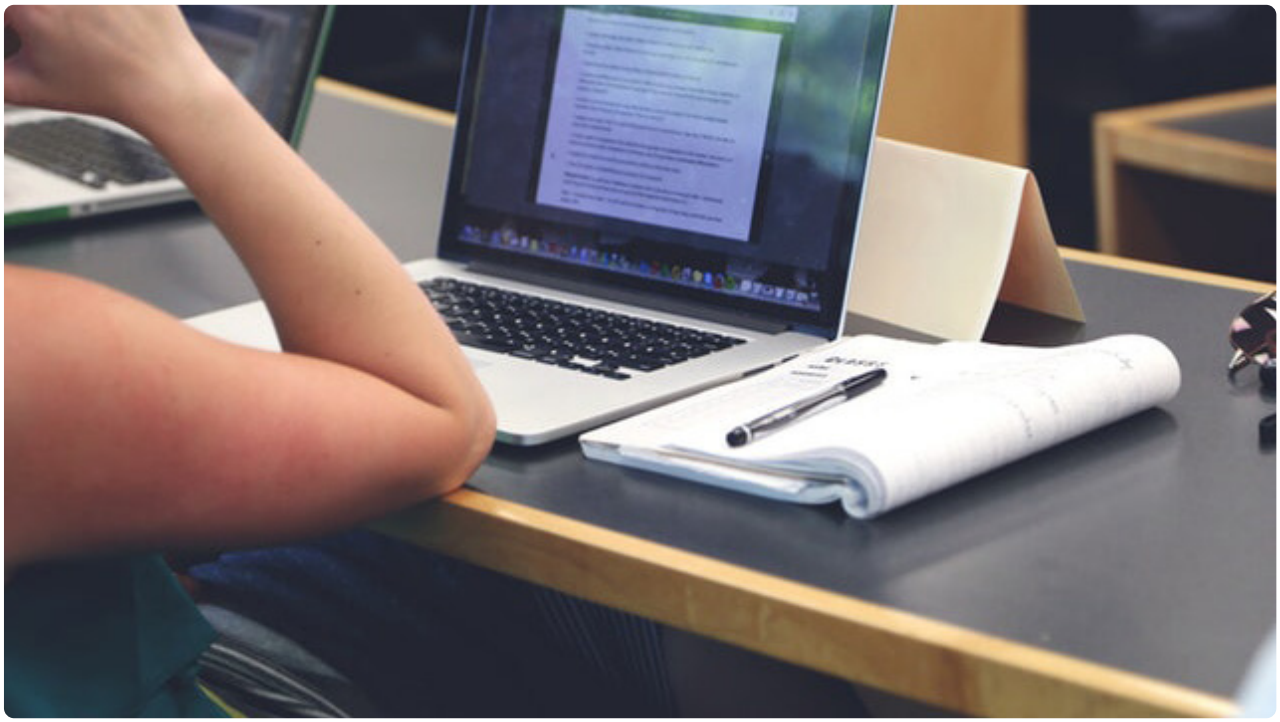
03 多线程开发如此简单—Java中如何编写多线程程序

更新时间：2019-09-10 16:30:55



> 学习这件事不在乎有没有人教你，最重要的是在于你自己有没有觉悟和恒心。
>
> —— 法布尔

1. Java 实现多线程的方式

前文介绍了多线程的各种应用场景，你是不是已经磨刀霍霍，迫不及待想进入 java 多线程的世界里了？别急，我们第一步要先得到进入多线程世界的钥匙，也就是如何在 java 中实现多线程。

在 java 中实现多线程有四种方式，如下：

1. 继承 Thread 类
2. 实现 Runnable 接口
3. 使用 FutureTask
4. 使用 Executor 框架

其中继承 Thread 类和实现 Runnable 接口是最基本的方式，但有一个共同的缺点 ---- 没有返回值。而 FutureTask 则解决了这个问题，后面会单独讲解。Executor 是 JDK 提供的多线程框架，功能十分强大，后面也会有章节专门讲解。本篇文章主要介绍前两种最基本的方式，目的是让你对多线程编程有初步的认识，带你打开多线程编程的大门。

前文我说过，无形的软件，都来自于有形的现实世界。我们在学习多线程的过程中，时刻以现实世界作为参照，理解起来就会容易很多。我们设想这样一个生活中的场景，看看程序如何实现。

小明是一位学生，今天不太开心。因为昨天英语课学习了一个新的单词，今天考试时他写错了。老师惩罚他抄写 100 遍。这个单词有点长，是什么单词呢？internationalization。看着眼熟吗？做过国际化开发的同学一定认识，这个单词因为太长，在 java 中被称为 i18n，也就是首字母 i 和尾字母 n 之间有 18 个字母。小明很苦恼，怎么能快点写完呢？

2. 单线程实现单词抄写

OK，下面我们通过程序来模拟小明抄写单词的任务。我们编写如下几个类：

1、Punishment.java

存储要抄写的单词，以及剩余的抄写次数。主要代码如下：

```
public class Punishment {
    private int leftCopyCount;
    private String wordToCopy;
}
```

2、Student.java

持有 Punishment 的引用。实现了抄写单词的 copyWord 方法。主要代码如下：

```
public class Student {
    private String name;
    private Punishment punishment;

    public Student(String name,Punishment punishment) {
        this.name=name;
        this.punishment = punishment;
    }

    public void copyWord() {
        int count = 0;
        String threadName = Thread.currentThread().getName();

        while (true) {
            if (punishment.getLeftCopyCount() > 0) {
                int leftCopyCount = punishment.getLeftCopyCount();
                System.out.println(threadName+"线程-"+name + "抄写" + punishment.getWordToCopy() + "。还要抄写" + --leftCopyCount + "次");
                punishment.setLeftCopyCount(leftCopyCount);
                count++;
            } else {
                break;
            }
        }

        System.out.println(threadName+"线程-"+name + "一共抄写了" + count + "次！");
    }
}
```

Student 构造函数传入 Punishment。copyWord 方法是根据惩罚内容。完成单词抄写的主要逻辑。

我们重点看一下 coppyWord 方法。count 变量是计数器，记录抄写的总次数。threadName 是本线程的名称，这里通过 Thread 的静态方法 currentThread 取得当前线程，然后通过 getName 方法获取线程名称。

在 while 循环体中，当 punishment 的剩余抄写次数大于 0 时，执行抄写逻辑，否则抄写任务完成，跳出循环。逻辑很简单，相信大家都能看懂。接下来我们通过 main 方法尝试运行，看看效果。main 方法代码如下：

```
public class StudentClient {
    public static void main(String[] args) {
        Punishment punishment = new Punishment(100,"internationalization");
        Student student = new Student("小明",punishment);
        student.copyWord();
    }
}
```

输出如下：

```
main线程-小明抄写internationalization。还要抄写99次
.....（中间省略）
main线程-小明抄写internationalization。还要抄写0次
main线程-小明一共抄写了100次！
```

在控制台可以清楚地看到小明抄写了 100 次单词。不过此时的代码并没有引入多线程，是单线程小明在工作。唯一看到的和线程沾边的就是日志中的 “main 线程”，这是通过 Thread.*currentThread*().getName () 获取的当前线程名称，也就是 main 函数所在的线程。

3. 继承 Thread 实现独立线程单词抄写

接下来我们尝试为小明单独起一个线程做这个事情，而不是在 main 线程中完成。回到我们所讲的主题，实现多线程的方式上，我们先采用继承 thread 类，重写 run 方法的方式。改版后，student 代码如下：

```
//1、继承Thread类
public class Student extends Thread{
    private String name;
    private Punishment punishment;

    public Student(String name, Punishment punishment) {
        //2、调用Thread构造方法，设置threadName
        super(name);
        this.name=name;
        this.punishment = punishment;
    }

    public void copyWord() {
        int count = 0;
        String threadName = Thread.currentThread().getName();

        while (true) {
            if (punishment.getLeftCopyCount() > 0) {
                int leftCopyCount = punishment.getLeftCopyCount();
                System.out.println(threadName+"线程-"+name + "抄写" + punishment.getWordToCopy() + "。还要抄写" + --leftCopyCount + "次");
                punishment.setLeftCopyCount(leftCopyCount);
                count++;
            } else {
                break;
            }
        }
        System.out.println(threadName+"线程-"+name + "一共抄写了" + count + "次！");
    }
    //3、重写run方法，调用copyWord完成任务
    @Override
    public void run(){
        copyWord();
    }
}
```

三个变化点在代码中已经标出。不再多说，只提醒下，在第 2 个点，我们设置了线程的名称，一会在输出中会看到带来的变化。

main 方法代码如下：

```
public class StudentClient {
    public static void main(String[] args) {
        Punishment punishment = new Punishment(100,"internationalization");
        Student student = new Student("小明",punishment);
        student.start();
    }
}
```

可以看到此时调用的不是 student 的 copyWord 方法，而是调用了 start 方法。start 方法是从 Thread 类继承而来，调用后线程进入就绪状态，等待 CPU 的调用。而 start 方法最终会触发执行 run 方法，在 run 方法中 copyWord 被执行。输出如下：

```
小明线程-小明抄写internationalization。还要抄写99次
.....（中间省略）
小明线程-小明抄写internationalization。还要抄写0次
小明线程-小明一共抄写了100次！
```

我们可以看到，现在不再是 main 线程在工作了，而是小明线程。这说明 student 已经工作在“小明”线程上。为了更加直观，我们在 student.start () 后面加一行代码:

```
System.out.println("Another thread will finish the punishment。 main thread is finished" );
```

再次运行程序，输出如下：

```
Another thread to finish the punishment。main thread is finished
小明线程-小明抄写internationalization。还要抄写99次
.....（中间省略）
小明线程-小明抄写internationalization。还要抄写0次
小明线程-小明一共抄写了100次！
```

可以看到主线程在 student.start () 后，会立即向下执行。而小明线程则在独立执行 copyWord 方法。这里你可以做个对比，单线程情况下，一定是在小明抄写的所有输出后才会输出 “main thread is finished”。

4. 多线程并发实现单词抄写

你心里一定在想，这个例子没有看到多线程的好处啊？是的，如果仅仅是小明一个人去完成任务，其实和单线程没有区别。但是假如小明找来了几个同学帮他一起写呢？

我们在 main 方法中启动多个线程一块完成单词抄写任务：

```
public static void main(String[] args) {
    Punishment punishment = new Punishment(100,"internationalization");

    Student xiaoming = new Student("小明",punishment);
    xiaoming.start();

    Student xiaozhang = new Student("小张",punishment);
    xiaozhang.start();

    Student xiao赵 = new Student("小赵",punishment);
    xiaozhang.start();
}
```

大家对这段代码的期望结果是什么呢？按照正常的逻辑，应该是小明先开始写，他会抄写的次数多一点，而小张和小赵抄写的次数少一点，但是三人抄写的总量应该是 100。不过事与愿违，我们在控制台可以看到如下输出：

```
小赵线程-小赵一共抄写了100次！
小明线程-小明一共抄写了100次！
小张线程-小张一共抄写了100次！
```

小明的工作量不但没有减少，还连累小张和小赵白白抄写了 100 遍，为什么会这样呢？！我在下篇专栏中会详细解答。这里我可以先肯定的告诉你，我们是有办法解决现在的问题，达到想要的执行效果。本篇文章我们还是聚焦在多线程如何实现上。

接下来，我们看另外一种多线程实现方式。

5. 实现 Runnable 接口，启用单独线程抄写单词

上面讲解了通过继承 Thread 的方式来实现多线程，接下来我们看看如何以实现 Runnable 接口的形式实现多线程。student 代码改造后如下：

```
public class Student implements Runnable{
    private String name;
    private Punishment punishment;

    public Student(String name, Punishment punishment) {
        this.name=name;
        this.punishment = punishment;
    }

    public void copyWord() {
        int count = 0;
        String threadName = Thread.currentThread().getName();

        while (true) {
            if (punishment.getLeftCopyCount() > 0) {
                int leftCopyCount = punishment.getLeftCopyCount();
                System.out.println(threadName+"线程-"+name + "抄写" + punishment.getWordToCopy() + "。还要抄写" + --leftCopyCount + "次");
                punishment.setLeftCopyCount(leftCopyCount);
                count++;
            } else {
                break;
            }
        }

        System.out.println(threadName+"线程-"+name + "一共抄写了" + count + "次！");
    }

    //重写run方法，完成任务。
    @Override
    public void run(){
        copyWord();
    }
}
```

和继承 thread 实现多线程的区别，在于现在是实现 runnable 接口。不过也是需要实现 run () 方法。另外由于 runnable 是接口，所以之前构造函数中调用父类构造函数的语句需要去掉。

我们再看看 StudentClient 的代码：

```
public class StudentClient {
    public static void main(String[] args) {
        Punishment punishment = new Punishment(100,"internationalization");
        Thread xiaoming = new Thread(new Student("小明",punishment),"小明");
        xiaoming.start();
    }
}
```

可以看到我们需要创建一个 thread，把实现了 runnable 接口的对象通过构造函数传递进去，Thread 构造函数的第二个参数是自定义的 thread name。之前由于 Student 就是 Thread 的子类，所以我们直接通过 new Student 就可以得到线程对象。最后都是通过调用 Thread 对象的 start 方法来启动线程。运行代码后发现输出结果和继承 thread 方式是一模一样的。

6. 总结

本篇讲解的内容非常基础，目的在于让大家对多线程开发有所感知，快速上手。建议大家自己把代码敲一边，体会两种启动线程方式的异同。此外，可以重点思考下，为什么多线程并发时，结果并不是我们所期望的。看一看你的答案是否和下篇专栏所写的原因一样。通过本篇学习，我们知道在 java 中启动多线程非常简单。但是，要想处理好多线程间的协调，并不是一个容易的事情。而多线程开发的难点也就在于此。下一节我们就来看看多线程开发中会遇到的问题。

```
}
```

02 绝对不仅仅是为了面试—我们为什么需要学习多线程

04 人多力量未必大—并发可能会遇到的问题