## Java创建线程之后，直接调用start()方法和run()的区别

答：

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，可以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。然后CPU再调度其它线程。

2.run（）方法当作普通方法的方式调用。程序还是要顺序执行，要等待run方法体执行完毕后，才可继续执行下面的代码； 程序中只有主线程——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。

记住：多线程就是分时利用CPU，宏观上让所有线程一起执行 ，也叫并发

## 常用的线程池模式以及不同线程池的使用场景

答： 由于在Thread-Per-Message Pattern中对于每一个请求都会生成启动一个线程，而线程的启动是很花费时间的工作，所以鉴于此，提出了Worker Thread，重复利用已经启动的线程。

线程池：

Worker Thread，也称为工人线程或背景线程，不过一般都称为线程池。该模式主要在于，事先启动一定数目的工作线程。当没有请求工作的时候，所有的工人线程都会等待新的请求过来，一旦有工作到达，就马上从线程池中唤醒某个线程来执行任务，执行完毕后继续在线程池中等待任务池的工作请求的到达。

任务池：主要是存储接受请求的集合，利用它可以缓冲接受到的请求，可以设置大小来表示同时能够接受最大请求数目。这个任务池主要是供线程池来访问。

线程池：这个是工作线程所在的集合，可以通过设置它的大小来提供并发处理的工作量。对于线程池的大小，可以事先生成一定数目的线程，根据实际情况来动态增加或者减少线程数目。线程池的大小不是越大越好，线程的切换也会耗时的。

Worker Thread的所有参与者：

　1，Client参与者，发送Request的参与者

　2，Channel参与者，负责缓存Request的请求，初始化启动线程，分配工作线程

　3，Worker参与者，具体执行Request的工作线程

　4，Request参与者

注意：

将在Worker线程内部等待任务池非空的方式称为正向等待。

将在Channel线程提供Worker线程来判断任务池非空的方式称为反向等待。