# 多线程之间如何实现通讯

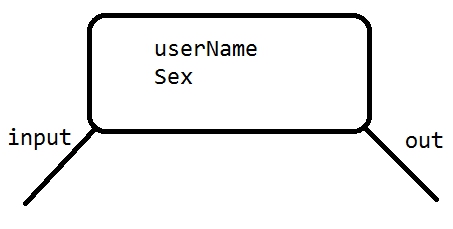
## 什么是多线程之间通讯？

多线程之间通讯，其实就是多个线程在操作同一个资源，但是操作的动作不同。

画图演示

## 多线程之间通讯需求

需求:第一个线程写入(input)用户，另一个线程取读取(out)用户.实现读一个，写一个操作。



代码实现基本实现

### 共享资源源实体类

|  |
| --- |
| **class Res {**  **public String userSex;**  **public String userName;**  **}** |

### 输入线程资源

|  |
| --- |
| class IntThrad extends Thread {  private Res res;  public IntThrad(Res res) {  this.res = res;  }  @Override  public void run() {  int count = 0;  while (true) {  if (count == 0) {  res.userName = "余胜军";  res.userSex = "男";  } else {  res.userName = "小紅";  res.userSex = "女";  }  count = (count + 1) % 2;  }  }  } |

### 输出线程

|  |
| --- |
| **class OutThread extends Thread {**  **private Res res;**  **public OutThread(Res res) {**  **this.res = res;**  **}**  **@Override**  **public void run() {**  **while (true) {**  **System.*out*.println(res.userName + "--" + res.userSex);**  **}**  **}**  **}** |

### 运行代码

|  |
| --- |
| Res res = **new** Res();  IntThrad intThrad = **new** IntThrad(res);  OutThread outThread = **new** OutThread(res);  intThrad.start();  outThread.start(); |

### 运行代码

|  |
| --- |
| C:\Users\ADMINI~1\AppData\Local\Temp\ksohtml\wps9026.tmp.jpg |

注意：数据发生错乱，造成线程安全问题

### 解决线程安全问题

#### IntThrad 加上synchronized

|  |
| --- |
| **class** IntThrad **extends** Thread {  **private** Res res;  **public** IntThrad(Res res) {  **this**.res = res;  }  @Override  **public** **void** run() {  **int** count = 0;  **while** (**true**) {  **synchronized** (res) {  **if** (count == 0) {  res.userName = "余胜军";  res.userSex = "男";  } **else** {  res.userName = "小紅";  res.userSex = "女";  }  count = (count + 1) % 2;  }  }  }  } |

#### 输出线程加上synchronized

|  |
| --- |
| **class Res {**  **public String userName;**  **public String sex;**  **}**  **class InputThread extends Thread {**  **private Res res;**  **public InputThread(Res res) {**  **this.res = res;**  **}**  **@Override**  **public void run() {**  **int count = 0;**  **while (true) {**  **synchronized (res) {**  **if (count == 0) {**  **res.userName = "余胜军";**  **res.sex = "男";**  **} else {**  **res.userName = "小红";**  **res.sex = "女";**  **}**  **count = (count + 1) % 2;**  **}**  **}**  **}**  **}**  **class OutThrad extends Thread {**  **private Res res;**  **public OutThrad(Res res) {**  **this.res = res;**  **}**  **@Override**  **public void run() {**  **while (true) {**  **synchronized (res) {**  **System.*out*.println(res.userName + "," + res.sex);**  **}**  **}**  **}**  **}**  **public class ThreadDemo01 {**  **public static void main(String[] args) {**  **Res res = new Res();**  **InputThread inputThread = new InputThread(res);**  **OutThrad outThrad = new OutThrad(res);**  **inputThread.start();**  **outThrad.start();**  **}**  **}** |

# wait、notify方法

1.因为涉及到对象锁,他们必须都放在synchronized中来使用. Wait、Notify一定要在synchronized里面进行使用。

2.Wait必须暂定当前正在执行的线程,并释放资源锁,让其他线程可以有机会运行

3. notify/notifyall: 唤醒因锁池中的线程,使之运行

**注意:一定要在线程同步中使用,并且是同一个锁的资源**

|  |
| --- |
| **class Res {**  **public String userSex;**  **public String userName;**  **//线程通讯标识**  **public boolean flag = false;**  **}** |

|  |
| --- |
| class IntThrad extends Thread {  private Res res;  public IntThrad(Res res) {  this.res = res;  }  @Override  public void run() {  int count = 0;  while (true) {  synchronized (res) {  if (res.flag) {  try {  // 当前线程变为等待，但是可以释放锁  res.wait();  } catch (Exception e) {  }  }  if (count == 0) {  res.userName = "余胜军";  res.userSex = "男";  } else {  res.userName = "小紅";  res.userSex = "女";  }  count = (count + 1) % 2;  res.flag = true;  // 唤醒当前线程  res.notify();  }  }  }  } |

|  |
| --- |
| class OutThread extends Thread {  private Res res;  public OutThread(Res res) {  this.res = res;  }  @Override  public void run() {  while (true) {  synchronized (res) {  if (!res.flag) {  try {  res.wait();  } catch (Exception e) {  // TODO: handle exception  }  }  System.*out*.println(res.userName + "--" + res.userSex);  res.flag = false;  res.notify();  }  }  }  } |

|  |
| --- |
| public class ThreaCommun {  public static void main(String[] args) {  Res res = new Res();  IntThrad intThrad = new IntThrad(res);  OutThread outThread = new OutThread(res);  intThrad.start();  outThread.start();  }  } |

# wait与sleep区别

对于sleep()方法，我们首先要知道该方法是属于Thread类中的。而wait()方法，则是属于Object类中的。

sleep()方法导致了程序暂停执行指定的时间，让出cpu该其他线程，但是他的监控状态依然保持者，当指定的时间到了又会自动恢复运行状态。

在调用sleep()方法的过程中，线程不会释放对象锁。

而当调用wait()方法的时候，线程会放弃对象锁，进入等待此对象的等待锁定池，只有针对此对象调用notify()方法后本线程才进入对象锁定池准备

获取对象锁进入运行状态。

# Lock锁

在 jdk1.5 之后，并发包中新增了 Lock 接口(以及相关实现类)用来实现锁功能，Lock 接口提供了与 synchronized 关键字类似的同步功能，但需要在使用时手动获取锁和释放锁。

## Lock写法

|  |
| --- |
| Lock **lock** = **new** ReentrantLock();  **lock**.**lock**();  **try**{  *//可能会出现线程安全的操作*  }**finally**{  *//一定在finally中释放锁*  *//也不能把获取锁在try中进行，因为有可能在获取锁的时候抛出异常*  **lock**.ublock();  } |

## Lock与synchronized 关键字的区别

Lock 接口可以尝试非阻塞地获取锁 当前线程尝试获取锁。如果这一时刻锁没有被其他线程获取到，则成功获取并持有锁。  
Lock 接口能被中断地获取锁 与 synchronized 不同，获取到锁的线程能够响应中断，当获取到的锁的线程被中断时，中断异常将会被抛出，同时锁会被释放。

Lock 接口在指定的截止时间之前获取锁，如果截止时间到了依旧无法获取锁，则返回。

## Condition用法

Condition的功能类似于在传统的线程技术中的,Object.wait()和Object.notify()的功能。

### 代码

|  |
| --- |
| **Condition condition = lock.newCondition();**  **res. condition.await(); 类似wait**  **res. Condition. Signal() 类似notify** |

|  |
| --- |
| class Res {  public String userName;  public String sex;  public boolean flag = false;  Lock lock = new ReentrantLock();  }  class InputThread extends Thread {  private Res res;  Condition newCondition;  public InputThread(Res res, Condition newCondition) {  this.res = res;  this.newCondition=newCondition;  }  @Override  public void run() {  int count = 0;  while (true) {  // synchronized (res) {  try {  res.lock.lock();  if (res.flag) {  try {  // res.wait();  newCondition.await();  } catch (Exception e) {  // TODO: handle exception  }  }  if (count == 0) {  res.userName = "余胜军";  res.sex = "男";  } else {  res.userName = "小红";  res.sex = "女";  }  count = (count + 1) % 2;  res.flag = true;  // res.notify();  newCondition.signal();  } catch (Exception e) {  // TODO: handle exception  }finally {  res.lock.unlock();  }  }  // }  }  }  class OutThrad extends Thread {  private Res res;  private Condition newCondition;  public OutThrad(Res res,Condition newCondition) {  this.res = res;  this.newCondition=newCondition;  }  @Override  public void run() {  while (true) {  // synchronized (res) {  try {  res.lock.lock();  if (!res.flag) {  try {  // res.wait();  newCondition.await();  } catch (Exception e) {  // TODO: handle exception  }  }  System.*out*.println(res.userName + "," + res.sex);  res.flag = false;  // res.notify();  newCondition.signal();  } catch (Exception e) {  // TODO: handle exception  }finally {  res.lock.unlock();  }  // }  }  }  }  public class ThreadDemo01 {  public static void main(String[] args) {  Res res = new Res();  Condition newCondition = res.lock.newCondition();  InputThread inputThread = new InputThread(res,newCondition);  OutThrad outThrad = new OutThrad(res,newCondition);  inputThread.start();  outThrad.start();  }  } |