## AbstractQueuedSynchronizer

## ReentrantLock

跟synchronized相同，是可重入的重量级锁。但是其用法则相当不同，首先ReentrantLock要显式的调用lock方法表示接下来的这段代码已经被当前线程锁住，其他线程需要执行时需要拿到这个锁才能执行，而当前线程在执行完之后要显式的释放锁。

### 2.1、ReentrantLock()

方法作用：无参构造方法，默认新建非公平同步器

①调用内部类NonfairSync无参构造赋值给成员变量sync；

②依次调用父类的默认构造函数，各级父类无参构造函数没有做任何事情；

### 2.2、ReentrantLock(boolean fair)

方法作用：带布尔参数构造方法，fair为true，创建公平同步器，否则创建非公平同步器

### 2.3、void lock()

方法作用：获得锁

假如创建的是非公平同步器：

根据假设，本方法实际就是调用了内部类NonfairSync（非公平同步器

）的lock()方法；

调用本方法的的当前线程就是要抢占本锁的锁资源；

首先要将NonfairSync的成员变量state设置为1，条件是此时state的值必须是0（没有其他线程占用锁资源），这个过程在本地方法中完成，并且是同步的；

如果成功将state设置为1，则说明当前线程抢占到锁资源，则将NonfairSync的成员变量exclusiveOwnerThread赋值为当前线程，exclusiveOwnerThread的作用是保存占用锁资源的线程，此时整个方法执行完毕；

如果没有抢占到锁资源（设置state失败），则获取当前state的值，如果state=0（此时没有线程占用锁资源），则再次尝试将state的值设置为1，如果成功，则将NonfairSync的成员变量exclusiveOwnerThread赋值为当前线程，此时整个方法执行完毕；如果state=1（此时有线程占用锁资源），则判断是否是当前线程自己占用了锁资源，如果是，则将state的值加1（重入锁，多次获取锁资源），此时整个方法执行完毕；

如果再次设置state的值失败，则将当前线程保存进新建的节点中，并放到节点双向链表的队尾（若此时链表中没有节点，队尾即是队首），此过程也是写在本地方法中的，是同步的，如果失败则一直尝试，直到成功为止；

接下来，判断新建的当前线程节点的前节点是否为首部节点，并且

假如创建的是非公平同步器：

①调用成员变量sync的lock()方法（及调用NonfairSync的lock()方法）；

②调用AbstractQueuedSynchronizer（AQS）的compareAndSetState(int expect, int update)方法，expect默认为0，update默认为1；

③调用成员变量unsafe的compareAndSwapInt(this, stateOffset, expect, update)，this代表本非公平同步器sync，stateOffset默认值为0，expect默认为0，update默认为1；此方法希作用：当成员变量state的值为0时，则将state设置为1并返回true，否则state值保持不变并返回false；

④如果第③步返回true，则调用Thread.currentThread()获取当前线程，然后调用AbstractOwnableSynchronizer的setExclusiveOwnerThread(Thread thread)方法将当前线程赋值给成员变量exclusiveOwnerThread（此线程就是独占所有者线程）；到此本方法全部执行完毕；

⑤如果第③步返回false，则调用AQS的acquire(int arg)方法，参数arg默认为1；

⑥在acquire(int arg)方法中先调用NonfairSync的tryAcquire(arg)，tryAcquire(arg)中再调用Sync的nonfairTryAcquire(arg)方法；Thread.currentThread()获取当前线程，获取state的值；

如果state=0（没有线程占用锁资源），则调用AQS的compareAndSetState(int expect, int update)方法（抢占锁资源），expect默认为0，update=arg（默认为1）；

如果compareAndSetState返回true，说明抢占成功，将当前线程赋值给成员变量exclusiveOwnerThread（此线程就是独占所有者线程），并结束nonfairTryAcquire(arg)方法，返回true回到acquire(int arg)方法中，acquire(int arg)方法也直接结束，整个lock()方法结束（当前线程抢占锁资源成功）；

如果compareAndSetState返回false，说明抢占失败，结束nonfairTryAcquire(arg)方法，返回false回到acquire(int arg)方法中；进入addWaiter(Node mode)方法，参数是静态常量EXCLUSIVE（此常量为null），新建节点node，传入当前线程作为节点维护的线程，并传入EXCLUSIVE作为此节点的nextWaiter（下一个等待的节点）；判断队尾的节点tail是否为空；如果不为空，则将新节点node的前节点设置为队尾节点，然后调用compareAndSetTail(tail, node)将新节点node设置为队尾节点，如果返回true（node成功替换tail成为队尾节点），则将tail的下一个节点设置为node，如果返回false（tail已经被其他线程节点替换了），则调用enq(final Node node)方法，将新建节点传入，