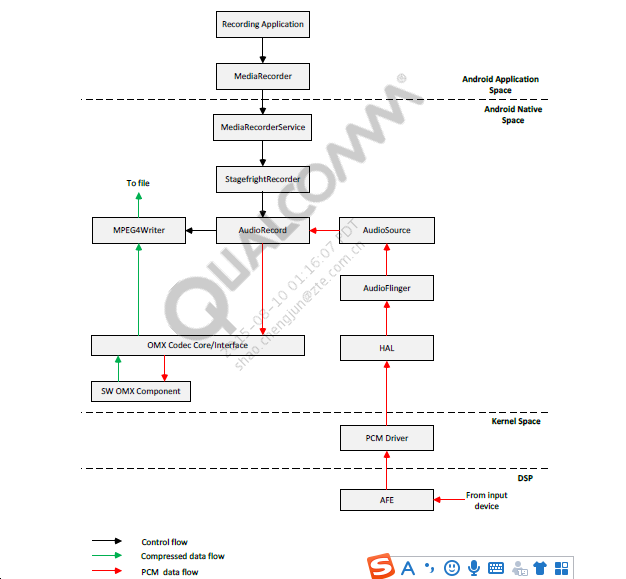


Reading:



用户空间log看：

In the Android Native layer, the recording flow is initiated via MediaPlayerService。The MediaRecorder component creates a StagefrightRecorder instance.

The following MediaRecorder APIs are called in sequence to initialize the recording session。接下来MediaRecorder相关API顺序的初始化录音一些资源：比如MIC,采样率，Output format，Encoder ，

12-13 15:48:55.642 215 1758 V MediaRecorderService: setAudioSource(1) 12-13 15:48:55.642 215 1758 V StagefrightRecorder: setAudioSource: 1

12-13 15:48:55.642 14853 14853 V MediaRecorder: setParameters(audio-param-sampling-rate=48000) 12-13 15:48:55.642 215 1120 V StagefrightRecorder: setParameters: audio-param-sampling-rate=48000。

之后MediaRecorder 调用prepare() command,录音开始。

StagefrightRecorder now creates AudioSource, AudioRecord (which internally creates AudioTrack)。之后audiorecorder打开输入设备：

AudioRecord then opens an input corresponding to the input device.

12-13 15:48:55.652 215 1758 V AudioPolicyManager: getDeviceForInputSource()input source 1, device 80000004 12-13 15:48:55.652 215 1758 V AudioPolicyManager: getInput() inputSource 1, samplingRate 48000, format 1, channelMask c, acoustics 0 12-13 15:48:55.652 215 1758 V AudioFlinger: openInput() openInputStream returned input 0xb8547028, SamplingRate 48000, Format 1, Channels c, status 0

12-13 15:48:55.652 215 1758 V AudioFlinger: openInput() created record thread: ID 22 thread 0xb8548a18。

在audiopoliccemanager():调用getDeviceForInputSource()input source。调用到audioflinger openInput() openInputStream打开输入流同时创建一个录音线程。

接下来The AAC encoder that is used to encode the input PCM stream is instantiated next; in this case, it is a software encoder.Input and output buffers are allocated for the encoder to use, after which it moves to an idle state. The encoder moves into the running state after some media configuration is performed.

之后encoder通知框架其初始化完成并启动录制过程。 这涉及启动AudioRecord和RecordTrack线程。

之后The input opened previously is now started which causes the audio path to be set up and calibration for the input device to be pushed.：

AudioPolicyManagerBase: startInput() input 22；AudioPolicyManager: getDeviceForInputSource()input source 1,；AudioPolicyManagerBase: AudioPolicyManager::startInput()；AudioFlinger: RecordThread: loop starting；AudioFlinger: Signal record thread；调用到hal层，msm8974\_platform: platform\_update\_usecase\_from\_source: input source :1

HAL层开始：

audio\_hw\_primary: start\_input\_stream: enter: usecase(6)；

msm8974\_platform: platform\_get\_input\_snd\_device: enter: out\_device(0) in\_device(0x4) 12-13 15:48:55.682 215 14937 V msm8974\_platform: platform\_get\_input\_snd\_device: exit: in\_snd\_device(handset-stereo-dmic-ef)

得到输入输出设备；

audio\_hw\_primary: enable\_snd\_device: snd\_device(65: handset-stereo-dmic-ef)使能设备；

card\_id (0) SOUND\_CARD and device\_id (0) is AUDIO\_RECORD\_PCM\_DEVICE defined in hardware/qcom/audio/hal/msm8974/platform.h.

12-13 15:48:55.692 215 14937 V audio\_hw\_primary: start\_input\_stream: Opening PCM device card\_id(0) device\_id(0), channels 2

The mic device has the ACDB ID 34 associated with it. Calibration data for the mic is sent to the DSP, as shown in the following log comment.设备根据usecase得到调用xml文件中控件通路，然后根据ID调用DSP算法拓扑；

输入PCM数据从AudioStreamInALSA读取，并通过AudioTrack传递给AudioSource。

The OMX layer reads this data and sends it to the encoder instance by calling emptyBuffer(). The encoder uses EmptyBufferDone() CB to inform the OMX core that it is done consuming the buffer. This is performed in a loop for all input buffers. The OMX layer then reads encoded data from the encoder instance by calling fillBuffer(). The encoder uses FillBufferDone() CB to inform the OMX core that the buffer is filled. This is performed in a loop for all input buffers.

OMX层读取此数据并通过调用emptyBuffer（）将其发送到编码器实例。 编码器使用EmptyBufferDone（）CB来通知OMX核心它已经消耗了缓冲区。 这是在所有输入缓冲区的循环中执行的。然后，OMX层通过调用fillBuffer（）从编码器实例读取编码数据。 编码器使用FillBufferDone（）CB通知OMX内核缓冲区已填满。 这是在所有输入缓冲区的循环中执行的。来自编码器的read（）调用返回，MPEG4Writer拉取编码数据。 它将数据打包成最后写入文件的块。 read（）调用在循环中完成，MPEG4Writer被阻塞，直到编码器提供更多数据。这一直持续到记录停止，这导致StagefrightRecorder，MPEG4Writer和编码器停止跟随其他组件。

Kernel log：

PCM path from BACKEND DAI ID 3 (MSM\_BACKEND\_DAI\_SLIMBUS\_0\_TX) to FRONTEND DAI ID 0 (MSM\_FRONTEND\_DAI\_MULTIMEDIA1) is set up first.

<7>[ 5734.833855] msm\_pcm\_routing\_process\_audio: reg 3 val 0 set 1 <7>[ 5734.834152] msm\_pcm\_hw\_params: perf: 0 <7>[ 5734.834158] msm\_pcm\_hw\_params Opening 2-ch PCM read stream

<7>[ 5734.834164] \_\_q6asm\_open\_read:session[1]

Recording is started by setting the hardware parameters and calling msm\_pcm\_capture\_prepare followed by the msm\_pcm\_trigger() function. PCM data is copied from DSP buffers.

<7>[ 5734.837409] msm\_pcm\_hw\_params: session ID 1 <7>[ 5734.837421] q6asm\_audio\_client\_buf\_alloc\_contiguous: session[1]bufsz[3840]bufcnt[2] <7>[ 5734.906823] msm\_pcm\_capture\_prepare <7>[ 5734.906830] Samp\_rate = 48000 <7>[ 5734.906834] Channel = 2

<7>[ 5734.910126] msm\_pcm\_trigger: Trigger start <7>[ 5734.910134] session[1]

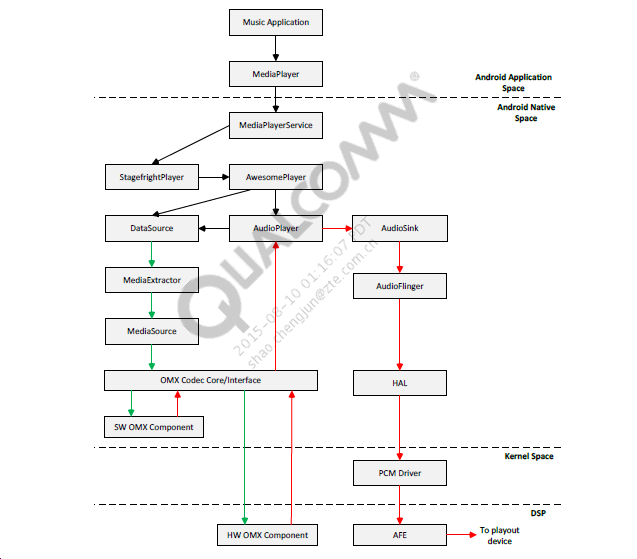
The pcm\_irq\_pos value increments by the size of a buffer each time a READ returns. 每次READ返回时，pcm\_irq\_pos值都会增加缓冲区的大小。

Capture gets stopped by sending SNDRV\_PCM\_TRIGGER\_STOP.

通过发送SNDRV\_PCM\_TRIGGER\_STOP来停止捕获。

DSP停止填充缓冲区并将它们返回给驱动程序（pcm\_ireq\_pos保持不变）。 驱动程序释放所有缓冲区。DSP buffers are released.

**Deep buffer playback:**



用户log：

Audio playback is initiated from the application layer through the MediaPlayer service. MediaPlayer in turn uses the Stagefright framework to fulfill the actual playout. StagefrightPlayer instantiates AwesomePlayer in the playback scenario

通过MediaPlayer服务从应用层启动音频播放。 MediaPlayer又使用Stagefright框架来完成实际的播出。 StagefrightPlayer在回放场景中实例化AwesomePlayer。

Once MediaPlayer, StagefrightPlayer, and AwesomePlayer instances are created, the input file data is extracted using an appropriate extractor and a MediaSource is created from it.

创建MediaPlayer，StagefrightPlayer和AwesomePlayer实例后，使用适当的提取器提取输入文件数据，并从中创建MediaSource。

12-12 17:01:50.667 6394 6394 V MediaPlayer: prepare。

The prepare() API is then invoked, which instantiates the decoder that is used to decode the compressed media. In this case, an MP3 file is to be played out so the software MP3 decoder provided by Google is instantiated. Input and output buffers for the decoder are also allocated as part of its creation and it is started.

然后调用prepare（）API，其实例化用于解码压缩媒体的解码器。 在这种情况下，将播放MP3文件，以便实例化由Google提供的软件MP3解码器。 解码器的输入和输出缓冲区也作为其创建的一部分进行分配并启动。

12-12 17:01:50.667 218 6483 V OMXCodec: [OMX.google.mp3.decoder] Now Executing.

12-12 17:01:50.667 6394 6406 V MediaPlayer: prepared

12-12 17:01:50.667 6394 6394 V MediaPlayer: prepare complete - status=0

The decoder moves into the executing state and waits for its buffers to be filled by the source. The prepare sequence is complete.

解码器进入执行状态并等待其缓冲区被源填充。 准备序列完成。

12-12 17:01:50.737 6394 6394 V MediaPlayer: start 12-12 17:01:50.737 218 218 V StagefrightPlayer: start。

Once initialized, the start () API is invoked, which kicks off media playback. StagefrightPlayer creates an instance of AudioPlayer and starts it to perform the actual playout of media.

初始化后，将调用start（）API，从而启动媒体播放。 StagefrightPlayer创建一个AudioPlayer实例并启动它以执行媒体的实际播出。

StagefrightPlayer through the AudioPlayer provides data to and reads data from the decoder buffers.

12-12 17:01:50.747 218 218 V AudioSink: open(44100, 2, 0x0, 0x1, 4, 37 0x8) 12-12 17:01:50.747 218 218 V AudioPolicyService: getOutput() 12-12 17:01:50.747 218 218 V legacy\_audio\_policy\_hal: audio\_io\_handle\_t android\_audio\_legacy::ap\_get\_output(audio\_policy\*, audio\_stream\_type\_t, uint32\_t, audio\_format\_t, audio\_channel\_mask\_t, audio\_output\_flags\_t, const audio\_offload\_info\_t\*): tid 218 12-12 17:01:50.747 218 218 V AudioPolicyManagerBase: getOutput() device 8, stream 3, samplingRate 0, format 0, channelMask 3, flags 0

12-12 17:01:50.747 218 218 V AudioSystem: getSamplingRate() streamType 3, output 2, sampling rate 48000

12-12 17:01:50.747 218 218 V AudioSink: creating new AudioTrack 12-12 17:01:50.747 218 218 V AudioTrack: sampleRate 44100, channelMask 0x3, format 1

12-12 17:01:50.747 218 218 V AudioTrack: createTrack\_l() output 2 afLatency 160

12-12 17:01:50.747 218 218 V AudioFlinger: createTrack() sessionId: 37

12-12 17:01:50.747 218 218 V AudioMixer: add track (1)

12-12 17:01:50.747 218 218 V AudioSink: setVolume

12-12 17:01:50.757 218 218 V AudioSink: open() DONE status 0

AudioTrack在接收参数创建的时候，就会将设置的steamtype保存在对应的AudioAttributes当中（AudioAttributes是一个描述关于音频流的信息的属性集合的类）。在android系统中，系统封装的对象是一层一层往下调用的。所以，在我们创建了一个java的AudioTrack对象的时候，其实在同时，在C++当中，我们已经做了很多操作了。在创建java层的AudioTrack对象时，对应的jni也创建出一个C++的AudioTrack对象，并且传入了部分参数和调用了其方法。AudioTrack的无参构造方法只是进行了一些参数的初始化，那么，具体是AudioTrack初始化是进行在哪里呢？发现jni层在创建完AudioTrack对象后，根据memoryMode的不同而进行了不同的AudioTrack->set()操作，在AudioTrack的set()中，除了部分的参数判断和设置之外，我们可以看到，他调用了自身的createTrack\_l()进行了进一步的设置。AudioTrack从这里开始，与AudioFlinger等进行大量的交互：获取句柄，获取输出，创建IAudioTrack指针对象等等。所以接下来，就是AudioFlinger的相关内容了。，AudioTrack在调用createTrack\_l()的方法的时候，开始通过AudioSystem获取output。所以下面我们来看看AudioSystem的getOutputForAttr().AudioSystem只是作为一个过渡，然后通过获取AudioPolicyService的句柄去getOutputForAttr()。我们继续跟踪AudioPolicyService的情况，会发现其实他只是在AudioPolicyService中也只是作为一个过渡，真正进行getOutputForAttr()的，在AudioPolicyManager之中. 在AudioPolicyManager的getOutputForAttr()中，我们可以发现关键点在strategy的获取与device的获取当中。

之前创建的AudioSink已打开，它实例化AudioTrack实例并将其添加到混音器。 音量和音效效果在AudioMixer中设置。

12-12 17:01:50.757 218 218 V AudioSink: start

12-12 17:01:50.757 218 218 V AudioFlinger: start(4097), calling pid 6394 session 37 12-12 17:01:50.757 218 218 V AudioFlinger: ? => ACTIVE (4097) on thread 0xb749b7b8

12-12 17:01:50.757 218 218 V AudioPolicyService: startOutput()

12-12 17:01:50.757 218 218 V AudioPolicyManagerBase: startOutput() output 2, stream 3, session 37

12-12 17:01:50.757 218 218 V AudioPolicyManagerBase: getNewDevice() selected device 8

12-12 17:01:50.757 218 218 V AudioPolicyManagerBase: setOutputDevice() output 2 device 0008 delayMs 0

12-12 17:01:50.757 218 218 V AudioPolicyManager: checkAndSetVolume: index 10 output 2 device 8 12-12 17:01:50.757 218 218 V AudioFlinger: signal playback thread 12-12 17:01:50.757 218 592 V AudioFlinger: anticipated start

Once the AudioSink is opened, it is started which initiates audio path setup to the actual device where playback is needed.

12-12 17:01:50.757 218 592 V audio\_hw\_primary: start\_output\_stream: enter: usecase(0: deep-buffer-playback) devices(0x8)

12-12 17:01:50.757 218 592 V msm8974\_platform: platform\_get\_output\_snd\_device: enter: output devices(0x8) 12-12 17:01:50.757 218 592 V msm8974\_platform: platform\_get\_output\_snd\_device: exit: snd\_device(headphones) 12-12 17:01:50.757 218 592 D audio\_hw\_primary: select\_devices: out\_snd\_device(4: headphones) in\_snd\_device(0: ) 12-12 17:01:50.757 218 592 D hardware\_info: hw\_info\_append\_hw\_type : device\_name = headphones 12-12 17:01:50.757 218 592 V audio\_hw\_primary: enable\_snd\_device: snd\_device(4: headphones) is already active 12-12 17:01:50.757 218 592 V audio\_hw\_primary: enable\_audio\_route: enter: usecase(0) 12-12 17:01:50.757 218 592 V audio\_hw\_primary: enable\_audio\_route: apply mixer path: deep-buffer-playback

The output path for playback is opened up through Audio HAL. Device is headphones defined in /hardware/qcom/audio/hal/msm8974/platform.c. Use case (0) is USECASE\_AUDIO\_ PLAYBACK\_DEEP\_BUFFER defined in hardware/qcom/audio/hal/audio\_hw.h.

此时，设置端到端路径并从设备播放音频。 AudioPlayer跟踪时间戳。 只要输入数据可用，解码器就继续解码输入媒体。

The audio file has reached its end and playback is complete. All components are stopped and output device routing is reset.