# 六维度超声波气象站

## 项目背景

气象信息如气温、湿度、紫外线等级、PM2.5等信息与人们的生活息息相关。虽然气象类APP为我们提供了在日常生活中足够的气象信息，但对于客制化的用途（如定点环境监测），气象类APP所提供的信息就无法很好的满足。而成品气象站价格普遍在千元级别，对于小型项目价格昂贵且客制化程度较低，故需要自行搭建气象站。

传统气象站对风速、风向的测量都采用机械旋转式的风速仪、风向仪。机械式风向、风速仪在测量时存在启动速度，当风速过小时，机械风向、风速仪便无法进行测量。且机械风向、风速仪存在体积庞大、机械损耗的问题。

而超声波气象站可以很好的解决这一问题，超声波气象站是利用超声波模组对声速进行测量的一种气象站。由于风速会引起声速的变化：超声波顺风传播时，声速将加快；逆风时风速将减慢；通过测量一组正交的声速矢量，进行合成分析后可以分析出当前的风速、风向信息。并且超声波传感器体积较小，通常在2cm2左右，可以大幅度降低气象站的整体体积。

使用涂鸦平台的MCU低代码模式进行超声波气象站开发除了能够降低项目成本与高度客制化之外，基于涂鸦的模板还能够实现快速物联网开发与设备面板设计。

## 气象站简介

超声波气象站的测量维度包括：

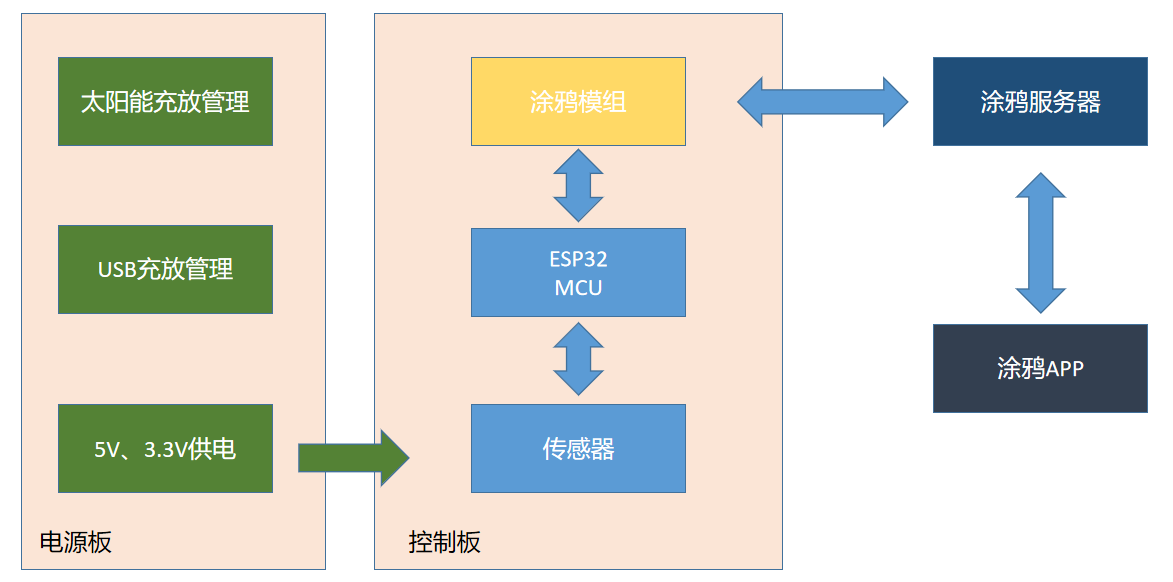
* 温度测量（DHT22）
* 湿度测量（DHT22）
* 气压测量（BMP180）
* 风向、风速测量（防水超声波测距模组）
* 紫外线等级测量（CJMCU-GUVA-S12SD）
* PM2.5测量（GP2Y1014AU）

即六个维度的气象信息的测量。

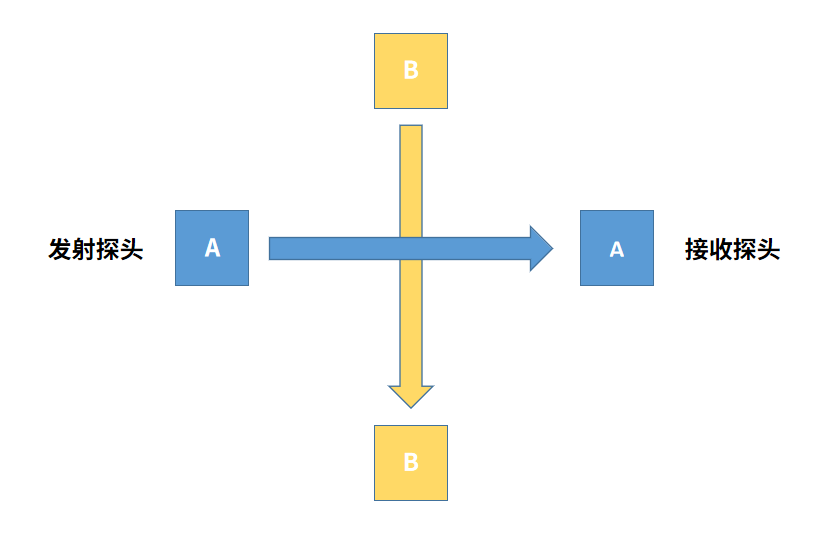
气象站采用ESP32作为主控，使用Arduino平台进行开发，开发IDE为Visual Studio Code 配合Platform IO。

物联网通讯方面使用涂鸦WBR3模组，使用低代码MCU开发形式。控制界面(APP)使用涂鸦的客制化面板。

## 系统框架



## 超声波风速测量原理



超声波探头布置如图所示，两组探头互相垂直分布。

* 此时若A组探头间距离S已知，声速C已知，且此时风速为0。则此时超声波的行进速度T为声速，可算出无风时超声波从发射探头移动到接收探头的时间T1=S/C。
* 若此时存在风向为发射探头指向接收探头，风速为1m/s的风，则超声波行进速度变为C+1m/s，同样可以算出该情况下超声波从发射探头移动到接收探头的时间T2\*\*=S/(C+1)。

在距离已知、声速已知的情况下，通过获取超声波模块返回的超声波行进时间T2，可以推算出A组探头方向的风速V1=(S / T2) - (S / T1)。同理可推算出B组探头方向的风速V2。

由于A、B组探头垂直分布，故当存在任意方向、任意大小的风时，可通过矢量合成的方法计算出风速和风向。

## 模块驱动

**DHT22**

DHT22使用[DHT-Sensor-Library](https://github.com/adafruit/DHT-sensor-library)进行驱动，详细方法参考GitHub页面，使用该库需要依赖[Adafruit Unified Sensor Driver](https://github.com/adafruit/Adafruit\_Sensor)库。

```c++

DHT dht(DHT\_PIN, DHT\_TYPE);

float h = dht.readHumidity();

float t = dht.readTemperature();

if(isnan(h) || isnan(t)){

Serial.println("Failed to read from DHT sensor.");

return;

}

```

此处h获取值为湿度，t获取值为温度。

**BMP180**

BMP180使用[Adafruit BMP085 Library](https://github.com/adafruit/Adafruit-BMP085-Library)驱动，同上，需要依赖[Adafruit BrushIO](https://github.com/adafruit/Adafruit\_BusIO)库。

```c++

Adafruit\_BMP085 bmp;

air\_pressure = bmp.readPressure()

```

**超声波模组**

```c++

// Clears the trigPin condition

digitalWrite(TRIG\_1, LOW);

delayMicroseconds(2);

// Sets the trigPin HIGH (ACTIVE) for 10 microseconds

digitalWrite(TRIG\_1, HIGH);

delayMicroseconds(20);

digitalWrite(TRIG\_1, LOW);

// Reads the echoPin, returns the sound wave travel time in microseconds

duration = pulseIn(ECHO\_1, HIGH);

```

超声波模组使用脉冲测量获取反射时间，代码如上。后续通过布置探头位置，分析两组超声波模组的反射时间可获得风速、风向信息。

**CJMCU-GUVA-S12SD紫外线模组**

该紫外线模组输出为电压模拟量，`analogRead()`函数可以实现紫外线等级的测量。

**GP2Y1014AU PM2.5模组**

```C++

digitalWrite(DUST\_IN, LOW);

delayMicroseconds(SAMPLING\_TIME);

int voMeasured = analogRead(DUST\_OUT);

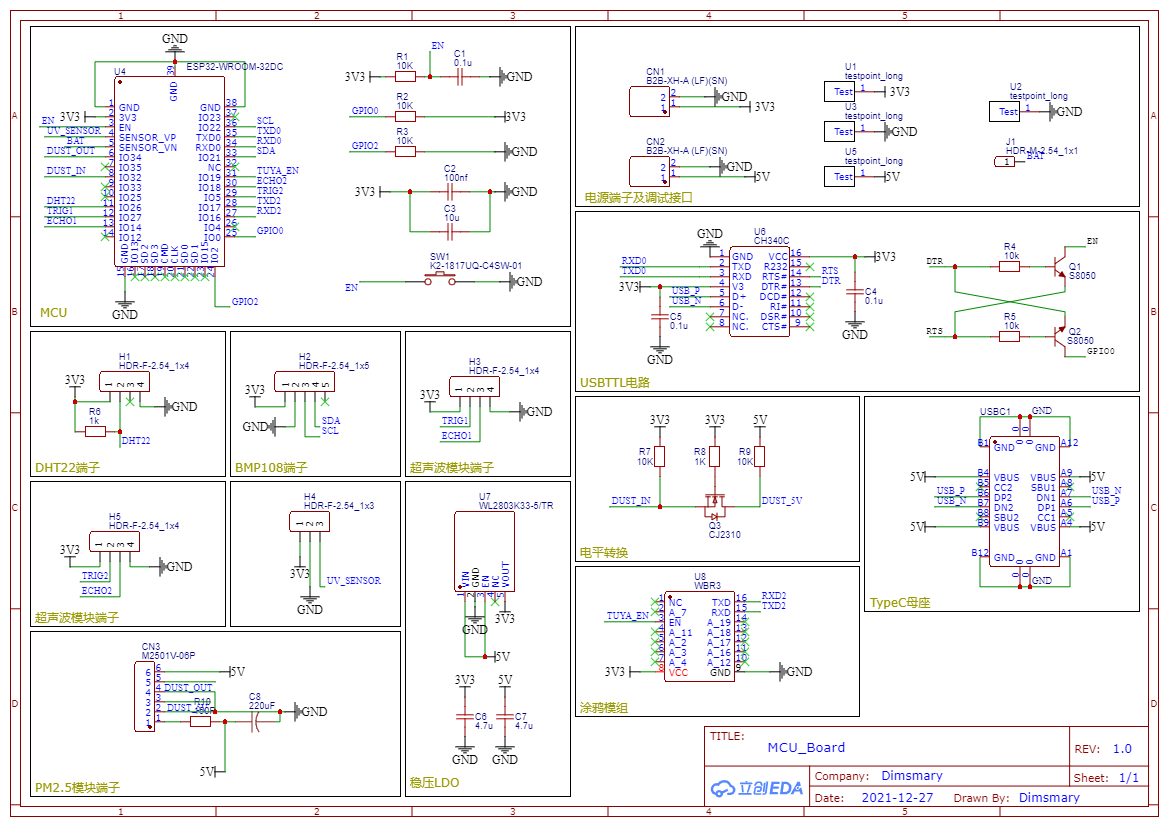
delayMicroseconds(DELTA\_TIME);

digitalWrite(DUST\_IN, HIGH);

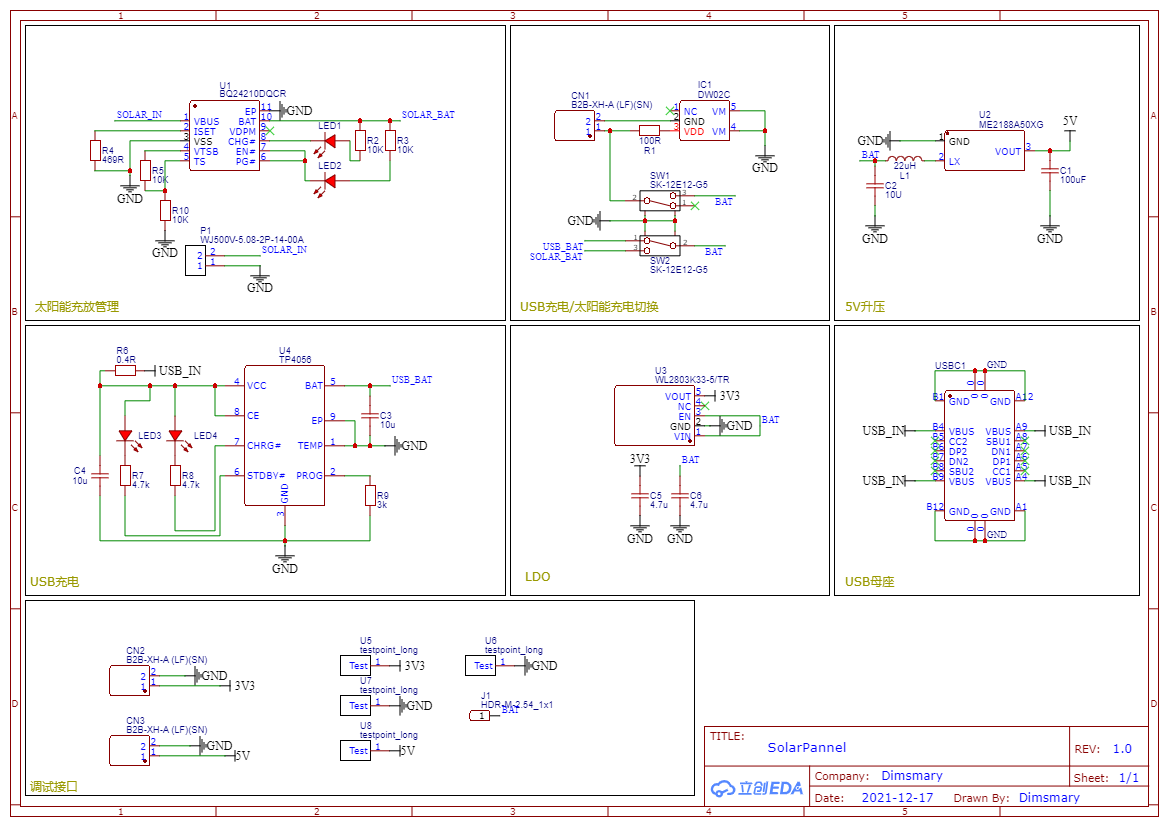
```

通过给模组的LED驱动脚施加脉冲，测量输出脚电压，可以实现对pm2.5的浓度测量。

**原理图：**

****

## 太阳能充放电设计



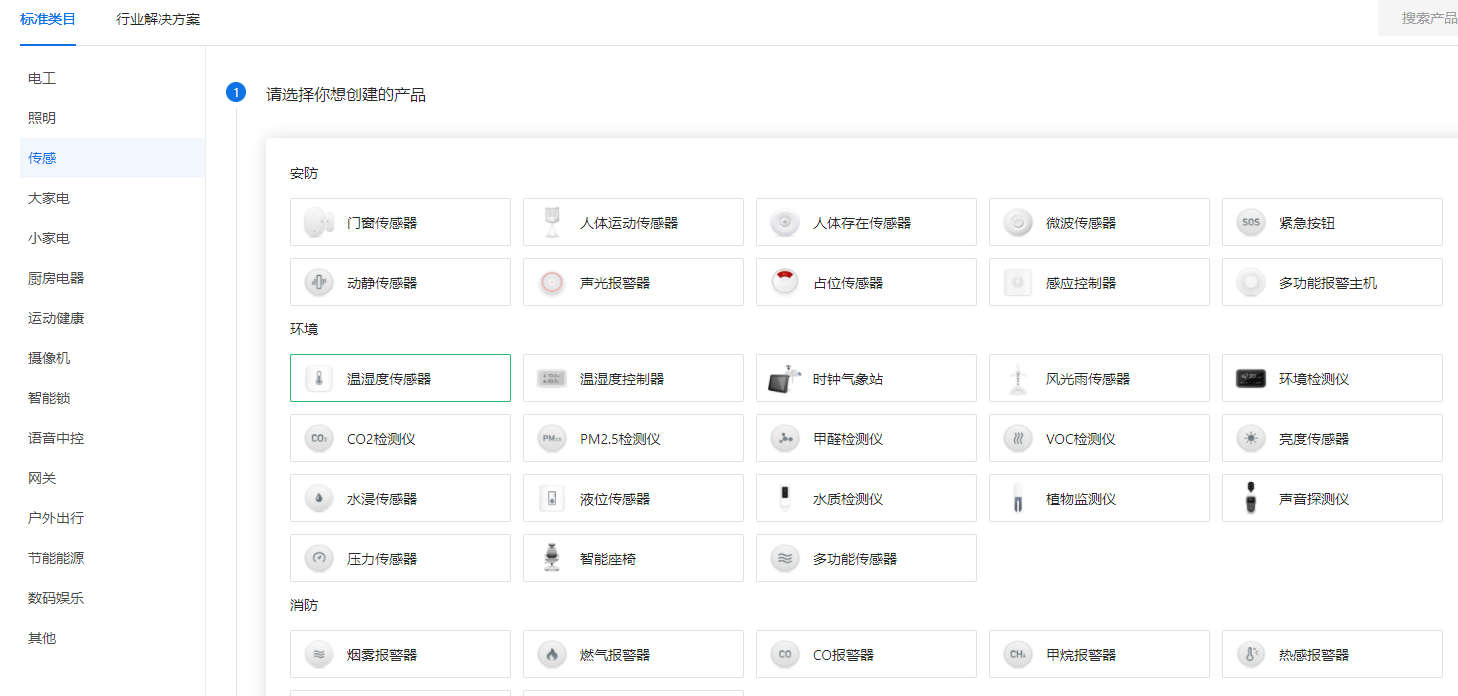
总体充放电系统原理图如图，太阳能充放电采用BQ24210DQCR作为控制。

## 涂鸦智能接入

这次的开发方式为MCU低代码开发，即是利用涂鸦模组进行开发。涂鸦模组负责物联网信息处理，MCU仅需要通过串口控制涂鸦模组进行物联网信息的通讯。

**创建产品**

在进行涂鸦智能的开发时，需要进行产品创建。作为气象站开发，可以选择温湿度传感器。



**DP点创建**

| DP ID | 功能点名称 | 标识符 | 数据传输类型 | 数据类型 | 功能点属性 |
| --- | --- | --- | --- | --- | --- |
| 1 | 温度 | temp\_current | 只上报（ro） | 数值型（Value） | 数值范围: -200-600, 间距: 1, 倍数: 1, 单位: ℃ |
| 2 | 湿度 | humidity\_value | 只上报（ro） | 数值型（Value） | 数值范围: 0-100, 间距: 1, 倍数: 0, 单位: % |
| 4 | 电池电量 | battery\_percentage | 只上报（ro） | 数值型（Value） | 数值范围: 0-100, 间距: 1, 倍数: 0, 单位: % |

除了这些标准功能中的DP点之外，因为我们还需要风速、风向这些数据，所以还需要添加自定义的DP点：

| DP ID | 功能点名称 | 标识符 | 数据传输类型 | 数据类型 | 功能点属性 |
| --- | --- | --- | --- | --- | --- |
| 101 | PM2.5浓度 | pm25 | 只上报（ro） | 数值型（Value） | 数值范围: 0-100, 间距: 1, 倍数: 0, 单位: |
| 102 | 风向角 | wind\_value | 只上报（ro） | 数值型（Value） | 数值范围: 0-364, 间距: 1, 倍数: 1, 单位: |
| 103 | 风向 | wind\_direction | 只上报（ro） | 字符型（String） |  |
| 104 | 风速 | wind\_speed | 只上报（ro） | 数值型（Value） | 数值范围: 0-100, 间距: 1, 倍数: 1, 单位: |
| 105 | 紫外线等级 | uv\_level | 只上报（ro） | 数值型（Value） | 数值范围: 0-10, 间距: 1, 倍数: 0, 单位: |
| 106 | 气压 | pressure | 只上报（ro） | 数值型（Value） | 数值范围: 0-1999, 间距: 1, 倍数: 0, 单位: hPa |

在功能点中，DP ID为该功能点的标识，MCU在控制涂鸦模组进行数据上传时需要用到。

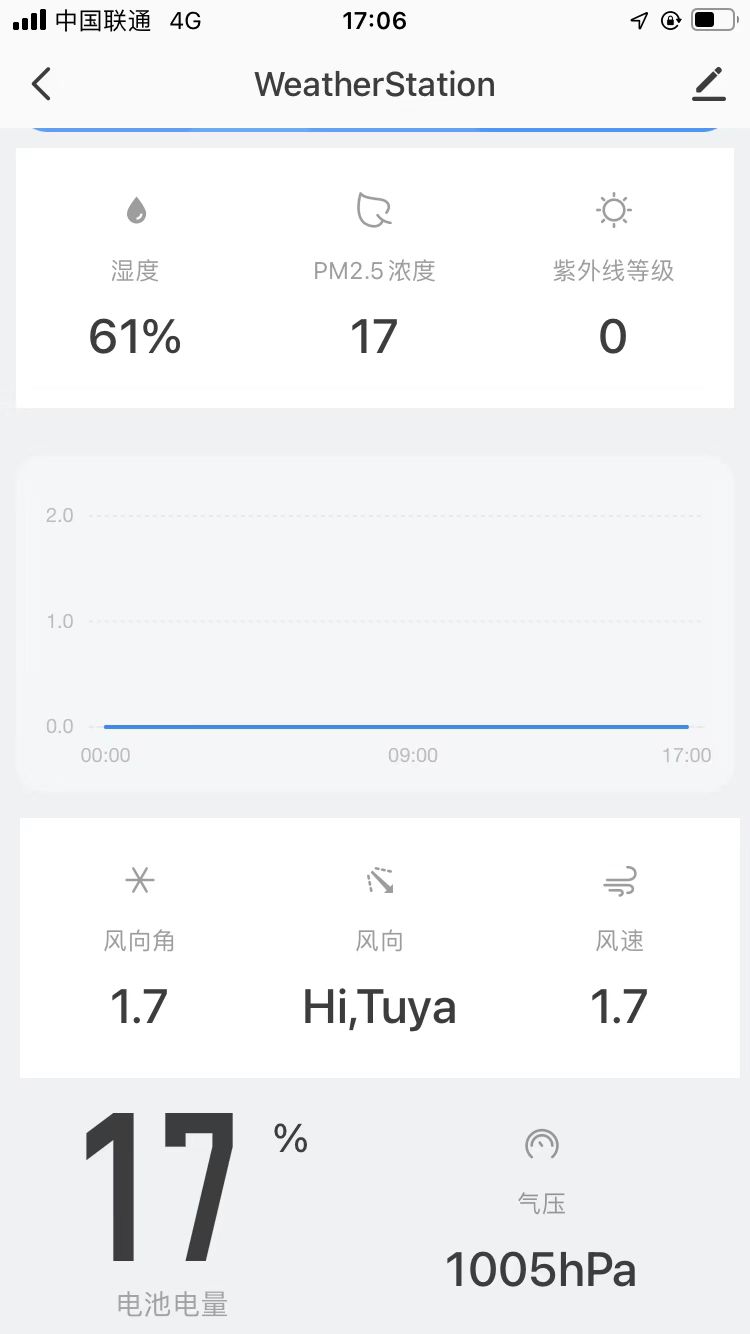
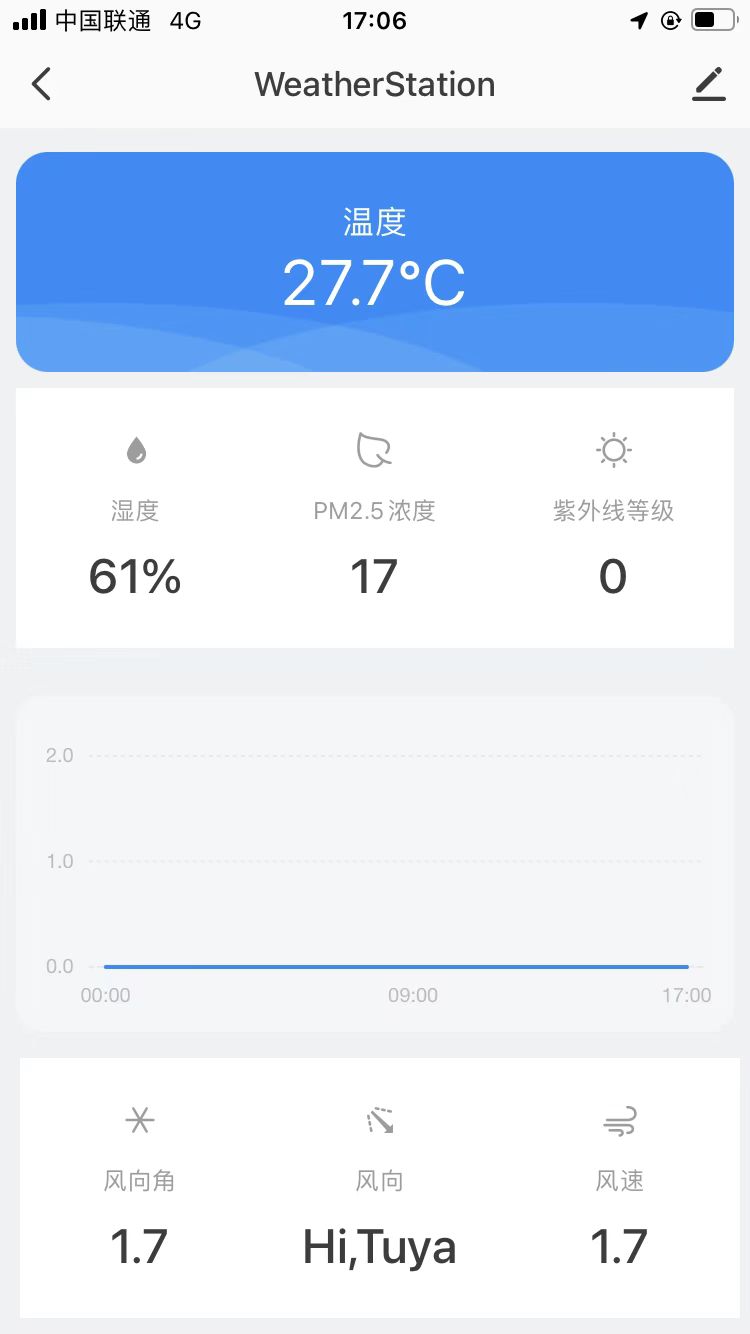
功能点属性中的倍数为0时，MCU上传的数据被识别为上传的数据本身。若倍数为1，则此时若上传数据为50时，该数据将被涂鸦除以10^1，即50会被识别为5。

**面板编辑**

在产品开发的面板选项卡，可以对自己的APP面板进行自定义。

我使用的面板类型为:面板类型：Studio 面板-个人面板

下面是在手机中的实机效果：



**硬件开发**

在完成了传感器驱动、电源板管理后，需要与涂鸦模组进行通讯，实现信息上报。

在这里我选择的是WBR3 Wi-Fi & Bluetooth 模组。

该模组在初次上电时，可以打开智能生活APP实现配网和面板加载，在完成了配网之后就可以进行实时开发了。

**Arduino SDK**

虽然涂鸦提供了Arduino的SDK，但是在实际操作中发现该SDK仅能在Arduino UNO(ATMEGA 328P)中编译通过，在其他型号的单片机编译中都出现了报错，ESP32也不例外。

通过查阅SDK的源代码，发现在ESP32下的编译主要是Software Serial库的兼容性问题，通过简单修改后可以完成编译，实现基础功能，但格林时间以及实时时钟功能未能实现。

魔改SDK地址：https://github.com/Dimsmary/WeatherStation/tree/main/OutDoor/lib/tuya-wifi-mcu-sdk-arduino-library-master

下面是SDK的实际使用代码：

// define dp

#define temp\_pid 1

#define humi\_pid 2

#define battery\_pid 4

#define pm25\_pid 101

#define wind\_angle\_pid 102

#define wind\_direct\_pid 103

#define wind\_speed\_pid 104

#define uv\_level\_pid 105

#define pressure\_pid 106

// Create TuyaWifi object

TuyaWifi my\_device;

// input your device pid here

unsigned char pid[] = {""};

// select mcu version, 1.0.0 as commonly use

unsigned char mcu\_ver[] = {"1.0.0"};

// create an array to restore the datapoint

unsigned char dp\_array[][2] =

{

{temp\_pid, DP\_TYPE\_VALUE},

{humi\_pid, DP\_TYPE\_VALUE},

{battery\_pid, DP\_TYPE\_VALUE},

{pm25\_pid, DP\_TYPE\_VALUE},

{wind\_angle\_pid, DP\_TYPE\_VALUE},

{wind\_direct\_pid, DP\_TYPE\_STRING},

{wind\_speed\_pid, DP\_TYPE\_VALUE},

{uv\_level\_pid, DP\_TYPE\_VALUE},

{pressure\_pid, DP\_TYPE\_VALUE},

};

unsigned char dp\_process(unsigned char dpid,const unsigned char value[], unsigned short length)

{

/\* all DP only report \*/

return TY\_SUCCESS;

}

void dp\_update\_all(void)

{

}

void setup(){

...

my\_device.init(pid, mcu\_ver);

my\_device.set\_dp\_cmd\_total(dp\_array, 9);

// register call back function

my\_device.dp\_process\_func\_register(dp\_process);

my\_device.dp\_update\_all\_func\_register(dp\_update\_all);

}

void loop() {

// SDK handle

my\_device.uart\_service();

// Check the wifi connection

if ((my\_device.mcu\_get\_wifi\_work\_state() == WIFI\_CONNECTED) || (my\_device.mcu\_get\_wifi\_work\_state() == WIFI\_CONN\_CLOUD)) {

update\_sensor();

// report all the data

my\_device.mcu\_dp\_update(temp\_pid, temp, 1);

my\_device.mcu\_dp\_update(humi\_pid, humi, 1);

my\_device.mcu\_dp\_update(battery\_pid, battery\_level, 1);

my\_device.mcu\_dp\_update(pm25\_pid, pm25, 1);

my\_device.mcu\_dp\_update(wind\_angle\_pid, wind\_angle, 1);

my\_device.mcu\_dp\_update(wind\_speed\_pid, wind\_speed, (sizeof(dp\_string\_value) / sizeof(dp\_string\_value[0])));

my\_device.mcu\_dp\_update(wind\_speed\_pid, wind\_speed, 1);

my\_device.mcu\_dp\_update(uv\_level\_pid, uv\_index, 1);

my\_device.mcu\_dp\_update(pressure\_pid, air\_pressure, 1);

}

...

## MCU工作流程

