

3）java.util.concurrent.lock包(锁)

# Semaphore

Semaphore 是 synchronized 的加强版，作用是控制线程的并发数量。就这一点而言，单纯的synchronized 关键字是实现不了的。

例如目前有三个线程A、B、C，线程B、C需要等到线程A初始化数据之后，线程B、C才能开始执行，这时就可以用到Semaphore

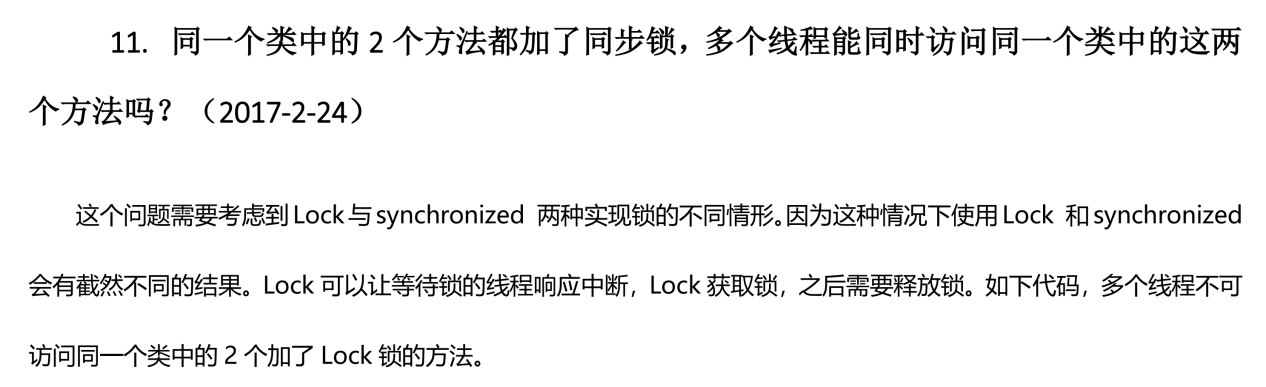
private Semaphore semaphore = new Semaphore(0);

线程A初始化完成后，就调用方法semaphore.release(2);相当于释放两个通路，此时B、C就可以执行了。

1. C执行时要调用方法：semaphore.acquire(1)，都申请一个通路。

这样就可以保证A先执行。

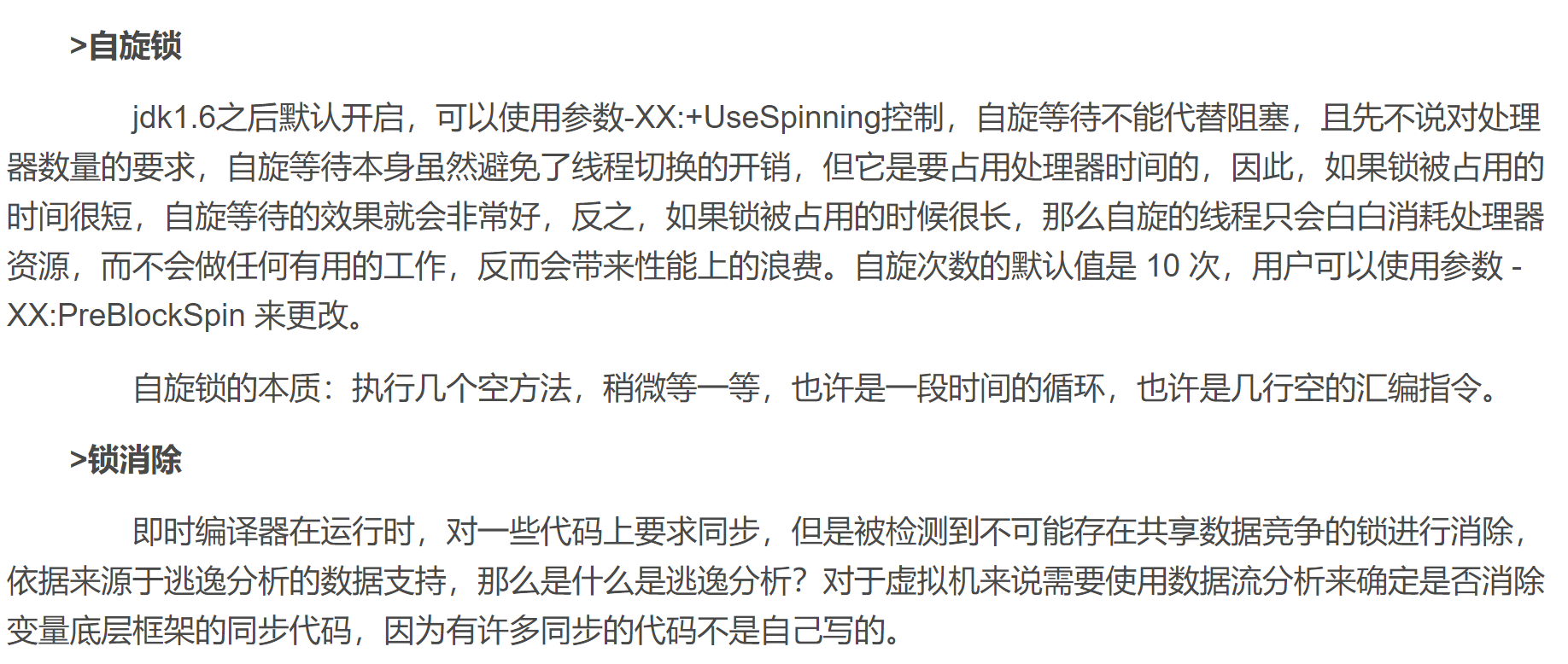
# Lock与synchronize

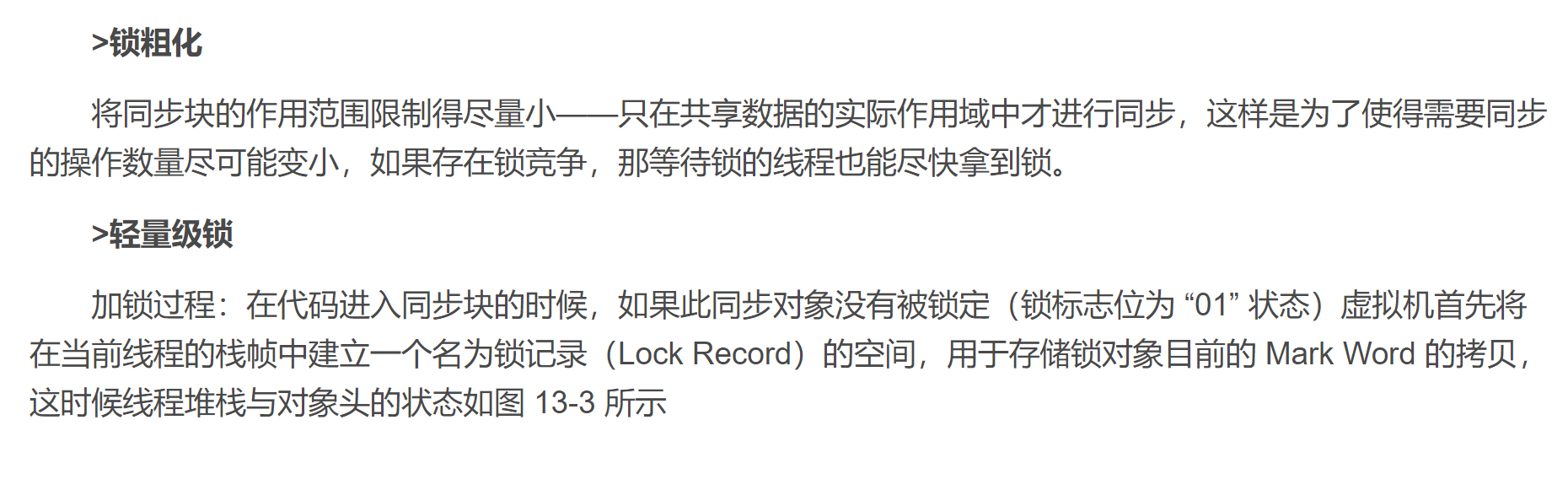


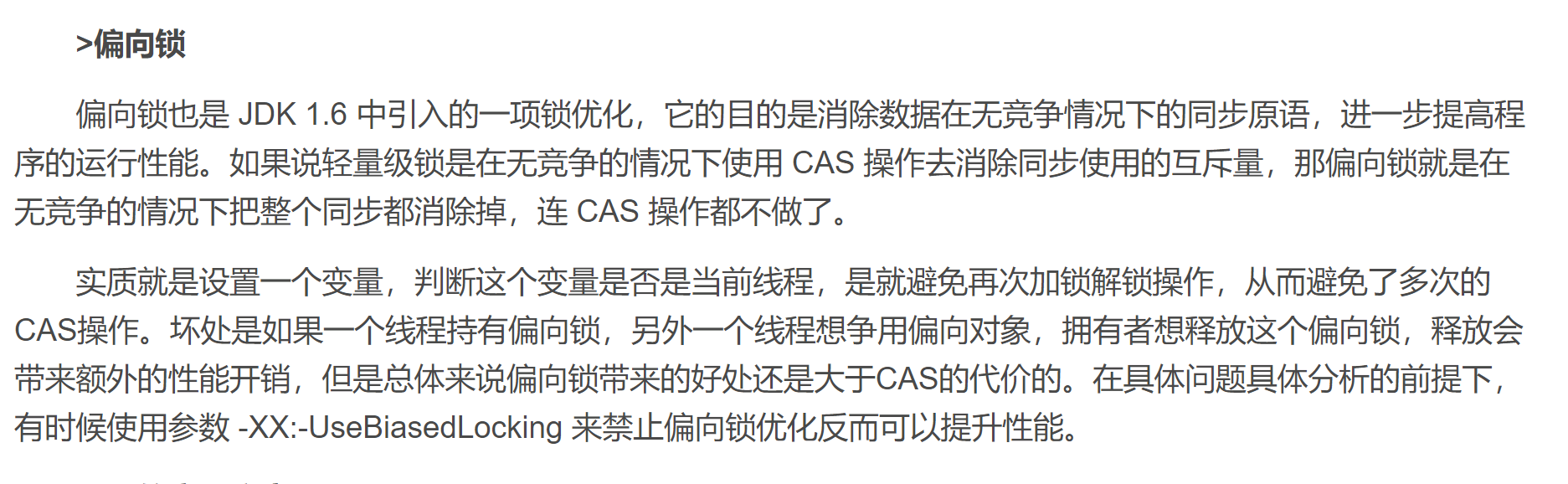
Synchronize：当我们访问同一个类对象的时候。相当于同一把锁。可以访问该对象的其他的synchronize方法。



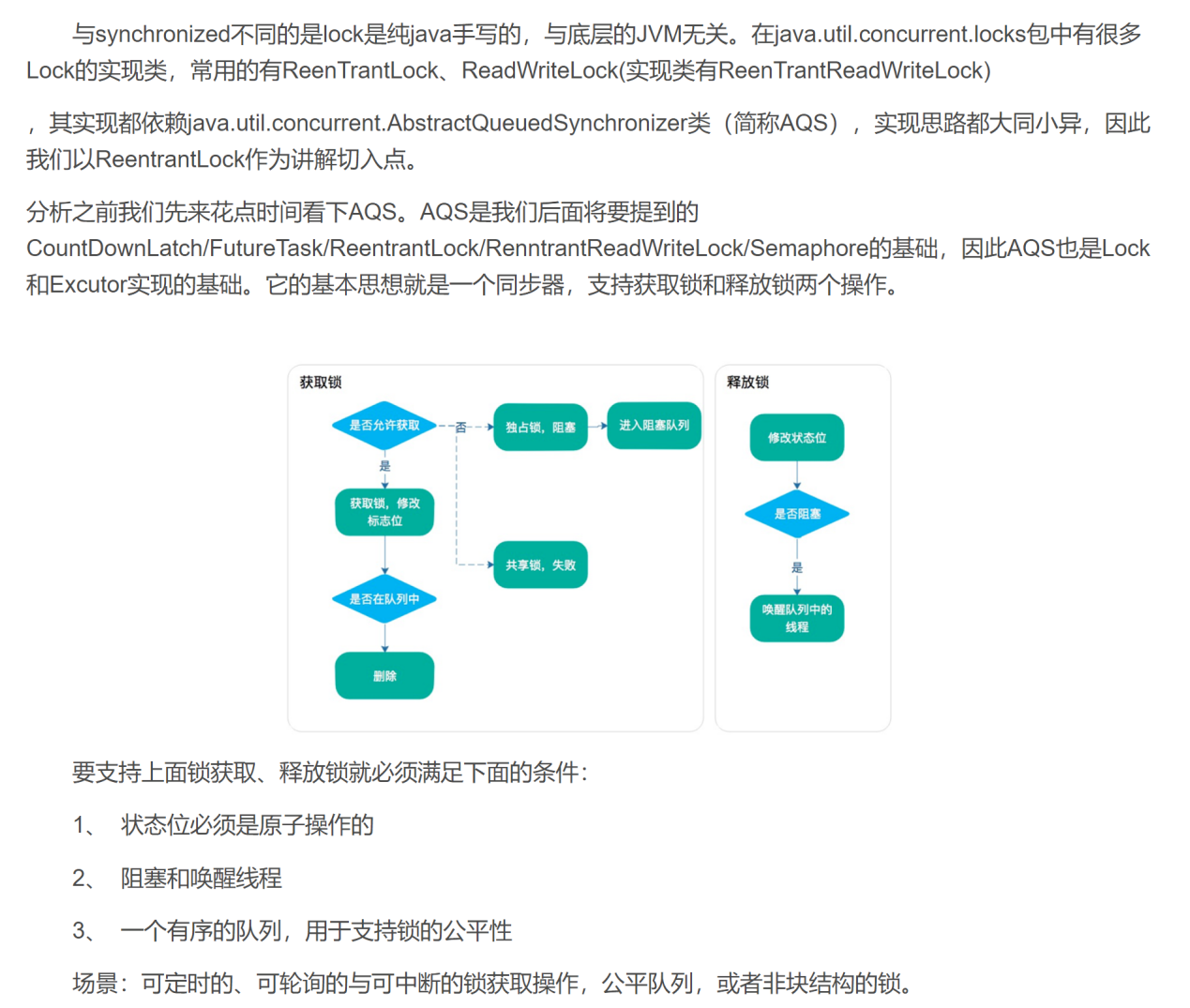
## Synchronize





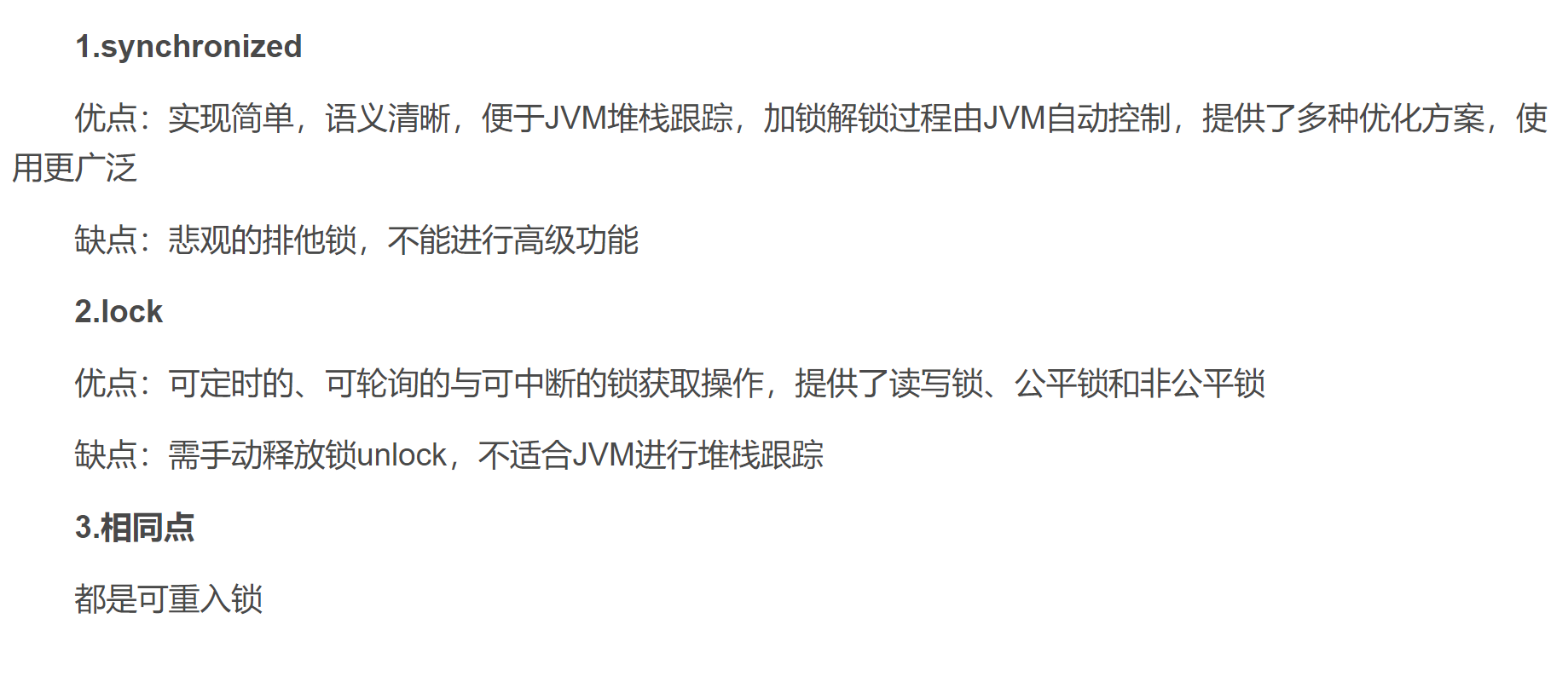


## Lock





## 异同



# 什么情况下会死锁，死锁了怎么办？

## 加锁顺序

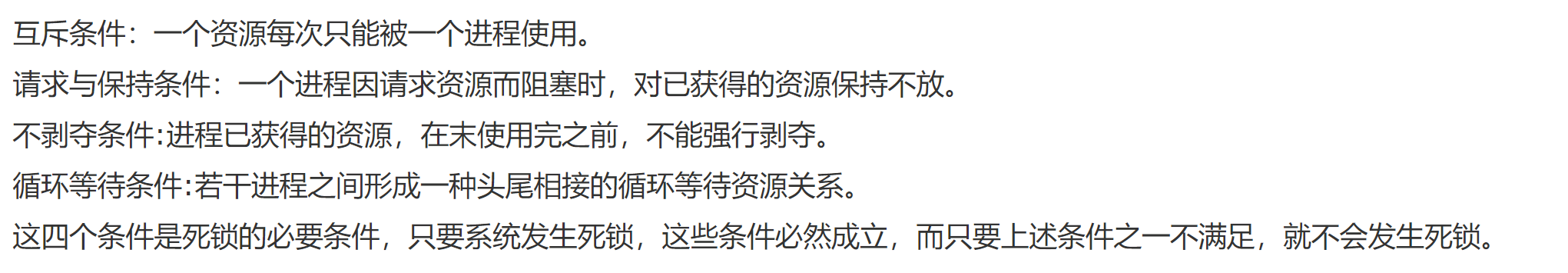
## 加锁时限

获取锁的时候添加一个时限，超过时限则放弃对锁的请求，并释放自己占有的锁。

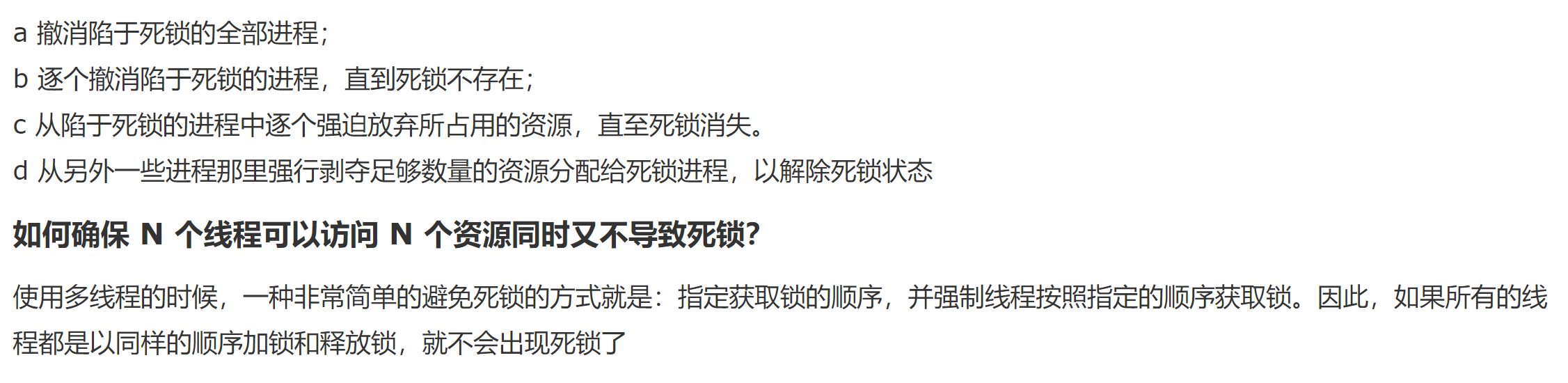
# 死锁

两个线程或者进程都在等待对方执行完毕之后才能继续执行，这样就陷入了无限的等待中。

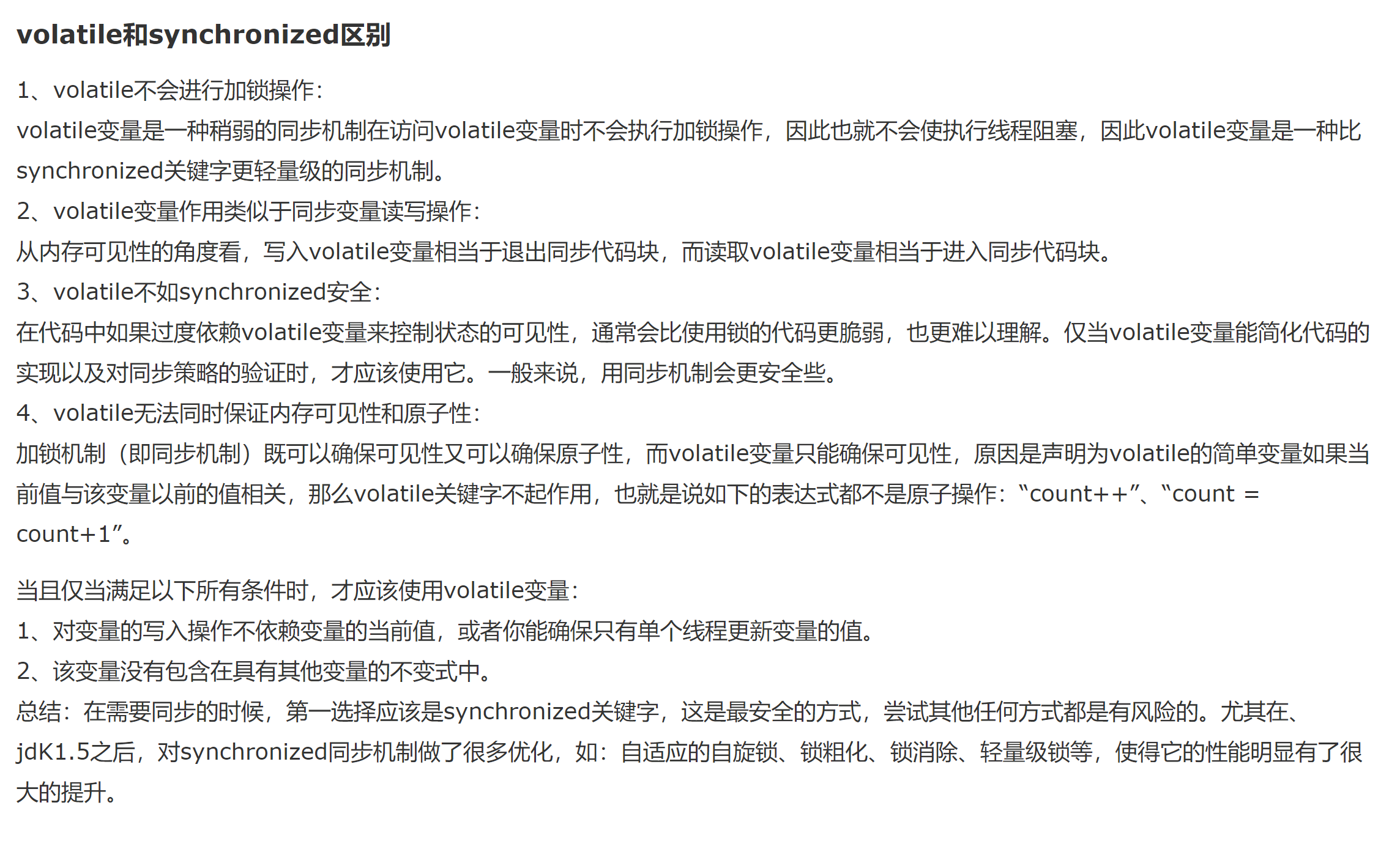
# 死锁的必要条件



# 死锁的解决办法

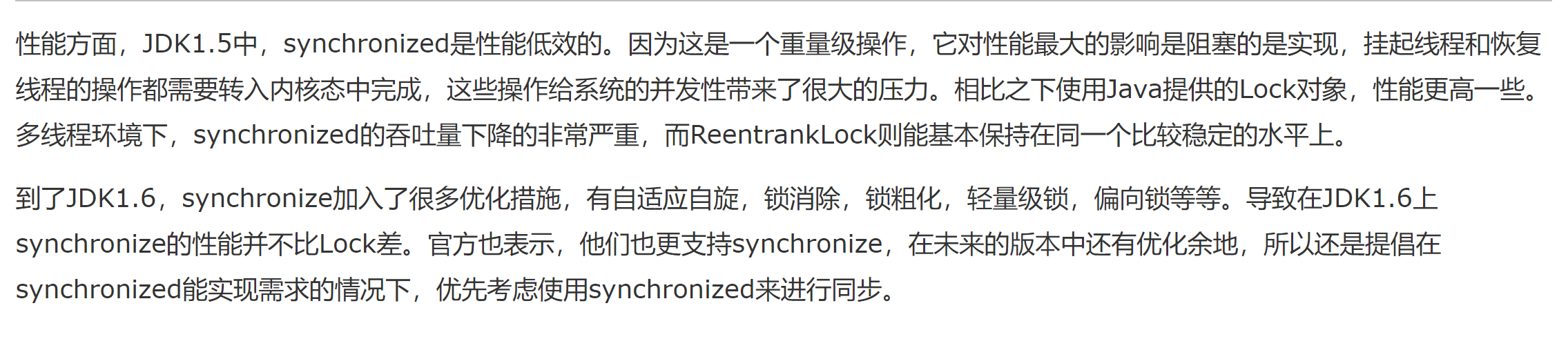


# Volatile和synchronized区别



# Synchronized和lock





# Synchronized [ˈsɪŋkrənaɪzd]和ReentrantLock

Synchronized，它就是一个：非公平，悲观，独享，互斥，可重入的重量级锁

ReentrantLock，它是一个：默认非公平但可实现公平的，悲观，独享，互斥，可重入，重量级锁。

ReentrantReadWriteLocK，它是一个，默认非公平但可实现公平的，悲观，写独享，读共享，读写，可重入，重量级锁。

如果使用 synchronized ，如果A不释放，B将一直等下去，不能被中断

如果使用ReentrantLock，如果A不释放，可以使B在等待了足够长的时间以后，中断等待，而干别的事情

## ReentrantLock获取锁的三种方式

lock(), 如果获取了锁立即返回，如果别的线程持有锁，当前线程则一直处于休眠状态，直到获取锁

tryLock(), 如果获取了锁立即返回true，如果别的线程正持有锁，立即返回false

tryLock(long timeout,TimeUnit unit)， 如果获取了锁定立即返回true，如果别的线程正持有锁，会等待参数给定的时间，在等待的过程中，如果获取了锁定，就返回true，如果等待超时，返回false；

lockInterruptibly:如果获取了锁定立即返回，如果没有获取锁定，当前线程处于休眠状态，直到获取锁定，或者当前线程被别的线程中断

## 选择依据

在资源竞争不是很激烈的情况下，Synchronized的性能要优于ReetrantLock，但是在资源竞争很激烈的情况下，Synchronized的性能会下降几十倍，但是ReetrantLock的性能能维持常态；

# 锁的分类

## 公平锁和非公平锁

公平锁指的是多个线程按照申请锁的顺序来获取锁。非公平锁指的是多个线程获取锁的顺序并不是按照申请锁的顺序。

对于ReentrantLock而言，通过构造函数指定该锁是不是公平锁，默认是给公平锁。

非公平锁的优点在于吞吐量比公平锁大，synchronized而言，是一种非公平锁。

## 乐观锁和悲观锁

乐观锁和悲观锁不是指具体什么类型的锁，而是指看待并发同步的角度。悲观锁认为对于同一个数据的并发操作，一定会发生修改。悲观锁就是加锁。

## 独享锁和共享锁

独享锁是指该锁一次只能被一个线程所持有。共享锁是该锁可以被多个线程持有。

对于ReentrantLock而言，是独享锁。

对于ReentrantReadWriteLock，读是共享锁，写是独享锁。

## 互斥锁和读写锁

ReentrantLock就是互斥锁，ReentrantReadWriteLock就是读写锁。

## 可重入锁

又名递归锁，就是进入method1时已经获取到锁，在method1中调用了method2（method2也有锁），此时进入method2时就不用再次获取锁。因为在method1中已经获取到了锁。

ReentrantLock和synchronized都是可重入锁。好处时可一定程度避免死锁。