

原 RTMPdump 源代码分析 1：main()函数

2013年10月22日 20:45:36 阅读数：26245

=====

RTMPdump(libRTMP) 源代码分析系列文章：

[RTMPdump 源代码分析 1：main\(\)函数](#)

[RTMPDump \(libRTMP\) 源代码分析2：解析RTMP地址——RTMP_ParseURL\(\)](#)

[RTMPdump \(libRTMP\) 源代码分析3：AMF编码](#)

[RTMPdump \(libRTMP\) 源代码分析4：连接第一步——握手 \(HandShake\)](#)

[RTMPdump \(libRTMP\) 源代码分析5：建立一个流媒体连接 \(NetConnection部分\)](#)

[RTMPdump \(libRTMP\) 源代码分析6：建立一个流媒体连接 \(NetStream部分 1\)](#)

[RTMPdump \(libRTMP\) 源代码分析7：建立一个流媒体连接 \(NetStream部分 2\)](#)

[RTMPdump \(libRTMP\) 源代码分析8：发送消息 \(Message\)](#)

[RTMPdump \(libRTMP\) 源代码分析9：接收消息 \(Message\) \(接收视音频数据\)](#)

[RTMPdump \(libRTMP\) 源代码分析10：处理各种消息 \(Message\)](#)

=====

rtmpdump 是一个用来处理 RTMP 流媒体的工具包,支持 rtmp://, rtmpt://, rtmpe://, rtmpte://, and rtmps:// 等。之前在学习RTMP协议的时候,发现没有讲它源代码的,只好自己分析,现在打算把自己学习的成果写出来,可能结果不一定都对,先暂且记录一下。


函数调用结构图

RTMPDump (libRTMP)的整体的函数调用结构图如下图所示。


[单击查看大图](#)

详细分析

使用RTMPdump下载一个流媒体的大致流程是这样的：

```
[cpp]    
1. RTMP_Init();//初始化结构体  
2. InitSockets();//初始化Socket  
3. RTMP_ParseURL();//解析输入URL  
4. RTMP_SetupStream();//一些设置  
5. fopen();//打开文件,准备写入  
6. RTMP_Connect();//建立NetConnection  
7. RTMP_ConnectStream();//建立NetStream  
8. Download();//下载函数  
9. RTMP_Close();//关闭连接  
10. fclose();//关闭文件  
11. CleanupSockets();//清理Socket
```

其中Download()主要是使用RTMP_Read()进行下载的。

注：可以参考：[RTMP流媒体播放过程](#)

下面贴上自己注释的RTMPDump源代码。注意以下几点：

- 1.此RTMPDump已经被移植进VC 2010 的 MFC的工程,所以main()函数已经被改名为rtmpdump(),而且参数也改了,传进来一个MFC窗口的句柄。不过功能没怎么改 (控制台程序移植到MFC以后, main()就不是程序的入口了,所以main()名字改成什么是无所谓的)
- 2.里面有很多提取信息的代码形如：rtmp.dlg->AppendCInfo("开始初始化Socket...");这些代码是我为了获取RTMP信息而自己加的,并不影响程序的执行。

[cpp]  

```

1. int rtmpdump(LPVOID lpParam,int argc,char **argv)
2. {
3.
4.     extern char *optarg;
5.     //一定要设置, 否则只能运行一次
6.     extern int optind;
7.     optind=0;
8.     int nStatus = RD_SUCCESS;
9.     double percent = 0;
10.    double duration = 0.0;
11.
12.    int nSkipKeyFrames = DEF_SKIPFRM; // skip this number of keyframes when resuming
13.
14.    int bOverrideBufferTime = FALSE; // if the user specifies a buffer time override this is true
15.    int bStdoutMode = TRUE; // if true print the stream directly to stdout, messages go to stderr
16.    int bResume = FALSE; // true in resume mode
17.    uint32_t dSeek = 0; // seek position in resume mode, 0 otherwise
18.    uint32_t bufferTime = DEF_BUFTIME;
19.
20.    // meta header and initial frame for the resume mode (they are read from the file and compared with
21.    // the stream we are trying to continue
22.    char *metaHeader = 0;
23.    uint32_t nMetaHeaderSize = 0;
24.
25.    // video keyframe for matching
26.    char *initialFrame = 0;
27.    uint32_t nInitialFrameSize = 0;
28.    int initialFrameType = 0; // tye: audio or video
29.
30.    AVval hostname = { 0, 0 };
31.    AVval playpath = { 0, 0 };
32.    AVval subscribepath = { 0, 0 };
33.    int port = -1;
34.    int protocol = RTMP_PROTOCOL_UNDEFINED;
35.    int retries = 0;
36.    int bLiveStream = FALSE; // 是直播流吗? then we can't seek/resume
37.    int bHashes = FALSE; // display byte counters not hashes by default
38.
39.    long int timeout = DEF_TIMEOUT; // timeout connection after 120 seconds
40.    uint32_t dStartOffset = 0; // 非直播流搜寻点seek position in non-live mode
41.    uint32_t dStopOffset = 0;
42.    RTMP rtmp = { 0 };
43.
44.    AVval swfUrl = { 0, 0 };
45.    AVval tcUrl = { 0, 0 };
46.    AVval pageUrl = { 0, 0 };
47.    AVval app = { 0, 0 };
48.    AVval auth = { 0, 0 };
49.    AVval swfHash = { 0, 0 };
50.    uint32_t swfSize = 0;
51.    AVval flashVer = { 0, 0 };
52.    AVval sockshost = { 0, 0 };
53.
54.    #ifdef CRYPTO
55.        int swfAge = 30; /* 30 days for SWF cache by default */
56.        int swfVfy = 0;
57.        unsigned char hash[RTMP_SWF_HASHLEN];
58.    #endif
59.
60.    char *flvFile = 0;
61.
62.    signal(SIGINT, sigIntHandler);
63.    signal(SIGTERM, sigIntHandler);
64.    #ifndef WIN32
65.        signal(SIGHUP, sigIntHandler);
66.        signal(SIGPIPE, sigIntHandler);
67.        signal(SIGQUIT, sigIntHandler);
68.    #endif
69.
70.    RTMP_debuglevel = RTMP_LOGINFO;
71.
72.    //首先搜寻“--quiet”选项
73.    int index = 0;
74.    while (index < argc)
75.    {
76.        if (strcmp(argv[index], "--quiet") == 0
77.        || strcmp(argv[index], "-q") == 0)
78.            RTMP_debuglevel = RTMP_LOGCRIT;
79.        index++;
80.    }
81.
82.    #define RTMPDUMP_VERSION "1.0"
83.    RTMP_LogPrintf("RTMP流媒体下载 %s\n", RTMPDUMP_VERSION);
84.    RTMP_LogPrintf
85.    ("2012 雷霄骅 中国传媒大学/信息工程学院/通信与信息系统/数字电视技术\n");
86.    //RTMP_LogPrintf("输入 -h 获取命令选项\n");
87.    RTMP_Init(&rtmp);
88.    //句柄-----
89.    rtmp.dlg=(CSpecialPRTMPDlG *)lpParam;
90.    //-----
91.    rtmp.dlg->AppendCInfo("开始初始化Socket ");

```

```

91.     rtmp.dlg->AppendCInfo( "初始化Socket失败!",
92.     //-----
93.     if (!InitSockets())
94.     {
95.         //-----
96.         rtmp.dlg->AppendCInfo("初始化Socket失败!");
97.         //-----
98.         RTMP_Log(RTMP_LOGERROR,
99.             "Couldn't load sockets support on your platform, exiting!");
100.        return RD_FAILED;
101.    }
102.    //-----
103.    rtmp.dlg->AppendCInfo("成功初始化Socket");
104.    //-----
105.    /* sleep(30); */
106.
107.
108.
109.    int opt;
110.    /* struct option longopts[] = {
111.        {"help", 0, NULL, 'h'},
112.        {"host", 1, NULL, 'n'},
113.        {"port", 1, NULL, 'c'},
114.        {"socks", 1, NULL, 'S'},
115.        {"protocol", 1, NULL, 'l'},
116.        {"playpath", 1, NULL, 'y'},
117.        {"playlist", 0, NULL, 'Y'},
118.        {"rtmp", 1, NULL, 'r'},
119.        {"swfUrl", 1, NULL, 's'},
120.        {"tcUrl", 1, NULL, 't'},
121.        {"pageUrl", 1, NULL, 'p'},
122.        {"app", 1, NULL, 'a'},
123.        {"auth", 1, NULL, 'u'},
124.        {"conn", 1, NULL, 'C'},
125.    #ifdef CRYPTO
126.        {"swfhash", 1, NULL, 'w'},
127.        {"swfsize", 1, NULL, 'x'},
128.        {"swfVfy", 1, NULL, 'W'},
129.        {"swfAge", 1, NULL, 'X'},
130.    #endif
131.        {"flashVer", 1, NULL, 'f'},
132.        {"live", 0, NULL, 'v'},
133.        {"flv", 1, NULL, 'o'},
134.        {"resume", 0, NULL, 'e'},
135.        {"timeout", 1, NULL, 'm'},
136.        {"buffer", 1, NULL, 'b'},
137.        {"skip", 1, NULL, 'k'},
138.        {"subscribe", 1, NULL, 'd'},
139.        {"start", 1, NULL, 'A'},
140.        {"stop", 1, NULL, 'B'},
141.        {"token", 1, NULL, 'T'},
142.        {"hashes", 0, NULL, '#'},
143.        {"debug", 0, NULL, 'z'},
144.        {"quiet", 0, NULL, 'q'},
145.        {"verbose", 0, NULL, 'V'},
146.        {0, 0, 0, 0}
147.    };*/
148.    //分析命令行参数, 注意用法。
149.    //选项都是一个字母, 后面有冒号的代表该选项还有相关参数
150.    //一直循环直到获取所有的opt
151.    while ((opt =
152.        getopt(/*_long*/(argc, argv,
153.            "hVveqzr:s:t:p:a:b:f:o:u:C:n:c:l:y:Ym:k:d:A:B:T:w:x:W:X:S:#"/*,
154.            longopts, NULL*/)) != -1)
155.    {
156.        //不同的选项做不同的处理
157.        switch (opt)
158.        {
159.            case 'h':
160.                usage(argv[0]);
161.                return RD_SUCCESS;
162.            #ifdef CRYPTO
163.            case 'w':
164.                {
165.                    int res = hex2bin(optarg, &swfHash.av_val);
166.                    if (res != RTMP_SWF_HASHLEN)
167.                    {
168.                        swfHash.av_val = NULL;
169.                        RTMP_Log(RTMP_LOGWARNING,
170.                            "Couldn't parse swf hash hex string, not hexstring or not %d bytes, ignoring!", RTMP_SWF_HASHLEN);
171.                    }
172.                    swfHash.av_len = RTMP_SWF_HASHLEN;
173.                    break;
174.                }
175.            case 'x':
176.                {
177.                    int size = atoi(optarg);
178.                    if (size <= 0)
179.                    {
180.                        RTMP_Log(RTMP_LOGERROR, "SWF Size must be at least 1, ignoring\n");
181.                    }
182.                    else

```

```

183.     {
184.         swfSize = size;
185.     }
186.     break;
187. }
188. case 'W':
189.     STR2AVAL(swfUrl, optarg);
190.     swfVfy = 1;
191.     break;
192. case 'X':
193.     {
194.         int num = atoi(optarg);
195.         if (num < 0)
196.         {
197.             RTMP_Log(RTMP_LOGERROR, "SWF Age must be non-negative, ignoring\n");
198.         }
199.         else
200.         {
201.             swfAge = num;
202.         }
203.     }
204.     break;
205. #endif
206. case 'k':
207.     nSkipKeyFrames = atoi(optarg);
208.     if (nSkipKeyFrames < 0)
209.     {
210.         RTMP_Log(RTMP_LOGERROR,
211.             "Number of keyframes skipped must be greater or equal zero, using zero!");
212.         nSkipKeyFrames = 0;
213.     }
214.     else
215.     {
216.         RTMP_Log(RTMP_LOGDEBUG, "Number of skipped key frames for resume: %d",
217.             nSkipKeyFrames);
218.     }
219.     break;
220. case 'b':
221.     {
222.         int32_t bt = atol(optarg);
223.         if (bt < 0)
224.         {
225.             RTMP_Log(RTMP_LOGERROR,
226.                 "Buffer time must be greater than zero, ignoring the specified value %d!",
227.                 bt);
228.         }
229.         else
230.         {
231.             bufferTime = bt;
232.             bOverrideBufferTime = TRUE;
233.         }
234.         break;
235.     }
236. //直播流
237. case 'v':
238.     //-----
239.     rtmp.dlg->AppendCInfo("该RTMP的URL是一个直播流");
240.     //-----
241.     bLiveStream = TRUE;    // no seeking or resuming possible!
242.     break;
243. case 'd':
244.     STR2AVAL(subscribepath, optarg);
245.     break;
246. case 'n':
247.     STR2AVAL(hostname, optarg);
248.     break;
249. case 'c':
250.     port = atoi(optarg);
251.     break;
252. case 'l':
253.     protocol = atoi(optarg);
254.     if (protocol < RTMP_PROTOCOL_RTMP || protocol > RTMP_PROTOCOL_RTMPPTS)
255.     {
256.         RTMP_Log(RTMP_LOGERROR, "Unknown protocol specified: %d", protocol);
257.         return RD_FAILED;
258.     }
259.     break;
260. case 'y':
261.     STR2AVAL(playpath, optarg);
262.     break;
263. case 'Y':
264.     RTMP_SetOpt(&rtmp, &av_playlist, (AVal *)&av_true);
265.     break;
266. //路径参数-r
267. case 'r':
268.     {
269.         AVal parsedHost, parsedApp, parsedPlaypath;
270.         unsigned int parsedPort = 0;
271.         int parsedProtocol = RTMP_PROTOCOL_UNDEFINED;
272.         //解析URL。注optarg指向参数 (URL)
273.         RTMP_LogPrintf("RTMP URL : %s\n", optarg);

```

```

274. //-----
275. rtmp.dlg->AppendCInfo("解析RTMP的URL...");
276. //-----
277. if (!RTMP_ParseURL
278. (optarg, &parsedProtocol, &parsedHost, &parsedPort,
279. &parsedPlaypath, &parsedApp))
280. {
281. //-----
282. rtmp.dlg->AppendCInfo("解析RTMP的URL失败!");
283. //-----
284. RTMP_Log(RTMP_LOGWARNING, "无法解析 url (%s)!",
285. optarg);
286. }
287. else
288. {
289. //-----
290. rtmp.dlg->AppendCInfo("解析RTMP的URL成功");
291. //-----
292. //把解析出来的数据赋值
293. if (!hostname.av_len)
294. hostname = parsedHost;
295. if (port == -1)
296. port = parsedPort;
297. if (playpath.av_len == 0 && parsedPlaypath.av_len)
298. {
299. playpath = parsedPlaypath;
300. }
301. if (protocol == RTMP_PROTOCOL_UNDEFINED)
302. protocol = parsedProtocol;
303. if (app.av_len == 0 && parsedApp.av_len)
304. {
305. app = parsedApp;
306. }
307. }
308. break;
309. }
310. case 's':
311. STR2AVAL(swfUrl, optarg);
312. break;
313. case 't':
314. STR2AVAL(tcUrl, optarg);
315. break;
316. case 'p':
317. STR2AVAL(pageUrl, optarg);
318. break;
319. case 'a':
320. STR2AVAL(app, optarg);
321. break;
322. case 'f':
323. STR2AVAL(flashVer, optarg);
324. break;
325. //指定输出文件
326. case 'o':
327. flvFile = optarg;
328. if (strcmp(flvFile, "-"))
329. bStdoutMode = FALSE;
330.
331. break;
332. case 'e':
333. bResume = TRUE;
334. break;
335. case 'u':
336. STR2AVAL(auth, optarg);
337. break;
338. case 'C': {
339. AVa1 av;
340. STR2AVAL(av, optarg);
341. if (!RTMP_SetOpt(&rtmp, &av_conn, &av))
342. {
343. RTMP_Log(RTMP_LOGERROR, "Invalid AMF parameter: %s", optarg);
344. return RD_FAILED;
345. }
346. }
347. break;
348. case 'm':
349. timeout = atoi(optarg);
350. break;
351. case 'A':
352. dStartOffset = (int) (atof(optarg) * 1000.0);
353. break;
354. case 'B':
355. dStopOffset = (int) (atof(optarg) * 1000.0);
356. break;
357. case 'T': {
358. AVa1 token;
359. STR2AVAL(token, optarg);
360. RTMP_SetOpt(&rtmp, &av_token, &token);
361. }
362. break;
363. case '#':
364. bHashes = TRUE;

```

```

365.         break;
366.     case 'q':
367.         RTMP_debugLevel = RTMP_LOGCRIT;
368.         break;
369.     case 'V':
370.         RTMP_debugLevel = RTMP_LOGDEBUG;
371.         break;
372.     case 'z':
373.         RTMP_debugLevel = RTMP_LOGALL;
374.         break;
375.     case 'S':
376.         STR2AVAL(sockshost, optarg);
377.         break;
378.     default:
379.         RTMP_LogPrintf("unknown option: %c\n", opt);
380.         usage(argv[0]);
381.         return RD_FAILED;
382.         break;
383.     }
384. }
385.
386. if (!hostname.av_len)
387. {
388.     RTMP_Log(RTMP_LOGERROR,
389.         "您必须指定 主机名(hostname) (--host) 或 url (-r \"rtmp://host[:port]/playpath\") 包含 a hostname");
390.     return RD_FAILED;
391. }
392. if (playpath.av_len == 0)
393. {
394.     RTMP_Log(RTMP_LOGERROR,
395.         "您必须指定 播放路径(playpath) (--playpath) 或 url (-r \"rtmp://host[:port]/playpath\") 包含 a playpath");
396.     return RD_FAILED;
397. }
398.
399. if (protocol == RTMP_PROTOCOL_UNDEFINED)
400. {
401.     RTMP_Log(RTMP_LOGWARNING,
402.         "您没有指定 协议(protocol) (--protocol) 或 rtmp url (-r), 默认协议 RTMP");
403.     protocol = RTMP_PROTOCOL_RTMP;
404. }
405. if (port == -1)
406. {
407.     RTMP_Log(RTMP_LOGWARNING,
408.         "您没有指定 端口(port) (--port) 或 rtmp url (-r), 默认端口 1935");
409.     port = 0;
410. }
411. if (port == 0)
412. {
413.     if (protocol & RTMP_FEATURE_SSL)
414.         port = 443;
415.     else if (protocol & RTMP_FEATURE_HTTP)
416.         port = 80;
417.     else
418.         port = 1935;
419. }
420.
421. if (flvFile == 0)
422. {
423.     RTMP_Log(RTMP_LOGWARNING,
424.         "请指定一个输出文件 (-o filename), using stdout");
425.     bStdoutMode = TRUE;
426. }
427.
428. if (bStdoutMode && bResume)
429. {
430.     RTMP_Log(RTMP_LOGWARNING,
431.         "Can't resume in stdout mode, ignoring --resume option");
432.     bResume = FALSE;
433. }
434.
435. if (bLiveStream && bResume)
436. {
437.     RTMP_Log(RTMP_LOGWARNING, "Can't resume live stream, ignoring --resume option");
438.     bResume = FALSE;
439. }
440.
441. #ifdef CRYPTO
442. if (swfVfy)
443. {
444.     if (RTMP_HashSWF(swfUrl.av_val, (unsigned int *)&swfSize, hash, swfAge) == 0)
445.     {
446.         swfHash.av_val = (char *)hash;
447.         swfHash.av_len = RTMP_SWF_HASHLEN;
448.     }
449. }
450.
451. if (swfHash.av_len == 0 && swfSize > 0)
452. {
453.     RTMP_Log(RTMP_LOGWARNING,
454.         "Ignoring SWF size, supply also the hash with --swfhash");
455.     swfSize = 0;

```

```

456.     }
457.
458.     if (swfHash.av_len != 0 && swfSize == 0)
459.     {
460.         RTMP_Log(RTMP_LOGWARNING,
461.             "Ignoring SWF hash, supply also the swf size with --swfsize");
462.         swfHash.av_len = 0;
463.         swfHash.av_val = NULL;
464.     }
465. #endif
466.
467.     if (tcUrl.av_len == 0)
468.     {
469.         char str[512] = { 0 };
470.
471.         tcUrl.av_len = snprintf(str, 511, "%s://%s:%d/%s",
472.             RTMPProtocolStringsLower[protocol], hostname.av_len,
473.             hostname.av_val, port, app.av_len, app.av_val);
474.         tcUrl.av_val = (char *) malloc(tcUrl.av_len + 1);
475.         strcpy(tcUrl.av_val, str);
476.     }
477.
478.     int first = 1;
479.
480.     // User defined seek offset
481.     if (dStartOffset > 0)
482.     {
483.         //直播流
484.         if (bLiveStream)
485.         {
486.             RTMP_Log(RTMP_LOGWARNING,
487.                 "Can't seek in a live stream, ignoring --start option");
488.             dStartOffset = 0;
489.         }
490.     }
491.     //-----
492.     rtmp.dlg->AppendCInfo("开始初始化RTMP连接的参数...");
493.     //-----
494.     //设置
495.     RTMP_SetupStream(&rtmp, protocol, &hostname, port, &sockshost, &playpath,
496.         &tcUrl, &swfUrl, &pageUrl, &app, &auth, &swfHash, swfSize,
497.         &flashVer, &subscribePath, dSeek, dStopOffset, bLiveStream, timeout);
498.     //此处设置参数-----
499.     rtmp.dlg->AppendCInfo("成功初始化RTMP连接的参数");
500.     //-----
501.     char *temp=(char *)malloc(MAX_URL_LENGTH);
502.
503.     memcpy(temp,rtmp.Link.hostname.av_val,rtmp.Link.hostname.av_len);
504.     temp[rtmp.Link.hostname.av_len]='\0';
505.     rtmp.dlg->AppendB_R_L_Info("主机名",temp);
506.
507.     itoa(rtmp.Link.port,temp,10);
508.     rtmp.dlg->AppendB_R_L_Info("端口号",temp);
509.
510.     memcpy(temp,rtmp.Link.app.av_val,rtmp.Link.app.av_len);
511.     temp[rtmp.Link.app.av_len]='\0';
512.     rtmp.dlg->AppendB_R_L_Info("应用程序",temp);
513.
514.     memcpy(temp,rtmp.Link.playpath.av_val,rtmp.Link.playpath.av_len);
515.     temp[rtmp.Link.playpath.av_len]='\0';
516.     rtmp.dlg->AppendB_R_L_Info("路径",temp);
517.
518.
519.     //-----
520.
521.     /* Try to keep the stream moving if it pauses on us */
522.     if (!bLiveStream && !(protocol & RTMP_FEATURE_HTTP))
523.         rtmp.Link.lFlags |= RTMP_LF_BUFEX;
524.
525.     off_t size = 0;
526.
527.     // ok,我们必须获得timestamp of the last keyframe (only keyframes are seekable) / last audio frame (audio only streams)
528.     if (bResume)
529.     {
530.         //打开文件, 输出的文件(Resume)
531.         nStatus =
532.             OpenResumeFile(flvFile, &file, &size, &metaHeader, &nMetaHeaderSize,
533.                 &duration);
534.         if (nStatus == RD_FAILED)
535.             goto clean;
536.
537.         if (!file)
538.         {
539.             // file does not exist, so go back into normal mode
540.             bResume = FALSE; // we are back in fresh file mode (otherwise finalizing file won't be done)
541.         }
542.         else
543.         {
544.             //获取最后一个关键帧
545.             nStatus = GetLastKeyframe(file, nSkipKeyFrames,
546.                 &dSeek, &initialFrame,

```

```

547.         &initialFrameType, &initialFrameSize);
548.     if (nStatus == RD_FAILED)
549.     {
550.         RTMP_Log(RTMP_LOGDEBUG, "Failed to get last keyframe.");
551.         goto clean;
552.     }
553.
554.     if (dSeek == 0)
555.     {
556.         RTMP_Log(RTMP_LOGDEBUG,
557.             "Last keyframe is first frame in stream, switching from resume to normal mode!");
558.         bResume = FALSE;
559.     }
560. }
561. }
562. //如果输出文件不存在
563. if (!file)
564. {
565.     if (bStdoutMode)
566.     {
567.         //直接输出到stdout
568.         file = stdout;
569.         SET_BINMODE(file);
570.     }
571.     else
572.     {
573.         //打开一个文件
574.         //w+b 读写打开或建立一个二进制文件，允许读和写。
575.         //-----
576.         rtmp.dlg->AppendCInfo("创建输出文件...");
577.         //-----
578.         file = fopen(flvFile, "w+b");
579.         if (file == 0)
580.         {
581.             //-----
582.             rtmp.dlg->AppendCInfo("创建输出文件失败!");
583.             //-----
584.             RTMP_LogPrintf("Failed to open file! %s\n", flvFile);
585.             return RD_FAILED;
586.         }
587.         rtmp.dlg->AppendCInfo("成功创建输出文件");
588.     }
589. }
590.
591. #ifdef _DEBUG
592.     netstackdump = fopen("netstackdump", "wb");
593.     netstackdump_read = fopen("netstackdump_read", "wb");
594. #endif
595.
596. while (!RTMP_ctrlC)
597. {
598.     RTMP_Log(RTMP_LOGDEBUG, "Setting buffer time to: %dms", bufferTime);
599.     //设置Buffer时间
600.     //-----
601.     rtmp.dlg->AppendCInfo("设置缓冲(Buffer)的时间");
602.     //-----
603.     RTMP_SetBufferMS(&rtmp, bufferTime);
604.     //第一次执行
605.     if (first)
606.     {
607.         first = 0;
608.         RTMP_LogPrintf("开始建立连接!\n");
609.         //-----
610.         rtmp.dlg->AppendCInfo("开始建立连接 (NetConnection) ...");
611.         //-----
612.         //建立连接(Connect)
613.         if (!RTMP_Connect(&rtmp, NULL))
614.         {
615.             //-----
616.             rtmp.dlg->AppendCInfo("建立连接 (NetConnection) 失败!");
617.             //-----
618.             nStatus = RD_FAILED;
619.             break;
620.         }
621.         //-----
622.         rtmp.dlg->AppendCInfo("成功建立连接 (NetConnection) ");
623.         //-----
624.         //RTMP_Log(RTMP_LOGINFO, "已链接...");
625.
626.         // User defined seek offset
627.         if (dStartOffset > 0)
628.         {
629.             // Don't need the start offset if resuming an existing file
630.             if (bResume)
631.             {
632.                 RTMP_Log(RTMP_LOGWARNING,
633.                     "Can't seek a resumed stream, ignoring --start option");
634.                 dStartOffset = 0;
635.             }
636.             else
637.             {
638.                 dSeek = dStartOffset;

```



```

638.         dSeek = dStartOffset;
639.     }
640. }
641.
642. // Calculate the length of the stream to still play
643. if (dStopOffset > 0)
644. {
645.     // Quit if start seek is past required stop offset
646.     if (dStopOffset <= dSeek)
647.     {
648.         RTMP_LogPrintf("Already Completed\n");
649.         nStatus = RD_SUCCESS;
650.         break;
651.     }
652. }
653. //创建流(Stream) (发送connect命令消息后处理传来的数据)
654. itoa(rtmp.m_inChunkSize,temp,10);
655. rtmp.dlg->AppendB_R_Info("输入Chunk大小",temp);
656. itoa(rtmp.m_outChunkSize,temp,10);
657. rtmp.dlg->AppendB_R_Info("输出Chunk大小",temp);
658. itoa(rtmp.m_stream_id,temp,10);
659. rtmp.dlg->AppendB_R_Info("Stream ID",temp);
660. itoa(rtmp.m_nBufferMS,temp,10);
661. rtmp.dlg->AppendB_R_Info("Buffer时长 (ms) ",temp);
662. itoa(rtmp.m_nServerBW,temp,10);
663. rtmp.dlg->AppendB_R_Info("ServerBW",temp);
664. itoa(rtmp.m_nClientBW,temp,10);
665. rtmp.dlg->AppendB_R_Info("ClientBW",temp);
666. itoa((int)rtmp.m_fEncoding,temp,10);
667. rtmp.dlg->AppendB_R_Info("命令消息编码方法",temp);
668. itoa((int)rtmp.m_fDuration,temp,10);
669. rtmp.dlg->AppendB_R_Info("时长 (s) ",temp);
670.
671. rtmp.dlg->ShowBInfo();
672. free(temp);
673. //-----
674. rtmp.dlg->AppendCInfo("开始建立网络流 (NetStream) ");
675. //-----
676. if (!RTMP_ConnectStream(&rtmp, dSeek))
677. {
678.     //-----
679.     rtmp.dlg->AppendCInfo("建立网络流 (NetStream) 失败!");
680.     //-----
681.     nStatus = RD_FAILED;
682.     break;
683. }
684. //-----
685. rtmp.dlg->AppendCInfo("成功建立网络流 (NetStream) !");
686. //-----
687. }
688. else
689. {
690.     nInitialFrameSize = 0;
691.
692.     if (retries)
693.     {
694.         RTMP_Log(RTMP_LOGERROR, "Failed to resume the stream\n\n");
695.         if (!RTMP_IsTimedout(&rtmp))
696.             nStatus = RD_FAILED;
697.         else
698.             nStatus = RD_INCOMPLETE;
699.         break;
700.     }
701.     RTMP_Log(RTMP_LOGINFO, "Connection timed out, trying to resume.\n\n");
702.     /* Did we already try pausing, and it still didn't work? */
703.     if (rtmp.m_pausing == 3)
704.     {
705.         /* Only one try at reconnecting... */
706.         retries = 1;
707.         dSeek = rtmp.m_pauseStamp;
708.         if (dStopOffset > 0)
709.         {
710.             if (dStopOffset <= dSeek)
711.             {
712.                 RTMP_LogPrintf("Already Completed\n");
713.                 nStatus = RD_SUCCESS;
714.                 break;
715.             }
716.         }
717.         if (!RTMP_ReconnectStream(&rtmp, dSeek))
718.         {
719.             RTMP_Log(RTMP_LOGERROR, "Failed to resume the stream\n\n");
720.             if (!RTMP_IsTimedout(&rtmp))
721.                 nStatus = RD_FAILED;
722.             else
723.                 nStatus = RD_INCOMPLETE;
724.             break;
725.         }
726.     }
727.     else if (!RTMP_ToggleStream(&rtmp))
728.     {
729.         RTMP_Log(RTMP_LOGERROR, "Failed to resume the stream\n\n");

```

```

730.         if (!RTMP_IsTimeout(&rtmp))
731.             nStatus = RD_FAILED;
732.         else
733.             nStatus = RD_INCOMPLETE;
734.         break;
735.     }
736.     bResume = TRUE;
737. }
738. //-----
739.
740. //-----
741. rtmp.dlg->AppendCInfo("开始将媒体数据写入文件");
742. //-----
743. //下载,写入文件
744. nStatus = Download(&rtmp, file, dSeek, dStopOffset, duration, bResume,
745.                   metaHeader, nMetaHeaderSize, initialFrame,
746.                   initialFrameType, nInitialFrameSize,
747.                   nSkipKeyFrames, bStdoutMode, bLiveStream, bHashes,
748.                   bOverrideBufferTime, bufferTime, &percent);
749. free(initialFrame);
750. initialFrame = NULL;
751.
752. /* If we succeeded, we're done.
753. */
754. if (nStatus != RD_INCOMPLETE || !RTMP_IsTimeout(&rtmp) || bLiveStream)
755.     break;
756. }
757. //当下载完的时候
758. if (nStatus == RD_SUCCESS)
759. {
760.     //-----
761.     rtmp.dlg->AppendCInfo("写入文件完成");
762.     //-----
763.     RTMP_LogPrintf("Download complete\n");
764. }
765. //没下载完的时候
766. else if (nStatus == RD_INCOMPLETE)
767. {
768.     //-----
769.     rtmp.dlg->AppendCInfo("写入文件可能不完整");
770.     //-----
771.     RTMP_LogPrintf
772.     ("Download may be incomplete (downloaded about %.2f%%), try resuming\n",
773.      percent);
774. }
775. //后续清理工作
776. clean:
777.     //-----
778.     rtmp.dlg->AppendCInfo("关闭连接");
779.     //-----
780.     RTMP_Log(RTMP_LOGDEBUG, "Closing connection.\n");
781.     RTMP_Close(&rtmp);
782.     rtmp.dlg->AppendCInfo("关闭文件");
783.     if (file != 0)
784.         fclose(file);
785.     rtmp.dlg->AppendCInfo("关闭Socket");
786.     CleanupSockets();
787.
788. #ifdef _DEBUG
789.     if (netstackdump != 0)
790.         fclose(netstackdump);
791.     if (netstackdump_read != 0)
792.         fclose(netstackdump_read);
793. #endif
794.     return nStatus;
795. }

```

其中InitSocket()代码很简单，初始化了Socket，如下：

```

[cpp]
1. // 初始化 sockets
2. int
3. InitSockets()
4. {
5.     #ifdef WIN32
6.         WORD version;
7.         WSADATA wsaData;
8.
9.         version = MAKEWORD(1, 1);
10.        return (WSAStartup(version, &wsaData) == 0);
11.    #else
12.        return TRUE;
13.    #endif
14. }

```

CleanupSockets()则更简单：

```

1. inline void
2. CleanupSockets()
3. {
4.     #ifdef WIN32
5.         WSACleanup();
6.     #endif
7. }

```

Download()函数则比较复杂：

```

1. int
2. Download(RTMP * rtmp,          // connected RTMP object
3.     FILE * file, uint32_t dSeek, uint32_t dStopOffset, double duration, int bResume, char *metaHeader, uint32_t nMetaHeaderSize, char *initialFrame, int initialFrameType, uint32_t nInitialFrameSize, int nSkipKeyFrames, int bStdoutMode, int bLiveStream, int bHashes, int bOverrideBufferTime, uint32_t bufferTime, double *percent) // percentage downloaded [out]
4. {
5.     int32_t now, lastUpdate;
6.     int bufferSize = 64 * 1024;
7.     char *buffer = (char *) malloc(bufferSize);
8.     int nRead = 0;
9.
10.    //long ftell(FILE *stream);
11.    //返回当前文件指针
12.    RTMP_LogPrintf("开始下载!\n");
13.    off_t size = ftello(file);
14.    unsigned long lastPercent = 0;
15.    //时间戳
16.    rtmp->m_read.timestamp = dSeek;
17.
18.    *percent = 0.0;
19.
20.    if (rtmp->m_read.timestamp)
21.    {
22.        RTMP_Log(RTMP_LOGDEBUG, "Continuing at TS: %d ms\n", rtmp->m_read.timestamp);
23.    }
24.    //是直播
25.    if (bLiveStream)
26.    {
27.        RTMP_LogPrintf("直播流\n");
28.    }
29.    else
30.    {
31.        // print initial status
32.        // Workaround to exit with 0 if the file is fully (> 99.9%) downloaded
33.        if (duration > 0)
34.        {
35.            if ((double) rtmp->m_read.timestamp >= (double) duration * 999.0)
36.            {
37.                RTMP_LogPrintf("Already Completed at: %.3f sec Duration=%.3f sec\n",
38.                    (double) rtmp->m_read.timestamp / 1000.0,
39.                    (double) duration / 1000.0);
40.                return RD_SUCCESS;
41.            }
42.        }
43.        else
44.        {
45.            *percent = ((double) rtmp->m_read.timestamp) / (duration * 1000.0) * 100.0;
46.            *percent = ((double) (int) (*percent * 10.0)) / 10.0;
47.            RTMP_LogPrintf("%s download at: %.3f kB / %.3f sec (%.1f%%)\n",
48.                bResume ? "Resuming" : "Starting",
49.                (double) size / 1024.0, (double) rtmp->m_read.timestamp / 1000.0,
50.                *percent);
51.        }
52.    }
53.    else
54.    {
55.        RTMP_LogPrintf("%s download at: %.3f kB\n",
56.            bResume ? "Resuming" : "Starting",
57.            (double) size / 1024.0);
58.    }
59.
60.    if (dStopOffset > 0)
61.        RTMP_LogPrintf("For duration: %.3f sec\n", (double) (dStopOffset - dSeek) / 1000.0);
62.
63.    //各种设置参数到rtmp连接
64.    if (bResume && nInitialFrameSize > 0)
65.        rtmp->m_read.flags |= RTMP_READ_RESUME;
66.    rtmp->m_read.initialFrameType = initialFrameType;
67.    rtmp->m_read.nResumeTS = dSeek;
68.    rtmp->m_read.metaHeader = metaHeader;
69.    rtmp->m_read.initialFrame = initialFrame;
70.    rtmp->m_read.nMetaHeaderSize = nMetaHeaderSize;
71.    rtmp->m_read.nInitialFrameSize = nInitialFrameSize;
72.
73.    now = RTMP_GetTime();
74.    lastUpdate = now - 1000;
75.

```

```

75. do
76. {
77. //从rtmp中把bufferSize (64k) 个数据读入buffer
78. nRead = RTMP_Read(rtmp, buffer, bufferSize);
79. //RTMP_LogPrintf("nRead: %d\n", nRead);
80. if (nRead > 0)
81. {
82. //函数: size_t fwrite(const void* buffer, size_t size, size_t count, FILE* stream);
83. //向文件读入写入一个数据块。返回值: 返回实际写入的数据块数目
84. // (1) buffer: 是一个指针, 对fwrite来说, 是要输出数据的地址。
85. // (2) size: 要写入内容的单字节数;
86. // (3) count: 要进行写入size字节的数据项的个数;
87. // (4) stream: 目标文件指针。
88. // (5) 返回实际写入的数据项个数count。
89. //关键。把buffer里面的数据写成文件
90. if (fwrite(buffer, sizeof(unsigned char), nRead, file) !=
91. (size_t) nRead)
92. {
93. RTMP_Log(RTMP_LOGERROR, "%s: Failed writing, exiting!", __FUNCTION__);
94. free(buffer);
95. return RD_FAILED;
96. }
97. //记录已经写入的字节数
98. size += nRead;
99.
100. //RTMP_LogPrintf("write %dbytes (%.1f kB)\n", nRead, nRead/1024.0);
101. if (duration <= 0) // if duration unknown try to get it from the stream (onMetaData)
102. duration = RTMP_GetDuration(rtmp);
103.
104. if (duration > 0)
105. {
106. // make sure we claim to have enough buffer time!
107. if (!bOverrideBufferTime && bufferTime < (duration * 1000.0))
108. {
109. bufferTime = (uint32_t) (duration * 1000.0) + 5000; // 再加5s以确保buffertime足够长
110.
111. RTMP_Log(RTMP_LOGDEBUG,
112. "Detected that buffer time is less than duration, resetting to: %dms",
113. bufferTime);
114. //重设Buffer长度
115. RTMP_SetBufferMS(rtmp, bufferTime);
116. //给服务器发送UserControl消息通知Buffer改变
117. RTMP_UpdateBufferMS(rtmp);
118. }
119. //计算百分比
120. *percent = ((double) rtmp->m_read.timestamp) / (duration * 1000.0) * 100.0;
121. *percent = ((double) (int) (*percent * 10.0)) / 10.0;
122. if (bHashes)
123. {
124. if (lastPercent + 1 <= *percent)
125. {
126. RTMP_LogStatus("#");
127. lastPercent = (unsigned long) *percent;
128. }
129. }
130. else
131. {
132. //设置显示数据的更新间隔200ms
133. now = RTMP_GetTime();
134. if (abs(now - lastUpdate) > 200)
135. {
136. RTMP_LogStatus("\r%.3f kB / %.2f sec (%.1f%%)",
137. (double) size / 1024.0,
138. (double) (rtmp->m_read.timestamp) / 1000.0, *percent);
139. lastUpdate = now;
140. }
141. }
142. }
143. else
144. {
145. //现在距离开机的毫秒数
146. now = RTMP_GetTime();
147. //每间隔200ms刷新一次数据
148. if (abs(now - lastUpdate) > 200)
149. {
150. if (bHashes)
151. RTMP_LogStatus("#");
152. else
153. //size为已写入文件的字节数
154. RTMP_LogStatus("\r%.3f kB / %.2f sec", (double) size / 1024.0,
155. (double) (rtmp->m_read.timestamp) / 1000.0);
156. lastUpdate = now;
157. }
158. }
159. }
160. #ifndef _DEBUG
161. else
162. {
163. RTMP_Log(RTMP_LOGDEBUG, "zero read!");
164. }
165. #endif
166.

```

```

167.     }
168.     while (!RTMP_ctrlC && nRead > -1 && RTMP_IsConnected(rtmp) && !RTMP_IsTimedout(rtmp));
169.     free(buffer);
170.     if (nRead < 0)
171.         //nRead是读取情况
172.         nRead = rtmp->m_read.status;
173.
174.     /* Final status update */
175.     if (!bHashes)
176.     {
177.         if (duration > 0)
178.         {
179.             *percent = ((double) rtmp->m_read.timestamp) / (duration * 1000.0) * 100.0;
180.             *percent = ((double) (int) (*percent * 10.0)) / 10.0;
181.             //输出
182.             RTMP_LogStatus("\r%.3f kB / %.2f sec (%.1f%%)",
183.                 (double) size / 1024.0,
184.                 (double) (rtmp->m_read.timestamp) / 1000.0, *percent);
185.         }
186.         else
187.         {
188.             RTMP_LogStatus("\r%.3f kB / %.2f sec", (double) size / 1024.0,
189.                 (double) (rtmp->m_read.timestamp) / 1000.0);
190.         }
191.     }
192.
193.     RTMP_Log(RTMP_LOGDEBUG, "RTMP_Read returned: %d", nRead);
194.     //读取错误
195.     if (bResume && nRead == -2)
196.     {
197.         RTMP_LogPrintf("Couldn't resume FLV file, try --skip %d\n\n",
198.             nSkipKeyFrames + 1);
199.         return RD_FAILED;
200.     }
201.     //读取正确
202.     if (nRead == -3)
203.         return RD_SUCCESS;
204.     //没读完...
205.     if ((duration > 0 && *percent < 99.9) || RTMP_ctrlC || nRead < 0
206.         || RTMP_IsTimedout(rtmp))
207.     {
208.         return RD_INCOMPLETE;
209.     }
210.
211.     return RD_SUCCESS;
212. }

```

以上内容是我能理解到的rtmpdump.c里面的内容。

rtmpdump源代码（Linux）：<http://download.csdn.net/detail/leixiaohua1020/6376561>

rtmpdump源代码（VC 2005 工程）：<http://download.csdn.net/detail/leixiaohua1020/6563163>

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/12952977>

文章标签：[rtmpdump](#) [rtmp](#) [源代码](#) [flash](#) [socket](#)

个人分类：[libRTMP](#)

所属专栏：[开源多媒体项目源代码分析](#)

此PDF由spygg生成,请尊重原作者版权!!!

我的邮箱:liushidc@163.com