

第七章 ReentrantLock总结

常用方式：



```
int a = 12;
//注意：通常情况下，这个会设置成一个类变量，比如说Segment中的段锁与copyOnWriteArrayList中的全局锁
final ReentrantLock lock = new ReentrantLock();

lock.lock();//获取锁
try {
    a++; //业务逻辑
} catch (Exception e) {
} finally{
    lock.unlock();//释放锁
}
```



1、非公平锁获取锁的步骤lock()

基于CAS尝试将state（锁数量）从0设置为1

- A、如果设置成功，设置当前线程为独占锁的线程；
- B、如果设置失败，还会再获取一次锁数量，

B1、如果锁数量为0，再基于CAS尝试将state（锁数量）从0设置为1一次，如果设置成功，设置当前线程为独占锁的线程；

B2、如果锁数量不为0或者上边的尝试又失败了，查看当前线程是不是已经是独占锁的线程了，如果是，则将当前的锁数量+1；

如果不是，则将该线程封装在一个Node内，并加入到等待队列中去。等待被其前一个线程节点唤醒。

2、公平锁获取锁的步骤lock()

获取一次锁数量，


B1、如果锁数量为0，如果当前线程是等待队列中的头节点，基于CAS尝试将state（锁数量）从0设置为1一次，如果设置成功，设置当前线程为独占锁的线程；

B2、如果锁数量不为0或者当前线程不是等待队列中的头节点或者上边的尝试又失败了，查看当前线程是不是已经是独占锁的线程了，如果是，则将当前的锁数量+1；如果不是，则将该线程封装在一个Node内，并加入到等待队列中去。等待被其前一个线程节点唤醒。

3、解锁的步骤


- 1）获取当前的锁数量，然后用这个锁数量减去解锁的数量（这里为1），最后得出结果c
- 2）判断当前线程是不是独占锁的线程，如果不是，抛出异常
- 3）如果c==0，说明锁被成功释放，将当前的独占线程置为null，锁数量置为0，返回true
- 4）如果c!=0，说明释放锁失败，锁数量置为c，返回false
- 5）如果锁被释放成功的话，唤醒距离头节点最近的一个非取消的节点

4、ReentrantLock的四种获取锁方法对比



```
/**
 *获取一个锁
 *三种情况：
 *1、如果当下这个锁没有被任何线程（包括当前线程）持有，则立即获取锁，锁数量==1，之后再执行相应的业务逻辑
 *2、如果当前线程正在持有这个锁，那么锁数量+1，之后再执行相应的业务逻辑
 *3、如果当下锁被另一个线程所持有，则当前线程处于休眠状态，直到获得锁之后，当前线程被唤醒，锁数量==1，再执行相应的业务逻辑
 * 简言之，获取锁，如果锁无法获取，当前线程被阻塞，直到锁可以获取并获取成功为止。
 */
public void lock() {
    sync.lock();//调用NonfairSync（非公平锁）或FairSync（公平锁）的lock()方法
}

/**
 * lockInterruptibly():
 * 1、 在当前线程没有被中断的情况下获取锁。
 * 2、如果获取成功，方法结束。
 * 3、如果锁无法获取，当前线程被阻塞，直到下面情况发生：
 * 1) 当前线程(被唤醒后)成功获取锁
 */
```



```
* 2) 当前线程被其他线程中断
*/

public void lockInterruptibly() throws InterruptedException {
    sync.acquireInterruptibly(1);
}

/**
 * 四点注意：
 * 1、如果其他线程没有持有这个锁，立即返回true，设置锁的数量为1
 * 2、如果当前线程已经持有这个锁了，那么锁数量+1，立即返回true
 * 3、如果其他线程占有这个锁，该方法立即返回false
 * 4、要注意：这个锁是非公平锁（即无法用于公平锁策略中）
 */

public boolean tryLock() {
    return sync.nonfairTryAcquire(1);
}

/**
 * 在指定时间内这个锁某个时刻没有被其他线程占有并且当前线程没有被中断，获得这个锁
 *
 * 1) 如果没有其他线程占有这个锁，则获得这个锁，立即返回true，锁数量+1
 * 2) 如果当前线程已经持有这个锁了，那么锁数量+1，立即返回true
 * 3) 如果这个锁被其他线程持有，当前线程阻塞，直到下面三种情况发生：
 * A、当前线程（被唤醒后）成功的获取锁
 * B、其他线程中断了当前线程
 * C、指定时间超时（如果时间<=0，根本就不会等待）
 *
 * 如果该锁被用于公平锁：如果有其他线程在等待，即使判断出没有其他线程占有这个锁，当前线程也不会获取这个锁
 */

public boolean tryLock(long timeout, TimeUnit unit) throws InterruptedException {
    return sync.tryAcquireNanos(1, unit.toNanos(timeout));
}
```

