摘 要

随着人民生活水平的提升，生活中的音乐播放工具也在不断变化。从以前的随身听，DVD，到现在的手机电脑，音乐播放工具的进步让我们可以更加便捷的聆听音乐，但是在汽车播放器方面，还有很多在使用原始的光盘播放，或者是U盘播放，这极大地妨碍了我们随时随地便捷的欣赏音乐，所以我要设计一款智能音箱，能够使用手机进行音乐的播放，暂停等功能，为了使音乐的播放更加方便，加入服务器，只要有网，可以连接服务器，随时播放音乐。

下面分析并确定要实现的功能，进行方案的论证与选择。在硬件方面，需要能够运行非实时操作系统的CPU，所以选择Cortex-A53，选择Linux操作系统作为嵌入式系统，音频解码选择软件解码，电路设计采用AD绘制，加入网络模块。软件方面，服务器采用腾讯云，使用Ubuntu系统作为服务器，同时使用socket做网络编程。

本文介绍了一种基于以太网的智能音箱设计与实现。在设计中加入网络模块用于连接网络服务器，收发来自客户端的命令。使用socket基于TCP/IP实现自己的网络协议，通过自定义的网络协议实现客户端，服务器端，嵌入式端的三端通讯。

制作完成的智能音箱能够实现使用客户端远程控制智能音箱的播放，暂停，停止，上一曲，下一曲，音量的放大减小功能，达到了便捷欣赏音乐的需求。

关键词： 智能音箱； 嵌入式； 以太网； 网络编程

**Abstract**

With the improvement of people's living standards, the music playing tools in life are also changing constantly.From the previous walkman, DVD, mobile phone computer until now, the progress of music playback tools that we can more convenient to listen to music, but in terms of car player, there are many in the use of the original CD playback, or U disk, this greatly hampered our convenient to enjoy music anytime and anywhere, so I want to design a intelligent speakers,Can use the mobile phone for music playback, pause and other functions, in order to make music playback more convenient, join the server, as long as there is a network, you can connect to the server, at any time to play music.

The following analysis and determine to achieve the function, the demonstration and choice of the scheme.In terms of hardware, CPU capable of running non-real-time operating system is required, so Cortex-A53 is selected, Linux operating system is selected as the embedded system, audio decoding software is selected to decode, circuit design adopts AD drawing, and network module is added.In terms of software, Tencent cloud is used as the server, Ubuntu system is used as the server, and socket is used for network programming.

This paper introduces the design and implementation of an intelligent speaker based on Ethernet.Add a network module in the design to connect to the network server and send and receive commands from the client.Using Socket based on TCP/IP to achieve their own network protocol, through the custom network protocol to achieve client, server, embedded end of the three-terminal communication.

The finished intelligent speaker can realize the use of the client remote control of the intelligent speaker playback, pause, stop, the last song, the next song, volume amplification and reduction function, to achieve the needs of convenient music appreciation.

**Keywords:** Intelligent speaker; Embedded; Ethernet; Network programming 目 录

[1. 绪论 1](#_Toc11358401)

[1.1 研究背景与意义 1](#_Toc11358402)

[1.2 智能音箱的发展现状 2](#_Toc11358403)

[1.3 论文结构安排 3](#_Toc11358404)

[2. 系统的总体方案设计 4](#_Toc11358405)

[2.1 引言 5](#_Toc11358406)

[2.2 设计要求 5](#_Toc11358407)

[2.3 方案选择 6](#_Toc11358408)

[2.3.1 CPU的选择 6](#_Toc11358409)

[2.3.2 网络模块的选择 7](#_Toc11358410)

[2.3.3 音频模块的选择 7](#_Toc11358410)

[2.3.4 硬件的总体方案设计 8](#_Toc11358410)

[2.4 本章小结 9](#_Toc11358411)

[3. 系统的嵌入式端设计 1](#_Toc11358412)0

[3.1 引言 1](#_Toc11358413)1

[3.2 整体方案设计 1](#_Toc11358414)2

[3.2.1 嵌入式系统烧录 1](#_Toc11358415)3

[3.2.2 开发环境搭建 1](#_Toc11358416)5

[3.2.3 音频功能开发 1](#_Toc11358417)8

[3.2.4 网络功能开发 2](#_Toc11358418)0

[3.3 本章小结 2](#_Toc11358420)4

[4. 系统的服务器端设计 2](#_Toc11358421)5

[4.1 引言 2](#_Toc11358422)6

[4.2 云服务器环境搭建 2](#_Toc11358423)7

[4.3 服务器程序流程图 2](#_Toc11358424)8

[4.3.1 服务器程序实现 2](#_Toc11358425)9

[4.3.2 服务器程序部署 3](#_Toc11358426)0

[4.3.3 服务器与嵌入式端通讯测试 3](#_Toc11358427)1

[4.4 本章小结 3](#_Toc11358430)2

[5. 系统的客户端设计 3](#_Toc11358421)3

[5.1 引言 3](#_Toc11358422)4

[5.2 客户端UI界面 3](#_Toc11358423)5

[5.3 客户端程序流程图 3](#_Toc11358424)6

[5.4 客户端程序实现 3](#_Toc11358424)6

[5.5 客户端程序测试与编译 3](#_Toc11358424)7

[5.6 三端通讯测试 3](#_Toc11358424)7

[5.7 本章小结 3](#_Toc11358430)8

[结束语 3](#_Toc11358431)9

[参考文献 4](#_Toc11358432)0

[致谢 4](#_Toc11358433)1

# 1. 绪论

## 1.1 研究背景与意义

随着人民生活水平的提升，对于文化娱乐的需求日益增长，在此情况下音乐是满足人们精神需求的重要途径。音乐是人类在感悟、表达、体会对世界的理解，释放灵魂的压力。音乐与其它艺术形式在满足的需求方面有着天然优势，那就是“伴随”，我们可以仅仅通过萦绕在头脑中的旋律满回想起家的温暖，爱的感动，悲怆与震撼等等的感情。就像影视剧的煽情离不开音乐，冲锋也要号角，音乐是在记录情感，音乐与灵魂同在。

音乐充斥在我们的生活中，现在的大部分音箱是不能跨越端到端的距离，我们需要的是根据自己的喜好建立一个音乐库，我们在任何端都可以上传自己喜欢的音乐，并且可以随时随地听到音乐。其中车载音箱是一个典型代表，在车载音箱方面，如今的中低端市场上还有很多音箱使用光盘，U盘作为存储介质进行音乐播放，如果想要播放自己喜欢的音乐则需要事先网络下载，不能快速高效地聆听自己喜欢的音乐。

目前国内的汽车音箱，大多数都设在汽车用品和汽车美容装饰店，担任操作的是缺乏音箱经验和音箱知识的小工，仅仅用器材的品牌和价格对车主进行宣传，使本来就对音箱不熟悉的车主误认为这便是汽车音箱改装的全部内容。有些改装后的音箱，其效果和器材性能不仅没有得到正常发挥，甚至还损坏了原车的电器系统，给车主留下了日后安全上的隐患。因此智能音箱的研究是可以切实提升人们在精神需求的追求，更快捷地寻找音乐，更动听的音乐播放，更方便的音乐存储，这三方面是提升音乐追求的途径。

## 1.2 智能音箱的发展现状

微型化、数字化、专业化、影视化是智能音箱的发展趋势。

微型化音箱。微型台式组合音箱已有较长的发展史，在10多年前就已经出现高级超小型组合音箱。但由于听音喇叭、立体声电唱机、录音卡座没有很好解决，所以一直停留在较低的档次上。为了创造小巧的音箱世界，不但要从放大器、控制部件、左右音箱上下功夫，还得从调谐器、CD唱机和录音卡座方面一起考虑。

数字化音箱是在解决模拟音箱噪声的失真问题时发展而成。音箱采用了数字技术之后，记录的数字信号从取样频率到量化特性，有较高的解像度，没有色抖动，得到是非常清晰的图像。而且可以和上位机互换，这与模拟录放像设备无法比拟。数字录音可以把时间、人名、地址一起录入带中，采用微型键盘来完成编目工作，更换曲目编号，再加上遥控功能，使你能够自动地搜索需要的曲目，使用方便。影视听设备一体化。数字音箱随着电声技术、影视技术、计算机技术的发展，它们在家庭中可以构成浑然一体的多媒体影视音频系统。这样的系统，能听到各种在输入端增添各种需要的信号输入和功能转换，通过电脑处理就能使众看到各种图像和听到各种声音。

超薄平板音箱，平板音箱的出现使家庭组合（家庭影院、背景音乐）向超薄方向发展成为了可能自1998年到2012年平板音箱也经历了几个发展阶段，1998年三诺公司引进平板音箱，到华龙帝声、托微克继承和发展，到2008年成都天翔研发出中国自己平板发音技术"VT"称“第五代平板发音技术”，平板音箱技术在中国的发展越来越趋于成熟，薄型壁画超薄音箱在家庭中的使用趋于成熟化！

嵌入式音箱，嵌入式音箱的出现源于1998年的成都福韵，经历了十几年的发展，嵌入式音箱在“家庭中央网络音箱系统”、“嵌入式家庭影院”中发挥了很大的作用；成都天翔结合平板音箱和嵌入式音箱的特长，研发出新一代家庭音箱系统“家庭养生音箱系统”，为家庭音箱的发展做出了试探型的一步。

## 1.3 论文结构安排

本文研究的内容是基于以太网的智能音箱的设计与实现，各章节具体安排如下：

（1）第一章为绪论，说明了研究基于以太网的智能音箱的背景和意义，阐述了国内外目前的发展状况和未来的发展方向，还加入了对论文的结构安排的简单描述。

（2）第二章首先对设计要求进行了说明，接着进行了该设计的方案选择，包含了嵌入式系统的类型选择和声音控制方案的选择。

（3）第三章介绍了系统的嵌入式端设计，首先是三端通讯的总体设计方案，然后是系统框图，接下来分别介绍了协议文件、网络模块、控制模块和数据传输模块，其中包括Linux系统的烧录，开发环境的搭建，以及音频测试。

（4）第四章介绍了系统的服务器端设计，首先介绍了网络协议的转发，以及对嵌入式端以及客户端的识别方法，然后是对于云服务器的介绍和云服务器的搭建，云服务器和嵌入式端的通讯测试。

（5）第五章介绍了系统的客户端设计，首先介绍各个模块的主要功能和代码，从UI排布，到具体功能实现的逻辑。接下来是使用定义的网络协议将指令发送到服务器，同时也可以接收由服务器从嵌入式端收来的消息。最后是综合测试，进行三端的通讯连接测试，完成基于以太网的智能音箱。

# 2. 系统的总体方案设计

## 2.1 引言

作为一款基于以太网的智能音箱，芯片的选择十分重要，如果一个芯片不具有MMU内存管理单元，那么是无法运行非实时操作系统的，对于编写复杂程序有很大阻碍，所以在满足设计要求的基础上，使用一款强大的CPU是完成设计的关键。除此之外，元器件的功耗、成本和使用寿命等也需纳入考虑范围。

## 2.2 设计要求

设计一款智能音箱，要达到方便，快捷的目的。该智能音箱需要实现以下功能：

1. 客户端可以连接服务器，获取音乐目录；
2. 客户端可以控制嵌入式端的播放；
3. 客户端可以控制嵌入式端的暂停；
4. 客户端可以控制嵌入式端的停止；
5. 客户端可以控制嵌入式端的上一曲；
6. 客户端可以控制嵌入式端的下一曲；
7. 客户端可以控制嵌入式端的音量增减；

## 2.3 方案选择

### 2.3.1 CPU的选择

采用STM32F103作为系统CPU。STM32F103是一种入门级微处理器，STM32F系列属于中低端的32位ARM微控制器，该系列芯片是意法半导体（ST）公司出品，其内核是Cortex-M3。STM32F103的优点是：芯片集成定时器Timer，CAN，ADC，SPI，I2C，USB，UART等多种外设功能，缺点是：因为属于中低端产品，因此不具有MMU内存管理单元，可以运行如FreeRTOS，uclinux，μC/OS-II等实时操作系统，对于编写复杂程序来说比较考验功底。

方案二：采用Cortex-A53作为CPU。Cortex-A53属于高端的64位ARM微控制器，甚至于部分手机处理器的内核也是采用的Cortex-A53，具有MMU内存管理单元，这里选择RK3328芯片，其具有4核64位的Cortex-A53处理器，40PIN扩展接口，包含了常用的GPIO、I2S、SPI、I2C、UART、PWM、SPDIF等，独有的EMMC存储器扩展接口，支持外部增加EMMC存储器，在解码能力方面：支持4KVP9，4K10bitsH265/H264视频解码，高达60fps，支持1080P(VC-1,MPEG-1/2/4,VP8)多格式音视频解码。可以流畅运行Android，Linux操作系统

基于以上分析，拟定方案二，采用RK3328芯片作为控制器。

### 2.3.2 网络模块的选择

市面上有很多网络模块可以选择，最常见的莫过于ESP8266，ESP32 的核心电路只需要 20 个左右的电阻电容电感、1 个无源晶振及 1 个 SPI flash 组成。ESP32 集成了完整 的发射/接收射频功能，包括天线开关，射频 balun，功率放大器，低噪放大器，过滤器，电源管理模块和先进的自校准电路。ESP32 的高度集成使得其外围电路设计比较简单。为了能够更好地保证 ESP32 工作性能，

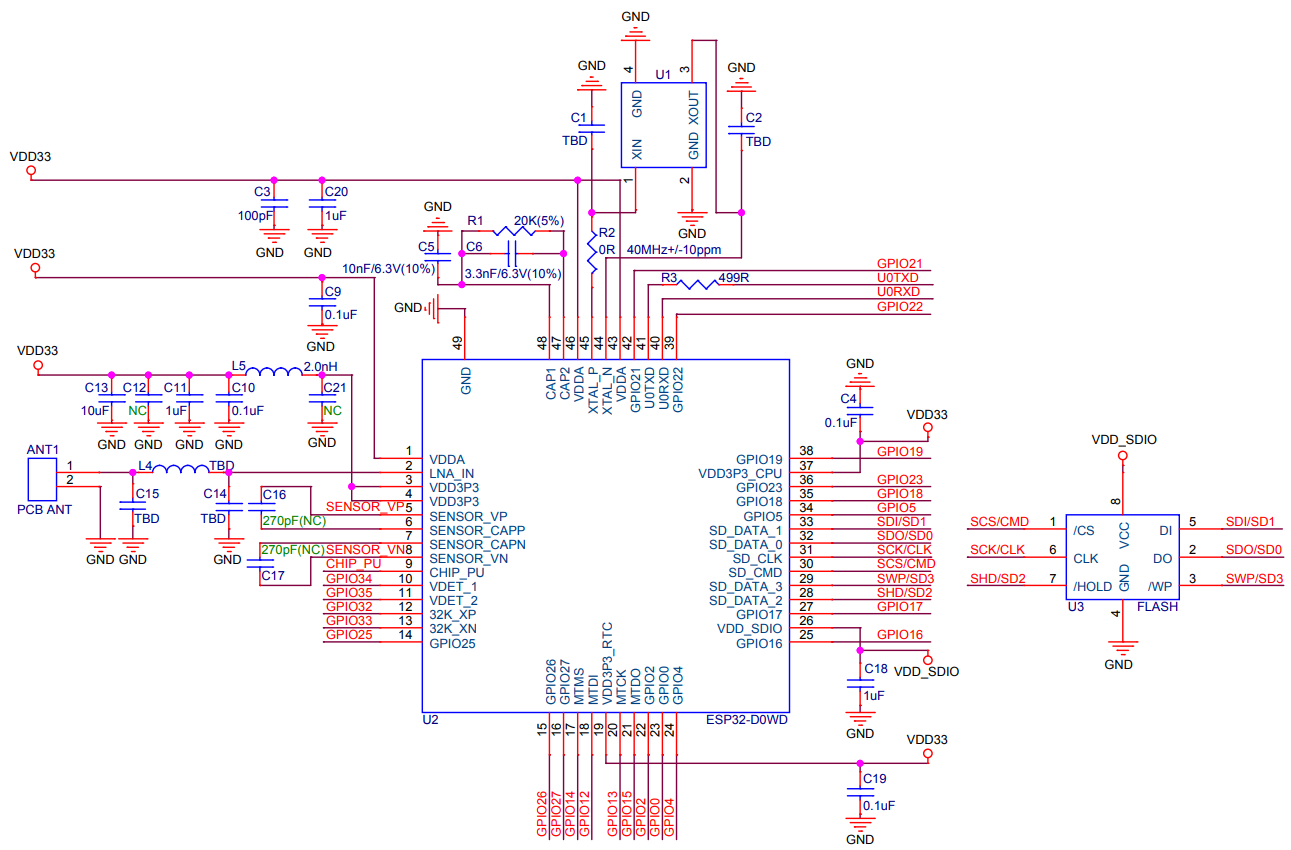


图2.1 ESP8266原理图

ESP32的pin19及pin37分别为RTC电源管脚和CPU电源输入管脚。数字电源工作电压范围为1.8V~3.6V。电路中靠近数字电源管脚处分别添加0.1µF电容。VDD\_SDIO管脚(pin26)可配置输出1.8V（Boot启动时，需GPIO12的值为1），或输出3.3V（Boot启动时，需GPIO12的值为0，默认状态），给外部电路使用。当VDD\_SDIO处于1.8V模式时，由ESP32内部的LDO产生。LDO能提供的最大电流为40mA，输出电压范围为1.65V~2.0V。当VDD\_SDIO输出为1.8V时，建议在VDD\_SDIO管脚处添加2kΩ对地电阻及4.7µF对地电容。当VDD\_SDIO处于3.3V模式时，由VDD3P3\_RTC通过约6Ω电阻直接供电。因此，VDD\_SDIO相对VDD3P3\_RTC会有一定电压降。当VDD\_SDIO输出3.3V时，建议在VDD\_SDIO靠近管脚处添加1µF的滤波电容。VDD\_SDIO也可由外部电源供电。另外，当使用VDD\_SDIO给外部3.3Vflash/PSRAM供电时，需要满足flash/PSRAM的工作电压要求，一般应保证电压在3.0V以上。

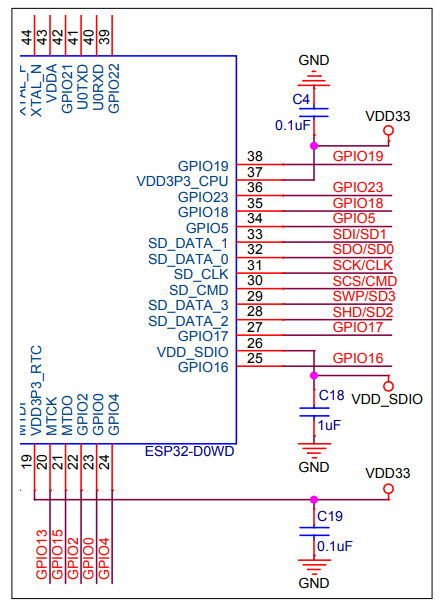


图2.2 ESP8266的数字电源原理图

ESP32的pin1、pin3、pin4、pin43、pin46为模拟电源。该部分电源需要注意的是当ESP32工作在TX时，瞬

间电流会加大，往往引起电源的轨道塌陷。所以在电路设计时建议在电源走线上增加一个10µF电容，该电容

可与0.1µF电容搭配使用。另外，在靠近pin3，pin4管脚还需添加LC滤波电路，用于抑制高频谐波，同时请

该电感的额定电流最好在500mA及以上。

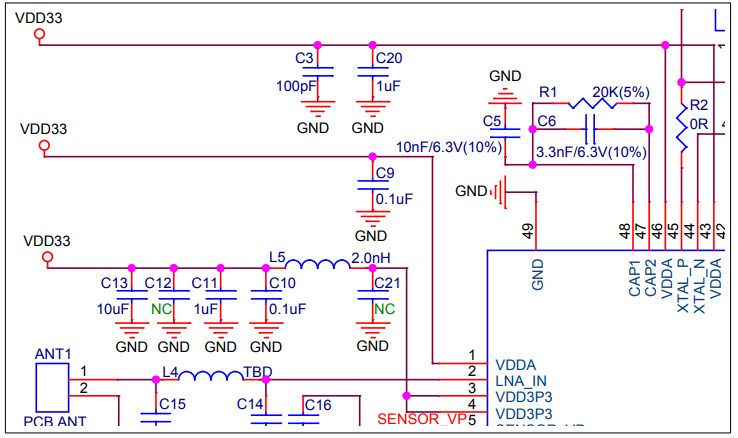


图2.3 ESP8266的模拟电源原理图

### 2.3.3音频模块的选择

音频模块选择LM386，LM386是美国国家半导体公司生产的音频[功率放大器](https://baike.baidu.com/item/%E5%8A%9F%E7%8E%87%E6%94%BE%E5%A4%A7%E5%99%A8/1267443" \t "https://baike.baidu.com/item/_blank)，主要应用于低电压消费类产品。为使外围元件最少，[电压增益](https://baike.baidu.com/item/%E7%94%B5%E5%8E%8B%E5%A2%9E%E7%9B%8A" \t "https://baike.baidu.com/item/_blank)内置为20。但在1脚和8脚之间增加一只外接电阻和电容，便可将电压增益调为任意值，直至 200。输入端以地为参考，同时输出端被自动偏置到电源电压的一半，在6V电源电压下，它的静态功耗仅为24mW，所以LM386特别适用于电池供电的场合。LM386的特性有：静态功耗低，约为4mA，可用于电池供电；工作电压范围宽。外围元件少；电压增益可调，20-200；低失真度等。



图2.4 LM386实物图

### 2.3.4 硬件的总体方案设计

定了控制器和控制方案后，需要给出整体的设计方案。第三章主要描述了系统的硬件设计和所用元器件介绍，硬件部分设计的的电路原理图如图3.1所示，根据原理图制作出的智能音箱实物如图3.2所示。

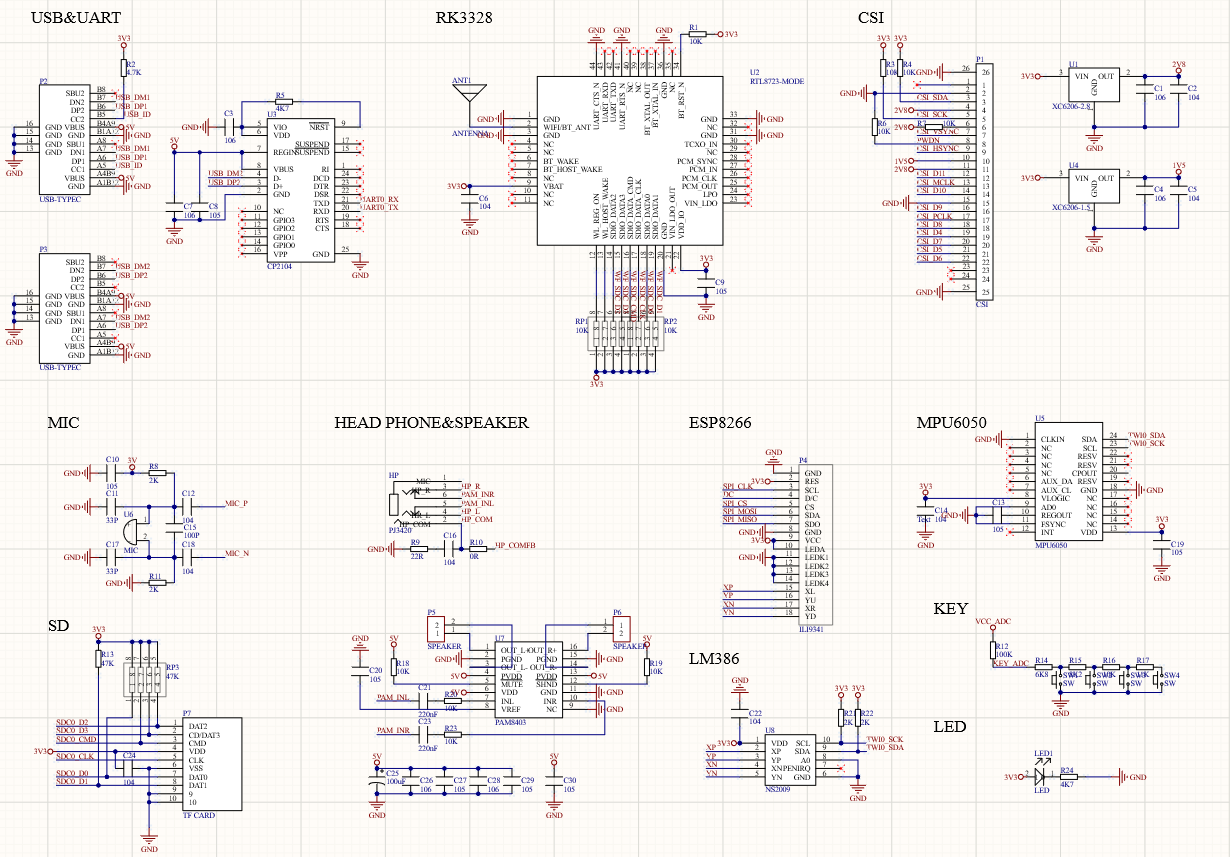


图2.5 总电路原理图

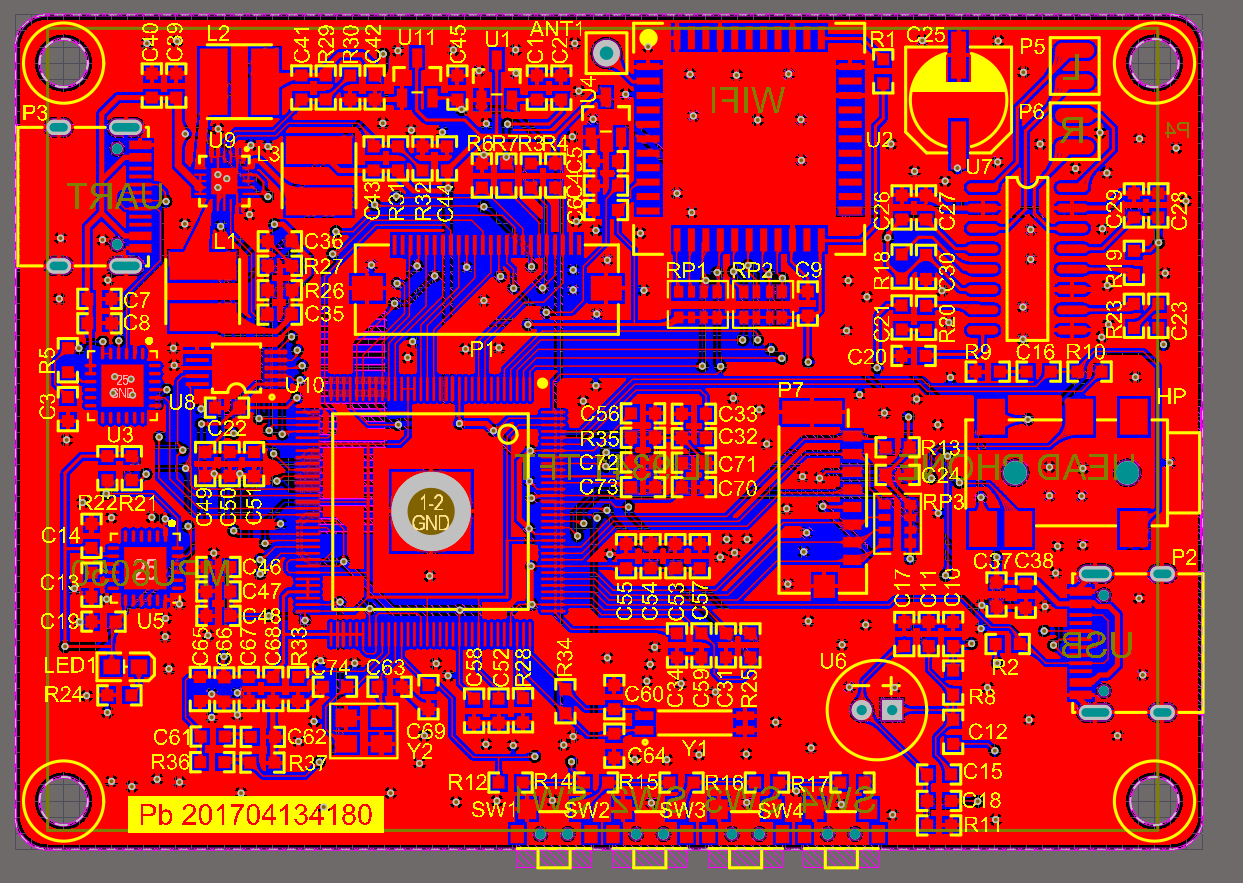


图2.6 PCB电路图

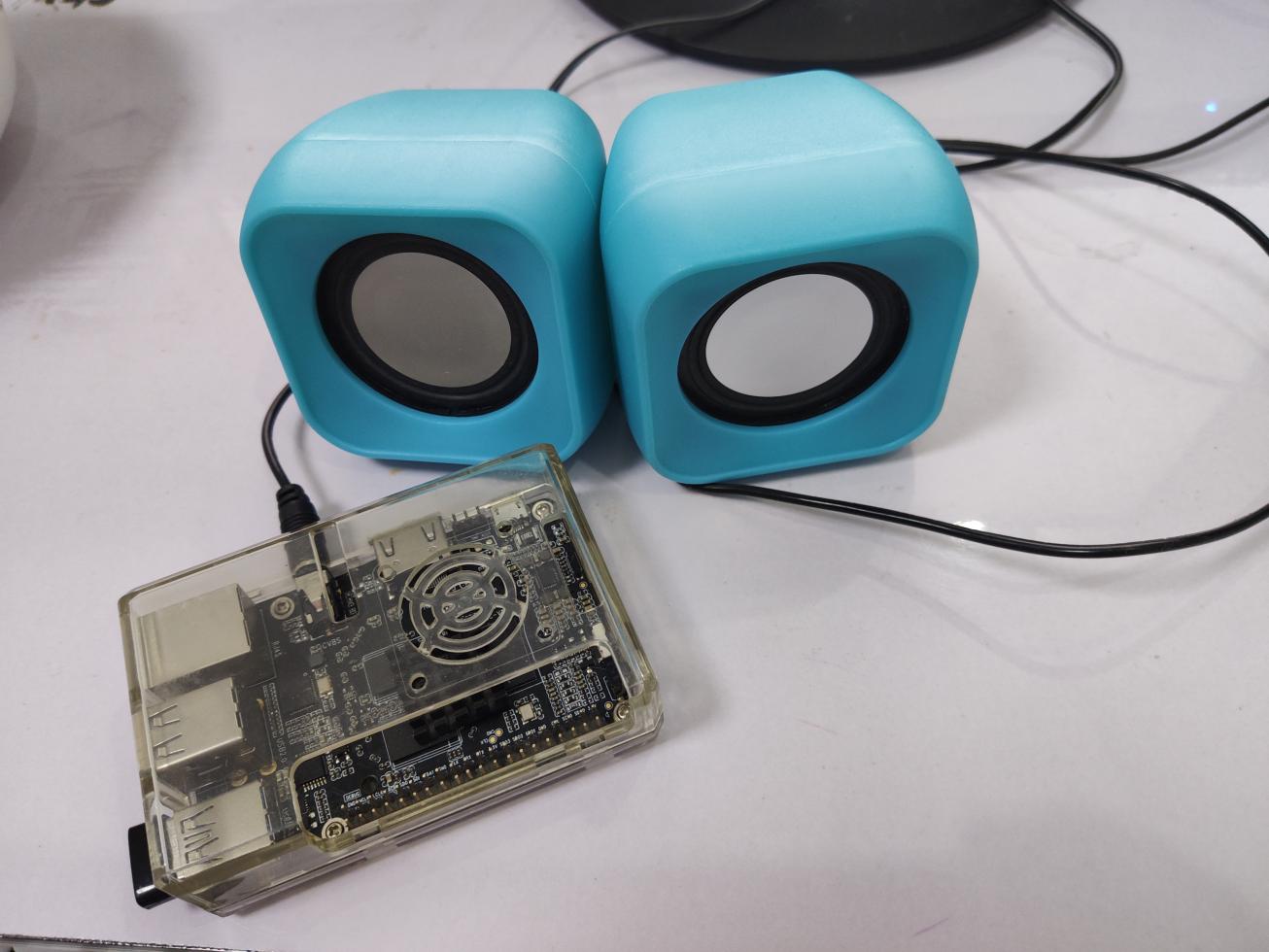


图2.7 智能音箱实物

## 2.4 本章小结

本章介绍了硬件的方案选择，以及其各类参数的情况，确定了使用RK3328作为智能音箱的控制器，ESP8266作为网络模块，LM386作为音频模块的方案，最后给出了硬件的总体方案设计。

# 3. 系统的嵌入式端设计

## 3.1 引言

嵌入式端首先需要植入Linux系统，这里选择Ubuntu18.04系统作为嵌入式端的系统，系统烧录到SD卡中，嵌入式板子从SD卡启动系统。系统启动后需要安装运行环境，为方便三端通讯，所以需要一款平台语言进行程序设计，目前的选择有Java和C#，C、C++四款语言。

C语言方便嵌入式，但是对于服务器和客户端开发不友好，图形界面以及各类库比较少，且文件配置繁琐，开发难度最大。

C++语言嵌入式，客户端，服务器都有较好效果，类库丰富，唯一缺点就是程序内容较多，如果是多人合作开发，是最好的选择，对于单人来说，工作量较大，开发难度较难。

Java语言在嵌入式方面需要安装虚拟机，但得益于强大的处理器，所以效率也不低，在服务器方面略逊C++，但是客户端效果最好，开发手机客户端比较适宜。

C#语言和Java类似，在嵌入式工作需要安装dotnet虚拟机，服务器逊于C++，客户端可以采用Unity来进行跨平台编译，开发比较适宜。

综合工作量，开发难度，最后选择C#作为三端的共同语言。

## 3.2 整体方案设计

底层采用json做网络协议的解析和编写，使用socket做TCP/IP的消息发送和接收，在服务器端和嵌入式端使用反射调用消息解析后的函数，同时将消息传入。嵌入式端使用mpg123做音频解析，使用pulseaudio做声音控制。服务器端使用Ubuntu18.04系统，使用dotnet虚拟机运行C#服务器软件。客户端使用unity开发，Android打包。

### 3.2.1 嵌入式系统烧录

将SD卡插入读卡器，运行Win32DiskImager，将系统写入SD卡，启动电源。

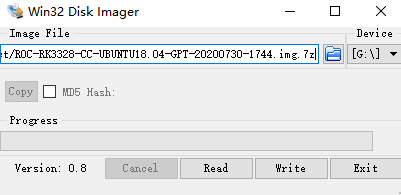


图3.1 系统烧录

### 

图3.2系统桌面

### 3.2.2 开发环境搭建

需要进行远程控制，安装SSH，FTP进行命令行控制，文件传输，安装x11vnc显示远程桌面，安装dotnet开发包

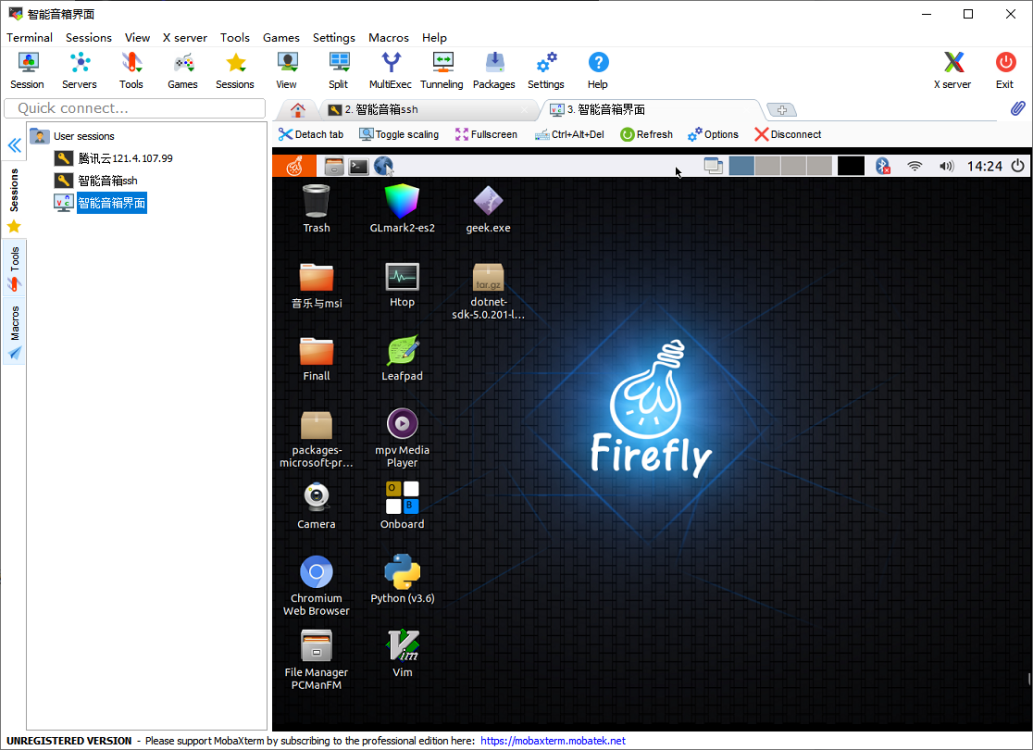


图3.3 远程桌面控制

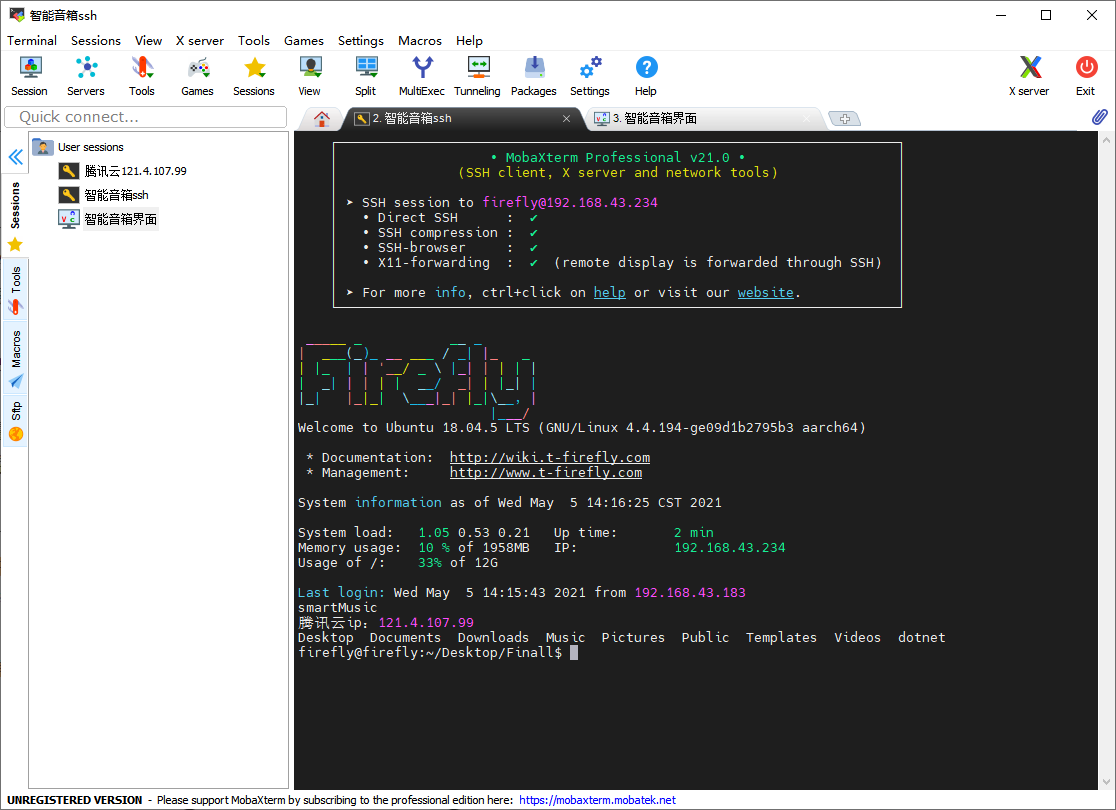


图3.4 ssh控制

完成所有配置后可以通过vnc与ssh进行嵌入式端的远程开发，不需要每次插拔数据线，开发完成只需要拖进窗口，即可执行程序。

### 3.2.3 音频功能开发

首先做好离线功能，测试离线功能没有问题的情况下，加入网络模块。播放音乐使用mpg123，其中Nuget可以找到相关的库，同时也可以找到LitJson，极大地方便了开发，不需要去另外下载安装。

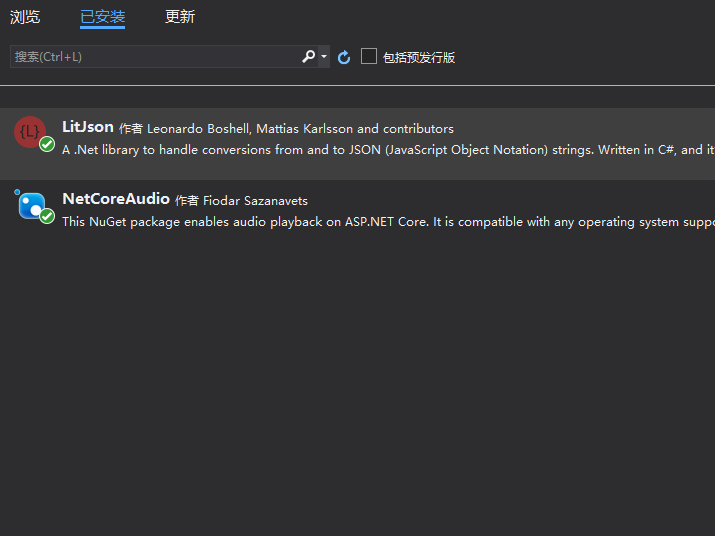


图3.5 库文件安装

程序流程图如下：

（1）目录功能：

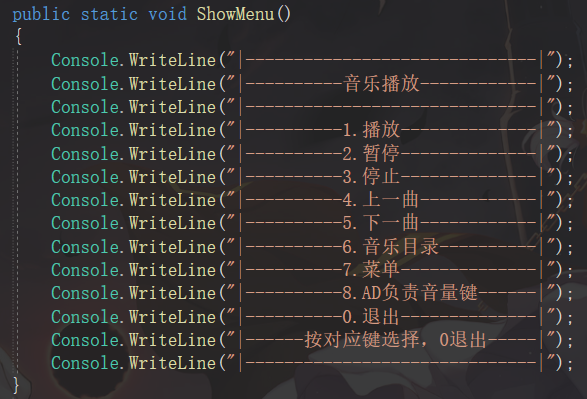


图3.6 目录功能

1. 按键检测功能：使用while循环检测按键是否按下，按下则触发对应函数。

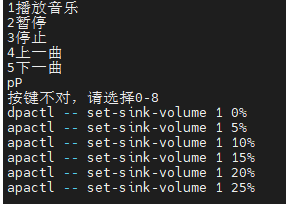


图3.7 按键功能

1. 获取指定目录的音乐：通过IO函数来获取文件路径，同时获取文件名。

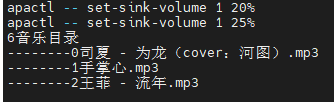


图3.8 获取音乐目录

（4）上一曲功能：将当前音乐停止，然后根据音乐目录播放上一曲音乐。

public static void UpMusic()

{

player.Stop();

//获得当前选中项,且往上

int index = currtMusicInt;

index--;

if (index < 0)

{

index = musicListPath.Count - 1;

}

currtMusicInt = index;//选中索引 往上

currtMusic = musicListPath[index];

priPlaymusic(currtMusic);

SendCurrentMusic();

}

（5）下一曲功能：将当前音乐停止，然后根据音乐目录播放下一曲音乐。

public static void DownMusic()

{

player.Stop();

//获得当前选中项,且往下

int index = currtMusicInt;

index++;

if (index == musicListPath.Count)

{

index = 0;

}

currtMusicInt = index;//选中索引 往下

currtMusic = musicListPath[index];

priPlaymusic(currtMusic);

SendCurrentMusic();

}

（6）暂停音乐功能：使用NetCoreAudio的函数Pause进行音乐的暂停

public static void PauseMusic()

{

try

{

player.Pause();

}

catch (Exception)

{

}

}

（7）停止音乐功能：使用NetCoreAudio的函数Pause进行音乐的停止

public static void StopMusic()

{

try

{

player.Stop();

}

catch (Exception)

{

}

}

1. 音量加减的功能：要控制嵌入式的板子音量增加或者减少，需要一个控制器，不同的系统采用的控制器不同，常见的如alsamixe，,pulseaudio，前者多用于服务器和PC，后者多用于嵌入式和微型系统，这里选择pulseaudio，安装后，使用命令查找声音控制的方法，可以看到以set开头的命令是控制声音的方法。

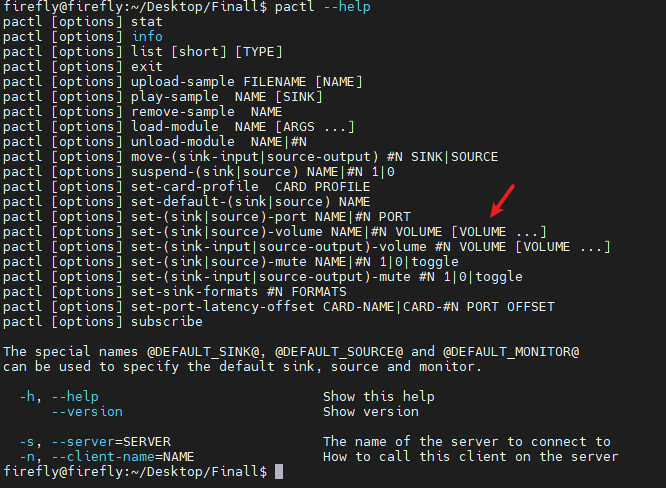


图3.9 音量控制命令

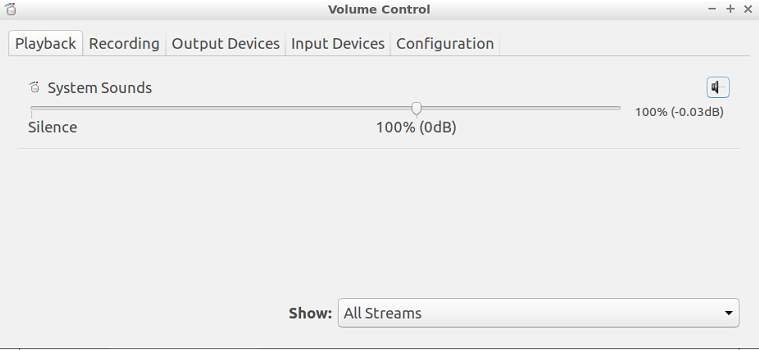


图3.10 音量控制面板

开发完成后效果如下：

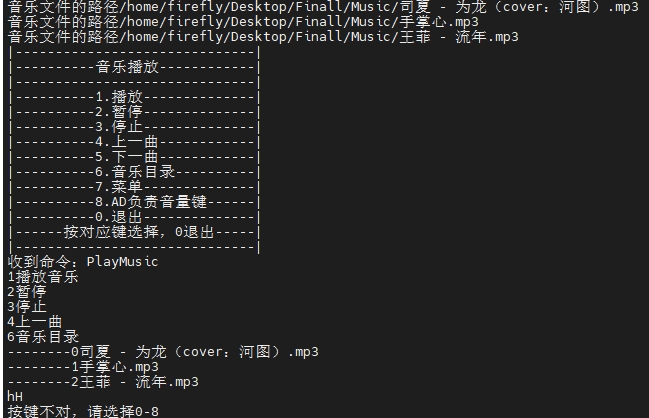


图3.11 嵌入式端程序界面

### 3.2.4 网络功能开发

网络在我们身边再常见不过，每天我们都在和网络打交道，那么我们的网络程序是如何运行的呢，通常来说，我们有七层网络结构，这七层网络结构是我们网络运行的根本，从最底层向上分为物理层，数据链路层，网络层，传输层，会话层，表示层，应用层。  
 物理层：物理层负责最后将信息编码成电流脉冲或其它信号用于网上传输。

数据链路层:数据链路层通过物理网络链路，可以简单的理解为：规定了0和1的分包形式，确定了网络数据包的形式。

网络层：网络层负责在源和终点之间建立连接，可以理解为，此处需要确定计算机的位置，IPv4，IPv6即定义在这一层。

传输层：传输层向高层，可以理解为每一个应用程序都会在网卡注册一个端口号，该层就是端口与端口的通信，常用的（TCP／IP）协议即在这一层。

会话层：会话层建立、管理和终止表示层与实体之间的通信会话，建立一个连接（自动的手机信息、自动的网络寻址）。

表示层:可以理解为解决不同系统之间的通信。

应用层:OSI 的应用层协议包括文件的传输、访问及管理协议(FTAM) ,以及文件虚拟终端协议(VIP)和公用管理系统信息(CMIP)等。  
 我们常见的HTTP，HTTPS，FTP，MQTT等等协议都是应用层协议，因为不同的协议携带的信息都不同，执行的方式也不一样，所以除了使用应用层的网络协议外，还有很多程序使用传输层传输自己定义的协议，在这里的智能音箱采用传输层的TCP/IP传输自定义协议。

在自定义协议之前，首先要明确协议分为几部分，智能音箱协议分为两部分，协议头和协议体，协议头表明此协议是什么，协议体则是具体信息，用一个类来定义协议。

第一个协议是实例化协议，此协议表明实例化对象是谁，当嵌入式端或者客户端上线之后，第一件事就是发送实例化协议，告诉所有网络对象，我上线了。

public class MsgInstance : MsgBase

{

public MsgInstance() : base("MsgInstance")

{

ids = new IDs();

}

/// <summary>

/// 客户端送 实体名

/// </summary>

public string id = "";

// public string id = "";

/// <summary>

/// 服务端回

/// </summary>

public IDs ids; //error List 要经过class包裹才能用Json

}

public class IDs

{

public List<string> entitys;

}

实例化协议完成之后，还需要命令协议，此协议表示客户端发送给嵌入式端的命令，内容包括播放，暂停，停止，上一曲，下一曲，音量增减，用一个枚举类型表示命令。

public class MsgOrder : MsgBase

{

public MsgOrder(string order) : base("MsgOrder")

{

this.order = order;

}

public MsgOrder() : base("MsgOrder")

{

}

/// <summary>

/// 命令客户端发，服务器转发

/// </summary>

public string order;

}

public class MsgTest : MsgBase

{

public MsgTest() : base("MsgTest")

{

}

}

public enum SmartOrder

{

PlayMusic, StopMusic, PauseMusic, DownMusic, UpMusic, MusicList

}

最后是协议的使用，当我们需要发送消息的时候，只需要新建一个协议对象，然后将协议对象内容填好，即可使用socket发送协议，例如实例化协议，我们只需要填写次端姓名即可。

MsgInstance msg = new MsgInstance(clientName);

SendMessage(msg, socketSend);

有了协议，下面是发送，使用socket做TCP/IP连接，需要目标IP地址，即服务器地址，目标程序端口，有了这些之后，使用socket对象的send函数即可发送消息到对方，在具体使用中，有著名的三次握手连接，四次挥手连接，不过这是理论层面的使用，在实际操作中，客户端与服务器端则是通过send，Receive进行消息的收发。

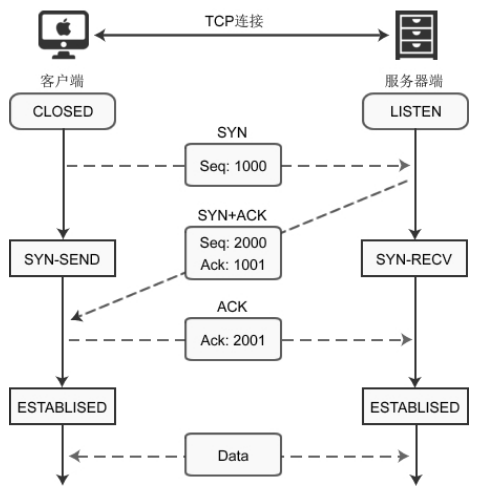


图3.12 TCP连接示意图

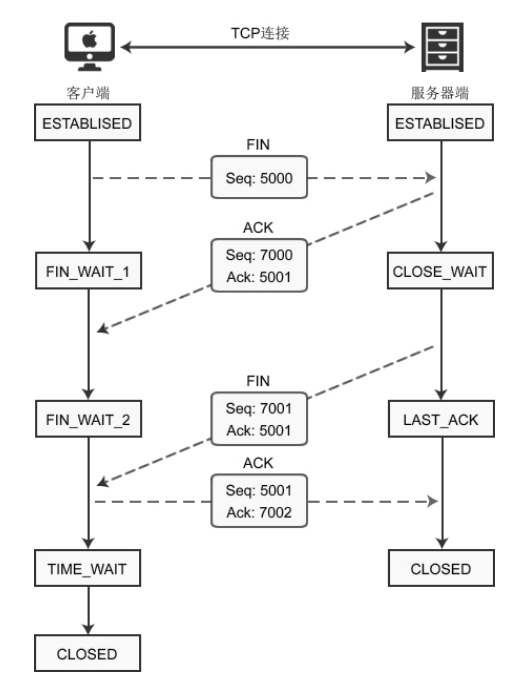


图3.13 TCP断开示意图

消息发送的问题解决之后，那么发送的消息怎么解析呢，网络传输的数据都是二进制数据，是机器码，要想变成可以识别的数据，需要进行消息解析，有很多解析方法，手动解析是最原始的方法，将接收到的二进制根据0和1的数量与位置翻译成对应数据就是手动解析，在电报年代，多采用这种方式，密码本，摩尔斯电码（摩斯密码），都是手动解析带来的产物。

在现代社会，有很多序列化反序列化工具可以使用，序列化与反序列化就是将二进制进行不同的转换，常见的有谷歌的protubuf，Json，Hessian，Thrift等等。每个工具又衍生出很多序列化反序列化库，这里采用LitJson，这就是一个基于Json的库，可以将符合Json条件的内容进行封装发送，接收解析。

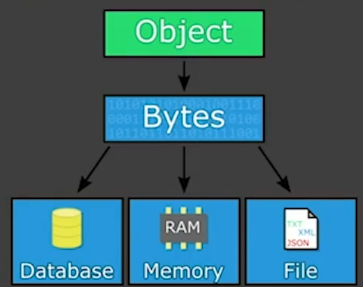


图3.14 数据序列化示意图

/// <summary>

/// 编码协议名

/// </summary>

/// <param name="msgBase"></param>

/// <returns></returns>

public static byte[] EncodeName(MsgBase msgBase)

{//名字和长度

byte[] nameBytes=System.Text.Encoding.UTF8.GetBytes(msgBase.protoName);

Int16 len = (Int16)nameBytes.Length;

//申请Bytes数组

byte[] bytes = new byte[2 + len];

//组装2字节的长度信息

bytes[0] = (byte)(len % 256);

bytes[1] = (byte)(len / 256);

//组装名字bytes

Array.Copy(nameBytes, 0, bytes, 2, len);

return bytes;

}

/// <summary>

/// 编码协议体

/// </summary>

/// <param name="msgBase">协议对象（需要编码的数据）</param>

/// <returns></returns>

public static byte[] Encode(MsgBase msgBase)

{

string s = JsonMapper.ToJson(msgBase);

return System.Text.Encoding.UTF8.GetBytes(s);

}

/// <summary>

/// 解码

/// </summary>

/// <param name="protoName">协议名</param>

/// <param name="bytes">缓冲区</param>

/// <param name="offset">起始位置</param>

/// <param name="count">长度</param>

/// <returns></returns>

public static MsgBase Decode(string protoName, byte[] bytes, int offset, int count)

{

string s = System.Text.Encoding.UTF8.GetString(bytes, offset, count);

Msg Basemsgbase = (MsgBase)JsonMapper.ToObject(s, Type.GetType(protoName));

return msgbase;

}

/// <summary>

/// 解码协议名

/// </summary>

/// <param name="bytes"></param>

/// <param name="offset"></param>

/// <param name="count"></param>

/// <returns></returns>

public static string DecodeName(byte[] bytes, int offset, out int count)

{

//必须大于2字节

if (offset + 2 > bytes.Length)

{

count = 0;

return "";

}

//读取长度

Int16 len = (Int16)((bytes[offset + 1 << 8]) | bytes[offset]);

//长度必须足够

if (offset + 2 + len > bytes.Length)

{

count = 0;

return "";

}

//解析

count = 2 + len;

string name = System.Text.Encoding.UTF8.GetString(bytes, offset + 2, len);

return name;

}

## 3.3 本章小结

在这一章首先介绍了嵌入式端的系统烧录，环境搭建，然后介绍了各个功能的实现方式，最后介绍了网络功能的实现。

# 4. 系统的服务器端设计

## 4.1 引言

服务器在我们生活中随处可见，几乎所有的网络应用都有服务器，而云服务器(Elastic Compute Service, ECS)则是一种简单高效、安全可靠、处理能力可弹性伸缩的服务器，这种服务器不依赖于物理服务器，只需要一个远程控制端即可实现对服务器的操作，维护成本低，使用方便，管理方式比物理[服务器](https://baike.baidu.com/item/%E6%9C%8D%E5%8A%A1%E5%99%A8/100571" \t "https://baike.baidu.com/item/%E4%BA%91%E6%9C%8D%E5%8A%A1%E5%99%A8/_blank)更简单高效。无需购买硬件，即可迅速创建或释放任意云服务器，对于小应用，小体量程序来说，云服务器是最方便，最实惠的选择。

## 4.2 云服务器环境搭建

首先去各大云服务提供商购买一台云服务器，主流云服务器有阿里云，华为云，腾讯云，这里选择腾讯云，腾讯云最便宜，阿里云价格略贵，但是用户量最多，可靠性最高。购买云服务之后，就可以在控制台登录云服务器了，第一次登录需要VNC登录，之后可以采取远程SSH登录。



图4.1 云服务器控制台

配置ssh远程访问服务器，使用命令，sudo /etc/init.d/ssh start 开启ssh服务，update-rc.d ssh enable ，设置ssh开机自启，修改sshd\_config远程root登录，重启ssh，使用命令sudo service ssh restart。

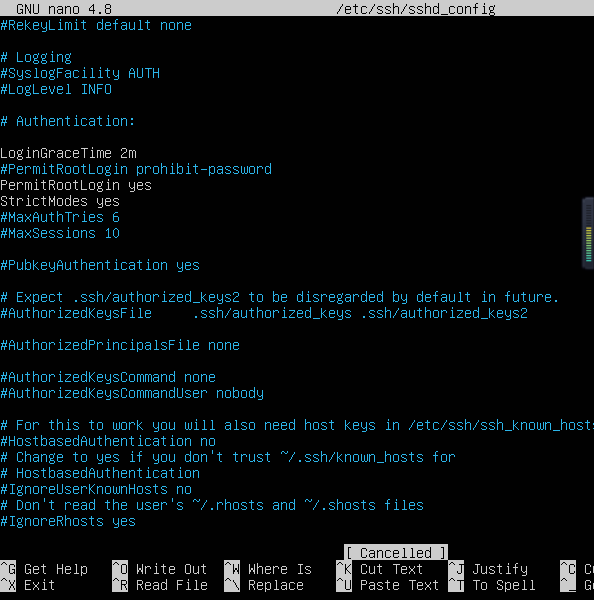


图4.2 ssh配置

配置FTP文件传输，首先开启22端口，使用命令sudo ufw allow 22，开启 FTP服务，使用命令：sudo apt-get install vsftpd，连接失败

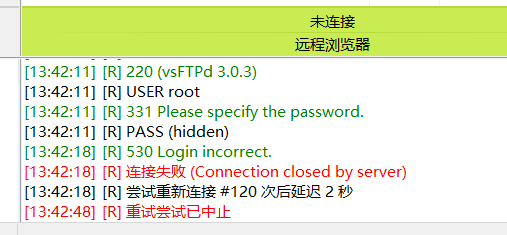


图4.3 FTP连接

和ssh一样，需要配置root登录权限，打开/etc/ftpusers，将root前面的#注释掉，即可对远程服务进行文件传输。

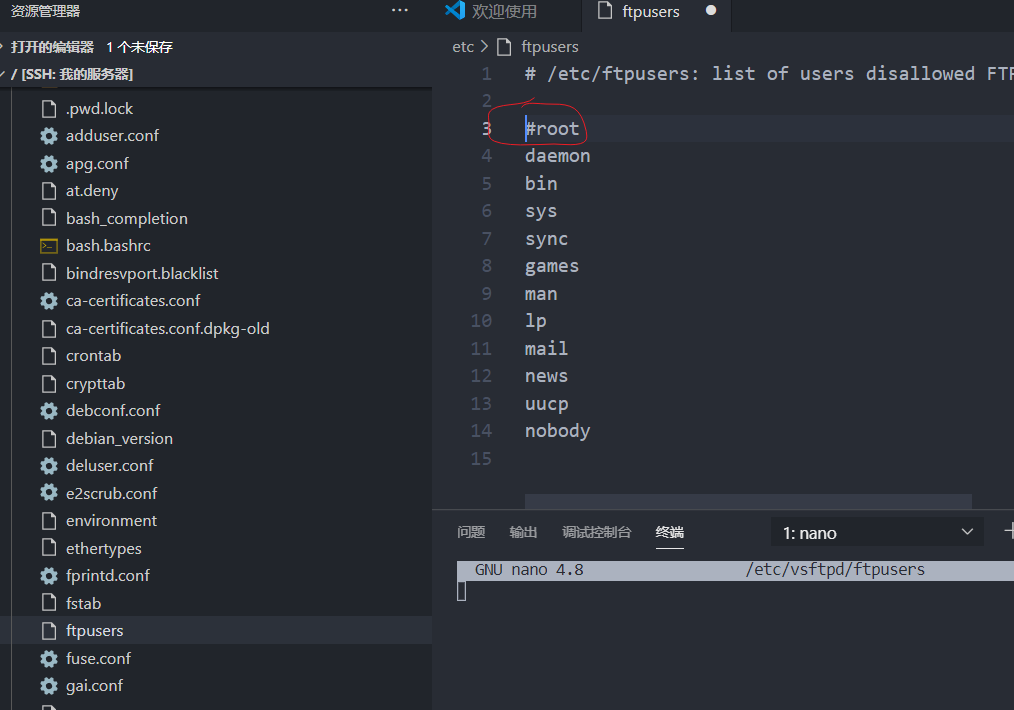


图4.4 登录root修改

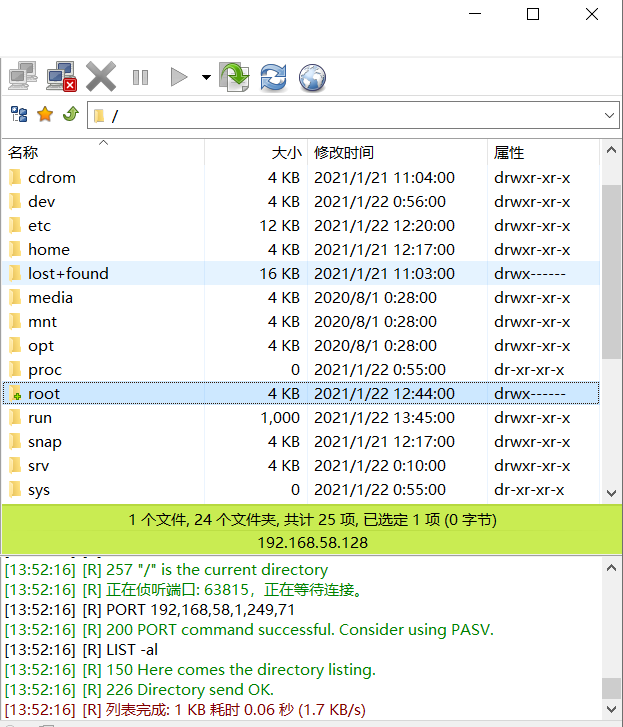


图4.5 FTP连接成功

## 4.3 服务器程序流程图

服务器需要做的是协议分发，所以需要接受协议和识别分发对象，将内容发到指定的客户端或者服务器。



图4.6 服务器程序流程图

### 4.3.1 服务器程序实现

根据流程图，首先将协议文件全部拿过来，因为都是使用C#语言编写，所以三端的通讯协议文件是相同的。

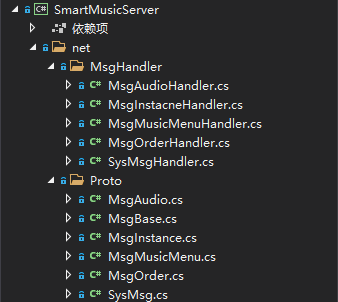


图4.7 消息协议文件

编写网络模块，与嵌入式端不同的是，因为服务器要与多个对象进行交互，所以存在线程占用问题，如果收发网络协议都在主线程进行，则会出现主线程的堵塞与占用。线程是[操作系统](https://baike.baidu.com/item/%E6%93%8D%E4%BD%9C%E7%B3%BB%E7%BB%9F" \t "https://baike.baidu.com/item/%E7%BA%BF%E7%A8%8B/_blank)进行运算[调度](https://baike.baidu.com/item/%E8%B0%83%E5%BA%A6" \t "https://baike.baidu.com/item/%E7%BA%BF%E7%A8%8B/_blank)的最小单位，它被包含于[进程](https://baike.baidu.com/item/%E8%BF%9B%E7%A8%8B" \t "https://baike.baidu.com/item/%E7%BA%BF%E7%A8%8B/_blank)中，是[进程](https://baike.baidu.com/item/%E8%BF%9B%E7%A8%8B" \t "https://baike.baidu.com/item/%E7%BA%BF%E7%A8%8B/_blank)中的实际运作单位。一条线程指[进程](https://baike.baidu.com/item/%E8%BF%9B%E7%A8%8B" \t "https://baike.baidu.com/item/%E7%BA%BF%E7%A8%8B/_blank)中一个单一顺序的控制流，一个进程可以同时运行多个线程，而每条线程并行执行不同的任务。

Main函数所在线程为主线程，是程序执行的入口，服务器为了处理多个对象的网络消息，因此采用多线程方式处理网络，又因为接收消息会堵塞线程，所以采用异步方式接收消息，综上，服务器采用多线程异步处理网络消息。

作为服务器，在主线程监听是否退出服务器，同时监听是否有对象通过TCP/IP连接，如果有对象连接，则开启新的线程给网络对象。

主线程：

private static void Main(string[] args)

{

Console.WriteLine("Hello SmartMusicServer!");

Thread listemThread = new Thread(LisTenNet);

listemThread.IsBackground = true;

listemThread.Start(8888);

Console.WriteLine("按下0，退出服务器");

while (true)

{

ConsoleKeyInfo info = Console.ReadKey();

//主线程

if (info.Key == ConsoleKey.D0)

{

return;

}

}

}

网络监听线程：

public static void StarLoop(int listenPort)

{

//socket

listenfd = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);

//Bind

IPAddress IPAdr = IPAddress.Parse("0.0.0.0");

IPEndPoint ipEp = new IPEndPoint(IPAdr, listenPort);

listenfd.Bind(ipEp);

//Listen

listenfd.Listen(0);

Console.WriteLine("[服务器]启动成功");

//Loop

while (true)

{

ResetCheckRead();//重置checkRead

Socket.Select(checkRead, null, null, 1000);

//检查可读对象

for (int i = checkRead.Count - 1; i >= 0; i--)

{

Socket s = checkRead[i];

if (s == listenfd)

{

ReadListenfd(s);

}

else

{

ReadClientfd(s);

}

}

}

}

接收到消息时候，下一步就是消息分发，服务器采用反射的方式分发消息，反射可以获取运行时各个函数的相关信息，并执行调用。

//通过反射分发消息

MethodInfo mi = typeof(MsgHandler).GetMethod(protoName);

object[] o = { state, msgBase };

if (mi != null)

{

mi.Invoke(null, o);

}

else

{

Console.WriteLine("OnReceiveData invoke fail:mi=null " + protoName);

}

### 4.3.2 服务器程序部署

服务器程序编写完成后，需要部署在云服务器上，首先从微软下载dotnet运行包，然后安装。

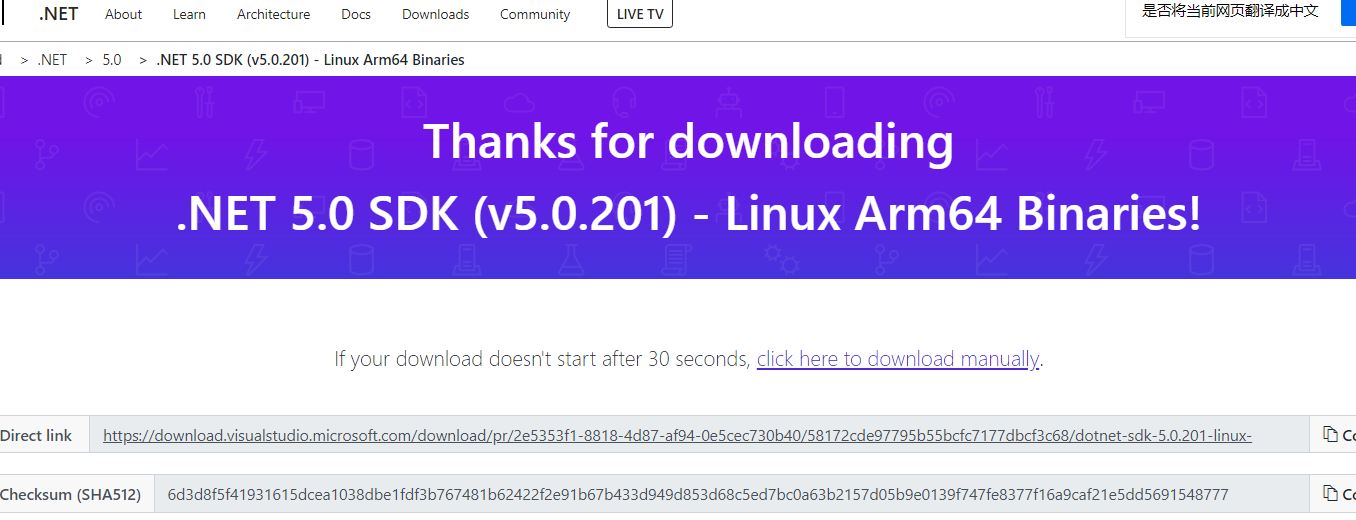


图4.8 dotnet运行包

通过FTP上传服务器程序，进入程序目录，dotnet run即可。

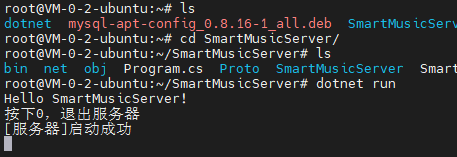


图4.9 服务器启动

### 4.3.3 服务器与嵌入式端通讯测试

服务器程序编写完成后，开始测试与嵌入式端的通讯，打开嵌入式端，发送实例化协议，服务器收到实例化对象，即通讯测试完成。

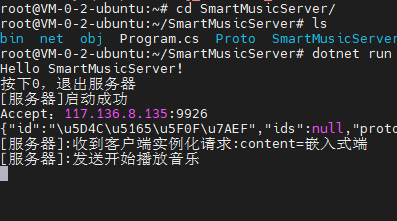


图4.10 服务器发送消息

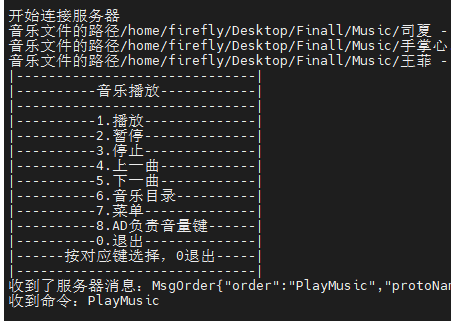


图4.11 嵌入式端收到消息

当退出服务器会发现服务器程序也随之关闭，因为通过ssh打开的程序，属于子进程，父进程是bash终端，当终端关闭，父进程结束，子进程也随之结束，因此需要安装screen

，进入screen运行的程序不会随终端的结束而结束。

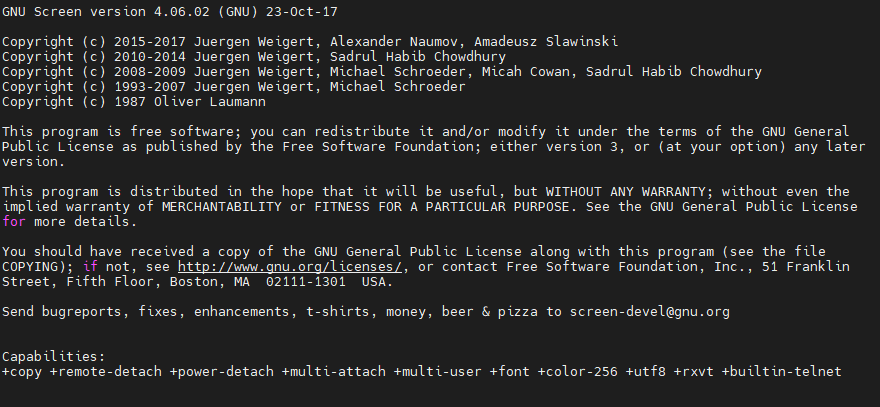


图4.12 Screen界面

## 4.4 本章小结

本章首先对云服务器做了介绍，接着给出了服务器软件的流程图，然后对服务器编写多线程异步程序的过程进行了介绍，最后介绍了服务器程序在云服务器的部署方法，云服务器程序持久运行的策略。通过服务器的开发可以发现，多线程在网络编程中极为重要。

# 系统的客户端设计

## 5.1引言

在客户端方面，我采用Unity配合C#的跨平台方案，Unity 是实时3D互动内容创作和运营的平台 。包括从游戏开发、建筑、汽车设计、影视、APP应用在内的所有内容，都可以借助Unity平台将创意变成现实。

## 5.2客户端UI界面

Unity所见即所得的UI搭建可以快速制作UI界面，根据功能，一共七个按键，分别是播放，暂停，停止，上一曲，下一曲，连接服务器，获取音乐列表，除此之外，还有一个声音滑块，控制声音的大小。

下方有一个白色区域，是负责音乐列表的展示



图5.1 客户端界面

## 5.3客户端程序流程图

客户端首先要连接服务器，想服务器发送实例化请求，告诉服务器我是客户端，然后通过不同按键向服务器发送命令，服务器收到客户端命令，将命令转发给嵌入式端，嵌入式端收到服务器转发的客户端命令，执行对应函数即可。



图5.2 客户端程序流程图

## 5.4客户端程序实现

要实现按钮的功能，Unity的UGUI系统有方便的Button组件，将要执行的函数绑定到对应的Button组件上，即可实现点击按钮发送命令。

程序初始化，按钮功能添加，设置目标IP地址和端口：

private void Start()

{

root = GameObject.Find("Canvas").GetComponent<Transform>();

play = root.Find("Play").GetComponent<Button>();

stop = root.Find("Stop").GetComponent<Button>();

pause = root.Find("Pause").GetComponent<Button>();

downMusic = root.Find("DownMusic").GetComponent<Button>();

upMusic = root.Find("UpMusic").GetComponent<Button>();

connect = root.Find("Connect").GetComponent<Button>();

MusicState = root.Find("MusicState").GetComponent<Text>();

IPAddress1 = root.Find("IPAddress").GetComponent<InputField>();

IPAddress1.text = "121.4.107.99";

MusicList=root.Find("MusicList").GetComponent<Transform>().gameObject; musicListConnect=root.Find("MusicList/MusicListConrent").GetComponent；

MusicListbtn = root.Find("MusicListBtn").GetComponent<Button>();

Volume = root.Find("Volume").GetComponent<Slider>();

play.onClick.AddListener(PlayMusic);

stop.onClick.AddListener(StopMusic);

pause.onClick.AddListener(PauseMusic);

downMusic.onClick.AddListener(DownMusic);

upMusic.onClick.AddListener(UpMusic);

connect.onClick.AddListener(ConnectServer);

MusicListbtn.onClick.AddListener(GetMusicList);

}

声音功能由滑块控制，滑块从左到右会返回一个float，这个值由0到1，对应滑块从最左到最右，这是一个Slider对象。我们每隔0.1秒检测滑块的值是否发生变化，如果发生变化则说明移动了滑块。

if (Mathf.Abs(currentVolume-Volume.value)>=0.1&&time>=timeall)

{

currentVolume = Volume.value;

int curVolume = (int)(Volume.value \* 100);

Debug.Log("现在的音量为：" + curVolume);

MsgAudio msg = new MsgAudio();

msg.audio = curVolume;

NetManager.Send(msg);

time = 0;

}

## 5.5客户端程序测试与编译

客户端程序运行，点击连接服务器，获取音乐列表，可以看到服务受到手机客户端的实例化请求，而服务器受到获取音乐列表的消息后发现，嵌入式端未上线，因此打印出消息嵌入式端未上线。

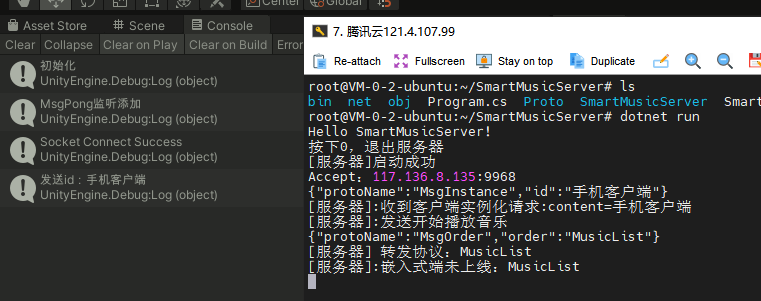


图5.3 服务器收到客户端消息

将客户端编译成可在手机运行的Android程序，打开Build Settings，选择要编译的目标平台，然后点击Build即可完成客户端的编译。

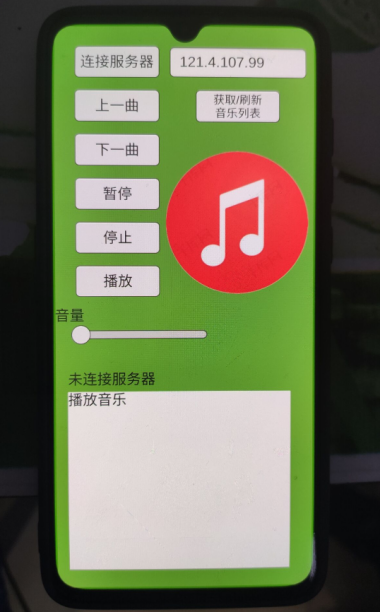


图5.4 客户端编译完成

## 5.6三端通讯测试

客户端完成编译后，现在三端全部完成，需要做三端的通讯测试，首先打开服务器，使用Screen 持久化运行服务器程序。然后打开嵌入式端，嵌入式端连接服务器。最后打开客户端，客户端连接服务器。点击播放，暂停，停止，上一曲，下一曲，滑动音量测试程序是否如期运行。



图5.5 客户端发送命令

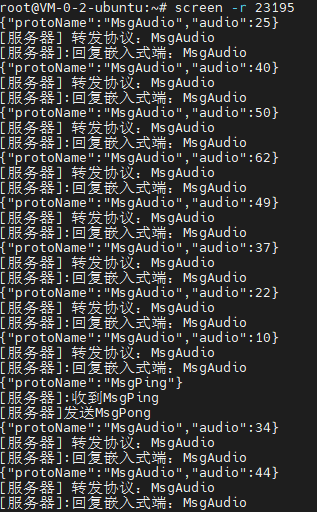


图5.6 服务器端转发命令

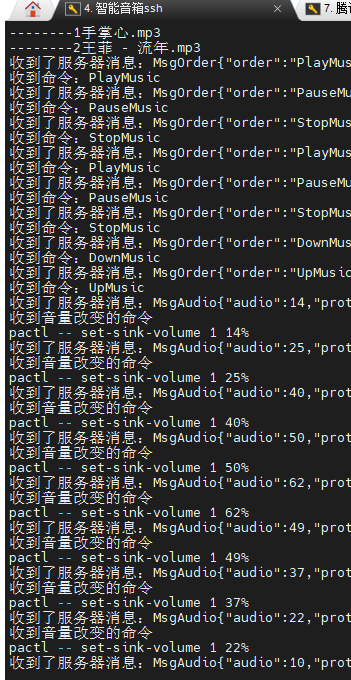


图5.7 嵌入式端收到的命令

## 5.7本章小结

本章首先阐述了Uunity跨平台解决方案是什么，然后详细阐述了如何使用Unity编写客户端程序，其中Button按钮与Slider滑块是实现客户端功能的关键。最后通过三端测试通讯，表明客户端的运行是没有问题的，可以做到将命令发送给服务器，再由服务器转发命令到嵌入式端，做到了不同平台通过网络进行通讯的目的。

# 结束语

经过三个多月的努力，本次毕业设计——基于以太网的智能音箱设计与实现已经完成。从中完整体会到了一个智能音箱从无到有的设计过程，让我受益良多。方案论证和设计仿真的过程，将嵌入式系统，计算机网络等课程再次温习了一遍，受益良多。通过这次设计掌握了从硬件选型，系统烧录，环境搭建，不同平台开发环境的差异等等方面积累了硬件与软件设计开发的经验。

本设计是以RK3288为控制器的智能音箱系统，本文在硬件上，对硬件系统的模块进行了讲解，讲述了不同硬件产品之间的差异，以及较优解。在软件部分则给出了从嵌入式端，服务器端，客户端的三端程序的设计流程图，并且详细描述了其中的重点难点，尤其在嵌入式端，对于非实时操作系统的多线程异步编程有一定参考价值。除此之外，还描述了网络编程的使用，基于Socket，使用TCP/IP传输Json编写的自定义网络协议，，在最后部分，则通过三端程序的切实测试，完成了验证，达到了预期效果，通过手机操作嵌入式端播放自己想要的音乐，实现了播放，暂停，上一曲，下一曲，音量增减功能。

这三个月的学习和设计是我在大学四年中所学知识的整合与提高的过程，今后我将不断加深理论研究，同时增强实践能力，争取在以后的工作和学习中取得更大的进步。

虽然本设计目前已经达到了最开始需求的几个功能，但是仍然存在一些问题。比如没有考虑网络波动的情况，当网路交差时，可能出现粘包现象，使得网络协议读取错误，目前采取的协议头协议体，可以加入消息长度信息，当读取的消息不足消息长度，则在缓冲区等待，足够之后再进行解析，可以有效避免粘包问题。除此之外，还有嵌入式端开启后不能自动启动程序，需要在引导程序区加入自己要启动的程序即可实现开机启动。再次回顾整个设计，我思考后发现其实还可以添加一些功能，比如服务器上传音乐，嵌入式端下载音乐，或者多端上传，嵌入式端根据不同端播放不同音乐等等的功能，都可以提高智能音箱的便捷化程度。

参考文献

[1] W. Richard Stevens，Bill Fenner，Andrew M.Rudoff．UNIX网络编程[M]．北京：人民邮电出版社，2019：112~350

[2] 宋敬彬．Linux网络编程[M]．北京：清华大学出版社，2014：88~122

[3] 段利国．网络编程实用教程[M]. 北京：人民邮电出版社，2016：11~88

[4] 弓雷．ARM嵌入式Linux系统开发详解[M]．北京：清华大学出版社，2014：20~120

[5] [孙天泽](http://search.dangdang.com/?key2=%CB%EF%CC%EC%D4%F3&medium=01&category_path=01.00.00.00.00.00" \t "http://product.dangdang.com/_blank)．嵌入式Linux开发技术[M]．北京：航空航天大学出版社，2011：65~68

[6] Y C. Shen. Auger recombination in InGaN measured by photoluminescence[J]．Appl. Phys, 2007, 91(14): 101~141

[7] 阎映炳,刘雪洁．Linux系统与网络管理教程[M].北京:电子工业出版社,2019.

[8] Dapeng Li, Yongqiang Ye. Phase compensation second-order repetitive control for CVCF PWM inverters[J]. International Journal of Electronics, 2018, 105(2): 324~336

[9] 何进,谢松巍. 基于Socket的TCP/IP网络通讯模式研究[J]. 计算机应用研究，2019，18（9）:54~62

[10] Yangyang Tang, Nghia Tang, Zhiyuan Zhou. PWM-skipping technique for overshoot and undershoot mitigation[J]. International Journal of Electronics Letters, 2017, 5(4): 395~401

[11] 李扬,耿昌宇,张丽芬．基于socket通讯模式下的跨平台数据平台[J].北京理工大学学报，2019，22（1）：75~86

[12] 夏凡，陈蜀宇，龙昌生．基于Framebuffer的嵌入式Linux图形库设计[J].计算机系统应用，2019，21（5）：86~128

[13] 胡耀民,厉伟．Linux系统及网络管理[M].北京:人民邮电出版社,2019.

致谢

首先要感谢的是我的指导老师，卢少武老师。本设计的研究工作是在卢少武老师的精心指导和督促下完成的。从选题到写论文，老师给了我很多帮助，不仅是题目的确定和论文结构的指导，还能在我松懈的时候能督促我抓紧进度。老师在毕设中告诉我们以后不管做什么，都要深入研究的道理一定会谨记。

在此还要感谢我的室友陈世立，在制作实物的过程中，他不仅给我提供了很多材料，在我遇到问题时还会帮我一起检查电路板，最终完成了实物的制作。同时感谢室友戴思铭和冯心钰，在我调试实物时一直配合我，并且在我写论文过程中在格式上和软件使用上给我很大的帮助。

再次感谢所有帮助过我的老师和同学！