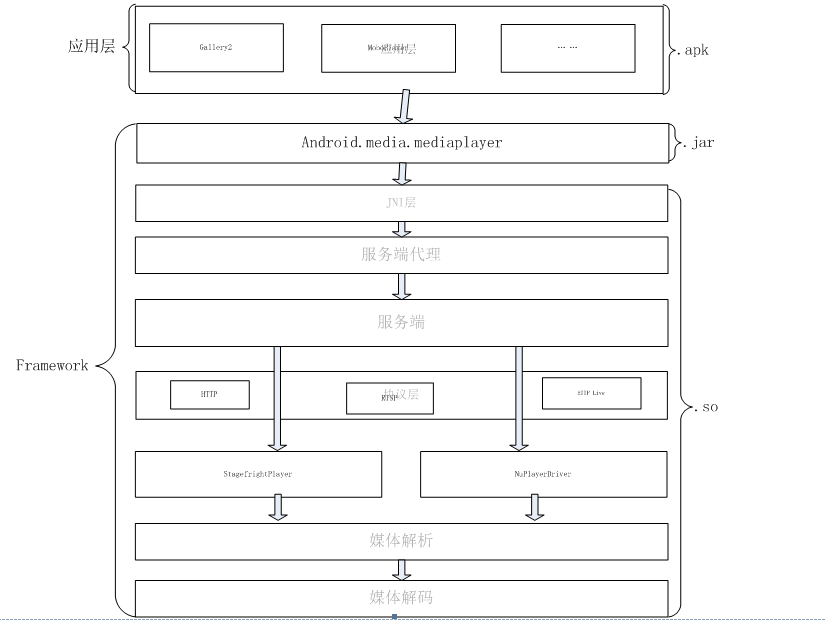
# 概述

## android多媒体结构图（上下文关系）



android多媒体结构图

从文件角度看多媒体架构：从最上层到底层调用关系依次是apk、jar、.so。   
apk就是图库，jar包是javaAPI，.so是C++代码。

## 代码目录结构

android5.1媒体的相关代码都放到framework/av/的目录中，   
其中视频相关的代码在media/下

|----- media

| |

| |-- libmedia //一个服务端的代理，对应用层提供相关的接口，和服务端交互

| |

| |-- libmediaplayerservice //服务段 （分发器，交换机的作用，个人理解）

| |

| |-- libstagefright //一个服务端的分支，支持工作（分流，解码，传输）

| |

| |-- mediaserver //android多媒体后台核心服务进程

| |

| |-- mtp //文件夹操作媒体的相关东西

| |

进一步看libstagefright目录

|-- stagefright

| |

| |-- codecs //提供解码器实现

| |

| |-- colorconversion //颜色空间转换

| |

| |-- foundation //基本数据结构的实现

| |

| |-- httplive //m3u8解析

| |

| |-- id3 // ID3 TAG解析（一般用于MP3格式的metadata容器）

| |

| |-- include //基本头文件

| |

| |-- matroska //matroska文件解析

| |

| |-- mpeg2ts //mpeg2ts文件解析和数据获取一些处理

| |

| |-- mp4 //MP4的格式解析

| |

| |-- omx //IOMX接口实现

| |

| |-- rtsp //rtsp文件解析

| |

| |-- wifi-display //关于wifi的处理

| |

| |-- yuv //YUV数据的处理

| |

| |-- AwesomePlayer //android核心播放器代码实现文件

| |

## 发展历史

因为现在android设备厂商都是用的自己的VPU（视觉处理单元）硬件设备去解码（不用自带的软解，效率太低），所以对应 有hal层的实现，其实hal只是修改了libstagefright目录中的omx和component目录的相关代码，**增加了自己的解码器组件**。再单独新建了hal层目录。总体还是遵循stagefright框架的。

接下来聊一下android多媒体框架的发展史：

android2.3以前多媒体框架用的是opencore，现在改成了stagefright框架，其中两种框架都使用了openmax标准接口（简称omx），openmax是NVIDIA和KHRONOS公司推出的多媒体应用程序标准，共分三层：应用层、集成层、开发层。其中集成层用于引擎插件（解码器）。前面提到的compoment目录就是包含组件。

四、从进程线程角度看架构

结合图一，多媒体视频播放涉及到的进程主要有gallery、mediaserver、servicemanager、surfaceflinger。

其中mediaserver进程是整个多媒体的核心，里面包含多个线程运行(有audioflinger、mediaplayerservice、cameraservice、audiopolicyservice)。当然也涉及到进程间通信的binder问题。

# Screenrecord

## 常用命令

限制视频录制时间为10s,如果不限制,默认180s

命令行显示log

指定视频分辨率大小

指定视频的比特率为6Mbps,如果不指定,默认为4Mbps. 你可以增加比特率以提高视频质量

screenrecord --time-limit 10 –verbose --size 1280\*720 --bit-rate 6000000 /sdcard/demo.mp4

adb shell screenrecord –help

参考：<https://blog.csdn.net/gdutxiaoxu/article/details/69802895>

## 源码分析

format->setInt32(**"width"**, gVideoWidth);  
format->setInt32(**"height"**, gVideoHeight);  
format->setString(**"mime"**, kMimeTypeAvc);  
format->setInt32(**"color-format"**, OMX\_COLOR\_FormatAndroidOpaque);  
format->setInt32(**"bitrate"**, gBitRate);  
format->setFloat(**"frame-rate"**, displayFps);  
format->setInt32(**"i-frame-interval"**, 10);

### 配置信号处理器

recordScreen开始的时候。

按ctl+C 触发**SIGINT**

终端断开触发SIGHUP

**static** status\_t configureSignals() {  
 **struct** sigaction act;  
 memset(&act, 0, **sizeof**(act));  
 act.sa\_handler = signalCatcher;  
 **if** (sigaction(**SIGINT**, &act, &gOrigSigactionINT) != 0) {  
 status\_t err = -errno;  
 fprintf(stderr, **"Unable to configure SIGINT handler: %s\n"**,  
 strerror(errno));  
 **return** err;  
 }  
**…**  
 signal(SIGPIPE, SIG\_IGN);  
 **return** NO\_ERROR;  
}

#### 中断信号signalCatcher

gStopRequested = **true**;主要是这个变量的控制

sigaction(SIGINT, &gOrigSigactionINT, NULL);  
sigaction(SIGHUP, &gOrigSigactionHUP, NULL);

### 告诉系统有新文件产生

其实所有生产新的文件都应该去触发一下这个事件

notifyMediaScanner(fileName)

**const char**\* kCommand = **"/system/bin/am"**;  
**const char**\* **const** argv[] = {  
 kCommand,  
 **"broadcast"**,  
 **"-a"**,  
 **"android.intent.action.MEDIA\_SCANNER\_SCAN\_FILE"**,  
 **"-d"**,  
 fileUrl.string(),  
 NULL  
};

execv(kCommand, **const\_cast**<**char**\* **const**\*>(argv));

pid\_t pid = fork();  
**if** (pid > 0) {  
 *// parent; wait for the child, mostly to make the verbose-mode output  
 // look right, but also to check for and log failures* **int** status;  
 pid\_t actualPid = TEMP\_FAILURE\_RETRY(waitpid(pid, &status, 0));  
 **if** (actualPid != pid) {  
 ALOGW(**"waitpid(%d) returned %d (errno=%d)"**, pid, actualPid, errno);  
 } **else if** (status != 0) {  
 ALOGW(**"'am broadcast' exited with status=%d"**, status);  
 } **else** {  
 ALOGV(**"'am broadcast' exited successfully"**);  
 }  
} **else** {  
 **if** (!gVerbose) {  
 *// non-verbose, suppress 'am' output* ALOGV(**"closing stdout/stderr in child"**);  
 **int** fd = open(**"/dev/null"**, O\_WRONLY);  
 **if** (fd >= 0) {  
 dup2(fd, STDOUT\_FILENO);  
 dup2(fd, STDERR\_FILENO);  
 close(fd);  
 }  
 }  
 execv(kCommand, **const\_cast**<**char**\* **const**\*>(argv));  
 ALOGE(**"execv(%s) failed: %s\n"**, kCommand, strerror(errno));  
 exit(1);  
}

## [调用 screenrecord 实现录屏](https://www.cnblogs.com/duanguyuan/p/5971645.html)

通过Process p = Runtime.getRuntime().exec(cmd)的方式调用shell命令

需要android.permission.INTERACT\_ACROSS\_USERS\_FULL 或者 android.permission.INTERACT\_ACROSS\_USERS 权限，而这个权限是system app的权限

使用说明screenrecord –verbose

## 参考

https://blog.csdn.net/zxccxzzxz/article/details/54150396

## REF

[android多媒体分析（1）-整体架构](http://blog.csdn.net/w6980112/article/details/50408085)

[Android Multimedia框架总结（五）多媒体基础概念](http://blog.csdn.net/hejjunlin/article/details/52431887)

[Android多媒体开发(一)----MediaPlayer框架开始](http://blog.csdn.net/u010164190/article/details/54588100)