多线程

课程目标

1. 理解进程和线程
2. 理解多线程的基本概念
3. 实现多线程(重点)
4. 线程的几种状态(重点理解)
5. 理解线程同步(锁)

>>模拟抢票程序

1. 理解进程和线程
2. 什么是进程:进程是应用程序执行的实例，有独立的内存空间和系统资源.

注意:一个程序一个进程

1. 什么是线程:

线程是cpu调度和分派的基本单元，可以完成独立的顺序控制流程.

一个线程完成一件事情(一件任务)

1. 多线程
2. 什么是多线程:

如果在一个进程中同时运行了多个线程，用来完成不同的工作，则称之为”多线程”.

注意:多个线程交替占用cpu资源，而非真正的并行执行.

1. 多线程的好处.
2. 充分利用cpu资源提高效率
3. 简化编程模型
4. 带来良好的用户体验
5. 理解主线程(main方法即为主线程)

|  |
| --- |
| **public static void** main(String[] args) { *//主线程  //获取主线程对象* Thread main=Thread.*currentThread*();  *//设置主线程名称* main.setName(**"主线程"**);  *//获取线程名称* System.***out***.println(main.getName()); } |

1. 多线程的创建方式:

4.1 通过继承Thread类

4.2 实现java.lang.runable接口(推荐)

1. 多线程的使用步骤:

定义线程🡪创建线程对象🡪启动线程🡪终止线程

1. 创建线程
2. 第一方式:继承Thread类创建线程

1.1定义线程类:创建一个类继承Thread类，并重写run方法

|  |
| --- |
| *//定义线程类* **public class** MyThread **extends** Thread {  @Override  **public void** run() { *//线程做任务的方法*  *//子线程完成的任务代码块* **for**(**int** i=10;i<=20;i++){  System.***out***.println(i);  }  } } |

1.2在主线程中创建线程对象并启动

|  |
| --- |
| *//2.创建子线程对象* MyThread myThread=**new** MyThread();  myThread.setName(**"线程A:"**); //线程对象取名 *//3.启动线程* myThread.start(); |

1. 实现Runnable接口创建线程(推荐用)

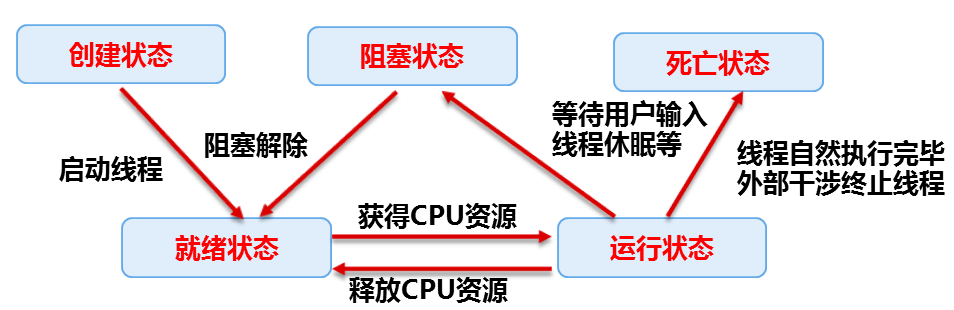
2.1创建一个类去实现Runnable接口并实现run方法

|  |
| --- |
| **public class** MyThreadRun **implements** Runnable {  @Override  **public void** run() { *//子线程执行的代码块* String name=Thread.*currentThread*().getName();  **for** (**int** i=1;i<=10;i++){  System.***out***.println(**"我是"**+name+**":"**+i);  }  } } |

2.2在主线程中创建子线程对象并启动

|  |
| --- |
| *//2.创建子线程对象  //2.1定义线程任务对象* MyThreadRun myThreadRun=**new** MyThreadRun(); *//2.2创建线程* Thread thread=**new** Thread(myThreadRun); thread.setName(**"线程A"**); thread.start(); |

四、线程的状态(面试)



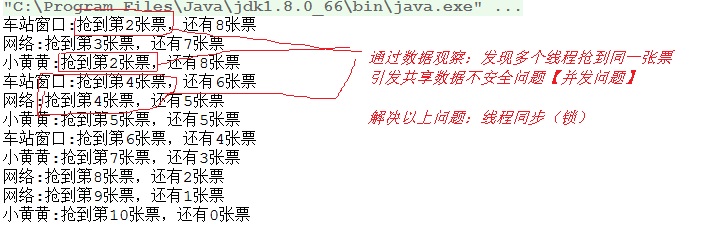
1. 综合练习:抢票示例
2. 定义类

|  |
| --- |
| *//抢票类* **public class** Ticket **implements** Runnable {  *//定义变量存放余票* **private int count**=10;  **private int num**=0; *//已售多少张票* @Override  **public void** run() {  **while**(**true**){  **if**(**count**==0){  **break**;  }   *//售票* **count**--;  **num**++;   String name=Thread.*currentThread*().getName();  System.***out***.println(name+**":抢到第"**+**num**+**"张票，还有"**+**count**+**"张票"**);   **try** {  Thread.*sleep*(500); *//网络延时* } **catch** (InterruptedException e) {  e.printStackTrace();  }  }  } } |

1. 创建多线程模拟购票

|  |
| --- |
| **public class** TestTicket {  **public static void** main(String[] args) {  *//创建线程  //1.定义任务对象* Ticket ticket=**new** Ticket();  *//2.创建线程对象* Thread t1=**new** Thread(ticket,**"车站窗口"**);  Thread t2=**new** Thread(ticket,**"网络"**);  Thread t3=**new** Thread(ticket,**"小黄黄"**);  *//3.启动* t1.start();  t2.start();  t3.start();  } } |

注意:运行结果



1. 解决多线程引发数据共享问题

答:线程同步(称为锁)

1. 同步代码块

|  |
| --- |
| *@Override 同步代码块 public void run() {  while(true){  synchronized (this){ //同步代码块 一次只一个人操作  if(count==0){  break;  }   //售票  count--;  num++;   String name=Thread.currentThread().getName();  System.out.println(name+":抢到第"+num+"张票，还有"+count+"张票");   try {  Thread.sleep(500); //网络延时  } catch (InterruptedException e) {  e.printStackTrace();  }  }  } }* |

1. 同步方法

|  |
| --- |
| **public void** run() {  **while**(**true**){  **if**(sell()){  **break**; *//退出环循* }  } }  *//编写方法抢票* **public synchronized boolean** sell(){ *//同步方法* **if**(**count**==0){  **return true**;  }   *//售票* **count**--;  **num**++;   String name=Thread.*currentThread*().getName();  System.***out***.println(name+**":抢到第"**+**num**+**"张票，还有"**+**count**+**"张票"**);   **try** {  Thread.*sleep*(500); *//网络延时* } **catch** (InterruptedException e) {  e.printStackTrace();  }  **return false**; } |

扩展: *lambda表达式创建线程*

|  |
| --- |
| *//扩展:lambda表达式 // new Thread(new Runnable() { // @Override // public void run() { // for(int i=1;i<10;i++){ // System.out.println(i); // } // } // }).start();* **new** Thread(()->{  **for**(**int** i=1;i<=10;i++){  System.***out***.println(i);  }  }).start(); |