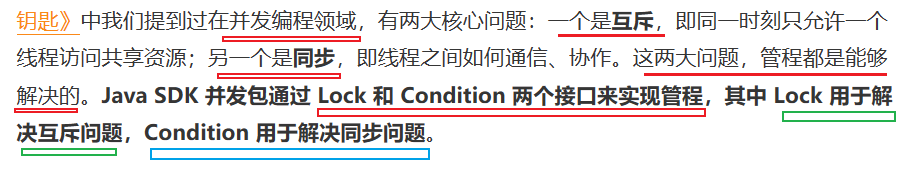
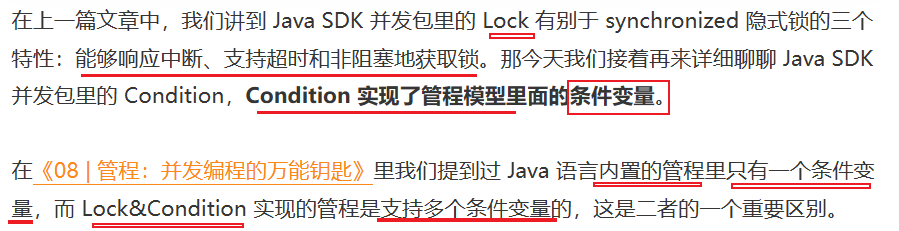
# Lock（上）

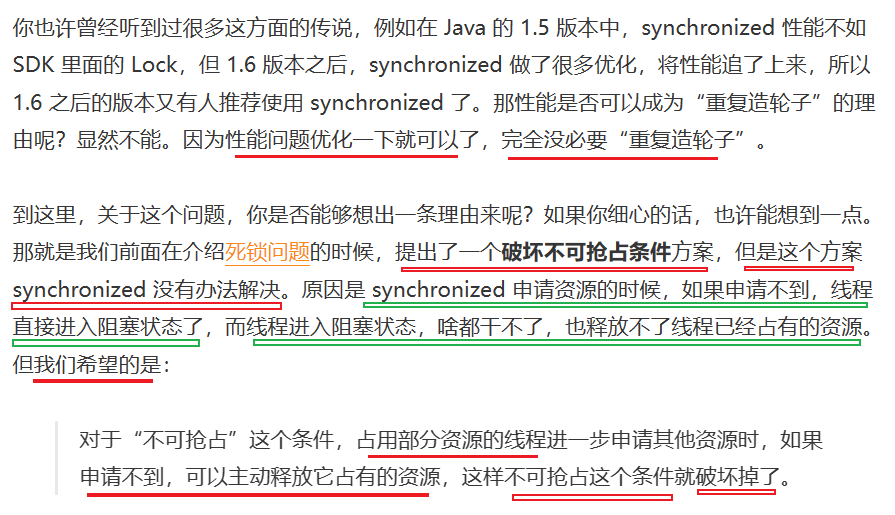
## Lock&Condition与synchronized的对比

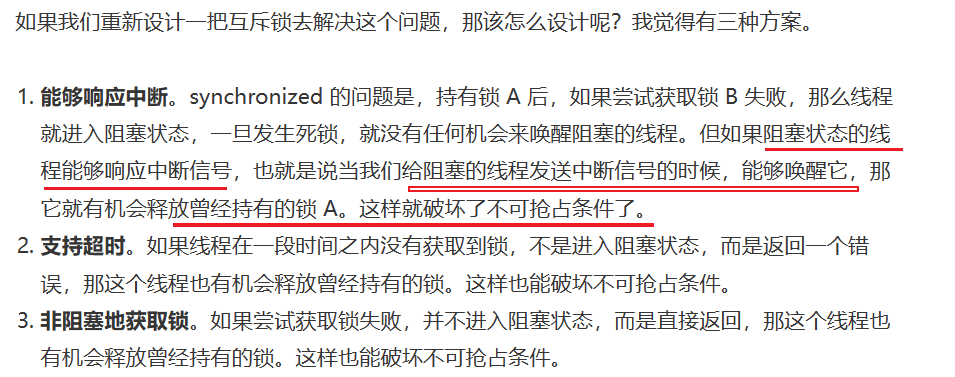
并发包其实就是管程的另一种实现方式





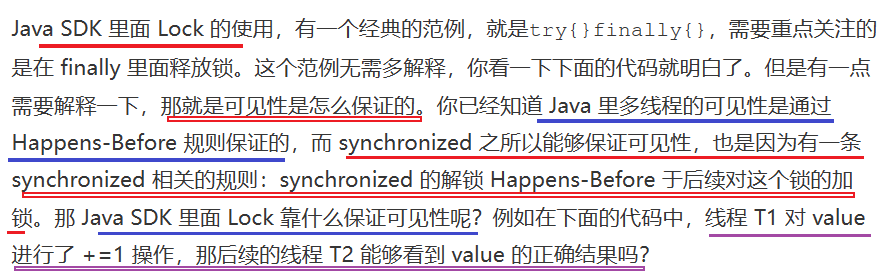
## 再造管程的理由

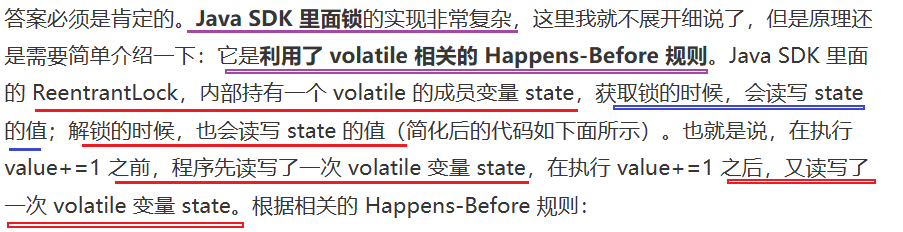


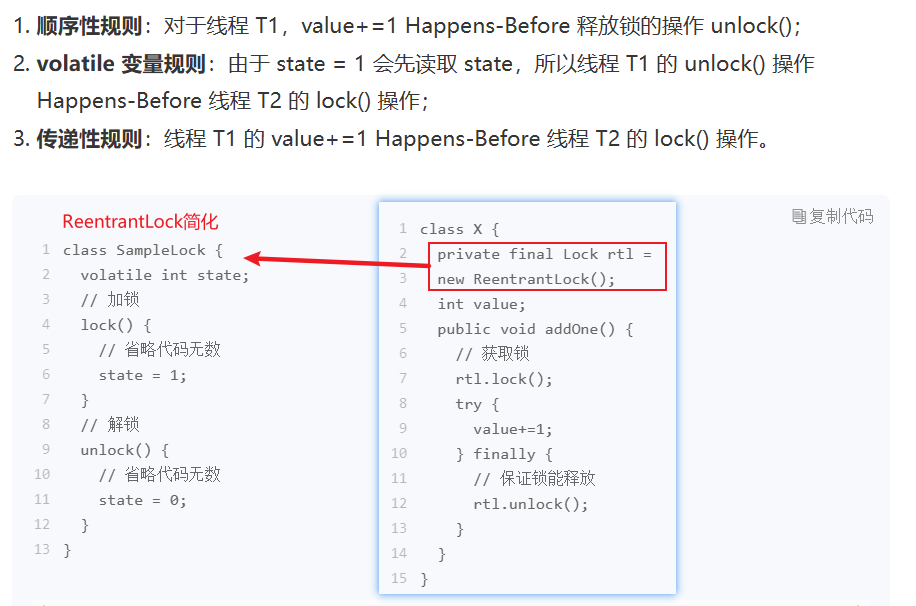




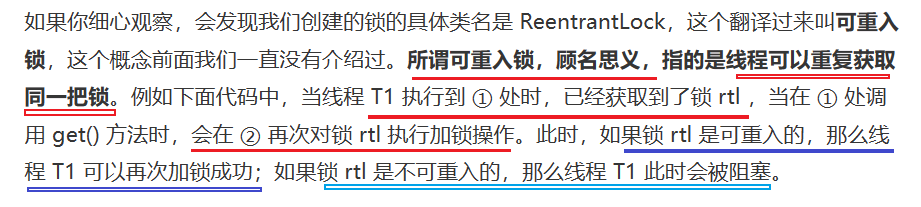
## 如何保证可见性

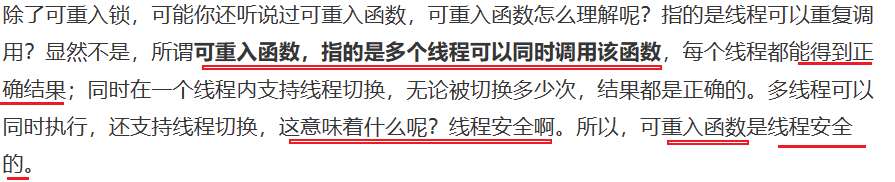
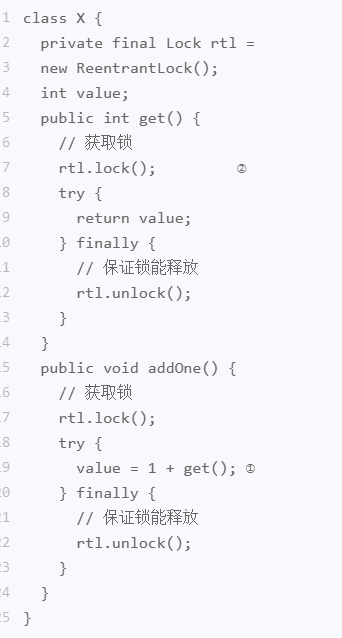




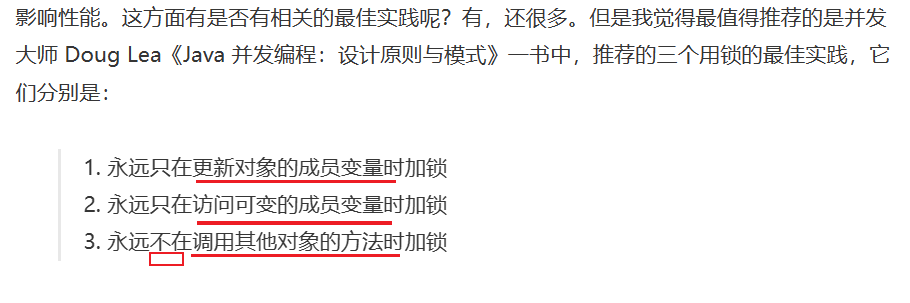


## 什么是可重入锁





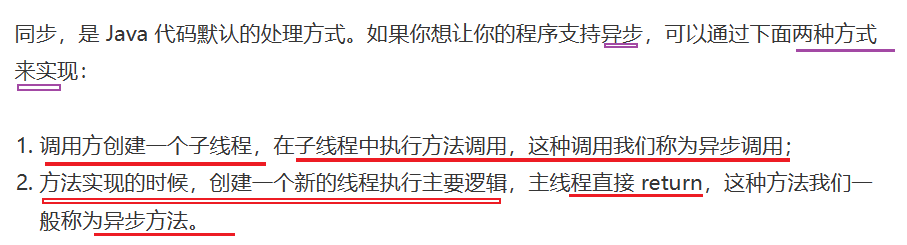
## 实践准则



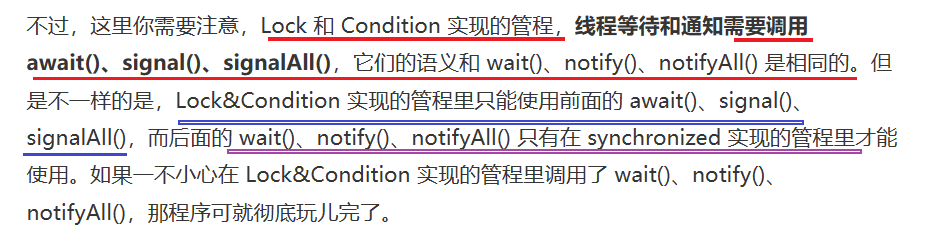
# Condition（下）

## 同步与异步





## Lock&Condition总结

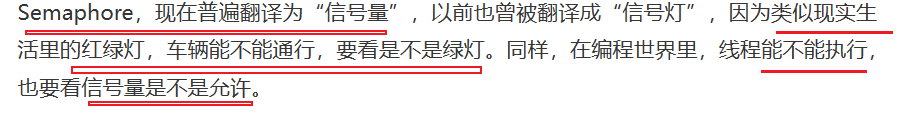
 **所谓可重入锁，顾名思义，指的是线程可以重复获取同一把锁**。

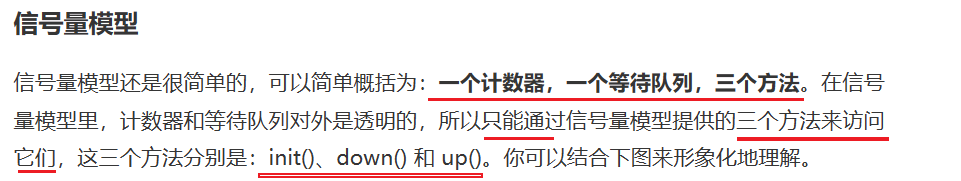
Lock&Condition 是管程的一种实现，所以能否用好 Lock 和 Condition 要看你对管程模型理解得怎么样。

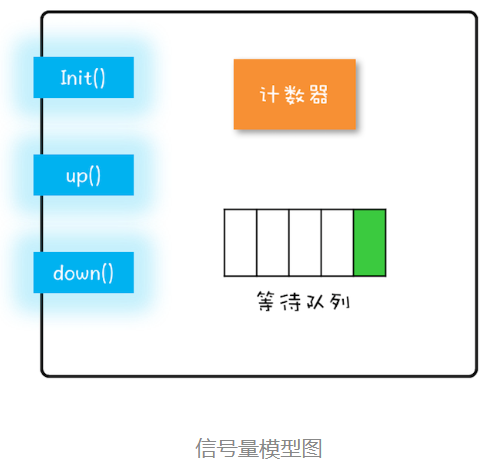
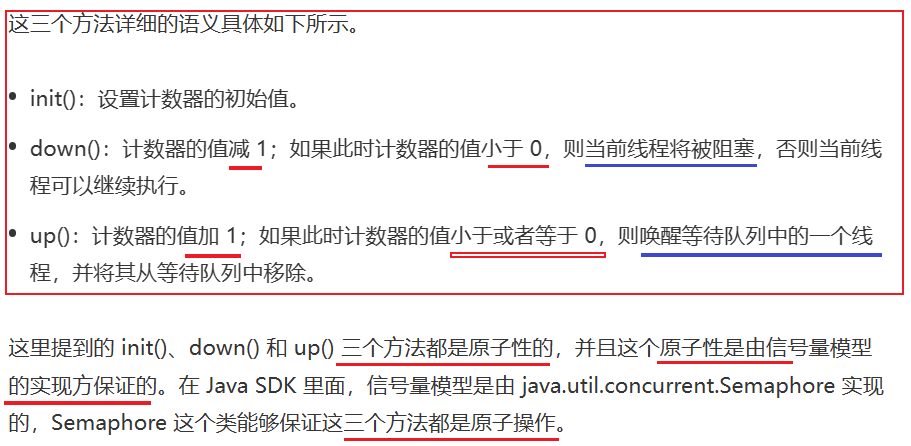
[《Java 并发编程的艺术》](time://mall?url=https%3A%2F%2Fh5.youzan.com%2Fv2%2Fgoods%2F35z7jjvd4r4oo)一书的第 5 章《Java 中的锁》，里面详细介绍了实现原理，我觉得写得非常好。

Lock&Condition 实现的管程相对于 synchronized 实现的管程来说更加灵活、功能也更丰富。

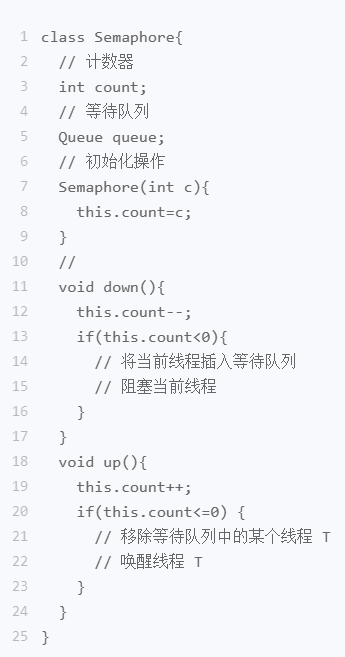
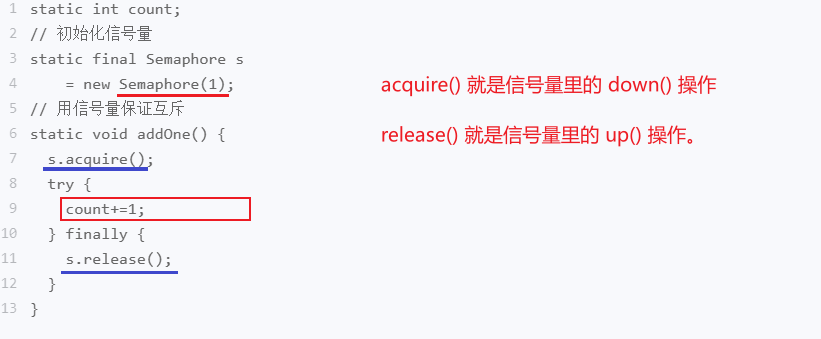
# 16 | Semaphore：如何快速实现一个限流器？





## 如何使用信号量



下面我们再来分析一下，信号量是如何保证互斥的。

假设两个线程 T1 和 T2 同时访问 addOne() 方法，当它们同时调用 acquire() 的时候，由于 acquire() 是一个原子操作，所以只能有一个线程（假设 T1）把信号量里的计数器减为 0，另外一个线程（T2）则是将计数器减为 -1。

对于线程 T1， 信号量里面的计数器的值是 0，大于等于 0，所以线程 T1 会继续执行； 对于线程 T2，信号量里面的计数器的值是 -1，小于 0，按照信号量模型里对 down()

操作的描述，线程 T2 将被阻塞。

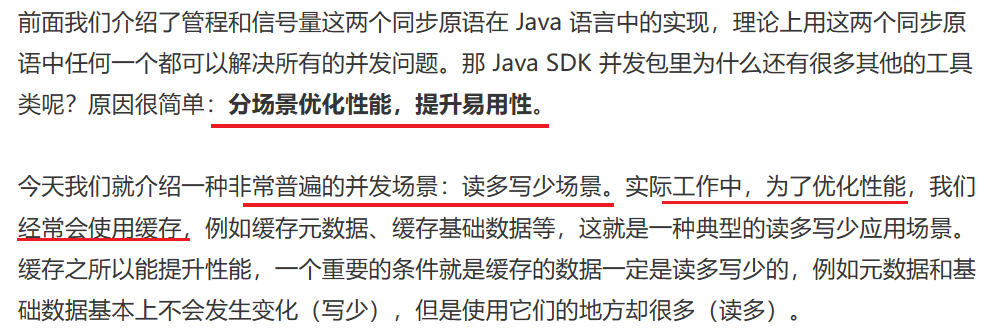
所以此时只有线程 T1 会进入临界区执行count+=1；。

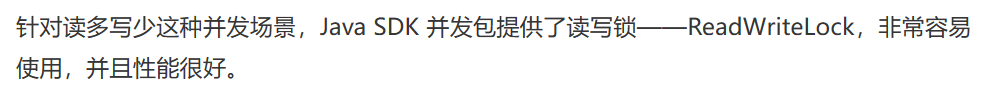
当线程 T1 执行 release() 操作，也就是 up() 操作的时候，信号量里计数器的值是 -1，加 1 之后的值是 0，小于等于 0，按照信号量模型里对 up() 操作的描述，此时等待队列中的 T2 将会被唤醒。

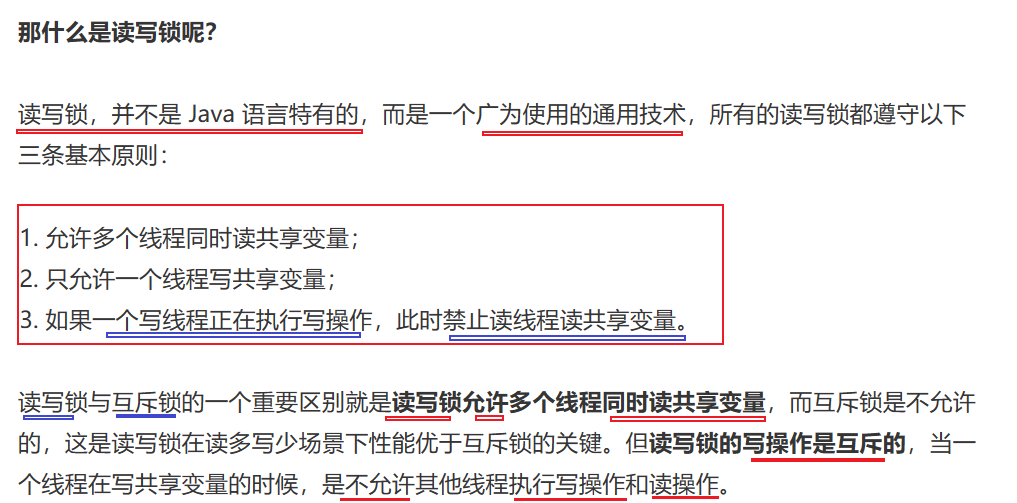
于是 T2 在 T1 执行完临界区代码之后才获得了进入临界区执行的机会，从而保证了互斥性。

# 17 | ReadWriteLock：如何快速实现一个完备的缓存？

## 读写锁原理



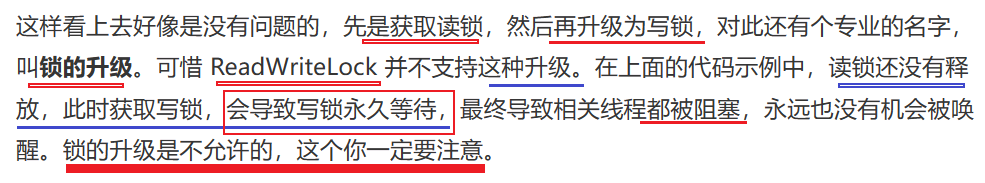




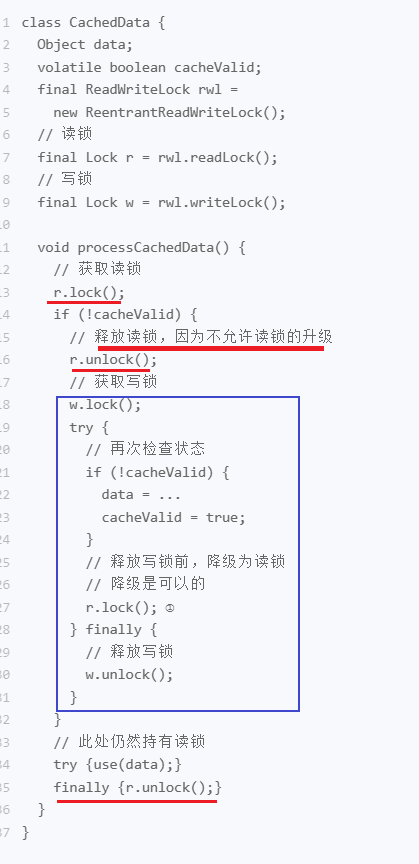
也就是说 **读锁与写锁是互斥的**

## 锁的升级

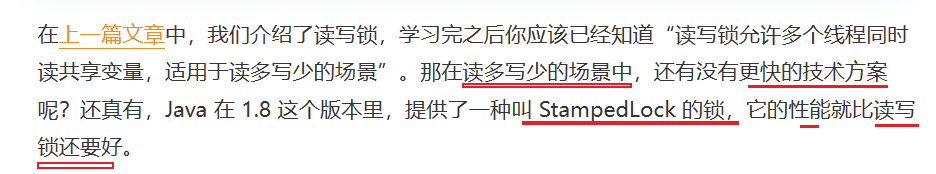




## 锁降级



# 18 | StampedLock：有没有比读写锁更快的锁？



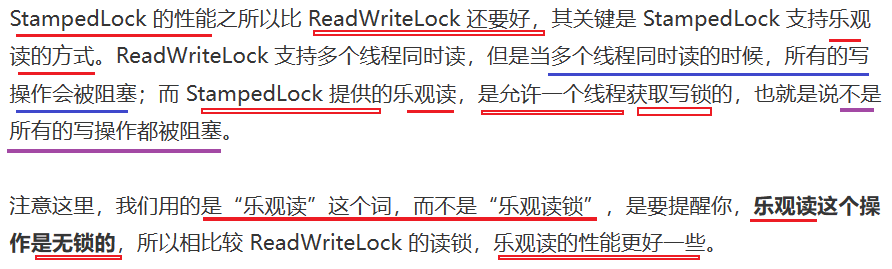
## 与读写锁的区别

使用上 StampedLock 和ReadWriteLock 有哪些区别。

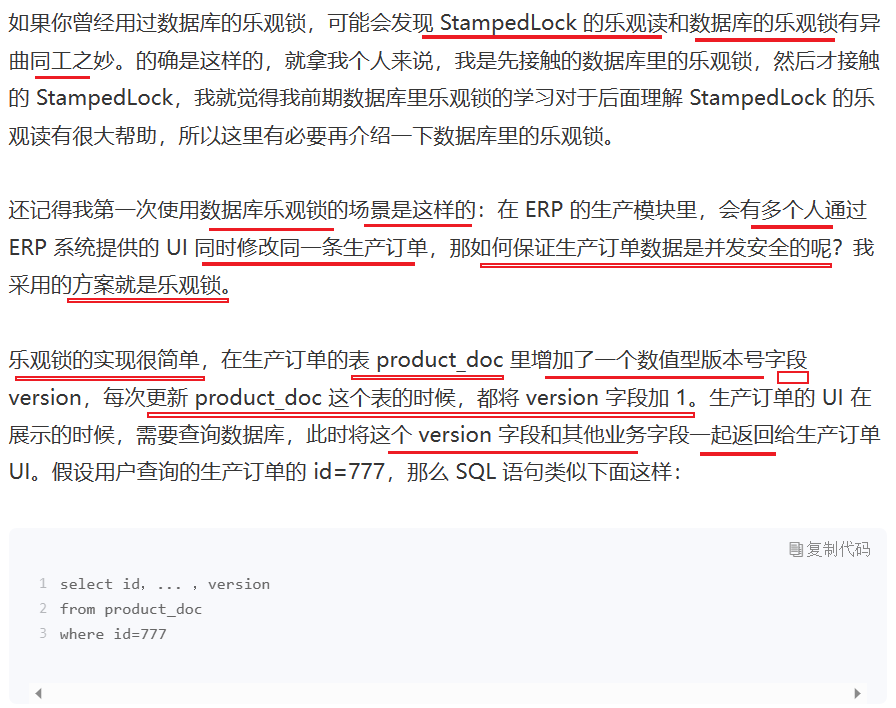
ReadWriteLock 支持两种模式：一种是读锁，一种是写锁。而 StampedLock 支持三种模式，分别是：**写锁、悲观读锁和乐观读**。

**相同点时**：写锁、悲观读锁的语义和 ReadWriteLock 的写锁、读锁的语义非常类似，允许多个线程同时获取悲观读锁，但是只允许一个线程获取写锁，写锁和悲观读锁是互斥的。

**不同的是**：StampedLock 里的写锁和悲观读锁加锁成功之后，都会返回一个 stamp；然后解锁的时候，需要传入这个 stamp。相关的示例代码如下。



## 如何理解乐观读



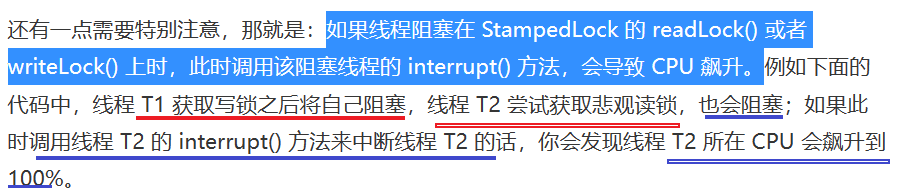


## 注意

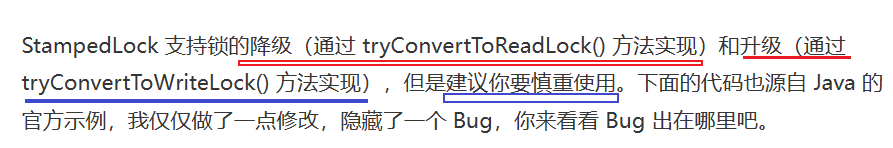
1.StampedLock 的功能仅仅是 ReadWriteLock 的子集

2.StampedLock 不支持重入

3.StampedLock 的悲观读锁、写锁都不支持条件变量。







**读锁与写锁是互斥的**