# Understanding ActivityManagerService

## 应用程序的启动过程

当某一component需要运行的时候，如果此component所在的应用没有运行，那么系统就会为此应用分配一个进程，然后把此应用程序attach到此进程中，当此应用程序不在被需要时，系统会终止此进程；由于系统是把每个应用程序放在一个单独的进程中运行的，所以一个Application中的component不能直接启动运行另一个Application中的component，但是可以通过发送一个intent消息给系统，让系统去启动某一component。

下面分析一下，当启动一个Activity并且此Activity所在的应用没有运行的时候，所经过的路径(最好同时参考一下源码)：

* 应用通过调用某一个Activity对象的startActivity方法，试图启动另一个应用程序的Activity。
* Activity:startActivityForResult 对于普通的Activity(没有parent，也就是说此Activity不是嵌套在其他Activity中)，将会调用Instrumentation:execStartActivity
* Instrumentation:execStartActivity 当没有ActivityMonitor block此Activity时，会调用 ActivityManagerNative.getDefault().startActivity，最终调用的是ActivityManagerService:startActivity
* ActivityManagerService:startActivity
* ActivityStack:startActivityMayWait 做一些intent的resolve后继续调用下一函数
* ActivityStack:startActivityLocked(IApplicationThread caller,Intent intent, String resolvedType, Uri[] grantedUriPermissions, int grantedMode, ActivityInfo aInfo, IBinder resultTo, String resultWho, int requestCode, int callingPid, int callingUid, boolean onlyIfNeeded, boolean componentSpecified) 做一些鉴权，并创建ActivityRecord对象，然后转调ActivityStack:startActivityUncheckedLocked
* ActivityStack:startActivityUncheckedLocked 根据intent的flag，确定一些信息如：是否为此Activity启动一个新的task
* ActivityStack:startActivityLocked(ActivityRecord r, boolean newTask,boolean doResume) 为此Activity准备一些信息如: task, history, window等
* ActivityStack:resumeTopActivityLocked() 对其他的可见Activity做一些处理
* ActivityStack:startSpecificActivityLocked() 当对应的application不存在的时候，调用mService.startProcessLocked重新启动一个进程
* ActivityManagerService:startProcessLocked(String processName,ApplicationInfo info, boolean knownToBeDead, int intentFlags,String hostingType, ComponentName hostingName, boolean allowWhileBooting) 构建进程信息(ProcessRecord), 然后转调 startProcessLocked(app, hostingType, hostingNameStr);
* ActivityManagerService:startProcessLocked(ProcessRecord app,String hostingType, String hostingNameStr): 如此创建进程来运行ActivityThread，然后完善ProcessRecord；由此可见，ActivityThread是进程的主体

int pid = Process.start("android.app.ActivityThread",mSimpleProcessManagement ? app.processName : null, uid, uid,gids, debugFlags, null);

* Process:start
* Process:startViaZygote: 创建一个进程来运行ActivityThread:main
* Process:zygoteSendArgsAndGetPid: 这里是通过与zygote进程建立一个socket来通信的，让zygote根据传递的参数来建立一个进程。zygote进程中有一个loop，不断的接受新的connections请求，具体函数是ZygoteInit:runSelectLoopMethod，当有消息时，会调用ZygoteConnection:runOnce来进行参数解析，并调用用c实现的Zygote.forkAndSpecialize函数，该函数的实现在：dalvik\vm\native\dalvik\_system\_Zygote.c: Dalvik\_dalvik\_system\_Zygote\_forkAndSpecialize(const u4\* args,JValue\* pResult)此函数最后返回子进程的pid
* 再来看ActivityThread:main():
* ActivityThread:attach()
* ActivityManagerService:attachApplication()
* ActivityManagerService:attachApplicationLocked()
* ActivityStack:realStartActivityLocked()
* ActivityThread:ApplicationThread:scheduleLaunchActivity()
* ActivityThread:queueOrSendMessage(LAUNCH\_ACTIVITY, ...)此后LAUNCH\_ACTIVITY消息就由ActivityThread:H:handleMessage()来处理，此处理函数是在ActivityThread的loop对象的循环里的调用的。此handleMessage函数之后的调用链为：ActivityThread:handleLaunchActivity()-->ActivityThread:performLaunchActivity()
* Instrumentation:newActivity():这里完成了Activity的创建

## ActivityManagerService的静态继承树

