JT同学
码龄4年 暂无认证
105 原创 | 2万+ 周排名 | 91万+ 总排名 | 46万+ 访问 | 等级
5440 积分 | 1861 粉丝 | 1016 获赞 | 346 评论 | 3945 收藏
私信 关注

搜博主文章

热门文章
从零开始写一个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-RtspServer 25554
从零开始写一个RTSP服务器（二）RTSP协议的实现 21860

最新评论
我的开源项目-RtspServer
kaifa3k: 我测试 ./h264_rtsp_server test.h264 也会这样的问题？请问解决了吗？m...
使用mp4v2封装H.264成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成mp4最简单示例
ai604233436: h264的标准中，好像默认的设置的90KHZ。我所以我理解timeScale...
使用mp4v2封装H.264成mp4最简单示例
ai604233436: 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...

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（五）RTP传输AAC

原创 JT同学 于 2019-08-11 16:12:59 发布 8058 收藏 37 版权
分类专栏： 从零开始写一个RTSP服务器 文章标签： rtsp 音视频 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-RtspServer

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 RTSP/TCP的RTSP服务器

从零开始写一个RTSP服务器（五）RTP传输AAC

文章目录

本文实现目标：使用vlc打开sdp文件可以听到音频

一、RTP封装

这一部分在前面的文章已经介绍过，放到这里只是怕你没有看前面的文章

1.1 RTP数据结构

RTP包格式前面已经比较详细的介绍过，参考从零开始写一个RTSP服务器（一）不一样的RTSP协议讲解

看一张RTP头的格式图回忆一下

V	P	X	CC	M	PT	序列号
时戳						
同步信源(SSRC)标识符						
特约信源(CSRC)标识符						
...						

RTP报头格式

每个RTP包都包含这样一个RTP头部和RTP载荷，为了方便，我将这个头部封装成一个结构体，还有发送包封装成一个函数，下面来看一看

- **RTP头结构体**

```
/*
 * 作者: _JT_
 * 博客: https://blog.csdn.net/weixin_42462202
 */

 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;
 };
```

其中的 `:n` 是一种位表示法，这个结构体跟RTP的头部一一对应

RTP的发包函数

RTP包

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

这是我封装的一个RTP包，包含一个RTP头部和RTP载荷，`uint8_t payload[0]` 并不占用空间，它表示rtp头部接下来紧跟着的地址

RTP的发包函数

```
/*
 * 函数功能: 发送RTP包
 * 参数 socket: 表示本机的udp套接字
 * 参数 ip: 表示目的ip地址
 * 参数 port: 表示目的的端口号
 * 参数 rtpPacket: 表示rtp包
 * 参数 dataSize: 表示rtp包中载荷的大小
 * 返回值: 发送字节数
 */
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;
}
```

目录
从零开始写一个RTSP服务器（五）RTP...
文章目录
一、RTP封装
1.1 RTP数据结构
1.2 源码
二、AAC的RTP打包
2.1 AAC格式
2.2 AAC的RTP打包方式
2.3 AAC RTP包的时间戳计算
2.4 源码
三、AAC的sdp媒体描述

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
从零开始写一个RTSP服... 10篇
live555源码分析与应用 9篇
Linux驱动 19篇

仔细看这个函数你应该可以看懂

我们设置好一个包之后，就会调用这个函数发送指定目标

这个函数中多处使用`htons`等函数，是因为RTP是采用网络字节序（大端模式），所以要将主机字节字节序转换为网络字节序

下面给出源码，`rtp.h`和`rtp.c`，这两个文件在后面讲经常使用

1.2 源码

rtp.h

```c
/*
 * 作者: _JT_
 * 博客: https://blog.csdn.net/weixin_42462202
 */

#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_H_
```

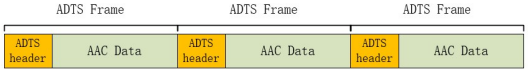
rtp.c

```c
/*
 * 作者: _JT_
 * 博客: https://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;
}
```

二、AAC的RTP打包

2.1 AAC格式

AAC音频文件有一帧一帧的ADTS帧组成，每个ADTS帧包含ADTS头部和AAC数据，如下所示



ADTS头部的大小通常为`7个字节`，包含着这一帧数据的信息，内容如下

Syntax	No. of bits	Mnemonic
adts_fixed_header()		
{		
syncword;	12	bslbf
ID;	1	bslbf
layer;	2	uimsbf
protection_absent;	1	bslbf
profile;	2	uimsbf
sampling_frequency_index;	4	uimsbf
private_bit;	1	bslbf
channel_configuration;	3	uimsbf
original_copy;	1	bslbf
home;	1	bslbf
}		

Syntax	No. of bits	Mnemonic
adts_variable_header()		
{		
copyright_identification_bit;	1	bslbf
copyright_identification_start;	1	bslbf

aac_frame_length;	13	bslbf
adts_buffer_fullness;	11	bslbf
number_of_raw_data_blocks_in_frame;	2	uimsfb
}		

各字段的意思如下

syncword

总是0xFFF, 代表一个ADTS帧的开始, 用于同步.

ID

MPEG Version: 0 for MPEG-4，1 for MPEG-2

Layer

always: '00'

protection_absent

Warning, set to 1 if there is no CRC and 0 if there is CRC

profile

表示使用哪个级别的AAC，如01 Low Complexity(LC) – AAC LC

sampling_frequency_index

采样率的下标

samplingFrequencyIndex	Value
0x0	96000
0x1	88200
0x2	64000
0x3	48000
0x4	44100
0x5	32000
0x6	24000
0x7	22050
0x8	16000
0x9	12000
0xa	11025
0xb	8000
0xc	7350
0xd	reserved
0xe	reserved
0xf	escape value

aac_frame_length

一个ADTS帧的长度包括ADTS头和AAC原始流

adts_buffer_fullness

0x7FF 说明是码率可变的码流

number_of_raw_data_blocks_in_frame

表示ADTS帧中有number_of_raw_data_blocks_in_frame + 1个AAC原始帧

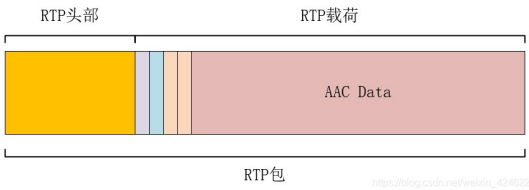
这里主要记住ADTS头部通常为7个字节，并且头部包含`aac_frame_length`，表示ADTS帧的大小

2.2 AAC的RTP打包方式

AAC的RTP打包方式并没有向H.264那样丰富，我知道的只有一种方式，原因主要是AAC一帧数据大小都是几百个字节，不会向H.264那么少则几个字节，多则几千

AAC的RTP打包方式就是将ADTS帧取出ADTS头部，取出AAC数据，每帧数据封装成一个RTP包

需要注意的是，并不是将AAC数据直接拷贝到RTP的载荷中。AAC封装成RTP包，在RTP载荷中的前四个字节是有特殊含义的，然后再是AAC数据，如下图所示



其中RTP载荷的一个字节为0x00，第二个字节为0x10

第三个字节和第四个字节保存AAC Data的大小，最多只能保存13bit，第三个字节保存数据大小的高八位，第四个字节的高5位保存数据大小的低5位

2.3 AAC RTP包的时间戳计算

假设音频的采样率位44100，即每秒钟采样44100次

AAC一般将1024次采样编码成一帧，所以一秒就有44100/1024=43帧

RTP包发送的每一帧数据的时间增量为44100/43=1025

每一帧数据的时间间隔为1000/43=23ms

2.4 源码

下面给出rtp发送aac文件的源码，该程序从aac文件中提取每一帧的AAC数据，然后RTP打包发送到目的

如何获取AAC Data?
这个示例是先读取7字节的ADTS头部，然后获得该帧大小，进而读取出AAC Data

rtp_aac.c

```c
/*
 * 作者: _JT_
 * 博客: https://blog.csdn.net/weixin_42462202
 */

#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <string.h>

#include "rtp.h"

#define AAC_FILE    "test.aac"
#define CLIENT_PORT 9832

struct AdtsHeader
{
    unsigned int syncword;  //12 bit 同步字 '1111 1111 1111'，说明一个ADTS帧的开始
    unsigned int id;        //1 bit MPEG 标示符， 0 for MPEG-4, 1 for MPEG-2
    unsigned int layer;     //2 bit 总是'00'
    unsigned int protectionAbsent;  //1 bit 1表示没有crc，0表示有crc
    unsigned int profile;           //1 bit 表示使用哪个级别的AAC
    unsigned int samplingFreqIndex; //4 bit 表示使用的采样频率
    unsigned int privateBit;        //1 bit
    unsigned int channelCfg; //3 bit 表示声道数
    unsigned int originalCopy;         //1 bit
    unsigned int home;                  //1 bit

    /*下面的为改变的参数即每一帧都不同*/
    unsigned int copyrightIdentificationBit;   //1 bit
    unsigned int copyrightIdentificationStart; //1 bit
    unsigned int aacFrameLength;               //13 bit 一个ADTS帧的长度包括ADTS头和AAC原始流
    unsigned int adtsBufferFullness;           //11 bit 0x7FF 说明是码率可变的码流

    /* number_of_raw_data_blocks_in_frame
     * 表示ADTS帧中有number_of_raw_data_blocks_in_frame + 1个AAC原始帧
     * 所以说number_of_raw_data_blocks_in_frame == 0
     * 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)
     */
    unsigned int numberOfRawDataBlockInFrame; //2 bit
};

static int parseAdtsHeader(uint8_t* in, struct AdtsHeader* res)
{
    static int frame_number = 0;
    memset(res,0,sizeof(*res));

    if ((in[0] == 0xFF)&&((in[1] & 0xF0) == 0xF0))
    {
```

```
        res->id = ((unsigned int) in[1] & 0x08) >> 3;
        printf("adts:id  %d\n", res->id);
        res->layer = ((unsigned int) in[1] & 0x06) >> 1;
        printf( "adts:layer  %d\n", res->layer);
        res->protectionAbsent = (unsigned int) in[1] & 0x01;
        printf( "adts:protection_absent  %d\n", res->protectionAbsent);
        res->profile = ((unsigned int) in[2] & 0xc0) >> 6;
        printf( "adts:profile  %d\n", res->profile);
        res->samplingFreqIndex = ((unsigned int) in[2] & 0x3c) >> 2;
        printf( "adts:sf_index  %d\n", res->samplingFreqIndex);
        res->privateBit = ((unsigned int) in[2] & 0x02) >> 1;
        printf( "adts:pritvate_bit  %d\n", res->privateBit);
        res->channelCfg = ((((unsigned int) in[2] & 0x01) << 2) | (((unsigned int) in[3] & 0xc0) >> 6));
        printf( "adts:channel_configuration  %d\n", res->channelCfg);
        res->originalCopy = ((unsigned int) in[3] & 0x20) >> 5;
        printf( "adts:original  %d\n", res->originalCopy);
        res->home = ((unsigned int) in[3] & 0x10) >> 4;
        printf( "adts:home  %d\n", res->home);
        res->copyrightIdentificationBit = ((unsigned int) in[3] & 0x08) >> 3;
        printf( "adts:copyright_identification_bit  %d\n", res->copyrightIdentificationBit);
        res->copyrightIdentificationStart = (unsigned int) in[3] & 0x04 >> 2;
        printf( "adts:copyright_identification_start  %d\n", res->copyrightIdentificationStart);
        res->aacFrameLength = (((((unsigned int) in[3]) & 0x03) << 11) |
                               (((unsigned int)in[4] & 0xFF) << 3) |
                                   ((unsigned int)in[5] & 0xE0) >> 5) ;
        printf( "adts:aac_frame_length  %d\n", res->aacFrameLength);
        res->adtsBufferFullness = (((unsigned int) in[5] & 0x1f) << 6 |
                                   ((unsigned int) in[6] & 0xfc) >> 2);
        printf( "adts:adts_buffer_fullness  %d\n", res->adtsBufferFullness);
        res->numberOfRawDataBlockInFrame = ((unsigned int) in[6] & 0x03);
        printf( "adts:no_raw_data_blocks_in_frame  %d\n", res->numberOfRawDataBlockInFrame);

        return 0;
    }
    else
    {
        printf("failed to parse adts header\n");
        return -1;
    }
}

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

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

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

    return fd;
}

static int rtpSendAACFrame(int socket, char* ip, int16_t port,
                           struct RtpPacket* rtpPacket, uint8_t* frame, uint32_t frameSize)
{
    int ret;

    rtpPacket->payload[0] = 0x00;
    rtpPacket->payload[1] = 0x10;
    rtpPacket->payload[2] = (frameSize & 0x1FE0) >> 5; //高8位
    rtpPacket->payload[3] = (frameSize & 0x1F) << 3; //低5位

    memcpy(rtpPacket->payload+4, frame, frameSize);

    ret = rtpSendPacket(socket, ip, port, rtpPacket, frameSize+4);
    if(ret < 0)
    {
        printf("failed to send rtp packet\n");
        return -1;
    }

    rtpPacket->rtpHeader.seq++;

    /*
     * 如果采样频率是44100
     * 一般AAC每个1024个采样为一帧
     * 所以一秒就有 44100 / 1024 = 43帧
     * 时间增量就是 44100 / 43 = 1025
     * 一帧的时间为 1 / 43 = 23ms
     */
    rtpPacket->rtpHeader.timestamp += 1025;

    return 0;
}

int main(int argc, char* argv[])
{
    int fd;
    int ret;
    int socket;
    uint8_t* frame;
    struct AdtsHeader adtsHeader;
    struct RtpPacket* rtpPacket;

    if(argc != 2)
    {
        printf("Usage: %s <dest ip>\n", argv[0]);
        return -1;
    }

    fd = open(AAC_FILE, O_RDONLY);
    if(fd < 0)
    {
        printf("failed to open %s\n", AAC_FILE);
        return -1;
    }

    socket = createUdpSocket();
    if(socket < 0)
    {
        printf("failed to create udp socket\n");
        return -1;
    }

    frame = (uint8_t*)malloc(5000);
    rtpPacket = malloc(5000);

    rtpHeaderInit(rtpPacket, 0, 0, 0, RTP_VESION, RTP_PAYLOAD_TYPE_AAC, 1, 0, 0, 0x32411);

    while(1)
    {
        printf("--------------------------------\n");

        ret = read(fd, frame, 7);
        if(ret <= 0)
        {
            lseek(fd, 0, SEEK_SET);
            continue;
        }

        if(parseAdtsHeader(frame, &adtsHeader) < 0)
        {
            printf("parse err\n");
            break;
        }

        ret = read(fd, frame, adtsHeader.aacFrameLength-7);
        if(ret < 0)
        {
            printf("read err\n");
            break;
        }

        rtpSendAACFrame(socket, argv[1], CLIENT_PORT,
                        rtpPacket, frame, adtsHeader.aacFrameLength-7);

        usleep(23000);
    }

    close(fd);
    close(socket);

    free(frame);
    free(rtpPacket);

    return 0;
}
```

三、AAC的sdp媒体描述

下面给出AAC的媒体描述信息

```
m=audio 9832 RTP/AVP 97
a=rtpmap:97 mpeg4-generic/44100/2
a=fmtp:97 SizeLength=13;
c=IN IP4 127.0.0.1
```

这个一个媒体级的sdp描述，关于sdp文件描述详情可看从零开始写一个RTSP服务器（一）不一样的RTSP协议讲解

**m=audio 9832 RTP/AVP 97 **

格式为 m=<媒体类型> <端口号> <传输协议> <媒体格式 >
媒体类型：audio，表示这是一个音频流

端口号：9832，**表示UDP发送的目的端口为9832**

传输协议：RTP/AVP，表示RTP OVER UDP，通过UDP发送RTP包

媒体格式：表示负载类型(payload type)，一般使用97表示AAC

a=rtpmap:97 mpeg4-generic/44100/2

格式为a=rtpmap:<媒体格式><编码格式>/<时钟频率> /[channel]

mpeg4-generic表示编码，44100表示时钟频率，2表示双通道

c=IN IP4 127.0.0.1

IN：表示internet

IP4：表示IPV4

127.0.0.1：**表示UDP发送的目的地址为127.0.0.1**

特别注意：这段sdp文件描述的udp发送的目的IP为127.0.0.1，目的端口为9832

四、测试

将上面给出的源码 `rtp.c` 、 `rtp.h` 、 `rtp_h264.c` 保存下来，sdp文件保存为 `rtp_aac.sdp`

注意：该程序默认打开的是 `test.aac` ，如果你没有音频源，可以从RtspServer的example目录下获取

编译运行

```
# gcc rtp.c rtp_aac.c
# ./a.out 127.0.0.1
```

这里的ip地址必须跟sdp里描述的目标地址一致

使用vlc打开sdp文件

```
# vlc rtp_aac.sdp
```

到这里就可以听到音频了，下一篇文章讲解如何写一个发送AAC的RTSP服务器