[Java并发编程：Lock - Matrix海子 - 博客园 (cnblogs.com)](https://www.cnblogs.com/dolphin0520/p/3923167.html) 参考文章

Synchronized关键字和lock

这两者的区别是？

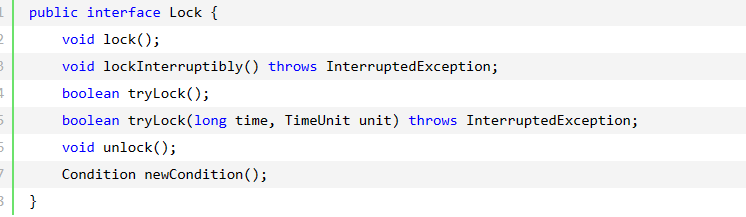
不用锁能怎么保证线程安全？

什么情况适合用哪个？🡪**竞争激烈，且可能会频繁失败的场景，不太适合自旋尝试获取乐观锁（cpu开销太大了）**

能设置过期时间吗？🡪**实际上并不能设置具体过期时间，只能是“tryLock()，获取锁失败的情况就返回，不执行对应的逻辑（如进行一次insert操作）”**

**Synchronized**用于修饰代码块，在代码块执行结束或者发生异常之前，均不会将锁释放，且其他竞争同一个锁的线程会陷入阻塞状态🡪且这个阻塞等待过程不能被中断

**Lock**则不是关键字，而是一个接口/类，而且不像synchronized关键字，不会自动释放锁（因此一般会配合trycatch语句使用，并在finally中进行锁的释放，**以避免死锁**）



方法的不同：lock()方法会阻塞等待锁的释放再获取；tryLock()则会在上锁失败/成功后立即返回，或者设置一个获取锁的超时时间；lockInterruptibly()则是“可打断的阻塞获取”，可以调用线程对象的**interrupt()方法**（和多线程有关，需要复习）中断阻塞等待过程

**ReentrantLock(可重入锁)**：

和synchronized关键字一样，都是可重入锁（即同一个线程可以多次获取同一个锁）；但ReentrantLock可以决定是否为“公平锁”/“非公平锁”🡪公平锁是“阻塞队列中最早进去的线程更快获取锁”；非公平锁则是“锁可获取后，所有等待该锁的线程一起竞争”

**不用锁怎么保证线程安全？🡪乐观锁**：

CAS（compare and swap）操作，指的是将“当前值”和“之前读取时的值”进行比较，相同的情况才进行写操作【并且该操作需要有“**原子性**”】🡪通过加版本号可以解决**“aba问题”**

可以基于类似下面这种的“**原子类**”实现

