[**java多线程总结二**](http://www.cnblogs.com/rollenholt/archive/2011/09/15/2178030.html)

之前也总结了一篇文章《java多线程总结》，地址：<http://www.cnblogs.com/rollenholt/archive/2011/08/28/2156357.html>

这个就叫做第二篇吧，呵呵。

线程一般有6个状态：

新建状态：NEW

可运行状态：RUNNABLE

休眠状态：TIMED\_WAITING

等待状态：WAITING

阻塞状态:BLOCKED

终止状态“TERMINATED

当我们使用new创建线程之后，线程处于新建状态，当调用start方法之后，线程出于可运行状态，当线程需要获得对象的内置锁，而这个锁被其他线程所占用的时候，线程就出于阻塞状态，当线程等待其他线程通知调度表可以运行时，线程处于等待状态，当一个含有时间参数的方法，必须sleep（）方法，可以让线程处于计时等待状态，当run（）方法运行完毕或者出现异常，线程处于终止状态。

[?](http://www.cnblogs.com/rollenholt/archive/2011/09/15/2178030.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 | package Thread;    public class ThreadStateDemo{      public static void main(String[] args) throws Exception{          ThreadState state = new ThreadState();          Thread demo = new Thread(state);          System.out.println("新建状态：" + demo.getState());          demo.start();          System.out.println("可运行状态：" + demo.getState());          Thread.sleep(100);          System.out.println("休眠状态：" + demo.getState());          Thread.sleep(1000);          System.out.println("等待状态：" + demo.getState());          state.notifyWait();          System.out.println("阻塞状态:" + demo.getState());          Thread.sleep(1000);          System.out.println("终止状态“" + demo.getState());      }  }    class ThreadState implements Runnable{        @Override      public void run(){          try{              waitForASecond();              waitForAYear();          }catch(Exception e){              e.printStackTrace();          }        }        // 当前线程等待1秒      public synchronized void waitForASecond() throws Exception{          wait(1000);      }        // 当前线程一直等待      public synchronized void waitForAYear() throws Exception{          wait();      }        // 唤醒线程      public synchronized void notifyWait() throws Exception{          notify();      }  } |

【运行结果】：

新建状态：NEW

可运行状态：RUNNABLE

休眠状态：TIMED\_WAITING

等待状态：WAITING

阻塞状态:BLOCKED

终止状态“TERMINATED

线程组表示一个线程线程的集合，线程组中也可以包含其他的线程组。线程组构成一棵树。

[?](http://www.cnblogs.com/rollenholt/archive/2011/09/15/2178030.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 | package Thread;    import java.util.ArrayList;  import java.util.List;    public class ThreadGroupDemo{      public static void main(String[] args){          for(String str : getThreadGroups(GetRootThreadGroups())){              System.out.println(str);          }      }        // 获得根线程组      private static ThreadGroup GetRootThreadGroups(){          // 获得当前的线程组          ThreadGroup rootGroup = Thread.currentThread().getThreadGroup();          while(true){              if(rootGroup.getParent() != null){                  rootGroup = rootGroup.getParent();              }else{                  break;              }          }          return rootGroup;      }        // 获得给定线程组中所有线程名      public static List<String> getThreads(ThreadGroup group){          List<String> threadList = new ArrayList<String>();          Thread[] threads = new Thread[group.activeCount()];          int count = group.enumerate(threads, false);          for(int i = 0; i < count; i++){              threadList.add(group.getName() + " 线程组 " + threads[i].getName());          }          return threadList;      }        // 获得线程组中所有子线程组      public static List<String> getThreadGroups(ThreadGroup group){          List<String> threadList = getThreads(group);          ThreadGroup[] groups = new ThreadGroup[group.activeGroupCount()];          int count = group.enumerate(groups, false);          for(int i = 0; i < count; i++){              threadList.addAll(getThreads(groups[i]));          }          return threadList;      }    } |

【运行结果】：

system 线程组 Reference Handler

system 线程组 Finalizer

system 线程组 Signal Dispatcher

system 线程组 Attach Listener

main 线程组 main

**使用守护线程**

java中的线程分为2类，用户线程和守护线程，守护线程主要为其他线程提供服务，守护线程会随时被中断，所以一般不要再守护线程中使用需要释放资源的资源，比如输入输出流等，守护线程一般都是后台线程，如果虚拟机只剩下守护线程，虚拟机就会退出。

[?](http://www.cnblogs.com/rollenholt/archive/2011/09/15/2178030.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 | package Thread;    public class DaemonThreadTest{      public static void main(String[] args){          Thread worker = new Thread(new Worker());          Thread timer = new Thread(new Timer());          //设置守护线程          timer.setDaemon(true);          worker.start();          timer.start();      }  }    class Worker implements Runnable{      @Override      public void run(){          for(int i = 0; i < 5; ++i){              System.out.println("rollen真帅! 第" + i + "次");          }      }  }    class Timer implements Runnable{      @Override      public void run(){          long currentTime = System.currentTimeMillis();          long processTime = 0;          while(true){              if((System.currentTimeMillis() - currentTime) > processTime){                  processTime = System.currentTimeMillis() - currentTime;                  System.out.println("程序运行时间：" + processTime);              }          }      }  } |

rollen真帅! 第0次

程序运行时间：1

程序运行时间：2

程序运行时间：3

程序运行时间：4

程序运行时间：5

rollen真帅! 第1次

rollen真帅! 第2次

rollen真帅! 第3次

rollen真帅! 第4次

程序运行时间：6

**终止指定的线程**

虽然在Thread类中提供了stop（）方法可以终止线程，但是由于其固有的不安全性，所以一般不要采用，本例子只是起到抛砖引玉的作用。

[?](http://www.cnblogs.com/rollenholt/archive/2011/09/15/2178030.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 | package Thread;    import java.awt.FlowLayout;  import java.awt.event.ActionEvent;  import java.awt.event.ActionListener;    import javax.swing.JButton;  import javax.swing.JFrame;  import javax.swing.JLabel;  import javax.swing.JPanel;    public class ThreadStopDemo extends JFrame{      public ThreadStopDemo(){          panel.setLayout(new FlowLayout(FlowLayout.CENTER));          panel.add(label);          panel.add(startButton);          panel.add(endButton);          setContentPane(panel);          setSize(200, 300);          setVisible(true);          setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);            startButton.addActionListener(new ActionListener(){              @Override              public void actionPerformed(ActionEvent e){                  counter = new CounterThread();                  new Thread(counter).start();              }          });          endButton.addActionListener(new ActionListener(){              @Override              public void actionPerformed(ActionEvent e){                  if(counter == null){                      return;                  }                  counter.setStop(false);              }          });      }        public static void main(String[] args){          new ThreadStopDemo();      }        class CounterThread implements Runnable{          @Override          public void run(){              while(this.flag){                  try{                      Thread.sleep(500);                  }catch(Exception e){                      e.printStackTrace();                  }                  label.setText("更新" + (count++) + "更新");              }          }            public void setStop(boolean flag){              this.flag = flag;          }            private int count = 0;          private boolean flag = true;      }        private CounterThread counter = null;      private final JPanel panel = new JPanel();      private final JLabel label = new JLabel("更新0次");      private final JButton startButton = new JButton("开始");      private final JButton endButton = new JButton("结束");  } |

【运行结果】：

**线程的插队**

在编写多线程的程序的时候，经常会遇到让一个线程优先于另外i个线程运行的情况，此时，除了设置这个线程的优先级高（不推荐这种方法）之外，更加直接的办法是采用Thread类中的join（）方法。当插队的线程运行结束之后，其他的线程才能运行。

[?](http://www.cnblogs.com/rollenholt/archive/2011/09/15/2178030.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 | package Thread;    public class ThreadJoinDemo{      public static void main(String[] args){          Thread demo1 = new Thread(new EmergencyThread());          demo1.start();          for(int i = 0; i < 5; ++i){              try{                  Thread.sleep(100);              }catch(InterruptedException e){                  e.printStackTrace();              }              System.out.println("正常情况 :" + i + "号车开始出发");              try{                  //开始插队                  demo1.join();              }catch(InterruptedException e){                  e.printStackTrace();              }          }      }  }    class EmergencyThread implements Runnable{      @Override      public void run(){          for(int i = 0; i < 5; ++i){              try{                  Thread.sleep(100);              }catch(InterruptedException e){                  e.printStackTrace();              }              System.out.println("紧急情况 ：" + i + "号车开始出发");          }      }  } |

【运行结果】：

正常情况 :0号车开始出发

紧急情况：0号车开始出发

紧急情况：1号车开始出发

紧急情况：2号车开始出发

紧急情况：3号车开始出发

紧急情况：4号车开始出发

正常情况 :1号车开始出发

正常情况 :2号车开始出发

正常情况 :3号车开始出发

正常情况 :4号车开始出发

如果我们去掉join哪一行的话，运行结果：（结果不唯一）

紧急情况：0号车开始出发

正常情况 :0号车开始出发

紧急情况：1号车开始出发

正常情况 :1号车开始出发

正常情况 :2号车开始出发

紧急情况：2号车开始出发

紧急情况：3号车开始出发

正常情况 :3号车开始出发

正常情况 :4号车开始出发

紧急情况：4号车开始出发

**线程的同步**

多线程编程的一个重要原因是实现数据的共享，但是如果两个线程同时修改一个数据的话，则会产生同步问题。

下面采用一个2个人同时往银行存钱的例子，银行卡初始金额为100元。每次存10元，大家仔细查看余额。（对于简单的多线程，出错的概率很小，今天很不巧，我一向地下的RP今天居然爆发了，实验了很多次，都没错，最后终于出现了）

先看一下结果吧：

案例说两侧不能出现一样余额的。

代码如下：

[?](http://www.cnblogs.com/rollenholt/archive/2011/09/15/2178030.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 | package Thread;    import java.awt.GridLayout;  import java.awt.event.ActionEvent;  import java.awt.event.ActionListener;    import javax.swing.JButton;  import javax.swing.JFrame;  import javax.swing.JLabel;  import javax.swing.JPanel;  import javax.swing.JScrollPane;  import javax.swing.JTextArea;    public class UnSynBank extends JFrame{      public UnSynBank(){          panel.setLayout(new GridLayout(2, 2, 3, 3));          panel.add(label1);          panel.add(label2);          JScrollPane js1 = new JScrollPane(oneArea,                  JScrollPane.VERTICAL\_SCROLLBAR\_ALWAYS,                  JScrollPane.HORIZONTAL\_SCROLLBAR\_ALWAYS);          panel.add(js1);          JScrollPane js2 = new JScrollPane(twoArea,                  JScrollPane.VERTICAL\_SCROLLBAR\_ALWAYS,                  JScrollPane.HORIZONTAL\_SCROLLBAR\_ALWAYS);          panel.add(js2);            panel2.add(panel);          panel2.add(statrButton);          setContentPane(panel2);            statrButton.addActionListener(new ActionListener(){              @Override              public void actionPerformed(ActionEvent e){                  Thread demo1 = new Thread(new Transfer(bank, oneArea));                  demo1.start();                  Thread demo2 = new Thread(new Transfer(bank, twoArea));                  demo2.start();              }          });            setSize(300, 400);          setVisible(true);          setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);      }        public static void main(String[] args){          new UnSynBank();      }        private final Bank bank = new Bank();      JPanel panel = new JPanel();      JPanel panel2 = new JPanel();      private final JButton statrButton = new JButton("开始存钱");      private final JLabel label1 = new JLabel("一号线程");      private final JLabel label2 = new JLabel("二号线程");      private final JTextArea oneArea = new JTextArea(5, 10);      private final JTextArea twoArea = new JTextArea(5, 10);  }    /\*\*   \* 表示银行的类   \* \*/  class Bank{      public Bank(){        }        public void deposit(int money){          account += money;      }        public int getAccount(){          return account;      }        private int account = 100;  }    /\*\*   \* 表示往账户存钱   \* \*/  class Transfer implements Runnable{      public Transfer(){        }        public Transfer(Bank bank, JTextArea textArea){          this.bank = bank;          this.textArea = textArea;      }        @Override      public void run(){          for(int i = 0; i < 20; ++i){              bank.deposit(10);              String str = textArea.getText();              textArea.setText(str + "账户余额为：" + bank.getAccount() + "\n");          }      }        private Bank bank = null;      private JTextArea textArea = null;  } |

因为一个进程中的所有线程会共享进程中的资源，所以当一个线程还没有将修改之后的结果保存的时候，另外一个线程却进行读取，这样自然会产生错误，所以这个时候就需要我们采用同步来解决问题的。

**使用同步方法实现同步**

所谓同步方法，就是用**synchronized修饰的方法，之所以这十几个字母能解决困难的同步问题，这是和java中的内置锁密切相关的，每一个java对象中有一个内置锁，如果方法使用了synchronized进行修饰的话，内置锁会保护整个方法，也就是在调用方法之前，需要、获得内置锁，否则就会处于阻塞状态，当然这个关键字也可以修饰静态方法，如果调用静态方法，就会锁住整个类**

但是要提醒一下大家，同步是一种高开销的操作，所以应该尽量减少需要同步的内容

**使用特殊域变量实现同步**

**volatile提供了一种免锁的机制，使用这个关键字修饰的域相当于告诉虚拟机，这个域可能会被其他的线程跟新，因此每次读取这个域的时候都需要重新计算，而不是使用寄存器中的值，这个关键字不会提供任何的原子操作，也不能用来修饰final类型的变量、**

 修改后代码为：

[?](http://www.cnblogs.com/rollenholt/archive/2011/09/15/2178030.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 | package com.java;  import java.awt.GridLayout;  import java.awt.event.ActionEvent;  import java.awt.event.ActionListener;    import javax.swing.JButton;  import javax.swing.JFrame;  import javax.swing.JLabel;  import javax.swing.JPanel;  import javax.swing.JScrollPane;  import javax.swing.JTextArea;    public class UnSynBank extends JFrame{      public UnSynBank(){          panel.setLayout(new GridLayout(2, 2, 3, 3));          panel.add(label1);          panel.add(label2);          JScrollPane js1 = new JScrollPane(oneArea,                  JScrollPane.VERTICAL\_SCROLLBAR\_ALWAYS,                  JScrollPane.HORIZONTAL\_SCROLLBAR\_ALWAYS);          panel.add(js1);          JScrollPane js2 = new JScrollPane(twoArea,                  JScrollPane.VERTICAL\_SCROLLBAR\_ALWAYS,                  JScrollPane.HORIZONTAL\_SCROLLBAR\_ALWAYS);          panel.add(js2);            panel2.add(panel);          panel2.add(statrButton);          setContentPane(panel2);            statrButton.addActionListener(new ActionListener(){              @Override              public void actionPerformed(ActionEvent e){                  Thread demo1 = new Thread(new Transfer(bank, oneArea));                  demo1.start();                  Thread demo2 = new Thread(new Transfer(bank, twoArea));                  demo2.start();              }          });            setSize(300, 400);          setVisible(true);          setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);      }        public static void main(String[] args){          new UnSynBank();      }        private final Bank bank = new Bank();      JPanel panel = new JPanel();      JPanel panel2 = new JPanel();      private final JButton statrButton = new JButton("开始存钱");      private final JLabel label1 = new JLabel("一号线程");      private final JLabel label2 = new JLabel("二号线程");      private final JTextArea oneArea = new JTextArea(5, 10);      private final JTextArea twoArea = new JTextArea(5, 10);  }    /\*\*   \* 表示银行的类   \* \*/  class Bank{      public Bank(){        }        public synchronized void deposit(int money){          account += money;      }        public int getAccount(){          return account;      }        private volatile int account = 100;  }    /\*\*   \* 表示往账户存钱   \* \*/  class Transfer implements Runnable{      public Transfer(){        }        public Transfer(Bank bank, JTextArea textArea){          this.bank = bank;          this.textArea = textArea;      }        @Override      public void run(){          for(int i = 0; i < 20; ++i){              bank.deposit(10);              String str = textArea.getText();              textArea.setText(str + "账户余额为：" + bank.getAccount() + "\n");          }      }        private Bank bank = null;      private JTextArea textArea = null;  } |

提醒一下：关于安全域的并发访问：

多线程中的非同步问题出现在对于域的读写上的时候，如果让域自身避免这个问题的话，则不需要修改操作的方法，在java中有3中域自身就可以避免非同步问题：final域，使用volatile域，以及有锁保护的域。

**使用重入锁实现线程同步**

[?](http://www.cnblogs.com/rollenholt/archive/2011/09/15/2178030.html)

|  |  |
| --- | --- |
| 1  2 | import java.util.concurrent.locks.Lock;  import java.util.concurrent.locks.ReentrantLock; |

可以在程序中添加上面两行代码，。然后将Bank修改为：

[?](http://www.cnblogs.com/rollenholt/archive/2011/09/15/2178030.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | class Bank{      public Bank(){        }      public void deposit(int money){          lock.lock();          try{              account += money;          }finally{              lock.unlock();          }      }      public int getAccount(){          return account;      }      private final Lock lock = new ReentrantLock();      private int account = 100;  } |

这样也可以解决非同步问题。至于这个类，大家可以自行去查看API，我只是在这里提醒一下，如果synchronized能够满足需求的话，就使用synchronized关键字，因为这个可以简化代码，如果需要更加高级的功能的时候，就使用Lock对象，在使用ReentrantLock的时候，一定要注意及时释放锁，否则程序会出现死锁。

**使用线程局部变量实现线程同步**

这个例子演示的是两个线程同时修改一个变量，运行结果：

可以发现，每个线程完成修改之后的副本是完全独立的，如果使用TreadLocal来管理变量，则每个使用这个变量的线程都会获得这个变量的一个副本。，并且可以随意修改这个副本，每个线程之间不会影响。

TreadLocal和同步机制都是为了解决多线程中的相同变量访问冲突的问题的，前者采用的是空间还时间，后者采用的是时间换空间

代码如下：

[?](http://www.cnblogs.com/rollenholt/archive/2011/09/15/2178030.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | class Bank{      public Bank(){        }      public void deposit(int money){          account.set(account.get() + money);      }      public int getAccount(){          return account.get();      }      private static ThreadLocal<Integer> account = new ThreadLocal<Integer>(){          @Override          protected Integer initialValue(){              return 100;          }      };  } |

**线程之间的通信**

还记得我在我的笔记java IO总结中给出了一个使用管道流进行线程之间通信的例子：

[?](http://www.cnblogs.com/rollenholt/archive/2011/09/15/2178030.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 | package Thread;    import java.io.IOException;  import java.io.PipedInputStream;  import java.io.PipedOutputStream;    /\*\*   \* 使用管道流进行线程之间的通信   \* \*/  public class PipedTreadDemo{      public static void main(String[] args){          Sender send = new Sender();          Reciver rec = new Reciver();          try{              send.getOut().connect(rec.getReciver());          }catch(Exception e){              e.printStackTrace();          }          new Thread(send).start();          new Thread(rec).start();      }  }    /\*\*   \* 消息发送类   \* \*/  class Sender implements Runnable{      private PipedOutputStream out = null;        public Sender(){          out = new PipedOutputStream();      }        public PipedOutputStream getOut(){          return this.out;      }        @Override      public void run(){          String str = "rollen holt";          try{              out.write(str.getBytes());          }catch(IOException e){              e.printStackTrace();          }          try{              out.close();          }catch(IOException e){              e.printStackTrace();          }      }  }    /\*\*   \* 消息接受类   \* \*/  class Reciver implements Runnable{      private PipedInputStream input = null;        public Reciver(){          input = new PipedInputStream();      }        public PipedInputStream getReciver(){          return this.input;      }        @Override      public void run(){          byte[] bytes = new byte[1024];          int len = 0;          try{              len = input.read(bytes);          }catch(IOException e){              e.printStackTrace();          }          try{              input.close();          }catch(IOException e){              e.printStackTrace();          }          System.out.println("读取的内容为：" + new String(bytes, 0, len));      }  } |

【运行结果】：

读取的内容为：rollen holt

下面，我们在同步的前提下，在举出一个线程通信的例子：

首先摆出运行结果再说：

程序代码如下：

[?](http://www.cnblogs.com/rollenholt/archive/2011/09/15/2178030.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  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137 | package Thread;    /\*\*   \* 线程之间的通信   \* \*/  import java.awt.GridLayout;  import java.awt.event.ActionEvent;  import java.awt.event.ActionListener;    import javax.swing.JButton;  import javax.swing.JFrame;  import javax.swing.JLabel;  import javax.swing.JPanel;  import javax.swing.JScrollPane;  import javax.swing.JTextArea;    public class TreadCommunicate extends JFrame{      public TreadCommunicate(){          panel.setLayout(new GridLayout(2, 2, 3, 3));          panel.add(label1);          panel.add(label2);          JScrollPane js1 = new JScrollPane(oneArea,                  JScrollPane.VERTICAL\_SCROLLBAR\_ALWAYS,                  JScrollPane.HORIZONTAL\_SCROLLBAR\_ALWAYS);          panel.add(js1);          JScrollPane js2 = new JScrollPane(twoArea,                  JScrollPane.VERTICAL\_SCROLLBAR\_ALWAYS,                  JScrollPane.HORIZONTAL\_SCROLLBAR\_ALWAYS);          panel.add(js2);            statrButton.addActionListener(new ActionListener(){              @Override              public void actionPerformed(ActionEvent e){                  Sender sender = new Sender();                  Thread demo1 = new Thread(sender);                  Thread demo2 = new Thread(new Receiver(sender));                  demo1.start();                  demo2.start();              }          });            panel2.add(panel);          panel2.add(statrButton);          setContentPane(panel2);            setSize(300, 400);          setVisible(true);          setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);      }        public static void main(String[] args){          new TreadCommunicate();      }        /\*\*       \* 卖家       \* \*/      class Sender implements Runnable{            @Override          public void run(){              for(int i = 0; i < 5; ++i){                  // 如果已经发送，那么就等待                  while(isValid){                      Thread.yield();                  }                  product = products[i];                  String text = oneArea.getText();                  oneArea.setText(text + "发送" + product + "\n");                  try{                      Thread.sleep(1000);                  }catch(Exception e){                      e.printStackTrace();                  }                  isValid = true;              }          }            public boolean isisValid(){              return this.isValid;          }            public void setValid(boolean flag){              this.isValid = flag;          }            public String getProduct(){              return product;          }            private volatile String[] products = { "《金 瓶梅》", "《红楼梦》", "《平凡的世界》",                  "《流氓老师》", "《西游记》" };          private volatile boolean isValid = false;          private volatile String product;      }// end sender        /\*\*       \* 买家       \* \*/      class Receiver implements Runnable{          public Receiver(){            }            public Receiver(Sender sender){              this.sender = sender;          }            @Override          public void run(){              for(int i = 0; i < 5; ++i){                  // 如果没有发送，就等待                  while(!sender.isisValid()){                      Thread.yield();                  }                  String test = twoArea.getText();                  twoArea.setText(test + "接受到" + sender.getProduct() + "\n");                  try{                      Thread.sleep(1000);                  }catch(Exception e){                      e.printStackTrace();                  }                  sender.setValid(false);              }          }            private Sender sender;      }        JPanel panel = new JPanel();      JPanel panel2 = new JPanel();      private final JButton statrButton = new JButton("开始交易");      private final JLabel label1 = new JLabel("卖家");      private final JLabel label2 = new JLabel("买家");      private final JTextArea oneArea = new JTextArea(5, 10);      private final JTextArea twoArea = new JTextArea(5, 10);  } |

**死锁的范例**

下面绝对不是本人蛋疼的写出这个一个更加叫人蛋疼的程序。只是给出了一个例子：

[?](http://www.cnblogs.com/rollenholt/archive/2011/09/15/2178030.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 | /\*\*   \* 简单的死锁   \* \*/  public class DeadLockDemo implements Runnable{      @Override      public void run(){          // 获得当前线程的名字          String str = Thread.currentThread().getName();          System.out.println(str + ": flag= " + flag);          if(flag){              synchronized (obj1){                  try{                      Thread.sleep(1000);                  }catch(Exception e){                      e.printStackTrace();                  }                  System.out.println(str + "已经进入同步快obj1，准备进入同步快obj2");                  synchronized (obj2){                      System.out.println(str + "已经进入同步快obj2");                  }              }          }          if(!flag){              synchronized (obj2){                  try{                      Thread.sleep(1000);                  }catch(Exception e){                      e.printStackTrace();                  }                  System.out.println(str + "已经进入同步快obj2，准备进入同步快obj1");                  synchronized (obj1){                      System.out.println(str + "已经进入同步快obj1");                  }              }          }      }        public static void main(String[] args){          DeadLockDemo demo1 = new DeadLockDemo();          DeadLockDemo demo2 = new DeadLockDemo();          demo1.flag = true;          demo2.flag = false;          new Thread(demo1).start();          new Thread(demo2).start();      }        private boolean flag;      private final Object obj1 = new Object();      private final Object obj2 = new Object();  } |

【运行结果】

（我承认我今天RP爆发，我运行了10次，还是没出现死锁那种情况，但是这个程序确实可以产生死锁的，哪位运行这个程序，要是产生了死锁，麻烦说一下，谢谢）

**使用线程池优化多线程编程**

这个例子使用的是Executors类，读者自行查看API，因为要解说的话，就太多了。

下面这个例子给出了使用线程池和不使用线程池的情况下的效率的问题。

[?](http://www.cnblogs.com/rollenholt/archive/2011/09/15/2178030.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 | package Thread;    import java.util.concurrent.ExecutorService;  import java.util.concurrent.Executors;    /\*\*   \* 使用线程池优化多线程编程   \* \*/  public class ThreadPoolDemo{      public static void main(String[] args){          Runtime run = Runtime.getRuntime();          // 为了减少误差          run.gc();          long currentTime = System.currentTimeMillis();          long freemonery = run.freeMemory();          for(int i = 0; i < 10000; ++i){              new Thread(new Temo()).start();          }          System.out.println("独立运行10000个线程占用内存为："                  + (freemonery - run.freeMemory()));          System.out.println("独立运行10000个线程占用时间为："                  + (System.currentTimeMillis() - currentTime));          // 下面使用线程池来试试          run.gc();          freemonery = run.freeMemory();          currentTime = System.currentTimeMillis();          ExecutorService executorService = Executors.newFixedThreadPool(3);          for(int i = 0; i < 10000; ++i){              executorService.submit(new Temo());          }          System.out.println("使用线程池运行10000个线程占用内存为："                  + (freemonery - run.freeMemory()));          System.out.println("使用线程池运行10000个线程占用时间为："                  + (System.currentTimeMillis() - currentTime));      }  }    class Temo implements Runnable{      @Override      public void run(){          count++;      }        private int count = 0;  } |

【运行结果】：

独立运行10000个线程占用内存为：3490440

独立运行10000个线程占用时间为：1808

使用线程池运行10000个线程占用内存为：1237424

使用线程池运行10000个线程占用时间为：62

**关于哲学家就餐的问题**

由于代码比较长，所以单独列出为一篇文章

地址：<http://www.cnblogs.com/rollenholt/archive/2011/09/15/2178004.html>

**使用信号量实现线程同步**

现在我们继续回答之前银行存款的问题，相信大家还没有忘记，哈哈，真是不好意思，本来线程同步这一块应该整理在一起的。

一个信号量有3中操作，而且他们全部都是原子的，初始化，增加，减少。增加可以为一个进程解除阻塞，减少可以为一个进程进入阻塞。

Semaphore类是一个技术信号量，从概念上信号量维持了一个许可集。

现在我们继续看上面的银行存款问题：

[?](http://www.cnblogs.com/rollenholt/archive/2011/09/15/2178030.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  120 | package Thread;    import java.awt.GridLayout;  import java.awt.event.ActionEvent;  import java.awt.event.ActionListener;  import java.util.concurrent.Semaphore;    import javax.swing.JButton;  import javax.swing.JFrame;  import javax.swing.JLabel;  import javax.swing.JPanel;  import javax.swing.JScrollPane;  import javax.swing.JTextArea;    public class synDemo extends JFrame{      public synDemo(){          panel.setLayout(new GridLayout(2, 2, 3, 3));          panel.add(label1);          panel.add(label2);          JScrollPane js1 = new JScrollPane(oneArea,                  JScrollPane.VERTICAL\_SCROLLBAR\_ALWAYS,                  JScrollPane.HORIZONTAL\_SCROLLBAR\_ALWAYS);          panel.add(js1);          JScrollPane js2 = new JScrollPane(twoArea,                  JScrollPane.VERTICAL\_SCROLLBAR\_ALWAYS,                  JScrollPane.HORIZONTAL\_SCROLLBAR\_ALWAYS);          panel.add(js2);            panel2.add(panel);          panel2.add(statrButton);          setContentPane(panel2);            statrButton.addActionListener(new ActionListener(){              @Override              public void actionPerformed(ActionEvent e){                    Thread demo1 = new Thread(new Transfer1(bank, oneArea,                          semaphore));                  demo1.start();                  Thread demo2 = new Thread(new Transfer1(bank, twoArea,                          semaphore));                  demo2.start();              }          });            setSize(300, 400);          setVisible(true);          setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);      }        public static void main(String[] args){          new synDemo();      }        Semaphore semaphore = new Semaphore(1, true);      private final Bank1 bank = new Bank1();      JPanel panel = new JPanel();      JPanel panel2 = new JPanel();      private final JButton statrButton = new JButton("开始存钱");      private final JLabel label1 = new JLabel("一号线程");      private final JLabel label2 = new JLabel("二号线程");      private final JTextArea oneArea = new JTextArea(5, 10);      private final JTextArea twoArea = new JTextArea(5, 10);  }    /\*\*   \* 表示银行的类   \* \*/  class Bank1{      public Bank1(){        }        public void deposit(int money){          account += money;      }        public int getAccount(){          return account;      }        private int account;  }    /\*\*   \* 表示往账户存钱   \* \*/  class Transfer1 implements Runnable{      public Transfer1(){        }        public Transfer1(Bank1 bank, JTextArea textArea, Semaphore semaphore){          this.bank = bank;          this.textArea = textArea;          this.semaphore = semaphore;      }        @Override      public void run(){          for(int i = 0; i < 20; ++i){              // 获得许可              try{                  semaphore.acquire();              }catch(InterruptedException e){                  e.printStackTrace();              }              bank.deposit(10);              String str = textArea.getText();              textArea.setText(str + "账户余额为：" + bank.getAccount() + "\n");              // 释放许可              semaphore.release();          }      }        // 注意      private Semaphore semaphore;      private Bank1 bank = null;      private JTextArea textArea = null;  } |

运行结果：

**使用原子变量实现线程同步**

[?](http://www.cnblogs.com/rollenholt/archive/2011/09/15/2178030.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 | package Thread;    import java.awt.GridLayout;  import java.awt.event.ActionEvent;  import java.awt.event.ActionListener;  import java.util.concurrent.atomic.AtomicInteger;    import javax.swing.JButton;  import javax.swing.JFrame;  import javax.swing.JLabel;  import javax.swing.JPanel;  import javax.swing.JScrollPane;  import javax.swing.JTextArea;    public class synDemo extends JFrame{      public synDemo(){          panel.setLayout(new GridLayout(2, 2, 3, 3));          panel.add(label1);          panel.add(label2);          JScrollPane js1 = new JScrollPane(oneArea,                  JScrollPane.VERTICAL\_SCROLLBAR\_ALWAYS,                  JScrollPane.HORIZONTAL\_SCROLLBAR\_ALWAYS);          panel.add(js1);          JScrollPane js2 = new JScrollPane(twoArea,                  JScrollPane.VERTICAL\_SCROLLBAR\_ALWAYS,                  JScrollPane.HORIZONTAL\_SCROLLBAR\_ALWAYS);          panel.add(js2);            panel2.add(panel);          panel2.add(statrButton);          setContentPane(panel2);            statrButton.addActionListener(new ActionListener(){              @Override              public void actionPerformed(ActionEvent e){                    Thread demo1 = new Thread(new Transfer1(bank, oneArea));                  demo1.start();                  Thread demo2 = new Thread(new Transfer1(bank, twoArea));                  demo2.start();              }          });            setSize(300, 400);          setVisible(true);          setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);      }        public static void main(String[] args){          new synDemo();      }        private final Bank1 bank = new Bank1();      JPanel panel = new JPanel();      JPanel panel2 = new JPanel();      private final JButton statrButton = new JButton("开始存钱");      private final JLabel label1 = new JLabel("一号线程");      private final JLabel label2 = new JLabel("二号线程");      private final JTextArea oneArea = new JTextArea(5, 10);      private final JTextArea twoArea = new JTextArea(5, 10);  }    /\*\*   \* 表示银行的类   \* \*/  class Bank1{      public Bank1(){        }        public void deposit(int money){          account.addAndGet(money);      }        public int getAccount(){          return account.get();      }      //注意      private final AtomicInteger account = new AtomicInteger(100);  }    /\*\*   \* 表示往账户存钱   \* \*/  class Transfer1 implements Runnable{      public Transfer1(){        }        public Transfer1(Bank1 bank, JTextArea textArea){          this.bank = bank;          this.textArea = textArea;      }        @Override      public void run(){          for(int i = 0; i < 20; ++i){              bank.deposit(10);              String str = textArea.getText();              textArea.setText(str + "账户余额为：" + bank.getAccount() + "\n");            }      }        private Bank1 bank = null;      private JTextArea textArea = null;  } |

【运行结果】：