**暨南大学本科实验报告专用纸**

课程名称 物联网应用课程设计 成绩评定

实验项目名称 综合实验-办公楼智能控制系统 指导教师 李晓帆

实验项目编号 0814400102 实验项目类型 验证 实验地点 B305

学生姓名 郑晓鹏 学号 2022101950 学院 智能科学与工程学院 专业 物联网工程 实验时间 2024 年 11 月 5 日

**实验目的**

1. 理解物联网技术在碳排放监测中的应用：掌握如何使用传感器和物联网平台进行碳排放数据的采集和分析。
2. 设计和实现一个物联网综合应用系统：通过硬件和软件的集成，设计一个能够实时监测、分析和优化能耗和碳排放的系统。

**实验过程**

1. 确定实验方案，构建系统结构框图

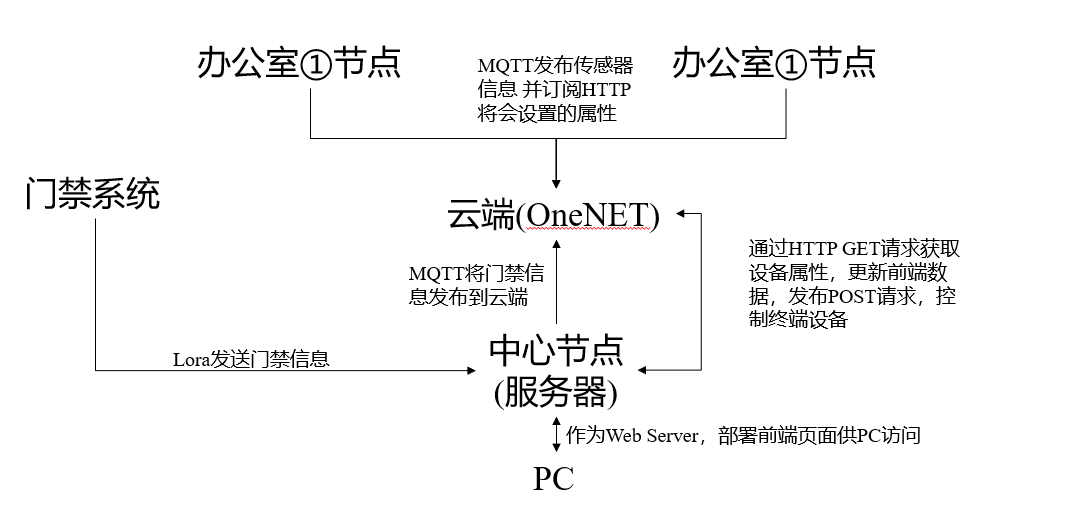
为构建办公楼智能控制系统模型机，首先确定实验模拟的办公楼场景为：门禁系统、办公室①及办公室②，另需部署中心节点(服务器)、云端，以便将各个节点进行组网构建系统。

部署门禁系统节点：以Arduino Mega 2560作为主控，外接RFID读卡器和LoRa模块，以便将门禁信息通过LoRa点对点发送至中心节点并转发保存至云端。

部署办公室①节点及办公室②节点：以ESP8266作为主控，传感器采用：温湿度传感器、火焰传感器、人体红外传感器、CO2/CH2O传感器、光敏传感器；通过传感器数据进行自动控制逻辑控制终端设备：LED灯珠(模拟灯光系统)、继电器①(模拟控制空调)、继电器②(模拟控制排气系统)；同时通过ESP8266 WiFi模组将传感器数据以及设备工作状态采用MQTT协议上传到云端以便Web页面获取动态数据并展示，同时订阅前端下行控制信息对应的MQTT主题。

部署中心节点(服务器)：以ESP32-S3作为主控，外接LoRa模块与门禁系统进行点对点通信，将从LoRa模块接收到的门禁信息转发至云端，以供Web页面拉取并显示。同时部署在ESP32-S3上部署服务器，将Web页面相关文件挂载在文件系统上，以供服务器根据响应访问对应的文件。

部署云端：为了存储传感器以及设备运行记录等数据，同时便于整个系统各个节点对这些数据进行访问，因此利用云平台的特性，将这些数据存储至云端，每个想要访问数据的节点直接对云平台的数据进行读写即可，可以理解为各个节点的共享内存。对比常用的云平台，OneNET不需要购买增值服务也能很好的符合我的实验要求，其增值部分可视化的功能可以通过前端来实现，因此我选用OneNET平台并部署云端。



*系统框图*

1. 系统的各个节点的功能实现
2. 云端

首先进入OneNET官网(<https://open.iot.10086.cn/>)，注册账号，打开开发者中心，创建产品，节点类型为“直连设备”，接入协议为“MQTT”，数据协议为“OneJson”，联网方式为“Wi-Fi”。



一个产品下可以挂载多个同类型的设备，这里的“设备”指的是MQTT客户端，即为ESP8266/ESP32，每个传感器以及控制终端的数据可以通过“设备属性”体现，比如温度、湿度可以作为office\_1节点的temp属性和humi属性进行存储并体现，所有的设备属性即为“物模型”，同类型指的是共用一套物模型，也就是说一个产品下的所有设备的物模型是一样的。

由于本系统需要将门禁信息、温湿度传感器、CO2/CH2O传感器、人体传感器、火焰传感器、排气、灯、空调信息上传至云端，因此创建对应物模型如下并导入。

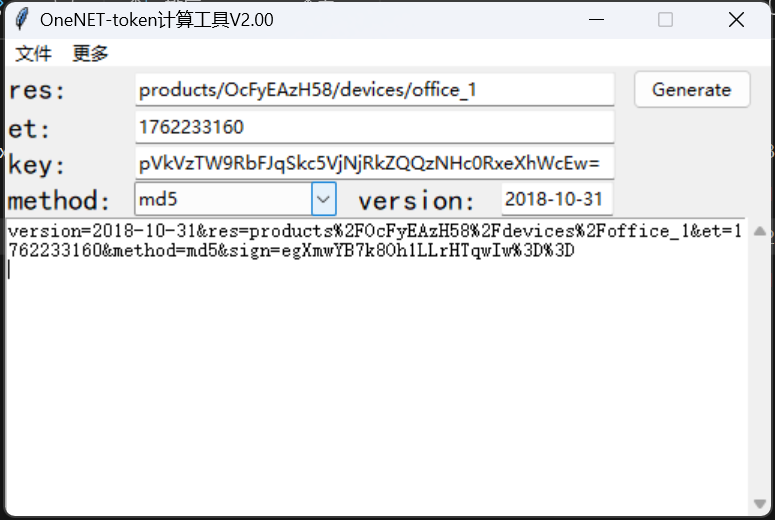
|  |
| --- |
| {      "properties": [        {          "identifier": "access",          "dataType": {            "type": "string"          }        },        {          "identifier": "airconductor",          "dataType": {            "type": "bool"          }        },        {          "identifier": "ch2o",          "dataType": {            "type": "float"          }        },        {          "identifier": "co2",          "dataType": {            "type": "int32"          }        },        {          "identifier": "exist",          "dataType": {            "type": "bool"          }        },        {          "identifier": "fan",          "dataType": {            "type": "bool"          }        },        {          "identifier": "fire",          "dataType": {            "type": "bool"          }        },        {          "identifier": "humi",          "dataType": {            "type": "float"          }        },        {          "identifier": "light",          "dataType": {            "type": "bool"          }        },        {          "identifier": "temp",          "dataType": {            "type": "float"          }        }      ],      "events": [],      "services": [],      "combs": []  } |

产品物模型创建完毕后，接着创建三个设备。设备名称分别为“office\_1”、“office\_2”、“center”，分别对应办公室①节点，办公室②节点以及中心节点。



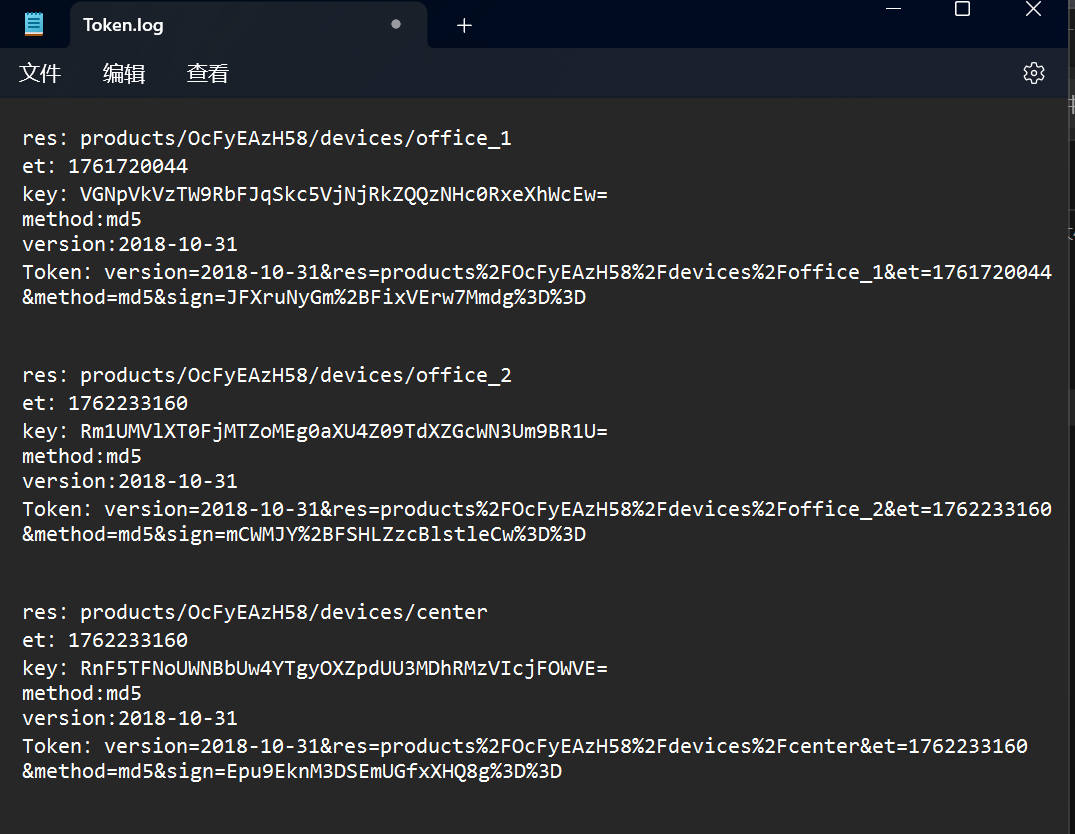
将产品ID、各个设备的ID以及其对应的access\_key复制下来，以便生成token供MQTT以及HTTP通信时进行安全鉴权：

|  |
| --- |
| 产品ID：OcFyEAzH58，  设备ID：office\_1  access\_key：VGNpVkVzTW9RbFJqSkc5VjNjRkZQQzNHc0RxeXhWcEw=  设备ID：office\_2  access\_key：Rm1UMVlXT0FjMTZoMEg0aXU4Z09TdXZGcWN3Um9BR1U=  设备ID：center  access\_key：RnF5TFNoUWNBbUw4YTgyOXZpdUU3MDhRMzVIcjFOWVE= |

在官网下载OneNET-token计算工具：[OneNET - 中国移动物联网开放平台](https://open.iot.10086.cn/doc/v5/fuse/detail/1487)并打开，设置一个token过期时间并使用[Unix时间戳(Unix timestamp)在线转换工具-BeJSON.com](https://www.bejson.com/convert/unix/)生成为秒级的Unix时间戳即为参数et，由于我们进行MQTT以及HTTP拉取以及上传数据时都是针对设备的，因此res字段应该为：products/{产品id}/devices/{设备名}，key为设备的access\_key，method以及version采用默认值即可。

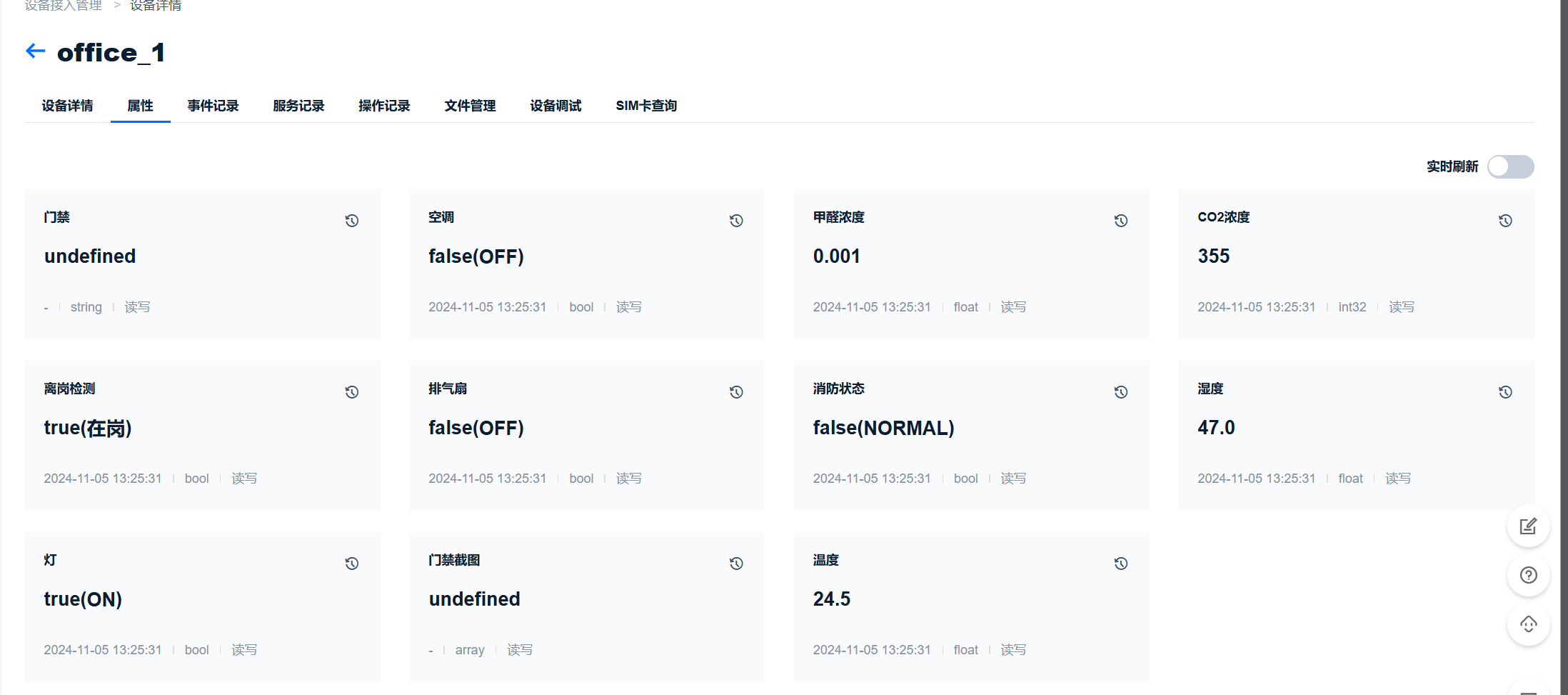
*以office\_1为例*

如上生成的version开头的字段即为该设备的token，同理生成office\_2以及center的token。生成记录会在token助手的同级目录下生成一个Token.log文件。

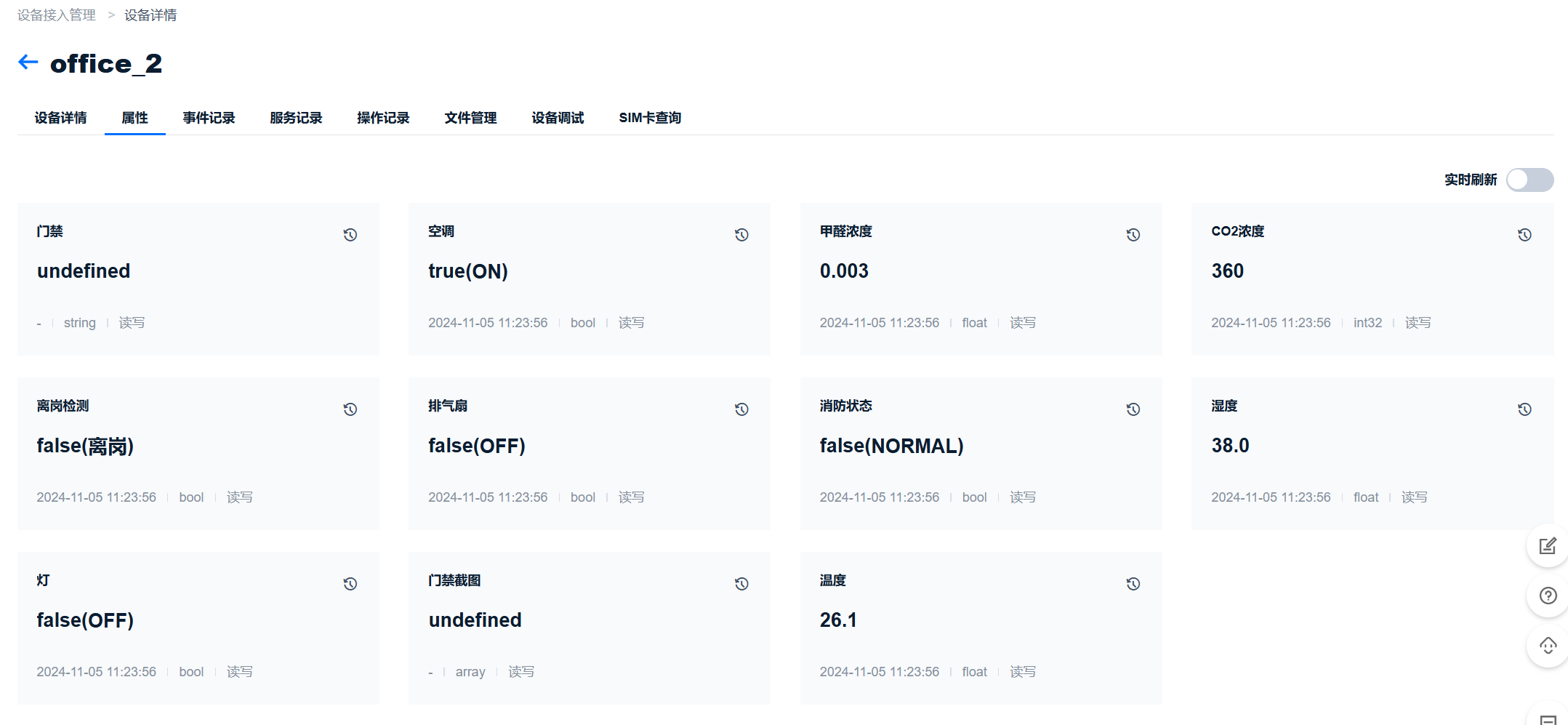


*Token.log文件内容*

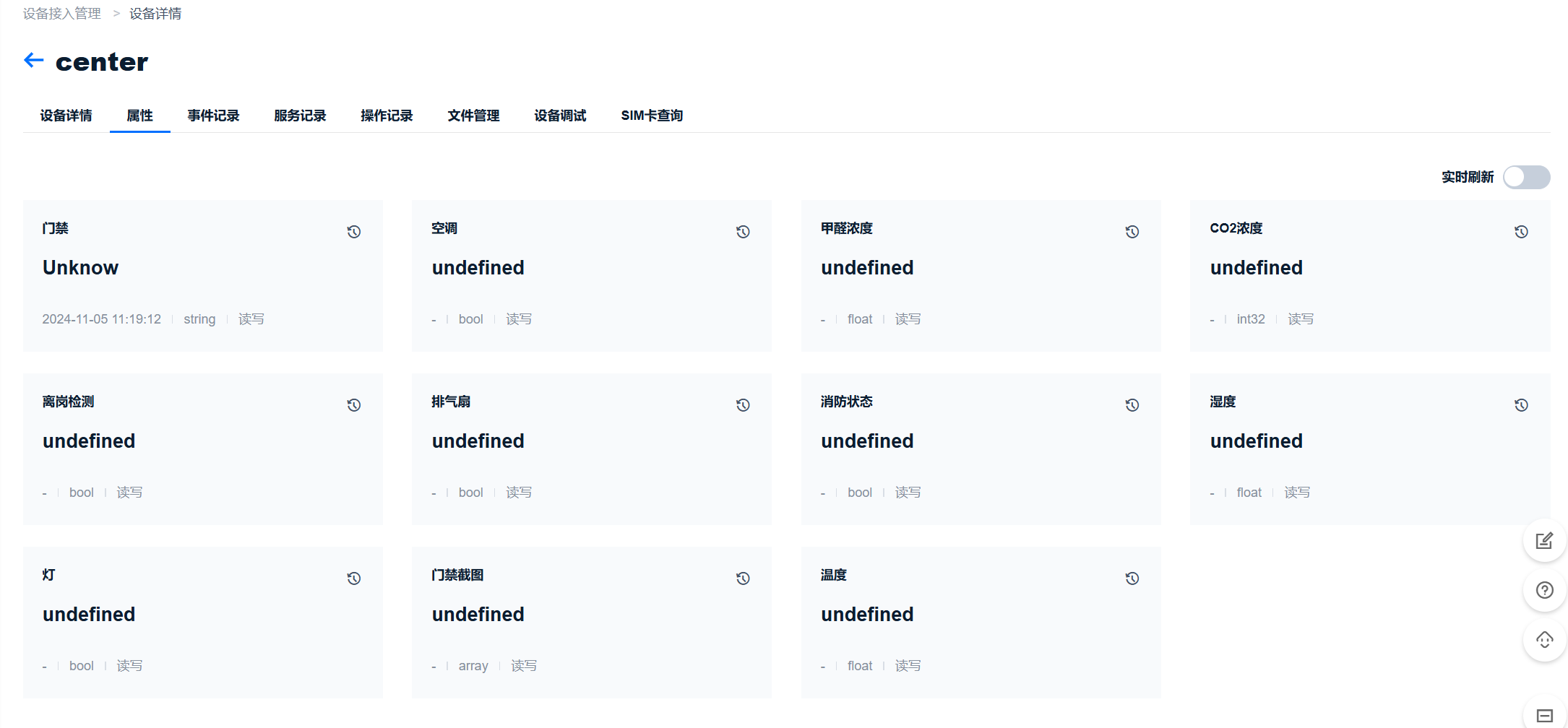
至此云端的准备工作基本完成，一下为各个节点的属性页面预览，通过设备属性面板也能得知各个设备节点属性的实时数据。



*office\_1 节点设备属性面板*



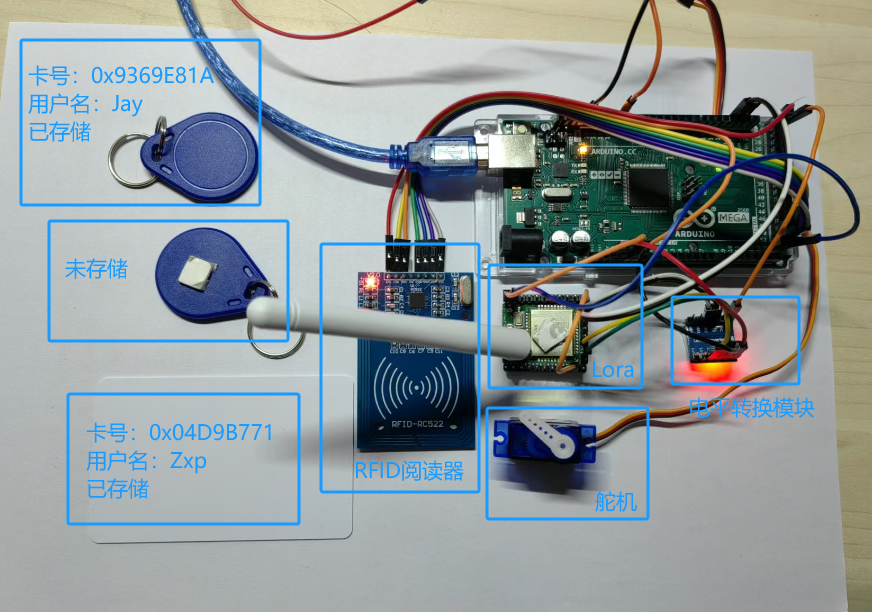
*office\_2 节点设备属性面板*



*center 节点设备属性面板*

1. 门禁系统

使用实验平台自带的Arduino Mega 2560、MC522 RFID读卡器、S50卡、舵机，以及自行购置的LoRa模组。读卡器与Arduino采用SPI通信，接线为：RST-49，MISO-50，MOSI-51，SCK-52，SDA-53，VCC接Arduino的5V引脚，GND接GND。由于Arduino逻辑电平是5V，而所用的LoRa模组的逻辑电平为3.3V，为了保障通信正常加装一个电平转换模块将串口1发出的5V信号转换为3.3V给LoRa模块。



使用Arduino IDE的RFID.h库函数，读取对应的RFID标签的UID。

|  |
| --- |
| void RFID\_Read(void)  {    if(rfid.findCard(PICC\_REQIDL, read\_id) == MI\_OK)    {      // 防冲突检测,读取卡序列号      if(rfid.anticoll(read\_id) == MI\_OK)      {        RFID\_Check();   // 读取到卡号,进行验证      }      //选卡（锁定卡片，防止多数读取）      rfid.selectTag(read\_id);    }    rfid.halt();  //命令卡片进入休眠状态  } |

由于本平台配备的S50卡卡号长度位8位，经过测试已知两张卡的卡号分别为Ox04D9B771、0x9369E81A, 用户名分别定义为：“Zxp”“Jay”。为了将卡号和用户名对应，同时方便后续做扩展，因此采用结构体模拟键值对来存储卡号信息。

|  |
| --- |
| struct Record  {    unsigned char id[MAX\_LEN];    String name;  }record[RECORD\_MAX\_NUM]; |

LoRa模块是通过串口与Arduino Mega进行通信的。由于Arduino的串口0是调试串口，为了避免麻烦，使用串口1与LoRa模块进行通信。为了信息传输安全，采用串口发送AT指令给LoRa模块，而不采用透传的方法，因而接线为TX1-DBG\_RX，RX1-DBG\_TX，该串口的工作波特率为115200。

需要使用到的使用的AT指令分别为：AT+NCONFIG=D2D,<ADDR>,<FREQGRP>,<LEMODE>,<TXPWR>,<RATE>\r\n

AT+SETKEY=ADD,<KEY>,<CRC32>\r\n

AT+SEND=<LEN>,<DATA>,<DST>,<WAKE>,<QOS>,<TTL>\r\n

首先，对LoRa模块进行初始化。通过串口1对发送AT指令配置点对点模式，同时为了保障数据安全设置密钥。

AT+NCONFIG=D2D,1122,10,0,20,2\r\n

发射节点地址1122，频点组号10，节能模式typeA，发射功率20dbm，比特率19.2kbps。

AT+SETKEY=ADD,18025573826180255738261802557382,C48A5FAE\r\n

设置密钥为18025573826180255738261802557382，C48A5FAE为密钥对应的CRC校验码。

初始化代码如下：

|  |
| --- |
| void Lora\_Init(void)  {    Serial1.begin(115200);    Serial1.println("AT+NCONFIG=D2D,1122,10,0,20,2");    delay(500);    Serial1.println("AT+SETKEY=ADD,18025573826180255738261802557382,C48A5FAE");    delay(500);  } |

由于LoRa通信只能发送字节数据，而我想要发送的门禁信息为用户名(已知的卡号发送对应用户名，陌生的卡号发送Unknow)，因此需要进行信源编码。我采用的编码方式为将字符串分为单个字符并分别发送其对应的ASCII码。编写函数如下：

|  |
| --- |
| String string2HexString(const unsigned char\* name)  {    String hexStr = "";    while (\*name)    {      // 将每个字符转换为两位十六进制      char byteHex[3] = {0};      sprintf(byteHex, "%02X", (unsigned char)\*name);      hexStr += byteHex;      name++;    }    return hexStr;  } |

编码后的数据便可以使用AT指令经过LoRa模块发送给中心节点，但是由于用户名的长度不一且没有什么明确的边界，如果不做处理接收端不好进行读取的判断，因此要进行信道编码。在要发送的字节串前增加帧头0xFF，帧尾0x0A，以便接收端根据帧头作为开始接受标志，帧尾作为接收结束标志位。

|  |
| --- |
| void Lora\_SendName(const char name[])  {    String hexStr = string2HexString(name);    // 构建AT命令的基础部分    String command = "AT+SEND=";      command += (hexStr.length()/2 + 2);    command += ",FF" + hexStr + "0A,3344,1,0,0";    // 0xFF作为起始符，0x0A作为终止符 目标节点3344 发送唤醒位 不重传 不转发      // 通过串口发送命令    Serial.println(command);    Serial1.println(command);  } |

当读取到卡号后，与已存储的卡号进行比对，若遍历到有符合的卡号，则通过LoRa模块发送用户名给中心节点，同时控制舵机开门并关门；若遍历完没找到符合的卡号，则发送Unknow给中心节点。

|  |
| --- |
| void RFID\_Check(void)  {    char checkFlag = 0;    for(int i = 0; i < RECORD\_MAX\_NUM; i++)    {      if(strncmp(read\_id, record[i].id, 4) == 0)      {        // 门禁验证成功        checkFlag = 1;        Serial.print("Welcome "); Serial.println(record[i].name.c\_str());        Lora\_SendName(record[i].name.c\_str());        Servo\_Control();        break;      }    }    if(!checkFlag)    {      Serial.println("Unknow");      Lora\_SendName("Unknow");    }  } |

1. 办公室节点

使用ESP8266作为主控，配备温湿度传感器、光敏传感器、火焰传感器、人体红外传感器、CO2/CH2O传感器、LED灯珠、继电器①、继电器②。

温湿度传感器选用的是DHT11模块，其内部集成了模数转换电路，同时会根据其定义的数据传输协议将温湿度传感器数据输出为串行数据，只需要根据给定的时序逻辑读取即可。由于Arduino开发平台有DHT11的库函数，因此我们并不需要根据其时序重写其读取函数，只需要将模块的数据引脚连接至单片机的数字引脚并实例化DHT类即可调用相关函数，大大降低了开发复杂度。

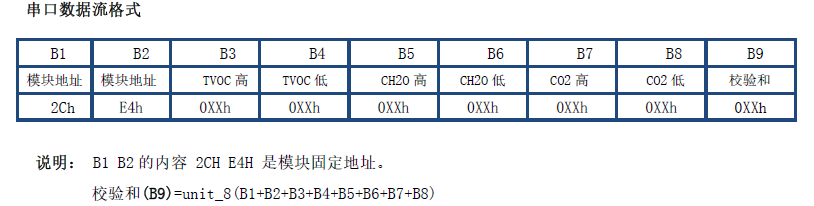
|  |
| --- |
| float temp = 0.0;  float humi = 0.0;  int pinDHT = D7;  DHT dht(pinDHT, DHT11); // 实例化DHT类  void DHT11\_Init(void)  {    dht.begin();  }  void getTempHumi(void)  {    humi = dht.readHumidity();    temp = dht.readTemperature();    bool readSuccess;    if((isnan(humi) || isnan(temp)))  readSuccess =  false;    else readSuccess = true;  } |

光敏传感器是根据光照强度改变光敏电阻的阻值，输出为模拟量，因此需要接ESP8266的模拟引脚A0，通过分压值判断光照的强度，光照强度越大，光敏电阻阻值越低，分压值越低，AD采样读取到的值也就越小。设定一个阈值，当采样值高于阈值，即光照强度小于一定程度，进行逻辑判断控制开灯。

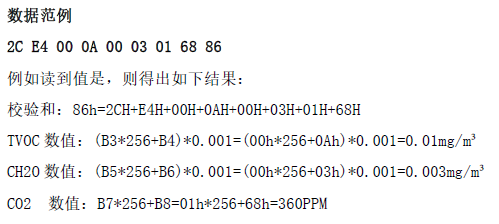
火焰传感器可以输出数字量和模拟量，由于ESP8266只有1路ADC，因此只能舍弃更为精确的模拟量采用数字量。通过调节其上的电位器可以改变对火焰特定波的灵敏度，当范围内检测到火焰，数字口会输出高电平，否则输出低电平，因此直接将数字口接在单片机的数字引脚并配置为输入模式即可。

人体红外传感器选用HC-SR501模块，由于人体温约为37℃，会辐射出波长10μm左右的红外线，模块加装的菲涅尔透镜可以将周围的红外信号折射到传感器上。通过调节两个电位器可以改变灵敏度和触发时长，当检测到有人便会输出高电，即输出数字量，直接将数据引脚接入单片机的数字引脚并配置为输入模式即可。

由于CO2/CH2O传感器是通过串口回传数据的，而ESP8266的串口1只有TX能正常使用，RX由于和别的外设复用了引脚不能使用。因而不得不采用默认串口0的RX接收传感器的数据流，而默认串口0的RX用于烧录程序，因此在每次程序烧录时需要断开该传感器和ESP8266串口的连接(注意不要直接断电，该传感器每次断电后都需要1分钟时间预热)。数据流格式如下：



因此为了获取CO2和CH2O的数据，只需要将串口流数据接收，然后分别将B5、B6、B7、B8的数据用于换算即可。换算关系如下：



以上便是关于传感器部分的介绍，接下来介绍系统自动控制逻辑。(以下所说的置位为置true/1；置false/0称为清零)首先优先级最高的是来自前端控制的终端设备(灯光、排气、空调)运行允许位(lightAllow、fanAllow、airConductorAllow)，若该标志位为false，即不允许终端设备运行，具有强制关闭的功能；该标志位为true才允许自动控制系统的介入，进而根据诸如光照强度、温度、气体质量等实测数据进行自动控制终端设备。

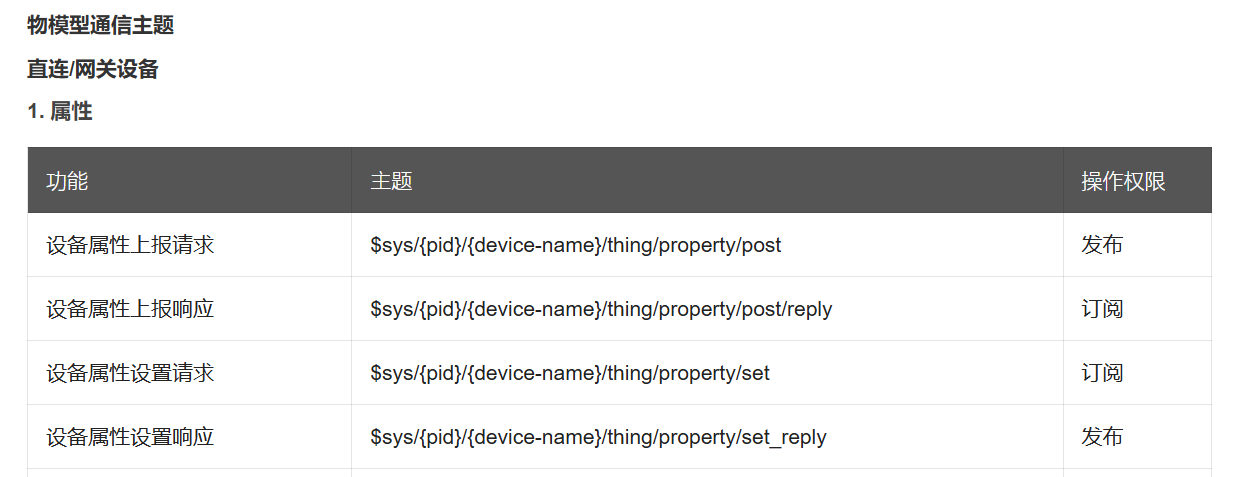
人体红外传感器检测到有人在座位上后，会置在岗标志位(existState)；当光线低于阈值后，同时在岗标志位为true且灯光运行允许位为true，且才会执行开灯的逻辑，同时置灯状态标志位(lightState)。考虑到实际人可能会临时短暂离岗，因此在岗标志位并不介入空调与排气系统的控制。当温度高于阈值(tempThreshold)且空调运行允许位为true，才会执行开启空调的逻辑，同时置空调状态标志位。当二氧化碳浓度高于阈值(co2Threshold)或者甲醛含量高于阈值(ch2oThreshold)，同时排气运行允许位为true，才会执行开启排气系统的逻辑，同时置排气系统状态标志位。当有运行允许位为false，或者自动控制逻辑结果为false，则会清零终端设备状态标志位，并关闭设备。

执行控制逻辑的同时会定时将所有的传感器数据以及设备运行数据(各个标志位)上报给云端。下面介绍ESP8266通过MQTT与OneNET的通信逻辑。MQTT采用了发布/订阅的消息传递模式，发布者（Publisher）发送消息到指定的主题（Topic），而订阅者（Subscriber）则订阅这些主题以接收消息，由于Arduino IDE有关MQTT的库，所以不对MQTT协议进行过多的介绍。我所用的MQTT库是PubSubClient，版本是2.8。需要注意的是，由于我一次上报的MQTT报文长度大约是200字节左右，加上心跳包等包头包尾信息已经超过256字节，而该版本的报文长度限制是256,导致报文发送显示成功，实则发送失败，需要将PubSubClient.h的宏定义MQTT\_MAX\_PACKET\_SIZE改为512。此外由于采用的数据传输格式为JSON，Arduino IDE的语言环境并不会对JSON格式进行解析，因此需要导入相关库进行额外处理，我所用的是ArduinoJson，版本号为6.15.1(注意：高版本的ArduinoJson并不兼容ESP8266)。另外为了实现程序能多线程运行，为了方便不移植RTOS，采用Ticker库创建定时器模拟多线程，版本号为2.0.0，需要注意的是，一个Ticker对象只能绑定一个任务函数句柄，且一个任务持续事件不可过长，以免进入中断嵌套造成程序死锁，不过Ticker类执行过程中会隐式执行喂狗操作，因此死锁后没能及时喂狗，看门狗会自动复位程序，但是为了程序的健壮性不建议这么操作。

OneNET的MQTT服务器域名为mqtts.heclouds.com，端口号为1883，在通过MQTT访问时需要提供产品id、设备id以及通过access\_key计算出的访问token进行安全验证。



<https://open.iot.10086.cn/doc/v5/fuse/detail/920> 此链接为OneNET的MQTT通信主题列表，本实验使用到的为设备属性上报请求、设备属性上报响应、设备属性设置请求、设备属性设置响应。



(以下以office\_1节点的代码进行举例解释，office\_2与office\_1代码差别只有设备名和token的宏定义不同)为了程序编写的方便，增强可读性，将上面这几个topic字符串进行宏定义，注意由于宏定义的是字符串，同时为了提高程序的可移植性，方便更换设备名和产品名，因此宏定义主题字符串的时候采用字符串字面量和宏替换结合来定义几个主题宏，如下。

|  |
| --- |
| #define product\_id  "OcFyEAzH58"  #define device\_id  "office\_1"  #define ONENET\_TOPIC\_PROP\_POST "$sys/" product\_id "/" device\_id "/" "thing/property/post"   // 设备属性上报请求主题  #define ONENET\_TOPIC\_PROP\_SET "$sys/" product\_id "/" device\_id "/" "thing/property/set"  #define ONENET\_TOPIC\_PROP\_POST\_REPLY "$sys/" product\_id "/" device\_id "/" "thing/property/post/reply"   // 上报响应  #define ONENET\_TOPIC\_PROP\_SET\_REPLY "$sys/" product\_id "/" device\_id "/" "thing/property/set\_reply" |

同时为了方便，将OneJSON消息格式也做宏定义，顺便将需要上报的JSON参数列表也做宏定义，同时由于上报Bool类型需要上传的是“true”和“false”，因此顺便定义一个宏函数，如下。

|  |
| --- |
| #define ONENET\_TOPIC\_PROP\_FORMAT "{\"id\":\"%u\",\"version\":\"1.0\",\"params\":%s}"     // 设备属性上报格式模板  #define JSON\_PARAMS\_FORMAT "{\"temp\":{\"value\":%.1f},\"humi\":{\"value\":%.1f},\"co2\":{\"value\":%d},\"ch2o\":{\"value\":%.3f},\"airconductor\":{\"value\":%s},\"exist\":{\"value\":%s},\"fan\":{\"value\":%s},\"fire\":{\"value\":%s},\"light\":{\"value\":%s}}"  #define BOOL2STRING(state) state ? "true" : "false" |

以上准备工作做完后，属性上报的工作量就大大减少了，只需要用sprintf函数调用JSON消息格式宏和params参数宏即可，同时又能很好的提高代码的可读性，如下。

|  |
| --- |
| // 上报设备属性  void OneNet\_Prop\_Post(void)  {    if(client.connected())    {      char params[256];   // 属性参数      char jsonBuf[256];  // json数据包缓冲区      sprintf(params, JSON\_PARAMS\_FORMAT, temp, humi, co2, ch2o, BOOL2STRING(airConductorState), BOOL2STRING(existState), BOOL2STRING(fanState), BOOL2STRING(fireState), BOOL2STRING(lightState));      // Serial.println(params);      sprintf(jsonBuf, ONENET\_TOPIC\_PROP\_FORMAT, postMsgId++, params);      Serial.println(jsonBuf);      if(client.publish(ONENET\_TOPIC\_PROP\_POST, jsonBuf))      {        Serial.println("Post property success!");      }      else      {        Serial.println("Post property failed~");      }    }  } |

此外还需要定义回调函数对设置设备属性主题进行订阅，监听是否有来自前端的HTTP设置请求，并解析JSON报文进行相应的设备允许位的置位、清零逻辑处理，如下。

|  |
| --- |
| // 当订阅的主题有消息时，会调用此函数  void callback(char\* topic, byte\* payload, unsigned int length)  {    Serial.print("Message arrived[");    Serial.print(topic);    Serial.print("]");    Serial.println();    if(strcmp(topic, ONENET\_TOPIC\_PROP\_SET)==0)    {      DynamicJsonDocument doc(100); //创建json文档用于解析消息内容 一次不会有那么多字节，100字节够用了      DeserializationError error = deserializeJson(doc, payload);   // 解析消息内容      if(error)      {        Serial.print(F("deserializeJson() failed"));        Serial.println(error.c\_str());        return;      }      JsonObject setAlinkMsgObj = doc.as<JsonObject>();   // 创建json文档的根对象      JsonObject params = setAlinkMsgObj["params"];   // 获取params对象      if(params.containsKey("temp"))      {        temp = params["temp"];        Serial.print("Set temp:");        Serial.println(temp);      }      if(params.containsKey("humi"))      {        humi = params["humi"];        Serial.print("Set humi:");        Serial.println(humi);      }      if(params.containsKey("fire"))      {        fireState = params["fire"];        Serial.print("Set fireState:");        Serial.println(fireState);      }      if(params.containsKey("light"))      {        lightAllow = params["light"];        Serial.println("Set light");      }      if(params.containsKey("airconductor"))      {        airConductorAllow = params["airconductor"];        Serial.println("Set air");      }      if(params.containsKey("fan"))      {        fanAllow = params["fan"];        Serial.println("Set fan");      }      serializeJsonPretty(setAlinkMsgObj, Serial);    // 打印json文档内容      Serial.println();      String str = setAlinkMsgObj["id"];    // 获取消息id      char SendBuf[100];    // 响应消息缓冲区      sprintf(SendBuf, "{\"id\":\"%s\",\"code\":200,\"msg\":\"success\"}", str.c\_str());    // 构造响应消息      Serial.println(SendBuf);      delay(100);      if(client.publish(ONENET\_TOPIC\_PROP\_SET\_REPLY, SendBuf))    // 发布响应消息      {        Serial.println("Send set reply success!");      }      else      {        Serial.println("Send set reply failed~");      }    }  } |

接着便是配置WiFi，创建PubSubClient对象，连接至OneNET平台，订阅相关主题，设置回调函数，绑定Ticker和属性上报函数，每1秒上报1次设备属性即传感器数据，并在loop函数中调用client.loop循环运行MQTT客户端。没什么好讲的详见源代码。

1. 中心节点(服务器)

中心节点即作为MQTT客户端，将LoRa接收到的门禁信息转发到OneNET，又作为Web服务器，部署前端页面。前端采用Vue3+element开发，并通过npm打包挂载到文件系统，并将文件系统制作映像烧录至Flash中，因此对Flash的大小有一定要求，所以采用的是ESP32-S3 R16N8具有16M的Flash。且为了方便我所用的开发环境是VSCode+Platform，值得一提的是由于CDN污染且没有国内镜像源，配置环境过程很漫长。

关于MQTT的部分和office\_1类似，只是改一下device\_id和token，且上传的数据也只有门禁信息，其他大差不差。关于接收LoRa模块的数据，采用的是LoRa模块的透传模式，即LoRa模块会将接收到的数据直接通过AT串口传输，改串口的工作波特率为9600，引脚分别是AT\_TX、AT\_RX。

|  |
| --- |
| void Lora\_Receive(void)  {      if (customSerial.available() >= 5)      {          int index = 0;          char incomingByte = customSerial.read(); // 读取一个字节          if (incomingByte == 0xFF)    // 检测到起始字节          {              customSerial.println("Start byte detected.");              while (customSerial.available())              {                  customSerial.println("have data");                  buffer = customSerial.readStringUntil(0x0A);  // 结束字节              }              buffer += "\0";              accessFlag = true;              Serial.println(buffer);          }      }  } |

关于前端页面的，我采用了饿了么开源的Vue3组件Element-plus，详情页面：<https://element-plus.org/zh-CN/component/overview.html>，并运用了其中的Container布局容器、Layout布局、Card组件、Avatar头像、Collapse折叠面板、Timeline时间线、Scrollbar滚动条等等构建前端页面整体并刻画细节。此外为了显示实时数据动态曲线，使用ECharts组件构建温湿度二氧化碳实时曲线，源代码较为复杂不进行详细分析，考虑假期有时间整理思路并出一期视频。



*前端页面*

1. 后台数据处理

编写Python脚本，向OneNET的接口发送HTTP GET请求获取系统所有设备属性的历史记录，并将获得的JSON数据解析，通过时间戳计算设备总的运行时间，给各个设备确定一个模拟功率，计算系统总的能耗，与不使用本系统相比(各用电设备一直运行)，得出节约的电能，进而得到本系统减少碳排放量。

**实验结果**

实验结果见视频。

**实验总结**

通过本次实验，复习了过去所学知识并将之应用于实际，加深了关于单片机、传感器、计算机网络相关知识的理解，还借此机会学习了前端页面设计相关知识。整个系统工作稳定符合预期，实验过程较为顺利，总体感觉良好。