

## 原 最简单的基于librtmp的示例：发布（FLV通过RTMP发布）

2014年12月26日 00:09:33 阅读数：30571

最简单的基于libRTMP的示例系列文章列表：

[最简单的基于librtmp的示例：接收（RTMP保存为FLV）](#)

[最简单的基于librtmp的示例：发布（FLV通过RTMP发布）](#)

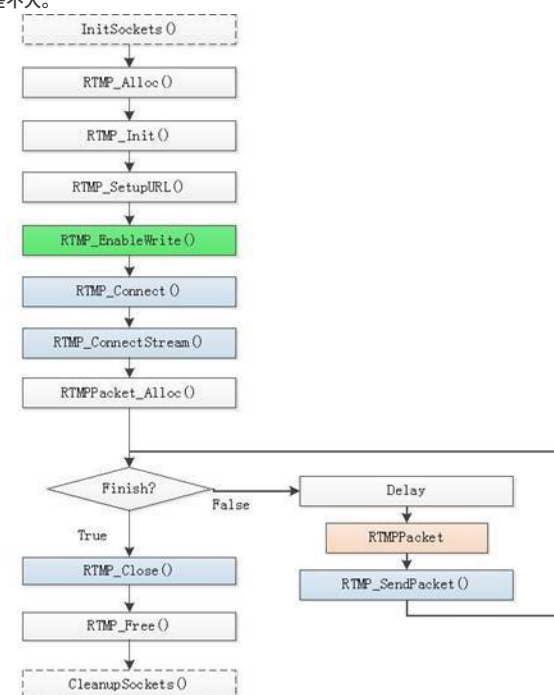
[最简单的基于librtmp的示例：发布H.264（H.264通过RTMP发布）](#)

本文记录一个基于libRTMP的发布流媒体的程序：Simplest libRTMP Send FLV。该程序可以将本地FLV文件发布到RTMP流媒体服务器。是最简单的基于libRTMP的流媒体发布示例。



### 流程图

使用librtmp发布RTMP流的可以使用两种API：RTMP\_SendPacket()和RTMP\_Write()。使用RTMP\_SendPacket()发布流的时候的函数执行流程图如下图所示。使用RTMP\_Write()发布流的时候的函数执行流程图相差不大。



Simplest Librtmp Send FLV  
雷霄骅 (Lei Xiaohua)  
Communication University of China / Digital TV Technology  
Email: leixiaohua1020@126.com [blog.csdn.net/leixiaohua1020](http://blog.csdn.net/leixiaohua1020)  
Website: <http://blog.csdn.net/leixiaohua1020>

流程图中关键函数的作用如下所列：

InitSockets()：初始化Socket

RTMP\_Alloc()：为结构体“RTMP”分配内存。

RTMP\_Init()：初始化结构体“RTMP”中的成员变量。

RTMP\_SetupURL()：设置输入的RTMP连接的URL。

RTMP\_EnableWrite()：发布流的时候必须要使用。如果不使用则代表接收流。

RTMP\_Connect()：建立RTMP连接，创建一个RTMP协议规范中的NetConnection。

RTMP\_ConnectStream(): 创建一个RTMP协议规范中的NetStream。

Delay: 发布流过程中的延时, 保证按正常播放速度发送数据。

RTMP\_SendPacket(): 发送一个RTMP数据RTMPPacket。

RTMP\_Close(): 关闭RTMP连接。

RTMP\_Free(): 释放结构体"RTMP"。

CleanupSockets(): 关闭Socket。

## 源代码

源代码中包含了使用两种API函数RTMP\_SendPacket()和RTMP\_Write()发布流媒体的源代码, 如下所示。

```
[cpp]    
1.  /**  
2.   * Simplest Librtmp Send FLV  
3.   *  
4.   * 雷霄骅, 张晖  
5.   * leixiaohua1020@126.com  
6.   * zhanghuicuc@gmail.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.  * 本程序用于将FLV格式的视音频文件使用RTMP推送至RTMP流媒体服务器。  
12.  * This program can send local flv file to net server as a rtmp live stream.  
13.  */  
14.  
15.  #include <stdio.h>  
16.  #include <stdlib.h>  
17.  #include <string.h>  
18.  #include <stdint.h>  
19.  #ifndef WIN32  
20.  #include <unistd.h>  
21.  #endif  
22.  
23.  
24.  #include "librtmp/rtmp_sys.h"  
25.  #include "librtmp/log.h"  
26.  
27.  #define HTON16(x)  ((x>>8&0xff)|(x<<8&0xff00))  
28.  #define HTON24(x)  ((x>>16&0xff)|(x<<16&0xff0000)|(x&0xff00))  
29.  #define HTON32(x)  ((x>>24&0xff)|(x>>8&0xff00)|\  
30.    (x<<8&0xff0000)|(x<<24&0xff000000))  
31.  #define HTONTIME(x) ((x>>16&0xff)|(x<<16&0xff0000)|(x&0xff00)|(x&0xff000000))  
32.  
33.  /*read 1 byte*/  
34.  int ReadU8(uint32_t *u8, FILE*fp){  
35.      if(fread(u8, 1, 1, fp)!=1)  
36.          return 0;  
37.      return 1;  
38.  }  
39.  /*read 2 byte*/  
40.  int ReadU16(uint32_t *u16, FILE*fp){  
41.      if(fread(u16, 2, 1, fp)!=1)  
42.          return 0;  
43.      *u16=HTON16(*u16);  
44.      return 1;  
45.  }  
46.  /*read 3 byte*/  
47.  int ReadU24(uint32_t *u24, FILE*fp){  
48.      if(fread(u24, 3, 1, fp)!=1)  
49.          return 0;  
50.      *u24=HTON24(*u24);  
51.      return 1;  
52.  }  
53.  /*read 4 byte*/  
54.  int ReadU32(uint32_t *u32, FILE*fp){  
55.      if(fread(u32, 4, 1, fp)!=1)  
56.          return 0;  
57.      *u32=HTON32(*u32);  
58.      return 1;  
59.  }  
60.  /*read 1 byte, and loopback 1 byte at once*/  
61.  int PeekU8(uint32_t *u8, FILE*fp){  
62.      if(fread(u8, 1, 1, fp)!=1)  
63.          return 0;  
64.      fseek(fp, -1, SEEK_CUR);  
65.      return 1;  
66.  }  
67.  /*read 4 byte and convert to time format*/  
68.  int ReadTime(uint32_t *utime, FILE*fp){  
69.      if(fread(utime, 4, 1, fp)!=1)  
70.          return 0;  
71.      *utime=HTONTIME(*utime);  
72.      return 1;  
73.  }  
74.  
75.  int InitSockets()  
76.  {
```

```

77.     WORD version;
78.     WSADATA wsaData;
79.     version=MAKWORD(2,2);
80.     return (WSAStartup(version, &wsaData) == 0);
81. }
82.
83. void CleanupSockets()
84. {
85.     WSACleanup();
86. }
87.
88. //Publish using RTMP SendPacket()
89. int publish_using_packet(){
90.     RTMP *rtmp=NULL;
91.     RTMPPacket *packet=NULL;
92.     uint32_t start_time=0;
93.     uint32_t now_time=0;
94.     //the timestamp of the previous frame
95.     long pre_frame_time=0;
96.     long lasttime=0;
97.     int bNextIsKey=1;
98.     uint32_t preTagsize=0;
99.
100.    //packet attributes
101.    uint32_t type=0;
102.    uint32_t datalength=0;
103.    uint32_t timestamp=0;
104.    uint32_t streamid=0;
105.
106.    FILE*fp=NULL;
107.    fp=fopen("cuc_ieschool.flv","rb");
108.    if (!fp){
109.        RTMP_LogPrintf("Open File Error.\n");
110.        CleanupSockets();
111.        return -1;
112.    }
113.
114.    /* set log level */
115.    //RTMP_LogLevel loglvl=RTMP_LOGDEBUG;
116.    //RTMP_LogSetLevel(loglvl);
117.
118.    if (!InitSockets()){
119.        RTMP_LogPrintf("Init Socket Err\n");
120.        return -1;
121.    }
122.
123.    rtmp=RTMP_Alloc();
124.    RTMP_Init(rtmp);
125.    //set connection timeout,default 30s
126.    rtmp->Link.timeout=5;
127.    if(!RTMP_SetupURL(rtmp,"rtmp://localhost/publishlive/livestream"))
128.    {
129.        RTMP_Log(RTMP_LOGERROR,"SetupURL Err\n");
130.        RTMP_Free(rtmp);
131.        CleanupSockets();
132.        return -1;
133.    }
134.
135.    //if unable,the AMF command would be 'play' instead of 'publish'
136.    RTMP_EnableWrite(rtmp);
137.
138.    if (!RTMP_Connect(rtmp,NULL)){
139.        RTMP_Log(RTMP_LOGERROR,"Connect Err\n");
140.        RTMP_Free(rtmp);
141.        CleanupSockets();
142.        return -1;
143.    }
144.
145.    if (!RTMP_ConnectStream(rtmp,0)){
146.        RTMP_Log(RTMP_LOGERROR,"ConnectStream Err\n");
147.        RTMP_Close(rtmp);
148.        RTMP_Free(rtmp);
149.        CleanupSockets();
150.        return -1;
151.    }
152.
153.    packet=(RTMPPacket*)malloc(sizeof(RTMPPacket));
154.    RTMPPacket_Alloc(packet,1024*64);
155.    RTMPPacket_Reset(packet);
156.
157.    packet->m_hasAbsTimestamp = 0;
158.    packet->m_nChannel = 0x04;
159.    packet->m_nInfoField2 = rtmp->m_stream_id;
160.
161.    RTMP_LogPrintf("Start to send data ...\n");
162.
163.    //jump over FLV Header
164.    fseek(fp,9,SEEK_SET);
165.    //jump over previousTagSizen
166.    fseek(fp,4,SEEK_CUR);
167.    start_time=RTMP_GetTime();

```

```

168.     while(1)
169.     {
170.         if(((now_time=RTMP_GetTime())-start_time)
171.            <(pre_frame_time)) && bNextIsKey){
172.             //wait for 1 sec if the send process is too fast
173.             //this mechanism is not very good,need some improvement
174.             if(pre_frame_time>lasttime){
175.                 RTMP_LogPrintf("TimeStamp:%8lu ms\n",pre_frame_time);
176.                 lasttime=pre_frame_time;
177.             }
178.             Sleep(1000);
179.             continue;
180.         }
181.
182.         //not quite the same as FLV spec
183.         if(!ReadU8(&type,fp))
184.             break;
185.         if(!ReadU24(&datalength,fp))
186.             break;
187.         if(!ReadTime(&tamp,fp))
188.             break;
189.         if(!ReadU24(&streamid,fp))
190.             break;
191.
192.         if (type!=0x08&&type!=0x09){
193.             //jump over non_audio and non_video frame,
194.             //jump over next previousTagSizen at the same time
195.             fseek(fp,datalength+4,SEEK_CUR);
196.             continue;
197.         }
198.
199.         if(fread(packet->m_body,1,datalength,fp)!=datalength)
200.             break;
201.
202.         packet->m_headerType = RTMP_PACKET_SIZE_LARGE;
203.         packet->m_nTimeStamp = timestamp;
204.         packet->m_packetType = type;
205.         packet->m_nBodySize = datalength;
206.         pre_frame_time=timestamp;
207.
208.         if (!RTMP_IsConnected(rtmp)){
209.             RTMP_Log(RTMP_LOGERROR,"rtmp is not connect\n");
210.             break;
211.         }
212.         if (!RTMP_SendPacket(rtmp,packet,0)){
213.             RTMP_Log(RTMP_LOGERROR,"Send Error\n");
214.             break;
215.         }
216.
217.         if(!ReadU32(&preTagsize,fp))
218.             break;
219.
220.         if(!PeekU8(&type,fp))
221.             break;
222.         if(type==0x09){
223.             if(fseek(fp,11,SEEK_CUR)!=0)
224.                 break;
225.             if(!PeekU8(&type,fp)){
226.                 break;
227.             }
228.             if(type==0x17)
229.                 bNextIsKey=1;
230.             else
231.                 bNextIsKey=0;
232.
233.             fseek(fp,-11,SEEK_CUR);
234.         }
235.     }
236.
237.     RTMP_LogPrintf("\nSend Data Over\n");
238.
239.     if(fp)
240.         fclose(fp);
241.
242.     if (rtmp!=NULL){
243.         RTMP_Close(rtmp);
244.         RTMP_Free(rtmp);
245.         rtmp=NULL;
246.     }
247.     if (packet!=NULL){
248.         RTMPPacket_Free(packet);
249.         free(packet);
250.         packet=NULL;
251.     }
252.
253.     CleanupSockets();
254.     return 0;
255. }
256.
257. //Publish using RTMP_Write()
258. int publish_using_write(){

```

```

259.     uint32_t start_time=0;
260.     uint32_t now_time=0;
261.     uint32_t pre_frame_time=0;
262.     uint32_t lasttime=0;
263.     int bNextIsKey=0;
264.     char* pFileBuf=NULL;
265.
266.     //read from tag header
267.     uint32_t type=0;
268.     uint32_t datalength=0;
269.     uint32_t timestamp=0;
270.
271.     RTMP *rtmp=NULL;
272.
273.     FILE*fp=NULL;
274.     fp=fopen("cuc_ieschool.flv","rb");
275.     if (!fp){
276.         RTMP_LogPrintf("Open File Error.\n");
277.         CleanupSockets();
278.         return -1;
279.     }
280.
281.     /* set log level */
282.     //RTMP_LogLevel loglvl=RTMP_LOGDEBUG;
283.     //RTMP_LogSetLevel(loglvl);
284.
285.     if (!InitSockets()){
286.         RTMP_LogPrintf("Init Socket Err\n");
287.         return -1;
288.     }
289.
290.     rtmp=RTMP_Alloc();
291.     RTMP_Init(rtmp);
292.     //set connection timeout,default 30s
293.     rtmp->Link.timeout=5;
294.     if(!RTMP_SetupURL(rtmp, "rtmp://localhost/publishlive/livestream"))
295.     {
296.         RTMP_Log(RTMP_LOGERROR, "SetupURL Err\n");
297.         RTMP_Free(rtmp);
298.         CleanupSockets();
299.         return -1;
300.     }
301.
302.     RTMP_EnableWrite(rtmp);
303.     //1hour
304.     RTMP_SetBufferMS(rtmp, 3600*1000);
305.     if (!RTMP_Connect(rtmp, NULL)){
306.         RTMP_Log(RTMP_LOGERROR, "Connect Err\n");
307.         RTMP_Free(rtmp);
308.         CleanupSockets();
309.         return -1;
310.     }
311.
312.     if (!RTMP_ConnectStream(rtmp, 0)){
313.         RTMP_Log(RTMP_LOGERROR, "ConnectStream Err\n");
314.         RTMP_Close(rtmp);
315.         RTMP_Free(rtmp);
316.         CleanupSockets();
317.         return -1;
318.     }
319.
320.     printf("Start to send data ...\n");
321.
322.     //jump over FLV Header
323.     fseek(fp, 9, SEEK_SET);
324.     //jump over previousTagSizen
325.     fseek(fp, 4, SEEK_CUR);
326.     start_time=RTMP_GetTime();
327.     while(1)
328.     {
329.         if((((now_time=RTMP_GetTime())-start_time)
330.             <(pre_frame_time)) && bNextIsKey){
331.             //wait for 1 sec if the send process is too fast
332.             //this mechanism is not very good, need some improvement
333.             if(pre_frame_time>lasttime){
334.                 RTMP_LogPrintf("TimeStamp:%8lu ms\n", pre_frame_time);
335.                 lasttime=pre_frame_time;
336.             }
337.             Sleep(1000);
338.             continue;
339.         }
340.
341.         //jump over type
342.         fseek(fp, 1, SEEK_CUR);
343.         if(!ReadU24(&datalength, fp))
344.             break;
345.         if(!ReadTime(&tamp, fp))
346.             break;
347.         //jump back
348.         fseek(fp, -8, SEEK_CUR);
349.

```

```

350.         pFileBuf=(char*)malloc(11+dataLength+4);
351.         memset(pFileBuf,0,11+dataLength+4);
352.         if(fread(pFileBuf,1,11+dataLength+4,fp)!=(11+dataLength+4))
353.             break;
354.
355.         pre_frame_time=timestamp;
356.
357.         if (!RTMP_IsConnected(rtmp)){
358.             RTMP_Log(RTMP_LOGERROR,"rtmp is not connect\n");
359.             break;
360.         }
361.         if (!RTMP_Write(rtmp,pFileBuf,11+dataLength+4)){
362.             RTMP_Log(RTMP_LOGERROR,"Rtmp Write Error\n");
363.             break;
364.         }
365.
366.         free(pFileBuf);
367.         pFileBuf=NULL;
368.
369.         if(!PeekU8(&type,fp))
370.             break;
371.         if(type==0x09){
372.             if(fseek(fp,11,SEEK_CUR)!=0)
373.                 break;
374.             if(!PeekU8(&type,fp)){
375.                 break;
376.             }
377.             if(type==0x17)
378.                 bNextIsKey=1;
379.             else
380.                 bNextIsKey=0;
381.             fseek(fp,-11,SEEK_CUR);
382.         }
383.     }
384.
385.     RTMP_LogPrintf("\nSend Data Over\n");
386.
387.     if(fp)
388.         fclose(fp);
389.
390.     if (rtmp!=NULL){
391.         RTMP_Close(rtmp);
392.         RTMP_Free(rtmp);
393.         rtmp=NULL;
394.     }
395.
396.     if(pFileBuf){
397.         free(pFileBuf);
398.         pFileBuf=NULL;
399.     }
400.
401.     CleanupSockets();
402.     return 0;
403. }
404.
405. int main(int argc, char* argv[]){
406.     //2 Methods:
407.     publish_using_packet();
408.     //publish_using_write();
409.     return 0;
410. }

```

## 运行结果

程序运行后，会将“cuc\_ieschool.flv”文件以直播流的形式发布到“rtmp://localhost/publishlive/livestream”的URL。修改文件名称和RTMP的URL可以实现将任意flv文件发布到任意RTMP的URL。

## 下载

Simplest LibRTMP Example

项目主页

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

Github：[https://github.com/leixiaohua1020/simplest\\_librtmp\\_example](https://github.com/leixiaohua1020/simplest_librtmp_example)

开源中国：[http://git.oschina.net/leixiaohua1020/simplest\\_librtmp\\_example](http://git.oschina.net/leixiaohua1020/simplest_librtmp_example)

CSDN下载：<http://download.csdn.net/detail/leixiaohua1020/8291757>

本工程包含了LibRTMP的使用示例，包含如下子工程：

simplest\_librtmp\_receive: 接收RTMP流媒体并在本地保存成FLV格式的文件。

simplest\_librtmp\_send\_flv: 将FLV格式的视音频文件使用RTMP推送至RTMP流媒体服务器。

simplest\_librtmp\_send264: 将内存中的H.264数据推送至RTMP流媒体服务器。

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/42104945>

文章标签：flv RTMP 流媒体 libRTMP

个人分类：我的开源项目 libRTMP

此PDF由spygg生成,请尊重原作者版权!!!

我的邮箱:liushidc@163.com