wait 和 notify：

1、wait()方法

2、notify()方法

3、notifyAll()方法

4、wait 和 sleep 的对比（面试题）

wait 和 notify：

wait ： 等待

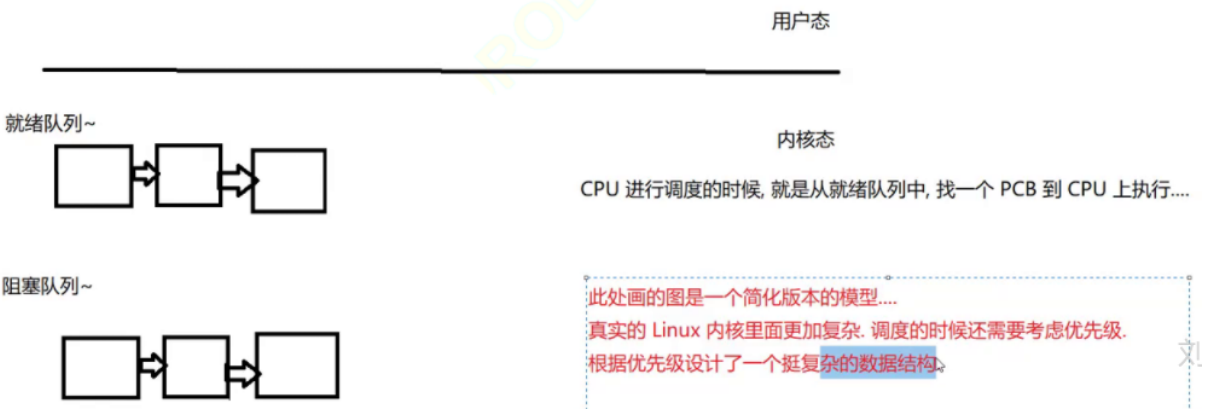
notify：通知

用来协同多个线程之间的执行顺序。

由于线程之间是抢占式执行的, 因此线程之间执行的先后顺序难以预知.

但是实际开发中有时候我们希望合理的协调多个线程之间的执行先后顺序.

通过 wait 和 notify 机制就可以来控制线程之间的执行顺序，让多个线程之间更好的相互配合!



wait 做了三件事：

1.让当前线程阻塞等待（让这个线程的 PCB 从就绪对列中拿到等待就队列中），并准备接受通知

2.释放当前锁，要想使用 wait / notify,必须搭配 synchronized,需要先拿到锁，才有资格谈 wait

3.满足一定条件时被唤醒，重新尝试获取到这个锁

注：1 和 2 是要原子的完成

wait, notify, notifyAll 都是 Object 类的方法.比如：线程1 和 线程2 中的对象1调用了wait，必须要有个线程2，也调用了对象1的notify，才能唤醒线程1.如果是线程2 调用了对象2的 notify，就无法唤醒线程1.

1、wait()方法

wait 做的事情:

a. 使当前执行代码的线程进行等待. (把线程放到等待队列中)

b. 释放当前的锁

c. 满足一定条件时被唤醒, 重新尝试获取这个锁.

注：wait 要搭配 synchronized 来使用. 脱离 synchronized 使用 wait 会直接抛出异常.

如果没有把 wait 放在 synchronized 内部, 就能看到这个异常.

```
synchronized (object) {
    object.wait();
}
```

调用 wait 的对象和锁对象也得是同一个.

wait 结束等待的条件:

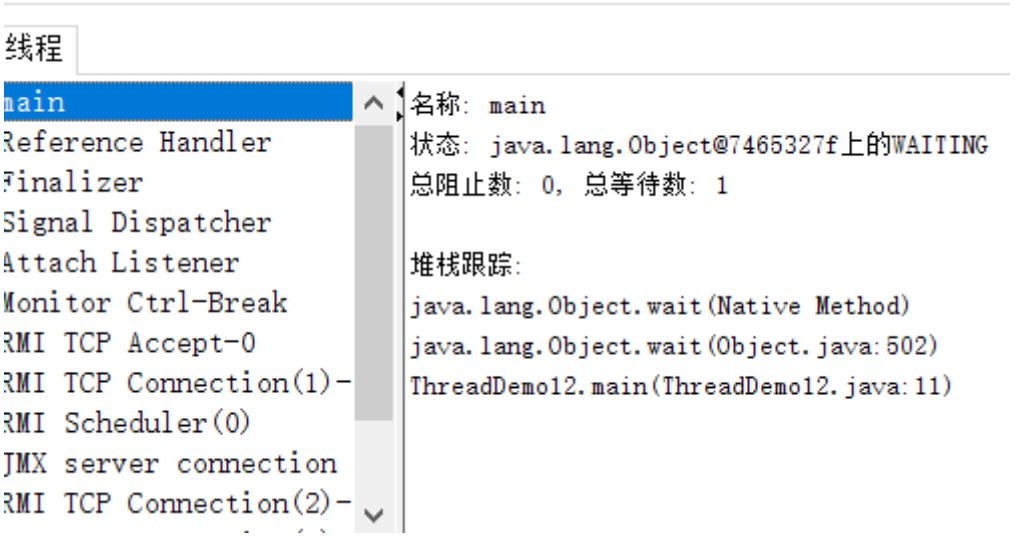
1.其他线程调用该对象的 notify 方法.

2.wait 等待时间超时 (wait 方法提供一个带有 timeout 参数的版本, 来指定等待时间).

3.其他线程调用该等待线程的 interrupted 方法, 导致 wait 抛出 InterruptedException 异常.

代码示例: 观察wait()方法使用

```
1 public class ThreadDemo12 {
2  public static void main(String[] args) throws InterruptedExcept
ion {
3  Object o = new Object();
4  synchronized(o) {
5  System.out.println("wait 之前");
6  o.wait();
7  System.out.println("wait 之后");
8  }
9  }
10 }
```



这样在执行到object.wait()之后就一直等待下去，那么程序肯定不能一直这么等待下去了。这个时候就

需要使用到了另外一个方法唤醒的方法notify()。

2、notify()方法

notify 方法是唤醒等待的线程.

1.方法notify()也要在同步方法或同步块中调用，该方法是用来通知那些可能等待该对象的对象锁的其它线程，对其发出通知notify，并使它们重新获取该对象的对象锁。

2.如果有多个线程等待，则有线程调度器随机挑选出一个呈 wait 状态的线程。(并没有 "先来后到")

3.在notify()方法后，当前线程不会马上释放该对象锁，要等到执行notify()方法的线程将程序执行完，也就是退出同步代码块之后才会释放对象

锁。

代码示例: 使用notify()方法唤醒线程

1. 创建 WaitTask 类, 对应一个线程, run 内部循环调用 wait.
2. 创建 NotifyTask 类, 对应另一个线程, 在 run 内部调用一次 notify
3. 注意, WaitTask 和 NotifyTask 内部持有同一个 Object locker.

WaitTask 和 NotifyTask 要想配合就需要搭配同一个 Object.

```
public class ThreadDemo13 {
 static class WaitTask implements Runnable{
 Object locker = null;
 public WaitTask(Object locker) {
 this.locker = locker;
 }

 @Override
 public void run() {
  synchronized (locker) {
  // 进行 wait 的线程
  System.out.println("wait 开始");
  try {
  // 直接调用 wait 相当于 this.wait(),也就是针对于 WaitTask 对象来进行等待
  // 但是在 NotifyTask 中要求针对同一个对象进行通知,然而,在 NotifyTask 中
  // 并没有容易拿到 WaitTask 实例
  locker.wait();
  } catch (InterruptedException e) {
  e.printStackTrace();
  }
  System.out.println("wait 结束");
  }
  }
  }

  static class NotifyTask implements Runnable{
  Object locker = null;
```

```
28
29  public NotifyTask(Object locker) {
30  this.locker = locker;
31  }
32
33  @Override
34  public void run() {
35  // 进行 notify 的线程
36  synchronized (locker){
37  System.out.println("notify 开始");
38  // 随机唤醒一个线程
39  locker.notify();
40  System.out.println("notify 结束");
41  }
42  }
43  }
44  public static void main(String[] args) throws InterruptedExcep
tion {
45  // 为了解决刚才的问题,专门创建一个对象,去负责进行加锁/通知操作
46  Object locker = new Object();
47  Thread t10 = new Thread(new WaitTask(locker));
48  Thread t11 = new Thread(new WaitTask(locker));
49  Thread t12 = new Thread(new WaitTask(locker));
50  Thread t2 = new Thread(new NotifyTask(locker));
51  t10.start();
52  t11.start();
53  t12.start();
54  Thread.sleep(3000);
55  t2.start();
56  }
57 }
```

3、notifyAll()方法

notify方法只是唤醒某一个等待线程. 使用notifyAll方法可以一次唤醒所有的等待线程.

范例：使用notifyAll()方法唤醒所有等待线程, 在上面的代码基础上做出修改.

- 创建 3 个 WaitTask 实例. 1 个 NotifyTask 实例.

```
1  public static void main(String[] args) throws InterruptedExcept
ion {
2  // 为了解决刚才的问题,专门创建一个对象,去负责进行加锁/通知操作
3  Object locker = new Object();
4  Thread t10 = new Thread(new WaitTask(locker));
5  Thread t11 = new Thread(new WaitTask(locker));
6  Thread t12 = new Thread(new WaitTask(locker));
7  Thread t2 = new Thread(new NotifyTask(locker));
8  t10.start();
9  t11.start();
10  t12.start();
11  Thread.sleep(3000);
12  t2.start();
13  }
```

此时可以看到, 调用 notify 只能唤醒一个线程.

- 修改 NotifyTask 中的 run 方法, 把 notify 替换成 notifyAll

```
1 public void run() {
2  synchronized (locker) {
3  System.out.println("notify 开始");
4  locker.notifyAll();
5  System.out.println("notify 结束");
6  }
7 }
```

此时可以看到, 调用 notifyAll 能同时唤醒 3 个wait 中的线程

注意: 虽然是同时唤醒 3 个线程, 但是这 3 个线程需要竞争锁. 所以并不是同时执行, 而仍然是有先有后的执行.

4、wait 和 sleep 的对比（面试题）

其实理论上 wait 和 sleep 完全是没有可比性的，因为一个是用于线程之间的通信的，一个是让线程阻塞一段时间，

1.sleep操作需要一个固定的时间来阻塞等待，wait 既可以指定时间，也可以无限等待

2. wait 唤醒可以通过 notify 或者 interrupt 或者时间到来唤醒，sleep 唤醒通过时间到或者 interrupt 唤醒

3. wait 的主要用途是为了协调线程之间的先后顺序，这样的场景并不适用 sleep，sleep 单纯让该线程休眠，并不涉及到多个线程的配合