## 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：

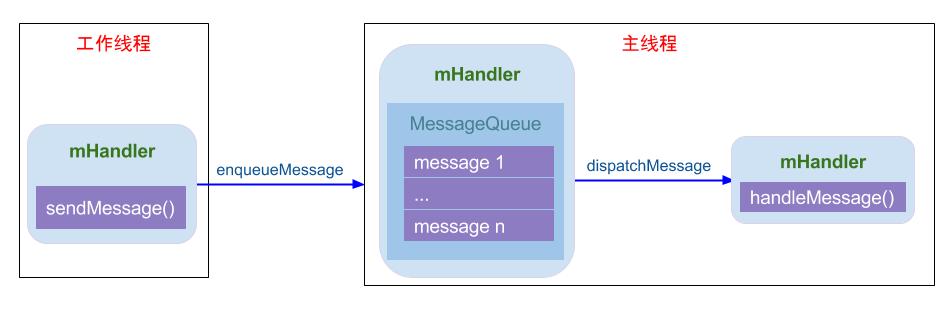
* zygote：用于孵化进程，系统进程system\_server孵化进程时便通过socket向zygote进程发起请求；
* installd：用于安装App的守护进程，上层PackageManagerService很多实现最终都是交给它来完成；
* lmkd：lowmemorykiller的守护进程，Java层的LowMemoryKiller最终都是由lmkd来完成；
* adbd：这个也不用说，用于服务adb；
* logcatd:这个不用说，用于服务logcat；
* vold：即volume Daemon，是存储类的守护进程，用于负责如USB、Sdcard等存储设备的事件处理。

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