# 1. 进程和线程概述

每个运行的程序相当于一个“进程”，现在的操作系统是多进程的。比如我们可以同时运行“网易云音乐”和浏览器，边听歌曲边上网。其实CPU在某个时刻只能运行一个程序，只不过是操作系统在安排CPU频繁的切换所运行的进程，由于速度很快，所以人们感觉多个程序是同时进行的。

进程是系统进行资源分配和调用的一个基本单位，多进程提高了CPU的利用率。

而在一个进程中，又可以运行多个线程，这也是通过抢占CPU时间来运行的。我们以前写的程序只有一个执行流程，就是单线程程序。一个程序中也能“同时”进行多个执行流程，这就是多线程程序。比如一个安全软件能同时进行病毒查杀和电脑清理，是两个流程在“同时”进行，因此应用了多线程。

线程是依赖于进程的。对于开发者，主要考虑的是本程序的执行流程，也就是多线程。比如编写一个GUI程序，当进行到某一步时，此步骤需要执行大量计算并且耗时，如果使用单线程，那么界面此时是无响应的，必须等此步骤执行完毕，界面才有响应。若采用多线程，可另外开辟一个线程专门用于计算，此时该线程和界面线程抢占CPU时间，由于切换速度极快，因此用户不会感到界面无响应。

多进程和多线程都是并发，多进程是操作系统层面的并发，多线程是当前进程的并发，既然有并发，必然需要考虑处理多线程的“隐患”，比如线程安全、同步、死锁等问题，这个后面会讲（建议了解操作系统知识）。而对于开发者，考虑的主要就是本进程中的多线程。

# 2. 编写多线程程序

对于单线程程序，由java命令运行程序，运行程序时就启动JVM进程，接着由该JVM进程创建一个主线程调用main()方法。

在程序中创建新线程需要使用Thread类，通过查看此类的API，可知有两种方式创建新线程。下面进行讲解。

## 2.1 方式1：自定义类继承Thread类

通过自定义类继承Thread类实现多线程，步骤如下：

（1）新建类继承自Thread；

（2）重写run()方法，此方法中的代码就是该新线程需要执行的代码。

（3）需要创建并运行此线程时，就创建此类对象，并调用start()方法开始线程。这样新线程一旦获得CPU时间就会执行run()中的代码。

例子：

（1）自定义类继承Thread并重写run()方法

|  |
| --- |
| **package** com.zhang.test;  **public class** MyThread **extends** Thread {  @Override  **public void** run() {  *// 执行1000次输出* **for**(**int** i = 0; i < 1000; i++) {  System.***out***.println(i);  }  } } |

（2）主类创建多个线程

|  |
| --- |
| **package** com.zhang.test;  **public class** Demo {  **public static void** main(String[] args) {  *// 创建两个MyThread线程，让他们运行看效果* MyThread t1 = **new** MyThread();  MyThread t2 = **new** MyThread();  t1.start();  t2.start();  } } |

运行结果发现，两遍0-999的数字很可能会交叉输出，这就是多线程。如果是单线程的话，必定是先输出完一遍0-999，再输出一遍0-999，而不会交叉输出。如果看不到交叉输出的效果，可把循环次数再变大，因为很可能CPU执行很快，在一个线程首次得到CPU时就把0-999全部输出完毕了。

注意启动线程调用的是start()方法，而不是run()方法，若调用run()方法，那么JVM认为这是一个普通的方法调用，是没有多线程效果的。注意，一个线程对象只能启用一次，即只能调用一次start()方法，若调用一次以上，就会抛出“IllegalThreadStateException”异常。

当然，可以直接用匿名内部类的方式创建并开始一个线程，例如：

|  |
| --- |
| **package** com.zhang.test;  **public class** Demo {  **public static void** main(String[] args) {  **new** Thread() {  @Override  **public void** run() {  System.***out***.println(**"我是一个新线程"**);  }  }.start();  } } |

## 2.2 方式2：实现Runnable接口

此方法的步骤如下：

（1）定义一个类实现Runnable接口。同样重写其中的run()方法。

（2）用Thread类的一个重载创建对象，将Runnable接口实现类传递进Thread构造中。

（3）调用Thread对象的start()方法。

接口实现多线程的好处：避免由于Java单继承带来的局限性，这样可以使Runnable的实现类继承其他的类。使用第一种方式的话，自定义类只能继承Thread类了。

例子：

（1）MyThread实现Runnable：

|  |
| --- |
| **package** com.zhang.test;  **public class** MyThread **implements** Runnable {  @Override  **public void** run() {  **for**(**int** i = 0; i < 1000; i++) {  System.***out***.println(i);  }  } } |

（2）主类

|  |
| --- |
| **package** com.zhang.test;  **public class** Demo {  **public static void** main(String[] args) {  MyThread mt = **new** MyThread();  Thread t1 = **new** Thread(mt); *// 将Runnable实现类传递进去* Thread t2 = **new** Thread(mt);  *// 启动* t1.start();  t2.start();  } } |

也可以通过匿名内部类实现：

|  |
| --- |
| **package** com.zhang.test;  **public class** Demo {  **public static void** main(String[] args) {  **new** Thread(**new** Runnable() {  @Override  **public void** run() {  System.***out***.println(**"我是一个新线程"**);  }  }).start();  } } |

那么就有个很有趣的问题，使用new Thread()方式，既能在括号中传递一个实现Runnable的匿名内部类，把new Thread看成对象，也能将new Thread()本身当做匿名内部类，重写run方法。即：

|  |
| --- |
| **package** com.zhang.test;  **public class** Demo {  **public static void** main(String[] args) {  **new** Thread(**new** Runnable() {  @Override  **public void** run() {  System.***out***.println(**"这是2"**);  }  }){  @Override  **public void** run() {  System.***out***.println(**"这是1"**);  }  }.start();  } } |

此时输出结果是“这是1”，因为start()还是被Thread的匿名内部类对象调用的，传递的Runnable匿名内部类没有什么作用。不过也没人用这种方式，只是好玩。

# 3. 线程常用方法

## 3.1 普通操作

下面是Thread类常用的方法：

（1）public static Thread currentThread()：静态方法，返回当前正在执行的线程对象。

（2）public final String getName()：获得线程名称。默认线程的名称是“Thread-序号”，序号是JVM定的。

（3）public final void setName(String name)：设置线程的名称。此方法应该在start()线程之前操作。

对于线程的名称，可在创建线程时就设置，因为有构造方法：

Thread()、Thread(Runnable target)、Thread(Runnable target, String name)、Thread(String name)。如果是自定义线程类，可以借用父类的Thread(String)构造。

例如：

|  |
| --- |
| **package** com.zhang.test;  **public class** MyThread **extends** Thread {  **public** MyThread(String name) {  **super**(name);  }  @Override  **public void** run() {  System.***out***.println(**"此线程名称"** + **this**.getName());  } } |

然后MyThread也能通过new MyThread(“名字”)设置线程名了。若想获得主线程名，自然是通过Thread.currentThread().getName()的方式，主线程的名字就是“main”。

如果上述代码采用的是实现Runnable接口，就没法用this.getName()了，那么就用Thread.currentThread().getName()。

（4）设置和获取线程的优先级：

public final int getPriority()：获得线程优先级，默认是5，

public final void setPriority(int new Priority)：设置线程优先级。优先级从低到高可设置的范围是1-10。优先级就是一种抢占式调度模型，优先让优先级高的线程使用 CPU。

## 3.2 操作线程方法

（1）线程休眠：public static void sleep(long millis)：在指定的毫秒数内让当前正在执行的线程休眠（暂停执行），注意这是静态方法。

（2）设置后台线程（守护线程）：public final void setDaemon(boolean on)：如果设置为 true，则将该线程标记为守护线程。将该线程标记为守护线程或用户线程。当正在运行的线程都是守护线程时，Java 虚拟机退出。该方法必须在启动线程前调用。

守护进程就是在后台默默运行的线程，当所有的非后台线程结束时，程序也就终止了，同时还会杀死进程中的所有后台线程，也就是说，只要有非后台线程还在运行，程序就不会终止，执行main方法的主线程就是一个非后台线程。

如果不设置后台线程，那么就当所有的线程执行完后程序才结束。比如一个GUI程序，可能有很多后台线程，那么当主界面关闭后，程序就会终止，后台线程也会随着主线程关闭而被结束。如果此程序中都是用户线程，那么即使用户关闭了主界面，很可能程序也不会被关闭，还有线程在执行。这就是setDaemon()的特点。