# Todo

**mPm**.hasSystemFeature(PackageManager.***FEATURE\_TELEVISION***)

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

# 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 矢量图

## Why SVG

**可缩放矢量图形**（英语：**Scalable Vector Graphics**，**SVG**）是一种基于可扩展标记语言（XML），用于描述二维矢量图形的图形格式。SVG由W3C制定，是一个开放标准。——摘自[维基百科](https://link.jianshu.com?t=https:/zh.wikipedia.org/wiki/%E5%8F%AF%E7%B8%AE%E6%94%BE%E5%90%91%E9%87%8F%E5%9C%96%E5%BD%A2)

.svg格式相对于.jpg、.png甚至.webp具有较多优势，我认为核心有两点：

* **省时间**。图像与分辨率无关，收放自如，适配安卓机坑爹的分辨率真是一劳永逸；
* **省空间**。体积小，一般复杂图像也能在数KB搞定，图标更不在话下。

## VectorDrawable

VectorDrawable是Google从Android 5.0开始引入的一个新的Drawable子类，能够加载矢量图

Android中的VectorDrawable只支持SVG的部分属性，相当于阉割版。

它虽然是个类，但是一般通过配置xml再设置到要使用的控件上。在Android工程中，在资源文件夹res/drawable/的目录下（没有则需新建），通过<vector></vector>标签描述，例如svg\_ic\_arrow\_right.xml：

<vector xmlns:android="http://schemas.android.com/apk/res/android"

android:width="8dp"

android:height="8dp"

android:viewportHeight="24.0"

android:viewportWidth="24.0">

<path

android:fillColor="#ffffff"

android:pathData="M12,4l-1.41,1.41L16.17,11H4v2h12.17l-5.58,5.59L12,20l8,-8z"/>

</vector>

作者：woxingxiao  
链接：https://www.jianshu.com/p/0555b8c1d26a  
來源：简书  
简书著作权归作者所有，任何形式的转载都请联系作者获得授权并注明出处。

何形式的转载都请联系作者获得授权并注明出处。

## 获得矢量图

### 图片资源编译工具Metro Studio

模板相当的多！支持SVG/PNG!



inker在线绘制矢量图的网站[http://app.inker.co](https://link.jianshu.com/?t=http://app.inker.co/)

svg收费的！

矢量编辑软件。比如：安装illustrator／sketch／Coreldraw/InDesign等。

Illustrator：要注册，试用一下，

Sketch只支持苹果的

### [iconfont](https://link.jianshu.com/?t=http://iconfont.cn)

前言部分已经墙裂推荐过，感觉我已经离不开它了=￣ω￣=

第一步，搜索你要的资源名字，中英文一般都会有结果。比如“arrow”

https://www.jianshu.com/p/0555b8c1d26a

### 标准svg 转->android svg

文件里好多标签Android是不认识的。不过没关系，有三种解决办法：

#### svg2VectorDrawable

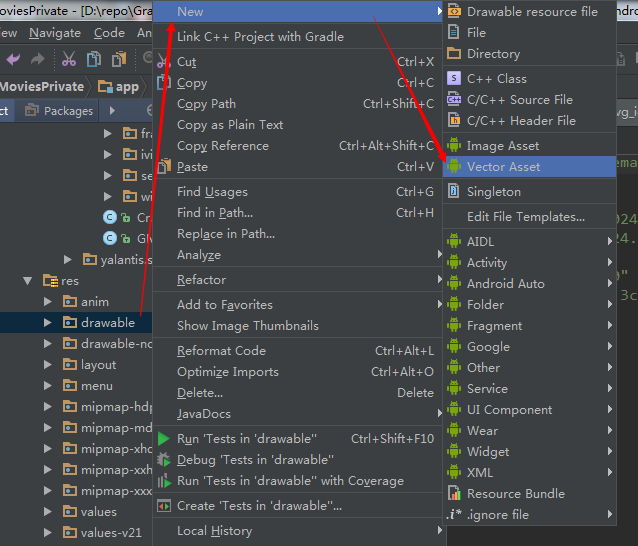
* 大招。先放大招，大招之下，后两种可以自动忽略。经大神[@天之界线2010](https://www.jianshu.com/u/denPHP)在评论区力荐的[svgtoandroid](https://link.jianshu.com?t=https:/github.com/misakuo/svgtoandroid)插件，用过之后果然神清气爽。安装：File -> Setting -> Plugins -> Browser repositories -> 搜“svg2VectorDrawable” -> 安装并重启Android Studio，再次进来后顶部工具栏会多一个图标：

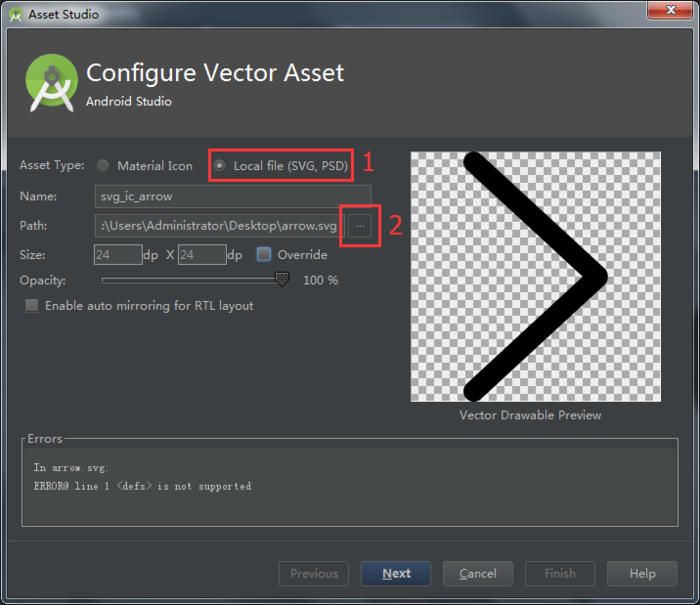
#### 手动编辑xml

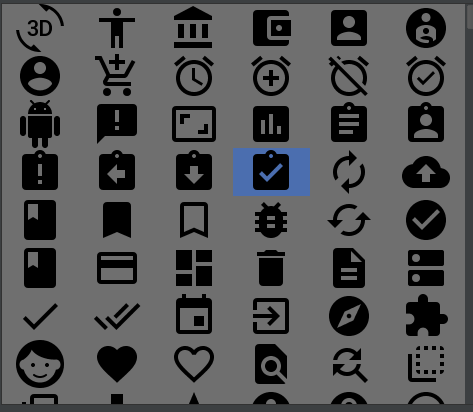
新建一个<vector></vector>标签的xml文件，通过观察文件内容，很容易获取到关键信息。width height自然对应<vector/>中宽高，viewBox后两位数字是分别对应<vector/>中的viewportWidth和viewportHeight，往下<path/>中的d的数据的对应<vector/>中<path/>中的pathData。fillColor自己手动设置。

#### 自带工具android studio

提供了丰富的图片资源，可以右键module,new->vector asset选择，如下







这个会主动展平层级结构

### 常见问题

#### 无法预览

有些时候，可以预览，有些时候不行，为啥呢？

但是可以运行

#### 小数点太长了

无法正常渲染

保持为整数就好了吧。。。

## vector 属性

android:name 定义该drawable的名字

android:width 定义该 drawable 的内部(intrinsic)宽度,支持所有 Android 系统支持的尺寸，通常使用 dp

android:height 定义该 drawable 的内部(intrinsic)高度,支持所有 Android 系统支持的尺寸，通常使用 dp

android:viewportWidth 定义矢量图视图的宽度，视图就是矢量图 path 路径数据所绘制的虚拟画布

android:viewportHeight 定义矢量图视图的高度，视图就是矢量图 path 路径数据所绘制的虚拟画布

android:tint 定义该 drawable 的 tint 颜色。默认是没有 tint 颜色的

android:tintMode 定义 tint 颜色的 Porter-Duff blending 模式，默认值为 src\_in

android:autoMirrored 设置当系统为 RTL (right-to-left) 布局的时候，是否自动镜像该图片。比如 阿拉伯语。

android:alpha 该图片的透明度属性

<group>里面可以定义多了<path>，这样可以方便管理多个<path>

## vector 标签 pathData 语法

Sdf

**[javascript]** [view plain](https://blog.csdn.net/qq1028850792/article/details/80287021) [copy](https://blog.csdn.net/qq1028850792/article/details/80287021)

1. <!--上面的手掌对应的代码实现-->
2. <vector xmlns:android="http://schemas.android.com/apk/res/android"
3. android:width="24dp"
4. android:height="24dp"
5. android:viewportWidth="24"
6. android:viewportHeight="24">
7. <path
8. android:fillColor="#000000"
9. android:pathData="
10. M22,23 q0,4 -4,4 h-7 q-2,0 -3,-1 T1,16 q-0.6,-0.8 0,-2 t5,3
11. q1,1 2,0 T8,4 q0,-1 0.9,-1.1 t1.1,1 1.5,9 q0.25,0.5 0.5,0.5
12. t0.5,-0.5 0,-11 q0.2,-1 1.1,-1.1 t1.1,1.1 1,11 q0.25,0.5 0.5,0.5
13. t0.5,-0.5 0.5,-9 q0.2,-1 1,-1 t1,1 0.5,9 q0.25,0.5 0.5,0.5
14. t0.5,-0.5 1.2,-6.5 q0.3,-1 1,-1 t0.8,1 -0.8,6 T22,23"/>
15. </vector>

### 基本语法

pathData 的指令基本都是由字母跟若干数字组成，数字之间可以用空格或者逗号隔开 (其实逗号会被忽略掉，加上逗号只是一些习惯的问题)。一般来说指令字母分为大小写两种,**大写的字母是基于原点的坐标系(偏移量)，即绝对位置；小写字母是基于当前点坐标系(偏移量)，即相对位置。**

**每个命令都有大小写形式，大写代表后面的参数是绝对坐标，小写表示相对坐标，相对于上一个点的位置。参数之间用空格或逗号隔开。**

### 移动M

move to 移动绘制点，作用相当于把画笔落在哪一点。

**M x,y (m dx, dy)** 移动虚拟画笔到对应的点，但是并不绘制。一开始的时候默认是在(0,0)。

### 直线L

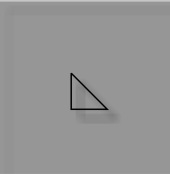
line to 直线，就是一条直线，**注意**，只是直线，直线是没有宽度的，所以你什么也看不到。

android:strokeColor="#333330" android:strokeWidth="10" 设置颜色和线宽

* **L x,y (l dx, dy)** 从当前点划一条直线到对应的点。
* **H x (h dx)** 从当前点绘制水平线，相当于l x,0
* **V y (v dy)** 从当前点绘制垂直线，相当于l 0,y

**[javascript]** [view plain](https://blog.csdn.net/qq1028850792/article/details/80287021) [copy](https://blog.csdn.net/qq1028850792/article/details/80287021)

1. <vector xmlns:android="http://schemas.android.com/apk/res/android"
2. android:width="24dp"
3. android:height="24dp"
4. android:viewportWidth="24"
5. android:viewportHeight="24">
6. <path
7. android:fillColor="#0000"
8. android:strokeColor="#000"
9. android:strokeWidth="0.2"
10. android:pathData=" M10,10 L10,15 L15,15 L10,10"/>
11. </vector>



将上述代码 android:pathData=" M10,10 L10,15 L15,15 L10,10" 替换成以下代码效果相同

android:pathData="M10,10 l 0,5 l 5,0 l-5,-5"

android:pathData="M10,10 V 15 H 15 L10,10"

android:pathData="M10,10 v 5 h 5 l-5,-5"

### 闭合Z

* **Z(或z)** 从结束点绘制一条直线到开始点，闭合路径

上面的图形型也可以由以下代码绘制  
android:pathData="M10,10 v 5 h 5 z"

### 弧线

ellipse 圆弧

* **A rx,ry x-axis-rotation large-arc-flag,sweepflag x,y**
* **a rx,ry x-axis-rotation large-arc-flag,sweepflag dx,dy**

rx ry 椭圆半径  
x-axis-rotation x轴旋转角度  
large-arc-flag 为0时表示取小弧度，1时取大弧度（要长的还是短的）  
sweep-flag 0取逆时针方向，1取顺时针方向  
x,y (dx,dy) 终点的位置

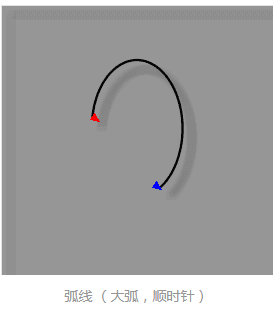
### 画圆

A（圆的x轴半径，y轴半径，x轴旋转角度，0/1-小/大弧度，0/1-逆/顺时针，圆的终点位置x点，圆的终点位置y点）

**Android 绘图和shape圆形**

**https://blog.csdn.net/linghu\_java/article/details/46403817**

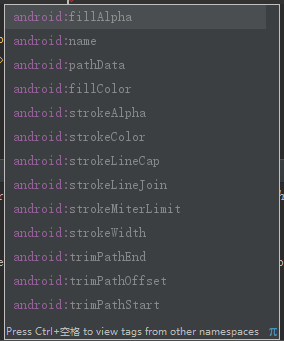
这个弧线的指令比起直线就相对复杂得多了，7个参数容易搞混了。来看个例子  
android:pathData="M8,10 a4,6 0 1,1 6 6"



[Android vector 标签 pathData 详解](https://www.jianshu.com/p/a3cb1e23c2c4)

https://www.cnblogs.com/yuhanghzsd/p/5466846.html

## path其他android属性



android:name 定义该 path 的名字，这样在其他地方可以通过名字来引用这个路径

android:pathData 和 SVG 中 d 元素一样的路径信息。

android:fillColor 定义填充路径的颜色，如果没有定义则不填充路径

android:strokeColor 定义如何绘制路径边框，如果没有定义则不显示边框

android:strokeWidth 定义路径边框的粗细尺寸

android:strokeAlpha 定义路径边框的透明度

android:fillAlpha 定义填充路径颜色的透明度

android:trimPathStart 从路径起始位置截断路径的比率，取值范围从 0 到1

android:trimPathEnd 从路径结束位置截断路径的比率，取值范围从 0 到1

android:trimPathOffset 设置路径截取的范围 Shift trim region (allows showed region to include the start and end), in the range from 0 to 1.

android:strokeLineCap 设置路径线帽的形状，取值为 butt, round, square.

android:strokeLineJoin 设置路径交界处的连接方式，取值为 miter,round,bevel.

android:strokeMiterLimit 设置斜角的上限，Sets the Miter limit for a stroked path.

## <group>

Fg

android:name 定义 group 的名字

android:rotation 定义该 group 的路径旋转多少度，这样图片就被旋转了，注意写数字的时候别晕了。

android:pivotX 定义缩放和旋转该 group 时候的 X 参考点。该值相对于 vector 的 viewport 值来指定的。

android:pivotY 定义缩放和旋转该 group 时候的 Y 参考点。该值相对于 vector 的 viewport 值来指定的。

android:scaleX 定义 X 轴的缩放倍数

android:scaleY 定义 Y 轴的缩放倍数

android:translateX 定义移动 X 轴的位移。相对于 vector 的 viewport 值来指定的。

android:translateY 定义移动 Y 轴的位移。相对于 vector 的 viewport 值来指定的。

<**vector xmlns:android="http://schemas.android.com/apk/res/android"  
 android:width="24dp"  
 android:height="24dp"  
 android:viewportWidth="72"  
 android:viewportHeight="72"**>  
 <**group  
 android:translateX="0.0"  
 android:translateY="42.0"**>  
 <**path  
 android:fillColor="?attr/wifi\_signal\_color"  
 android:pathData="M18.0,8.0l-1.0,0.0L17.0,6.0c0.0,-2.76 -2.24,-5.0 -5.0,-5.0S7.0,3.24 7.0,6.0l0.0,2.0L6.0,8.0c-1.1,0.0 -2.0,0.9 -2.0,2.0l0.0,10.0c0.0,1.0 0.9,2.0 2.0,2.0l12.0,0.0c1.1,0.0 2.0,-0.9 2.0,-2.0L20.0,10.0c0.0,-1.1 -0.9,-2.0 -2.0,-2.0zm-6.0,9.0c-1.1,0.0 -2.0,-0.9 -2.0,-2.0s0.9,-2.0 2.0,-2.0 2.0,0.9 2.0,2.0 -0.9,2.0 -2.0,2.0zm3.1,-9.0L8.9,8.0L8.9,6.0c0.0,-1.71 1.39,-3.1 3.1,-3.1 1.71,0.0 3.1,1.39 3.1,3.1l0.0,2.0z"**/>  
 </**group**>  
</**vector**>

## 性能问题

矢量图会有性能问题么，相比于png

<https://blog.csdn.net/a1961613299/article/details/50512729>

理论上讲，SVG的效率可能会不如PNG好，这是因为它需要运行时的计算和对应平台的渲染绘制。而且对于PNG来说的另一优势是在开启硬件加速的设备上，绘制Bitmap一个非常快速的过程。可以想象，让SVG不比PNG慢将是一件很有挑战的事情。

[微信的矢量化解决方案——WeChatSVGCode](https://blog.csdn.net/a1961613299/article/details/50512729)

# 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)

# 栈管理

我们知道,a ctivity 在 AMS 中的形式是 ActivityRecord,task 在 AMS 中的形式为TaskRecord,进程在 AMS 中的管理形式为 ProcessRecord。

**Android系统界面用户体验组织（TASKS），**，用户体验到的直观界面都是由很多Tasks组成的。比如，我们从最近任务栏，就可以看到很多TASKS。用户可以随意的切换到其中的一个TASK。也可以按HOME键退出一个TASK。TASK的组织形式是ActivityStack，一个ActivityStack是由好多Activity组成的堆栈。从Android系统设计者的角度来看，一个Task定义了一组行为，而这组行为是由跨多个应用程序的多个Activity组织而构成的。这个打破了以往由应用程序（进程）来定义的资源边界。一个Task就是一个场景的实现。从此构建系统用户体验和行为的边界在于TASK，而非应用程序或者进程。

## 概念

### Task

Task 是activities的集合，通过activity stack来管理，依靠先进后出队列来实现;每个task中都至少有一个activity，新实例出来的activity置于栈顶，Task可以被切换到后台

### Activity Stack

如上所诉，Activity承担了大量的显示和交互工作，从某种角度上将，我们看见的应用程序就是许多个Activity的组合。为了让这许多 Activity协同工作而不至于产生混乱，Android平台设计了一种堆栈机制用于管理Activity，其遵循先进后出的原则，系统总是显示位于栈顶的Activity，从逻辑上将，位于栈顶的Activity也就是最后打开的Activity，这也是符合逻辑的。

在操作应用程序时，每次启动新的Activity，都会将此压入Activity Stack，当用户执行返回操作时，移除Activity Stack顶上的Activity，这样就实现了返回上一个Activty的功能。直到用户一直返回到Home Screen，这时候可以理解为移除了Activity Stack所有的Activity，这个Activity Stack不再存在，应用程序也结束了运行。

可以通过 adb shell dumpsys 查看 TASKS的ActivityStacks

### task的taskAffinity

 任务归属，taskAffinity 这个属性主要是决定持有每个activity属于哪个task。 默认情况下，同一个包中的activity共享同一个affinity（任务共用性）。

### **task的launchMode**

 standard(default)：standard，标准的Activity是可以随意插入到TASK中去的一个组织结构，可以去TaskA，也可以去TaskB，也可以去TaskC，直接并无任何的联系

 singleTop，如果在任务的栈顶正好存在该Activity的实例，就重用该实例，否则就创建新的实例并放入栈顶。

 singleTask，如果在栈中已经有该Activity的实例，就重用该实例(会调用实例的onNewIntent())。重用时，会让该实例回到栈顶，因此在它上面的实例将会被移除栈。如果栈中不存在该实例，将会创建新的实例放入栈中。

 singleInstance

## 源码分析

### 主要涉及4个类

**1) ActivityRecord：activity记录的基本数据结构，**存在历史栈的一个实例，代表一个Activity。

**2) TaskRecord：**Activity栈，内部维护一个ArrayList<ActivityRecord>

**3) ActivityStack：**并不是一个Activity栈，真正意义上的Activity栈是TaskRecord，这个类是负责管理各个Activity栈，内部维护一个ArrayList<TaskRecord>

**4) ActivityStackSupervisor：**内部持有一个ActivityStack，而ActivityStack内部也持有ActivityStackSupervisor，相当于ActivityStack的辅助管理类

#### **ActivityRecord**

/frameworks/base/services/core/java/com/android/server/am/ActivityRecord.java

**void** **[dump](http://androidxref.com/5.1.0_r1/s?refs=dump&project=frameworks)**([PrintWriter](http://androidxref.com/5.1.0_r1/s?defs=PrintWriter&project=frameworks) **[pw](http://androidxref.com/5.1.0_r1/s?refs=pw&project=frameworks)**, [String](http://androidxref.com/5.1.0_r1/s?defs=String&project=frameworks) **[prefix](http://androidxref.com/5.1.0_r1/s?refs=prefix&project=frameworks)**) 定义的方法用于dumpsys activity对于2.2.1.1. ActivityRecord的打印，

final class ActivityRecord {

TaskRecord **task**; // the task this is in.

final IApplicationToken.Stub appToken;

final int userId;

int theme;

int launchMode;

...

}

成员变量task表示自己所在的TaskRecord，这样要找到自己所在的TaskRecord就不必遍历查找了。

#### TaskRecord

final class TaskRecord {

/\*\* List of all activities in the task arranged in history order \*/

final ArrayList<ActivityRecord> mActivities;

/\*\* Current stack \*/

ActivityStack stack;

}

同样的道理，成员变量stack表示自己所在的ActivityStack

#### ActivityStack

final class ActivityStack {

private ArrayList<TaskRecord> **mTaskHistory** = new ArrayList<>();

/\*\* Run all ActivityStacks through this \*/

final ActivityStackSupervisor mStackSupervisor;

ActivityStack(ActivityStackSupervisor.ActivityContainer activityContainer, RecentTasks recentTasks) {

mStackSupervisor = activityContainer.getOuter();

...

}

}

#### ActivityStackSupervisor

public final class ActivityStackSupervisor {

private ActivityStack mFocusedStack;

}

### 场景分析

#### 二、场景解析

##### 1、从桌面第一次启动App

startActivityLocked里构造一个ActivityRecord  
新建一个TaskRecord，并存入mTaskHistory  
ActivityRecord存入mActivities



final int startActivityUncheckedLocked(...) {

final int startActivityUncheckedLocked(...) {

if (reuseTask == null) {

r.setTask(targetStack.createTaskRecord(...);

...

targetStack.startActivityLocked(r, newTask, doResume, keepCurTransition, options);

...

}

}

1) TaskRecord存入mTaskHistory

TaskRecord createTaskRecord(int taskId, ActivityInfo info, Intent intent,

IVoiceInteractionSession voiceSession, IVoiceInteractor voiceInteractor,

boolean toTop) {

TaskRecord task = new TaskRecord(mService, taskId, info, intent, voiceSession,

voiceInteractor);

addTask(task, toTop, false);

return task;

}

void addTask(final TaskRecord task, final boolean toTop, boolean moving) {

task.stack = this;

if (toTop) {

insertTaskAtTop(task, null);

} else {

mTaskHistory.add(0, task);

updateTaskMovement(task, false);

}

...

}

private void insertTaskAtTop(TaskRecord task, ActivityRecord newActivity) {

...

mTaskHistory.add(taskNdx, task);

updateTaskMovement(task, true);

}

2) ActivityRecord存入mActivities



final void startActivityLocked(ActivityRecord r, boolean newTask, ...) {

...

task = mTaskHistory.get(taskNdx);

...

task.addActivityToTop(r);

}

void addActivityToTop(ActivityRecord r) {

addActivityAtIndex(mActivities.size(), r);

}

void addActivityAtIndex(int index, ActivityRecord r) {

...

mActivities.add(index, r);

...

}

##### 2、App启动一个Activity

会不会新建一个TaskRecord取决于launchMode，默认的standard模式不会创建新的TaskRecord  
构造一个ActivityRecord存入mActivities，与上面第二步一样

##### 3、回退



/\*\* @return true if this was the last activity in the task \*/

boolean removeActivity(ActivityRecord r) {

mActivities.remove(r);

...

if (mActivities.isEmpty()) {

return !mReuseTask;

}

...

return false;

}

#### 4、再次回退，回到桌面

从mActivities移除当前ActivityRecord与上面一样，只是当mActivities为空时，会触发mTaskHistory移除当前TaskRecord，如果mTaskHistory为空，则切换到桌面，给mStackSupervisor.mFocusedStack重新赋值

private void removeActivityFromHistoryLocked(ActivityRecord r, String reason) {

...

final TaskRecord task = r.task;

if (task != null && task.removeActivity(r)) {

if (mStackSupervisor.isFrontStack(this) && task == topTask() &&

task.isOverHomeStack()) {

mStackSupervisor.moveHomeStackTaskToTop(task.getTaskToReturnTo(), reason);

}

removeTask(task, reason);

}

}

void removeTask(TaskRecord task, String reason, boolean notMoving) {

...

mTaskHistory.remove(task);

...

if (mTaskHistory.isEmpty()) {

final boolean notHomeStack = !isHomeStack();

if (isOnHomeDisplay()) {

String myReason = reason + " leftTaskHistoryEmpty";

if (mFullscreen || !adjustFocusToNextVisibleStackLocked(null, myReason)) {

mStackSupervisor.moveHomeStack(notHomeStack, myReason);

}

}

...

}

...

task.stack = null;

}

如果不是从Activity调用startActivity，那么目标Activity就不知道自己该属于哪个TaskRecord，所以得指定FLAG\_ACTIVITY\_NEW\_TASK，就会新建一个TaskRecord

作者：风风风筝  
链接：http://www.jianshu.com/p/dbea4645cbc7  
來源：简书  
著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。

## REF

[Activity（三）栈管理](http://www.jianshu.com/p/dbea4645cbc7)

# 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