

并发基础篇 (6) : 深入线程 Thread 类的 start() 方法和 run()方法

一、初识

java 的线程是通过 java.lang.Thread 类来实现的。VM 启动时会有一个由主方法所定义的线程。可以通过创建 Thread 的实例来创建新的线程。每个线程都是通过某个特定 Thread 对象所对应的方法 run () 来完成其操作的，方法 run()称为线程体。通过调用 Thread 类的 start()方法来启动一个线程。

在 Java 当中，线程通常都有五种状态，**创建、就绪、运行、阻塞和死亡**。

- 第一是**创建状态**。在生成线程对象，并没有调用该对象的 start 方法，这是线程处于创建状态。
- 第二是**就绪状态**。当调用了线程对象的 start 方法之后，该线程就进入了就绪状态，但是此时线程调度程序还没有把该线程设置为当前线程，此时处于就绪状态。在线程运行之后，从等待或者睡眠中回来之后，也会处于就绪状态。
- 第三是**运行状态**。线程调度程序将处于就绪状态的线程设置为当前线程，此时线程就进入了运行状态，开始运行 run 函数当中的代码。
- 第四是**阻塞状态**。线程正在运行的时候，被暂停，通常是为了等待某个时间的发生(比如说某项资源就绪)之后再继续运行。sleep,suspend , wait 等方法都可以导致线程阻塞。
- 第五是**死亡状态**。如果一个线程的 run 方法执行结束或者调用 stop 方法后，该线程就会死亡。对于已经死亡的线程，无法再使用 start 方法令其进入就绪。

二、start () 方法

1、为什么需要 start 方法；它的作用是什么？

start () 方法来启动线程，真正实现了多线程运行。

start 方法的作用就是将线程由 NEW 状态，变为 RUNABLE 状态。当线程创建成功时，线程处于 NEW (新建) 状态，如果你不调用 start()方法，那么线程永远处于 NEW 状态。调用 start()后，才会变为 RUNABLE 状态，线程才可以运行。

2、调用 start () 方法后，线程是不是马上执行？

线程不是马上执行的；准确来说，调用 start()方法后，线程的状态是“READY(就绪)”状态，而不是“RUNNING (运行中)”状态（关于线程的状态详细。线程要等待 CPU 调度，不同的 JVM 有不同的调度算法，线程何时被调度是未知的。因此，start () 方法的被调用顺序不能决定线程的执行顺序。

注意：

由于在线程的生命周期中，线程的状态由 NEW ----> RUNABLE 只会发生一次，因此，一个线程只能调用 start() 方法一次，多次启动一个线程是非法的。特别是当线程已经结束执行后，不能再重新启动。

三、run() 方法

1、run 方法又是一个什么样的方法？run 方法与 start 方法有什么关联？

run () 方法当作普通方法的方式调用

run() 其实是一个普通方法，只不过当线程调用了 start() 方法后，一旦线程被 CPU 调度，处于运行状态，那么线程才会去调用这个 run () 方法；

2、run () 方法的执行是不是需要线程调用 start () 方法

上面说了，run () 方法是一个普通的对象方法，因此，不需要线程调用 start () 后才可以调用的。可以线程对象可以随时随地调用 run 方法。

Example1：

```
Thread t1 = new Thread(new MyTask(1));
Thread t2 = new Thread(new MyTask(2));

t1.run();
t2.run();
```

上面的输出结果是固定的：

```
count 的值：1
count 的值：2
```

再看另一个实例：

```
Thread t1 = new Thread(new MyTask());
Thread t2 = new Thread(new MyTask());

t1.start();
t2.start();
```

这个输出结果不是固定的，因为线程的运行没法预测。运行结果可能不一样。

MyTask 类：

```
//实现 Runnable 接口
class MyTask implements Runnable{
    int count;
    public MyTask(int count) {
        this.count=count;
    }
    @Override
    public void run() {
```

```
        System.out.println("count 的值 : "+count);
    }
}
```

Example2 :

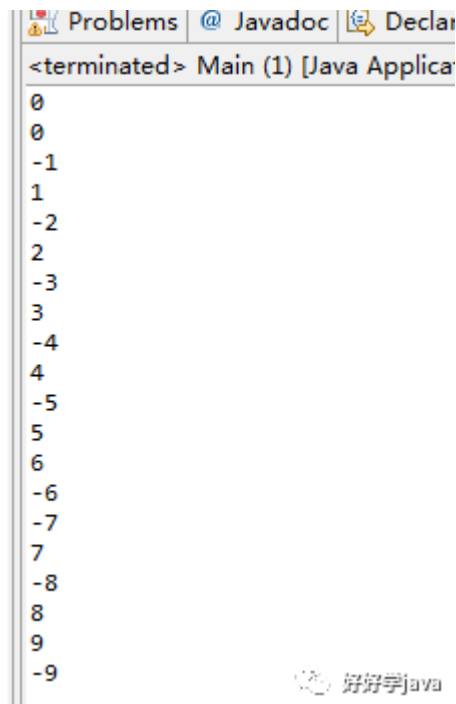
1、用 start 方法启动线程

```
public class Main {
    public static void main(String[] args) {
        Thread t1 = new Thread(new T1());
        Thread t2 = new Thread(new T2());
        t1.start();
        t2.start();
    }
}

class T1 implements Runnable {
    public void run() {
        try {
            for(int i=0;i<10;i++){
                System.out.println(i);
                Thread.sleep(100); //模拟耗时任务
            }
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}

class T2 implements Runnable {
    public void run() {
        try {
            for(int i=0;i>-10;i--){
                System.out.println(i);
                Thread.sleep(100); //模拟耗时任务
            }
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

结果：



```
<terminated> Main (1) [Java Applica
0
0
-1
1
-2
2
-3
3
-4
4
-5
5
6
-6
-7
7
-8
8
9
-9
```

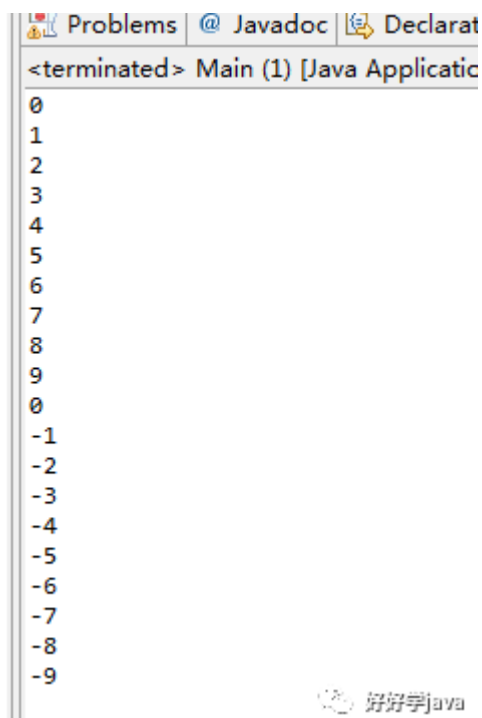
好好学java

说明两线程是并发执行的。

2、先用 run 方法启动线程

将上面的 start()改为 run()

```
public class Main {
    public static void main(String[] args) {
        Thread t1 = new Thread(new T1());
        Thread t2 = new Thread(new T2());
        t1.run();
        t2.run();
    }
}
```



```
<terminated> Main (1) [Java Applicat
0
1
2
3
4
5
6
7
8
9
0
-1
-2
-3
-4
-5
-6
-7
-8
-9
```

好好学java

说明两线程实际是顺序执行的。

总结:

通过实例 1 和实例和我们可以知道 start 方法是用于启动线程的，可以实现并发，而 run 方法只是一个普通方法，是不能实现并发的，只是在并发执行的时候会调用。

四、start()方法和 run()方法源码解析（基于JDK1.7.0_40）

```
public synchronized void start() {
    // 如果线程不是"就绪状态"，则抛出异常！
    if (threadStatus != 0)
        throw new IllegalThreadStateException();
    // 将线程添加到 ThreadGroup 中
    group.add(this);
    boolean started = false;
    try {
        // 通过 start0()启动线程,新线程会调用 run()方法
        start0();
        // 设置 started 标记=true
        started = true;
    } finally {
        try {
            if (!started) {
                group.threadStartFailed(this);
            }
        } catch (Throwable ignore) {}
    }
}
```

```
public void run() {
    if (target != null) {
        target.run();
    }
}
```

五、真正理解 Thread 类

Thread 类的对象其实也是一个 java 对象，只不过每一个 Thread 类的对象对应着一个线程。Thread 类的对象就是提供给用户用于操作线程、获取线程的信息。真正的底层线程用户是看不到的了。

因此，当一个线程结束了，死掉了，对应的 Thread 的对象仍能调用，除了 start() 方法外的所有方法（死亡的线程不能再次启动），如 run()、getName()、getPriority() 等等。

//简单起见，使用匿名内部类的方法来创建线程

```
Thread thread = new Thread(){
    @Override
    public void run() {
        System.out.println("Thread 对象的 run 方法被执行了");
    }
};

//线程启动
thread.start();

//用循环去监听线程 thread 是否还活着，只有当线程 thread 已经结束了，才跳出循环
while(thread.isAlive()){
    //线程 thread 结束了，但仍能调用 thread 对象的大部分方法
    System.out.println("线程"+thread.getName()+"的状态："+thread.getState()+"---优先级："+thread.getPriority());

    //调用 run 方法
    thread.run();

    //当线程结束时，start 方法不能调用，下面的方法将会抛出异常
    thread.start();
}
```