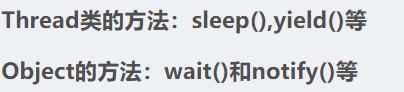


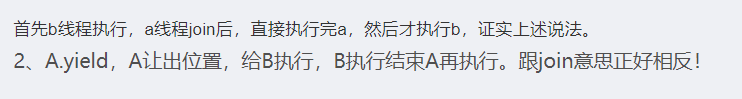
<https://www.jianshu.com/p/ea9d768b3f0b>

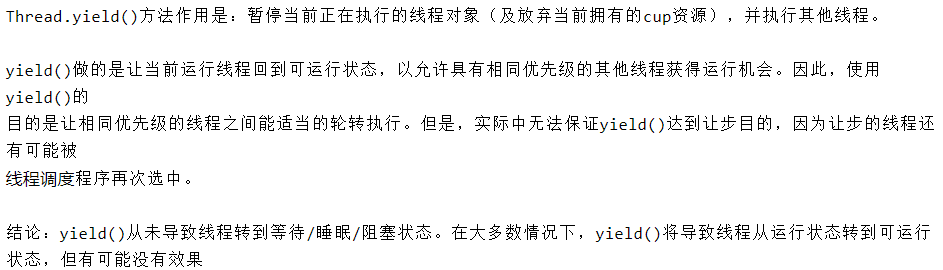


sleep/yield/join wait/notify/notifyAll

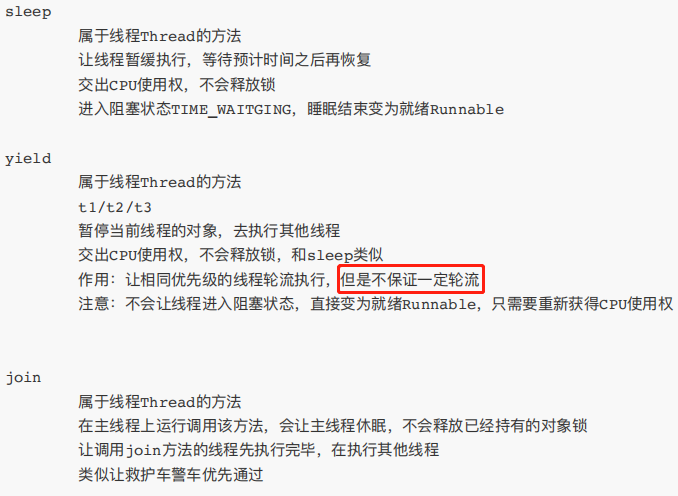
前三个是Thread方法，后三个是Object方法

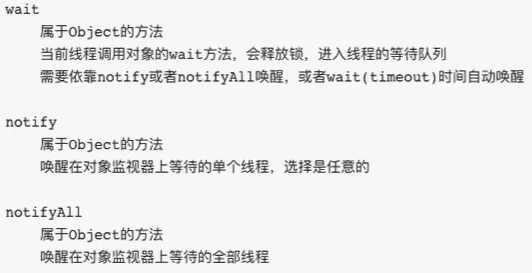
Yield直接运行到就绪，sleep和wait会阻塞



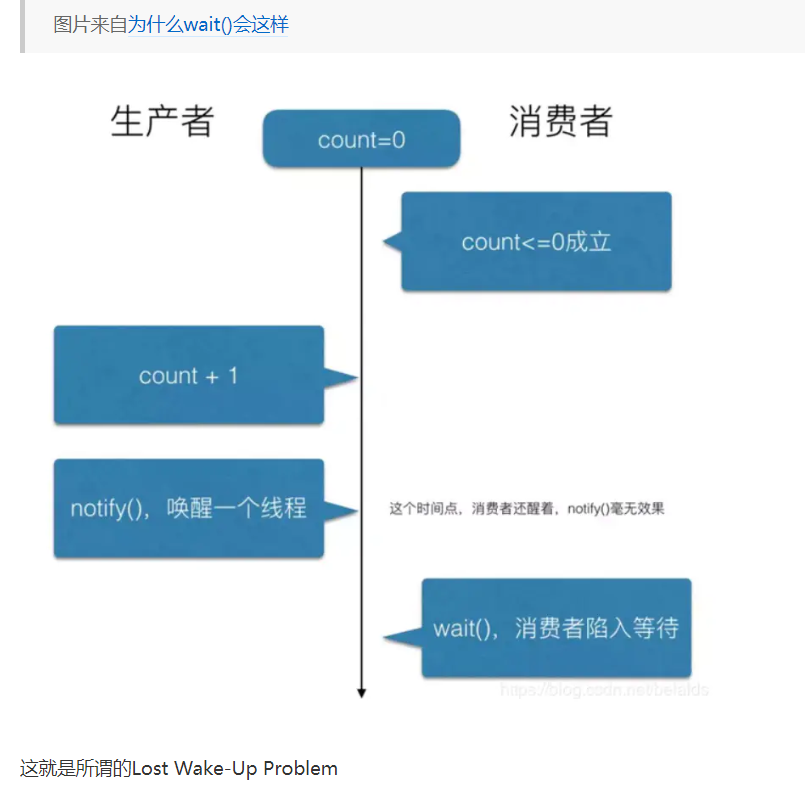


Join强行插队，yield让出当前线程





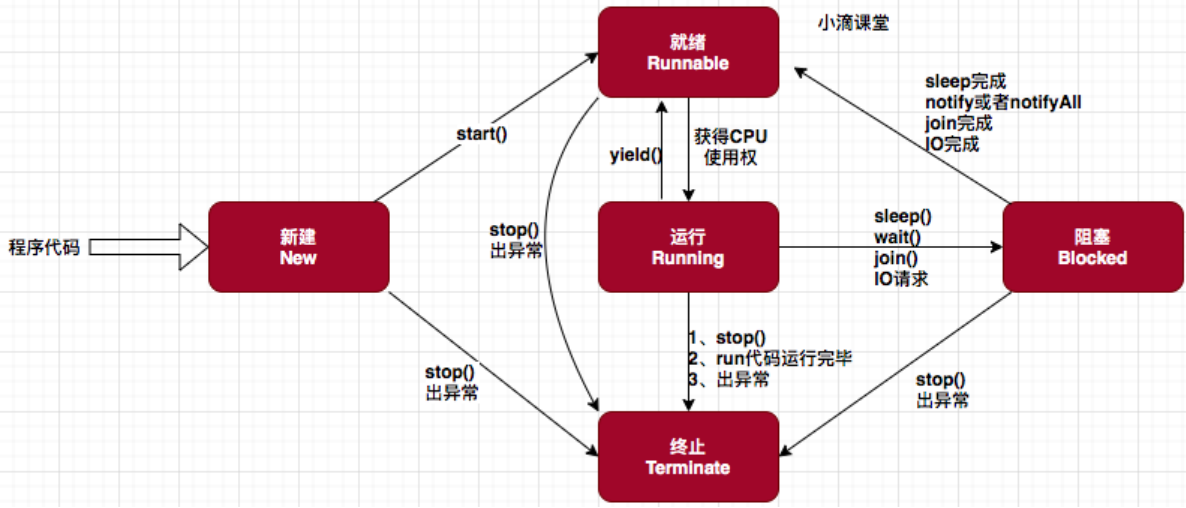


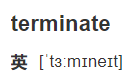


A,B,C三个线程,B先唤醒A线程还没操作完剩下的，C又使得A线程进入等到状态。此时对于A来说就是丢失唤醒。加了syn等B操作完C才能让A等待，此时才不会出现

生产者唤醒的时候锁住，操作完成再释放锁，就不会造成lost wake-up了

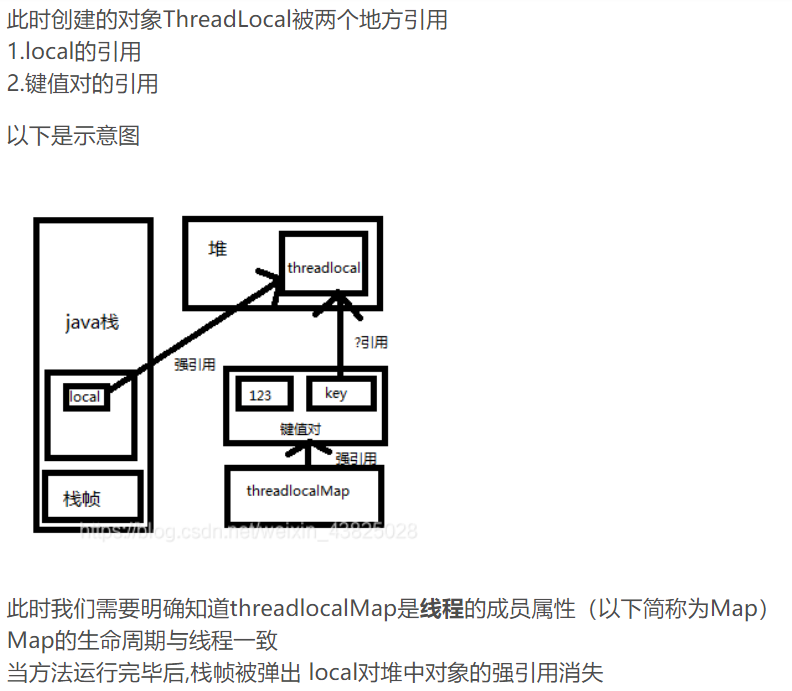
<https://www.cnblogs.com/vipstone/p/13354552.html>





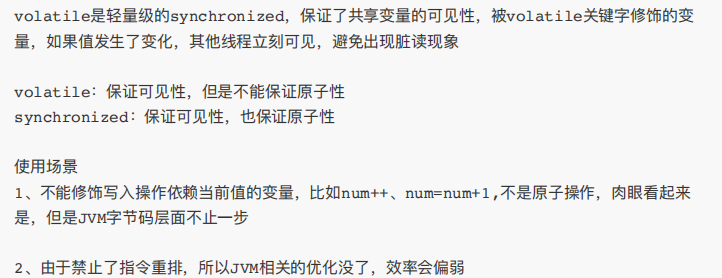


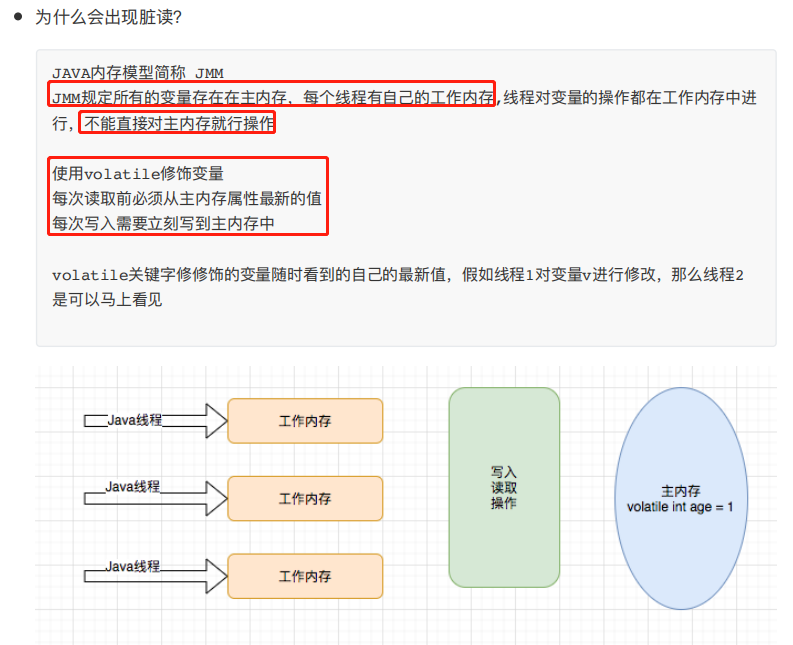
<https://blog.csdn.net/weixin_43825028/article/details/106424425>



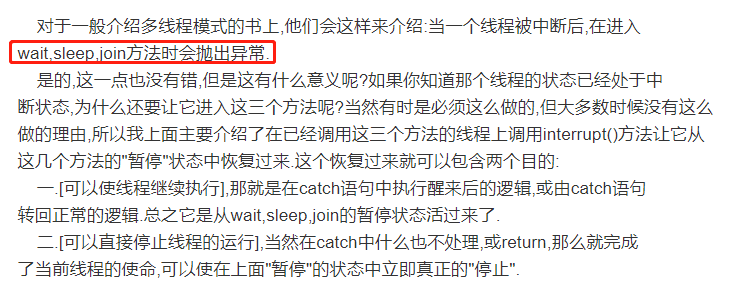
堆和方法区是共享的，栈和程序计数器不是共享的

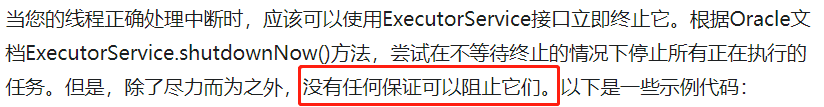


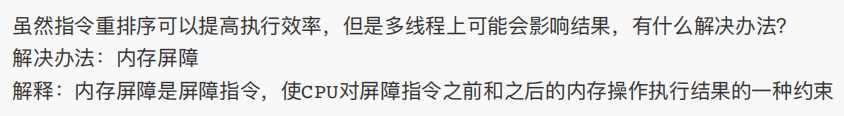




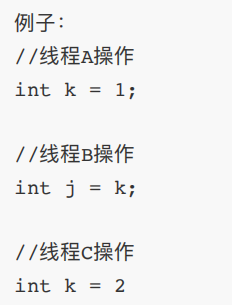
Semaphore信号量，主要用于限流



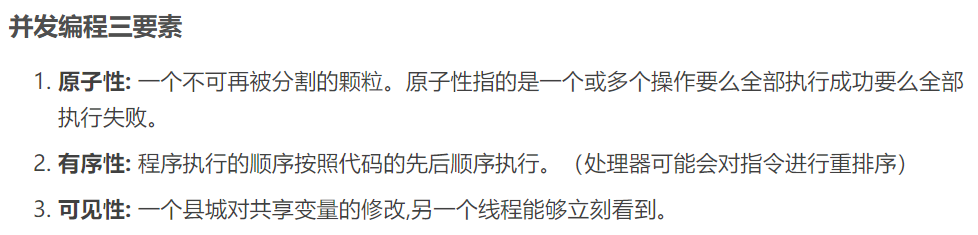


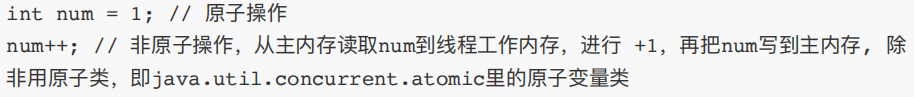


JAVA本身保证了先⾏发⽣原则，happens-before，使得指令重拍不会乱排序



线程C不会排到A,B之间



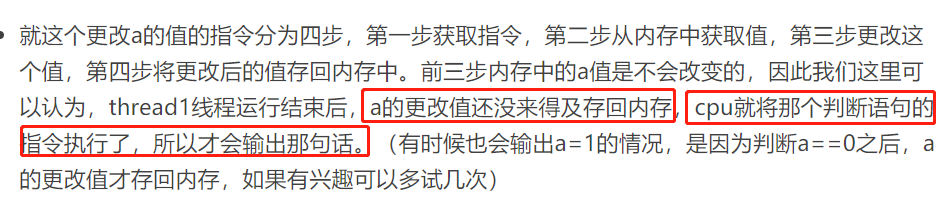
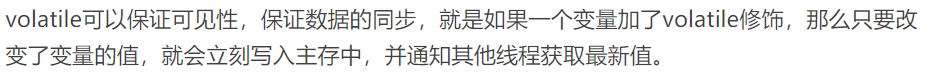


指令重拍导致flag先执行

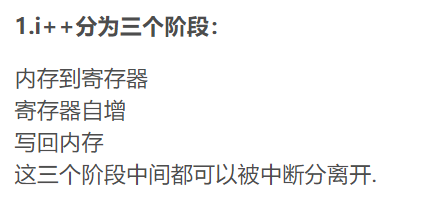
volatile用专业点更广泛的说法就是“对变量的写操作不依赖于当前值且该变量没有包含在其他具体变量的不变式中”。

<https://blog.csdn.net/weixin_34162695/article/details/91964301>

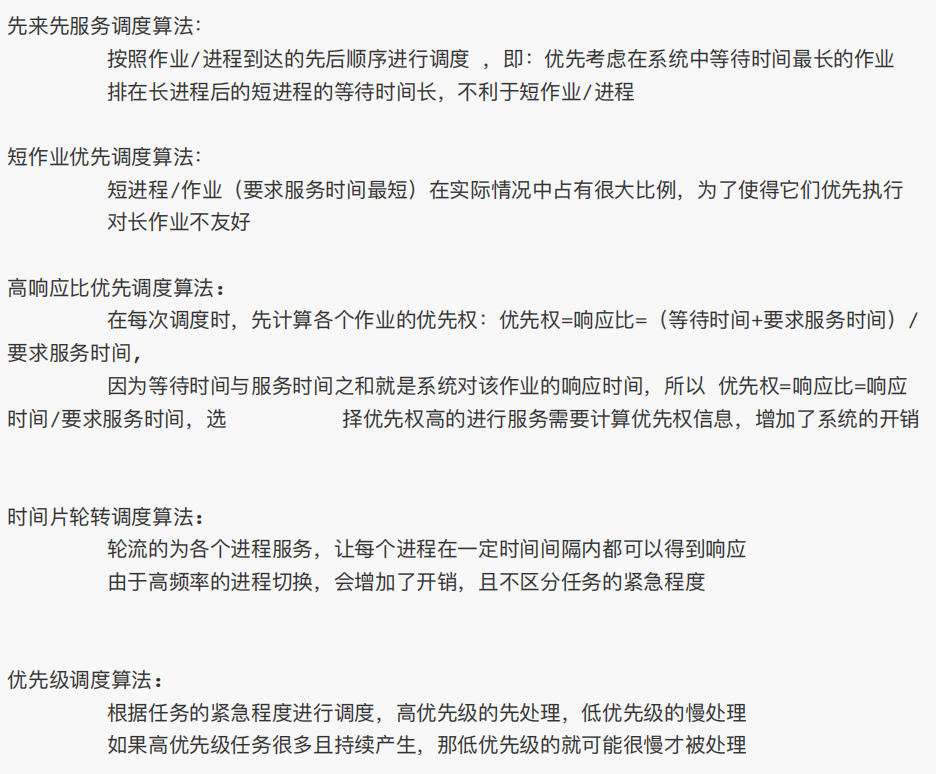
<https://blog.csdn.net/weixin_42809083/article/details/105394667>



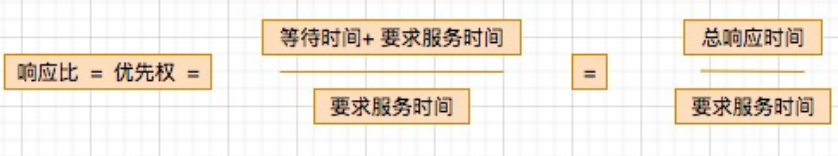
**在单线程环境下，指令执行的最终效果应当与其在顺序执行下的效果一致，否则这种优化便会失去意义。这句话有个专业术语叫做as-if-serial semantics (as-if-serial语义)**

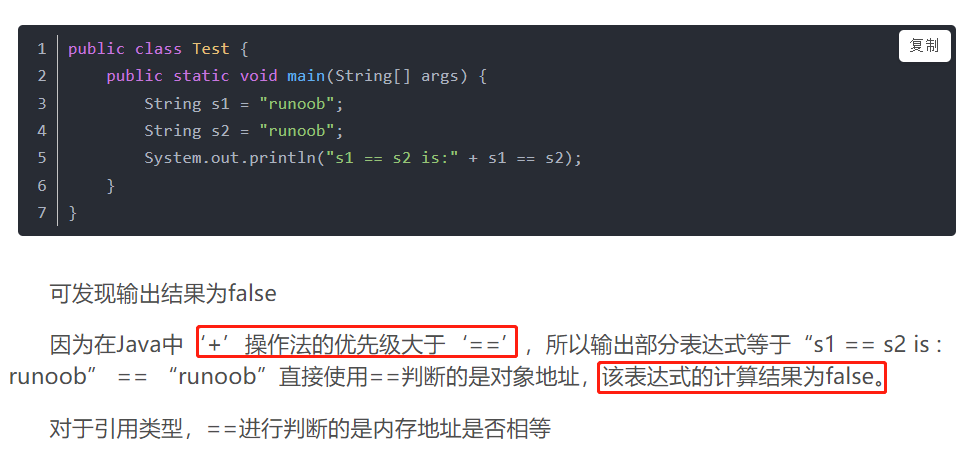


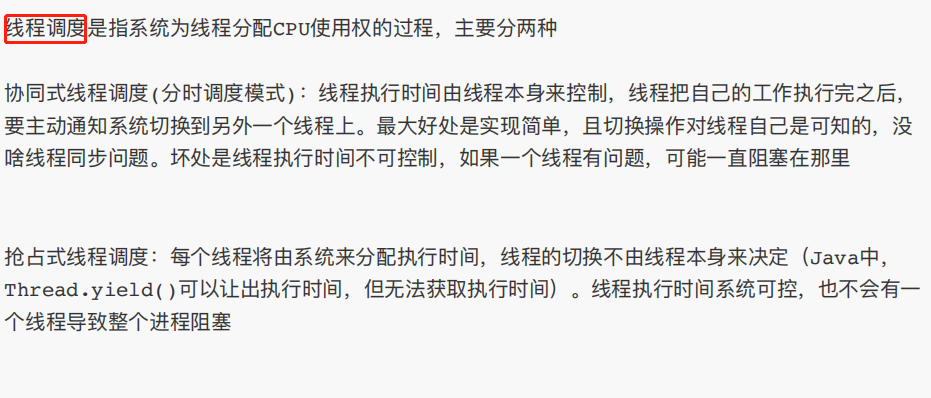
所以两个线程可能同时拿到寄存器，加一后返回，导致两次都是0到1的过程而不是0到1再到2





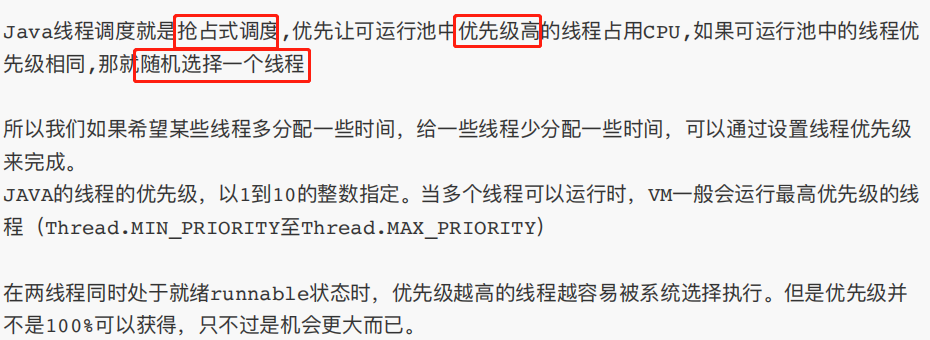


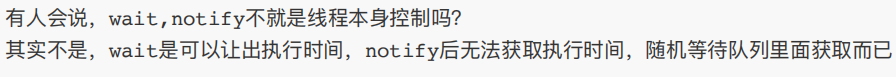




协同式是线程一个一个来，一个个处理

抢占式线程调度不是来个线程处理一个，而是cup决定先执行哪一个

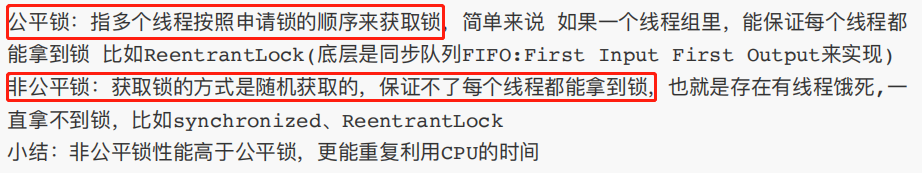




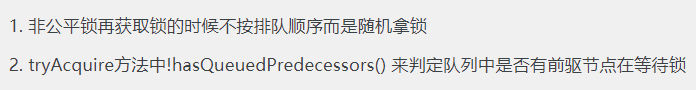
Notify不是唤醒就立刻执行了，而是进入就绪态，随机等到cpu执行



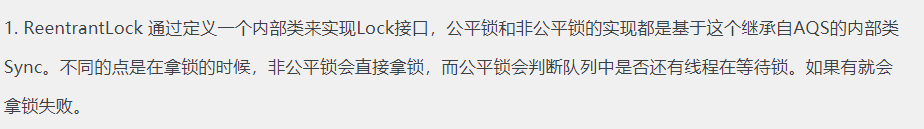
乐观锁一般都是通过版本号来同步，比如cas

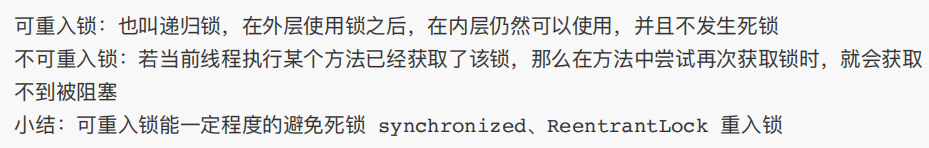






A释放锁，B进来刚好拿到锁，公平锁B还要判断是否有先前等待队列，非公平锁直接就拿到了





不可重复锁直接触发等待状态了



可重入是很重要的，线程获取了该锁，后面都可以继续使用

