【Android 音视频开发打怪升级：FFmpeg音视频编解码篇】六、FFmpeg简单合成MP4：视屏解封与重新封装 - 简书

In this article you can learn

> Use FFmpeg to simply decapsulate and repackage audio and video, without decoding and encoding, to pave the way for the next article to explain how to re-encode and package edited videos.

I. Introduction

In the previous article, the decoding of FFmpg video and how to use OpenGL to edit and render the video have been explained in detail. The most important thing is to encode and save the edited video.

Of course, before understanding how to encode, you must first understand how to encapsulate the encoded audio and video, which will have a multiplier effect.

In " Audio and Video Decapsulation and Encapsulation: Generate an MP4 ", the native function of Android is used to realize the repackaging of audio and video. FFmpeg is the same, but the process is more cumbersome.

2. Initialize the package parameters

We know that to encapsulate encoded data into Mp4, we need to know the parameters related to audio and video encoding, such as encoding format, video width and height, number of audio channels, frame rate, bit rate, etc. Let's take a look at how to initialize them first.

First, define a packer `FFRepacker`:

```
// ff_repack.h

class FFRepack {
private:
    const char *TAG = "FFRepack";

    AVFormatContext *m_in_format_cxt;

    AVFormatContext *m_out_format_cxt;

    int OpenSrcFile(char *srcPath);

    int InitMuxerParams(char *destPath);

public:
    FFRepack(JNIEnv *env,jstring in_path, jstring out_path);
};
```

The initialization process is divided into two steps: opening the original video file and initializing the packaging parameters.

```
// ff_repack.cpp

FFRepack::FFRepack(JNIEnv *env, jstring in_path, jstring out_path) {
    const char *srcPath = env->GetStringUTFChars(in_path, NULL);
    const char *destPath = env->GetStringUTFChars(out_path, NULL);

    // 打开原视频文件，并获取相关参数
    if (OpenSrcFile(srcPath) >= 0) {
        // 初始化打包参数
        if (InitMuxerParams(destPath)) {
            LOGE(TAG, "Init muxer params fail")
        }
    } else {
        LOGE(TAG, "Open src file fail")
    }
}
```

Open the original video and get the original video parameters

The code is very simple and has been explained in the article using FFMpeg decoding. as follows:

```
// ff_repack.cpp

int FFRepack::OpenSrcFile(const char *srcPath) {
    // 打开文件
    if ((avformat_open_input(&m_in_format_cxt, srcPath, 0, 0)) < 0) {
        LOGE(TAG, "Fail to open input file")
        return -1;
    }

    // 获取音视频参数
    if ((avformat_find_stream_info(m_in_format_cxt, 0)) < 0) {
        LOGE(TAG, "Fail to retrieve input stream information")
        return -1;
    }

    return 0;
}
```

Initialize packing parameters

Initializing the packaging parameters is a little more complicated. The main process is:

Find which audio and video streams are in the original video, and add the corresponding stream channel and initialize the corresponding encoding parameters for the target video (that is, the repackaged video file).

Next, open the target storage file using the initialized context.

Finally, write the video header information to the target file.

The code is as follows, please see the comments for the main process:

```
//ff_repack.cpp

int FFRepack::InitMuxerParams(const char *destPath) {
    // 初始化输出上下文
    if (avformat_alloc_output_context2(&m_out_format_cxt, NULL, NULL, destPath) < 0)
{
        return -1;
    }

    // 查找原视频所有媒体流
    for (int i = 0; i < m_in_format_cxt->nb_streams; ++i) {
        // 获取媒体流
        AVStream *in_stream = m_in_format_cxt->streams[i];

        // 为目标文件创建输出流
        AVStream *out_stream = avformat_new_stream(m_out_format_cxt, NULL);
        if (!out_stream) {
            LOGE(TAG, "Fail to allocate output stream")
            return -1;
        }

        // 复制原视频数据流参数到目标输出流
        if (avcodec_parameters_copy(out_stream->codecpar, in_stream->codecpar) < 0) {
            LOGE(TAG, "Fail to copy input context to output stream")
            return -1;
        }
    }

    // 打开目标文件
    if (avio_open(&m_out_format_cxt->pb, destPath, AVIO_FLAG_WRITE) < 0) {
        LOGE(TAG, "Could not open output file %s ", destPath);
        return -1;
    }

    // 写入文件头信息
    if (avformat_write_header(m_out_format_cxt, NULL) < 0) {
        LOGE(TAG, "Error occurred when opening output file");
        return -1;
    } else {
        LOGE(TAG, "Write file header success");
    }

    return 0;
}
```

Above, the initialization has been completed, and the audio and video can be unpacked and re-packed below.

3. Decapsulation of the original video

Added a `Start` method to enable repackaging

```
// ff_repack.h

class FFRepack {
    // 省略其他...

public:
    // 省略其他...

    void Start();
};
```

The specific implementation is as follows:

```
// ff_repack.cpp

void FFRepack::Start() {
    LOGE(TAG, "Start repacking ....")
    AVPacket pkt;
    while (1) {
        // 读取数据
        if (av_read_frame(m_in_format_cxt, &pkt)) {
            LOGE(TAG, "End of video，write trailer")

            // 释放数据帧
            av_packet_unref(&pkt);

            // 读取完毕，写入结尾信息
            av_write_trailer(m_out_format_cxt);

            break;
        }

        // 写入一帧数据
        Write(pkt);
    }

    // 释放资源
    Release();
}
```

Unpacking is still very simple. It was also introduced in the previous decoding article, mainly to read the data `AVPacket` into . Then call the `Write` method to write the frame data to the target file. Let's take a look at the `Write` method .

Fourth, the target video package

Add a `Write` method .

```
// ff_repack.h

class FFRepack {
    // 省略其他...

public:
    // 省略其他...

    void Write(AVPacket pkt);
};

// ff_repacker.cpp

void FFRepack::Write(AVPacket pkt) {

    // 获取数据对应的输入/输出流
    AVStream *in_stream = m_in_format_cxt->streams[pkt.stream_index];
    AVStream *out_stream = m_out_format_cxt->streams[pkt.stream_index];

    // 转换时间基对应的 PTS/DTS
    int rounding = (AV_ROUND_NEAR_INF | AV_ROUND_PASS_MINMAX);
    pkt.pts = av_rescale_q_rnd(pkt.pts, in_stream->time_base, out_stream->time_base,
                               (AVRounding)rounding);
    pkt.dts = av_rescale_q_rnd(pkt.dts, in_stream->time_base, out_stream->time_base,
                               (AVRounding)rounding);

    pkt.duration = av_rescale_q(pkt.duration, in_stream->time_base, out_stream-
>time_base);

    pkt.pos = -1;

    // 将数据写入目标文件
    if (av_interleaved_write_frame(m_out_format_cxt, &pkt) < 0) {
        LOGE(TAG, "Error to muxing packet: %x", ret)
    }
}
```

The process is very simple. After converting the `pts` sum `dts`, `duration` the data can be written.

Before writing data, the stream where the frame data is located and the data stream written are obtained. This is because, before writing, the time of the data needs to be converted.

Time units in FFmpeg

We know that each frame of audio and video data has its corresponding time stamp, and the control of audio and video playback can be realized according to this time stamp.

`FFmpeg` The timestamp in is not our actual time, it is a special value. And `FFmpeg` in , there is `时间基` a concept called , which is `时间基` the time unit in . `FFmpeg`

> [The value of the timestamp] 乘以 [time base] is the [actual time], and the unit is seconds.

In other words, the `FFmpeg` value of `时间基` the varies with .

`FFmpeg` There are also different time bases in different stages and different encapsulation formats. Therefore, when encapsulating frame data, it is necessary to perform "timestamp" conversion according to their respective time bases to ensure that the final calculated actual time is consistent.

Of course, in order to facilitate the conversion `FFmpeg` , it provides us with a conversion method, and handles data overflow and rounding problems.

```
av_rescale_q_rnd(int64_t a, AVRational bq,AVRational cq,enum AVRounding rnd)
```

Its internals are simple: return (a × bq / cq).

which is:

```
x（目标时间戳值） * cq（目标时间基）= a（原时间戳值） * bq（原时间基）

=》=》=》=》=》=》

x = a * bq / cq
```

When all data frames are read, the end information needs to be `av_write_trailer` written , so that the file is considered complete and the video can be played normally.

5. Release resources

Finally, the previously opened resources need to be closed to avoid memory leaks.

Add a `Release` method :

```
// ff_repack.h

class FFRepack {
    // 省略其他...

public:
    // 省略其他...

    void Release();
};
```

```
//ff_repack.cpp

void FFRepack::Release() {
    LOGE(TAG, "Finish repacking, release resources")
    // 关闭输入
    if (m_in_format_cxt) {
        avformat_close_input(&m_in_format_cxt);
    }

    // 关闭输出
    if (m_out_format_cxt) {
        avio_close(m_out_format_cxt->pb);
        avformat_free_context(m_out_format_cxt);
    }
}
```

Six, call repackaging

Added JNI interface

`native-lib.cpp` In , add `JNI` interface

```
// native-lib.cpp

extern "C" {

    // 省略其他 ...

    JNIEXPORT jint JNICALL
    Java_com_cxp_learningvideo_FFRepackActivity_createRepack(JNIEnv *env,
                                                             jobject  /* this */,
                                                             jstring srcPath,
                                                             jstring destPath) {
        FFRepack *repack = new FFRepack(env, srcPath, destPath);
        return (jint) repack;
    }

    JNIEXPORT void JNICALL
    Java_com_cxp_learningvideo_FFRepackActivity_startRepack(JNIEnv *env,
                                                            jobject  /* this */,
                                                            jint repack) {
        FFRepack *ffRepack = (FFRepack *) repack;
        ffRepack->Start();
    }
}
```

add page

```
// FFRepackActivity.kt

class FFRepackActivity: AppCompatActivity() {

    private var ffRepack: Int = 0

    private val srcPath = Environment.getExternalStorageDirectory().absolutePath +
"/mvtest.mp4"
    private val destPath = Environment.getExternalStorageDirectory().absolutePath +
"/mvtest_repack.mp4"

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_ff_repack)

        ffRepack = createRepack(srcPath, destPath)
    }

    fun onStartClick(view: View) {
        if (ffRepack != 0) {
            thread {
                startRepack(ffRepack)
            }
        }
    }

    private external fun createRepack(srcPath: String, destPath: String): Int

    private external fun startRepack(repack: Int)

    companion object {
        init {
            System.loadLibrary("native-lib")
        }
    }
}
```

The above is the process of using FFmpeg to unpack and encapsulate, which is relatively simple, mainly to prepare for subsequent video editing, encoding, and encapsulation.