# 多媒体播放器解决方案

作者：任伟

撰写日期：2018/1/19

电话：18101021839

Email: rwhanson@hotmail.com

目录

[多媒体播放器解决方案 1](#_Toc504407214)

[1 MediaPlayer框架 1](#_Toc504407215)

[1.1 mediaserver 1](#_Toc504407216)

[1.2 MediaPlayerService 1](#_Toc504407217)

[1.2.1 MediaPlayerService::create() 2](#_Toc504407218)

[1.2.2 MediaPlayerService::Client::setDataSource() 2](#_Toc504407219)

[1.2.3 MediaPlayerService::Client::prepareAsync() 2](#_Toc504407220)

[1.2.4 MediaPlayerService::Client::start() 2](#_Toc504407221)

[1.2.5 MediaPlayerService::Client::stop() 2](#_Toc504407222)

[1.2.6 MediaPlayerService::Client::pause() 3](#_Toc504407223)

[1.2.7 MediaPlayerService::Client::reset() 3](#_Toc504407224)

[1.2.8 MediaPlayerService::Client::seekTo(int msec) 3](#_Toc504407225)

[1.3 Nuplayer 3](#_Toc504407226)

[1.3.1 NuPlayer::setDataSourceAsync() 3](#_Toc504407227)

[1.3.2 NuPlayer::prepareAsync() 4](#_Toc504407228)

[1.3.3 NuPlayer::start() 6](#_Toc504407229)

[1.3.4 NuPlayer:: seekToAsync () 9](#_Toc504407230)

[1.4 解码器 11](#_Toc504407231)

[1.4.1 EMPTY\_BUFFER\_DONE 12](#_Toc504407232)

[1.4.2 FILL\_BUFFER\_DONE 14](#_Toc504407233)

[1.4.3 音视频同步 16](#_Toc504407234)

[1.5 播放器组件交互图 18](#_Toc504407235)

[2 FFmpeg 19](#_Toc504407236)

[2.1 av\_register\_all() 19](#_Toc504407237)

[2.2 avformat\_alloc\_context() 19](#_Toc504407238)

[2.3 avformat\_open\_input() 19](#_Toc504407239)

[2.4 avformat\_find\_stream\_info() 19](#_Toc504407240)

[2.5 avcodec\_alloc\_context3() 20](#_Toc504407241)

[2.6 avcodec\_parameters\_to\_context() 20](#_Toc504407242)

[2.7 avcodec\_find\_decoder() 21](#_Toc504407243)

[2.8 avcodec\_open2() 21](#_Toc504407244)

[2.9 avcodec\_send\_packet() 21](#_Toc504407245)

[2.10 avcodec\_receive\_frame() 21](#_Toc504407246)

[2.11 av\_read\_frame() 22](#_Toc504407247)

[2.12 avformat\_seek\_file() 23](#_Toc504407248)

[3调试 24](#_Toc504407249)

[3.1视频卡住 24](#_Toc504407250)

[3.2音视频不同步 24](#_Toc504407251)

[3.3视频平滑处理 25](#_Toc504407252)

[3.4库crash 25](#_Toc504407253)

## 1 MediaPlayer框架

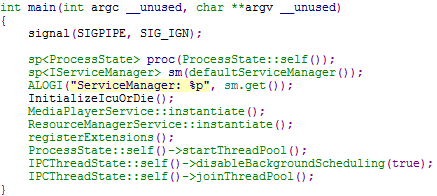
Android作为目前主流的终端系统，研究其上的多媒体播放器框架，是必不可少的。下述code的分析都是基于android 7.0版本。

## 1.1 mediaserver

所有关于多媒体的相关操作，作为一个native进程，独立在后台运行，见mediaserver进程。

位置：

C:\Users\renwei4\AppData\Local\Temp\1516080993(1).png

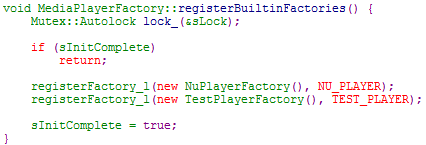


其中MediaPlayerService::instantiate();为注册播放器service，见C:\Users\renwei4\AppData\Local\Temp\1516081254(1).png。

## 1.2 MediaPlayerService

实例化接口中，通过MediaPlayerFactory注册android 支持的各种播放器，见MediaPlayerFactory::registerBuiltinFactories接口，

位置：C:\Users\renwei4\AppData\Local\Temp\1516082463(1).png



备注：android7.0后已经取消stageftight\_player，代码都在，可以参考上图添加。另外，如果想增加一个自定义的播放器，参考如上code，定义添加。比如：

合作方\*\*\*\*\*\*，在其中增加了一个名字为\*\*\*PLAYE的播放器（本质上还是nuplayer，只是植入了ffmpeg，独立出来了）。

重点阐述播放器的核心接口。

### 1.2.1 MediaPlayerService::create()

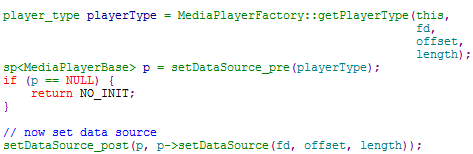
其核心

C:\Users\renwei4\AppData\Local\Temp\1516083370(1).png

为需要申请播放器的app，创建一个播放器实例。（不同的app，创建不同的播放器client，目的支持多播放，实际场景使用的不多）。

### 1.2.2 MediaPlayerService::Client::setDataSource()

该接口有多个，主要分本地和网络。核心



首先通过MediaPlayerFactory找到匹配的播放器类型，实例化一个具体的播放器，比如Nuplayer，最后给具体的播放器设置片源地址。上述为本地，网络的类似。

### 1.2.3 MediaPlayerService::Client::prepareAsync()

其核心

C:\Users\renwei4\AppData\Local\Temp\1516084055(1).png

具体播放器，比如Nuplayer进行一系列播放前的准备工作。

### 1.2.4 MediaPlayerService::Client::start()

其核心

C:\Users\renwei4\AppData\Local\Temp\1516084243(1).png

具体播放器，比如Nuplayer播放视频。

### 1.2.5 MediaPlayerService::Client::stop()

其核心

C:\Users\renwei4\AppData\Local\Temp\1516084339(1).png

具体播放器，比如Nuplayer停止播放。

### 1.2.6 MediaPlayerService::Client::pause()

其核心

C:\Users\renwei4\AppData\Local\Temp\1516084774(1).png

具体播放器，比如Nuplayer暂停播放。

### 1.2.7 MediaPlayerService::Client::reset()

其核心

C:\Users\renwei4\AppData\Local\Temp\1516084530(1).png

具体播放器，比如Nuplayer复位到初始状态。

### 1.2.8 MediaPlayerService::Client::seekTo(int msec)

其核心

C:\Users\renwei4\AppData\Local\Temp\1516084436(1).png

具体播放器，比如Nuplayer seek视频。

## 1.3 Nuplayer

Nuplayer与MediaPlayerService的桥梁是NuPlayerDriver。NuPlayerDriver实例化时Nuplayer也被实例化。对于播放器的操作也是通过NuPlayerDriver到达Nuplayer对应的接口。

### 1.3.1 NuPlayer::setDataSourceAsync()

该接口有多个，对应MediaPlayerService::Client::setDataSource()，核心：

根据片源地址，实例化source对象，保存片源地址url或fd，主要是对后续不同来源对象的操作，比如

本地: sp<GenericSource> source =new GenericSource(notify, mUIDValid, mUID);

网络：sp<Source> source = new HTTPLiveSource(notify, httpService, url, headers);

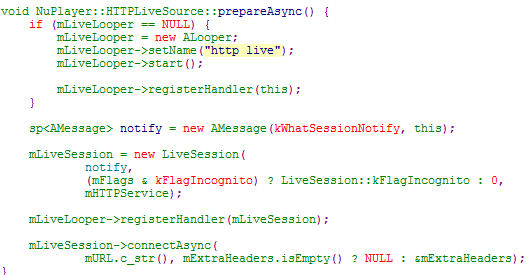
本地设计到本地文件的读取；网络涉及到http请求，数据download问题。

会通过kWhatSetDataSource消息，异步到内部线程处理，保存上述的source对象。

关键点，要留意source实例化中的notify，其为kWhatSourceNotify消息，这是后续source实例回传相关信息给Nuplayer的通路，要清晰。

### 1.3.2 NuPlayer::prepareAsync()

对应MediaPlayerService::Client::prepareAsync()，通过kWhatPrepare消息，异步到内部线程，在mSource->prepareAsync()接口处理，对应不同的source。以HTTPLiveSource为例，位置：C:\Users\renwei4\AppData\Local\Temp\1516086893(1).png。



核心：

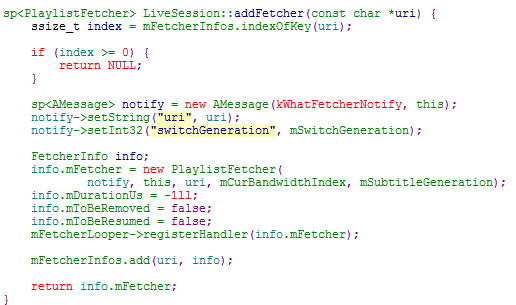
1）首先实例化LiveSession对象，该对象非常重要，管理video/audio/subtitle packet队列（AnotherPacketSource）。

位置：C:\Users\renwei4\AppData\Local\Temp\1516088560(1).png

关键点，要留意实例化中的notify，其为kWhatSessionNotify消息，这是后续LiveSession实例回传相关信息给HTTPLiveSource的通路，要清晰。

2）LiveSession对象connectAsync，通过kWhatConnect消息，异步到LiveSession::onConnect接口。核心：

2.1）在LiveSession::addFetcher()中实例化PlaylistFetcher对象



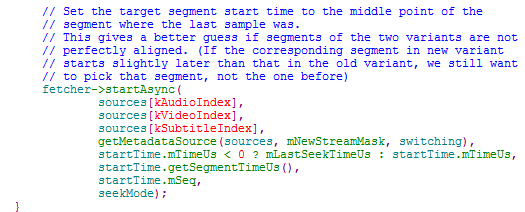
关键点，要留意实例化中的notify，其为kWhatFetcherNotify消息，这是后续PlaylistFetcher实例回传相关信息给LiveSession的通路，要清晰。

2.2）之后PlaylistFetcher对象调用fetchPlaylistAsync，通过kWhatFetchPlaylist消息，异步download数据流，比如m3u8文件和ts流。见HTTPDownloader::fetchPlaylist()，位置C:\Users\renwei4\AppData\Local\Temp\1516089499(1).png，核心

下载m3u8文件，ssize\_t err = fetchFile(url, &buffer, &actualUrl);

解析m3u8文件sp<M3UParser> playlist = new M3UParser(actualUrl.string(), buffer->data(), buffer->size());其中内部的parse()涉及HLS，m3u8文件格式，单独一章再描述。

2.3）解析完m3u8后，PlaylistFetcher通过kWhatFetcherNotify消息回传PlaylistFetched到LiveSession对象，在LiveSession::onMasterPlaylistFetched()接口，继续到LiveSession::changeConfiguration()接口，继续到LiveSession::onChangeConfiguration3()接口，下载ts流，如下PlaylistFetcher::startAsync()。这里的描述仅仅是核心流程。



上述接口，又会异步到PlaylistFetcher::onStart()接口，调用postMonitorQueue()启动kWhatMonitorQueue消息，周期的发送，决定是否下载ts流。

决策机制在PlaylistFetcher::onMonitorQueue()接口：

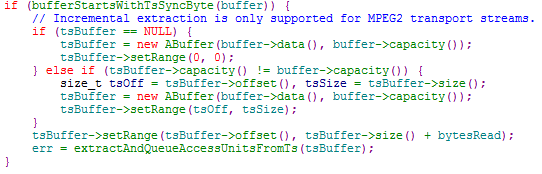
基于video/audio/subtitle anotherPacketSource，计算缓存可以播放时长的最小值；

如果该最小值小于初始定义的30000000ll（30秒，这里我们可以修改，优化体验），就发送kWhatDownloadNext消息，触发onDownloadNext()下载ts流（看到了吧，到这里，才是真正的下载ts流）：

C:\Users\renwei4\AppData\Local\Temp\1516092261(1).png

其中kDownloadBlockSize被定义为47k，目的是188的整数倍（256），一个ts包为188个字节。

const int32\_t PlaylistFetcher::kDownloadBlockSize = 47 \* 1024;

之后就要解析ts流，PlaylistFetcher::extractAndQueueAccessUnitsFromTs()中会初始化ATSParser实例，再循环188字节解析。具体的解析，另外一章描述。 

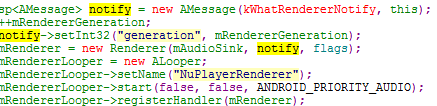
### 1.3.3 NuPlayer::start()

对应MediaPlayerService::Client::start()，通过kWhatStart消息，异步到内部线程，在NuPlayer::onStart(int64\_t startPositionUs)接口处理。

在NuPlayer::onStart中

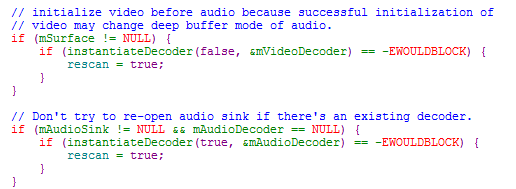
1）获取audio/video的 meta；

2）实例化Renderer对象，



关键点，要留意实例化中的notify，其为kWhatRendererNotify消息，这是后续Renderer实例回传相关信息给NuPlayer的通路，要清晰。

3）调用postScanSources()，通过kWhatScanSources消息，异步到内部线程，实例化audio和video解码器。一般video比audio先解码，所以先初始化video解码器。



instantiateDecoder()接口中，会实例化具体的audio/video解码器Decoder对象。详情如下继续描述。

3.1）通过source获取format，目的获取媒体基本信息，知道audio/video的格式等等，这个format很重要，后续使用。

sp<AMessage> format = mSource->getFormat(audio);

3.2）实例化解码器对象

sp<AMessage> notify = new AMessage(kWhatAudioNotify, this);

\*decoder = new Decoder(notify, mSource, mPID, mRenderer);

sp<AMessage> notify = new AMessage(kWhatVideoNotify, this);

\*decoder = new Decoder( notify, mSource, mPID, mRenderer, mSurface, mCCDecoder);

decoder 类型为基类NuPlayer::DecoderBase。

关键点，要留意实例化中的notify，其为kWhatAudioNotify或kWhatVideoNotify消息，这是后续Decoder实例回传相关信息给NuPlayer的通路，要清晰。

这里再继续描述一下实例化Decoder对象：NuPlayer::Decoder::Decoder()，

位置：C:\Users\renwei4\AppData\Local\Temp\1516094529(1).png

3.3）配置解码器

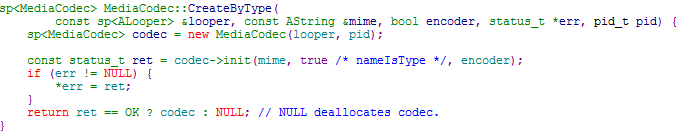
(\*decoder)->init(); 这个接口只是looper，没什么可说的。

(\*decoder)->configure(format); 关键处理在这里，通过kWhatConfigure消息异步到NuPlayer::Decoder::onConfigure()处理。

C:\Users\renwei4\AppData\Local\Temp\1516096082(1).png

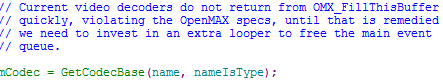
看到了吧，之前说很重要的format。根据format里面描述的audio/video encode类型，指定具体的解码器类型。

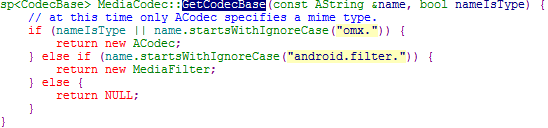
C:\Users\renwei4\AppData\Local\Temp\1516096008(1).png



CreateByType() 先实例化一个MediaCodec，再init。

MediaCodec::init()接口，





看到了，对于使用omx的解码器，使用的是Acodec，MediaCodec仅仅是桥梁而已。

C:\Users\renwei4\AppData\Local\Temp\1516097370(1).png

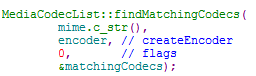
关键点，给Acodec设置kWhatCodecNotify消息，这是后续Acodec实例回传相关信息给MediaCodec的通路，要清晰。

之后通过kWhatInit消息，异步到

ACodec::initiateAllocateComponent(const sp<AMessage> &msg)，通过

kWhatAllocateComponent消息，再异步到

ACodec::UninitializedState::onAllocateComponent(const sp<AMessage> &msg)



其实就是通过解析/etc/media\_codecs.xml，匹配mime.c\_str()，得到对应的解码库。

C:\Users\renwei4\AppData\Local\Temp\1516168042(1).png

实例化CodecObserver，监听omx返回的消息，这个很重要，用于omx给Acodec发送消息。

C:\Users\renwei4\AppData\Local\Temp\1516168217(1).png

omx创建解码器。

最后C:\Users\renwei4\AppData\Local\Temp\1516169074(1).png切换Acodec状态。（Acodec有个很复杂的状态机）

回到NuPlayer::Decoder::onConfigure()接口，MediaCodec::CreateByType()结束后，调用MediaCodec的configure接口：

C:\Users\renwei4\AppData\Local\Temp\1516096577(1).png

MediaCodec::configure()通过kWhatConfigure消息，异步到ACodec::initiateConfigureComponent(const sp<AMessage> &msg)，再通过kWhatConfigureComponent消息，异步到ACodec::LoadedState::onConfigureComponent()处理，继续，真正的处理在这个接口

ACodec::configureCodec(); 这个接口必须关注，通过openmax设置解码器：解码/编码、一些参数（宽高…）

关键点，给MediaCodec设置kWhatCodecNotify消息，这是后续MediaCodec实例回传相关信息给NuPlayer::Decoder（NuplayerDecoder）的通路，要清晰。 见下面。是不是有点面熟（Acodec实例回传相关信息给MediaCodec的通路，也是kWhatCodecNotify消息，不要混淆了啊）。

C:\Users\renwei4\AppData\Local\Temp\1516170363(1).png

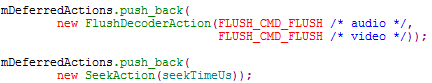
NuPlayer::Decoder::onConfigure()接口最后通过start运行解码器了。

到此Nuplayer开始播放了，后续就是解码器的相关动作。

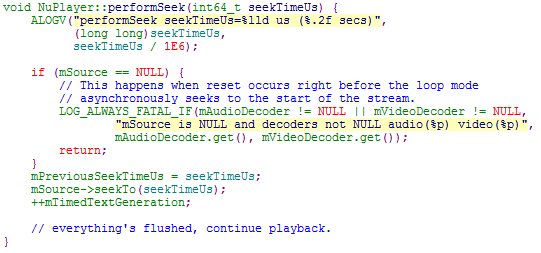
### 1.3.4 NuPlayer:: seekToAsync ()

对应MediaPlayerService::Client::seekTo ()，通过kWhatSeek消息，异步到内部线程，

首先清空audio/video解码器，然后再执行seek操作，如



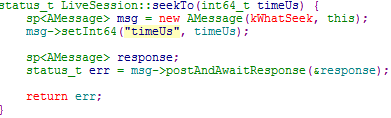
后续会到NuPlayer::performSeek(int64\_t seekTimeUs)接口。



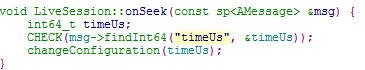
继续：依然以http m3u8文件例子。

C:\Users\renwei4\AppData\Local\Temp\1516265095(1).png

继续：

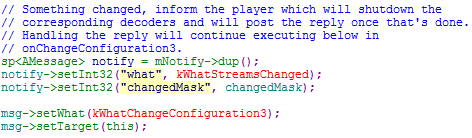


异步到LiveSession::onSeek()



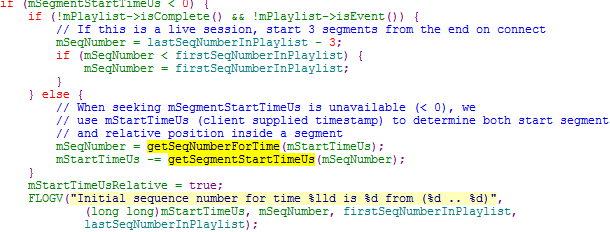
在LiveSession::changeConfiguration()接口中，对于seek，在LiveSession::onChangeConfiguration2接口中，

先清空所有AnotherPacketSource中的数据包，有可能码流切换，发生url改变，通过kWhatStreamsChanged消息，触发相应的解码器shutdown。



之后通过onChangeConfiguration3()，fetcher 数据。经过PlaylistFetcher::onStart()，到PlaylistFetcher::onDownloadNext()。

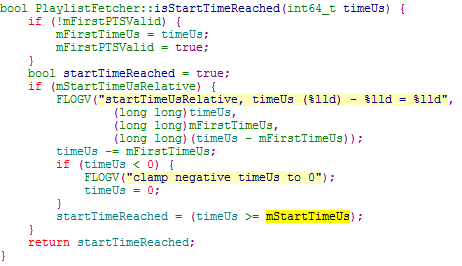
这里重点阐述一下，seek的时间戳问题。见PlaylistFetcher::initDownloadState()中



getSeqNumberForTime()用于计算需要seek的时间点所在的ts段。

之后，计算seek点，所在ts段的相对位置mStartTimeUs。有了这个相对位置后，在后续的ts解析接口PlaylistFetcher::extractAndQueueAccessUnitsFromTs()中，获取audio/video AnotherPacketSource队列中的数据包，计算其pts值是否大于等于mStartTimeUs。如果小于，则丢弃。如下：





Seek之后就是播放的流程了。见上述播放的分析。

## 1.4 解码器

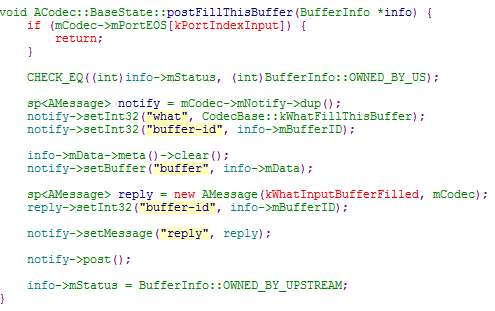
对于音视频播放，本章节只关注解码器，编码器不做讨论范围。Android7.0后不论是Nuplayer，还是3rd app通过MediaCodec接口，都是对接ACodec进行硬件 解码，核心就是关于数据流的读取与流向，需要重点关注Acodec里面的CodecObserver。这个是继承了BnOMXObserver类，已在上文解码器的configure过程中提到。对于播放，我们只需要知道关于omx\_message::EMPTY\_BUFFER\_DONE 和 omx\_message::FILL\_BUFFER\_DONE，openmax通知Acodec的2个消息处理。

### 1.4.1 EMPTY\_BUFFER\_DONE

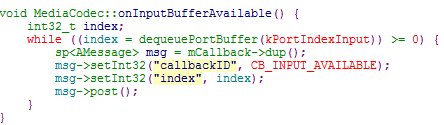
CodecObserver监听到omx\_message::EMPTY\_BUFFER\_DONE后，异步到onOMXEmptyBufferDone处理，如下，作用是解码器通知播放器，没有数据，需要给解码器送数据了。

ACodec::BaseState::onOMXEmptyBufferDone(IOMX::buffer\_id bufferID, int fenceFd)，其中

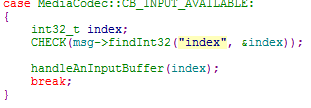
C:\Users\renwei4\AppData\Local\Temp\1516171470(1).png



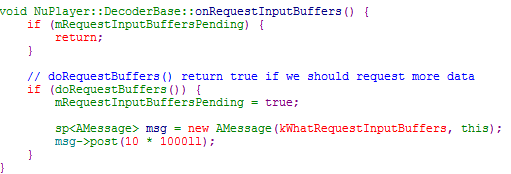
看到了吧，通过notify回传CodecBase::kWhatFillThisBuffer，notify，就是上文说的关键点，这里是Acodec回传kWhatCodecNotify消息给MediaCodec。核心是MediaCodec::onInputBufferAvailable()的处理，如下：

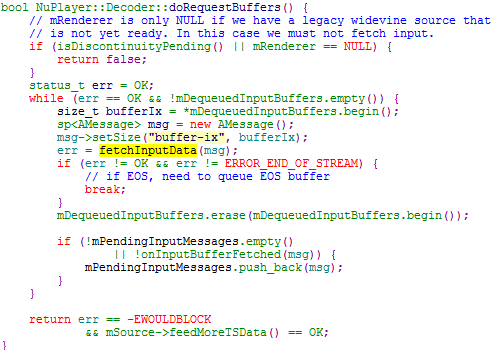


继续，MediaCodec通过kWhatCodecNotify消息回传到NuPlayer::Decoder（NuplayerDecoder），关键点啊，没忘记上文说的吧，处理如下：

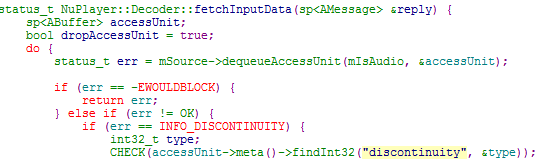


handleAnInputBuffer(size\_t index) 关于buffer的申请，复杂，我们只需要关注最后onRequestInputBuffers()接口，该接口会10毫秒周期触发doRequestBuffers()接口，进一步又调用fetchInputData()接口，这才是解码器真正去取原始码流数据的地方。





fetchInputData要点代码如下：

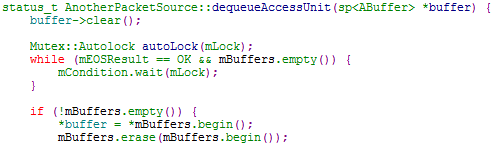


mSource->dequeueAccessUnit经历NuPlayer::HTTPLiveSource::dequeueAccessUnit—>

LiveSession::dequeueAccessUnit

C:\Users\renwei4\AppData\Local\Temp\1516174137(1).png

本质上就是从AnotherPacketSource队列上去数据包，如下



### 1.4.2 FILL\_BUFFER\_DONE

CodecObserver监听到omx\_message:: FILL\_BUFFER\_DONE后，异步到onOMXFillBufferDone处理，如下，作用是解码器通知播放器，已经解析出了清流，请播放器取出并显示。

ACodec::BaseState::onOMXFillBufferDone(

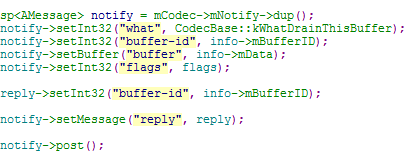
IOMX::buffer\_id bufferID,

size\_t rangeOffset, size\_t rangeLength,

OMX\_U32 flags,

int64\_t timeUs,

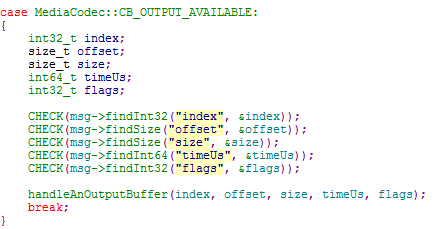
int fenceFd)，其中



通过notify回传CodecBase:: kWhatDrainThisBuffer，notify，就是上文说的关键点，这里是Acodec回传kWhatCodecNotify消息给MediaCodec。核心是MediaCodec:: onOutputBufferAvailable ()的处理，如下：

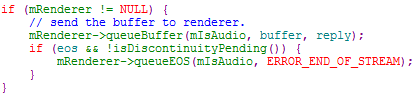


继续，MediaCodec通过kWhatCodecNotify消息回传到NuPlayer::Decoder（NuplayerDecoder），关键点啊，没忘记上文说的吧，处理如下：



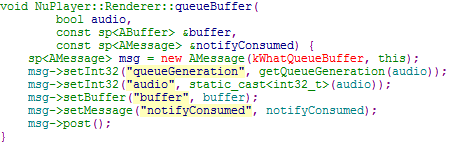
handleAnOutputBuffer()主要是读取解码出来的数据buffer，去render。核心截图：

C:\Users\renwei4\AppData\Local\Temp\1516175974(1).png

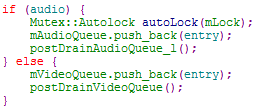


kWhatRenderBuffer消息用于NuplayerRender给NuplayerDecoder返回消息的通路，记住。

位置：C:\Users\renwei4\AppData\Local\Temp\1516176466(1).png

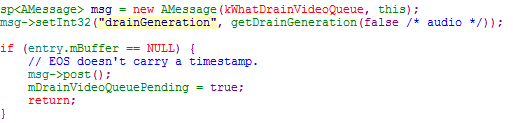


通过kWhatQueueBuffer消息异步到NuPlayer::Renderer::onQueueBuffer(const sp<AMessage> &msg)，其中postDrainAudioQueue\_l 和 postDrainVideoQueue获取audio和video数据。后续就是音视频同步了。



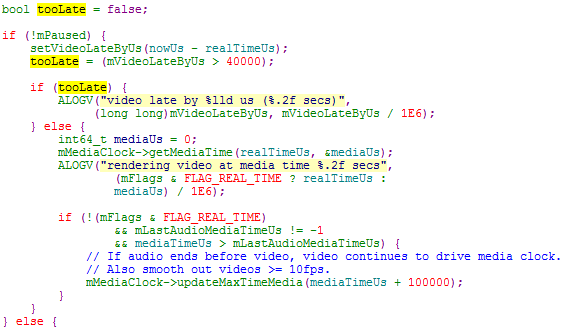
### 1.4.3 音视频同步

关于音视频同步，我重点描述上文Nuplayer render中postDrainVideoQueue()接口。

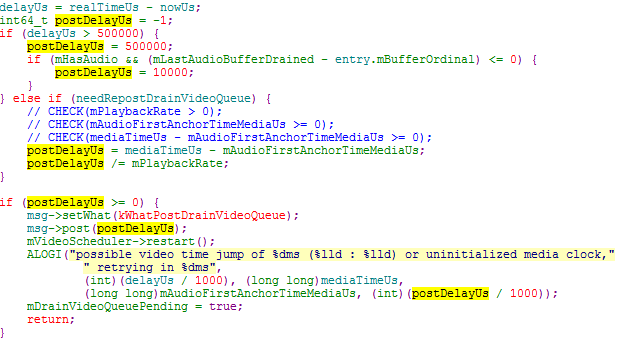


通过kWhatDrainVideoQueue消息异步到NuPlayer::Renderer::onDrainVideoQueue()接口取数据。

重点在这：



如果video的播放时间戳比当前系统时间晚了40毫秒，就丢弃该video帧。

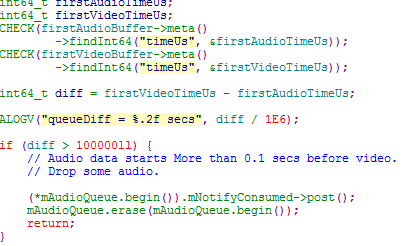


这里是video的播放时间戳比系统时间早了500毫秒以上，就暂时不再取帧，延迟处理，目的是等待系统时间赶上来。

一句话，Nuplayer关于AV同步：video早于audio，就等；video晚于audio，就丢弃video帧。

到这里，同步还没有完成，再回到NuPlayer::Renderer::onQueueBuffer()

通常video先被解码，所以需要比较audio和video首帧是否同步：



如上，如果首帧video比audio晚了1秒以上，就丢弃audio首帧数据。

提一下，video的渲染是通过VideoFrameScheduler完成的见VideoFrameScheduler::schedule(nsecs\_t renderTime)。

## 1.5 播放器组件交互图

上述从源码分析了各个接口及组件的交互和消息机制。基于此，我画一个Nuplayer组件交互图。

黑线表示组件之间的调用流程；红线表示组件返回的消息处理。由此需要熟悉红线的消息流程，这个也是对应上文描述的关键点，很多开发人员，对这个是很模糊的，不清晰。

Render

RenderNotify

SourceNotify SessionNotify

LiveSession

HttpLiveSource

Nuplayer

VideoNotify AudioNotify

FetchNotify

VideoDecoder

AudioDecoder

CodecNotify CodecNotify

ATSParser

PlayListFetcher

MediaCodec

CodecNotify

Acodec

M3uParser

omx\_message::EMPTY\_BUFFER\_DONE/

OMX

omx\_message:: FILL\_BUFFER\_DONE

## 2 FFmpeg

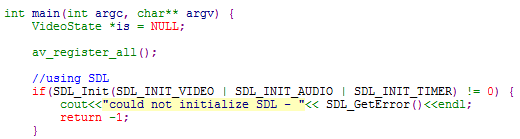
一套可以用来记录、转换数字音频、视频，并能将其转化为流的音视频开源内核，支持多平台，linux、ios、android等等。很多播放器，如ijkplayer、VLC，都封装了该库，可以说是ffmpeg的二次开发。熟悉其架构和开发流程，是必不可少的。它自带的ffplay，是一个很完善的应用播放器。

目前最新的版本是3.4。对于audio/video的输出/显示，需要使用SDL，其最新版本2.0。知道SDL就行，不需要重点研究。

核心接口如下。

## 2.1 av\_register\_all()

注册所有支持的mutex/demux continer格式、编解码器、文件访问io协议（如file、http）。这个接口需要被调用在最开始位置，如：



这里也配置使用了SDL，了解即可。

## 2.2 avformat\_alloc\_context()

分配AVFormatContext对象，是一个贯穿始终的数据结构，很多函数都要用到它作为参数。它是FFmpeg解封装（flv，mp4，rmvb，avi）功能的结构体。

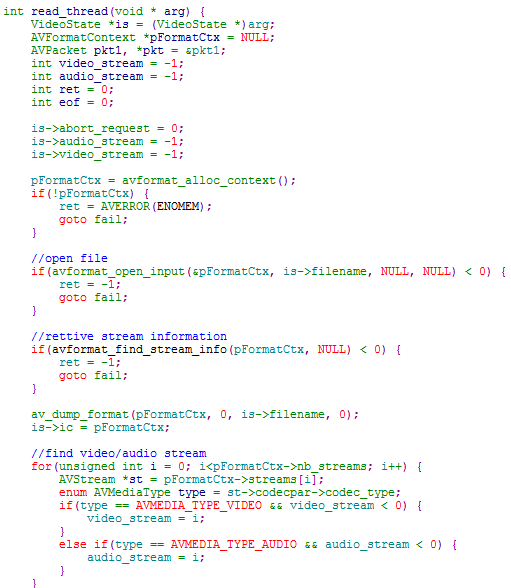
## 2.3 avformat\_open\_input()

该函数用于打开多媒体数据并且获得一些相关的信息。

## 2.4 avformat\_find\_stream\_info()

该函数可以读取一部分视音频数据并且获得一些相关的信息。Audio/video/subtitle信息分别由AVStream结构体管理。

针对这几个接口的使用方式，如下：



avformat\_find\_stream\_info()之后，我们就可以获取audio/video/subtitle的stream id，如上的video\_stream/audio\_stream变量。

## 2.5 avcodec\_alloc\_context3()

分配AVCodecContext，关于 编解码相关信息的结构体。

## 2.6 avcodec\_parameters\_to\_context()

该函数是通过Audio/video/subtitle相关信息，得到对应的AVCodecContext，是3.0版本上的新接口（相比之前，3.0改动比较大，开发接口变动也大）。

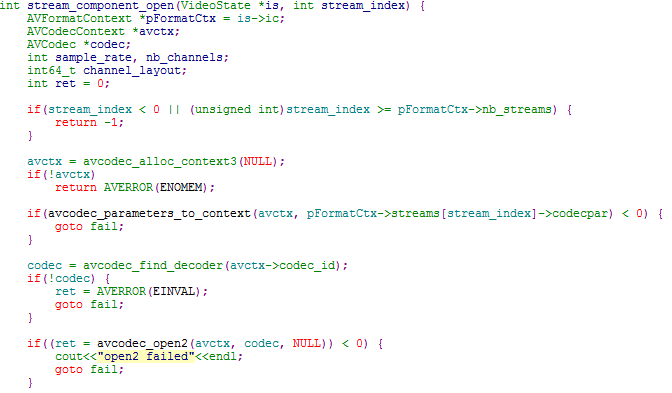
## 2.7 avcodec\_find\_decoder()

通过codec id获取对应的AVCodec。

## 2.8 avcodec\_open2()

打开编解码器。

针对这几个接口的使用方式，如下：

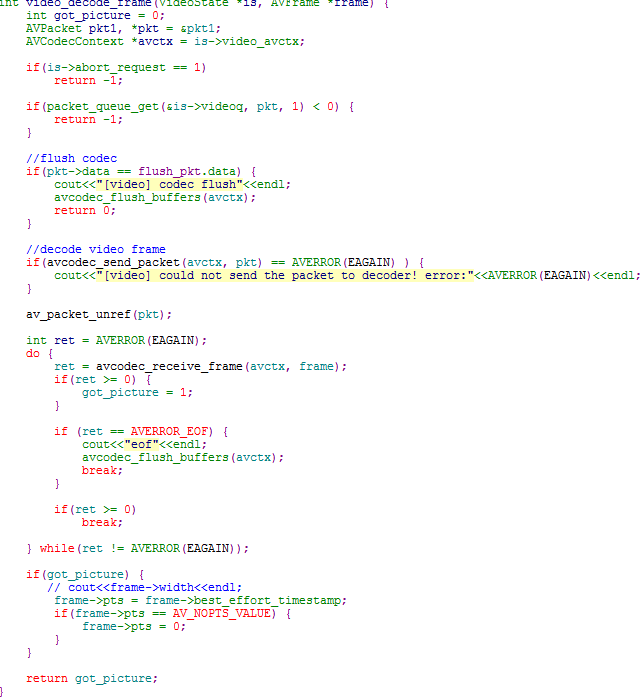


## 2.9 avcodec\_send\_packet()

向解码器发送数据，即packet。

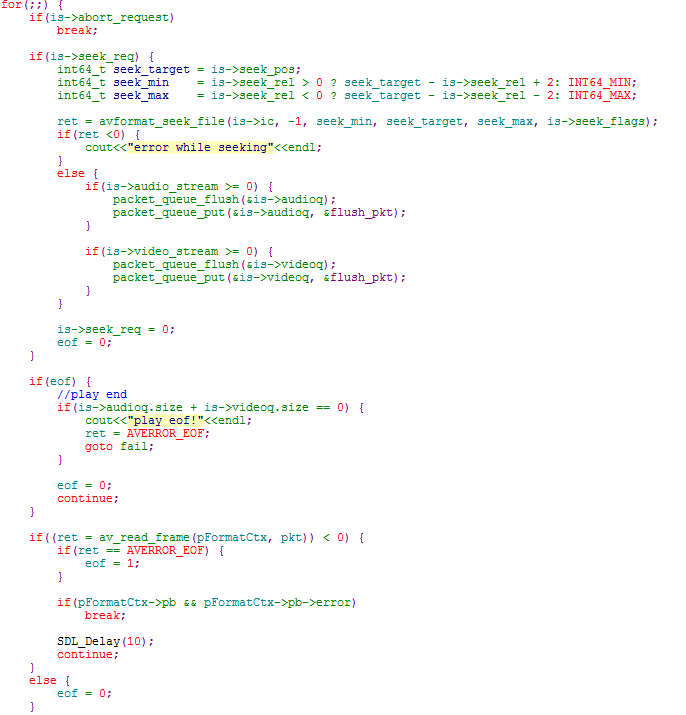
## 2.10 avcodec\_receive\_frame()

接受解码器解码出的清流，即frame。与avcodec\_send\_packet()成对使用。一般情况，对于video，一个packet，一个frame；而对于audio，一个packet，极大可能多个frame。参考如下代码：



## 2.11 av\_read\_frame()

读取packet，需要在一个无限循环中读取原始packet。如下代码：



## 2.12 avformat\_seek\_file()

seek操作，可以参考上述代码，已经给出使用方法。

以上列出的是必要的接口，必须明白。通过上述接口设计一个播放器，是有很多技巧的，多线程，多buffer一定要考虑的，比如，audio/video线程、audio/video packet buffer，audio/video frame buffer。

至于，如何考虑这些因素，去设计一个完善的播放器，可以参考我在github上分享的源码，与我技术交流，实现是比较复杂。限于篇幅，不在这里贴出代码。

https://github.com/hansonLGE/Player\_custom\_ffmpeg

## 3调试

不论是Android mediaplayer还是ffmpeg，都是独立的多媒体播放器框架，也可以结合使用。比如，\*\*\*\*\*\*的解决方案：

1 使用ffmpeg 做片源的demux，即使用AVFormat解析各个片源容器，分离audio、video。

2 使用 nuplayer的播放策略，进行解码并显示。

## 3.1视频卡住

实际场景，常常会遇到播放器卡住了，不播放了。

调试方法： 筛选log中getCurrentPosition，只要某个时间戳不变化了，就可知道被卡住的时间点，再排查前面的log信息，即可知道可能的原因：download不到数据或解码器不吐数据。

Line 5846: \_03\_09\_25\_19:01-03 09:24:06.635 I/NuPlayerDriver( 4153): [getCurrentPosition] position : 5739 msec

Line 5852: \_03\_09\_25\_19:01-03 09:24:06.774 I/NuPlayerDriver( 4153): [getCurrentPosition] position : 5739 msec

Line 5857: \_03\_09\_25\_19:01-03 09:24:06.774 I/NuPlayerDriver( 4153): [getCurrentPosition] position : 5739 msec

Line 5863: \_03\_09\_25\_19:01-03 09:24:06.914 I/NuPlayerDriver( 4153): [getCurrentPosition] position : 5739 msec

Line 5868: \_03\_09\_25\_19:01-03 09:24:06.915 I/NuPlayerDriver( 4153): [getCurrentPosition] position : 5739 msec

Line 5874: \_03\_09\_25\_19:01-03 09:24:07.056 I/NuPlayerDriver( 4153): [getCurrentPosition] position : 5739 msec

Line 5879: \_03\_09\_25\_19:01-03 09:24:07.056 I/NuPlayerDriver( 4153): [getCurrentPosition] position : 5739 msec

Line 5885: \_03\_09\_25\_19:01-03 09:24:07.195 I/NNuPlayerDriver( 4153): [getCurrentPosition] position : 5739 msec

Line 5890: \_03\_09\_25\_19:01-03 09:24:07.196 I/NNuPlayerDriver( 4153): [getCurrentPosition] position : 5739 msec

Line 5896: \_03\_09\_25\_19:01-03 09:24:07.337 I/NuPlayerDriver( 4153): [getCurrentPosition] position : 5739 msec

比如上述就是\*\*\*\*\*\*平台的一个video卡住不走的例子，被卡在了5739 msec。正常情况，这个值是一直递增的。

往前排查，发现存在

01-12 18:18:14.636 I/FFmpegExtractor(10823): No more packets from ffmpeg.

这就说明，ffmpeg读取不到数据，需要排查U盘自身问题，是否读写出了问题。

## 3.2音视频不同步

调试方法：筛选log中late by，即可知道video延迟了多少秒，被丢弃了。在这里，我们可以优化这个延迟区间大小。另外，常常会遇到系统性能问题，导致IO吞吐异常，给解码器送数据慢了，也会增加延迟丢帧，这种情况，就要具体分析是什么原因导致性能降低了。

比如，

video late by 539612 us

如果这个延迟很小，可以考虑放大阈值（nuplayer默认40毫秒，mtk平台放大到了400毫秒）；如果延迟很大1秒以上，就要排查解码器自身了，不是播放器问题了，设计到性能，送packet给解码器慢的问题了。

## 3.3视频平滑处理

这个问题主要是对于vod片源，涉及到m3u8文件。对于HLS协议，支持高低码流的切换，这个过程会发生DISCONTINUITY标记（也有可能m3u8文件主动插入该标记）。对于这个标记的处理，需要释放解码器，并重新实例化解码器。造成video画面顿一下，mtk平台更会闪黑屏。这个问题，我们可以尝试在

NuPlayer::Decoder::fetchInputData()中做video不释放解码器，规避处理。

## 3.4库crash

经常对遇到mediaserver crash问题。通常是其中的某个so库crash。这个问题需要定位带symbol的so文件。如下：

$ addr2line -e libmedia.so 00069106  
/\*\*\*\*\*\*/frameworks/av/media/libmedia/Visualizer.cpp:290

通过addr2line反查询crash的地址，可以知道对应的源代码crash问题，进一步分析，即可做相关规避处理。