提供下载的PDF图书仅提供预览和简介，请支持正版图书。

更多资源请访问:<http://www.tushu007.com>