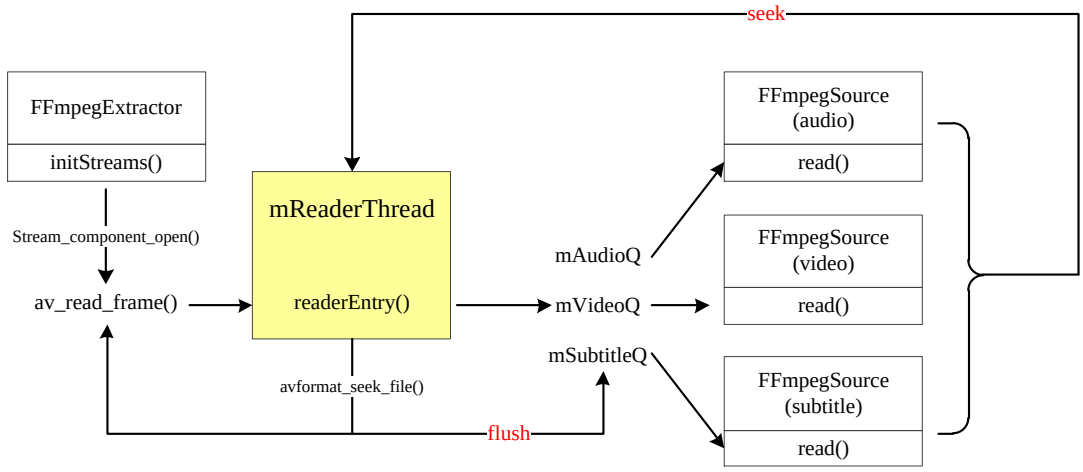
FFmpegExtractor 类分析



1. *initStreams()*

- 初始化 FFmpeg(ffmpeg_utils::initFFmpeg)，主要是注册编解码器、复用以及解复用器、android source，android source 是用来将 android framework 中的 datasource 和 FFmpeg 联系起来的，如 NuCachedSource2，这是 FFmpeg 的数据源。
- 通过 avformat_alloc_context()生成 AVFormatContext 对象(mFormatCtx)，然后通过 avformat_open_input()将其初始化。
- 通过 av_find_best_stream()方法找到音视频流和字幕流（优先选择码率最高，已解复用的帧最多的流）。
- 初始化 mVideoQ、mStaticAudioQ 和 mOutImgSubtitleQ。
- 通过 stream_component_open()打开视频流。
- 初始化一个 audio queue，并放入 mAudioQueueMap 中，索引为 step 3 中找到的音频流索引，然后打开该音频流。
- 初始化一个 subtitle queue，放入 mSubtitleQueueMap，同时分别在 mSubtitleTimeMap、mSubtitleStreamIdxList 和 mDoubleLanguageMap 中增加一个元素，索引为 step 3 中找到的字幕流索引，然后打开该字幕流。
- 用 for 循环打开剩余的所有流，如果是音频流，初始化一个 audio queue，并放入 mAudioQueueMap；如果是字幕流，初始化一个 subtitle queue，放入 mSubtitleQueueMap，同时分别在 mSubtitleTimeMap、mSubtitleStreamIdxList 和 mDoubleLanguageMap 中增加一个元素。
- 通过 parseOutBandSubtitle()处理所有外挂字幕

2. *stream_component_open(int streamIndex)*

- 根据 streamIndex 从 AVFormatContext 对象中获取 AVCodecContext 对象(formatCtx->streams[stream_index]->codec)，并将相应 stream 的 discard 属性设置为 AVDISCARD_DEFAULT。

- 从 AVCodecContext 对象中获取 codec 的类型，并根据 codec 类型做相应的初始化

操作：

Video: 初始化 mVideoStreamIdx、mVideoStream，生成一个 TrackInfo 对象，将该视频流的信息(stream_index、meta(setVideoFormat())、mVideoStream、mVideoQ)赋给 TrackInfo 对象，然后将这个 TrackInfo 对象放入 mTracks 容器里。

Audio: 初始化 mAudioStreamIdx、mAudioStream，生成一个 TrackInfo 对象，将该音频流的信息(stream_index、meta(setAudioFormat())、audioStream、mAudioQ)赋给 TrackInfo 对象，然后将这个 TrackInfo 对象放入 mTracks 容器里。

Subtitle: 初始化 mSubtitleStreamIdx、mSubtitleStream，生成一个 TrackInfo 对象，将该字幕流的信息(stream_index、meta(setSubtitleFormat())、subtitleStream、subtitleQueue)赋给 TrackInfo 对象，然后将这个 TrackInfo 对象放入 mTracks 容器里。

3. getTrackMetaData()

返回 mTracks 中某一元素的 meta，这个 meta 是在 stream_component_open()里创建的。

4. getMetaData()

遍历 mTracks，对每一个 track，调用 getTrackMetaData()，取得 mimeType，如果是音频，则会向 mMeta 里面添加一些信息，其他类型不做处理，然后返回 mMeta。

5. getTrack(size_t index)

根据 index 值返回一个 FFmpegSource 对象，该对象包括但不仅限于 trackInfo 里面的属性(mIndex、mStream、mQueue、mMeta)，对于 AVC 和 HEVC 还会设置 mIsAVC 和 mIsHEVC 的值。PS：如果存在外挂字幕，index 会先被转化一次。

6. changeStream(const sp<MetaData> &meta, AVMediaType type, int idx)

当切音轨或字幕时会调用到这个方法，其主要功能是开始新的流，通过更改 AVFormatContext 里面的 stream 里的 discard 属性来实现。

- 当 type 为 audio 时，将新流的 discard 属性设置为 AVDISCARD_DEFAULT，设置 mIsAudioChanged 和 mStreamChanged 为 true，更新 mAudioStreamIdx。
- 当 type 为 subtitle 时，将新流的 discard 属性设置为 AVDISCARD_DEFAULT，如果之前的字幕流为外挂字幕，则将之前字幕流的 diacard 属性设置为 AVDISCARD_ALL，最后将 mIsSubtitleChanged 和 mStreamChanged 设为 true，更新 mSubtitleStreamIdx。

7. stream_seek()

当上层应用读取数据时加入了 seek 的标志，FFmpegSource 会调用到这个方法。这个方法会判断 seek 操作是否有必要进行(切换音轨不会触发 seek 操作)，如果有必要，则设置 mSeekReq 为 true，这样 readerEntry()方法在读取源数据时会做一些处理(详见 readerEntry()方法介绍)，另外设定 seek 操作时的位置边界(mSeekMin 和 mSeekMax)，即 seek 完成后的位置要在这两个位置之间。这可能是 ffmpeg 内部 seek 操作时需要找到关键帧(或其它？)，所以即使我们指定了 mSeekPos，最后 seek 到的位置应该是 mSeekPos 附近的一个关键点。

8. FFmpegSource::start(MetaData *params)

调用 changeStream()来启动一个新的流。

9. FFmpegSource::getFormat()

与 getTrackMetaData()方法一样，返回 mTracks 中某一元素的 meta。

10. FFmpegSource::read(MediaBuffer **buffer, const ReadOptions *options)

解码器取数据会调用到这个方法，该方法返回的是解复用但未解码的数据，它从 FFmpegExtractor 的音频帧队列、视频帧队列和字幕帧队列中取出数据，放入一个 MediaBuffer 对象，然后将这个对象返回给解码器。

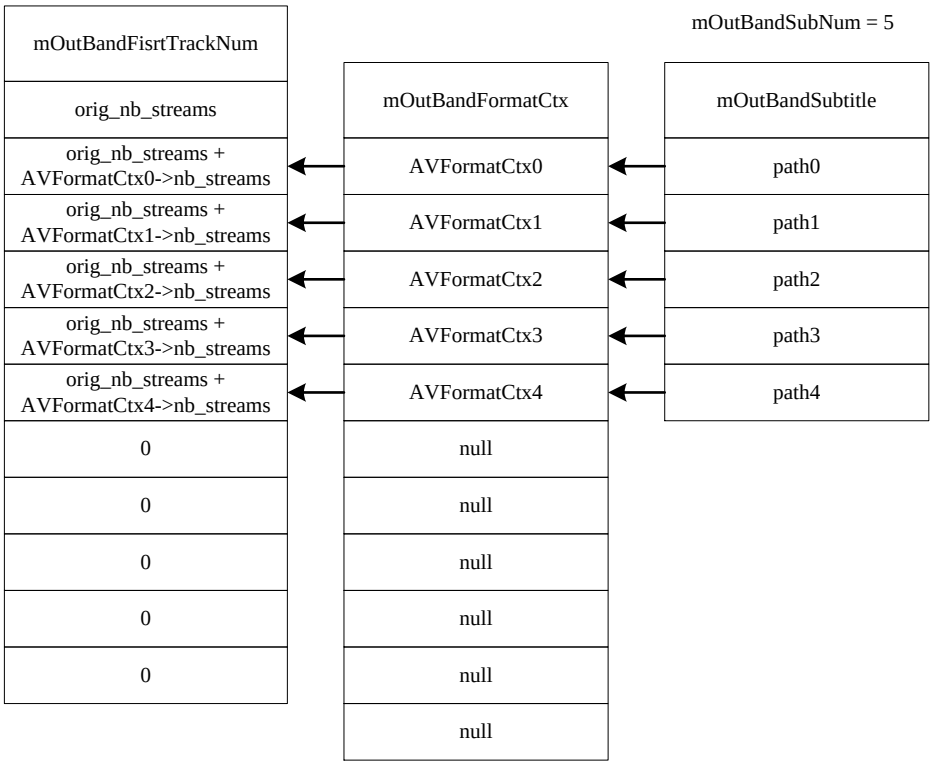
- 从 options 读取 seek 标志，如果存在，则通过 stream_seek 方法来初始化 seek 需要的环境，包括设置 mSeekReq、mSeekMin、mSeekMax、mSeekPos 等值。
- 根据当前的流类型从相应的帧队列中获取一个 AVPacket 对象，这个对象包含了指向原始数据块的指针。
- 音频和字幕可能存在多个队列，但只有当前队列，flush 掉左右非当前音频队列和字幕队列。
- 如果有 seek 操作正在处理，则丢弃所有出 flush packet 意外的所有其它 packet(readerEmtry()方法看到 mSeekReq 为 true 时会 flush 掉当前的音频帧队列、视频帧队列和字幕帧队列，然后 put 一个 flush packet，因此 flush packet 之后的 packet 才是 seek 之后的 packet)。
- 拿到了 flush packet 之后，设置 waitKeyPkt 为 true，意味着等待关键帧(I 帧)，这是为了避免 seek 后短暂的花屏、黑屏等现象。等待关键帧是有限制的，最多等待 iMaxDropNum(300)个 packet，否则强行结束等待。
- 更新 mVideoLastReadTime、mAudioLastReadTime 和 mSubtitleLastReadTime 的值、
- 生成一个 MediaBuffer 对象，根据 packet 里面的值来初始化 MediaBuffer，然后返回该对象。

11. readSubtitleSize(const char *path, int32_t *width, int32_t *height)

- 从字幕文件(path)中读取到 width 和 height 值。

12. parseOutBandSubtitle(bool needSelect)

- 根据 mOutBandSubNum，用 for 循环来依次处理每个字幕文件，mOutBandSubNum 是由外部设定的。
- 每处理一个字幕文件时，生成一个 AVFormatContext 对象，并放入 mOutBandFormatCtx 数组中。
- 通过 readSubtitleSize()方法读取字幕文件里的 width 和 height 值。
- 找到字幕文件中的字幕流数目，然后在 mOutBandFisrtTrackNum[]当前值得基础上加上当前字幕文件中的字幕流数目，作为 mOutBandFisrtTrackNum[]中的下一个值。
- 打开每一个字幕文件中的所有字幕流



13. *startReaderThread()*

- 创建线程 mReaderThread，启动 mReaderThread 来读取源数据，主要就是调用了 readerEntry()方法。

14. *readerEntry()*

readerEntry()里面包含了一个无限循环，其主要功能是读取源数据，放入视频帧队列、音频帧队列、字幕帧队列，然后发送队列更新信号，唤醒阻塞的队列读取线程，但是在此之前需要处理一些特殊情况，如 seek 操作、队列已满、音轨切换、eof 等。

- 如果有 seek 操作等待处理，通过 avformat_seek_file()方法 seek 到指定位置，flush 掉当前音频帧队列、视频帧队列和所有的字幕帧队列。
- 如果当前的音频帧队列、视频帧队列和字幕帧队列的总大小超过了规定值，或者当前的音频帧队列、视频帧队列和字幕帧队列都超过了各自的最小长度，线程睡眠 10ms，然后进入下一次循环继续读取。
- 如果到达了文件结尾，在当前音频帧队列、视频队列和所有的字幕帧队列加入一个空帧，然后进入下一次循环继续读取。PS：每两次循环遇到一个 eof，会不停地插入空帧，直到 NuPlayer 发现 eof 来主动停止？
- 如果以上 3 种情况都未满足，则通过 av_read_frame()读取一帧源数据，对读取到的数据做相应处理，然后放入相应队列，进入下一循环继续读取数据。

15. *findMatchingContainer()*

根据 AVFormatContext 里面的 AVInputFormat 里面的 name 来获取相应的 mimeType。

FORMAT	MIMETYPE
mpeg	video/mp2p
mpegts	video/mp2t
mov,mp4,m4a,3gp,3g2,mj2	video/mp4
matroska,webm	video/x-matroska
asf	video/x-ms-asf
rm	video/vnd.rn-realvideo
flv	video/x-flv
swf	video/x-flv
ape	audio/x-ape
flac	audio/flac
ac3	audio/ac3
wav	audio/x-wav
ogg	application/ogg
vc1	video/vc1
hevc	video/hevc
wmv	video/x-ms-wmv
mp3	audio/mpeg

16. adjustContainerIfNeeded()

如果 AVFormatContext 中只有 audio codec，没有 video codec，则根据 audio codec 的类型，将 mimeType 更换成相应的 mimeType。

17. adjustMPEG4Confidence()

18. adjustMPEG2TSConfidence()

19. adjustMKVConfidence()

20. adjustCodecConfidence()

21. adjustConfidenceIfNeeded()

通过判断 AVFormatContext 里面的 audio codec 和 video codec 是否符合 mimeType 的类型，来修改分数（confidence），修改的基准值是 mimeType 所指示的类型的标准值。adjustConfidenceIfNeeded()方法根据 mimeType 调用 adjustMPEG4Confidence()，adjustMPEG2TSConfidence()，adjustMKVConfidence()和 adjustCodecConfidence()来进行分数修改。

22. SniffFFMPEGCommon()

- 初始化 FFmpeg（initFFmpeg）
- 生成一个 AVFormatContext，并通过 avformat_open_input()方法初始化
- 通过 avformat_find_stream_info()方法找到流信息
- 通过 findMatchingContainer()找到 mimeType
- 通过 adjustContainerIfNeeded()和 adjustConfidenceIfNeeded()来调整 mimeType 和 confidence。

23. BetterSniffFFMPEG()

- 通过 SniffFFMPEGCommon()方法找到 mimeType
- 向 meta 里面添加”extended-extractor-url”字段，如：{“extended-extractor-url”, “android-source:0xf34321c0”}。

24. LegacySniffFFMPEG()

与 BetterSniffFFMPEG()相似，在 BetterSniffFFMPEG()方法失败的情况下，会重新调用 SniffFFMPEGCommon()来找 mimeType，不过传入的 url 会添加更多的信息，如从”android-source:0xf34321c0”变成” android-source: 0xf34321c0|file:xxx”。向 meta 里面添加的”extended-extractor-url”字段也变成了相应的内容。

25. SniffFFMPEG()

- 调用 BetterSniffFFMPEG()或者 LegacySniffFFMPEG()得到 mimeType 和 confidence。
- 再向 meta 里面添加三个字段：
  - { "extended-extractor", "extended-extractor" }
  - { "extended-extractor-subtype", "ffmpegextractor" }
  - { "extended-extractor-mime", mimeType}
- 如果 confidence 大于 0.08，则再向 meta 里面添加一个字段：
  { "extended-extractor-use", "ffmpegextractor" }