

音视频数据处理入门系列文章：

[音视频数据处理入门：RGB、YUV像素数据处理](#)

[音视频数据处理入门：PCM音频采样数据处理](#)

[音视频数据处理入门：H.264视频码流解析](#)

[音视频数据处理入门：AAC音频码流解析](#)

[音视频数据处理入门：FLV封装格式解析](#)

[音视频数据处理入门：UDP-RTP协议解析](#)

前两篇文章介绍了音频码流处理程序和视频码流处理程序，本文介绍将他们打包到一起后的数据——封装格式数据的处理程序。封装格式数据在视频播放器中的位置如下所示。



本文中的程序是一个FLV封装格式解析程序。该程序可以从FLV中分析得到它的基本单元Tag，并且可以简单解析Tag首部的字段。通过修改该程序可以实现不同的FLV格式数据处理功能。

## 原理

FLV封装格式是由一个FLV Header文件头和一个一个的Tag组成的。Tag中包含了音频数据以及视频数据。FLV的结构如下图所示。



有关FLV的格式本文不再做记录。可以参考文章《[音视频编解码学习工程：FLV封装格式分析器](#)》。本文的程序实现了FLV中的FLV Header和Tag的解析，并可以分离出其中的音频流。

## 代码

整个程序位于simplest\_flv\_parser()函数中，如下所示。

```
[cpp]
1. /**
2.  * 最简单的音视频数据处理示例
3.  * Simplest MediaData Test
4.  *
5.  * 雷霄骅 Lei Xiaohua
6.  * leixiaohua1020@126.com
7.  * 中国传媒大学 / 数字电视技术
```

```

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. * 本项目包含如下几种音视频测试示例：
12. * (1) 像素数据处理程序。包含RGB和YUV像素格式处理的函数。
13. * (2) 音频采样数据处理程序。包含PCM音频采样格式处理的函数。
14. * (3) H.264码流分析程序。可以分离并解析NALU。
15. * (4) AAC码流分析程序。可以分离并解析ADTS帧。
16. * (5) FLV封装格式分析程序。可以将FLV中的MP3音频码流分离出来。
17. * (6) UDP-RTP协议分析程序。可以将分析UDP/RTP/MPEG-TS数据包。
18. *
19. * This project contains following samples to handling multimedia data:
20. * (1) Video pixel data handling program. It contains several examples to handle RGB and YUV data.
21. * (2) Audio sample data handling program. It contains several examples to handle PCM data.
22. * (3) H.264 stream analysis program. It can parse H.264 bitstream and analysis NALU of stream.
23. * (4) AAC stream analysis program. It can parse AAC bitstream and analysis ADTS frame of stream.
24. * (5) FLV format analysis program. It can analysis FLV file and extract MP3 audio stream.
25. * (6) UDP-RTP protocol analysis program. It can analysis UDP/RTP/MPEG-TS Packet.
26. *
27. */
28. #include <stdio.h>
29. #include <stdlib.h>
30. #include <string.h>
31.
32. //Important!
33. #pragma pack(1)
34.
35.
36. #define TAG_TYPE_SCRIPT 18
37. #define TAG_TYPE_AUDIO 8
38. #define TAG_TYPE_VIDEO 9
39.
40. typedef unsigned char byte;
41. typedef unsigned int uint;
42.
43. typedef struct {
44.     byte Signature[3];
45.     byte Version;
46.     byte Flags;
47.     uint DataOffset;
48. } FLV_HEADER;
49.
50. typedef struct {
51.     byte TagType;
52.     byte DataSize[3];
53.     byte Timestamp[3];
54.     uint Reserved;
55. } TAG_HEADER;
56.
57.
58. //reverse_bytes - turn a BigEndian byte array into a LittleEndian integer
59. uint reverse_bytes(byte *p, char c) {
60.     int r = 0;
61.     int i;
62.     for (i=0; i<c; i++)
63.         r |= ( *(p+i) << (((c-1)*8)-8*i));
64.     return r;
65. }
66.
67. /**
68.  * Analysis FLV file
69.  * @param url    Location of input FLV file.
70.  */
71.
72. int simplest_flv_parser(char *url){
73.
74.     //whether output audio/video stream
75.     int output_a=1;
76.     int output_v=1;
77.     //-----
78.     FILE *ifh=NULL,*vfh=NULL, *afh = NULL;
79.
80.     //FILE *myout=fopen("output_log.txt","wb+");
81.     FILE *myout=stdout;
82.
83.     FLV_HEADER flv;
84.     TAG_HEADER tagheader;
85.     uint previoustagsize, previoustagsize_z=0;
86.     uint ts=0, ts_new=0;
87.
88.     ifh = fopen(url, "rb+");
89.     if ( ifh== NULL) {
90.         printf("Failed to open files!");
91.         return -1;
92.     }
93.
94.     //FLV file header
95.     fread((char *)&flv,1,sizeof(FLV_HEADER),ifh);
96.
97.     fprintf(myout,"===== FLV Header =====\n");
98.     fprintf(myout,"Signature: 0x %c %c %c\n",flv.Signature[0],flv.Signature[1],flv.Signature[2]);

```

```

99.     fprintf(myout, "Version:    0x %X\n", flv.Version);
100.    fprintf(myout, "Flags :    0x %X\n", flv.Flags);
101.    fprintf(myout, "HeaderSize: 0x %X\n", reverse_bytes((byte *)&flv.DataOffset, sizeof(flv.DataOffset)));
102.    fprintf(myout, "=====\\n");
103.
104.    //move the file pointer to the end of the header
105.    fseek(ifh, reverse_bytes((byte *)&flv.DataOffset, sizeof(flv.DataOffset)), SEEK_SET);
106.
107.    //process each tag
108.    do {
109.
110.        previoustagsize = _getw(ifh);
111.
112.        fread((void *)&tagheader, sizeof(TAG_HEADER), 1, ifh);
113.
114.        //int temp_datasize1=reverse_bytes((byte *)&tagheader.DataSize, sizeof(tagheader.DataSize));
115.        int tagheader_datasize=tagheader.DataSize[0]*65536+tagheader.DataSize[1]*256+tagheader.DataSize[2];
116.        int tagheader_timestamp=tagheader.Timestamp[0]*65536+tagheader.Timestamp[1]*256+tagheader.Timestamp[2];
117.
118.        char tagtype_str[10];
119.        switch(tagheader.TagType){
120.        case TAG_TYPE_AUDIO:sprintf(tagtype_str, "AUDIO");break;
121.        case TAG_TYPE_VIDEO:sprintf(tagtype_str, "VIDEO");break;
122.        case TAG_TYPE_SCRIPT:sprintf(tagtype_str, "SCRIPT");break;
123.        default:sprintf(tagtype_str, "UNKNOWN");break;
124.        }
125.        fprintf(myout, "[%5s] %6d %6d |", tagtype_str, tagheader_datasize, tagheader_timestamp);
126.
127.        //if we are not past the end of file, process the tag
128.        if (feof(ifh)) {
129.            break;
130.        }
131.
132.        //process tag by type
133.        switch (tagheader.TagType) {
134.
135.        case TAG_TYPE_AUDIO:{
136.            char audiotag_str[100]={0};
137.            strcat(audiotag_str, "| ");
138.            char tagdata_first_byte;
139.            tagdata_first_byte=fgetc(ifh);
140.            int x=tagdata_first_byte&0xF0;
141.            x=x>>4;
142.            switch (x)
143.            {
144.            case 0:strcat(audiotag_str, "Linear PCM, platform endian");break;
145.            case 1:strcat(audiotag_str, "ADPCM");break;
146.            case 2:strcat(audiotag_str, "MP3");break;
147.            case 3:strcat(audiotag_str, "Linear PCM, little endian");break;
148.            case 4:strcat(audiotag_str, "Nellymoser 16-kHz mono");break;
149.            case 5:strcat(audiotag_str, "Nellymoser 8-kHz mono");break;
150.            case 6:strcat(audiotag_str, "Nellymoser");break;
151.            case 7:strcat(audiotag_str, "G.711 A-law logarithmic PCM");break;
152.            case 8:strcat(audiotag_str, "G.711 mu-law logarithmic PCM");break;
153.            case 9:strcat(audiotag_str, "reserved");break;
154.            case 10:strcat(audiotag_str, "AAC");break;
155.            case 11:strcat(audiotag_str, "Speex");break;
156.            case 14:strcat(audiotag_str, "MP3 8-Khz");break;
157.            case 15:strcat(audiotag_str, "Device-specific sound");break;
158.            default:strcat(audiotag_str, "UNKNOWN");break;
159.            }
160.            strcat(audiotag_str, "| ");
161.            x=tagdata_first_byte&0x0C;
162.            x=x>>2;
163.            switch (x)
164.            {
165.            case 0:strcat(audiotag_str, "5.5-kHz");break;
166.            case 1:strcat(audiotag_str, "1-kHz");break;
167.            case 2:strcat(audiotag_str, "22-kHz");break;
168.            case 3:strcat(audiotag_str, "44-kHz");break;
169.            default:strcat(audiotag_str, "UNKNOWN");break;
170.            }
171.            strcat(audiotag_str, "| ");
172.            x=tagdata_first_byte&0x02;
173.            x=x>>1;
174.            switch (x)
175.            {
176.            case 0:strcat(audiotag_str, "8Bit");break;
177.            case 1:strcat(audiotag_str, "16Bit");break;
178.            default:strcat(audiotag_str, "UNKNOWN");break;
179.            }
180.            strcat(audiotag_str, "| ");
181.            x=tagdata_first_byte&0x01;
182.            switch (x)
183.            {
184.            case 0:strcat(audiotag_str, "Mono");break;
185.            case 1:strcat(audiotag_str, "Stereo");break;
186.            default:strcat(audiotag_str, "UNKNOWN");break;
187.            }
188.            fprintf(myout, "%s", audiotag_str);
189.

```

```

190.         //if the output file hasn't been opened, open it.
191.         if(output_a!=0&&afh == NULL){
192.             afh = fopen("output.mp3", "wb");
193.         }
194.
195.         //TagData - First Byte Data
196.         int data_size=reverse_bytes((byte *)&tagheader.DataSize, sizeof(tagheader.DataSize))-1;
197.         if(output_a!=0){
198.             //TagData+1
199.             for (int i=0; i<data_size; i++)
200.                 fputc(fgetc(ifh),afh);
201.
202.         }else{
203.             for (int i=0; i<data_size; i++)
204.                 fgetc(ifh);
205.         }
206.         break;
207.     }
208.     case TAG_TYPE_VIDEO:{
209.         char videotag_str[100]={0};
210.         strcat(videotag_str,"| ");
211.         char tagdata_first_byte;
212.         tagdata_first_byte=fgetc(ifh);
213.         int x=tagdata_first_byte&0xF0;
214.         x=x>>4;
215.         switch (x)
216.         {
217.             case 1:strcat(videotag_str,"key frame ");break;
218.             case 2:strcat(videotag_str,"inter frame");break;
219.             case 3:strcat(videotag_str,"disposable inter frame");break;
220.             case 4:strcat(videotag_str,"generated keyframe");break;
221.             case 5:strcat(videotag_str,"video info/command frame");break;
222.             default:strcat(videotag_str,"UNKNOWN");break;
223.         }
224.         strcat(videotag_str,"| ");
225.         x=tagdata_first_byte&0x0F;
226.         switch (x)
227.         {
228.             case 1:strcat(videotag_str,"JPEG (currently unused)");break;
229.             case 2:strcat(videotag_str,"Sorenson H.263");break;
230.             case 3:strcat(videotag_str,"Screen video");break;
231.             case 4:strcat(videotag_str,"On2 VP6");break;
232.             case 5:strcat(videotag_str,"On2 VP6 with alpha channel");break;
233.             case 6:strcat(videotag_str,"Screen video version 2");break;
234.             case 7:strcat(videotag_str,"AVC");break;
235.             default:strcat(videotag_str,"UNKNOWN");break;
236.         }
237.         fprintf(myout,"%s",videotag_str);
238.
239.         fseek(ifh, -1, SEEK_CUR);
240.         //if the output file hasn't been opened, open it.
241.         if (vfh == NULL&&output_v!=0) {
242.             //write the flv header (reuse the original file's hdr) and first previoustagsize
243.             vfh = fopen("output.flv", "wb");
244.             fwrite((char *)&flv,1, sizeof(flv),vfh);
245.             fwrite((char *)&previoustagsize_z,1,sizeof(previoustagsize_z),vfh);
246.         }
247.     #if 0
248.         //Change Timestamp
249.         //Get Timestamp
250.         ts = reverse_bytes((byte *)&tagheader.Timestamp, sizeof(tagheader.Timestamp));
251.         ts=ts*2;
252.         //Writeback Timestamp
253.         ts_new = reverse_bytes((byte *)&ts, sizeof(ts));
254.         memcpy(&tagheader.Timestamp, ((char *)&ts_new) + 1, sizeof(tagheader.Timestamp));
255.     #endif
256.
257.
258.         //TagData + Previous Tag Size
259.         int data_size=reverse_bytes((byte *)&tagheader.DataSize, sizeof(tagheader.DataSize))+4;
260.         if(output_v!=0){
261.             //TagHeader
262.             fwrite((char *)&tagheader,1, sizeof(tagheader),vfh);
263.             //TagData
264.             for (int i=0; i<data_size; i++)
265.                 fputc(fgetc(ifh),vfh);
266.         }else{
267.             for (int i=0; i<data_size; i++)
268.                 fgetc(ifh);
269.         }
270.         //rewind 4 bytes, because we need to read the previoustagsize again for the loop's sake
271.         fseek(ifh, -4, SEEK_CUR);
272.
273.         break;
274.     }
275.     default:
276.
277.         //skip the data of this tag
278.         fseek(ifh, reverse_bytes((byte *)&tagheader.DataSize, sizeof(tagheader.DataSize)), SEEK_CUR);
279.     }
280.

```

```

281.         fprintf(myout, "\n");
282.
283.     } while (!feof(ifh));
284.
285.
286.     _fcloseall();
287.
288.     return 0;
289. }

```

上文中的函数调用方法如下所示。

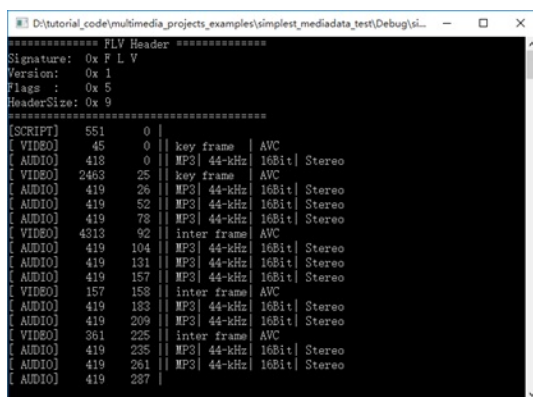
```

1. simplest_flv_parser("cuc_ieschool.flv");

```

## 结果

本程序的输入为一个FLV的文件路径，输出为FLV的统计数据，如下图所示。



```

D:\tutorial_code\multimedia_projects_examples\simplest_mediatadata_test\Debug\si...
===== FLV Header =====
Signature: 0x F L V
Version: 0x 1
Flags : 0x 5
HeaderSize: 0x 9
=====
[SCRIPT] 551 0 |
[VIDEO] 45 0 | key frame | AVC
[AUDIO] 418 0 | MP3 | 44-kHz | 16Bit | Stereo
[VIDEO] 2463 25 | key frame | AVC
[AUDIO] 419 26 | MP3 | 44-kHz | 16Bit | Stereo
[AUDIO] 419 52 | MP3 | 44-kHz | 16Bit | Stereo
[AUDIO] 419 78 | MP3 | 44-kHz | 16Bit | Stereo
[VIDEO] 4313 92 | inter frame | AVC
[AUDIO] 419 104 | MP3 | 44-kHz | 16Bit | Stereo
[AUDIO] 419 131 | MP3 | 44-kHz | 16Bit | Stereo
[AUDIO] 419 157 | MP3 | 44-kHz | 16Bit | Stereo
[VIDEO] 157 158 | inter frame | AVC
[AUDIO] 419 183 | MP3 | 44-kHz | 16Bit | Stereo
[AUDIO] 419 209 | MP3 | 44-kHz | 16Bit | Stereo
[VIDEO] 361 225 | inter frame | AVC
[AUDIO] 419 235 | MP3 | 44-kHz | 16Bit | Stereo
[AUDIO] 419 261 | MP3 | 44-kHz | 16Bit | Stereo
[AUDIO] 419 287 |

```

此外本程序还可以分离FLV中的视频码流和音频码流。需要注意的是本程序并不能分离一些特定类型的音频（例如AAC）和视频，这一工作有待以后有时间再完成。

## 下载

### Simplest mediatadata test

#### 项目主页

SourceForge： <https://sourceforge.net/projects/simplest-mediatadata-test/>

Github： [https://github.com/leixiaohua1020/simplest\\_mediatadata\\_test](https://github.com/leixiaohua1020/simplest_mediatadata_test)

开源中国： [http://git.oschina.net/leixiaohua1020/simplest\\_mediatadata\\_test](http://git.oschina.net/leixiaohua1020/simplest_mediatadata_test)

CSDN下载地址： <http://download.csdn.net/detail/leixiaohua1020/9422409>

本项目包含如下几种视音频数据解析示例：

- (1)像素数据处理程序。包含RGB和YUV像素格式处理的函数。
- (2)音频采样数据处理程序。包含PCM音频采样格式处理的函数。
- (3)H.264码流分析程序。可以分离并解析NALU。
- (4)AAC码流分析程序。可以分离并解析ADTS帧。
- (5)FLV封装格式分析程序。可以将FLV中的MP3音频码流分离出来。
- (6)UDP-RTP协议分析程序。可以将分析UDP/RTP/MPEG-TS数据包。

**雷霄骅 (Lei Xiaohua)**

leixiaohua1020@126.com

<http://blog.csdn.net/leixiaohua1020>

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/50535082>

文章标签：[FLV](#) [封装格式](#) [分离](#) [音频](#) [视频](#)

个人分类：[我的开源项目](#)

此PDF由spygg生成,请尊重原作者版权!!!  
我的邮箱:liushidc@163.com