原 视音频数据处理入门：UDP-RTP协议解析

2016年01月31日 21:39:37　阅读数：65360

==========

视音频数据处理入门系列文章：

视音频数据处理入门：RGB、YUV像素数据处理

视音频数据处理入门：PCM音频采样数据处理

视音频数据处理入门：H.264视频码流解析

视音频数据处理入门：AAC音频码流解析

视音频数据处理入门：FLV封装格式解析

视音频数据处理入门：UDP-RTP协议解析

==========

本文介绍网络协议数据的处理程序。网络协议数据在视频播放器中的位置如下所示。

本文中的程序是一个UDP/RTP协议流媒体数据解析器。该程序可以分析UDP协议中的RTP 包头中的内容，以及RTP负载中MPEG-TS封装格式的信息。通过修改该程序可以实现不同的UDP/RTP协议数据处理功能。

原理

MPEG-TS封装格式数据打包为RTP/UDP协议然后发送出去的流程如下图所示。图中首先每7个MPEG-TS Packet打包为一个RTP，然后每个RTP再打包为一个UDP。其中打包RTP的方法就是在MPEG-TS数据前面加上RTP Header，而打包RTP的方法就是在RTP数据前面加上UDP Header。

有关MPEG-TS、RTP、UDP的知识不再详细介绍，可以参考相关的文档了解其中的细节信息。本文记录的程序是一个收取流媒体的程序，因此本文程序的流程和上述发送MPEG-TS的流程正好是相反的。该程序可以通过Socket编程收取UDP包，解析其中的RTP包的信息，然后再解析RTP包中MPEG-TS Packet的信息。

代码

整个程序位于simplest_udp_parser()函数中，如下所示。

```cpp
/**
 * 最简单的视音频数据处理示例
 * Simplest MediaData Test
 *
 * 雷霄骅 Lei Xiaohua
 * leixiaohua1020@126.com
 * 中国传媒大学/数字电视技术
 * Communication University of China / Digital TV Technology
 * http://blog.csdn.net/leixiaohua1020
 *
 * 本项目包含如下几种视音频测试示例：
 *  (1)像素数据处理程序。包含RGB和YUV像素格式处理的函数。
 *  (2)音频采样数据处理程序。包含PCM音频采样格式处理的函数。
 *  (3)H.264码流分析程序。可以分离并解析NALU。
 *  (4)AAC码流分析程序。可以分离并解析ADTS帧。
 *  (5)FLV封装格式分析程序。可以将FLV中的MP3音频码流分离出来。
 *  (6)UDP-RTP协议分析程序。可以将分析UDP/RTP/MPEG-TS数据包。
 *
 * This project contains following samples to handling multimedia data:
 *  (1) Video pixel data handling program. It contains several examples to handle RGB and YUV data.
 *  (2) Audio sample data handling program. It contains several examples to handle PCM data.
 *  (3) H.264 stream analysis program. It can parse H.264 bitstream and analysis NALU of stream.
 *  (4) AAC stream analysis program. It can parse AAC bitstream and analysis ADTS frame of stream.
 *  (5) FLV format analysis program. It can analysis FLV file and extract MP3 audio stream.
 *  (6) UDP-RTP protocol analysis program. It can analysis UDP/RTP/MPEG-TS Packet.
 *
 */
#include <stdio.h>
#include <winsock2.h>

#pragma comment(lib, "ws2_32.lib")
```

```
#pragma pack(1)

/*
 * [memo] FFmpeg stream Command:
 * ffmpeg -re -i sintel.ts -f mpegts udp://127.0.0.1:8880
 * ffmpeg -re -i sintel.ts -f rtp_mpegts udp://127.0.0.1:8880
 */

typedef struct RTP_FIXED_HEADER{
    /* byte 0 */
    unsigned char csrc_len:4;       /* expect 0 */
    unsigned char extension:1;      /* expect 1 */
    unsigned char padding:1;        /* expect 0 */
    unsigned char version:2;        /* expect 2 */
    /* byte 1 */
    unsigned char payload:7;
    unsigned char marker:1;        /* expect 1 */
    /* bytes 2, 3 */
    unsigned short seq_no;
    /* bytes 4-7 */
    unsigned  long timestamp;
    /* bytes 8-11 */
    unsigned long ssrc;            /* stream number is used here. */
} RTP_FIXED_HEADER;

typedef struct MPEGTS_FIXED_HEADER {
    unsigned sync_byte: 8;
    unsigned transport_error_indicator: 1;
    unsigned payload_unit_start_indicator: 1;
    unsigned transport_priority: 1;
    unsigned PID: 13;
    unsigned scrambling_control: 2;
    unsigned adaptation_field_exist: 2;
    unsigned continuity_counter: 4;
} MPEGTS_FIXED_HEADER;



int simplest_udp_parser(int port)
{
    WSADATA wsaData;
    WORD sockVersion = MAKEWORD(2,2);
    int cnt=0;

    //FILE *myout=fopen("output_log.txt","wb+");
    FILE *myout=stdout;

    FILE *fp1=fopen("output_dump.ts","wb+");

    if(WSAStartup(sockVersion, &wsaData) != 0){
        return 0;
    }

    SOCKET serSocket = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
    if(serSocket == INVALID_SOCKET){
        printf("socket error !");
        return 0;
    }

    sockaddr_in serAddr;
    serAddr.sin_family = AF_INET;
    serAddr.sin_port = htons(port);
    serAddr.sin_addr.S_un.S_addr = INADDR_ANY;
    if(bind(serSocket, (sockaddr *)&serAddr, sizeof(serAddr)) == SOCKET_ERROR){
        printf("bind error !");
        closesocket(serSocket);
        return 0;
    }

    sockaddr_in remoteAddr;
    int nAddrLen = sizeof(remoteAddr);

    //How to parse?
    int parse_rtp=1;
    int parse_mpegts=1;

    printf("Listening on port %d\n",port);

    char recvData[10000];
    while (1){

        int pktsize = recvfrom(serSocket, recvData, 10000, 0, (sockaddr *)&remoteAddr, &nAddrLen);
        if (pktsize > 0){
            //printf("Addr:%s\r\n",inet_ntoa(remoteAddr.sin_addr));
            //printf("packet size:%d\r\n",pktsize);
            //Parse RTP
            //
            if(parse_rtp!=0){
                char payload_str[10]={0};
                RTP_FIXED_HEADER rtp_header;
                int rtp_header_size=sizeof(RTP_FIXED_HEADER);
```

```cpp
                int rtp_header_size=sizeof(RTP_FIXED_HEADER);
                //RTP Header
                memcpy((void *)&rtp_header,recvData,rtp_header_size);

                //RFC3551
                char payload=rtp_header.payload;
                switch(payload){
                case 0:
                case 1:
                case 2:
                case 3:
                case 4:
                case 5:
                case 6:
                case 7:
                case 8:
                case 9:
                case 10:
                case 11:
                case 12:
                case 13:
                case 14:
                case 15:
                case 16:
                case 17:
                case 18: sprintf(payload_str,"Audio");break;
                case 31: sprintf(payload_str,"H.261");break;
                case 32: sprintf(payload_str,"MPV");break;
                case 33: sprintf(payload_str,"MP2T");break;
                case 34: sprintf(payload_str,"H.263");break;
                case 96: sprintf(payload_str,"H.264");break;
                default:sprintf(payload_str,"other");break;
                }

                unsigned int timestamp=ntohl(rtp_header.timestamp);
                unsigned int seq_no=ntohs(rtp_header.seq_no);

                fprintf(myout,"[RTP Pkt] %5d| %5s| %10u| %5d| %5d|\n",cnt,payload_str,timestamp,seq_no,pktsize);

                //RTP Data
                char *rtp_data=recvData+rtp_header_size;
                int rtp_data_size=pktsize-rtp_header_size;
                fwrite(rtp_data,rtp_data_size,1,fp1);

                //Parse MPEGTS
                if(parse_mpegts!=0&&payload==33){
                    MPEGTS_FIXED_HEADER mpegts_header;
                    for(int i=0;i<rtp_data_size;i=i+188){
                        if(rtp_data[i]!=0x47)
                            break;
                        //MPEGTS Header
                        //memcpy((void *)&mpegts_header,rtp_data+i,sizeof(MPEGTS_FIXED_HEADER));
                        fprintf(myout,"   [MPEGTS Pkt]\n");
                    }
                }

            }else{
                fprintf(myout,"[UDP Pkt] %5d| %5d|\n",cnt,pktsize);
                fwrite(recvData,pktsize,1,fp1);
            }

            cnt++;
        }
    }
    closesocket(serSocket);
    WSACleanup();
    fclose(fp1);

    return 0;
}
```

上文中的函数调用方法如下所示。

```cpp
simplest_udp_parser(8880);
```

结果

本程序输入为本机的一个端口号，输出为UDP/RTP/MPEG-TS的解析结果。程序开始运行后，可以使用推流软件向本机的udp://127.0.0.1:8880地址进行推流。例如可以使用VLC Media Player的“打开媒体”对话框中的“串流”功能（位于“播放”按钮旁边的小三角按钮的菜单中）。在该功能的对话框中添加一个“RTP / MPEG Transport Stream”的新目标。

也可以使用FFmpeg对本机的8880端口进行推流。下面的命令可以推流UDP封装的MPEG-TS。

```
[plain]
1. ffmpeg -re -i sintel.ts -f mpegts udp://127.0.0.1:8880
```

下面的命令可以推流首先经过RTP封装，然后经过UDP封装的MPEG-TS。

```
[plain]
1. ffmpeg -re -i sintel.ts -f rtp_mpegts udp://127.0.0.1:8880
```

推流之后，本文的程序会通过Socket接收到UDP包并且解析其中的数据。解析的结果如下图所示。

下载

Simplest mediadata test

项目主页

SourceForge： https://sourceforge.net/projects/simplest-mediadata-test/

Github： https://github.com/leixiaohua1020/simplest_mediadata_test

开源中国： http://git.oschina.net/leixiaohua1020/simplest_mediadata_test

CSDN下载地址： http://download.csdn.net/detail/leixiaohua1020/9422409

本项目包含如下几种视音频数据解析示例：
(1)像素数据处理程序。包含RGB和YUV像素格式处理的函数。
(2)音频采样数据处理程序。包含PCM音频采样格式处理的函数。
(3)H.264码流分析程序。可以分离并解析NALU。
(4)AAC码流分析程序。可以分离并解析ADTS帧。
(5)FLV封装格式分析程序。可以将FLV中的MP3音频码流分离出来。
(6)UDP-RTP协议分析程序。可以将分析UDP/RTP/MPEG-TS数据包。

雷霄骅 (Lei Xiaohua)
leixiaohua1020@126.com
http://blog.csdn.net/leixiaohua1020

文章标签： udp rtp mpegts 流媒体
个人分类： 我的开源项目