# 基于树莓派与微信的智能家居感知系统

## 摘要

智能家居感知系统是一个基于智能家居设备对室内进行实时监控及管理的平台，目的在于能够智能控制和管理家电设备，并且实时地对家庭环境的安全进行监控，进而为人们提供一种更加舒适安全、智能高效的生活工作环境，提供人们的生活质量。

开发语言采用现今流行的Python，框架采用异步非阻塞的Tornado实现，数据库采用非关系型数据库Redis，用户交互采用当下人们普遍使用的微信公众号，控制中心采用Raspberry Pi，Sensor控制器采用便捷的单片机Arduino。该系统耦合性低，高并发，学习成本低，上手迅速，有很好的用户体验。

智能家居感知系统主要包括四大部分：后台业务系统、SOCKET系统、家居控制中心系统和SENSOR控制子系统。这是一个集用户管理、设备管理、传感器管理于一体的智能家居系统，能够协助用户方便的进行室内管理与监控。

关键词： PYTHON，TORNADO，Raspberry Pi，Arduino，智能家居

## ABSTRACT

Cognitive system of intelligent home is a smart home devices on the interior of real-time monitoring and management platform based on. The purpose is to to intelligent control and management of appliances, and in real time on the family environment and safety monitoring, and for people to provide a more safe and comfortable, intelligent, living and working environment, the quality of people's lives.

With today's popular Python language development and framework of the use of asynchronous non blocking tornado, the database uses non relational database redis, user interaction with the current widespread use of the micro channel public number, the control center the raspberry PI, the sensor controller using convenient microcontroller Arduino. The system with low coupling, high concurrency, low cost, fast learning, good user experience.

Home intelligent perception system mainly includes four parts: background business system, socket system, home control center system and sensor control subsystem. This is a set of user management, equipment management, sensor management in one of the intelligent home system, can help users of laboratory management and monitoring.

Keywords: PYTHON, TORNADO, Raspberry Pi, Arduino, Intelligent Home Furnishing

## 绪论

### 一、课题来源及意义

随着互联网电子信息技术的快速发展，生活的信息化智能化已经成为发展的必然趋势。仔细观察身边的事物，很容易发现：信息化已经在深深了改变人们的生活习惯和工作方式，同时也对人们的传统住宅提出了挑战，可以看到越来越多的人更希望家居能够智能化，希望它能够自动控制管理家居设备并能够对室内的环境做出实时地处理，为人们提供一种安全舒适与便捷高效的工作生活环境。

系统的前端入口是一个极为方便便捷的微信公众号，用户只需要简单的关注一下公众号并绑定家居控制中心就能够进行家居控制与监控，同时用户也可以通过统一的web界面实时预览控制室内所有的设备；此外，用户还可以通过语音与公众号进行交流，获取自己需要的信息或者进行相应的设备管理。

系统的后端包括两大部分：基本业务与SOCKET连接。基本业务主要进行一系列的操作与管理：设备管理、用户管理、用户与设备绑定、传感器数据管理、智能交流等；SOCKET连接主要用来维护家居控制中心与服务器的长连接管理，并提供RPC服务供基本业务调用来实现实时控制的功能。

### 二、发展状况

2014年，随着Google收购Nest Labs，智能家居的理念再现生机重新变得火热起来，国内很多互联网公司都积极地参与其中并筹划自己的项目，希望能够抓住新一轮的浪潮。极路由、小米路由、360路由、智能电视、智能音响、智能摄像头、智能插座等智能家居设备都以一个单一的身份出现在市场之上，但还没有一个完善的统一的低成本的智能家居解决方案。

该课题旨在让智能家居的概念更进一层，不只停留在单一的硬件设备上，而是提供一套完善的智能家居解决方案，化繁为简，让用户不再安装一大堆智能设备控制软件，不再需要繁杂的步骤去部署安装智能设备；该解决方案能够实现设备自动部署，而且为用户提供一个统一的入口去管理与控制所有的家居设备，从而真正地让用户感觉到智能家居带来的舒适便捷与安全贴心。

## 第一章 设计思想

### 第一节 微信公众号

前端入口为微信公众号，用户通过公众号内的二维码扫描可以方便的绑定客户端，底部菜单可以方便的查看家里的温度湿度以及控制电灯的开关，另外提供一个统一的web入口，用户可以方便的查看所有的家居设备并进行相应的管理

### 第二节 后台业务系统

后台业务系统用来对各个基本业务进行管理控制，可以分成以下模块：

1. 用户管理
   1. 用户注册、销毁等基本帐户功能
   2. 客户端绑定
2. 设备管理
   1. 新增设备并生成相应二维码
   2. 传感器管理：增加传感器、更新传感器信息、存储传感器数据、获取传感器数据
3. 聊天Robot
   1. 处理用户发送的消息
   2. 调用图灵机器人api，进行基础聊天
   3. 根据用户发送的指令，管理家居设备

### 第三节 SOCKET长连接

通过TCP的keep-alive特性，建立客户端与服务器端的长连接，以保证客户端不会被路由器踢掉，从而实现用户实时控制设备的功能，包括以下两个模块：

1. SOCKET建立、保持与剔除

* 将建立起来的SOCKET与设备唯一标示，以字典的方式放入内存中，如果SOCKET断开或者通信出现异常则剔除该连接，并且提供一定的超时检查机制以避免服务器端阻塞无响应

1. RPC服务，供后台业务系统调用

* 通过RPC暴露响应的信息发送接口，让其他系统可以方便的借助socket发送消息到客户端

### 第四节 家居控制中心

家居控制中心通过ZIGBEE协议来管理各个传感器，是一个服务器后端与传感器控制端通信的通道。

1. SOCKET

* 借助于TCP的keep-alive特性，建立与服务器端的长连接，使其能够进行实时控制与响应。

1. 串口ZIGBEE

* 控制中心借助ZIGBEE模块建立的个域网，管理与控制各个传感器，并将sensor数据上传到服务器。

### 第五节 SENSOR控制子系统

该系统主要用于和控制中心的交互，比如：数据上传、控制设备的开关等

## 第二章 相关技术及开发工具

### 第一节 相关技术

#### 一、PYTHON

1. Python简介

* Python是一种面向对象、解释型的计算机程序设计语言，语法简洁而清晰，具有丰富和强大的类库，简单易学，跨平台，可以快速生成程序的原型。Python可以处理好多好多事情，小到日常任务，大到web网站甚至网络游戏的后台，很受广大用户的欢迎。

1. Python优势

* 简单易学，容易上手，拥有丰富强大的类库，能够快速生成程序的原型。

#### 二、TORNADO

1. Tornado简介

* Tornado是FriendFeed使用的可扩展的非阻塞web服务器及相关工具的开源版本，该框架不同与现在主流的Web服务器框架，它是非阻塞式服务器，速度相当的快。

1. Tornado优势 异步非阻塞、每秒钟能够数以千计的连接，对于实时的Web应用非常有用。

#### 三、SOCKET

1. Socket简介

* Socket是应用层与TCP/IP协议族通信的中间软件抽象层，准确来说是一组接口，俗称套接字，是不同网络应用程序之间进行通信的通道。

1. 长短Socket

* 程序中建立的Socket连接，如果没有特殊说明的话主要指的是短连接，该连接会在一段时间之后断开活着被相关的路由器踢掉。为了实现基于Socket的长连接，则需要客户端每隔一小段时间就向服务器端发送一些信息（该信息可以是无用的，也可以是专门设计过的），实现方案主要分为以下两种：
  1. 利用TCP/IP的keep-alive特性，定时向服务器发送冗余信息
  2. 根据业务需求定义相应的数据包，并定时向服务器发送自定义的数据包

#### 四、HTML5

Html5将会取代1999年制定的HTML4.01、XHTML1.0标准，让互联网应用在迅速发展的时候，可以为桌面和移动平台带来无缝衔接的丰富内容，将会推动Web进入新的时代。当今支持Html5的浏览器越来越多：Chrome35+、Safari、Firefox、Opera、IE9+，很多手机端的应用也开始使用h5来开发，可以看到HTML5的春天来了，一定会迎来一番热潮。

#### 五、RESTFUL

REST指的是一组架构约束条件和原则，并没有明确的定义，只要满足这些约束条件和原则的应用程序或者设计RESTFUL。其原则为，客户端与服务器之间交互在请求之间是无状态的，但客户端与服务器的每个请求都必须包含理解请求所必需的信息，一般为标准的HTTP方法，比如GET、PUT、DELET、POST等。

#### 六、XML

xml为可扩展标记语言，是标准通用标记语言的子集，是一种用于标记电子文件使其具有结构性的标记语言。通过此标记，计算机之间可以处理包含各种信息的东西，可以用来标记数据、定义数据类型；此外，它非常适合互联网传输，并提供统一的方法来描述和交换独立于应用程序或供应商的结构化数据。

#### 七、ZIGBEE

1. Zigbee简介

* ZigBee是基于IEEE802.15.4标准的低功耗局域网协议，用来提供一种短距离、低功耗的个域网。由于其近距离、低复杂度、自组织、低功耗、低数据率的特点，其主要适用低速短距离传输。

1. Zigbee优势

* 低复杂度、自组织、低功耗，可以方便的组建个域网，并进行基本的管理工作。

#### 八、Serial Port

串口即串行接口，是指数据一位一位地顺序传送，通信线路简单，只需要一对传输线就可以进行双向通信，成本低廉特别适合远距离通信。

#### 九、Bootstrap

Bootstrap是一个基于HTML、CSS和JS的框架，旨在用于开发响应式布局、移动设备优先的WEB项目，它是为所有开发开发者、所有应用场景而设计的，可以让前端开发变得更快速、更简单。

### 第二节 开发工具

#### 一、Pycharm

PyCharm是一种Python IDE，带有一整套可以帮助用户在使用Python语言开发时提高其效率的工具，比如调试、语法高亮、Project管理、代码跳转、智能提示、自动完成、单元测试、版本控制。此外，该IDE提供了一些高级功能，以用于支持Django框架下的专业Web开发等。

#### 二、REDIS

1. Redis简介

* Redis是一个开源的支持网络、可基于内存亦可持久化的日志型、Key-Value数据库，内置string、list、set、zset、hansh等多种存储类型，简单易用，快速高效。

1. Redis优势

* 由于Key-Value的特性，不再需要建立初始的表结构，可以随存随用，支持过期时间，可以使开发达到事半功倍的效果。

#### 三、Git

1. Git简介

* Git是一个开源的分布式版本控制系统，用以有效、高速的处理从很小到非常大的项目版本管理。相对于传统的集中式仓库最大的区别为，每一个开发者在工作过程中不用联网可以是离线的，可以将程序提交到本地仓库，联网的时候再把本地更新推送到远程服务器。

1. Git优势
   1. 适合分布式开发，便于团队开发
   2. 速度快、灵活
   3. 版本库本地化，支持离线提交，相对独立不影响协同开发
   4. 支持快速切换分支方便合并，比较合并性能好
   5. 分布式版本库，无单点故障，内容完整性较好

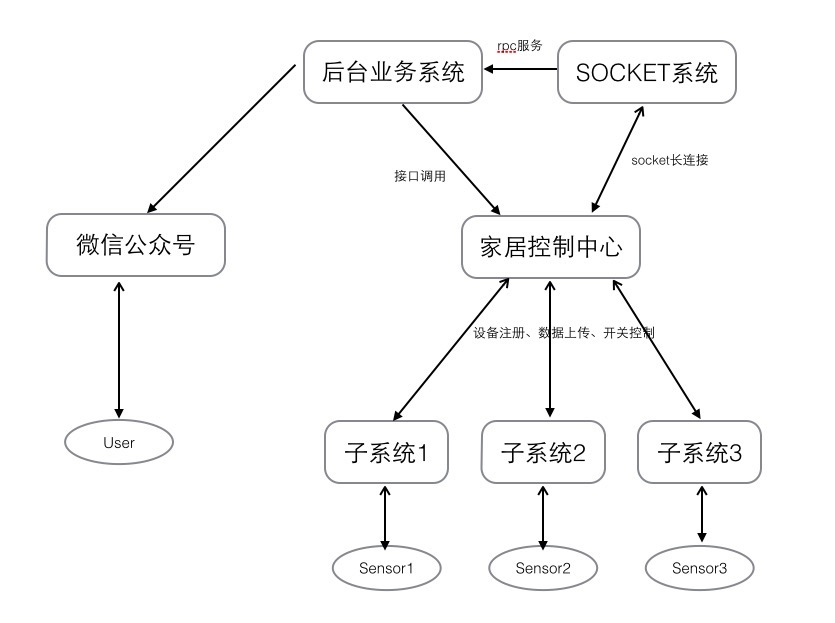
## 第三章 概要设计

### 第一节 框架设计

#### 一、整体框架

智能家居感知系统主要包括五大功能模块：微信公众号、后台业务系统、SOCKET系统、家居控制中心系统和设备控制子系统。

整体框架图如下所示：



后台业务系统：负责基本的用户管理、设备管理、数据存取、设备绑定等业务。

SOCKET系统：负责socket长连接管理，并提供rpc服务供其他业务方调用

微信公众号：负责与微信公众平台的对接

家居控制中心：负责室内各个设备的管理与控制，并支持设备的动态添加与删除

子系统：负责具体的某一个设备的管理

#### 二、后台业务系统

业务系统主要用来处理系统的具体业务，主要包括以下模块：

1. 帐户管理
   1. 用户关注微信公众号时，系统会自动生成新用户，并把用户的基本资料写入
   2. 用户取消关注微信公众号时，系统会将用户状态设置为已注销状态，不清楚用户信息，以防用户再次关注
   3. 用户扫描家居控制中心的二维码，可以绑定相应的家居客户端，当前暂只能绑定一个家居客户端
2. 设备管理
   1. 管理员可以增加新设备，并生成相应的二维码
   2. 查询指定设备的二维码
   3. 为指定的设备添加传感器
   4. 传感器数据的存储、更新
   5. 提供统一的web入口，供用户查看并管理所有设备, 界面如下所示:

* 

1. 聊天机器人
   1. 解析用户发送的语音消息（微信可以帮我们自动处理）
   2. 根据不同的指令做出不同的响应，并返回给微信公众号

#### 三、SOCKET系统

SOCKET系统主要用于保持客户端的实时在线，并实时处理信息，主要包括以下两个模块:

1. Socket模块
   1. 建立与客户端的socket长连接，保持客户端一直在线
   2. 处理客户端与服务器端的实时信息
2. Rpc服务模块

* 暴露所需的rpc接口，使后台业务系统可以通过相应的rpc接口，实时地控制客户端

#### 四、家居控制中心

家居控制中心主要用于协调各个室内设备与服务器端的关系，并对室内设备进行基本的管理控制，主要包括以下模块:

1. Socket模块

* 与服务器端建立一条socket长连接，实时接收服务器端的指令

1. ZigBee模块

* 借助于ZigBee模块，控制中心以无线局域网的方式向受控的设备发送指令信号或者接受受控设备的数据

1. Deamon模块

* 定时检查网络和程序的有效性，如果客户端断开或丧失了与服务器建立的Socket连接，则会重新启动客户端程序使其重新建立Socket连接

#### 五、微信公众号系统

该系统主要用于对接微信公众号后台，实时响应用户在公众号的操作。按照微信消息的类别，系统大概可以分为以下模块：

1. 文本模块

* 处理用户的Text消息

1. Event模块

* 处理不同类别的Event事件消息，比如：菜单Click与View、扫描Scan等

1. Voice模块

* 处理用户发送的语音消息

#### 六、Sensor控制子系统

每个Sensor都拥有一个控制系统，可以分为以下两个模块：

1. 管理模块

* 用于对传感器的注册、控制、上传温度等

1. 通信模块

* 借助于ZigBee模块，建立与控制中心的通信，接收控制中心的信号指令或者向控制中心上传数据

### 第二节 数据库设计

#### 一、数据表设计

数据存储采用了Nosql数据库Redis，则不需要提前建立所需的数据表，只需要在使用的时候建立即可；相对于传统的数据库而言，redis是一个基于key-value的数据库，使用起来方便便捷，开发效率很高。

鉴于Redis内置了很多存储数据结构，为了更高效的进行开发以及使用，需要提前确定下来表的数量、名称以及将要使用的存储结构，具体详情参考下面：

1. device:list

* 采用内置的set集合存储，存储的为各个设备的唯一编号

1. device:id

* 采用内置的incr存储，自增长，存储的为最大的device序列号

1. device:{{id}}

* 采用内置的hash存储，存储的为设备的详细信息，比如二维码地址、二维码ticket、tag标签等
* 注：id是变化的，为设备的id

1. {{id}}:sensors

* 采用内置的string存储，存储的为序列化之后的sensors的dict结构

1. sensor:data:{{device\_id}}:{{sensor\_id}} 不同类型的sensor采用不同的数据结构存储，控制型sensor采用string存储，value型sensor采用有序集合存储，存储的为sensor的value
2. user:list

* 采用内置的set集合存储，存储的为各个用户的唯一编号

1. user:id

* 采用内置的incr存储，存储的为最大的用户id

1. user:{{id}}

* 采用内置的hash存储，存储的为用户的详细信息，比如用户的微信openid、绑定的设备id

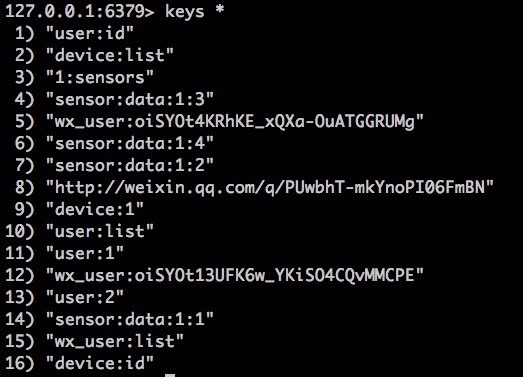
1. wx\_user:list

* 采用内置的无序集合存储，存储的为各个用户的微信openid

1. wx\_user:{{openid}}

* 采用内置的hash存储，存储的为微信用户的详细信息，比如用户的系统id、用户的微信基本资料等

数据库中存在的所有表，如下所示：



## 第四章 详细设计

### 第一节 微信公众号

用户在微信公众号内的所有操作，微信开发者平台都会以xml格式向开发者注册的url发送消息，基本的消息结构如下

<xml>  
 <ToUserName><![CDATA[toUser]]></ToUserName>  
 <FromUserName><![CDATA[fromUser]]></FromUserName>   
 <CreateTime>1348831860</CreateTime>  
 <MsgType><![CDATA[msg\_type]]></MsgType>  
 <Content><![CDATA[content]]></Content>  
 <MsgId>1234567890123456</MsgId>  
 </xml>

ToUserName和FromUserName分别为消息的发送者和接受者，都为其对应的微信Openid，开发时需要着重关注MsgType、Event、Content或EventKey参数，通过这些参数可以知晓用户的操作，并作出相应地回应。

#### 1.1 底部菜单

微信公众号支持底部菜单，可以包含三个一级菜单，每个一级菜单下面最多可以包含五个二级菜单。通过底部菜单，用户可以很方便地进行相关设备的控制与管理，底部菜单UI如下所示：

一级菜单:



二级菜单：





微信底部菜单采用json创建，json文件如下所示:

{  
 "button": [  
 {  
 "type": "click",  
 "name": "温度湿度",  
 "key": "REAL\_TEMPERATURE"  
 },  
 {  
 "name": "电灯",  
 "sub\_button": [  
 {  
 "type": "click",  
 "name": "打开",  
 "key": "LIGHT\_ON"  
 },  
 {  
 "type": "click",  
 "name": "关闭",  
 "key": "LIGHT\_OFF"  
 }  
 ]  
 },  
 {  
 "name": "个人中心",  
 "sub\_button": [  
 {  
 "type": "scancode\_waitmsg",  
 "name": "客户端绑定",  
 "key": "BIND"  
 },  
 {  
 "type": "view",  
 "name": "我的设备",  
 "url": "http://sun.codemagic.tk/weixin/oauth"  
 }  
 ]  
 }  
 ]  
}

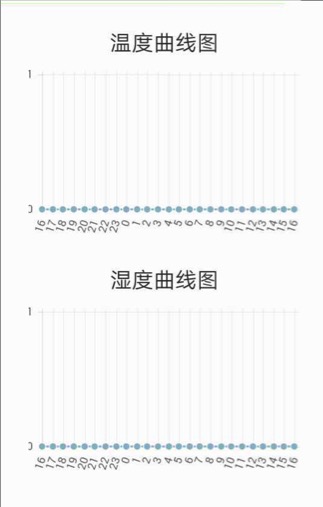
#### 1.2 温度湿度

当用户获取温度湿度时，返回给用户的为单条图文消息（包含过去五个小时的温度湿度和查看所有的链接），界面如下图所示：



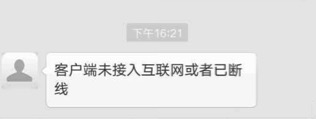
注：如果某个时间点没有温度湿度，则会显示暂无温度湿度

如果用户点击上面的查看原文链接后，将跳转到过去24小时温度湿度的展示界面，如下所示:



#### 1.3 电灯开关

当用户点击电灯开关的时候，如果家居客户端丧失与服务器端的连接，则会给出友好的提示，如下所示：



如果成功打开电灯或者关闭电灯，则会给出如下提示：



#### 1.4 绑定设备

用户点击底部二级菜单绑定设备时，会自动调用微信的二维码扫描组件，扫描对应设备的二维码既可以绑定新的设备。

注：当前版本，一个用户同时只能绑定一个设备，一个设备可以支持多个用户绑定。

#### 1.5 我的设备

当用户点击我的设备的时候，会跳转到统一的web管理入口，如下所示：



其中开关按钮使用Bootstrap Switch绘制而成，js代码如下

{% for sensor in led\_sensors %}  
 {% set id = sensor['id'] %}  
 $('#{{ id }}').bootstrapSwitch();  
 $('#{{ id }}').on('switchChange.bootstrapSwitch', switch\_light);  
 {% end %}  
$("[name='my-checkbox']").bootstrapSwitch();

点击开关按钮时，使用Ajax向后台发送请求，请求失败则重新加载界面，相关js代码如下

function switch\_light(event, state){  
 sensor\_id = this.id;  
 if(state){  
 value = 1;  
 }else{  
 value = 0;  
 }  
 url = "/device/" + device\_id + "/sensor/" + sensor\_id;  
 var data = {  
 value: value  
 }  
 data = JSON.stringify(data);  
 $.ajax({  
 type: "PUT",  
 url: url,  
 contentType: "application/json; charset=utf-8",  
 data: data,  
 dataType: "json",  
 success: function (msg, status, xhr) {  
 errcode = msg.errcode  
 if (errcode==0){  
 console.log('success')  
 }else if (errcode == -1){  
 console.log(msg.errmsg)  
 window.location.reload()  
 }  
 },  
 error: function (msg) {  
 console.log(msg);  
 }  
 });  
 }

温度湿度图表采用精巧的Chart.js绘制，js代码如下

var hum\_ctx = $("#hum").get(0).getContext("2d");  
var tem\_ctx = $("#tem").get(0).getContext("2d");  
  
//This will get the first returned node in the jQuery collection.  
var hum\_chart = new Chart(hum\_ctx);  
var tem\_chart = new Chart(tem\_ctx);  
var hum\_data = {  
 labels: {{ labels }},  
 datasets: [  
 {  
 strokeColor: "rgba(151,187,205,1)",  
 pointColor: "rgba(151,187,205,1)",  
 pointStrokeColor: "white",  
 data: {{ hum\_data }}  
 }  
 ]  
}  
var tem\_data = {  
 labels: {{ labels}},  
 datasets: [  
 {  
  
 strokeColor: "rgba(151,187,205,1)",  
 pointColor: "rgba(151,187,205,1)",  
 pointStrokeColor: "white",  
 data: {{ tem\_data }}  
 }  
 ]  
}  
var options = {  
 datasetFill: false,  
 tooltipEvents: ["mousemove"],  
 tooltipTemplate: "<%= value %>"  
  
}  
hum\_chart.Line(hum\_data, options);  
tem\_chart.Line(tem\_data, options);

注：上面js代码中用双大括号括起来的变量，需要后台相关数据的渲染，使用的是Jinjia2模板的语法

### 第二节 后台业务系统

#### 2.1 帐户管理

##### 2.1.1 用户注册

当用户关注微信公众号时，系统会自动帮用户注册，注册操作对于用户来说是透明的。公众号二维码分两种：1. 普通的公众号二维码 2. 包含设备信息的二维码，故逻辑需要根据不同类别的二维码进行相应地处理。

def \_subscribe(self):  
 if self.redis.sismember("wx\_user:list", self.from\_user):  
 user\_id = int(  
 self.redis.hget("wx\_user:%s" % self.from\_user, 'uid'))  
 if 'EventKey' in self.msg and self.msg.get('EventKey'):  
 event\_key = self.msg.get('EventKey')[8:]  
 # 暂时一个用户只能绑定一个树莓派  
 self.redis.hset("user:%d" % user\_id, "device\_id", event\_key)  
 return ('恭喜您已经成功绑定家居客户端，可以试试下方菜单', BaseMsg.TEXT\_PLAIN)  
  
 else:  
 user\_id = self.redis.incr("user:id")  
 self.redis.sadd("user:list", user\_id)  
 self.redis.sadd("wx\_user:list", self.from\_user)  
 self.redis.hset("wx\_user:%s" % self.from\_user, 'uid', user\_id)  
 self.redis.hset("user:%d" % user\_id, "openid", self.from\_user)  
  
 if 'EventKey' in self.msg and self.msg.get('EventKey'):  
 event\_key = self.msg.get('EventKey')[8:]  
 # 暂时一个用户只能绑定一个树莓派  
 self.redis.hset("user:%d" % user\_id, "device\_id", event\_key)  
 return ('感谢关注家居小助手！您已经成功绑定家居客户端，可以试试下方菜单', BaseMsg.TEXT\_PLAIN)  
  
 return ('感谢关注家居小助手！为了提供更方便的服务，您需要绑定客户端', BaseMsg.TEXT\_PLAIN)

##### 2.1.2 用户注销

用户取消对公众号的关注时，只是设置用户的状态为注销，并不真正删除用户，代码如下：

def \_unsubscribe(self):  
 return ('成功取消关注', BaseMsg.TEXT\_PLAIN) # 不删除用户

##### 2.1.3 绑定家居客户端

绑定家居客户端，用户可以有两种选择：1. 使用微信全局的二维码扫描组件 2. 公众号内部的二维码扫描组件；两种扫描的方式向服务器端发送的消息不同，需要进行不同处理。

1. 全局的二维码扫描事件

# 处理扫描事件  
def \_scan(self):  
 # 这里缺少用户绑定新的树莓派客户端的检测  
 device\_id = self.msg.get('EventKey')  
 self.\_bind(device\_id)  
  
 return ('恭喜你已经成功绑定家居客户端', BaseMsg.TEXT\_PLAIN)  
  
# 处理绑定关系  
def \_bind(self, device\_id):  
 user\_id = int(self.redis.hget("wx\_user:%s" % self.from\_user, 'uid'))  
 self.redis.hset("user:%d" % user\_id, "device\_id", device\_id)  
 logging.info('user:%s bind device:%s' % (user\_id, device\_id))

1. 公众号内部的二维码扫描事件

def \_scan\_waitmsg(self):  
 event\_key = self.msg.get('EventKey')  
 if event\_key == 'BIND':  
 scan\_code\_info = self.msg['ScanCodeInfo']  
 scan\_res = scan\_code\_info['ScanResult']  
 if scan\_res and self.redis.exists(scan\_res):  
 device\_id = int(self.redis.get(scan\_res))  
 if device\_id:  
 self.\_bind(device\_id)  
 return '恭喜您已成功绑定设备', BaseMsg.TEXT\_PLAIN  
 else:  
 return '设备违法, 请使用原厂设备', BaseMsg.TEXT\_PLAIN  
 else:  
 logging.info('%r' % scan\_code\_info)  
 return '%s %s' % \  
 (scan\_code\_info['ScanType'],  
 scan\_code\_info['ScanResult']), BaseMsg.TEXT\_PLAIN  
 else:  
 return event\_key, BaseMsg.TEXT\_PLAIN

#### 2.2 设备管理

##### 2.2.1 增加设备，生成二维码

后台业务暂时还没有管理界面，但用户可以通过相应地接口来进行基础的信息管理，增加设备的接口为: {{DOMAIN}}/device/qr?q=generate&admin=hanliang, 设备二维码是通过curl调用微信二维码api生成的，其中包含着相应的设备信息，核心代码如下：

device\_id = self.redis.incr("device:id")  
self.redis.sadd("device:list", device\_id)  
access\_token = self.\_get\_access\_token()  
post\_data = {"action\_name": "QR\_LIMIT\_SCENE",  
 "action\_info": {"scene": {"scene\_id": device\_id}}}  
res = self.curl.post(('https://api.weixin.qq.com/cgi-bin/qrcode/'  
 'create?access\_token=%s'),  
 (access\_token,),  
 json.dumps(post\_data))  
res\_dict = json.loads(res)  
qr\_ticket = res\_dict.get('ticket')  
qr\_url = res\_dict.get('url')  
self.redis.hset("device:%d" % device\_id, 'qr\_ticket', qr\_ticket)  
self.redis.hset("device:%d" % device\_id, 'qr\_url', qr\_url)  
self.redis.set("%s" % qr\_url, device\_id)  
  
# 向客户端返回图片  
self.write(('<img src="https://mp.weixin.qq.com/cgi-bin/'  
 'showqrcode?ticket=%s" width=250px>') % qr\_ticket)

##### 2.2.2 查询指定设备的二维码

查询接口为: {{DOMAIN}}/device/qr?q=query&device\_id=1, 返回的为对应的二维码图片, 代码如下:

device\_id = int(self.get\_argument('device\_id', 0))  
if self.redis.sismember("device:list", device\_id):  
 qr\_ticket = self.redis.hget(  
 "device:%d" % device\_id, "qr\_ticket")  
 self.write(('<img src="https://mp.weixin.qq.com/cgi-bin/'  
 'showqrcode?ticket=%s" width=250px>') %  
 (qr\_ticket,))  
else:  
 self.write("Device does not exist !!!")

##### 2.2.3 添加传感器

当有室内设备向家居控制中心注册设备时，控制中心会调用相应地传感器增加接口注册设备；接口为: {{DOMAIN}}/device/(+)，调用方法为PUT，数据为json格式的。

1. 附加数据

{  
"device\_id": {{device\_id}},  
"sensor\_id": {{sensor\_id}},  
"sensor\_type": {{sensor\_type}}  
}

1. 核心代码如下

# Restful风格，处理put请求  
def put(self, device\_id):  
 logging.info('receive device info: %r', self.request.body)  
 data = json.loads(self.request.body)  
 sensor\_id, sensor\_type = \  
 data['sensor\_id'], data['sensor\_type']  
 sensor\_info = {'id': sensor\_id, 'type': sensor\_type}  
 device\_manager = DeviceManager(device\_id)  
 device\_manager.update\_device(sensor\_info)  
 self.write('update device info')  
   
# 更新device  
def update\_device(self, sensor\_info):  
 logging.info('update device %s sensor %r', self.device\_id, sensor\_info)  
 sensor\_id = sensor\_info['id']  
 sensor\_type = sensor\_info['type']  
 self.sensors.update({str(sensor\_id): {'type': sensor\_type}})  
 self.\_persist\_sensors()

##### 2.2.4 传感器数据存储、获取

当前版本只支持两种类型的传感器：开关类型传感器与温度湿度传感器或地理坐标，后续根据需求会增加更多类型传感器的支持。

1. 存储

上传数据格式为json，基本格式如下:

{  
"device\_id": {{device\_id}},  
"sensor\_id": {{sensor\_id}},  
"sensor\_value": {{sensor\_value}}  
}

核心处理逻辑，如下所示：

sensor\_manager = SensorManager(device\_id, sensor\_id)  
sensor\_type = sensor\_manager.get\_sensor\_type()  
data = json.loads(self.request.body)  
value = data['value']  
if sensor\_type == sensor\_manager.HUMTEM\_TYPE:  
 sensor\_manager.update\_sensor\_data(value)  
 self.write('Update sensor success')  
elif sensor\_type == sensor\_manager.LED\_TYPE:  
 value = int(value)  
 if value > 0:  
 value = 1  
 else:  
 value = 0  
 command = \  
 {'device\_id': device\_id,  
 'sensor\_id': sensor\_id,  
 'sensor\_value': value}  
  
 conn = rpyc.connect('127.0.0.1', 8889)  
 res = conn.root.handle\_msg(json.dumps(command))  
 conn.close()  
 self.set\_status(200)  
 self.set\_header("Content-Type", "application/json")  
 if res:  
 sensor\_manager.update\_sensor\_data(value)  
 res = {  
 'errcode': 0,  
 'errmsg': '',  
 'msg': 'Update sensor success'  
 }  
 else:  
 res = {  
 'errcode': -1,  
 'errmsg': '客户端未接入互联网或者已断线',  
 'msg': ''  
 }  
 self.write(json.dumps(res))

注：上面数据存储的逻辑中，如果sensor为开关型传感器，则会调用SOCKET系统提供的与客户端实时通信的rpc服务来控制家电设备的开关; 如果sensor为数值型传感器(温度湿度传感器等)，则存储当前时间戳与数据的对应关系。

##### 2.2.5 统一的web管理入口

不同的用户拥有不同的设备，其对应的管理入口也不相同，因此在用户进入到相应地管理界面之前需要进行基本的用户授权验证，当前版本采用微信的Oauth获得用户资料进行相应的权限管理；web界面采用Jinjia2模板和Bootstrap框架编写，既能够很好地与python集成，又能够动态适配pc和手机。

1. Oauth核心代码

class OauthHandler(tornado.web.RequestHandler):  
 """ 获取用来换取access\_token的code  
 """  
  
 def get(self):  
 redirect\_uri = \  
 config.wx\_oauth\_uri % (  
 config.appid, config.wx\_redirect\_uri,  
 config.wx\_snsapi\_base, random.choice(string.digits)  
 )  
 self.redirect(redirect\_uri)  
   
   
class AuthHandler(tornado.web.RequestHandler):  
 """ 通过code获取openid和access\_token  
 """  
 def get(self):  
 auth\_code = self.get\_argument('code', '')  
 if auth\_code:  
 auth\_uri = \  
 config.wx\_auth\_uri % (  
 config.appid, config.appsecret,  
 auth\_code,  
 )  
 res = mcurl.CurlHelper().get(auth\_uri, resp\_type='json')  
 logging.info('Get userinfo: %r', res)  
 if 'openid' in res:  
 r = \  
 redis.Redis(  
 host=config.redis\_host,  
 port=config.redis\_port,  
 db=config.redis\_db  
 )  
 u\_openid = res['openid']  
 uid = int(r.hget("wx\_user:%s" % u\_openid, 'uid'))  
 logging.info('uid: %s', uid)  
 device\_id = int(r.hget("user:%s" % uid, "device\_id"))  
 logging.info('uid: %s, device\_id: %s', uid, device\_id)  
 if device\_id:  
 self.redirect('/device/%s' % device\_id)  
 else:  
 self.redirect('error/101')  
 else:  
 self.write('授权失败，请重新授权')

1. 获取用户所能管理所有的设备以及其当前的状态

sensor\_type = int(self.get\_argument('type', 0))  
device\_manager = DeviceManager(device\_id)  
sensors = device\_manager.get\_all\_sensors(sensor\_type)  
logging.info(sensors)  
sensors = sorted(sensors.items(), key=lambda x: x[0])  
logging.info(sensors)  
  
humtem\_sensors = []  
led\_sensors = []  
tags = ['Home', 'Parlor', 'Bedroom', 'Kitchen']  
index = 0  
for sid, sinfo in sensors:  
 stype = sinfo['type']  
 tag = tags[index]  
 index += 1  
 sensor\_manager = SensorManager(device\_id, sid)  
 if stype == sensor\_manager.HUMTEM\_TYPE:  
 value = sensor\_manager.retrieve\_sensor\_data()[0] # 温度，湿度  
 humtem\_sensors.append(  
 {  
 "id": sid,  
 "tag": tag,  
 "value": value  
 }  
 )  
 elif stype == sensor\_manager.LED\_TYPE:  
 value = sensor\_manager.retrieve\_sensor\_data()  
 led\_sensors.append(  
 {  
 "id": sid,  
 "tag": tag,  
 "value": value  
 }  
 )  
  
self.render(  
 'sensors.html', device\_id=device\_id,  
 humtem\_sensors=humtem\_sensors,  
 led\_sensors=led\_sensors)

#### 2.3 聊天机器人

##### 2.3.1 解析用户消息

用户和机器人聊天可以发送两种类型的消息：文字型消息和语音型消息, 为了便于程序的识别，统一需要将消息转换为文字型消息。如果我们需要将语音自动转换为文字型消息，只需要打开微信公众平台的语音识别接口, 微信公众号平台便会自动的把语音消息以文字的格式推送给开发者业务网关。

##### 2.3.2 处理指令，回应消息

检查用户发送的消息中是否包含我们预先设置的控制或者管理指令，比如：开灯、关灯、温度湿度, 如果包含预先设置的指令则会调用相应的模块进行消息处理，否则则调用图灵机器人开放的人工智能接口进行对话，核心代码如下:

class VoiceMsg(object):  
 '''Voice Msg  
 '''  
 TEM\_COMMAND = "温度"  
 HUM\_COMMAND = "湿度"  
 LED\_COMMAND = "电灯"  
 LED\_COMMAND\_1 = "灯"  
 OPEN\_COMMAND = "开"  
 CLOSE\_COMMAND = "关"  
  
 def \_\_init\_\_(self, msg):  
 self.redis = redis.Redis(  
 host=config.redis\_host, port=config.redis\_port, db=config.redis\_db)  
 self.msg = msg  
 self.to\_user = msg.get('ToUserName')  
 self.from\_user = msg.get('FromUserName')  
 self.content = msg.get('Recognition').encode('utf-8')  
  
 def handle(self):  
 logging.info('Content: %r', self.content)  
 uid = int(self.redis.hget("wx\_user:%s" % self.from\_user, 'uid'))  
 if self.redis.hexists("user:%d" % uid, "device\_id"):  
 device\_id = int(self.redis.hget("user:%d" % uid, "device\_id"))  
 else:  
 device\_id = None  
 if self.TEM\_COMMAND in self.content or \  
 self.HUM\_COMMAND in self.content:  
 resp\_msg, resp\_msg\_type = self.\_handle\_hum\_tem\_msg(device\_id)  
 elif self.LED\_COMMAND in self.content or \  
 self.LED\_COMMAND\_1 in self.content:  
 if self.OPEN\_COMMAND in self.content:  
 resp\_msg, resp\_msg\_type = self.\_handle\_led\_msg(device\_id, 1)  
 elif self.CLOSE\_COMMAND in self.content:  
 resp\_msg, resp\_msg\_type = self.\_handle\_led\_msg(device\_id, 0)  
 else:  
 # resp\_msg, resp\_msg\_type = '要关还是开呢', BaseMsg.TEXT\_PLAIN  
 resp\_msg, resp\_msg\_type = self.\_handle\_tl\_msg()  
 else:  
 resp\_msg, resp\_msg\_type = self.\_handle\_tl\_msg()  
  
 curr\_timestamp = int(time.time())  
 if resp\_msg\_type == BaseMsg.TEXT\_PLAIN:  
 return config.TextTpl % (self.from\_user, self.to\_user,  
 curr\_timestamp, resp\_msg)  
 elif resp\_msg\_type == BaseMsg.TEXT\_MULTI:  
 items = ''  
 for content in resp\_msg:  
 items += config.MultiItemTpl % (  
 content['title'], content['description'],  
 content['picurl'], content['url'])  
 return config.MultiTextTpl % (self.from\_user, self.to\_user,  
 curr\_timestamp, len(resp\_msg), items)  
  
 def \_handle\_tl\_msg(self):  
 return tl\_msg.TlMsg(self.from\_user, self.content).get()  
  
 def \_handle\_hum\_tem\_msg(self, device\_id):  
 ...  
  
 def \_handle\_led\_msg(self, device\_id, value):  
 ...

注：图灵机器人接口代码参见代码附录

### 第三节 SOCKET系统

Socket系统主要用于管理服务器与众多客户端建立的长连接, 以及处理服务器端与客户端的实时信息, 基本上可以划分为两个模块：Socket连接管理与实时交流

#### 3.1 Socket连接

为了保持客户端与服务器端的持续的socket连接，需要发送定时的心跳信息，当前版本采用借助TCP/IP的keep-alive属性由客户端定时发送heartbeat来保持持续连接，代码如下：

class SocketServer(threading.Thread):  
  
 def \_\_init\_\_(self, host, port):  
 threading.Thread.\_\_init\_\_(self)  
 self.host = host  
 self.port = port  
 self.ss = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
 self.ss.setsockopt(socket.SOL\_SOCKET, socket.SO\_REUSEADDR, 1)  
 self.ss.bind((host, port))  
 self.ss.listen(100)  
 self.conns = {}  
 self.redis = redis.Redis(  
 host=config.redis\_host, port=config.redis\_port, db=config.redis\_db)  
  
 def serve(self):  
 logger.info('socket\_server starting...')  
 while True:  
 conn, addr = self.ss.accept()  
 login\_msg = conn.recv(2048)  
 try:  
 login\_msg = json.loads(login\_msg)  
 device\_id = int(login\_msg['device\_id'])  
 info = login\_msg['info']  
 if info == 'login' and \  
 self.redis.sismember('device:list', device\_id):  
 # 将conn与device\_id 对应起来  
 logger.info('device: %s login', device\_id)  
 self.conns[device\_id] = conn  
 response = {  
 'status': 0, 'err\_msg': '', 'info': "Login Success"}  
 conn.send(json.dumps(response))  
 else:  
 response = {'status': -1,  
 'err\_msg': 'invalid device\_id',  
 'info': 'Login error'}  
 conn.send(json.dumps(response))  
 except Exception, e:  
 logger.error('conn error: %r', e)  
  
 def close(self):  
 # 关闭客户端连接  
 for key in self.clients.keys():  
 try:  
 self.conns[key].close()  
 except Exception, e:  
 logger.error('close socket error: %r', e)  
 self.ss.close()  
  
 def run(self):  
 self.serve()

注：程序中是以device\_id来标示不同的socket连接的，所有的socket连接信息以dict方式放在内存之中。

#### 3.2 实时交流

与客户端的实时交流，需要借助客户端与服务器端建立的socket连接，而socket系统又不知道用户需要发送的信息或者指令，鉴于此socket系统需要为其他业务系统提供一个可以实时发送信息的接口。接口可以有很多种实现方式，这里采用了基于rpc的解决方案，提供了一套rpc接口，其他业务系统只需要调用就好了，核心代码如下所示:

class RpcService(Service):  
 """ RpcService，基于rpc提供一系列的接口  
 """  
  
 def exposed\_handle\_msg(self, msg):  
 ''' 消息处理接口，实时向客户端发送消息  
 '''  
 logger.info('receive msg: %r', msg)  
 msg\_dict = json.loads(msg)  
 device\_id = int(msg\_dict['device\_id'])  
 sensor\_id = int(msg\_dict['sensor\_id'])  
 sensor\_value = int(msg\_dict['sensor\_value'])  
 if device\_id in socket\_server.conns and \  
 socket\_server.conns[device\_id]:  
 conn = socket\_server.conns[device\_id]  
 req\_msg = {"device\_id": device\_id,  
 "sensor\_id": sensor\_id,  
 "sensor\_value": sensor\_value}  
 try:  
 conn.send(json.dumps(req\_msg)) # 这里客户端最好确认一下  
 conn.recv(2048) # 不接收数据，会直接踢掉客户端  
 return True  
 except Exception, e:  
 logger.error('socket send or recv error: %r', e)  
 conn.close()  
 socket\_server.conns.pop(device\_id) # 不接收数据的话，会直接导致删掉客户端  
 logger.info('close device %s socket', device\_id)  
 return False  
 else:  
 return False

### 第四节 家居控制中心

家居控制中心作为所有家居设备的协调器和控制器，主要负责各个家居设备的控制或者相关数据的上传，一定程度上可以看做服务器端与各个设备之间的一个中介，可以帮助用户更加方便的部署或扩展各个智能家居设备。

#### 4.1 Socket模块

借助TCP的keep-alive属性发送定时的heartbeat，与服务器保持实时连接，核心代码如下：

# keep-alive hearbeat  
def set\_keepalive\_linux(sock, after\_idle\_sec=1, interval\_sec=3, max\_fails=5):  
 """Set TCP keepalive on an open socket.  
 It activates after 1 second (after\_idle\_sec) of idleness,  
 then sends a keepalive ping once every 3 seconds (interval\_sec),  
 and closes the connection after 5 failed ping (max\_fails), or 15 seconds  
 """  
 sock.setsockopt(socket.SOL\_SOCKET, socket.SO\_KEEPALIVE, 1)  
 sock.setsockopt(socket.IPPROTO\_TCP, socket.TCP\_KEEPIDLE, after\_idle\_sec)  
 sock.setsockopt(socket.IPPROTO\_TCP, socket.TCP\_KEEPINTVL, interval\_sec)  
 sock.setsockopt(socket.IPPROTO\_TCP, socket.TCP\_KEEPCNT, max\_fails)  
   
  
def start():  
 s = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
 system\_name = platform.system()  
 if system\_name == 'Darwin':  
 set\_keepalive\_osx(s)  
 elif system\_name == 'Linux':  
 set\_keepalive\_linux(s)  
 s.connect((settings['server\_host'], settings['server\_port']))  
 login(s)  
   
# 登陆，进行服务器认证  
def login(s):  
 login\_msg = {'device\_id': settings['device\_id'], 'info': 'login'}  
 s.send(json.dumps(login\_msg))  
 res = json.loads(s.recv(1024))  
 status = res['status']  
 if status == 0:  
 logger.info('Login Success')  
 handleMessage(s)  
 elif status == -1:  
 print res['err\_msg']  
 s.close()

#### 4.2 Zigbee模块

为了让用户更加方便的部署各个家居设备，各家居设备并不直接连入互联网，家居设备与服务器端的通信都是通过控制中心来通信的，这时候选择一个好的将家居设备与控制中心连接起来的解决方案就至关重要了，国际上有几种用户智能家居的互联协议：Zigbee、Z-Wave等，其中的ZigBee比较成熟支持大范围的节点部署。该系统采用了ZigBee协议建立了与各个家居设备的通信，核心代码如下所示：

class ZigManager(object):  
 '''ZigManager  
 '''  
 SERIAL\_PORT = '/dev/ttyAMA0'  
 SERIAL\_RATE = 38400  
  
 CMD\_REGISTER = '1'  
 CMD\_DATA = '2'  
  
 SLEEP\_INTERVAL = 120 # 调小时间间隔，防止5分钟发送的数据是错误的，这样可以重试3次  
  
 def \_\_init\_\_(self):  
 self.com = serial.Serial(self.SERIAL\_PORT, self.SERIAL\_RATE)  
  
 def read(self):  
 data = self.com.readline()  
 logger.info('Read msg: %r', data)  
 return data  
  
 def write(self, msg):  
 msg += "\\"  
 length = self.com.write(msg)  
 logger.info('Write msg: %r, len is: %s', msg, length)  
  
 def monitor(self):  
 while True:  
 data = self.read().split()  
 logger.info(data)  
 device\_id = settings['device\_id']  
 cmd = data[0]  
 sensor\_id = int(data[1])  
 sensor\_type = int(data[2])  
 if cmd == self.CMD\_REGISTER:  
 sensor = Sensor(device\_id, sensor\_id, sensor\_type)  
 SensorManager.register(sensor)  
 elif cmd == self.CMD\_DATA:  
 sensor\_value = data[3]  
 sensor = Sensor(device\_id, sensor\_id, sensor\_type, sensor\_value)  
 SensorManager.upload(sensor)  
  
 ...

#### 4.3 Deamon模块

家中网络有可能出现波动，有可能在某一段时间内断网，为了让家居控制中心能够动态的适应状况，让其在网络连通的时候自动重启，继续保持与服务器的连接，特别开发了deamon模块动态检测网络的连通性并在合适的时候重启网络，核心代码如下所示：

def manage():  
 network\_status = 0  
 while True:  
 if network\_test('www.baidu.com'):  
 # logger.info('network is ok')  
 if network\_status == 0:  
 restart\_client() # restart client  
 network\_status = 1  
 time.sleep(60)  
 elif network\_status == 1:  
 # 需要检测client是否存在，不存在则重启  
 keep\_client\_exist() # 确保客户端存在  
 else:  
 logger.error('network error')  
 network\_status == 0  
 time.sleep(20)  
  
  
def restart\_client():  
 os.system("ps -ef | grep bin/start.py | grep -v grep | awk '{print $2}'"  
 "| sed -e 's/^/kill -9 /g' | sh -")  
 os.system("cd /data/raspberry-pi-wechat-client && PYTHONPATH=. "  
 "python bin/start.py &")  
 logger.info('restart client ...')  
  
  
def keep\_client\_exist():  
 count = int(os.popen("ps -ef | grep bin/start.py | grep -v grep"  
 "| wc -l").read())  
 time.sleep(1)  
 if count == 0:  
 restart\_client()  
  
  
def stop\_client():  
 os.system('''ps -ef | grep bin/start.py | grep -v grep | awk '{print  
 $2}' | sed -e "s/^/kill -9 /g" | sh -''')  
 logger.info('stop client ...')

### 第五节 Sensor控制子系统

Sensor控制子系统用于对具体的Sensor进行相应地控制管理，每一个Sensor应该有一个属于自己的专门管理系统来管理Sensor并负责与控制中心的交互。该系统借助于Arduino（单片机）来实现，主要包括两个部分：Zigbee通信与信息处理。

#### 5.1 ZigBee

Zigbee通信过程中需要将信息组装成对方能够理解的格式或者将对方发送过来的信息解析出来，其中涉及到数据包接收解析与组装发送，核心代码如下所示：

/\*  
\*\* 循环, 判断是否接受到新消息  
\*/  
void loop() {  
 if(sSerial.available()){  
 unsigned char count = sSerial.readBytesUntil(END\_FLAG, rx\_data, 90);  
 unpack\_msg(rx\_data, count);  
 Serial.println("Receive msg: " + String((char \*)rx\_data));  
 handle\_msg((char \*)rx\_data);  
 }  
  
 if (Serial.available()){  
 Serial.println(Serial.read());  
 }  
}  
  
  
/\*\*  
\*\* 向控制中心发送指令  
\*/  
void send\_msg(String msg){  
 len = msg.length();  
 pack\_msg(msg, tx\_data, len);  
 sSerial.write(tx\_data, len+1);  
 Serial.println("Send msg: " + msg);  
}  
  
  
/\*\*  
\*\* 封装将要发送的消息，在消息末尾添加\n  
\*\* @param msg 将要发送的消息  
\*/  
void pack\_msg(String msg, char \*msg\_arr, unsigned char n){  
 for(char i=0; i < n; i++){  
 msg\_arr[i] = msg.charAt(i);  
 }  
 msg\_arr[n] = NL; //添加换行符  
}  
  
  
/\*\*  
\*\* 解析发送的信息，去除结束符  
\*/  
void unpack\_msg(unsigned char \* msg, unsigned char n){  
 for(unsigned char i=0; i<n-1; i++){ // 去除结束符  
 msg[i] -= 128;  
 }  
}

#### 5.2 信息处理

不同的Sensor控制系统有不同的信息处理方式，这里以开关型设备为例，其主要代码如下：

/\*\*  
\*\*处理控制中心发送过来的命令  
\*/  
void handle\_msg(char \*msg){  
 char \*valPosition = strtok(msg, " "); // msg sensor\_id sensor\_value  
 int data[4];  
 char i = 0;  
 while(valPosition!=NULL){  
 data[i] = int(\*valPosition) - 48;  
 valPosition = strtok(NULL, " ");  
 i = i + 1 ;  
 }  
  
 if (data[0] == SENSOR\_ID){  
 Serial.println("led");  
 if(int(data[1]) > 0){  
 digitalWrite(LED\_PIN, HIGH);  
 }else{  
 digitalWrite(LED\_PIN, LOW);  
 }  
 }  
}

注： 当前系统指令的格式为 "{{sensor\_id}} {{sensor\_value}}", 在处理的时候需要对字符串进行拆分；另外，当前系统各个ZigBee芯片之间的通信方式为广播类型的，控制中心发送的指令所有子系统都能够接收的到，在信息处理的时首先需要判断sensor\_id是否为自己，然后再进行基本的处理。

## 第五章 系统测试

## 第六章 总结与展望

## 致谢

## 参考文献

## 附录：部分源程序清单