## Week09讲义

线程

### 一 回顾

进程---正在执行的程序

线程--进程的执行线路，可以使一条也可以是多条

线程的创建。方法1

1. 写一个类继承Thread类
2. 重写父类Thread的run（）方法，在run方法中放线程要做的操作
3. 创建线程类对象
4. 用这个线程对象的start方法启动线程。

Start方法的功能1启动线程2调用线程的run方法

### 二 例题

多线程买票

/\*

\* 创建线程，方法1

\* 多个线程买票

\*/

//1写一个类继承Thread

class SellTicket extends Thread{

static int tickets=10;//一共100张票

public void run(){

//2重写run方法，线程要做的操作。买票。

while (tickets>0) {

System.out.println(Thread.currentThread().getName()+" sell no."+tickets--);

}

}

}

public class TestSellTicket {

public static void main(String[] args) {

// TODO Auto-generated method stub

//3创建线程类SellTicket类的对象

SellTicket t1=new SellTicket();

SellTicket t2=new SellTicket();

SellTicket t3=new SellTicket();

//4启动线程

t1.start();

t2.start();

t3.start();

}

}

说明：

1. Thread.currentThread().getName()

### 三 创建线程的另一种方法

步骤：

1. 写一个类，实现Runnable接口
2. 接口中有一个抽象方法run，实现这个run方法，把线程要做的操作放在run方法
3. 创建类的实例
4. 创建Thread线程对象，将类的实例作为参数传递给Thread构造函数
5. 启动线程

例题

/\*

\* 多线程买票

\* 方法2

\*/

//1类实现Runnable接口

class SellTicket2 implements Runnable {

int tickets = 10;// 100张票

public void run() {

// TODO Auto-generated method stub

// 2重写run方法，线程要做的操作。买票。

while (tickets > 0) {

System.out.println(Thread.currentThread().getName() + " sell no."

+ tickets--);

}

}

}

public class TestSellTicket2 {

public static void main(String[] args) {

// TODO Auto-generated method stub

//3创建Sellticket2类的实例r1

SellTicket2 r1=new SellTicket2();

//4创建线程类对象，将r1作为参数传递给Thread

Thread t1=new Thread(r1);

Thread t2=new Thread(r1);

Thread t3=new Thread(r1);

//启动线程

t1.start();

t2.start();

t3.start();

}

}

### 四、2种方法的区别

方法1，因为采用继承的方法去继承Thread类，有一些问题

Class Student extends Thread{}

Student类继承Person类

Class Student extends Person{}

不能采用方法1，只能用方法2

Class Student extends Person implements Runnable{}

### 五 多线程的安全问题

1产生的原因

多个线程，对同一个资源进行操作。

某些操作步骤必须由同一个线程连续的执行，不能被其他线程打断。

例如：自动取款机-------资源

同学1--线程

同学2--线程

同学3--线程

取款步骤：

1插卡

2输密码

3取钱

4取卡

2解决方法

同步操作

1. 同步代码块

同一时刻，只能由一个线程连续执行的代码块，syntronized关键字，需要的参数是Object类型，作为同步代码块的锁，注意，多个线程必须使用同一把锁

synchronized (锁) {

System.out.println(Thread.currentThread().getName()

+ " sell no." + tickets);

try {

Thread.sleep(100);

} catch (InterruptedException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

tickets--;

}

1. 同步函数

如果一个函数全部代码都需要同步，声明为同步函数。

public syntronized void add(int n) {

System.out.println(Thread.currentThread().getName());

count=count+n;

try {

Thread.sleep(100);

} catch (InterruptedException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

System.out.println("班费="+count);

}

注意：

同步对象=锁，持有锁的线程可以执行同步的代码，没有锁的线程，即使获得cpu的执行权，也不能执行同步代码，（死锁）。同步函数一样有锁，调用这个方法的对象就是锁（this）。

六 守护线程（后台线程）

当其他线程（非守护线程）结束后，守护线程自动结束

/\*

\* 模擬學校答疑，

\* 教師

\* 学生

\* 教室里面 教师要持续90分钟，学生持续时间不一定10,5,3

\*/

class DAYI implements Runnable{

int time;

public DAYI(int t){

time=t;

}

public void run() {

// TODO Auto-generated method stub

//持续时间一分一分的减少，显示

for (int i = time; i >0; i--) {

System.out.println(Thread.currentThread().getName()+"---"+i);

}

System.out.println(Thread.currentThread().getName()+"----------end");

}

}

public class TestDeamon {

/\*\*

\* @param args

\*/

public static void main(String[] args) {

// TODO Auto-generated method stub

DAYI teacher=new DAYI(900);

DAYI stu=new DAYI(10);

DAYI stu2=new DAYI(3);

Thread t1=new Thread(teacher,"teacher ");

Thread t2=new Thread(stu,"student1");

Thread t3=new Thread(stu2,"student2");

//设置t1线程为守护线程

t1.setDaemon(true);

//t2t3设置成用户线程

t2.setDaemon(false);

t3.setDaemon(false);

t1.start();

t2.start();

t3.start();

System.out.println("main thread end.");

}

}