分类号：TP37　　密　　级：公开

论文编号：

贵州大学

2018 届硕士学位论文

基于 H.264 与 H.265 的低延时视频监控系统的设计与实现

学生姓名：黄天驰

导师姓名：李泽平

学科专业：计算机技术

研究方向：计算机网络与流媒体技术

中国 ▪ 贵州 ▪ 贵阳

2018 年 5 月

目录

基于 H.264 与 H.265 的低延时视频监控系统的设计与实现

摘要

近年来，随着互联网技术的发展，流媒体直播应用得到了迅速的推广普及。与此同时，以视频监控为需求核心的流媒体系统也得到了广泛的关注和应用，用户希望在任何环境下使用不同的设备观看视频监控，监控地点包括，家，工作地点，社会场地等。目前使用传统流媒体直播架构来应对挑战，但是方案延时较高，并不能满足视频监控系统的低延时基本需求。因此提出一个新的视频监控系统架构让其同时拥有高并发性与低延时特性是非常有必要的。本文提出并实现了一个新的基于 H.264 与 H.265 的低延时视频监控系统。通过该系统，用户可以通过不同平台客户端低延时观看市面上大多数支持 RTSP 协议的摄像头的实时视频流。本文的主要工作如下：

1. 介绍了流媒体技术以及数字视频监控技术中的核心技术，通过比对并结合两者各自优点，提出了一种新的视频监控系统架构方法，兼顾了流媒体系统中的高并发特点与视频监控系统的低延时特性。
2. 根据提出的架构方案研究并实现了完整的视频监控系统，该系统包括视频服务器，移动客户端，桌面客户端，视频推流器与数据库中间件。本文将系统的模块逐一设计并实现。
3. 在提出的架构基础上，本文结合队列模型，研究了多种符合需求的特殊数据结构与算法并将其加入了系统，内容包括：H.264 与 H.265 在低延时视频传输中关键部分，如：改进的帧头判断算法；改进的 RingBuffer 数据结构；探究关键帧帧缓存数量对视频低延时的影响；改进 Live555 开源库的代码等。
4. 为了验证与评价提出的系统，本文实现了一个测试环境。测试结果表明：低延时视频监控系统设计方案是可行的，系统符合低延时的评价标准。

关键词：实时监控系统，流媒体技术，低延时，直播

Design and Implementation of Low Latency Video Monitor System based on H.264 and H.265

Abstract

Recent years have seen the great impact of Internet technology, and streaming media applications have been rapidly promoted and popularized. Meanwhile, the streaming media system with video monitor as the core of demand has also received extensive attention and applications. Users want to use different devices to watch video surveillance in any environment, including locations, homes, workplaces, and social venues. Thus, how to send high bit rate video images under low delay becomes the frontier and challenge for the development of video surveillance technology. Till now, we use traditional streaming live broadcast architecture to tackle the problem. However, the architecture has a high latency for video transmission and does not meet the low latency basic requirements of video surveillance systems. We, therefore, find that it is very necessary to propose a new video monitor system architecture that allows it to have both high bitrate and low latency. On the basis of high concurrency, this paper proposes and implements a novel low-latency video monitor system based on H.264 and H.265. The system includes multiple modules such as server and client. Through this system, the user can watch the real-time video stream of most cameras supporting the RTSP protocol on the market with low latency from different platforms. The main contributions of this article are shown as follows:

1. Introduced the core technology of streaming media technology and digital video surveillance technology. By comparing and

combining the advantages of both, a new video surveillance system architecture method was proposed, taking into account the characteristics of high concurrency in streaming media systems. The low latency characteristics of video surveillance systems.

2. According to the proposed architecture scheme, a complete video surveillance system is studied and implemented. The system consists of multiple sub-modules, including video server, mobile client, desktop client, video streamer and database middleware. This article will design and implement the system modules one by one.
3. Based on the architecture, this paper studies a variety of special data structures and algorithms that meet the requirements and adds them to the system. The research contents include: Studying the key parts of H.264 and H.265 in low-latency video transmission, such as: Improved frame header judgment algorithm; Improved RingBuffer data structure; Exploring the impact of key frame buffer number on video low latency; Improve the Live555 open source library code and so on.
4. To verify and evaluate the system proposed in this paper, a test environment was implemented. The test results show that the system meets the low latency evaluation criteria.

Keywords: real-time monitoring system, multimedia, low latency, live streaming

第一章 绪论

1.1 研究背景及意义

目前，随着计算机自动化技术的不断发展以及网络传输速率的不断增加，视频监控系统正在迅速发展。当今社会人口众多，随着人们对安全性的要求的提高与经济条件的逐渐改善，用户对监控摄像头需求越来越高，其覆盖范围也越来越广，传统视频监控系统渐渐无法满足现在用户的需求[1]。

中国在线直播规模正在日益扩大，从图 1-1，2017 年上半年中国在线直播行业研究报告我们得出，2017 年用户规模为 3.92 亿，比较 2016 年增长了 26.5%，2019 年预计 4.95 亿，同时在线直播技术也日趋成熟。中国视频行业规模同样也与日俱增，如图 1-2，2017 年上半年中国视频监控行业研究报告的数据中，2017 年的视频行业规模达到将近 2000 亿，较 2016 年增长 13%，并预计在 2019 年突破 2500 亿。

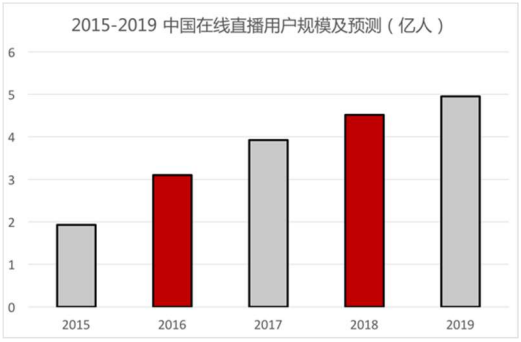

图 1-1 2017 年上半年中国在线直播行业用户规模及预测

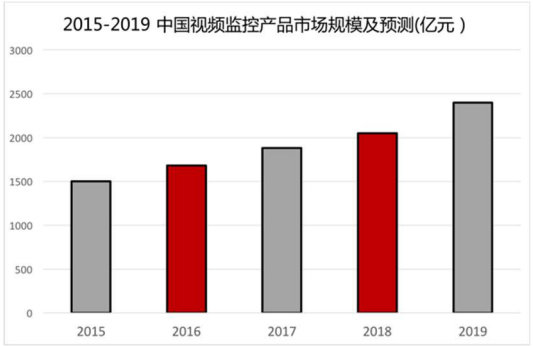

图 1-2 2017 年中国视频监控产品市场规模及预测

在线直播系统与视频监控系统之间的侧重点不同，但是可以相互视频监控系统经历了三个时代[2]。分别是模拟时代，半数字时代和现在的全数字时代。

第一代:模拟时代,视频以模拟方式采用同轴电缆进行传输,并由控制主机进行模拟处理。

第二代:半数字时代,视频以模拟方式采用同轴电缆进行传输,由多媒体控制主机或硬盘录像主机(DVR)进行数字处理与存储。

第三代:全数字时代,视频从前端图像采集设备输出时即为数字信号,并以网络为传输媒介,基于国际通用的 TCP/IP 协议,采用流媒体技术实现视频在网上的多路复用传输,并通过设在网上的网络虚拟(数字)矩阵控制主机,来实现对整个监控系统的指挥、调度、存储、控制等功能。此外报警、门禁、巡更等前端设备输出的数字信号也可转换后通过网络进行传输并在同一平台上进行管理和控制。近年来以视频监控为需求核心的流媒体系统[3]得到了广泛的应用。比如安防监控，家庭监控以及人流统计等[4]。对于视频监控系统而言， 流媒体转发服务器的用户服务人数，转发服务的时延，接入设备的种类数等决定了其优劣。而三个评价标准中前两个同时也是现代 Web 服务器的评价标准之一，所以重新设计一款视频监控，并将现代抗并发理念与转发理念融入其中将会得到优良的效果[5]。与此同时，数字视频监控系统也是目前是研究热点，特别是在家庭，公司等小型环境。文献[6] 介绍了一种数字视频监控系统,给出该系统的具体实现方案,并对其中的关键技术进行了深入分析.该系统结构合理、功能完善、运行可靠.通过测试白天和夜晚的监控效果图,该视频监控系统达到了最初技术要求,具有很好的实用性.文献[7]用流行的分布式 C / S 架构对系统进行设计，并采用流媒体技术，通过 IP 多播技术，RTP / RTCP 实时传送视频监控流。

	视频直播系统	实时视频监控系统
高并发	是	否
延时（ms）	~2000	<300
支持实时协议类型	RTMP, HTTP-FLV	RTSP
多平台支持	是	否
Codec	h.264,h.265	h.264,h.265

图 1-3 在线直播架构与视频监控架构属性对比图

从目前国内外的研究现状来看，视频监控系统主要分为嵌入式视频监控系统，集群视频监控系统和智能监控系统。然而目前在市面上并没有针对低延时的开源视频监控系统的实现，所以本次论文研究低延时的开源视频监控的实现是非常有意义的。目前在线直播系统架构已经趋于稳定，可以满足高并发，高用户量的需求，且能够在多平台运行观看视频流。但是由于大用户量架构以及技术原因，在线视频系统架构并没有解决低延时的需求，而低延时是评价视频监控系统优劣的重要标准。两个系统架构的对比如图 1-3 所示，本文旨在结合在线直播系统架构中的高并发优势以及在线视频系统架构的低延时优势，提出并实现一个稳定的低延时视频监控系统，并能投入使用。

1.2 研究现状

目前国内外对于视频监控系统研究有以下几个方面：

1. 嵌入式视频监控系统[43]。

由于智能家居的普及，视频监控系统渐渐走进普通家庭，很明显，以往的大型视频监控系统架构并不适合家庭使用，所以目前针对智能家居的嵌入式视频监控系统是研究热点之一。

文献[8]采用了海思Hi3518E+OV9712方案,设计并实现了无线网络高清实时监控系统。基于 B/S 架构,通过移植轻量级网络服务器 Boa,实现了客户端 Web 访问。文献[9]设计了一种基于 ARM 的移动视频监控系统,介绍嵌入式系统下视频压缩、编解码库的移植与应用,视频流媒体的传输与控制。文献[10]在 OMAP 平台上运用Gstreamer多媒体框架，实现了一个集视频采集、视频编码和视频传输的C/S结构网络视频监控系统。文献[11]提出了一种基于嵌入式Linux远程视频监控系统。系统以嵌入式 Linux 和控制器 S3C2440 为核心平台，通过嵌入式平台建立Web服务器Boa和视频服务器，利用基于TCP/IP的socket编程实现网络通信，将 USB 摄像头采集的图像数据进行压缩并通过网络传输传送到视频服务器客户端。文献[12]提出了一种以ARM9处理器为开发硬件平台和嵌入式Linux系统为软件开发环境的新方法,采用中星微 zc301 摄像头作为视频前端采集,利用TCP/IP 协议技术实现网络通信。文献[13]设计并实现了一个嵌入式的网络视频监控系统,在开发板上构建流媒体视频服务器,通过内核提供的应用程序接口采集

视频,经过编码、封装、打包后传输到网路上进行实时传输；在计算机上运行客户端,在网络上接收数据后解压缩进行显示和存储,以实现实时视频监控的目的。文献[14]针对视频监控系统在智能家居中对异常情况处理不够智能、响应速度较慢的问题,提出嵌入式视频监控端的方案。文献[15]提出一种基于嵌入式 Linux 的无线视频监控系统。整个系统目的是满足个人用户和小型场合的实时监控,主要突出两方面的优势:其一,安装和维护简单;其二,价格相对较低。文献[15]在嵌入式 Linux 的 ARM9 平台上实现网络视频监控,主要完成的是服务器端的各部分功能。

数字视频监控系统在手机上的研究也很多，文献[16]基于 Android 的开源特性,在系统软件平台之上设计开发视频监控系统。介绍系统的功能需求和 Android 应用开发过程中的技术要领。

2.大型集群视频监控系统。

随着大数据应用以及互联网+的发展与普及，城市的摄像头将被汇总入一个系统中，所以研究将视频监控系统与现在的大数据技术有机结合也是研究热点之一。

文献[17]开发了一个实时化、智能化的视频监控系统。系统核心以 ARM11 核心版 S3C6410 和高性能工业级 GSM/GPRS 模块 M10 为基础。文献[18]根据国家会展中心工程建设经验,分析了超大规模建筑的数字视频监控系统的安全需求,提出了 511 安全防护体系,据此对前端产品、网络架构和控制核心系统进行设计,并总结了工程实施经验,为类似的项目的建设提供参考。文献[19]对基于 LINUX 与 H.264 的安全视频监控系统进行了必要地阐述。

3.智能监控系统。随着人工智能，图像处理技术的日益完善，人们希望计算机智能监控最终能代替传统的肉眼观察监控。虽然目前智能监控系统只是起辅助作用，但是对一些特定场景来说效果卓越。

文献[20]提出一种基于 DM6437 的智能视频监控系统解决方案。在图像处理算法方面，将 ViBe 算法中采用第一帧图像建模和基于随机策略进行模型更新的思想用于改进传统码本算法的训练和更新阶段，提出一种基于随机码本的运动目标检测算法以提高检测效果。文献[21]设计了一套基于视觉传感技术的智能视频

监控系统,根据实际需要,在嵌入式平台上配合图像传感器和视觉跟踪 TLD 算法,实现了对塔机吊臂运动轨迹的追踪定位、判断和预警。测试结果表明:系统的检测率和正确率均能达到 95%以上。文献[22]提出了一种视频监控中的考生异常行为识别方法 ICanny-ABC-SVM.该算法从视频监控中提取考生行为图像,采用改进 Canny 算子对图像进行边缘检测;通过提取图像的不变矩特征,并将特征向量输入人工蜂群优化支持向量机中进行学习,构建考生行为分类器。文献[23]设计了基于 Hadoop 的分布式监控平台系统。通过对 Hadoop 技术的研究，着重对应用服务器的进程、站点及日志进行监控设计，利用云监控和大数据分析技术对采集的监控数据进行分析提供异常、报警等分析服务，为云平台用户提供稳定的云监控。

国外的研究点主要在与现有产业结合的智能视频监控。

文献[24]介绍了使用 3 个临床案例的自动视频监控系统的可行性结果，旨在评估涉及临床情况的受试者。方法和人群在装有用于日常生活活动的日常用品的观察室中进行研究。临床情景的总体目标是使参与者能够进行一组在观察室的情况下可以实现的日常任务。情况分为三个步骤，涵盖基本到更复杂的活动：（1）指导活动，（2）半指导活动，（3）无指导活动。使用由视觉组件和事件识别组件组成的自动视频监视系统来进行对研究的每个参与者的评估。

文献[25]实现了固定背景场景中视频监控系统的智能控制。 通过在线更新参考图像来消除由时间和日光的变化引起的对参考图像的影响。 为了减少计算复杂度和加快处理，提出了基于模糊方法的真实颜色空间的均匀压缩算法。

文献[26]介绍了一种新型的移动视频监控系统。接收机是具有 PHS（个人手持电话系统）卡的 PDA（个人数字助理）。发送器是基于 PC 的视频编码系统，其通过 ISDN-TA 连接到 ISDN 线路.实现了相机选择，远程相机控制和高分辨率快照功能。

文献[27]提出一种新颖的姿势分类系统，直接从视频序列分析人类运动。在该系统中，每个运动序列被转换成姿势序列。为了更好地表征序列中的姿势，我们将其三角形化为三角形网格，从中提取两个特征：骨架特征和质心上下文特征。第一特征用作对象的粗略表示，而第二特征用于导出更精细的描述。

文献[28]提出一个多摄像机视频监控的框架。 该框架包括三个阶段：检测，表示和识别。 检测阶段处理多源时空数据融合，以有效和可靠地从视频中提取运动轨迹。表示阶段总结原始轨迹数据，以构造运动事件的分级，不变和内容丰富的描述。最后，识别阶段处理数据描述符上的事件分类和识别。

1.3 论文主要研究内容

本文旨在编写一个完整的跨平台的低延时视频监控系统。该架构主要分为采集层，服务层和接受层。服务层的视频服务器收集来自采集层的 IP 摄像头，推流器，上游视频服务器的数据，与其他两者不同的是推流器采用推流的方式将数据给视频服务器，随后视频服务器将这些数据转发给位于接受层移动设备，桌面设备等等客户端以及下游视频服务器作为新一轮的服务层转发。本文系统将主要的设计点如下：

1. 设计并实现支持多种协议的视频监控服务器。底层使用开源库 libevent 或者直接使用 epoll 或者 icop 架构搭建，全篇代码使用 c++11 编写，减少平台相关性。
2. 设计并实现支持读取多种协议的视频监控客户端核心模块，并在 linux,windows,mac osx,android 平台实现观看视频监控。
3. 设计并实现基于视频监控客户端，并在 windows,android 上添加用户登录，显示用户可以查看的摄像头。
4. 设计并实现可推送至视频监控服务器的推流器。

同时，本文将着重研究以下内容。

1. H.264 与 H.265 的低延时特性：系统传输协议为 TCP 协议，其中视频传输格式为 H.264 或 H.265。所以需要研究 H.264 与 H.265 在低延时视频传输中的设置，包括 gop_size，即研究 I 帧的间隔对视频低延时的影响；研究 I 帧缓存数量对视频低延时的影响，等。研究 H.264 与 H.265 视频流相互转换时的参数设置，选择低延时下的最佳方案。
2. 分析时下流行的视频直播解决方案的优劣：时下流行的视频直播解决方案有，基于 RTMP 的视频直播解决方案，基于 UDP 的视频直播解决方案等。研究两个解决方案并指出各个解决方案的优缺点。

3. 研究基于 RTMP 协议的视频直播解决方案中的推流器的实现，并编写低延时视频监控系统的推流器：RTMP 架构的推流器思想是客户端主动连接服务器并将视频数据上传给服务器，推流器可以在客户端没有公网 ip 或客户端可以连接服务器，但是服务器无法主动连接到客户端的情况下使用，所以研究学习推流器中对重连机制，被动重传等行为是非常有必要的。
4. 研究 RTSP 协议并通过 live555 开源库解析 RTSP 协议，加入低延时视频监控服务器中。本文视频直播架构中的视频直播服务器将解析 RTSP 协议并使用 c++编写，故需研究调用 Live555 开源库解析 RTSP 协议。
5. 研究推流器与视频服务器之间的重连机制：推流器是在采集层设备在内网中，即视频服务器访问不到，然而内网设备能访问到视频服务器的情形下使用。推流器将采集位于内网的 IP 摄像头并将采集到的视频数据推送给视频服务器。所以视频服务器与推送端之间的重连机制将是值得研究的。

1.4 论文组织结构

本文主要由八章内容组成，其中：

第一章介绍了本文的研究背景以及研究意义，随后简单描述了国内外的研究现状，并针对研究现状描述了论文的主要研究内容以及研究方向。

第二章是介绍了论文的研究基础，主要涉及流媒体技术，流媒体传输协议，视频编码原理与标准，并简单介绍了几个流媒体开源项目。

第三章针对系统的业务分析进行功能分析，包括系统用例分析、系统功能包图。同时在业务分析和功能分析的基础上进行了功能数据分析，通过概念类图、类图缩略类图、实体类关系图（和数据库表结构）对数据库进行了详细分析与设计。

第四章是本文的核心内容之一，详细介绍了系统设计。介绍了系统开发环境，不用子系统使用了不同的开发环境开发，随后详细介绍了视频服务器子系统，数据库中间件子系统，桌面客户端的设计。

第五章介绍了系统几个重要的依赖库的编译与安装，以及各个子系统视频服务器，数据库中间件，桌面客户端的编译与运行。

第六章主要讲述了实现低延时视频监控的几个关键技术，其中涉及 H264，H265 的帧头判断算法，关键帧算法，并提出了带有 I 帧缓存的特殊 RingBuffer 数据结构，最后改进了 LIVE555 中接收数据的核心代码。

第七章本章搭建了一个真实环境进行了低延时测试，经过测试，画面与现实的延时在 300 毫秒以内，满足低延时要求。

第八章总结了本文所做的工作，同时对进一步的工作进行了展望。

第二章 论文研究基础

2.1 流媒体技术分析

流媒体是以流式传输的方式在网络上进行播放的媒体格式。其中流式传输是指将媒体数据进行压缩后通过网络分段发送的传输方式。目前流式传输主要有实时流式传输和顺序流式传输两种[2]。

顺序流式传输又称为渐进流传输[2]，主要采用 HTTP 协议[40,41,44,45,47]作为传输协议，将媒体数据分段传输到本地，本地下载缓存后再进行播放。该技术只需要最普通的 HTTP 服务器即可实现，并可直接接入 CDN 网络[2]。目前 HLS(HTTP Live Streaming)协议[42,46,48]主要使用该方法进行传输。

实时流式传输主要采用特殊的网络协议以实时性高为目的进行传输。普遍用于对实时性要求较高的服务，例如 IP 摄像头，网络直播等。该方式通常采用 RTSP[35],RTP[35]，RTMP[36]等实时协议进行数据交互。近年来使用 HTTP 协议作为载体的实时流式传输协议的应用渐渐增多[1,47]，特别是 HTTP-FLV[55]和 HTTP-H264[44,45]两种协议已经成为网络直播的主要直播方式之一。

2.2 流媒体传输协议简介

由流媒体技术分析得知，不同流式传输的传输协议不同。本文主要使用实时流式传输，支持该传输方式的协议有 RTSP[35]，RTMP，HTTP-FLV 等。

RTSP（实时流协议）是应用层协议，控制实时数据的传送 。RTSP 提供了一个可扩展框架，可以受控、按需传输实时数据。该实时数据包括现场数据与存储的数据。本协议可控制多个数据发送会话，并提供了多种传输路径，例如 TCP，UDP 等。

RTMP 协议的全称是 Real Time Messaging Protocol,即实时消息传送协议，由 Adobe 公司提出。RTMP 本身并没有限制传输格式，所以既可以传输视频与音频，也可传输控制指令甚至是自定义聊天信息。RTMP 协议使用 TCP 协议作为其传输层协议，故丢包率小，用户体验度高。雷[39]指出，RTMP 的缺点在于对服务器性能和传输带宽要求较高。不过随着计算机硬件与新型处理机制例如协程，异步事件机制的发展，这些缺点将愈加微不足道。RTMP 协议结构如图 2-1 所示。

图 2-1 RTMP 协议结构图

HTTP-FLV 与 RTMP 协议实际上传输数据一样，数据都是 FLV 文件的 tag。HTTP-FLV 可做看做为一个无限大的 HTTP 流的文件，与之相比 RTMP 只能用于直播。但是 HTTP-FLV 可以在 80 端口进行 HTTP 通信，协议穿透性强。RTMP 内部结构如图 2-2 所示。HTTP-FLV 在 FLV 协议外面封装了一个 HTTP 协议。目前，HTTP-FLV 与 RTMP 协议是如今直播平台主选的直播方式。

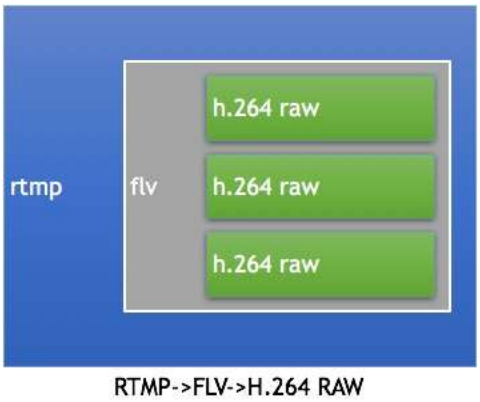

图 2-2 RTMP 协议与 FLV 封装的联系

2.3 视频编码原理与标准

视频压缩编码目的在于去除数字化视频数据中的冗余信息，从而缓解视频文件的存储与传输时的压力。视频的压缩研究始于 20 世纪 50 年代，出现了针对图像信息冗余的哈夫曼码以及基于像素的差分预测编码等，这时期的压缩技术都属于无损压缩。从 70 年代开始出现了有损压缩，例如运动补偿预测、离散余弦变换（DCT）等，逐渐形成了基于预测变换的混合编码框架。与此同时，ISO（标准化组织）/IEC MPEG 工作组与 ITU-T VCEG（国际电信联盟）工作组在此基础上制定了 MPEG-1、H.261 编码标准，并在后续推出了 MPEG-2、MPEG-4、H.263 标准。

21 世纪初，ITU-T/ISO/IEC（国际电信联盟）视频编码联合协作小组 JCT-VC 提出了第二代视频编码标准 H.264/AVC[51]（Advanced Video Coding）。其中混合编码框架如图 2-3 所示：

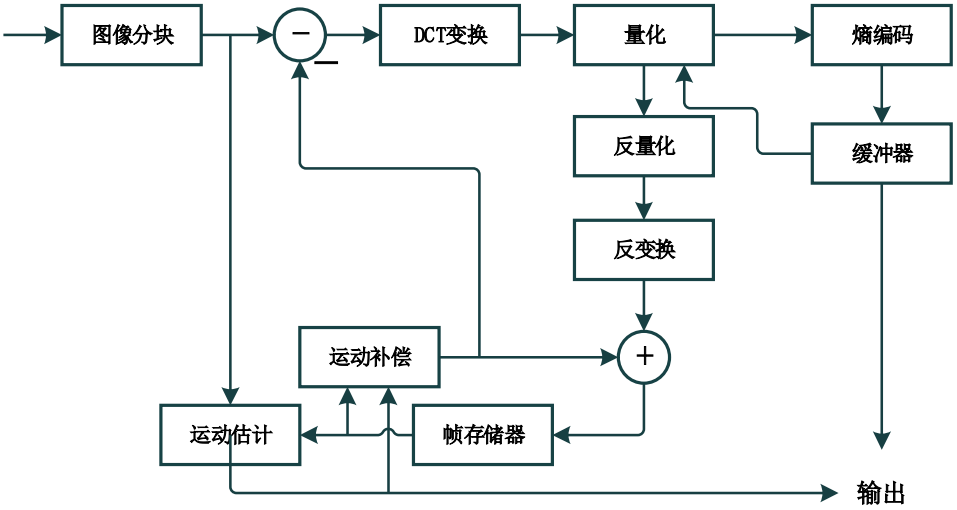

图 2-3 H264 混合编码框架图

首先将输入图像分块，分块后的图像块与经过运动补偿的预测图像 A 相减后得到差值图像 X。然后对差值图像 X 进行 DCT 变换和量化，量化后的数据可以输入到熵编码器进行编码，然后编码完成后将码流保存在缓存器中，等待输出；量化的数据也可以进行反量化和反变换，然后与运动补偿输出的图像 A 相加得到新的预测图像 X'，最终预测图像块 X'将被送至帧存储器中进行后续操作。

但是随着数字视频应用的快速发展，为了追求更为极致的用户体验，视频的空间分辨率经历了从 CIF（352×288）到标清（480p，720×576）再到高清（720p,1080p）甚至超高清 4k,8k 的迅猛提升。视频编码器芯片化，并行化的需求日益加剧。面对新的应用需求，传统视频编码技术已经无法满足。

为此，视频编码联合工作组 JCT-VC 于 2013 年 1 月正式颁布了 H.265/HEVC 视频编码标准草案。HEVC 的目标就是提高视频编码效率，在相同的图像质量下，压缩率比 H.264/AVC high profile 提升 50%，同时支持各类规格的视频，并在计算复杂度、压缩率、鲁棒性和处理延时之间妥善折中处理[12]。

HEVC[50]仍然采用了基于运动补偿的混合编码框架。编码原理和基本结构上与之前的标准基本一致，即通过帧内/帧间预测消除时间域和空间域的相关性；对

运动补偿后的预测残差进行 DCT 变换和量化消除空间相关性；最后统计冗余和量化噪声则通过自适应熵编码和环路滤波技术进行消除。HEVC 编码框架如图 2-2 所示：

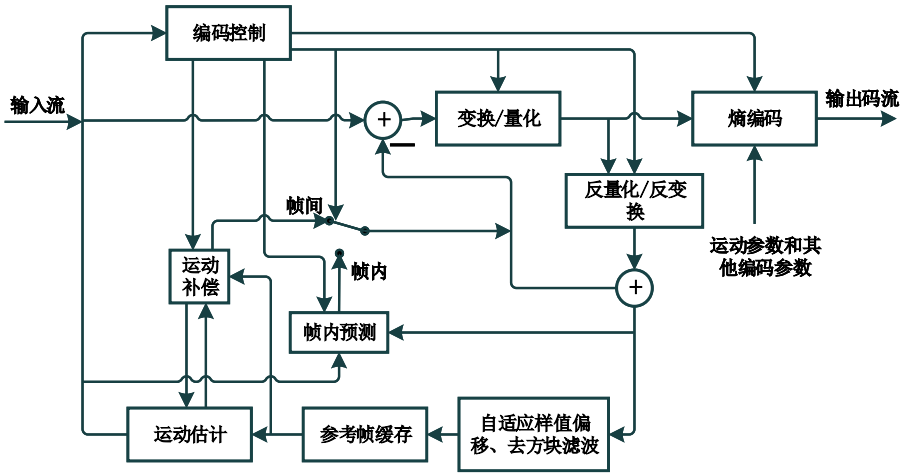

图 2-4 H265 混合编码框架图

2.4 流媒体开源项目简介

2.4.1 Live555 开源流媒体解决方案

Live555[34]是一个跨平台开源流媒体解决方案，使用标准 c++编写完成，没有使用 STL 库。它实现了对 RTSP（支持 HTTP，TCP，UDP 等多种方式传输）、RTP/RTCP 等标准流媒体传输协议的支持. 而且支持多种音视频编码格式的多媒体数据的流化、接收和处理, 包括 H.265，H.264，MPEG，MJPEG 等视频和多种音频编码格式。由于 Live555 拥有良好的结构化设计, 所以很容易扩展对其他多媒体格式的支持。Live555 主要由四大框架组成，分别是: BasicUsageEnvironment, groupsock, liveMedia, UsageEnvironment。

UsageEnvironment 模块是对系统环境的抽象，要用于消息的输入输出和用户交互功能。其中还包括一个重要模块 TaskScheduler。此模块完整实现了异步处理事件，当然，一切都是单线程操作的。Live555 通过精妙的代码完成了类似异步的编程模式，单线程下通过不停的任务切换做到多线程效果。

BasicUsageEnvironment 模块是 UsageEnvironment 的一个控制台应用的实现。它针对控制台的输入输出和信号响应进行具体实现。

GroupSock 模块用于实现数据包的发送和接收。其底层是一个循环 select 判定的大循环，将收到的报文通过 sock 编号分发，所以他支持单线程下的多播功能，同时，也支持单线程的单播功能。

LiveMedia 模块是 Live555 重要的模块。该模块声明了一个抽象类 Medium，其他所有类都派生自该类。

2.4.2 FFMPEG 开源流媒体解码库

FFmpeg[33]是一套 C 语言编写的跨平台开源流媒体编解码库，采用 LGPL 或 GPL 许可证发布。它提供了几乎完整的流媒体解决方案，包括编解码，裁剪，滤镜，转换格式等。用户只需调用接口即可完成原本复杂的编解码过程，所以该库被大量知名播放器使用，包括 POTPLAYER,VLC,BIGPOT 等。

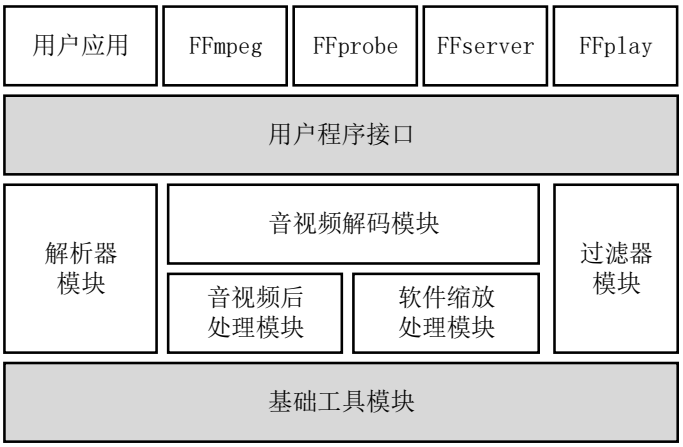

图 2-5 FFMPEG 开源库框架图

FFmpeg 同时给用户提供了四个自带的可执行程序，分别是进行格式转换的 FFmpeg，简单的万能视频播放器 FFplay，视频格式分析工具 FFprobe，以及流媒体转发服务器 FFserver。这些程序都是开源的。FFmpeg 开源库的框图如图 2-5 所示。

本文着重使用了 FFmpeg 的音视频解码模块，并针对解码模块做了深入的研究。解码的详细步骤如下：第一步，初始化 FFmpeg，初始化 avcodec 类。第二步，找到 h264 或者 h265 的解码器。第三步，打开解码器，并初始化帧包。最后，将帧包填满数据，并调用核心解码函数，解码函数输出结果。详细流程图如图 2-6 所示。

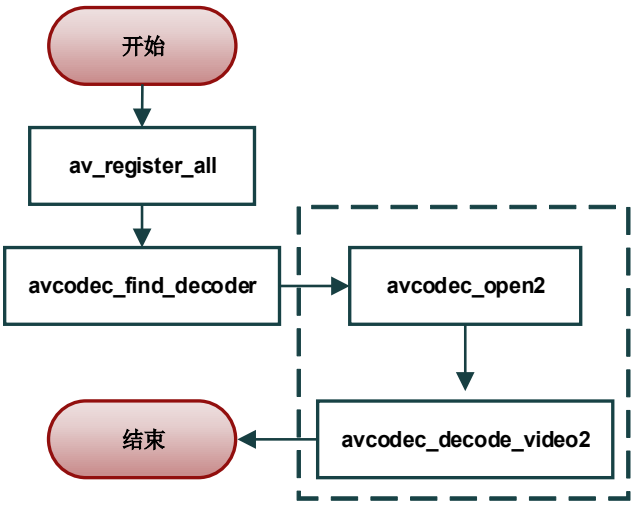

图 2-6 FFmpeg 解码流程图

2.5 低延时的定义

延时在视频监控中的定义为，从视频画面从采集到编码，传输，解码，最后在屏幕上显示所用的时间。延时的定义如图 2-7 所示。其中根据延时大小，可以将延时分类为伪实时，准实时和真实时三种。

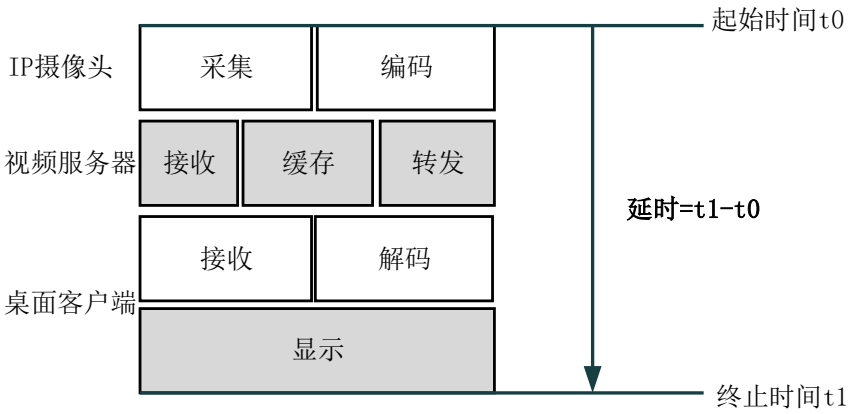

图 2-7 延时的定义

伪实时：视频消费延时超过 3 秒，单向观看实时。

准实时： 视频消费延时 1 ~ 3 秒，能进行双方互动但互动有障碍。

真实时：视频消费延时 < 1 秒，平均 500[54][55] 毫秒。

结合 WebRTC[55]中提到的对低延时的定义，本文最终选择 300 毫秒作为低延时标准。

2.6 本章小结

本章主要对本文中所有涉及到的相关技术进行了概要简述。首先分析了流媒体技术，介绍了市面上的两种流媒体传输方式。随后着重介绍了实时流方式中使用的流媒体协议 RTSP，RTMP 和 HTTP-FLV。最后，介绍了文中使用频繁的两个开源流媒体项目 Live555 和 FFmpeg，并简述了调用 FFmpeg 进行解码的步骤。

第三章 系统规划与结构设计

3.1 需求分析

低延时视频监控系统的总体要求是：针对市面上支持 RTSP 的普通监控摄像头，系统能通过 RTSP 协议获取到摄像头的视频流信息，并且给予用户归档权限，使用户能更加方便的观看与记录摄像头的视频流信息。

本系统对于软件性能的需求主要从界面操作，可用性，鲁棒性，执行效率，以及扩展性这五个方面提出，以下来简单分析介绍：

- 界面操作方面，本系统的视频客户端子系统应当做到界面简洁美观大方，吸引人；在显示信息上应该符合用户的认知特征以及工作经验，同时减少计算机软件相关术语，界面简洁，不需要出现多余的控件引起误操作。
- 软件可用性方面，本系统的视频服务器子系统应尽可能简单易用，操作难度低。
- 软件鲁棒性方面，本系统应有详细的信息提示，包括出错信息、提示信息等；本系统应尽量减少错误发生几率；当不可避免的错误发生时，应尽可能给出解决问题的提示信息；应有应对软件硬件错误、故障的恢复方案。
- 软件执行效率方面，本系统应快速稳定执行；网络等硬件正常工作情况下，一次典型的数据维护过程应在 2 秒之内完成；网络等硬件正常工作情况下，一次典型查询过程应在 3 秒之内给出结果。
- 软件扩展性方面，本系统应与相关的软件良好配合使用，并且能够较为方便的添加新的功能和模块。

3.2 系统组成

本系统根据需求主要分为三大核心模块，数据储存模块，监控视频流媒体服务器模块以及监控视频查看客户端模块。

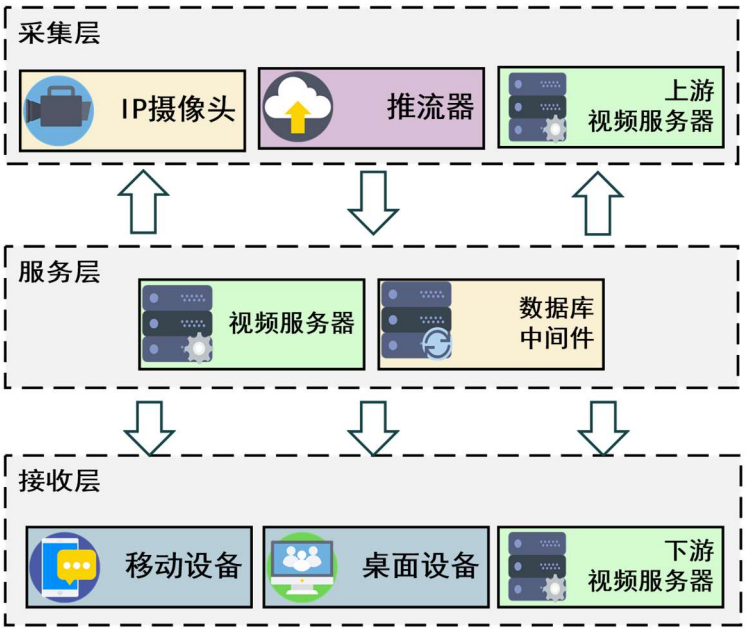

图 3-1 低延时视频监控系统组成

1. **数据储存模块：**本模块的主要功能是存取数据，包括摄像头详细设置参数，用户参数，用户可以观看到的摄像头列表，多服务器列表，多集迭代目录表。并使用 XML 技术作为协议栈与外界交互，外界通过该模块取得数据并存储。
2. **监控视频流媒体服务器模块：**本模块的主要功能是读取摄像头 RTSP 数据流，将 RTSP 数据流中的 H.264 与 H.265 格式的视频数据转码为自定义协议数据流，并转发给客户端，客户端通过访问监控视频流媒体服务器模块获取摄像头数据流。在满足基本功能的同时还有一些基本智能功能例如移动侦测。
3. **监控视频查看客户端模块：**本模块的主要功能是让用户编辑和查看监控视频流媒体服务器转发的摄像头视频数据流。客户端模块又详细分为 Windows 客户端核心查看模块，Android 客户端查看模块，OSX 客户端查看模块，以及推流客户端。系统应用示意图如图 3-1 所示。

3.3 软件架构设计

低延时视频监控系统是多线程，多任务的。主要由五个程序组成，分别是推流器，视频服务器，移动客户端，桌面客户端，数据库中间件，如图 3-2 所示。

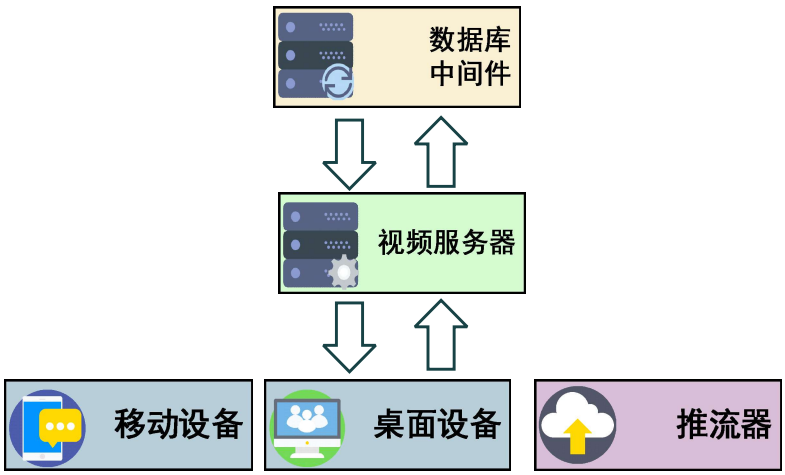

图 3-2 低延时视频架构设计

视频服务器主要由接收模块，缓存模块，报文生成模块和转发模块四个部分组成，这些模块之间互相有调用被调用的关系。数据库中间件主要由转发模块和数据库查询更改模块组成，模块与模块之间有被调用关系。桌面客户端主要由桌面 UI 模块和视频显示核心模块组成，UI 层调用核心模块。推流器主要由前端采集模块，缓存模块，编码模块和推送模块组成。

3.4 软件模块的划分与设计

低延时视频监控系统在软件功能上包括多个功能模块，而这些模块可以互相不影响，不干扰各自完成一项功能，并可以在不更改其他模块的情况下单独对某个模块进行进一步的改进。各个程序的各个模块的介绍如下。

3.4.1 视频服务器

视频服务器主要由接收模块，缓存模块，数据库模块，报文生成模块和转发模块五个部分组成。

接收模块

该模块主要完成接收传来的视频数据，包括 ip 摄像头，上游视频服务器与推流器，并将视频数据拆包为纯视频流格式并存入缓存模块，每一个接收模块将占用一个线程资源。

缓存模块

该模块主要接收来自接收模块的纯视频流数据并将它保存入队列中，缓存模块采用生产者消费者模式，并使用特殊的数据结构保存。报文生成模块将作为消费者读取缓存模块的数据。

数据库模块

该模块主要与数据库中间件进行通信，两者使用自定义报文进行通信。模块用于查询数据库中的内容并反馈给接收模块。

报文生成模块

该模块主要用于将纯视频流数据封装为其他视频流格式，目前包括 http-flv 格式，自定义报文格式与 tcp-h264/h265 格式。报文生成完毕后将被转发模块进行调用并转发给其他请求端。

转发模块

该模块主要处理请求端请求的视频并将视频发送给请求端，视频将通过读取报文生成模块生成的视频流获取。

3.4.2 数据库中间件

数据库中间件主要由转发模块和发送模块两个部分组成。

转发模块

该模块主要用于接收来自视频服务器发送的查询或者更改报文，同时将查询模块查询出的结果封装报文后发送给请求端。

数据库查询更改模块

该模块主要查询模块接收的报文，通过 sqlite 内部函数得到返回值后调用发送模块。

3.4.3 桌面客户端

桌面客户端主要由桌面 UI 模块和视频显示核心模块组成，UI 层调用核心模块。

桌面 UI 模块

该模块主要完成界面的初始化，与数据库中间件连接，登录，显示视频列表，与视频核心播放模块联动播放实时视频流。

视频显示核心模块

该模块主要用于播放实时视频流，运行时嵌入 UI 模块。

获取设备 ID 流程

桌面客户端将主动连接视频服务器并提供想要连接的设备 ID。详细步骤如图 3-6 所示。

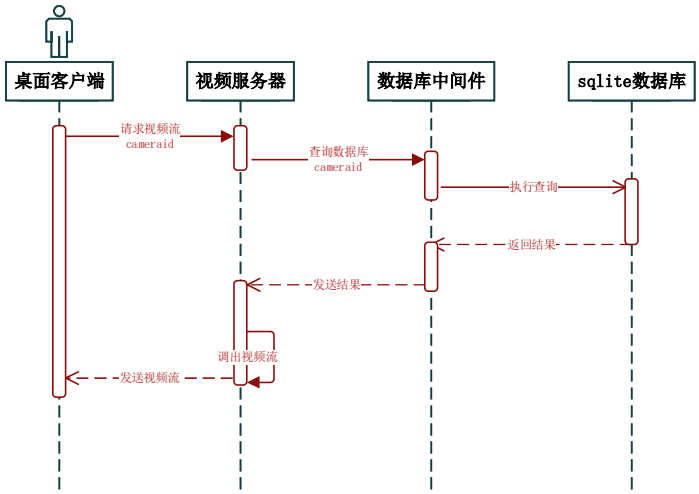

图 3-6 桌面客户端获取设备 ID 流程

如图 3-6 所示，桌面客户端为客户端发起者，在发送视频内部编号 cameraid 给服务器后，视频服务器发送查询 cameraid 消息给数据库中间件，数据库中间件异步查询 sqlite 数据库，并将结果逐层传回视频服务器。视频服务器根据返回的视频编号详细信息调度其他的流媒体模块，并将视频数据包推还给桌面客户端。

客户端用例分析

根据总体用例分析，针对用户以及管理员制作各自的子用户分析。用户需要拥有获取摄像头 ID，获取数据库信息，修改数据库信息等内容，管理员需要拥有查看用户在线状态，查看当前服务器状态等用例内容，如图 3-7 所示。在此将用户用例进行解析。

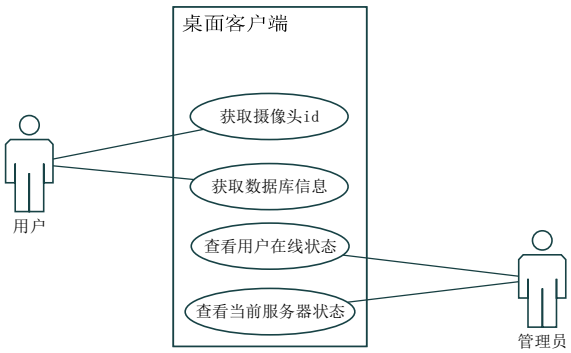

图 3-7 桌面客户端总体用例分析

描述项	说明
名称	用户获取摄像头
描述	描述了用户获取摄像头 id 的流媒体的过程
参与者	用户
前置条件	必须是用户角色已经登录到了系统
后置条件	获取摄像头 id 对应的实时视频流
基本操作流	HTTP-GET：摄像头 id 号

表 3-1 获取摄像头用例描述

描述项	说明
名称	获取数据库信息
描述	描述了客户端获取数据库信息的过程
参与者	用户

前置条件	必须是用户角色登录系统
后置条件	获取后端数据库内信息
基本操作流	（1）发送数据库申请报文 （2）等待回收报文。

表 3-2 获取数据库信息用例描述

UI 模块

该模块主要完成界面的初始化，与数据库中间件连接，登录，显示视频列表，与视频核心播放模块联动播放实时视频流。

JNI 模块

该模块主要用于播放实时视频流，由 UI 模块调用执行

3.4.4 推流器

推流器主要由前端采集模块，缓存模块，编码模块和推送模块组成。如图 3-9 所示。

前端采集模块

该模块主要用于采集前端摄像头，桌面与文件中的数据，并将数据转为 YUV 格式存入缓存模块中。

缓存模块

该模块主要接收来自前端采集模块的数据并将它保存如队列中，缓存模块采用生产者消费者模式，并使用特殊的数据结构保存。编码模块将作为消费者读取缓存模块的数据。

编码模块

该模块主要将缓存模块中的数据编码为 h.264 格式并保存，让推送模块负责推送。

推送模块

该模块主要负责将 H264 格式数据推送到请求端。

3.5 数据库逻辑结构设计

本系统数据库中间件部分中的数据库部分将使用轻量级数据库 sqlite。数据库设计是把现实世界中一定范围内存在的应用处理和数据抽象成一个数据库的具体结构的过程[36]。具体的讲，就是对于一个给定的应用环境，提供一个确定最优数据模型与处理模式的逻辑设计，以及一个确定数据库存储结构与存取方法的物理设计，建立能反映现实世界信息和信息联系，满足用户要求，能被某个数据库管理系统所接受，同时能实现系统目标并有效存取数据的数据库。

本系统数据库的设计尽量考虑了将来程序变更的灵活性，并满足第三范式要求。本系统的需求活动针对着系统中涉及的各类数据做了详细的调研，根据功能需求分析，在 sqlite 中建立了一个名字为 cyclops 的数据库，并且创建了多个数据表，并且对象之间存在依属关系。

3.5.1 摄像头表

用来保存摄像头的所有可用信息，包括摄像头 ID 号，以及其关联的视频服务器 ID 号，端口号，摄像头名称，摄像头短名称。摄像头描述，摄像头的云台控制类别，摄像头云台的串口控制号，声音 ID，流媒体化的 IP 号，若是硬盘化后的硬盘位置，摄像头总数，真实的摄像头 ID 号（用来区别伪装的摄像头 ID 号），如表 3-3 所示。

Name	Declared Type	Type	Size	Not Null	Primary Key
CameraID	integer	integer	0	TRUE	TRUE
VideoServerID	integer	integer	0	TRUE	TRUE
Port	integer	integer	0	TRUE	FALSE
Name	varchar(255)	varchar	255	FALSE	FALSE
SubName	varchar(50)	varchar	50	FALSE	FALSE
Description	varchar(500)	varchar	500	FALSE	FALSE

PTZControl	integer	integer	0	FALSE	FALSE
SerialPort	integer	integer	0	FALSE	FALSE
Audio	integer	integer	0	FALSE	FALSE
MultiCastIP	varchar(50)	varchar	50	FALSE	FALSE
defaultdir	varchar(100)	varchar	100	FALSE	FALSE
CameraNum	integer	integer	0	FALSE	FALSE
RelCameraID	integer	integer	0	FALSE	FALSE

表 3-3 摄像头表

3.5.2 多服务器表

用来保存所有服务器的连接方法，包括服务器 ID，服务器名称，内网 IP，内网端口号，外网 IP，外网端口号，限制大小，记录磁盘，自动删除等。如表 3-4 所示。

Name	Declared	Type	Size	Not Null	Primary Key
ServerID	integer	integer	0	TRUE	TRUE
ServerName	varchar(100)	varchar	100	FALSE	FALSE
InternalIP	varchar(50)	varchar	50	FALSE	FALSE
InternalPort	integer	integer	0	FALSE	FALSE
ExternalIP	varchar(50)	varchar	50	FALSE	FALSE
ExternalPort	integer	integer	0	FALSE	FALSE
DeviceLimitCode	varchar(50)	varchar	50	FALSE	FALSE
RecordFolder	varchar(100)	varchar	100	FALSE	FALSE
AutoDelete	integer	integer	0	FALSE	FALSE
DeviceLimit	integer	integer	0	FALSE	FALSE

表 3-4 多服务器表

3.5.3 用户摄像头表

用来灵活切换和绑定实际摄像头 ID 和组 ID，包括用户摄像头 ID，组 ID，摄像头 ID，摄像头云台类型，如表 3-5 所示

Name	Declared Type	Type	Size	Not Null	Primary Key
UserCameraID	integer	integer	0	TRUE	TRUE
GroupID	integer	integer	0	TRUE	TRUE
CameraID	integer	integer	0	TRUE	TRUE
CameraPTZ	integer	integer	0	FALSE	FALSE

表 3-5 用户摄像头表

3.5.4 用户组表

用来存取树形结构用户组与摄像头数据之间的联系，包括用户组 ID，用户 ID，姓名，是否私有，树结构类型，描述，主要 ID，上层组 ID 等。如表 3-6 所示。

Name	Declared Type	Type	Size	Not Null	Primary Key
GroupID	integer	integer	0	TRUE	TRUE
UserID	integer	integer	0	TRUE	TRUE
Name	varchar(50)	varchar	50	TRUE	TRUE
private	integer	integer	0	FALSE	FALSE
type	integer	integer	0	FALSE	FALSE
Description	varchar(50)	varchar	50	FALSE	FALSE
GroupDomainID	integer	integer	0	FALSE	FALSE
topGROUPID	integer	integer	0	TRUE	TRUE
topGROUPID1	integer	integer	0	TRUE	TRUE

表 3-6 用户组表

3.5.5 多级迭代目录表

用来制作三级以上的多级目录新增的表，包括置顶用户组 ID,置顶用户组名称，置顶描述，用户 ID，用户组范围 ID，用户组上层组 ID 等，如表 3-7 所示：

Name	Declared Type	Type	Size	Not Null	Primary Key
topGroupID	integer	integer	0	TRUE	TRUE
topGroupName	varchar(20)	varchar	20	TRUE	FALSE
topDescription	varchar(50)	varchar	50	FALSE	FALSE
UserID	integer	integer	0	TRUE	TRUE
GroupDomainID	integer	integer	0	FALSE	FALSE
topGroupParent	integer	integer	0	TRUE	TRUE

表 3-7 多级迭代目录表

3.5.6 用户信息表

用来记录用户出示信息，包括用户 ID，用户名称，用户密码，用户真实名称，权限，以及用户描述等，如表 3-8 所示

Name	Declared Type	Type	Size	Not Null	Primary Key
UserID	integer	integer	0	TRUE	TRUE
UserName	varchar(50)	varchar	50	TRUE	FALSE
UserPwd	varchar(50)	varchar	50	FALSE	FALSE
RealName	varchar(50)	varchar	50	FALSE	FALSE
FullControl	smallint	smallint	0	FALSE	FALSE
Description	varchar(300)	varchar	300	FALSE	FALSE

表 3-8 用户信息表

3.5.7 摄像头数据详细表

用来存储某个种类的摄像头的连接方式或者连接手段，包括摄像头类型ID，摄像头类型名称，流媒体类型，端口数目，连接字符串最小化，连接字符串普通化，连接字符串最大化，声音地址，云台信息等。如表 3-9 所示：

Name	Declared Type	Type	Size	Not Null	Primary Key
VSTypeID	integer	integer	0	TRUE	TRUE
VSType	varchar(50)	varchar	50	TRUE	FALSE
ocxType	integer	integer	0	FALSE	FALSE
protocol	integer	integer	0	FALSE	FALSE
StreamType	integer	integer	0	FALSE	FALSE
PortCount	integer	integer	0	FALSE	FALSE
HalfSize	varchar(100)	varchar	100	FALSE	FALSE
FullSize	varchar(100)	varchar	100	FALSE	FALSE
HugeSize	varchar(100)	varchar	100	FALSE	FALSE
SoundUrl	varchar(100)	varchar	100	FALSE	FALSE
RecordSoundURL	varchar(100)	varchar	100	FALSE	FALSE
AudioParameterURL	varchar(100)	varchar	100	FALSE	FALSE
CanCDM	integer	integer	0	FALSE	FALSE

表 3-9 摄像头数据详细表

3.6 系统开发环境

本低延时视频监控系统主要采用 C/S（Client/Server）架构，并且根据第三章分析出的实际系统需求进行适当的改良以及细化。最终的目标是设计出一套高内聚，低耦合的软件系统架构。

本系统中的视频监控服务器涉及到一系列试试数据传输，以及服务器转发能力，负载调节的问题。为此，该服务器的转发能力以及视频数据协议的优劣对于本系统的性能有非常重要的影响。

用户对于本系统软件界面提出了具体要求，需要简约的风格，通过尽量少的按钮去做更多的事情，与此同时还要求界面直观，易学易用等。因此本系统的界面设计经过了精心打磨。

优秀的系统开发环境是优秀的系统的前提，本系统开发环境较为复杂。本节将本低延时视频监控系统中各个部分的系统开发环境。

视频监控服务器，推流器，视频显示核心模块，数据库中间件的开发主要在 windows 下的 visual studio community 2013 完成，系统开发环境图如 4-1 所示。考虑到跨平台移植特性，软件编程语言为 c++11。c++11 丰富了 std 库，使编写服务器程序变得更加容易，并加入 auto,foreach 等关键词，使开发者不需要在选择 std 的迭代器类型中迷失。在 windows 完成编译之后，使用一份代码在 linux 下通过 gcc,make 完成编译，其中自行编写 configure 与 makefile。在 linux 下调试工具使用 gdb。

桌面开发环境主要在 windows 下的 visual studio community 2013 完成，系统开发环境图如 4-2 所示。软件编程语言为 c# .net framework 4.0。.net framework 4.0 新增了 lambda,continuations，动态调度，linq 等功能，使开发者更轻松开发桌面客户端。

3.7 本章小结

本章首先分析了低延时视频监控系统的需求，并根据需求将系统分为三大模块，随后以三大模块为基准进行了系统划分，并将系统划分为四个子系统。本章大致介绍了每个子系统下的每个模块的大致作用。在桌面客户端子系统的分析阶段着重分析了获取 ID 过程，并简单的进行了用例分析。随后介绍了数据库逻辑结构设计，最后简单描述了系统的开发环境。

第四章 系统详细设计

本章将根据第三章划分的四大子系统逐一进行详细设计与分析。

4.1 服务器详细设计

服务器主要由接收模块，缓存模块，数据库模块，报文生成模块与转发模块组成，服务器系统运行图如图 4-1 所示。用户发送观看视频请求给服务器的转发模块，随后转发模块通过数据库模块查询对应的 ID 编号，如果没有则通知接收模块创建新对象并开启拉流。服务器将视频流从接收模块存入缓存模块。随后服务器将视频流通过报文生成模块转成用户希望的视频流格式，最后通过转发模块将数据发回用户。

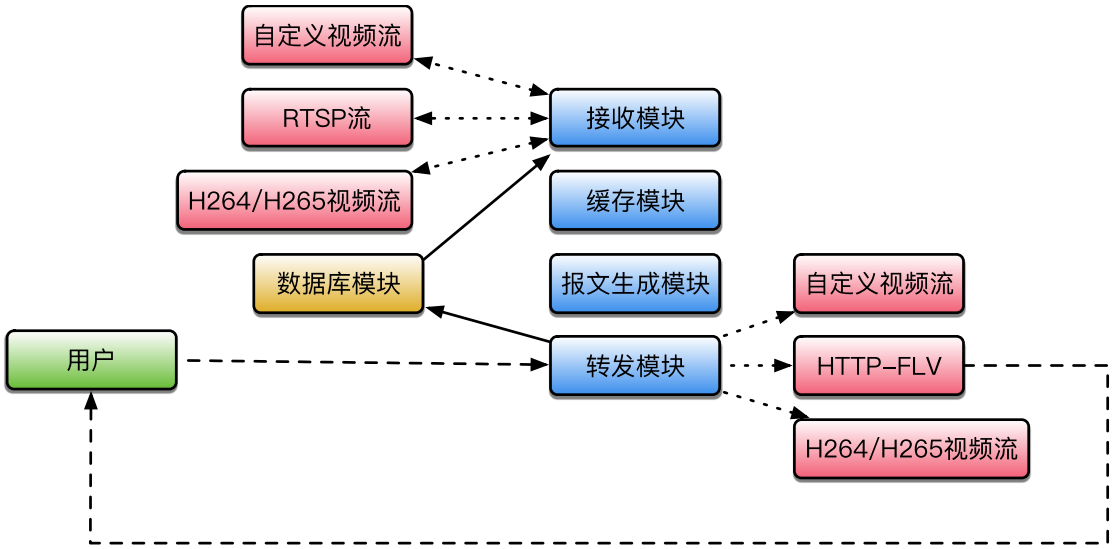

图 4-1 服务器系统运行图

4.1.1 接收模块

基于开发需求，接收的模块将接收来自外部数据并解析出视频流，保存到缓存模块中。接收模块能够接收 RTSP 流，自定义数据流，h264 文件数据流，FFmpeg 解码后的视频流与推流器发送来的流。

本模块创建了多个类用于接收不同的流。其中，MythLive555Decoder 类用于解析 RTSP 流，mythStreamDecoder 类用于解析自定义数据流，mythH264Decoder 类用于解析 h264 文件数据流，mythFFmpegDecoder 类用于解

析经过 FFmpeg 解码后的视频流，mythProxyDecoder 类用于解析推流器发送来的视频流。

mythVirtualDecoder 类：

该类是所有流媒体输出的基类，该类的类图如图 4-9 所示。该类只有两个接口 start 和 stop，所有其余的流媒体类可在他基础上做开发。同时，他继承了 mythListFactory，该类基于工厂模式开发，是缓存模块的核心类。作为基类他可计算单位时间内的数据流大小，随后将数据保存在内部变量 m_count 中供继承类调用。

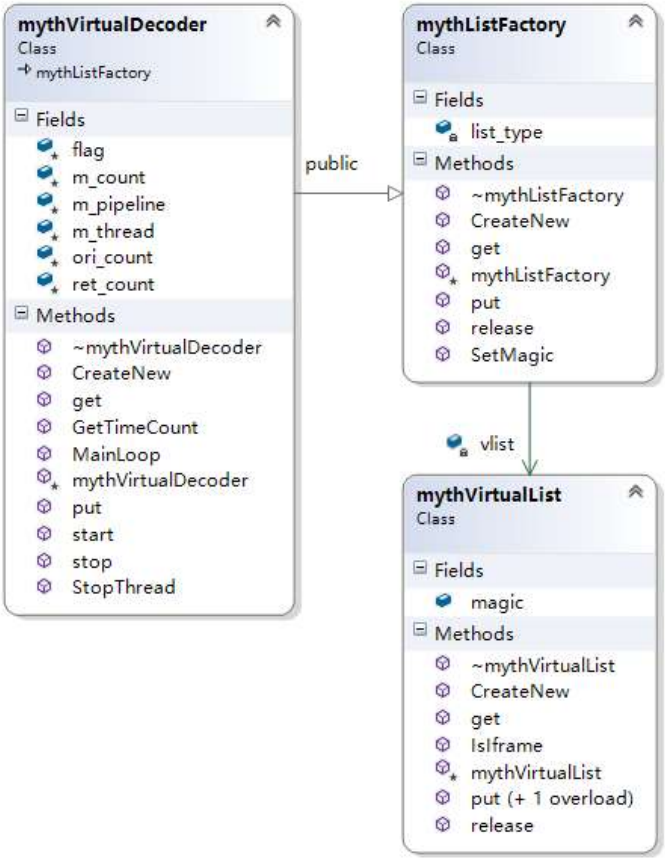

图 4-2 mythVirtualDecoder 类图

myhStreamDecoder 类：

该类继承于 mythVirtualDecoder，重写了 MainLoop 和 stop，主要用于 HTTP 流格式的收取并且将其保存如继承的 mythListFactory 类中。该类如图 4-2 所示。

MythStreamDecoder 在收取 HTTP 流时会调用 mythSocket 类，mythSocket 类图如图 4-3 所示，该类对 HTTP 协议有了更多优化，包括一个全指针算法 socket_ReceiveDataLn2，用于解析自定义报文协议，其算法将在关键算法中单独讲解。MythSocket 类拥有 native,libevent,libcurl 三种底层驱动模式。其中 native 模式效率最高，默认编译模式下将使用 native 模式。

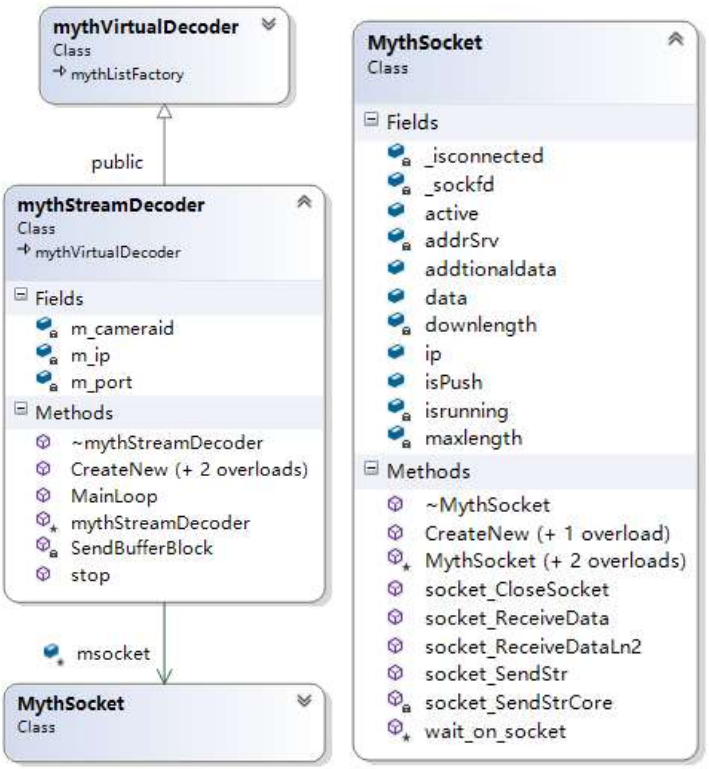

图 4-3 MythStreamDecoder 类图

myhLive555Decoder 类：

该类继承于 mythVirtualDecoder，是接收 RTSP 报文的核心类，该类的作用是接收 RTSP 数据流并将他存入内部缓存模块中。该模块核心是基于开源库 live555 深度二次开发的 StreamSink 类，其详细类图 4-4 所示。

Live555decoder 内部调用了 live555 的封装总类 myhthRTSP,其中所有 RTSP 的行为和报文行为，而获取报文行为都以静态存在，动态调用的方法实现，效率和实用性大大增强。

RTSP 协议介绍在第二章中有具体描述。RTSP 协议请求流流程图如图 4-5 所示。通过 createNew 函数将 RTSPlink,username,password 传入实例化后的对象后，mythRTSP 将自动进行 RTSP 的获取流程，并返回一个封装的 h264 流类 H264VideoStreamSink。H264VideoStreamSink 类图如图 4-6 所示。

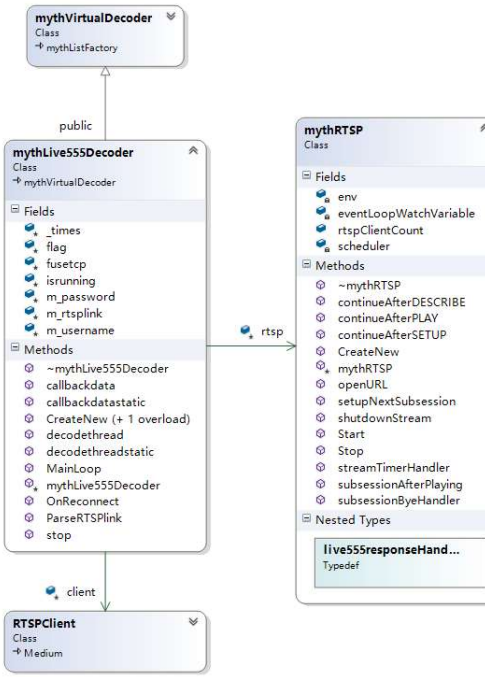

图 4-4 mythLive555Decoder 类图

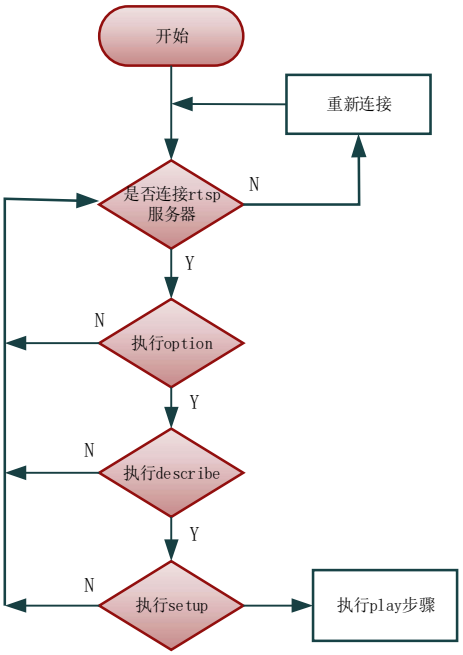

图 4-5 RTSP 协议流程图

类继承缓存模块，采取异步存取方式，将 live555 中回调的数据收集，并保存在缓存模块供读取。在所有类中，StreamSink，H264or5VideoStreamSink，H264VideoStreamSink 为新增类，其余为 live555 本身就有的类。StreamSink 继承自 MediaSink,主要继承 MediaSink 中的几个回调函数，分别是 afterGettingFrame 和 continuePlaying。afterGettingFrame 回调来自于已经完成收取数据后，即 RTSP 传输的裸数据。当然，如果单单用这个数据去解码定不能返回正确的码流，故还需要建立一个类 H264or5VideoStreamSink 继承 StreamSink，同之前的 StreamSink 相同，H264or5VideoStreamSink 也重写了 afterGettingFrame。

在成功获取并发送 sps 和 pps 之后，即可按照 H264VideoStreamSink 类中回调函数 afterGettingFrame 的数据直接操作。目前比较推荐的做法有两种，第一种是构造一个指定长数组，将回调接收的数据拷贝进定长数组中，当数组接收的数据达到最大值时就产生回调，也就是 FIFO 模式。第二种即本例中使用缓存模块，将数据存到链表中，等待其他程序调用时再读取数据。两种方法优劣各有千秋，但是本系统为了追求低延时使用了第二种几乎无延时的方法。

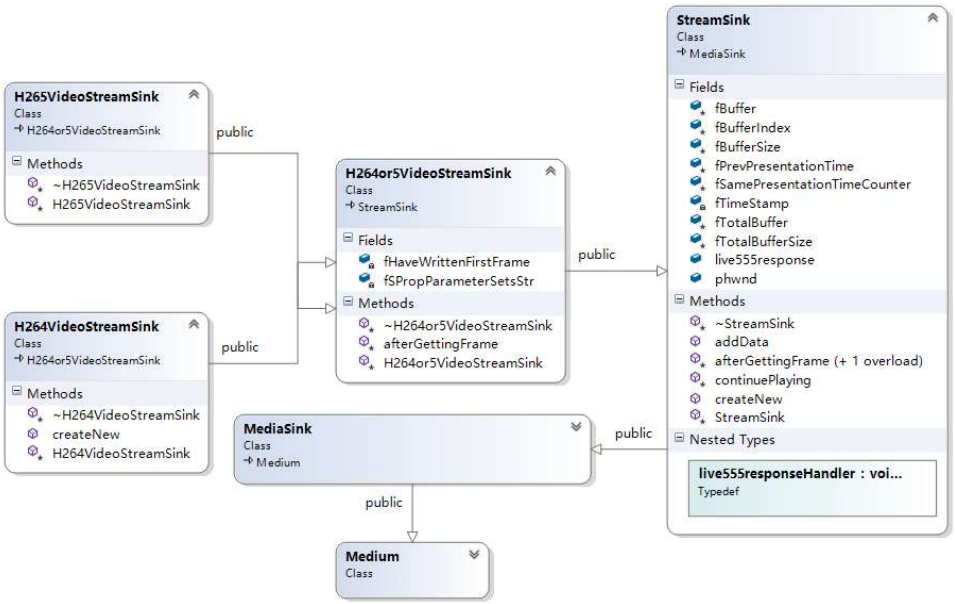

图 4-6 StreamSink 类图

MythH264Decoder 类：

该类继承于 mythVirtualDecoder，可解析 h264 裸码流文件并保存如缓存模块中。类中主要涉及 h264 的解析步骤。H264 解析步骤主要参考 h264 格式并尝试将其解析为每一个独立帧包，类中队读取文件有直接加载入内存读取和从文件中读取两种方式，在此介绍直接加载入内存读取方式的流程。

流程首先通过 fopen 打开文件并获取文件长度，随后申请与文件长度相等的内存大小，并通过 getnal 过程读取 h264 视频包帧头位置，如果帧头位置为 0 则表明文件读取完毕，退出程序。如果帧头位置不为 0，则继续判断是否读取过帧率，如果没读过则继续判断分割出的帧是否是 sps，如果是则读取该视频

流的 fps。将分割出的帧存入缓存模块中并根据 fps 进行延时，随后进行下一次 getnal 操作。操作流程如图 4-7 所示。

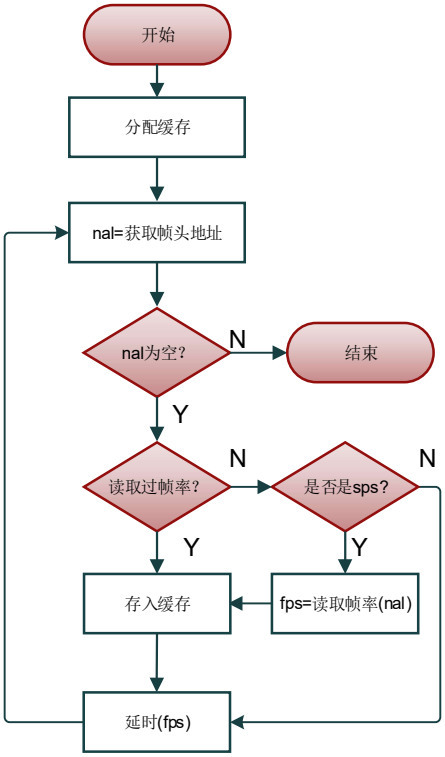

图 4-7 读取 H264 文件流程图

MythFFmpegDecoder 类

该类继承于 mythVirtualDecoder，可通过 FFmpeg 将 h264 视频流从各种封装格式中提取出，封装格式包括 mp4,mkv,ts,rtmp 等。该类将通过 av_read_frame 从文件中读取出 h264 视频流，并通过 h264_mp4toannexb 过滤器将 h264_mp4 视频流转为纯 h264 视频流并存入缓存模块。流程图如图 4-8 所示。

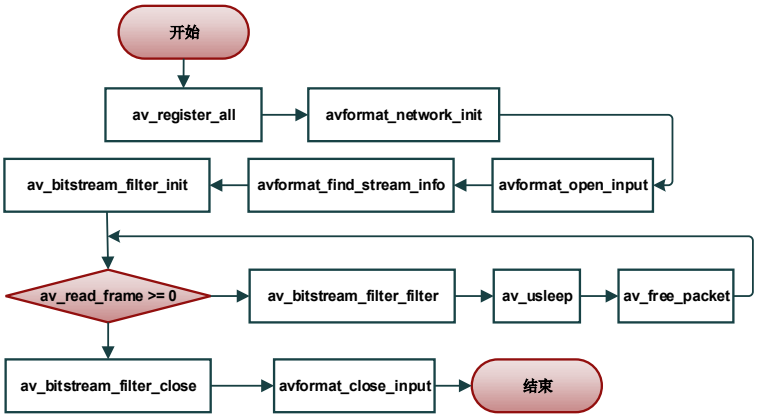

图 4-8 FFMPEG 使用过滤器获取 H264 流程图

注：MythFFmpegDecoder 类需要编译条件中打开 enable-pipeline 选项并重新编译。

MythProxyDecoder 类

该类继承于 mythVirtualDecoder，通过 MythSocket 类接收请求端发送的推流。接收推流设计的重连等算法将在第五章中详细阐述。类图如图 4-9 所示。

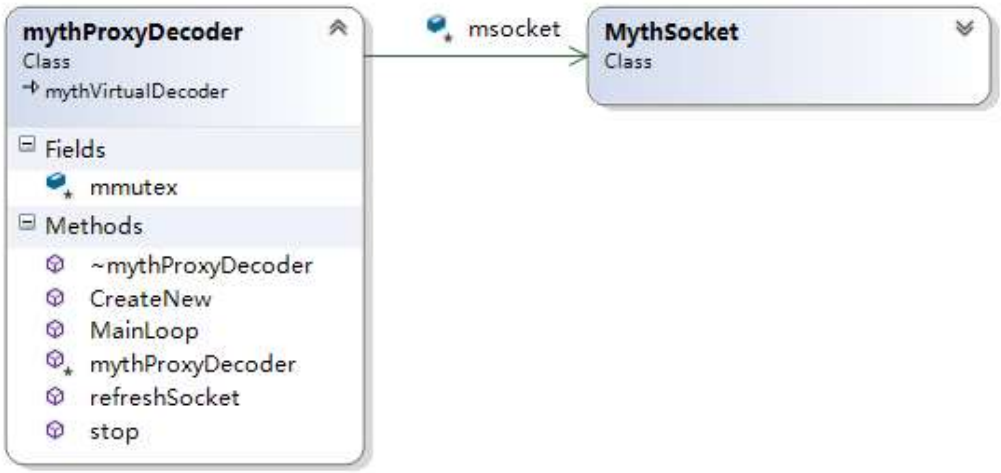

图 4-9 MythProxyDecoder 类图

4.1.2 缓存模块

基于开发需求，模块将接收模块解析出的视频帧进行缓存，供报文生成模块调用，要求尽量不要产生内存分配，也不会发生内存泄漏。缓存模块主要由以工厂模式编写的 mythListFactory 类，缓存虚类 mythVirtualList，大内存链表 mythAvlist 类与普通链表类 mythNodeList 类组成，如图 4-10 所示。

MythVirtualList 类：

该类是缓存虚类，只有 put 和 get 两个函数事件供生产者或消费者进行调用，并拥有一个简单的判断函数 isIframe 用于判断当前链表头的帧头是关键帧，h264 与 h265 的关键帧算法将之后着重介绍。

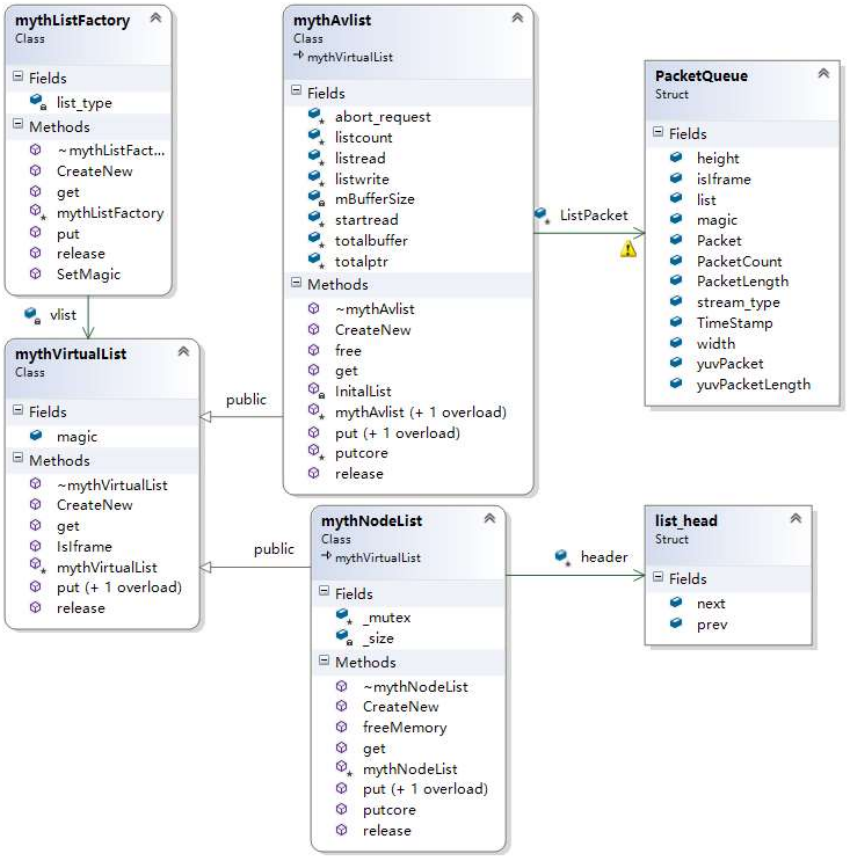

图 4-10 MythVirtualList 类图

MythAvlist 类：

该类是服务器底层数据结构中最重要的类。内部是改进的 RingBuffer 数据结构。RingBuffer 数据结构的详细算法将在第五章算法中着重介绍。

MythAvlist 有两个模板导出函数 put 和 get，分别代表取链表头数据和存数据到链表尾。与其他链表频繁 malloc 申请内存并 free 释放内存不同，内存块链表不需要申请和释放，该架构将妥善解决服务器稳定性差的问题。

导出的数据被保存在一个结构体 PacketQueue 中。PacketQueue 中可以存储任意长度的字符串数据或者 yuv 数据。

MythNodeList 类：

该类是服务器在 ARM 平台运行时使用的缓存模块核心，其内部是普通的双链表，生产者将数据保存在链表的尾端，消费者将永远读取链表的首端，当尾端等于首端时则表示链表内没有数据。当链表长度过大，链表将主动抛弃首端数据。该类用于较小内存，服务人数较少的 ARM 平台中。缺点在于重复申请内存有可能产生内存碎片，需要通过 linux 下对 malloc 的优化来避免。

PacketQueue 类：

该类用于保存缓存模块中每一个包的数据，每包数据包括数据指针 Packet，数据长度 PacketLength，数据类型 stream_type,时间戳 timestamp 等。

4.2 报文生成模块

报文生成模块主要用于将纯视频流数据封装为其他视频流格式，目前包括 http-flv 格式，自定义报文格式与 TCP-h264/h265 格式。报文生成完毕后将被转发模块进行调用并转发给其他请求端。该模块主要由自定义报文生成类 mythStreamClient，裸 h264 视频流生成类 mythH264Client 与 flv 格式视频流生成类 mythFLVClient 组成，他们都继承与 mythBaseClient，并由工厂模式入口 mythClientFactory 负责调用。该类图如图 4-11 所示。

MythStreamClient 类：

该类是对 mythSocket 类行为的封装，用于制作 HTTP 流供用户以及服务器内部进行传输，由于网络透传，支持浏览器，客户端接收更加方便等种种原因，传输使用了 HTTP 封装协议，而 HTTP 封装也是服务器对所有其他协议或数据的最后表述方式。HTTP 封装如图 4-12 所示。服务器必须自己填充 stamp 邮戳信

息，用于客户端还原数据包真实显示时间，同时需要记下包长度，并且明确显示在每个包的 HTTP 头的 content_length 属性中，用于校验数据是否正确。最后使用分隔符boundary给予强制分割包，用于处理粘性较大的socket系统中。配合 content_length 以及 boundary 标志间隔可以轻松判断是否将一帧完整收入，若非，则抛弃该帧，直接读取下一帧。在读取 HTTP 数据包的算法中，网络状况良好的情况下，可以通过读取第一包数据包中的 content_length 直接预估下一包的 content_length 位置。

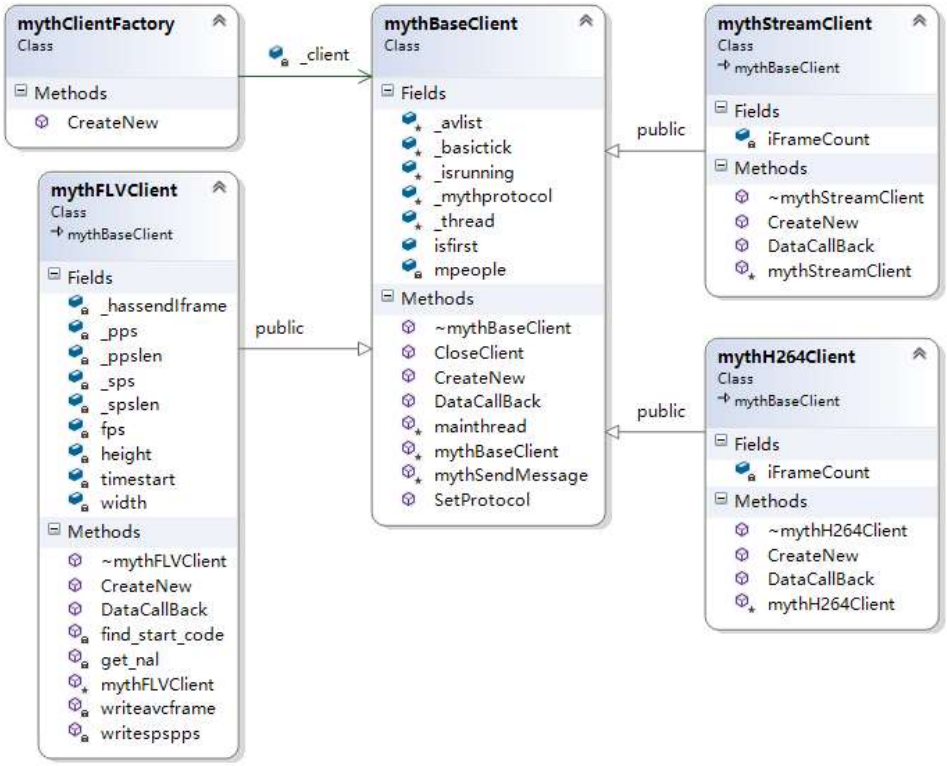

图 4-11 报文生成模块类图

第一包数据	HTTP/1.1 200 OK	
	Server	
	Connection	
	Content-Type	
	boundary	
包 1	Http header	Content-Type
		Content_Length
		Stamp
	H264 data	Byte[]

	Http end	boundary	
包 2	Http header	Content-Type	
		Content_Length	
		Stamp	
	H264 data	Byte[]	
	Http end	boundary	

图 4-12 HTTP 封装协议栈

MythBaseClient 类

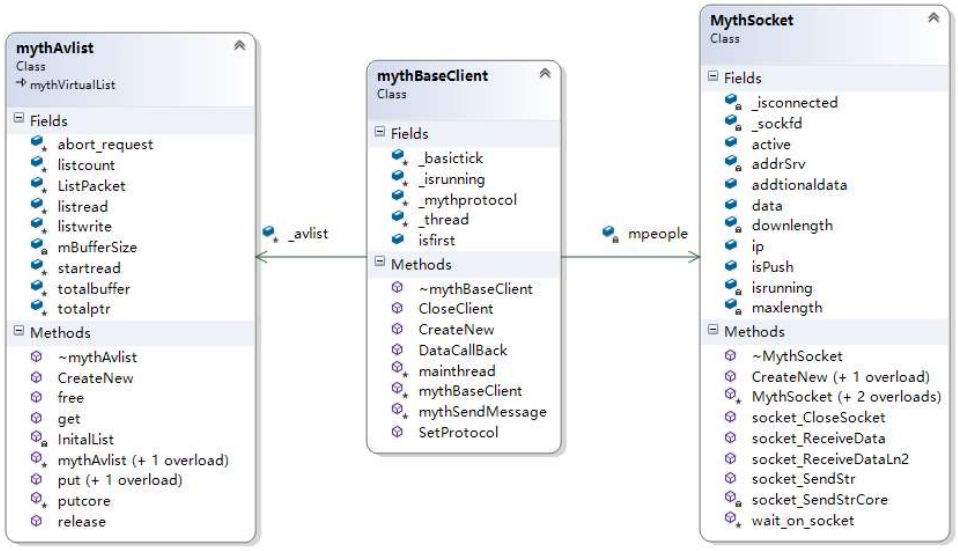

图 4-13 mythBaseClient 类图

MythBaseClient 类的类图如图 4-13 所示，每一个 mythBaseClient 类都继承了 mythAvlist，内部都有自己的内存链表，用于控制发送的速率，在接收外部发送指令后，会将其先保存如内存链表中，再异步将数据推送出 mythSocket 的 socket_send 中。这样就不会出现因为用户的问题造成服务器假死的问题。

MythH264Client 类

该类是对 mythSocket 类行为的封装，用于制作 HTTP 流供用户以及服务器内部进行传输，由于网络透传原因，传输使用了 http-h264 封装方式，HTTP 封装也是服务器对所有其他协议或数据的最后表述方式。http-h264 封装方式如图 4-14 所示：

第一包数据	HTTP/1.1 200 OK	
	Server	
	Connection	
	Content-Type	
包 1	H264 data	Byte[]
包 2	H264 data	Byte[]
包 n	H264 data	Byte[]

图 4-14 TCP 封装协议栈

该类主要用于与 kurento-media-server 交互数据，通过 webrtc 信道发送 h264 裸码流数据，随后在 kurento-media-server 中进行转码，封包为 vp8 后推送到网页客户端。

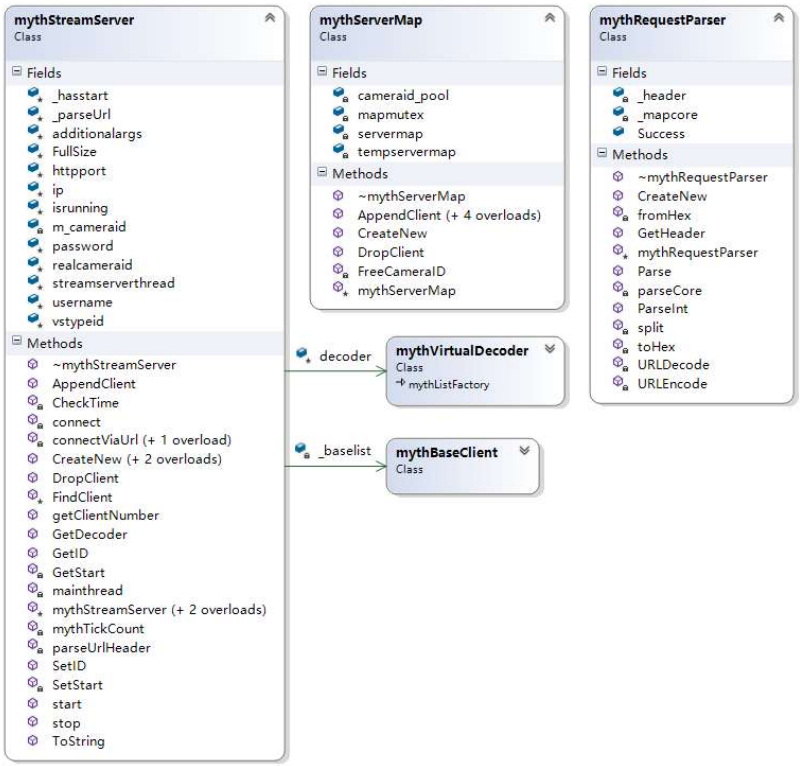

图 4-15 转发模块类图

4.2.1 转发模块

转发模块是视频服务器中的重要核心，他负责接收请求端的请求，并根据

请求内容寻找相应的缓存模块管理类，并将请求端的 ID 加入到缓存模块管理类中，最后负责将数据发送回请求端。该模块使用 libevent 开源库完成，具有优秀的并发能力与处理能力。该模块主要由 libevent 作为核心的 mythStreamMapServer,管理生产者和消费者的 mythStreamServer 类，服务器调度类 mythServerMap 类，请求解析类 mythRequestParser 类组成。如图 4-15 所示。

mythStreamServer 类

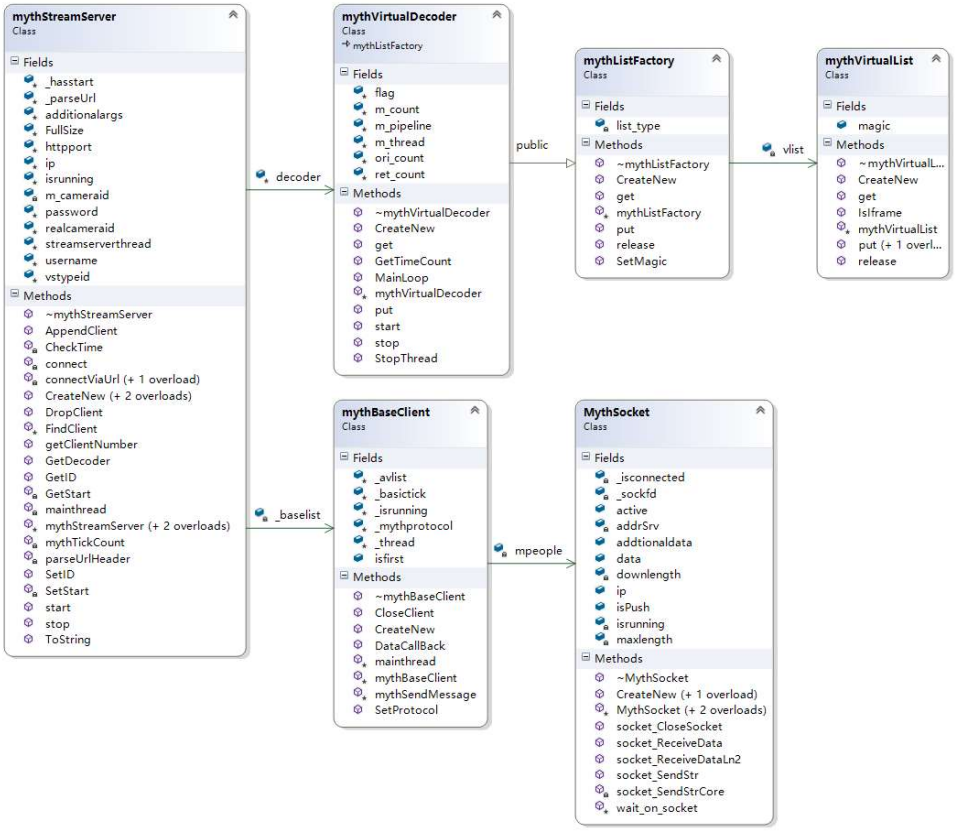

图 4-16 mythStreamServer 类图

该类是对 mythVirtualDecoder 以及 mythBaseClient 类的综合封装调度，使用的是工厂模式以及生产者消费者模式，类图如图 4-16 所示，mythStreamServer 类中通过 CreateNew 得到 camera 编号的实例化对象，该对象类似于容器用来封存多个同时请求该 camera 编号的用户，随后内部开启轮询观察，在生产者有新的视频帧后，通知类中的所有用户并将数据流推送给用

户，其中，mythBaseClient 中的推送行为可以异步，也可以是同步，以当时服务器负载来判断。同时,mythStreamServer 中可以调用 AppenedClient 和 DropClient 来删除或增加新的 mythBaseClient 进入，两者的增加时间复杂度接近 O(1)，由于可选异步操作，故所有该对象能承受最多 100 路的瞬间负载。内部私有函数 connect 用来连接数据库中间件进行判断 camera 编号是属于何种类型的数据流，并且类似于工厂模式调用相应的 decoder 并把他保存在对象内部供自己调用，整个行为不对外公开，故保持极高的高内聚低耦合状态。

与之前所有的类相同的是，他的普通接口也有 start 和 stop 两个事件，用这两个事件控制 server 的开和关，可用于点播状态。

myhStreamMapServer 类

该方法是整个服务器的请求端入口，承担的任务是将用户传来的报文中的 camera 编号分割出来，并且分配到相应的 mythStreamServer 中。如果之前没有生成指定 camera 编号的 MythSreamServer 对象则重新生成一个。找寻 mythcameraServer 是否存在的算法复杂度为 O(logn)，是标准红黑树

mythStreamMapServer 底层是 libevent 开源库，默认使用 epoll 上边缘触发模式，默认过滤效率较差的 select 模式。Libevent 库是通过异步调用实现的高效率，所以编写 libevent 程序也较为困难，不过 c++11 的新特性 lambda 表达式很方便为我们提供了解决方案。最终我们基于 libevent 库完成 mythStreamMapServer 代码。并采用了异步方式。

4.3 数据库中间件详细设计

数据库中间件是视频服务器和客户端连接数据库的中间层，可作为一层保护使用，同时作为中间层可以任意更换数据库类型而不需改动服务器代码，类似 orm 架构。数据库中间件主要由转发模块和数据库查询更改模块组成，其中在实际开发中，转发模块在同一个类中。数据查询更改模块查询的数据库为 sqlite，在文中将详细介绍查询流程。

4.3.1 转发模块

该模块与视频服务器中的转发模块相似，主要用于对请求端传送的报文进行监听收取，在收取报文后可以传送到 sqlite 调用模块，完成调用后返回给请求端。本质上是一个简单的 tcp 服务器。转发模块主要包括服务器基类 mythVirtualServer，服务器主类 mythBaseServer 和 socket 客户端封装类 People 类。模块类图如图 4-17 所示。

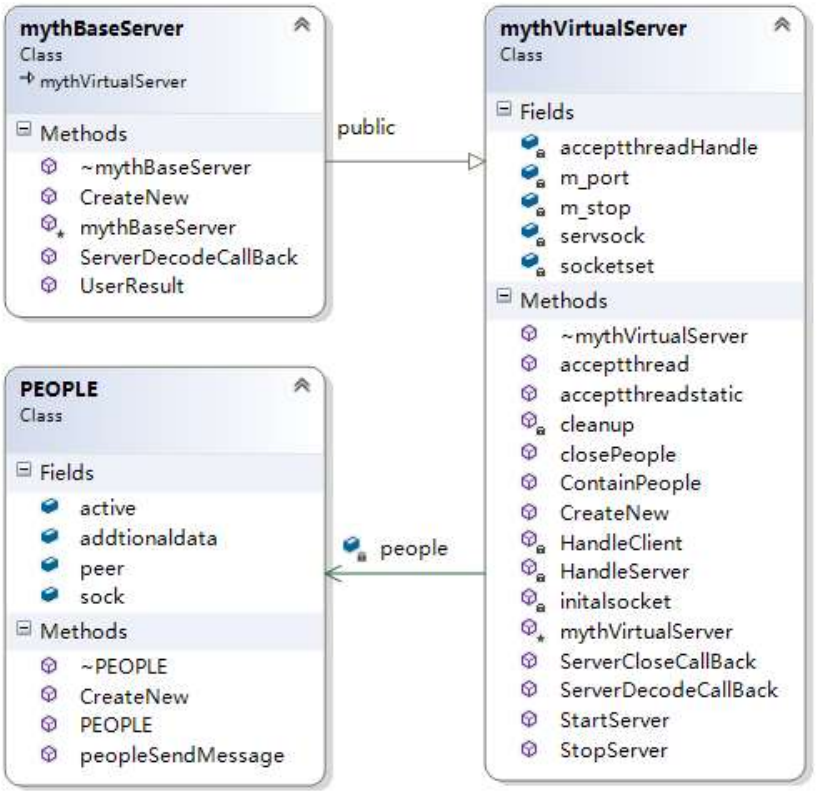

图 4-17 报文模块类图

MythVirtualServer 类

该类是所有服务器的基类，使用 createNew 作为实例化入口，传送参数为端口号。Start 函数自动开启监听模式。该类留下了两个接口 Server Close Callback 和 Server Decode Callback 提供回调。该服务器类基于 SDL_net，SDL_net 基于 SDL，由于与本系统实现方法无关故不展开讨论。

MythBaseServer 类

该类继承于 MythVirtualServer 类，重写了两个回调，主要处理服务器接收到的报文信息，并且启动 mythvirtualsqlite 类进行对数据库进行异步查询。

使用异步查询避免了由于 sqlite 查询时间过长而导致程序假死的情况，能够在其他线程查询的同时继续服务。

目前报文采用 HTTP 协议+XML 协议。为了做到与数据库主引擎分离，自定义报文可以与主数据库采用语言可完全不同，同时也可相同。报文使用例子如图 4-18 所示。为更使全文更通俗易懂以下采用明文示例：

```
GET /scripts/dbnet.dll?param=
<XML>
<function>SQL_SELECT</function>
<Content>SELECT%20A.CAMERAPTZ,A.USERCAMERAID,%20B.CAMERAID%20,%20B.CAMERANUM,B.NAME,B.SUBNAME,%20B.PORT,B.PTZCONTROL,B.SERIALPORT,B.AUDIO,%20C.HTTPPORT,C.IP,C.VSTYPEID,C.NAME%20AS%20SERVERNAME,C.VIDEOSERVERID,D.NAME%20AS%20GROUPNAME,D.GROUPID,D.TYPE%20AS%20GROUPTYPE%20,B.PTZCONTROL,B.AUDIO,B.DESCRIPTION,C.USERNAME,C.PASSWORD,FROM%20USERCAMERA%20A,%20CAMERA%20B,%20VIDEOSERVER%20C,%20GROUPS%20D%20WHERE%20D.USERID%20=%201014%20%20AND%20A.GROUPID%20=%20D.GROUPID%20AND%20A.CAMERAID%20=%20B.CAMERAID%20AND%20B.VIDEOSERVERID%20=%20C.VIDEOSERVERID%20ORDER%20BY%20D.DESCRIPTION%20ASC,D.GROUPID%20ASC,A.USERCAMERAID%20ASC</Content>
</XML>
HTTP/1.0
```

图 4-18 数据库中间请求协议实例

```
<?xml version="1.0" encoding="gb2312"?>
<XML>
<TableSchema>CameraPTZ,UserCameraID,CameraID,CameraNum,Name,SubName,Port </TableSchema>
    <TableContent>
        <line>
            <CameraPTZ>20</CameraPTZ>
            <UserCameraID>1165</UserCameraID>
            <CameraID>1731</CameraID>
            <CameraNum>0</CameraNum>
            <Name>180.168.218.195</Name>
            <SubName>9008</SubName>
            <Port>1</Port>
        </line>
    </TableContent>
</XML>
```

图 4-19 数据库中间件请求回复实例

服务器在调用 xml 信息读取 value 后，按照读取的 sql 语句或者其他查询语句对数据库检索查询，之后将信息汇总，编码为 xml 后传回给客户端。服务

器发送报文如下所示，为使原理更易懂，报文示例采用明文表示，如图 4-19 所示。TableSchema 表示返回的所有字段名称，TableContent 则为所有查询到的数据，每一段查询的数据结果用 line 封装，依据 TableSchema 可以轻松查询到所有字段对应的数据。至此查询阶段结束。

PEOPLE 类

该类为一个简单的 tcp 客户端，既可以主动发起连接，也可以在绑定 sock 的情况下主动推送信息给用户。核心过程为 sendMessage，其中内部所有操作都是异步完成，即不会出现因为 sqlite 查询时间过长而导致的程序假死情况。

4.3.2 数据库查询更改模块

该模块主要用于对 sqlite 进行增删查改操作，该类使用了 sqlite3 和 iconv 两个开源库完成查询。模块主要由数据库查询类 mythVirtualSqlite 类，数据库结果储存类 mythSQLResult 类组成。

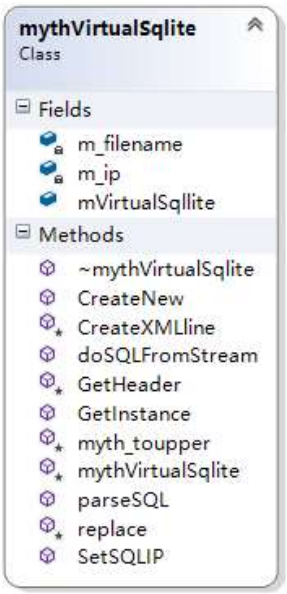

图 4-20 mythVirtualSqlite 类图

mythVirtualSqlite 类

该模块主要用于使用单例模式调用 sqlite 库，并读取数据库资源。在收到 mythBaseServer 的请求后异步或者阻塞增删查改，并将返回值退还给 mythBaseServer 中的 people 类供其发送。类图如 4-20 所示。

sqlite 的优点有本地化，处理小型数据库任务时效率可能比其他大型数据库要高，体积小，无需安装，便于携带，跨平台可嵌入等等。所以选择 sqlite 的原因就是他轻量可以在本地部署。

sqlite 允许用户使用动态链接，静态链接甚至包含源码等方式调用。由于跨平台的特殊需求，我在此选择直接将 sqlite.c 文件作为源码包含在工程内。此架构优势在于只需要一份 makefile 即可跨平台，而不需重新编译每个平台下的 sqlite，劣势在于由于 sqlite.c 大小达到 5M 多，故编译起来速度偏慢，故不适合在开发期间使用。

调用 sqlite 数据库做增删查改动作非常容易，只需要下面几个函数进行简单的搭配即可运作，sqlite 的核心 api 函数如表 4-1 所示。

API 函数	意义
sqlite3_open	打开 db 数据库
sqlite3_get_table	阻塞读取数据库数据
sqlite3_free_table	释放内存
sqlite3_exec	异步读取数据库数据
sqlite3_close	关闭整个查询

表 4-1 sqlite 核心 API

在这里有一点要强调，由于 sqlite 内部编码为 utf-8，故在处理中文数据库时需用 iconv 进行转码,iconv 调用机器简单，只需使用几个函数即可运作。Iconv 核心 api 函数如表 4-2 所示。

API 函数	意义
iconv_open	打开转换库
iconv	转换字符串
iconv_close	关闭转换库

表 4-2 iconv 核心 API

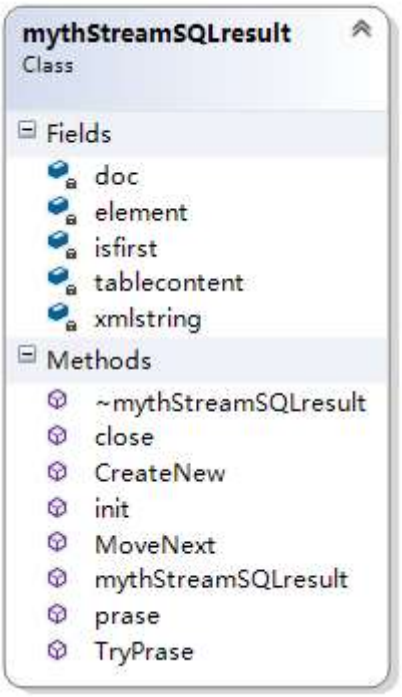

图 4-21 mythStreamSQLresult 类图

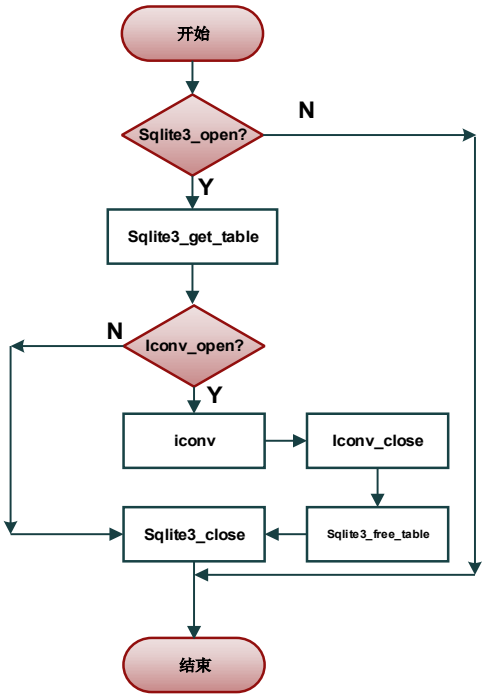

图 4-22 查询数据流程图

mythvirtualSqlite 的返回值是一个类 mythSQLresult，类中包括多项返回值，类图如图 4-21 所示；查询流程图如图 4-22 所示。

4.4 桌面客户端详细设计

桌面客户端主要由桌面 UI 模块和视频显示核心模块组成，UI 模块调用核心模块。其中 UI 模块使用 c#.net framework4.0 编写。视频显示核心模块使用 c++编写并编译成 dll 供 UI 模块调用。

4.4.1 UI 模块

UI 模块主要用于与用户进行交互，模块主要由用户登录类 login，用户查看类 winmain，数据库查询类 drizzle 组成。

Login 类

该类主要用于用户登录，由三个输入框，两个按钮组成。三个框分别是输入用户名，输入密码，输入 IP，两个按钮分别是登录，退出。该类的类图如图 4-23 所示。用户登录成功后将会记录 userid 编号并进入 winmain 类。

Winmain 类

该类主要用于将视频流，可观看视频列表呈现给用户。Winmain 类中主要由一个树形列表，和一个视频控件组成。该类的类图如图 4-24 所示。树形列表经过 sql 语句之后重新排列，根据数据库表内情况，数据库查询语句涉及多表联查，非常复杂，一般通过事务调用。数据库查询语句如图 4-25 所示。

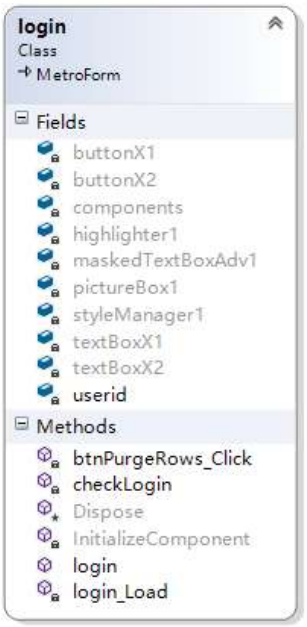

图 4-23 Login 类图

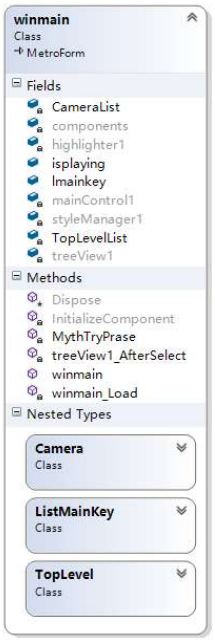

图 4-24 winmain 类

SELECT a.CameraPTZ,a.UserCameraID, b.cameraid , b.CAMERANUM,b.name,b.subname, b.port,b.ptzcontrol,b.SerialPort,b.audio, c.httpport,c.ip,c.vstypeid,c.name As ServerName,c.videoserverid,d.name AS groupname,d.groupid,D.type as grouptype ,b.ptzcontrol,b.audio,b.description,c.username,c.password,CASE when MultiCastIP is NULL or MultiCastIP = '' then '' else (select externalip from server where serverid = multicastIP) end as MultiCastIP FROM UserCamera a, Camera b, videoserver c, groups d WHERE d.userid = {0} AND a.groupid = d.groupid AND a.cameraid = b.cameraid AND b.videoserverid = c.videoserverid order by d.Description asc,d.groupid asc,a.usercameraid asc

图 4-25 winmain 类数据库查询语句实例

在查询完数据后，进入整合阶段。建立两个字典，分别是一级目录的字典和二级目录的字典。每次读取出新的一级目录就创建一级目录词条，否则添加进入一级目录词条对应的二级目录字典。视频控件主要由自定义控件 MainControl 类和 API 核心类 MythAvPlayer 组成。他们两个是聚合关系。该模块结构图如图 4-25 所示。

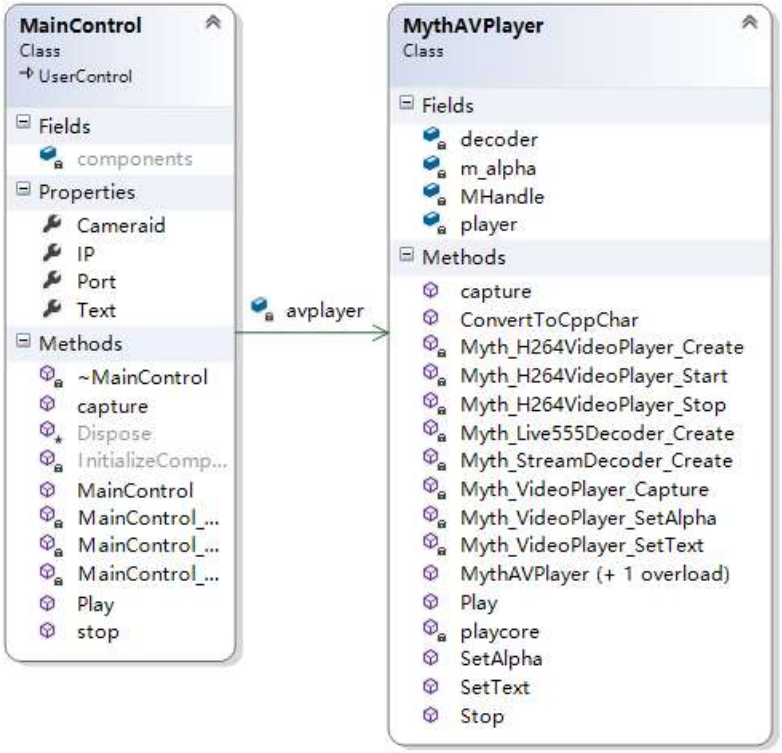

图 4-26 视频控件类图

MainControl 类主要由三个事件，控制播放的 play，控制停止的 stop，负责截图的 capture。内部有一个 MythAvPlayer 类，该类主要调用 c++核心 cyclops.dll 的导出函数。由类图可知，dll 导出函数如表 4-3 所示。

导出函数名称	导出函数作用
Myth_StreamDecoder_Create	创建自定义协议解码器
Myth_Live555Decoder_Create	创建 RTSP 协议解码器
Myth_H264VideoPlayer_Create	创建 h264/h265 播放器

Myth_H264VideoPlayer_Start	播放解码器解码出的视频
Myth_H264VideoPlayer_Stop	停止播放解码器解码出的视频
Myth_VideoPlayer_Capture	截图当前帧
Myth_VideoPlayer_SetText	在播放器上绘制字符串
Myth_VideoPlayer_SetAlpha	设置播放器透明度

表 4-3 视频控件导出函数表

使用时，首先使用导出函数创建一个播放器，随后调用导出函数的播放 API 即可播放。综上所述，用户可通过绘制的控件句柄 handle，RTSP 连接字符串 RTSP，用户名 username，密码 password 四个参数创建并播放了相应的视频流。

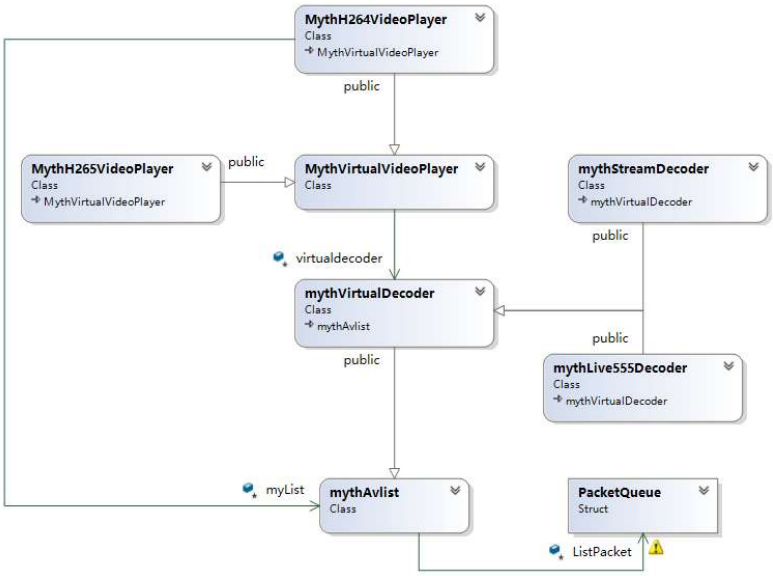

图 4-27 视频核心显示模块类图

4.4.2 视频显示核心模块

视频显示核心模块用于客户端显示视频流。模块在 windows 下使用动态链接库的方式进行交互。模块主要由 H265 视频显示类 MythH265VideoPlayer 类，H264 视频显示类 MythH264VideoPlayer 类，视频显示虚类 MythVirtualPlayer 类，解码虚类 MythVirutalDecoder 类与它的所有基类。其中

MythVirtualDecoder 类中的所有类都与视频服务器中的类相同，在此不加赘述。模块结构图如图 4-27 所示。

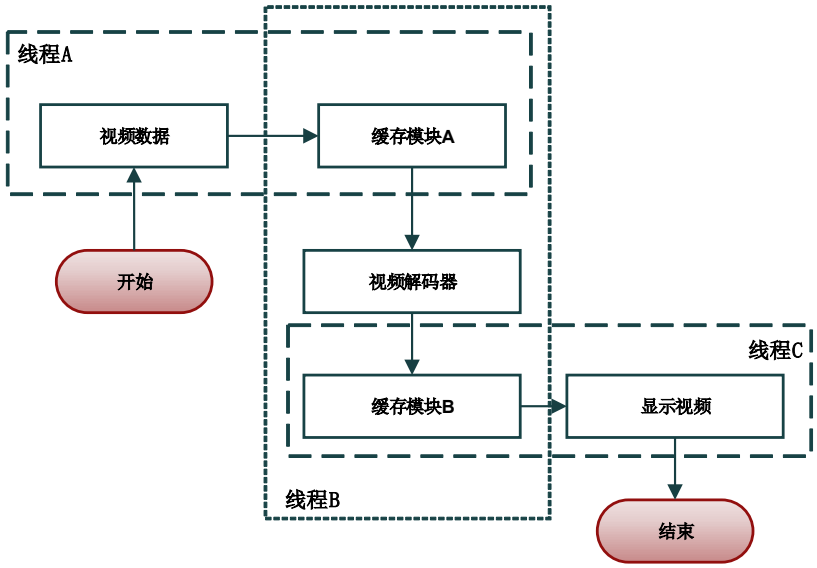

图 4-28 多线程解码播放流程图

MythH264VideoPlayer 类与 MythH265VideoPlayer 类

两个类结构大致相同，在此以 MythH264VideoPlayer 类详述。类中包括 SDL2[33]绘图函数与 ffmpeg 解码函数。该类主要有三个线程，线程 A 负责接收 VirualDecoder 类传来的视频数据，随后储存在缓存模块 A 中。线程 B 负责将缓存模块 A 中的视频数据放到 ffmpeg 解码函数中，解码出 YUV 图像并储存在缓存模块 B 中。线程 C 负责将缓存模块 B 的 YUV 图像通过 SDL2 绘图函数绘制在界面上。缓存模块 A 和缓存模块 B 都有跳帧技术，应付解码或者显示延迟的情况，牺牲画面流畅性，减少延迟。三个线程的流程图如图 4-28 所示。

4.5 推流器详细设计

由系统设计可知，推流器主要由前端采集模块，缓存模块，编码模块和推送模块组成。其中缓存模块和推送模块与数据库中间件中的缓存模块与发送模块完全相同。故在本章主要介绍推流器特有的前端采集模块和编码模块。

4.5.1 前端采集模块

该模块主要用于采集前端摄像头原始视频流，并以生产者的身份将它缓存到缓存模块中，供发送模块进行消费者读取。当编码速度小于采集速度时，可以调用缓存模块的丢帧指令通过画面跳帧来跟进速度。该模块由摄像头采集模块 myth CameraDecoder 类，自定义协议解码模块 mythStreamDecoder 类，该类与视频服务器中的 mythStreamDecoder 类完全相同，且推流器中的缓存模块与与数据库中间件中的缓存模块相同，模块包括 mythVirtual-Decoder,mythAvlist 和结构体 Pa-cketQueue。该模块结构图如图 4-29 所示。

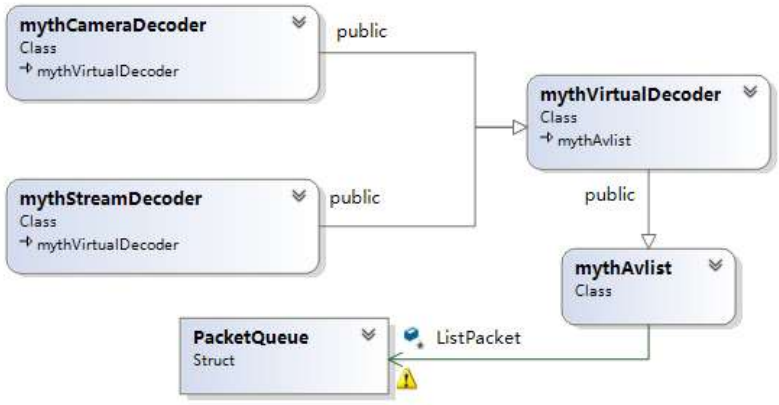

图 4-29 前端采集模块类图

MythCameraDecoder 类

该类主要采集前端摄像头原始视频流，本推流器中的前端摄像头指的是笔记本电脑自带的摄像头或者 USB 摄像头，该类读取摄像头数据使用了 opencv 开源库，并封装入 mythCamera 类中。

该类使用 c 语言 API 调用 opencv，其中 cv 和 cxcore,highgui 使用频率较高。使用 opencv 读取摄像头的主要核心代码在 cv 内，详细名称如表 4-4 所示：

函数名称	用途
cvCreateCameraCapture	创建摄像头捕捉对象
cvQueryFrame	获取摄像头数据
cvReleaseImage	释放摄像头捕捉对象

表 4-4 opencv 获取摄像头 API

cvCreateCameraCapture 调用参数一般使用 CV_CAP_ANY，让 opencv 自动判断，例如在 windows 下会自动变成 CV_CAP_DSHOW。若创建对象成功则返回 CvCapture 对象。

cvQueryFrame 参数即之前创建的 CvCapture 对象，若成功则返回一帧 IplImage。一般来说返回的数据为 argb 格式，需要调用 FFmpeg 将其转化为 yuv 格式。然而 opencv 中也有 argb 转 yuv 的方法，不过由于申请内存次数较多而放弃。

cvReleaseImage 参数即之前创建的 CvCapture 对象，在调用摄像头完毕后直接调用函数关闭所有操作。同时需释放 IplImage 指针。

mythCamera 类是基于 opencv 的三个基本读取摄像头封装的类，如图 4-48 所示，该类使用单例模式保证程序运行后只有一个 mythCamera 类存在。MythCameraDecoder 类调用 Capture 返回摄像头采集到的图像的 argb 矩阵，调用 CloseCamera 关闭摄像头。

4.5.2 编码模块

编码模块负责将 Capture 后的 RGB 矩阵转为 H.264 格式视频流。该模块由 mythFFmpegEncoder 类组成。该类与 mythCameraDecoder 类是聚合关系。

MythFFmpegEncoder 类使用了 FFmpeg 开源库作为编码核心。FFmpeg 最核心同时也是最强大的功能就是视频格式转换。与一般官方出品的格式转换不同，FFmpeg 可以转换几乎所有主流视频格式。当然，有许多转换也是通过第三方库的链接完成，例如这次编码使用的 libx264。所有第三方格式转换方法最后将使用规定的函数调用格式注册进 FFmpeg 的总表中。目前 FFmpeg 编码的核心函数是 avcodec_encode_video2，根据之前设置的输入参数判断调用的编码方式等。整个编码流程如图 4-30 所示。

编码完成后将 h264 数据从结构体 AvPacket 中取出，并存入缓存模块中。由于在平板，手机或其他便携设备的主频不够，在读取摄像头后使用单线程编码很可能会造成数据堵塞，导致画面卡顿，故 cc 使用多线程操作，一根线程专用读取摄像头，另外一根线程专用编码。两者通过缓存模块通信，做到类似异步操作。在机器较好的环境中，编码线程将会等待摄像头线程读取到数据后再编码，而在机器较差的环境中，编码线程将进行跳帧以赶上摄像头读取速度，由

于摄像头线程保存的数据为裸 yuv 数据流，故跳帧不会出现 h264 非 I 帧跳帧的现象。

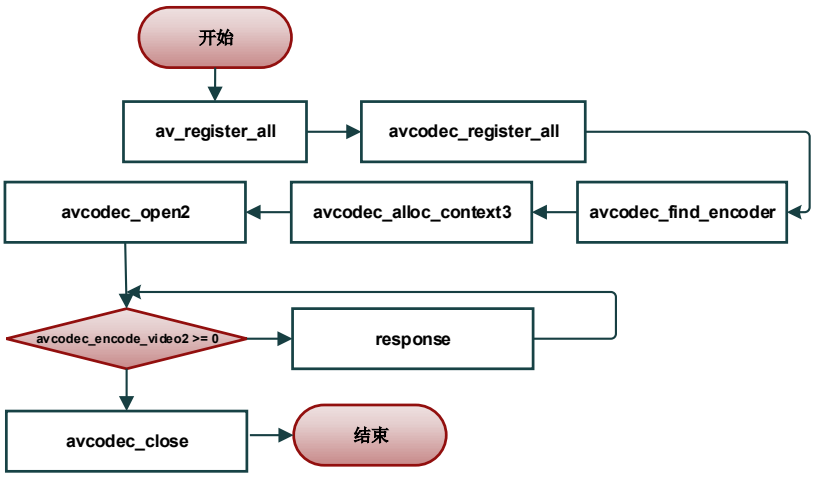

图 4-30 使用 FFMPEG 编码流程图

关于 rgb 转 yuv 操作，该类调用了 FFmpeg 中 sws_scale 算法。该算法可以将任意图像储存方式快速转化为其他任意的图像储存方式，且可通过修改参数以达到速度和图像转化质量间的平衡，因为部分转化是有损转化，例如 yuv 至 rgb。由于视频流直播不需要较高质量，故使用了最快算法。Rgb 转 yuv 代码如图 4-31 所示：

```
Input:video width:width;video height:height;argb data from opencv capture:src
Output:yuv data from FFmpeg sws_scale:dst
1.  img_convert_ctx←sws_getContext(width, height, PIX_FMT_RGB24,width, height,
    PIX_FMT_YUV420P,SWS_FAST_BILINEAR, nothing, nothing, nothing);
2.  rgbsrc[]={src, nothing. nothing };
3.  srcwidth[] ←{width>>2,width>>2,width>>2};
4.  dstwidth[]={width,width<<1,width<<1};
5.  sws_scale(img_convert_ctx, (const uint8_t *const*) rgb_src, srcwidth, 0,
    height,(uint8_t *const*) dst, dstwidth);
6.  sws_freeContext(img_convert_ctx);
7.  return dst;
```

图 4-31 使用 FFMPEG 进行 RGB 转 YUV 代码

4.6 本章小结

本章详细介绍了系统设计。首先介绍了系统开发环境，不用子系统使用了不同的开发环境开发。第二步详细介绍了视频服务器子系统，其详细给出了使用 LIVE555 读取 RTSP 流的方法。第三步介绍了数据库中间件子系统的设计，其中详细介绍了调用 iconv 和 sqlite 开源库的方法。第四步介绍了桌面客户端，其中详细介绍了调用视频核心显示模块的方法。第五步介绍了推流器的设计，其中详细介绍了调用 OpenCV 进行前端采集的方法，以及使用 libx264 进行编码的方法。最后介绍了视频显示核心模块，其中详细介绍了使用三个线程完成从视频数据到显示视频的流程。

第五章 系统的编译与运行

5.1 系统依赖库编译

本系统依赖开源库 FFmpeg ,libx264，live555,SDL2,libevent,tinyxml,cjson,sqlite,iconv,opencv,SDL2_net，大多数开源库的安装并不复杂，只需要通过简单的 configure，make，make install 即可按成开源库的编译。然而极少数的几个开源库安装选项必须值得注意。本小节将着重介绍 FFmpeg+libx264 的编译与安装与 live555 的编译与安装。

5.2 FFmpeg+libx264 的编译与安装

FFmpeg 是一个大型跨平台媒体集合，可以通过配置编译不同功能的链接库，且许可证也不相同。本程序的 FFmpeg 选用 gpl 许可证。

注：gpl 不是宽松许可证

首先，编译 YASM。YASM 是跨平台的汇编编译器，一般用来编译底层算法。在 x264 和 FFmpeg 的底层中都有 yasm 的身影。当然，你可以使用—disable-yasm 来取消汇编编译器改用 c 代码编译。效率可能会有一定损失，用在部分不支持 yasm 的机型或对效率要求不高，追求快速编译的环境中。YASM 编译脚本如图 5-1 所示。

```
1>  yasm
    #Yasm is an assembler and is recommended for x264 and FFmpeg.
    wget http://www.tortall.net/projects/yasm/releases/yasm-1.2.0.tar.gz
    tar xzvf yasm-1.2.0.tar.gz
    cd yasm-1.2.0
    ./configure
    make
    sudo checkinstall --pkgname=yasm --pkgversion="1.2.0" --backup=no \
      --deldoc=yes --fstrans=no --default
```

图 5-1 YASM 编译脚本

其次，编译 libx264。Libx264 是 vlc 下一款非常优秀的 h264 格式编解码器。其效率目前是所有开源 h264 编解码器中最高的，唯一的缺点在于它使用的 gpl 协议是强约束协议。即使用者无论如何，只要使用 libx264 必须开放其源代码并在代码中加入 license。于是，FFmpeg 组委会自行编写了一款 h264 解码器，详见 FFmpeg 源码中的 h264dec.c。其解码效率几乎与 libx264 相差无几，与 libx264 不同的是，FFmpeg 在这方面使用了 lgpl 协议，即若动态调用 FFmpeg 库则不需要开放源代码。故若有对 h264 解码的需求调用 FFmpeg 可不需公布源代码。但是 FFmpeg 内部不自带 h264 编码器，故必须将协议增加至 gpl 以让 FFmpeg 增加编码功能。Libx264 编译脚本如图 5-2 所示。

```
    2>libx264
    H.264 video encoder. The following commands will get the current source files, compile, and install x264.
    wget ftp://ftp.videolan.org/pub/x264/snapshots/last_x264.tar.bz2
    tar -jxvf last_x264.tar.bz2
    cd x264*
    ./configure --enable-shared --enable-static
    make
    make install
```

图 5-2 libx264 编译脚本

在完成以上步骤后，即可按照正常方法编译 FFmpeg，但是请注意，必须开启 gpl 选项和 x264 选项，否则有可能编译失败。FFmpeg 编译脚本如图 5-3 所示。

```
    3>FFmpeg
    wget http://FFmpeg.org/releases/FFmpeg-2.7.1.tar.bz2
    tar -jxvf FFmpeg-2.7.1.tar.bz2
    cd FFmpeg-2.7.1
    ./configure --enable-gpl --enable-libx264 --enable-shared --disable-ffplay
    make
    make install
```

图 5-3 FFMPEG 编译脚本

在完成所有步骤后，运行以下指令以确认编译成功，编译成功的话如下图所示，在 encoder 列表中会出现 libx264 以及 libx264rgb 两个编码器，如图 5-4 所示。FFmpeg -encoders|grep libx264。

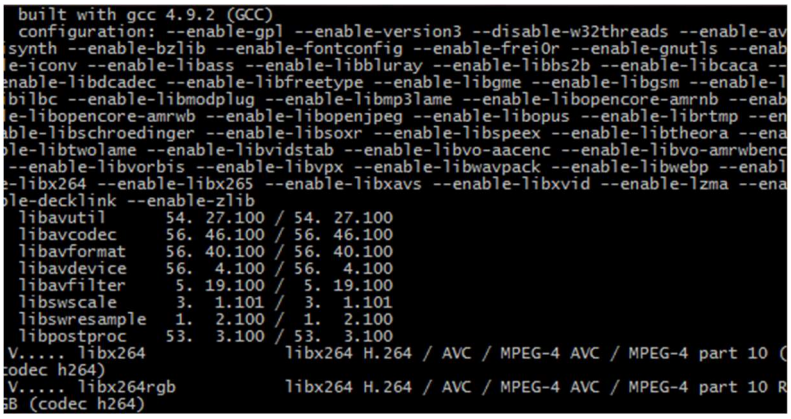

图 5-4 FFMPEG 确认是否编译 libx264

5.2.1 Live555 的编译与安装

Live555 是开源读取 RTSP 协议的开源库，他的编译模式不属于 gnu 规范。与 configure,make,make install 的步骤不同的是，Live555 在编译初期就需指定编译的系统。编译脚本如图 5-5 所示，在 genMakefiles 后需要填上当前运行的操作系统的类型。

```
cd live555
chmod 755 genMakefiles
./genMakefiles linux
make -j && sudo make install
```

图 5-5 Live555 编译脚本

5.3 视频服务器编译与运行

视频服务器目前完全开源在 oschina 上。在编译之前先确定目录下有最新的 libevent 库，live555 库,libx264 库和 ffmpeg 库。确认无误后即可通过 git

工具下载，随后运行 configure 文件进行简单配置，默认 live555 编译平台为 linux。有多个编译选项提供选择，如表 5-1 所示。

编译参数	含义
enable-pipeline	打开 h264 与 h265 的转换适配类
enable-mingw	使用 mingw 编译器编译 win 版本
enable-libevent	使用 libevent 作为并发底层

表 5-1 流媒体服务器编译参数

在 linux 下启动的方式非常简单，只需要输入 mythmultikast 即可完成启动，在启动之前需要在 mythconfig.ini 中配置数据库中间件的 IP 地址以便访问。服务器开启后如图 5-6 所示，本例采用 ubuntu14.04LTS 系统运行服务器，日志准确的记录下开启服务器的时间。服务器会监听 5834 端口，使用 select 或者 epoll 的方法进行 socket 轮询。

```
[root@localhost mythmultikast]# ./mythmultikast
[2017-06-27 21:28:49.003]Initalizing Servermap
[2017-06-27 21:28:49.004]Libevent Current Using Method: epoll
[2017-06-27 21:28:49.004]Server IP: 0.0.0.0 ---  5834
[2017-06-27 21:28:49.004]MythMultiKast in libevent: stable version.
```

图 5-6 流媒体服务器运行画面

服务器开启后记录了大量日志，日志文件在服务器文件目录下以 yyyy-mm-dd.log 的命名方法进行存储。日志记录的条目日期精确到毫秒。如今天是 2017 年 6 月 27 日，则可以通过 cat 2017-06-27.log 进行日志查看。通过日志查看可以检查大部分服务器崩溃问题。日志查看如图 5-7 所示。

```
[root@localhost mythmultikast]# cat 2017-06-27.log
[2017-06-27 21:28:49.003]Initalizing Servermap
[2017-06-27 21:28:49.004]Libevent Current Using Method: epoll
[2017-06-27 21:28:49.004]Server IP: 0.0.0.0 ---  5834
[2017-06-27 21:28:49.004]MythMultiKast in libevent: stable version.
```

图 5-7 流媒体服务器日志查看

5.4 桌面客户端设计与运行

在设计桌面客户端之前，首先明确了客户端的设计要求。首先，需要支持固定分辨率 1200x743；其次，界面直观、易学易用，还要支持快速切换风格。在界面风格上采用 metro 简约风格，整体风格简单使用，一目了然。明确总体要求后，继续进行 UI 模块的分块登录界面和视频查看界面的需求明确。

5.4.1 登录界面设计

登录界面主要包含用户登录、配置系统两大功能，如果是试用版软件还应显 示试用期。 用户账号采取用户账号登录模式，并且不提供用户注册功能，需要管理员在后台自行注册。登录界面如图 5-8 所示。

图 5-8 登录界面

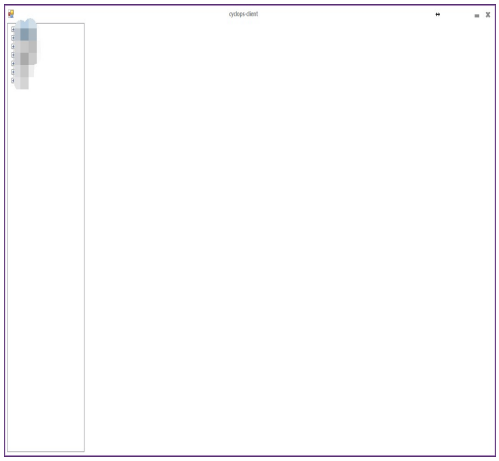

图 5-9 视频查看界面

5.4.2 视频查看界面设计

视频查看界面的工作区分为左右两部分，左侧是一个列表，当中包含着该用户权限下能够查看的视频监控设备以及其 ID，右侧是视频监控窗口，用户通过该窗口查看实时的视频监控图像。视频查看界面如图 5-9 所示。点击左侧一栏并看开树形列表，点击父节点之后进入子节点，随后鼠标单击子节点即可观看实时视频流。点击其他子节点可以切换其他实时视频流。

5.5 数据库中间件与推流器的编译与运行

数据库中间件与视频服务器一样可以跨平台运行，在这里主要介绍windows 下的编译方法。Win 下只需使用 visual studio2013 即可完成编译，确保在编译前目录下有 SDL2 库，SDL2_net 库，sqlite3 库和 iconv 库。四个库都可静态编译如应用程序内。启动数据库中间件服务器之后效果如图 5-10 所示，数据库中间件监听 5830 端口，并检测出自己的 IP 地址供视频服务器和桌面客户端调用。

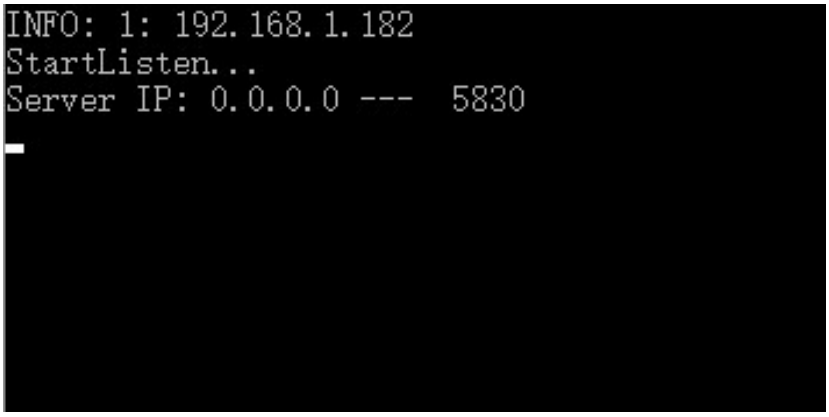

图 5-10 数据库中间件运行图

同理，推流器也可在 windows 下通过 visual studio2013 直接编译完成。在编译前请确认编译目录中有 opencv 库，SDL2 库,ffmpeg 库和 SDL2_net 库。编译过程在此不加赘述。 应用程序 myth-StreamPusher 的启动命令为：

```
MythStreamPusher stream://src/id stream://dst/id

或      MythStreamPusher camera:// stream://dst/id
```

分别用于启动自定义协议推流和摄像头数据推流。

5.6 本章小结

本章主要介绍了系统的编译与运行。首先介绍了系统几个重要的依赖库的编译与安装，随后分别介绍了各个子系统视频服务器，数据库中间件，桌面客户端的编译与运行。

第六章 关键技术实现

本系统为了满足低延时特性，基于排队论的经典排队模型 M/M/1，实现了许多特殊算法，例如帧头判断算法，关键帧判断算法，构造了许多特殊的数据结构，例如特殊的 ringbuffer 数据结构以及关键帧缓存技术，改进了开源库的源代码，例如 live555 的接收数据部分。本章将介绍这些关键技术。

6.1 M/M/1 排队模型

M/M/1 排队模型是一个单服务的排队模型，用于研究单输入输出的排队系统的概率，从而解决与得出该模型的在不同情况下的最优设计与最优控制。M/M/1 排队模型作为排队论中最简单的模型，应用广泛，尤其在计算机网络仿真中。到达时间间隔服从指数分布，则到达过程为泊松过程，且接受完服务和到达相互独立，服务时间分布为指数分布。到达和服务都是随机的，单服务台并满足先到先服务原则（FIFS），队列长度与可加入队列的个数都是无限。

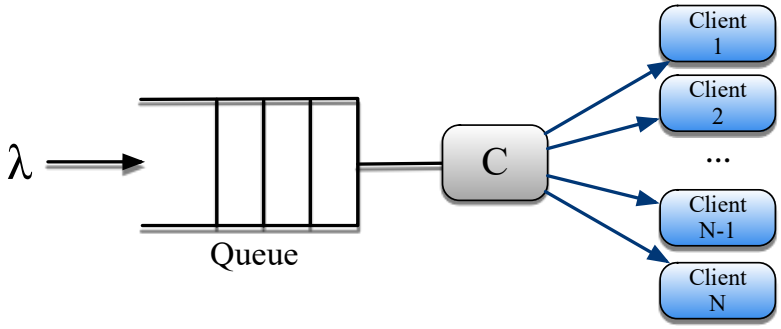

图 6-1 实时监控系统 M/M/1 模型

如图 6-1 所示，在实时监控流媒体服务场景中，我们根据排队论对每个实时视频流 i 都建立了 M/M/1 排队模型，其中定义如下：

λ_i：指定视频流 i 经过编码器编码后的数据包个数，其中数据包大小为一个 MTU 大小（1500bytes）。

C_i：指定视频流 i 分配到的服务带宽大小，本文不考虑可用带宽（ABW）带来的影响，默认为固定值。

N_i：指定视频流 i 服务的客户端个数。

那么服务速率 $\mu_i=C_i/N_i$，即随着客户端个数的增加，对于客户端来说，被服务的带宽将会变小。则在只分析一个实时视频流的情况下，我们可以得到一个包再系统中的平均逗留时间 T 为：

$$T=\frac{1}{\mu-\lambda}=\frac{N}{C-N\lambda} \quad (6\text{-}1)$$

由公式 6-1 可得，随着服务客户端个数 N 的增加，包的服务平均逗留时间逐渐增加，并且没有最大值，如图 6-2 所示，当 λ=104，C=4370,即实时视频码率为 1.2mbps,出口带宽 C=50mbps 时，客户端个数 N=31 时有明显的突增。

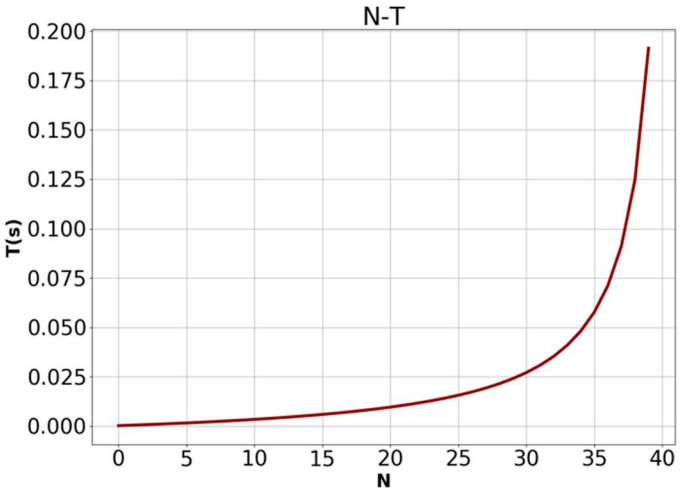

图 6-2 客户端个数 N 与平均逗留时间 T 在 λ=104，C=4370 下的关系图

同时我们可以得到系统的平均包数 N 为：

$$N=\frac{\rho}{1-\rho}=\lambda\frac{N}{C-N\lambda}=\lambda T \quad (6\text{-}2)$$

公式 6-2 意味着系统队列中平均包的个数与输入速率和队列延时的成绩有关。在实时监控系统中，我们定义延时不能超过 300 毫秒，即 T=0.3，则队列最大长度 N_{max} 为：

$$N_{max}=\lambda T_{max}=\frac{in\times 1024\times 1024}{1500\times 8.0}0.3=26.21in$$

其中 in 为输入实时视频码率，单位为 mbps。于是队列最大长度将通过视频码率动态调整。

6.2 H.264 与 H.265 帧头判断算法

在视频监控中，获取视频包的视频帧起始位置非常重要，对于接收模块来说，每收到一段数据就会开始寻找视频帧的起始位置，并在下一次寻找到视频帧其实位置后确定一个视频帧的数据。故 H.264，H.265 的帧头判断算法将极大提升接收模块的效率。

从 rfc 官方文档中可得知，H.264 与 H.265 都以 00 00 00 01 或者 00 00 01 开始。故判断帧只要判断两种情况即可。目前主流的判断方法为一位一位判断 00 00 00 01 或者 00 00 01，并没有利用计算机 cpu 对齐的优势。故本算法利用计算机 32 位对齐的优势，每次处理 4 个字节。根据两种情况分别进行两种判断，其算法如图 6-3 所示。

Algorithm 1 Find the start code for H.264 and H.265

Input: Nal unit: μ
Output: Startcode Index: int

```
1: procedure FINDSTARTCODE(μ)
2:     tmp ← μ & 0x00FFFFFF;
3:     if tmp = 0 :
4:         return 3;
5:     elseif μ = 0x10000000:
6:         return 3;
7:     else :
8:         return 0;
```

图 6-3 H264 与 H265 帧头判断算法

6.3 H.264 关键帧算法

在视频监控中，若因网络状况不佳，机器不够好等其他种种原因造成无法跟上直播流时，可以选择跳帧算法尽快跟上直播流，损失流畅度换取低延迟。然而跳帧的位置即是可以重新独立解码的关键帧数据。故必须要让服务器快速判断改帧是否是 I 帧并将其存入大内存静态链表包中。

在查阅官方 H264 资料中，可取得 NAL 类型表。NAL 全称 Network Abstract Layer，即网络抽象层。NAL 单元是 NAL 的基本语法结构，它包含一个字节的头

信息和一系列来自 VCL 的称为原始字节序列载荷（RBSP）的字节流。NAL 类型在 h264 包的头位置，即若 NAL 对应的 slice 为一帧的开始，则在{00 ,00 ,00, 01}之后；若在帧中，则在{00,00,01}之后。NALU 结果如图 6-4 所示。

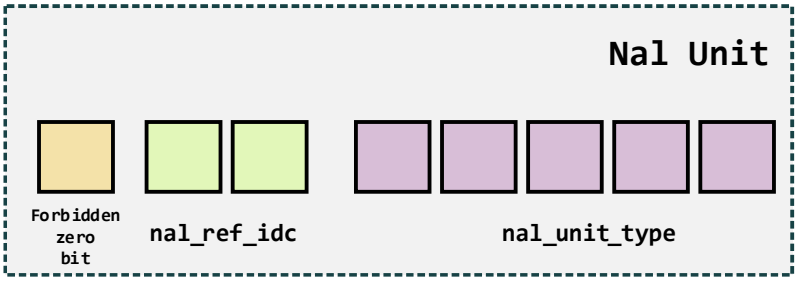

图 6-4 NALU 数据结构

nal_unit_type	NAL 类型	C
0	未使用	
1	不分区，非 IDR 图像的片	2,3,4
2	片分区 A	2
3	片分区 B	3
4	片分区 C	4
5	IDR 图像中的片	2,3
6	补充增强信息单元（SEI）	5
7	序列参数集	0
8	图像参数集	1
9	分界符	6
10	序列结束	7
11	码流结束	8
12	填充	9
13..23	保留	
24..31	未使用	

表 6-1 nal_unit_type 类型表

如表6-1所示，nal_unit_type可判断是否是关键帧，位置在NAL的低五位中。从表中可以判断出I帧，即NAL类型为5。一个编码视频序列由一串连续的存储单元组成，使用同一序列参数集。每个视频序列可独立解码。编码序列的开始是即时刷新存储单元（IDR）。IDR是一个I帧图像，表示后面的图像不用参考以前的图像。一个NAL单元流可包含一个或更多的编码视频序列。H.264的参数集又分为序列参数集（Sequence parameter set）和图像参数集（Picture parameter set）。序列参数集包括一个图像序列的所有信息，即两个IDR图像间的所有图像信息。多个不同的序列和图像参数集存储在解码器中，编码器依据每个编码分片的头部的存储位置来选择适当的参数集，图像参数集本身也包括使用的序列参数集参考信息。

经过以上研究，针对表中的数据，我们可以判断出：若图像在刷新储存单元中，即nal_unit_type为5，则肯定可以将其判断为I帧；若图像在参数序列集中，即nal_unit_type为7，则该帧拥有sps头，即后面也有完整帧；若图像在参数序列集中，即nal_unit_type为8，则该帧拥有pps头，即后面也有完整帧。在libx264中，nal_unit_type已经被整理为以下常用项。于是可以推出在关键帧状态下NAL的取值范围。如果当前帧属于刷新储存单元时，nal_unit_type为5，其二进制表达为101，扩充到5位表达为00 101。nal_ref_idc是优先级算法，取值只有可能为00,01,10,11。加上头帧的forbidden type恒为0，故可以通过简单的排列组合得到以下数据:

(0 00 00101)B = (5)H,(0 01 00101)B = (25)H

(0 10 00101)B = (45)H,(0 11 00101)B = (65)H

当帧属于SPS时，nal_unit_type为7，其二进制表达为111，扩充到5位表达为00 111。nal_ref_idc是优先级算法，取值只有可能为00,01,10,11。加上头帧的forbidden type恒为0，故可以通过简单的排列组合得到以下数据:

(0 00 00111)B = (7)H,(0 01 00111)B = (27)H

(0 10 00111)B = (47)H.(0 11 00111)B = (67)H

当帧属于PPS时，nal_unit_type为8，其二进制表达为1000，扩充到5位表达为01 000。nal_ref_idc是优先级算法，取值只有可能为00,01,10,11。加上头帧的forbidden type恒为0，故可以通过简单的排列组合得到以下数据:

(0 00 01000)B = (8)H,(0 01 01000)B = (28)H

(0 10 01000)B = (48)H,(0 11 01000)B = (68)H

在实际使用中，nal_ref_idc 优先级为 0 的概率极少，即使有，也可以因为优先级过低直接抛弃。故可直接过滤掉 nal_ref_idc 为 00 的情况。那么可以直接得出结论：**NAL 值在 0x25,0x45,0x65,0x27,0x47,0x67,0x28,0x48,0x68 时该帧为关键帧**。既然已经得到了九组特殊数据，那么我们可以直接写出判断算法。算法时间复杂度为 O(n)，采用轮询判断每个数字，得到正确解后直接返回。算法 6-4 可以应付大多数情况。若对算法执行时间不满意，可以使用改进算法:假设 nal_ref_idc 的所有优先级在考虑范围内，那么需要排除 forbidden type 为 1 的情况，并获取后五位 bit 的数据值。根据需求写出蒙版 mask=(1 00 11111)B=(9F)H，蒙版与原数据与运算如图 6-7 所示。与目前主流市场的 0x1F 判断方法不同，0x9F 过滤了 forbidden type 为 1，增加了对 android 部分硬编码错误的机型的支持。算法如图 6-8 所示，算法的时间复杂度仍为 O(1)，只需与运算一次即可得出正确结果。同理可得 H.265 的关键帧判断参数为 0x7E。

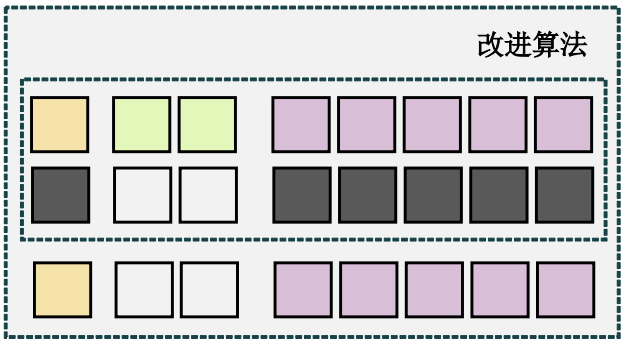

图 6-7 关键帧改进算法

Algorithm 2 Find the start code for H.264 and H.265

Input: H.264 stream data: data, Data Index: n;
Output: Is I frame or not:True/False;

```
1: procedure IsFrame(data, n)
2:     if FindStartCode(&data[n]) > 0 :
3:         KeyFrame ← data[p];
4:         if KeyFrame & 0x9f = 5:
5:             return true;
6:     return false;
```

图 6-8 关键帧改进算法代码

6.4 Ringbuffer 数据结构

链表相比于线性表顺序结构，操作复杂。由于不必须按顺序存储，链表在插入的时候可以达到 O(1)的复杂度，比另一种线性表顺序表快得多，但是查找一个节点或者访问特定编号的节点则需要 O(n)的时间，而线性表和顺序表相应的时间复杂度分别是 O(logn)和 O(1)。但是由于链表需要频繁申请内存，在有限的时间内可能不会出现问题，但是运行一定时间后，非常容易造成内存碎片的增多导致申请内存失败的现象。故因此试用了 ringbuffer 数据结构。ringbuffer 基本架构图如图 6-9 所示：

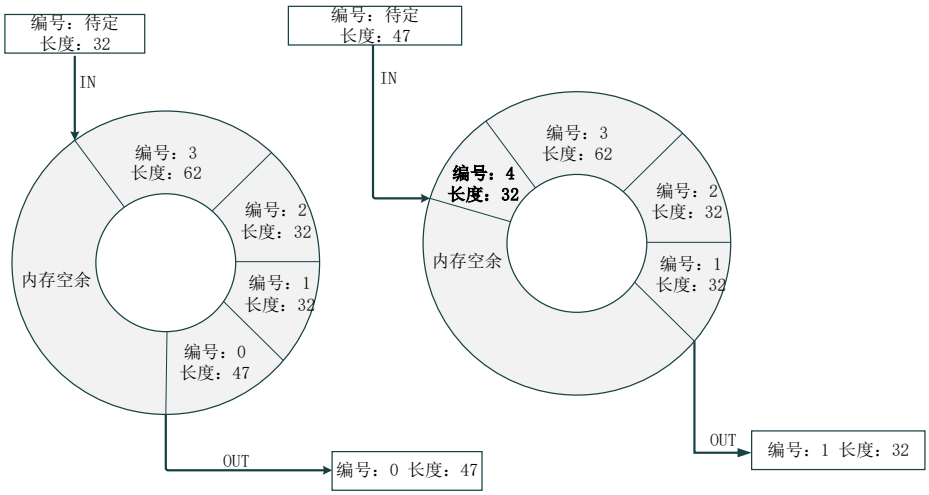

图 6-9 ringbuffer 基本数据结构图

首先，系统先申请一个大内存，基本大小为 10M，用于储存视频数据时，基本大小切换为 50M。随后申请一个循环链表，大小为设定的链表数，在应用中应取 25 以上。

在外部调用接口进行数据储存时，首先进入大内存区域，将数据完全复制进内存，随后记下其头指针以及数据块大小，存入当前循环链表中的元素，循环链表指向下一个空数据头指针保存区域。当写入数据大小超过大内存申请块的剩余内存时，会有两个解决方案，第一种会将数据会截取为两半，而保存的数据依然是其头指针位置以及数据块大小。读取的时候往往需要对这块部分做拼接。另一种是直接回到大内存头部开始写，浪费最后那块内存区域。在实际使用中，后者的效率更高点，因为前者虽然利用了最后一点内存，但是需要反复复制内存两次，而后者在读取时只要取得指针即可。

增加关键帧缓存，跳帧功能的 ringbuffer 数据结构

H264/H265 大致拥有三种类型帧，分别是 I 帧，P 帧和 B 帧。其中 I 帧是关键帧，P 和 B 帧需要依赖 I 帧才能生成画面。I 帧出现的频率由 gop_size 这个参数决定，一般设定为一秒的帧数 25，即 gop_size。I,P,B 帧与 gop_size 的关系如图 6-10 所示。

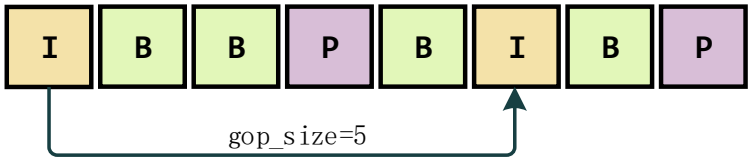

图 6-10 gop_size 示意图

Ringbuffer 数据结构本身没有判断是否关键帧的功能，每次取出的数据帧有可能是 I 帧，P 帧和 B 帧，而首帧不是 I 帧的情况下并不能形成画面。在某些平台上甚至无法形成画面。所以需要扩充 Ringbuffer 数据结构，增加 I 帧缓存功能。每次外部调用取数据前先判断取出的数据是否是 I 帧，如果是则直接输出，如果不是则输出最近的一个 I 帧，随后补上应有的数据，如图 6-11 所示。

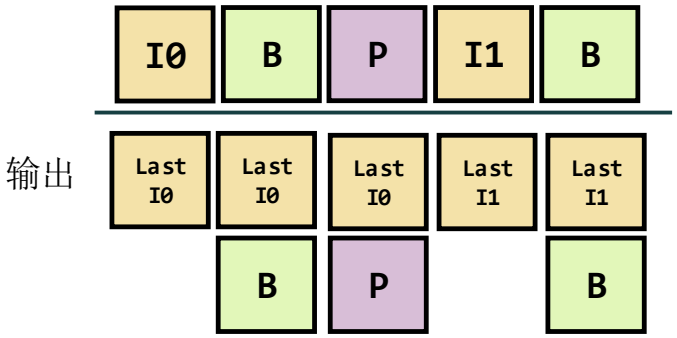

图 6-11 I 帧缓存示意图

在外部调用接口进行数据读取时，往往需要占用一个线程以及一个指针。必须首先判断存储指针是否与读取指针相同，以及判断读指针与存指针是否不在一轮内，即是否两者差距大于链表数，若大于链表数，则进行跳帧，跳帧幅度为一个链表数。若读指针与存指针在一轮内，则返回读指针在的位置，读取完毕后读指针向后一位。线程必须一直对链表进行读取工作，间歇读取将会开启跳帧设置。读取完毕一个数据包后将采用回调机制通知程序进行下一步操作。

在此建议使用多线程同步读取与写入，由此可以达成不错的异步效果。在多线程同步操作时，算法如图 6-12 与 6-13 所示：

Algorithm 3 RingBuffer: Put

Input: packet data sources:data , packet data length:len;
Output: Data Index

```
1:  procedure RingBufferPut(data, len)
2:      if listwriteptr ≥ buffercount :
3:          listwriteptr ← 0;
4:      endif
5:      buf[listwriteptr].length = len
6:      if totalptr + len ≥ bufferlength :
7:          totalptr ← 0;
8:      endif
9:      CopyMemory(totalptr + totalbuf, data, len);
10:     totalptr ← totalptr + len;
11:     buf[listwriteptr].packet ← totalbuf + totalptr;
12:     listwriteptr ← listwriteptr + 1;
13:     return listwriteptr;
```

图 6-12 avlist-put 代码

Algorithm 4 RingBuffer: Get

Input: Is Free Packet:freepacket;
Output: Packet:pkt;

```
1:  procedure RingBufferGet(freepacket)
2:      if listwriteptr–listreadptr ≥ 1 or (listwriteptr = 0 and listreadptr =
    bufferlength) :
3:          returnNone;
4:      else
5:          ret = buf[listreadptr];
6:          if freepacket = 0then
7:              if listwriteptr–listreadptr > QueuingMax :
8:                  for i ← listwriteptr + 1 to listreadptr − 1 :
9:                      if isFrame(buf[i]) :
10:                         listreadptr ← listreadptr + 1;
11:                     endif
12:                 end for
13:             else
14:                 listreadptr ← listreadptr + 1;
15:             endif
16:         endif
17:     endif
18:     pkt ← buf[listread];
```

图 6-13 avlist-get 跳帧代码

在算法中，将记录大内存指针的循环链表替换为了更加高效的循环线性表格式，通过简单的余数算法确保指针在可以控制的范围，避免出现内存溢出。在读取表数据的算法中，采用了简单的跳帧判断，即当读指针和写指针出现M/M/1 算法中算出的最大个数 N 以上的差距时就强制将读取指针前移 N-1 个到写指针的后面一位。当然，在实际应用中，这里涉及了 H264/H265 格式中关键帧判断算法，即前进的步数是距离写指针最近的关键帧位置，用户便不会因为关键帧的丢失看到短暂的灰屏。在没有跳帧的情况下，读包和写包的算法时间复杂度皆为 O(1),在跳帧的情况下，取决于关键帧判断算法，读包的最坏时间复杂度至多为 O(n)，最好时间复杂度为 O(1)。

6.5 Live555 核心代码改进

Live555 开源库在实际运行中存在问题，即无法判断网络断开，IP 摄像头断电等极端断开连接的情况，select 循环将会一直执行监听，而不是重新连接 IP 摄像头或判断错误类型。改进地点在 BasicTaskScheduler0 类中，singlestep()在网络异常断开情况下将不会退出子程，在内部循环继续侦听，如图 6-14 所示。

```
void BasicTaskScheduler0::doEventLoop(char* watchVariable) {
  // Repeatedly loop, handling readble sockets and timed events:
  while (1) {
    if (watchVariable != NULL && *watchVariable != 0) break;
    SingleStep();                                    //不会退出
  }
}
```

图 6-14 原 singlestep 代码图

深究其主要原因在 BasicTaskScheduler 类中的 singlestep 函数中。在 83 行代码如图 6-16，live555 在读取 socket 失败之后会进行重连。但是 RTSP 并不都是以 udp 的方式连接的，在以 tcp 连接的模式下便完全失效。所以最终选

择不再 singlestep()中完成重连，代码经过简单的修改后，给 singlestep 增加了一个 int 返回值，当连接断开后直接返回 1，其他情况返回 0，如图 6-15 与图 6-16 所示。

```
    int selectResult = select(fMaxNumSockets, &readSet, &writeSet,
&exceptionSet, &tv_timeToDelay);
    if (selectResult < 0) {
      int err = WSAGetLastError();
      if (err == WSAEINVAL && readSet.fd_count == 0) {
        err = EINTR;
        int dummySocketNum = socket(AF_INET, SOCK_DGRAM, 0);
        FD_SET((unsigned)dummySocketNum, &fReadSet);
    }
```

图 6-15 原重连代码图

```
    int selectResult = select(fMaxNumSockets, &readSet, &writeSet,
&exceptionSet, &tv_timeToDelay);
    if (selectResult < 0) {
       return 1;
    }
```

图 6-16 更改后的重连代码图

Singlestep 经过改造后多了一个返回值，随后在 BasicTaskScheduler 的 doeventloop 函数中增加一个判断，如果 singlestep 返回值大于 0 则退出 loop 循环，如图 6-17 所示。

至此修复了 live555 的网络断开接收不正确的问题。

```
    void BasicTaskScheduler0::doEventLoop(char volatile* watchVariable) {
      while (1) {
        if (watchVariable != NULL && *watchVariable != 0) break;
              if (SingleStep() > 0){
                    break;
              }}}
```

图 6-17 更改后的 singlestep 代码图

6.6 本章小结

本章介绍了实现低延时视频监控的几个关键技术，首先将实时视频服务抽象成了 M/M/1 模型，随后仔细分析了队列长度与延时的影响，随后研究了 H264，H265 的帧头判断算法，关键帧算法；并根据模型结论与帧头判断算法提出了带有 I 帧缓存的特殊 RingBuffer 数据结构，最后改进了 LIVE555 中接收数据的核心代码，让其更加贴合系统。

第七章 系统测试

在完成各个程序的设计与运行后，进入系统整合运行。系统整合运行主要由添加摄像头到数据库，启动数据库中间件，启动视频服务器，随后打开客户端进行查看。

7.1 测试环境

测试设备主要由三个 IP 摄像头，一台服务器，两台电脑，一部手机组成。

IP 摄像头为三个不同品牌的不同信号，采取的连接原理不同，三台摄像头参数如表 7-1 所示，三个摄像头如图 7-1 所示。

摄像头编号	分辨率	码率(kbps)	编码方式	传输方式
A	1280x720	2048	h.265	有线
B	1920x1080	3214	h.264	有线
C	1280x720	1538	h.264	无线

表 7-1 摄像头参数表

图 7-1 实验摄像头

服务器端安装在一台 ubuntu 操作系统的 PC 机上。Windows 客户端安装在一台 windows 操作系统的 PC 机上，采用 i3cpu，16G 内存；mac 客户端安装在一台 mac 电脑上，mac osx 操作系统为 el captain。Android 客户端安装在一台安卓手机上，运行 android5.0 系统。

网络环境为实验室局域网，带宽为 100Mbps。各终端间通过路由器进行通讯。采用有线与无线混合连接。客户端同时请求三路 IP 摄像头视频，并在本地播放。不同平台下的客户端将同时登录服务器查看视频流。

在观看实时视频之前，必须将视频监控设备添加入数据库中。我们在此使用手动添加的方式。首先将视频监控设备连入指定网段使用 nodejs[31]-onvif 套件读取设备的信息，例如 RTSP 地址，摄像头 IP 地址，用户名，密码等。用户只要使用网页登录/search 目录即可查看当前网段下所有视频监控设备的 RTSP 地址。在获取到 RTSP 地址后，管理员使用 sqlite 管理器手动将信息添加到数据库中间件，并重启数据库中间件服务器。管理器在本例中使用 sqlitespy。

配置完数据库后，双击启动数据库中间件程序。启动完毕后根据数据库中间件提供的 ip 地址对视频服务器的配置 ini 进行简单的配置。本例中数据库中间件的地址是 192.168.1.106，则在视频服务器的配置 ini 中，填入该地址并保存。写入完成后输入命令开启视频服务器。

在开启视频服务器后，双击打开视频客户端，在输入用户名和密码后进入视频查看界面，点击之前已经添加的摄像头，将出现添加的摄像头传来的画面，如图 7-2 所示。至此系统整合运行所有流程结束。

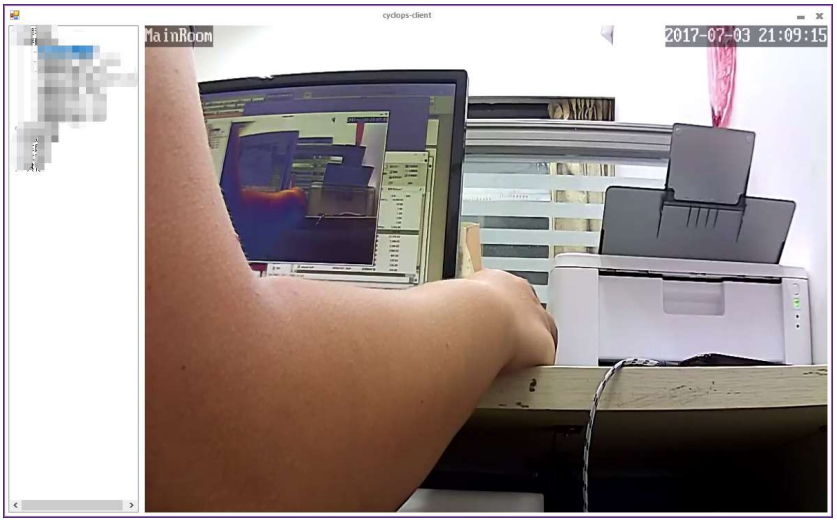

图 7-2 桌面客户端查看视频流图

本系统除了 windows 平台客户端以外，还有 android 平台客户端，mac osx 平台客户端与网页客户端。其中 mac osx 平台客户端和网页客户端没有读取列表等功能，只有视频播放显示核心模块的移植。以下将简单介绍各平台客户端的运行情况。

7.2 客户端运行

移动客户端运行

移动客户端共有 4 个子模块，它们分别是：登录模块，设置服务器模块，视频设备列表查询模块，监控视频观看模块。其中登录模块较为简单，只要一个输入用户名的输入框，一个输入密码的输入框和选定的 IP 地址滚动菜单就行。程序界面如下图所示，其中设置服务器指向设置服务器模块，登录模块界面如图 7-3 所示。

图 7-3 登录模块界面

图 7-4 设置服务器界面

设置服务器模块主要用于预存储服务器信息，用户在该页面分别填写完整服务器名称，IP 地址，端口号，登录的用户名和密码后即可增加进存储库中。同时还可以相同的步骤修改项目和删除项目。模块图如图 7-4 所示。视频设备列表查询模块主要用于将服务器查询到的视频监控摄像头以及他所在的分组呈现在用户面前，并通过用户的触摸选择打开分组或者调用摄像头。该界面清晰

直观，用户可以轻易选择自己想要观看的摄像头。视频设备列表查询界面如图7-5所示。

图7-5 视频设备列表查询界面

图7-6 监控视频播放模块

监控视频播放模块是核心模块，他用于查看流媒体服务器传来的视频流并实时播放。为了兼容不同机型，该模块为用户准备了三种引擎。分别为兼容模式引擎，适合安卓低版本用户，稳定模式引擎，适合安卓中高版本用户以及硬解码引擎，适合安卓高版本用户。监控视频播放模块界面如图7-6所示。

Mac osx 平台客户端运行

Mac 平台下运行的是视频核心显示模块，通过命令打开视频监控，如图7-7所示。

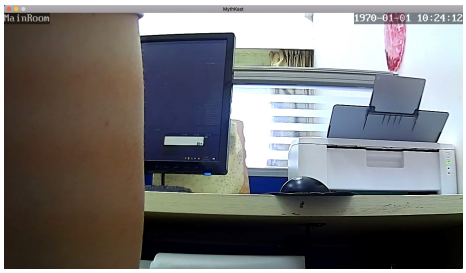

图7-7 MAC 平台客户端播放界面

图7-8 linux 数字矩阵画

Linux 下数字矩阵

Linux 系统下的移植为一个数字矩阵，从左至右分别显示了A,B,C,D四个摄像头的四个实时画面。Linux 系统为Ubuntu14.04，使用Opengl显示实时画面。如图7-8所示。

7.3 视频监控低延时测试

本文将分别在进行三个实验以测试低延时系统的性能，测试包括局域网中的测试，网络模拟器中的测试以及真实网络环境中的测试。其中推流器将在真实网络环境被测试。

7.3.1 局域网测试与结果

低延时测试需记录从 IP 摄像头拍摄到画面开始到桌面客户端显示画面结束的时间。为此，本文设计了一套特殊的低延时测试环境记录数据。测试由一个 IP 摄像头，一台视频服务器，一个手持计时器，一个桌面客户端组成。所有设备都位于一个局域网的同个路由网段内。测试环境拓扑图如图 7-10 所示。

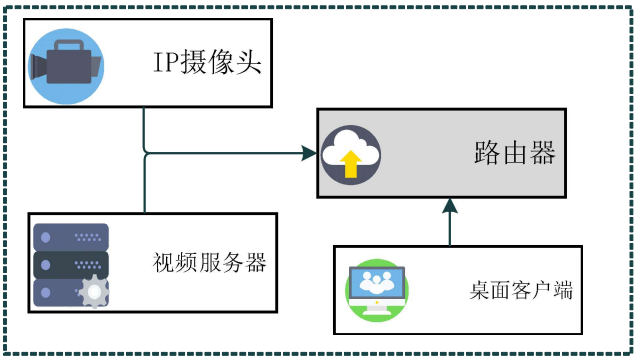

图 7-10 测试环境拓扑图

详细测试方法为，将手持计时器放置在桌面客户端前，并将 IP 摄像头对准桌面客户端，IP 摄像头将拍下桌面客户端与手持计时器上显示的时间。视频服务器将使用录像功能录下实时视频流，最后保存为 FLV 文件。测试时间为 30 分钟，录像采用 5 分钟一个切片，一共 6 个文件的方式存储。

图 7-11 录像视频截图

随后分析录像文件，每 10 秒钟进行一次截图，并计算桌面客户端屏幕上的时间与手持计时器记录的时间之差，录像部分采样如图 7-11 所示。

为避免采样过于密集，本文采用随机采样 300 个点作为采样数据。采样数据格式将表示为 P_n（T_n，D_n）。其中 T 代表该数据当前时间，D 代表客户端显示时间 t_0 与计时器时间 t_1 之差，即延时 D。实验在各种网络情况下将 srs，nginx-rtmp-module 以及本文方案进行延时的对比。其中，基于 RTMP 的方案使用开源插件 ProxyProtocol 进行摄像头视频流收集。

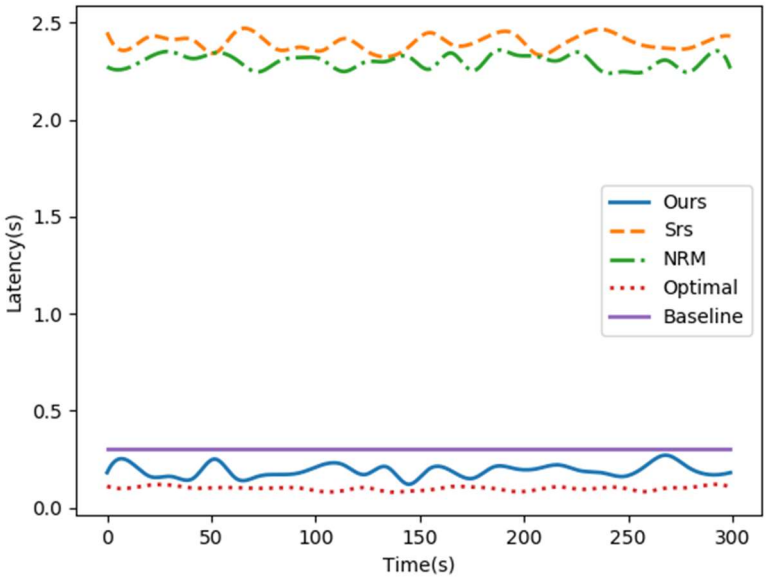

图 7-12 局域网下低延时测试结果图

本文在部署的局域网下对比了三个方案，测试结果如图 7-12 所示。由图可知，我们的方案延时皆低于标准值 300 毫秒，平均值在 189 毫秒左右。符合低延时定义，而其余两个方案平均时间都高于 300 毫秒，并且画面延迟抖动相对较大。故我们的方案在低延时特性上优于市面上的流媒体直播方案。

7.3.2 模拟环境下对不同网络情况的测试与测试结果

本文在模拟环境下对不同的网络情况对三个方案进行了低延时测试。模拟环境使用了 Cellsim[29]，一种根据饱和带宽测试网络延迟与发送速率的工具。Cellsim 的原理与改进将文后详细叙述。实验分别测试了有线网络，无线网络，以及 4G 网络。实验记录了整个会话的平均延时。

Cellsim 原理与改进

为了测试我们的模型，我们更加注重排队延迟而不是单向延迟。所以，我们的模拟器应该模拟在不同网络条件下进出数据包的过程，并跟踪时间戳，从而获得相应的排队延迟。受 Mahimahi[30]与 Cellsim 的启发，我们使用饱和网络带宽记录来生成排队延迟数据。如图 7-13 所示，假设数据包到达和离开的分布符合泊松过程，我们使用发送饱和网络轨迹中的比特率和带宽作为到达率 λ 并分别保留利率 μ 。

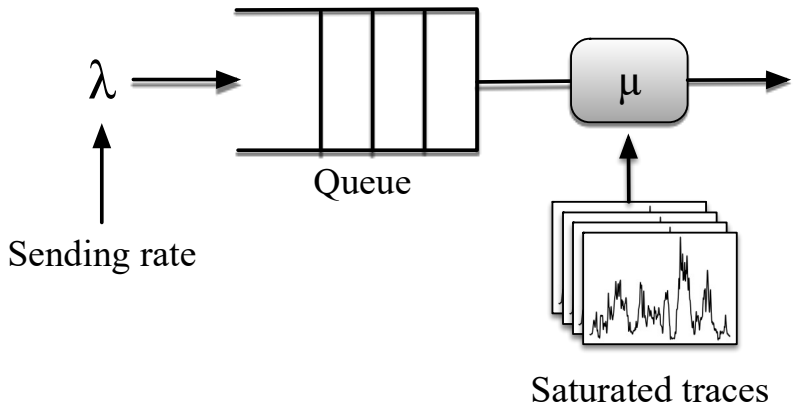

图 7-13 Cellsim 原理与结构图

具体而言，我们将发送者和接收者之间的连接看作由单个设备转发，并且设备产生的排队延迟被称为'自我延迟'(self-inflected delay)。该设备由三个队列组成，分别命名为'input'，'output'和'limbo'。一种算法被设计为基于相应的分组级递送轨迹从每个队列释放分组。队列中的每个元素都是数据包传送的传送概率，这意味着 MTU 大小的数据包将被传送的时间（以毫秒为单位）。对于每个周期（t_1, t_2），离线网络模拟器比较'input'中的前端数据包与'limbo'中的前端数据包的时间戳。如果'input'队列中的时间戳大于或等于模拟器会将前端数据包推入“输出”队列的底部，然后计算这两个数据包之间的延迟梯度，相反，如果“输入”队列中的时间戳记输出机会将被浪费比'limbo'队列中的队列小，模拟器只会将前端数据包弹出'limbo'队列，通过这个过程，模拟器返回平均自我延迟和总字节数。

饱和带宽数据收集

为了评估我们提出的系统，我们必须做的第一件事就是生成饱和网络跟踪数据集。然而，这些类型的网络痕迹很难被记录，即使是公共数据集也非常有限。例如，Cellsim 仅提供少量饱和网络轨迹，描述蜂窝网络状况而不是所有网络环境，这几乎不能使我们的神经网络收敛。因此，我们考虑以两种方式收集数据集：

- 包级网络饱和带宽[52]：我们从 2018 年 1 月收集的快手的所有平台应用程序中使用专有的数据包级实时会话状态数据集。快手是中国的领先平台，在全球拥有超过 7 亿用户，并且发布了数百万原始视频在它每天。在 Ledbat 提出的单向延迟估计方法的驱动下，我们从分组数据集生成 2300 条真实网络轨迹。
- 块级网络带宽[53]：我们还收集由不同网络数据集组成的混合网络轨迹数据集，例如 FCC 数据集和 Norway 数据集。 FCC 数据集是宽带数据集，Norway 数据集主要收集在 3G / HSDPA 环境中。简而言之，我们从数据集中生成 1,000 条网络轨迹。
- 合成网络饱和带宽数据集[53]：我们使用马尔可夫模型生成一个合成数据集，其中每个状态代表上述范围内的平均吞吐量。因此，我们创建了一个包含一组网络条件的超过 500 条饱和带宽数据集。

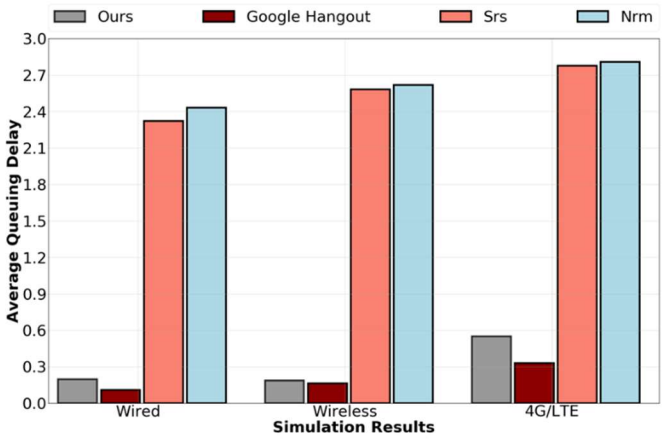

图 7-14 各种网络情况下的平均排队延时图

实验结果

根据实验测试内容，结果如图 7-14 所示。实验新增了一个基于 UDP 的方案 Google Hangout，该方案优势在于使用 UDP 协议获得更低的延时。由由图可知，我们的方案延时在各个环境表现都比使用 TCP 协议的方案都要低，而且只比使用 UDP 方案的 Google Hangout 略微高。甚至在有线网络下，我们的方案比各个方案都占优。

7.3.3 推流器性能测试与结果

本文随后比较了推流器与传统 RTMP 服务的性能。推流器推送架构如图 7-15 所示，推流器将视频流通过 TCP 协议推送给服务器，随后服务器再分发给客户端。RTMP 推送架构如图 7-16 所示，RTMP 推流端将通过 UDP 协议将 RTMP 视频流发送给 RTMP 服务器，并且分发给 RTMP 客户端。在这里，我们将 RTMP 推流端从 TCP 协议设置为 UDP 协议，以体现出我们推流器性能。实验分别比较了有线网络，无线网络和 4G 网络下两种架构的延时。服务器位于山东青岛，RTT 为 120ms。两种推送器从贵州贵阳教育网将摄像头视频流实时推送给服务器，并让服务器将视频流发送回贵州贵阳的另一个客户端内。与实验 1 相同，我们记录了视频上显示的延时。

实验结果如图 7-17 所示，文中提出的推流器在各个网络情况中的表现都优于 RTMP 提出的架构，且平均性能提升了 68%，最高提升于无线网络中，性能提升了 63%。由此可知推流器达到了设计需求。

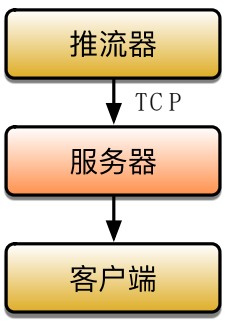

图 7-15 推流器推送流程及架构

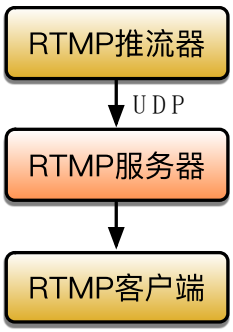

图 7-16 RTMP 推流器推送流程及架构

Network	Ours(s)	RTMP(s)	Improvement(%)
Wired	0.37	1.32	71.97%
Wireless	0.40	1.58	74.68%
4G/LTE	0.51	1.40	63.57%

图 7-17 推送器性能实验结果

7.4 本章小结

本章搭建了一个真实环境，使用了三个不同品牌不同规格的测试摄像头进行测试。在使用 nodejs-onvif 插件添加摄像头后，在桌面客户端和移动客户端中都看到了流畅的画面。随后进行了低延时测试，经过测试，画面与现实的延时在 300 毫秒以内，满足低延时要求。

第八章 总结与展望

8.1 研究工作总结

本文重点研究了在今天的互联网，视频直播业务与视频监控行业的迅速发展与融合的背景下，直播架构与视频监控目标在于提出一个新的视频监控系统架构让其同时拥有高并发性与低延时特性。本文结合高并发特性提出并实现了一个新的基于 H.264 与 H.265 的低延时视频监控系统。通过该系统，用户可以通过不同平台客户端低延时观看市面上大多数支持 RTSP 协议的摄像头的实时视频流。

比对流媒体直播架构有实时视频监控架构是构造低延时视频监控系统的前提。在本文中介绍了流媒体技术以及数字视频监控技术中的核心技术，同时提出了一种新的视频监控系统架构方法，同时满足了流媒体系统中的高并发特点与视频监控系统的低延时特性。

本文之后根据提出的架构方案研究并实现了完整的视频监控系统，该系统包括视频服务器，移动客户端，桌面客户端，视频推流器与数据库中间件。随后将系统的模块逐一设计并实现。

低延时实时监控系统需要新的算法支撑。本文研究了多种符合需求的特殊数据结构与算法并将其加入了系统，内容包括：H.264 与 H.265 在低延时视频传输中关键部分，如：改进的帧头判断算法；改进的 RingBuffer 数据结构；探究关键帧帧缓存数量对视频低延时的影响；改进 Live555 开源库的代码等。

最后，本文实现了一个测试环境用于验证与评价提出的系统。测试结果表明：低延时视频监控系统设计方案是可行的，系统符合低延时的评价标准。本系统全套代码已在码云（oschina）开源，网址为：

https://gitee.com/godka/mythmultikast

8.2 研究工作展望

本系统研究与实现了新的基于 H.264 与 H.265 的低延时视频监控系统。作者认为未来需要进一步研究的内容包括：

[1] 实际场景中，实时监控直播经常出现期望编码码率与实际编码码率不同的情况，而且有时相差较大，最终会影响低延时特性。于是如何为编码器设定一个可以接收的编码码率范围而不是一个值成为了解决该问题的关键。

[2] 在视频直播中不能盲目追求低延时，因为低延时也意味着低传输速度，低码率。为此，视频的画面质量与延时之间应当存在一个均衡，能够使用户在较低延时的体验下观看较清晰画面质量的视频直播流。延时与当前网络状态有关，画面质量与当前视频呈现的内容有关，两者如何取较优值将是下一个研究方向。

致谢

在论文完成之际，我首先衷心感谢导师李泽平教授对我硕士期间学习和研究工作的精心指导，从最初的论文选题到实验，到最后的写作定稿，都离不开李老师耐心的指导和无私的帮助。李老师严谨的治学态度和开朗的人生态度，不但是我在校学习和从事研究的良师，更使我在做人做事方面受益匪浅。

其次，我还要感谢李泽平实验室的其他老师和同学在学习和研究工作中给予的帮助和指导。在校的学习生活即将结束，回顾两年多的学习经历，在你们的帮助下我收获颇丰，感到无限欣慰，这段经历将使我终生难忘。

感谢我的家人和朋友，在我攻读硕士阶段给予的支持和鼓励。

最后，衷心地感谢在百忙之中参与论文评阅和答辩工作的各位专家、教授。

参考文献

[1] 骆云志，刘治红. 视频监控技术发展综述[J]. 兵工自动化，2009，28(1):1-3.

[2] 刘伟琪. 网络视频监控技术发展现状与展望[J]. 无线互联科技，2016(14):37-38.

[3] 叶斌. 视频监控技术发展与现状[J]. 中国传媒科技，2012(12):235-236.

[4] 徐琛. 视频监控技术的发展与现状[J]. 产业与科技论坛，2011，10(18):66-66.

[5] 信师国，刘庆磊，刘全宾. 网络视频监控系统现状和发展趋势[J]. 信息技术与信息化，2010(1):23-25.

[6] 吕世良，王晓茜，刘金国. 数字视频监控系统设计与实现[J]. 测控技术，2014，33(2):80-82.

[7] 冷俊. 基于 C/S 结构的远程数字视频监控系统[J]. 通信世界，2004(36):26-27.

[8] 孙兵剑，朱伟兴. 基于 H.264 的猪舍实时视频监控系统的研究[J]. 信息技术，2016(6).

[9] 吴大中，胡江浪，WUDazhong,等. 基于 ARM 的移动视频监控系统设计[J]. 现代电子技术，2016，39(4):123-127.

[10] 陈乐，吴蒙. 基于 OMAP 和 Gstreamer 的网络视频监控系统[J]. 计算机技术与发展，2016，26(9).

[11] 陈张荣，贾俊铖，严建峰. 基于嵌入式系统的网络视频监控系统设计[J]. 仪表技术与传感器，2016(2):39-41.

[12] 郝卫东，李静. 基于 Linux 的嵌入式网络视频监控系统研究与设计[J]. 计算机系统应用，2008，17(8):69-72.

[13] 惠晓威，王克. 移动视频监控系统的实现[J]. 计算机应用与软件，2014(1):148-150.

[14] 张杰. 基于嵌入式 Linux 和 H.264 的视频监控系统[D]. 浙江工业大学，2014.

[15] 张鹏. 基于 MPEG-4 的嵌入式远程视频监控系统的设计[D]. 太原理工大学，2008.

[16] 王溢琴，秦振吉，芦彩林. 基于嵌入式的智能家居之视频监控系统设计[J]. 计算机测量与控制，2014，22(11).

[17] 王中杰，WANGZhong-jie. 实验室智能视频监控系统开发[J]. 自动化技术与应用，2016，35(5):115-117.

[18] 杨通清，胡琦. 超大规模建筑的数字视频监控系统建设[J]. 智能建筑与智慧城市，2016(8).

[19] 刘凯. 基于 LINUX 与 H.264 的安全视频监控系统[J]. 网络安全技术与应用，2016(6).

[20] 方浩，李艾华，王涛. 基于 DM6437 的智能视频监控系统设计与实现[J]. 计算机应用与软件，2016(2):192-196.

[21] 孙宏军，赵作霖，徐冠群. 塔式起重机机器视觉监控系统设计[J]. 传感器与微系统，2016(8).

[22] 张国玲, 范颖. 视频监控中的考生异常行为识别[J]. 控制工程, 2016, 23(4).

[23] 刘殊. 基于 Hadoop 的分布式云监控平台系统的研究与设计[J]. 电子设计工程, 2016, 24(15).

[24] Romdhane R, Mulin E, Derreumeaux A, et al. Automatic video monitoring system for assessment of Alzheimer's Disease symptoms[J]. The journal of nutrition, health & aging, 2012, 16(3):213-218.

[25] Zhang B L, Chang S J, Li J W, et al. Intelligent control of video monitoring system based on the color histogram analysis[J]. Medical Oncology, 2006, 55(12):6399-6404.

[26] Vol. N. A Novel Mobile Video Monitoring System Using a PDA Terminal[J]. Ieice Transactions on Communications, 2002, 85(10):2191-2197.

[27] Hsieh J W, Hsu Y T, Liao H Y M, et al. Video-Based Human Movement Analysis and Its Application to Surveillance Systems[J]. IEEE Transactions on Multimedia, 2008, 10(3):372-384.

[28] Jiao L, Wu Y, Wu G, et al. Anatomy of a multicamera video surveillance system[J]. Multimedia Systems, 2004, 10(2):144-163.

[29] Winstein K, Sivaraman A, Balakrishnan H, et al. Stochastic forecasts achieve high throughput and low delay over cellular networks[C]. networked systems design and implementation, 2013: 459-471.

[30] Netravali R, Sivaraman A, Das S, et al. Mahimahi: accurate record-and-replay for HTTP[C]. usenix annual technical conference, 2015: 417-429.

[31] Tilkov S, Vinoski S. Node.js: Using JavaScript to Build High-Performance Network Programs[J]. IEEE Internet Computing, 2010, 14(6): 80-83.

[32] FFmpeg.Fast Forward mpeg [EB/OL].https://ffmpeg.org

[33] SDL2.Simple Direct Layer[EB/OL].https://libsdl.org

[34] Live555. [EB/OL].https://live555.com

[35] Akhlaq M, Sheltami T R. RTSP: An Accurate and Energy-Efficient Protocol for Clock Synchronization in WSNs[J]. IEEE Transactions on Instrumentation and Measurement, 2013, 62(3): 578-589.

[36] Jian Huang, DongMei Wu and XiaoPei Liu, "Implementation of the RTMP server based on embedded system," Computer Science and Information Processing (CSIP), 2012 International Conference on, Xi'an, Shaanxi, 2012, pp. 160-162.

[37] HU Guoqiang, ZHOU Zhaoyong, XIN Zhaoxia. Design and implementation of open source live system based on SRS. Modern Electronics Technique, 2016, pp. 36-39+43.

[38] P. Zhao, J. Li, J. Xi and X. Gou, "A Mobile Real-Time Video System Using RTMP," Computational Intelligence and Communication Networks (CICN), 2012 Fourth International Conference on, Mathura, 2012, pp. 61-64.

[39] X. Lei, X. Jiang and C. Wang, "Design and implementation of streaming media processing software based on RTMP," Image and Signal Processing (CISP), 2012 5th International Congress on, Chongqing, 2012, pp. 192-196.

[40] H. T. Le, H. N. Nguyen, N. Pham Ngoc, A. T. Pham, H. Le Minh and T. C. Thang, "Quality-driven bitrate adaptation method for HTTP live-streaming," 2015 IEEE International Conference on Communication Workshop (ICCW), London, 2015, pp. 1771-1776.

[41] Pantos R,May E W.HTTP Live Streaming draft-pantos-httplive-streaming-20. https://tools.ietf.org/html/draft-pantos-http-live-streaming - 16 . 2015

[42] 基于 Android 的 HLS 播放器的实现和优化 T. C. Thang, H. T. Le, A. T. Pham and Y. M. Ro, "An Evaluation of Bitrate Adaptation Methods for HTTP Live Streaming," in IEEE Journal on Selected Areas in Communications, vol. 32, no. 4, pp. 693-705, April 2014.

[43] 王瑞,杨杰,唐鼎. 基于 Android 和嵌入式 Web 的视频监控系统的研究与实现[J]. 电子设计工程,2016,08:191-193.

[44] ZHANG Yongqiang, YU Boping, et al. Design and Implementation of a Campus Network TV System Supported Mobile Devices. Journal of Wuhan University (Natural Science Edition). 2012, pp. 365-370.

[45] P. Chakraborty, S. Dev and R. H. Naganur, "Dynamic HTTP Live Streaming Method for Live Feeds," 2015 International Conference on Computational Intelligence and Communication Networks (CICN), Jabalpur, 2015, pp. 1394-1398.

[46] 徐祥男,富坤,罗淑贞,耿恒山,耿跃华. HLS 直播流媒体传输系统的冗余优化[J]. 电视技术,2014,16:61-64.

[47] 李云飞,谢伟凯,鲁晨平,张智强,申瑞民. 面向直播 HTTP Streaming 系统的 HTTP 缓存服务器行为优化[J]. 计算机工程与应用,2012,10:68-74.

[48] 罗淑贞,耿恒山,徐祥男,孙豪赛,高艳,李钦,谢因. 基于 HLS 协议的流媒体直播系统的研究和改进[J]. 郑州大学学报(工学版),2014,05:36-39.

[49] Jin F. Design and Implementation Video on Demand System Based on FMS +FLV[C]. international conference on genetic and evolutionary computing, 2010: 398-401.

[50] Sullivan G J, Ohm J, Han W, et al. Overview of the High Efficiency Video Coding (HEVC) Standard[J]. IEEE Transactions on Circuits and Systems for Video Technology, 2012, 22(12): 1649-1668.

[51] Sullivan G J, Topiwala P, Luthra A, et al. The H.264/AVC Advanced Video Coding Standard: Overview and Introduction to the Fidelity Range Extensions[J]. Proceedings of SPIE, 2004: 454-474.

[52] Tianchi Huang, Rui-Xiao Zhang, Chao Zhou, and Lifeng Sun. 2018. Delay-Constrained Rate Control for Real-Time Video Streaming with Bounded Neural Network. In Proceedings of MMSys 2018 Nossdav Workshop (NOSSDAV'18). Amsterdam, The Netherlands, June 15, 2018

[53] Mao H, Netravali R, Alizadeh M. Neural adaptive video streaming with pensieve[C]//Proceedings of the Conference of the ACM Special

Interest Group on Data Communication. ACM, 2017: 197-210.

[54] 巴堃. 监控视频低延迟 H.264 编码关键技术研究[D].上海交通大学,2013.

[55] Carlucci G, De Cicco L, Holmer S, et al. Analysis and design of the google congestion control for web real-time communication (WebRTC)[C]. acm multimedia, 2016.

附录 I 攻读硕士学位期间取得的研究成果

攻读硕士学位期间的获奖情况

[1] **一等奖学金**，贵州大学，2016（1st）

[2] **国家奖学金**，贵州大学，2017（top 2%）

[3] **校级优秀毕业生**，贵州大学，2018

发表的学术论文

[1] **The author**, 李泽平, Jun Zhang. 2018. Design and implementation of living streaming system based on multi-service nodes collaboration. IAEAC'17, Chongqing, China, April 18, 2018（EI 收录，已接收）

[2] **The author**, Rui-Xiao Zhang, Chao Zhou, and Lifeng Sun. 2018. Delay-Constrained Rate Control for Real-Time Video Streaming with Bounded Neural Network. In Proceedings of MMSys 2018 Nossdav Workshop (NOSSDAV' 18). Amsterdam, The Netherlands, June 15, 2018(EI 收录，已接收，CCF-B 类)

专利软著

[1] **黄天驰**，王磊：掌上监控：中国，软件著作权

[2] **黄天驰**，王磊：mythClient 监控 PC 客户端软件：中国，软件著作权

[3] **黄天驰**，王磊：掌上监控服务器端软件：中国，软件著作权

[4] **黄天驰**，李泽平：浏览器中无插件播放实时监控的方法：中国，专利受理

参与的课题研究

[1] 基于对等网的可扩展流媒体分发模型研究. 国家自然科学基金（项目编号：61462014）

[2] P2P 流媒体分发服务关键技术研究. 贵州省优秀科技教育人才省长专项基金（项目编号：黔省专合字（2011）34 号）

[3] P2P 流媒体分发与服务关键技术研究. 贵州省科技厅基金（项目编号：黔科合 J 字[2011]2201 号）

[4] 多 CDN 自适应流媒体分发的用户体验与网络资源联合优化研究. 国家自然科学基金（项目编号：61472204）

[5] 网络可视媒体智能处理. 国家自然科学基金（项目编号：61521002）

[6] 网络多媒体北京市重点实验室. Z161100005016051

图版

表板