西安邮电大学
毕业设计（论文）

题目：基于嵌入式的智能遥控器

学院：自动化

专业：自动化

班级：自动 1404

学生姓名：管宏娟

学号：06141112

导师姓名：张永健/马翔 职称：高级工程师/助理工程师

起止时间：2017 年 12 月 5 日 至 2018 年 6 月 10 日

毕业设计（论文）声明书

本人所提交的毕业论文《基于嵌入式的智能遥控器》是本人在指导教师指导下独立研究、写作的成果，论文中所引用他人的文献、数据、图件、资料均已明确标注；对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式注明并表示感谢。

本人完全理解《西安邮电大学本科毕业设计（论文）管理办法》的各项规定并自愿遵守。

本人深知本声明书的法律责任，违规后果由本人承担。

论文作者签名：

日期：　　年　月　日

西安邮电大学本科毕业设计(论文)选题审批表

<table>
<tr><td>申 报 人</td><td>张永健/马翔</td><td>职 称</td><td>高级工程师/
助理工程师</td><td>学 院</td><td colspan="3">自动化</td></tr>
<tr><td>题目名称</td><td colspan="7">基于嵌入式的智能遥控器</td></tr>
<tr><td>题目来源</td><td>科研</td><td colspan="2"></td><td>教学</td><td></td><td>其它</td><td>√</td></tr>
<tr><td>题目类型</td><td>硬件
设计</td><td>√</td><td>软件
设计</td><td></td><td>论文</td><td></td><td>艺术
作品</td><td></td></tr>
<tr><td>题目性质</td><td colspan="2">应用研究</td><td colspan="2">√</td><td colspan="2">理论研究</td><td></td></tr>
<tr><td>题目
简述</td><td colspan="7">传统家电设备每一台都需要一个红外遥控器,并且不支持远程控制,因此通过实现一个统一的万能遥控器,抛弃繁多的遥控器,并通过自学习丰富遥控码库适配海量设备,具有一定的远程控制功能,让其成为家庭智能控制中心。</td></tr>
<tr><td>对学
生知
识与
能力
要求</td><td colspan="7">掌握微机原理与接口技术及单片机原理及应用,理解嵌入式系统的基本组织结构与工作原理,具有一定的模拟电路技术基础及网络通信原理,熟悉常用传感器原理及使用方法,具有嵌入式软件设计能力及解决实际问题的动手能力。</td></tr>
<tr><td>具体
任务
以及
预期
目标</td><td colspan="7">设计基于嵌入式的万能遥控器, 可远程控制家庭内的各种红外设备,并具有遥控码自学习功能,适配更多的设备.</td></tr>
<tr><td>时间
进度</td><td colspan="7">2017 年 12 月 05 日—2017 年 12 月 10 日 选取毕设题目
2017 年 12 月 11 日—2018 年 01 月 06 日 撰写提交开题报告
2018 年 01 月 07 日—2018 年 03 月 04 日 确定系统架构设计方案
2018 年 03 月 05 日—2018 年 03 月 31 日 系统软件程序设计
2018 年 04 月 01 日—2018 年 04 月 15 日 软硬件的联合调试
2018 年 04 月 16 日—2018 年 04 月 31 日 系统功能完善
2018 年 05 月 01 日—2018 年 05 月 25 日 撰写毕业设计论文
2018 年 05 月 26 日—2018 年 06 月 01 日 修改、装订论文
2018 年 06 月 02 日—2018 年 06 月 10 日 准备毕业答辩</td></tr>
<tr><td>系（教研室）主任签字</td><td colspan="3">年 月 日</td><td>主管院长
签字</td><td colspan="3">年 月 日</td></tr>
</table>

西安邮电大学本科毕业设计（论文）开题报告

学生姓名	管宏娟	学号	06141112	专业班级	自动 1404
指导教师	张永健/马翔	题目	基于嵌入式的智能遥控器		

选题目的（为什么选该题）

最近几年来，科学技术随着经济的迅猛发展而日新月异，人们的生活水平也随之有了很大的提高。各式各样的家用电器进入到普通老百姓的日常生活中，遥控器也已成了普通百姓家庭中必须的装置。由于红外线遥控器拥有很多的优点，所以绝大多数家用电器都是采用这种遥控方式的。但是，这些不同品牌的遥控器之间不能互相的替代使用，这给人们的生活带来了诸多的不便。为了解决这个问题，减少家用电器遥控器的数量，为人们的生活带来更大的便利，一种智能型的红外线遥控器被许多的厂家设计并生产出来了，并且称之为智能遥控器，由于科技的进步与发展，自动化、智能化日益成为一种发展趋势,我们未来家庭的日常生活必然与智能化密切相关,因此选择此课题也是研究我们未来的一种生活的方式,是让智能化走进我们日常生活一种探索。

前期基础（已学课程、掌握的工具，资料积累、软硬件条件等）

1、 计算机网络：网络通信协议
2、 红外协议：了解遥控器的基带通信协议（NEC 协议）
3、 Python 和 Django：建立 Web，远程控制万能遥控器
4、 lirc：将红外信号转化为指令编码
5、 可编程控制器：红外接收、红外发射

要研究和解决的问题（做什么）

通过使用可编程控制器实现一种可通过自学习来丰富遥控码库并且可以远程控制的万能遥控器

工作思路和方案（怎么做）

按键：

1、 按下学习键，开始接收红外信号，lirc 将红外信号转为指令编码
2、 按下编号键，将学习到的指令编码和编号存数据库中
3、 按键模式下，一个编号键只能存一个红外指令，重复使用同一编号存储红外指令将覆盖原有指令

Web：

1、 自命名保存，同一家用电器的红外指令存在数据库中与自己定义的分辨不同家电名字有关系
2、 在 Web 端选择家电名字，进入使用界面，不同操作会触发不同动作，将关联的红外指令通过可编程控制器发出

指导教师意见

签字：　　年　月　日

西安邮电大学毕业设计（论文）成绩评定表

<table>
<tr><td>学生姓名</td><td>管宏娟</td><td>性别</td><td>女</td><td>学号</td><td>06141112</td><td>专业
班级</td><td>自动 1404 班</td></tr>
<tr><td>课题名称</td><td colspan="7">基于嵌入式的智能遥控器</td></tr>
<tr><td>指导
教师
意见</td><td colspan="7">评分（百分制）：　　指导教师(签字)：_____　　年　月　日</td></tr>
<tr><td>评阅
教师
意见</td><td colspan="7">评分（百分制）：　　评阅教师(签字)：_____　　年　月　日</td></tr>
<tr><td>验收
小组
意见</td><td colspan="7">评分（百分制）：　　验收教师(签字)：_____　　年　月　日</td></tr>
<tr><td>答辩
小组
意见</td><td colspan="7">评分（百分制）：　　答辩小组组长(签字)：_____　　年　月　日</td></tr>
<tr><td>评分比例</td><td colspan="7">指导教师评分　(%) 评阅教师评分　(%)　验收小组评分　(%)　答辩小组评分　(%)</td></tr>
<tr><td>学生总评
成绩</td><td colspan="2">百分制成绩</td><td></td><td colspan="2">等级制成绩</td><td colspan="2"></td></tr>
<tr><td>答辩委员
会意见</td><td colspan="7">毕业论文(设计)最终成绩(等级)：

学院答辩委员会主任(签字、学院盖章)：　　年　月　日</td></tr>
</table>

目　录

摘　要

近年来，随着人们生活质量不断提高以及科学技术的突飞猛进，智能家居的发展日益成为一种趋势，红外遥控仍然是现今家用电器的主流控制方式，但一对一的遥控和完全键盘的操作方式已经不能满足现代家庭的家电控制需求，对可以存储多种遥控器代码或者能进行代码学习并且能够远程控制家电的多功能遥控器的研究一直以来是遥控器开发的趋势。为进一步满足现代家庭的家电控制需求，与嵌入式 Linux 系统结合，开发具有自学习红外遥控代码功能带有网页版遥控器具有一定的实用意义。 通过对遥控器的深入研究，提出了基于嵌入式 Linux 的智能遥控器系统设计方案。系统硬件平台采用 ARM 体系结构的树莓派开发板，根据系统需求设计了学习电路和遥控电路。系统软件平台采用嵌入式 Linux 系统，设计了远程遥控家电的网页功能。 在本课题的软硬件环境下，对各相关程序进行了必要的测试分析。测试结果表明：系统与主机通信顺利；红外驱动在系统开启时可以正确有效的驱动红外器件；红外代码学习和遥控功能都能正确实现，界面相对友好。系统还可以根据需要对软件进行二次开发，增加其他的娱乐功能，市场应用前景较好。

关键词：树莓派；红外遥控；远程遥控

ABSTRACT

In recent years, with the continuous improvement of people's quality of life and the rapid advancement of science and technology, the development of smart homes has become a trend. Infrared remote control is still the mainstream control method of current home appliances, but one-on-one remote control and full keyboard operation Already unable to meet the home appliance control demand of the modern family, the research on the multi-function remote control that can store a variety of remote control codes or can learn codes and can remotely control home appliances has been the trend of remote control development. In order to further meet the modern home appliance control requirements, combined with the embedded Linux system, the development of self-learning infrared remote control code function with a Web version of the remote control has certain practical significance. Through the in-depth study of the remote control, an intelligent remote control system based on embedded Linux is proposed. The system hardware platform adopts the Raspberry Pi development board of the ARM architecture, and the learning circuit and the remote control circuit are designed according to the system requirements. The system software platform uses an embedded Linux system to design Web functions for remote-controlled home appliances. In the software and hardware environment of this subject, necessary test analysis was performed on each related program. The test results show that the system communicates well with the host; the infrared drive can drive the infrared device correctly and effectively when the system is turned on; infrared code learning and remote-control functions can be correctly implemented, and the interface is relatively friendly. The system can also be used for secondary development of the software as needed, adding other entertainment functions, and the market has a good application prospect.

Key words： Raspberry Pi; Infrared remote control; Remote control

第一章　引言

1.1 课题背景

在科学技术产业的蓬勃发展的基础上，智能化发展逐渐成为一种新常态，而人们对于生活质量的重视程度也在不断提高，因此家居的智能化逐渐进入了人们的生活当中，而通用遥控器的出现更方便了人们的日常。当人们在大冬天顶着极大的压力辛辛苦苦完成高强度的工作后，下班路上冒着风雪一路吹着冷风，冻得瑟瑟发抖的时候，肯定会想早早回去就能进入暖和而又舒适的房间，此时只需要拿着手机登录进入万能遥控器的远程遥控系统，然后打开空调、调好温度，即可一回到家就能感受到家的温暖；当人们在晚上看电视时有些犯困就直接去卧室上床睡觉，又想起来电视忘记关机却懒得下床的时候，万能遥控器又能帮助用户解决这普通遥控器根本不可能解决的一大难题；当家里各种电器设备的遥控器太多，有些遥控器在外观上还很相似导致人们总是拿错的时候，还有一些家电的遥控器根本找不到的时候，此时此刻只需要一个万能遥控器即可，根本不需要如此多的遥控器，不但占地方，还容易混淆，一个万能遥控器便能轻松帮人们解决这些烦恼。

我国目前以及未来的研发方向就是远程遥控，这就解决了当即使用户离家电设备很远，也可以在较远的距离外想要启动和关闭的时候对其进行控制的问题。

虽然市面上设计的万能遥控器和家里的普通遥控器在外形设计上是较为相似的，但它可以进行双向通信，并在设计层次上和常见的嵌入式结构非常类似，主要包含嵌入式芯片、嵌入式操作系统和嵌入式程序等，不过在嵌入式操作系统里面加入了 Web 系统，即一个页面支持，主要便于进行远程控制。本文设计的遥控器能够完成两个重要功能，一是具有自学习能力，可适配海量设备，二是能够进行远程控制。

本课题主要运用了 Python、SQLite3、Redis、HTML、Django 框架以及 Raspberry Pi 的 GPIO 编程、Ubuntu 系统以及 lirc 等技术，设计了可以通过自学习功能来适配不同型号不同品牌的家电设备并且可以实现远程遥控的万能遥控器，解决了家中多个家用电器只能使用配套的遥控器的问题，也满足了高速发展的科技下新时代人们对于更加智能、方便、快捷、舒适的生活需求。

1.2 课题任务

本课题所研究的遥控器主要由红外遥控码接收模块、遥控按键控制模块、用户登录模块以及红外遥控码发射模块这几部分构成。其中红外遥控码接收模块以及红外遥控码发射模块用的是一块红外拓展板，此拓展板上集成了红外接收器和红外发射器，并且还有两个可用的按键，分别对应 GPIO27 和 GPIO18。红外接

收器实际上采用一个特殊的三极管，而红外发射器自身也是一个特殊的能够发光的二极管；遥控按键控制模块主要采用了两块 4*4 矩阵键盘；用户登录模块是用来让用户能够对家用电器进行远程控制，整个系统主要是利用 Raspberry Pi 完成的。Raspberry Pi 是一款拥有 ARM 核、计算速度快、轻便小巧的计算机主板，它的主要优势是拥有一颗 ARM11 系列的、运算频率为 700MHz 的 CPU 芯片。Raspberry Pi 和普通电脑一样拥有硬盘，它仅需要一张 SD 卡就能充当硬盘，树莓派板子上包括了四个 USB 接口，同时还有音频和 HDMI 输出口，能够处理音频、视频等文件，还有一个 RJ-45 接口，可以用来连接网线。由于 Raspberry Pi 不但具有能耗低、便携性强、GPIO 口多等特性，并且可以安装各种 Linux 的发行版本，而要想实现远程控制则需要搭建 Web，所以用树莓派来打造基于嵌入式的万能遥控器也就比较方便。

本设计主要使用了 lirc 软件，它的英文意思是基于 Linux 的红外线遥控，当用户需要遥控家电设备时，只需要按下遥控键盘的按键，后台就会判断出按的是哪个按键，根据不同的按键，lirc 会根据不同的键值查找相应的红外编码，并且将其发出，这样使用者就可以对需要的家用电器进行控制了。

1.3 论文结构

本论文的章节安排如下：

第一章：主要介绍课题提出的背景以及需要完成的任务；

第二章：主要讲本系统中所涉及的主要技术。

第三章：主要介绍万能遥控器的功能与设计方案；

第四章：主要描述在硬件方面的实现；

第五章：主要描述远程遥控功能的实现；

第六章：总结；

第七章：表达作者对基于嵌入式的万能遥控器未来的期待。

第二章　基于嵌入式的万能遥控器相关技术

本章节主要介绍本系统所用到的一些主要技术，如 Python、SQLite3、Redis、HTML、Django 框架以及 Raspberry Pi 的 GPIO 编程、Ubuntu 系统以及 lirc。

2.1 Python 介绍

2.1.1 Python 简介

Python 是一种简洁并且能简单上手的非编译语言，大部分人也叫它为脚本语言，它没有编译这个环节，和 PHP、Shell 以及 Perl 语言有些许相似之处。

使用 Python 语言对一个函数的设计，或者是作为面向对象的设计，它都具有非常强的可读性，相比于类似 Java、C++等语言中一些英文关键字的设计，Python 语言以其独特的标点符号、命名方式、对齐方式都为人所赞美。虽然 Python 语言的运算速度与一些其他语言相比较慢，但是其依靠各种强大的内置库和丰富的第三方库都让 Python 变得越来越火热，尤其是现在机器学习库的发布，更是让 Python 隐隐约约有超过 Java 使用量的趋势。

2.1.2 Python 特点

Python 的主要特点是：上手简单、方便维护和阅读、丰富的第三方库以及跨平台兼容、各种数据库支持。

2.2 SQLite3 介绍

SQLite3 不是一种重量级的数据库，可以支持标准的 SQL 语句。Sqlite3 与其他主流数据库一样，也支持事务操作，但是却没有像其他数据库一样连接繁琐，它是不需要依赖服务器存在的、不需要进行繁琐配置的一种可以实现自给自足的轻量级数据库。

SQLite3 不需要配置，也就意味着它根本就不用进行安装，无论是 Linux 系统还是 Windows 系统，都自带了 SQLite3 的环境，并且如果要用 Python 使用 SQLite3，只需要安装 SQLite3 的 Python 库，即可简单使用 API 进行事务的 ACID 操作。

2.3 Redis 介绍

Redis 和 Mango DB 都是非关系型的，但是 Redis 的主要目的并不是用来作数据库的。Redis 是基于内存的，不需要进行磁盘 IO 的读写，所以查询速度非常快，因此 Redis 一般用于一些查询密集的设备用作内存缓存，并且 Redis 是采用一个键对应一个键值来存储的，所以不管是在线程中或者是不同的会话中，都能获取到设置的值，这样就可以用来做在线和离线的一种交互。

Redis 支持开发者自定义每一个 key 的超时时间，当时间到了之后，Redis

会自动删除该键以及该键对应的键值，这样可以用来实现做验证码的功能。

同时，Redis 在数据的操作中都是原子操作，不会发生线程安全问题。

Redis 具有非常多的数据类型操作，如字符串、列表、集合、哈希值、有序集合这五种，其中 Sorted Set（有序集合）是一个没有重复并且每个元素带有一个 Score 的数据类型，可以很方便的做一个热榜排行的功能。

2.4 HTML 介绍

开发人员可以在 HTML 内部使用各种标记标签来描述你的网页设计。但是直接这样设计的网页并不整洁，也不美观，所以开发人员可以在编写了 HTML 代码后使用 CSS 来美化用户的页面，使页面主题更加的突出，而在需要进行前后端交互的时候，开发者可以使用 JS 代码完成，它是由客户端执行的，用来控制前端中的各种逻辑。

2.5 Django 框架介绍

Django 是一个用来开发网页应用的框架，底层基于 Python 语言，支持便捷高效的开发和设计。Django 是由许多杰出的程序开发者维护的一个开源并且免费的、优秀的框架，由于它可以处理开发人员在 Web 建站中遇到的大部分问题，因此开发人员可以更加专注地关注于自身程序代码逻辑，而并不需要闭门造车。

快速：Django 旨在快速的为开发人员搭建一个 Web 应用程序。

安全：防止 SQL 注入，帮助使用 Django 的程序员避免了很多常见的安全错误。

实时性：可以在服务运行时修改代码，并自动热部署。

2.6 Raspberry Pi 的 GPIO 编程

Raspberry Pi 是一种非常轻巧的便携式电脑，虽然体积比较小，但是功能十分齐全。Raspberry Pi 的引脚包括两个 5V 引脚、两个 3.3V 引脚、8 个 GND 引脚以及 28 个可以控制的 GPIO 引脚。Raspberry Pi 实物图由图 2.1 所示，引脚对照表如图 2.2 所示。

图 2.1　Raspberry Pi 实物图

BCM	wPi	Name	Mode	V	Physical	V	Mode	Name	wPi	BCM
		3.3v			1 || 2			5v		
2	8	SDA.1	IN	1	3 || 4			5V		
3	9	SCL.1	IN	1	5 || 6			0v		
4	7	GPIO. 7	IN	1	7 || 8	1	ALT0	TxD	15	14
		0v			9 || 10	1	ALT0	RxD	16	15
17	0	GPIO. 0	OUT	0	11 || 12	0	IN	GPIO. 1	1	18
27	2	GPIO. 2	IN	0	13 || 14			0v		
22	3	GPIO. 3	IN	0	15 || 16	0	IN	GPIO. 4	4	23
		3.3v			17 || 18	0	IN	GPIO. 5	5	24
10	12	MOSI	ALT0	0	19 || 20			0v		
9	13	MISO	ALT0	0	21 || 22	0	IN	GPIO. 6	6	25
11	14	SCLK	ALT0	0	23 || 24	1	OUT	CE0	10	8
		0v			25 || 26	1	OUT	CE1	11	7
0	30	SDA.0	IN	1	27 || 28	1	IN	SCL.0	31	1
5	21	GPIO.21	IN	1	29 || 30			0v		
6	22	GPIO.22	IN	1	31 || 32	0	IN	GPIO.26	26	12
13	23	GPIO.23	IN	0	33 || 34			0v		
19	24	GPIO.24	IN	0	35 || 36	0	IN	GPIO.27	27	16
26	25	GPIO.25	IN	0	37 || 38	0	IN	GPIO.28	28	20
		0v			39 || 40	0	IN	GPIO.29	29	21
BCM	wPi	Name	Mode	V	Physical	V	Mode	Name	wPi	BCM

图 2.2　Raspberry Pi 引脚对照

Raspberry Pi 的引脚图及 GPIO 整体电路图分别如图 2.3 和图 2.4 所示。

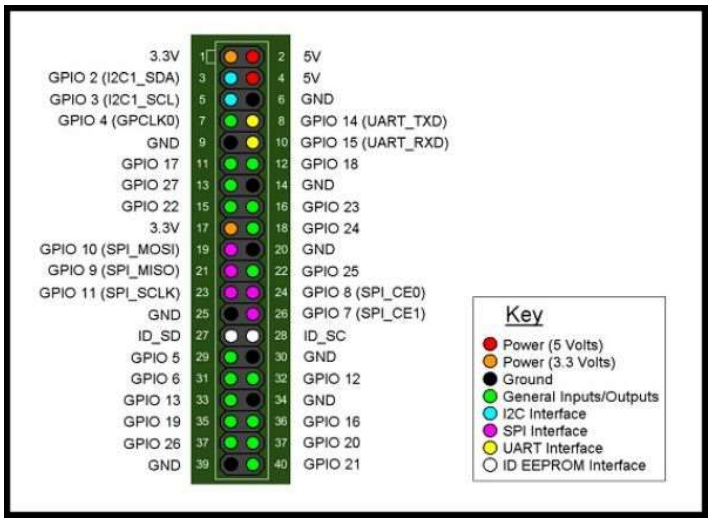

图 2.3　Raspberry Pi 整体引脚图

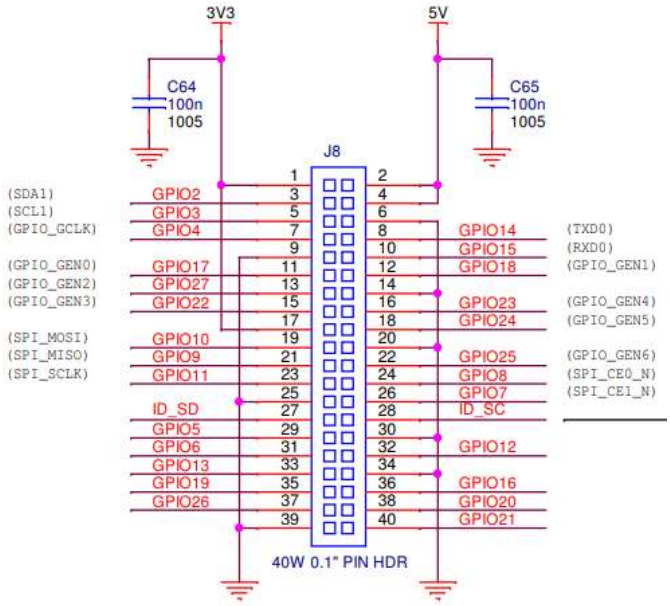

图 2.4　Raspberry Pi 的 GPIO 引脚电路图

Raspberry Pi 的复位硬启动电路如图 2.5 所示。

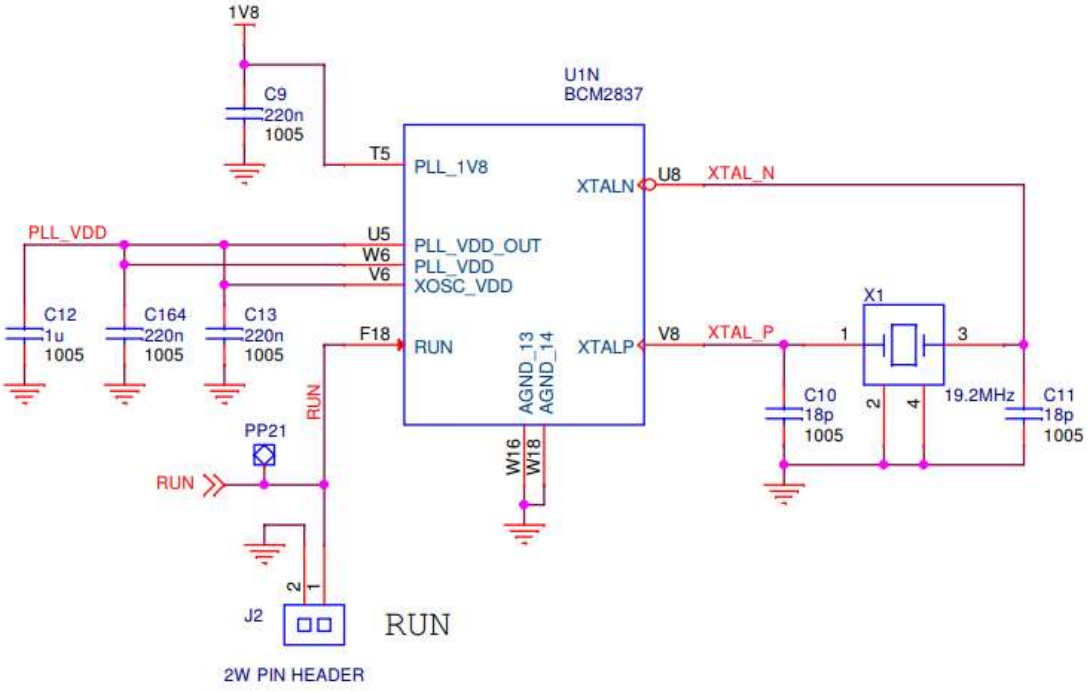

图 2.5　Raspberry Pi 的复位电路图

Raspberry Pi 底层规定其引脚工作方式包括以下两种，一种是 Board，这种工作方式将使用 Board 编号将引脚接入电路，不用考虑不同 Raspberry Pi 不同版本中每个引脚代表不同功能，只与板子中的引脚编号有关；一种是采用 BCM 工作方式，当采用 BCM 方式时，接入的引脚必须是 BCM 编号对应的 Board 编号。

Python 程序能够方便的设置 Raspberry Pi 的 GPIO 口的输入或者输出情况。当 Python 程序设置 Raspberry 中某个 GPIO 为输入状态时，可以自行焊接上拉电阻来解决输入为 float 类型，也可以通过软件设置上拉状态。当一个 GPIO 口被定义成输出状态时，可以由 Python 程序控制输出高低状态，同时也可以通过软件来读取此时输出状态是什么。

Raspberry Pi 的 GPIO 管脚设置为输入时，可以在代码中为该引脚注册一个中断，当电平变化时触发中断，执行相应函数。

2.7 lirc

lirc 是一个包，允许用户解码和发送许多（但不是所有）常用遥控器的红外信号。

最近的 Linux 内核可以将一些红外遥控器用作常规输入设备。有时候这会让 lirc 变得多余，但是 lirc 提供了更多的灵活性和功能，并且在许多场景中仍然是正确的工具。

lirc 的底层实现是 lircd 守护进程，该守护进程主要实现了对使用者给它发送的红外信号进行解码操作，并在套接字上提供信息。如果硬件支持，它还接受要发送的 IR 信号的命令。

lirc 允许使用者遥控控制电脑。只需按一下按钮，即可将 X11 事件发送到应用程序，启动程序等等。可能的应用是显而易见的：红外鼠标，电视调谐卡或 CD-ROM 遥控器，远程关机，用计算机编程 VCR 和/或卫星调谐器等，在 Raspberry Pi 上安装使用 lirc 已经很平常。

2.7.1 安装 lirc

在 Raspberry Pi 下很容易就能完成 lirc 的安装，只需要在终端执行图 2.6 的命令。

```
sudo apt-get install lirc
```

图 2.6　linux 安装 lirc

安装完之后执行图 2.7 的命令：

```
sudo leafpad /etc/lirc/hardware.conf
```

图 2.7　leafpad 打开 lirc 配置文件

修改成图 2.8 的内容：

```
LIRCD_ATGS=""
DRIVER="default"
DEVICE="/dev/lirc0"
MODULES="lirc-rpi"
```

图 2.8　更改 lirc 的相关信息

2.7.2 配置 lirc 引脚

lirc 需要配合红外接收器和红外发射模块使用的，因此用户需要在 Raspberry Pi 中配置红外拓展板中红外接收和发送引脚。

执行如图 2.9 所示命令，就可以打开 Raspberry Pi 的模块定义文件：

```
sudo leafpad /etc/modules
```

图 2.9　打开 Raspberry Pi 模块文件

添加如图 2.10 里面的内容：

```
lirc-dev
lirc-rpi gpio_in_pin=18 gpio_out_pin=17
# 采用的是 BCM 模式
```

图 2.10　为 lirc 添加相关信息

保存配置后，重启 Raspberry Pi 才能使得其配置生效。要使用 lirc 录制红外，则只需在终端输入如图 2.11 的命令。其中的 name 参数是为你接下来录制的设备取一个名字，名字可以是任意的，本课题采用的是通过 UUID 来为每一个设备命名，以防止学习的重复。

```
sudo /etc/init.d/lirc stop
sudo irrecord -d /dev/lirc0 name
```

图 2.11　开始使用 lirc 录制红外

录制开始时，lirc 要求使用者必须拿要学习的遥控器按下该遥控器的每一个按键，每一个至少按一次，一共需要按满 80 次，lirc 通过接收到的信号来计算该遥控器的 gap 值，计算完毕后，才能正式开始学习功能。学习期间，lirc 要求使用者必须键入 lirc 库里面存在的按键值，输入完之后，才接收对应遥控器的按键的红外信号。当使用者不再进行学习时，按回车即可退出学习。

学习完 lirc 会将学习到的按键信息以文本文件保存在当前目录，文件名为使用者录制时设定的 name 参数。但是现在还不能使用刚学习完的成果，使用者还需要将这个文件复制到/etc/lirc/lircd.conf 中才能使用。使用 lirc 发射红外指令的命令如图 2.12 所示。

```
sudo irsend “SEND_ONCE” name key_name
```

图 2.12　使用 lirc 发送红外指令

在该命令中，name 代表的是 lirc 录制时的 name 参数，key_name 代表的时使用者为每个学习到的按键取的名字。

第三章　基于嵌入式的万能遥控器的设计与功能

本章节主要是较为概略的讲述整个系统的基本情况如系统的模块框架、系统的开发流程图以及系统的设计方案，具体的硬件内容实现和软件内容实现将在后面章节中一一讲述。

3.1 整体架构

基于嵌入式的万能遥控器由 Web 远程端和硬件端两部分构成，其中软件部分有：信息验证模块（登录模块、登出模块、修改密码模块）、信息展示模块（远程遥控器模块、学习完后的遥控器在 Web 端命名模块）以及远程遥控模块（通过网络请求来远程遥控），硬件组成为：遥控器学习模块（键盘扫描模块、红外指令存储模块）以及遥控器的使用模块（查询当前遥控器属于哪个设备模块、发送红外指令模块）。其整体架构如图 3.1 所示：

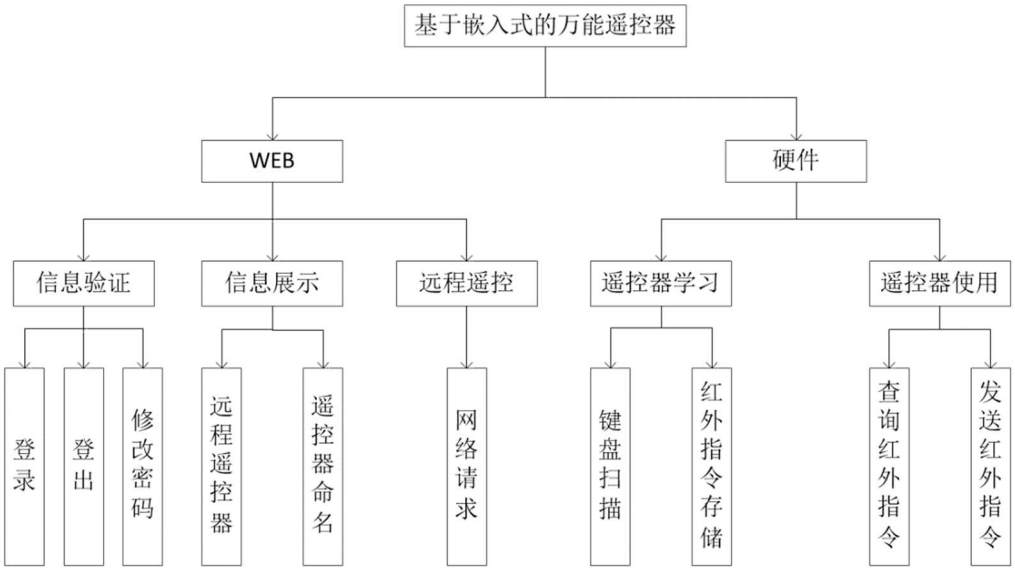

图 3.1　基于嵌入式的万能遥控器整体架构图

远程遥控系统通过登录模块进行用户名密码验证，验证通过后将可进入系统，进入系统后可以控制远程设备，通过设计一个虚拟遥控器，按每一个按键都会触发一个网络请求，后台会将对应的红外编码发送出去。在开发板中，编写了键盘扫描程序，不同的按键代表了不同的信息，用户按下学习键即可开始学习一台新的设备。

3.2 整体功能

基于嵌入式的万能遥控器使用 lirc 和红外拓展板发射和接收红外指令，在学习功能中，会在 SQLite 数据库保存每一次学习一个新的遥控时学习了哪些按键。

远程遥控使用 Django 搭建一个网站，使用者可以通过登录网站进入到远程遥控系统，并选择想要遥控的设备，点击按钮即可远程遥控设备。基于嵌入式的远程遥控器功能示意图如图 3.2 所示。

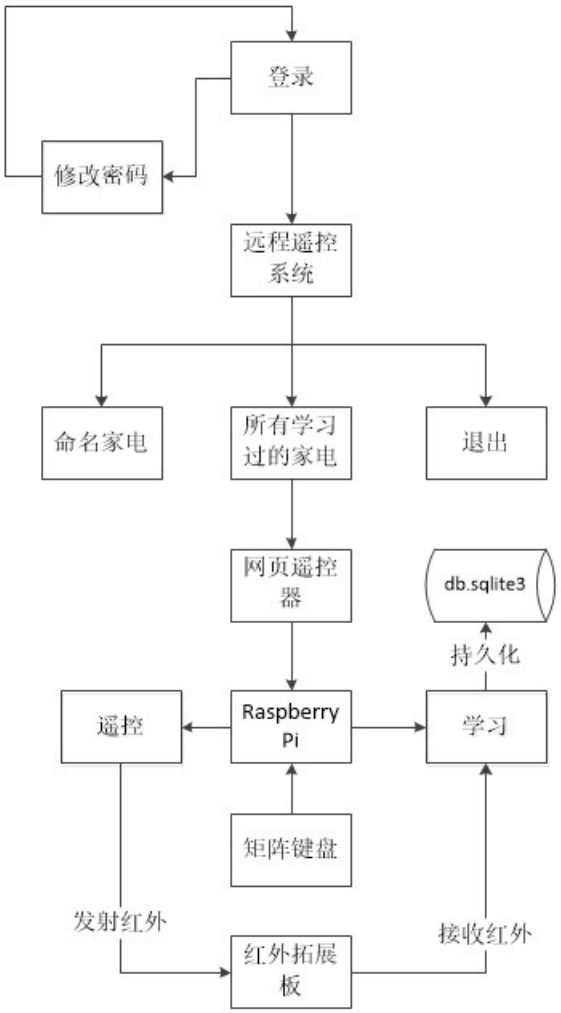

图 3.2　万能遥控器系统功能流程图

使用远程遥控设备需要进行登录，避免被人随意使用。在学习完成一个新的设备时需要对学习过的设备产生的文本文件的路径以及学习了的按键进行持久化操作。

本课题采取 SQLite3 作为数据库，数据库的名字为 db.sqlite3，它是 Web 部分与硬件交互的桥梁。硬件将学习完成的情况持久化到数据库中，Web 部分从数据库中读取，以实现远程功能。

3.3 设计原则

低功耗设计：整个系统以树莓派作为开发板，仅需 USB 供电，并且其余部

分也是选用的功耗低且可控功耗的器件。

模块化设计：整个系统以分模块设计原则，每个模块单独实现功能再整合成完整系统。每个模块下面又分了很多小功能，每个小功能也尽量是实现高内聚低耦合方式，所以不会产生重复开发的情况，让系统变得简洁、高效。

能优化则优化：整个系统的开发过程中，不管是软件还是硬件，对于能优化的部分都进行了优化，比如说开发中硬件的绝大部分都是采用杜邦线连接，而不是在电路板中焊接，减少了电路与电路之间的干扰，使得系统稳定性以及抗干扰能力都在很大程度上有所增强。

第四章　硬件设计

此章节将详细的描述整个硬件的部分，包括整个硬件电路的设计、红外模块电路的设计、按键模块的设计以及指示灯模块电路设计。

4.1 硬件整体方案

尽管硬件是模块化设计的，但是每个模块之间都互相建立了联系。整个系统以 Raspberry Pi 作为主控，用 2 个 4*4 的矩阵键盘作为遥控键值的输入，以红外拓展板作为红外信号的接收和发射，Raspberry Pi 的 GPIO 编程作为整个业务的逻辑判断，设计图如图 4.1 所示。

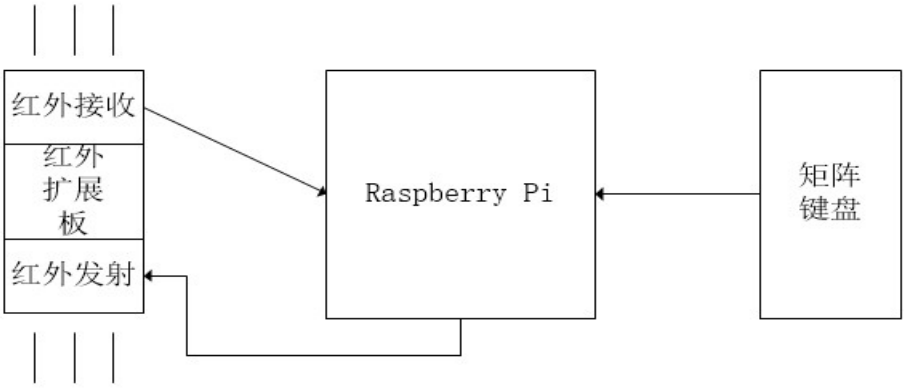

图 4.1　嵌入式万能遥控器硬件设计

4.2 红外拓展板与 Raspberry Pi

4.2.1 红外拓展板

红外拓展实物图如图 4.2 所示。

图 4.2　红外拓展板

这个红外拓展板是针对 Raspberry Pi 的专用红外控制拓展板，用户可以使用该拓展板配合 Raspberry Pi 完成各种遥控和被遥控的功能。红外拓展板使用 5v 及 3.3v 配合供电，红外接收管脚连 Raspberry Pi 的 GPIO18（BCM），红外发射管脚接 Raspberry Pi 的 GPIO17（BCM）。这个红外拓展板的工作特点是：

（1）红外线接收特点：工作的频率为 38KHZ，接收的距离是 18-20m；

（2）红外线发射特点：波长是 940nm，发射的距离为 7-8m；

（3）可以配合 Raspberry Pi 进行 lirc 的使用。

红外拓展板中红外接收器电路如图 4.3 所示。

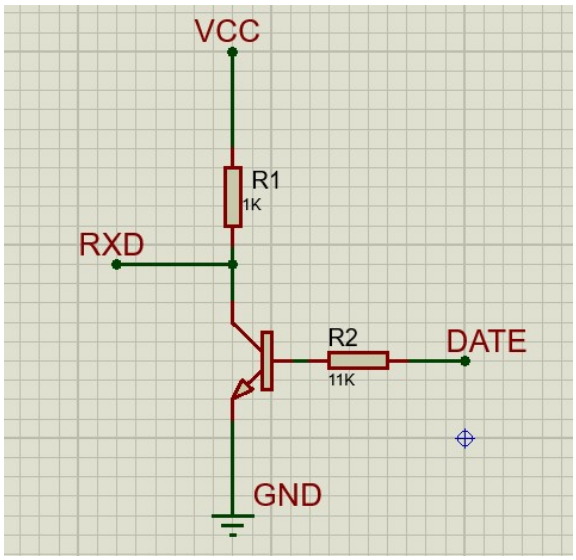

图 4.3　红外接收电路

红外拓展板中红外发射器的电路如图 4.4 所示。

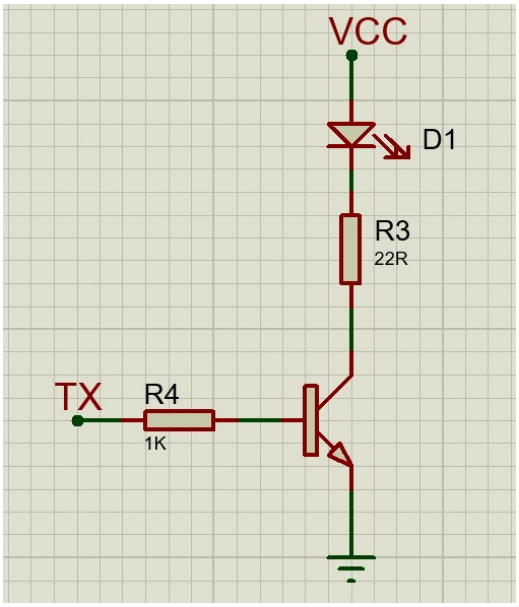

图 4.4　红外发射电路

4.2.2 NEC 红外协议

现在市面上的红外遥控器基本都是基于 NEC 协议的。NEC 的载波为 38Khz，协议规定：Logic"1"时间为 2.25 ms，Logic"0"时间为 1.12ms，其中不管是 Logic"1"还是 Logic"0"脉冲时间都为 560us。如图 4.5 所示。

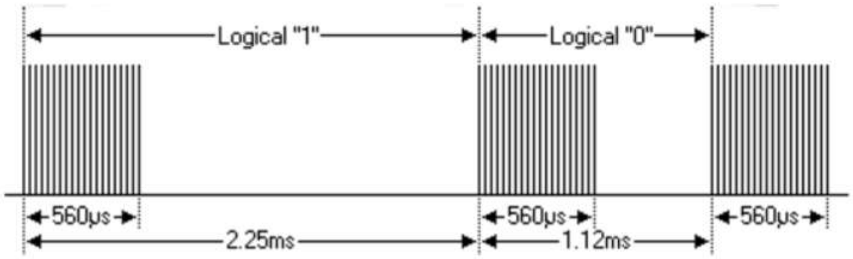

图 4.5　NEC 协议逻辑 01 示意图

NEC 协议规定了其自身的格式：先发送 13.5ms 的开始码，它是由 9ms 的高电平和 4.5ms 的低电平构成，接下来发送的是地址码，共 8bit，然后发送的是 8bit 的地址反码，第五部分是命令码，这个是用来判断按下的是哪一个按键的，最后一部分是命令反码，整个协议中命令反码的作用是用来做数据校验的。NEC 脉冲链如图 4.6 所示：

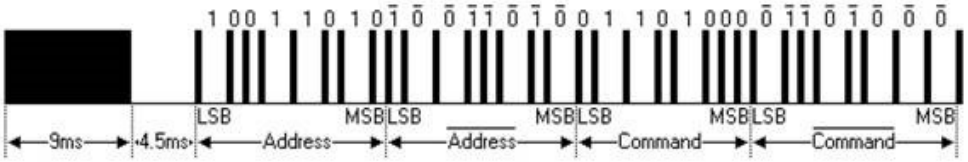

图 4.6　NEC 脉冲链示意图

当一直按着一个键时，NEC 并不是一直按照其脉冲链发送上述的信号，而是第一次的时候严格按照其协议发送，而后面发送的都是重复码。重复码是循环发送的，间隔为 110ms，NEC 重复发送载波协议如图 4.7 所示，上重复码如图 4.8 所示。

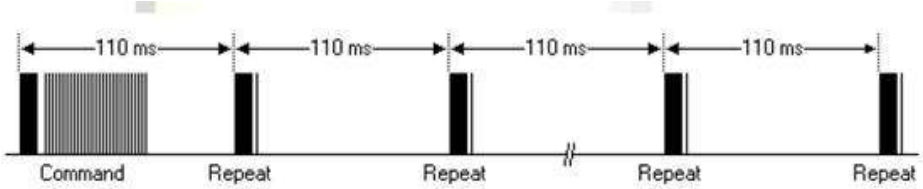

图 4.7　NEC 重复发送载波协议示意图

图 4.8　NEC 重复码

4.2.3 红外编码

所谓的编码其实就是相对于红外发射来讲的，按照 NEC 规定的协议，在高电平时间内，Raspberry Pi 的 GPIO17 （BCM）输出固定频率载波；低电平则直接输出低。红外接收头接到载波时输出高电平，没有载波的情况下就会输出低电平，就完成了数据解码。

4.3 Raspberry Pi 和 2 个 4*4 键盘

基于嵌入式的万能遥控器一共设计了 24 个按键，如果直接焊接 24 个按钮并使用 24 个引脚作为输入的话，那么不仅代码工作量要增加很多，而且大多数代码都是一些重复性的，还会导致树莓派的 GPIO 管脚将被占用多达 24 个，这样既不好管理，也让代码冗余，所以选择了 2 个 4*4 的矩阵键盘。

4*4 矩阵键盘是使用 16 个按键组合设计的一种特殊的键盘，它只需要接树莓的上 8 个 GPIO 口，通过程序设计进行键盘扫描，程序可以很轻易的获取到用户按下的到底是什么键。

键盘扫描原理：如图 4.9 所示，由于该键盘是 4*4 的，有四行四列，所以会有八个引脚，假设这八个引脚分别是 Raspberry Pi 的 GPIO10、GPIO4、GPIO3、GPIO2、GPIO24、GPIO23、GPIO15 和 GPIO14，前四个引脚的初始状态为[1，1，1，1]，后四个引脚的初始状态为[0，0，0，0]，当有按键按下时一定会有后四个引脚中的某个低电平把前四个引脚中的某个高电平拉低，也就是说前四个引脚和后四个引脚中分别会有一个低电平，其余引脚皆为高电平，当两次状态不一致时，程序就判断为有按键按下。具体是哪个按键以及判断过程如下：

首先程序将 GPIO14 引脚置低电平，其余均为高电平，如果此时引脚状态不是[1，1，1，1，1，1，1，0]，那么按下的按键一定是为第一行中的某个，这是因为按键按下将导致电路导通，高电平被拉低，有以下几种情况：

（1）若为 S1 按下，GPIO2 引脚的电平被拉低，则输出值为 1110 1110；

（2）若为 S2 按下，GPIO3 引脚的电平被拉低，则输出值为 1101 1110；

（3）若为 S3 按下，GPIO4 引脚的电平被拉低，则输出值为 1011 1110；

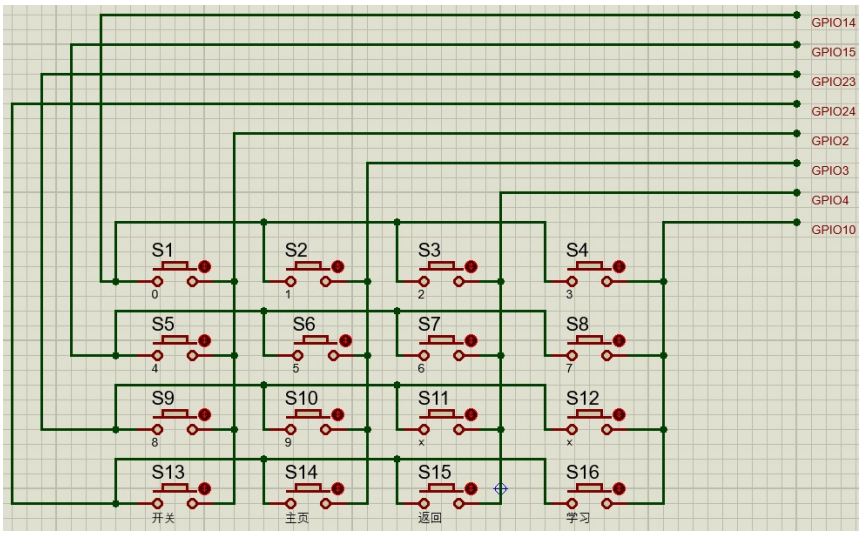

图 4.9　设计的矩阵键盘 1

（4）若为 S4 按下，GPIO10 引脚的电平被拉低，则输出值为 0111 1110；

那如果引脚状态就是设置的[1，1，1，1，1，1，1，0]呢，则代表按键按下的不在第一行，程序将以此对 GPIO15、GPIO23、GPIO24 置低电平，并进行上述同样原理的判断。

所以，在键盘扫描过程中，会从第一行开始逐行扫描，并且无论哪一行的哪个键被按下引脚状态都会唯一对应一个特定的值，通过一个字典做一个 key-value 的映射，由此就可以判断出按下的具体是哪个键。

图 4.9 是本系统设计的第一块 4*4 矩阵键盘，其按键与对应的键值都有在图中标出。第二块 4*4 矩阵键盘电路如图 4.10 所示。

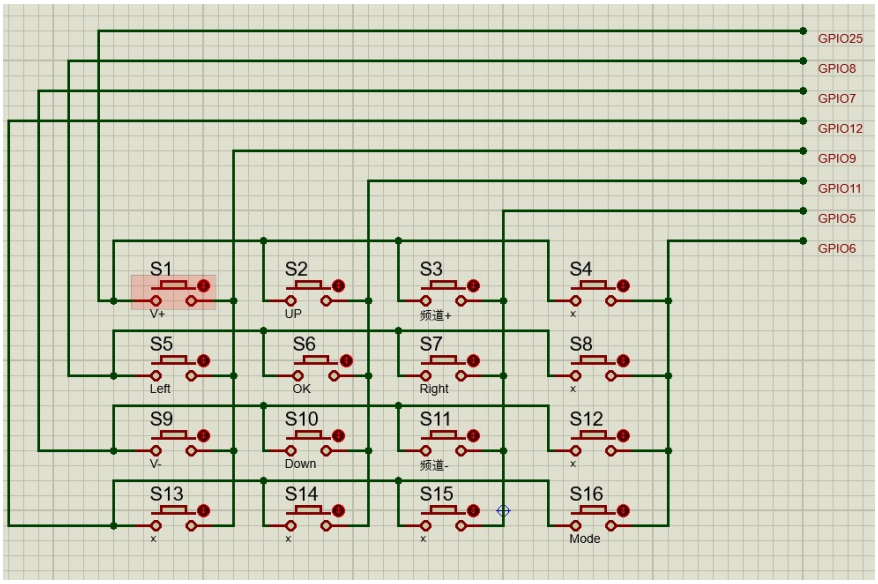

图 4.10　设计的矩阵键盘 2

本系统的键盘扫描程序是通过 Python 的第三方库 pad4pi 实现的，pad4pi 是

专门为 Raspberry Pi 实现键盘扫描的一个第三方库，使用者仅需要在创建一个 keypad 程序时传入选择的 4 个行 GPIO 引脚、4 个列 GPIO 引脚和一个 4*4 的列表 KEYPAD，KEYPAD 里面存储的是每个按键对应的键值。pad4pi 内部也是根据上述原理做的一个程序，它先将使用者给出的行列引脚进行内部绑定，然后创建一个字典，由于在 4*4 键盘中，每个按键按下程序设计使得 8 个引脚的输出都是固定的值，这个值它在内部先定义了 16 个字符串，在 KEYPAD 列表传进去时，pad4pi 内部首先创建一个字典，然后会对列表进行循环，然后在字典中追加一一对应的 key-value 对，当程序读取到 8 个引脚状态改变时并扫描到哪一行时，直接根据此时的引脚状态获取到按键值，并将此按键值传递到中断函数中，作为中断函数的参数。图 4.11 是安装 pad4pi 的指令。

```
:~# sudo pip3 install pad4pi
```

图 4.11　安装 pad4pi

4.4 Raspberry Pi 和提示灯

系统设计了 4 个提示灯，编号为 ABCD，提示灯的亮灭与其功能对应见表 4.1 所示。

表 4.1　提示灯说明

编号	说明
A	灯亮表示在学习状态
B	灯亮表示需要按下矩阵键盘的按键
C	灯亮代表刚才的按键的学习没有成功
D	灯闪烁代表发射红外成功

当在学习状态是，A 灯会亮；当 B 灯亮时则说明使用者应该要按下矩阵键盘中的某个按键，好让程序能获得一个键值，使 lirc 能继续学习；当 C 灯亮时，说明使用者虽然按下了按键，但是 10s 内没有收到红外信号；而 D 灯是当使用者把 Raspberry Pi 作为普通遥控器的时候，如果按下一个有用的按键，则 D 灯闪烁。相关电路如下图 4.12 所示。

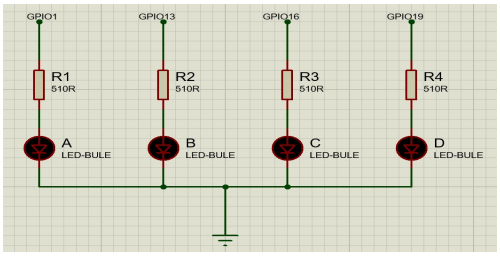

图 4.12　提示灯电路图

这四个引脚定义在 OUT_PINS 列表中，对应关系见表 4.2 所示。

表 4.2　指示灯管脚编号

编号	引脚
OUT_PINS[0]	GPIO1
OUT_PINS[1]	GPIO13
OUT_PINS[2]	GPIO16
OUT_PINS[3]	GPIO19

4.5 学习功能与遥控功能的实现

学习与遥控整体程序流程图如图 4.13 所示。

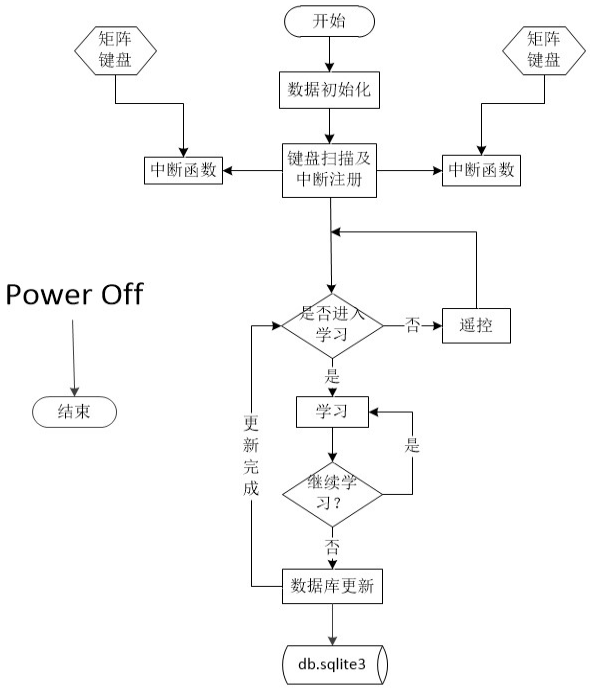

图 4.13　学习与遥控流程图

4.5.1 学习功能

系统学习与本地遥控是独立运行的，不依赖于 Web 系统，由 Python 单独启动一个进程运行。代码里面通过一个死循环来确保程序一直运行而不结束，通过对键盘的扫描并实现中断，在中断函数里面更改当前按键信息，循环判断按键是否是学习键来开始学习、退出学习或者是普通遥控功能。当学习结束之后进入数据库更新操作。

系统学习功能与遥控功能的实现依赖于 lirc 软件，但是 lirc 是基于 Linux 系统的，所以用 Python 操作的话必须要使用 Python 与 Linux 交互的工具，这里采用的是 Python 的第三方库 Pexpect 和 Subprocess。首先安装这两个库，执行如图

4.14 所示的命令。

```
:~# sudo pip3 install pexpect && pip3 install subprocess
```

图 4.14　安装 Pexpect 和 Subprocess

Pexpect 能很方便的与 Raspberry Pi 的 Linux 系统通信，当使用 Pexpect 执行一条 Linux 命令时，它会创建一个阻塞的线程，直到这条 Linux 命令运行结束或者异常退出，在线程阻塞时，Pexpect 可以对 Linux 命令的返回结果进行匹配，匹配成功返回匹配的索引，开发者可以根据不同的索引做出相应的操作。如果什么也匹配不到，则会一直阻塞，直到用户主动发送退出指令或者超时。Subprocess 则是可以运行 Linux 命令并将结果以字符串形式返回。这两个库是完成学习功能的基石。

数据初始化：本课题采用的 Raspberry Pi 的 GPIO 工作方式为 BCM 方式，所以在开始的时候就设置了 GPIO 的工作方式。在树莓派（Raspberry Pi ）的 GPIO 工作过程中，如果特定功能的引脚设置为其它用途，将会发出警告，所以代码中将警告设置为不警告。此外还初始化了几个全局变量，如表 4.3 所示。

表 4.3　全局变量及其意义

全局变量	意义
study	Bool 类型，用来判断是否按下学习键
study_list	列表类型，保存学习中按下的按键值
press_key	字符串类型，保存当前按下的按键值
remote_name	字符串类型，保存当前遥控器的名字

键盘扫描和中断的注册：首先定义矩阵键盘对应键值、行引脚和列引脚，由于本课题使用了两个键盘，所以不管是键值、行引脚列引脚都要定义两种，如图 4.15 所示。接着将定义的值分别传入 pad4pi，这样键盘扫描程序就定完成，再使用 pad4pi 为每个键盘注册中断，中断函数为同一个（printKey）。

```
KEYPAD1 = [
    ["KEY_VOLUMEUP", "KEY_UP", "KEY_CHANNELUP", "None"],
    ["KEY_LEFT", "KEY_OK", "KEY_RIGHT", "None"],
    ["KEY_VOLUMEDOWN", "KEY_DOWN", "KEY_CHANNELDOWN", "None"],
    ["None", "None", "None", "KEY_MODE"]
]
KEYPAD2 = [
    ['KEY_0', 'KEY_1', 'KEY_2', 'KEY_3'],
    ['KEY_4', 'KEY_5', 'KEY_6', 'KEY_7'],
    ['KEY_8', 'KEY_9', 'None', 'None'],
    ['KEY_POWER', 'KEY_HOME', 'KEY_BACK', 'study']
]
COL_PINS1 = [2, 3, 4, 10]  # BCM numbering
ROW_PINS1 = [24, 23, 15, 14]  # BCM numbering
COL_PINS2 = [9, 11, 5, 6]  # BCM numbering
ROW_PINS2 = [12, 7, 8, 25]  # BCM numbering
factory1 = rpi_gpio.KeypadFactory()
factory2 = rpi_gpio.KeypadFactory()
keypad1 = factory1.create_keypad(keypad=KEYPAD1, row_pins=ROW_PINS1, col_pins=COL_PINS1, key_delay=200)
keypad2 = factory2.create_keypad(keypad=KEYPAD2, row_pins=ROW_PINS2, col_pins=COL_PINS2, key_delay=200)
keypad1.registerKeyPressHandler(printKey)
keypad2.registerKeyPressHandler(printKey)
```

图 4.15　键盘扫描及中断函数注册

中断函数里面的内容很简单，它带了一个参数 key，进入中断时，这个 key 值就是当前使用者按下的那个键的键值。中断函数流程如图 4.16。

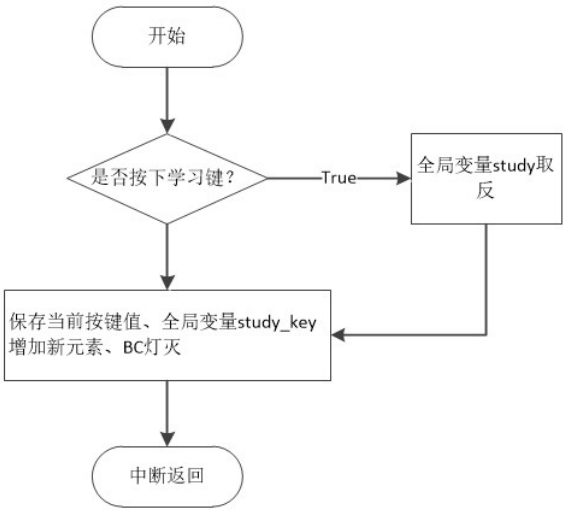

图 4.16　触发中断时的逻辑处理

学习逻辑功能实现：当使用者第一次按下学习键时，全局变量 study 将会在中断函数中变成 True，程序运行学习功能。学习功能逻辑使用的是利用 pexpect 执行 lirc 的录制红外的命令，命令执行后，可能产生的结果一共有 7 中，如图 4.17 所示。

```
li = [
    'Press RETURN to continue.',
    'Press RETURN now to start recording.',
    'irrecord: no data for 10 secs, aborting',
    'Please enter the name for the next button (press <ENTER> to finish recording)',
    'The last button did not seem to generate any signal.\nPress RETURN to continue.',
    pexpect.EOF,
    pexpect.TIMEOUT
]
```

图 4.17　录制过程中可能返回的结果

现在说明这 7 种情况：

① lirc 录制指令刚运行时，需要按下回车才能继续执行；

② lirc 需要再接收一个回车，才能开始录制，计算 gap 值；

③ 在开始录制时，如果 10s 内没有接收到红外信号，Linux 命令退出程序逻辑在匹配退出时处理，此时不处理；

④ 输入一个想要的键值，按回车继续

⑤ 输入键值后，10s 没有接收到红外信号；

⑥ Linux 命令执行完毕

⑦ 超出一定时间没有匹配到 Linux 返回的上述任何一个值，超时继续等待，

此时不处理。

学习的整个功能流程图如图 4.18 所示。

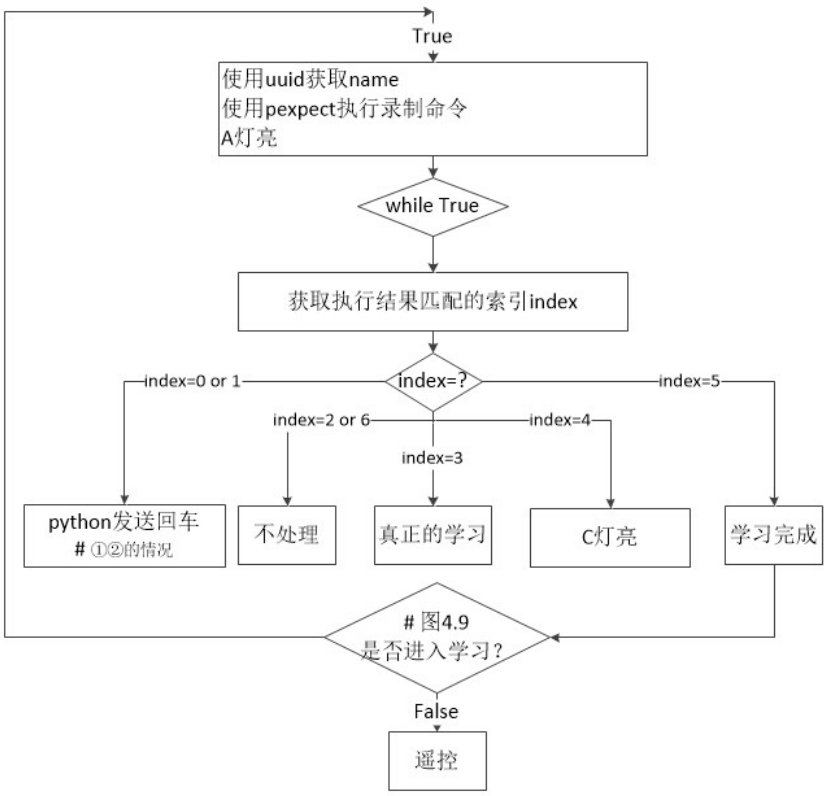

图 4.18　学习功能流程示意

当 index 为 3 时，对应了情况④，此时需要等待使用按下按键触发中断，将 press_key 赋值，程序会将 press_key 发送到 Linux 进程，重复如此，直到用户再次按下学习键，学习结束。这个逻辑流程图如图 4.19 所示。

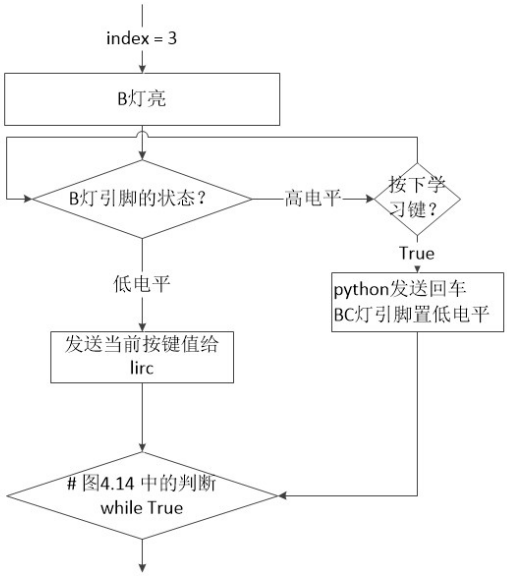

图 4.19　按键与学习

当学习完成后，程序会将当前学习好的文本内容复制到 lirc 的当前设备信息中，并且调用 update 函数，将学习完成的按键持久化到数据库中，流程图如图 4.20 所示。

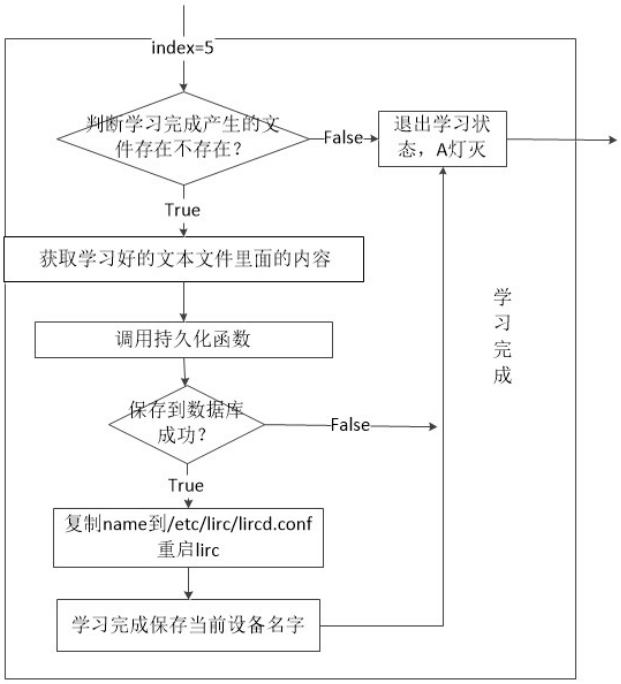

图 4.20　学习完成功能示意流程图

Web 和硬件能够交互就是依靠这最后一步了，将学习好的遥控信息持久化到数据库中，update 函数流程图如图 4.21 所示。

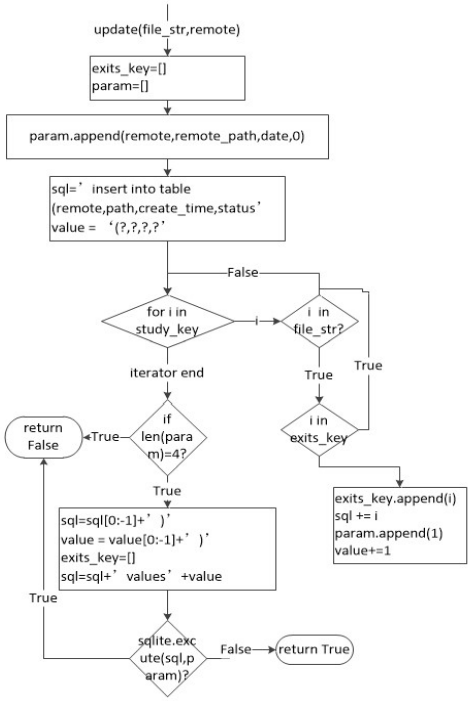

图 4.21　数据持久化流程示意图

4.5.2 遥控功能

如果使用者没有按下学习键，那么此时的 Raspberry Pi 就相当于一个普通的遥控器，使用者可以按下任何一个键（如果该键学习过），Raspberry Pi 将通过 lirc 发送相应的红外信号。万能遥控器的遥控功能逻辑示意图如图 4.22 所示：

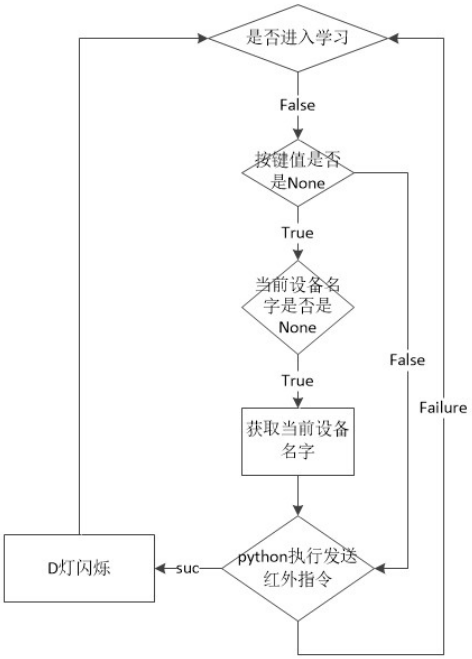

图 4.22　万能遥控的遥控功能示意

因为 lirc 发送红外是需要两个条件的，第一个是设备的名字，第二个是设备的按键名字，所以只有两个都不为空的时候才能进行发射。但是，有时候会有没有把矩阵键盘上的所有按键都学完，那么就会发送失败，即什么也不操作，发送成功时让 D 灯闪一下。获取 remote_name 时是依赖 lirc 的打印当前设备名字的命令：”sudo irsend list ‘’ ‘’”，这条命令会返回一条字符串，格式下：”irsend:*”，”*” 代表的就是当前设备的名字，程序使用了字符串分割来获取 remote_name。

4.6 硬件设计总结

硬件设计就是整个学习功能和遥控功能的实现，依赖于 Raspberry Pi、红外拓展板、指示灯和两个矩阵键盘。在整个设计中，利用 Visio 绘画了多种可能的流程图，设计过程中进行了大量的实验，让学习与遥控部分变得更加的可靠。

第五章　Web 端设计

本章节将详细描述 Web 端部分，包括了数据的设计、信息验证模块、远程遥控模块、设备命名模块。

5.1 Web 端功能流程

Web 端使用 Django 快速开始的一个项目，项目命名为 xupt-ghj，一共分为 6 大模块，每个模块之间都是采用的 RESTful 设计。其功能流程图如图 5.1 所示。

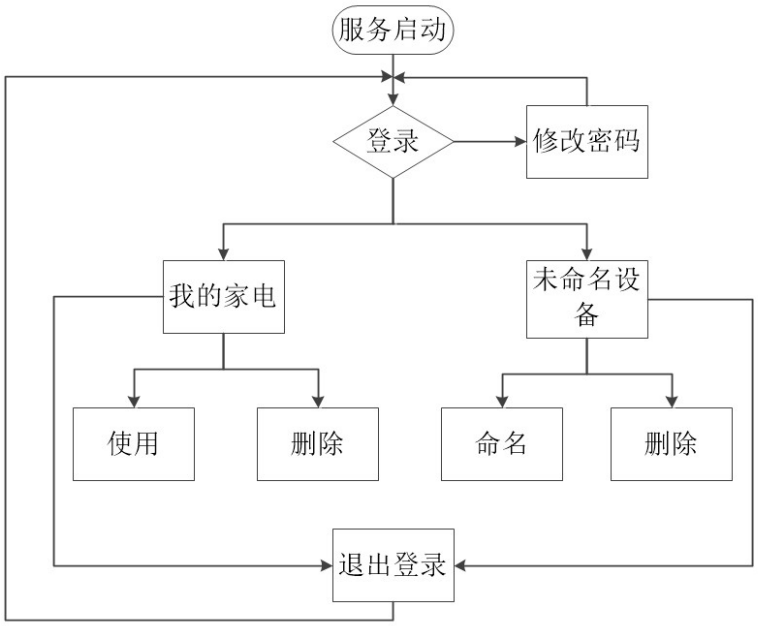

图 5.1　Web 端功能流程图

5.2 基于嵌入式的万能遥控器的数据库设计

5.2.1 用户表设计

用户表表的名字是 T_XUPT_USER，此表里面存储着一些登录验证信息，如账号密码姓名等。表结构设计见表 5.1 所示。

表 5.1　用户表（T_XUPT_USER）

字段名称	数据类型	是否可空	字段描述
id	VARCHAR(255)	N	主键字段，登录账户
user_name	VARCHAR(255)	N	姓名
pass_word	VARCHAR(255)	N	密码

使用者在 Django 的 model 中定义字段类型是通过数据库的建表操作，再经

由 Django 自动的创建表。表 T_XUPT_USER 的定义如图 5.2 所示：

```
class User(models.Model):
    user_name = models.CharField('姓名', max_length=255, null=True)
    pass_word = models.CharField('密码', max_length=255)
    id = models.CharField('账号', max_length=255, primary_key=id)

    class Meta:
        db_table = 'T_XUPT_USER'
```

图 5.2　表 T_XUPT_USER 的设计

5.2.2 设备表设计

设备表的名字是 T_XUPT_DEVICE，它里面存储的是设备的真实名字、学习过的按键、绝对路径、使用者起的名字以及一个用来表示是否命名完成的状态。表结构设计如表 5.2 所示。

表 5.2　设备表（T_XUPT_DEVICE）

字段名称	数据类型	是否可空	字段描述
id	INTEGER	N	主键字段，自增长
name	VARCHAR(255)	Y	自定义设备的名字，唯一
remote	VARCHAR(255)	N	设备的真实名字，唯一
path	VARCHAR(255)	N	设备的绝对路径
status	INTEGER	Y	0 表示没有命名，1 表示已完成命名
create_time	VARCHAR(255)	Y	该设备的学习时间
key_power	VARCHAR(255)	Y	开关按键，1 代表学习了，0 代表没有学习
key_one	VARCHAR(255)	Y	按键 1，1 代表学习了，0 代表没有学习
key_two	VARCHAR(255)	Y	按键 2，1 代表学习了，0 代表没有学习
key_three	VARCHAR(255)	Y	按键 3，1 代表学习了，0 代表没有学习
key_four	VARCHAR(255)	Y	按键 4，1 代表学习了，0 代表没有学习
key_five	VARCHA(255)	Y	按键 5，1 代表学习了，0 代表没有学习

key_six	VARCHAR(255)	Y	按键 6，1 代表学习了，0 代表没有学习
key_seven	VARCHAR(255)	Y	按键 7，1 代表学习了，0 代表没有学习
key_eight	VARCHAR(255)	Y	按键 8，1 代表学习了，0 代表没有学习
key_nine	VARCHAR(255)	Y	按键 9，1 代表学习了，0 代表没有学习
key_zero	VARCHAR(255)	Y	按键 0，1 代表学习了，0 代表没有学习
key_up	VARCHAR(255)	Y	上按键，1 代表学习了，0 代表没有学习
key_down	VARCHAR(255)	Y	下按键，1 代表学习了，0 代表没有学习
key_left	VARCHAR(255)	Y	左按键，1 代表学习了，0 代表没有学习
key_right	VARCHAR(255)	Y	右按键，1 代表学习了，0 代表没有学习
key_ok	VARCHAR(255)	Y	ok 键，1 代表学习了，0 代表没有学习
key_back	VARCHAR(255)	Y	返回键，1 代表学习了，0 代表没有学习
key_home	VARCHAR(255)	Y	主页，1 代表学习了，0 代表没有学习
key_mode	VARCHAR(255)	Y	空调按键（暂定），1 代表学习了，0 代表没有学习
key_volume_up	VARCHAR(255)	Y	音量加，1 代表学习了，0 代表没有学习
key_volume_down	VARCHAR(255)	Y	音量减，1 代表学习了，0 代表没有学习
key_channel_up	VARCHAR(255)	Y	频道加，1 代表学习了，0 代表没有学习
key_channel_down	VARCHAR(255)	Y	频道减，1 代表学习了，0 代表没有学习

表 T_XUPT_DEVICE 的字段有点多，因为它包含了矩阵键盘的每个按键值，而系统设计的时候采用了两个矩阵键盘一共使用了 23 个学习键值。设备表使用 Django 的定义如图 5.2 所示。

```
class Device(models.Model):
    name = models.CharField('自定义设备名字', max_length=255, unique=True, null=True)
    remote = models.CharField('自动生成设备名字', max_length=255, unique=True)
    path = models.CharField('设备路径', max_length=255, unique=True)
    status = models.IntegerField('是否已完成命名，0没有，1完成', null=True)
    create_time = models.CharField('设备创建时间', max_length=255, null=True)
    key_power = models.CharField('开关键', max_length=255, null=True)
    key_one = models.CharField('1键', max_length=255, null=True)
    key_two = models.CharField('2键', max_length=255, null=True)
    key_three = models.CharField('3键', max_length=255, null=True)
    key_four = models.CharField('4键', max_length=255, null=True)
    key_five = models.CharField('5键', max_length=255, null=True)
    key_six = models.CharField('6键', max_length=255, null=True)
    key_seven = models.CharField('7键', max_length=255, null=True)
    key_eight = models.CharField('8键', max_length=255, null=True)
    key_nine = models.CharField('9键', max_length=255, null=True)
    key_zero = models.CharField('0键', max_length=255, null=True)
    key_up = models.CharField('上键', max_length=255, null=True)
    key_down = models.CharField('下键', max_length=255, null=True)
    key_left = models.CharField('左键', max_length=255, null=True)
    key_right = models.CharField('右键', max_length=255, null=True)
    key_ok = models.CharField('ok键', max_length=255, null=True)
    key_back = models.CharField('返回键', max_length=255, null=True)
    key_home = models.CharField('主页键', max_length=255, null=True)
    key_mode = models.CharField('空调键', max_length=255, null=True)
    key_volume_up = models.CharField('音量加键', max_length=255, null=True)
    key_volume_down = models.CharField('音量减键', max_length=255, null=True)
    key_channel_up = models.CharField('频道加键', max_length=255, null=True)
    key_channel_down = models.CharField('频道减键', max_length=255, null=True)

    class Meta:
        db_table = 'T_XUPT_DEVICE'
```

图 5.3　表 T_XUPT_DEVICE 的设计

5.2.3 生成表

表设计完成后，在命令行先执行如图 5.4 的命令，这个命令意思是说创建了一些数据库日志，在之后使用者可以通过这些日志进行数据表的字段的恢复；然后再执行如图 5.5 的命令，这是说读取数据库日志，再进行数据库的更新。

```
>python manage.py makemigrations
```

图 5.3　创建数据库日志

```
>python manage.py migrate
```

图 5.3　更新数据库

接下来 django 就在 sqlite3 数据库中创建了这两个表。

5.3 信息验证模块

信息验证模块包括两个模块，一个是登录模块，这是进入远程遥控系统的唯一路径；一个是密码修改模块，这是考虑到使用者忘记密码无法登录时，用户可以通过修改密码的方式去重置密码，再进行登录。框架图如图 5.4 所示。

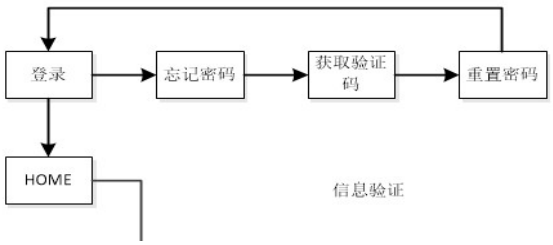

图 5.4　信息验证框架

5.3.1 登录模块

登录模块是整个远程遥控系统的起点，页面使用 jQuery 进行渲染修饰，当用户输入完账号密码点击登录时，首先 JS 会对使用者的输入进行非空校验，如果账号密码有一个为空都会进行提示不能为空，校验通过后，浏览器会向后台发送一个携带了用户名密码的请求到后台，整个逻辑如图 5.5 所示。

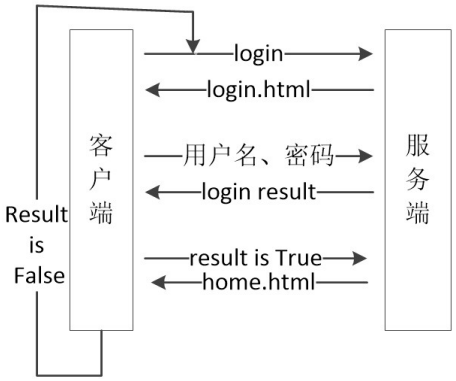

图 5.5　登录逻辑

后台接收到用户名密码后，进行登录校验，校验成功，则在当前会话中保存用户的登录信息，本系统保存的是用户的账号，然后返回 ok。如果登录校验失败了，就会返回空的字符串。前台收到返回的消息时，则进行判断，是 ok 则跳 home 页面，否则停留在 login 页面。Login 页面如图 5.6 所示：

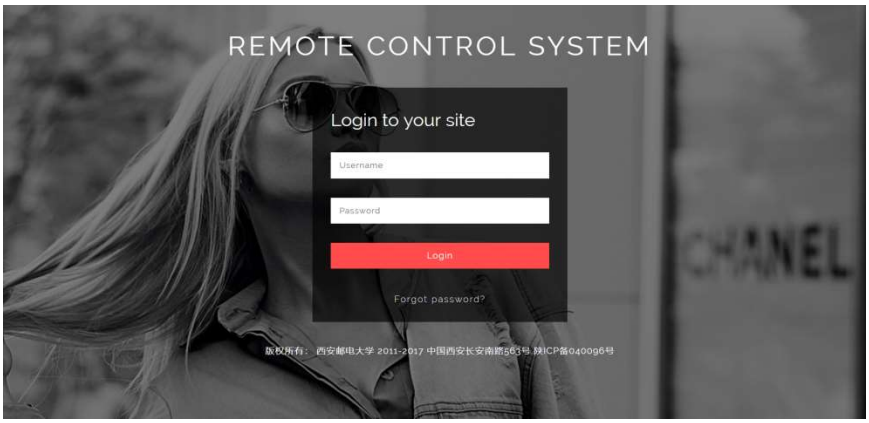

图 5.6　登录页面

当用户名或密码为空时，提示如图 5.7 所示：

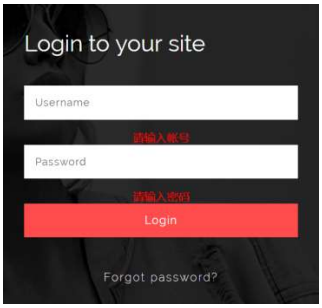

图 5.7　账号密码为空时提示

5.3.2 修改密码

使用者可以通过点击登录页面中的”Forgot password?”来进入修改密码页面，忘记密码的页面如图 5.8 所示：

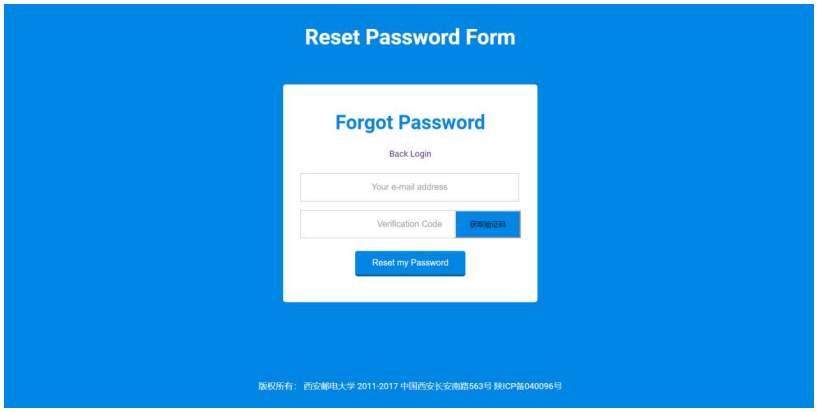

图 5.8　忘记密码页面

使用者通过输入自己的邮箱来获取验证码，验证码为 4 位随机数字，保存在 Redis 数据库中，并设置了一分钟有效期，使用者必须要验证码失效之前进行验证码的确认。流程图如图 5.9 所示。

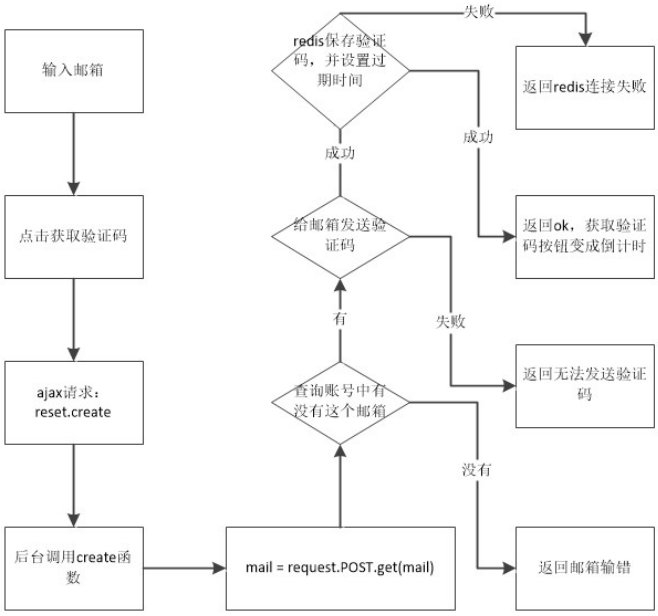

图 5.9　获取验证码流程

其中给邮箱发送验证码功能是通过以在 163 申请的一个邮箱账号配合 python 内置的 smtplib 库和 MIMEText，验证码通过 4 次循环生成 4 个随机数生成。

使用者得到验证码后，可进行验证码校验，校验通过后 redis 将删除验证码，并将邮箱作为 key 重新设置 value 值为 True，超时时间为 5 分钟，使用者需要在 5 分钟内进行密码修改，否则需要重新获取验证码。密码修改界面如图 5.10 所示。

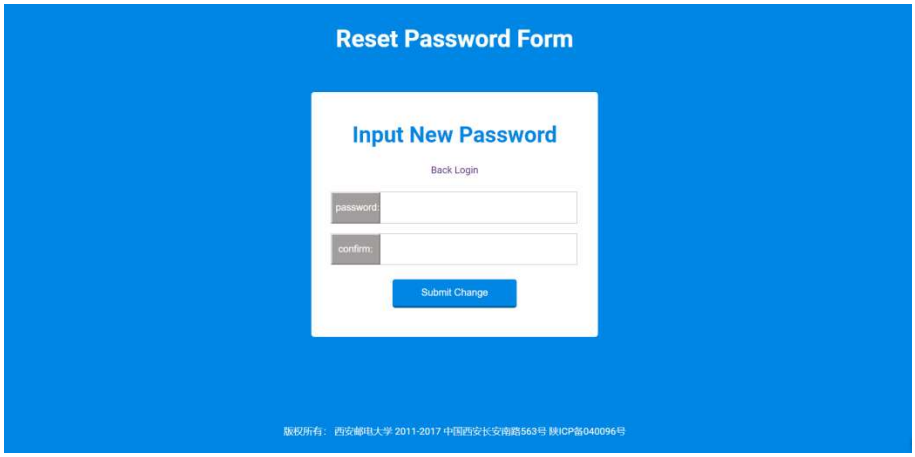

图 5.10　更改使用者的密码界面

使用者需要输入两次密码，在后台和前端都有进行两次输入的密码是否一致的验证，两次一致时，密码才会修改成功，然后返回到登录页面。

使用者登录成功后将看到 home 页面，home 页面如图 5.11 所示。

图 5.11　主页面显示

Home 页面中可以从左侧功能导航栏进入查看未命名设备和我的家电页面。

5.4 信息展示

在硬件设计部分，红外学习功能使用的是 lirc，它在录制时需要指定一个 name 参数，学习代码中使用的是 UUID 来代替 name 参数，但是 UUID 仅仅只是一串在同一时空下不重复的字符串，本身并没有任何意义。所以在学习完成之后，使用者需要登录到系统中，此时系统会提示刚刚学习的设备没有命名，没有命名的设备是不允许被使用的。

使用者可以通过任何页面的右上角的信息进入未命名页面，学习过的所有未命名的设备都会在这个页面显示，如图 5.12 所示。

图 5.12 未命名设备的展示

这些数据由跳转到这个页面时候携带而来的，在跳转到这个页面之前，后台先进行查询操作，由于没有命名的设备，在数据库中的记录中的 state 字段会是 0，所以未命名设备查询所有设备并过滤 state 字段为 0，相关逻辑如图 5.13 所示。

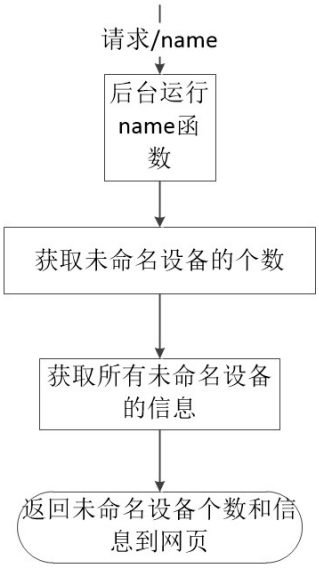

图 5.13　获取未命名设备

使用者点击删除时，ajax（网络请求）请求会携带该设备的 id，后台收到 id 之后，执行 sql 删除语句。删除不能恢复。

点击命名时，弹出输入框，如图 5.14 所示：

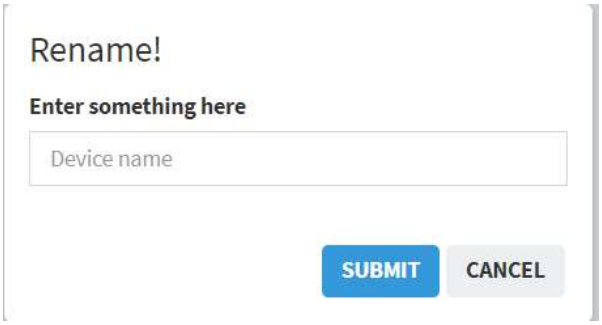

图 5.14　输入新的名字

确定后，后台处理逻辑流程图如图 5.15 所示。

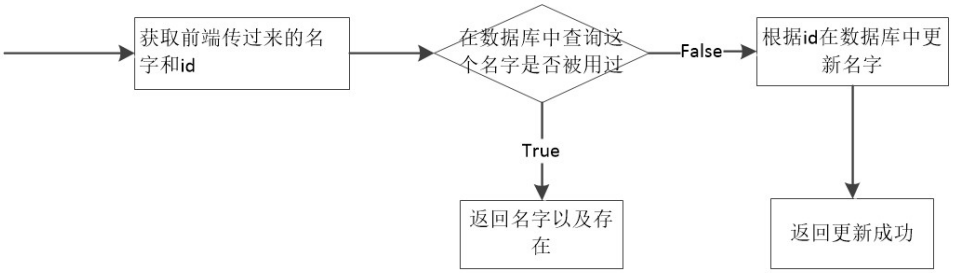

图 5.15　重命名的逻辑处理

重命名逻辑判定同名设备是不行的（一个家庭里面可以做到为每一个自己学习的设备起一个不同的名字），命名成功后，将刷新本页面，同时该设备不再在该页面中，会出现在“我的家电”页面中。

5.5 远程遥控

我的家电页面展示了所有学习好并且已命名的家电信息，在前端的设计中只显示了序号、自己命名的名字、创建时间、类型以及操作列。家电信息的获取和上文中未命名设备信息的获取原理是一样的。

操作列同样有两个功能，一个是使用该设备，一个是删除该设备，删除逻辑和未命名设备的删除是同一个，所以系统设计的时候也只是设计了一个 API，两个删除都是使用 ajax 请求同一个动作。

整个远程遥控系统的中心在于点击了使用设备。本课题设计了一个网页版的虚拟遥控器，如图 5.16 所示，当使用者点击使用按钮时，则会进入虚拟按键界面。这个按键的所有按键都是与硬件中的两个 4*4 矩阵键盘的键值是一一对应的。

图 5.16　网页虚拟遥控器示意

由于存在学习过程中只学了一部分按键，所以可能不会所有按键都有效，本课题的设计是如果该设备没有学习那个按键，则使这个按键禁用，让使用者不能点击，如图 5.17 所示：

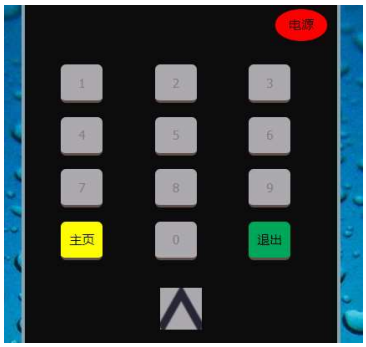

图 5.17　尚未学习的按键示意

没有学习到的按键会变灰且不能被点击。这个功能的实现是基于 jQuery 完成的。逻辑如图 5.18 所示：

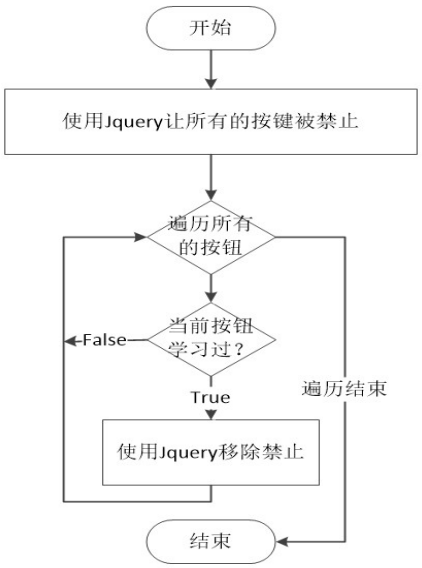

图 5.18 按键能否点击逻辑图

部分代码如图 5.19 所示。

```
$('#remote button').attr('disabled', 'true');
{% for mode in device %}
    {% if mode.key_one == '1' %}
        $("#KEY_1").removeAttr('disabled');
    {% endif %}
    {% if mode.key_two == '1' %}
        $("#KEY_2").removeAttr('disabled');
    {% endif %}
    ...
    ...
    {% if mode.key_power == '1' %}
        $("#KEY_POWER").removeAttr('disabled');
    {% endif %}
{% endfor %}
```

图 5.19 按键能否点击代码

系统设计时为每个按钮都增加了一个 value 属性，值为”name-key”，name 是使用者当前使用的设备名字，key 为每个按钮对应的键值。同时设计时也为每一个按钮添加了一个点击事情，所有的点击事情指定了同一个函数，这个函数将接受点击的那个按键的 value 值。远程遥控前端 JS 逻辑如图 5.20 所示。

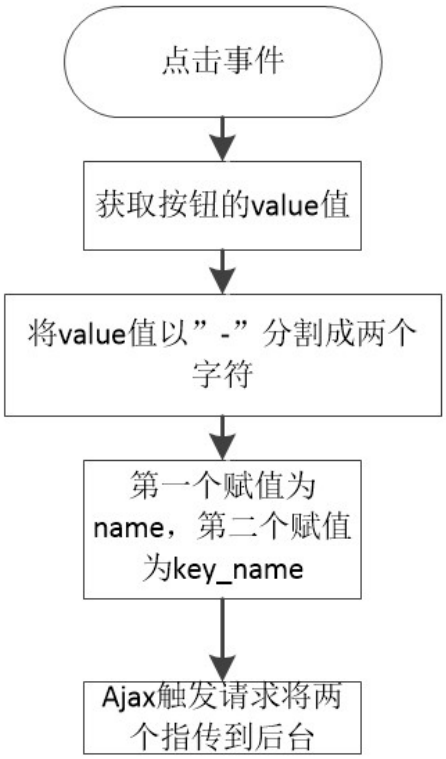

图 5.20　远程遥控发出网络请求

经过 ajax 请求，后台就能拿到对应的设备名字以及键值了，后台相应的发送红外的代码如图 5.21 所示。

```
def send(device, key):
    try:
        device = Device.objects.filter(name=device).values('path', 'remote')
        result = subprocess.getoutput('sudo cp ' + device.get()['path'] + ' /etc/lirc/lircd.conf')
        if len(result) != 0:
            return False
        send_com = 'sudo irsend SEND_ONCE ' + device.get()['remote'] + ' ' + key
        result = subprocess.getoutput(send_com)
        if 'Connection refused' in result:
            os.system('sudo /etc/init.d/lirc restart')
            os.system('sudo lircd start')
            os.system('sudo lircd -d /dev/lirc0')
            result = subprocess.getoutput(send_com)
        if len(result) == 0:
            return True
        else:
            return False
    except:
        print(traceback.print_exc())
        return False
```

图 5.21　远程遥控发射红外代码

首先通过自定义的名字来获取想要使用的设备的绝对路径和当前设备的真实名字（UUID），然后将其复制到 lirc 的设备默认路径。接着使用 subprocess 运行 lirc 红外发送命令，如果返回结果包含拒绝信息，那么、则证明是 lirc 没有启动，所以启动 lirc 后需要再发送一次红外指令，若是没有返回字符串则代表发送成功，则直接返回 True，前端将提示发送成功；有返回字符串，则表示红外发射

失败，前端提示发送失败信息。

5.6 Web 端总结

Web 端设计了远程遥控系统，使用者需要登录才能进行远程遥控。在远程遥控系统中，设计了一个虚拟的遥控器界面，使用者可以通过点击对应的按钮来实现远程遥控。Web 端的设计查阅了资料，课题设计里每一种情况都有做考虑，提高了系统的容错率。

第六章　总结

整个系统设计是一个从无到有的过程，从开始拿到毕设课题时的迷茫到现在完成了毕设，整个过程有太多的艰辛，伴随着太多的谷歌查资料、图书馆看书。所有相关技术的一点点积累，才逐渐有了硬件、Web 的联系交互，才逐渐有了硬件分哪些模块、Web 分哪些模块的想法。

系统设计的过程中遇到过许多问题，在这个自我积累的过程中，也收获颇多。尽管想清楚怎么分模块，但是对于各个模块的功能实现还是一筹莫展，经过多次上网搜索和去图书馆参阅资料，最终发现了 lirc 这个开源软件包。确定了基石，那么建高楼的材料就更好确定了，因为 lirc 是运行在 Linux 系统上，所以便采用了 Raspberry Pi 作为整个系统的容器。

选好材料后，对模块的划分又更加明确了，然后是查看文档，将相关的技术熟悉起来。

设计的过程无疑是非常消耗时间且无聊的，但是在这个过程中能学到很多东西却又让人非常开心，尤其是当一步一步设计都有成果时让人非常雀跃。

基于嵌入式的万能遥控器还不是一个非常完善的作品，它目前有的功能：

① 学习功能

② 遥控功能

③ 远程遥控功能

接下来作者会不断创新、学习，去完善更多的功能。

第七章　展望

近年来，红外遥控器的发展也越来越好，许多遥控器上都具备了学习功能，然而绝大多数遥控器其实并不具有学习功能，更不要说要实现远程遥控了。人们的需求在增加，一个一个的普通遥控器几乎难以满足人们的对高质量的生活标准的追求，因为一旦当遥控器坏了还需要去买一个专门的新遥控器。基于嵌入式的万能遥控器既支持学习，又支持远程遥控，使得人们远在千里之外都能遥控自己的家电设备，所以基于嵌入式的万能遥控器不但具有非常深远的意义，也拥有着一个美好上的市场前景。

本系统的设计仅仅采用了网页版的远程遥控，但这仍然是完全不够的，基于嵌入式的万能遥控器还有开发的必要性，可以介入微信的控制、可以适配安卓 APP 或者是 IOS 苹果应用的控制，也可以接入一个网络摄像头，这样人们还可以试试查看设备的运行情况，或者增加自定义定时的功能，去创造出一个越来越智能的化遥控器，使人们的生活更加便捷！

致　谢

不知不觉大四的生活已经接近尾声，这一学期开学以来就开始毕业设计的准备，从开始的毫无头绪到后来各种思路想法的一点点积累，查阅相关资料，向老师向同学请教不懂的地方……一步一步地进行着我的毕业设计，直到毕业设计真正完成，其中遇到了数不胜数的困难。但是，由于老师同学的帮助与鼓励，给了我继续前行不断进步的勇气与动力。

首先，在本次论文设计过程中，感谢我的学校，给了我学习的机会。

然后非常感谢我的指导教师——马翔老师，从论文的选题、开题报告的撰写、资料的查找，到结构的完善，都给予悉心指导，使我顺利成文。这片论文的每个实验细节和每个数据，都离不开他的细心指导，他也在我迷茫的时候给了我很多启示以及很多解决问题的方法，并提出了很多指导性的建议。在学习中，老师从选题指导、论文框架到细节修改，都给予了细致的指导，提出了很多宝贵的意见，老师以其严谨求实的治学态度、高度的敬业精神、兢兢业业、孜孜以求的工作作风和大胆创新的进取精神对我产生着重要影响。他渊博的知识、开阔的视野和敏锐的思维给了我深深的启迪。这篇论文是在马老师的精心指导和大力支持下才完成的

最后感谢所有授我以业的老师，没有这些年知识的积淀，我没有这么大的动力和信心完成这篇论文。感恩之余，诚恳地请各位老师对我的论文多加批评指正，使我及时完善论文的不足之处。

参考文献

[1] 李文胜.基于树莓派的嵌入式 Linux 开发教学探索[J].电子技术与软件工程,2014(09):219-220.

[2] 汪琴. 基于树莓派的高级语言程序设计类课程教学研究[D].重庆师范大学,2017.

[3] 王冉阳.基于 Django 和 Python 的 Web 开发[J].电脑编程技巧与维护,2009(02):56-58.

[4] 张露,马丽.数据库设计[J].安阳工学院学报,2007(04):76-79.

[5] 杜国祥,石俊杰.SQLite 嵌入式数据库的应用[J].电脑编程技巧与维护,2010(14):43-47.

[6] 马豫星.Redis 数据库特性分析[J].物联网技术,2015,5(03):105-106.

[7] 李泽光.基于单片机的红外遥控器解码器的设计[J].现代电子技术,2007(09):36-37+40.

[8] 芦健,彭军,颜自勇,陈文芗.自学习型智能红外遥控器设计[J].国外电子测量技术,2006(08):63-66.

[9] 唐永瑞,张达敏.基于 Ajax 与 MVC 模式的信息系统的研究与设计[J].电子技术应用,2014,40(02):128-131.

[10] 邱小果.编程实现基于 Cookie 验证的 HTTP 请求的发送[J].微型机与应用,2003(07):35-37.

[11] G. Abla,D.P. Schissel,B.G. Penaflor,G. Wallace. A remote control room at DIII-D[J]. Fusion Engineering and Design,2007,83(2).

[12] Grehan, Rick. Pillars of Python: Django Web framework[J]. InfoWorld.com,2011.

[13] Brigitte Ringbauer,Frank Heidmann. Usability von Smart Home User Interfaces — Herausforderungen und Lösungsansätze (Smart Home Usability — Challenges and Solutions)[J]. i-com,2006,5(1/2006).

附录 A　学习与遥控功能代码

```
import sys
import codecs
import os
import RPi.GPIO as GPIO
import time
from pad4pi import rpi_gpio
import pexpect
import uuid
import traceback
import sqlite3
import datetime
import subprocess

GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)

# ****************************************#


key_map = {
        'KEY_0': 'key_zero',
        'KEY_1': 'key_one',
        'KEY_2': 'key_two',
        'KEY_3': 'key_three',
        'KEY_4': 'key_four',
        'KEY_5': 'key_five',
        'KEY_6': 'key_six',
        'KEY_7': 'key_seven',
        'KEY_8': 'key_eight',
        'KEY_9': 'key_nine',
        'KEY_VOLUMEUP': 'key_volume_up',
        'KEY_VOLUMEDOWN': 'key_volume_down',
        'KEY_CHANNELUP': 'key_channel_up',
```

```
    'KEY_CHANNELDOWN': 'key_channel_down',
}


def printKey(key):
    # print(key)
    global study
    global study_key
    global press_key
    if key == 'study':
        '''print(key)'''
        study = not study
        # return
    if key != 'None':
        press_key = key
        study_key.append(key)
        GPIO.output(OUT_PINS[1], GPIO.LOW)
        GPIO.output(OUT_PINS[2], GPIO.LOW)


KEYPAD1 = [
    ["KEY_VOLUMEUP", "KEY_UP", "KEY_CHANNELUP", "None"],
    ["KEY_LEFT", "KEY_OK", "KEY_RIGHT", "None"],
    ["KEY_VOLUMEDOWN", "KEY_DOWN", "KEY_CHANNELDOWN", "None"],
    ["None", "None", "None", "KEY_MODE"]
]
KEYPAD2 = [
    ['KEY_0', 'KEY_1', 'KEY_2', 'KEY_3'],
    ['KEY_4', 'KEY_5', 'KEY_6', 'KEY_7'],
    ['KEY_8', 'KEY_9', 'None', 'None'],
    ['KEY_POWER', 'KEY_HOME', 'KEY_BACK', 'study']
]
COL_PINS1 = [2, 3, 4, 10]    # BCM numbering
ROW_PINS1 = [24, 23, 15, 14]    # BCM numbering
COL_PINS2 = [9, 11, 5, 6]    # BCM numbering
```

```
ROW_PINS2 = [12, 7, 8, 25]    # BCM numbering
factory1 = rpi_gpio.KeypadFactory()
factory2 = rpi_gpio.KeypadFactory()
keypad1 = factory1.create_keypad(keypad=KEYPAD1, row_pins=ROW_PINS1,
col_pins=COL_PINS1, key_delay=200)
keypad2 = factory2.create_keypad(keypad=KEYPAD2, row_pins=ROW_PINS2,
col_pins=COL_PINS2, key_delay=200)
keypad1.registerKeyPressHandler(prin       tKey)
keypad2.registerKeyPressHandler(printKey)
'''
1 13 16 19 20 21 26
'''
OUT_PINS = [20, 13, 16, 19, 1, 21, 26]
GPIO.setup(OUT_PINS, GPIO.OUT)
GPIO.output(OUT_PINS, GPIO.LOW)
study = False
press_key = None
li = [
      'Press RETURN to continue.',
      'Press RETURN now to start recording.',
      'irrecord: no data for 10 secs, aborting',
      'Please enter the name for the next button (press <ENTER> to finish
recording)',
      'The last button did not seem to generate any signal.\nPress RETURN
to continue.',
      pexpect.EOF,
      pexpect.TIMEOUT
]

# ****************************************#
study_key = []
remote_name = None


def update(file_str, remote):
```

```
        exits_key = []
        param = []
        param.append(remote)
        param.append(os.path.abspath(remote))

param.append(datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S"))
        param.append(0)
        value = '(?,?,?,?,'
        sql = 'insert into T_XUPT_DEVICE (remote, path,create_time,status,'
        for i in study_key:
            if i in file_str:
                if i in exits_key:
                    continue
                exits_key.append(i)
                try:
                    sql += key_map[i] + ','
                except:
                    sql += i + ','
                param.append(1)
                value += '?,'
        if len(param) == 4:
            return False
        sql = sql[0:-1] + ')'
        value = value[0:-1] + ')'
        sql = sql + 'values' + value
        exits_key = []
        conn = sqlite3.connect('../db.sqlite3')
        cursor = conn.cursor()
        try:
            cursor.execute(sql, param)
            conn.commit()
            return True
        except:
            print(traceback.print_exc())
        print(sql)
```

```
    return False


# ****************************************#

# printKey will be called each time a keypad button is pressed

if __name__ == '__main__':

    try:
        while True:
            if study:
                remote_name = None
                # 保存当前设备的名字（当前设备的名字初始化为 none）
                GPIO.output(OUT_PINS[0], GPIO.HIGH)
                # 进入学习状态使 A 灯变亮
                os.system('sudo /etc/init.d/lirc stop')
                # 录制之前关闭 lirc
                name = str(uuid.uuid1())
                # 先获取一个 uuid 字符串
                p = pexpect.spawnu('sudo irrecord -f -d /dev/lirc0 ' + name,
logfile=sys.stdout)
                # 执行这条录制命令
                while True:
                    # 循环匹配，因为录制命令是持续的状态
                    index = p.expect_exact(li, timeout=200)
                    # 匹配 li 中可能结果返回对应的索引
                    if index == 0 or index == 1:
                        p.sendline()
                    #       如果索引是 0 或者 1 的时候给 lirc 发送回车
                    elif index == 2:
                        pass
                    #       如果索引是 2 的时候不做任何操作
                    elif index == 3:
                        # 如果索引为3，lirc 提示接收按键值，让 B 灯亮
```

```
                    GPIO.output(OUT_PINS[1], GPIO.HIGH)
                    try:
                        while GPIO.input(OUT_PINS[1]):
                            # 只要B灯一直亮着（对应的引脚一直是高电平），则循环判断全局变量study的状态
                            if not study:
                                # 如果按下study 给lirc发送回车，退出学习
                                p.sendline()
                                GPIO.output(OUT_PINS[1], GPIO.LOW)
                                GPIO.output(OUT_PINS[2], GPIO.LOW)
                                # 让BC灯接的引脚变为低电平
                                continue
                    except Exception:
                        # 程序出现异常退出匹配
                        print("exception")
                        break
                    p.sendline(press_key)
                # 发送当前按键值
                elif index == 4:
                    # 发送了按键值，但是没有接到红外信号，C灯亮
                    GPIO.output(OUT_PINS[2], GPIO.HIGH)
                elif index == 5:
                    # 索引为5，录制命令结束
                    if os.path.exists(name):
                        # 判断是否生成该设备，没有生成则直接退出，学习成功则
                        # 将学习完成的一些信息持久化到sqlite3数据库中
                        device = codecs.open(name).read().decode()
                        # 获取当前设备中的内容
                        iscp = update(device, name)
                        # 调用持久化函数，参数为设备内容以及设备名字
                        if iscp:
```

```
                                    # 如果存到数据库成功的话，将设备复制
到/etc/lirc/lircd.conf
                                    # 重启 lirc
                                    os.system('sudo cp ' + name + '
/etc/lirc/lircd.conf')
                                    os.system('sudo /etc/init.d/lirc start')
                                    remote_name = name
                                #           保存当前设备的名字
                                    break
                            elif index == 6:
                                # 如果无法匹配到结果，并超时，则重新继续等待
匹配
                                    pass
                        # 学习结束，A 灯灭，全局变量 study 取 False
                        print('end')
                        study = False
                        GPIO.output(OUT_PINS[0], GPIO.LOW)
                    else:
                        if press_key is not None:
                            # 如果当前按键值不是 None 才执行，是 None 说明按键
没有用
                            if remote_name is None:
                                # 如果当前设备名字是 None（即不知道当前设备的
名字）
                                result = subprocess.getoutput('sudo irsend list ""
""')
                                # 获取设备名字
                                if 'refused' in result:
                                    # 重启 lirc
                                    os.system('sudo /etc/init.d/lirc restart')
                                    os.system('sudo lircd start')
                                    os.system('sudo lircd -d /dev/lirc0')
                                    result = subprocess.getoutput('sudo irsend list
"" ""')
                                #       再次获取获取设备名字
```

```
                    if len(result) != 0:
                        remote_name = result.split(':')[1]
                #           如果设备名字返回成功则保存当前设备名字
                    result = subprocess.getoutput('sudo irsend SEND_ONCE
' + remote_name + ' ' + press_key)
                    # 发送红外
                    if len(result) == 0:
                        # 发送成功没有返回信息，所以长度为0，D 灯闪一
下
                        GPIO.output(OUT_PINS[3], GPIO.HIGH)
                        time.sleep(0.3)
                        GPIO.output(OUT_PINS[3], GPIO.LOW)
                        print(press_key, 'suc')
                    else:
                        # 长度不为0 说明红外发送失败
                        print(press_key, 'error')
                    press_key = None
        except:
            # 运行报错退出，恢复原 GPIO 状态
            print(traceback.print_exc())
            GPIO.cleanup()
        GPIO.cleanup()
```