

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：57313

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）](#)

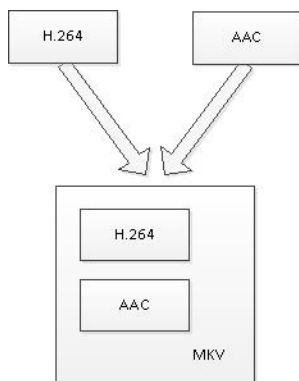
[最简单的基于FFmpeg的封装格式处理：视音频复用器（muxer）](#)

[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。

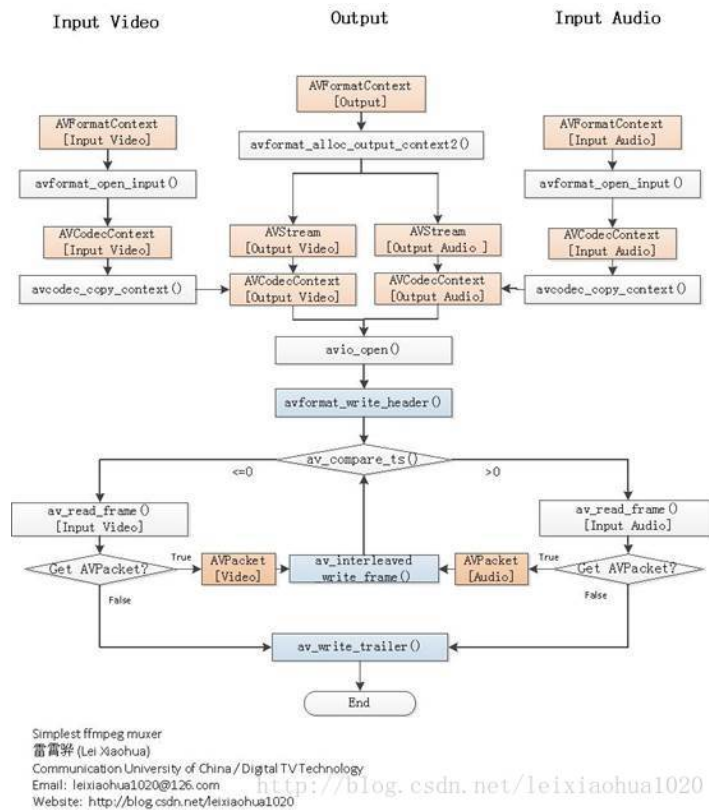
本文记录一个基于FFmpeg的视音频复用器(Simplest FFmpeg muxer)。视音频复用器(Muxer)即是将视频压缩数据(例如H.264)和音频压缩数据(例如AAC)合并到一个封装格式数据(例如MKV)中去。如图所示。在这个过程中并不涉及到编码和解码。



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：

```

1.  /**
2.   * 最简单的基于FFmpeg的视音频复用器
3.   *  Simplest FFmpeg Muxer
4.   *
5.   *  雷霄骅 Lei Xiaohua
6.   *  leixiaohua1020@126.com
7.   *  中国传媒大学/数字电视技术
8.   *  Communication University of China / Digital TV Technology
9.   *  http://blog.csdn.net/leixiaohua1020
10.  *
11.  *  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。
12.  *  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
13.  *  MPEG2TS封装格式的文件。
14.  *  需要注意的是本程序并不改变视音频的编码格式。
15.  *
16.  *  This software mux a video bitstream and a audio bitstream
17.  *  together into a file.
18.  *  In this example, it mux a H.264 bitstream (in MPEG2TS) and
19.  *  a AAC bitstream file together into MP4 format file.
20.  *
21.  */
22.
  
```

```

23. #include <stdio.h>
24.
25. #define __STDC_CONSTANT_MACROS
26.
27. #ifdef _WIN32
28. //Windows
29. extern "C"
30. {
31. #include "libavformat/avformat.h"
32. };
33. #else
34. //Linux...
35. #ifdef __cplusplus
36. extern "C"
37. {
38. #endif
39. #include <libavformat/avformat.h>
40. #ifdef __cplusplus
41. };
42. #endif
43. #endif
44.
45. /*
46. FIX: H.264 in some container format (FLV, MP4, MKV etc.) need
47. "h264_mp4toannexb" bitstream filter (BSF)
48. *Add SPS,PPS in front of IDR frame
49. *Add start code ("0,0,0,1") in front of NALU
50. H.264 in some container (MPEG2TS) don't need this BSF.
51. */
52. // '1': Use H.264 Bitstream Filter
53. #define USE_H264BSF 0
54.
55. /*
56. FIX:AAC in some container format (FLV, MP4, MKV etc.) need
57. "aac_adtstoasc" bitstream filter (BSF)
58. */
59. // '1': Use AAC Bitstream Filter
60. #define USE_AACBSF 0
61.
62.
63.
64. int main(int argc, char* argv[])
65. {
66.     AVOutputFormat *ofmt = NULL;
67.     //Input AVFormatContext and Output AVFormatContext
68.     AVFormatContext *ifmt_ctx_v = NULL, *ifmt_ctx_a = NULL, *ofmt_ctx = NULL;
69.     AVPacket pkt;
70.     int ret, i;
71.     int videoindex_v=-1, videoindex_out=-1;
72.     int audioindex_a=-1, audioindex_out=-1;
73.     int frame_index=0;
74.     int64_t cur_pts_v=0, cur_pts_a=0;
75.
76.     //const char *in_filename_v = "cuc_ieschool.ts";//Input file URL
77.     const char *in_filename_v = "cuc_ieschool.h264";
78.     //const char *in_filename_a = "cuc_ieschool.mp3";
79.     //const char *in_filename_a = "gowest.m4a";
80.     //const char *in_filename_a = "gowest.aac";
81.     const char *in_filename_a = "huoyuanjia.mp3";
82.
83.     const char *out_filename = "cuc_ieschool.mp4";//Output file URL
84.     av_register_all();
85.     //Input
86.     if ((ret = avformat_open_input(&ifmt_ctx_v, in_filename_v, 0, 0)) < 0) {
87.         printf( "Could not open input file.");
88.         goto end;
89.     }
90.     if ((ret = avformat_find_stream_info(ifmt_ctx_v, 0)) < 0) {
91.         printf( "Failed to retrieve input stream information");
92.         goto end;
93.     }
94.
95.     if ((ret = avformat_open_input(&ifmt_ctx_a, in_filename_a, 0, 0)) < 0) {
96.         printf( "Could not open input file.");
97.         goto end;
98.     }
99.     if ((ret = avformat_find_stream_info(ifmt_ctx_a, 0)) < 0) {
100.         printf( "Failed to retrieve input stream information");
101.         goto end;
102.     }
103.     printf("=====Input Information=====\\n");
104.     av_dump_format(ifmt_ctx_v, 0, in_filename_v, 0);
105.     av_dump_format(ifmt_ctx_a, 0, in_filename_a, 0);
106.     printf("=====\\n");
107.     //Output
108.     avformat_alloc_output_context2(&ofmt_ctx, NULL, NULL, out_filename);
109.     if (!ofmt_ctx) {
110.         printf( "Could not create output context\\n");
111.         ret = AVERROR_UNKNOWN;
112.         goto end;
113.     }

```

```

114. ofmt = ofmt_ctx->oformat;
115.
116. for (i = 0; i < ifmt_ctx_v->nb_streams; i++) {
117.     //Create output AVStream according to input AVStream
118.     if(ifmt_ctx_v->streams[i]->codec->codec_type==AVMEDIA_TYPE_VIDEO){
119.         AVStream *in_stream = ifmt_ctx_v->streams[i];
120.         AVStream *out_stream = avformat_new_stream(ofmt_ctx, in_stream->codec->codec);
121.         videoindex_v=i;
122.         if (!out_stream) {
123.             printf("Failed allocating output stream\n");
124.             ret = AVERROR_UNKNOWN;
125.             goto end;
126.         }
127.         videoindex_out=out_stream->index;
128.         //Copy the settings of AVCodecContext
129.         if (avcodec_copy_context(out_stream->codec, in_stream->codec) < 0) {
130.             printf("Failed to copy context from input to output stream codec context\n");
131.             goto end;
132.         }
133.         out_stream->codec->codec_tag = 0;
134.         if (ofmt_ctx->oformat->flags & AVFMT_GLOBALHEADER)
135.             out_stream->codec->flags |= CODEC_FLAG_GLOBAL_HEADER;
136.         break;
137.     }
138. }
139.
140. for (i = 0; i < ifmt_ctx_a->nb_streams; i++) {
141.     //Create output AVStream according to input AVStream
142.     if(ifmt_ctx_a->streams[i]->codec->codec_type==AVMEDIA_TYPE_AUDIO){
143.         AVStream *in_stream = ifmt_ctx_a->streams[i];
144.         AVStream *out_stream = avformat_new_stream(ofmt_ctx, in_stream->codec->codec);
145.         audioindex_a=i;
146.         if (!out_stream) {
147.             printf("Failed allocating output stream\n");
148.             ret = AVERROR_UNKNOWN;
149.             goto end;
150.         }
151.         audioindex_out=out_stream->index;
152.         //Copy the settings of AVCodecContext
153.         if (avcodec_copy_context(out_stream->codec, in_stream->codec) < 0) {
154.             printf("Failed to copy context from input to output stream codec context\n");
155.             goto end;
156.         }
157.         out_stream->codec->codec_tag = 0;
158.         if (ofmt_ctx->oformat->flags & AVFMT_GLOBALHEADER)
159.             out_stream->codec->flags |= CODEC_FLAG_GLOBAL_HEADER;
160.
161.         break;
162.     }
163. }
164.
165. printf("====Output Information====\n");
166. av_dump_format(ofmt_ctx, 0, out_filename, 1);
167. printf("====\n");
168. //Open output file
169. if (!(ofmt->flags & AVFMT_NOFILE)) {
170.     if (avio_open(&ofmt_ctx->pb, out_filename, AVIO_FLAG_WRITE) < 0) {
171.         printf("Could not open output file '%s'", out_filename);
172.         goto end;
173.     }
174. }
175. //Write file header
176. if (avformat_write_header(ofmt_ctx, NULL) < 0) {
177.     printf("Error occurred when opening output file\n");
178.     goto end;
179. }
180.
181.
182. //FIX
183. #if USE_H264BSF
184. AVBitStreamFilterContext* h264bsfc = av_bitstream_filter_init("h264_mp4toannexb");
185. #endif
186. #if USE_AACBSF
187. AVBitStreamFilterContext* aacbsfc = av_bitstream_filter_init("aac_adtstoasc");
188. #endif
189.
190. while (1) {
191.     AVFormatContext *ifmt_ctx;
192.     int stream_index=0;
193.     AVStream *in_stream, *out_stream;
194.
195.     //Get an AVPacket
196.     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){
197.         ifmt_ctx=ifmt_ctx_v;
198.         stream_index=videoindex_out;
199.
200.         if (av_read_frame(ifmt_ctx, &pkt) >= 0){
201.             do{
202.                 in_stream = ifmt_ctx->streams[pkt.stream_index];
203.                 out_stream = ofmt_ctx->streams[stream_index];

```

```

204.         if(pkt.stream_index==videoindex_v){
205.             //FIX:No PTS (Example: Raw H.264)
206.             //Simple Write PTS
207.             if(pkt.pts==AV_NOPTS_VALUE){
208.                 //Write PTS
209.                 AVRational time_base1=in_stream->time_base;
210.                 //Duration between 2 frames (us)
211.                 int64_t calc_duration=(double)AV_TIME_BASE/av_q2d(in_stream->r_frame_rate);
212.                 //Parameters
213.                 pkt.pts=(double)(frame_index*calc_duration)/(double)(av_q2d(time_base1)*AV_TIME_BASE);
214.                 pkt.dts=pkt.pts;
215.                 pkt.duration=(double)calc_duration/(double)(av_q2d(time_base1)*AV_TIME_BASE);
216.                 frame_index++;
217.             }
218.
219.
220.             cur_pts_v=pkt.pts;
221.             break;
222.         }
223.     }while(av_read_frame(ifmt_ctx, &pkt) >= 0);
224. }else{
225.     break;
226. }
227. }else{
228.     ifmt_ctx=ifmt_ctx_a;
229.     stream_index=audioindex_out;
230.     if(av_read_frame(ifmt_ctx, &pkt) >= 0){
231.         do{
232.             in_stream = ifmt_ctx->streams[pkt.stream_index];
233.             out_stream = ofmt_ctx->streams[stream_index];
234.
235.             if(pkt.stream_index==audioindex_a){
236.
237.                 //FIX:No PTS
238.                 //Simple Write PTS
239.                 if(pkt.pts==AV_NOPTS_VALUE){
240.                     //Write PTS
241.                     AVRational time_base1=in_stream->time_base;
242.                     //Duration between 2 frames (us)
243.                     int64_t calc_duration=(double)AV_TIME_BASE/av_q2d(in_stream->r_frame_rate);
244.                     //Parameters
245.                     pkt.pts=(double)(frame_index*calc_duration)/(double)(av_q2d(time_base1)*AV_TIME_BASE);
246.                     pkt.dts=pkt.pts;
247.                     pkt.duration=(double)calc_duration/(double)(av_q2d(time_base1)*AV_TIME_BASE);
248.                     frame_index++;
249.                 }
250.                 cur_pts_a=pkt.pts;
251.
252.                 break;
253.             }
254.         }while(av_read_frame(ifmt_ctx, &pkt) >= 0);
255.     }else{
256.         break;
257.     }
258. }
259.
260.
261. //FIX:Bitstream Filter
262. #if USE_H264BSF
263.     av_bitstream_filter_filter(h264bsfc, in_stream->codec, NULL, &pkt.data, &pkt.size, pkt.data, pkt.size, 0);
264. #endif
265. #if USE_AACBSF
266.     av_bitstream_filter_filter(aacbsfc, out_stream->codec, NULL, &pkt.data, &pkt.size, pkt.data, pkt.size, 0);
267. #endif
268.
269.
270. //Convert PTS/DTS
271. 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));
272. 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));
273. pkt.duration = av_rescale_q(pkt.duration, in_stream->time_base, out_stream->time_base);
274. pkt.pos = -1;
275. pkt.stream_index=stream_index;
276.
277. 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);

```

```

293.         av_ditstream_filter_close(aacOSTC);
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 != AVEERROR_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的代码在旧版本中已经包含：

```

1. //FIX: No PTS
2. //Simple Write PTS
3. if(pkt.pts==AV_NOPTS_VALUE){
4.     //Write PTS
5.     AVRational time_base1=in_stream->time_base;
6.     //Duration between 2 frames (us)
7.     int64_t calc_duration=(double)AV_TIME_BASE/av_q2d(in_stream->r_frame_rate);
8.     //Parameters
9.     pkt.pts=(double)(frame_index*calc_duration)/(double)(av_q2d(time_base1)*AV_TIME_BASE);
10.    pkt.dts=pkt.pts;
11.    pkt.duration=(double)calc_duration/(double)(av_q2d(time_base1)*AV_TIME_BASE);
12.    frame_index++;
13. }

```

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。

```

1. ::VS2010 Environment
2. call "D:\Program Files\Microsoft Visual Studio 10.0\VC\vcvarsall.bat"
3. ::include
4. @set INCLUDE=include;%INCLUDE%
5. ::lib
6. @set LIB=lib;%LIB%
7. ::compile and link
8. cl simplest_ffmpeg_muxer.cpp /link avcodec.lib avformat.lib avutil.lib ^
9. 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。

```

1. g++ simplest_ffmpeg_muxer.cpp -g -o simplest_ffmpeg_muxer.exe \
2. -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。

```

1. gcc simplest_ffmpeg_muxer.cpp -g -o simplest_ffmpeg_muxer.out -I /usr/local/include -L /usr/local/lib \
2. -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。

版权声明：本文为博主原创文章，未经博主允许不得转载。 <https://blog.csdn.net/leixiaohua1020/article/details/39802913>

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](#)

此PDF由spygg生成,请尊重原作者版权!!!

我的邮箱:liushidc@163.com