Java中Runnable和Thread的区别

在java中可有两种方式实现多线程，一种是继承Thread类，一种是实现Runnable接口；Thread类是在java.lang包中定义的。一个类只要继承了Thread类同时覆写了本类中的run()方法就可以实现多线程操作了，但是一个类只能继承一个父类，这是此方法的局限。

作者：wwww1988600 来源：wwww1988600的博客 | 2012-03-01 14:04

在java中可有两种方式实现多线程，一种是继承Thread类，一种是实现Runnable接口；Thread类是在java.lang包中定义的。一个类只要继承了Thread类同时覆写了本类中的run()方法就可以实现多线程操作了，但是一个类只能继承一个父类，这是此方法的局限。

下面看例子：

```
1. package org.thread.demo;
2. class MyThread extends Thread{
3. private String name;
4. public MyThread(String name) {
5. super();
6. this.name = name;
7. }
8. public void run(){
9. for(int i=0;i<10;i++){
10. System.out.println("线程开始："+this.name+",i="+i);
11. }
12. }
13. }
14. package org.thread.demo;
15. public class ThreadDemo01 {
16. public static void main(String[] args) {
17. MyThread mt1=new MyThread("线程a");
18. MyThread mt2=new MyThread("线程b");
19. mt1.run();
20. mt2.run();
21. }
22. }
```

但是，此时结果很有规律，先第一个对象执行，然后第二个对象执行，并没有相互运行。在JDK的文档中可以发现，一旦调用start()方法，则会通过JVM找到run()方法。下面启动start()方法启动线程：

```
1. package org.thread.demo;
2. public class ThreadDemo01 {
3. public static void main(String[] args) {
```

```
MyThread mt1=new MyThread("线程a");
MyThread mt2=new MyThread("线程b");
mt1.start();
mt2.start();
}
};
```

这样程序可以正常完成交互式运行。那么为啥非要使用start();方法启动多线程呢？

在JDK的安装路径下，src.zip是全部的java源程序，通过此代码找到Thread中的start()方法的定义，可以发现此方法中使用了private native void start0();其中native关键字表示可以调用操作系统的底层函数，那么这样的技术成为JNI技术（java Native Interface）

Runnable接口

在实际开发中一个多线程的操作很少使用Thread类，而是通过Runnable接口完成。

```
public interface Runnable{
public void run();
}
```

例子：

```
package org.runnable.demo;
class MyThread implements Runnable{
private String name;
public MyThread(String name) {
this.name = name;
}
public void run(){
for(int i=0;i<100;i++){
System.out.println("线程开始："+this.name+",i="+i);
}
}
};
```

但是在使用Runnable定义的子类中没有start()方法，只有Thread类中才有。此时观察Thread类，有一个构造方法：public Thread(Runnable targer)此构造方法接受Runnable的子类实例，也就是说可以通过Thread类来启动Runnable实现的多线程。（start()可以协调系统的资源）：

```
package org.runnable.demo;
import org.runnable.demo.MyThread;
```

```
public class ThreadDemo01 {
public static void main(String[] args) {
MyThread mt1=new MyThread("线程a");
MyThread mt2=new MyThread("线程b");
new Thread(mt1).start();
new Thread(mt2).start();
}
}
```

两种实现方式的区别和联系：

在程序开发中只要是多线程肯定永远以实现Runnable接口为主，因为实现Runnable接口相比继承Thread类有如下好处：

> 避免点继承的局限，一个类可以继承多个接口。
> 适合于资源的共享

以卖票程序为例，通过Thread类完成：

```
package org.demo.dff;
class MyThread extends Thread{
private int ticket=10;
public void run(){
for(int i=0;i<20;i++){
if(this.ticket>0){
System.out.println("卖票：ticket"+this.ticket--);
}
}
}
};
```

下面通过三个线程对象，同时卖票：

```
package org.demo.dff;
public class ThreadTicket {
public static void main(String[] args) {
MyThread mt1=new MyThread();
MyThread mt2=new MyThread();
MyThread mt3=new MyThread();
mt1.start();//每个线程都各卖了10张，共卖了30张票
mt2.start();//但实际只有10张票，每个线程都卖自己的票
mt3.start();//没有达到资源共享
}
}
```

如果用Runnable就可以实现资源共享，下面看例子：

```
package org.demo.runnable;
class MyThread implements Runnable{
private int ticket=10;
public void run(){
for(int i=0;i<20;i++){
if(this.ticket>0){
System.out.println("卖票：ticket"+this.ticket--);
}
}
}
}
package org.demo.runnable;
public class RunnableTicket {
public static void main(String[] args) {
MyThread mt=new MyThread();
new Thread(mt).start();//同一个mt，但是在Thread中就不可以，如果用同一
new Thread(mt).start();//个实例化对象mt，就会出现异常
new Thread(mt).start();
}
};
```

虽然现在程序中有三个线程，但是一共卖了10张票，也就是说使用Runnable实现多线程可以达到资源共享目的。

Runnable接口和Thread之间的联系：

public class Thread extends Object implements Runnable

发现Thread类也是Runnable接口的子类。
