java并发编程实践笔记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_java_E5_B9_B6_E5_8F_91_c104_644777.htm 1, 保证线程安全的三种方法: a, 不要跨线程访问共享变量b, 使共享变量是final类型的c, 将共享变量的操作加上同步 2, 一开始就将类设计成线程安全的, 比在后期重新修复它,更容易. 3, 编写多线程程序, 首先保证它是正确的, 其次再考虑性能. 4, 无状态或只读对象永远是线程安全的. 5, 不要将一个共享变量裸露在多线程环境下(无同步或不可变性保护) 6, 多线程环境下的延迟加载需要同步的保护, 因为延迟加载会造成对象重复实例化 7, 对于volatile声明的数值类型变量进行运算, 往往是不安全的(volatile只能保证可见性, 不能保证原子性).详见volatile原理与技巧中, 脏数据问题讨论. 8, 当一个线程请求获得它自己占有的锁时(同一把锁的嵌套使用), 我们称该锁为可重入锁.在jdk1.5并发包中, 提供了可重入锁的java实现-ReentrantLock. 9, 每个共享变量,都应该由一个唯一确定的锁保护.创建与变量相同数目的ReentrantLock, 使他们负责每个变量的线程安全. 10,虽然缩小同步块的范围, 可以提升系统性能.但在保证原子性的情况下, 不可将原子操作分解成多个synchronized块. 11, 在没有同步的情况下, 编译器与处理器运行时的指令执行顺序可能完全出乎意料.原因是, 编译器或处理器为了优化自身执行效率, 而对指令进行了的重排序(reordering). 12, 当一个线程在没有同步的情况下读取变量, 它可能会得到一个过期值, 但是至少它可以看到那个线程在当时设定的一个真实数值. 而不是凭空而来的值. 这种安全保证, 称之为最低限的安全性(out-of-thin-air safety) 在开发并发应用

程序时, 有时为了大幅度提高系统的吞吐量与性能, 会采用这种无保障的做法.但是针对, 数值的运算, 仍旧是被否决的. 13, volatile变量,只能保证可见性, 无法保证原子性. 14, 某些耗时较长的网络操作或IO, 确保执行时, 不要占有锁. 15, 发布(publish)对象, 指的是使它能够被当前范围之外的代码所使用.(引用传递)对象逸出(escape), 指的是一个对象在尚未准备好时将它发布. 原则: 为防止逸出, 对象必须要被完全构造完后, 才可以被发布(最好的解决方式是采用同步) this关键字引用对象逸出 例子: 在构造函数中, 开启线程, 并将自身对象this传入线程, 造成引用传递.而此时, 构造函数尚未执行完, 就会发生对象逸出了. 16, 必要时, 使用ThreadLocal变量确保线程封闭性(封闭线程往往是比较安全的, 但一定程度上会造成性能损耗)封闭对象的例子在实际使用过程中, 比较常见, 例如 hibernate openSessionInView机制, jdbc的connection机制. 17, 单一不可变对象往往是线程安全的(复杂不可变对象需要保证其内部成员变量也是不可变的)良好的多线程编程习惯是: 将所有的域都声明为final, 除非它们是可变的 