https://mp.weixin.qq.com/s?\_\_biz=MzA5MTkxMDQ4MQ==&mid=2648933120&idx=1&sn=63ffe3ff64dcaf0418816febfd1e129a&chksm=88621b3ebf159228df5f5a501160fafa5d87412a4f03298867ec9325c0be57cd8e329f3b5ad1&token=476165288&lang=zh\_CN&scene=21#wechat\_redirect

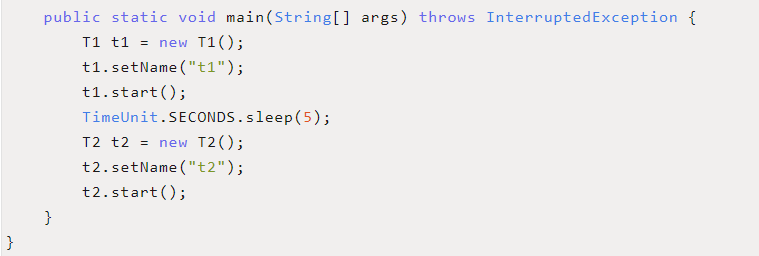
# JUC中的Condition对象

## 本文内容

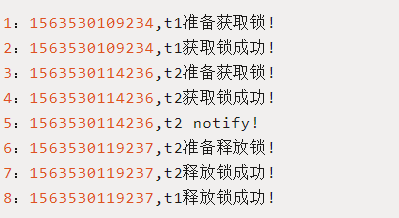
1. synchronized中实现线程等待和唤醒
2. Condition简介及常用方法介绍及相关示例
3. 使用Condition实现生产者消费者
4. 使用Condition实现同步阻塞队列

## synchronized中等待和唤醒线程示例





输出：



代码结合输出的结果我们分析一下：

1. 线程t1先获取锁，然后调用了wait()方法将线程置为等待状态，然后会释放lock的锁
2. 主线程等待5秒之后，启动线程t2，t2获取到了锁，结果中1、3行时间相差5秒左右
3. t2调用lock.notify()方法，准备将等待在lock上的线程t1唤醒，notify()方法之后又休眠了5秒，看一下输出的5、8可知，**notify()方法之后，t1并不能立即被唤醒，需要等到t2将synchronized块执行完毕，释放锁之后，t1才被唤醒**
4. wait()方法和notify()方法必须放在同步块内调用（synchronized块内），否则会报错。

## Condition使用简介

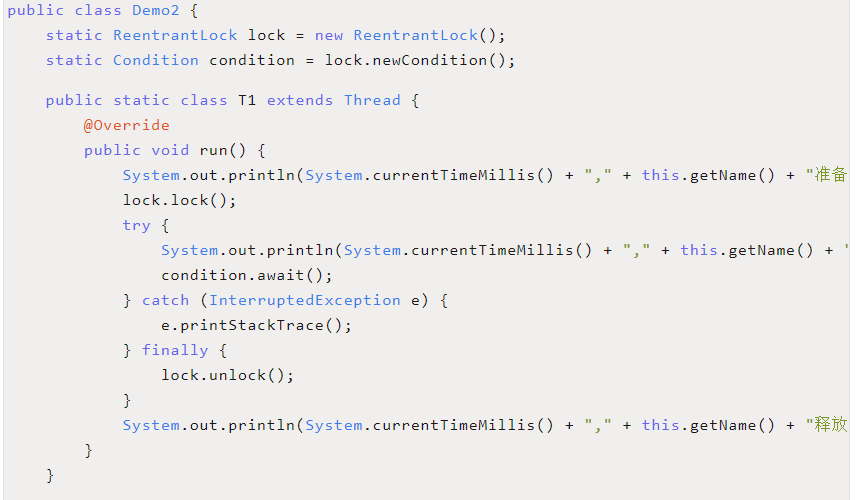
任何一个java对象都天然继承于Object类，在线程间实现通信的往往会应用到Object的几个方法，比如wait()、wait(long timeout)、wait(long timeout, int nanos)与notify()、notifyAll()几个方法实现等待/通知机制，同样的， 在java Lock体系下依然会有同样的方法实现等待/通知机制。

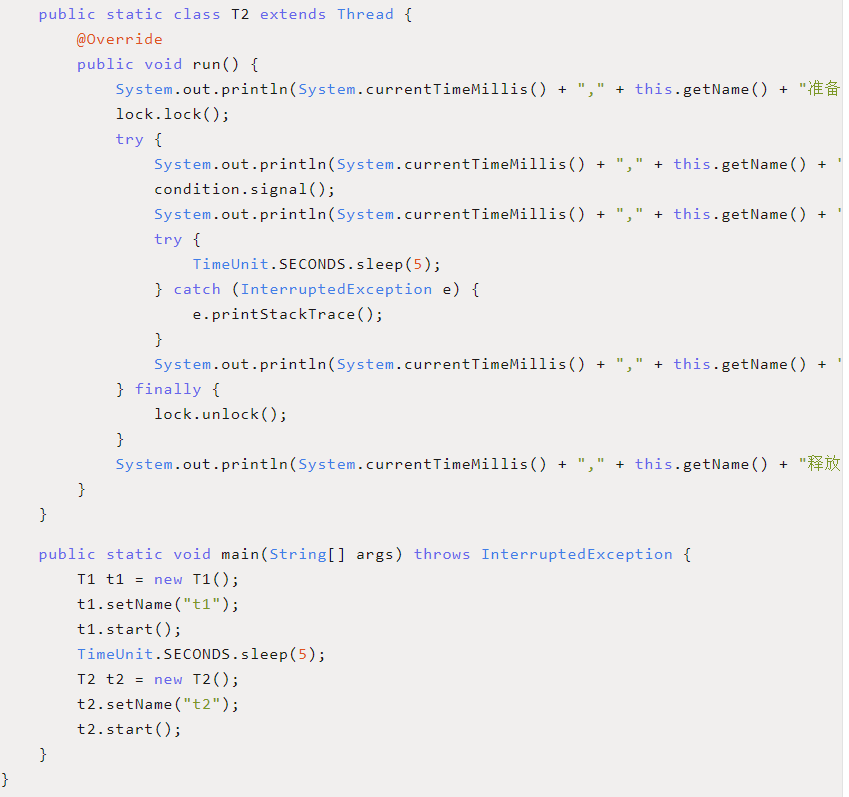
从整体上来看Object的wait和notify/notify是与对象监视器配合完成线程间的等待/通知机制，而Condition与Lock配合完成等待通知机制，前者是java底层级别的，后者是语言级别的，具有更高的可控制性和扩展性。两者除了在使用方式上不同外，在功能特性上还是有很多的不同：

* Condition能够支持不响应中断，而通过使用Object方式不支持
* Condition能够支持多个等待队列（new 多个Condition对象），而Object方式只能支持一个
* Condition能够支持超时时间的设置，而Object不支持

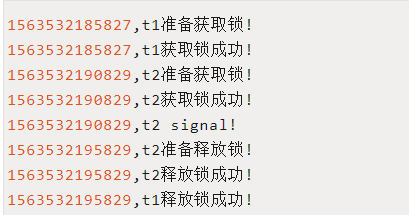
Condition由ReentrantLock对象创建，并且可以同时创建多个，**Condition接口在使用前必须先调用ReentrantLock的lock()方法获得锁**，之后调用Condition接口的await()将释放锁，并且在该Condition上等待，直到有其他线程调用Condition的signal()方法唤醒线程，使用方式和wait()、notify()类似。

示例代码：





输出：



输出的结果和使用synchronized关键字的实例类似。

Condition.await()方法和Object.wait()方法类似，当使用Condition.await()方法时，需要先获取Condition对象关联的ReentrantLock的锁，**在Condition.await()方法被调用时，当前线程会释放这个锁，并且当前线程会进行等待（处于阻塞状态）**。在signal()方法被调用后，系统会从Condition对象的等待队列中唤醒一个线程，一旦线程被唤醒，被唤醒的线程会尝试重新获取锁，一旦获取成功，就可以继续执行了。因此，在signal被调用后，一般需要释放相关的锁，让给其他被唤醒的线程，让他可以继续执行。

## Condition常用方法

Condition接口提供的常用方法有：

### **和Object中wait类似的方法**

1. void await() throws InterruptedException:当前线程进入等待状态，如果其他线程调用condition的signal或者signalAll方法并且当前线程获取Lock从await方法返回，如果在等待状态中被中断会抛出被中断异常；
2. long awaitNanos(long nanosTimeout)：当前线程进入等待状态直到被通知，中断或者超时；
3. boolean await(long time, TimeUnit unit) throws InterruptedException：同第二种，支持自定义时间单位，false：表示方法超时之后自动返回的，true：表示等待还未超时时，await方法就返回了（超时之前，被其他线程唤醒了）
4. boolean awaitUntil(Date deadline) throws InterruptedException：当前线程进入等待状态直到被通知，中断或者到了某个时间
5. **void awaitUninterruptibly();：当前线程进入等待状态，不会响应线程中断操作，只能通过唤醒的方式让线程继续**

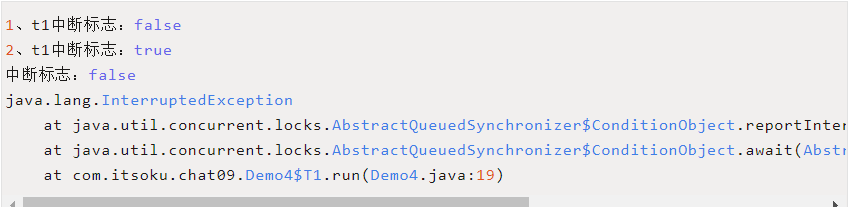
### 和Object的notify/notifyAll类似的方法

1. void signal()：唤醒一个等待在condition上的线程，将该线程从等待队列中转移到同步队列中，如果在同步队列中能够竞争到Lock则可以从等待方法中返回。
2. void signalAll()：与1的区别在于能够唤醒所有等待在condition上的线程

## Condition.await()过程中被打断



输出：

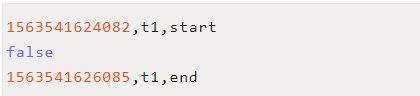


调用condition.await()之后，线程进入阻塞中，调用t1.interrupt()，给t1线程发送中断信号，await()方法内部会检测到线程中断信号，然后触发 InterruptedException异常，线程中断标志被清除。从输出结果中可以看出，线程t1中断标志的变换过程：false->true->false

## await(long time, TimeUnit unit)超时之后自动返回



输出：

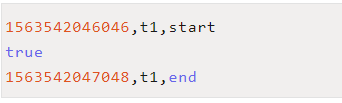


t1线程等待2秒之后，自动返回继续执行，最后await方法返回false，**await返回false表示超时之后自动返回。**

## await(long time, TimeUnit unit)超时之前被唤醒



输出：

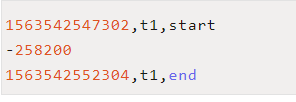


t1线程中调用 condition.await(5,TimeUnit.SECONDS);方法会释放锁，等待5秒，主线程休眠1秒，然后获取锁，之后调用signal()方法唤醒t1，输出结果中发现await后过了1秒（1、3行输出结果的时间差），await方法就返回了，并且返回值是true。**true表示await方法超时之前被其他线程唤醒了。**

## long awaitNanos(long nanosTimeout)超时返回



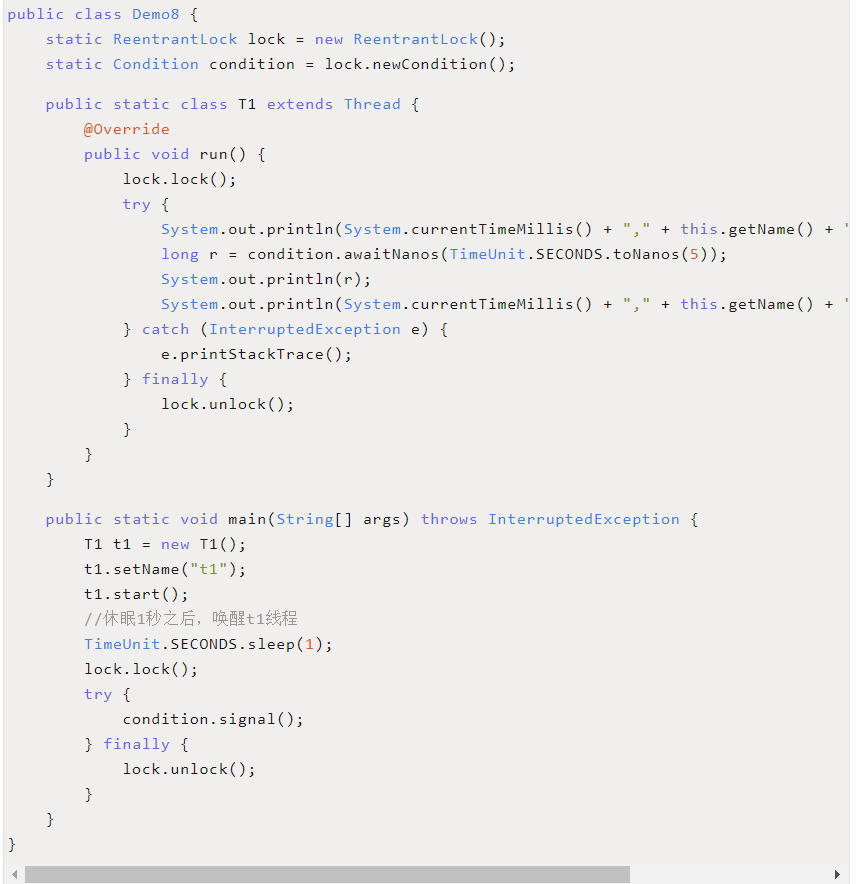
输出：



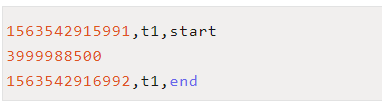
awaitNanos参数为纳秒，可以调用TimeUnit中的一些方法将时间转换为纳秒。

t1调用await方法等待5秒超时返回，**返回结果为负数，表示超时之后返回的。**

## waitNanos(long nanosTimeout)超时之前被唤醒



输出：



t1中调用await休眠5秒，主线程休眠1秒之后，调用signal()唤醒线程t1，await方法返回正数，表示返回时距离超时时间还有多久，将近4秒，返回正数表示，线程在超时之前被唤醒了。

其他几个有参的await方法和无参的await方法一样，线程调用interrupt()方法时，这些方法都会触发InterruptedException异常，并且线程的中断标志会被清除。

## 同一个锁支持创建多个Condition

使用两个Condition来实现一个阻塞队列的例子：







代码非常容易理解，创建了一个阻塞队列，大小为3，队列满的时候，会被阻塞，等待其他线程去消费，队列中的元素被消费之后，会唤醒生产者，生产数据进入队列。上面代码将队列大小置为1，可以实现同步阻塞队列，生产1个元素之后，生产者会被阻塞，待消费者消费队列中的元素之后，生产者才能继续工作。

## Object的监视器方法与Condition接口的对比

| **对比项** | **Object 监视器方法** | **Condition** |
| --- | --- | --- |
| 前置条件 | 获取对象的锁 | 调用Lock.lock获取锁，调用Lock.newCondition()获取Condition对象 |
| 调用方式 | 直接调用，如：object.wait() | 直接调用，如：condition.await() |
| 等待队列个数 | 一个 | 多个，使用多个condition实现 |
| 当前线程释放锁并进入等待状态 | 支持 | 支持 |
| 当前线程释放锁进入等待状态中不响应中断 | 不支持 | 支持 |
| 当前线程释放锁并进入超时等待状态 | 支持 | 支持 |
| 当前线程释放锁并进入等待状态到将来某个时间 | 不支持 | 支持 |
| 唤醒等待队列中的一个线程 | 支持 | 支持 |
| 唤醒等待队列中的全部线程 | 支持 | 支持 |

## 总结

1. 使用condition的步骤：创建condition对象，获取锁，然后调用condition的方法
2. 一个ReentrantLock支持创建多个condition对象
3. Void await() throws InterruptedException;方法会释放锁，让当前线程等待，支持唤醒，支持线程中断
4. Void awaitUninterruptibly();方法会释放锁，让当前线程等待，支持唤醒，不支持线程中断
5. Long awaitNanos(long nanosTimeout) throws InterruptedException;参数为纳秒，此方法会释放锁，让当前线程等待，支持唤醒，支持中断。超时之后返回的，结果为负数；超时之前被唤醒返回的，结果为正数（表示返回时距离超时时间相差的纳秒数）
6. Boolean await(longtime,TimeUnitunit) throws InterruptedException;方法会释放锁，让当前线程等待，支持唤醒，支持中断。超时之后返回的，结果为false；超时之前被唤醒返回的，结果为true
7. Boolean awaitUntil(Date deadline) throws InterruptedException;参数表示超时的截止时间点，方法会释放锁，让当前线程等待，支持唤醒，支持中断。超时之后返回的，结果为false；超时之前被唤醒返回的，结果为true
8. Void signal();会唤醒一个等待中的线程，然后被唤醒的线程会被加入同步队列，去尝试获取锁
9. Void signalAll();会唤醒所有等待中的线程，将所有等待中的线程加入同步队列，然后去尝试获取锁