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

2013年10月23日 00:41:04 阅读数：12730

==========

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)

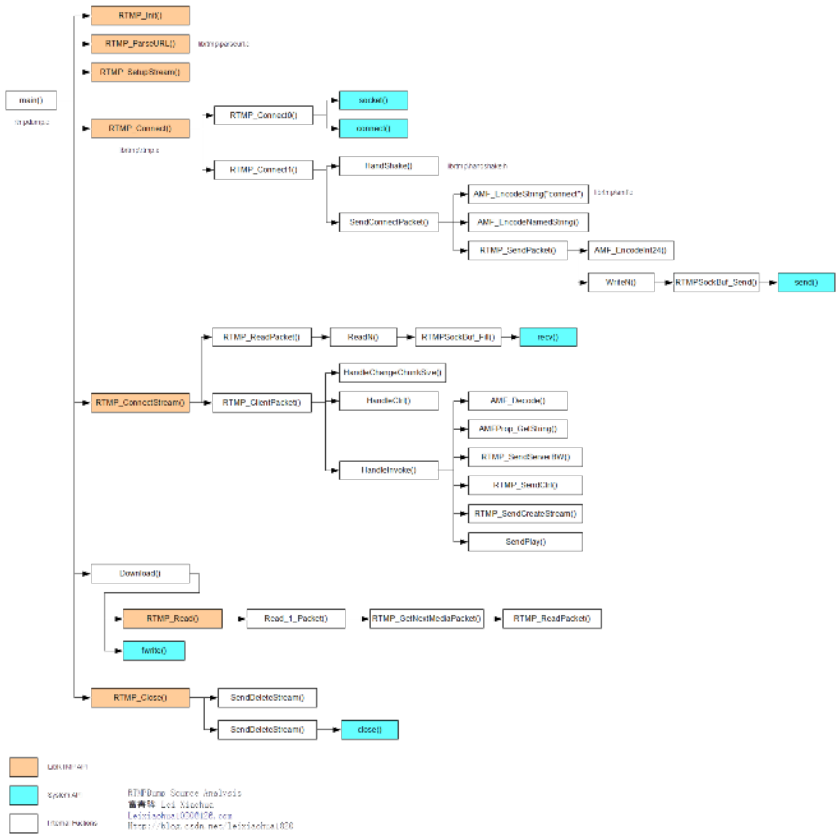
RTMPdump (libRTMP) 源代码分析9： 接收消息 (Message) (接收视音频数据)

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

前文已经分析了 RTMPdump中建立一个NetConnection的过程： RTMPdump 源代码分析 5： 建立一个流媒体连接 （NetConnection部分）

多余的话不多说，下面先来看看RTMP_ConnectStream()，该函数主要用于在NetConnection基础上建立一个NetStream。

RTMP_ConnectStream()

```cpp
//创建流
int
RTMP_ConnectStream(RTMP *r, int seekTime)
{
  RTMPPacket packet = { 0 };

  /* seekTime was already set by SetupStream / SetupURL.
   * This is only needed by ReconnectStream.
   */
  if (seekTime > 0)
    r->Link.seekTime = seekTime;

  r->m_mediaChannel = 0;

  while (!r->m_bPlaying && RTMP_IsConnected(r) && RTMP_ReadPacket(r, &packet))
    {
      if (RTMPPacket_IsReady(&packet))
    {
      if (!packet.m_nBodySize)
        continue;
      if ((packet.m_packetType == RTMP_PACKET_TYPE_AUDIO) ||
          (packet.m_packetType == RTMP_PACKET_TYPE_VIDEO) ||
          (packet.m_packetType == RTMP_PACKET_TYPE_INFO))
        {
          RTMP_Log(RTMP_LOGWARNING, "Received FLV packet before play()! Ignoring.");
          RTMPPacket_Free(&packet);
          continue;
        }
      //处理Packet！
      //----------------
      r->dlg->AppendCInfo("建立网络流：处理收到的数据。开始处理收到的数据");
      //----------------------------
      RTMP_ClientPacket(r, &packet);
      //----------------
      r->dlg->AppendCInfo("建立网络流：处理收到的数据。处理完毕，清除数据。");
      //----------------------------
      RTMPPacket_Free(&packet);
    }
    }

  return r->m_bPlaying;
}
```

乍一看，这个函数的代码量好像挺少的，实际上不然，其复杂度还是挺高的。我觉得比RTMP_Connect()要复杂不少。

其关键就在于这个While()循环。首先，循环的三个条件都满足，就能进行循环。只有出错或者建立网络流（NetStream）的步骤完成后，才能跳出循环。

在这个函数中有两个函数尤为重要：

RTMP_ReadPacket()

RTMP_ClientPacket()

第一个函数的作用是读取通过Socket接收下来的消息（Message）包，但是不做任何处理。第二个函数则是处理消息（Message），并做出响应。这两个函数结合，就可以完成接收消息然后响应消息的步骤。

下面来开一下RTMP_ReadPacket()：

```cpp
//读取收下来的Chunk
int
RTMP_ReadPacket(RTMP *r, RTMPPacket *packet)
{
    //packet 存读取完后的的数据
    //Chunk Header最大值18
  uint8_t hbuf[RTMP_MAX_HEADER_SIZE] = { 0 };
    //header 指向的是从Socket中收下来的数据
  char *header = (char *)hbuf;
  int nSize, hSize, nToRead, nChunk;
  int didAlloc = FALSE;

  RTMP_Log(RTMP_LOGDEBUG2, "%s: fd=%d", __FUNCTION__, r->m_sb.sb_socket);
  //收下来的数据存入hbuf
  if (ReadN(r, (char *)hbuf, 1) == 0)
    {
      RTMP_Log(RTMP_LOGERROR, "%s, failed to read RTMP packet header", __FUNCTION__);
      return FALSE;
    }
  //块类型fmt
```

```
  packet->m_headerType = (hbuf[0] & 0xc0) >> 6;
  //块流ID（2-63）
  packet->m_nChannel = (hbuf[0] & 0x3f);
  header++;
  //块流ID第1字节为0时，块流ID占2个字节
  if (packet->m_nChannel == 0)
    {
      if (ReadN(r, (char *)&hbuf[1], 1) != 1)
    {
      RTMP_Log(RTMP_LOGERROR, "%s, failed to read RTMP packet header 2nd byte",
          __FUNCTION__);
      return FALSE;
    }
      //计算块流ID（64-319）
      packet->m_nChannel = hbuf[1];
      packet->m_nChannel += 64;
      header++;
    }
  //块流ID第1字节为0时，块流ID占3个字节
  else if (packet->m_nChannel == 1)
    {
      int tmp;
      if (ReadN(r, (char *)&hbuf[1], 2) != 2)
    {
      RTMP_Log(RTMP_LOGERROR, "%s, failed to read RTMP packet header 3nd byte",
          __FUNCTION__);
      return FALSE;
    }
      tmp = (hbuf[2] << 8) + hbuf[1];
      //计算块流ID（64-65599）
      packet->m_nChannel = tmp + 64;
      RTMP_Log(RTMP_LOGDEBUG, "%s, m_nChannel: %0x", __FUNCTION__, packet->m_nChannel);
      header += 2;
    }
  //ChunkHeader的大小（4种）
  nSize = packetSize[packet->m_headerType];

  if (nSize == RTMP_LARGE_HEADER_SIZE)  /* if we get a full header the timestamp is absolute */
    packet->m_hasAbsTimestamp = TRUE;    //11字节的完整ChunkMsgHeader的TimeStamp是绝对值

  else if (nSize < RTMP_LARGE_HEADER_SIZE)
    {               /* using values from the last message of this channel */
      if (r->m_vecChannelsIn[packet->m_nChannel])
    memcpy(packet, r->m_vecChannelsIn[packet->m_nChannel],
           sizeof(RTMPPacket));
    }

  nSize--;

  if (nSize > 0 && ReadN(r, header, nSize) != nSize)
    {
      RTMP_Log(RTMP_LOGERROR, "%s, failed to read RTMP packet header. type: %x",
      __FUNCTION__, (unsigned int)hbuf[0]);
      return FALSE;
    }

  hSize = nSize + (header - (char *)hbuf);

  if (nSize >= 3)
    {
    //TimeStamp(注意 BigEndian to SmallEndian)(11，7，3字节首部都有)
      packet->m_nTimeStamp = AMF_DecodeInt24(header);

      /*RTMP_Log(RTMP_LOGDEBUG, "%s, reading RTMP packet chunk on channel %x, headersz %i, timestamp %i, abs timestamp %i", __FUNCTIO
, packet.m_nChannel, nSize, packet.m_nTimeStamp, packet.m_hasAbsTimestamp); */
    //消息长度(11，7字节首部都有)
      if (nSize >= 6)
    {
      packet->m_nBodySize = AMF_DecodeInt24(header + 3);
      packet->m_nBytesRead = 0;
      RTMPPacket_Free(packet);
    //(11，7字节首部都有)
      if (nSize > 6)
        {
          //Msg type ID
          packet->m_packetType = header[6];
          //Msg Stream ID
          if (nSize == 11)
        packet->m_nInfoField2 = DecodeInt32LE(header + 7);
        }
    }
      //Extend TimeStamp
      if (packet->m_nTimeStamp == 0xffffff)
    {
      if (ReadN(r, header + nSize, 4) != 4)
        {
          RTMP_Log(RTMP_LOGERROR, "%s, failed to read extended timestamp",
          __FUNCTION__);
          return FALSE;
        }
      packet->m nTimeStamp = AMF DecodeInt32(header + nSize);
```

```
        hSize += 4;
      }
      }

    RTMP_LogHexString(RTMP_LOGDEBUG2, (uint8_t *)hbuf, hSize);

    if (packet->m_nBodySize > 0 && packet->m_body == NULL)
      {
        if (!RTMPPacket_Alloc(packet, packet->m_nBodySize))
      {
        RTMP_Log(RTMP_LOGDEBUG, "%s, failed to allocate packet", __FUNCTION__);
        return FALSE;
      }
        didAlloc = TRUE;
        packet->m_headerType = (hbuf[0] & 0xc0) >> 6;
      }

    nToRead = packet->m_nBodySize - packet->m_nBytesRead;
    nChunk = r->m_inChunkSize;
    if (nToRead < nChunk)
      nChunk = nToRead;

    /* Does the caller want the raw chunk? */
    if (packet->m_chunk)
      {
        packet->m_chunk->c_headerSize = hSize;
        memcpy(packet->m_chunk->c_header, hbuf, hSize);
        packet->m_chunk->c_chunk = packet->m_body + packet->m_nBytesRead;
        packet->m_chunk->c_chunkSize = nChunk;
      }

    if (ReadN(r, packet->m_body + packet->m_nBytesRead, nChunk) != nChunk)
      {
        RTMP_Log(RTMP_LOGERROR, "%s, failed to read RTMP packet body. len: %lu",
        __FUNCTION__, packet->m_nBodySize);
        return FALSE;
      }

    RTMP_LogHexString(RTMP_LOGDEBUG2, (uint8_t *)packet->m_body + packet->m_nBytesRead, nChunk);

    packet->m_nBytesRead += nChunk;

    /* keep the packet as ref for other packets on this channel */
    if (!r->m_vecChannelsIn[packet->m_nChannel])
      r->m_vecChannelsIn[packet->m_nChannel] = (RTMPPacket *) malloc(sizeof(RTMPPacket));
    memcpy(r->m_vecChannelsIn[packet->m_nChannel], packet, sizeof(RTMPPacket));
    //读取完毕
    if (RTMPPacket_IsReady(packet))
      {
        /* make packet's timestamp absolute */
        if (!packet->m_hasAbsTimestamp)
      packet->m_nTimeStamp += r->m_channelTimestamp[packet->m_nChannel]; /* timestamps seem to be always relative!! */

        r->m_channelTimestamp[packet->m_nChannel] = packet->m_nTimeStamp;

        /* reset the data from the stored packet. we keep the header since we may use it later if a new packet for this channel */
        /* arrives and requests to re-use some info (small packet header) */
        r->m_vecChannelsIn[packet->m_nChannel]->m_body = NULL;
        r->m_vecChannelsIn[packet->m_nChannel]->m_nBytesRead = 0;
        r->m_vecChannelsIn[packet->m_nChannel]->m_hasAbsTimestamp = FALSE;   /* can only be false if we reuse header */
      }
    else
      {
        packet->m_body = NULL; /* so it won't be erased on free */
      }

    return TRUE;
}
```

在这里要注意的是，接收下来的实际上是块（Chunk）而不是消息（Message），因为消息（Message）在网络上传播的时候，实际上要分割成块（Chunk）。

这里解析的就是块（Chunk）

可参考：RTMP规范简单分析

具体的解析代码我就不多说了，直接参考RTMP协议规范就可以了，一个字节一个字节的解析就OK了。

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 流媒体

个人分类： libRTMP

所属专栏： 开源多媒体项目源代码分析