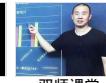
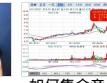
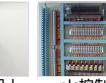


资讯 | 安全 | 论坛 | 下载 | 读书 | 程序开发 | 数据库 | 系统 | 网络 | 电子书 | 微信学院 | 站长学院 | QQ | 考试  
频道栏目 | 软件开发 | web前端 | Web开发 | 移动开发 | 综合编程 | 登录 | 注册









首页 > 程序开发 > 移动开发 > Android > 正文

## android 多媒体框架服务之StagefrightPlayer和OMXCodec实现原理学习

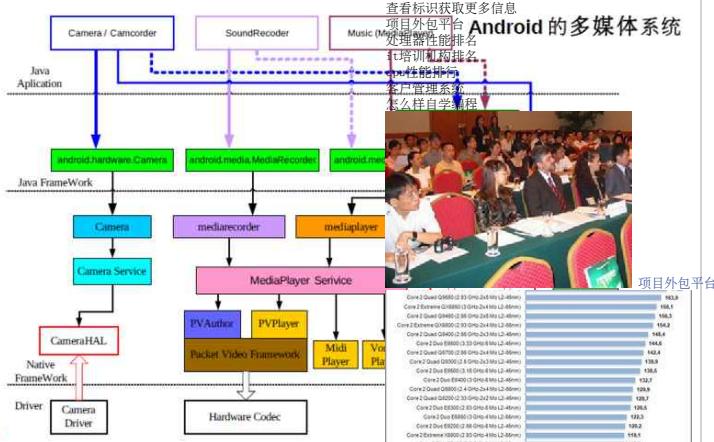
2016-08-29 09:27:03      0个评论      来源: 拖延症补丁.exe      收藏      我要投稿

[阀门维修](#)
[ubuntu](#)
[雷克萨斯上市](#)
[澳门一日游攻略](#)

[免费云主机试用一年](#)
[项目外包平台](#)
[nvidia驱动](#)
[虹吸雨水](#)
[乳胶紧身衣](#)
[一点点加盟](#)
[cpu性能排行](#)

[光端机](#)
[Linux运维](#)
[综合布线](#)
[群控系统](#)
[电脑编程入门自学](#)
[买狗网](#)
[希腊房价](#)

### 1. 框架结构



该图展示了Android多媒体系统的架构。从上到下，层级如下：

- Java Application** 层：包含 Camera / Camcorder、SoundRecorder 和 Music (Music Player)。
- Java Framework** 层：包含 android.hardware.Camera、android.media.MediaRecorder 和 android.media。
- Native Framework** 层：包含 Camera HAL、Camera Service、MediaPlayer Service、PVAudio 和 PVPlayer。
- Driver** 层：包含 Camera Driver 和 Hardware Codec。

各模块之间的关系：

- Java Application** 层通过 Java Framework 层与 Native Framework 层交互。
- Native Framework** 层通过 Camera HAL 与 Driver 层交互。
- MediaPlayer Service** 通过 PVAudio 和 PVPlayer 与 Native Framework 层交互。
- MediaPlayer Service** 通过 Camera Service 与 Java Framework 层交互。
- MediaPlayer Service** 通过 MediaPlayer 与 Driver 层交互。

右侧有“项目外包平台”、“培训课程”、“客户管理系统”、“怎么样自学编程”等链接。

#### 1.1 StageFright和openCore和NuPlayer



该图展示了MediaPlayer的架构：

- Java** 层：包含 MediaPlayer。
- Native** 层：包含 JNI\_android\_media\_MediaPlayer 和 MediaPlayerService。
- OpenCORE** 层：包含 MediaPlayer lib。

MediaPlayerService 通过 create 方法与 OpenCORE 层交互。

右侧有“处理器性能排名”、“项目外包”、“培训课程”、“客户管理系统”等链接。

上图可知, stagefright是在MediaPlayerService这一层实现的。

```

/* target="_blank"
class="keylink">>vcmXk7KiwdC1xKos1NrRodPDb3B1
/8rzcoTg+rb6twPrBy7bgtM6x5Lavo6y009fu10fPyLXE
/W1Nq1xE51UGxheWVYRJpdVmYo6zU2rmk1/e/qsq8vdC
/cHLT3B1bkNvcmXL+dLttRPcGVuQ29yZbxEwcu94r32v
/8rzclNrR3b34uf2zzNbQ0ruw47a8ysfPyMG91ta
/8rzcsqK05qOsLyLu689TZ1NrEs7j2s0axvbQvavG5NLG
/2jrNTnz8hBbmRyb21k1tDKudPDtcTKx1NOYWd1ZnPZ2
/Tw9PasqW3xc34wufB9809z0X0xLz+o6y1q8rH1Nq688C0tcRBbmRyb21kIEy
/qsq8TnVqbGF5ZXJpvaw9pb+qyrzM5rT6wctTdGFnZWZyaWdodKosxL

```

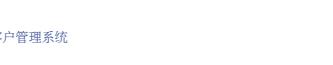


文章 | 推荐

- Echarts的学习及使用
- Sublime Text 主题模版
- Windows环境下搭建React Native
- spring注解的使用
- SpringMVC+Spring+Mybatis整合配置文件
- Spring Boot实战之单元测试
- 单点登录之apacheds之springMVC下操作
- spring和hibernate的整合



亲子度假，欢乐一夏  
暑期限时2大1小亲子早餐仅需30元起。  
即刻预订





#### OpenMAX简介

android系统中的编解码器部分用的是openmax，以后会有很多硬件厂商都可以遵循这个标准来做的自己的实现，发挥自己的优势。因为大部分的机顶盒芯片产品硬件的编解码是它的专利，所以不能在开源的平台上。以后手机高清视频硬解码也会是个趋势。

第一层：OpenMaxDL (DevelopmentLayer, 开发层)

第二层：OpenMaxIL (IntegrationLayer, 集成层)

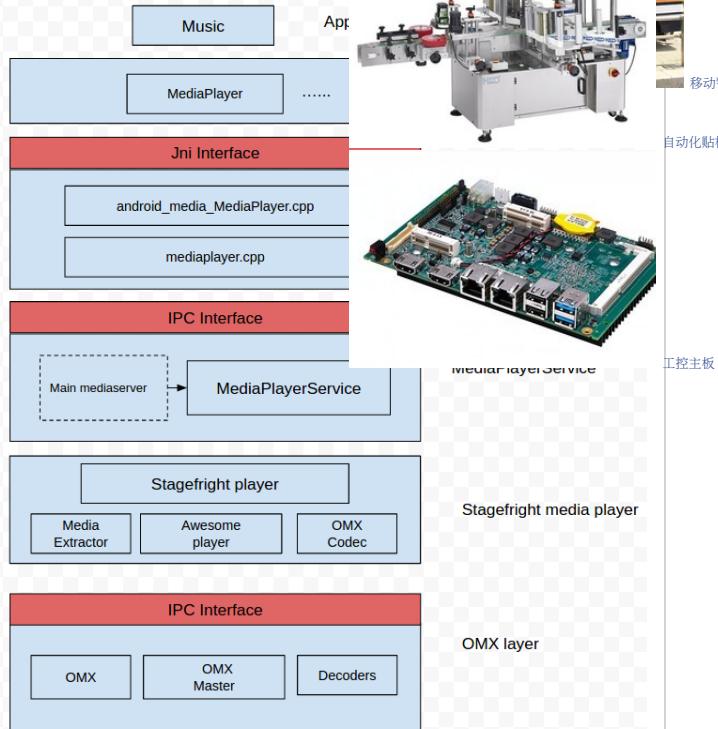
第三层：OpenMaxAL (ApplictionLayer, 应用层)

OpenMaxIL 处在中间层的位置，OpenMAX IL作为音频，视频，图像，交互，并以统一的行为支持组件（例如资源和皮肤）。这个接口是用户层的底层接口应用于嵌入式或/和移动设备。它提供一个不存在这种标准化的接口，编解码器供应商必须写私有的驱动。主要目的是使用特征集合为编解码器提供一个系统抽象，从而解决这个问题。

reference2 .

### 1.3 StageFright

基于Stagefright的MediaPlayer框架的结构:



stageFright is a player .

上图可以看出播放过程主要涉及3个进程: app端进程, 媒体框架服务(stageFright), OMX服

务.实际使用还经常用到一个专门做跨进程内存共享管理的进程(MemoryDeal).注意后面会经常讲到”客户端”,有时并不是指app端,要区分开来.

- 应用层和framework层?使用到MediaPlayer的应用很多,最常见的就是Music和Video,如果要了解这些应用的实现可以看下AOSP代码中的packages/apps,这些代码中用到了frameworks/base/media所提供的MediaPlayer接口,这些接口都十分简单,我们只需要知道这些接口的具体功能就可以开发出一款功能较为齐全的Music Player.

- Native Media Player 层:

应用层的native实现.通过Binder机制和Service交互.

Media Player Service 部分:

从Native层发出的IPC请求将会由Media Player Service 部分进行处理.MediaPlayerService是在frameworks/av/media/mediaserver/main\_mediaserver.cpp的main方法中初始化的,在main方法中还启动了多个Android系统服务比如AudioFlinger, CameraService等,实例化Media Player Service 子系统的工作包括MediaPlayerService对象的创建,以及内置底层MediaPlayer播放框架工厂的注册,一旦MediaPlayerService 服务启动,MediaPlayerService将会接受Native MediaPlayer 层的IPC请求,并且为每个操作media内容的请求实例化一个MediaPlayerService::Client 对象, Client有一个createPlayer 的方法可以使用特定的工厂类为某个特定的类型创建一个本地media player ,后续的发向native层的请求都会交给刚刚提到的native 层的 media palyer来处理,这里的 media player指的是StagefrightPlayer或者Nuplayerdriver.但是我们这里先不讨论Nuplayerdriver.

分析文件:

StageFright主要是对AwesomePlayer的封装.AwesomePlayer是事件驱动的播放器.本文主要分析它对视频流的处理.重点在AwesomePlayer::onVideoEvent函数.其中从流中read packet,parse,decode .都在mVideoSource->read(&mVideoBuffer, &options);函数完成.该函数在OMXCodec.cpp实现.其中read(extract)和parse 在AwesomePlayer调用其它组件(如MPEG4Extractor)完成,参数mVideoBuffer即为解码后的帧图像,decode则是调用OMXCodec的服务接口.也就是解码时又通过Binder做了一次跨进程通信.关于OMXCodec Service的一些文件:

- 接口定义:

IOMX.h

- 客户端类:

OMXCodec.cpp

OMXClient.cpp

IOMX.cpp (BpOMX类/BnOMX类)

- 服务端类:

OMX.cpp

OMXNodeInstance.cpp

示例函数

fillOutputBuffer. 见[源码](#)书签和注释.

以上类是通过Binder机制实现的.因为这是我第一个学习的android Framework.下面先简单介绍

Binder机制:

## 2. Binder机制简单介绍

binder是android 系统下的一种IPC机制.是进程间交互的一种方式。在开发android应用时,脑袋一定要一直保持C/S结构的思想。

android应用的开发说白了就是通过android提供的一系列的服务来完成自己的目的,咱们刚才也的那个播放器的apk也是需要android提供的播放器的服务来完成的。

apk是一个独立的进程, android的系统服务也是很多个独立的进程。binder的功能就是把client 和 service 连接起来。

### 2.1 入门实例

[Binder简单实例 - lansehai的专栏 - 博客频道 - CSDN.NET](#)

### 2.2 总结

- a. IInterface 的继承类里声明将要跨进程调用的函数,声明为纯虚函数.(如IOMX.h中IOMX类).
- a. 如果服务端和客户端的创建都在同一个进程中, interface\_cast会直接获得xx的Bn实例,就是相当于直接声明了一个xx类型. OMXClient#getOMX()
- c. BnBinder是Service端接口.binder是代理(BpBinder).Service实现业务功能.客户端需要实现发送功能.

### 2.3 Imemory

调用OMXCodec Component需要进程间共享内存.android在实现进程共享内存时使用Binder和匿名共享内存实现了一套共享内存机制。

MemoryHeapBase和MemoryBase.前者是匿名共享内存类,以页为单位.后者是基于MemoryHeapBase的匿名共享内存,使用偏移值表示,ta解决多个clients一个Service 内存共享问题.

#### 重要函数总结:

##### getMemory

成员函数getMemory用来获取内部的MemoryHeapBase对象的IMemoryHeap接口.如果成员变量mHeap的值为NULL,就表示这个BpMemory对象尚未建立好匿名共享内存,于是,就会通过一个Binder进程间调用去Server端请求匿名共享内存信息,在这些信息中,最重要的就是这个Server端的MemoryHeapBase对象的引用heap了,通过这个引用可以在Client端进程中创建一个BpMemoryHeap远程接口,最后将这个BpMemoryHeap远程接口保存在成员变量mHeap中,同时,从Server端获得的信息还包括这块匿名共享内存整个匿名共享内存中的偏移位置以及大小.这样,这个BpMemory对象中的匿名共享内存就准备就绪了.

##### pointer()

成员函数pointer()用来获取内部所维护的匿名共享内存的基地址;

##### size()

成员函数size()用来获取内部所维护的匿名共享内存的大小; offset()

用来获取内部所维护的这部分匿名共享内存整个匿名共享内存中的偏移量.

#### 使用实例:

在BpSharedBuffer类的成员函数transact中,向Server端发出了一个请求代码为GET\_BUFFER的Binder进程间调用请求,请求Server端返回一个匿名共享内存对象的远程接口IMemory,它实际指向的是一个BpMemory对象,获得了这个对象之后,就将它返回给调用者;

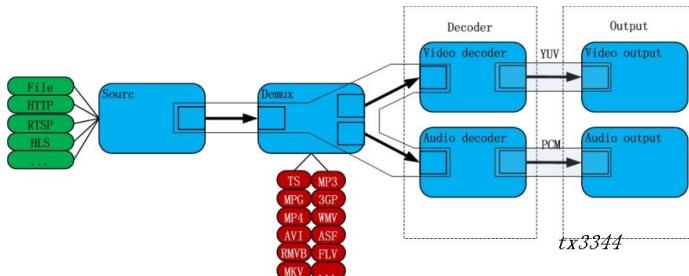
在BnSharedBuffer类的成员函数onTransact中,当它接收到从Client端发送过来的代码为GET\_BUFFER的Binder进程间调用请求后,便调用其子类的getBuffer成员函数来获一个匿名共享内存对象接口IMemory,它实际指向的是一个MemoryBase对象,获得了这个对象之后,就把它返回给Client端.

(详细示例见链接最后一节)

## 2.4 IOMX

IOMX定义了OMXCodec的接口,如fillBuffer,emptyBuffer.

## 3. StageFright底层具体实现



说白了既然StageFright是个播放器,那么它至少有4大部分: source、demux、decoder、output.

1.source : 数据源, 数据的来源不一定都是本地file, 也有可能是网路上的各种协议例如: http、rtsp、hls等。

2.demux解复用: 视频文件一般情况下都是把音视频的ES流交织的通过某种规则放在一起. 这种规则就是容器规则. 现在有很多不同的容器格式. 如ts、mp4、flv、mkv、avi、rmvb等等. demux的功能就是把音视频的ES流从容器中剥离出来, 然后分别送到不同的解码器中.

3.decoder解码: 解码器--播放器的核心模块. 分为音频和视频解码器.

4.output输出: 输出部分分为音频和视频输出. 解码后的音频 (pcm) 和视频 (yuv) 的原始数据需要得到音视频的output模块的支持才能真正的让人的感官系统 (眼和耳) 辨识到.

所以, 播放器大致分成上述4部分. 怎么抽象的实现这4大部分、以及找到一种合理的方式将这几部分组织并运动起来. 是每个播放器不同的实现方式而已.

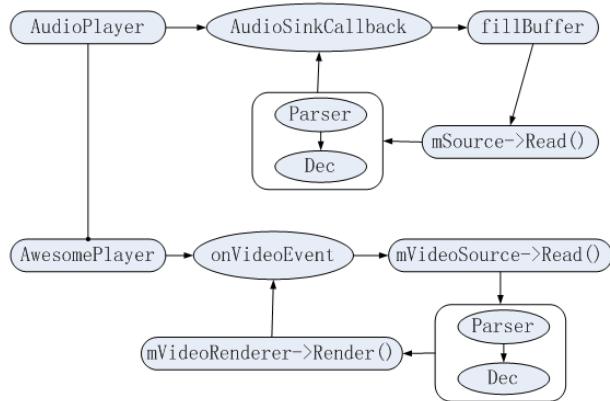
AwesomePlayer是实现播放的底层操作者, 它在StagefrightPlayer初始化的时候被创建, 它负责将对应的音频视频和对应的解码器对应起来. 这里涉及到了MediaExtractor, 它会从媒体文件中抽取到有效的头信息. 并返回对应的引用. 在准备播放的时候AwesomePlayer通过OMXCodec来根据媒体文件类型创建解码器, 解码器是驻留在OMX子系统上 (OMX是OpenMAX在Android上面的实现), 这些解码器主要用于处理内存缓冲, 转化成原始数据格式, 这部分的实现代码主要在frameworks/av/media/libstagefright/omx 以及frameworks/av/media/libstagefright/codecs 目录下, Stagefright Media Player和OMX部件(Component)是通过IPC方式交互的.

AwesomePlayer最终会处理应用层发出的播放, 暂停, 停止等请求, 这些请求往往和媒体类型有关联对于音频文件.AwesomePlayer 将会创建一个AudioPlayer来对文件进行处理, 比如当前文件只有音频

部分需要播放，这时候AwesomePlayer将会调用AudioPlayer::start()进行播放，一旦用户提交了其他新的请求AudioPlayer会使用MediaSource对象来和底层的OMX子系统进行交互。

### 3.1 StageFright和OMXCodec通信

StageFright数据处理流程



重点在read函数.从流中read packet,parse,decode .都在mVideoSource->read(&mVideoBuffer, &options);函数完成.

### 3.2 底层进程通信对缓存的管理:

read函数将数据读到缓存并处理后,如何传到OMXCodec Service?

read循环最终调用了:OMXCodec::drainInputBuffer(BufferInfo \*info),最终通过emptyBuffer传递缓存.

```

1 |   <code><code><code><code><code><code>status_t emptyBuffer(      ?
2 |           node_id node,
3 |           buffer_id buffer,
4 |           OMX_U32 range_offset, OMX_U32 range_length,
5 |           OMX_U32 flags, OMX_TICKS timestamp)</code></code></code>

```

通过分别分析客户端和服务端的emptyBuffer函数得到以下信息:

服务端:

a. 约定buffer\_id 为OMX\_BUFFERHEADERTYPE \*指针.

客户端

如何对应客户端读出的缓存给服务端?

```

1 |   <code><code><code><code><code><code>    err = mOMX->empti
2 |           mNode, info->mBuffer, 0, offset,
3 |           flags, timestampUs);</code></code></code></code></code>

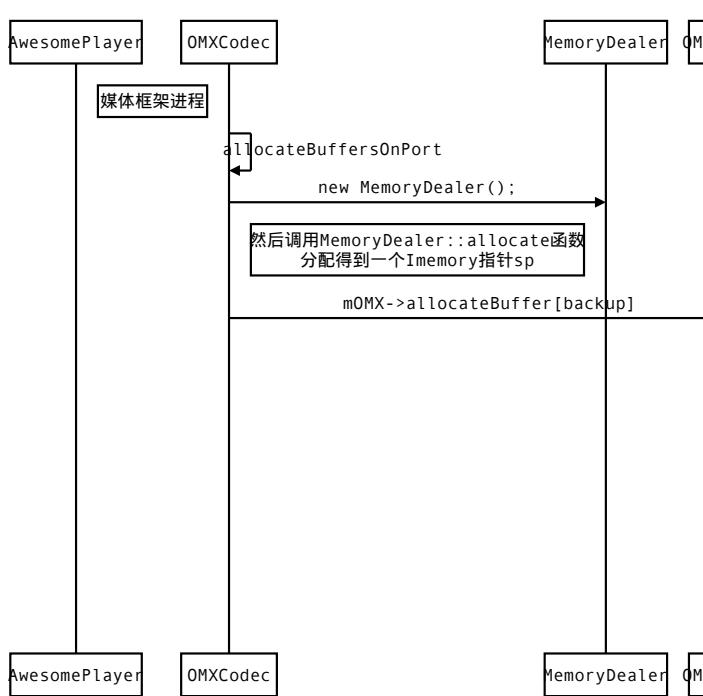
```

mBuffer即为BufferId.实质为指针.服务端转为OMX\_BUFFERHEADERTYPE \*类型后,调用CopyToOMX(header),调用OpenMAX接口,将输入buffer拷贝到Codec硬件内存.

解码后调用fillOutputBuffer将缓存从Codec拷贝回来,同样是跨进程调用,实现函数是fillBuffer.服务端fillBuffer实现把缓存拷贝到对应的buffer\_id.

如何初始化缓存?

[http://img1.ph.126.net/gasrK49\\_HwQvjPERL5s2\\_g=/6599299576238819530.png](http://img1.ph.126.net/gasrK49_HwQvjPERL5s2_g=/6599299576238819530.png)--  
客户端allocateBuffersOnPort函数初始化缓存池.这个函数比较重要,直接画图来看他的调用和IPC通信:



note right of OMXNodeInstance.cpp:调用Codec接口的地方,也就是调用芯片厂商提供的接口。

例如OMX\_AllocateBuffer分配得到.OMX\_BUFFERHEADERTYPE 指针.再通过reply写回客户端,也就是客户端可以得到一个OMX\_BUFFERHEADERTYPE 的指针.OMX\_BUFFERHEADERTYPE的一个重要成员 BufferMeta将Imemory的共享内存封装起来.最后函数返回客户端一个buffer\_id,实际就是一个在服务端保存的OMX\_BUFFERHEADERTYPE指针.然后把这个指针绑定到对应的portIndex, MMMediaPlayer使用时再绑定到对应的MediaInfo上.

这样下次OMXCodec调用drainInputBuffer这样的函数时,就能通过MediaInfo在OMXCodec Component找到对应bufferId ,进而通过emptyBuffer这样的跨进程调用通知服务端(Component),服务端有bufferId后,转为BufferHeader指针后遍知道该使用那块共享内存了.

再补充几个其中的几个信息:

通过MemoryDealer分配匿名共享内存.通过mOMX->allocateBufferWithBackup(mNode, portIndex, mem, &buffer);分配指定大小共享缓存.并绑定到buffer\_id.(以下带mOMX 字样的都是IPC通信,对应的服务器端函数在OMXNodeInstance.cpp).也就是客户端的每个bufferInfo保存了一个Service对应的OMX\_BUFFERHEADERTYPE对象指针. 将客户端的共享缓存地址通过mem->pointer();赋值给info::mData. 注意MediaBuffer的初始化:info.mMediaBuffer = new MediaBuffer(info.mData, info.mSize);MediaBuffer::mData 即为info.mData.也就是MediaSource,例如MPEG4Extractor 的read /parse 操作也是直接对共享缓存操作. OMXCodec::allocateBuffersOnPort为每个buffer分配缓存和index.同时保存缓存信息到服务端:mOMX->storeMetaDataInBuffers. 以及把缓存队列传到MediaSource(此例是MPEG4Extractor):mSource->setBuffers(buffers) 服务端分配缓存见OMXNodeInstance::allocateBufferbackup.服务端的共享内存指针保存在BufferMeta buffer\_meta.例如emptyBuffer调用CopyTOOMX可以把芯片缓存拷贝到共享内存缓存.

顺便学习以下如何写Binder机制中客户端的发送函数,一个BpBinder函数是长这样的:

```

1  <code><code><code><code><code><code><code><code><code><code><code>
2      Parcel data, reply;
3      data.writeInterfaceToken(IOMX::getInterfaceDescriptor());
4      data.writeIntPtr((intptr_t)node);
5      data.writeInt32(port_index);
6      data.writeStrongBinder(params->asBinder());
7      remote()->transact(ALLOC_BUFFER_WITH_BACKUP, data, &reply);
8      status_t err = reply.readInt32();
9  //...
10     *buffer = (void*)reply.readIntPtr();
11
12     return err;
13 }</code></code></code></code></code></code></code></code>
  
```

通过IMemory的学习可知,IOMX客户端通过MemoryDealer构建MemoryBase对象是IMemory的远程接口实现.而OMX Component的接收onTransact中,读取的是一个匿名共享内存对象的远程接口IMemory,它实质指向一个BpMemory对象.接收完通过interface\_cast(data.readStrongBinder())转为IMemory调用.

/todo

- node\_id 是啥? 怎么区分remoteOMX & localOMX?

OMXCodec.cpp不是直接调用OMXCodec的客户端接口,中间还封装了一个类OMXClient.cpp.当判断使用

remoteOMX时才调用remote接口.即BpOMX.而BpOMX在IOMX.cpp实现.node\_id是区分local/remote的关键.猜测noteId是标记媒体框架服务(MediaPlayer)的客户端程序.

Given a node\_id and the calling process' pid, returns true iff the implementation of the OMX interface lives in the same process.

猜测是如果同进程调用的话不再重新映射共享内存?

OMXObserver是干么用的?

如何读取packet,Source Input?

err = mSource->read(&srcBuffer, &options); 保存在 MediaBuffer::mData.

## MediaCodec和OMXCodec的关系.

其实在openmax接口设计中,他不能用来当编解码.通过他的组件可以组成一个完整的播放器,包括sourc、demux、decode、output.

1. android系统中只用openmax来做code,所以android向上抽象了一层OMXCodec,提供给上层播放器用.

播放器中音视频解码器mVideosource、mAudiosource都是OMXCodec的实例。

2. OMXCodec通过IOMX 依赖binder机制 获得 OMX服务, OMX服务 才是openmax 在android中实现。

3. OMX把软编解码和硬件编解码统一看作插件的形式管理起来。

由文中可知,MediaPlayer(stageFright),MediaCodec 都是调用OMXCodec,stageFright和MediaCodec是平行关系,无相互调用. OMXCodec和ACodec都是更底层的东西.

## OMXCodec和ACodec

NuPlayer调用ACodec(支持网络流).



[点击复制链接 与好友分享!](#) [回本站首页](#)

相关TAG标签 [框架](#) [原理](#) [多媒体](#)

上一篇: Material design当中Drawable新特性

下一篇: AsyncTask异步加载图片 进度条显示进度

### 相关文章

一个Android框架下实现登录和Google检  
Android多媒体框架OpenCore  
android游戏开发框架libgdx的使用 (一)  
android游戏开发框架libgdx的使用 (二)  
android游戏开发框架libgdx的使用 (三)  
android游戏开发框架libgdx的使用 (五)

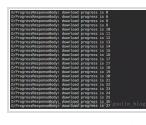
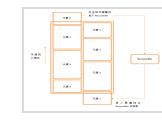
android-->游戏框架  
android游戏开发框架libgdx的使用 (一)  
【Android游戏开发之四】Android 游戏  
android游戏开发框架libgdx的使用 (四)  
android游戏开发框架libgdx的使用 (六)

热门专题推荐 [python](#) [div+css](#) [css教程](#) [html5](#) [html教程](#) [jquery](#)

Android SDK [php](#) [mysql](#) [oracle](#)



### 图文推荐



来说两句吧...

[畅言一下](#)

还没有评论，快来抢沙发吧！

**畅言商业版**

7月25日-7月31日VIP服务低价风暴来袭，5折优惠嗨翻暑假！活动期间开通VIP服务任意充值买一送一！买一个月送一个月，买一年送一年！



**畅言VIP服务 5 折优惠**

买一月送一月，买一年送一年！  
现在就加入畅言VIP大家庭！

畅言：国内最大第三方评论系统，为您专业护航。

[查看详情>](#)

广告

红黑联盟正在使用畅言