**1.2.3 实例变量与线程安全**

**1.2.3.1 不共享数据**

(1)

|  |
| --- |
| package com.thread.www;  public class MyThread3 extends Thread{  private int count = 5;  public MyThread3(String name) {  super();  this.setName(name);//设置线程名称  }  @Override  public void run() {  super.run();  while(count > 0) {  count--;  System.out.println("由" + this.currentThread().getName() + "计算，count=" +  count );  }  }  } |

(2)

|  |
| --- |
| package test;  import com.thread.www.MyThread3;  public class RunMyThread3 {  public static void main(String[] args) {  MyThread3 a = new MyThread3("A");  MyThread3 b = new MyThread3("B");  MyThread3 c = new MyThread3("C");  a.start();  b.start();  c.start();  }  } |

(3)结果:一共创建了3线程，每个线程都有各自的count变量，自己减少自己的count变量。变量不共享

|  |
| --- |
| 由A计算，count=3  由A计算，count=2  由A计算，count=1  由A计算，count=0  由C计算，count=4  由B计算，count=4  由B计算，count=3  由B计算，count=2  由B计算，count=1  由B计算，count=0  由C计算，count=3  由C计算，count=2  由C计算，count=1  由C计算，count=0 |

**1.2.3.2 共享数据**

1.2.3.2.1 例1:出现非线程安全问题

(1) MyThread4.java

|  |
| --- |
| package com.thread.www;  public class MyThread4 extends Thread {  private int count = 5;  @Override  public void run() {  super.run();  count--;  //此示例不要用for语句，因为使用同步后，其它线程就得不到线程运行的机会了，  //会一直由一个线程做减法运算  System.out.println("由" + this.currentThread().getName() + "计算count=" + count);  }  } |

(2) TestMyThread4.java

|  |
| --- |
| package test;  import com.thread.www.MyThread4;  public class TestMyThread4 {  public static void main(String[] args) {  MyThread4 myThread4 = new MyThread4();  Thread a = new Thread(myThread4, "a");  Thread b = new Thread(myThread4, "b");  Thread c = new Thread(myThread4, "c");  Thread d = new Thread(myThread4, "d");  Thread e = new Thread(myThread4, "e");  a.start();  b.start();  c.start();  d.start();  e.start();  }  } |

(3)结果: c和b线程同时对count进行处理，产生”非线程安全” 问题

|  |
| --- |
| 由a计算count=4  由c计算count=2  由b计算count=2  由e计算count=1  由d计算count=0 |

1.2.3.2.2 例2:

需要使多个线程间同步，即按顺序排队的方式进行减1操作

(1) 在例1基础上，更改 MyThread4.java代码如下：

|  |
| --- |
| package com.thread.www;  public class MyThread4 extends Thread {  private int count = 5;  @Override  synchronized public void run() {  super.run();  count--;  //此示例不要用for语句，因为使用同步后，其它线程就得不到线程运行的机会了，  //会一直由一个线程做减法运算  System.out.println("由" + this.currentThread().getName() + "计算count=" + count);  }  } |

(2)结果:

|  |
| --- |
| 由a计算count=4  由b计算count=3  由d计算count=2  由c计算count=1  由e计算count=0 |

synchronized使多个线程在执行run方法时，以排队方式处理。当一个线程调用run前，先判断run方法有没被上锁，若上锁，说明有其它线程在调用run方法，必须等其它线程对run方法调用结束后才可执行run方法。

Synchronized可在任意对象及方法前加锁，被加锁的代码称为”互斥区”或“临界区”。

如果不能拿到锁，该线程会不断的去尝试拿这把锁，直到能够拿到为止，且多个线程会同时争抢这把锁。

非线程安全:多个线程对同一个对象中的同一个实例变量进行操作时，会出现值更改、值不同步的情况，进而影响程序执行流程。

**1.2.4 留意i—与System.out.println()的异常**

1. MyThread5.java

|  |
| --- |
| package com.thread.www;  public class MyThread5 extends Thread {  private int i = 5;  @Override  public void run() {  System.out.println("i=" + (i--) + "threadname=" + Thread.currentThread().getName());  }  } |

2. TestMyThread5.java

|  |
| --- |
| package test;  import com.thread.www.MyThread5;  public class TestMyThread5 {  public static void main(String[] args) {  MyThread5 myThread5 = new MyThread5();  Thread a = new Thread(myThread5);  Thread b = new Thread(myThread5);  Thread c = new Thread(myThread5);  Thread d = new Thread(myThread5);  Thread e = new Thread(myThread5);  a.start();  b.start();  c.start();  d.start();  e.start();  }  } |

3.结果

|  |
| --- |
| i=5threadname=Thread-1  i=3threadname=Thread-4  i=4threadname=Thread-2  i=2threadname=Thread-3  i=1threadname=Thread-5 |

虽然println()方法在内部是同步的，但i--操作是在println之前发生的，所以有非线程同步的问题的概率。

|  |
| --- |
| public void println(String x) {  synchronized (this) {  print(x);  newLine();  }  } |

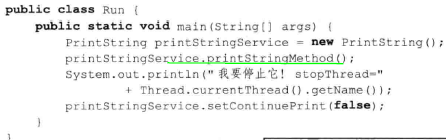
**1.3 currentThread()**

**2.3 volatile关键字**

作用:使变量在多个线程间可见。

**2.3.1死循环**

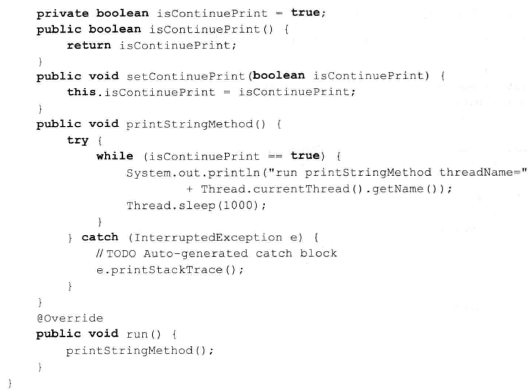




程序陷入死循环，原因:main线程一直在处理while()循环,程序不能执行后续代码。

**2.3.2解决同步死循环(但会出现异步死循环)**







运行在-server服务器模式中64bit的JVM上时，会出现死循环.

**2.3.3 解决异步死循环**

**3.3 ThreadLocal**

<http://blog.csdn.net/lufeng20/article/details/24314381>