

JT同学

经验4年

暂无认证

105

2万+

91万+

46万+

原创

周排名

总排名

访问

等级

5440

1861

1016

346

3945

积分

粉丝

获赞

评论

收藏



私信

关注

搜索

搜索

热门文章

从开始写一个RTSP服务器（一）RTSP协议讲解

57162

C++ deque的用法与示例

38004

Linux I2C驱动框架（超详细）

35491

我的开源项目-RtpServer

25554

从开始写一个RTSP服务器（二）RTSP协议的实现

21860

最新评论

我的开源项目-RtpServer

kaifa3k: 我测试 h264_rtp_server test.h264 也会这样的问题？请问解决了没？ m...

使用mp4v2封装H.264或mp4最简单示例

热心兔斯基: 应该是播放器的原因。换个播放器试试

使用mp4v2封装H.264或mp4最简单示例

al604233436: h264的标准中，好像默认的设置是90KHz。所以我理解timeScale...

使用mp4v2封装H.264或mp4最简单示例

al604233436: 有遇到，不知道原因呢。

深入学习Linux摄像头（二）v4l2驱动框架

qq_43745897: 请问往缓存区里面采集图像数据的帧率是多少呢？请问如果缓存区里...

您愿意向朋友推荐“博客详情页”吗？

强烈推荐

推荐

一般般

不推荐

强烈不推荐

最新文章

深入浅出MySQL事务（二）MVCC的实现原理

2020年 4篇

深入浅出MySQL事务（一）事务隔离

2019年 95篇

深入浅出MySQL索引（二）InnoDB存储引擎的索引

2018年 7篇

从零开始写一个RTSP服务器（六）一个传输AAC的RTSP服务器

JT同学

于 2019-08-12 16:43:27 发布

4594 收藏 14

分类专栏:

从零开始写一个RTSP服务器

文章标签:

rtsp流媒体aac

从零开始写一个RTS...

专栏收录该内容

187 订阅

10 篇文章

订阅专栏

从零开始写一个RTSP服务器系列

★我的开源项目-RtpServer

从零开始写一个RTSP服务器（一）RTSP协议讲解

从零开始写一个RTSP服务器（二）RTSP协议的实现

从零开始写一个RTSP服务器（三）RTP传输H.264

从零开始写一个RTSP服务器（四）一个传输H.264的RTSP服务器

从零开始写一个RTSP服务器（五）RTP传输AAC

从零开始写一个RTSP服务器（六）一个传输AAC的RTSP服务器

从零开始写一个RTSP服务器（七）多播传输RTP包

从零开始写一个RTSP服务器（八）一个多播的RTSP服务器

从零开始写一个RTSP服务器（九）一个RTP OVER RTP/TCP的RTSP服务器

从零开始写一个RTSP服务器（六）一个传输AAC的RTSP服务器

文章目录

从零开始写一个RTSP服务器（六）一个传输AAC的RTSP服务器

- 一、建立套接字
- 二、接收客户端连接
- 三、解析命令
- 四、处理请求
- 4.1 OPTIONS
- 4.2 DESCRIBE
- 4.3 SETUP
- 4.4 PLAY
- 五、AAC RTP打包发送
- 六、源码
- aac_rtp_server.c
- rtp.h
- rtp.c
- 七、测试

本文主要是结合

本文主要是结合

从零开始写一个RTSP服务器（一）不一样的RTSP协议讲解

从零开始写一个RTSP服务器（二）RTSP协议的实现

从零开始写一个RTSP服务器（五）RTP传输AAC

这几篇文章的内容总结，然后写出完成一个传输AAC的RTSP服务器，所有的知识点都在这几篇文章中，在着此篇文章前建议先认真看一看前面那几篇文章

本文不会讲解新知识点。如果你仔细看看前几篇文章，相信你不需要看本文就能够完成一个传输AAC的RTSP服务器

下面主要是介绍我提供的示例代码的运行流程和其中的一些细节

一、建立套接字

一开始进入main函数后，就监听服务器tcp套接字，绑定端口号，然后开始监听

然后再分别建立用于RTP和RTCP的udp套接字，绑定好端口

然后进入循环中开始服务

```
1 /*
2  * 作者: _JT_
3  * 博客: https://blog.csdn.net/weixin_42462282
4  */
5
6 main()
7 {
8     /* 创建服务器tcp套接字, 绑定端口, 监听 */
9     serverSocketfd = createTcpSocket();
10    bindSocketAddr(serverSocketfd, "0.0.0.0", SERVER_PORT);
11    listen(serverSocketfd, 10);
12
13    /* 建立用于RTP和RTCP的udp套接字, 绑定好端口 */
14    serverRtpSocketfd = createUdpSocket();
15    serverRtcpSocketfd = createUdpSocket();
16    bindSocketAddr(serverRtpSocketfd, "0.0.0.0", SERVER_RTP_PORT);
17    bindSocketAddr(serverRtcpSocketfd, "0.0.0.0", SERVER_RTCP_PORT);
18
19    while(1)
20    {
21        ...
22    }
23 }
```

二、接收客户端连接

在while循环中接收客户端，然后调用doClient服务

```
1 main()
2 {
3     ...
4     while(1)
5     {
6         clientSocketfd = acceptClient(serverSocketfd, clientIp, &clientPort);
7         doClient(clientSocketfd, clientIp, clientPort, serverRtpSocketfd, serverRtcpSocketfd);
8     }
9 }
```

上面其实就是一个TCP服务器的基本步骤，没有什么特别的

下面来看一看doClient函数

三、解析命令

doClient就是一个while循环（这是一个同时只能服务一个客户的服务器），不断地接收命令解析命令，然后调用相应地操作

```
1 /*
2  * 作者: _JT_
3  * 博客: https://blog.csdn.net/weixin_42462282
4  */
5
6 doClient()
7 {
8     while(1)
9     {
10        recv(clientSocketfd, rBuf, BUF_MAX_SIZE, 0);
11        ...
12        sscanf(line, "%s %s %s\n", method, url, version);
13        ...
14        sscanf(line, "CSeq: %d\n", &cseq);
15        ...
16    }
17 }
```

四、处理请求

在解析完客户端命令后，会调用相应的请求，处理完之后接收打印到sBuf中，然后发送给客户端

```
1 /*
2  * 作者: _JT_
3  * 博客: https://blog.csdn.net/weixin_42462282
4  */
5
6 doClient()
7 {
8     while(1)
9     {
10        ...
11        /* 处理请求 */
```

目录

从零开始写一个RTSP服务器（六）一个...

文章目录

一、建立套接字

二、接收客户端连接

三、解析命令

四、处理请求

4.1 OPTIONS

4.2 DESCRIBE

4.3 SETUP

4.4 PLAY

五、AAC RTP打包发送

分类专栏

C/C++	1篇
STL源码剖析	18篇
MySQL	4篇
Linux内核	14篇
nginx	
从零开始写一个RTSP服...	10篇
live555源码分析与应用	9篇
Linux驱动	19篇

```
12         if(!strcmp(method, "OPTIONS"))
13             handleCmd_OPTIONS(sBuf, cseq);
14         else if(!strcmp(method, "DESCRIBE"))
15             handleCmd_DESCRIBE(sBuf, cseq, url);
16         else if(!strcmp(method, "SETUP"))
17             handleCmd_SETUP(sBuf, cseq, clientRtpPort);
18         else if(!strcmp(method, "PLAY"))
19             handleCmd_PLAY(sBuf, cseq);
20
21         /* 返回结果 */
22         send(clientSockfd, sBuf, strlen(sBuf), 0);
23     }
24 }
```

下面来看看各个请求的行动

4.1 OPTIONS

返回可用方法

```
1  /*
2  * 作者: _JT_
3  * 博客: https://blog.csdn.net/weixin_42462282
4  */
5
6  static int handleCmd_OPTIONS(char* result, int cseq)
7  {
8      sprintf(result, "RTSP/1.0 200 OK\r\n"
9              "CSeq: %d\r\n"
10             "Public: OPTIONS, DESCRIBE, SETUP, PLAY\r\n"
11             "\r\n",
12             cseq);
13
14     return 0;
15 }
```

4.2 DESCRIBE

返回sdp文件信息，注意这个示例的sdp文件和从零开始写一个RTSP服务器（四）一个传输H.264的RTSP服务器中的sdp文件是不一样的，这是很重要的

```
1  /*
2  * 作者: _JT_
3  * 博客: https://blog.csdn.net/weixin_42462282
4  */
5
6  static int handleCmd_DESCRIBE(char* result, int cseq, char* url)
7  {
8      char sdp[500];
9      char localIp[100];
10
11     sscanf(url, "rtsp://%[*]:", localIp);
12
13     sprintf(sdp, "v=0\r\n"
14             "o=- 951d 1 IN IP4 %s\r\n"
15             "s=- 0\r\n"
```

4.3 SETUP

SETUP过程发送服务端RTP端口和RTCP端口

```
1  /*
2  * 作者: _JT_
3  * 博客: https://blog.csdn.net/weixin_42462282
4  */
5
6  static int handleCmd_SETUP(char* result, int cseq, int clientRtpPort)
7  {
8      sprintf(result, "RTSP/1.0 200 OK\r\n"
9              "CSeq: %d\r\n"
10             "Transport: RTP/AVP;unicast;client_port=%d-%d;server_port=%d-%d\r\n"
11             "Session: 66334873\r\n"
12             "\r\n",
13             cseq,
14             clientRtpPort,
15             clientRtpPort+1,
16             SERVER_RTP_PORT,
17             SERVER_RTCP_PORT);
18
19     return 0;
20 }
```

4.4 PLAY

PLAY操作回复后，会开始发送RTP包

```
1  /*
2  * 作者: _JT_
3  * 博客: https://blog.csdn.net/weixin_42462282
4  */
5
6  static int handleCmd_PLAY(char* result, int cseq)
7  {
8      sprintf(result, "RTSP/1.0 200 OK\r\n"
9              "CSeq: %d\r\n"
10             "Range: npt=0.000-1\r\n"
11             "Session: 66334873; timeout=60\r\n\r\n",
12             cseq);
13
14     return 0;
15 }
```

五、AAC RTP打包发送

先读取ADTS头，得到一帧的大小，然后再读取AAC Data，再透过RTP打包传输

```
1  /*
2  * 作者: _JT_
3  * 博客: https://blog.csdn.net/weixin_42462282
4  */
5
6  doClient()
7  {
8      while(1)
9      {
10         ...
11
12         send(clientSockfd, sBuf, strlen(sBuf), 0);
13
14         if(!strcmp(method, "PLAY"))
15         {
16             while(1)
17             {
18                 /* 读取ADTS头部 */
19                 read(fd, frame, 7);
20
21                 /* 解析头部 */
22                 parseAdtsHeader(frame, &adtsHeader);
23
24                 /* 读取一帧 */
25                 read(fd, frame, adtsHeader.aacFrameLength-7);
26
27                 /* RTP打包发送 */
28                 rtpSendAACFrame(localRtpSockfd, clientIP, clientRtpPort,
29                               rtpPacket, frame, adtsHeader.aacFrameLength-7);
30
31             }
32         }
33     }
34 }
35 }
```

看一看AAC的RTP打包发送过程

先填充RTP载荷前4个字节，然后发送RTP包

发送后序列号增加，时间戳增加

```
1  /*
2  * 作者: _JT_
3  * 博客: https://blog.csdn.net/weixin_42462282
4  */
5
6  static int rtpSendAACFrame(int socket, const char* ip, int16_t port,
7                             struct RtpPacket* rtpPacket, uint8_t* frame, uint32_t frameSize)
8  {
9      /* 填充前4个字节 */
10     rtpPacket->payload[0] = 0x00;
11     rtpPacket->payload[1] = 0x10;
12     rtpPacket->payload[2] = (frameSize & 0x1FE0) >> 5; //高8位
13     rtpPacket->payload[3] = (frameSize & 0x1F) << 3; //低5位
```

六、源码

总共由三个文件 `aac_rtsp_server.c`、`rtp.h`、`rtp.c`

`aac_rtsp_server.c`

```
1  /*
2  * 作者: _3T_
3  * 博客: https://blog.csdn.net/weixin_42462202
4  */
5
6  #include <stdio.h>
7  #include <stdlib.h>
8  #include <stdint.h>
9  #include <string.h>
10 #include <sys/types.h>
11 #include <sys/socket.h>
12 #include <sys/socket.h>
13 #include <netinet/in.h>
14 #include <arpa/inet.h>
15 #include <time.h>
16 #include <sys/types.h>
17 #include <sys/stat.h>
18 #include <fcntl.h>
19
20 #include "rtp.h"
21
22 #define SERVER_PORT    8554
23 #define SERVER_RTP_PORT 55532
24 #define SERVER_RTSP_PORT 55533
25 #define BUF_MAX_SIZE   (1024*1024)
26 #define AAC_FILE_NAME   "test.aac"
27
28 static int createTcpSocket()
29 {
30     int sockfd;
31     int on = 1;
32
33     sockfd = socket(AF_INET, SOCK_STREAM, 0);
34     if(sockfd < 0)
35         return -1;
36
37     setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, (const char*)0, sizeof(on));
38
39     return sockfd;
40 }
41
42 static int createUdpSocket()
43 {
44     int sockfd;
45     int on = 1;
46
47     sockfd = socket(AF_INET, SOCK_DGRAM, 0);
48     if(sockfd < 0)
49         return -1;
50
51     setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, (const char*)0, sizeof(on));
52
53     return sockfd;
54 }
55
56 static int bindSocketAddr(int sockfd, const char* ip, int port)
57 {
58     struct sockaddr_in addr;
59
60     addr.sin_family = AF_INET;
61     addr.sin_port = htons(port);
62     addr.sin_addr.s_addr = inet_addr(ip);
63
64     if(bind(sockfd, (struct sockaddr*)&addr, sizeof(struct sockaddr)) < 0)
65         return -1;
66
67     return 0;
68 }
69
70 struct AdtsHeader
71 {
72     unsigned int syncword; //12 bit 同步字 '1111 1111 1111', 说明一个ADTS帧的开始
73     unsigned int id; //1 bit MPEG 标识符, 0 for MPEG-4, 1 for MPEG-2
74     unsigned int layer; //2 bit 总是'00'
75     unsigned int protectionAbsent; //1 bit 1表示没有crc, 0表示有crc
76     unsigned int profile; //1 bit 表示使用哪个级别的AAC
77     unsigned int samplingFreqIndex; //4 bit 表示使用的采样频率
78     unsigned int privateBit; //1 bit
79     unsigned int channelCfg; //3 bit 表示声道数
80     unsigned int originalCopy; //1 bit
81     unsigned int home; //1 bit
82
83     /*下面的为改变的参数则每一帧都不同*/
84     unsigned int copyrightIdentificationBit; //1 bit
85     unsigned int copyrightIdentificationStart; //1 bit
86     unsigned int aacFrameLength; //13 bit 一个ADTS帧的长度包括ADTS头和AAC原始流
87     unsigned int adtsBufferFullness; //11 bit 0x7FF 说明是频率可变的码流
88
89     /* number_of_raw_data_blocks_in_frame
90     * 表示ADTS帧中number_of_raw_data_blocks_in_frame + 1个AAC原始帧
91     * 所以该number_of_raw_data_blocks_in_frame = 0
92     * 表示说ADTS帧中每一个AAC数据块并不是说没有。(一个AAC原始帧包含一段时间为1024个采样及相关数据)
93     */
94     unsigned int numberOfRawDataBlockInFrame; //2 bit
95 };
96
97 static int parseAdtsHeader(uint8_t* in, struct AdtsHeader* res)
98 {
99     static int frame_number = 0;
100     memset(res, 0, sizeof(*res));
101
102     if (((in[0] == 0xFF) && ((in[1] & 0xF0) == 0xF0))
103     {
104         res->id = (((unsigned int) in[1] & 0x08) >> 3);
105         res->layer = (((unsigned int) in[1] & 0x06) >> 1);
106         res->protectionAbsent = ((unsigned int) in[1] & 0x01);
107         res->profile = (((unsigned int) in[2] & 0xc0) >> 6);
108         res->samplingFreqIndex = (((unsigned int) in[2] & 0x3c) >> 2);
109         res->privateBit = (((unsigned int) in[2] & 0x02) >> 1);
110         res->channelCfg = (((unsigned int) in[2] & 0x01) << 2) | (((unsigned int) in[3] & 0xc0) >> 6));
111         res->originalCopy = (((unsigned int) in[3] & 0x20) >> 5);
112         res->home = (((unsigned int) in[3] & 0x10) >> 4);
113         res->copyrightIdentificationBit = (((unsigned int) in[3] & 0x08) >> 3);
114         res->copyrightIdentificationStart = ((unsigned int) in[3] & 0x04) >> 2;
115         res->aacFrameLength = (((((unsigned int) in[3]) & 0x03) << 11) |
116             (((unsigned int) in[4] & 0xFF) << 3) |
117             ((unsigned int) in[5] & 0xF8) >> 5) );
118         res->adtsBufferFullness = (((unsigned int) in[5] & 0xF) << 6) |
119             ((unsigned int) in[6] & 0xFc) >> 2);
120         res->numberOfRawDataBlockInFrame = ((unsigned int) in[6] & 0x03);
121
122         return 0;
123     }
124     else
125     {
126         printf("failed to parse adts header\n");
127         return -1;
128     }
129 }
130
131 static int rtpSendAACFrame(int socket, const char* ip, int16_t port,
132                             struct RtpPacket* rtpPacket, uint8_t* frame, uint32_t frameSize)
133 {
134     int ret;
135
136     rtpPacket->payload[0] = 0x00;
137     rtpPacket->payload[1] = 0x10;
138     rtpPacket->payload[2] = (frameSize & 0x1f00) >> 5; //高8位
139     rtpPacket->payload[3] = (frameSize & 0xf) << 3; //低8位
140
141     memcpy(rtpPacket->payload+4, frame, frameSize);
142
143     ret = rtpSendPacket(socket, ip, port, rtpPacket, frameSize+4);
144     if(ret < 0)
145     {
146         printf("failed to send rtp packet\n");
147         return -1;
148     }
149
150     rtpPacket->rtpHeader.seq++;
151
152     /*
153     * 如果采样频率是44100
154     * 一般AAC每个1024个采样为一帧
155     * 所以一秒就有 44100 / 1024 = 43帧
156     */
157 }
```

```

156 * 时间间隔是 44100 / 43 = 1025
157 * 一般的时间为 1 / 43 = 23ms
158 */
159 rtpPacket->rtpHeader.timestamp += 1025;
160
161 return 0;
162 }
163
164 static int acceptClient(int sockfd, char* ip, int* port)
165 {
166     int clientfd;
167     socklen_t len = 0;
168     struct sockaddr_in addr;
169     memset(&addr, 0, sizeof(addr));
170     len = sizeof(addr);
171
172     clientfd = accept(sockfd, (struct sockaddr*)&addr, &len);
173     if(clientfd < 0)
174         return -1;
175
176     strcpy(ip, inet_ntoa(addr.sin_addr));
177     *port = ntohs(addr.sin_port);
178
179     return clientfd;
180 }
181
182 static char* getlineFromBuf(char* buf, char* line)
183 {
184     while(*buf != '\n')
185     {
186         *line = *buf;
187         line++;
188         buf++;
189     }
190
191     *line = '\n';
192     ++line;
193     *line = '\0';
194
195     ++buf;
196     return buf;
197 }
198
199 static int handleCmd_OPTIONS(char* result, int cseq)
200 {
201     sprintf(result, "RTSP/1.0 200 OK\r\n"
202             "CSeq: %d\r\n"
203             "Public: OPTIONS, DESCRIBE, SETUP, PLAY\r\n"
204             "\r\n",
205             cseq);
206
207     return 0;
208 }
209
210 static int handleCmd_DESCRIBE(char* result, int cseq, char* url)
211 {
212     char sdp[500];
213     char localIp[100];
214
215     sscanf(url, "rtsp://[~:~]:", localIp);
216
217     sprintf(sdp, "v=0\r\n"
218             "o=- 9812 1 IN IP4 %s\r\n"
219             "t=0 0\r\n"
220             "a=control:* \r\n"
221             "m=audio 0 RTP/AVP 97 \r\n"
222             "a=rtpmap:97 mpeg4-generic/44100/2 \r\n"
223             "a=fmtp:97 Sizerlength=13 \r\n"
224             "a=control:track0 \r\n"
225             "time(NULL), localIp);
226
227     sprintf(result, "RTSP/1.0 200 OK\r\nCSeq: %d\r\n"
228             "Content-Base: %s\r\n"
229             "Content-type: application/sdp\r\n"
230             "Content-length: %d\r\n\r\n"
231             "%s",
232             cseq,
233             url,
234             strlen(sdp),
235             sdp);
236
237     return 0;
238 }
239
240 static int handleCmd_SETUP(char* result, int cseq, int clientRtpPort)
241 {
242     sprintf(result, "RTSP/1.0 200 OK\r\n"
243             "CSeq: %d\r\n"
244             "Transport: RTP/AVP;unicast;client_port=%d-%d;server_port=%d-%d\r\n"
245             "Session: 66334873\r\n"
246             "\r\n",
247             cseq,
248             clientRtpPort,
249             clientRtpPort+1,
250             SERVER_RTP_PORT,
251             SERVER_RTP_PORT,
252             SERVER_RTCP_PORT);
253
254     return 0;
255 }
256
257 static int handleCmd_PLAY(char* result, int cseq)
258 {
259     sprintf(result, "RTSP/1.0 200 OK\r\n"
260             "CSeq: %d\r\n"
261             "Range: npt=0.000- \r\n"
262             "Session: 66334873; timeout=60 \r\n\r\n",
263             cseq);
264
265     return 0;
266 }
267
268 static void doClient(int clientSockfd, const char* clientIP, int clientPort,
269                    int serverRtpSockfd, int serverRtcpSockfd)
270 {
271     char method[40];
272     char url[100];
273     char version[40];
274     int cseq;
275     int clientRtpPort, clientRtcpPort;
276     char* bufPtr;
277     char* rBuf = malloc(BUF_MAX_SIZE);
278     char* sBuf = malloc(BUF_MAX_SIZE);
279     char line[400];
280
281     while(1)
282     {
283         int recvlen;
284
285         recvlen = recv(clientSockfd, rBuf, BUF_MAX_SIZE, 0);
286         if(recvlen <= 0)
287             goto out;
288
289         rBuf[recvlen] = '\0';
290         printf("-----C>S-----\n");
291         printf("%s", rBuf);
292
293         /* 解析方法 */
294         bufPtr = getlineFromBuf(rBuf, line);
295         if(sscanf(line, "%s %s %s\r\n", method, url, version) != 3)
296         {
297             printf("parse err\n");
298             goto out;
299         }
300
301         /* 解析序列号 */
302         bufPtr = getlineFromBuf(bufPtr, line);
303         if(sscanf(line, "CSeq: %d\r\n", &cseq) != 1)
304         {
305             printf("parse err\n");
306             goto out;
307         }
308
309         /* 如果是SETUP, 那么就用解析client_port */
310         if(!strcmp(method, "SETUP"))
311         {
312             while(1)
313             {
314                 bufPtr = getlineFromBuf(bufPtr, line);
315                 if(!strcmp(line, "Transport:"))
316                 {
317                     sscanf(line, "Transport: RTP/AVP;unicast;client_port=%d-%d\r\n",
318                             &clientRtpPort, &clientRtcpPort);
319

```

```

320         break;
321     }
322 }
323 }
324
325 if(!strcmp(method, "OPTIONS"))
326 {
327     if(handleCmd_OPTIONS(sBuf, cseq))
328     {
329         printf("failed to handle options\n");
330         goto out;
331     }
332 }
333 else if(!strcmp(method, "DESCRIBE"))
334 {
335     if(handleCmd_DESCRIBE(sBuf, cseq, url))
336     {
337         printf("failed to handle describe\n");
338         goto out;
339     }
340 }
341 else if(!strcmp(method, "SETUP"))
342 {
343     if(handleCmd_SETUP(sBuf, cseq, clientRtpPort))
344     {
345         printf("failed to handle setup\n");
346         goto out;
347     }
348 }
349 else if(!strcmp(method, "PLAY"))
350 {
351     if(handleCmd_PLAY(sBuf, cseq))
352     {
353         printf("failed to handle play\n");
354         goto out;
355     }
356 }
357 else
358 {
359     goto out;
360 }
361
362 printf("-----S>C-----\n");
363 printf("Ss", sBuf);
364 send(clientSockfd, sBuf, strlen(sBuf), 0);
365
366 if(!strcmp(method, "PLAY"))
367 {
368     struct AdtsHeader adtsHeader;
369     struct RtpPacket* rtpPacket;
370     uint8_t* frame;
371     int ret;
372
373     int fd = open(AAC_FILE_NAME, O_RDONLY);
374     if(fd < 0)
375     {
376         printf("failed to open %s\n", AAC_FILE_NAME);
377         goto out;
378     }
379
380     frame = (uint8_t*)malloc(5000);
381     rtpPacket = malloc(5000);
382
383     rtpHeaderInit(rtpPacket, 0, 0, 0, RTP_VERSION, RTP_PAYLOAD_TYPE_AAC, 1, 0, 0, 0x32411);
384
385     while(1)
386     {
387         ret = read(fd, frame, 7);
388         if(ret <= 0)
389         {
390             break;
391         }
392
393         if(parseAdtsHeader(frame, &adtsHeader) < 0)
394         {
395             printf("parse err\n");
396             break;
397         }
398
399         ret = read(fd, frame, adtsHeader.aacFrameLength-7);
400         if(ret < 0)
401         {
402             printf("read err\n");
403             break;
404         }
405
406         rtpSendAACFrame(serverRtpSockfd, clientIP, clientRtpPort,
407             rtpPacket, frame, adtsHeader.aacFrameLength-7);
408
409         usleep(23000);
410     }
411
412     free(frame);
413     free(rtpPacket);
414 }
415 }
416 out:
417 close(clientSockfd);
418 free(rBuf);
419 free(sBuf);
420 }
421
422 int main(int argc, char* argv[])
423 {
424     int serverSockfd;
425     int serverRtpSockfd, serverRtcpSockfd;
426     int ret;
427
428     serverSockfd = createTcpSocket();
429     if(serverSockfd < 0)
430     {
431         printf("failed to create tcp socket\n");
432         return -1;
433     }
434
435     ret = bindSocketAddr(serverSockfd, "0.0.0.0", SERVER_PORT);
436     if(ret < 0)
437     {
438         printf("failed to bind addr\n");
439         return -1;
440     }
441
442     ret = listen(serverSockfd, 10);
443     if(ret < 0)
444     {
445         printf("failed to listen\n");
446         return -1;
447     }
448
449     serverRtpSockfd = createUdpSocket();
450     serverRtcpSockfd = createUdpSocket();
451     if(serverRtpSockfd < 0 || serverRtcpSockfd < 0)
452     {
453         printf("failed to create udp socket\n");
454         return -1;
455     }
456
457     if(bindSocketAddr(serverRtpSockfd, "0.0.0.0", SERVER_RTP_PORT) < 0 ||
458        bindSocketAddr(serverRtcpSockfd, "0.0.0.0", SERVER_RTCP_PORT) < 0)
459     {
460         printf("failed to bind addr\n");
461         return -1;
462     }
463
464     printf("rtsp://127.0.0.1:%d\n", SERVER_PORT);
465
466     while(1)
467     {
468         int clientSockfd;
469         char clientIp[40];
470         int clientPort;
471
472         clientSockfd = acceptClient(serverSockfd, clientIp, &clientPort);
473         if(clientSockfd < 0)
474         {
475             printf("failed to accept client\n");
476             return -1;
477         }
478
479         printf("accept client;client ip:%s,client port:%d\n", clientIp, clientPort);
480
481         doClient(clientSockfd, clientIp, clientPort, serverRtpSockfd, serverRtcpSockfd);
482     }
483 }

```

```
484     return 0;
485 }
```

rtp.h

```
1  /*
2   * 作者: _JT_
3   * 博客: https://blog.csdn.net/weixin_42462282
4   */
5
6  #ifndef _RTP_H_
7  #define _RTP_H_
8  #include <stdint.h>
9
10 #define RTP_VERSION      2
11
12 #define RTP_PAYLOAD_TYPE_H264  96
13 #define RTP_PAYLOAD_TYPE_AAC   97
14
15 #define RTP_HEADER_SIZE      12
```

rtp.c

```
1  /*
2   * 作者: _JT_
3   * 博客: https://blog.csdn.net/weixin_42462282
4   */
5
6  #include <sys/types.h>
7  #include <sys/socket.h>
8  #include <arpa/inet.h>
9  #include <netinet/in.h>
10 #include <arpa/inet.h>
11
12 #include "rtp.h"
13
14 void rtpHeaderInit(struct RtpPacket* rtpPacket, uint8_t csrcLen, uint8_t extension,
15                  uint8_t padding, uint8_t version, uint8_t payloadType, uint8_t marker,
16                  uint16_t seq, uint32_t timestamp, uint32_t ssrc)
17 {
18     rtpPacket->rtpHeader.csrcLen = csrclen;
19     rtpPacket->rtpHeader.extension = extension;
20     rtpPacket->rtpHeader.padding = padding;
21     rtpPacket->rtpHeader.version = version;
22     rtpPacket->rtpHeader.payloadType = payloadType;
23     rtpPacket->rtpHeader.marker = marker;
24     rtpPacket->rtpHeader.seq = seq;
25     rtpPacket->rtpHeader.timestamp = timestamp;
26     rtpPacket->rtpHeader.ssrc = ssrc;
27 }
28
29 int rtpSendPacket(int socket, const char* ip, int16_t port, struct RtpPacket* rtpPacket, uint32_t dataSize)
30 {
31     struct sockaddr_in addr;
32     int ret;
33
34     addr.sin_family = AF_INET;
35     addr.sin_port = htons(port);
36     addr.sin_addr.s_addr = inet_addr(ip);
37
38     rtpPacket->rtpHeader.seq = htons(rtpPacket->rtpHeader.seq);
39     rtpPacket->rtpHeader.timestamp = htonl(rtpPacket->rtpHeader.timestamp);
40     rtpPacket->rtpHeader.ssrc = htonl(rtpPacket->rtpHeader.ssrc);
41
42     ret = sendto(socket, (void*)rtpPacket, dataSize-RTP_HEADER_SIZE, 0,
43                (struct sockaddr*)&addr, sizeof(addr));
44
45     rtpPacket->rtpHeader.seq = ntohs(rtpPacket->rtpHeader.seq);
46     rtpPacket->rtpHeader.timestamp = ntohl(rtpPacket->rtpHeader.timestamp);
47     rtpPacket->rtpHeader.ssrc = ntohl(rtpPacket->rtpHeader.ssrc);
48
49     return ret;
50 }
```

七、测试

将aac_rtsp_server.c、rtp.h、rtp.c这三个文件保存下来

编译运行，默认会打开test.aac的音频文件，如果你没有音频源的话，可以从RtspServer的example目录下获取

```
1 # gcc aac_rtsp_server.c rtp.c
2 # ./a.out
```

运行之后会打印一个url

```
1 | rtsp://127.0.0.1:8554
```

再vlc打开即可听到音频

rtsp_rtp_h2648Mjpeg—java版本最简单全实现	01-22
<p> 0.所有相关的资料、代码、工具都放在百度云盘中 </p> <p> 1.java语言实现 2.rtsp_rtp_h2648Mjpeg协议最简单全实现。无控件 3.可以和...	
RtspServer.RTSP服务器，支持传输H.264和AAC格式的音视频	05-11
RtspServer 项目介绍 使用C++实现的一个RTSP服务器 功能介绍 支持H264、AAC的音视频编解码 支持传输H264格式的流媒体文件和AAC格式的音频文件 支...	
rtsp协议详解_详解RTSP封装和打包AAC实战分析(2)	weixin_39960147的博文 482
0.引为为了更好地理解本篇文章，可以先阅读前面几篇文章，文章列表如下: 详解RTSP打包AAC实战分析(1) 详解RTSP协议之H264封装和解包实战详解RTSP协...	
将rtsp流保存为aac	08-24
获取rtsp流件的音频并保存为可播放的aac文件	
rtsp解析实现音视频实时传输	08-31
rtsp协议解析实现音视频实时传输	
live555学习笔记6	wodemazi 86
六 建立RTSP会话 首先更正一个概念: ServerMediaSession原先说代表一个流。其实是不准确的。它代表的是server端的一个媒体的名字，而说ServerM...	
rtsp 服务器搭建	yinhshipin007的博文 4838
rtsp 服务器搭建 最新案例 今天我们搭建这个 rtsp 服务器的名称叫做: ZLMediaKit，它是一个基于 C++11 的高性能运营级流媒体服务框架。类似我之前给大家搭...	
RTSP服务器: RTP传输AAC流	weixin_43796767的博文 626
参考文章: AAC音频流解析 RTP打包传输H264码流 业务流程: 1) 读取ADTS头(7字节)，解析得到aac帧的信息(频率、声道、帧长度) 2) 读取a...	
如何解析RTSP中的AAC音频流	VivianLab 649
一般来说，一般音频数据的数据量是很小的，在RTSP传输中，一个RTP包就可以传输，不需要经过U-A等形式分包。因此，一个承载一批AAC的RTP，应...	
SETUP命令处理	黔桂西源 2886
live555学习(八) —SETUP命令处理 2012-03-16 09:50:39 分类: Live555学习笔记 排版 序号 订阅 SETUP命令概述 首先更正一个概念: S...	8...
java实现传输AAC的RTSP服务	胖牛的博客 220
java实现传输AAC的RTSP服务, AAC文件解析成rtp包	
RTSP服务器: 一个传输h264/aac的RTSP服务器	weixin_43796767的博文 406
综合RTSP协议实现与RTP传输H264码流RTSP传输aac码流, 即可实现一个传输H264码流的RTSP服务器。思路很简单, 基于RTSP与VLC的交互流程, 在...	
RTSP流H265(hvc) + AAC关键点详解! 史上最全RTSP+hvc 交互全记录!	一只海星的主页 1327
背景: 4012年来了, H265已经退出小众, 日趋流行! 在视频大数据时代, 高压缩优势也越来越明显! 所以! 搞一波RTSP+HVC+AAC取流支持! 网上资...	
从零开始写一个RTSP服务器	NBA_1的博文 101
https://blog.csdn.net/weixin_42462202/article/details/98986535	
rtsp协议传输视频流, 保持客户端和服务端连接的方法	奥手的煎山药师的博文 1000
系列文章目录 《rtsp协议理解》 《rtsp (udp、tcp) 协议详解》 文章目录系列文章目录前言为什么要监测rtsp的连接状态使用RTSP协议中的GET_PARA...	
RTSP协议的一些分析 (六) ——使用RTP传输AAC文件	yangguoyu80238的博文 1042
RTSP的33条命令, 我已经在前面的文章中讲过, 这里不做赘述。 一、 AAC的RTP打包 1.1 AAC格式 详见音频编码基础。 1.2 AAC的RTP打包方式 AAC...	
RTSP协议与G711 AAC	wzl_whu的专栏 2338
文章目录文章rtsp over tcp + G711OPTIONSDESCRIBESETUPPLAYEARDOWNRTP over TCP的封装格式RTSPS RTP over UDP + AAC 文档 如果想...	
RTSP/SDP中的AAC配置	wzl_whu的专栏 4496
RTSP的音频使用AAC格式, SDP的内容差不多是这样的 v=0 o=- 16128587303007558 182 16128587303007558 182 IN IP4 WINDOWS-7SIDU9Q s=Unna...	
RTSP (H.264+AAC) 音视频编解码心得—linux C	flwz6572463的专栏 1370
调试RTSP遇到的问题 RTSP的开源代码很少, 移植方法也很多, 所以在这里不做过多的描述。我主要来为大家讲述下移植过程中遇到的一些容易忽视的问...	
RTSP协议的一些分析 (七) ——传输AAC的RTSP服务器	yangguoyu80238的博文 249
直接上代码。 #include <stdio.h> #include <stdlib.h> #include <string.h> #include <time.h> #include <fcntl.h> #include <unistd.h> #i...	
AAC ADTS格式分析	weixin_30378623的博文 169
转自: https://blog.csdn.net/jay100500/article/details/52955232 https://blog.csdn.net/andhyuabing/article/details/40983423 https://blog.csdn.net/liukun...	

“相关推荐”对您有帮助么?

😄 非常有帮助

😄 有帮助

😐 一般

😞 没帮助

😞 非常没帮助

©2022 CSDN 皮肤主题: 数字20 设计师: CSDN官方博普 返回顶部

关于我们 招贤纳士 商务合作 寻求报道 400-660-0108 kefu@csdn.net 在线客服 工作时间 8:30-22:00

公安备案号 11010502030143 京ICP备19004658号 京公网安备 2020 1039-16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 账号被盗 隐私政策 版权申诉 营业执照 1995-2022 北京卓创未来网络科技有限公司

JT同学 关注

👍 14

👍 2

🔖 1

🔖 2

专栏目录