# “FSing”音乐平台的设计与实现

## ——流媒体子系统

计算机与信息科学学院　软件工程专业　2015 级　闵娴有

指导老师　龚伟

**摘 要：**“FSing”音乐是一款在线音乐平台，参考已有的产品“网易云音乐”，提供在线音乐播放、歌曲评论、推荐歌单等基本功能。该流媒体子系统采用 C/S（Client/Server）架构，FFmpeg 解码、Jrtplib 传输、多线程编程、Socket 网络编程等技术，以及 C++编程语言，对子系统需求和功能进行了设计与实现。本设计文档阐述了该音乐平台中流媒体子系统的详细设计，包含音频在线播放、FFmpeg 解码、RTSP 命令传输、Jrtplib 音频流传输等技术的实现。

**关键字：**流媒体; RTSP; Jrtplib; FFmpeg;多线程

**Abstract:** The “FSing” music is an online music platform. It provides features such as online playing, song commenting and recommended songlist referring the existing product “netease cloud music”. The streaming media subsystem adopts C/S(Client/Server) architecture and uses FFmpeg decoding, Jrtplibtr ansmission, multi-threaded programming, Socket network programming and other technologies and C++ programming language to design and implement the subsystem’s requirements and functions. This design document describes the detailed design of the streaming media subsystem in the “FSing” music platform , including the realization of audio online play, FFmpeg decoding，RTSP command transfers, Jrtplib audio stream transmission and other technologies.

**Key words:** stream media; RTSP; Jrtplib; FFmpeg; multi-threaded

# 绪论

当今时代，在线音乐平台已经成为了人们欣赏音乐、社会交往的虚拟空间。故该项目旨在设计一款集在线听歌、歌曲评论、歌单推荐于一体的在线音乐平台软件。本文档主要阐述了流媒体子系统的详细设计与实现,描述了流媒体子系统的关键技术、分析模型以及关键功能的代码实现。

# 1 架构设计

## 1.1 系统架构

本音乐平台系统采用 C/S 架构，主要设计两个服务器：RTSP Media Server（流媒体服务器）用于实现流媒体在线播放，FSing Server（数据服务器）用于管理客户端数据和处理客户端的数据请求。如图 1-1 为 FSing 音乐平台的架构图。

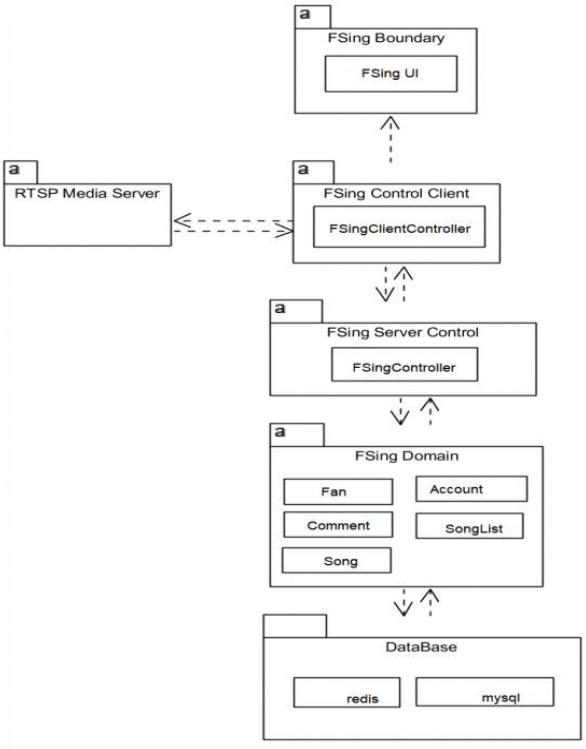


**图 1-1 系统总架构图**

本文档主要描述流媒体子系统的实现与设计，即 RTSP Media Server 以及与客户端的交互。本流媒体子系统提供对于客户端 RTSP 命令的接受与处理，以及

媒体流的传输，重点在于设计流媒体服务器与客户端之间的交互。如下图 1-2 为子系统的系统架构图，主要分为服务器层和客户端层，在个层中，有三个用于交互的类，RTSPSessioin 用于 RTSP 会话的交互，RTPSession 用于 RTP 数据传输交互，CDecodeSrc 用于解码音频文件，这三个类的详细设计，如下第 3 部分：详细设计。另外流媒体子系统主要存储在线音频文件，对于音频的详细信息，如：歌曲 id、专辑、时长、歌手等，都存储于本项目的另一个数据服务器中。

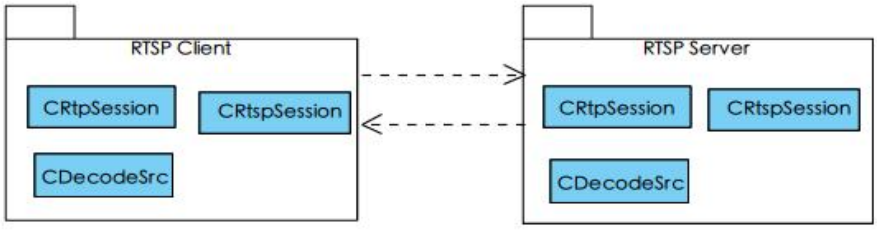


**图 1–2　流媒体子系统架构图**

### 1.2　流媒体协议的选择——RTSP 协议

常见的流媒体协议有：RTSP 协议、RTP 协议、RTCP 协议、RTMP 协议、MMS 协议、HLS 协议等等，在本流媒体子系统中选用 RTSP+RTP+RTCP。选择 RTSP 协议的主要原因在于，它具有双向传输、可自定义增加方法和参数等特性。它在体系结构上位于 RTP 和 RTCP 之上，通常使用 TCP 或者 UDP 完成媒体数据传输，在实现实时传输时，一般采用 UDP+RTP+组播实现。[1]在本项目中使用的到 Jrtplib 库已经将 RTP 和 RTCP 协议封装实现。因此在本流媒体子系统中主在实现了 RTSP 实时流协议。

### 1.3　流媒体编解码技术选择——FFmpeg 编解码

常见的音频编解码技术有：FFmpeg、VLC。FFmpeg 是一个用于音视频编解码、格式转换等功能的跨平台开源库，而 VLC 是一个多媒体播放器，它支持众多音视频的解码器和档案格式，在使用时直接调用接口即可。对于本流媒体子系统，重在自主设计实现音频的解码和在线播放传输，所以选用 FFmpeg 库进行实现。音频解码的主要作用是将音频码流解码得到音频采样数据（PCM 等）。音频的编码格式有 mp3、AAC 等，本流媒体子系统中主要实现了在线音频 mp3 数据的解码。

本流媒体子系统使用 FFmpeg 解码和 SDL 库转换播放。在使用 FFmpeg 解码过程中主要需要用到三个库：

（1）libavcodec/avcodec.h 库，提供视音频编解码相关的接口函数和数据结构，包括注册所有的容器格式和编解码器、解码上下文结构 AVCodecContext、解码器结构 AVCodec 和存放音频码流的数据结构 AVPacket 等。[2]

（2）libavformat/avformat 库，提供解析封装格式相关函数和数据结构，包括存储媒体文件信息的结构 AVFormatContext 和读取音频帧函数等。[2]

（3）Libswresample/swresample 库，提供音频的采样数据格式转换相关函数，包含存放音频转换参数的结构体 SwrContext 等。[2]

### 14 流媒体传输选择——Jrtplib 传输

关于实时流媒体传输的开源库中，主要有两个：live555 和 Jrtplib。Live555 将 RTP、RTSP、RTCP 基于一身，是一个功能齐全的，强大的集合体，但是对于本项目规模、时间以及库本身的学习理解难度方面来讲，live555 过于复杂和庞大。而 Jrtplib 是一个基于 C++、面向对象的 RTP 封装库，同时也提供 RTCP 的传输，但是它没有提供 RTSP 的实现，因此该库相对 live555 来说，较为容易理解和轻便。因此在本子系统中使用 Jrtplib 实现媒体流的传输。具体传输流程，在代码实现中，详细介绍。[3]

### 15 定义并发任务

流媒体技术是一种专门用于网络多媒体信息传播和处理的技术，该技术能够在网络上实现传输和播放同时进行的实时工作模式，它常规视音频媒体之间的不同在于，流媒体可以边下载边播放。[4]因此，在本流媒体子系统中需要使用到多线程技术，将解码和传输、接收和播放分别置于不同的线程中，多个工作相互独立，从而实现边下载边播放，提高了音频解码的处理，减少了播放的延时。

### 16 缓冲队列

当使用两个线程独立处理数据接收和解码时，如果解码线程中处理一个帧的时间较长，这时处理速度跟不上接收速度，造成 UDP 缓冲区满，而被刷新，就

会造成丢包，因此在具体实现中，使用了数据缓冲队列，缓存中存放的是某个时间段内接收到的媒体流数据，并且解码线程不断从缓存中读取数据后，便会将该数据从缓冲队列中清除，因此，缓冲队列中的数据也是动态的，这样在有新的媒体流数据进入队列时，大大减少了因缓冲区满，而造成丢包的可能性。[5]

# 2 流媒体子系统详细设计

## 2.1 抽象类设计

在流媒体子系统具体实现过程中，采用的 socket 网络编程技术和多线程技术，使用现有的 pthread 线程库和 sys/socket 库，主要体现在应用类 CSock 和 CThread，分别作为客户端-服务器的套接字基类和子系统线程的管理类。同时，对于所有 CRtspSession 会话的管理，使用 CRtspSrv 类。

### 2.1.1 套接字基类

CSock 套接字基类，提供套接字基础的 send()发送消息，recv()接收消息、bind()绑定连接端 IP 和端口、open()打开 Socket 函数。该类主要用于 RTSP 会话的建立以及消息的接收和发送。如图 2-1 为 CSock 类图。

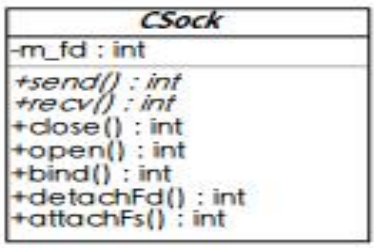


图 2-1 CSock 类图

### 2.1.2 线程管理类

CThread 用于管理子系统所有线程，在流媒体子系统中所有需要创建新线程来执行操作的类，都继承于 CThread 线程类，该类提供了一个静态的线程函数 thread_fun()函数，作为 pthread_create()函数创建线程时需要的静态函数指针参

数。然后各个类重载 CThread 类的 thread_pro()线程执行函数，每个线程在 thread_fun()函数中，动态绑定，调用相应的线程执行函数完成相应的处理。如图 2-2 为 CThread 类图。

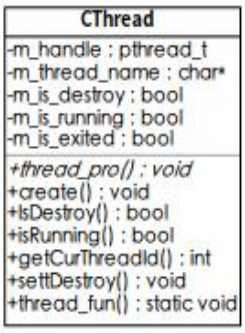


图 2-2　CThread 类图

## 2.2　边界设计

### 2.2.1　RTSP 协议设计

RTSP 消息基本格式为(客户端请求消息格式）：方法+URL+版本+回车换行（CRLF)+CSeq:序列号。如下对参数进行详细介绍：

（1）方法：请求服务器执行的操作命令，在本流媒体子系统的，主要实现了 OPTIONS、DESCIBE、SETUP、PAUSE、TEARDOWN 五个命令。

（2）URL：接收方的 RUL，包括使用的协议、接收方 IP 和端口、请求播放的媒体文件名，例：rtsp://192.168.43.32:8554/1.mp3。

（3）版本号：一般为 RTSP/1.0。

（4）序列号：每发送一个消息，序列加 1,发送的某一命令，和该命令返回的响应消息序列号相同，保证了消息的处理顺序。因为子系统媒体流的传输使用 Jrtplib 库，而在 Jrtplib 创建传输会话时，需要提供客户端的端口号，并且该端口号需要是未被占用的，因此在 RTSP 协议实现设计中，新增加了一个命令 RTSP_CLIENT_PORT,用于告知服务端客户端用于接收媒体流数据的端口号。

RTSP 服务器相应消息基本格式：一般为处理结果+序列号。处理结果：一般成功为 200 OK,若失败，可自定义相关错误码，如：404,未找到音频文件。序列号：请求消息的序列号。

流媒体子系统中,所有的音频文件命名与音频文件在数据服务器上存储的 ID 相同,因为在海量的歌源中，名字相同的歌曲不计其数，唯一不同的则是数据库中歌曲存储的 ID。在此，以 PLAY 命令为例，做 RTSP 命令的详细介绍：

客户端发送 PLAY 命令，指定播放 1.mp3 文件。

PLAY rtsp://192.168.43.32:8554/1.mp3 RTSP/1.0

CSeq: 3

Session: 12345678

服务器接收处理成功，返回 200 OK。

RTSP/1.0 200 OK

CSeq: 3

Session: 12345678

RTP-Info: url=rtsp://192.168.43.32:8554/1.mp3[6]

### 2.2.2 RTSP 会话管理类

流媒体服务器从程序启动到结束，只存在一个 CRtspSrv 类对象，该对象管理整个子系统所有的 RTSP 会话。如图 2-3 为 CRtspSrv 类图,通过将 CRtspSession 类和 CRtspSrv 类设计成一对多的组合关系,将 CRtspSession 对象指定同 CRtspSrv 有着相同的生命周期，从而实现 CRtspSrv 对 RTSP 会话的管理。

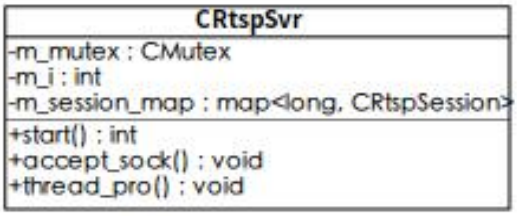


图 2-3 CRtspSrv 类图

### 2.2.3 会话类交互设计

在流媒体子系统中主要设计了 2 个会话类,用于客户端-服务器的 RTSP、RTP 会话的连接和消息处理。如图 2-4 所示为 RTSPSession 类和 RTPSession 类的详细设计类图，以及两个会话类之间的多重性关系。

CRtspSession: RTSP 会话类，继承自 CThread 类。当客户端请求 SETUP 命令

时，创建该对象，并开启 RTSP 会话线程，该类主要用于对 RTSP 命令的接收、解析、处理和传输。

CRtpSession: RTP 会话类，继承自 CThread 类。当接收到客户端 PLAY 命令时，创建该对象，开启 RTP 传输线程。该类主要用于从缓冲区读取并传输流媒体数据。

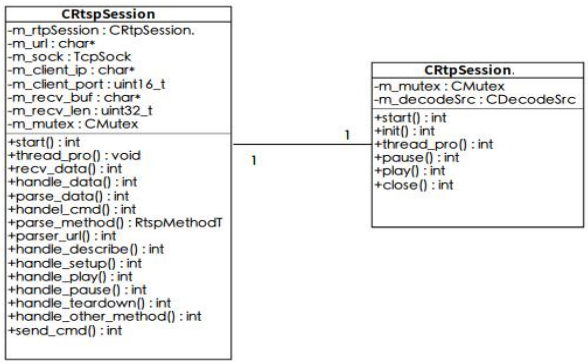


图 2–4　会话类

2.2.4　媒体流管理设计

CDecodeSrc 是一个继承自 CThread 的类。该类提供了一个 packetNode 数据类型，存储媒体流数据，并将每段媒体流数据存放在 std::list 结构中，实现先入先出的缓存队列。该类主要用于解码在线音频文件，并将解码得到的码流放入待传输缓冲区。当收到 DESCRIBE 命令是，创建该对象，开启解码线程，等待传输数据。如图 2-5 为 CDecodecSrc 的类图设计。

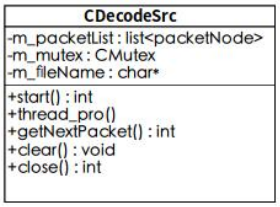


图 2–5　CDecodecSrc 类图

2.2.5　RTSP 交互过程

当请求播放在线音乐时，首先客户端会像流媒体服务器发送一个 DESCRIBE 命令，服务器收到后，返回一个 SDP 描述信息，包含音频流数量、媒体类型等信息。然后客户端解析 SDP 描述，并向服务器发送 SETUP 命令，请求建立流媒体会话，若服务器与客户端建立连接成功。接着客户端发送 PLAY 命令，告知服

务器开始传输媒体流。交互过程如图 2-6。

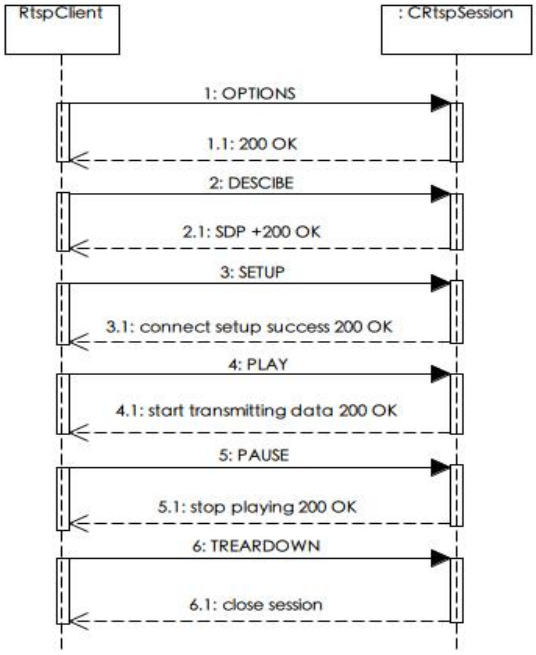


图 2-6　RTSP 协议交互图

## 2.3　服务端交互设计

（1）启动服务器：创建 CRtspSrv 对象，开启监听线程，监听客户端的连。

（2）当监听到新连接时，CRtspSrv 对象创建一个新的 CRtspSession 对象，开启 RTSP 会话线程。

（3）CRtspSession 线程轮询检测是否有客户端的请求消息，并解析消息类型，将其交给不同的处理函数处理，并将处理结果返回给客户端。

（4）当服务器收到 SETUP 命令时，创建解码线程，准备数据，等待传输。

（5）当服务器收到 PLAY 命令时，创建 RTP 会话线程，传输流媒体数据。直到传输完成或者客户端主动请求 TREADOWN 终止 RTSP 会话。

## 2.4　客户端交互设计

（1）当用户点击播放在线音频时，读取本地配置文件，获取服务器的 IP 和端口，向流媒体发送一系列 RTSP 请求消息。

（2）创建接收和解码两个线程，同时进行接收和播放。

（3）接收 UI 界面传来的暂停或切换歌曲操作，向服务器发送相应请求。

## 2.5　交互模型

交互模型描述了对象之间的交互,各个对象如何协作完成整个系统的行为。[7]

### 2.5.1 解码传输顺序图

当收到客户端传入的 PLAY 命令时，调用 CRtpSession 对象的 start()函数，开启 RTP 数据传输线程，并通过 GetNextPacket()函数不断从缓冲区读取。当传输完成时，自动调用 CRtspSrv 的 notify_fun（ ）函数通知会话管理对象该 RTSP 会话已经结束，请销毁。如图 2-7 为解码传输顺序图。

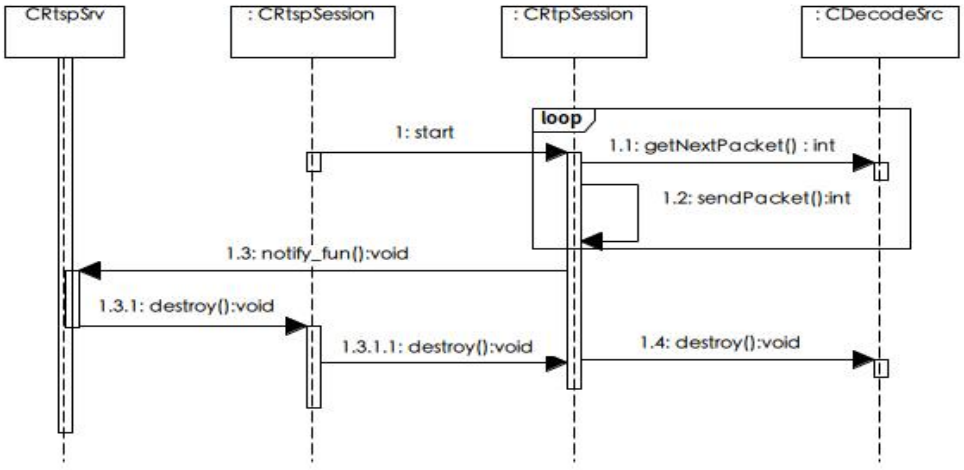


图 2–7　解码传输顺序图

### 2.5.2 解码播放顺序图

客户端用户点击播放在线音乐，创建流媒体会话，接收数据，放入缓冲区，然后解码线程从缓冲区中获取数据，并播放。如图 2–8 为解码播放顺序图。

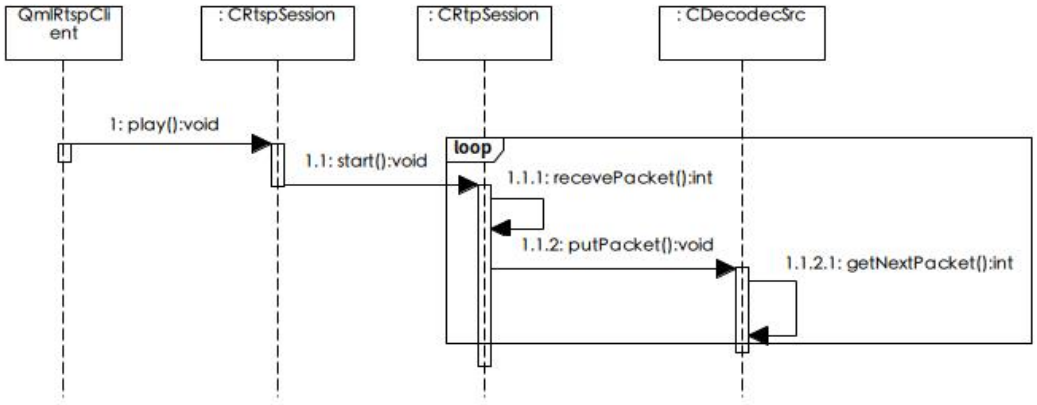


图 2–8　解码播放顺序图

# 3　流媒体子系统实现

## 3.1 关键实现代码

### 3.1.1 Jrtplib 传输数据

在使用 Jrtplib 进行媒体流传输时流程，如图 3-1 代码所示:

（1）首先需要创建 RTPSession 对象来表示 RTP 会话。

（2）然后再调用 Create()函数对会话进行初始化，但是，在 Create 之前，还需要创建两个对象作为会话初始化的参数，一个是 RTPSessionParams 对象，用于设置会话的时间戳等重要的参数，另一个是 RTPUDPv4TransmissionParams 对象，用于设置 RTP 会话的本地端口信息。

（3）RTP 会话建立成功之后，接着就是传输数据，首先需要设置好数据接收的目标 IP 和端口，通过调用 AddDestination()函数实现。

（4）最后调用 SendPacket 方法，向目标地址发送媒体流数据。

```
RTPSession session;

RTPSessionParams sessionparams;
sessionparams.SetOwnTimestampUnit(1.0 / 90000.0);
sessionparams.SetAcceptOwnPackets(true);

RTPUDPv4TransmissionParams transparams;
transparams.SetPortbase(m_server_port); //这个端口必须未被占用

int status = session.Create(sessionparams, &transparams);
if (status < 0) {...}
//    while(1){
//        if(m_is_new_connect == false){}

RTPIPv4Address addr(ntohl(inet_addr(m_client_ip)), m_client_port);
cout << m_client_ip << " " << m_client_port << endl;
status = session.AddDestination(addr);

if (status < 0) {...}

session.SetDefaultPayloadType(96);          //rtp头的payload类型
session.SetDefaultMark(false);              //rtp头MarkerBut的默认值
session.SetDefaultTimestampIncrement(90000.0 / 25.0);   //时间戳增量3600

//        session.SetDefaultMark(true);
//        RTPTime delay(10); {...}
while(1){
    while(IsDestroyed() == false){
        if(m_is_Play == true){
            if((bufLength = m_decodeSrc.getNextPacket(buffer,SEND_BUFFE_SIZE,pts)) > 0 ){
                session.SendPacket(buffer,bufLength);
/                 cout <<"Send paccket  " << bufLength << "bytes" << endl;
                //  {...}
            }
        }
        currentTime = time(nullptr);
        if((currentTime - startTime) >= 0.7){
            RTPTime::Wait(RTPTime(1,0));
            startTime = time(nullptr);
        }
```

图 3-1 Jrtplib 传输

### 3.1.2 FFmpeg 解码

流媒体子系统实现了从内存中读取数据，这跟从本地文件或者 URL 地址中获取数据是有很大差别的，因为在解码时，需要提供一个路径给 avformat_open_input()函数，用于获取媒体数据，而从网络中抓取的数据包是存

在于内存中的，无法提供一个路径。[8]

（1）首先使用 av_gister_all（ ）注册所有的容器格式和解码器。

（2）创建 AVIOContext 对象，管理输入输出数据缓冲。并设置缓冲数据获取的方法：fill_iobuffor()和存放数据的对象 iobuffer 。然后设置 AVFormatContext 的输入输出缓冲区成员 pb 为该 AVIOContext 对象。该结构是实现从内存中读取数据最主要的数据结构。如图为 fill_iobuffer()函数的实现。

（3）探测是否有网络数据，并用获得的网络流，初始化 AVFormatContext 对象。使能从该对象读取解码器类型和获取帧对象。

（4）查找音频流出现的第一个位置

（5）获取解码器上下文 AVCodecContext 和解码器 AVCodec 对象。

（6）打开解码器 avcodec_open2()

（7）打开音响设备 SDL_OpenAudio()。

（8）不断从码流中提取出帧数据，av_read_frame(),判断若是音频流，则传给 swr_convert()将帧数据根据参数进行转换

（9）最后使用 SDL_MixAudio 对音频数据进行混音并播放

（10）重复以上 9-10 步骤。

### 3.1.3 CRtspSrv 管理所有的 RTSP 会话

CRtspSrv 类管理所有的 RTSP 会话,当接收到新连接时，创建一个新的 CRtspSession 对象,并调用该对象的函数 start()创建会话线程,实现代码如图 3-2。

```
void CRtspSvr::accept_sock( int fd )
{
    LogInfo( "Accept a new connect, the fd: %d\n", fd );
    CRtspSession* session = new CRtspSession;
    session->setClientIP(m_client_ip);
    session->setServerPort(8544 + i);     ⚠ implicit conversion loses integer pre
    i += 2;

//    session->setServerPort()
    //    session->setClientPort(m_client_port);

    if( session->Start( fd, notify_fun, (long)this ) >= 0 ){
        m_mutex.Enter();
        m_session_map[fd] = session;
        m_mutex.Leave();
    }else
        delete session;
}
```

**图 3-2　创建 RTSP 会话**

### 3.1.4 监听客户端连接

监听线程轮询检测是否有新的客户端连接，当接收到新连接时，调用动态绑定到的 CRtspSrv 对象的 accept_socket()函数，创建新的 RTSP 会话线程。

### 3.1.5 RTSP 命令的判断和处理

hand_cmd()函数判断命令类型，并调用相应的处理函数。命令类型是定义的一个枚举类型，先通过 parse_method( ）函数，从接收到的消息中提取出命令字段，并返回相应的枚举。代码实现如图 3-3。

```
int CRtspSession::handle_cmd( const char* data, int len )
{
    RtspMethodT method = parse_method( data, len );
    if( method == RTSP_METHOD_MAX ){
        LogError( "unsupported this method\n" );
        return -1;
    }
    parser_common( data, len );
    switch( method ){
    case RTSP_DESCRIBE:
        return handle_describe( data, len );
    case RTSP_SETUP:
        return handle_setup( data, len );
    case RTSP_PLAY:
        return handle_play( data, len );
    case RTSP_PAUSE:
        return handle_pause();
    case RTSP_TEARDOWN:
        return handle_teardowm();
    default:
        return handle_other_method();
    }
    return 0;
}
```

**图 3-3　RTSP 命令处理**

### 3.1.6 通过配置文件获取服务器信息

服务器的 URL 信息存放在本地 config 配置文件中，如果服务器 URL 变更，可以直接更改配置文件，而无需更改代码，快捷方便。实现代码请参考项目源码。

### 3.1.7　缓冲队列

使用自定义的结构类型 PacketNode 存放流媒体数据，如图 3-4 为缓存中数据的结构。使用 List 实现队列的先入先出，接收和解码两个线程之间维护了一个队列，因此使用到了锁机制，实现了一个典型的“生产者-消费者”的模型，同时，使用缓存还能弥补延迟和抖动带来的影响。

```
    struct PacketNode{
//        std::shared_ptr<uint8_t> buf;
        uint8_t *buf;           //buf要先用new分配内存再使用
        int length;    //<=1024
        int64_t pts;
    };
```

**图 3-4　缓存结构**

从缓冲队列不断读取缓存中的数据进行播放，播放完成后缓冲区的该部分数据被立即清除,如图 3-5 为接收线程代码实现。

```
// 丢包 ，数据的处理速度满足不了接收方的接收速度，导致udp缓存满之后，被刷新而丢包
int error_status = session.Poll();
checkerror(error_status);
session.BeginDataAccess();
if (session.GotoFirstSourceWithData())
{
    //            cout <<"_____________________________________________" << endl;
    do
    {
        RTPPacket *pack;

        while ((pack = session.GetNextPacket()) != nullptr)
        {
              cout <<"get Pack" << endl;
            int nLen = pack->GetPayloadLength();    ⚠implicit conversion loses int
            unsigned char *pPayData = pack->GetPayloadData();
            //将包序号写入文件:
            int nSeqNum = pack->GetSequenceNumber();
            write << nSeqNum << endl;
            //                    cout <<"Receive packet,Sceq: " << nSeqNum <<" ,si

            PacketNode temNode;
            temNode.length = nLen;
            temNode.buf = new uint8_t[nLen];
            memcpy(temNode.buf, pPayData, nLen);    ⚠implicit conversion changes s
            m_encodeSrc.put_packetNode(temNode);            //数据存放入缓存区

            session.DeletePacket(pack);
        }
        RTPTime::Wait(RTPTime(1,0));
    } while (session.GotoNextSourceWithData());
}
session.EndDataAccess();
```

**图 3-5　接收线程**

## 4　结束语

经过几个月的努力，流媒体子系统的实现基本上达到了开题报告预期的结果。主要实现了：在线音频的解码、媒体数据的传输、多线程控制边缓存边播放等功能。该项目的主要难点在与 RTSP 协议的实现和媒体流数据的传输，对于媒体流传输，本子系统中使用的已有的 Jrtplib 库进行传输，因此，在接下来的时间中，将进一步研究流媒体的 RTP 传输实现。

**参考文献：**


[1] CSDN 博客，https://blog.csdn.net/u013203733/article/details/73869227.

[2] CSDN 博客，https://blog.csdn.net/leixiaohua1020/article/details/14215755.

[3] 博客园，http://www.cnblogs.com/ansersion/p/5079758.html.

[4] 杨劲. 网络流媒体传输的自适应技术[J]. 中国有线电视, 2003：20-22.

[5] CSDN 博客， https://blog.csdn.net/YU_coder/article/details/79836917.
[6] CSDN 博客，https://blog.csdn.net/caoshangpa/article/details/53191630.
[7] Michael Blah、.James Rumbaugh. UML 面向对象建模与设计（第二版）[M]. 北京：人民邮电出版社，2011：14.
[8] CSDN 博客，https://blog.csdn.net/leixiaohua1020/article/details/12980423.