原 RTMPdump 源代码分析 1： main()函数

===

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)

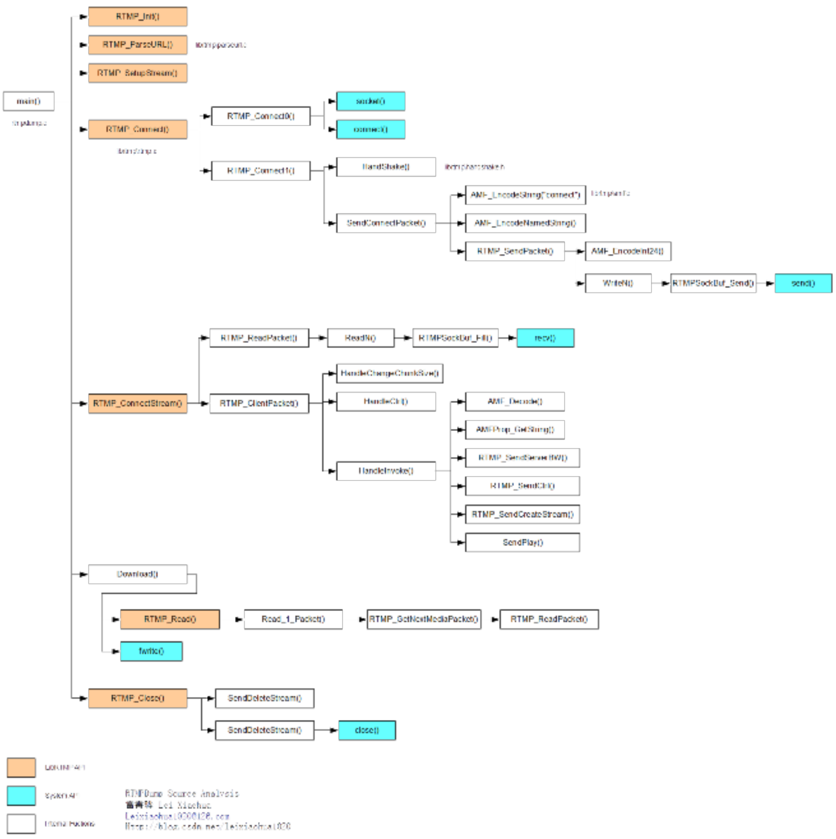
RTMPdump (libRTMP) 源代码分析10： 处理各种消息 (Message)

===

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]

```cpp
RTMP_Init();//初始化结构体
InitSockets();//初始化Socket
RTMP_ParseURL();//解析输入URL
RTMP_SetupStream();//一些设置
fopen();//打开文件，准备写入
RTMP_Connect();//建立NetConnection
RTMP_ConnectStream()//建立NetStream
Download();//下载函数
RTMP_Close();//关闭连接
fclose();//关闭文件
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]

```cpp
int rtmpdump(LPVOID lpParam,int argc,char **argv)
{

  extern char *optarg;
  //一定要设置，否则只能运行一次
  extern int optind;
  optind=0;
  int nStatus = RD_SUCCESS;
  double percent = 0;
  double duration = 0.0;

  int nSkipKeyFrames = DEF_SKIPFRM; // skip this number of keyframes when resuming

  int bOverrideBufferTime = FALSE;  // if the user specifies a buffer time override this is true
  int bStdoutMode = TRUE;   // if true print the stream directly to stdout, messages go to stderr
  int bResume = FALSE;      // true in resume mode
  uint32_t dSeek = 0;       // seek position in resume mode, 0 otherwise
  uint32_t bufferTime = DEF_BUFTIME;

  // meta header and initial frame for the resume mode (they are read from the file and compared with
  // the stream we are trying to continue
  char *metaHeader = 0;
  uint32_t nMetaHeaderSize = 0;

  // video keyframe for matching
  char *initialFrame = 0;
  uint32_t nInitialFrameSize = 0;
  int initialFrameType = 0; // tye: audio or video

  AVal hostname = { 0, 0 };
  AVal playpath = { 0, 0 };
  AVal subscribepath = { 0, 0 };
  int port = -1;
  int protocol = RTMP_PROTOCOL_UNDEFINED;
  int retries = 0;
  int bLiveStream = FALSE;  // 是直播流吗? then we can't seek/resume
  int bHashes = FALSE;      // display byte counters not hashes by default

  long int timeout = DEF_TIMEOUT;   // timeout connection after 120 seconds
  uint32_t dStartOffset = 0;    // 非直播流搜寻点seek position in non-live mode
  uint32_t dStopOffset = 0;
  RTMP rtmp = { 0 };

  AVal swfUrl = { 0, 0 };
  AVal tcUrl = { 0, 0 };
  AVal pageUrl = { 0, 0 };
  AVal app = { 0, 0 };
  AVal auth = { 0, 0 };
  AVal swfHash = { 0, 0 };
  uint32_t swfSize = 0;
  AVal flashVer = { 0, 0 };
  AVal sockshost = { 0, 0 };

#ifdef CRYPTO
  int swfAge = 30;  /* 30 days for SWF cache by default */
```

```
  int swfVfy = 0;
  unsigned char hash[RTMP_SWF_HASHLEN];
#endif

  char *flvFile = 0;

  signal(SIGINT, sigIntHandler);
  signal(SIGTERM, sigIntHandler);
#ifndef WIN32
  signal(SIGHUP, sigIntHandler);
  signal(SIGPIPE, sigIntHandler);
  signal(SIGQUIT, sigIntHandler);
#endif

  RTMP_debuglevel = RTMP_LOGINFO;

  //首先搜寻“ --quiet”选项
  int index = 0;
  while (index < argc)
    {
      if (strcmp(argv[index], "--quiet") == 0
      || strcmp(argv[index], "-q") == 0)
    RTMP_debuglevel = RTMP_LOGCRIT;
      index++;
    }
#define RTMPDUMP_VERSION "1.0"
  RTMP_LogPrintf("RTMP流媒体下载 %s\n", RTMPDUMP_VERSION);
  RTMP_LogPrintf
    ("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");
  //RTMP_LogPrintf("输入 -h 获取命令选项\n");
    RTMP_Init(&rtmp);
    //句柄---------------------------
    rtmp.dlg=(CSpecialPRTMPDlg *)lpParam;
    //-------------------------------
    //---------------------
    rtmp.dlg->AppendCInfo("开始初始化Socket...");
    //---------------------------
    if (!InitSockets())
    {
        //---------------------
        rtmp.dlg->AppendCInfo("初始化Socket失败！");
        //---------------------------
        RTMP_Log(RTMP_LOGERROR,
            "Couldn't load sockets support on your platform, exiting!");
        return RD_FAILED;
    }
  //---------------------
  rtmp.dlg->AppendCInfo("成功初始化Socket");
  //---------------------------
  /* sleep(30); */



  int opt;
/*  struct option longopts[] = {
    {"help", 0, NULL, 'h'},
    {"host", 1, NULL, 'n'},
    {"port", 1, NULL, 'c'},
    {"socks", 1, NULL, 'S'},
    {"protocol", 1, NULL, 'l'},
    {"playpath", 1, NULL, 'y'},
    {"playlist", 0, NULL, 'Y'},
    {"rtmp", 1, NULL, 'r'},
    {"swfUrl", 1, NULL, 's'},
    {"tcUrl", 1, NULL, 't'},
    {"pageUrl", 1, NULL, 'p'},
    {"app", 1, NULL, 'a'},
    {"auth", 1, NULL, 'u'},
    {"conn", 1, NULL, 'C'},
#ifdef CRYPTO
    {"swfhash", 1, NULL, 'w'},
    {"swfsize", 1, NULL, 'x'},
    {"swfVfy", 1, NULL, 'W'},
    {"swfAge", 1, NULL, 'X'},
#endif
    {"flashVer", 1, NULL, 'f'},
    {"live", 0, NULL, 'v'},
    {"flv", 1, NULL, 'o'},
    {"resume", 0, NULL, 'e'},
    {"timeout", 1, NULL, 'm'},
    {"buffer", 1, NULL, 'b'},
    {"skip", 1, NULL, 'k'},
    {"subscribe", 1, NULL, 'd'},
    {"start", 1, NULL, 'A'},
    {"stop", 1, NULL, 'B'},
    {"token", 1, NULL, 'T'},
    {"hashes", 0, NULL, '#'},
    {"debug", 0, NULL, 'z'},
    {"quiet", 0, NULL, 'q'},
    {"verbose", 0, NULL, 'V'},
    {0, 0, 0, 0}
```

```
  };*/
  //分析命令行参数，注意用法。
  //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
  //一直循环直到获取所有的opt
  while ((opt =
      getopt/*_long*/(argc, argv,
              "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:#"/*,
              longopts, NULL*/)) != -1)
    {
    //不同的选项做不同的处理
      switch (opt)
    {
    case 'h':
      usage(argv[0]);
      return RD_SUCCESS;
#ifdef CRYPTO
    case 'w':
      {
        int res = hex2bin(optarg, &swfHash.av_val);
        if (res != RTMP_SWF_HASHLEN)
          {
        swfHash.av_val = NULL;
        RTMP_Log(RTMP_LOGWARNING,
            "Couldn't parse swf hash hex string, not hexstring or not %d bytes, ignoring!", RTMP_SWF_HASHLEN);
          }
        swfHash.av_len = RTMP_SWF_HASHLEN;
        break;
      }
    case 'x':
      {
        int size = atoi(optarg);
        if (size <= 0)
          {
        RTMP_Log(RTMP_LOGERROR, "SWF Size must be at least 1, ignoring\n");
          }
        else
          {
        swfSize = size;
          }
        break;
      }
        case 'W':
      STR2AVAL(swfUrl, optarg);
      swfVfy = 1;
          break;
        case 'X':
      {
        int num = atoi(optarg);
        if (num < 0)
          {
        RTMP_Log(RTMP_LOGERROR, "SWF Age must be non-negative, ignoring\n");
          }
        else
          {
        swfAge = num;
          }
      }
          break;
#endif
    case 'k':
      nSkipKeyFrames = atoi(optarg);
      if (nSkipKeyFrames < 0)
        {
          RTMP_Log(RTMP_LOGERROR,
          "Number of keyframes skipped must be greater or equal zero, using zero!");
          nSkipKeyFrames = 0;
        }
      else
        {
          RTMP_Log(RTMP_LOGDEBUG, "Number of skipped key frames for resume: %d",
          nSkipKeyFrames);
        }
      break;
    case 'b':
      {
        int32_t bt = atol(optarg);
        if (bt < 0)
          {
        RTMP_Log(RTMP_LOGERROR,
            "Buffer time must be greater than zero, ignoring the specified value %d!",
            bt);
          }
        else
          {
        bufferTime = bt;
        bOverrideBufferTime = TRUE;
          }
        break;
      }
    //直播流
    case 'v':
```

```
        //----------------
        rtmp.dlg->AppendCInfo("该RTMP的URL是一个直播流");
        //----------------
      bLiveStream = TRUE;   // no seeking or resuming possible!
      break;
    case 'd':
      STR2AVAL(subscribepath, optarg);
      break;
    case 'n':
      STR2AVAL(hostname, optarg);
      break;
    case 'c':
      port = atoi(optarg);
      break;
    case 'l':
      protocol = atoi(optarg);
      if (protocol < RTMP_PROTOCOL_RTMP || protocol > RTMP_PROTOCOL_RTMPTS)
        {
          RTMP_Log(RTMP_LOGERROR, "Unknown protocol specified: %d", protocol);
          return RD_FAILED;
        }
      break;
    case 'y':
      STR2AVAL(playpath, optarg);
      break;
    case 'Y':
      RTMP_SetOpt(&rtmp, &av_playlist, (AVal *)&av_true);
      break;
      //路径参数-r
    case 'r':
      {
        AVal parsedHost, parsedApp, parsedPlaypath;
        unsigned int parsedPort = 0;
        int parsedProtocol = RTMP_PROTOCOL_UNDEFINED;
        //解析URL。注optarg指向参数（URL）
        RTMP_LogPrintf("RTMP URL : %s\n",optarg);
        //----------------
        rtmp.dlg->AppendCInfo("解析RTMP的URL...");
        //----------------
        if (!RTMP_ParseURL
        (optarg, &parsedProtocol, &parsedHost, &parsedPort,
         &parsedPlaypath, &parsedApp))
          {
            //----------------
            rtmp.dlg->AppendCInfo("解析RTMP的URL失败！");
            //----------------
        RTMP_Log(RTMP_LOGWARNING, "无法解析 url (%s)!",
            optarg);
          }
        else
          {
            //----------------
            rtmp.dlg->AppendCInfo("解析RTMP的URL成功");
            //----------------
        //把解析出来的数据赋值
        if (!hostname.av_len)
          hostname = parsedHost;
        if (port == -1)
          port = parsedPort;
        if (playpath.av_len == 0 && parsedPlaypath.av_len)
          {
            playpath = parsedPlaypath;
          }
        if (protocol == RTMP_PROTOCOL_UNDEFINED)
          protocol = parsedProtocol;
        if (app.av_len == 0 && parsedApp.av_len)
          {
            app = parsedApp;
          }
          }
          break;
      }
    case 's':
      STR2AVAL(swfUrl, optarg);
      break;
    case 't':
      STR2AVAL(tcUrl, optarg);
      break;
    case 'p':
      STR2AVAL(pageUrl, optarg);
      break;
    case 'a':
      STR2AVAL(app, optarg);
      break;
    case 'f':
      STR2AVAL(flashVer, optarg);
      break;
    //指定输出文件
    case 'o':
      flvFile = optarg;
      if (strcmp(flvFile, "-"))
```

```
        bStdoutMode = FALSE;

      break;
    case 'e':
      bResume = TRUE;
      break;
    case 'u':
      STR2AVAL(auth, optarg);
      break;
    case 'C': {
      AVal av;
      STR2AVAL(av, optarg);
      if (!RTMP_SetOpt(&rtmp, &av_conn, &av))
        {
          RTMP_Log(RTMP_LOGERROR, "Invalid AMF parameter: %s", optarg);
          return RD_FAILED;
        }
      }
      break;
    case 'm':
      timeout = atoi(optarg);
      break;
    case 'A':
      dStartOffset = (int) (atof(optarg) * 1000.0);
      break;
    case 'B':
      dStopOffset = (int) (atof(optarg) * 1000.0);
      break;
    case 'T': {
      AVal token;
      STR2AVAL(token, optarg);
      RTMP_SetOpt(&rtmp, &av_token, &token);
      }
      break;
    case '#':
      bHashes = TRUE;
      break;
    case 'q':
      RTMP_debuglevel = RTMP_LOGCRIT;
      break;
    case 'V':
      RTMP_debuglevel = RTMP_LOGDEBUG;
      break;
    case 'z':
      RTMP_debuglevel = RTMP_LOGALL;
      break;
    case 'S':
      STR2AVAL(sockshost, optarg);
      break;
    default:
      RTMP_LogPrintf("unknown option: %c\n", opt);
      usage(argv[0]);
      return RD_FAILED;
      break;
    }
    }

  if (!hostname.av_len)
    {
      RTMP_Log(RTMP_LOGERROR,
      "您必须指定 主机名(hostname) (--host) 或 url (-r \"rtmp://host[:port]/playpath\") 包含 a hostname");
      return RD_FAILED;
    }
  if (playpath.av_len == 0)
    {
      RTMP_Log(RTMP_LOGERROR,
      "您必须指定 播放路径(playpath) (--playpath) 或 url (-r \"rtmp://host[:port]/playpath\") 包含 a playpath");
      return RD_FAILED;
    }

  if (protocol == RTMP_PROTOCOL_UNDEFINED)
    {
      RTMP_Log(RTMP_LOGWARNING,
      "您没有指定 协议(protocol) (--protocol) 或 rtmp url (-r), 默认协议 RTMP");
      protocol = RTMP_PROTOCOL_RTMP;
    }
  if (port == -1)
    {
      RTMP_Log(RTMP_LOGWARNING,
      "您没有指定 端口(port) (--port) 或 rtmp url (-r), 默认端口 1935");
      port = 0;
    }
  if (port == 0)
    {
      if (protocol & RTMP_FEATURE_SSL)
    port = 443;
      else if (protocol & RTMP_FEATURE_HTTP)
    port = 80;
      else
    port = 1935;
    }
```

```
    if (flvFile == 0)
      {
        RTMP_Log(RTMP_LOGWARNING,
        "请指定一个输出文件 (-o filename), using stdout");
        bStdoutMode = TRUE;
      }

    if (bStdoutMode && bResume)
      {
        RTMP_Log(RTMP_LOGWARNING,
        "Can't resume in stdout mode, ignoring --resume option");
        bResume = FALSE;
      }

    if (bLiveStream && bResume)
      {
        RTMP_Log(RTMP_LOGWARNING, "Can't resume live stream, ignoring --resume option");
        bResume = FALSE;
      }

  #ifdef CRYPTO
    if (swfVfy)
      {
        if (RTMP_HashSWF(swfUrl.av_val, (unsigned int *)&swfSize, hash, swfAge) == 0)
          {
            swfHash.av_val = (char *)hash;
            swfHash.av_len = RTMP_SWF_HASHLEN;
          }
      }

    if (swfHash.av_len == 0 && swfSize > 0)
      {
        RTMP_Log(RTMP_LOGWARNING,
        "Ignoring SWF size, supply also the hash with --swfhash");
        swfSize = 0;
      }

    if (swfHash.av_len != 0 && swfSize == 0)
      {
        RTMP_Log(RTMP_LOGWARNING,
        "Ignoring SWF hash, supply also the swf size  with --swfsize");
        swfHash.av_len = 0;
        swfHash.av_val = NULL;
      }
  #endif

    if (tcUrl.av_len == 0)
      {
        char str[512] = { 0 };

        tcUrl.av_len = snprintf(str, 511, "%s://%.*s:%d/%.*s",
             RTMPProtocolStringsLower[protocol], hostname.av_len,
             hostname.av_val, port, app.av_len, app.av_val);
        tcUrl.av_val = (char *) malloc(tcUrl.av_len + 1);
        strcpy(tcUrl.av_val, str);
      }

    int first = 1;

    // User defined seek offset
    if (dStartOffset > 0)
      {
        //直播流
        if (bLiveStream)
      {
        RTMP_Log(RTMP_LOGWARNING,
            "Can't seek in a live stream, ignoring --start option");
        dStartOffset = 0;
      }
      }
    //----------------
    rtmp.dlg->AppendCInfo("开始初始化RTMP连接的参数...");
    //----------------
    //设置
    RTMP_SetupStream(&rtmp, protocol, &hostname, port, &sockshost, &playpath,
             &tcUrl, &swfUrl, &pageUrl, &app, &auth, &swfHash, swfSize,
             &flashVer, &subscribepath, dSeek, dStopOffset, bLiveStream, timeout);
    //此处设置参数----------------
    rtmp.dlg->AppendCInfo("成功初始化RTMP连接的参数");
    //----------------------------
    char *temp=(char *)malloc(MAX_URL_LENGTH);

    memcpy(temp,rtmp.Link.hostname.av_val,rtmp.Link.hostname.av_len);
    temp[rtmp.Link.hostname.av_len]='\0';
    rtmp.dlg->AppendB_R_L_Info("主机名",temp);

    itoa(rtmp.Link.port,temp,10);
    rtmp.dlg->AppendB_R_L_Info("端口号",temp);

    memcpy(temp,rtmp.Link.app.av_val,rtmp.Link.app.av_len);
    temp[rtmp.Link.app.av_len]='\0';
```

```
  temp[rtmp.Link.app.av_len]='\0';
  rtmp.dlg->AppendB_R_L_Info("应用程序",temp);

  memcpy(temp,rtmp.Link.playpath.av_val,rtmp.Link.playpath.av_len);
  temp[rtmp.Link.playpath.av_len]='\0';
  rtmp.dlg->AppendB_R_L_Info("路径",temp);


  //----------------------------

  /* Try to keep the stream moving if it pauses on us */
  if (!bLiveStream && !(protocol & RTMP_FEATURE_HTTP))
    rtmp.Link.lFlags |= RTMP_LF_BUFX;

  off_t size = 0;

  // ok,我们必须获得timestamp of the last keyframe (only keyframes are seekable) / last audio frame (audio only streams)
  if (bResume)
    {
    //打开文件，输出的文件(Resume)
      nStatus =
    OpenResumeFile(flvFile, &file, &size, &metaHeader, &nMetaHeaderSize,
               &duration);
      if (nStatus == RD_FAILED)
    goto clean;

      if (!file)
    {
      // file does not exist, so go back into normal mode
      bResume = FALSE;  // we are back in fresh file mode (otherwise finalizing file won't be done)
    }
      else
    {
    //获取最后一个关键帧
      nStatus = GetLastKeyframe(file, nSkipKeyFrames,
                     &dSeek, &initialFrame,
                     &initialFrameType, &nInitialFrameSize);
      if (nStatus == RD_FAILED)
        {
          RTMP_Log(RTMP_LOGDEBUG, "Failed to get last keyframe.");
          goto clean;
        }

      if (dSeek == 0)
        {
          RTMP_Log(RTMP_LOGDEBUG,
          "Last keyframe is first frame in stream, switching from resume to normal mode!");
          bResume = FALSE;
        }
    }
    }
  //如果输出文件不存在
  if (!file)
    {
      if (bStdoutMode)
    {
    //直接输出到stdout
      file = stdout;
      SET_BINMODE(file);
    }
      else
    {
    //打开一个文件
    //w+b 读写打开或建立一个二进制文件，允许读和写。
        //-----------------
        rtmp.dlg->AppendCInfo("创建输出文件...");
        //---------------------------
      file = fopen(flvFile, "w+b");
      if (file == 0)
        {
            //-----------------
            rtmp.dlg->AppendCInfo("创建输出文件失败！");
            //---------------------------
          RTMP_LogPrintf("Failed to open file! %s\n", flvFile);
          return RD_FAILED;
        }
      rtmp.dlg->AppendCInfo("成功创建输出文件");
    }
    }

#ifdef _DEBUG
  netstackdump = fopen("netstackdump", "wb");
  netstackdump_read = fopen("netstackdump_read", "wb");
#endif

  while (!RTMP_ctrlC)
    {
      RTMP_Log(RTMP_LOGDEBUG, "Setting buffer time to: %dms", bufferTime);
      //设置Buffer时间
      //-----------------
      rtmp.dlg->AppendCInfo("设置缓冲(Buffer)的时间");
      //---------------------------
```

```
    RTMP_SetBufferMS(&rtmp, bufferTime);
    //第一次执行
    if (first)
  {
    first = 0;
    RTMP_LogPrintf("开始建立连接！\n");
    //-----------------
    rtmp.dlg->AppendCInfo("开始建立连接（NetConnection）...");
    //----------------------------
    //建立连接(Connect)
    if (!RTMP_Connect(&rtmp, NULL))
      {
          //-----------------
          rtmp.dlg->AppendCInfo("建立连接（NetConnection）失败！");
          //----------------------------
        nStatus = RD_FAILED;
        break;
      }
    //-----------------
    rtmp.dlg->AppendCInfo("成功建立连接（NetConnection）");
    //----------------------------
    //RTMP_Log(RTMP_LOGINFO, "已链接...");

    // User defined seek offset
    if (dStartOffset > 0)
      {
        // Don't need the start offset if resuming an existing file
        if (bResume)
      {
        RTMP_Log(RTMP_LOGWARNING,
            "Can't seek a resumed stream, ignoring --start option");
        dStartOffset = 0;
      }
        else
      {
        dSeek = dStartOffset;
      }
      }

    // Calculate the length of the stream to still play
    if (dStopOffset > 0)
      {
        // Quit if start seek is past required stop offset
        if (dStopOffset <= dSeek)
      {
        RTMP_LogPrintf("Already Completed\n");
        nStatus = RD_SUCCESS;
        break;
      }
      }
    //创建流(Stream)（发送connect命令消息后处理传来的数据）
    itoa(rtmp.m_inChunkSize,temp,10);
    rtmp.dlg->AppendB_R_Info("输入Chunk大小",temp);
    itoa(rtmp.m_outChunkSize,temp,10);
    rtmp.dlg->AppendB_R_Info("输出Chunk大小",temp);
    itoa(rtmp.m_stream_id,temp,10);
    rtmp.dlg->AppendB_R_Info("Stream ID",temp);
    itoa(rtmp.m_nBufferMS,temp,10);
    rtmp.dlg->AppendB_R_Info("Buffer时长（ms）",temp);
    itoa(rtmp.m_nServerBW,temp,10);
    rtmp.dlg->AppendB_R_Info("ServerBW",temp);
    itoa(rtmp.m_nClientBW,temp,10);
    rtmp.dlg->AppendB_R_Info("ClientBW",temp);
    itoa((int)rtmp.m_fEncoding,temp,10);
    rtmp.dlg->AppendB_R_Info("命令消息编码方法",temp);
    itoa((int)rtmp.m_fDuration,temp,10);
    rtmp.dlg->AppendB_R_Info("时长（s）",temp);

    rtmp.dlg->ShowBInfo();
    free(temp);
    //-----------------
    rtmp.dlg->AppendCInfo("开始建立网络流（NetStream）");
    //----------------------------
    if (!RTMP_ConnectStream(&rtmp, dSeek))
      {
      //-----------------
      rtmp.dlg->AppendCInfo("建立网络流（NetStream）失败！");
      //-----------------
        nStatus = RD_FAILED;
        break;
      }
    //-----------------
    rtmp.dlg->AppendCInfo("成功建立网络流（NetStream）！");
    //-----------------
  }
    else
  {
    nInitialFrameSize = 0;

        if (retries)
          {
```

```
          RTMP_Log(RTMP_LOGERROR, "Failed to resume the stream\n\n");
          if (!RTMP_IsTimedout(&rtmp))
            nStatus = RD_FAILED;
          else
            nStatus = RD_INCOMPLETE;
          break;
            }
      RTMP_Log(RTMP_LOGINFO, "Connection timed out, trying to resume.\n\n");
          /* Did we already try pausing, and it still didn't work? */
          if (rtmp.m_pausing == 3)
            {
              /* Only one try at reconnecting... */
              retries = 1;
              dSeek = rtmp.m_pauseStamp;
              if (dStopOffset > 0)
                {
                  if (dStopOffset <= dSeek)
                    {
                      RTMP_LogPrintf("Already Completed\n");
              nStatus = RD_SUCCESS;
              break;
                    }
                }
              if (!RTMP_ReconnectStream(&rtmp, dSeek))
                {
              RTMP_Log(RTMP_LOGERROR, "Failed to resume the stream\n\n");
              if (!RTMP_IsTimedout(&rtmp))
            nStatus = RD_FAILED;
              else
            nStatus = RD_INCOMPLETE;
              break;
                }
            }
      else if (!RTMP_ToggleStream(&rtmp))
        {
          RTMP_Log(RTMP_LOGERROR, "Failed to resume the stream\n\n");
          if (!RTMP_IsTimedout(&rtmp))
        nStatus = RD_FAILED;
          else
        nStatus = RD_INCOMPLETE;
          break;
        }
      bResume = TRUE;
    }
    //----------------

    //----------------
    rtmp.dlg->AppendCInfo("开始将媒体数据写入文件");
    //----------------
      //下载,写入文件
      nStatus = Download(&rtmp, file, dSeek, dStopOffset, duration, bResume,
             metaHeader, nMetaHeaderSize, initialFrame,
             initialFrameType, nInitialFrameSize,
             nSkipKeyFrames, bStdoutMode, bLiveStream, bHashes,
             bOverrideBufferTime, bufferTime, &percent);
      free(initialFrame);
      initialFrame = NULL;

      /* If we succeeded, we're done.
       */
      if (nStatus != RD_INCOMPLETE || !RTMP_IsTimedout(&rtmp) || bLiveStream)
    break;
    }
    //当下载完的时候
  if (nStatus == RD_SUCCESS)
    {
        //----------------
        rtmp.dlg->AppendCInfo("写入文件完成");
        //----------------
      RTMP_LogPrintf("Download complete\n");
    }
    //没下载完的时候
  else if (nStatus == RD_INCOMPLETE)
    {
        //----------------
        rtmp.dlg->AppendCInfo("写入文件可能不完整");
        //----------------
      RTMP_LogPrintf
    ("Download may be incomplete (downloaded about %.2f%%), try resuming\n",
     percent);
    }
  //后续清理工作
clean:
  //----------------
  rtmp.dlg->AppendCInfo("关闭连接");
  //----------------
  RTMP_Log(RTMP_LOGDEBUG, "Closing connection.\n");
  RTMP_Close(&rtmp);
  rtmp.dlg->AppendCInfo("关闭文件");
  if (file != 0)
    fclose(file);
```

```
  rtmp.dlg->AppendCInfo("关闭Socket");
  CleanupSockets();

#ifdef _DEBUG
  if (netstackdump != 0)
    fclose(netstackdump);
  if (netstackdump_read != 0)
    fclose(netstackdump_read);
#endif
  return nStatus;
}
```

其中InitSocket()代码很简单，初始化了Socket，如下：

[cpp]

```
// 初始化 sockets
int
InitSockets()
{
#ifdef WIN32
  WORD version;
  WSADATA wsaData;

  version = MAKEWORD(1, 1);
  return (WSAStartup(version, &wsaData) == 0);
#else
  return TRUE;
#endif
}
```

CleanupSockets()则更简单：

[cpp]

```
inline void
CleanupSockets()
{
#ifdef WIN32
  WSACleanup();
#endif
}
```

Download()函数则比较复杂：

[cpp]

```
int
Download(RTMP * rtmp,       // connected RTMP object
     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]
{
  int32_t now, lastUpdate;
  int bufferSize = 64 * 1024;
  char *buffer = (char *) malloc(bufferSize);
  int nRead = 0;

  //long ftell(FILE *stream);
  //返回当前文件指针
  RTMP_LogPrintf("开始下载！\n");
  off_t size = ftello(file);
  unsigned long lastPercent = 0;
  //时间戳
  rtmp->m_read.timestamp = dSeek;

  *percent = 0.0;

  if (rtmp->m_read.timestamp)
    {
      RTMP_Log(RTMP_LOGDEBUG, "Continuing at TS: %d ms\n", rtmp->m_read.timestamp);
    }
  //是直播
  if (bLiveStream)
    {
      RTMP_LogPrintf("直播流\n");
    }
  else
    {
      // print initial status
      // Workaround to exit with 0 if the file is fully (> 99.9%) downloaded
      if (duration > 0)
    {
      if ((double) rtmp->m_read.timestamp >= (double) duration * 999.0)
        {
          RTMP_LogPrintf("Already Completed at: %.3f sec Duration=%.3f sec\n",
            (double) rtmp->m_read.timestamp / 1000.0,
            (double) duration / 1000.0);
```

```
            return RD_SUCCESS;
          }
        else
          {
            *percent = ((double) rtmp->m_read.timestamp) / (duration * 1000.0) * 100.0;
            *percent = ((double) (int) (*percent * 10.0)) / 10.0;
            RTMP_LogPrintf("%s download at: %.3f kB / %.3f sec (%.1f%%)\n",
              bResume ? "Resuming" : "Starting",
              (double) size / 1024.0, (double) rtmp->m_read.timestamp / 1000.0,
              *percent);
          }
      }
        else
      {
        RTMP_LogPrintf("%s download at: %.3f kB\n",
              bResume ? "Resuming" : "Starting",
              (double) size / 1024.0);
      }
      }

    if (dStopOffset > 0)
      RTMP_LogPrintf("For duration: %.3f sec\n", (double) (dStopOffset - dSeek) / 1000.0);

    //各种设置参数到rtmp连接
    if (bResume && nInitialFrameSize > 0)
    rtmp->m_read.flags |= RTMP_READ_RESUME;
    rtmp->m_read.initialFrameType = initialFrameType;
    rtmp->m_read.nResumeTS = dSeek;
    rtmp->m_read.metaHeader = metaHeader;
    rtmp->m_read.initialFrame = initialFrame;
    rtmp->m_read.nMetaHeaderSize = nMetaHeaderSize;
    rtmp->m_read.nInitialFrameSize = nInitialFrameSize;

    now = RTMP_GetTime();
    lastUpdate = now - 1000;
    do
      {
      //从rtmp中把bufferSize（64k）个数据读入buffer
        nRead = RTMP_Read(rtmp, buffer, bufferSize);
        //RTMP_LogPrintf("nRead: %d\n", nRead);
        if (nRead > 0)
      {
      //函数：size_t fwrite(const void* buffer,size_t size,size_t count,FILE* stream);
      //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
      //（1）buffer：是一个指针，对fwrite来说，是要输出数据的地址。
      //（2）size：要写入内容的单字节数；
      //（3）count:要进行写入size字节的数据项的个数；
      //（4）stream:目标文件指针。
      //（5）返回实际写入的数据项个数count。
      //关键。把buffer里面的数据写成文件
          if (fwrite(buffer, sizeof(unsigned char), nRead, file) !=
            (size_t) nRead)
          {
            RTMP_Log(RTMP_LOGERROR, "%s: Failed writing, exiting!", __FUNCTION__);
            free(buffer);
            return RD_FAILED;
          }
          //记录已经写入的字节数
        size += nRead;

        //RTMP_LogPrintf("write %dbytes (%.1f kB)\n", nRead, nRead/1024.0);
        if (duration <= 0) // if duration unknown try to get it from the stream (onMetaData)
          duration = RTMP_GetDuration(rtmp);

        if (duration > 0)
          {
            // make sure we claim to have enough buffer time!
            if (!bOverrideBufferTime && bufferTime < (duration * 1000.0))
          {
            bufferTime = (uint32_t) (duration * 1000.0) + 5000;   // 再加5s以确保buffertime足够长

            RTMP_Log(RTMP_LOGDEBUG,
                "Detected that buffer time is less than duration, resetting to: %dms",
                bufferTime);
            //重设Buffer长度
            RTMP_SetBufferMS(rtmp, bufferTime);
            //给服务器发送UserControl消息通知Buffer改变
            RTMP_UpdateBufferMS(rtmp);
          }
            //计算百分比
            *percent = ((double) rtmp->m_read.timestamp) / (duration * 1000.0) * 100.0;
            *percent = ((double) (int) (*percent * 10.0)) / 10.0;
            if (bHashes)
          {
            if (lastPercent + 1 <= *percent)
              {
                RTMP_LogStatus("#");
                lastPercent = (unsigned long) *percent;
              }
          }
            else
```

```
        {
            //设置显示数据的更新间隔200ms
          now = RTMP_GetTime();
          if (abs(now - lastUpdate) > 200)
            {
              RTMP_LogStatus("\r%.3f kB / %.2f sec (%.1f%%)",
                (double) size / 1024.0,
                (double) (rtmp->m_read.timestamp) / 1000.0, *percent);
              lastUpdate = now;
            }
        }
        }
      else
        {
        //现在距离开机的毫秒数
          now = RTMP_GetTime();
          //每间隔200ms刷新一次数据
          if (abs(now - lastUpdate) > 200)
        {
          if (bHashes)
            RTMP_LogStatus("#");
          else
            //size为已写入文件的字节数
            RTMP_LogStatus("\r%.3f kB / %.2f sec", (double) size / 1024.0,
                  (double) (rtmp->m_read.timestamp) / 1000.0);
          lastUpdate = now;
        }
        }
    }
#ifdef _DEBUG
      else
    {
      RTMP_Log(RTMP_LOGDEBUG, "zero read!");
    }
#endif

    }
  while (!RTMP_ctrlC && nRead > -1 && RTMP_IsConnected(rtmp) && !RTMP_IsTimedout(rtmp));
  free(buffer);
  if (nRead < 0)
    //nRead是读取情况
    nRead = rtmp->m_read.status;

  /* Final status update */
  if (!bHashes)
    {
      if (duration > 0)
    {
      *percent = ((double) rtmp->m_read.timestamp) / (duration * 1000.0) * 100.0;
      *percent = ((double) (int) (*percent * 10.0)) / 10.0;
      //输出
      RTMP_LogStatus("\r%.3f kB / %.2f sec (%.1f%%)",
        (double) size / 1024.0,
        (double) (rtmp->m_read.timestamp) / 1000.0, *percent);
    }
      else
    {
      RTMP_LogStatus("\r%.3f kB / %.2f sec", (double) size / 1024.0,
        (double) (rtmp->m_read.timestamp) / 1000.0);
    }
    }

  RTMP_Log(RTMP_LOGDEBUG, "RTMP_Read returned: %d", nRead);
  //读取错误
  if (bResume && nRead == -2)
    {
      RTMP_LogPrintf("Couldn't resume FLV file, try --skip %d\n\n",
        nSkipKeyFrames + 1);
      return RD_FAILED;
    }
  //读取正确
  if (nRead == -3)
    return RD_SUCCESS;
  //没读完...
  if ((duration > 0 && *percent < 99.9) || RTMP_ctrlC || nRead < 0
      || RTMP_IsTimedout(rtmp))
    {
      return RD_INCOMPLETE;
    }

  return RD_SUCCESS;
}
```

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

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