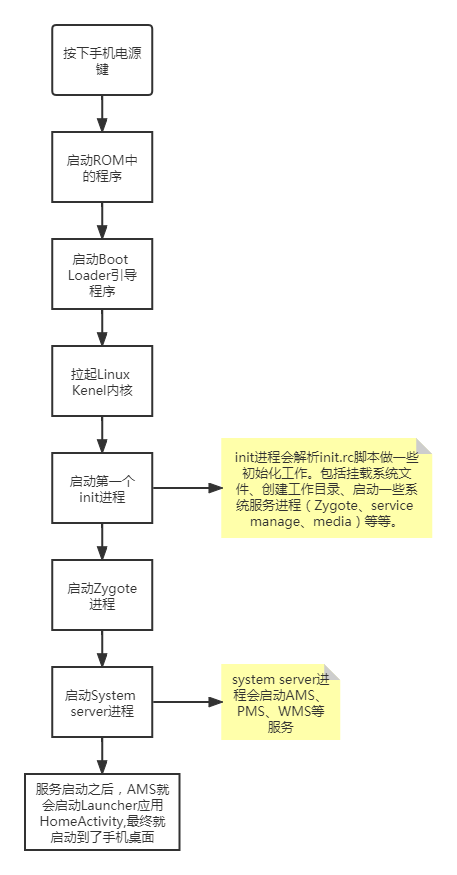
# Android启动流程

## 1.你了解 Android 系统启动流程吗？

首先，从你按下手机上的电源按钮开始，会执行在ROM上的一段程序，然后就会启动引导程序****BootLoader****到内存中，接着引导程序会启动底层Linux系统，Linux系统会启动第一个进程init进程。

然后init 进程会解析init.rc 脚本做一些初始化工作，包括挂载文件系统、创建工作目录以及启动系统服务进程等，其中系统服务进程包括 Zygote、service manager、media 等。在 Zygote 中会进一步去启动 system\_server 进程，然后在 system\_server 进程中会启动 AMS、WMS、PMS 等服务，等这些服务启动之后，AMS 中就会打开 Launcher 应用的 home Activity，最终就启动到了手机的 "桌面"。



## 2.system\_server 为什么要在 Zygote 中启动，而不是由 init 直接启动呢？

Zygote 作为一个孵化器，可以提前加载一些资源，这样 fork() 时基于 Copy-On-Write 机制创建的其他进程就能直接使用这些资源，而不用重新加载。比如 system\_server 就可以直接使用 Zygote 中的 JNI函数、共享库、常用的类、以及主题资源。

## 3.为什么要专门使用 Zygote 进程去孵化应用进程，而不是让 system\_server 去孵化呢？

首先 system\_server 相比 Zygote 多运行了 AMS、WMS 等服务，这些对一个应用程序来说是不需要的。另外进程的 fork() 对多线程不友好，fork只会会将发起调用的线程拷贝到子进程，这可能会导致死锁，而system\_server 中肯定是有很多线程的。

## 4.能说说具体是怎么导致死锁的吗？

在 POSIX 标准中，fork 的行为是这样的：复制整个用户空间的数据（通常使用 copy-on-write 的策略，所以可以实现的速度很快）以及所有系统对象，然后仅复制当前线程到子进程。这里：所有父进程中别的线程，到了子进程中都是突然蒸发掉的对于锁来说，从 OS 看，每个锁有一个所有者，即最后一次 lock 它的线程。假设这么一个环境，在 fork之前，有一个子线程 lock 了某个锁，获得了对锁的所有权。fork 以后，在子进程中，所有的额外线程都人间蒸发了。而锁却被正常复制了，在子进程看来，这个锁没有主人，所以没有任何人可以对它解锁。当子进程想 lock 这个锁时，不再有任何手段可以解开了。程序发生死锁。

## 5.Zygote 为什么不采用 Binder 机制进行 IPC 通信？

Binder 机制中存在 Binder 线程池，是多线程的，如果 Zygote 采用 Binder 的话就存在上面说的fork() 与 多线程的问题了。其实严格来说，Binder 机制不一定要多线程，所谓的 Binder 线程只不过是在循环读取 Binder 驱动的消息而已，只注册一个 Binder 线程也是可以工作的，比如 service manager就是这样的。实际上 Zygote 尽管没有采取 Binder 机制，它也不是单线程的，但它在 fork() 前主动停止了其他线程，fork() 后重新启动了。

## 6.名词解释

ROM:

只读存储器;程序存储器

copy-on-write：

写入时复制（英语：Copy-on-write，简称COW）是一种计算机程序设计领域的优化策略。其核心思想是，如果有多个调用者（callers）同时要求相同资源（如内存或磁盘上的数据存储），他们会共同获取相同的指针指向相同的资源，直到某个调用者试图修改资源的内容时，系统才会真正复制一份专用副本（private copy）给该调用者，而其他调用者所见到的最初的资源仍然保持不变。这过程对其他的调用者都是透明的（transparently）。此作法主要的优点是如果调用者没有修改该资源，就不会有副本（private copy）被创建，因此多个调用者只是读取操作时可以共享同一份资源。