多核时代考验Java代码编写习惯Java认证考试 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/644/2021_2022__E5_A4_9A_E6_A0_B8_E6_97_B6_E4_c104_644523.htm 我承认，这个标题是有点夸大其辞了。显然正确的Java还是有的，甚至有不少，但是我感觉这相对于Java代码的总量来说可能只是微不足道的一小部分。为什么我会有这么极端的一个说法呢？ 这又回到了Java内存模型问题上，以及对于在代码运行时“ 计算机 ”中发生了些什么的不符合人们直觉思维的。为了避免有人说我对Java有偏见，我得先声明，我非常清楚Java是第一种试图提供可靠，可用，跨平台的多线程编程环境的语言。这是很难做到的。在初试尝试时一些细节问题上出现了重大错误也应该不会有人觉得奇怪。那此问题到JSR-133就被解决了，从Java 5开始就被广泛部署开来。百考试题－全国最大教育类网站(100test.com) 但是，即使用的较新的Java环境，带有更宽松、可预见性更强的的内存模型，还是会有很多错误发生，而我们也粗心大意根本没有注意这些问题。这是怎么回事呢？必读书目《Java并发编程实践》（Java Concurrency in Practice，51CTO读书频道有这本书的试读。后面我还会提到这本书）的作者Brian Goetz说得好：“ ……由于常用的处理器（Intel和Sparc）都提供比JVM所需更强的存储能力，即使许多开发人员经常错误地使用同步和volatile，但是因为部署的处理器架构能提供很强的存储能力，所以得以侥幸避免出错。 ”这面这段引用是他这个月所作的讲话的一个摘要。我真希望我能去听讲。随着越来越多的电脑拥有了多个多核处理器，我们以前依赖的内存行为开始消失了，我们这些Java开

发人员也从以前那种侥幸的、错误的方式中醒悟过来。
51CTO编辑推荐：哪种语言将统治多核时代 再看函数式语言特性 这一切的失败仿佛早就注定。我们学习Java的许多方式，不管是通过示例程序，课堂教学或是书籍，都导致了这些错误。所以我们有了根深蒂固的错误而且危险的思维习惯，并且还导致我们以一种错误的方式来思考程序……这也导致了在部署过程中发生一些不起眼的错误，这些错误日后极难被找出和修复。 难道只有多线程程序中才有这个问题？还有，专家所写的代码中难道不会少一点这些问题吗？不是这样的。想想使用Swing图形界面工具包开发的程序。或者，更常见的，运行在Servlet容器中的一个服务端程序。在这种两种情况下，你都被迫进入了多线程环境。 我想我们大多数人都对如何处理竞争条件和死锁以及如何用同步解决问题有好的办法，特别是当涉及到多个变量时。这些是很难学会的，但是长期以来好的例子，还有大环境的影响，让我们在理解和运用多线程编程上走上正轨。 即便如此，像双检测锁定这样一些烂方法的存在说明我们的理解还有问题。而我们的东西在什么地方散架也只是一个明不明显的问题。如果不合适地使用同步或volatile来声明变量，即使是很稳定不变的的值也有可能在其它线程中完全看不到。我们所理解的一个像共享池一样的主存实在是过于简化了，这导致我们对于代码将有哪些行为和哪些是安全的会作出错误的判断。 最近我们的一个产品在用户多核处理器上的站点出现了问题，这让我对这一点更加刻骨铭心。这个问题我们自己没法重现，在测试中也没出现过，但是通过禁用一个处理器，这个问题就可以解决。此后不久，我的一位同事根据他在《Java并发编程实践》中

所读到的一些内容为当地的Java用户组作了一次演讲，这让我想起来我一直想读这本书，我也该抽时间读读这本书了。 我也这样做了。这还真是让我大开眼界。它明确了一些我以前在JavaOne以及网上听说过的，却从来没有融会贯通的一些问题。它让我更明白了Java代码的真正意义，以及如何写安全的Java代码。我们团队一直一起阅读这本书，有好几个星期都在午餐后讨论这本书，它让我们能修正我们的Java代码，使其在多处理器上也正常运行。多处理器曾经很昂贵并且不稳定，但现在正成为主流。所以这并不能说是为时尚早我们还希望能在之前就深入了解这些呢。将所学的这些灵活运用，让我们建立起写无误代码的信心还需要一些时间，但至少我们走上了正轨。 每个写Java的人都应该读读这本书，为了他们自己，还有他们的用户。Brian Goetz为IBM developerWorks所写的两篇文章（一、二）可以作为我这里所提到的这些问题的一个总结，但是那并不能代替拥有大量分析、建议还有范例的《Java并发编程实践》。 不过，即使我们在发现这些问题上变得很厉害了，这问题也还是不那么容易解决。这也促使我思考并认真看看Scala，它是一门较新的语言，可以在 JVM 中和其它的语言一起运行，并通过更可靠和可信的函数式编程平台解决棘手的并行编程部分，函数式编程里大多数都是不可变的对象。Scala那种无共享，基于角色的消息传递并行机制是源于运行在JVM外的Erlang语言（可参考51CTO之前发布的Scala和Erlang，以及多核主导的未来一文），这种机制很有前景。但是，这又是今后的另一个话题了。 