# Todo

PTP模式切换

**震动**

[**https://blog.csdn.net/u010164190/article/details/51866419**](https://blog.csdn.net/u010164190/article/details/51866419)

**外设架构**

**https://blog.csdn.net/A8316124/article/details/78560507**

**https://blog.csdn.net/maochengtao/article/details/46849451**

**服务怎么添加**

**https://blog.csdn.net/csdn1126274345/article/details/79549124**

等关键字来适配系统feat

https://source.android.com/devices/tech/

系统裁剪

Adb disable-versity 原理，标志存在system里面的？重刷system之后，需要重新disable

Libcore

Libnativehelper

Developers

Toolchain

Tools

Development

bionic

dji.json 网络配置

C:\Users\key.guan\Desktop\20172300.log ANR剖析

C:\Users\key.guan\Desktop\anr\_log.log

Chaechrevorey:MISUC 为何不会擦除，LAST\_LOG

Ps等Android命令研究

if (mSystemPropertyUpdater == null) {

mSystemPropertyUpdater = new Runnable() {

@Override public void run() {

synchronized (mLock) {

for (int i = mRoots.size() - 1; i >= 0; --i) {

mRoots.get(i).loadSystemProperties();

}

}

}

};

SystemProperties.addChangeCallback(mSystemPropertyUpdater);

}

* [https://blog.csdn.net/s278777851/article/details/6956226](javascript:%20void%200)

*只更新系统分区*

*Data分区的数据是不会变的*

*然后每次开机为何都那么慢呢？*

*06-15 10:03:30.270 3271-3271/? E/dex2oat: Could not create image space with image file '/system/framework/boot.art'. Attempting to fall back to imageless running. Error was: Only the zygote can create the global boot image.*

*Attempted image: /system/framework/boot.art*

*06-15 10:03:30.271 3271-3271/? E/dex2oat: Dex file fallback disabled, cannot continue without image.*

*Failed to create runtime*

*06-15 10:03:30.290 3273-3273/? E/SchedPolicy: open of /dev/blkio/tasks failed: Permission denied*

*open of /dev/blkio/bg\_non\_interactive/tasks failed: Permission denied*



* 17:05

[https://github.com/KoVszone/GamePad](javascript:%20void%200)

* 17:24

[http://www.technorange.com/2017/01/how-to-map-gamepad-or-joystick-on-android-to-play-games-using-usb-bt-joycenter/](javascript:%20void%200)

[https://blog.csdn.net/AWNUXCVBN/article/details/12285761](javascript:%20void%200)

* 20:2

[https://blog.csdn.net/luoshengyang/article/details/6882903](javascript:%20void%200)

JNI层，JNI的架构和实现方式，虚拟机原理？

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

# 专家

**Yuanhuihui**

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

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

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

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

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

# 交流

# 基本概念

安卓总体架构是在 Linux内核基础上，增加硬件抽象层（HAL），运行库，java虚拟机，程序框架等组成的，具体如下图 --------------------- 本文来自 angle\_birds 的CSDN 博客 ，全文地址请点击：<https://blog.csdn.net/angle_birds/article/details/16801533?utm_source=copy>



安卓的应用程序是从application framework层架构上建立的。所有APK应用程序都是通过framework层来运行的。application framework是google写好的，除非自己深度定制，一般是不会更改这个层的。对于驱动开发来讲，我们要做的就是让framework层能认识并操作我们的硬件设备就OK了。因此我们关心主要有3个层面： linux Kernel层 HAL层 JNI层

1.       linuxKernel：是google在linux内核基础上，专门为移动设备优化后的内核，增加修改一些东西，担修改的不多，对于内核驱动来讲，基本没有修改，做过linux驱动开发的人应该很容易理解。

2.       HAL，硬件抽象层：简单来说，就是对Linux 内核驱动程序的封装，向上提供接口，屏蔽低层的实现细节。也就是说，把对硬件的支持分成了两层，一层放在用户空间（User Space），一层放在内核空间（Kernel Space），其中，硬件抽象层运行在用户空间。用户空间不属于内核不必遵守GPL协议，各个厂商可以把与自己硬件设备相关，具有商业机密的一些代码放在HAL层。

3.       JNI层：提供java和底层C、C++的动态链接库的接口。我理解的是JNI就是一个代理，可以把C和C++生成的接口函数翻译成Java可用，提供给framework层。



## 振动系统开发过程

VibratorService即安卓震动服务，是Android系统中一个和硬件相关的服务，管理和驱动着设备的振动器。在Android手持设备，如手机，平板等，振动器是不可或缺的硬件设备，在给用户震动反馈的用户交互中发挥了举足轻重的作用。 既然是和硬件相关的服务，那么以Android系统的架构模式，VibratorService将对应以下架构 --------------------- 本文来自 FamilyYuan 的CSDN 博客 ，全文地址请点击：https://blog.csdn.net/myfriend0/article/details/55210074?utm\_source=copy



public class VibratorService extends IVibratorService.Stub implements InputManager.InputDeviceListener { } ---------------------

接上代码： public class VibratorService extends IVibratorService.Stub implements InputManager.InputDeviceListener { } 1 2 3 上述代码可以看到，VibratorService继承了IVibratorService.Stub，也就是说，VibratorService直接支持Android IPC Binder通信，那么上层APP就可以通过AIDL使用VibratorService，驱动设备上的振动器，从而给用户震动的反馈。 VibratorService提供哪些接口可以给上层使用呢？看看接口IVibratorService： public interface IVibratorService extends android.os.IInterface{ public boolean hasVibrator(); public void vibrate(); public void vibratePattern(); public void cancelVibrate(); } 1 2 3 4 5 6 上述代码可以看到，VibratorService提供上述四个接口给上层调用，hasVibrator()判断是当前设备否有支持振动器，然后通过vibrate()或vibratePattern()驱动震动器发起震动，前者只需要关心参数milliseconds，即震动的时间，时间结束后，震动停止，后者需要关心的参数有pattern和repeat，pattern是long型的数组，保存的是每次震动持续的时间，即milliseconds，repeat固然就是重复的次数。所以vibrate()是一次性振动器，vibratePattern()是重复多次震动。cancelVibrate()固然是取消震动了 ---------------------

### VibratorService驱动硬件

Sdf

以APP请求震动服务为例，看看这个过程，首先是APP调用Vibrator.vibrate()，然后IPC通信到VibratorService到：

### Java应用层

public class VibratorService extends IVibratorService.Stub

implements InputManager.InputDeviceListener {

@Override // Binder call

public void vibrate(int uid, String opPkg, long milliseconds, int usageHint,

IBinder token) {

if (mContext.checkCallingOrSelfPermission(android.Manifest.permission.VIBRATE)

!= PackageManager.PERMISSION\_GRANTED) {

throw new SecurityException("Requires VIBRATE permission");

}

Vibration vib = new Vibration(token, milliseconds, usageHint, uid, opPkg);

try {

synchronized (mVibrations) {

......

startVibrationLocked(vib);

}

} finally {

Binder.restoreCallingIdentity(ident);

}

}

}

看上述代码，首先是通过checkCallingOrSelfPermission()检测调用者APP是否已经有android.Manifest.permission.VIBRATE这个权限，所以APP要使用震动服务，需要在AndroidManifest.xml中声明权限android.Manifest.permission.VIBRATE的值android.permission.VIBRATE权限，权限保护等级为normal，即安装时授权。

随后，把token, milliseconds, usageHint, uid, opPkg封装到Vibration对象中，那么就把震动更好地抽象成一次震动对象，通过startVibrationLocked继续传输：

private void startVibrationLocked(final Vibration vib) {

......

if (vib.mTimeout != 0) {

doVibratorOn(vib.mTimeout, vib.mUid, vib.mUsageHint);

mH.postDelayed(mVibrationRunnable, vib.mTimeout);

} else {

// mThread better be null here. doCancelVibrate should always be

// called before startNextVibrationLocked or startVibrationLocked.

mThread = new VibrateThread(vib);

mThread.start();

}

}

1

2

3

4

5

这里有一个分支，判断依据是vib.mTimeout，vib.mTimeout什么时候等于0，什么时候不等于0呢？回顾Vibration的初始化过程：

public class VibratorService extends IVibratorService.Stub

implements InputManager.InputDeviceListener {

@Override // Binder call

public void vibrate(int uid, String opPkg, long milliseconds, int usageHint,

IBinder token) {

......

Vibration vib = new Vibration(token, milliseconds, usageHint, uid, opPkg);

......

}

}

private class Vibration implements IBinder.DeathRecipient {

private final IBinder mToken;

private final long mTimeout;

......

Vibration(IBinder token, long millis, int usageHint, int uid, String opPkg) {

this(token, millis, null, 0, usageHint, uid, opPkg);

}

private Vibration(IBinder token, long millis, long[] pattern,

int repeat, int usageHint, int uid, String opPkg) {

mToken = token;

mTimeout = millis;

......

}

在vibrate()这个方法中可以看到，实例化Vibration对象时，传入了milliseconds，在Vibration的构造方法中，把milliseconds赋值给mTimeout，也就是说mTimeout保存的是震动持续的时间。前文提到，启动震动有两个接口，如下：

public class SystemVibrator extends Vibrator {

......

public void vibrate(..... long milliseconds .....) {}

public void vibrate(..... long[] pattern, int repeat .....) {

}

1

5

一个是带milliseconds的作为参数的，另外一个是不带milliseconds参数的，前文有说明，前者是一次性震动，后者是重复多次震动，那么，也就是在下面的代码中：

private void startVibrationLocked(final Vibration vib) {

......

if (vib.mTimeout != 0) {

doVibratorOn(vib.mTimeout, vib.mUid, vib.mUsageHint);

mH.postDelayed(mVibrationRunnable, vib.mTimeout);

} else {

// mThread better be null here. doCancelVibrate should always be

// called before startNextVibrationLocked or startVibrationLocked.

mThread = new VibrateThread(vib);

mThread.start();

}

}

1

2

3

4

5

6

7

8

9

10

11

12

13

一次性震动走的是vib.mTimeout != 0的情况，重复震动的走的是!(vib.mTimeout != 0)的情况，那么这里的代码就很符合这个实际情况了，vib.mTimeout != 0时，直接调用doVibratorOn()启动震动，而重复震动时，通过VibrateThread线程实现重复震动的功能，这里就不在赘述这个重复的实现过程了。

接着追踪doVibratorOn()方法：

private void doVibratorOn(long millis, int uid, int usageHint) {

synchronized (mInputDeviceVibrators) {

......

if (vibratorCount != 0) {

......

for (int i = 0; i < vibratorCount; i++) {

mInputDeviceVibrators.get(i).vibrate(millis, attributes);

}

} else {

vibratorOn(millis);

}

}

}

1

2

3

4

5

6

7

8

9

10

11

12

13

14

这里会考虑多个振动器的情况，本文默认只有一个，则直接调用vibratorOn(millis)这个方法：

public class VibratorService extends IVibratorService.Stub

implements InputManager.InputDeviceListener {

native static void vibratorOn(long milliseconds);

}

1

2

3

4

如上述代码，vibratorOn()是一个native方法，也就通过JNI，调用到Native framework了，传递的参数只有一个，就是mills，震动持续的时间。

### JNI框架层

VibratorService的JNI实现是frameworks/base/services/core/jni/com\_android\_server\_VibratorService.cpp中，继续看vibratorOn()这个方法：

static void vibratorOn(JNIEnv\* /\* env \*/, jobject /\* clazz \*/, jlong timeout\_ms)

{

if (gVibraDevice) {

int err = gVibraDevice->vibrator\_on(gVibraDevice, timeout\_ms);

......

} else {

ALOGW("Tried to vibrate but there is no vibrator device.");

}

}

1

### 10硬件抽象层 –

Android 封装了对底层驱动的调用，成为硬件抽象层。 ---

这里有个对象gVibraDevice，gVibraDevice实际代表的就是振动器，有振动器，才会实例化gVibraDevice。gVibraDevice调用了vibrator\_on函数。先看gVibraDevice的定义，在hardware/libhardware/include/hardware/vibrator.h中：

typedef struct vibrator\_device {

struct hw\_device\_t common;

/\*\* Turn on vibrator

\*/

int (\*vibrator\_on)(struct vibrator\_device\* vibradev, unsigned int timeout\_ms);

/\*\* Turn off vibrator

\*/

int (\*vibrator\_off)(struct vibrator\_device\* vibradev);

} vibrator\_device\_t;

1

2

vibrator.h中定义了vibrator\_device\_t，以及它的函数vibrator\_on和vibrator\_off，再继续看调用gVibraDevice->vibrator\_on(gVibraDevice, timeout\_ms)如何发起震动。vibrator\_on的具体实现是在hardware/libhardware/modules/vibrator/vibrator.c 中，先看如下代码：

static int vibra\_open(const hw\_module\_t\* module, const char\* id \_\_unused,

hw\_device\_t\*\* device \_\_unused) {

......

vibrator\_device\_t \*vibradev = calloc(1, sizeof(vibrator\_device\_t));

......

vibradev->vibrator\_on = vibra\_on;

vibradev->vibrator\_off = vibra\_off;

\*device = (hw\_device\_t \*) vibradev;

return 0;

}

1

2

3

4

上述C语言代码中，vibra\_open函数中在初始化震动器是就被调用了，vibra\_open在vibrator\_on的前面，在本文就不再赘述这个过程。通过上述代码可知，gVibraDevice->vibrator\_on(gVibraDevice, timeout\_ms)实际是调用了函数vibra\_on，继续看vibra\_on的实现：

static int vibra\_on(vibrator\_device\_t\* vibradev \_\_unused, unsigned int timeout\_ms)

{

/\* constant on, up to maximum allowed time \*/

return sendit(timeout\_ms);

}

---------------------

直接调用了函数sendit(timeout\_ms)，往下看：

static int sendit(unsigned int timeout\_ms)

{

int to\_write, written, ret, fd;

char value[20]; /\* large enough for millions of years \*/

fd = TEMP\_FAILURE\_RETRY(open(THE\_DEVICE, O\_RDWR));

to\_write = snprintf(value, sizeof(value), "%u\n", timeout\_ms);

written = TEMP\_FAILURE\_RETRY(write(fd, value, to\_write));

......

return ret;

}

1

在这里，打开设备，写入数据。到此，本文就不再往下继续分析这个过程了，

### 驱动层g

就一个设备描述文件来搞的，如此简单

跟平台有关系

MTK6573

驱动实现移植 以MTK 6573平台为例  ./mediatek/platform/mt6573/kernel/drivers/vibrator/vibrator.c 操作设备

首先打开手机调试，连接USB，执行adb shell，进入/sys/devices/timed\_output/vibrator/ 执行

 echo "10000" > enable 发现手机在震动

# echo "10000" > enable

echo "10000"  > enable  10000 enable

执行 cat enable 可以查看当前震动时间剩余数：

# cat enable  cat enable  0

https://blog.csdn.net/jzjhome/article/details/71173144

但是我们的驱动在另一块芯片实现的

因此

### 参考

Android System Server大纲之VibratorService

https://blog.csdn.net/myfriend0/article/details/55210074

s://blog.csdn.net/angle\_birds/article/details/16801533?utm\_source=copy

# 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。

# Android开篇

## 概述

ndroid系统非常庞大，底层是采用Linux作为基底，上层采用带有虚拟机的Java层，通过通过JNI技术，将上下打通，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。



为了能够更深入地掌握Android整个架构思想，以及每块之间是如何衔接与配合工作的，计划以Android系统启动过程为主线，来详细展开对Android全方位的分析，争取各个击破

## 系统启动

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。



自我介绍：

### 进程视角

f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

Df

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

启动Kernel的0号进程，初始化进程管理、内存管理，加载驱动程序等相关工作；

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（2号进程），是Linux系统的内核进程，是所有内核进程的鼻祖；

#### Native

init进程启动Media Server、servicemanager等重要服务

init进程孵化出各种用户守护进程；

init进程孵化出Zygote进程，这是第一个Java进程，包含虚拟机等内容；

#### Framework

Zygote进程，是由init通过解析init.rc文件后，fork出来生成的，主要工作包含：

加载ZygoteInit类，注册Zygote Socket服务端套接字；

加载虚拟机；

preloadClasses；

preloadResouces；

Zygote进程fork出System Server进程，System Server是Zygote孵化的第一个进程，地位非常重要；

由Media Server负责启动 C++ framework，包含AudioFlinger，Camera Service等服务；

由System Server负责启动 Java framework，包含ActivityManagerService,PowerManagerService等服务；

#### App

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；

## 计划提纲

F

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”。

（1）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—init篇]

[Android系统启动—Zygote篇]

[Android系统启动—SystemServer篇]

（2）再则就是在整个架构中有大量的服务，通过Binder系列文章，可知所有服务都是基于Binder来交互的，那么接下来，需要抓核心服务来重点分析，计划分别用文章来对核心服务展开剖析：

Android服务篇-ActivityManagerService

Android服务篇-PackageManagerService

Android服务篇-PowerManagerService

Android服务篇-BatteryService

Android服务篇-WindowManagerService

（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。

# Android文件说明

## **jar与sources.jar**

编译后的class文件。这种文件是看不到源码的

Attach Source.关联源码。对应的source.jar里面

## **jar和aar区别：**

\*.jar：只包含了class文件与清单文件，不包含资源文件，如图片等所有res中的文件。

\*.aar：包含所有资源，class以及res资源文件全部包含

如果你只是一个简单的类库那么使用生成的\*.jar文件即可；如果你的是一个UI库，包含一些自己写的控件布局文件以及字体等资源文件那么就只能使用\*.aar文件。

注意：在更改Android的系统应用时，要引用一个aar文件。之前引用的主要是jar包，而aar文件包含Android的资源文件，如：布局、样式、图片等，如果按照源码中jar的引用方式会遇到编译不过的问题，提示找不到相关的资源文件。

LOCAL\_AAPT\_FLAGS := \ --auto-add-overlay \ --extra-packages <aar package name>

[引用一个aar文件](http://www.voidcn.com/article/p-cmriynxn-bqy.html)

# Native

Midware也是这么用的

注册函数

静态注册

模仿systemUI就好了吧 建立一下java的TcpLinkHandler

修改link源码，完成TcpLinkHandler的注册加载：主要在main.cpp调用？

Java代码，声明native方法

总是加载失败

依赖

[引用jar及so文件](https://blog.csdn.net/zuiwuyuan/article/details/48201655)

# QA

## 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)

# startService的原理分析

http://blog.csdn.net/luoshengyang/article/details/6677029

# android 通信

Application-四大组件通信

## Activity通信

隐式调用，有利于降低发送者和接收者之间的耦合，它一般用在没有明确指出目标组件名称的前提下

### Intent Filter match过程

#### 使用规范

隐式 Intent

Intent intent = new Intent();

intent.setAction("com.wooyun.test"); startActivity(intent);

* a. action匹配规则：要求intent中的action 存在 且 必须和过滤规则中的其中一个相同 区分大小写；
* b. category匹配规则：系统会默认加上一个android.intent.category.DEAFAULT，所以intent中可以不存在category，但如果存在就必须匹配其中一个；
* c. data匹配规则：data由两部分组成，mimeType和URI，要求和action相似。如果没有指定URI，URI但默认值为content和file（schema）

### 源码分析

https://www.yuanmas.com/info/9ezZpgB0y6.html

主线流程：先match action, 再match data, 最后match category



# QA

1. adb install –r A.apk 执行原理？tmp目录作用，INSTALL\_FAILED\_ILLEGITIMATE\_APK各种安装报错原因，cnt的作用，为啥可以。
2. 系统目录结构是啥
3. App和系统签名的原理是什么？
4. 在sws下正常运行as，执行一次系统签名呢

framework如何更新呢？

pms包管理服务分析-PackageManagerService构造函数和包扫描过程

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

限制第三方cnt原理

updated-package 是否是判断已经升级的标志，在settings界面的时候

SystemServer工作原理，Zygote如何启动的？

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

# 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)

# Task

Settings.Global

Wifi热点的ip的查询

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