# ReentrantLock分析

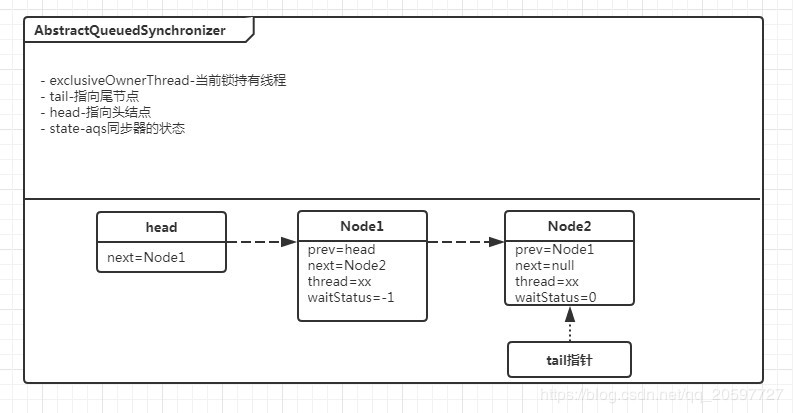
## ReentrantLock实现核心-AQS

### AQS概念

AQS（AbstractQueuedSynchronizer），队列同步器，在juc包中的工具类依赖AQS来实现同步控制。

AQS可以被当做是一个同步监视器的实现，并且具有排队功能。当线程尝试获取AQS的锁时，如果AQS已经被别的线程获取锁，那么将会新建一个Node节点，并且加入到AQS的等待队列中，这个队列也由AQS本身自己维护。当锁被释放时，唤醒下一个节点尝试获取锁。

AQS结构图：



说明：

exclusiveOwnerThread：在互斥模式下，表示当前持有锁的线程；

head：指向队列的头节点。Head为空时表示没有等待的线程；

tail：指向队列的尾节点；

state：表示AQS同步器的状态。

state在不同类中含义可能不太一样，如：

在ReentrantLock中表示AQS的锁是否已被占用，0表示未占用，>=1表示已占用，当大于1时表示被同一线程多次重入锁。

在CountDownLatch中表示计数器剩余次数，当达到0时唤醒等待线程。

在Semaphore中表示AQS还可以被获取锁的次数，获取一次就减1，当到达0时，尝试获取的线程将会阻塞。

### Node结构

Node节点是AQS管理的等待队列的节点元素，除head节点外，其它每个节点代表一个正在等待线程的队列。

Node参数说明：

prev：前置节点

next：后置节点

thread：代表的线程

waitStatus：节点的等待状态。

waitStatus有三种值：

1表示节点已经取消，即线程可能已经中断，不需要再等待获取锁，在后续代码会中处理跳过waitStatus为1的节点；

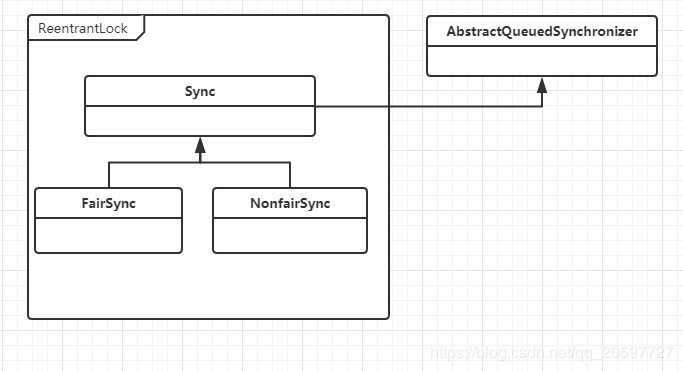
-1表示当前节点的后置节点代表的线程需要被挂起；

-2表示当前线程正在等待的是Condition锁。

## ReentrantLock与AQS关系

ReentrantLock实现核心是基于AQS ，ReentrantLock里面有FairSync和NonfairSync（默认）这两个内部类，它们通过继承AbstractQueuedSynchronizer来实现两种同步互斥方案：公平锁和非公平锁。在ReentrantLock中lock和unlock操作，都由FairSync和NonfairSync实际完成。

它们关系如下：



ReentrantLock使用：

ReentrantLock lock = new ReentrantLock();

lock.lock(); // 加锁

... // 获得锁后执行逻辑

lock.unlock(); // 解锁

ReentrantLock构造器如下：

public ReentrantLock() {

sync = new NonfairSync();

}

public ReentrantLock(boolean fair) {

sync = fair ? new FairSync() : new NonfairSync();

}