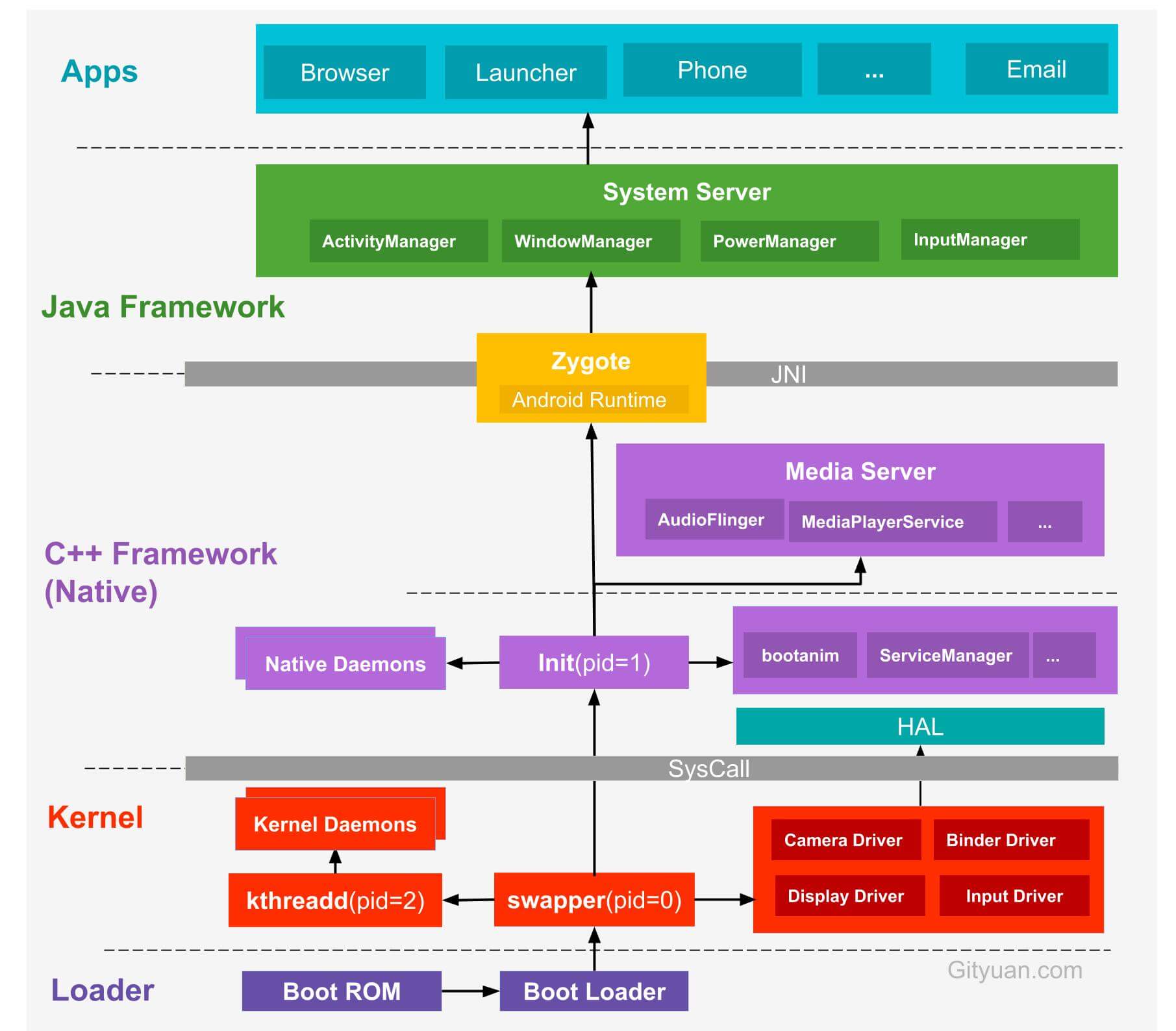
**引言**

本文讲解从开机到app显示画面的流程，但不分析源码，如果想阅读源码请到参考文章中查阅。

本文把这段流程分为三部分：

* 从开机到显示应用列表
* 从点击应用图标到Activity创建成功
* 从Activity创建成功到显示画面

**从开机到显示应用列表**

先看流程图：

开机加电后，CPU先执行**预设代码**、加载ROM中的引导程序**Bootloader**和Linux内核到RAM内存中去，然后初始化各种软硬件环境、加载驱动程序、挂载根文件系统，执行**init进程**。

init进程会启动各种系统本地服务，如**SM**（ServiceManager）、MS（Media Server）、bootanim（开机动画）等，然后init进程会在解析init.rc文件后fork()出**Zygote进程**。

Zygote会启动Java虚拟机，通过jni进入Zygote的java代码中，并创建**socket**实现IPC进程通讯，然后启动**SS**（SystemServer）进程。

SS进程负责启动和管理整个framework，包括**AMS**（ActivityManagerService）、**WMS**（WindowManagerService）、PMS（PowerManagerService）等服务、同时启动binder线程池，当SS进程将系统服务启动就绪以后，就会通知AMS启动Home。

AMS通过Intent隐式启动的方式启动**Launcher**，Launcher根据已安装应用解析对应的xml、通过findViewById()获得一个RecycleView、加载应用图标、最后成功展示App列表。

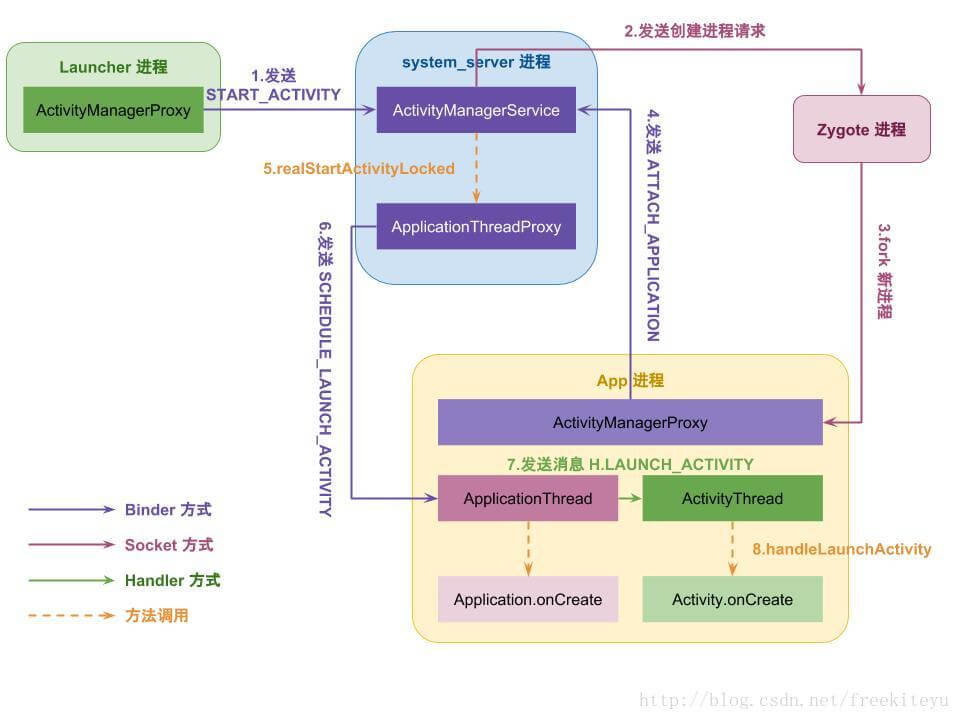
**解释**

* **预设代码**：cpu制造厂商会预设一个地址，这个地址是各厂家约定统一的，Android手机会将固态存储设备ROM预先映射到该地址上；
* **Bootloader**：类似BIOS，在系统加载前，用以初始化硬件设备，建立内存空间的映像图，为最终调用系统内核准备好环境；
* **init进程**：init进程时Android系统中用户进程的鼻祖进程，主要作用是启动系统本地服务、fork出Zygote进程；
* **SM**：ServiceManager是一个守护进程，它维护着系统服务和客户端的binder通信；
* **Zygote进程**：Zygote进程是所有Java进程的父进程，我们的APP都是由Zygote进程fork出来的；
* **socket**：一种独立于协议用于两个应用程序之间的数据传输的网络编程接口，是IPC中的一种；（但是在Android中一般使用Binder来实现IPC，这里使用socket的原因后面有写到）
* **SS**：Framework两大重要进程之一（另一个是Zygote），载着framework的核心服务，系统里面重要的服务都是SS开启的；
* **AMS**：服务端对象，负责系统中所有Activity的生命周期，打开App、Activity的开启、暂停、关闭都需要AMS来控制；
* **WMS**：窗口管理服务，窗口的启动、添加、删除、大小、层级都是由WMS管理；（下面会解释什么是窗口）
* **Launcher**：Launcher就是系统桌面，主要用来启动应用桌面，同时管理快捷方式和其他组件，本质上也是一个应用程序，和我们的App一样，也是继承自Activity，有自己的AndroidManifest；（所以才可以被AMS用Intent启动）

**Question 1： Zygote进程为什么使用Socket而不是Binder？** fork不允许存在多线程，而Binder通讯恰巧就是多线程；

**Question 2：什么是窗口？** Android系统中的窗体是屏幕上的一块用于绘制各种UI元素并能够响应应用户输入的一个矩形区域，从原理上来讲，窗体的概念是独自占有一个Surface实例的显示区域，比如Dialog、Activity的界面、壁纸、状态栏以及Toast等都是窗体；

**从点击应用图标到Activity创建成功**

先看流程图：

*//然后点击应用图标后，先检查要打卡的Activity是否存在*

**-->** **Launcher.**startActivitySafely**()**

**-->** **Launcher.**startActivity**()**

**-->** **Activity.**startActivity**()**

**-->** **Activity.**startActivityForResult**()**

*//然后获取AMS的代理AMP*

**-->** **Instrumentation.**execStartActivity**()**

**-->** **ActivityManagerNative.**getDefault**().**startActivity**()**

**-->** **ActivityManagerProxy.**startActivity**()**

**-->** **ActivityManagerService.**startActivity**()**

**-->** startActivityAsUser**(**intent**,** requestCode**,** userId**)**

**-->** **ActivityStackSupervisor.**startActivityMayWait**()**

**-->** **ActivityStackSupervisor.**resolveActivity**()**

**-->** **ActivityStackSupervisor.**startActivityLocked**()**

**-->** **new** **ActivityRecord对象**，获取ActivityStack

**-->** 找到ActivityStack后Launcher**.**onPause**()**

*//准备启动进程*

**-->** **ActivityManagerService.**startProcessLocked**()**

*//通过socket通知Zygote创建进程*

**-->** zygoteSendArgsAndGetResult**()**

*//创建ActivityThread*

**-->** **ActivityThread.**main**()**

*//告诉AMS我已经创建好了*

**-->** **ActivityThread.**attach**()**

**-->** **ActivityManagerProxy.**attachApplication**()**

**-->** **ActivityMangerService.**attachApplication**()**

*//找到Application实例并初始化*

**-->** **ActivityMangerService.**attachApplicationLocked**()**

**-->** **ApplicationThread.**bindApplication**()**

*//创建Application*

**-->** **AcitvityThread.**bindApplication**()**

**-->** **Application.**oncreate**()**

*//启动Activity*

**-->** **ActivityStackSupervisor.**attachApplicationLocked**()**

**-->** **ActivityStackSupervisor.**realStartActivityLocked**()**

**-->** **ActivityThread.**scheduleLaunchActivity**()**

*//进入UI线程*

**-->** handleLaunchActivity**()**

**-->** performLaunchActivity**()**

*//创建Activity实例*

**-->** **Instrumentation.**newActivity**()**

**-->** **Activity.**onCreate**()**

**解释**

* **ActivityThread**：App的真正入口。当开启App之后，会调用main()开始运行，开启消息循环队列，这就是传说中的UI线程或者叫主线程。与ActivityManagerServices配合，一起完成Activity的管理工作；
* **ApplicationThread**：用来实现ActivityManagerService与ActivityThread之间的交互。在ActivityManagerService需要管理相关Application中的Activity的生命周期时，通过ApplicationThread的代理对象与ActivityThread通讯；
* **Instrumentation**：可以理解为应用进程的管家，每个应用程序只有一个，每个Activity内都有该对象的引用，ActivityThread要创建或暂停某个Activity时，都需要通过Instrumentation来进行具体的操作；
* **ActivityStack**：Activity在AMS的栈管理，用来记录已经启动的Activity的先后关系，状态信息等。通过ActivityStack决定是否需要启动新的进程；
* **ActivityRecord**：ActivityStack的管理对象，每个Activity在AMS对应一个ActivityRecord，来记录Activity的状态以及其他的管理信息。其实就是服务器端的Activity对象的映像；

**Question 1： 如何判断APP是否已经启动？** AMS会保存一个ProcessRecord信息，有两部分构成，“uid + process”，每个应用工程序都有自己的uid，而process就是AndroidManifest.xml中Application的process属性，默认为package名。每次在新建新进程前的时候会先判断这个 ProcessRecord 是否已存在，如果已经存在就不会新建进程了，这就属于应用内打开 Activity 的过程了。

**从Activity创建成功到显示画面**

onCreate()方法中先执行setContentView()方法将对应的xml文件传入，之后会去调用window.setContentView()，最终会在这里创建**Decorview**并填充标题栏、状态栏，然后获取**contentParent**，然后调用LayoutInflater.inflate解析xml文件获取根root（ViewRootImpl），通过root.addView()将contentParent添加到**ViewRootImpl**中去，至此onCreate()结束。

开始onResume()阶段，在开始会向H类发送一个消息，然后在ActivityThread中获取之前创建的Decorview并调用windowManager.add()，最后在windowManager中将窗口和窗口的参数传到root.setView()，然后ViewRoot通过Binder调用WMS，使WMS所在的SS进程接收到按键事件时，可以回调到该root，同时ViewRoot会向自己的handler发送一条消息，然后进行处理（performTraversals），之后开始绘制过程（在Surface的canvas上绘制）。

先利用MeasureSpec完成onmeasure()，然后在onlayout()中确定各元素的坐标，ondraw()负责将view画到canvas上，再通过Surface进行跨进程最终调用Native层的**SGL**、openGI，最后再去调用硬件CPU进行渲染操作，最终界面显示在你眼前

**解释**

* **DecorView**：界面的根View，PhoneWindow的内部类
* **contentParent**：所有View的根View，在DecorView里面
* **ViewRootImpl**：ViewRoot是GUI管理系统与GUI呈现系统之间的桥梁，WindowManager通过ViewRootImpl与DecorView起联系。并且，View的绘制流程都是由ViewRootImpl发起的
* **SGL**：底层的2D图形渲染引擎

**参考文章：**

* http://gityuan.com/android/
* https://www.jianshu.com/p/af8e1ebdf902
* https://blog.csdn.net/lb84\_007/article/details/47774973
* https://blog.csdn.net/freekiteyu/article/details/79318031
* https://blog.csdn.net/qq\_39037047/article/details/88066589#
* https://baijiahao.baidu.com/s?id=1617072163535121555&wfr=spider&for=pc