java中，wait和notify两个方法是一对，wait方法阻塞当前线程，而notify是唤醒被wait方法阻塞的线程。
需要说明的是，wait和notify方法都是Object的实例方法，要执行这两个方法，有一个前提就是，当前线程必须获得对象的锁。
当前线程A获得对象的锁后，然后进入临界区（同步代码块），调用对象的wait方法，则线程A释放对象的锁后，不用等到执行完临界区，因为线程A会被阻塞在当前位置，同时cpu的相关寄存器会记住当前位置的堆栈信息，然后进入阻塞状态，线程A让出cpu，不再参与cpu的竞争，同时wait方法内部会不断地轮询线程A的interruptStatus状态位，以判断当前阻塞的状态是否被中断，等待其他线程调用A的notify来唤醒资源。然后线程B获取对象的锁之后，进入临界区，执行对象的notify方法，这时候，线程B不会立即释放对象的锁同时唤醒线程A，而是要等到线程B执行完同步代码块后，这时候才会释放对象的锁。

```
import java.util.ArrayList;
import java.util.List;

public class MyList {

    private static List<String> list = new ArrayList<String>();

    public static void add() {
        list.add("anyString");
    }

    public static int size() {
        return list.size();
    }
}


public class ThreadA extends Thread {

    private Object lock;

    public ThreadA(Object lock) {
        super();
```

```
        this.lock = lock;
    }

    @Override
    public void run() {
        try {
            synchronized (lock) {
                if (MyList.size() != 5) {
                    System.out.println("wait begin "
                            + System.currentTimeMillis());
                    lock.wait();
                    System.out.println("wait end  "
                            + System.currentTimeMillis());
                }
            }
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}


public class ThreadB extends Thread {
    private Object lock;

    public ThreadB(Object lock) {
        super();
        this.lock = lock;
    }

    @Override
    public void run() {
        try {
            synchronized (lock) {
                for (int i = 0; i < 10; i++) {
```

```
                    MyList.add();
                    if (MyList.size() == 5) {
                        lock.notify();
                        System.out.println("已经发出了通知");
                    }
                    System.out.println("添加了" + (i + 1) + "个元素!");
                    Thread.sleep(1000);
                }
            }
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}

public class Run {

    public static void main(String[] args) {

        try {
            Object lock = new Object();

            ThreadA a = new ThreadA(lock);
            a.start();

            Thread.sleep(50);

            ThreadB b = new ThreadB(lock);
            b.start();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

线程A要等待某个条件满足时(list.size()==5)，才执行操作。线程B则向list中添加元素，改变list 的size。

A,B之间如何通信的呢？也就是说，线程A如何知道 list.size() 已经为5了呢?

这里用到了Object类的 wait() 和 notify() 方法。

当条件未满足时(list.size() !=5)，线程A调用wait() 放弃CPU，并进入阻塞状态。---不像②while轮询那样占用CPU

当条件满足时，线程B调用 notify()通知 线程A，所谓通知线程A，就是唤醒线程A，并让它进入可运行状态。

这种方式的一个好处就是CPU的利用率提高了。

执行结果：

```
wait begin 1528766276221
添加了1个元素!
添加了2个元素!
添加了3个元素!
添加了4个元素!
已经发出了通知
添加了5个元素!
添加了6个元素!
添加了7个元素!
添加了8个元素!
添加了9个元素!
添加了10个元素!
wait end  1528766286402
```