

最简单的基于FFmpeg的推流器（以推送RTMP为例）

2014年10月06日 00:35:29 阅读数：155744

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》

本文记录一个最简单的基于FFmpeg的推流器 (simplest ffmpeg streamer)。推流器的作用就是将本地的视频数据推送至流媒体服务器。本文记录的推流器, 可以将本地的 MOV / AVI / MKV / MP4 / FLV 等格式的媒体文件, 通过流媒体协议 (例如RTMP, HTTP, UDP, TCP, RTP等等) 以直播流的形式推送出去。由于流媒体协议种类繁多, 不一一记录。在这里记录将本地文件以RTMP直播流的形式推送至RTMP流媒体服务器 (例如Flash Media Server, Red5, Wowza等等) 的方法。

在这个推流器的基础上可以进行多种方式的修改, 实现各式各样的推流器。例如：

* 将输入文件改为网络流URL, 可以实现转流器。

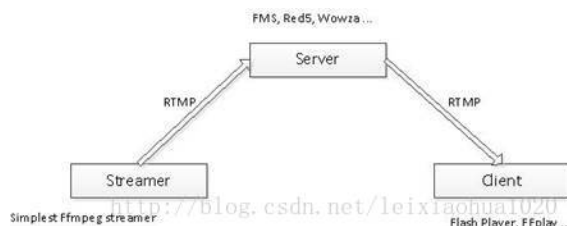
* 将输入的文件改为回调函数 (内存读取) 的形式, 可以推送内存中的视频数据。

* 将输入文件改为系统设备 (通过libavdevice), 同时加上编码的功能, 可以实现实时推流器 (现场直播)。

PS: 本程序并不包含视频转码的功能。

简介

RTMP推流器 (Streamer) 的在流媒体系统中的作用可以用下图表示。首先将视频数据以RTMP的形式发送到流媒体服务器端 (Server, 比如FMS, Red5, Wowza等), 然后客户端 (一般为Flash Player) 通过访问流媒体服务器就可以收看实时流了。



运行本程序之前需要先运行RTMP流媒体服务器, 并在流媒体服务器上建立相应的Application。有关流媒体服务器的操作不在本文的论述范围内, 在此不再详述。本程序运行后, 即可通过RTMP客户端 (例如 Flash Player, FFplay等等) 收看推送的直播流。

需要要注意的地方

封装格式

RTMP采用的封装格式是FLV。因此在指定输出流媒体的时候需要指定其封装格式为“flv”。同理, 其他流媒体协议也需要指定其封装格式。例如采用UDP推送流媒体的时候, 可以指定其封装格式为“mpegts”。

延时

发送流媒体的数据的时候需要延时。不然的话, FFmpeg处理数据速度很快, 瞬间就能把所有的数据发送出去, 流媒体服务器是接受不了的。因此需要按照视频实际的帧率发送数据。本文记录的推流器在视频帧与帧之间采用了av_usleep()函数休眠的方式来延迟发送。这样就可以按照视频的帧率发送数据了, 参考代码如下。

```
[cpp]
1.  //...
2.  int64_t start_time=av_gettime();
3.  while (1) {
4.      //...
5.      //Important:Delay
6.      if(pkt.stream_index==videoindex){
7.          AVRational time_base=ifmt_ctx->streams[videoindex]->time_base;
8.          AVRational time_base_q={1,AV_TIME_BASE};
9.          int64_t pts_time = av_rescale_q(pkt.dts, time_base, time_base_q);
10.         int64_t now_time = av_gettime() - start_time;
11.         if (pts_time > now_time)
12.             av_usleep(pts_time - now_time);
13.     }
14.     //...
15. }
16. //...
```

PTS/DTS问题

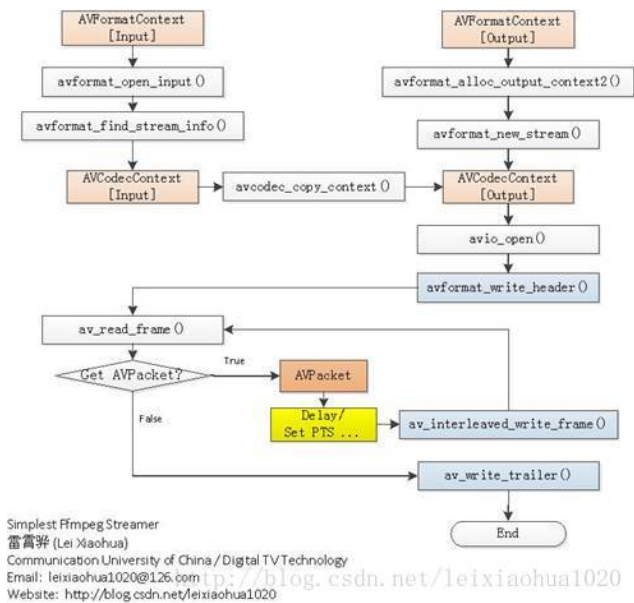
没有封装格式的裸流（例如H.264裸流）是不包含PTS、DTS这些参数的。在发送这种数据的时候，需要自己计算并写入AVPacket的PTS，DTS，duration等参数。这里还没有深入研究，简单写了一点代码，如下所示。

```
[cpp]
1. //FIX:No PTS (Example: Raw H.264)
2. //Simple Write PTS
3. if(pkt.pts==AV_NOPTS_VALUE){
4.     //Write PTS
5.     AVRational time_base1=ifmt_ctx->streams[videoindex]->time_base;
6.     //Duration between 2 frames (us)
7.     int64_t calc_duration=(double)AV_TIME_BASE/av_q2d(ifmt_ctx->streams[videoindex]->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. }
```

程序流程图

程序的流程图如下图所示。可以看出和《最简单的基于FFMPEG的封装格式转换器（无编解码）》中的封装格式转换器比较类似。它们之间比较明显的区别在于：

- 1. Streamer输出为URL
- 2. Streamer包含了延时部分



代码

代码如下。

```
[cpp]
1. /**
2.  * 最简单的基于FFmpeg的推流器（推送RTMP）
3.  * Simplest FFMpeg Streamer (Send 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.  * 本例子实现了推送本地视频至流媒体服务器（以RTMP为例）。
12.  * 是使用FFmpeg进行流媒体推送最简单的教程。
13.  *
14.  * This example stream local media files to streaming media
15.  * server (Use RTMP as example).
16.  * It's the simplest FFMpeg streamer.
17.  *
18.  */
19.
20. #include <stdio.h>
21.
22. #define __STDC_CONSTANT_MACROS
23.
24. #ifdef _WIN32
25. //Windows
```

```

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. int main(int argc, char* argv[])
47. {
48.     AVOutputFormat *ofmt = NULL;
49.     //输入对应一个AVFormatContext, 输出对应一个AVFormatContext
50.     // (Input AVFormatContext and Output AVFormatContext)
51.     AVFormatContext *ifmt_ctx = NULL, *ofmt_ctx = NULL;
52.     AVPacket pkt;
53.     const char *in_filename, *out_filename;
54.     int ret, i;
55.     int videoindex=-1;
56.     int frame_index=0;
57.     int64_t start_time=0;
58.     //in_filename = "cuc_ieschool.mov";
59.     //in_filename = "cuc_ieschool.mkv";
60.     //in_filename = "cuc_ieschool.ts";
61.     //in_filename = "cuc_ieschool.mp4";
62.     //in_filename = "cuc_ieschool.h264";
63.     in_filename = "cuc_ieschool.flv";//输入URL (Input file URL)
64.     //in_filename = "shanghai03_p.h264";
65.
66.     out_filename = "rtmp://localhost/publishlive/livestream";//输出 URL (Output URL) [RTMP]
67.     //out_filename = "rtp://233.233.233.233:6666";//输出 URL (Output URL) [UDP]
68.
69.     av_register_all();
70.     //Network
71.     avformat_network_init();
72.     //输入 (Input)
73.     if ((ret = avformat_open_input(&ifmt_ctx, in_filename, 0, 0)) < 0) {
74.         printf( "Could not open input file.");
75.         goto end;
76.     }
77.     if ((ret = avformat_find_stream_info(ifmt_ctx, 0)) < 0) {
78.         printf( "Failed to retrieve input stream information");
79.         goto end;
80.     }
81.
82.     for(i=0; i<ifmt_ctx->nb_streams; i++)
83.         if(ifmt_ctx->streams[i]->codec->codec_type==AVMEDIA_TYPE_VIDEO){
84.             videoindex=i;
85.             break;
86.         }
87.
88.     av_dump_format(ifmt_ctx, 0, in_filename, 0);
89.
90.     //输出 (Output)
91.
92.     avformat_alloc_output_context2(&ofmt_ctx, NULL, "flv", out_filename); //RTMP
93.     //avformat_alloc_output_context2(&ofmt_ctx, NULL, "mpegts", out_filename);//UDP
94.
95.     if (!ofmt_ctx) {
96.         printf( "Could not create output context\n");
97.         ret = AERROR_UNKNOWN;
98.         goto end;
99.     }
100.     ofmt = ofmt_ctx->oformat;
101.     for (i = 0; i < ifmt_ctx->nb_streams; i++) {
102.         //根据输入流创建输出流 (Create output AVStream according to input AVStream)
103.         AVStream *in_stream = ifmt_ctx->streams[i];
104.         AVStream *out_stream = avformat_new_stream(ofmt_ctx, in_stream->codec->codec);
105.         if (!out_stream) {
106.             printf( "Failed allocating output stream\n");
107.             ret = AERROR_UNKNOWN;
108.             goto end;
109.         }
110.         //复制AVCodecContext的设置 (Copy the settings of AVCodecContext)
111.         ret = avcodec_copy_context(out_stream->codec, in_stream->codec);
112.         if (ret < 0) {
113.             printf( "Failed to copy context from input to output stream codec context\n");
114.             goto end;
115.         }
116.         out_stream->codec->codec_tag = 0;

```

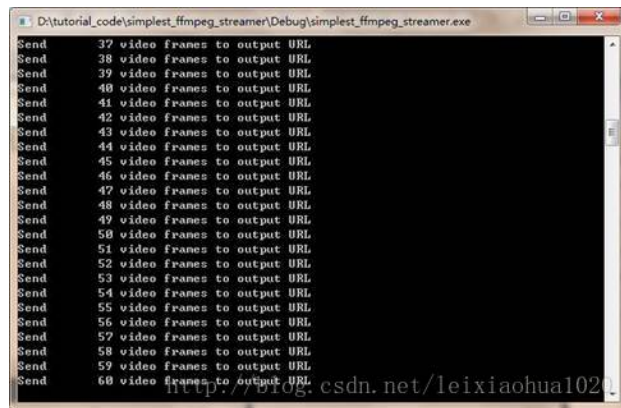
```

117.     if (ofmt_ctx->oformat->flags & AVFMT_GLOBALHEADER)
118.         out_stream->codec->flags |= CODEC_FLAG_GLOBAL_HEADER;
119.     }
120.     //Dump Format-----
121.     av_dump_format(ofmt_ctx, 0, out_filename, 1);
122.     //打开输出URL (Open output URL)
123.     if (!(ofmt->flags & AVFMT_NOFILE)) {
124.         ret = avio_open(&ofmt_ctx->pb, out_filename, AVIO_FLAG_WRITE);
125.         if (ret < 0) {
126.             printf( "Could not open output URL '%s'", out_filename);
127.             goto end;
128.         }
129.     }
130.     //写文件头 (Write file header)
131.     ret = avformat_write_header(ofmt_ctx, NULL);
132.     if (ret < 0) {
133.         printf( "Error occurred when opening output URL\n");
134.         goto end;
135.     }
136.
137.     start_time=av_gettime();
138.     while (1) {
139.         AVStream *in_stream, *out_stream;
140.         //获取一个AVPacket (Get an AVPacket)
141.         ret = av_read_frame(ifmt_ctx, &pkt);
142.         if (ret < 0)
143.             break;
144.         //FIX: No PTS (Example: Raw H.264)
145.         //Simple Write PTS
146.         if(pkt.pts==AV_NOPTS_VALUE){
147.             //Write PTS
148.             AVRational time_base1=ifmt_ctx->streams[videoindex]->time_base;
149.             //Duration between 2 frames (us)
150.             int64_t calc_duration=(double)AV_TIME_BASE/av_q2d(ifmt_ctx->streams[videoindex]->r_frame_rate);
151.             //Parameters
152.             pkt.pts=(double)(frame_index*calc_duration)/(double)(av_q2d(time_base1)*AV_TIME_BASE);
153.             pkt.dts=pkt.pts;
154.             pkt.duration=(double)calc_duration/(double)(av_q2d(time_base1)*AV_TIME_BASE);
155.         }
156.         //Important:Delay
157.         if(pkt.stream_index==videoindex){
158.             AVRational time_base=ifmt_ctx->streams[videoindex]->time_base;
159.             AVRational time_base_q={1,AV_TIME_BASE};
160.             int64_t pts_time = av_rescale_q(pkt.dts, time_base, time_base_q);
161.             int64_t now_time = av_gettime() - start_time;
162.             if (pts_time > now_time)
163.                 av_usleep(pts_time - now_time);
164.         }
165.
166.         in_stream = ifmt_ctx->streams[pkt.stream_index];
167.         out_stream = ofmt_ctx->streams[pkt.stream_index];
168.         /* copy packet */
169.         //转换PTS/DTS (Convert PTS/DTS)
170.         pkt.pts = av_rescale_q_rnd(pkt.pts, in_stream->time_base, out_stream->time_base, (AVRounding)
171. (AV_ROUND_NEAR_INF|AV_ROUND_PASS_MINMAX));
172.         pkt.dts = av_rescale_q_rnd(pkt.dts, in_stream->time_base, out_stream->time_base, (AVRounding)
173. (AV_ROUND_NEAR_INF|AV_ROUND_PASS_MINMAX));
174.         pkt.duration = av_rescale_q(pkt.duration, in_stream->time_base, out_stream->time_base);
175.         pkt.pos = -1;
176.         //Print to Screen
177.         if(pkt.stream_index==videoindex){
178.             printf("Send %8d video frames to output URL\n",frame_index);
179.             frame_index++;
180.         }
181.         //ret = av_write_frame(ofmt_ctx, &pkt);
182.         ret = av_interleaved_write_frame(ofmt_ctx, &pkt);
183.
184.         if (ret < 0) {
185.             printf( "Error muxing packet\n");
186.             break;
187.         }
188.         av_free_packet(&pkt);
189.
190.     }
191.     //写文件尾 (Write file trailer)
192.     av_write_trailer(ofmt_ctx);
193. end:
194.     avformat_close_input(&ifmt_ctx);
195.     /* close output */
196.     if (ofmt_ctx && !(ofmt->flags & AVFMT_NOFILE))
197.         avio_close(ofmt_ctx->pb);
198.     avformat_free_context(ofmt_ctx);
199.     if (ret < 0 && ret != AVERROR_EOF) {
200.         printf( "Error occurred.\n");
201.         return -1;
202.     }
203.     return 0;
204. }

```

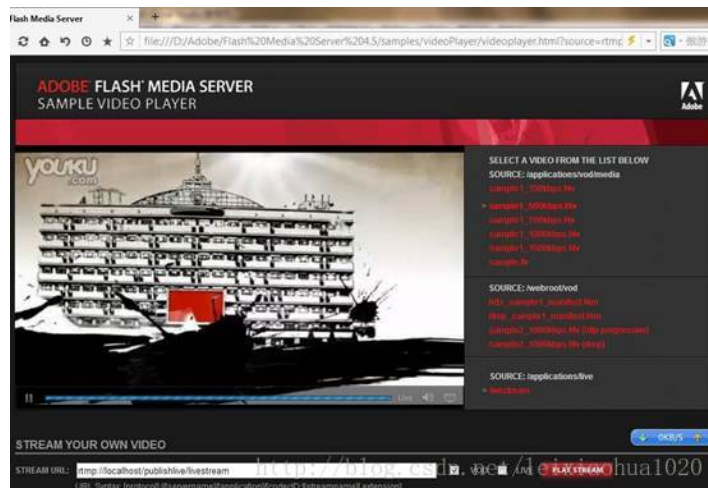
结果

程序开始运行后。截图如下所示。



可以通过网页播放器播放推送的直播流。

例如下图所示,使用Flash Media Server 的Samples文件夹下的videoPlayer播放直播流的截图如下图所示。(直播地址:rtmp://localhost/publishlive/livestream)



此外,也可以通过FFplay这样的客户端播放直播流。

下载

simplest ffmpeg streamer

项目主页

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

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

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

CSDN下载地址 :

<http://download.csdn.net/detail/leixiaohua1020/8005311>

更新-1.1 (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]
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_streamer.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。

```
[plain]
1.  g++ simplest_ffmpeg_streamer.cpp -g -o simplest_ffmpeg_streamer.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。

```
[plain]
1.  gcc simplest_ffmpeg_streamer.cpp -g -o simplest_ffmpeg_streamer.out \
2.  -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/leixiaohua1020/8445455>

SourceForge上已经更新。

更新-1.2 (2015.7.17) =====

增加了下列工程：

simplest_ffmpeg_receiver: 将流媒体数据保存成本地文件。

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

SourceForge、Github等上面已经更新。

文章标签：[ffmpeg](#) [推流](#) [rtmp](#) [flv](#) [流媒体](#)

个人分类：[FFMPEG](#) [libRTMP](#) [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