**第5课 Java 的多线程**

|  |  |  |
| --- | --- | --- |
| **课 题** | Java 的多线程 | |
| **课 时** | 4课时（180 min）。 | |
| **教学目标** | **知识技能目标：**  1．理解什么是进程、线程以及程序。  2．掌握什么是多线程程序，理解其作用。  **思政育人目标：**  让学生通过学习Java 的多线程，为了建立这些线程正在同步执行的感觉，Java 快速地把控制从一个线程切换到另一个线程。 | |
| **教学重难点** | **教学重点：**龟兔赛跑  **教学难点：**程序、进程与线程 | |
| **教学方法** | 讲授法、问答法、讨论法 | |
| **教学用具** | 电脑、投影仪、多媒体课件、教材 | |
| **教学设计** | 第1节课：考勤（2min）--知识讲解（40min）--作业布置（3min）  第2节课：知识讲解（40min）--课堂小结（3min）--作业布置（2min）  第3节课：知识讲解（40min）--课堂小结（3min）--作业布置（2min）  第4节课：知识讲解（40min）--课堂小结（3min）--作业布置（2min） | |
| **教学过程** | **主 要 教 学 内 容 及 步 骤** | **设计意图** |
| **考勤**  **（2min）** | ■【教师】清点上课人数，记录好考勤  ■【学生】班干部报请假人员及原因 | 培养学生的组织纪律性,掌握学生的出勤情况 |
| **知识讲解**  （40min） | **【教师】**展示龟兔赛跑（一）  **任务描述**  编写程序，运用多线程技术来实现龟兔赛跑的过程。  **任务目标**  1. 能够理解进程、线程以及程序的定义。  2. 能够掌握什么是多线程程序，理解其作用。  3. 能够掌握在 Java 语言系统中创建多线程程序的方法，以及实现多线程程序所需要的类。  4. 能够掌握多线程程序设计要点，理解线程的生命周期。  5. 能够理解线程异步的概念，掌握怎样处理线程间的同步异步问题。  6. 懂得“奋斗不止”的道理，做一个“一心多用”的人。  **预备知识**  在编程时，我们会面临需要在程序中同时运行不同功能的情况，这时就必须采用多线程技术。那么在 Java 中怎样来解决这个问题呢？它有什么相关的规定与相关的类与方法？多线程同步问题在面向对象程序设计中程序员都会定义一些类以便解决实际中遇到的问题，那怎样遵循 Java 的语法规则来定义一个类呢？  **一、程序、进程与线程**  **（一）程序**  程序是一段静态的代码，它是应用软件执行的蓝本。  **（二）进程**  进程是程序的一次动态执行过程，它对应了从代码加载、执行至执行完毕的一个完整过程，这个过程也是进程本身从产生、发展至消亡的过程。如果把银行一天工作比作一个进程，那么早上打上班铃是进程开始，晚上打下班铃是进程的结束。  **（三）线程**  线程是比进程更小的执行单位。一个进程在其执行过程中，可以产生多个线程，形成多条执行线索。每条线索，即每个线程也有它自身的产生、存在和消亡的过程，也是一个动态的概念。就像银行一天的工作开始后，可以有多个不同的“线程”为客户服务，如财会部门、出纳部门、保安部门等。我们知道，每个进程都有一段专用的内存区域，与此不同的是，线程间可以共享相同的内存单元（包括代码与数据），并利用这些共享单元来实现数据交换、实时通信与必要的同步操作。例如：在银行一天的工作开始后，财会部门、出纳部门、保安部门这三个线程共享银行的电力资源，财会部门、出纳部门可能共享银行的账目数据等。多线程的程序能更好地表达和解决现实世界的具体问题，是计算机应用开发和程序设计的一个必然发展趋势。  **二、线程调度与优先级**  处于就绪状态的线程首先进入就绪队列排队等候处理器资源，同一时刻在就绪队列中的线程可能有多个。多线程系统会给每个线程自动分配一个线程的优先级，任务较紧急重要的线程，其优先级就较高；相反则较低。在线程排队时，优先级高的线程可以排在较前的位置，能优先享用到处理器资源，而优先级较低的线程则只能等到排在它前面的优先级高的线程执行完毕之后才能获得处理器资源。对于优先级相同线程，则遵循队列的“先进先出”的原则，即先进入就绪状态排队线程被优先分配到处理器资源，随后才为后进入队列的线程服务。  当一个在就绪队列中排队的线程被分配到处理器资源而进入运行状态之后，这个线程就称为是被“调度”或被线程调度管理器选中了。线程调度管理器负责管理线程排队和处理器在线程间的分配，一般都配有一个精心设计的线程调度算法。在 Java 系统中，线程调度依据优先级基础上的“先到先服务”原则。  **【学生】**思考、讨论。 | **展示龟兔赛跑（一），让学生更加仔细阅读，从而激发学生学习欲望。** |
| **作业布置**（3min） | **【教师】**布置课后作业  简述线程。 | 通过课后练习，使学生巩固所学新知识 |
| **知识讲解**  （40min） | **【教师】**展示龟兔赛跑（二）  **三、如何在程序中实现多线程**  在程序中实现多线程有两个途径：创建 Thread 类的子类或实现 Runnable 接口。无论采用哪种途径，程序员可以控制的关键性操作有两个：  （1）定义用户线程的操作，即定义用户线程的 run( ) 方法；  （2）在适当时候建立用户线程实例。  下面就分别探讨两条不同途径是如何分别完成这两个关键性操作的。  **（一）用 Thread 类的子类创建线程**  Thread 类的构造函数如下：  Thread()  Thread(Runnable target)  Thread(Runnable target, String name)  Thread(String name)  Thread(ThreadGroup group, Runnable target)  Thread(ThreadGroup group, Runnable target, String name)  Thread(ThreadGroup group, String name)  在这个途径中，用户程序需要创建自己的 Thread 类的子类（Thread 在 java.lang 包中），并在子类中重新定义自己的 run( ) 方法，该 run( ) 方法中包含了线程的操作。这样程序需要建立自己的线程时，只需要创建一个已定义好的 Thread 子类的实例就可以了。  当你创建的线程调用 start( ) 方法开始运行时，run( ) 方法将被自动执行。请看下面的例子：  public class ThreadTest  {  static LeftHand left;  static RightHand right;  public static void main(String args[])  {  left = new LeftHand();  right = new RightHand(); // 创建两个线程  left.start();  // 线程开始运行后 ,Lefthand 类中的 run 方法将被执行  right.start();  }  }  class LeftHand extends Thread  {  public void run()  {  for(int i = 0;i<=5;i++)  {  System.out.println("I am student");  try  {  sleep(500);  }catch(InterruptedException e){}  }  }  }  class RightHand extends Thread  {  public void run()  {  for(int i = 0;i<=5;i++)  {  System.out.println("I am oookkk");  try  {  sleep(300);  }  catch(InterruptedException e){}  }  }  }  程序运行结果如图 5−1 所示。    在上述例子中，在 main 主线程中创建了两个新的线程 left 和 right。当 left 调用 start( ) 开始运行时，类 LeftHand 中的 run( ) 将自动被执行。  我们来分析一下上面程序的输出结果。left 线程首先开始执行，这时 LeftHand 类中的 run 方法开始执行，输出“I am student”后，left 主动“休息”500 毫秒，让出了 CPU。这时正在排队等待 CPU 的 right 线程的 run 方法马上被执行，输出“I am oookkk”，right 再主动让出 CPU300 毫秒后又来排队等待 CPU 服务，这时 right 发现 left还没有“醒来”，即没有来排队抢占 CPU，因此 left 的 run 方法被执行，又输出“I am oookkk”……的程序执行结果。  **（二）实现 Runnable 接口**  public class MultiThread  {  public static void main(String args[])  {  Threads s1 = new Threads(1);  Threads s2 = new Threads(2);  Thread t1 = new Thread(s1);  Thread t2 = new Thread(s2);  t1.start();  t2.start();  }  }  class Threads implements Runnable  {  int id;  Threads(int i)  {  id = i;  }  public void run()  {  for(int i = 0;i<20;i++)  System.out.println(id);  }  }  在上述例子中，编写了一个类 Threads 实现了 Runnable 接口，并且覆盖了 Runnable接口中的 run 方法。主类 MultiThread 中以 Threads 类的对象 s1、s2 作为参数构造了Thread 类的两个对象 t1、t2，即生成两个线程，通过调用 start( ) 方法启动这两个线程的运行。  注意：Threads 不是线程类。  **【学生】**思考、讨论。 | **通过教师讲解，了解龟兔赛跑（二）的基本理论知识。** |
| **课堂小结**  （3min） | 【**教师**】**回顾和总结本节课的知识点。**  **这节课我们一起学习了龟兔赛跑（二），让学生了解如何实现 Runnable 接口。** | 通过对所学知识的回顾，培养学生的归纳总结能力 |
| **作业布置**（2min） | **【教师】**布置课后作业  上面程序包括几个线程？ | 通过课后练习，使学生巩固所学新知识 |
| **知识讲解**（45min） | **【教师】**展示龟兔赛跑（三）  **四、多线程程序设计要点**  **（一）线程的生命周期**  每个 Java 程序都有一个默认的主线程。对于应用程序，主线程是 main( ) 方法执行的线索（所以称之为主线程，而不称作进程，是因为我们的程序在执行时，计算机仍可以浏览网页，也可以后台打印等）。Java 语言使用 Thread 类及其子类的对象来表示线程，新建的线程在它的一个完整的生命周期中通常要经历的五种状态，如图 5−2 所示。    （1）新建：当一个 Thread 类或其子类的对象被声明并创建时，新生的线程对象处于新建状态。此时它已经有了相应的内存空间和其他资源。  （2）就绪：处于新建状态的线程被启动后，将进入线程队列排队等待 CPU 服务，此时它已经具备了运行的条件，一旦轮到它来享用 CPU 资源时，就可以脱离创建它的主线程独立开始自己的生命周期了。  （3）运行：当就绪状态的线程被调度并获得处理器资源时，便进入运行状态。每一个 Thread 类及其子类的对象都有一个重要的 run( ) 方法，当线程对象被调度执行时，它将自动调用本对象的 run( ) 方法，从第一句开始顺次执行，run( ) 方法定义了这个线程的操作和功能。  （4）阻塞：一个正在执行的线程在某些特殊情况下，如被人为挂起或需要执行费时的输入输出操作时，将让出 CPU 并暂时中止自己的执行，进入阻塞状态。阻塞时它不能进入排队队列，只有当引起阻塞的原因被消除时，线程才可以转入就绪状态，重新进到线程队列中排队等待 CPU 资源，以便从原来终止处开始继续运行。  （5）死亡：处于死亡状态的线程不具有继续运行的能力。线程死亡的原因有二：一个是正常运行的线程完成了它的全部工作，即执行完了 run( ) 方法的最后一个语句并退出；另一个是线程被提前强制性地终止。  **（二）线程间同步**  前面所提到的线程都是独立的，而且异步执行，也就是说每个线程都包含了运行时所需要的数据或方法，而不需要外部的资源或方法，也不必关心其他线程的状态或行为。但是经常有一些同时运行的线程需要共享数据，此时就需考虑其他线程的状态和行为，否则就不能保证程序运行结果的正确性。下面的例子说明了此问题。  class Stack  {  int idx = 0; // 堆栈指针的初始值为 0  char[ ] data = new char[6]; // 堆栈有 6 个字符的空间  public void push(char c) // 压栈操作  {  data[idx] = c; // 数据入栈  idx++; // 指针向上移动一位  }  public char pop() // 出栈操作  {  idx--; // 指针向下移动一位  return data[idx]; // 数据出栈  }  }  两个线程 A 和 B 在同时使用 Stack 的同一个实例对象，A 正在往堆栈里 push 一个数据，B 则要从堆栈中 pop 一个数据。由于线程 A 和 B 在对 Stack 对象的操作上的不完整性，将会导致操作的失败，具体过程如下所示。  （1）操作之前：  data = | p | q | | | | | idx = 2  （2）A 执行 push 中的第一个语句，将 r 推入堆栈；  data = | p | q | r | | | | idx = 2  （3）A 还未执行 idx+ + 语句，A 的执行被 B 中断，B 执行 pop 方法，返回 q：  data = | p | q | r | | | | idx = 1  （4）A 继续执行 push 的第二个语句：  data = | p | q | r | | , | | idx = 2  最后的结果相当于 r 没有入栈。产生这种问题的原因在于对共享数据访问操作的不完整性。为解决操作的不完整性问题，在 Java 语言中，引入了对象互斥锁的概念来保证共享数据操作的完整性。每个对象都对应于一个可称为“互斥锁”的标记，这个标记用来保证在任意时刻，只能有一个线程访问该对象。关键字 synchronized 与对象的互斥锁联系。当某个对象用 synchronized 修饰时，表明该对象在任意时刻只能由一个线程访问。  public void push(char c)  {  synchronized(this) //this 表示 Stack 的当前对象  {  data[idx] = c;  idx++;  }  }  public char pop()  {  synchronized(this) //this 表示 Stack 的当前对象  {  idx--;  return data[idx];  }  }  synchronized 除了像上面讲的放在对象前面限制一段代码的执行外，还可以放在方法声明中，表示整个方法为同步方法。  public synchronized void push(char c){……}  注意：如果 synchronized 用在类声明中，则表明该类中的所有方法都是 synchronized 的。我们把系统中使用某类资源的线程称为消费者，产生或释放同类资源的线程称为生产者。在下面的 Java 的应用程序中，生产者线程向文件中写数据，消费者从文件中读数据，这样，在这个程序中同时运行的两个线程共享同一个文件资源。通过下面的例子，我们来了解怎样使它们同步。  import java.lang.Math;  class SyncStack // 同步堆栈类  {  private int index = 0; // 堆栈指针初始值为 0  private char []buffer = new char[6]; // 堆栈有 6 个字符的空间  public synchronized void push(char c) // 加上互斥锁  {  while(index==buffer.length) // 堆栈已满 , 不能压栈  {  try  {  this.wait(); // 等待 , 直到有数据出栈  }  catch(InterruptedException e){}  }  this.notify(); // 通知其他线程把数据出栈  buffer[index] = c; // 数据入栈  index++; // 指针向上移动  }  public synchronized char pop() // 加上互斥锁  {  while(index==0) // 堆栈无数据 , 不能出栈  {  try  {  this.wait(); // 等待其他线程把数据入栈  }  catch(InterruptedException e){}  }  this.notify(); // 通知其他线程入栈  index--; // 指针向下移动  return buffer[index]; // 数据出栈  }  }  class Producer implements Runnable // 生产者类  {  SyncStack theStack; // 生产者类生成的字母都保存到同步堆栈中  public Producer(SyncStack s)  {  theStack = s;  }  public void run()  {  char c;  for(int i = 0; i<20; i++)  {  c = (char)(Math.random()\*26+'A'); // 随机产生 20 个字符  theStack.push(c); // 把字符入栈  System.out.println("Produced: "+c); // 打印字符  try  {  Thread.sleep((int)(Math.random()\*1000)); // 每产生一个字符线程就睡眠  }  catch(InterruptedException e){}  }  }  }  class Consumer implements Runnable // 消费者类  {  SyncStack theStack; // 消费者类获得的字符都来自同步堆栈  public Consumer(SyncStack s)  {  theStack = s;  }  public void run()  {  char c;  for(int i = 0;i<20;i++)  {  c = theStack.pop(); // 从堆栈中读取字符  System.out.println("Consumed: "+c); // 打印字符  try  {  Thread.sleep((int)(Math.random()\*1000)); // 每读取一个字符线程就睡眠  }  catch(InterruptedException e){}  }  }  }  public class SyncTest  {  public static void main(String args[])  {  SyncStack stack = new SyncStack(); // 消费者类对象和生产者类对象所操作  的是同一个同步堆栈  Runnable source = new Producer(stack);  Runnable sink = new Consumer(stack);  Thread t1 = new Thread(source); // 线程实例化  Thread t2 = new Thread(sink); // 线程实例化  t1.start(); // 线程启动  t2.start(); // 线程启动  }  }  程序运行结果如图 5−3 所示。    类 Producer 是生产者模型，其中的 run( ) 方法中定义了生产者线程所做的操作，循环调用 push( ) 方法，将生产的 20 个字母送入堆栈中，每次执行完 push 操作后，调用sleep( ) 方法睡眠一段随机时间，以给其他线程执行的机会。类 Consumer 是消费者模型，循环调用 pop( ) 方法，从堆栈中取出一个数据，一共取 20 次，每次执行完 pop 操作后，调用 sleep( ) 方法睡眠一段随机时间，以给其他线程执行的机会。  程序执行结果：Produced:V  Consumed:V  Produced:E  Consumed:E  Produced:P  Produced:L  ......  Consumed:L  Consumed:P  在上述的例子中，通过运用 wait( ) 和 notify( ) 方法来实现线程的同步，在同步中还会用到 notifyAll( ) 方法。一般来说，每个共享对象的互斥锁存在两个队列，一个是锁等待队列，另一个是锁申请队列，锁申请队列中的第一个线程可以对该共享对象进行操作，而锁等待队列中的线程在某些情况下将移入到锁申请队列。  **任务实施**  /\* 源程序位于文件夹 Ex0501/  import java.applet.\*;  import java.awt.\*;  class Animal extends Thread  {  String name = null;  long sleeptime = 0;  long speed1 = 10;  long speed2 = 5;  long distance1 = 0;  long distance2 = 0;  long length = 1000;  long asleeptime = 1000;  List list = null;  public Animal(String name, long sleeptime, List list)  {  this.name = name;  this.sleeptime = sleeptime;  this.list = list;  }  public void run()  {  while(name.equals(" 兔子 "))  {  if(distance1<length)  {  if(distance1!=800)  {  distance1 = distance1+sleeptime\*speed1;  try  {  sleep(sleeptime);  }  catch(Exception e)  {  System.out.println("Error");  }  list.addItem(name+" 跑了 "+distance1+" 米 !");  }  else  {  try  {  list.addItem(name+" 开始睡觉 !");  sleep(asleeptime);  list.addItem(name+" 睡醒后又开始跑了 !");  distance1 = distance1+sleeptime\*speed1;  list.addItem(name+" 跑了 "+distance1+" 米 !");  }  catch(Exception e)  {  Sytem.out.println("Error");  }  }  }  else  {  list.addItem(name+" 到达终点 !");  this.stop();  }  }  while(name.equals(" 乌龟 "))  {  if(distance2<length)  {  distance2 = distance2+sleeptime\*speed2;  try  {  this.sleep(sleeptime);  }  catch(Exception e){}  list.addItem(name+" 跑了 "+distance2+" 米 !");  }  else  {  list.addItem(name+" 到达终点 !");  this.stop();  }  }  public class Rabbit extends Applet  {  public void init()  {  setLayout(new BorderLayout());  List list = new List();  add("Center",list);  Animal rabbit = new Animal(" 兔子 ",10,list);  rabbit.start();  Animal tortoise = new Animal(" 乌龟 ",10,list);  tortoise.start();  }  }  程序运行结果如图 5−4 所示。    本例中启动了两个线程：rabbit 和tortoise。大家可以从运行结果中看到，两个线程并不是严格地交替运行或按先后次序运行，这是由于线程在操作系统中的运行特征所决定的。  **【学生】**思考、讨论。 | **教师通过龟兔赛跑（三）展示，让学生了解龟兔赛跑（三）的基本理论知识。** |
| **课堂小结**  （3min） | 【**教师**】**回顾和总结本节课的知识点。**  **这节课我们一起学习了龟兔赛跑（三），让学生了解锁申请队列中的第一个线程可以对该共享对象进行操作。** | 通过对所学知识的回顾，培养学生的归纳总结能力 |
| **作业布置**（2min） | 【**教师**】**布置课后作业**  **简述线程的生命周期。** | 通过课后练习，使学生巩固所学新知识 |
| **知识讲解**（45min） | **【教师】**展示思考与练习  **一、简答题**  （1）什么是线程？它和进程有什么区别？适用方向是什么？  （2）Java 的线程是如何实现的？  （3）Java 的线程是如何调度的？  （4）Java 中的多线程有什么特点？  **二、选择题**  （1）线程有（　　）种状态。  A. 4 B. 5 C. 6 D. 7  （2）下列选项中（　　）反映了 Java 程序并行机制的特点。  A. 安全性 B. 多线程 C. 跨平台 D. 可移植  （3）在 Thread 类中能启动线程体的方法是（　　）。  A. start( ) B. resume( ) C. init( ) D. run( )  （4）在 Thread 类中能运行线程体的方法是（　　）。  A. start( ) B. resume( ) C. init( ) D. run( )  （5）创建一个线程对象后，只有通过调用线程类的（　　）方法，才能将它启动。  A. sleep( ) B. start( ) C. wait( ) D. stop( )  （6）定义线程类时，可以通过实现接口 Runnable 来实现，必须实现的方法是  （　　）。  A. run( ) B. resume( ) C. init( ) D. start( )  （7）Java 语言中不能够使线程进入堵塞状态的方法是（　　）。  A. sleep( ) B. suspend( ) C. wait( ) D. stop( )  **实训任务**  （1）创建两个线程，每个线程打印出线程名字后再睡眠，给其他线程以执行的机会，主线程也要打印出线程名字后再睡眠，每个线程前后共睡眠 5 次。要求分别采用从Thread 中继承和实现 Runnable 接口两种方式来实现程序（即写两个程序）。  （2）编写三个线程分别显示各自的运行时间，第一个线程每隔 1 秒钟运行一次，第二个线程每隔 5 秒钟运行一次，第三个线程每隔 10 秒钟运行一次。  **【学生】**思考、讨论。 | **教师通过思考与练习展示，让学生了解思考与练习的基本理论知识。** |
| **课堂小结**  （3min） | 【**教师**】**回顾和总结本节课的知识点。**  **这节课我们一起学习了思考与练习，让学生可以更加巩固所学知识。** | 通过对所学知识的回顾，培养学生的归纳总结能力 |
| **作业布置**（2min） | 【**教师**】**布置课后作业**  **同步和互斥的原理是如何实现的？** | 通过课后练习，使学生巩固所学新知识 |
| **教学反思** | 在教学方面，对教材中的重点和难点内容，做到心中有数，并经常了解学生的态度，知识基础，学习方法，增强教学的针对性，从而选择适当的教学方法、手段和现代教学媒体，提高学生学习的主动性。 | |