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

1. 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。

1. getTrackMetaData()

返回mTracks中某一元素的meta，这个meta是在stream\_component\_open()里创建的。

1. 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。

1. 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。

1. 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。

1. 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。

1. FFmpegSource::start(MetaData \*params)

调用changeStream()来启动一个新的流。

1. FFmpegSource::getFormat()

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

1. 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。

1. getLastReadTime()
2. setLastReadTime()

这里的read是指FFmpegSource::read()，set是指每当FFmpegSource读取到一个packet，就要用这个packet里面的pts时间来更新mVideoLastReadTime、mAudioLastReadTime或mSubtitleLastReadTime；而get则是返回这三个时间。

1. dropOtherAudioData(int streamIdx, int64\_t readTimeUs)

删除所有非当前队列的音频帧队列里面所有pts小于当前pts的packet，其结果是：当前音频帧队列里面包含了所有的packet，而其它音频帧队列只包含当前时间之后的packet。这个方法在FFmpegSource::read()方法中调用，即没获取一次数据就会调用一次。

1. dropOtherSubtitleData(int streamIdx )

与dropOtherAudioData()相似，删除所有非当前队列的字幕帧队列里面所有pts小于当前音频时间的packet。

1. readSubtitleSize(const char \*path, int32\_t \*width, int32\_t \*height)

* 从字幕文件(path)中读取到width和height值。

1. 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



1. getActualFormatContext(int index)

该方法从mOutBandFormatCtx数组中找到当前流的AVFormatContext，只对外挂字幕流有作用。

1. getActualID(int index)

该方法返回当前流在其自己所属的AVFormatContext中的index，只对外挂字幕流有作用。

1. 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。

1. 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。

1. findMatchingContainer()

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



1. 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。

1. adjustMPEG4Confidence()
2. adjustMPEG2TSConfidence()
3. adjustMKVConfidence()
4. adjustCodecConfidence()
5. 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。

1. 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。

1. 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”}。

1. 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。

1. 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" }