<http://lavasoft.blog.51cto.com/62575/99151/>

**一、定义线程**

1、扩展java.lang.Thread类。

此类中有个run()方法，应该注意其用法：

public void **run**()

如果该线程是使用独立的 Runnable 运行对象构造的，则调用该 Runnable 对象的 run 方法；否则，该方法不执行任何操作并返回。

Thread 的子类应该重写该方法。

2、实现java.lang.Runnable接口。

void **run**()

使用实现接口 Runnable 的对象创建一个线程时，启动该线程将导致在独立执行的线程中调用对象的 run 方法。

方法 run 的常规协定是，它可能执行任何所需的操作。

**二、实例化线程**

1、如果是扩展java.lang.Thread类的线程，则直接new即可。

2、如果是实现了java.lang.Runnable接口的类，则用Thread的构造方法：

Thread(Runnable target)   
Thread(Runnable target, String name)   
Thread(ThreadGroup group, Runnable target)   
Thread(ThreadGroup group, Runnable target, String name)   
Thread(ThreadGroup group, Runnable target, String name, long stackSize)

**三、启动线程**

在线程的Thread对象上调用start()方法，而不是run()或者别的方法。

在调用start()方法之前：线程处于新状态中，新状态指有一个Thread对象，但还没有一个真正的线程。

在调用start()方法之后：发生了一系列复杂的事情

启动新的执行线程（具有新的调用栈）；

该线程从新状态转移到可运行状态；

当该线程获得机会执行时，其目标run()方法将运行。

注意：对Java来说，run()方法没有任何特别之处。像main()方法一样，它只是新线程知道调用的方法名称(和签名)。因此，在Runnable上或者Thread上调用run方法是合法的。但并不启动新的线程。

**四、例子**

1、实现Runnable接口的多线程例子

/\*\*   
\* 实现Runnable接口的类   
\*   
\* @author leizhimin 2008-9-13 18:12:10   
\*/   
public class DoSomething implements Runnable {   
    private String name;   
  
    public DoSomething(String name) {   
        this.name = name;   
    }   
  
    public void run() {   
        for (int i = 0; i < 5; i++) {   
            for (long k = 0; k < 100000000; k++) ;   
            System.out.println(name + ": " + i);   
        }   
    }   
}

/\*\*   
\* 测试Runnable类实现的多线程程序   
\*   
\* @author leizhimin 2008-9-13 18:15:02   
\*/   
public class TestRunnable {   
    public static void main(String[] args) {   
        DoSomething ds1 = new DoSomething("阿三");   
        DoSomething ds2 = new DoSomething("李四");   
  
        Thread t1 = new Thread(ds1);   
        Thread t2 = new Thread(ds2);   
  
        t1.start();   
        t2.start();   
    }   
}

执行结果：

李四: 0   
阿三: 0   
李四: 1   
阿三: 1   
李四: 2   
李四: 3   
阿三: 2   
李四: 4   
阿三: 3   
阿三: 4   
  
Process finished with exit code 0

2、扩展Thread类实现的多线程例子

/\*\*   
\* 测试扩展Thread类实现的多线程程序   
\*   
\* @author leizhimin 2008-9-13 18:22:13   
\*/   
public class TestThread extends Thread{   
    public TestThread(String name) {   
        super(name);   
    }   
  
    public void run() {   
        for(int i = 0;i<5;i++){   
            for(long k= 0; k <100000000;k++);   
            System.out.println(this.getName()+" :"+i);   
        }   
    }   
  
    public static void main(String[] args) {   
        Thread t1 = new TestThread("阿三");   
        Thread t2 = new TestThread("李四");   
        t1.start();   
        t2.start();   
    }   
}

执行结果：

阿三 :0   
李四 :0   
阿三 :1   
李四 :1   
阿三 :2   
李四 :2   
阿三 :3   
阿三 :4   
李四 :3   
李四 :4   
  
Process finished with exit code 0

对于上面的多线程程序代码来说，输出的结果是不确定的。其中的一条语句for(long k= 0; k <100000000;k++);是用来模拟一个非常耗时的操作的。

**五、一些常见问题**

1、线程的名字，一个运行中的线程总是有名字的，名字有两个来源，一个是虚拟机自己给的名字，一个是你自己的定的名字。在没有指定线程名字的情况下，虚拟机总会为线程指定名字，并且主线程的名字总是mian，非主线程的名字不确定。

2、线程都可以设置名字，也可以获取线程的名字，连主线程也不例外。

3、获取当前线程的对象的方法是：Thread.currentThread()；

4、在上面的代码中，只能保证：每个线程都将启动，每个线程都将运行直到完成。一系列线程以某种顺序启动并不意味着将按该顺序执行。对于任何一组启动的线程来说，调度程序不能保证其执行次序，持续时间也无法保证。

5、当线程目标run()方法结束时该线程完成。

6、一旦线程启动，它就永远不能再重新启动。只有一个新的线程可以被启动，并且只能一次。一个可运行的线程或死线程可以被重新启动。

7、线程的调度是JVM的一部分，在一个CPU的机器上上，实际上一次只能运行一个线程。一次只有一个线程栈执行。JVM线程调度程序决定实际运行哪个处于可运行状态的线程。

众多可运行线程中的某一个会被选中做为当前线程。可运行线程被选择运行的顺序是没有保障的。

8、尽管通常采用队列形式，但这是没有保障的。队列形式是指当一个线程完成“一轮”时，它移到可运行队列的尾部等待，直到它最终排队到该队列的前端为止，它才能被再次选中。事实上，我们把它称为可运行池而不是一个可运行队列，目的是帮助认识线程并不都是以某种有保障的顺序排列唱呢个一个队列的事实。

9、尽管我们没有无法控制线程调度程序，但可以通过别的方式来影响线程调度的方式。

本文出自 “[熔 岩](http://lavasoft.blog.51cto.com/)” 博客，请务必保留此出处<http://lavasoft.blog.51cto.com/62575/99151>