原 最简单的基于FFmpeg的封装格式处理：视音频复用器（muxer）

2014年10月09日 00:47:17　阅读数：57477

==========

最简单的基于FFmpeg的封装格式处理系列文章列表：

最简单的基于FFmpeg的封装格式处理：视音频分离器简化版（demuxer-simple）

最简单的基于FFmpeg的封装格式处理：视音频分离器（demuxer）

最简单的基于FFmpeg的封装格式处理：视音频复用器（muxer）

最简单的基于FFMPEG的封装格式处理：封装格式转换（remuxer）

==========

简介

打算记录一下基于FFmpeg的封装格式处理方面的例子。包括了视音频分离，复用，封装格式转换。这是第3篇。

本文记录一个基于FFmpeg的视音频复用器(Simplest FFmpeg muxer)。视音频复用器(Muxer)即是将视频压缩数据(例如H.264)和音频压缩数据（例如AAC）合并到一个封装格式数据（例如MKV）中去。如图所示。在这个过程中并不涉及到编码和解码。

本文记录的程序将一个H.264编码的视频码流文件和一个MP3编码的音频码流文件，合成为一个MP4封装格式的文件。

流程

程序的流程如下图所示。从流程图中可以看出，一共初始化了3个AVFormatContext，其中2个用于输入，1个用于输出。3个AVFormatContext初始化之后，通过avcodec_copy_context()函数可以将输入视频/音频的参数拷贝至输出视频/音频的AVCodecContext结构体。然后分别调用视频输入流和音频输入流的av_read_frame()，从视频输入流中取出视频的AVPacket，音频输入流中取出音频的AVPacket，分别将取出的AVPacket写入到输出文件中即可。其间用到了一个不太常见的函数av_compare_ts()，是比较时间戳用的。通过该函数可以决定该写入视频还是音频。

单击查看更清晰的图片

本文介绍的视音频复用器，输入的视频不一定是H.264裸流文件，音频也不一定是纯音频文件。可以选择两个封装过的视音频文件作为输入。程序会从视频输入文件中“挑”出视频流，音频输入文件中“挑”出音频流，再将“挑选”出来的视音频流复用起来。
PS1：对于某些封装格式（例如MP4/FLV/MKV等）中的H.264，需要用到名称为“h264_mp4toannexb”的bitstream filter。
PS2：对于某些封装格式（例如MP4/FLV/MKV等）中的AAC，需要用到名称为“aac_adtstoasc”的bitstream filter。

简单介绍一下流程中各个重要函数的意义：

avformat_open_input()：打开输入文件。
avcodec_copy_context()：赋值AVCodecContext的参数。
avformat_alloc_output_context2()：初始化输出文件。
avio_open()：打开输出文件。
avformat_write_header()：写入文件头。
av_compare_ts()：比较时间戳，决定写入视频还是写入音频。这个函数相对要少见一些。
av_read_frame()：从输入文件读取一个AVPacket。
av_interleaved_write_frame()：写入一个AVPacket到输出文件。
av_write_trailer()：写入文件尾。

代码

下面贴上代码：

```cpp
/**
 * 最简单的基于FFmpeg的视音频复用器
 * Simplest FFmpeg Muxer
 *
 * 雷霄骅 Lei Xiaohua
 * leixiaohua1020@126.com
```

```
 * 中国传媒大学/数字电视技术
 * Communication University of China / Digital TV Technology
 * http://blog.csdn.net/leixiaohua1020
 *
 * 本程序可以将视频码流和音频码流打包到一种封装格式中。
 * 程序中将AAC编码的音频码流和H.264编码的视频码流打包成
 * MPEG2TS封装格式的文件。
 * 需要注意的是本程序并不改变视音频的编码格式。
 *
 * This software mux a video bitstream and a audio bitstream
 * together into a file.
 * In this example, it mux a H.264 bitstream (in MPEG2TS) and
 * a AAC bitstream file together into MP4 format file.
 *
 */

#include <stdio.h>

#define __STDC_CONSTANT_MACROS

#ifdef _WIN32
//Windows
extern "C"
{
#include "libavformat/avformat.h"
};
#else
//Linux...
#ifdef __cplusplus
extern "C"
{
#endif
#include <libavformat/avformat.h>
#ifdef __cplusplus
};
#endif
#endif

/*
FIX: H.264 in some container format (FLV, MP4, MKV etc.) need
"h264_mp4toannexb" bitstream filter (BSF)
  *Add SPS,PPS in front of IDR frame
  *Add start code ("0,0,0,1") in front of NALU
H.264 in some container (MPEG2TS) don't need this BSF.
*/
//'1': Use H.264 Bitstream Filter
#define USE_H264BSF 0

/*
FIX:AAC in some container format (FLV, MP4, MKV etc.) need
"aac_adtstoasc" bitstream filter (BSF)
*/
//'1': Use AAC Bitstream Filter
#define USE_AACBSF 0



int main(int argc, char* argv[])
{
    AVOutputFormat *ofmt = NULL;
    //Input AVFormatContext and Output AVFormatContext
    AVFormatContext *ifmt_ctx_v = NULL, *ifmt_ctx_a = NULL,*ofmt_ctx = NULL;
    AVPacket pkt;
    int ret, i;
    int videoindex_v=-1,videoindex_out=-1;
    int audioindex_a=-1,audioindex_out=-1;
    int frame_index=0;
    int64_t cur_pts_v=0,cur_pts_a=0;

    //const char *in_filename_v = "cuc_ieschool.ts";//Input file URL
    const char *in_filename_v = "cuc_ieschool.h264";
    //const char *in_filename_a = "cuc_ieschool.mp3";
    //const char *in_filename_a = "gowest.m4a";
    //const char *in_filename_a = "gowest.aac";
    const char *in_filename_a = "huoyuanjia.mp3";

    const char *out_filename = "cuc_ieschool.mp4";//Output file URL
    av_register_all();
    //Input
    if ((ret = avformat_open_input(&ifmt_ctx_v, in_filename_v, 0, 0)) < 0) {
        printf( "Could not open input file.");
        goto end;
    }
    if ((ret = avformat_find_stream_info(ifmt_ctx_v, 0)) < 0) {
        printf( "Failed to retrieve input stream information");
        goto end;
    }

    if ((ret = avformat_open_input(&ifmt_ctx_a, in_filename_a, 0, 0)) < 0) {
        printf( "Could not open input file.");
        goto end;
    }
```

```
    }
    if ((ret = avformat_find_stream_info(ifmt_ctx_a, 0)) < 0) {
        printf( "Failed to retrieve input stream information");
        goto end;
    }
    printf("===========Input Information==========\n");
    av_dump_format(ifmt_ctx_v, 0, in_filename_v, 0);
    av_dump_format(ifmt_ctx_a, 0, in_filename_a, 0);
    printf("======================================\n");
    //Output
    avformat_alloc_output_context2(&ofmt_ctx, NULL, NULL, out_filename);
    if (!ofmt_ctx) {
        printf( "Could not create output context\n");
        ret = AVERROR_UNKNOWN;
        goto end;
    }
    ofmt = ofmt_ctx->oformat;

    for (i = 0; i < ifmt_ctx_v->nb_streams; i++) {
        //Create output AVStream according to input AVStream
        if(ifmt_ctx_v->streams[i]->codec->codec_type==AVMEDIA_TYPE_VIDEO){
        AVStream *in_stream = ifmt_ctx_v->streams[i];
        AVStream *out_stream = avformat_new_stream(ofmt_ctx, in_stream->codec->codec);
        videoindex_v=i;
        if (!out_stream) {
            printf( "Failed allocating output stream\n");
            ret = AVERROR_UNKNOWN;
            goto end;
        }
        videoindex_out=out_stream->index;
        //Copy the settings of AVCodecContext
        if (avcodec_copy_context(out_stream->codec, in_stream->codec) < 0) {
            printf( "Failed to copy context from input to output stream codec context\n");
            goto end;
        }
        out_stream->codec->codec_tag = 0;
        if (ofmt_ctx->oformat->flags & AVFMT_GLOBALHEADER)
            out_stream->codec->flags |= CODEC_FLAG_GLOBAL_HEADER;
        break;
        }
    }

    for (i = 0; i < ifmt_ctx_a->nb_streams; i++) {
        //Create output AVStream according to input AVStream
        if(ifmt_ctx_a->streams[i]->codec->codec_type==AVMEDIA_TYPE_AUDIO){
            AVStream *in_stream = ifmt_ctx_a->streams[i];
            AVStream *out_stream = avformat_new_stream(ofmt_ctx, in_stream->codec->codec);
            audioindex_a=i;
            if (!out_stream) {
                printf( "Failed allocating output stream\n");
                ret = AVERROR_UNKNOWN;
                goto end;
            }
            audioindex_out=out_stream->index;
            //Copy the settings of AVCodecContext
            if (avcodec_copy_context(out_stream->codec, in_stream->codec) < 0) {
                printf( "Failed to copy context from input to output stream codec context\n");
                goto end;
            }
            out_stream->codec->codec_tag = 0;
            if (ofmt_ctx->oformat->flags & AVFMT_GLOBALHEADER)
                out_stream->codec->flags |= CODEC_FLAG_GLOBAL_HEADER;

            break;
        }
    }

    printf("==========Output Information==========\n");
    av_dump_format(ofmt_ctx, 0, out_filename, 1);
    printf("======================================\n");
    //Open output file
    if (!(ofmt->flags & AVFMT_NOFILE)) {
        if (avio_open(&ofmt_ctx->pb, out_filename, AVIO_FLAG_WRITE) < 0) {
            printf( "Could not open output file '%s'", out_filename);
            goto end;
        }
    }
    //Write file header
    if (avformat_write_header(ofmt_ctx, NULL) < 0) {
        printf( "Error occurred when opening output file\n");
        goto end;
    }


    //FIX
#if USE_H264BSF
    AVBitStreamFilterContext* h264bsfc =  av_bitstream_filter_init("h264_mp4toannexb");
#endif
#if USE_AACBSF
    AVBitStreamFilterContext* aacbsfc =  av_bitstream_filter_init("aac_adtstoasc");
#endif
```

```
    while (1) {
        AVFormatContext *ifmt_ctx;
        int stream_index=0;
        AVStream *in_stream, *out_stream;

        //Get an AVPacket
        if(av_compare_ts(cur_pts_v,ifmt_ctx_v->streams[videoindex_v]->time_base,cur_pts_a,ifmt_ctx_a->streams[audioindex_a]->time_base) <= 0){
            ifmt_ctx=ifmt_ctx_v;
            stream_index=videoindex_out;

            if(av_read_frame(ifmt_ctx, &pkt) >= 0){
                do{
                    in_stream  = ifmt_ctx->streams[pkt.stream_index];
                    out_stream = ofmt_ctx->streams[stream_index];

                    if(pkt.stream_index==videoindex_v){
                        //FIX：No PTS (Example: Raw H.264)
                        //Simple Write PTS
                        if(pkt.pts==AV_NOPTS_VALUE){
                            //Write PTS
                            AVRational time_base1=in_stream->time_base;
                            //Duration between 2 frames (us)
                            int64_t calc_duration=(double)AV_TIME_BASE/av_q2d(in_stream->r_frame_rate);
                            //Parameters
                            pkt.pts=(double)(frame_index*calc_duration)/(double)(av_q2d(time_base1)*AV_TIME_BASE);
                            pkt.dts=pkt.pts;
                            pkt.duration=(double)calc_duration/(double)(av_q2d(time_base1)*AV_TIME_BASE);
                            frame_index++;
                        }

                        cur_pts_v=pkt.pts;
                        break;
                    }
                }while(av_read_frame(ifmt_ctx, &pkt) >= 0);
            }else{
                break;
            }
        }else{
            ifmt_ctx=ifmt_ctx_a;
            stream_index=audioindex_out;
            if(av_read_frame(ifmt_ctx, &pkt) >= 0){
                do{
                    in_stream  = ifmt_ctx->streams[pkt.stream_index];
                    out_stream = ofmt_ctx->streams[stream_index];

                    if(pkt.stream_index==audioindex_a){

                        //FIX：No PTS
                        //Simple Write PTS
                        if(pkt.pts==AV_NOPTS_VALUE){
                            //Write PTS
                            AVRational time_base1=in_stream->time_base;
                            //Duration between 2 frames (us)
                            int64_t calc_duration=(double)AV_TIME_BASE/av_q2d(in_stream->r_frame_rate);
                            //Parameters
                            pkt.pts=(double)(frame_index*calc_duration)/(double)(av_q2d(time_base1)*AV_TIME_BASE);
                            pkt.dts=pkt.pts;
                            pkt.duration=(double)calc_duration/(double)(av_q2d(time_base1)*AV_TIME_BASE);
                            frame_index++;
                        }
                        cur_pts_a=pkt.pts;

                        break;
                    }
                }while(av_read_frame(ifmt_ctx, &pkt) >= 0);
            }else{
                break;
            }

        }

        //FIX:Bitstream Filter
#if USE_H264BSF
        av_bitstream_filter_filter(h264bsfc, in_stream->codec, NULL, &pkt.data, &pkt.size, pkt.data, pkt.size, 0);
#endif
#if USE_AACBSF
        av_bitstream_filter_filter(aacbsfc, out_stream->codec, NULL, &pkt.data, &pkt.size, pkt.data, pkt.size, 0);
#endif


        //Convert PTS/DTS
        pkt.pts = av_rescale_q_rnd(pkt.pts, in_stream->time_base, out_stream->time_base, (AVRounding)(AV_ROUND_NEAR_INF|AV_ROUND_PASS_MINMAX));
        pkt.dts = av_rescale_q_rnd(pkt.dts, in_stream->time_base, out_stream->time_base, (AVRounding)(AV_ROUND_NEAR_INF|AV_ROUND_PASS_MINMAX));
        pkt.duration = av_rescale_q(pkt.duration, in_stream->time_base, out_stream->time_base);
        pkt.pos = -1;
        pkt.stream_index=stream_index;

        printf("Write 1 Packet. size:%5d\tpts:%lld\n",pkt.size,pkt.pts);
```

```
278.         //Write
279.         if (av_interleaved_write_frame(ofmt_ctx, &pkt) < 0) {
280.             printf( "Error muxing packet\n");
281.             break;
282.         }
283.         av_free_packet(&pkt);
284.
285.     }
286.     //Write file trailer
287.     av_write_trailer(ofmt_ctx);
288.
289. #if USE_H264BSF
290.     av_bitstream_filter_close(h264bsfc);
291. #endif
292. #if USE_AACBSF
293.     av_bitstream_filter_close(aacbsfc);
294. #endif
295.
296. end:
297.     avformat_close_input(&ifmt_ctx_v);
298.     avformat_close_input(&ifmt_ctx_a);
299.     /* close output */
300.     if (ofmt_ctx && !(ofmt->flags & AVFMT_NOFILE))
301.         avio_close(ofmt_ctx->pb);
302.     avformat_free_context(ofmt_ctx);
303.     if (ret < 0 && ret != AVERROR_EOF) {
304.         printf( "Error occurred.\n");
305.         return -1;
306.     }
307.     return 0;
308. }
```

结果

输入文件为：
视频：cuc_ieschool.ts
音频：huoyuanjia.mp3

输出文件为：
cuc_ieschool.mp4
输出的文件视频为“cuc_ieschool”，配合“霍元甲”的音频。

下载

simplest ffmpeg format

项目主页

SourceForge： https://sourceforge.net/projects/simplestffmpegformat/

Github： https://github.com/leixiaohua1020/simplest_ffmpeg_format

开源中国： http://git.oschina.net/leixiaohua1020/simplest_ffmpeg_format

CSDN下载：
http://download.csdn.net/detail/leixiaohua1020/8005317

工程中包含4个例子：

simplest_ffmpeg_demuxer_simple：视音频分离器（简化版）。

simplest_ffmpeg_demuxer：视音频分离器。

simplest_ffmpeg_muxer：视音频复用器。

simplest_ffmpeg_remuxer：封装格式转换器。

更新-1.1==========

修复了以下问题：

(1)Release版本下的运行问题

(2)simplest_ffmpeg_muxer封装H.264裸流的时候丢失声音的错误

关于simplest_ffmpeg_muxer封装H.264裸流的时候丢失声音的问题目前已经解决。根源在于H.264裸流没有PTS，因此必须手动写入PTS。写入PTS的代码在在旧版本中已经包含：

```cpp
//FIX：No PTS
//Simple Write PTS
if(pkt.pts==AV_NOPTS_VALUE){
    //Write PTS
    AVRational time_base1=in_stream->time_base;
    //Duration between 2 frames (us)
    int64_t calc_duration=(double)AV_TIME_BASE/av_q2d(in_stream->r_frame_rate);
    //Parameters
    pkt.pts=(double)(frame_index*calc_duration)/(double)(av_q2d(time_base1)*AV_TIME_BASE);
    pkt.dts=pkt.pts;
    pkt.duration=(double)calc_duration/(double)(av_q2d(time_base1)*AV_TIME_BASE);
    frame_index++;
}
```

但是旧版本中这段代码的位置放错了，应该放在av_read_frame()之后，cur_pts_a/cur_pts_v赋值之前。换句话说，也就说要把这段代码“前移”。修改后问题解决。
CSDN下载地址：

http://download.csdn.net/detail/leixiaohua1020/8284309

更新-1.2 (2015.2.13)==========

这次考虑到了跨平台的要求，调整了源代码。经过这次调整之后，源代码可以在以下平台编译通过：

VC++：打开sln文件即可编译，无需配置。

cl.exe：打开compile_cl.bat即可命令行下使用cl.exe进行编译，注意可能需要按照VC的安装路径调整脚本里面的参数。编译命令如下。

```plain
::VS2010 Environment
call "D:\Program Files\Microsoft Visual Studio 10.0\VC\vcvarsall.bat"
::include
@set INCLUDE=include;%INCLUDE%
::lib
@set LIB=lib;%LIB%
::compile and link
cl simplest_ffmpeg_muxer.cpp /link avcodec.lib avformat.lib avutil.lib ^
avdevice.lib avfilter.lib postproc.lib swresample.lib swscale.lib /OPT:NOREF
```

MinGW：MinGW命令行下运行compile_mingw.sh即可使用MinGW的g++进行编译。编译命令如下。

```plain
g++ simplest_ffmpeg_muxer.cpp -g -o simplest_ffmpeg_muxer.exe \
-I /usr/local/include -L /usr/local/lib -lavformat -lavcodec -lavutil
```

GCC：Linux或者MacOS命令行下运行compile_gcc.sh即可使用GCC进行编译。编译命令如下。

```plain
gcc simplest_ffmpeg_muxer.cpp -g -o simplest_ffmpeg_muxer.out -I /usr/local/include -L /usr/local/lib \
-lavformat -lavcodec -lavutil
```

PS：相关的编译命令已经保存到了工程文件夹中

CSDN下载地址： http://download.csdn.net/detail/leixiaohua1020/8445303
SourceForge上已经更新。

文章标签： ffmpeg mux 复用 封装

个人分类： FFMPEG 我的开源项目

所属专栏： FFmpeg