1. wait() 方法

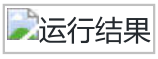
使线程暂停运行。

- 使当前执行代码的线程进行等待。wait 是 Object类 的方法，该方法是用来将当前线程置入 预执行队列种，并且在 wait 所在的代码处停止执行，直到接到通知(notify)或被中断为止。
- wait 只能在同步方法或同步代码块中被调用。如果调用 wait 时，没有持有适当的锁，会抛出异常。
- wait 执行后，当前线程释放锁，线程与其他线程竞争重新获取锁 notify方法(使停止的进程继续运行)。

来看一个例子。

```
class MyThread implements Runnable {

    @Override
    public void run() {
        // wait() 方法是 Object类 的方法
        Object object = new Object();
        // wait() 方法必须定义在同步方法或同步代码块中
        synchronized (object) {
            System.out.println("进入线程"+Thread.currentThread().getName());
            try {
                object.wait();
            } catch (InterruptedException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
            System.out.println("退出线程"+Thread.currentThread().getName());
        }
    }
}

public class Test {
    public static void main(String[] args) {
        new Thread(new MyThread(), "wait线程").start();
    }
}
```

运行结果：



因为没有得到通知(notify)或被中断，所以程序一直在运行着。

2. notify() 方法

唤醒一个正在等待的进程。

- 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对象的对象锁的其他线程，对其发出通知 notify，并使他们重新获得该对象的对象锁。

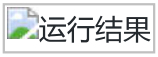
- 另外，notify方法唤醒之后，当前线程不会马上释放对象锁，而是要等到当前 notify方法 的同步块/方法将程序执行完，即退出同步代码块之后才会释放对象锁，再执行还行的wait进程。
- 如果多个线程等待，则线程规化器随机挑选出一个呈 wait 状态的线程。

  再看一个例子：

```
class MyThread implements Runnable {

    private boolean flag;
    private Object Object;

    public MyThread(boolean flag, Object object) {
        super();
        this.flag = flag;
        this.Object = object;
    }

    public void waitMethod() {
        synchronized (Object) {
            while(true) {
                try {
                    System.out.println("wait方法开始 "+Thread.currentThread().getName());
                    Object.wait();
                    System.out.println("wait方法结束 "+Thread.currentThread().getName());
                    return;
                } catch (InterruptedException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }

            }
        }
    }

    public void notifyMethod() {
        synchronized (Object) {
            while(true) {
                System.out.println("notify方法开始 "+Thread.currentThread().getName());
                Object.notify();
                System.out.println("notify方法结束 "+Thread.currentThread().getName());
                return;
            }
        }
    }

    @Override
    public void run() {
        if(flag) {
            this.waitMethod();
        } else {
            this.notifyMethod();
        }
    }
}


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

```
        Object object = new Object();
        MyThread waitThread = new MyThread(true, object);
        MyThread notifyThread = new MyThread(false, object);

        new Thread(waitThread, "wait线程").start();
        Thread.sleep(3000);
        new Thread(notifyThread, "notify线程").start();
        System.out.println("main方法结束");
    }
}
```

运行结果：



另外，如果有多个线程等待，则有线程规划器随机挑选出一个呈wait状态的线程进行唤醒。 我们将上面的程序进行一些修改。

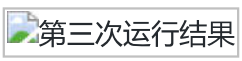
```
public class Test {
    public static void main(String[] args) throws InterruptedException {
        Object object = new Object();
        MyThread waitThread1 = new MyThread(true, object);
        MyThread waitThread2 = new MyThread(true, object);
        MyThread waitThread3 = new MyThread(true, object);
        MyThread notifyThread = new MyThread(false, object);

        new Thread(waitThread1, "wait1线程").start();
        new Thread(waitThread2, "wait2线程").start();
        new Thread(waitThread3, "wait3线程").start();
        Thread.sleep(3000);
        new Thread(notifyThread, "notify线程").start();
        System.out.println("main方法结束");
    }
}
```

我们将这个程序运行三次，看一下三次的结果。

第一次运行结果：

第二次运行结果：

第三次运行结果：



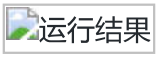
我们可以明显的看出，同样的程序，运行三次，唤醒的线程会不一样，这就证明了线程规划器随机挑选出一个呈wait状态的线程进行唤醒。

3. notifyAll() 方法

唤醒所有等待的进程。　　将上面例子中 MyThread类 notifyMethod方法进行修改如下：

```
public void notifyMethod() {
    synchronized (Object) {
        while(true) {
            System.out.println("notify方法开始 "+Thread.currentThread().getName());
            // 唤醒所有等待的线程
            Object.notifyAll();
            System.out.println("notify方法结束 "+Thread.currentThread().getName());
            return;
        }
    }
}
```

运行结果：



同样，notifyAll 唤醒多个进程的顺序也是不定的。