电子科技大学

UNIVERSITY OF ELECTRONIC SCIENCE AND TECHNOLOGY OF CHINA

硕士学位论文

MASTER THESIS

论文题目 基于RTMP协议的流媒体系统的设计实现

学科专业 计算机系统结构

学　　号 201221060228

作者姓名 张　印

指导教师 鲁晓军　副教授

独创性声明

本人声明所呈交的学位论文是本人在导师指导下进行的研究工作及取得的研究成果。据我所知，除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得电子科技大学或其它教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示谢意。

作者签名：张印 日期：2015年12月28日

论文使用授权

本学位论文作者完全了解电子科技大学有关保留、使用学位论文的规定，有权保留并向国家有关部门或机构送交论文的复印件和磁盘，允许论文被查阅和借阅。本人授权电子科技大学可以将学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存、汇编学位论文。

（保密的学位论文在解密后应遵守此规定）

作者签名：张印 导师签名：

日期：2015年12月28日

分类号 _____ 密级 _____

UDC [注1] _____

学 位 论 文

基于 RTMP 协议的流媒体系统的设计实现

（题名和副题名）

张 印

（作者姓名）

指导教师 鲁晓军 副教授

电子科技大学 成 都

（姓名、职称、单位名称）

申请学位级别 硕士 学科专业 计算机系统结构

提交论文日期 2015.09.23 论文答辩日期 2015.09.25

学位授予单位和日期 电子科技大学 2015 年 12 月 26 日

答辩委员会主席 _____

评阅人 _____

注 1：注明《国际十进分类法 UDC》的类号。

Design and Implementation of Streaming Media System Based on Rtmp Protocol

A Master Thesis Submitted to

University of Electronic Science and Technology of China

Major: **Computer Architecture**

Author: **Yin Zhang**

Supervisor: **Associate Professor XiaoJun Lu**

School: **Computer Science and Engineering**

摘 要

随着现代人们生活质量的提高，网络通信技术以及带宽技术的日益成熟化，流媒体服务器的应用也与日俱增，流媒体技术具有连续性和传输性，即流媒体技术可以利用网络实时的传递音频和视频数据，是优于其他传统多媒体下载传输的技术。流媒体技术突破了 TCP/IP 网络的限制，飞速发展，在视频点播直播、网络会议，智能家居，小区安全监测等等领域都发挥了重要的作用。已经成为了与人们日常生活息息相关的多媒体技术，经过几年的高速发展，Linux 系统与 Android 系统各方面性能都趋于完善，便于流媒体服务器的架构与移植。本工程涉及到的是视频监控领域，不仅仅要讨论流媒体服务器的涉及和实现，还要研究视频数据的采集和压缩。FFMPEG 则提供了以系列的音视频编解码工具。

结合以上讨论，以 Linux 系统和 Android 系统为平台，设计了一个基于 RTMP 协议的多平台流媒体视频监控服务器。在原有的 FFMpeg 源码的基础上，对其进行编译移植，以实现视频数据的采集压缩，同时本文针对 RTMP 协议架构，结合 Linux 系统和 Android 系统的特点，对ＲＴＭＰ基本原理、协议结构、核心技术以及在 Linux 系统和 Android 操作系统中的方案设计、工作流程、代码实现等进行了阐述和深入研究。综上所述，本文中主要做了以下几个工作：

1）、研究了 Linux 系统和 Android 系统架构以及不同平台的移植差异性，和流媒体服务器在不同平台的启动方式。

2）、研究了 FFMpeg 源码，熟悉 FFMpeg 源码内相关的数据结构，研究了其编解码流程及相关文件，在此基础上对 FFMpeg 源码进行编译移植到不同平台。

3）、流媒体客户端的设计实现。

4）、研究了 RTMP 实时消息传输协议，熟悉其每个工作流程并研究其每个工作过程所涉及的报文细节，了解每次通信的报文格式和报文含义等，并在此基础上结合 Linux 系统和 Android 系统的特点实现流媒体视频监控服务器。

关键词：FFMPEG,RTMP,Linux,Android,视频监控

ABSTRACT

With the enhancement of quality life, the popularization of information technology and the maturation of bandwidth, the application of streaming media has been enjoying an ascending status. The streaming media, possessing the consecutiveness and transmission features, is able to transmit audio and video data outweighing other traditional multimedia downloading transmission technology, meanwhile, it breaks the limits of TCP/IP network, plays a keynote role in request broadcasting, call conference, smart home and residency security monitoring, and permeates people's daily life. After year's development, the Linux and Android system has stepped onto a desirable stage, on which the construction and transplanting of media server is easily conducted. This project mainly relates to video monitoring, not only discussing the streaming media's involvement and accomplishment, but also researching into video data collecting and compressing, in which

FFMPEG offers series of audio and video decoding tools.

Based on the above discussion, the thesis designs a multi-platform Streaming Media Video Monitoring Server based on RTMP Protocol on both Linux and Android system. Besides, compiling and grafting are conducted on the basis of the existed FFMPeg source code to realize video data collecting and compressing. In the meantime, targeting at the structure of RTMP Protocol, the thesis penetrates into the basic theory, structure of protocol and core technology of RTMP and program designing, working flow and code implementation in Linux and Android system. The main procedures are as follows:

1)、Study system structure of Linux and Android, grafting differences and various activating manners of Streaming Media Server on different platforms;

2)、Explore FFMPeg source code to learn related data structure and the work flow of encoding and decoding and related documents to compile and graft the FFMPeg source code onto different paltforms.

3)、Desing and implement Streaming Media client

4)、Go through RTMP real-time information transmitting protocol to analyze each work flow and details of messages, including forms and contents, and implement Streaming Media Video Monitoring Server with features of Linux and Android system.

Keywords: RTMP, FFMPEG, Android, Linux,Streaming Media

目 录

第一章 绪论

1.1 课题研究背景

RTMP的全称为Real Time Messaging Protocol(实时消息传输协议），它是由Adebe 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。它是一个基于TCP的一个协议簇，其中包括RTMP基本协议以及RTMPT/RTMPE/RTMPS等多种协议变种，RTMPE协议是在RTMP协议的基础上增加了加密的功能，RTMPT协议是封装在HTTP请求之上的，可以穿透防火墙，RTMPS协议类似RTMPT，它增加了TLS/SSL的安全功能，而RTMP协议是工作在TCP之上的明文协议，使用的端口为1935，该协议有以下3个特点：

1）、RTMP协议是用于对网络音视频数据的实时传输，在TCP/IP协议的基础上进行的网络封装。

2）、RTMP协议可以理解为是一个封装协议，即按照RTMP协议格式，将数据封装成RTMP特有的块消息，可以封装多种格式的音视频数据，如AMF、FLV等。

3）、RTMP协议通过在一个固定的连接上建立多个传输通道传输流数据，其中传输流数据的大小是按照RTMP消息格式固定设置的。

RTMP协议是一种设计用来进行实时数据通信的网络协议，它的载体平台为Flash/AIR平台，还有一些支持RTMP协议的流媒体/交互服务器之间都可以进行音视频和数据的通信。支持此协议的软件包括Adobe Media Server/Ultrant Media Server/red5等等。

因特网作为一种新的媒体和信息传播方式，目前已逐步深入到全世界的每一个角落，同广播、报纸、杂志等传统媒体一样，Internet正逐步成为信息数据传播的重要通道之一，在TCP/IP网络横行的网络环境下，新型的视频标准格式日益出新，解压缩效率显著的提高，高压缩比能提供很好的网络传输效率，这也是解压缩技术推动流媒体技术发展的原因，流媒体技术就是在网络宽带技术和视频解压缩技术的相互协调下产生，发展，成熟的。

流媒体[1]的的概念是在Internet网络中，利用TCP/IP协议栈进行网络通信，利用多媒体技术进行数据收集的实时连续的流式传输技术。在流媒体技术中，对媒体文件并不是完整的送达目的端，而是以流的形式连续的传输，在流式媒体的数据流随时传送随时播放，只是在开始有一些延迟，流媒体实现的关键技术就是流式传输。

由于Linux系统[2]开源代码的发布，Linux自身具有的设备独立性，可靠的系统

安全，良好的可移植性，以及丰富的网络功能，这些特性使得Linux系统成为众多服务器的首选架设平台，

在2007年，Android系统的源代码公布与众，这样一来，以Android系统[3]为平台的各种产品有了飞速的发展，想智能手机，家电市场等等，这些领域很多产品都是以Android系统为平台设计实现。即使有些产品未使用Android系统，也会为其提供相应的接口，使之能与Android系统兼容，如HDMI接口就是用于兼容Android系统的。Android系统的飞速发展使得很多公司企业借助这一平台，不断的进行研发，不断的开发出便利智能的产品以满足用户的多层次需要。

随着 Android 系统在手机、平板这样的手持设备上得以应用后，人们便热衷于间很多生活需求移植到 Android 平台下，如本工程涉及到的视频监控需求。谈及到视频监控首先必须要采集视频原始数据并进行编解码，Android 系统中就包含了 OpenCore 编解码框架，遗憾的是，OpenCore 编解码框架智能解码 3GP 和 Mp4 格式，这一点对于本工程来说非常不利，本工程是欲将采集的数据进行网络传输，所以必须要有较高的压缩比，这样才能更好的利用网络带宽，故引进了 FFMPEG 开源库，FFMPEG 原本是在 Linux 下开发的音视频解码系统，由于其可移植性良好，在 Android 系统下同样能够使用，FFMPEG 的优点就是可以编解码多种音视频格式数据，本工程中利用到的 H.264 格式数据就是 FFMPEG 所支持的，其高压缩比也满足网络传输的项目要求，故本文的研究重点之一便是将 FFMPEG 在 Linux 和 Android 系统上得以实现。虽然 Android 系统自带的多媒体系统 OpenCore 可以实现一些简单视频格式的编解码，但是由于其所能解码的容器格式有限，这点削弱了用户的使用体验。

1.2 课题相关技术研究现状

现如今，人们可以通过视频来获取外界大量的信息，视频相比较于音频和文字内容，具备这更好的直观和可视效果，能更加清晰的描述传达事物信息，在和会和科技高速发展的今天，各个行业中都离不开视频监控，这使得视频监控技术越来越得到了人们的重视，视频监控技术的质量和实时性也得到了普遍的提高，其中对于监控系统而言，有线监控和无线监控成为了监控的两种手段，视屏监控技术已经从最早的传统视频监控发到到初具模型的有线网络数字监控再到现在较为流行的无线视频监控。针对视频监控技术，国内外都进行了激烈的讨论研究，视频监控技术也是计算机领域里的热门话题。

近年来，网络带宽的逐年提升，多媒体压缩编码技术也日趋成熟，种种因素使得流媒体这一概念摆到了用户需求上，全球的流媒体市场都在飞速的向前发展，

旧时以文本和图片形式的传播手段越来越无法满足用户想要快速高效的获取信息的渴望了，Cisco 机构就做过这样的统计，名为 VNI 统计，统计中提到，2005 年、2011 年、2015 年流媒体流量占全球互联网总流量的百分比，从仅有的 5%上升到 40%最后高达 62%，另外，流媒体不仅在电脑这个载体的需求量大，在平板电脑和智能手机等领域都有着很高的需求。

如今流媒体技术正处于高速发展的状态下，许多电视台都开展了流媒体业务，像 CNTV、中国体育电视台、江西电视台等多家地方电视台，以及优酷网，直播吧等流媒体服务商都进行了流媒体系统的相关业务。许多流媒体系统传输数据使用的 RTMP 协议因此得到了非常广泛的应用。本文将会对其特点进行详细的分析，并 Linux 系统和 Android 系统上搭建一个基于 RTMP 协议的流媒体系统。

另一方面，Android 系统[4]由 Google 公司在 2010 年发布，自发布开始该平台就以意想不到的发展速度迅速取得了世界范围内多于一半的智能手机操作系统的市场。自 2010 年 Google 发布了开源系统 Android 的源代码起，该平台就以意想不到的速度迅速取得了全球大半个手机操作系统的市场。2014 年 10 月的市场调查数据显示，Android 系统在全球的智能设备所使用的操作系统市场占有率已经高达 62%，中国一些发达的城市中，市场占有率为 78%，这些数据都说明了 Android 系统正越来越融入了人们的生活，自从 Android 系统开源发布以来，全国的各个电子产品生产厂家就花大精力开发生产智能产品。十分看好 Android 平台的发展前景。为 Android 系统的发展提供了广阔的空间。

1.3 课题研究的目的和意义

RTMP 是目前应用最广的实时消息传输协议之一，在不远的将来，大部分的音/视频数据都将通过网络传至远端，流媒体网络通信的发展前景非常诱人，但要将网络的大量的音/视频数据进行传输并不容易，RTMP 以及 FFMPEG 便致力于这一工作，在该工程中，我们利用 FFMPEG 的高压缩比的特性将数据进行压缩，便于网络传输，在利用 RTMP 通信协议构建的流媒体平台，将数据传播。

这里需要提到的是，本系统使用到了网络通信技术和多媒体技术，多媒体技术就是对视频音频数据的采集、编解码、播放，两种技术相辅相成，相互促进发展，流媒体技术的研究少不了对多媒体视频技术的掌握。本文设计实现了基于 FFmpeg 和 RTMP 协议的流媒体多平台视频采集传输系统[5]，采用模块化的思想，自 20 世纪末电视产生以来，越来越多的媒体介质出现在人们的日常生活中，交流方式由原来的人——人，转化成了人——网络——人，该工程主要将 RTMP 流媒体服务器应用于视频监控领域，使使用者在远端任意一个有网络的地方，打开一

个网页就能进行实时监控，同时能进行多点监控。例如是以下的场景：在家庭生活中，可以在家中搭建好流媒体服务器以及摄像头，搭建好后在办公室就能监控到家中的情况，可以看看家中独处的孩子是否安全，家中设备是否会出问题等等。

1.4 课题研究的主要内容与实现的主要功能

本工程的研究主要侧重两个方面，一是基于 RTMP 协议的流媒体服务器[6]的构建，并将 RTMP 流媒体服务器应用于视频监控[7]，二是将流媒体服务器移植到 Linux 系统及 Android 系统。对于第一个功能我们需要完成的主要工作有：研究 RTMP 协议[8]内容，认清 RTMP 通信细节和方式，熟悉协议的每个工作流程及每个工作流程涉及的具体协议，研究 FFMPEG 源码架构，利用 FFMPEG 对视频数据进行采集和压缩。对于第二个功能我们需要完成的主要工作有：研究 FFMPEG 的编译移植，JNI 的调用，以及 Android 服务的启动调用，编译 FFMPEG 动态库，实现其在 Android 端的移植。最后以网页内嵌播放器的形式作为客户端，对远端 RTMP 流媒体服务器进行实时监控。

1.5 论文整体结构

本文主要通过六个章节来对本系统进行描述：

第一章：绪论。对研究课题大观上的掌控，涉及到相关技术的发展情况以及发展背景。并且对设计本系统的实用性做出了论述。

第二章：对本系统研究所涉及到的基础技术和理论知识进行了描述，本文中涉及了 Linux 系统及网络栈架构、Android 系统简介、RTMP 实时消息传输协议、FFMpeg 源码[9]等多方面的知识。

第三章：本章是系统的关键设计部分，从系统功能着手，分析了本系统应提供的功能，并对系统进行了模块划分。

第四章：本章为全文的重点，在这一章中，详细的阐述了基于 RTMP 协议的流媒体服务器的实现，该实现主要分为两大模块，视频数据采集压缩模块的实现和 RTMP 协议通信模块的实现。在本章中对视频数据采集压缩模块涉及的 FFMpeg 库进行了详细讨论，并将它编译成动态库，使其在 Android 系统上顺畅运行，对系统另一模块 RTMP 的协议进行了详细的分析，并在其基础上对其进行了实现。

第五章：本章主要是对全文工作的一些总结，在已有的基础上查找自己的不足，对下一步的工作作出了展望。

1.6 本章小结

本章内容主要是说明了本文的选题依据，通过对相关文件的查阅了解描素了相关技术的国内外的发展研究现状，同时也介绍了本文的主要的研究课题、研究目的以及研究意义。在本章的 1.4 节里系统的概述了本文相关的研究方法，在本文的最后对全篇的结构进行了系统的概述，并阐述了每章所做的主要工作。

第二章 相关理论与技术基础

2.1 Linux 系统及网络栈架构简介

Linux 系统是由 UNIX 发展而来[10]，经过一步步的完善，使其现有的性能、可靠性、系统管理、安全性、功能都有了很大的保障，Linux 以其公开的开发过程，完善的操作系统功能，使其具有强大的生命力，独有的文件系统结构，即只有一个文件树，更是方便学习、开发、管理。RedHat、Slackware、Debian、Turbo 等 都是 Linux 的常见版本，它们分别由一些团体或者企业开发，主要进行集成工作，即将 Linux 内核[11]与一些支持 Linux 的工具和应用程序集成，从而产生不同的版本，在网络方面，Linux 操作系统最大特性之一就是它的网络栈，来源于 BSD 的网络栈，下表是 Linux 网络栈的架构表如表 2-1：

表 2-1 网络栈架构表

	Example
User/Application	Firefox browser
Application layer	HTTP
Transport layer	TCP
Network layer	IP
Link layer	Ethernet driver
Hardware layer	Ethernet

表中最上面是用户操作的空间层，即应用层，底部是物理层，中间是网络子系统，即内核空间，流经网络栈内部的是 socket 缓冲区（sk_buffs）,它负责在源和汇点之间传送报文数据。

当用户发起网络调用时，通过系统调用接口进入内核，最后调用./net/socket.c 中的 sys_socketcall 结束该过程，然后进一步调用分路发送到指定目标。系统调用接口还有另外一种描述，就是将普通文件操作当成网络 I/O。也就是说有些标准文件操作可以应用于网络对象，就像操作普通文件一样，

Socket 层是一个协议无关接口，该层有一组通用的函数来支持各种不同的协议，像 TCP / UDP 协议[12]自然不在话下，另外还可以支持 IP、裸以太网和其他传输协议，如 SCTP。通过网络栈进行的通信都要对 socket 进行操作，在 Linux 操作系统中，socket 的结构是 struct sock,这个结构是在 linux/include/net/sock.h 中定义的。

Socket 所需要的所有的状态信息都存储在这个结构体中，

像 TCP/UDP 这样的协议都是在 linux/net/ipv4/af_inet.c 文件中一个名为 inet_init 的函数中进行初始化的，它使用 proto_register 函数来注册每个内嵌协议。

Linux 支持 TCP/IP 网络，Linux 系统包含了各种服务器，其系统本身就能作为多种服务器进行使用，同时 Linux 系统支持多种网络协议，是现在服务器构建最广的系统平台之一。本工程把 Linux 构建成一个流媒体服务器，并移植到 Android 整台，实现视频的传输监控，更重要的是，Linux 代码的完全开放有助于它的网络安全，这个是网络的生命所在。

2.2 Android 系统简介

Google 公司在 2007 年 11 月发布了一个基于 Linux2.6 内核的手机操作系统，且以开源的形式面向所有人。Android 系统拥有 Linux 全部特征且系统小巧，故主要使用与手机、平板这些小巧的编写设备上。

Android 操作系统的底层是开放的 Linux 内核，因此涵盖了 Linux 系统的所有特性，底层是由 Linux 内核构成，上层的应用程序的开发是通过 java 语言来完成调用，通过上层的 java 编程语言来调用底层平台的所有核心功能。Android 系统的特新有很多，诸如多媒体支持、数据库、以及 Google 特有的综合网络浏览器等等。相比较其他系统的 SDK 和 Android 系统的 SDK，Android 不仅具有开放性，同时系统还提供了独有的新型的系统 API，这样做的好处是极大程度的方便了开发者调用系统功能，上层应用程序能更搞笑的进行系统调用。

将 Android 系统比较与其他当下流行的操作系统，像 iPhone、Windows Phone 等，Android 系统有着开放性、可移植性、应用程序间平等且无界限以及无缝结合的 Google 应用等优势。

Android 系统采用的结构是分层的结构[13]，从底层至上层一个有四层，分别是应用程序层、应用程序框架层、系统运行库层和 Linux 内核层，其系统结构图如图 2-1 所示。

<table>
<tr><td colspan="2">APPLICATION
Home Contacts Phone Browser ...</td></tr>
<tr><td colspan="2">APPLICATION FRAMEWKORK
Activity Manager Window Manager Content Providers
View System ...</td></tr>
<tr><td>LIBR ARIES
Surface Manager Media Framework
SQLite SGL</td><td>RUNTIME
Core Libraries
Dalivik Virtual Machine</td></tr>
<tr><td colspan="2">LINUX KERNEL
Display Driver Camera Driver Binder Driver USB Driver ...</td></tr>
</table>

图 2-1 Android 系统结构

1）、Android 应用程序层：Android 将一系列核心应用程序包（如图 2-2 所示）一起发布，JAVA 编程语言用来编写 Android 上层的所有应用程序。该层是和用户界面设计打交道的主要层。这是 Android 系统的最上层，负责和用户打交道，是和用户交互的一层，Android 平台不仅仅是操作系统，其中也包含了许多应用程序，像拨号程序、e-mail 程序、自带的相机功能、日历、地图、联系人、SMS 短信服务程序等应用程序，用户通过这些应用程序和 Android 系统进行交互应用操作。Android 系统存在一个其他手机操作系统所不具有的特点，即应用程序都可以通过对 Android 源码的重新编译或程序人员的重新进行替换的，这些应用程序并不是固化在 Android 系统内部，这样使得 Andoird 应用程序的可开发性强，即 Android 系统比其他的系统更加人性化和便于二次开发，加速了 Android 的发展。

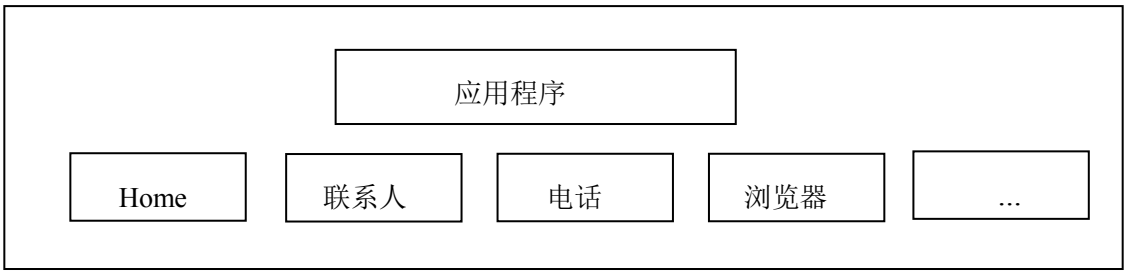

图 2-2 应用程序层结构

2）、应用程序框架层：Android 应用程序框架（如图 2-3 所示）保证了应用程序间平等和无界限开发的特性，程序人员通过对应用程序框架层的使用来进行上层应用层的程序开发，通过调用应用程序框架层中的系统 API，进行应用程序及服

务的复用。Android 系统开发过程中，常用的四大组件分别有：Activities、Intents、Services 以及 Content Providers。

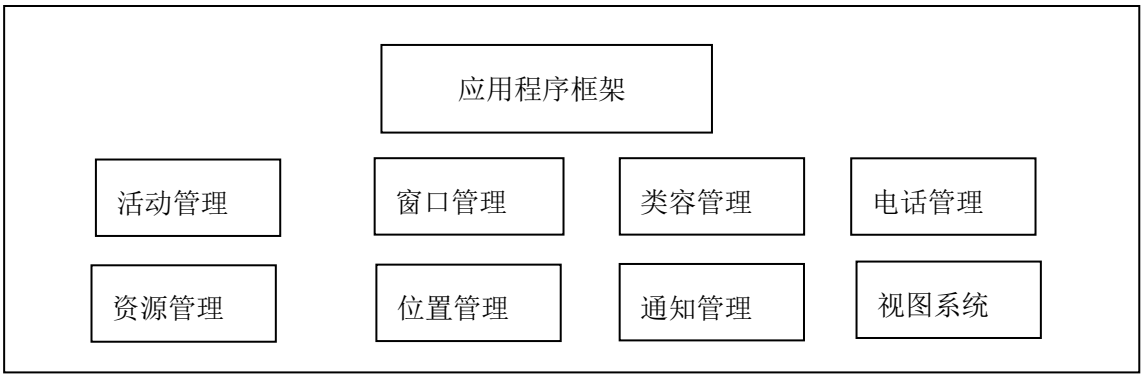

图 2-3 应用程序框架结构

核心运行库层：核心运行成可以分为两块内容，系统库和运行环境，系统库包含系统各个组件所使用的 C/C++集合的库（如图 2-4 所示），同过 Android 应用程序框架调用系统库来为开发者提供系统服务，另一块内容为运行环境，Android 的运行环境是 Dalvik Java 虚拟机和基础的 Java 类库组成的，Android 开发的应用程序都是运行在这个环境之中，每个程序在自己所有的进程里，在 Java 编译器将程序里所有的类进行编译后，再通过 SDK 中的“该层为 Android 的中间件层，也可以理解为 Android 系统与 Linux 内核的桥梁层，因为 Android 系统是基于 Linux 内核的，Android 系统就是基于此层来对 Linux 内核进行升华复用，此层均为 C 或 C++语言编写。Android 运行时库有两部分：一个为核心库另一个为 Dalvik 虚拟机。核心库部分实现了 Java 语言核心库大部分的功能，该部分是由应用程序框架层通过 JNI 调用的方式从而调用底层程序库的接口。其中，中间件又有两个部分，分别是：核心库和运行时(libraries & Android runtime)，核心运行库的另一部分核心库则包括 SurfaceManager、MediaFramework 媒体库、SQlite 数据库等。其中 SurfaceManager 负责在屏幕端 2D 或者 3D 内容的显示，MediaFrameWork 媒体库主要负责图像及音视频的输出，SQLite 数据库则是 Android 系统自带提供给用户使用的基于嵌入式的轻量级的关系数据库。

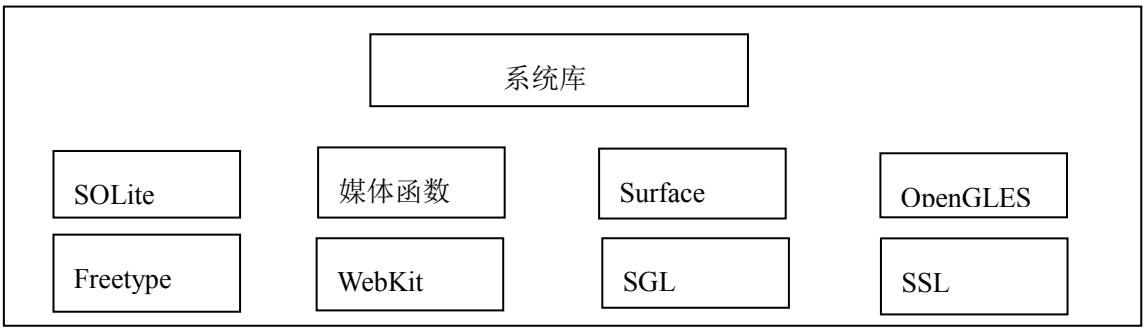

图 2-4 系统库组成

图 2-5 运行环境结构

4）、Linux 内核层：Android 系统内核（如图 2-6 所示）采用的是 Linux2.6 以上的版本，抽象于硬件和软件之间，驱动模型，内存管理等功能都需要 LInux 内核层的相关控制，其主要功能有进程管理、内存管理、硬件驱动、网络协议栈、安全等，同时，它是软硬件之间的抽象层，即为软件和硬件之间的桥梁，透明了内核的底层实现，方便了程序开发人员的开发。

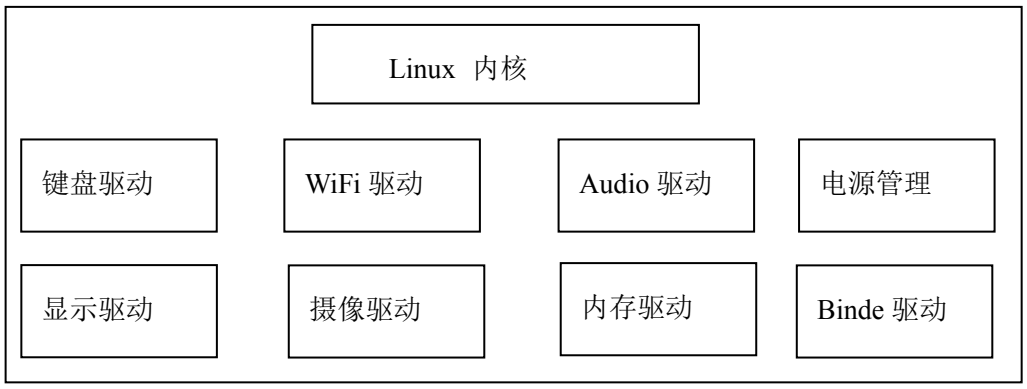

图 2-6 系统内核结构

2.3 流媒体技术分析

2.3.1 流媒体技术原理

所谓的流媒体是指采用流式传输的方式在 Internet 播放的媒体格式，流媒体又叫流式媒体，一个视频采集端服务器将采集的视频分组成网络包，传送到网络上，另一端即客户端通过解压设备对这些数据进行解压组装后，将数据原样显示出来。网络音视频技术和移动通信技术的发展进步推动了流媒体技术的提升，给流媒体技术提供了技术保障，其中就涉及到流媒体技术从数据的采集到播放整个过程。

流媒体传输技术是基于 FTP/TCP 协议，在此基础上发展而来的，故器数据具有实时性，时序性，连续性的特点，使得流媒体数据要有严格的时序关系，流媒体数据将由服务器端分组成若干个待发的数据分组，然后按照特定的实时协议将分组一次在网络上进行传输，另一端的客户将接收到的分组按照实时协议的规格和视频格式重新组装起来，从而形成一个最后的数据文件。

流媒体技术能够实时传送并即刻播放的特性，即使在开始会有些许等待延迟，但仍然在实时性要求高的领域得到了广泛的应用。然而若在网络速率不稳定，网络环境复杂的情况下进行流媒体传输，视频将会出现停滞，断续的现象。

编码器，流媒体服务器，以及客户端播放器这三个部分就能组成一个完整的流媒体控制系统，如图 2-7，各个模块之间通过 RTP/RTSP/TCP/http/rtmp 等协议进行通信数据交换，编码器的作用是将采集的数据通过视频编码技术，编码成网络能够识别的流媒体格式文件，便于在网络上传输，服务器则是负责接收和转发网络上传输的流媒体数据，客户端则是将编码的数据解码以便在客户端播放流媒体视频数据。

流式传输技术需要相关的协议来支持，来协调双方有规则的通信，在实现方案中，传输控制信息一般用 HTTP/TCP 协议，而传输实时数据则是根据 RTMP/RTP/UDP 等协议。

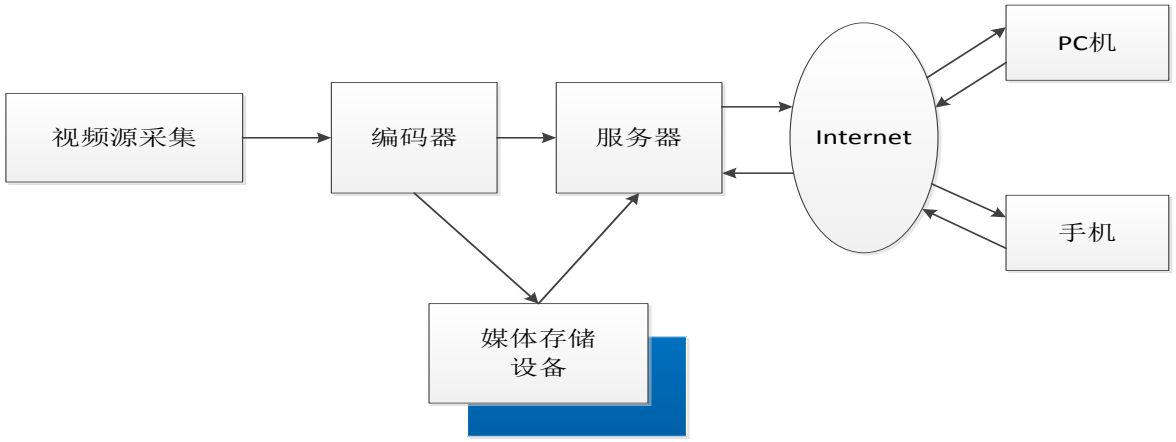

图 2-7 流媒体系统组成

由于服务器收集的实时音视频数据在网络传输中要被分解为多个小的分组，网络传输过程中选路协议和路由的动态变化等等因素，都会导致被拆分的分组到达目的端的时延不同即顺序不同，先发的数据包可能会在网络上延迟更久到达目的，所以流媒体传输需要缓存，需要用缓存系统来控制网络变化和选路协议带来的传输延迟，抖动等问题，从而保证数据分组包能顺序真确到达，通常高速缓存所需的容量并不大，因为高速缓存的内容可以不断的刷新空间可以重复利用。

顺序流媒体传输（Progerssive Streaming)和实时流媒体传输（Real_Time Streaming)是流媒体传输的两种方式。实时传输过程如图 2-8 所示。

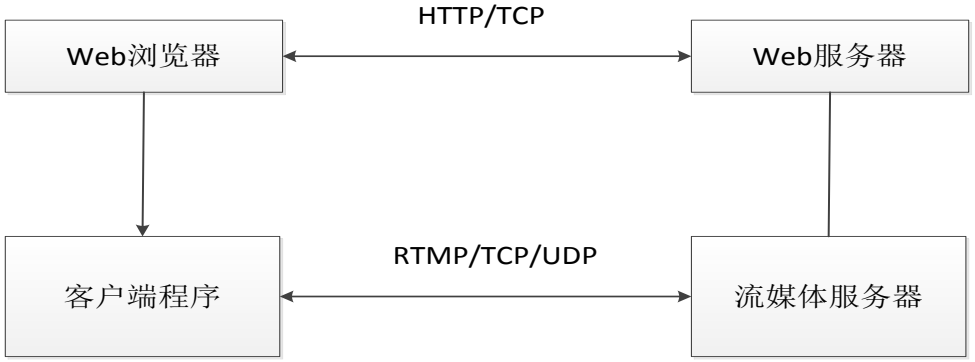

图 2-8 实时传输过程

（1）、如图中描述的那样，Web 浏览器和 Web 服务器之间利用 HTTP/TCP 协议进行的通信，交换两端的控制信息，这里的通信过程是要在用户根据需要选择了流媒体服务之后进行的。

（2）、传输实时数据的客户端和服务起基于特定的实时流协议进行通信，按

照协议的数据格式和传输规则，传输实时流协议所需要的控制报文和实时流数据。

在流媒体传输中，使用 RTMP/TCP/UDP 协议来和音视频服务器建立通信连接，同过协议双方约定通信步骤和传送内容。完成流媒体传输过程。

2.4 Java 本地调用介绍

2.4.1 Java 上层调用底层 C 代码

在 Android 系统中，Java 编程语言使用 native 表示本地方法[14]，比如说 public native void hello();这就是个简单的共有本地函数，之后我们便可以通过 jdk 提供的命令 javah 来生成一个 JNI 调用的函数名，通过执行 javah 命令的执行，便会生成一个包含我们需要的 JNI 调用的头文件，在这个例子中，生成的函数为 void Java_Class Name_hello(JNIEnv env,jclass cl)。在本工程中，将流媒体服务器移植到 Android 上时，使用的就是这种方法，将 RTMP 网络服务器生成 JNI 调用函数，让 Android 上层可以调用。

2.4.2 Java 本地调用的参数和返回值

Java 本地调用所必须面对的问题就是不同语言之间，他们类型表示的不同，即语言之间数字传递的过程中，他们参数的类型所占的大小会出现不同。比如说 int 类型数据，在一些平台为 16b 大小，而在另一些平台上为 32b 大小。因此，必

须对 Java 语言和 C 语言的数据类型的对应关系有个清楚的认识，如表 2-2 所示：

表 2-2 数据类型大小表

C 语言	java 语言	字节
boolean	jboolean	1
byte	jbyte	1
char	jchar	2
short	jshort	2
int	jint	4
long	jlong	8
float	jfloat	4
double	jdouble	8

所有对 JNI 函数中有一个 Env 指针，这个指针是函数表指针的指针，作为 JNI 调用函数的第一个参数，为了堆函数进行解应用，我们必须要在 JNI 调用前加上（*env)->。

2.4.3 调用 Java 代码

上两个章节阐述了 java 上层调用底层 C 代码，那么反之能否让 C 代码去调用上层的 java 代码[15]，答案是肯定的，这一小节主要介绍如何使用 C 代码去调用 java 代码，其中方法就设计到静态调用方法和非静态调用方法，首先介绍非静态调用方法，比如一下函数 Public native static void fprint(PrintReader out,String s,double x),C 函数调用 Java 方法，就可以通过这种形式进行调用，此函数就能让 C 函数调用 Java 中类 PrintReader 的接口 print 方法，C 调用 java 方法的通用格式如下：

（*env)->callXxxMethod(env,implicit parameter,methodID,explicit parameters)；

返回值类型不同替换相应的 Xxx 如 void 类型则为 callvoidMethod 等等。对于静态方法的调用，几本上和非静态方法调用非常相似，主要差别在于静态方法调用要用 GetStaticMethodID 和 CallStaticXxxMethod 函数，另外调用的时候必须调用类的对象。

2.5 FFmpeg 源码理论

2.5.1 H.264 视频压缩编码技术分析

地方地 ITU-TVCEG 与 ISO/IEC MPEG 标准化委员会联合研究出了 H.264 视频数据的相关格式要求，称为 H.264 标准，H.264 标准是对 MPEG-4 和 H.263 标准的提升和改善，不仅能保持与他们能获得相同的视频质量，同时，该标准对编码效率有了大幅度的提升。另外，H.264 标准的音视频数据能够搞好的适应当下网络传播环境，网络适应（Network-Friendly)性更优于其他标准，由于其高压缩比和网络传输效率高等优点，H.264 标准在实时会话和实时非会话中都有着广泛的运用。

如图 2-10 所示，整个 H.264 标准可大致分为两个模块，较底层的视频编码层和网络适应层，网络适应层的功能是按照标准所规定的数据格式，将音视频数据封装成标准的 H.264 数据格式，提高数据在网络上的传输效率，视频编码层则用于显示视频内容。在网络适应层中，还有相应的信令，这些信令操控与一个基于分组方式的接口，这个接口连接了视频编码层和网络适应层，使 H.264 标准中的各个模块能协同工作，视频编码层使能取得高效编码效率而网络适应层能让音视频数据具有更好的网络适应性。网络适应性也就意味着数据有着良好的网络亲和性与容错性。在各式各样的复杂网络环境中都能很好的适应，高效的传输。

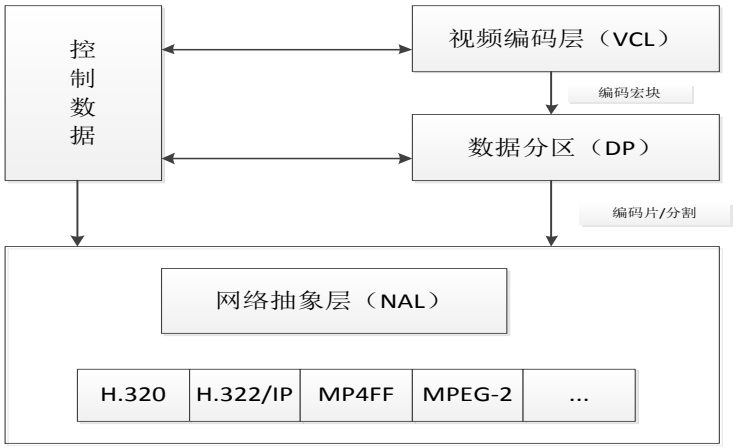

图 2-10 H.264 分层设计

H.264 编解码器是在 MPEG、H.261、H.263 等编解码器进行的改良的一个协议标准，因此他们都有的相近的编码模块和相同的编码器功能单元，像预测、变换、量化和熵编码等，然而 H.264 为了达到更好的编码压缩比和网络适应性，在每一个功能单元上有都进行了改良变化。

如图 2-11 所示，该图描述的是 H.264 标准的编码器结构流程图。流程图中显示了两条数据压缩与解压通路，从左向右的为从一帧数据经过量化和熵编码等步骤转化为 H.264 格式数据的编码过程，从右向左的则为有 H.264 标准格式数据反向逆推的解码过程。

其中，H.264 标准是采用帧内或帧间模式进行解码的，解码的对象是以 16*16 像素大小的宏块为单位的 X_n。下面分别解释帧间编码部分和帧内编码部分。

帧间编码部分：X_n 与 X_{n-1} 经 ME（运动估计）产生运动矢量，该运动矢量和 X_{n-1} 经 MC（运动补偿）生成预测值 P，X_n 和 P 做差得到帧间编码残差 D_n，然后经过 T（变换）、Q（量化）X、重排序、熵编码生成 NAL 码率。

帧内编码部分：X 经 Q^{-1}（反量化）、T^{-1}（反变化）获得 D'_n,D'_n 和 P（预测值）做和生成为滤波的 uX'_n。uX'_n 一方面经过滤波生成重建帧 X'_n，另一方面和 X_n 一起经过帧内预测选择、帧内预测产生预测值 P。

无论是采用哪种方式，都会由重建帧形成一个预测宏块 P，并用于之后的算法。

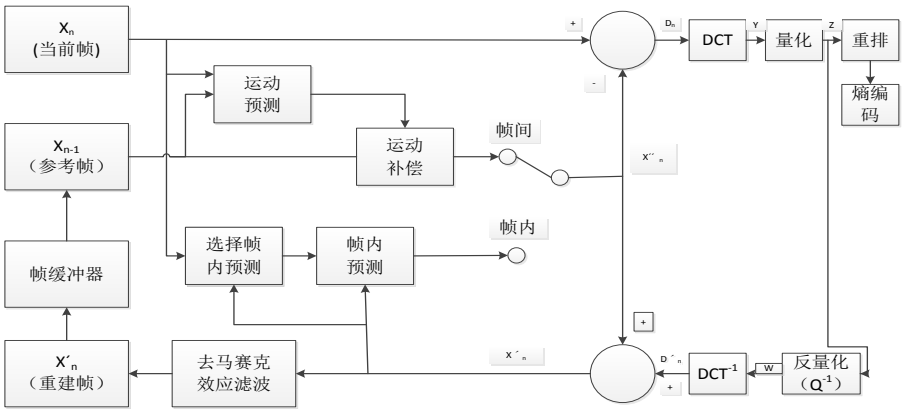

图 2-11 H.264 编码器示意图

2.5.2 FFmpeg 源码框架

开源的 FFMPEG 是由 C 语言写的，在 Linux 系统下使用可把它编译成动态链接库，在程序中进行调用即可，在 Android 系统中，若想移植 FFMPEG 则必须通过 JNI，通过编译 FFMPEG 成动态链接库 libFFMpeg.so，这样可以使其可以在 Android 系统中实现 C/C++的混合调用。FFMpeg 是当前最为流行的音视频编码问题的开源代码之一，本课题之所以在两个平台下都使用了 FFMpeg 库，就是因为 FFMpeg 的跨平台的优势[16]，并且在 FFMPeg 源码中包含了格式众多的编解码器，可以涵盖众多的音视频格式，如 FLV、MPEG2和 H263等格式。FFMpeg 其内部对各种编解码器的支持设计了统一的接口。各种编解码器需要用到的资源、变量和特性是通过一些公用的结构体来描述的，调用方便清晰。FFMpeg 源码文件中包含有如下 Library：

Libavcodec：这就是 FFMpeg 通过自己定义不同的格式来编解码的库。

Libavformat：用来处理解析视频文件并将包含在其中的流分离出来，例如 mux 和 demux。

Libavutil：此函数库可以实现许多数学运算功能，如最小公倍数，反码求和计算 CRC 校验码。同时也能通过此函数库实现数据本机格式与网络格式的双向转换。

Libavcore：这个类库是最新版本的 FFmpeg 才加进去的，是一个编译核心库。

Libavdevice：这是 FFmpeg 中的一个和多媒体设备交互的类库，使用这个库可以读取多媒体设备的数据，或者输出数据到指定的多媒体设备上。

Libavswscale：实现了各种图像像素格式的转换，例如 YUV 与 RGB 之间的转换；以及图像大小缩放（例如 640*360 拉伸为 1280*720）功能，同时还做了相应指令集的优化。

下面介绍 FFmpeg 中常用的重要的数据结构：

AVCodecContext：这是 FFmpeg 中最重要最复杂的结构体之一，其描述了编解码器的上下文。包含了所有编解码器索要用到的参数，主要用在编解码的过程中。通过调用者或者系统函数进行初始化，初始化函数分别有 av_open_input_file 和 av_find_stream_info，当调用系统函数是，系统函数是通过判断文件头信息和媒体流内的头部信息来对结构体进行的初始化。

AVStream：是存储每一个视频/音频流信息的结构体。其中包含流 ID、私有数据、时间戳等等信息。

AVFormatContext：该结构体包含了较多的码流参数，描述了一个媒体流的基本信息。

AVFrame：该结构体是保存了解码后和原始的音视频信息，通过函数 av_frame_alloc 初始化。

通过以上描述的 FFmpeg 中的结构体，可以实现解码过程，核心代码如下：

```
Struct AVCodec *codec=NULL;
Struct AVCodecContext *pCodecCtx=NULL;
Struct AVFrame *pFrame=NULL;
.....
Avcodec_init();
Avcodec_register_all();
Codec=avcodec_find_decoder(CODEC_ID_H264);
pCodecCtx=avcodec_alloc_context();
Avcodec_open(pCodecCtx,codec);
Picture=avcodec_alloc_frame();
Avcodec_decode_video(pCodecCtx,pFrame,&frameFinished,rawData,bytesRemaining);
```

总体流程就是定义一些结构体，并将其初始化，注册解码器，并且找到 H.264 解码器（CODEC_ID_H264），分配解码器内存，打开解码器，打开帧缓冲，解码。

2.5.3 FFmpeg 的优势

本工程主要利用了 FFMPeg 源码库中对 H.264 视频格式进行编解码的模块，FFMpeg 中最常用的两种编解码技术就是 H.264 和 MPEG4，比较 H.264 和 MPEG4，H.264 压缩比要高于 MPEG4 ，高压缩比使 H.264 能在网络率更高，更适合视频监控，故 H.264 为现有的视频监控中最常用的视频格式。同时，H.264 具有着低码流、高质量图像、容错能力强及网络适应性强等特点[17]，就相同的视频文件面言，采用 MPEG2 技术压缩的视频文件的大小是采用 H.264 编解码技术的视频文件的 8

倍，其效率可想而知，就算是 MPEG4 的数据大小也为 H.264 大小的 3 倍[18]。H.264 的这一特性大大的降低了视频在网络传输过程中所产生的流量，节省了用户的延迟时间，提高了监控效率，这些特点都提高了用户体验，受到了开发人员的青睐。基于低失真的编解码算法而形成是 H.264 编解码技术的主要特点，故监控画面的清晰度也得到了保障，并且流畅的高质量图像也是 H.264 压缩技术所能提供的。更重要的是，H.264 还提供了一系列的处理编解码及网络传输出错问题的处理机制，如此一来，网络传输容错能力的提高，使得监控系统更加稳定可靠，使其在网络不稳定时也能够保证正确的将数据包传输到用户的客户端，并且保证了在网络传输中的低丢包率。更加可靠的网络传输是视频监控面临的重要问题，同时，H.264 提供的视频编码和网络提取模块都能更好的保证视频格式数据在网络上能正常高效的传输。

2.6 RTMP 协议

2.6.1 RTMP 协议简述

RTMP 协议即 Real Time Messaging Protocol（实时消息传输协议）[18]，它是基于 TCP 协议的，是一个协议簇，其中就包括了 RTMP 基本协议以及 RTMPT/RTMPS/RTMPE/等多种变种[19]，RTMP 及其相关协议在 OSI 模型的位置如图 2-12 所示。本工程只利用了 RTMP 基本协议，故对于其他变种协议不做探讨，RTMP 协议是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的。支持该协议的软件包括 Adobe Media Server/Ultrant Media Server/red5 等，

RTMP 协议是基于 RTMP 消息块流进行传输通信的，它可以处理任何传送消息流的协议，每一个消息都包含了时间戳和负载类型标志，RTMP 消息块流和 RTMP 一起适用于多样性音视频应用程序，像点到点和点到多的实时直播，还有 vod 服务，以及交互式视频会议等 。

<table>
<tr><td rowspan="2">应用层</td><td rowspan="2">HTTP</td><td>RTMP</td><td>RTMPT</td></tr>
<tr><td>RTMPS</td><td>RTMPE</td></tr>
<tr><td>传输层</td><td>TCP</td><td colspan="2">UDP</td></tr>
<tr><td>网络层</td><td colspan="3">IP</td></tr>
<tr><td>数据链路层</td><td colspan="3">网络相关协议</td></tr>
<tr><td>物理层</td><td colspan="3">无线传输介质</td></tr>
</table>

图 2-12 相关协议在 OSI 模型的位置

当配合像 TCP 这样的可靠传输协议使用时，RTMP 消息块流提供可靠地规则时间戳的端到端全信息的传送，穿过多层流，但是 RTMp 消息块流不提供任何控制的优先级别和相似形式，但开发人员可以用于高层协议提供这样的优先级，例如，视频直播服务可以基于每个消息的发送时间和答复时间选择丢弃视频消息，使慢的客户端能及时接受到音频消息。

RTMP 块流也包含着自己的带内协议控制消息，并且提供了让更高层的协议嵌入用户控制消息的机制。

2.6.2 RTMP 协议通信机制

（1）、握手，一个 RTMP 通信握手开始，RTMP 协议握手区分与 TCP/IP 协议握手，这里的握手是通过 RTMP 协议自定义的 6 个消息块来完成，分别是 C0,C1,C2,S0,S1,S2，前三个消息快是由客户端发送，后三个消息快是由服务器端发送。发送的消息块必须有序的等待发送，发送流程图如图 2-16。

当发起 RTMP 连接，客户端便发送 C0 和 C1 消息块，意味着握手的开始。服务器接收到这连个消息块后便发送 S0 和 S1 消息块作为应答。之后由通信双方按序发送 C2 和 S2 消息块，握手过程结束。客户端开始发送请求命令字段，服务器则响应相应的命令字段或者数据消息快。

图 2-13 是 C 0 和 S 0 的消息格式图。

0	1	2	3	4	5	6	7

图 2-13 C0S0 消息格式

C 0 和 S 0 是单独的一个字节，在 C 0 中这个字段表示客户端要求的 RTMP 版本，而在 S 0 中则表示服务器对客户端的要求版本的确认，如若服务端不支持版本则以 3 返回，客户端或许会选择 3 以下的版本，或者放弃握手。

图 2-14 是 C 1 和 S 1 的消息格式图。

0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
time (4 bytes)
zero (4 bytes)
random bytes
random bytes cont ……

图 2-14 C1S1 消息格式

C 1 和 S 1 的数据包有 1 5 3 6 个字节长，由以下几个字段组成。

时间字段： 4 字节，本字段包含的是时间戳，这个时间戳是发送这个数据块的端点的后续块的时间起始点，是个基准相对值，故可以是 0 ，也可以是其他的任何值，之后的通信时间要这这个基准值为标准，为了同步多个流，端点可能发送的是其块流的当前值。

零字段： 4 字节，该字段必须全部为零。

随即数据字段： 1 5 2 8 字节，这个字段可以包含任何任意的值，因为每个终端必须区分自己初始化的握手的数据和对方初始化的握手的返回数据，这个数据应该发送一些随机数，但是并不需要加密安全的随机值，或者动态值。

图 2-15 是 C 2 和 S 2 消息格式图。

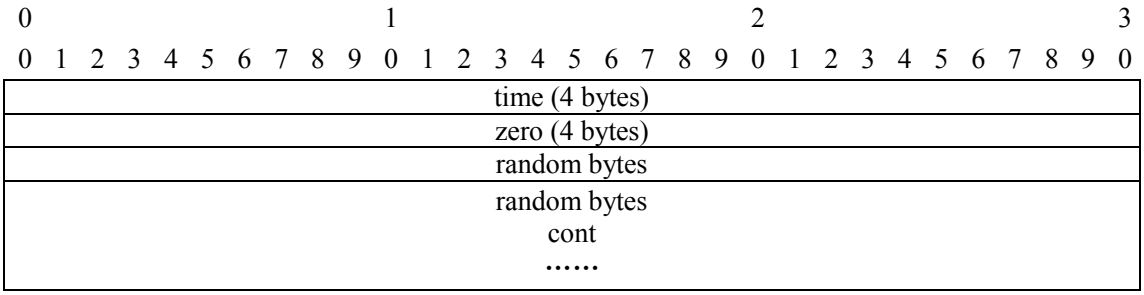

图 2-15 C2S2 消息格式

C 2 和 S 2 消息有 1 5 3 6 个字节长，只是对 S 1 和 C 1 的回复确认，由以下字段组成。

时间： 4 字节，本字段必须包含对等端发送的时间，即对 C 2 来说是 S 1 ，对 S 2 来说是 C 1 。

时间 2 ： 4 字节，本子段是时间戳字段，必须在被通信对端读取之后发送。

随机回复字段： 1 5 2 8 字节，本子段是对 RTMP 通信对端发送的随机数据的确认，将接收到的对端的随机字段原样返回以确认通信双方。即对 C 2 来说是 S 1 中的随机数据，对 S 2 来说是 C 1 中的随机数据。

握手过程图如图 2-16。

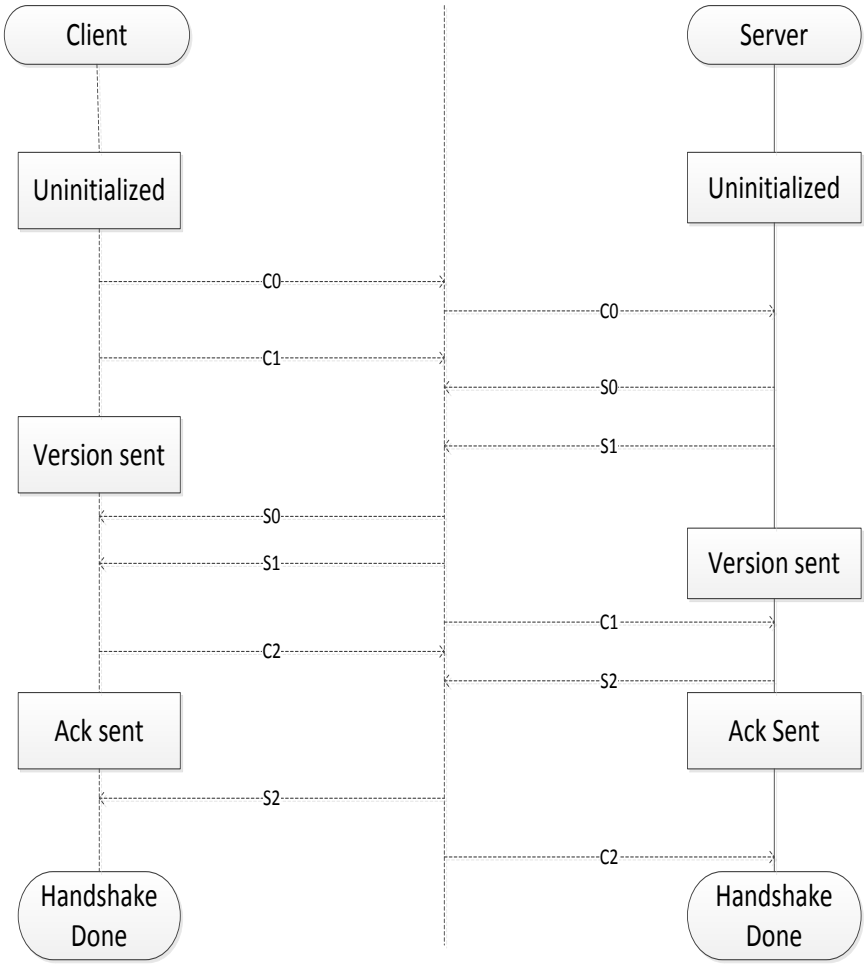

图 2-16 协议握手过程

表 2-3 的表格描述了在握手阶段的各个状态

（2）、 消息分块，在握手之后，连接就开始复用一个或者多个的消息块，每个消息块流负责承载来自一个消息流的一类消息。每一个消息块的创造都关联到一个唯一的块流 ID，以标识它，消息块流在网络上传输。在传输过程中，消息块之间必须依次发送，即每一个消息块在下一个消息块之前必须完全被发送，在接收端，第个块都根据块 ID 被组装成消息，消息分块使得高层协议的大消息侵害成小的消息，保证大的低优先级消息不阻塞小的高优先级消息。分块把原本应该在消息中包含的信息压缩在块头中从而减少了小块消息发送的开销。另外，块大小是可以配置的，最大块是 6 5 5 3 5 字节，最小块是 1 2 8 字节，块越大 CPU 的使用效率越低，这样大的写入，在低带宽的情况下容易产生其他内容的延迟，块的大小对每个方向都是保持独立的。图 2-17 是消息块格式图。

表 2-3 握手状态表

阶段	描述
未初始化	期间协议的版本将被发送，客户端利用 C0 包发送协议版本，如果服务器支持客户端所要求的版本。就回应客户端，并且发送 S0 和 S1 字段，如果不能，能连接结束
版本发送	在上一个阶段之后，客户端便等待服务器 S1 字段的到来，服务器端则需等待客户端的 C1 字段包，收到 S1 字段包后，客户端发送 C2 字段，而服务器收到 C1 后，便向客户端发送 S2,便进入下一个阶段。
ACK 发送	客户端等待 S2 字段包服务器端等待 C2 字段包
握手完成	客户端和服务器开始交换信息

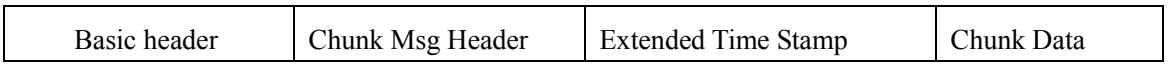

图 2-17 消息块格式

每一个消息块有包头和数据组成，包头又由三部分组成，分别是块基本头，消息块消息头和扩展时间戳。

块基本头：1 到 3 个字节，本字段包含块流 ID 和消息块的类型，消息块类型决定了消息包头的编码格式，长度取决于块流 ID，块流 ID 是可变长字段。

RTMP 协议支持 6 5 5 9 7 种流，ID 从 3-65599。ID 0、1、2 作为保留。0，表示 ID 的范围是 64-319（第二个字节+64）；1，表示 ID 范围是 64-65599（第三个字节*256+第二个字节+64）；2 表示低层协议消息。没有其他的字节来表示流 ID。3-63 表示完整的流 ID。3-63 之间的值表示完整的流 ID。没有其他的字节表示流 ID。

块流 ID 2 － 6 3 可以用 1 字节表示，如图 2-18。

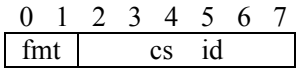

图 2-18 单字节流 ID 结构

块流 ID 6 4 － 3 1 9 可以用 2 字节表示，ID 计算是（第二个字节＋6 4 ）如图 2-19。

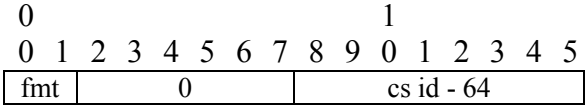

图 2-19 双字节流 ID 结构

块流 ID64-65599 可以表示为 3 个字节，ID 计算为第三个字节*255+第二个字节+64，如图 2-20 所示。

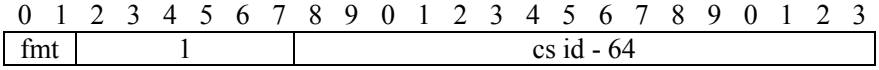

图 2-20 3 字节结构图

Fmt:2 位，对应消息块头中的 4 种不同类型的消息块流。

消息块消息头： 0 ， 3 ， 7 ，或者 1 1 个字节，这个字段编码要发送的消息的信息，长度可以利用在消息块头中详细的消息块类型来决定[20]，即 Fmt 字段值来决定。

类型 0：对应的块长度为 11 字节，在一个块流开始和时间戳返回的时候必须要发送这种块。格式图如图 2-21。

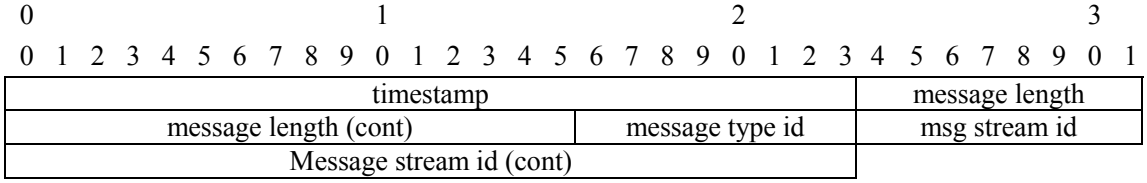

图 2-21 类型 0 消息格式

类型 1：对应的块长度占 7 个字节，消息流 ID 不包含在本块中，块的消息流 ID 与先前的块相同。具有可变大小的消息的流，在第一个消息之后的每个消息的第一个块应该使用这个格式。格式图如图 2-22。

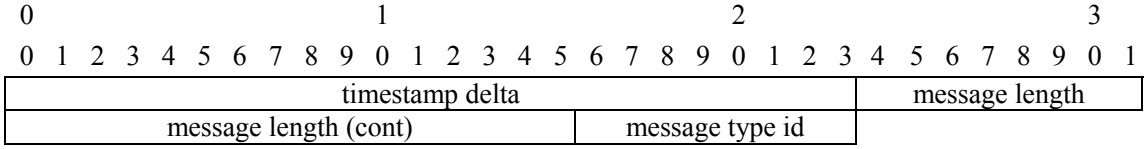

图 2-22 类型 1 消息格式

类型 2 ：对应的块长度占有 3 个字节，既不包含流 ID，也不包含消息长度，本块使用的流 ID 和消息长度与先前的块相同，具有固定大小的消息的流，在类型 0 消息块后应该利用这种格式作为每一个消息的第一个消息块。格式图如图 2-23。

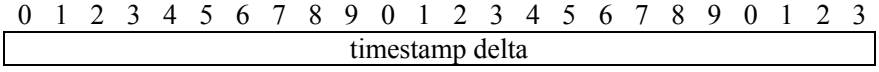

图 2-23 类型 2 消息格式

类型 3 ：此类消息块没有头，流 ID，消息长度时间戳，这个类型的消息块在之前的消息块中取值，当单一的消息被分裂成消息块，所有的消息块除了第一个，

应该使用这种类型，流由同样大小的消息组成。

扩展时间戳： 0 个或 4 字节，本字段必须在发送普通时间戳（普通时间戳是指块消息头中的时间戳）设置为 0 x f f f f f f 时发送，正常时间戳为其他值时都不应发送本值，即扩展时间戳是在普通时间戳的表达范围之外使用的，普通时间戳无法表示的较大值时用扩展时间戳来表示，若普通时间戳可以表示，刚不应发送本值。

2.7 本章小结

本章主要是详细的讲解了本文中所需要的各种技术，包括Linux系统以及网络栈、Android系统架构、FFMpeg源码、RTMP协议的相关内容。第一节中介绍了Linux系统以及Linux系统以及Linux系统网络栈架构，第二节中详细的介绍了Android系统的四层结构，第三节主要是对FFMpeg源码进行了系统概括，详细的介绍了FFMpeg所包含的三个库，并对库中的主要文件分别进行了系统的描述。文中的最后一节则对RTMP协议进行了详尽的解说，主要描述了RTMP协议概念，消息块格式，以及RTMP的通信机制。

第三章 基于 RTMP 协议的多平台多媒体流服务器总体设计

今年来，在物业和家居管理的领域里，对视频监控的需求越来越多，对监控的实时性和质量也越来越高，本系统就是根据这一现象提出的设计思想，对人们感兴趣的安全需求高的场所进行实时监控的 PC/手机版。通过本系统，可以对监控目标进行移动多用户监控，实时监控，一定成都上可以实现家居智能化，设计思想是摄像头前端采集视频信息，网络服务起对采集视频进行压缩，分组，封装等操作，将用于网络传输的实时数据利用 Socket 通信的方式由无线网络发出，Web 网页上的视频监控终端以及手机上的视频监控终端，接受视频数据包，并经过组包，解压缩等处理，将服务器发送过来的实时监控数据复原并播放，实现监控效果，下面分别介绍各个功能模块的需求和实现思想：

视频采集模块：用户将摄像头固定在感兴趣的监控目标区域，并采集监控区域的实时视频图像。

视频图像压缩：在服务器端对采集的视频进行压缩，分组，封装，并与网络服务器通信，将监控数据传给网络服务器。

视频发送：在 RTMP 网络服务器端对接受的数据进行流量控制，使用 Socket 通信技术与远端监控终端取得联系，并把数据由网络发送。

视频接受：在客户端即远程监控终端上，用户可以首先发起链接请求，告知服务器监控需求，并开始接受实时视频监控数据，查看远端监控状况。

视频存储：对于监控终端是 Web 网页内嵌播放器而言，将视频存储与远端 RTMP 服务器上，并支持点播，而对于 Android 平台的手机用户来说，可将视频存与手机中，以便日后观看。

基于 RTMP 协议的多平台多媒体流服务器主要包含两个大的模块，一个是利用 FFMpeg 开放库进行视频的采集压缩存储，另一个是利用 RTMP、TCP/IP 等网络协议栈对压缩视频数据进行封装发送以及客户端与服务器端的通信。以 H.264 格式对视频进行采集，并打包成 FLV 格式的视频数据，这里采用 FLV 视频数据格式是因为 FLV 的帧数据格式与 RTMP 协议通信数据块流格式相似，便于 RTMP 协议将视频数据打包封装 RTMP 数据包，并进行发送。通过 RTMP、TCP/IP 协议将视频数据进行一对一和一对多的传输，实现多客户端的视频监控。并将 RTMP 服务器在 Android 系统上移植实现，在 Android 上层启动一个服务，通过 JNI 调用底层的 FFMpeg 库代码进行视频数据的采集，同时也调用 RTMP 协议相关实现代码对相应端口进行监听，以实现 RTMP 协议包的传输，实现 Android 平台下的多用

户视频监控。在本工程中，对于 Linux 系统，我们使用 Ubuntu 做为服务器构建平台，对于 Andriod 系统，我们是利用 smart-box 服务器构建平台，在 smart-box 上运行 Android 系统和 RTMP 多媒体视频监控服务器，由于这两个模块两种平台并不冲突，所以我们分成独立的两个模块两个平台分别进行设计。

3.1 系统总体设计

基于 RTMP 协议的流媒体视频监控服务器可以对用户感兴趣的监控场所进行实时监控，并且与 3G 网络或无线网络相连，其中设计的功能有视频采集捕捉、视频压缩编码、网络实时数据传输等等。由于网络环境复杂，数据传输的路径不确定等因素，在本系统的设计之初，考虑了一下几点。

(1) 清晰度：视频监控的质量高低与观看视频的清晰度直接挂钩，而清晰读与采集摄像头的设备好坏、视频压缩所选取的标准协议以及网络传输视频数据的效率都有着直接的关系。

(2) 连续性：这需要对采集视频模块进行控制，对高速缓存的控制，以及对监控终端的控制，从而来保证视频监控图像的连续性。

(3) 实时性：这是设计本系统的本质要求，数据必须实时的发送、到达、被处理，即为视频监控的延迟长短，好的实时视频监控系统无法接受过长时间的延迟。而监控的延迟与网络带宽，网络环境等因素有关。

系统总体设计图如图 3-1 所示，简单的来说，采集终端、网络服务器、监控终端这三个部分就构成了实时视频监控系统。服务器是采用 PC 机作为硬件支持，在其上构建一个 RTMP 网络服务器和视频压缩服务器，将压缩好的图像压缩打包封装，最后由 Socket 传输通信至监控终端。客户端采用带有 Android 操作系统的移动终端（只能手机、平板电脑）或者 Web 内嵌播放器形式的监控终端。

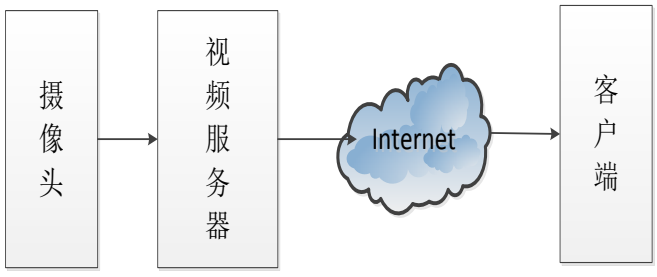

图 3-1 系统总体结构

服务器端采用的是 PC 机作为硬件支持，操作系统平台为 Linux 系统和 Android 系统。服务器端的主要功能有：在 Linux 系统下，通过摄像头驱动采集数据，压缩

编码，在由网络服务器进行 Socket 传输。在 Android 系统下，通过 JMF 完成视频采集捕捉过程，按照 H.264 标准对采集数据进行压缩编码，最后实现 Socket 传输通信。系统的服务器总体架构图如图 3-2 所示。

服务器框架主要是由四个大部分组成的，分别是底层的视频采集模块，对采集数据进行视频编码模块，通信双方协同遵守的流媒体协议模块和最后的上层发送通信模块。模块设计图如下。

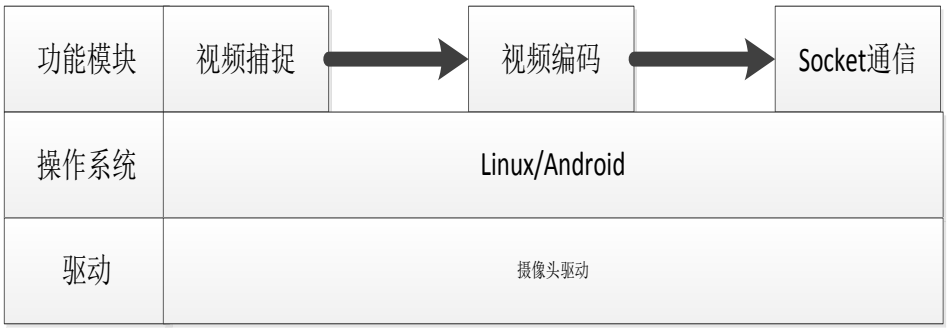

图 3-2 系统服务器端总体架构

3.2 FFmpeg 视频采集压缩功能模块设计

视频采集模块[21]是集成在RTMP多媒体服务器当中，根据RTMP服务器模块的端口监听信息，通过将H.264视频格式数据转换成FLV视频格式数据，为RTMP视频监控服务器与客户端的通信做好数据准备，并将采集的监控数据进行备份，定量定时的存于本地。故FFMpeg视频采集模块需要实现的功能有：

1）、判断是否有可用的摄像头，并将信息反馈给RTMP服务器。

2）、得到RTMP服务器的开始采集信息后，将H.264格式数据转换成FLV数据，并将打包好的数据发送给RTMP服务器。

3）、将采集好的视频信息以FLV文件方式定时定量的存于本地，即只存某段时间或某个特定大小的监控数据，以便于反查过往监控数据时利用，也避免了存储数据太多而导致服务器崩溃。FFMpeg视频采集压缩[22]功能模块图如图3-3所示。

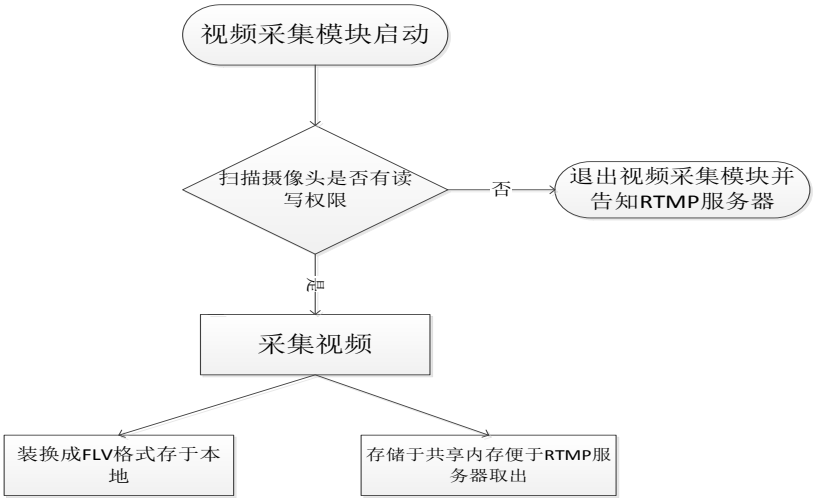

图 3-3 视频采集功能模块

3.3 Linux 系统下 RTMP 协议模块设计

Linux 系统下的 RTMP 多媒体视频监控服务器[23]模块主要功能有：

1）、对特定端口进行监听（此工程为 1935 号端口，此端口为 RTMP 协议通信的默认端口），对客户端的连接请求做出应答。

2）、与视频采集模块通信，控制视频采集模块何时开始采集数据。何时中断采集。

3）、将视频采集模块采集的数据进行封装打包成 RTMP 协议数据包。将视频数据以 RTMP 协议块的数据形式发送给客户端。

4）、记录不同客户端的信息，如 IP 地址，发送数据记录等，完成多客户视频监控功能。

Linux 系统下的 RTMP 多媒体视频监控服务器模块功能图如图 3-4。

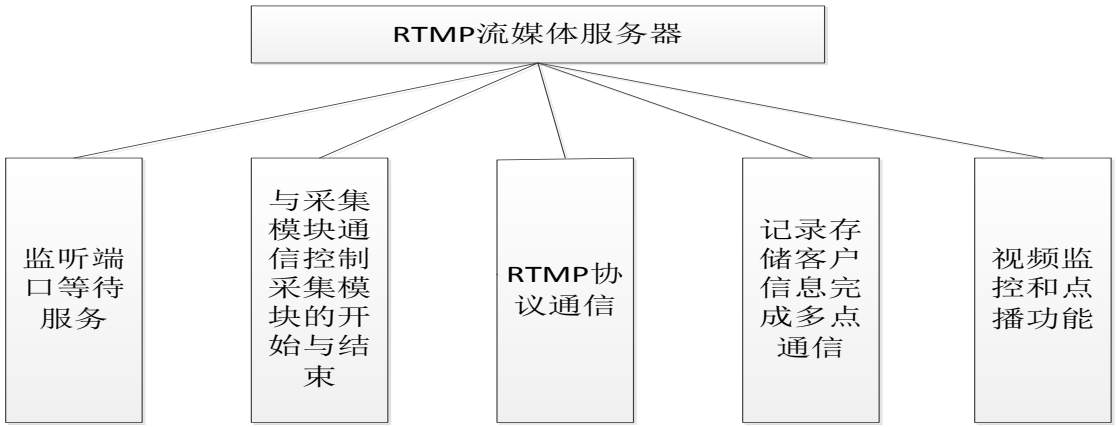

图 3-4 服务器功能模块图

下面介绍在 Linux 系统下的 RTMP 多媒体视频监控的服务器流程图，如图 3-5。

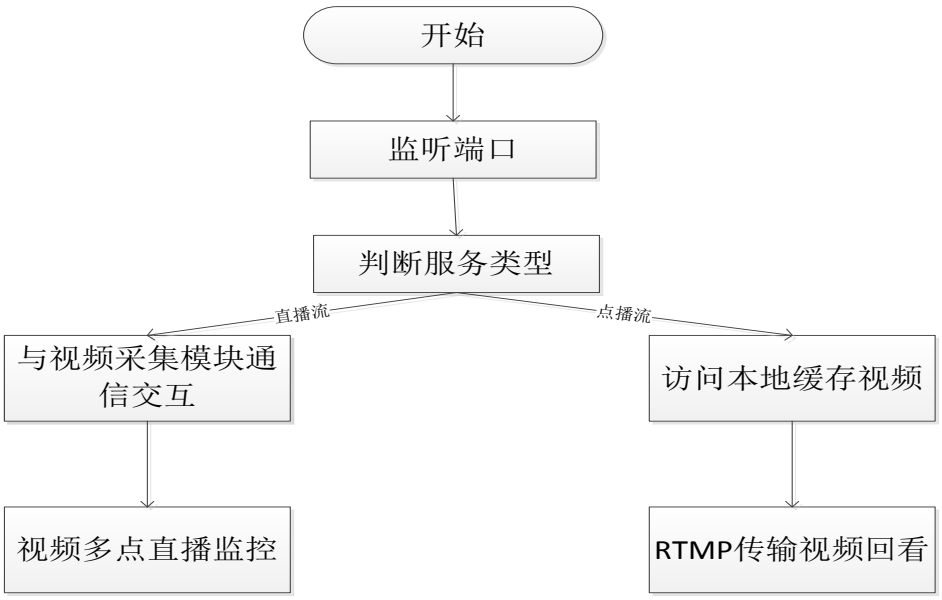

图 3-5 Linux 系统服务器总体流程

该图详细描述了在 Linux 系统下，RTMP 服务器是如何工作的，是如何完成多用户功能，是如何完成与 RTMP 客户端的通信，是如果完成与视频采集模块的协同工作的，启动服务器后，RTMP 视频监控服务器便在指定端口监听客户端的到来，当第一个客户端来后，记录客户端信息到客户缓存池，之后便开启视频采集模块功能，将采集到的视频打包成 RTMP 协议块包格式，并将视频数据发送给客户缓存池中的指定客户，之后再有客户到来，便再将客户信息存放于客户缓存池中，将采集视频数据往客户缓存池中发放到所有客户，实现多客户的视频监控，将客户逐个退出监控时，便将客户缓存池中的客户信息清空，当最后一个 RTMP 客户端退出监控时，便停止视频采集模块。

3.4 Android 系统下 RTMP 协议模块设计

Android 系统下，服务器端的视频采集模块的相关功能要通过由 Sun 公司研究开发出来的 JMF 框架来完成的，JMF 的作用是将收集完成的音视频数据添加到应用程序和 Applets 中，同时也能很好的获取视频流并将视频流播放。JMF 中提供了大量的工具包，透明了底层实现，能更便于开发调用者对其操作，屏蔽了底层细节。根据前两节的功能模块的设计与描述，已经设计出了 RTMP 视频监控服务器在 Linux 系统下的监控流程，此节描述将 RTMP 视频监控服务器移植到 Android

系统下的设计。思路为将 FFMpeg 库编码成 FFMpeg.so 动态库[24]，在 Android 上层应用程序中，开发一个应用程序，以启动一个服务，以 JNI 调用的方式，调用以 C 语言编写的 FFMpeg 和 RTMP 视频监控服务器，在 Android 下的 RTMP 视频监控服务器功能与在 Linux 一致，只是服务器实现流程有所不同，下图为在 Android 平台下实现的 RTMP 视频监控服务器的流程设计图如图 3-6。

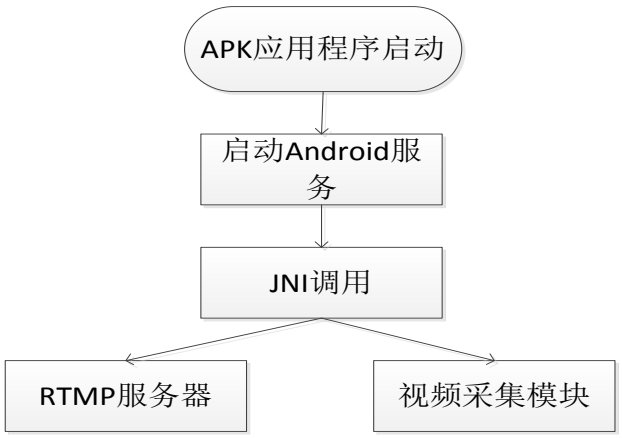

图 3-6 Android 系统服务器流程设计

图中详细描述了在 Android 系统下 RTMP 服务器[25]是如何启动，如何工作的，先是在上层应用程序中启动一个服务，再通过这个服务接口遂层调用 FFMpeg 库代码实现视频采集功能和 RTMP 协议的视频监控功能。

3.5 服务器与客户端操作界面流程图

3.5.1 Linux 系统下服务器操作界面

Linux 下，RTMP 流媒体服务器以控制台应用程序方式运行[26]，启动服务器后便开始监听，当客户到来时，判断服务类型，记录客户信息，启动视频采集模块，同时将 RTMP 服务器与客户端进行 RTMP 通信，显示客户信息，客户缓存池信息，采集数据信息等。当客户逐个退出时显示相应的客户缓存池信息，退出客户个人信息，以及采集模块运行信息等。

3.5.2 Android 系统下服务器操作界面

图 3-5 所示即为 Android 系统下服务器操作界面[27]图，启动 Android 下 APK，打开用户界面，点击启动按钮，同过打开一个本地服务在后台运行，再通过 JNI 调用运行 RTMP 网络服务器的 JNI 接口，是 RTMP 协议服务器在 Android 平台下

得以运行，让它监听指定端口，等待与客户端通信，客户来临后，启动视频监控模块与 RTMP 服务器通信模块相应功能，相应流程图如图 3-7。

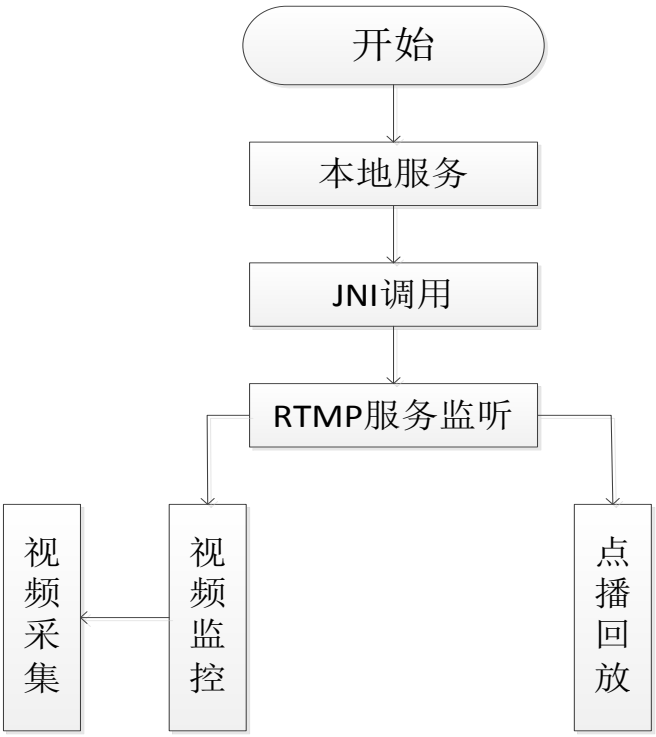

图 3-7 Android 系统服务器逻辑设计

3.5.3 客户端操作界面

客户端界面设计了两种形式，一种为 Web 网页内嵌播放器，另一种为运行 Android 系统的移动终端（智能手机或平板电脑等）。

对于网页内嵌播放器的形式，客户端操作界面以网页内嵌播放器的形式，在一个网页中设置了两个按钮，分别提供给客户两个不同的服务，即为“视频监控”服务和“点播回看”服务，点击不同按钮分别进入不同页面与服务器通信，将远端服务器监控采集的视频数据经过 RTMP 协议发送到客户端进行监控播放和点播回看，客户端支持播放与暂停功能，同时也可以动态的进行多点视频监控。

对于运行 Android 系统的移动终端，应用程序输入服务器的 IP 地址和端口号，向服务器发送监控请求，服务器接收到客户端的数据请求后，向 Android 移动终端发送监控采集的视频数据。Android 移动终端接收到数据后，调用 FFmpeg 相关库进行解码播放。

根据以上流程的分析，客户端主要包括了三个部分：第一个部分为用户与终端的交互，需要用户输入相应信息已连接服务器，此部分即为人机交互。第二部

分是连接上服务器后，接收从服务器传送过来的监控数据，即为视频数据接收。最后则是利用开源 FFmpeg 库对相应视频格式的数据进行解码播放，即解码显示部分。流程图如图 3-8 所示。

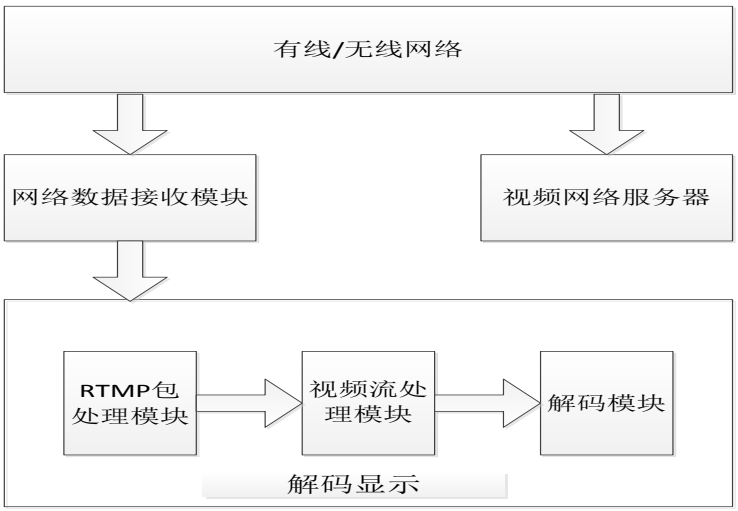

图 3-8 客户端总体结构图

客户端的主要功能就是通过用户设置好服务器的 IP 地址和端口号，来向服务器发出监控请求报文。在网络流媒体服务器接收到请求后，便开始采集视频并发送回客户端，客户端接收到以后利用 H.264 标准格式（本系统才用的是 H.264 实现编解码）进行解码播放。根据其上对 Linux 系统和 Android 系统的不同客户端数据流程的描述，其流程图如图 3-9 所示。

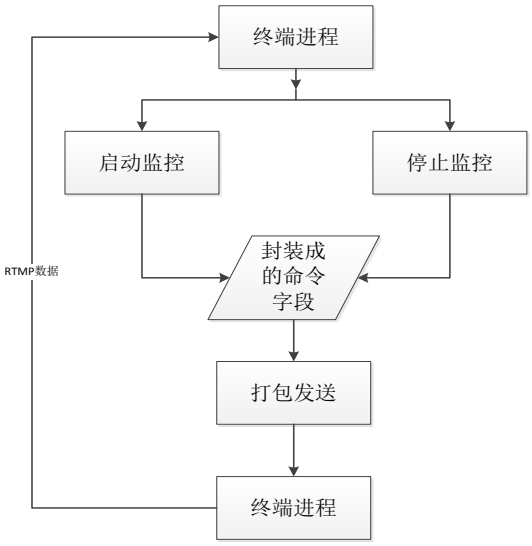

图 3-9 客户端数据流程图

3.6 本章小结

本章中主要对基于 RTMP 协议的多平台多媒体视频监控服务器的功能模块的设计与分析，主要采用的是模块化的思想，把功能进行模块化的分析实现，对于要实现的功能进行了划分，并在文中列出了详细的模块划分图。本章中将该工程主要划分为两个模块，视频采集压缩模块和 RTMP 协议通信模块，并对两个模块分别画出了详细的流程图，以及 Linux 和 Android 两个不同的平台的设计实现都画出了详细的流程界面图，参照该流程图我们可以知道整个设计的工作流程和工作方式，通过本章，对全文得到一个概括的思路。

第四章 基于 RTMP 协议的多平台流媒体视频监控报务器的实现

4.1 视频采集模块的实现

本工程利用了 FFMpeg 开源库实现视频采集压缩模块，扫描摄像头进行视频采集，将采集的 H.264 格式视频流数据进行 FLV 格式的封装，并将数据传递给 RTMP 流媒体服务器，同时将监控数据定时定量的存储到本地，FFMpeg 库附带了支持许多视频格式的解码器[28]，这里选用 H.264 格式并将其封装成 FLV 格式的数据，是因为 H.264 数据格式的压缩比高，对于网络传输有着更好的带宽利用率，非常适合网络传输，是多媒体编程中最常用的编解码方案，这一特性也一定程度的降低了监控时用户产生的数据流量，而 FLV 数据的帧封装格式，与 RTMP 协议通信的 RTMP 块流数据的打包格式相似，因此将 H.264 数据封装成 FLV 格式能更方便的进行 RTMP 协议通信。在此章节中重点介绍利用 FFMpeg 库进行视频采集模块的实现细节。

本工程在 Linux 系统下，利用了 V4L4 驱动接口及其相关控制指令，将摄像头的采集的 YUV422 数据映射到用户缓存区，由自己编写的 YUV422 转化 YUV420 函数将视频数据进行转化，通过开源库 FFmpeg 中 H.264 编码器对数据进行编码压缩。在 Android 系统下，要用到了 Java Media Framework 的概念和其功能，他是一个标准的扩展框架，允许用户制作纯音频流和视频流。

这里涉及到V4L2以及JMF的概念，V4L2是Linux上的驱动程序[30]，用于对设备进行操作控制，屏蔽底层设备特性，是开发人员可以方便的获取不同厂商摄像头采集的数据，对此驱动程序的了解有助于我们知道如何对摄像头监控数据进行采集，V4L2驱动程序提供给开发者一些指令，从而对摄像头进行格式制定的视频数据的采集，它将待访问的设备在开发者看来就想是Linux文件系统下挂载的一个文件一样，这样很大程度的方便了开发者，屏蔽了不同摄像头的访问差异性，在此工程中，路径名为/usr/include/linux/videodev2.h文件中定义的一些常用的结构体是和V4L2驱动程序密切相关的[31]，其中主要的结构体及其代表含义如表4-1所示：

表 4-1 V4L2 主要结构体

Struct v4l2_requestbuffers reqbufs	申请指定数量的帧缓存
Struct v4l2_capability cap	判断这个是否是输入视频设备
Struct v4l2_input input	视频的输入
Struct v4l2_format fmt	帧的格式，比如之前提到的高度等
Struct v4l2_buffer buf	代表驱动中的一帧数据
Struct v4l2_std_id std id	视频制式，如：V4L2_STD_PAL_B
Struct v4l2_queryctrl query	对查询的控制
Struct v4l2_control control	具体所需要的控制值
Struct v4l2_standard std	视频的制式，如PAL,NTSC

另外，V4L2 驱动程序[32]还定义了一些命令，来对设备文件进行相应的控制和操作，对这些驱动命令的了解有助于明白整个驱动控制逻辑，表 4-2 中介绍了开发过程中涉及到的主要命令及相关命令的解释。

表 4-2 V4L2 驱动命令

VIDIOC_QUERYCAP	该命令为查询驱动功能命令
VIDIOC_REQBUFS	该命令为请求摄像头分配内存命令
VIDIOC_S_CROP	该命令为设置视频信息的边框命令
VIDIOC_S_FMT	该命令为设置当前系统驱动所支持的数据格式命令
VIDIOC_G_CROP	该命令为读取视频信号的边框命令
VIDIOC_STREAMOFF	该命令为结束视频显示函数命令
VIDIOC_STREAMON	该命令为开始视频显示函数命令
VIDIOC_QUERYBUF	该命令为地址映射命令，即将命令VIDIOC_REQBUFS分配的缓存通过地址映射到用户空间
VIDIOC_ENUM_FMT	该命令为获取该系统驱动所支持的数据格式命令
VIDIOC_G_FMT	该命令为读取当前系统驱动的数据捕获格式命令

续表 4-2 V4L2 驱动命令

VIDIOC_CROPCAP	该命令为对驱动修剪能力的查询命令
VIDIOC_TRY_FMT	该命令为改变格式命令
VIDIOC_QBUF	该命令为从缓存中读取数据命令
VIDIOC_DQBUF	该命令为把数据放回缓存队列命令
VIDIOC_QUERYSTD	该命令为检查当前系统所需要的一些指标命令

而 JMF 框架的主要功能如下：

（1）、利用 JMF 功能可以使多种音视频格式在应用程序和 Java Applet 中播放。诸如 MPEG、AVI、MIDI、QuickTime、AU 等等。

（2）、JMF 能够实时的将网络上的视频音频流接收并且播放。

（3）、采集终端可以是摄像头、麦克风等设备，JMF 能识别并且利用这些终端设备可控的捕捉音频和视频，同时还能将采集的音视频数据保存在本地。

（4）、对接收或者发送的流媒体文件，可以进行相应的格式标准的转换。更利于网络传输和终端播放。

（5）、各种格式的流媒体文件 JMF 都能将其上传至互联网上。

4.1.1 设备扫描采集数据

FFMpeg 是一个开源的多媒体开发工具，其中包含了对多种音视频数据格式的编解码。本节主要讨论Linux系统下和Android系统下如何利用FFMPEG采集摄像头数据[29]的详细过程，具体流程图如图4-1。

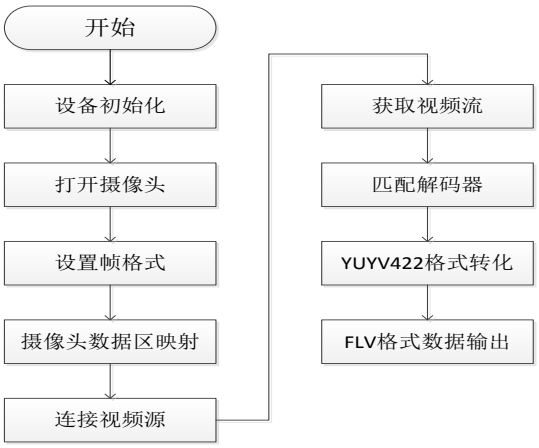

图 4-1 视频采集模块流程

对于摄像头的数据采集的具体流程，我们进行了如下解释，首先扫描并打开

本地路径的设备文件，即打开摄像头，判断是否有读写权限，本工程中路径名为/dev/video0，接下来便是获取摄像头的一些性能的参数，查看设备文件所具有的功能，判断是否具有视频输入等功能，若摄像头具有视频输入功能，则选择正确的视频输入方式，这是由于一个摄像头视频设备可能会包含有多重视频的输入方式，选择合适的视频输入方式，使采集数据能正确输入并收集。之后便是设置视频相应的帧格式，其中最为主要的也是我们最为关心的就是高度和宽度两个因素，
设置好后就可以动态的向视频驱动程序申请指定个数的帧缓冲，用于存放从摄像头传来的最原始的数据，一般个数不超过5个，接下来系统调用函数mmap()，将上一步申请到的对应的缓存映射到指定的用户空间，通过这一步的映射，就省去了以后要将采集到的数据从内核空间赋值到用户空间的步骤了。将我们申请的缓存放入队列这种结构体中，在队列中可以很方便的存放从摄像头获取到的数据。完成以上预备工作后，现在就可以对摄像头数据进行采集了，按帧采集完成后便可存放于队列，之后RTMP服务器便可从队列中获取所需要的视频监控数据，处理完数据后，将缓存重入队尾，循环利用所申请的缓存，最后摄像头视频监控采集停止，关闭视频设备，这是摄像头采集数据的全过程。

利用JMF框架的在Android系统下完成的视频采集模块的主要步骤如下：

（1）、由CaptureDeviceManager对象来得知系统中所将要用到的音视频采集设备，采集设备信息的由来就是通过查询CaptureDeviceManager对象。

（2）、调用getDeviceList()方法，其返回值是相应系统使用的采集设备的CaptureDeviceInfo对象。

（3）、利用上一步获得的CapturedeviceInfo对象获取MediaLocator对象，然后利用这个对象创建一个DataSource对象。

（4）、使用上一步创建的DataSource对象再创建一个和播放相关的Player对象或者是Processor对象。

（5）、最后调用start()方法，开始截取多媒体数据。

4.1.2 FFmpeg 压缩编码过程及函数实现

FFMPEG 所具有的主要功能就是提供了不同格式的编解码器，对不同音视频格式的要求都能完成相应的编解码功能。本节中，我们从函数代码的角度，详细介绍 FFMPEG 是如何将视频数据进行压缩和编码的。整个压缩编码流程如图 4-2。

这个流程图主要是领用 FFMPEG 提供的 API 将采集到的视频数据压缩编码成我们需要的格式，其中第一步便是像系统注册 FFMPEG 所支持的所有格式的编码器，这是利用 FFMPEG 库的前提，所调用的函数为 av_register_all()，此函数的调

用将所有的编码格式都注册到我们的开发环境中去，之后便通过 find_encoder()来打开工程所需要的指定的编码格式，同时分配一些存储压缩后的数据的缓存空间，最后调用其中的压缩编码函数功能，将从摄像头取得的视频数据压缩成指定格式的数据，便于存储利用，提高之后的网络带宽利用率，最后写入缓存中[33]，下面将对整个过程中所涉及的一些重要的函数进行详细的介绍。

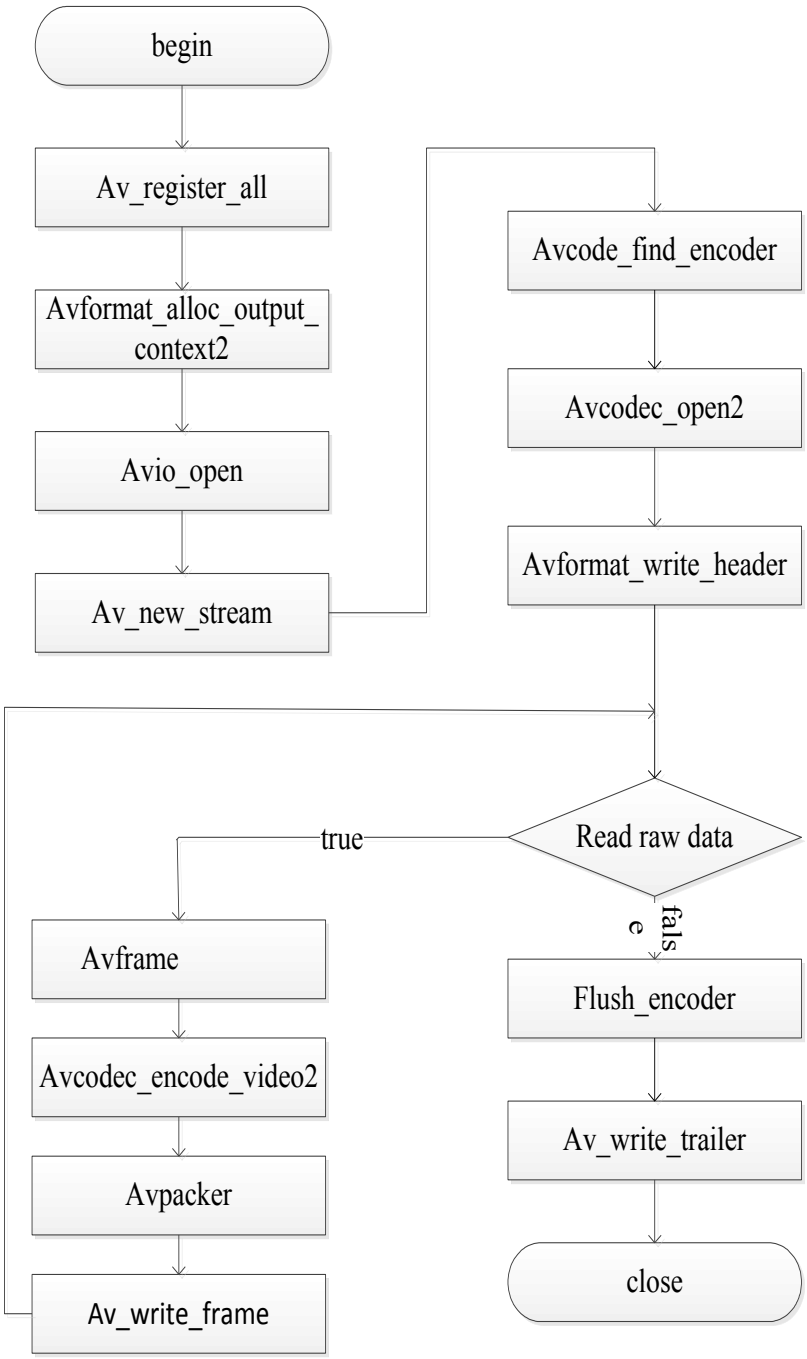

图 4-2 压缩编码函数流程

这个流程图主要是领用 FFMPEG 提供的 API 将采集到的视频数据压缩编码成

我们需要的格式，其中第一步便是像系统注册 FFMPEG 所支持的所有格式的编码器，这是利用 FFMPEG 库的前提，所调用的函数为 av_register_all()，此函数的调用将所有的编码格式都注册到我们的开发环境中去，之后便通过 find_encoder()来打开工程所需要的指定的编码格式，同时分配一些存储压缩后的数据的缓存空间，最后调用其中的压缩编码函数功能，将从摄像头取得的视频数据压缩成指定格式的数据，便于存储利用，提高之后的网络带宽利用率，最后写入缓存中[33]，下面将对整个过程中所涉及的一些重要的函数进行详细的介绍。

av_register_all()：顾名思义，该函数是注册所有 FFMPEG 支持的编解码器，它是所有要利用 FFMPEG 库功能的工程必须首先调用的一个库函数，该函数又被称作为注册复用器，FFMPEG 库中其他一些启用编码器和复用器的函数等等再该函数为调用之前都是无法正常使用的。

avformat_alloc_output_context2():该函数的具体功能是对 AVFormatContext 这个结构体进行初始化的，这个结构体是记录着我们采集的视频文件的信息参数的，如相应的文件格式、文件组织类型、文件长度、文件的访问时间等等，而此函数的主要作用就是初始化这些视频文件信息参数。

avio_open():当采集编码视频数据过程进行到要生成视频文件的时候，我们就需要调用该函数，打开我们将要编码后的视频数据写入到我们事先指定的文件中。

Av_new_stream():利用该函数，可以成功的创建一个新的视频流，这样我们可以将压缩编码后的视频数据流化，对于开发人员来说，对于流的处理则要方便的多。

Avcodec_find_encoder():从函数名我们就能够看出来，该函数的主要功能就是查找对应的编码器，编码器的 ID 作为函数参数传入其中，若函数调用成功则返回要求视频格式所对应的编码器，若函数调用失败则返回 NULL。

Avcodec_open2():该函数是根据上一个函数所找到的对应的编码器，从 avcodec_find_encoder()成功调用返回的对应编码器 ID，以 ID 作为参数传入到 avcodec_open2()中，并打开 ID 对应的编码器，为之后的编码工作做准备。

Avformat_write_header():当我们需要生成视频文件的时候，就需要调用该函数了，该函数是将指定的格式的文件的一些相关的信息作为文件头，写入到文件中去。

Avcodec_encode_video2():该函数是具体的编码函数，即编码一帧的视频数据，即将存储 YUV 像素数据的 AVFrame 结构编码为存储 H.264 等格式 AVPacket。

Av_write_frame():顾名思义，该函数是将编码后形成的新数据，即编码后的视频流写入到文件中。

Av_write_trailer():将对应文件格式的文件尾信息写入到文件中，当然，对于一些没有文件尾的封装格式的文件，是不需要调用此函数的，想 MPEG2TS。

在 avcodec_encode_videio2()函数中，是将原始数据进行 H.264 标准格式的编码函数。H.264 编码功能流程图如图 4-3 所示。H.264 编码器主要是由网络抽象层和视频编码层组成。视频编码层（VCL）负责的是按照 H.264 标准，将音视频进行高效的压缩编码。VCL 的处理对象是 16*16 像素为单位的宏块，使用的方法有基于块的运动补偿预测、变换编码、熵编码。而网络抽象层则负责将 VCL 截取的数据进行分割和打包封装，发至相应的网络适配器接口。

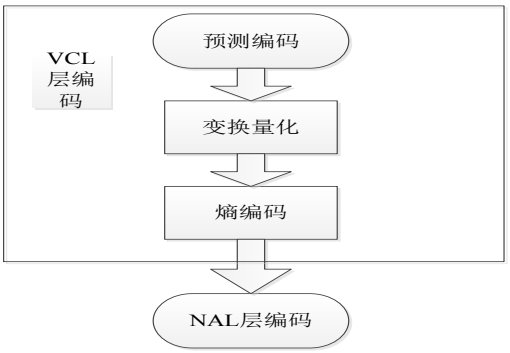

图 4-3 H.264 编码流程图

H.264 标准编码器的编码函数流程图如 4-4 所示。

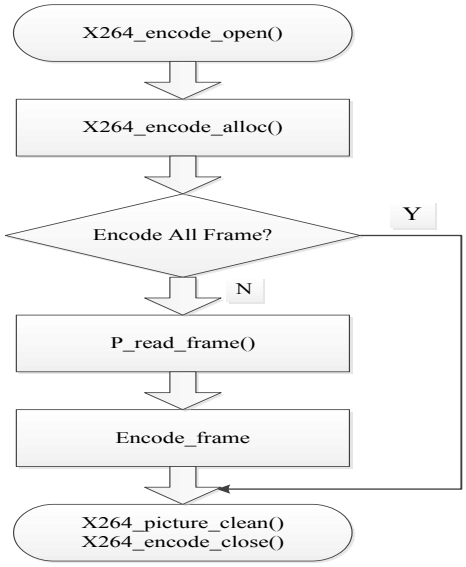

图 4-4 编码函数流程

X264_encoder_open()函数是打开编码需要的相应结构体并对其进行初始化。X264_picture_alloc()函数是给调用者分配并且初始化了一帧字节数的空间。

P_read_frame()函数是通过一次一帧的读取方式，来读取摄像头采集的音视频数据，然后将数据存入 x264_picture_alloc()申请的缓存空间中。

Encode_frame()函数则是具体的按照 H.264 标准对一帧数据进行编码，将数据转化成 H.264 格式数据。

4.2 Linux 平台下流媒体服务器系统的实现

在 Linux 系统下用于视频监控的流媒体服务器[34]主要有两个模块，一个是视频采集压缩模块，另一个是基于 RTMP 协议的网络服务器模块，本节讨论基于 RTMP 协议的网络服务器模块的架构和实现细节。

4.2.1 基于 RTMP 协议的网络服务器的架构分析

本工程所讨论的流媒体服务器的主要设计实现难点就是网络服务器[35]的架构设计和 RTMP 协议的实现，一个好的网络服务器的框架，能减少很多网络问题，如网络数据包的延迟，数据包传输过程中的丢失，UDP 报文不可靠等等，同时，一个好的网络架构也能为之后的功能扩充做一个很好的架构铺垫。本文所涉及的网络服务器的架构流程图如图 4-5。

如图所示，当启动整个的 RTMP 流媒体服务器时，首先要进行的是网络监听，等待客户端的到来以提供 RTMP 服务，此时视频采集模块是没有启动的，当服务器收到一个连接请求时，若请求服务类型为直播流时，便开始判断是否此次连接请求是服务器上的第一次连接请求，通过这个判断可以告知服务器是否需要启动视频采集模块，若是第一个连接客户端，则启动视频采集模块进行视频采集，并等待视频采集模块的信号，当采集模块采集到数据并放入到缓存区时，采集模块会发送信号给 RTMP 服务器，告诉服务器此时缓存有数据，可来读取。若不是则说明已经有若干个客户端正在和服务器通信，同时也说明视频采集模块已经启动，此时可以直接将客户信息放入客户信息缓存区，待发送监控时想客户缓存区提取客户信息一并发送，当从数据缓存中读取数据并打包封装成 RTMP 块包时，每封装一帧数据包就要判断一次连接是否断开，这样做的目的是可以最短时间内让服务器知道网络连接状况，以至于服务器不会做无用之功，当判断出网络连接正常时，便按照 RTMP 协议，以块的形式发送监控数据，另外，若服务类型为点播流时，此时，服务器则打开存储于本地的 FLV 文件，获取其中 TAG 信息，并将取得的信息封装成 RTMP 块包，最后再按照 RTMP 协议格式，将封装好的块包发送至远端客户端，以上便是整个服务器的架构流程[36]。

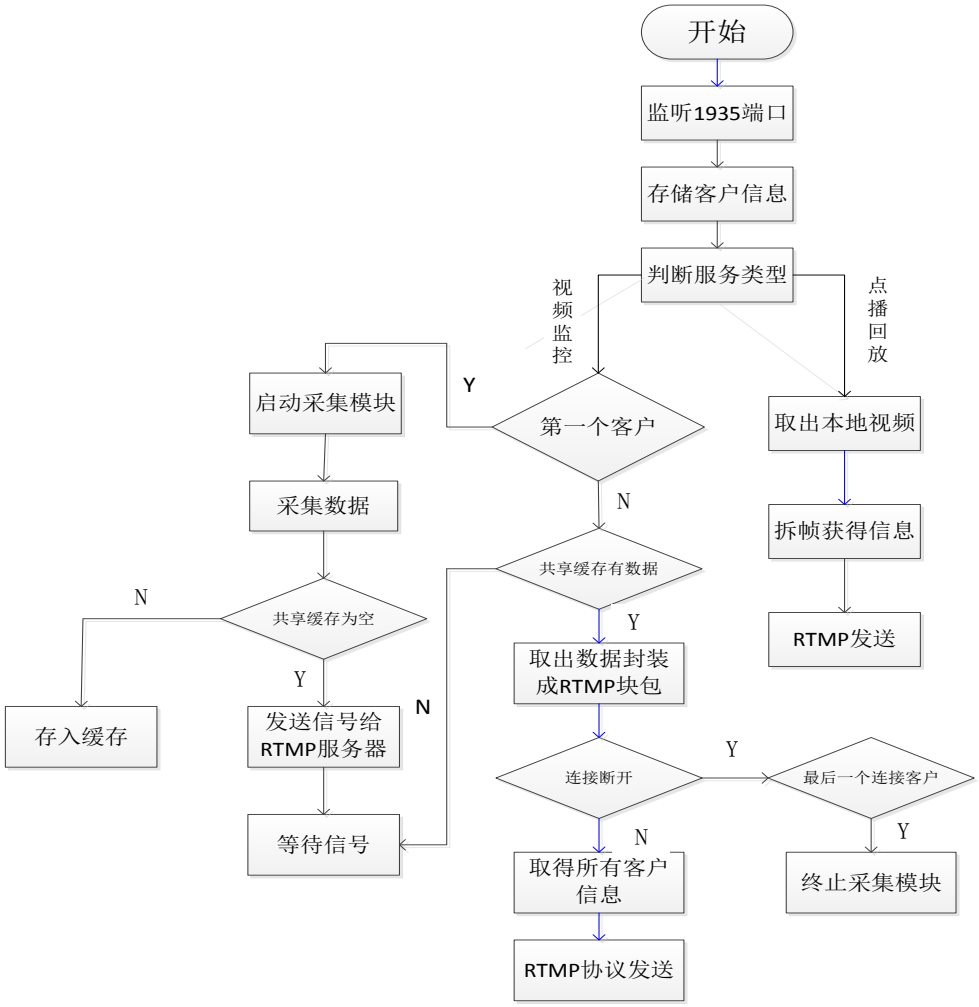

图 4-5 网络服务器架构

4.2.2 网络服务器的实现

根据前一节所描诉的流媒体网络服务器的架构分析，这一节讨论具体的实现[37]过程，我们必须先启动 RTMP 服务器的监听端口，以监听即将到来的监控请求服务。和网络监听的一般流程一样，也要经历申请套接字 ID，绑定端口，监听等一系列过程，代码流程图如图 4-6 所示。

在 startStreaming()函数中将会生成一个套接字标识符，并与本地 IP 地址以及 1935 号（RTMP 服务默认监听端口）端口绑定，对即将到来的请求进行等待监听。

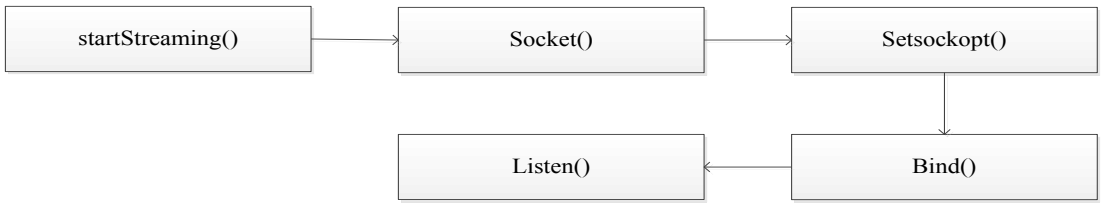

图 4-6 网络服务器代码流程

当请求来临时，动态生成一个 STREAMING_SERVER 结构体，该结构体细节如表 4-3 所示：

表 4-3 STREAMING_SERVER 结构体

Int	Socket
Int	State
Int	streamID
Int	Arglen
Int	Argc
Uint32_t	Filetime
AVal	Filename
Char	*connect

该结构体是用于与客户端通信，记录连接通信的细节信息的结构体，如套接字 ID，通信通道 ID，最后一次下载数据时间记录（用于点播），最后一次下载路径等等，之后便单独启动线程负责来与对应客户端通信传输数据，当线程 accept 到客户端信息时，便把客户端信息存入客户端缓存池，如 IP 地址，端口号等等，便于下次连接的判断以及复用。进入 doServe（）后，首先要动态生成一个 RTMP 结构体实例，这个结构体非常重要，可以理解为与客户端建立会话的实例，是存于服务器上记录会话信息的 RTMP 协议结构体,其中记录了输出输入块大小，流 ID，时间戳等一系列信息。

调用 RTMP_Init()初始化 RTMP 结构体后，赋之相应的初始值便正是进入 RTMP 协议的内容，与之相关的函数流程图如图 4-7。

进入 RTMP 协议内容后，首当其冲的就是 RTMP 协议握手，这不同于 TCP 协议的 3 次握手，是 RTMP 协议规定的以 RTMP 协议方式达成的服务器客户端交换信息，告诉双方即将进行的是 RTMP 协议服务，数据传输，当然，整个 RTMP 协议是建立在双方 TCP/IP 协议通信之上的，这里利用函数 RTMP_Serve()来完成握手功能。

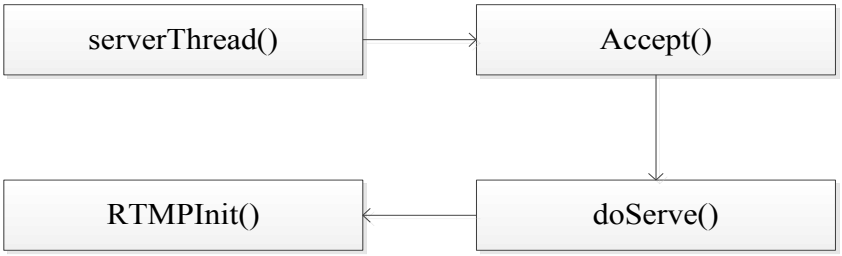

图 4-7 RTMP 协议函数执行流程

服务器端先读取客户端传来的 1 字节数据，即为双方约定的 RTMP 通信的版本号。

之后再记录服务器的本地时间戳，便与客户端进行第一次握手，完成之后继续 ReadN（）读取客户端内容进行第二次握手，直至握手完成，流程图如图 4-8。

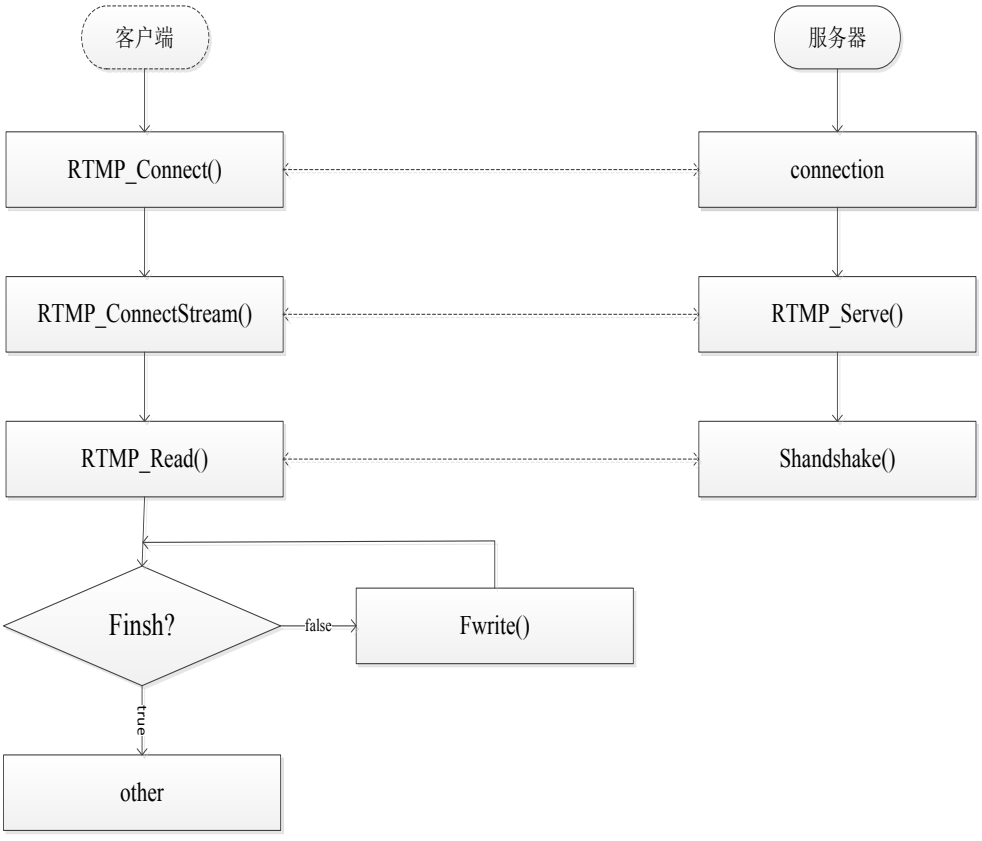

图 4-8 RTMP 握手函数逻辑

可以看到，RTMP 握手过程，是通信双方达成协议的过程，整个握手过程的重点就是服务器端的 ShandShake()函数，和客户端的 HandShake()函数，流程图中，

将客户端的 HandShake()函数拆分为 RTMP_Read()，Finish，Fwrite()三个环节是便于理解，当握手开始时，及客户端进入 HandShake()函数后，客户端发送 C0、C1 块，服务器收到二者之一后发送 S0 和 S1 块，接下来，当客户端收齐 S0 和 S1 后，开始发送 C2。当服务器收齐 C0 和 C1 后，开始发送 S2。最后，当客户端和服务器分别都收到 S2 和 C2 后，RTMP 协议握手阶段宣告完成。客户端进入下一个阶段，发送一些命令信息或者一些请求命令等。

握手完成后服务器线程再通过循环判断连接是否断开来持续的为客户端提供 RTMP 协议服务。进入 ServePacket（）后，会根据 RTMPPacket 结构体中的 m_packetType 数据类型字段来提供不同服务，数据类型有以下若干种，有改变通道包大小的类型、音频类型、数据类型等等，对于本工程，由于是要利用 RTMP 协议完成视频数据的传输，故这里客户端最终会向服务器发来 play 命令。下面将讨论 RTMP 流媒体播放命令所要涉及的通信过程以及通信报文。RTMP 协议规定，要使用 RTMP 协议播放一个流媒体有两个步骤前提，其一，需要建立一个网络连接（NetConnection)，这个网络连接代表的是服务器端应用程序和客户端之间的基础的连通关系，值得一提的是服务器和客户端智能建立一个网络连接。其二，建立一个网络流（NetStream)。当握手完成之后，客户端与服务器便开始建立网络连接，其具体步骤如下：

1）、客户端发送命令消息中的“连接”(Connect)命令到服务器，同过这个命令，告知服务器，客户端请求与一个服务应用实例建立连接。

2）、当服务器接受到上一步客户端发来的 connect 命令消息后，便发送确认窗口大小协议至客户端，同时连接到在连接命令中提到的应用程序。

3）、服务器再次发送设置带宽协议消息到客户端。

4）、当客户端处理设置带宽协议消息后，也要发送一个确认窗口大小协议消息到服务器端。

5）、服务器发送用户控制消息中的“流开始”消息到客户端通知客户端即将开始创建网络流。

6）、最后，服务器发送命令消息中的“结果”(_result)字段，通知客户端连接的状态。

建立完网络连接后，通信双方便开始建立网络流（NetStream），建立网络流的通信过程如下：

1）、同样，首先客户端需要发送命令消息中的“创建流”(createStream)命令到服务器端。

2）、当服务器接受到客户端在上一步发送的“创建流”命令后，便发送命令

消息中的“结果”(_result)字段，通过这个字段通知客户端流的状态。

建立网络流的过程相对简单，之所以这样设计是因为客户端与服务器之间可以创建多个网络流，但只能创建一个网络连接。接下来便是发送“播放”(play)命令，具体过程如下。

1）、客户端向服务器发送命令消息中的“播放”(play)命令。

2）、接收到播放命令后，服务器发送设置块大小（ChunkSize)协议消息，经过测试发现，此协议消息也可不发，客户端与服务器仍然可以正常通信，发送设置块大小协议消息的目的是可以根据不同的网络环境，对应的设置大小不同的“块大小”，这样可以提高 RTMP 数据传输效率，也可以提高网络的带宽利用率。

3）、服务器发送用户控制消息中的“streambegin”，通过这条消息，服务器可以告知客户端使用的流 ID。

4）、在这一步中，如果播放命令成功的话，服务器将发送命令消息中的“相应状态”消息，即为 NetStream.Play.Start 和 NetStream.Play.reset。服务器通过这一步中发给客户端的信息来告知客户端整个播放命令已经执行成功。

5）、在此之后，便是按照 RTMP 协议规定服务器想客户端发送视频数据。

Play 命令客户端与服务器所需要交换的信息以及流程如图 4-9。

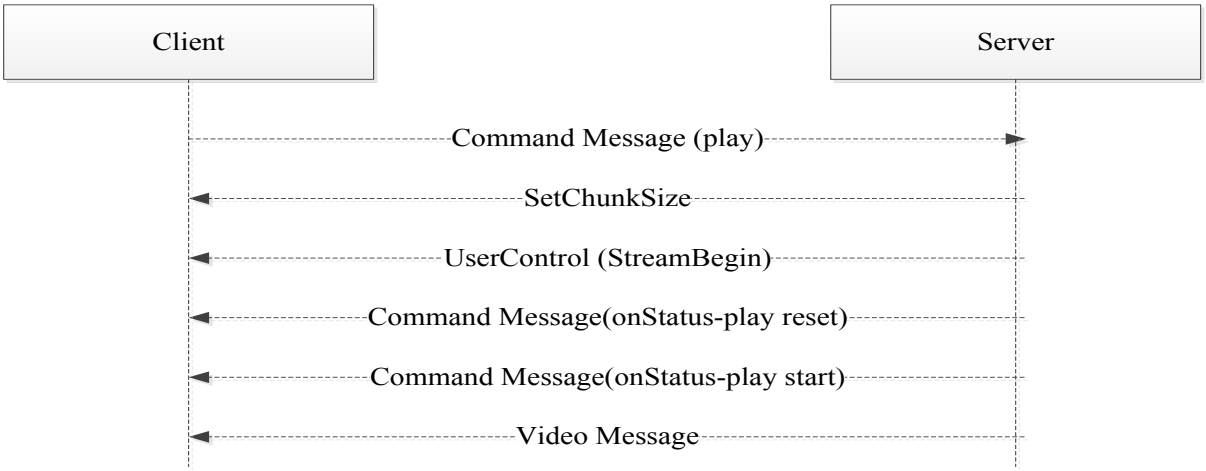

图 4-9 播放命令报文传输过程

这里主要进入 RTMP_PACKET_TYPE_INFO 分支，通过判断服务请求类型，这里是视频监控请求，经过上诉的信息交互后，最后的 Play 请求都为转化为 av_play 类型。进入 av_play 后，便将此次客户信息存入客户信息缓存池，如果是第一个连接到服务器的客户，便开始启动视频采集模块，在通过判断客户的请求类型，这里是通过发送给服务器的 URL 连接中的应用名来判断是直播流请求还是点播流请

求，如果不是第一个连接请求，只需要等待服务器将取出的视频数据发送至注册到客户信息缓存池中的所有客户，判断好相应分支后，便进入了本视频监控工程的主要功能分支，接下来面临的问题是如何将数据已 RTMP 协议规定的方式传输至客户端。以上通信过程主要代码流程图如图 4-10，发送视频数据之前用于发送命令消息的函数表如表 4-4：

图 4-10 发送数据函数流程

表 4-4 发送命令函数表

SendSetCheunkSizeP2()	设置块大小信息
SendResetResult()	发送 Reset 命令字段
SendPlayResult()	发送 Start 命令字段

接下来我们详细讨论如何将视频采集获得的 H.264 数据以 RTMP 形式打包并且发送到客户端，在此之前我们必须明白 FLV 视频格式的细节，应为我们要将视频采集的数据以 FLV 视频格式存放于本地，支持远端客户点播复看，并且同时，FLV 数据格式与 RTMP 发送数据所需要的信息内容相似，因此明白 FLV 数据格式细节是完成 RTMP 协议发送数据的前提，那么 FLV 数据格式是以怎么样的形式内容排列的呢？FLV 格式的音视频数据是以 tag 的形式逐个的排列的，整体图如图 4-11 所示。

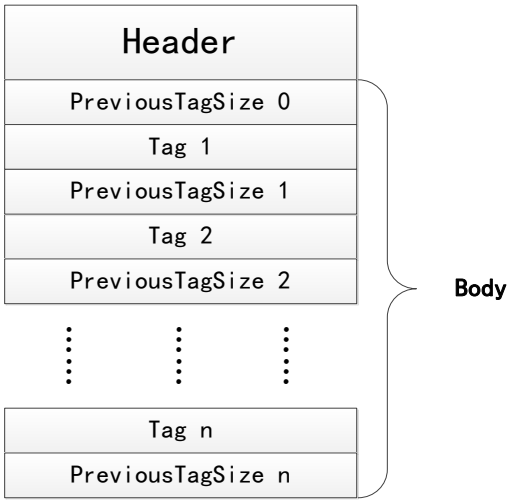

图 4-11 FLV 数据组成

FLV（Flash Video）是现在非常流行的流媒体格式，由于其视频文件体积轻巧、封装播放简单等特点，使其很适合在网络上进行应用，目前主流的视频网站无一例外的使用了 FLV 格式。另外由于当前浏览器与 Flash Player 紧密的结合，使得网页播放 FLV 视频轻而易举，这也是 FLV 流行的原因之一，FLV 视频格式是 Adobe 公司设计开发的，目前已经免费开放，下面我们就了解以下 FLV 文件格式，FLV 的格式很简单，是由 header 部分和一个个 tag 组成的，其中 Previous Tag Size 紧跟在每个 tag 之后，占 4 个字节表示一个 UI32 类型的数值，表示前面一个 tag 的大小，需要注意的是，Previous Tag Size0 的值总是为 0。而 tag 的类型又分为 3 种：音频数据、视频数据、脚本数据。本工程主要讲解的是视频数据。每个 tag 的信息包含着 tag 类型、数据长度、时间戳、扩展时间戳、StreamsID 和数据区，这些信息的占用空间和解释如表 4-5 所示：

表 4-5 FLV 数据组成

名称	长度	介绍
Tag 类型	1bytes	8:音频；9：视频；18：脚本；其他：保留
数据区长度	3bytes	在数据区的长度
时间戳	3bytes	整数，单位是毫秒，对于脚本型的 tag 总是 0
时间戳扩展	1bytes	将时间戳扩展为 4bytes, 代表高 8 位
StreamsID	3bytes	总是 0
数据区（data)	由数据区长度决定	数据实体

介绍完 FLV 数据的格式后，我们回到正题，如何将 FLV 数据信息进行提取，以便于 RTMP 协议的利用，前面讲到，这里我们已经启动了视频采集模块，并将采集的视频数据以 FLV 格式存放于本地，而另一模块 RTMP 服务器是和视频采集模块共享一段数据缓存区的，缓存区有了视频采集模块采集的数据，便发送信号给 RTMP 流媒体服务器，有数据可取出，即有视频监控数据可发送至客户端。当然，如前所述，在取出 FLV 格式数据之前服务器必须与客户端进行 RTMP 协议通信，发送一些协议控制命令字段，如改变通道包大小命令，ResetResult 命令和 PlayResult 命令等。

完成控制命令的通行交互后，服务器便开始从共享数据缓存读取数据，进行

信息提取分离赋值，主要代码如下：

```
GetFromQueue()                    //从共享缓存中提取数据
read_live_type()                  //获取数据帧类型
read_live_datalength()            //获取数据长度字段信息
read_live_time()                  //获取时间戳字段信息
read_live_streamid()              //获取 Streamid 字段信息
```

这里是从 FLV 数据的每个 tag 中按字节读取数据类型、数据长度、数据时间戳、StreamID 以及数据内容。并将这些信息封装到 RTMPPacket 结构体所实例化的结构中去，最后在判断连接是否断开，若连接正常则通过 RTMP_SendPacket（）函数将 RTMPPacket 数据包发向至客户端。以上是如何将采集视频进行有效信息提取并封装打包的全过程，而 RTMP_SendPacket()是按 RTMP 协议规则对数据进行发送，下面介绍如何代码实现 RTMP 消息快发送的全过程。提及 RTMP 消息快的发送，本工程中，发送 connect 命令使用的函数是 SendConnectPacket()，发送 createstream 命令使用的是 RTMP_SendCreateStream()，发送 realeaseStream 命令使用的是 SendReleaseStream()，发送 publish 命令使用的是 SendPublish()等等，在此不再一一列举，这里用到的具体思路都是将命令进行 AMF 编码，在声明一个 RTMPPacket 类型的结构体，然后设置结构体中的数值属性，最后统一交给 RTMP_SendPacket()进行发送，其中 RTMPPacket 结构体非常重要，本工程中所有要发送的 RTMP 协议命令信息，数据信息都需要打包成 RTMPPacket，最后统一发送，RTMPPacket 类型的结构体定义如下，一个 RTMPPacket 对应 RTMP 协议规范里面的一个快（Chunk)。

```
typedef struct RTMPPacket
{
 uint8_t    m_headerType;uint8_t    m_packetType;uint8_t m_hasAbsTimestamp;
 int    m_nChannel;uint32_t    m_nTimeStamp;int32_t    m_nInfoField2;
 uint32_t    m_nBodySize;uint32_t    m_nBytesRead;RTMPChunk    *m_chunk;
 char    *m_body;}RTMPPacket;
```

其中 m_headerType 字段：块的 4 中类型字段。

m_packetType 字段：消息类型 ID（1-7 协议控制消息；8,9 音视频；10 以后为 AMF 编码消息）

m_hasAbsTimestamp：时间戳字段

m_nChannel:块流 ID 字段

m_nInfoField2:消息流 ID 字段

m_nBodySize:消息长度字段

以上解释了 RTMPPacket 结构体中的一些关键字段，这些信息都记录着 RTMP 通信所必须的通道，数据，时间等信息。

明白了 RTMPPacket 结构体，接下来讨论 RTMP_SendPacket()如何发送 RTMPPacket 包的，该函数收是通过获取 ChunkMsgHeader 的类型，并与前一个 Chunk 对比，确定时间戳，确定包头大小等一系列过程进行发送的，具体的说就是按照 RTMP 协议规定将消息按块传输，之后又利用 WriteN()函数发送每一个块消息，在 WriteN()函数中我们又调用了 RTMPSockBuf_Send()函数来完成数据的发送功能，最终，我们调用了系统 Socket 的 send()函数完成了数据的发送。代码流程图如图 4-12。

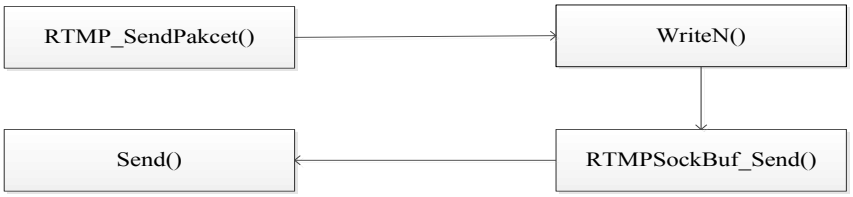

图 4-12 发送数据函数流程

以上讲解了如何利用 RTMP_SendPacket()来发送 RTMP 块包数据的，那么 RTMP 流媒体服务器又是如何接受 RTMP 块包数据的呢，本工程是利用 RTMP_Read()函数通过层层调用来实现 RTMP 协议数据的读取的，读取数据代码流程图如图 4-13。

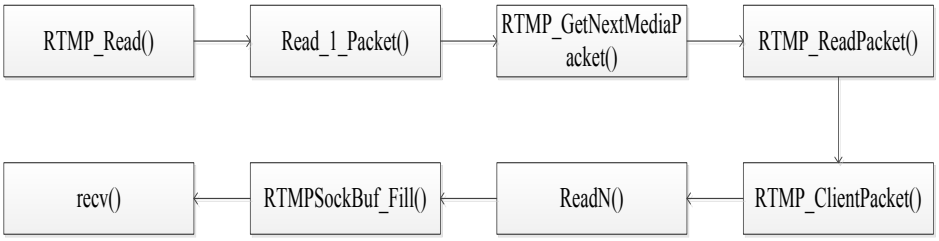

图 4-13 读取数据函数流程

如图所示，RTMP_Read 首先通过 Read_1_Packet 以及 RTMP_GetNextMediaPacket 来从网络上读取一个 RTMPPacket 的数据并进行一些数据的设置，属性的判断，接下来便是 RTMP_ReadPacket 和 RTMP_ClientPacket 函数，这两个函数中，前者是负责从网络上读取数据，后者是负责对读取的数据进行相应的处理，这部分与建立 RTMP 链接的网络流的时候很相似。其中

RTMP_ClientPacket 函数可以说是消息分流函数，处理了 RTMP 各种不同的消息，利用一个 switch case 语句堆 RTMP 各种不同的消息类型做了判断，如 0x 08 代表的是音频数据，0x 09 代表的是视频数据，本工程中，从发起一个 RTMP 链接到接受视频数据要处理很多消息，都要经过这个函数进行判断。在 RTMP_ReadPacket 函数中，是以字节为单位，逐个的读取，一直重复着简单的“劳动”，直至读取数据完成。最后仍然需要调用 Socket 的 recv 函数来从网络中读取数据。

而这些 RTMP 协议封装的视频数据最后都会进行 AMF 格式压缩，在客户端和服务器端分别会对接收到的 AMF 格式数据进行解压缩还原。AMF 协议是 Action Message Format 协议的简称，即动作消息格式，是一种数据格式规范，针对的是二进制格式数据。AMF 类似于 WebService 中的 XML，区别在于前者为二进制数据，后者为文本数据。但他们本质上都是用于数据交互和远程过程调用。下面介绍两类工程中涉及的编码规则。

1）、无符号的可变长度的整数，编码位数是 29 位。

AMF 协议采用了一种压缩方法，将整数的表示进行压缩，使得有限的表示位数能表示更大范围的整数数据。AMF 规定一个 4 字节的整数，它的前 3 位将拿来表示最后一个字节是否仍然用来表示这个整数，这样的规定标准使得 AMF 编码的表示整数范围在 2^{29}~1。而这一切的规范都能使得网络传输效率变高，高压缩比数据是网络传输不可忽视的问题。AMF 规范数字范围见表 4-6。表 4-6 无符号 29 位整数编码

编码整数范围（十六进制）	所需位数（二进制）
0x00000000~0x0000007F	0xxxxxxx
0x00000080~0x00003FFF	1xxxxxxx 0xxxxxx
0x00004000~0x001FFFFF	1xxxxxxx 1xxxxxxx 0xxxxxx
0x00200000~0x3FFFFFFF	1xxxxxxx 1xxxxxxx 1xxxxxxx xxxxxxx
0x40000000~0xFFFFFFFF	溢出

2）、UTF-8 编码格式的字符串数据

在 AMF 中，对于字符串数据，是采用的 8 位统一传输格式进行的编码，即 UTF-8 编码格式。UTF-8 编码风格是将字符串长度存放在字节头部，之后是一些可变长度的编码序列。通常将字节的第一位标识为字符串的类型。可分别存储为 0 或 1，UTF-8 编码表见表 4-7。

表 4-7 字符串和 UFT-8 编码

UTF-8 范围（十六进制）	所需位数（二进制）
0x00000000~0x0000007F	0xxxxxxx
0x00000080~0x000007FF	110xxxxx 10xxxxx
0x00008000~0x0000FFFF	1xxxxxxx 1xxxxxxx 0xxxxxx
0x00010000~0x0010FFFF	11110xxx 10xxxxxx 10xxxxxx 10xxxxx

在 AMF3 编码协议中的数据类型一共有 13 个，其所有类型极其解释见表 4-8。

表 4-8 AMF3 数据类型

数据类型	标识（十六进制）
Undefined type(未定义类型)	0x00：表示未定义的数据
Null type(NULL 类型)	0x01：表示无信息
False type(false 类型)	0x02：表示 bool 数值类型数据
Ture type(true 类型)	0x03：表示 bool 数值类型
Integer type(整数类型)	0x04：表示整形数据
Double type(浮点型)	0x05：表示浮点数据
String type(字符串类型)	0x06：表示字符串数据
XML document type(XML 文档数据)	0x07：表示 XML 文档对象
Data type(日期数据)	0x08：表示日期数据
Array type(数组)	0x09：表示数组
Object type（对象数据）	0x0A：表示对象实体
XML type(XML 类型)	0x0B：XML 数据，但是会被字符表示
Bytearray type(字节数组)	0x0C：字节数组，但是前缀会用整数表示

4.3 Android 平台下流媒体服务器系统的实现

Android 相对其他操作系统 Windows Phone 和 IOS 等，具有非常大的优点和优势，在优势方面，Android 平台首先就是其开放性，开发的平台允许任何移动终端厂商加入到 Android 中来，其次 Android 系统的市场份额大，超过 80%，iOS 不到 13%，而 Windows Phone 则不到 4%；另外 Android 上的应用程序发展迅速，使得 Android 的推广流行更省与其他平台，等等因素使得 Android 系统较比与其他系统更加的受到开发者的青睐。本节中介绍如何在 Android 系统下完成基于 RTMP 协

议的流媒体服务器，以及本工程在 Android 系统下的开发逻辑和实现流程。

4.3.1 Android 平台 JNI 调用的实现

本工程在 Android 下实现主要是同过在 Android 上层通过 java 代码来调用底层 RTMP 流媒体服务器的 C 代码来实现的。这里就必须要同过 JNI 调用，即 java 本地调用，它使得 Java 虚拟机内部运行的 Java 代码能够与其他编程语言编写的应用程序和库进行交互操作。通过 JAVAH 命令将 RTMP 服务器接口函数封装成 JNI 函数，在通过上层服务接口来步步调用以实现服务器功能。

在 Android 系统中的 JNI 调用是让上层的 Java 应用程序可以调用底层用 C/C++ 语言编写的语言程序。在 Android 系统中喝多 Java 都是有本地接口的。并将这些本地接口注册到系统中去。JNI 调用过程在 Android 系统的层次结构中的逻辑图如图 4-14 所示。在本系统中。Java 层提供对上层的应用接口这里是 NoticeService.start()，使用的是 java 语言编写的，网络服务器是由网络数据层来完成，负责 RTMP 数据的接收和传输工程，是利用 C 语言进行编写。

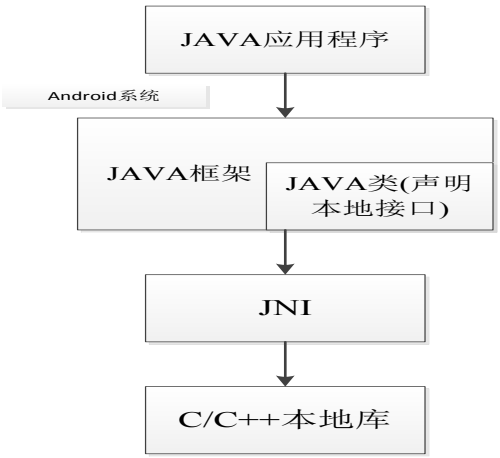

图 4-14 JNI 结构图

流媒体服务器在 Android 系统下的 JNI 调用流程从最上层到底层的调用函数流程图如图 4-15。

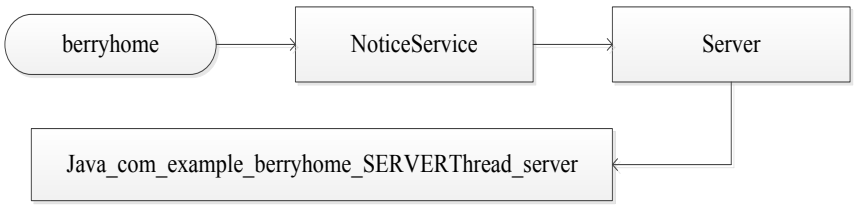

图 4-15 系统 JNI 函数调用过程

4.3.2 网络监控系统的移植与实现

由于在 Android 下实现的流媒体视频监控服务器是将前一章节描述的 RTMP 服务器进行了移植，所沿用了上章中所讨论的网络服务器的设计逻辑与内容，故本节中不在对网络服务器部分做过多赘述，重点讨论如何将代码进行移植和 Android 平台下逻辑思路的设计，首先要将 FFMPEG 库进行动态编译，生成动态库，这里还要提到的是，FFMPEG 结构庞大，如果 Android 平台是运行在时能手机、平板电脑这些终端的话，由于这些终端的资源宝贵且有限，这时就需要将 FFMPEG 进行有选择的编译，使 FFMPEG 不占用过多的资源，以使我们能够顺利的在 Android 端运行 FFMPEG 库，在这种情况下，可以对 FFMPEG 库进行精简删除，去除不需要的视频格式的相关代码，轻化 FFMPEG 库，使其便于移植到存量较小的移动终端上。在 FFMPEG 库中，libavcodec 和 libavutil 库是涉及对 H.264 视频格式的相关操作的库文件，对他们进行轻化缩减。第一种方法就是找到 FFMPEG 源码中关于 H264 的源文件，删除多余的文件，这样能缩减 FFMPEG 库的同时也能对 H.264 数据进行编解码操作，另外一种方法是在编译的时候修改 config.sh 配置文件的参数，这种方式也能选择性的对 FFMPEG 代码进行编译，config.sh 需更改的命令如下：

```
--disable-encoder=libx264 \              //屏蔽解码模块
--enable-decoder=h264 \          //使能解码模块，并使其能够解码所有264格式
--enable-muxer=h264 \
--enable-demuxer=h264 \
```

这般修改配置文件的参数后，再对FFMPEG源码编译时，FFMPEG就只会选择与H.264格式相关的编码功能进行编译从而生成库文件，这样一来，生成的库很大程度上减少了其代码量和所占用的内存，达到了节省存储空间和裁剪优化的目的。

当然，如若对资源充足的终端来说，就不必进行裁剪优化，可将 FFMPEG 整体编译，这样做的好处是便于以后的工程拓展与补充。

现在在 Android 系统下能成功调用 FFMPEG 库完成视频采集功能模块了，接

下来讨论在 Android 下如何调用 RTMP 网络服务器，这里的实现逻辑是开发一个 Android 应用程序，其中启动一个服务，在服务中通过 java 本地调用来调用生成的 RTMP 服务器接口的 JNI 接口。

4.4 客户端的设计与实现

4.4.1 网页内嵌播放器客户端的实现

本工程设计的客户端的一种形式是在网页中内嵌播放器的形式来进行视频监控，这样有着监控方便，操作简单，在任何有网络的地方，只要登录正确网站就能实现对 RTMP 流媒体服务器所采集的视频数据的监控，同时支持多点监控，这里选择的视频播放器是 JW Player，JW Player 是一款非常优秀的网页媒体播放器，H.264 (.mp4, .mov, .f4v)、FLV (.flv) 、3GPP (.3gp, .3g2)、OGG Theora (.ogv) 和 WebM (.webm)视频格式，MP3 (.mp3)、AAC (.aac, .m4a)、OGG Vorbis (.ogg) 和 WAV (.wav)音频，同时也支持 swf 和图片(gif 、jpg、png)和 YouTube 格式视频。故 JW Player 是非常适合本工程的需求的一款开源播放器，本工程的设计是在网页中设置按钮让用户选择是监控还是点播，若是监控则服务器传回监控视频，若是点播则服务器传回相应本地 FLV 视频格式文件。

4.4.2 移动设备客户端的实现

Android 应用程序是由 Activity、BroadcastRecevier、ContentProvider 和 Service 四个部分组成的。然而根据不同的应用程序的需要，有些 Android 应用程序只需要其中的部分模块。下面分别介绍 Android 应用程序的四个部分。

Activity：这是应用程序的入口，主要负责界面显示，接受事件（如按键），控制显示跳转。每一个 Activity 都是以一个独立的类的形式存在与应用程序中，而 Activity 的生命周期也和进程很类似，有创建（OnCreate）、开始（onStart）、暂停（onPause）、唤醒（onResume）、停止（onStop）、重启（onRestart）、销毁（onDestroy)等状态。如图 4-16 所示。

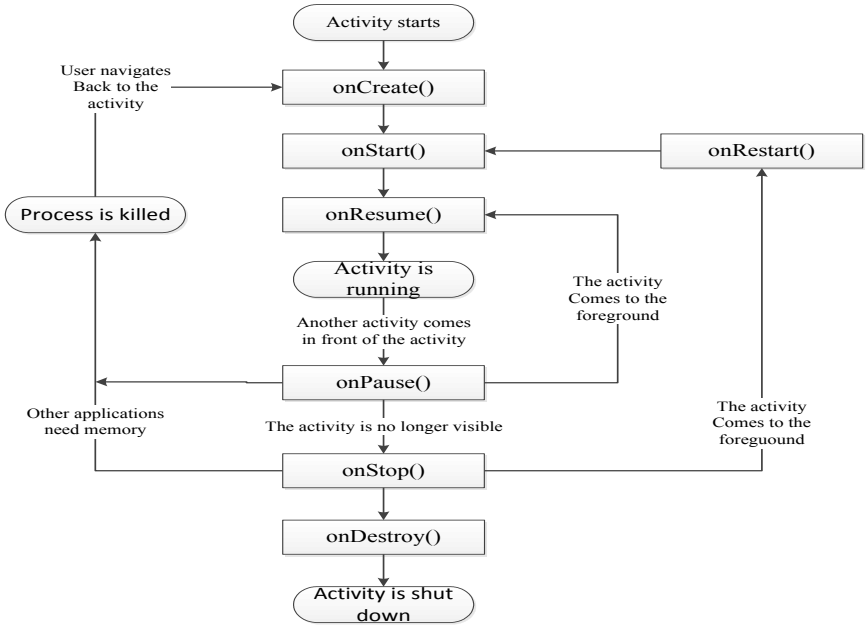

图 4-16 Activity 的生命周期

BroadcastReceiver：用于注册和接受系统广播的事件，同时也可以接受自定义的广播事件。

ContentProvider：提供数据库查询接口，有了这样的接口可以保证其他应用程序可以共享数据，同时可以对外（其他应用程序）公开数据查询服务。在 Android 系统中，每一个应用程序都有这自己独有的运行环境，都是在各自的进程中运行，当相互之间需要数据共享，数据通信时，就需要 Content Provider 在不同虚拟机之间进行数据传递，来解决程序间的数据共享问题。

Service：每一个 Service 都是一个独立的进程，可以一直单独的在后台运行，同时可以与其他组件（如 Activity、Service、ContentProvider）通过远程连接绑定（bind）进行交互。Service 组件还可以对外（其他应用程序）提供服务交互接口，并且生命周期不依赖其他组件的存在与否。

在 Android 平台的移动客户端上实现视频解码功能中。Android 上层的架构是不支持 FFmpeg 视频解码库的，因此需要用到 JNI 技术，使用上层的 Java 代码去调用底层的 FFmpeg 解码库。移动客户端的实现逻辑是首先通过网络协议（RTMP）接受网络视频数据，在截取一次解码的数据块，并且调用 H.264 解码器对视频监控数据进行解码，最后在将视频数据显示播放。解码流程图如图 4-17 所示。

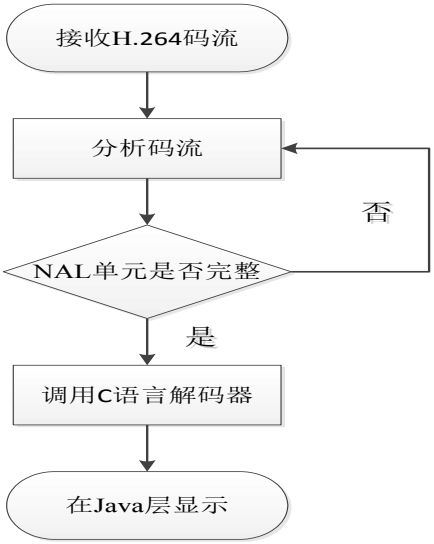

图 4-17 解码流程

在 JNI 调用中，Java 层的本地函数接口声明如下。

Public native int myinitDecoder(int wid,int heig);

Public native init myuninitDecoder();

Public native init myDecodingNal(byte[] inputdata,int inputSize,byte[] output);

这些函数接口分别实现了分配内存，解码器的初始化。释放申请的内存，和对 NAL 数据单元进行解码。他们分别对应的 JNI 接口函数如下：

Jint Java_com_example_berryhome_myInitDecoder(JNIEnv *env,jobject this,jint wid,jint hei);

Jint Java_com_example_berryhome_myUninitDecoder(JNIEnv *env,jobject this);

Jint Java_com_example_berryhome_myDecodingNal(JNIEnv *env,jobject this,jbyteArray input,jint nalLen,jbyteArray output);

通过以上三个函数在底层调用 FFmpeg 库中有关解码 H.264 数据的相关代码。

4.5 本章小结

本章内容主要是在前一章系统设计内容的基础上，对其进行了模块化的分析实现，本工程的实现共分为两大部分，视频采集模块的实现和 RTMP 流媒体网络服务器模块的实现，而这两个模块又分别在 Linux 系统和 Android 系统上得以实现，最后还谈及了监控客户端的设计与实现，在本章节中第一部分主要谈及了视频采集模块的细节和流程。第二部分着重介绍了本系统中 RTMP 协议模块的实现，对 RTMP 通信前通行中通行后的每个细节和工作流程都进行了分析，并且对从服务器启动到 RTMP 协议启动流程都有着详细介绍，之后谈及了将系统移植到 Android

平台上的方法和设计逻辑，以及如有需要则要涉及到的裁剪优化，最后则谈及了网页客户端的优势和实现。

第五章 系统测试

在这个基于 RTMP 协议的多平台流媒体监控系统的设计过程中，测试环节是不可缺少的一个重要环节，测试是对系统的稳定性、正确性、高效性进行判断的重要依据，对视频采集、H.264 编码，RTMP 视频传输等环节都要通过不断测试来调控。能客观的评估本系统的使用价值和适用领域，对系统作出正确的评价，以此为依据，更好的提高系统的容错性和准确性，使系统的性能更加完善，功能更加优化。

5.1 测试环境及测试内容

测试环境设备主要是 PC 机运行网络服务器，网页和移动只能设备作为客户端，多点进行的系统测试，测试环境如图 5-1 所示。

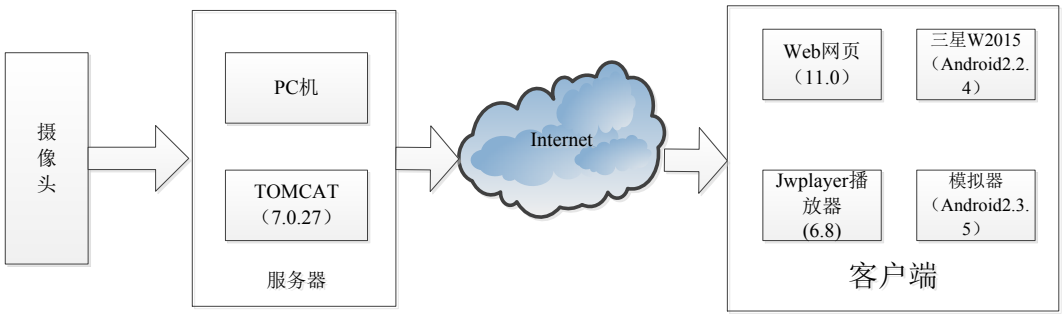

图 5-1 系统测试环境图

测试环境主要包括一下几点：

(1)、硬件测试平台：运行 Linux 系统 PC 机，运行 Android 系统的 smart_TV box。

(2)、软件测试平台：ubuntu10.10，Android4.0，JMF2.1，Tomcat7.0.27。

(3)、监控设备：YUV 摄像头。

(4)、网络传输媒介：有限网络，Wi-Fi，3G 网络。

(5)、测试终端：Web11.0，jwplayer(6.8)、(模拟器(Android2.3.5)、三星 W2015（Android2.2.4）。

测试内容有：采集视频的正确性与连续性测试；网络传输的实时性能、延迟问题、传输正确性测试、智能设备的接受数据解码显示播放性能测试。

5.2 Web 客户端测试步骤

将多台测试机与服务器构建在同一局域网内，测试多点视频监控。RTMP 服务器是存在与 PC 机上，运行的是 Ubuntu10.10，服务器 PC 机的 IP 地址为：192.168.1.100。让 RTMP 服务器开始运行。每台测试机打开 TOMCAT 服务器上的测试网页，打开方式是通过才本机上输入 URL 地址。

URL 地址为 http://192.168.1.100:8888/jwplayer/Remotemonitoring.html。点击播放按钮进行多点视频监控。监控效果如图 5-2 所示。

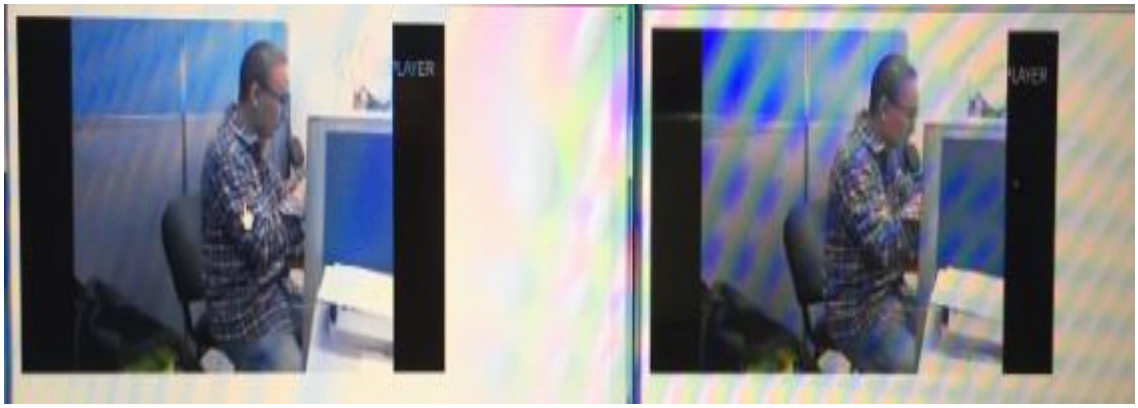

图 5-2 多用户 web 视频监控

5.3 智能设备移动客户端测试步骤

同样，首先让智能设备移动客户端存与 RTMP 服务器至相同局域网内，可以使用 WIFI 链接。RTMP 网络服务器是存于智能盒子上的，运行的是 Android 系统。RTMP 服务器的 IP 地址仍然是：192.168.1.100，此时测试客户端的地址为：192.168.1.101，并让服务器运行。

图 5-3 客户端运行界面

打开 Eclipse，并在包浏览器窗口视图中打开测试客户端的应用程序工程名 berryhome。再“Run As”“Android Application”来启动安装运行测试客户端程序。屏幕如图 5-3 所示。

点击添加按钮，准备在测试移动客户端中添加欲连接的网络服务起的 IP 地址和连接端口号信息。输入完之后通过“保存”按钮，便可将服务器信息存入后台 SQLite 中。如图 5-4 所示。

图 5-4 输入信息图

之后返回登陆列表，即服务器列表屏幕，点击连接按钮，对测试区域进行实时视频监控，如图 5-6 所示。

点击保存按钮，将自己监控到的感兴趣的画面保存到本地只能设备上。保存路径在登陆列表中可以设置，这里的路径为/mnt/sdcard/myimages 下。如图 5-6 所示。

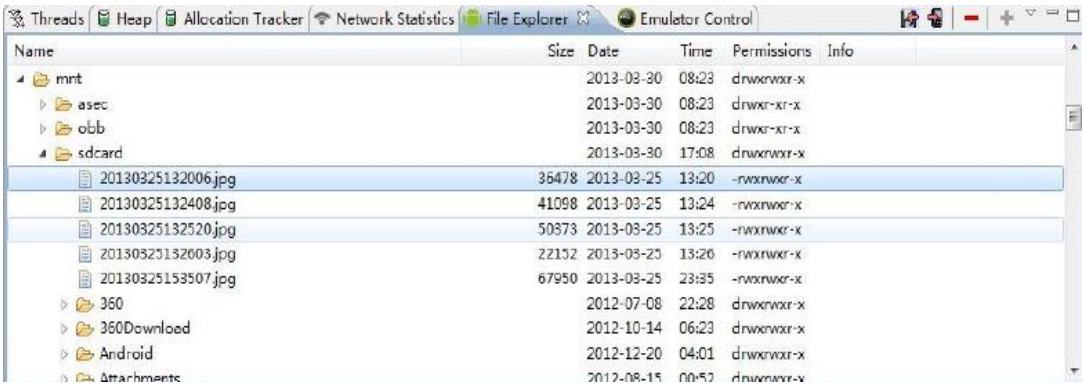

图 5-6 将图像图存本地

5.4 测试结果

上述测试系统全是搭建在同一个局域网内完成测试的，也可通过修改路由端口映射来达到外网监控的目的，将运行 Ubuntu 的 PC 机和运行 Web 测试网页的测试机共连在局域网中。另外将运行在智能盒子上（Android 系统）的 RTMP 网络服务器和带有 Android 操作系统的只能移动终端通过 Wi-Fi 网络进行无线网络连接。在 Web 终端和移动终端上查看本系统是否能传回实时的高质量的监控画面。图 5-7、图 5-8 分别表示模拟机和真机实时监控的界面图。

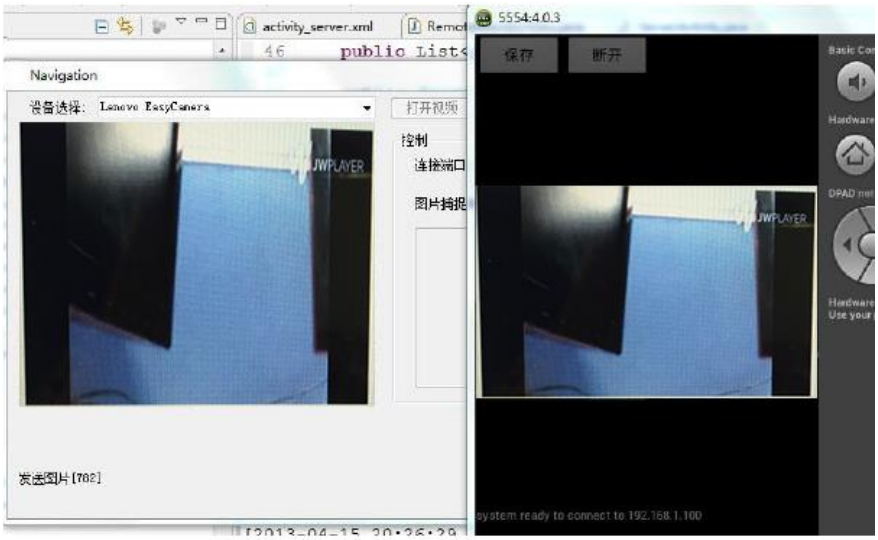

图 5-7 模拟器实时监控图

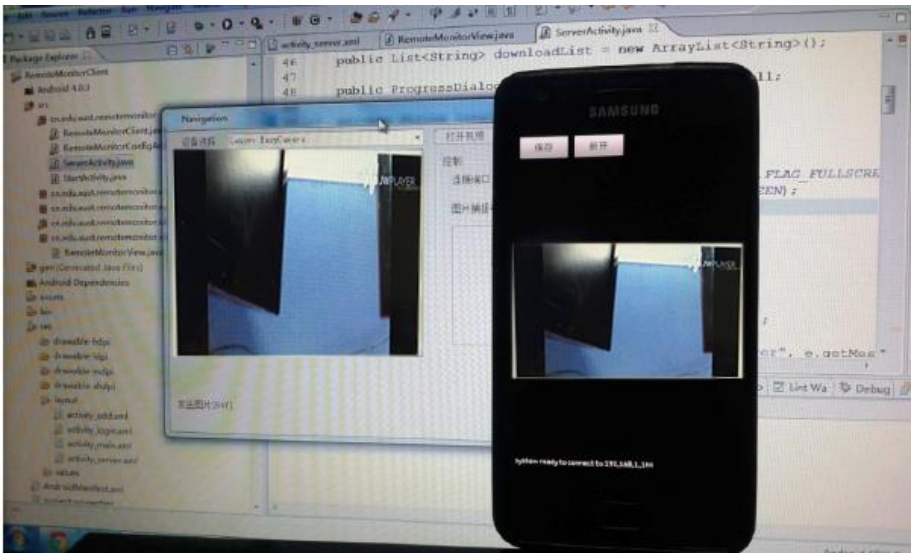

图 5-8 真机实时监控图

第六章 总结与展望

6.1 工作总结

本文先介绍了当今市场上流媒体现今的发展现状及发展趋势。人们对流媒体网络服务器的要求越来越多，如网络视频会议、视频监控等等。这就必须要面对一个问题，即如何提高网络带宽利用率。使得大量的监控视频数据能在网络上高效率的传输。这里我们选择了 FFMPEG 开源库，利用 FFMPEG 里的 H.264 视频数据格式，以 H.264 的格式对监控数据进行采集，并且它高压缩比的特性更适合于网络上的数据传输。另外，对于网络流媒体服务器，我们采用了已经相对成熟的 RTMP 协议。并且把基于 RTMP 协议的流媒体视频监控服务器分别实现在了 Linux 平台和 Android 平台上。只对以上问题，本文对 FFMPEG 视频采集压缩，RTMP 协议，Android 系统分析和流媒体服务器的移植进行了深入的研究，并做了如下的工作：

1）、对 FFMPEG 源代码进行了深入研究，详细探讨了 FFMPEG 库从数据采集到编解码的全过程，并且将 FFMPEG 移植到 Linux 系统和 Android 系统中去，对于 Android 系统，由于 FFMPEG 的移植，可以使本身自带支持少量视频格式的 Android 系统又增加了对多种视屏格式的支持。另外，对于一些存储空间有限，空间宝贵的终端来说，我们设计了将 FFMPEG 源代码裁剪编译的方案，缩短 FFMPEG 所占用的内存，节约了空间。

2）、介绍了 RTMP 协议架构，对其协议的每一个通信细节都做了研究分析，如 RTMP 控制命令的介绍、RTMP 协议通信时序、RTMP 协议 Chunk 格式、RTMP 协议消息发送与接收机制以及服务器与客户端监控通信之前所需要发送的 RTMP 协议报文。

3）、将 RTMP 流媒体视频监控服务器架构于 Linux 系统与 Android 系统，设计了详细的网络流媒体服务器的逻辑架构，并与视频采集模块密切联系，对于 Android 系统，又介绍了 java 本地调用的内容，以及如何在 Android 上层调用 RTMP 流媒体网络服务器的逻辑结构。

4）、网页视频监控客户端的设计，以网页内嵌播放器的形式，方便简单，客户端可以暂停监控，也可多点监控，对于监控过的视频数据将会定量的存于 RTMP 流媒体服务器中，客户端可以通过点播的形式对监控过的数据进行复看。

6.2 后期展望

随着网络技术在现代社会技术发展中占有越来越重要的位置，跟多信息，数据都会越来越倾向于以网络为载体进行传播，其中，流媒体网络服务器的需求必将越来越多，是更多的人们加入了流媒体开发的行列，本文针对于视频监控这一应用，在 Linux 系统和 Android 系统上实现了一个流媒体服务器系统。利用了 FFMPEG 开源库和 RTMP 协议，做了对二者的详细研究和移植工作。

在未来的生活中，数据网络化将越来越流行与人们的生活，像家庭智能，远程调控等等，而流媒体服务器正式基于将数据网络化的使得数据在网络中能安全，快速的到达目的端。这样的一些网络数据传输就要基于双方约定的网络协议，对于视频监控来说，规定在网络中双方传输视频数据的协议有 RTP、RTSP、RTMP 等等，本文选取了 RTMP 消息传输协议进行研究，在 RTMP 协议的实现上，我们根据协议内容，从握手到通信结束，从如何发送 RTMP 信息块到如何接受 RTMP 信息块，以及 RTMP 通信所发送的命令交互信息都做了详细研究。但是由于个人技术有限，网络服务器架构的设计可能有些纰漏，以及监控客户端的功能和便捷性还有一些不完善。望读者批评指正。上述不足，我均会在我日后的研究中，努力学习，加以改进。

致 谢

三年的研究生生活就快要接近尾声了，这三年的历练，让我收获良多，不仅仅收获了知识，同时也收获了关心自己的朋友，三年间，充实的每一天让我感觉不到时光的流逝，不知不觉中就走完了三年，提到这三年中认识了许多帮助过我的朋友和老师，我心中就会充满暖意和感激，因此，在这里，怀揣着感恩的心，对那些帮助过我的人们郑重的说声谢谢你们。

这三年时光里，我首先需要感谢的是我的导师鲁晓军老师，您以严谨的管理，认真的面对每一个细节，对知识永远充满着激情，这些态度都深深的感染着我，让我看清了自己的许多不足，意识到自己之前不以未然的坏毛病，在这里我郑重的感谢导师这三年对我的监督催促和悉心教导，并且真心的向导师说一声对不起，这三年给您添了很多麻烦了，虽然是这样，可你依然至始至终都对我们充满着耐心，永远让我们感受到您的关怀，并且把您的知识都毫无保留的交给我们，这些年我们一起做项目，一起探讨，一起学习，一起进步，虽然我们即将毕业，即将离开这个学校，但是我永远会记住您，记住您的处事态度，记住您对知识的执着，努力的向前走。

同时我还要感谢实验室里的同学毛小洁、郑午辰、刘智等人的帮助，是你们回答了我一遍又一遍的问题，对于我的问题，无论是简单还是复杂，你们都会耐心的帮助我，并且会在繁忙之中抽出时间给我详细分析讲解我的疑惑，有了你们的帮助才有了这个项目的完成，才有了我的进步。研究生的三年有你们，无论是在研究室的研究工作还是日常的生活工作，都充满了欢声笑语，开心的度过了每一天。这里不仅要感谢这些同学和老师，更好感谢我的父母，是你们从小对我的教育和培养，一直到我现在，每时每刻都让我感受到了你们对我的关怀。你们的鼓舞给了我极大的动力，你们的安慰是我面对困难的信心。正是因为这些人的存在，这些人对我的帮助，我会一直告诫自己不要懈怠，要不断努力，不断进步，绝不辜负你们对我的期望，最后还要感谢评审此论文的老师们，感谢你们在百忙之中评阅我的论文并对我的论文提出了您宝贵的意见。

参考文献

[1]. 李炳林. 流媒体技术及应用[J]. 电力系统自动化, 2001,25:68-71.DOI:10.3321/j .issn: 1000-1026. 2001. 24. 019.

[2]. 田志刚. 嵌入式 Linux 系统及其应用研究[J]. 网络新媒体技术, 2003, 24(5):290-295. DOI:10.3969/j.issn.2095-347X.2003.05.009.

[3]. 李凯. Android 操作系统分析与移植[D]. 华南理工大学, 2011.

[4]. 卜宪宪. 关于网络通信技术的发展的探讨[J]. 科技传播, 2012, (4).

[5]. 陈义钦, 覃团发, 陈俊江等. 基于 ARM9 的视频采集传输系统[J]. 电子工程师, 2007, 33(5):21-24. DOI:10.3969/j.issn.1674-4888.2007.05.007.

[6]. 张兴明, 耿晨歌, 汪乐宇. 网络流媒体服务器的机制与实现研究[J]. 计算机工程与设计, 2004, 25(1):61-64. DOI:10.3969/j.issn.1000-7024.2004.01.020.

[7]. 夏惊涛, 卢娜, 穆道生等. 视频监控系统浅析[J]. 兵工自动化, 2006, 25(2):84-85. DOI:10.3969/j.issn.1006-1576.2006.02.039.

[8]. 张冲, 杨灿, 杨泽铨等. RTMP 协议在 P2P 流媒体系统中的应用[J]. 电视技术, 2009, 33. DOI:10.3969/j.issn.1002-8692.2009.z2.062.

[9]. 王彤. 基于 FFmpeg 的 H.264 解码器实现[D]. 大连理工大学, 2011.

[10]. 张岩. 基于 Linux 内核及系统调用的文件系统管理的研究与实现[D]. 北京邮电大学, 2007.

[11]. 杨娇娟. 浅谈 TCP/IP 协议[J]. 数字技术与应用, 2012, (3):220-220.

[12]. 张锦盛, 朱晓晶. 使用 Android 应用框架原理对学校 WEB 系统开发平台业务层的框架升级[J]. 计算机光盘软件与应用, 2014, (8):188-189.

[13]. 安百俊, 高栋, 张伟等. 通过 Java 调用本地方法[J]. 微处理机, 2011, 32(2):41-44. DOI:10.3969/j.issn.1002-2279.2011.02.013.

[14]. 刘冬. 在 C 或 C＋＋中调用 Java 方法[J]. 开放系统世界, 2003:106-112.

[15]. 胡聪, 周甜, 唐璐丹. 基于 FFMPEG 的跨平台视频编解码研究[J]. 武汉理工大学学报, 2011, (11):139-142.

[16]. 邓中亮,段大高,崔岩松等. 基于 H.264 的视频编/解码与控制技术[M].北京: 北京邮电大学出版社, 2010,124-126.

[17]. 王旭鹏, 同济大学电子与信息工程学院, 上海 201804. 基于 Rtmp 和 Http 双协议流媒体视频点播系统[J]. 电脑知识与技术, 2011, 07(1):226-228. DOI:10.3969/j.issn.1009-3044.2011.01.088.

[18]. 张龙华. 基于 SIP/RTMP 跨平台远程教育系统设计与实现[D]. 武汉理工大学, 2012.

[19]. 辛长春, 娄小平, 吕乃光. 基于 FFmpeg 的远程视频监控系统编解码[J]. 电子技术, 2013, (1):3-5. DOI:10.3969/j.issn.1000-0755.2013.01.002.

[20]. 吴张顺, 张珣. 基于 FFmpeg 的视频编码存储研究与实现[J]. 杭州电子科技大学学报, 2006, 26(3):30-34. DOI:10.3969/j.issn.1001-9146.2006.03.008.

[21].[23] 袁学好. 网络摄像机中音频编码及 RTMP 传输技术研究[D]. 北京大学, 2013.

[22]. 胡成, 任平安, 李文莉. 基于 Android 系统的 FFmpeg 多媒体同步传输算法研究[J]. 计算机技术与发展, 2011, 21(10):85-87. DOI:10.3969/j.issn.1673-629X.2011.10.022.

[23]. 汪大磊. 基于 Android 的网络电视系统设计与实现[D]. 华南理工大学, 2012.

[24]. 李祥凯. 中文 Linux 标准之《中文 Linux 服务器系统技术要求》和《中文 Linux 用户界面规范》介绍[J]. 信息技术与标准化, 2005, (8):13-15. DOI:10.3969/j.issn.1671-539X.2005.08.004.

[25].[27] 吴增彬, 谢小鹏, 王苗苗. 基于 Android 平台的便携式修船数据管理系统的界面设计[J]. 电子设计工程, 2013, 21(3):60-63. DOI:10.3969/j.issn.1674-6236.2013.03.020.

[26]. 何圆圆, 何凯. 基于 FFmpeg 的 H.264 视频解码器的研究与实现[J]. 电脑知识与技术, 2012, (35).

[27]. 高扬. 基于 wifi 的流媒体监控系统的研究与实现[D]. 浙江工业大学, 2012.

[28]. 郝俊, 孟传良. 基于 V4L2 的 ARM11 USB 视频采集终端的设计与实现[J]. 贵州大学学报：自然科学版, 2011, 28(4):74-78. DOI:10.3969/j.issn.1000-5269.2011.04.019.

[29]. 张辉, 李新华, 刘波等. 基于 V4L2 的视频设备驱动开发与移植[J]. 电脑知识与技术, 2010, 06(15):3988-3990. DOI:10.3969/j.issn.1009-3044.2010.15.060.

[30]. Yinli L, Hongli Y, Pengpeng Z. The implementation of embedded image acquisition based on V4L2[J]. Electronics, Communications and Control (ICECC), 2011 International Conference on, 2011:549 - 552.

[31]. 杨钊. 基于 Android 的视频采集系统的设计与实现[D]. 西安电子科技大学, 2012. DOI:10.7666/d.d216622.

[32]. 陈小平, 王皖陵, 安徽工业大学计算机学院，安徽马鞍山 243002. Linux 下实时流媒体的编程实现[J]. 安徽工业大学学报：自然科学版, 2005, 22(3):293-297. DOI:10.3969/j.issn.1671-7872.2005.03.024.

[33]. 吴旭东. 高性能 Linux 网络服务器设计与实现[J]. 电脑编程技巧与维护, 2011, (20):89-90. DOI:10.3969/j.issn.1006-4052.2011.20.039.

[34]. 周又红. 基于 Linux 集群构建网络服务器的方案分析[J]. 韶关学院学报：自然科学版, 2005, 26(12):16-19. DOI:10.3969/j.issn.1007-5348.2005.12.006.

[35]. Huang J, Wu D, Liu X. Implementation of the RTMP server based on embedded system[C]. //Computer Science and Information Processing (CSIP), 2012 International Conference on. IEEE, 2012:160 - 162.

硕士学位论文

MASTER THESIS