**Java多线程编程**

**Thread类**

在java.lang包中，直接继承自Object类，并实现了Runnable接口。其封装了Java程序中一个线程（对象）需要的所有属性和方法。

从Thread类派生一个子类并创建该子类的对象，就可以产生一个新的线程。该子类应重写Thread类的run()方法，在run()方法中写入需要在新线程中执行的代码。该子类对象需调用start()方法来启动，启动后新线程将自动执行自己的run()方法。原线程将同时继续向下执行。

**Runnable接口**

如果程序员需要继承其他类（非Thread类），又要使用当前类实现多线程，则可以通过Runnable接口来实现。

该接口只包含一个无参的抽象方法run。

**Thread类常用构造方法**：

* public Thread()

//按默认名Thread-n,n从0开始构造新线程对象

* public Thread(Runnable target)

//以一个实现Runnable接口的类的对象为参数，按默认名Thread-n,n从0开始构造新线程对象

* public Thread(String name)

//按指定线程名构造新线程对象

* public Thread(Runnable target ,String name)

//以一个实现Runnable接口的类的对象为参数，并同时指定线程名，构造新线程对象

**Thread类常用方法：**

* public static Thread currentThread()

//返回当前正在运行的线程对象

* public static void yield()

//使当前线程对象暂停，允许别的线程开始运行

* public static void sleep(long millis)

//使当前线程暂停运行指定的毫秒数

* public void start()

//启动线程，JVM将调用此线程的run方法，结果是将同时运行两个线程：当前线程和执行run方法的线程

* public void run()

//Thread的所有子类应该重写此方法，内容为该线程应执行的任务

* public final void stop()

//停止线程运行

* public void interrupt()

//打断此线程

* public final void join()

//将当前线程中加入调用join方法的线程A，直到A死亡才能继续执行当前线程

**例1** 在新线程中完成计算某个整数的阶乘（继承Thread类）。

**public** **class** FactThreadTest {

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

System.***out***.println("main thread starts");

FactThread thread=**new** FactThread(10);

thread.start();

**try**{

Thread.*sleep*(1);}

**catch**(Exception e){}

//若启动新线程后希望主线程多持续一会再结束，可在start语句后加上让当前线程暂停1ms的语句，删除该try…catch再运行程序，比较两次的运行结果有什么不同

System.***out***.println("main thread ends");

}

}

**class** FactThread **extends** Thread{

**private** **int** num;

**public** FactThread(**int** num){

**this**.num=num;

}

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

**int** i=num;

**long** s=1;

System.***out***.println("new thread starts");

**while** (i>0){

s\*=i--; }

System.***out***.println("the fact of"+num+" is "+s);

System.***out***.println("new thread ends");

}

}

**例2** 在新线程中完成计算某个整数的阶乘（实现Runnable接口）。

**public** **class** FactThreadTest {

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

System.***out***.println("main thread starts");

FactThread thread=**new** FactThread(10);

**new** Thread(thread).start();

//调用Thread类跟Runnable接口有关的构造方法

**try**{

Thread.*sleep*(1);}

**catch**(Exception e){}

System.***out***.println("main thread ends");

}

}

**class** FactThread **implements** Runnable{

**private** **int** num;

**public** FactThread(**int** num){

**this**.num=num;

}

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

**int** i=num;

**long** s=1;

System.***out***.println("new thread starts");

**while** (i>0){

s\*=i--; }

System.***out***.println("the fact of "+num+" is "+s);

System.***out***.println("new thread ends");

}

}

**例3** 创建3个新线程，每个线程睡眠一段时间（0~6s），然后结束。

**public** **class** ThreadSleepTest {

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

TestThread thread1=new TestThread("thread1");

TestThread thread2=new TestThread("thread2");

TestThread thread3=new TestThread("thread3");

System.***out***.println("Starting threads");

thread1.start();

thread2.start();

thread3.start();

System.***out***.println("main thread ends");}

}

**class** TestThread **extends Thread**{

**private** **int** sleepTime;

**public** TestThread(String name){

**super(name)**;

sleepTime=(int)(Math.random()\*6000);

}

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

try{

System.***out***.println(

getName()+"going to sleep for"+sleepTime);

Thread.sleep(sleepTime);}

catch (InterruptedException e){}

System.***out***.println(getName()+"finished");

}

}

**例4** 使用Runnable接口实现例3的功能。

**public** **class** ThreadSleepTest {

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

TestThread t1=new TestThread("");

TestThread t2=new TestThread("");

TestThread t3=new TestThread("");

System.***out***.println("Starting threads");

**new** Thread(t1, "Thread1").start();

**new** Thread(t2, "Thread2").start();

**new** Thread(t3, "Thread3").start();

//分别以3个对象为参数创建3个新线程，第2个参数为新线程命名，并启动之

System.***out***.println("main thread ends");}

}

**class** TestThread **implements** Runnable {

**private** **int** sleepTime;

**public** TestThread(){ //无参构造方法

sleepTime=(int)(Math.random()\*6000);//获得随机休眠毫秒数

}

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

try{

System.***out***.println(

Thread.currentThread().getName()+

"going to sleep for"+sleepTime);

//区别于例3，因为不是继承Thread类，因而必须先调用currentThread()方法

Thread.sleep(sleepTime);}

catch (InterruptedException e){}

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

"finished");//运行结束，给出提示信息

}

}

**线程间的数据共享**

使用Runnable接口可以轻松实现多个线程共享相同的数据，只要用同一个实现了Runnable接口的实例作为参数创建多个线程就可以了。

**例5**修改例4，只用一个Runnable类型的对象为参数创建3个新线程。

**public** **class** ShareSleepTest {

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

TestThread t=new TestThread();

//只创建一个实现Runnable接口类的对象

System.***out***.println("Starting threads");

TestThread thread2=new TestThread("");

TestThread thread3=new TestThread("");

System.***out***.println("Starting threads");

**new** Thread(t, "Thread1").start();

**new** Thread(t, "Thread2").start();

**new** Thread(t, "Thread3").start();

//只用一个Runnable类型的对象为参数创建3个新线程，分别命名并启动之

System.***out***.println("main thread ends");}

}

**class** TestThread **implements** Runnable{

**private** **int** sleepTime;

**public** TestThread(){

sleepTime=(int)(Math.random()\*6000);

}

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

try{

System.***out***.println(

Thread.currentThread().getName()+

"going to sleep for"+sleepTime);

Thread.sleep(sleepTime);}

catch (InterruptedException e){}

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

"finished");

}

}

因为是用一个Runnable类型的对象创建的3个新线程，这3个新线程就共享了这个对象的私有成员sleepTime，在本次运行中，3个线程休眠了相同的时间。

**例6** 用3个线程模拟3个售票窗口，总共出售200张票。

**public** **class** SellTicketsTest {

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

SellTickets t=new SellTickets();

**new** Thread(t).start();

**new** Thread(t).start();

**new** Thread(t).start();

}

}

**class** SellTickets **implements** Runnable{

**private** **int** tickets=200;

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

**while** (tickets >0){

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

"is sell ticket"+ tickets--);}

}

}