Java多线程---ReentrantLock

ReentrantLock介绍

重入锁。同一个线程可以锁住多次，但是在使用完毕后，必须释放多次锁才能将锁完全释放，否则，还是该线程还是会处于阻塞状态。并且重入锁有两种，一种是公平锁，一种是非公平锁，公平锁会把锁按照顺序轮流交付线程执行，而非公平锁不会，非公平锁会乱序选择一个线程执行。

ReentrantLock使用

ReentrantLock.lock：加锁
ReentrantLock.tryLock:可输入时间，如果在指定时间内，获取到锁的话，那么就会返回true，否则返回false，如果没有指定时间的话，就立即去尝试获取锁
ReentrantLock.lockInterruptibly：可以被中断的锁，加锁的线程被调用interrupt方法后，可以被中断释放锁，而不会抛出异常
ReentrantLock.isHeldByCurrentThread：判断当前的Lock是否被当前线程Hold住
ReentrantLock.getHoldCount：判断当前的锁被多少个线程引用住
ReentrantLock.hasQueuedThread：判断传入的Thread是否已经存在于锁等待队列中
ReentrantLock.hasQueuedThreads：判断当前等待队列中是否有线程正在等待锁

ReentrantLock中的Node中waitStatus

Node中的waitStatus中会有五种变量:

1. CANCELLED：值为1。表示当前节点被取消了，由于超时或者中断的原因导致的取消，那么这个节点就永远会处于CANCELLED状态，不会在变成其他的状态了。典型的，一个被Cancelled节点永远不会被阻塞。
2. 0：除了四种状态之外的所有状态
3. SIGNAL：值为-1。表示当前节点的后继节点已经被阻塞或者即将被阻塞

了（通过LockSupport.park），所以当前节点必须unpark它的后继节点，当这个节点被释放或者取消的时候。为了避免竞争，acquire方法必须首先表明他们需要一个Singal，然后重试原子的获取操作，然后如果失败了，那么就阻塞。

4. `CONDITION`：值为-2。表示当前节点正处在一个Condition队列中，处在这种状态的节点不会被用来作为一个同步队列的节点，直到被transferred。而transferred带来的作用就是将状态值设置成了0的初始化状态。
5. `PROPAGATE`：值为-3。表示一个可传播的节点，它只有在`doReleaseShared`的时候会被设置，确保一个可传播的节点，解释其他的操作也在阻扰它。