JVM-线程安全与锁优化

一，线程安全

当一个对象被多个线程访问的时候，如果不考虑这些线程在运行时环境下的调度和交替运行，也不需要额外的同步，或者在调用方法进行任何其他的协调操作的时候，调用这个对象的行为都可以获得正确的结果，那么这个对象就是线程安全的。

二，线程安全的实现方法

- 互斥同步
  - 该方法是一种常见的并发正确性保障手段，同步是指在多个线程并发访问共享数据的时候，共享的数据在同一时刻只能被一个线程所调用，互斥是实现同步的实现方式，在这里面，互斥是因，同步是果。互斥是方法，同步是目的。
  - 使用synchronized关键字可以实现互斥同步。
  - 使用ReentrantLock关键字实现互斥同步。

    等待可中断:正在等待的线程可以选择放弃等待，改为处理其他事情。

    公平锁：多个线程等待同一个锁时，必须按照申请锁的时间的顺序来依次获得锁。但是其默认值是不公平的，Synahronized也是不公平的，任意等待的线程获取所得顺序是随机的，但是ReentrantLock可以通过设置布尔值的构造函数要求i使用公平锁。

    绑定多个对象：一个ReentrantLock对象可以同时绑定多个Conditon对象。
- 非阻塞同步

  互斥同步最主要的问题就是进程阻塞和唤醒所带来的性能问题，属于一种悲观的并发策略总是认为只要不加锁，肯定就会出现问题。

  CAS操作

  - CAS有3个操作数，内存值V，旧的预期值A，要修改的新值B。当且仅当预期值A和内存值V相同时，将内存值V修改为B，否则什么都不做。
  - 缺点

    CAS看起来很美，但这种操作显然无法涵盖并发下的所有场景，并且CAS从语义上来说也不是完美的，存在这样一个逻辑漏洞：如果一个变量V初次读取的时候是A值，并且在准备赋值的时候检查到它仍然是A值，那我们就能说明它的值没有被其他线程修改过了吗？如果在这段期间它的值曾经被改成了B，然后又改回A，那CAS操作就会误认为它从来没有被修改过。这个漏洞称为CAS操作的”ABA”问题。
    java.util.concurrent包为了解决这个问题，提供了一个带有标记的原子引用类”AtomicStampedReference”，它可以通过控制变量值的版本来保证CAS的正确性。不过目前来说这个类比较”鸡肋”，大部分情况下ABA问题并不会影响程序并发的正确性，如果需要解决ABA问题，使用传统的互斥同步可能回避原子类更加高效。