

原创

七伤之酒

于 2023-06-05 20:34:45 发布

 138

 收藏

版权

文章标签：[服务器](#)[网络](#)[tcp](#)

Rtsp^Q服务器搭建(荷载H264和AAC)

什么是RTSP协议^Q？

RTSP是一个实时传输流协议，是一个应用层的协议

通常说的RTSP包括RTSP协议、RTP协议、RTCP协议

对于这些协议的作用简单的理解如下

RTSP协议 负责服务器与客户端之间的请求与响应

RTP协议 负责传输媒体数据

RTCP协议 在RTP传输过程中提供传输信息

rtsp承载与rtp和rtcp之上，rtsp并不会发送媒体数据，而是使用rtp协议传输

rtp并没有规定发送方式，可以选择udp发送或者tcp发送

RTSP协议详解

rtsp的交互过程就是客户端请求，服务器响应，下面看一看请求和响应的数据格式

RTSP客户端请求

```
1 method url vesion\r\n
2 CSeq: x\r\n
3 xxx\r\n
4 ...
5 \r\n
```

method: 方法，表明这次请求的方法，rtsp定义了很多方法，稍后介绍

url: 格式一般为rtsp://ip:port/session，ip表主机ip，port表端口号，如果不写那么就是默认端口，rtsp的默认端口为554，session表明请求哪一个会话

version: 表示rtsp的版本，现在为RTSP/1.0

CSeq: 序列号，每个RTSP请求和响应都对应一个序列号，序列号是递增的

RTSP服务端的响应格式

```
1 vesion 200 OK\r\n
2 CSeq: x\r\n
3 xxx\r\n
4 ...
5 \r\n
```

version: 表示rtsp的版本，现在为 **RTSP/1.0**

CSeq: 序列号，这个必须与对应请求的序列号相同

RTSP请求的常用方法

方法	描述
OPTIONS	获取服务端提供的可用方法
DESCRIBE	向服务端获取对应会话的媒体描述信息
SETUP	向服务端发起建立请求，建立连接会话
PLAY	向服务端发起播放请求
TEARDOWN	向服务端发起关闭连接会话请求

OPTIONS

C->S

```
1 OPTIONS rtsp://192.168.31.115:8554/live RTSP/1.0\r\n
2 CSeq: 2\r\n
3 \r\n
```

S->C

```
1 RTSP/1.0 200 OK\r\n
2 CSeq: 2\r\n
3 Public: OPTIONS, DESCRIBE, SETUP, TEARDOWN, PLAY\r\n
4 \r\n
```

DESCRIBE

C->S

```
1 DESCRIBE rtsp://192.168.31.115:8554/live RTSP/1.0\r\n
2 CSeq: 3\r\n
3 Accept: application/sdp\r\n
4 \r\n
```

S->C

```
1 RTSP/1.0 200 OK \r\n
2 CSeq: 2\r\n
3 Content-Base: rtsp://192.168.31.115:8554\r\n
4 Content-type: application/sdp\r\n
5 Content-length: 311\r\n
6
7 v=0\r\n
8 o=-91685885859 1 IN IP4 192.168.72.129\r\n
9 t=0 0\r\n
10 a=control:\r\n
11 m=video 0 RTP/AVP 96\r\n
12 a=rtptime:96 H264/90000\r\n
13 a=control track0\r\n
14 m=audio 1 RTP/AVP/TCP 97\r\n
15 a=rtptime:97 mpeg4-generic/44100/2\r\n
16 a=fmtp:97 profile-level-id=1;mode=AAC-hbr;size-length=13;index-length=3;index-delta-length=3;config=1210;\r\n
17 a=control track1\r\n
18
```

SETUP

Rtsp服务器搭建(荷载H264和AAC)

什么是RTSP协议？

RTSP协议详解

RTSP客户端请求

RTSP服务端的响应格式

RTSP请求的常用方法

RTP协议




RTP头部

RTP建立

RtpHeader

RtpPacket

分类专栏

	C++	1篇
	socket	1篇
	C	3篇

C->S

```
1 SETUP rtsp://192.168.31.115:8554/live/track0 RTSP/1.0\r\n
2 CSeq: 4\r\n
3 Transport: RTP/AVP;unicast;client_port=54492-54493\r\n
4 \r\n
```

客户端发送建立请求，请求建立连接会话，准备接收音视频数据

解析一下Transport: RTP/AVP;unicast;client_port=54492-54493\r\n

RTSP/AVP：表示RTP通过UDP发送，如果是RTP/AVP/TCP则表示RTP通过TCP发送

unicast：表示单播，如果是multicast则表示多播

client_port=54492-54493：由于这里希望采用的是RTP OVER UDP，所以客户端发送了两个用于传输数据的端口，客户端已经将这两个端口绑定到两个udp套接字上，54492表示是RTP端口，54493表示RTCP端口(RTP端口为某个偶数，RTCP端口为RTP端口+1)

S->C

```
1 RTSP/1.0 200 OK\r\n
2 CSeq: 4\r\n
3 Transport: RTP/AVP;unicast;client_port=54492-54493;server_port=56400-56401\r\n
4 Session: 66334873\r\n
5 \r\n
```

服务器接收到请求之后，得知客户端要求采用RTP OVER UDP发送数据，单播，客户端用于传输RTP数据的端口为54492，RTCP的端口为54493

服务器也有两个udp套接字，绑定好两个端口，一个用于传输RTP，一个用于传输RTCP，这里的端口号为56400-56401

之后客户端会使用54492-54493这两端口和服务器通过udp传输数据，服务器会使用56400-56401这两端口和这个客户端传输数据

PLAY

C->S

```
1 PLAY rtsp://192.168.31.115:8554/live RTSP/1.0\r\n
2 CSeq: 5\r\n
3 Session: 66334873\r\n
4 Range: npt=0.000-\r\n
5 \r\n
```

客户端请求播放媒体

S->C

```
1 RTSP/1.0 200 OK\r\n
2 CSeq: 5\r\n
3 Range: npt=0.000-\r\n
4 Session: 66334873; timeout=60\r\n
5 \r\n
```

服务器回复之后，会开始使用RTP通过udp向客户端的54492端口发送数据

TEARDOWN

C->S

```
1 TEARDOWN rtsp://192.168.31.115:8554/live RTSP/1.0\r\n
2 CSeq: 6\r\n
3 Session: 66334873\r\n
4 \r\n
```

S->C

```
1 RTSP/1.0 200 OK\r\n
2 CSeq: 6\r\n
3 \r\n
```

RTP协议

RTP头部

[外链图片转存失败,源站可能有防盗链机制,建议将图片保存下来直接上传(img-bTovyteN-1685968165403)(图片/image-20230604215011643.png)]

版本号(V)：2Bit，用来标志使用RTP版本

填充位(S)：1Bit，如果该位置位，则该RTP包的尾部就包含填充的附加字节

扩展位(X)：1Bit，如果该位置位，则该RTP包的固定头部后面就跟着一个扩展头部

CSRC技术器(CC)：4Bit，含有固定头部后面跟着的CSRC的数据

标记位(M)：1Bit，该位的解释由配置文档来承担

载荷类型(PT)：7Bit，标识了RTP载荷的类型

序列号(SN)：16Bit，发送方在每发送完一个RTP包后就将该域的值增加1，可以由该域检测包的丢失及恢复

包的序列。序列号的初始值是随机的

时间戳：32比特，记录了该包中数据的第一个字节的采样时刻

同步源标识符(SSRC)：32比特，同步源就是RTP包源的来源。在同一个RTP会话中不能有两个相同的SSRC值

贡献源列表(CSRC List)：0-15项，每项32比特，这个不常用

RTP建立

RtpHeader

```
1 class RtpHeader
2 {
3 public:
4     /*byte 0*/
5     uint8_t csrcLen : 4;
6     uint8_t extension : 1;
7     uint8_t padding : 1;
8     uint8_t version : 2;
9     /*byte 1*/
10    uint8_t payloadType : 7;
11    uint8_t marker : 1;
12    /*bytes 2,3*/
13    uint16_t seq;
14    /*bytes 4-7*/
15    uint32_t timestamp;
16    /*bytes 8-11*/
17    uint32_t ssrc;
```

```
18 };
```

RtpPacket

```
1 class RtpPacket
2 {
3 public:
4     RtpHeader rtpHeader;
5     uint8_t payload[0];
6 };
```

初始化RTP包

```
1 void rtpHeaderInit(RtpPacket* rtpPacket, uint8_t csrclen, uint8_t extension,
2     uint8_t padding, uint8_t version, uint8_t payloadType, uint8_t marker,
3     uint16_t seq, uint32_t timestamp, uint32_t ssrc);
4
```

```
1 void rtpHeaderInit(RtpPacket* rtpPacket, uint8_t csrclen, uint8_t extension, uint8_t padding, uint8_t version,
2 {
3     rtpPacket->rtpHeader.csrclen = csrclen;
4     rtpPacket->rtpHeader.extension = extension;
5     rtpPacket->rtpHeader.padding = padding;
6     rtpPacket->rtpHeader.version = version;
7     rtpPacket->rtpHeader.payloadType = payloadType;
8     rtpPacket->rtpHeader.marker = marker;
9     rtpPacket->rtpHeader.seq = seq;
10    rtpPacket->rtpHeader.timestamp = timestamp;
11    rtpPacket->rtpHeader.ssrc = ssrc;
12 }
```

以TCP形式发送rtp数据包

```
1 int rtpSendPacketOverTcp(int clientSockfd, struct RtpPacket* rtpPacket, uint32_t dataSize, char channel);
```

```
1 int rtpSendPacketOverTcp(int clientSockfd, struct RtpPacket* rtpPacket, uint32_t dataSize, char channel)
2 {
3
4     rtpPacket->rtpHeader.seq = htons(rtpPacket->rtpHeader.seq);
5     rtpPacket->rtpHeader.timestamp = htonl(rtpPacket->rtpHeader.timestamp);
6     rtpPacket->rtpHeader.ssrc = htonl(rtpPacket->rtpHeader.ssrc);
7
8     uint32_t rtpSize = RTP_HEADER_SIZE + dataSize;
9     char* tempBuf = (char*)malloc(4 + rtpSize);
10    tempBuf[0] = 0x24; // 0x00;
11    tempBuf[1] = channel; // 0x00;
12    tempBuf[2] = (uint8_t)((rtpSize & 0xFF00) >> 8);
13    tempBuf[3] = (uint8_t)((rtpSize & 0xFF));
14    memcpy(tempBuf + 4, (char*)rtpPacket, rtpSize);
15
16    int ret = send(clientSockfd, tempBuf, 4 + rtpSize, 0);
17
18    rtpPacket->rtpHeader.seq = ntohs(rtpPacket->rtpHeader.seq);
19    rtpPacket->rtpHeader.timestamp = ntohl(rtpPacket->rtpHeader.timestamp);
20    rtpPacket->rtpHeader.ssrc = ntohl(rtpPacket->rtpHeader.ssrc);
21
22    free(tempBuf);
23    tempBuf = NULL;
24
25    return ret;
26 }
```

以UDP形式发送rtp数据包

```
1 int rtpSendPacketOverUdp(int serverRtpSockfd, const char* ip, int16_t port, struct RtpPacket* rtpPacket, uint32_t
```

```
1 int rtpSendPacketOverUdp(int serverRtpSockfd, const char* ip, int16_t port, struct RtpPacket* rtpPacket, uint32_t
2 {
3
4     struct sockaddr_in addr;
5     int ret;
6
7     addr.sin_family = AF_INET;
8     addr.sin_port = htons(port);
9     addr.sin_addr.s_addr = inet_addr(ip);
10
11    rtpPacket->rtpHeader.seq = htons(rtpPacket->rtpHeader.seq);
12    rtpPacket->rtpHeader.timestamp = htonl(rtpPacket->rtpHeader.timestamp);
13    rtpPacket->rtpHeader.ssrc = htonl(rtpPacket->rtpHeader.ssrc);
14
15    ret = sendto(serverRtpSockfd, (char*)rtpPacket, dataSize + RTP_HEADER_SIZE, 0,
16        (struct sockaddr*)&addr, sizeof(addr));
17
18    rtpPacket->rtpHeader.seq = ntohs(rtpPacket->rtpHeader.seq);
19    rtpPacket->rtpHeader.timestamp = ntohl(rtpPacket->rtpHeader.timestamp);
20    rtpPacket->rtpHeader.ssrc = ntohl(rtpPacket->rtpHeader.ssrc);
21
22    return ret;
23
24 }
```

一、建立套接字

程序从进入main函数之后就创建服务器TCP套接字，绑定端口，然后开始监听端口

```
1 //建立套接字
2 int ServerSocket;
3 int ServerRtpSocket, ServerRtpSocket;
4 //创建TCP套接字
5 ServerSocket = CreateTcpSocket();
6 if (ServerSocket < 0)
7 {
8     cout << "TCP create fail !!!" << endl;
9     exit(0);
10 }
11 //绑定端口和地址
12 if (BindSocketAddr(ServerSocket, "0.0.0.0", SERVER_PORT) < 0)
13 {
14     cout << "bind fail !!!" << endl;
15     exit(0);
16 }
17 //监听端口
18 if (listen(ServerSocket, 5) < 0)
19 {
20     cout << "listen fail !!!" << endl;
21     exit(0);
22 }
23 cout << "rtsp://" << SERVER_IP << ":" << SERVER_PORT << endl;
```

二、接受客户端连接

在while循环中接受客户端消息,并利用函数进行处理

```
1 while (true)
2 {
3     int ClientSocket;
4     char ClientIp[40];
5     int ClientPort;
6     //接收客户端消息
7     ClientSocket = AcceptClient(ServerSocket, ClientIp, &ClientPort);
8     if (ClientSocket < 0)
9     {
10         printf("failed to accept client\n");
11         return -1;
12     }
13     //打印客户端信息
14     cout << "accept client;client ip:" << ClientIp << ",client port:" << ClientPort << endl;
15     //接收消息并做出响应
16     doClient(ClientSocket, ClientIp, ClientPort);
17 }
```

三、解析请求

```
1 while (true) {
2     int recvLen;
3
4     recvLen = recv(clientSockfd, rBuf, BUF_MAX_SIZE, 0);
5     if (recvLen <= 0) {
6         break;
7     }
8
9     rBuf[recvLen] = '\0';
10    printf("Accept request rBuf = %s \n", rBuf);
11
12    const char* sep = "\n";
13    //获取第一行数据
14    char* line = strtok(rBuf, sep);
15    while (line) {
16        if (strstr(line, "OPTIONS") ||
17            strstr(line, "DESCRIBE") ||
18            strstr(line, "SETUP") ||
19            strstr(line, "PLAY")) {
20            if (sscanf(line, "%s %s %s\n", method, url, version) != 3) {
21                // error
22            }
23        }
24        else if (strstr(line, "CSeq")) {
25            if (sscanf(line, "CSeq: %d\n", &CSeq) != 1) {
26                // error
27            }
28        }
29        else if (!strcmp(line, "Transport:", strlen("Transport:"))) {
30            // Transport: RTP/AVP/UDP;unicast;client_port=13358-13359
31            // Transport: RTP/AVP;unicast;client_port=13358-13359
32
33            if (sscanf(line, "Transport: RTP/AVP/TCP;unicast;interleaved=0-1\n") != 0) {
34                // error
35                printf("parse Transport error \n");
36            }
37        }
38        //获取下一行数据
39        line = strtok(NULL, sep);
40    }
```

四、处理请求

解析完客户端命令后, 会调用相应的请求, 处理完之后将要发送的消息打印到sBuf发送给客户端

```
1 if (!strcmp(method, "OPTIONS")) {
2     if (handleCmd_OPTIONS(sBuf, CSeq))
3     {
4         printf("failed to handle options\n");
5         break;
6     }
7 }
8 else if (!strcmp(method, "DESCRIBE")) {
9     if (handleCmd_DESCRIBE(sBuf, CSeq, url))
10    {
11        printf("failed to handle describe\n");
12        break;
13    }
14 }
15 else if (!strcmp(method, "SETUP")) {
16     if (handleCmd_SETUP(sBuf, CSeq))
17     {
18        printf("failed to handle setup\n");
19        break;
20    }
21 }
22 else if (!strcmp(method, "PLAY")) {
23     if (handleCmd_PLAY(sBuf, CSeq))
24     {
25        printf("failed to handle play\n");
26        break;
27    }
28 }
29 else {
30     printf("Undefined method = %s \n", method);
31     break;
32 }
33 printf("Response sBuf = %s \n", sBuf);
34 //向客户端回复消息
35 send(clientSockfd, sBuf, strlen(sBuf), 0);
```

五、AAC RTP打包发送

接受到"PLAY"消息后, 服务器开始循环发送AAC数据

```
1 while (true)
2 {
3     //读取ADTS头部
4     ret = fread(frame, 1, 7, fp);
5     if (ret <= 0)
6     {
7         printf("fread err\n");
8         break;
9     }
10    printf("fread ret=%d \n", ret);
11    //解析头部
12    if (parseAdtsHeader(frame, &adtsHeader) < 0)
13    {
```

```

14     printf("parseAdtsHeader err\n");
15     break;
16 }
17 //读取一帧
18 ret = fread(frame, 1, adtsHeader.aacFrameLength - 7, fp);
19 if (ret <= 0)
20 {
21     printf("fread err\n");
22     break;
23 }
24 //Rtp打包发送
25 rtpSendAACFrame(clientSockfd,
26                 rtpPacket, frame, adtsHeader.aacFrameLength - 7);
27 usleep(23223); //1000/43.06 * 1000
28 }

```

六、H264 RTP打包发送

接收到"PLAY"消息后，服务器开始循环发送H264数据

```

1 while (true) {
2     frameSize = getFrameFromH264File(fp, frame, 500000);
3     if (frameSize < 0)
4     {
5         printf("Read %s end , frameSize=%d \n", H264_FILE_NAME, frameSize);
6         break;
7     }
8
9     if (startCode3(frame))
10        startCode = 3;
11    else
12        startCode = 4;
13
14    frameSize -= startCode;
15    rtpSendH264Frame(clientSockfd, rtpPacket, frame + startCode, frameSize);
16
17    rtpPacket->rtpHeader.timestamp += 90000 / 25;
18    usleep(40000); //1000/25 * 1000

```

函数实现

建立aacheader

```

1 struct AdtsHeader
2 {
3     unsigned int syncword; //12 bit 同步字 '1111 1111 1111', 说明一个ADTS帧的开始
4     unsigned int id; //1 bit MPEG 标示符, 0 for MPEG-4, 1 for MPEG-2
5     unsigned int layer; //2 bit 总是'00'
6     unsigned int protectionAbsent; //1 bit 1表示没有crc, 0表示有crc
7     unsigned int profile; //1 bit 表示使用哪个级别的AAC
8     unsigned int samplingFreqIndex; //4 bit 表示使用的采样频率
9     unsigned int privateBit; //1 bit
10    unsigned int channelCfg; //3 bit 表示声道数
11    unsigned int originalCopy; //1 bit
12    unsigned int home; //1 bit
13
14    /*下面的为改变的参数即每一帧都不同*/
15    unsigned int copyrightIdentificationBit; //1 bit
16    unsigned int copyrightIdentificationStart; //1 bit
17    unsigned int aacFrameLength; //13 bit 一个ADTS帧的长度包括ADTS头和AAC原始流
18    unsigned int adtsBufferFullness; //11 bit 0x7FF 说明是码率可变的码流
19
20    /* number_of_raw_data_blocks_in_frame
21     * 表示ADTS帧中有number_of_raw_data_blocks_in_frame + 1个AAC原始帧
22     * 所以说number_of_raw_data_blocks_in_frame == 0
23     * 表示说ADTS帧中有一个AAC数据块并不是说没有。(一个AAC原始帧包含一段时间内1024个采样及相关数据)
24     */
25    unsigned int numberOfRawDataBlockInFrame; //2 bit
26 };

```

解析aacheader

```

1 static int parseAdtsHeader(uint8_t* in, struct AdtsHeader* res)
2 {
3     static int frame_number = 0;
4     memset(res, 0, sizeof(*res));
5
6     if ((in[0] == 0xFF) && ((in[1] & 0xF0) == 0xF0))
7     {
8         r e s->id = ((unsigned int)in[1] & 0x08) >> 3;
9         r e s->layer = ((unsigned int)in[1] & 0x06) >> 1;
10        r e s->protectionAbsent = ((unsigned int)in[1] & 0x01);
11        r e s->profile = ((unsigned int)in[2] & 0xc0) >> 6;
12        r e s->samplingFreqIndex = ((unsigned int)in[2] & 0x3c) >> 2;
13        r e s->privateBit = ((unsigned int)in[2] & 0x02) >> 1;
14        r e s->channelCfg = (((unsigned int)in[2] & 0x01) << 2) | (((unsigned int)in[3] & 0xc0) >> 6);
15        r e s->originalCopy = ((unsigned int)in[3] & 0x20) >> 5;
16        r e s->home = ((unsigned int)in[3] & 0x10) >> 4;
17        r e s->copyrightIdentificationBit = ((unsigned int)in[3] & 0x08) >> 3;
18        r e s->copyrightIdentificationStart = ((unsigned int)in[3] & 0x04) >> 2;
19        r e s->aacFrameLength = (((unsigned int)in[3] & 0x03) << 11) |
20        (((unsigned int)in[4] & 0xFF) << 3) |
21        (((unsigned int)in[5] & 0xE0) >> 5);
22        r e s->adtsBufferFullness = (((unsigned int)in[5] & 0x1f) << 6 |
23        (((unsigned int)in[6] & 0xfc) >> 2);
24        r e s->numberOfRawDataBlockInFrame = ((unsigned int)in[6] & 0x03);
25
26        return 0;
27    }
28    else
29    {
30        printf("failed to parse adts header\n");
31        return -1;
32    }
33 }

```

用rtp格式打包并发送AAC音频流数据

```

1 static int rtpSendAACFrame(int clientSockfd,
2                             struct RtpPacket* rtpPacket, uint8_t* frame, uint32_t frameSize) {
3     int ret;
4
5     rtpPacket->payload[0] = 0x00;
6     rtpPacket->payload[1] = 0x10;
7     rtpPacket->payload[2] = (frameSize & 0x1FE0) >> 5; //高8位
8     rtpPacket->payload[3] = (frameSize & 0x1f) << 3; //低5位
9
10    memcpy(rtpPacket->payload + 4, frame, frameSize);
11
12
13    ret = rtpSendPacketOverTcp(clientSockfd, rtpPacket, frameSize + 4, 0x02);

```

```

14     if (ret < 0)
15     {
16         printf("failed to send rtp packet\n");
17         return -1;
18     }
19 }
20
21 rtpPacket->rtpHeader.seq++;
22
23 /*
24  * 如果采样频率是44100
25  * 一般AAC每个1024个采样为一帧
26  * 所以一秒就有 44100 / 1024 = 43帧
27  * 时间增量就是 44100 / 43 = 1025
28  * 一帧的时间为 1 / 43 = 23ms
29  */
30 rtpPacket->rtpHeader.timestamp += 1025;
31
32 return 0;
33 }

```

创建TCP套接字

```

1 static int CreateTcpSocket()
2 {
3     int sockfd;
4     int on=1;
5     sockfd = socket(AF_INET, SOCK_STREAM, 0);
6     if (sockfd < 0)
7         return -1;
8     setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, (const char*)&on, sizeof(on));
9     return sockfd;
10 }

```

创建UDP套接字

```

1 static int CreateUdpSocket()
2 {
3     int sockfd;
4     int on = 1;
5     sockfd = socket(AF_INET, SOCK_DGRAM, 0);
6     if (sockfd < 0)
7         return -1;
8     setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, (const char*)&on, sizeof(on));
9     return sockfd;
10 }

```

绑定端口和地址

```

1 static int BindSocketAddr(int sockfd,const char *ip,int port)
2 {
3     sockadd_in addr;
4     addr.sin_family = AF_INET;
5     addr.sin_port = htons(port);
6     addr.sin_addr.s_addr = inet_addr(ip);
7     if (bind(sockfd, (sockaddr*)&addr, sizeof(addr))<0)
8         return -1;
9     return 0;
10 }

```

连接客户端并接收客户端信息

```

1 static int AcceptClient(int sockfd,char *ip,int *port)
2 {
3     int clientfd;
4     socklen_t len = 0;
5     sockadd_in addr;
6     memset(&addr, 0, sizeof(addr));
7     len = sizeof(addr);
8     clientfd = accept(sockfd, (sockaddr*)&addr, &len);
9     if (clientfd < 0)
10         return -1;
11     strcpy(ip, inet_ntoa(addr.sin_addr));
12     *port = ntohs(addr.sin_port);
13     return clientfd;
14 }

```

判断是不是非h264码流(0 0 0 1)

```

1 static inline int startCode3(char* buf)
2 {
3     if (buf[0] == 0 && buf[1] == 0 && buf[2] == 1)
4         return -1;
5     else
6         return 0;
7 }

```

判断是不是非h264码流(0 0 0 0 1)

```

1 static inline int startCode4(char* buf)
2 {
3     if (buf[0] == 0 && buf[1] == 0 && buf[2] == 0 && buf[3]==1)
4         return -1;
5     else
6         return 0;
7 }

```

找下一段h264数据

```

1 static char* findNextStartCode(char* buf, int len)
2 {
3     int i;
4     if (len < 3)
5         return NULL;
6     for (i = 0; i < len - 3; i++)
7     {
8         if (startCode3(buf) || startCode4(buf))
9             return buf;
10         ++buf;
11     }
12     if (startCode3(buf))
13         return buf;
14     return NULL;
15 }

```

获得h264码流大小

```

1 static int getFrameFromH264File(int fd,char*frame,int size)
2 {
3     int rSize, frameSize;

```

```

char* nextStartCode;
if (fd < 0)
    return fd;
rSize = read(fd, frame, size);
if (!startCode3(frame) && !startCode4(frame))
    return -1;
nextStartCode = findNextStartCode(frame + 3, rSize - 3);
if (!nextStartCode)
    return -1;
else
{
    frameSize = nextStartCode - frame;
    lseek(fd, frameSize - rSize, SEEK_CUR);
}
return frameSize;
}

```

用RTP格式打包并发送H264视频流数据

```

1 static int rtpSendH264Frame(int clientSockfd,
2                             struct RtpPacket* rtpPacket, char* frame, uint32_t frameSize)
3 {
4
5     uint8_t naluType; // nalu第一个字节
6     int sendByte = 0;
7     int ret;
8
9     naluType = frame[0];
10
11     printf("%s frameSize=%d\n", __FUNCTION__, frameSize);
12
13     if (frameSize <= RTP_MAX_PKT_SIZE) // nalu长度小于最大包长: 单一NAL单元模式
14     {
15
16         /*      0 1 2 3 4 5 6 7 8 9
17         +-+-+-+-+-+-+-+-+
18         |F|NRI| Type   | a single NAL unit ... |
19         +-+-+-+-+-+-+-+-+
20
21         memcpy(rtpPacket->payload, frame, frameSize);
22         ret = rtpSendPacketOverTcp(clientSockfd, rtpPacket, frameSize, 0x00);
23         if (ret < 0)
24             return -1;
25
26         rtpPacket->rtpHeader.seq++;
27         sendByte += ret;
28         if ((naluType & 0x1F) == 7 || (naluType & 0x1F) == 8) // 如果是SPS、PPS就不需要加时间戳
29         {
30
31         }
32     }
33 }
34 else // nalu长度小于最大包长: 分片模式
35 {
36
37     /*      0      1      2
38     0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3
39     +-+-+-+-+-+-+-+-+
40     |FU indicator| FU header | FU payload ... |
41     +-+-+-+-+-+-+-+-+
42
43
44     /*      FU Indicator
45     /*      0 1 2 3 4 5 6 7
46     /*      +-+-+-+-+-+-+-+-+
47     /*      |F|NRI| Type   |
48     /*      +-----+
49
50
51
52     /*      FU Header
53     /*      0 1 2 3 4 5 6 7
54     /*      +-+-+-+-+-+-+-+-+
55     /*      |S|E|R| Type   |
56     /*      +-----+
57
58
59
60     int pktNum = frameSize / RTP_MAX_PKT_SIZE; // 有几个完整的包
61     int remainPktSize = frameSize % RTP_MAX_PKT_SIZE; // 剩余不完整包的大小
62     int i, pos = 1;
63
64     // 发送完整的包
65     for (i = 0; i < pktNum; i++)
66     {
67         rtpPacket->payload[0] = (naluType & 0x60) | 28;
68         rtpPacket->payload[1] = naluType & 0x1F;
69
70         if (i == 0) // 第一包数据
71             rtpPacket->payload[1] |= 0x80; // start
72         else if (remainPktSize == 0 && i == pktNum - 1) // 最后一包数据
73             rtpPacket->payload[1] |= 0x40; // end
74
75         memcpy(rtpPacket->payload + 2, frame + pos, RTP_MAX_PKT_SIZE);
76         ret = rtpSendPacketOverTcp(clientSockfd, rtpPacket, RTP_MAX_PKT_SIZE + 2, 0x00);
77         if (ret < 0)
78             return -1;
79
80         rtpPacket->rtpHeader.seq++;
81         sendByte += ret;
82         pos += RTP_MAX_PKT_SIZE;
83     }
84
85     // 发送剩余的数据
86     if (remainPktSize > 0)
87     {
88         rtpPacket->payload[0] = (naluType & 0x60) | 28;
89         rtpPacket->payload[1] = naluType & 0x1F;
90         rtpPacket->payload[1] |= 0x40; // end
91
92         memcpy(rtpPacket->payload + 2, frame + pos, remainPktSize + 2);
93         ret = rtpSendPacketOverTcp(clientSockfd, rtpPacket, remainPktSize + 2, 0x00);
94         if (ret < 0)
95             return -1;
96
97         rtpPacket->rtpHeader.seq++;
98         sendByte += ret;
99     }
100 }
101
102 return sendByte;
103
104

```

```
105 }
```

获取报文第一行数据

```
1 static char* GetLineFromBuf(char* rbuf, char* line)
2 {
3     while (*rbuf != '\n')
4     {
5         *line = *rbuf;
6         line++;
7         rbuf++;
8     }
9     *line = '\n';
10    ++line;
11    *line = '\0';
12    ++rbuf;
13    return rbuf;
14 }
```

对于客户端消息的回复

```
1 static int handleCmd_OPTIONS(char* result, int cseq)
2 {
3     sprintf(result, "RTSP/1.0 200 OK\r\n"
4         "CSeq: %d\r\n"
5         "Public: OPTIONS, DESCRIBE, SETUP, PLAY\r\n"
6         "\r\n",
7         c s e q);
8     return 0;
9 }
10 static int handleCmd_DESCRIBE(char* result, int cseq, char* url)
11 {
12     char sdp[500];
13     char localIp[100];
14     sscanf(url, "rtsp://[%s]:", localIp);
15     sprintf(sdp, "v=0\r\n"
16         "o=-9%d 1 IN IP4 %s\r\n"
17         "t=0 0\r\n"
18         "a=control:* \r\n"
19         "m=video 0 RTP/AVP 96 \r\n"
20         "a=rtptime:96 H264/90000 \r\n"
21         "a=control:track0 \r\n"
22         "m=audio 1 RTP/AVP/TCP 97 \r\n"
23         "a=rtptime:97 mpeg4-generic/44100/2 \r\n"
24         "a=fmtp:97 profile-level-id=1;mode=AAC-Hbr;size-length=13;index-length=3;index-delta-length=3;config=1210;\r\n"
25         "a=control:track1 \r\n",
26         time(NULL), localIp);
27     sprintf(result, "RTSP/1.0 200 OK \r\n"
28         "CSeq: %d\r\n"
29         "Content-Base: %s\r\n"
30         "Content-type: application/sdp\r\n"
31         "Content-length: %d\r\n"
32         "\r\n"
33         "%s", cseq, url, (int)strlen(sdp), sdp);
34     return 0;
35 }
36 static int handleCmd_SETUP(char* result, int cseq)
37 {
38     if (cseq == 3) {
39         sprintf(result, "RTSP/1.0 200 OK\r\n"
40             "CSeq: %d\r\n"
41             "Transport: RTP/AVP/TCP;unicast;interleaved=0-1\r\n"
42             "Session: 66334873\r\n"
43             "\r\n",
44             c s e q);
45     }
46     else if (cseq == 4) {
47         sprintf(result, "RTSP/1.0 200 OK\r\n"
48             "CSeq: %d\r\n"
49             "Transport: RTP/AVP/TCP;unicast;interleaved=2-3\r\n"
50             "Session: 66334873\r\n"
51             "\r\n",
52             c s e q);
53     }
54     return 0;
55 }
56 static int handleCmd_PLAY(char* result, int cseq)
57 {
58     sprintf(result, "RTSP/1.0 200 OK\r\n"
59         "CSeq: %d\r\n"
60         "Range: npt=0.000- \r\n"
61         "Session: 66334873; timeout=60\r\n"
62         "\r\n", cseq);
63     return 0;
64 }
```

接收并回复消息做出相应的响应

```
1 static void doClient(int clientSockfd, const char* clientIP, int clientPort) {
2
3     char method[40];
4     char url[100];
5     char version[40];
6     int CSeq;
7
8     char* rBuf = (char*)malloc(BUF_MAX_SIZE);
9     char* sBuf = (char*)malloc(BUF_MAX_SIZE);
10
11     while (true) {
12         int recvLen;
13
14         r e c v L e n = recv(clientSockfd, rBuf, BUF_MAX_SIZE, 0);
15         if (recvLen <= 0) {
16             break;
17         }
18
19         r B u f[recvLen] = '\0';
20         printf("Accept request rBuf = %s \n", rBuf);
21
22         const char* sep = "\n";
23
24         char* line = strtok(rBuf, sep);
25         while (line) {
26             if (strstr(line, "OPTIONS") ||
27                 strstr(line, "DESCRIBE") ||
28                 strstr(line, "SETUP") ||
29                 strstr(line, "PLAY")) {
30                 if (sscanf(line, "%s %s %s\r\n", method, url, version) != 3) {
31                     // error
32                 }
33             }
34             else if (strstr(line, "CSeq")) {
```



```

35     if (sscanf(line, "CSeq: %d\n", &CSeq) != 1) {
36         // error
37     }
38 }
39 else if (!strcmp(line, "Transport:", strlen("Transport:"))) {
40     // Transport: RTP/AVP/UDP;unicast;client_port=13358-13359
41     // Transport: RTP/AVP;unicast;client_port=13358-13359
42
43     if (sscanf(line, "Transport: RTP/AVP/TCP;unicast;interleaved=0-1\n") != 0) {
44         // error
45         printf("parse Transport error \n");
46     }
47 }
48 l i n e = strtok(NULL, sep);
49 }
50
51 if (!strcmp(method, "OPTIONS")) {
52     if (handleCmd_OPTIONS(sBuf, CSeq))
53     {
54         printf("failed to handle options\n");
55         break;
56     }
57 }
58 else if (!strcmp(method, "DESCRIBE")) {
59     if (handleCmd_DESCRIBE(sBuf, CSeq, url))
60     {
61         printf("failed to handle describe\n");
62         break;
63     }
64 }
65 else if (!strcmp(method, "SETUP")) {
66     if (handleCmd_SETUP(sBuf, CSeq))
67     {
68         printf("failed to handle setup\n");
69         break;
70     }
71 }
72 else if (!strcmp(method, "PLAY")) {
73     if (handleCmd_PLAY(sBuf, CSeq))
74     {
75         printf("failed to handle play\n");
76         break;
77     }
78 }
79 else {
80     printf("Undefined method = %s \n", method);
81     break;
82 }
83 printf("Response sBuf = %s \n", sBuf);
84
85 send(clientSockfd, sBuf, strlen(sBuf), 0);
86
87 //开始播放，发送RTP包
88 if (!strcmp(method, "PLAY")) {
89
90     s t d::thread t1([&]) {
91
92         int frameSize, startCode;
93         char* frame = new char [500000];
94         R t p P a c k e t* rtpPacket = new RtpPacket[500000];
95         int fp = open(H264_FILE_NAME, O_RDONLY);
96         if (!fp) {
97             printf("Read %s fail\n", H264_FILE_NAME);
98             return;
99         }
100         rtpHeaderInit(rtpPacket, 0, 0, 0, RTP_VESION, RTP_PAYLOAD_TYPE_H264, 0,
101             0, 0, 0x88923423);
102
103         printf("start play\n");
104
105         while (true) {
106             f r a m e S i z e = getFrameFromH264File(fp, frame, 500000);
107             if (frameSize < 0)
108             {
109                 printf("Read %s end , frameSize=%d \n", H264_FILE_NAME, frameSize);
110                 break;
111             }
112
113             if (startCode3(frame))
114                 s t a r t C o d e = 3;
115             else
116                 s t a r t C o d e = 4;
117
118             f r a m e S i z e -= startCode;
119             rtpSendH264Frame(clientSockfd, rtpPacket, frame + startCode, frameSize);
120
121             r t p P a c k e t->rtpHeader.timestamp += 90000 / 25;
122             usleep(40000); // 1000/25 * 1000
123         }
124         free(frame);
125         free(rtpPacket);
126     });
127
128     s t d::thread t2([&]) {
129         struct AdtsHeader adtsHeader;
130         struct RtpPacket* rtpPacket;
131         uint8_t* frame;
132         int ret;
133
134         F I L E* fp = fopen(AAC_FILE_NAME, "rb");
135         if (!fp) {
136             printf("Read %s fail\n", AAC_FILE_NAME);
137             return;
138         }
139
140         f r a m e = (uint8_t*)malloc(5000);
141         r t p P a c k e t = (struct RtpPacket*)malloc(5000);
142
143         rtpHeaderInit(rtpPacket, 0, 0, 0, RTP_VESION, RTP_PAYLOAD_TYPE_AAC, 1, 0, 0, 0x32411);
144
145         while (true)
146         {
147             //读取ADTS头部
148             r e t = fread(frame, 1, 7, fp);
149             if (ret <= 0)
150             {
151                 printf("fread err\n");
152                 break;
153             }
154             printf("#fread ret=%d \n", ret);
155             //解析头部
156             if (parseAdtsHeader(frame, &adtsHeader) < 0)

```

```

158         {
159             printf("parseAdtsHeader err\n");
160             break;
161         }
162         //读取一帧
163         r e t = fread(frame, 1, adtsHeader.aacFrameLength - 7, fp);
164         if (ret <= 0)
165         {
166             printf("fread err\n");
167             break;
168         }
169         //Rtp打包发送
170         rtpSendAACFrame(clientSockfd,
171             n t p a c k e t, frame, adtsHeader.aacFrameLength - 7);
172         usleep(23223); //1000/43.06 * 1000
173     }
174
175     free(frame);
176     free(rtpPacket);
177 });
178
179     t 1.join();
180     t 2.join();
181
182     break;
183 }
184
185     memset(method, 0, sizeof(method) / sizeof(char));
186     memset(url, 0, sizeof(url) / sizeof(char));
187     C S e q = 0;
188
189
190 }
191
192     close(clientSockfd);
193     free(rBuf);
194     free(sBuf);
195 }
196 }
197

```

源代码

rtp.h

```

1 #include<stdint.h>
2
3 #define RTP_VERSION      2
4
5 #define RTP_PAYLOAD_TYPE_H264  96
6 #define RTP_PAYLOAD_TYPE_AAC  97
7
8 #define RTP_HEADER_SIZE      12
9 #define RTP_MAX_PKT_SIZE    1400
10
11 class RtpHeader
12 {
13     public:
14     /*byte 0*/
15     uint8_t csrclen : 4;
16     uint8_t extension : 1;
17     uint8_t padding : 1;
18     uint8_t version : 2;
19     /*byte 1*/
20     uint8_t payloadType : 7;
21     uint8_t marker : 1;
22     /*bytes 2,3*/
23     uint16_t seq;
24     /*bytes 4-7*/
25     uint32_t timestamp;
26     /*bytes 8-11*/
27     uint32_t ssrc;
28 };
29
30 class RtpPacket
31 {
32     public:
33     RtpHeader rtpHeader;
34     uint8_t payload[0];
35 };
36
37 //初始化Rtp包
38 void rtpHeaderInit(RtpPacket* rtpPacket, uint8_t csrclen, uint8_t extension,
39     uint8_t padding, uint8_t version, uint8_t payloadType, uint8_t marker,
40     uint16_t seq, uint32_t timestamp, uint32_t ssrc);
41
42 //以tcp形式发送Rtp数据包
43 int rtpSendPacketOverTcp(int clientSockfd, struct RtpPacket* rtpPacket, uint32_t dataSize, char channel);
44
45 //以udp形式发送Rtp数据包
46 int rtpSendPacketOverUdp(int serverRtpSockfd, const char* ip, int16_t port, struct RtpPacket* rtpPacket, uint32_t
47

```

rtp.cpp

```

1 #include<sys/socket.h>
2 #include<arpa/inet.h>
3 #include<cstdlib>
4 #include<string.h>
5 #include"rtp.h"
6
7 void rtpHeaderInit(RtpPacket* rtpPacket, uint8_t csrclen, uint8_t extension, uint8_t padding, uint8_t version,
8 {
9     rtpPacket->rtpHeader.csrclen = csrclen;
10     rtpPacket->rtpHeader.extension = extension;
11     rtpPacket->rtpHeader.padding = padding;
12     rtpPacket->rtpHeader.version = version;
13     rtpPacket->rtpHeader.payloadType = payloadType;
14     rtpPacket->rtpHeader.marker = marker;
15     rtpPacket->rtpHeader.seq = seq;
16     rtpPacket->rtpHeader.timestamp = timestamp;
17     rtpPacket->rtpHeader.ssrc = ssrc;
18 }
19
20 int rtpSendPacketOverTcp(int clientSockfd, struct RtpPacket* rtpPacket, uint32_t dataSize, char channel)
21 {
22
23     rtpPacket->rtpHeader.seq = htons(rtpPacket->rtpHeader.seq);
24     rtpPacket->rtpHeader.timestamp = htonl(rtpPacket->rtpHeader.timestamp);
25     rtpPacket->rtpHeader.ssrc = htonl(rtpPacket->rtpHeader.ssrc);
26
27     uint32_t rtpSize = RTP_HEADER_SIZE + dataSize;
28     char* tempBuf = (char*)malloc(4 + rtpSize);
29     tempBuf[0] = 0x24; // $
30     tempBuf[1] = channel; // 0x00; //表示通道
31     tempBuf[2] = (uint8_t)((rtpSize & 0xFF00) >> 8);
32

```

```

32 tempBuf[3] = (uint8_t)((rtpSize & 0xFF);
33 memcpy(tempBuf + 4, (char*)rtpPacket, rtpSize);
34
35 int ret = send(clientSockfd, tempBuf, 4 + rtpSize, 0);
36
37 rtpPacket->rtpHeader.seq = ntohs(rtpPacket->rtpHeader.seq);
38 rtpPacket->rtpHeader.timestamp = ntohl(rtpPacket->rtpHeader.timestamp);
39 rtpPacket->rtpHeader.ssrc = ntohl(rtpPacket->rtpHeader.ssrc);
40
41 free(tempBuf);
42 tempBuf = NULL;
43
44 return ret;
45 }
46 int rtpSendPacketOverUdp(int serverRtpSockfd, const char* ip, int16_t port, struct RtpPacket* rtpPacket, uint32_t rtpSize)
47 {
48     struct sockaddr_in addr;
49     int ret;
50
51     addr.sin_family = AF_INET;
52     addr.sin_port = htons(port);
53     addr.sin_addr.s_addr = inet_addr(ip);
54
55     rtpPacket->rtpHeader.seq = htons(rtpPacket->rtpHeader.seq); //从主机字节序转为网络字节序
56     rtpPacket->rtpHeader.timestamp = htonl(rtpPacket->rtpHeader.timestamp);
57     rtpPacket->rtpHeader.ssrc = htonl(rtpPacket->rtpHeader.ssrc);
58
59     ret = sendto(serverRtpSockfd, (char*)rtpPacket, dataSize + RTP_HEADER_SIZE, 0,
60                 (struct sockaddr*)&addr, sizeof(addr));
61
62     rtpPacket->rtpHeader.seq = ntohs(rtpPacket->rtpHeader.seq);
63     rtpPacket->rtpHeader.timestamp = ntohl(rtpPacket->rtpHeader.timestamp);
64     rtpPacket->rtpHeader.ssrc = ntohl(rtpPacket->rtpHeader.ssrc);
65
66     return ret;
67 }
68
69 }
70

```

rtp_server.cpp

```

1 #include<iostream>
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include <stdint.h>
5 #include <string.h>
6 #include <sys/types.h>
7 #include <sys/socket.h>
8 #include <sys/socket.h>
9 #include <netinet/in.h>
10 #include <arpa/inet.h>
11 #include <time.h>
12 #include <sys/types.h>
13 #include <sys/stat.h>
14 #include <fcntl.h>
15 #include <assert.h>
16 #include<unistd.h>
17 #include<thread>
18 #include "rtp.h"
19 #define H264_FILE_NAME "test.h264"
20 #define AAC_FILE_NAME "test.aac"
21 #define SERVER_PORT 8554
22 #define SERVER_RTP_PORT 55532
23 #define SERVER_RTCP_PORT 55533
24 #define BUF_MAX_SIZE (1024*1024)
25 #define SERVER_IP "127.0.0.1"
26 using namespace std;
27 //建立aacHeader
28 struct AdtsHeader
29 {
30     unsigned int syncword; //12 bit 同步字 '1111 1111 1111', 说明一个ADTS帧的开始
31     unsigned int id; //1 bit MPEG 标示符, 0 for MPEG-4, 1 for MPEG-2
32     unsigned int layer; //2 bit 总是'00'
33     unsigned int protectionAbsent; //1 bit 1表示没有crc, 0表示有crc
34     unsigned int profile; //1 bit 表示使用哪个级别的AAC
35     unsigned int samplingFreqIndex; //4 bit 表示使用的采样频率
36     unsigned int privateBit; //1 bit
37     unsigned int channelCfg; //3 bit 表示声道数
38     unsigned int originalCopy; //1 bit
39     unsigned int home; //1 bit
40
41     /*下面的为改变的参数即每一帧都不同*/
42     unsigned int copyrightIdentificationBit; //1 bit
43     unsigned int copyrightIdentificationStart; //1 bit
44     unsigned int aacFrameLength; //13 bit 一个ADTS帧的长度包括ADTS头和AAC原始流
45     unsigned int adtsBufferFullness; //11 bit 0x7FF 说明是速率可变的码流
46
47     /* number_of_raw_data_blocks_in_frame
48      * 表示ADTS帧中有number_of_raw_data_blocks_in_frame + 1个AAC原始帧
49      * 所以说number_of_raw_data_blocks_in_frame == 0
50      * 表示说ADTS帧中有一个AAC数据块并不是说没有。(一个AAC原始帧包含一段时间内1024个采样及相关数据)
51      */
52     unsigned int numberOfRawDataBlockInFrame; //2 bit
53 };
54 //解析aacHeader
55 static int parseAdtsHeader(uint8_t* in, struct AdtsHeader* res)
56 {
57     static int frame_number = 0;
58     memset(res, 0, sizeof("res"));
59
60     if ((in[0] == 0xFF) && ((in[1] & 0xF0) == 0xF0))
61     {
62         r e s->id = ((unsigned int)in[1] & 0x08) >> 3;
63         r e s->layer = ((unsigned int)in[1] & 0x06) >> 1;
64         r e s->protectionAbsent = (unsigned int)in[1] & 0x01;
65         r e s->profile = ((unsigned int)in[2] & 0xc0) >> 6;
66         r e s->samplingFreqIndex = ((unsigned int)in[2] & 0x3c) >> 2;
67         r e s->privateBit = ((unsigned int)in[2] & 0x02) >> 1;
68         r e s->channelCfg = (((unsigned int)in[2] & 0x01) << 2 | (((unsigned int)in[3] & 0xc0) >> 6));
69         r e s->originalCopy = ((unsigned int)in[3] & 0x20) >> 5;
70         r e s->home = ((unsigned int)in[3] & 0x10) >> 4;
71         r e s->copyrightIdentificationBit = ((unsigned int)in[3] & 0x08) >> 3;
72         r e s->copyrightIdentificationStart = (unsigned int)in[3] & 0x04 >> 2;
73         r e s->aacFrameLength = (((((unsigned int)in[3]) & 0x03) << 11) |
74                               (((unsigned int)in[4] & 0xFF) << 3) |
75                               ((unsigned int)in[5] & 0xE0) >> 5);
76         r e s->adtsBufferFullness = (((unsigned int)in[5] & 0x1f) << 6 |
77                                   ((unsigned int)in[6] & 0xfc) >> 2);
78         r e s->numberOfRawDataBlockInFrame = ((unsigned int)in[6] & 0x03);
79
80         return 0;
81     }
82 }

```



七伤之酒

码龄2年

暂无认证

7

124w+

13w+

3419



原创

周排名

总排名

访问

等级

79

3

6

5

6

积分

粉丝

获赞

评论

收藏









私信

关注

搜博主文章



热门文章

C语言斐波那契数列解析  1370

C语言汉塔塔问题解析  1038

用函数输出100~200之间的素数  407

链表c++  240

Rtsp服务器搭建(RTSP协议实现)  171

最新评论

Rtsp服务器搭建(RTSP协议实现)

CSDN-Ada助手: 恭喜您成功搭建了RTSP服务器, 这篇博客非常有价值, 对我们这些...

rtsp简单服务器

CSDN-Ada助手: 恭喜您写了第7篇博客, 标题为"rtsp简单服务器"。这篇文章很有深度...

Rtsp服务器搭建(RTSP协议实现)

魚小飛: 支持老表👍

Rtsp服务器搭建(RTSP协议实现)

rmring364: 好文, 期待下个作品

c++网络

CSDN-Ada助手: 恭喜您写了第5篇博客, 标题为"c++网络"。看到您的不断创新工作, 我...

您愿意向朋友推荐"博客详情页"吗?











强烈不推荐

不推荐

一般般

推荐

强烈推荐

最新文章












Rtsp服务器搭建(RTSP协议实现)

c++网络

链表c++

2023年 3篇

2022年 4篇

- 
- 
- 
- 
- 
- 
- 
- 
- 
- 
- 
- 

 CSDN 博客 下载 学习 社区 GitCode

```

184         return 0;
185     }
186     //找下一段h264数据
187     static char* findNextStartCode(char* buf, int len)
188     {
189         int i;
190         if (len < 3)
191             return NULL;
192         for (i = 0; i < len - 3; i++)
193         {
194             if (startCode3(buf) || startCode4(buf))
195                 return buf;
196             ++buf;
197         }
198         if (startCode3(buf))
199             return buf;
200         return NULL;
201     }
202     //获得h264码流大小
203     static int getFrameFromH264File(int fd, char* frame, int size)
204     {

```




会员中心 消息 历史 创作中心 发布

RTsp服务器搭建(荷载H264和AAC)	1
什么是RTSP协议?	2
RTSP协议详解	3
RTSP客户端请求	4
RTSP服务端的响应格式	5
RTSP请求的常用方法	6
RTSP协议	7
RTP头部	8
RTSP建立	9
RtpHeader	10
RtpPacket	11

```

205 int rSize, frameSize;
206 char* nextStartCode;
207 if (fd < 0)
208     return fd;
209 rSize = read(fd, frame, size);
210 if (!startCode3(frame) && !startCode4(frame))
211     return -1;
212 nextStartCode = findNextStartCode(frame + 3, rSize - 3);
213 if (!nextStartCode)
214     return -1;
215 else
216 {
217     frameSize = nextStartCode - frame;
218     lseek(fd, frameSize - rSize, SEEK_CUR);
219 }
220 return frameSize;
221 }
222 //用tp格式打包并发送h264视频流数据
223 static int rtpSendH264Frame(int clientSockfd,
224     struct RtpPacket* rtpPacket, char* frame, uint32_t frameSize)
225 {
226
227     uint8_t naluType; // nalu第一个字节
228     int sendByte = 0;
229     int ret;
230
231     naluType = frame[0];
232
233     printf("%s frameSize=%d \n", __FUNCTION__, frameSize);
234
235     if (frameSize <= RTP_MAX_PKT_SIZE) // nalu长度小于最大包场: 单一NALU单元模式
236     {
237
238         /*      0 1 2 3 4 5 6 7 8 9
239         +-+-+-+-+-+-+-+-+
240         |F|NRI| Type   | a single NAL unit ... |
241         +-+-+-+-+-+-+-+-+
242
243         memcpy(rtpPacket->payload, frame, frameSize);
244         ret = rtpSendPacketOverTcp(clientSockfd, rtpPacket, frameSize, 0x00);
245         if (ret < 0)
246             return -1;
247
248         rtpPacket->rtpHeader.seq++;
249         sendByte ++ ret;
250         if ((naluType & 0x1F) == 7 || (naluType & 0x1F) == 8) // 如果是SPS、PPS就不需要加时间戳
251         {
252
253         }
254
255     }
256     else // nalu长度小于最大包: 分片模式
257     {
258
259         /*      0      1      2
260         0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3
261         +-+-+-+-+-+-+-+-+
262         |FU indicator | FU header | FU payload ... |
263         +-+-+-+-+-+-+-+-+
264
265
266
267         /*      FU Indicator
268         /*      0 1 2 3 4 5 6 7
269         /*      +-+-+-+-+-+-+-+-+
270         /*      |E|NRI| Type   |
271         /*      +-----+
272
273
274
275         /*      FU Header
276         /*      0 1 2 3 4 5 6 7
277         /*      +-+-+-+-+-+-+-+-+
278         /*      |S|E|R| Type   |
279         /*      +-----+
280
281
282         int pktNum = frameSize / RTP_MAX_PKT_SIZE; // 有几个完整的包
283         int remainPktSize = frameSize % RTP_MAX_PKT_SIZE; // 剩余不完整包的大小
284         int i, pos = 1;
285
286         // 发送完整的包
287         for (i = 0; i < pktNum; i++)
288         {
289             rtpPacket->payload[0] = (naluType & 0x60) | 28;
290             rtpPacket->payload[1] = naluType & 0x1F;
291
292             if (i == 0) //第一包数据
293                 rtpPacket->payload[1] |= 0x80; // start
294             else if (remainPktSize == 0 && i == pktNum - 1) //最后一包数据
295                 rtpPacket->payload[1] |= 0x40; // end
296
297             memcpy(rtpPacket->payload + 2, frame + pos, RTP_MAX_PKT_SIZE);
298             ret = rtpSendPacketOverTcp(clientSockfd, rtpPacket, RTP_MAX_PKT_SIZE + 2, 0x00);
299             if (ret < 0)
300                 return -1;
301
302             rtpPacket->rtpHeader.seq++;
303             sendByte ++ ret;
304             pos += RTP_MAX_PKT_SIZE;
305         }
306
307         // 发送剩余的数据
308         if (remainPktSize > 0)
309         {
310             rtpPacket->payload[0] = (naluType & 0x60) | 28;
311             rtpPacket->payload[1] = naluType & 0x1F;
312             rtpPacket->payload[1] |= 0x40; //end
313
314             memcpy(rtpPacket->payload + 2, frame + pos, remainPktSize + 2);
315             ret = rtpSendPacketOverTcp(clientSockfd, rtpPacket, remainPktSize + 2, 0x00);
316             if (ret < 0)
317                 return -1;
318
319             rtpPacket->rtpHeader.seq++;
320             sendByte ++ ret;
321         }
322     }
323
324     return sendByte;
325 }
326
327 }

```

分类专栏		
	C++	1篇
	stocket	1篇
	C	3篇

```

328 //获取报文第一行数据
329 static char* GetlineFromBuf(char* rbuf, char* line)
330 {
331     while (*rbuf != '\n')
332     {
333         *line = *rbuf;
334         l i n e++;
335         r b u f++;
336     }
337     *line = '\n';
338     ++line;
339     *line = '\0';
340     ++rbuf;
341     return rbuf;
342 }
343 static int handleCmd_OPTIONS(char* result, int cseq)
344 {
345     sprintf(result, "RTSP/1.0 200 OK\r\n"
346         "CSeq: %d\r\n"
347         "Public: OPTIONS, DESCRIBE, SETUP, PLAY\r\n"
348         "\r\n",
349         c s e q);
350     return 0;
351 }
352 static int handleCmd_DESCRIBE(char* result, int cseq, char* url)
353 {
354     char sdp[500];
355     char localIp[100];
356     sscanf(url, "rtsp://%[*]:", localIp);
357     sprintf(sdp, "v=0\r\n"
358         "o=-9%ld 1 IN IP4 %s\r\n"
359         "t=0 0\r\n"
360         "a=control:* \r\n"
361         "m=video 0 RTP/AVP 96\r\n"
362         "a=rtmpmap:96 H264/90000\r\n"
363         "a=control:track0\r\n"
364         "m=audio 1 RTP/AVP/TCP 97\r\n"
365         "a=rtmpmap:97 mpeg4-generic/44100/2\r\n"
366         "a=fmt:97 profile-level-id=1;mode=AAC-hbr;sizeLength=13;indexLength=3;indexDeltaLength=3;config=1210;\r\n"
367         "a=control:track1\r\n",
368         time(NULL), localIp);
369     sprintf(result, "RTSP/1.0 200 OK \r\n"
370         "CSeq: %d\r\n"
371         "Content-Base: %s\r\n"
372         "Content-type: application/sdp\r\n"
373         "Content-length: %d\r\n"
374         "\r\n"
375         "%s", cseq, url, (int)strlen(sdp), sdp);
376     return 0;
377 }
378 static int handleCmd_SETUP(char* result, int cseq)
379 {
380     if (cseq == 3) {
381         sprintf(result, "RTSP/1.0 200 OK\r\n"
382             "CSeq: %d\r\n"
383             "Transport: RTP/AVP/TCP;unicast;interleaved=0-1\r\n"
384             "Session: 66334873\r\n"
385             "\r\n",
386             c s e q);
387     }
388     else if (cseq == 4) {
389         sprintf(result, "RTSP/1.0 200 OK\r\n"
390             "CSeq: %d\r\n"
391             "Transport: RTP/AVP/TCP;unicast;interleaved=2-3\r\n"
392             "Session: 66334873\r\n"
393             "\r\n",
394             c s e q);
395     }
396     return 0;
397 }
398 static int handleCmd_PLAY(char* result, int cseq)
399 {
400     sprintf(result, "RTSP/1.0 200 OK\r\n"
401         "CSeq: %d\r\n"
402         "Range: npt=0.000-\r\n"
403         "Session: 66334873; timeout=60\r\n"
404         "\r\n", cseq);
405     return 0;
406 }
407 //接收并回复消息做出相应的响应
408 static void doClient(int clientSockfd, const char* clientIP, int clientPort) {
409     char method[40];
410     char url[100];
411     char version[40];
412     int CSeq;
413     char* rBuf = (char*)malloc(BUF_MAX_SIZE);
414     char* sBuf = (char*)malloc(BUF_MAX_SIZE);
415     while (true) {
416         int recvlen;
417         r e c v L e n = recv(clientSockfd, rBuf, BUF_MAX_SIZE, 0);
418         if (recvlen <= 0) {
419             break;
420         }
421         r B u f[recvlen] = '\0';
422         printf("Accept request rBuf = %s \n", rBuf);
423         const char* sep = "\n";
424         char* line = strtok(rBuf, sep);
425         while (line) {
426             if (strstr(line, "OPTIONS") ||
427                 strstr(line, "DESCRIBE") ||
428                 strstr(line, "SETUP") ||
429                 strstr(line, "PLAY")) {
430                 if (sscanf(line, "%s %s %s\r\n", method, url, version) != 3) {
431                     // error
432                 }
433             }
434             else if (strstr(line, "CSeq")) {
435                 if (sscanf(line, "CSeq: %d\r\n", &CSeq) != 1) {
436                     // error
437                 }
438             }
439             else if (strncmp(line, "Transport:", strlen("Transport:")) {
440                 // Transport: RTP/AVP/UDP;unicast;client_port=13358-13359
441                 // Transport: RTP/AVP;unicast;client_port=13358-13359
442                 if (sscanf(line, "Transport: RTP/AVP/TCP;unicast;interleaved=0-1\r\n") != 0) {

```

```

452         printf("parse Transport error \n");
453     }
454 }
455 l i n e = strtok(NULL, sep);
456 }
457
458 if (!strcmp(method, "OPTIONS")) {
459     if (handleCmd_OPTIONS(sBuf, CSeq))
460     {
461         printf("failed to handle options\n");
462         break;
463     }
464 }
465 else if (!strcmp(method, "DESCRIBE")) {
466     if (handleCmd_DESCRIBE(sBuf, CSeq, url))
467     {
468         printf("failed to handle describe\n");
469         break;
470     }
471 }
472 else if (!strcmp(method, "SETUP")) {
473     if (handleCmd_SETUP(sBuf, CSeq))
474     {
475         printf("failed to handle setup\n");
476         break;
477     }
478 }
479 else if (!strcmp(method, "PLAY")) {
480     if (handleCmd_PLAY(sBuf, CSeq))
481     {
482         printf("failed to handle play\n");
483         break;
484     }
485 }
486 else {
487     printf("Undefined method = %s \n", method);
488     break;
489 }
490 printf("Response sBuf = %s \n", sBuf);
491
492 send(clientSockfd, sBuf, strlen(sBuf), 0);
493
494
495 //开始播放，发送RTP包
496 if (!strcmp(method, "PLAY")) {
497
498     s t d::thread t1([&]() {
499
500         int frameSize, startCode;
501         char* frame = new char [500000];
502         R t p P a c k e t* rtpPacket = new RtpPacket[500000];
503         int fp = open(H264_FILE_NAME, O_RDONLY);
504         if (!fp) {
505             printf("Read %s fail\n", H264_FILE_NAME);
506             return;
507         }
508         rtpHeaderInit(rtpPacket, 0, 0, 0, RTP_VESION, RTP_PAYLOAD_TYPE_H264, 0,
509             0, 0x88923423);
510
511         printf("start play\n");
512
513         while (true) {
514             f r a m e S i z e = getFrameFromH264File(fp, frame, 500000);
515             if (frameSize < 0)
516             {
517                 printf("Read %s end , frameSize=%d \n", H264_FILE_NAME, frameSize);
518                 break;
519             }
520
521             if (startCode3(frame))
522                 s t a r t C o d e = 3;
523             else
524                 s t a r t C o d e = 4;
525
526             f r a m e S i z e -= startCode;
527             rtpSendH264Frame(clientSockfd, rtpPacket, frame + startCode, frameSize);
528
529             r t p P a c k e t->rtpHeader.timestamp += 90000 / 25;
530             usleep(40000); // 1000/25 = 1000
531         }
532         free(frame);
533         free(rtpPacket);
534     });
535
536     s t d::thread t2([&]() {
537         struct AdtsHeader adtsHeader;
538         struct RtpPacket* rtpPacket;
539         uint8_t* frame;
540         int ret;
541
542         F I L E* fp = fopen(AAC_FILE_NAME, "rb");
543         if (!fp) {
544             printf("Read %s fail\n", AAC_FILE_NAME);
545             return;
546         }
547
548         f r a m e = (uint8_t*)malloc(5000);
549         r t p P a c k e t = (struct RtpPacket*)malloc(5000);
550
551         rtpHeaderInit(rtpPacket, 0, 0, 0, RTP_VESION, RTP_PAYLOAD_TYPE_AAC, 1, 0, 0, 0x32411);
552
553         while (true)
554         {
555             //读取ADTS头部
556             r e t = fread(frame, 1, 7, fp);
557             if (ret <= 0)
558             {
559                 printf("fread err\n");
560                 break;
561             }
562             printf("fread ret=%d \n", ret);
563             //解析头部
564             if (parseAdtsHeader(frame, &adtsHeader) < 0)
565             {
566                 printf("parseAdtsHeader err\n");
567                 break;
568             }
569             //读取一帧
570             r e t = fread(frame, 1, adtsHeader.aacFrameLength - 7, fp);
571             if (ret <= 0)
572             {
573                 printf("fread err\n");

```

```

574         break;
575     }
576     //Rtp打包发送
577     rtpSendAACFrame(clientSockfd,
578         r t p P a c k e t, frame, adtsHeader.aacFrameLength - 7);
579     usleep(23223); //1000/43.06 * 1000
580 }
581
582     free(frame);
583     free(rtpPacket);
584     });
585
586     t 1.join();
587     t 2.join();
588
589     break;
590 }
591
592     memset(method, 0, sizeof(method) / sizeof(char));
593     memset(url, 0, sizeof(url) / sizeof(char));
594     C S e q = 0;
595
596
597 }
598
599     close(clientSockfd);
600     free(rBuf);
601     free(sBuf);
602 }
603 }
604 int main(int argc, char* argv[])
605 {
606     //建立套接字
607     int ServerSocket;
608     int ServerRtcpSocket, ServerRtpSocket;
609     //创建TCP套接字
610     ServerSocket = CreateTcpSocket();
611     if (ServerSocket < 0)
612     {
613         c o u t << "TCP create fail !!!" << endl;
614         exit(0);
615     }
616     //绑定端口和地址
617     if (BindSocketAddr(ServerSocket, "0.0.0.0", SERVER_PORT) < 0)
618     {
619         c o u t << "bind fail !!!" << endl;
620         exit(0);
621     }
622     //监听端口
623     if (listen(ServerSocket, 5) < 0)
624     {
625         c o u t << "listen !!!" << endl;
626         exit(0);
627     }
628     c o u t << "rtsp://" << SERVER_IP << ":" << SERVER_PORT << endl;
629     while (true)
630     {
631         int ClientSocket;
632         char ClientIp[40];
633         int ClientPort;
634         //接收客户端消息
635         ClientSocket = AcceptClient(ServerSocket, ClientIp, &ClientPort);
636         if (ClientSocket < 0)
637         {
638             printf("failed to accept client\n");
639             return -1;
640         }
641         //打印客户端信息
642         c o u t << "accept client;client ip:" << ClientIp << ",client port:" << ClientPort << endl;
643         //接收消息并做出响应
644         doClient(ClientSocket, ClientIp, ClientPort);
645     }
646     close(ServerSocket);
647     return 0;
648 }
649

```

具体实现

首先获取一个音视频文件(以test.mp4文件为例):

用ffmpeg将其拆解为h264和aac形式文件

```

1  ffmpeg -i test.mp4 -acodec copy -vn test.aac
2  ffmpeg -i test.mp4 -vcodec copy -an test.h264

```

将其存进程序的项目文件夹后运行程序

结果如下:

```

qishangzhijiu@qishangzhijiu-virtual-machine:~/projects/my_rtsp_server$ ./my_rtsp_server.out
rtsp://127.0.0.1:8554

```

CSDN @七伤之酒

因为是tcp形式,所以需要指定为tcp形式播放

播放命令为:

```

1  ffmpeg -i -rtsp_transport tcp rtsp://127.0.0.1:8554

```

结果如下:

```

qishangzhijiu@qishangzhijiu-virtual-machine:~/projects/my_rtsp_server$ ./my_rtsp_server.out
rtsp://127.0.0.1:8554
accept client:client ip:127.0.0.1;client port:48738
accept request rBuf = OPTIONS rtsp://127.0.0.1:8554 RTSP/1.0
CSeq: 1
User-Agent: Lavf58.76.100

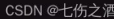
Response sBuf = RTSP/1.0 200 OK
CSeq: 1
Public: OPTIONS, DESCRIBE, SETUP, PLAY

accept request rBuf = DESCRIBE rtsp://127.0.0.1:8554 RTSP/1.0
Accept: application/sdp
CSeq: 2
User-Agent: Lavf58.76.100

Response sBuf = RTSP/1.0 200 OK
CSeq: 2
Content-Type: application/sdp
Content-length: 385

sdp
a=91885967394 1 IN IP4 127.0.0.1
t=0 0
a=control:1
m=video 0 RTP/AVP 96
a=rtcp:96 96 H264/90000
a=control:1:track0
a=media:0 0 RTP/AVP 97

```

 0
 
 0
 
 1
 

10-23

写评论

09-01

3714

weixin_66034721的博客 173

wtnu200的专栏 5855

wtu200的专栏 5855

09-11

08-21

01-22

554

RT

qq 25330791的博客 © 9226

rtsp流媒体服务器搭建

最新发布

07-21

 非常没帮助
  没帮助
  一般
  有帮助
  非常有帮助

公安备案号11010502030143 京ICP备19004658号 京网文〔2020〕1039-165号 经营性网站备案信息 北京互联网违法和不良信息举报中心 家长监护 网络110报警服务
中国互联网举报中心 Chrome商店下载 账号管理规范 版权与免责声明 版权申诉 出版许可证 营业执照 ©1999-2023北京创新乐知网络技术有限公司