[Android audio and video development and upgrade: audio and video hard decoding] 4. Audio and video unpacking and packaging: generate an MP4

Table of contents

In this article you can learn

> This article mainly explains the decapsulation and encapsulation process of audio and video, but does not involve the encoding and decoding of audio and video, but the complete process of encoding and decoding of audio and video, which will be explained in the next chapter after OpenGL. Mainly to have a basic understanding of the repackage of audio and video.

1. Unblocking audio and video

In the second article of this chapter [audio and video hard decoding process], it has been said that Android uses MediaExtractor to decapsulate audio and video data streams. Here, we simply go through it again.

The first step is to initialize the MediaExtractor

```
init {
    mExtractor = MediaExtractor()
    mExtractor?.setDataSource(path)
}
```

The second step, get the format of the audio or video

```
/**
 * 获取视频格式参数
 */
fun getVideoFormat(): MediaFormat? {
    for (i in 0 until mExtractor!!.trackCount) {
        val mediaFormat = mExtractor!!.getTrackFormat(i)
        val mime = mediaFormat.getString(MediaFormat.KEY_MIME)
        if (mime.startsWith("video/")) {
            mVideoTrack = i
            break
        }
    }
    return if (mVideoTrack >= 0)
        mExtractor!!.getTrackFormat(mVideoTrack)
    else null
}

/**
 * 获取音频格式参数
 */
fun getAudioFormat(): MediaFormat? {
    for (i in 0 until mExtractor!!.trackCount) {
        val mediaFormat = mExtractor!!.getTrackFormat(i)
        val mime = mediaFormat.getString(MediaFormat.KEY_MIME)
        if (mime.startsWith("audio/")) {
            mAudioTrack = i
            break
        }
    }
    return if (mAudioTrack >= 0) {
        mExtractor!!.getTrackFormat(mAudioTrack)
    } else null
}
```

The third step, read (separate) audio and video data

```
/**
 * 读取音视频数据
 */
fun readBuffer(byteBuffer: ByteBuffer): Int {
    byteBuffer.clear()
    selectSourceTrack()
    var readSampleCount = mExtractor!!.readSampleData(byteBuffer, 0)
    if (readSampleCount < 0) {
        return -1
    }
    //记录当前帧的时间戳
    mCurSampleTime = mExtractor!!.sampleTime
    //进入下一帧
    mExtractor!!.advance()
    return readSampleCount
}

/**
 * 选择通道
 */
private fun selectSourceTrack() {
    if (mVideoTrack >= 0) {
        mExtractor!!.selectTrack(mVideoTrack)
    } else if (mAudioTrack >= 0) {
        mExtractor!!.selectTrack(mAudioTrack)
    }
}
```

2. Audio and video packaging

Android natively provides an encapsulator, MediaMuxer, which is used to encapsulate the encoded audio and video stream data into files of a specified format. MediaMuxer supports three encapsulation formats: MP4, Webm, and 3GP. Generally use MP4 format.

It is also relatively simple to use, and is also divided into several steps:

The first step, initialization

```
class MMuxer {
    private val TAG = "MMuxer"

    private var mPath: String

    private var mMediaMuxer: MediaMuxer? = null

    private var mVideoTrackIndex = -1
    private var mAudioTrackIndex = -1

    private var mIsAudioTrackAdd = false
    private var mIsVideoTrackAdd = false

    private var mIsStart = false

    init {
        val fileName = "LVideo_" + SimpleDateFormat("yyyyMM_dd-
HHmmss").format(Date()) + ".mp4"
        val filePath =
Environment.getExternalStorageDirectory().absolutePath.toString() + "/"
        mPath = filePath + fileName
        mMediaMuxer = MediaMuxer(mPath,
MediaMuxer.OutputFormat.MUXER_OUTPUT_MPEG_4)
    }

    //......
}
```

The video and save path and format are specified here.

> The second step is to add audio and video tracks, set the audio and video data stream format, and start the encapsulator

```
class MMuxer {

    //......

    fun addVideoTrack(mediaFormat: MediaFormat) {
        if (mMediaMuxer != null) {
            mVideoTrackIndex = try {
                mMediaMuxer!!.addTrack(mediaFormat)
            } catch (e: Exception) {
                e.printStackTrace()
                return
            }
            mIsVideoTrackAdd = true
            startMuxer()
        }
    }

    fun addAudioTrack(mediaFormat: MediaFormat) {
        if (mMediaMuxer != null) {
            mAudioTrackIndex = try {
                mMediaMuxer!!.addTrack(mediaFormat)
            } catch (e: Exception) {
                e.printStackTrace()
                return
            }
            mIsAudioTrackAdd = true
            startMuxer()
        }
    }

    /**
     *忽略音频轨道
     */
    fun setNoAudio() {
        if (mIsAudioTrackAdd) return
        mIsAudioTrackAdd = true
        startMuxer()
    }

    /**
     *忽略视频轨道
     */
    fun setNoVideo() {
        if (mIsVideoTrackAdd) return
        mIsVideoTrackAdd = true
        startMuxer()
    }

    private fun startMuxer() {
        if (mIsAudioTrackAdd && mIsVideoTrackAdd) {
            mMediaMuxer?.start()
            mIsStart = true
            Log.i(TAG, "启动混合器，等待数据输入...")
        }
    }
```

```
    //......
}
```

Before opening the encapsulator, you first need to set the data format corresponding to the audio and video. This format comes from the MediaFormat obtained from the decapsulation of the audio and video, namely

MMExtractor#getVideoFormat()

MMExtractor#getAudioFormat()

After passing mMediaMuxer!!.addTrack(mediaFormat), the track index corresponding to the audio and video data will be returned, which is used to write the data to the correct data track when encapsulating the data.

Finally, determine whether the audio and video tracks have been configured, and start the wrapper.

> The third step, writing data, is also very simple, just write the data obtained by decapsulation.

```
class MMuexer {

    //......

    fun writeVideoData(byteBuffer: ByteBuffer, bufferInfo: MediaCodec.BufferInfo)
{
        if (mIsStart) {
            mMediaMuxer?.writeSampleData(mVideoTrackIndex, byteBuffer, bufferInfo)
        }
    }

    fun writeAudioData(byteBuffer: ByteBuffer, bufferInfo: MediaCodec.BufferInfo)
{
        if (mIsStart) {
            mMediaMuxer?.writeSampleData(mAudioTrackIndex, byteBuffer, bufferInfo)
        }
    }

    //......
}
```

> The fourth step, release the encapsulator to complete the encapsulation process

==This step is very important, it must be released before it can generate a usable complete MP4 file==

```
class MMuxer {

    //......

    fun release() {
        mIsAudioTrackAdd = false
        mIsVideoTrackAdd = false
        try {
            mMediaMuxer?.stop()
            mMediaMuxer?.release()
            mMediaMuxer = null
            Log.i(TAG, "混合器退出...")
        } catch (e: Exception) {
            e.printStackTrace()
        }
    }

    //......
}
```

3. Integrate the unpacking and encapsulation process

Through the above two steps, the most basic tool packaging has been completed, and then you only need to integrate them.

Create a new repackaging class MP4Repack

```
class MP4Repack(path: String) {

    private val TAG = "MP4Repack"

    //初始化音视频分离器
    private val mAExtractor: AudioExtractor = AudioExtractor(path)
    private val mVExtractor: VideoExtractor = VideoExtractor(path)

    //初始化封装器
    private val mMuxer: MMuxer = MMuxer()

    /**
     *启动重封装
     */
    fun start() {
        val audioFormat = mAExtractor.getFormat()
        val videoFormat = mVExtractor.getFormat()

        //判断是否有音频数据，没有音频数据则告诉封装器，忽略音频轨道
        if (audioFormat != null) {
            mMuxer.addAudioTrack(audioFormat)
        } else {
            mMuxer.setNoAudio()
        }
        //判断是否有视频数据，没有音频数据则告诉封装器，忽略视频轨道
        if (videoFormat != null) {
            mMuxer.addVideoTrack(videoFormat)
        } else {
            mMuxer.setNoVideo()
        }

        //启动线程
        Thread {
            val buffer = ByteBuffer.allocate(500 * 1024)
            val bufferInfo = MediaCodec.BufferInfo()

            //音频数据分离和写入
            if (audioFormat != null) {
                var size = mAExtractor.readBuffer(buffer)
                while (size > 0) {
                    bufferInfo.set(0, size, mAExtractor.getCurrentTimestamp(),
                                   mAExtractor.getSampleFlag())

                    mMuxer.writeAudioData(buffer, bufferInfo)

                    size = mAExtractor.readBuffer(buffer)
                }
            }

            //视频数据分离和写入
            if (videoFormat != null) {
                var size = mVExtractor.readBuffer(buffer)
                while (size > 0) {
                    bufferInfo.set(0, size, mVExtractor.getCurrentTimestamp(),
                                   mVExtractor.getSampleFlag())
```

```
                    mMuxer.writeVideoData(buffer, bufferInfo)

                    size = mVExtractor.readBuffer(buffer)
                }
            }
            mAExtractor.stop()
            mVExtractor.stop()
            mMuxer.release()
            Log.i(TAG, "MP4 重打包完成")
        }.start()
    }
}
```

First , audio and video splitters, and wrappers are defined;

Next , determine whether the video to be repackaged contains audio and video data, and if not, ignore the corresponding track;

Finally , start the thread and start unpacking and encapsulation, which is divided into two parts:

1. Audio data separation and writing
2. Video data separation and writing

One of the things to pay attention to is the parameter of BufferInfo

```
val bufferInfo = MediaCodec.BufferInfo()

bufferInfo.set(0, size, mVExtractor.getCurrentTimestamp(),
                mVExtractor.getSampleFlag())
```

> The first is offset, usually 0,
> the second is the data size, which is the data size of the current frame extracted by Extractor, and the
> third is the timestamp corresponding to the current frame. This timestamp is very important and affects whether the video can be played normally or not. Obtained through Extractor The
> fourth is the current frame type, such as video I/P/B frames, which can also be obtained through Extractor

Fourth, call the MediaRepack repackage tool to achieve repackage

The call is very simple, as follows:

```
private fun repack() {
    val path = Environment.getExternalStorageDirectory().absolutePath +
"/mvtest_2.mp4"

    val repack = MP4Repack(path)
    repack.start()
}
```

At this point, this chapter [Audio and Video Hard Decoding Articles] series of articles is over. There are four articles in this series, from [Introduction to Basic Knowledge of Audio and Video] -> [Android Audio Decoding Process] -> [Audio and Video Playback and Synchronization] ->[Video unpacking and encapsulation], a more comprehensive introduction to the hard decoding capabilities provided by the Android application system to achieve audio and video decoding.

Next, I will enter a series of articles on OpenGL rendering, which will further introduce audio and video rendering, re-encoding, encapsulation, etc., so stay tuned.