第一章　绪论

本章主要介绍嵌入式视频监控平台设计的背景与意义、项目内容和本论文结构。首先通过对视频监控系统的历史发展与现状的分析，以及近期嵌入式的发展，指出本设计的意义和价值。第二节介绍项目已完成内容，包括已实现的模块与功能。最后一节介绍本论文的章节结构和主要工作。

1.1 项目背景与意义

在经济和科技高速发展的当代，工业民防等领域对安全监控有着越来越大的需求。安全监控是一类利用图像、声音等信号传感器，对特定目标群体进行状态实时监控和记录的系统[1]，其中有些还具有在目标状态异常时发出报警的功能。在安全监控系统中，视频监控是最常见的形式之一。视频监控利用摄像头将图像光信号转换成电信号，然后将信号集中传输，根据其类别和利用方式的不同，将其进行处理后储存，或者直接显示在监控显示器上。

随着现代技术及文明的发展，各行各业对视频监控系统的需求越来越高。无论是用于安防的楼宇监控、路口探头，还是在工业领域应用的闭路电视监控系统，对视频监控的及时性、可靠性等系统性能都有较高的要求。视频监控系统也从诞生之初的模拟信号闭路电视监控，逐步发展成现在的数字视频监控、基于网络的视频监控系统[2]。

模拟信号闭路电视监控是最早出现的视频监控系统。如图 1-1 所示，传统的模拟闭路电视监控是以摄像机、专用视频图像发射器、模拟信号传输电缆、专用视频图像接收器、监视器为核心的视频监控系统[3]。这种系统需要铺设昂贵的模拟信号专用同轴线缆，接线比较复杂，且扩展性不高，添加新的设备往往是牵一发而动全身。模拟信号有容易受干扰而发生失真的特点，因此这种系统只能在一定的范围内部署使用，难以实现远距离传输。

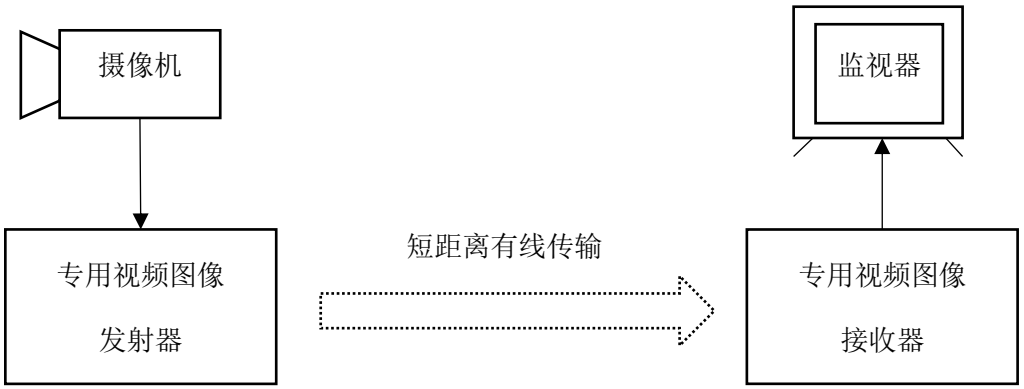

图 1-1 模拟信号闭路电视监控系统原理

数字视频监控是继模拟信号闭路电视监控之后的新技术，其结构与模拟信号闭路电视监控系统相仿，不同的是，在视频监控采集端和接收端分别加入了模拟-数字转换电路、数字-模拟转换电路，如图 1-2 所示。利用数字电路不易受干扰、线缆成本低及适合远程传输的特点，降低了系统部署成本，提升了图像传输质量，扩大了部署范围[4]。

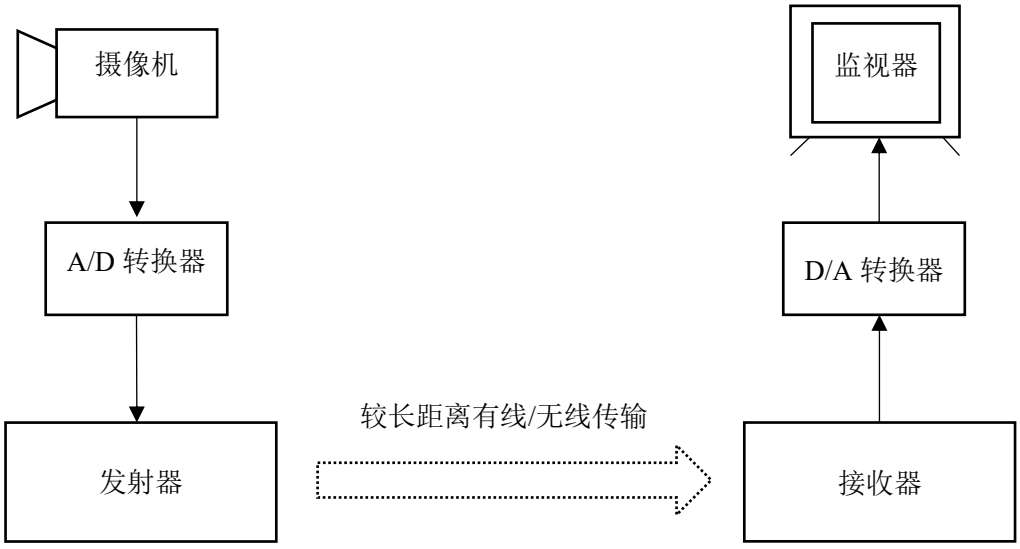

图 1-2 数字视频监控系统原理

基于网络的视频监控系统，则是在前者基础上更进一步，引入 TCP/IP 网络协议作为底层传输协议的实现，并引入了计算机技术用于处理视频，进一步增加了视频监控领域的标准化程度。如图 1-3 所示，该系统可利用现有的网络，降低系统部署成本。

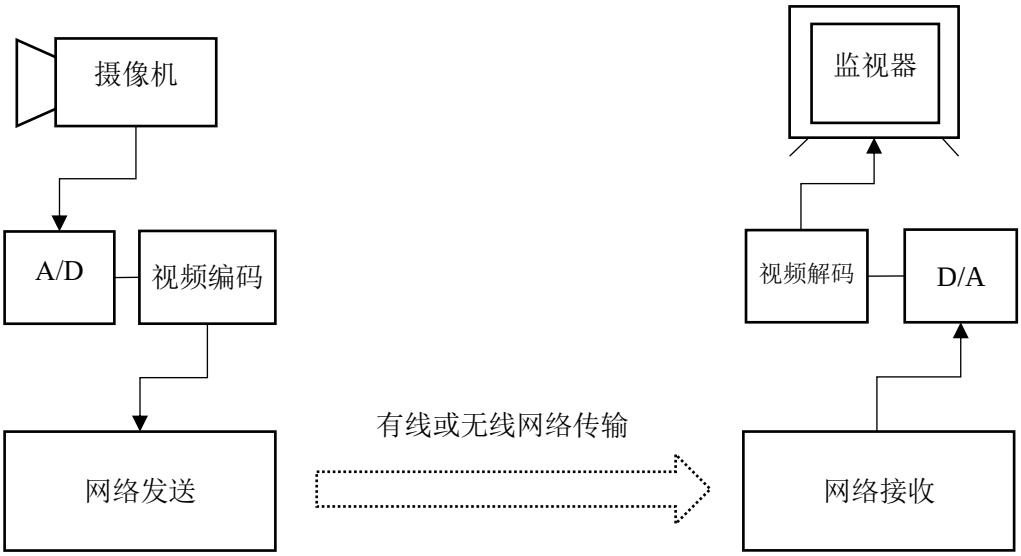

图 1-3 基于网络的视频监控系统原理

近年来，嵌入式系统取代专用芯片，越来越多地应用于小型设备上。嵌入式系统（Embedded System）以应用为中心，软硬件可以根据实际需要剪裁，适用于对功能、可靠性、成本、能耗有严格要求的专用计算机系统[5]。嵌入式系统软件功能丰富，开发成本低，有强大的运算处理能力和完善的网络软硬件支持，有不少网络视频监控设备开始采用嵌入式系统的方案。

本论文设计实现的嵌入式网络视频监控系统，运用了前文提到的嵌入式系统技术和网络视频传输技术，采用了高效的视频编码技术和传输协议，具有多功能、高性能、可扩展的优点，能够适应市场的需求，具有广泛的应用价值。

1.2 项目内容

本论文完成的视频监控平台主要运用了以下三种技术：

1. 嵌入式 Linux 系统技术。视频监控视频采集终端采用嵌入式系统环境，涉及到嵌入式 Linux 系统环境的剪裁构建、嵌入式 Linux 系统程序开发技术，实现了在小型嵌入式设备上运行视频监控程序的效果。
2. H.264 视频硬编码和实时传输协议（Real-time Transport Protocol，简称 RTP）技术。使用兼容 OpenMAX IL 标准的嵌入式芯片进行 H.264 视频硬编码，实

现实时压缩高清视频数据的功能；开发实现 RTP 协议，将视频数据采用 RTP 协议传输，保障了视频流传输的稳定性和高效性。

3. 服务器端 Web 开发技术。视频监控服务器端采用 Web 开发技术，实现了功能丰富、界面美观、易于使用的视频监控总控制台。

本文作者综合运用以上几种技术，最终实现了一个由嵌入式视频监控终端和服务器端组成的多功能、高性能的嵌入式网络视频监控平台。

1.3 本文结构

本文一共分为七个章节，各章内容安排如下：

第一章 绪论。介绍视频监控平台的研究背景和意义，简单介绍了视频监控系统的发展历史与现状，阐述了本文研究的项目内容，最后描述了本文的组织结构。

第二章 视频监控平台的总体设计。根据功能和性能需求，进行系统的总体设计，并进行嵌入式软硬件和服务器平台的选型。

第三章 嵌入式系统软件平台的构建。首先介绍了配置交叉编译环境方法，然后介绍了移植构建嵌入式 Linux 内核的步骤，最后介绍了构建 Linux 根文件系统的方法。

第四章 视频监控终端的实现。对视频监控终端进行总体设计，介绍了视频采集、H.264 硬编码及 RTP 传输模块的实现，讲述了通信接口模块与 OTA 更新模块设计的具体过程。

第五章 视频监控服务器平台软件的实现。介绍了服务器端软件的功能模块设计，阐述用户认证、终端接入、视频传输接收和系统管理等关键模块的设计思路及过程。

第六章 测试及成果展示。首先完整运行系统，测试各模块功能，其次进行了性能测试，并对测试结果进行了分析。

第七章 总结与展望。对所完成的设计进行总结评价，并展望后继研究。

本文作者的主要工作在第二章至第五章。

第二章　视频监控平台的总体设计

本章介绍视频监控平台的总体设计，首先介绍基本功能原理，通过对功能需求的分析，设计系统结构，阐述系统设计思路，并进行关键技术选型。

2.1 视频监控平台的基本原理

视频监控平台主要分为三大部分：嵌入式视频监控终端、视频监控服务器、网络基础设施，结构如图 2-1 所示。网络基础设施在大多数情况下无需专门建设，直接采用 100Mbps 或以上的快速以太网（Fast Ethernet Intranet）。嵌入式视频监控终端一般用摄像头采集图像，经过专用芯片处理为数字信号，再通过视频压缩算法编码，最后将压缩好的数据通过网络传输到服务器端。视频监控服务器负责接收视频数据进行处理、储存，同时向使用者提供管理界面，以供用户查看监控视频。

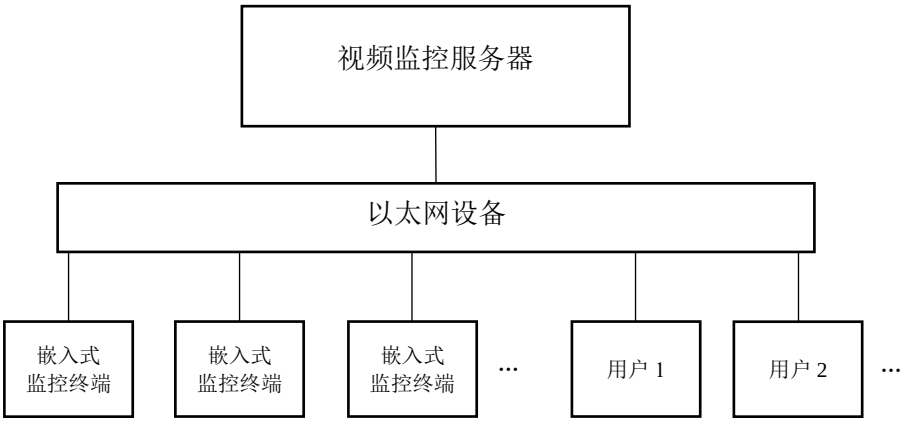

图 2-1 视频监控平台的结构示意图

2.2 视频监控平台的功能设计

视频监控平台由多种模块有机结合，具有如下功能：

1. 视频采集与视频存储功能是多路视频监控平台的基础功能。嵌入式监控端采集视频并传输到服务器，服务端将视频进行转码保存。
2. 视频监控服务端的所有管理功能如图 2-2 所示，主要分为以下四个部分：
   (1) 用户登录认证功能。为保证系统安全，防止未授权用户操作，提供用户登录认证功能，只有预先授权的用户才能访问系统。

(2) 监控视频查看功能。提供实时视频查看和历史视频查看功能，允许用户通过用户界面查看所有终端传来的实时画面预览图；支持用户点选特定终端，查看详细监控画面，调阅历史监控视频。

(3) 系统管理功能。为管理者提供服务端系统信息查看、用户管理、系统更新等管理功能。

(4) 设备远程管理功能。包括终端设备系统信息查看与管理、远程重启设备、远程 OTA 更新功能。可以查看到终端设备的地址、系统负载、开启时间和温度等各项系统参数，并能够对设备进行远程管理操作。

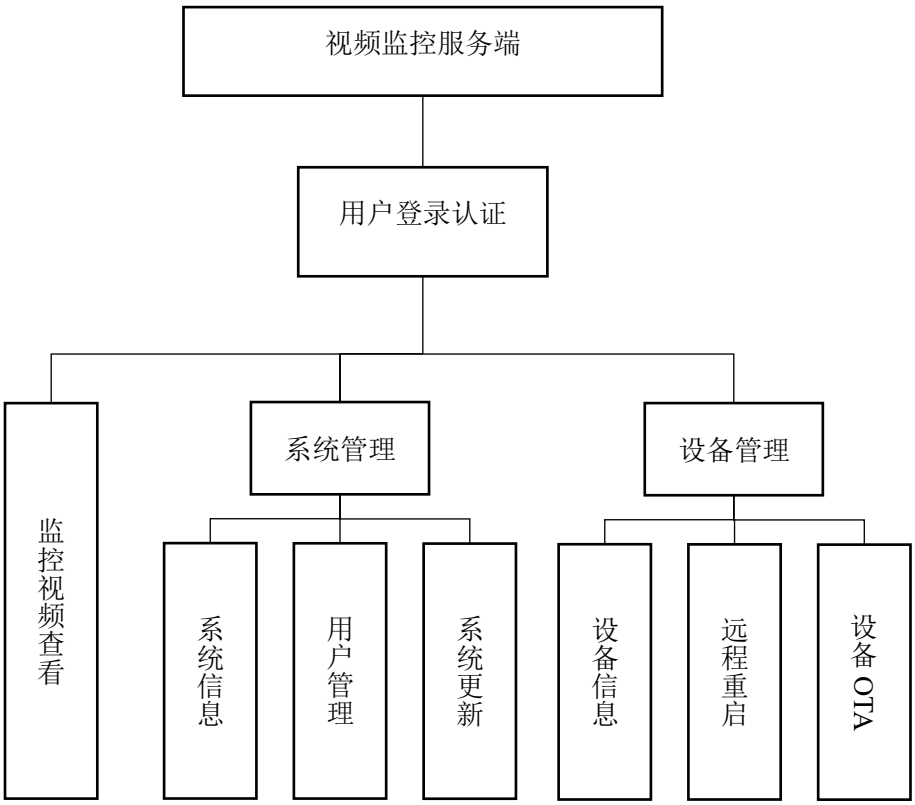

图 2-2 视频监控服务端的功能示意图

总之，视频采集终端主要负责视频的采集、编码处理和传输；服务器总控制端接收并存储视频，为管理者提供实时及历史视频查看、远程设备管理、系统管理等功能。

2.3 系统的设计思路

系统设计应尽量满足结构稳定、模块化、可扩展、可复用的原则。对于系统中最为常用的部分，将其作为基本模块编写，以增加代码模块复用度。对于系统中将来可能变

动的功能，将其设计为扩展模块，通过模块间调用动态执行。

嵌入式视频监控终端按功能划分，可分为三个主要模块：通信接口模块、视频处理模块、固件更新模块，如图 2-3 所示。各模块通过 Linux 进程方式进行隔离，由通信接口模块所在进程提供事件循环，控制其他模块进程的启动与结束。

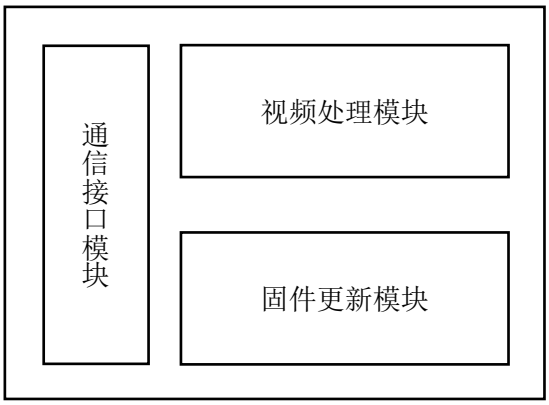

图 2-3 嵌入式视频监控终端的模块结构示意图

视频监控服务器按照功能划分，可粗分为通信接口模块、升级管理模块、Web 管理服务模块、视频接收模块，如图 2-4 所示。服务端由 Web 模块对外提供服务，并控制其他模块的运行状态。

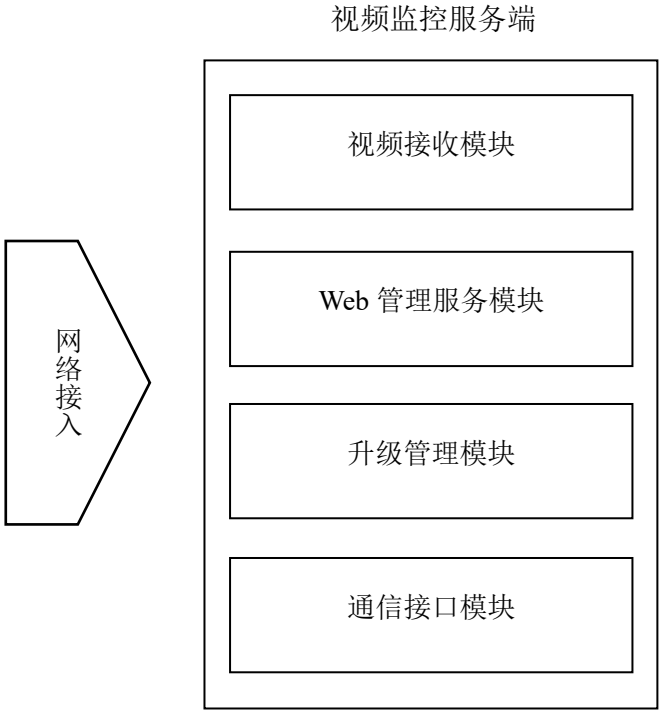

图 2-4 视频监控终端的模块结构示意图

2.4 技术选型

本设计需要用到多项技术，这些技术的选型，关乎到以后的功能实现和系统的性能。其中主要有嵌入式硬件开发板选型、嵌入式系统软件选型、视频压缩算法与传输协议选型、服务器平台技术选型。

2.4.1 嵌入式硬件开发板选型

嵌入式硬件开发板的选择取决于对嵌入式微处理器与外围电路（设备）的选择，嵌入式微处理器是嵌入式系统的核心，决定了嵌入式系统的性能优劣。目前主要使用的嵌入式处理器架构有 MIPS、ARM、x86 等[6]，它们各自具有以下特点：

- ARM 是高级精简指令集（Advanced RISC Machine）的简称，ARM 处理器方案具有很多优点，如体积小、低功耗、低成本、高性能，因此很多嵌入式设备都采用了该方案。
- MIPS 也是一种较为流行的嵌入式处理器方案，同样具有低成本、低功耗和体积小的特点，但其架构设计导致内存和缓存的容量有限，在任务量大且比较复杂时，相比 ARM 性能要差一些。
- X86 性能较高，但其采用复杂指令集（CISC），设计上导致功耗大，导致电路热设计时不得不让出一定的体积用于散热，造成系统体积偏大。相比 ARM 和 MIPS 的成本也较高。

本设计的视频采集处理模块对性能有较高的要求，同时功耗和体积又要控制在一定范围内，以满足室外工作的条件。因此，初步确定选择基于 ARM 的嵌入式处理器作为系统的微处理器。

基于 ARM 的开发板硬件选择非常多，“树莓派”（Raspberry Pi）是比较有名的嵌入式 Linux 开发板方案。Raspberry Pi Model B 的硬件规格如表 2-1 所示，其芯片基于 BCM2835 SoC 方案，具有高达 700MHz 的运行频率和 512MB 的 RAM，板上 IO 接口和硬件资源丰富，很适合用于嵌入式视频监控终端的开发。

表 2-1 Raspberry Pi Model B 的硬件规格[7]

Hardware	Spec
SoC	Broadcom BCM2835
CPU	ARM11 (ARMv6) with floating point, 700MHz
GPU	Broadcom VideoCore IV
RAM	512MB
Networking	10/100 wired Ethernet RJ45
Storage	SDIO card slot
GPIO	26-pin header, populated
Ports	HDMI, Composite Video, 3.5mm analogue audio-video jack, 2 x USB 2.0, Ethernet, Camera Serial Interface (CSI), Display Serial Interface (DSI)

该开发板采用的 BCM2835 处理器中内置了 H.264 硬件编解码的 GPU[8]，这意味着可以采用硬件 H.264 编码器进行视频处理，能够极大地提升系统性能，免去软件编码造成的额外系统开销。该开发板还具有一个快速以太网接口（10/100 Mbps Ethernet），能够满足本设计中网络传输的需要。

2.4.2 嵌入式操作系统选型

基于上一节选择的 Raspberry Pi Model B（BCM2835 SoC）开发板，可供选择的嵌入式系统环境主要有以下两种：

1. 直接进行板上 ARM 汇编，该方案要同时开发上层应用和底层汇编，虽然能取得很高的运行效率，但开发难度太高，工作量将非常巨大，项目代码难以维护。

2. 使用嵌入式 Linux 操作系统。嵌入式 Linux 操作系统在嵌入式领域应用十分广泛，开发比较容易。它的硬件兼容性好，具有系统接口标准统一的优点，程序可移植，且与大部分 UNIX 标准和 POSIX 标准兼容[9]。Linux 是开源软件，工业界和学术界的研究都非常多，几乎所有嵌入式硬件厂商都有专门为 Linux 内核编写驱动程序的团队。

通过以上的比较，嵌入式 Linux 操作系统更适合本项目。开放的源码、丰富的软件资源、良好的硬件兼容性、较高的可用性，无疑非常有利于我们的开发工作。

2.4.4 服务器平台技术选型

现代 Web 技术发展迅速，可供选择的服务器平台技术越来越多。Java 作为一门服务器语言，运行效率高，在多核多线程模式下能够充分利用 CPU、内存等资源。基于 Java 编写的 Spring 是一种流行的服务器平台技术，得益于 Java“一次编译，到处运行”的多平台兼容特性，可运行于包括 Linux、Windows、UNIX、BSD 等多种操作系统平台和处理器架构。Spring 框架遵循模块化的设计原则，其模块结构如图 2-5 所示。它先进的控制反转（Inversion of Control container，简称 IoC）与面向切片（Aspect Oriented Programming，简称 AOP）的设计思想，使软件开发过程中可以保持很好的工程性，有效降低模块耦合，提升工程质量[10]。

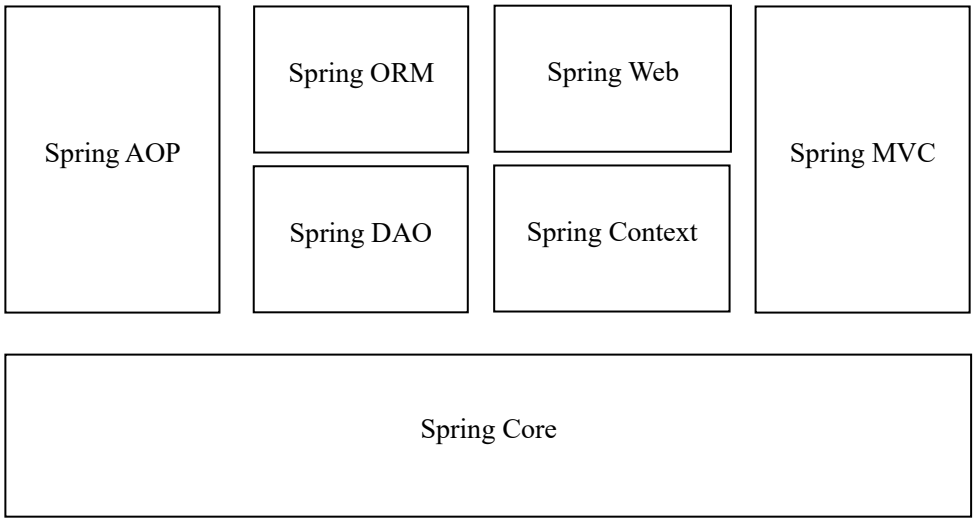

图 2-5 Spring Framework 框架结构示意图

常用的服务器操作系统有 Linux，Windows 和 BSD。其中，Windows 系统是 Microsoft 公司开发的操作系统，使用它需要先向 Microsoft 公司支付高昂的授权费用。BSD 系统是免费分发的，但其市场占有率相对较低，国内外的开发资料较少，使用起来颇为困难。而 Linux 系统具有免费、开源、文档资料多等优点，其开发、维护的成本低，最为适合作为视频监控的服务器平台。

2.5 本章小结

本章完成了系统的功能分析与总体设计。在这一章中，首先分析了监控平台的工作

原理，分析了监控平台应具有的各项功能，进行了系统的总体设计。并对系统中关键技术进行了分析和选型。我们选择了 Raspberry Pi Model B 开发板作为嵌入式硬件，Linux 系统作为嵌入式操作系统，选取了 Java、Spring、Linux 三者结合的服务器平台方案。嵌入式终端和服务器端分别采取了模块化设计。在后续章节中，将进一步研究各个部分的设计与实现。

第三章　嵌入式系统软件平台的构建

本章主要讲述嵌入式系统软件平台的构建，首先介绍交叉编译的概念，进行交叉编译环境的准备，然后介绍嵌入式 Linux 系统内核的裁剪构建，最后介绍根文件系统的。

3.1 交叉编译环境的准备

交叉编译是指在一种平台上讲源程序编译为另一种平台上可执行的目标代码[11]。我们的目标平台是嵌入式系统，虽然嵌入式处理器的性能足够运行我们的应用程序，但毕竟资源有限，并不能良好地完成编译任务，在嵌入式处理器上编译是不现实的。因此，编译任务应当交由性能更加强大的 PC 主机。

在交叉编译进行前，先要配置好交叉编译环境。交叉编译环境包括一个交叉编译器，和一组适用于目标平台的交叉编译配置参数，这两者都要针对目标平台进行筛选，以确保编译后代码能够在目标平台上运行，并且能够高效地运用目标平台的硬件资源。

本论文使用 Ubuntu Linux 16.04 作为 PC 宿主操作系统，针对 Raspberry Pi Model B（BCM2835）的交叉编译环境安装步骤如下：

1. 安装 Git 版本控制系统，以便后续克隆内核和工具链代码仓库。在 Linux 终端中执行安装命令。

   ```
   sudo apt-get install -y git
   ```

2. 下载交叉编译所需的工具链。

   ```
   git clone https://github.com/raspberrypi/tools.git
   ```

3. 将交叉编译器所在目录加入宿主机 Linux 用户的 PATH 环境变量中。

   ```
   echo 'export PATH=$PATH:/home/user/tools/gcc-linaro-arm-linux-gnueabihf-raspbian-x64/bin' >> ~/.profile
   ```

4. 执行 ls 命令，查看 bin 目录下的交叉编译套件。

   ```
   ar  c++  gcc      ld     ld.gold  objcopy  ranlib
   as  g++  gfortran  ld.bfd  nm      objdump  strip
   ```

上述步骤在本地安装了 linaro gcc/g++交叉编译器，以及 ld 链接器等交叉编译所需环境。至此，本节完成了对交叉编译环境的配置。

3.2 嵌入式 Linux 系统内核的构建

嵌入式 Linux 内核是整个嵌入式系统的重要组成部分，它提供了底层硬件统一接口，为应用提供了基础的运行环境。在嵌入式视频监控终端的设计中，因为硬件资源有限，我们并不需要一个包括所有硬件驱动的 Linux 系统内核，我们只需要使用关乎视频采集处理和网络传输的部分，所以我们需要自行构建一个精简的 Linux 内核。Linux 内核构建工作的重点在于内核裁剪，即才减掉不需要的内核模块，包括一些硬件设备驱动，未用到的文件系统模块等，以缩小系统映像体积，防止多余的系统资源消耗。本次经裁剪构建的内核体积仅为 3MB。

构建所需 Linux 内核的具体步骤如下：

1. 为宿主机安装必要的依赖项。

```
sudo apt-get install -y libncurses5-dev
```

2. 获得适用于 Raspberry Pi 的 Linux 内核源码。

```
git clone -depth=1 https://github.com/raspberrypi/linux.git
```

3. 进入源码目录，初始化内核配置。

```
cd linux
KERNEL=kernel
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf-
bcmrpi_def config
```

以上命令为 make 传入 ARCH 变量，指定了编译的目标环境，CROSS_COMPILE 变量指定了第一节中安装的交叉编译程序，在编译时会使用指定的交叉编译器和工具，如 arm-linux-gnueabihf-gcc 和 arm-linux=gnueabihf-ld 等。

4. 进入内核选项配置界面。

```
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- menuconfig
```

5. 在内核选项配置界面进行如下操作。

(1) “General setup”菜单中，取消“Support for paging of anonymous memory (swap)”项的选择，本设计的嵌入式终端没有对 swap 暂存空间的需要。

(2) “Device Drivers”，“Network device support”，“Ethernet driver support”中，仅选中“SMC (SMSC)/Western Digital devices”，其他项目取消勾选；“USB

Network Adapters”中仅选中“SMSC LAN95XX based USB 2.0 10/100 ethernet devices”；取消其他网络设备。

(3) 同上，针对其他设备作适当配置，保留 Video4Linux 和串口、GPIO 驱动，其余无用的驱动均取消勾选。

配置完成后，点选 ＜Save＞ 保存退出。

6. 编译内核镜像、模块和设备树文件。

```
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- zImage modules dtbs -j 4
```

Linux 内核编译完成后，将内核镜像（zImage）、模块（modules）和设备树（Device Tree Block，简称 dtb）分别安装到 SD 卡中对应的位置，完成本次 Linux 系统内核的构建。

3.3 嵌入式 Linux 根文件系统 rootfs 的构建

完整的嵌入式 Linux 操作系统不仅包含一个可用内核，还需要一个有效的根文件系统。根文件系统是包含根目录的文件系统，结构如图 3-1 所示，其中包含了 init 程序、一些基本的可执行组件和命令（例如 shell）、*.so 运行库、系统配置等文件，是支撑操作系统运行的基础。

嵌入式 Linux 通常不会使用体积庞大的 Bash shell 以及 grep，sed 等 GNU/Linux 程序，而是使用经过精简设计的 Busybox 可执行程序。Busybox 程序仅 1MB 左右，“麻雀虽小，五脏俱全”，其包含了 shell、grep、sed、awk、gunzip 等几十种命令的绝大多数功能，通过软链接的形式生成各系统命令，因此为构建嵌入式 Linux 系统的首选。

对于嵌入式应用，通用 Linux 系统使用的 Glibc 运行库体积庞大，功能臃肿，大多数功能是嵌入式应用不会涉及到的。因此我们一般不会选择在嵌入式系统中使用 Glibc，而是采用 uClibc 运行库，这是一个为嵌入式 Linux 专门设计的 libc 库，模拟了 Glibc 大多数功能，同时体积上进行了精简和优化，文件大小甚至不到 Glibc 的 1/20。

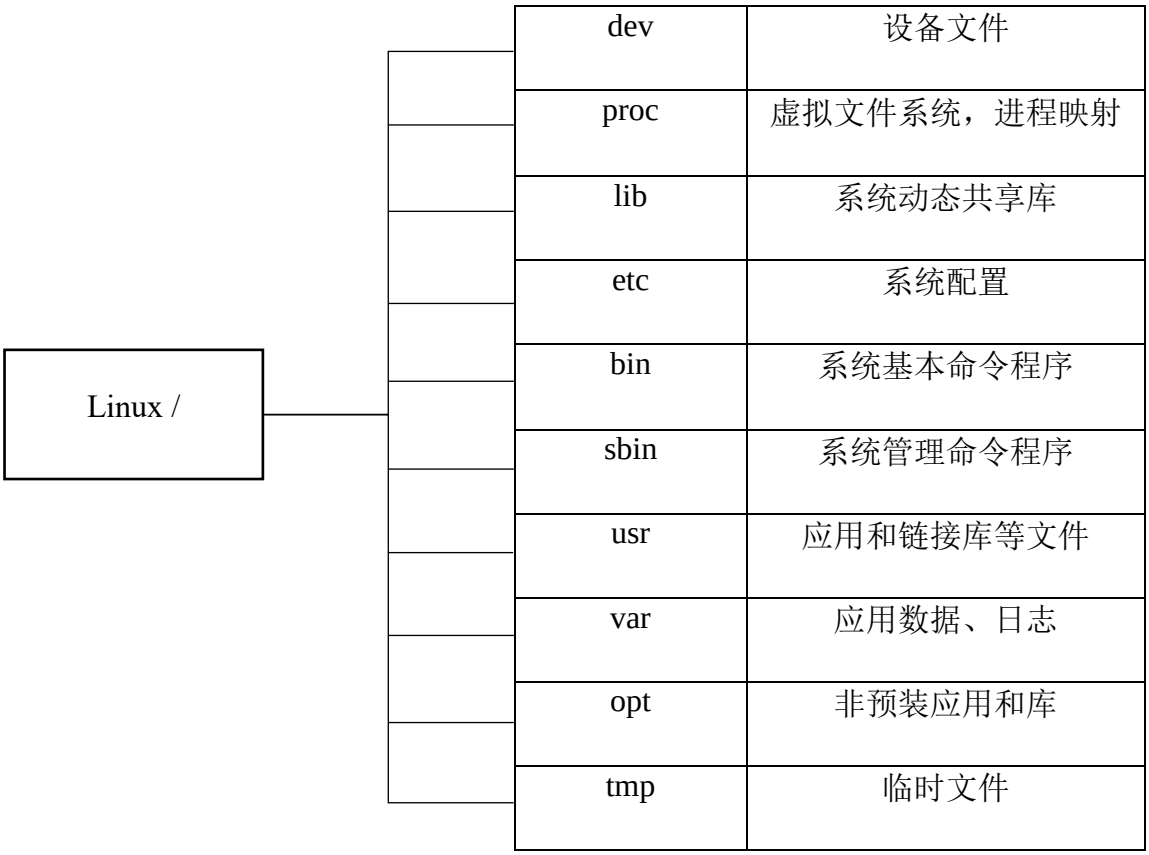

图 3-1 Linux 根目录结构

使用 Buildroot 组件（https://buildroot.org/）可生成嵌入式 Linux 所需的根文件系统。由于我们在上一步已经自行构建了内核，现只需要用 Buildroot 构建 Busybox 和 uClibc。完成生成 rootfs 镜像后，挂载镜像，通过 cp –ar 命令拷入所需要的内核模块和 BCM2835 硬编码所需的 bcm_host、openmaxil 等运行库，最后将镜像安装到 SD 卡镜像的主分区，即完成了 rootfs 的构建。

3.4 本章小结

本章主要介绍了嵌入式 Linux 操作系统的构建方法。首先建立了交叉编译环境，然后构建系统内核和根文件系统。完成嵌入式 Linux 系统的搭建后，即可继续进行视频监控终端应用程序的开发和部署调试工作。

第四章　视频监控终端的实现

本章介绍视频监控终端的实现，主要包括视频监控终端的总体框架，以及终端各模块的实现。

4.1 视频监控终端的总体框架

本设计的视频监控平台使用 C/S 架构。监控服务端（Server）负责维护整个系统的运行，控制设备接入。而视频监控终端作为客户端（Client），负责主动连接到服务端，将采集到的视频编码压缩处理，并发送到服务端。同时，监控终端与服务端保持 TCP 长连接，向服务器发送设备状态，接受并执行来自服务端的控制指令。

视频监控终端的模块大致划分如图 4-1，主要分为管理通信接口模块、视频处理模块、固件更新模块。

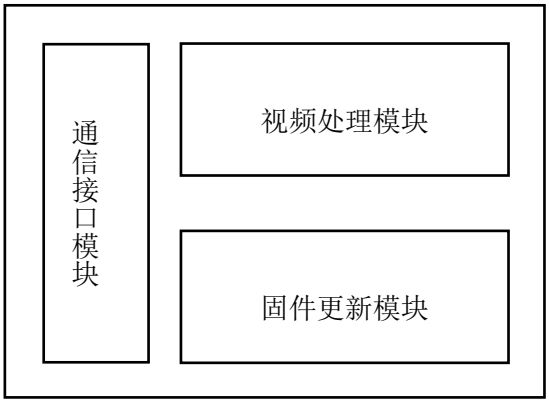

图 4-1 嵌入式视频监控终端的模块结构示意图

4.2 视频处理模块的实现

视频处理模块负责从摄像头采集视频，将采集到的原始数据进行编码压缩处理，最后使用 RTP 协议通过网络传输。

4.2.1 视频采集、压缩模块的实现

视频采集是通过 Linux 的 Video4Linux 接口实现的。Video for Linux（简称为 V4L），是 Linux 系统中用于支持摄像头视频捕获的标准 API。本论文采用兼容 V4L 标准的 USB

摄像头，系统中对应的设备文件为/dev/video0，video 设备属于无缓冲字符设备，可通过该设备文件进行视频采集。

从 V4L 中采集视频要依次调用各个接口函数，来打开设备文件描述符，初始化设备参数，然后开始捕获画面，采集视频。关键步骤如下：

1. 打开视频设备文件描述符（file descriptor），调用 `open("/dev/video0", O_RDWR | O_NONBLOCK, 0)` 函数。
2. 初始化设备，调用 `xioctl(fd, VIDIOC_QUERYCAP, &cap)` 获取摄像头设备信息，如设备标题、可用图像大小。
3. 设置 V4L2 的 Buffer 类型为视频捕获（V4L2_BUF_TYPE_VIDEO_CAPTURE）模式；设置 Buffer 数据获取方式为 mmap（V4L2_MEMORY_MMAP）。
4. 使用 mmap 捕获、采集视频。
5. 对视频进行编码压缩处理，此处调用了 BCM2835 的 OpenMAX IL 库和 Broadcom 公司提供的 bcm_host 库，bcm_host 库的头文件路径是 /opt/vc/include/bcm_host.h，OpenMAX IL Cilent 则需要从官方网站下载。

定义数据结构 enc_handle 用于编码处理：

```
COMPONENT_T *video_encode;       // video encode 组件
COMPONENT_T *list[5];            // tunneled 相连的组件列表
ILCLIENT_T *client;              // omxil 调用者对象
OMX_BUFFERHEADERTYPE *out;       // 编码输出的 buffer
void *combination_buffer;        // buffer 合并结果
unsigned long frame_counter;     // 当前帧计数
struct enc_param params;         // 编码参数，包含 fps bitrate 等
```

通过上述方式调用 OpenMAX IL API，实现对采集到的视频帧进行编码压缩，得到压缩后的视频数据。

4.2.2 视频传输模块的实现

视频传输模块负责将上一步骤中编码压缩的视频流稳定、高效地通过网络传输到服务端。受制于链路干扰，网络延迟和丢包始终是存在的，因此我们不能简单的将视频流直接发送出去，而是要在应用层引入实时传输协议（Real-time Transport Protocol，简

称 RTP）和实时传输控制协议（Real-time Transport Control Protocol，简称 RTCP）来保证传输的可靠性。

1. RTP 协议数据包由 RTP 头和 RTP 负载组成，头部格式如图 4-2 所示。

<table>
<tr><td>V=2</td><td>P</td><td>X</td><td>CC</td><td>M</td><td>PT</td><td>SN</td></tr>
<tr><td colspan="7">Timestamp</td></tr>
<tr><td colspan="7">SSRC identifier</td></tr>
<tr><td colspan="7">CSRC identifier
…</td></tr>
</table>

图 4-2 RTP 协议的头部格式

RTP 协议头各组成部分含义如下：

- V：版本号，2 位，标识 RTP 协议版本。
- P：填充位，1 位，置位的 RTP 包追加尾部填充字节。
- X：扩展位，1 位，置位的 RTP 头部包含扩展头部。
- CC：CSRC 计数器，4 位，指定 CSRC 列表所含项目的数目。
- M：标记位，1 位。
- PT：载荷类型，7 位，标识 RTP 载荷的类型。
- SN：序列号，16 位，发送者每个 RTP 包都将 SN 增加 1，接收方可以据此检测丢包和进行包序列恢复。
- Timestamp：时间戳，32 位，记录包的采样时刻。
- SSRC identifier：同步源标识符，32 位，随机选取的唯一值，RFC1889 建议使用 MD5 随机算法。
- CSRC identifier：贡献源列表，0~15 个，每个 32 位，标识由 muxer（合成器）合成的不同的源。

在视频监控平台中，目前只实现了单一视频流的传输，即 muxer 源是唯一的。RTP 发送端在每次发送数据时，加入 SN、Timestamp 标记，接收端根据这两个标记，对包进行标记与重排序，从而保证了接收顺序与发送顺序一致。

2. 实时传输控制协议（RTCP）

RTP 协议只负责数据包的传输，需要使用 RTCP 协议提供服务质量保证（QoS）。RTCP 的各传输端周期性地统计 RTP 发送数据包数量和数据包丢失数量等信息，使对

方可以针对性地动态调整传输速率与传输策略。

RFC3550 标准中规定，RTCP 有五种分组类型，本项目主要使用了 1 和 2 类型，定义如下：

1. 发送端报告分组（Sender Report，SR），向接收端报告当前发送状况，包括 RTP 流的 SSRC、时间戳和字节数等统计信息。
2. 接收端报告分组（Receiver Report，RR），向接收端报告当前接收的最新 RTP 流状况，包括 SSRC、时间戳、时钟时间、分组数和字节数。
3. 源点描述（Source Description Items）。
4. 终止传输（BYE）。
5. 特定应用（APP）。

采用 RTP 与 RTCP 协议，能够在一定程度上降低网络问题对视频的传输质量造成的不利影响，增加了视频监控平台的稳定性。

4.3 通信接口模块的实现

通信接口模块与服务端直接进行通信，主要负责接收来自服务端的控制指令，并周期性发送设备运行状态信息到服务端。本设计的通信接口模块采用 TCP 传输协议，由嵌入式监控终端发起连接到服务端，通过定义数据包的协议，实现通信功能。

本设计定义的数据包格式如下：

1. 监控终端向服务端发送设备上线请求：

P=0x49	V	T=1	SN
SN		(empty)	

图 4-3 设备上线请求协议格式

P：协议标头，8 位，固定值 0x49。

V：协议版本号，8 位，当前为 1。

T：请求包类型，4 位，上线请求为 1。

SN：设备序列号，32 位，全局唯一。

2. 监控终端向服务端周期性发送设备参数、遥测数据：

P=0x49	V	T=2	SN
SN		Payload	

图 4-4 设备参数数据上报协议格式

P、V、T、SN：同上。

Payload 由以下结构组成：

```
struct {
   char id[16];                    // 设备标识名
   char software_version[32];      // 软件版本号
   char hardware_version[32];      // 硬件版本号
   char ip[32];                    // 当前设置的 IP 地址
   char mac[20];                   // 设备 MAC 地址
   uint_8 temperature;             // 设备温度
   char uptime[32];                // 设备运行时长
   float load[3];                  // 系统平均负载
}
```

3. 监控终端向服务器端发送命令确认响应：

P=0x49	V	T=3	SN	
SN		ACK_T	RES	

图 4-5 命令确认响应协议格式

ACK_T：响应确认对应的服务端配置类型（T）。

RES：命令响应（执行）结果。0 为已收到，1 为执行成功，2 为执行失败。

4. 服务端向监控终端发送设备准许 RTP 传输响应：

P=0x50	V	T=1	(empty)

图 4-6 设备准许 RTP 传输协议格式

P：协议标头，8 位，固定值 0x50。

V：协议版本号，8 位，当前为 1。

T：请求包类型，4 位，准许传输响应为 1。

5. 服务端向监控终端发送远程重启指令：

P=0x50	V	T=2	Payload

图 4-7 远程重启指令协议格式

6. 服务端向监控终端发送固件更新通知：

P=0x50	V	T=3	Payload

图 4-8 固件更新协议格式

Payload 由以下结构组：

```
struct {
   char software_version[32]; // 新版本号
}
```

4.4 固件 OTA 更新模块的实现

嵌入式终端上运行的固件不是一成不变的，在将来有可能发现软件存在 Bug 或者安全漏洞。将所有设备召回是不现实的，一般较为高端的嵌入式设备，都支持固件空中更新（Over-the-Air，简称 OTA）功能。

本论文设计了一种较为轻便的固件 OTA 更新机制，流程如图 4-9 所示。存在 OTA 更新时，服务端向终端发送通知，终端固件 OTA 更新模块被调用，通过 HTTP 协议从服务端下载新版本固件，检查文件 SHA1 校验值是否正确。完成检查后，解压固件压缩包，执行其中的 patch.sh。该文件是一个提前编写好的，能够将新版固件写入到设备的 Linux Shell 脚本，实现如下：

代码 1　OTA 更新脚本（文件名：patsh.sh）

```
01  #!/bin/sh –xe
02  # videomon-endpoint 0.1.2 OTA patch
```

```
03  # 覆盖 vmond
04  cp –f vmond-0.1.2 /usr/bin/vmond && chmod +x /usr/bin/vmond
```

顺利运行完毕后，OTA 更新模块触发系统重新启动（reboot），重启后系统运行的是新版本固件，从而完成整个 OTA 更新的流程。

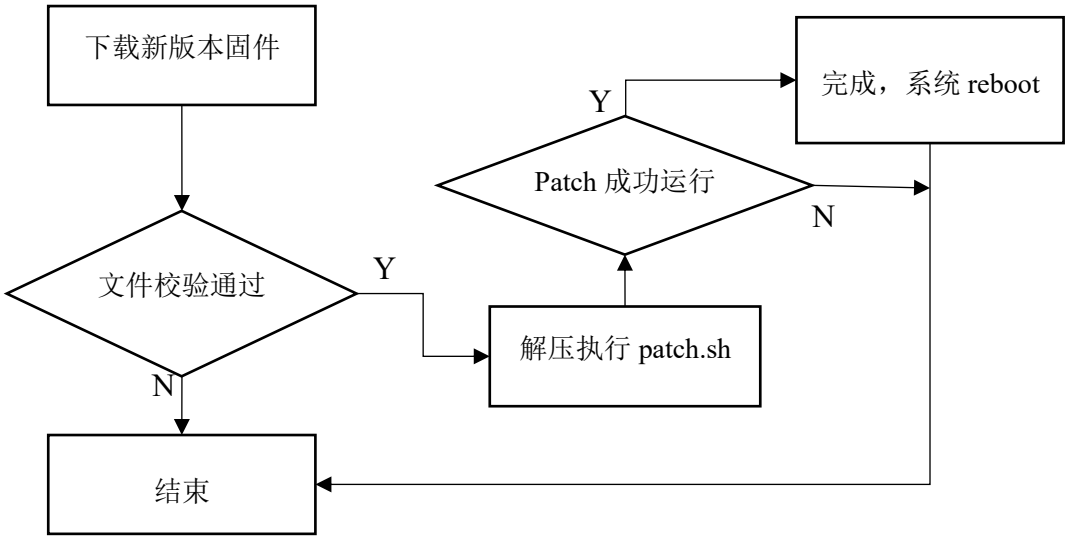

图 4-9 固件 OTA 更新模块工作流程

4.5 本章小结

本章在对嵌入式视频监控终端的功能进行分析的基础上，完成了监控终端的结构划分和各个功能模块的设计，完成了基于 V4L 的视频采集、基于 OpenMAX IL 的 H.264 视频硬编码、基于 RTP 与 RTCP 的视频传输协议、监控终端的通信模块以及固件 OTA 更新功能的实现。

第五章　视频监控服务器平台软件的实现

本章介绍视频监控服务器平台软件的实现，首先进行总体框架设计，然后进行各模块的实现工作，最后进行服务器 HTTP 软件的安装配置。

5.1 视频监控服务器平台的总体框架

根据之前的功能需求分析，视频监控服务器平台应具有视频采集与视频存储功能、基于 Web 的用户登录功能、视频查看功能、系统管理功能、设备管理功能。这些功能的联系，划分为视频接收处理模块、Web 管理服务模块、升级管理模块、通信接口模块。根据前期选型，Web 服务框架采用 Spring Framework，服务器使用 Linux。系统模块结构示意图如图 5-1 所示。

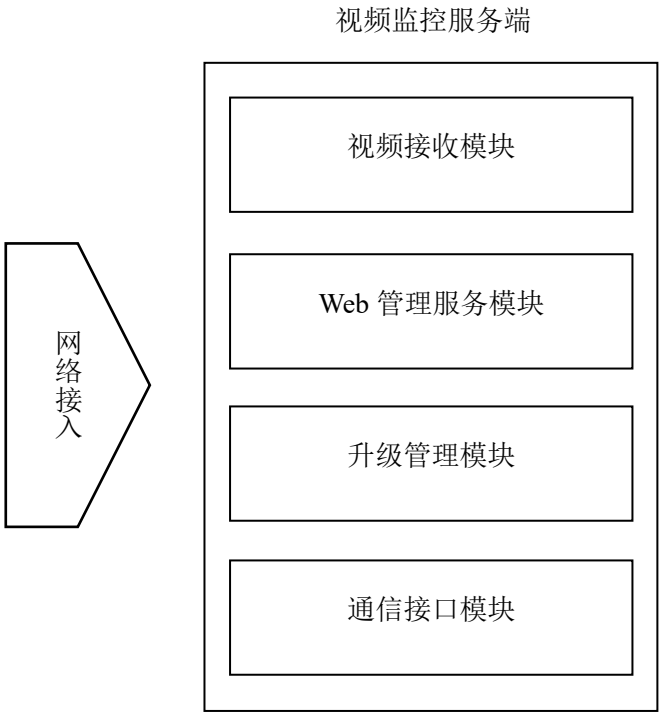

图 5-1 视频监控服务端的模块结构示意图

其中，通信接口模块接受来自监控终端的连接，接收处理设备发送过来的设备运行状态参数，向终端发送控制指令；升级管理模块从互联网服务器检查最新版本的服务端和监控终端版本，并执行版本升级功能；Web 管理服务模块提供管理用户进行所有操

作的入口，提供用户认证鉴权、视频查看、设备管理和系统管理的操作接口；视频接收模块负责接收 RTP 和 RTCP 协议的数据包，进行视频转码和储存处理。

5.2 视频监控服务器平台的实现

5.2.1 Web 管理服务模块的实现

Web 管理服务模块使用 Java / Spring 框架与 MySQL 数据库系统，要实现用户认证、系统管理等功能和接口。

对于用户认证鉴权功能，需要系统存储用户凭据信息。用户表的设计如下：

代码 2 用户表结构（文件名：User.kt）

```
@Id
@GeneratedValue(strategy = GenerationType.AUTO)
val id: Long = 0,                    // 用户 ID，全局唯一

@NotBlank
@Column(nullable = false, unique = true)
var username: String = "",    // 用户登录名，全局唯一

@Column(nullable = false)
var password: String = "",    // 用户密码的 bcrypt 加密值
```

在用户表中，系统内部使用了长整型数字 id 作为用户的唯一标识符，username 作为用户登录的用户名。用户密码经过不可逆的 bcrypt 算法加密，然后保存到数据库，保证了系统安全性，不会出现明文密码受攻击泄露的情况。

在用户发起登录请求时，Web 管理服务模块调用 Spring Security 实现判断系统中是否存在对应用户，判断用户密码输入是否正确。登录成功后，将用户跳转到视频监控总控制台；登录失败时，给出登录失败的提示。

系统管理的用户管理功能，即对系统用户进行查看、增加、删除、修改的接口，主要使用 Hibernate 的数据访问对象（Data Access Object，简称 DAO）实现。通过对数据库中的 User 表数据进行增删改，能够完整实现上述功能。

视频查看功能使用 HTTP 视频实时流（HLS）协议实现。允许已登录用户访问 HLS

生成器生成的视频直播文件，Web 前端使用第三方库 video.js，实现播放视频监控 HLS 的功能。

5.2.2 通信接口模块的实现

服务端的的通信接口模块使用 Java 程序实现。根据本文的通信协议定义（见第四章），服务端监听 TCP 15001 端口，作为视频监控平台的应用端口。服务端使用 Spring TCP Connection Factory 实现 TCP 端口的监听及通讯数据的接收和发送。服务端与监控终端的通信流程如图 5-2 所示。

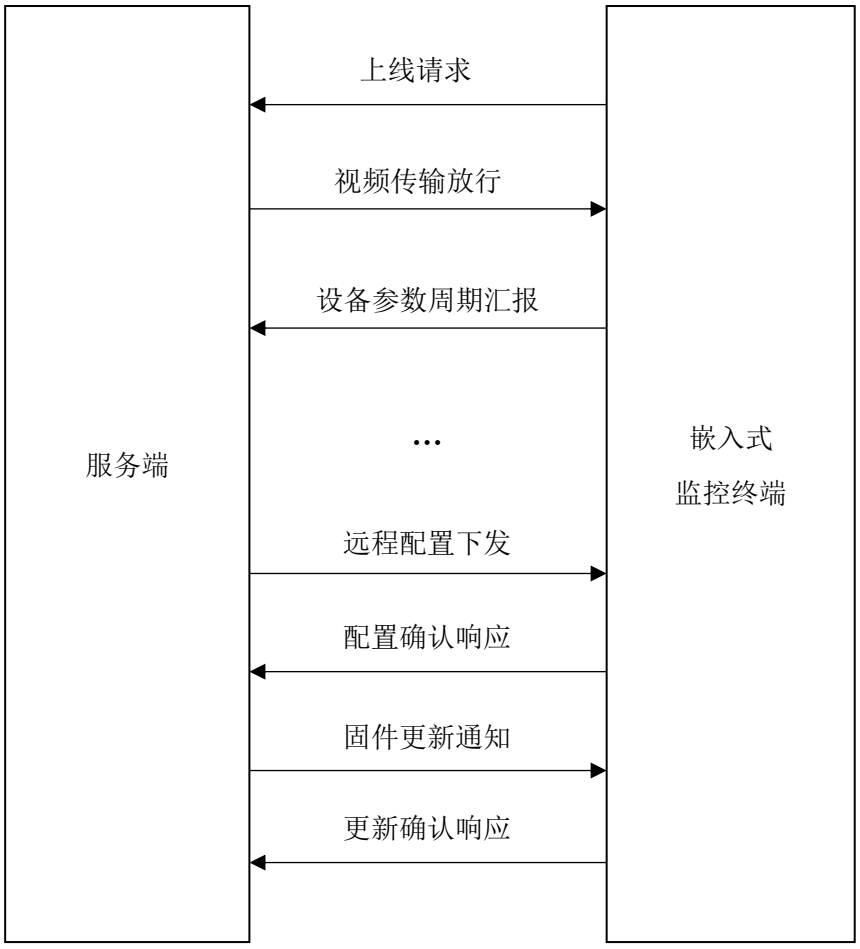

图 5-2 服务端与监控终端的通信流程示意图

5.2.3 视频接收处理模块的实现

服务端视频接收处理模块负责 RTP 及 RTCP 包的解析，H.264 视频储存，和 HTTP 实时视频流（HTTP Live Stream）生成功能，采用开源项目 FFMpeg 实现上述功能。

FFMpeg 是一个可以用来记录、转换数字视频，处理视频流的开源程序。

服务端为每个 RTP 连接，使用如下命令行参数启动 FFMpeg 进程，对 RTP 及 RTCP 封装的视频进行提取处理。

```
ffmpeg -i sdp/video0.sdp -vcodec libx264 \
  -f segment -segment_time 60 -strftime 1 record/%d.mp4 \
  -f hls -g 1 -hls_time 1 -hls_list_size 5 -hls_allow_cache
0 hls/video0.m3u8
```

以上命令行参数指定 FFMpeg 监听 UDP，对传入视频进行重新切片封装处理，每分钟进行分段存储到服务器，同时保存 HLS 实时视频列表，以供实时预览功能查看。

5.2.4 升级管理模块的实现

升级管理模块用于为服务端提供程序更新。其原理是，在互联网更新网站上提供升级包下载，并附带一个结构如下的元数据文件：

```
{
  “latest”: “0.1.2”,
  “file”: “http://<domain>/videomon-server-0.1.2.tar.gz”,
  “sha1”: “1e7b92c061548ab099def1c19aa764edb7a9d8b9”
}
```

当互联网上的版本高于当前服务器版本时，下载该版本对应的压缩包文件，进行 SHA1 哈希校验。校验检查通过后，解压运行 update.sh。程序文件更新完成后，重启服务器，完成平台整体的升级。

5.3 服务器 HTTP 软件的安装与配置

完成服务端程序的编写后，为了让服务端能响应外部 HTTP 访问，还需要在服务器上配置反向代理服务软件。

反向代理服务软件是一种特殊的 HTTP 服务器软件，它不会直接响应客户端的请求，而是根据一定的判断条件，将请求发送到配置指定的某个或某些“上游”后端服务器，由后者进行处理，反向代理服务器再将处理结果的响应传回到客户端。

当前最常用的反向代理服务器是 NGINX，它具有运行效率高、功能强大的特点，

具体有以下优势[12]：

1. 能够实现负载均衡功能，以保证服务高可用。在有多台后端服务器的情况下，使用反向代理的负载均衡功能，可以有效避免其中一台后端服务器故障导致的单点故障，提升系统可用性。
2. 支持 HTTPS 安全传输协议。本论文设计的视频监控平台对安全性有较高要求，如果客户端与服务端的信息通过没有加密的 HTTP 直接传输，极易遭到嗅探或中间人攻击，存在信息泄密的风险。而采用了反向代理服务的 HTTPS 功能后，通信均经过加密，即使数据报文被嗅探，也不能被攻击者非法解密。
3. 提供了日志功能，可以高度自由地定制接口请求日志，便于系统的安全审计和可用性统计。

在 Linux 服务器上安装 NGINX 反向代理服务器的步骤如下：

1. 安装 deb 依赖项

```
apt-get build-dep nginx
```

2. 下载 NGINX 的源码，并解压到当前路径

```
wget http://nginx.org/download/nginx-1.13.0.tar.gz
tar xzvf nginx-1.13.0.tar.gz
```

3. 进入源码目录，配置和编译

```
cd nginx-1.13.0
./configure --with-http_addition_module \      # HTTP 模块杂项
            --with-http_v2_module \          # HTTPS 支持
            --with-stream \               # TCP 代理
            --with-stream_ssl_module \      # TCP with SSL
            --with-threads
make -j16 && make install
```

通过以上步骤，NGINX 服务器编译安装完成。接下来要通过编写配置文件，使其能够将请求转发到我们的上游服务端程序。配置编写如下：

代码 3 服务器配置（文件名：videomon.conf）

```
01  # 定义一个上游服务器组
02  upstream videomon_backend {
```

```
03      server 127.0.0.1:8080;
04  }
05  # 定义一个 HTTP(S)服务器
06  server {
07      listen 80;
08      listen 443 ssl http2; # 此处配置了监听 HTTPS 加密连接
09      server_name videomon-dev-internal.senorsen.com; # 对外服
务域名
10      # 以下为 HTTPS 相关的安全配置
11      add_header Strict-Transport-Security "max-age=36000000;
"; # HSTS 安全设置
12      ssl_session_cache shared:SSL:20m;
13      ssl_session_timeout 360m;
14      ssl_protocols TLSv1 TLSv1.1 TLSv1.2; # 可用 TLS 协议列表
15      # 加密算法列表
16      ssl_ciphers EECDH+AES128:RSA+AES128:EECDH+AES256:
    RSA+AES256:!MD5;
17      ssl_prefer_server_ciphers   on;
18      ssl_certificate /srv/ssl/senorsen.com.crt;    # SSL 证书
19      ssl_certificate_key /srv/ssl/senorsen.com.key;# SSL 私钥
20      add_header Cache-Control no-cache;
21      # 强制跳转到 HTTPS 连接
22      if ($https != 'on') {
23          return 301 https://$host$request_uri;
24      }
25      location / {
26          proxy_pass http://videomon_backend;
27          proxy_set_header Host $host;
28          proxy_set_header X-Forwarded-For $remote_addr;
29          proxy_set_header X-Forward-Proto $proto;
30      }
31  }
```

通过本节的反向代理服务配置，服务端已经可以接受外来的 HTTPS 连接，进行 Web 服务。

5.4 本章小结

本章对视频监控服务端的设计和实现进行了研究。在本章中，首先根据视频监控平台应具有的各项功能，对服务端进行了模块划分。然后对 Web 管理服务模块、通信接口模块、视频接收处理模块、升级管理模块进行功能实现，实现了完整的视频监控服务端。最后，进行服务器 HTTP 服务器软件的安装配置，使视频监控服务端达到可用的状态。

第七章　总结与展望

7.1 总结

随着信息技术的高速发展，视频监控技术正向着数字化、网络化、智能化的方向发展。本设计的任务是研究基于嵌入式 Linux 系统的多路视频监控平台，主要完成了以下工作：

1. 完成嵌入式系统和服务器平台的技术选型。因本平台的功能需求多，考虑到 Raspberry Pi 官方技术文档齐全，Linux 开放源码、资源丰富，故选择了基于 BCM2835 SoC 的 Raspberry Pi Model B 开发板与嵌入式 Linux 的嵌入式系统方案。考虑到 Spring 框架技术成熟、先进，功能强大，选择其作为服务器端开发平台。

2. 完成了嵌入式系统软件平台的构建。这部分包括宿主机交叉编译环境的准备，嵌入式 Linux 系统内核的裁剪、构建，Linux 根文件系统的构建等。

3. 完成了视频监控终端的设计与实现。将视频监控终端分为视频处理模块、通信接口模块与固件 OTA 更新模块。视频处理模块中使用 Video4Linux 2 API 进行摄像头视频图像的采集，然后调用 OpenMAX IL 进行 H.264 视频硬编码压缩，最后编写 RTP 协议进行视频流的传输；为通信接口模块设计实现了通信协议；在固件 OTA 更新模块中，实现了基于网络远程更新设备固件的功能。

4. 完成了视频监控服务器平台的软件编写。使用 Java / Kotlin + Spring 技术方案，实现了由 Web 管理服务模块、通信接口模块、视频接收处理模块、升级管理模块组成的视频监控服务端。

本设计遵循模块化原则，将整个平台分为不同模块，模块之间通过接口调用。这种设计方法降低了模块间的耦合，提升了整体的灵活性和扩展性，当系统需要更换模块时，只需要做很少的修改，有利于日后系统的进一步开发。

经过以上工作，本文作者对嵌入式系统开发有了更进一步的了解，积累了很多嵌入式 Linux 开发的经验。不过因为时间有限，本设计也有一些不足之处：

1. 嵌入式硬件使用的是已经成型的 RPI 开发板，只适用于研究用途。可以自行设计基于 ARM 芯片的嵌入式系统软硬件平台，根据功能需求专门定制硬件。

2. 监控摄像头没有云台，不能远程操作移动摄像头视角。鉴于在安全监控中，移

动摄像头视角的需求普遍，将来可研究云台的实现。

3. 嵌入式终端代码中，有些地方对特定硬件有依赖，可能不利于将来的跨硬件平台的移植。可以进一步编写硬件抽象接口，为将来的移植打下更好的基础。

7.2 展望

随着计算机技术的发展，嵌入式监控平台的应用愈加广泛。基于嵌入式的监控系统利用计算机的强大处理能力，可高效地进行视频采集和处理，具有很强的实用性。

嵌入式监控平台的软硬件高效结合，高性能的硬件与自由组合的软件，使得多种多样的复杂功能得以实现。目前图像处理技术越来越先进，在图像识别技术的基础上，嵌入式还有望实现人员面部识别、自动报警等功能，从而更好地保障安全。总之，嵌入式监控平台正朝着数字化、网络化、智能化的方向不断发展，拥有非常广阔的前景。