笔记总链接：http://bbs.itheima.com/thread-200600-1-1.html

4、继承

4.12 包

对类文件进行分类管理。

给类提供多层命名空间。

写在程序文件的第一行。

类名的全称的是：包名.类名。

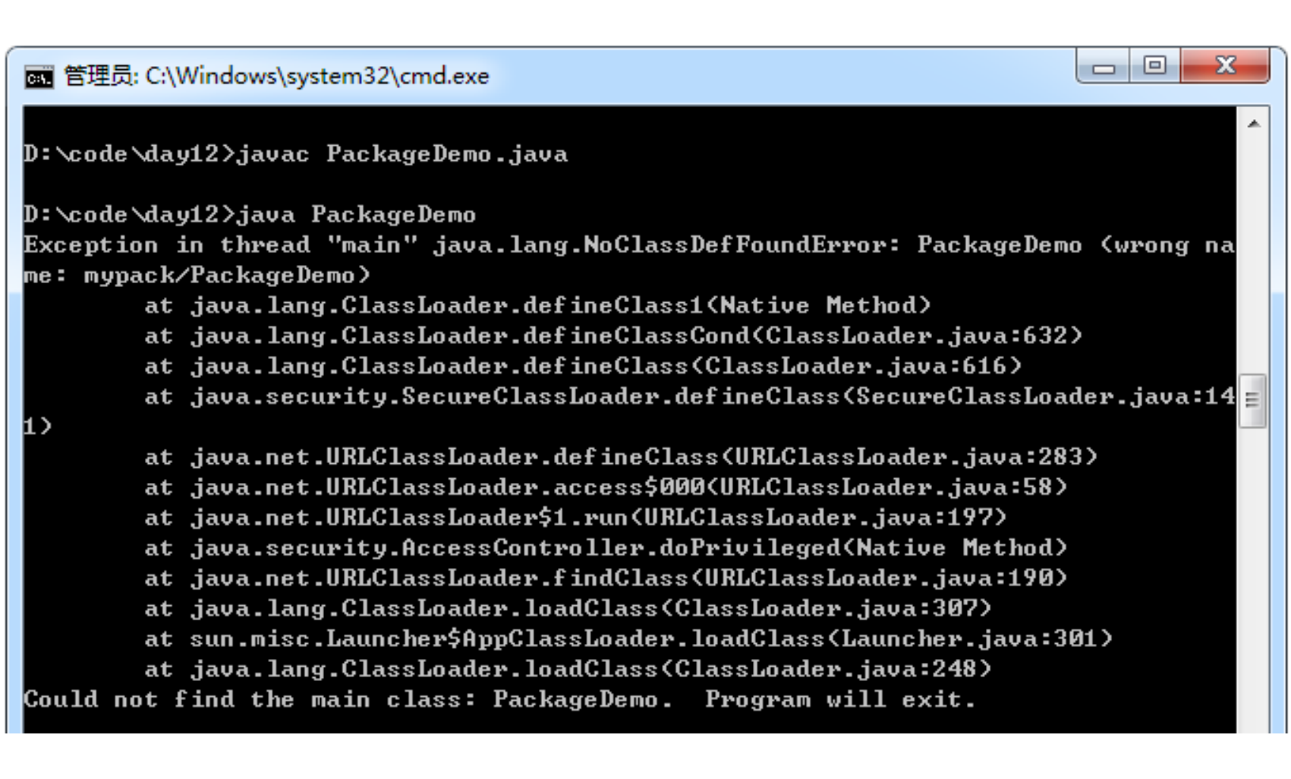
包也是一种封装形式。

示例1：

```
package mypack;

class PackageDemo{
      public static void main(String[] args){
           System.out.println("Hello Package!");
      }
}
```

复制代码

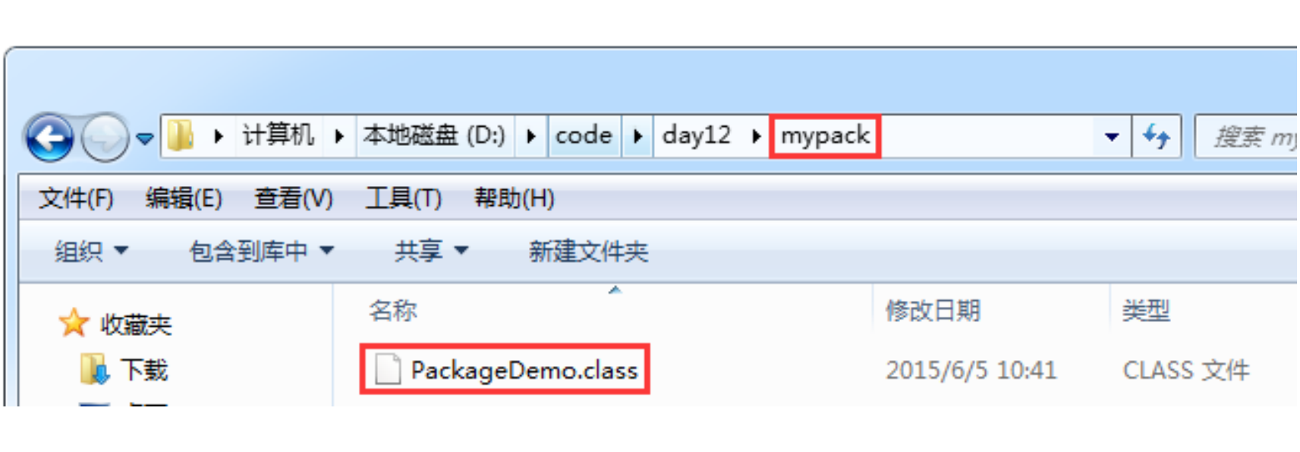
运行结果：



```
D:\code\day12>javac PackageDemo.java

D:\code\day12>java PackageDemo
Exception in thread "main" java.lang.NoClassDefFoundError: PackageDemo (wrong na
me: mypack/PackageDemo)
        at java.lang.ClassLoader.defineClass1(Native Method)
        at java.lang.ClassLoader.defineClassCond(ClassLoader.java:632)
        at java.lang.ClassLoader.defineClass(ClassLoader.java:616)
        at java.security.SecureClassLoader.defineClass(SecureClassLoader.java:14
1)
        at java.net.URLClassLoader.defineClass(URLClassLoader.java:283)
        at java.net.URLClassLoader.access$000(URLClassLoader.java:58)
        at java.net.URLClassLoader$1.run(URLClassLoader.java:197)
        at java.security.AccessController.doPrivileged(Native Method)
        at java.net.URLClassLoader.findClass(URLClassLoader.java:190)
        at java.lang.ClassLoader.loadClass(ClassLoader.java:307)
        at sun.misc.Launcher$AppClassLoader.loadClass(Launcher.java:301)
        at java.lang.ClassLoader.loadClass(ClassLoader.java:248)
Could not find the main class: PackageDemo.  Program will exit.
```

创建mypack文件夹，并且将PackageDemo.class放入其中，然后再次执行。

运行结果：



```
D:\code\day12>java mypack.PackageDemo
Hello Package!
```

也可以通过javac命令直接创建mypack文件夹，然后执行。

```
D:\code\day12>javac -d . PackageDemo.java

D:\code\day12>java mypack.PackageDemo
Hello Package!
```

示例2：

DemoA.java

```
package packa;

public class DemoA{
        public void show(){
              System.out.println("demoa show run");
        }
}
```

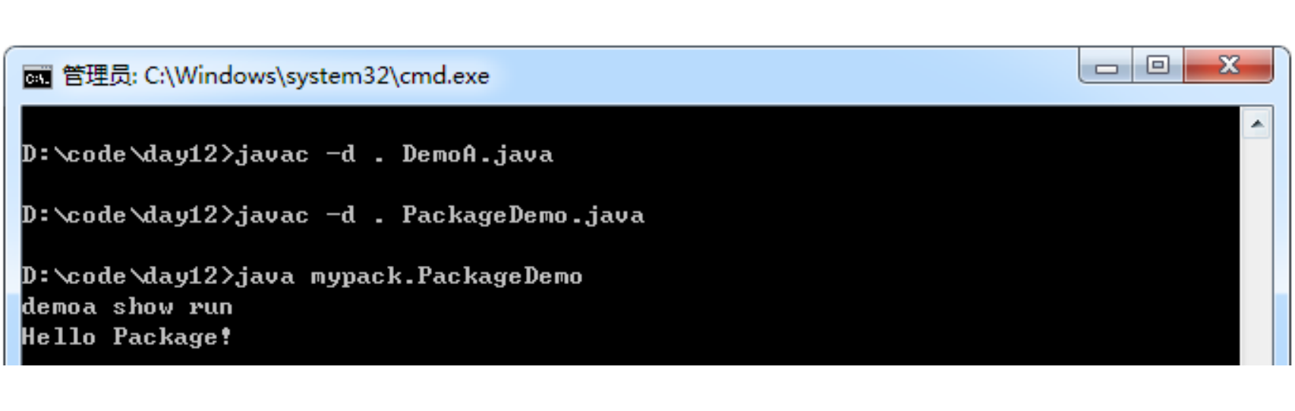
复制代码

PackageDemo.java

```
package mypack;

class PackageDemo{
        public static void main(String[] args){
               packa.DemoA d = new packa.DemoA();
               d.show();
               System.out.println("Hello Package!");
        }
}
```

复制代码

运行结果：



```
D:\code\day12>javac -d . DemoA.java

D:\code\day12>javac -d . PackageDemo.java

D:\code\day12>java mypack.PackageDemo
demoa show run
Hello Package!
```

P.S.

包与包之间的类进行访问，被访问的包中的类必须是public的，被访问的包中的类的方法也必须是public的。

示例3：

DemoA.java

```
package packa;

public class DemoA extends packb.DemoB{
        public void show(){
              method();
              System.out.println("demoa show run");
        }
}
```

复制代码

DemoB.java

```
package packb;
public class DemoB{
        protected void method(){
              System. out.println("demob show run" );
        }
}
```

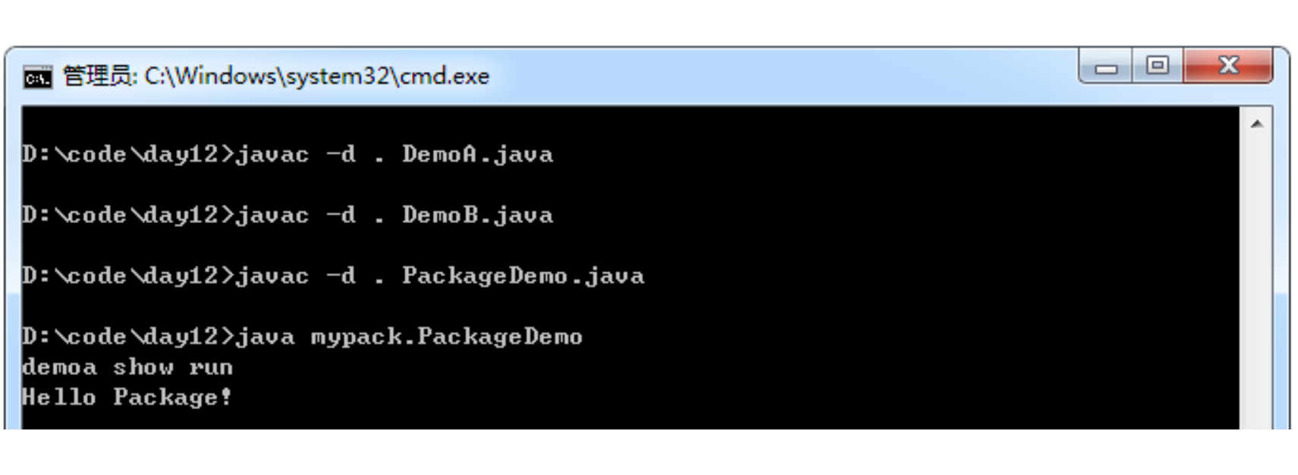
复制代码

PackageDemo.java

```
package mypack;

class PackageDemo{
        public static void main(String[] args){
               packa.DemoA d = new packa.DemoA();
               d.show();

               packb.DemoB b = new packb.DemoB();
               //b.method();//报错！无法访问DemoB中的protected修饰的method方法
               System.out.println("Hello Package!");
        }
}
```

复制代码

运行结果：



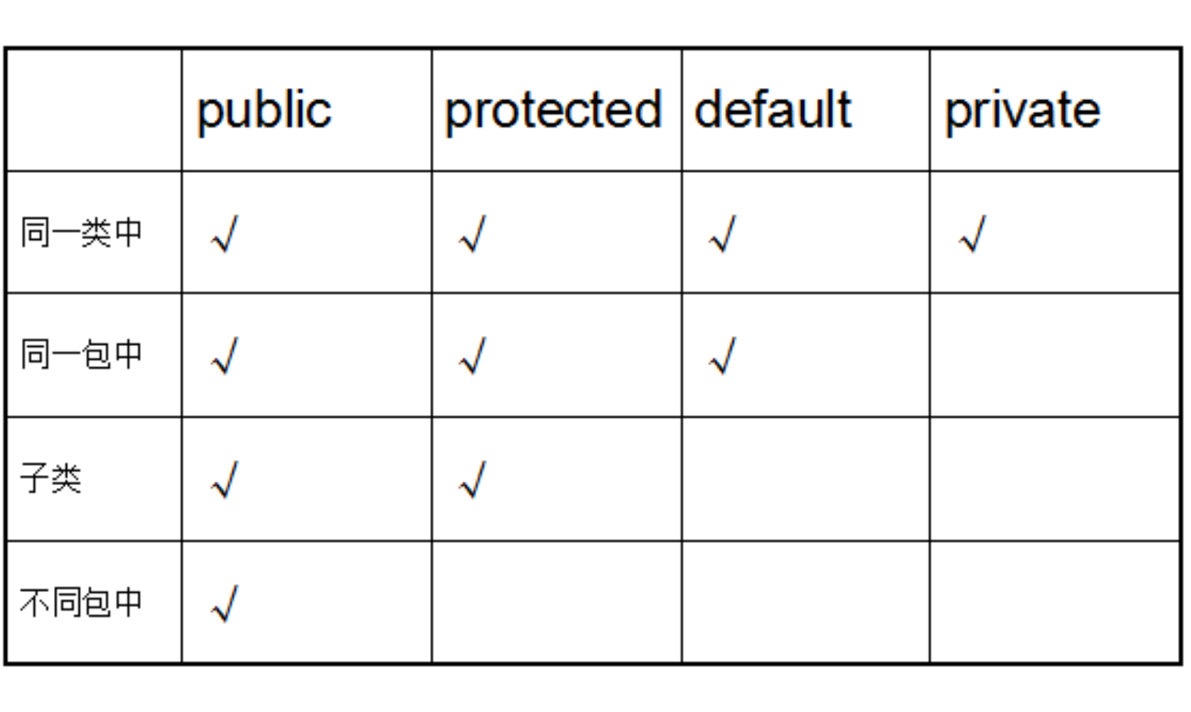
```
D:\code\day12>javac -d . DemoA.java

D:\code\day12>javac -d . DemoB.java

D:\code\day12>javac -d . PackageDemo.java

D:\code\day12>java mypack.PackageDemo
demoa show run
Hello Package!
```

包之间的访问：被访问的包中的类权限必须是public的。

类中的成员权限：public或者protected。

protected是为其他包中的子类提供的一种权限。

四种权限



	public	protected	default	private
同一类中	√	√	√	√
同一包中	√	√	√	
子类	√	√		
不同包中	√			

import

一个程序文件中只有一个package，但可以有多个import。

示例：

DemoA.java

```
package packa;

public class DemoA extends packb.DemoB{
        public void show(){
              method();
              System.out.println("demoa show run");
        }
}
```

复制代码

DemoB.java

```
package packb;
public class DemoB{
        protected void method(){
              System. out.println("demob show run" );
        }
}
```

复制代码

PackageDemo.java

```
package mypack;
//import packa.DemoA;//导入了 packa包中的DemoA类。
import packa.*; //导入了packa包中所有的类

class PackageDemo{
      public static void main(String[] args){
          DemoA d = new DemoA();
          d.show();
      }
}
```

复制代码

运行结果：

```
D:\code\day12>javac -d . PackageDemo.java

D:\code\day12>java mypack.PackageDemo
demob show run
demoa show run
```

示例：

有两个类：DemoA、DemoAbc。

所在文件目录如下：

packa\DemoA.class

packa\abc\DemoAbc.class

导包语句如下：

import packa.*;

import packa.abc.*;

Jar包

Java的压缩包。

方便项目的携带。

方便于使用，只要在classpath设置jar路径即可。

数据库驱动，SSH框架等都是以jar包体现的。

Jar包的操作：

通过jar.exe工具对jar的操作。

创建jar包：

jar -cvf mypack.jar packa packb

查看jar包

jar -tvf mypack.jar [>定向文件]

解压缩

jar -xvf mypack.jar

自定义jar包的清单文件

jar –cvfm mypack.jar mf.txt packa packb

5、多线程

5.1 多线程的概念

5.1.1 进程、线程、多进程的概念

进程：正在进行中的程序（直译）。

线程：进程中一个负责程序执行的控制单元（执行路径）。

P.S.

1、一个进程中可以有多个执行路径，称之为多线程。

2、一个进程中至少要有一个线程。

3、开启多个线程是为了同时运行多部分代码，每一个线程都有自己运行的内容，这个内容可以称为线程要执行的任务。

多线程的好处：解决了多部分代码同时运行的问题。

多线程的弊端：线程太多，会导致效率的降低。

其实，多个应用程序同时执行都是CPU在做着快速的切换完成的。这个切换是随机的。CPU的切换是需要花费时间的，从而导致了效率的降低。

JVM启动时启动了多条线程，至少有两个线程可以分析的出来：

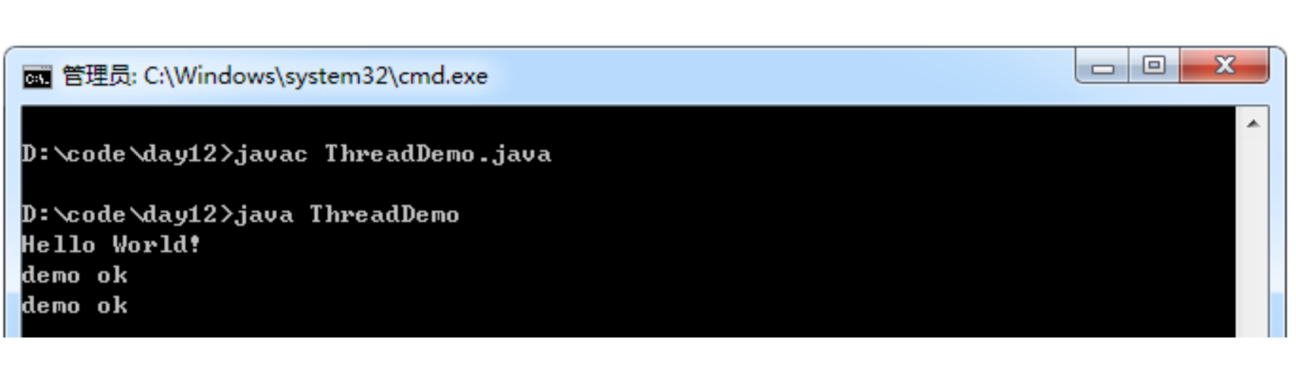
1. 执行main函数的线程，该线程的任务代码都定义在main函数中。
2. 负责垃圾回收的线程。

示例：

```
class Demo extends Object{
      public void finalize(){
          System.out.println("demo ok");
      }
}

class ThreadDemo{
      public static void main(String[] args){
           new Demo();
           new Demo();
           System.gc();
           new Demo();
           System.out.println("Hello World!");
      }
}
```

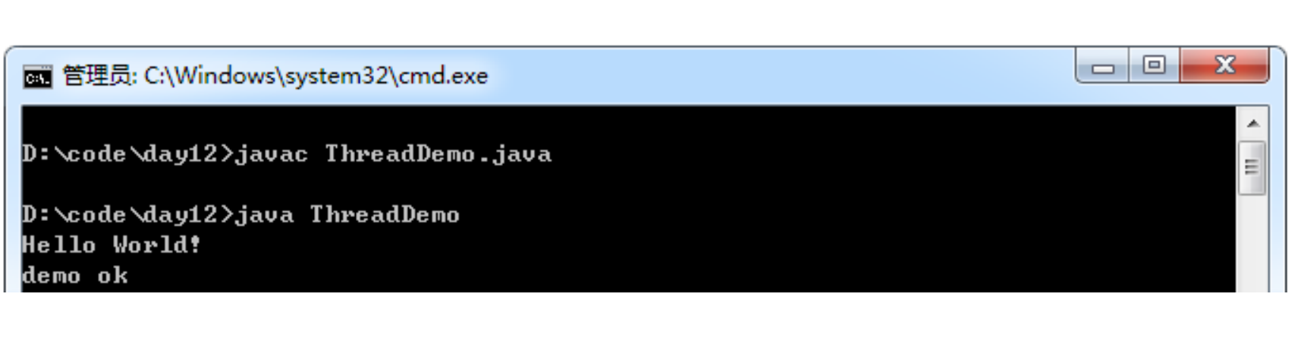
复制代码

运行结果：



```
D:\code\day12>javac ThreadDemo.java

D:\code\day12>java ThreadDemo
Hello World!
demo ok
demo ok
```

像这种情况，之所以先打印Hello World！再打印demo ok，是因为两条线程是分别执行的。



```
D:\code\day12>javac ThreadDemo.java

D:\code\day12>java ThreadDemo
Hello World!
demo ok
```

像这种情况，只打印一个demo ok，是因为在垃圾回收器还没回收第二个Demo对象的时候，JVM就已经结束了。

P.S.

System类的gc方法告诉垃圾回收器调用finalize方法，但不一定立即执行。

5.1.2 创建线程方式一：继承Thread类

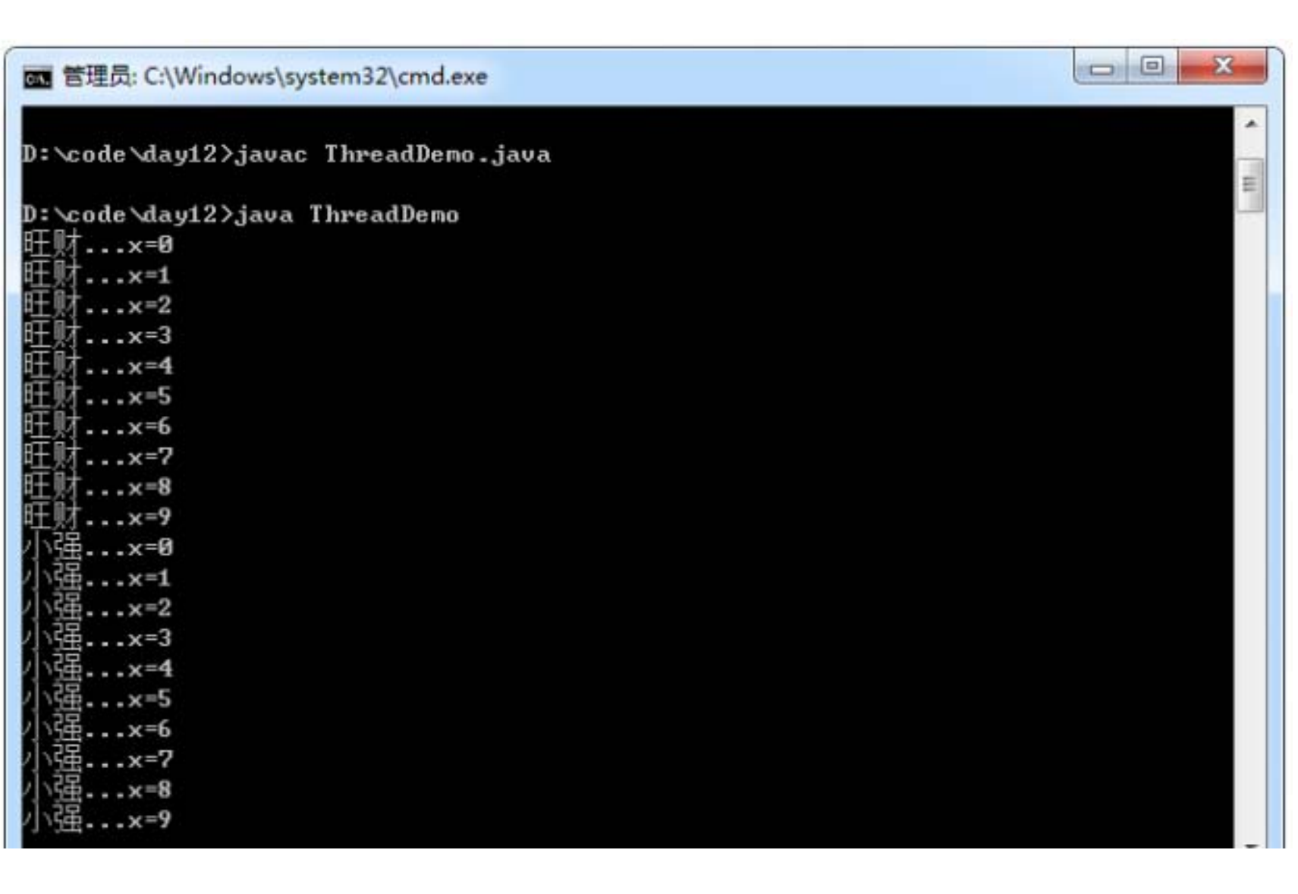
1. 定义一个类继承Thread类。
2. 覆盖Thread类中的run方法。
3. 直接创建Thread的子类对象创建线程。
4. 调用start方法开启线程并调用线程的任务run方法执行。

单线程程序示例：

```
class Demo{
      private String name ;
      Demo(String name){
           this.name = name;
      }
      public void show(){
           for(int x = 0; x < 10; x++){
                System.out.println(name + "...x=" + x);
           }
      }
}

class ThreadDemo{
      public static void main(String[] args){
           Demo d1 = new Demo("旺财");
           Demo d2 = new Demo("小强");
           d1.show();
           d2.show();
      }
}
```

复制代码

运行结果：



```
D:\code\day12>javac ThreadDemo.java

D:\code\day12>java ThreadDemo
旺财...x=0
旺财...x=1
旺财...x=2
旺财...x=3
旺财...x=4
旺财...x=5
旺财...x=6
旺财...x=7
旺财...x=8
旺财...x=9
小强...x=0
小强...x=1
小强...x=2
小强...x=3
小强...x=4
小强...x=5
小强...x=6
小强...x=7
小强...x=8
小强...x=9
```

可以看到在单线程程序中，只有上一句代码执行完，下一句代码才有执行的机会。

创建线程的目的就是为了开启一条执行路径，去运行指定的代码和其他代码实现同时运行，而运行的指定代码就是这个执行路径的任务。

jvm创建的主线程的任务都定义在了主函数中。而自定义的线程，它的任务在哪儿呢？

Thread类用于描述线程，线程是需要任务的。所以Thread类也有对任务的描述。这个任务就是通过Thread类中的run方法来体现。也就是说，run方法就是封装自定义线程运行任务的函数，run方法中定义的就是线程要运行的任务代码。

开启线程是为了运行指定代码，所以只有继承Thread类，并复写run方法，将运行的代码定义在run方法中即可。

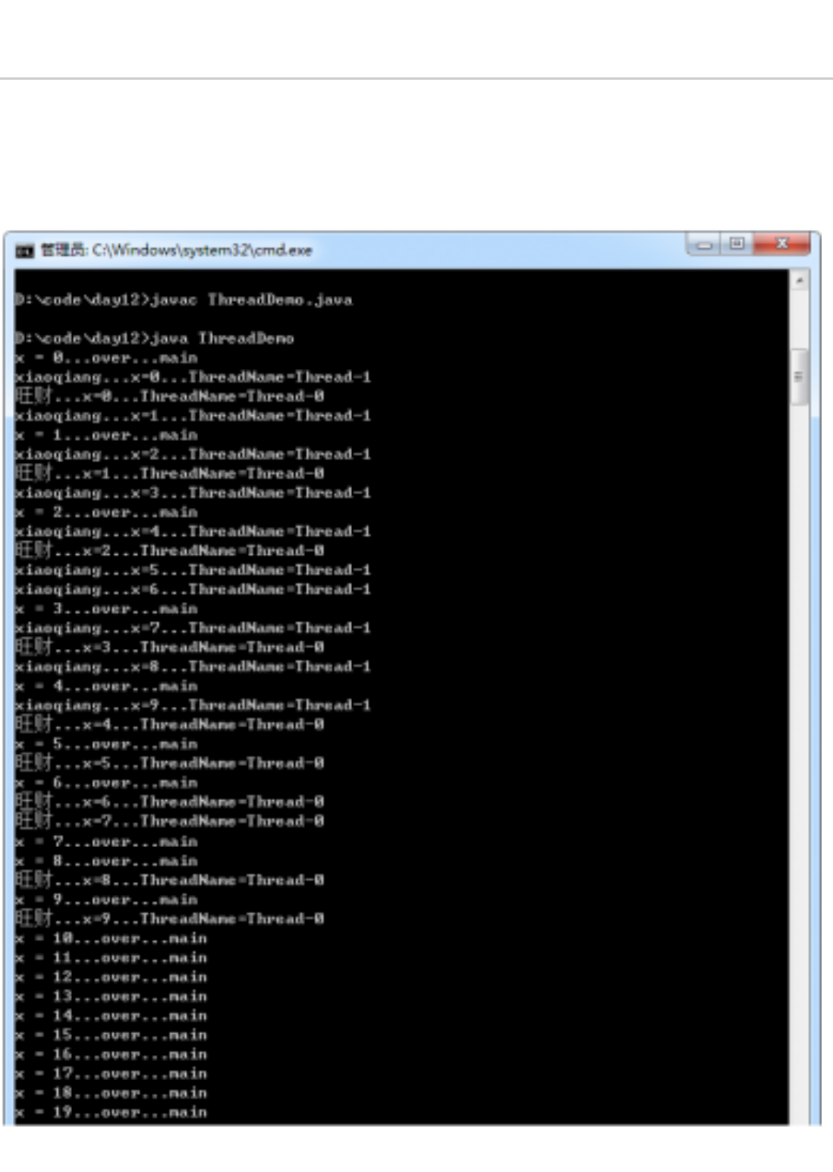
多线程程序示例：

```
class Demo extends Thread{
      private String name ;
      Demo(String name){
             this.name = name;
      }
       public void run(){
             for(int x = 0; x < 10; x++){
                  System.out.println(name + "...x=" + x + "...ThreadName=" +
Thread.currentThread ().getName());
            }
      }
```

```
}

class ThreadDemo{
        public static void main(String[] args){
                Demo d1 = new Demo("旺财");
                Demo d2 = new Demo("xiaoqiang");
                d1.start(); //开启线程，调用run方法。
                d2.start();
                for(int x = 0; x < 20; x++){
                        System.out.println("x = " + x + "...over..." +
Thread.currentThread().getName());
                }
        }
}
```

复制代码

运行结果：

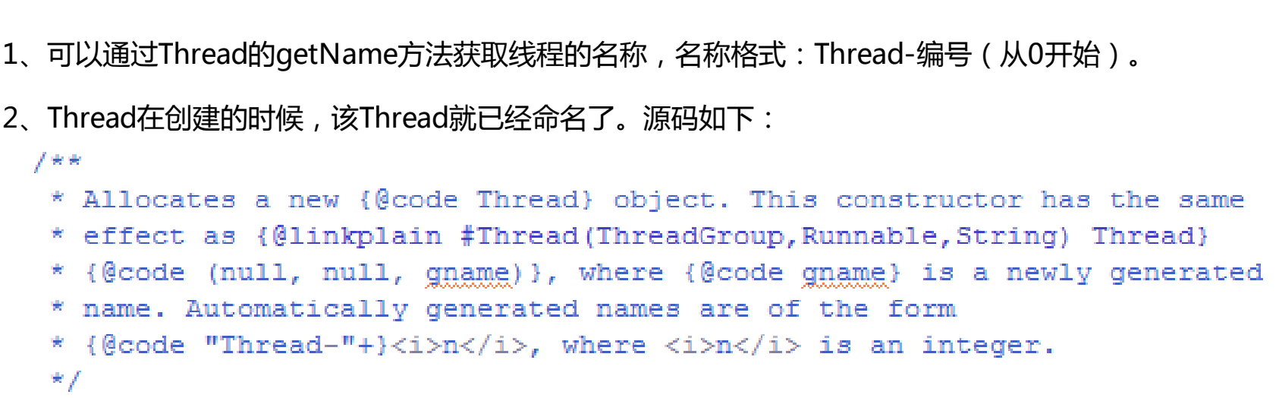


```
D:\code\day12>javac ThreadDemo.java

D:\code\day12>java ThreadDemo
x = 0...over...main
xiaoqiang...x=0...ThreadName=Thread-1
旺财...x=0...ThreadName=Thread-0
xiaoqiang...x=1...ThreadName=Thread-1
x = 1...over...main
xiaoqiang...x=2...ThreadName=Thread-1
旺财...x=1...ThreadName=Thread-0
xiaoqiang...x=3...ThreadName=Thread-1
x = 2...over...main
xiaoqiang...x=4...ThreadName=Thread-1
旺财...x=2...ThreadName=Thread-0
xiaoqiang...x=5...ThreadName=Thread-1
xiaoqiang...x=6...ThreadName=Thread-1
x = 3...over...main
xiaoqiang...x=7...ThreadName=Thread-1
旺财...x=3...ThreadName=Thread-0
xiaoqiang...x=8...ThreadName=Thread-1
x = 4...over...main
xiaoqiang...x=9...ThreadName=Thread-1
旺财...x=4...ThreadName=Thread-0
x = 5...over...main
旺财...x=5...ThreadName=Thread-0
x = 6...over...main
旺财...x=6...ThreadName=Thread-0
旺财...x=7...ThreadName=Thread-0
x = 7...over...main
x = 8...over...main
旺财...x=8...ThreadName=Thread-0
x = 9...over...main
旺财...x=9...ThreadName=Thread-0
x = 10...over...main
x = 11...over...main
x = 12...over...main
x = 13...over...main
x = 14...over...main
x = 15...over...main
x = 16...over...main
x = 17...over...main
x = 18...over...main
x = 19...over...main
```

P.S.

1、可以通过Thread的getName方法获取线程的名称，名称格式：Thread-编号（从0开始）。

2、Thread在创建的时候，该Thread就已经命名了。源码如下：



```
/**
 * Allocates a new {@code Thread} object. This constructor has the same
 * effect as {@linkplain #Thread(ThreadGroup,Runnable,String) Thread}
 * {@code (null, null, gname)}, where {@code gname} is a newly generated
 * name. Automatically generated names are of the form
 * {@code "Thread-"+}<i>n</i>, where <i>n</i> is an integer.
 */
public Thread() {
    init(null, null, "Thread-" + nextThreadNum(), 0);
}
```

~END~

~爱上海，爱黑马~