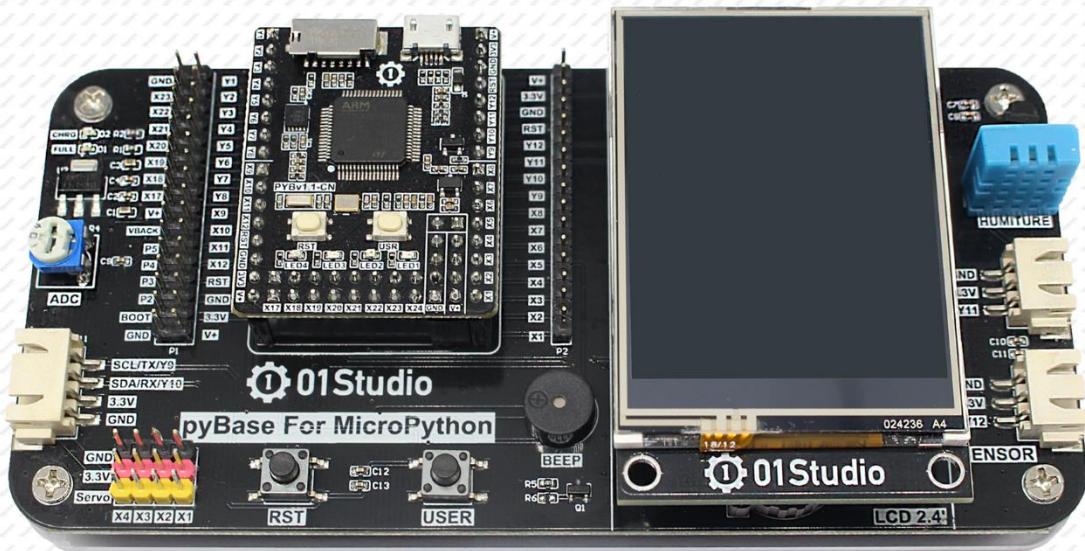


# MicroPython 从0到1

用python做嵌入式编程

(基于pyBoard STM32F405平台)

01Studio团队 编著

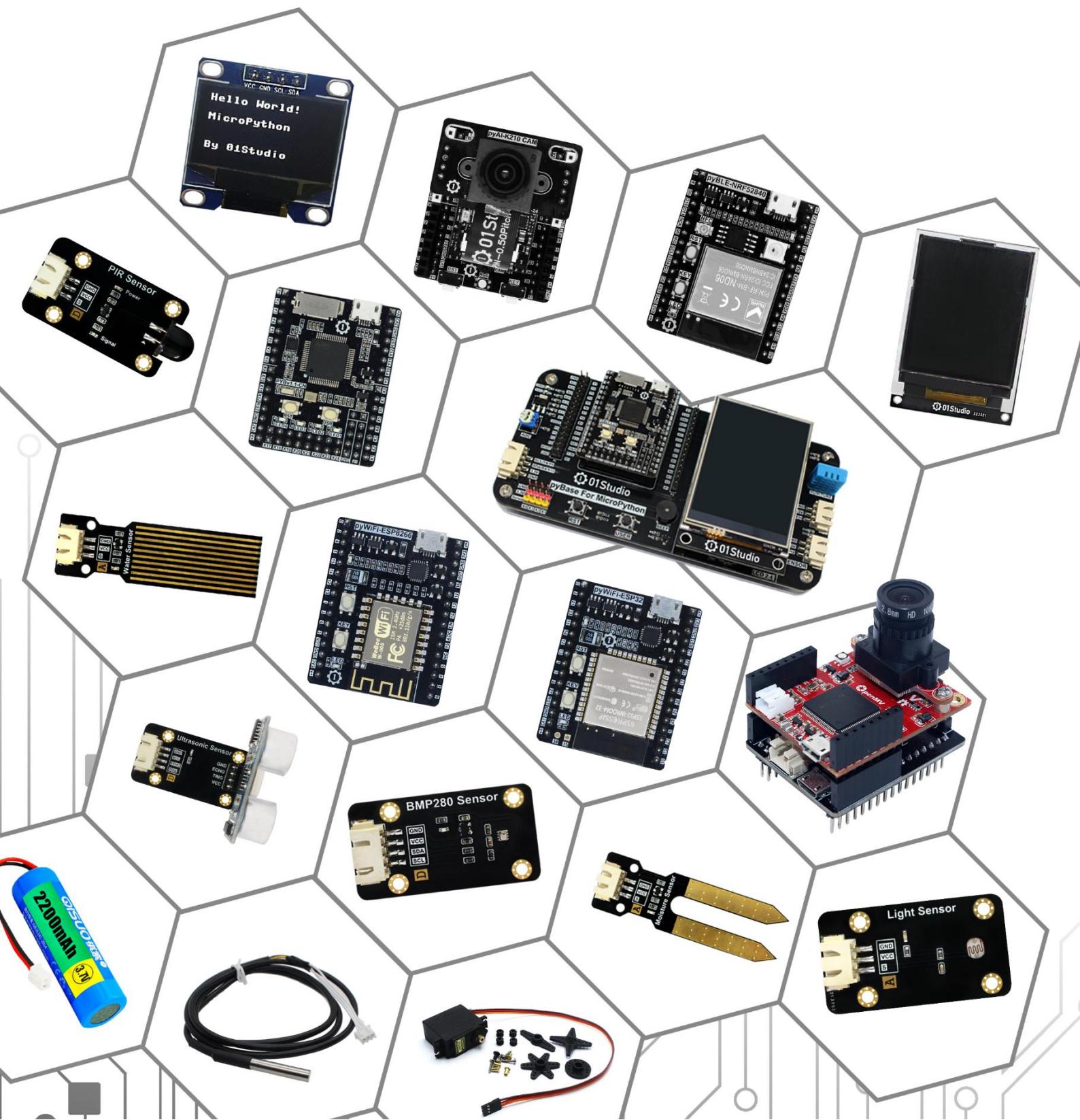


 01Studio

-让编程变得简单有趣-

# MicroPython

## 产品家族



# 前言

## 为什么学习 MicroPython?

单片机嵌入式编程经历了汇编、C 语言的发展历程，可以说是一次编程革命，其背后的原因是单片机的速度越来越快，集成度越来越高。而这一趋势并没停止，摩尔定律仍然适用。在未来，单片机上很可能直接跑机器语言。

在 2014 年，MicroPython 在英国诞生了，对于电子爱好者来说无疑拉开了新时代的序幕，用 python 这个每年用户量不断增长的编程语言来开发嵌入式，加上无数开源的函数模块，让嵌入式开发变得从未如此的简单。

MicroPython 致力于兼容 Python。因此，我们在学习完 MicroPython 后除了可以开发有趣的电子产品外，还可以继续深入使用 Python 语言去开发后台、人工智能等领域。

## 为什么要写《MicroPython 从 0 到 1》教程？

对于初学者，常常需要浪费大量时间来搭建开发环境和安装应用软件，以及需要从不同渠道寻找开发资料。MicroPython 作为新兴的东西，市面上的资料更是参差不齐，被搞得头昏目眩。

“让编程变得简单有趣”作为我们 01Studio 团队的使命，我们一直在寻找最优的学习方法。力求以最简单易懂的方式来讲述 MicroPython 的学习方法，从开发环境快速建立、基础实验、传感器实验到项目进阶实验。《MicroPython 从 0 到 1》诞生了。相比于传统的出版图书，我们会以更平易近人的口吻讲解实验，配上精美的插图，每一行代码都是自己的亲身经历，目的就是让大家更好的入门 MicroPython，将精力放在编程和开发中去。

## 为什么要打造 MicroPython 学习套件？

MicroPython 在中国是一个新兴的东西，前途无限，但是网上的学习模块套件参差不齐，大多是复制官方开源的开发板来设计，用过的就知道，外国的电路设计跟国内的风格很不同，甚至常常让初学者钻牛角尖。为此，01Studio 团队特意打造的中国风的 MicroPython 开发套件，《MicroPython 从 0 到 1》上的例程也是基于此板开发的，每个例程都能直接跑起。通过一系列系统学习，

你甚至可以用它来完成你的比赛或项目。

### 为什么要打造 01Studio 社区论坛？

早在数年前我们打造了 WeBee 品牌，专注物联网开发，到现在已经服务了数万名学生、教师和工程师等相关人员。早期依靠着群来维护，但随着开发者数量增加，我们发现仅依靠群或者技术支持人员已经不能满足需求。

社区论坛是很好的学习交流地方，在这里你可以学习到前人的经验，可以通过搜索内容来解决问题。为什么全世界很火的开源硬件都是由外国人做起来，我们认为很重要的一个点是源于开源和分享精神。大部分开发者都会想着从论坛或网站解决自身问题而不愿意奉献，而我们希望 01Studio 社区是一个大家乐于发表文章和分享心得的地方。我们始终始终坚持开源原则，包括书籍内容、所有例程代码和部分硬件模块的开源。

期待你加入 01Studio 社区大家庭，成为我们的一份子，一起成长。

【社区链接: [bbs.01studio.cc](http://bbs.01studio.cc)】

【微信公众号: 01Studio 社区】

【官方商城:[01studio.taobao.com](http://01studio.taobao.com)】

01Studio

2019.4 于深圳

## 版本说明

版本	日期	作者	更新内容
v1.0	2021-2-5	Jackey	1. 第1版（重构）
v1.1	2021-9-1	Jackey	1. 默认IDE从Mu换成Thonny; 2. 增加继电器实验 3. 更换MQTT在线服务器助手。
v1.2	2021-9-29	Jackey	1. 增加3.2寸（电阻触摸）显示和触摸实验。

## 版权声明

《MicroPython 从 0 到 1》由 **01Studio** 团队打造，已于深圳市版权局注册备案，任何单位或个人引用相关文字、图片或相关内容请注明出处【**01Studio**】，否则我们将保留追究相关法律责任的权利。

## 目录

第 1 章 MicroPython 简介 .....	7
1.1 MicroPython 是什么 .....	7
1.2 MicroPython 支持的微控制器平台 .....	8
1.3 MicroPython 相关学习资料 .....	10
1.3.1 01Studio 技术论坛 .....	10
1.3.2 MicroPython 库文档 .....	11
1.3.3 MicroPython 官方网站 .....	12
1.4 MicroPython 开发套件介绍 .....	13
1.4.1 STM32 平台 .....	14
1.4.2 pyBase .....	21
1.4.3 IOT/通讯模块 .....	23
1.4.4 传感器模块 .....	26
1.4.5 拓展配件 .....	33
第 2 章 Python 基础知识 .....	41
2.1 原始数据类型和运算符 .....	41
2.2 变量和集合 .....	46
2.3 流程控制和迭代器 .....	52
2.4 函数 .....	56
2.5 类 .....	59
2.6 模块 .....	61
2.7 高级用法 .....	62
第 3 章 开发环境快速建立 .....	64
3.1 基于 Windows .....	65
3.1.1 安装开发软件 Thonny IDE .....	65
3.1.2 开发套件使用 .....	67
3.1.3 附录 .....	91
3.2 基于 Mac OS .....	100
3.2.1 安装开发软件 Thonny .....	100
3.2.2 开发套件使用 .....	100
3.3 基于 Linux .....	101
3.3.1 安装开发软件 Thonny .....	101
3.3.2 开发套件使用 .....	101
第 4 章 基础实验 .....	102
4.1 点亮第一个 LED 灯 .....	103
4.2 流水灯 .....	107
4.3 按键 .....	114
4.4 GPIO .....	119
4.5 外部中断 .....	123
4.6 I2C 总线（OLED 显示屏） .....	127
4.7 RTC 实时时钟 .....	134
4.8 ADC .....	140
4.9 DAC .....	145

4.10 三轴加速度计.....	153
4.11 UART（串口通信）.....	158
4.12 LCD 显示屏 .....	165
4.13 电阻触摸屏.....	176
4.14 触摸屏按钮.....	182
第 5 章 传感器实验.....	189
5.1 温度传感器 DS18B20.....	190
5.2 温湿度传感器 DHT11.....	196
5.3 人体感应传感器.....	201
5.4 光敏传感器.....	207
5.5 土壤湿度传感器.....	212
5.6 水位传感器.....	218
5.7 大气压强传感器.....	225
5.8 超声波传感器.....	232
第 6 章 拓展实验.....	239
6.1 继电器.....	240
6.2 舵机.....	245
6.3 RGB 灯带.....	255
6.4 以太网模块.....	261
6.4.1 连接网络.....	264
6.4.2 Socket 通信.....	269
6.4.3 MQTT 通信 .....	281

# 第1章 MicroPython 简介

## 1.1 MicroPython 是什么

第一次接触 MicroPython 的时候，我就想这是个什么玩意，从字面意思来看，就是 Micro 加 Python。难道是阉割版的 Python？阉割后可以在微控制器上面跑？当然你也可以这么理解，我们来看看官方的说明：

“MicroPython 是 Python 3 编程语言的精简高效实现，包括 Python 标准库的一小部分，并且经过优化，可以在 Microcontrollers（微控制器）和有限的环境中运行。

MicroPython 包含许多高级功能，如交互式提示，任意精度整数，闭包，列表理解，生成器，异常处理等。然而它非常紧凑，可以在 256k 的代码空间和 16k 的 RAM 内运行。

MicroPython 旨在尽可能与普通 Python 兼容，以便您轻松地将代码从电脑传输到微控制器或者嵌入式系统。”

看完官方说明后，大家应该有所了解，Micropython 是指在微控制器上使用 Python 语言进行编程，学习过单片机和嵌入式开发的小伙伴应该都知道早期的单片机使用汇编语言来编程，随着微处理器的发展，后来逐步被 C 所取代，现在的微处理器集成度越来越高了，那么我们现在可以使用 Python 语言来开发了。

Python 的强大之处是封装了大量的库，开发者直接调用库函数则可以高效地完成大量复杂的开发工作。MicroPython 保留了这一特性，常用功能都封装到库中了，以及一些常用的传感器和组件都编写了专门的驱动，通过调用相关函数，就可以直接控制 LED、按键、伺服电机、PWM、AD/DA、UART、SPI、IIC 以及 DS18B20 温度传感器等等。以往需要花费数天编写才能实现的硬件功能代码，现在基于 MicroPython 开发只要十几分钟甚至几行代码就可以解决。真可谓：

“人生苦短，我用 Python 和 MicroPython”。

## 1.2 MicroPython 支持的微控制器平台

MicroPython 到目前为止已经可以在多种嵌入式硬件平台上运行：STM32、ESP8266、ESP32、CC3200、K210 等等。由于项目的开源特性，很多开发者在尝试将其移植到更多平台上。

MicroPython 最早支持的硬件平台是 STM32，开发板名称叫 pyboard。使用的芯片型号是：STM32F405RGT6，该芯片具备 1MB flash 和 196k SRAM，168MHZ 主频。也是目前学习资料最全的平台，本书主要是围绕 STM32 平台来进行编写。

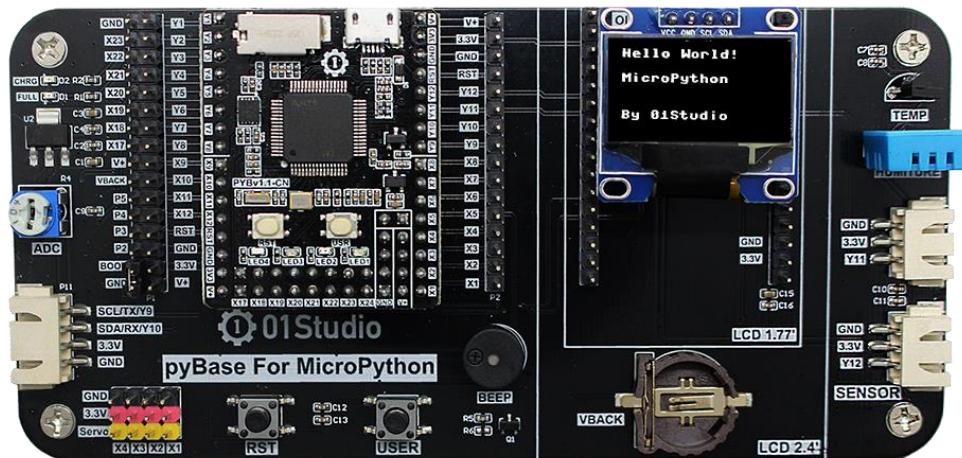


图 1-1 pyBoard 开发套件

除此之外，上海乐鑫的 WIFI 芯片 ESP8266/ESP32 也非常成熟。用户使用 MicroPython 可以快速开发物联网相关应用，实现 WIFI 无线连接。

除此之外，不少优秀的开源项目也是基于 MicroPython 衍生出来的，如机器视觉界 Arduino 之称的 OpenMV、人工智能芯片 K210 等。随着社区的日益成熟，MicroPython 必定将嵌入式编程推向新的高度。

以下是 O1Studio 其它 micropython 开发平台：

<b>STM32 平台</b>	<b>ESP8266 平台</b>	<b>ESP32 平台</b>	<b>NFR52840 平台</b>	<b>OpenMV4 平台</b>	<b>K210 平台</b>
pyBoard v1.1-CN	pyWiFi-ESP8266	pyWIFI-ESP32	pyBLE-NRF52840	pyAI-OpenMV4	pyAI-K210
					
					

图 1-2 01Studio MicroPython 系列开发平台

## 1.3 MicroPython 相关学习资料

### 1.3.1 01Studio 技术论坛

【论坛网址：[bbs.01studio.cc](http://bbs.01studio.cc)】

01Studio 社区是 MicroPython 开发者交流的社区论坛，我们以极简风格设计，开发者在学习过程中遇到问题可以到论坛搜索或者发帖提问，以提高学习效率。01Studio 团队也会在社区定期发布学习资源。

The screenshot shows the homepage of the 01Studio forum. On the left, there's a sidebar with categories like 'MicroPython开发板' (11 posts) and 'Linux Python开发板' (3 posts). The main content area has a search bar at the top. Below it, a post by user 'oxxxx' from August 26, 2021, with 11 replies, asks about using a counter to count external pulses on a breadboard. A reply by 'Jackey' from August 25, 2021, with 39 replies, discusses using MicroPython ESP32 to collect DHT11 data via Tencent Cloud. The right side features a '推荐内容' (Recommended Content) sidebar with links to various forum topics.

图 1-3 01Studio 技术论坛

### 1.3.2 MicroPython 库文档

限于篇幅，本教程部分实验只介绍关键的函数和模块应用，如果在学习过程中希望深入了解所有 MicroPython 函数和模块，请查阅：

**MicroPython 文档（中文）【网址：[docs.01studio.cc](https://docs.01studio.cc)】**



图 1-4 MicroPython 库文档（中文）

该文档是 01Studio 团队在维护的官方文档中文翻译版，我们力求保持与官方文档保持实时同步，降低开发者的学习门槛。

### 1.3.3 MicroPython 官方网站

【网址：[www.micropython.org](http://www.micropython.org)】

英文版官网有官方文档（DOCS）和英文论坛，适合比较英语比较好的小伙伴。

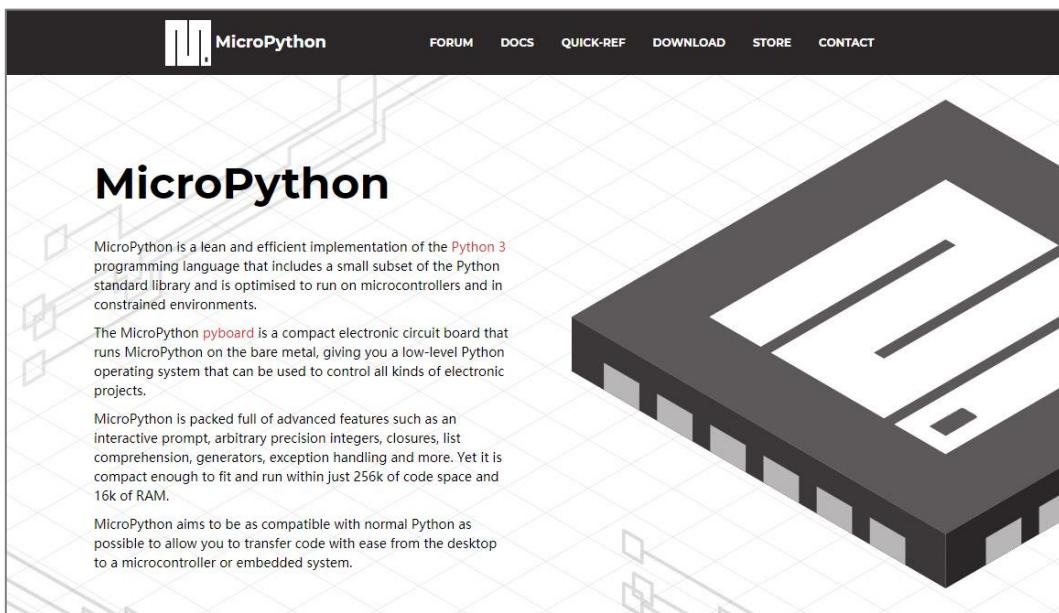


图 1-5 MicroPython 官网

## 1.4 MicroPython 开发套件介绍

为了让广大电子爱好者更方便地学习 MicroPython，01Studio 团队打造了一系列本土化的高性价比学习套件和周边模块。当前已支持 STM32 平台、ESP8266 平台、ESP32 平台、NRF52840 平台、OpenMV 平台、K210 平台以及周边传感器模块和配件外设。我们的开发平台采用核心板+底板形式设计，保留了 MicroPython 官方的兼容性，同时使开发者可以更好的连接外设，进行更多扩展性实验。

《MicroPython 从 0 到 1》上的例程也是基于本学习平台开发的，我们承诺资源会不断更新，保证所有代码程序能直接跑起。毫不夸张地说：你甚至可以将本教材的例程和实践应用在自己的产品研发和项目开发中去。

## 1.4.1 STM32 平台

### 1.4.1.1 pyBoard v1.1-CN

pyBaord v1.1-CN 主控芯片使用 STM32F405RGT6，即基于 STM32 平台，是 01Studio 在兼容官方 pyBoard v1.1 的情况下，对部分功能进行了改进，CN 代表 China 中国版，具体改进如下：

- (1) 按键和 LED 重新排列，让开发者使用更直观；
- (2) 增加锂电池输入接口（XH-2.54 2P 接口），位于板子背部；
- (3) 改进丝印方式，提高了清晰度。

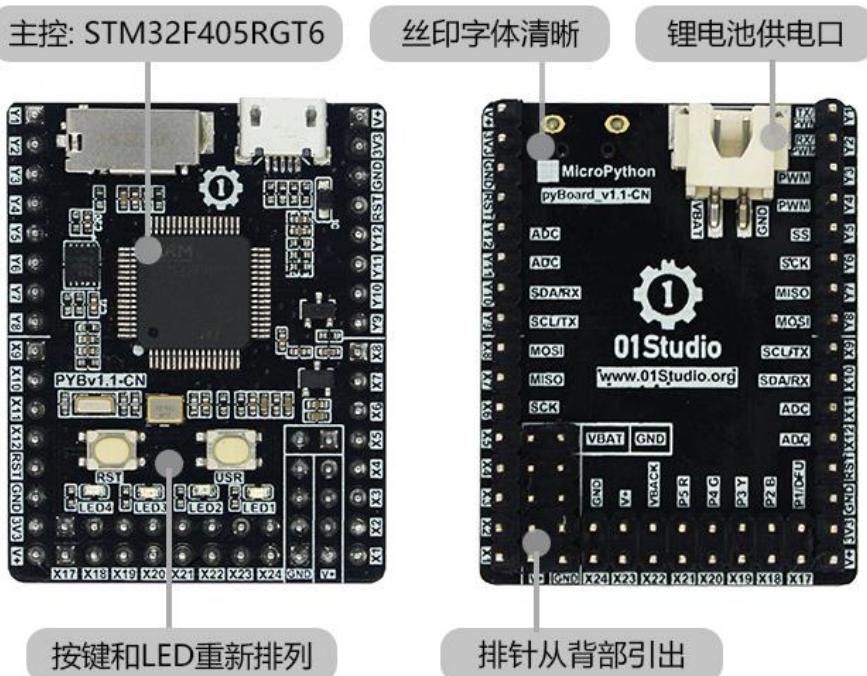


图 1-6 pyBoard v1.1-CN

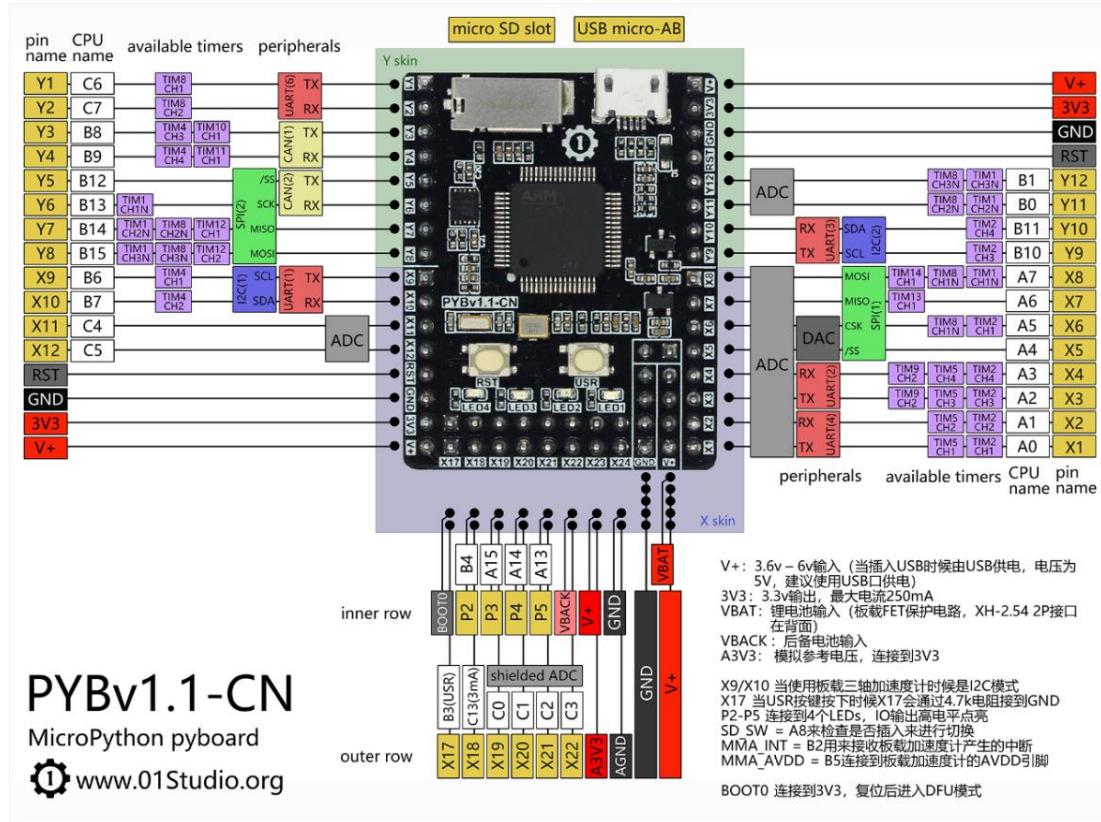


图 1-7 pyBoard v1.1-CN 引脚图

功能参数	
V+	3.6v – 6v 输入 (当插入 USB 时候由 USB 供电, 电压为 5V, 建议使用 USB 口供电)
3V3	3.3v 输出, 最大电流 250mA
VBAT	锂电池输入 (板载 FET 保护电路, XH-2.54 2P 接口在背面)
VBACK	后备电池输入
A3V3	模拟参考电压, 连接到 3V3
X9/X10	当使用板载三轴加速度计时候是 I2C 模式
X17	当 USR 按键按下时候 X17 会通过 4.7k 电阻接到 GND
P2-P5	连接到 4 个 LEDs, IO 输出高电平点亮
SD_SW	A8 来检查是否插入 SD 卡来进行切换
MMA_INT	B2 用来接收板载加速度计产生的中断

MMA_AVDD	B5 连接到板载加速度计的 AVDD 引脚
BOOT0	连接到 3V3，复位后进入 DFU 模式

表 1-1 pyBoard v1.1-CN 参数

#### 1.4.1.2 pyBoard Mini

pyBoard Mini 主控芯片使用 STM32F411CEU6，即基于 STM32 平台，是 pyBoard 的 Mini 版本，精简了部分功能，提高了性价比：

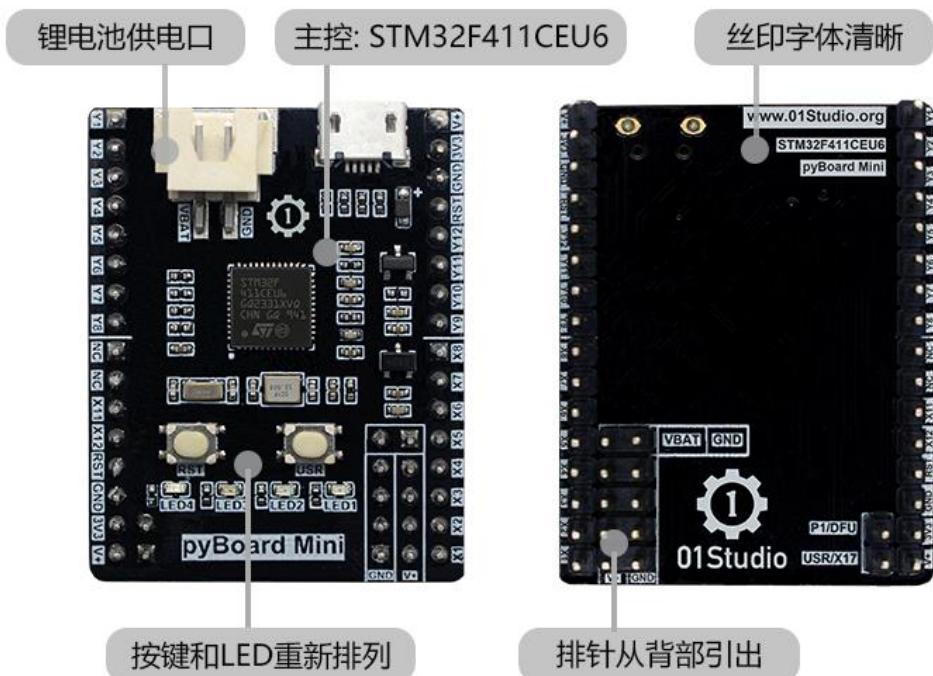


图 1-8 pyBoard Mini

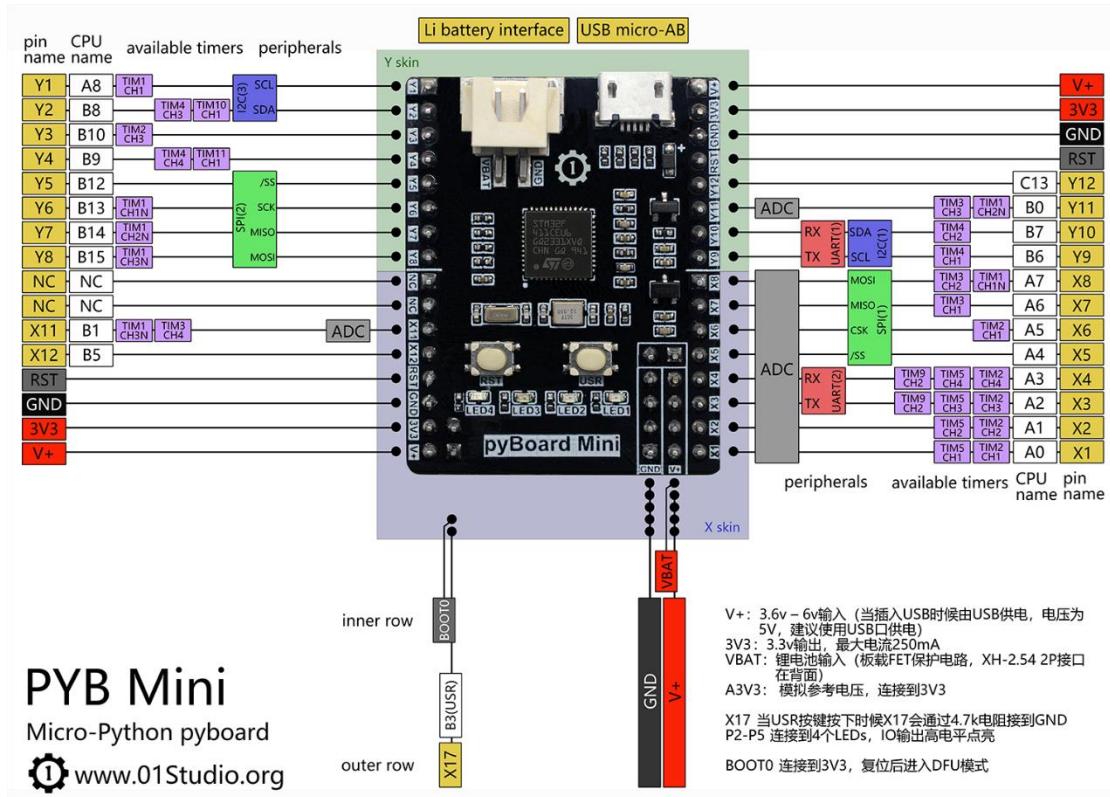


图 1-9 pyBoard Mini 引脚图

功能参数	
V+	3.6v – 6v 输入 (当插入 USB 时候由 USB 供电, 电压为 5V, 建议使用 USB 口供电)
3V3	3.3v 输出, 最大电流 250mA
VBAT	锂电池输入 (板载 FET 保护电路, XH-2.54 2P 接口在背面)
A3V3	模拟参考电压, 连接到 3V3
X17	当 USR 按键按下时候 X17 会通过 4.7k 电阻接到 GND
P2-P5	连接到 4 个 LEDs, IO 输出高电平点亮
BOOT0	连接到 3V3, 复位后进入 DFU 模式

表 1-2 pyBoard Mini 功能参数

### 1.4.1.3 pyBoard Plus

pyBaord Plus 主控芯片使用 STM32F407VGT6，即基于 STM32 平台，是 pyBoard 的 Plus 加强版本，相对于 pyboard 增加了 32 个通用 GPIO 和拓展功能：

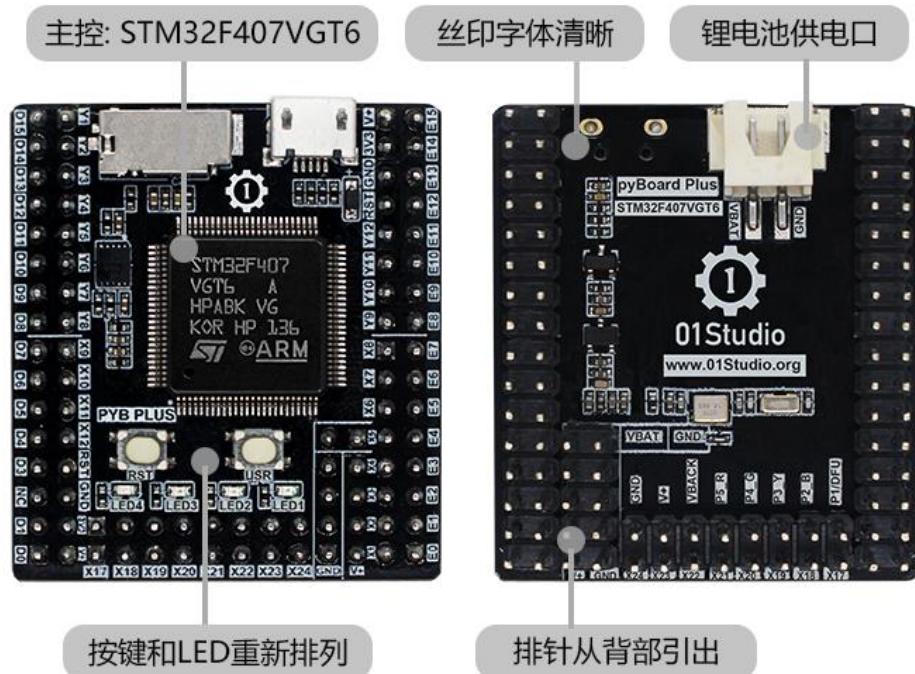


图 1-10 pyBoard Plus

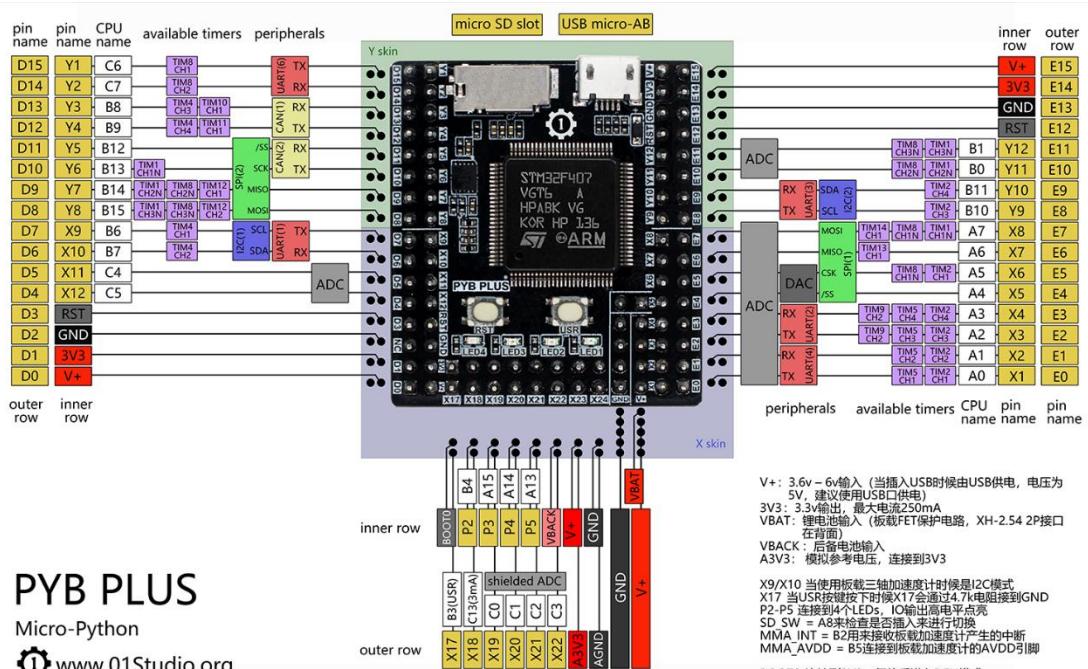


图 1-11 pyBoard Plus 引脚图

功能参数	
V+	3.6v - 6v 输入 (当插入 USB 时候由 USB 供电, 电压为 5V, 建议使用 USB 口供电)
3V3	3.3v 输出, 最大电流 250mA
VBAT	锂电池输入 (板载 FET 保护电路, XH-2.54 2P 接口在背面)
A3V3	模拟参考电压, 连接到 3V3
X9/X10	当使用板载三轴加速度计时候是 I2C 模式
X17	当 USR 按键按下时候 X17 会通过 4.7k 电阻接到 GND
P2-P5	连接到 4 个 LEDs, IO 输出高电平点亮
SD_SW	A8 来检查是否插入 SD 卡来进行切换
MMA_AVDD	B5 连接到板载加速度计的 AVDD 引脚
BOOT0	连接到 3V3, 复位后进入 DFU 模式

表 1-3 pyBoard Plus 功能参数

#### 1.4.1.4 pyBoard 开发套件

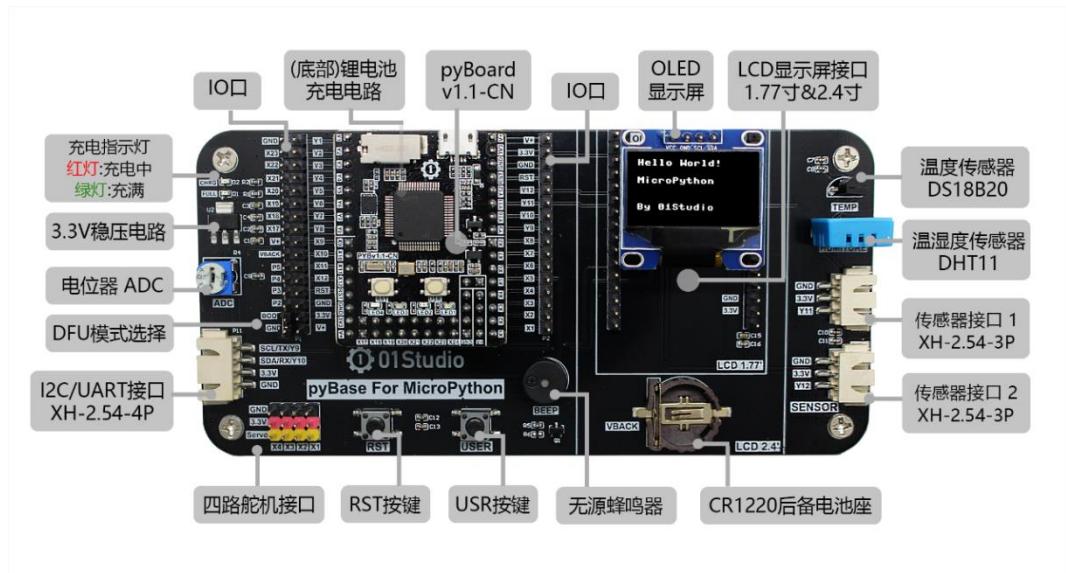


图 1-12 pyBoard 开发套件

主要配置	
核心板	pyBoard v1.1-CN/ pyBoard Mini/ pyBoard Plus
底板	pyBase
显示屏	0.9 寸 OLED 显示屏

表 1-4

### 1.4.2 pyBase

pyBase 是 01Studio 针对各 micropython 开发平台量身定制的底板，可以使用户它可以做更多的 MicroPython 实验，pyBase 同时设计了外设接口，扩展性非常强。以下是详细的功能说明：

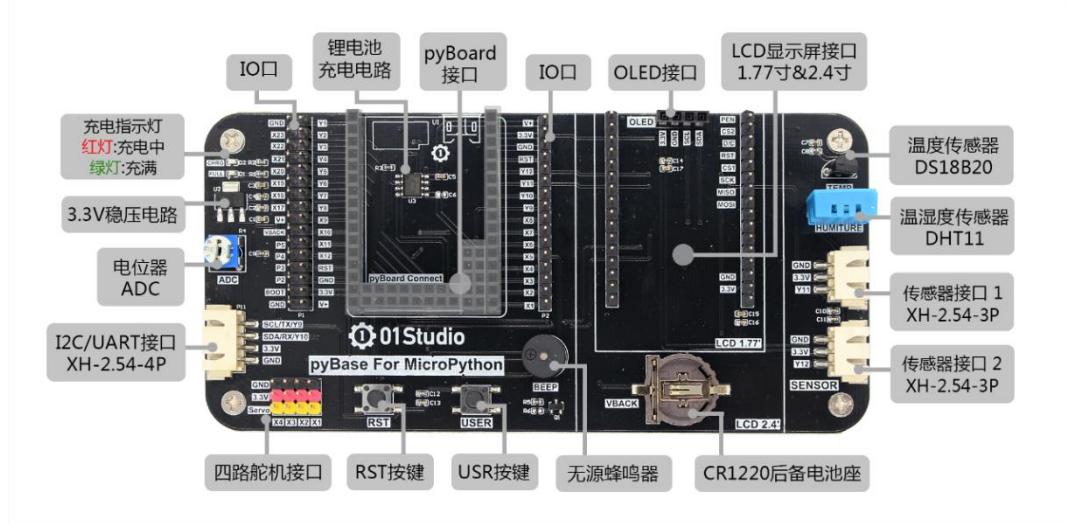


图 1-13 pyBase 功能说明

功能参数	
pyboard 接口	兼容 pyboard v1.1-CN 以及官方的 pyboard
2.54mm 排针	引出 pyboard 全部接口
按键 2 个	引出 RST 和 USR 按键
电位器	ADC 输入
无源蜂鸣器	DAC 输出
纽扣电池座	安装 CR1220 纽扣电池
Servo 接口	舵机接口 X1-X4 (连接舵机需要外接隔离电路)
OLED 接口	4P/0.96 寸/I2C/OLED 显示屏
1.77 寸 LCD 接口	适用于 01Studio 1.77 寸 LCD 显示屏
2.4 寸 LCD 接口	适用于 01Studio 2.4 寸 LCD 显示屏 (带电阻触摸)
温度传感器	DS18B20
温湿度传感器	DHT11
Sensor 接口 1	3P 防呆接口，用于外接传感器

Sensor 接口 1	3P 防呆接口，用于外接传感器
通讯接口	4P 防呆接口，用于外接 UART/I2C 设备
锂电池充电电路	对接入的锂电池进行充电（红灯->充电，绿灯->充满）

表 1-5

### 1.4.3 IOT/通讯模块

pyIOT 开源项目由 01Studio 发起，旨在为市面上成熟的串口物联网模块开发 MicroPython 库，让用户可以使用 python 编程快速实现各类物联网相关应用。

#### 1.4.3.1 pyIOT-BLE TLS01

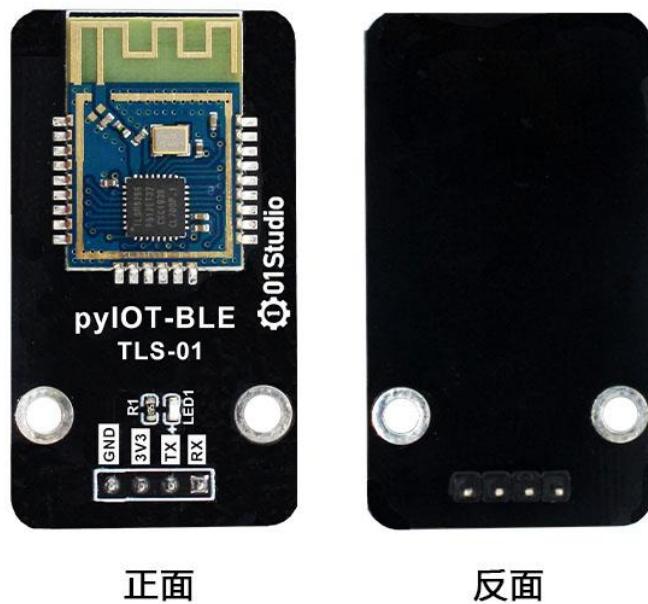


图 1-14

功能参数	
蓝牙主控	TLSR8266
控制方式	UART (串口)
蓝牙版本	BLE 4.0 (从机)
功耗	工作电流: <8.8mA, 广播电流: <1mA
引脚	GND, 3.3V, TX, RX
模块尺寸	4.5*2.5cm

表 1-6

### 1.4.3.2 pyIOT-LORA E22

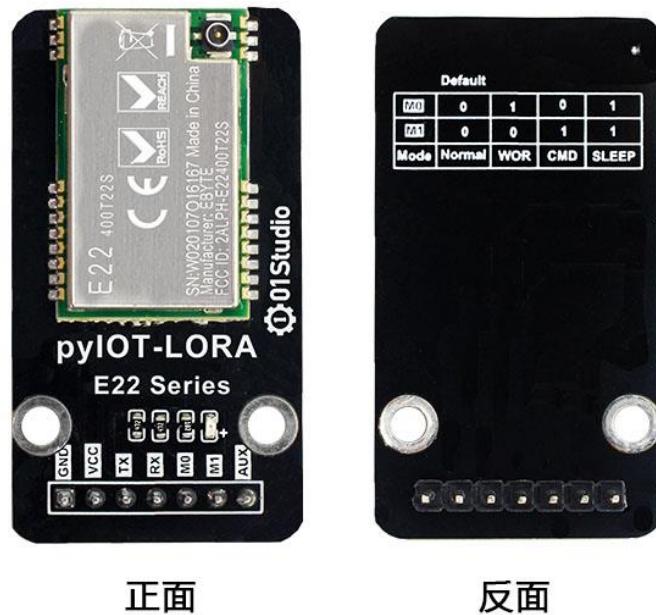


图 1-15

功能参数	
工作频段	410~493MHz (433M)
射频芯片	SX1268
发射功率	22dBm
通讯距离	最大: 5km
通讯接口	UART (串口)
天线	IPEX
发射电流	110mA
供电电压	2.3-5.2V
工作温度	-40°C - 85°C
波特率	1200 – 115200 bps (默认 9600)
空中速率	0.3k – 42.5kbps
接收长度	1024 字节 (自动分包)
模块尺寸	4.2*2.5cm

表 1-7

### 1.4.3.3 pyIOT-GPS

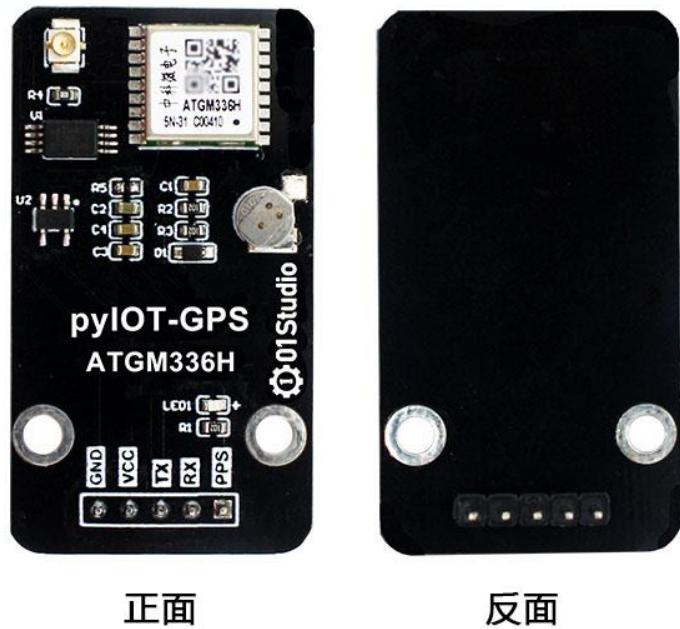


图 1-16

功能参数	
主控芯片	ATGM336H-5N
卫星信号	GPS/BDS/GLONASS
波特率	9600bps
工作电压	3.3V-5V
串口电平	3.3V 或 5V (自适应)
定位精度	2.5m (开阔地点)
功耗	工作: <25mA, 待机: <10uA (@3.3V)
模块尺寸	4.2*2.5cm

表 1-8

## 1.4.4 传感器模块

### 1.4.4.1 人体感应传感器模块

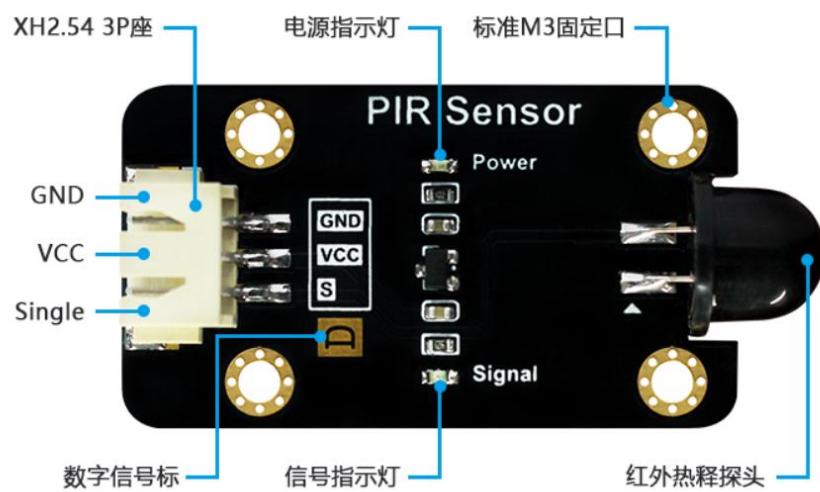


图 1-17 人体感应传感器模块

功能参数	
供电电压	3.3-5V
工作电流	<20 mA
工作温度	-20 至 65 °C
接口定义	XH2.54 防呆接口 (3Pin) 【GND、VCC、Single】
输出信号	数字信号： 检测到人体：高电平 3.3V；持续 3-8 秒 未检测到人体：低电平 0V。
感应角度	100°
感应距离	8 米
模块尺寸	4.5*2.5cm

表 1-9

#### 1.4.4.2 光敏传感器模块

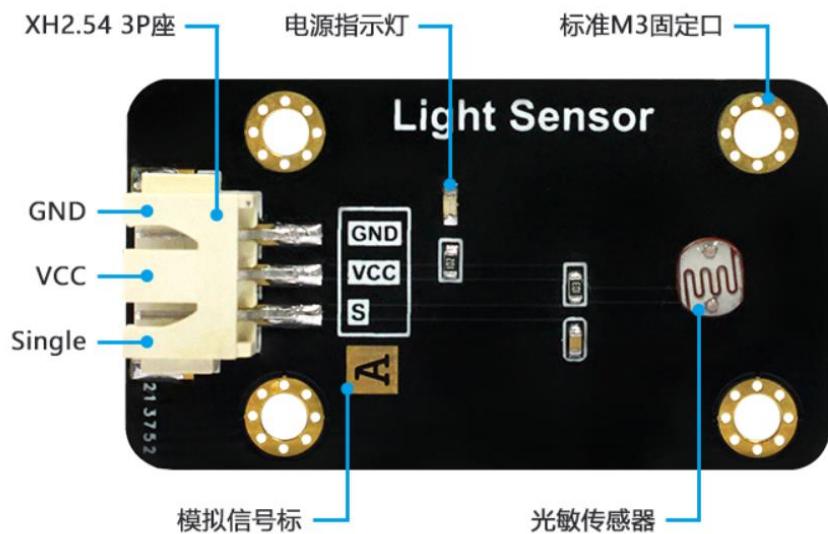


图 1-18 光敏传感器模块

功能参数	
供电电压	3.3-5V
工作电流	<20 mA
工作温度	-20 至 85°C
接口定义	XH2.54 防呆接口 (3Pin) 【GND、VCC、Single】
输出信号	模拟信号: 0-3.3V (VCC=3.3V 时)
模块尺寸	4.5*2.5cm

表 1-10

#### 1.4.4.3 土壤湿度传感器模块

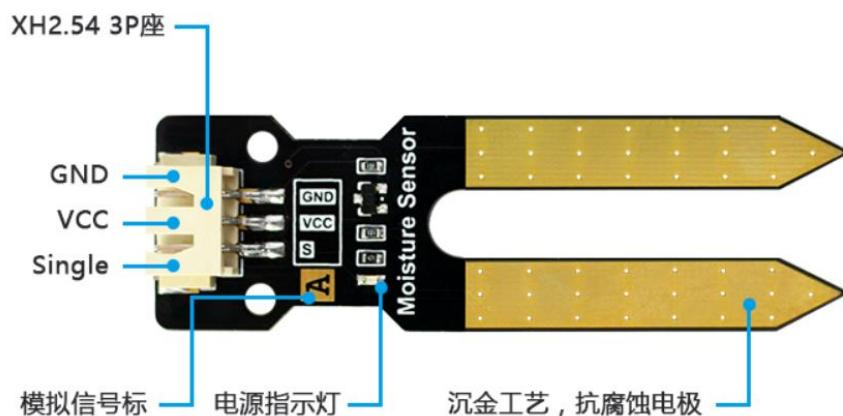


图 1-19 土壤湿度传感器模块

功能参数	
供电电压	3.3-5V
工作电流	<20 mA
工作温度	-40 至 85°C
接口定义	XH2.54 防呆接口 (3Pin) 【GND、VCC、Single】
输出信号	模拟信号: 0-3.3V (VCC=3.3V 时)
模块尺寸	6.6*2.0cm

表 1-11

#### 1.4.4.4 水位传感器模块

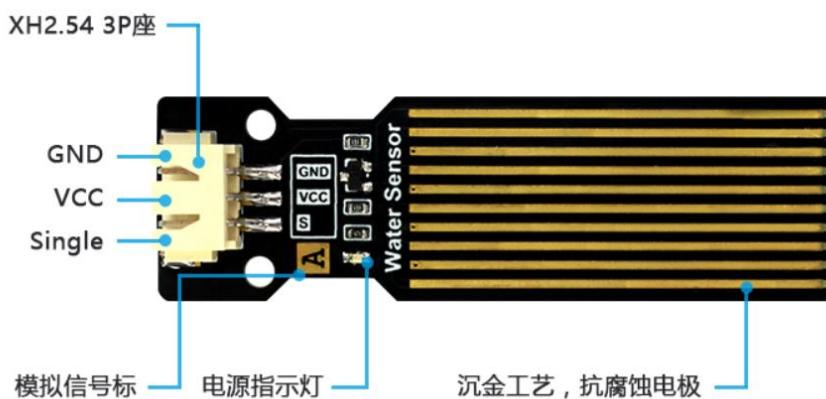


图 1-20 水位传感器模块

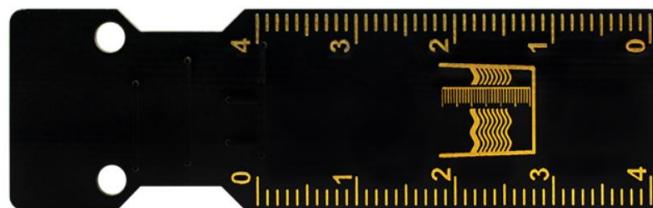


图 1-21 背面

功能参数	
供电电压	3.3-5V
工作电流	<20 mA
工作温度	0 至 60°C
接口定义	XH2.54 防呆接口 (3Pin) 【GND、VCC、Single】
输出信号	模拟信号：0-3.3V (VCC=3.3V 时)
模块尺寸	6.6*2.0cm
注意事项	水位传感器接口旁元件不能接触水，否则容易短路。所以测量时候要注意液体水位的高度。

表 1-12

#### 1.4.4.5 大气压强传感器模块

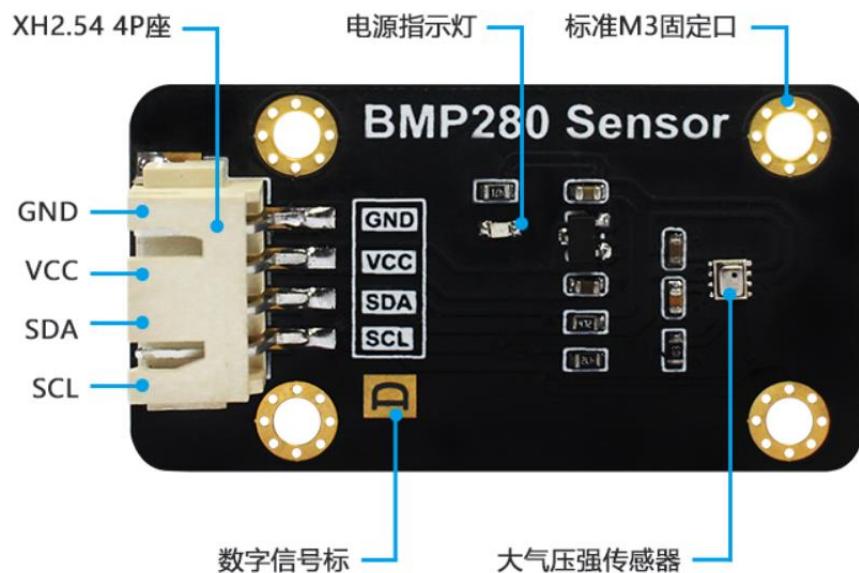


图 1-22 大气压强传感器模块

功能参数	
传感器型号	BMP280
供电电压	3.3-5V
工作电流	<20 mA
工作温度	-20 至 85°C
接口定义	XH2.54 防呆接口 (4Pin) 【GND、VCC、SDA、SCL】
通讯信号	I2C 总线
I2C 地址	0x76 (BMP280 的 SDO 默认下拉); 当 BMP280 的 SDO 引脚上拉时, I2C 地址为: 0x77
模块尺寸	4.5*2.5cm

表 1-13

#### 1.4.4.6 超声波模块

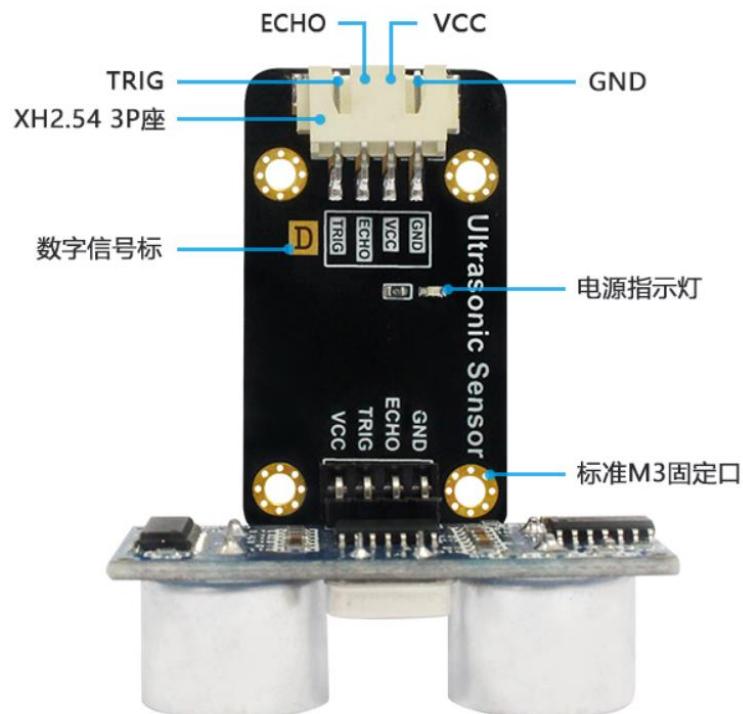


图 1-23 超声波模块

功能参数	
传感器型号	HCSR04
供电电压	3.3-5V
工作电流	<20 mA
工作温度	-20 至 85°C
接口定义	XH2.54 防呆接口 (4Pin) 【GND、VCC、ECHO、TRIG】
通讯信号	IO 数字接口
测量距离	2-450 cm
测量精度	0.5 cm
整体尺寸	5.5*4.5*3.0 cm

表 1-14

#### 1.4.4.7 温度传感器模块（金属探头封装）



图 1-24 金属探头温度传感器

功能参数	
供电电压	3.3-5V
工作温度	-55 至 125 °C
接口定义	XH2.54 防呆接口 (3Pin) 【GND、VCC、Single】
通讯信号	单总线
测量范围	-55 至 125 °C
测量精度	-0.5 °C
探头尺寸	5 * 0.6cm
引线长度	1 米

表 1-15

## 1.4.5 拓展配件

### 1.4.5.1 OLED 显示屏



图 1-25 OLED 显示屏

功能参数	
供电电压	3.3V
屏幕尺寸	0.9 寸
颜色参数	黑底白字
通讯方式	I2C 总线
接口定义	2.54mm 排针 (4Pin) 【VCC、GND、SCL、SDA】
整体尺寸	2.8*2.8cm

表 1-16

#### 1.4.5.2 1.77 寸 LCD 显示屏

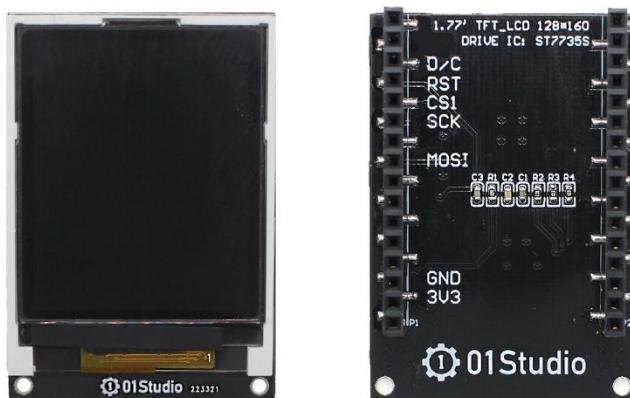


图 1-26 1.77 寸 LCD 显示屏

功能参数	
供电电压	3.3V
屏幕尺寸	1.77 寸
分辨率	128*160
颜色参数	TFT 彩色
驱动芯片	ST7735S
通讯方式	SPI 总线
接口定义	2.54mm 排母（兼容 pyboard v1.1 接口）
整体尺寸	5.0*3.5 cm

表 1-17

### 1.4.5.3 2.4 寸 LCD 显示屏（电阻触摸）

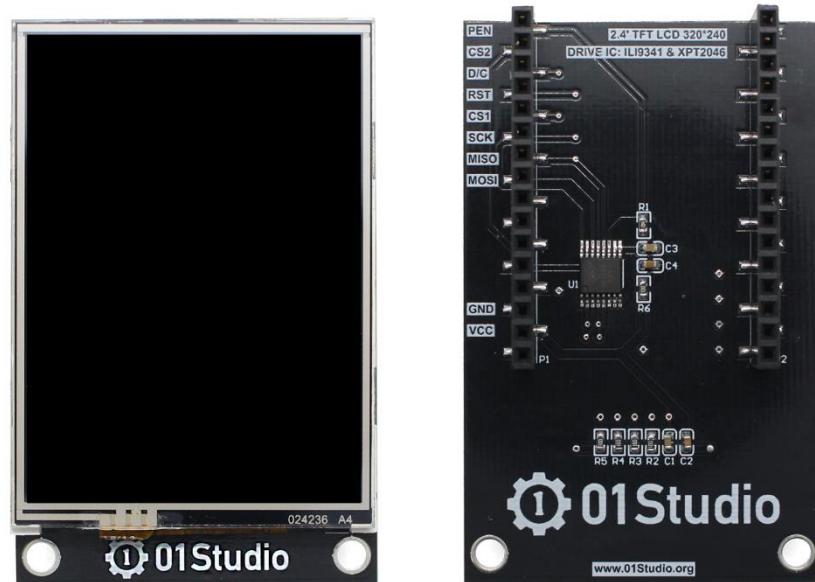


图 1-27 2.4 寸 LCD 显示屏（电阻触摸）

功能参数	
供电电压	3.3V
屏幕尺寸	2.4 寸
分辨率	240*320
颜色参数	TFT 彩色
驱动芯片	ILI9341+XPT2046
触摸方式	电阻屏
通讯方式	SPI 总线
接口定义	2.54mm 排母（兼容 pyboard v1.1 接口）
整体尺寸	6.6*4.2 cm

表 1-18

#### 1.4.5.4 3.2 寸 LCD 显示屏（电阻触摸）

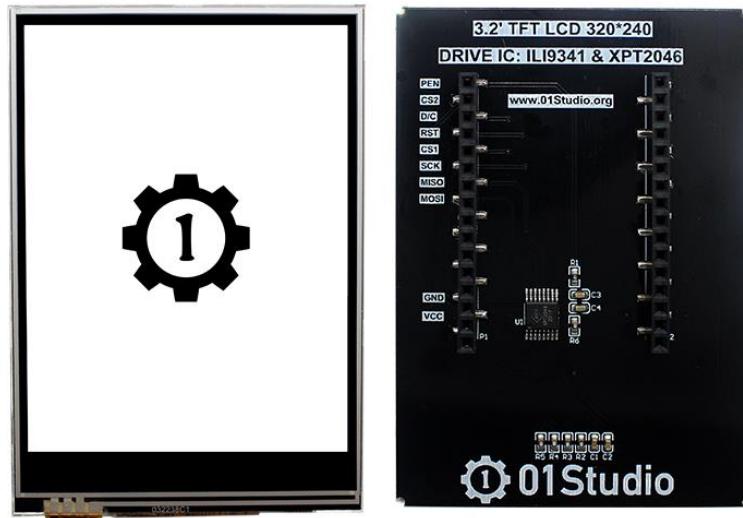


图 1-28 3.2 寸 LCD 显示屏（电阻触摸）

功能参数	
供电电压	3.3V
屏幕尺寸	3.2 寸
分辨率	240*320
颜色参数	TFT 彩色
驱动芯片	ILI9341+XPT2046
触摸方式	电阻屏
通讯方式	SPI 总线
接口定义	2.54mm 排母（兼容 pyboard v1.1 接口）
整体尺寸	7.7*5.5 cm

表 1-19

#### 1.4.5.5 W5500 以太网模块

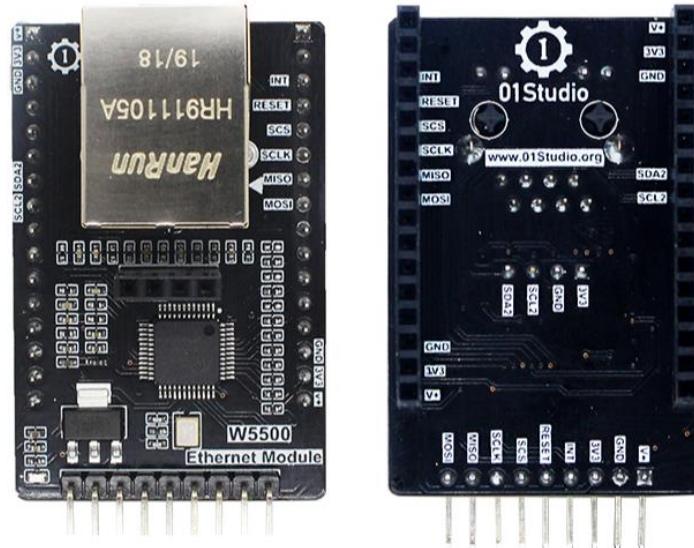


图 1-29 W5500 以太网模块

功能参数	
供电电压	3.3V 或 5V
主控芯片	W5500
控制方式	SPI 总线
接口定义	1、2.54mm 排母（兼容 pyboard v1.1 接口）; 2、2.54mm 排针
其它功能	支持 OLED 扩展接口
整体尺寸	6.6*4.2 cm

表 1-20

#### 1.4.5.6 舵机

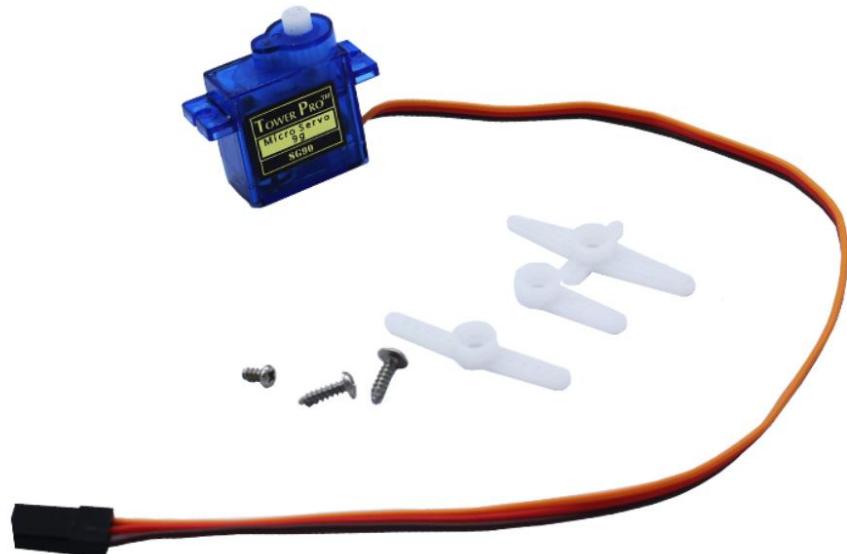


图 1-30 舵机

功能参数	
尺寸	32*30*1.1 mm
重量	15g
扭力	1.6kg/cm
接口	XH2.54 接口 (3Pin) 【GND (黑)、VCC (红)、Single (橙)】
工作电压	3.5-6V
工作温度	-30 °C - 60 °C
转动角度	180° 和 360° 连续旋转。
应用场景	固定翼、直升机 KT、机器人、机械臂、航模等

表 1-21

#### 1.4.5.7 RGB 灯带

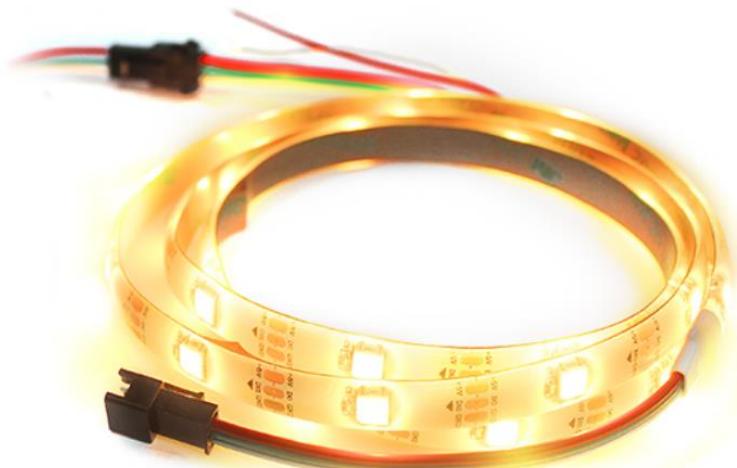


图 1-31 RGB 灯带

功能参数	
灯带长度	1 米
灯珠数量	30
颜色	RGB 全彩
接口定义	XH2.54 防呆接口 (3Pin) 【GND、VCC、Single】
驱动 IC	WS2812B
供电电压	3.3-5V
功率	每颗灯珠最高 0.3W
应用场景	流水灯/呼吸灯/节日灯饰布置等

表 1-22

#### 1.4.5.8 USB 转串口 TTL

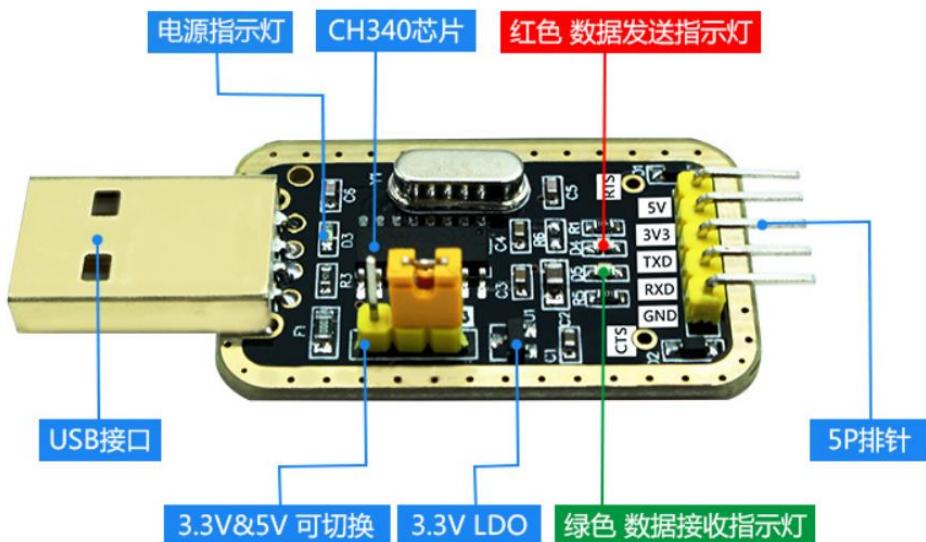


图 1-32 USB 转串口 TTL

功能参数	
驱动芯片	CH340
对外供电	5V 和 3.3V
支持电平	3.3V 和 5V 可切换
引脚接口	2.54MM 排针

表 1-23

## 第2章 Python 基础知识

由于 python 语言是学习 micropython 的基础，为了照顾没有 python 基础的朋友，我们一直在思考应该给大家提供怎么样的教程。Python 相关的书籍和学习资料相信大家能在网上找到不少，所以这里整理给大家的经典 python3 快速学习资料，你甚至可以使用它来当作 python 字典查阅！

【原著：Louie Dinh，翻译：Geoff Liu】

```
# 用井字符开头的是单行注释
```

```
""" 多行字符串用三个引号  
包裹，也常被用来做多  
行注释
```

```
"""
```

### 2.1 原始数据类型和运算符

```
#####
## 1. 原始数据类型和运算符
#####

# 整数
3 # => 3

# 算术没有什么出乎意料的
1 + 1 # => 2
8 - 1 # => 7
10 * 2 # => 20
```

```
# 但是除法例外，会自动转换成浮点数

35 / 5 # => 7.0

5 / 3 # => 1.6666666666666667


# 整数除法的结果都是向下取整

5 // 3      # => 1

5.0 // 3.0 # => 1.0 # 浮点数也可以

-5 // 3    # => -2

-5.0 // 3.0 # => -2.0


# 浮点数的运算结果也是浮点数

3 * 2.0 # => 6.0


# 模除

7 % 3 # => 1


# x 的 y 次方

2**4 # => 16


# 用括号决定优先级

(1 + 3) * 2 # => 8


# 布尔值

True

False


# 用 not 取非

not True # => False

not False # => True
```

```
# 逻辑运算符，注意 and 和 or 都是小写

True and False # => False

False or True # => True


# 整数也可以当作布尔值

0 and 2 # => 0

-5 or 0 # => -5

0 == False # => True

2 == True # => False

1 == True # => True


# 用==判断相等

1 == 1 # => True

2 == 1 # => False


# 用!=判断不等

1 != 1 # => False

2 != 1 # => True


# 比较大小

1 < 10 # => True

1 > 10 # => False

2 <= 2 # => True

2 >= 2 # => True


# 大小比较可以连起来！

1 < 2 < 3 # => True

2 < 3 < 2 # => False
```

```
# 字符串用单引双引都可以
"这是个字符串"
'这也是个字符串'

# 用加号连接字符串
"Hello " + "world!" # => "Hello world!"

# 字符串可以被当作字符列表
"This is a string"[0] # => 'T'

# 用.format 来格式化字符串
"{} can be {}".format("strings", "interpolated")

# 可以重复参数以节省时间
"{0} be nimble, {0} be quick, {0} jump over the {1}".format("Jack",
"candle stick")
# => "Jack be nimble, Jack be quick, Jack jump over the candle stick"

# 如果不想数参数，可以用关键字
"{name} wants to eat {food}".format(name="Bob", food="lasagna")
# => "Bob wants to eat lasagna"

# 如果你的 Python3 程序也要在 Python2.5 以下环境运行，也可以用老式的格式化语法
"%s can be %s the %s way" % ("strings", "interpolated", "old")

# None 是一个对象
None # => None

# 当与 None 进行比较时不要用 ==，要用 is。is 是用来比较两个变量是否指向同一个对
象。
```

```
"etc" is None # => False
None is None # => True

# None, 0, 空字符串, 空列表, 空字典都算是 False
# 所有其他值都是 True

bool(0) # => False
bool("") # => False
bool([]) # => False
bool({}) # => False
```

## 2.2 变量和集合

```
#####
## 2. 变量和集合
#####

# print 是内置的打印函数
print("I'm Python. Nice to meet you!")

# 在给变量赋值前不用提前声明
# 传统的变量命名是小写，用下划线分隔单词
some_var = 5
some_var # => 5

# 访问未赋值的变量会抛出异常
# 参考流程控制一段来学习异常处理
some_unknown_var # 抛出 NameError


# 用列表(list)储存序列
li = []
# 创建列表时也可以同时赋给元素
other_li = [4, 5, 6]

# 用 append 在列表最后追加元素
li.append(1)    # li 现在是[1]
li.append(2)    # li 现在是[1, 2]
li.append(4)    # li 现在是[1, 2, 4]
li.append(3)    # li 现在是[1, 2, 4, 3]

# 用 pop 从列表尾部删除
```

```
li.pop()          # => 3 且 li 现在是[1, 2, 4]
# 把 3 再放回去

li.append(3)    # li 变回[1, 2, 4, 3]

# 列表存取跟数组一样

li[0]  # => 1

# 取出最后一个元素

li[-1] # => 3

# 越界存取会造成 IndexError

li[4] # 抛出 IndexError

# 列表有切割语法

li[1:3] # => [2, 4]

# 取尾

li[2:] # => [4, 3]

# 取头

li[:3] # => [1, 2, 4]

# 隔一个取一个

li[::-2] # => [1, 4]

# 倒排列表

li[::-1] # => [3, 4, 2, 1]

# 可以用三个参数的任何组合来构建切割

# li[始:终:步伐]

# 用 del 删除任何一个元素

del li[2] # li is now [1, 2, 3]

# 列表可以相加

# 注意: li 和 other_li 的值都不变
```

```
li + other_li    # => [1, 2, 3, 4, 5, 6]

# 用 extend 拼接列表

li.extend(other_li)    # li 现在是[1, 2, 3, 4, 5, 6]

# 用 in 测试列表是否包含值

1 in li    # => True

# 用 len 取列表长度

len(li)    # => 6

# 元组是不可改变的序列

tup = (1, 2, 3)

tup[0]    # => 1

tup[0] = 3  # 抛出 TypeError

# 列表允许的操作元组大都可以

len(tup)    # => 3

tup + (4, 5, 6)    # => (1, 2, 3, 4, 5, 6)

tup[:2]    # => (1, 2)

2 in tup    # => True

# 可以把元组合成列表解包，赋值给变量

a, b, c = (1, 2, 3)    # 现在 a 是 1, b 是 2, c 是 3

# 元组周围的括号是可以省略的

d, e, f = 4, 5, 6

# 交换两个变量的值就这么简单

e, d = d, e    # 现在 d 是 5, e 是 4
```

```
# 用字典表达映射关系

empty_dict = {}

# 初始化的字典

filled_dict = {"one": 1, "two": 2, "three": 3}

# 用[]取值

filled_dict["one"] # => 1

# 用 keys 获得所有的键。

# 因为 keys 返回一个可迭代对象，所以在这里把结果包在 list 里。我们下面会详细介绍可迭代。

# 注意：字典键的顺序是不定的，你得到的结果可能和以下不同。

list(filled_dict.keys()) # => ["three", "two", "one"]

# 用 values 获得所有的值。跟 keys 一样，要用 list 包起来，顺序也可能不同。

list(filled_dict.values()) # => [3, 2, 1]

# 用 in 测试一个字典是否包含一个键

"one" in filled_dict # => True

1 in filled_dict # => False

# 访问不存在的键会导致 KeyError

filled_dict["four"] # KeyError

# 用 get 来避免 KeyError

filled_dict.get("one") # => 1
```

```
filled_dict.get("four")    # => None
# 当键不存在的时候 get 方法可以返回默认值
filled_dict.get("one", 4)   # => 1
filled_dict.get("four", 4)   # => 4

# setdefault 方法只有当键不存在的时候插入新值
filled_dict.setdefault("five", 5)  # filled_dict["five"]设为 5
filled_dict.setdefault("five", 6)  # filled_dict["five"]还是 5

# 字典赋值
filled_dict.update({"four":4}) # => {"one": 1, "two": 2, "three": 3,
"four": 4}
filled_dict["four"] = 4  # 另一种赋值方法

# 用 del 删除
del filled_dict["one"]  # 从 filled_dict 中把 one 删除

# 用 set 表达集合
empty_set = set()
# 初始化一个集合，语法跟字典相似。
some_set = {1, 1, 2, 2, 3, 4}  # some_set 现在是{1, 2, 3, 4}

# 可以把集合赋值于变量
filled_set = some_set

# 为集合添加元素
filled_set.add(5)  # filled_set 现在是{1, 2, 3, 4, 5}

# & 取交集
```

```
other_set = {3, 4, 5, 6}
filled_set & other_set    # => {3, 4, 5}

# | 取并集
filled_set | other_set    # => {1, 2, 3, 4, 5, 6}

# - 取补集
{1, 2, 3, 4} - {2, 3, 5}    # => {1, 4}

# in 测试集合是否包含元素
2 in filled_set    # => True
10 in filled_set   # => False
```

## 2.3 流程控制和迭代器

```
#####
## 3. 流程控制和迭代器
#####

# 先随便定义一个变量
some_var = 5

# 这是个 if 语句。注意缩进在 Python 里是有意义的
# 印出"some_var 比 10 小"
if some_var > 10:
    print("some_var 比 10 大")
elif some_var < 10:      # elif 句是可选的
    print("some_var 比 10 小")
else:                  # else 也是可选的
    print("some_var 就是 10")

"""

用 for 循环语句遍历列表
打印:
    dog is a mammal
    cat is a mammal
    mouse is a mammal
"""

for animal in ["dog", "cat", "mouse"]:
    print("{} is a mammal".format(animal))

"""


```

```
"range(number)"返回数字列表从 0 到给的数字
```

```
打印：
```

```
0
```

```
1
```

```
2
```

```
3
```

```
....
```

```
for i in range(4):
```

```
    print(i)
```

```
....
```

```
while 循环直到条件不满足
```

```
打印：
```

```
0
```

```
1
```

```
2
```

```
3
```

```
....
```

```
x = 0
```

```
while x < 4:
```

```
    print(x)
```

```
    x += 1 # x = x + 1 的简写
```

```
# 用 try/except 块处理异常状况
```

```
try:
```

```
    # 用 raise 抛出异常
```

```
    raise IndexError("This is an index error")
```

```
except IndexError as e:
```

```
    pass # pass 是无操作，但是应该在这里处理错误
```

```
except (TypeError, NameError):
```

```
pass      # 可以同时处理不同类的错误

else:    # else 语句是可选的，必须在所有的 except 之后

    print("All good!")  # 只有当 try 运行完没有错误的时候这句才会运行

# Python 提供一个叫做可迭代(iterable)的基本抽象。一个可迭代对象是可以被当作序
列

# 的对象。比如说上面 range 返回的对象就是可迭代的。

filled_dict = {"one": 1, "two": 2, "three": 3}

our_iterable = filled_dict.keys()

print(our_iterable) # => dict_keys(['one', 'two', 'three']), 是一个实现可
迭代接口的对象

# 可迭代对象可以遍历

for i in our_iterable:

    print(i)    # 打印 one, two, three

# 但是不可以随机访问

our_iterable[1] # 抛出 TypeError

# 可迭代对象知道怎么生成迭代器

our_iterator = iter(our_iterable)

# 迭代器是一个可以记住遍历的位置的对象

# 用 __next__ 可以取得下一个元素

our_iterator.__next__() # => "one"

# 再一次调取 __next__ 时会记得位置

our_iterator.__next__() # => "two"
```

```
our_iterator.__next__() # => "three"

# 当迭代器所有元素都取出后，会抛出 StopIteration
our_iterator.__next__() # 抛出 StopIteration

# 可以用 list 一次取出迭代器所有的元素
list(filled_dict.keys()) # Returns ["one", "two", "three"]
```

## 2.4 函数

```
#####
## 4. 函数
#####

# 用 def 定义新函数
def add(x, y):
    print("x is {} and y is {}".format(x, y))
    return x + y      # 用 return 语句返回

# 调用函数
add(5, 6)    # => 印出"x is 5 and y is 6"并且返回 11

# 也可以用关键字参数来调用函数
add(y=6, x=5)    # 关键字参数可以用任何顺序

# 我们可以定义一个可变参数函数
def varargs(*args):
    return args

varargs(1, 2, 3)    # => (1, 2, 3)

# 我们也可以定义一个关键字可变参数函数
def keyword_args(**kwargs):
    return kwargs
```

```

# 我们来看看结果是什么:

keyword_args(big="foot", loch="ness")    # => {"big": "foot", "loch": "ness"}


# 这两种可变参数可以混着用

def all_the_args(*args, **kwargs):
    print(args)
    print(kwargs)
    """
all_the_args(1, 2, a=3, b=4) prints:
(1, 2)
{"a": 3, "b": 4}
"""

# 调用可变参数函数时可以做跟上面相反的，用*展开序列，用**展开字典。

args = (1, 2, 3, 4)
kwargs = {"a": 3, "b": 4}
all_the_args(*args)    # 相当于 foo(1, 2, 3, 4)
all_the_args(**kwargs)  # 相当于 foo(a=3, b=4)
all_the_args(*args, **kwargs)  # 相当于 foo(1, 2, 3, 4, a=3, b=4)


# 函数作用域

x = 5


def setX(num):
    # 局部作用域的 x 和全局域的 x 是不同的
    x = num # => 43
    print (x) # => 43

```

```

def setGlobalX(num):
    global x
    print (x) # => 5
    x = num # 现在全局域的 x 被赋值
    print (x) # => 6

setX(43)
setGlobalX(6)

# 函数在 Python 是一等公民
def create_adder(x):
    def adder(y):
        return x + y
    return adder

add_10 = create_adder(10)
add_10(3) # => 13

# 也有匿名函数
(lambda x: x > 2)(3) # => True

# 内置的高阶函数
map(add_10, [1, 2, 3]) # => [11, 12, 13]
filter(lambda x: x > 5, [3, 4, 5, 6, 7]) # => [6, 7]

# 用列表推导式可以简化映射和过滤。列表推导式的返回值是另一个列表。
[add_10(i) for i in [1, 2, 3]] # => [11, 12, 13]
[x for x in [3, 4, 5, 6, 7] if x > 5] # => [6, 7]

```

## 2.5 类

```
#####
## 5. 类
#####

# 定义一个继承 object 的类
class Human(object):

    # 类属性，被所有此类的实例共用。
    species = "H. sapiens"

    # 构造方法，当实例被初始化时被调用。注意名字前后的双下划线，这是表明这个属
    # 性或方法对 Python 有特殊意义，但是允许用户自行定义。你自己取名时不应该用
    这

    # 种格式。
    def __init__(self, name):
        # Assign the argument to the instance's name attribute
        self.name = name

    # 实例方法，第一个参数总是 self，就是这个实例对象
    def say(self, msg):
        return "{name}: {message}".format(name=self.name, message=msg)

    # 类方法，被所有此类的实例共用。第一个参数是这个类对象。
    @classmethod
    def get_species(cls):
        return cls.species
```

```
# 静态方法。调用时没有实例或类的绑定。  
  
@staticmethod  
  
def grunt():  
    return "*grunt*"  
  
  
  
# 构造一个实例  
  
i = Human(name="Ian")  
print(i.say("hi"))      # 印出 "Ian: hi"  
  
  
j = Human("Joel")  
print(j.say("hello"))  # 印出 "Joel: hello"  
  
  
# 调用一个类方法  
i.get_species()  # => "H. sapiens"  
  
  
# 改一个共用的类属性  
Human.species = "H. neanderthalensis"  
i.get_species()  # => "H. neanderthalensis"  
j.get_species()  # => "H. neanderthalensis"  
  
  
# 调用静态方法  
Human.grunt()  # => "*grunt*"
```

## 2.6 模块

```
#####
## 6. 模块
#####

# 用 import 导入模块

import math

print(math.sqrt(16))  # => 4.0


# 也可以从模块中导入个别值

from math import ceil, floor

print(ceil(3.7))  # => 4.0
print(floor(3.7))  # => 3.0


# 可以导入一个模块中所有值

# 警告：不建议这么做

from math import *


# 如此缩写模块名字

import math as m

math.sqrt(16) == m.sqrt(16)  # => True


# Python 模块其实就是普通的 Python 文件。你可以自己写，然后导入，

# 模块的名字就是文件的名字。

# 你可以这样列出一个模块里所有的值

import math

dir(math)
```

## 2.7 高级用法

```
#####
## 7. 高级用法
#####

# 用生成器(generators)方便地写惰性运算

def double_numbers(iterable):
    for i in iterable:
        yield i + i

# 生成器只有在需要时才计算下一个值。它们每一次循环只生成一个值，而不是把所有的
# 值全部算好。
#
# range 的返回值也是一个生成器，不然一个 1 到 900000000 的列表会花很多时间和内
# 存。
#
# 如果你想用一个 Python 的关键字当作变量名，可以加一个下划线来区分。

range_ = range(1, 900000000)
# 当找到一个 >=30 的结果就会停
# 这意味着 `double_numbers` 不会生成大于 30 的数。
for i in double_numbers(range_):
    print(i)
    if i >= 30:
        break

# 装饰器(decorators)
# 这个例子中，beg 装饰 say
```

```
# beg 会先调用 say。如果返回的 say_please 为真， beg 会改变返回的字符串。
from functools import wraps

def beg(target_function):
    @wraps(target_function)
    def wrapper(*args, **kwargs):
        msg, say_please = target_function(*args, **kwargs)
        if say_please:
            return "{} {}".format(msg, "Please! I am poor :(")
        return msg

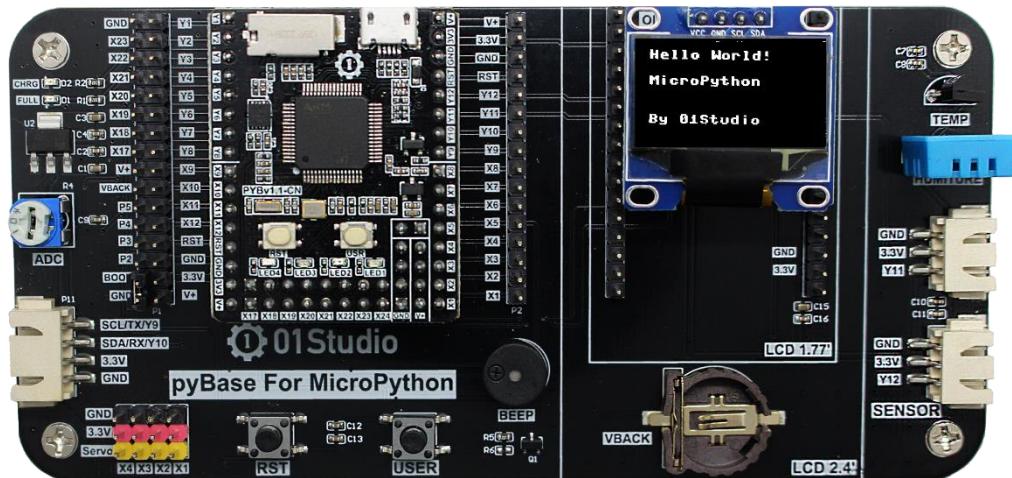
    return wrapper

@beg
def say(say_please=False):
    msg = "Can you buy me a beer?"
    return msg, say_please

print(say()) # Can you buy me a beer?
print(say(say_please=True)) # Can you buy me a beer? Please! I am
poor :(
```

## 第3章 开发环境快速建立

我们以往做嵌入式开发的时候，一般都会用到 IDE 开发软件，比如 KEIL、IAR 等，有些还需要专门的烧录器，这为嵌入式开发增加了门槛。而使用 MicroPython 的话，用户只需要 1 根 MicroUSB 数据线（俗称的安卓手机数据线）就可以开发了。pyBoard 上使用了 STM32 自身 flash 的一定空间作为 U 盘跟电脑交互，所有文件存放在这个几十 KB 空间的里，这使得我们在不同的 PC 平台，Windows/Mac/Linux 上可以直接打开 U 盘文件来开发，你甚至可以用自带的文档编辑器来编写你的程序。我们将会对 Windows、MAC、Linux 不同开发环境的搭建进行讲解。



pyBoard 开发套件

### 3.1 基于 Windows

Windows 是大部分开发人员的主流平台，毕竟微软凭借其 windows 操作系统在 PC 时代统治了几十年，大部分软件厂商还是愿意针对 windows 来开发桌面软件，但这也跟乔布斯英年早逝有关。扯远了，所以为了后面省事，我们还是推荐使用 Windows 作为首选的开发环境，推荐使用 Windows 10，微软官方已经宣布停止对 win 7 更新和维护了，win10 功能强大，有些驱动还能自动联网安装，免去人工安装的麻烦。

#### 3.1.1 安装开发软件 Thonny IDE

开发软件是指我们用来写代码的工具，Python 拥有众多的编程器，如果你之前已经熟练掌握 python 或已经使用 python 开发，那么可以直接使用你原来习惯的开发软件来编程。如果你是初学者或者喜欢简单而快速应用，我们使用官方推荐的 Thonny Python IDE。

Thonny Python IDE 是一款开源软件，以极简方式设计，对 MicroPython 的兼容性非常友善。而且支持 Windows、Mac OS、Linux、树莓派。由于开源，所以软件迭代速度非常快，功能日趋成熟。具体安装方法如下：

该软件可以在 零一科技（01Studio）MicroPython 开发套件配套资料\01-开发工具\01-Windows\MicroPython 开发软件(IDE)\Thonny 下获取。

你也可以在 <https://thonny.org/> 下载最新版，选择自己的开发平台进行下载安装即可(这里选择 Windows！)：



图 3-1 Thonny Python IDE 下载

下载完成后直接双击打开安装即可，安装完成后可以在桌面看到相关图标，打开软件如下：

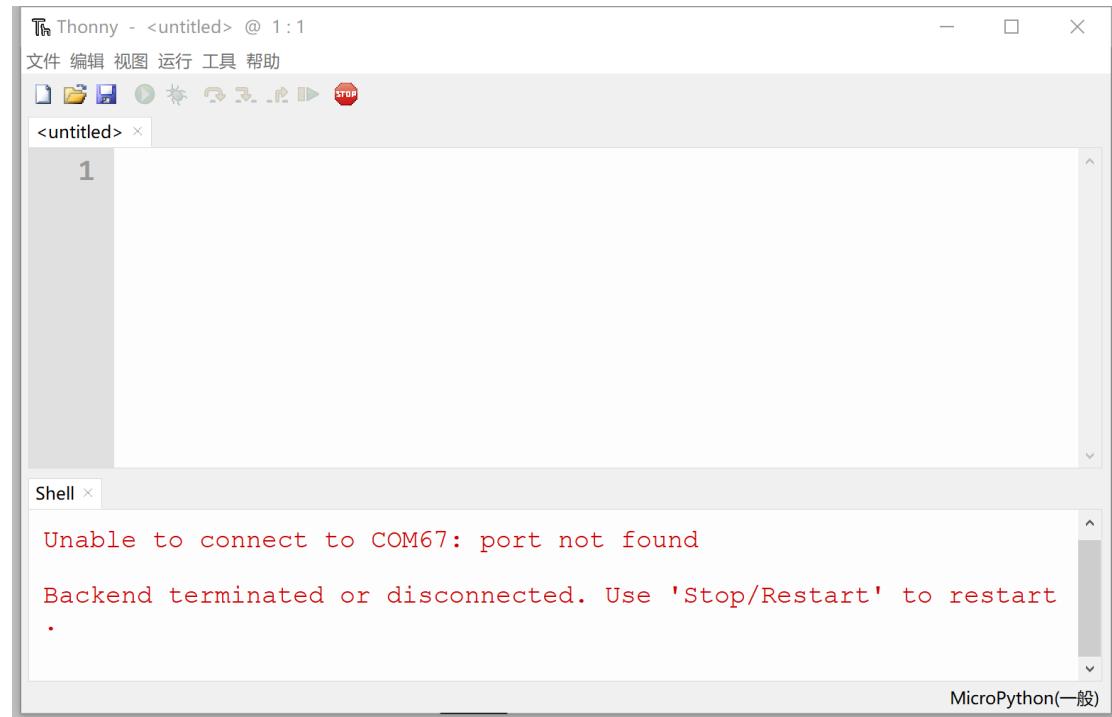


图 3-2 安装完成

至此，Thonny 安装完成。关于如何在 Thonny 上使用 pyBoard，我们在接下来的章节将详细讲解。

### 3.1.2 开发套件使用

#### 3.1.2.1 驱动安装

我们将 MicroPython 开发板通过 MicroUSB 数据线连接到电脑：

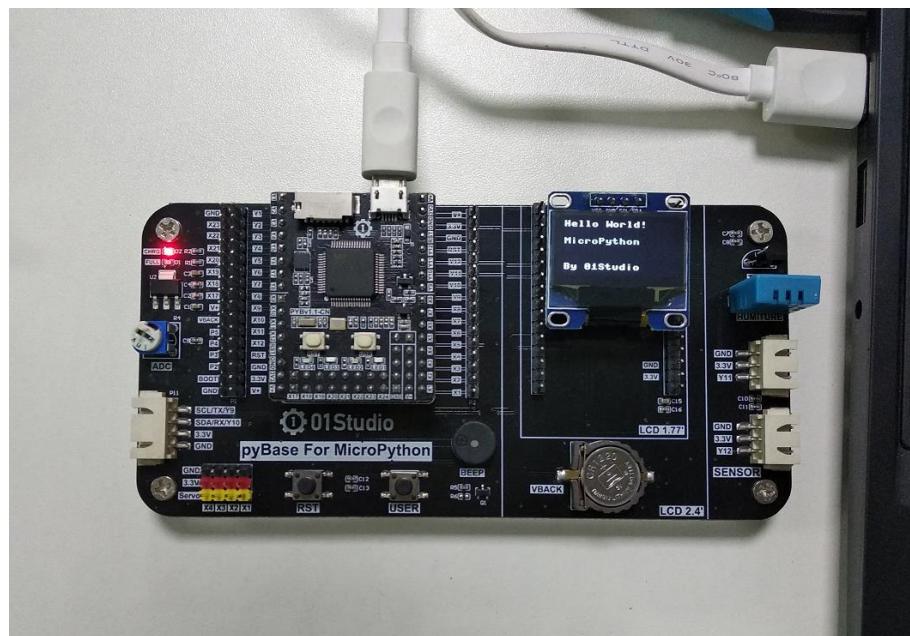


图 3-3 通过 MicroUSB 线连接到电脑

连接电脑后，会发现出现一个 U 盘，名字是 PYBFLASH，点击打开，看到以下文件（也可能包含了其它出厂测试例程文件）。

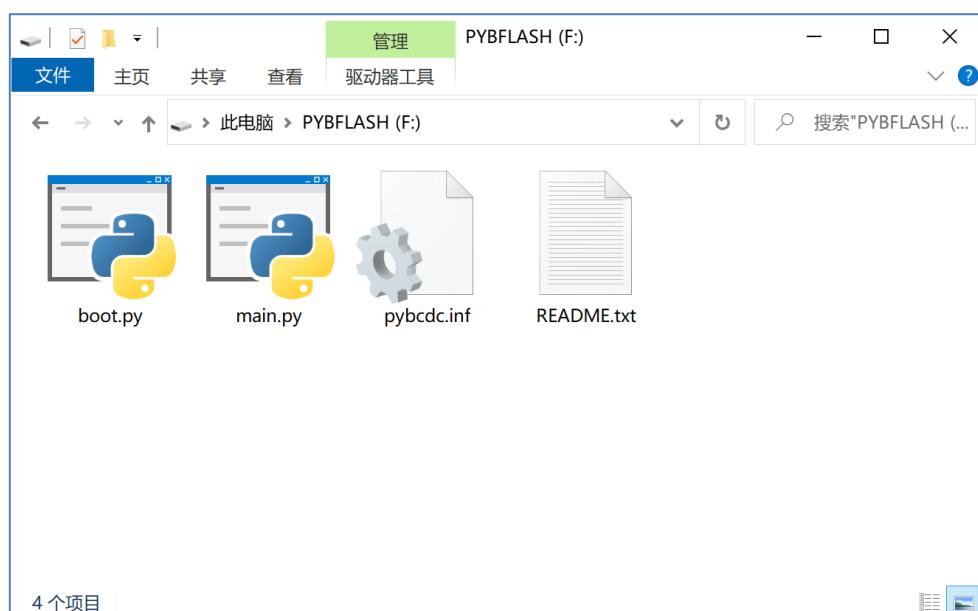


图 3-4 pyBoard 的系统文件

下面是对各个文件的简介，主要是前面 4 个文件构成 MicroPython 的文件系统：

1. **boot.py** : 系统启动文件;
2. **main.py**: 主函数代码文件;
3. **pybcd.inf**: USB 串口调试（REPL）驱动;
4. **README.txt**: 说明文件;
5. 其它出厂代码（会在后面例程讲解）。

我们还需要安装一个 USB 转串口的驱动（一般情况下 win10 系统会自动安装），如果没有自动安装，请按以下方式手动安装。

鼠标右键点击“我的电脑”—属性—设备管理器：系统找到了 pyBoard 虚拟串口新设备（部分 PC 可能不显示 pyboard,只要找到 COM 即可，不确定的可以拔插 pyboard，找到出现和消失的 COM 就是 pyboard 的串口 COM），驱动在 U 盘里，我们安装一下。

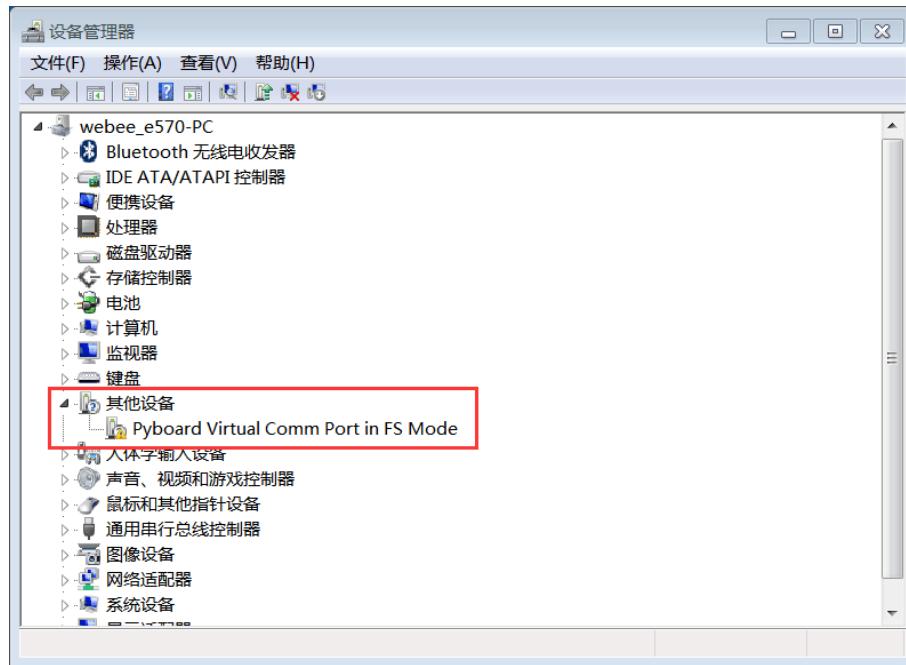


图 3-5 找到对应驱动

点击右键更新驱动软件：

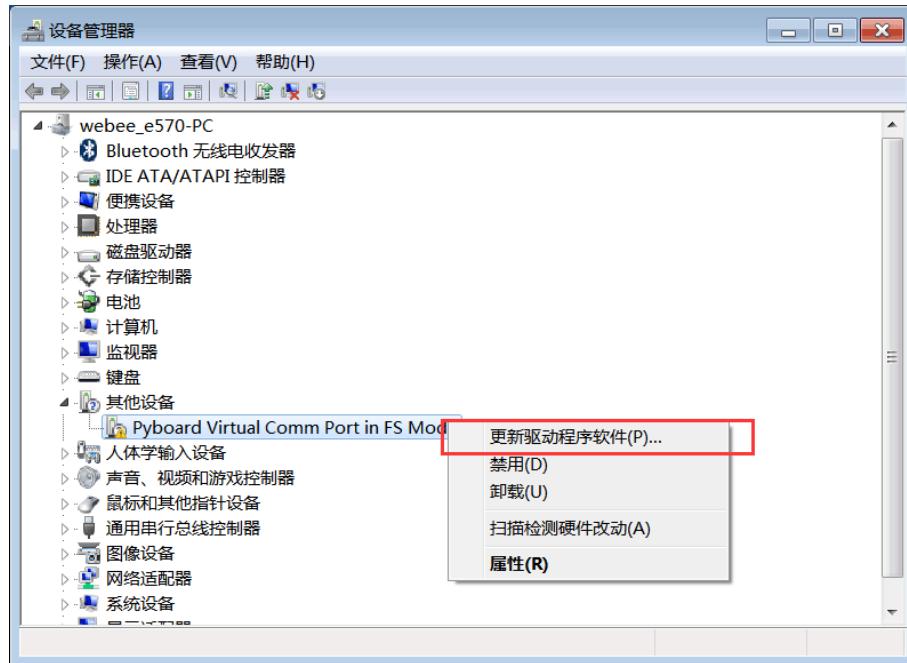


图 3-6 点击右键更新驱动

浏览计算机安装，选择路径是 PYBFLASH，也就是 U 盘盘符。

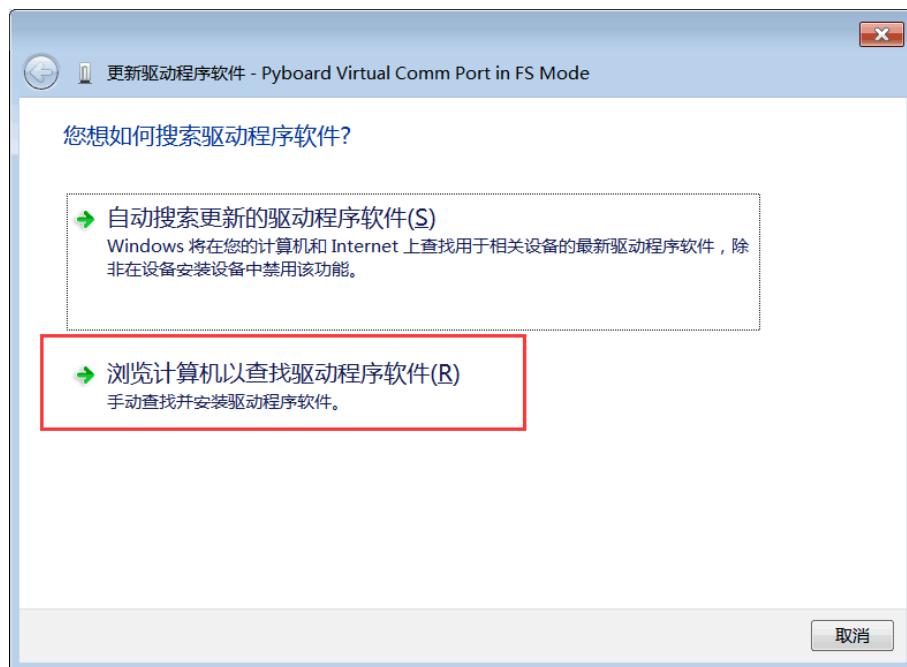


图 3-7

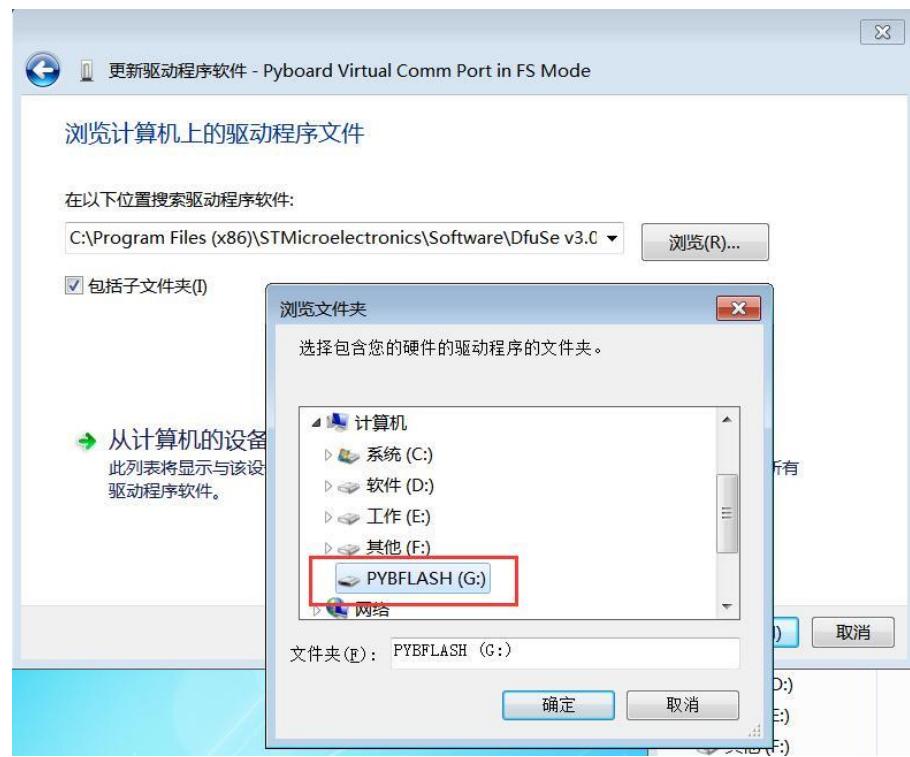


图 3-8 选择 PYBFLASH 盘符

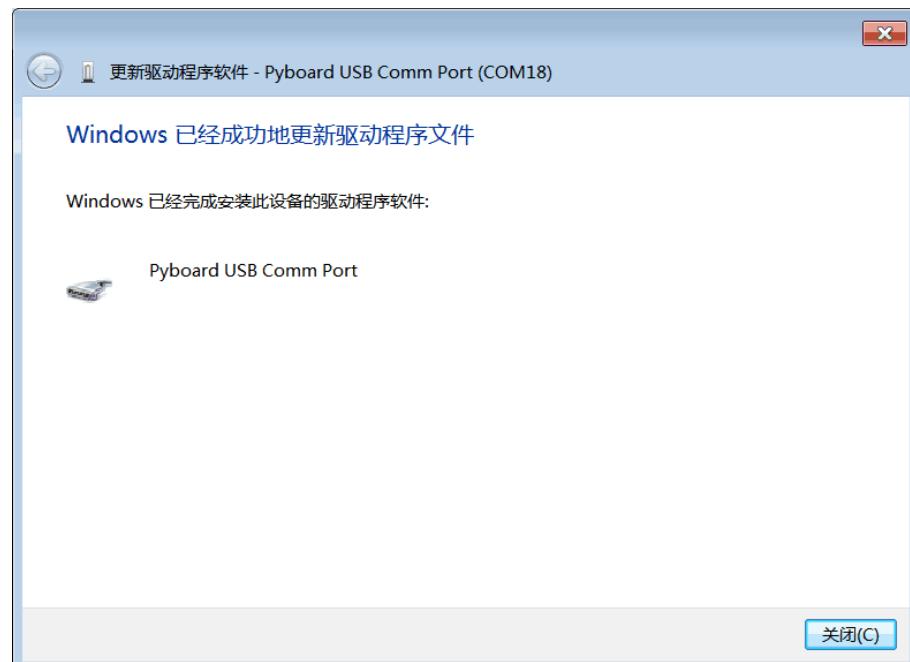


图 3-9 安装完成

安装完成后，我们看到，原来的叹号消失，出现了一个 COM18 串口号。这个串口号不同的电脑会显示不同，以后还会用到。

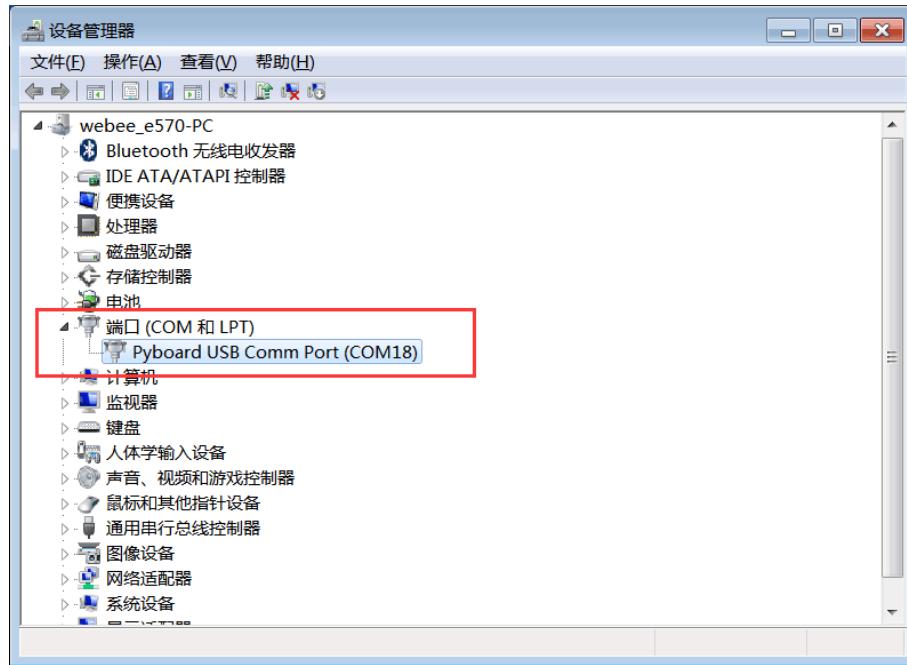


图 3-10 出现串口 COM 信息

### 注意事项：

部分用户的 Win7 系统由于使用的是 Ghost 版系统，Ghost 系统精简了很多系统文件，有可能导致驱动无法安装。我们不建议开发者安装 Ghost 的系统。这时候建议用户采取以下措施。

- (1) 使用 360 的系统重装功能，实测重装后驱动可以安装；
- (2) 升级成 Win10 系统。

我们更推荐方法 (2)，因为微软已经宣布停止对 Win7 进行更新，另外就是我们的教程都会基于 Win10 系统。

### 3.1.2.2 第一个实验

前面我们已经安装好了 Thonny IDE，接下来我们使用最简单的方式来做一个点亮 LED4 蓝色灯的实验，大家暂时先不用理解代码意思，后面章节会有解释。这里主要是为了让大家了解一下 MicroPython 编程软件 Thonny 的使用方法和原理。具体如下：

我们打开 Thonny，将 pyBoard 连接到电脑。点击右下角：



图 3-11 选择要连接的设备

在弹出的列表选择：Configure interpreter

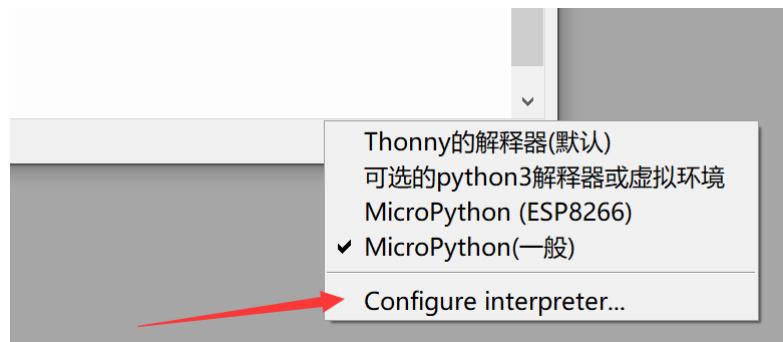


图 3-12

选择“MicroPython 一般”和 pyBoard 对应的串口号，点击确认。（当只有 1 个设备时候，端口可以选择自动获取，使用会更方便。）

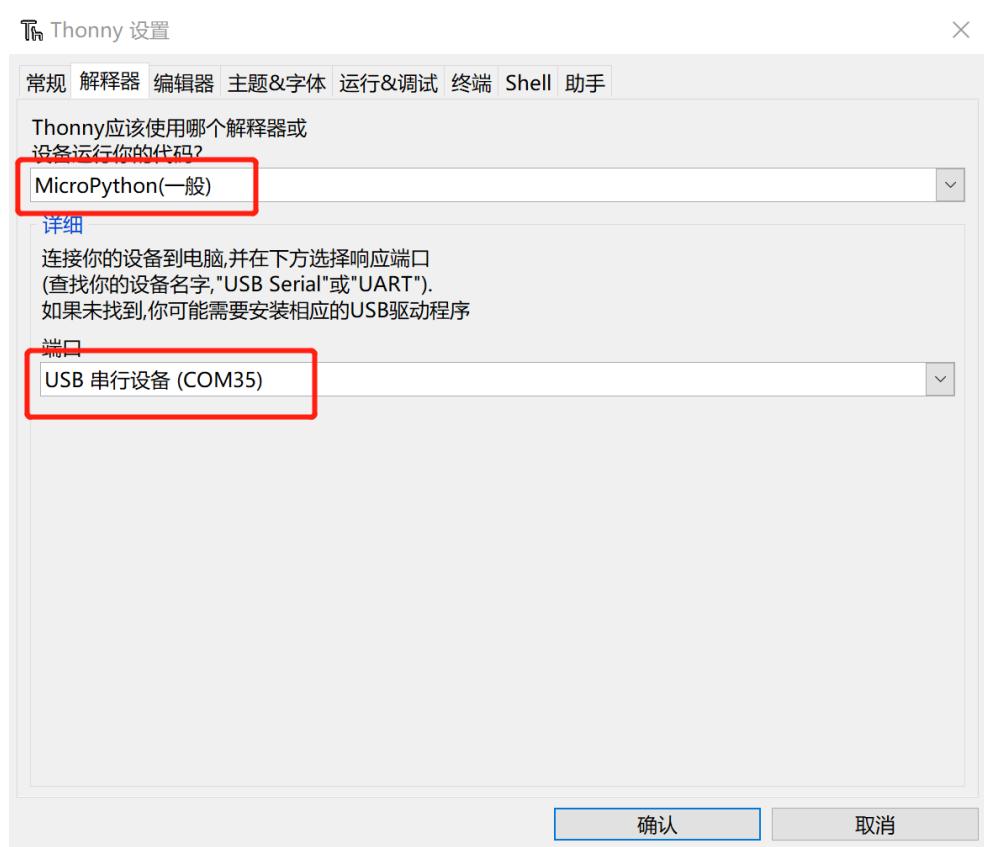


图 3-13

连接成功后可以在 shell（串口终端）看到固件的相关信息：

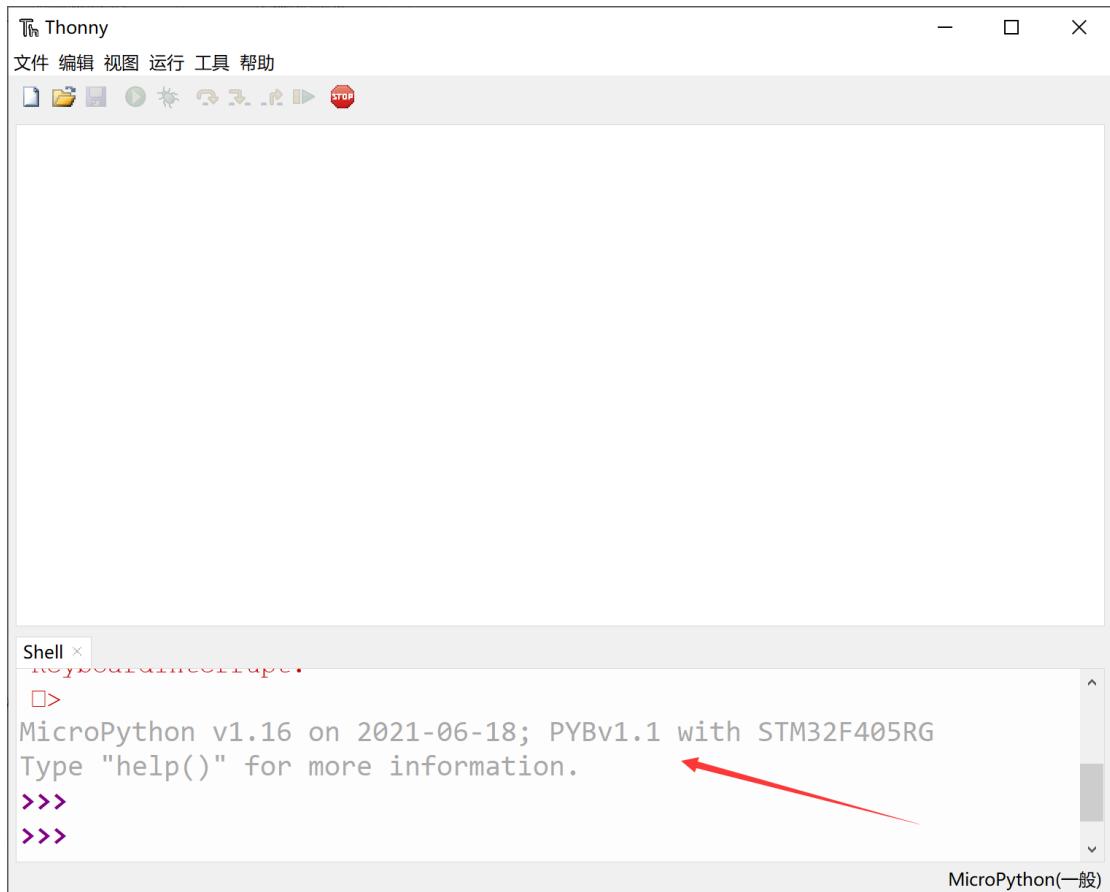


图 3-14 连接成功

接下来我们简单的测试一下代码，这里测试功能为主，更详细的代码讲解将在后面实验章节展开。在 Thonny 中点击打开资料包里面的例程文件：

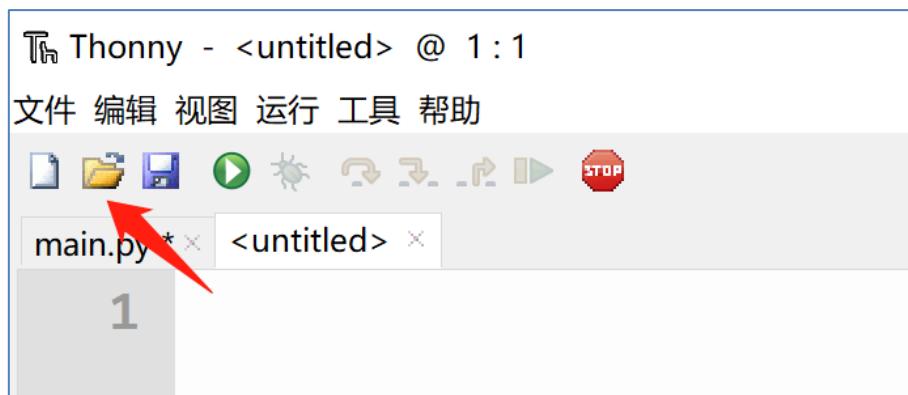


图 3-15

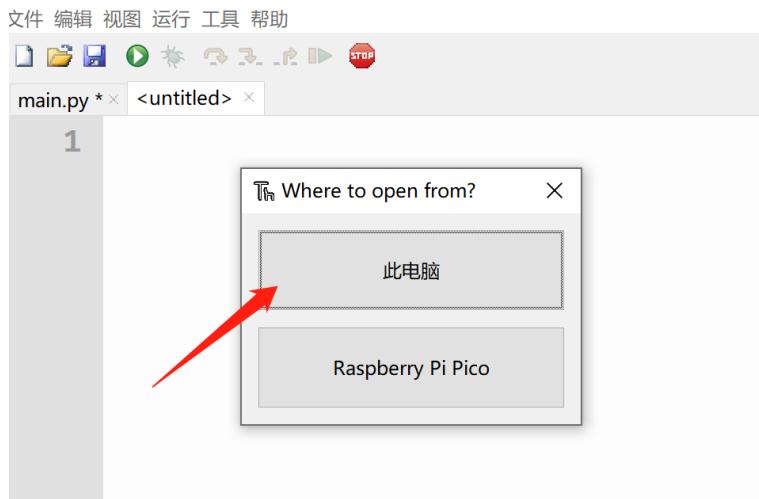


图 3-16 打开本地文件选择此电脑

选择资料包路径 零一科技（01Studio）MicroPython 开发套件（基于 pyBoard STM32F405 平台）配套资料\02-示例程序\1.基础实验\1.点亮第一个 LED 下的 py 文件。

A screenshot of the Thonny IDE interface. The title bar says "Thonny - D:\零一科技 (01Studio) MicroPython开发套件 (基于pyboard STM32F05平台) 配套资料\_202...". The code editor window is titled "main.py" and contains the following Python code:

```
1 """
2 实验名称: 点亮LED(4)蓝灯
3 版本: v1.0
4 日期: 2019.4
5 作者: 01Studio
6 """
7
8 from pyb import LED
9 LED(4).on()
```

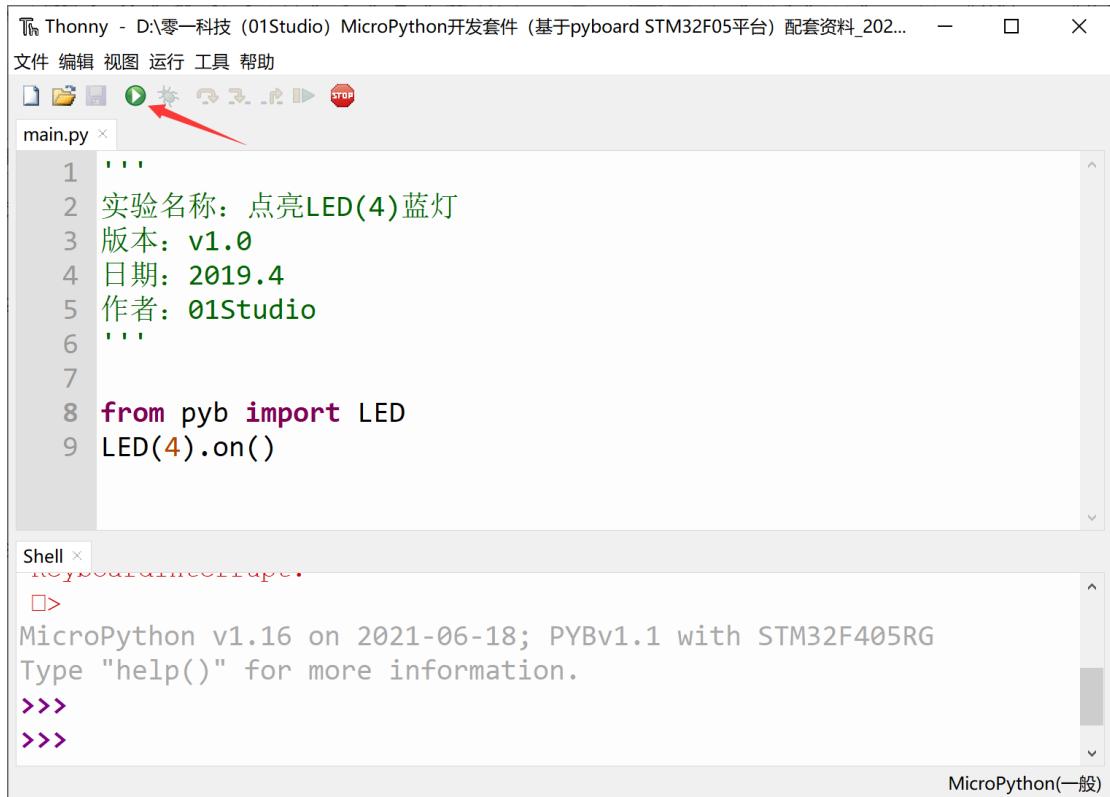
Below the code editor is a "Shell" tab with the following MicroPython session:

```
MicroPython v1.16 on 2021-06-18; PYBv1.1 with STM32F405RG
Type "help()" for more information.
>>>
>>>
```

The status bar at the bottom right says "MicroPython(一般)".

图 3-17 打开本地的 py 文件

然后点击绿色按钮“运行”：



The screenshot shows the Thonny IDE interface. At the top, there's a menu bar with File, Edit, View, Run, Tools, Help. Below the menu is a toolbar with icons for file operations and a green play button, which is highlighted with a red arrow. The main window contains a code editor with a file named 'main.py' containing the following code:

```
1  """
2  实验名称: 点亮LED(4)蓝灯
3  版本: v1.0
4  日期: 2019.4
5  作者: 01Studio
6  """
7
8 from pyb import LED
9 LED(4).on()
```

Below the code editor is a shell window titled 'Shell'. It shows the MicroPython environment setup:

```
MicroPython v1.16 on 2021-06-18; PYBv1.1 with STM32F405RG
Type "help()" for more information.
```

At the bottom right of the shell window, it says 'MicroPython(一般)'.

图 3-18

可以看到开发板的 LED 绿灯被点亮：

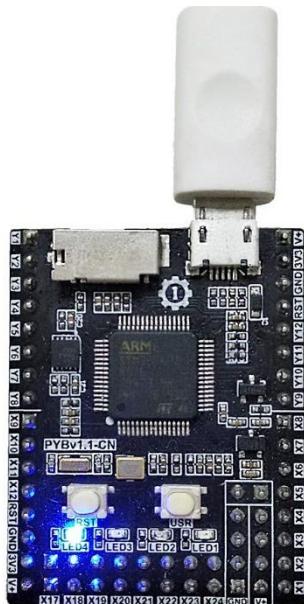


图 3-19 点亮 LED

### 3.1.2.3 REPL 串口交互调试

MicroPython 固件集成了交互解释器 REPL 【读取(Read)-运算(Eval)-输出(Print)-循环(Loop)】，开发者可以直接通过串口终端来调试开发板。

在 Thonny 中的 Shell 其实就是达芬奇的 REPL，通过串口交互的，Thonny 连接达芬奇后，我们在 Shell 里面输入 `print("Hello 01Studio!")`，按回车，可以看到打印出 Hello 01Studio 字符：



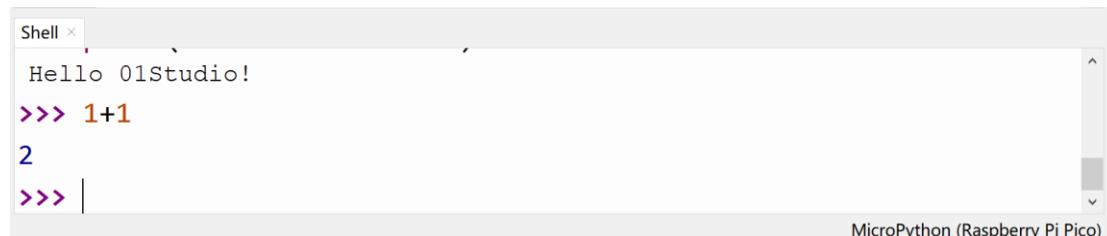
A screenshot of the Thonny IDE's Shell window. The window title is "Shell". The text area shows the following interaction:

```
Shell x
W1L111 KPY040
Type "help()" for more information.
>>> print("Hello 01Studio!")
Hello 01Studio!
>>> |
```

The output "Hello 01Studio!" is highlighted with a red arrow pointing to it. The status bar at the bottom right of the window says "MicroPython (Raspberry Pi Pico)".

图 3-20

再输入 `1+1`，按回车：



A screenshot of the Thonny IDE's Shell window. The window title is "Shell". The text area shows the following interaction:

```
Shell x
Hello 01studio!
>>> 1+1
2
>>> |
```

The output "2" is highlighted with a red arrow pointing to it. The status bar at the bottom right of the window says "MicroPython (Raspberry Pi Pico)".

图 3-21

接下来我们将上一节的三行代码逐行输入和逐行按回车，可以看到 LED 灯也被点亮：

```
from pyb import LED
LED(4).on()
```

```
Shell < Type "help()" for more information.  
>>>  
>>>  
>>> from pyb import LED  
>>> LED(4).on()  
>>>
```

MicroPython(一般)

图 3-22 逐行输入

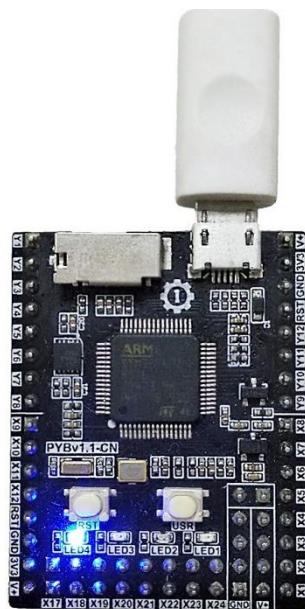


图 3-23 LED 被点亮

REPL 还有一个强大的功能就是打印错误的代码来调试程序，我们还是用上一节点亮 LED 的代码，将 `on` 改成 `onx`，然后运行代码，可以看到报错。指示出了错误的行数和错误原因，大大提高了编程调试的效率：

The screenshot shows the Thonny IDE interface. The top bar displays "Thonny - D:\零一科技 (01Studio) MicroPython开发套件 (基于pyboard STM32F05平台) 配套资料\_202...". The menu bar includes "文件" (File), "编辑" (Edit), "视图" (View), "运行" (Run), "工具" (Tools), and "帮助" (Help). Below the menu is a toolbar with icons for file operations and a stop button. The code editor window titled "main.py" contains the following Python code:

```
1 ...
2 实验名称: 点亮LED(4)蓝灯
3 版本: v1.0
4 日期: 2019.4
5 作者: 01Studio
6 ...
7
8 from pyb import LED
9 LED(4).onx()
```

A red arrow points to the line "LED(4).onx()", which is highlighted in orange. The bottom window is the "Shell" terminal, showing the following interaction:

```
>>> LED(4).on()
>>> %Run -c $EDITOR_CONTENT
Traceback (most recent call last):
  File "<stdin>", line 9, in <module>
AttributeError: 'LED' object has no attribute 'onx'
```

The error message "AttributeError: 'LED' object has no attribute 'onx'" is highlighted with a red box.

图 3-24 错误打印

### REPL 终端常用按键:

Ctrl + C : 打断正在运行的程序 (特别是含 While True: 的代码);

Ctrl + D : 软件复位开发板。

### 3.1.2.4 文件系统

pyBoard 开发板自带了 1 个 U 盘，可以用于存放代码、图片等文件。传输的方式有 2 种，第一种是直接复制到 U 盘。



图 3-25 开发板 U 盘

第二种方式是通过 Thonny IDE。IDE 连接开发板后，点击菜单栏 视图--文件：

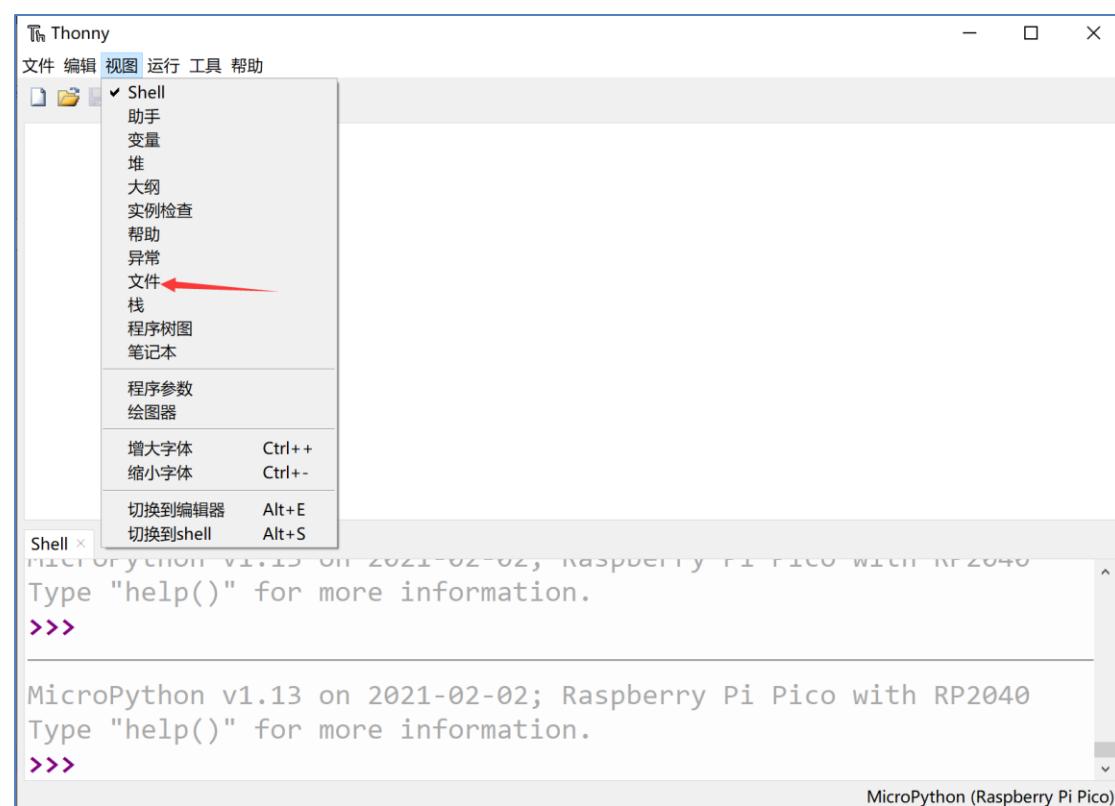


图 3-26

可以看到左边出现本地和开发板的实时文件浏览窗口：

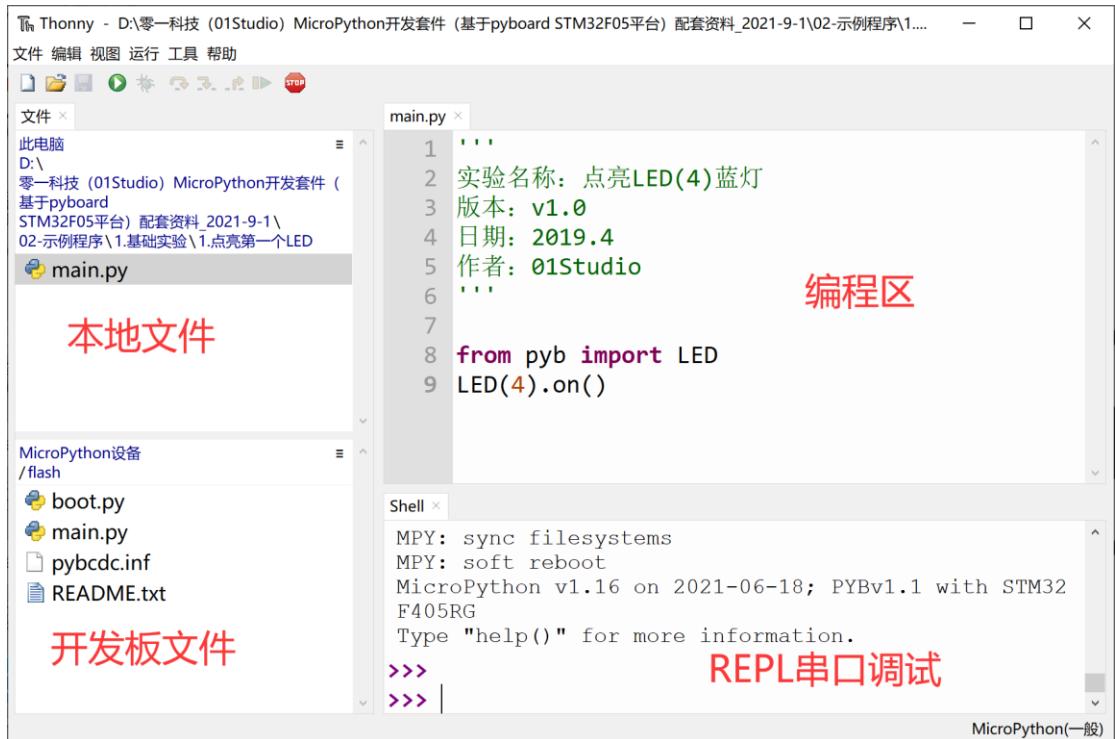


图 3-27

点击文件右键，然后点上载到/，即可将指定的 py 文件发送到开发板。

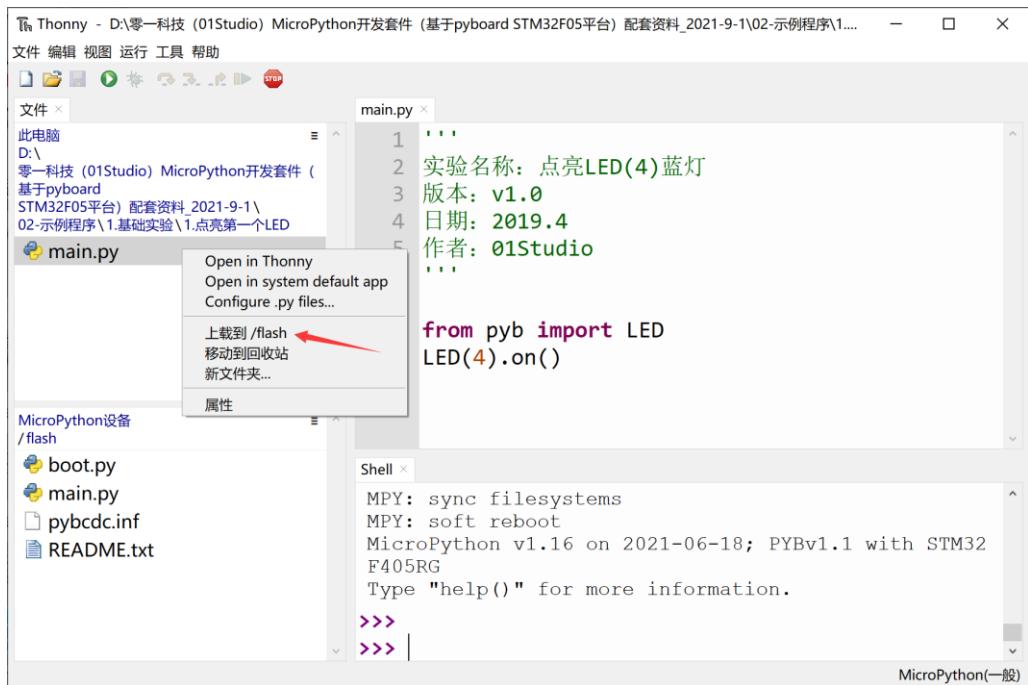


图 3-28

同样可以在这里点击文件右键，执行删除等相关功能。使用起来更方便。

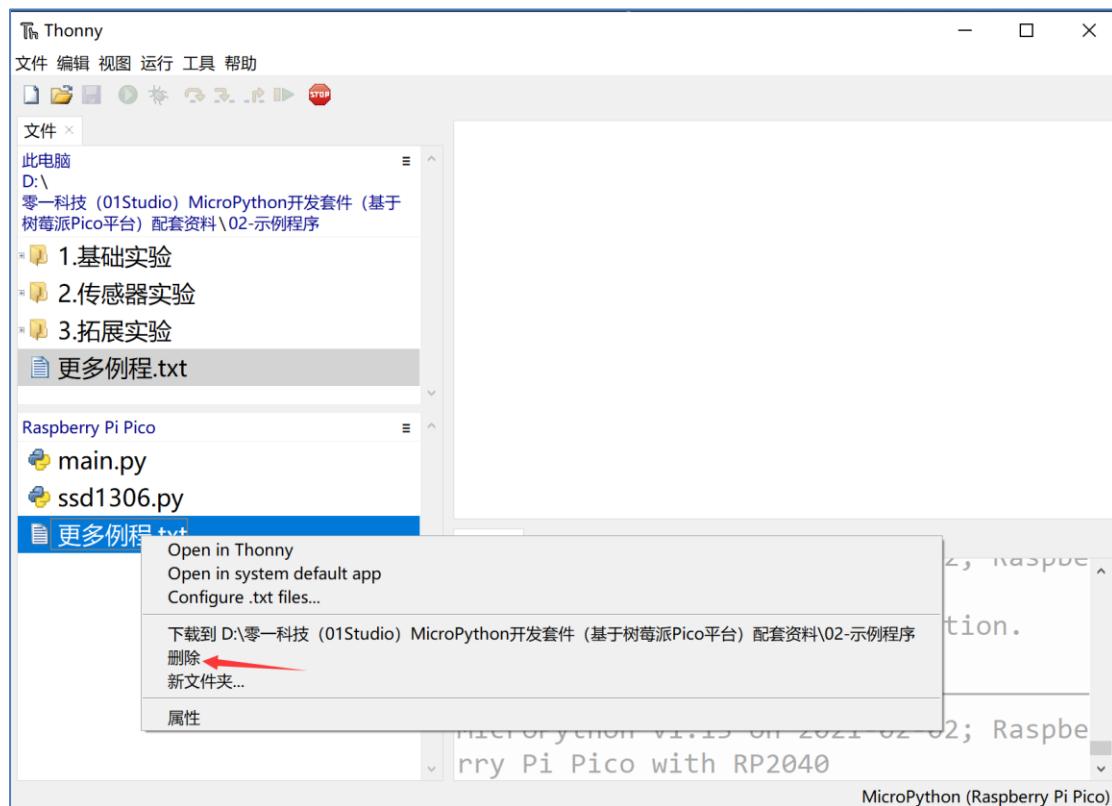


图 3-29

### 3.1.2.5 SD 卡

SD 卡插入后开发板自动识别，并以 SD 卡的 boot.py 和 main.py 为启动文件，屏蔽自带 Flash 里面文件。**其它使用方式和功能完全不变！**

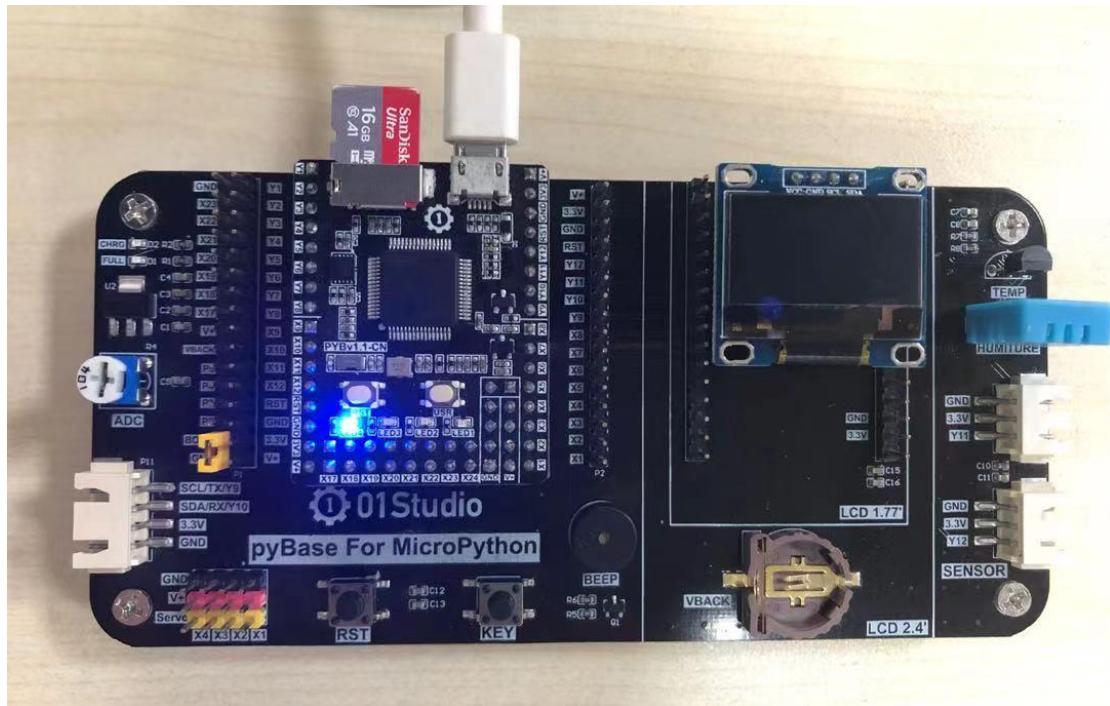


图 3-30

这里使用 16G MicroSD 卡，可以看到接入电脑后自动识别出对应容量的文件系统。



图 3-31 自动以 SD 容量为文件系统

### 3.1.2.6 恢复出厂设置

当 pyboard 出现异常时，可以通过以下方式修改启动顺序或者恢复出厂设置，方法如下：

按着 pyBoard 上的 USER 键不放，再按一下 RST 键，LED 灯会持续交替闪烁，当闪烁达到你想要的模式时候，松开 user 键，LED 灯会快速闪烁，板子会接着重新启动。

模式 1：只有绿灯亮，正常模式：先启动``boot.py``然后``main.py``。

模式 2：只有橙色灯亮，安全模式：启动时候不运行任何脚本。(仅 1 次有效)

模式 3：绿灯和橙灯同时亮，文件系统重置：文件系统恢复出厂状态，然后以安全模式启动。

如果你的文件系统损坏了，可以通过模式 3 启动修复它。

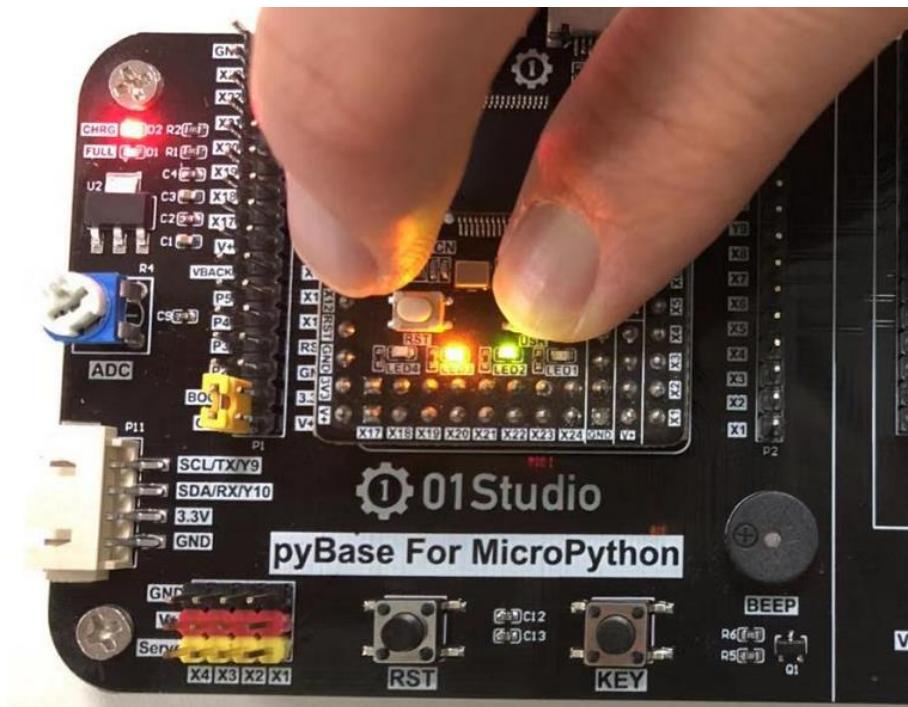


图 3-32 模式 3 启动

重新启动后红色 LED 灯会亮一会儿，待其熄灭后再使用。这时候打开 U 盘文件系统，可见是剩下 4 个原始文件：



图 3-33 恢复出厂后的文件系统

打开 main.py，里面是空白的内容。

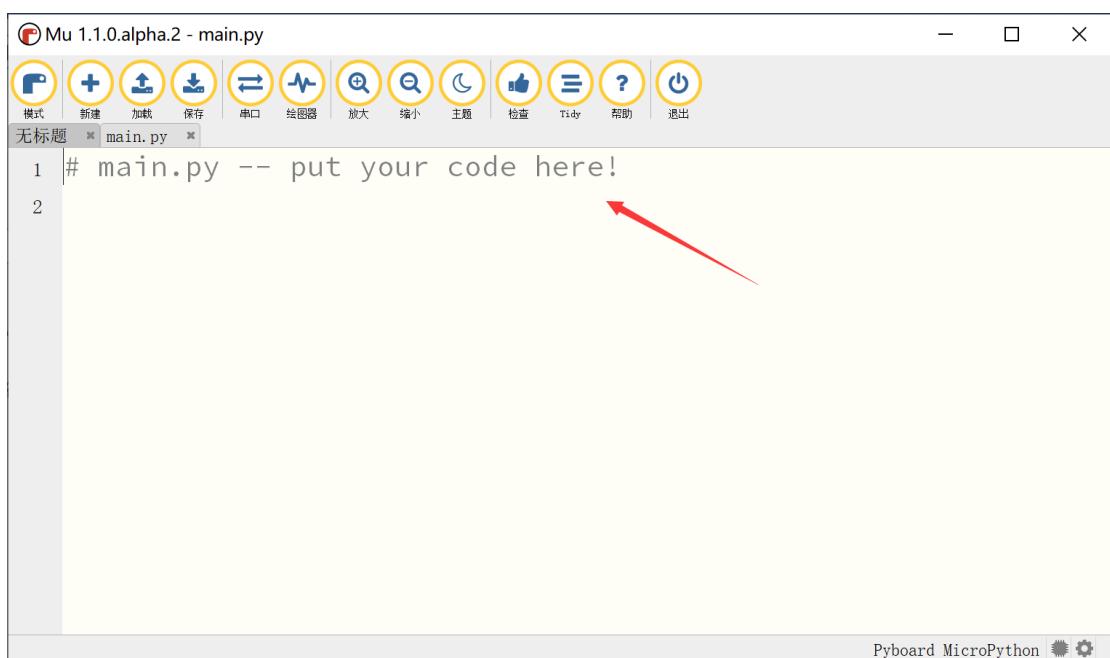


图 3-34 恢复出厂状态的 main.py 文件

### 3.1.2.7 固件更新

当 pyboard 上的固件意外丢失或者我们希望升级到较新版本固件时候，就需重新烧录 pyboard 的固件。Pyboard 上面是一个 M4 单片机，所以这个操作相当于给 M4 单片机重新烧录程序。

我们采用 DFU 的烧写方式，DFU 烧写方式优势是不需要 ST-LINK 一类仿真器，只需要安装一个软件即可。我们打开 MicroPython 开发套件配套资料\开发工具\Windows\固件更新工具\DfuSe Demo V3.0.5 目录下的 DfuSe 安装软件。DFU 软件安装完成后相当于电脑所需要的驱动也安装了。

将开发板上的默认连接的 BOOT-GND 跳线帽连接到 3.3V，让 pyboard 进入 DFU 烧写模式，如下图所示：

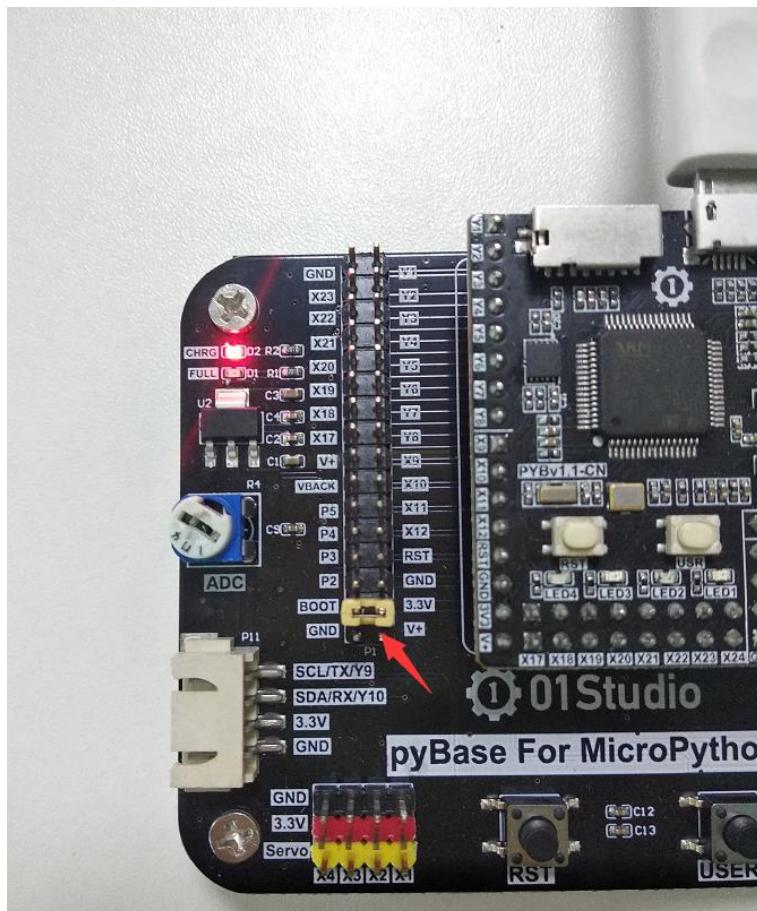


图 3-35 BOOT 跟 3.3V 通过跳线帽连起来

如果只有 pyBoard，则按下图方式用跳线帽或杜邦线将 DFU 与 3.3V 连接。



图 3-36 DFU 模式

打开刚刚安装的 DFU 软件，按下开发板 RST 复位键，可以见到 DFU 软件成功识别出 ST 的芯片。

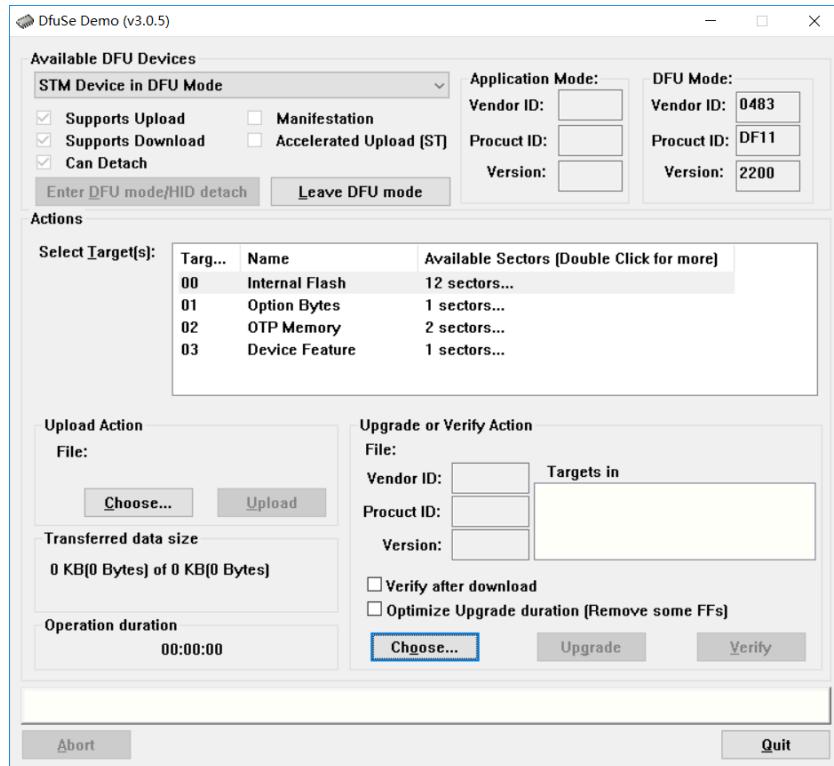


图 3-37

点击 choose，选择要升级的固件版本，在 MicroPython 开发板配套资料\相关固件\pyboard v1.1 目录下选择要升级的固件版本。

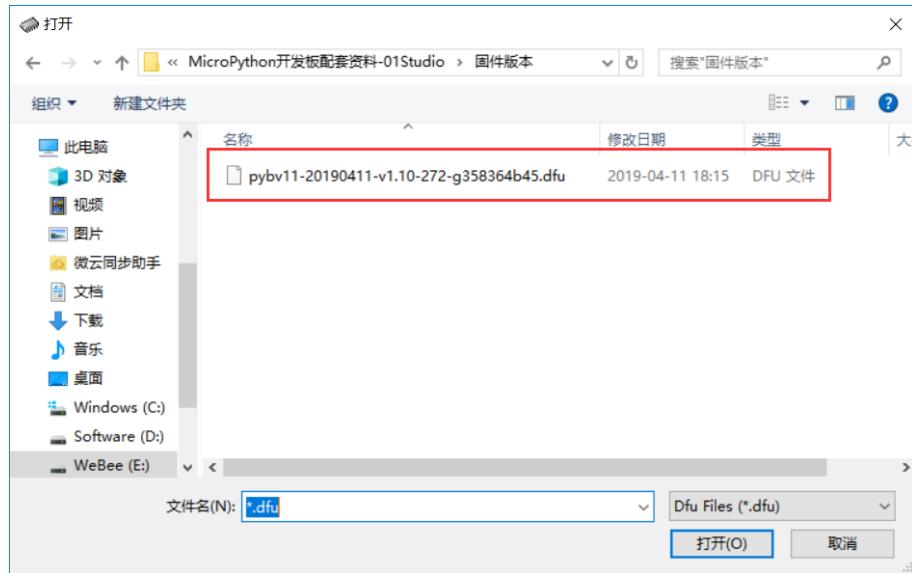


图 3-38 选择固件

勾选配置参数，点击 Upgrade 开始烧录。

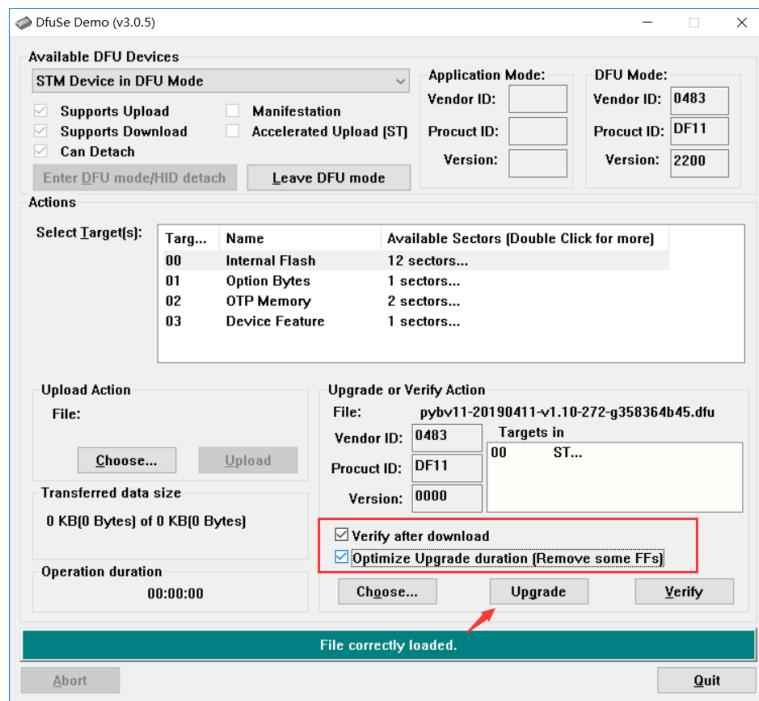


图 3-39

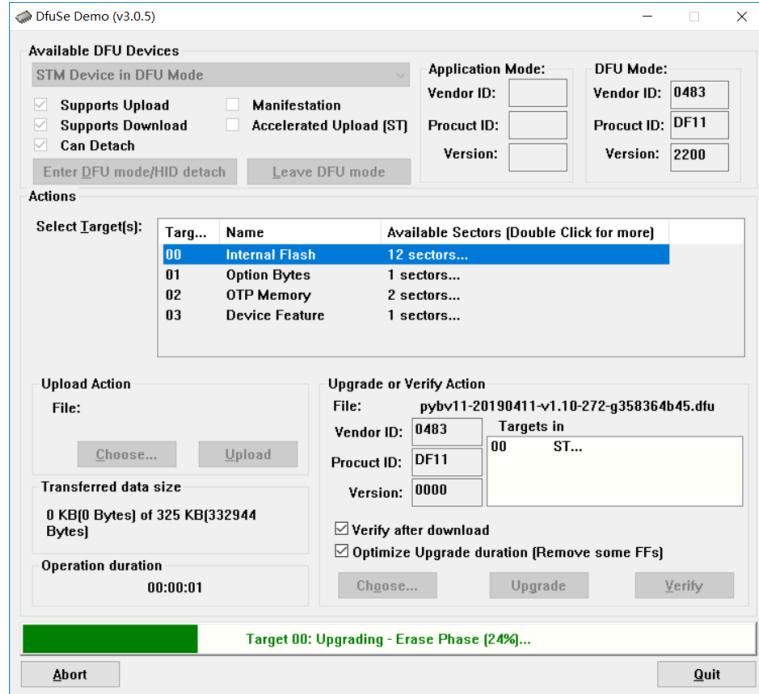


图 3-40 正在烧录

烧录完成后如下图所示：

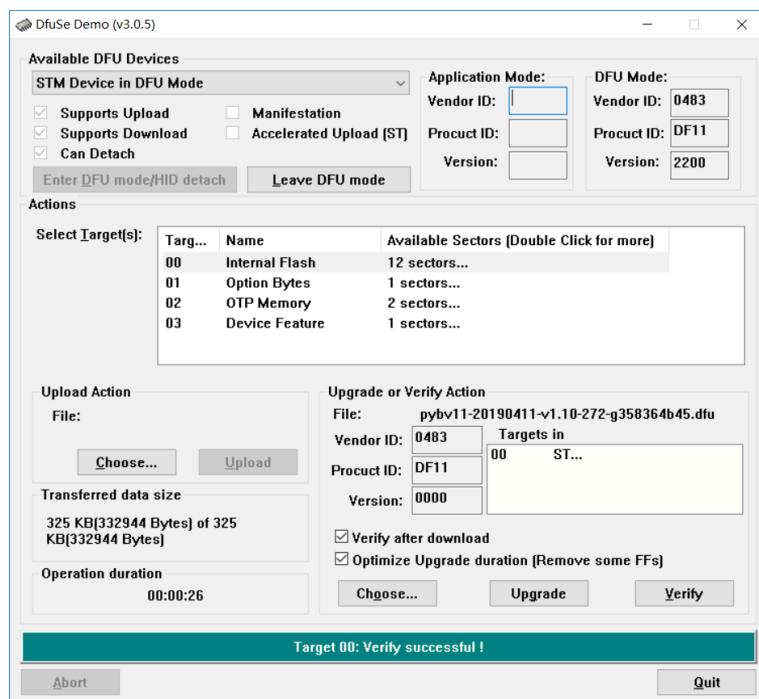


图 3-41 烧录完成

烧录完成后将 BOOT 跳线帽拔掉或者连接到开发板上的 GND 备用，重新按下复位键。可以见到系统检测到 pyboard 的 U 盘文件系统。说明固件更新完成。

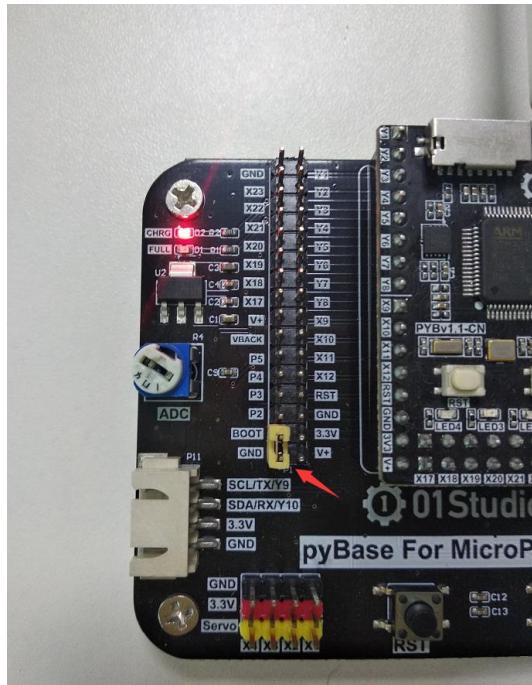


图 3-42



图 3-43

### 3.1.3 附录

#### 3.1.3.1 编程软件 pyCharm 安装和使用

前面提到的 Mu 编程软件针对 MicroPython 开发比较友好。但考虑到有些用户可能是从事 python 相关工作，习惯了大型的编程软件，因此这里我们补充一项常用到的一款编程软件 pyCharm 的安装和使用教程。

pyCharm 的社区版是免费的，功能强大。很多 python 的开发者也是使用它具体安装方法如下：

在 <https://www.jetbrains.com/pycharm/download> 下载 Windows 社区版。

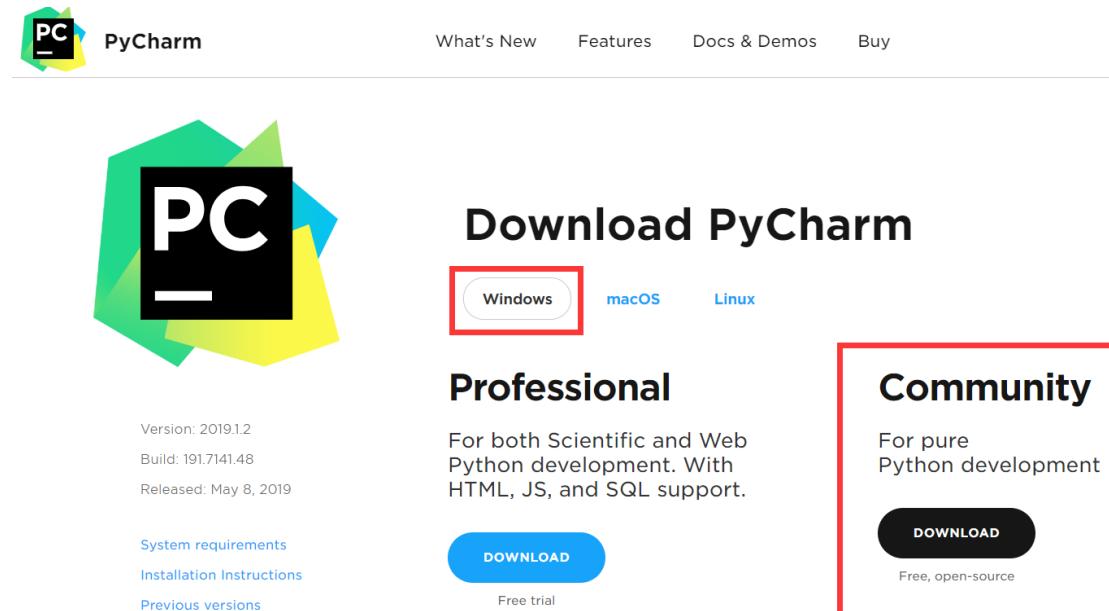


图 3-44 pyCharm 官方下载

下载完成后打开安装文件，选择自己喜欢的英文路径安装，出现配置界面，选择如下：根据自己的电脑配置选择 32bit 或者 64bit，环境变量 PATH 勾上，然后点击 NEXT，直至安装完成后重启电脑。

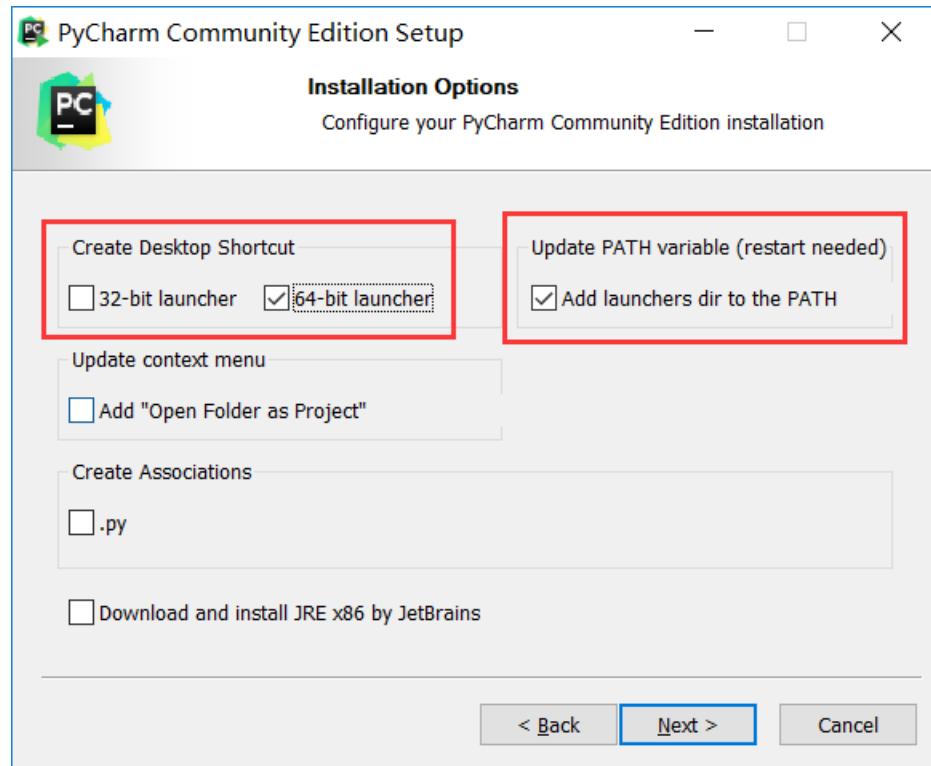


图 3-45 pyCharm 安装选项

安装完成重启后，我们打开 pyCharm，首次需要设置 pyCharm 的环境变量，环境变量这个词语看起来高大上，其实就是告诉 pyCharm 在编译 python 代码的时候，调用我们之前安装的 python3 应用作为解析器。否则是无法编译运行的。简单来说就是告诉 pyCharm 本电脑 python3 的安装路径，方法如下：

打开 pyCharm，点击 Configure—settings:

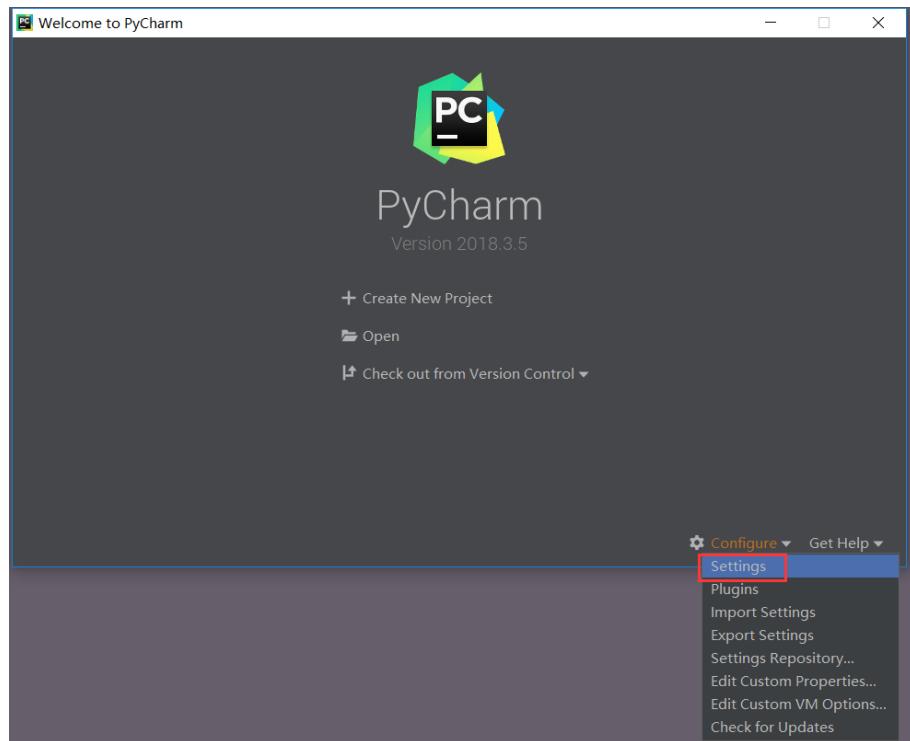


图 3-46

选择 Project Interpreter，点击右上角设置图标，选择 add:

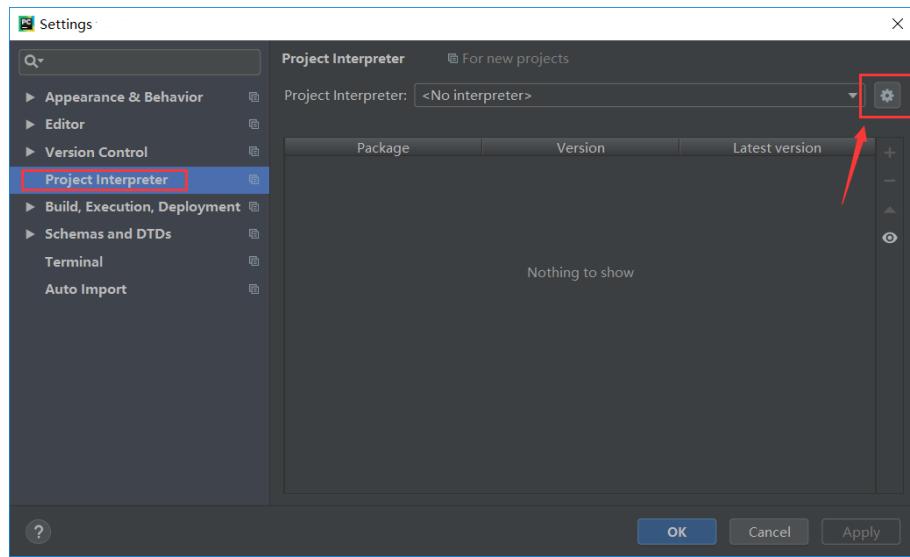


图 3-47 解析器配置

在弹出的新界面选择 Existing environment, 路径选择之前安装 python3 的路径, 选上 python.exe。另外注意勾上 make available to all projects, 以后就不用重复配置。

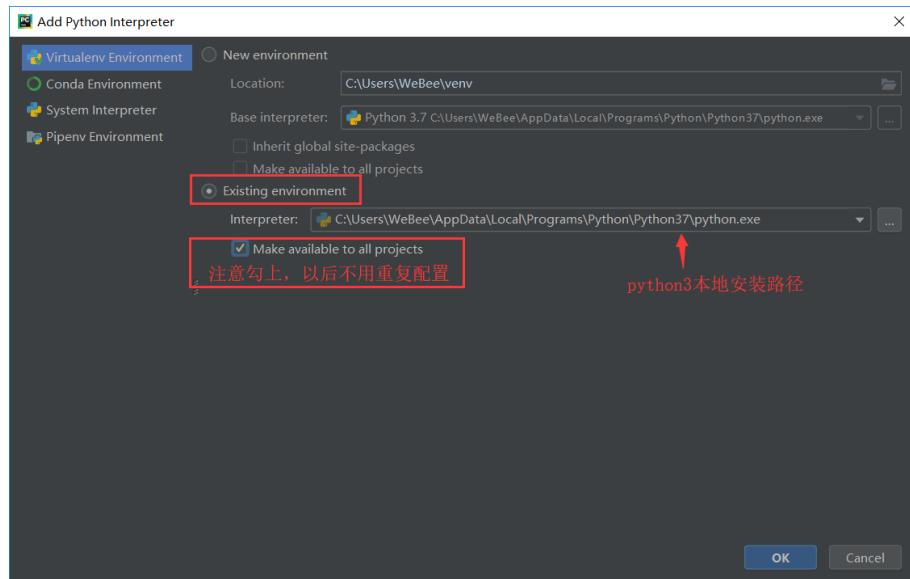


图 3-48 配置

关于 python3 安装路径, 可以参考以下方法查找, 在程序搜索 “python.exe” (注意这个名称要一字不漏对上), 点击右键-打开文件所在位置, 即可找到 python3 安装路径。



图 3-49

将路径复制到 pyCharm 配置即可。

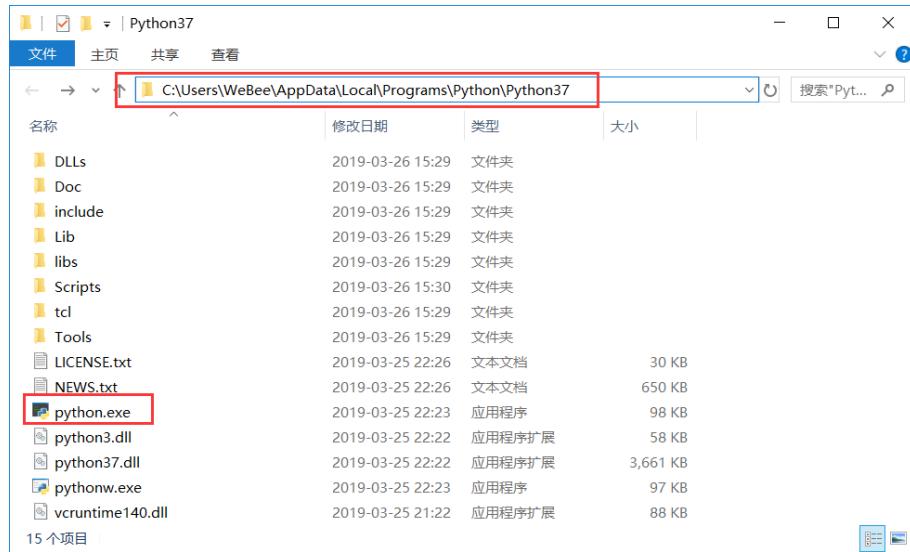


图 3-50 python3 安装路径

完成以上配置后，我们新建一个工程和文档测试一下。打开 pyCharm—Create New Project

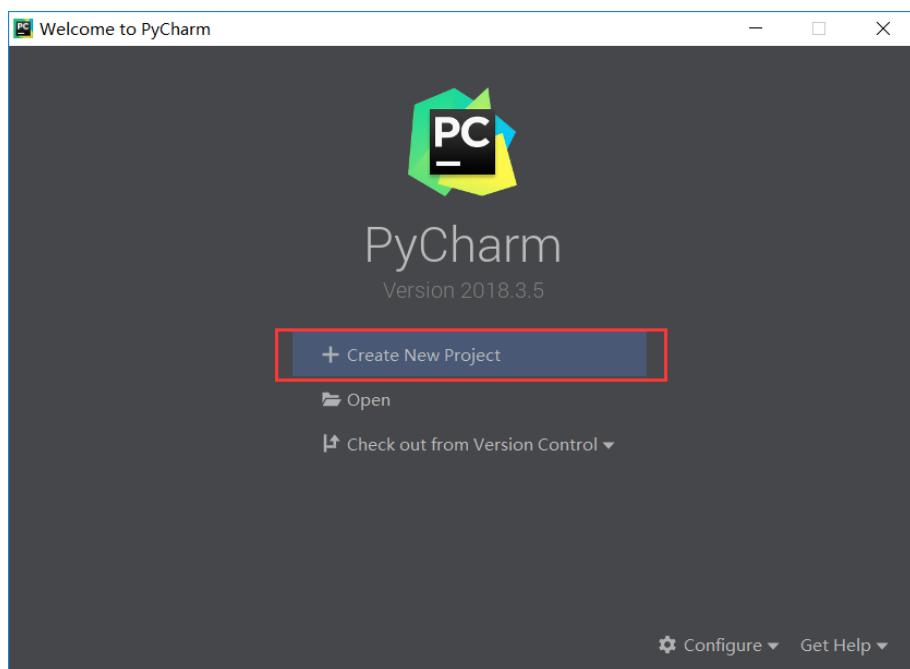


图 3-51 新建工程

在桌面或者自己喜欢的路径下新建一个名为 test 的工程

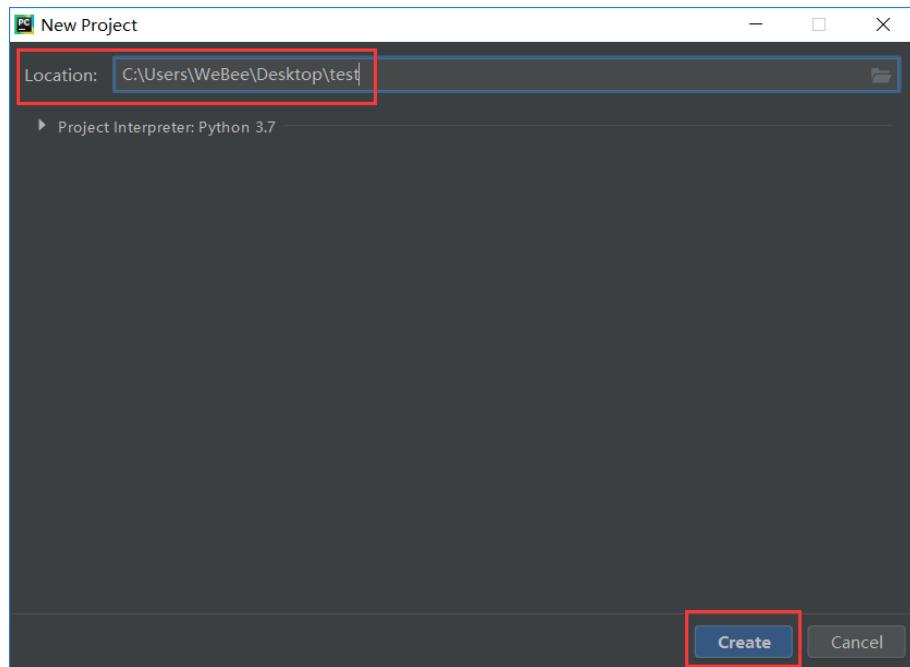


图 3-52 新建名为 test 的工程

接下来在 test 工程下新建 python 文件，在 project 下空白区域点击右键— new--Python File.

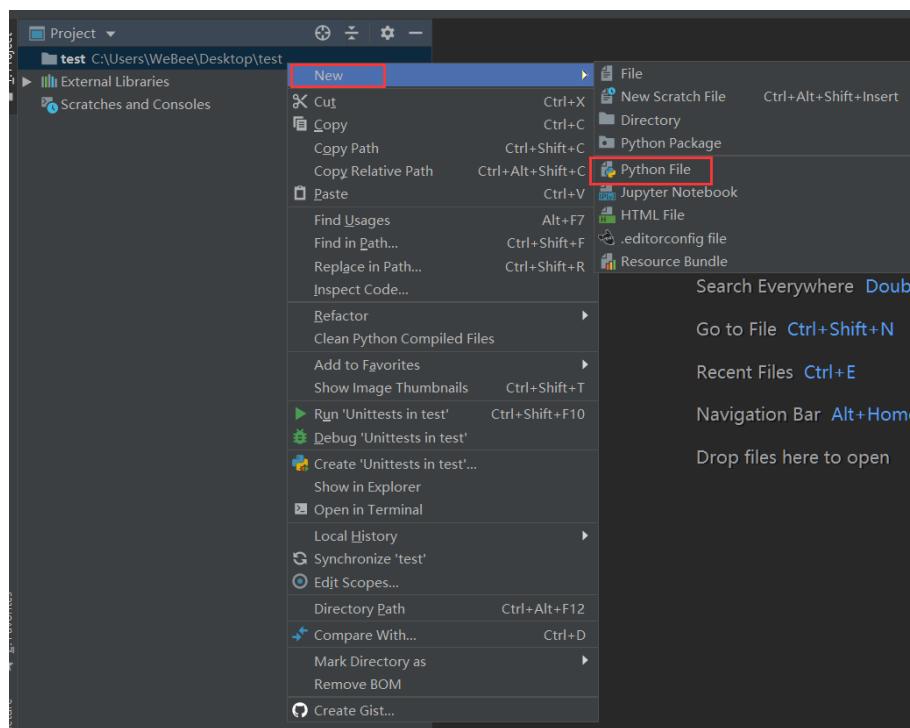


图 3-53 添加 python 文件

输入自己想要的 python file 名称，我们这里输入 test

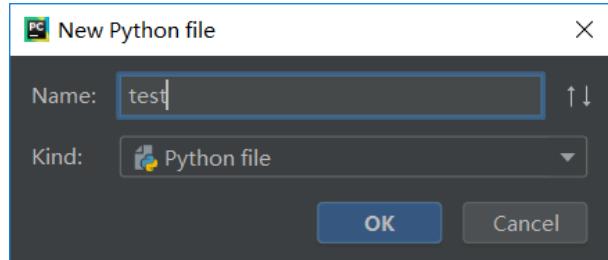


图 3-54

在编辑框输入 print("Hello O1Studio!")

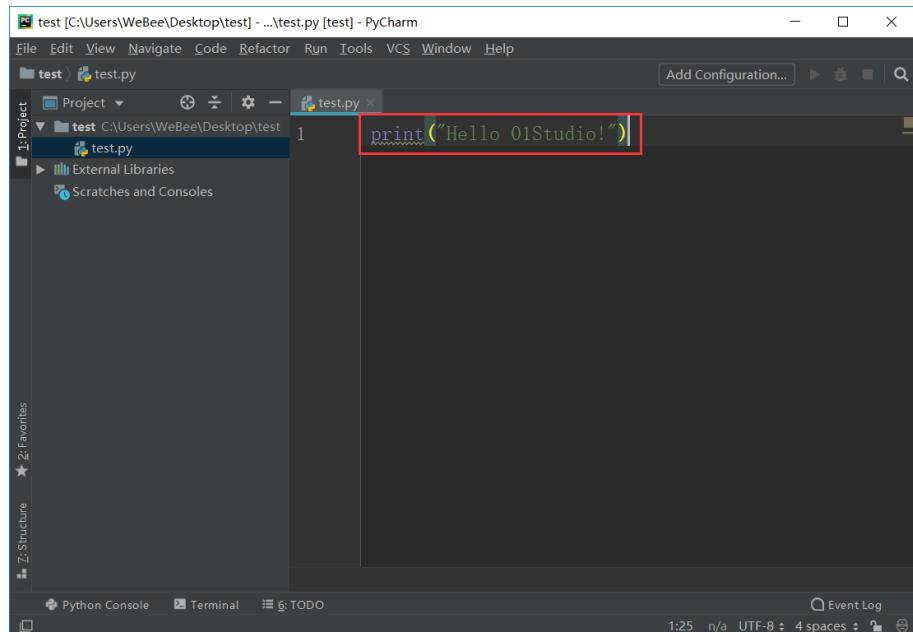


图 3-55

默认字体可能有点小，我们可以调整一下，点击 File—settings—Editor—Font，将 Size 调到 20。

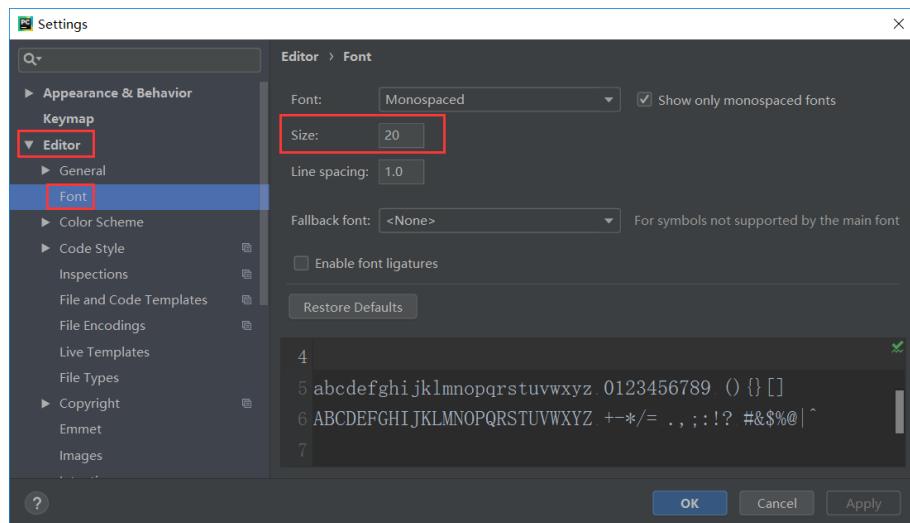


图 3-56 调整编辑框字体大小

点击 Run，选择 test (首次需要选择)

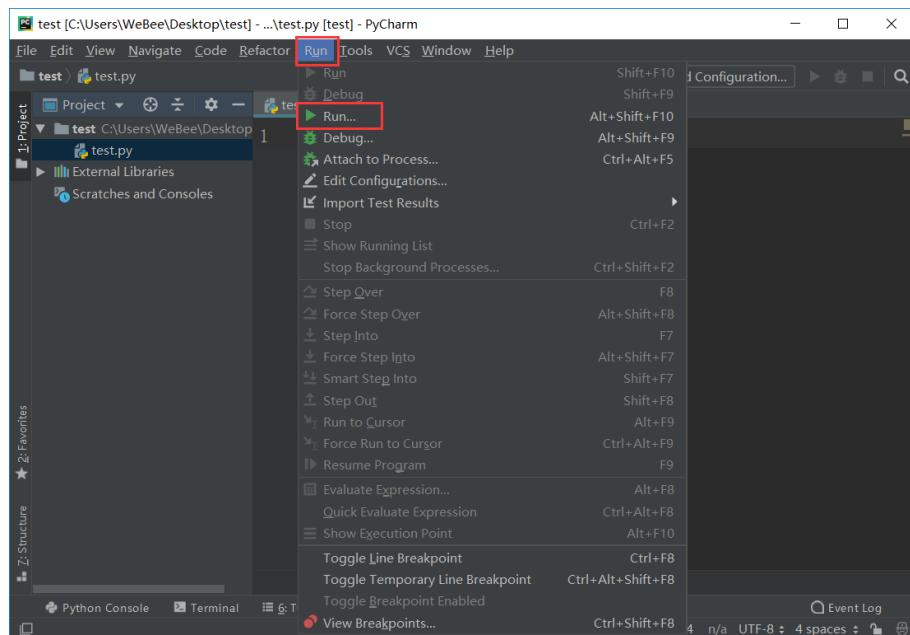


图 3-57 运行

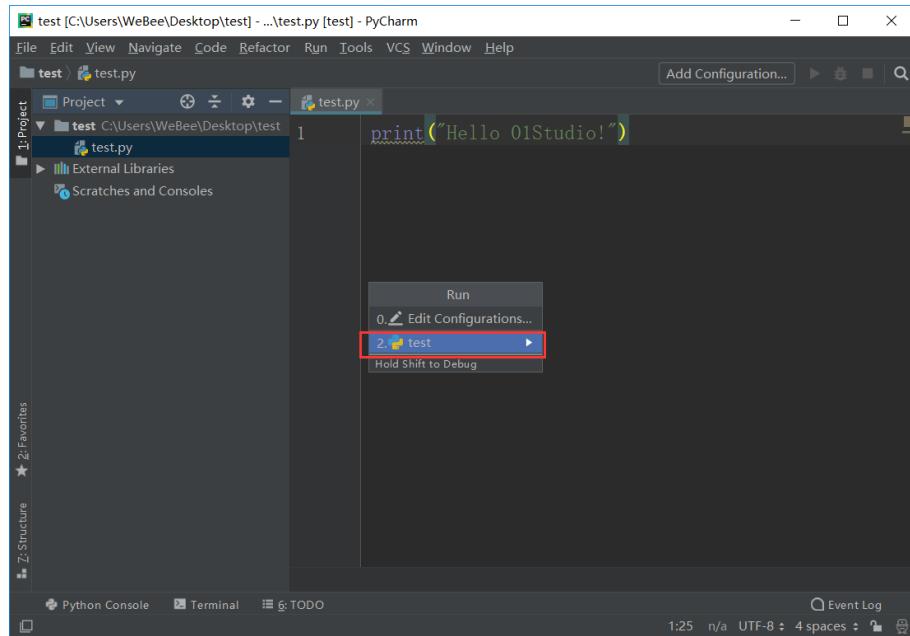


图 3-58 选择要运行的文件

我们看到输出框打印出来了 Hello 01Studio，说明运行成功。

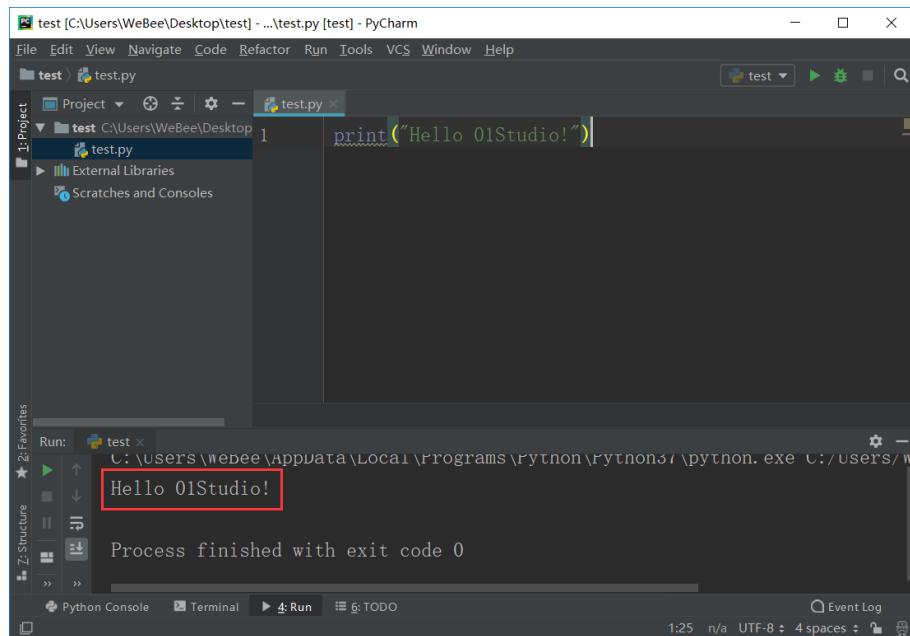


图 3-59 运行成功

至此，python 编译器安装和配置成功了。需要注意的是，在打开别人 python 项目的时候，有可能出现环境变量不匹配情况，重新按照以上方法设置一下即可。现在，你甚至可以用 pyCharm 来开发相关 python 应用了。

## 3.2 基于 Mac OS

还记得当年乔布斯从大信封里面拿出 macbook 么，苹果和微软在 PC 时代走了两个极端，乔布斯主导下的苹果要求软硬一体化，封闭，务求最佳的用户体验；而微软则主打开发，让操作系统兼容不同的 PC 厂家，使得其 windows 一度占据了 90% 的市场份额。从商业角度，盖茨毫无疑问是赢家，就个人而言，我更认可乔布斯的方式，现在的 win10 和微软自家的笔记本，某程度上都有一体化的思想。我用了半天的 Macbook 和 MAC 系统，如果不是要做硬件发，基本上可以放弃十多年的 windows 了。

### 3.2.1 安装开发软件 Thonny

在 <https://thonny.org/> 选择 Mac 版本下载：



图 3-60

Thonny IDE 在 MAC OS 环境下的安装和使用方法和 Windows 一样，具体请参考： [3.1.1 安装开发软件 Thonny IDE](#) 章节内容，这里不再重复。

### 3.2.2 开发套件使用

由于 Thonny IDE 基本可以满足开发板的编程、调试功能，因此直接参考 Windows 章节 [3.1.2 开发套件使用](#) 内容即可。

### 3.3 基于 Linux

Linux 系统为大多工程师热衷的开源操作系统，一般内部都集成了 python3 和 IDE，用户直接使用即可。本节教程同样适用于树莓派的 Linux 系统。

#### 3.3.1 安装开发软件 Thonny

在 <https://thonny.org/> 下载 Linux 版本的 Thonny IDE，按提示指令安装即可：



图 3-61

如果你实验树莓派开发板开发，它出厂自带 Thonny IDE，无需额外安装。

#### 3.3.2 开发套件使用

由于 Thonny IDE 基本可以满足开发板的编程、调试功能，因此直接参考 Windows 章节 [3.1.2 开发套件使用](#) 内容即可。

## 第4章 基础实验

MicroPython 更强调的是针对应用的学习，强大的底层库函数让我们可以直接关心功能的实现，也就是说我们只要理解和熟练相关的函数用法，就可以很好的玩转 MicroPython。它让我们可以做到不关心硬件和底层原理（当然有兴趣和能力的小伙伴可以深入研究）而直接跑起硬件。

基础实验是针对一些简单的实验学习讲解，可以让我们更快和更直接感受到 MicroPython 针对嵌入式开发的强大之处，我后面的学习和开发夯实基础。

请先看实验讲解格式预览，每一节我们都会以以下形式讲解，图文并茂，力求达到快速理解的作用：

- (1) **前言**: 简单介绍这个实验;
- (2) **实验平台**: 实验所用到的开发板、配件和接线说明;
- (3) **实验目的**: 本实验要实现的功能;
- (4) **实验讲解**: 对函数、代码、编程方法以及实验的详细讲解;
- (5) **实验结果**: 记录程序下载到开发板上的图片示例;
- (6) **总结**: 对实验进行总结。

## 4.1 点亮第一个 LED 灯

- **前言:**

相信大部分人开始学习嵌入式单片机编程都会从点亮 LED 开始，我们 MicroPython 的学习也不例外，通过点亮第一个 LED 能让你对编译环境和程序架构有一定的认识，为以后的学习和更大型的程序打下基础，增加信心。

- **实验平台:**

pyBoard v1.1-CN 或者 MicroPython 开发套件。

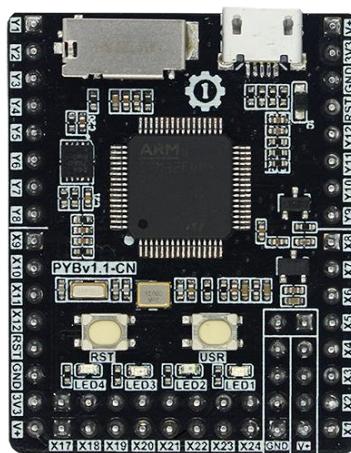


图 4-1 pyBoard v1.1-CN

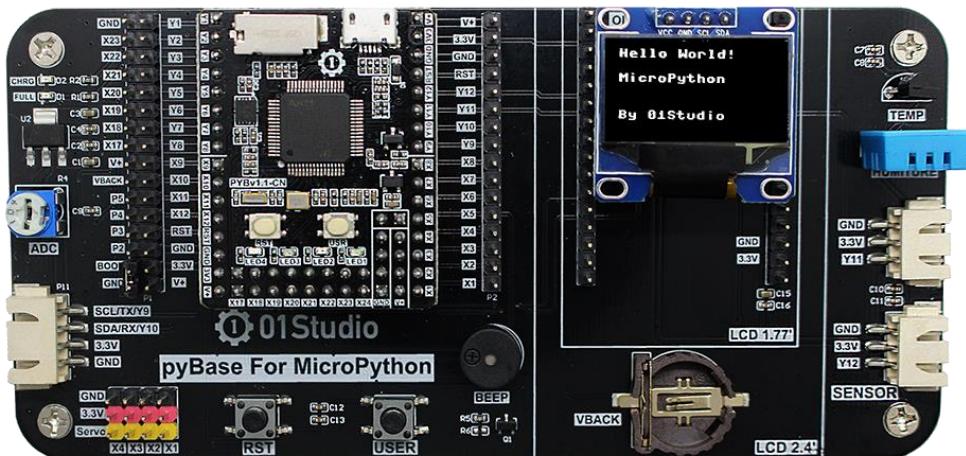


图 4-2 MicroPython 开发套件

- **实验目的:**

点亮 LED4 (蓝灯)。

- **实验讲解:**

pyboard 上总共有 4 个 LED，分别是 LED1(红色)、LED2(绿色)、LED3(黄色)、LED4(蓝色)；控制 LED 使用到 LED 对象。LED 的构造函数和使用方法如下表所示：

构造函数
<b>pyb.LED(id)</b>
构建 LED 对象。其中 id 是编号，1-4 分别对应 LED1 至 LED4。
使用方法
<b>LED.off()</b>
关闭 LED。
<b>LED.on()</b>
打开 LED。
<b>LED.toggle()</b>
打开/关闭 LED 状态切换（反转）
<b>LED.intensity([value])</b>
亮度调节，value 的值范围是 0-255。

表 4-1 LED 对象

上表对 MicroPython 的 LED 对象做了详细的说明，pyb 是大模块，LED 是 pyb 下面的其中一个小模块，在 python 编程里有两种方式引用相关模块：

方式 1 是：import pyb，然后通过 pyb.LED 来操作；

方式 2 是：from pyb import LED，意思是直接从 pyb 中引入 LED 模块，然后直接通过 LED 来操作。显然方式 2 会显得更直观和方便，本实验也是使用方式 2 来编程。代码编写流程如下：

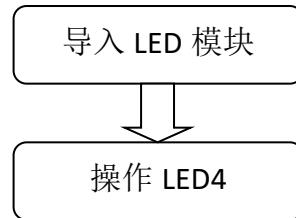


图 4-3 代码编写流程

以下是本实验示例代码：

```
...  
实验名称: 点亮 LED(4)蓝灯  
版本: v1.0  
日期: 2019.4  
作者: 01Studio  
...  
  
from pyb import LED    #从 pyb 导入 LED 模块  
LED(4).on()    #打开 LED4
```

代码有 2 种测试方式，第一种是直接在 IDE 里面运行 **py** 文件，这时候代码不会保存到 **flash**，掉电失效。

第二种是将 **py** 文件拷贝到 U 盘文件系统，然后复位开发板运行。这时候代码是存放在 **flash**。复位后能自动运行。

### ● 实验结果：

运行上面代码，便能看到实验结果。看到 LED4 蓝灯被成功点亮。

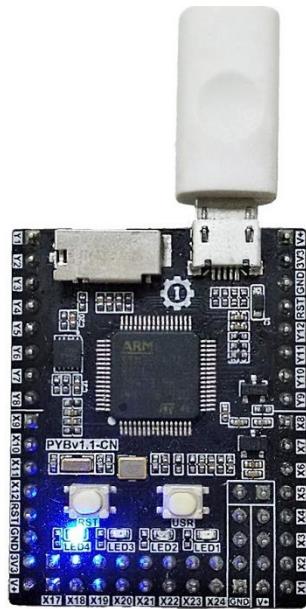


图 4-4 基于 pyBoard v1.1-CN

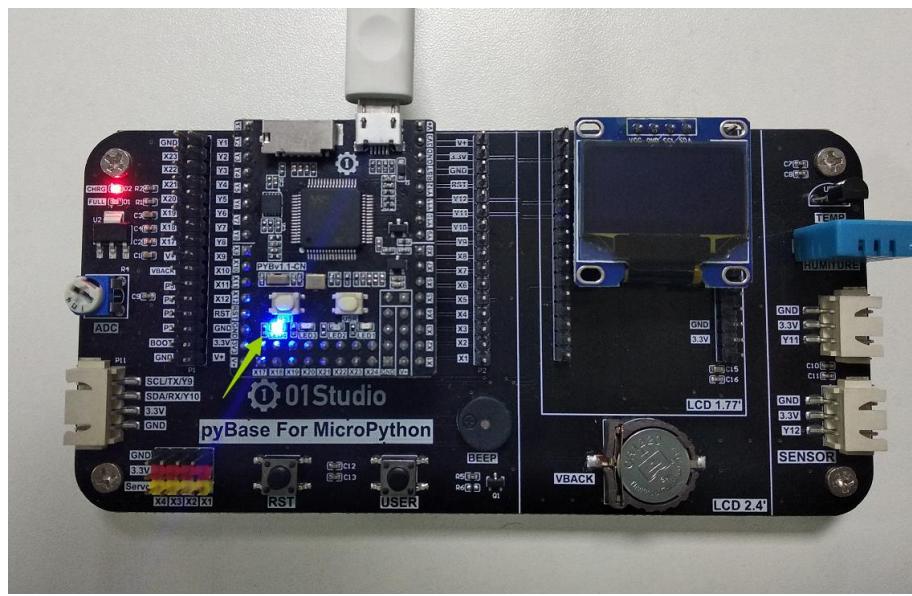


图 4-5 基于 MicroPython 开发套件

### ● 总结:

从第一个实验我们可以看到，使用 MicroPython 来开发关键是要学会构造函数和其使用方法，便可完成对相关对象的操作，在强大的模块函数支持下，实验只用了简单的两行代码便实现了点亮 LED 灯。

## 4.2 流水灯

### ● 前言：

通过上一节点亮 LED 灯的学习，我们已经对 micropython 的编程有了初步的了解，这一节来做一个功能稍微复杂一点的实验，流水灯。流水灯也叫跑马灯，也就是让几个 LED 来回亮灭，达到好像流水的效果。也是单片机开发学习的典型例子。

### ● 实验平台：

pyBoard v1.1-CN 或者 MicroPython 开发套件。

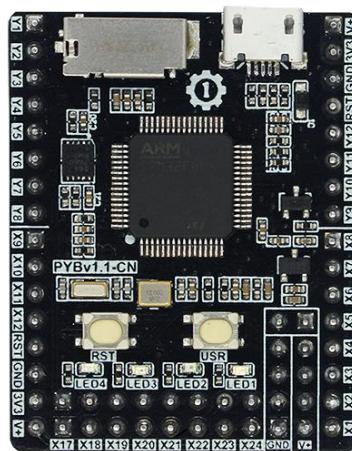


图 4-6 pyBoard v1.1-CN

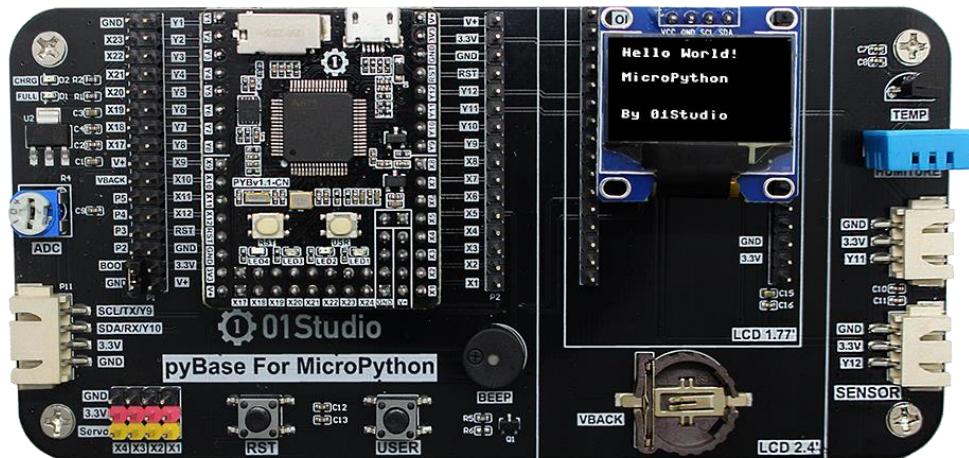


图 4-7 MicroPython 开发套件

### ● 实验目的:

流水灯。让 LED2-LED4 循环亮灭，达到像流水一样的效果。

### ● 实验讲解:

pyboard 上总共有 4 个 LED，分别是 LED1(红色)、LED2(绿色)、LED3(黄色)、LED4(蓝色)；控制 LED 使用到 LED 对象。上一章节我们已经学习过 LED 点亮，这里要实现固定时间来亮灭，需要用到 delay 延时的函数。具体如下：

LED 的构造函数和使用方法如下表所示：

构造函数	说明
<code>pyb.LED(id)</code>	其中 <code>id</code> 是编号，1-4 分别对应 LED1 至 LED4
使用方法	说明
<code>LED.off()</code>	关闭 LED
<code>LED.on()</code>	打开 LED
<code>LED.toggle()</code>	打开/关闭 LED 状态切换（反转）
<code>LED.intensity([value])</code>	亮度调节， <code>value</code> 的值范围是 0-255.

表 4-2 LED 对象

Delay 延时函数和使用方法如下表所示：

构造函数	说明
<code>pyb.delay(ms)</code>	毫秒级延时
<code>pyb.udelay(us)</code>	微秒级延时
使用方法	说明
<code>pyb.delay(1000)</code>	延时 1000ms
<code>pyb.udelay(1000)</code>	延时 1000us

表 4-3 延时对象

知道了函数的使用方法后，我们可以简单的梳理一下流程，首先导入 LED 和 Delay 模块，程序开始先让 LED2-LED4 灭掉，开启循环，依次点亮每个 LED，

延时 1 秒，关闭 LED。流程如下：

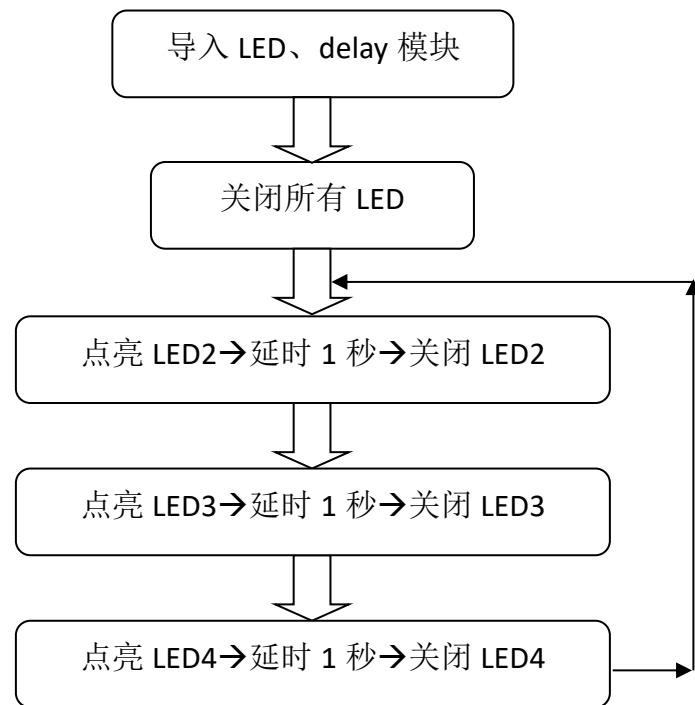


图 4-8 代码编写流程

以下是本实验参考程序代码：

```
...  
实验名称: 流水灯  
版本: v1.0  
日期: 2019.4  
作者: 01Studio  
...  
  
from pyb import LED, delay  #从 pyb 导入 LED 模块  
  
#关闭全部 LED
```

```
LED(2).off()
LED(3).off()
LED(4).off()

#while True 表示一直循环
while True:
    #LED2 亮 1 秒
    LED(2).on()
    pyb.delay(1000) #延时 1000ms, 即 1 秒
    LED(2).off()

    #LED3 亮 1 秒
    LED(3).on()
    pyb.delay(1000)
    LED(3).off()

    #LED4 亮 1 秒
    LED(4).on()
    pyb.delay(1000)
    LED(4).off()
```

上述代码没错是完整的按照编程思路来编写，但可以见到有很多格式相似的地方，这显得代码非常冗余。我们可以通过 `for` 函数来编写程序，由于是对 LED2/3/4 的操作，因此我们可以用 `for i in range(2,5):` 语句来修改。参考代码如下：

实验名称：流水灯

```
版本: v2.0  
日期: 2019.4  
作者: 01Studio  
...  
  
from pyb import LED,delay    #从 pyb 导入 LED 模块  
  
# 相当于 for i in [2, 3, 4], LED(i).off()执行 3 次, 分别是 LED 2, 3, 4  
for i in range(2,5):  
    LED(i).off()  
  
  
while True:  
    #使用 for 循环  
    for i in range(2,5):  
        LED(i).on()  
        delay(1000) #延时 1000 毫秒, 即 1 秒  
        LED(i).off()
```

修改后的代码一下子变得简约美观。Python 作为高级语言，使用起来非常灵活，常常能通过几行代码就可以表达复杂的逻辑。我们在编程的过程中应该多思考如何优化代码，才能让自己的编程水平得到较快的提升。

（需要注意的是如果在 **main.py** 文件里使用了类似 **while True:** 的死循环函数，**pyboard** 的 **REPL** 功能将失效。）

### ● 实验结果：

LED2、LED3、LED4 循环点亮，实现流水效果。

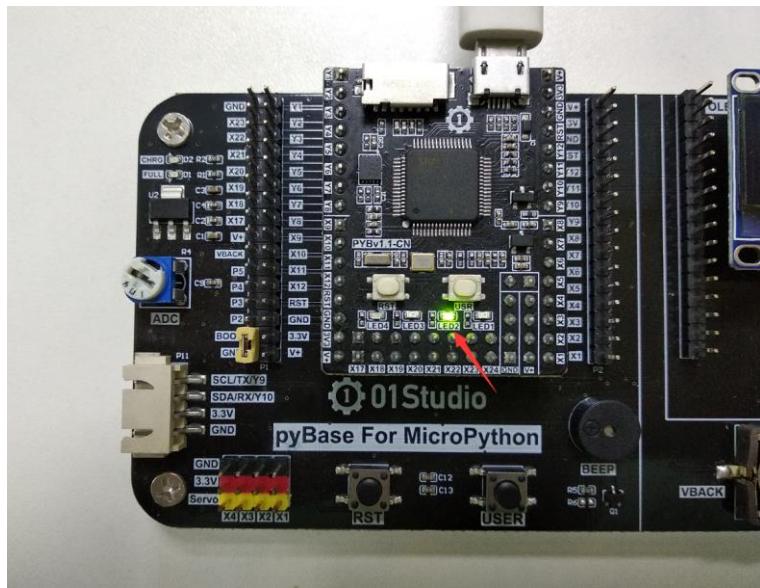


图 4-9

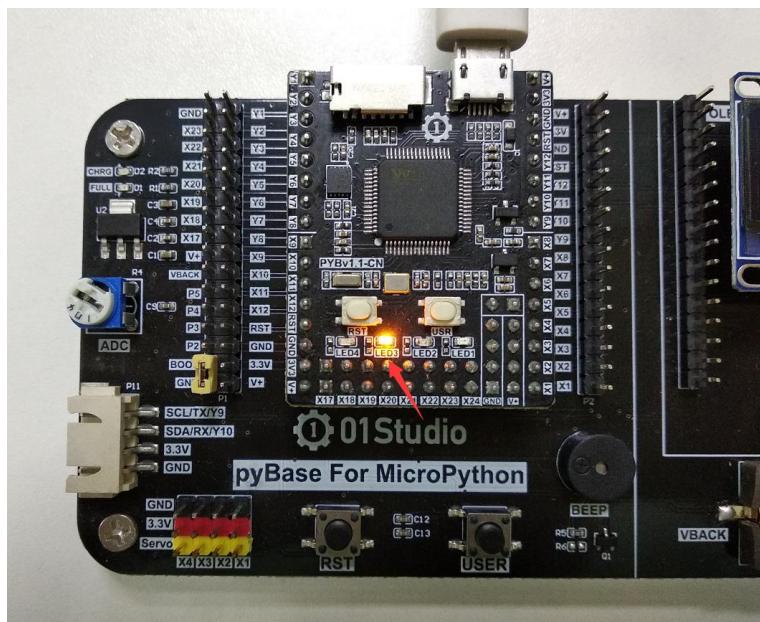


图 4-10

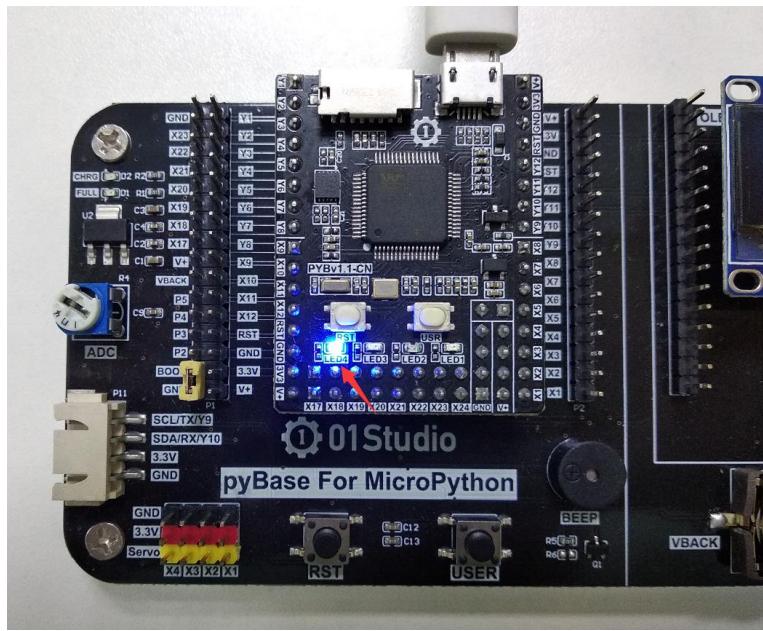


图 4-11

### ● 总结：

本节实验相对于第一节相比增加了延时函数的应用，可见 MicroPython 编程功能增加就是各类函数的叠加，当开发复杂功能时候，因为底层函数已经封装好，所以面向应用的开发使用起来非常方便。除此之外我们也体验了 Python 代码的简洁和高效，使得程序更好的阅读。

## 4.3 按键

- **前言：**

按键是最简单也最常见的输入设备，很多产品都离不开按键，包括早期的iphone，今天我们就来学习一下如何使用 MicroPython 来编写按键程序。有了按键输入功能，我们就可以做很多好玩的东西了。

- **实验平台：**

pyBoard v1.1-CN 或者 MicroPython 开发套件。

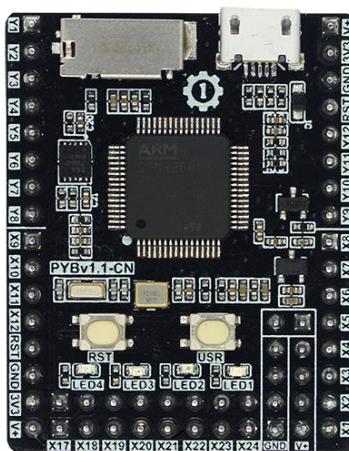


图 4-12 pyBoard v1.1-CN

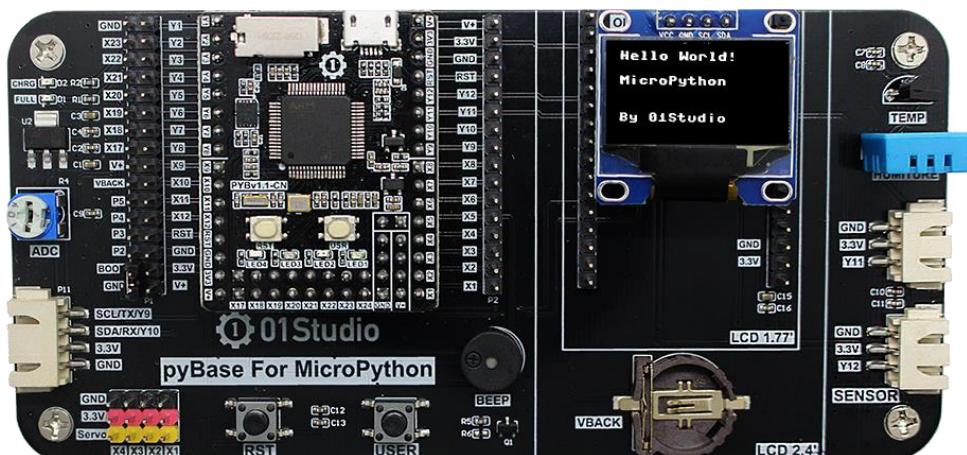


图 4-13 MicroPython 开发套件

- **实验目的:**

使用按键功能，通过检测按键被按下后，改变 LED(4)蓝灯的亮灭状态。

- **实验讲解:**

pyBoard 上有 2 个按键， RST 和 USER， RST 顾名思义是复位用的，所以真正自带可以用的就只有 1 个按键 USER。

让我们先来搞清楚 micropython 里面按键的构造函数和使用方法。

构造函数	说明
<code>pyb.Switch()</code>	<code>Switch</code> 代表了唯一的按键 <code>USER</code>
使用方法	说明
<code>Switch.value()</code>	读取按键状态，按下返回 <code>True</code> ，松开返回 <code>False</code>
<code>Switch.callback(fun)</code>	当按键被按下时候执行函数 <code>fun</code> 。

表 4-4 按键对象说明

跟之前的实验一样，首先导入 LED 和 Switch 模块，先定义函数 fun1()用来处理 LED(4)反转。通过 callback 指令使按键被按下时候执行 fun1()函数，流程如下：

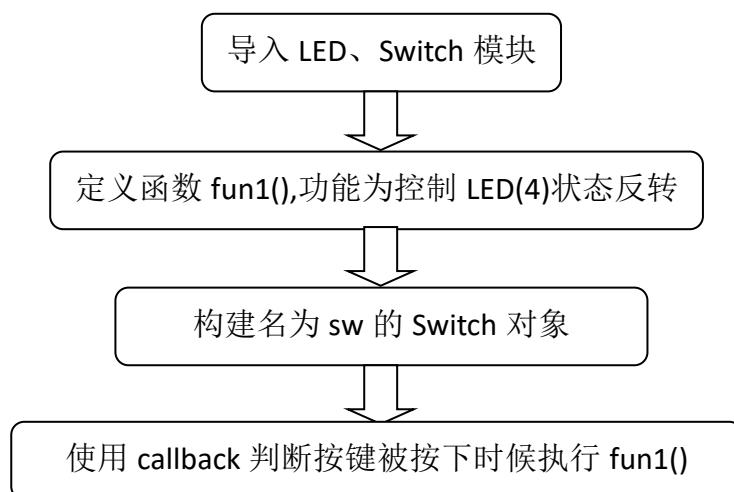


图 4-14 代码编写流程

参考程序代码如下：

```
...  
实验名称: 按键  
版本: v1.0  
日期: 2019.4  
作者: 01Studio  
...  
  
from pyb import LED,Switch  
  
def fun1():  
    LED(4).toggle()  
  
sw = Switch()      #定义按键对象名字为 sw  
sw.callback(fun1) #当按键被按下时, 执行函数 fun1(), 即 LED(4)状态反转
```

上述代码按照编写流程来实现，实际上我们还可以再优化一下。这里的函数 `fun1()` 要实现功能比较简单，我们可以使用 `lambda` 匿名函数表达式来优化代码（`lambda` 用法在 Python 基础知识“[2.4 函数](#)”章节有介绍）。优化后代码如下：

```
...  
实验名称: 按键  
版本: v2.0  
日期: 2019.4  
作者: 01Studio  
...
```

```
from pyb import LED,Switch

sw = Switch() #定义按键对象名字为 sw

sw.callback(lambda:LED(4).toggle()) #当按键被按下时， LED(4)状态反转
```

我们看到只需要三行代码就实现了我们想要的功能。

### ● 实验结果：

每次按下 pyBoard 上面的 USER 按键，可看到 LED (4) 蓝灯的状态相应改变。

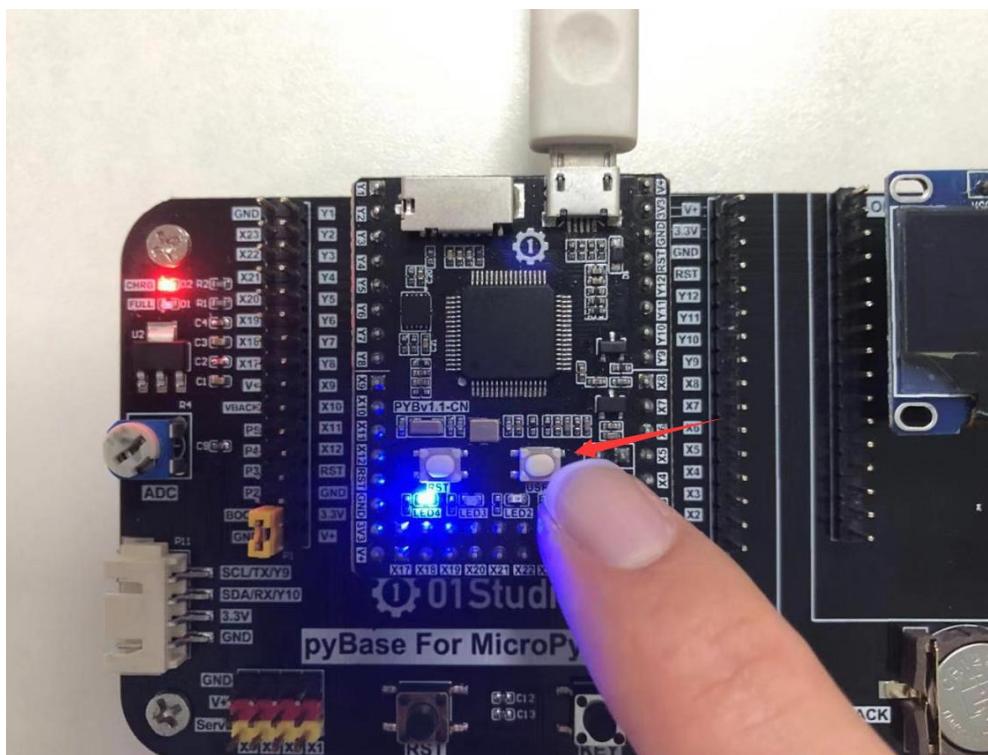


图 4-15 按键实验

### ● 总结：

按键作为我们学习的第一个输入设备，有了输入设备我们就可以跟硬件做人机交互了，这对后面的学习非常有意义。按键在 MicroPython 下开发也显得很简单，我们同样可以根据自己的理解和经验来优化代码。



## 4.4 GPIO

### ● 前言：

我们能看到 pyboard 和 MicroPython 开发套件上引出了非常多的引脚和 GPIO 口（General Purpose Input/Output，通用输入输出口），前两节的 LED 和按键实验背后原理都是使用 GPIO 来实现的，只是被提前封装好了。今天我们来做一下 GPIO 口的实验。

### ● 实验平台：

pyBoard v1.1-CN 或者 MicroPython 开发套件。

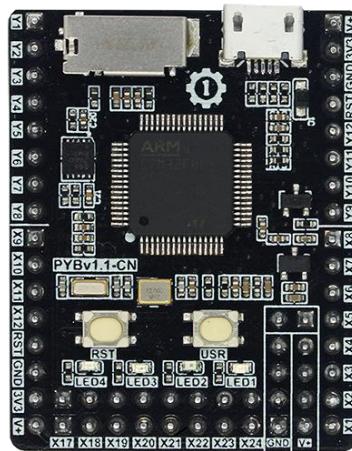


图 4-16 pyBoard v1.1-CN

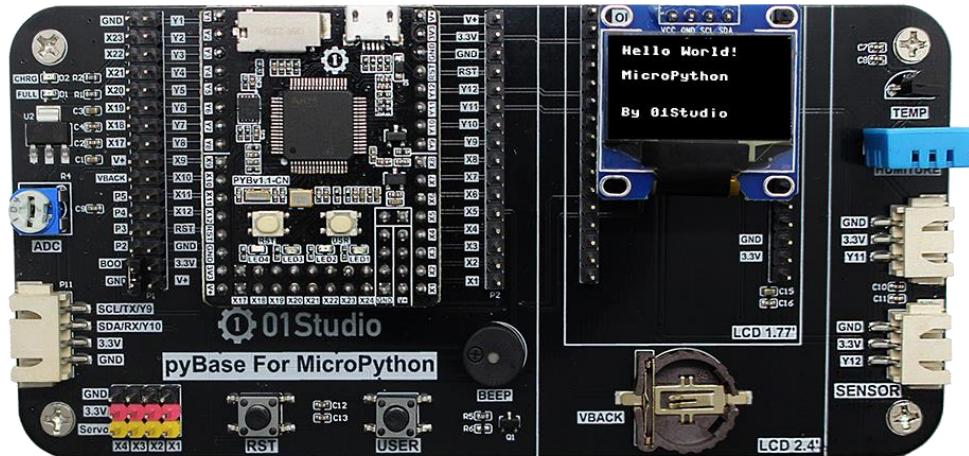


图 4-17 MicroPython 开发套件

- **实验目的:**

学会使用微处理器的 GPIO 的输入/输出，使用 GPIO 方式来控制 LED 和按键。

- **实验讲解:**

基本上 pyboard 的每个 IO 口都可以配置成特定的 GPIO 方式来进行应用。我们先来了解一下 GPIO 的构造函数和使用方法：

构造函数	说明
<code>pyb.Pin(id, mode , pull=Pin.PULL_NONE)</code>	<b>【id】</b> Pin 脚，如 X1,Y1 <b>【mode】</b> <code>Pin.IN</code> : 输入 <code>Pin.OUT_PP</code> : 输出，带推挽 <b>【pull】</b> <code>Pin.PULL_NONE</code> -没有上下拉电阻； <code>Pin.PULL_UP</code> -启用上拉电阻； <code>Pin.PULL_DOWN</code> -启用下拉电阻；
使用方法	说明
<code>Pin.high()</code>	引脚输出高电平
<code>Pin.low()</code>	引脚输出低电平
<code>Pin.value()</code>	在输入模式下获取引脚电平：返回 1 或者 0

表 4-5 GPIO 对象

可见 GPIO 对象使用非常简单，我们将 LED (4) 即 “P2” 引脚配置成输出，将 USER 按键即 “X17” 引脚配置成输入，实现当检测到按键被按下时候点亮 LED (4)，松开时关闭 LED (4)。代码编写流程如下：

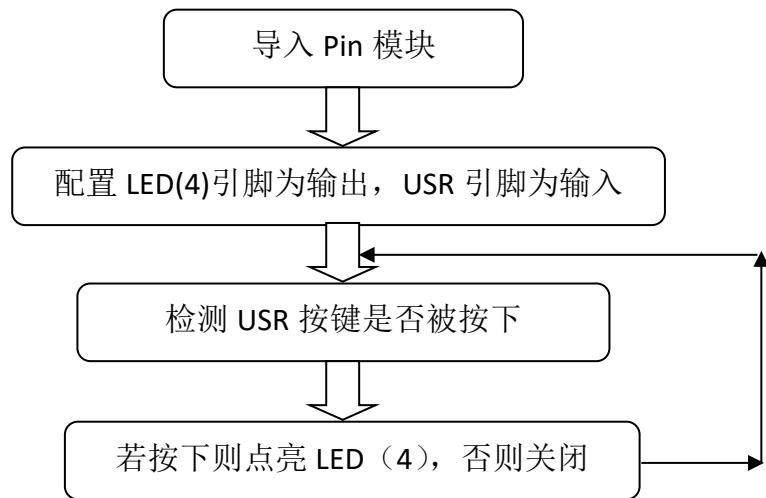


图 4-18 代码编写流程

参考程序代码如下：

```

...
实验名称: GPIO
版本: v1.0
日期: 2019-4-1
作者: 01Studio
社区: www.01studio.org
...

from pyb import Pin

#将 LED(4)-"B4"配置成推挽输出模式
p_out=Pin('B4',Pin.OUT_PP)

#将 USR 按键-"X17"配置为输入方式
p_in = Pin('X17', Pin.IN, Pin.PULL_UP)

while True:

```

```
if p_in.value() == 0: #USR 被按下接地  
    p_out.high()      #点亮 LED (4) 蓝灯  
  
else:  
    p_out.low()       #关闭 LED (4) 蓝灯
```

### ● 实验结果：

程序启动后，按下 USER 按键不放，LED(4)蓝灯保持点亮，松手后熄灭

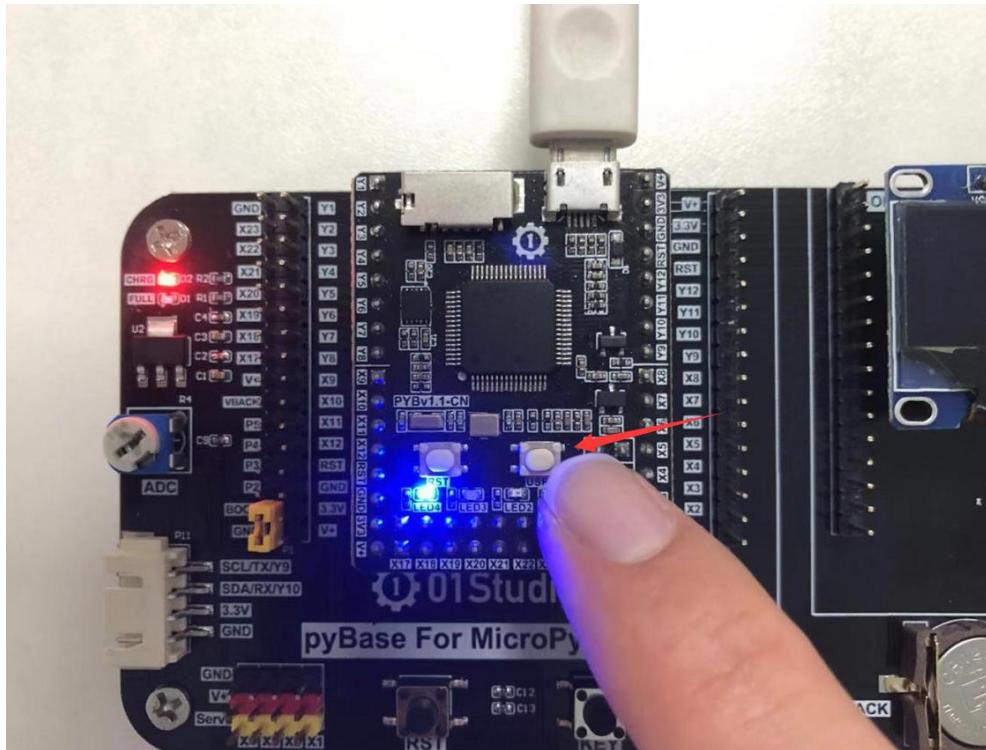


图 4-19 GPIO 实验结果

### ● 总结：

GPIO 是非常通用的实验和功能，学会了 GPIO，就可以把开发板所有的引脚为自己所用，灵活性很强。

## 4.5 外部中断

### ● 前言：

前面我们在做普通的 GPIO 时候，虽然能实现 IO 口输入输出功能，但代码是一直在检测 IO 输入口的变化，因此效率不高，特别是在一些特定的场合，比如某个按键，可能 1 天才按一下去执行相关功能，这样我们就浪费大量时间来实时检测按键的情况。

为了解决这样的问题，我们引入外部中断概念，顾名思义，就是当按键被按下(产生中断)时，我们才去执行相关功能。中断的应用非常普遍，而 pyboard 上基本每个 IO 口都具备中断功能。

### ● 实验平台：

MicroPython 开发套件。

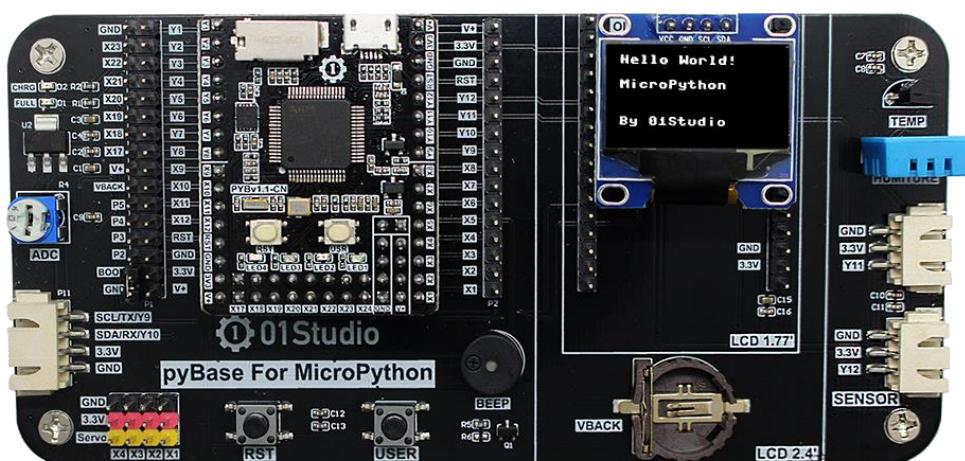


图 4-20 MicroPython 开发套件

### ● 实验目的：

利用中断方式来检查 pyBase 按键 KEY 状态，被按键被按下（产生外部中断）后使 LED (4) 蓝灯的状态反转。

### ● 实验讲解：

pyBase 的按键 KEY 连接到 pyboard 的 ‘Y1’ 引脚，pyboard 一共有 22 个中断线，16 个来自 GPIO 引脚，剩下 6 个来源于内部。先来看看其构造函数：

构造函数	说明
<code>pyb.ExtInt(pin, mode , pull, callback)</code>	<p><b>【Pin】</b> Pin 脚, 如 X1,Y1</p> <p><b>【mode】</b></p> <ul style="list-style-type: none"> <li><code>ExtInt.IRQ_RISING</code>: 上升沿触发</li> <li><code>ExtInt.IRQ_FALLING</code>: 下降沿触发</li> <li><code>ExtInt.IRQ_RISING_FALLING</code>: 上升或下降沿触发</li> </ul> <p><b>【pull】</b></p> <ul style="list-style-type: none"> <li><code>Pin.PULL_NONE</code>-没有上下拉电阻;</li> <li><code>Pin.PULL_UP</code>-启用上拉电阻;</li> <li><code>Pin.PULL_DOWN</code>-启用下拉电阻;</li> </ul> <p><b>【callback】</b> 中断后执行的回调函数</p>

表 4-6 外部中断对象

我们先来了解一下上升沿和下降沿的概念，由于按键 KEY 引脚是通过按键接到 GND，也就是我们所说的低电平“0”，所以当按键被按下再松开时，引脚先获得下降沿，再获得上升沿，如下图所示：

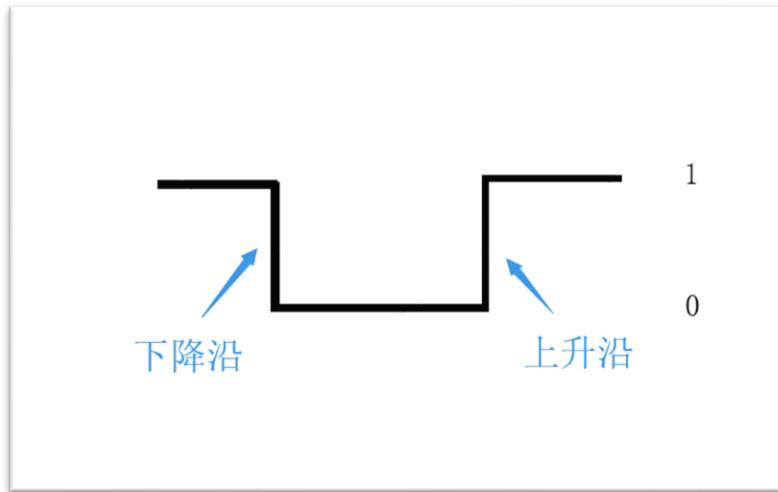


图 4-21 上升沿和下降沿

由此可见，我们可以选择下降沿方式触发外部中断，也就是当按键被按下

的时候立即产生中断。

编程思路中断跟按键章节类似，在初始化中断后，当系统检测到外部终端时候，执行 LED（4）状态反转的代码即可。

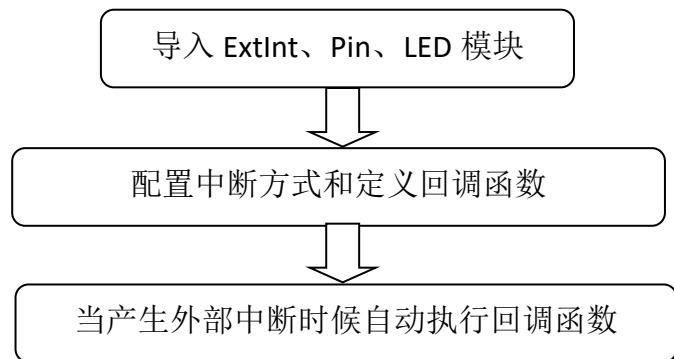


图 4-22 代码编写流程

参考程序代码如下：

```
...
实验名称: 外部中断
版本: v1.0
日期: 2019-4-1
作者: 01Studio
社区: www.01studio.org
...

from pyb import Pin,ExtInt,LED

callback = lambda e: LED(4).toggle()
ext = ExtInt(Pin('Y1'), ExtInt.IRQ_FALLING, Pin.PULL_UP, callback)
#下降沿触发，打开上拉电阻
```

### ● 实验结果：

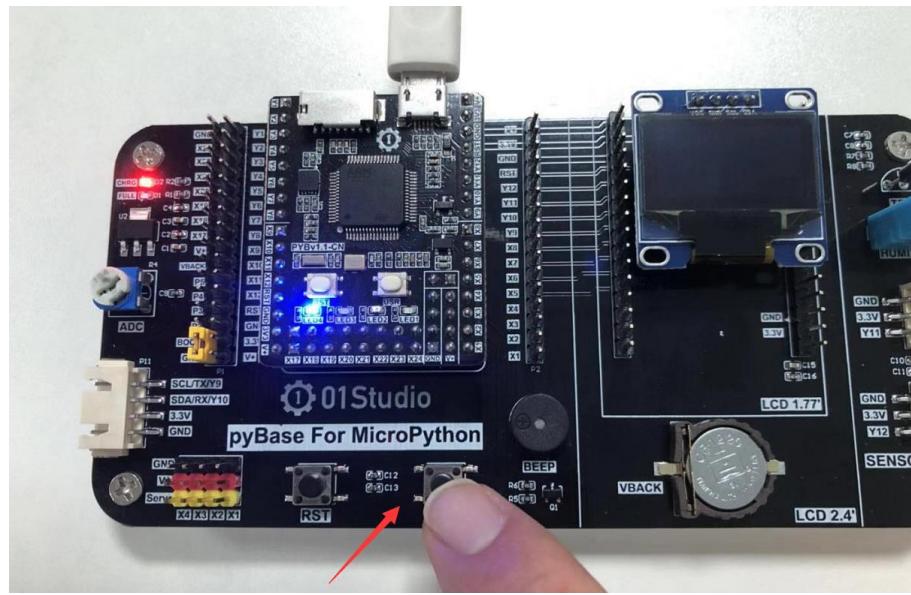


图 4-23 中断方式控制 LED (4)

### ● 总结:

从参考代码来看，只是用了 3 行代码就实现了实验功能，而且相对于使用 `while True` 实时检测函数来看，代码的效率大大增强。外部中断的应用非常广，除了普通的按键输入和电平检测外，很大一部分输入设备，比如传感器也是通过外部中断方式来实时检测，这个在后面的章节会讲述。

## 4.6 I2C 总线（OLED 显示屏）

### ● 前言：

在上一节学习了按键输入设备后，我们这一节先来学习输出设备 OLED 显示屏，其实之前的 LED 灯也算是输出设备，因为它们确切地告诉了我们硬件的状态。只是相对于只有亮灭的 LED 而言，显示屏可以显示更多的信息，体验更好。

本章节的 OLED 显示屏学习，实际上是在使用 I2C 的总线接口，pyBoard 是通过 I2C 总线与 OLED 显示屏通讯的。这章稍微复杂一点，我们把它放在了前面来学习，是因为学会了显示屏的使用，那么在后面的实验中可玩性就更强了。

### ● 实验平台：

MicroPython 开发套件。

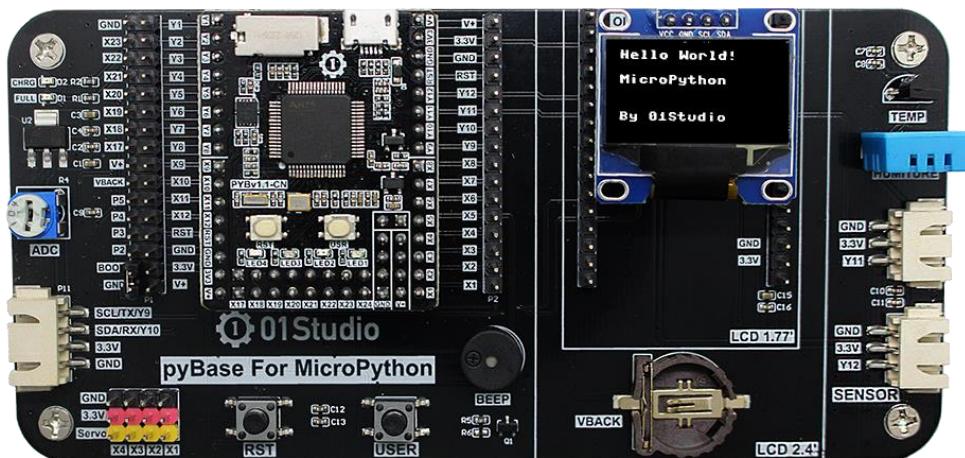


图 4-24 MicroPython 开发套件

### ● 实验目的：

学习使用 MicroPython 的 I2C 总线通讯编程和 OLED 显示屏的使用。

### ● 实验讲解：

#### 什么是 I2C？

I2C 是用于设备之间通信的双线协议，在物理层面，它由 2 条线组成：SCL 和 SDA，分别是时钟线和数据线。也就是说不通设备间通过这两根线就可以进行通信。

## 什么是 OLED 显示屏？

OLED 的特性是自己发光，不像 TFT LCD 需要背光，因此可视度和亮度均高，其次是电压需求低且省电效率高，加上反应快、重量轻、厚度薄，构造简单，成本低等特点。简单来说跟传统液晶的区别就是里面像素的材料是由一个个发光二极管组成，因为密度不高导致像素分辨率低，所以早期一般用作户外 LED 广告牌。随着技术的成熟，使得集成度越来越高。小屏也可以制作出较高的分辨率。



图 4-25 I2C 接口的 OLED

在了解完 I2C 和 OLED 显示屏后，我们先来看看开发板的原理图，也就是 MicroPython 上的 OLED 接口是如何连线的。`pyBoard` 有 2 个自带 I2C 接口，但 1 个用于三轴加速度传感器（X9,X10），另外一个(Y9,Y10)保留了日后其他设备接入使用。

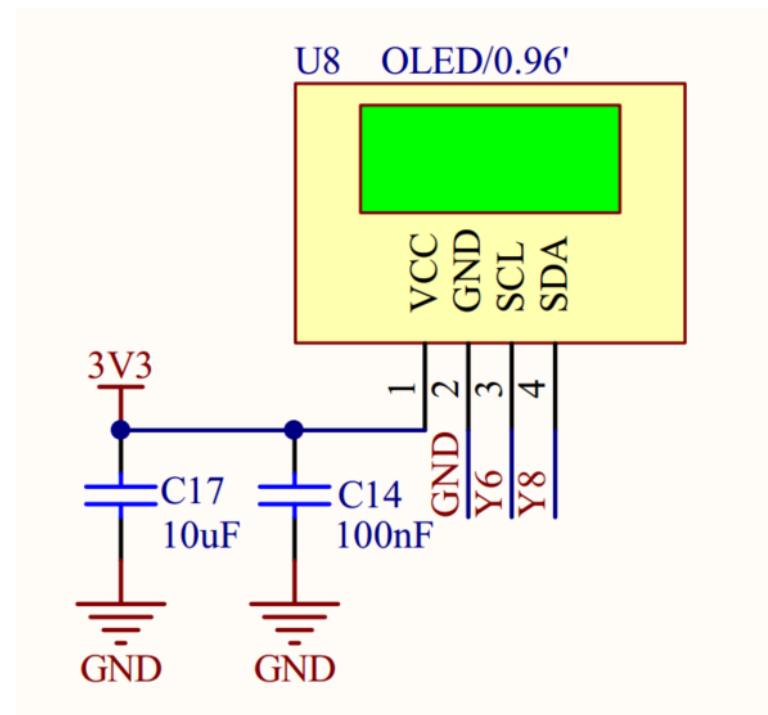


图 4-26 OLED 接口原理图

从上图可见连接到 OLED 的别是  $Y8 \rightarrow SDA$  和  $Y6 \rightarrow SCL$ 。尽管没有连接到自带的 I<sub>2</sub>C 接口，我们也可以通过软件模拟 I<sub>2</sub>C 方式来实现 OLED 的数据通信和使用。本例程将使用 MicroPython 的 Machine 模块来定义 Pin 口和 I<sub>2</sub>C 初始化。具体如下：

先来看看 Machine 模块里面 Pin 的用法：

构造函数	说明
<code>machine.Pin(id,.....)</code>	<code>id</code> 就是指 <code>pyboard</code> 的引脚
使用方法	说明
<code>machine.Pin("Y8")</code>	建立 <code>Pin("Y8")</code> 对象

表 4-7 Machine 的 Pin 对象

Machine 模块下 I<sub>2</sub>C 的用法：

构造函数	说明
<code>machine.I2C(SDA,SCL, ....)</code>	自定义 I <sub>2</sub> C 接口。

使用方法	说明
<code>machine.I2C.scan()</code>	返回扫描到的 I2C 设备地址，从 0x08 至 0x77 之间
<code>machine.I2C.write(buf)</code>	写数据到 I2C 设备。

表 4-8 I2C 对象

定义好 I2C 后，还需要驱动一下 OLED。这里我们已经写好了 OLED 的库函数，在 `ssd1306.py` 文件里面。开发者只需要拷贝到 `pyBoard` 文件系统里面，然后在 `main.py` 里面调用函数即可。人生苦短，我们学会调用函数即可，也就是注重顶层的应用，想深入的小伙伴也可以自行研究 `ssd1306.py` 文件代码。OLED 显示屏的使用方法如下：

使用方法	说明
<code>SSD1306_I2C(width,height,i2c,addr,.....)</code>	OELD 初始化： <code>width</code> : 屏幕宽像素； <code>height</code> : 屏幕高像素； <code>i2c</code> : 已经定义的 I2C 总线对象 <code>addr</code> : I2C 设备地址
<code>SSD1306_I2C.text("string",width,height)</code>	将 “ <code>string</code> ” 在写在指定位置
<code>SSD1306_I2C.show()</code>	显示内容
<code>SSD1306_I2C.fill(RGB)</code>	清屏： <code>RGB=0</code> 表示黑色， <code>RGB=1</code> 表示白色

表 4-9 OLED 显示屏对象

学习了 Pin、I2C、OLED 对象用法后我们通过流程图来理顺一下思路：

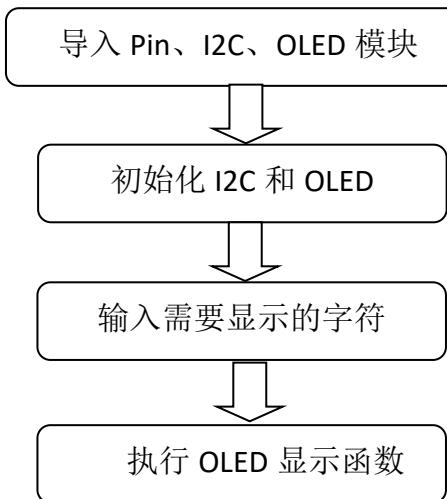


图 4-27 代码编写流程

参考代码如下：

```

...
实验名称: OLED 显示屏 (I2C 总线)
版本: v1.0
日期: 2019.4
作者: 01Studio
...

from machine import I2C,Pin          #从 machine 模块导入 I2C、Pin 子模块
from ssd1306 import SSD1306_I2C    #从 ssd1306 模块中导入 SSD1306_I2C 子模
块

#pyBoard I2C 初始化: sda--> Y8, scl --> Y6
i2c = I2C(sda=Pin("Y8"), scl=Pin("Y6"))

#OLED 显示屏初始化: 128*64 分辨率, I2C 地址是 0x3c
oled = SSD1306_I2C(128, 64, i2c, addr=0x3c)
oled.text("Hello World!", 0, 0)      #写入第 1 行内容
oled.text("MicroPython", 0, 20)       #写入第 2 行内容

```

```
oled.text("By 01Studio", 0, 50)      #写入第3行内容
```

```
oled.show()    #OLED 执行显示
```

上述代码中 OLED 的 I2C 地址是 0x3C,不同厂家的产品地址可能预设不一样，具体参考厂家的说明书。或者也可以通过 I2C.scan() 来获取设备地址。

另外记得将我们提供的示例代码中的 `ssd1306.py` 文件拷贝到 `pyboard` 文件系统下，跟 `main.py` 保持同一个路径。

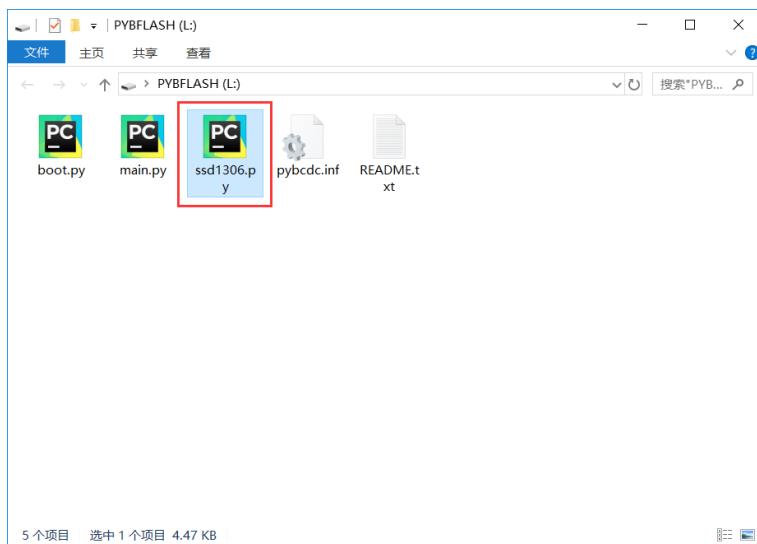
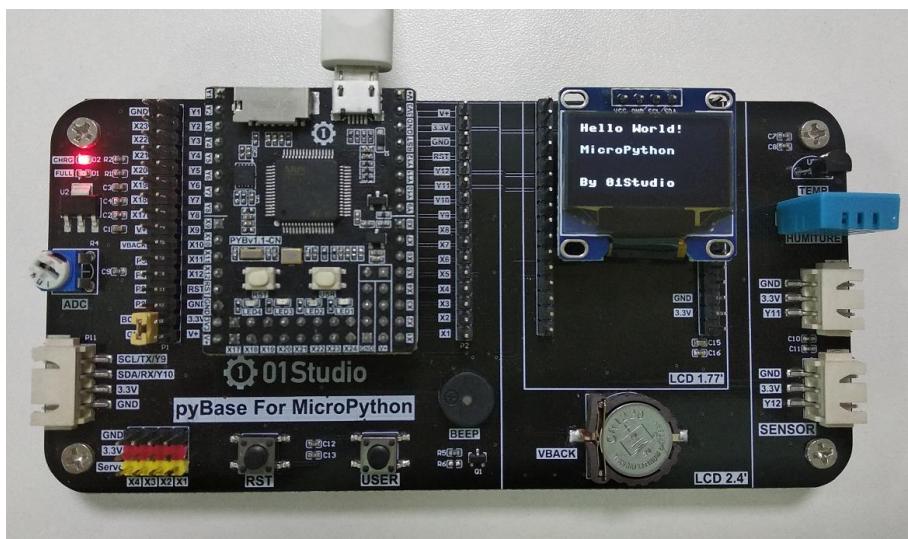


图 4-28 拷贝 `ssd1306.py` 文件

### ● 实验结果：



### 图 4-29 OLED 显示实验

#### ● 总结：

这一节我们学会了驱动 OLED 显示屏，换着以往如果从使用单片机从 0 开发的话你需要了解 I2C 总线原理，了解 OLED 显示屏的使用手册，编程 I2C 代码，有经验的嵌入式工程师搞不好也要弄个几天。现在基本半个小时解决问题。当然前提是别人已经给你搭好桥了，有了强大的底层驱动代码支持，我们只做好应用就好。

这一节学习的意义不仅在完成实验。在学习完 OLED 显示屏实验后，接下来我们的实验都可以使用这个 OLED 来跟用户交互了。这大大提高了实验的可观性。

## 4.7 RTC 实时时钟

- 前言：

时钟可以说我们日常最常用的东西了，手表、电脑、手机等等无时无刻不显示当前的时间。

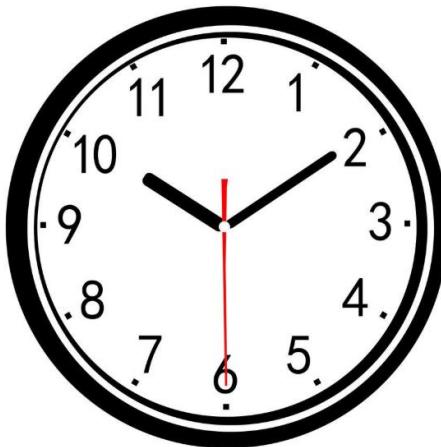


图 4-30 时钟

可以说每一个电子爱好者心中都希望拥有属于自己制作的一个电子时钟，接下来我们就用 MicroPython 开发板来制作一个属于自己的电子时钟。

- 实验平台：

MicroPython 开发套件。

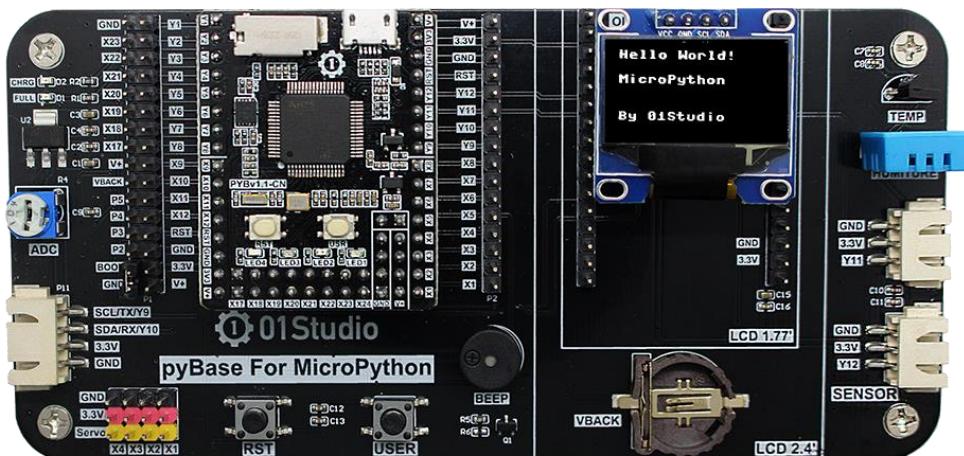


图 4-31 MicroPython 开发套件

● 实验目的：

学习 RTC 编程和制作电子时钟，使用 OLED 显示。

● 实验讲解：

实验的原理是读取 RTC 数据，然后通过 OLED 显示。毫无疑问，强大的 pyBoard 已经集成了内置时钟函数模块。具体如下：

构造函数	说明
<code>pyb.RTC()</code>	<code>RTC()</code> 是内置的实时时钟函数模块
使用方法	说明
<code>RTC().datetime((2019,4,1,1,0,0,0,0))</code>	设置日期时间：顺序分别是年，月，日，星期，时，分，秒，次级秒。 星期： <b>1-7</b> 表示周一至周日 次级秒： <b>255-0</b> 倒着计数
<code>RTC().datetime()</code>	获取当前日期时间。

表 4-10 RTC 实时时钟对象

从上表可以看到 `RTC()` 的使用方法，我们需要做的就是先设定时间，然后再获取当前芯片里的时间，通过 OLED 显示屏显示，如此循环。在循环里，如果一直获取日期时间数据会造成资源浪费，所以可以每隔第一段时间获取一次数据，又由于肉眼需要看到至少每秒刷新一次即可，这里每隔 300ms 获取一次数据，所以具体流程如下：

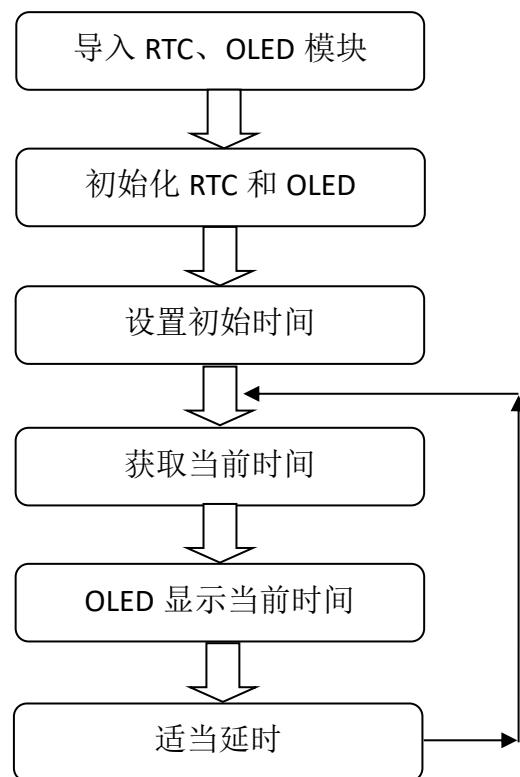


图 4-32 代码编写流程

参考代码如下：

```
...
实验名称: RTC 实时时钟
版本: v1.0
日期: 2019.4
作者: 01Studio
...

# 导入相关模块
from pyb import RTC
from machine import Pin, I2C
from ssd1306 import SSD1306_I2C

# 定义星期和时间（时分秒）显示字符列表
```

```

week = ['Mon', 'Tues', 'Wed', 'Thur', 'Fri', 'Sat', 'Sun']
time = ['', '', '']

# 初始化所有相关对象

i2c = I2C(sda=Pin("Y8"), scl=Pin("Y6"))

oled = SSD1306_I2C(128, 64, i2c, addr=0x3c)

rtc = RTC()

# 首次上电配置时间，按顺序分别是：年，月，日，星期，时，分，秒，次秒级；这里做
#了一个简单的判断，检查到当前年份不对就修改当前时间，开发者可以根据自己实际情况
#来修改。

if rtc.datetime()[0] != 2019:
    rtc.datetime((2019, 4, 1, 1, 0, 0, 0, 0))

while True:

    datetime = rtc.datetime() # 获取当前时间

    oled.fill(0) # 清屏显示黑色背景
    oled.text('01Studio', 0, 0) # 首行显示 01Studio
    oled.text('RTC Clock', 0, 15) # 次行显示实验名称

    # 显示日期，字符串可以直接用“+”来连接
    oled.text(str(datetime[0]) + '-' + str(datetime[1]) + '-' +
str(datetime[2]) + ' ' + week[(datetime[3] - 1)], 0, 40)

    # 显示时间需要判断时、分、秒的值否小于 10，如果小于 10，则在显示前面补“0”
    #达
    #到较佳的显示效果

```

```

for i in range(4, 7):
    if datetime[i] < 10:
        time[i - 4] = "0"
    else:
        time[i - 4] = ""

# 显示时间

oled.text(time[0] + str(datetime[4]) + ':' + time[1] +
str(datetime[5]) + ':' + time[2] + str(datetime[6]), 0, 55)

oled.show()

pyb.delay(300) #延时 300ms

```

由于实验要用到 OLED 显示屏，所以同样别忘了将示例代码该实验文件夹下的 ssd1306.py 文件复制到 pyboard 的文件系统里面。

### ● 实验结果：

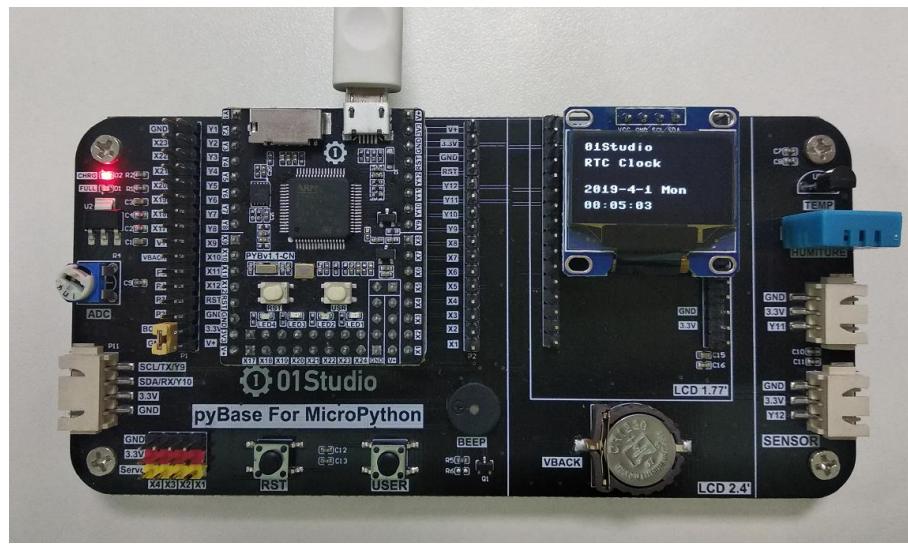


图 4-33 RTC 实时时钟实验

细心的用户或者会发现开发板的右下方有个小的纽扣电池，这个小电池的作用是后备电池，装上后开发板断电后时钟会继续跑，重新上电后时间是最新

的。如果没有这电池，那么开发板断电后时钟无法继续运行。

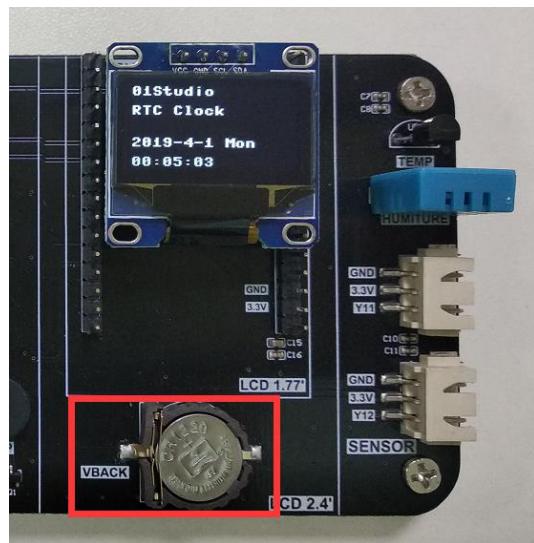


图 4-34 后备电池 (VBACK)

● 总结：

RTC 实时时钟的可玩性很强，我们还可以根据自己的风格来设定数字显示位置，以及加上一些属于自己的字符标识。打造自己的电子时钟。

## 4.8 ADC

### ● 前言：

ADC(analog to digital conversion) 模拟数字转换。意思就是将模拟信号转化成数字信号，由于单片机只能识别二级制数字，所以外界模拟信号常常会通过 ADC 转换成其可以识别的数字信息。常见的应用就是将变化的电压转成数字信号。

### ● 实验平台：

MicroPython 开发套件

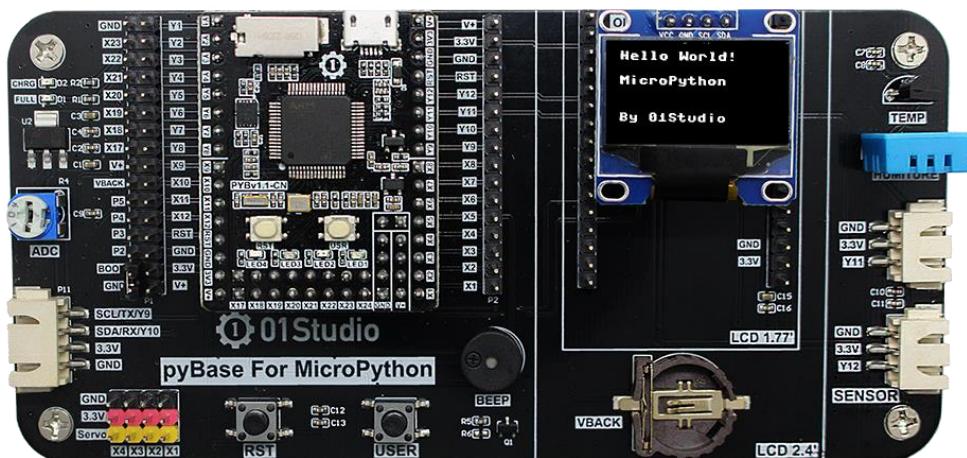


图 4-35 MicroPython 开发套件

### ● 实验目的：

通过编程调用 MicroPython 的内置 ADC 函数，实现测量 0-3.3V 电压，并显示到屏幕上。

### ● 实验讲解：

pyBoard 的 X7 引脚连接到了 pyBase 的电位器，通过电位器的调节可以使得 X7 引脚上的电压变化范围实现从 0-3.3V。

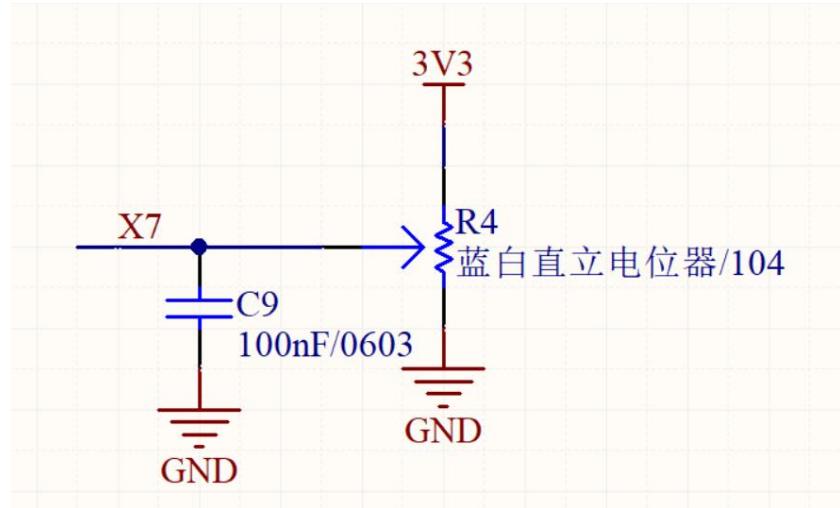


图 4-36 电位器原理图

在了解了原理图后，我们来看看内置 ADC 模块的函数使用方法。

构造函数	说明
<code>pyb.ADC(pin)</code>	开启某个 Pin 引脚的 ADC 功能
使用方法	说明
<code>ADC.read()</code>	读取 AD 值，返回 0-4095 (0v-3.3V)；自带 AD 的 测量的精度是 12 位， $2^{12}=4096$ 。

表 4-11 ADC 对象

你没看错，就这么简单。两句函数就可以获得 AD 数值，让我们来理顺一下编程逻辑。先导入相关模块，然后初始化模块。在循环中不断读取 ADC 的值，转化成电压值后在 OLED 上面显示，每隔 1 秒读取一次，具体如下：

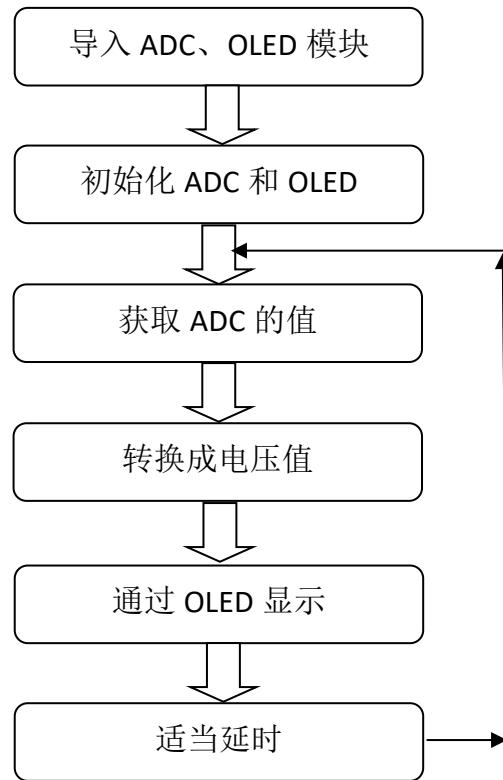


图 4-37 代码编写流程

实验参考代码：

```

...
实验名称: ADC-电压测量
版本: v1.0
日期: 2019.4
作者: 01Studio
说明: 通过对 ADC 数据采集, 转化成电压在显示屏上显示。ADC 精度 12 位, 电压 0-
3.3V。
...
#导入相关模块
import pyb
from machine import Pin,I2C
from ssd1306 import SSD1306_I2C

```

```
#初始化相关模块
i2c = I2C(sda=Pin("Y8"), scl=Pin("Y6"))

oled = SSD1306_I2C(128, 64, i2c, addr=0x3c)

adc = pyb.ADC('X7') #Pin='X7'

while True:

    oled.fill(0) # 清屏显示黑色背景
    oled.text('01Studio', 0, 0) # 首行显示 01Studio
    oled.text('ADC', 0, 15)      # 次行显示实验名称

    #获取 ADC 数值
    oled.text(str(adc.read()),0,40)
    oled.text('(4095)',40,40)

    #计算电压值，获得的数据 0-4095 相当于 0-3V，（%.2f'%）表示保留 2 位小数
    oled.text(str('%.2f'%(adc.read()/4095*3.3)),0,55)
    oled.text('V',40,55)

    oled.show()
    pyb.delay(1000)
```

- 实验结果：

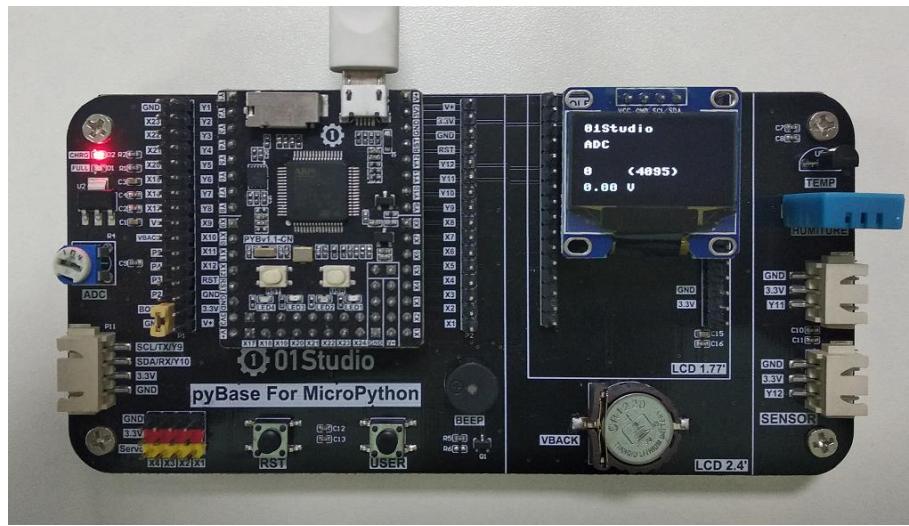


图 4-38 电位器顺时钟拧到尽头是 0V

通过调节电位器，可以发现电压在不断变化。

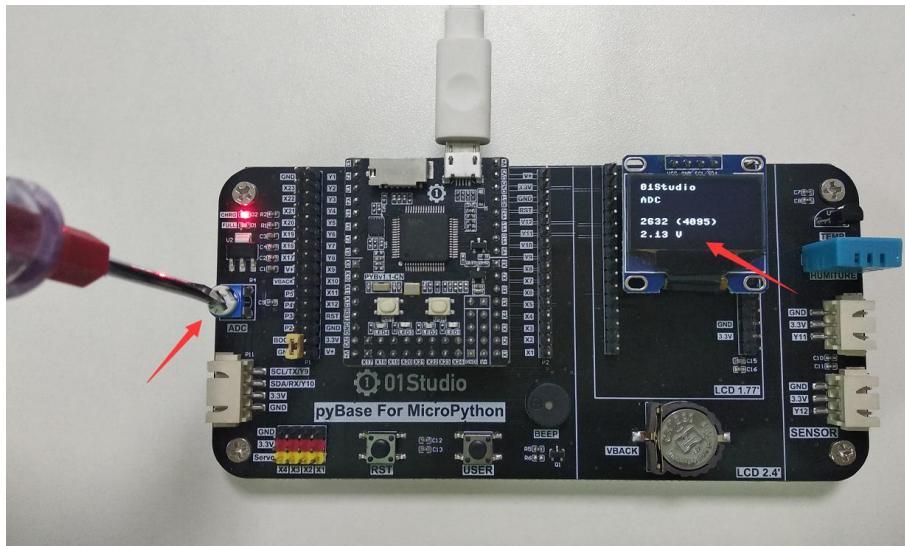


图 4-39 调节电位器来改变电压输入

- 总结：

这一节我们学习了 ADC 的应用。主要用于电压的检测，pyboard 上有很多 ADC 引脚，这意味着可以实现多路同时检测。有兴趣的小伙伴可以自行研究。

## 4.9 DAC

### ● 前言：

DAC (Digital-to-Analog Converter) 数字模拟转换器。我们前面做的 ADC 实验，是将模拟信号转化为了数字信号。这一节做的实验刚好是反转，我们将特定的数字信号通过 DAC 输出。输出的形式多种，如特定电压、信号波形（正弦波、方波、三角波等），有条件的用户可以用示波器来观看输出。本章节利用开发板上的无源蜂鸣器，通过 DAC 输出不同的频率来改变其音色。

### ● 实验平台：

MicroPython 开发套件。

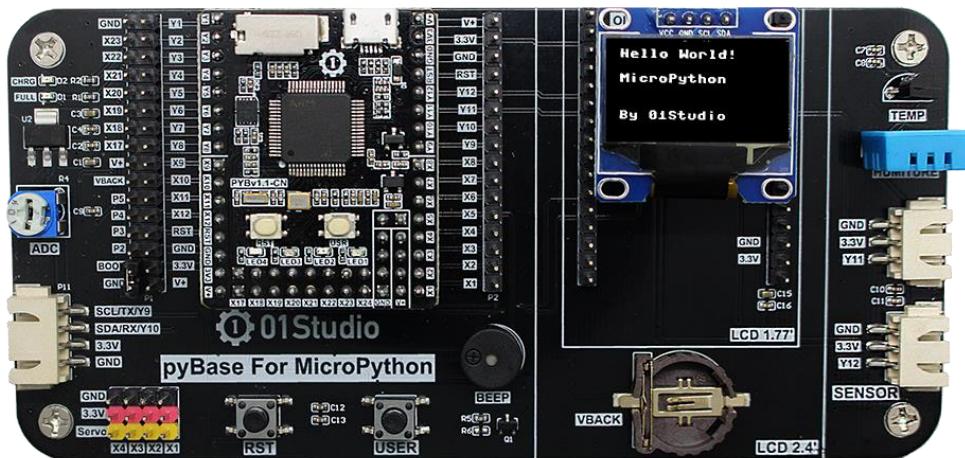


图 4-40 MicroPython 开发套件

### ● 实验目的：

通过 DAC 输出不同频率的方波来驱动蜂鸣器。

### ● 实验讲解：

蜂鸣器分有源蜂鸣器和无源蜂鸣器，有源蜂鸣器的使用方式非常简单，只需要接上电源，蜂鸣器就发声，断开电源就停止发声。而本实验用到的无源蜂鸣器，是需要给定指定的频率，才能发声的，而且可以通过改变频率来改变蜂鸣器的发声音色，以此来判定 pyBoard 的 DAC 输出频率是在变化的。

PyBoard 上有 2 个 DAC 引脚，分别是 X5 和 X6，其中 X5 连接了无源蜂鸣器。如下图所示：

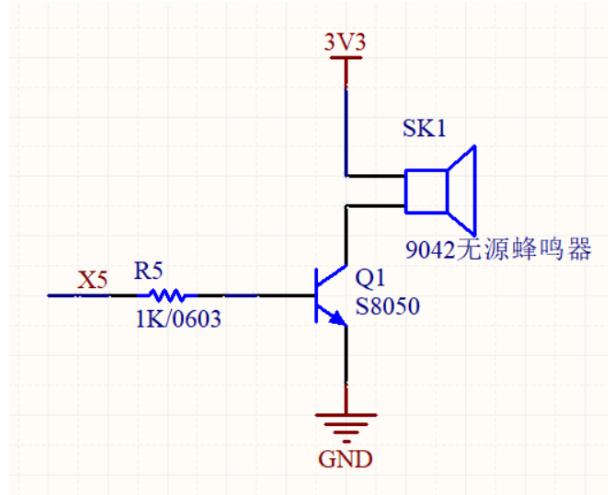


图 4-41 蜂鸣器原理图

在了解了原理图后，我们来看看内置 ADC 模块的函数使用方法。

构造函数	说明
<code>pyb.DAC(port, bits=8)</code>	Port 可以是 1 或者 2，分别对应 X5 和 X6； bits（精度）可以是 8 位或者 12 位。
使用方法	说明
<code>DAC.noise(freq)</code>	产生特定频率的噪声信号
<code>DAC.triangle(freq)</code>	产生特定频率三角波信号
<code>DAC.write(value)</code>	输出特定电压(0-3.3V)。value 的值位 0- $2^{bits}-1$ (例：8bits 那么范围就是 0-255)
<code>DAC.write_timed(data, freq, mode)</code>	将 data(字节数组)的内容以特定频率输出； mode 分别是 <code>DAC.NORMAL</code> 或 <code>DAC.CIRCULAR</code> (循环模式)

表 4-12 DAC 对象

无源蜂鸣器我们可以用特定频率的方波来驱动，方波的原理很简单，就是一定频率的高低电平转换，使用 `DAC.write(0)` 和 `DAC.write(255)` 交替输出即可，

可以通过延时函数来控制输出变换的次数。实际过程中我们没必要用低效率的延时函数，因为看到 DAC 模块里面已经定义好了特定格式输出的函数 `DAC.write_timed(data,freq,mode)`，将 mode 设定成循环模式，就可以循环不断地输出指定频率的信号了。

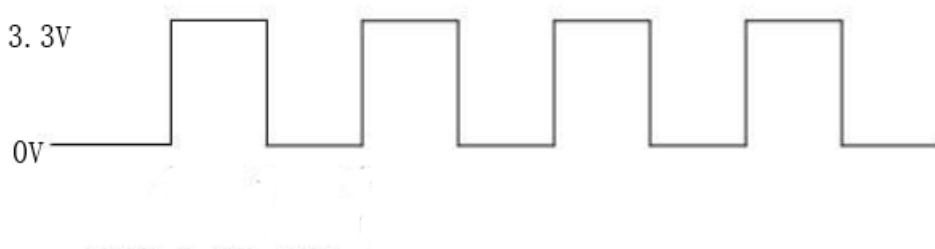


图 4-42 方波信号

结合我们前面的按键实验，我们可以通过 USER 按键按下来切换输出频率，并在 OLED 上直观体验。具体流程如下：

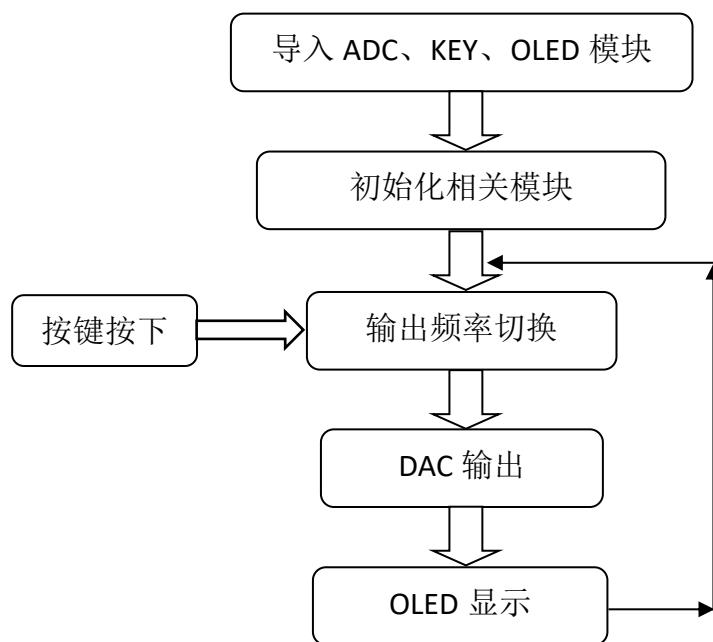


图 4-43 代码编程流程

参考代码如下：

```
...
实验名称: DAC-蜂鸣器
版本: v1.0
日期: 2019.4
作者: 01Studio
说明: 通过 USER 按键让 DAC 输出不同频率的方波来驱动蜂鸣器。
...
#导入相关模块
from pyb import DAC,Switch
from machine import Pin,I2C
from ssd1306 import SSD1306_I2C

#初始化相关模块
i2c = I2C(sda=Pin("Y8"), scl=Pin("Y6"))
oled = SSD1306_I2C(128, 64, i2c, addr=0x3c)

sw = Switch()      #定义按键对象名字为 sw
dac = DAC(1)      #定义 DAC 对象名字为 dac，输出引脚为 X5

#定义 4 组频率值: 1Hz、200Hz、1000Hz、5000Hz
freq=[1,200,1000,5000]

# 定义 8 位精度下方波的值。0、255 分别对应输出 0V、3.3V。需要定义成字节数组。
buf = bytearray(2)
buf[0]=0
buf[1]=255

key_node = 0  #按键标志位
```

```

i = 0          #用于选择频率数组

#####
# 按键和其回调函数
#####

def key():
    global key_node

    key_node = 1

sw.callback(key)  #当按键被按下时，执行函数 key()

#####
# OLED 初始显示
#####

oled.fill(0)  # 清屏显示黑色背景
oled.text('01Studio', 0, 0)  # 首行显示 01Studio
oled.text('DAC-Beep', 0, 15)  # 次行显示实验名称
oled.text('Pls Press USER', 0, 40)  # 显示当前频率
oled.show()

while True:

    if key_node==1: #按键被按下
        i = i+1
        if i == 4:
            i = 0
        key_node = 0 #清空按键标志位

        #DAC 输出指定频率
        dac.write_timed(buf, freq[i]*len(buf), mode=DAC.CIRCULAR)

```

```
#显示当前频率  
  
oled.fill(0) # 清屏显示黑色背景  
  
oled.text('01Studio', 0, 0) # 首行显示 01Studio  
  
oled.text('DAC-Beep', 0, 15) # 次行显示实验名称  
  
oled.text(str(freq[i]) + 'Hz', 0, 40) # 显示当前频率  
  
oled.show()
```

以上代码中 `DAC` 的函数使用非常简单，但为了让实验能更直观，加入按键和 `OLED` 模块后需要更多的代码来支持，但编程逻辑不变。以上代码有几个关键的地方需要补充说明一下：

- (1) 由于底层独特的操作方式，使得输入的数据 `buf` 必须是字节数组，所以需要加入 `bytearray()` 来定义。
- (2) `Key_node` 是全局变量，因此在 `key()` 函数里面用该变量必须添加 `global key_node` 代码，否则会在函数里面新建一个样的变量。
- (3) 变量 `i` 的作用是为了让 `DAC` 选择指定的频率数组 `freq` 输出

### ● 实验结果：

将实验代码文件拷贝到 `pyboard` 文件系统，红灯从亮到灭更新完固件后按下 `RST` 按键重启。发现 `OLED` 显示让我们按下 `USER` 键。

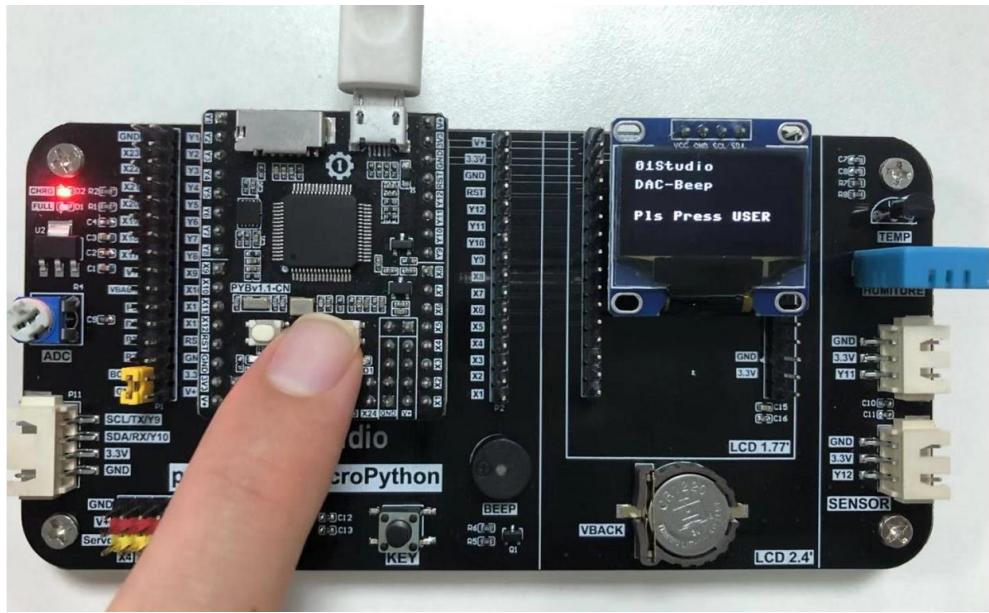


图 4-44 按下 USER 按键开始

每次按下后发现 OLED 上的频率示意发生变化，蜂鸣器的叫声也跟着变化。

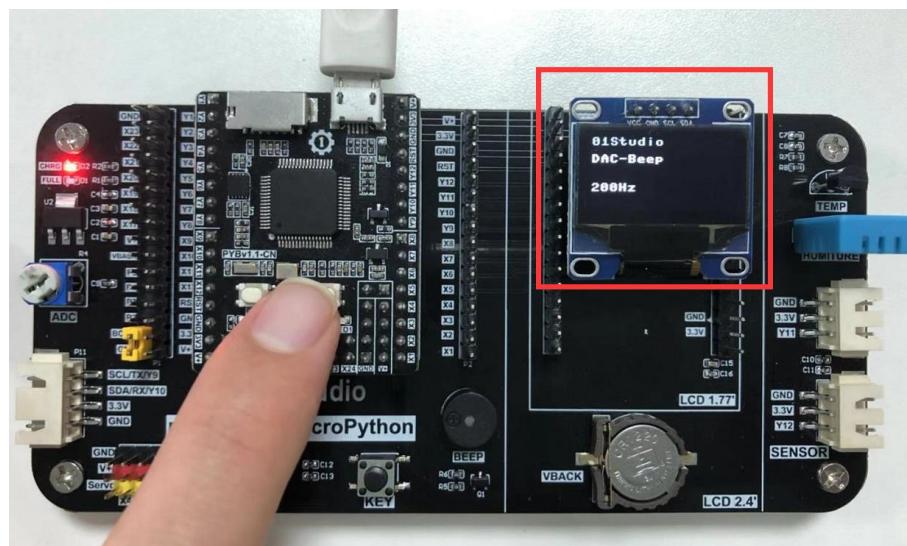


图 4-45 通过按键切换输出频率

有条件的朋友可以使用示波器连接 X5，观察信号波形的变化：

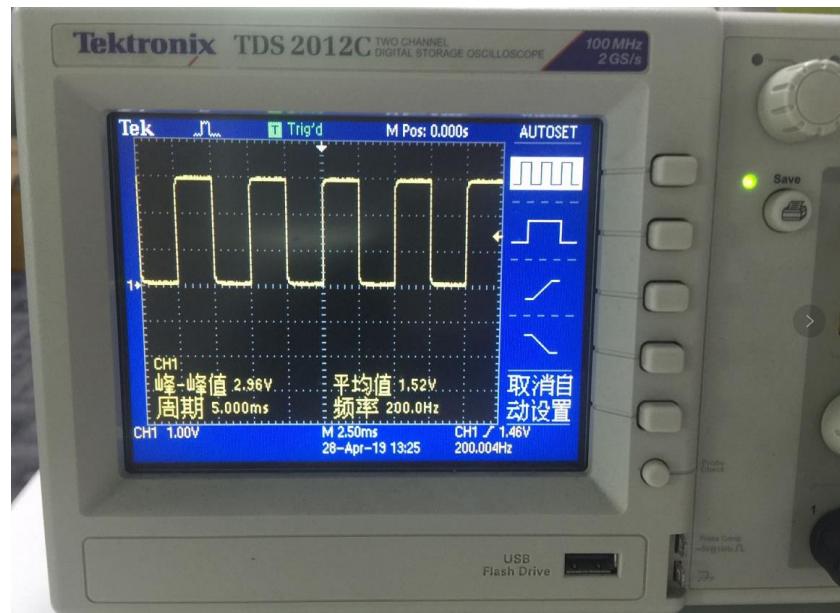


图 4-46 示波器显示

### ● 总结：

到了这一节，我们发现实验对象函数使用方法非常简单，而随着要实现功能的复杂化让编程的代码数量变多，逻辑也略显复杂。这是好事，让我们可以将更多精力放在应用上，做出更多好玩的创意。而不需要过多的关注复杂的底层代码开发。

## 4.10 三轴加速度计

- 前言：

三轴加速度传感器用途广泛，可以计算物体倾斜角度，加速度以及通过算法来计算步数，比如日常用的手环计步。Pyboard 上集成了一款三轴加速度计芯片（MMA7660），可以直接通过编程使用。

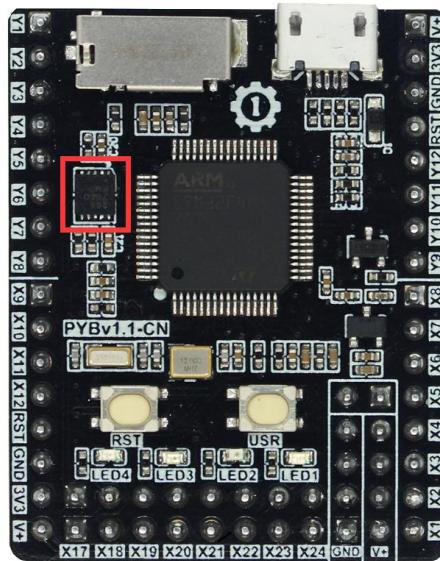


图 4-47 三轴加速度计芯片

- 实验平台：

MicroPython 开发套件。

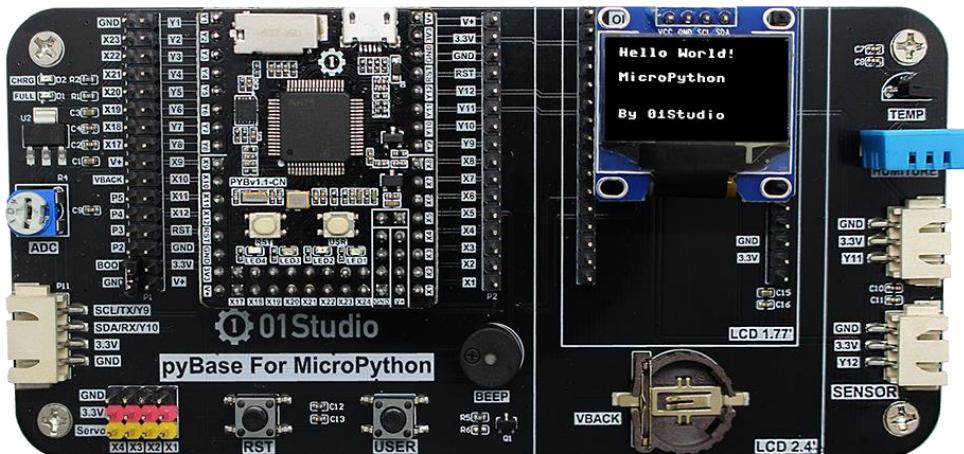


图 4-48 MicroPython 开发套件

### ● 实验目的:

了解三轴加速传感器的工作原理，通过编程获取其各个方向的数值（x 轴、y 轴、z 轴）并在 OLED 上显示。

### ● 实验讲解:

MMA7660 三轴加速度计的使用原理很简单，就是在 x、y、z 各个方向根据受力情况通过数据方式呈现。测量结果范围是 -32 至 31，大约为 -1.5g 至 1.5g (g 为重力加速度， $9.8\text{m/s}^2$ )。我们只需要知道以上 3 个方向的值，便可以计算出各个方向的加速度。

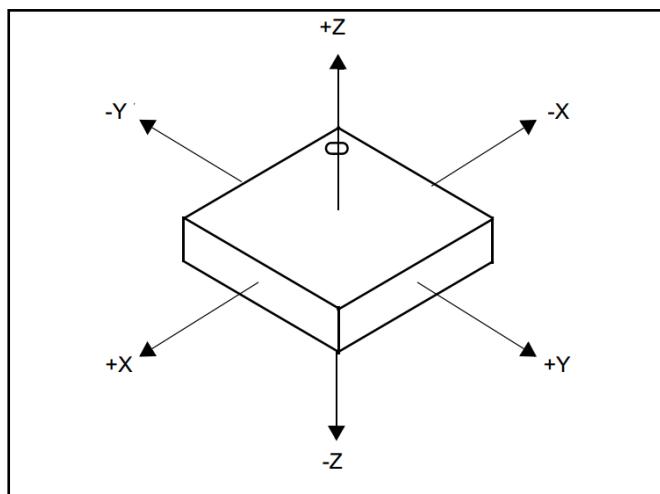


图 4-49 三轴加速度计原理

我们来了解一下加速度计的构造函数和使用方法：

构造函数	说明
<code>pyb.Accel</code>	加速度计构造
使用方法	说明
<code>Accel.x()</code>	获取 x 轴上的值
<code>Accel.y()</code>	获取 y 轴上的值
<code>Accel.z()</code>	获取 z 轴上的值

表 4-13

我们循环 500ms 采集一次，编程逻辑如下：

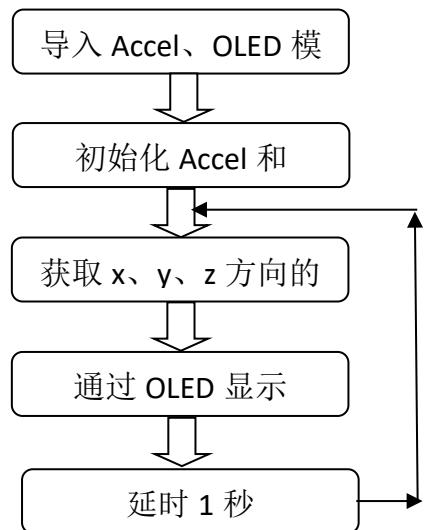


图 4-50 编程流程图

参考代码：

```

...
实验名称: DAC-蜂鸣器
版本: v1.0
日期: 2019.4
作者: 01Studio
说明: 通过编程获取其各个方向的数值（X 轴、Y 轴、Z 轴）并在 OLED 上显示。
...
import pyb
from machine import Pin,I2C
from ssd1306 import SSD1306_I2C

#初始化相关模块
i2c = I2C(sda=Pin("Y8"), scl=Pin("Y6"))
oled = SSD1306_I2C(128, 64, i2c, addr=0x3c)

```

```

accel = pyb.Accel()

while True:

    oled.fill(0) #清屏
    oled.text('01Studio', 0, 0)
    oled.text('Accel test:', 0, 15)

    #获取 x,y,z 的值并显示
    oled.text('X:' +str(accel.x()),0,40)
    oled.text('Y:' +str(accel.y()),44,40)
    oled.text('Z:' +str(accel.z()),88,40)
    oled.show()

    pyb.delay(1000) #延时 1s

```

### ● 实验结果：

运行后见到 OLED 显示 X/Y/Z 的值，用户可以根据翻转、抖动开发板等方式来观察 X/Y/Z 的值的事实变化情况。

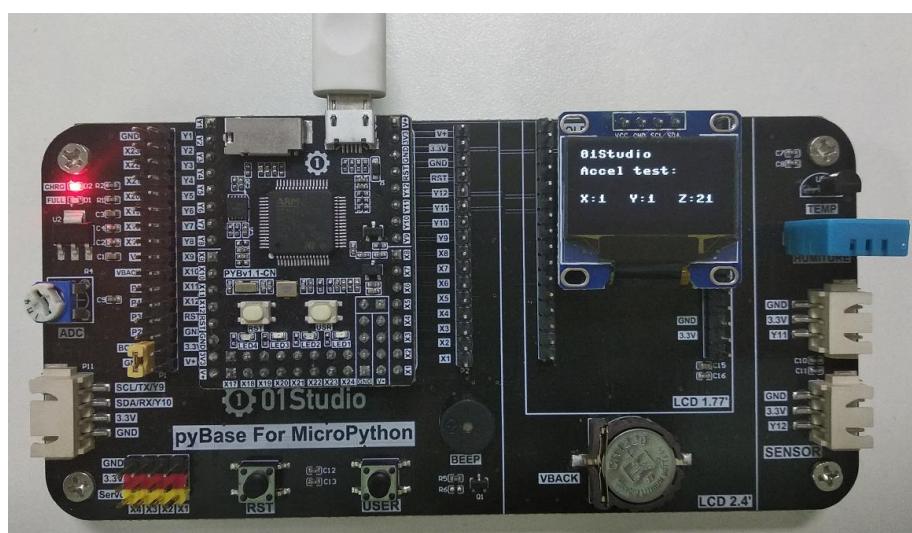


图 4-51

细心的用户或许会发现，当开发板静止在桌面水平面上，Z 的值始终保持在 20-21 左右。查看 MMA7660 芯片手册，看到 0.98g 也就是重力加速度的值约为 21，而板子平放静止的时候 Z 轴有个重力加速的，这不难理解。

6-bit result	Binary	2's Comp	g value	Angle X or Y		Angle Z
0	0	0	0.000g	0.00°		90.00°
1	1	1	0.047g	2.69°		87.31°
2	10	2	0.094g	5.38°		84.62°
3	11	3	0.141g	8.08°		81.92°
4	100	4	0.188g	10.81°		79.19°
5	101	5	0.234g	13.55°		76.45°
6	110	6	0.281g	16.33°		73.67°
7	111	7	0.328g	19.16°		70.84°
8	1000	8	0.375g	22.02°		67.98°
9	1001	9	0.422g	24.95°		65.05°
10	1010	10	0.469g	27.95°		62.05°
11	1011	11	0.516g	31.04°		58.96°
12	1100	12	0.563g	34.23°		55.77°
13	1101	13	0.609g	37.54°		52.46°
14	1110	14	0.656g	41.01°		48.99°
15	1111	15	0.703g	44.68°		45.32°
16	10000	16	0.750g	48.59°		41.41°
17	10001	17	0.797g	52.83°		37.17°
18	10010	18	0.844g	57.54°		32.46°
19	10011	19	0.891g	62.95°		27.05°
20	10100	20	0.938g	69.64°		20.36°
21	10101	21	0.984g	79.86°		10.14°
22	10110	22	1.031g			

图 4-52 重力加速度对应的值约为 21

### ● 总结：

在嵌入式编程中，有些时候我们需要学看芯片手册，手册不用全看，可以锻炼到的是学会从芯片手册中寻找关键信息，以理解实际产品的基本参数。

## 4.11 UART（串口通信）

### ● 前言：

串口是非常常用的通信接口，有很多工控产品、无线透传模块都是使用串口来收发指令和传输数据，这样用户就可以在无须考虑底层实现原理的前提下将各类串口功能模块灵活应用起来。

### ● 实验平台：

MicroPython 开发套件或 pyBoard v1.1-CN。

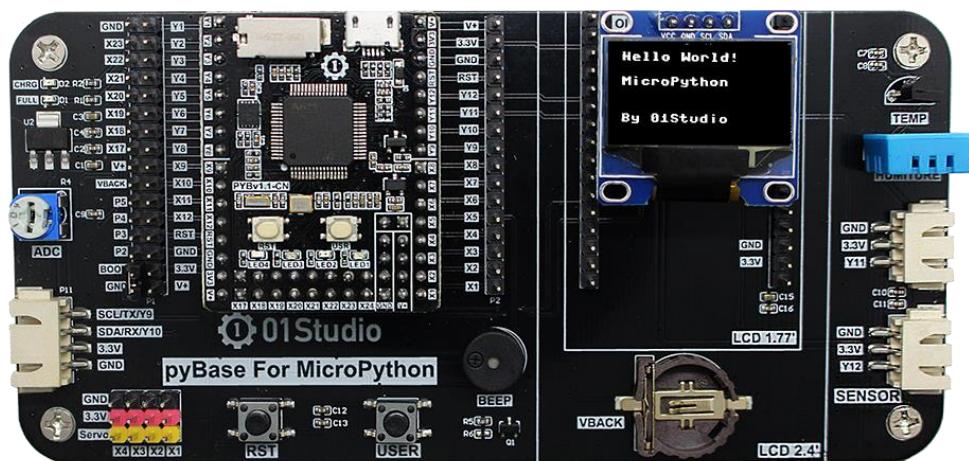


图 4-53 MicroPython 开发套件

### ● 实验目的：

编程实现串口收发数据。

### ● 实验讲解：

pyBoard v1.1-CN 一共有 6 个串口，编号是 1-6，其中串口 5 用于 REPL 调试，因此并没有开放使用。串口号和引脚关系如下

UART(1)	TX	X9
	RX	X10
UART(2)	TX	X3
	RX	X4
UART(3)	TX	Y9

	RX	Y10
UART(4)	TX	X1
	RX	X2
UART(6)	TX	Y1
	RX	Y2

表 4-14 pyBoard 串口引脚资源

我们来了解一下 `pyb` 串口对象的构造函数和使用方法：

构造函数
<code>uart=pyb.UART.init(baudrate, bits=8, parity=None, stop=1, *, timeout=0, flow=0, timout_char=0, read_buf_len=64)</code>
创建 UART 对象。
<p><b>【id】</b> 1-6 (5 已用于 REPL, 不能使用)</p> <p><b>【baudrate】</b> 波特率, 常用 115200、9600</p> <p><b>【bits】</b> 数据位</p> <p><b>【parity】</b> 校验; 默认 None, 0(偶校验), 1(奇校验)</p> <p><b>【stop】</b> 停止位, 默认 1</p> <p><b>【flow】</b> 流控</p> <p>...</p>
使用方法
<code>uart.deinit()</code>
关闭串口
<code>uart.any()</code>
返回等待读取的字节数据, 0 表示没有
<code>uart.read([nbytes])</code>
<b>【nbytes】</b> 读取字节数
<code>UART.readline()</code>
读行
<code>UART.write(buf)</code>
<b>【buf】</b> 串口 TX 写数据

\*更多使用说明请阅读官方文档：

[http://docs.micropython.01studio.org/zh\\_CN/latest/library/pyb.UART.html#pyb-uart](http://docs.micropython.01studio.org/zh_CN/latest/library/pyb.UART.html#pyb-uart)

表 4-15 PYB 的 UART 对象

我们可以用一个 USB 转 TTL 工具，配合电脑上位机串口助手来跟 MicroPython 开发板模拟通信。

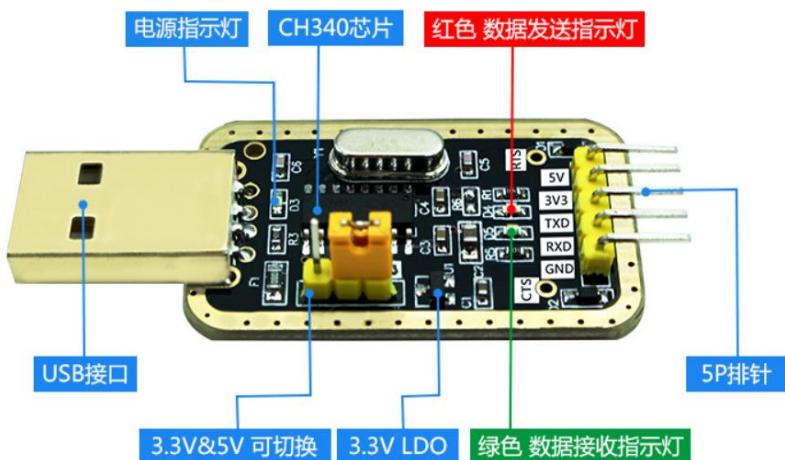


图 4-54 常用 USB 转串口工具

注意要使用 3.3V 电平的 USB 转串口 TTL 工具，本实验我们使用串口 3，也就是 Y9 (TX) 和 Y10 (RX)，接线示意图如下：



图 4-55 串口接线图

在本实验中我们可以先初始化串口，然后给串口发去一条信息，这样 PC 机

的串口助手就会在接收区显示出来，然后进入循环，当检测到有数据可以接收时候就将数据接收并打印，并通过 REPL 打印显示。代码编写流程图如下：

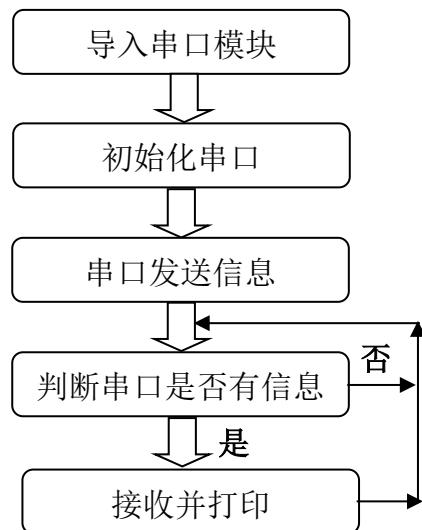


图 4-56 代码编程流程图

参考代码：

```
...
实验名称: 串口通信
版本: v1.0
日期: 2019.10
作者: 01Studio
说明: 通过编程实现串口通信, 跟电脑串口助手实现数据收发。
...
#导入串口模块
from pyb import UART

uart=UART(3,115200) #设置串口号 3 和波特率, TX--Y9,RX--Y10

uart.write('Hello 01Studio!')#发送一条数据
```

```
while True:

    #判断有无收到信息
    if uart.any():

        text=uart.read(64) #默认单次最多接收 64 字节
        print(text) #通过 REPL 打印串口 3 接收的数据
```

### ● 实验结果：

我们按照上述方式将 USB 转 TTL 的 TX 接到 Y10，RX 接到 Y9。GND 接一起，3.3V 可以选择接或不接。

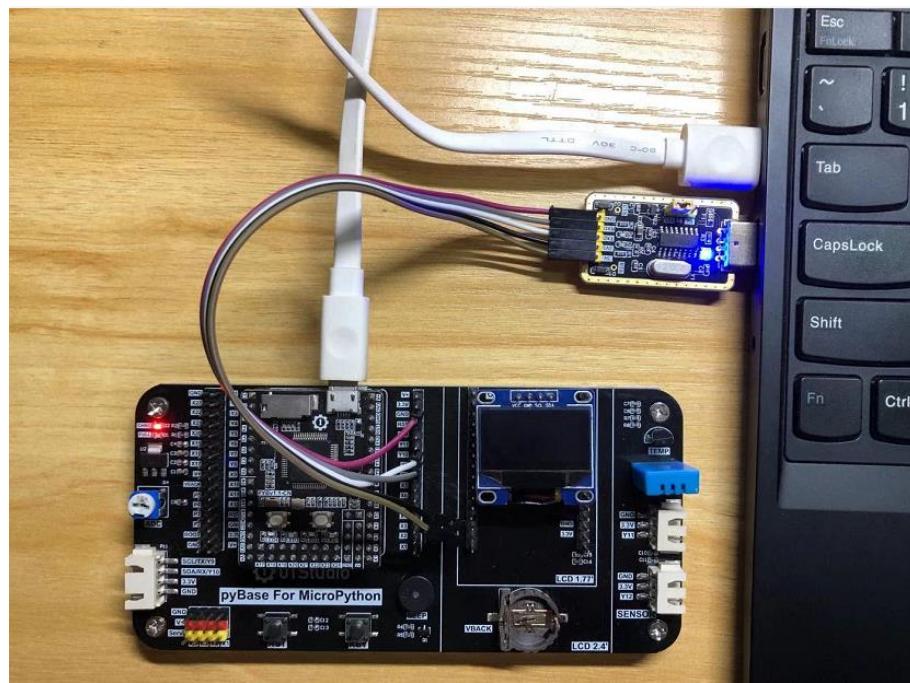


图 4-57 将 pyboard 和 usb ttl 工具同时接入电脑

这时候打开电脑的设备管理器，能看到 2 个 COM。写着 CH340 的是串口工具，另外一个则是 pyboard 的 REPL。如果 CH340 驱动没安装，则需要手动安装，驱动在：配套资料包→开发工具→windows→串口终端→CH340 文件夹下。



图 4-58

本实验要用到串口助手，打开配套资料包→开发工具→windows→串口终端工具下的【UartAssist.exe】软件。

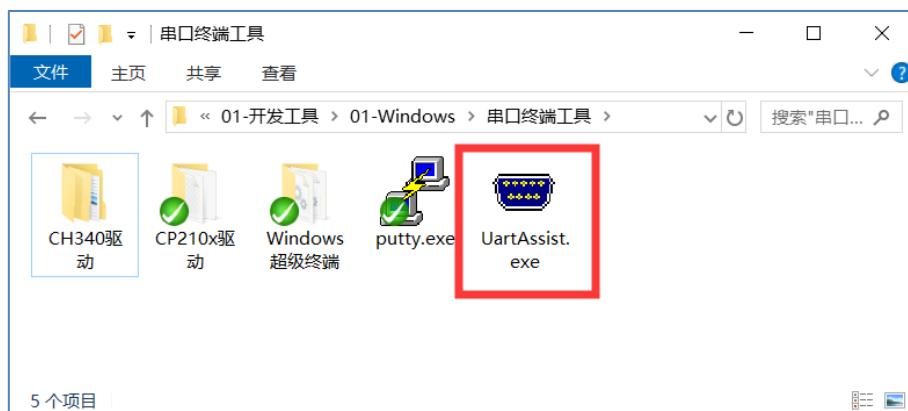


图 4-59

根据上图设备管理器里面的信息，将串口工具配置成 COM14，REPL 串口配置成 COM27（根据自己的串口号调整）。波特率 115200。运行程序，可以看到一开始串口助手收到 pyboard 上电发来的信息“Hello 01Studio!”。我们在串口助手的发送端输入“<http://www.01studio.org>”，点击发送，可以看到 pyboard 在接收到该信息后在 REPL 里面打印了出来。如下图所示：

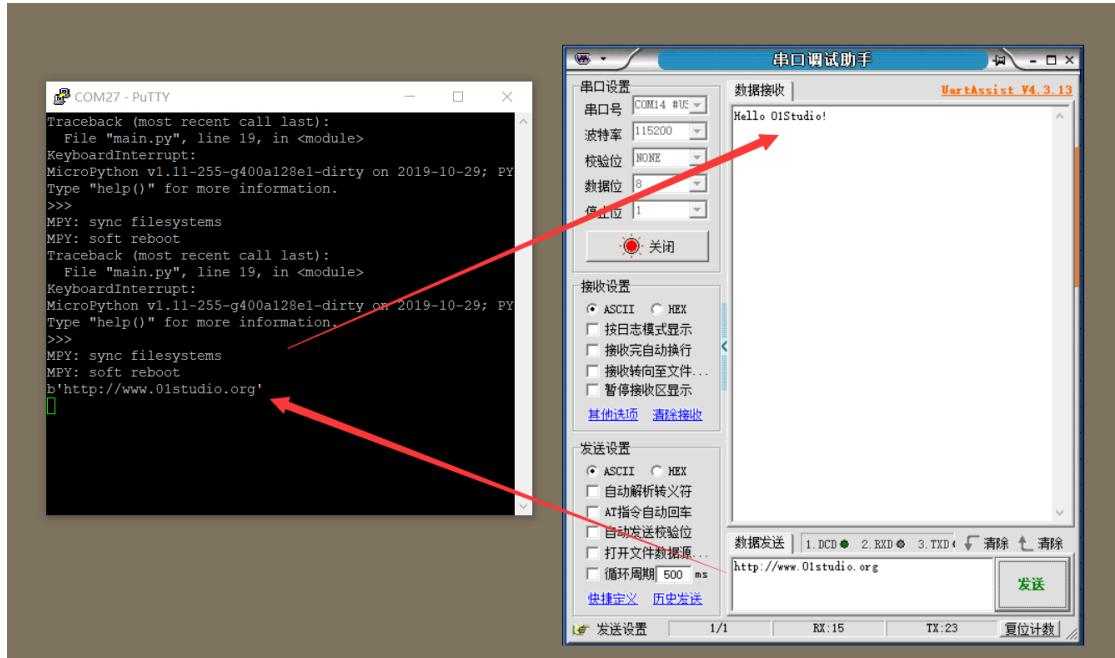


图 4-60 串口收发实验

### ● 总结：

通过本节我们学会了串口收发应用，`pyboard` 的串口资源非常丰富，因此可以接非常多的串口外设。从而实验更多的功能。

## 4.12 LCD 显示屏

### ● 前言：

前面用到的 OLED 显示屏虽然能显示信息，但是颜色只有黑白，而且分辨率也比较低 128x64，本节我们来学习 3.2 寸 TFT\_LCD 彩色显示屏的使用方法。

LCD 液晶显示屏是非常常见的一个外接显示设备，跟前面的 OLED 显示屏相比，LCD 会更常用一些，我们看到的手持设备、小型电器，很多都用到 LCD，部分配合触摸屏应用，能实现非常多的功能。

### ● 实验平台：

MicroPython 开发套件、3.2 寸 LCD（电阻触摸）。

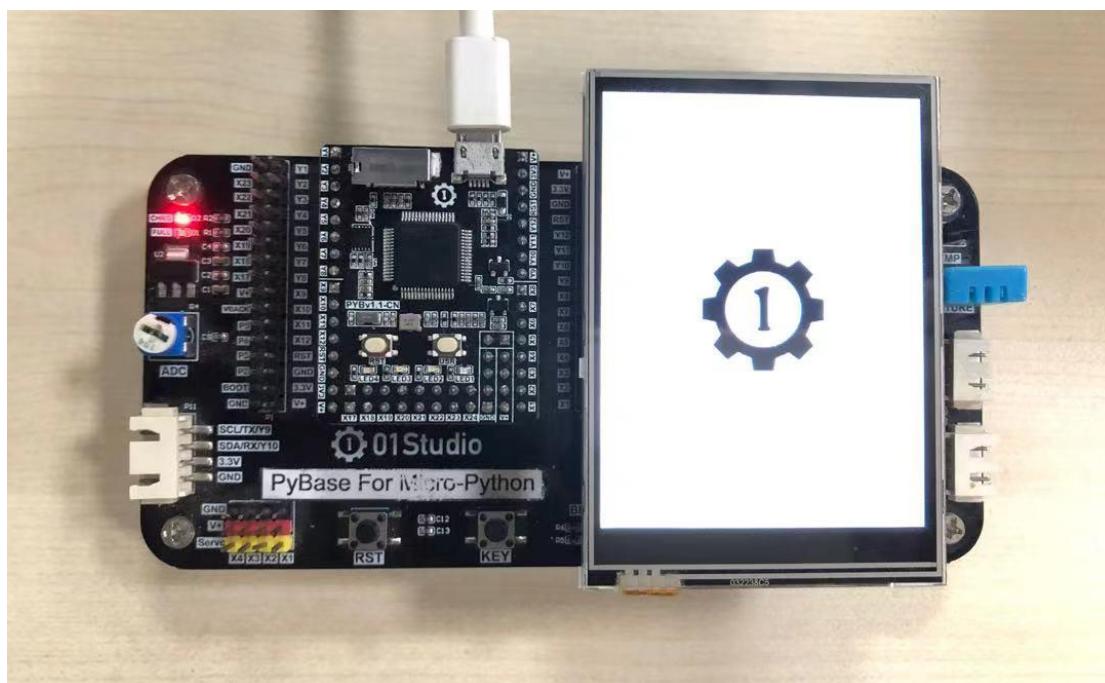


图 4-61 3.2 寸 LCD

### ● 实验目的：

通过 MicorPython 编程方式实现 LCD 的各种显示功能，包括画点、线、矩形、圆形、显示英文、显示图片等。

### ● 实验讲解：

我们先来看看实验所使用 LCD 的参数介绍：



图 4-62 3.2 寸 LCD 显示屏 (电阻触摸)

功能参数	
供电电压	3.3V
屏幕尺寸	3.2 寸
分辨率	240*320
颜色参数	TFT 彩色
驱动芯片	ILI9341 + XPT2046(触摸)
触摸方式	电阻屏
通讯方式	SPI 总线
接口定义	2.54mm 排母 (兼容 pyboard v1.1 接口)
整体尺寸	7.7*5.5 cm

表 4-16 功能参数

3.2 寸 LCD 跟 pyboard 的引脚连线如下表，从下表和图可以看到 LCD 实际连接到 pyboard 的 Y1-Y8, SPI(2)接口。

pyboard	LCD	LCD 引脚说明
Y1	PEN	触摸屏中断
Y2	CS2	触摸屏片选
Y3	D/C	数据/命令选择
Y4	RST	复位
Y5	CS1	LCD 片选
Y6	SCK	SPI 接口
Y7	MISO	SPI 接口
Y8	MOSI	SPI 接口
3.3V	VCC	电源 (3.3V 供电)
GND	GND	地

图 4-63 LCD 引脚连接和说明

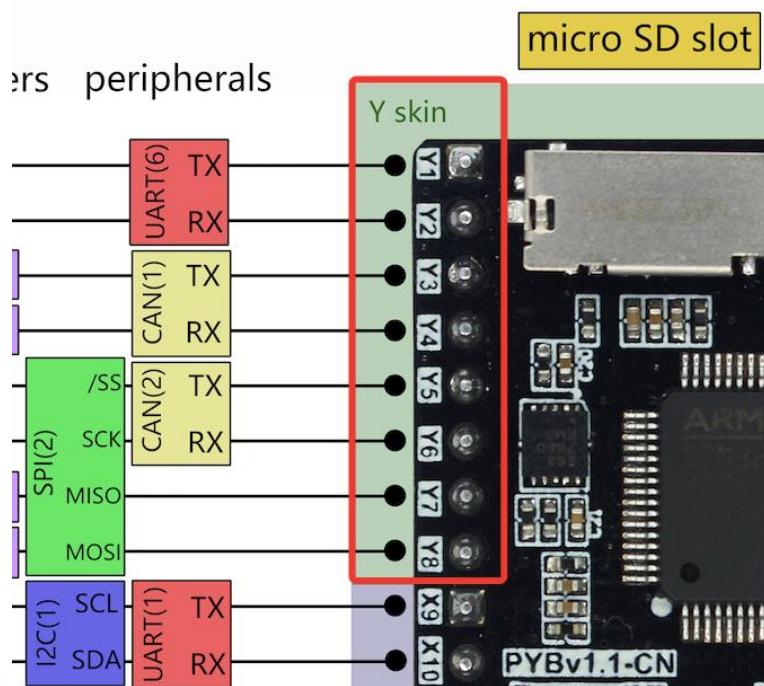


图 4-64 LCD 连接到 Y1-Y8

实验用的 LCD 是 3.2 寸，驱动是常见的 ILI9341，使用 SPI 接口跟 pyboard 通信，按以往嵌入式 C 语言开发，我们需要对 ILI9341 进行编程实现驱动，然后再建立各种描点、划线、以及显示图片函数。

使用 MicroPython 其实也需要做以上工作，但由于可读性和移植性强的特点，我们只需要搞清各个对象函数使如何使用即可。总的来说和之前一样，有构造函数和功能函数。构造函数解决的是初始化问题，告诉 `pyboard` 外设是怎么接线，是什么样的；而功能函数解决的则是使用问题，我们基于自己的需求直接调用相关功能函数，实现自己的功能即可！

我们管这些函数的集合叫驱动，驱动可以是预先在固件里面，也可以通过.py 文件存放在开发板文件系统。也就是说工程师已经将复杂的底层代码封装好，我们顶层直接使用 `python` 开发即可，人生苦短。我们来看看 `pyBoard` 固件中 3.2 寸 LCD 的构造函数和使用方法。

构造函数
<code>tftlcd.LCD32(portrait=1)</code>
构建 3.2 寸 LCD 对象。
<b>【portrait】</b> 设置屏幕方向： <ul style="list-style-type: none"><li>● 1 - 竖屏, 240*320, 默认</li><li>● 2 - 横屏, 320*240, 1 基础上顺时针旋转 90°</li><li>● 3 - 竖屏, 240*320, 1 基础上顺时针旋转 180°</li><li>● 4 - 横屏, 320*240, 1 基础上顺时针旋转 270°</li></ul>
使用方法
<code>LCD32.fill(color)</code>
<b>【color】</b> RGB 颜色数据；如(255,0,0)表示红色。
<code>LCD32.drawPixel(x,y,color)</code>
画点。 <b>【x】</b> :横坐标， <b>【y】</b> :纵坐标， <b>【color】</b> :颜色。
<code>LCD32.drawLine(x0,y0,x1,y1,color)</code>
画线段。 <b>【x0,y0】</b> :起始坐标， <b>【x1,y1】</b> :终点坐标， <b>【color】</b> :颜色
<code>LCD32.drawRect(x,y,width,height,color,border=1,fillcolor=None)</code>

画矩形。

【x,y】 :起始坐标,

【width】 :宽度,

【height】 :高度,

【color】 :颜色,

【border】 :边宽,

【fillcolor】 :填充颜色, 默认 None 为不填充

LCD32.drawCircle(x,y,radius,color,border=1,fillcolor=None)

画圆。

【x, y】 :圆心,

【radius】 :半径,

【color】 :颜色,

【border】 :边宽,

【fillcolor】 :填充颜色, 默认 None 为不填充

LCD32.printStr(text,x,y,color,backcolor=None,size=2)

写字符。

【text】 :字符,

【x,y】 :起始坐标,

【color】 :字体颜色;

【backcolor】 :字体背景颜色;

【size】 :字体尺寸 (1-小号, 2-标准, 3-中号, 4-大号)

LCD32.Picture(x,y,filename)

显示图片。支持图片格式类型: jpg、bmp

【x,y】 :起始坐标。

【filename】 : 图片路径+名称, 如: "/flash/cat.jpg"

(‘/’ 表示开发板的板载 flash 的根目录。)

表 4-17 3.2 寸 LCD 对象

有了上面的对象构造函数和使用说明，编程可以说是信手拈来了，我们在使用中将以上功能都跑一遍先看看编程流程图：

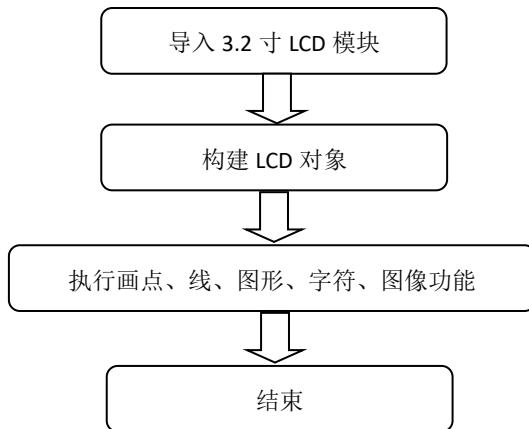


图 4-65 编程流程图

参考 main.py 函数代码如下：

```
...  
实验名称: 3.2 寸 LCD 液晶显示屏  
版本: v1.0  
日期: 2021.8  
作者: 01Studio  
实验平台: pyBoard  
说明: 通过编程实现 LCD 的各种显示功能, 包括填充、画点、线、矩形、圆形、显示英文、显示图片等。  
...  
  
#导入相关模块  
from tftlcd import LCD32  
import time  
  
#定义常用颜色  
RED = (255,0,0)
```

```
GREEN = (0,255,0)
BLUE = (0,0,255)
BLACK = (0,0,0)
WHITE = (255,255,255)

#####
# 构建 3.2 寸 LCD 对象并初始化
#####
d = LCD32(portrait=1) #默认方向竖屏

#填充白色
d.fill(WHITE)

#画点
d.drawPixel(5, 5, RED)

#画线段
d.drawLine(5, 10, 200, 10, RED)

#画矩形
d.drawRect(5, 30, 200, 40, RED, border=5)

#画圆
d.drawCircle(100, 120, 30, RED, border=5)

#写字符,4 种尺寸
d.printStr('Hello 01Studio', 10, 200, RED, size=1)
d.printStr('Hello 01Studio', 10, 230, GREEN, size=2)
d.printStr('Hello 01Studio', 10, 270, BLUE, size=3)
```

```
time.sleep(5) #等待 5 秒

#显示图片

d.Picture(0,0,"/flash/picture/1.jpg")

time.sleep(3)

d.Picture(0,0,"/flash/picture/2.jpg")

time.sleep(3)

d.Picture(0,0,"/flash/picture/01studio.jpg")
```

### ● 实验结果：

将示例程序的素材文件上传到 pyBoard 开发板。推荐直接用 U 盘拷贝然后复位开发板，速度更快。（也可以只上传单张图片，注意修改代码中文件的路径即可。）

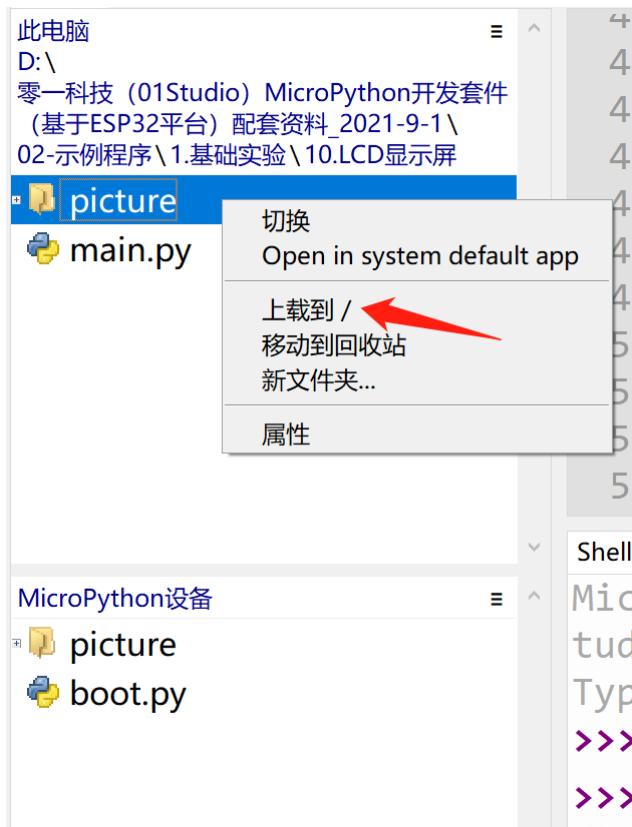


图 4-66 上传图片文件夹到 Flash

运行程序，可以看到 LCD 依次显示相关内容。



图 4-67 点、线、字符等显示



图 4-68 显示图片

由于我们 LCD 接口兼容 pyboard，因此也可以直接将 pyboard v1.1-CN 插到 LCD 背面，使用锂电池供电。如下图所示：

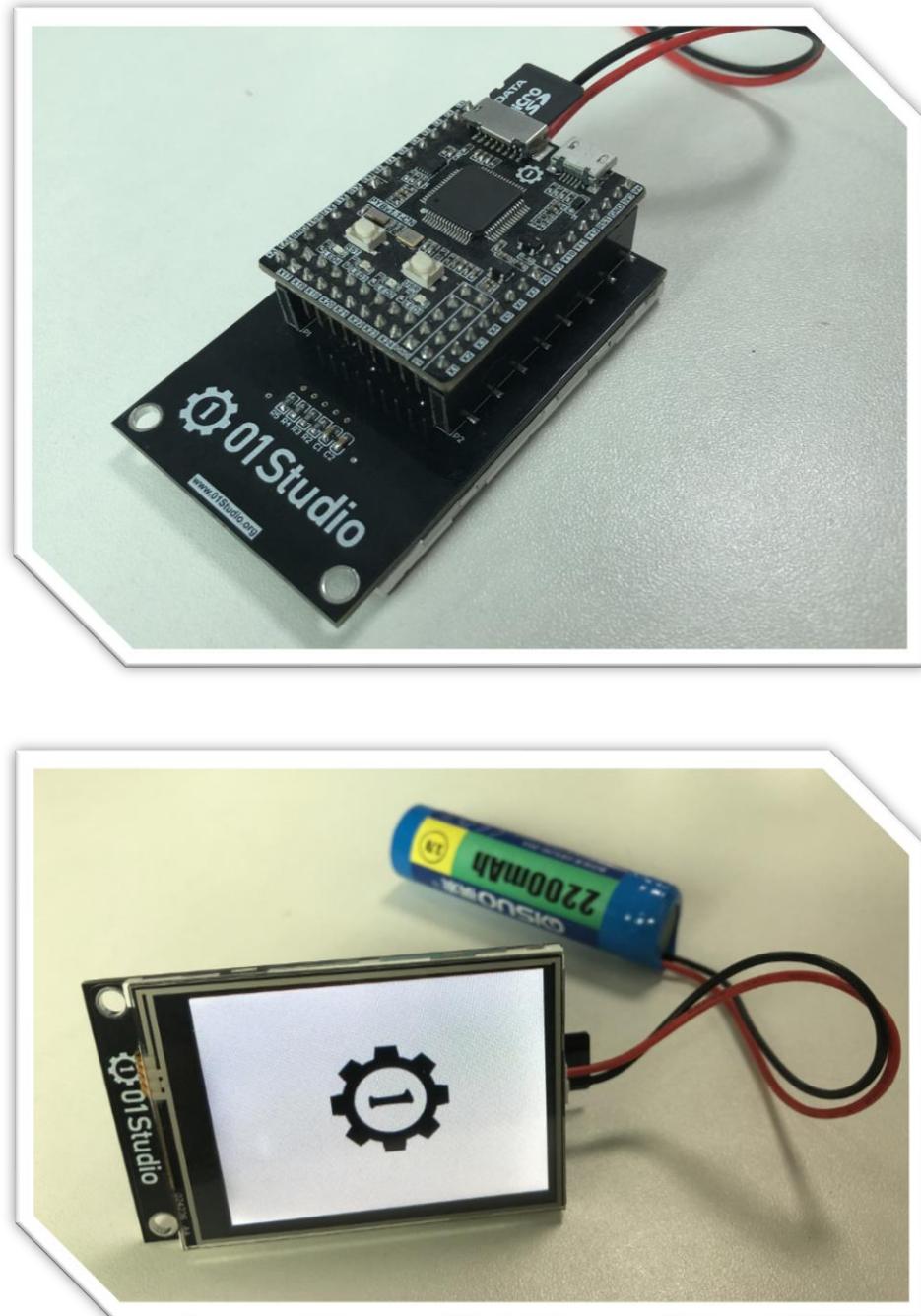


图 4-69 LCD 直接连接 pyboard v1.1-CN

- **总结：**

由于 MicroPython 的可移植性非常强，所以使用本实验代码简单修改即可用起 01Studio 配套的 1.5、1.8、2.4 寸各类 LCD，使用方法高度一致，详细可以看资料包配套代码。

通过本实验我们体验到了 LCD 使用 `micropython` 开发的简单和灵活性。我们轻松实现了 LCD 的各种常规操作，让用户将精力放在应用。相信随着 `micropython` 库函数的日益成熟，其性能和可玩性将变得更强大！

## 4.13 电阻触摸屏

- **前言:**

上一节我们学习了 LCD 实验，但 LCD 只能显示相关内容，跟人是缺乏交互的。好比我们的智能手机，如果只有显示不能触碰，那么就没有可玩性了。因此本节学习一下 3.2 寸 LCD 的电阻触摸屏使用方法。

- **实验平台:**

pyBoard 开发套件、3.2 寸显示屏（电阻触摸）。

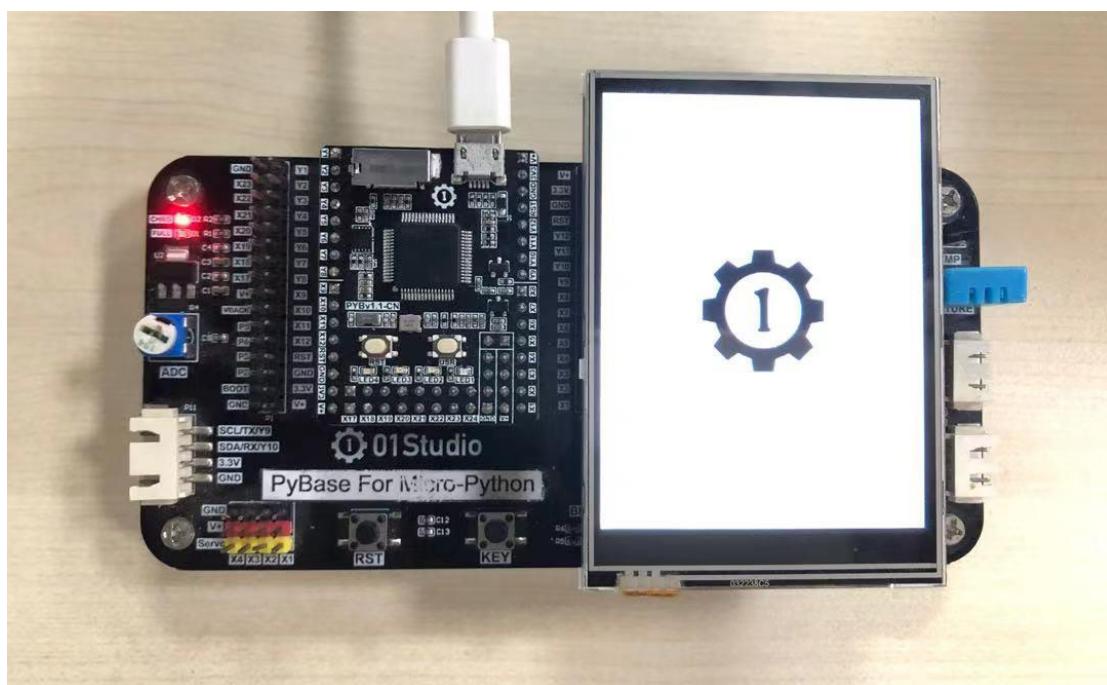


图 4-70 3.2 寸 LCD(电阻触摸)

- **实验目的:**

获取电阻触摸屏的坐标并画点标记。

- **实验讲解:**

01Studio 配套的 3.2 寸 LCD 上带电阻触摸屏，驱动芯片为 XPT2046。当手指按下时候，通过简单的编程即可返回一个坐标，我们来看看其 micropython 构造函数和使用方法：

构造函数
<b>touch.XPT2046(portrait=1)</b>
构建触摸屏对象。 <b>XPT2046</b> 表示驱动芯片型号。
【portrait】 设置屏幕方向：
<ul style="list-style-type: none"> <li>● 1 - 竖屏, 240*320, 默认</li> <li>● 2 - 横屏, 320*240, 1 基础上顺时针旋转 90°</li> <li>● 3 - 竖屏, 240*320, 1 基础上顺时针旋转 180°</li> <li>● 4 - 横屏, 320*240, 1 基础上顺时针旋转 270°</li> </ul>
使用方法
<b>XPT2046.tick_inc()</b>
手动刷新触摸。
<b>XPT2046.read()</b>
读取触摸屏数据, 返回 ( <b>states,x,y</b> )
【states】 -当前触摸状态: 0: 按下; 1: 移动; 2: 松开。
【x】 :触摸横坐标
【y】 :触摸纵坐标

表 4-18 XPT2046 触摸对象

学会了触摸对象用法后, 我们可以编程实现触摸后屏幕打点表示, 然后左上角显示当前触摸的坐标。另外再加入一个按键, 按下清空屏幕。编程流程图如下:

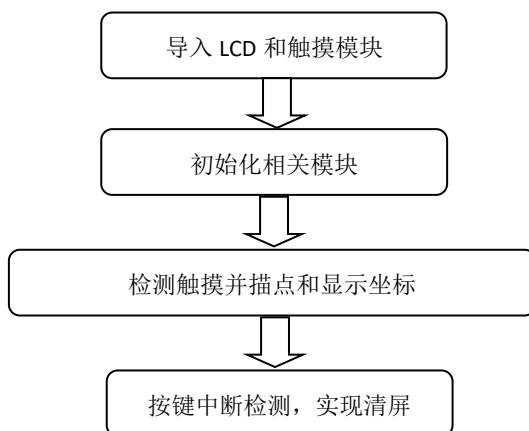


图 4-71 编程流程图

参考代码如下：

```
...
实验名称：电阻触摸屏
版本：v1.0
日期：2021.8
作者：01Studio
实验平台：pyBoard
说明：电阻触摸屏采集触摸信息
...
from touch import XPT2046
from tftlcd import LCD32
from machine import Pin
import time

#定义颜色
BLACK = (0,0,0)
WHITE = (255,255,255)
RED=(255,0,0)

#LCD 初始化
d = LCD32(portrait=1) #默认竖屏
d.fill(WHITE) #填充白色

#电阻触摸屏初始化，方向和 LCD 一致
t = XPT2046(portrait=1)

while True:

    data = t.read() #获取触摸屏坐标
```

```
print(data) #REPL 打印

#当产生触摸时

if data[0]!=2: #0: 按下; 1: 移动; 2: 松开

    #触摸坐标画圆

    d.drawCircle(data[1], data[2], 5, BLACK, fillcolor=BLACK)

    d.printStr(' (X:' +str('%03d'%data[1]) +'

Y:' +str('%03d'%data[2]) +') ',10,10,RED,size=1)

time.sleep_ms(20) #触摸响应间隔
```

### ● 实验结果：

运行程序，首次运行会自动提示进行触摸校准（电阻屏需要校准），按提示分别点击四个角落进行校准，如校准失败会自动重复。校准成功会自动保存一个“touch.cal”文件到开发板 flash，下次无须再校准。

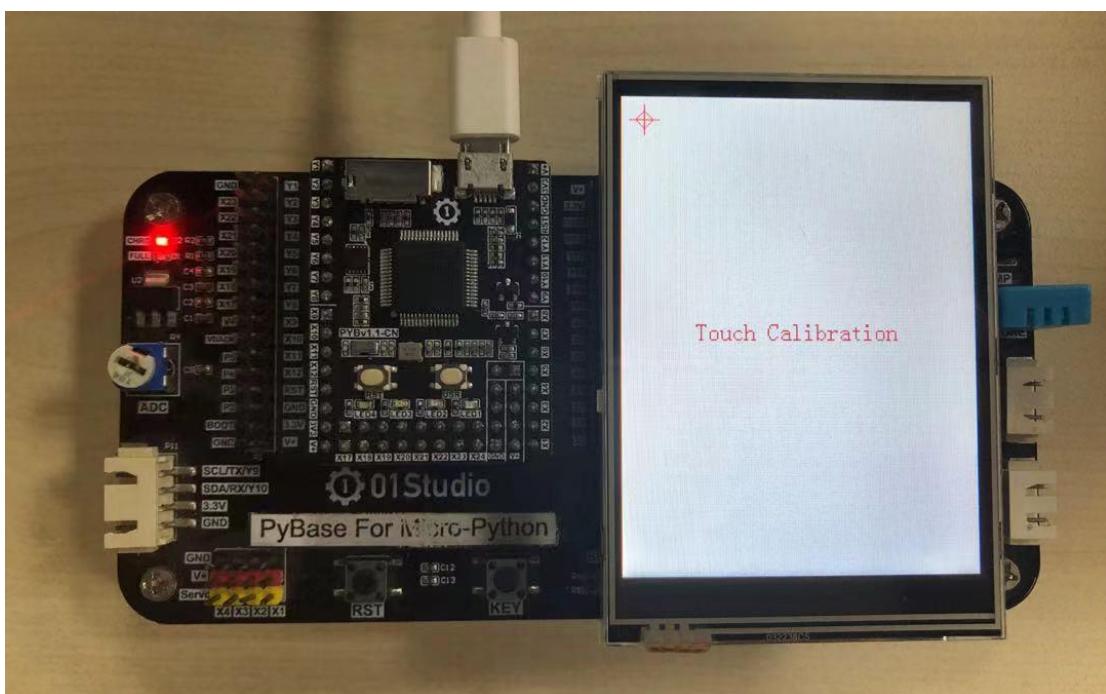


图 4-72 进入触摸校准

成功后出现空白画面，用手指触摸屏幕或者在屏幕上滑动，可以看到描点并在 LCD 左上角显示当前坐标。

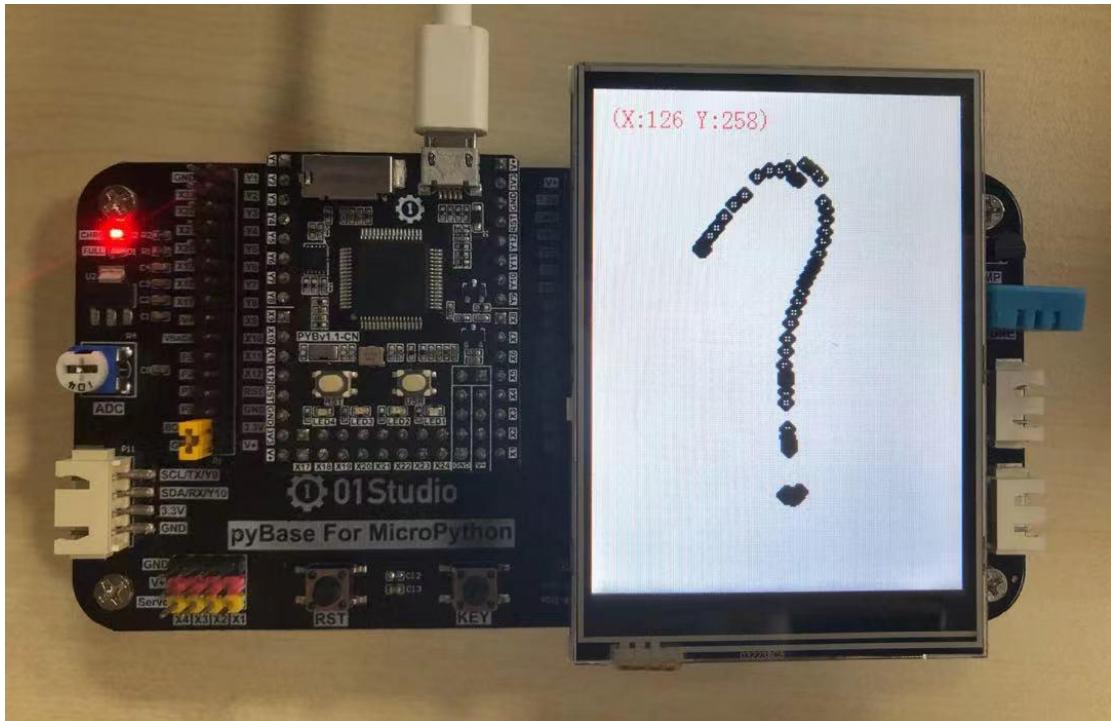


图 4-73 触摸实验

重启开发板，可以看到文件系统多了一个“touch.cal”，在运行电阻屏初始化时候会检测这个文件，如果存在则不进行校准，若想重新校准的用户可以把这个文件删除即可！

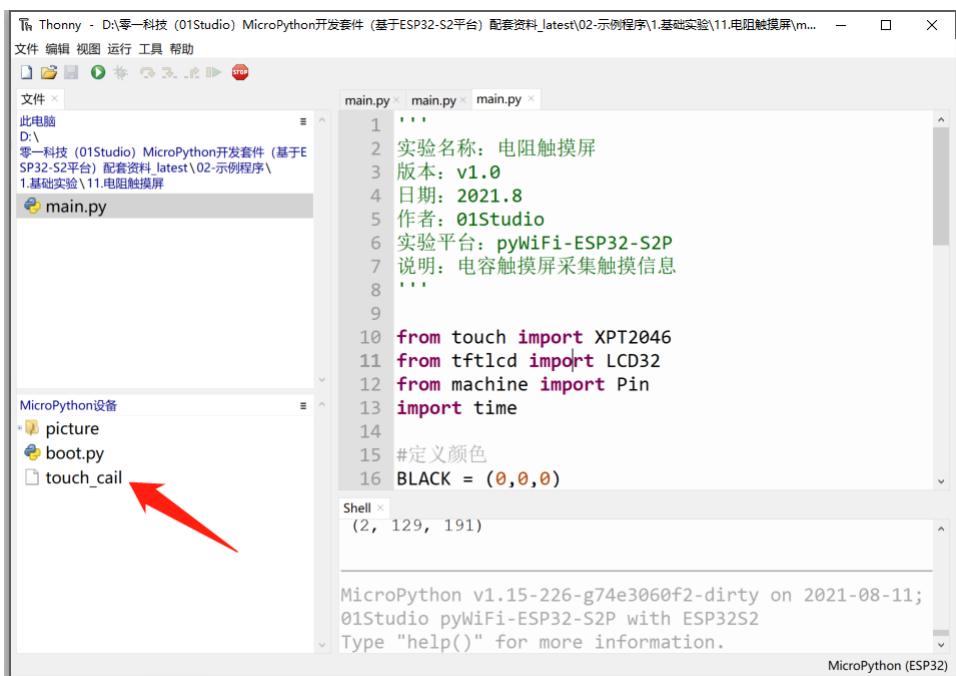


图 4-74 触摸校准文件

### ● 总结:

没有触摸屏的 LCD 就失去了灵魂，有了触摸屏，跟开发板的交互就变得有意思了。

## 4.14 触摸屏按钮

### ● 前言：

上两节我们分别学习了 LCD 显示和触摸屏操作，这一节我们就来整合一下，做一个好玩的东西，那就是生成触摸按钮。由于 pyWiFi-ESP32 开发板上只有 1 个功能按键，而使用 LCD 和触摸屏则可以创建非常多的按钮来执行我们的任务。而且操作更直观。

### ● 实验平台：

pyBoard 开发套件、3.2 寸显示屏（电阻触摸）。

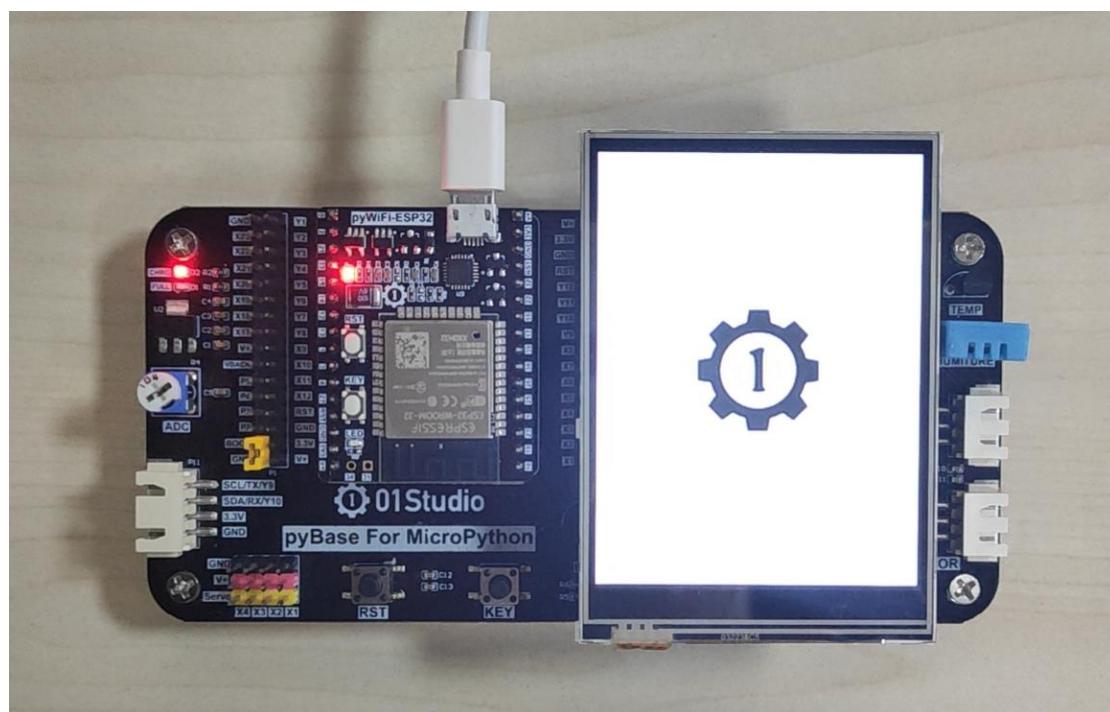


图 4-75 3.2 寸 LCD(电阻触摸)

### ● 实验目的：

生成触摸按钮，实现触摸控制 LED 灯状态变换。

### ● 实验讲解：

学习到现在的你可以思考一下，会用什么方法来实现这个功能呢？你或许会想在 LCD 上画一个填充矩形，然后获取触摸坐标，跟这个矩形位置对上后就判断这个按钮被按下了。

没错，这个原理是可行的，看似很简单，但我们还需考虑很多问题，比如触摸后用什么方式执行任务效率最高？如果区分不同按钮以及它们之间是否存在冲突。当你这么思考的时候，就是在开始构建一个简单 GUI（图形用户界面）。

还是那句，人生苦短，01studio 已经将底层封装好，用户只需要直接学会对象的使用即可。我们来看看触摸按钮对象：

构造函数
<b>gui.TouchButton(x,y,width,height,color,label,label_color,callback)</b>
构建触摸对象；定义在 <code>gui</code> 模块下。
<b>【x】</b> 按钮起始横坐标；
<b>【y】</b> 按钮起始纵坐标；
<b>【width】</b> 按钮宽度；
<b>【height】</b> 按钮高度；
<b>【color】</b> 按钮颜色；
<b>【label】</b> 按钮标签；
<b>【label_color】</b> 按钮标签颜色；
<b>【callback】</b> 按钮触发的回调函数，按下松开后触发。
<b>注意：当前最多定义 20 个，软件复位不会释放按键编号。</b>
使用方法
<b>gui.task_handler()</b>
执行所有任务。也就是执行按钮回调函数里面的代码。
<b>TouchButton.ID()</b>
获取当前触摸 ID 的编号。

表 4-19 触摸按钮对象

从上表可以看到只需要简单的语句便实现了触摸按钮的构建，我们可以构建 2 个按钮，分别控制 LED 和打印信息，编程思路如下：

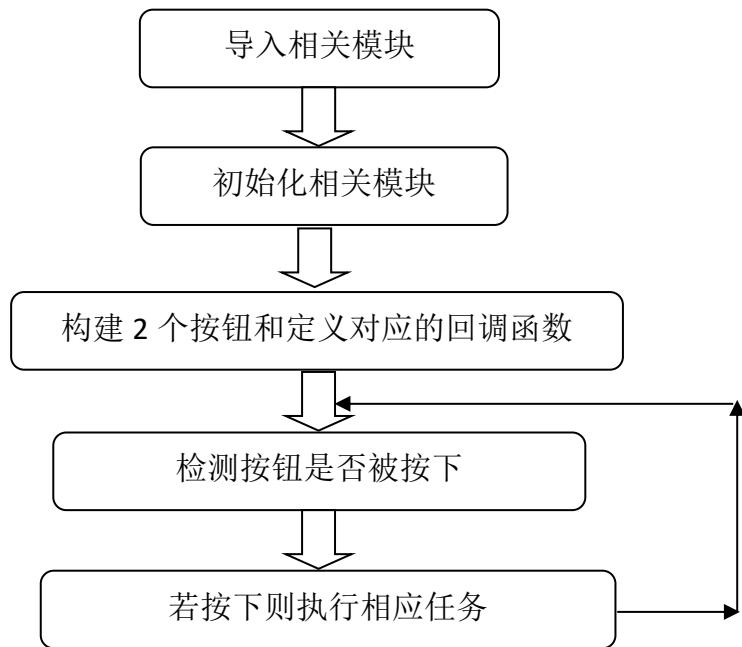


图 4-76 代码编写流程

参考代码如下：

```

...
实验名称: 触摸按钮
版本: v1.0
日期: 2021.9
作者: 01Studio
实验平台: pyBoard
说明: 编程实现触摸按钮控制 LED。
...

from tftlcd import LCD32
from touch import XPT2046
from machine import Pin
from pyb import LED,Timer
import gui,time

# 定义常用颜色

```

```

BLACK = (0,0,0)
WHITE = (255,255,255)
RED = (255,0,0)
GREEN = (0,255,0)
BLUE = (0,0,255)
ORANGE =(0xFF,0x7F,0x00) #橙色

#LCD 初始化
d = LCD32() #默认方向
d.fill(WHITE) #填充白色

#触摸屏初始化
t = XPT2046()#默认方向

#####
#定义 2 个按键和回调函数
#####
def fun1(B1):

    #LED 灯状态翻转
    LED(4).toggle()

def fun2(B2):

    print('Button is pressed!')

B1 = gui.TouchButton(80,50,80,50,BLUE,'LED',WHITE,fun1)
B2 = gui.TouchButton(80,120,80,50,RED,'Button',WHITE,fun2)

```

```
#####
#### 定时器用于触发按钮事件 ##
#####

tim_flag = 0

def count(tim):
    global tim_flag
    tim_flag = 1

tim = Timer(1,freq=50) #20ms 刷新一次
tim.callback(count)

while True:

    #执行按钮触发的任务
    if tim_flag == 1:

        t.tick_inc()
        gui.task_handler()

        tim_flag = 0
```

**提示：** **Micropython** 所有中断回调函数都不支持新的内存分配语句，否则会报错，因此可以通过定义全局变量（**global**），在回调函数修改，然后在主循环函数判断然后做出相应的控制。

### ● 实验结果：

运行代码，可以看到 LCD 显示屏上生成了 2 个按钮，点击 LED 按钮，可以看到 LED 蓝灯状态发生翻转。

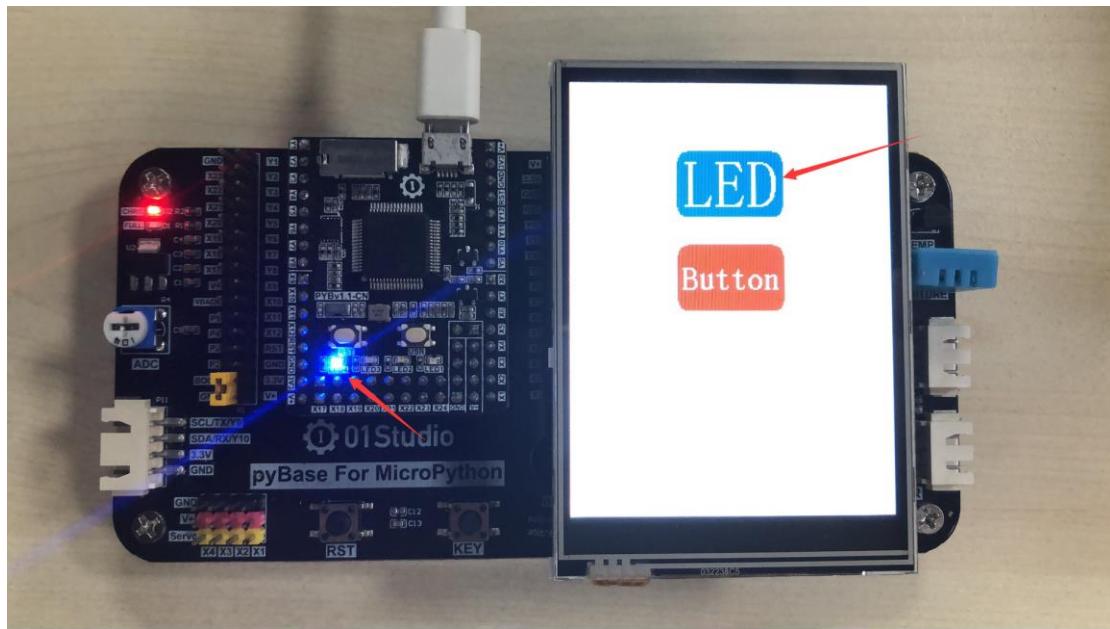


图 4-77 LED 控制

点击“Button”按钮，可以看到串口打印出“Button is Pressed!”信息。

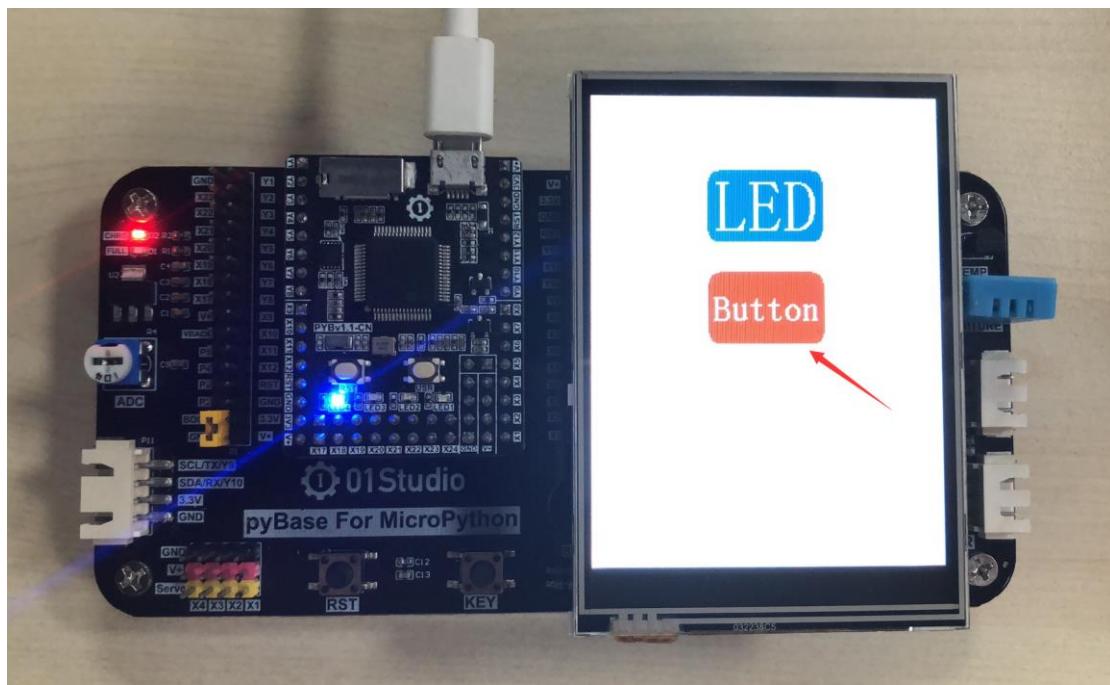


图 4-78 Button 按钮

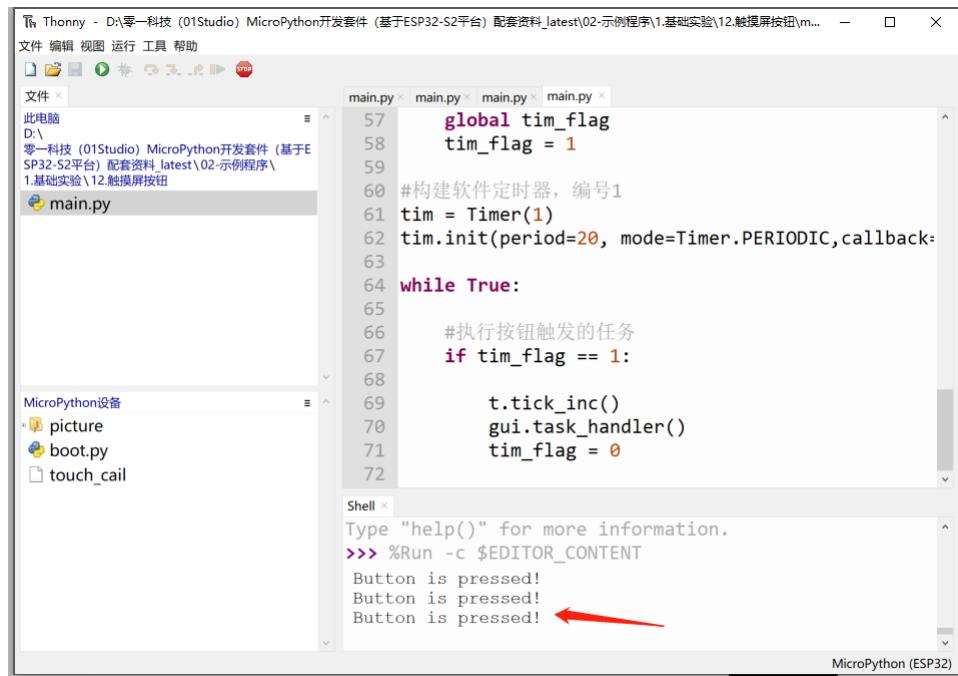


图 4-79 按钮 “Button” 回调功能

### ● 总结：

触摸按钮实现简单但却是用途非常广泛的功能，有了自定义按键，我们就可以通过触摸按键来实现所有外设设备的交互。让设备的控制变得更简单有趣。当前颜色、尺寸、标签等都为可修改项，大大提高了用户编程的灵活性。

## 第5章 传感器实验

基础实验让我们对 MicroPython 的操作和编程有了比较全面的了解，但其魅力绝不限于此。日常生活中我们会用到各式各样的外设或者传感器，还是那句，一个有经验的嵌入式开发工程师驱动一款未接触过的传感器的一般流程是：了解传感器原理、设计电路图、信号时序分析和编程。没个几天折腾不出来。

生活中有很多传感器已经是非常通用了，前人已经做好封装函数模块，我们直接调用函数即可。我们不需要将时间花在“怎么用”上，而更多的是考虑“用到什么地方”！

本章实验以最常见的传感器出发，讲述是如何通过 MicroPython 编程实现传感器应用的。实验还是基于我们的 MicroPython 开发板。而且实验例程将持续更新。

## 5.1 温度传感器 DS18B20

### ● 前言：

相信没有电子爱好者不知道 DS18B20 的，DS18B20 是常用的数字温度传感器，其输出的是数字信号，具有体积小，硬件开销低，抗干扰能力强，精度高的特点。DS18B20 数字温度传感器接线方便，封装成后可应用于多种场合，如管道式，螺纹式，磁铁吸附式，不锈钢封装式，型号多种多样。

主要根据应用场合的不同而改变其外观。封装后的 DS18B20 可用于电缆沟测温，高炉水循环测温，锅炉测温，机房测温，农业大棚测温，洁净室测温，弹药库测温等各种非极限温度场合。耐磨耐碰，体积小，使用方便，封装形式多样，适用于各种狭小空间设备数字测温和控制领域。

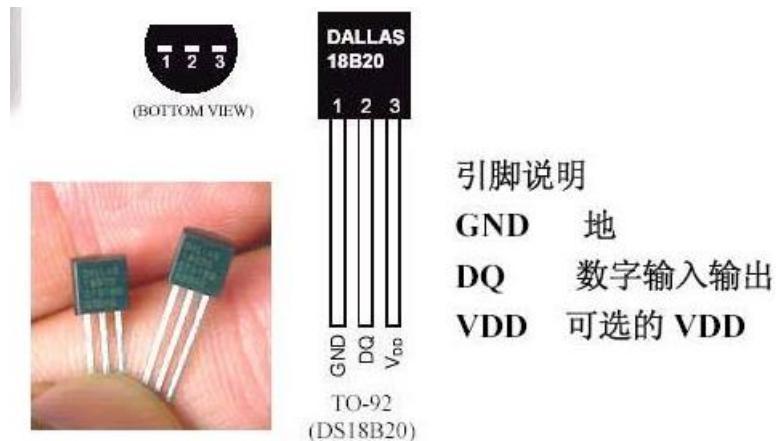


图 5-1 DS18B20 传感器



图 5-2 DS18B20 金属探头封装

- 实验平台：

MicroPython 开发套件。

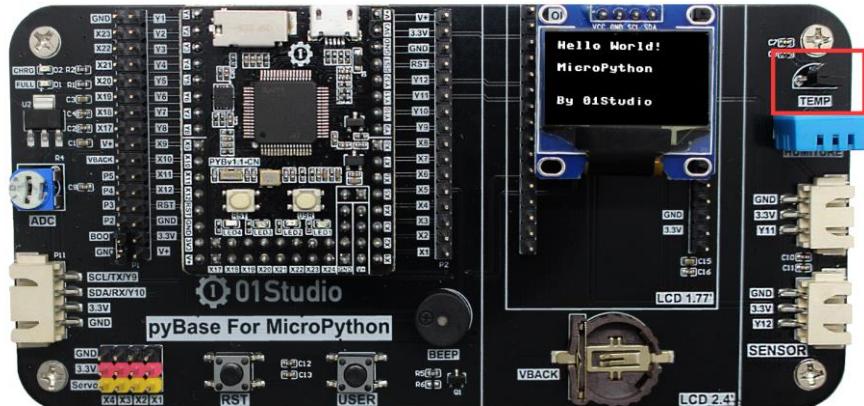


图 5-3 DS18B20 传感器在右上方

- 实验目的：

通过编程采集温度数据，并在 OLED 上显示。

- 实验讲解：

DS18B20 是单总线传感器，也就是说只占用 1 个 IO 口。我们来看看原理图：

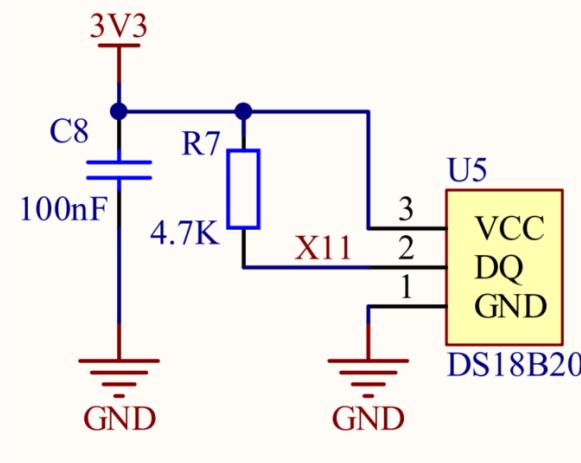


图 5-4 DS18B20 接线原理图

可以看到连接到 DS18B20 传感器是 pyBoard 的 X11 引脚。也就是说针对该引脚编写程序来驱动 DS18B20。那么我们需要自己来编写驱动么？如果你有兴趣的可以自己尝试一下。这部分我们 O1Studio 已经收集整理和编写好了，当然也包括 DS18B20。单总线模块文件是：onewire.py，DS18B20 模块的文件是 DS18X20.py，将示例程序文件夹里面以上这两个文件拷贝到 pyboard 文件系统中。

单总线模块说明如下

构造函数	说明
<b>OneWire(Pin)</b>	构造函数, Pin 为相关引脚
使用方法	说明
<b>ow=OneWire(Pin('X11'))</b>	将 X11 引脚使能单总线, 名字为 ow

表 5-1 单总线对象

DS18B20 模块说明如下:

构造函数	说明
<b>DS18X20(ow)</b>	构造函数, ow 为单总线对象名称
使用方法	说明
<b>DS18X20(ow).scan()</b>	扫描单总线上所有传感器地址, 返回列表
<b>DS18X20(ow).convert_temp()</b>	温度转换, 结果存放在地址列表中
<b>DS18X20(ow).read_temp(list[0])</b>	获取总线第一个 DS18B20 的温度值

表 5-2 DS18X20 对象

大部分场景下温度的变化不会太频繁, 我们可以每隔 1 秒采集一次, 显示精度为小数点后 2 位, 基本满足大部分应用需求。编程逻辑如下:

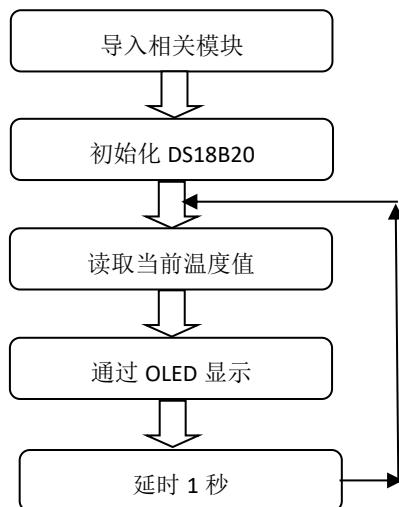


图 5-5 代码编写流程图

参考代码：

```
'''  
  
实验名称：温度传感器 DS18B20  
版本：v1.0  
日期：2019.4  
作者：01Studio  
说明：通过编程采集温度数据，并在 OLED 上显示。。  
'''  
  
#引用相关模块  
  
from pyb import delay  
from machine import Pin,I2C  
from ssd1306 import SSD1306_I2C  
from onewire import OneWire  
from ds18x20 import DS18X20  
  
#初始化相关模块  
  
i2c = I2C(sda=Pin("Y8"), scl=Pin("Y6")) # 软件模拟 I2C  
oled = SSD1306_I2C(128, 64, i2c, addr=0x3c)  
  
#初始化 DS18B20  
  
ow= OneWire(Pin('X11')) #使能单总线  
ds = DS18X20(ow)          #传感器是 DS18B20  
rom = ds.scan()            #扫描单总线上的传感器地址，支持多个传感器同时连接  
  
while True:  
  
    ds.convert_temp()      #温度采集转换  
    temp = ds.read_temp(rom[0]) #温度显示,rom[0]为第 1 个 DS18B20
```

```

#数据显示

oled.fill(0)    #清屏背景黑色

oled.text('MicroPython', 0, 0)

oled.text('Temp test:', 0, 20)

oled.text(str('%.2f'%temp)+' C', 0, 40) #显示 temp,保留 2 位小数

oled.show()

delay(1000)#延时 1 秒

```

### ● 实验结果：

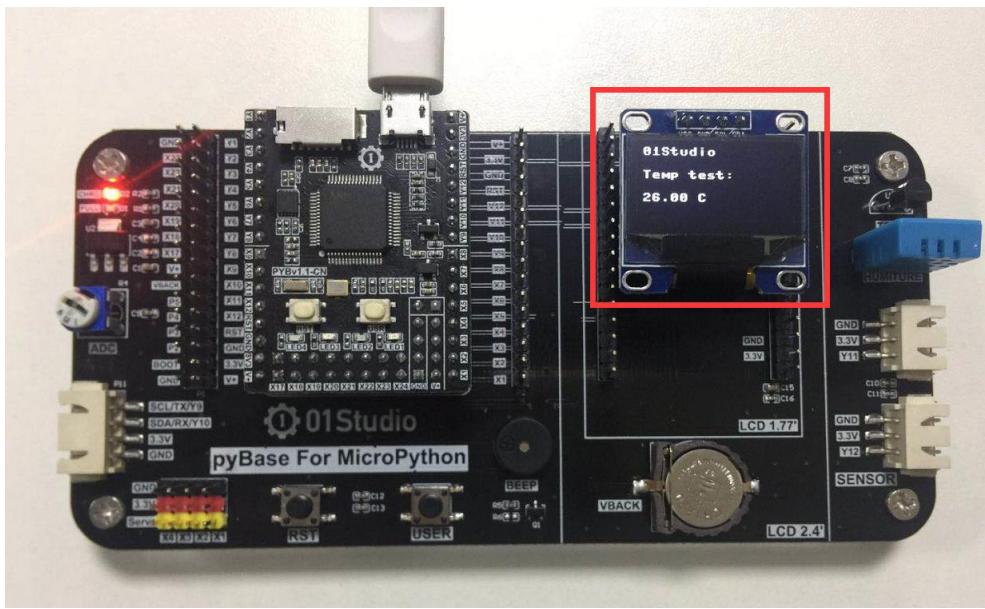


图 5-6 温度检测实验

MicroPython 开发板预留了外界传感器接口，只要接线正确就可以进行更多的传感器实验。我们将带金属探头的 DS18B20 传感器接到中间的传感器母座，其连接的 IO 是“Y11”，所以要将原程序代码的 `ow=OneWire(Pin('X19'))` 改成 `ow=OneWire(Pin('Y11'))`，保存后复位。

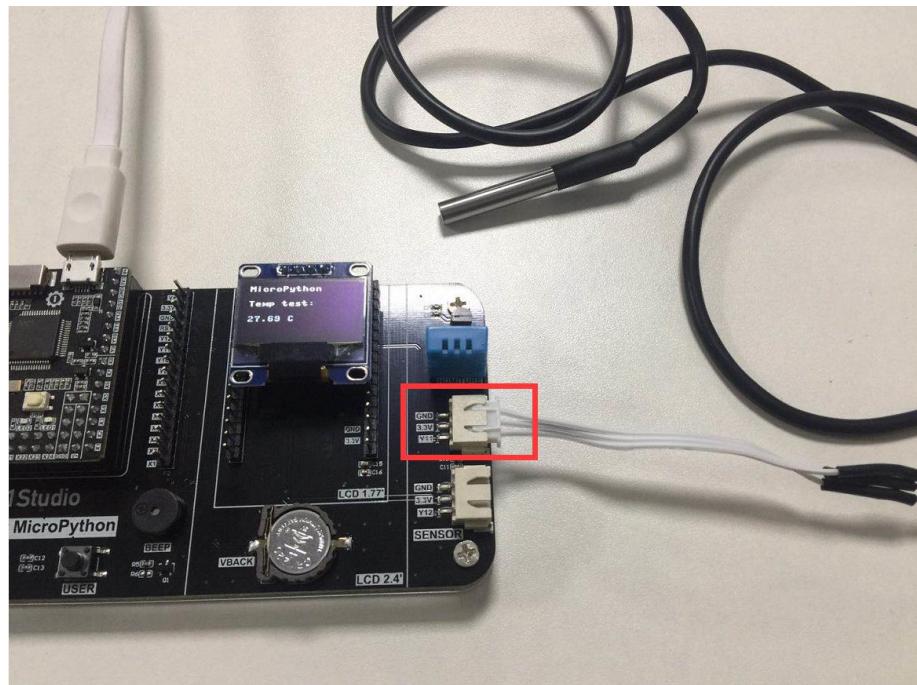


图 5-7 外接温度传感器

我们将金属探头放在冰水里面，可以见到测试到的温度是 3 点多℃。

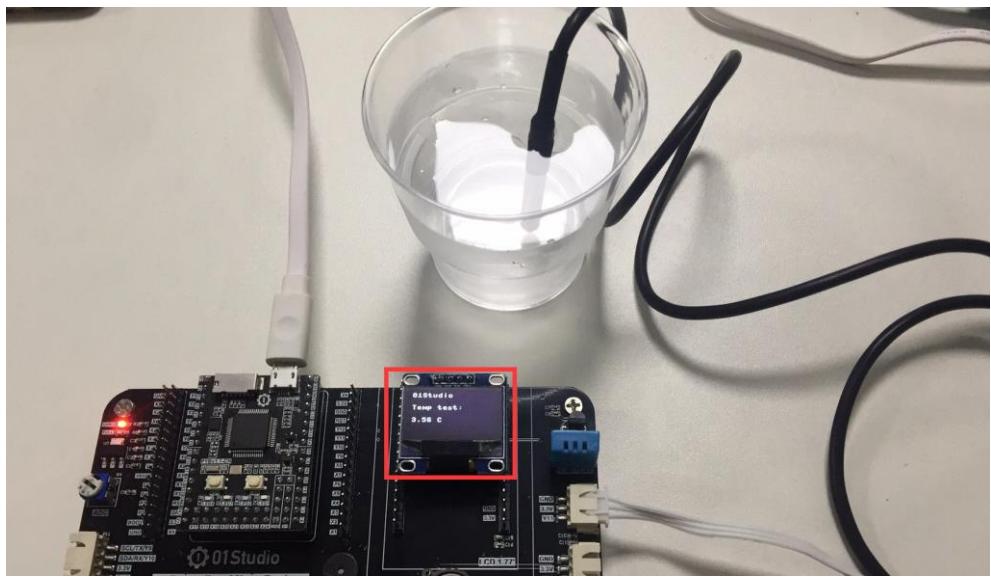


图 5-8 实现金属探头 DS18B20 测量

### ● 总结：

DS18B20 作为我们第一个实验传感器，使用 MicroPython 编程非常容易就用起来了，而且精度和稳定性丝毫没有影响。温度传感器只是一个敲门砖，接下来我们将会学习更多的传感器应用。

## 5.2 温湿度传感器 DHT11

### ● 前言：

温湿度也是我们日常非常常见的指标，我们使用的是 DHT11 数字温湿度传感器。这是一款含有已校准数字信号输出的温湿度复合传感器，它应用专用的数字模块采集技术和温湿度传感技术，确保产品具有极高的可靠性和卓越的长期稳定性。

DHT11 具有小体积、极低的功耗，信号传输距离可达 20 米以上，使其成为给类应用甚至最为苛刻的应用场合的最佳选择。产品为 4 针单排引脚封装，连接方便。

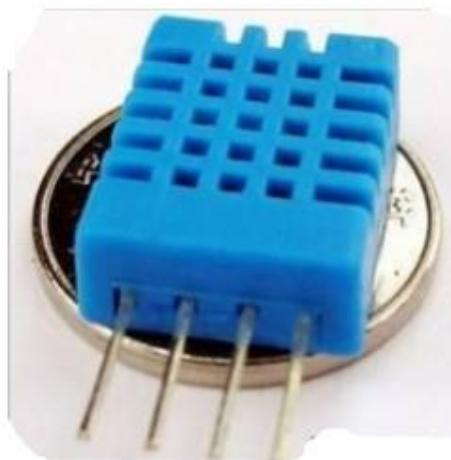


图 5-9 DHT11 温湿度传感器

### ● 实验平台：

MicroPython 开发套件。

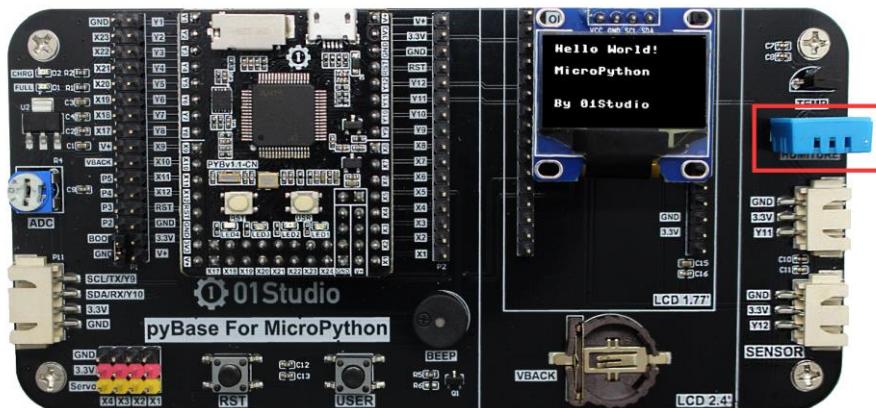


图 5-10 DHT11 传感器在右上方

- 实验目的：

通过编程采集温湿度数据，并在 OLED 上显示。

- 实验讲解：

DHT11 虽然有 4 个引脚，但其中第 3 个引脚是悬空的，也就是说 DHT11 也是单总线的传感器，只占用 1 个 IO 口。

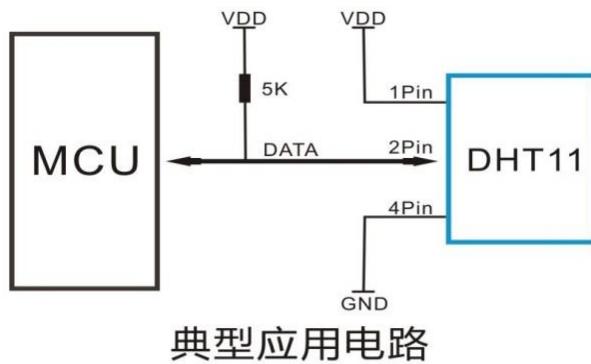


图 5-11 DHT11 应用

我们来看看 DHT11 在开发板上的接线图：

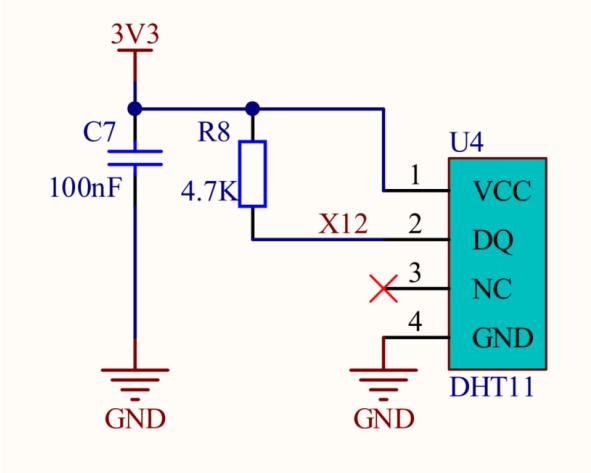


图 5-12 DHT11 接线图

可以看到 DHT11 连接到 pyboard 的 ‘X12’ 引脚，可以针对该引脚编程来驱动 DHT11 传感器，模块文件是配套示例程序下的 dht.py，需要拷贝到 pyboard 的文件系统。函数模块说明如下：

构造函数	说明
DHT11 (Pin)	指定连接 DHT11 的引脚
使用方法	说明
DHT11().measure()	温湿度数据测量
DHT11().temperature()	获取温度值
DHT11().humidity()	获取湿度值

表 5-3

建议上电先延时 1 秒，让 DHT11 稳定后再开设读取。代码编写流程如下：

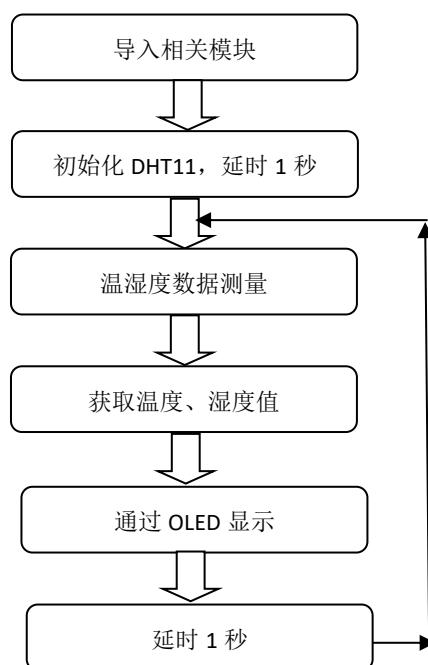


图 5-13 代码编程流程

参考代码如下：

```

...
实验名称: 温湿度传感器 DHT11
版本: v1.0
日期: 2019.4
作者: 01Studio
  
```

说明：通过编程采集温湿度数据，并在 OLED 上显示。。

```
...
#引入相关模块
from pyb import delay
from machine import Pin,I2C
from ssd1306 import SSD1306_I2C
from dht import DHT11

#初始化相关模块
i2c = I2C(sda=Pin("Y8"), scl=Pin("Y6"))
oled = SSD1306_I2C(128, 64, i2c, addr=0x3c)

#创建 DTH11 对象 dt
dt=DHT11(Pin('X12')) #'X20'连接到开发板上的 DHT11
delay(1000)           #首次启动停顿 1 秒然传感器稳定

while True:

    dt.measure()          #温湿度采集
    te=dt.temperature()   #获取温度值
    dh=dt.humidity()      #获取湿度值

    oled.fill(0) #清屏背景黑色
    oled.text('01Studio', 0, 0)
    oled.text('DHT11 test:',0,15)

    #温度显示
    oled.text(str(te)+ ' C',0,40)
```

```
#湿度显示  
oled.text(str(dh) + ' %', 48, 40)  
  
oled.show()  
  
delay(1000) #每隔 1 秒采集一次
```

### ● 实验结果：

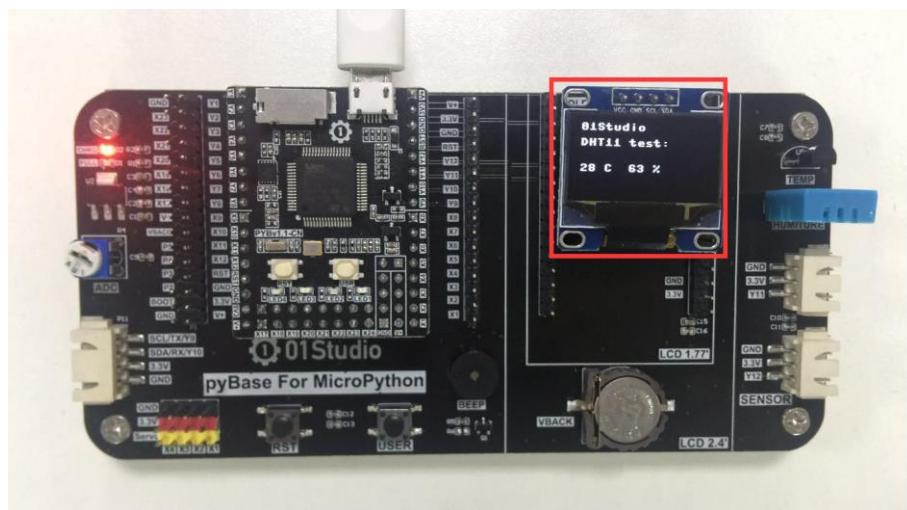


图 5-14 DHT11 实验结果

### ● 总结：

通过本节学习我们学会了使用 MicroPython 来驱动 DHT11 温湿度传感器，DHT11 性价比比较高，是很适合学习使用的，但精度和响应速度有点低，需要更高要求应用的用户可以使用 DHT22 或者其他更高级的传感器。

## 5.3 人体感应传感器

### ● 前言：

人体感应传感器，在室内安防应用非常普遍，其原理是由探测元件将探测到人体的红外辐射转变成微弱的电压信号，经过放大后输出。为了提高探测器的探测灵敏度以增大探测距离，一般在探测器的前方装设一个塑料的菲涅尔透镜，它和放大电路相配合，可将信号放大 70dB 以上，这样就可以测出 5~10 米范围内人的行动。

### ● 实验平台：

MicroPython 开发套件和人体感应传感器模块。

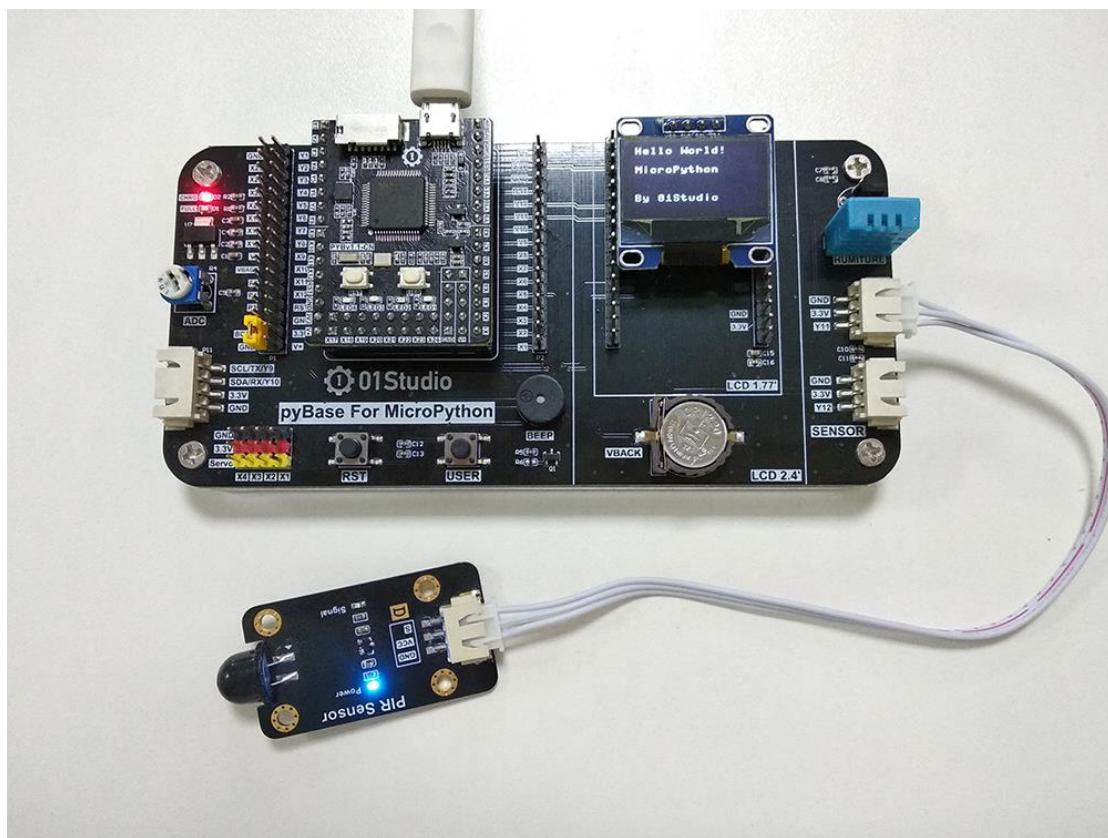


图 5-15 开发套件和传感器接线

### ● 实验目的：

通过外部中断编程来检测人体感应模块，当有人出现时候 OLED 通过“Get People!!!”闪烁提示。

## ● 实验讲解：

我们先来看看人体感应传感器的介绍：

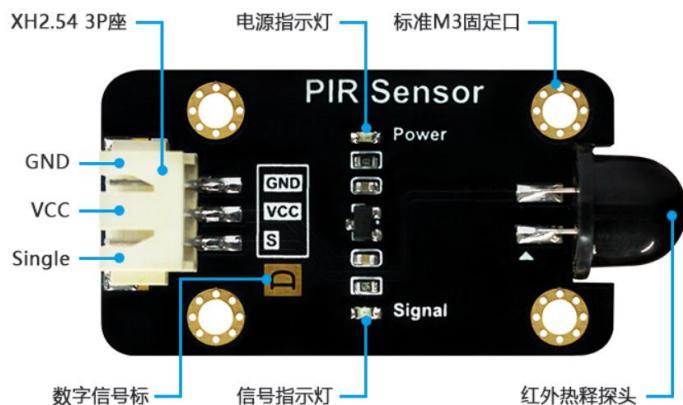


图 5-16 人体感应传感器模块

功能参数	
供电电压	3.3-5V
工作电流	<20 mA
工作温度	-20 至 65°C
接口定义	XH2.54 防呆接口 (3Pin) 【GND、VCC、Single】
输出信号	数字信号： 检测到人体：高电平 3.3V；持续 3-8 秒 未检测到人体：低电平 0V。
感应角度	100°
感应距离	8 米
模块尺寸	4.5*2.5cm

表 5-4 人体感应传感器模块参数说明

从上表可以知，当检测到人体红外时候，传感器的输出信号为高电平并持续 3-5 秒。如下图所示：

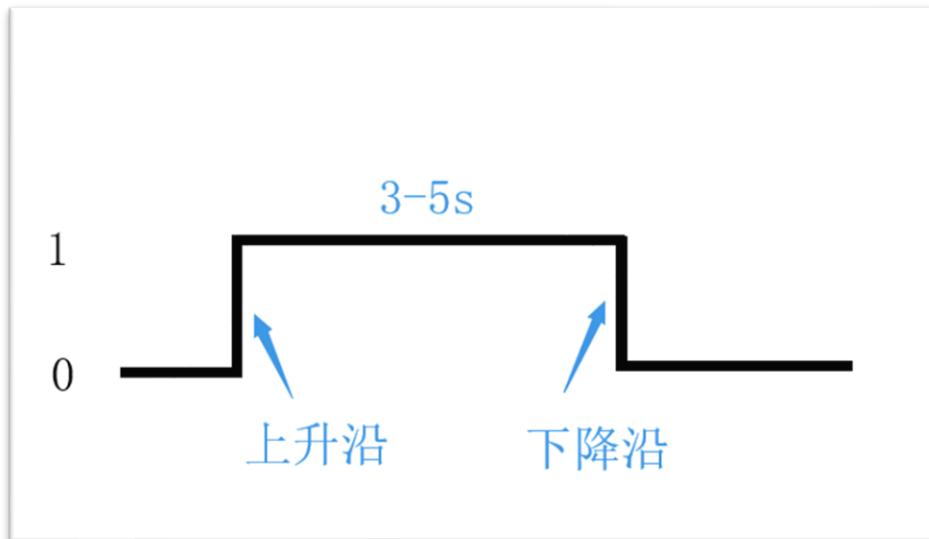


图 5-17 人体感应传感器电平变化

由此可见，可以使用外部中断结合上升沿的出发方式来编程实现相关功能。传感器的输出引脚连接 Sensor1 接口，即连接到 pybaord 的“Y11”引脚。编程方法可以是当“Y11”引脚产生中断时候，说明传感器检测到人体红外线，此时可以在 OLED 上显示相关信息。

关于外部中断的编程方法可以参考基础实验的外部中断章节，这里不再重复，具体的编程流程图如下：

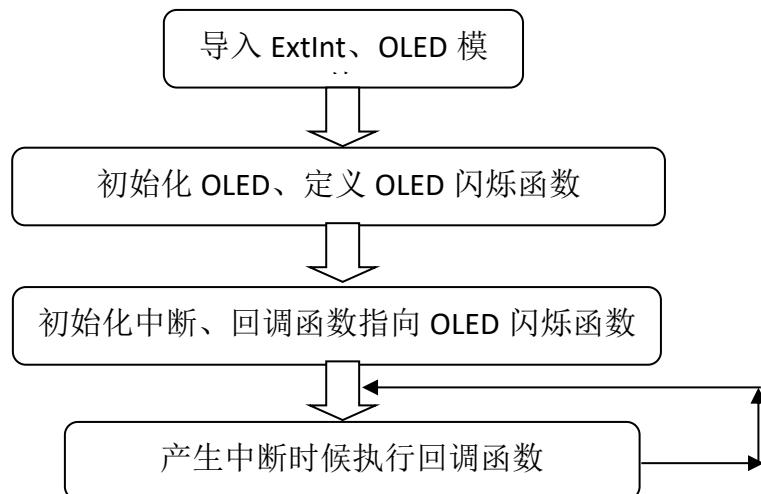


图 5-18 代码编写流程

参考例程代码如下：

```
...
实验名称：人体感应传感器
版本：v1.0
日期：2019-5-1
作者：01Studio
社区：www.01studio.org
...

#导入相关模块
from pyb import ExtInt
from machine import I2C,Pin
from ssd1306 import SSD1306_I2C

#初始化 OLED 模块
i2c = I2C(sda=Pin("Y8"), scl=Pin("Y6"))
oled = SSD1306_I2C(128, 64, i2c, addr=0x3c)

#OLED 初始信息显示
oled.fill(0) # 清屏背景黑色
oled.text("01Studio", 0, 0) # 写入第 1 行内容
oled.text("Human body test:", 0, 15) # 写入第 2 行内容
oled.show() # OLED 执行显示

def Display(): #Get People 闪烁 5 次效果！

    for i in range(5):
        oled.fill(0) # 清屏背景黑色
        oled.text("01Studio", 0, 0) # 写入第 1 行内容
```

```

oled.text("Human body test:", 0, 15) # 写入第 2 行内容
oled.text("Get People!!!", 0, 40) # 写入第 3 行内容
oled.show() # OLED 执行显示
pyb.delay(500)

oled.fill(0) # 清屏背景黑色
oled.text("01Studio", 0, 0) # 写入第 1 行内容
oled.text("Human body test:", 0, 15) # 写入第 2 行内容
oled.text(" ", 0, 40) # 写入第 3 行内容
oled.show() # OLED 执行显示
pyb.delay(500)

callback=lambda e: Display() #回调函数指向 Display()函数
#上升沿触发，打开上拉电阻
ext = ExtInt(Pin('Y11'), ExtInt.IRQ_RISING, Pin.PULL_UP, callback)

```

### ● 实验结果：

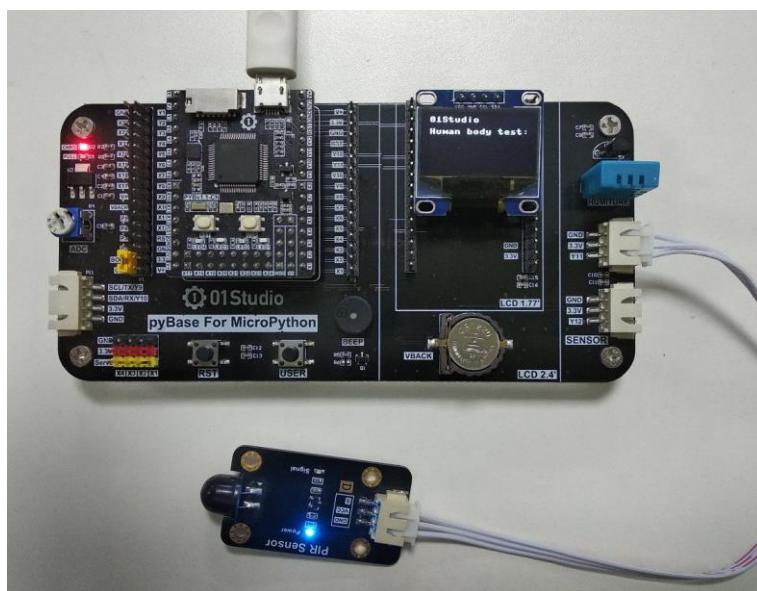


图 5-19 没有检测到到人体

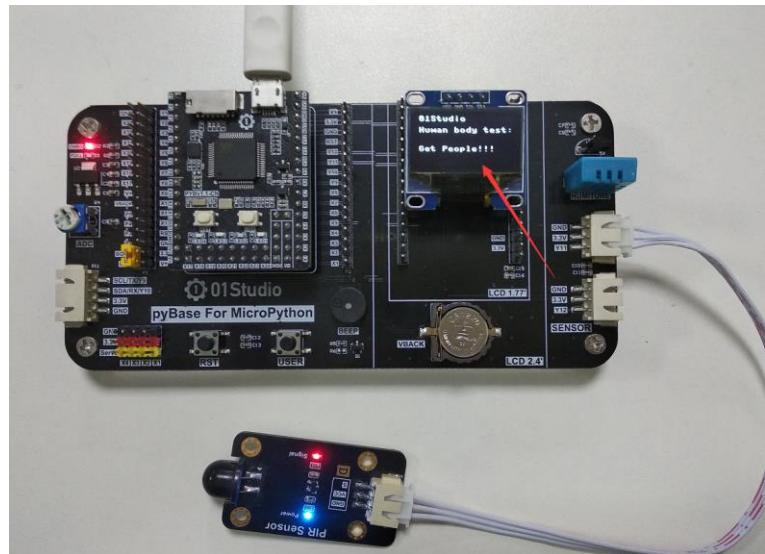


图 5-20 检测到人体，OLED 闪烁

### ● 总结：

本节通过简单的中断方式便实现了对人体感应传感器的检测，人体感应传感器的应用非常广泛，特别是在安防领域，结合其它硬件模块可以实现当发现有人入侵时，执行发出警报、实现远程提醒等功能。

## 5.4 光敏传感器

- **前言：**

光敏传感器实际是一个可以检测光照强度的传感器，可以应用于我们日常生活中植物光照强度、室内光线检测、以及某些场合的亮度检测。传感器的原理就是将外界模拟变化的信号转变成数字信号（电压值）让单片机出来，处理方式就是常用的 ADC（模拟数字转换）。

- **实验平台：**

MicroPython 开发套件和光敏传感器模块。

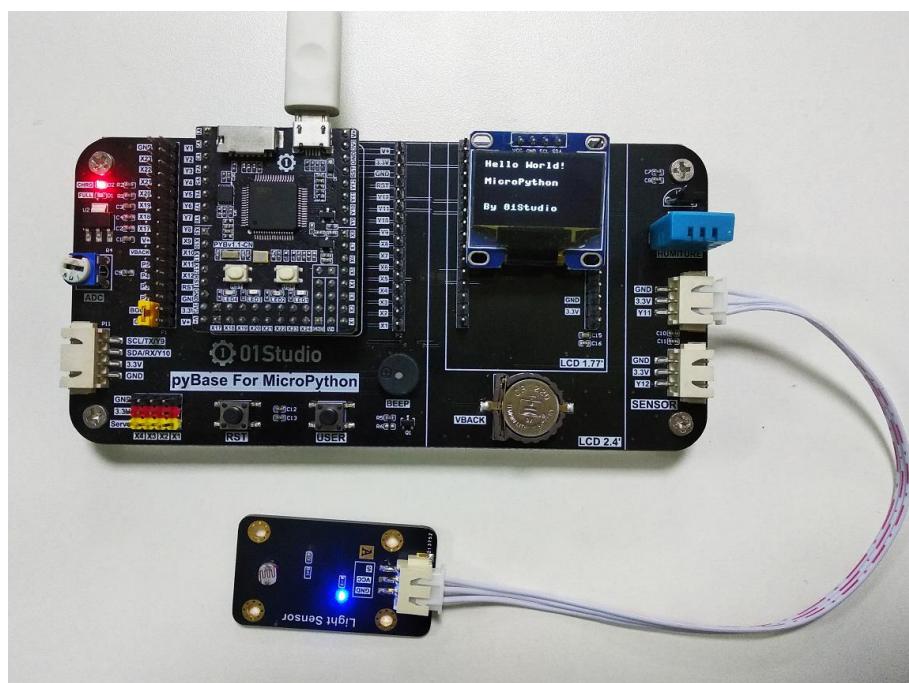


图 5-21 开发套件和光敏传感器接线

- **实验目的：**

采集当前环境的光照强度并在 OLED 显示，显示方式为：Bright-强，Normal-中等，Weak-弱。

- **实验讲解：**

我们先来看看光敏传感器模块的介绍：

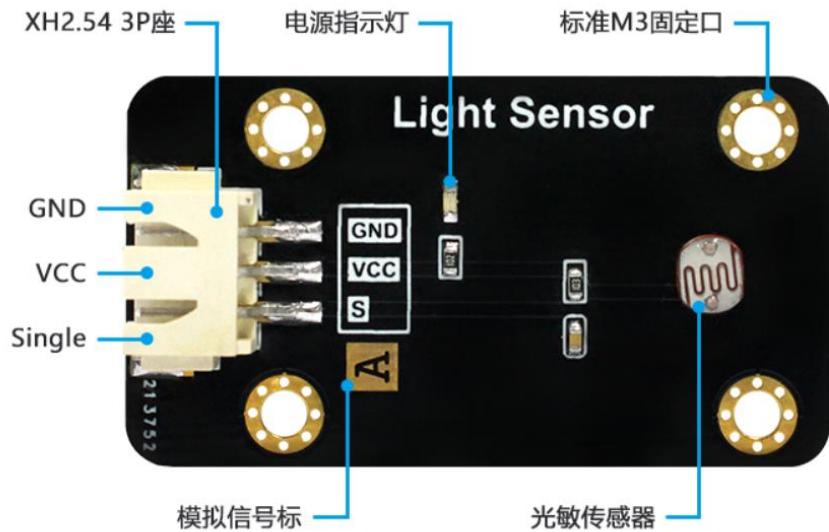


图 5-22 光敏传感器模块

功能参数	
供电电压	3.3-5V
工作电流	<20 mA
工作温度	-20 至 85°C
接口定义	XH2.54 防呆接口 (3Pin) 【GND、VCC、Single】
输出信号	模拟信号: 0-3.3V (VCC=3.3V 时)
模块尺寸	4.5*2.5 cm

表 5-5

从上表可以看到，光敏传感器输出的是模拟信号：0-3.3V，这代表了外界的光照强度。接近 0V 时光线最强，接近 3.3V 时光线最弱。因此我们可以使用基础实验-ADC 学习过的内容来编程。

光敏传感器接在传感器接口 1，对应的引脚是“Y11”。ADC 使用 12bit 精度，即最大值为  $2^{12}-1=4095$ 。然后我们将检测到的数值 0-4095 分成三段，分别代表光线强：【0-1365】，中等：【1365-2730】，弱：【2730-4095】。当然开发者可以根据自己实际情况来调整数值。编程流程图如下：

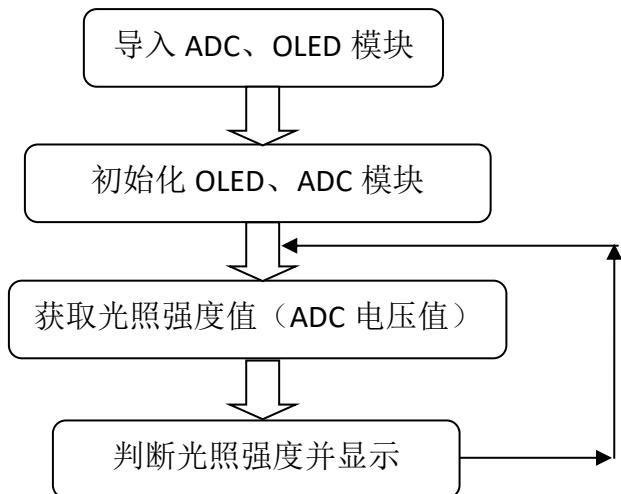


图 5-23 代码编写流程

参考程序代码如下：

```

...
实验名称: 光敏传感器
版本: v1.0
日期: 2019.5
作者: 01Studio 【www.01Studio.org】
说明: 通过光敏传感器对外界环境光照强度测量并显示。
...

#导入相关模块
import pyb
from machine import Pin,I2C
from ssd1306 import SSD1306_I2C

#初始化相关模块
i2c = I2C(sda=Pin("Y8"), scl=Pin("Y6"))
oled = SSD1306_I2C(128, 64, i2c, addr=0x3c)

```

```
#初始化 ADC,Pin='Y11'
Light = pyb.ADC('Y11')

while True:

    oled.fill(0)  # 清屏显示黑色背景
    oled.text('01Studio', 0, 0)  # 首行显示 01Studio
    oled.text('Light test:', 0, 15)      # 次行显示实验名称

    value=Light.read() #获取 ADC 数值

    #显示数值
    oled.text(str(value)+ ' (4095)',0,40)
    #计算电压值, 获得的数据 0-4095 相当于 0-3V, ('%.2f'%) 表示保留 2 位小数
    oled.text(str('%.2f'%(value/4095*3.3))+ ' V',0,55)

    #判断光照强度, 分 3 档显示。
    if 0 < value <=1365:
        oled.text('Bright', 60, 55)

    if 1365 < value <= 2730:
        oled.text('Normal', 60, 55)

    if 2730 < value <= 4095:
        oled.text('Weak ', 60, 55)

    oled.show()
    pyb.delay(1000)
```

- **实验结果：**

用手遮挡住光敏传感器，可以见到 OLED 显示光线强势是弱 Weak:

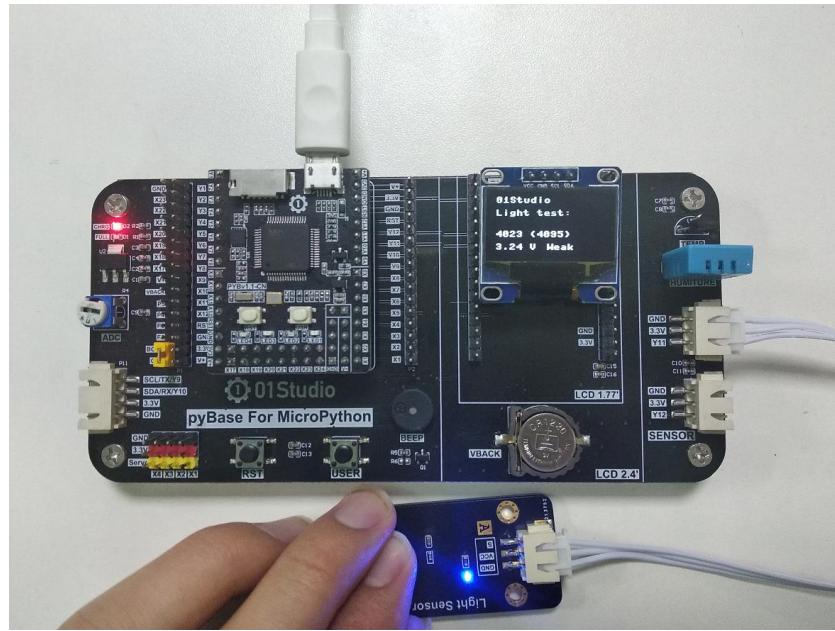


图 5-24 光线强度弱

打开手机手电筒，照在光敏传感器上，可以见到光线强度是强 Bright:

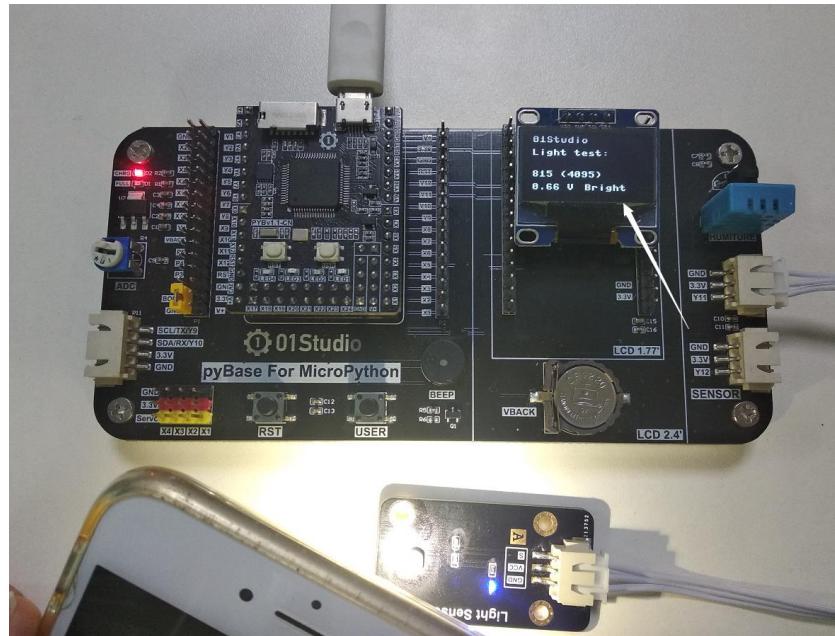


图 5-25 光线强度强

- **总结：**

从实验可以看到，光敏传感器背后的原理是对 ADC 的应用，实现了改功能后。我们可以自行扩展深入，制作自己喜欢的电子产品。

## 5.5 土壤湿度传感器

- **前言：**

土壤湿度传感器用于检测盆栽泥土的湿度，当泥土干枯时候，我们就需要给植物浇水了。这个用途非常广泛，如自动灌溉。

- **实验平台：**

MicroPython 开发套件和光敏传感器模块。



图 5-26 开发套件和传感器连接图

- **实验目的：**

采集盆栽土壤的的光照强度并在 OLED 显示，显示方式为：Dry-干，Normal-中等，Wet-湿。

- **实验讲解：**

我们先来看看土壤湿度传感器模块的介绍：

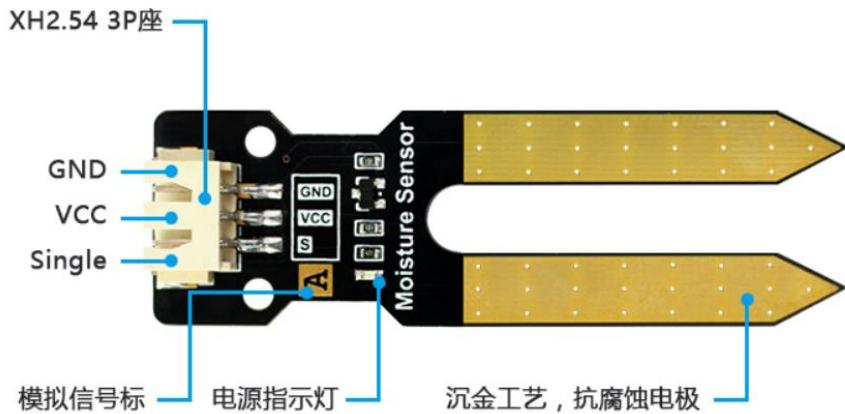


图 5-27 土壤湿度传感器模块

功能参数	
供电电压	3.3-5V
工作电流	<20 mA
工作温度	-40 至 85°C
接口定义	XH2.54 防呆接口（3Pin）【GND、VCC、Single】
输出信号	模拟信号：0-3.3V （VCC=3.3V 时）
模块尺寸	6.6*2.0cm

表 5-6

从上表可以看到，土壤湿度传感器输出的是模拟信号：0-3.3V，这代表土壤的湿度情况。接近 0V 时湿度为干燥，接近 3.3V 时，湿度情况为湿润。因此我们可以使用基础实验-ADC 学习过的内容来编程。

土壤湿度传感器接在传感器接口 1，对应的引脚是“Y11”。ADC 使用 12bit 精度，即最大值为  $2^{12}-1=4095$ 。然后我们根据实际测试数据将检测到的数值 0-4095 分成三段，分别代表土壤干燥：【0-1247】，中等：【1247-2238】，湿润：【2238-4095】。当然开发者可以根据自己实际情况来调整数值。编程流程图如下：

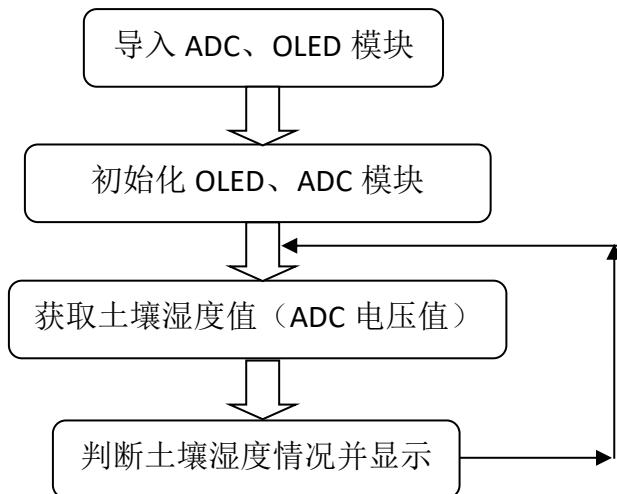


图 5-28 代码编写流程

参考程序代码如下：

```

...
实验名称：土壤湿度传感器
版本：v1.0
日期：2019.5
作者：01Studio 【www.01Studio.org】
说明：通过土壤湿度传感器对土壤湿度测量并显示。
...

#导入相关模块
import pyb
from machine import Pin,I2C
from ssd1306 import SSD1306_I2C

#初始化相关模块
i2c = I2C(sda=Pin("Y8"), scl=Pin("Y6"))
oled = SSD1306_I2C(128, 64, i2c, addr=0x3c)

```

```
#初始化 ADC,Pin='Y11'
Soil = pyb.ADC('Y11')

while True:

    oled.fill(0) # 清屏显示黑色背景
    oled.text('01Studio', 0, 0) # 首行显示 01Studio
    oled.text('Soil test:', 0, 15)      # 次行显示实验名称

    value=Soil.read() #获取 ADC 数值

    #显示数值
    oled.text(str(value)+' (4095)',0,40)
    #计算电压值，获得的数据 0-4095 相当于 0-3V，('%.2f'%) 表示保留 2 位小数
    oled.text(str('%.2f'%(value/4095*3.3))+' V',0,55)

    #判断土壤湿度，分 3 档显示。
    if 0 <= value <=1247:
        oled.text('Dry', 60, 55)

    if 1247 < value <= 2238:
        oled.text('Normal', 60, 55)

    if 2238 < value <= 4095:
        oled.text('Wet ', 60, 55)

    oled.show()
    pyb.delay(1000)
```

- 实验结果：

将传感器插到干燥的土壤中，可以见到 OLED 显示干燥-Dry:



图 5-29 干燥的土壤

从传感器侧面往土壤浇水，让土壤变得湿润。**注意不要浇到传感器电路上！**



图 5-30 浇水

可以看到 OLED 显示湿润-Wet:



图 5-31 湿润的土壤

● 总结:

从实验可以看到，土壤湿度传感器背后的原理是对 ADC 的应用，实现了该功能后。我们可以自行扩展深入，制作自己喜欢的电子产品。

## 5.6 水位传感器

- **前言：**

水位（液位）传感器是一款简单易用、小巧轻便、水位/水滴识别检测传感器，传感器通过一系列的暴露的平行导线线迹测量其水滴/水量大小使得传感器输出的模拟信号产生相应的变化，轻松完成水量到模拟信号的转换，从而判断水位。

- **实验平台：**

MicroPython 开发套件和水位传感器模块。

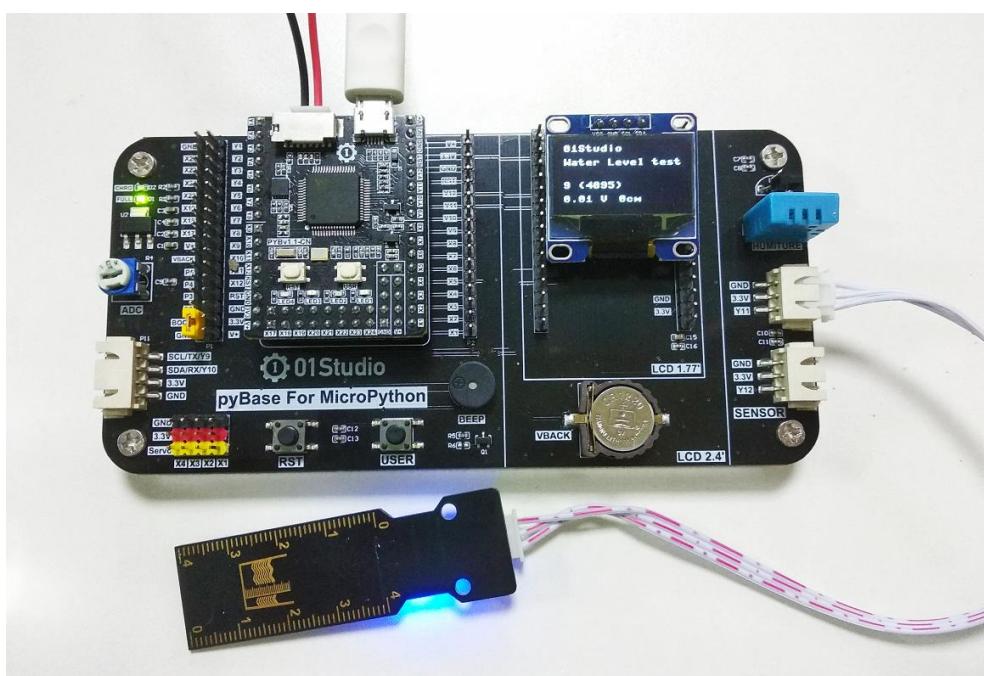


图 5-32 开发套件和水位传感器接线图

- **实验目的：**

测量水位高度并在 OLED 显示，显示方式为：0-4cm。

- **实验讲解：**

我们先来看看水位传感器的介绍：

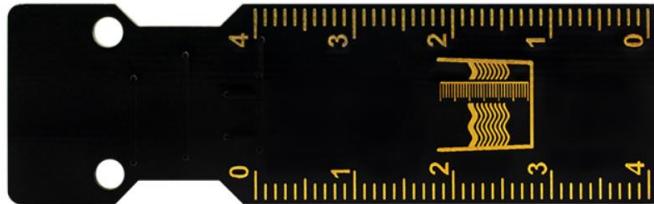
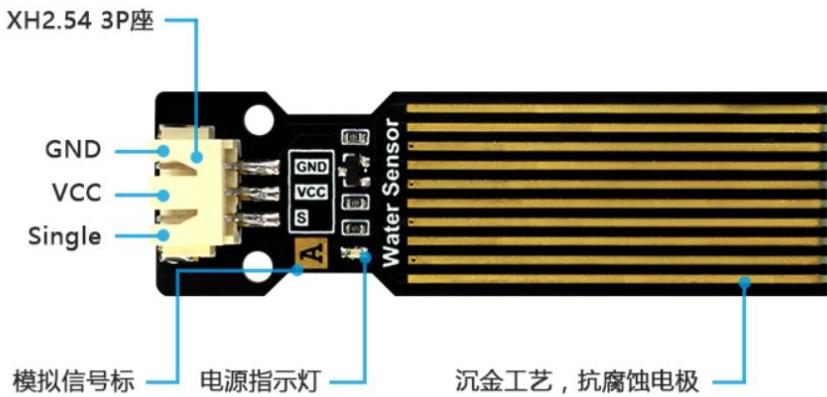


图 5-33 水位传感器模块正反面

功能参数	
供电电压	3.3-5V
工作电流	<20 mA
工作温度	0 至 60°C
接口定义	XH2.54 防呆接口 (3Pin) 【GND、VCC、Single】
输出信号	模拟信号: 0-3.3V (VCC=3.3V 时)
模块尺寸	6.6*2.0cm
注意事项	水位传感器接口旁元件不能接触水，否则容易短路。所以测量时候要注意液体水位的高度。

表 5-7

从上表可以看到，水位传感器的量程为 4cm，输出的是模拟信号：0-3.3V，这代表水位高度情况。接近 0V 时湿度为 0cm 或没有放进液体，电压升高，意味着水位升高。因此我们可以使用基础实验-ADC 学习过的内容来编程。

水位传感器接在传感器接口 1，对应的引脚是“Y11”。ADC 使用 12bit 精度，

即最大值为  $2^{12}-1=4095$ 。我们使用的是普通桶装水，根据实际测试数据将检测到的数值 0-4095 分成 5 档。分别代表水位高度 0cm: 【0-600】，1cm: 【600-900】，2cm: 【900-1200】，3cm: 【1200-1300】，4cm: 【>1300】。

当然开发者可以根据自己测量液体不同或其它情况来调整数值。编程流程图如下：

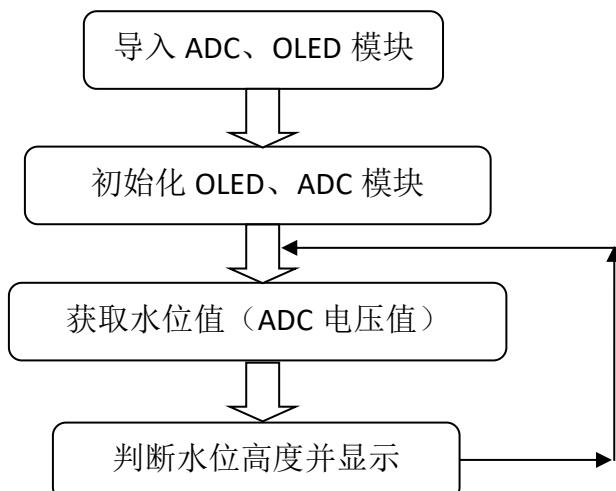


图 5-34 代码编写流程

参考程序代码如下：

```
...
实验名称: 水位传感器
版本: v1.0
日期: 2019.5
作者: 01Studio 【www.01Studio.org】
说明: 通过水位传感器对水位测量并显示。
...
#导入相关模块
import pyb
from machine import Pin,I2C
from ssd1306 import SSD1306_I2C
```

```

#初始化相关模块

i2c = I2C(sda=Pin("Y8"), scl=Pin("Y6"))

oled = SSD1306_I2C(128, 64, i2c, addr=0x3c)

#初始化 ADC,Pin='Y11'

Water_level = pyb.ADC('Y11')

while True:

    oled.fill(0) # 清屏显示黑色背景

    oled.text('01Studio', 0, 0) # 首行显示 01Studio

    oled.text('Water Level test', 0, 15) # 次行显示实验名称

    value=Water_level.read() #获取 ADC 数值

    #显示数值

    oled.text(str(value) + ' (4095)', 0, 40)

    #计算电压值，获得的数据 0-4095 相当于 0-3V，('%.2f'%') 表示保留 2 位小数

    oled.text(str('%.2f'%(value/4095*3.3)) + ' V', 0, 55)

    #判断水位，分 5 档显示，0-4cm

    if 0 <= value <=600:

        oled.text('0cm', 60, 55)

    if 600 < value <= 900:

        oled.text('1cm', 60, 55)

    if 900 < value <= 1200:

        oled.text('2cm', 60, 55)

```

```
if 1200 < value <= 1300:  
    oled.text('3cm', 60, 55)  
  
if 1300 < value:  
    oled.text('4cm', 60, 55)  
  
oled.show()  
pyb.delay(1000)
```

### ● 实验结果：

当没插入水中时，OLED 显示当前水位为 0cm:

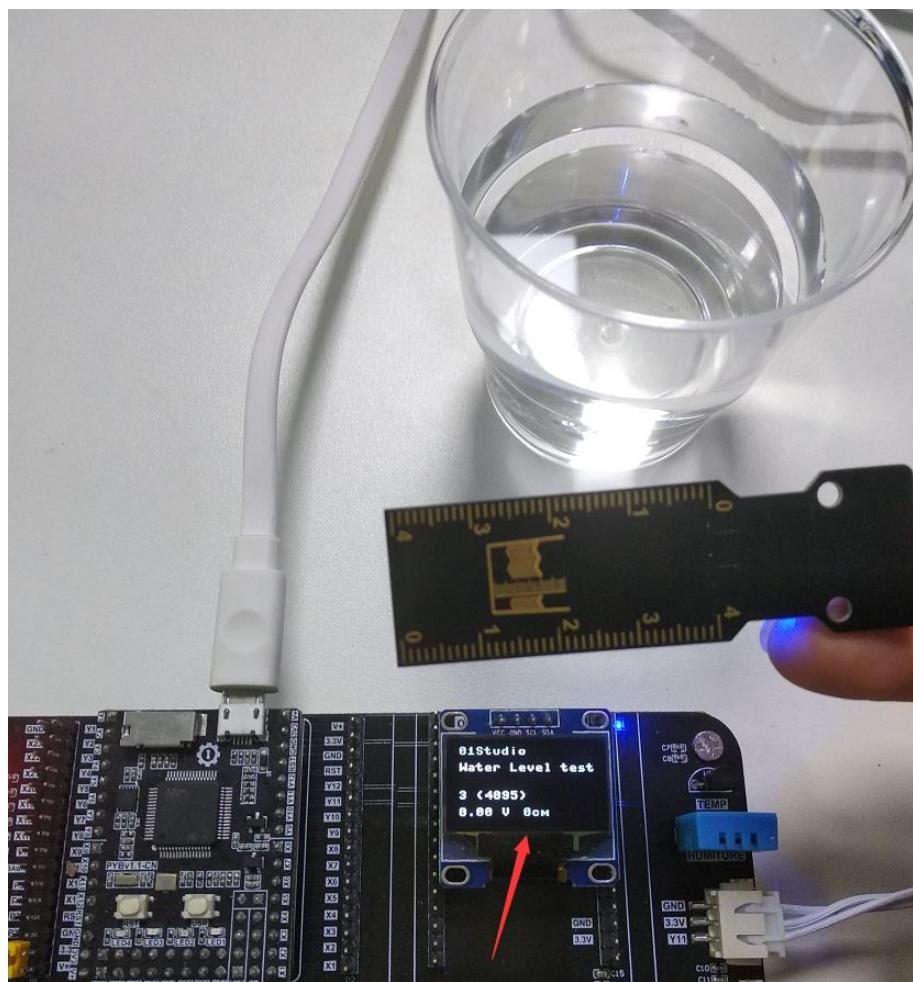


图 5-35

将水位传感器模块慢慢放入水中，可以看到 OLED 显示从 1cm-4cm 变化。  
(注意量程是 4cm，请勿将超出的电路部分放置水中以免发生短路。)



图 5-36 水位接近 1cm

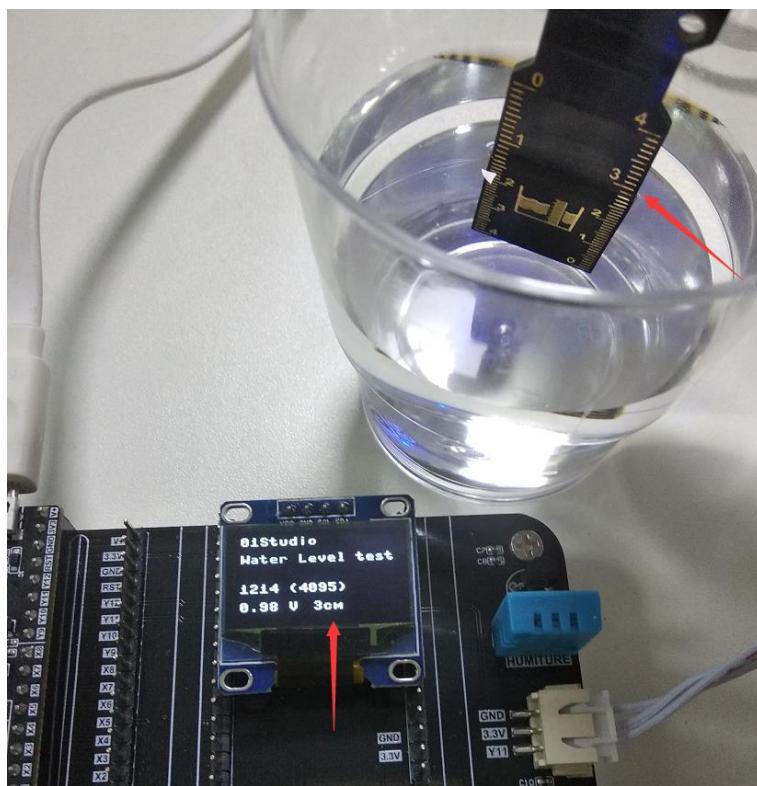


图 5-37 水位接近 3cm

● 总结：

从实验可以见到，这款水位传感器除了可以用来测量水位（液位）高度外，还可以用来测量室外下雨情况和家里的淹水情况。有兴趣的小伙伴可以动手制作一下！

## 5.7 大气压强传感器

- **前言：**

本实验中大气压强传感器模块使用的是 **BMP280**，这是一款专为移动应用而设计的高精度大气压传感器，传感器模块采用极其紧凑的封装，其小尺寸和低功耗可以应用在手机、GPS 模块、手表的电池供电设备中。BMP280 传感器除了可以测量大气压强，还可以测量温度（温度精度不高）以及通过计算公式来换算出海拔高度。

- **实验平台：**

MicroPython 开发套件和大气压强传感器模块，传感器连接到 I2C/UART 扩展接口。

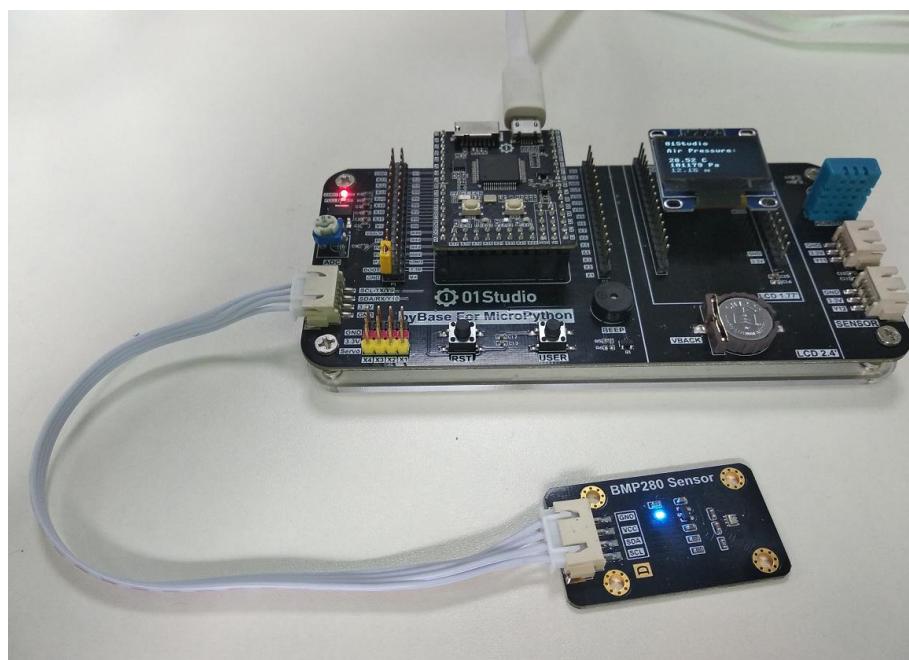


图 5-38 开发套件和大气压强传感器接线方法

- **实验目的：**

编程测量当前环境的大气压强、温度信息，将大气压强通过公式计算出当前海拔高度，并在 OLED 显示。打造自己的气压计！

- **实验讲解：**

我们先来看看大气压强传感器模块的介绍：

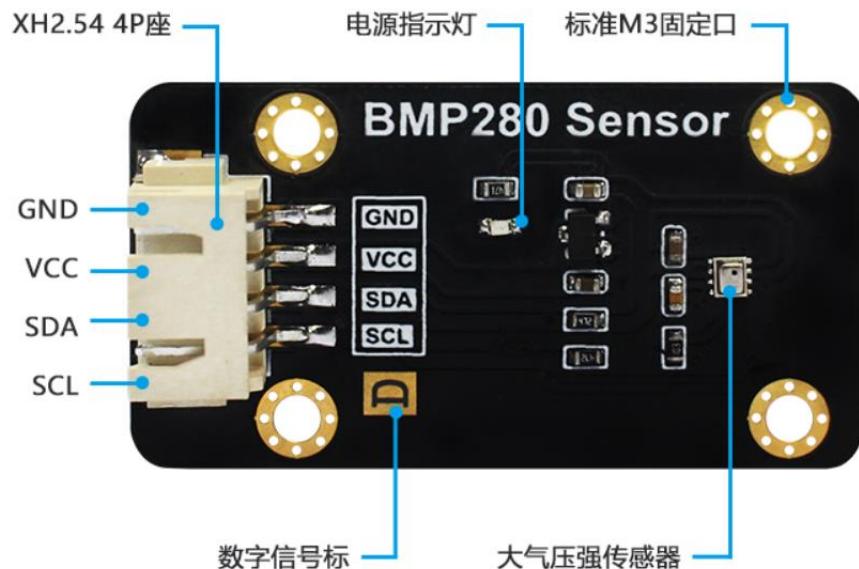


图 5-39 大气压强传感器模块

功能参数	
传感器型号	BMP280
供电电压	3.3-5V
工作电流	<20 mA
工作温度	-20 至 85°C
接口定义	XH2.54 防呆接口 (4Pin) 【GND、VCC、SDA、SCL】
通讯信号	I2C 总线
I2C 地址	0x76 (BMP280 的 SDO 默认下拉); 当 BMP280 的 SDO 引脚上拉时, I2C 地址为: 0x77
模块尺寸	4.5*2.5cm

表 5-8

从上面介绍可以看到, BMP280 是一款通过 I2C 接口驱动的传感器。连接到我们的 I2C (2) 外扩接口上。我们通过前面学习的 I2C 接口使用的方式, 即可以对该模块实现数据通讯。

标准大气压是指把温度为 0°C、纬度 45 度海平面 (海拔为 0 米) 上的气压

称为 1 个大气压，其数值为 101325 帕斯卡（Pa）。

大气压同海拔高度的关系： $P = P_0 \times (1 - H/44300)^{5.256}$

因此计算高度公式为： $H = 44300 * (1 - (P/P_0)^{(1/5.256)})$

式中：H 为海拔高度，P<sub>0</sub>=大气压（0°C，101325Pa）

从上面公式可以看到，高度是通过大气压强换算出来的，从物理学的角度我们可以知道，高度越高的地方，空气越稀薄，大气压强越低。通过气压的变化我们就可以计算出海拔高度；但是这存在特定条件，那就是温度为 0°C 的时候，而温度越高的地方，空气越稀薄，大气压强就越低。因此高度数据理论上需要做温度补偿，因此本实验的高度值换算存在轻微误差。有兴趣的小伙伴可以自行深入研究。

MicroPython 的强大之处是其有丰富的模块和函数库，一旦模块建立了，那么后来者使用起来就非常简单，无须再去做底层驱动的开发。从而实现面向对象的编程，而当有需要的时候又可以去改底层代码，可以说是进可攻退可守，非常灵活。在这里我们直接调用已经编写好的驱动文件 `bmp280.py`，该文件实现了对大气压、温度、高度的测量和计算。用户直接使用即可，具体如下：

构造函数	说明
<code>bmp280.BMP280(I2C)</code>	定义传感器的 I2C 接口。
使用方法	说明
<code>getTemp()</code>	获取温度数据
<code>getPress()</code>	获取气压数据
<code>getAltitude()</code>	获取高度数据

表 5-9 大气压强传感器 BMP280 对象

打开示例程序的 `bmp280.py` 文件，找到 `getAltitude()` 的定义函数，可以看到计算公式跟前面分析的一样，这里保留了 2 位小数：

```
# Calculating absolute altitude  
  
def getAltitude(self):  
  
    return '%.2f'%(44330*(1-(self.getPress()/101325)**(1/5.256)))
```

理解了传感器原理和模块使用方法后，我们可以整理出编程思路，流程图如下：

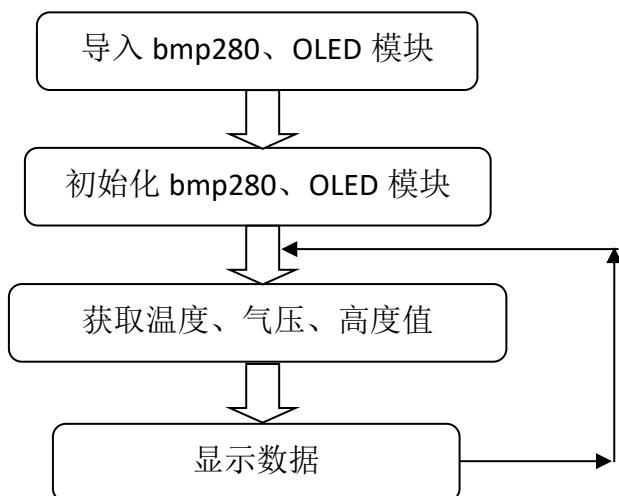


图 5-40 代码编写流程

参考程序代码如下：

```
...  
  
实验名称：大气压强传感器  
版本：v1.0  
日期：2019.5.1  
作者：01Studio 【www.01Studio.org】  
说明：测量 BMP280 温度、气压和计算海拔值，并在 OLED 上显示。。  
...
```

```
import pyb
import bmp280
from machine import Pin,I2C
from ssd1306 import SSD1306_I2C

#初始化 OLED
i2c = I2C(sda=Pin("Y8"), scl=Pin("Y6"))
oled = SSD1306_I2C(128, 64, i2c, addr=0x3c)

#初始化 BMP280, I2C 接口 2
BMP = bmp280.BMP280(I2C(2))

while True:

    oled.fill(0) # 清屏,背景黑色
    oled.text('01Studio', 0, 0)
    oled.text('Air Pressure:', 0, 15)

    # 温度显示
    oled.text(str(BMP.getTemp()) + ' C', 0, 35)
    # 湿度显示
    oled.text(str(BMP.getPress()) + ' Pa', 0, 45)
    # 海拔显示
    oled.text(str(BMP.getAltitude()) + ' m', 0, 55)

    oled.show()

    pyb.delay(1000) # 每隔 1 秒采集一次
```

### ● 实验结果：

程序烧写完复位后可以见到实验结果：温度：28.36°C，气压：101160 Pa，高度：13.74 M。该数据的实测地点深圳市，楼层5楼，可见数据还是相当合理。

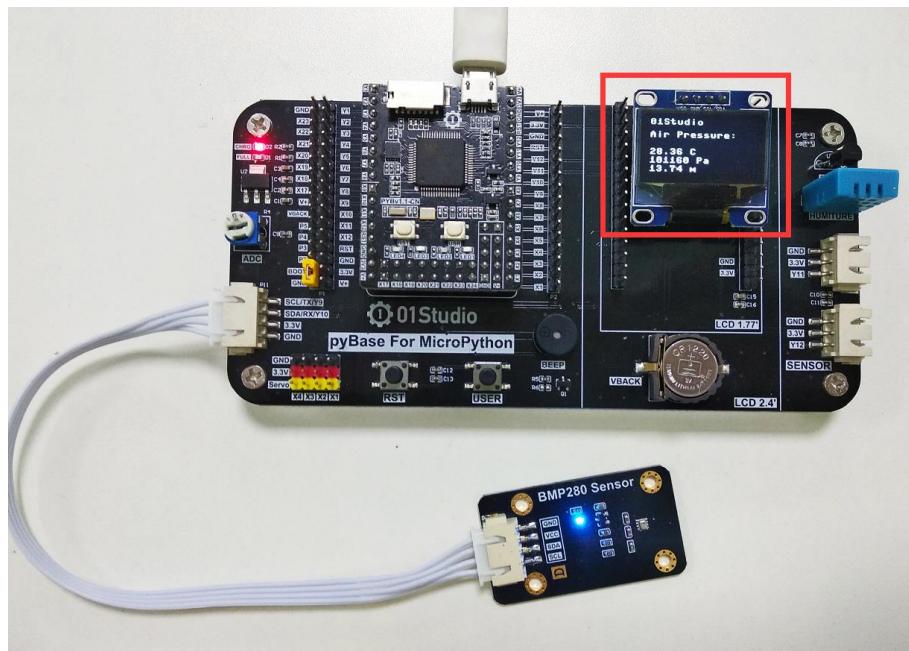


图 5-41 气压测量

开发板接上锂电池供电，尝试上下楼，会发现气压和海拔数值相应的变化。

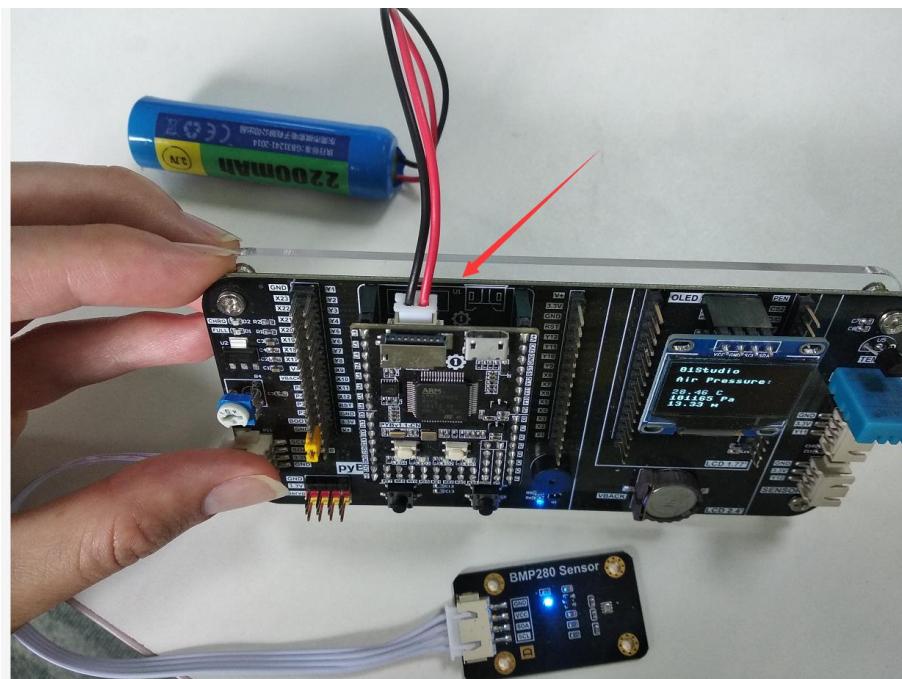


图 5-42 锂电池供电

● **总结:**

本节实现了大气压强传感器模块 BMP280 的应用，而且实验结果的精度很高，有兴趣的小伙伴可以结合自己情况，打造一个属于自己的气压计，然后到户外暴走！

## 5.8 超声波传感器

- **前言：**

超声波传感器是一款测量距离的传感器。其原理是利用声波在遇到障碍物反射接收结合声波在空气中传播的速度计算的得出。在测量、避障小车，无人驾驶等领域都有相关应用。

- **实验平台：**

MicroPython 开发套件和超声波传感器模块，传感器模块连接到 I2C/UART 扩展接口。

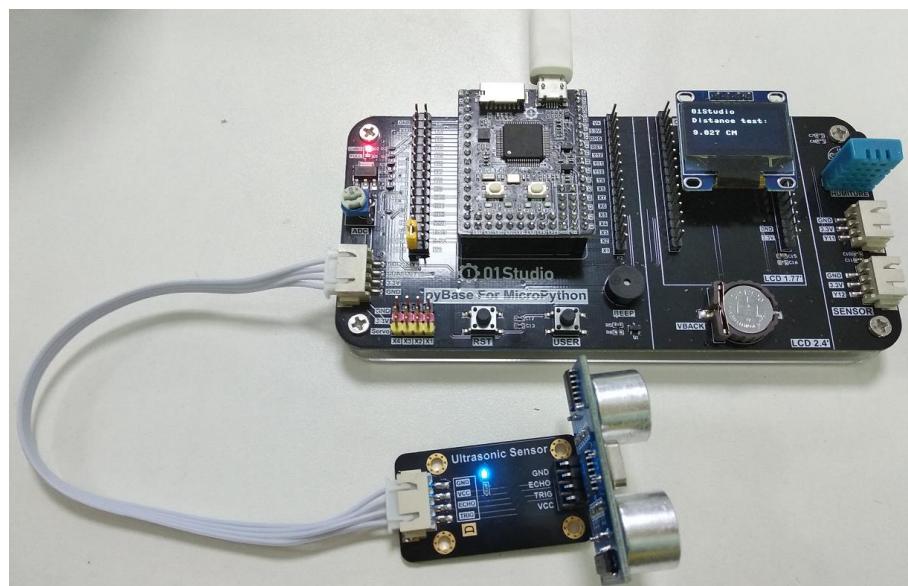


图 5-43 开发套件与传感器连接

- **实验目的：**

测量距离并在 OLED 上显示相关数据。

- **实验讲解：**

我们先来看看超声波传感器模块的介绍：

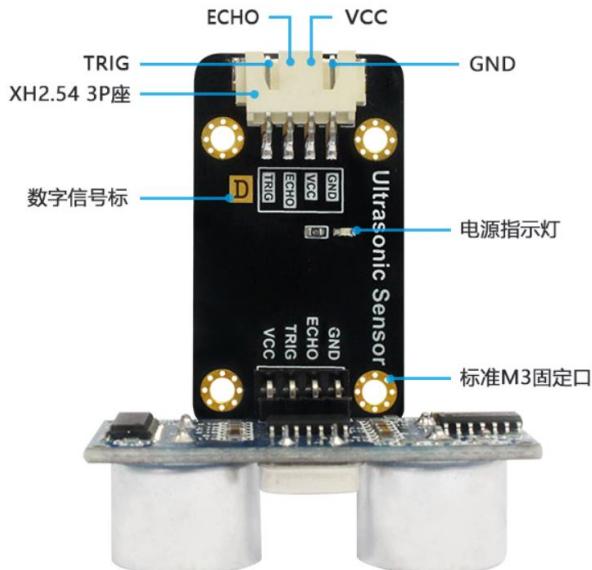


图 5-44 超声波模块

功能参数	
传感器型号	HCSR04
供电电压	3.3-5V
工作电流	<20 mA
工作温度	-20 至 85 °C
接口定义	XH2.54 防呆接口 (4Pin) 【GND、VCC、ECHO、TRIG】
通讯信号	IO 数字接口
测量距离	2-450 cm
测量精度	0.5 cm
整体尺寸	5.5*4.5*3.0 cm

表 5-10

超声波传感器模块使用两个 IO 口分别控制超声波发送和接收，工作原理如下：

1. 给超声波模块接入电源和地；
2. 给脉冲触发引脚 (trig) 输入一个长为 20us 的高电平方波；

3. 输入方波后，模块会自动发射 8 个 40KHz 的声波，与此同时回波引脚（echo）端的电平会由 0 变为 1；（此时应该启动定时器计时）
4. 当超声波返回被模块接收到时，回波引脚端的电平会由 1 变为 0；（此时应该停止定时器计数），定时器记下的这个时间即为超声波由发射到返回的总时长；
5. 根据声音在空气中的速度为 340 米/秒，即可计算出所测的距离。

要学习和应用传感器，学会看懂传感器的时序图是很关键的，所以我们来看一下超声波传感器 HCSR04 的时序触发图。

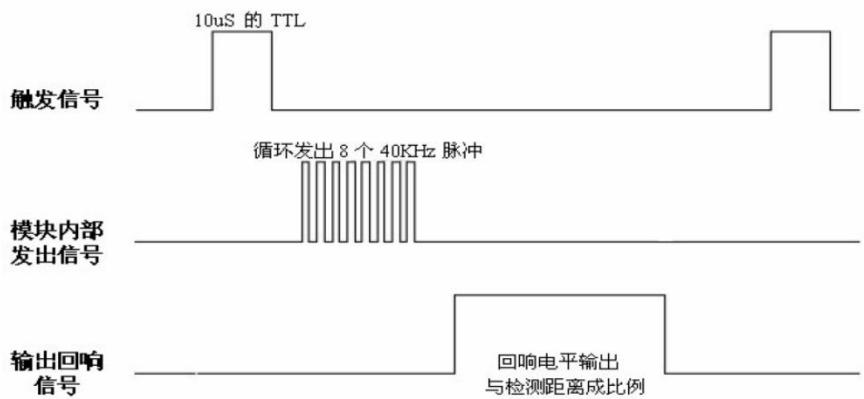


图 5-45 超声波传感器 HCSR04 时序图

以上普及了超声波传感器的原理，我们已经将其集成在 `HCSR04.py` 文件，如想了解代码原理可以打开 `HCSR04.py` 文件查看代码实现原理。使用 MicroPython 开发的用户只需要直接使用即可。使用方法如下：

构造函数	说明
<code>HCSR04.HCSR04(trig, echo)</code>	定义连接传感器的 trig 和 echo 引脚
使用方法	说明
<code>getDistance()</code>	获取距离数据

表 5-11 超声波传感器 HCSR04 对象

从上表看到，超声波传感器的使用方法非常简单，使用 2 个代码就完成对距离信息的采集，代码编写流程如下：

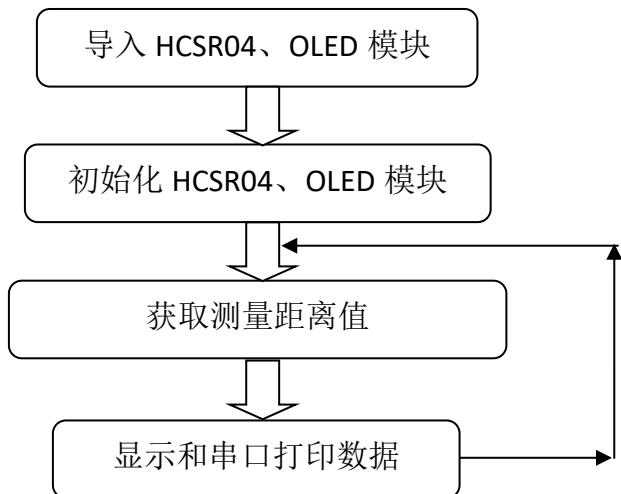


图 5-46 代码编写流程

主程序 `main.py` 参考代码如下：

```

...
实验名称: 超声波传感器
版本: v1.0
日期: 2019.5.1
作者: 01Studio 【www.01Studio.org】
说明: 通过超声波传感器测距，并在 OLED 上显示。
...

from HCSR04 import HCSR04      #子文件夹下的调用方式
from pyb import Pin,delay
from machine import Pin,I2C
from ssd1306 import SSD1306_I2C

#初始化 OLED
i2c = I2C(sda=Pin("Y8"), scl=Pin("Y6"))
oled = SSD1306_I2C(128, 64, i2c, addr=0x3c)

```

```
#初始化接口 trig='Y9',echo='Y10'  
trig = Pin('Y9',Pin.OUT_PP)  
echo = Pin('Y10',Pin.IN)  
HC=HCSR04(trig,echo)  
  
while True:  
  
    oled.fill(0) # 清屏,背景黑色  
    oled.text('01Studio', 0, 0)  
    oled.text('Distance test:', 0, 15)  
  
    Distance = HC.getDistance() #测量距离  
  
    # OLED 显示距离  
    oled.text(str(Distance) + ' CM', 0, 35)  
  
    oled.show()  
  
    #串口打印  
    print(str(Distance) +' CM')  
  
    delay(1000) #每秒采集 1 次
```

### ● 实验结果：

在超声波传感器 30cm 外放置障碍物，可以看到 OLED 显示屏实时显示距离数据约为 30cm。

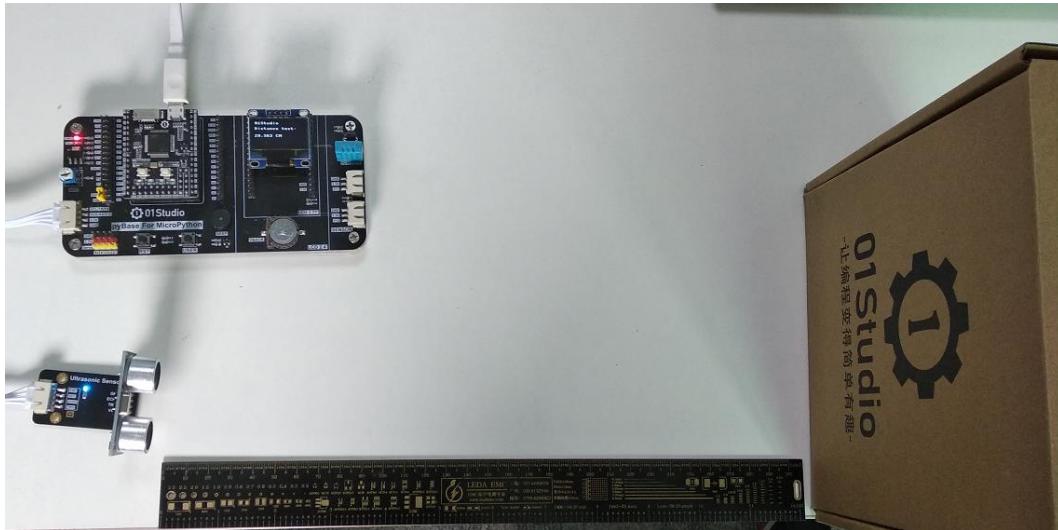


图 5-47 障碍物距离为 30cm

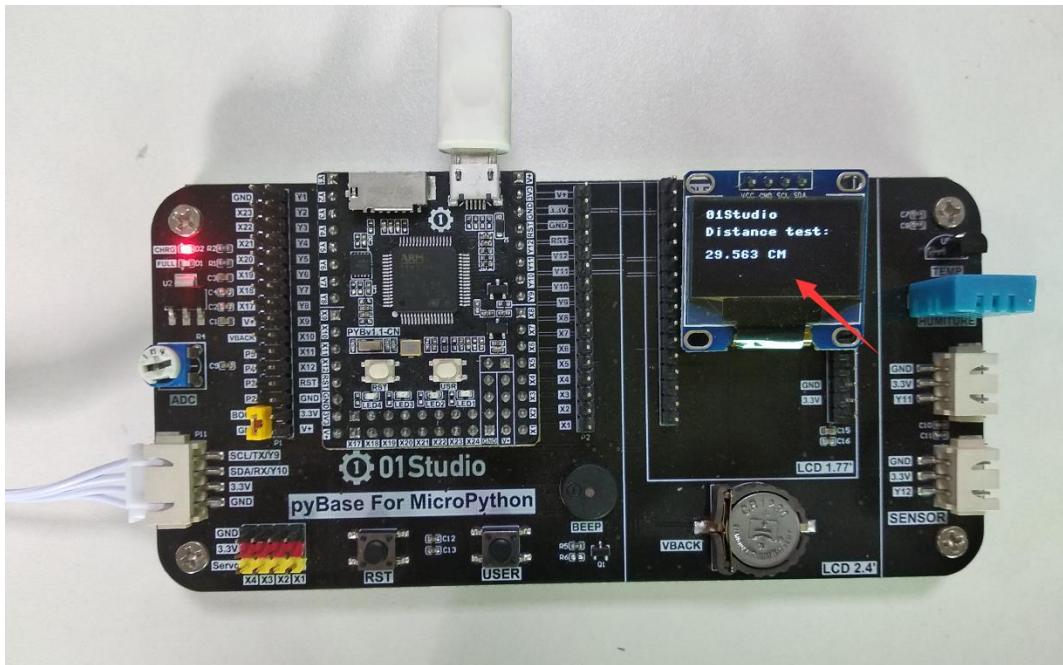


图 5-48 实验结果约为 30cm

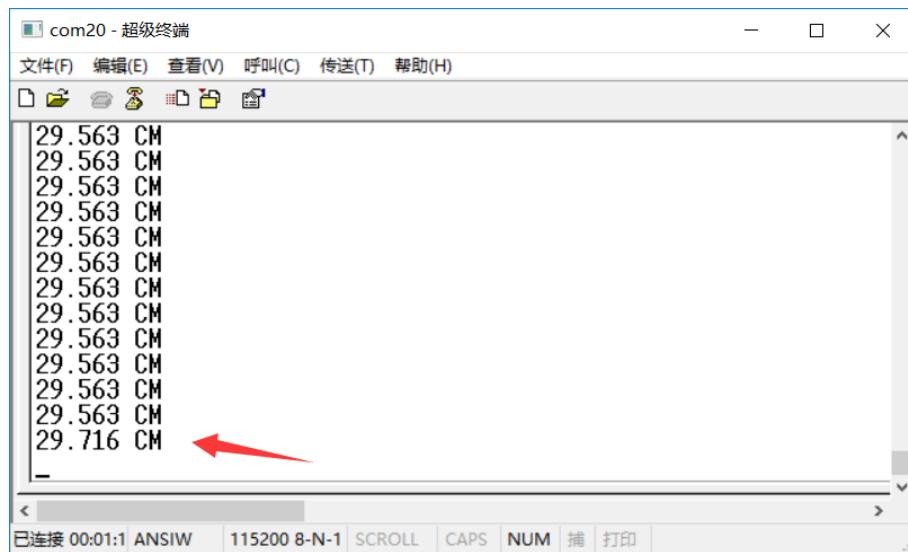


图 5-49 超级终端同步打印实验结果

### ● 总结：

除去 OLED 显示代码，我们实际上只用了 2 行代码：初始化和调用测量函数就实现了对超声波传感器测距的应用。这让我们再一次感受到了 MicroPython 的魅力。赶快动手制作自己的避障小车和其他好玩的创作吧。

## 第6章 拓展实验

在学习了 MicroPython 前面内容后，相信我们已经对其编程方法有了一定的认识，那么除了较为简单的基础实验外，我们还可以使用 MicroPython 开发板连接非常多的外接设备，如常见的舵机、LCD 等。

这些外设的编程难度分两块：如果是仅仅面向应用的话，那么我们可以使用非常简单的方式就用起来；由于代码是完全开源，所以如果想深入学习，则可以看一下驱动代码的实验原理，甚至自行编写驱动程序。从而一起丰富 MicroPython 的周边的生态产品。

## 6.1 继电器

- **前言:**

我们知道我们的开发板 GPIO 输出的电平是 3.3V 的，这是不能直接控制一些高电压的设备，比如电灯（220V）。这时候就可以使用我们常用的低压控制高压元件—继电器。

- **实验平台:**

pyBoard 开发套件和继电器模块。

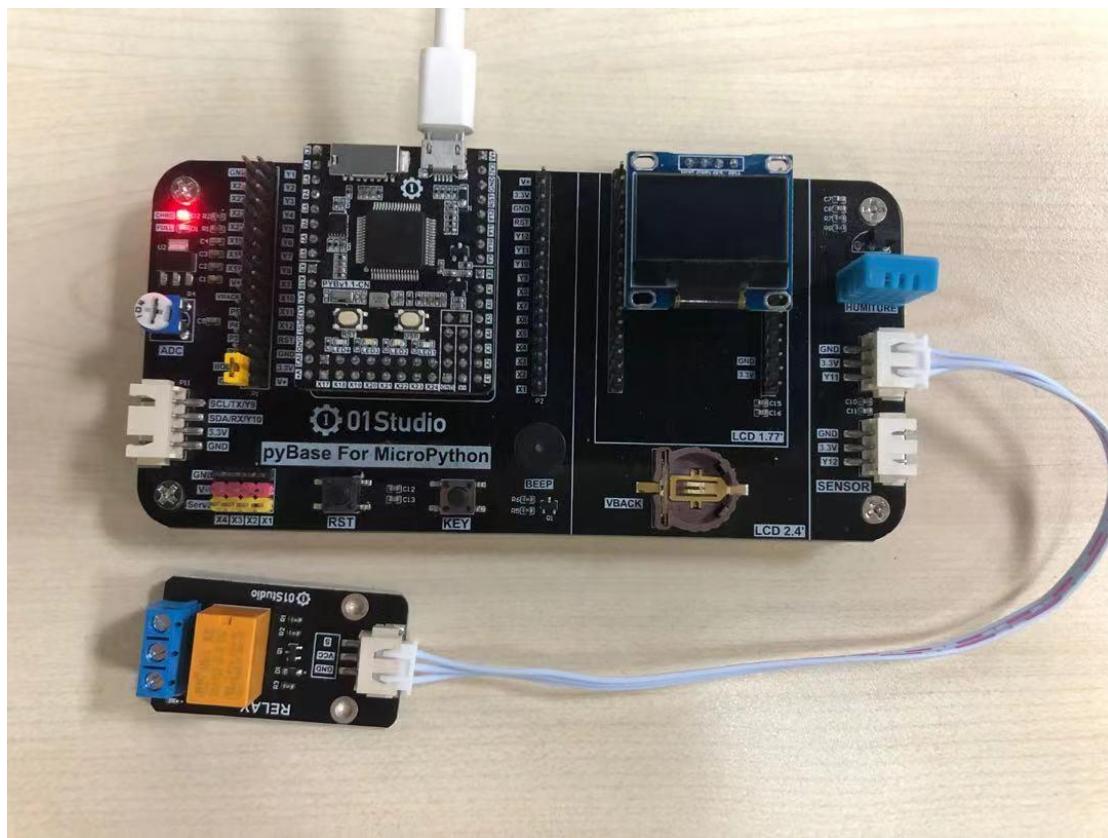


图 6-1 开发套件与传感器连接

- **实验目的:**

使用按键控制继电器通断。

- **实验讲解:**

我们先来看看继电器模块的介绍：



图 6-2 超声波模块

功能参数	
供电电压	3.3V
触发方式	低电平触发
高压连接	最大 250V/3A
接口定义	XH2.54 防呆接口 (3Pin) 【GND、VCC、IO】
整体尺寸	4.5*2.5 cm

表 6-1 继电器模块功能参数

继电器可以理解成是一个开关，相当于我们平时家里面的电灯开关面板一样，只是现在使用单片机的 GPIO 来控制。继电器低压控制高压电器一个比较典型的接线图如下图所示：

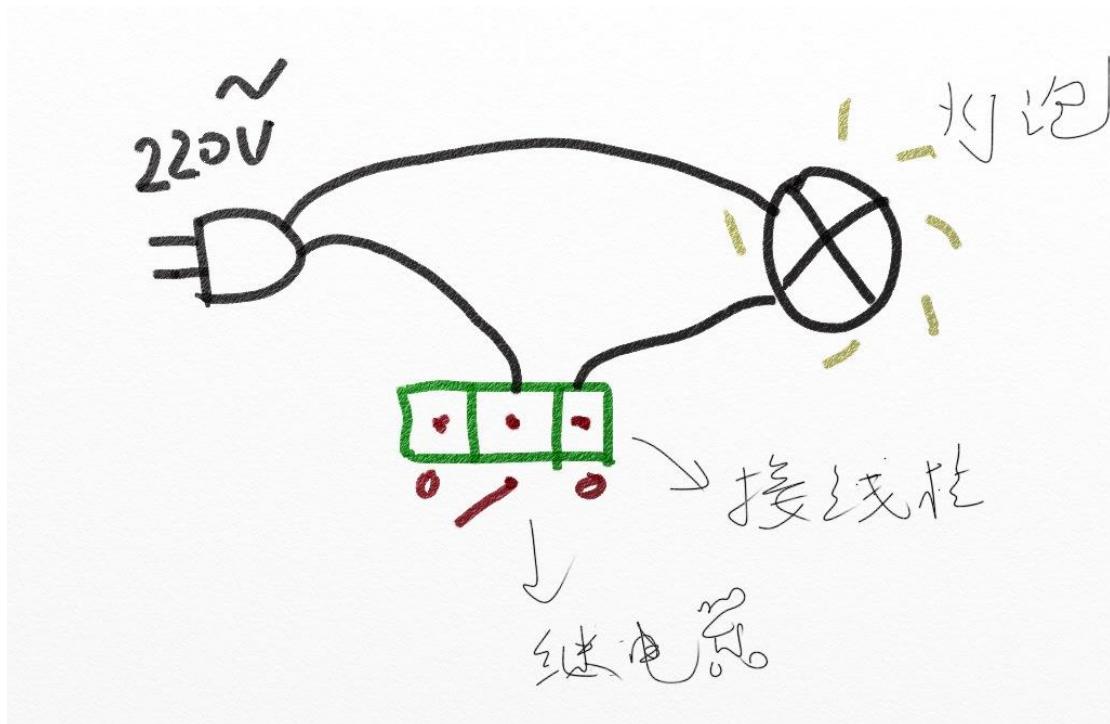


图 6-3 继电器接线

01Studio 的继电器模块的控制原理非常简单，跟 LED 控制方式一样，只是使用低电平 ‘0’ 表示继电器开，高电平 ‘1’ 表示继电器关。

我们可以使用按键外部中断来使用继电器。继电器连接到 pyBase 的‘Y11’引脚，相当于连接到，编程流程图如下：

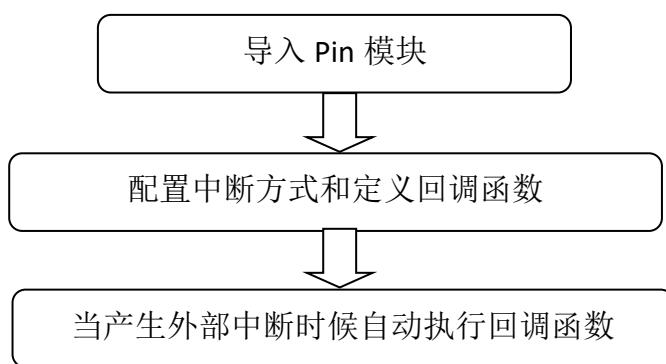


图 6-4 代码编写流程

主程序 main.py 参考代码如下：

```
'''

实验名称: 继电器

版本: v1.0

日期: 2021.8

作者: 01Studio

社区: www.01studio.cc

'''


#导入相关模块

from machine import Pin

import time


relay=Pin('Y11',Pin.OUT,value=1) #构建继电器对象,默认断开

KEY=Pin('Y1',Pin.IN,Pin.PULL_UP) #构建 KEY 对象

state=0 #LED 引脚状态


#LED 状态翻转函数

def fun(KEY):

    global state

    time.sleep_ms(10) #消除抖动

    if KEY.value()==0: #确认按键被按下

        state = not state

        relay.value(state)

KEY.irq(fun,Pin.IRQ_FALLING) #定义中断, 下降沿触发
```

- 实验结果：

通过按键来控制继电器通断：

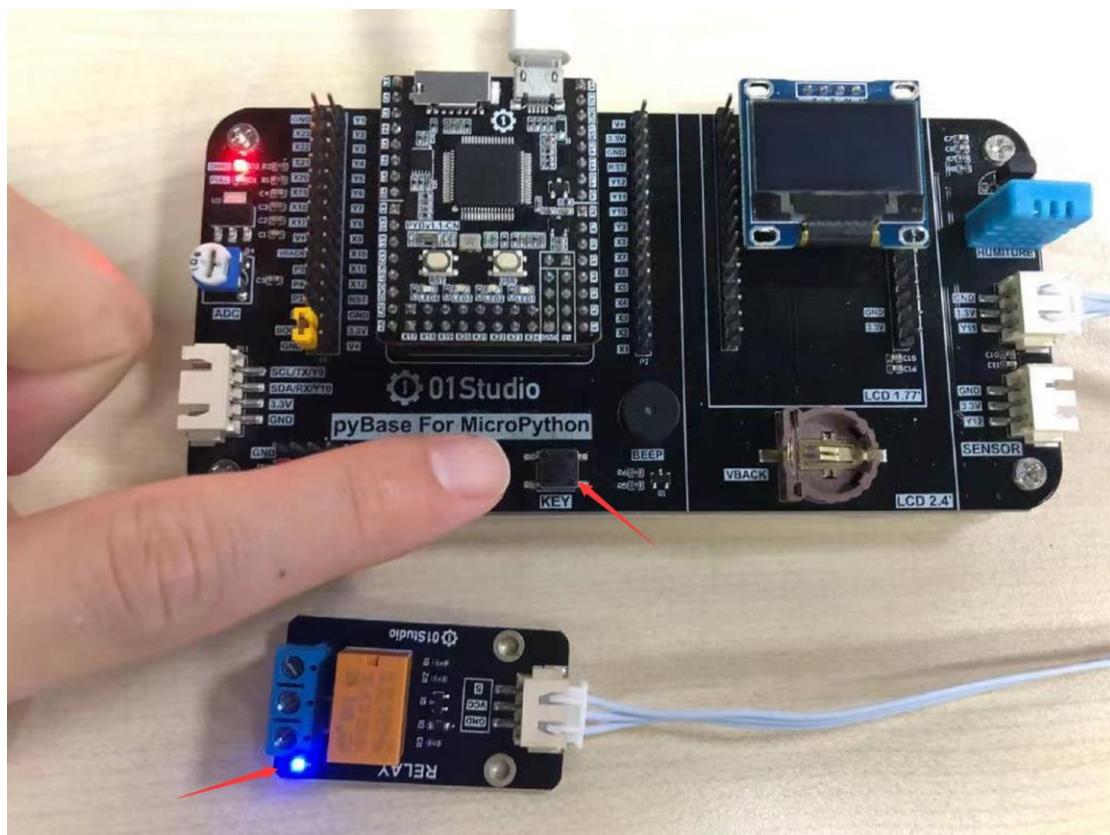


图 6-5 继电器控制

- 总结：

继电器的控制方式非常简单，用途非常广。只需要一个简单的 GPIO 高低电平即可实现控制。

## 6.2 舵机

### ● 前言：

舵机又叫伺服电机，是一个可以旋转特定角度的电机，可转动角度通常是 $90^\circ$ 、 $180^\circ$  和  $360^\circ$  ( $360^\circ$  可以连续旋转)。我们看到的机器人身上就有非常多的舵机，它们抬手或者摇头的动作往往是通过舵机完成，因此机器人身上的舵机越多，意味着动作越灵活。

### ● 实验平台：

MicroPython 开发套件和舵机。

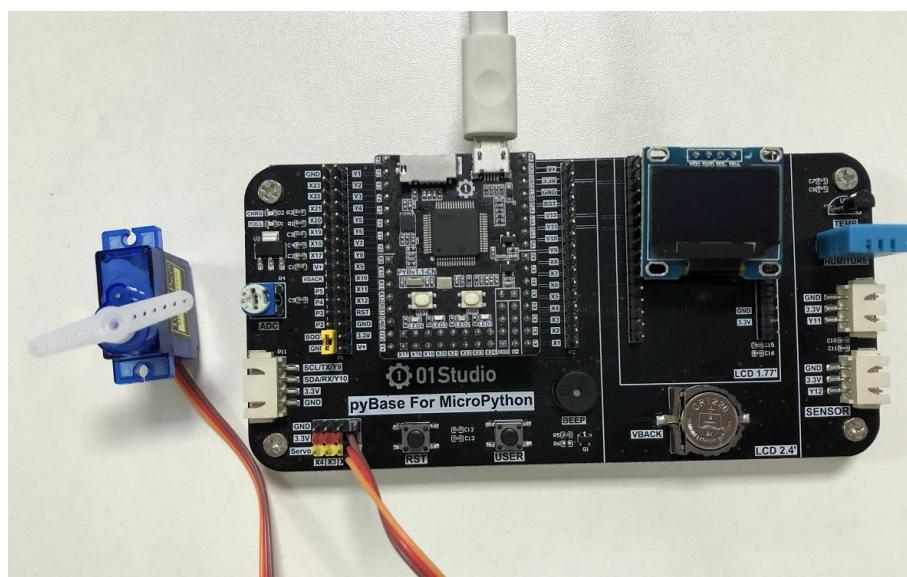


图 6-6 舵机接线

通常情况下：黑色表示 GND，红色表示 VCC，橙色表示信号线。

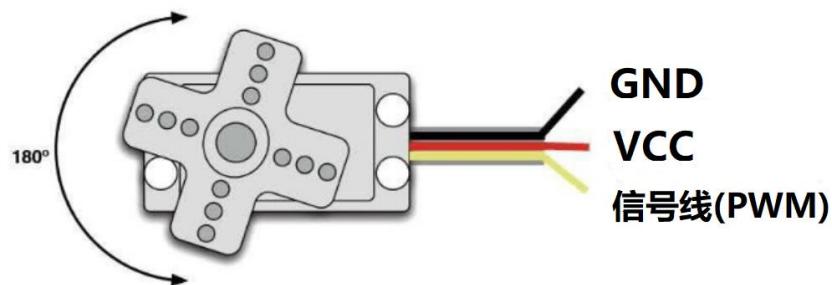


图 6-7 舵机示意图

- **实验目的:**

通过编程实现对舵机的控制。

- **实验讲解:**

伺服电机对象通过 3 线（信号，电源，地）控制，pyboard 和 MicroPython 开发板上有 4 个位置可以插这些电机，分别对应 X1-X4 引脚。

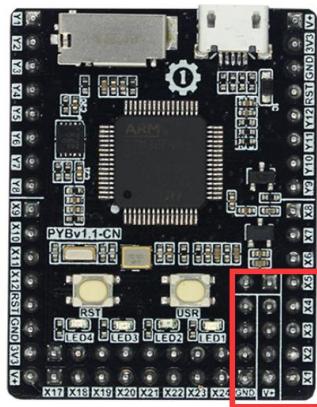


图 6-8 pyBoard v1.1-CN4 路舵机接口

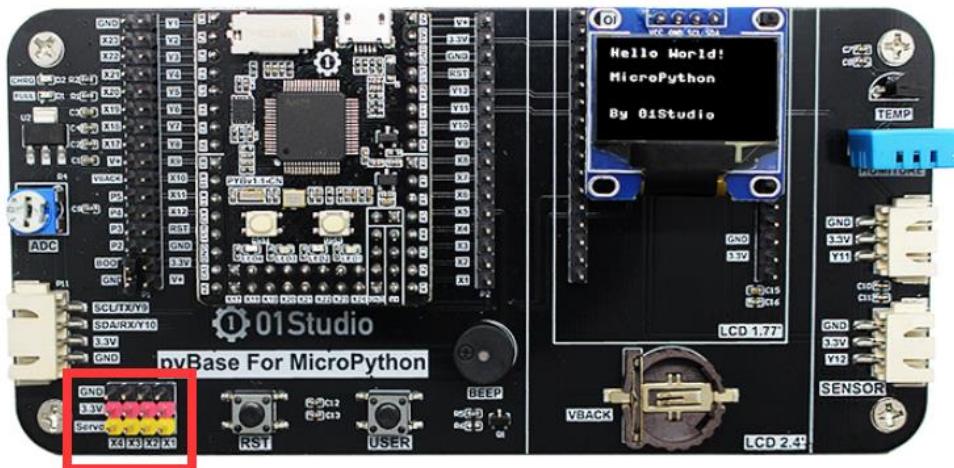


图 6-9 MicroPython 开发板 4 路舵机接口

180° 舵机的控制一般需要一个 20ms 左右的时基脉冲，该脉冲的高电平部分一般为 0.5ms-2.5ms 范围内的角度控制脉冲部分，总间隔为 2ms。以 180 度角度伺服为例，在 MicroPython 编程对应的控制关系是从-90° 至 90°，示例图如下：

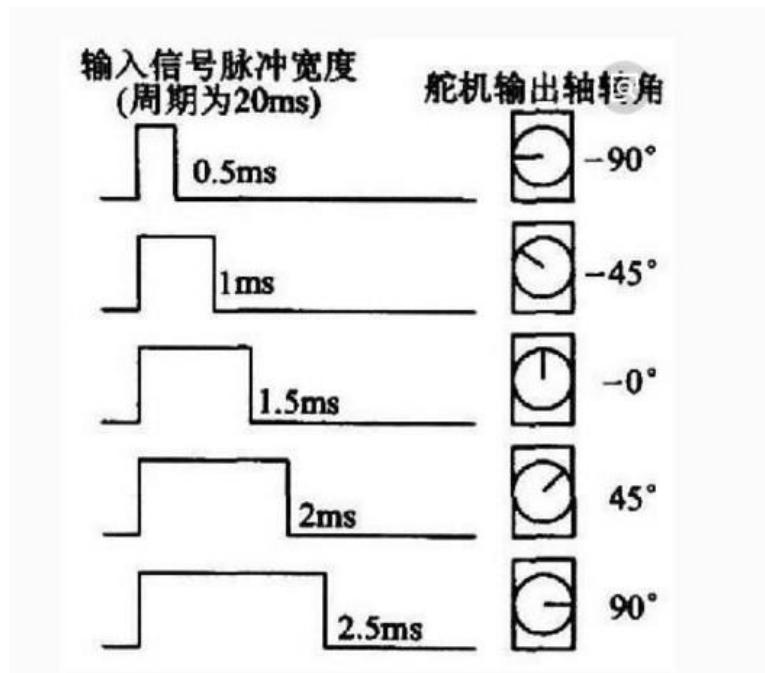


图 6-10 PWM 与角度关系

而对于 360° 连续旋转舵机，上面的脉冲表则对应从正向最大速度旋转到反向最大速度旋转的过程。

pyBoard 固件自带了舵机（Servo）的对象模块，我们可以直接调用，下面是 Servo 对象说明。

构造函数
<code>s=pyb.Servo(id)</code>
构建舵机对象。id 是 1-4，对应引脚 X1-X4
使用方法
<code>s.angle(angle,time=0)</code>
角度控制。angle:角度[-90 至 90]; time:旋转到指定角度的时间，单位 ms， 默认 0 表示最快
<code>s.speed(speed,time=0)</code>
速度控制（针对 360 度连续旋转舵机）。speed:速度[-100 至 100]，time: 开始启动到指定速度的时间，单位 ms， 默认 0 表示以最快

表 6-2 舵机对象

可以看到 MicroPython 让舵机控制变得非常简单，代码编程流程图如下：

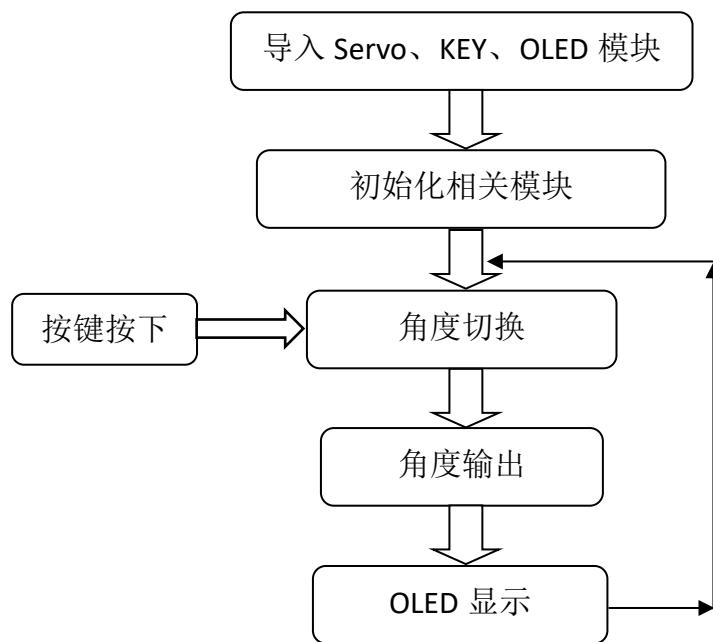


图 6-11 编程流程图

参考代码如下：

```

...
实验名称: 舵机(Servo)
版本: v1.0
日期: 2019.7
作者: 01Studio
说明: 通过 USER 按键让舵机旋转不同角度。
...
#导入相关模块
from pyb import Servo,Switch
from machine import Pin,I2C
from ssd1306 import SSD1306_I2C

```

```

#初始化相关模块

i2c = I2C(sda=Pin("Y8"), scl=Pin("Y6"))

oled = SSD1306_I2C(128, 64, i2c, addr=0x3c)


sw = Switch()      #定义按键对象名字为 sw
s1 = Servo(1)      #构建舵机对象 s1, 输出引脚为 X1


#定义 5 组角度: -90、-45、0、45、90
angle=[ -90,-45,0,45,90]

key_node = 0  #按键标志位
i = 0          #用于选择角度


#####
# 按键和其回调函数
#####

def key():
    global key_node
    key_node = 1


sw.callback(key)  #当按键被按下时, 执行函数 key()

#####
# OLED 初始显示
#####

oled.fill(0)  # 清屏显示黑色背景
oled.text('01Studio', 0, 0)  # 首行显示 01Studio
oled.text('Servo-180', 0, 15)  # 次行显示实验名称
oled.text(str(angle[i]), 0, 35)  # 显示当前角度
oled.text('Pls Press USER', 0, 55)  #提示按按键

```

```
oled.show()

#X1 指定角度,启动时 i=0, 默认-90°
s1.angle(angle[i])

while True:

    if key_node==1: #按键被按下
        i = i+1
        if i == 5:
            i = 0
        key_node = 0 #清空按键标志位

    #X1 指定角度
    s1.angle(angle[i])

    #显示当前频率
    oled.fill(0) # 清屏显示黑色背景
    oled.text('01Studio', 0, 0) # 首行显示 01Studio
    oled.text('Servo-180', 0, 15) # 次行显示实验名称
    oled.text(str(angle[i]), 0, 35) # 显示当前角度
    oled.text('Pls Press USER', 0, 55) #提示按按键
    oled.show()
```

### ● 实验结果:

将 180° 舵机插到 X1 三线接口，通过按键按下实现舵机的不同角度，同时在 OLED 上显示角度值。

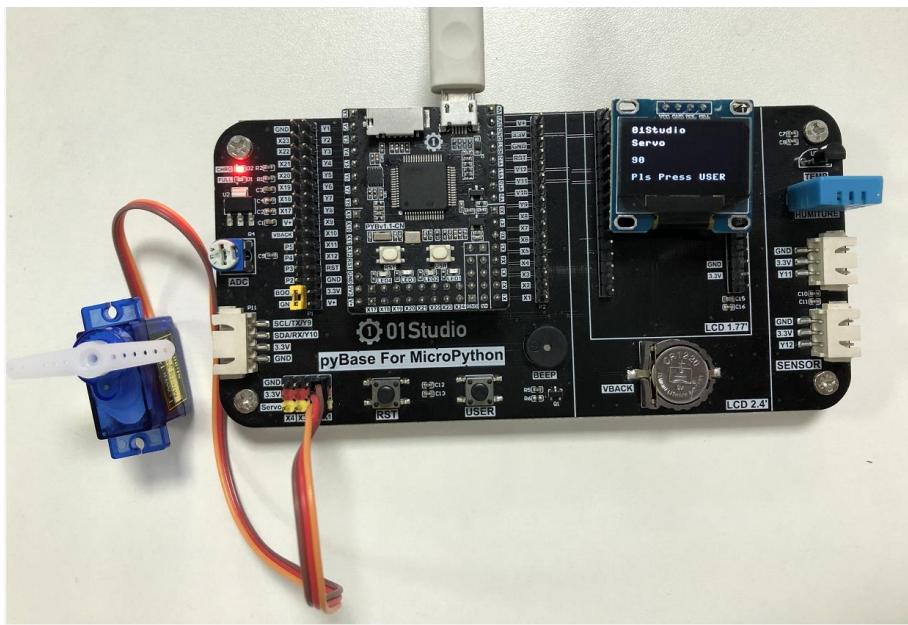


图 6-12 舵机角度 90°

### ● 实验拓展：

我们刚刚实现了 180° 舵机的角度控制，现在来做一下 360° 连续旋转舵机的实验，360° 连续旋转舵机可以实现直流减速电机功能，用在小车或者航模上。实验的基础条件和方式不变，我们只需要将角度参数【-90 至 90】改成速度【-100 至 100】即可。修改后的程序代码如下：

```
...
实验名称: 舵机(Servo)-360° 连续旋转
版本: v1.0
日期: 2019.7
作者: 01Studio
说明: 通过 USER 按键让舵机实现不同速度旋转。
...
```

```
#导入相关模块
from pyb import Servo,Switch
from machine import Pin,I2C
from ssd1306 import SSD1306_I2C
```

```

#初始化相关模块

i2c = I2C(sda=Pin("X12"), scl=Pin("X11"))

oled = SSD1306_I2C(128, 64, i2c, addr=0x3c)

sw = Switch()      #定义按键对象名字为 sw
s1 = Servo(1)      #构建舵机对象 s1, 输出引脚为 X1

#定义 5 组速度值
speed=[ -100, -50, 0, 50, 100]

key_node = 0  #按键标志位
i = 0          #用于选择角度

#####
# 按键和其回调函数
#####

def key():

    global key_node

    key_node = 1

sw.callback(key)  #当按键被按下时, 执行函数 key()

#####
# OLED 初始显示
#####

oled.fill(0)  # 清屏显示黑色背景
oled.text('01Studio', 0, 0)  # 首行显示 01Studio
oled.text('Servo-360', 0, 15)  # 次行显示实验名称
oled.text(str(speed[i]) , 0, 35)  # 显示当前速度值

```

```

oled.text('Pls Press USER', 0, 55) #提示按按键
oled.show()

#X1 指定角度,启动时 i=0, 默认-90°
s1.speed(speed[i])

while True:

    if key_node==1: #按键被按下
        i = i+1
        if i == 5:
            i = 0
        key_node = 0 #清空按键标志位

    #X1 指定角度
    s1.speed(speed[i])

    #显示当前频率
    oled.fill(0) # 清屏显示黑色背景
    oled.text('01Studio', 0, 0) # 首行显示 01Studio
    oled.text('Servo-360', 0, 15) # 次行显示实验名称
    oled.text(str(speed[i]), 0, 35) # 显示当前速度值
    oled.text('Pls Press USER', 0, 55) #提示按按键
    oled.show()

```

实验结果如下：

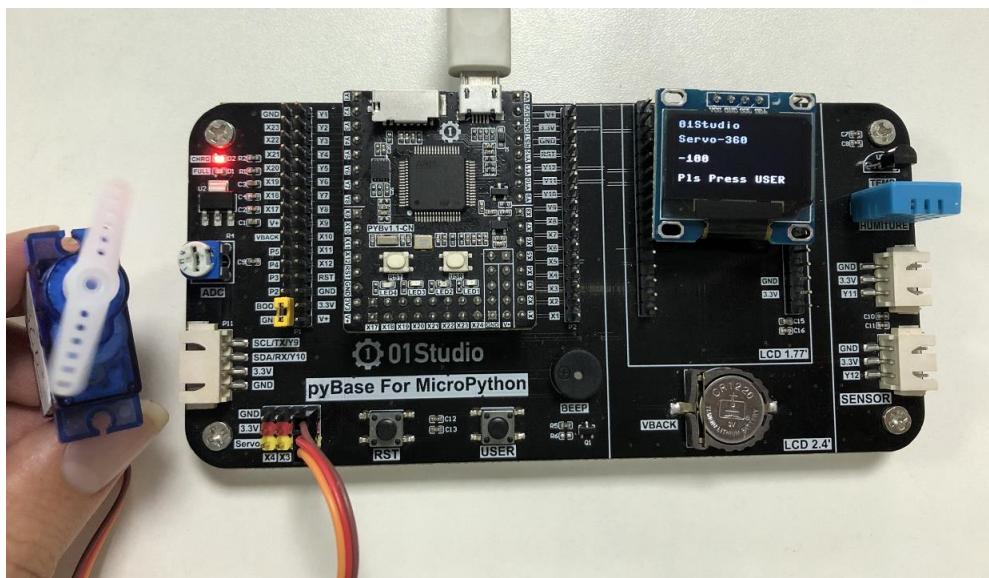


图 6-13 360°连续旋转舵机

● 总结：

通过本节我们学会了使用不同类型的舵机，通过多路电机的组合使用，可以实现模型、航模、小车、机器人的实验。

## 6.3 RGB 灯带

- **前言：**

RGB 灯带是彩灯的一种，我们看到长长的灯带以及 LED 彩色显示屏，都是采用一个个灯珠组合而成。在本章节我们将通过编程实现 RGB 灯带的控制，可以应用到家居布置、节日气氛的场景中去。

- **实验平台：**

MicroPython 开发套件和 RGB 灯带。连接到“Y11”接口。

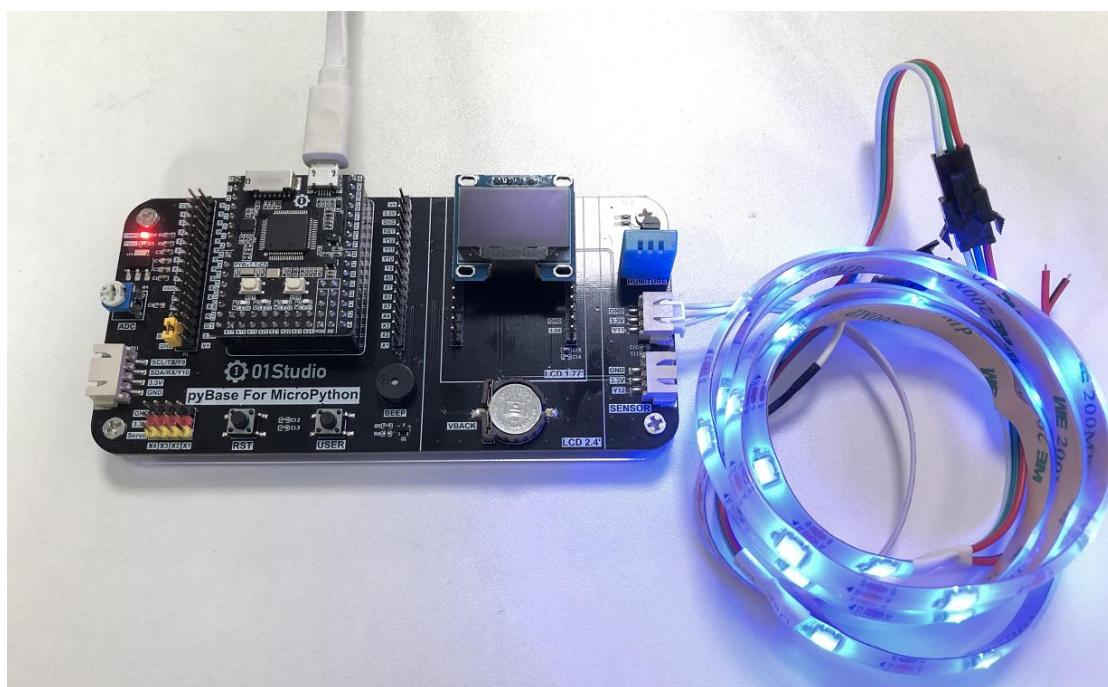


图 6-14 RGB 灯带接线方式

- **实验目的：**

通过编程实现灯带循环显示红色（RED）、绿色（GREEN）和蓝色（BLUE）。

- **实验讲解：**

先来介绍一下本实验用到的 RGB 灯带。

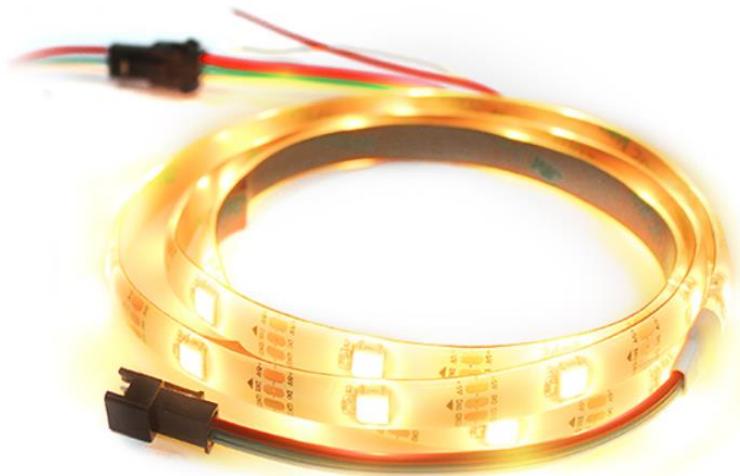


图 6-15 RGB 灯带

该灯带为 3 线接口，长 1 米，有 30 颗灯珠，每个灯珠里面都有一个驱动 IC，型号是 WS2812B，单总线驱动。每个灯珠首尾相连。灯带末端预留了接口可以串联更多的灯带。也就是说只需要通过 1 个 GPIO 口就可以控制了数十上百的灯珠了。

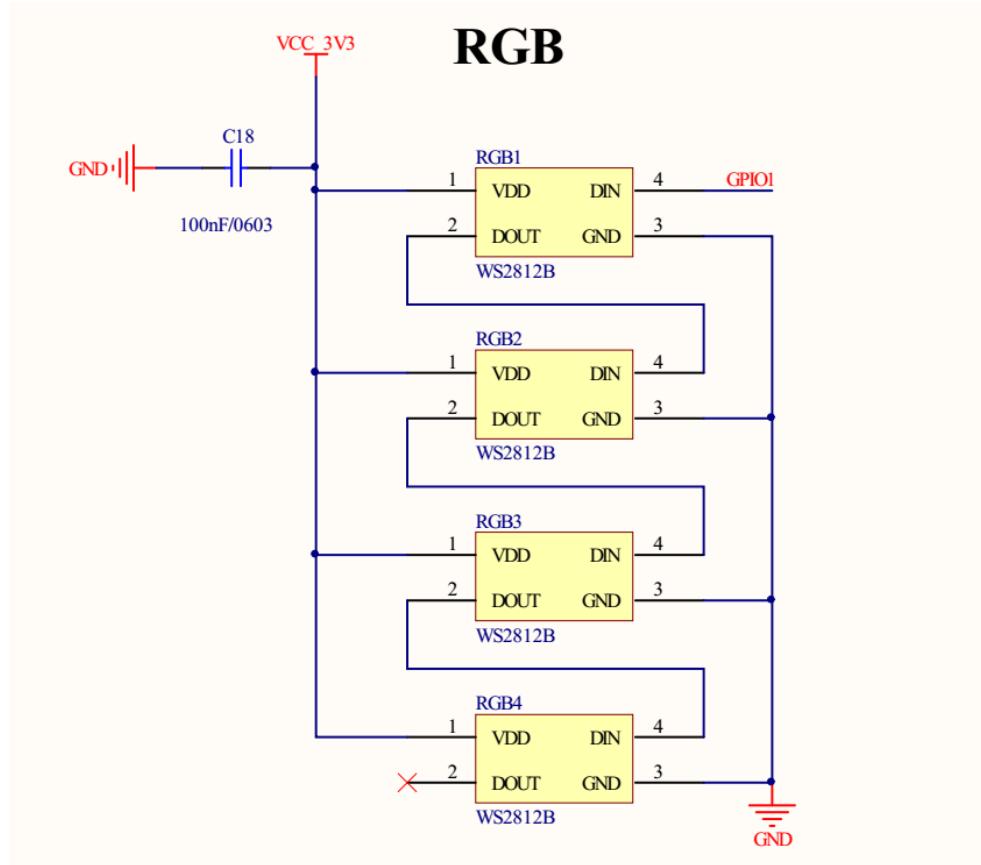


图 6-16 灯珠串联接线方式

了解了接线方式后，我们知道颜色是由最基本的三种颜色的不同亮度混合出新颜色。这 3 个最基本的颜色顺序分别是红，绿，蓝（RGB）。这里每个颜色的亮度级别从 0-255,0 表示没有， 255 表示最亮。如（255,0,0）则表示红色最亮。

GPIO 口就是将这些数据逐一发送给灯带。本实验灯带有 30 个灯珠，则连续发送 30 次，通过 SPI 方式发送。可见在灯珠数量多的时候，传输数据量是非常大的，这些我们通过驱动文件 ws2812.py 实现，这使得用户可以轻松地使用 MicroPython 编程实现相关功能，对象说明如下：

构造函数
<pre>strip=WS2812(spi_bus, led_count)</pre>
构建灯带对象。spi_bus:灯带控制引脚（MOSI），led_count:灯珠数量；
使用方法
<pre>strip.show(data)</pre>
灯带显示。data:颜色数据。例：1 个灯珠显示红色为 data=[(255,0,0)]; 2 个灯珠显示绿色为 data=[(0,255,0),(0,255,0)];如此类推。

表 6-3 灯带对象

编程流程图如下：

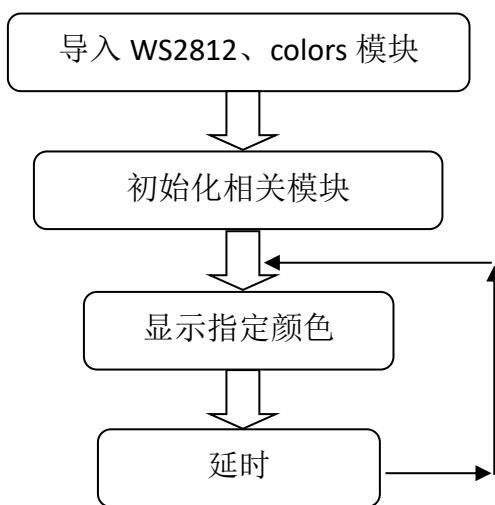


图 6-17 编程流程图

main.py 参考程序代码如下：

```
'''  
  
实验名称：RGB 灯带  
版本：v1.0  
日期：2019.7  
作者：01Studio  
说明：RGB 灯带控制。  
'''  
  
  
  
  


```
from ws2812 import WS2812  
from colors import *  
from machine import Pin  
import pyb  
  
#定义灯带连接引脚，Y11 接口  
LED = Pin('Y11',Pin.OUT,value=0)  
  
#构建 RGB 灯带对象，定义控制引脚和灯珠数量  
strip = WS2812(spi_bus=LED, led_count=30)  
  
#灯带填色函数，灯珠数量为 led_count  
def fill_color(color):  
    data=[]  
    for i in range (strip.led_count):  
        data.append(color)  
    return data  
  
#清空 RGB 灯带颜色  
strip.show(fill_color(EMPTY))
```


```

```
while True:
    strip.show(fill_color(RED))
    pyb.delay(1000)

    strip.show(fill_color(GREEN))
    pyb.delay(1000)

    strip.show(fill_color(BLUE))
    pyb.delay(1000)
```

我们简单讲解一下本实验代码：

主文件的颜色可以直接调用是因为在 `colors.py` 文件中包含了颜色的定义，用户可以根据自己的实际情况来补充。

```
#colors define
EMPTY=(0,0,0)
RED=(255,0,0)
GREEN=(0,255,0)
BLUE=(0,0,255)
WHITE=(255,255,255)
```

另外我们定义了一个颜色填充函数，里面是一个循环，`append` 的作用是在后面添加数据，返回的结果 `data[]` 是对灯带各个灯珠颜色的定义。

```
#灯带填色函数,灯珠数量为 led_count
def fill_color(color):
    data=[]
    for i in range (strip.led_count):
        data.append(color)
    return data
```

- **实验结果：**

将程序下载到 MicroPython 开发板，可以看到灯带的颜色在红、绿、蓝三色中循环变化。

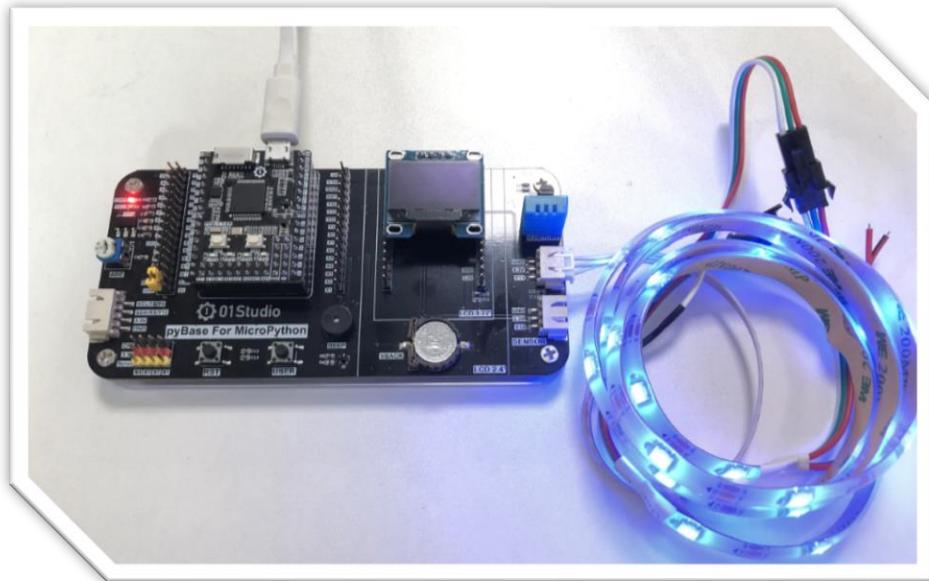


图 6-18 实验结果

- **总结：**

RGB 灯带的单总线特点使得我们可以轻易的增加和减少灯带的数量而无需过多的修改程序。但需要注意的是如果灯带数量过多，那么需要外接电源供电，否则会因为电流不足而影响使用。

## 6.4 以太网模块

我们先来看看 01Studio W5500 以太网模块参数介绍：

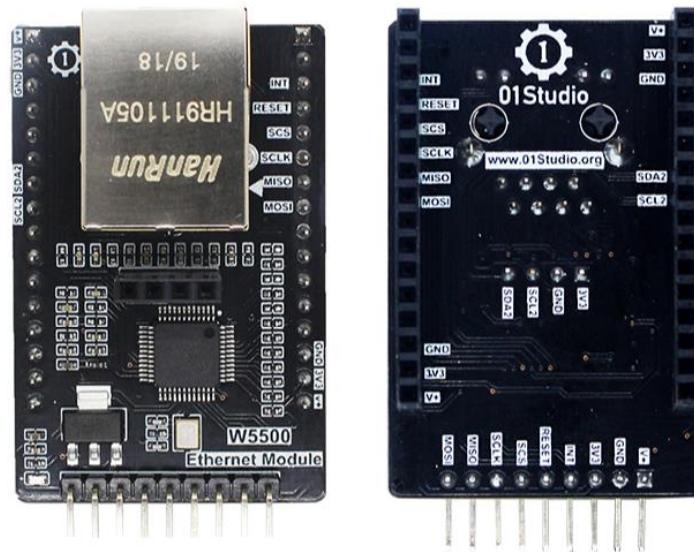


图 6-19 W5500 以太网模块

功能参数	
供电电压	3.3V 或 5V
主控芯片	W5500
控制方式	SPI 总线
接口定义	1、2.54mm 排母（兼容 pyboard v1.1 接口）； 2、2.54mm 排针
其它功能	支持 OLED 扩展接口
整体尺寸	6.6*4.2 cm

表 6-4

pyBoard 推荐引脚接口如下：

pyboard	W5500 以太网模块	引脚说明
Y3	INT	中断
Y4	RST	复位
Y5	CS	片选
Y6	SCK	SPI 接口
Y7	MISO	SPI 接口
Y8	MOSI	SPI 接口
3.3V	VCC	电源 (3.3V 供电)
GND	GND	地
V+	V+	电源 (3.6-6V 输入)

表 6-5 推荐接线

pyBoard v1.1-CN 可以通过连接 W5500 以太网模块实现网线连接路由器上网。需要先给 pyBoard v1.1-CN 烧录带 network 的 pyboard 固件，路径在：零一科技（01Studio）MicroPython 开发套件配套资料\03-相关固件\01-pyboard v1.1\网络版，烧写方法参考前面开发环境建立—固件更新章节内容。

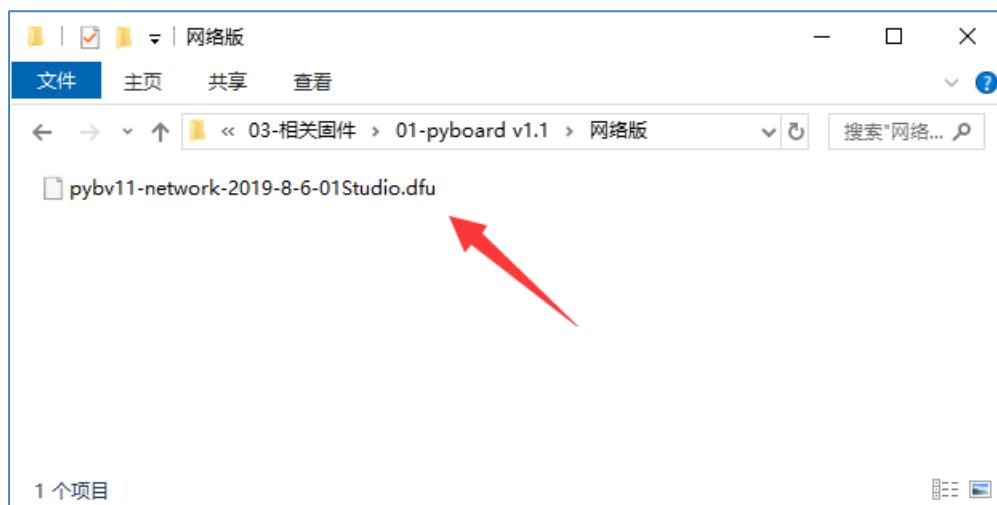


图 6-20 网络版固件

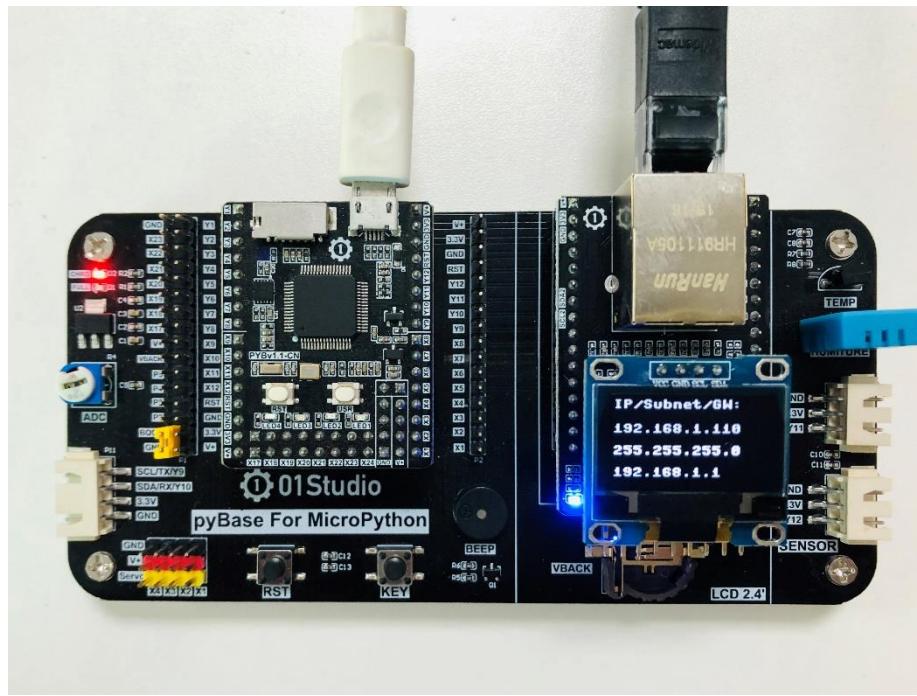


图 6-21 以太网连接

### 6.4.1 连接网络

- **前言:**

万物互联是趋势，pyBoard 不具备 WiFi 功能，但其强大的库基础了 W5500 以太网模块函数代码，我们可以轻松通过 MicroPython 编程实现 pyboard 连接互联网。

- **实验平台:**

pyBoard 开发套件和 W5500 以太网模块。（可以选配 OLED）

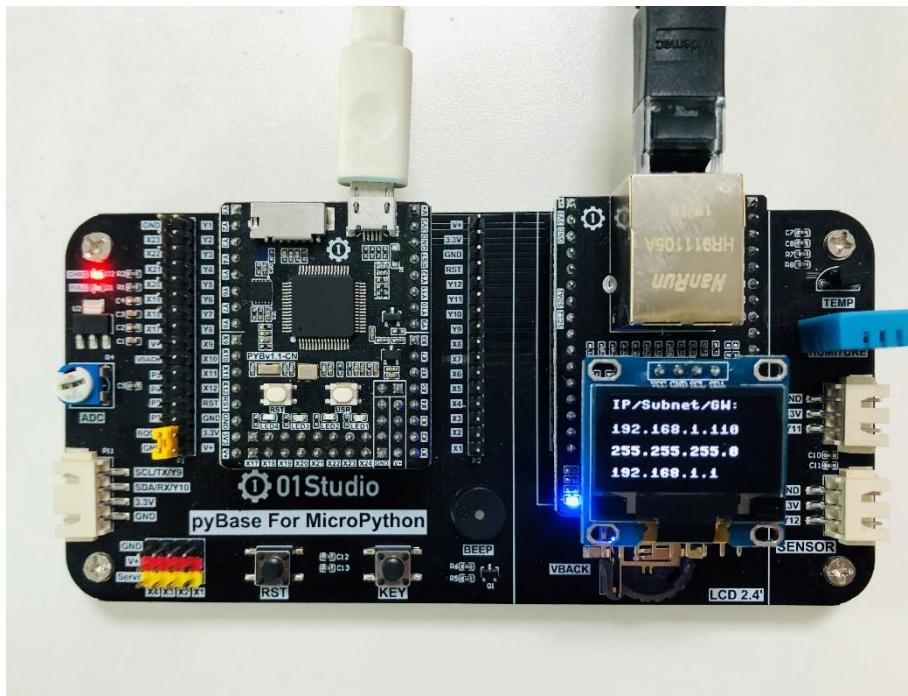


图 6-22 pyBoard 开发套件

- **实验目的:**

编程实现连接路由器，将 IP 地址等相关信息通过串口终端打印。

- **实验讲解:**

连接路由器上网是我们每天都做的事情，日常生活中我们只需要知道路由器的账号和密码，就能使用电脑或者手机连接到无线路由器，然后上网冲浪。

pybaord 带 network 固件已经集成了 network 模块，开发者使用内置的 network 模块函数可以非常方便地连接上路由器。我们先来看看 network 模块的

构造函数和使用方法。

构造函数
<pre>nic = network.WIZNET5K(spi, pin_cs, pin_RST)</pre>
构建 W5500 以太网模块连接对象。 【spi】 SPI 引脚； 【pin_CS】 模块片选引脚 【pin_RST】 模块中断引脚
使用方法
<pre>nic.active(True)</pre>
激活网络。True：开启，False：关闭。
<pre>nic.isconnected()</pre>
检查设备是否已经连接上。返回 True：已连接；False：未连接。
<pre>wlan.ifconfig([ip, subnet, gateway, dns])</pre>
设备信息配置。 【手动分配 IP】 ip：IP 地址；subnet：子网掩码；gateway：网关地址；dns：DNS 【自动分配 IP】 'dhcp' (如果参数为空，则返回当前连接信息。)

图 6-23 W5500 以太网模块对象

从上表可以看到 MicroPython 通过模块封装，让联网变得非常简单。代码编写流程如下：

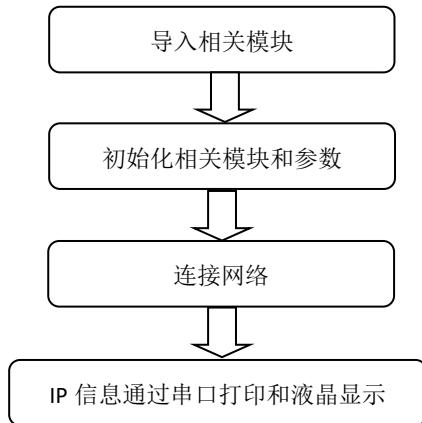


图 6-24 代码编写流程图

实验参考代码如下：

```

...
实验名称: W5500 以太网模块连接网络
版本: v1.0
日期: 2019.11
作者: 01Studio
说明: 通过 Socket 编程实现 pyBoard+W5500 以太网模块连接网络。
...

import network,usocket,pyb
from machine import I2C,Pin
from ssd1306 import SSD1306_I2C

#初始化相关模块, OLED 引脚换成了 Y9、Y10
i2c = I2C(sda=Pin('Y10'), scl=Pin('Y9'))
oled = SSD1306_I2C(128, 64, i2c, addr=0x3c)

#以太网模块初始化
nic = network.WIZNET5K(pyb.SPI(2), 'Y5', 'Y4')

```

```
nic.active(True)
nic.ifconfig('dhcp')

#判断网络是否连接成功
if nic.isconnected():

    print(nic.ifconfig()) #打印 IP 信息

    #OLED 数据显示
    oled.fill(0)      #清屏背景黑色
    oled.text('IP/Subnet/GW:',0,0)
    oled.text(nic.ifconfig()[0], 0, 20)
    oled.text(nic.ifconfig()[1],0,38)
    oled.text(nic.ifconfig()[2],0,56)
    oled.show()
```

上面代码在 `main.py` 文件中，我们也可以新建一个 `net.py` 文件，然后通过模块引用方式来供 `main.py` 调用，以提高程序的灵活性。

### ● 实验结果：

运行程序，可以观察到连接成功后串口终端打印 IP 等信息。

```
('192.168.1.111', '255.255.255.0', '192.168.1.1', '192.168.1.1')
MicroPython v1.11-255-g400a128e1-dirty on 2019-10-29; PYBv1.1 with STM32F405RG
Type "help()" for more information.
>>> █
```

图 6-25

液晶屏显示：

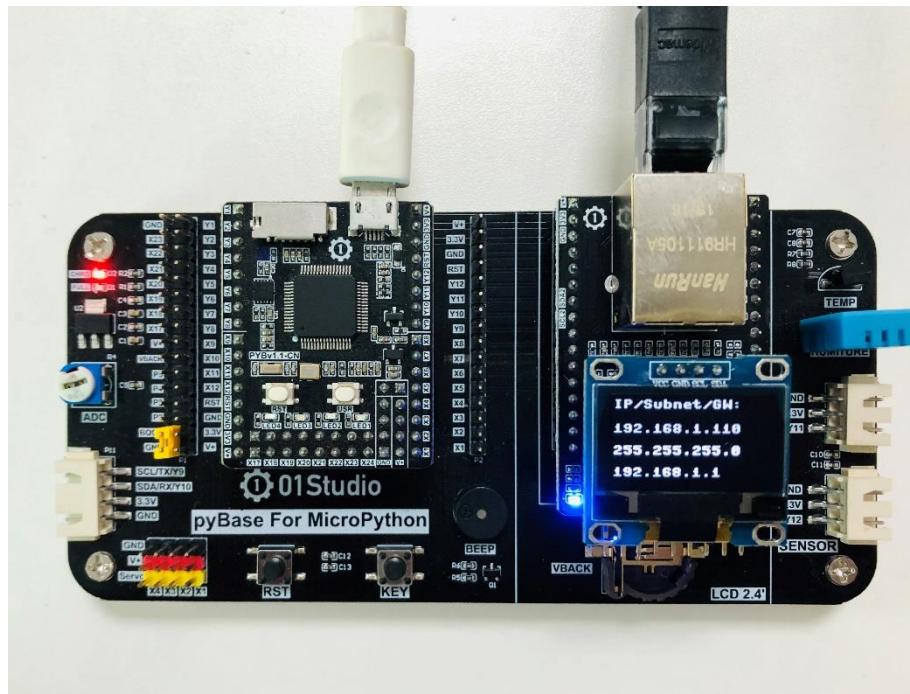


图 6-26 连接网络成功

### ● 总结：

本节是联网应用的基础，成功连接到路由器的实验后，后面就可以做 socket 和 MQTT 等相关网络通信的应用了。

## 6.4.2 Socket 通信

- 前言：

上一节我们学习了如何通过 MicroPython 编程实现 pyBoard+W5500 模块连接路由器。这一节我们则来学习一下 Socket 通信实验。Socket 几乎是整个互联网通信的基础。

- 实验平台：

pyBoard 开发套件和 W5500 以太网模块。

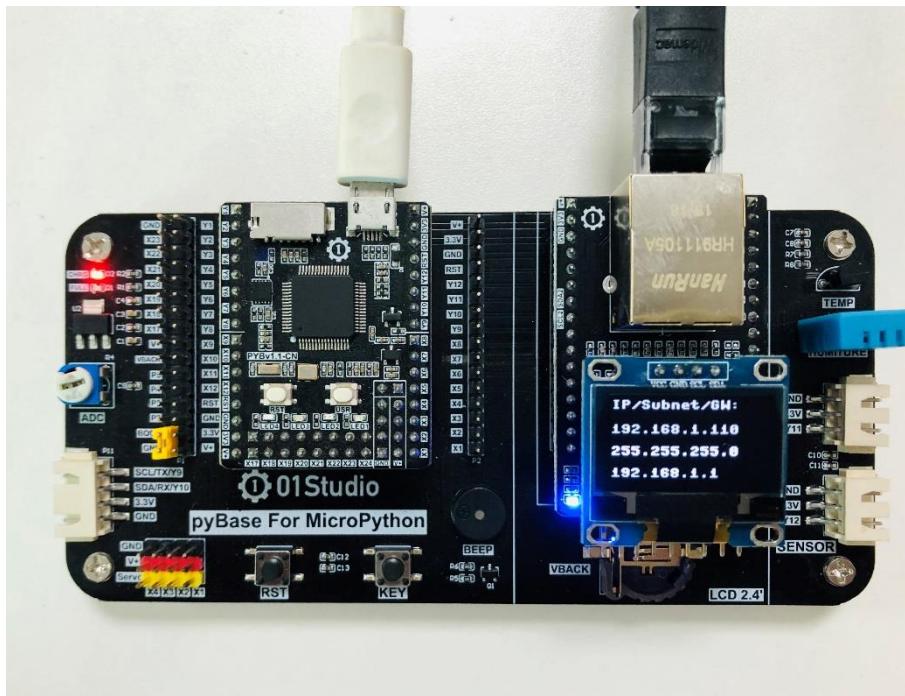


图 6-27 pyBoard 开发套件

- 实验目的：

通过 Socket 编程实现 pyBoard 与电脑服务器助手建立连接，相互收发数据。

- 实验讲解：

Socket 我们听得非常多了，但由于网络工程是一门系统工程，涉及的知识非常广，概念也很多，任何一个知识点都能找出一堆厚厚的的书，因此我们经常会混淆。在这里，我们尝试以最容易理解的方式来讲述 Socket，如果需要全面了解，可以自行查阅相关资料学习。

我们先来看看网络层级模型图，这是构成网络通信的基础：

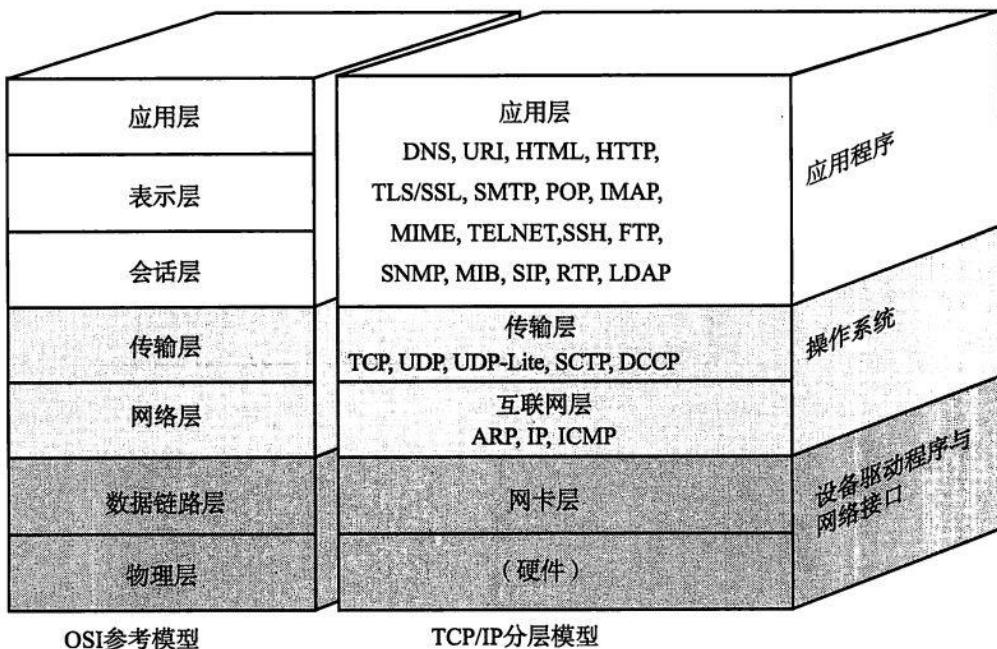


图 6-28 网络层级模型

我们看看 TCP/IP 模型的传输层和应用层，传输层比较熟悉的概念是 TCP 和 UDP，UDP 协议基本就没有对 IP 层的数据进行任何的处理了。而 TCP 协议还加入了更加复杂的传输控制，比如滑动的数据发送窗口（Slice Window），以及接收确认和重发机制，以达到数据的可靠传送。应用层中网页常用的则是 HTTP。那么我们先来解析一下这 TCP 和 HTTP 两者的关系。

我们知道网络通信是最基础是依赖于 IP 和端口的，HTTP 一般情况下默认使用端口 80。举个简单的例子：我们逛淘宝，浏览器会向淘宝网的网址（本质是 IP）和端口发起请求，而淘宝网收到请求后响应，向我们手机返回相关网页数据信息，实现了网页交互的过程。而这里就会引出一个多人连接的问题，很多人访问淘宝网，实际上接收到网页信息后就断开连接，否则淘宝网的服务器是无法支撑这么多人长时间的连接的，哪怕能支持，也非常占资源。

也就是应用层的 HTTP 通过传输层进行数据通信时，TCP 会遇到同时为多个应用程序进程提供并发服务的问题。多个 TCP 连接或多个应用程序进程可能需要通过同一个 TCP 协议端口传输数据。为了区别不同的应用程序进程和连接，许多计算机操作系统为应用程序与 TCP / IP 协议交互提供了套接字(Socket)接口。应

用层可以和传输层通过 **Socket** 接口，区分来自不同应用程序进程或网络连接的通信，实现数据传输的并发服务。

简单来说，**Socket** 抽象层介于传输层和应用层之间，跟 **TCP/IP** 并没有必然的联系。**Socket** 编程接口在设计的时候，就希望也能适应其他的网络协议。

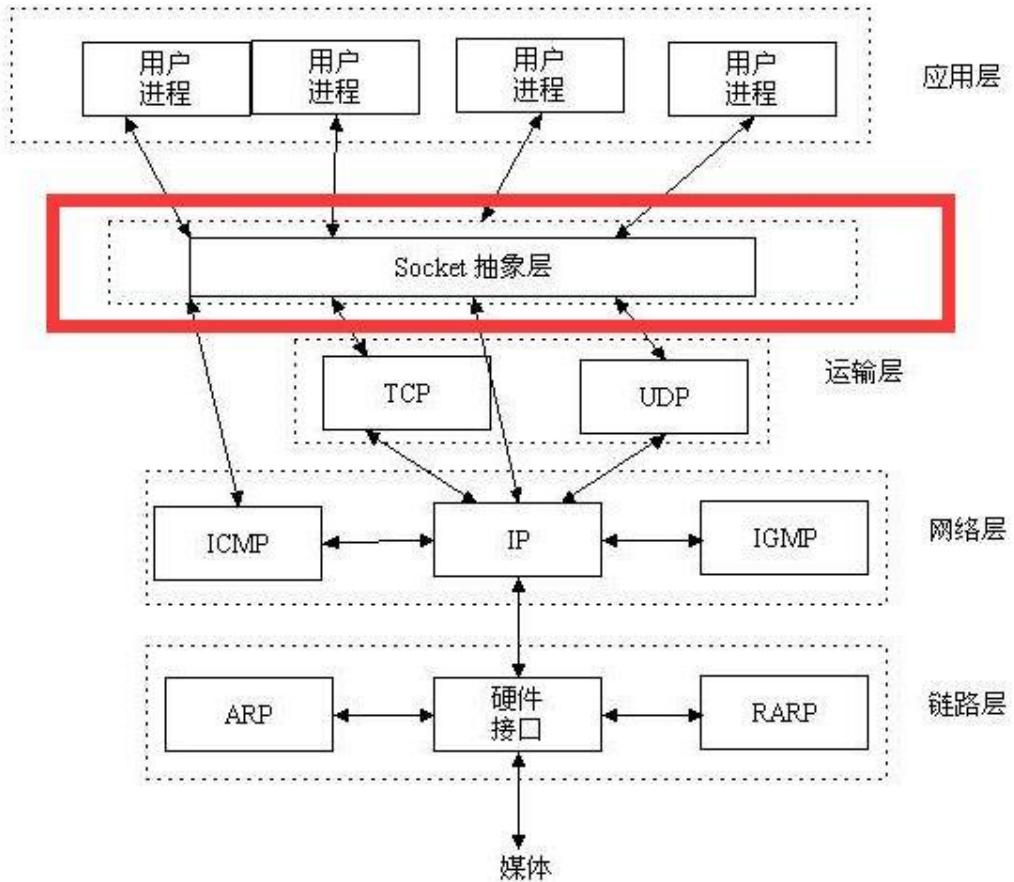


图 6-29 **Socket** 抽象层

套接字（socket）是通信的基石，是支持 **TCP/IP** 协议的网络通信的基本操作单元。它是网络通信过程中端点的抽象表示，包含进行网络通信必须的五种信息：**连接使用的协议（通常是 TCP 或 UDP），本地主机的 IP 地址，本地进程的协议端口，远地主机的 IP 地址，远地进程的协议端口。**

所以，**socket** 的出现只是可以更方便的使用 **TCP/IP** 协议栈而已，简单理解就是其对 **TCP/IP** 进行了抽象，形成了几个最基本的函数接口。比如 `create`, `listen`, `accept`, `connect`, `read` 和 `write` 等等。以下是通讯流程：

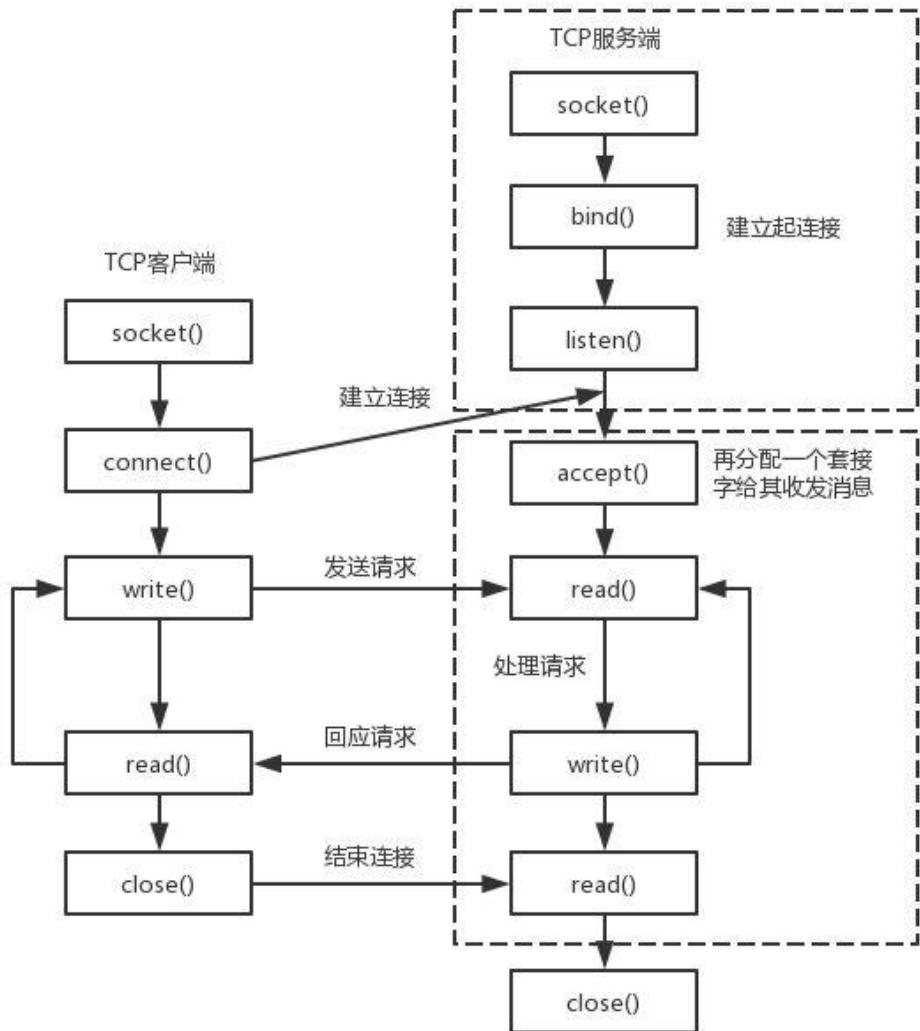


图 6-30 Socket 通信过程

从上图可以看到，建了 Socket 通信需要一个服务器端和一个客户端，以本实验为例，**pyBoard** 作为客户端，电脑使用网络调试助手作为服务器端，双方使用 TCP 协议传输。对于客户端，则需要知道电脑端的 IP 和端口号即可建立连接。  
(端口可以自定义，范围在 0~65535，注意不占用常用的 80 等端口即可。)

以上的内容，简单来说就是如果用户面向应用来说，那么 OpenMV 只需要知道**通讯协议是 TCP 或 UDP、服务器的 IP 和端口号**这 3 个信息，即可向服务器发起连接和发送信息。就这么简单。

MicroPython 已经封装好相关模块 usocket, 跟传统的 socket 大部分兼容, 两者均可使用, 本实验使用 usocket, 对象如下介绍:

构造函数
<pre>s=usocket.socket(af=AF_INET, type=SOCK_STREAM,proto=IPPROTO_TCP)</pre>
构建 usocket 对象。 af: AF_INET→IPV4, AF_INET6 → IPV6; type: SOCK_STREAM→TCP, SOCK_DGRAM→UDP; proto: IPPROTO_TCP→TCP 协议, IPPROTO_UDP→UDP 协议。 (如果要构建 TCP 连接, 可以使用默认参数配置, 即不输入任何参数。)
使用方法
<pre>addr=usocket.getaddrinfo('www.01studio.org', 80)[0][-1]</pre>
获取 Socket 通信格式地址。返回: ('47.91.208.161', 80)
<pre>s.connect(address)</pre>
创建连接。address: 地址格式为 IP+端口。例: ('192.168.1.115', 10000)
<pre>s.send(bytes)</pre>
发送。bytes: 发送内容格式为字节
<pre>s.recv(bufsize)</pre>
接收数据。bufsize: 单次最大接收字节个数。
<pre>s.bind(address)</pre>
绑定, 用于服务器角色
<pre>s.listen([backlog])</pre>
监听, 用于服务器角色。backlog: 允许连接个数, 必须大于 0。
<pre>s.accept()</pre>
接受连接, 用于服务器角色。
*其它更多用法请阅读 MicroPython 文档: (搜索:usocket) <a href="http://docs.micropython.01studio.org/zh_CN/latest/library/usocket.html">http://docs.micropython.01studio.org/zh_CN/latest/library/usocket.html</a>

表 6-6 Socket 对象

本实验中 pyBoard 属于客户端, 因此只用到客户端的函数即可。实验代码编写流程如下:

导入相关模块

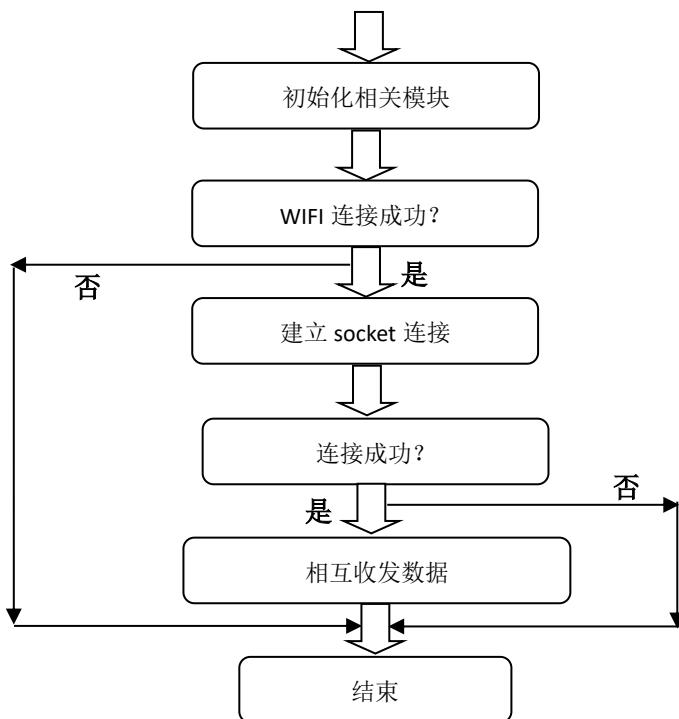


图 6-31 代码编写流程图

实验参考代码：

```

...
实验名称: W5500 以太网模块 Socket 通信
版本: v1.0
日期: 2019.11
作者: 01Studio
说明: 通过 Socket 编程实现 pyBoard+W5500 以太网模块与电脑服务器助手建立 TCP 连接, 相互收发数据。
...
import network,usocket,pyb
from machine import I2C,Pin
from ssd1306 import SSD1306_I2C
#初始化相关模块
  
```

```
i2c = I2C(sda=Pin('Y10'), scl=Pin('Y9'))  
oled = SSD1306_I2C(128, 64, i2c, addr=0x3c)  
  
#socket 数据接收中断标志位  
socket_node = 0  
  
#初始化以太网模块  
nic = network.WIZNET5K(pyb.SPI(2), pyb.Pin.board.Y5, pyb.Pin.board.Y4)  
nic.active(True)  
nic.ifconfig('dhcp')  
  
#判断网络是否连接成功  
if nic.isconnected():  
  
    print(nic.ifconfig()) #打印 IP 信息  
    #OLED 显示  
    oled.fill(0)    #清屏背景黑色  
    oled.text('IP/Subnet/GW:',0,0)  
    oled.text(nic.ifconfig()[0], 0, 20)  
    oled.text(nic.ifconfig()[1],0,38)  
    oled.text(nic.ifconfig()[2],0,56)  
    oled.show()  
  
    #创建 socket 连接 TCP 类似，连接成功后发送“Hello 01Studio!” 给服务器。  
    s=usocket.socket()  
    addr=( '192.168.1.116' ,10000) #服务器 IP 和端口  
    s.connect(addr)  
    s.send('Hello 01Studio!')  
  
#开启定时器，周期 100ms，执行 socket 通信接收任务
```

```
def fun(tim):
    global socket_node
    socket_node = 1
    pyb.LED(3).toggle()

tim = pyb.Timer(1,freq=10)
tim.callback(fun)

while True:
    if socket_node:
        text=s.recv(128) #单次最多接收 128 字节
        if text == '':
            pass
        else: #打印接收到的信息为字节，可以通过 decode('utf-8')转成字符串
            print(text)
            s.send('I got:' +text.decode('utf-8'))

    socket_node=0
```

网络连接代码在上一节已经讲解，这里不再重复，程序在连接成功后建了 Socket 连接，连接成功发送 ‘Hello 01Studio!’ 信息到服务器。另外 OpenMV 定时器设定了每 100ms 处理从服务器接收到的数据。将接收到数据通过串口打印和重新发送给服务器。

### ● 实验结果：

先在电脑端打开网络调试助手并建立服务器，软件路径在：零一科技

(01Studio) MicroPython 开发套件配套资料\01-开发工具\01-Windows\网络调试助手 下的 NetAssist.exe , 直接双击打开即可!

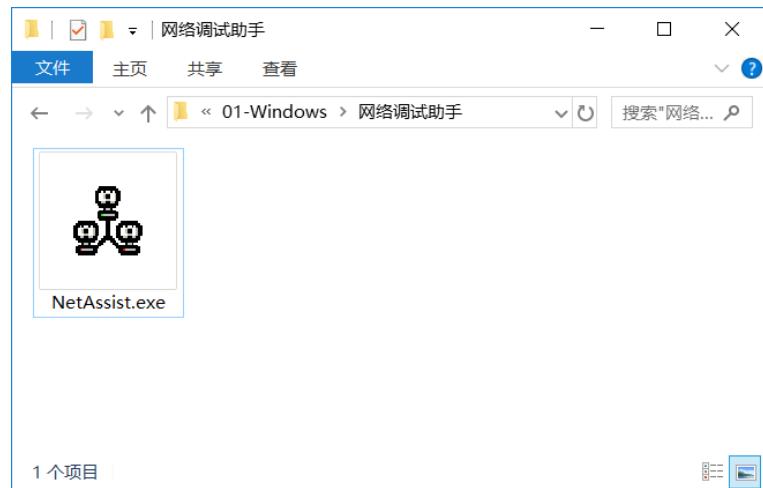


图 6-32 网络调试助手

以下是新建服务器的方法，打开网络调试助手后在左上角协议类型选择 TCP Server；中间的本地 IP 地址是自动识别的，不要修改，这个就是服务器的 IP 地址。然后端口写 10000（0-65535 都可以。），点击连接，成功后红点亮。如下图：



图 6-33 TCP 服务器配置

在时候服务器已经在监听状态！用户需要根据自己的实际情况自己的服务器 IP 地址+端口。即修改上面的代码以下部分内容。（服务器 IP 和端口可以在网络调试助手找到。）

```
addr=('192.168.1.115',10000) #服务器IP和端口
```

下载程序，开发板成功连接网络后，发起了 socket 连接，连接成功可以可以看到网络调试助手收到了开发板发来的信息。在下方列表多了一个连接对象，点击选中：

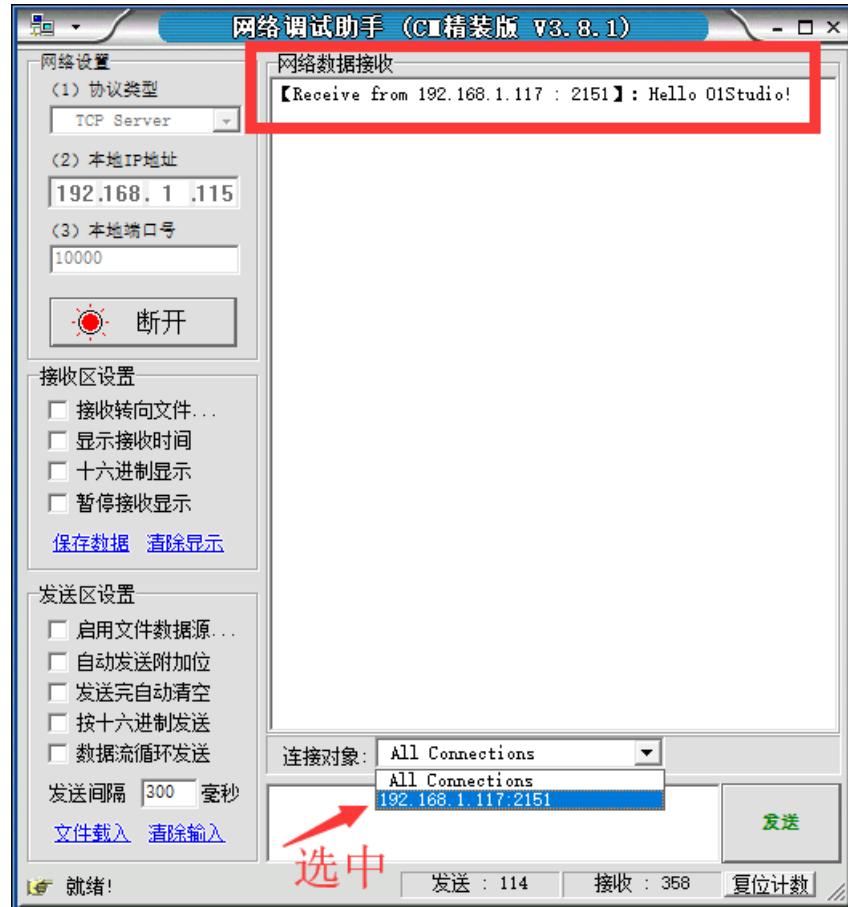


图 6-34 选中连接对象

选中后我们在发送框输入信息“Hi”，点击发送，可以看到开发板的REPL打印出来信息 Hi。为字节数据。另外由于程序将收到的信息发回给服务器，所以在网络调试助手中也接收到开发板返回的信息：I got:Hi。

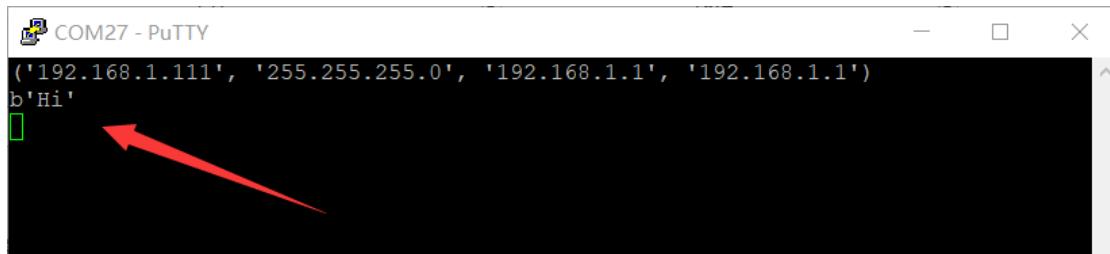


图 6-35

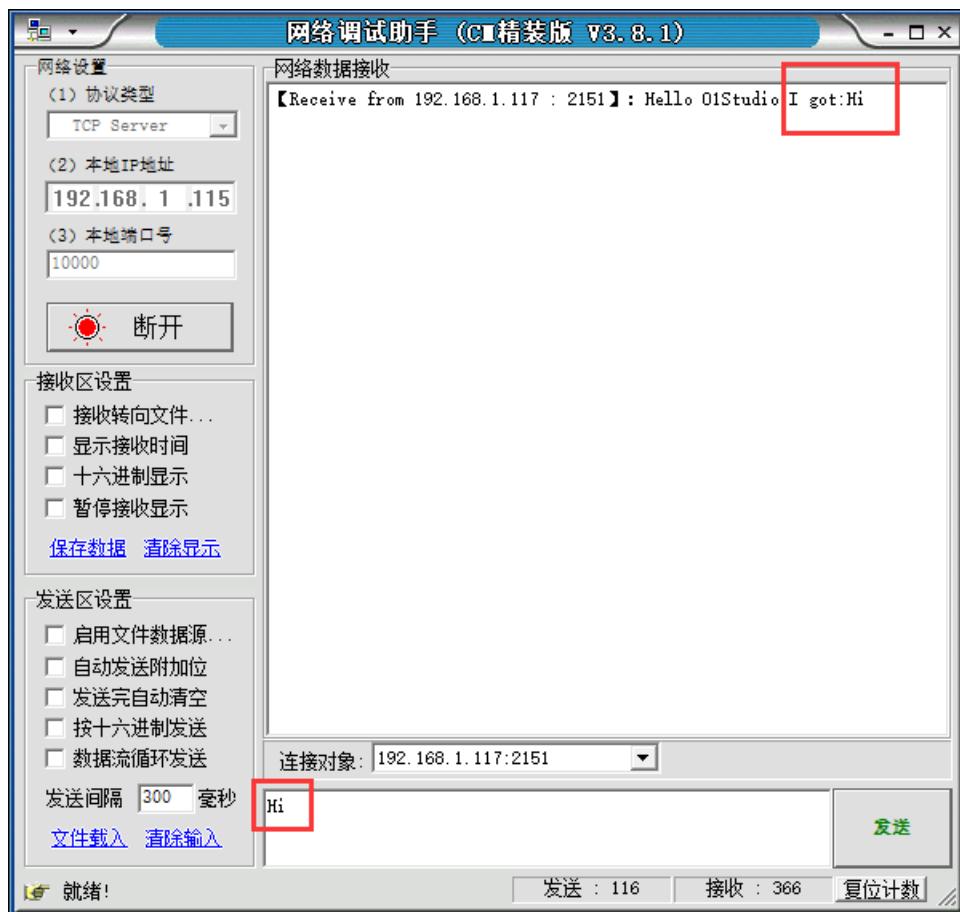


图 6-36 服务器发送数据

### ● 总结：

通过本节学习，我们了解了 `socket` 通信原理以及使用 MicroPython 进行 `socket` 编程并且通信的实验。得益于优秀的封装，让我们可以直接面向 `socket` 对象编程就可以快速实现 `socket` 通信，从而开发更多的网络应用，例如将前面采集到的传感器数据发送到服务器。

### 6.4.3 MQTT 通信

- **前言：**

上一节，我们学习了 `Socket` 通信，当服务器和客户端建立起连接时，就可以相互通信了。在互联网应用大多使用 `WebSocket` 接口来传输数据。而在物联网应用中，常常出现这样的情况：海量的传感器，需要时刻保持在线，传输数据量非常低，有着大量用户使用。如果仍然使用 `socket` 作为通信，那么服务器的压力和通讯框架的设计随着数量的上升将变得异常复杂！

那么有无一个框架协议来解决这个问题呢，答案是有的。那就是 `MQTT`(消息队列遥测传输)。

- **实验平台：**

`pyBoard` 开发套件和 W5500 以太网模块。

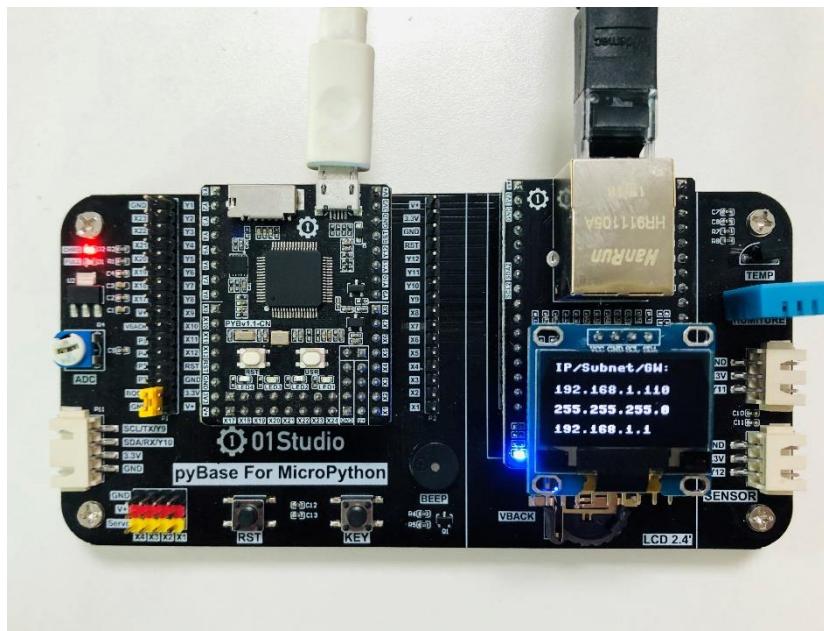


图 6-37 `pyBoard` 开发套件

- **实验目的：**

通过编程让 `pyBoard` 实现 `MQTT` 协议信息的发布和订阅（接收）。

- **实验讲解：**

`MQTT` 是 IBM 于 1999 年提出的，和 `HTTP` 一样属于应用层，它工作在 `TCP/IP`

协议族上，通常还会调用 socket 接口。是一个基于客户端-服务器的消息发布/订阅传输协议。其特点是协议是轻量、简单、开放和易于实现的，这些特点使它适用范围非常广泛。在很多情况下，包括受限的环境中，如：机器与机器（M2M）通信和物联网（IoT）。其在，通过卫星链路通信传感器、偶尔拨号的医疗设备、智能家居、及一些小型化设备中已广泛使用。

总结下来 MQTT 有如下特性/优势：

- 异步消息协议
- 面向长连接
- 双向数据传输
- 协议轻量级
- 被动数据获取

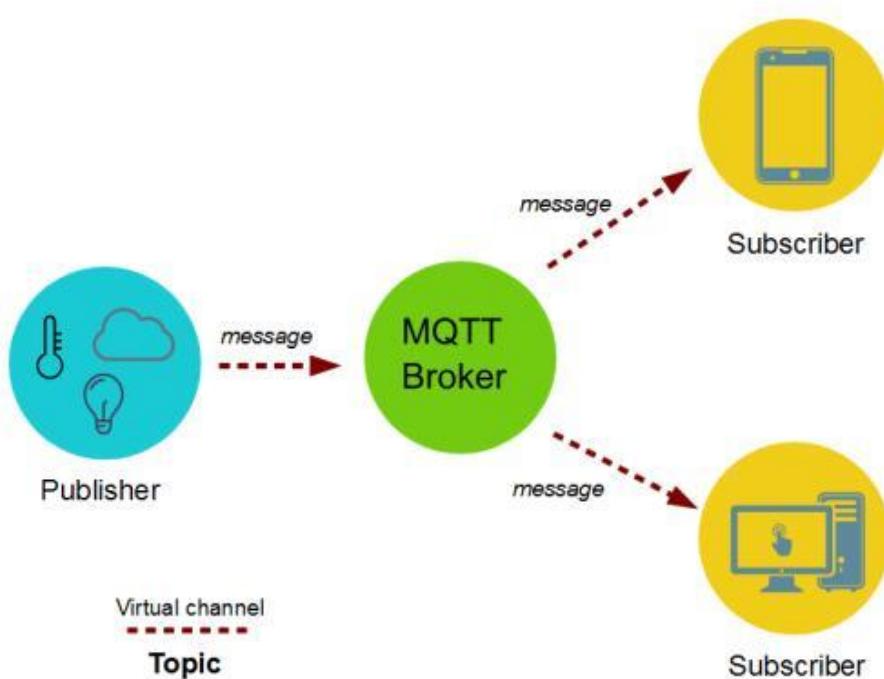


图 6-38 MQTT 通信流程

从上图可以看到，MQTT 通信的角色有两个，分别是服务器和客户端。服务器只负责中转数据，不做存储；客户端可以是信息发送者或订阅者，也可以同时是两者。具体如下图：

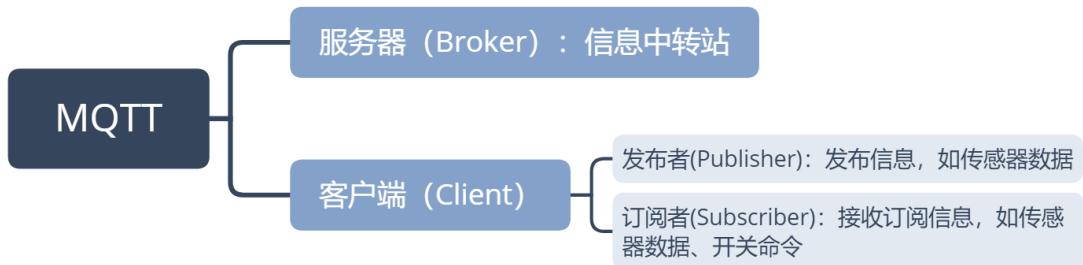


图 6-39 MQTT 角色说明

确定了角色后是如何传输数据呢？下表示 MQTT 最基本的数据帧格式，例如温度传感器发布主题“Temperature”编号,消息是“25”（表示温度）。那么所有订阅了这个主题编号的客户端（手机应用）就会收到相关信息，从而实现通信。如下表所示：

MQTT 数据帧格式	
Topic ID (主题编号)	Message (消息)
Temperature	25

表 6-7 MQTT 数据格式

由于特殊的发布/订阅机制，服务器不需要存储数据（当然也可以在服务器的设备上建立一个客户端来订阅保存信息），因此非常适合海量设备的传输。

人生苦短，而 MicroPython 已经封装好了 MQTT 客户端的库文件。让我们的应用变得简单美妙。OpenMV 的 MQTT 模块文件为例程文件夹里面的 `mqtt.py` 文件。使用方法如下：

构造函数
<code>client= mqtt.MQTTClient(client_id, server, port)</code>
构建 MQTT 客户端对象。
<code>client_id</code> : 客户端 ID，具有唯一性；
<code>server</code> : 服务器地址，可以是 IP 或者网址；
<code>port</code> : 服务器端口。（默认是 1883，服务器通常采用的端口，也可以自定义。）
使用方法
<code>client.connect()</code>

连接到服务器。
<code>client.publish(TOPIC,message)</code>
发布。TOPIC: 主题编号; message: 信息内容, 例: 'Hello 01Studio!'
<code>client.subscribe(TOPIC)</code>
订阅。TOPIC: 主题编号。
<code>client.set_callback(callback)</code>
设置回调函数。callback: 订阅后如果接收到信息, 就执行相名称的回调函数。
<code>client.check_msg()</code>
检查订阅信息。如收到信息就执行设置过的回调函数 callback。

表 6-8 MQTT 客户端对象

由于客户端分为发布者和订阅者角色, 因此为了方便大家更好理解, 本实验分开两个案例来编程, 分别为发布者和订阅者。再结合 MQTT 网络调试助手来测试。代表编写流程图如下:

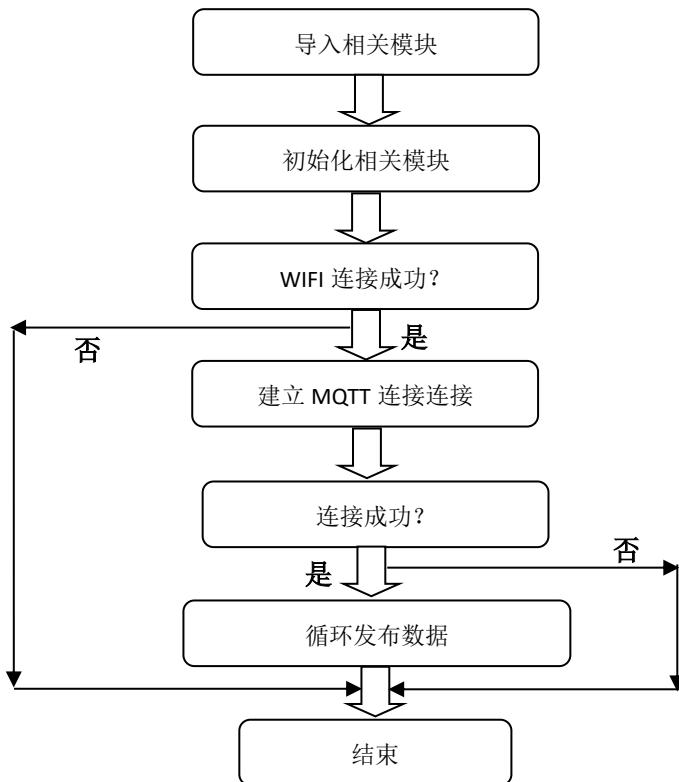


图 6-40 发布者代码编写流程

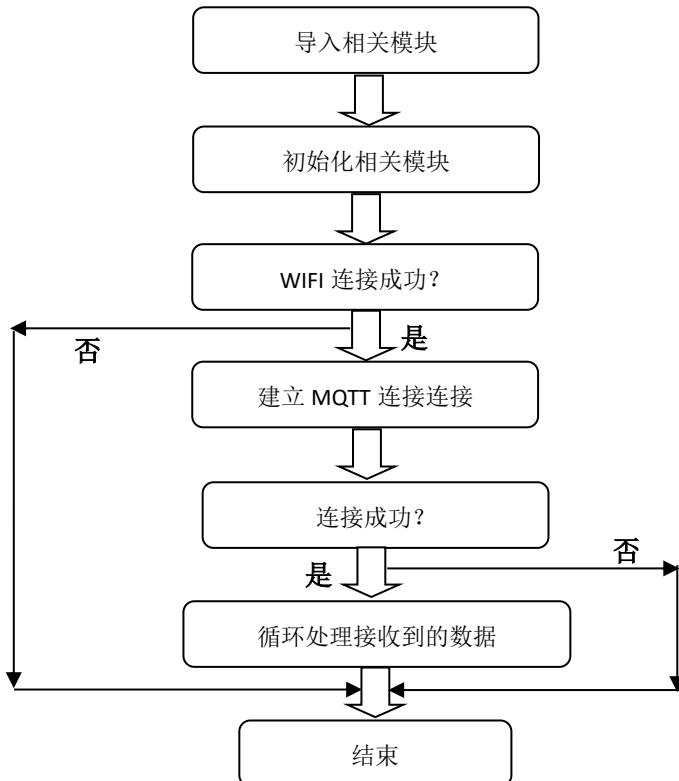


图 6-41 订阅者代码编写流程

发布者（publish）参考代码：

```

...
实验名称: W5500 以太网模块 MQTT 通信
版本: v1.0
日期: 2019.11
作者: 01Studio
说明: 通过 Socket 编程实现 pyBoard+W5500 以太网模 MQTT 通信 发布者
(publish)。
...

```

```

import network,pyb,time
from machine import I2C,Pin
from ssd1306 import SSD1306_I2C
from simple import MQTTClient

```

```

#初始化 OLED
i2c = I2C(sda=Pin('Y10'), scl=Pin('Y9'))
oled = SSD1306_I2C(128, 64, i2c, addr=0x3c)

#初始化以太网模块
nic = network.WIZNET5K(pyb.SPI(2), pyb.Pin.board.Y5, pyb.Pin.board.Y4)
nic.active(True)
nic.ifconfig('dhcp')

#判断网络是否连接成功
if nic.isconnected():

    print(nic.ifconfig()) #打印 IP 信息

    #OLED 数据显示
    oled.fill(0)    #清屏背景黑色
    oled.text('IP/Subnet/GW:',0,0)
    oled.text(nic.ifconfig()[0], 0, 20)
    oled.text(nic.ifconfig()[1],0,38)
    oled.text(nic.ifconfig()[2],0,56)
    oled.show()

SERVER = 'mq.tongxinmao.com'
PORT = 18830
CLIENT_ID = '01Studio-pyBoard' # 客户端 ID
TOPIC = '/public/01Studio/1' # TOPIC 名称

client = MQTTClient(CLIENT_ID, SERVER, PORT)
client.connect()

```

```
while (True):
    client.publish(TOPIC, "Hello 01Studio!") #发布消息
    time.sleep_ms(1000) #延时 1 秒
```

订阅者（subscribe）参考代码：

```
...
实验名称: W5500 以太网模块 MQTT 通信
版本: v1.0
日期: 2019.11
作者: 01Studio
说明: 通过 Socket 编程实现 pyBoard+W5500 以太网模 MQTT 通信 订阅者
(subscribe)。
...
import network,pyb,time
from machine import I2C,Pin
from ssd1306 import SSD1306_I2C
from simple import MQTTClient

#初始化 OLED
i2c = I2C(sda=Pin('Y10'), scl=Pin('Y9'))
oled = SSD1306_I2C(128, 64, i2c, addr=0x3c)

#初始化以太网模块
nic = network.WIZNET5K(pyb.SPI(2), pyb.Pin.board.Y5, pyb.Pin.board.Y4)
nic.active(True)
nic.ifconfig('dhcp')
```

```
#设置 MQTT 回调函数,有信息时候执行

def MQTT_callback(topic, msg):
    print('topic: {}'.format(topic))
    print('msg: {}'.format(msg))

#判断网络是否连接成功

if nic.isconnected():

    print(nic.ifconfig()) #打印 IP 信息

    #OLED 数据显示

    oled.fill(0)    #清屏背景黑色

    oled.text('IP/Subnet/GW:',0,0)

    oled.text(nic.ifconfig()[0], 0, 20)

    oled.text(nic.ifconfig()[1],0,38)

    oled.text(nic.ifconfig()[2],0,56)

    oled.show()

SERVER = 'mq.tongxinmao.com'

PORT = 18830

CLIENT_ID = '01Studio-pyBoard' # 客户端 ID

TOPIC = '/public/01Studio/1' # TOPIC 名称

client = MQTTClient(CLIENT_ID, SERVER, PORT)

client.set_callback(MQTT_callback) #配置回调函数

client.connect()

client.subscribe(TOPIC) #订阅主题
```

```

while (True):
    client.check_msg() #检测是否收到信息，收到则执行回调函数打印。
    time.sleep_ms(300) #接收间隔

```

从以上代码可以看到发布者和订阅者的编程方式相近，另外本实验需要一个 MQTT 服务器（Broker），这里使用的是跟我们将用到的 MQTT 在线网络助手同一个服务器和端口。

```

SERVER = 'mq.tongxinmao.com'
PORT = 18830

```

### ● 实验结果：

我们将 01Studio 示例程序中的 `simple.py` 文件拷贝到文件系统，复位开发板。

为了方便测试，我们可以使用 MQTT 网络助手进行调试。这里推荐一个在线 MQTT 网络调试助手：

<http://www.tongxinmao.com/txm/webmqtt.php#collapseOne>

打开上面网址，即可看到 MQTT 在线调试助手。可以配置基本信息，这里完全默认即可，点击连接。

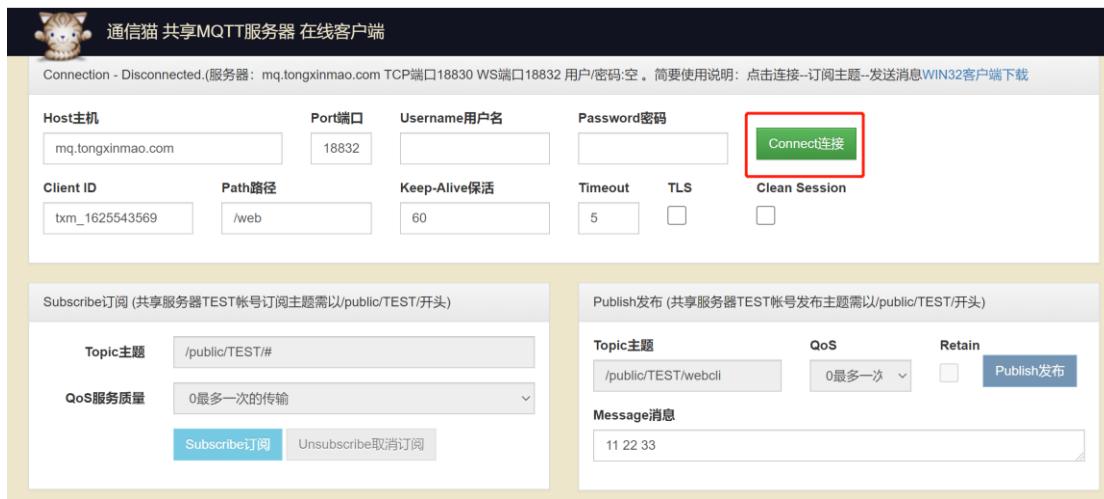


图 6-42 MQTT 在线助手

连接成功可以看到显示。



图 6-43 MQTT 助手连接成功

### 测试“发布者”代码：

我们先测试“发布者”代码。将“发布者”代码下载到开发板，然后在 MQTT 助手中订阅主题修改为：'/public/01Studio/1'（跟代码发布的主题一致），QOS 选择 0 即可。然后点击订阅主题。

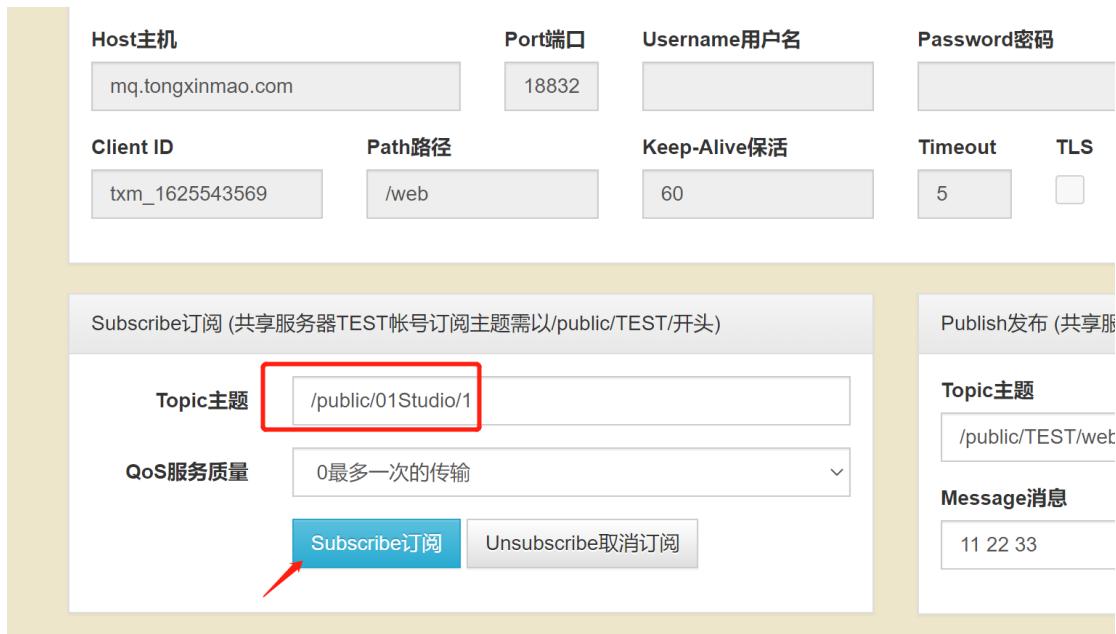


图 6-44 订阅主题

订阅后运行开发板 publish 发布程序，可以看到最下方接收框收到来自开发板发布的信息。

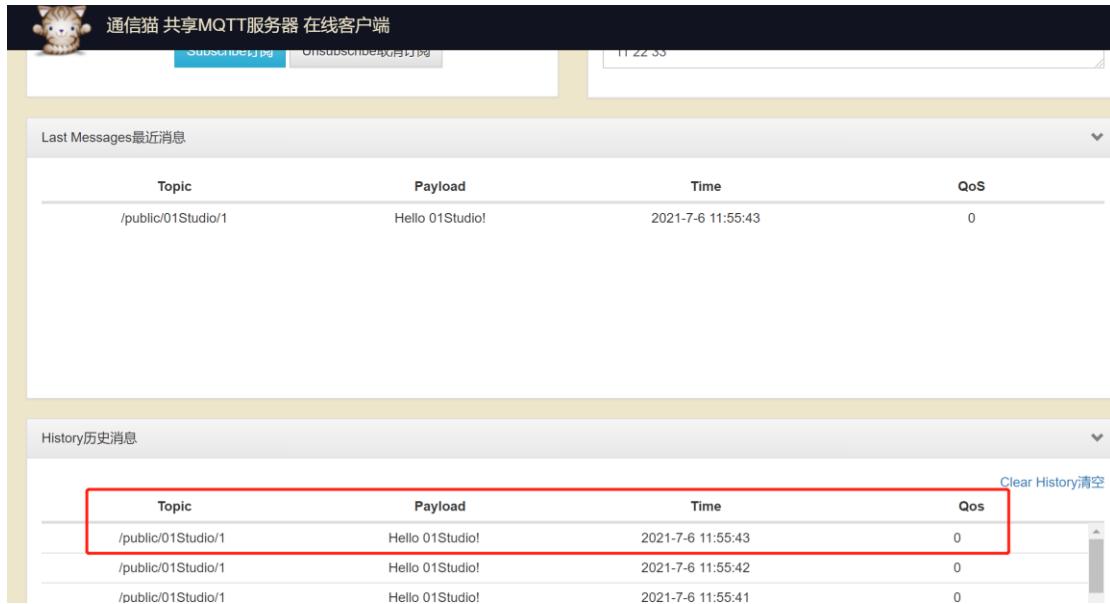


图 6-45 MQTT 助手收到开发板发布的信息

### 测试“订阅者”代码：

“订阅者”代码测试方法跟“发布者”相反。将“订阅者”代码下载到开发板，然后在电脑 MQTT 助手中发布主题修改为：'/public/01Studio/1'（跟代码发布的主题一致。）在下方空白框输入“Hello 01Studio!”。

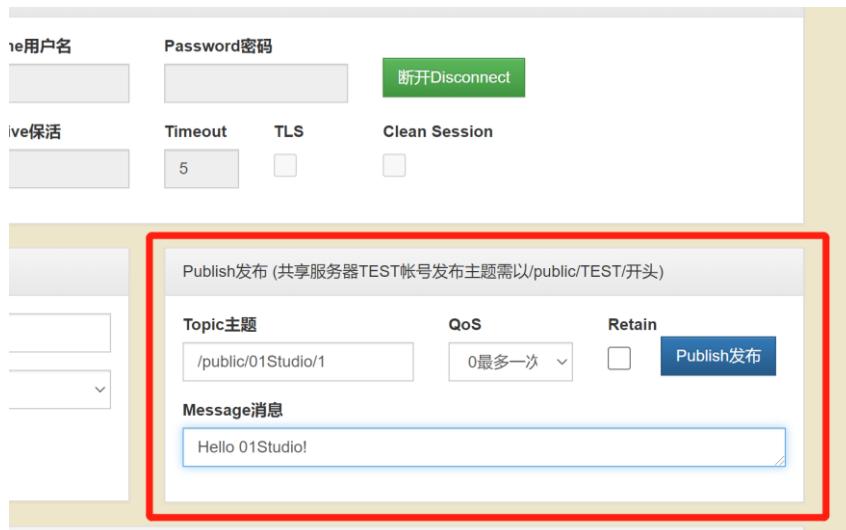


图 6-46 输入即将发布的主题和信息

开发板运行“订阅者”代码，成功连接后回到 MQTT 助手点击【发布】按钮发布信息，可以在开发板的 REPL 看到接收到的订阅信息“Hello 01Studio!”被打印出来了（数据格式为字节数据）。

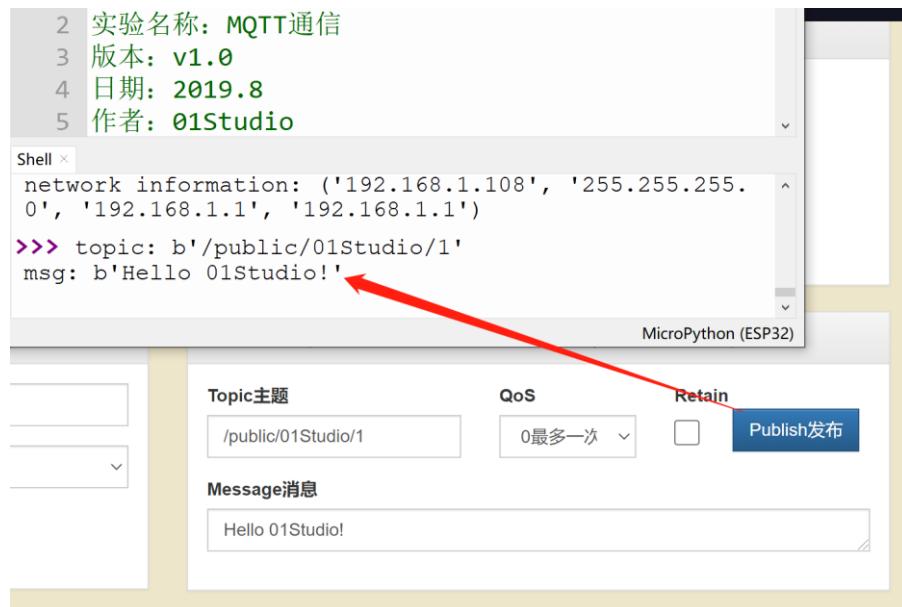


图 6-47 开发板接收到相关信息并打印

当然你也可以在同一个 MQTT 在线助手下测试发布和订阅功能，只需要将主题设置一致即可，如下图所示：



图 6-48 客户端同时发布和订阅信息

### ● 总结：

通过本节我们了解了 MQTT 通信原理以及成功实现通信。目前市面上大部分物联网云平台支持 MQTT，原理大同小异。大家可以基于不同平台协议来开发，实现自己的物联网设备远程连接。

# MicroBit 从0到1

用方块开始你的编程之旅

01Studio团队 编著



01Studio  
-让编程变得简单有趣-

# MicroPython 从0到1

用python做嵌入式编程

(基于pyBoard STM32F405平台)

01Studio团队 编著

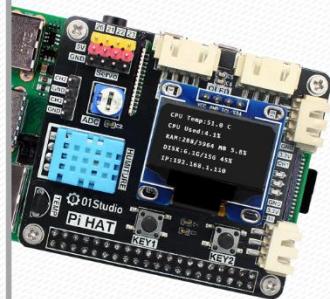


01Studio  
-让编程变得简单有趣-

# 树莓派从0到1

(基于树莓派4B平台)

01Studio团队 编著



01Studio  
-让编程变得简单有趣-

# 基于pyBoard STM32F405平台

MicroPython  
从0到1  
用python做嵌入式编程  
(基于pyBoard STM32F405平台)  
01Studio团队 编著



关注公众号，免费下载