Condition

概念

Condition是在java1.5中才出现的，它用来替代传统的Object的wait()、notify()实现线程间的协作，相比使用Object的wait()、notify()，使用Condition的await()、signal()这种方式实现线程间协作更加安全和高效。因此通常来说比较推荐使用Condition，阻塞队列实际上是使用了Condition来模拟线程间协作

Condition和Object对象的wait()、notify()的区别

1. Condition可以具体的让某一个线程等待，可以唤醒指定的线程，Object只能唤醒当前的线程或者唤醒全部的线程
2. Condition必须和Lock锁配合来使用，Object必须配合synchronized使用

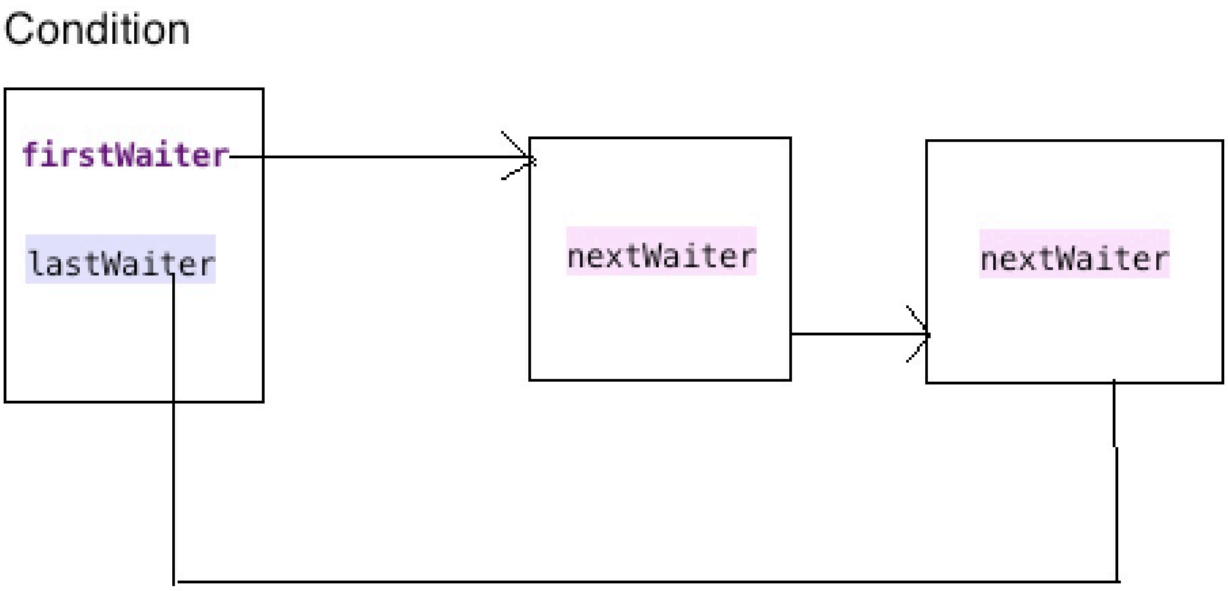
Condition实现类的源码分析

Condition是一个接口，它的实现类只有一个就ConditionObject它是AbstractQueuedLongSynchronizer的内部类，我们主要看它的await、signal方法和等待队列

等待队列
等待队列是一个FIFO的队列，在队列中每一个节点Node都包含着一个线程的引用，该线程就是在该Condition对象上等待的线程，如果一个线程调用了condition.await()方法，该线程将会释放锁，构造成一个Node加入到等待队列，进行等待。

```
1 public class ConditionObject implements Condition, java.io.Serializable {
2
3    private transient Node firstWaiter;
4
5    private transient Node lastWaiter;
6
7    public ConditionObject() {
8    }
9 }
```

一个Condition包含一个等待队列，当前线程调用condition.await()方法得时候，将会以当前线程构造节点，并且将节点加入到等待队列，等待队列得结构图如下所示



队列中的一个节点包含一个线程
我们再来分析一下Condition.await()方法

```
1 public final void await() throws InterruptedException {
2     if (Thread.interrupted())
3         throw new InterruptedException();
4     Node node = addConditionWaiter();
5     long savedState = fullyRelease(node);
6     int interruptMode = 0;
7     while (!isOnSyncQueue(node)) {
8         LockSupport.park(this);
9         if ((interruptMode = checkInterruptWhileWaiting(node)) != 0)
10            break;
11    }
12    if (acquireQueued(node, savedState) && interruptMode != THROW_IE)
13        interruptMode = REINTERRUPT;
14    if (node.nextWaiter != null) // clean up if cancelled
15        unlinkCancelledWaiters();
16    if (interruptMode != 0)
17        reportInterruptAfterWait(interruptMode);
18 }
```

我们来一步步分析一下这部分源码
Node node = addConditionWaiter();当前线程加入到等待队列中
我们在具体的看看addConditionWaiter的源码

```
1 private Node addConditionWaiter() {
2     Node t = lastWaiter;//暂存等待队列的尾节点
3     // If lastWaiter is cancelled, clean out.
4     清除等待队列队尾的状态不是等待的节点
5     if (t != null && t.waitStatus != Node.CONDITION) {
6         //清除队列中所有状态不为Condition的节点
7         unlinkCancelledWaiters();
8         t = lastWaiter;
9     }
10    //创建一个Node节点，把当前的线程设置到节点中的线程上，设置节点的状态
11    Node node = new Node(Thread.currentThread(), Node.CONDITION);
12    //如果firstWaiter为null，说明等待队列还没有加入任何等待的线程
```

```
    if (t == null)
        firstWaiter = node;
    else
        //如果不为null说明等待队列之前已经加入过等待线程
        //把当前线程加入到等待队列的队尾
        t.nextWaiter = node;
    lastWaiter = node;
    return node;
}
```

我们在来详细分析一下unlinkCancelledWaiters的代码

```
private void unlinkCancelledWaiters() {
    Node t = firstWaiter;
    Node trail = null;
    while (t != null) {
        Node next = t.nextWaiter;
        if (t.waitStatus != Node.CONDITION) {
            t.nextWaiter = null;
            if (trail == null)
                firstWaiter = next;
            else
                trail.nextWaiter = next;
            if (next == null)
                lastWaiter = trail;
        }
        else
            trail = t;
        t = next;
    }
}
```

把队尾的节点设置为null,分为了2中情况，第一，设置队尾设置为null之后该队列就没有任何元素了
第二，设置队尾设置为null之后该队列任然有元素
总体来说思路就是，把队尾设置为null

long savedState = fullyRelease(node);当前线程进入到等待队列的时候必须释放所，这个方法就是释放锁的方法
在java.util.concurrent.locks.AbstractQueuedLongSynchronizer类中

```
final long fullyRelease(Node node) {
    boolean failed = true;
    try {
        //判断锁的状态
        long savedState = getState();
        //释放锁，改变锁的状态
        //把持有锁的线程从队列中移除
        //设置释放锁是否成功
        if (release(savedState)) {
            failed = false;
            return savedState;
        } else {
            throw new IllegalMonitorStateException();
        }
    } finally {
        if (failed)
            node.waitStatus = Node.CANCELLED;
    }
```

```
}
```

```
while (!isOnSyncQueue(node)) {
    //线程挂起，线程处于等待状态
    LockSupport.park(this);
    if ((interruptMode = checkInterruptWhileWaiting(node)) != 0)
        break;
}
```

判断Node节点是否在同步队列上，如果在，说明该线程还没有被唤醒没有竞争锁的资格不会执行，如果不在说明该线程有执行权，不处于等待状态
if (acquireQueued(node, savedState) && interruptMode != THROW_IE)
interruptMode = REINTERRUPT;
把线程加入到同步队列中