多线程：

进程和线程

假设进程是一个程序，那么在这个程序的基础运行的一系列子程序，我们叫做线程。

当我们结束一个进程的时候，线程也就停止了。反过来，线程如果停止了，进程是不一定停止。

Java本身是一种支持多线程处理的语言

在Java提供两种方式来实现多线程：

1. 继承Thread类
2. 通过实现Runnable接口

|  |
| --- |
| **package** com.wancy.ThreadDemo;  **public** **class** ThreadDemo01 **extends** Thread {  **private** String name;    **public** ThreadDemo01(String name){  **super**();  **this**.name=name;  }  **public** **void** run(){  **for**(**int** i=0;i<10;i++){  System.*out*.println("线程的运行"+name+i);  }  }    } |

|  |
| --- |
| **package** com.wancy.ThreadDemo;  **public** **class** ThreadTest {  **public** **static** **void** main(String[] args) {  ThreadDemo01 th1=**new** ThreadDemo01("线程A");  ThreadDemo01 th2=**new** ThreadDemo01("线程B");  th1.run();  th2.run();  }  } |

我们发现，如果直接调用类里面的run（），发现其实线程的执行并没有交互，是顺序执行的。

|  |
| --- |
| **package** com.wancy.ThreadDemo;  **public** **class** ThreadTest {  **public** **static** **void** main(String[] args) {  ThreadDemo01 th1=**new** ThreadDemo01("线程A");  ThreadDemo01 th2=**new** ThreadDemo01("线程B");  th1.start();  th2.start();  }  } |

当我们使用了start（）方法之后，输出的结果实现了线程的交互，从而达到了多线程的目的。

但是直接继承Thread类他有单一继承的局限性，所有我们有了下面的方法。

|  |
| --- |
| **package** com.wancy.ThreadDemo;  **public** **class** ThreadTest {  **public** **static** **void** main(String[] args) {  ThreadDemo01 th1=**new** ThreadDemo01("线程A");  ThreadDemo01 th2=**new** ThreadDemo01("线程B");  **new** Thread(th1).start();  **new** Thread(th2).start();  }  }  **package** com.wancy.ThreadDemo;  **public** **class** ThreadDemo01 **implements** Runnable {  **private** String name;    **public** ThreadDemo01(String name){  **super**();  **this**.name=name;  }  **public** **void** run(){  **for**(**int** i=0;i<10;i++){  System.*out*.println("线程的运行"+name+i);  }  }    } |

通过代码我们发现，实例化runnable接口来实现多线程的方法其实就是为了解决单一继承的局限而产生的，事实上，他还是通过实例化Thread类来实现多线程。

|  |
| --- |
| **package** com.wancy.ThreadDemo;  **public** **class** SaleTicket **extends** Thread {  **private** **int** ticket=10;  **public** **void** run(){  //现在加入的每个售票窗口有50个人在等待买票  **for**(**int** i=0;i<50;i++){  **if**(ticket>0){  System.*out*.println("卖票成功，剩余票数为："+ --ticket);  }  }  }  }  **package** com.wancy.ThreadDemo;  **public** **class** Sale {  **public** **static** **void** main(String[] args) {  SaleTicket s1=**new** SaleTicket();  SaleTicket s2=**new** SaleTicket();  SaleTicket s3=**new** SaleTicket();  SaleTicket s4=**new** SaleTicket();  s1.start();  s2.start();  s3.start();  s4.start();    }  } |

通过以上实例，我们发现，每个窗口各自卖了10张票，而实际上总共只有10张票，这就与实际情况不相符，从而说明了直接继承Thread类的方法无法进行资源共享。

|  |
| --- |
| **package** com.wancy.ThreadDemo;  **public** **class** SaleTicket **implements** Runnable{  **private** **int** ticket=10;  **public** **void** run(){  //现在加入的每个售票窗口有50个人在等待买票  **for**(**int** i=0;i<50;i++){  **if**(ticket>0){  System.*out*.println("卖票成功，剩余票数为："+ --ticket);    }  }  }  }  **package** com.wancy.ThreadDemo;  **public** **class** Sale {  **public** **static** **void** main(String[] args) {  SaleTicket s1=**new** SaleTicket();  SaleTicket s2=**new** SaleTicket();  SaleTicket s3=**new** SaleTicket();  SaleTicket s4=**new** SaleTicket();  **new** Thread(s1).start();  **new** Thread(s2).start();  **new** Thread(s3).start();  **new** Thread(s4).start();    }  } |

通过以上代码我们看到，虽然遇到了线程同步的问题，但是上，这种方法已经实现了资源的共享。

|  |  |
| --- | --- |
| 直接继承Thread | 实现Runnable |
| 单继承的局限（不能共享资源） | 一个类可以实现多个接口，从而实现了资源的共享 |
| 两者的联系：Runnable其实还是需要通过Thread来启动多线程的 | |

那么实际应用当中，我们很少使用直接继承，大部分时候我们使用实现runnable接口的方法。

看一下Thread类的几个重要的方法：

设置线程的名称：public final void **setName**([String](mk:@MSITStore:C:\Documents%20and%20Settings\Administrator\桌面\JDK_API_1_6_zh_CN.CHM::/java/lang/String.html) name)

取得线程的名称: public final [String](mk:@MSITStore:C:\Documents%20and%20Settings\Administrator\桌面\JDK_API_1_6_zh_CN.CHM::/java/lang/String.html) **getName**()

一旦设置好了线程名字，特别是已经开启线程了之后，我们尽量不要去修改线程的名字。

public **Thread**([Runnable](mk:@MSITStore:C:\Documents%20and%20Settings\Administrator\桌面\JDK_API_1_6_zh_CN.CHM::/java/lang/Runnable.html) target) 为Runnable接口分配新的Thread对象

public **Thread**([Runnable](mk:@MSITStore:C:\Documents%20and%20Settings\Administrator\桌面\JDK_API_1_6_zh_CN.CHM::/java/lang/Runnable.html) target, [String](mk:@MSITStore:C:\Documents%20and%20Settings\Administrator\桌面\JDK_API_1_6_zh_CN.CHM::/java/lang/String.html) name)

为Runnable接口分配新的Thread对象，同时设置线程的名字。

public static [Thread](mk:@MSITStore:C:\Documents%20and%20Settings\Administrator\桌面\JDK_API_1_6_zh_CN.CHM::/java/lang/Thread.html) **currentThread**() 获得当前正在运行的线程

|  |
| --- |
| **package** com.wancy.ThreadDemo;  **public** **class** ThreadTest {  **public** **static** **void** main(String[] args) {  ThreadDemo01 th1=**new** ThreadDemo01();  ThreadDemo01 th2=**new** ThreadDemo01();  **new** Thread(th1,"线程A").start();  **new** Thread(th2,"线程B").start();  }  }  **package** com.wancy.ThreadDemo;  **public** **class** ThreadDemo01 **implements** Runnable {  **private** String name;    // public ThreadDemo01(String name){  // super();  // this.name=name;  // }  **public** **void** run(){  **for**(**int** i=0;i<3;i++){  System.*out*.println(i);  System.*out*.println(Thread.*currentThread*().getName());    }  }    } |

main方法本身也是一个线程，而且是优先级是为默认的优先级

|  |
| --- |
| **package** com.wancy.ThreadDemo;  **public** **class** ThreadTest {  **public** **static** **void** main(String[] args) {  ThreadDemo01 th1=**new** ThreadDemo01();    **new** Thread(th1,"线程A").start();    th1.run();  }  } |

线程的优先级：

设置线程优先级的方法（优先级最高的线程是优先执行的）

public final void **setPriority**(int newPriority)

public final int **getPriority**()

线程的[优先级](http://zhidao.baidu.com/search?word=%E4%BC%98%E5%85%88%E7%BA%A7&fr=qb_search_exp&ie=utf8)代表该线程的重要程度，当有多个线程同时处于可[执行状态](http://zhidao.baidu.com/search?word=%E6%89%A7%E8%A1%8C%E7%8A%B6%E6%80%81&fr=qb_search_exp&ie=utf8)并等待获得 CPU 时间时，[线程调度](http://zhidao.baidu.com/search?word=%E7%BA%BF%E7%A8%8B%E8%B0%83%E5%BA%A6&fr=qb_search_exp&ie=utf8)系统根据各个线程的[优先级](http://zhidao.baidu.com/search?word=%E4%BC%98%E5%85%88%E7%BA%A7&fr=qb_search_exp&ie=utf8)来决定给谁分配 CPU 时间，[优先级](http://zhidao.baidu.com/search?word=%E4%BC%98%E5%85%88%E7%BA%A7&fr=qb_search_exp&ie=utf8)高的线程有更大的机会获得 CPU 时间，优先级低的线程也不是没有机会，只是机会要小一些罢了。

我们如果没有给一个线程定义优先级，那么他的优先级就是默认的优先级

了解多线程的同步以及死锁：

1. 使用同步代码块（所谓的同步代码块，就是用synchronized定义之后的{}里面的代码）

|  |
| --- |
| **package** com.wancy.ThreadDemo;  **public** **class** SaleTicket **implements** Runnable{  **private** **int** ticket=10;    **public** **void** run(){  //现在加入的每个售票窗口有50个人在等待买票  **for**(**int** i=0;i<50;i++){  **synchronized** (**this**){  **if**(ticket>0){  System.*out*.println("卖票成功，剩余票数为："+ --ticket);    }    }  }  }    } |

1. 使用同步方法

|  |
| --- |
| **package** com.wancy.ThreadDemo;  **public** **class** SaleTicket **implements** Runnable{  **private** **int** ticket=10;    **public** **void** run(){  //现在加入的每个售票窗口有50个人在等待买票  **for**(**int** i=0;i<50;i++){    **this**.maip();  }  }  **public** **synchronized** **void** maip(){  **if**(ticket>0){  System.*out*.println("卖票成功，剩余票数为："+ --ticket);    }    }    }  **package** com.wancy.ThreadDemo;  **public** **class** Sale {  **public** **static** **void** main(String[] args) {  SaleTicket s1=**new** SaleTicket();    **new** Thread(s1).start();  **new** Thread(s1).start();  **new** Thread(s1).start();  **new** Thread(s1).start();    }  } |

死锁：

过多的同步可能导致产生死锁的问题。

在使用同步处理的情况下，一定要考虑清楚会不会产生死锁，如果会，能不能避免掉。