链滴

java：一个 wait（timeout） 引出的你可能对锁的误解

作者：mainlove

原文链接：https://ld246.com/article/1488015279637

来源网站：链滴

在很多讲wait（long timeout）的例子，都会用下面类似的代码：

```
public class RunA implements Runnable {

    private Object lock;

    public RunA(Object lock) {
        this.lock = lock;
    }

   @Override
    public void run() {

        synchronized (lock){
            try  {
                System.out.println("A begin");
               // lock.wait(); // 永远等待着，不会执行下去
                lock.wait(2000);// 等待了2秒之后，继续执行下去
                System.out.println("A end");
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

举这样的例子显然是没有任何意义的，在这里用wait（2000）和sleep（2000） 有什么区别呢?

wait和sleep显然是有很大的区别，但区别不只是wait会把lock释放掉，然我们引入一个新的搅和线程B

```
public class RunB implements Runnable {

    private Object lock;

    public RunB(Object lock) {
        this.lock = lock;
    }

    @Override
   public void run() {

        synchronized (lock) {
            System.out.println("b come");
            while (true) {
            }
        }
    }
}
```

B仅仅是握住锁，然后永远不释放，然后回到我们的主舞台main函数：

```
public static void main(String[] args) throws InterruptedException {

    Object lock = new Object();
    Thread threadA = new Thread(new RunA(lock));
    threadA.start();
    threadA.wait();

    Thread.sleep(1000);

    Thread threadB = new Thread(new RunB(lock));
    threadB.start();

}
```

然后再run一下，发现A end ying永远不会打印了，咦，为啥wait（2000）之后没有被唤醒执行下去呢?

仔细想想A显示获得了锁，然后wait（2000）交出了锁，然后B拿到了锁，这个时候过了2秒，A确实被唤醒了，但很可惜A永远也不会得到锁了，对于临界区永远只能有一个线程在执行，不可能出现两临界区同时在执行代码的可能，所以被唤醒之后，还需要去争抢锁，并不是唤醒了就能继续执行代码的

一个线程被唤醒可能有一下四种情况

1. 其它的线程调用 obj.notify()，且当前线程T，正好是被选中唤醒的。
2. 其它的线程调用 obj.notifyAll()。
3. 其它线程中断T。
4. 指定的等待时间（timeout）超时，（时间精度会有些误差）。

但我想说一下一个完整的过程是，**唤醒之后需要去抢到临界区的锁，才能真正把代码执行下去**，光有醒是不够的

大多数时候我们忽略唤醒之后需要去抢到临界区的锁，是因为notify用的多的关系，触发notify的线必然有锁，只会唤醒一个线程，所以被唤醒的线程必然得到锁！于是大家就会产生一个被唤醒一定能行点的错觉

而我觉得很多文章没有指明这一点，然当你意识到了这个之后，对notifyAll就也不会有一起全部唤醒行的想当然理解了，notifyAll只是让大家都去抢临界区，所有的wait notify，都是为了**保护临界区永只能有一个在执行**，分析问题还是从源头入手，见笑。。