摘 要

本次综合课程设计主要任务是采用 Qt5.8、opencv 和 FFmpeg 实现音视频采集和音视频推流，FFmepg 开源库把所采集的原始视频数据进行编码成 H.264 压缩格式，采用 ACC 编码方式把所采集的原始音频数据编码，再把编码之后的音视频压缩数据使用 RTMP 协议进行网络传输。使用 Wowza 流媒体服务器实现推流终端的搭建。同时通过构建的的前端网页对实时传输的音视频流进行播放。

关键词：FFmpeg，RTMP，ACC，直播推流

目 录

第一章 复杂工程问题归纳与实施方案可行性研究

需求分析与复杂工程问题归纳

1.1.1 需求分析

网络直播的走红是多方因素共同作用的结果。2014 年，李克强在世界互联网大会上提出“大众创业、万众创新”，之后关于互联网的一系列的政策、措施开始出现，推动了互联网和各个领域的融合，同时也促进了网络新形态的出现，为网络直播的井喷奠定了基础。政策的支持带动了互联网经济的发展，越来越多的中国人卷入网络直播，网络直播的持续走高致使民间资本纷纷跟进，投资热促使了各大直播平台争相抢占先机。

直播是指广播电视节目的后期合成、播出同时进行的播出方式，是充分体现广播电视媒介传播优势的播出方式。电视现场直播为在现场随着时间的发生、发展进程同时制作和播出电视节目的不出方式。网络直播吸取和延续了互联网的优势，利用视讯方式进行网上现场直播，把产品展示、相关会议、背景介绍、方案 测评、网上调查、对话访谈、在线培训等内容现场发布到互联网上，利用互联网 的直观、快速、表现形式好、内容丰富、交互性强、地域不受限制等特点，加强活动现场的推广效果。从技术上来讲，网络的发展打破了现实和虚拟的界限，只需一部手机就可以实现人与人之间随时、随地、随播的实时交流互动。内容生产上的随意性让网络直播生活化，吃饭、睡觉、唱歌等日常化场景成为直播内容，扩大了市场，满足了各类人群的不同需求；传播渠道的多样化使同一事件得以从不同视角展现在人们面前，让人们的判断更加真实客观。

软件使用对象为非技术人员，也就是说使用该软件的用户将不具备专业性知识。所以我们需要将所有技术层面设置进行封装处理，在用户也就是主播端进行音视频采集，对采集到的音频数据和视频数据分别进行处理。对音频数据进行抽样，再对其进行 PCM 编码，编码方式有多种多样，我们将结合实际用户使用体验选择最高效的编码方式。将视频文件编码，视频格式（RGB）变为 H.264。数据流图如图 1-1 所示；

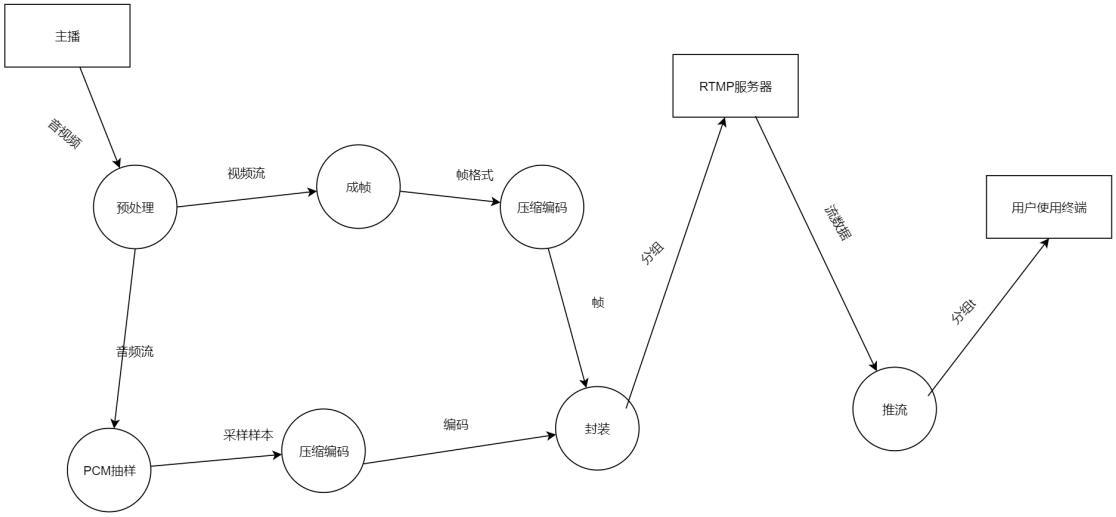

图 1-1 数据流图

1.1.2 复杂工程问题归纳

新媒体数据发布平台的实现难点主要体现在数据采集、音视频编码、推流和服务器 4 个方面。

（1）数据采集：需要掌握 windows 音视频编程基础知识，包括音视频采集格式，windows 系统提供的音视频 API，音视频编码标准等。用以实现软硬件的统一与结合。

（2）编码：采集的音视频一般为原始的没有经过压缩，占用空间较大，音频一般为 PCM 格式。因此需要经过编码以提高信息传输效率与音视频质量。音视频的编码有许多的标准，如何选择标准同时兼顾软硬件的实现是一个比较关键的问题。

（3）推流：推流主流采用 RTMP 协议，因此我们使用 RTMP 协议作为推流基础，但是如何实现 RTMP 的推流，如何保证音视频传输是稳定性依然是一个巨大的挑战。

实施方案可行性研究

1.2.1 国内外研究现状

网络直播实现了“颜值经济”向“荷尔蒙经济”的转变，传统的审美理念、认同标准正在慢慢消解，群体不再仅仅注重外在的鲜明特质，而转向对“感觉”“状态”“对话”的关注上，传统的社群被消解，网络直播的人们生活在虚拟的世界中。网络直播的低门槛消解了原有的话语表达体系，完成了自我的再塑造。网络直播使主播凭借个人魅力加冕为王，主播和粉丝、粉丝和粉丝之间的互动催生出新的消费方式。国内有不少直播平台，如斗鱼、虎牙、花椒、企鹅电竞、bilibili 等，均含有大量的用户。下图是进几年国内直播用户数统计。

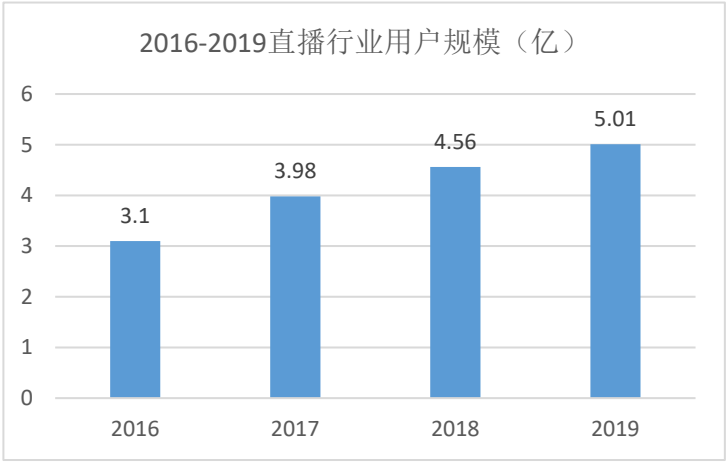

图 1-2 直播行业发展图

直播行业也创造了相当可观的收入，以下是对 2019 年直播行业收入的统计。可以看出，直播行业具有较大的潜力。如图 1-2 所示；

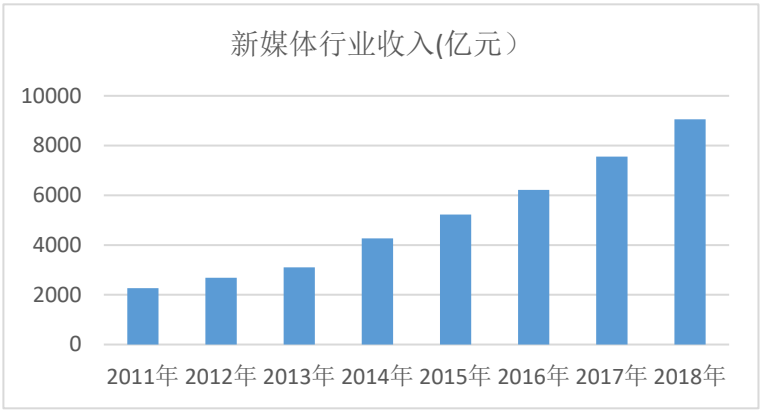

图 1-3 直播行业收入柱状图

直播行业作为互联网衍生物，在发展初期痛点也十分明显，盈利模式单一，主要依靠付费用户。单一的商业模式和产品种类并不能促使直播行业的稳步前行。而在整体行业逐步成熟的背景下，“直播+”让行业的价值进一步释放。直播+公益，直播+电商、直播+音乐和直播+电竞纷纷成为各个平台突破=天花板的主要方式。

而相应的技术基础也逐渐成熟，由于开源运动的发展，端到端的直播实现已经有一条完整的技术路线。开源项目如 FFmpeg、opencv 等为音视频编码提供了相当完整的规范标准。最近几年兴起的流媒体传输协议也有比较成熟的封装。如 RTMP 协议在 FFmpeg 中有比较高层的实现，使得开发流媒体服务应用变得更加容易。

1.2.2 复杂问题可行性分析

实现音视频编解码。通过使用开源的比较成熟的 FFmpeg 库和 opencv 库来做相应的处理. FFmpeg 是一套可以用来记录、转换数字音频、视频，并能将其转化为流的开源计算机程序。采用 LGPL 或 GPL 许可证。它提供了录制、转换以及流化音视频的完整解决方案。它包含了非常先进的音频/视频编解码库 libavcodec，为了保证高可移植性和编解码质量，libavcodec 里很多 code 都是从头开发的。

推拉流的实现。推流是将输入视频数据推送至流媒体服务器，输入视频数据可以是本地视频文件(avi，mp4，flv.....),也可以是内存视频数据，或者摄像头等系统设备，也可以是网络流 URL。FFmpeg 提供了完整的封装，能够为推流提供支撑。

表 1-1 FFmpeg 数据结构

数据结构	说明
AVformatContext	FFmpeg 中表示 format 上下文的结构体，是主要的外部接口结构体，包含了多媒体文件或流的基本信息，用于文件输入和输出操作。
AVCodecContext	FFmpeg 中表示 codec 上下文结构体，其成员包含了与编码和解码有关的参数。
AVCodec	编解码结构体，包含了编解码器信息。
AVStream	描述媒体流的结构体，包含了媒体流的有关信息。
AVPcaket	用于暂存解复用后、解码前媒体数据及附加信息的结构体。
AVFrame	用于暂存未压缩编码媒体数据及附加信息的结构体。
AVCodecID	解码器 ID。

得到了经过编码后的音视频流后，利用 RTMP 封装该流，然后将该流推送到例如 SRS 或者 Wowza 流媒体等服务器，生成拉流的 URL 进行发布，以供客户

使用。常见的第三方流媒体服务器有以下几种：

表 1-2 AMS/wowza/red5/SRS 流媒体服务器产品对比

	Sewise	Wowza	Adobe Flash	Microsoft IIS
Flash(HTTP Streaming)	√	√	√	×
IPhone/iPad(HTTP Streaming)	√	√	√	×
Silverlight(Smooth Streaming)	×	√	×	√
IPCAM(RTSP)	√	√	×	√
HTTP-TS	√	√	×	×
RTMP(Flash & H.264/ACC)	√	√	√	×
RTSP/RTP(H.264/ACC;TCP)	√	√	×	×
图形界面管理	√	×	√	×
HTTP(TS Streaming)	√	×	×	√

第二章 针对复杂工程问题的方案设计与实现

2.1 针对复杂工程问题的方案设计

2.1.1 总体方案设计

系统由以下 4 个部分构成：

1、数据采集层，用于采集系统音频和视频数据，传递到下一层的编码层。

2、编码层，用于对上一层数据的封装，压缩。以降低码率，提高信息的传播效率，在现有的带宽前提下，提高音视频的质量。

3、RTMP 推流层，主要实现对上一层传来的音视频流的封装，和推送到 Wowza 服务器的关键步骤。

4、Wowza 服务器层，主要实现 Wowza 服务器的本地搭建，构建高可靠性的流媒体服务。为推流端实现后端支持，同时为下游应用提供统一的访问接口。

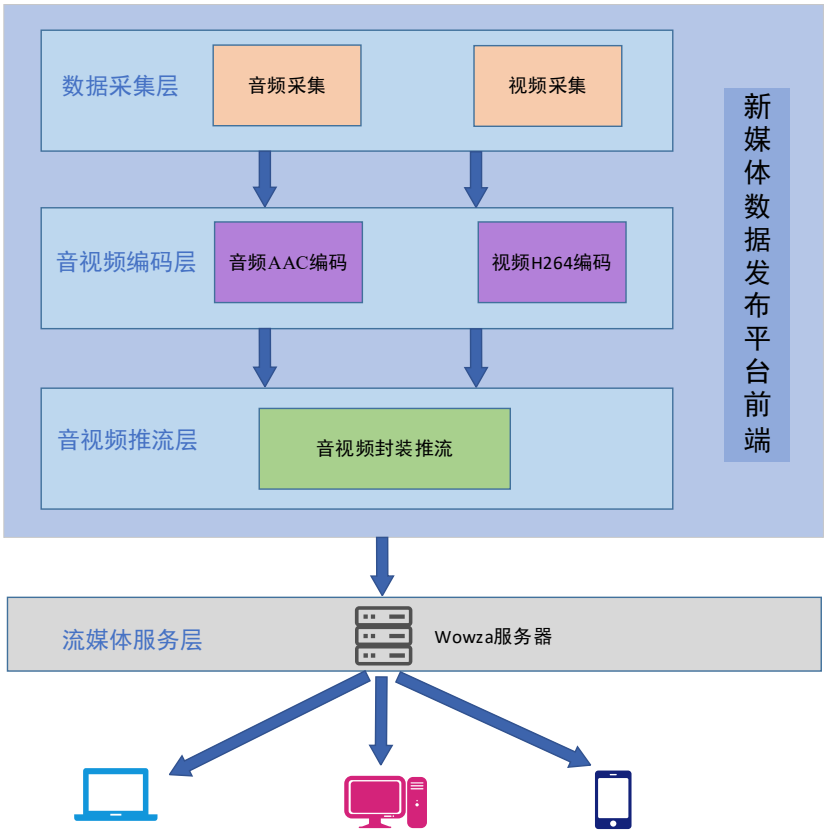

图 2-1 系统架构

这几个结构有机结合，初步实现了新媒体数据发布平台的搭建。

2.1.2 详细方案设计

2.1.2.1 音视频数据采集

音频采集通过 QT 库中的 QAudioInput ，QIODevice 进行 PCM 脉冲编码调制采集，将采集到的音频帧封装到固定的数据结构中，包括帧数据，帧大小和时间戳。供编码层调用。

视频采集通过 opencv 库中的 VideoCapture 采集，初始化获取 height、width、fps 信息。将每一帧读取到 opencv 库中的 Mat 类中。 再进一步封装帧数据，帧的大小，以及时间戳封装到与音频相同的数据结构中。

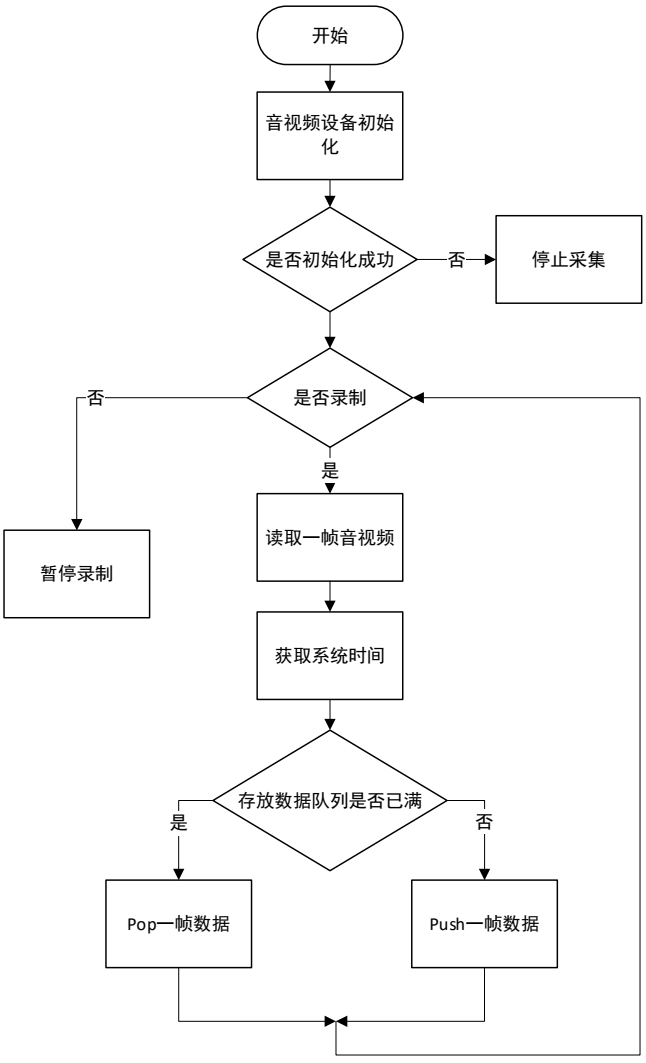

图 2-2 音视频采集

封装过程中，由于网络延迟、流媒体处理性能的限制，以及采集过程中设备中断等问题，并不能保证所有数据都能够完全推送到 Wowza 流媒体服务器，因此，必须适当舍弃一部分数据。考虑到流媒体服务的实时性，我们选择舍弃时间最久远的数据。因此，固定长度的队列是理想的解决方案。我们构造一个固定长度的存放音视频帧的队列，具体在实现推流时，pop 一帧数据，在采集端采集数据时，当发现队列已经满，就再 pop 一帧，这样保证存放到队列中的数据是最新的版本，并且长度大小受到限制。

2.1.2.2 音视频编码

音频编码：通过调用 FFmpeg 库，进行音视频的编码。首先初始化 AAC 格式音频编码器（AAC 系统包含了滤波器组、心理声学模型、量化与编码、预测、TNS、立体声处理和增益控制等多种高效的编码工具。这些模块或过程的有机组合形成了 AAC 系统的基本编解码流程），包括采样率，通道数，比特率，采样格式等，下图是不同的 pcm 格式，主要区别在于位深和通道数。然后在对原始输出进行重采样。再将重采样的数据进行编码。编码成 AAC 格式，便于下一阶段进一步封装成 FLV 的流媒体数据格式。

图 2-3 pcm 格式

具体编码时，先将原始数据封装成一固定格式的数据帧，AAC 音频文件的每一帧由 ADTS Header 和 AAC Audio Data 组成。结构如图 2-4 所示。随后后台新建线程处理帧，通过 FFmpeg 相关接口获取到刚刚处理完成的帧。最后把这个帧和每一帧的 index 结合，交给 RTMP 推流端处理。

视频编码时，跟音频处理类似，首先第一步初始化编码器，创建一个编码 H264

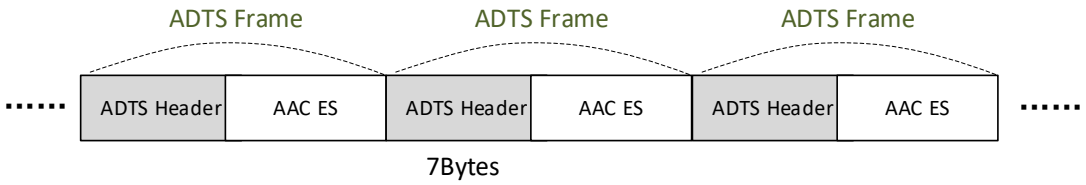

图 2-4 AAC 结构

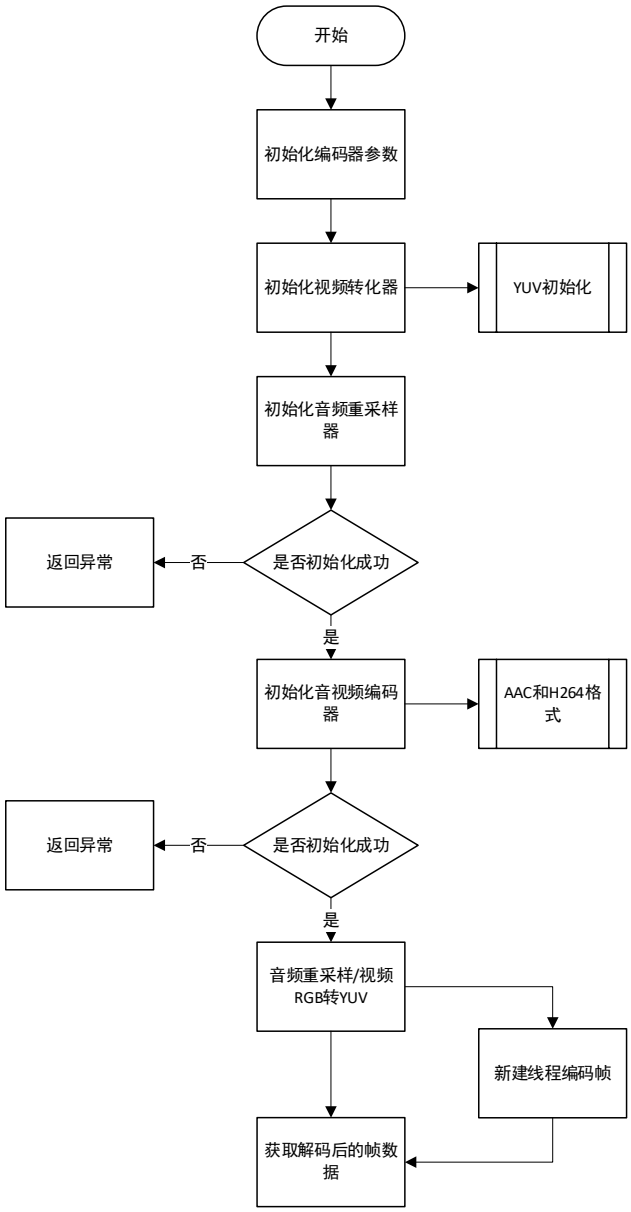

图 2-5 音视频编码

压缩标准的视频编码器，初始化，比特率，宽度和高度，帧率等信息，还包括画面组大小（即每一帧数据由几个视频帧构成）和色彩格式 YUV。获取到帧后，首先

进行 RGB 转 YUV 格式的预处理。YUV 格式对视频编码时，会考虑人类的感知能力，所以允许降低色度的带宽，通过降低色彩带宽，能够在一定程度上降低码率，同时，从感官角度上来看，视频观看者不会觉察到任何与 RGB 格式的区别。因此，进行色彩格式的转换对于提升视频质量来说是十分必要的。然后将处理后的数据格式转化为 AVFrame。通过 avcodec_send_frame 与 avcodec_receive_packet 的配合，完成编码。这种多线程处理方式对于提升程序性能有较大的益处，同时还能防止阻塞主线程导致的程序卡顿。最后将编码后的数据传递给 RTMP 推流端。总体流程如图 2-5 所示。

2.1.2.3 音视频同步推流

推流通过 FFmpeg 相关的库，使用 Rtmp 协议，完成音视频处理的最关键一步。推流时，首先初始化封装器，包括输出 flv 格式，推流服务器地址等信息，初始完成后，第一次推流时注册所有封装器，再注册所有网络协议。注册完成后，发送封装头信息。此时完成所有初始化操作。

具体推流时，通过控制器调用发送命令，将编码层的输出，包括音频和视频经过时间基准变换（即统一时间，以推流上下文为基准，而不是音视频采集的时间戳），封装成 RTMP 格式，同时写入到 RTMP 流中，完成推流。RTMP 协议是应用层协议，是要靠底层可靠的传输层协议（通常是 TCP）来保证信息传输的可靠性的。在基于传输层协议的链接建立完成后，RTMP 协议也要客户端和服务器通过“握手”来建立基于传输层链接之上的 RTMP Connection 链接，在 Connection 链接上会传输一些控制信息。RTMP 协议传输时会对数据做自己的格式化，这种格式的消息称之为 RTMP Message，而实际传输的时候为了更好地实现多路复用、分包和信息的公平性，发送端会把 Message 划分为带有 Message ID 的 Chunk，每个 Chunk 可能是一个单独的 Message，也可能是 Message 的一部分，在接受端会根据 chunk 中包含的 data 的长度，message id 和 message 的长度把 chunk 还原成完整的 Message，从而实现信息的收发。

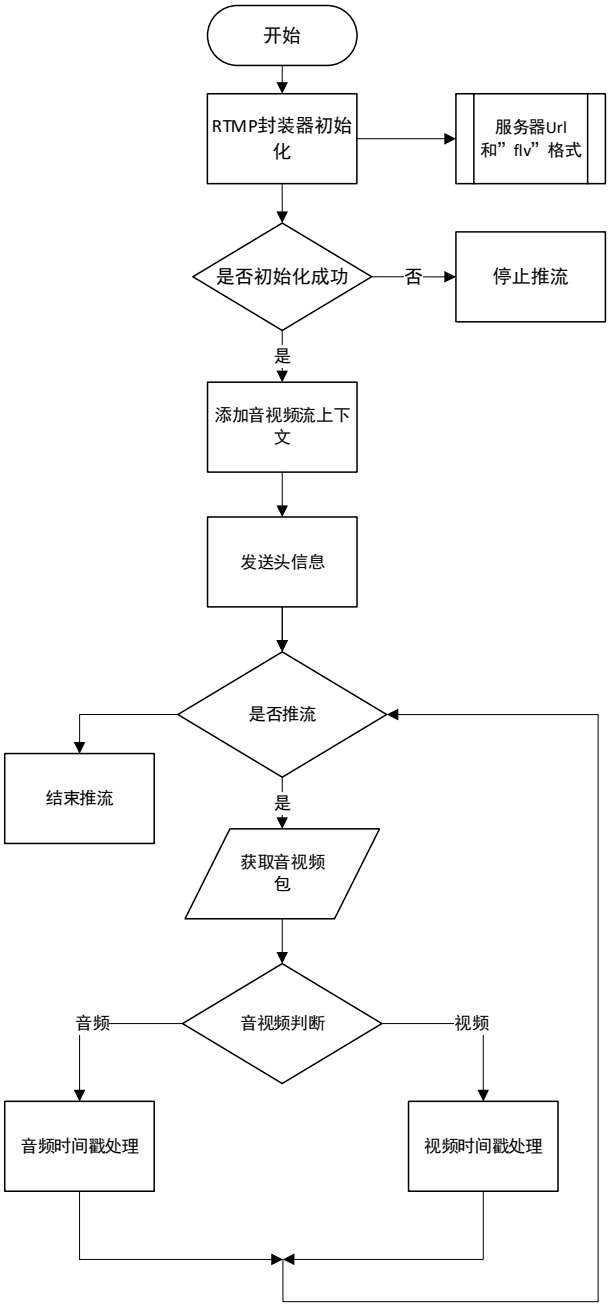

图 2-6 推流流程图

2.1.2.4 Wowza 服务器搭建

流媒体服务器提供的流式传输方式有两种：顺序流式传输和实时流式传输两种方式，顺序流式传输是顺序下载，在下载文件的同时用户可观看在线媒体。如果

使用普通的 HTTP 服务器，将音视频数据以从头至尾方式发送，则为顺序流媒体传输。这种方式显然不适合于处理多用户同时观看的直播平台。实时流式传输总是实时传送，特别适合现场事件。实时流式传输必须匹配连接带宽，这意味着图像质量会因网络速度降低而变差。在流式传输时，流媒体数据具有实时性，等时性等基本特点，流服务器和客户终端要保证各种媒体间的同步关系，因此，流媒体传输对“最大延时”,“延时抖动”等 QoS 参数都有严格要求。

1) 我们综合成本以及时间考虑，选择 Wowza 作为我们的流媒体服务器。搭建环境为本地计算机，运行系统为 windows10。搭建步骤如下：
2) 到官网[1]下载 Wowza 服务器。同时获取学生免费密钥。
3) 安装 Wowza 到本地计算机，开启 Wowza 服务，同时开放 8080 端口，提供给本地计算机外的设备上的用户访问。
4) 进入本地 Wowza 网页控制台，新建流媒体服务器，同时配置密码以及开放权限等信息。
5) 安装 Adobe Flash Player 等必要播放 RTMP 的网页插件。此时，会生成流媒体服务器的访问地址。

2.2 针对复杂工程问题的方案实现

我们系统类图如图 2-7 所示，主要包含 3 个部分，音视频采集、音视频编码、音视频推流。音视频采集部分由两个子类分别完成，通过 Qt 自带的音视频处理库将采样得来的信息传递到编码层。同样继承自 DataThread 类的 Controller 方法是控制层，主要目的为了初始化音视频编码器和 Rtmp 推流的初始化。编码层中 MediaEncode 类为编码相关，主要目的是将音视频采集层传递过来的音视频信息编码压缩成 ACC 的音频格式或者 H.264 的视频格式，因为原始 PCM 音频采样信息和 RGB 视频采样信息较多，如果不经过压缩，难以在现有的网络基础上实现高分辨率的视频直播。因此，编码层是非常有必要的。

2.2.1 开发环境选择

直播推流要使用到 FFmpeg 库，FFmpeg 是一个自由软件，包含完整的音视频解码编码推流解决方案，基于 FFmpeg 的底层开发主要使用 C 或 C++，因此我们

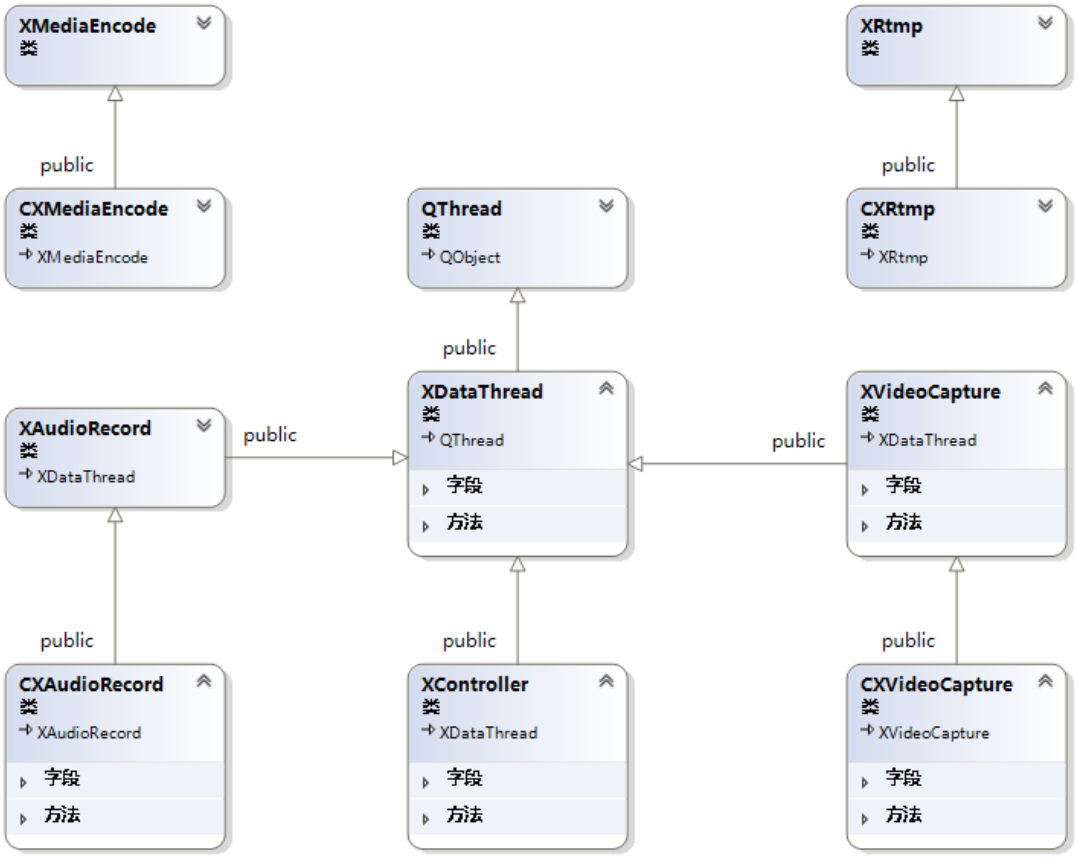

图 2-7 类图

使用 C++作为开发语言，选择的 IDE 为 VS Studio2019。FFmpeg 选择最新的 4.2.1 版本。

我们需要构建图形化交互界面，因此需要基于 C 的图形界面设计工具。Qt 是一个跨平台应用程序和 UI 开发框架。使用 Qt 只需一次性开发应用程序，无须重新编写源代码，便可跨不同桌面和嵌入式操作系统部署这些应用程序。Qt 也包含 QtCreator 图形化设计 IDE，便于我们快速构建图形化交互界面。同时 VS Studio 集成了 Qt 插件，使得我们较为容易的利用 VS Studio 的强大功能的同时能够编辑图形化界面。Qt 库中也包含大量封装好的模块，例如音频的采集，这些功能降低了开发音视频图形化界面的难度。Qt 版本我们选择最新的 5.13。

直播推流中比较重要的部分是流媒体服务器的搭建，流媒体服务器主要实现推流和拉流的功能。市场上有许多流媒体服务器，包括 AMS(FMS)流媒体服务器系统、wowza 流媒体服务器系统、Red5 流媒体服务器系统、SRS 流媒体服务器系统等。为了构建较为稳定的新媒体发布与实现系统，我们选择使用第三方流媒体服

务器，以构建高稳定性的流媒体服务源。在这些流媒体服务器中，Wowza 流服务器支持的音视频格式完整，并且提供学生免费试用版本，因此我们选择 Wowza 服务器作为推流的目标。

我们也实现了简单的推流前端，使用 WebStorm 和 Chrome 作为开发调式工具，除此之外，我们也使用了开源的播放器 VLC 和开源推流客户端 OBS 作为程序完成开发前的调试软件。我们的新媒体数据发布平台是基于 Windows10 系统开发完成。

2.2.2 音视频采集实现

多媒体模块的导入 Qt5.8 版本中的多媒体模块提供了一组丰富的 QML 类型和 C++ 类以满足多媒体开发的需要，另外还提 供了必要的 API 接口来访问照相机和录音机设备。要在 Qt5.8 以上的版本中使用多媒体模块，首先应该在工程中进行如下的设置：

1. FFMpeg 头文件及动态库的添加把编译好的 FFmpegSDK 文件 放到一个文件夹内，首先在工程中设置好头文件路径以及需要的库文件，在工程中的文件内包含使用的头文件，注意的是，因为 FFMepg 库的接口都是 C 函数头文件中也没有 extern “C”的声明，而 Qt 是 C++ 环境，所以在使用头文件时，需要添加 extern “C”。

2. 麦克风音频的采集和编码 Qt 中的 QAudioInput 类可以进行本地的麦克风、收音机等设备的音频采集，而 QAudioOutput 类可以实现把本地的音频数据输出到音频输出设备，实现音频的播放功能。具体的步骤如下：

（1）获取声音的输入输出设备，可以直接获取默认的设备，也可以在程序中查找声音输入输出设备，然后指定，我们采用直接获取默认设备的方式。

（2）设置 QAudioFormat 格式，QAudioInput 和 QAudioOutput 对象的 QAudioFormat 格式应一致，否则播放的声音就与输入的声音不同。

（4）打开声音输入输出设备，将分别返回 QIODevice 指针。

（5）当声音输入设备采集到声音数据后将触发 readyRead()信号，所以将此信号与该类的 captureDataFromDevice() 函数连接即可在此函数内处理得到的原始数据。

下面是部分关键代码：

代码 2-1 音频输入

```
class CXAudioRecord :public XAudioRecord{
public:
  void run() {
    int readSize = nbSamples*channels*sampleByte;
    char *buf = new char[readSize];
    while (!isExit){
      if (input->bytesReady() < readSize){
        QThread::msleep(1);
        continue;
      }
      while (size != readSize){
        int len = io->read(buf + size, readSize - size);
        if (len < 0)break;
        size += len;
      }
      if (size != readSize) continue;
      long pts = GetCurTime();
      Push(XData(buf, readSize,pts));
    }
    delete buf;
  }
  bool Init(){
    QAudioFormat fmt;
    fmt.setSampleRate(sampleRate);//样本率44k
    fmt.setChannelCount(channels);//声道数2
    fmt.setSampleSize(sampleByte * 8);//样本字节16
    fmt.setCodec("audio/pcm");
    fmt.setByteOrder(QAudioFormat::LittleEndian);
    fmt.setSampleType(QAudioFormat::UnSignedInt);
    QAudioDeviceInfo info = QAudioDeviceInfo::defaultInputDevice();
    input = new QAudioInput(fmt);
    io = input->start();
    return !io?:false:true;
  }
XAudioRecord *XAudioRecord::Get(XAUDIOTYPE type){
  static CXAudioRecord record;
  return &record;
}
```

代码 2-2 视频输入

```
class CXVideoCapture :public XVideoCapture{
public:
  VideoCapture cam;
  void run()
  {
    Mat frame;
    while (!isExit){
      if (!cam.read(frame)){
        msleep(1);
        continue;
      }
      if (frame.empty()){
        msleep(1);
        continue;
      }
      fmutex.lock();
      for (int i = 0; i < filters.size(); i++){
        Mat des;
        filters[i]->Filter(&frame, &des);
        frame = des;
      }
      fmutex.unlock();
      XData d((char *)frame.data,
frame.cols*frame.rows*frame.elemSize(),GetCurTime());
      Push(d);//将数据放到队列中
    }
  }
  bool Init(int camIndex =0 ){
    cam.open(camIndex);
    if (!cam.isOpened()){
      return false;
    }
    cout <<camIndex<< "cam open success!" << endl;
    width = cam.get(CAP_PROP_FRAME_WIDTH);
    height = cam.get(CAP_PROP_FRAME_HEIGHT);
    fps = cam.get(CAP_PROP_FPS);
    if (fps == 0)fps = 25;
    return true;
  }
```

2.2.3 音视频编码实现

目前，主流的 H.264 开源编码器有 JM、T264 和 x264141：JM 程序结构冗长，编码复杂度高，不适合实时视频传输。AAC 音频编码器种类繁多，主要的编码器有 Fraunhofer IIS、FhG、Nero AAC、QuickTime / iTune、FAAC、Divx AAC。

FFmpeg，提供了录制、转换以及流化视音频的完整解决方案，支 MPEG、DivX、MPEG4、FLV 等 40 多种编码和 AVI、MPEG、OGG、Matroska 等 90 多种解码，同时也支持很多第三方的编解码库，音频编解码库如 libfaac、libfdk aac、libmp31ame、libvo—aacenc、libopus、libvorbis 等，视频编解码库如 libx264 / libx264rgb、libxvid、mpe92、png 等。我们借助 FFmpeg 完成 h.264 视频的编码、AAC 音频的编码。由于 FFmpeg 注册初始化处理是全局的，当程序某线程第一个自定义的编解码器对象

代码 2-3 音频编码

```
class CXMediaEncode :public XMediaEncode{
public:
  bool InitAudioCodec(){
    if (!(ac=CreateCodec(AV_CODEC_ID_AAC))){
      return false;
    }
    ac->bit_rate = 40000;
    ac->sample_rate = sampleRate;
    ac->sample_fmt = AV_SAMPLE_FMT_FLTP;
    ac->channels = channels;
    ac->channel_layout = av_get_default_channel_layout(channels);
    return openCodec(&ac);
  }
  bool InitVideoCodec(){
    if (!(vc=CreateCodec(AV_CODEC_ID_H264))) {
      return false;
    }
    vc->bit_rate = 50 * 1024 * 8;//压缩后每秒视频的bit位大小 1秒50kB
    vc->width = outWidth;
    vc->height = outHeight;
    vc->framerate = { fps,1 };
    vc->gop_size = 15;
    vc->max_b_frames = 0;
    vc->pix_fmt = AV_PIX_FMT_YUV420P;
    return openCodec(&vc);
  }
```

被创建时完成注册初始化处理就可以了，以后的对象不需要再次进行注册初始化。

代码 2-4 编码

```
XData EncodeAudio(XData frame){
    XData r;
    if (frame.data <= 0 || !frame.data)return r;
    AVFrame *p = (AVFrame *)frame.data;
    if (lasta == p->pts){
      p->pts += 1200;
    }
    lasta = p->pts;
    int ret = avcodec_send_frame(ac, p);
    if (ret != 0)return r;
    av_packet_unref(&apack);
    ret = avcodec_receive_packet(ac, &apack);
    if (ret != 0)return r;
    r.data = (char *)&apack;
    r.size = apack.size;
    r.pts = frame.pts;
    return r;
  }
  XData EncodeVideo(XData frame){
    av_packet_unref(&vpack);
    XData r;
    if (frame.data <= 0 || !frame.data)return r;
    AVFrame *p = (AVFrame *)frame.data;
    int ret = avcodec_send_frame(vc, p);
    if (ret != 0) return r;
    ret = avcodec_receive_packet(vc, &vpack);
    if (ret != 0||vpack.size<=0) return r;
    r.data = (char *)&vpack;
    r.size = vpack.size;
    r.pts = frame.pts;
    return r;
  }
```

2.2.4 音视频推流

FFmpeg 中对影音数据的处理，可以划分为协议层、容器层、编码层与原始数据层四个层次。协议层提供网络协议收发功能，可以接收或推送含封装格式的媒体

流。协议层由 libavformat 库及第三方库(如 librtmp)提供支持。容器层处理各种封装格式。容器层由 libavformat 库提供支持。编码层处理音视频编码及解码。编码层由各种丰富的编解码器(libavcodec 库及第三方编解码库(如 libx264))提供支持。原始数据层处理未编码的原始音视频帧。原始数据层由各种丰富的音视频滤镜(libavfilter 库)提供支持。

本次课程设计使用的的收流与推流的功能，属于协议层的处理。FFmpeg 中 libavformat 库提供了丰富的协议处理及封装格式处理功能，在打开输入/输出时，FFmpeg 会根据输入 URL/输出 URL 探测输入/输出格式，选择合适的协议和封装格式。例如，如果输出 URL 是 rtmp://192.168.0.104/live，那么 FFmpeg 打开输出时，会确定使用 rtmp 协议，封装格式为 flv。FFmpeg 中打开输入/输出的内部处理细节用户不必关注。

关于使用的 RTMP 协议，RTMP 协议封包由一个包头和一个包体组成,包头可以是 4 种长度的任意一种:12, 8, 4, 1 byte(s).完整的 RTMP 包头应该是 12bytes,

表 2-7rtmp 协议

用途	大小（Byte）	含义
Head-Type	1	包头
TIMMER	3	时间戳
AMFSize	3	数据大小
AMFType	1	数据类型
StreamID	4	流 ID

包含了时间戳,AMFSize,AMFType,StreamID 信息, 8 字节的包头只纪录了时间戳,AMFSize,AMFType,其他字节的包头纪录信息依次类推。包体最大长度默认为 128 字节,通过 chunkSize 可改变包体最大长度,通常当一段 AFM 数据超过 128 字节后,超过 128 的部分就放到了其他的 RTMP 封包中。

DTS 即解码时间戳，告诉播放器该在什么时候解码这一帧的数据；PTS 即显示时间戳，告诉播放器该在什么时候显示这一帧的数据。虽然 DTS、PTS 是用于指导播放端的行为，但它们是在编码的时候由编码器生成的。要实现音视频同步，通常需要选择一个参考时钟，参考时钟上的时间是线性递增的，编码音视频流时依据参考时钟上的时间给每帧数据打上时间戳。在播放时，读取数据帧上的时间戳，同时参考当前参考时钟上的时间来安排播放。这里的说的时间戳就是我们前面说的 PTS。在实现中，我们选择同步音频和视频到外部时钟自定义数据结构类型 XData，不仅存放音视频帧裸数据内容，也包含自定义长整型变量 pts 记录当前时间戳，在开始采集和开始处理音频、视频数据时分别获取当前基于计算机系统时间

的时间戳，那么在正式处理每一帧音视频时两者可以得到每一帧音视频的时间戳，而且是基于相同的时间基数进行运算，从而达到音视频的同步。

代码 2-5 RTMP 推流

```
class CXRtmp :public XRtmp{
public:
  bool Init(const char *url){
    int ret = avformat_alloc_output_context2(&ic, 0, "flv", url);
    this->url = url;
    if (ret != 0){
      char buf[1024] = { 0 };
      av_strerror(ret, buf, sizeof(buf) - 1);
      cout<<buf<<endl;
      return false;
    }
    return true;
  }
  int AddStream(const AVCodecContext *c){
    if (!c)return -1;
    AVStream *st = avformat_new_stream(ic, NULL);
    st->codecpar->codec_tag = 0;
    avcodec_parameters_from_context(st->codecpar,c);
    av_dump_format(ic, 0,url.c_str(), 1);
    if (c->codec_type == AVMEDIA_TYPE_VIDEO){
      vc = c;
      vs = st;
    }
    else if (c->codec_type == AVMEDIA_TYPE_AUDIO){
      ac = c;
      as = st;
    }
    return st->index;
  }
  bool SendHead(){
    int ret = avio_open(&ic->pb, url.c_str(), AVIO_FLAG_WRITE);
    if (ret != 0){
      char buf[1024] = { 0 };
      av_strerror(ret, buf, sizeof(buf) - 1);
      cout<<buf<<endl;
      return false;
    }
```

```
  ret = avformat_write_header(ic, NULL);
  if (ret != 0){
    char buf[1024] = { 0 };
    av_strerror(ret, buf, sizeof(buf) - 1);
    return false;
  }
  return true;
}
bool SendFrame(XData d,int streamIndex)
{
  if (!d.data||d.size <= 0 )return false;
  AVPacket *pack = (AVPacket *)d.data;
  pack->stream_index = streamIndex;
  AVRational stime;
  AVRational dtime;
  if (vs&&vc&&pack->stream_index == vs->index){
    stime = vc->time_base;//上下文
    dtime = vs->time_base;//视频流
  }
  else if (as&&ac&&pack->stream_index == as->index){
    stime = ac->time_base;
    dtime = as->time_base;
  }
  else{
    return false;
  }
  pack->pts = av_rescale_q(pack->pts, stime, dtime);//时间基准转换
  pack->dts = av_rescale_q(pack->dts, stime, dtime);
  pack->duration = av_rescale_q(pack->duration, stime, dtime);
  int ret = av_interleaved_write_frame(ic, pack);
  if (ret == 0){
    return true;
  }
  return false;
    }
```

2.2.5 Wowza 部署

Wowza Streaming Engine 是强大的媒体服务器软件，提供可靠性强，流畅度高的高质量视频和音频到任何设备的传输。无论是在云端 Wowza Streaming Engine 都

可以提供强大的组件，让我们不仅可以保证工作流的流畅性还有安全性。owza Streaming Engine 4 (也就是著名的 Wowza Media Server)是一个高性能、可扩展的流媒体服务器软件，支持直播、VOD、在线视频聊天、远程录制功能，它支持多种播放器技术，包括：

1.Adobe HTTP Dynamic Streaming (HDS)协议，Adobe Flash 播放器

2.Apple HTTP Live Streaming (HLS)协议， iPhone, iPad, iPod touch 设备, Safari 浏览器, Quicktime 播放器

3.Microsoft：Smooth Streaming 协议. Microsoft Silverlight 技术

4.MPEG-DASH streaming. DASH clients.Real Time Streaming Protocol (RTSP/RTP). QuickTime 播放器, VLC 媒体播放器, 以及许多移动终端

5.MPEG-2 Transport Streams (MPEG-TS). 机顶盒和 IPTV 。

Wowza 也融合了其他系统和第三方解决方案，提供开发 IDE 工具，组件和 APIs，可以很方便地开发自定义的模块，用以扩展新的功能。

Wowza 也比较方便部署，可以在一台或者多台电脑上运行 Wowza Streaming Engine，也可以在云端部署 Wowza 软件。包括 AmazonEC2，Google Computer Engine 等 Wowza 服务器得分布式部署原理如下图所示；

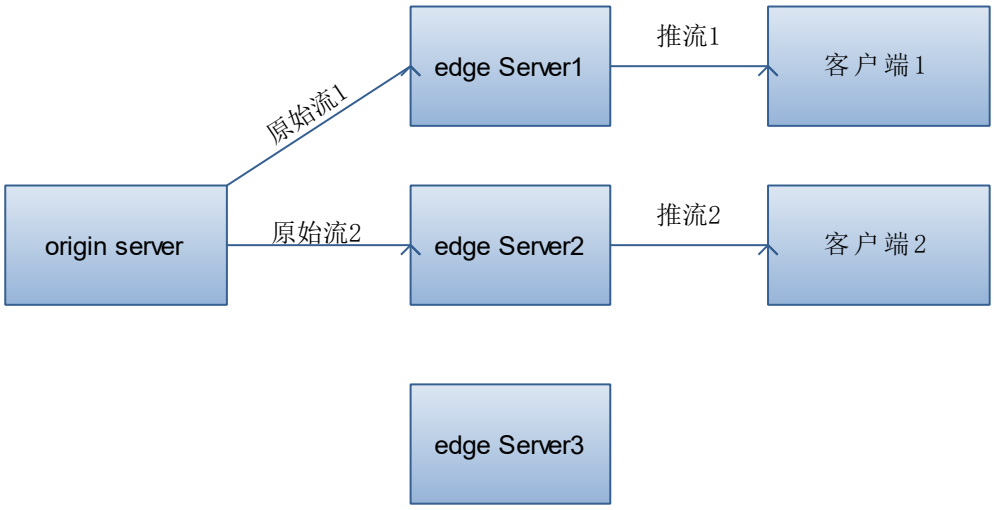

图 2-8 部署原理

第三章 测试环境构建与测试驱动开发

测试环境：

- 操作系统：Windows10
- 线上服务器：Wowza 流媒体服务器
- 编译环境：Visual Studio 2019，Qt Creator
- 测试软件：VLC 播放器，Chrome

我们在 windows10 上对我们的系统进行了测试，如下图所示，我们的系统实现了基本的推流拉流功能，还能够推送本地视频到服务器，服务器提供相应的接口能够支持多个用户的访问需求。包括第三方播放器和我们自己做的网页前端，均能够接收到音视频流。

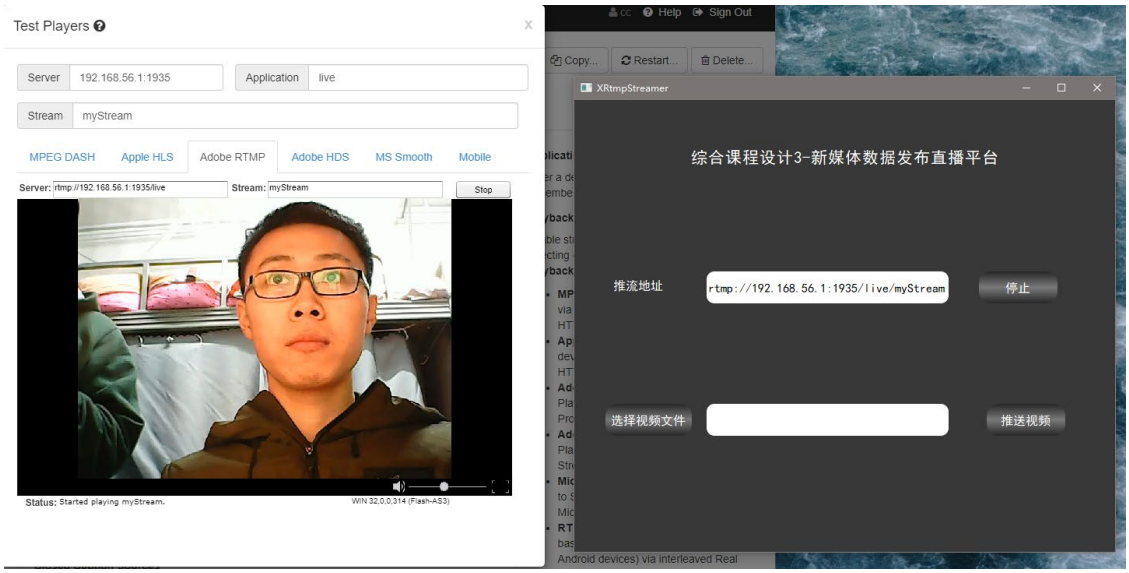

图 3-1 直播推流

下图是通过推送本地视频源所得到的测试图：

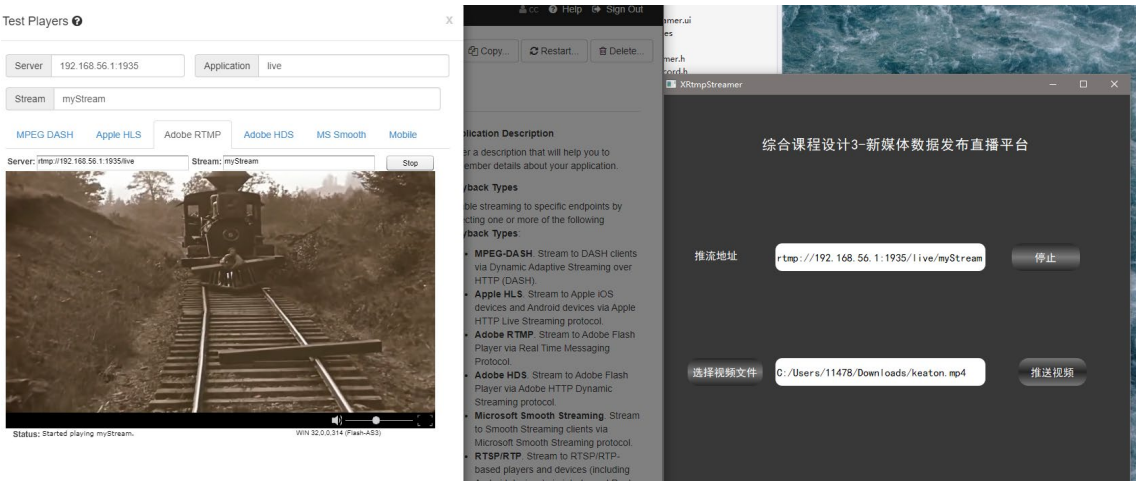

图 3-2 文件推流

下图通过第三方拉流软件 VLC 实现的拉流测试。

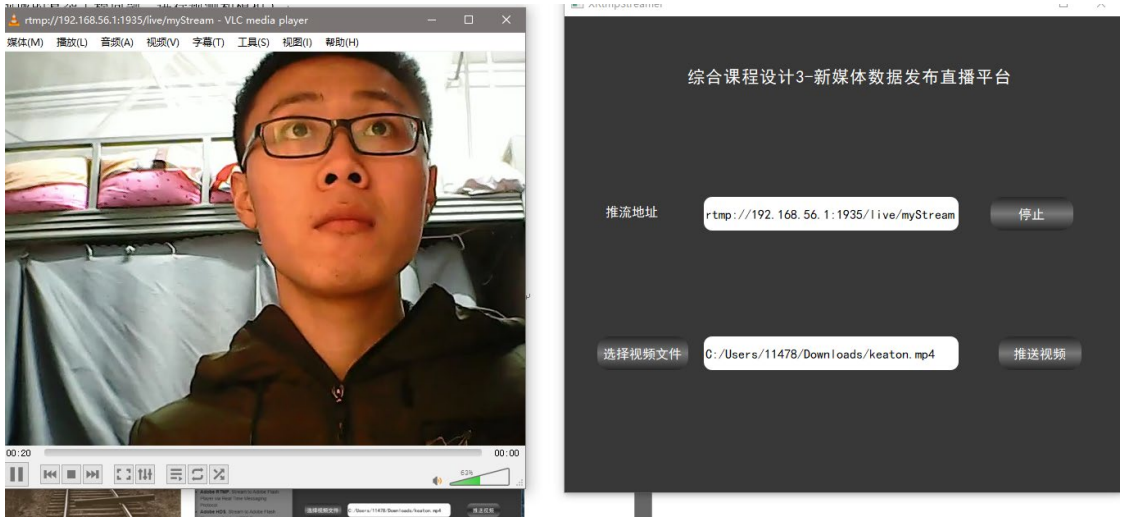

图 3-3 VLC 播放器拉流

下图是 web 端拉流测试。

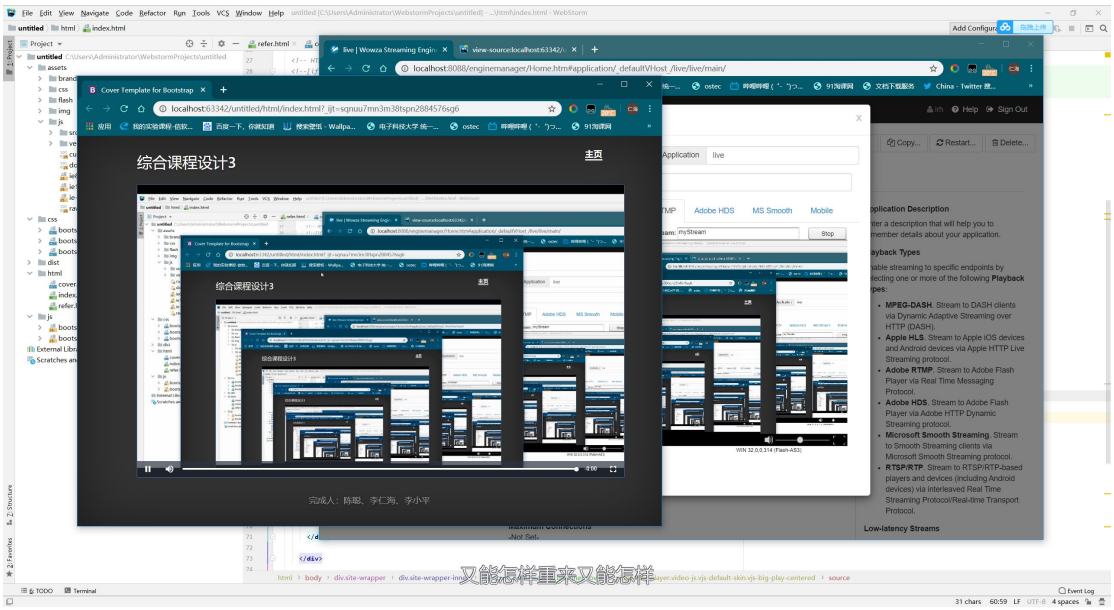

图 3-4 web 网页端拉流

至此，我们的新媒体数据发布与实现系统已经完成。实现了基本的推流与拉流功能，能够为用户提供基础的流媒体访问服务。

第四章 知识技能学习情况

4.1 Microsoft Visual Studio 和 C++

项目中，C++是主要编程实现语言，VS 是主要开发环境。 在本次设计的项目中采用的是 VS 集成开发环境，VS 是 Microsoft Visual Studio 的简称。VS 是一个基本完整的开发工具集，它包括了整个软件生命周期中所需要 的大部分工具，如 UML 工具、代码管控工具、集成开发环境(IDE)等等，同时也是 目前最流行的 Windows 平台应用程序的集成开发环境。因为 VS 支持 C++编程语言，因而本次项目采取的是 C++为实现音视频采集 编码推流的主要编程语言。C++是 C 语言的继承，它既可以进行 C 语言的过程化 程序设计，又可以进行以抽象数据类型为特点的基于对象的程序设计，还可以进行 以继承和多态为特点的面向对象的程序设计。C++不仅拥有计算机高效运行的实用性特征，同时还致力于提高大规模程序的编程质量与程序设计语言的问题描述能力。C++语言灵活，运算符的数据结构丰富、具有结构化控制语句、程序执行效率高，而且同时具有高级语言与汇编语言的优点。

4.2 OpenCV

在项目中，将利用 OpenCV 库中的函数对摄像头采集视频进行处理。 计算机视觉市场巨大而且持续增长，且这方面没有标准 API，如今的计算机视 觉软件大概有以下三种：

1. 研究代码（慢，不稳定，独立并与其他库不兼容）
2. 耗费很高的商业化工具（比如 Halcon，MATLAB+Simulink）
3. 依赖硬件的一些特别的解决方案（比如视频监控，医疗设备）由于 OpenCV 用 C++语言编写，它的主要接口也是 C++语言，因而在本次设计项目中通过使用 OpenCV 库当中的一些函数完成了通过电脑的摄像头对视频进 行压缩编码。OpenCV 是一个基于 BSD 许可（开源）发行的跨平台计算机视觉库， 可以运行在 Linux、Windows、Android 和 Mac OS 操作系统上。它轻量而且高效— —由一系列 C 函数和少量 C++ 类构成，同时提供了 Python、Ruby、MATLAB 等语言的

接口，实现了图像处理和计算机视觉方面的很多通用算法。

4.3 Qt

在项目中，我们会利用 Qt 对推流端进行简单的前端页面框架设计以及声卡采 集音频的处理。Qt 是一个 1991 年由 Qt Company 开发的跨平台 C++图形用户界面应用程序开发框架。它既可以开发 GUI 程序，也可用于开发非 GUI 程序，比如控制台工具和 服务器。Qt 是面向对象的框架，使用特殊的代码生成扩展（称为元对象编译器 Meta Object Compiler, moc）以及一些宏，Qt 很容易扩展，并且允许真正的组件编程。Qt Creator 是一个用于 Qt 开发的轻量级跨平台集成开发环境。Qt Creator 可带来两大关键益处：提供首个专为支持跨平台开发而设计的集成开发环境 (IDE)，并确保首次接触 Qt 框架的开发人员能迅速上手和操作。即使不开发 Qt 应用程序，Qt Creator 也是一个简单易用且功能强大的 IDE。

4.4 PCM 编码与音频重采样

PCM 脉冲编码调制是通过抽样、量化以及编码过程，将采集到的音频以及视频进行压缩编码。脉冲编码调制是数字通信的编码方式之一。主要过程是将话音、图像等模拟信号每隔一定时间进行取样，使其离散化，同时将抽样值按分层单位四舍五入取整量化，同时将抽样值按一组二进制码来表示抽样脉冲的幅值。对于音频重采样的目的是为了能够采集数据文件，能够将原本采集到的格式为 PCM 的流文件转变为格式为 AAC，从而进行 FLV 封装然后推流到服务器。频率对应于时间轴线，振幅对应于电平轴线。波是无限光滑的，弦线可以看成由无数点组成，由于存储空间是相对有限的，数字编码过程中，必须对弦线的点进行采样。采样的过程就是抽取某点的频率值，在一秒中内抽取的点越多，获取得频率信息更丰富。为了复原波形，一次振动中，必须有 2 个点的采样，人耳能够感觉到的最高 频率为 20kHz，因此要满足人耳的听觉要求，则需要至少每秒进行 40k 次采样，即 40kHz 的采样率。常见的 CD，采 44.1kHz。之后需要获得该频率的能量值 并量化，用于表示信号强度。量化电平数为 2 的整数次幂，16bit 的采样大小，即 2 的 16 次方。

音频重采样分为上采样和下采样，即插值和抽取。在实现有理数级重采样时，则是将上采样和下采样做结合。由数字信号处理中，时域信号和频域信号的时-频

对偶特性可知：时域的抽取，对应频域的延拓；时域的插值，对应频域的压缩。如果对信号的频率成分不做限制的话，频域的延拓可能会引发频谱混迭；频域的压缩来引起频谱镜像相应。因此在下采样前，要经过滤波器滤波来防止混迭，即抗混迭滤波；上采样后也要经过滤波处理，即抗镜像滤波。

4.5 RTMP

RTMP 是 Real Time Messaging Protocol（实时消息传输协议）的首字母缩写。该协议基于 TCP，是一个协议族，包括 RTMP 基 RTMPT/RTMPS/RTMPE 等多种变种。RTMP 是一种设计用来进行实时数据通信 的网络协议，主要用来在 Flash/AIR 平台和支持 RTMP 协议的流媒体/交互服务器 之间进行音视频和数据通信。 它是 Adobe Systems 公司为 Flash 播放器和服务器之间音频、视频和数据传 输开发的开放协议就像一个用来装数据包的容器，这些数据既可以是 AMF 格式 的数据，也可以是 FLV 中的视/音频数据。一个单一的连接可以通过不同的通道 传输多路网络流，这些通道中的包都是按照固定大小的包传输的。 播放一个 RTMP 协议的流媒体需要经过：握手、建立链接、建立流、播放/ 发送四个步骤。握手成功之后，需要在建立链接阶段去建立客户端和服务器之间 的“网络链接”。建立流阶段用于建立客户端和服务器之间的“网络流”。播放阶段用于传输音视频数据。

第五章 分工协作与交流情况

为实现整个系统，成员除了需要完成对于相关理论知识的学习、VS 开发环境的学习与搭建，以及通过视频学习推流编码的原理、拉流的相关原理以及播放界面的设计之外，还需要按照分组情况按时完成分配的任务。具体分组情况与任务如表 5-1 所示；

表 5-1 小组任务分工

成员	任务简介	具体任务
陈聪	推流总体设计与实现	完成系统架构设计工作，完成代码的编写
李仁海	前端界面设计	实现拉流前端与文档写作
李小平	Wowza 服务器搭建	搭建 Wowza 服务器，文档写作

5.2 小组交流情况

为了了解小组成员任务完成的情况，小组会定时召开小组内部交流会议，共同讨论在学习与操作的过程中所遇到的问题；每月小组定期向代管老师汇报工作进度，并汇集小组成员无法解决的问题来请教老师。

表 5-2 小组交流情况

时间	交流内容	备注
周五	小组成员之间对接工作情况，交流在系统实现的过程中所遇到的理论或者实践的问题，共同商量解决问题的方案，对无法解决的问题进行记录	主要有服务器搭建，Qt 使用，FFmpeg 编码实现方法
月初	当前工作进展，同时与老师沟通，汇报遇到的问题与计划	

参考文献

[1] Wowza 官网 https://www.wowza.com/, 2020.1.1

[2] FFmpeg 官网. http://ffmpeg.org/ , 2020.1.1

[3] S.G.Mallat. 钱能 C++程序设计教程（第 2 版）[M].北京：清华大学出版社，2005，6-7

[4] openCV 官网 https://opencv.org/, 2020.1.1

[5] Qt Creator，维基百科，2013-11-12

致谢

本报告的工作是在我的指导教师黄俊老师的悉心指导下完成的，经过黄老师的悉心教导，我们顺利的完成了该项目。

在完成过程中，我们特别感谢学校能够提供这样的机会与平台，监督我们完成这样的实际项目。通过这次综合课程设计 III，我们学到了不少实际工程应用知识，为今后的工作打下了基础，最后再次感谢老师的耐心指导。