

## 最简单的基于FFMPEG的推流器附件：收流器

2015年07月18日 08:47:14 阅读数：17655

最简单的基于FFmpeg的推流器系列文章列表：

《最简单的基于FFmpeg的推流器（以推送RTMP为例）》

《最简单的基于FFMPEG的推流器附件：收流器》

出于对《最简单的基于FFmpeg的推流器》的补充，本文记录一个最简单的基于FFmpeg的收流器。收流器和推流器的作用正好相反：推流器用于将本地文件以流媒体的形式发送出去，而收流器用于将流媒体内容保存为本地文件。

本文记录的推流器可以将RTMP流媒体保存成为一个本地的FLV文件。由于FFmpeg本身支持很多的流媒体协议和封装格式，所以也支持其它的封装格式和流媒体协议。

## 源代码

```
[cpp]
1.  /**
2.   * 最简单的基于FFmpeg的收流器（接收RTMP）
3.   * Simplest Ffmpeg Receiver (Receive RTMP)
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.  * 本例子将流媒体数据（以RTMP为例）保存成本地文件。
12.  * 是使用FFmpeg进行流媒体接收最简单的教程。
13.  *
14.  * This example saves streaming media data (Use RTMP as example)
15.  * as a local file.
16.  * It's the simplest Ffmpeg stream receiver.
17.  *
18.  */
19.
20. #include <stdio.h>
21.
22. #define __STDC_CONSTANT_MACROS
23.
24. #ifdef _WIN32
25. //Windows
26. extern "C"
27. {
28. #include "libavformat/avformat.h"
29. #include "libavutil/mathematics.h"
30. #include "libavutil/time.h"
31. };
32. #else
33. //Linux...
34. #ifdef __cplusplus
35. extern "C"
36. {
37. #endif
38. #include <libavformat/avformat.h>
39. #include <libavutil/mathematics.h>
40. #include <libavutil/time.h>
41. #ifdef __cplusplus
42. };
43. #endif
44. #endif
45.
46. //!': Use H.264 Bitstream Filter
47. #define USE_H264BSF 0
48.
49. int main(int argc, char* argv[])
50. {
51.     AVOutputFormat *ofmt = NULL;
52.     //Input AVFormatContext and Output AVFormatContext
```

```

53.     AVFormatContext *ifmt_ctx = NULL, *ofmt_ctx = NULL;
54.     AVPacket pkt;
55.     const char *in_filename, *out_filename;
56.     int ret, i;
57.     int videoindex=-1;
58.     int frame_index=0;
59.     in_filename = "rtmp://live.hkstv.hk.lxdns.com/live/hks";
60.     //in_filename = "rtp://233.233.233.233:6666";
61.     //out_filename = "receive.ts";
62.     //out_filename = "receive.mkv";
63.     out_filename = "receive.flv";
64.
65.     av_register_all();
66.     //Network
67.     avformat_network_init();
68.     //Input
69.     if ((ret = avformat_open_input(&ifmt_ctx, in_filename, 0, 0)) < 0) {
70.         printf( "Could not open input file.");
71.         goto end;
72.     }
73.     if ((ret = avformat_find_stream_info(ifmt_ctx, 0)) < 0) {
74.         printf( "Failed to retrieve input stream information");
75.         goto end;
76.     }
77.
78.     for(i=0; i<ifmt_ctx->nb_streams; i++)
79.         if(ifmt_ctx->streams[i]->codec->codec_type==AVMEDIA_TYPE_VIDEO){
80.             videoindex=i;
81.             break;
82.         }
83.
84.     av_dump_format(ifmt_ctx, 0, in_filename, 0);
85.
86.     //Output
87.     avformat_alloc_output_context2(&ofmt_ctx, NULL, NULL, out_filename); //RTMP
88.
89.     if (!ofmt_ctx) {
90.         printf( "Could not create output context\n");
91.         ret = AERROR_UNKNOWN;
92.         goto end;
93.     }
94.     ofmt = ofmt_ctx->oformat;
95.     for (i = 0; i < ifmt_ctx->nb_streams; i++) {
96.         //Create output AVStream according to input AVStream
97.         AVStream *in_stream = ifmt_ctx->streams[i];
98.         AVStream *out_stream = avformat_new_stream(ofmt_ctx, in_stream->codec->codec);
99.         if (!out_stream) {
100.             printf( "Failed allocating output stream\n");
101.             ret = AERROR_UNKNOWN;
102.             goto end;
103.         }
104.         //Copy the settings of AVCodecContext
105.         ret = avcodec_copy_context(out_stream->codec, in_stream->codec);
106.         if (ret < 0) {
107.             printf( "Failed to copy context from input to output stream codec context\n");
108.             goto end;
109.         }
110.         out_stream->codec->codec_tag = 0;
111.         if (ofmt_ctx->oformat->flags & AVFMT_GLOBALHEADER)
112.             out_stream->codec->flags |= CODEC_FLAG_GLOBAL_HEADER;
113.     }
114.     //Dump Format-----
115.     av_dump_format(ofmt_ctx, 0, out_filename, 1);
116.     //Open output URL
117.     if (!(ofmt->flags & AVFMT_NOFILE)) {
118.         ret = avio_open(&ofmt_ctx->pb, out_filename, AVIO_FLAG_WRITE);
119.         if (ret < 0) {
120.             printf( "Could not open output URL '%s'", out_filename);
121.             goto end;
122.         }
123.     }
124.     //Write file header
125.     ret = avformat_write_header(ofmt_ctx, NULL);
126.     if (ret < 0) {
127.         printf( "Error occurred when opening output URL\n");
128.         goto end;
129.     }
130.
131. #if USE_H264BSF
132.     AVBitStreamFilterContext* h264bsfc = av_bitstream_filter_init("h264_mp4toannexb");
133. #endif
134.
135.     while (1) {
136.         AVStream *in_stream, *out_stream;
137.         //Get an AVPacket
138.         ret = av_read_frame(ifmt_ctx, &pkt);
139.         if (ret < 0)
140.             break;
141.
142.         in_stream = ifmt_ctx->streams[pkt.stream_index];
143.         out_stream = ofmt_ctx->streams[pkt.stream_index];
144.         //Send packet */

```

```

144.         // Copy packet //
145.         //Convert PTS/DTS
146.         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));
147.         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));
148.         pkt.duration = av_rescale_q(pkt.duration, in_stream->time_base, out_stream->time_base);
149.         pkt.pos = -1;
150.         //Print to Screen
151.         if(pkt.stream_index==videoindex){
152.             printf("Receive %8d video frames from input URL\n",frame_index);
153.             frame_index++;
154.
155.             #if USE_H264BSF
156.                 av_bitstream_filter_filter(h264bsfc, in_stream->codec, NULL, &pkt.data, &pkt.size, pkt.data, pkt.size, 0);
157.             #endif
158.             }
159.             //ret = av_write_frame(ofmt_ctx, &pkt);
160.             ret = av_interleaved_write_frame(ofmt_ctx, &pkt);
161.
162.             if (ret < 0) {
163.                 printf( "Error muxing packet\n");
164.                 break;
165.             }
166.
167.             av_free_packet(&pkt);
168.
169.         }
170.
171.         #if USE_H264BSF
172.             av_bitstream_filter_close(h264bsfc);
173.         #endif
174.
175.         //Write file trailer
176.         av_write_trailer(ofmt_ctx);
177.     end:
178.         avformat_close_input(&ifmt_ctx);
179.         /* close output */
180.         if (ofmt_ctx && !(ofmt->flags & AVFMT_NOFILE))
181.             avio_close(ofmt_ctx->pb);
182.         avformat_free_context(ofmt_ctx);
183.         if (ret < 0 && ret != AERROR_EOF) {
184.             printf( "Error occurred.\n");
185.             return -1;
186.         }
187.         return 0;
188.     }

```

## 运行结果

程序运行之后，即可获取流媒体数据并且在本地保存成一个视频文件。

## 下载

### simplest ffmpeg streamer

#### 项目主页

SourceForge： <https://sourceforge.net/projects/simplestffmpegstreamer/>

Github： [https://github.com/leixiaohua1020/simplest\\_ffmpeg\\_streamer](https://github.com/leixiaohua1020/simplest_ffmpeg_streamer)

开源中国： [http://git.oschina.net/leixiaohua1020/simplest\\_ffmpeg\\_streamer](http://git.oschina.net/leixiaohua1020/simplest_ffmpeg_streamer)

CSDN下载地址： <http://download.csdn.net/detail/leixiaohua1020/8924345>

解决方案包含2个项目：

simplest\_ffmpeg\_streamer: 将本地视频文件推送至流媒体服务器。

simplest\_ffmpeg\_receiver: 将流媒体数据保存成本地文件。

文章标签：[FFmpeg](#) [流媒体](#) [视频](#) [RTMP](#)

个人分类：[FFMPEG](#)

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](mailto:liushidc@163.com)