reentrantLock知识点

一、一些基本概念

1.ReentrantLock是一个可重入的互斥锁。

互斥：ReentrantLock在同一个时间只能被一个线程持有；重入：ReentrantLock可以被单个线程多次获取

ReentrantLock分为“公平锁”和“非公平锁”，ReentrantLock通过一个FIFO的等待队列来管理获得该锁的所有线程。在公平锁的机制下，线程依次排队获得锁；在非公平锁的机制下，如果锁是可获取的，就会从队列中取出，不管是不是在队头。

Condition是需要和Lock联合使用的：通过Condition中的await()方法，能让线程阻塞[类似于wait()]；通过Condition的signal()方法，能让唤醒线程[类似于notify()]

2. AQS，即AbstractQueuedSynchronizer类，他是java中管理“锁”的抽象类，AQS是独占锁和共享锁的公共父类

3.AQS的锁类别：

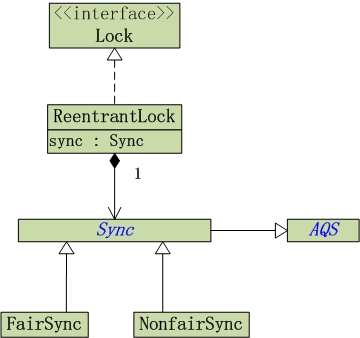
独占锁：锁在一个时间点只能被一个线程占有，根据锁的获得机制，划分为“公平锁”和“非公平锁”。公平锁，获取锁线程是按照FIFO队列依次获取锁；非公平锁，当有线程要获取锁时，它就会直接获取锁。独占锁的典型就是ReentrantLock，ReentrantReadWriteLock.WriteLock也是独占锁。

共享锁：能被多个线程同时拥有，被共享的锁。ReentrantReadWriteLock.ReadLock，CyclicBarrier， CountDownLatch和Semaphore都是共享锁。

4.CLH队列

CLH是AQS"等待锁"的线程队列，存放的是等待获取该锁的线程。他是一种基于链表非阻塞的FIFO队列，并发条件下不会发生阻塞，是通过自旋锁和cas保证节点插入和移除的原子性

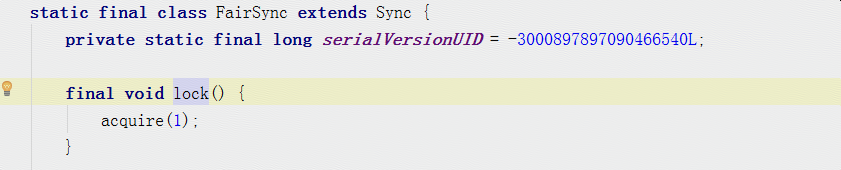
二、ReentrantLock的数据结构



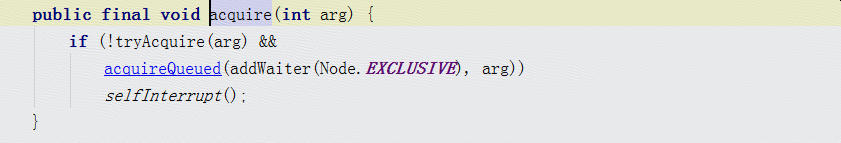
ReentrantLock默认实现的是非公平锁

三、公平锁的获取过程

New ReentrantLock(true)，指定的锁就是公平锁。获取锁时，会调用ReentrantLock的lock()方法，lock()方法调用的是内部抽象类Sync的方法lock()，会调用Sync的子类FairSync中的lock()方法



调用的是AQS类中的acquire()方法：



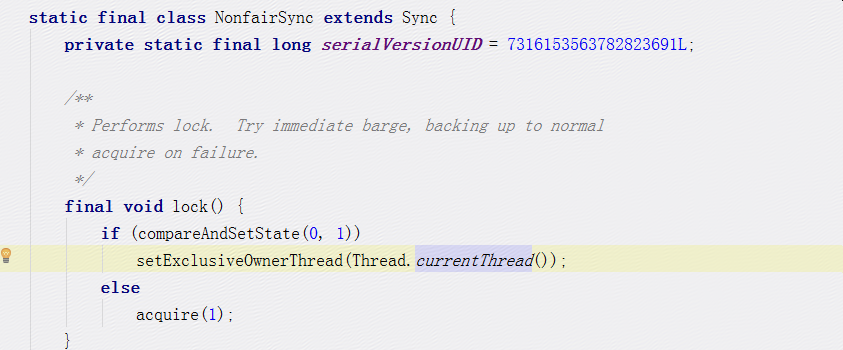
acquire()方法的最终目的就是获得公平锁：具体实现是在子类的tryAcquire()方法中。

(1)先是通过tryAcquire()尝试获得锁，如果成功直接返回，如果失败，再通过acquireQueued()获取锁

(2)尝试失败的情况下，会先通过addWaiter()来将“当前线程”加入到“CLH队列”末尾；然后调用acquireQueued()，在CLH队列中排队等待获取锁，再此过程中线程处于休眠状态，直到获取锁后才返回。如果在休眠等待过程中被中断过，则调用selfInterrupt()来自己产生中断。

四、非公平锁的获取过程

New ReentrantLock()默认就是使用的非公平锁，获取锁是，会调用NonfairSync类的lock()方法：



公平锁与非公平锁的lock的区别在于

公平锁：会直接调用acquire()方法

非公平锁：先通过CAS操作将status从0变成1，如果成功就直接拿到了锁，如果失败就会放入CLH队尾进行排队

非公平锁的tryAcquire()方法会尝试获取锁

(1)如果“锁”是空闲的，当前线程直接拿到锁

(2)如果“锁”的持有者是当前线程，更改锁状态即可

(3)上述两条都不成立，认为尝试失败，就会放入队列

五、公平锁与非公平锁的区别：

公平锁在尝试获取锁时，即使锁没有被任何线程持有，也会判断当前线程是都在CLH队列的表头，如果是才获取锁。

非公平锁在尝试获取锁时，如果锁没有被任何线程持有，直接得到锁，如果被其他线程持有，那么会放入CLH队列的队尾进行排队

六、补充知识点

当前线程每执行一次lock，都会调用一次AQS中的acquire()方法，锁状态status都会+1，每次unlock,都会执行AQS的release()方法，锁状态status就会-1，当status为0的时候，说明当前线程释放了锁。

非公平锁比公平锁效率高的原因：

一个实际例子：

1).线程1,2,3竞争锁，线程1成功获取，2,3进入队列进行排队

2).线程1执行完毕释放锁，status变为0，唤醒第一个排队的线程2

3).此时线程4尝试获取锁，由于线程2被唤醒，所以线程2,4开始竞争锁

4).线程4成功获取到锁，status由0变1，线程2获取失败，继续排队

上述例子就是非公平锁获取锁的过程，那么非公平锁比公平锁效率高的原因就是：

1).线程1先将status设为0，然后去唤醒线程2，这个过程是由时间间隔的

2).如果线程1将status设置为0的时候，线程4通过CAS成功获取到锁，而在线程2尝试获取锁之前，线程4执行完毕，并没有耽误线程2的获取，这就增加了系统的吞吐量。

由此可见，非公平锁适合加锁时间非常短的任务。

参考博客：

<http://www.cnblogs.com/xrq730/p/7056614.html>

<http://www.cnblogs.com/xrq730/p/7067904.html>

<http://www.cnblogs.com/xrq730/p/7096084.html>

公平锁与非公平锁定义在Reentrantlock中，当线程获取不到锁，会放入clh阻塞队列中，公平锁就顺序获取，非公平插队获取。Lock是广义的内置锁

wait，notify是object的方法，必须在synchronized代码块中执行，wait后释放锁，当前线程放入wait\_set队列中，另一线程调用notify是会中队列中随机选择一个进行释放。

await,signal是condition中的方法，condition从lock获取。一个lock包含无数个condition，也就是可以new无数个。可以理解为广义的内置条件队列

AQS中队列保存的元素是Node实体结点，队列头结点是一个空的node。AQS包含两种模式的锁，独占模式和共享模式，独占模式下节点被唤醒后只需要将该节点设置成head就可以了。共享锁某一个节点被设置成head后，如果后继节点是SHARED状态，那么会将共享状态向后传播。

独占锁和共享锁具体实现都有公平锁和非公平锁。ReentrantLock是独占锁，Semaphore和countDownLatch是共享锁

lock.newCondition()实际获取的是AQS中的ConditionObject实体，condition.await()方法是没有cas原子操作的，因为他是在lock下执行的。await()的实现原理就是构建等待队列。await调用后，也是将线程以node的形式保存到了队列中。队列头元素就保存了线程信息。调用condition.signal()后，await()后面的代码不会立即执行，因为释放的node放入了AQS的队列尾部，轮到它执行的时候，才会执行。

独占锁和共享锁的区别：

1.独占锁同时只有一条线程可以acquire获取所成功，共享锁同时可以有多条线程acquire成功，比如semaphore

2.独占锁每次只能唤醒一个Node，共享锁每次唤醒的时候可以将状态向后传播，即可以唤醒多个Node，比如countDownLatch