封面被删除

无意义内容被删除，仅保留需求、技术内容。

# 系统需求分析与概要设计

## 系统需求分析

### 系统简介

“基于物联网的家庭电器智能控制系统”主要包含以下模块：

1. 用户客户端：为用户提供多个获取家庭环境信息和控制设备的入口，允许用户对家庭电器做自定义的配置；
2. 管理员后台：作为系统的管理后台，便于维护人员保障系统运行；
3. 服务端：处理各种前端入口的用户请求，作为联系用户与硬件设备的桥梁中转数据；
4. 智能硬件：能够检测环境数据，与服务器建立通信信道，进行数据交互。

总的来说，需要设计和实现一个安全可靠、智能、价格低廉且具有较高定制性和可玩性的智能家居硬件系统。

### 角色分析

系统主要的使用对象如下：

1. 普通用户（消费者）：可以注册、登录账号，账号是用户使用本智能家居系统的凭据。在APP等多个控制入口内可以绑定智能设备，查看设备上报的环境数据，向设备发出控制指令，以及定制智能化任务等；
2. 普通管理员：能够在管理后台查看用户数量，设备活跃度，新增/修改产品，新增/修改产品下的设备，审计系统Api的请求日志；
3. 超级管理员：除拥有普通管理员的权限外，超级管理员还可以增加普通管理员。

### 业务功能分析

智能家居客户端（安卓和iOS平台APP、微信小程序和网页）的主要功能点为：

1. 用户打开时提示需要注册并登录账号；
2. 登陆后可以绑定配套的智能设备，绑定流程为：扫描附近可绑定的设备，用户选择其一，在用户完成无线网络名称和密码后，将其发送给智能设备，连接成功后上报服务器，即完成设备绑定；
3. 绑定后即可在客户端首页看到所绑定的设备快捷控制块，基于不同的产品，该控制按钮上展示的内容也有所不同，如温湿度传感器展示上一次上报的数据，继电器这类接收控制指令的设备快捷控制块则展示操作按钮，点击后能发送常用的控制指令。所有产品都会展示产品图片和设备在线状态；
4. 点击整个快捷控制块，将进入设备详情页面。对于不同产品，该页面内容也有所不同；
5. 客户端首页右下角放置一个发起语音控制的按钮，在用户点击按钮后开始录音，并在用户说话时给予实时反馈。用户有稍长的停顿即认为结束控制指令，此时将指令发给服务端解析执行并返回语音反馈，于客户端播放；
6. 客户端下方第二个Tab是设备智能化，该页内部分为左右两个标签页，左侧为用户配置的自动化任务，展示的每条任务配置包含配置名、前置判断条件、执行动作和该配置是否启用；
7. 智能化Tab右标签是用户设置的自动化任务执行记录，用户可以在该页了解配置的生效情况；
8. 客户端下方第三个Tab是个人中心，上方展示用户名和用户拥有的智能设备数量。该页面底部有修改密码和退出登录按钮；
9. 修改密码需要提供原密码鉴权，新密码需要重复输入来确认，注册亦然。

管理员后台的主要功能点为：

1. 后端登录页仅提供登录功能；
2. 登陆成功后进入后台首页，首页布局为左侧菜单栏，右侧顶部导航栏，右侧下方大面积区域用于各子菜单页面内容展示。首页默认内容为一些运维数据的看板；
3. 后台提供人员管理功能，管理员可以查看注册的普通用户基础资料、绑定的智能设备，可以为普通用户重置密码。超级管理员还可以新增普通管理员；
4. 后台提供智能产品的新增/管理功能，在完成新产品的研发后在此添加为系统可绑定产品，也可以修改产品的名称/启用状态，不允许删除产品。
5. 后台提供智能硬件的新增/管理功能，在完成设备的生产后，在此将新增设备（依赖于产品）供用户绑定。管理员还可在该页面查看设备的绑定情况，必要情况下还可以修改或解除设备的绑定状态；
6. 后台提供Api接口的请求日志记录/查询功能，用于排查系统故障或审计；对于部分敏感数据要脱敏存储

后端服务的主要功能点为：

1. 前后端分离，与前端的数据交互使用json，接口的设计要符合RESTful风格；
2. 选用可靠的接口鉴权方式；
3. 敏感信息（如密码等）选用可靠的加密存储方式；
4. 使用MQTT协议与智能设备建立稳定、便捷、快速且低功耗的通信链路；
5. 收集智能设备上报的环境数据，向设备转发用户发出的控制指令

智能硬件的主要功能点为：

1. 支持动态配网，而不是将需要的无线网络名称、密码等信息硬编码在固件中；
2. 支持OTA更新固件；
3. 与服务器建立通信链路，定时上报环境信息，接受服务器为用户转发或因自动任务直接发送的控制指令，并解析执行；

## 系统概要设计

### 系统整体架构

“基于物联网的家庭电器智能控制系统”包含多个子系统，总体结构如下图所示：

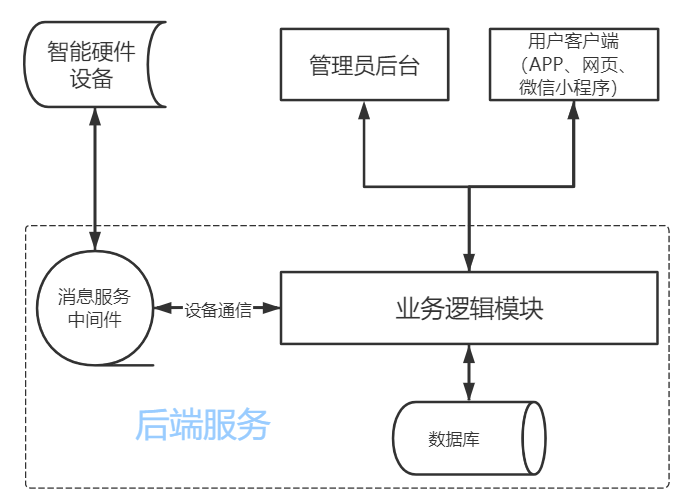


图 2‑1系统总体结构

### 各子系统模块架构

后端服务系统是系统的运行基础，包含数据库连接层，业务层和对外接口层，系统内部会对各类请求记录日志。架构如下图所示：

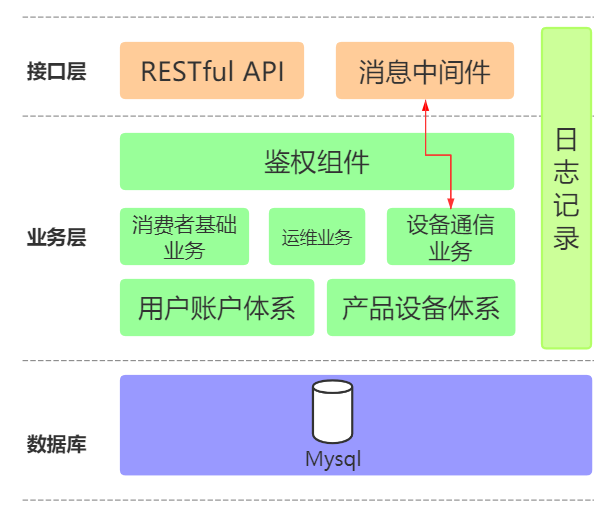


图 2‑2后端服务架构

智能硬件的固件包含以下主要功能：接收绑定设备时用户发送的无线网络配置信息并连接，与服务器建立通信链路，定时上报设备信息，随时接收控制指令并解析执行。具体组成模块如下图所示：

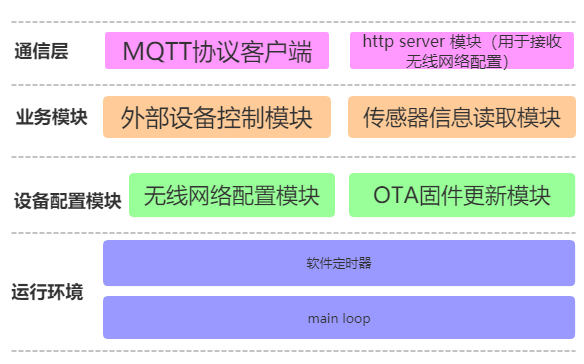


图 2‑3智能硬件架构

管理员后台是一个WEB页面，该子系统的组成模块如下图所示：

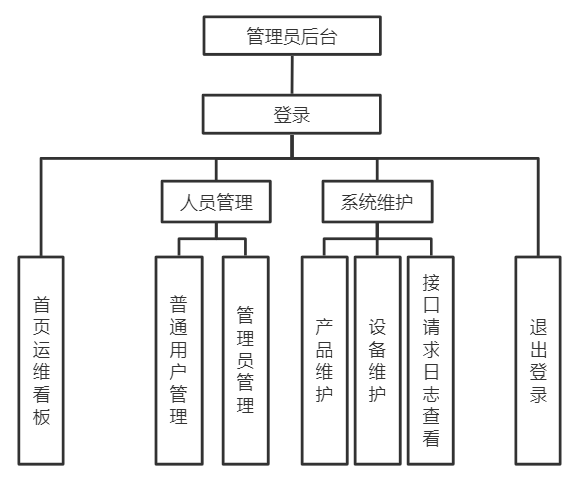


图 2‑4管理员后台架构

消费者客户端分为多平台，界面风格统一，但部分功能会因平台能力差异而有所不同。大致的功能模块组成如下图所示：

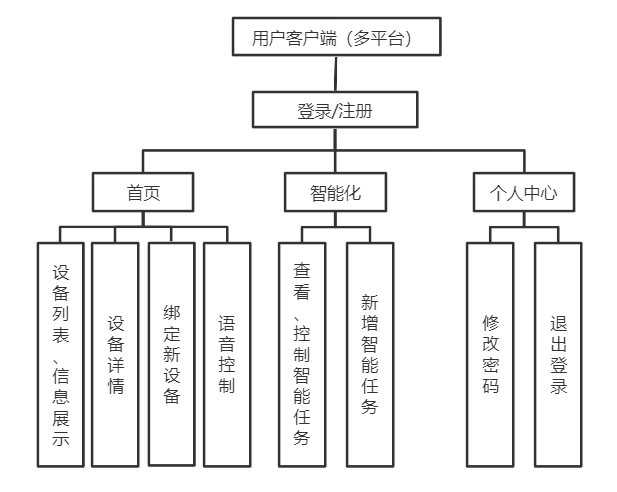


图 2‑5用户客户端功能架构

## 可行性分析与技术选型

### 可行性分析

当前智能家居市场上存在着种类繁多的智能家居产品，其中以家居安防、传感器、生活起居、娱乐影音等几类最为常见。要实现一套用户体验良好的智能家居体系，需要具备一些基本要素。下面就根据如何这些基础要素给出初步可行性分析，对于详细的技术选型将在下节进行讨论。

1. 使用便利性

用户对产品产生印象的主要来源是使用过程的操作体验，例如在开始使用设备时，应能快速简易地将设备成功安装，这需要客户端内给与用户足够的引导和操作暗示[7]；在设备的后续使用中，应能实现随时随地查看、控制智能设备（前提是设备在线）。在技术方面，在智能设备配网时，可以提前获取用户设备接入的网络名称作为设备工作时使用的默认无线网络名称；连接到室内无线网络后，设备可以通过连接到服务端的消息中心，以MQTT这一物联网常用的轻量级通信协议来建立通信链路，从而实现更高效、低功耗、接近实时地上报设备信息、接收设备控制指令并解析执行[8]；

1. 硬件稳定性

在设备工作时，难免遇到无线网络不稳定、突然断电等情况，因此在开发设备固件的过程中应为种可能的意外事件做好处理方式。例如网络中断时应暂停设备信息上报（减少传感器读取以降低功耗），定时尝试重新接入网络；

1. 系统安全性

智能硬件与消费者的生活起居距离非常近，能接触到用户的大量隐私数据，因此系统的安全性就显得尤为重要。在账户体系中，用户密码绝不应被以明文的方式保存；系统的各个业务接口鉴权时应该严格校验用户是否拥有其要操作的资源，避免水平权限漏洞；管理员后台还要注意区分用户角色来鉴权，避免垂直权限漏洞[9]；

### 开发技术选型

上一节从本系统的产品体验方面大致讨论了几个技术方面的可行性，更为具体地，下面给出经过一系列调查研究后确定的系统开发技术选型。

1. 后端服务

基础业务部分，使用基于Python的Flask来快速成型。Flask是一个简单易学、拓展性极高的WEB开发微框架，可以轻松结合大量可靠的第三方库来使用，比如用Flask\_sqlalchemy 加入ORM功能，Flask-RESTful提供了快速构建 RESTful API的支持[10]。不过，Flask也有一些缺陷，比如它并不适合直接作为大规模生产环境的后端服务器来使用，对读取大型资源文件的请求的响应性能不够好，所以现在一般会配合Nginx来使用[11]，组成如下图所示的四层服务架构

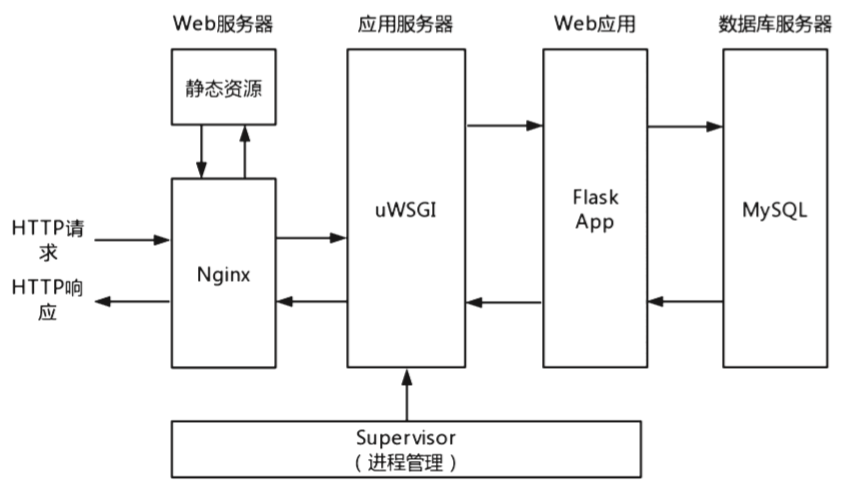


图 2‑6四层服务架构

Nginx对外开放端口，当客户端发起请求时由Nginx进行分流，静态请求直接由Nginx返回，业务请求则按配置文件转发给uWSGI。uWSGI是Flask的基础，会按请求 URL将其分发到对应的Flask App函数进行处理，得到返回结果后将其传递给Nginx并返回客户端[12]。Supervisor是Linux上独有的uWSGI进程管理工具，可在其进程异常终止时及时拉起，以保障后端服务的可用性。如需进一步优化并发性能，还可用Gunicorn + Gevent来替代uWSGI。

本系统选择以RESTful API作为服务框架对外的唯一接口。API是一系列预定义的函数，承担着系统间数据交互的任务，RESTful是一种接口设计风格，它将所有的请求抽象成对特定资源的操作，每个请求中应包含功能描述。RESTful 风格的API具有一系列优点，例如它能更高效地利用缓存以优化性能，对软件的依赖性较小，在软件演化的进程中有着长期的兼容性[13]。

前后端数据交互的过程中，本设计选择JWT（JSON Web Token）作为鉴权载体，这是目前较为流行的认证方案。它轻量且安全可靠，还能能够按需承载 payload，方便轻松满足前端较为简单的业务需求。例如，需要区分用户角色时，在payload 添加角色字段即可，基于JWT的加密特性，可以认为这是一种安全可信的解决方案[14]。

在本设计中，消息服务中间件起到连接后端服务与智能硬件的通信桥梁角色，我选择了ActiveMQ这一性能强劲且十分流行的消息系统。ActiveMQ启动后监听特定端口，后端服务和智能硬件以MQTT协议与其分别建立连接后，即可通过订阅/发布的模式（如下图所示）收发各自需要的信息。例如某硬件启动并联网后，订阅deviceControl/tempAndHumi7933F6这一消息主题，另外定时向deviceReport/ tempAndHumi7933F6这一消息主题上报硬件信息，而后端服务方面可以通过订阅deviceReport/#（#代表匹配主题下任意层次的消息）来接收所有设备的上报信息，通过向deviceControl/tempAndHumi7933F6发布设备控制指令就能使指定设备收到指令并执行。

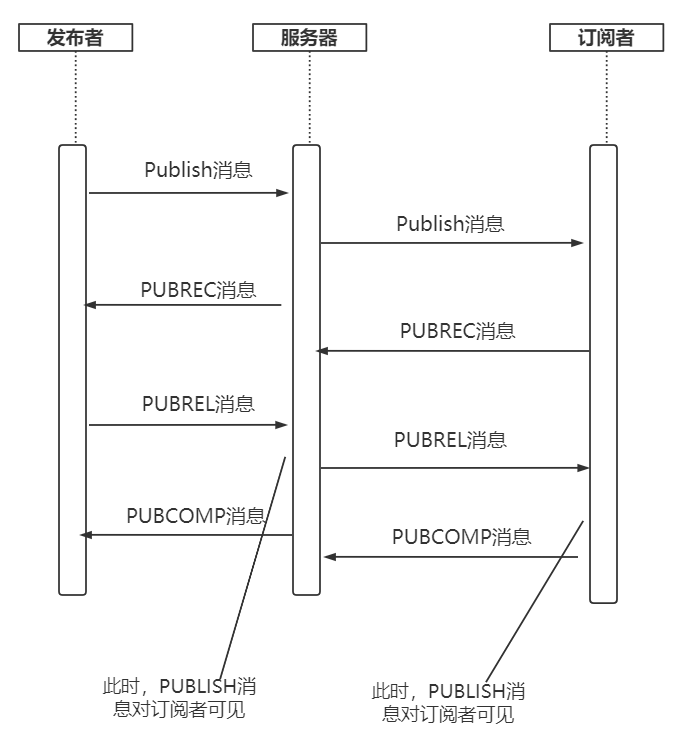


图 2‑7发布/订阅模式

MQTT协议（Message Queuing Telemetry Transport，消息队列遥测传输协议），它是ActiveMQ支持的众多协议之一，简洁轻量，易于实现，在传输过程中力求将数据量降到必要的最低数，还提供了QoS（服务质量管理）功能，非常适合用在智能家居这类受限于代码体积和硬件能耗，有时网络条件不稳定的产品上。

1. 智能硬件

本设计中需要基于智能家居服务框架实现两个智能硬件产品：温湿度传感器和继电器控制器（可以用于控制灯具等各类家用电器）。经过研究，我发现基于ESP8266的最小系统即可实现以上功能，无需使用额外的控制器，能够降低硬件成本、功耗，提高系统稳定性。ESP8266是上海乐鑫于2014年推出的一款基于WiFi高度集成的面向物联网应用的控制芯片，它体积小巧，性价比高，支持完整TCP/IP协议栈，支持SoftAP、Station和两者共存的工作模式，对于本设计的智能家居硬件开发非常合适[15]。ESP8266上可用的引脚定义如下表：

表 2‑1模块ESP8266可用引脚说明

|  |  |
| --- | --- |
| 引脚名 | 说明 |
| RX | 接收数据；如设置为输入输出引脚，则为GPIO3 |
| GPIO0 | 工作模式选择，悬空时正常启动，低电平时为下载模式，可烧写固件 |
| GPIO2 | 开机上电时必须为高电平；是WiFi工作状态指示灯 |
| GND | GND |
| VCC | 3.3V供电，不可直接5V供电 |
| Reset | 低电平时复位 |
| CP\_PD | 高电平时工作，低电平会关掉供电 |
| TX | 发送数据；如设置为输入输出引脚，则为GPIO1；开机上电时禁止下拉 |

在硬件实现上，使用DHT11这已校准温湿度复合传感器提供温湿度的数字信号输出，由ESP8266定期从GPIO引脚读取数值，解析后上报到消息中间件，后端服务收到消息后存储归档。继电器控制器的实现，则是在ESP8266初始化时将一个GPIO设定为输出模式，即可使用该引脚作为ESP01-Relay继电器控制信号，当ESP8266接收到来自消息中间件的设备控制信号时，按指令修改该引脚的电平就能实现控制继电器开闭，进而控制用电器。

智能硬件的固件开发上，我选择基于Arduino IDE，可以方便地调用诸如解析DHT11信号的库，能显著加快开发速度。

1. 管理员后台

该设计中为管理员提供一个WEB的运维后台，基于对前端技术的了解，我选择使用了Vue.js框架和ElementUI组件库来实现它。使用 Axios 发起请求调用后端服务接口，这是一种前后端分离的开发模式，有助于提高开发效率，帮助前端开发更关注组件化、性能优化、快速迭代。

vue.js 是一套渐进式的前端开发框架，以其易用、灵活且高效的优点而广受欢迎。区别于其他重型前端开发框架，Vue 在设计上是基于数据驱动的，在使用它的过程中是较为符合思考逻辑的[16]。从最初刀耕火种直接操作 DOM 的年代，到后来逐渐出现的 jQuery、Angular 等等前端开发库，前端技术逐渐向组件化、业务化发展，大大简化了开发复杂度，也能减少出错概率。Element UI 是一套基于 Vue 2.0 的桌面端组件库，提供了配套设计资源，助力网站快速成型[17]。Axios 是一个基于 promise 的 HTTP 库，在现在前端开发中较为常用。

1. 用户客户端

本设计预计提供多个消费者使用的客户端作为设备控制入口，例如手机APP、微信小程序、H5网页，要分别实现这些平台的适配其实难度并不高，毕竟可以复用大量代码，但这样做无疑会带来巨大的工作量。经过调研了解到，现在技术上有多种只需一次开发即可打包发布到多种平台的开发框架，uni-app就是其中之一，它是一个允许开发者使用Vue.js开发多种前端应用的框架。开发者只需编写一次代码，配合框架提供的条件编译，即可生成 iOS、安卓、网页以及微信/QQ平台小程序等多个平台的应用[18]。即使不用于跨端适配工具，它同时也是一个极好的微信小程序开发框架。

选择uni-app+Vue.js的跨端开发模式后，其他内容与管理员后台的开发技术选型类似。

## 本章小结

本章的主要内容是在拆分本设计的各个子系统的过程中，从多角度分析了系统需求，按前期调研进行了可行性分析，并给出了技术选型思路。

# 系统详细设计与实现

## 后端服务的设计与实现

系统流程在第二章系统需求分析里已被详细介绍过，这里不再赘述，这里直接从数据库表结构设计和后端关键部分代码实现开始。

### 数据库表结构设计

1. 用户表 users

用户表存储着用户的基本信息，表结构如下：

表 3‑1用户表字段

|  |  |  |  |
| --- | --- | --- | --- |
| 字段 | 注释 | 类型 | 长度 |
| user\_id | 用户id，主键 | int | 11 |
| username | 用户名 | varchar | 64 |
| password\_hash | 用户密码hash结果 | varchar | 128 |
| user\_role | 用户角色 | varchar | 128 |
| user\_setting | 用户设置 | varchar | 10240 |
| create\_at | 注册时间 | datetime | 0 |
| updated\_at | 上次修改时间 | datetime | 0 |

1. 硬件产品表 hardwareProduct

智能家居产品与产品的关系：研发出一类产品（product）后将批量生产得到硬件设备(device)。产品表的表结构如下：

表 3‑2产品表字段

|  |  |  |  |
| --- | --- | --- | --- |
| 字段 | 注释 | 类型 | 长度 |
| product\_id | 产品id，主键 | varchar | 256 |
| product\_name | 产品名称 | varchar | 256 |
| smart\_provide\_param | 在智能化时该产品能提供哪些输出参数 | varchar | 2048 |
| smart\_accept\_action | 在智能化时该产品能接收哪些输入操作 | varchar | 2048 |
| latest\_firmware\_version | 最新固件版本 | varchar | 2048 |
| is\_enable | 是否启用 | tinyint | 1 |
| create\_at | 创建时间 | datetime | 0 |

1. 硬件设备表 hardwareDevice

用户购买到的是硬件设备(device)，每个设备属于一种产品（product）。设备表的表结构如下：

表 3‑3硬件设备表字段

|  |  |  |  |
| --- | --- | --- | --- |
| 字段 | 注释 | 类型 | 长度 |
| device\_id | 设备id，主键 | varchar | 256 |
| product\_id | 产品id，外键到hardwareProduct. product\_id | varchar | 256 |
| bind\_to\_user\_id | 当前绑定的用户，外键到users.user\_id | int | 11 |
| device\_config | 设备配置信息 | varchar | 10240 |
| create\_at | 创建时间 | datetime | 0 |
| updated\_at | 上次修改时间 | datetime | 0 |

1. 设备信息上报表 deviceReportRecord

存储智能设备定时上报的信息（如环境温度，设备状态），表结构如下：

表 3‑4设备信息上报表

|  |  |  |  |
| --- | --- | --- | --- |
| 字段 | 注释 | 类型 | 长度 |
| log\_id | 日志id，主键 | int | 11 |
| device\_id | 设备id，外键到hardwareDevice.device\_id | varchar | 256 |
| log\_type | 日志类型 | varchar | 256 |
| log\_detail | 记录详情 | varchar | 10240 |
| create\_at | 创建日期 | datetime | 0 |

1. 智能化配置表 deviceSmartConfig

存储用户设置的智能化配置，表结构如下：

表 3‑5智能化配置表结构

|  |  |  |  |
| --- | --- | --- | --- |
| 字段 | 注释 | 类型 | 长度 |
| config\_id | 配置id，主键 | int | 11 |
| config\_name | 配置名称 | varchar | 128 |
| set\_by\_user\_id | 所属用户id，外键到users.user\_id | int | 11 |
| smart\_config | 配置详细信息 | varchar | 10240 |
| is\_enable | 是否启用 | tinyint | 1 |
| create\_at | 创建时间 | datetime | 0 |

1. 智能化执行记录表 deviceSmartLog

存储用户的智能化任务执行记录，表结构如下：

表 3‑6智能化执行记录表结构

|  |  |  |  |
| --- | --- | --- | --- |
| 字段 | 注释 | 类型 | 长度 |
| log\_id | 日志id，主键 | int | 11 |
| config\_id | 配置id，外键到deviceSmartConfig.config\_id | int | 11 |
| set\_by\_user\_id | 所属用户id，外键到users.user\_id | int | 11 |
| create\_at | 创建时间 | datetime | 0 |

1. 后端接口请求日志记录表 apiAccessLog

存储后端Api的脱敏后的请求记录，用于系统debug和运维分析等需求，表结构如下：

表 3‑7后端接口请求日志记录表结构

|  |  |  |  |
| --- | --- | --- | --- |
| 字段 | 注释 | 类型 | 长度 |
| id | 记录id，主键 | int | 11 |
| user\_id | 所属用户id，外键到users.user\_id | int | 11 |
| access\_url | 请求的url | varchar | 2048 |
| access\_method | 请求方法 | varchar | 64 |
| access\_body | 请求数据，脱敏后 | varchar | 10240 |
| from\_ip | 来源id | varchar | 64 |
| create\_at | 创建时间 | datetime | 0 |

### 后端部分关键点技术实现

1. 接口鉴权与请求日志记录

后端服务提供了大量接口，且不同接口对于不同角色的用户授权也不一致，若按用户角色强行拆分，则难以满足RESTful的API风格，所以我想到在JWT（JSON Web Token）所承载的payload中增加用户角色字段，在Flask-restful的各接口资源方法（resource）上加一层装饰器，下面分别介绍这两项技术要点。

JWT是一个用于网络应用间的轻量级鉴权承载方式，不需要在服务端保持会话信息（无状态），有利于系统水平扩容。

装饰器（decorator）是Python语言中面向切片思想的一种应用，能给被包装的函数增加一层方法，在目标函数被调用时提前调用，这一特点非常适合用于给函数添加功能或接口鉴权[19]。具体实现上，我准备了一个用于JWT鉴权的类，大致代码结构如下：

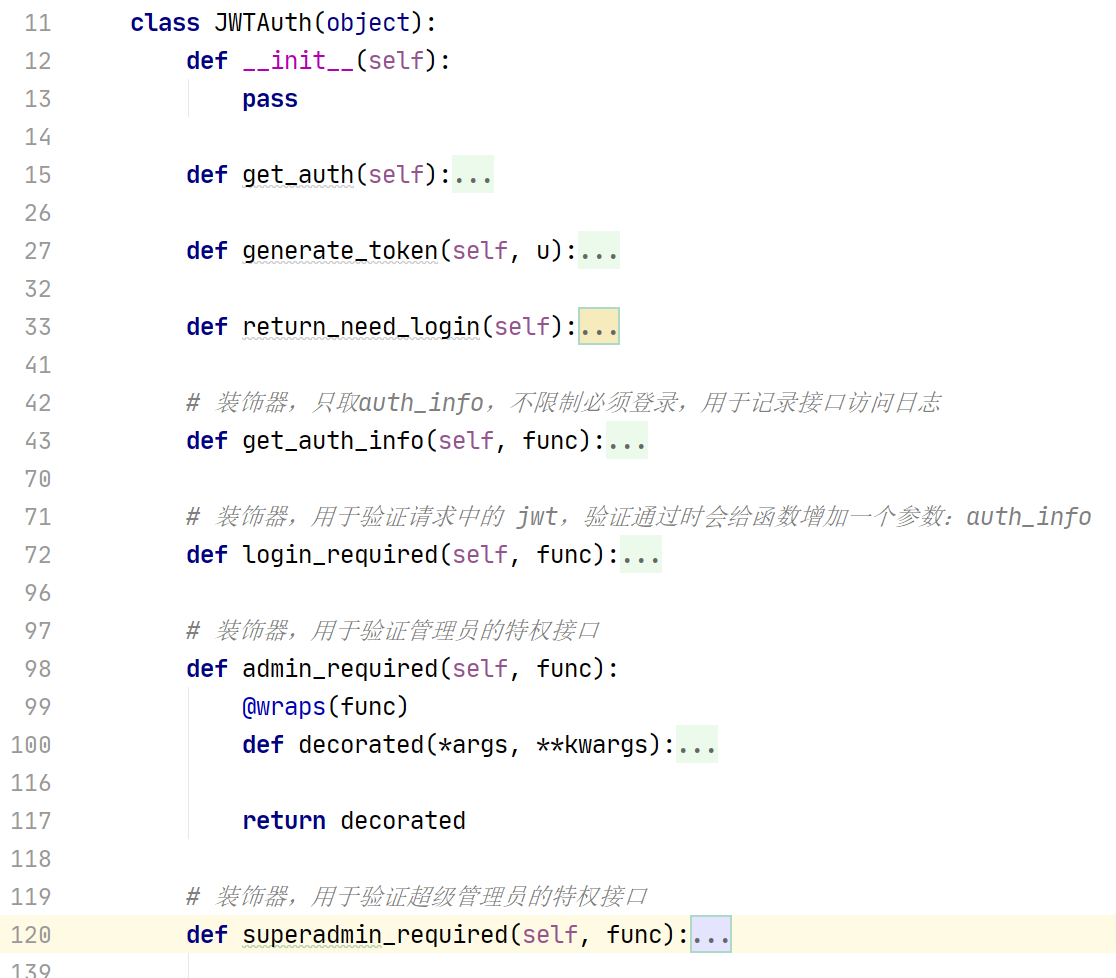


图 3‑1使用装饰器JWT鉴权代码结构

以登录鉴权为例，该装饰器逻辑如下：



图 3‑2以登录鉴权代码逻辑为例

对于全部接口请求日志的记录，在自行编写的记录函数外包裹@ app.before\_request和 @ jAuth.get\_auth\_info 装饰器即可实现记录每一个对后端api的请求。

1. ORM框架、用户密码存储

对于使用了数据库的系统，在开发过程中直接编写sql查询语句的风险和难度都是极大的，一方面开发人员可能忽视了对sql注入的防范，另一方面，缺乏经验的开发人员可能会写出性能极差的查询语句，甚至从数据库获得错误的结果。而ORM（对象关系映射）将开发人员从繁重的sql语句编写中解放了出来，它能使程序像操作原生对象一样对数据库内的对象进行处理，一张表抽象成一个类，而每一行则成为一个对象[20]。

在Flask中使用ORM是非常简单的，只需要像下图这样初始化flask\_sqlalchemy即可：



图 3‑3使用ORM框架

数据库表与Python对象的映射关系存储在/models中，这里以users表的模型（如下图）举例，同时给出本设计中对用户密码的处理方式。



图 3‑4以users表为例展示对象映射

一个对用户负责的系统绝不应该将用户敏感信息（尤其是密码）明文存储，简单的加密或摘要算法（如单纯md5）也是不可取的。在本设计中，我选择使用werkzeug.security提供的generate\_password\_hash来计算用户密码的摘要来存储，该方法对用户同一密码每次摘要生成的密文都不一样，能避免hash碰撞和彩虹表攻击。在密文不同的情况下，每次用户登录时就无法自己通过对比字符串的方式判断密码是否正确了，需要使用该包提供的check\_password\_hash方法传入正确hash和待测试密码来校验。

1. 硬件智能化任务的设计

硬件智能化（联动）抽象来看就是if-then，从数据库硬件产品表的表结构设计中可以看到，产品有着智能化配置可提供输出参数、智能化配置可接收的输入操作。这里以温湿度传感器为例，它可提供的输出参数如下：

[

{

"type": "int",

"field": "temperature",

"description": "温度",

"unit": "摄氏度"

},

{

"type": "int",

"field": "humidity",

"description": "湿度",

"unit": "%"

}

]

无可接收的输入操作。

继电器控制器可输出的参数如下：

[

{

"type": "bool",

"field": "relayStatus",

"description": "开关状态（自动转换）",

"unit": ""

}

]

可接收的输入操作如下：

{

"paylay": "{\"command\":\"relayControl\",\"relayStatus\":\"#param1#\"}"

}

在智能化配置记录表中，存在类似这样的智能化配置：

{

"product\_id": "tempAndHumi",

"product\_name": "温湿度监测器",

"device\_id": "tempAndHumi7933F6",

"field": "temperature",

"fieldName": "温度",

"operator": ">",

"rightValue": "29",

"rightUnit": "℃",

"actionDeviceId": "relayControl79CC49",

"actionProductId": "relayControl",

"actionProductName": "继电器",

"payloadKeyword": "闭合",

"payloadKeywordUnit": "",

"actionPayload": "{'command':'relayControl','relayStatus':'on'}"

}

基于如上配置，后端服务每次接收到设备上报的状态信息时，读取该设备关联的用户所设置的全部状态为已启用的智能化配置记录，逐一对比device\_id，如果有当前上报信息设备的配置，则按如下方式，将智能化配置转换为Python表达式，来判断是否需要执行操作：



图 3‑5后端对智能化任务的处理

## 智能硬件的设计与实现

硬件的基础的开发部分除了因为刚接触硬件开发导致多花了些时间外，没有太多难点，这里只记录硬件的启动流程和一些技术难点的实现和问题排查过程。

### 硬件运行过程

要正常启动ESP8266，需要将GPIO0悬空。启动后，本设计的各硬件基本都是如下运行逻辑：

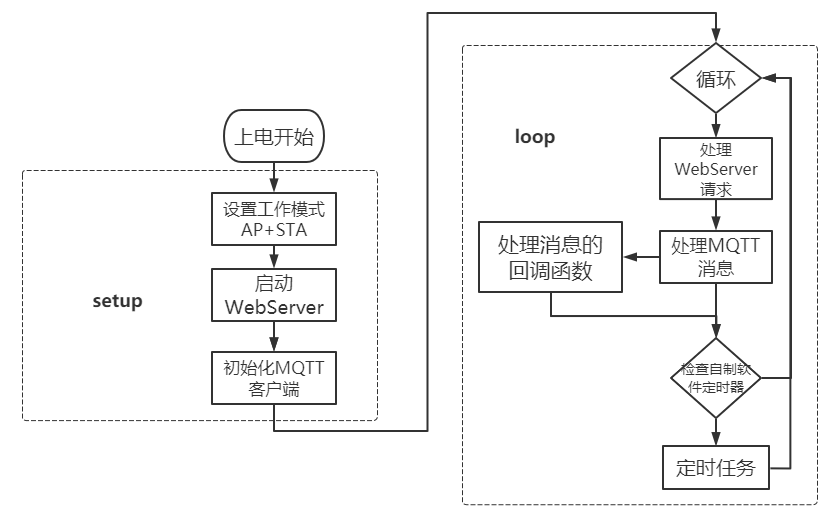


图 3‑6本设计智能硬件的运行过程

### 动态配网的设计与实现

动态配网是本设计中较为重要的一部分，在前期收集资料的过程中看到很多智能家居设计里都将要使用的无线网络名称和密码硬编码在设备固件中，这显然不符合智能家居硬件的实际使用状况。要实现动态配网，有这几种实现方式：

1. Soft AP。智能设备以AP模式运行，用户客户端连接到该AP后向设备发送要使用的无线网络信息；设备端收到网络配置后连接。
2. SmartConfig。这是一种快速配网方式，用户客户端向周围发送包含加密网络配置的UDP组播或广播包，智能设备收到后解析并连接。
3. 智能设备的外部接口（如通过声波、摄像头获取网络信息）。用户客户端在屏幕上展示包含网络配置的二维码，或将配置编码后以音频为载体播放，设备通过摄像头或麦克风获取连接配置后连接。

各方式优缺点对比如下表[21]：

表 3‑8三种配网方式的优缺点对比

|  |  |  |
| --- | --- | --- |
| 配网方式 | 优势 | 缺点 |
| Soft AP | 稳定可靠，成功率高 | 操作复杂 |
| SmartConfig | 操作简捷，用户体验好 | 配网速度稍慢，成功率有时较低 |
| 音视频接口 | 安全可靠 | 需要设备支持，增加设备成本 |

结合本系统的用户客户端实现方式，最终选择了使用Soft AP的方式进行配网，设备上电后设置AP+STA的工作方式，已自身设备ID作为AP名称。

### 在线OTA

设备交付给用户后，就不可能要求用户通过手动烧写的方式升级固件，所以应在系统内提供直接升级的途径。好在ESP8266提供了基于双用户的OTA升级机制，引用ESP8266HTTPUpdate.h库后，调用ESP8266HTTPUpdate::update(const String& url, const String& currentVersion)方法，传入要升级的固件下载地址和当前固件版本（可选）即可完成自动升级。

### 判断设备在线状态的实现

智能家居系统中，需要对硬件设备在线状态做正确的展示，设备上线后发送报告，后端服务能轻松获取上线事件。而设备离线的处理就显得复杂些了，一开始的实现方式是定时发送心跳报告，但这会对后端服务造成较大不必要的负载，对设备功耗控制也不利。查阅资料发现，MQTT协议无法直接获取主题发布者的在线状态，但可以通过如下方式实现在线状态检测：智能硬件初始化MQTT客户端时，通过EspMQTTClient::enableLastWillMessage(const char\* topic, const char\* message, const bool retain)方法设置一个LWT（Last Will and Testament，最后遗嘱消息），该配置包含LWT将发往的主题、消息内容、QoS。在硬件客户端离线（如底层网络IO异常，或未能及时响应Keep Alive间隔内的信息交互）时，消息中间件将向LWT配置的消息主题发送预置的消息，后端服务收到后即可将设备标记为离线状态[22]。

### 硬件设计与实现过程中的故障排查

1. 面包板上运行正常，紧凑设计后硬件出错的问题

为控制硬件体积和精简供电系统，对温湿度传感器、继电器控制器这两个硬件我分别选择了ESP-01+DHT11最小系统板和ESP-01S+Relay最小系统板，但开发中购买的这两个IoT-MCU初代版本均有缺陷。

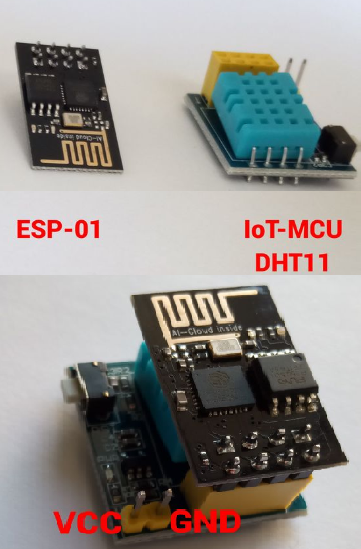


图 3‑7基于ESP-01的最小系统

前者未能处理好CH\_PD引脚的拉高，在表 2‑1模块ESP8266可用引脚说明中可知该引脚是模块供电使能，系统上电后该引脚低电平将导致模块不能正常运作，而在开发板上ESP-01、DHT11各自独立供电时能正常运行，最终通过自行焊接CH\_PD和VCC引脚修复。另外从上图可以看到，ESP-01在DHT11正上方，而ESP-01高负载工作时温度较高，会对温湿度传感器测量造成干扰，故最终通过连线引出连接。

在继电器控制器设计中遇到类似的问题，面包板上各自独立供电时正常工作，ESP-01S插在Relay模块上时出现异常。为了将ESP-01S的信号输出引脚与Relay模块上继电器控制信号引脚对应，我在固件设计时选择GPIO0作为信号引脚，但参见上面引脚说明可知该引脚用于设置系统Boot方式，低电平时将进入下载模式而不是正常启动固件。固件源码中设置引脚为输出脚的pinMode(relayContronGPIO, OUTPUT)显然发生在系统Boot之后，这会在系统上电时Relay模块偶现向GPIO引脚供电出错时导致系统无法正常运行。为了不影响WiFi工作状态指示灯（关联GPIO2），设计中使用了RX引脚（GPIO3）作为继电器控制信号输出引脚。

1. 软件定时器内网络有问题

为实现定时上报设备状态信息，设计中原计划使用ESP8266提供的内置软件定时器的os\_timer\_setfn、os\_timer\_arm来处理定时任务。但在软件定时器的回调函数内，我注意到网络状态时常出现异常的情况，多次debug未果后我选择自行实现软件定时器。使用micros()返回的系统上电时间作为时钟源，在系统每次loop时对比是否达到定时任务的运行间隔来决定是否执行定时任务的回调。实现中注意到该函数返回值为unsigned long的微秒数，因此大约会在70分钟后溢出，代码中对此情况做了较为取巧的处理（直接相减比较，允许结果也溢出），使软件定时器能正常运行。

1. MQTT长消息收发失败

在调试硬件与消息中间件的通信时，我注意到当消息长度达到100字节左右时将会无法正常收发，在临界长度时会给出“MQTT! Your message may be truncated, please change MQTT\_MAX\_PACKET\_SIZE of PubSubClient.h to a higher value.”的提示，检查后发现PubSubClient.h中给MQTT\_MAX\_PACKET\_SIZE默认值仅为128字节（除去协议开销可供消息使用的空间即100字节左右），覆盖MQTT\_MAX\_PACKET\_SIZE后消息收发正常。

## 管理员后台的设计与实现

管理员后台是一个基于Vue.js框架实现的WEB页面，前后端分离，与后端服务使用json格式进行数据交互。这里记录系统实现时涉及到的几个关键技术点。

### 前端路由和鉴权

基于Vue.js框架开发的前端应用其实仅有一个html入口，Vue初始化时挂载于这个html的body根节点，这样形成的应用被称为SPA（Single-page application，一页式网站）。当仅有一个页面时，需要依赖Vue官方提供的vue-router作为页面路径管理系统，它的本质作用是建立url与页面的映射关系。在App.vue的模板中加入<router-view></router-view>，vue-router会按照router配置将当前url转换为需要展示的页面并替换router-view节点。

前端应用的源码（也可能是高度压缩混淆的）是直接暴露给用户的，也即任何用户都能看到全部页面的源码（也有一些措施来避免，例如加密分包名和动态添加路由，但这里仅讨论通常实现），也能尝试渲染、操作可能并不具备访问权限的页面。所以用户访问页面时的登陆状态鉴权、对不同用户区分不同页面权限就显得尤为重要，好在vue-router可以通过导航守卫（router.beforeEach）来实现在页面跳转/加载时加入自定义的鉴权逻辑。例如系统后端页面（除登录外）都需要用户持有登录信息，伪代码实现如下：



图 3‑8前端路由跳转时的鉴权逻辑伪代码

### 前端开发与部署时的跨域问题

跨域是一个浏览器问题，它是浏览器的众多安全策略之一。广义的跨域包括资源的跳转、嵌入和JavaScript发起的网络请求或DOM跨域操作，通常前端开发中遇到的是跨域是指受SOP（Same origin policy，同源策略）限制使得JavaScript无法发送网络请求的情况。同源策略避免了在服务器不知情的情况下跨域访问，如果缺少了它，WEB浏览器将极易受到XSS、CSRF等攻击的威胁[23]。

跨域问题有多种解决方案，在本系统的开发和发布过程中分别使用了Node.js中间件代理跨域、Nginx反向代理跨域两种方式。在通过vue-cli初始化的前端项目中，可在vue.config.js找到devServer配置项，它基于http-proxy-middleware实现，能处理开发过程中JavaScript发来的网络请求（不写host则默认当前host），并按配置转发到指定接口，读取响应后作为本host的返回，对前端浏览器而言未发生跨域行为。在发布环境，本设计使用Nginx的反向代理功能来实现转发API请求，参考配置如下：

location /SHSSAPI {

proxy\_pass http://server-host:port; #后端服务接口

proxy\_set\_header Host $host;

proxy\_set\_header X-Real-IP $remote\_addr;# 用于后端服务记录来源IP

proxy\_set\_header REMOTE-HOST $remote\_addr;

proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

}

location /shss {

expires -1;

root html; # 前端项目打包后的目录

try\_files $uri $uri/ /index.html;

}

## 用户客户端的设计与实现

因为本设计中要提供的控制入口众多（如安卓和iOS平台的APP、微信小程序、H5页面等），在实现多端适配时我选择了DCloud研发的uni-app跨端开发框架。这里介绍开发中一些关键技术问题的处理。

### 跨端适配

Uni-app团队耗费大量精力将绝大多数常用API封装为uni命名空间下统一的调用方式，使用这些API即可保证多端兼容，能实现大多数业务的跨端开发。但也有部分接口因平台能力和开放性等原因而有所不同，如果直接依据平台这一环境变量在代码中堆叠过多if-else，一方面会降低代码执行效率，增大代码体积，另一方面也不便于代码维护，给后续升级带来巨大的麻烦。

针对这类特殊情况，uni-app参考c语言的编译过程，给出了条件编译的解决方案，在代码编译期即实现了平台能力区分和个性化构建。例如，由于微信小程序和H5平台对设备WiFi控制能力的限制，绑定新设备的操作仅能在APP端实现，我可以借助#ifdef APP-PLUS #endif的代码块，在pages.json中注释掉设备绑定页面，在首页模板中注释掉前往设备绑定页面的按钮组件。

### 动态配网的客户端实现

在本章第二节中对比讨论了目前智能设备流行的动态配网方式，由于uni-app的APP端开发能力限制，我选择了Soft AP的配网方式，下面给出部分关键技术实现方式。

1. 扫描附近可绑定设备

在移动设备上扫描WiFi列表是需要定位权限的（因为WiFi列表可用于粗略定位），首先需要确认定位权限申请，关键代码如下：

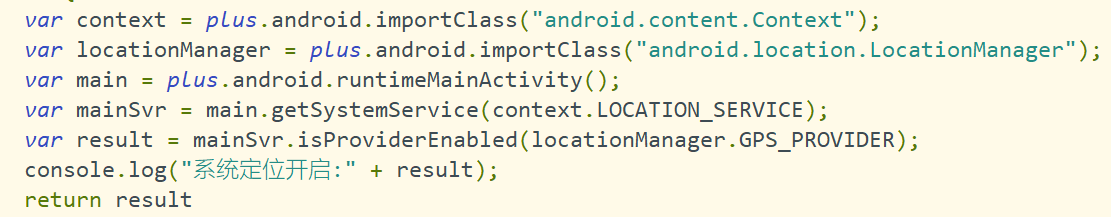


图 3‑9获取定位权限



图 3‑10初始化WiFi和扫描

从上图可以看到JavaScript代码中甚至像安卓原生开发一样引入了Java类，这其实是基于Native.js（简称NJS），将Java原生对象转义映射为JS对象，封装了一条通过JS语法直接调用Native Java接口的通道，开发者可以通过它调用任何一个系统API，给开发性能更好、功能更全的APP提供了无限可能。

1. 连接到指定WiFi

与扫描WiFi列表实现相差不多，不再赘述。

1. 向设备发送网络配置信息

设备在启动后启动了一个HttpServer，APP端在确认连接到设备后，使用uni.request这一API向设备上的http接口发送网络数据，由设备解析连接。

1. 从设备WiFi断开

设备建立的AP是没有正常网络的，因此发送完配置信息后要及时断开，避免影响用户的使用体验。

在整个配网过程中，有非常多可能的意外情况需要妥善处理，在一些仅支持同步的系统API调用过程中也要给用户尽量及时准确的交互反馈。

### 语音指令控制的实现

语音控制设备大致分为语音识别、语义解析、指令执行、结果返回为语音和语音播放等步骤，下面依次介绍技术实现。

1. 录音与语音识别

Uni-app封装了讯飞或百度的语音识别API调用方式，只需配置调用key等信息后调用API，即可轻松获得语音识别结果，将其发给后端服务处理。

1. 语义解析和执行

自然语言语义识别要求的技术较高，初期实现仅使用了分词和低技术含量的if-else AI。解析要获取信息或控制的设备，读取状态或执行语音包含的payload后给出返回结果。另外对于解析失败的语音指令，使用一些引导词兜底。

1. 执行结果转换为语音

语音合成直接使用了腾讯云提供的服务，音色非常自然，通过API调用即可返回base64编码的音频，将其返回给用户。

1. 语音播放

上一步返回base64编码的音频其实给用户端播放增加了开发难度，如果是直接url网络地址，可以直接调用uni-app的全局音视频管理来轻松播放。而对于base64编码后的音频，则需要先转为文件保存，再播放本地文件。部分关键实现如下，同样使用了Native.js开发：



图 3‑11转换base64编码后的音频为文件

在播放音频的实现中，遇到了提示为未知错误的问题，这根本没有任何有用的调试信息，给debug带来困难。尝试使用手机APP播放生成的音频后，发现有些APP给出了时长错误的报错信息，但尝试多种修补办法也均为奏效，只能将修复思路往前面的流程考虑。注意到腾讯云提供的语音合成API支持选择返回的音频格式，默认为wav，还可以选择mp3。通常来讲mp3是压缩过的，容错性相较wav也会更好[24]，在尝试选择mp3格式的返回后这个问题得到了解决，返回的数据包大小也降低了高达80%。

## 本章小结

这个章节详细介绍了各子系统的设计方案细节，对实现过程中的一些技术难点介绍了详细的实现方式。

# 系统成果展示与测试

## 硬件产品

本设计中，我基于前期完成的智能家居服务框架制作了温湿度传感器、继电器控制器这两个智能硬件，成品图如下：

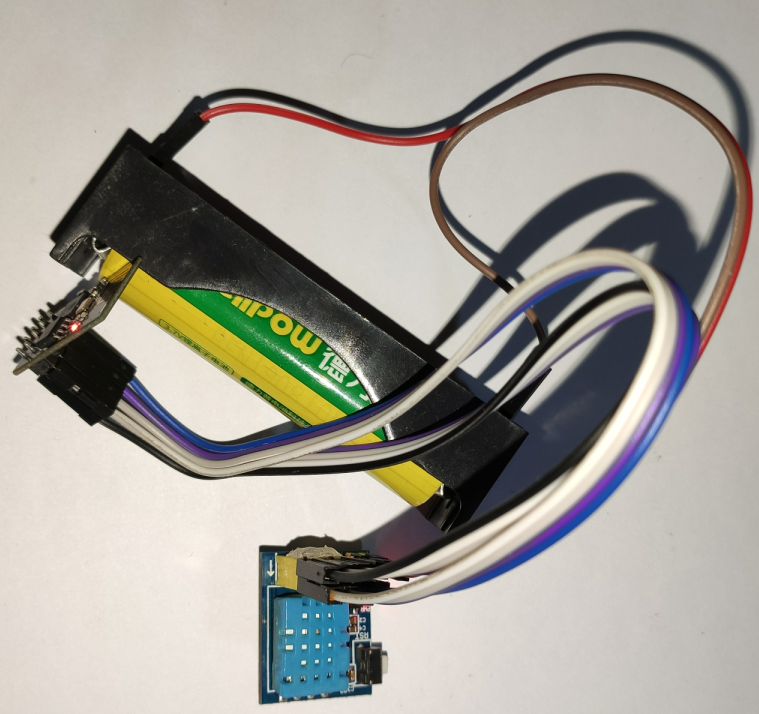


图 4‑1温湿度传感器设备图

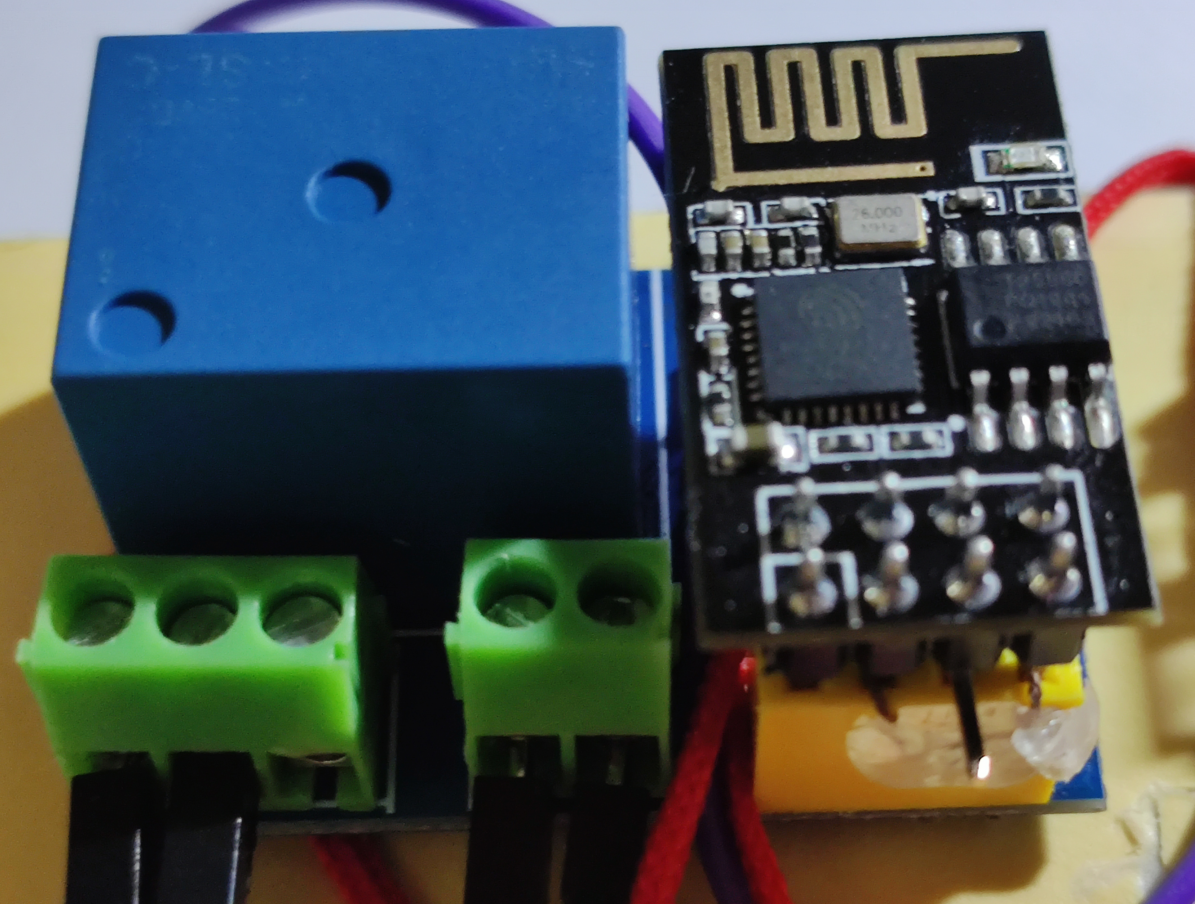


图 4‑2继电器控制器设备图

## 管理员后台

为了便于展示，在截图前强制放大了页面显示百分比，实际访问页面展示效果较为正常。

### 登录

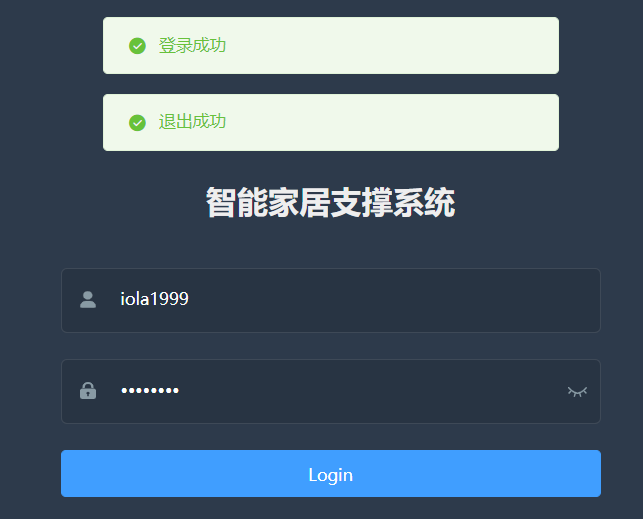


图 4‑3管理员后台登录

### 管理员后台首页

管理员登录后默认导航到后台首页看板，该页展示系统的部分运维数据。



图 4‑4管理员后台首页

### 人员管理

管理员可查看系统上的普通用户资料、绑定的设备，可以协助普通用户重置登录密码。

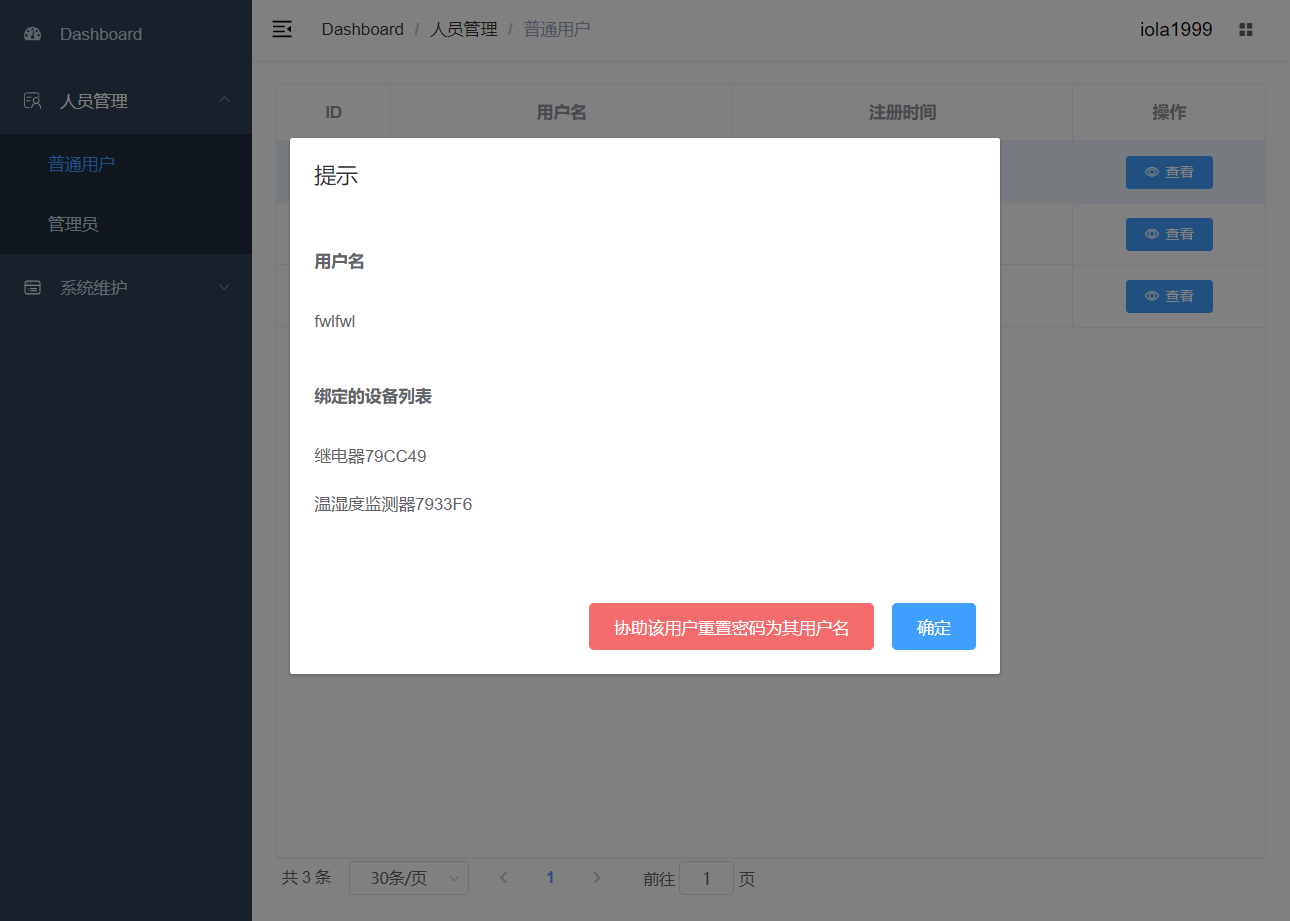


图 4‑5查看与编辑普通用户

超级管理员可查看和新增管理员。

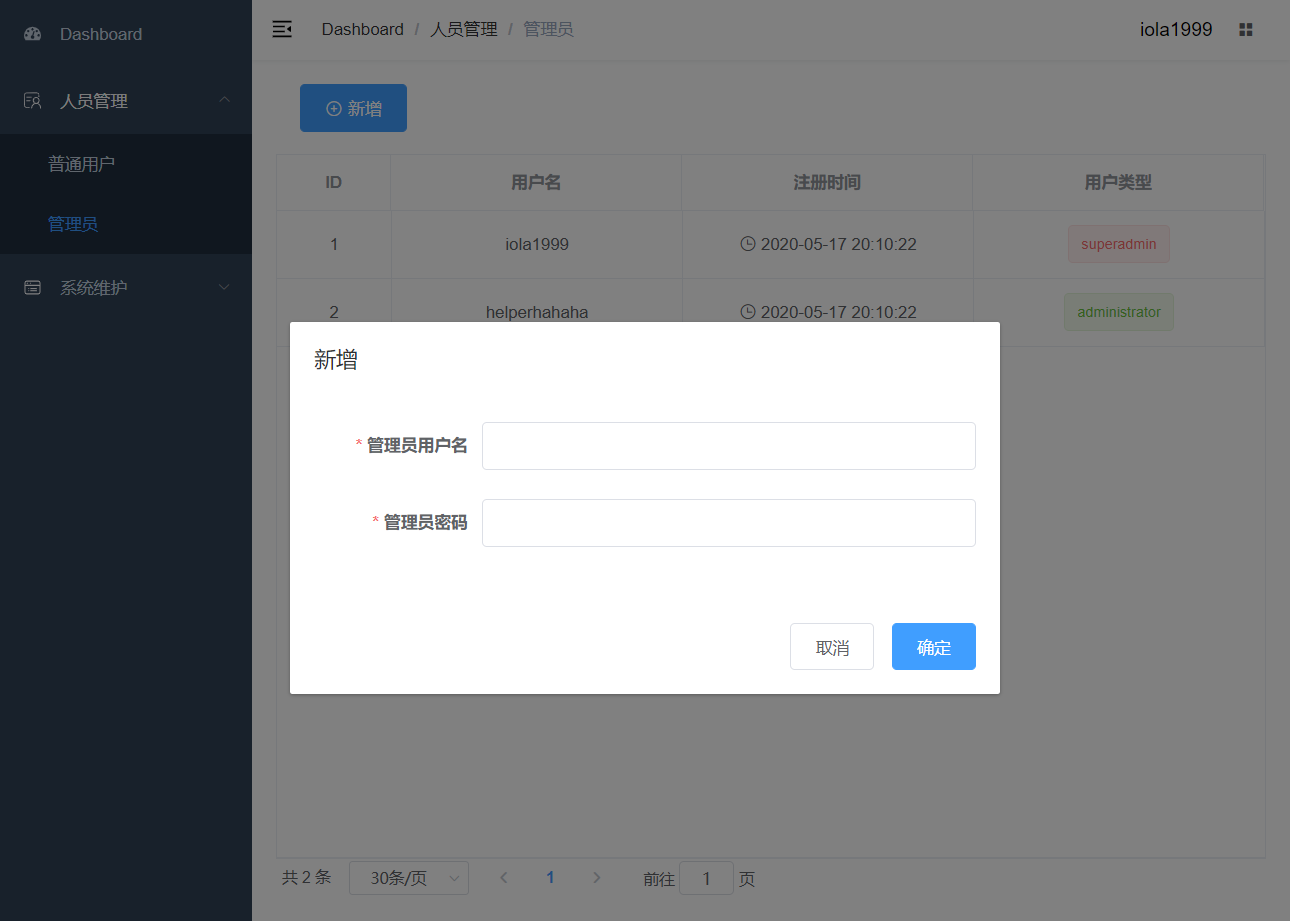


图 4‑6查看与新增管理员

### 系统维护

管理员可以在此查看当前系统上的智能产品（product），编辑产品的启用状态和其他资料。完成新产品研发后，管理员在此处新增该产品。

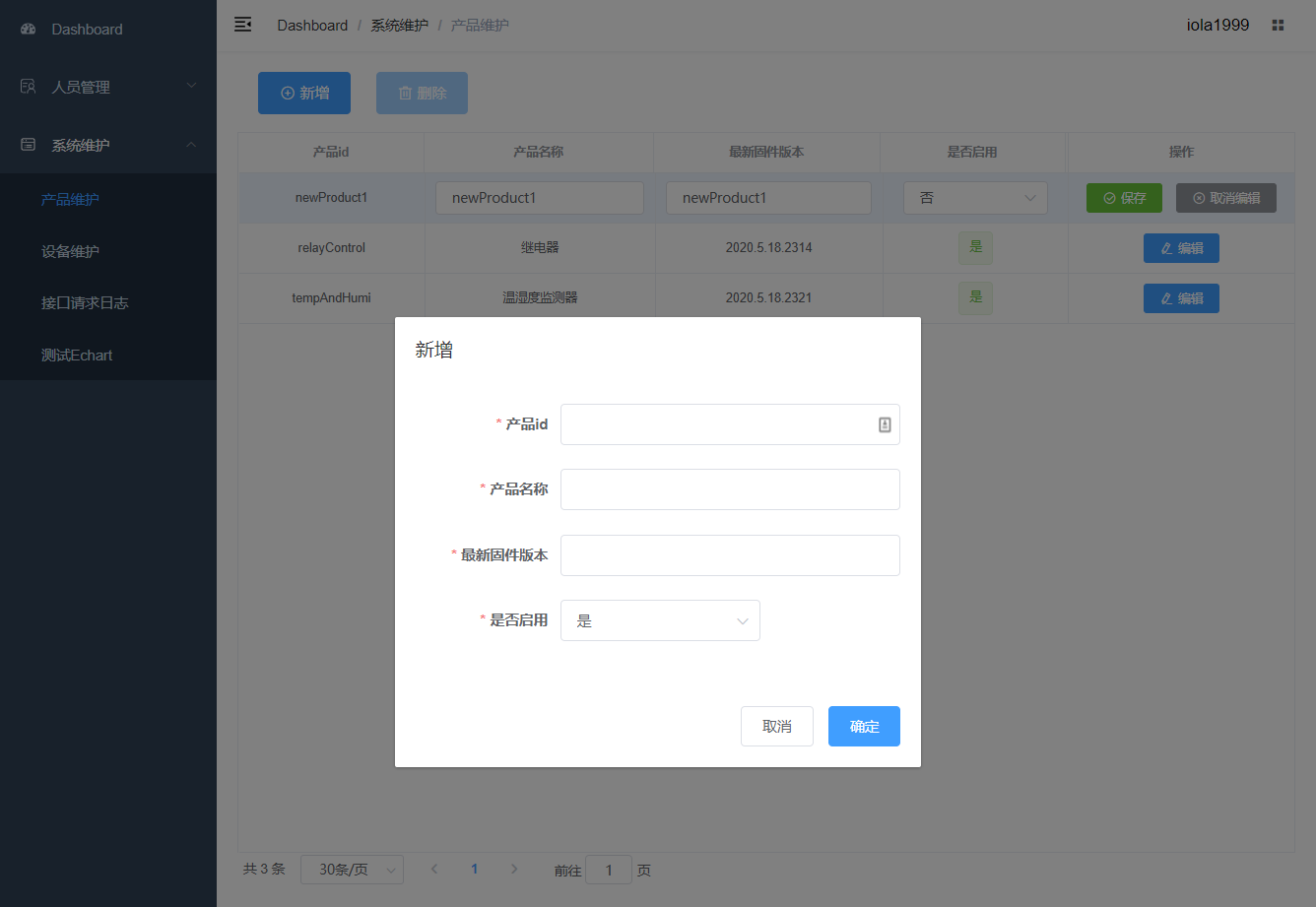


图 4‑7查看、新增与编辑产品

产品（product）被批量生产后得到设备（device），管理员在此将设备添加到系统中。还可在此查看各设备的绑定状态，必要时可管理绑定者。

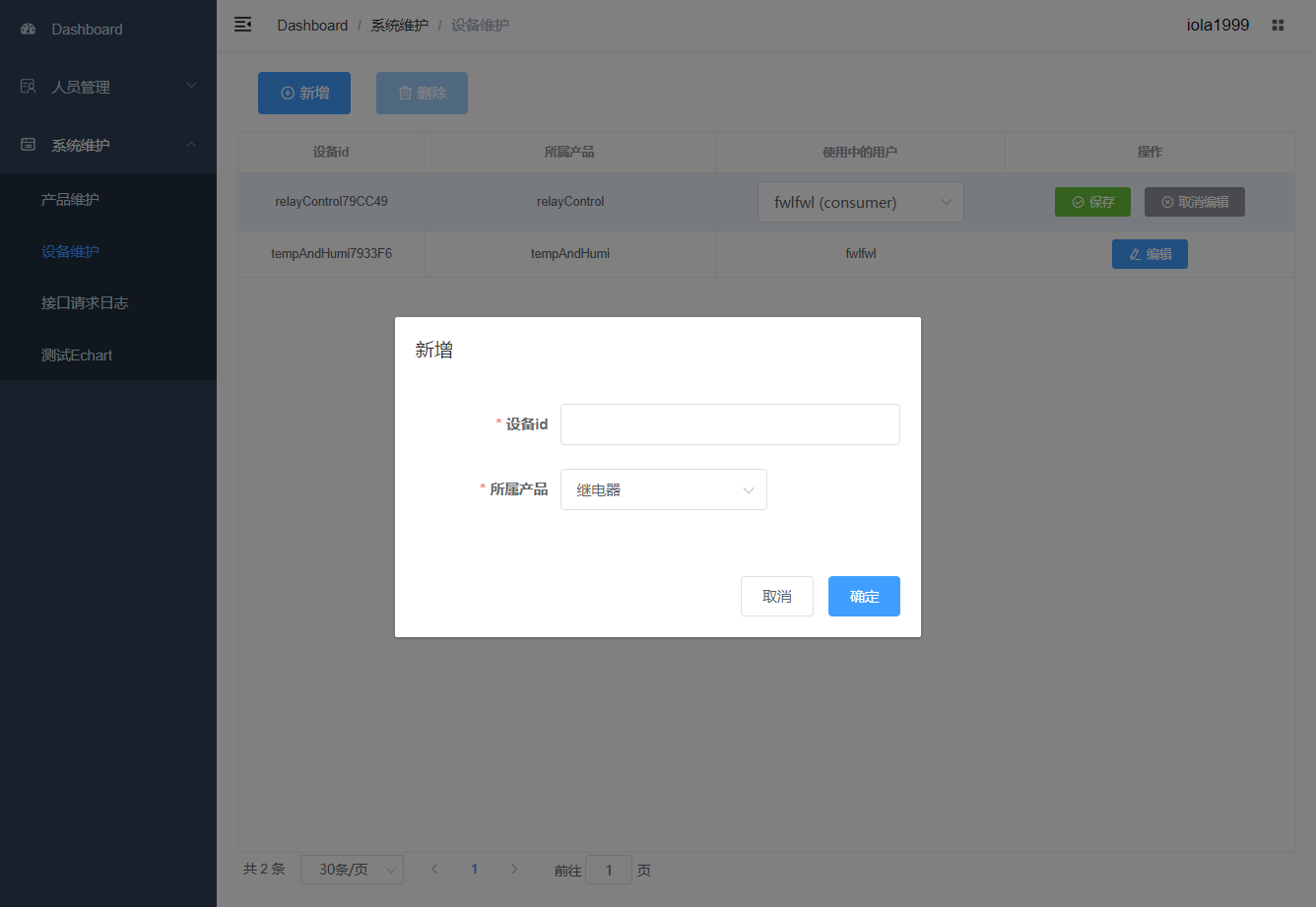


图 4‑8查看、新增与编辑设备

管理员可在接口请求日志页面查看系统后端API的请求记录，用于debug和审计、运维等。

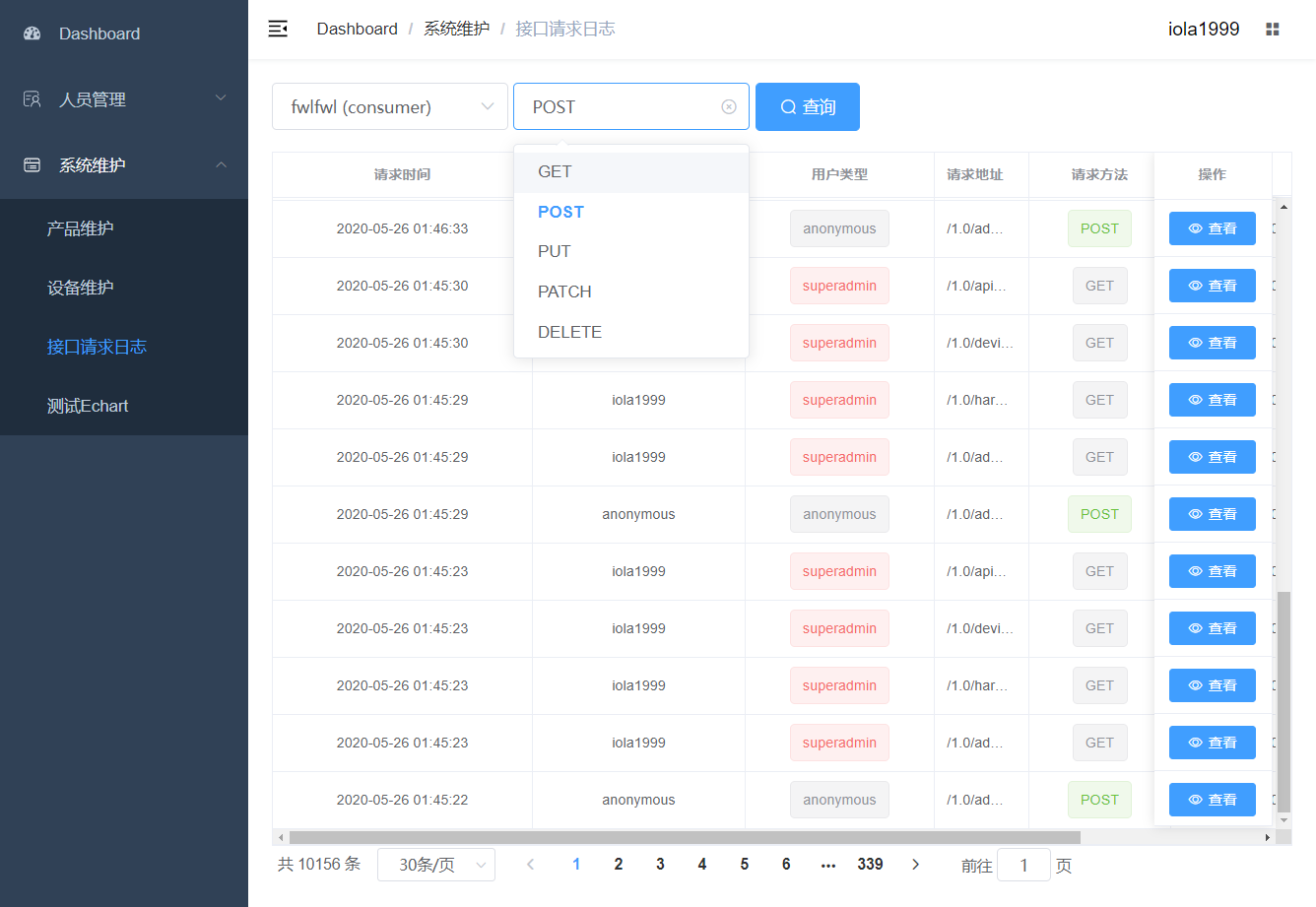


图 4‑9查看系统接口请求日志

## 用户客户端

由于各端样式基本一致，这里仅选用功能最全的APP端截图演示。

### 登录、注册与修改密码

APP启动后检查本地是否保存了用户的登陆状态，如未登录则跳转到登陆页面。登陆页面可切换为注册表单。用户登录后在个人中心可进入修改密码页面。

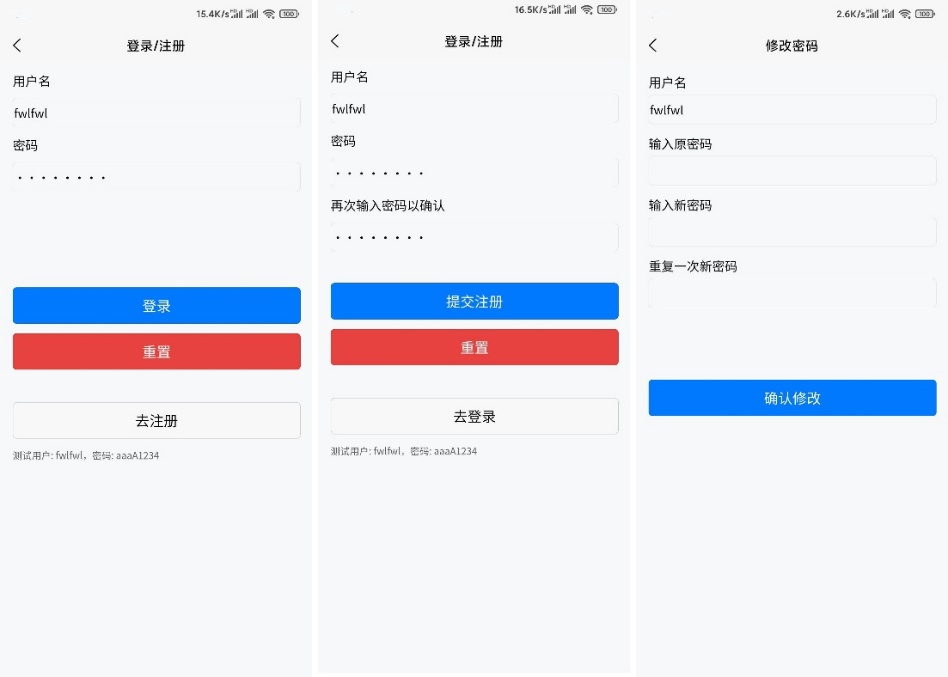


图 4‑10登录、注册与修改密码

### 登陆后Tab页面

用户登陆后底部有三个Tab，分别是首页、智能、个人中心。

首页展示用户绑定的设备列表，右上方是绑定设备按钮，右下方是语音控制按钮。个人中心上方展示用户名，底部是修改密码和退出登录按钮。

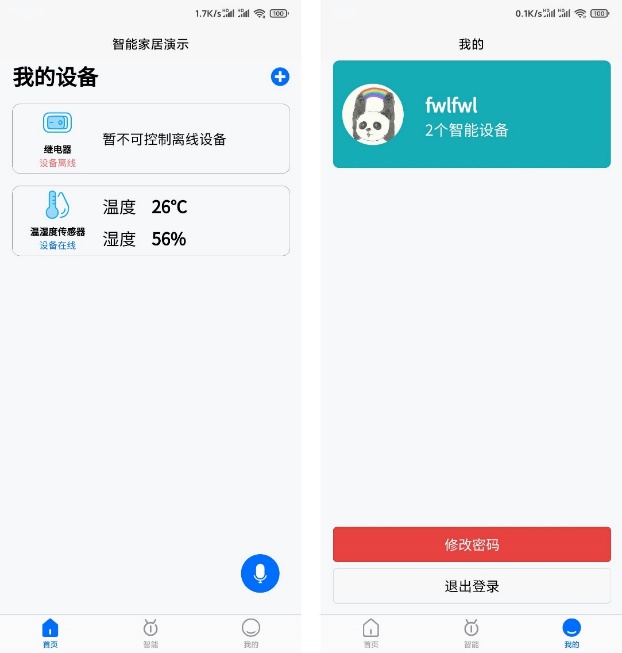


图 4‑11客户端首页和个人中心

智能Tab页面分左右两个标签页，左边是用户设置的智能化配置和启用情况，右边是智能化配置的执行记录。



图 4‑12智能化配置和执行记录

### 各业务操作页面

点击首页的设备按钮，可以进入设备详情页面，页面内容因产品种类而异。

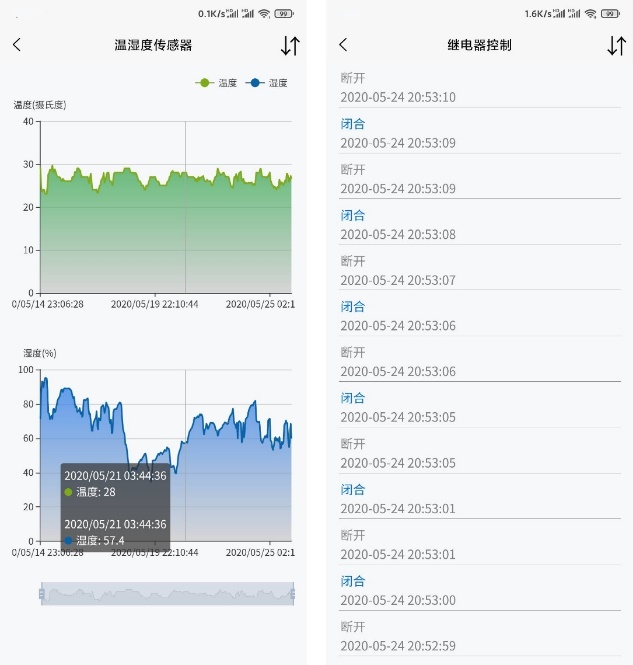


图 4‑13示例设备的设备详情页

点击首页右上角加号，可以绑定新设备（或修改已绑定设备的网络连接信息）。进入页面后检查权限，扫描WiFi后给出可绑定的设备列表。用户点击一条设备，弹出无线网络名称和密码输入框。用户点击确定后开始绑定流程。

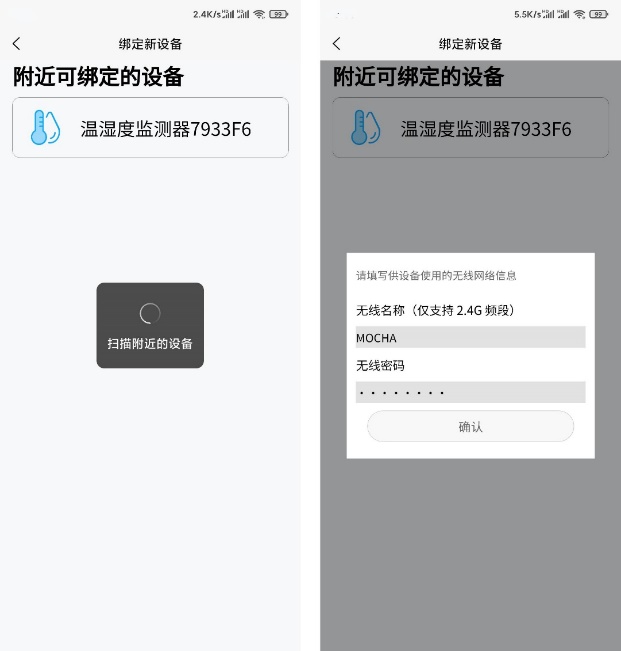


图 4‑14绑定设备过程

点击首页右下角的语音控制按钮，会开始语音识别，实时展示当前识别结果，短暂停顿后即认为结束语音指令，上传处理后返回响应结果，在屏幕顶部展示并语音播放。

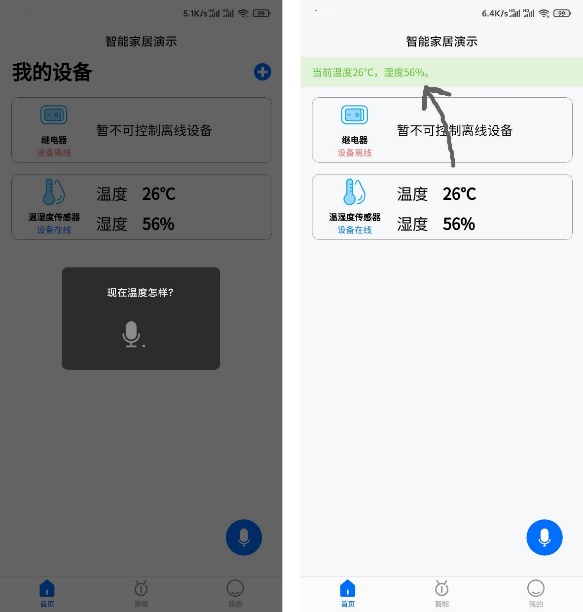


图 4‑15语音指令控制

## 系统测试

### 业务流程测试

为测试本系统的功能性和稳定性，我按照智能硬件产品从研发开始到用户日常使用的流程，对系统进行了一系列测试工作。

在服务端分别启动ActiveMQ和后端服务：

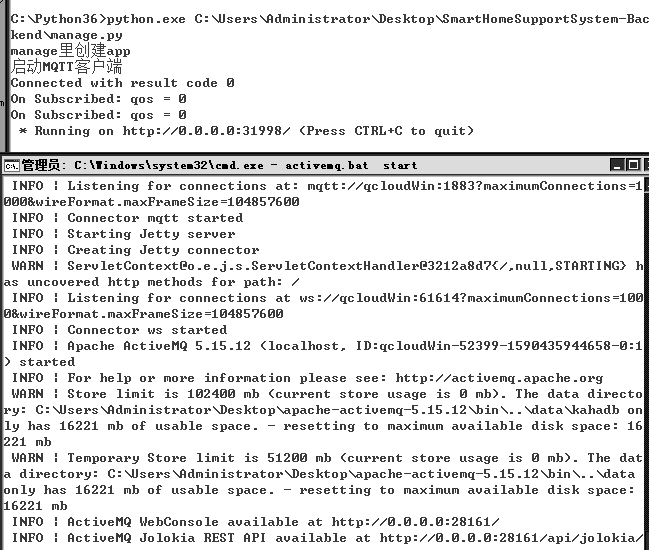


图 4‑16启动后端服务

在Arduino中编译烧写固件。

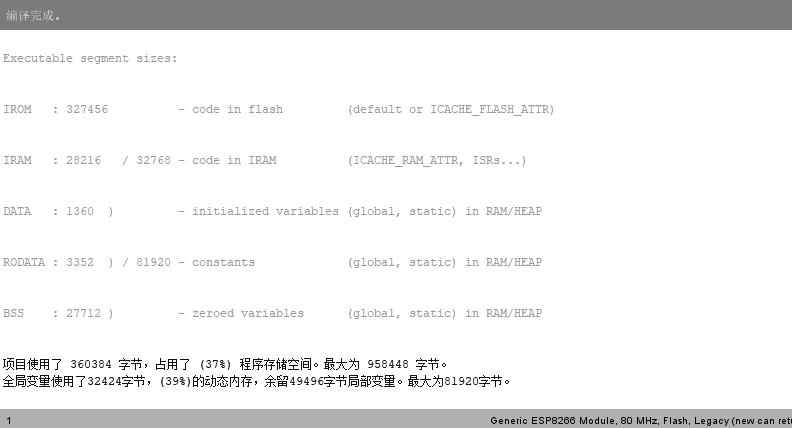


图 4‑17在Arduino中编译固件

以超级管理员身份前往管理员后台，新建一位普通管理员，以普通管理员身份登录后台，添加智能产品并设为启用状态，在添加类别为该产品的智能设备。

用户端安装APP，注册新用户并登录，在首页右上角点击绑定新设备，能够成功走完绑定流程，设备正常联网并上报信息。

在首页尝试设备快捷操作按钮，能够使设备正常开关。



图 4‑18通过首页快捷操作按钮控制设备

将设备供电直接断开，约十秒左右快捷操作按钮处显示设备已离线。

恢复供电，二十秒内设备重新上线，说明设备能够从意外事件中正常恢复且能够较稳定重连运行。

修改密码后，使用旧密码登录提示用户名或密码错误，换新密码后能够正常登录使用。

### 后端服务接口压测

后端运行环境为腾讯云1C1G1M Windows服务器上单线程Flask，数据库为阿里云1C2G5M服务器上的Mysql5.7，在个人本机Jmeter进行接口压测，压测线程数均为10线程。

表 4‑1后端服务接口压测QPS

|  |  |  |
| --- | --- | --- |
| 请求接口 | 开启请求日志记录时QPS | 关闭请求日志记录时QPS |
| 空接口 | 12.1 | 310 |
| 登录接口 | 8.6 | 13.4 |

从测试结果可知，后端接口瓶颈在于数据库连接性能上，在配置Flask-SQLAlchemy连接参数SQLALCHEMY\_POOL\_SIZE = 30（数据库连接池大小，默认5），以及Flask启动参数threaded = False 开启多线程后，其他条件不变时得到如下压测结果：

表 4‑2调整后端运行参数后的压测QPS

|  |  |  |
| --- | --- | --- |
| 请求接口 | 开启请求日志记录时QPS | 关闭请求日志记录时QPS |
| 空接口 | 59 | 476 |
| 登录接口 | 54 | 67 |

此时瓶颈是数据库传输性能和服务器出口带宽，若将后端服务与数据库放置于同一服务器上，QPS将会有更显著的提升[25]。

## 本章小结

本章展示了系统设计开发成果，记录了对系统进行的测试工作，表明该设计能够较为稳定地提供智能家居服务。

# **总结与展望**

## 总结

### 开发总结

在设计并实现本系统的过程中，我先对系统进行了拆分，分析了每一块的需求。经过初步调研后给出基础的可行性分析，深度研究进行了技术选型。在正式的系统开发中，我选择先建立数据库和管理员的管理后台，有利于顺利进行后续开发时的调试工作。完成后端基础的业务接口开发后，我开始着手智能硬件固件的开发工作。调通硬件与后端服务的数据链路后，我开始一边进行客户端APP的开发，一边优化硬件固件。完成整个系统开发后我对整个流程进行了多次测试。

在试用的一段时间内，该系统能够稳定运行，且拥有较高的可定制性，适合研究与进一步开发。

### 存在的问题

本系统的设计与实现过程中有以下几处比较遗憾的地方：

1. 智能化任务配置界面的实现过于复杂，各种输入条件能产生多种组合，导致难以生成页面表单，最终只好暂时移除智能化配置UI；
2. 未能优化好硬件功耗控制，使用2500mAh的18650锂电池供电仅能连续工作不到三天；
3. 后端服务的数据库连接性能较差，QPS过低，由于对后端开发经验不足，短时间内不便切换开发语言来优化性能；
4. 受限于APP开发方式的限制，难以接入乐鑫提供的ESP8266 Smart Config SDK，只能选择Soft AP这种稍复杂的配网方式；
5. 后端开发过程中未添加单元测试，全程靠人工Postman测试接口可用性；
6. 界面UI设计比较简陋，部分场景缺乏交互动画。

### 解决方案

针对以上问题，在设计中我考虑过如下对应的解决/优化方案：

1. 重新梳理智能化配置界面操作逻辑，尝试优化该页面；可以根据用户拥有的产品种类，自动提示一些可能的硬件联动组合；
2. 后续智能硬件可以考虑低功耗蓝牙+网关的组合，还可以优化设备数据上报频率的算法，以降低平均功耗；
3. 考虑更换其他性能较好的后端开发语言，或尝试优化flask\_sqlalchemy的连接池等设定，或将后端服务部署在Linux系统上即可利用threaded多线程或Gunicorn协程来大幅优化性能[26]；
4. 优化Soft AP配网方式中给用户的提示；改用APP原生开发；
5. 给后端开发接入pytest等测试框架，以测试驱动开发，降低bug率；
6. 学习UI设计的知识，优化界面。

## 展望

随着物联网技术和人工智能的不断发展，我相信未来智能家居产品智能化的趋势会不断增强，越来越多的家庭电器会发生智能化改造，能够给人们带来更加便捷高效的智能家居体验。

**参考文献**

都是乱引用的，没有意义，被删除。

**致谢**

被删除