如有你有帮助，请购买下载，谢谢！

# Java 多线程编程详解

## 一：理解多线程

多线程是这样一种机制,它允许在程序中并发执行多个指令流,每个指令流都称为一个线程,彼此间互相独立。

线程又称为轻量级进程，它和进程一样拥有独立的执行控制，由操作系统负责调度，区别在于线程没有独立的存储空间，而是和所属进程中的其它线程共享一个存储空间，这使得线程间的通信远较进程简单。

多个线程的执行是并发的，也就是在逻辑上“同时”，而不管是否是物理上的“同时”。如果系统只有一个 CPU，那么真正的“同时”是不可能的，但是由于 CPU 的速度非常快，用户感觉不到其中的区别，因此我们也不用关心它，只需要设想各个线程是同时执行即可。

多线程和传统的单线程在程序设计上最大的区别在于，由于各个线程的控制流彼此独立，使得各个线程之间的代码是乱序执行的，由此带来的线程调度，同步等问题，将在以后探讨。

## 二：在 Java 中实现多线程

我们不妨设想，为了创建一个新的线程，我们需要做些什么？很显然，我们必须指明这个线程所要执行的代码，而这就是在 Java 中实现多线程我们所需要做的一切！

真是神奇！Java 是如何做到这一点的？通过类！作为一个完全面向对象的语言，Java 提供了类 来方便多线程编程，这个类提供了大量的方法来方便我们控制自己的各个线程，我们以后的讨论都将围绕这个类进行。

那么如何提供给 Java 我们要线程执行的代码呢？让我们来看一看 Thread 类。Thread 类最重要的方法是 run() ，它为 Thread 类的方法 start() 所调用，提供我们的线程所要执行的代码。为了指定我们自己的代码，只需要覆盖它！

### 方法一：继承 Thread 类，覆盖方法 run()

我们在创建的 Thread 类的子类中重写 run()， 加入线程所要执行的代码即可。

下面是一个例子：

```
public class MyThread extends Thread {
int count= 1, number;
public MyThread(int num) {
number = num;
" 创 建 线 程 " + number);
}
public void run() {
while(true) {
" 线 程 " + number + ":  计 数 " + count);
```

```
if(++count== 6) return;
}
}
public static void main(String args[]) {
for(int i = 0; i < 5; i++) new MyThread(i+1).start();
}
}
```

这种方法简单明了，符合大家的习惯，但是，它也有一个很大的缺点，那就是如果我们的类已经从一个类继承（如小程序必须继承自 Applet 类），则无法再继承 Thread 类，这时如果我们又不想建立一个新的类，应该怎么办呢？

我们不妨来探索一种新的方法：我们不创建 Thread 类的子类，而是直接使用它，那么我们只能将我们的方法作为参数传递给 Thread 类的实例，有点类似回调函数。但是 Java 没有指针，我们只能传递一个包含这个方法的类的实例。那么如何限制这个类必须包含这一方法呢？当然是使用接口！（虽然抽象类也可满足，但是需要继承，而我们之所以要采用这种新方法，不就是为了避免继承带来的限制吗？）

Java 提供了接口 来支持这种方法。

方法二：实现 Runnable 接口

Runnable 接口只有一个方法 run()，我们声明自己的类实现 Runnable 接口并提供这一方法，将我们的线程代码写入其中，就完成了这一部分的任务。

但是 Runnable 接口并没有任何对线程的支持，我们还必须创建 Thread 类的实例，这一点通过 Thread 类的构造函数

public Thread(Runnable target); 来实现。

下面是一个例子：

```
public class MyThread implements Runnable {
int count= 1, number;
public MyThread(int num) {
number = num;
" 创建线程 " + number);
}
public void run() {
while(true) {
" 线程 " + number + ": 计数 " + count);
if(++count== 6) return;
}
}
public static void main(String args[]) {
```

```
for(int i = 0; i < 5; i++) new Thread(new MyThread(i+1)).start();
}
}
```

严格地说，创建 Thread 子类的实例也是可行的，但是必须注意的是，该子类必须没有覆盖 Thread 类的 run 方法，否则该线程执行的将是子类的 run 方法，而不是我们用以实现 Runnable 接口的类的 run 方法，对此大家不妨试验一下。

使用 Runnable 接口来实现多线程使得我们能够在一个类中包容所有的代码，有利于封装，它的缺点在于，我们只能使用一套代码，若想创建多个线程并使各个线程执行不同的代码，则仍必须额外创建类，如果这样的话，在大多数情况下也许还不如直接用多个类分别继承 Thread 来得紧凑。

综上所述，两种方法各有千秋，大家可以灵活运用。

下面让我们一起来研究一下多线程使用中的一些问题。

三：线程的四种状态

1. 新状态：线程已被创建但尚未执行（start() 尚未被调用）。
2. 可执行状态：线程可以执行，虽然不一定正在执行。CPU 时间随时可能被分配给该线程，从而使得它执行。
3. 死亡状态：正常情况下 run() 返回使得线程死亡。调用 stop() 或 destroy() 亦有同样效果，但是不被推荐，前者会产生异常，后者是强制终止，不会释放锁。
4. 阻塞状态：线程不会被分配 CPU 时间，无法执行。

四：线程的优先级

线程的优先级代表该线程的重要程度，当有多个线程同时处于可执行状态并等待获得 CPU 时间时，线程调度系统根据各个线程的优先级来决定给谁分配 CPU 时间，优先级高的线程有更大的机会获得 CPU 时间，优先级低的线程也不是没有机会，只是机会要小一些罢了。

你可以调用 Thread 类的方法 getPriority() 和 setPriority() 来存取线程的优先级，线程的优先级界于 1(MIN_PRIORITY) 和 10(MAX_PRIORITY) 之间，缺省是 5(NORM_PRIORITY) 。

五：线程的同步

由于同一进程的多个线程共享同一片存储空间，在带来方便的同时，也带来了访问冲突这个严重的问题。Java 语言提供了专门机制以解决这种冲突，有效避免了同一个数据对象被多个线程同时访问。

由于我们可以通过 private 关键字来保证数据对象只能被方法访问，所以我们只需针对方法提出一套机制，这套机制就是 synchronized 关键字，它包括两种用法：synchronized 方法和 synchronized 块。

1. synchronized 方法：通过在方法声明中加入 synchronized 关键字来声明 synchronized 方法。如：