java线程中run和start方法的区别

1.start（）方法来启动线程，真正实现了多线程运行，这时无需等待run方法体代码执行完毕而直接继续执行下面的代码：

通过调用Thread类的start()方法来启动一个线程，

这时此线程是处于就绪状态，

并没有运行。

然后通过此Thread类调用方法run()来完成其运行操作的，

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，

此线程终止，

而CPU再运行其它线程。

2.run（）方法当作普通方法的方式调用，程序还是要顺序执行，还是要等待run方法体执行完毕后才可继续执行下面的代码：

而如果直接用Run方法，

这只是调用一个方法而已，

程序中依然只有主线程--这一个线程，

其程序执行路径还是只有一条，

这样就没有达到写线程的目的。

终止线程的三种方法

1. 使用退出标志，使线程正常退出，也就是当run方法完成后线程终止。

2. 使用stop方法强行终止线程（这个方法不推荐使用，因为stop和suspend、resume一样，也可能发生不可预料的结果）。

3. 使用interrupt方法中断线程。

使用interrupt方法来终端线程可分为两种情况：

（1）线程处于阻塞状态，如使用了sleep方法。

（2）使用while（！isInterrupted（））{……}来判断线程是否被中断。

在第一种情况下使用interrupt方法，sleep方法将抛出一个InterruptedException例外，而在第二种情况下线程将直接退出。下面的代码演示了在第一种情况下使用interrupt方法

wait() 与notify()/notifyAll()

这三个方法都是Object的方法，并不是线程的方法！

wait():释放占有的对象锁，线程进入等待池，释放cpu,而其他正在等待的线程即可抢占此锁，获得锁的线程即可运行程序。而sleep()不同的是，线程调用此方法后，会休眠一段时间，休眠期间，会暂时释放cpu，但并不释放对象锁。也就是说，在休眠期间，其他线程依然无法进入此代码内部。休眠结束，线程重新获得cpu,执行代码。wait()和sleep()最大的不同在于wait()会释放对象锁，而sleep()不会！

notify(): 该方法会唤醒因为调用对象的wait()而等待的线程，其实就是对对象锁的唤醒，从而使得wait()的线程可以有机会获取对象锁。调用notify()后，并不会立即释放锁，而是继续执行当前代码，直到synchronized中的代码全部执行完毕，才会释放对象锁。JVM则会在等待的线程中调度一个线程去获得对象锁，执行代码。需要注意的是，wait()和notify()必须在synchronized代码块中调用。

notifyAll()则是唤醒所有等待的线程

4、volatile与synchronized

1）volatile本质是在告诉jvm当前变量在寄存器中的值是不确定的,需要从主存中读取,synchronized则是锁定当前变量,只有当前线程可以访问该变量,其他线程被阻塞住.  
2）volatile仅能使用在变量级别,synchronized则可以使用在变量,方法.  
3）volatile仅能实现变量的修改可见性,而synchronized则可以保证变量的修改可见性和原子性.

　　《Java编程思想》上说，定义long或double变量时，如果使用volatile关键字，就会获得（简单的赋值与返回操作）原子性  
4）volatile不会造成线程的阻塞,而synchronized可能会造成线程的阻塞.

5、当一个域的值依赖于它之前的值时，volatile就无法工作了，如n=n+1,n++等。如果某个域的值受到其他域的值的限制，那么volatile也无法工作，如Range类的lower和upper边界，必须遵循lower<=upper的限制。

6、使用volatile而不是synchronized的唯一安全的情况是类中只有一个可变的域。

**请叙述一下您对线程池的理解？**（2015-11-25）

（如果问到了这样的问题，可以展开的说一下线程池如何用、线程池的好处、线程池的启动策略）

合理利用线程池能够带来三个好处。

第一：降低资源消耗。通过重复利用已创建的线程降低线程创建和销毁造成的消耗。

第二：提高响应速度。当任务到达时，任务可以不需要等到线程创建就能立即执行。

第三：提高线程的可管理性。线程是稀缺资源，如果无限制的创建，不仅会消耗系统资源，还会降低系统的稳定性，使用线程池可以进行统一的分配，调优和监控

**Android中的四类线程池**

Android中最常见的四类具有不同特性的线程池分别为FixThreadPool、CachedThreadPool、ScheduleThreadPool以及SingleThreadExecutor。

newCachedThreadPool创建一个可缓存线程池，如果线程池长度超过处理需要，可灵活回收空闲线程，若无可回收，则新建线程。  
newFixedThreadPool 创建一个定长线程池，可控制线程最大并发数，超出的线程会在队列中等待。  
newScheduledThreadPool 创建一个定长线程池，支持定时及周期性任务执行。  
newSingleThreadExecutor 创建一个单线程化的线程池，它只会用唯一的工作线程来执行任务，保证所有任务按照指定顺序(FIFO, LIFO, 优先级)执行。

使用demo

一 ExecutorService fixedThreadPool = Executors.newFixedThreadPool(3);

二

for (int i = 0; i < 10; i++) {

final int index = i;

fixedThreadPool.execute(new Runnable() {

public void run() {

try {

System.out.println(index);

Thread.sleep(2000);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

});

}