原创 风雨兼程8023 于 2020-06-02 19:17:56 发布

分类专栏：音视频编解码 文章标签：rtsp h.264

把RTSP协议的一些分析（三）——简单的rtsp服务器的实现和RTSP协议的一些分析（四）——RTP传输H.264结合在一起，就是本文的内容，这里不做多介绍，直接上代码。

rtp.h

```c
#ifndef _RTP_H_
#define _RTP_H_
#include <stdint.h>

#define RTP_VESION              2

#define RTP_PAYLOAD_TYPE_H264   96
#define RTP_PAYLOAD_TYPE_AAC    97

#define RTP_HEADER_SIZE         12
#define RTP_MAX_PKT_SIZE        1400

/*
 *
 *    0                   1                   2                   3
 *    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
 *   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 *   |V=2|P|X|  CC   |M|     PT      |       sequence number         |
 *   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 *   |                           timestamp                           |
 *   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 *   |           synchronization source (SSRC) identifier            |
 *   +=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
 *   |            contributing source (CSRC) identifiers             |
 *   :                             ....                              :
 *   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 *
 */
struct RtpHeader
{
    /* byte 0 */
    uint8_t csrcLen:4;
    uint8_t extension:1;
    uint8_t padding:1;
    uint8_t version:2;

    /* byte 1 */
    uint8_t payloadType:7;
    uint8_t marker:1;

    /* bytes 2,3 */
    uint16_t seq;

    /* bytes 4-7 */
    uint32_t timestamp;

    /* bytes 8-11 */
    uint32_t ssrc;
};

struct RtpPacket
{
    struct RtpHeader rtpHeader;
    uint8_t payload[0];
};

void rtpHeaderInit(struct RtpPacket* rtpPacket, uint8_t csrcLen, uint8_t extension,
                    uint8_t padding, uint8_t version, uint8_t payloadType, uint8_t marker,
                   uint16_t seq, uint32_t timestamp, uint32_t ssrc);
int rtpSendPacket(int socket, char* ip, int16_t port, struct RtpPacket* rtpPacket, uint32_t dataSize);

#endif
```

rtp.c

```c
/*
 * 浣滆€咃細_JT_
 * 鍗氬  锛歨ttps://blog.csdn.net/weixin_42462202
 */

#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#include <arpa/inet.h>

#include "rtp.h"

void rtpHeaderInit(struct RtpPacket* rtpPacket, uint8_t csrcLen, uint8_t extension,
                    uint8_t padding, uint8_t version, uint8_t payloadType, uint8_t marker,
                   uint16_t seq, uint32_t timestamp, uint32_t ssrc)
{
    rtpPacket->rtpHeader.csrcLen = csrcLen;
    rtpPacket->rtpHeader.extension = extension;
    rtpPacket->rtpHeader.padding = padding;
    rtpPacket->rtpHeader.version = version;
    rtpPacket->rtpHeader.payloadType =  payloadType;
    rtpPacket->rtpHeader.marker = marker;
    rtpPacket->rtpHeader.seq = seq;
    rtpPacket->rtpHeader.timestamp = timestamp;
    rtpPacket->rtpHeader.ssrc = ssrc;
}

int rtpSendPacket(int socket, char* ip, int16_t port, struct RtpPacket* rtpPacket, uint32_t dataSize)
{
    struct sockaddr_in addr;
    int ret;

    addr.sin_family = AF_INET;
    addr.sin_port = htons(port);
    addr.sin_addr.s_addr = inet_addr(ip);

    rtpPacket->rtpHeader.seq = htons(rtpPacket->rtpHeader.seq);
    rtpPacket->rtpHeader.timestamp = htonl(rtpPacket->rtpHeader.timestamp);
    rtpPacket->rtpHeader.ssrc = htonl(rtpPacket->rtpHeader.ssrc);

```

```
    ret = sendto(socket, (void*)rtpPacket, dataSize+RTP_HEADER_SIZE, 0,
                 (struct sockaddr*)&addr, sizeof(addr));

    rtpPacket->rtpHeader.seq = ntohs(rtpPacket->rtpHeader.seq);
    rtpPacket->rtpHeader.timestamp = ntohl(rtpPacket->rtpHeader.timestamp);
    rtpPacket->rtpHeader.ssrc = ntohl(rtpPacket->rtpHeader.ssrc);

    return ret;
}
```

h264_rtsp_server.c

```
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <string.h>
#include <time.h>

#include <fcntl.h>
#include <unistd.h>

#include <sys/types.h>
#include <sys/socket.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include "rtp.h"

#define H264_FILE_NAME  "test.h264"
#define CLIENT_IP       "10.14.33.103"
#define CLIENT_PORT     9832

#define BUF_MAX_SIZE    (1024*1024)
#define SERVER_RTP_PORT  55532
#define SERVER_RTCP_PORT 55533
#define SERVER_PORT     8554

#define HDR_CSEQ        "CSeq"
#define HDR_RANGE       "Range"
#define HDR_SESSION     "Session"
#define HDR_TRANSPORT   "Transport"

#define RTSP_METHOD_OPTIONS  "OPTIONS"
#define RTSP_METHOD_DESCRIBE "DESCRIBE"
#define RTSP_METHOD_SETUP   "SETUP"
#define RTSP_METHOD_PLAY    "PLAY"

typedef struct _RTSP_HDR_PARAM_
{
    unsigned int rtsp_cseq;
    unsigned int rtsp_session;
    unsigned int rtsp_clientRtpPort;
    unsigned int rtsp_clientRtcpPort;
    unsigned char rtsp_method[20];
    char rtsp_InBuffer[BUF_MAX_SIZE]; /*接收缓冲区*/
    char rtsp_OutBuffer[BUF_MAX_SIZE];/*发送缓冲区*/
}RTSP_HDR_PARAM;


static int createTcpSocket()
{
    int sockfd;
    int on = 1;

    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if(sockfd < 0)
        return -1;

    setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, (const char*)&on, sizeof(on));

    return sockfd;
}

static int createUdpSocket()
{
    int sockfd;
    int on = 1;

    sockfd = socket(AF_INET, SOCK_DGRAM, 0);
    if(sockfd < 0)
        return -1;

    setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, (const char*)&on, sizeof(on));

    return sockfd;
}

static int bindSocketAddr(int sockfd, const char* ip, int port)
{
    struct sockaddr_in addr;

     addr.sin_family    = AF_INET;
     addr.sin_port      = htons(port);
    addr.sin_addr.s_addr  = inet_addr(ip);

    if(bind(sockfd, (struct sockaddr *)&addr, sizeof(struct sockaddr)) < 0)
        return -1;

    return 0;
}

static int acceptClient(int sockfd, char* ip, int* port)
{
    int clientfd;
    socklen_t len = 0;
    struct sockaddr_in addr;

    memset(&addr, 0, sizeof(addr));
    len = sizeof(addr);

    clientfd = accept(sockfd, (struct sockaddr *)&addr, &len);
    if(clientfd < 0)
        return -1;

    strcpy(ip, inet_ntoa(addr.sin_addr));
    *port = ntohs(addr.sin_port);

    return clientfd;
}

static int RTSP_HandleMethodOPTIONS(RTSP_HDR_PARAM* pstRtspHdrParam)
{
```

```
    sprintf(pstRtspHdrParam->rtsp_OutBuffer,
                    "RTSP/1.0 200 OK\r\n"
                    "CSeq: %d\r\n"
                    "Public: OPTIONS, DESCRIBE, SETUP, PLAY\r\n"
                    "\r\n",
                    pstRtspHdrParam->rtsp_cseq);

    return 0;

}

static int RTSP_HandleMethodDESCRIBE(RTSP_HDR_PARAM* pstRtspHdrParam)
{
    char sdp[500];
    char localIp[100];
    char url[255];
    sscanf(pstRtspHdrParam->rtsp_InBuffer, "%*s %256s",url);
    sscanf(pstRtspHdrParam->rtsp_InBuffer, "DESCRIBE rtsp://%[^:]:", localIp);
    sprintf(sdp, "v=0\r\n"
                 "o=- 9%ld 1 IN IP4 %s\r\n"
                 "t=0 0\r\n"
                 "a=control:*\r\n"
                 "m=video 0 RTP/AVP 96\r\n"
                 "a=rtpmap:96 H264/90000\r\n"
                 "a=control:track0\r\n",
                 time(NULL), localIp);

    sprintf(pstRtspHdrParam->rtsp_OutBuffer,
                    "RTSP/1.0 200 OK\r\n"
                    "CSeq: %d\r\n"
                    "Content-Base: %s\r\n"
                    "Content-type: application/sdp\r\n"
                    "Content-length: %d\r\n\r\n"
                    "%s",
                    pstRtspHdrParam->rtsp_cseq,
                    url,
                    strlen(sdp),
                    sdp);
    return 0;

}

static int RTSP_HandleMethodSETUP(RTSP_HDR_PARAM* pstRtspHdrParam)
{
    char *p = NULL;
    if((p = strstr(pstRtspHdrParam->rtsp_InBuffer,HDR_TRANSPORT)))
    {
        sscanf(p, "Transport: RTP/AVP;unicast;client_port=%d-%d\r\n",
    &pstRtspHdrParam->rtsp_clientRtpPort, &pstRtspHdrParam->rtsp_clientRtcpPort);
    }

    sprintf(pstRtspHdrParam->rtsp_OutBuffer,
                    "RTSP/1.0 200 OK\r\n"
                    "CSeq: %d\r\n"
                    "Transport: RTP/AVP;unicast;client_port=%d-%d;server_port=%d-%d\r\n"
                    "Session: 66334873\r\n"
                    "\r\n",
                    pstRtspHdrParam->rtsp_cseq,
                    pstRtspHdrParam->rtsp_clientRtpPort,
                    pstRtspHdrParam->rtsp_clientRtcpPort,
                    SERVER_RTP_PORT,
                    SERVER_RTCP_PORT);
    return 0;
}

static int RTSP_HandleMethodPLAY(RTSP_HDR_PARAM* pstRtspHdrParam)
{
    sprintf(pstRtspHdrParam->rtsp_OutBuffer,
                    "RTSP/1.0 200 OK\r\n"
                    "CSeq: %d\r\n"
                    "Range: npt=0.000-\r\n"
                    "Session: 66334873; timeout=60\r\n"
                    "\r\n",
                    pstRtspHdrParam->rtsp_cseq);
    return 0;
}


static int RTSP_HandleMethod(RTSP_HDR_PARAM* pstRtspHdrParam)
{
    if(!strcmp(pstRtspHdrParam->rtsp_method,RTSP_METHOD_OPTIONS))
    {
        if(RTSP_HandleMethodOPTIONS(pstRtspHdrParam))
        {
            printf("failed to handle OPTIONS\n");
            return -1;
        }
    }
    else if(!strcmp(pstRtspHdrParam->rtsp_method,RTSP_METHOD_DESCRIBE))
    {
        if(RTSP_HandleMethodDESCRIBE(pstRtspHdrParam))
        {
            printf("failed to handle DESCRIBE\n");
            return -1;
        }
    }
    else if(!strcmp(pstRtspHdrParam->rtsp_method,RTSP_METHOD_SETUP))
    {
        if(RTSP_HandleMethodSETUP(pstRtspHdrParam))
        {
            printf("failed to handle SETUP\n");
            return -1;
        }
    }
    else if(!strcmp(pstRtspHdrParam->rtsp_method,RTSP_METHOD_PLAY))
    {
        if(RTSP_HandleMethodPLAY(pstRtspHdrParam))
        {
            printf("failed to handle PLAY\n");
            return -1;
        }
    }

    return 0;

}

static inline int startCode4(char* buf)
{
    if(buf[0] == 0 && buf[1] == 0 && buf[2] == 0 && buf[3] == 1)
        return 1;
    else
        return 0;
```

```
}

static char* findNextStartCode(char* buf, int len)
{
    int i;

    if(len < 4)
        return NULL;

    for(i = 0; i < len-4; i++)
    {
        if(startCode4(buf))
            return buf;

        buf++;
    }

    return NULL;
}


static int getFrameFromH264File(int fd, char* frame, int size)
{
    int rSize, frameSize;
    char* nextStartCode;

    if(fd < 0)
        return fd;

    rSize = read(fd, frame, size);
    if(!startCode4(frame))
        return -1;

    nextStartCode = findNextStartCode(frame+4, rSize-4);
    if(!nextStartCode)
    {
        lseek(fd, 0, SEEK_SET);
        frameSize = rSize;
    }
    else
    {
        frameSize = (nextStartCode-frame);
        lseek(fd, frameSize-rSize, SEEK_CUR);
    }

    return frameSize;
}

static int rtpSendH264Frame(int socket, char* ip, int16_t port,
                            struct RtpPacket* rtpPacket, uint8_t* frame, uint32_t frameSize)
{
    uint8_t naluType; // nalu第一个字节
    int sendBytes = 0;
    int ret;

    naluType = frame[0];

    if (frameSize <= RTP_MAX_PKT_SIZE) // nalu长度小于最大包场：单一NALU单元模式
    {
        /*
         *   0 1 2 3 4 5 6 7 8 9
         *  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
         *  |F|NRI|  Type   | a single NAL unit ... |
         *  +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
         */
        memcpy(rtpPacket->payload, frame, frameSize);
        ret = rtpSendPacket(socket, ip, port, rtpPacket, frameSize);
        if(ret < 0)
            return -1;

        rtpPacket->rtpHeader.seq++;
        sendBytes += ret;
        if ((naluType & 0x1F) == 7 || (naluType & 0x1F) == 8) // 如果是SPS、PPS就不需要加时间戳
            goto out;
    }
    else // nalu长度小于最大包场：分片模式
    {
        /*
         *  0                   1                   2
         *  0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3
         * +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
         * | FU indicator  |   FU header   |   FU payload   ...  |
         * +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
         */

        /*
         *     FU Indicator
         *    0 1 2 3 4 5 6 7
         *   +-+-+-+-+-+-+-+-+
         *   |F|NRI|  Type   |
         *   +---------------+
         */

        /*
         *      FU Header
         *    0 1 2 3 4 5 6 7
         *   +-+-+-+-+-+-+-+-+
         *   |S|E|R|  Type   |
         *   +---------------+
         */

        int pktNum = frameSize / RTP_MAX_PKT_SIZE;       // 有几个完整的包
        int remainPktSize = frameSize % RTP_MAX_PKT_SIZE; // 剩余不完整包的大小
        int i, pos = 1;

        /* 发送完整的包 */
        for (i = 0; i < pktNum; i++)
        {
            rtpPacket->payload[0] = (naluType & 0x60) | 28;
            rtpPacket->payload[1] = naluType & 0x1F;

            if (i == 0) //第一包数据
                rtpPacket->payload[1] |= 0x80; // start
            else if (remainPktSize == 0 && i == pktNum - 1) //最后一包数据
                rtpPacket->payload[1] |= 0x40; // end

            memcpy(rtpPacket->payload+2, frame+pos, RTP_MAX_PKT_SIZE);
            ret = rtpSendPacket(socket, ip, port, rtpPacket, RTP_MAX_PKT_SIZE+2);
            if(ret < 0)
                return -1;

            rtpPacket->rtpHeader.seq++;
            sendBytes += ret;
```

```
            pos += RTP_MAX_PKT_SIZE;
        }

        /* 发送剩余的数据 */
        if (remainPktSize > 0)
        {
            rtpPacket->payload[0] = (naluType & 0x60) | 28;
            rtpPacket->payload[1] = naluType & 0x1F;
            rtpPacket->payload[1] |= 0x40; //end

            memcpy(rtpPacket->payload+2, frame+pos, remainPktSize+2);
            ret = rtpSendPacket(socket, ip, port, rtpPacket, remainPktSize+2);
            if(ret < 0)
                return -1;

            rtpPacket->rtpHeader.seq++;
            sendBytes += ret;
        }
    }

out:

    return sendBytes;
}

/*
*OPTIONS rtsp://10.1.74.190:8554 RTSP/1.0
*CSeq: 2
*User-Agent: LibVLC/3.0.4 (LIVE555 Streaming Media v2016.11.28)
*
*DESCRIBE rtsp://10.1.74.190:8554 RTSP/1.0
*CSeq: 3
*User-Agent: LibVLC/3.0.4 (LIVE555 Streaming Media v2016.11.28)
*Accept: application/sdp
*
*
*SETUP rtsp://10.1.74.190:8554/track0 RTSP/1.0
*CSeq: 4
*User-Agent: LibVLC/3.0.4 (LIVE555 Streaming Media v2016.11.28)
*Transport: RTP/AVP;unicast;client_port=63244-63245
*
*PLAY rtsp://10.1.74.190:8554 RTSP/1.0
*CSeq: 5
*User-Agent: LibVLC/3.0.4 (LIVE555 Streaming Media v2016.11.28)
*Session: 66334873
*Range: npt=0.000-
*
*/
static void doClient(int clientSockfd, const char* clientIP, int clientPort,
                        int serverRtpSockfd, int serverRtcpSockfd)
{
    int recvLen;
    int iRet = -1;
    char *p = NULL;
 RTSP_HDR_PARAM stRtspHdrParam;

    while(1)
    {
      recvLen = recv(clientSockfd, stRtspHdrParam.rtsp_InBuffer, BUF_MAX_SIZE, 0);
        if(recvLen <= 0)
            goto out;
  stRtspHdrParam.rtsp_InBuffer[recvLen] = '\0';
        printf("---------------C->S--------------\n");
        printf("%s\n",stRtspHdrParam.rtsp_InBuffer);

        if (!sscanf(stRtspHdrParam.rtsp_InBuffer, "%s",stRtspHdrParam.rtsp_method))
        {
            printf("prase url failed\n");
            goto out;
        }

        if((p = strstr(stRtspHdrParam.rtsp_InBuffer,HDR_CSEQ)) != NULL)
     {
            if(!sscanf(p, "%*s %d",&stRtspHdrParam.rtsp_cseq))
            {
                printf("prase Cseq failed\n");
                goto out;
            }
        }

  iRet = RTSP_HandleMethod(&stRtspHdrParam);
        if(iRet != 0)
        {
            printf("RTSP_HandleMethod failed\n");
            goto out;
        }

        printf("---------------S->C--------------\n");
        printf("%s\n",stRtspHdrParam.rtsp_OutBuffer);
        send(clientSockfd, stRtspHdrParam.rtsp_OutBuffer, strlen(stRtspHdrParam.rtsp_OutBuffer), 0);

        /* 开始播放，发送RTP包 */
        if(!strcmp(stRtspHdrParam.rtsp_method,RTSP_METHOD_PLAY))
        {
            int frameSize, startCode;
            char* frame = malloc(500000);
            struct RtpPacket* rtpPacket = (struct RtpPacket*)malloc(500000);
            int fd = open(H264_FILE_NAME, O_RDONLY);
            if(fd < 0)
            {
                printf("fd = %d\n",fd);
                goto out;
            }
            rtpHeaderInit(rtpPacket, 0, 0, 0, RTP_VESION, RTP_PAYLOAD_TYPE_H264, 0,
                          0, 0, 0x88923423);

            printf("start play\n");
            printf("client ip:%s\n", clientIP);
            printf("client port:%d\n", stRtspHdrParam.rtsp_clientRtpPort);

            while(1)
            {
      frameSize = getFrameFromH264File(fd, frame, 500000);
                if(frameSize < 0)
                {
                    printf("read err\n");
                    continue;
                }

                if(startCode4(frame))
          startCode = 4;

      frameSize -= startCode;
```

```
                rtpSendH264Frame(serverRtpSockfd, CLIENT_IP, stRtspHdrParam.rtsp_clientRtpPort,
            rtpPacket, frame+startCode, frameSize);
    rtpPacket->rtpHeader.timestamp += 90000/25;

                usleep(1000*1000/25);
            }
            free(frame);
            free(rtpPacket);
            goto out;
        }

    }

out:
    close(clientSockfd);

}

int main(int argc, char* argv[])
{
    int serverSockfd;
    int serverRtpSockfd, serverRtcpSockfd;
    int ret;

    /* 1. 创建TCP  socket */
    serverSockfd = createTcpSocket();
    if(serverSockfd < 0)
    {
        printf("failed to create tcp socket\n");
        return -1;
    }

    /* 2. 绑定TCP  socket */
 ret = bindSocketAddr(serverSockfd, "0.0.0.0", SERVER_PORT);
    if(ret < 0)
    {
        printf("failed to bind addr\n");
        return -1;
    }

    /* 3. 监听 */
    ret = listen(serverSockfd, 10);
    if(ret < 0)
    {
        printf("failed to listen\n");
        return -1;
    }


    /* 4. 创建UDP socket */
    serverRtpSockfd = createUdpSocket();
    serverRtcpSockfd = createUdpSocket();
    if(serverRtpSockfd < 0 || serverRtcpSockfd < 0)
    {
        printf("failed to create udp socket\n");
        return -1;
    }

    /* 5. 绑定UDP  socket */
    if(bindSocketAddr(serverRtpSockfd, "0.0.0.0", SERVER_RTP_PORT) < 0 ||
        bindSocketAddr(serverRtcpSockfd, "0.0.0.0", SERVER_RTCP_PORT) < 0)
    {
        printf("failed to bind addr\n");
        return -1;
    }

    while(1)
    {
        int clientSockfd;
        char clientIp[40];
        int clientPort;

 clientSockfd = acceptClient(serverSockfd, clientIp, &clientPort);
        if(clientSockfd < 0)
        {
            printf("failed to accept client\n");
            return -1;
        }

        printf("accept client;client ip:%s,client port:%d\n", clientIp, clientPort);

        doClient(clientSockfd, clientIp, clientPort, serverRtpSockfd, serverRtcpSockfd);

    }

}
```

RTSP协议的一些分析（四）——RTP传输H.264 yangguoyu8023的博客 1402
文章目录 一、RTP封装 1.1 RTP数据结构 1.2 RTP包的结构以及发包函数 1.3 源码 二、H.264的RTP打包 2.1H.264格式以及H.264的RTP打包方式 2.2 H…

H.264NLU和RTSP协议理解 weixin_40673765的博客 851
h.264分为两部分组成：VCL 和 NAL VCL： 是指如何进行视频压缩算法的纯视频流的部分，这里不做说明（这里是大公司给完成的部分） NAL：是指网络…

写一段c++ 使用ffmpeg库，解析rtsp流并转码到H264 weixin_42579969的博客 280
以下是一段使用ffmpeg库解析rtsp流并将其转码为H264的C代码： ``` #include <libavcodec/avcodec.h> #include <libavformat/avformat.h> #include <libs…

RTSP服务器：RTP打包传输H264码流 weixin_43796767的博客 1522
H264码流与RTP协议分析 1）H264码流由一个个NALU组成，每个NALU前面都有起始码（00 00 00 01或00 00 01），充当分隔符作用。每个NALU有1字…

RTSP协议服务端代码，H264视频流直播 05-26
RTSP协议服务端代码，H264视频流播放，非阻塞多线程，可运行

h264裸流，可用于rtmp和rtsp推流测试 10-21
h264裸流，可用于rtmp和rtsp推流测试，分辨率800*,608，绝对可用

海康摄像头解码rtsp以及h.264为yuv的代码 09-10
主要用于海康摄像头的rtsp协议解码成h.264的视频，再进行硬解码成可用的nv12流，传下去用作人脸识别

live.rar_H.264_h.264 rtsp_live_live555 h.264_rtsp 07-13
H.264 RTSP 串流 (live 555) 视窗版本 (THE Makefile had modified for VC 2008 BUILD)

rtsp传输h264和h265视频推流c代码 01-09
rtsp传输h264和h265视频推流c代码，亲测可用 修改成h265发送：需要修改这里，其他底层的都改好了 （1）strcat(pDescr,"H264/90000");//strcat(pDesc…

java实现传输H.264的RTSP服务 jqh的博客 1379
java实现RTSP服务：H264文件码流，封装成Rtp包。

RTSP视频推流方法汇总 7961
RTSP推流，opencv，rtsp-simple-server，ffmpeg，gstreamer，live555

4、传输H.264的RTSP服务器 huabiaochen的博客 666
本篇文章的目的：这篇文章的目的是实现一个播放H.264的RTSP服务器，相信如果是仔细看了前面几篇文章的话，这简直是易如反掌，就是把前面的东西…

FFmpeg javaUDP 输出H264/265的码流 (传输协议RTP RTSP ONVIF) 03-27
FFmpeg是一个非常流行的音视频处理库，它支持各种音视频格式的编解码、转码、截取、合并等操作。其中，输出H.264/H.265的码流可以通过以下命令…

“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