

书名、作者、ISBN

购书单 电子图书 豆瓣书店 2021年度榜单 2021书影音报告 购物车

Java并发编程实战



作者: [Brian Goetz](#) / [Tim Peierls](#) / [Joshua Bloch](#) / [Joseph Bowbeer](#) / [David Holmes](#) / [Doug Lea](#)

出版社: [机械工业出版社](#)

原作名: [Java Concurrency in Practice](#)

译者: [童云兰](#)

出版年: 2012-2

页数: 293

定价: 69.00元

装帧: 平装

丛书: [华章专业开发者丛书](#)

ISBN: 9787111370048

豆瓣评分

9.0 1530人评价

5星	64.2%
4星	27.8%
3星	6.1%
2星	1.0%
1星	0.8%

想读 在读 读过 评价: ☆☆☆☆☆

[写笔记](#) [写书评](#) [加入购书单](#) [分享到](#)

推荐

内容简介

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。

在线试读：

 豆瓣阅读

 得到

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。

目录

- 对本书的赞誉
- 译者序
- 前言
- 第1章 简介1
 - 1.1 并发简史1
 - 1.2 线程的优势2
 - 1.2.1 发挥多处理器的强大能力2
 - 1.2.2 建模的简单性3
 - 1.2.3 异步事件的简化处理3
 - 1.2.4 响应更灵敏的用户界面4
 - 1.3 线程带来的风险4
 - 1.3.1 安全性问题5
 - 1.3.2 活跃性问题7
 - 1.3.3 性能问题7
 - 1.4 线程无处不在7
- 第一部分 基础知识
- 第2章 线程安全性11
 - 2.1 什么是线程安全性13

- 2.2 原子性14
 - 2.2.1 竞态条件15
 - 2.2.2 示例：延迟初始化中的竞态条件16
 - 2.2.3 复合操作17
- 2.3 加锁机制18
 - 2.3.1 内置锁20
 - 2.3.2 重入21
- 2.4 用锁来保护状态22
- 2.5 活跃性与性能23
- 第3章 对象的共享27
 - 3.1 可见性27
 - 3.1.1 失效数据28
 - 3.1.2 非原子的64位操作29
 - 3.1.3 加锁与可见性30
 - 3.1.4 Volatile变量 30
 - 3.2 发布与逸出32
 - 3.3 线程封闭35
 - 3.3.1 Ad-hoc线程封闭35
 - 3.3.2 栈封闭36
 - 3.3.3 ThreadLocal类37
 - 3.4 不变性38
 - 3.4.1 Final域39
 - 3.4.2 示例：使用Volatile类型来发布不可变对象40
 - 3.5 安全发布41
 - 3.5.1 不正确的发布：正确的对象被破坏42
 - 3.5.2 不可变对象与初始化安全性42
 - 3.5.3 安全发布的常用模式43
 - 3.5.4 事实不可变对象44
 - 3.5.5 可变对象44
 - 3.5.6 安全地共享对象44
- 第4章 对象的组合46
 - 4.1 设计线程安全的类46
 - 4.1.1 收集同步需求47
 - 4.1.2 依赖状态的操作48
 - 4.1.3 状态的所有权48
 - 4.2 实例封闭49
 - 4.2.1 Java监视器模式51
 - 4.2.2 示例：车辆追踪51
 - 4.3 线程安全性的委托53
 - 4.3.1 示例：基于委托的车辆追踪器54
 - 4.3.2 独立的状态变量55
 - 4.3.3 当委托失效时56
 - 4.3.4 发布底层的状态变量57
 - 4.3.5 示例：发布状态的车辆追踪器58
 - 4.4 在现有的线程安全类中添加功能59
 - 4.4.1 客户端加锁机制60
 - 4.4.2 组合62
 - 4.5 将同步策略文档化62
- 第5章 基础构建模块66
 - 5.1 同步容器类66
 - 5.1.1 同步容器类的问题66
 - 5.1.2 迭代器与ConcurrentModificationException68
 - 5.1.3 隐藏迭代器69
 - 5.2 并发容器70
 - 5.2.1 ConcurrentHashMap71
 - 5.2.2 额外的原子Map操作72
 - 5.2.3 CopyOnWriteArrayList72
 - 5.3 阻塞队列和生产者-消费者模式73
 - 5.3.1 示例：桌面搜索75
 - 5.3.2 串行线程封闭76
 - 5.3.3 双端队列与工作窃取77
 - 5.4 阻塞方法与中断方法77
 - 5.5 同步工具类78
 - 5.5.1 闭锁79
 - 5.5.2 FutureTask80
 - 5.5.3 信号量82
 - 5.5.4 栅栏83
 - 5.6 构建高效且可伸缩的结果缓存85
- 第二部分 结构化并发应用程序
- 第6章 任务执行93


- 6.1 在线程中执行任务93
 - 6.1.1 串行地执行任务94
 - 6.1.2 显式地为任务创建线程94
 - 6.1.3 无限制创建线程的不足95
- 6.2 Executor框架96
 - 6.2.1 示例：基于Executor的Web服务器97
 - 6.2.2 执行策略98
 - 6.2.3 线程池98
 - 6.2.4 Executor的生命周期99
 - 6.2.5 延迟任务与周期任务101
- 6.3 找出可利用的并行性102
 - 6.3.1 示例：串行的页面渲染器102
 - 6.3.2 携带结果的任务Callable与Future103
 - 6.3.3 示例：使用Future实现页面渲染器104
 - 6.3.4 在异构任务并行化中存在的局限106
 - 6.3.5 CompletionService:Executor与BlockingQueue106
 - 6.3.6 示例：使用CompletionService实现页面渲染器107
 - 6.3.7 为任务设置时限108
 - 6.3.8 示例：旅行预定门户网站109
- 第7章 取消与关闭111
 - 7.1 任务取消111
 - 7.1.1 中断113
 - 7.1.2 中断策略116
 - 7.1.3 响应中断117
 - 7.1.4 示例：计时运行118
 - 7.1.5 通过Future来实现取消120
 - 7.1.6 处理不可中断的阻塞121
 - 7.1.7 采用newTaskFor来封装非标准的取消122
 - 7.2 停止基于线程的服务124
 - 7.2.1 示例：日志服务124
 - 7.2.2 关闭ExecutorService127
 - 7.2.3 “毒丸”对象128
 - 7.2.4 示例：只执行一次的服务129
 - 7.2.5 shutdownNow的局限性130
 - 7.3 处理非正常的线程终止132
 - 7.4 JVM关闭135
 - 7.4.1 关闭钩子135
 - 7.4.2 守护线程136
 - 7.4.3 终结器136
- 第8章 线程池的使用138
 - 8.1 在任务与执行策略之间的隐性耦合138
 - 8.1.1 线程饥饿死锁139
 - 8.1.2 运行时间较长的任务140
 - 8.2 设置线程池的大小140
 - 8.3 配置ThreadPoolExecutor141
 - 8.3.1 线程的创建与销毁142
 - 8.3.2 管理队列任务142
 - 8.3.3 饱和策略144
 - 8.3.4 线程工厂146
 - 8.3.5 在调用构造函数后再定制ThreadPoolExecutor147
 - 8.4 扩展 ThreadPoolExecutor148
 - 8.5 递归算法的并行化149
- 第9章 图形用户界面应用程序156
 - 9.1 为什么GUI是单线程的156
 - 9.1.1 串行事件处理157
 - 9.1.2 Swing中的线程封闭机制158
 - 9.2 短时间的GUI任务160
 - 9.3 长时间的GUI任务161
 - 9.3.1 取消162
 - 9.3.2 进度标识和完成标识163
 - 9.3.3 SwingWorker165
 - 9.4 共享数据模型165
 - 9.4.1 线程安全的数据模型166
 - 9.4.2 分解数据模型166
 - 9.5 其他形式的单线程子系统167
- 第三部分 活跃性、性能与测试
- 第10章 避免活跃性危险169
 - 10.1 死锁169
 - 10.1.1 锁顺序死锁170
 - 10.1.2 动态的锁顺序死锁171

- 10.1.3 在协作对象之间发生的死锁174
- 10.1.4 开放调用175
- 10.1.5 资源死锁177
- 10.2 死锁的避免与诊断178
 - 10.2.1 支持定时的锁178
 - 10.2.2 通过线程转储信息来分析死锁178
- 10.3 其他活跃性危险180
 - 10.3.1 饥饿180
 - 10.3.2 糟糕的响应性181
 - 10.3.3 活锁181
- 第11章 性能与可伸缩性183
 - 11.1 对性能的思考183
 - 11.1.1 性能与可伸缩性184
 - 11.1.2 评估各种性能权衡因素185
 - 11.2 Amdahl定律186
 - 11.2.1 示例：在各种框架中隐藏的串行部分188
 - 11.2.2 Amdahl定律的应用189
 - 11.3 线程引入的开销189
 - 11.3.1 上下文切换190
 - 11.3.2 内存同步190
 - 11.3.3 阻塞192
 - 11.4 减少锁的竞争192
 - 11.4.1 缩小锁的范围（“快进快出”）193
 - 11.4.2 减小锁的粒度195
 - 11.4.3 锁分段196
 - 11.4.4 避免热点域197
 - 11.4.5 一些替代独占锁的方法198
 - 11.4.6 监测CPU的利用率199
 - 11.4.7 向对象池说“不”200
 - 11.5 示例：比较Map的性能200
 - 11.6 减少上下文切换的开销201
- 第12章 并发程序的测试204
 - 12.1 正确性测试205
 - 12.1.1 基本的单元测试206
 - 12.1.2 对阻塞操作的测试207
 - 12.1.3 安全性测试208
 - 12.1.4 资源管理的测试212
 - 12.1.5 使用回调213
 - 12.1.6 产生更多的交替操作214
 - 12.2 性能测试215
 - 12.2.1 在PutTakeTest中增加计时功能215
 - 12.2.2 多种算法的比较217
 - 12.2.3 响应性衡量218
 - 12.3 避免性能测试的陷阱220
 - 12.3.1 垃圾回收220
 - 12.3.2 动态编译220
 - 12.3.3 对代码路径的不真实采样222
 - 12.3.4 不真实的竞争程度222
 - 12.3.5 无用代码的消除223
 - 12.4 其他的测试方法224
 - 12.4.1 代码审查224
 - 12.4.2 静态分析工具224
 - 12.4.3 面向方面的测试技术226
 - 12.4.4 分析与监测工具226
- 第四部分 高级主题
- 第13章 显式锁227
 - 13.1 Lock与 ReentrantLock227
 - 13.1.1 轮询锁与定时锁228
 - 13.1.2 可中断的锁获取操作230
 - 13.1.3 非块结构的加锁231
 - 13.2 性能考虑因素231
 - 13.3 公平性232
 - 13.4 在synchronized和ReentrantLock之间进行选择234
 - 13.5 读-写锁235
- 第14章 构建自定义的同步工具238
 - 14.1 状态依赖性的管理238
 - 14.1.1 示例：将前提条件的失败传递给调用者240
 - 14.1.2 示例：通过轮询与休眠来实现简单的阻塞241
 - 14.1.3 条件队列243
 - 14.2 使用条件队列244

- 14.2.1 条件谓词244
- 14.2.2 过早唤醒245
- 14.2.3 丢失的信号246
- 14.2.4 通知247
- 14.2.5 示例：阀门类248
- 14.2.6 子类的安全问题249
- 14.2.7 封装条件队列250
- 14.2.8 入口协议与出口协议250
- 14.3 显式的Condition对象251
- 14.4 Synchronizer剖析253
- 14.5 AbstractQueuedSynchronizer254
- 14.6 java.util.concurrent同步器类中的 AQS257
 - 14.6.1 ReentrantLock257
 - 14.6.2 Semaphore与CountDownLatch258
 - 14.6.3 FutureTask259
 - 14.6.4 ReentrantReadWriteLock259
- 第15章 原子变量与非阻塞同步机制261
 - 15.1 锁的劣势261
 - 15.2 硬件对并发的支持262
 - 15.2.1 比较并交换263
 - 15.2.2 非阻塞的计数器264
 - 15.2.3 JVM对CAS的支持265
 - 15.3 原子变量类265
 - 15.3.1 原子变量是一种“更好的volatile”266
 - 15.3.2 性能比较：锁与原子变量267
 - 15.4 非阻塞算法270
 - 15.4.1 非阻塞的栈270
 - 15.4.2 非阻塞的链表272
 - 15.4.3 原子的域更新器274
 - 15.4.4 ABA问题275
- 第16章 Java内存模型277
 - 16.1 什么是内存模型，为什么需要它277
 - 16.1.1 平台的内存模型278
 - 16.1.2 重排序278
 - 16.1.3 Java内存模型简介280
 - 16.1.4 借助同步281
 - 16.2 发布283
 - 16.2.1 不安全的发布283
 - 16.2.2 安全的发布284
 - 16.2.3 安全初始化模式284
 - 16.2.4 双重检查加锁286
 - 16.3 初始化过程中的安全性287
- 附录A 并发性标注289
- 参考文献291
-(收起)

原文摘录 (全部)


加锁机制既可以确保可见性又可以确保原子性，而 volatile 变量只能确保可见性。当且仅当满足以下所有条件时，才应该使用 volatile 变量： - 对变量的写入操作不依赖变量的当前值，或者你能确保只有单个线程更新变量的值。 - 该变量不会与其它状态变量一起纳入到不变性条件中。 - 在访问变量时不需要加锁。 ([查看原文](#))

 红色有角F叔

20 回复 3赞 2014-05-07 09:42:03

—— 引自章节：volatile 变量

当满足以下条件时，对象才是不可变的： - 对象创建以后其状态就不可修改 - 对象的所有域都是 final 类型 - 对象时正确创建的（在对象的构造期间，this 引用没有逸出） 从技术上来看，不可变对象并不需要将其所有的域都声明为 final 类型，例如 String 就是这种情况，这就要对类的良性数据竞争情况做精确的分析，因此需要深入理解 Java 的内存模型。... 自己在编码时不要这么做。 ([查看原文](#))

 红色有角F叔

1赞 2014-05-09 00:07:06

—— 引自章节：不变性

> 全部原文摘录

丛书信息

[华章专业开发者丛书 \(共11册\)](#), 这套丛书还有 《Windows7脚本编程和命令行工具指南》, 《Python编程实践》, 《Linux内核设计与实现(原书第3版)》, 《Web开发敏捷之道》, 《测试驱动的JavaScript开发》 等。

喜欢读"Java并发编程实战"的人也喜欢的电子书 ·····

支持 Web、iPhone、iPad、Android 阅读器



实现领域驱动设计
19.80元



深入理解Java虚拟机
JVM高级特性与最佳实践
23.99元



企业应用架构模式
经典重读
25.00元

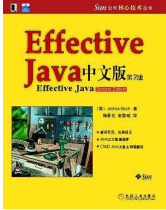


七周七并发模型
23.50元




性能之巅
29.80元

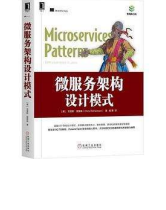
喜欢读"Java并发编程实战"的人也喜欢 ·····




Effective java 中文版
版 (第2版)




Head First 设计模式
式 (中文版)



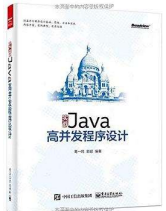
微服务架构设计模式




Netty实战



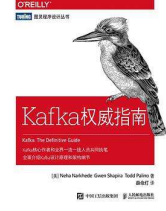
微服务设计



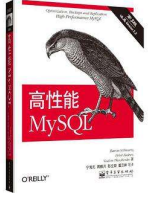
实战Java高并发程序设计



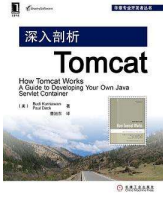
数据密集型应用系统设计



Kafka权威指南



高性能MySQL(第3版)



深入剖析Tomcat

短评 ····· (全部 509 条)

我来说两句

热门 / 最新 / 好友

- 叉

2014-02-10 00:34:27

4 有用

和operating system同调律百分之八十的书，作为os与jvm的交叉内容存在，算是已经解释得很详尽了，不过老实说看这书常常感觉浪费生命，很多东西向第9、10章的内容完全可以稍稍带过就行
- 永叹月

2015-07-18 13:17:07

2 有用

浅尝辄止...
- ayanamist

2014-12-01 23:03:24

9 有用

书的内容其实略有些单薄，对于“怎么做才是对的”讲的并不是很多
- bluedavy

2012-07-27 10:42:49

1 有用

...原来只是再出版了下...书还是当年的那本，好吧。
- 高桥

2019-07-31 10:35:14

2 有用

还行吧，这个本书有点老了，基本都是以java5、6来讲的，并发上也大都是通用的并发操作，并发的处理在哪里都是类似，本书结合java语言来讲解举例了一下。java这么多年了，这本书应该出第二个版了。

> 更多短评 509 条

Java并发编程实战的书评 ····· (全部 60 条)

我要写书评

热门 / 最新 / 好友 / 只看本版本的评论



学习机器人

2010-05-29 11:16:27

电子工业出版社2007版

中文版的翻译者就是个罪犯

英文版还是不错的，但是中文版的译者典型的没有技术功底，介绍上说什么专家，我日，他妈狗屁，翻译的非常差劲，有些句子都不通顺，都不知道自己去理解，直接就放在书上，你丫有没有良知，书籍是什么，是希望，是神圣的，你们这些译者简直就是犯罪，不过要是英文功底不好，还... (展开)

△ 50 ▽ 5 25回应



kiral

2012-11-05 23:15:13

并发编程必读书籍

请注意这本书叫《Java并发编程实战》，和《Java并发编程实践》是两本书，前者翻译的非常好，后者的翻译我基本看不懂。本书关于并发编程的细节介绍得非常详细，看得出有很多实践功底，而不是一个理论派，建议每一个学并发的同学看看。(展开)

△ 21 ▽ 4回应



米迦勒维菊

2014-06-23 16:52:13

翻译错误真是随处即拾

整体上还是可以看的，不过很多地方看不懂只是因为翻译不恰当。这本书本身值五颗星。译文：一个对象是否需要是线程安全的，取决于它是否被多个线程访问。这指的是程序中访问对象的方式，而不是对象要实现的功能。原文：Whether an object needs to be thread-safe depen... (展开)

△ 13 ▽ 0回应



阿拉丁的灯

2008-10-20 09:12:38

电子工业出版社2007版

看过的讲并发编程的最清晰的书

这本书名为《Java并发编程实践》有些抹杀了它的价值，其中并非只讲述了Java的多线程设施，对一般的并发编程的rationale也有相当透彻的阐述。之前看过各种线程库，pThread, Boost Thread, Java Thread, Qt Thread，感觉Java的线程模型还是相对比较清晰的。只要能读懂一点Java的... (展开)

△ 15 ▽ 3 4回应



Optimus Prime

2013-08-25 23:40:50

一段未完的奇妙的旅程

这本书的前半部分我读的非常仔细，但后半部分则跳跃了很多，并没有完整的阅读。当然原因肯定不是这本书不够好，而是我本人的内力实在是太差了，越看越羞愧。所谓实践是检验真理的唯一标准，要想理解书上的精髓，唯有多样实践，方能有所收获。现在的我越来越感觉自己有些“眼高手... (展开)

△ 6 ▽ 3回应



yychao

2013-02-06 15:34:29

翻译质量太差啦，建议翻译者跳出认知假设来翻译，质量好些

翻译质量不敢恭维，建议不好翻译的地方还是给原文好了；完全没有办法和原版比较，读完后，只能参照原本的一些英文去理解一些关键概念，痛苦呀；可能翻译者已经完全理解书中的内容，但是翻译时的还要注意一些地方：1.翻译内容不要假设背景知识 ---读者没有这些背景知识 2.不... (展开)

△ 7 ▽ 0回应



阿丹

2007-09-17 11:39:01

电子工业出版社2007版

java并发编程的力作

终于读完，对于原著英文版来说，绝对是力荐的。看看作者列表，一个个响亮的名字，这本书的价值就不言而喻了。可贵的是书中不仅仅是详细介绍了jdk5引入的concurrent包的使用和基本原理，并且对线程安全性的设计、性能、死锁和可伸缩性的讨论也蕴含着丰富实践经验。中文版翻... (展开)

△ 7 ▽ 2回应

 ikcd

2015-07-11 16:46:52

Addison-Wesley Professional2006版

不是教你如何使用库，而是教你原理

之前看到有人推荐这本书，于是就花了将近三个月的时间来看这本书， 前一周在看 《jvm 上的并发编程》，感觉不怎么好，原理没怎么清楚，晦涩难懂，于是换成这本书，豁然开朗。 虽然现在都是 java8了，这本书里面说的是 java5/6的事情，但是，如果不懂这些，那么 java8中的一些... [\(展开\)](#)

△ 6


▽ 0回应

> 更多书评 60篇

读书笔记 ······ (共93篇)

按有用程度 按页码先后 最新笔记

我来写笔记




第20页 2.3.1 内置锁

SamDeepThink (运动是所有事情的基本)

1、内置锁是保证操作原子性的一种手段。 2、被synchronize包住的代码块被一个锁保护着，进入代码块的线程必须获得锁才能进入。 默认情况下，如果不为synchronize关键字指定对应的锁的话，synchronize就以所属的对象作为锁 3、当使用synchronize的时候，如果另外一个线程没法获得锁，它就只能等待或者阻塞，由此会带来上下文切换的开销，CPU利用率很低。

2017-02-03 19:21:52 3人喜欢




volatile 变量

红色有角F叔 (勇敢困难 不怕牛牛)

加锁机制既可以确保可见性又可以确保原子性，而 volatile 变量只能确保可见性。 当且仅当满足以下所有条件时，才应该使用 volatile 变量： - 对变量的写入操作不依赖变量的当前值，或者你能确保只有单个线程更新变量的值。 - 该变量不会与其它状态变量一起纳入到不变性条件中。 - 在访问变量时不需要加锁。 [\(20回应\)](#)

2014-05-07 09:42:03 3人喜欢




第39页 不变性 final域

peiyuc

final域能确保初始化过程的安全性，从而可以不受限制的访问不可变对象，并在共享这些对象时无需同步。 该书对于final的安全初始化只是一句话带过，并没有详细描述final关键的内存语义。导致后面类的不变性认识有偏差。 文章理解java内存模型[\[http://www.infoq.com/cn/minibooks/java_memory_model\]](http://www.infoq.com/cn/minibooks/java_memory_model)有详细的讲解final的内存语义和实现方法 final域可以确保对象在对别的线程可见之前，域已初始化完成；普通域不能保证这一点

2016-07-22 19:33:30 1人喜欢



第45页

2sin18° (维天之命，于穆不已)

1 对象尽可能封闭在线程内部，可以用栈封闭、ThreadLocal 等方式 2 对于不可变对象，如所有字段都是 final，创建后不被修改，构造函数也没有泄漏引用的对象，可以在线程间安全地传递 3 对于事实不可变对象，可以通过static 声明中初始化，或引用保存在volatile、atomicReference、final引用、锁保护域、线程安全容器（SynchronizedMap、ConcurrentMap、BlockingQueue、ConcurrentLinkedList）中。 4 对于可变对象，除了类似于事...

2014-11-01 16:19:39 1人喜欢

> 更多读书笔记 (共93篇)

论坛 ······

现在看会不会知识滞后?	来自1	1 回应	2020-10-05 16:04:53
p272页	来自盛消		2019-10-03 19:25:37
07年出一版，今年又出一版，这两版什么区别	来自Biubiu	3 回应	2012-06-16 17:11:23
国内什么时候出影印版啊	来自lntoo		2012-06-16 17:10:37