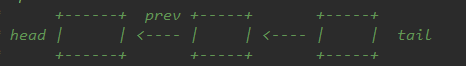
AQS学习

AQS，AbstractQueuedSynchronizer，即队列同步器。它是构建锁或者其他同步组件的基础框架（如ReentrantLock、ReentrantReadWriteLock、Semaphore等）

Node 节点

自旋锁 CLH FIFO



包括: shared mode 共享模式 exclusive mode 独占模式

状态: waitStatus

CANCELLED = 1 取消

SIGNAL = -1 当前节点-1的时候需要通知下一个节点停止阻塞

CONDITION = -2 等待队列,调用signal()后才会进入同步队列,并设置为0

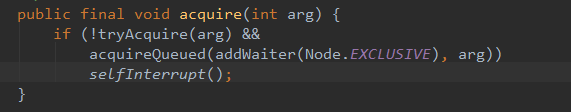
PROPAGATE = -3 传播,用于共享队列,下一次共享同步将无条件传播

0 : 新加入的节点

CAS : compare and swap

方法解析:

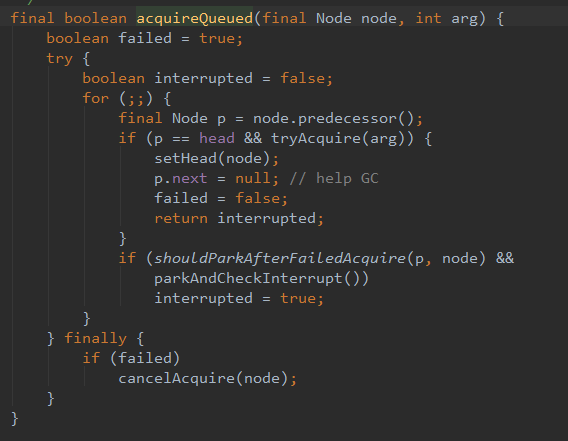
acquire() 独占模式同步状态的获取



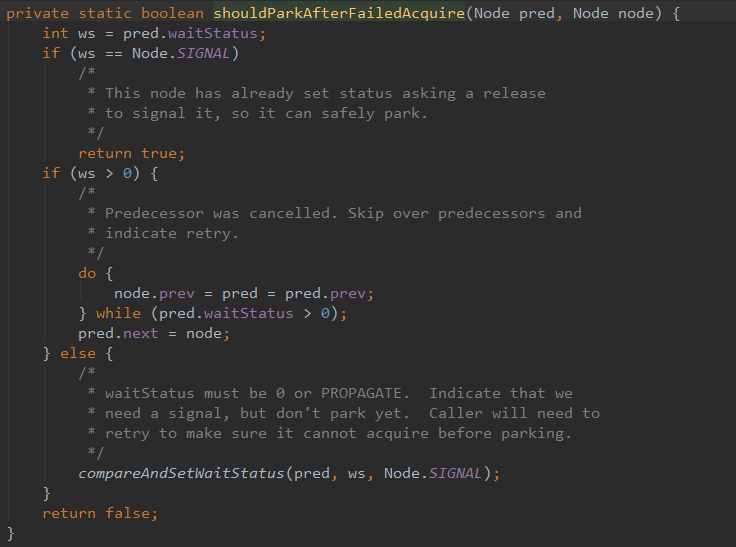
首先调用tryAcquire() 尝试获取同步状态(相当于锁) 如果失败则调用

addWaiter(Node.EXCLUSIVE) 将该线程放入队列的尾部

acquireQueued(final Node node, int arg) 对已经在队列中的节点不间断自旋获取

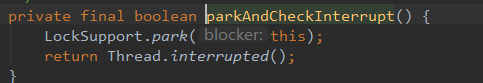


首先查看当前节点的前一个节点是否为head节点,如果是,同时获取同步成功,返回,并将当前节点设置为head节点.获取失败后调用shouldParkAfterFailedAcquire(Node pred, Node node)判断是否应该挂起.



如果当前节点的前节点为SIGNAL状态则应该挂起并等待唤醒.如果前节点是取消节点则删除.其它若前节点状态为为0或者传播状态则将其设置为SIGNAL,以便在下次判断是将当前节点挂起.

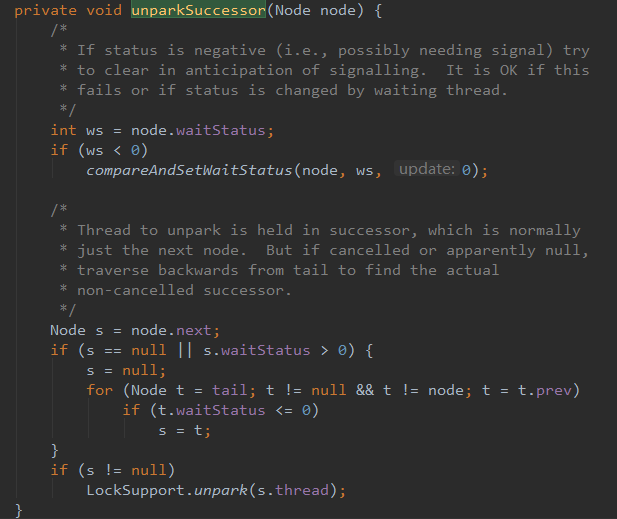
最后返回interrupted = true 将当前线程阻塞.



release(int arg) 独占模式同步状态的释放

首先调用子类的tryRelease()方法释放锁.

unparkSuccessor(Node node) 然后唤醒后继节点



首先将当前释放的节点状态设置为0,清除信号量,然后循环从tail往前找到后继节点,然后调用LockSupport.unpark(s.thread) 唤醒.

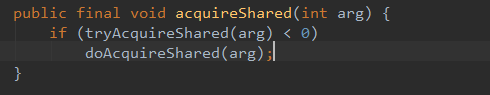
doAcquireInterruptibly(int arg) 独占模式的中断获取方法,如果线程被中断则抛出异常.

tryAcquireNanos(int arg,long nanos)。该方法为acquireInterruptibly方法的进一步增强，它除了响应中断外，还有超时控制。即如果当前线程没有在指定时间内获取同步状态，则会返回false，否则返回true.

shared mode 共享模式

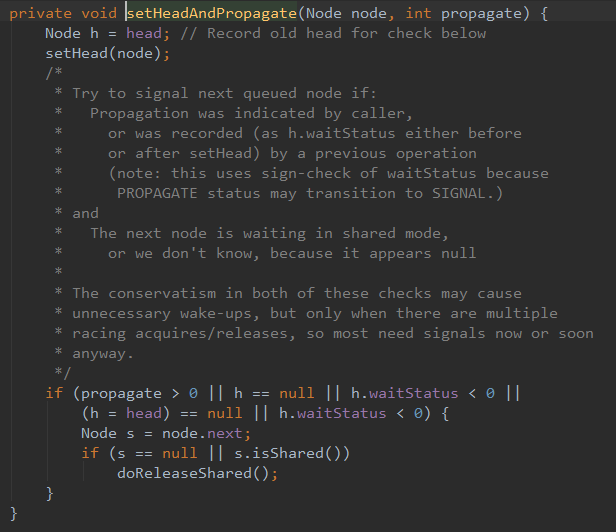
acquireShared 获取共享同步状态

tryAcquiredShared 尝试获取同步状态,返回负数表示获取失败,0表示当前获取节点获取成功但后继节点获取失败,正数表示都可以成功.

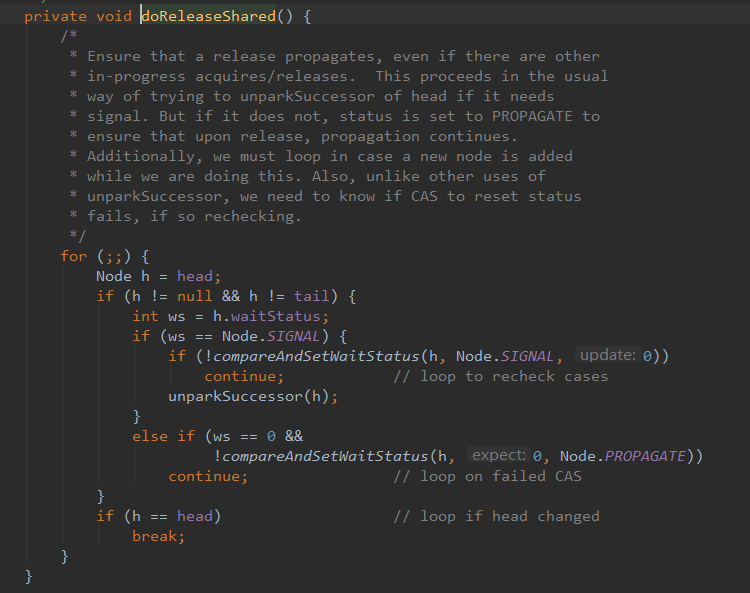


doAcquiredShared 通过自旋获取同步状态,大部分和独占锁相同,区别在于方法

setHeadAndPropagate(node, r)



获取同步状态成功后,共享模式首先会重新设置头节点(同独占模式) 接着会传播给后继节点. 方法 doReleaseShared()



在这个方法里,首先自旋判断当前头节点状态是否为-1 如果是,则循环设置为0,成功后调用

unparkSuccessor(h) 上面有分析过的方法,唤醒后继节点.

状态变为0后再设置为-3 变为PROPAGATE状态. 不从-1 直接转为 -3 是因为在 unparkSuccessor 方法中会再次设置为0.