## 他山之石

**Yuanhuihui-update？？**

<https://github.com/yuanhuihui?tab=repositories>

<https://blog.csdn.net/Luoshengyang>

<https://www.zhihu.com/people/gityuan/posts>

<https://toutiao.io/subjects/73076>

<http://www.androiddevtools.cn/>

<https://blog.csdn.net/gityuan>

1. 系统目录结构是啥

恢复出厂究竟干了啥

rwxrwxrwx root root 2017-09-15 10:10 bugreports -> /data/data/com.android.shell/files/bugreports

alreadyDexOpted.add(frameworkDir.getPath() + "/framework-res.apk");总是失败的原因

adb install执行原理

权限方式来控制第三方安装

最好是在拷贝文件之前就处理下

机子都能root了，其实就没有必要处理版本问题了

Root和非root区别是什么？是否存在root也无法修改的build.prop

书籍: 2017年06月深入理解Android内核设计思想 第2版（上下册）

2017年04月 深入浅出Android源代码：基于Android 6.0和实际开发案例剖析

~~2015年11月 Android系统优化从入门到精通~~

2016年10月 深入解析Android虚拟机

2015年07月深入理解Android系统

2015年06月 构建嵌入式Android系统

深入理解android 卷1 2 3

[RK3399][Android7.1] 调试笔记: <http://blog.csdn.net/kris_fei/article/category/7318364>

Android 安全框架 -- 总概http://blog.csdn.net/blue\_rush/article/details/55045546

<https://security.tencent.com/index.php/blog/msg/38>

not accessible for the namespace "classloader-namespace"

https://www.jianshu.com/p/a4af2bdcc3c0

<https://www.jianshu.com/p/4be3d1dafbec>

crash info--------------------------------

java.lang.UnsatisfiedLinkError: dlopen failed: library "/system/lib64/libAppUpgradeLink.so" needed or dlopened by "/system/lib64/libnativeloader.so" is not accessible for the namespace "classloader-namespace"

at java.lang.Runtime.loadLibrary0(Runtime.java:989)

at java.lang.System.loadLibrary(System.java:1562)

at com.dpad.core.data.DpadLinkUtil.<clinit>(DpadLinkUtil.java:6)

Android 安全设计

[如何自学Android？](https://zhuanlan.zhihu.com/p/20708611)

自动化埋点，xml方式或者activity,application 配置就好了吧

<https://blog.csdn.net/laizixingxingdewo/article/details/79276633>

AspectJ

<https://blog.csdn.net/cmder1000/article/details/70193988>

<https://blog.csdn.net/hackeey/article/details/78016979>

https://segmentfault.com/a/1190000015025152

系统裁剪

Libcore

Libnativehelper

Developers

Toolchain

Tools

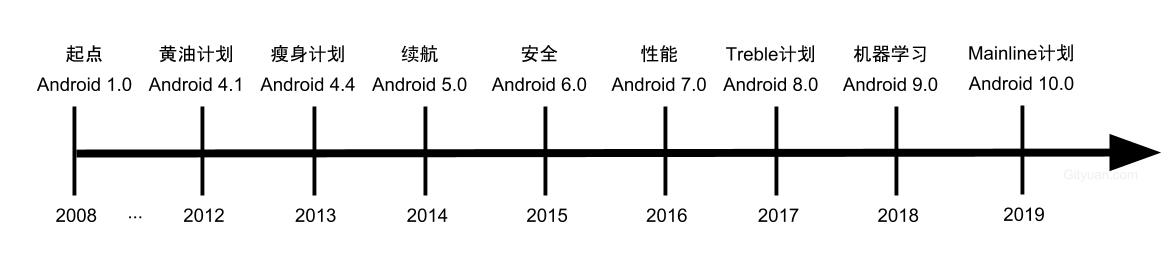
Development

Bionic

# Android技术架构演进与未

众所周知，Android是谷歌开发的一款基于Linux的开源操作系统，每年迭代一次大版本升级。 小米、华为、OPPO、VIVO、三星等各大厂商对Android原生系统进行二次开发衍生出具有各家特色的系统（比如MIUI），为手机、电视、平板电脑、手表等数十亿设备提供平台支持，使得Android作为全球最受欢迎的移动操作系统。Android诞生至今已有10余年，这一路走来Android遇到哪些问题？大版本升级朝着什么方向演进？Android的未来如何？

## 系统演进



系统演进趋势：每个Android大版本的更新迭代前行，历经10余年，在用户体验、流畅性、续航、安全、隐私、机器学习等方面都取得较大的改进。图中是每个大版本中最具代表性的特征标记在图中，并不代表着该版本全部特征，同样专项计划也不是只在某一个版本执行，比如续航和性能优化，每一个版本都在持续改进中，Treble计划也一直在迭代至今。

* 从Android 1.0发展到Android 4.0，系统各项功能和特性迭代到一个较完善的阶段；
* Android 4.1系统，Google开展了黄油计划（Project Butter），为了让Android系统摆脱UI交互上的严重滞后感，希望能像“黄油”一样顺滑。 核心原理是系统框架中的渲染和动画统一采用垂直同步技术(VSYNC)，以及三重缓冲技术(Triple Buffer)，让滑动、翻页等操作更加一致与顺滑。
* Android 4.4系统，Google开展了瘦身计划（Project Svelte），力求降低安卓系统的内存使用，解决低端机型升级难的问题，让Android 4.4可正常运行在所有Android手机，从而减少安卓系统继续碎片化。UI设计上，支持新的“沉浸式模式”，用户界面由过去的黑色与蓝色为主的色调转向带有透明度的浅色系，视觉语言变得更加明亮与现代化。
* Android 5.0系统，Google开展了伏特计划（Project Volta），力求提升续航能力，这方面Google落后于业界厂商，厂商直面用户对续航尤为迫切，往往系统资源管控更为严格。另外，系统采用全新的ART，抛弃Dalvik虚拟机，大幅提升运行效率。UI设计上，使用全新的扁平化Material Design设计风格，更加清新与质感的设计，统一Android设备的外观和使用体验。
* Android 6.0系统，Google引入新的运行时权限，让用户能够更好地了解和控制权限；引入了Doze模式，进一步提升电池续航能力。UI设计上，新增夜间模式，大幅改进通知栏，让通知更简洁。
* Android 7.0系统，引入新的JIT编译器，对AOT编译器的补充，可节省存储空间和加快更新速度；进一步优化Doze唤醒机制；UI设计上，支持分屏功能；
* Android 8.0系统，Google开展了计划（Project Treble），重新架构Android，将安卓系统框架与Vendor层解耦，力求彻底解决安卓碎片化这一老大难的问题，这是安卓系统架构最大的变化。系统层面加强对后台服务、广播、位置的管控限制。UI设计上，改进通知栏，智能文本选择和自动填充功能。
* Android 9.0系统，引入神经网络API，采用机器学习的思路来预测用户使用习惯来做省电优化，继续强化Treble计划；文件系统(sdcardf/F2FS)持续提升；私有API的限制进一步规范化Android生态，强化隐私和安全，硬件安全性模块以及统一生物识别身份验证界面。 UI设计上，新的手势导航，加强支持刘海屏，UI搜索界面使用到机器学习，AI正在逐步强化Android系统。多相机？？？无线方面，支持多摄像头和wifi室内导航！！！
* Android 10.0系统，Google开展了主线计划（Project Mainline），相关模块（Modules）不允许厂商直接修改，只能由Google应用商店来更新升级，强化用户隐私、系统安全与兼容性。支持脸部生物识别。
* 系统不断演进，但整体架构基本没有改变，如下图所示。



### 发展历程

先来看看Android系统的发展过程，从2008年发布Android 1.0系统，直到2019年即将发布Android 10.0系统，下面列举些重要的时间节点。

* 2003年10月，Andy Rubin团队创办Android公司；
* 2005年8月，谷歌收购Android公司，Andy Rubin担任谷歌工程部副总裁继续负责Android项目；
* 2008年9月，谷歌正式发布Android 1.0系统；
* 2011年1月，Android系统设备的用户总数达到了1.35亿，成为智能手机领域占有量第一的系统；
* 2011年8月，Android手机占据全球智能机市场48%份额，并在亚太地区市场占据统治地位，终结了Symbian系统的霸主地位，跃居全球第一；
* 2012年1月，谷歌Android Market已有10万开发者，推出超过40万应用；
* 2013年11月，Android 4.4正式发布，系统更智能、UI更现代；
* 2013年到2018年，这个阶段安卓进入飞速发展期，被升级的有摄像头、内存、机身、芯片等，原来的3.5寸小屏已退出历史舞台，全面屏、刘海屏、水滴屏已成为当下主流屏幕方案。

## 应用演进

Android系统离不开各App来提供丰富的功能，下面再来简单说一说应用的一些技术演进。

### 移动端跨平台技术

从最开始以Cordova为基础(依赖于WebView)的Hybrid混合开发技术，到React Native的桥接（将JS转为Native）的技术，再到最新的Flutter技术。Flutter是Google发布的全新的移动跨平台UI框架，渲染引擎依靠跨平台的Skia图形库来实现，依赖系统的只有图形绘制相关的接口，可以在最大程度上保证不同平台、不同设备的体验一致性，逻辑处理使用Dart语言，执行效率比JavaScript高。

另外，Google内部正在开发的另一个操作系统Fuchsia的UI layer采用的是Flutter，也就是说Flutter天然可以支持Android、IOS以及未来的Fuchsia。在大前端方向，对于跨平台开发中一直在不断迭代中寻找更好、更优的解决方案，目前来看Flutter还是更有优势。

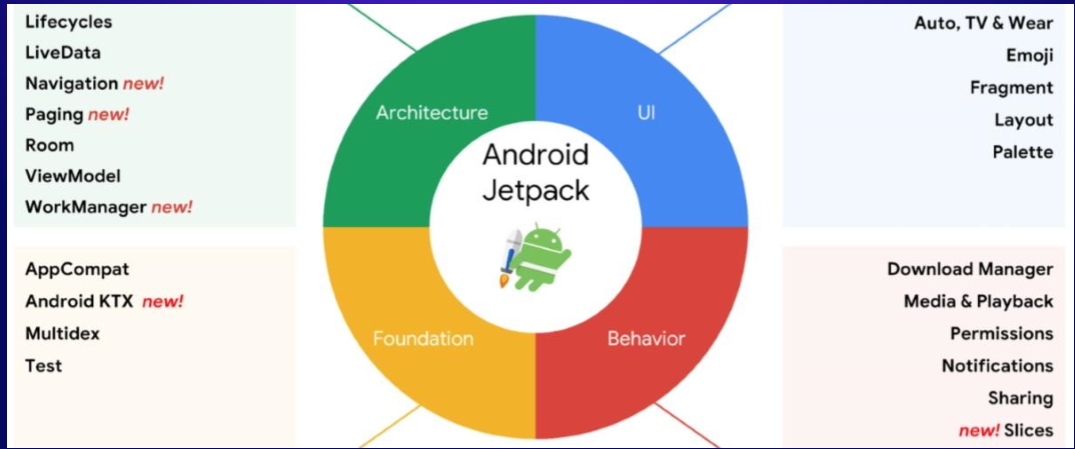
### 热修复与插件化技术

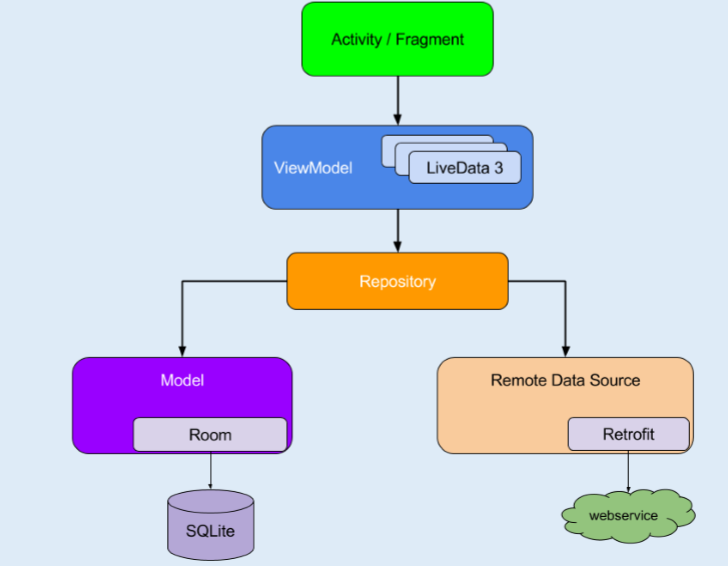
热修复的主要应用场景是为了让用户无感得修复线上缺陷，比如Tinker，Andfix，Sophix等。插件化是为了减少模块耦合，可减少主程序的规模，可按需加载，比如DroidPlugin，OpenAtlas等。关于各个热修复与插件化的细节不再展开，这里就说一点，Android 7.0对Native的NDK的调用限制是手铐，尤其是Android 9.0对Java层SDK的调用限制就是脚铐，那么对于Android应用想再搞插件化之类的黑科技便是带着脚手铐跳舞，能跳但舞姿可能不太美观。没啥前途

### 应用架构

MVC模式（Model–view–controller）但Activity类过于臃肿，为解决这个问题，有了MVP（Model–view–presenter），presenter不仅要操作数据，而且要更新view；再到MVVM（Model-View-ViewModel）解决了MVP大量的手动View和Model同步的问题，提供双向绑定机制

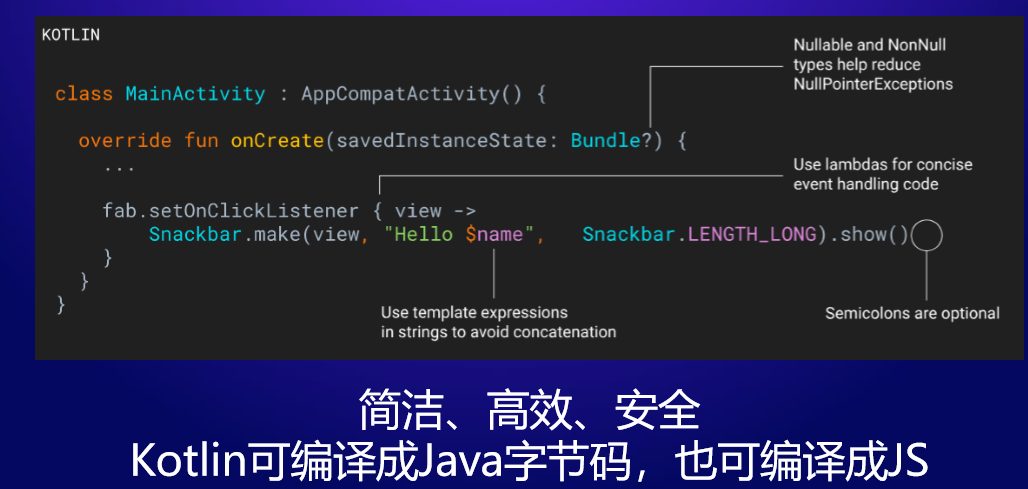
#### Jetpack





### Kotlin

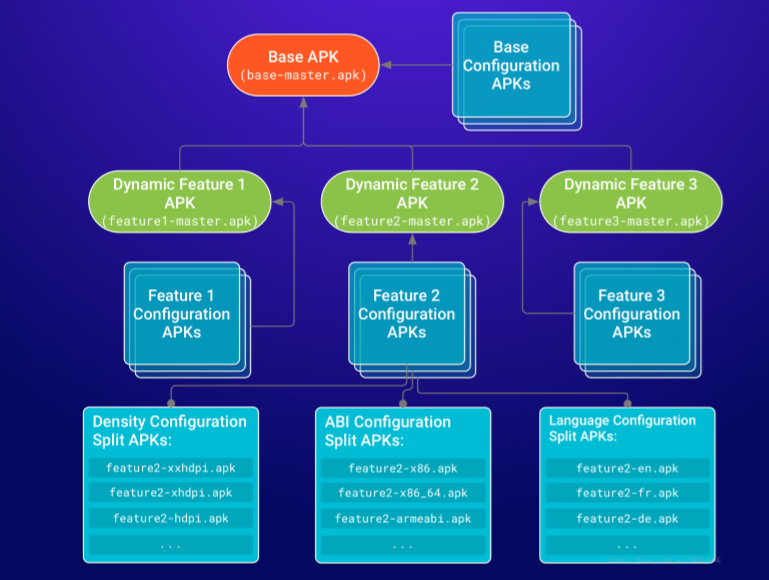
是Google推荐的官方静态编程语言，与Java互通，可相互转换。Kotlin编译成Java字节码，也可以编译成JavaScript，运行在没有JVM的设备上，简洁安全。使用Kotlin更快速地编写Android应用，可以提高开发者的工作效率，少编写样板代码，**被称之为 Android 世界的Swift**。谷歌开发者社区做过一个问卷调查，大概有40%的Android开发者已使用过Kotlin。这里并非鼓励大家一定都要使用Kotlin，学习新语言就像一次投资，要权衡团队成本与收益之间的利弊。效率提升10%而已



### App Bundle

随着应用不断演讲，功能越来越复杂，且应用针对不同屏幕设备、不同国家语言资源都打包在同一个App，导致应用包不断增大，据统计自2012年以来应用包大小增长5倍。虽然现在手机的存储空间越来越大，但用户照片、视频等媒体文件品质在逐渐提升，导致设备可用空间逐渐紧缩。为此Google在去年Google I/O大会讲述Android引入新的App动态化框架（即Android App Bundle，缩写为AAB）。利用Split Apk完成动态加载，使用AAB动态下发方式，可显著缩小应用体积，减少对存储空间的占用。

国内用的很少，应该用起来



#### 好处

#### Size更小

* Google has claimed that it can reduce app sizes up to 50 percent
* Its own apps like YouTube and other apps like LinkedIn which saw a 23% file reduction
* 安装更快
* 支持动态发布

#### 限制

* 仅限于通过 Google Play 发布的应用，（Google进一步巩固自身生态）
* 需要加入到 Google 的 beta program enroll your app in app signing by Google Play in the Play Console
* 最低支持版本Android 5.0 (API level 21)
* 低于Android 5.0 (API level 21) 的版本GooglePlay会优化Size，但不支持动态交付。

#### 成本

* 需要升级到Android Studio 3.2修改工程以便支持App Bundle格式
* 集成Play Core Library

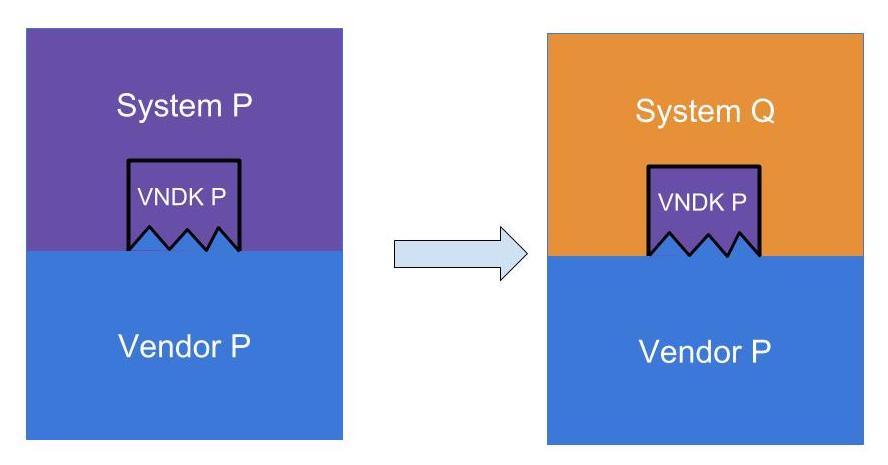
## 未来可能

可能的未来

改善碎片化问题（Treble） • 改进隐私安全与兼容性（Mainline） • 推广图形接口Vulkan • 移动端落地人工智能 • 提升虚拟机 • 统一安卓与物联网的互联互通

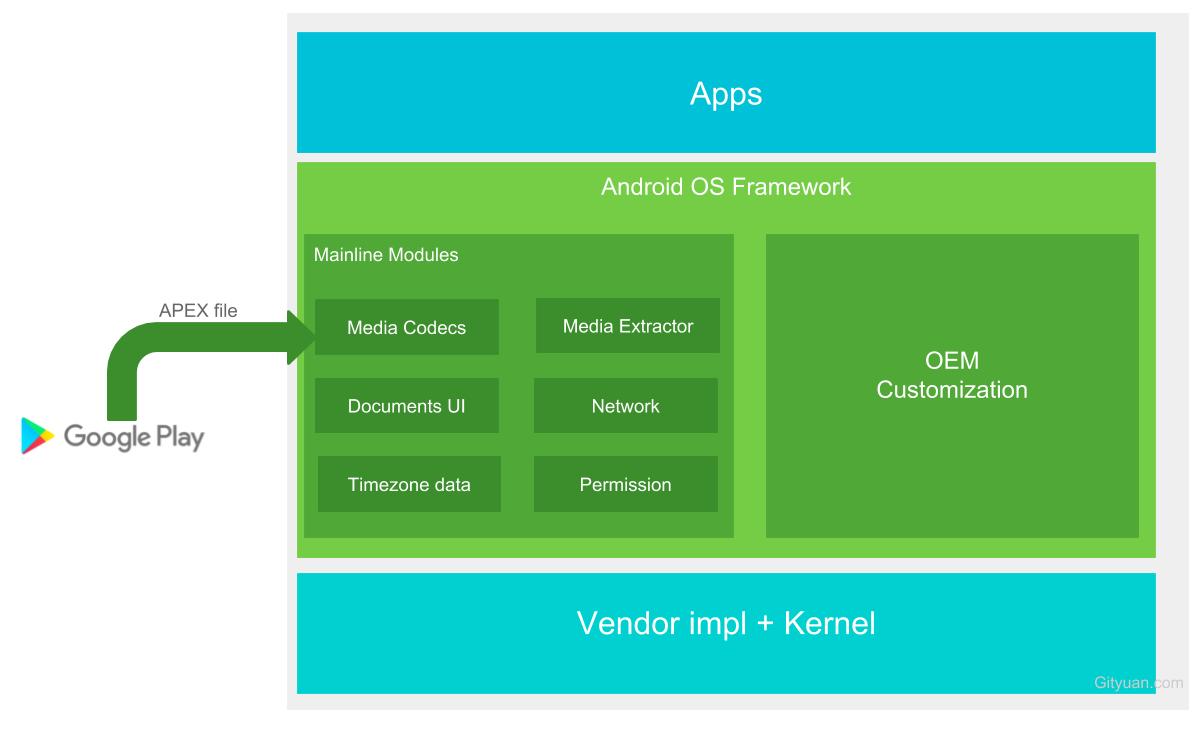
### Treble Project

碎片化

Android历经10余年的迭代，在流畅性、内存、续航、安全、隐私等方面都取得很大的进步，但Android系统的碎片化一直是痛点问题，带来不一致的用户体验。Android的开放性，是其长久发展的主要原因，让大多数的厂商都选择Android系统，但开放性的背后是碎片化，从Android诞生至今问题就一直存在，Google一直在努力从技术角度来解决碎片化问题。从Android 8.0提出Treble项目，重新架构系统将system与vendor解耦合，用于加快Android新版本的适配，效果并不明显，Google继续在后续的Android P以及Android Q一直在不遗余力地持续完善Treble项目，力争加快系统升级速度。如下图，目标是希望在保持Vendor不变的情况下，可以独立升级System模块。  


### Project Mainline

Android系统碎片化，让安全、隐私问题存在风险，且存在体验不一致性问题，但老版本手机的OTA维护升级对厂商来说成本是昂贵的，Ｇoogle感觉到对Android系统掌控力度不足，要想彻底改变，除非不让各大厂商定制化，这势必导致Android手机完全同质化，手机厂商就没法玩了，等于自掘坟墓，Google肯定不会这么干。于是，Google在Android 10.0提出了”Project Mainline“，将对隐私、安全、兼容性造成重大影响的少数模块独立成module，每个module打包成APEX格式（一种类似于APK的新格式），由Google通过应用商店定期来升级，从而保证低版本的手机不会因为碎片化而得不到隐私、安全与兼容性的更新。这些module是由Google维护的主线，各大厂商只能跟Google沟通并将代码upstream到AOSP主线。Google花费了大量的人力在努力完善并推行Mainline，Google希望统一管控的机制，厂商希望最大的自由定制空间，这是一场有趣的角逐，笔者跟团队一起跟Google协商落地module的落地计划，最终将某些module影响较大模块争取Android 11再上线，Mainline更新机制如下图所示。



### Google布局

2014年，发布Android Wear、Android TV 及Android Auto

• 2016年，发布Android Things智能设备系统 •

2016年，开始秘密研发Fuchsia系统

5G + AI的赋能之下，智能汽车、智能家居、 IOT都将会有广阔的市场前景。

#### Fuchsia

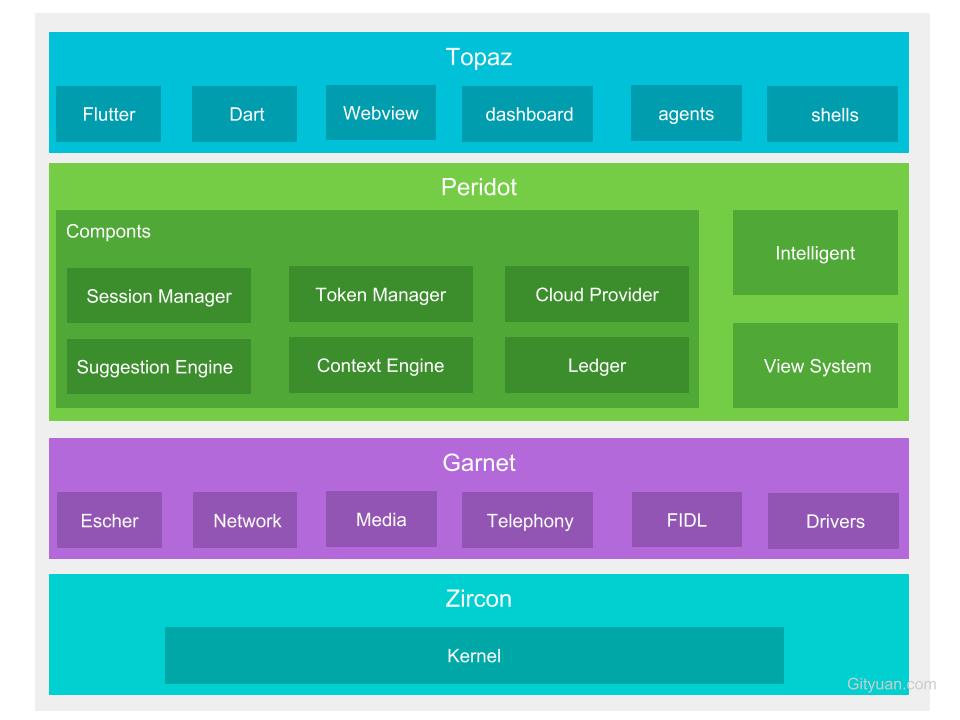
2016年开始，Google有一群超过百人的工程师团队秘密研发一款名为Fuchsia的新系统，该团队很豪华，有来自Android、iPhone、WebOS、Chrome、Flutter等核心工程师，这么多优秀的人在一起研发这个项目，的确值得期待。Fuchsia的内核采用Zircon，UI层采用Flutter框架，底层渲染Escher，支持Vulkan作为底层Graphics API。

提到Vulkan，顺便说一下，去年笔者在美国跟Android团队讨论到并问及Vulkan未来的规划，Google表示未来几年会大力推广Vulkan技术，Vulkan是一种跨平台的高性能低开销的图形接口，在移动设备上比OpenGL ES有着更出色的表现。Vulkan将会是未来Android平台的一个发展方向，尤其是游戏领域，比如王者荣耀Vulkan版本。

在2017年5月，Google的Android工程副总裁Dave Burke称Fuchsia是早期实验项目，而在谷歌内部有很多这类实验项目，存在很大未知变数。2017年11月，Google研发人员表示Fuchsia支持Swift，足见打造统一操作系统的野心。在过去的Google I/O大会只字未提及Fuchsia，Google官方回应不清楚Fuchsia会在出现在什么设备。

Fuchsia会是Android的终结者吗? 笔者认为至少未来五年内不太可能取代Android。当年为了和苹果iOS抗衡，Android系统研发作为Google重中之重，在这种情况下，Android诞生依然花费了Google 3年时间。而Fuchsia只是公司目前的实验项目，且Fuchsia并非基于业界成熟Linux内核，而是采用全新Zircon内核，项目工程路还很远。

笔者大致研究了一下Fuchsia系统源码，总结了一下Fuchsia的整个技术架构图如下。



从Fuchsia技术架构来看，内核层zircon的基础LK是专为嵌入式应用中小型系统设计的内核，代码简洁，适合嵌入式设备和高性能设备，比如IOT、移动可穿戴设备等，目前这些领域还没有标准化级别的垄断者。以及在框架层中有着语音交互、云端以及智能化等模块，由此笔者揣测未来Fuchsia率先应用在音控等智能设备。

Fuchsia基于功能的模块化操作系统，应该会使各组件模块能独立升级更新能力，保证体验一致性。Fuchsia在IOT领域占据一定份额后，加之其良好的跨平台，可以再逐步渗透到移动手机、笔记本电脑等设备，进而三位一体，打造手机、电脑与IOT完美的互联互通的统一平台体验，让多端设备都离不开Fuchsia。在2018年10月，在“蓝牙特别兴趣小组（Bluetooth SIG）”举办的UnPlugFest（UPF）测试大会上，Google再展示了Fuchsia与Android设备的互联性，可以窥见一斑。

Fuchsia的定位更是物联网，再是一统江湖，但路途漫长，至少要５年甚至更远。未来不可知，当然说不定Fuchsia作为实验项目，一直待在实验室，不过这种可能性比较小，做不到一统江湖，在IOT领域发光发亮还是大有可为的。

## 展望未来

追求极致，超预期：打一深进，工具化高效化

勤于思考和总结，提升内功修为：高通，binder问题，不要盲目去解决问题。100个学习，把事情做好

终生学习：大龄程序员！！

#### 操作系统

移动操作系统的演变过程，从按键交互的塞班功能机到触摸屏交互的Android/IOS智能机，从小屏幕手机到全面屏、刘海屏、水滴屏。任何系统无非干两件事：输入和输出，接收到外部输入信号后经过操作系统处理后输出信息。

* 从按键式交互到触屏式交互，伴随着塞班系统到Android系统的转变。未来的交互方式一定会更加生物智能化，当下的触屏交互可以理解成人类的触觉输入方式，未来将朝着人们更常见的听觉输入（语音）和视觉输入（身体姿势、表情等），甚至嗅觉输入（气味变化），也会伴随着新的操作系统的诞生。需要更加无缝地切入生活，而不是“安静，你吵到我的TNT”方式。
* 屏幕从小尺寸到大尺寸，并没有引发操作系统变革，因为技术创新是非连续性，非连续性才会引发第二曲线，诞生新技术。从1960年大型机，到1990年个人笔记本，再到现在的智能手机，设备本身越来越小。未来的设备如果发展非连续变革，可能不再需要实体硬件，随处可输出，一张白纸、一面墙，到那时操作系统的UI架构必然全新的变化。

前面提到Fuchsia系统，笔者认为至少未来五年内不太可能取代Android，但未来可期。新操作系统的崛起源于降维打击，直线超车很难，需要有非连续变革，如果只是某种程度上的改进，很难突破用户习惯、厂商以及生态圈的阻碍。Fuchsia需要降维打击，比如Fuchsia在IOT领域以及新的交互方式都很出色，加上万物无缝式的互联互通的平台，拥有跨平台型特性的Fuchsia有机会成为超级平台。

Android发展至今，已成为全球用户量最广泛的移动操作系统，手机行业竞争异常激烈，经过几番洗牌，国内手机厂商主要是华米OV四大公司，笔者预测在未来五年内国内手机厂商可能只有TOP3，那么Android的未来在哪里呢？

Google在2014年发布Android Wear智能手表系统、Android TV系统以及Android Auto汽车系统，在2016年发布的Android Things智能设备，这些全方位构建安卓的生态圈。在未来在人工智能和5G的赋能下，智能汽车、智能家居、IOT都将会有广阔的市场前景。但就目前人工智能的奇点还没到来，技术还处于前期阶段，一旦奇点来临将会爆炸式发展，或将重新定义生活方式。

汽车的智能化和互联网化是未来一大趋势，Google这两年确实在汽车领域发力，Android Auto在过去一年的用户增长250％。天生的移动特性加上越来越多的互联网服务需求，汽车需要一个具备多种感知能力的系统，或将成为是继手机、电视后Android的下一重点开拓领域。受到驾驶安全的限制，车载场景正好需要将以往的触屏按钮的交互方式，转向语音交互和生物感知，车舱内是天然的语音交互场景，而不再是“安静，你吵到我的TNT”，语音和图像识别、人工智能等技术或许会在车载领域得到更大的发展。

#### 职业发展

随着Android的发展，有些人对Android未来感到茫然，经常收到读者私信问，“前辈，从事Android是不是没有前途，找工作困难，希望能给点建议？”。早在2010年市场上有大量Android招聘，基本懂一点Java基础的就可以搞Android，当时是移动互联网创业的高峰期，号称只要做个App就可以创业。“风来了猪都能找到工作，风停了最先摔死的都是猪”，如果你觉得找工作难，那一定是你在混日子，Android中高级以上的人才一直都非常稀缺。

只要在Android领域深耕，做到极致，努力成为这个方向的专家，有精力再提升工程架构思维，软件工程思想都是相通的，境界会得到提升，即便再学习新东西也会非常快。只要一个领域做到极致，即便Android被淘汰了，换新领域面试官依然会相信你也能做到极致。千万不能用一年的工作能力混十年工作经历，否则你的市场价格连一年都不如，成为工作困难户。

#### Android

Android系统迭代更新10余年，在用户体验、性能、功耗、安全、隐私等方面都取得很大的进步，后续版本会持续在内存、文件系统、虚拟机、图形图像等方向优化。随着Android系统功能越来越多，系统架构中有些模块未来可能会被重构，某些服务大锁制约性能，比如Android 8.0优化过binder大锁让性能显著提升。关于图形方面，Vulkan将会是未来Android平台的一个发展方向，尤其是游戏领域。

人工智能在Android系统上目前效果不太显著，Google未来应该还会持续投入，比如在AI预测用户行为加上相应后台管控策略用于提升手机续航。碎片化仍是当下最主要的问题，碎片化也导致用户隐私、安全和体验一致性方面得不到保障，Google专门成立团队致力于Android Mainline，从Android Q开始规划Mainline，未来的版本都将逐步迭代更新。最后说一点，App兼容性问题比较严重，据Google实验统计Android Q系统在全球Top 1000应用的兼容性不达标率4.3%，而中国Top 1000应用的兼容性不达标率17.6%，可见国内Android生态圈更为严峻得多，Google对隐藏API的限制就是一步长远之棋，短期内导致应用不兼容加剧，长期来看生态圈会逐步健康，最理想的情况就是Android系统大版本升级而App兼容性问题不再有，后续Google应和厂商会加强跟主流应用协作规范应用，搭建良好健康的Android生态圈。

以上是笔者对Android系统及对未来的一些拙见，欢迎业界同仁一起探讨。

## 引用

## 小结

支持多摄像头

WIFI室内导航

插件化和热修复没有意义

大前端，目前来看Flutter还是更有优势

# Android架构

## Android架构

Google提供的4层架构图，是非常经典，但只是如垒砖般的方式，简单地分层，而不足表达Android整个系统的启动过程，环环相扣的连接关系，本文更多的是以进程的视角，以分层的架构来诠释Android系统的全貌。



**图解：** Android系统启动过程由上图从下往上的一个过程：Loader -> Kernel -> Native -> Framework -> App，接来下简要说说每个过程：

### 概述

Android系统非常庞大，底层是采用Linux作为基底，上层采用包含虚拟机的Java层以及Native层，通过系统调用(Syscall)连通系统的内核空间与用户空间。用户空间主要采用C++和Java代码，通过JNI技术打通用户空间的Java层和Native层(C++/C)，从而融为一体

下图是Google提供的一张经典的4层架构图，从下往上，依次分为Linux内核，系统库和Android Runtime，应用框架层，应用程序层这4层架构，每一层都包含大量的子模块或子系统。



这只是如垒砖般地分层，并没有表达Android整个系统的内部架构、运行机理，以及各个模块之间是如何衔接与配合工作的。**为了更深入地掌握Android整个架构思想以及各个模块在Android系统所处的地位与价值，计划以Android系统启动过程为主线，以进程的视角来诠释Android M系统全貌**，全方位的深度剖析各个模块功能，争取各个击破。这样才能犹如庖丁解牛，解决、分析问题则能游刃有余。

### 进程视角

深红色：代表0号进程，是在进入刚进入启动时创建的，内核启动完成后便退出；

浅红色：init/kthreadd/Zygote，这3个进程分别会创建大量的内核守护进程、用户空间守护进程以及应用进程，地位主要创建了大量子进程(注意，此处说的不是子线程)；

深紫色：system server/ media server/ servicemanager，这3个进程并不是用于创建子进程，而是对于整个Android架构，有着非常重要的意义；

深蓝色：内核守护进程、用户空间守护进程以及应用进程，这些都是由“深红色”fork生成的；

浅蓝色：各种系统服务、驱动等相关信息。

### 分层视角

开机过程是从图中最下方Loader开始，经过 -> Kernel -> Native -> Framework，一路直至最上层的App层启动。下面来进一步说明

#### Loader

Boot ROM: 当按下电源开机键，引导芯片代码从预设定处(固化在ROM)开始执行，加载引导程序到RAM；

Boot Loader：是启动Android OS之前的引导程序，主要是检查RAM，初始化硬件参数等功能；

#### Kernel

Kernel层是指Android内核层，到这里才刚刚开始进入Android系统。

启动Kernel的swapper进程(pid=0)：该进程又称为idle进程, 系统初始化过程Kernel由无到有开创的第一个进程, 用于初始化进程管理、内存管理，加载Display,Camera Driver，Binder Driver等相关工作；

启动init进程(1号进程)，是Linux系统的用户空间进程，也就是Native层的进程的鼻祖；

启动kthreadd进程（pid=2）：是Linux系统的内核进程，会创建内核工作线程kworkder，软中断线程ksoftirqd，thermal等内核守护进程。kthreadd进程是所有内核进程的鼻祖。

#### Native

这里的Native层主要包括init孵化来的用户空间的守护进程、HAL层以及开机动画等。启动init进程(pid=1),是Linux系统的用户进程，init进程是所有用户进程的鼻祖。

* init进程会孵化出ueventd、logd、healthd、installd、adbd、lmkd等用户守护进程；
* init进程还启动servicemanager(binder服务管家)、bootanim(开机动画)等重要服务
* init进程孵化出Zygote进程，Zygote进程是Android系统的第一个Java进程(即虚拟机进程)，Zygote是所有Java进程的父进程，Zygote进程本身是由init进程孵化而来的

#### Framework

* Zygote进程，是由init进程通过解析init.rc文件后fork生成的，Zygote进程主要包含：
  + 加载ZygoteInit类，注册Zygote Socket服务端套接字；
  + 加载虚拟机；
  + preloadClasses；
  + preloadResouces。
* System Server进程，是由Zygote进程fork而来，System Server是Zygote孵化的第一个进程，System Server负责启动和管理整个Java framework，包含ActivityManager，PowerManager等服务。
* Media Server进程，是由init进程fork而来，负责启动和管理整个C++ framework，包含AudioFlinger，Camera Service等服务

#### App

此后所有的app启动由System Server调用，然后让Zygote fork，Zygote进程孵化出Home进程，这便是用户看到的桌面App；Zygote进程Browser，Phone等App进程；每个App至少运行在一个进程上；

## 通信基石

无论是Android系统，还是各种Linux衍生系统，各个组件、模块往往运行在各种不同的进程和线程内，这里就必然涉及进程/线程之间的通信。

对于IPC(Inter-Process Communication, 进程间通信)，Linux现有管道、消息队列、共享内存、套接字、信号量、信号这些IPC机制，Android额外还有Binder IPC机制，Android OS中的Zygote进程的IPC采用的是Socket机制，在上层system server、media server以及上层App之间更多的是采用Binder IPC方式来完成跨进程间的通信。

对于Android上层架构中，很多时候是在同一个进程的线程之间需要相互通信，例如同一个进程的主线程与工作线程之间的通信，往往采用的Handler消息机制。

想深入理解Android内核层架构，必须先深入理解Linux现有的IPC机制；对于Android上层架构，则最常用的通信方式是Binder、Socket、Handler，当然也有少量其他的IPC方式，比如杀进程Process.killProcess()采用的是signal方式。下面说说Binder、Socket、Handler：

另外，不同层的通信，api设计也纳入了这类，主要有SysCall，JNI。

### Binder

Binder作为Android系统提供的一种IPC机制，无论从系统开发还是应用开发，都是Android系统中最重要的组成，也是最难理解的一块知识点，想了解[为什么Android要采用Binder作为IPC机制？](https://www.zhihu.com/question/39440766/answer/89210950) 可查看我在知乎上的回答。深入了解Binder机制，最好的方法便是阅读源码，借用Linux鼻祖Linus Torvalds曾说过的一句话：Read The Fucking Source Code。下面简要说说Binder IPC原理。

**Binder IPC原理**

Binder通信采用c/s架构，从组件视角来说，包含Client、Server、ServiceManager以及binder驱动，其中ServiceManager用于管理系统中的各种服务。



* 想进一步了解Binder，可查看[Binder系列—开篇](http://gityuan.com/2015/10/31/binder-prepare/)，Binder系列花费了13篇文章的篇幅，从源码角度出发来，讲述Driver、Native、Framework、App四个层面的整个完整流程。根据有些读者反馈这个系列还是不好理解，这个binder涉及的层次跨度比较大,知识量比较广, 建议大家先知道binder是用于进程间通信,有个大致概念就可以.先去学习系统基本知识,等后面有一定功力再进一步深入研究Binder.

**原理篇**

| **序号** | **文章名** | **概述** |
| --- | --- | --- |
| 0 | [Binder系列—开篇](http://gityuan.com/2015/10/31/binder-prepare/) | Binder概述 |
| 1 | [Binder系列3—启动Service Manager](http://gityuan.com/2015/11/07/binder-start-sm/) | ServiceManager守护进程 注册和查询服务 |
| 2 | [Binder系列4—获取Service Manager](http://gityuan.com/2015/11/08/binder-get-sm/) | 获取代理对象BpServiceManager |
| 3 | [Binder系列5—注册服务(addService)](http://gityuan.com/2015/11/14/binder-add-service/) | 注册Media服务 |
| 4 | [Binder系列6—获取服务(getService)](http://gityuan.com/2015/11/15/binder-get-service/) | 获取Media代理，以及DeathRecipient |
| 5 | [Binder系列7—framework层分析](http://gityuan.com/2015/11/21/binder-framework/) | framework层服务注册和查询，Binder注册 |
| 6 | [理解Binder线程池的管理](http://gityuan.com/2016/10/29/binder-thread-pool/) | Binder的startThreadPool过程 |
| 7 | [彻底理解Android Binder通信架构](http://gityuan.com/2016/09/04/binder-start-service/) | startService为主线 |
| 8 | [Binder系列10—总结](http://gityuan.com/2015/11/28/binder-summary/) | Binder的简单总结 |
| 9 | [Binder IPC的权限控制](http://gityuan.com/2016/03/05/binder-clearCallingIdentity/) | clearCallingIdentity/restoreCallingIdentity |
| 10 | [Binder死亡通知机制之linkToDeath](http://gityuan.com/2016/10/03/binder_linktodeath/) | Binder死亡通知机制 |

**驱动篇:**

|  |  |  |
| --- | --- | --- |
| 1 | [Binder系列1—Binder Driver初探](http://gityuan.com/2015/11/01/binder-driver/) | 驱动open/mmap/ioctl，以及binder结构体 |
| 2 | [Binder系列2—Binder Driver再探](http://gityuan.com/2015/11/02/binder-driver-2/) | Binder通信协议，内存机制 |

**使用篇:**

|  |  |  |
| --- | --- | --- |
| 1 | [Binder系列8—如何使用Binder](http://gityuan.com/2015/11/22/binder-use/) | Native层、Framwrok层自定义Binder服务 |
| 2 | [Binder系列9—如何使用AIDL](http://gityuan.com/2015/11/23/binder-aidl/) | App层自定义Binder服务 |

### Socket

Socket通信方式也是C/S架构，比Binder简单很多。在Android系统中采用Socket通信方式的主要：

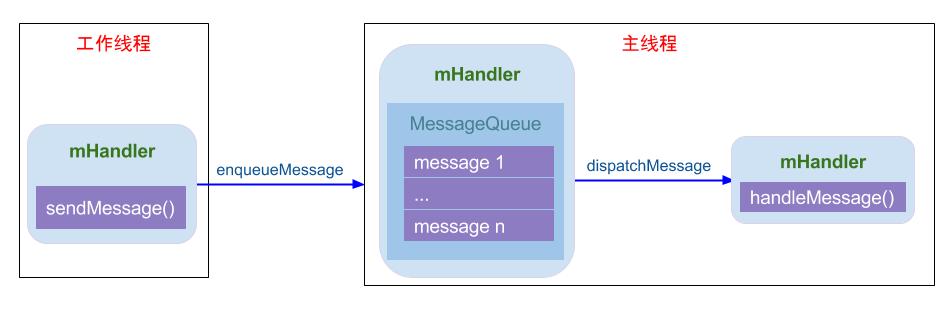
* zygote：用于孵化进程，系统进程system\_server孵化进程时便通过socket向zygote进程发起请求；
* installd：用于安装App的守护进程，上层PackageManagerService很多实现最终都是交给它来完成；
* lmkd：lowmemorykiller的守护进程，Java层的LowMemoryKiller最终都是由lmkd来完成；
* adbd：这个也不用说，用于服务adb；
* logcatd:这个不用说，用于服务logcat；
* vold：即volume Daemon，是存储类的守护进程，用于负责如USB、Sdcard等存储设备的事件处理。

等等还有很多，这里不一一列举，Socket方式更多的用于Android framework层与native层之间的通信。Socket通信方式相对于binder非常简单，所以一直没有写相关文章，为了成一个体系，下次再补上。

### Handler

**Binder/Socket用于进程间通信，而Handler消息机制用于同进程的线程间通信**，Handler消息机制是由一组MessageQueue、Message、Looper、Handler共同组成的，为了方便且称之为Handler消息机制。

有人可能会疑惑，为何Binder/Socket用于进程间通信，能否用于线程间通信呢？答案是肯定，对于两个具有独立地址空间的进程通信都可以，当然也能用于共享内存空间的两个线程间通信，这就好比杀鸡用牛刀。接着可能还有人会疑惑，那handler消息机制能否用于进程间通信？答案是不能，Handler只能用于共享内存地址空间的两个线程间通信，即同进程的两个线程间通信。很多时候，Handler是工作线程向UI主线程发送消息，即App应用中只有主线程能更新UI，其他工作线程往往是完成相应工作后，通过Handler告知主线程需要做出相应地UI更新操作，Handler分发相应的消息给UI主线程去完成，如下图：



由于工作线程与主线程共享地址空间，即Handler实例对象mHandler位于线程间共享的内存堆上，工作线程与主线程都能直接使用该对象，只需要注意多线程的同步问题。工作线程通过mHandler向其成员变量MessageQueue中添加新Message，主线程一直处于loop()方法内，当收到新的Message时按照一定规则分发给相应的handleMessage()方法来处理。所以说，而Handler消息机制用于同进程的线程间通信的核心是线程间共享内存空间，而不同进程拥有不同的地址空间，也就不能用handler来实现进程间通信。

上图只是Handler消息机制的一种处理流程，是不是只能工作线程向UI主线程发消息呢，其实不然，可以是UI线程向工作线程发送消息，也可以是多个工作线程之间通过handler发送消息。更多关于Handler消息机制文章：

* [Android消息机制-Handler(framework篇)](http://gityuan.com/2015/12/26/handler-message-framework/)
* [Android消息机制-Handler(native篇)](http://gityuan.com/2015/12/27/handler-message-native/)
* [Android消息机制3-Handler(实战)](http://gityuan.com/2016/01/01/handler-message-usage/)

要理解framework层源码，掌握这3种基本的进程/线程间通信方式是非常有必要，当然Linux还有不少其他的IPC机制，比如共享内存、信号、信号量，在源码中也有体现，如果想全面彻底地掌握Android系统，还是需要对每一种IPCd机制都有所了解

### API调用（ Syscall && JNI）

这是通信基石之基础

* Native与Kernel之间有一层系统调用(SysCall)层，见[Linux系统调用(Syscall)原理](http://gityuan.com/2016/05/21/syscall/);
* Java层与Native(C/C++)层之间的纽带JNI，见[Android JNI原理分析](http://gityuan.com/2016/05/28/android-jni/)

# 核心提纲

通过前面对系统启动的介绍，相信大家对Android已然“知全貌”，那么接下来需要“抓核心，理思路”。

## 系统启动系列

在整个开机流程中，有几个非常重要的进程，分别是init、Zygote、SystemServer进程。接下来，计划用三篇文章来分别阐述：



[Android系统启动-概述](http://gityuan.com/2016/02/01/android-booting/): Android系统中极其重要进程：init, zygote, system\_server, servicemanager 进程:

|  |  |  |
| --- | --- | --- |
| 序号 | 进程启动 | 概述 |
| 1 | [init进程](http://gityuan.com/2016/02/05/android-init/) | Linux系统中用户空间的第一个进程, Init.main |
| 2 | [zygote进程](http://gityuan.com/2016/02/13/android-zygote/) | 所有Ａpp进程的父进程, ZygoteInit.main |
| 3 | [system\_server进程(上篇)](http://gityuan.com/2016/02/14/android-system-server/) | 系统各大服务的载体, forkSystemServer过程 |
| 4 | [system\_server进程(下篇)](http://gityuan.com/2016/02/20/android-system-server-2/) | 系统各大服务的载体, SystemServer.main |
| 5 | [servicemanager进程](http://gityuan.com/2015/11/07/binder-start-sm/) | binder服务的大管家, 守护进程循环运行在binder\_loop |
| 6 | [app进程](http://gityuan.com/2016/03/26/app-process-create/) | 通过Process.start启动App进程, ActivityThread.main |

再来看看守护进程(进程名一般以d为后缀，比如logd), 先介绍以下部分,后缀再增加.

* [debuggerd](http://gityuan.com/2016/06/15/android-debuggerd/)
* [installd](http://gityuan.com/2016/11/13/android-installd)
* [lmkd](http://gityuan.com/2016/09/17/android-lowmemorykiller/)

## 4.2 系统稳定性系列

Android系稳定性主要是异常崩溃(crash)和执行超时(timeout), [Android系统稳定性简述](http://gityuan.com/2016/06/19/stability_summary/) :

|  |  |  |
| --- | --- | --- |
| 序号 | 文章名 | 概述 |
| 1 | [理解Android ANR的触发原理](http://gityuan.com/2016/07/02/android-anr/) | 触发ANR的场景以及机理 |
| 2 | [Input系统—ANR原理分析](http://gityuan.com/2017/01/01/input-anr/) | input触发ANR的原理 |
| 3 | [理解Android ANR的信息收集过程](http://gityuan.com/2016/12/02/app-not-response/) | AMS.appNotResponding过程分析,收集traces |
| 4 | [ART虚拟机之Trace原理](http://gityuan.com/2016/11/26/art-trace/) | kill -3 信息收集过程 |
| 5 | [Native进程之Trace原理](http://gityuan.com/2016/11/27/native-traces/) | debuggerd -b 信息收集过程 |
| 6 | [WatchDog工作原理](http://gityuan.com/2016/06/21/watchdog/) | WatchDog触发机制 |
| 7 | [理解Java Crash处理流程](http://gityuan.com/2016/06/24/app-crash/) | AMS.handleApplicationCrash过程分析 |
| 8 | [理解Native Crash处理流程](http://gityuan.com/2016/06/25/android-native-crash/) | debuggerd守护进程 |

## 4.3 Android进程系列

进程对于系统非常重要，系统运转，各种服务、组件的载体都依托于进程，对进程理解越深刻，越能掌握系统整体架构。那么先来看看进程相关：

|  |  |  |
| --- | --- | --- |
| 序号 | 文章名 | 概述 |
| 1 | [理解Android进程创建流程](http://gityuan.com/2016/03/26/app-process-create/) | Process.start过程分析 |
| 2 | [理解杀进程的实现原理](http://gityuan.com/2016/04/16/kill-signal/) | Process.killProcess过程分析 |
| 3 | [Android四大组件与进程启动的关系](http://gityuan.com/2016/10/09/app-process-create-2/) | AMS.startProcessLocked过程分析组件与进程 |
| 4 | [Android进程绝杀技–forceStop](http://gityuan.com/2016/10/22/force-stop/) | force-stop过程分析彻底移除组件与杀进程 |
| 5 | [理解Android线程创建流程](http://gityuan.com/2016/09/24/android-thread/) | 3种不同线程的创建过程 |
| 6 | [彻底理解Android Binder通信架构](http://gityuan.com/2016/09/04/binder-start-service/) | 以start-service为线,阐述进程间通信机理 |
| 7 | [理解Binder线程池的管理](http://gityuan.com/2016/10/29/binder-thread-pool/) | Zygote fork的进程都默认开启binder线程池 |
| 8 | [Android进程生命周期与ADJ](http://gityuan.com/2015/10/01/process-lifecycle/) | 进程adj, processState以及lmk |
| 9 | [Android LowMemoryKiller原理分析](http://gityuan.com/2016/09/17/android-lowmemorykiller/) | lmk原理分析 |
| 10 | [进程优先级](http://gityuan.com/2015/10/01/process-priority/) | 进程nice,thread priority以及scheduler |
| 11 | [Android进程调度之adj算法](http://gityuan.com/2016/08/07/android-adj/) | updateOomAdjLocked过程 |
| 12 | [Android进程整理](http://gityuan.com/2015/12/19/android-process-category/) | 整理系统的所有进程/线程 |

## 4.4 四大组件系列

对于App来说，Android应用的四大组件Activity，Service，Broadcast Receiver， Content Provider最为核心，接下分别展开介绍：

|  |  |  |
| --- | --- | --- |
| 序号 | 文章名 | 类别 |
| 1 | [startActivity启动过程分析](http://gityuan.com/2016/03/12/start-activity/) | Activity |
| 2 | [简述Activity生命周期](http://gityuan.com/2016/03/18/start-activity-cycle/) | Activity |
| 3 | [startService启动过程分析](http://gityuan.com/2016/03/06/start-service/) | Service |
| 4 | [bindService启动过程分析](http://gityuan.com/2016/05/01/bind-service/) | Service |
| 5 | [以Binder视角来看Service启动](http://gityuan.com/2016/09/04/binder-start-service/) | Service |
| 6 | [Android Broadcast广播机制分析](http://gityuan.com/2016/06/04/broadcast-receiver/) | Broadcast |
| 7 | [理解ContentProvider原理](http://gityuan.com/2016/07/30/content-provider/) | ContentProvider |
| 8 | [ContentProvider引用计数](http://gityuan.com/2016/05/03/content_provider_release/) | ContentProvider |
| 9 | [Activity与Service生命周期](http://gityuan.com/2015/05/31/android-lifecycle/) | Activity&&Service |
| 10 | [简述Activity与Window关系](http://gityuan.com/2017/04/16/activity-with-window/) | Activity&&Window |
| 11 | [四大组件之综述](http://gityuan.com/2017/05/19/ams-abstract/) | AMS |
| 12 | [四大组件之ServiceRecord](http://gityuan.com/2017/05/25/service_record/) | Service |
| 13 | [四大组件之BroadcastRecord](http://gityuan.com/2017/06/03/broadcast_record/) | Broadcast |
| 14 | [四大组件之ContentProviderRecord](http://gityuan.com/2017/06/04/content_provider_record/) | ContentProvider |
| 15 | [理解Android Context](http://gityuan.com/2017/04/09/android_context/) | Context |
| 16 | [理解Application创建过程](http://gityuan.com/2017/04/02/android-application/) | Application |
| 17 | [unbindService流程分析](http://gityuan.com/2016/05/02/unbind-service/) | Service |
| 18 | [四大组件之ActivityRecord](http://gityuan.com/2017/06/11/activity_record/) | Activity |
| 19 | [AMS总结(一)](http://gityuan.com/2017/06/25/ams_summary_1/) | AMS |

## 4.5 图形系统系列

图形也是整个系统非常复杂且重要的一个系列，涉及WindowManager,SurfaceFlinger.

|  |  |  |
| --- | --- | --- |
| 序号 | 文章名 | 类别 |
| 1 | [WindowManager启动篇](http://gityuan.com/2017/01/08/windowmanger/) | Window |
| 2 | [WMS之启动窗口篇](http://gityuan.com/2017/01/15/wms_starting_window/) | Window |
| 3 | [以Window视角来看startActivity](http://gityuan.com/2017/01/22/start-activity-wms/) | Window |
| 4 | [Android图形系统概述](http://gityuan.com/2017/02/05/graphic_arch/) | SurfaceFlinger |
| 5 | [SurfaceFlinger启动篇](http://gityuan.com/2017/02/11/surface_flinger/) | SurfaceFlinger |
| 6 | [SurfaceFlinger绘图篇](http://gityuan.com/2017/02/18/surface_flinger_2/) | SurfaceFlinger |
| 7 | [Choreographer原理](http://gityuan.com/2017/02/25/choreographer/) | Choreographer |

## 4.6 系统服务篇

再则就是在整个架构中有大量的服务，都是基于[Binder](http://gityuan.com/2015/10/31/binder-prepare/)来交互的，计划针对部分核心服务来重点分析：

系统服务的注册过程, 见[Android系统服务的注册方式](http://gityuan.com/2016/10/01/system_service_common/)

* AMS服务
  + [AMS启动过程（一）](http://gityuan.com/2016/02/21/activity-manager-service/)
  + 更多组件篇[见小节4.3]
* Input系统
  + [Input系统—启动篇](http://gityuan.com/2016/12/10/input-manager/)
  + [Input系统—InputReader线程](http://gityuan.com/2016/12/11/input-reader/)
  + [Input系统—InputDispatcher线程](http://gityuan.com/2016/12/17/input-dispatcher/)
  + [Input系统—UI线程](http://gityuan.com/2016/12/24/input-ui/)
  + [Input系统—进程交互](http://gityuan.com/2016/12/31/input-ipc/)
  + [Input系统—ANR原理分析](http://gityuan.com/2017/01/01/input-anr/)
* PKMS服务
  + [PackageManager启动篇](http://gityuan.com/2016/11/06/packagemanagerservice)
  + [Installd守护进程](http://gityuan.com/2016/11/13/android-installd)
* Alarm服务
  + [理解AlarmManager机制](http://gityuan.com/2017/03/12/alarm_manager_service/)
* JobScheduler服务
  + [理解JobScheduler机制](http://gityuan.com/2017/03/10/job_scheduler_service/)
* BatteryService
  + [Android耗电统计算法](http://gityuan.com/2016/01/10/power_rank/)
* PMS服务
* DropBox服务
  + [DropBoxManager启动篇](http://gityuan.com/2016/06/12/DropBoxManagerService/)
* UserManagerService
  + [多用户管理UserManager](http://gityuan.com/2016/11/20/user_manager/)
* 更多服务介绍, 敬请期待…

## 4.7 内存&&存储篇

* 内存篇
  + [Android LowMemoryKiller原理分析](http://gityuan.com/2016/09/17/android-lowmemorykiller/)
  + [Linux内存管理](http://gityuan.com/2015/10/30/kernel-memory/)
  + [Android内存分析命令](http://gityuan.com/2016/01/02/memory-analysis-command/)
* 存储篇
  + [Android存储系统之源码篇](http://gityuan.com/2016/07/17/android-io/)
  + [Android存储系统之架构篇](http://gityuan.com/2016/07/23/android-io-arch)
* Linux驱动篇
  + 敬请期待
* dalvik/art
  + [ART虚拟机之Trace原理](http://gityuan.com/2016/11/26/art-trace/)

## 4.8 工具篇

最后，说说Android相关的一些常用命令和工具以及调试手段.

|  |  |  |
| --- | --- | --- |
| 序号 | 文章名 | 类别 |
| 1 | [理解Android编译命令](http://gityuan.com/2016/03/19/android-build/) | build |
| 2 | [性能工具Systrace](http://gityuan.com/2016/01/17/systrace/) | systrace |
| 3 | [Android内存分析命令](http://gityuan.com/2016/01/02/memory-analysis-command/) | Memory |
| 4 | [ps进程命令](http://gityuan.com/2015/10/11/ps-command/) | Process |
| 5 | [Am命令用法](http://gityuan.com/2016/02/27/am-command/) | Am |
| 6 | [Pm命令用法](http://gityuan.com/2016/02/28/pm-command/) | Pm |
| 7 | [调试系列1：bugreport源码篇](http://gityuan.com/2016/06/10/bugreport/) | bugreport |
| 8 | [调试系列2：bugreport实战篇](http://gityuan.com/2016/06/11/bugreport-2/) | bugreport |
| 9 | [dumpsys命令用法](http://gityuan.com/2016/05/14/dumpsys-command/) | dumpsys |

（3）对于App来说，Android应用的四大组件Activity，Service，Broadcast Receiver， Content Provider最为核心，那么我们需要分别展开对其他的分解：

Android组件-Activity

Android组件-Service

Android组件-Broadcast Receiver

Android组件-Content Provider

（4）有了这些，中间还缺少关于虚拟机ART的介绍，会需要对ART分析，后续还需要开展对ART虚拟机的一系列文章。另外，从架构中还有很多一块没有提及，那便是Linux Kernel，这部分内容，计划从进程，内存，IO的视角展开分析。

Linux内核-进程篇

Linux内核-内存篇

Linux内核-IO篇

Linux内核-驱动篇

（5）最后，对整个架构回顾，从性能角度谈谈如何优化的问题，这是一个很大的话题，涉及面之广，会贯穿整个过程。

# 重要模块分布图

本人在尝试对安卓系统的部分代码进行编译时,经常发现需要查找某个so库的源码进行修改,但是苦于安卓so库的命名方式各异,且在源码中的位置各不相同,查找起来很费时间。于是把其中的一些so库所对应的源码路径记录了下来,在这里分享给大家。

libEGL.so------frameworks/native/opengl/libs/EGL/

libandroid.so——frameworks/base/native/android

libandroid\_runtime.so——frameworks/base/core/jni

libandroidfw.so——frameworks/base/libs/androidfw

libaudioutils.so——system/media/audio\_utils

libbinder.so——frameworks/native/libs/binder

libbluedroid.so——system/bluetooth/bluedroid

libc.so——bionic/libc

libcamera\_client.so——frameworks/av/camera

libcorkscrew.so——system/core/libcorkscrew

libcpustats.so——frameworks/native/libs/cpustats

libcrypto.so——external/openssl

libcutils.so——system/core/libcutils

libdbus.so——external/dbus/dbus

libdvm.so——dalvik/vm

libemoji.so——frameworks/opt/emoji

libETC1.so——frameworks/native/opengl/libs

libgccdemangle——external/gcc-demangle

libgui.so——frameworks/native/libs/gui

libgabi++.so——abi/cpp

libGLESv1\_CM.so——frameworks/native/opengl/libs

libharfbuzz.so——external/harfbuzz

libhwui.so——frameworks/base/libs/hwui

libhardware\_legacy.so——hardware/libhardware\_legacy

libjpeg.so——external/jpeg

libmedia.so——frameworks/av/media/libmedia

libmedia\_native.so——frameworks/av/media/libmedia\_native

libnetutils.so——system/core/libnetutils

libstagefright\_foundation.so——frameworks/av/media/libstagefright/foundation

libsonivox.so——external/sonivox

libspeexresampler——external/speex

libstlport.so——external/stlport

libssl.so——external/openssl

libui.so——frameworks/native/libs/ui

libutils.so——frameworks/native/libs/utils

libusbhost.so——system/core/libusbhost

## 子系统分布



# android 手机是如何研发出来的

android手机组成：硬件（CPU芯片，触屏，话筒，扬声器，相机，天线，电池，PCB，各种IC等） ＋ 软件：android系统（内核，rom系统，第三方配件驱动等）

oem厂商基于某一平台（高通，联发科，展讯）规划好自己的产品，买来硬件（或者自己生产），组装，烧录系统（根据自己的硬件设备，配件，修改后的aosp，加上第三方驱动）。

现在的像高通等芯片厂商的集成度，越来越高，研制一部android手机的门槛越来越低，但是做一部，高性能，流畅，高度优化，美观，高用户体验的产品，还是需要投入大量资源的。比如基于高通8953平台，研发一部android手机，8953平台就是骁龙625，高通会拿到 aosp源码，针对自己的平台加入相关的驱动，接口代码，厂商购买8953平台，可以拿到 高通修改后的aosp源码，平台说明文档，其他硬件接口文档等。如果厂商想使用莱卡的摄像头，三星的OLED曲面屏幕，这些配件厂商会提供相关的硬件驱动和接口文档，oem厂商集成到系统中，可以进行个性化定制，包括功能，性能，ui等。

# 参考

[Android系统应用的开发和测试](http://www.epubit.com.cn/book/onlinechapter/30644)

[Android 源代码目录结构1 - bionic](http://blog.csdn.net/kickxxx/article/details/6927272)

[Android init.rc文件解析过程详解(一)](http://blog.csdn.net/mk1111/article/details/16357327)

[Android的权限机制之—— “沙箱”机制sharedUserId和签名](http://dengzhangtao.iteye.com/blog/1989065)

Android系统开篇：

http://gityuan.com/android/#%E4%B8%80%E5%BC%95%E8%A8%80