**[java多线程总结](http://www.cnblogs.com/rollenholt/archive/2011/08/28/2156357.html)**

以前没有写笔记的习惯，现在慢慢的发现及时总结是多么的重要了，呵呵。虽然才大二，但是也快要毕业了，要加油了。

这一篇文章主要关于java多线程，主要还是以例子来驱动的。因为讲解多线程的书籍和文章已经很多了，所以我也不好意思多说，呵呵、大家可以去参考一些那些书籍。我这个文章主要关于实际的一些问题。同时也算是我以后复习的资料吧，。呵呵大家多多指教。

同时希望多结交一些技术上的朋友。谢谢。

----------------------------------------------------------------------------------------------------------------------------------------------------

**java中的多线程**

在java中要想实现多线程，有两种手段，一种是继续Thread类，另外一种是实现Runable接口。

对于直接继承Thread的类来说，代码大致框架是：

[?](http://www.cnblogs.com/rollenholt/archive/2011/08/28/2156357.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | class 类名 extends Thread{  方法1;  方法2；  …  public void run(){  // other code…  }  属性1；  属性2；  …    } |

先看一个简单的例子：

[?](http://www.cnblogs.com/rollenholt/archive/2011/08/28/2156357.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28 | /\*\*   \* @author Rollen-Holt 继承Thread类,直接调用run方法   \* \*/  class hello extends Thread {        public hello() {        }        public hello(String name) {          this.name = name;      }        public void run() {          for (int i = 0; i < 5; i++) {              System.out.println(name + "运行     " + i);          }      }        public static void main(String[] args) {          hello h1=new hello("A");          hello h2=new hello("B");          h1.run();          h2.run();      }        private String name;  } |

【运行结果】：

A运行     0

A运行     1

A运行     2

A运行     3

A运行     4

B运行     0

B运行     1

B运行     2

B运行     3

B运行     4

我们会发现这些都是顺序执行的，说明我们的调用方法不对，应该调用的是start（）方法。

当我们把上面的主函数修改为如下所示的时候：

[?](http://www.cnblogs.com/rollenholt/archive/2011/08/28/2156357.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | public static void main(String[] args) {          hello h1=new hello("A");          hello h2=new hello("B");          h1.start();          h2.start();      } |

然后运行程序，输出的可能的结果如下：

A运行     0

B运行     0

B运行     1

B运行     2

B运行     3

B运行     4

A运行     1

A运行     2

A运行     3

A运行     4

因为需要用到CPU的资源，所以每次的运行结果基本是都不一样的，呵呵。

注意：虽然我们在这里调用的是start（）方法，但是实际上调用的还是run（）方法的主体。

**那么：为什么我们不能直接调用run（）方法呢？**

**我的理解是：线程的运行需要本地操作系统的支持。**

**如果你查看start的源代码的时候，会发现：**

[?](http://www.cnblogs.com/rollenholt/archive/2011/08/28/2156357.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | public synchronized void start() {          /\*\*       \* This method is not invoked for the main method thread or "system"       \* group threads created/set up by the VM. Any new functionality added       \* to this method in the future may have to also be added to the VM.       \*       \* A zero status value corresponds to state "NEW".           \*/          if (threadStatus != 0 || this != me)              throw new IllegalThreadStateException();          group.add(this);          start0();          if (stopBeforeStart) {          stop0(throwableFromStop);      }  }  private native void start0(); |

**注意我用红色加粗的那一条语句，说明此处调用的是start0（）。并且这个这个方法用了native关键字，次关键字表示调用本地操作系统的函数。因为多线程的实现需要本地操作系统的支持。**

但是start方法重复调用的话，会出现java.lang.IllegalThreadStateException异常。

**通过实现Runnable接口：**

大致框架是：

[?](http://www.cnblogs.com/rollenholt/archive/2011/08/28/2156357.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | class 类名 implements Runnable{  方法1;  方法2；  …  public void run(){  // other code…  }  属性1；  属性2；  …    } |

来先看一个小例子吧：

[?](http://www.cnblogs.com/rollenholt/archive/2011/08/28/2156357.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30 | /\*\*   \* @author Rollen-Holt 实现Runnable接口   \* \*/  class hello implements Runnable {        public hello() {        }        public hello(String name) {          this.name = name;      }        public void run() {          for (int i = 0; i < 5; i++) {              System.out.println(name + "运行     " + i);          }      }        public static void main(String[] args) {          hello h1=new hello("线程A");          Thread demo= new Thread(h1);          hello h2=new hello("线程Ｂ");          Thread demo1=new Thread(h2);          demo.start();          demo1.start();      }        private String name;  } |

【可能的运行结果】：

线程A运行     0

线程Ｂ运行     0

线程Ｂ运行     1

线程Ｂ运行     2

线程Ｂ运行     3

线程Ｂ运行     4

线程A运行     1

线程A运行     2

线程A运行     3

线程A运行     4

**关于选择继承Thread还是实现Runnable接口？**

其实Thread也是实现Runnable接口的**：**

[?](http://www.cnblogs.com/rollenholt/archive/2011/08/28/2156357.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | class Thread implements Runnable {      //…  public void run() {          if (target != null) {               target.run();          }          }  } |

其实Thread中的run方法调用的是Runnable接口的run方法。不知道大家发现没有，Thread和Runnable都实现了run方法，这种操作模式其实就是代理模式。关于代理模式，我曾经写过一个小例子呵呵，大家有兴趣的话可以看一下：<http://www.cnblogs.com/rollenholt/archive/2011/08/18/2144847.html>

**Thread和Runnable的区别：**

**如果一个类继承Thread，则不适合资源共享。但是如果实现了Runable接口的话，则很容易的实现资源共享。**

[?](http://www.cnblogs.com/rollenholt/archive/2011/08/28/2156357.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | /\*\*   \* @author Rollen-Holt 继承Thread类，不能资源共享   \* \*/  class hello extends Thread {      public void run() {          for (int i = 0; i < 7; i++) {              if (count > 0) {                  System.out.println("count= " + count--);              }          }      }        public static void main(String[] args) {          hello h1 = new hello();          hello h2 = new hello();          hello h3 = new hello();          h1.start();          h2.start();          h3.start();      }        private int count = 5;  } |

【运行结果】：

count= 5

count= 4

count= 3

count= 2

count= 1

count= 5

count= 4

count= 3

count= 2

count= 1

count= 5

count= 4

count= 3

count= 2

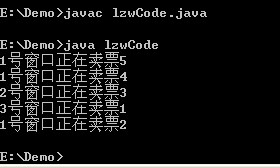
count= 1

**大家可以想象，如果这个是一个买票系统的话，如果count表示的是车票的数量的话，说明并没有实现资源的共享。**

**我们换为Runnable接口**

[?](http://www.cnblogs.com/rollenholt/archive/2011/08/28/2156357.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | class MyThread implements Runnable{        private int ticket = 5;  //5张票        public void run() {          for (int i=0; i<=20; i++) {              if (this.ticket > 0) {                  System.out.println(Thread.currentThread().getName()+ "正在卖票"+this.ticket--);              }          }      }  }  public class lzwCode {        public static void main(String [] args) {          MyThread my = new MyThread();          new Thread(my, "1号窗口").start();          new Thread(my, "2号窗口").start();          new Thread(my, "3号窗口").start();      }  } |



【运行结果】：

count= 5

count= 4

count= 3

count= 2

count= 1

**总结一下吧：**

**实现Runnable接口比继承Thread类所具有的优势：**

**1）：适合多个相同的程序代码的线程去处理同一个资源**

**2）：可以避免java中的单继承的限制**

**3）：增加程序的健壮性，代码可以被多个线程共享，代码和数据独立。**

**所以，本人建议大家劲量实现接口。**

[?](http://www.cnblogs.com/rollenholt/archive/2011/08/28/2156357.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | /\*\*   \* @author Rollen-Holt   \* 取得线程的名称   \* \*/  class hello implements Runnable {      public void run() {          for (int i = 0; i < 3; i++) {              System.out.println(Thread.currentThread().getName());          }      }        public static void main(String[] args) {          hello he = new hello();          new Thread(he,"A").start();          new Thread(he,"B").start();          new Thread(he).start();      }  } |

【运行结果】：

A

A

A

B

B

B

Thread-0

Thread-0

Thread-0

说明如果我们没有指定名字的话，系统自动提供名字。

**提醒一下大家：main方法其实也是一个线程。在java中所以的线程都是同时启动的，至于什么时候，哪个先执行，完全看谁先得到CPU的资源。**

**在java中，每次程序运行至少启动2个线程。一个是main线程，一个是垃圾收集线程。因为每当使用java命令执行一个类的时候，实际上都会启动一个ＪＶＭ，每一个ｊＶＭ实习在就是在操作系统中启动了一个进程。**

判断线程是否启动

[?](http://www.cnblogs.com/rollenholt/archive/2011/08/28/2156357.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | /\*\*   \* @author Rollen-Holt 判断线程是否启动   \* \*/  class hello implements Runnable {      public void run() {          for (int i = 0; i < 3; i++) {              System.out.println(Thread.currentThread().getName());          }      }        public static void main(String[] args) {          hello he = new hello();          Thread demo = new Thread(he);          System.out.println("线程启动之前---》" + demo.isAlive());          demo.start();          System.out.println("线程启动之后---》" + demo.isAlive());      }  } |

【运行结果】

线程启动之前---》false

线程启动之后---》true

Thread-0

Thread-0

Thread-0

**主线程也有可能在子线程结束之前结束。并且子线程不受影响，不会因为主线程的结束而结束。**

线程的强制执行：

[?](http://www.cnblogs.com/rollenholt/archive/2011/08/28/2156357.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26 | /\*\*       \* @author Rollen-Holt 线程的强制执行       \* \*/      class hello implements Runnable {          public void run() {              for (int i = 0; i < 3; i++) {                  System.out.println(Thread.currentThread().getName());              }          }            public static void main(String[] args) {              hello he = new hello();              Thread demo = new Thread(he,"线程");              demo.start();              for(int i=0;i<50;++i){                  if(i>10){                      try{                          demo.join();  //强制执行demo                      }catch (Exception e) {                          e.printStackTrace();                      }                  }                  System.out.println("main 线程执行-->"+i);              }          }      } |

【运行的结果】：

main 线程执行-->0

main 线程执行-->1

main 线程执行-->2

main 线程执行-->3

main 线程执行-->4

main 线程执行-->5

main 线程执行-->6

main 线程执行-->7

main 线程执行-->8

main 线程执行-->9

main 线程执行-->10

线程

线程

线程

main 线程执行-->11

main 线程执行-->12

main 线程执行-->13

．．．

线程的休眠：

[?](http://www.cnblogs.com/rollenholt/archive/2011/08/28/2156357.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | /\*\*   \* @author Rollen-Holt 线程的休眠   \* \*/  class hello implements Runnable {      public void run() {          for (int i = 0; i < 3; i++) {              try {                  Thread.sleep(2000);              } catch (Exception e) {                  e.printStackTrace();              }              System.out.println(Thread.currentThread().getName() + i);          }      }        public static void main(String[] args) {          hello he = new hello();          Thread demo = new Thread(he, "线程");          demo.start();      }  } |

【运行结果】：（结果每隔2s输出一个）

线程0

线程1

线程2

线程的中断：

[?](http://www.cnblogs.com/rollenholt/archive/2011/08/28/2156357.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28 | /\*\*   \* @author Rollen-Holt 线程的中断   \* \*/  class hello implements Runnable {      public void run() {          System.out.println("执行run方法");          try {              Thread.sleep(10000);              System.out.println("线程完成休眠");          } catch (Exception e) {              System.out.println("休眠被打断");              return;  //返回到程序的调用处          }          System.out.println("线程正常终止");      }        public static void main(String[] args) {          hello he = new hello();          Thread demo = new Thread(he, "线程");          demo.start();          try{              Thread.sleep(2000);          }catch (Exception e) {              e.printStackTrace();          }          demo.interrupt(); //2s后中断线程      }  } |

【运行结果】：

执行run方法

休眠被打断

**在java程序中，只要前台有一个线程在运行，整个java程序进程不会小时，所以此时可以设置一个后台线程，这样即使java进程小时了，此后台线程依然能够继续运行。**

[?](http://www.cnblogs.com/rollenholt/archive/2011/08/28/2156357.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | /\*\*   \* @author Rollen-Holt 后台线程   \* \*/  class hello implements Runnable {      public void run() {          while (true) {              System.out.println(Thread.currentThread().getName() + "在运行");          }      }        public static void main(String[] args) {          hello he = new hello();          Thread demo = new Thread(he, "线程");          demo.setDaemon(true);          demo.start();      }  } |

虽然有一个死循环，但是程序还是可以执行完的。因为在死循环中的线程操作已经设置为后台运行了。

线程的优先级：

[?](http://www.cnblogs.com/rollenholt/archive/2011/08/28/2156357.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | /\*\*   \* @author Rollen-Holt 线程的优先级   \* \*/  class hello implements Runnable {      public void run() {          for(int i=0;i<5;++i){              System.out.println(Thread.currentThread().getName()+"运行"+i);          }      }        public static void main(String[] args) {          Thread h1=new Thread(new hello(),"A");          Thread h2=new Thread(new hello(),"B");          Thread h3=new Thread(new hello(),"C");          h1.setPriority(8);          h2.setPriority(2);          h3.setPriority(6);          h1.start();          h2.start();          h3.start();        }  } |

【运行结果】：

A运行0

A运行1

A运行2

A运行3

A运行4

B运行0

C运行0

C运行1

C运行2

C运行3

C运行4

B运行1

B运行2

B运行3

B运行4

。但是请读者不要误以为优先级越高就先执行。谁先执行还是取决于谁先去的CPU的资源、

另外，主线程的优先级是5.

线程的礼让。

在线程操作中，也可以使用yield（）方法，将一个线程的操作暂时交给其他线程执行。

[?](http://www.cnblogs.com/rollenholt/archive/2011/08/28/2156357.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | /\*\*   \* @author Rollen-Holt 线程的优先级   \* \*/  class hello implements Runnable {      public void run() {          for(int i=0;i<5;++i){              System.out.println(Thread.currentThread().getName()+"运行"+i);              if(i==3){                  System.out.println("线程的礼让");                  Thread.currentThread().yield();              }          }      }        public static void main(String[] args) {          Thread h1=new Thread(new hello(),"A");          Thread h2=new Thread(new hello(),"B");          h1.start();          h2.start();        }  } |

A运行0

A运行1

A运行2

A运行3

线程的礼让

A运行4

B运行0

B运行1

B运行2

B运行3

线程的礼让

B运行4

**同步和死锁：**

**【问题引出】:**比如说对于买票系统，有下面的代码：

[?](http://www.cnblogs.com/rollenholt/archive/2011/08/28/2156357.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28 | /\*\*   \* @author Rollen-Holt   \* \*/  class hello implements Runnable {      public void run() {          for(int i=0;i<10;++i){              if(count>0){                  try{                      Thread.sleep(1000);                  }catch(InterruptedException e){                      e.printStackTrace();                  }                  System.out.println(count--);              }          }      }        public static void main(String[] args) {          hello he=new hello();          Thread h1=new Thread(he);          Thread h2=new Thread(he);          Thread h3=new Thread(he);          h1.start();          h2.start();          h3.start();      }      private int count=5;  } |

【运行结果】：

5

4

3

2

1

0

-1

这里出现了-1，显然这个是错的。，应该票数不能为负值。

如果想解决这种问题，就需要使用同步。所谓同步就是在统一时间段中只有有一个线程运行，

其他的线程必须等到这个线程结束之后才能继续执行。

**【使用线程同步解决问题】**

采用同步的话，可以使用同步代码块和同步方法两种来完成。

【同步代码块】：

语法格式：

synchronized（同步对象）{

 //需要同步的代码

}

但是一般都把当前对象this作为同步对象。

比如对于上面的买票的问题，如下：

[?](http://www.cnblogs.com/rollenholt/archive/2011/08/28/2156357.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30 | /\*\*   \* @author Rollen-Holt   \* \*/  class hello implements Runnable {      public void run() {          for(int i=0;i<10;++i){              synchronized (this) {                  if(count>0){                      try{                          Thread.sleep(1000);                      }catch(InterruptedException e){                          e.printStackTrace();                      }                      System.out.println(count--);                  }              }          }      }        public static void main(String[] args) {          hello he=new hello();          Thread h1=new Thread(he);          Thread h2=new Thread(he);          Thread h3=new Thread(he);          h1.start();          h2.start();          h3.start();      }      private int count=5;  } |

【运行结果】：（每一秒输出一个结果）

5

4

3

2

1

【同步方法】

也可以采用同步方法。

语法格式为**synchronized 方法返回类型方法名（参数列表）{**

**// 其他代码**

**}**

**现在，我们采用同步方法解决上面的问题。**

[?](http://www.cnblogs.com/rollenholt/archive/2011/08/28/2156357.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33 | /\*\*   \* @author Rollen-Holt   \* \*/  class hello implements Runnable {      public void run() {          for (int i = 0; i < 10; ++i) {              sale();          }      }        public synchronized void sale() {          if (count > 0) {              try {                  Thread.sleep(1000);              } catch (InterruptedException e) {                  e.printStackTrace();              }              System.out.println(count--);          }      }        public static void main(String[] args) {          hello he = new hello();          Thread h1 = new Thread(he);          Thread h2 = new Thread(he);          Thread h3 = new Thread(he);          h1.start();          h2.start();          h3.start();      }        private int count = 5;  } |

【运行结果】（每秒输出一个）

5

4

3

2

1

提醒一下，当多个线程共享一个资源的时候需要进行同步，但是过多的同步可能导致死锁。

此处列举经典的生产者和消费者问题。

【生产者和消费者问题】

先看一段有问题的代码。

[?](http://www.cnblogs.com/rollenholt/archive/2011/08/28/2156357.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89 | class Info {        public String getName() {          return name;      }        public void setName(String name) {          this.name = name;      }        public int getAge() {          return age;      }        public void setAge(int age) {          this.age = age;      }        private String name = "Rollen";      private int age = 20;  }    /\*\*   \* 生产者   \* \*/  class Producer implements Runnable{      private Info info=null;      Producer(Info info){          this.info=info;      }        public void run(){          boolean flag=false;          for(int i=0;i<25;++i){              if(flag){                  this.info.setName("Rollen");                  try{                      Thread.sleep(100);                  }catch (Exception e) {                      e.printStackTrace();                  }                  this.info.setAge(20);                  flag=false;              }else{                  this.info.setName("chunGe");                  try{                      Thread.sleep(100);                  }catch (Exception e) {                      e.printStackTrace();                  }                  this.info.setAge(100);                  flag=true;              }          }      }  }  /\*\*   \* 消费者类   \* \*/  class Consumer implements Runnable{      private Info info=null;      public Consumer(Info info){          this.info=info;      }        public void run(){          for(int i=0;i<25;++i){              try{                  Thread.sleep(100);              }catch (Exception e) {                  e.printStackTrace();              }              System.out.println(this.info.getName()+"<---->"+this.info.getAge());          }      }  }    /\*\*   \* 测试类   \* \*/  class hello{      public static void main(String[] args) {          Info info=new Info();          Producer pro=new Producer(info);          Consumer con=new Consumer(info);          new Thread(pro).start();          new Thread(con).start();      }  } |

【运行结果】：

Rollen<---->100

chunGe<---->20

chunGe<---->100

Rollen<---->100

chunGe<---->20

Rollen<---->100

Rollen<---->100

Rollen<---->100

chunGe<---->20

chunGe<---->20

chunGe<---->20

Rollen<---->100

chunGe<---->20

Rollen<---->100

chunGe<---->20

Rollen<---->100

chunGe<---->20

Rollen<---->100

chunGe<---->20

Rollen<---->100

chunGe<---->20

Rollen<---->100

chunGe<---->20

Rollen<---->100

chunGe<---->20

大家可以从结果中看到，名字和年龄并没有对于。

**那么如何解决呢？**

**1）加入同步**

**2）加入等待和唤醒**

先来看看加入同步会是如何。

[?](http://www.cnblogs.com/rollenholt/archive/2011/08/28/2156357.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99 | class Info {        public String getName() {          return name;      }        public void setName(String name) {          this.name = name;      }        public int getAge() {          return age;      }        public void setAge(int age) {          this.age = age;      }        public synchronized void set(String name, int age){          this.name=name;          try{              Thread.sleep(100);          }catch (Exception e) {              e.printStackTrace();          }          this.age=age;      }        public synchronized void get(){          try{              Thread.sleep(100);          }catch (Exception e) {              e.printStackTrace();          }          System.out.println(this.getName()+"<===>"+this.getAge());      }      private String name = "Rollen";      private int age = 20;  }    /\*\*   \* 生产者   \* \*/  class Producer implements Runnable {      private Info info = null;        Producer(Info info) {          this.info = info;      }        public void run() {          boolean flag = false;          for (int i = 0; i < 25; ++i) {              if (flag) {                    this.info.set("Rollen", 20);                  flag = false;              } else {                  this.info.set("ChunGe", 100);                  flag = true;              }          }      }  }    /\*\*   \* 消费者类   \* \*/  class Consumer implements Runnable {      private Info info = null;        public Consumer(Info info) {          this.info = info;      }        public void run() {          for (int i = 0; i < 25; ++i) {              try {                  Thread.sleep(100);              } catch (Exception e) {                  e.printStackTrace();              }              this.info.get();          }      }  }    /\*\*   \* 测试类   \* \*/  class hello {      public static void main(String[] args) {          Info info = new Info();          Producer pro = new Producer(info);          Consumer con = new Consumer(info);          new Thread(pro).start();          new Thread(con).start();      }  } |

【运行结果】：

Rollen<===>20

ChunGe<===>100

ChunGe<===>100

ChunGe<===>100

ChunGe<===>100

ChunGe<===>100

Rollen<===>20

ChunGe<===>100

ChunGe<===>100

ChunGe<===>100

ChunGe<===>100

ChunGe<===>100

ChunGe<===>100

ChunGe<===>100

ChunGe<===>100

ChunGe<===>100

ChunGe<===>100

ChunGe<===>100

ChunGe<===>100

ChunGe<===>100

ChunGe<===>100

ChunGe<===>100

ChunGe<===>100

ChunGe<===>100

ChunGe<===>100

从运行结果来看，错乱的问题解决了，现在是Rollen 对应20，ChunGe对于100

，但是还是出现了重复读取的问题，也肯定有重复覆盖的问题。如果想解决这个问题，就需要使用Object类帮忙了、

，我们可以使用其中的等待和唤醒操作。

要完成上面的功能，我们只需要修改Info类饥渴，在其中加上标志位，并且通过判断标志位完成等待和唤醒的操作，代码如下：

[?](http://www.cnblogs.com/rollenholt/archive/2011/08/28/2156357.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119 | class Info {        public String getName() {          return name;      }        public void setName(String name) {          this.name = name;      }        public int getAge() {          return age;      }        public void setAge(int age) {          this.age = age;      }        public synchronized void set(String name, int age){          if(!flag){              try{                  super.wait();              }catch (Exception e) {                  e.printStackTrace();              }          }          this.name=name;          try{              Thread.sleep(100);          }catch (Exception e) {              e.printStackTrace();          }          this.age=age;          flag=false;          super.notify();      }        public synchronized void get(){          if(flag){              try{                  super.wait();              }catch (Exception e) {                  e.printStackTrace();              }          }            try{              Thread.sleep(100);          }catch (Exception e) {              e.printStackTrace();          }          System.out.println(this.getName()+"<===>"+this.getAge());          flag=true;          super.notify();      }      private String name = "Rollen";      private int age = 20;      private boolean flag=false;  }    /\*\*   \* 生产者   \* \*/  class Producer implements Runnable {      private Info info = null;        Producer(Info info) {          this.info = info;      }        public void run() {          boolean flag = false;          for (int i = 0; i < 25; ++i) {              if (flag) {                    this.info.set("Rollen", 20);                  flag = false;              } else {                  this.info.set("ChunGe", 100);                  flag = true;              }          }      }  }    /\*\*   \* 消费者类   \* \*/  class Consumer implements Runnable {      private Info info = null;        public Consumer(Info info) {          this.info = info;      }        public void run() {          for (int i = 0; i < 25; ++i) {              try {                  Thread.sleep(100);              } catch (Exception e) {                  e.printStackTrace();              }              this.info.get();          }      }  }    /\*\*   \* 测试类   \* \*/  class hello {      public static void main(String[] args) {          Info info = new Info();          Producer pro = new Producer(info);          Consumer con = new Consumer(info);          new Thread(pro).start();          new Thread(con).start();      }  } |

[?](http://www.cnblogs.com/rollenholt/archive/2011/08/28/2156357.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27 | 【程序运行结果】：  Rollen<===>20  ChunGe<===>100  Rollen<===>20  ChunGe<===>100  Rollen<===>20  ChunGe<===>100  Rollen<===>20  ChunGe<===>100  Rollen<===>20  ChunGe<===>100  Rollen<===>20  ChunGe<===>100  Rollen<===>20  ChunGe<===>100  Rollen<===>20  ChunGe<===>100  Rollen<===>20  ChunGe<===>100  Rollen<===>20  ChunGe<===>100  Rollen<===>20  ChunGe<===>100  Rollen<===>20  ChunGe<===>100  Rollen<===>20  先在看结果就可以知道，之前的问题完全解决。 |

《完》