**Android中为什么主线程不会因为Looper.loop()里的死循环卡死？**

这里涉及线程，先说说说进程/线程，**进程：**每个app运行时前首先创建一个进程，该进程是由Zygote fork出来的，用于承载App上运行的各种Activity/Service等组件。进程对于上层应用来说是完全透明的，这也是google有意为之，让App程序都是运行在Android Runtime。大多数情况一个App就运行在一个进程中，除非在AndroidManifest.xml中配置Android:process属性，或通过native代码fork进程。

**线程：**线程对应用来说非常常见，比如每次new Thread().start都会创建一个新的线程。该线程与App所在进程之间资源共享，从Linux角度来说进程与线程除了是否共享资源外，并没有本质的区别，都是一个task\_struct结构体**，在CPU看来进程或线程无非就是一段可执行的代码，CPU采用CFS调度算法，保证每个task都尽可能公平的享有CPU时间片**

有了这么准备，再说说死循环问题：

对于线程既然是一段可执行的代码，当可执行代码执行完成后，线程生命周期便该终止了，线程退出。而对于主线程，我们是绝不希望会被运行一段时间，自己就退出，那么如何保证能一直存活呢？**简单做法就是可执行代码是能一直执行下去的，死循环便能保证不会被退出，**例如，binder线程也是采用死循环的方法，通过循环方式不同与Binder驱动进行读写操作，当然并非简单地死循环，无消息时会休眠。但这里可能又引发了另一个问题，既然是死循环又如何去处理其他事务呢？通过创建新线程的方式。

真正会卡死主线程的操作是在回调方法onCreate/onStart/onResume等操作时间过长，会导致掉帧，甚至发生ANR，looper.loop本身不会导致应用卡死。

**Activity的生命周期是怎么实现在死循环体外能够执行起来的？**

ActivityThread的内部类H继承于Handler，通过handler消息机制，简单说Handler机制用于同一个进程的线程间通信。

**Activity的生命周期都是依靠主线程的Looper.loop，当收到不同Message时则采用相应措施：**  
在H.handleMessage(msg)方法中，根据接收到不同的msg，执行相应的生命周期。

比如收到msg=H.LAUNCH\_ACTIVITY，则调用ActivityThread.handleLaunchActivity()方法，最终会通过反射机制，创建Activity实例，然后再执行Activity.onCreate()等方法；  
再比如收到msg=H.PAUSE\_ACTIVITY，则调用ActivityThread.handlePauseActivity()方法，最终会执行Activity.onPause()等方法。 上述过程，我只挑核心逻辑讲，真正该过程远比这复杂。

ActivityThread是应用程序的入口，这里你可以看到写Java程序时司空见惯的main方法，而main方法正是整个Java程序的入口。

ActivityThread的main方法主要就是做消息循环，一旦退出消息循环，那么你的程序也就可以退出了。

从消息队列中取消息可能会阻塞，取到消息会做出相应的处理。如果某个消息处理时间过长，就可能会影响UI线程的刷新速率，造成卡顿的现象。