毕向东老师多线程学习笔记 1
用 Thread 实现多线程

随着 CPU 频率的增长出现停滞（目前处在 4-5G 的瓶颈），CPU 转向多核方向发展。多线程作为实现软件并发执行的一个重要方法，也就具有越来越重要的地位。

Java 在多线程处理方面性能超群、功能强大。而且 Java 语言进行多线程处理很简单，所以程序员利用多线程技术能编写出非常有效率的程序来充分利用 CPU，使 CPU 的空闲时间能够保持在最低限度。

进程是程序的一次动态执行过程，它经历了从代码加载、执行到执行完毕的一个完整过程，这个过程也是进程本身从产生、发展到最终消亡的过程。多进程操作系统能同时运行多个进程（程序），由于 CPU 具备分时机制，所以每个进程都能循环获得自己的 CPU 时间片。由于 CPU 执行速度非常快，使得所有程序好象是在“同时”运行一样。

线程也称为轻量级进程，是程序执行的最小单元。一个标准的线程由线程 ID，当前指令指针 PC，寄存器集合和堆栈组成。一般一个进程由一个到多个线程组成，各线程之间共享程序的内存空间（包括代码段、数据段、堆等）及一些进程资源(如打开文件和信号)。一个经典的线程与进程的关系如图 1 所示。

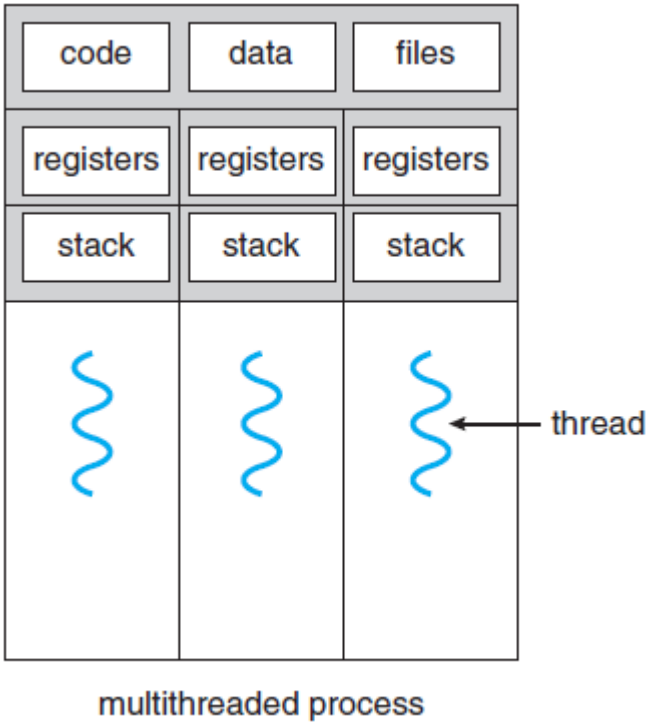


图 1 进程内的线程。
图片来源：操作系统概念第 9 版第四章第 1 节
或程序员的自我修养第一章第 1.6 节

本节先总结 java 创建线程的 2 种方法。
一、创建线程的第一种方法，通过继承 Thread 类实现多线程。
步骤：

1、定义一个“线程”类继承 Thread 类。
2、复写 Thread 类的 run 方法。
3、通过“线程”类的对象调用父类 Thread 的 start()方法，来启动“线程”类的 run()方法内的线程体。
格式：

```
class 线程类名 extends Thread /
{
    属性
    方法…
    修饰符 run(){ // 复写Thread类里的run()方法
    以线程处理的程序;
}
```

例如：用多线程实现买票小程序。

```
    public class ThreadDemo {
        public static void main(String [] args)
        {
            //定义了 4 个线程类的匿名对象，分别启动各自的线程
            new TestThread().start();//
            new TestThread().start();//
            new TestThread().start();//
            new TestThread().start();//
        }
    }
    //“线程”类
    class TestThread extends Thread {
        private int tickets=5;//一共只卖 5 张票
        public void run() {
            while(true) {
                if(tickets>0)
                    System.out.println(Thread.currentThread().getName()+"
                    出售票"+tickets--);
        }
    }
}
```

输出结果为：

```
Thread-0 出售票 5
Thread-0 出售票 4
Thread-3 出售票 5
Thread-2 出售票 5
Thread-2 出售票 4
Thread-1 出售票 5
Thread-1 出售票 4
Thread-2 出售票 3
Thread-3 出售票 4
```

Thread-0 出售票 3
Thread-3 出售票 3
Thread-2 出售票 2
Thread-1 出售票 3
Thread-2 出售票 1
Thread-3 出售票 2
Thread-3 出售票 1
Thread-0 出售票 2
Thread-0 出售票 1
Thread-1 出售票 2
Thread-1 出售票 1

由输出结果可以看出：

4 个线程各自卖了 5 张票，一个卖了 20 张票，不符合设计要求。原因是这 4 个线程对象分别调用了各自的 start 方法，启动各自的 run 方法内的线程体，这四个线程对象各自拥有自己的资源，所以用继承 Thread 类方法无法达到数据共享的目的。

可以通过 static 关键字把需要共享的数据静态化，让四个线程对象共享一个静态成员变量，例如把成员变量 tickets 静态化，把上例中的 private int tickets=5;修改为

```
private static int tickets=20;
```

输出结果为：

Thread-0 出售票 20
Thread-0 出售票 16
Thread-3 出售票 17
Thread-2 出售票 18
Thread-1 出售票 19
Thread-2 出售票 13
Thread-3 出售票 14
Thread-0 出售票 15
Thread-3 出售票 10
Thread-2 出售票 11
Thread-1 出售票 12
Thread-2 出售票 7
Thread-3 出售票 8
Thread-0 出售票 9
Thread-3 出售票 4
Thread-2 出售票 5
Thread-2 出售票 1
Thread-1 出售票 6
Thread-3 出售票 2
Thread-0 出售票 3

由输出结果可看出 4 个线程共享一个静态变量 tickets = 20，将同一数量的票分别卖出，符合设计要求。但实际工程中一般不在线程中使用静态变量，因为他的生命周期太长，相当于 C 语言的全局变量，一直占用内存空间(静态变量在内存的数据区分配)，直到整个程序执行结束，静态变量才销毁。

当一个类继承 Thread 类之后，这个类的对象无论调用多少次 start()方法，结果都只有一个线程在运行，如下所示：

```
public class ThreadDemo
{
    public static void main(String [] args)
    {
        TestThread t=new TestThread();//新建一个线程对象
        // 一个线程对象只能启动一次，此处启动了 4 次。
        t.start();//1
        t.start();//2
        t.start();//3
        t.start();//4
    }
}
```

字数有限，输出结果省略(可参考毕向东老师的视频教程)，由毕老师视频可知，程序运行时出现异常，且只有 1 个线程的 run 方法被执行，其他 3 个线程都无法执行。用毕老师的话就是一个运动员在跑 4x100m 是，发令员只能在运动员跑第一圈时打响发令枪，而不能在第二三四圈时都打发令枪。

由上面的 3 个小实例可看出用继承 Thread 类的方法实现多线程时会有以下缺点：

1. 用 Thread 类无法达到资源共享的目的，
2. 用 Thread 类实现多线程时最好不用静态变量来实现数据共享。
3. 一个类继承 Thread 类后，无论这个类的对象调用多少次 start()方法，结果都只有一个线程在运行。

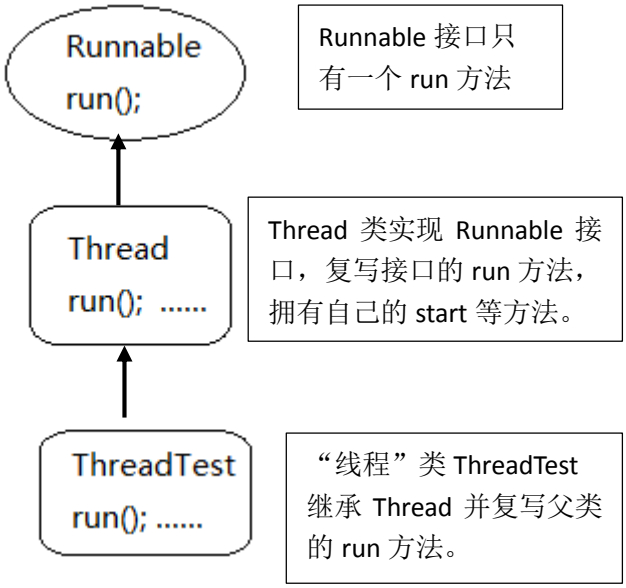
最后总结一下 start()方法和 run()方法的区别。

当线程对象调用父类 Theard 类的 start()方法后，start()方法告诉虚拟机的线程调度器，该对象内的 run()方法处于就绪状态，此时 run()方法并没有运行，一旦得到 cpu 时间片后，就开始执行 run（）方法，这里方法 run（）称为线程体，它包含了要执行的这个线程的内容，Run 方法运行结束，此线程随即终止。

Run()方法只是 Thread 类或 Runnable 接口的子类中的一个普通的方法而已，如果直接调用用 Run 方法，程序中依然只有主线程这一个线程，程序执行路径还是只有一条，还是要顺序执行，要等待 run 方法体执行完毕后才可继续执行下面的代码，这样就没有达到写线程的目的。

总结：调用 start 方法方可启动线程，而 run 方法只是 thread 或 runnable 的一个普通方法调用，还是在主线程里执行。

由 java API 可知 Thread 类实现 Runnable 接口，所以 Thread 类中的中的 run 方法是复写 Runnable 接口的 run 方法。所以个人感觉，当一个“线程”类继承 Thread 类时，复写 Thread 的 run 方法时，具相当于间接复写 Runnable 的 run 方法。如下图所示。


Runnable
run();
Runnable 接口只有一个 run 方法
Thread
run(); .....
Thread 类实现 Runnable 接口，复写接口的 run 方法，拥有自己的 start 等方法。
ThreadTest
run(); .....
“线程”类 ThreadTest 继承 Thread 并复写父类的 run 方法。