# 多线程

## 1.单进程处理，dos ，只要有病毒，就会立刻又反应，这个就是单进程

## 2.对于window来说，就是多进程的，出现病毒也不会发现，进程中有多个线程，进程消失，线程也消失

# 两种方式实现多线程

# 1.继承Tread 类 在子类中重写 run()方法

## 1.创建一个线程的操作类，需要重写 run 方法

**public** **class** MyThred **extends** Thread{

//重写 run 方法

@Override

**public** **void** run() {

**super**.run();

}

}

## 2.创建一个构造方法

**public** **class** MyThred **extends** Thread{

String name;

**public** MyThred(String name) {

**super**();

**this**.name = name;

}

//重写 run 方法

@Override

**public** **void** run() {

// **TODO** Auto-generated method stub

**super**.run();

}

}

## 3.run 方法中执行循环

**for**(**int** i=0;i<10;i++) //表示循环 10 次

{

System.*out*.println("name:"+name+i);

}

## 4.这样就是完成了一个线程的操作类，就该启动这个线程了

//这个就是一个多线程的操作类 ，下面就该启动线程了

**public** **class** MyThred **extends** Thread{

String name;

**public** MyThred(String name) {

**super**();

**this**.name = name;

}

//重写 run 方法

@Override

**public** **void** run() {

// **TODO** Auto-generated method stub

**super**.run();

**for**(**int** i=0;i<10;i++) //表示循环 10 次

{

System.*out*.println("name:"+name+i);

}

}

}

## 5.1 进行测试，4.1先使用run调用

**public** **class** ThreadDemo\_1

{

**public** **static** **void** main(String args[])

{

MyThred myThred1 = **new** MyThred("zhang");

MyThred myThred2 = **new** MyThred("gao");

myThred1.run();

System.*out*.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*8");

myThred2. run ();

}

}

## 5.2测试结果，此时的执行发现非常有规律，并没有实现交互的运行，先是前面的

name:zhang0

name:zhang1

**name:zhang2**

**name:zhang3**

**name:zhang4**

**name:zhang5**

**name:zhang6**

**name:zhang7**

**name:zhang8**

name:zhang9

\*\*\*\*\*\*\*\*\*\*\*\*\*\*8

name:gao0

name:gao1

name:gao2

name:gao3

name:gao4

name:gao5

name:gao6

name:gao7

name:gao8

name:gao9

## 6使用start 测试调用，\*\*\*\*\*\*\*\*\*\*\*\*\*\*8

myThred1.start();

System.*out*.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*8");

myThred2.start();

## 6.2测试结果，发现是交互运行的

\*\*\*\*\*\*\*\*\*\*\*\*\*\*8

name:zhang0

name:zhang1

name:zhang2

name:zhang3

name:zhang4

name:zhang5

name:gao0

name:zhang6

name:gao1

name:zhang7

name:gao2

name:zhang8

name:gao3

name:zhang9

name:gao4

name:gao5

name:gao6

name:gao7

name:gao8

name:gao9

# 1.2.结果分析，

## 1.main （）方法本身也是线程， 所以上面的线程是3，但是main方法比较简单，一个时间片就会运行完毕，所以他的执行结果先显示出来 \*\*\*\*\*\*\*\*\*\*\*\*\*\*8（对于简单的），一个线程中至少有两个线程，一个是main一个是关闭的线程

## 2.但是线程必须被系统调度获得cpu权才可以运行，所以线程不是由程序本身控制的，而是由系统调度决定的。所以上面的是一个可能的运行结果。

## 3.有限制，一个类只能继承一个父类

# 2.通过runnable 接口，然后重写接口中的run方法

**public** **class** MyThred **implements** Runnable{

String name;

**public** MyThred(String name) {

**super**();

**this**.name = name;

}

@Override

**public** **void** run() {

// **TODO** Auto-generated method stub

**for**(**int** i=0;i<10;i++) //表示循环 10 次

{

System.*out*.println("name:"+name+i);

}

}

}

## 2.继承接口Runnable 的类没有start方法了，这个时候我们，使用下面的方法可以启动线程

**public** **class** ThreadDemo\_1

{

**public** **static** **void** main(String args[])

{

MyThred myThred1 = **new** MyThred("zhang");

Thread thread1 = **new** Thread(myThred1);

Thread thread2 = **new** Thread(myThred1);

//调用线程体

thread1.start();

System.*out*.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*8");

thread2.start();

}

}

## 3.运行，发现也是交替进行

\*\*\*\*\*\*\*\*\*\*\*\*\*\*8

name:zhang0

name:zhang1

name:zhang2

name:zhang3

name:zhang4

name:zhang5

name:gao0

name:zhang6

name:gao1

name:gao2

name:zhang7

name:zhang8

name:gao3

name:gao4

name:gao5

name:gao6

name:gao7

name:gao8

name:gao9

name:zhang9

# 3.两种方法的区别

## 1.程序中只要是多线程，都是实现runnable 接口为正统操作，

## 2.避免了单继承的局限，一个类可以实现多个接口，

## 3.适合于资源的共享

# 4.以售票来看看二者的好处,一共有5张

# 1.继承Thread

**public** **class** MyThred **extends** Thread{

**private** **int** tickets=20;

//重写 run 方法

@Override

**public** **void** run() {

// **TODO** Auto-generated method stub

**super**.run();

**while**(**true**)

{

**if**(tickets>0)

System.*out*.println(Thread.*currentThread*().getName()+"出售票"+tickets--);

}

}

}

## 1.2多线程操作

**public** **class** ThreadDemo\_1

{

**public** **static** **void** main(String args[])

{

MyThred myThred1 = **new** MyThred();

MyThred myThred2 = **new** MyThred();

myThred1.start();

System.*out*.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*8");

myThred2.start();

}

}

## 1.3.运行结果，发现和我们的实际不一样，出现了10张

\*\*\*\*\*\*\*\*\*\*\*\*\*\*8

Thread-0出售票5

Thread-0出售票4

Thread-0出售票3

Thread-0出售票2

Thread-0出售票1

Thread-1出售票5

Thread-1出售票4

Thread-1出售票3

Thread-1出售票2

Thread-1出售票1

# 2.Runnable接口

## 1.线程开启

**public** **class** ThreadDemo\_1

{

**public** **static** **void** main(String args[])

{

InterRunnable interRunnable = **new** InterRunnable();

Thread thread = **new** Thread(interRunnable);

Thread thread2 = **new** Thread(interRunnable);

thread.start();

thread2.start();

}

}

## 1.2测试，这样就是两个线程解决了资源共享

Thread-1出售票4

Thread-0出售票5

Thread-0出售票3

Thread-0出售票2

Thread-0出售票1

# 5.线程的操作的一些方法（Thread中的方法）

## 1.取得线程的名字

**public** **class** InterRunnable **implements** Runnable{

//重写 run 方法

@Override

**public** **void** run() {

// **TODO** Auto-generated method stub

**for**(**int** i=0;i<10;i++) //表示循环 10 次

{

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

}

}

}

InterRunnable extendsThread = **new** InterRunnable();

Thread thread = **new** Thread(extendsThread, "线程1");

Thread thread2 = **new** Thread(extendsThread, "线程2");

Thread thread3 = **new** Thread(extendsThread, "线程3");

thread.start();

thread2.start();

thread3.start();

}

### 1.2运行结果

线程1

线程3

线程2

线程3

线程1

线程3

线程2

线程3

线程1

线程3

线程2

线程3

线程3

线程3

线程3

## 2.main线程出现

**public** **static** **void** main(String[] args)

{

InterRunnable nterRunnable = **new** InterRunnable();

Thread thread = **new** Thread(nterRunnable, "线程1");

Thread thread2 = **new** Thread(nterRunnable, "线程2");

Thread thread3 = **new** Thread(nterRunnable, "线程3");

thread.start();

thread2.start();

thread3.start();

nterRunnable.run();

}

**public** **class** InterRunnable **implements** Runnable{

//重写 run 方法

@Override

**public** **void** run() {

// **TODO** Auto-generated method stub

**for**(**int** i=0;i<10;i++) //表示循环 10 次

{

System.*out*.println(Thread.*currentThread*().getName()+"线程正在运行");

}

}

### 2..2测试结果

main线程正在运行

线程2线程正在运行

线程1线程正在运行

main线程正在运行

main线程正在运行

线程1线程正在运行

main线程正在运行

线程1线程正在运行

main线程正在运行

线程1线程正在运行

main线程正在运行

线程1线程正在运行

main线程正在运行

## 3.休眠Thread.*sleep*(500);

**public** **class** SleepRunnable **implements** Runnable{

**public** **void** run() {

// **TODO** Auto-generated method stub

**for**(**int** i=0;i<10;i++) //表示循环 10 次

{

**try** {

//每个休眠500毫秒

Thread.*sleep*(500);

} **catch** (InterruptedException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

System.*out*.println(Thread.*currentThread*().getName()+"线程正在运行");

}

}

}

### 3.2测试

SleepRunnable sleepRunnable = **new** SleepRunnable();

Thread thread = **new** Thread(sleepRunnable, "线程1");

Thread thread2 = **new** Thread(sleepRunnable, "线程2");

Thread thread3 = **new** Thread(sleepRunnable, "线程3");

thread.start();

thread2.start();

thread3.start();

sleepRunnable.run();

### 3.3测试结果

#### 发现程序执行变慢，这个是 4 个三个出现的（上面是4个线程）

## 4.线程休眠中断（这样休眠就会出现异常）

**private** **static** **void** Interupt() {

InteruptRunnable interuptRunnable = **new** InteruptRunnable();

Thread thread = **new** Thread(interuptRunnable, "线程1");

thread.start();

**try** {

thread.*sleep*(200);

} **catch** (InterruptedException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

这里是main 线程，率先执行，的

System.*out*.println("这里的方法是中端");

thread.interrupt();

System.*out*.println("这里的方法是中端");

}

### 4.2 InteruptRunnable 线程

**public** **class** InteruptRunnable **implements** Runnable{

**public** **void** run() {

// **TODO** Auto-generated method stub

System.*out*.println(" 1. 进入线程");

**try** {

//每个休眠500毫秒//

System.*out*.println("2. 线程将要 休眠");

Thread.*sleep*(500);

System.*out*.println("3. 线程正在休眠");

} **catch** (InterruptedException e) {

// **TODO** Auto-generated catch block

System.*out*.println("4. 线程中断休眠");

**return**; //一，当线程休眠中断之后就会发生异常 旦有了异常,返回方法调用处

}

System.*out*.println(Thread.*currentThread*().getName()+"5. 线程正在运行");

}

}

### 4.3进行测试

1. 进入线程

2. 线程将要 休眠

这里的方法是中端

4. 线程中断休眠

这里的方法是中端 有了return 不会向下面走了

# 5.线程优先级别thread1.setPriority

InterRunnable myThred1 = **new** InterRunnable();

Thread thread1 = **new** Thread(myThred1,"线程1");

Thread thread2 = **new** Thread(myThred1,"线程2");

Thread thread3 = **new** Thread(myThred1,"线程3");

//调用线程体

thread1.setPriority(Thread.*NORM\_PRIORITY*);

thread2.setPriority(Thread.*MAX\_PRIORITY*);

thread3.setPriority(Thread.*MIN\_PRIORITY*);

thread1.start();

System.*out*.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*8");

thread2.start();

thread3.start();

}

## 5.1测试结果

\*\*\*\*\*\*\*\*\*\*\*\*\*\*8

线程2正在运行

线程2正在运行

线程2正在运行

线程2正在运行

线程2正在运行

线程1正在运行

线程1正在运行

线程1正在运行

线程1正在运行

线程1正在运行

线程3正在运行

线程3正在运行

线程3正在运行

线程3正在运行

线程3正在运行

## 5.2main的优先级，是普通的优先级

**public** **static** **void** main(String args[])

{

InterRunnable myThred1 = **new** InterRunnable();

Thread thread1 = **new** Thread(myThred1,"线程1");

Thread thread2 = **new** Thread(myThred1,"线程2");

Thread thread3 = **new** Thread(myThred1,"线程3");

//调用线程体

thread1.setPriority(Thread.*NORM\_PRIORITY*);

thread2.setPriority(Thread.*MAX\_PRIORITY*);

thread3.setPriority(Thread.*MIN\_PRIORITY*);

thread1.start();

System.*out*.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*8");

thread2.start();

thread3.start();

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

System.*out*.println(Thread.*MAX\_PRIORITY*+"max");

System.*out*.println(Thread.*MIN\_PRIORITY*+"min");

System.*out*.println(Thread.*NORM\_PRIORITY*+"norm");

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*8

线程1正在运行

线程1正在运行

线程1正在运行

线程1正在运行

线程1正在运行

main

线程2正在运行

10max

线程2正在运行

1min

线程2正在运行

5norm

线程2正在运行

线程2正在运行

线程3正在运行

线程3正在运行

线程3正在运行

线程3正在运行

线程3正在运行

## 5.3测试

6.