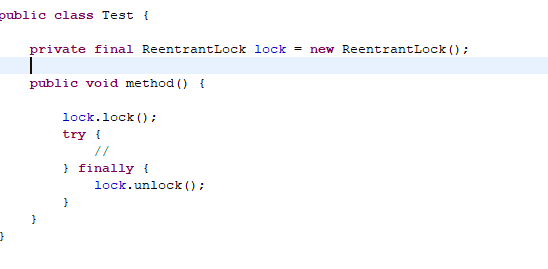
# ReentrantLock(jdk1.7.0\_67)

ReentrantLock是一种可重入互斥锁的实现，具有与Synchronized修饰的方法和代码块相同的行为和语义，而且增加了一些新的行为，例如：超时获取锁等。

最后成功调用lock但还没有调用unLock的线程将获得锁。当锁没有被其他线程获得时，线程调用lock将立即返回并获得锁，如果线程已经获得了锁，lock方法也会立即返回，否则，lock方法将阻塞。可以使用isHeldByCurrentThread和getHoldCount方法检查线程是否已经获得了锁。

ReentrantLock提供了公平锁与非公平锁两种实现机制，可以通过构造器传参指定使用公平锁还是非公平锁，默认使用的是非公平锁，传入true表示使用公平锁。公平锁的获得偏向于等待时间最长的线程，而非公平锁的获得不能保证特定的顺序。需要注意的是，tryLock方法对公平锁不敏感，下面可以看到tryLock的实现。

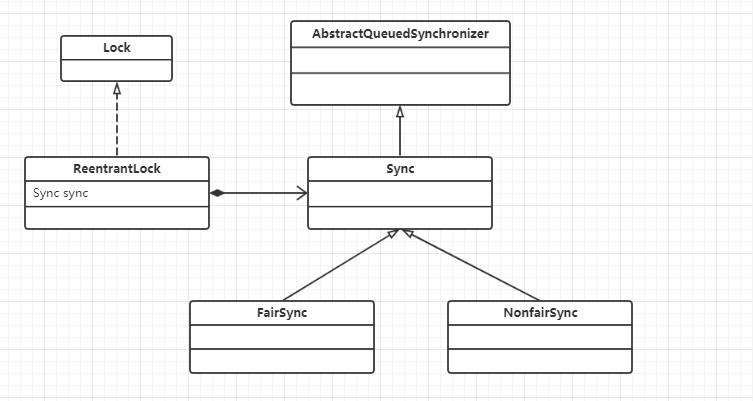
下面给出了了一个ReentrantLock典型的使用方式。



# 源码分析

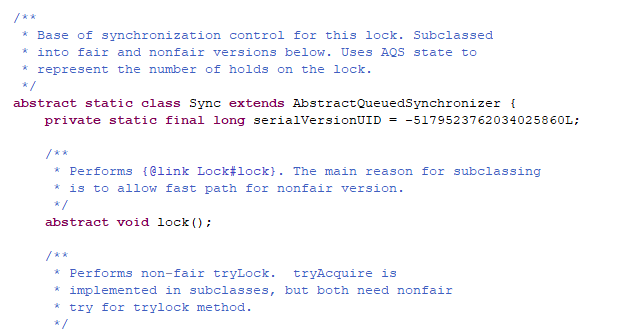
## 1.1类图

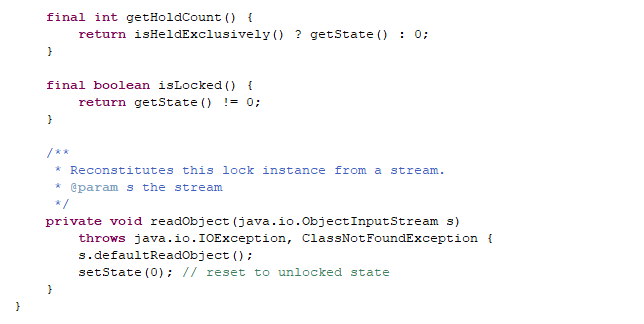
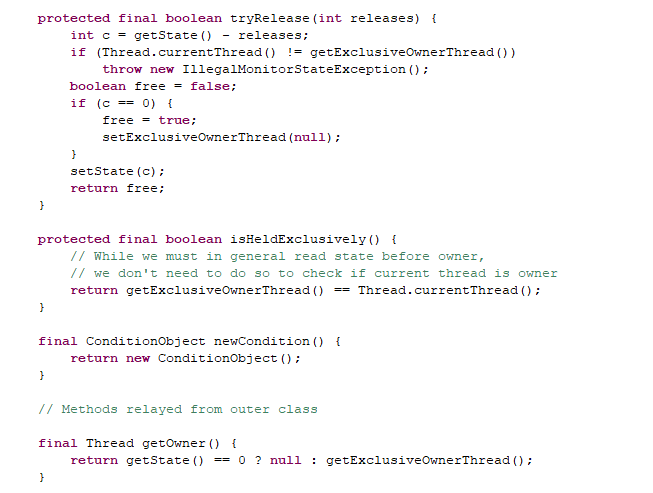
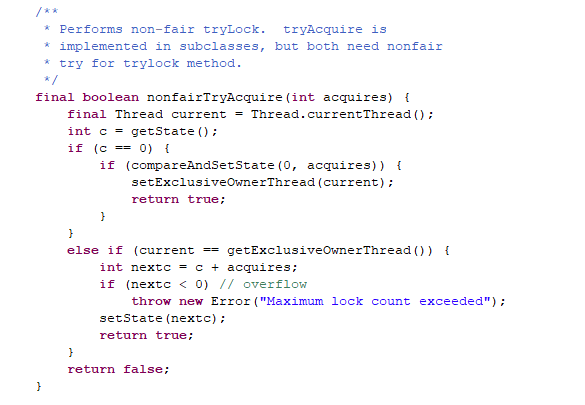
先看一下ReentrantLock的类图。



ReentrantLock实现了Lock接口，ReentrantLock中方法的实现委托给了Sync，Sync类继承了AbstractQueuedSynchronizer类，在讲AbstractQueuedSynchronizer源码时提到，AbstractQueuedSynchronizer中的大多数模板方法是供Lock接口方法调用的。ReentrantLock中定义了两个Sync的实现，FairSync实现了公平锁的行为，NonFairSync实现了非公平锁的行为。

## 1.2Sync





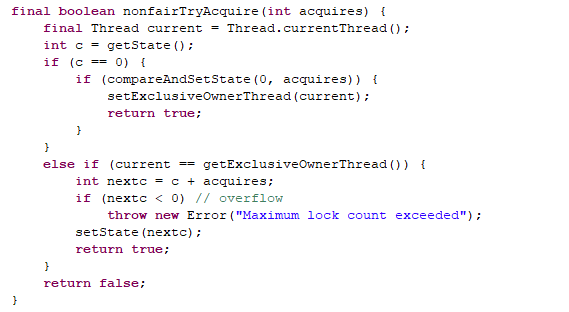
### 1.2.1lock

lock为一个抽象方法，ReentrantLock的lock方法委托给该方法。

### 1.2.2nonfairAcquire

看一下nonfairTryAcquire方法。根据方法名就可以猜到这个方法是提供给非公平锁使用的，前文提到，NonfairSync实现了非公平锁的实现，那么不应该将nonfairTryAcquire放到Sync的子类NonfairSync中更合适吗？其实无论在公平锁还是在非公平锁中，tryLock方法都是调用了nonfairAcquire的，所以才将nonfairTryAcquire提到了Sync中。

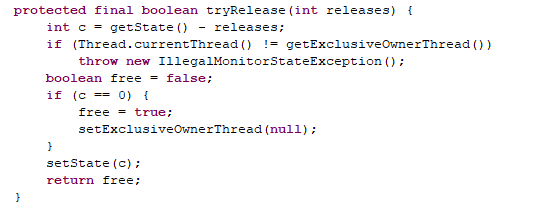
接下来分析一下nonfairAcquire的实现。



1. 获取同步状态并赋值给c。
2. 如果c==0，表示还没有线程获得锁，CAS更新同步状态，更新成功后设置所属线程为当前线程并返回true，更新失败返回false。
3. 如果c!=0，表示已经有线程获得了锁，如果当前线程获得了锁，将同步状态加acquires并返回true，否则返回false。

### 1.2.3tryRelease

release方法会调用这个方法释放锁。

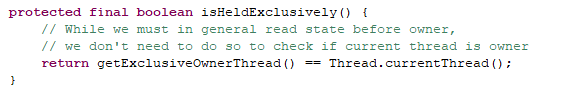


1. 获取同步状态-releases并赋值给c。
2. 判断当前线程是否为锁的所属线程。如果不是抛出IllegalMonitorStateException异常。
3. free=ture表示完全释放，因为可重入锁允许线程多次获得锁，只有在线程

完全释放之后将free设置为true，并所属线程设置为null。

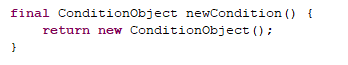
1. 更新同步状态，返回free，只有当锁完全释放后才会返回true。

### 1.2.4isHeldExclusively

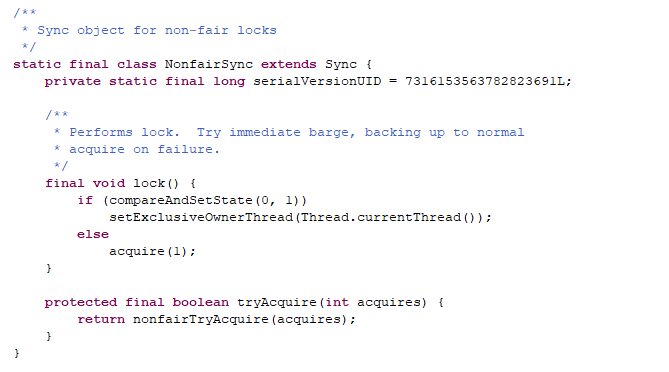


判断当前线程是否持有锁。

### 1.2.5newCondition



## 1.3NonfairSync



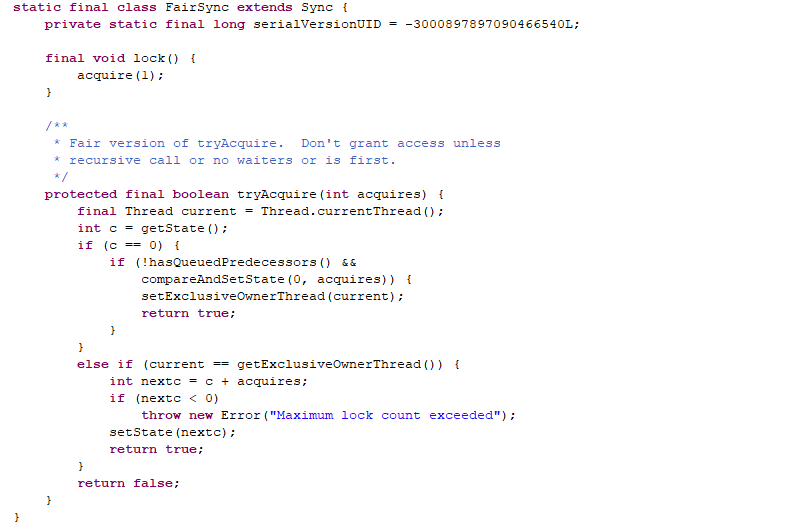
### 1.3.1lock

lock方法首先尝试设置同步状态，如果成功设置锁所属线程，失败调用阻塞方法acquire直到获得锁，acquire调用了tryAcquire方法。

### 1.3.2tryAcquire

调用了nonfairTryAcquire方法，这里不再赘述。

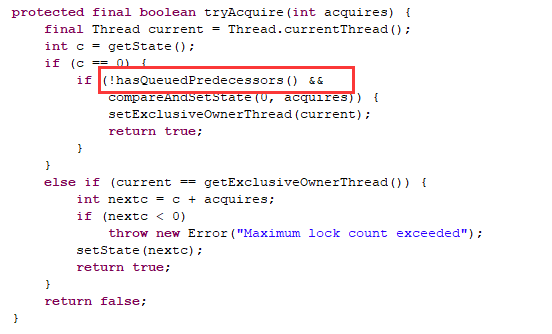
## 1.4FairSync



### 1.4.1lock

直接调用了阻塞方法acquire。

### 1.4.2tryAcquire



红圈部分是与非公平锁实现的主要区别。公平锁在判断c==0之后，没有直接设置同步状态而是先判断了同步队列中是否存在前驱节点，如果存在直接返回false，不存在才会设置同步状态。

# ReentrantLock

## 2.1构造器



默认构造器使用了非公平锁。



指定使用公平锁与非公平锁的构造器。

## 2.2tryLock



可以看到，tryLock调用了sync的nonfairTryAcquire方法，对公平锁的设置不敏感。

## 2.3其他方法

ReentrantLock中的大多数方法都是委托给Sync来实现的，这里不再一一说明。

# 公平锁与非公平锁的优缺点

公平锁会降低系统的吞吐量，但可以减少线程饥饿现象的发生。

非公平锁系统的吞吐量较高，但可能会出现饥饿现象。