- 基本概念
  - 对象的状态是指存储在状态变量(如实例或静态域)中的数据
  - 共享意味着变量可以由多个线程同时访问
  - 可变意味着变量的值在其生命周期内可以发生变化
  - 一个对象是否需要线程安全,取决于它是否需要被多个线程访问,指的是对象被访问的方式,而不是对象要实现的功能
  - Java中的同步机制
    - synchronized,独占锁
    - volatile
    - Explicit Lock,显式锁
    - 原子变量
  - 防止出现多线程错误的方式
    - 不在线程之间共享该状态变量
    - 将状态变量修改为不可变的变量
    - 在访问状态变量时使用同步
- 原则与方法
  - 首先使代码正确的运行,然后再提高代码的速度
  - 当性能测试结果和应用需求告诉你必须提高性能,以及测量结果表明这种优化在实际环境中确实能带来性能提升时,才进行优化
- 线程安全性
  - 正确性的含义是,某个类的行为与其规范完全一致
  - 当多个线程访问某个类时,这个类始终都能表现出正确的行为,那么就称这个类是线程安全的
    - 不管运行时环境采用何种调度方式
    - 不管这些线程将如何交替执行
    - 在主调代码中不需要任何额外的同步或协同
    - 类的行为都是正确的
  - 无状态对象一定是线程安全的

    - 竞态条件
        - 与数据竞争(data race)是不同的概念
        - 两个可以同时存在
    - 复合操作
        - 包含一组必须以原子方式执行的操作以确保线程的安全性
        - 在一个无状态的类中添加一个状态时,如果该状态完全由线程安全的对象来管理,那么这个类仍然是线程安全的(由一个变为多个时,不一定如此)
- 加锁机制
    - 要保持状态的一致性,就需要在单个原子操作中更新所有相关的状态变量
    - 内置锁
        - 同步代码块
            - 作为锁的对象的引用
            - 由这个锁保护的代码块
        - 重入
            - 内置锁是可重入的
            - 某个线程试图获取一个已经由它自己持有的锁,那么这个请求就会成功
            - 获取锁的操作的粒度是线程而不是调用
- 用锁来保护状态
    - 对于可能被多个线程同时访问的可变状态变量,在访问它时都需要持有同一个锁
    - 每个共享的和可变的变量都应该只由一个锁来保护,从而使得维护人员知道是哪一个锁
    - 一种常见的加锁约定
        - 所有可变的状态都封装在对象内部
        - 通过对象内置锁对所有访问可变状态的代码路径进行同步
        - 在该对象上可变状态不会发生并发访问
    - 在不变性条件中的每个变量都必须由同一个锁来保护
    - 单个操作的原子性不能保证复合操作也是原子的
- 活跃性与性能
    - 不良并发应用程序:可同时调用的数量,不仅受到可用资源的限制,还受到应用程序本身

结构的限制

- 如果持有锁的时间过长,那么都会带来活跃性或者性能问题
- 当执行较长时间的计算或者可能无法快速完成的操作时,(例如网络IO或控制台IO),一定不要持有锁
-