第一章 线程（上）

1.1 线程的基本概念

一个线程是一个程序内部的顺序控制流。

多进程是指操作系统中，能同时运行多个任务（程序）。

多线程是在同一个应用程序中，有多个顺序流同时进行。

1.进程和线程的比较

（1）每个进程都有独立的代码和数据空间（进程上下文），进程的切换开销大。

（2）线程是轻量的进程，同一类线程共享代码和数据空间，每个线程都有独立的运行栈和程序计数器（PC），线程切换开销小。

2.线程的概念模型

线程在运行中，必须得到 CPU 资源。在 Java 中，就有一个类 java.lang.Thread 来模拟一个 CPU。CPU 所执行的代码传递给 Thread 两类，CPU 所处理的数据，也传递给 Thread 类。

（1）线程体

java 的线程是通过 java.lang.Thread 来实现的，每个线程都是通过某个特定的 Thread 对象的方法 run()来完成其操作的，方法 run()称为线程体。

（2）构造线程的两种方法

①定义一个线程类，它继承类 Thread 并重写其中的方法 run()；

②提供一个实现接口 Runnable 的类作为线程的目标对象，在初始化一个 Thread 类或者 Thread 子类的线程对象时，把目标对象传递给这个线程实例，由该目标对象提供线程体 run()。

public Thread(ThreadGroup group, Runnable target, String name);

1.2 通过 Thread 类创建线程

上一节讲了线程有两种创建方法，本节将讲解第一种创建方法。

（1）Thread 类

Thread 类直接继承了 Object 类，并实现了 Runnable 接口，位于 java.lang 包中，它封装了线程对象需要的属性和方法。

从 Thread 类派生一个子类，并创建子类的对象。子类应该重写 Thread 类中的 run 方法，写入需要在新线程中执行的语句段。调用 start 方法来启动新线程，自动进入 run 方法。

例：在新线程中完成计算某个整数的阶乘

代码：

```
public class FactorialThreadTester{
    public static void main(String[] args){
        System.out.println("main thread starts");
        FactorialThread thread   = new FactorialThread(10);
```

```
            thread.start();//将自动进入 run()方法
            System.out.println("main thread ends");
        }
}
class FactorialThread extends Thread{
        private int num;
        public FactorialThread(int num){
            this.num = num;}
        public void run(){
            int i =num;
            int result = 1;
            System.out.println("new thread started");
            while(i>0)
            {result =result*i; i =i-1;}
        System.out.println("The factorial of " + num +
" is " + result + "and end.");
        }
}
```

运行结果：

```
main thread starts
main thread ends
new thread started
The factorial of 10 is 3628800 and end
```

程序分析：

说明主线程就是一个线程，主线程执行完成后，新线程才开始执行并且执行完；main 方法调用 thread.start()方法启动新线程并不等待其 run 方法返回就继续运行，线程的 run 方法在一边独自运行，不影响原来的 main 方法的运行。

1.3 线程的休眠

例：1.2 中程序启动新线程后希望主线程多持续一会儿再结束，可以在 start 语句后加上让当前线程休眠 1 毫秒的语句：

代码：

```
public class FactorialThreadTester{
        public static void main(String[] args){
            System.out.println("main thread starts");
            FactorialThread thread    = new FactorialThread(10);
            thread.start();//将自动进入 run()方法
            try(Thread.sleep(1);}//添加休眠 1 毫秒
            catch(Exception e){};
            System.out.println("main thread ends");
        }
}
```

运行结果：

main thread starts
new thread started
The factorial of 10 is 3628800 and end
main thread ends

程序分析：

说明主线程就是一个线程，主线程执行开始后，进行休眠 1 秒钟，与此同时新线程启动并运行，运行完成后，等待之前的 1 毫秒结束，那么主线程继续开始执行。

1.4 Thread 类详解

Thread 类常用的 API 方法如图 1，图 2，图 3 所示。

图 1

图 2

图 3

注意：

1.调用了 start()方法并不代表程序立即启动，其是否启动由线程调度器来决定。

1.5 通过 Runnable 接口创建线程

本节来介绍第二种创建线程的方法。

1.Runnable 接口

（1）它只有一个方法，就是 run()方法。

（2）Thread 类实现了 Runnable 接口；

（3）便于多个线程共享资源；

（4）Java 不支持多继承，如果已经继承了某个基类，因此便需要 Runnable 接口来生成多线程；

（5）以实现 Runnable 的对象为参数建立新的线程；

（6）start 方法启动线程就会运行 run()方法。

例：使用 Runnable 接口实现上面单线程的例子

代码：

```
public class FactorialThreadTester{
    public static void main(String[] args){
        System.out.println("main thread starts");
        FactorialThread t   = new FactorialThread(10);//实现 Runnable 类
        new Thread(t).start();//运行 FactorialThread 的 run()方法
        System.out.println("new thread started, main thread ends ");
    }
}
```

```
class FactorialThread implements Runnable{
    private int num;
    public FactorialThread(int num){
        this.num = num;}
    public void run(){
        int i =num;
        int result = 1;
        while(i>0)
        {result =result*i; i =i-1;}
    System.out.println("The factorial of " + num + " is " + result + "and end.");
    }
}
```

运行结果：

同上

例：使用 Runnable 接口实现多线程的例子

代码：

```
public class ThreadSleepTester{
    public static void main(String[] args){
        TestThread thread1 = new TestThread();
        TestThread thread2 = new TestThread();
        TestThread thread3 = new TestThread();
        System.out.println("Starting threads");

        new Thread(thread1,"thread1").start();//运行 FactorialThread 的 run()方法
        new Thread(thread2,"thread2").start();
        new Thread(thread3,"thread3").start();
        System.out.println("Threads started, main thread ends\n");
    }
}

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);
        }
        catch(InterruptedException exception){}
        System.out.println(Thread.currentThread().getName()   + "finished.");
    }
}
```

运行结果（随机结果，不固定）:

```
Starting threads
Threads started, main thread ends

thread1 going to sleep for 4325
thread1finished.
thread2 going to sleep for 920
thread2finished.
thread3 going to sleep for 1207
thread3finished.
```

两种线程构造方式的比较：

1.使用 Runnable 接口，可以将 CPU、代码和数据分开，形成清晰的模型；还可以从其它类继承。

2.直接继承 Thread 类：编写简单、直接继承，重写 run 方法，不能再从其他类继承。

1.6 线程内部的数据共享

同一个线程类可以实例化出很多线程，本节将介绍如何实现线程内部的数据共享。

用同一个实现了 Runnable 接口的对象作为参数创建多个线程，多个线程可以共享同一对象中的相同的数据。

例：只用一个 Runnable 类型的对象为参数创建 3 个新线程

代码：

```
public class ShareTargetTester{
    public static void main(String[] args){
        TestThread threadobj = new TestThread();
        System.out.println("Starting threads");

        new Thread(threadobj,"thread1").start();//运行 FactorialThread 的 run()方法
        new Thread(threadobj,"thread2").start();
        new Thread(threadobj,"thread3").start();
        System.out.println("Threads started, main thread ends\n");
    }
}

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 exception){}
            System.out.println(Thread.currentThread().getName()   + "finished.");
        }
}
```

运行结果（数字随机）：

```
Starting threads
Threads started, main thread ends

thread1 going to sleep for 4811
thread2 going to sleep for 4811
thread3 going to sleep for 4811
(中间有暂停)
thread1finished.
thread3finished.
thread2finished.
```

结果分析：

用一个 Runnable 类型对象创建的 3 个新线程，这三个线程就共享了这个对象的私有成员 sleepTime，因此三者休眠时间相同。

我们需要明白，独立且同时运行的线程有时候需要共享一些数据并且考虑到彼此的状态和动作。

例：用三个线程模拟三个售票窗口，总共售出 200 张票：

-用 3 个线程模仿 3 个售票口的售票行为；

-这 3 个线程应该共享 200 张票的数据。

代码：

```
public class SellTicketsTester{
    public static void main(String[] args){
        SellTickets t = new SellTickets();
        new Thread(t).start();
        new Thread(t).start();
        new Thread(t).start();//构造 3 个线程并启动
    }
}

class SellTickets implements Runnable{
    private int tickets = 200;
    public void run()
    {
        while(tickets > 0){
            System.outprint.ln(Thread.currentThread().getName()
            +" is selling ticket " +tickets--);
        }
    }
```

```
}
```

运行结果：

```
Thread-1 is selling ticket 20
Thread-0 is selling ticket 19
Thread-1 is selling ticket 18
.....
Thread-1 is selling ticket 1
Thread-0 is selling ticket 0
```

说明：

在这个例子中，创建了 3 个线程，每个线程调用的是同一个 SellTickets 对象中的 run()方法，访问的是同一个对象中的变量(tickets)。

如果是通过创建 Thread 类的子类来模拟的话，再创建 3 个新线程，则每个线程都会有各自的方法和变量，虽然方法相同，但是变量却是各自有各自的 200 张票，因而结果变成各自卖出的票数，和要求不符合。