

JT同学

粉丝4年

暂无认证

105

2万+

91万+

46万+

原创

周排名

总排名

访问量

等级

5440

1861

1016

346

3945

积分

粉丝

获赞

评论

收藏











私信

关注

搜索主文章

Q

热门文章

从零基础写一个RTSP服务器（一）RTSP协议讲解

57178

C++ deque的用法与示例

38014

Linux I2C驱动框架（超详细）

35491

我的开源项目-RtpServer

25563

从零基础写一个RTSP服务器（二）RTSP协议的实现

21864

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成mp4最简单示例
热心鬼斯比: 应该是播放器的原因。换个播放器试试
使用mp4v2封装H.264成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meScale...
使用mp4v2封装H.264成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？

😄

😐

😐

😐

😐

强烈不推荐

不推荐

一般般

推荐

强烈推荐

最新文章

深入浅出MySQL事务（二）MVCC的实现原理

深入浅出MySQL事务（一）事务隔离

深入浅出MySQL索引（二）InnoDB存储引擎的索引

2020年 4篇

2019年 95篇

2018年 7篇

从零开始写一个RTSP服务器（七）多播传输RTP包

JT同学

于 2019-08-12 19:42:28 发布

5574 收藏 18

分类专栏:

从零开始写一个RTSP服务器

文章标签:

rtsp

rtsp

多播

从零开始写一个RTSP...

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（七）多播传输RTP包

文章目录

从零开始写一个RTSP服务器（七）多播传输RTP包

一、多播

1.1 多播简介

1.2 多播示例

二、多播的udp描述文件

三、多播传输RTP包实现

3.1 实现

3.2 源码

multicast.c

rtsp.h

rtsp.c

四、测试

一、多播

1.1 多播简介

单播地址标识某个IP接口，广播地址标识某个子网的所有IP接口，多播地址标识一组IP接口

单播和多播是两个极端，多播则在这两者之间折衷

多播数据报文只由加入多播组的应用的宿主机的接口接收

IPv4的D类地址是IPv4的多播地址。范围是（224.0.0.0-239.255.255.255）

下面这张图对多播地址进行细分

范 围	IPv6范围	IPv4	
		TTL范围	可管理范围
接口局部	1	0	
链路局部	2	1	224.0.0.0到224.0.0.255
网段局部	5	<32	239.255.0.0到239.255.255.255
组织机构局部	8		239.192.0.0到239.195.255.255
全球	14	≤255	224.0.1.0到238.255.255.255

图21-3 IPv4和IPv6多播地址范围

1.2 多播示例

下面讲解一下多播的客户端与服务端编程

服务端

多播服务端并不需要什么特殊的操作，只需要创建udp套接字，然后向多播地址指定端口发送数据就行

客户端

多播客户端需要做的工作是，创建udp套接字，绑定多播端口，加入多播组

下面是示例代码

服务端

```
1 /*
2  *broadcast_server.c - 多播服务器程序
3  */
4 #include <sys/types.h>
5 #include <sys/socket.h>
6 #include <netinet/in.h>
7 #include <arpa/inet.h>
8 #include <time.h>
9 #include <string.h>
10 #include <stdio.h>
11 #include <unistd.h>
12 #include <stdlib.h>
13
14 #define MCAST_PORT 8888
15 #define MCAST_ADDR "239.255.255.11"
16 #define MCAST_DATA "BROADCAST TEST DATA" /*多播发送的数据*/
17 #define MCAST_INTERVAL 1 /*发送间隔时间*/
18
19 int main(int argc, char*argv)
20 {
21     struct sockaddr_in mcast_addr;
22     int fd = socket(AF_INET, SOCK_DGRAM, 0); /*建立套接字*/
23     if (fd == -1)
24     {
25         perror("socket()");
26         exit(1);
27     }
28
29     memset(&mcast_addr, 0, sizeof(mcast_addr)); /*初始化IP多播地址为0*/
30     mcast_addr.sin_family = AF_INET; /*设置协议族行为AF*/
31     mcast_addr.sin_addr.s_addr = inet_addr(MCAST_ADDR); /*设置多播IP地址*/
32     mcast_addr.sin_port = htons(MCAST_PORT); /*设置多播端口*/
33
34     /*向多播地址发送数据*/
35     while(1)
36     {
37         int n = sendto(fd, MCAST_DATA, sizeof(MCAST_DATA), 0, (struct sockaddr*)&mcast_addr, sizeof(mcast_addr));
38         if( n < 0)
39         {
40             perror("sendto()");
41             exit(1);
42         }
43         sleep(MCAST_INTERVAL); /*等待一段时间*/
44     }
45
46     return 0;
47 }
```

客户端

```
1 /*
2  *broadcast_client.c - 多播的客户端
3  */
4 #include <sys/types.h>
5 #include <sys/socket.h>
6 #include <netinet/in.h>
7 #include <arpa/inet.h>
8 #include <time.h>
9 #include <string.h>
10 #include <stdio.h>
11 #include <unistd.h>
12 #include <stdlib.h>
13
14 #define MCAST_PORT 8888
15 #define MCAST_ADDR "239.255.255.11"
```

目录

从零基础写一个RTSP服务器（七）多播...

文章目录

一、多播

1.1 多播简介

1.2 多播示例

二、多播的udp描述文件

三、多播传输RTP包实现

3.1 实现

3.2 源码

四、测试

分类专栏

C/C++

1篇

STL源码剖析

18篇

MySQL

4篇

Linux内核

14篇

分布式

nginx

从零基础写一个RTSP服...

10篇

live555源码分析与应用

9篇

Linux驱动

19篇

```
16 #define MCAST_INTERVAL 1 /*发送间隔时间*/
17 #define BUFF_SIZE 256 /*接收缓冲区大小*/
18
19 int main(int argc, char*argv[])
20 {
21     struct sockaddr_in local_addr; /*本地地址*/
22
23     int fd = socket(AF_INET, SOCK_DGRAM, 0); /*建立套接字*/
24     if (fd == -1)
25     {
26         perror("socket()");
27         exit(1);
28     }
29
30     int yes = 1;
31     if (setsockopt(fd, SOL_SOCKET, SO_REUSEADDR, &yes, sizeof(yes)) < 0)
32     {
33         perror("Reusing ADDR failed");
34         exit(1);
35     }
36
37     /*初始化本地地址*/
38     memset(&local_addr, 0, sizeof(local_addr));
39     local_addr.sin_family = AF_INET;
40     local_addr.sin_addr.s_addr = htonl(INADDR_ANY);
41     local_addr.sin_port = htons(MCAST_PORT);
42
43     /*绑定socket*/
44     int err = bind(fd, (struct sockaddr*)&local_addr, sizeof(local_addr));
45     if (err < 0)
46     {
47         perror("bind()");
48         exit(1);
49     }
50
51     /*设置循环许可*/
52     int loop = 1;
53     err = setsockopt(fd, IPPROTO_IP, IP_MULTICAST_LOOP, &loop, sizeof(loop));
54     if (err < 0)
55     {
56         perror("setsockopt():IP_MULTICAST_LOOP");
57         exit(1);
58     }
59
60     /*加入多播组*/
61     struct ip_mreq mreq;
62     mreq.imr_multiaddr.s_addr = inet_addr(MCAST_ADDR); /*多播地址*/
63     mreq.imr_interface.s_addr = htonl(INADDR_ANY); /*本地网络接口为默认*/
64
65     /*将本机加入多播组*/
66     err = setsockopt(fd, IPPROTO_IP, IP_ADD_MEMBERSHIP, &mreq, sizeof(mreq));
67     if (err < 0)
68     {
69         perror("setsockopt():IP_ADD_MEMBERSHIP");
70         exit(1);
71     }
72
73     int times = 0;
74     int addr_len = sizeof(local_addr);
75     char buff[BUFF_SIZE];
76     int n = 0;
77
78     /*循环接收多播组的消息，5次后退出*/
79     for (times = 0; times < 5; times++)
80     {
81         memset(buff, 0, BUFF_SIZE); /*清空接收缓冲区*/
82
83         /*接收数据*/
84         n = recvfrom(fd, buff, BUFF_SIZE, 0, (struct sockaddr*)&local_addr, &addr_len);
85         if (n == -1)
86         {
87             perror("recvfrom()");
88         }
89         /*打印信息*/
90         printf("Recv %dst message from server:%s\n", times, buff);
91         sleep(MCAST_INTERVAL);
92     }
93
94     /*退出多播组*/
95     err = setsockopt(fd, IPPROTO_IP, IP_DROP_MEMBERSHIP, &mreq, sizeof(mreq));
96
97     close(fd);
98     return 0;
99 }
```

需要好好看一看这个示例，了解多播编程的流程

二、多播的sdp描述文件

```
1 m=video 9832 RTP/AVP 96
2 c=IN IP4 239.255.255.11/255
3 a=rtpmap:96 H264/90000
4 a=framerate:25
```

这是一个多播H.264的sdp文件

相关的讲解前面已经讲了很多了（详情看：[从零开始写一个RTSP服务器（一）不一样的RTSP协议讲解](#)）

这里需要注意的是目的端口号为 **9832**，多播IP为 **239.255.255.11**

三、多播传输RTP包实现

3.1 实现

对于我们来说，我们实现的是服务端，所以非常简单，无需过多操作，只需要把[从零开始写一个RTSP服务器（三）RTP传输H.264中的目的IP和目的端口](#)改为目的多播IP和目的多播端口就行

关于RTP打包这里不再重复，前面已经讲解得非常详细了

3.2 源码

multicast.c

```
1 /*
2  * 作者: _3T_
3  * 博客: https://blog.csdn.net/weixin_42462282
4  */
5
6 #include <stdio.h>
7 #include <stdlib.h>
8 #include <sys/types.h>
9 #include <sys/socket.h>
10 #include <sys/stat.h>
11 #include <fcntl.h>
12 #include <unistd.h>
13 #include <string.h>
14
15 #include "rtp.h"
16
17 #define H264_FILE_NAME "test.h264"
18 #define MULTICAST_IP "239.255.255.11"
19 #define MULTICAST_PORT 9832
20 #define FPS 25
21
22 static inline int startCode3(char* buf)
23 {
24     if(buf[0] == 0 && buf[1] == 0 && buf[2] == 1)
25         return 1;
26     else
27         return 0;
28 }
29
30 static inline int startCode4(char* buf)
31 {
32     if(buf[0] == 0 && buf[1] == 0 && buf[2] == 0 && buf[3] == 1)
33         return 1;
34     else
35         return 0;
36 }
37
38 static char* findNextStartCode(char* buf, int len)
39 {
40     int i;
41
42     if(len < 3)
43         return NULL;
44
45     for(i = 0; i < len-3; ++i)
46     {
47         if(startCode3(buf) || startCode4(buf))
48             return buf;
49     }
```

```

51     }
52     }
53     if(startCode3(buf))
54         return buf;
55
56     return NULL;
57 }
58
59 static int getFrameFromH264File(int fd, char* frame, int size)
60 {
61     int rSize, frameSize;
62     char* nextStartCode;
63
64     if(fd < 0)
65         return fd;
66
67     rSize = read(fd, frame, size);
68     if(!startCode3(frame) && !startCode4(frame))
69         return -1;
70
71     nextStartCode = findNextStartCode(frame+3, rSize-3);
72     if(!nextStartCode)
73     {
74         lseek(fd, 0, SEEK_SET);
75         frameSize = rSize;
76     }
77     else
78     {
79         frameSize = (nextStartCode - frame);
80         lseek(fd, frameSize - rSize, SEEK_CUR);
81     }
82
83     return frameSize;
84 }
85
86 static int createUdpSocket()
87 {
88     int fd;
89     int on = 1;
90
91     fd = socket(AF_INET, SOCK_DGRAM, 0);
92     if(fd < 0)
93         return -1;
94
95     setsockopt(fd, SOL_SOCKET, SO_REUSEADDR, (const char*)0, sizeof(on));
96
97     return fd;
98 }
99
100 static int rtpSendH264Frame(int socket, char* ip, int16_t port,
101                             struct RtpPacket* rtpPacket, uint8_t* frame, uint32_t frameSize)
102 {
103     uint8_t naluType; // nalu第一个字节
104     int sendBytes = 0;
105     int ret;
106
107     naluType = frame[0];
108
109     if (frameSize <= RTP_MAX_PKT_SIZE) // nalu长度小于最大包长: 单一NALU单元模式
110     {
111         /*
112          * 0 1 2 3 4 5 6 7 8 9
113          * +-----+-----+-----+-----+
114          * |F|NRI| Type | 0 single NAL unit ... |
115          * +-----+-----+-----+-----+
116          */
117         memcpy(rtpPacket->payload, frame, frameSize);
118         ret = rtpSendPacket(socket, ip, port, rtpPacket, frameSize);
119         if(ret < 0)
120             return -1;
121
122         rtpPacket->rtpHeader.seq++;
123         sendBytes += ret;
124         if ((naluType & 0x1F) == 7 || (naluType & 0x1F) == 8) // 如果是SPS、PPS就不需要加时间戳
125             goto out;
126     }
127     else // nalu长度小于最大包长: 分片模式
128     {
129         /*
130          * 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3
131          * +-----+-----+-----+-----+-----+-----+
132          * | FU indicator | FU header | FU payload ... |
133          * +-----+-----+-----+-----+-----+
134          */
135
136         /*
137          * FU Indicator
138          * 0 1 2 3 4 5 6 7
139          * +-----+-----+
140          * |F|NRI| Type |
141          * +-----+
142          */
143
144         /*
145          * FU Header
146          * 0 1 2 3 4 5 6 7
147          * +-----+-----+
148          * |S|E|R| Type |
149          * +-----+-----+
150          */
151
152         int pktNum = frameSize / RTP_MAX_PKT_SIZE; // 有几个完整的包
153         int remainPktSize = frameSize % RTP_MAX_PKT_SIZE; // 剩余不完整包的大小
154         int i, pos = 1;
155
156         /* 发送完整的包 */
157         for (i = 0; i < pktNum; i++)
158         {
159             rtpPacket->payload[0] = (naluType & 0x60) | 28;
160             rtpPacket->payload[1] = naluType & 0x1F;
161
162             if (i == 0) // 第一包数据
163                 rtpPacket->payload[1] |= 0x80; // start
164             else if (remainPktSize == 0 && i == pktNum - 1) // 最后一包数据
165                 rtpPacket->payload[1] |= 0x40; // end
166
167             memcpy(rtpPacket->payload+2, frame-pos, RTP_MAX_PKT_SIZE);
168             ret = rtpSendPacket(socket, ip, port, rtpPacket, RTP_MAX_PKT_SIZE+2);
169             if(ret < 0)
170                 return -1;
171
172             rtpPacket->rtpHeader.seq++;
173             sendBytes += ret;
174             pos += RTP_MAX_PKT_SIZE;
175         }
176
177         /* 发送剩余的数据 */
178         if (remainPktSize > 0)
179         {
180             rtpPacket->payload[0] = (naluType & 0x60) | 28;
181             rtpPacket->payload[1] = naluType & 0x1F;
182             rtpPacket->payload[1] |= 0x40; //end
183
184             memcpy(rtpPacket->payload+2, frame-pos, remainPktSize+2);
185             ret = rtpSendPacket(socket, ip, port, rtpPacket, remainPktSize+2);
186             if(ret < 0)
187                 return -1;
188
189             rtpPacket->rtpHeader.seq++;
190             sendBytes += ret;
191         }
192     }
193 }
194
195 out:
196
197     return sendBytes;
198 }
199
200 int main(int argc, char* argv[])
201 {
202     int socket;
203     int fd;
204     int fps = 25;
205     int startCode;
206     struct RtpPacket* rtpPacket;
207     uint8_t* frame;
208     uint32_t frameSize;
209
210     fd = open(H264_FILE_NAME, O_RDONLY);
211     if(fd < 0)
212     {
213         printf("failed to open %s\n", H264_FILE_NAME);

```

```

214     return -1;
215 }
216
217 socket = createIdgSocket();
218 if(socket < 0)
219 {
220     printf("failed to create socket\n");
221     return -1;
222 }
223
224 rtpPacket = (struct RtpPacket*)malloc(500000);
225 frame = (uint8_t*)malloc(500000);
226
227 rtpHeaderInit(rtpPacket, 0, 0, 0, RTP_VERSION, RTP_PAYLOAD_TYPE_H264, 0,
228              0, 0, 0x88023423);
229
230 while(1)
231 {
232     frameSize = getFrameFromH264File(fd, frame, 500000);
233     if(frameSize < 0)
234     {
235         printf("read err\n");
236         continue;
237     }
238
239     if(startCode3(frame))
240         startCode = 3;
241     else
242         startCode = 4;
243
244     frameSize -= startCode;
245     rtpSendH264Frame(socket, MULTICAST_IP, MULTICAST_PORT,
246                    rtpPacket, frame+startCode, frameSize);
247     rtpPacket->rtpHeader.timestamp += 90000/FPS;
248
249     usleep(1000*1000/fps);
250 }
251
252 free(rtpPacket);
253 free(frame);
254
255 return 0;
256 }

```

rtp.h

```

1  /*
2  * 作者: _3T_
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
16 #define RTP_MAX_PKT_SIZE           1400
17
18 /*
19 *
20 *   0               1               2               3
21 *   7 6 5 4 3 2 1 0|7 6 5 4 3 2 1 0|7 6 5 4 3 2 1 0|7 6 5 4 3 2 1 0
22 *   +-----+-----+-----+-----+
23 *   |V2|P|X| CC |M| PT | sequence number |
24 *   +-----+-----+-----+-----+
25 *   | timestamp |
26 *   +-----+-----+-----+-----+
27 *   | synchronization source (SSRC) identifier |
28 *   +-----+-----+-----+-----+
29 *   | contributing source (CSRC) identifiers |
30 *   :               ....               :
31 *   +-----+-----+-----+-----+
32 *
33 */
34 struct RtpHeader
35 {
36     /* byte 0 */
37     uint8_t csrclen:4;
38     uint8_t extension:1;
39     uint8_t padding:1;
40     uint8_t version:2;
41
42     /* byte 1 */
43     uint8_t payloadType:7;
44     uint8_t marker:1;
45
46     /* bytes 2,3 */
47     uint16_t seq;
48
49     /* bytes 4-7 */
50     uint32_t timestamp;
51
52     /* bytes 8-11 */
53     uint32_t ssrc;
54 };
55
56 struct RtpPacket
57 {
58     struct RtpHeader rtpHeader;
59     uint8_t payload[0];
60 };
61
62 void rtpHeaderInit(struct RtpPacket* rtpPacket, uint8_t csrclen, uint8_t extension,
63                  uint8_t padding, uint8_t version, uint8_t payloadType, uint8_t marker,
64                  uint16_t seq, uint32_t timestamp, uint32_t ssrc);
65 int rtpSendPacket(int socket, const char* ip, int16_t port, struct RtpPacket* rtpPacket, uint32_t dataSize);
66
67 #endif // _RTP_H_

```

rtp.c

```

1  /*
2  * 作者: _3T_
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 sockadd_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         return ret;
49     }
50 }
```

四、测试

将 `multicast.c`、`rtsp.h`、`rtsp.c` 保存下来

编译运行，程序默认会打开 `test.h264` 的视频文件，如果你没有视频源的话，可以从 `RtspServer` 的 `example` 目录下获取

```
1 # gcc multicast.c rtsp.c
2 # ./a.out
```

将sdp保存为multicast.sdp

```
1 a=type:broadcast
2 a=rtcp-mcast: reflection
3 m=video 9832 RTP/AVP 96
4 c=IN IP4 239.255.255.11/255
5 a=rtmpmap:96 H264/900000
6 a=framerate:25
```

使用vlc打开sdp文件

```
1 # vlc multicast.sdp
```



RTSP/RTMP 服务器+客户端 C++ 源代码	03-29
RTSP/RTMP服务器+客户端 C++ 源代码	
rtsp客户端程序解析rtsp-tcp协议	04-10
这是一个rtsp协议解析的简单程序，通过与服务器进行rtsp交互，PLAY方法以后解析tcp生成h264文件能播放(fmpeg命令行)。结构简单，一看即懂，没有...	
使用netty进行rtsp服务端开发zip	01-02
使用netty进行rtsp服务端开发 一个使用netty写的rtsp服务器。 目前支持H264、H265、AAC格式的流文件上传与存储。 H264、H265、AAC格式流文件的...	
rtsp抓包直播-tcp(udp)	11-23
rtsp、直播、rtcp(udp)。 关联文章：https://mp.csdn.net/postedit	
rtsp 报文转发、RTPS代理与转发服务	weixin_42508903的博文 366
Proxy介绍利用beverend实现网络连接和线程池。通过tcp连接的方式实现rtsp消息转发，再通过udp连接进行rtcp与rtsp转发。 报文解析使用到了Qt库，请参...	
7、多播的RTSP服务器	huabaochen的博文 804
本文目的：实现一个多播传输H.264的RTSP服务器 一、多播的RTSP交互过程 我们在前面文章从零开始写一个RTSP服务器（二）RTSP协议的实现中实...	
3.2 RTSP 协议详解	每天都在捕鱼 1335
RTSP协议详解 一般是服务器 是被动 在一些游戏服务器 是主动 RTSP协议是基于RTP(数据包)和RTCP(控制命令 UDP)之上的 RTSP使用RTP传输媒体数据...	
基于Netty的的RTSP服务器（播放H264和AAC）	while_black的博文 1279
以上是我需要大牛的设计思路的文章，我的设计思路本质上还是来源于大牛，因此不再重复， 我在他的基础上加上了对AAC音频播放的功能，即同时播...	
RTSP与netty	wsl_whu的专栏 6388
rtsp协议的格式与rtcp协议的格式是一样的， 因此可以使用netty的http解析器来处理rtsp交互数据。 netty中自带了一个RtpspDecoder，但是它几乎没啥事...	
基于Java的RTSP 服务	qq_332103388的博文 8553
因公司项目需求，需要开发一个基于java的RTSP服务，支持RTTUPD和RTTPT/CP模式。本人在这方面也是小白一个，于是各种谷歌、百度查...	
Rtsp之tcp包解析	1693
rtcp包解析，纠结了好久好久，一直没有好的解决方案，最近琢磨了一下，下面给我的一些思路。 rtcp包解析存在以下问题： 1、包的序号会存在乱序的可...	
流媒体传输协议介绍	machhh的专栏 5183
流媒体传输协议介绍一、RTSP协议介绍什么是rtsp? RTSP协议以客户/服务器方式工作，，如：暂停继续、后退、前进等。它是一个多媒体播放控制协议...	
rtsp 基于RTP 解包代码	Nine days 638
buf_in 一般是由 live555 client 获得的数据，buf_out是解包成 H264编码格式的数据，该数据直接发给解码器 就完成了解码的流程。 rtcp解包代码def...	
rtsp组播搭建和rtcp组播实现	blweenyf的专栏 6652
rtsp组播在一些场景下比单播更合适，比如电子教室等，单播每一路都要占相同带宽，带宽要求比较高，并且路数多了也容易丢包。组播只占一路带宽。...	
RTSP学习之RTTP(实时传输协议)简介【整理】	zg8893的专栏 3511
1、RTTP协议简介 RTP(Real-timeTransport Protocol)，由 IETF（http://www.ietf.org） 定义为传输音频、视频、模拟数据...	
从零编写rtsp-client端	Canok 2071
目录 简介： github工程源码： rtsp协议简介 rtsp相关的一些问题 简介： 使用live555接受rtsp流，发现在 使用udp传输的过程中，h264数据丢帧，播放端...	
java应用netty服务端和客户端	05-30
java应用netty服务端和客户端示例。 客户端和服务端的node对象目录必须一致	
Springboot实现Netty-websocket+rtsp+ffmpeg+jsmpeg.js实现视频播放支持ws和http模式 最新发布	weixin_40093962的博文 6115
思路 1、前端是无法直接接收rtsp源流的视频，所以需要ffmpeg进行转码。 2、ffmpeg只能推送TCP或者HTTP协议还不支持ws协议。 大致流程图。 ...	
java开发h265.EasyMedia: Springboot、netty实现的http-flv、websocket-flv直播点播，支持rtsp、h264、h26..._weixin_29110051的博文 1404	
EasyMedia介绍Springboot、netty实现的http-flv、websocket-flv流媒体服务(可用于直播点播)，支持rtsp、h264、h265等、rtmp等多种源，h5和h5播放(不...	

“相关推荐”对你有帮助么？

非常没帮助 没帮助 一般 有帮助 非常有帮助

©2022 CSDN 皮肤主题：数字20 设计师：CSDN官方博普 返回首页

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19046568号 京公网安备11010502030143 网络信息安全管理中心 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 版权与隐私声明 版权申诉 出版物经营许可证 营业执照 ©1999-2022北京创新乐知网络技术有限公司