1.1可中断锁

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

本质上：最终会调用AQS的 **acquireInterruptibly(1);模板方法**

```
public final void acquireInterruptibly(int arg) throws InterruptedExcepti
on {
 //增加了对异常的状态的判断
 //如果检测线程中断的状态改变的化，抛出中断异常后方法直接退出
 if (Thread.interrupted())
 throw new InterruptedException();
 if (!tryAcquire(arg))
 //同步状态获取失败，调用下面这个方法。
 doAcquireInterruptibly(arg);
}



protected final boolean tryAcquire(int acquires) {
    return nonfairTryAcquire(acquires); //调用 nonfairTryAcquire 此时 acq
uires==1 (想要获得锁)
}
nonfairTryAcquire（acquires:1）
final boolean nonfairTryAcquire(int acquires) {
    //当前线程
    final Thread current = Thread.currentThread();
    int c = getState();//获得 当前锁的状态（1：有锁 0：无锁）
    if (c == 0) { //当前锁状态 为无锁
        if (compareAndSetState(0, acquires)) {  //尝试CAS获取锁 acquire为
1
            setExclusiveOwnerThread(current);   //将当前线程设置为当前独占
线程
            return true;        //返回true - tryAcquire() 返回true —— acq
uire获得同步状态退出
        }
    }
    else if (current == getExclusiveOwnerThread()) {//当前锁状态为有锁 判
断当前获得锁是否是当前线程（锁的重入）
        int nextc = c + acquires;         //锁标记设置为 c+1 ： 引用计数+1
锁的重入
```

```
30          if (nextc < 0) // overflow        //如果锁标记小于0  则抛出异常 最大锁计数超出
31              throw new Error("Maximum lock count exceeded");
32          setState(nextc);                    //锁标记大于0，则更新锁标记+1（引用计数）
33          return true;//返回true - tryAcquire() 返回true —— acquire获得同步状态退出
34                      //（当前锁就是本线程持有，只是进行了锁的重入）
35      }
36      return false; //当前或的锁的线程不是本线程，返回false:
37   //tryAcquire() 返回false 进入 doAcquireInterruptibly(arg);
38  }
```

//同步状态获取失败，调用doAcquireInterruptibly（）方法。

```
1  private void doAcquireInterruptibly(int arg)
2   throws InterruptedException {
3   final Node node = addWaiter(Node.EXCLUSIVE);
4   boolean failed = true;
5   try {
6   for (;;) {
7   final Node p = node.predecessor();
8   if (p == head && tryAcquire(arg)) {
9   setHead(node);
10   p.next = null; // help GC
11   failed = false;
12   return;
13   }
14   if (shouldParkAfterFailedAcquire(p, node) &&
15   parkAndCheckInterrupt())
16   //即
17   //线程被阻阻塞是若检测到中断抛出中断异常退出
18   throw new InterruptedException();
19   }
20   } finally {
21   if (failed)
22   cancelAcquire(node);
23   }
24  }
```