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

2013年10月23日 14:17:40 阅读数：9925

==========

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)

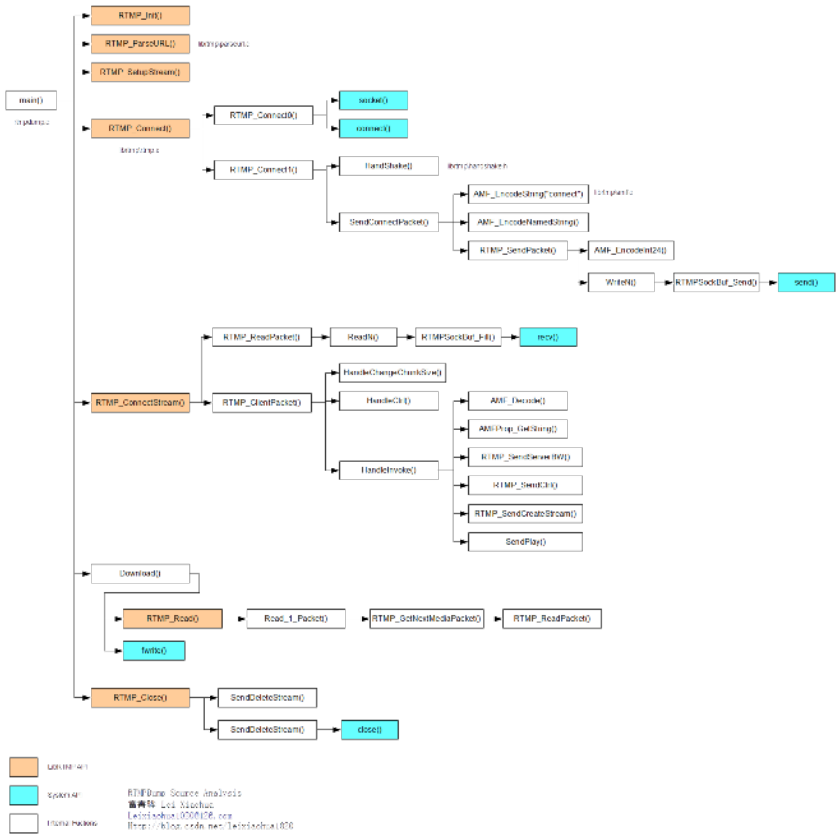
RTMPdump (libRTMP) 源代码分析9： 接收消息 (Message) (接收视音频数据)

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

==========

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

书接上回：RTMPdump 源代码分析 6： 建立一个流媒体连接 （NetStream部分 1）

上回说到，有两个函数尤为重要：

RTMP_ReadPacket()

RTMP_ClientPacket()

而且分析了第一个函数。现在我们再来看看第二个函数吧。 第二个函数的主要作用是：处理消息（Message），并做出响应。

先把带注释的代码贴上：

```cpp
//处理接收到的Chunk
int
RTMP_ClientPacket(RTMP *r, RTMPPacket *packet)
{
  int bHasMediaPacket = 0;
  switch (packet->m_packetType)
    {
    //RTMP消息类型ID=1,设置块大小
    case 0x01:
      /* chunk size */
        //----------------
        r->dlg->AppendCInfo("处理收到的数据。消息 Set Chunk Size (typeID=1)。");
        //----------------------------
        RTMP_LogPrintf("处理消息 Set Chunk Size (typeID=1)\n");
      HandleChangeChunkSize(r, packet);
      break;
    //RTMP消息类型ID=3，致谢
    case 0x03:
      /* bytes read report */
      RTMP_Log(RTMP_LOGDEBUG, "%s, received: bytes read report", __FUNCTION__);
      break;
    //RTMP消息类型ID=4，用户控制
    case 0x04:
      /* ctrl */
        //----------------
        r->dlg->AppendCInfo("处理收到的数据。消息 User Control (typeID=4)。");
        //----------------------------
        RTMP_LogPrintf("处理消息 User Control (typeID=4)\n");
      HandleCtrl(r, packet);
      break;
    //RTMP消息类型ID=5
    case 0x05:
      /* server bw */
        //----------------
        r->dlg->AppendCInfo("处理收到的数据。消息 Window Acknowledgement Size (typeID=5)。");
        //----------------------------
        RTMP_LogPrintf("处理消息 Window Acknowledgement Size (typeID=5)\n");
      HandleServerBW(r, packet);
      break;
    //RTMP消息类型ID=6
    case 0x06:
      /* client bw */
        //----------------
        r->dlg->AppendCInfo("处理收到的数据。消息 Set Peer Bandwidth (typeID=6)。");
        //----------------------------
        RTMP_LogPrintf("处理消息 Set Peer Bandwidth (typeID=6)\n");
      HandleClientBW(r, packet);
      break;
    //RTMP消息类型ID=8，音频数据
    case 0x08:
      /* audio data */
      /*RTMP_Log(RTMP_LOGDEBUG, "%s, received: audio %lu bytes", __FUNCTION__, packet.m_nBodySize); */
      HandleAudio(r, packet);
      bHasMediaPacket = 1;
      if (!r->m_mediaChannel)
    r->m_mediaChannel = packet->m_nChannel;
      if (!r->m_pausing)
    r->m_mediaStamp = packet->m_nTimeStamp;
      break;
    //RTMP消息类型ID=9，视频数据
    case 0x09:
      /* video data */
      /*RTMP_Log(RTMP_LOGDEBUG, "%s, received: video %lu bytes", __FUNCTION__, packet.m_nBodySize); */
      HandleVideo(r, packet);
      bHasMediaPacket = 1;
      if (!r->m_mediaChannel)
    r->m_mediaChannel = packet->m_nChannel;
      if (!r->m_pausing)
    r->m_mediaStamp = packet->m_nTimeStamp;
      break;
    //RTMP消息类型ID=15，AMF3编码，忽略
    case 0x0F:          /* flex stream send */
      RTMP_Log(RTMP_LOGDEBUG,
```

```
      "%s, flex stream send, size %lu bytes, not supported, ignoring",
      __FUNCTION__, packet->m_nBodySize);
      break;
    //RTMP消息类型ID=16, AMF3编码, 忽略
    case 0x10:          /* flex shared object */
      RTMP_Log(RTMP_LOGDEBUG,
      "%s, flex shared object, size %lu bytes, not supported, ignoring",
      __FUNCTION__, packet->m_nBodySize);
      break;
    //RTMP消息类型ID=17, AMF3编码, 忽略
    case 0x11:          /* flex message */
      {
    RTMP_Log(RTMP_LOGDEBUG,
        "%s, flex message, size %lu bytes, not fully supported",
        __FUNCTION__, packet->m_nBodySize);
    /*RTMP_LogHex(packet.m_body, packet.m_nBodySize); */

    /* some DEBUG code */
#if 0
       RTMP_LIB_AMFObject obj;
       int nRes = obj.Decode(packet.m_body+1, packet.m_nBodySize-1);
       if(nRes < 0) {
       RTMP_Log(RTMP_LOGERROR, "%s, error decoding AMF3 packet", __FUNCTION__);
       /*return; */
       }

       obj.Dump();
#endif

    if (HandleInvoke(r, packet->m_body + 1, packet->m_nBodySize - 1) == 1)
      bHasMediaPacket = 2;
    break;
      }
    //RTMP消息类型ID=18, AMF0编码, 数据消息
    case 0x12:
      /* metadata (notify) */

      RTMP_Log(RTMP_LOGDEBUG, "%s, received: notify %lu bytes", __FUNCTION__,
      packet->m_nBodySize);
      //处理元数据,暂时注释
      /*
      if (HandleMetadata(r, packet->m_body, packet->m_nBodySize))
    bHasMediaPacket = 1;
      break;
      */
    //RTMP消息类型ID=19, AMF0编码, 忽略
    case 0x13:
      RTMP_Log(RTMP_LOGDEBUG, "%s, shared object, not supported, ignoring",
      __FUNCTION__);
      break;
    //RTMP消息类型ID=20, AMF0编码, 命令消息
    //处理命令消息！
    case 0x14:
        //----------------
        r->dlg->AppendCInfo("处理收到的数据。消息 命令 (AMF0编码) (typeID=20)。");
        //----------------------------
      /* invoke */
      RTMP_Log(RTMP_LOGDEBUG, "%s, received: invoke %lu bytes", __FUNCTION__,
      packet->m_nBodySize);
      RTMP_LogPrintf("处理命令消息 (typeID=20, AMF0编码)\n");
      /*RTMP_LogHex(packet.m_body, packet.m_nBodySize); */

      if (HandleInvoke(r, packet->m_body, packet->m_nBodySize) == 1)
    bHasMediaPacket = 2;
      break;
    //RTMP消息类型ID=22
    case 0x16:
      {
    /* go through FLV packets and handle metadata packets */
    unsigned int pos = 0;
    uint32_t nTimeStamp = packet->m_nTimeStamp;

    while (pos + 11 < packet->m_nBodySize)
      {
        uint32_t dataSize = AMF_DecodeInt24(packet->m_body + pos + 1);   /* size without header (11) and prevTagSize (4) */

        if (pos + 11 + dataSize + 4 > packet->m_nBodySize)
          {
        RTMP_Log(RTMP_LOGWARNING, "Stream corrupt?!");
        break;
          }
        if (packet->m_body[pos] == 0x12)
          {
        HandleMetadata(r, packet->m_body + pos + 11, dataSize);
          }
        else if (packet->m_body[pos] == 8 || packet->m_body[pos] == 9)
          {
        nTimeStamp = AMF_DecodeInt24(packet->m_body + pos + 4);
        nTimeStamp |= (packet->m_body[pos + 7] << 24);
          }
        pos += (11 + dataSize + 4);
```

```
      }
    if (!r->m_pausing)
      r->m_mediaStamp = nTimeStamp;

    /* FLV tag(s) */
    /*RTMP_Log(RTMP_LOGDEBUG, "%s, received: FLV tag(s) %lu bytes", __FUNCTION__, packet.m_nBodySize); */
    bHasMediaPacket = 1;
    break;
      }
    default:
      RTMP_Log(RTMP_LOGDEBUG, "%s, unknown packet type received: 0x%02x", __FUNCTION__,
      packet->m_packetType);
#ifdef _DEBUG
      RTMP_LogHex(RTMP_LOGDEBUG, (const uint8_t *)packet->m_body, packet->m_nBodySize);
#endif
    }

  return bHasMediaPacket;
}
```

里面注释的比较多，可以看出，大体的思路是，根据接收到的消息（Message）类型的不同，做出不同的响应。例如收到的消息类型为0x01，那么就是设置块（Chunk）大小的协议，那么就调用相应的函数进行处理。

因此，本函数可以说是程序的灵魂，收到的各种命令消息都要经过本函数的判断决定调用哪个函数进行相应的处理。

在这里注意一下消息类型为0x14的消息，即消息类型ID为20的消息，是AMF0编码的命令消息。这在RTMP连接中是非常常见的，比如说各种控制命令：播放，暂停，停止等等。我们来仔细看看它的调用。

可以发现它调用了HandleInvoke()函数来处理服务器发来的AMF0编码的命令，来看看细节：

```cpp
/* Returns 0 for OK/Failed/error, 1 for 'Stop or Complete' */
static int
HandleInvoke(RTMP *r, const char *body, unsigned int nBodySize)
{
  AMFObject obj;
  AVal method;
  int txn;
  int ret = 0, nRes;
  if (body[0] != 0x02)      /* make sure it is a string method name we start with */
    {
      RTMP_Log(RTMP_LOGWARNING, "%s, Sanity failed. no string method in invoke packet",
      __FUNCTION__);
      return 0;
    }

  nRes = AMF_Decode(&obj, body, nBodySize, FALSE);
  if (nRes < 0)
    {
      RTMP_Log(RTMP_LOGERROR, "%s, error decoding invoke packet", __FUNCTION__);
      return 0;
    }

  AMF_Dump(&obj);
  AMFProp_GetString(AMF_GetProp(&obj, NULL, 0), &method);
  txn = (int)AMFProp_GetNumber(AMF_GetProp(&obj, NULL, 1));
  RTMP_Log(RTMP_LOGDEBUG, "%s, server invoking <%s>", __FUNCTION__, method.av_val);

  if (AVMATCH(&method, &av__result))
    {
      AVal methodInvoked = {0};
      int i;

      for (i=0; i<r->m_numCalls; i++) {
    if (r->m_methodCalls[i].num == txn) {
      methodInvoked = r->m_methodCalls[i].name;
      AV_erase(r->m_methodCalls, &r->m_numCalls, i, FALSE);
      break;
    }
      }
      if (!methodInvoked.av_val) {
        RTMP_Log(RTMP_LOGDEBUG, "%s, received result id %d without matching request",
      __FUNCTION__, txn);
    goto leave;
      }
      //----------------
      char temp_str[100];
      sprintf(temp_str,"接收数据。消息 %s 的 Result",methodInvoked.av_val);
      r->dlg->AppendCInfo(temp_str);
      //----------------------------
      RTMP_Log(RTMP_LOGDEBUG, "%s, received result for method call <%s>", __FUNCTION__,
      methodInvoked.av_val);

      if (AVMATCH(&methodInvoked, &av_connect))
    {
        //----------------
        r->dlg->AppendMLInfo(20,0,"命令消息","Result (Connect)");
```

```
        //----------------------------
      if (r->Link.token.av_len)
        {
          AMFObjectProperty p;
          if (RTMP_FindFirstMatchingProperty(&obj, &av_secureToken, &p))
        {
          DecodeTEA(&r->Link.token, &p.p_vu.p_aval);
          SendSecureTokenResponse(r, &p.p_vu.p_aval);
        }
        }
      if (r->Link.protocol & RTMP_FEATURE_WRITE)
        {
          SendReleaseStream(r);
          SendFCPublish(r);
        }
      else
        {
            //----------------
            r->dlg->AppendCInfo("发送数据。消息 Window Acknowledgement Size (typeID=5)。");
            //----------------------------
            RTMP_LogPrintf("发送消息Window Acknowledgement Size(typeID=5)\n");
          RTMP_SendServerBW(r);
          RTMP_SendCtrl(r, 3, 0, 300);
        }
      //----------------
      r->dlg->AppendCInfo("发送数据。消息 命令 (typeID=20) (CreateStream)。");
      //----------------------------
      RTMP_LogPrintf("发送命令消息“CreateStream” (typeID=20)\n");
      RTMP_SendCreateStream(r);

      if (!(r->Link.protocol & RTMP_FEATURE_WRITE))
        {
          /* Send the FCSubscribe if live stream or if subscribepath is set */
          if (r->Link.subscribepath.av_len)
            SendFCSubscribe(r, &r->Link.subscribepath);
          else if (r->Link.lFlags & RTMP_LF_LIVE)
            SendFCSubscribe(r, &r->Link.playpath);
        }
    }
      else if (AVMATCH(&methodInvoked, &av_createStream))
    {
        //----------------
        r->dlg->AppendMLInfo(20,0,"命令消息","Result (CreateStream)");
        //----------------------------
      r->m_stream_id = (int)AMFProp_GetNumber(AMF_GetProp(&obj, NULL, 3));

      if (r->Link.protocol & RTMP_FEATURE_WRITE)
        {
          SendPublish(r);
        }
      else
        {
          if (r->Link.lFlags & RTMP_LF_PLST)
            SendPlaylist(r);
          //----------------
          r->dlg->AppendCInfo("发送数据。消息 命令 (typeID=20) (Play)。");
          //----------------------------
          RTMP_LogPrintf("发送命令消息“play” (typeID=20)\n");
          SendPlay(r);
          RTMP_SendCtrl(r, 3, r->m_stream_id, r->m_nBufferMS);
        }
    }
      else if (AVMATCH(&methodInvoked, &av_play) ||
        AVMATCH(&methodInvoked, &av_publish))
    {
        //----------------
        r->dlg->AppendMLInfo(20,0,"命令消息","Result (Play or Publish)");
        //----------------------------
      r->m_bPlaying = TRUE;
    }
      free(methodInvoked.av_val);
    }
  else if (AVMATCH(&method, &av_onBWDone))
    {
        //----------------
        r->dlg->AppendMLInfo(20,0,"命令消息","onBWDone");
        //----------------------------
      if (!r->m_nBWCheckCounter)
        SendCheckBW(r);
    }
  else if (AVMATCH(&method, &av_onFCSubscribe))
    {
      /* SendOnFCSubscribe(); */
    }
  else if (AVMATCH(&method, &av_onFCUnsubscribe))
    {
        //----------------
        r->dlg->AppendMLInfo(20,0,"命令消息","onFCUnsubscribe");
        //----------------------------
      RTMP_Close(r);
      ret = 1;
    }
```

```
    }
  else if (AVMATCH(&method, &av_ping))
    {
        //---------------
        r->dlg->AppendMLInfo(20,0,"命令消息","Ping");
        //----------------------------
      SendPong(r, txn);
    }
  else if (AVMATCH(&method, &av__onbwcheck))
    {
        //---------------
        r->dlg->AppendMLInfo(20,0,"命令消息","onBWcheck");
        //----------------------------
      SendCheckBWResult(r, txn);
    }
  else if (AVMATCH(&method, &av__onbwdone))
    {
        //---------------
        r->dlg->AppendMLInfo(20,0,"命令消息","onBWdone");
        //----------------------------
      int i;
      for (i = 0; i < r->m_numCalls; i++)
    if (AVMATCH(&r->m_methodCalls[i].name, &av__checkbw))
      {
        AV_erase(r->m_methodCalls, &r->m_numCalls, i, TRUE);
        break;
      }
    }
  else if (AVMATCH(&method, &av__error))
    {
        //---------------
        r->dlg->AppendMLInfo(20,0,"命令消息","error");
        //----------------------------
      RTMP_Log(RTMP_LOGERROR, "rtmp server sent error");
    }
  else if (AVMATCH(&method, &av_close))
    {
        //---------------
        r->dlg->AppendMLInfo(20,0,"命令消息","close");
        //----------------------------
      RTMP_Log(RTMP_LOGERROR, "rtmp server requested close");
      RTMP_Close(r);
    }
  else if (AVMATCH(&method, &av_onStatus))
    {
        //---------------
        r->dlg->AppendMLInfo(20,0,"命令消息","onStatus");
        //----------------------------
      AMFObject obj2;
      AVal code, level;
      AMFProp_GetObject(AMF_GetProp(&obj, NULL, 3), &obj2);
      AMFProp_GetString(AMF_GetProp(&obj2, &av_code, -1), &code);
      AMFProp_GetString(AMF_GetProp(&obj2, &av_level, -1), &level);

      RTMP_Log(RTMP_LOGDEBUG, "%s, onStatus: %s", __FUNCTION__, code.av_val);
      if (AVMATCH(&code, &av_NetStream_Failed)
      || AVMATCH(&code, &av_NetStream_Play_Failed)
      || AVMATCH(&code, &av_NetStream_Play_StreamNotFound)
      || AVMATCH(&code, &av_NetConnection_Connect_InvalidApp))
    {
      r->m_stream_id = -1;
      RTMP_Close(r);
      RTMP_Log(RTMP_LOGERROR, "Closing connection: %s", code.av_val);
    }

      else if (AVMATCH(&code, &av_NetStream_Play_Start))
    {
      int i;
      r->m_bPlaying = TRUE;
      for (i = 0; i < r->m_numCalls; i++)
        {
          if (AVMATCH(&r->m_methodCalls[i].name, &av_play))
        {
          AV_erase(r->m_methodCalls, &r->m_numCalls, i, TRUE);
          break;
        }
        }
    }

      else if (AVMATCH(&code, &av_NetStream_Publish_Start))
    {
      int i;
      r->m_bPlaying = TRUE;
      for (i = 0; i < r->m_numCalls; i++)
        {
          if (AVMATCH(&r->m_methodCalls[i].name, &av_publish))
        {
          AV_erase(r->m_methodCalls, &r->m_numCalls, i, TRUE);
          break;
        }
        }
    }
```

```
    }

      /* Return 1 if this is a Play.Complete or Play.Stop */
      else if (AVMATCH(&code, &av_NetStream_Play_Complete)
      || AVMATCH(&code, &av_NetStream_Play_Stop)
      || AVMATCH(&code, &av_NetStream_Play_UnpublishNotify))
    {
      RTMP_Close(r);
      ret = 1;
    }

      else if (AVMATCH(&code, &av_NetStream_Seek_Notify))
        {
      r->m_read.flags &= ~RTMP_READ_SEEKING;
    }

      else if (AVMATCH(&code, &av_NetStream_Pause_Notify))
        {
      if (r->m_pausing == 1 || r->m_pausing == 2)
      {
        RTMP_SendPause(r, FALSE, r->m_pauseStamp);
        r->m_pausing = 3;
      }
    }
    }
  else if (AVMATCH(&method, &av_playlist_ready))
    {
        //---------------
        r->dlg->AppendMLInfo(20,0,"命令消息","playlist_ready");
        //----------------------------
      int i;
      for (i = 0; i < r->m_numCalls; i++)
        {
          if (AVMATCH(&r->m_methodCalls[i].name, &av_set_playlist))
        {
          AV_erase(r->m_methodCalls, &r->m_numCalls, i, TRUE);
          break;
        }
        }
    }
  else
    {

    }
leave:
  AMF_Reset(&obj);
  return ret;
}

int
RTMP_FindFirstMatchingProperty(AMFObject *obj, const AVal *name,
                   AMFObjectProperty * p)
{
  int n;
  /* this is a small object search to locate the "duration" property */
  for (n = 0; n < obj->o_num; n++)
    {
      AMFObjectProperty *prop = AMF_GetProp(obj, NULL, n);

      if (AVMATCH(&prop->p_name, name))
    {
      *p = *prop;
      return TRUE;
    }

      if (prop->p_type == AMF_OBJECT)
    {
      if (RTMP_FindFirstMatchingProperty(&prop->p_vu.p_object, name, p))
        return TRUE;
    }
    }
  return FALSE;
}
```

该函数主要做了以下几步：

1.调用AMF_Decode()解码AMF命令数据

2.调用AMFProp_GetString()获取具体命令的字符串

3.调用AVMATCH()比较字符串，不同的命令做不同的处理，例如以下几个：

```cpp
AVMATCH(&methodInvoked, &av_connect)
AVMATCH(&methodInvoked, &av_createStream)
AVMATCH(&methodInvoked, &av_play)
AVMATCH(&methodInvoked, &av_publish)
AVMATCH(&method, &av_onBWDone)
```

等等，不一一例举了

具体的处理过程如下所示。在这里说一个“建立网络流”（createStream）的例子，通常发生在建立网络连接（NetConnection）之后，播放（Play）之前。

```cpp
else if (AVMATCH(&methodInvoked, &av_createStream))
    {
        //---------------
        r->dlg->AppendMLInfo(20,0,"命令消息","Result (CreateStream)");
        //----------------------------
      r->m_stream_id = (int)AMFProp_GetNumber(AMF_GetProp(&obj, NULL, 3));

      if (r->Link.protocol & RTMP_FEATURE_WRITE)
        {
          SendPublish(r);
        }
      else
        {
          if (r->Link.lFlags & RTMP_LF_PLST)
            SendPlaylist(r);
          //---------------
          r->dlg->AppendCInfo("发送数据。消息 命令 (typeID=20) (Play)。");
          //----------------------------
          RTMP_LogPrintf("发送命令消息“play” (typeID=20)\n");
          SendPlay(r);
          RTMP_SendCtrl(r, 3, r->m_stream_id, r->m_nBufferMS);
        }
    }
```

由代码可见，程序先获取了stream_id，然后发送了两个消息（Message），分别是SendPlaylist()和SendPlay()，用于获取播放列表，以及开始播放流媒体数据。

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 连接 流媒体

个人分类：libRTMP

所属专栏：开源多媒体项目源代码分析