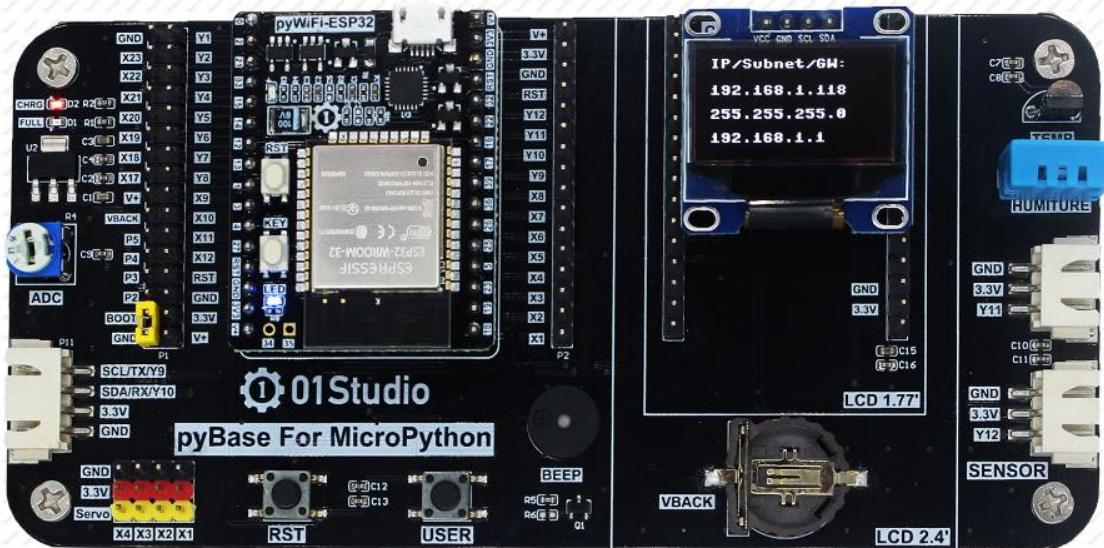


MicroPython 从0到1

用python做嵌入式编程

(基于ESP32平台)

01Studio团队 编著

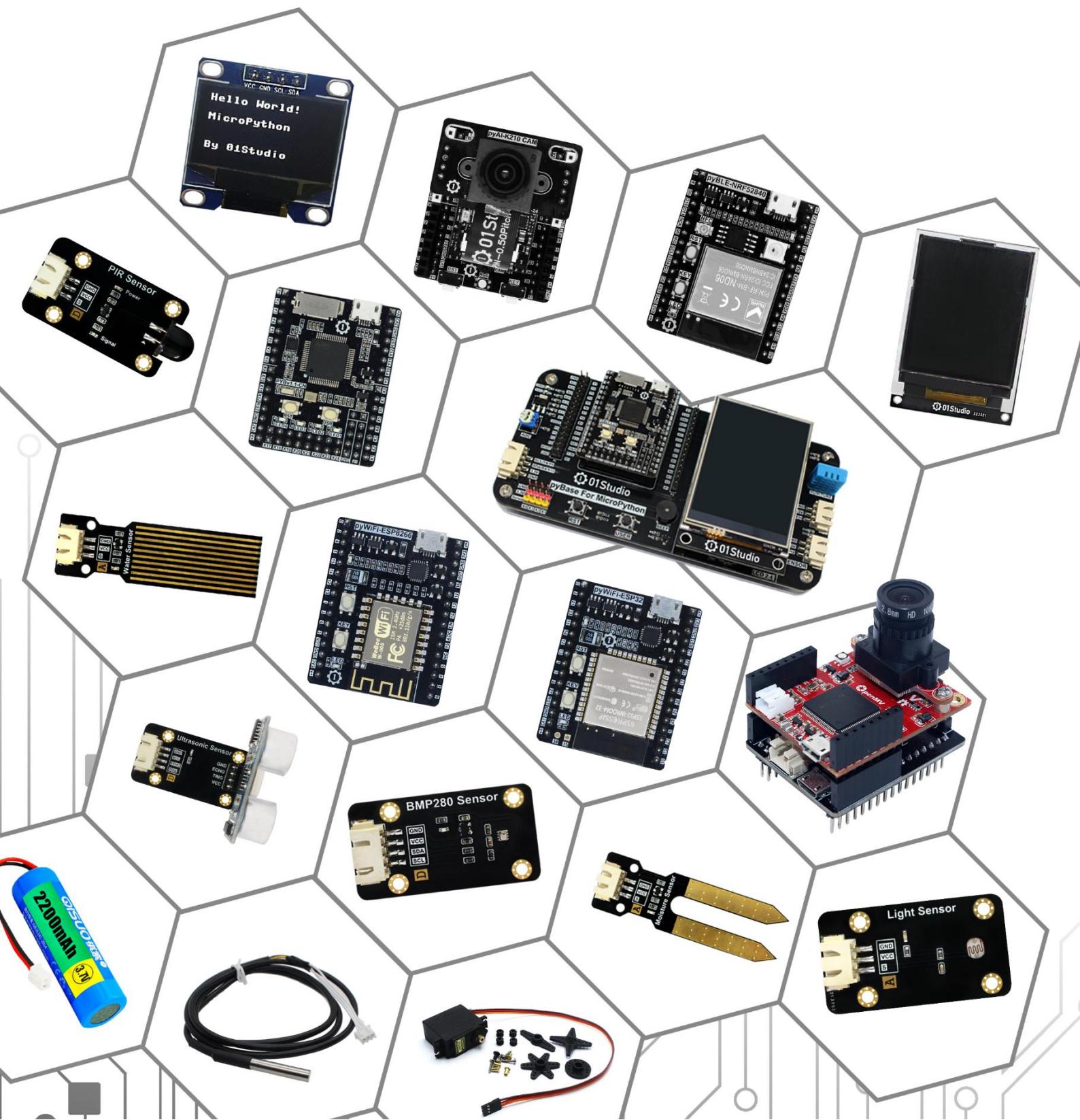


① 01Studio

-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。而这一趋势并没有停止，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，但随着开发者的数量增加，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: www.01Studio.org】

【微信公众号: 01Studio 社区】

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	2021-6-18	Jackey	1. 增加：基础实验—UART 串口通讯
v1. 2	2021-9-1	Jackey	1. 默认 IDE 从 Mu 改成 Thonny； 2. 增加 3.2 寸电阻触摸显示屏相关实验； 3. 增加继电器实验； 4. 更改 MQTT 在线助手。

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	8
1.1 MicroPython 是什么	8
1.2 MicroPython 支持的微控制器平台	9
1.3 MicroPython 相关学习资料	11
1.3.1 01Studio 技术论坛	11
1.3.2 MicroPython 库文档	12
1.3.3 MicroPython 官方网站	13
1.4 MicroPython 开发套件介绍	14
1.4.1 ESP32 平台	15
1.4.2 pyBase	20
1.4.3 IOT/通讯模块	22
1.4.4 传感器模块	25
1.4.5 拓展配件	32
第 2 章 Python 基础知识	39
2.1 原始数据类型和运算符	39
2.2 变量和集合	44
2.3 流程控制和迭代器	50
2.4 函数	54
2.5 类	57
2.6 模块	59
2.7 高级用法	60
第 3 章 开发环境快速建立	62
3.1 基于 Windows	63
3.1.1 安装开发软件 Thonny	63
3.1.2 开发套件使用	65
3.2 基于 Mac OS	82
3.2.1 安装开发软件 Thonny	82
3.2.2 开发套件使用	82
3.3 基于 Linux (树莓派)	85
3.3.1 安装开发软件 Thonny	85
3.3.2 开发套件使用	85
第 4 章 基础实验	86
4.1 点亮第一个 LED	87
4.2 按键	92
4.3 外部中断	97
4.4 定时器	102
4.5 I2C 总线 (OLED 显示屏)	106
4.6 RTC 实时时钟	113
4.7 ADC (电位器)	119
4.8 PWM (蜂鸣器)	124
4.9 UART (串口通信)	130
4.10 LCD 显示屏	137

4.11 电阻触摸屏	146
4.12 触摸屏按钮	152
第 5 章 传感器实验	159
5.1 温度传感器 DS18B20.....	160
5.2 温湿度传感器 DHT11.....	168
5.3 人体感应传感器	174
5.4 光敏传感器	180
5.5 土壤湿度传感器	186
5.6 水位传感器	192
5.7 大气压强传感器	199
5.8 超声波传感器	206
第 6 章 拓展实验	213
6.1 继电器模块	214
6.2 舵机	219
6.3 RGB 灯带	225
第 7 章 WiFi 应用	230
7.1 连接无线路由器	231
7.2 Socket 通信	237
7.3 MQTT 通信	249
7.4 WebREPL	263
第 8 章 LVGL (LittleVGL)	267
8.1 LVGL (LittleVGL) 简介	268
8.2 LCD 和触摸屏校准	269
8.3 小部件实验	277
8.3.1 Basc object (基础对象)	278
8.3.2 Bar (进度条)	282
8.3.3 Button (按钮)	286
8.3.4 Button Matrix (矩阵键盘)	290
8.3.5 Calendar (日历)	295
8.3.6 CheckBox (复选框)	299
8.3.7 Chart (图表)	302
8.3.8 Container (容器)	306
8.3.9 Drawdown List (下拉列表)	310
8.3.10 Gauge (计量器)	314
8.3.11 Keyboard (键盘)	318
8.3.12 Label (标签)	322
8.3.13 Line (线)	326
8.3.14 List (菜单)	330
8.3.15 Message box (消息框)	334
8.3.16 Page (页)	338
8.3.17 Roller (滚筒)	342
8.3.18 Slider (滑动条)	346
8.3.19 Spinner (旋转加载条)	350
8.3.20 Swtich (开关)	354

8.3.21 Table (表格)	359
8.3.22 Tabview (选项卡视图)	364
8.3.23 Textarea (文本区域)	368
8.3.24 Window (窗口)	371
8.4 项目应用	375
8.4.1 网络时钟	375

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。

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。本书主要是围绕 ESP32 平台来进行编写。

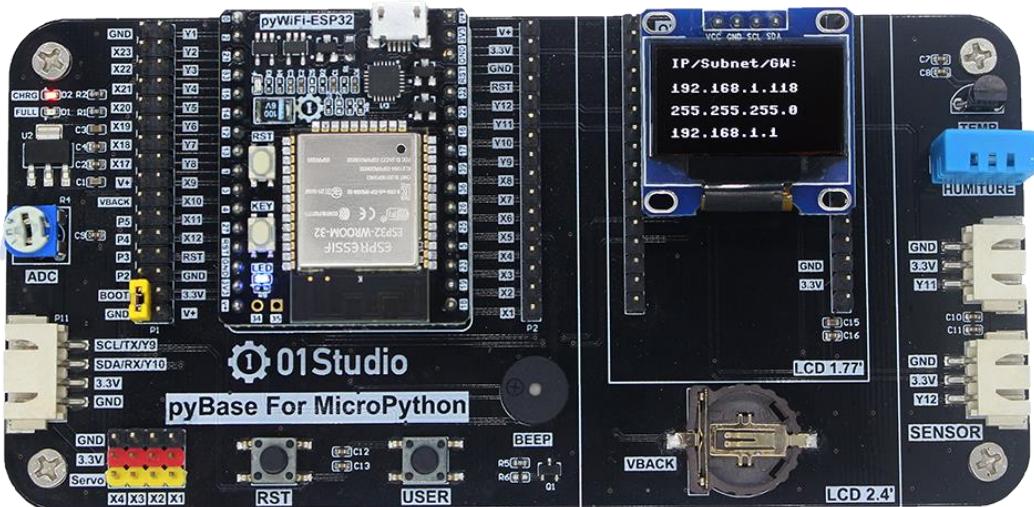


图 1-1 pyWiFi-ESP32 开发套件

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 01Studio 其它 micropython 开发平台：

STM32 平台	ESP8266 平台	ESP32 平台	NFR52840 平台	OpenMV4 平台	K210 平台
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】

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 'oxxxxy' from August 26, 2021, with 11 replies, asks about using a counter to count external pulses on a breadboard. Another post by 'Jackey' from August 25, 2021, with 39 replies, discusses collecting DHT11 data using MicroPython on ESP32. To the right, a sidebar titled '推荐内容' lists several official posts, each with a thumbnail, title, and a '精华' (highlighted) button.

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】

The screenshot shows the homepage of the MicroPython documentation. The header includes the 'MicroPython' logo, the '01Studio' logo, and the version '1.15'. A search bar is present. The main content area has a title 'MicroPython 文档 (中文)' and a welcome message: '欢迎! 你在浏览的是MicroPython文档 (中文)，由01Studio团队实时维护更新。v1.15, last updated 22 Jul 2021.' Below this, there's a section titled 'MicroPython通用文档:' with two columns of links: '相关库' (pyboard, 哥伦布, 达芬奇, ESP8266, ESP32-C3, ESP32) and '相关语言' (MicroPython 语言特征). Another section titled 'MicroPython区别' (MicroPython与CPython的区别) is also shown. At the bottom, there's a section titled '特定平台的参考手册和教程:' with links to 'pyboard 快速参考手册' and '哥伦布 (STM32F407) 快速参考手册'.

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】

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。

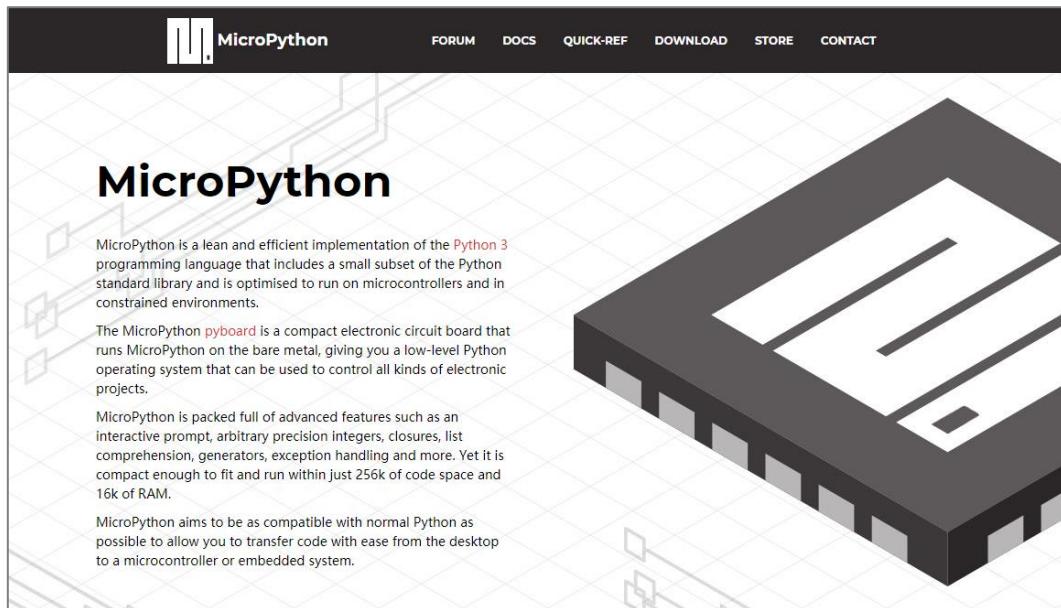


图 1-5 MicroPython 官网

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 ESP32 平台

1.4.1.1 pyWiFi-ESP32

pyWiFi-ESP32 是由 01Studio 设计研发，基于 ESP32 平台，接口兼容 MicroPython 的 pyBoard，主要特点如下：

- ◆ 兼容 pyBoard 接口
- ◆ 自动下载电路
- ◆ 板载锂电池输入接口和充电电路
- ◆ 按键和 LED 排列整齐，丝印清晰
- ◆ 全 IO 引出

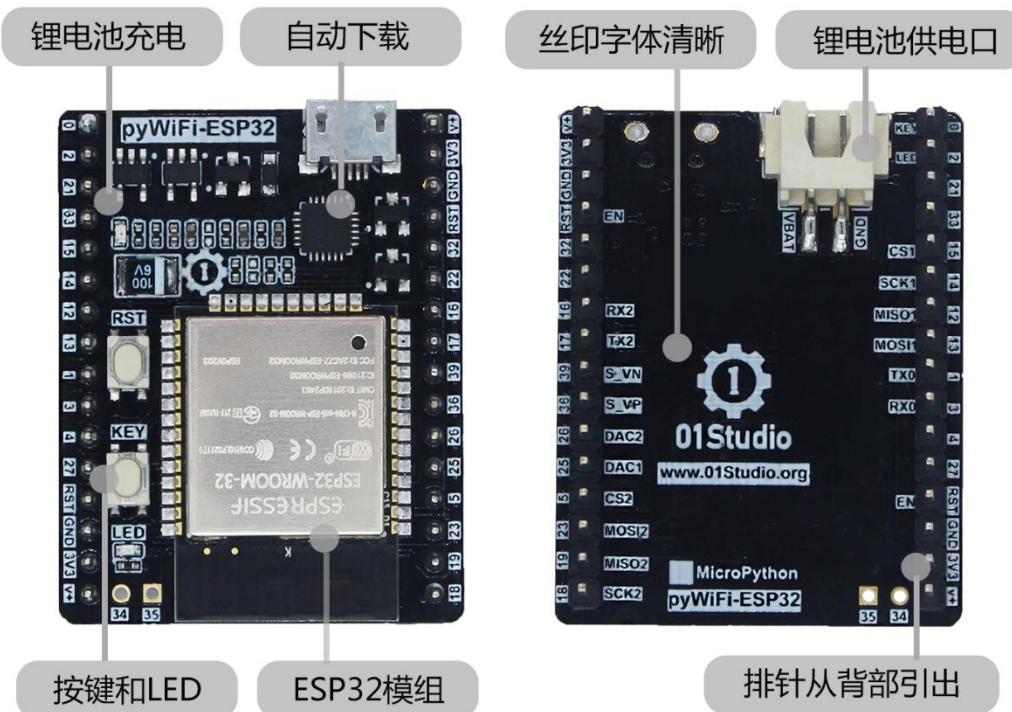


图 1-6 pyWiFi-ESP32

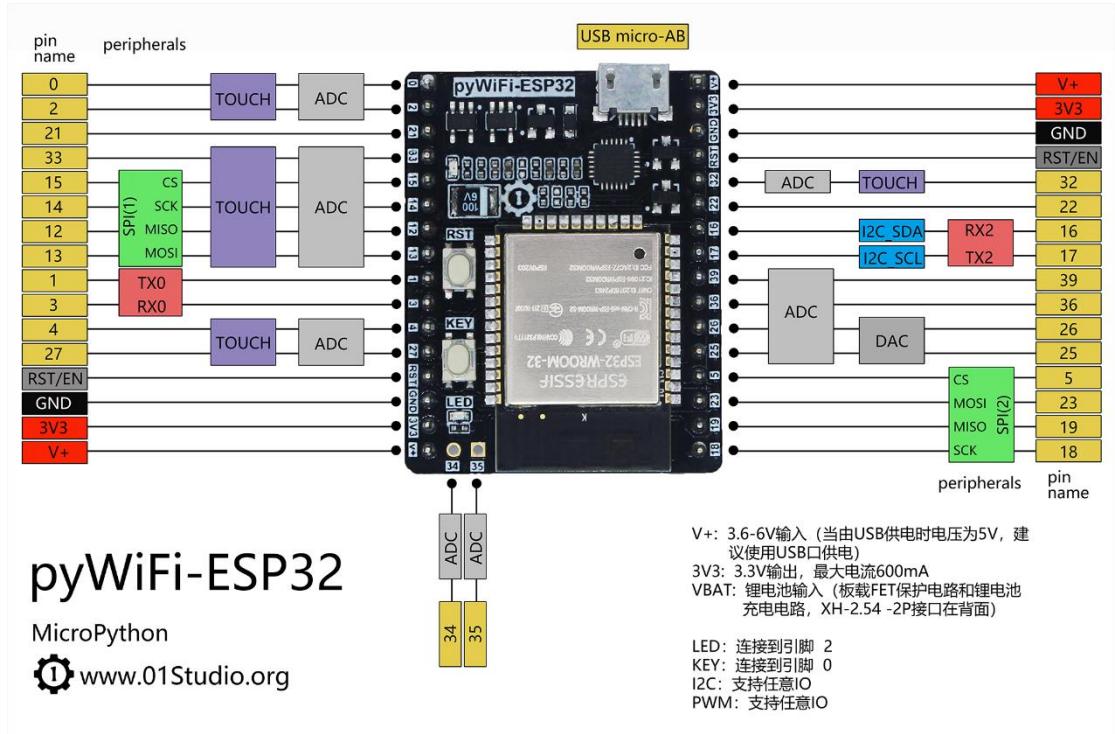


图 1-7 pyWiFi-ESP32 引脚图

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 600mA
VBAT	锂电池输入 (板载 FET 保护电路和锂电池充电电路, XH-2.54 2P 接口在背面)
LED	连接到引脚 2
KEY	连接到引脚 0
I2C	支持任意 IO
PWM	支持任意 IO

表 1-1 pyWiFi-ESP32 功能参数表

1.4.1.2 pyWiFi-ESP32P

pyWiFi-ESP32P 是由 01Studio 设计研发，基于 ESP32 平台，接口兼容 MicroPython 的 pyBoard，相对于 pyWiFi-ESP32 内置了 **8MB PSRAM**，**GPIO 16,17** 被占用。主要特点如下：

- ◆ 兼容 pyBoard 接口
- ◆ 自动下载电路
- ◆ 板载锂电池输入接口和充电电路
- ◆ 按键和 LED 排列整齐，丝印清晰
- ◆ 全 IO 引出

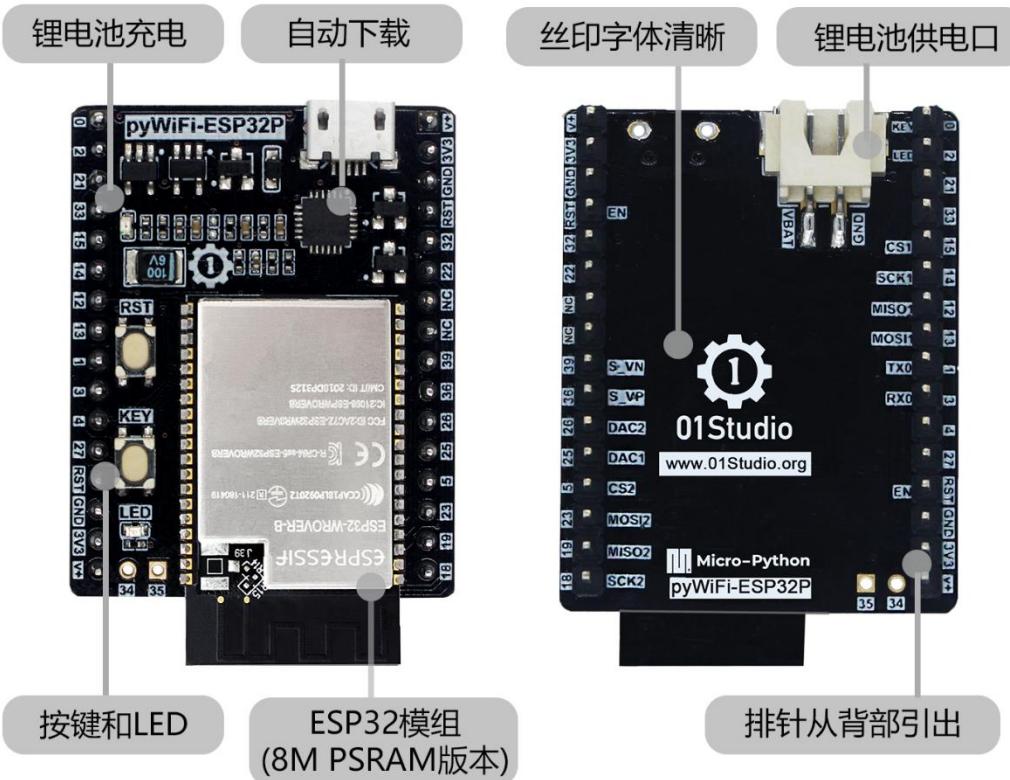


图 1-8 pyWiFi-ESP32P

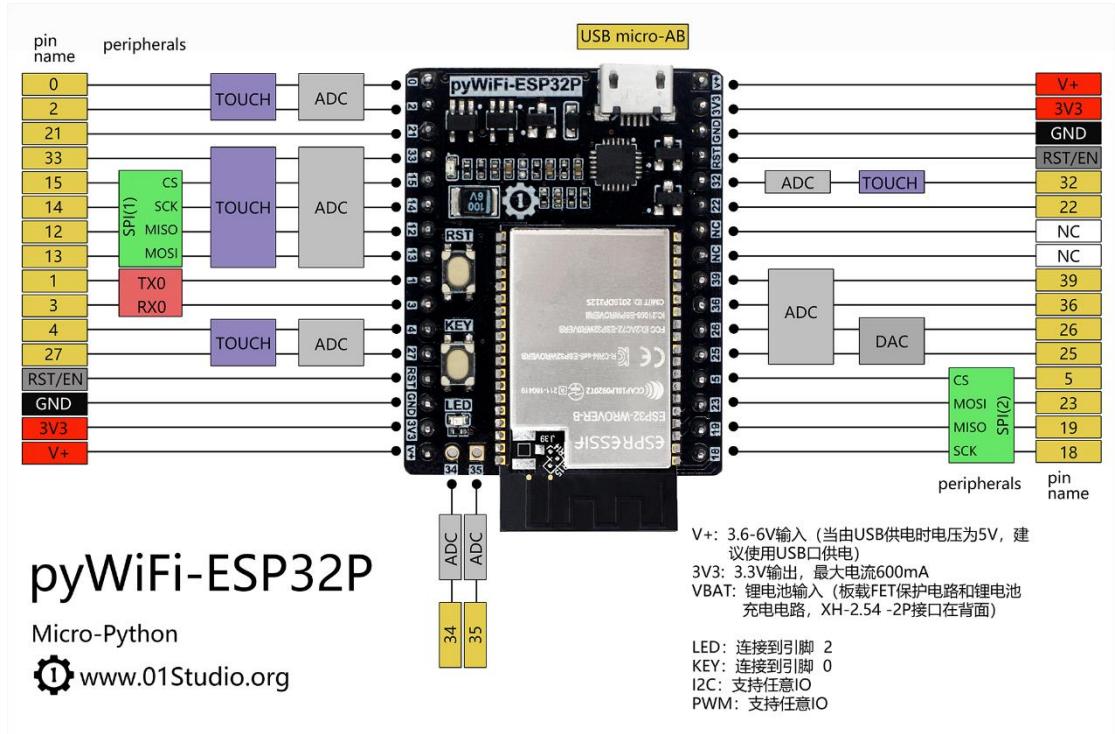


图 1-9 pyWiFi-ESP32P 引脚图

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 600mA
VBAT	锂电池输入 (板载 FET 保护电路和锂电池充电电路, XH-2.54 2P 接口在背面)
LED	连接到引脚 2
KEY	连接到引脚 0
I2C	支持任意 IO
PWM	支持任意 IO
拓展内存	8MB PSRAM

表 1-2 pyWiFi-ESP32P 功能参数表

1.4.1.3 pyWiFi-ESP32 开发套件

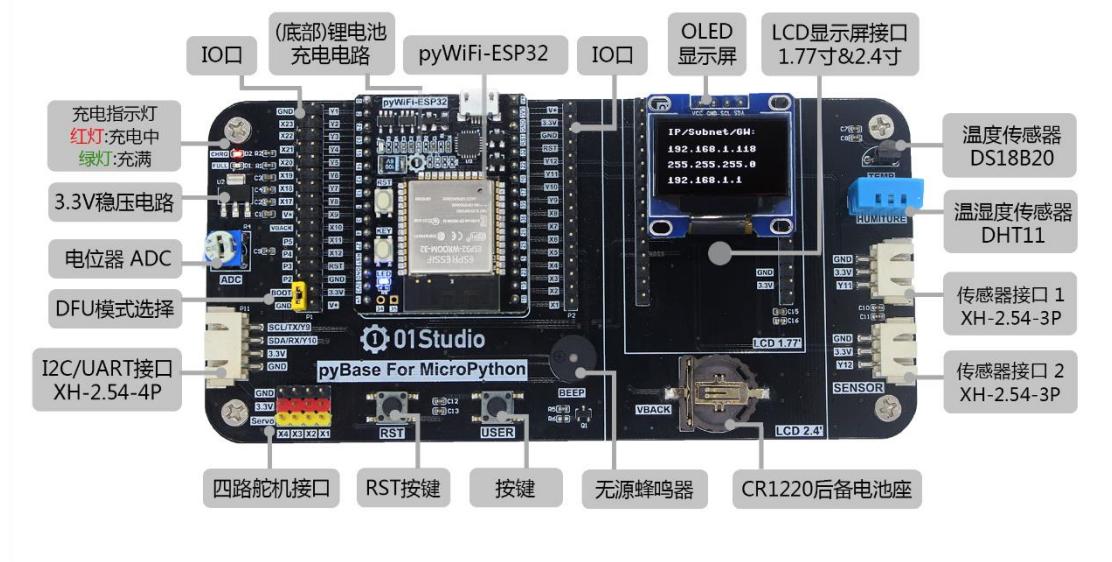


图 1-10 pyWiFi-ESP32 开发套件

主要配置	
核心板	pyWiFi-ESP32
底板	pyBase
显示屏	0.9 寸 OLED 显示屏

表 1-3

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：

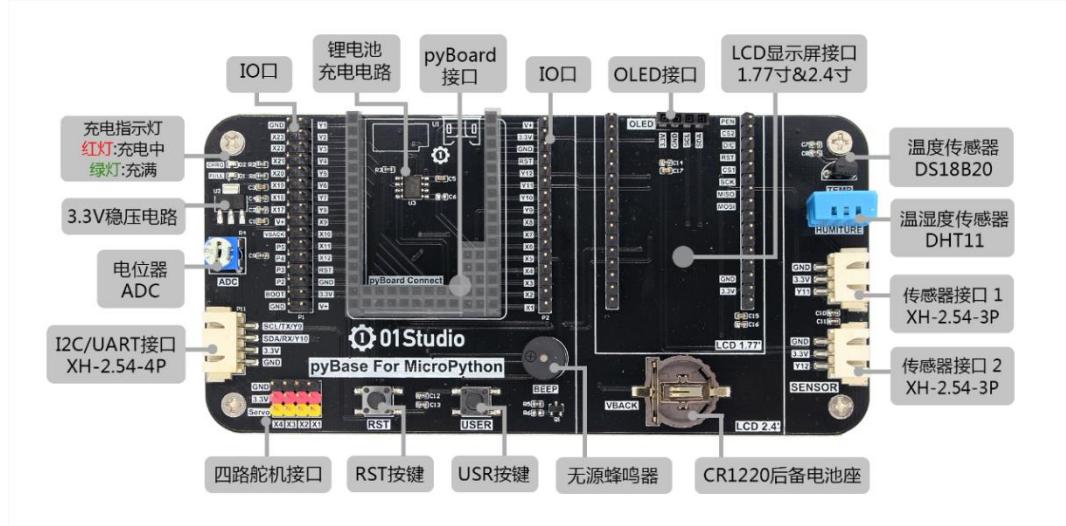


图 1-11 pyBase 功能说明

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 1-4

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

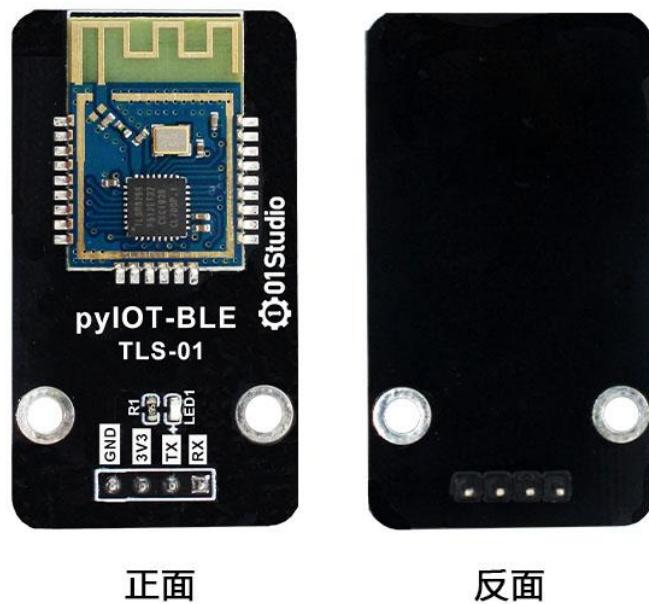


图 1-12

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 1-5

1.4.3.2 pyIOT-LORA E22

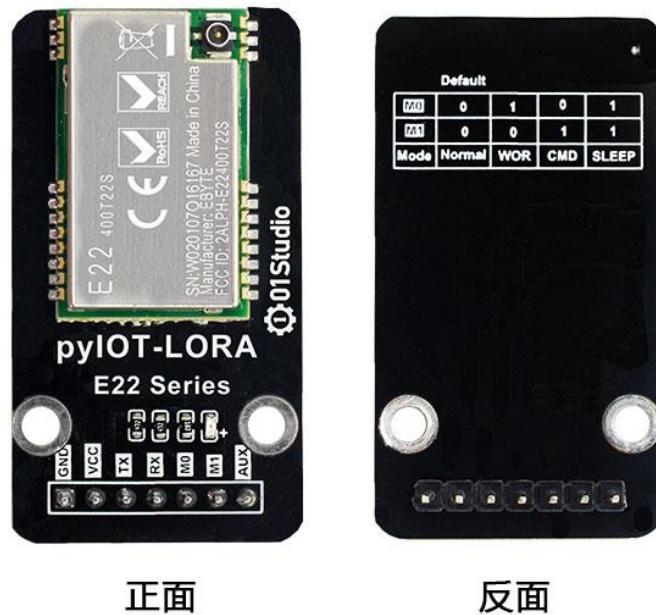


图 1-13

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 1-6

1.4.3.3 pyIOT-GPS



图 1-14

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 1-7

1.4.4 传感器模块

1.4.4.1 人体感应传感器模块

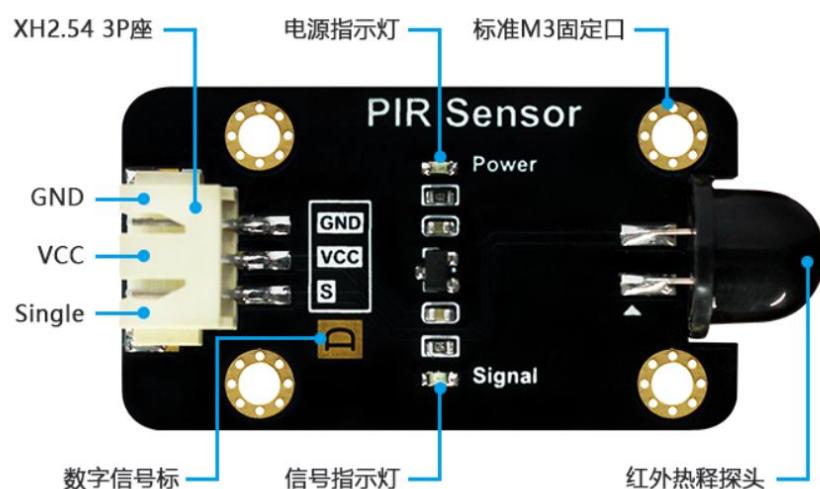


图 1-15 人体感应传感器模块

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 1-8

1.4.4.2 光敏传感器模块

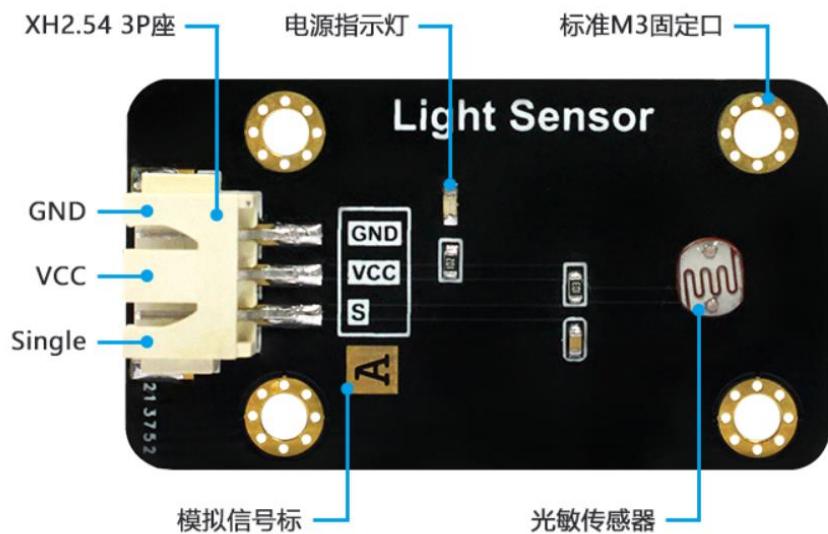


图 1-16 光敏传感器模块

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 1-9

1.4.4.3 土壤湿度传感器模块

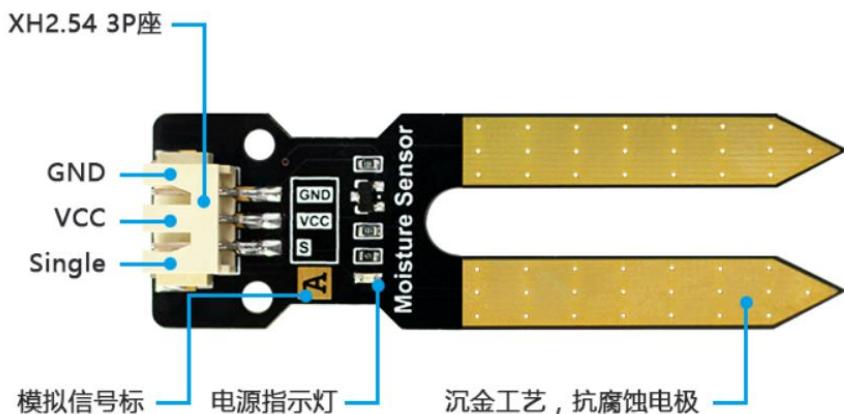


图 1-17 土壤湿度传感器模块

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 1-10

1.4.4.4 水位传感器模块

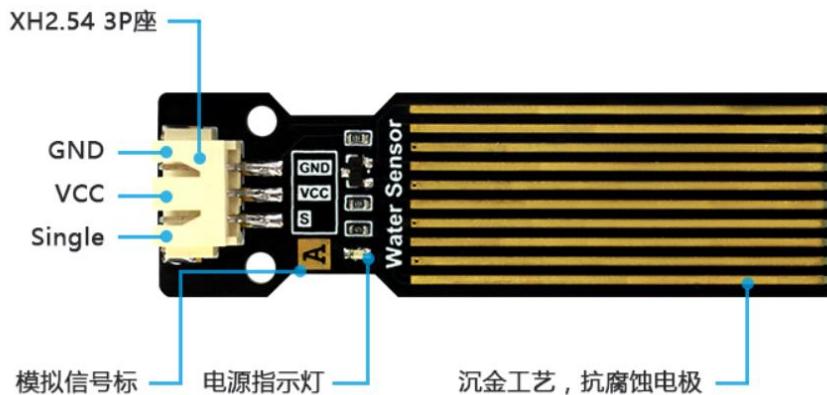


图 1-18 水位传感器模块

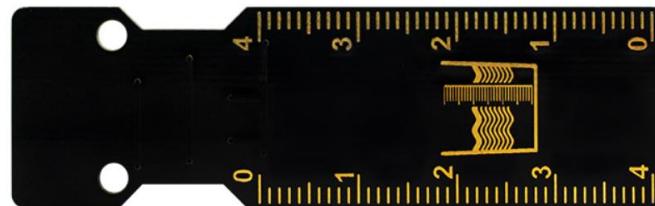


图 1-19 背面

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 1-11

1.4.4.5 大气压强传感器模块

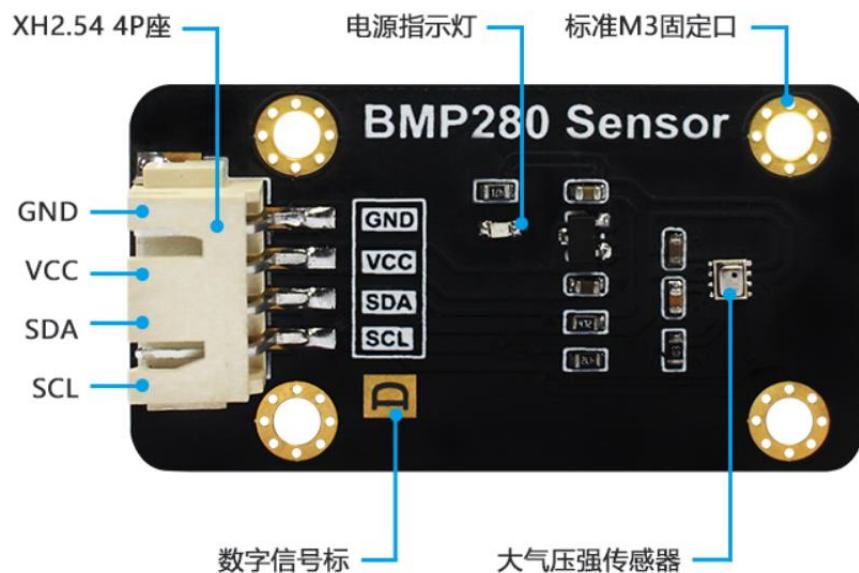


图 1-20 大气压强传感器模块

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 1-12

1.4.4.6 超声波模块

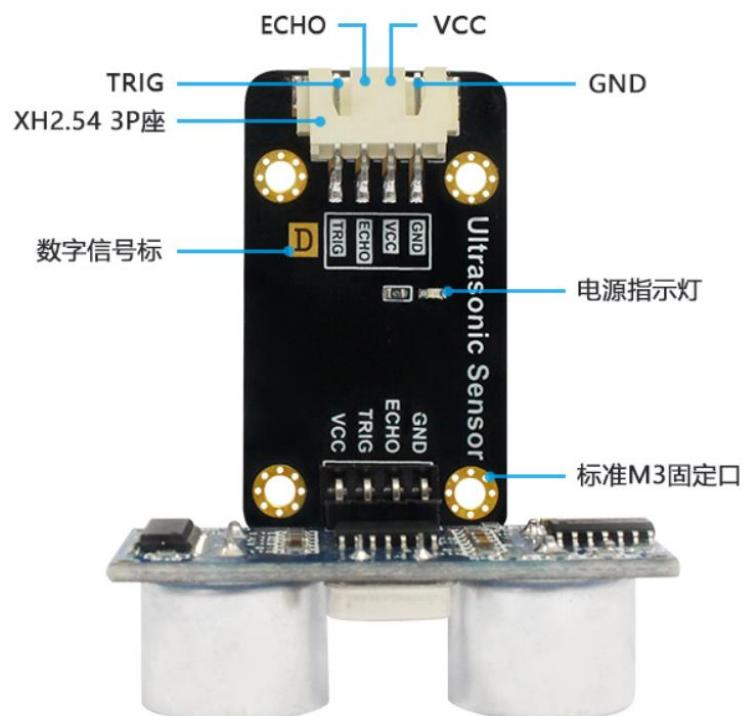


图 1-21 超声波模块

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 1-13

1.4.4.7 温度传感器模块（金属探头封装）



图 1-22 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℃
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℃
测量精度	-0.5℃
探头尺寸	5*0.6cm
引线长度	1 米

表 1-14

1.4.5 拓展配件

1.4.5.1 OLED 显示屏



图 1-23 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 1-15

1.4.5.2 1.77 寸 LCD 显示屏

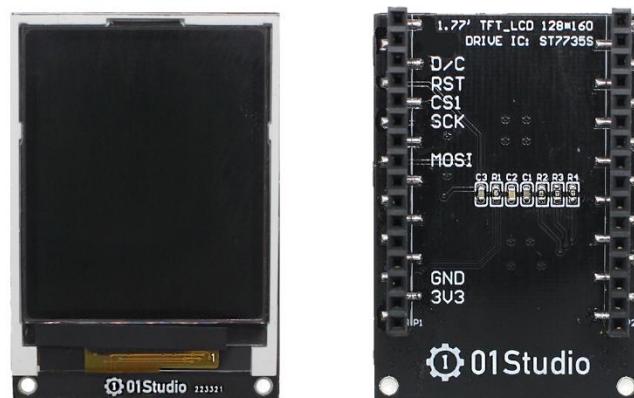


图 1-24 1.77 寸 LCD 显示屏

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 1-16

1.4.5.3 2.4 寸 LCD 显示屏（电阻触摸）

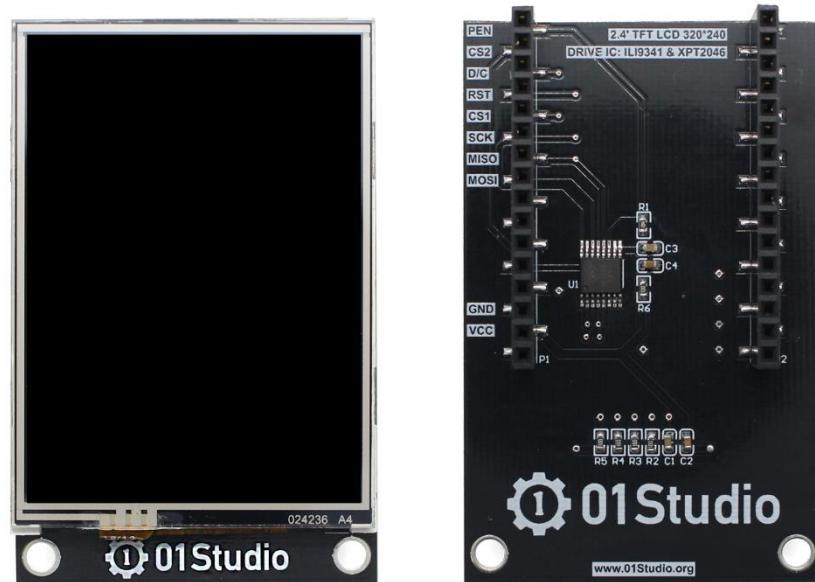


图 1-25 2.4 寸 LCD 显示屏（电阻触摸）

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 1-17

1.4.5.4 3.2 寸 LCD 显示屏（电阻触摸）

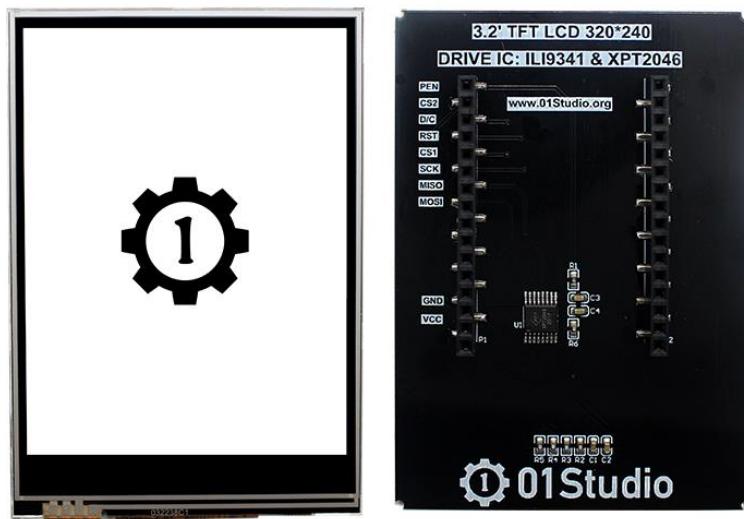


图 1-26 3.2 寸 LCD 显示屏（电阻触摸）

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 1-18

1.4.5.5 舵机

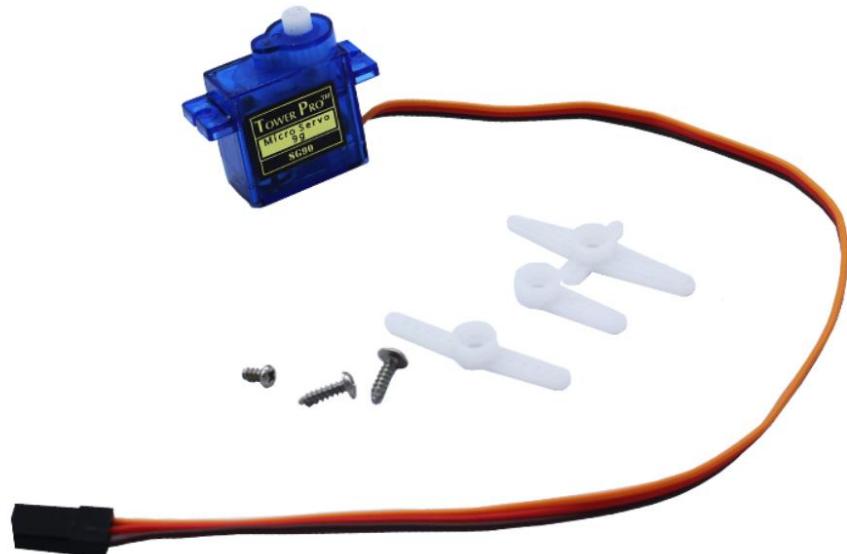


图 1-27 舵机

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 1-19

1.4.5.6 RGB 灯带

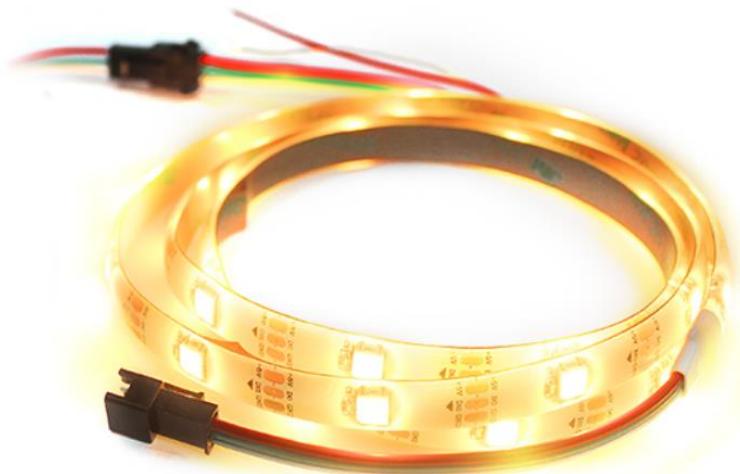


图 1-28 RGB 灯带

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 1-20

1.4.5.7 USB 转串口 TTL

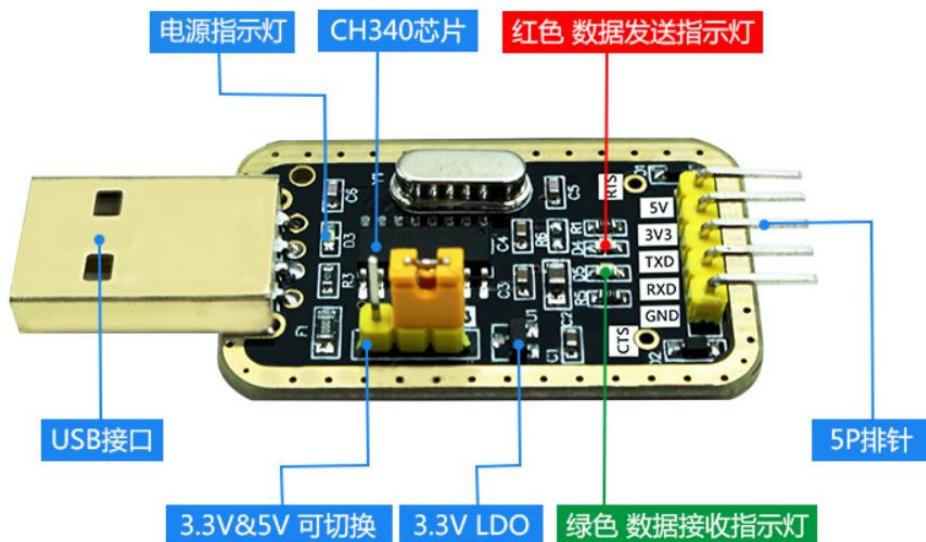


图 1-29 USB 转串口 TTL

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 1-21

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

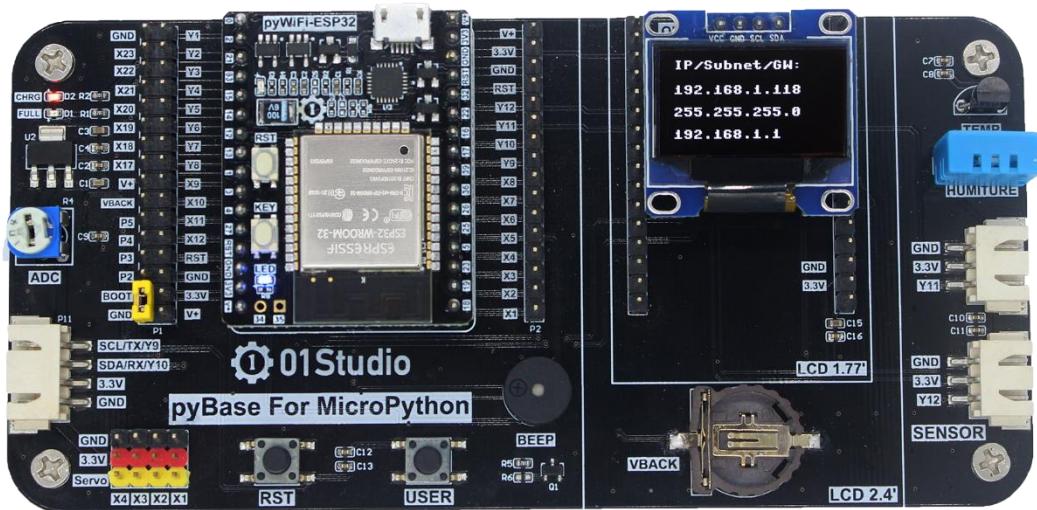
@beg
def say(say_please=False):
    msg = "Can you buy me a beer?"
    return msg, say_please

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

ESP32 是继 ESP8266 后上海乐鑫公司主推的一款芯片，可以理解支持 ESP8266 的超级升级版。相对于 ESP8266 而言，ESP32 芯片的资源非常丰富，除了拥有更多的引脚外设外，还同时支持 WIFI、蓝牙功能。

如果你已经学习过 MicroPython 基于 ESP8266 平台的内容，那么恭喜你，本部分内容的学习将会变得轻松，因为它们的开发环境和编程方式都非常相似。当然拥有更多资源和 ESP32 更适合开发一些高级的实验。让我们开始学习吧！



pyWiFi-ESP32 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

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。默认提供 Windows 的安装包。当然你也可以在在 <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：

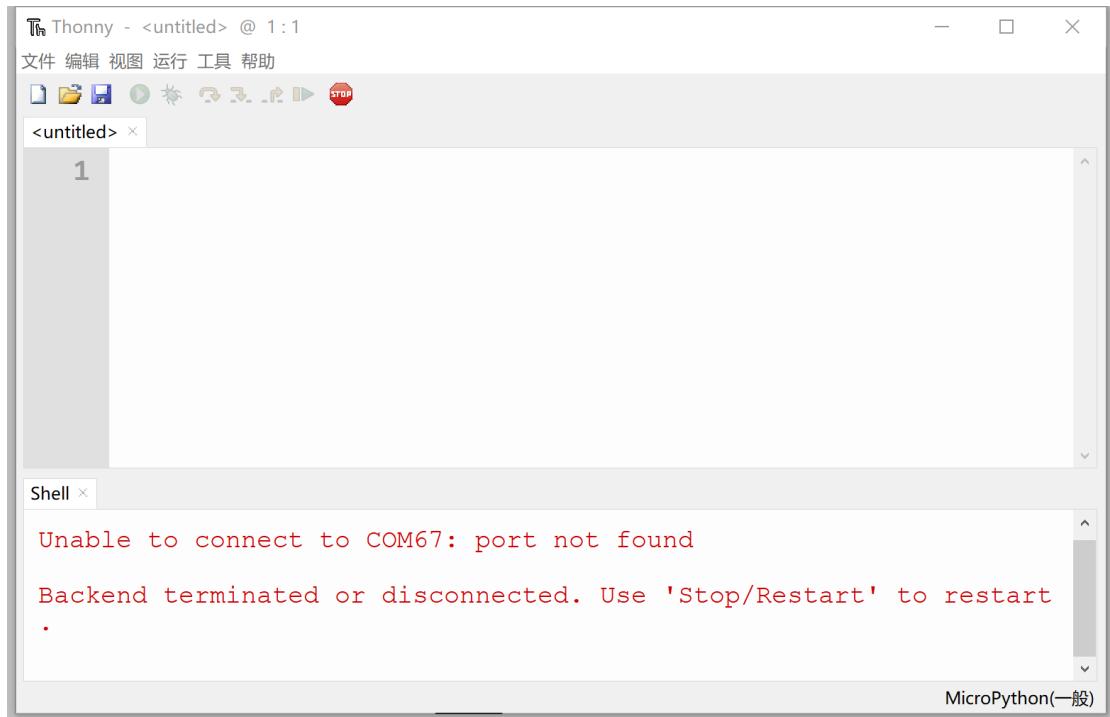


图 3-2 安装完成

至此，Thonny 安装完成。关于如何在 Thonny 上使用 pyWiFi-ESP32，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

主要是安装 USB 转串口驱动。我们将 pyWiFi-ESP32 开发板通过 MicroUSB 数据线连接到电脑：

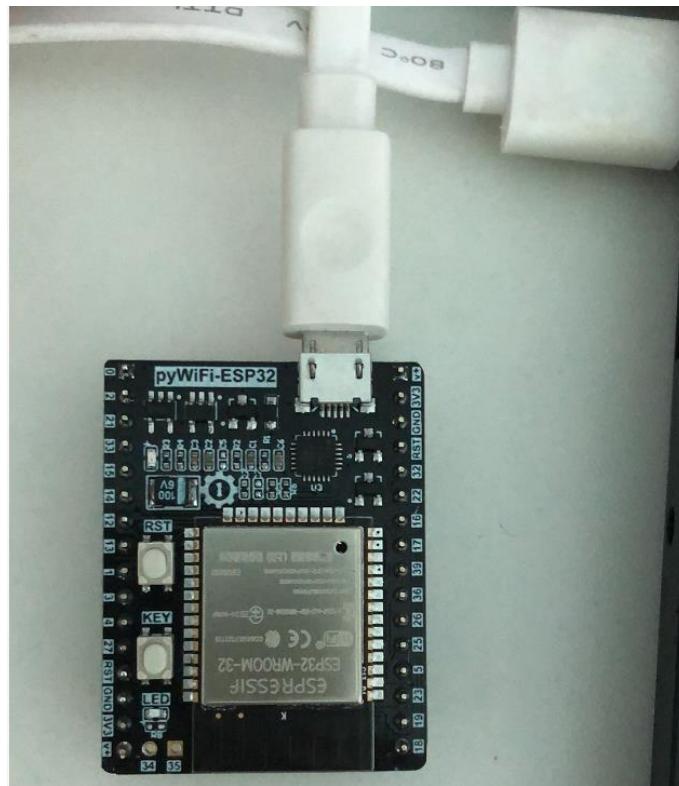


图 3-3 通过 MicroUSB 线连接到电脑

如果你的操作系统是 Win10,一般情况下能自动安装。鼠标右键点击“我的电脑”—属性—设备管理器： 出现串口号说明安装成功，如下图所示。

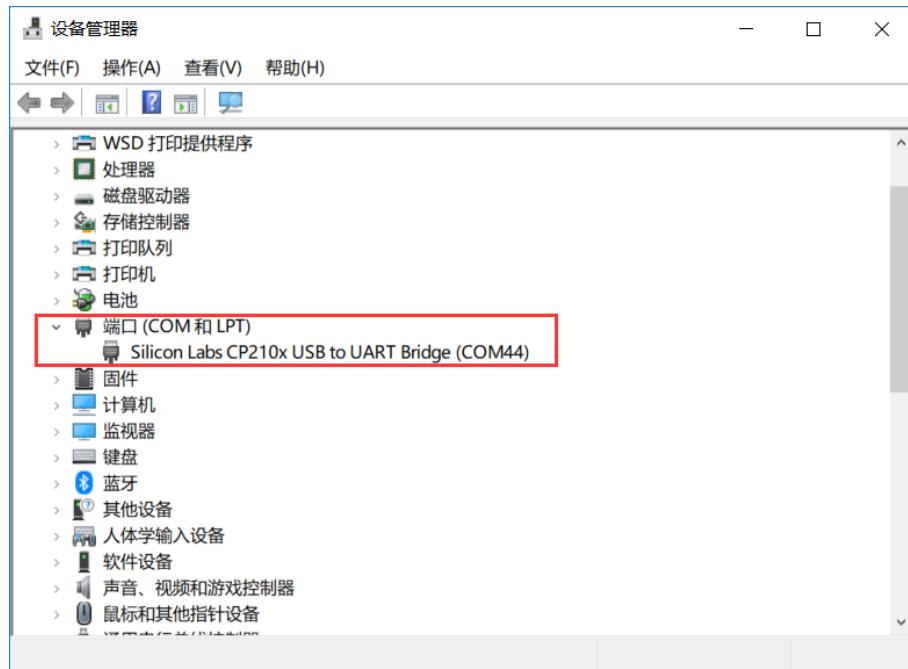


图 3-4 串口驱动成功安装

如果无法安装，请手动安装驱动，方法如下：

不能自动安装时候，设备会出现黄色叹号，这时候点击设备右键，选择“更新驱动程序”，选择“浏览计算机查找驱动”：

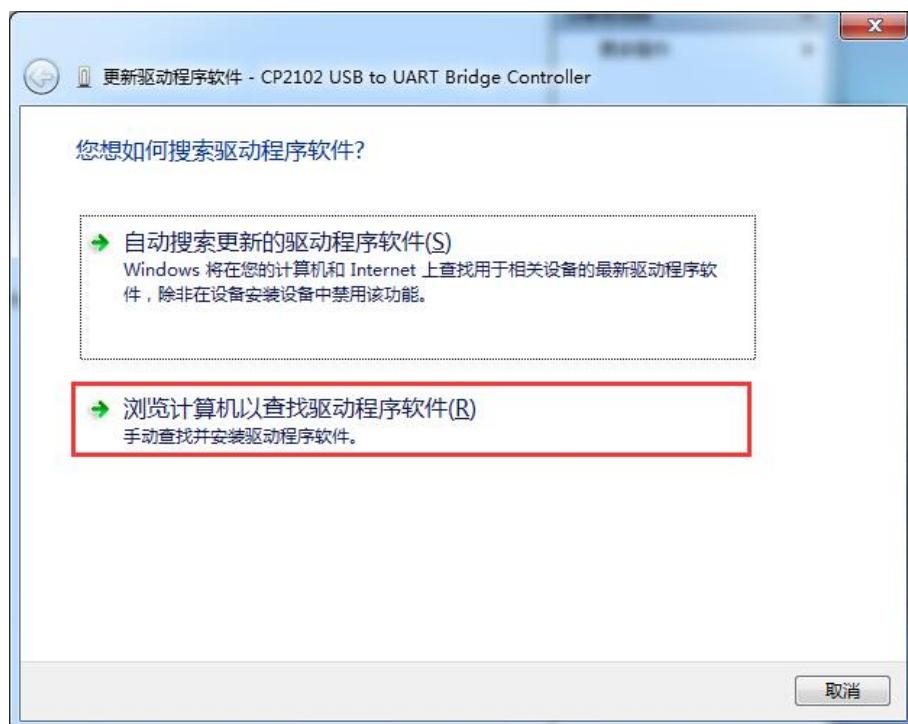


图 3-5

驱动路径选择：零一科技（01Studio）MicroPython 开发套件配套资料\01-开发工具\01-Windows\串口终端工具\CP210x 驱动，点击确认后即可自动安装：

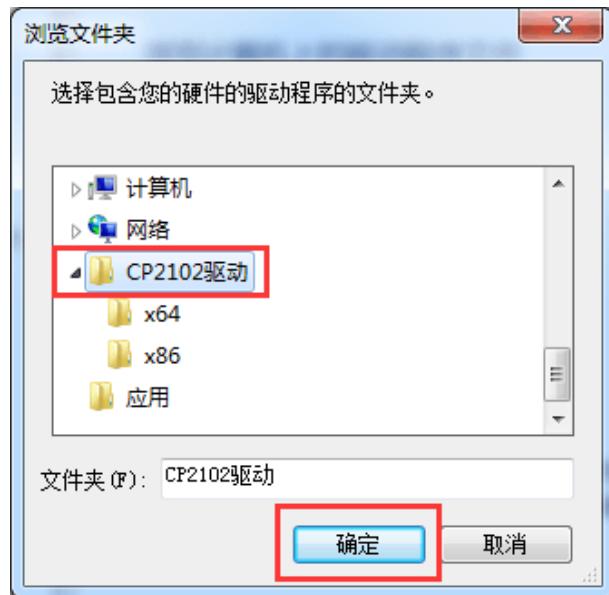


图 3-6

安装成功后如下图：

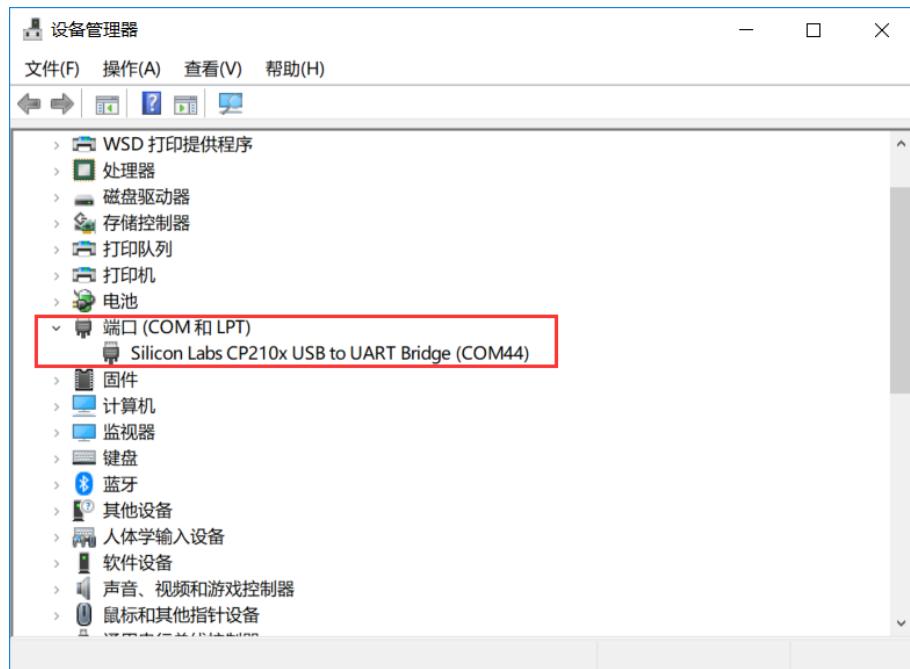


图 3-7 串口驱动安装成功

3.1.2.2 REPL 串口交互调试

pyWiFi-ESP32 的 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。

我们打开 Thonny，将开发板连接到电脑。点击右下角：

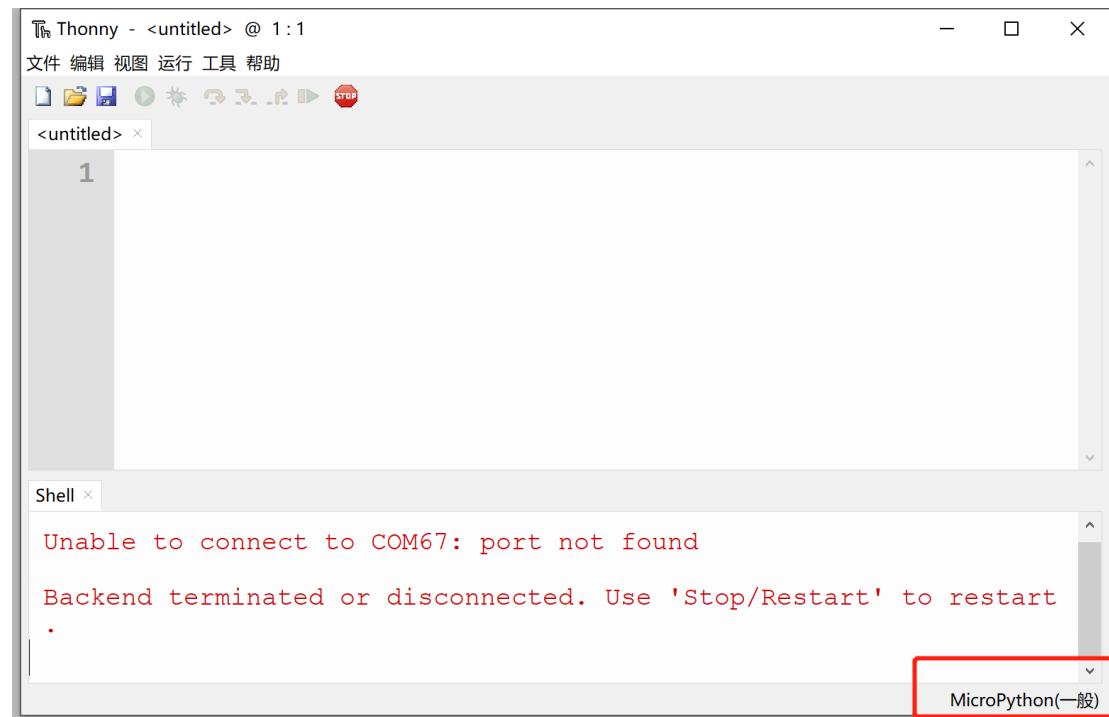


图 3-8 选择要连接的设备

在弹出的列表选择：Configure interpreter

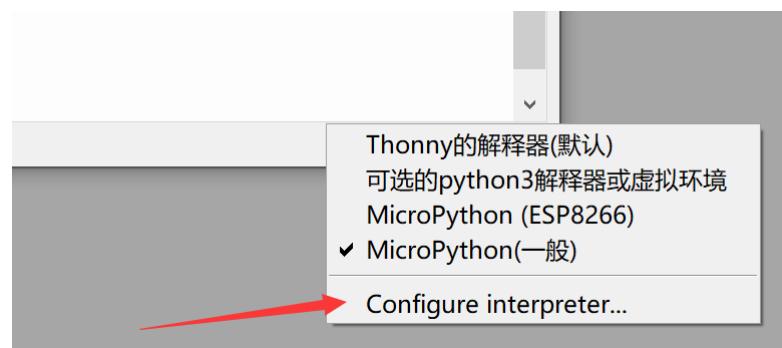


图 3-9

选择“MicroPython（ESP32）”和开发板对应的串口号，点击确认。

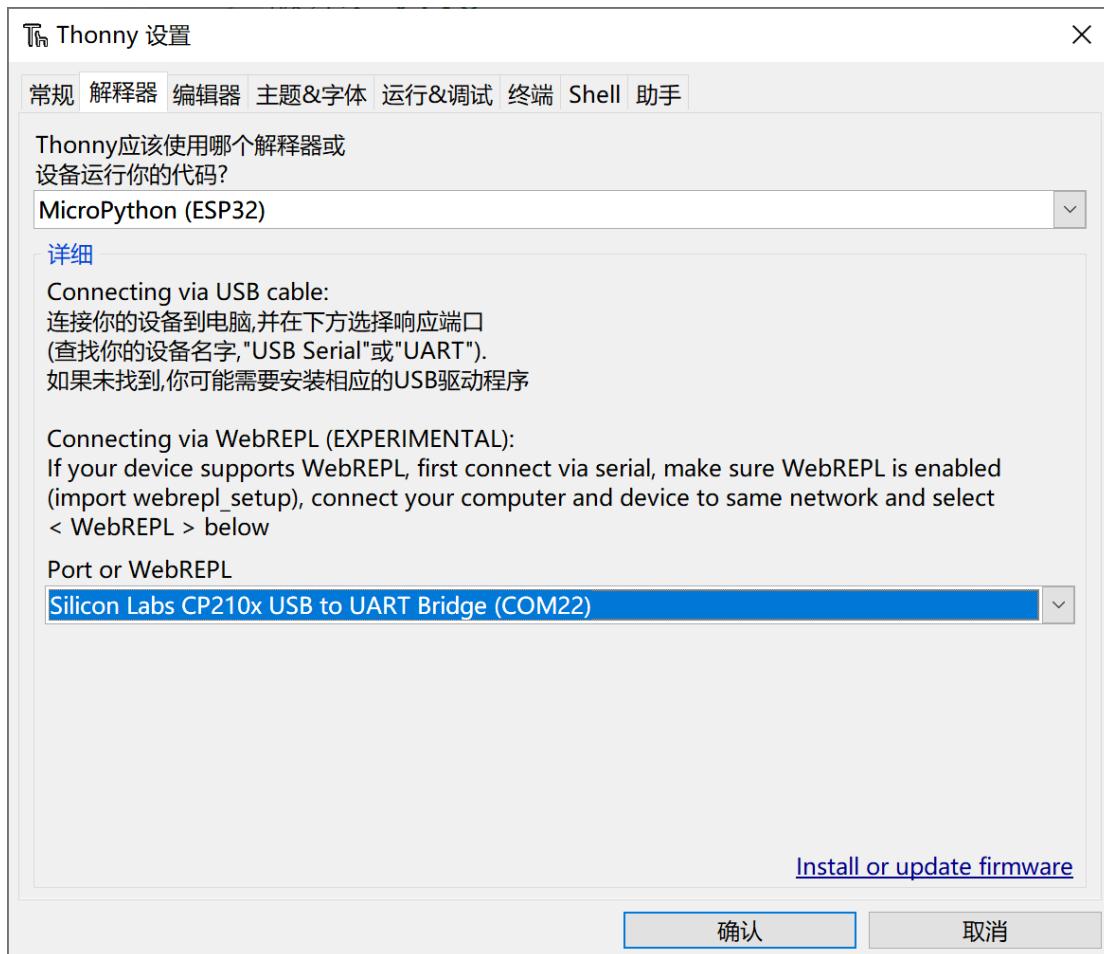


图 3-10 选择开发板类型

连接成功后可以在 shell（串口终端）看到固件的相关信息：

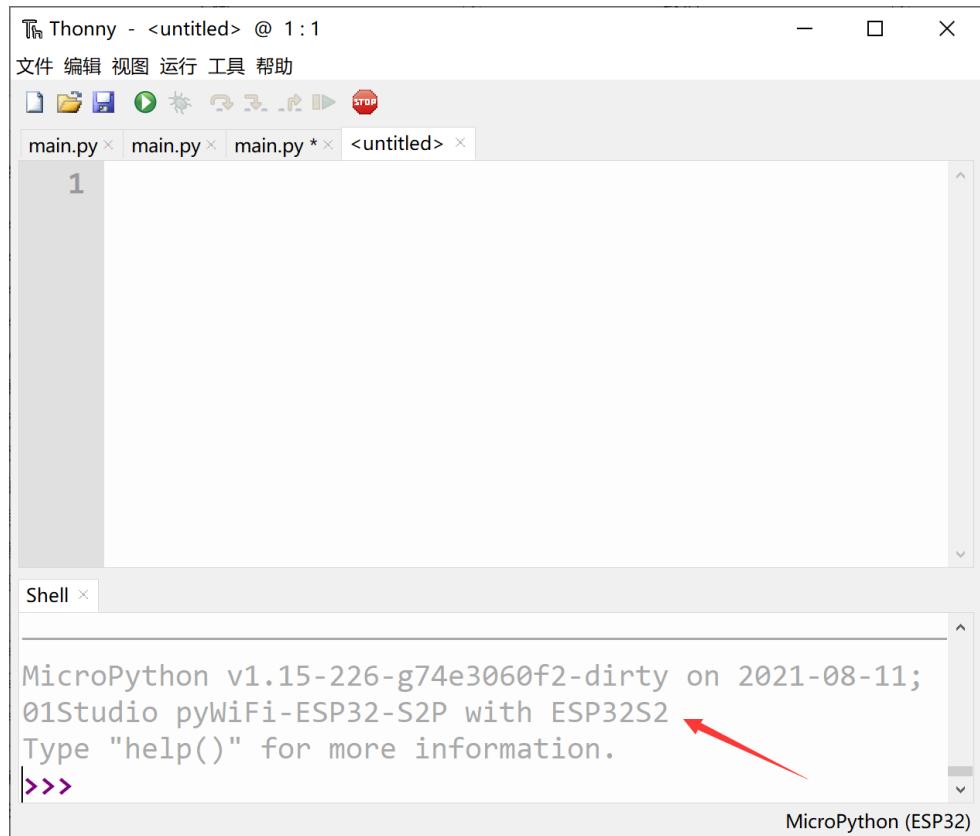


图 3-11 连接成功

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：



图 3-12

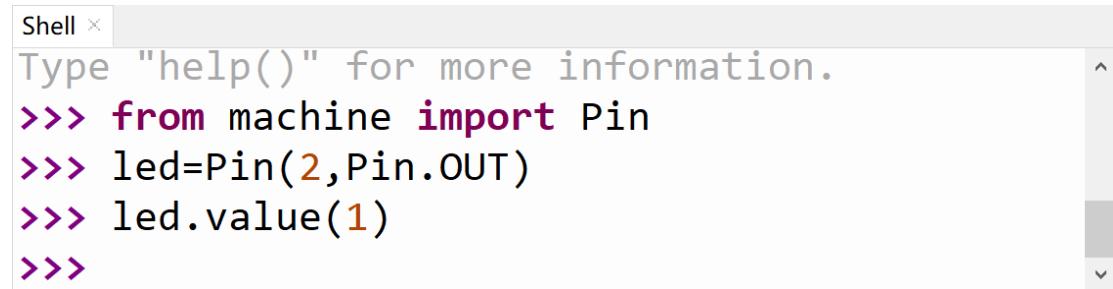
再输入 `1+1`，按回车：



图 3-13

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 machine import Pin  
  
LED = Pin(2, Pin.OUT)  
  
LED.value(1)
```



The screenshot shows a terminal window titled "Shell". It displays the following text:
Type "help()" for more information.
>>> from machine import Pin
>>> led=Pin(2,Pin.OUT)
>>> led.value(1)
>>>

图 3-14 逐行输入



图 3-15 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，在后面代码运行时候，如果程序出错，出错信息将通过 REPL 打印。

The screenshot shows a MicroPython development environment. At the top is a menu bar with '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 like Open, Save, and Run. The main area has two tabs: 'main.py' and 'Shell'. The 'main.py' tab contains Python code for a project, including comments and imports. The 'Shell' tab shows the REPL interface where a command is run, resulting in a traceback for an AttributeError. A red arrow points to the error message in the shell output.

```
main.py
1 ...
2 实验名称: 点亮板载LED灯
3 版本: v1.0
4 日期: 2021-1
5 作者: 01Studio
6 社区: www.01studio.org
7 ...
8
9 #导入Pin模块
10 from machine import Pin
11
12 LED = Pin(25, Pin.OUT) #构建LED对象
13 LED.valu(1) #点亮LED,高电平点亮

Shell <input>
>>> %Run -c $EDITOR_CONTENT
Traceback (most recent call last):
  File "<stdin>", line 13, in <module>
AttributeError: 'Pin' object has no attribute 'valu'
>>>

MicroPython (Raspberry Pi Pico)
```

图 3-16 错误打印

REPL 终端常用键盘按键:

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。

3.1.2.3 文件系统

pyWiFi-ESP32 里面内置了文件系统，可以简单理解成上电后运行的 python 文件，这个可以通过 Thonny 非常方便地读写。

点击 视图--文件：

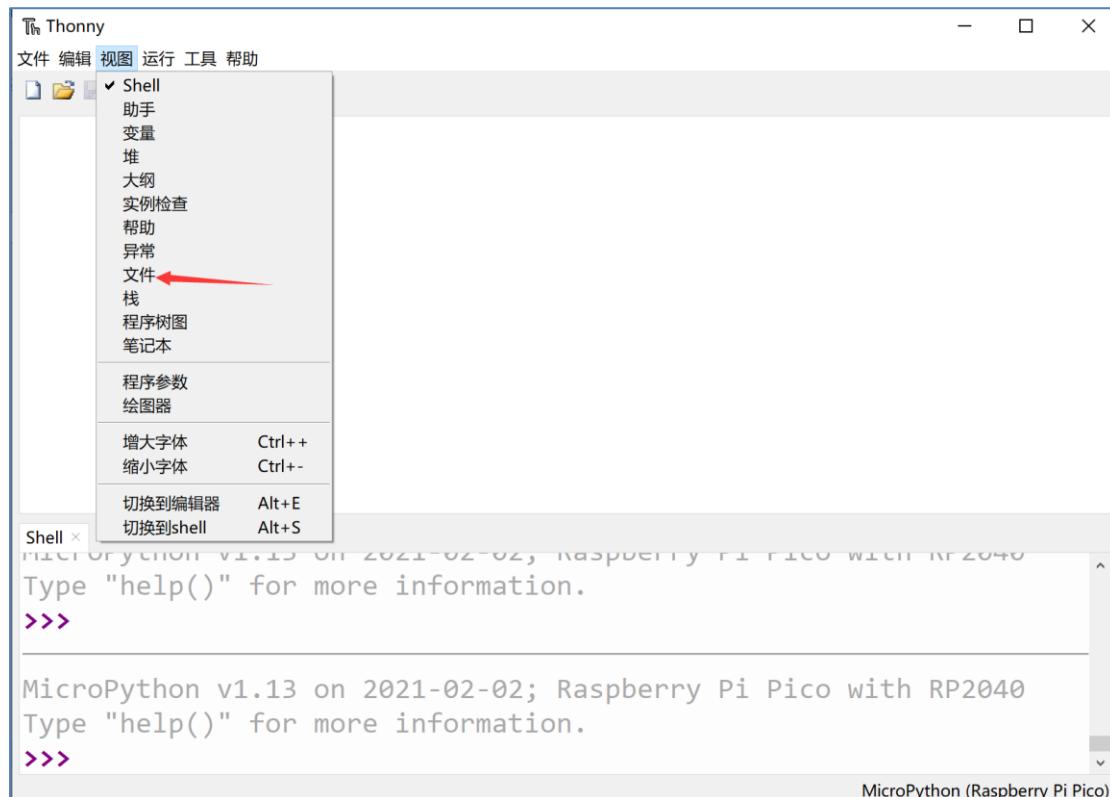


图 3-17

可以看到左边出现本地和开发板的实时文件浏览窗口：



图 3-18

在本地文件点击右键—上传到即可将相关文件发送到开发板，也可以将开发板上的文件发送到本地，非常方便。

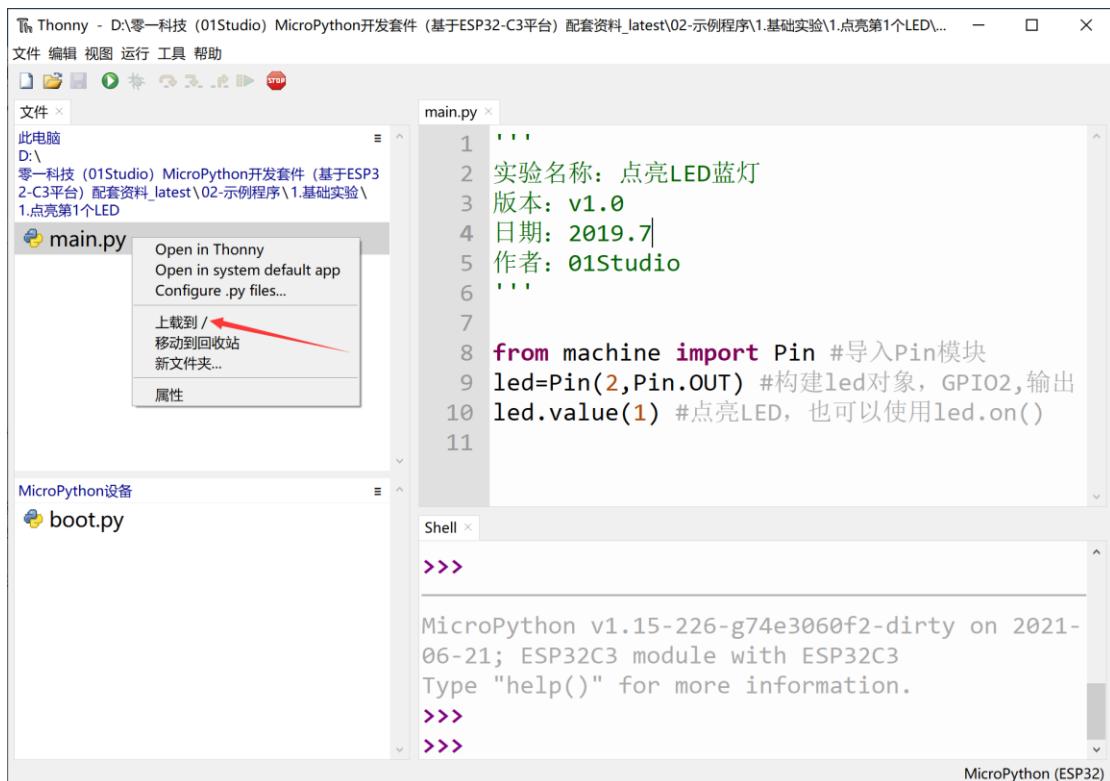


图 3-19 发送文件

3.1.2.4 代码测试

前面我们已经安装好了 Thonny IDE 和配置，接下来我们使用最简单的方式来做一个点亮 LED 蓝灯的实验，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：

连接开发板，在 thonny 左上角本地文件区域找到 零一科技（01Studio）MicroPython 开发套件（基于 ESP32 平台）配套资料\02-示例程序\1.基础实验\1.点亮第 1 个 LED 下的 main.py 文件，双击打开后看到右边编程区出现相关代码。

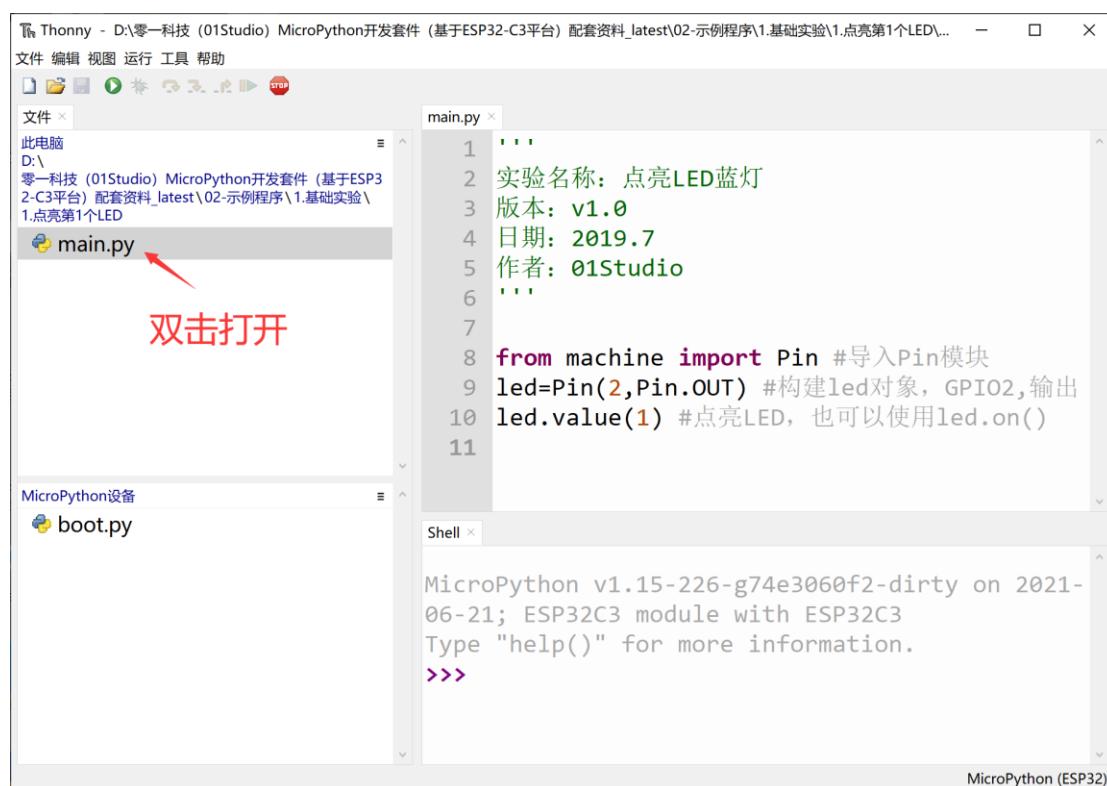


图 3-20

点击 运行—运行当前脚本 或者直接点绿色按钮：

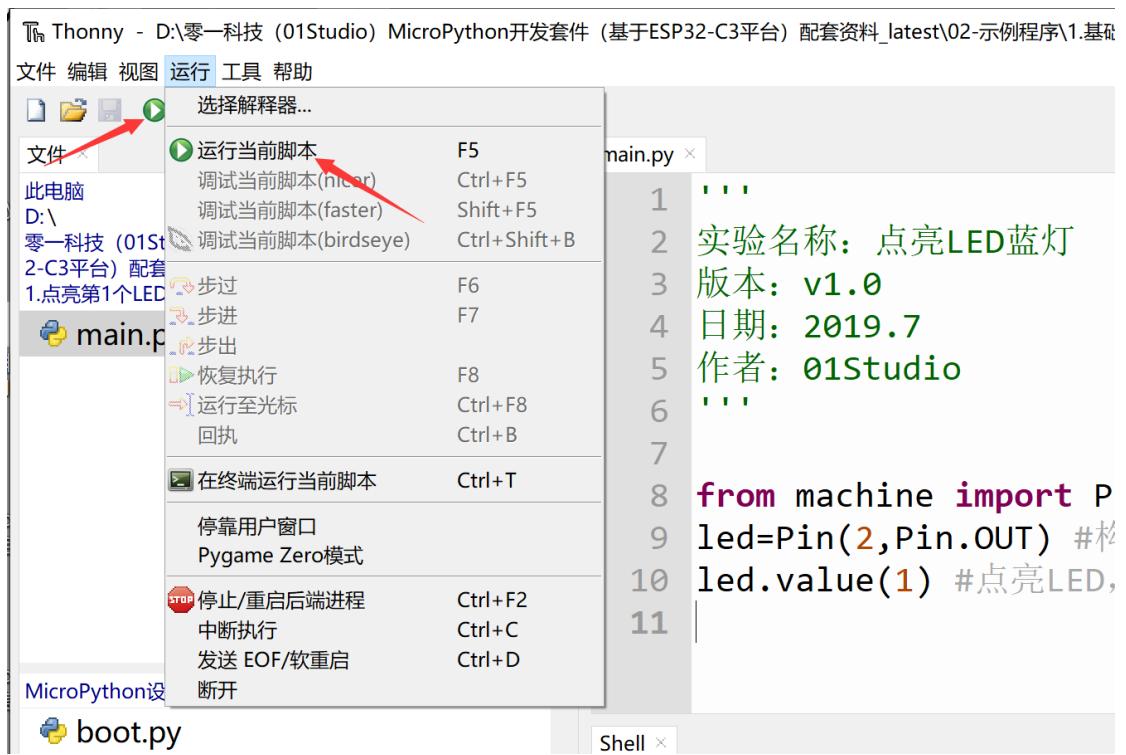


图 3-21 运行例程

这时候可以看到开发板上的蓝灯被点亮：



图 3-22 蓝灯被点亮

运行功能代码是保存在开发板的 RAM（内存）里面，断电后丢失，那么如何实现开发板上电运行我们的代码呢？方法如下：

Micropython 上电默认先运行名字为 `boot.py` 文件，然后在运行 `main.py` 文件，如果没有 `boot.py` 那么直接运行 `main.py`。

boot.py: 一般用于配置初始化参数；

main.py: 主程序

也就是我们只需要将代码以 `main.py` 文件发送到开发板，那么开发板就可以实现上电运行相关程序。

我们将 LED 例程的 `main.py` 发送到开发板

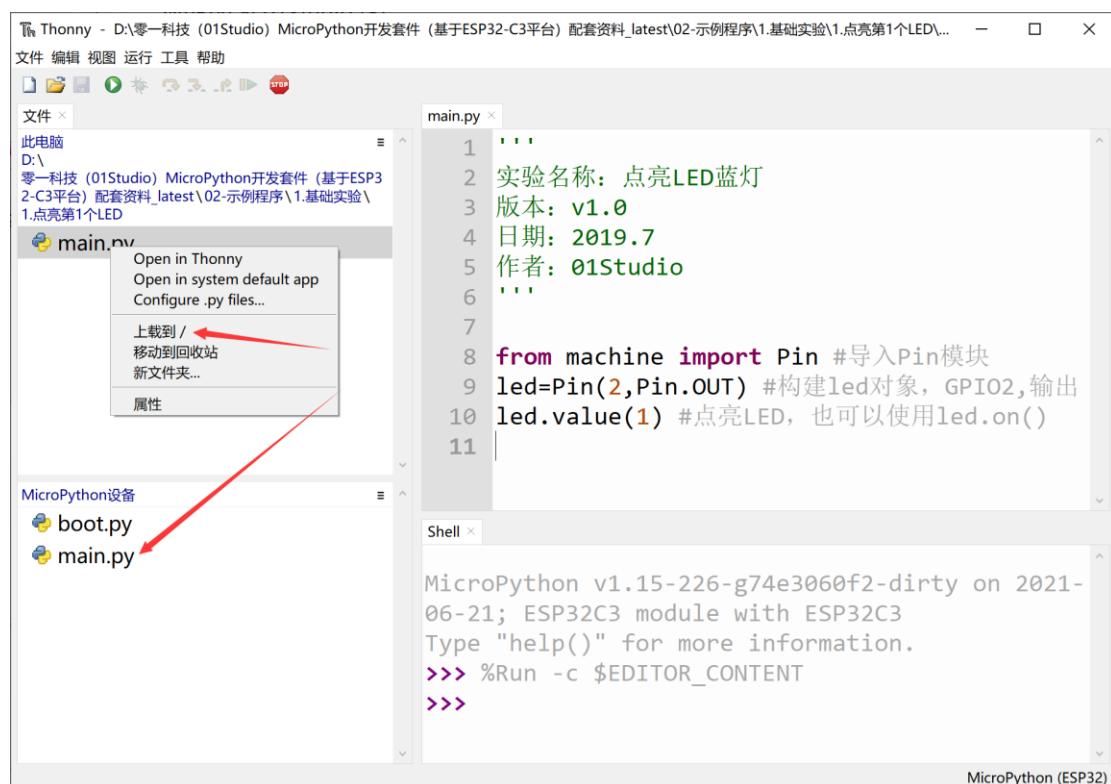


图 3-23 将 LED 例程的 `main.py` 发送到开发板

按下开发板的复位键，可以看到 LED 蓝灯被点亮：



图 3-24 代码离线运行

3.1.2.5 固件更新

我们的开发板出厂已经烧录好了固件，固件更新是指重新烧写开发板的出厂文件或者是升级的固件，使用上海乐鑫提供的官方软件烧录：

找到路径：[零一科技（01Studio）MicroPython 开发套件配套资料\01-开发工具\01-Windows\固件更新工具\flash_download_tools_v3.8.8](#) 下的 flash_download_tools_v3.8.8.exe 软件，双击打开。

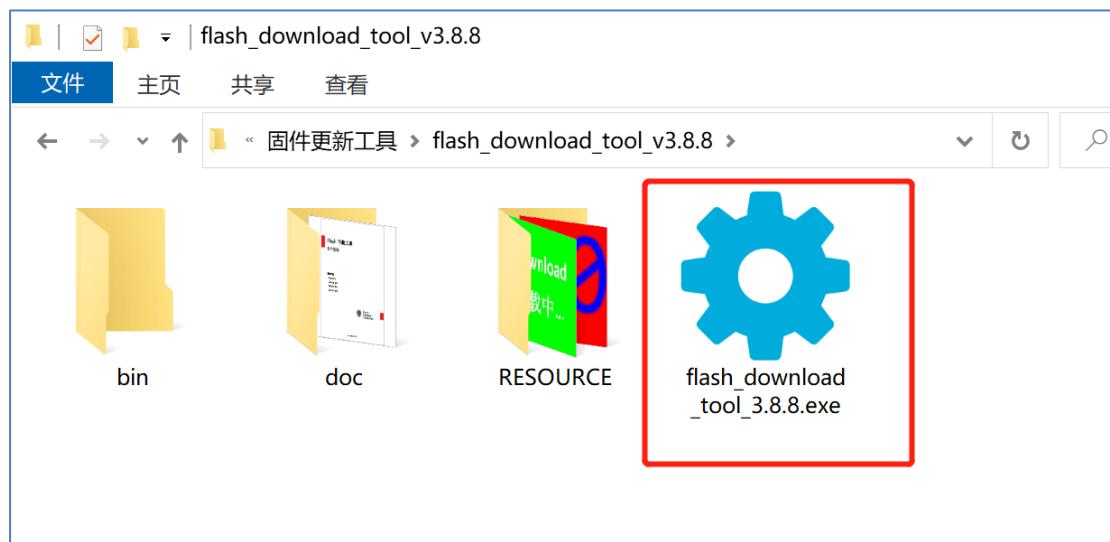


图 3-25

芯片这里选择 ESP32，develop 开发者模式，然后点击 OK：

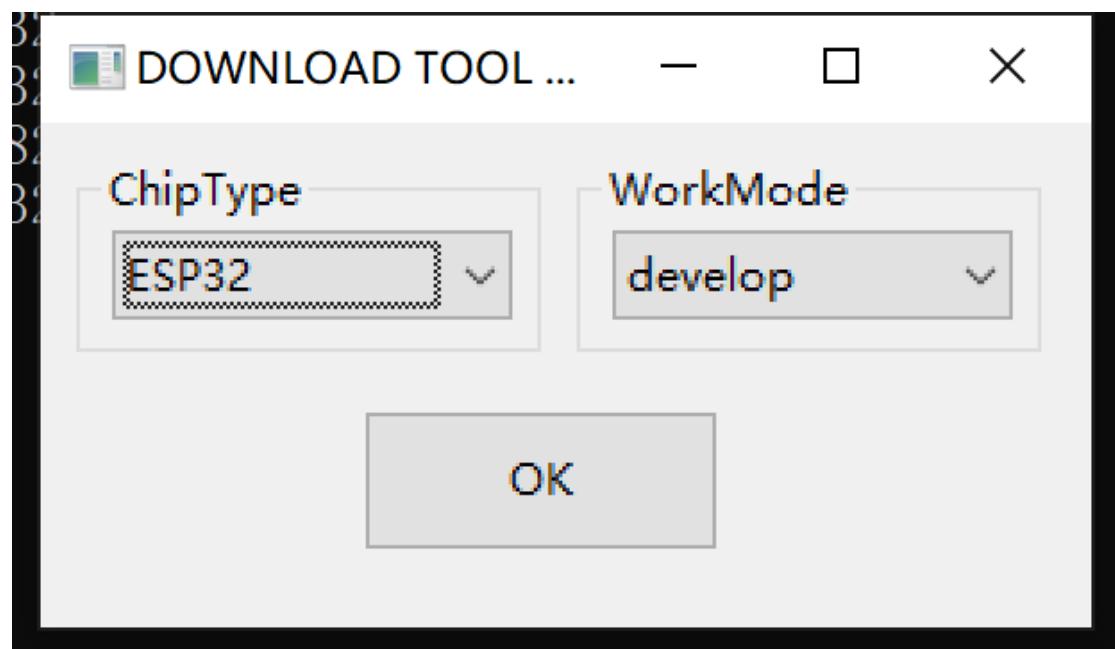


图 3-26 选择 ESP32

选择 SPIDownload，在下图箭头位置点击，选择要烧录固件。固件位置位于：
零一科技（01Studio）MicroPython 开发套件配套资料\03-相关固件目录下。

其它配置选项也请参考下图，注意下载地址是 **0x1000**。（COM 串口是选择自己的串口，在设备管理器查询。）

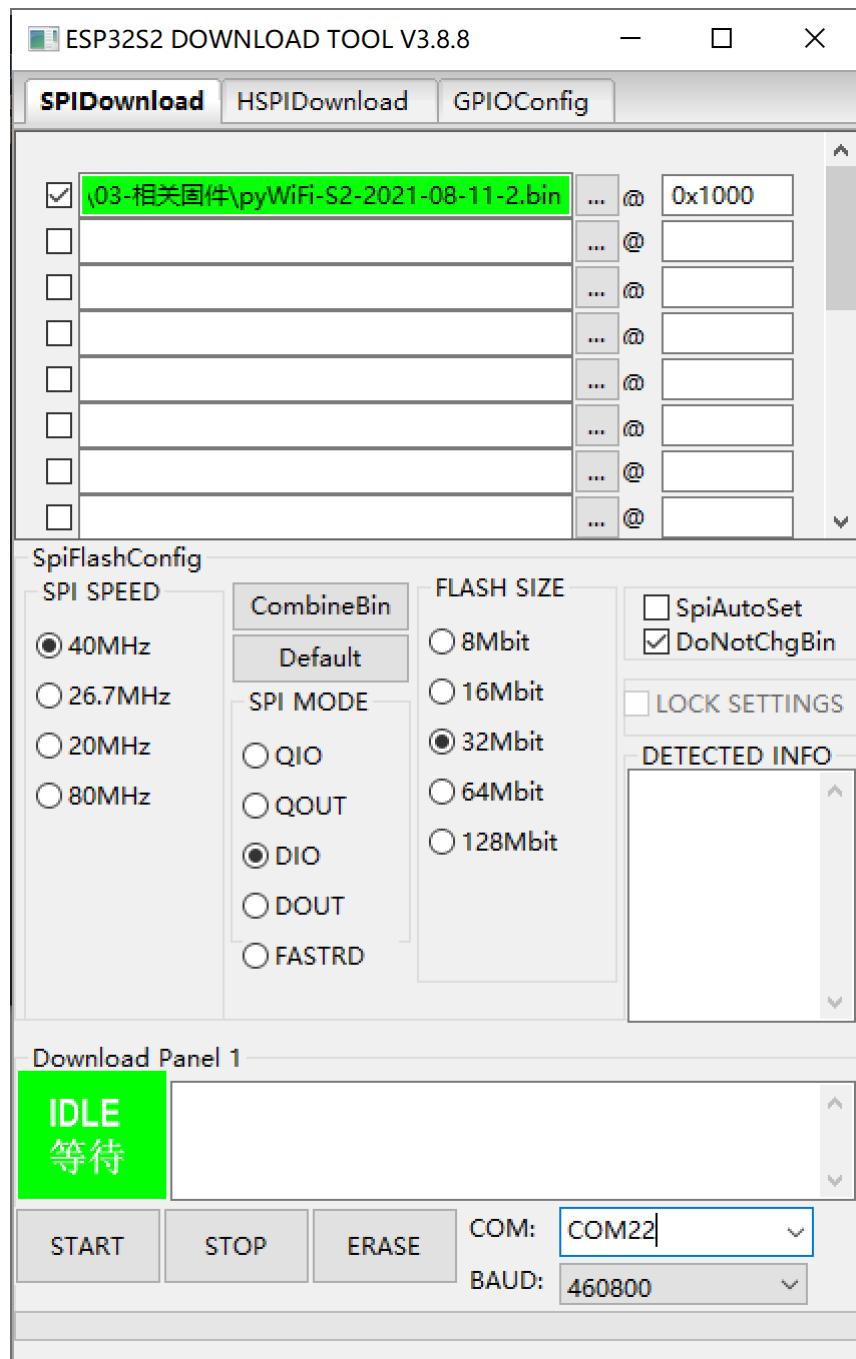


图 3-27 配置

配置好后，先点击“ERASE”按钮刷除模块里面内容。点击软件下方“ERASE”按钮，刷除成功后，左边绿色框出现完成字样。

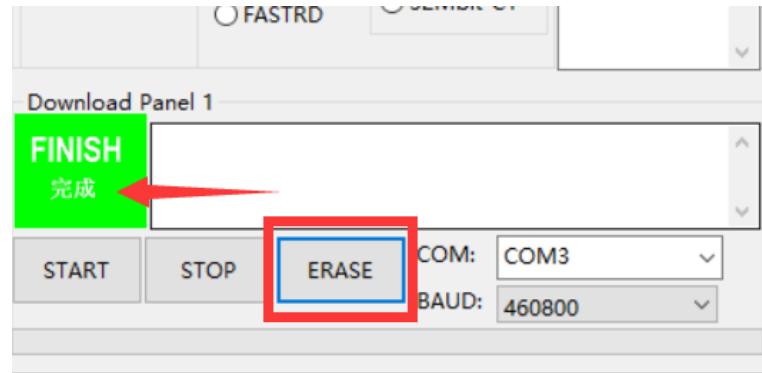


图 3-28 刷除 flash 成功

刷除成功后，点击“START”按钮开始烧录，烧录完成有左边绿色框出现“完成”字样。完成后记得点“stop”按钮或者关闭软件释放串口。

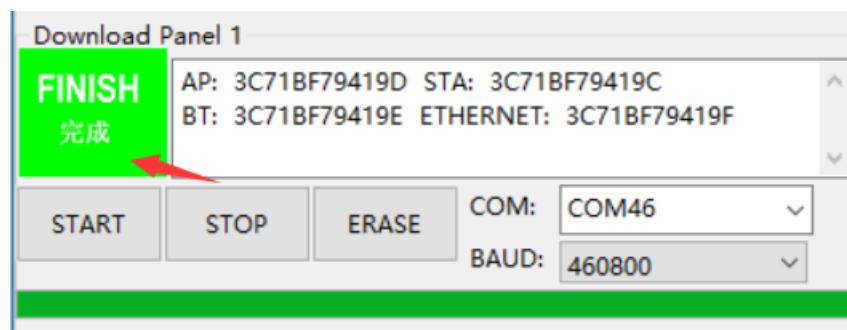


图 3-29 烧录成功

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，力求最佳的用户体验；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：

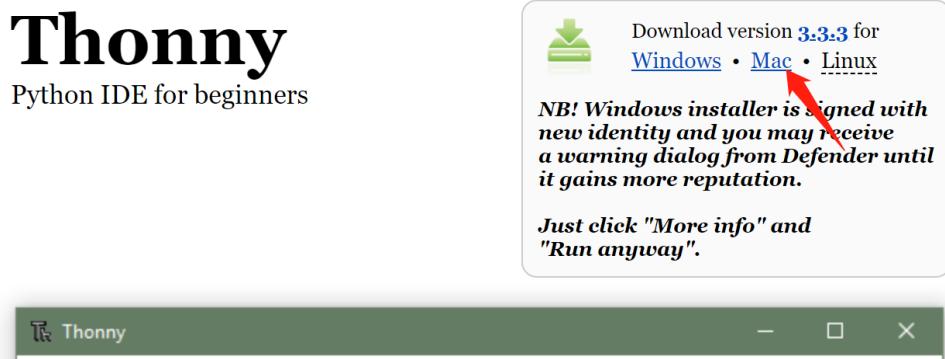


图 3-30

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](#) 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。

3.2.2.1 固件更新

01Studio 的开发板出厂已经烧录好了固件，固件更新是指重新烧写开发板的出厂文件或者是升级的固件，Mac OS 下没有跟 Windows 一样提供可视化 Flash 烧写软件，因此需要通过 esptool.py 工具结合终端命令来烧写固件。

打开终端，输入以下命令安装 esptool：

```
pip install esptool
```

安装完成后即可使用 esptool 工具来烧写固件，步骤如下：

第一步：刷除 Flash：

在终端输入下面命令按回车。

```
esptool.py --port /dev/tty.SLAB_USBtoUART erase_flash
```

刷除成功后出现以下信息

```
MacBook-Air:~ jackey$ esptool.py --port /dev/tty.SLAB_USBtoUART erase_flash
esptool.py v2.6
Serial port /dev/tty.SLAB_USBtoUART
Connecting.....
Detecting chip type... ESP8266
Chip is ESP8266EX
Features: WiFi
MAC: b4:e6:2d:52:91:a6
Uploading stub...
Running stub...
Stub running...
Erasing flash (this may take a while)...
Chip erase completed successfully in 2.5s
```

图 3-31 刷除成功提示

第二步：烧写固件：

先将固件放到 MacBook 桌面，终端需要通过 “cd Desktop/” 命令进入到桌面位置（目的是让终端当前位置和固件在同一路径下）：

在终端输入下面命令按回车，esp32-20180511-v1.9.4.bin 是固件名称，可以根据自己的实际情况修改，输入时候可以通过 tab 键补全（需要注意的是 ESP32 下载固件指令和 ESP8266 区别如下：（ESP32 固件从 0x1000 地址开始烧录。）

```
esptool.py --chip esp32 --port /dev/tty.SLAB_USBtoUART write_flash -z  
0x1000 esp32-20180511-v1.9.4.bin
```

烧录成功如下图：

```
MacBook-Air:Desktop jackey$ esptool.py --chip esp32 --port /dev/tty.SLAB_USBtoUART write_flash -z 0x1000 esp32-20190529-v1.11.bin  
esptool.py v2.6  
Serial port /dev/tty.SLAB_USBtoUART  
Connecting.....  
Chip is ESP32D0WDQ6 (revision 1)  
Features: WiFi, BT, Dual Core, 240MHz, VRef calibration in efuse, Coding Scheme None  
MAC: a4:cf:12:5f:11:54  
Uploading stub...  
Running stub...  
Stub running...  
Configuring flash size...  
Auto-detected Flash size: 4MB  
Compressed 1146864 bytes to 717504...  
Wrote 1146864 bytes (717504 compressed) at 0x00001000 in 63.3 seconds (effective 144.9 kbit/s)...  
Hash of data verified.
```

图 3-32 固件烧录成功

另外就是 ESP32 固件没有固件检测功能，因此 `esp.check_fw()` 指令无效。

3.3 基于 Linux (树莓派)

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：

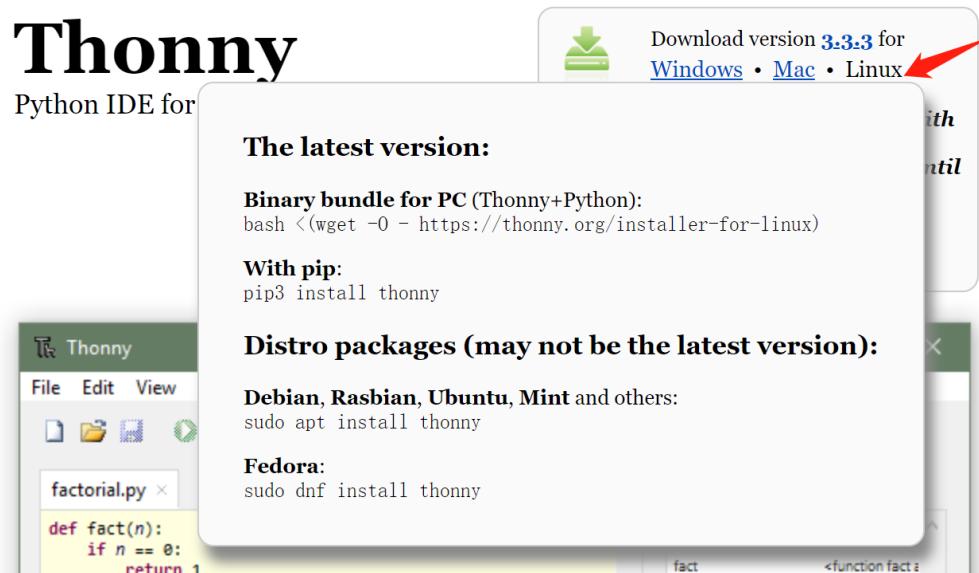


图 3-33

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，力求达到快速理解和学习的作用：

- (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

- **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:**

pyWiFi-ESP32。

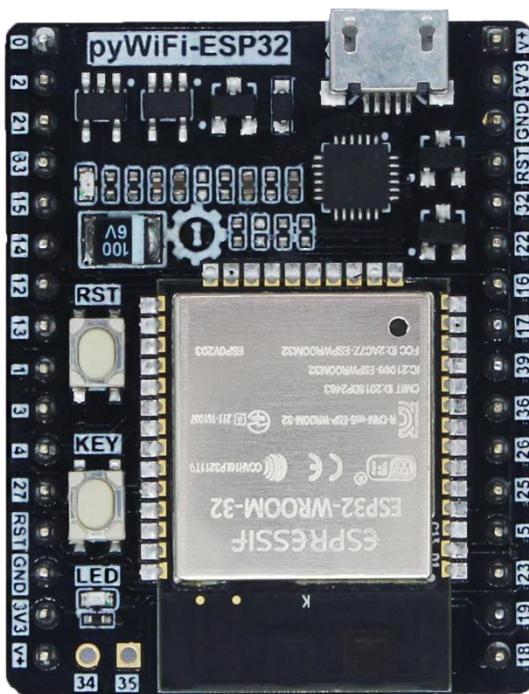


图 4-1 pyWiFi-ESP32

- **实验目的:**

点亮 LED（蓝灯）。

- **实验讲解:**

pyWiFi-ESP32 上有 1 个 LED（蓝色），控制 LED 使用 machine 中的 Pin 对象，其构造函数和使用方法如下：

构造函数
<code>led=machine.Pin(id,mode,pull)</code>
构建 led 对象。 id:引脚编号； mode:输入输出方式； pull:上下拉电阻配置。
使用方法
<code>led.value([x])</code>
引脚电平值。输出状态： x=0 表示低电平， x=1 表示高电平； 输入状态： 无须参数，返回当前引脚值。
<code>led.on()</code>
使引脚输出高电平 “1”。
<code>led.off()</code>
使引脚输出低电平 “0”。

表 4-1 Pin 对象

上表对 MicroPython 的 machine 中 Pin 对象做了详细的说明， machine 是大模块， Pin 是 machine 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 machine，然后通过 machine.Pin 来操作；

方式 2 是： from machine import Pin,意思是直接从 machine 中引入 Pin 模块，然后直接通过构建 led 对象来操作。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：

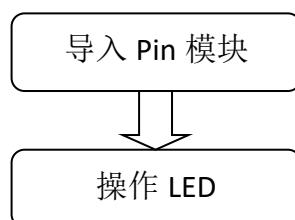


图 4-2 代码编写流程

LED 跟模块引脚 2 相连，通过输出高电平方式点亮。

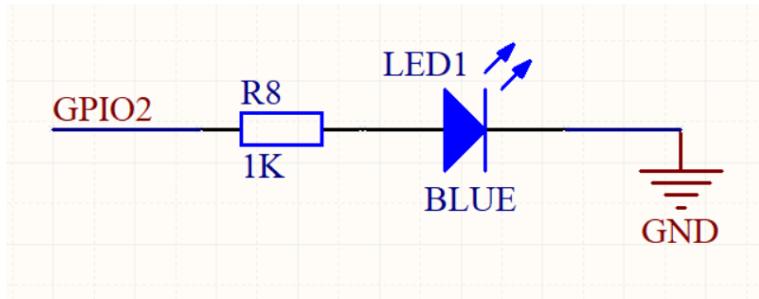


图 4-3 LED 接线图

参考代码如下：

```

...
实验名称: 点亮 LED 蓝灯
版本: v1.0
日期: 2019.7
作者: 01Studio
...
from machine import Pin #导入 Pin 模块
led=Pin(2,Pin.OUT) #构建 led 对象, GPIO2, 输出
led.value(1) #点亮 LED, 也可以使用 led.on()

```

运行程序有两个方法：

方法一：

编写好代码后点击 Mu 上方的“运行”按钮，可以直接观察到代码运行情况。这个方法不会将程序代码保存到 ESP32 模块的 flash 里面。这注意是方便调试使用。

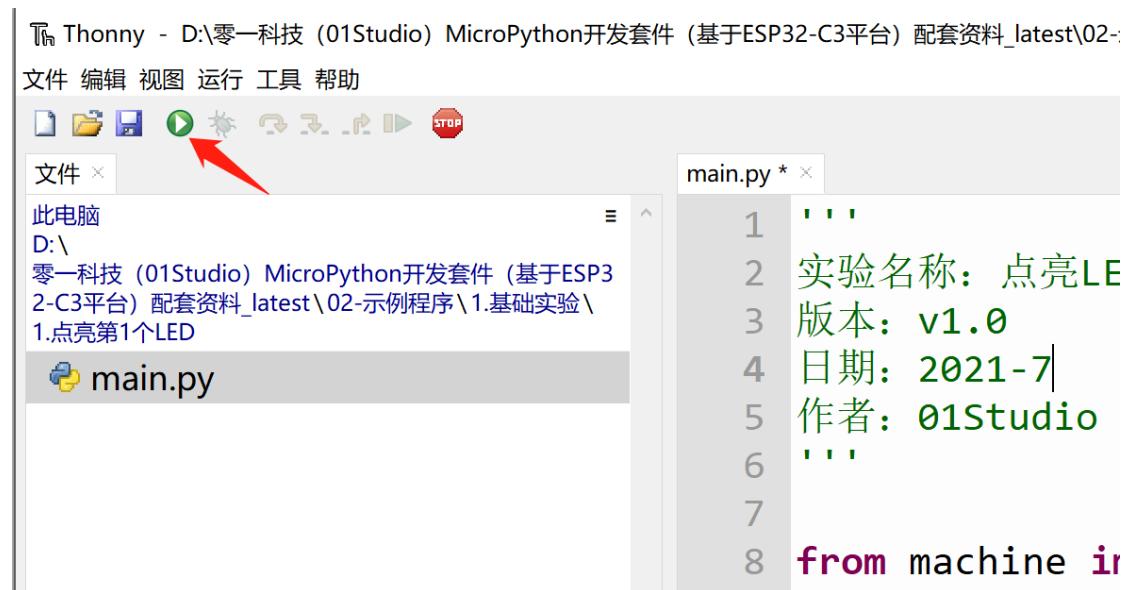


图 4-4 运行仿真

方法二：

将新建的文件保存名称为“main.py”的py文件，使用Mu的文件功能，从本地拖动到设备。然后按下复位按键，设备运行相关代码，这个方式相当于将程序烧录到设备flash，可以脱机使用。操作方法参考：[3.1.2.3 文件系统](#)章节内容。

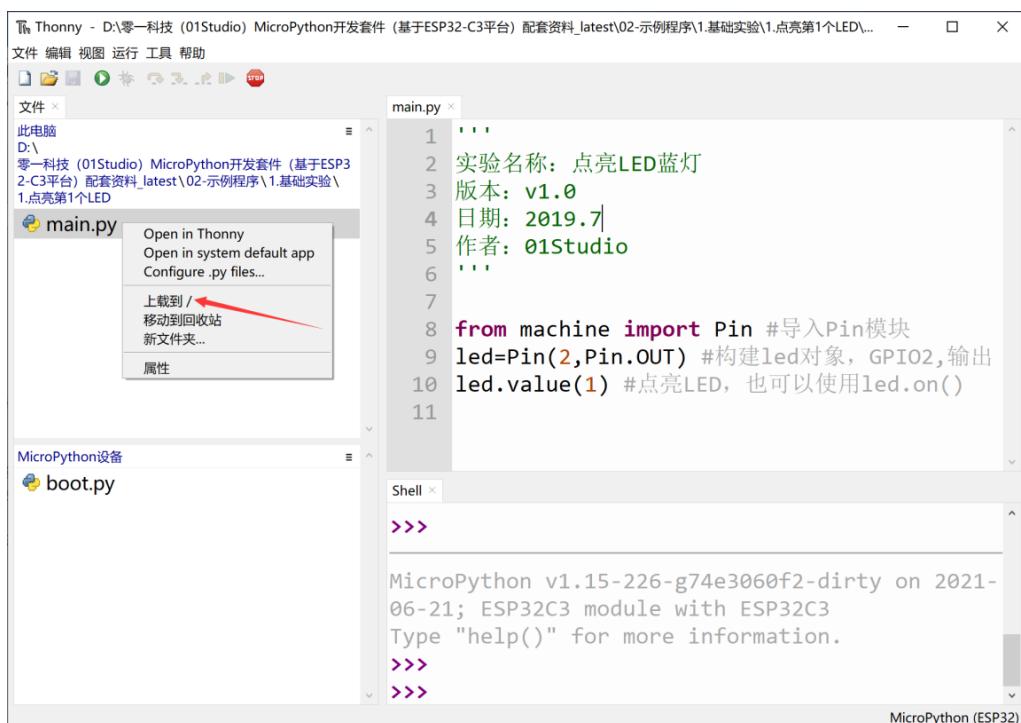


图 4-5 拷贝文件到设备

- **实验结果：**

下载程序，可以看到 LED（蓝灯）成功点亮。



图 4-6

- **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 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:

pyWiFi-ESP32。

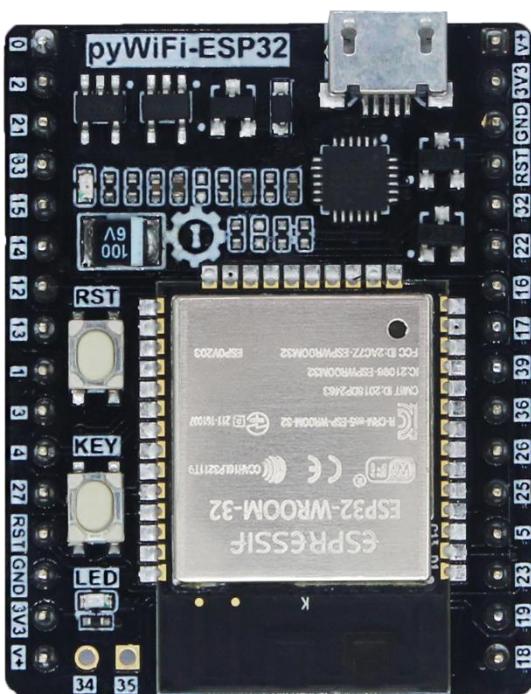


图 4-7 pyWiFi-ESP32

● 实验目的:

使用按键功能，通过检测按键被按下后，改变 LED（蓝灯）的亮灭状态。

● 实验讲解:

pyWiFi-ESP32 开发板上有 2 个按键，`RST` 和 `KEY`，`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 `KEY`。

让我们先来搞清楚 MicroPython 里面 Pin 模块实现按键的构造函数和使用方法。

构造函数
<code>KEY=machine.Pin(id, mode, pull)</code>
构建按键对象。id:引脚编号； mode:输入输出方式； pull:上下拉电阻配置。
使用方法
<code>KEY.value()</code>
引脚电平值。输入状态：无须参数，返回当前引脚值 0 或者 1。

表 4-2 KEY 对象

可以看到跟上一节 LED 一样，只是输入/输出状态的一个改变。从下面原理图可以看到，我们只需要在开发板上电后判断 KEY 引脚的电平，当被按下时候引脚为低电平“0”。

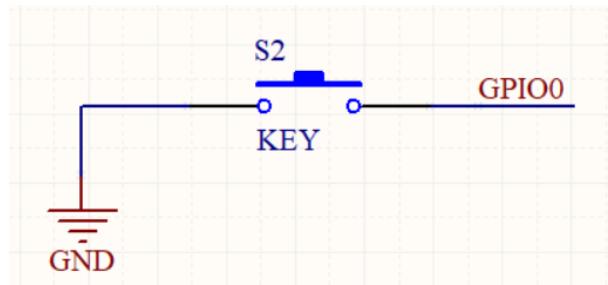


图 4-8 按键原理图

按键被按下时候可能会发生抖动，抖动如下图，有可能造成误判，因此我们需要使用延时函数来进行消抖：

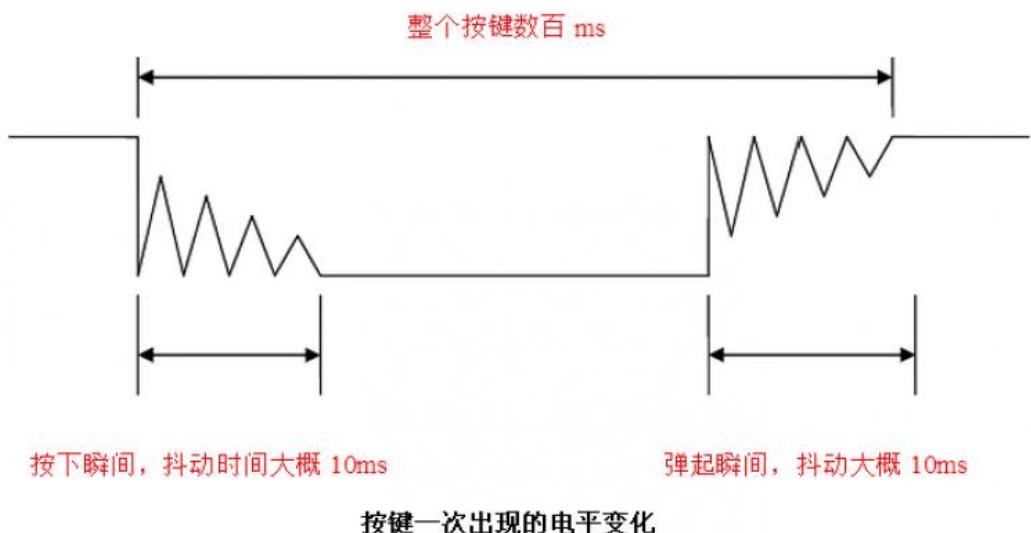


图 4-9 按键按下过程

常用的方法就是当检测按键值为 0 时，延时一段时间，大约 10ms，再判断按键引脚值仍然是 0，是的话说明按键被按下。延时使用 time 模块，使用方法如下：

```
import time

time.sleep(1)          # 睡眠 1 秒
time.sleep_ms(500)      # 睡眠 500 毫秒
time.sleep_us(10)       # 睡眠 10 微妙

start = time.ticks_ms() # 获取毫秒计时器开始值

delta = time.ticks_diff(time.ticks_ms(), start) # 计算从上电开始到当前时间的差值
```

编程流程图如下：

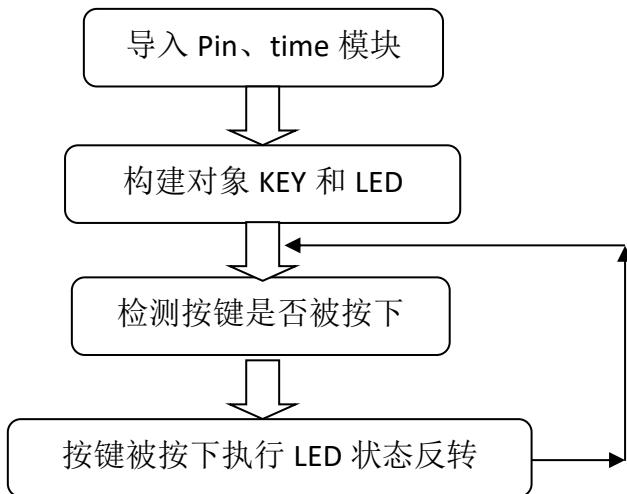


图 4-10 代码编写流程

实验参考代码如下：

```
'''  
实验名称: 按键  
版本: v1.0  
日期: 2019.8  
作者: 01Studio  
说明: 通过按键改变 LED 的亮灭状态  
'''  
  
from machine import Pin  
import time  
  
  
LED=Pin(2,Pin.OUT) #构建 LED 对象,开始熄灭  
KEY=Pin(0,Pin.IN,Pin.PULL_UP) #构建 KEY 对象  
state=0 #LED 引脚状态  
  
  
while True:  
    if KEY.value()==0: #按键被按下  
        time.sleep_ms(10) #消除抖动  
        if KEY.value()==0: #确认按键被按下  
            state=not state #使用 not 语句而非~语句  
            LED.value(state) #LED 状态翻转  
            while not KEY.value(): #检测按键是否松开  
                pass
```

从上面代码可以看到，初始化各个对象后，进入循环，当检测到 KEY 的值为 0（按键被按下）时候，先做了 10ms 的延时，再次判断；

state 为 LED 状态的值，每次按键按下后通过使用 not 来改变。这里注意的是在 python 里使用 ‘not’ 而不是 ‘~’ 的方式。not 返回的是 True 和 False，即 0,1。而~ 是取反操作，会导致出错。

- **实验结果：**

将以上代码编写到 main.py 文件，拷贝到设备。复位后即可运行程序。可以看到每当按键 KEY 被按下后，LED 的亮灭状态发生改变。如下图所示：



图 4-11

- **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。可以看到按键在 MicroPython 下开发也很简单。

4.3 外部中断

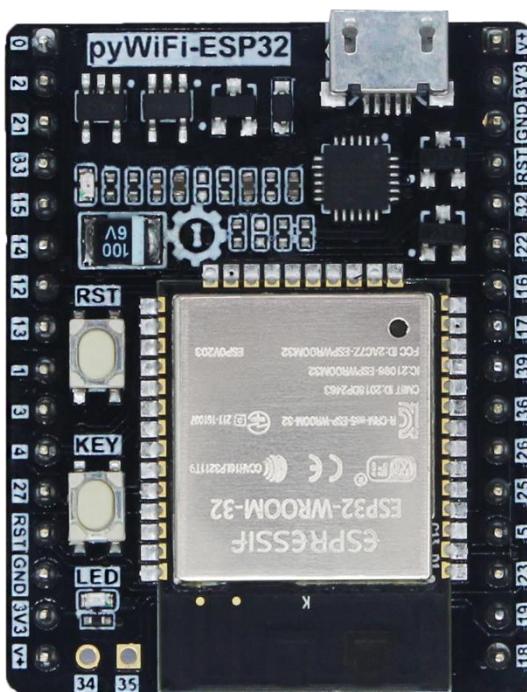
● 前言：

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 天才按下一次去执行相关功能，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。这大大节省了 CPU 的资源，因此中断的在实际项目的应用非常普遍。

● 实验平台：

pyWiFi-ESP32 开发板。



● 实验讲解：

外部中断也是通过 machine 模块的 Pin 子模块来配置，我们先来看看其配构造函数和使用方法：

构造函数
<pre>KEY=machine.Pin(id,mode,pull)</pre>
构建按键对象。id:引脚编号; mode:输入输出方式; pull:上下拉电阻配置。
使用方法
<pre>KEY.irq(handler,trigger)</pre>
配置中断模式。 handler:中断执行的回调函数; trigger: 触发中断的方式, 共 4 种, 分别是 Pin.IRQ_FALLING (下降沿触发)、 Pin.IRQ_RISING (上升沿触发)、Pin.IRQ_LOW_LEVEL (低电平触发)、 Pin.IRQ_HIGH_LEVEL (高电平触发)。

表 4-3 按键对象

上升沿和下降沿触发统称边沿触发。从上一节按键可以看到，按键被按下时一个引脚值从 1 到 0 变化的过程，边沿触发就是指这个过程。

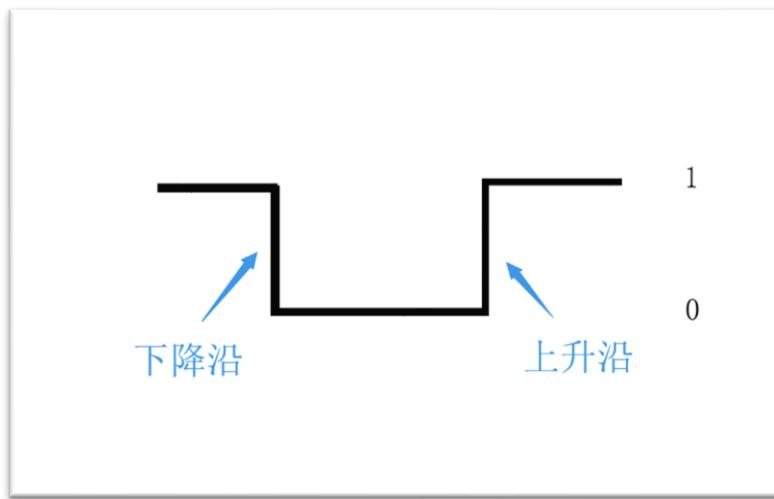


图 4-13 边沿触发

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 亮灭状态反转的代码即可。

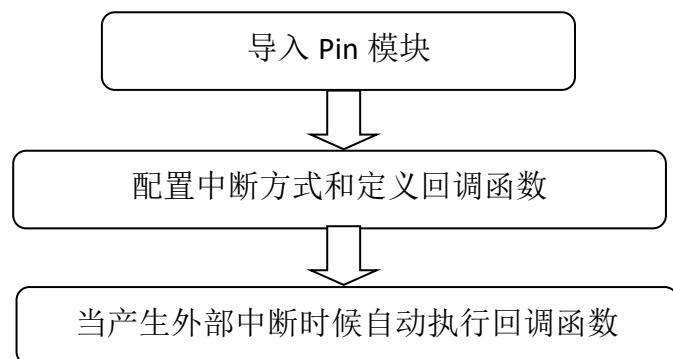


图 4-14 代码编写流程

参考程序代码如下：

实验名称：外部中断
版本：v1.0
日期：2019.8
作者：01Studio
说明：通过按键改变 LED 的亮灭状态（外部中断方式）
...


```
from machine import Pin

import time

LED=Pin(2,Pin.OUT) #构建 LED 对象,开始熄灭
KEY=Pin(0,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

    LED.value(state)

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

以上代码中需要注意的地方：

- 1、state 是全局变量，因此在 fun 函数里面用该变量必须添加 global state 代码，否则会在函数里面新建一个样的变量造成冲突。
- 2、在定义回调函数 fun 的时候，需要将 Pin 对象 KEY 传递进去。

● 实验结果：

将 main.py 文件拷贝到设备，复位。



图 4-15 实验现象

- **总结：**

从参考代码来看，只是用了几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，出来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。

4.4 定时器

● 前言：

定时器，顾名思义就是用来计时的，我们常常会设定计时或闹钟，然后时间到了就告诉我们要做什么了。单片机也是这样，通过定时器可以完成各种预设好的任务。

● 实验平台:

pyWiFi-ESP32。

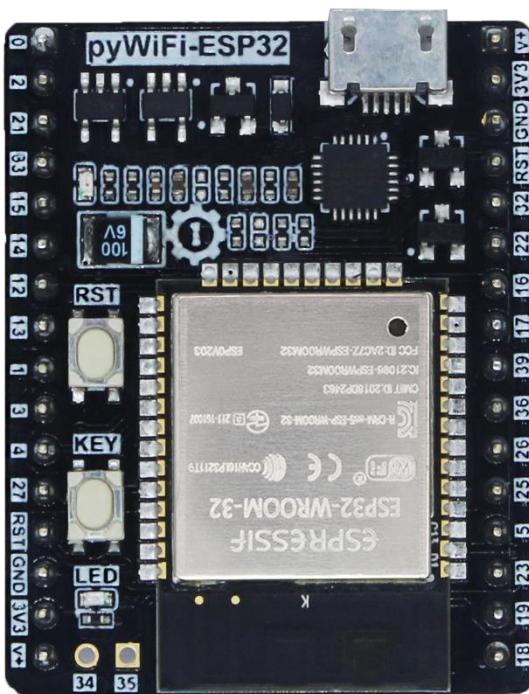


图 4-16 pyWiFi-ESP32

● 实验目的:

通过定时器让 LED 周期性每秒闪烁 1 次。

● 实验讲解:

ESP32 内置 RTOS（实时操作系统）定时器，在 machine 的 Timer 模块中。通过 MicroPython 可以轻松编程使用。我们也是只需要了解其构造对象函数和使用方法即可！

构造函数
<code>tim=machine.Timer(-1)</code>
构建定时器对象。RTOS 定时器编号为-1;
使用方法
<code>tim.init(period, mode, callback)</code>
定时器初始化。 period:单位为 ms; mode: 2 种工作模式, Timer.ONE_SHOT (执行一次)、Timer.PERIODIC (周期性); callback:定时器中断后的回调函数。

表 4-4 定时器对象

定时器到了预设指定时间后，也会产生中断，因此跟外部中断的编程方式类似，代码编程流程图如下：

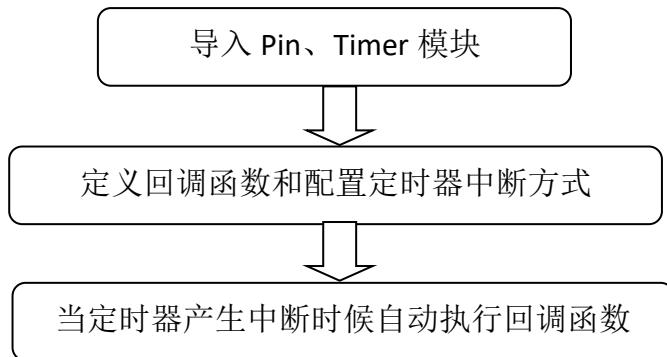


图 4-17 代码编写流程

参考代码如下：

```

...
实验名称: 定时器
版本: v1.0
日期: 2019.7
作者: 01Studio
说明: 通过定时器让 LED 周期性每秒闪烁 1 次
...
  
```

```
from machine import Pin,Timer

led=Pin(2,Pin.OUT)
Counter = 0
Fun_Num = 0

def fun(tim):
    global Counter
    Counter = Counter + 1
    print(Counter)
    led.value(Counter%2)

#开启 RTOS 定时器，编号为-1
tim = Timer(-1)
tim.init(period=1000, mode=Timer.PERIODIC,callback=fun) #周期为 1000ms
```

● 实验结果：



图 4-18 LED 以周期 1 秒的间隔闪烁

- **总结：**

本节实验介绍了 RTOS 定时器的使用方式，有用户可能会认为使用延时延时也可以实现这个功能，但相比于延时函数，定时器的好处就是不占用 CPU 资源。

4.5 I2C 总线（OLED 显示屏）

- **前言：**

前面学习了按键输入设备后，这一节我们来学习输出设备 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，pyWiFi-ESP32 是通过 I2C 总线与 OLED 显示屏通讯的。这章稍微复杂一点，但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：**

pyWiFi-ESP32 和 pyBase 开发底板。

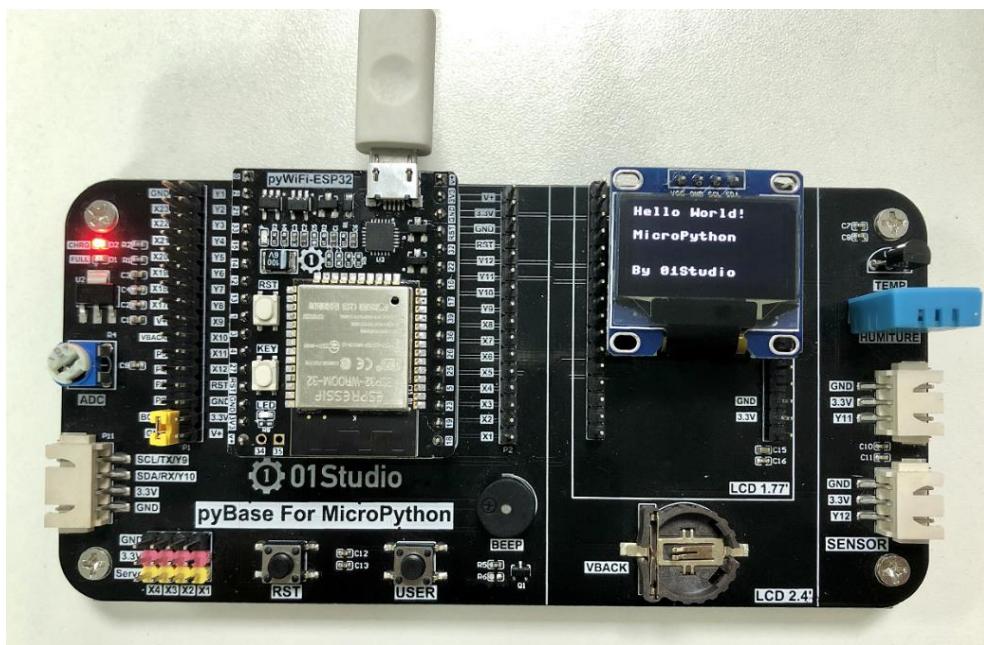


图 4-19 pyWiFi-ESP32 开发套件

- **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 4-20 I2C 接口的 OLED

在了解完 I2C 和 OLED 显示屏后，我们先来看看 pyBase 开发板的原理图，也就是上面的 OLED 接口是如何连线的。

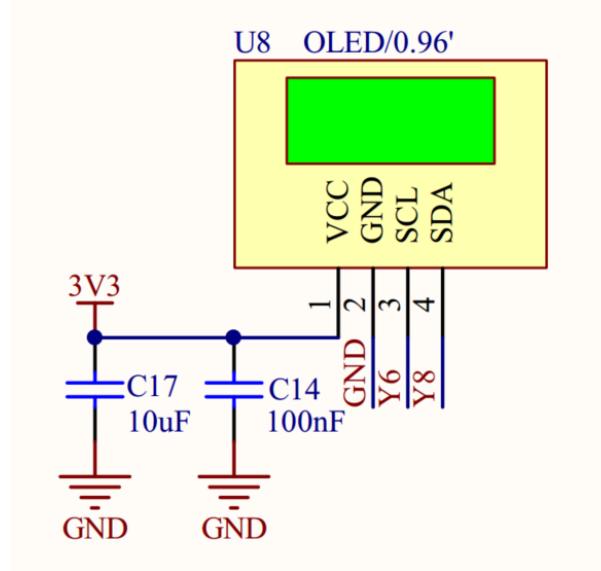


图 4-21 OLED 接口原理图

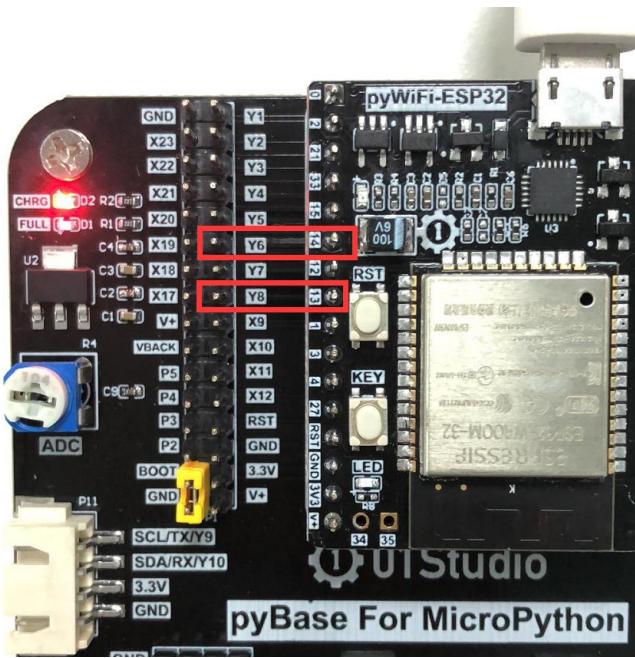


图 4-22 pyWiFi-ESP32 和 pyBase 的 OLED 连线图

从上图可见连接到 OLED 的别是 Y6→SCL 和 Y8→SDA。跟 pyWiFi-ESP32 的 14、13 引脚相连。本实验将使用 MicroPython 的 Machine 模块来定义 Pin 口和 I2C 初始化。具体如下：

构造函数
<code>i2c = machine.I2C(scl,sda)</code>
构建 I2C 对象。 <code>scl</code> :时钟引脚; <code>sda</code> :数据引脚。
使用方法
<code>i2c.scan()</code>
扫描 I2C 总线的设备。返回地址, 如: 0x3c;
<code>i2c.readfrom(addr,nbytes)</code>
从指定地址读数据。 <code>addr</code> :指定设备地址; <code>nbytes</code> :读取字节数;
<code>i2c.write(buf)</code>
写数据。 <code>buf</code> :数据内容;
*其它更多用法请阅读 MicroPython 文档:
中文文档链接: http://docs.micropython.01studio.org/

表 4-5 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 显示屏对象介绍如下：

构造函数
<code>oled = SSD1306_I2C(width, height, i2c, addr)</code>
构 OLED 显示屏对象。 <code>width</code> :屏幕宽像素； <code>height</code> : 屏幕高像素； <code>i2c</code> :定义好的 I2C 对象； <code>addr</code> :显示屏设备地址。
使用方法
<code>oled.text(string,x,y)</code>
将 <code>string</code> 字符写在指定为位置。 <code>string</code> : 字符； <code>x</code> :横坐标； <code>y</code> :纵坐标。
<code>oled.show()</code>
执行显示。
<code>oled.fill(RGB)</code>
清屏。 <code>RGB</code> : <code>0</code> 表示黑色， <code>1</code> 表示白色。

表 4-6 OLED 对象

学习了 I2C、OLED 对象用法后我们通过编程流程图来理顺一下思路：

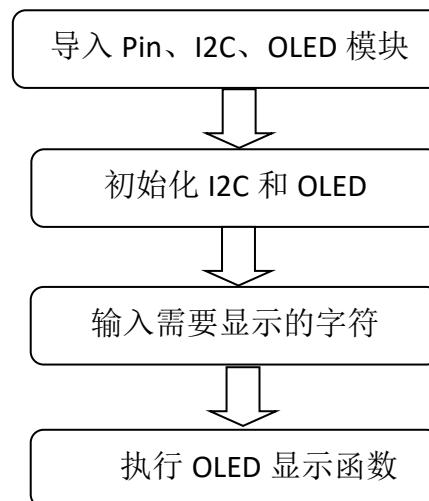


图 4-23 代码编写流程

main.py 文件参考代码如下：

```
'''  
实验名称: I2C 总线(OLED 显示屏)  
版本: v1.0  
日期: 2019.7  
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  
  
i2c = I2C(sda=Pin(13), scl=Pin(14)) #I2C 初始化: sda-->13, scl -->14  
  
#OLED 显示屏初始化: 128*64 分辨率,OLED 的 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 驱动文件拷贝到 pyWiFi-ESP32 的文件系统下，跟 main.py 保持同一个路径。

```

1  """
2  实验名称: I2C总线(OLED显示屏)
3  版本: v1.0
4  日期: 2019.7
5  作者: 01Studio
6  """
7
8
9  from machine import I2C,Pin      #从machine模块导入I2C、Pin子模块
10 from ssd1306 import SSD1306_I2C #从ssd1306模块中导入SSD1306_I2C子模块
11
12 i2c = I2C(sda=Pin(5), scl=Pin(4)) #I2C初始化: sda--> 5, scl --> 4

```

Filesystem on ESP board

Files on your device:	在电脑上的文件:
boot.py ssd1306.py main.py	main.py ssd1306.py

图 4-24 将 main.py、ssd1306.py 拷贝到设备

● 实验结果:

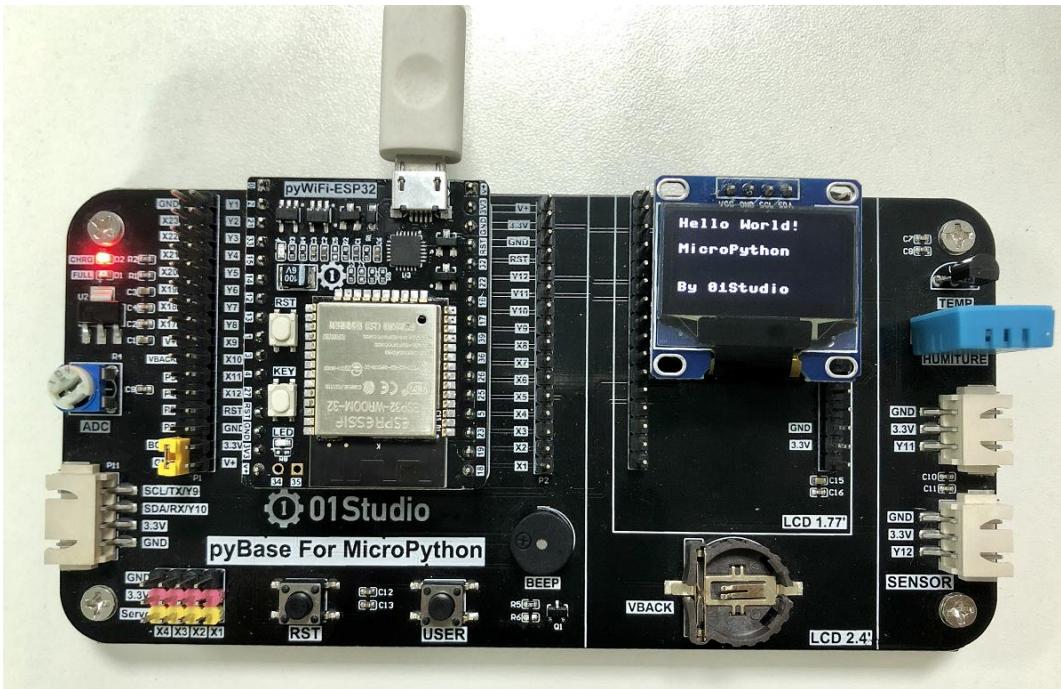


图 4-25 OLED 实验结果

● 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。

4.6 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。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。



图 4-26 时钟

- 实验平台：

pyWiFi-ESP32 和 pyBase 开发底板。

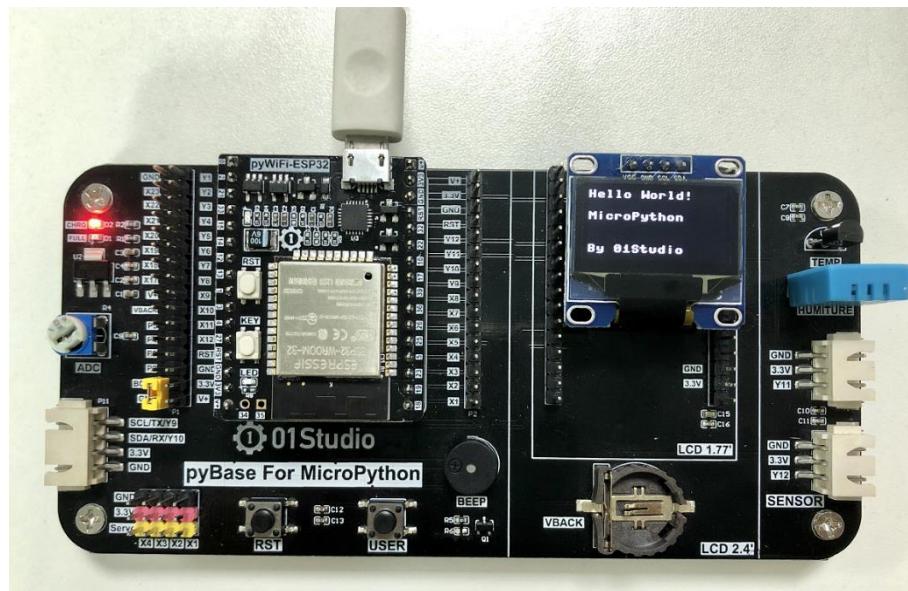


图 4-27 pyWiFi-ESP32 开发套件

- **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 MicroPython 已经集成了内置时钟函数模块。位于 `machine` 的 RTC 模块中，具体介绍如下：

构造函数
<code>rtc=machine.RTC()</code>
构建 RTC 对象。
使用方法
<code>rtc.datetime((2019, 4, 1, 0, 0, 0, 0))</code>
设置日期和时间。按顺序分别是：(年，月，日，星期，时，分，秒，微秒)，其中星期使用 0-6 表示周一至周日。
<code>rtc.datetime()</code>
获取当前日期和时间。

表 4-7

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，使用前面学习过的 RTOS 定时器来计时，具体编程流程如下：

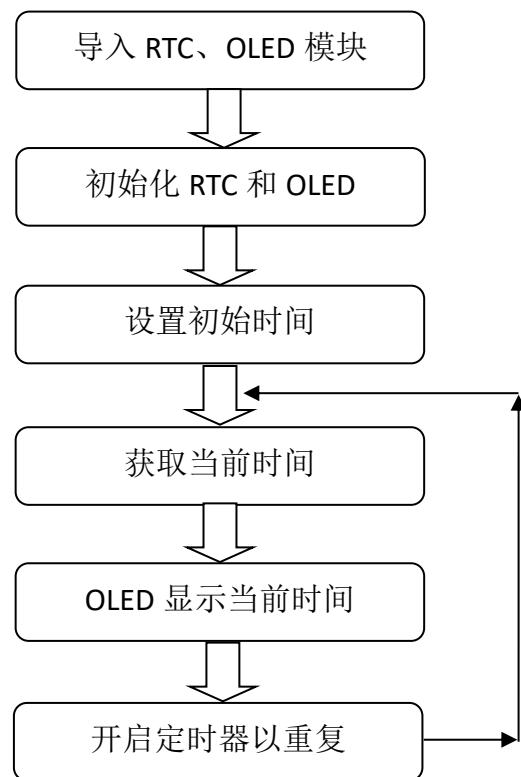


图 4-28

实验参考代码如下：

```

...
实验名称: RTC 实时时钟
版本: v1.0
日期: 2019.7
作者: 01Studio
...

# 导入相关模块
from machine import Pin, I2C, RTC, Timer
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_list = ['', '', '']

# 初始化所有相关对象

i2c = I2C(sda=Pin(13), scl=Pin(14)) #I2C 初始化: sda-->13, scl -->14
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, 0, 0, 0, 0))

def RTC_Run(tim):
    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]], 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”以
    # 达到较佳的显示效果

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

```

```

        time_list[i - 4] = ""

# 显示时间

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

oled.show()

#开启 RTOS 定时器

tim = Timer(-1)

tim.init(period=300, mode=Timer.PERIODIC, callback=RTC_Run) #周期 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 pyWiFi-ESP32 的文件系统里面。

● 实验结果：

烧录程序复位后，可以看到时钟开始跑起来。

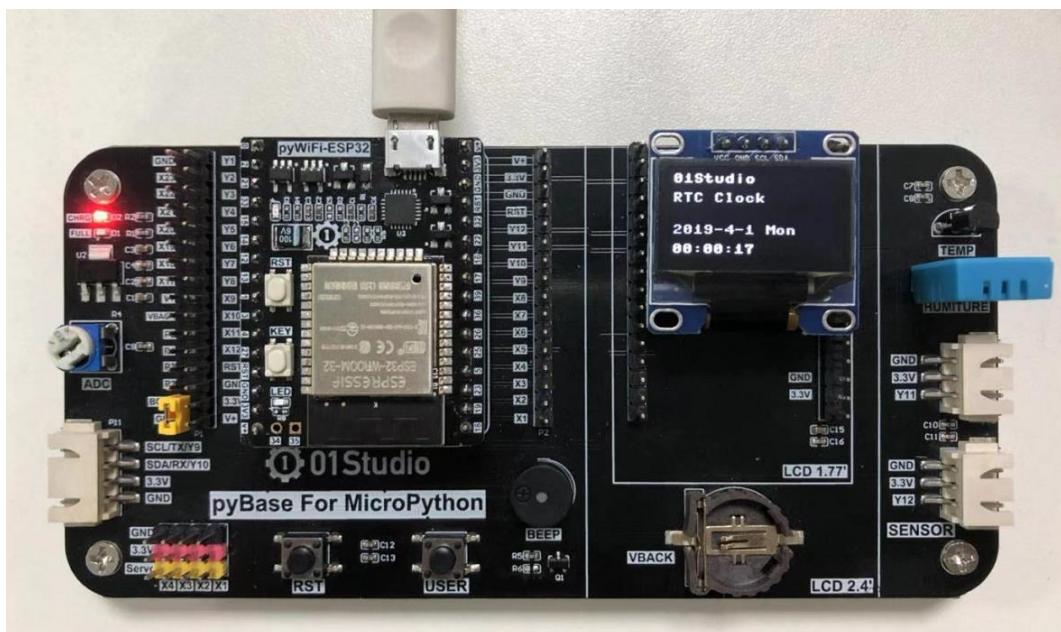


图 4-29 RTC 实时时钟实验

由于 ESP32 没有后备电池引脚，所以不支持掉电保存。因此 pybase 上面的纽扣电池是不起作用的。

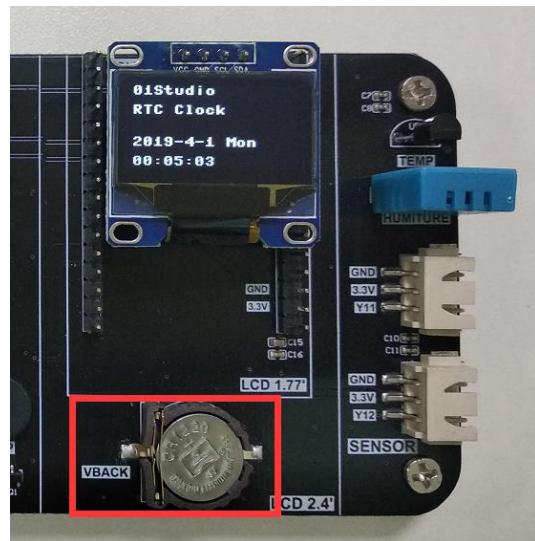


图 4-30 后备电池 (VBACK)

● 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。

4.7 ADC (电位器)

● 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：

pyWiFi-ESP32 和 pyBase 开发底板。

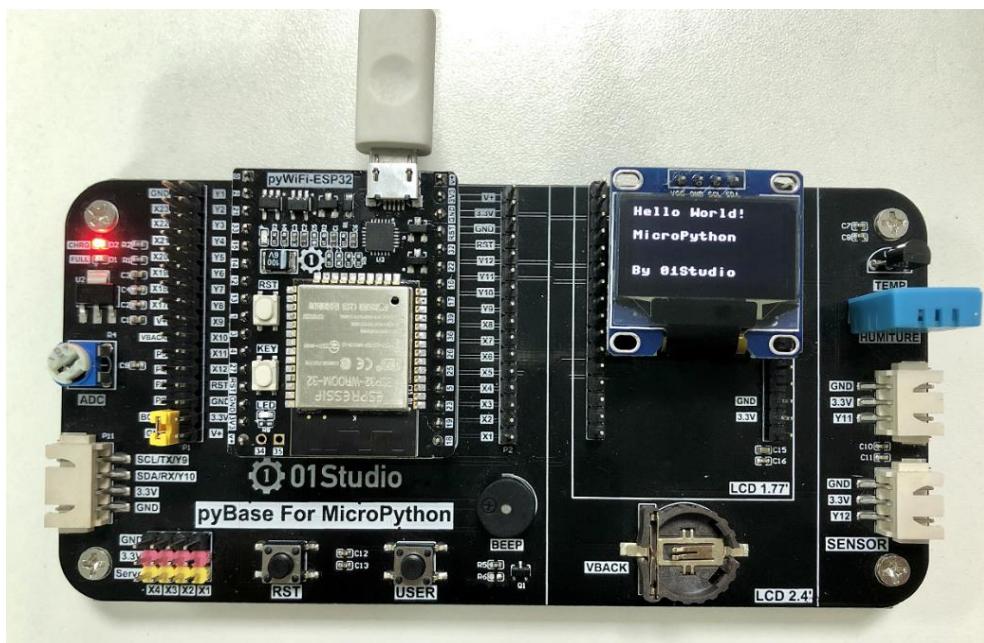


图 4-31 pyWiFi-ESP32 开发套件

● 实验目的：

通过编程调用 MicroPython 的内置 ADC 函数，实现测量输入电压，并显示到屏幕上。

● 实验讲解：

pyBase 开发底板的 X7 引脚连接到了电位器，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。

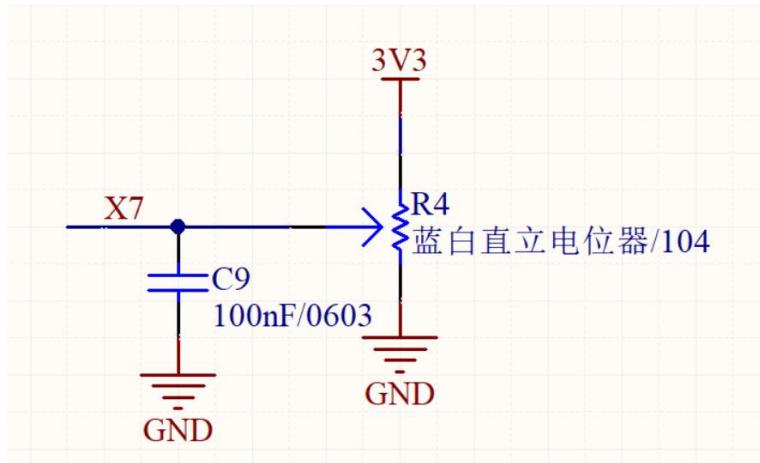


图 4-32 电位器原理图

电位器引脚对应 pyBase 的 X7, 实际是跟 pyWiFi-ESP32 的 36 引脚 ADC 输入引脚相连。ESP32 的 ADC 默认只能测量 0-1V 的量程, 但 ESP32 内部集成了衰减器, 最大支持 11dB 衰减, 通过配置衰减器最多能测量 3V 左右的电压。我们来看看 ADC 模块的构造函数和使用方法。

构造函数
<code>adc=machine.ADC(Pin(id))</code>
构建 ADC 对象。id: ESP32 的 ADC 在 32-39 引脚上。
使用方法
<code>adc.read()</code>
获取 ADC 值。测量精度是 12 位, 返回 0-4095 (表示 0-1V)。
<code>adc.attention(attention)</code>
配置衰减器。配置衰减器能增加电压测量范围, 但是以精度为代价的。 attention: 衰减设置 ADC.ATTN_0DB: 0dB 衰减, 最大输入电压为 1.00v - 这是默认配置; ADC.ATTN_2_5DB: 2.5dB 衰减, 最大输入电压约为 1.34v; ADC.ATTN_6DB: 6dB 衰减, 最大输入电压约为 2.00v; ADC.ATTN_11DB: 11dB 衰减, 最大输入电压约为 3.3v。

表 4-8 ADC 对象

你没看错，就这么简单，两句函数就可以获得 ADC 数值。我们将在本实验中以默认的量程 0-1V 来测试。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 300 毫秒读取一次，具体如下：

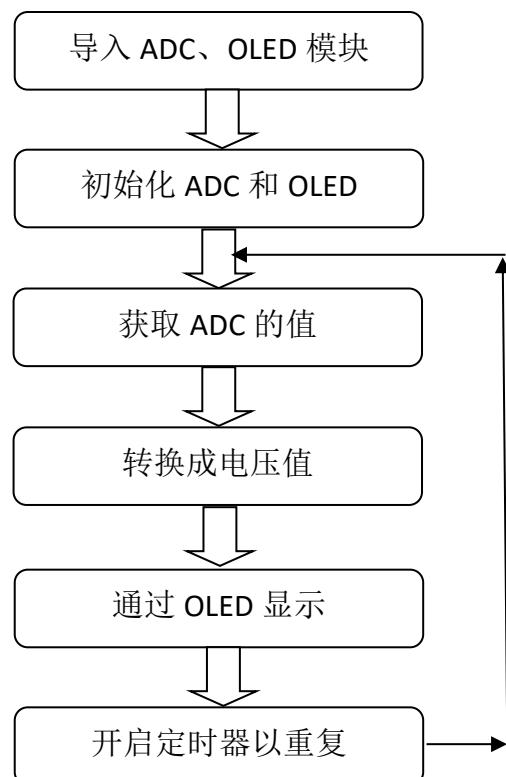


图 4-33 代码编写流程

实验参考代码：

```
...
实验名称: ADC-电压测量
版本: v1.0
日期: 2019.7
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 0-1V。
...
```

```
#导入相关模块

from machine import Pin,I2C,ADC,Timer

from ssd1306 import SSD1306_I2C


#初始化相关模块

i2c = I2C(sda=Pin(13), scl=Pin(14)) #I2C 初始化: sda-->13, scl -->14
oled = SSD1306_I2C(128, 64, i2c, addr=0x3c)

adc = ADC(Pin(36)) #36 引脚跟 pyBase 的电位器相连接


def ADC_Test(tim):

    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 0-1V，('%.2f'%) 表示保留 2 位小数
    oled.text(str('%.2f'%(adc.read()/4095)),0,55)
    oled.text('V',40,55)

    oled.show()

#开启 RTOS 定时器

tim = Timer(-1)

tim.init(period=300, mode=Timer.PERIODIC, callback=ADC_Test) #周期 300ms
```

● 实验结果：

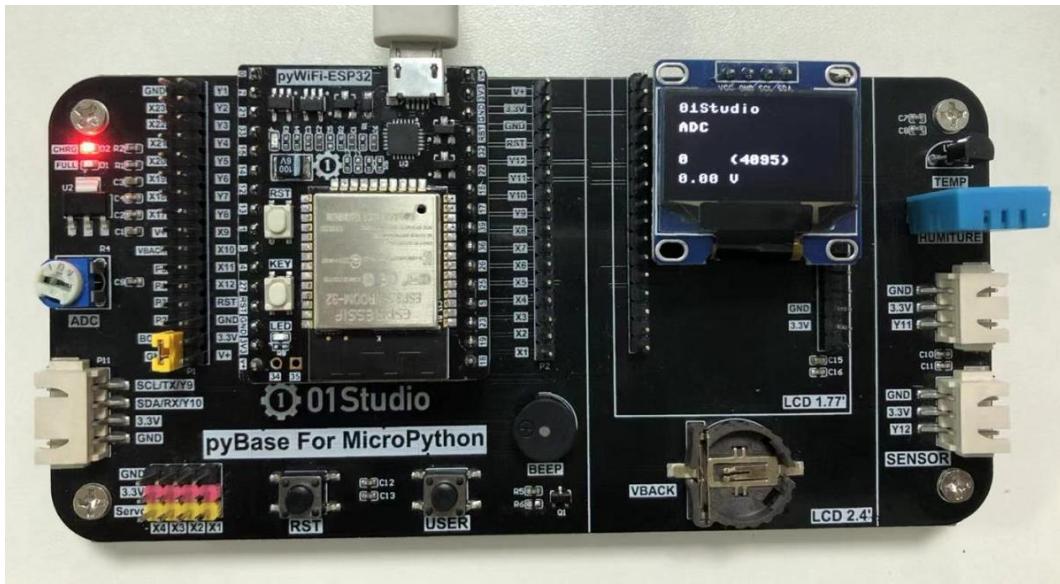


图 4-34 电位器顺时钟拧到尽头是 0V

通过调节电位器，可以发现电压在不断变化。由于本实验的电压测量量程限制，当超出 1V 量程后，始终显示电压在 1V，测量值是 4095。

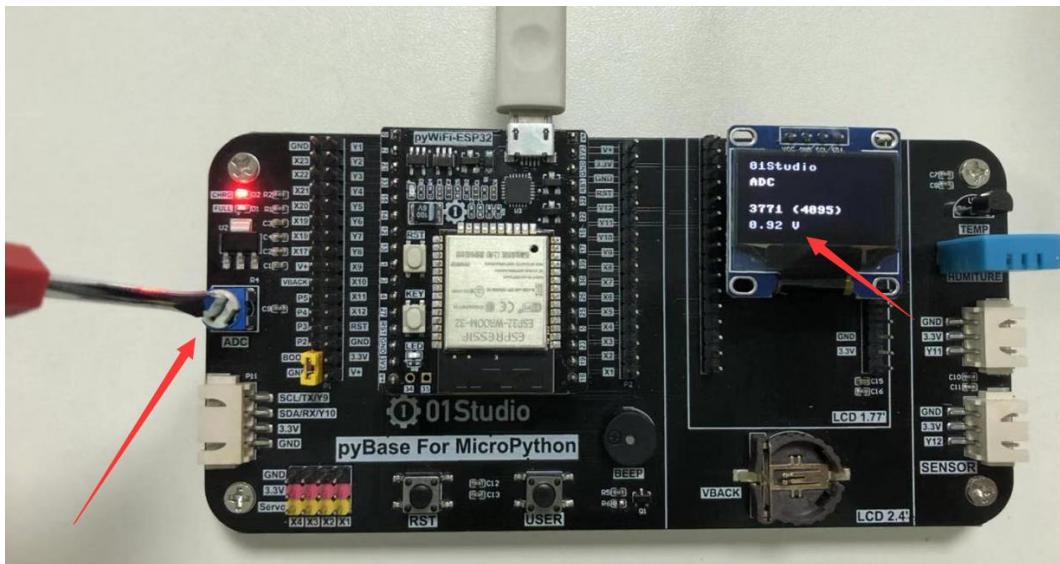


图 4-35 调节电位器来改变电压输入

● 总结：

这一节我们学习了 ADC 的应用，主要用于电压的检测。有兴趣的用户可以尝试使用其衰减器测试，可以扩充电压量程，但精度会有所下降。

4.8 PWM (蜂鸣器)

● 前言：

上一节的 ADC 是信号输入，这节的 PWM 就是一个信号输出。PWM（脉冲宽度调制），主要用于输出不同频率、占空比（一个周期内高电平出现时间占总时间比例）的方波。以实现固定频率或平均电压输出。

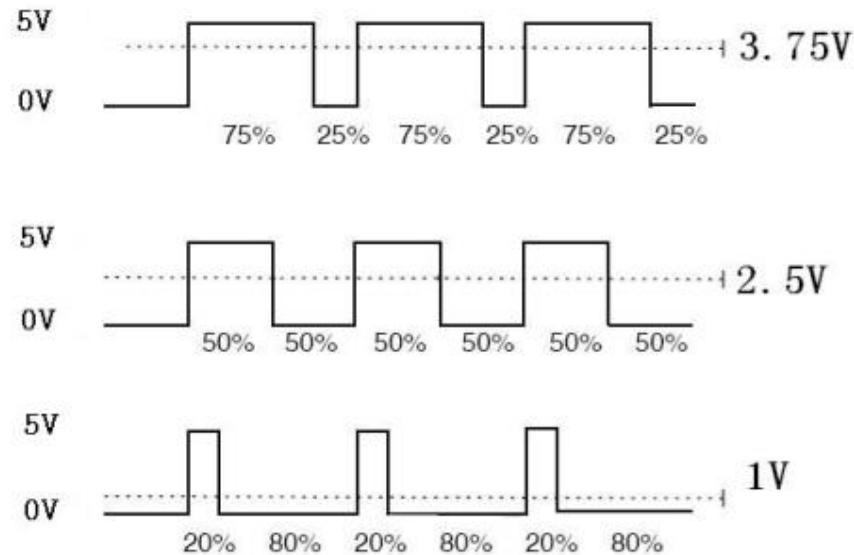


图 4-36 PWM 波形示例

● 实验平台：

pyWiFi-ESP32 和 pyBase 开发底板。

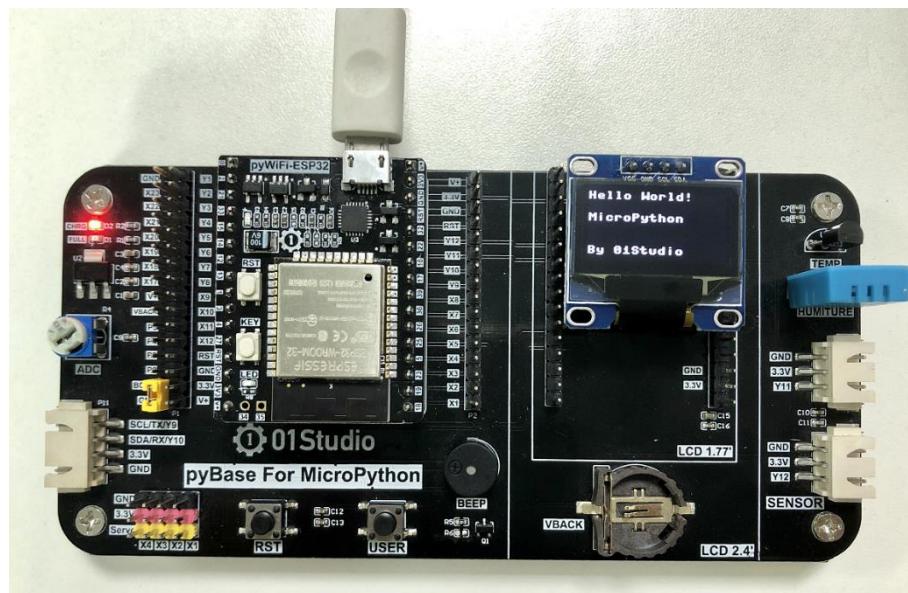


图 4-37 pyWiFi-ESP32 开发套件

● 实验目的:

通过不同频率的 PWM 信号输出，驱动无源蜂鸣器发出不同频率的声音。

● 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 pyWiFi-ESP32 的 PWM 输出频率是在变化的。

pyBase 开发底板上的无源蜂鸣器连接到引脚 X5。如下图所示：

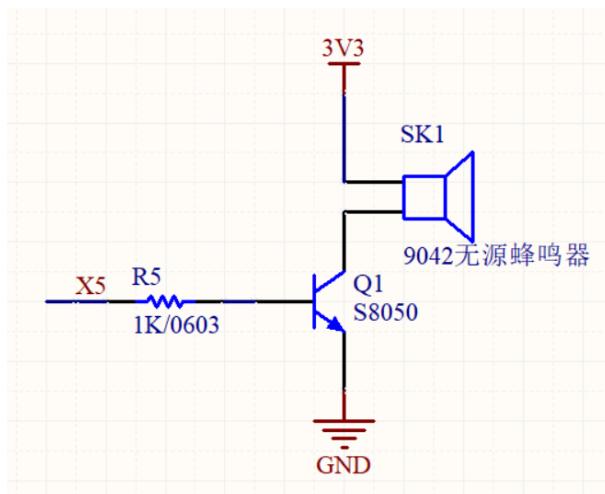


图 4-38 蜂鸣器原理图

PWM 可以通过 ESP32 所有 GPIO 引脚输出。所有通道都有 1 个特定的频率，从 1 到 40M 之间（单位是 Hz）。占空比的值为 0 至 1023 之间。在本实验中我们用到引脚 25。

先看看 PWM 模块对象：

构造函数
<pre>pwm25=machine.PWM(machine.Pin(id),freq,duty)</pre>
构建 PWM 对象。id:引脚编号； freq:频率值； duty:占空比； 配置完后 PWM 自动生效。
使用方法
<pre>pwm25.freq(freq)</pre>
设置频率。freq:频率值在 1-1000 之间， freq 为空时表示获取当前频率值。

<code>pwm25.duty(duty)</code>
设置占空比。 <code>duty</code> : 占空比在 0-1023 之间, <code>duty</code> 为空时表示获取当前占空比值。
<code>pwm25.deinit()</code>
关闭 PWM。

表 4-9 PWM 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, 可以简单理解成占空比为 50% 的 PWM 输出。

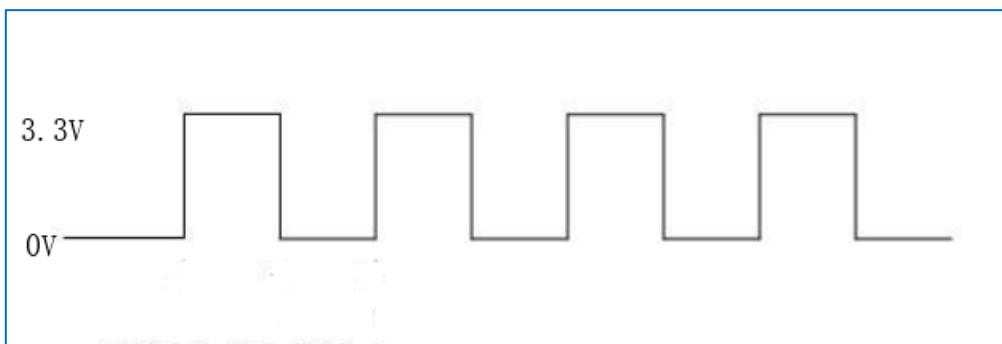


图 4-39 方波信号

结合上述讲解, 总结出代码编写流程图如下:

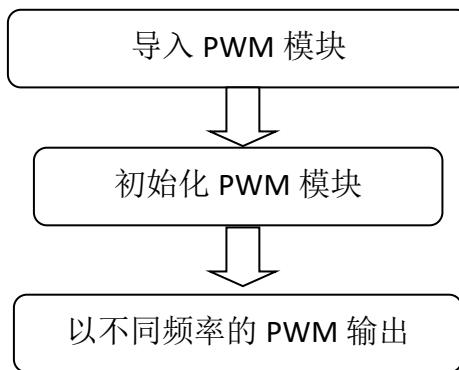


图 4-40 代码编写流程图

实验参考代码:

实验名称: PWM

版本: v1.0

日期: 2019.7

作者: 01Studio

说明: 通过不同频率的 PWM 信号输出, 驱动无源蜂鸣器发出不同频率的声音。

```
from machine import Pin, PWM
```

```
import time
```

```
Beep = PWM(Pin(25), freq=0, duty=512) # 在同一语句下创建和配置 PWM
```

#蜂鸣器发出频率 200Hz 响声

```
Beep.freq(200)
```

```
time.sleep_ms(1000)
```

#蜂鸣器发出频率 400Hz 响声

```
Beep.freq(400)
```

```
time.sleep_ms(1000)
```

#蜂鸣器发出频率 600Hz 响声

```
Beep.freq(600)
```

```
time.sleep_ms(1000)
```

#蜂鸣器发出频率 800Hz 响声

```
Beep.freq(800)
```

```
time.sleep_ms(1000)
```

```
#蜂鸣器发出频率 1000Hz 响声
```

```
Beep.freq(1000)
```

```
time.sleep_ms(1000)
```

```
#停止
```

```
Beep.deinit()
```

● 实验结果：

下载程序，复位后可以听到蜂鸣器依次发出不同频率的响声。

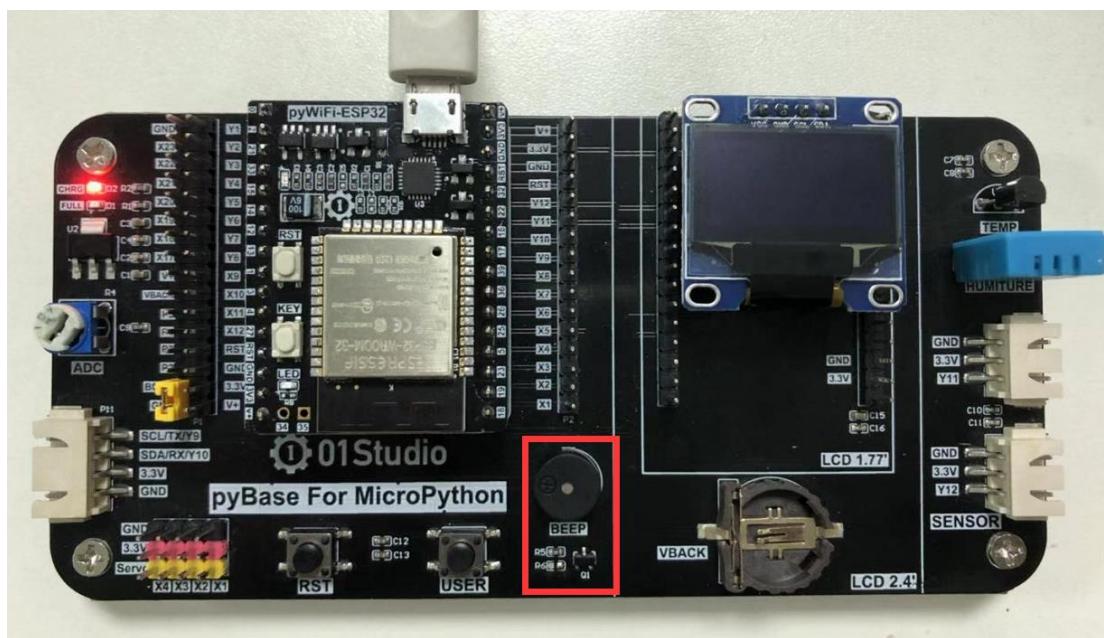


图 4-41 实验现象

有条件的朋友可以使用示波器测量 pyWiFi-ESP32 的 25 引脚或 pyBase 的 X5 接口，观察信号波形的变化：

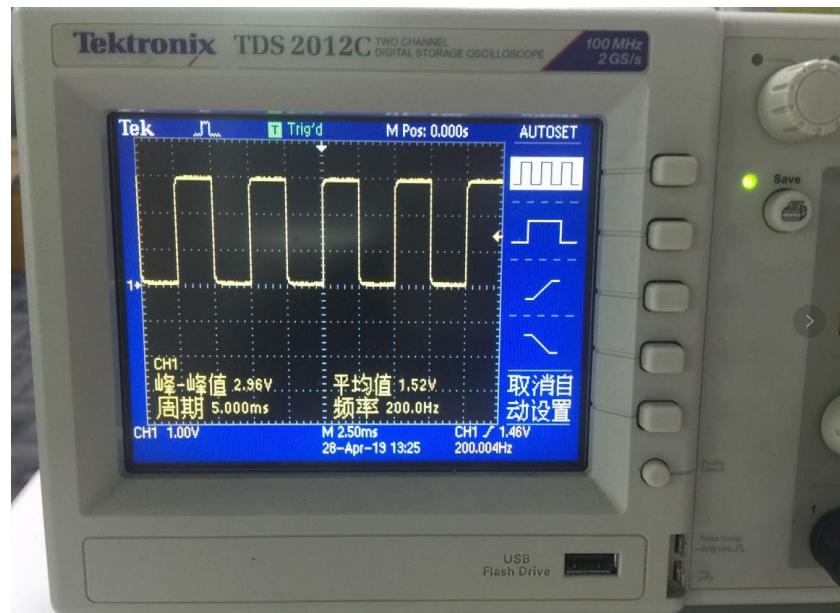


图 4-42 示波器显示波形

● 总结：

到了这一节，我们发现实验中对象函数使用方法非常简单，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。而随着要实现功能的复杂化让编程的代码数量变多，逻辑也将略显复杂。

4.9 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：

pyWiFi-ESP32 开发套件，需要使用 USB 转 TTL 工具。

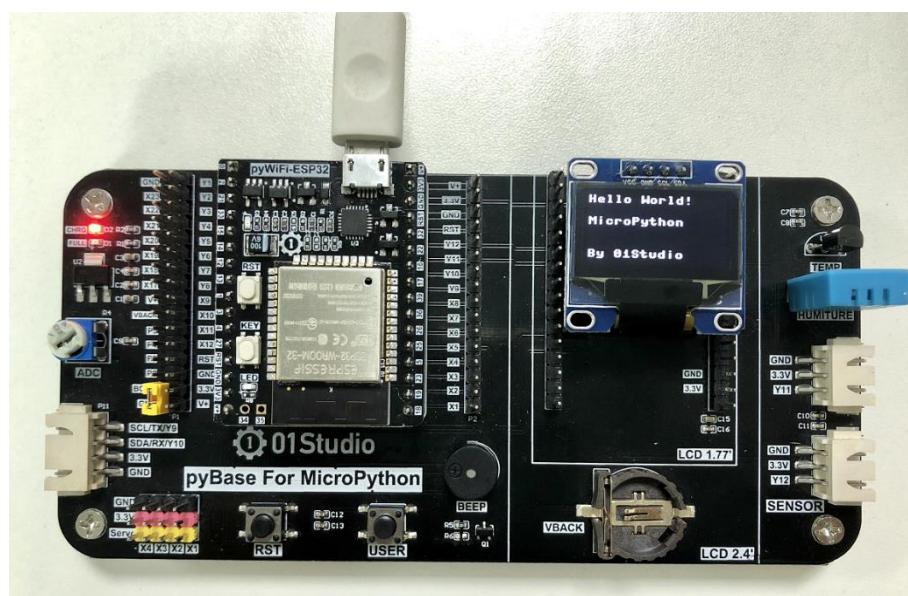


图 4-43pyWiFi-ESP32 开发板

● 实验目的：

编程实现串口收发数据。

● 实验讲解：

pyWiFi-ESP32 开发板一共有 3 个串口，编号是 0-2，如下表：

UART(0)	TX	1
	RX	3
UART(1)	TX	10

	RX	9
UART(2)	TX	17
	RX	16

表 4-10 ESP32 串口引脚资源

UART0 用于下载和 REPL 调试，UART1 用于模块内部连接 Flash 没有引出，因此可以使用 UART2 来调试。

我们来了解一下串口对象的构造函数和使用方法：

构造函数
<code>uart=machine.UART(id,baudrate,tx=None,rx=None bits=8, parity=None, stop=1,...)</code>
创建 UART 对象。
【id】 0-2 【baudrate】 波特率，常用 115200、9600 【tx】 自定义 IO 【rx】 自定义 IO 【bits】 数据位 【parity】 校验；默认 None, 0(偶校验), 1(奇校验) 【stop】 停止位，默认 1 ...
特别说明： ESP32 的 UART 引脚映射到其它 IO 来使用，比如 UART2 默认引脚是 TX—17,RX—16；用户可以通过构造函数定义 tx=18,rx=19 的方式来改变串口引脚，实现更灵活的应用。
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

uart.read([nbytes])
【nbytes】读取字节数
UART.readline()
读行
UART.write(buf)
【buf】串口 TX 写数据
*更多使用说明请阅读官方文档： http://docs.01studio.org/esp32/quickref.html#uart-serial-bus

表 4-11 machine 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。

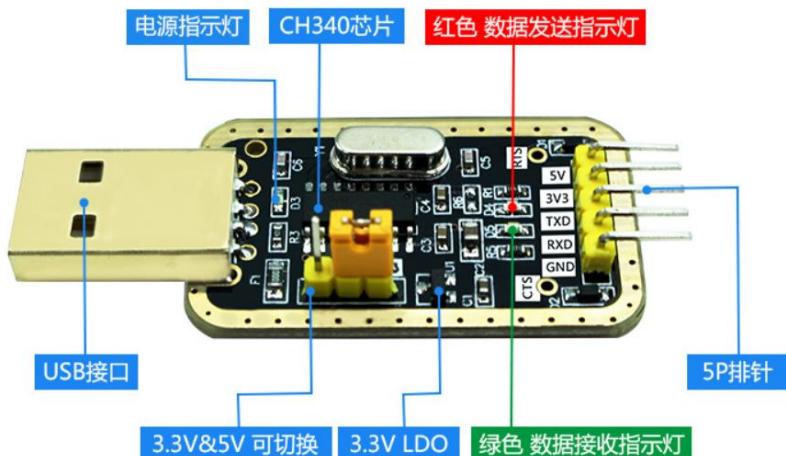


图 4-44 常用 USB 转串口工具

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 2，也就是 17 (TX) 和 16 (RX)，接线示意图如下：



图 4-45 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：

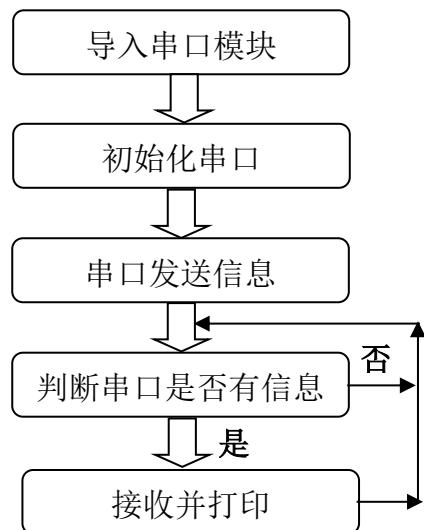


图 4-46 代码编程流程图

参考代码：

```
...
实验名称：串口通信
版本：v1.0
日期：2021.6
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。
平台：pyWiFi-ESP32
参考：http://docs.01studio.org/esp32/quickref.html#uart-serial-bus
...
#导入串口模块
from machine import UART
```

```
uart=UART(2,115200) #设置串口号 2 和波特率,TX--Y9--17,RX--Y10--16

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(128) #接收 128 个字符

        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 接到开发板的 RX (16), USB 转 TTL 的 RX 接到开发板的 TX (17)。GND 接一起，3.3V 可以选择接或不接。

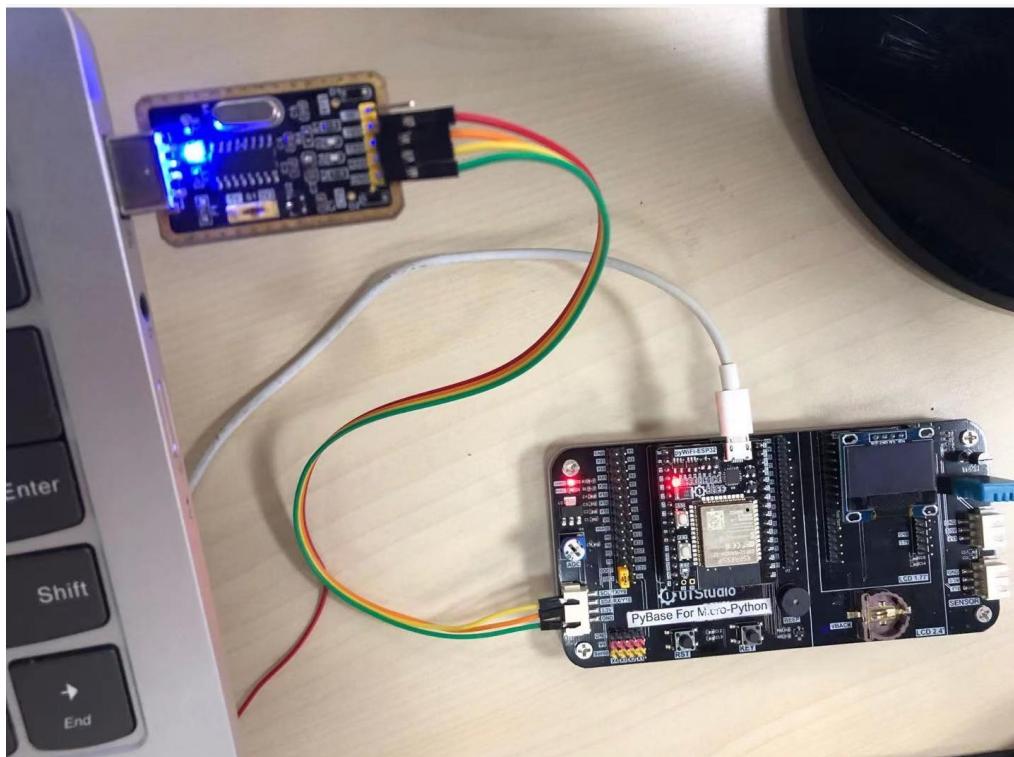


图 4-47 将 pyWiFi-ESP32 和 usb ttl 工具同时接入电脑

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 pyWiFi-ESP32 的串口 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。

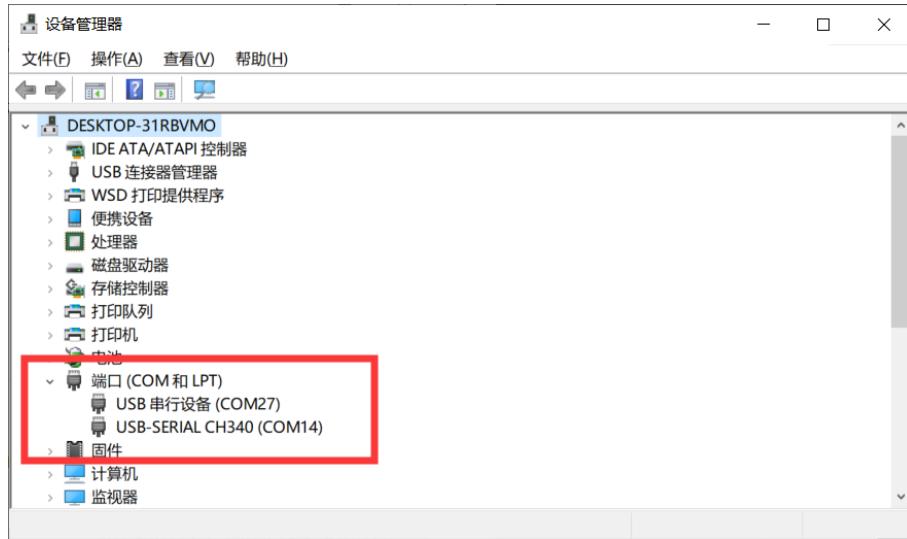


图 4-48

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。

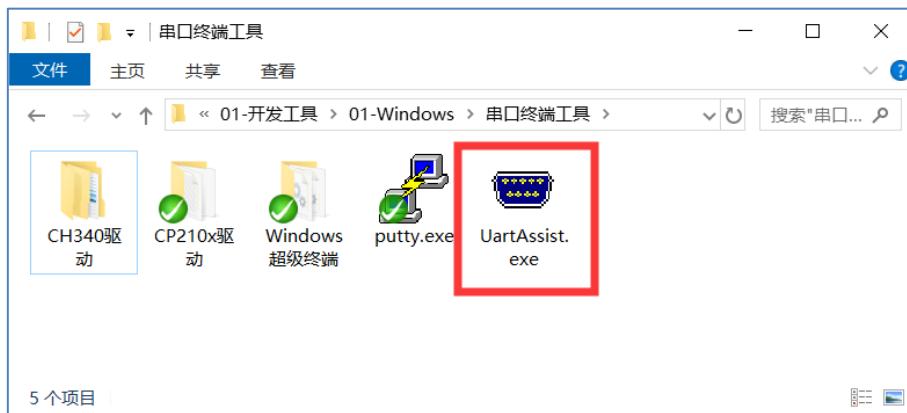


图 4-49

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开发板上电发来的信息“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 pyWiFi-ESP32 在接收到该信息后在 REPL 里面打印了出来。如下图所示：

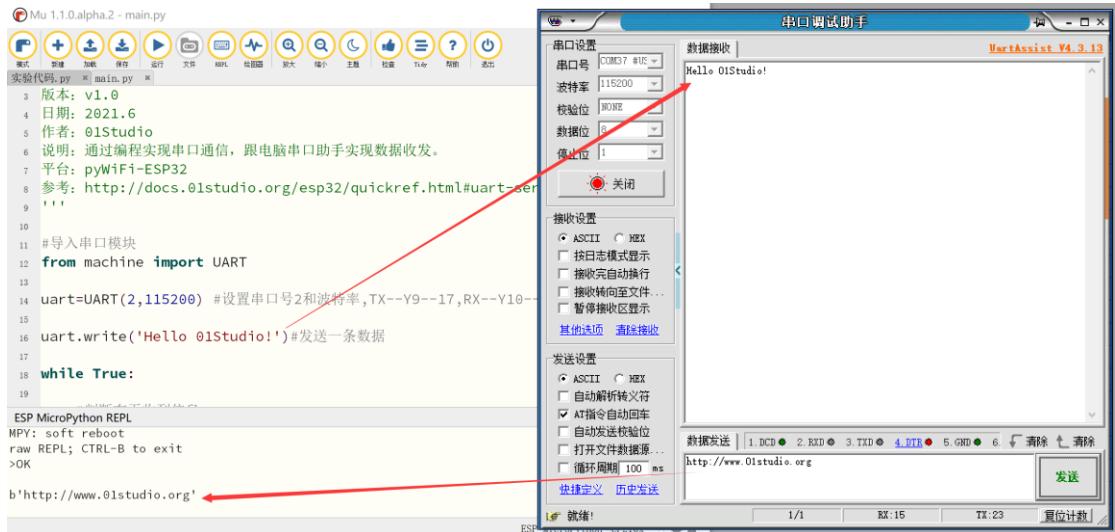


图 4-50 串口收发实验

● 总结:

通过本节我们学会了串口收发应用，pyWiFi-ESP32 的串口资源非常丰富，因此可以接非常多的串口外设。从而实现更多的功能。

4.10 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。

● 实验平台：

pyWiFi-ESP32 开发套件、3.2 寸显示屏（电阻触摸）。

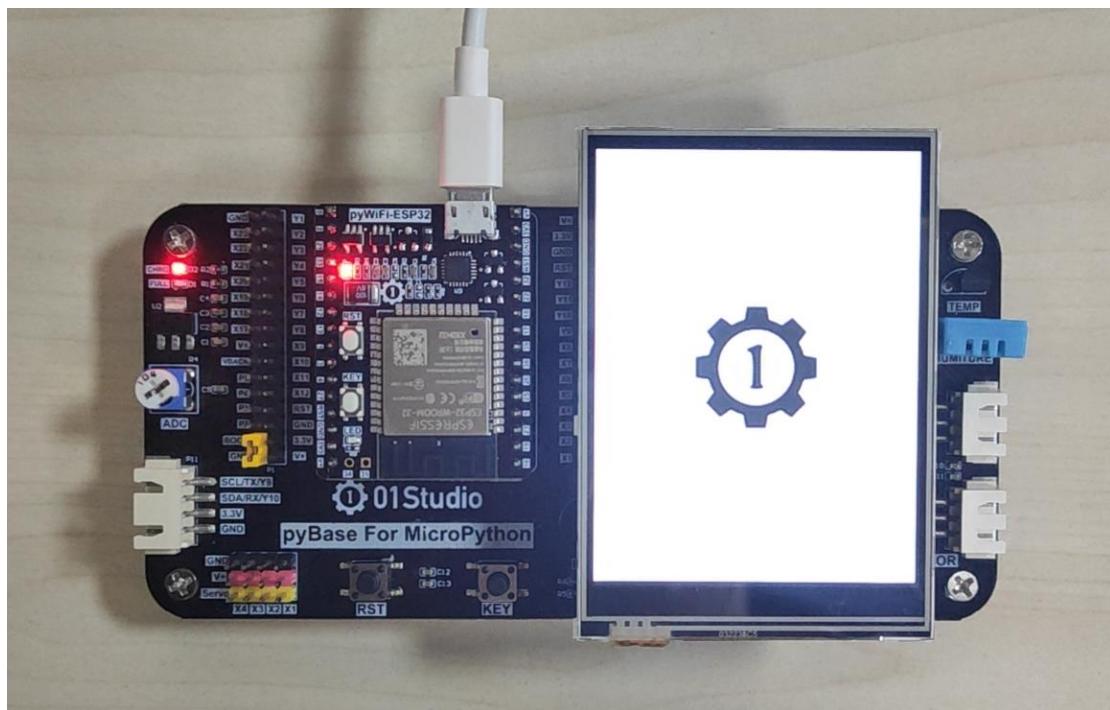


图 4-51 开发板+3.2 寸显示屏

● 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：

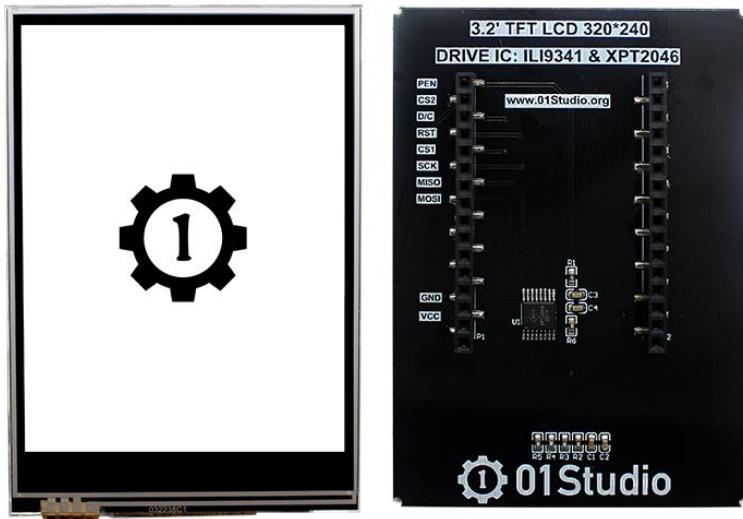


图 4-52 3.2 寸 LCD 显示屏 (电阻触摸)

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 4-12 功能参数

实验用的 LCD 是 3.2 寸，驱动是的 ILI9341，使用 SPI 方式跟 ESP32 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。总的来说和前面实验一样，有构造函数和功能函数。构造函数解决的是初始化问题，告诉开发板该外设是怎么接线，初始化参数如何，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。我们来看看达芬奇开发板 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。
【portrait】 设置屏幕方向： <ul style="list-style-type: none">● 1 - 竖屏, 240*320 , 默认● 2 - 横屏, 320*240 , 1 基础上顺时针旋转 90°● 3 - 竖屏, 240*320 , 1 基础上顺时针旋转 180°● 4 - 横屏, 320*240 , 1 基础上顺时针旋转 270°
使用方法
<code>LCD32.fill(color)</code>
【color】 RGB 颜色数据；如(255,0,0)表示红色。
<code>LCD32.drawPixel(x,y,color)</code>
画点。 【x】 :横坐标， 【y】 :纵坐标， 【color】 :颜色。
<code>LCD32.drawLine(x0,y0,x1,y1,color)</code>
画线段。 【x0,y0】 :起始坐标， 【x1,y1】 :终点坐标， 【color】 :颜色

LCD32.drawRect(x,y,width,height,color,border=1,fillcolor=None)

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: "/cat.jpg"

(‘/’ 表示开发板的板载 flash 的根目录。)

表 4-13 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：

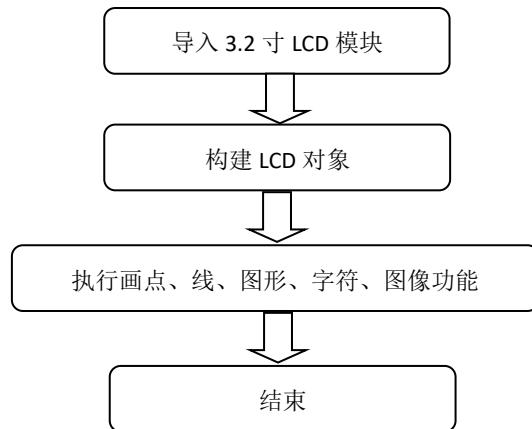


图 4-53 编程流程图

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: pyWiFi-ESP32  
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,"/picture/1.jpg")
time.sleep(3)
d.Picture(0,0,"/picture/2.jpg")
time.sleep(3)
d.Picture(0,0,"/picture/01studio.jpg")
```

● 实验结果：

将示例程序的素材文件上传到 pyWiFi-ESP32 开发板。(也可以只上传单张图片，注意修改代码中文件的路径即可。)

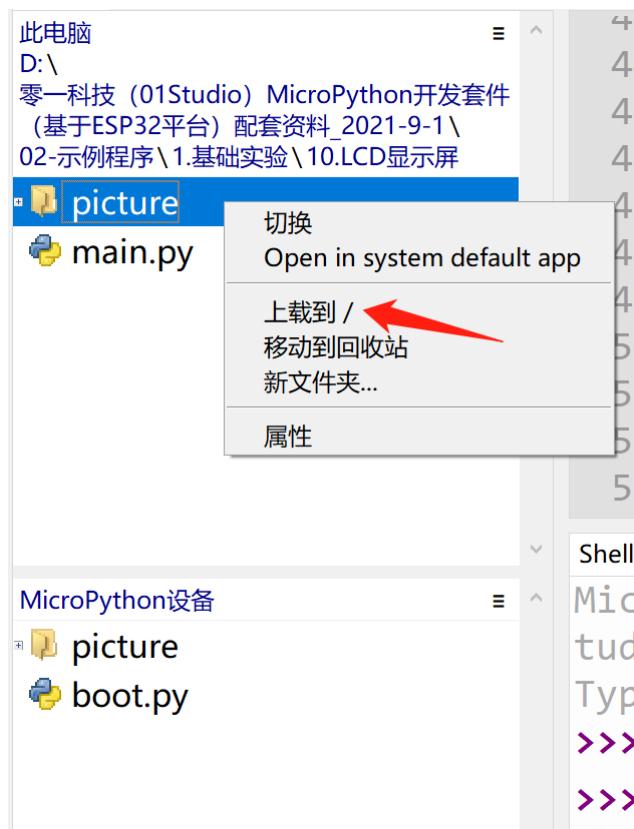


图 4-54 上传图片文件夹到 Flash

运行程序，可以看到 LCD 依次显示相关内容。

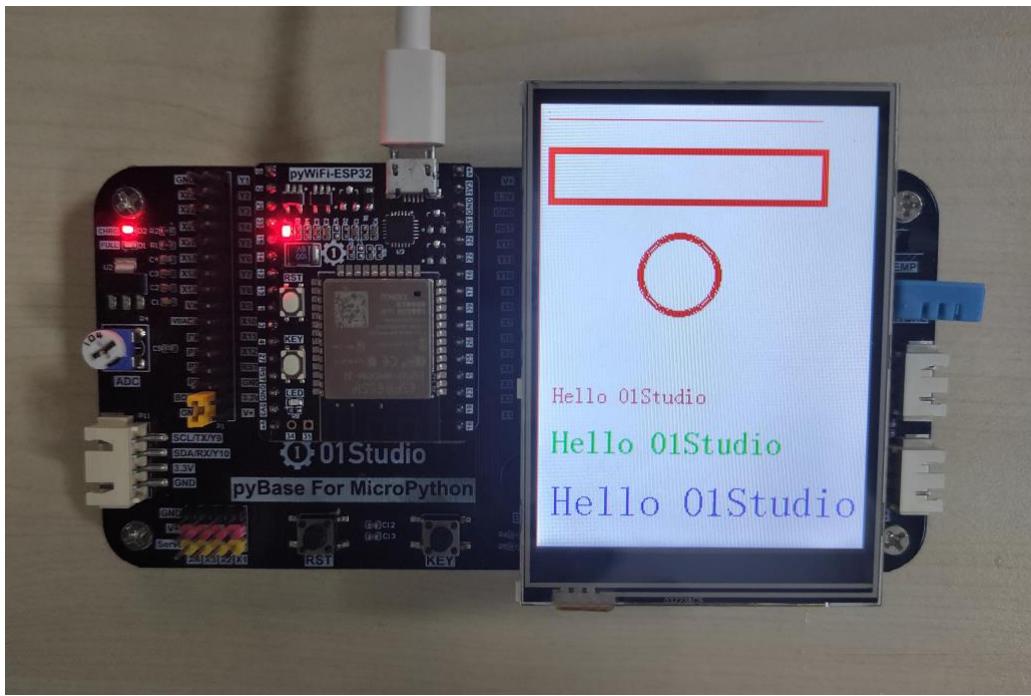


图 4-55 点、线、字符等显示



图 4-56 显示图片

- **总结：**

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！

4.11 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:**

pyWiFi-ESP32 开发套件、3.2 寸显示屏（电阻触摸）。

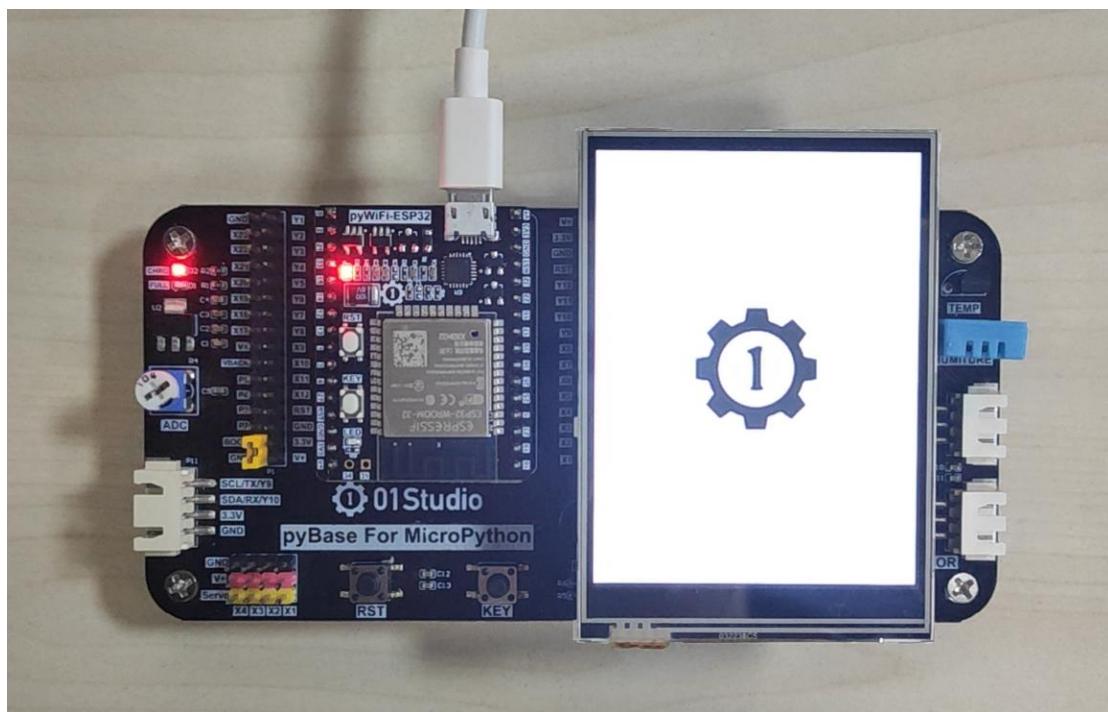


图 4-57 3.2 寸 LCD(电阻触摸)

- **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
touch.XPT2046(portrait=1)
构建触摸屏对象。 XPT2046 表示驱动芯片型号。
<p>【portrait】 设置屏幕方向：</p> <ul style="list-style-type: none"> ● 1 - 竖屏, 240*320 , 默认 ● 2 - 横屏, 320*240 , 1 基础上顺时针旋转 90° ● 3 - 竖屏, 240*320 , 1 基础上顺时针旋转 180° ● 4 - 横屏, 320*240 , 1 基础上顺时针旋转 270°
使用方法
XPT2046.tick_inc()
手动刷新触摸。
XPT2046.read()
读取触摸屏数据，返回 (states,x,y)
【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 4-14 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：

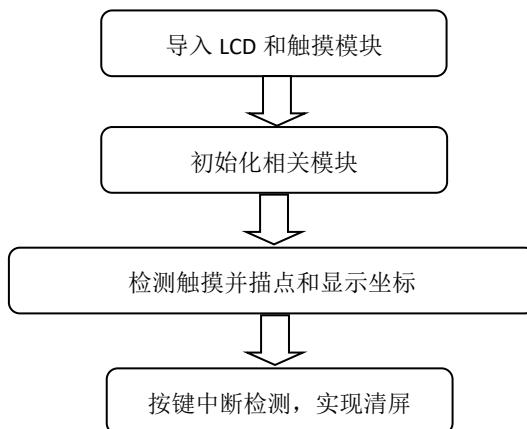


图 4-58 编程流程图

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：pyWiFi-ESP32
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 4-59 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。

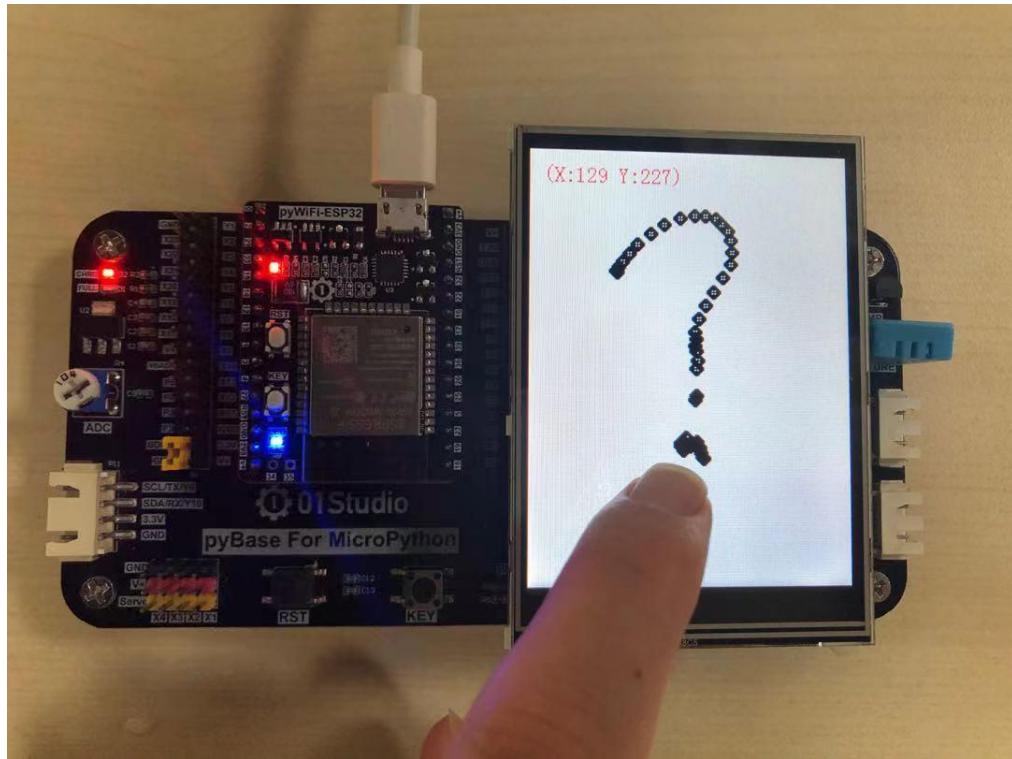


图 4-60 触摸实验

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！

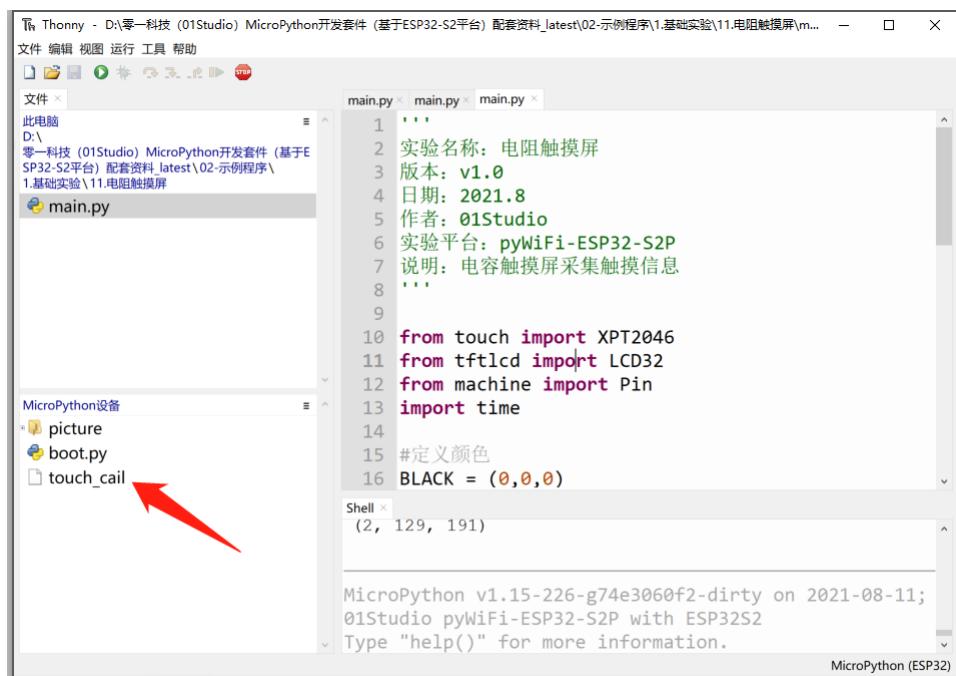


图 4-61 触摸校准文件

● 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。

4.12 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:**

pyWiFi-ESP32 开发套件、3.2 寸显示屏（电阻触摸）。

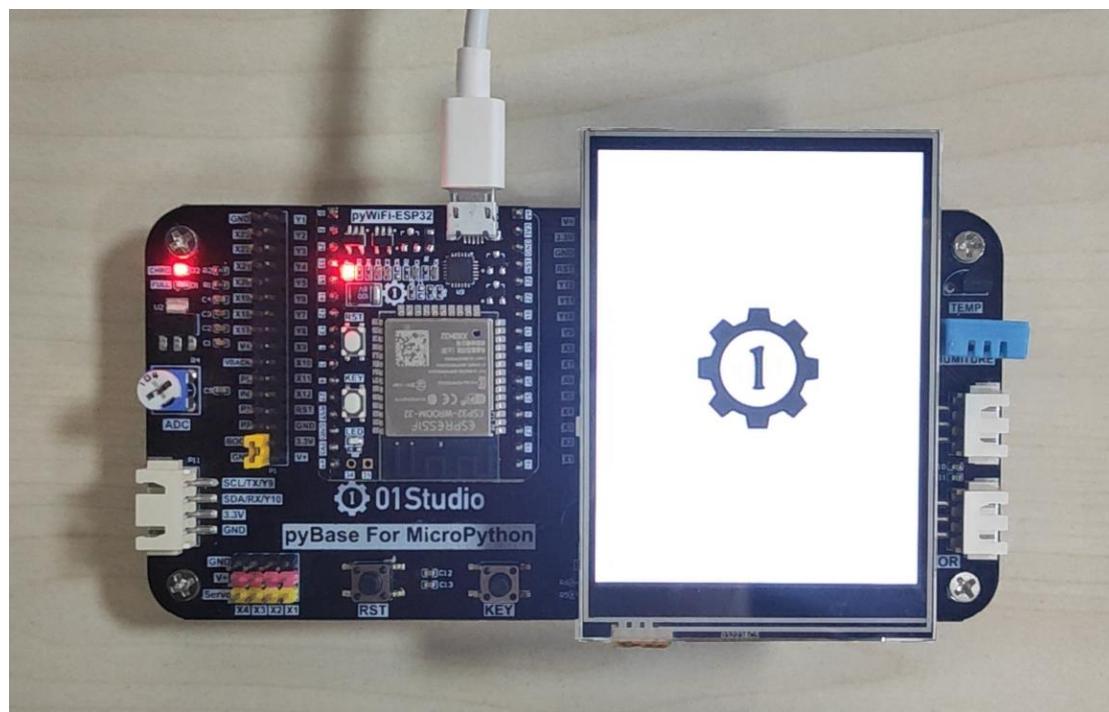


图 4-62 3.2 寸 LCD(电阻触摸)

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

表 4-15 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：

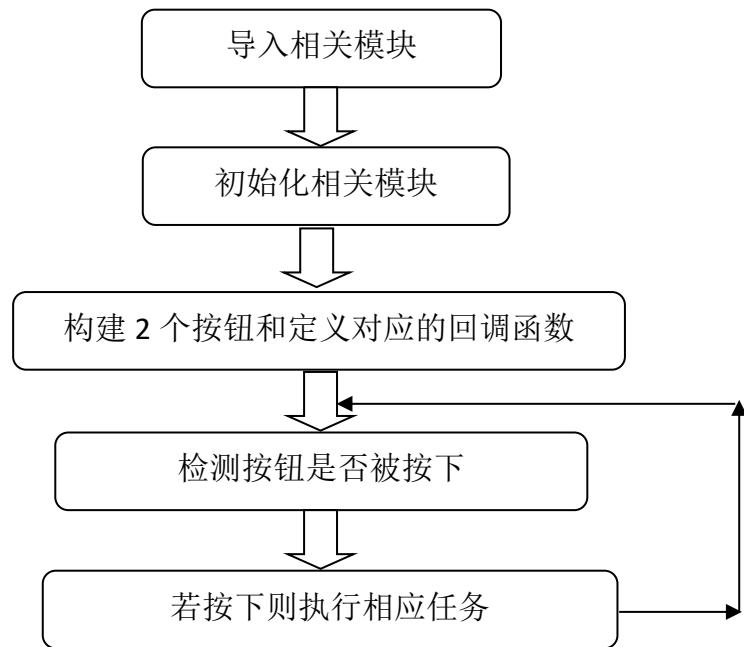


图 4-63 代码编写流程

参考代码如下：

```

...
实验名称: 触摸按钮
版本: v1.0
日期: 2021.8
作者: 01Studio
社区: www.01studio.cc
说明: 编程实现触摸按钮控制 LED。
...

from tftlcd import LCD32
from touch import XPT2046
from machine import Timer,Pin
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

LED=Pin(2,Pin.OUT) #构建 LED 对象,开始熄灭
state=0 #LED 引脚状态

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

    #LED 灯状态翻转
    global state
    state=not state #使用 not 语句而非~语句
    LED.value(state) #LED 状态翻转

def fun2(B2):

    print('Button is pressed!')
```

```
B1 = gui.TouchButton(80,50,80,50,BLUE,'LED1',WHITE,fun1)
B2 = gui.TouchButton(80,120,80,50,RED,'Button',WHITE,fun2)

#####
# ##### 定时器用于扫描按钮触发事件 ##
#####

tim_flag = 0

def count(tim):
    global tim_flag
    tim_flag = 1

#构建软件定时器，编号 1
tim = Timer(1)
tim.init(period=20, mode=Timer.PERIODIC,callback=count) #周期为 20ms

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。

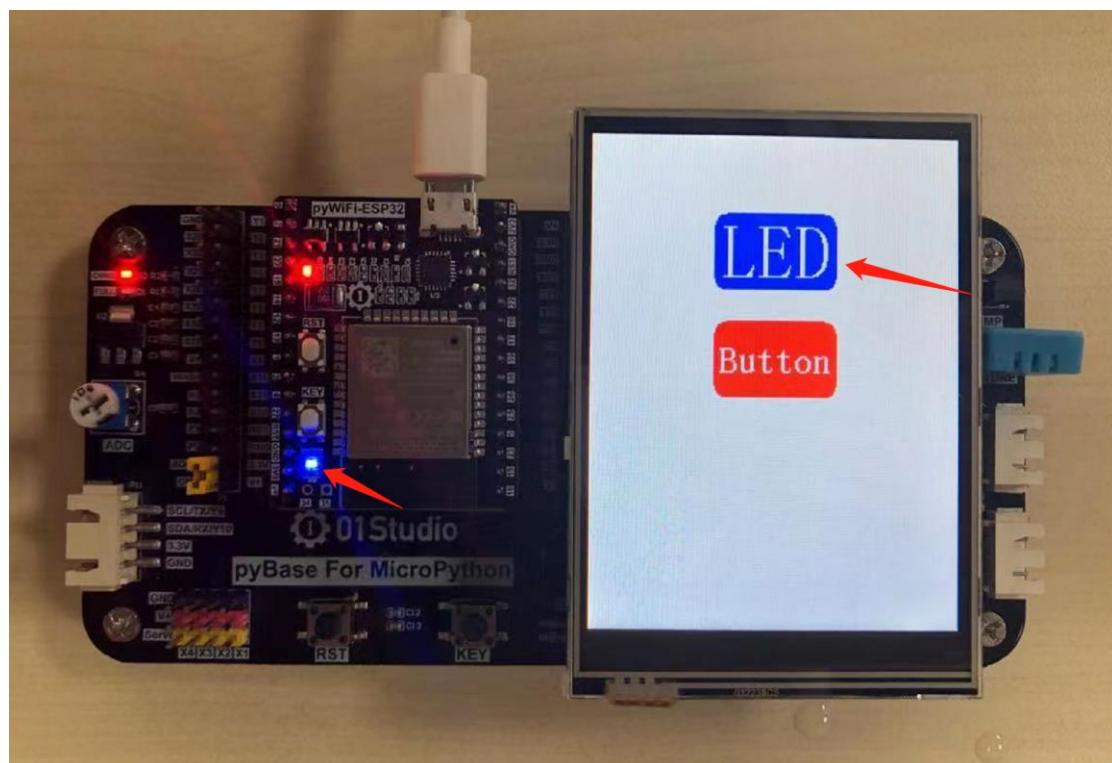


图 4-64 LED 控制

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

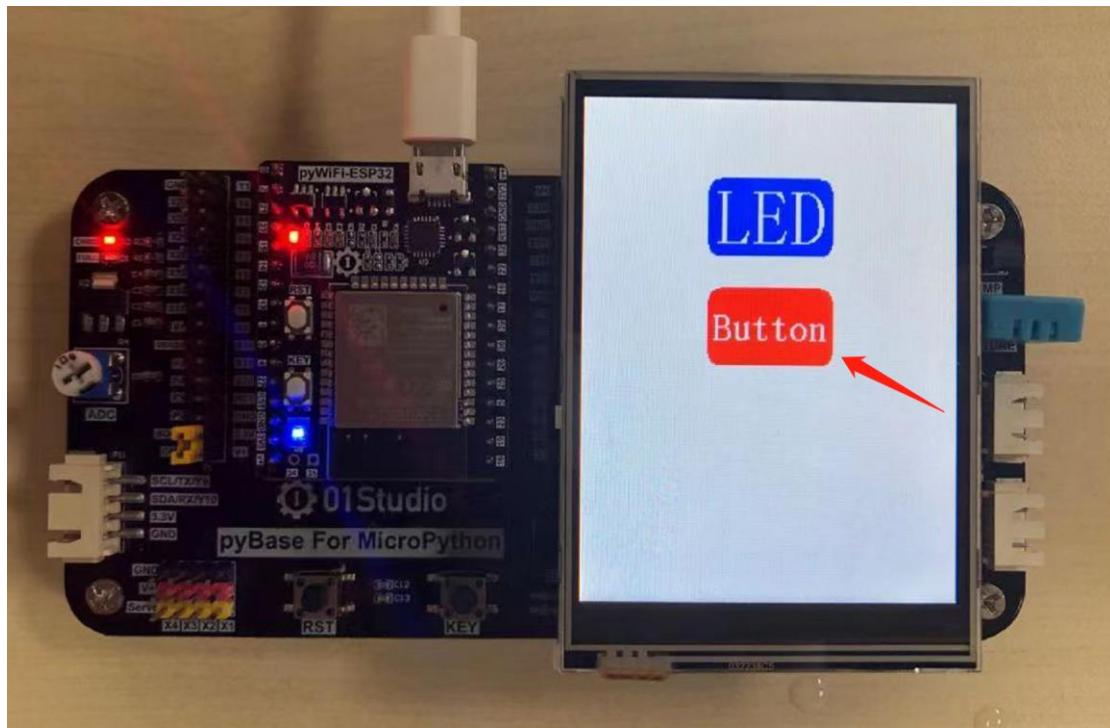


图 4-65 Button 按钮

```

57     global tim_flag
58     tim_flag = 1
59
60     #构建软件定时器，编号1
61     tim = Timer(1)
62     tim.init(period=20, mode=Timer.PERIODIC,callback=
63
64     while True:
65
66         #执行按钮触发的任务
67         if tim_flag == 1:
68
69             t.tick_inc()
70             gui.task_handler()
71             tim_flag = 0
72
    
```

Shell >>> %Run -c \$EDITOR_CONTENT
 Type "help()" for more information.
 >>> Button is pressed!
 Button is pressed!
 Button is pressed! ←

图 4-66 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。

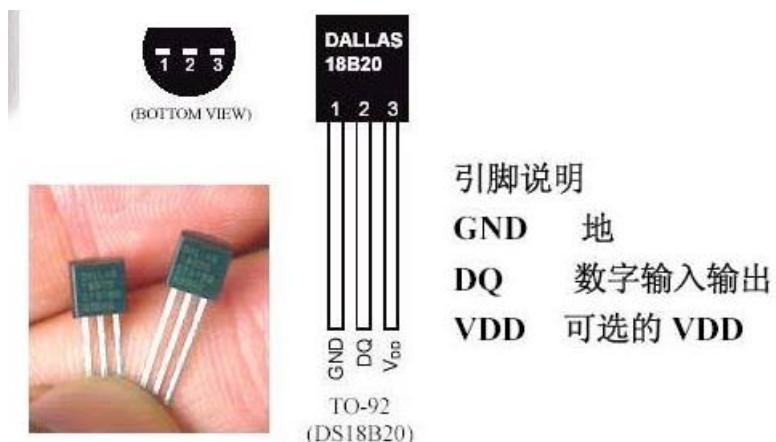


图 5-1 DS18B20 传感器



图 5-2 DS18B20 金属探头封装

- 实验平台：

pyWiFi-ESP32 和 pyBase 开发底板。DS18B20 温度传感器位于右上方。

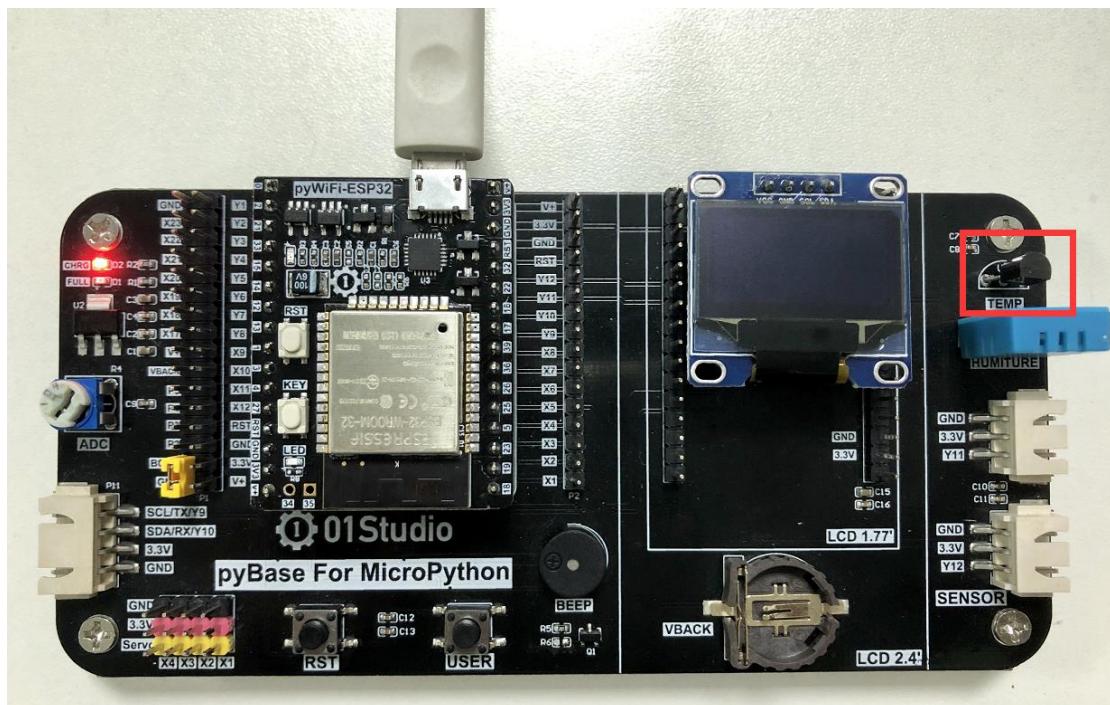


图 5-3 pyWiFi-ESP32 开发套件

- 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 是单总线驱动（onewire）传感器，也就是说只占用 1 个 IO 口。我们来看看原理图：

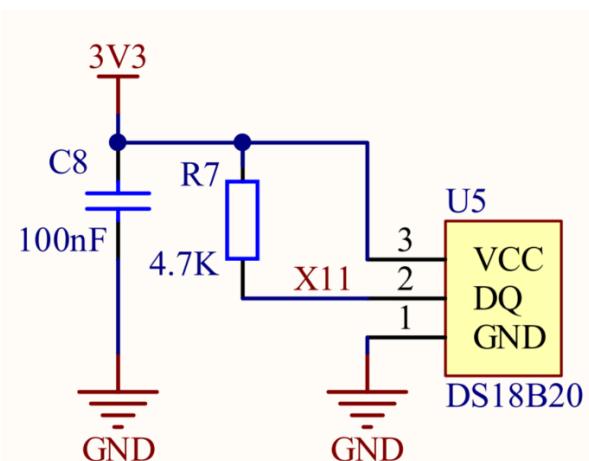


图 5-4 DS18B20 接线原理图

可以看到 DS18B20 传感器连接到了 pyBase 的 X11 引脚上。也就是连接到 pyWiFi-ESP32 的引脚 4，如下图所示：

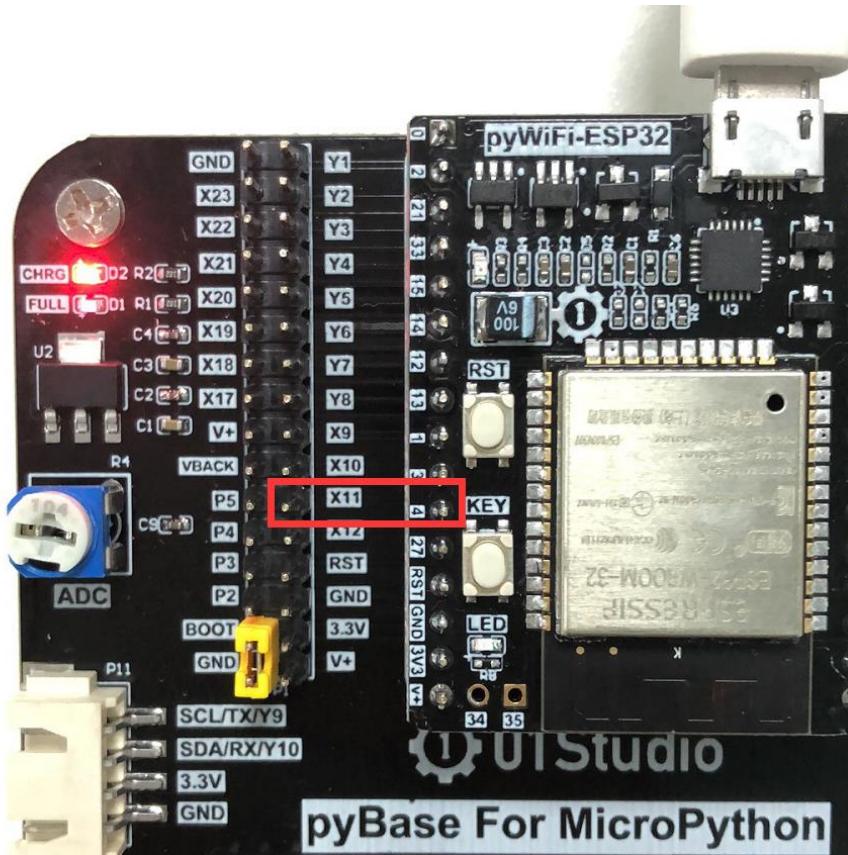


图 5-5 引脚 4 连接到 pybase 底板的 DS18B20

也就是说我们需要针对引脚 4 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，单总线模块文件是：onewire.py，DS18B20 模块的文件是 ds18x20.py。如果你学习过前面基于 STM32 平台应该不陌生。而对于 ESP32，这两个模块已经集成到了初始化固件中，也就是说我们可以直接在 main.py 导入模块并调用即可！

单总线模块（onewire）和 ds18x20 模块说明如下：

构造函数
<code>ow=onewire.OneWire(machine.Pin(id))</code>
构建单总线对象。 <code>id</code> :引脚编号；
使用方法
<code>ow.scan()</code>
扫描总线上的设备。返回设备地址，支持多设备同时挂载。
<code>ow.reset()</code>
总线设备复位。
<code>ow.readbyte()</code>
读 1 个字节。
<code>ow.writebyte(0x12)</code>
写入 1 个字节。
<code>ow.write('123')</code>
写入多个字节。
<code>ow.select_rom(b'12345678')</code>
根据 ROM 编号选择总线上指定设备。

图 5-6 单总线对象

构造函数
<code>ds=ds18x20.DS18X20(ow)</code>
构建 DS18B20 传感器对象。 <code>ow</code> :定义好的单总线对象；
使用方法
<code>ds.scan()</code>
扫描总线上的设备。返回设备地址，支持多设备同时挂载。
<code>ds.convert_temp()</code>
温度转换。
<code>ds.read_temp(rom)</code>
获取温度值。 <code>rom</code> : 表示对应的设备号。

图 5-7 DS18B20 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：

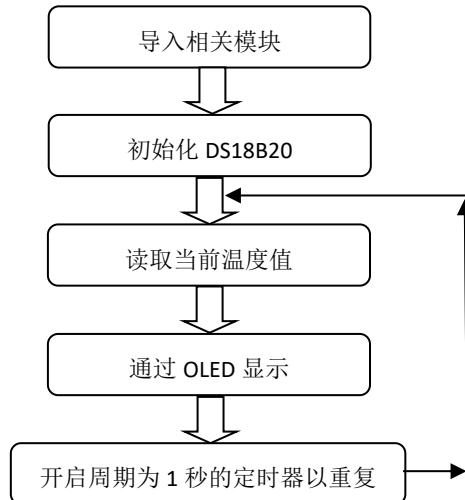


图 5-8 代码编写流程图

实验参考代码：

```
...
实验名称: 温度传感器 DS18B20
版本: v1.0
日期: 2019.7
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 machine import Pin,I2C,Timer
from ssd1306 import SSD1306_I2C
import onewire,ds18x20

#初始化相关模块
```

```

i2c = I2C(sda=Pin(13), scl=Pin(14))
oled = SSD1306_I2C(128, 64, i2c, addr=0x3c)

#初始化 DS18B20

ow= onewire.OneWire(Pin(4)) #使能单总线
ds = ds18x20.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

def temp_get(tim):
    ds.convert_temp()
    temp = ds.read_temp(rom[0]) #温度显示,rom[0]为第 1 个 DS18B20

    #OLED 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()

#开启 RTOS 定时器，编号为-1
tim = Timer(-1)
#定时器周期为 1000ms
tim.init(period=1000, mode=Timer.PERIODIC,callback=temp_get)

```

实验使用到 OLED 显示屏，记得同时将 ssd1306.py 文件拷贝到设备中。

- 实验结果：

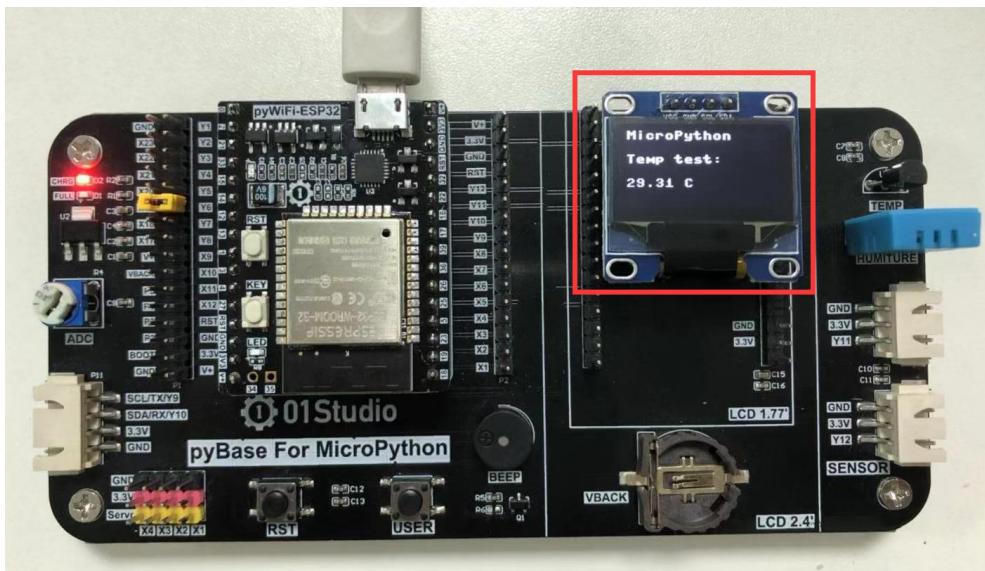


图 5-9 温度检测实验

- 实验拓展：

pyBase 开发底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 pyBase 右侧上面的传感器母座，其连接到 pyBase 的“Y11”引脚，也就是对应 pyWiFi-ESP32 的引脚 22。所以只要将原程序代码：`ow= onewire.OneWire(Pin(14))` 改成 `ow= onewire.OneWire(Pin(22))`，即可驱动外接的 DS18B20。

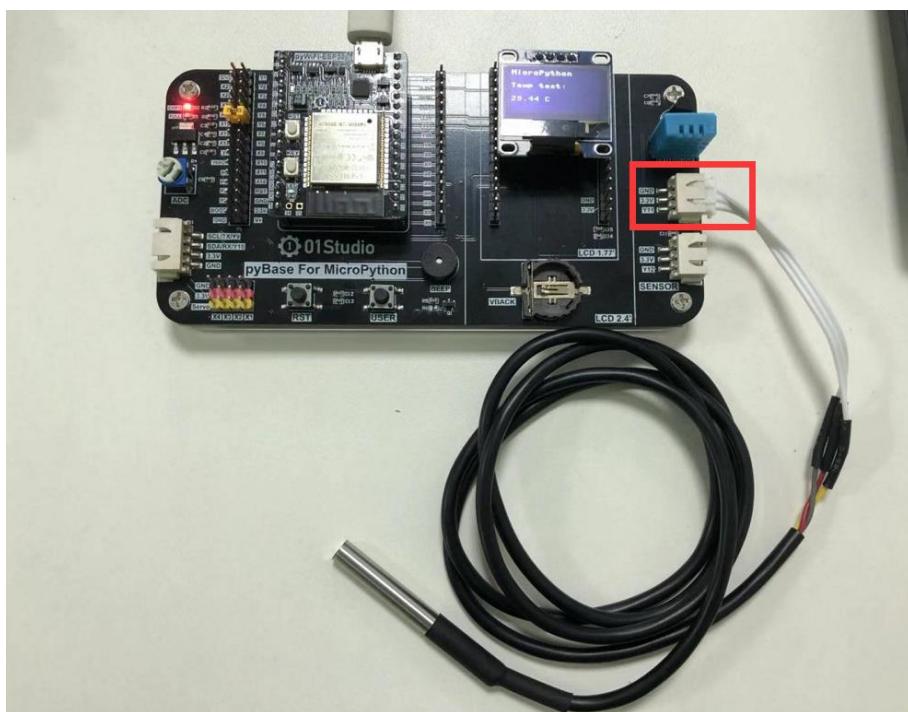


图 5-10 外接温度传感器

我们将金属探头放在水里面，可以见到测试到的温度。

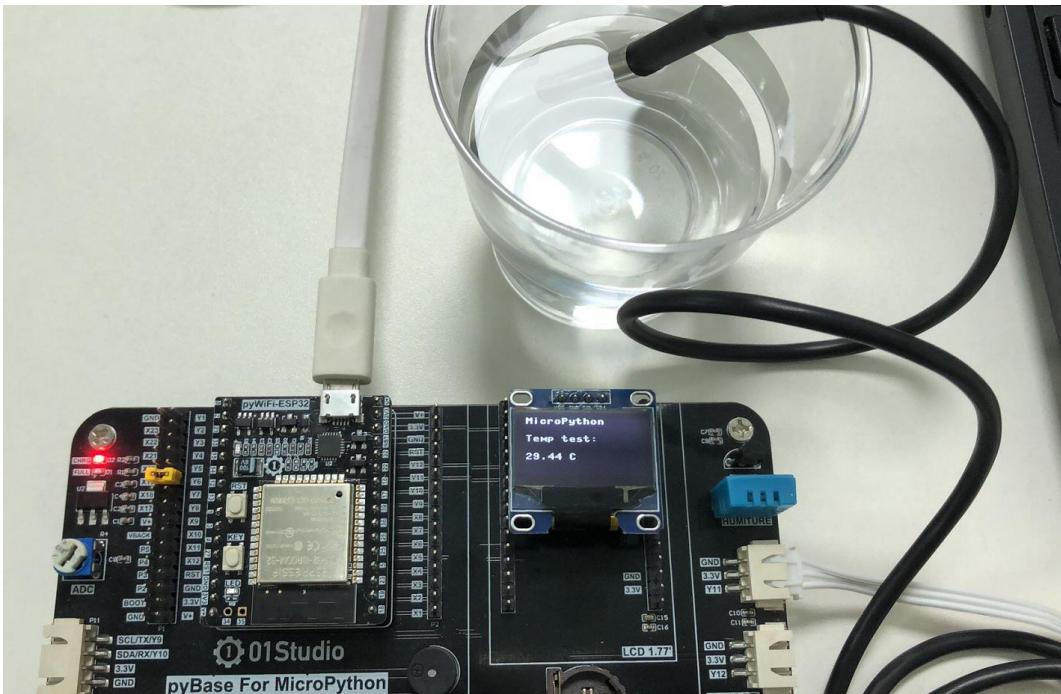


图 5-11 实现金属探头 DS18B20 测量

● 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。

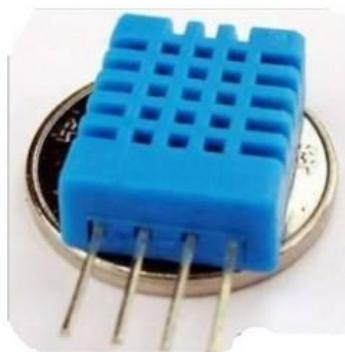


图 5-12 DHT11 温湿度传感器

● 实验平台：

pyWiFi-ESP32 和 pyBase 开发底板。DHT11 温湿度传感器位于右上方。

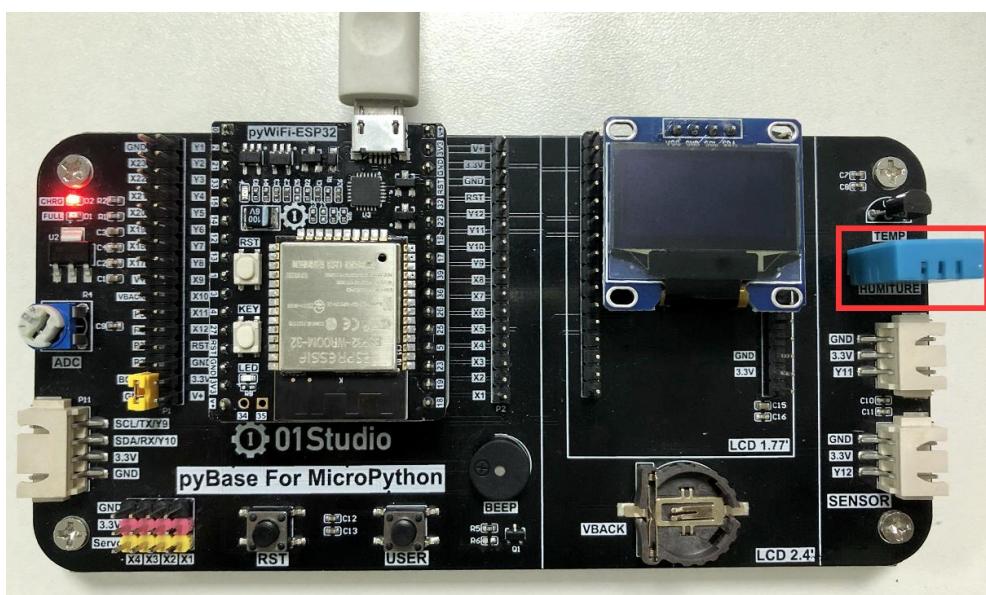


图 5-13 温湿度传感器 DHT11 位于右上方

- 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。

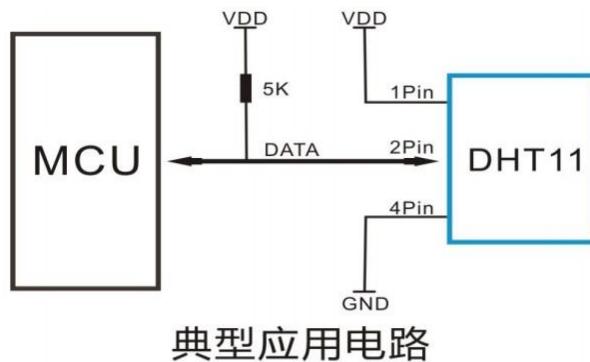


图 5-14 DHT11 应用电路

我们来看看 DHT11 在开发板上的接线图:

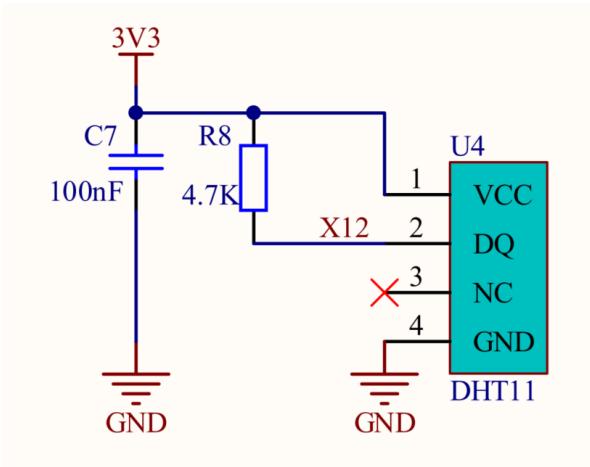


图 5-15 DHT11 接线图

可以看到 DHT11 连接到 pyBase 的 ‘X12’ 引脚，也就是连接到 pyWiFi-ESP32 的引脚 27，如下图所示：

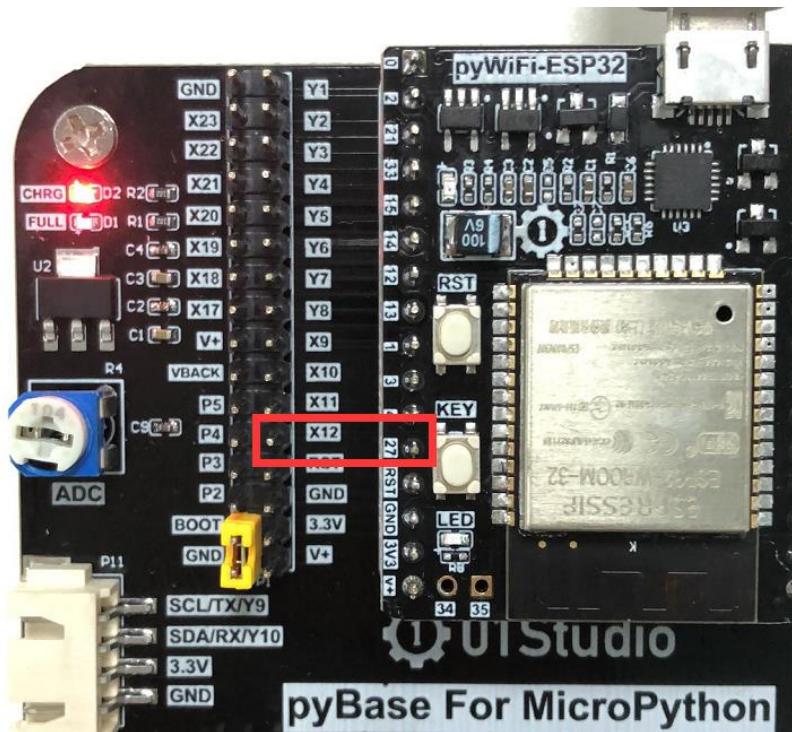


图 5-16 引脚 27 连接到底板的 DHT11 传感器

因此可以针对引脚 27 编程来驱动 DHT11 传感器，模块文件是 `dht.py`，如果你学习过前面基于 STM32 平台应该不陌生。而对于 ESP32，这个模块已经集成到了初始化固件中，也就是说我们可以直接在 `main.py` 导入模块并调用即可。函数模块说明如下：

构造函数
<code>d = dht.DHT11(machine.Pin(id))</code>
构建 DHT11 传感器对象。 <code>id</code> :传感器所连接的引脚；
使用方法
<code>d.measure()</code>
测量温湿度。
<code>d.temperature()</code>
获取温度值。
<code>d.humidity()</code>
获取湿度值。

表 5-1 DHT11 对象

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

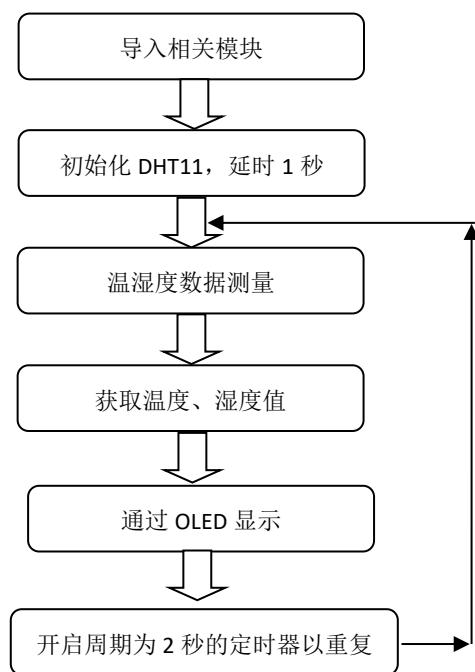


图 5-17 代码编写流程图

实验参考代码：

```
...  
实验名称: 湿湿度传感器 DHT11  
版本: v1.0  
日期: 2019.7  
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 machine import Pin,I2C,Timer  
from ssd1306 import SSD1306_I2C  
import dht,time
```

```
#初始化相关模块

i2c = I2C(sda=Pin(13), scl=Pin(14))

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


#创建 DTH11 对象

d = dht.DHT11(Pin(27)) #传感器连接到引脚 15

time.sleep(1) #首次启动停顿 1 秒让传感器稳定


def dht_get(tim):

    d.measure()          #温湿度采集


    #OLED 显示温湿度

    oled.fill(0) #清屏背景黑色

    oled.text('01Studio', 0, 0)

    oled.text('DHT11 test:',0,15)

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

    oled.text(str(d.humidity())+' %',48,40) #湿度显示

    oled.show()


#开启 RTOS 定时器，编号为-1，周期 2 秒

tim = Timer(-1)

tim.init(period=2000, mode=Timer.PERIODIC,callback=dht_get)
```

- 实验结果：

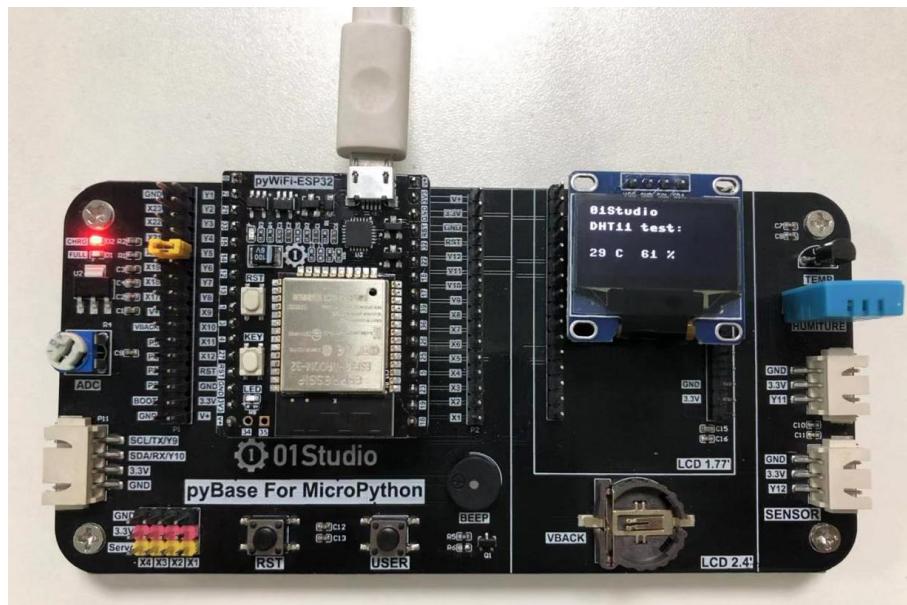


图 5-18 DHT11 实验结果

- 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：

pyWiFi-ESP32 开发套件和人体感应传感器模块。

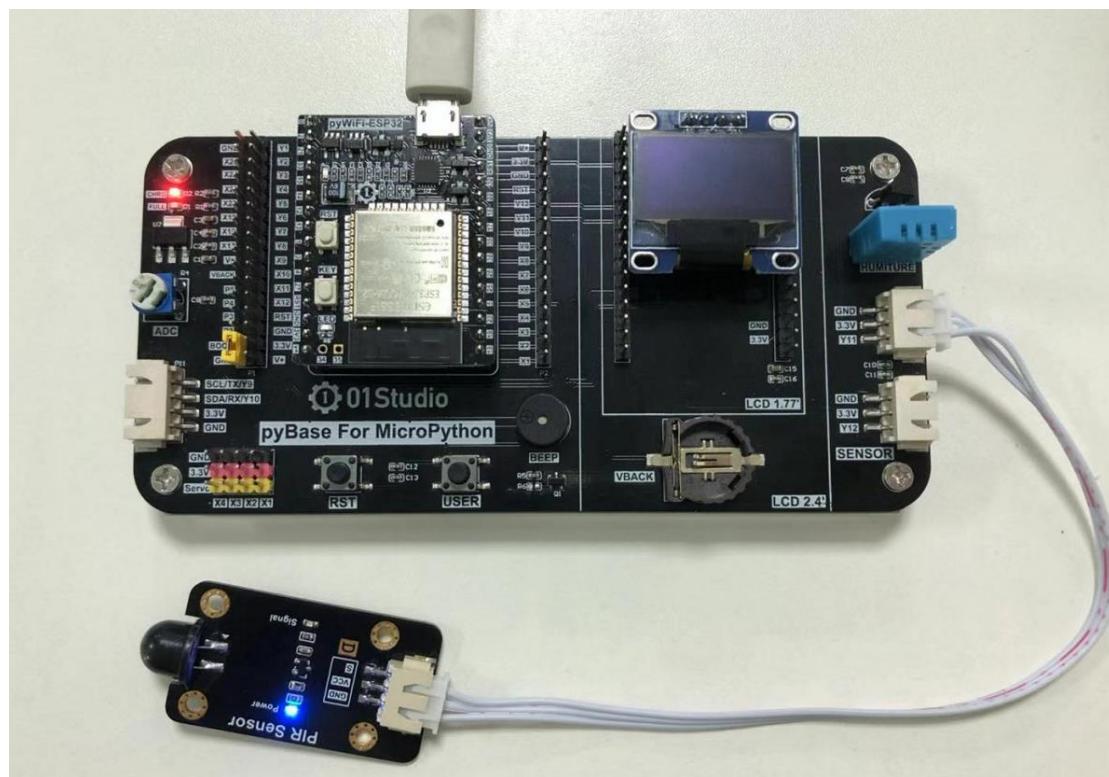


图 5-19 开发套件和传感器接线

● 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：

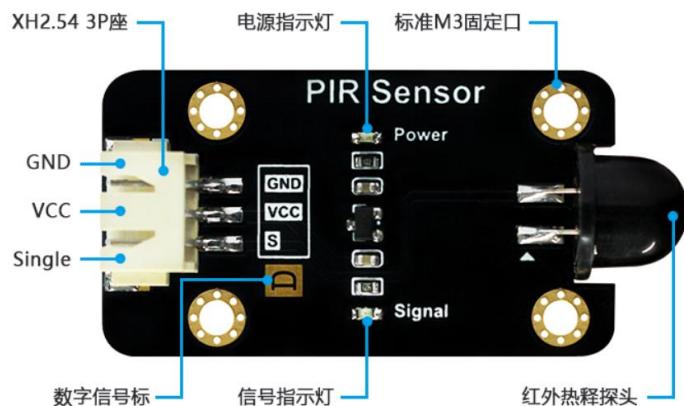


图 5-20 人体感应传感器模块

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

图 5-21 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：

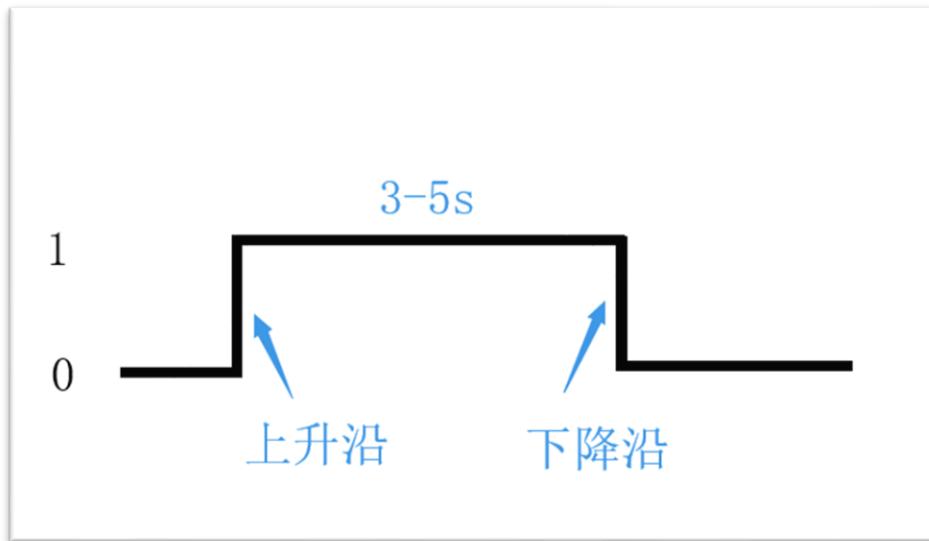


图 5-22 人体感应传感器电平变化

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 pyBase 的“Y11”引脚。也就是 pyWiFi-ESP32 的‘22’引脚。编程方法可以是当“22”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。（有关中断的介绍可以看上一章“外部中断”内容。）

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：

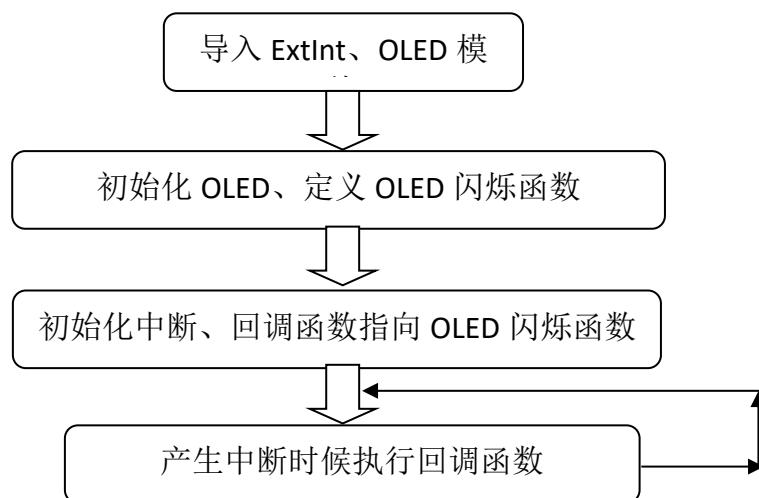


图 5-23 代码编写流程

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：2019.8
作者：01Studio (www.01studio.org)
说明：人体红外感应传感器应用
...

import time

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
i2c = I2C(sda=Pin(13), scl=Pin(14))

#OLED 显示屏初始化：128*64 分辨率，OLED 的 I2C 地址是 0x3c
oled = SSD1306_I2C(128, 64, i2c, addr=0x3c)

Human=Pin(22,Pin.IN,Pin.PULL_UP) #构建人体红外对象

#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 fun(Human): #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  
time.sleep_ms(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  
time.sleep_ms(500)
```

Human.irq(fun,Pin.IRQ_RISING) # 定义中断，上升沿触发

● 实验结果：

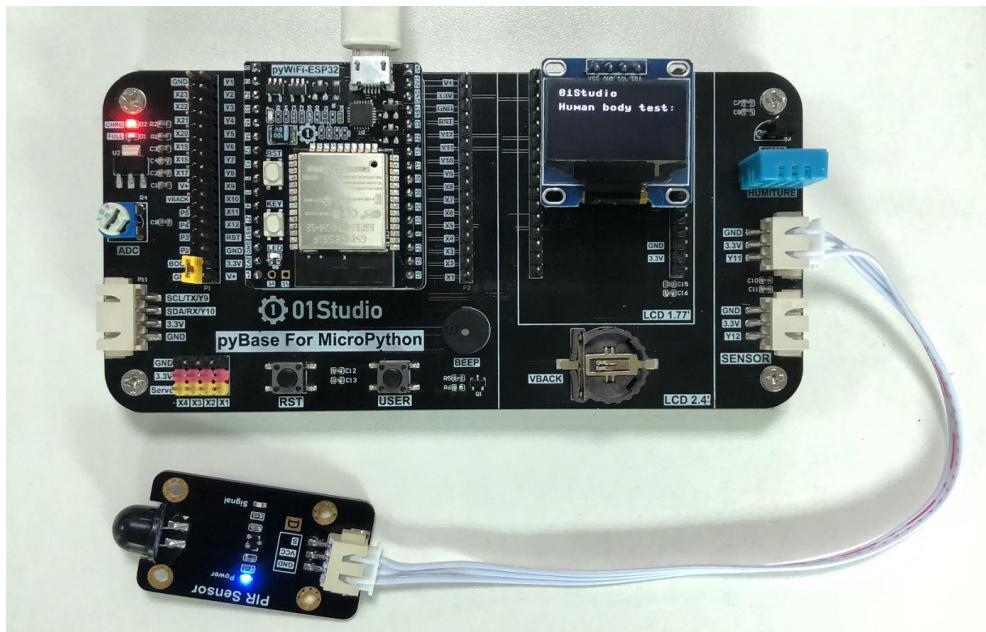


图 5-24 没有检测到到人体

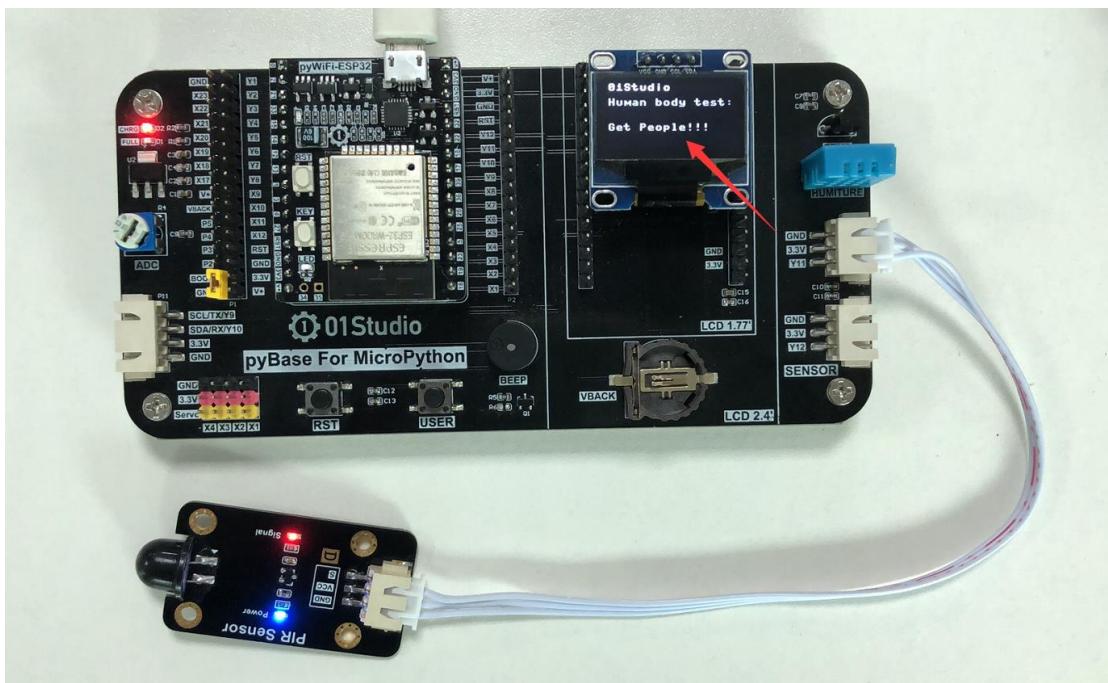


图 5-25 检测到人体，OLED 闪烁

● 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：**

pyWiFi-ESP32 开发套件开发套件和光敏传感器模块。

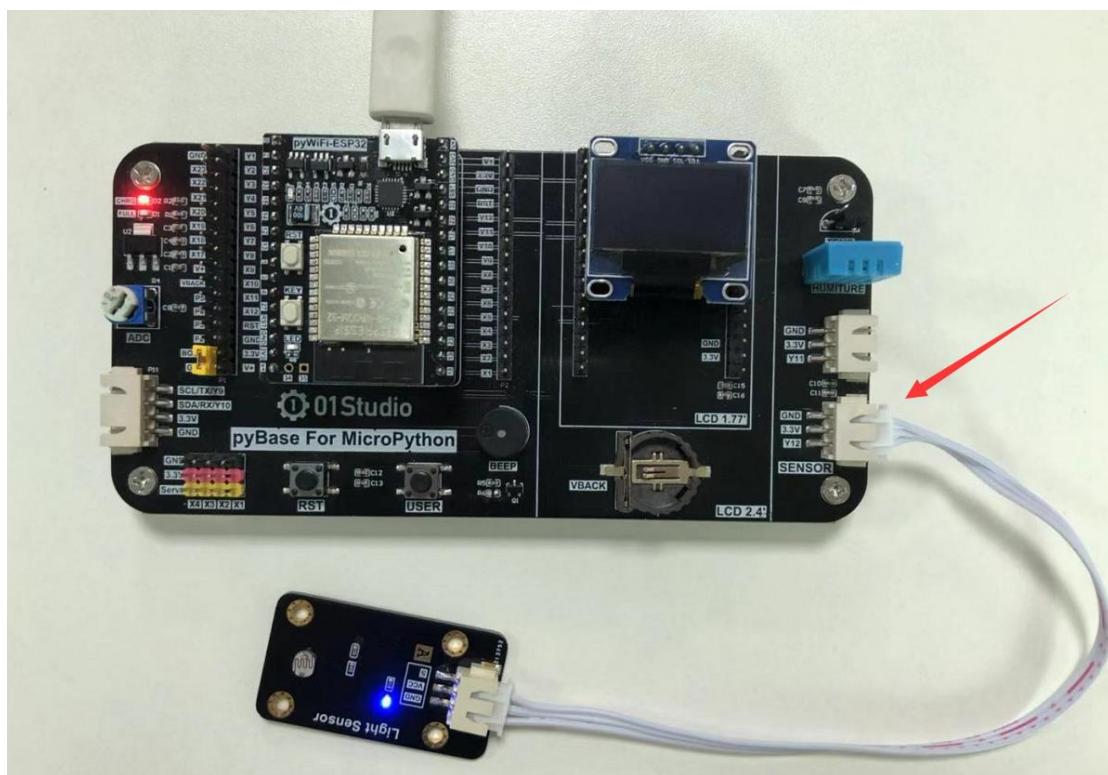


图 5-26 开发套件和光敏传感器接线

- **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：

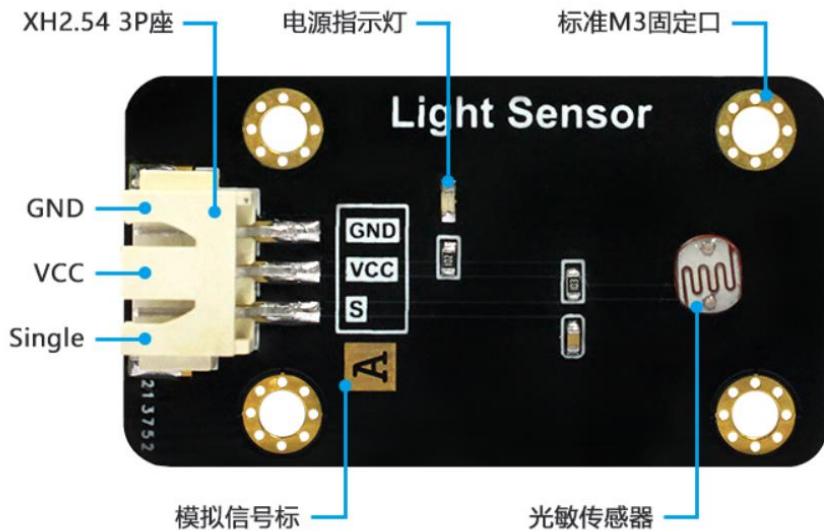


图 5-27 光敏传感器模块

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 5-2

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 2，对应 pyBase 的引脚是“Y12”，即 pyWiFi-ESP32 的‘32’引脚。我们使用内部衰减器将测量电压量程从 0-1V 增加至 0-3.3V。（具体参考上一章“ADC”部分章节内容。）

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：

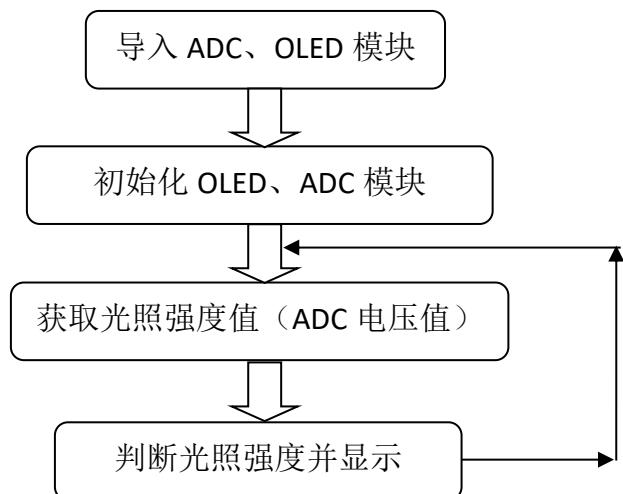


图 5-28 代码编写流程

参考程序代码如下：

```
...
实验名称: 光敏传感器
版本: v1.0
日期: 2019.8
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

from machine import Pin,I2C,ADC,Timer
from ssd1306 import SSD1306_I2C

#初始化相关模块
```

```

i2c = I2C(sda=Pin(13), scl=Pin(14))
oled = SSD1306_I2C(128, 64, i2c, addr=0x3c)

#初始化 ADC, Pin=36, 11DB 衰减, 测量电压 0-3.3V
Light = ADC(Pin(32))
Light.atten(ADC.ATTN_11DB)

#中断回调函数
def fun(tim):

    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 0-3.3V, ('%.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()

#开启 RTOS 定时器
tim = Timer(-1)

tim.init(period=1000, mode=Timer.PERIODIC, callback=fun) #周期 1s
```

● 实验结果：

用手遮挡住光敏传感器，可以见到 OLED 显示光线强度是弱 Weak:

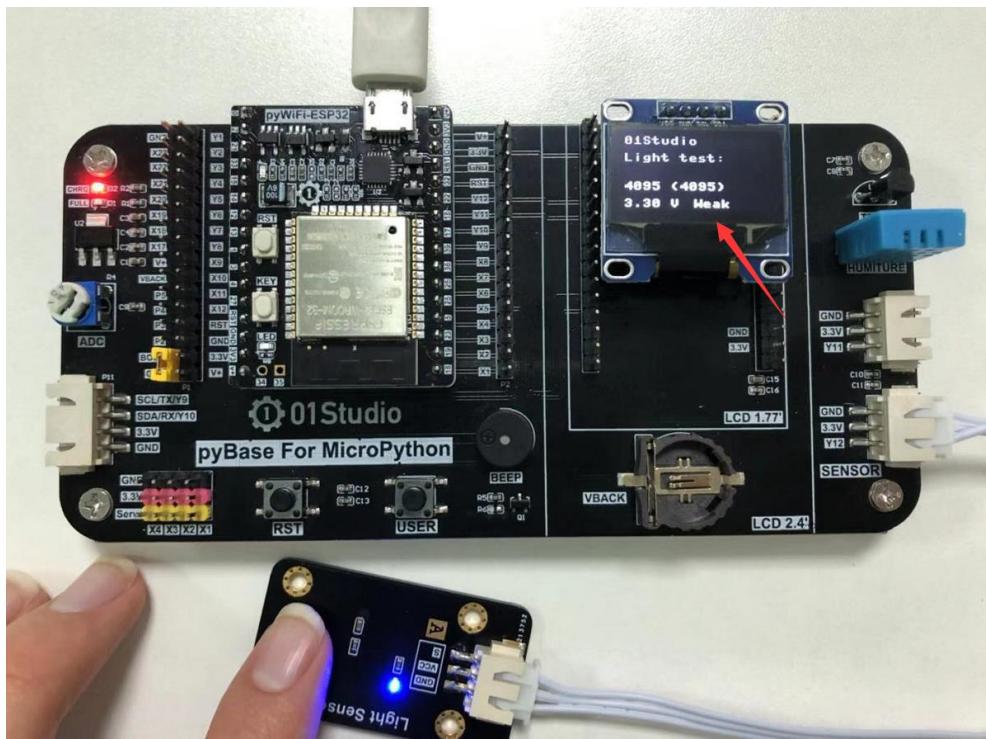


图 5-29 光线强度弱

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:

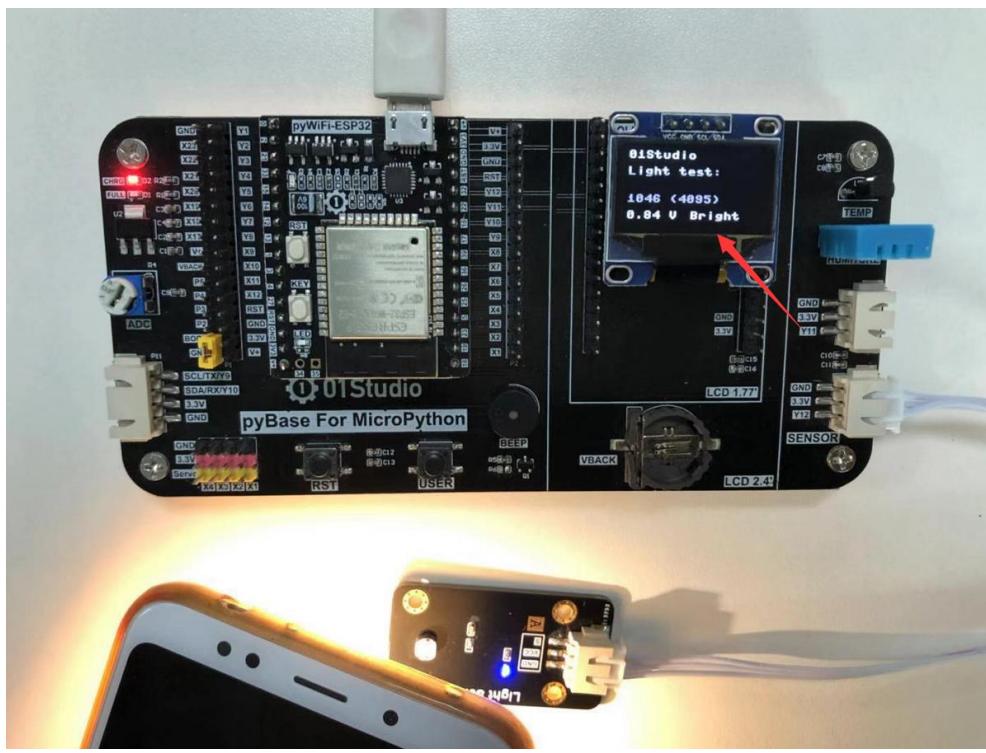


图 5-30 光线强度强

● 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：**

pyWiFi-ESP32 开发套件和土壤湿度传感器模块。

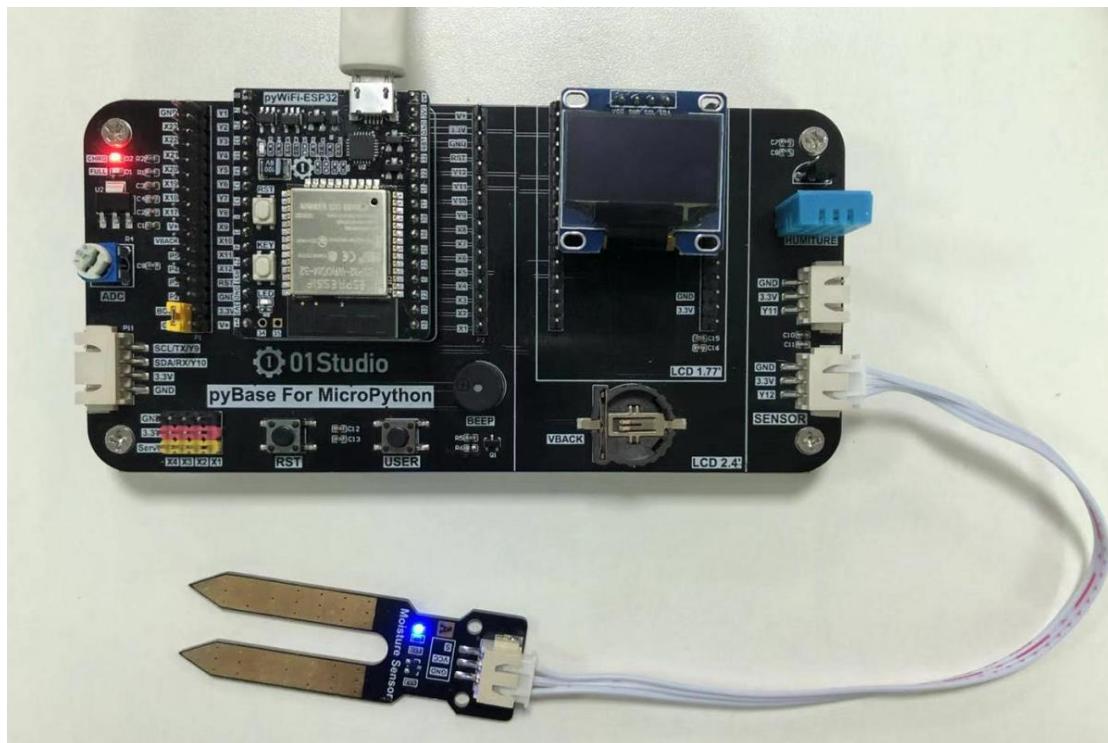


图 5-31 开发套件和传感器连接图

- **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：

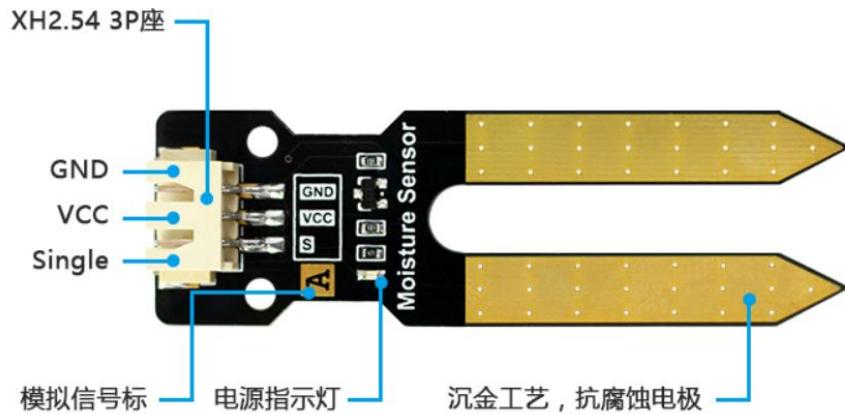


图 5-32 土壤湿度传感器模块

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 5-3

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 2，对应 pyBase 的引脚是“Y12”，也就是 pyWiFi-ESP32 的‘32’引脚，ADC 使用内部 11DB 衰减器，将量程增至 0-3.3V，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：

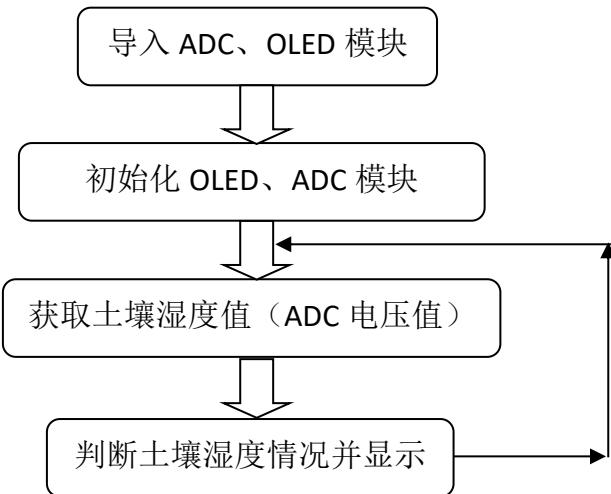


图 5-33 代码编写流程

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：2019.8
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

from machine import Pin,I2C,ADC,Timer
from ssd1306 import SSD1306_I2C

#初始化相关模块
i2c = I2C(sda=Pin(13), scl=Pin(14))
oled = SSD1306_I2C(128, 64, i2c, addr=0x3c)

```

```
#初始化 ADC,Pin 是 32, 开启 11DB 衰减, 量程增至 0-3.3V
Soil = ADC(Pin(32))
Soil.atten(ADC.ATTN_11DB)

#中断回调函数
def fun(tim):

    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 0-3.3V, ('%.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()

#开启 RTOS 定时器
tim = Timer(-1)

tim.init(period=1000, mode=Timer.PERIODIC, callback=fun) #周期 1s
```

● 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:

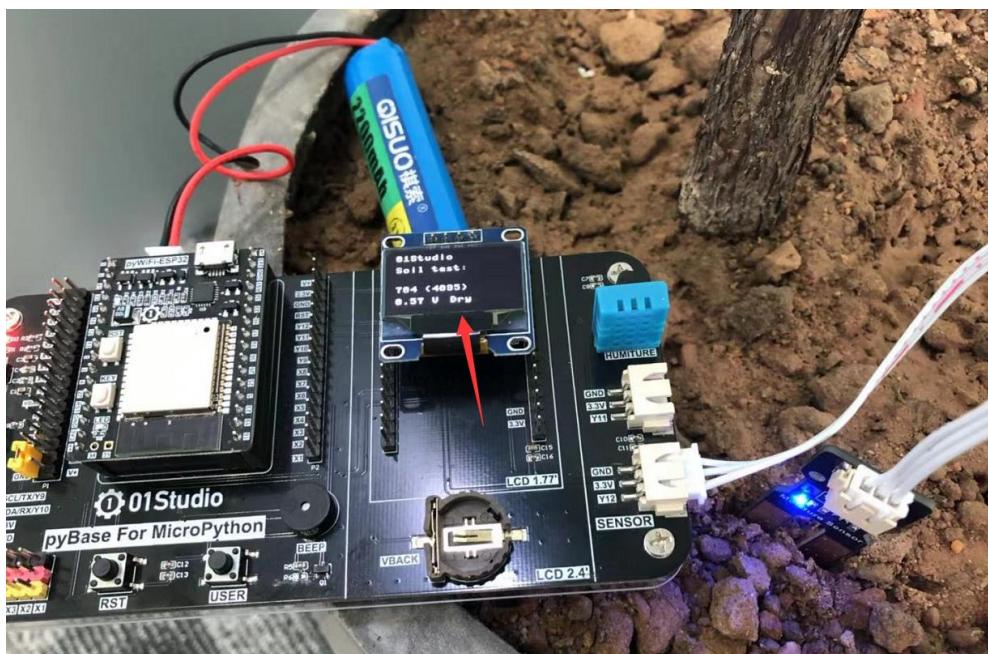


图 5-34 干燥的土壤

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 5-35 浇水

可以看到 OLED 显示湿润-Wet:



图 5-36 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：**

pyWiFi-ESP32 开发套件和水位传感器模块。

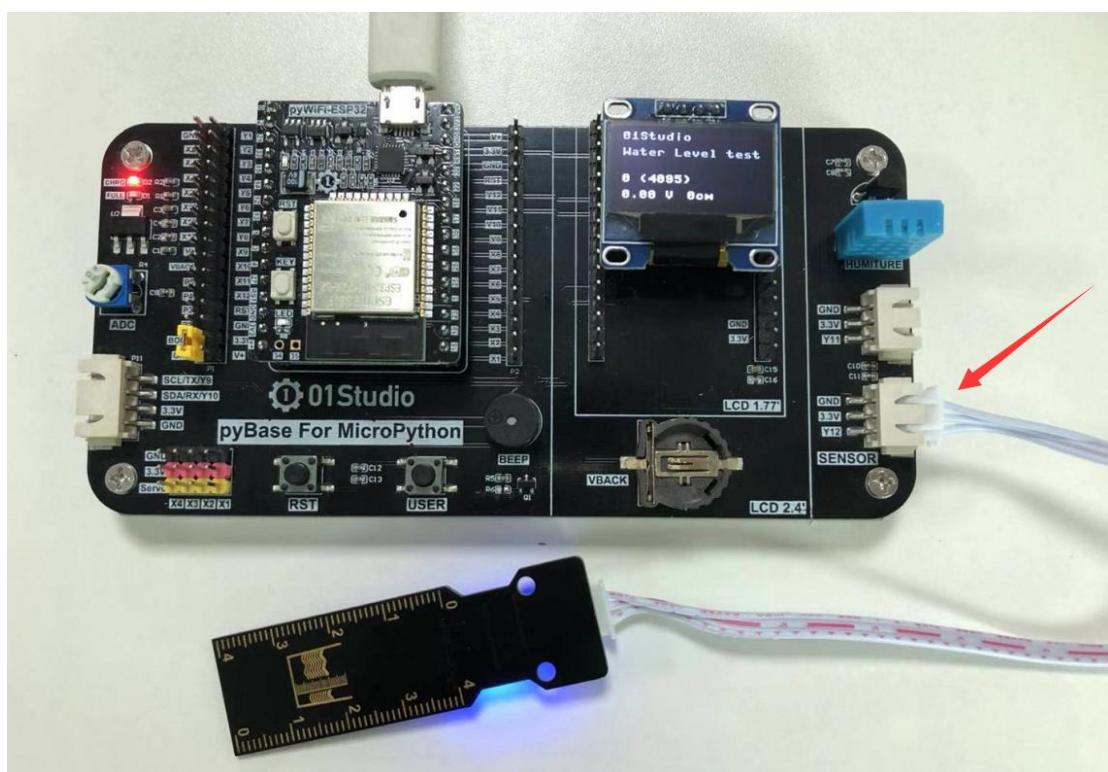


图 5-37 开发套件和水位传感器接线图

- **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：

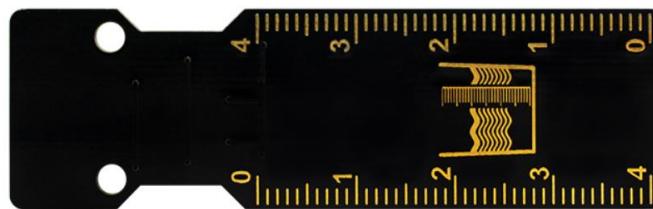
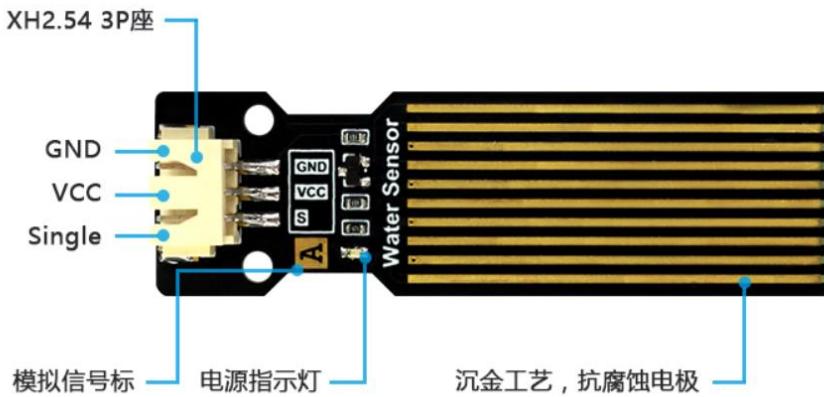


图 5-38 水位传感器模块正反面

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。

图 5-39

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（蒸馏水实测 0-1V）。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 2，对应 pyBase 的引脚是“Y12”。也就是 pyWiFi-ESP32 的引脚‘32’。ADC 使用默认量程 0-1V, 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:【0-1300】，1cm:【1300-2300】，2cm:【2300-3300】，3cm:【3300-3800】，4cm: 【>3800】。

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：

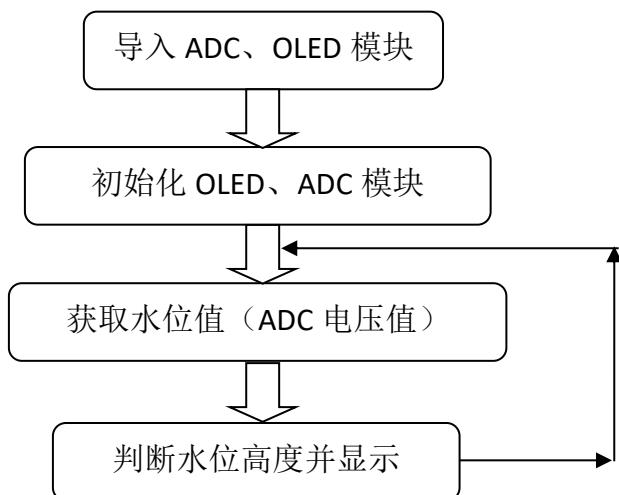


图 5-40 代码编写流程

参考程序代码如下：

```
...
实验名称: 水位传感器
版本: v1.0
日期: 2019.8
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
from machine import Pin,I2C,ADC,Timer
from ssd1306 import SSD1306_I2C
```

```

#初始化相关模块

i2c = I2C(sda=Pin(13), scl=Pin(14))
oled = SSD1306_I2C(128, 64, i2c, addr=0x3c)

#初始化 ADC, Pin 是引脚 32, 默认量程 0-1V
Water_level = ADC(Pin(32))

#中断回调函数

def fun(tim):

    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 0-1V, ('%.2f'%) 表示保留 2 位小数
    oled.text(str('%.2f'%(value/4095))+ ' V', 0, 55)

    #判断水位, 分 5 档显示, 0-4cm
    if 0 <= value <=1300:
        oled.text('0cm', 60, 55)

    if 1300 < value <= 2300:
        oled.text('1cm', 60, 55)

    if 2300 < value <= 3300:

```

```

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

if 3300 < value <= 3800:
    oled.text('3cm', 60, 55)

if 3800 <= value:
    oled.text('4cm', 60, 55)

oled.show()

#开启 RTOS 定时器
tim = Timer(-1)
tim.init(period=1000, mode=Timer.PERIODIC, callback=fun) #周期 1s

```

● 实验结果：

当没插入水中时，OLED 显示当前水位为 0cm:

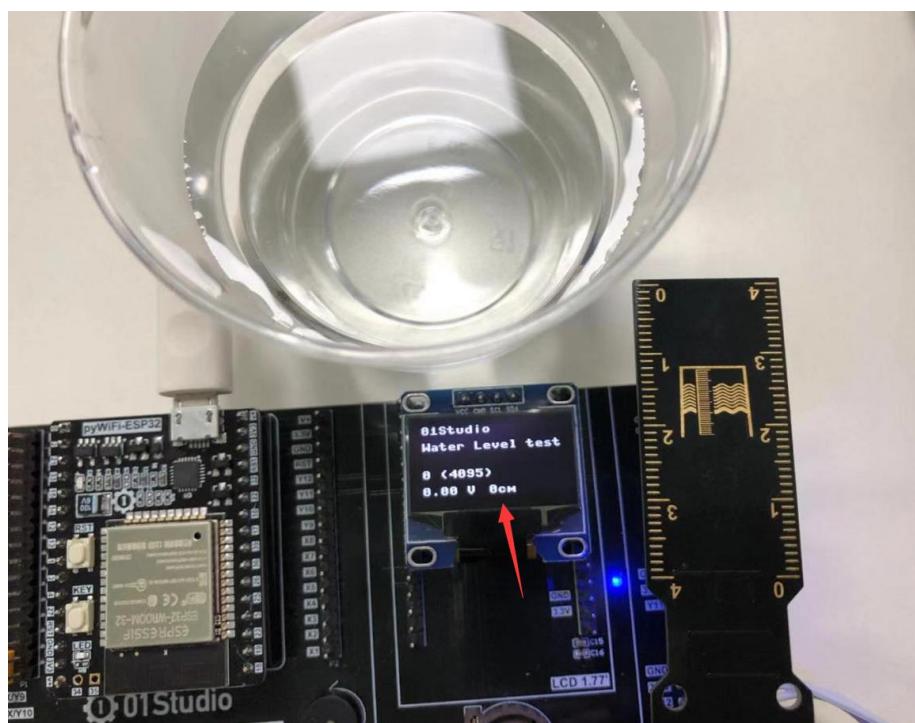


图 5-41

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。**)

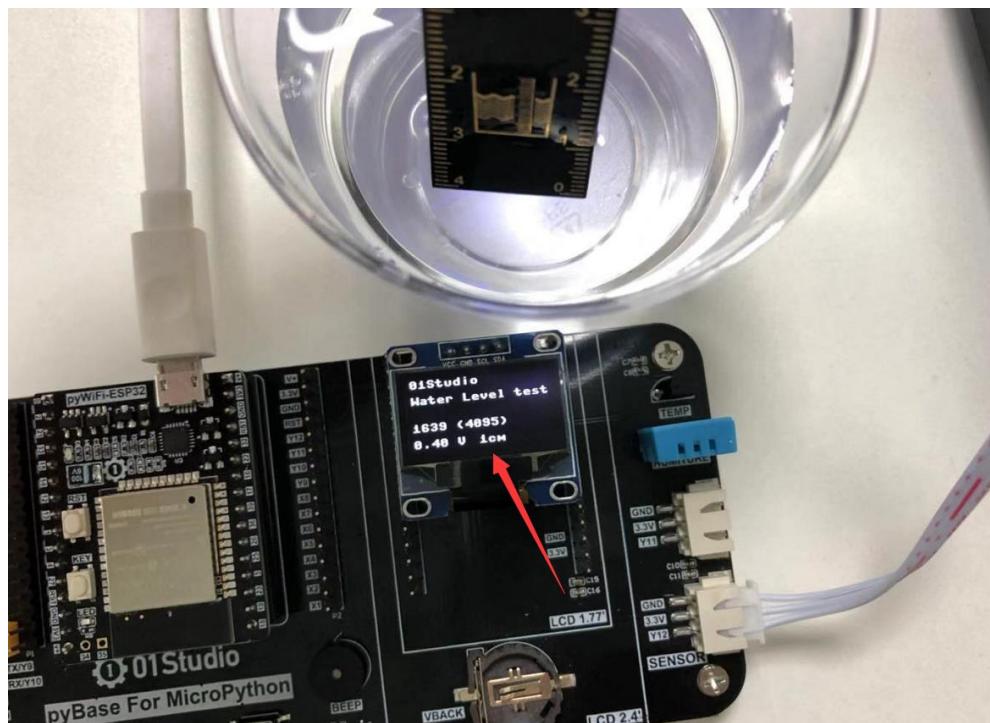


图 5-42 水位接近 1cm

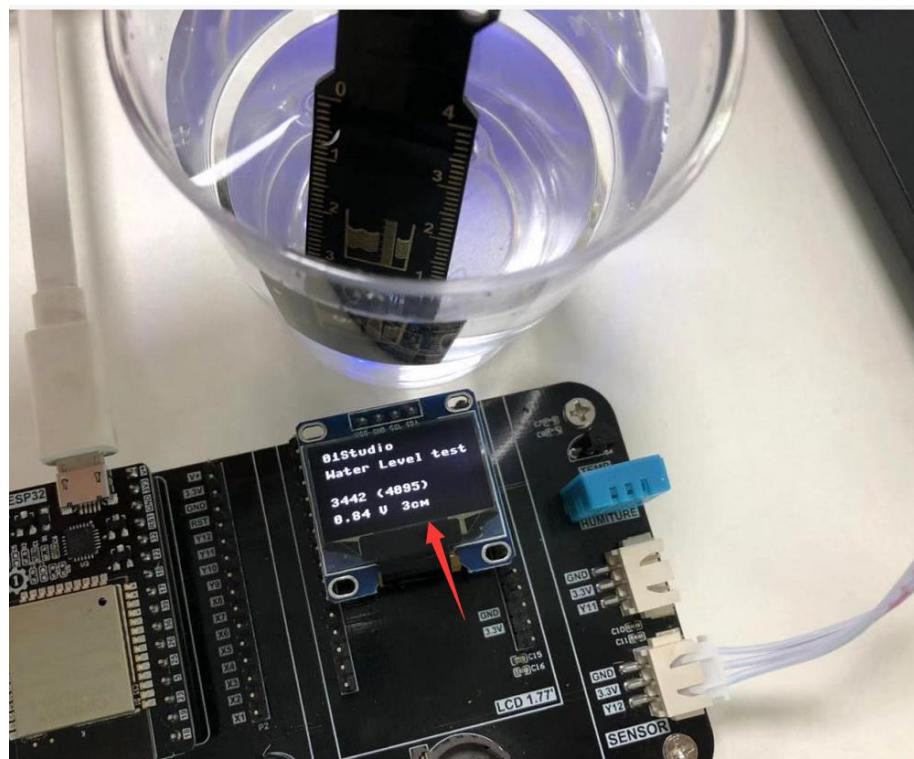


图 5-43 水位接近 3cm

● 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：

pyWiFi-ESP32 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。

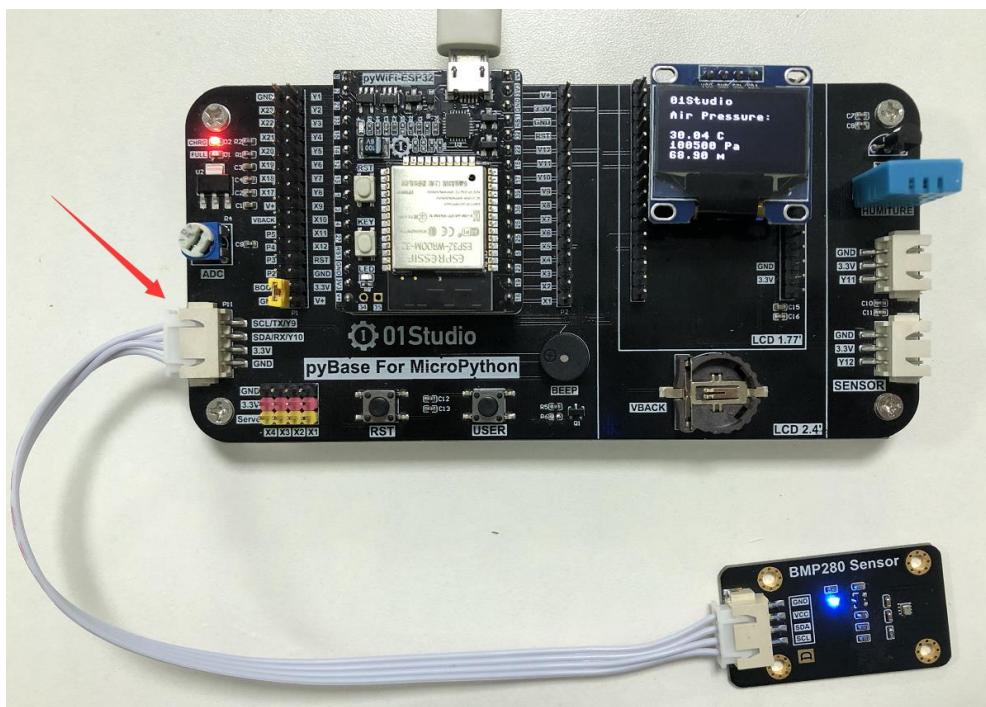


图 5-44 开发套件和大气压强传感器接线方法

● 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：

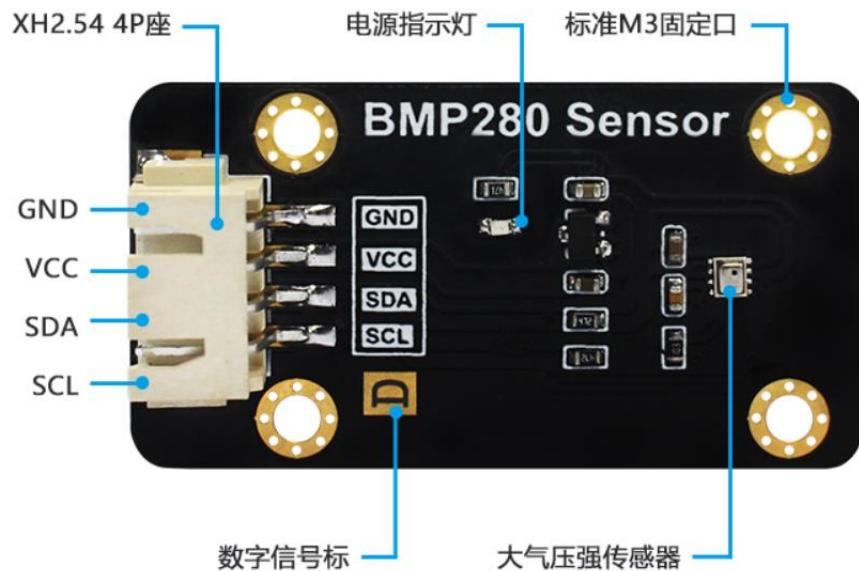


图 5-45 大气压强传感器模块

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

图 5-46

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 pyBase 左下角的 I2C 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，P₀=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
使用方法	说明
<code>bmp280.BMP280(I2C)</code>	定义传感器的 I2C 接口。
<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 5-4 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：

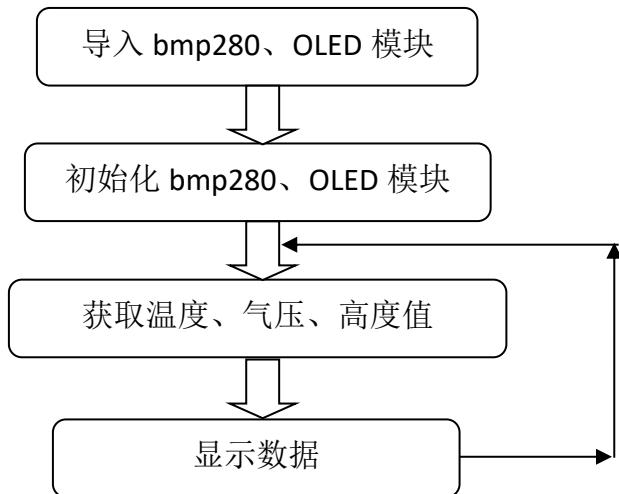


图 5-47 代码编写流程

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: 2019.8  
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 time  
import bmp280
```

```

from machine import Pin,I2C,Timer
from ssd1306 import SSD1306_I2C

#初始化 OLED
i2c = I2C(sda=Pin(13), scl=Pin(14))
oled = SSD1306_I2C(128, 64, i2c, addr=0x3c)

#初始化 BMP280, 定义第二个 I2C 接口 i2c2 用于连接 BMP280
i2c2 = I2C(sda=Pin(16), scl=Pin(17))
BMP = bmp280.BMP280(i2c2)

#中断回调函数
def fun(tim):

    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()

#开启 RTOS 定时器
tim = Timer(-1)

tim.init(period=1000, mode=Timer.PERIODIC, callback=fun) #周期 1s

```

● 实验结果：

程序烧写完复位后可以见到实验结果。

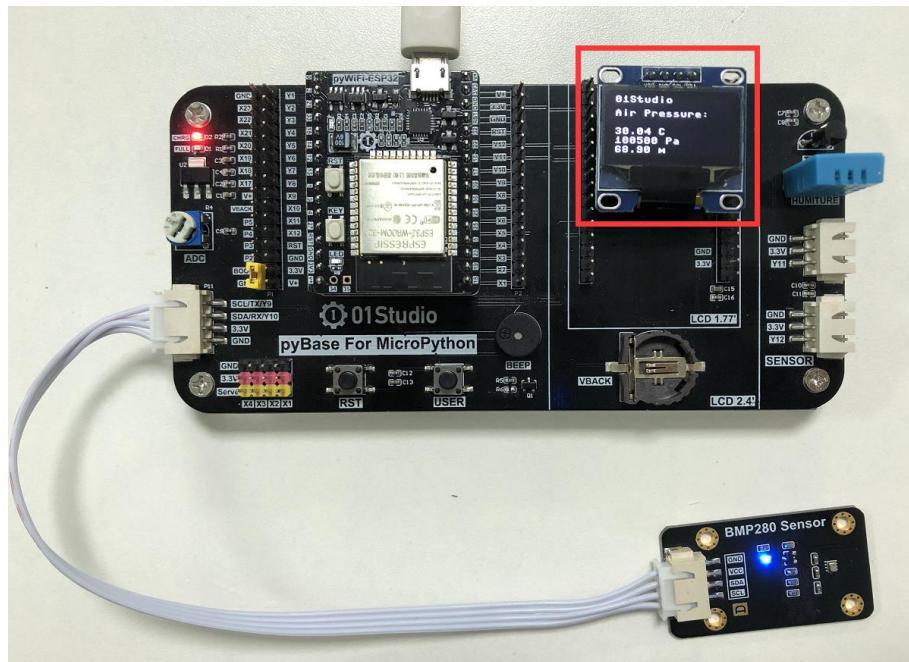


图 5-48 气压测量

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。

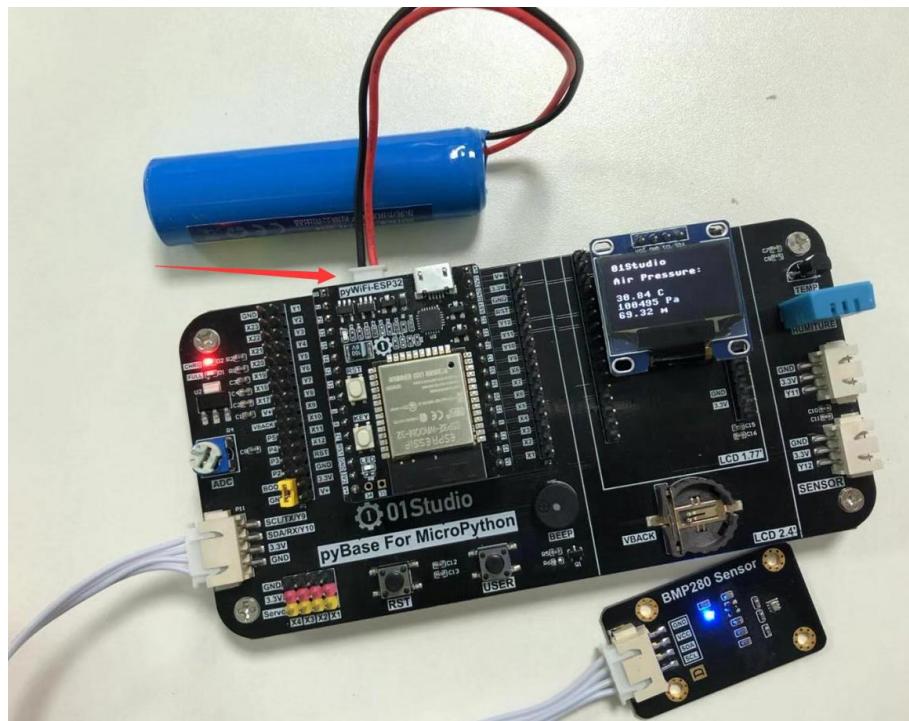


图 5-49 锂电池供电

● 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：**

pyWiFi-ESP32 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。

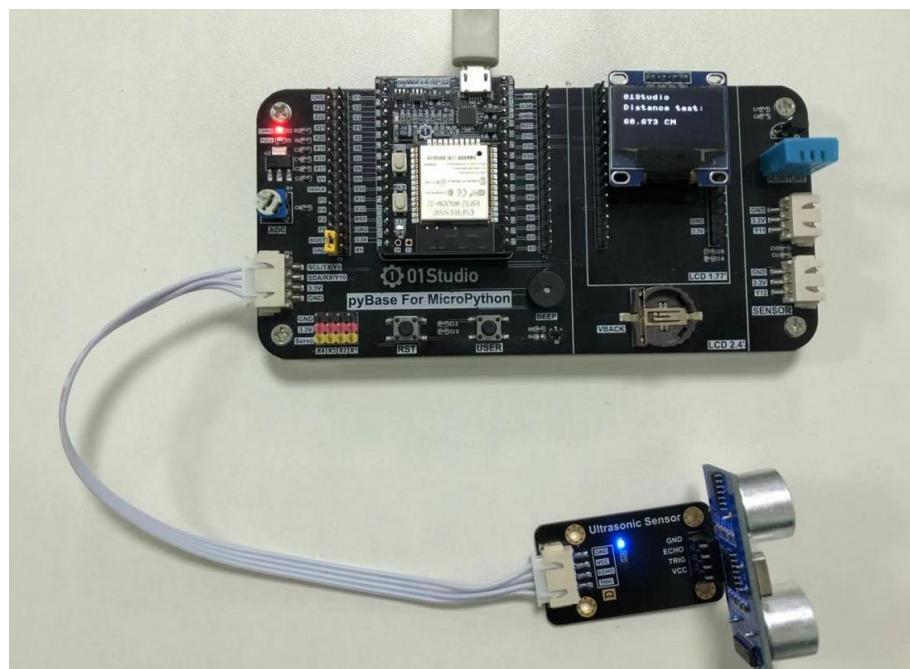


图 5-50 开发套件与传感器连接

- **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：

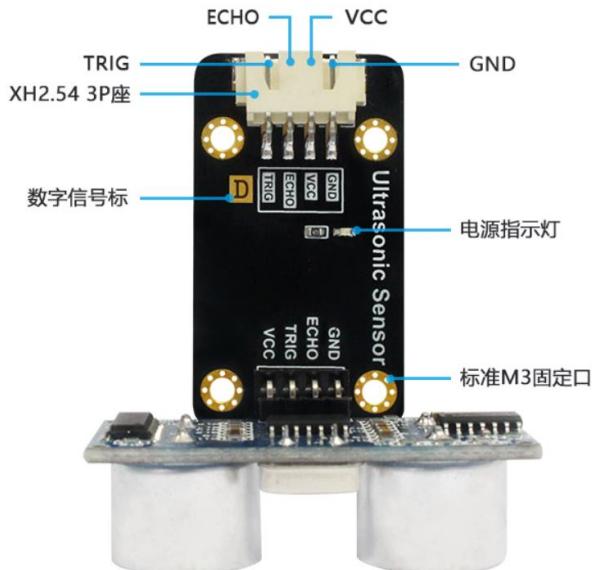


图 5-51 超声波模块

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 5-5

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。

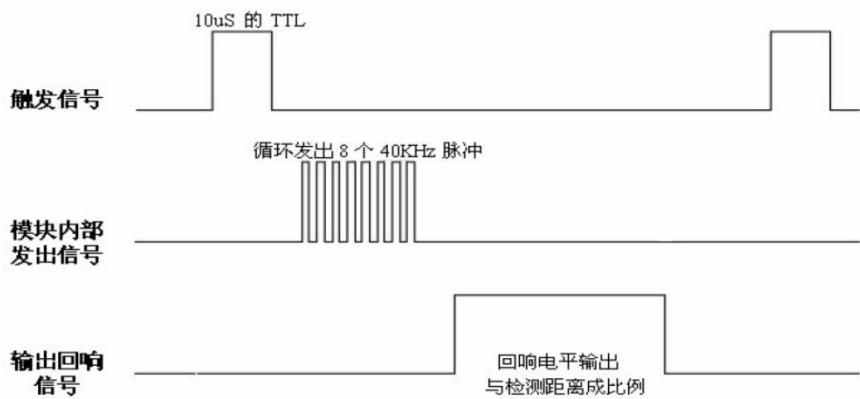


图 5-52 超声波传感器 HCSR04 时序图

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 5-6 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：

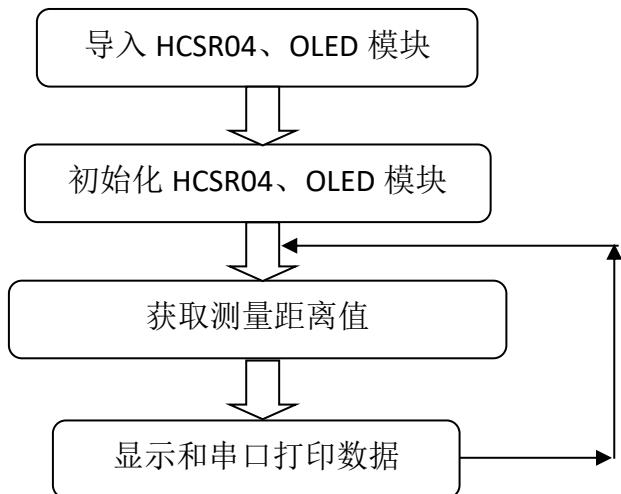


图 5-53 代码编写流程

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: 2019.8
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 machine import Pin,I2C,Timer
from ssd1306 import SSD1306_I2C

#初始化 OLED
i2c = I2C(sda=Pin(13), scl=Pin(14))
oled = SSD1306_I2C(128, 64, i2c, addr=0x3c)

```

```
#初始化接口 trig=17,echo=16
trig = Pin(17,Pin.OUT)
echo = Pin(16,Pin.IN)
HC=HCSR04(trig,echo)

#中断回调函数
def fun(tim):

    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')

#开启 RTOS 定时器
tim = Timer(-1)
tim.init(period=1000, mode=Timer.PERIODIC, callback=fun) #周期 1s
```

● 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。

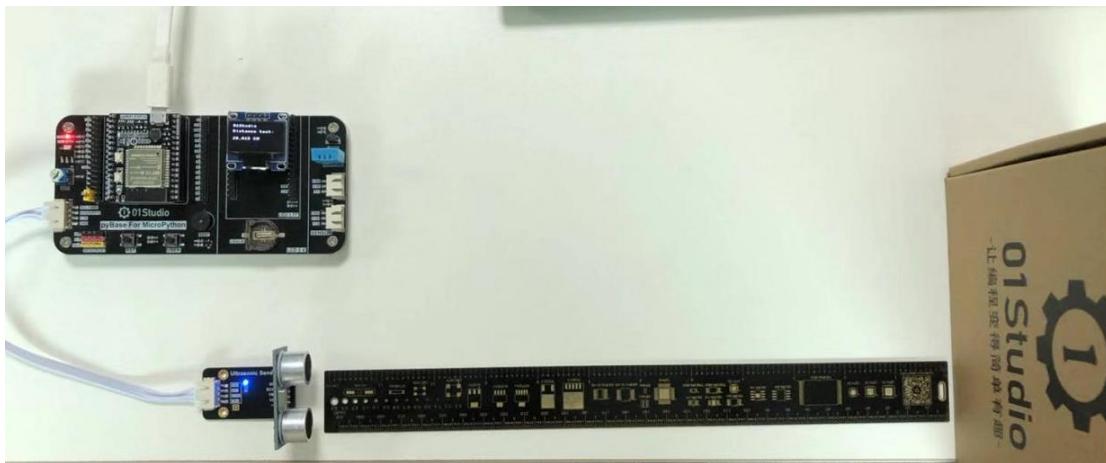


图 5-54 障碍物距离为 30cm

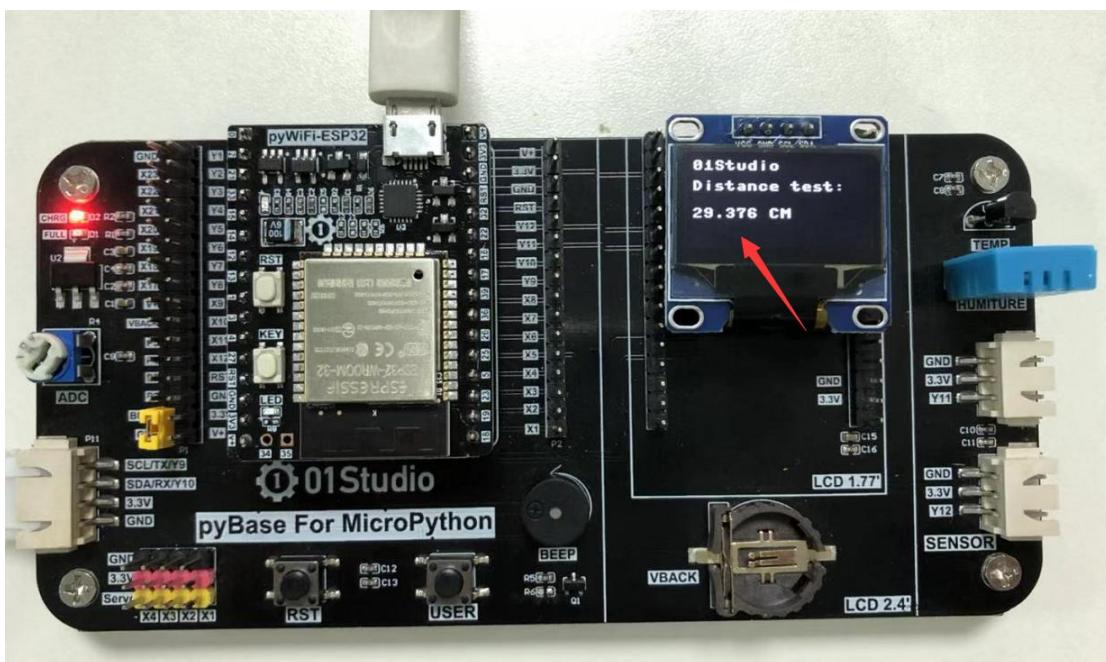


图 5-55 实验结果约为 30cm

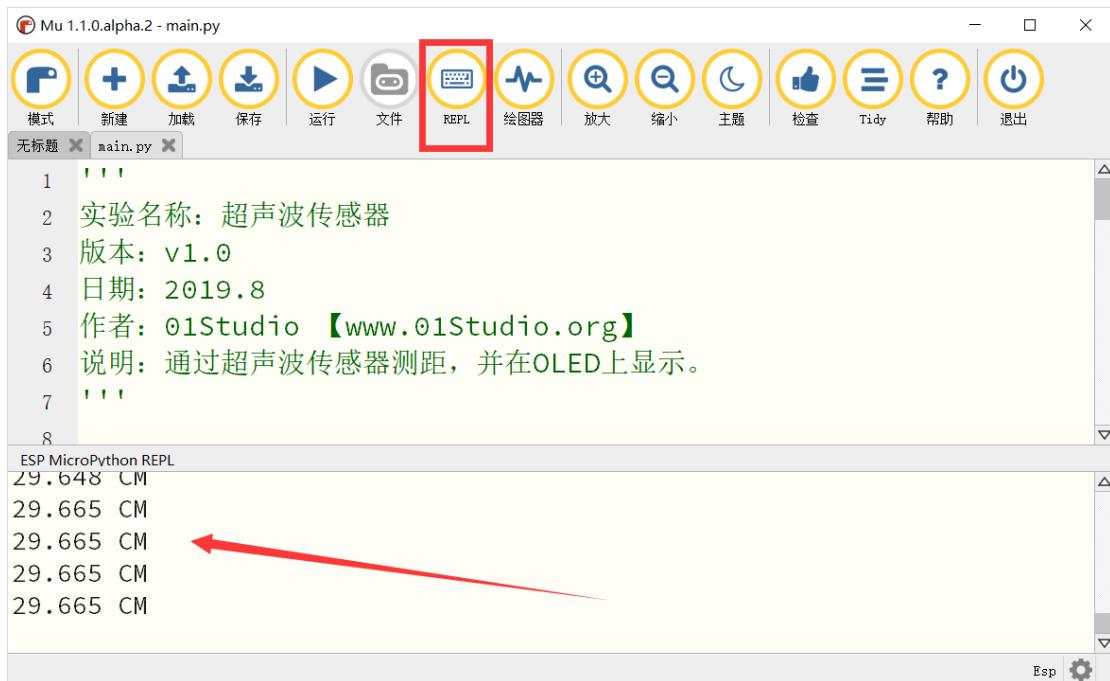


图 5-56 REPL 同步打印实验结果

● 总结:

出去 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 继电器模块

- **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:**

pyWiFi-ESP32 开发套件和继电器模块。

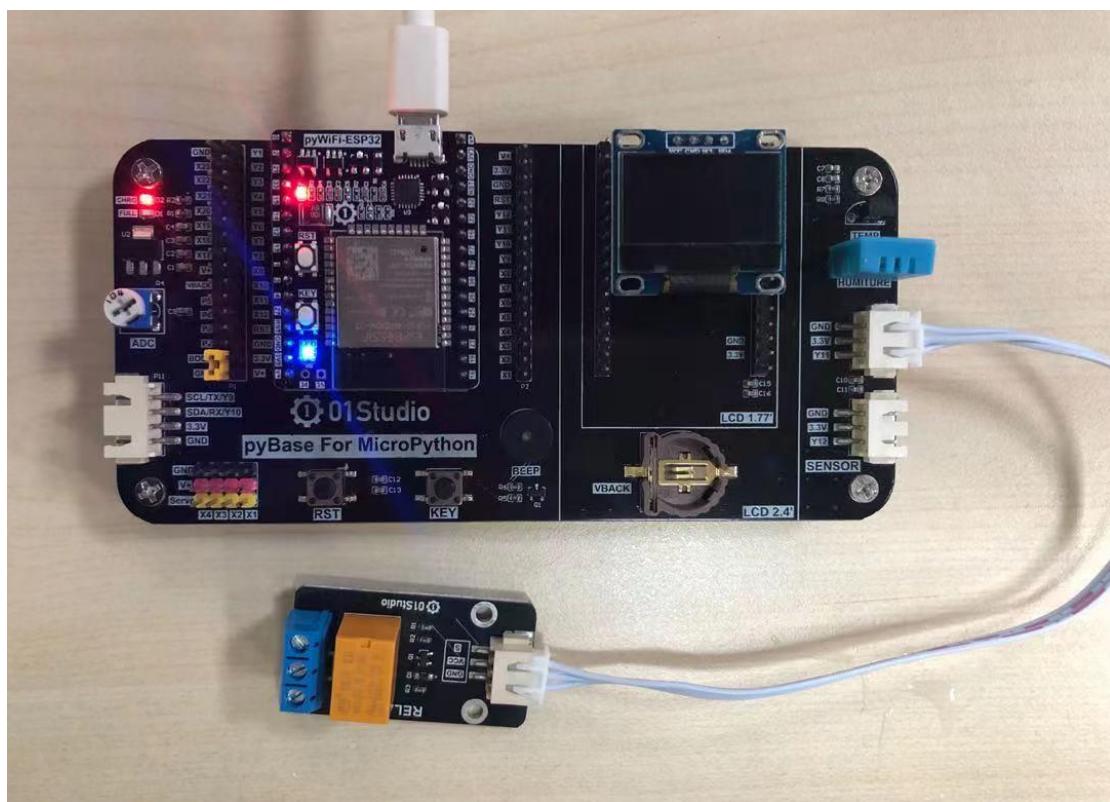


图 6-1 开发套件与传感器连接

- **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：

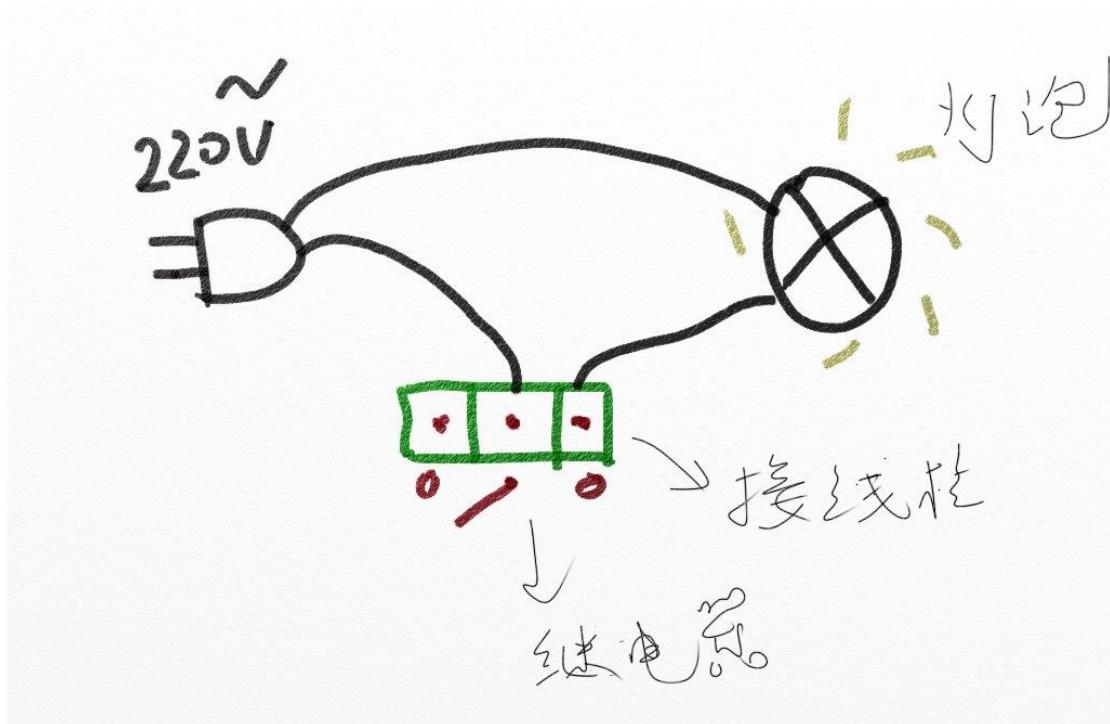


图 6-3 继电器接线

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。

我们可以参考基础实验—按键外部中断实验例程来使用继电器，请参考：错误!未找到引用源。错误!未找到引用源。 章节内容，这里不再重复！

继电器连接到 pyBase 的‘Y11’引脚，相当于连接到 ESP32 的引脚 22，编程流程图如下：

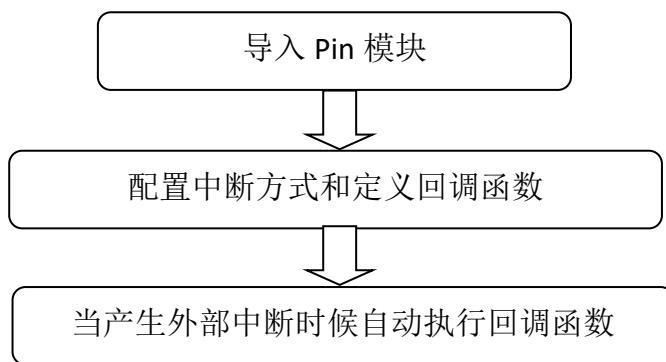


图 6-4 代码编写流程

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
说明: 通过按键改变继电器通断状态(外部中断方式)
...

#导入相关模块
from machine import Pin
import time

relay=Pin(22,Pin.OUT,value=1) #构建继电器对象,默认断开
KEY=Pin(0,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：

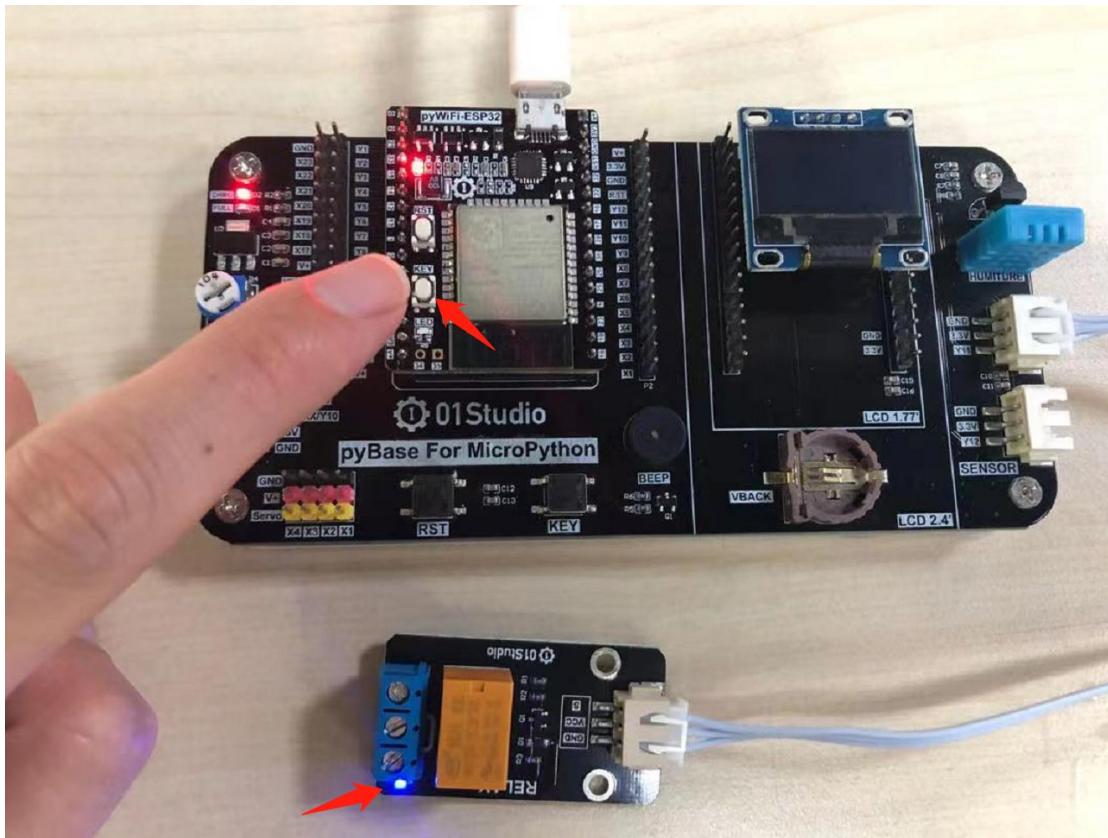


图 6-5 继电器控制

- 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° 、 180° 和 360° (360° 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：

pyWiFi-ESP32 开发套件和舵机。

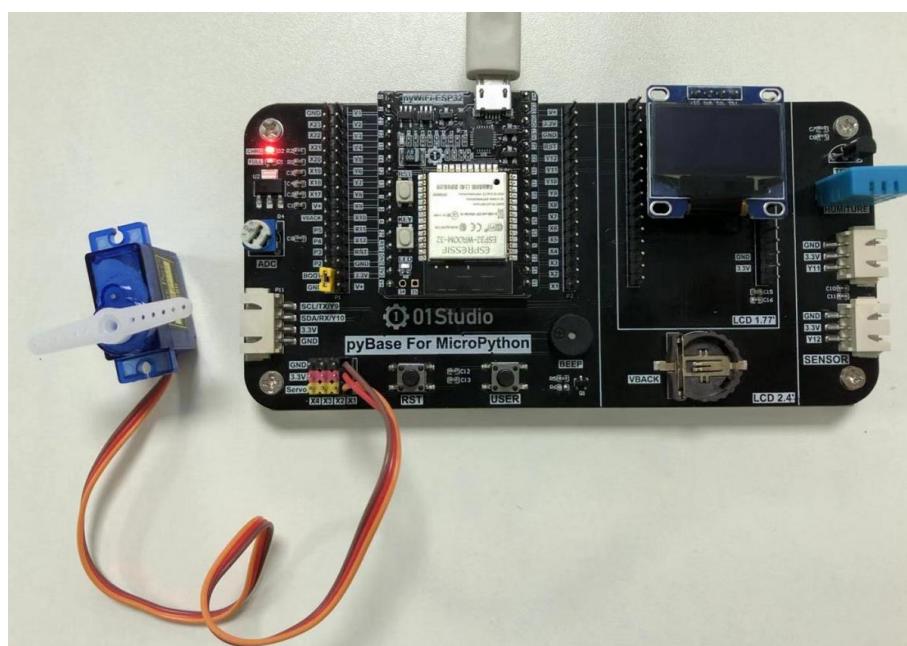


图 6-6 舵机接线

通常情况下：黑色表示 GND，红色表示 VCC，橙色表示信号线。

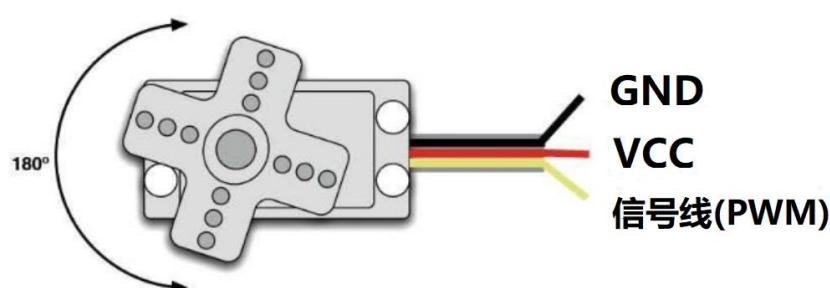


图 6-7 舵机示意图

- **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, pyBase 上有 4 个位置可以插这些电机, 分别是 X1-X4 引脚。对应 pyWiFi-ESP32 的 18、19、23、5 脚。

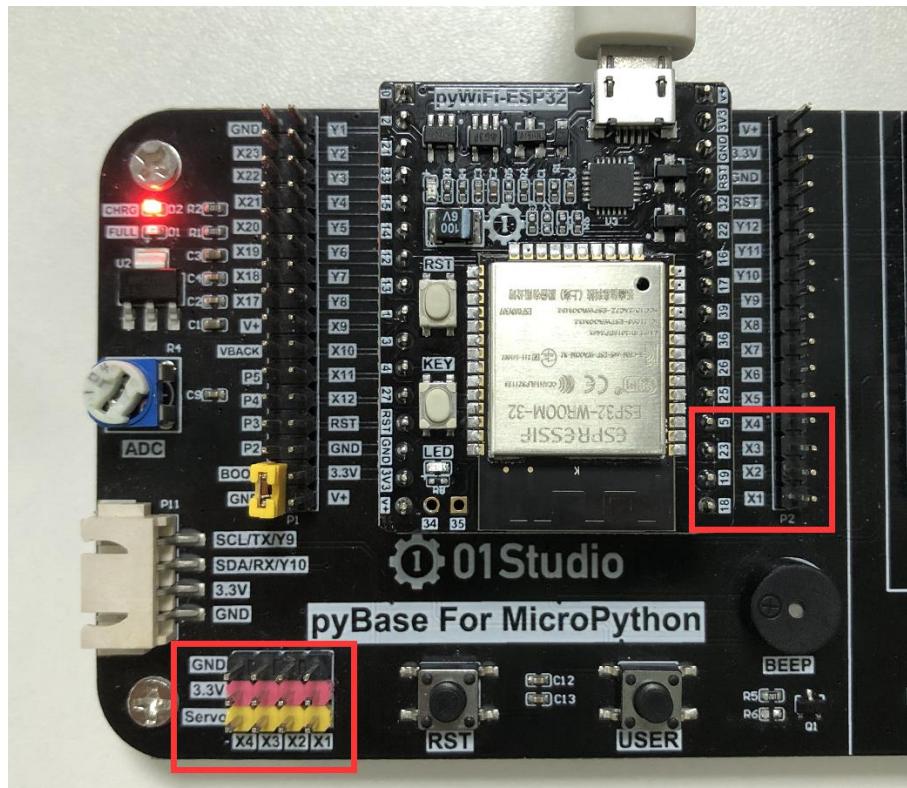


图 6-8 pyBase 的 4 路舵机接口

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° , 示意图如下:

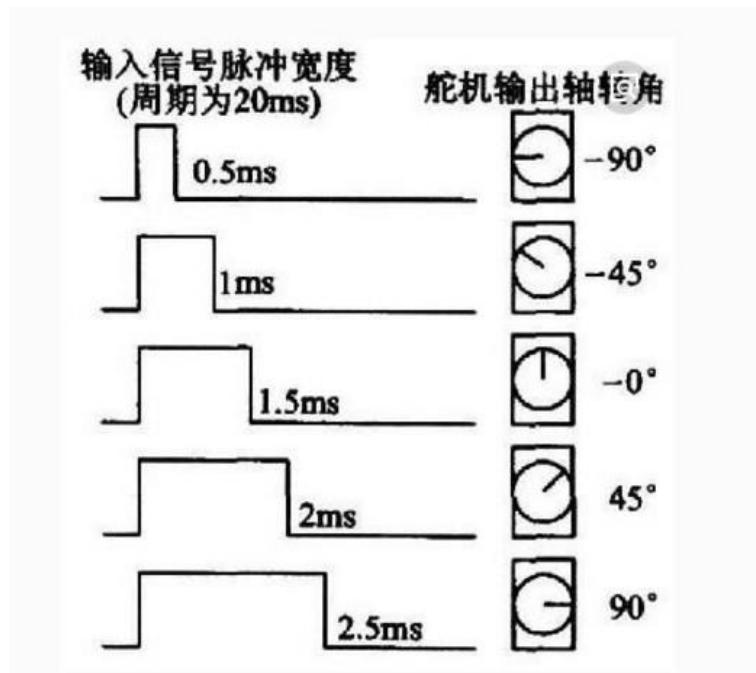


图 6-9 PWM 与角度关系

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。

如果过你学习过前面基于 STM32 平台的舵机实验，那就知道在 STM32 平台集成了舵机模块，使用起来非常方便。当前的 ESP32 平台并没有集成 Servo 模块，但从上面可以看到上面是通过 PWM 来控制的，我们可以直接写 PWM 函数驱动即可。代码编程流程图如下：

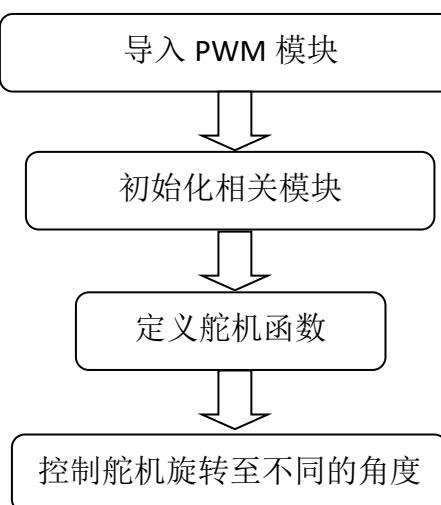


图 6-10 编程流程图

参考代码如下：

```
...
实验名称: 舵机控制
版本: v1.0
日期: 2019.8
作者: 01Studio 【www.01Studio.org】
说明: 通过编程控制舵机旋转到不同角度
...
from machine import Pin, PWM
import time

S1 = PWM(Pin(18), freq=50, duty=0) # Servo1 的引脚是 18
...
说明: 舵机控制函数
功能: 180 度舵机: angle:-90 至 90 表示相应的角度
      360 连续旋转度舵机: angle:-90 至 90 旋转方向和速度值。
...
def Servo(servo,angle):
    S1.duty(int(((angle+90)*2/180+0.5)/20*1023))

#-90 度
Servo(S1,-90)
time.sleep(1)

#-90 度
Servo(S1,-45)
```

```
time.sleep(1)
```

```
#-90 度
```

```
Servo(S1,0)
```

```
time.sleep(1)
```

```
#-90 度
```

```
Servo(S1,45)
```

```
time.sleep(1)
```

```
#-90 度
```

```
Servo(S1,90)
```

```
time.sleep(1)
```

● 实验结果：

将 180° 舵机插到 pyBase 的 X1 的三线接口，下载程序并按下复位。可以看到舵机依次旋转至不同角度。

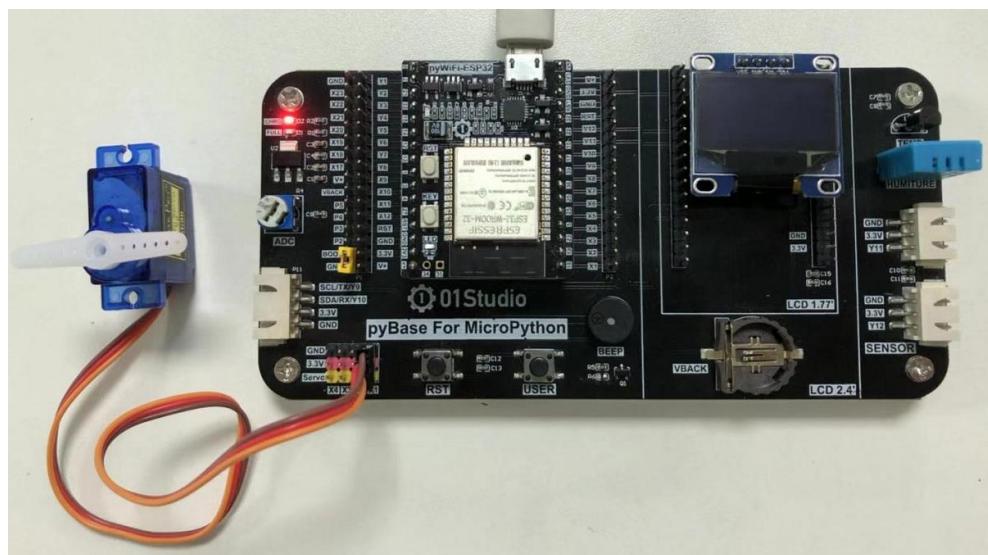


图 6-11 舵机角度旋转至不同角度

● 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。

实验的代码不变，参数【-90 至 90】代表旋转方向和速度值大小。插上 360° 连续旋转舵机。可以看到舵机的旋转速度和方向逐渐变变化。

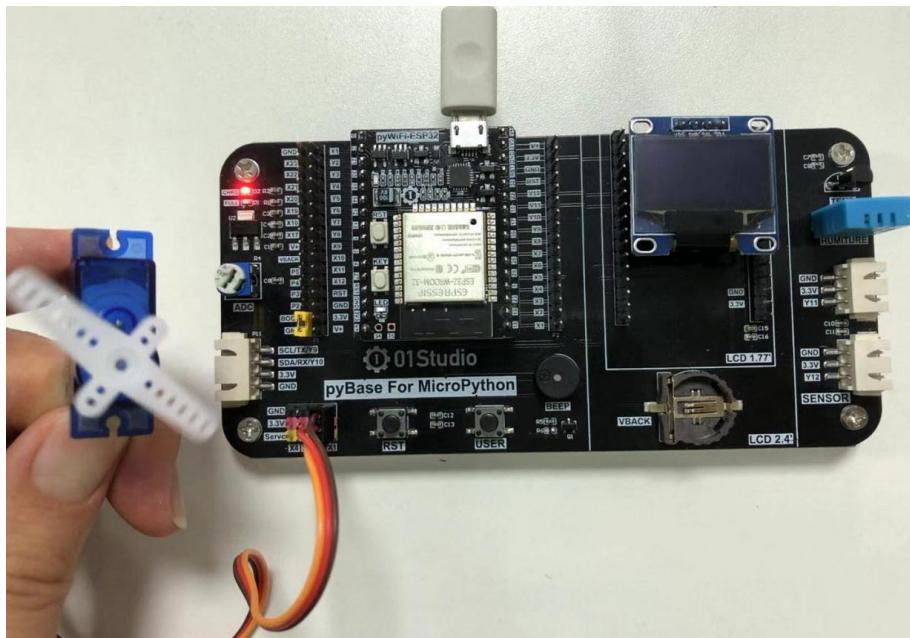


图 6-12 360° 连续旋转舵机

● 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：**

pyWiFi-ESP32 开发套件和 RGB 灯带。连接 pyBase 到“Y11”接口。

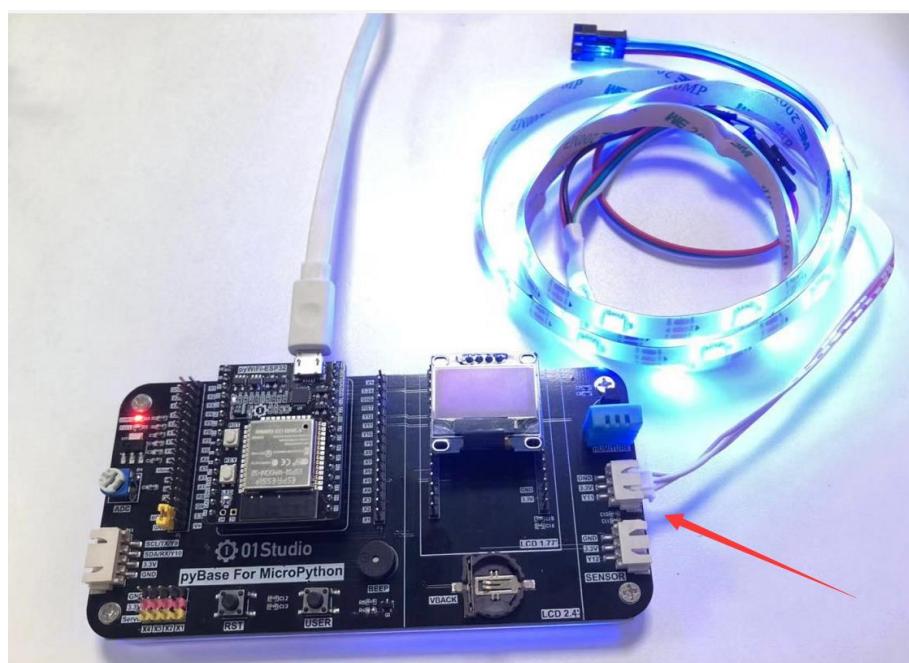


图 6-13 RGB 灯带接线方式

- **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。

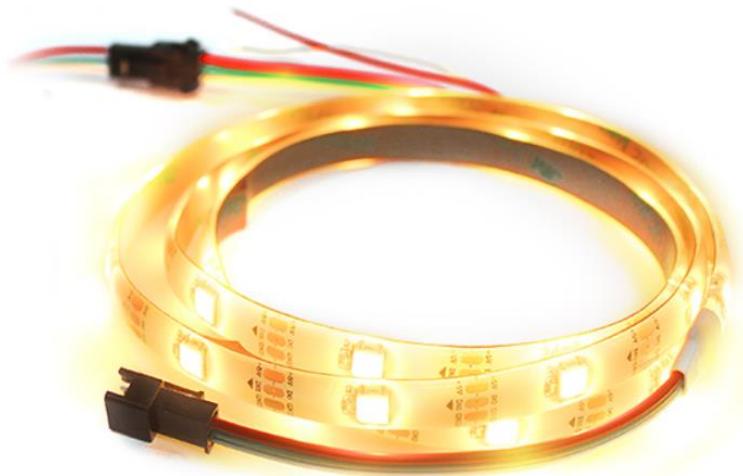


图 6-14 RGB 灯带

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。

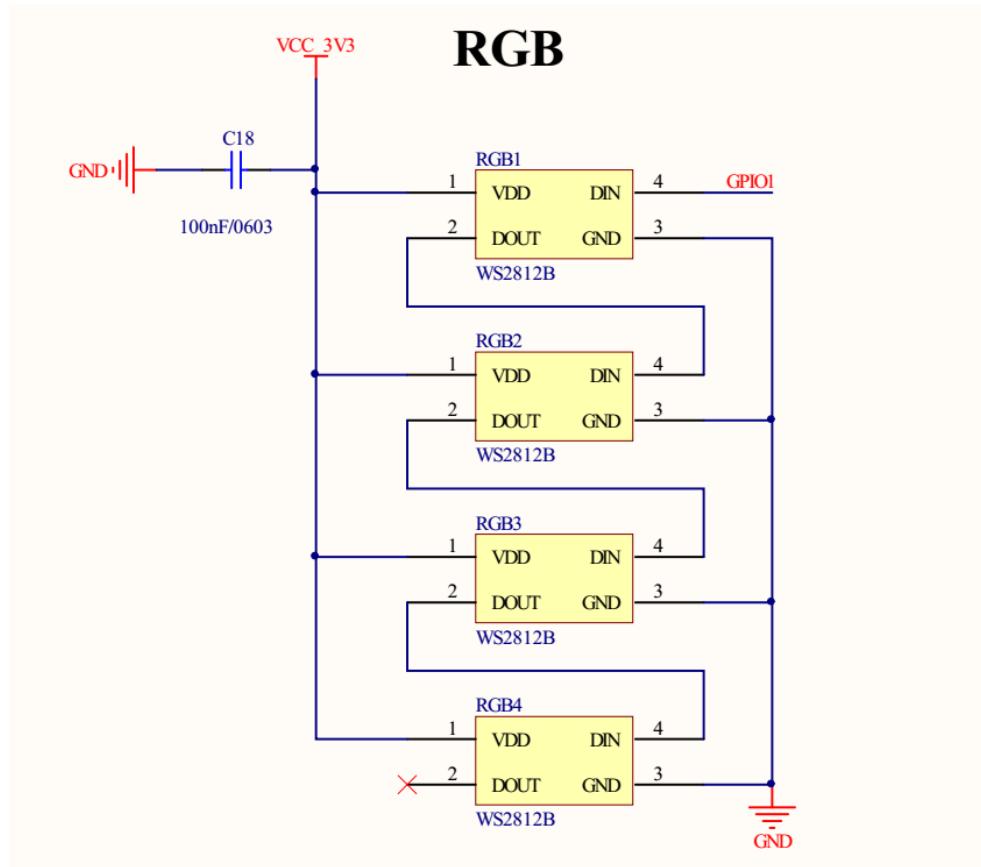


图 6-15 灯珠串联接线方式

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。

ESP32 的 MicroPython 固件集成了彩灯驱动模块 NeoPixel，适用于 WS2812B 驱动的灯珠。因此我们可以直接使用。说明如下：

构造函数
<code>np=neopixel.NeoPixel(pin, led_count)</code>
构建灯带对象。 <code>pin</code> :灯带控制引脚, <code>led_count</code> :灯珠数量;
使用方法
<code>np[0]=(R,G,B)</code>
设置第 1 个灯珠参数。
<code>np.write()</code>
往灯带写入设置的数据。

表 6-2 灯带对象

编程流程图如下：

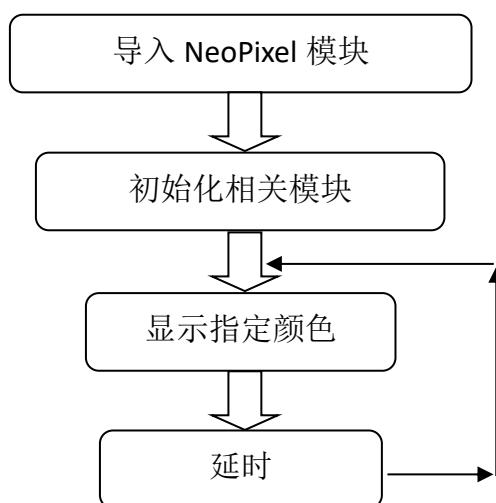


图 6-16 编程流程图

main.py 参考程序代码如下：

```
'''  
实验名称：RGB 彩灯控制  
版本：v1.0  
日期：2019.8  
作者：01Studio 【www.01Studio.org】  
说明：通过编程实现灯带不同颜色的变化。  
'''  
  
import time  
  
from machine import Pin,Timer  
  
from neopixel import NeoPixel  
  
  
#定义红、绿、蓝三种颜色  
RED=(255,0,0)  
GREEN=(0,255,0)  
BLUE=(0,0,255)  
  
  
#22 引脚连接灯带，灯珠数量 30  
pin = Pin(22, Pin.OUT)  
np = NeoPixel(pin, 30)  
  
  
#设置灯珠颜色，本实验供 30 个灯珠  
def Color_buf(color):  
    for i in range(30):  
        np[i]=color  
  
  
while True:  
  
    Color_buf(RED) #红色
```

```
np.write()      # 写入数据  
time.sleep(1)  
  
Color_buf(GREEN) # 红色  
np.write()      # 写入数据  
time.sleep(1)  
  
Color_buf(BLUE) # 红色  
np.write()      # 写入数据  
time.sleep(1)
```

● 实验结果：

将程序下载到开发套件，可以看到灯带的颜色在红、绿、蓝三色中循环变化。

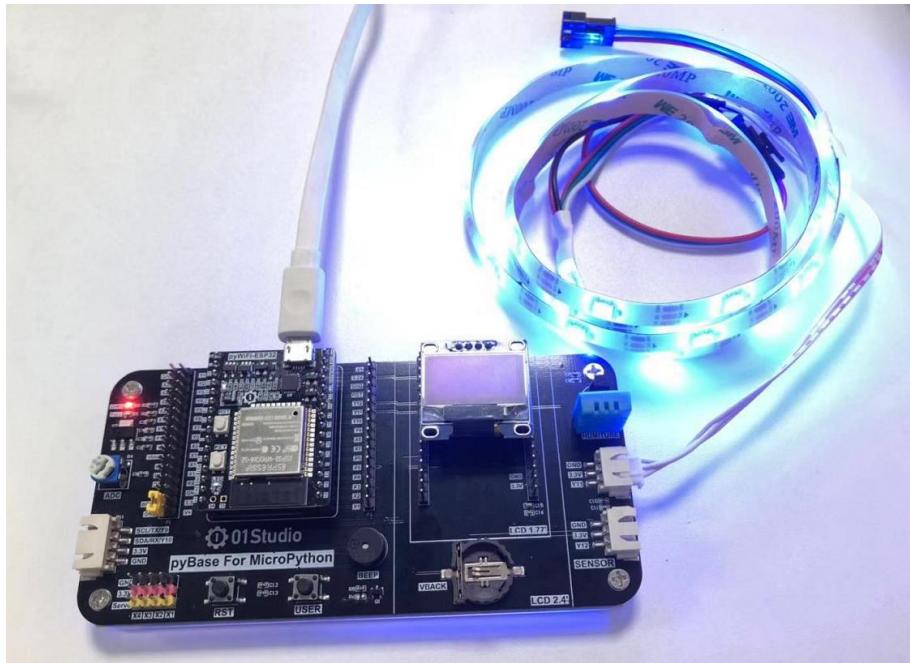


图 6-17 实验结果

● 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。

第7章 WiFi 应用

通过前面的实验，我们已经对 ESP32 有了一定的了解。从本章开始，将迎来非常重要实用的内容，那就是 WiFi 应用。ESP32 就是为 WiFi 无线连接而生的。通过本章内容，我们可以看到基于 MicroPython 的 WiFi 开发是多么的简单而美妙。物联网的学习变得非常简单有趣！事不宜迟，马上开始学习。

7.1 连接无线路由器

● 前言：

WIFI 是物联网中非常重要的角色，现在基本上家家户户都有 WIFI 网络了，通过 WIFI 接入到互联网，成了智能家居产品普遍的选择。而要想上网，首先需要连接上无线路由器。这一节我们就来学习如何通过 MicroPython 编程连上路由器。

● 实验平台：

pyWiFi-ESP32 和 pyBase 开发底板。

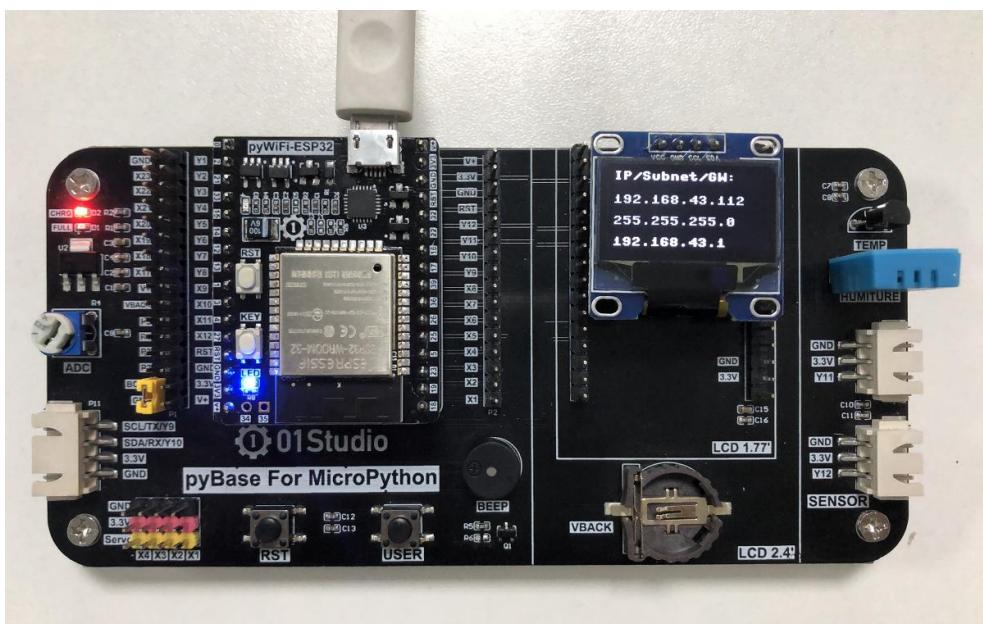


图 7-1 pyWiFi-ESP32 开发套件

● 实验目的：

编程实现连接路由器，将 IP 地址等相关信息通过 OLED 显示（只支持 2.4G 网络）。

● 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。

MicroPython 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。但往往也有各种连接失败的情况，如密码不正

确等。这时候我们只需要再加上一些简单的判断机制，避免陷入连接失败的死循环即可！

我们先来看看 network 基于 WiFi (WLAN 模块) 的构造函数和使用方法。

构造函数
wlan = network.WLAN(interface_id)
构建 WIFI 连接对象。interface_id:分为热点 network.AP_IF 和客户端 network.STA_IF 模式。
使用方法
wlan.active([is_active])
激活 wlan 接口。True: 激活; False:关闭。
wlan.scan ()
扫描允许访问的 SSID。
wlan.isconnected()
检查设备是否已经连接上。返回 True:已连接; False: 未连接。
wlan.connected(ssid,password)
WIFI 连接。ssid:账号; password: 密码。
wlan.ifconfig([ip,subnet,gateway,dns])
设备信息配置。ip: IP 地址; subnet:子网掩码; gateway:网关地址; dns:DNS 信息。(如果参数为空，则返回当前连接信息。)
wlan.disconnect()
断开连接。

表 7-1 WLAN 对象

从上表可以看到 MicroPython 通过模块封装，让 WIFI 联网变得非常简单。模块包含热点 AP 模块和客户端 STA 模式，热点 AP 是指电脑端直接连接 ESP32 发出的热点实现连接，但这样你的电脑就不能上网了，因此我们一般情况下都是使用 STA 模式。也就是电脑和设备同时连接到相同网段的路由器上。

模块上电后可以先判断是否已经连接到网络，如果是则无需再次连接，否的

话则进入 WiFi 连接状态，指示灯闪烁，连接成功后指示灯常亮，IP 等相关信息通过 OLED 显示和串口打印。另外需要配置超时 15 秒还没连接成功时执行取消连接，避免因无法连接而陷入死循环。代码编写流程如下：

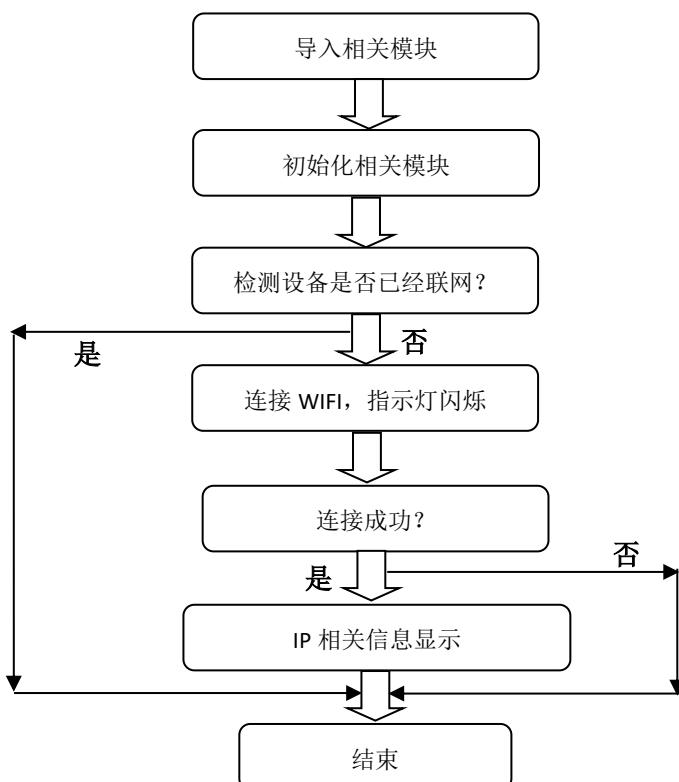


图 7-2 代码编写流程图

实验参考代码如下：

```
...  
实验名称: 连接无线路由器  
版本: v1.0  
日期: 2019.8  
作者: 01Studio  
说明: 编程实现连接路由器, 将 IP 地址等相关信息通过 OLED 显示 (只支持 2.4G 网络)。  
...  
import network,time  
from machine import I2C,Pin
```

```
from ssd1306 import SSD1306_I2C

#初始化相关模块

i2c = I2C(sda=Pin(13), scl=Pin(14))
oled = SSD1306_I2C(128, 64, i2c, addr=0x3c)

#WIFI 连接函数

def WIFI_Connect():

    WIFI_LED=Pin(2, Pin.OUT) #初始化 WIFI 指示灯

    wlan = network.WLAN(network.STA_IF) #STA 模式
    wlan.active(True) #激活接口
    start_time=time.time() #记录时间做超时判断

    if not wlan.isconnected():
        print('connecting to network...')
        wlan.connect('01Studio', '88888888') #输入 WIFI 账号密码

    while not wlan.isconnected():

        #LED 闪烁提示
        WIFI_LED.value(1)
        time.sleep_ms(300)
        WIFI_LED.value(0)
        time.sleep_ms(300)

    #超时判断,15 秒没连接成功判定为超时
    if time.time()-start_time > 15 :
        print('WIFI Connected Timeout!')
```

```
break

if wlan.isconnected():
    #LED 点亮
    WIFI_LED.value(1)

    #串口打印信息
    print('network information:', wlan.ifconfig())

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

#执行 WIFI 连接函数
WIFI_Connect()
```

上面代码在 `main.py` 文件中，将 `WIFI` 连接封装成了子函数形式，我们也可以新建一个 `WIFI.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：

下载程序，可以观察到上电后 `pyWiFi-ESP32` 上的 LED 快速闪烁，连接成功后 LED 常亮，OLED 显示当前 IP、子网掩发、网关的地址信息。

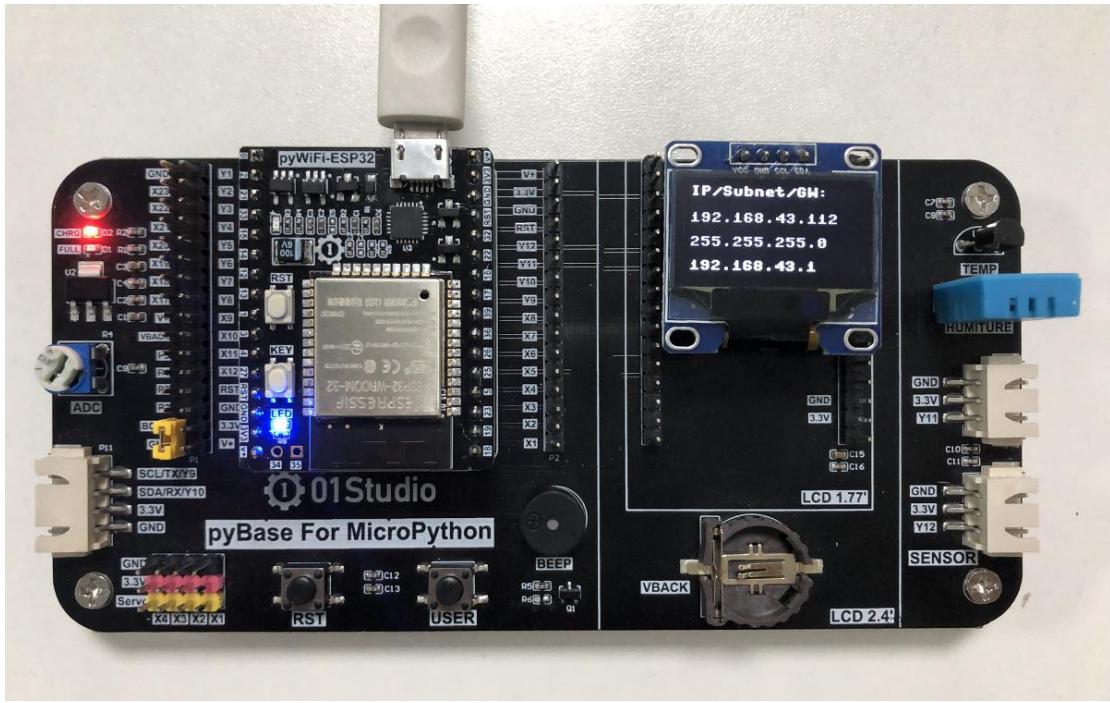


图 7-3

● 总结：

本节是 WiFi 应用的基础，成功连接到无线路由器的实验后，后面就可以做 socket 等相关网络通信的应用了。

7.2 Socket 通信

● 前言：

上一节我们学习了如何通过 MicroPython 编程实现 pyWiFi-ESP32 模块连接到无线路由器。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：

pyWiFi-ESP32 和 pyBase 开发底板。

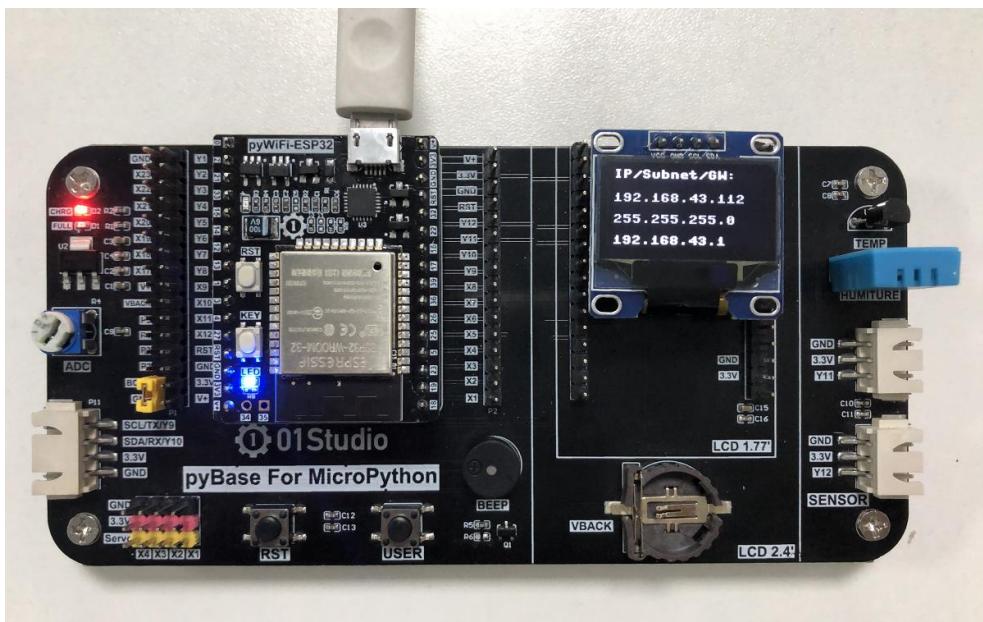


图 7-4 pyWiFi-ESP32 开发套件

● 实验目的：

通过 Socket 编程实现 pyWiFi-ESP32 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：

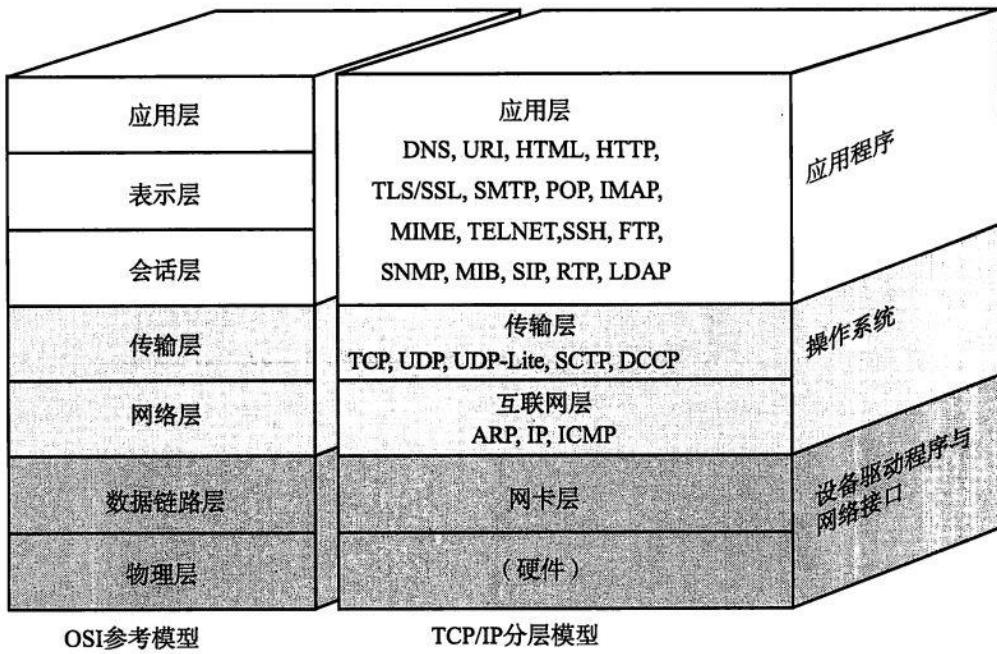


图 7-5 网络层级模型

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，UPD 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。

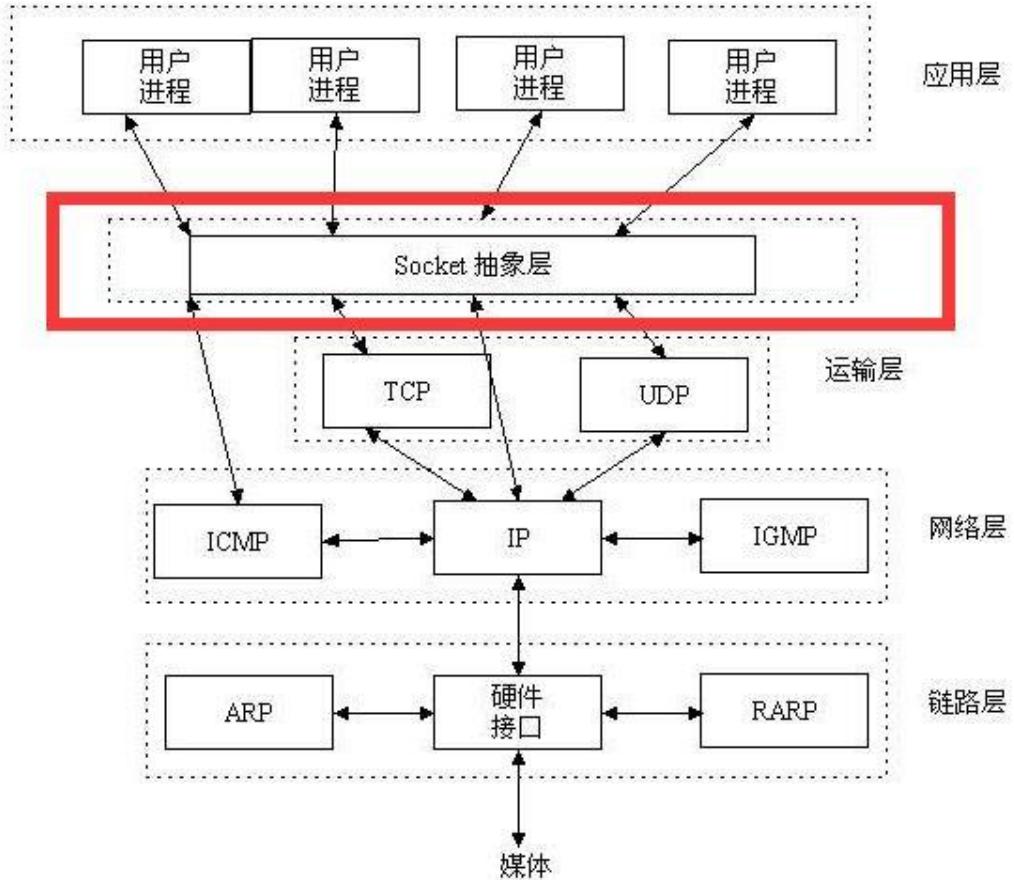


图 7-6 Socket 抽象层

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：

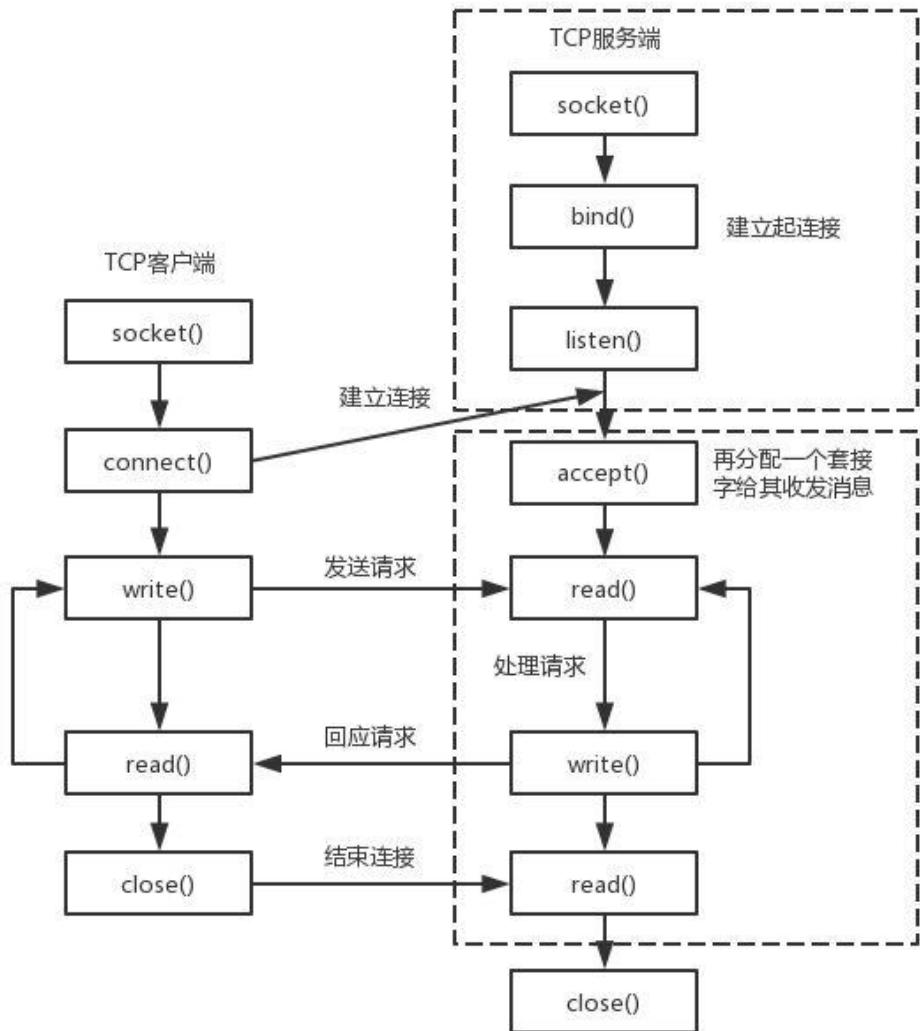


图 7-7 Socket 通信过程

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，pyWiFi-ESP32 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 ESP32 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) 中文文档链接: http://docs.micropython.01studio.org/

表 7-2 Socket 对象

本实验中 pyWiFi-ESP32 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:

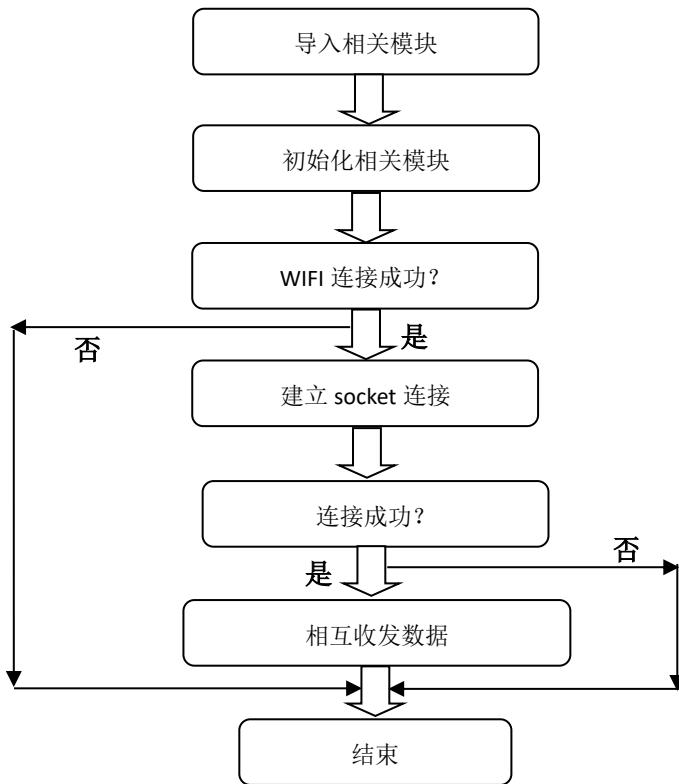


图 7-8 代码编写流程图

实验参考代码：

```

...
实验名称: 连接无线路由器
版本: v1.0
日期: 2019.8
作者: 01Studio
说明: 通过 Socket 编程实现 pyWiFi-ESP32 与电脑服务器助手建立 TCP 连接, 相互收发
数据。
...
#导入相关模块
import network,usocket,time
from machine import I2C,Pin,Timer
from ssd1306 import SSD1306_I2C

```

```

#初始化相关模块

i2c = I2C(sda=Pin(13), scl=Pin(14))

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


# WIFI 连接函数

def WIFI_Connect():

    WIFI_LED=Pin(2, Pin.OUT) #初始化 WIFI 指示灯


    wlan = network.WLAN(network.STA_IF) #STA 模式

    wlan.active(True)                  #激活接口

    start_time=time.time()            #记录时间做超时判断


    if not wlan.isconnected():

        print('connecting to network...')

        wlan.connect('01Studio', '88888888') #输入 WIFI 账号密码

        while not wlan.isconnected():

            #LED 闪烁提示

            WIFI_LED.value(1)

            time.sleep_ms(300)

            WIFI_LED.value(0)

            time.sleep_ms(300)

    #超时判断,15 秒没连接成功判定为超时

    if time.time()-start_time > 15 :

        print('WIFI Connected Timeout!')

        break


    if wlan.isconnected():

        #LED 点亮

```

```
WIFI_LED.value(1)

#串口打印信息
print('network information:', wlan.ifconfig())

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

return True

else:
    return False

def Socket_fun(tim):

    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'))

#判断 WIFI 是否连接成功
if WIFI_Connect():
```

```

#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=('192.168.1.115',10000) #服务器 IP 和端口

s.connect(addr)

s.send('Hello 01Studio!')


#开启 RTOS 定时器，编号为-1，周期 300ms，执行 socket 通信接收任务

tim = Timer(-1)

tim.init(period=300, mode=Timer.PERIODIC,callback=Socket_fun)

```

WIFI 连接代码在上一节已经讲解，这里不再重复，WIFI 连接成功后返回 True，否则返回 False。程序在返回连接成功后建了 Socket 连接，连接成功发送 ‘Hello 01Studio!’ 信息到服务器。另外 RTOS 定时器设定了每 300ms 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，软件在 零一科技（01Studio）MicroPython 开发套件配套资料_latest\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！

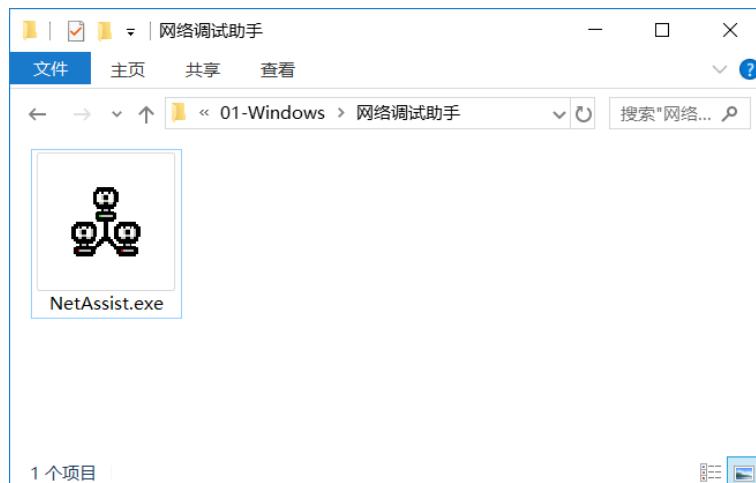


图 7-9 网络调试助手

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：

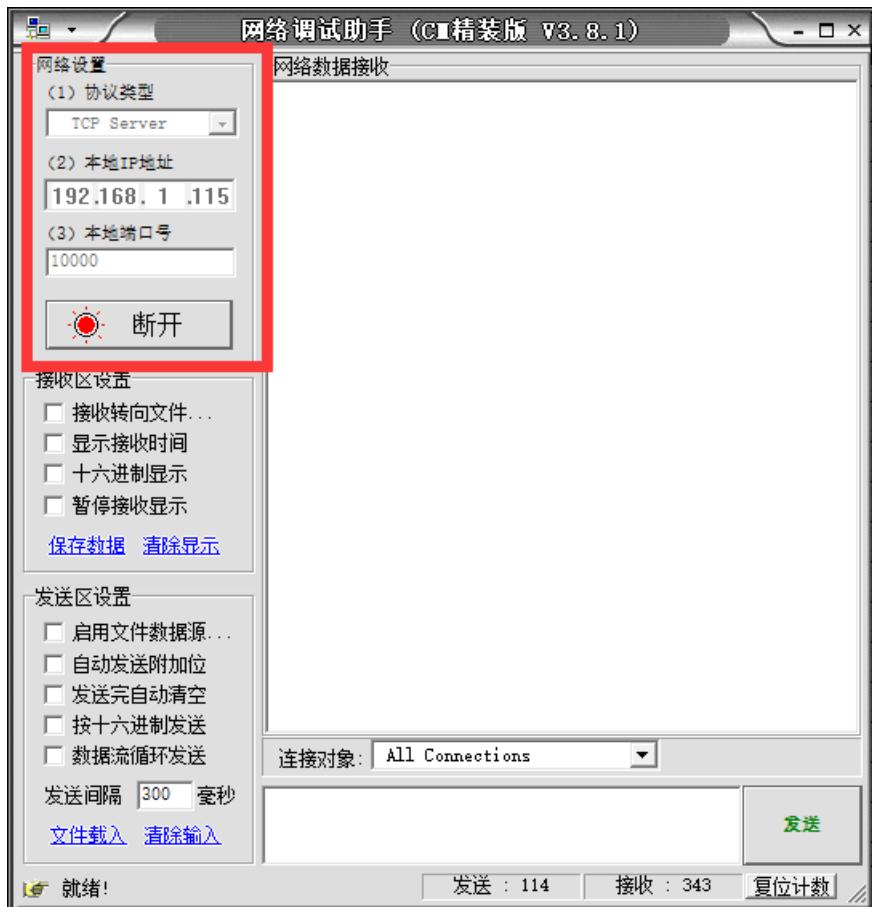


图 7-10 TCP 服务器配置

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输入 WIFI 信息和服务器 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。）

```
wlan.connect('01Studio', '88888888') #输入 WIFI 账号密码  
addr=('192.168.1.115',10000) #服务器 IP 和端口
```

下载程序，开发板成功连接 WIFI 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：

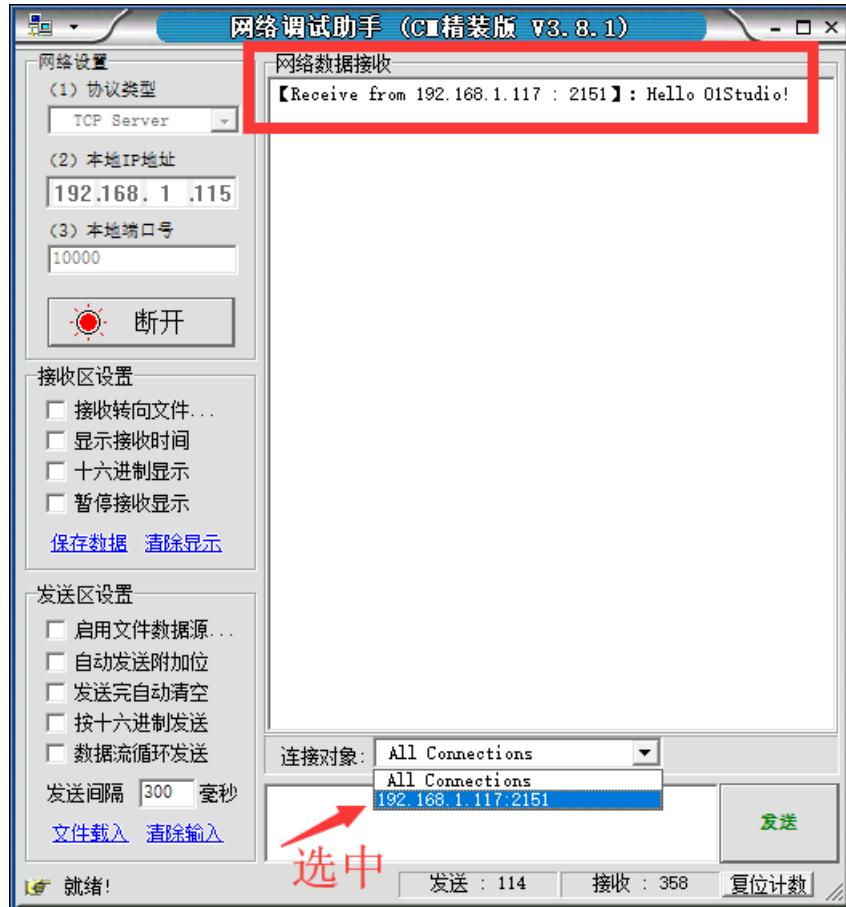


图 7-11 选中连接对象

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。

```
MicroPython v1.11-8-g48dcbe60 on 2019-05-29; ESP module with ESP8266
Type "help()" for more information.
>>> Warning: Comparison between bytes and str
b'Hi' ←
```

图 7-12

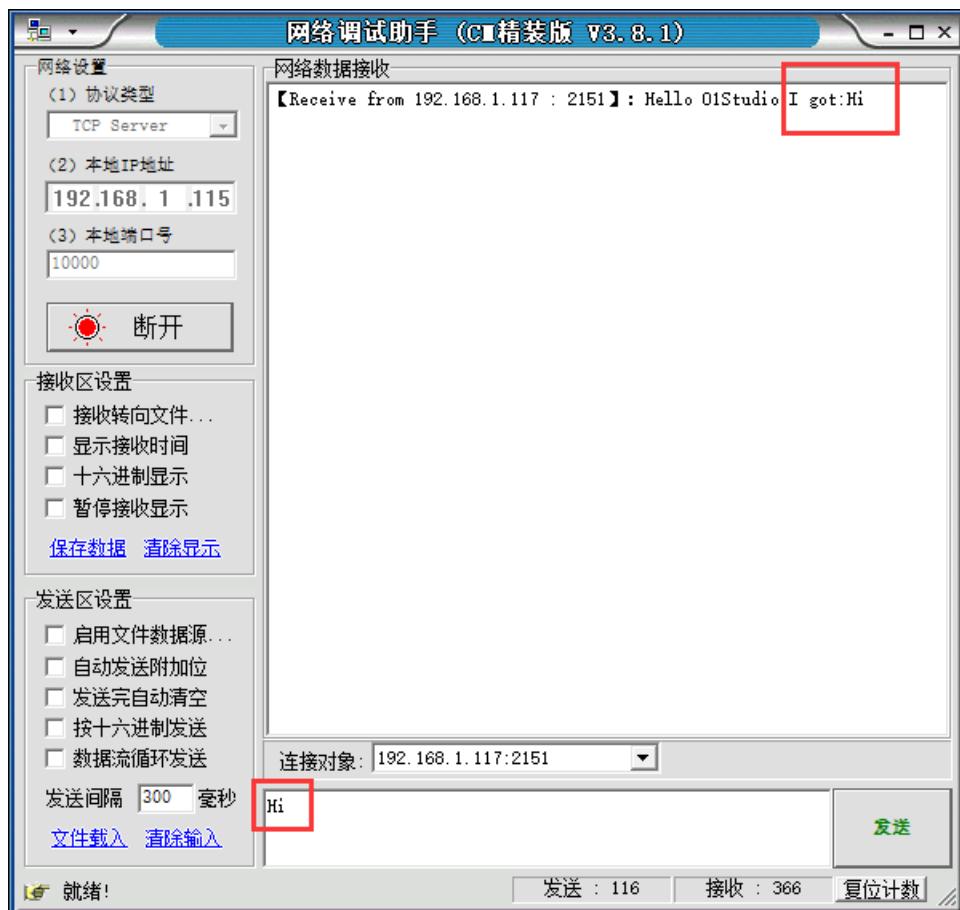


图 7-13 服务器发送数据

● 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。

7.3 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：

pyWiFi-ESP32 和 pyBase 开发底板。

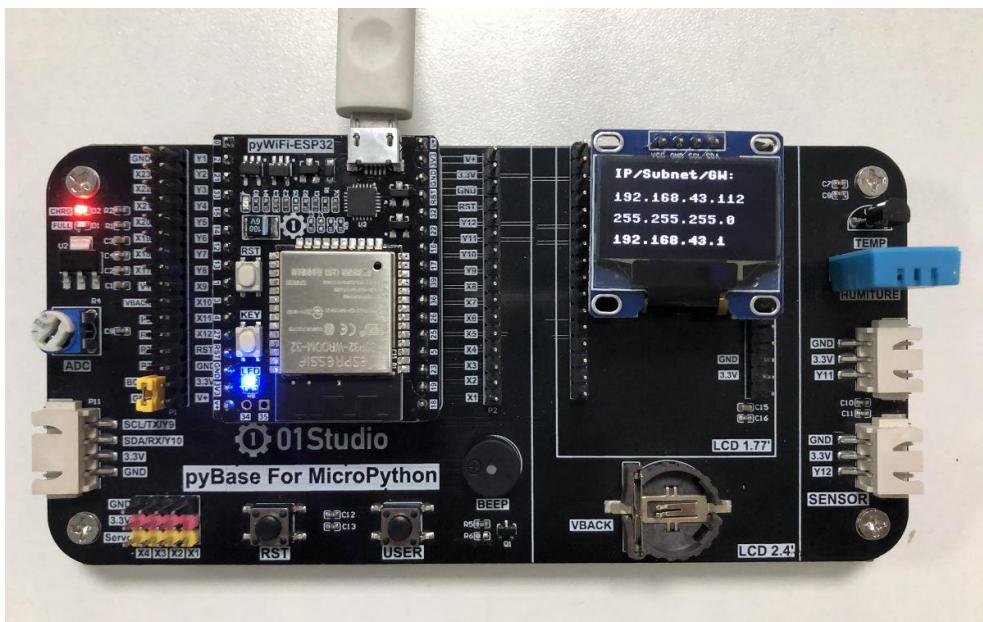


图 7-14 pyWiFi-ESP32 开发套件

● 实验目的：

通过编程实现 pyWiFi-ESP32 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

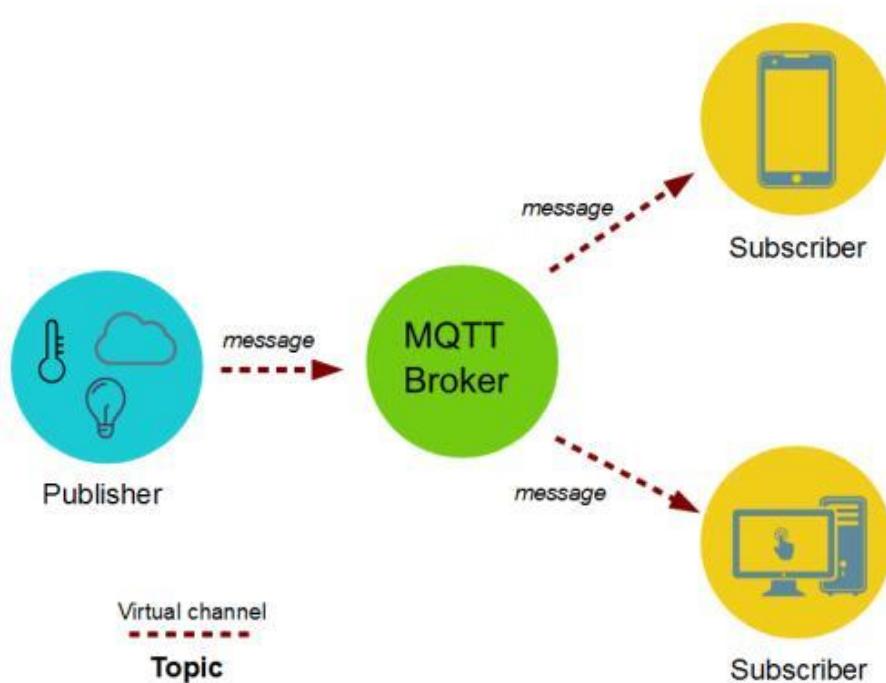


图 7-15 MQTT 通信流程

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：

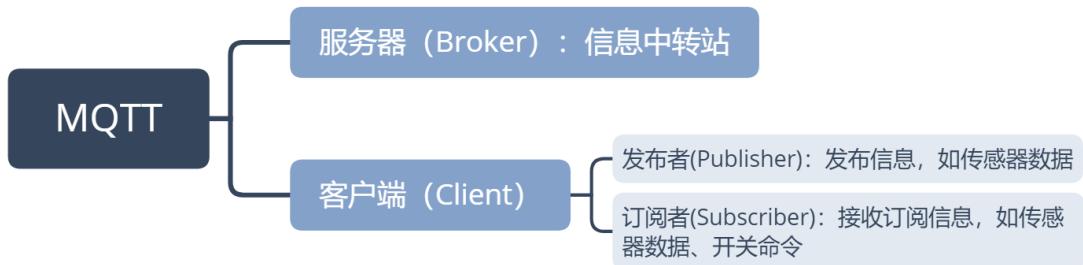


图 7-16 MQTT 角色说明

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 7-3 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。MQTT 模块文件为例程文件夹里面的 `simple.py` 文件。使用方法如下：

构造函数
<code>client=simple.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 7-4 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:

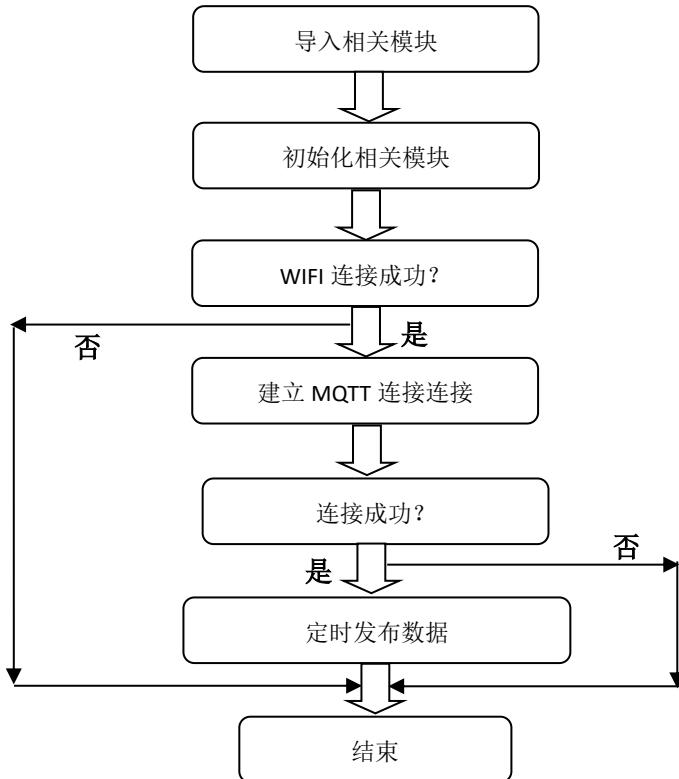


图 7-17 发布者代码编写流程

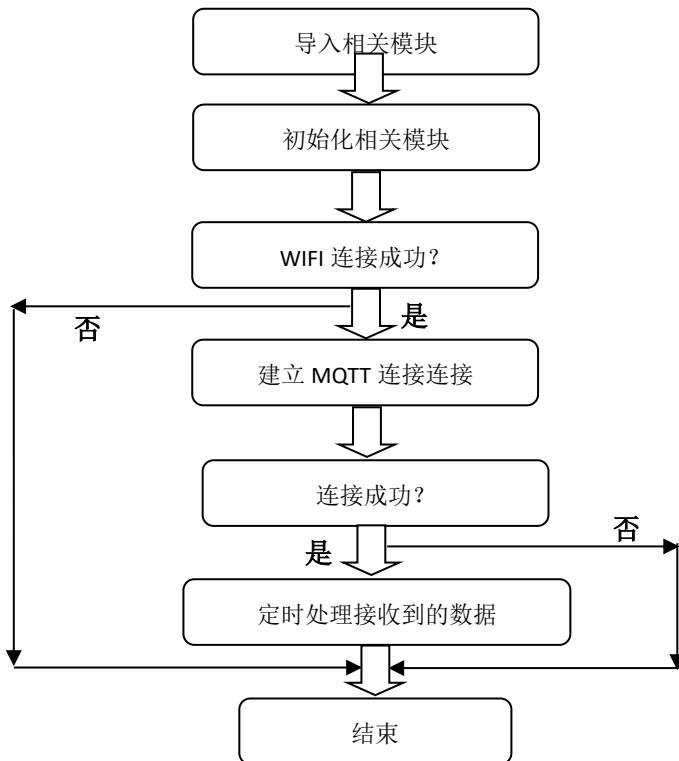


图 7-18 订阅者代码编写流程

发布者（publish）参考代码：

```

...
实验名称: MQTT 通信
版本: v1.0
日期: 2019.8
作者: 01Studio
说明: 编程实现 MQTT 通信, 实现发布数据。
...

import network,time
from simple import MQTTClient #导入 MQTT 板块
from machine import I2C,Pin,Timer
from ssd1306 import SSD1306_I2C
#初始化相关模块

```

```

i2c = I2C(sda=Pin(13), scl=Pin(14))
oled = SSD1306_I2C(128, 64, i2c, addr=0x3c)

#WIFI 连接函数

def WIFI_Connect():

    WIFI_LED=Pin(2, Pin.OUT) #初始化 WIFI 指示灯

    wlan = network.WLAN(network.STA_IF) #STA 模式
    wlan.active(True)                 #激活接口
    start_time=time.time()           #记录时间做超时判断

    if not wlan.isconnected():
        print('connecting to network...')
        wlan.connect('01Studio', '88888888') #输入 WIFI 账号密码

        while not wlan.isconnected():

            #LED 闪烁提示
            WIFI_LED.value(1)
            time.sleep_ms(300)
            WIFI_LED.value(0)
            time.sleep_ms(300)

        #超时判断,15 秒没连接成功判定为超时
        if time.time()-start_time > 15 :
            print('WIFI Connected Timeout!')
            break

    if wlan.isconnected():

```

```

#LED 点亮

WIFI_LED.value(1)

#串口打印信息

print('network information:', wlan.ifconfig())


#OLED 显示数据（如果没接 OLED， 请将下面代码屏蔽）

oled.fill(0)    #清屏背景黑色

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

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

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

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

oled.show()

return True


else:

    return False


#发布数据任务

def MQTT_Send(topic):

    client.publish(TOPIC, 'Hello 01Studio!')


#执行 WIFI 连接函数并判断是否已经连接成功

if WIFI_Connect():

    SERVER = 'mq.tongxinmao.com'

    PORT = 18830

    CLIENT_ID = 'ESP-32' # 客户端 ID

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

    client = MQTTClient(CLIENT_ID, SERVER, PORT)

```

```
client.connect()

#开启 RTOS 定时器，编号为-1,周期 1000ms，执行 socket 通信接收任务
tim = Timer(-1)

tim.init(period=1000, mode=Timer.PERIODIC,callback=MQTT_Send)
```

订阅者（subscribe）参考代码：

```
'''

实验名称：MQTT 通信

版本：v1.0

日期：2019.8

作者：01Studio

说明：编程实现 MQTT 通信，实现订阅（接收）数据。

'''

import network,time

from simple import MQTTClient #导入 MQTT 板块

from machine import I2C,Pin,Timer

from ssd1306 import SSD1306_I2C


#初始化相关模块

i2c = I2C(sda=Pin(13), scl=Pin(14))

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


#WIFI 连接函数

def WIFI_Connect():

    WIFI_LED=Pin(2, Pin.OUT) #初始化 WIFI 指示灯


    wlan = network.WLAN(network.STA_IF) #STA 模式
```

```

wlan.active(True) #激活接口

start_time=time.time() #记录时间做超时判断


if not wlan.isconnected():

    print('connecting to network...')

    wlan.connect('01Studio', '88888888') #输入 WIFI 账号密码


    while not wlan.isconnected():

        #LED 闪烁提示

        WIFI_LED.value(1)

        time.sleep_ms(300)

        WIFI_LED.value(0)

        time.sleep_ms(300)

#超时判断,15 秒没连接成功判定为超时

if time.time()-start_time > 15 :

    print('WIFI Connected Timeout!')

    break


if wlan.isconnected():

    #LED 点亮

    WIFI_LED.value(1)


    #串口打印信息

    print('network information:', wlan.ifconfig())


#OLED 数据显示（如果没接 OLED, 请将下面代码屏蔽）

oled.fill(0) #清屏背景黑色

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

```

```

        oled.text(wlan.ifconfig()[0], 0, 20)
        oled.text(wlan.ifconfig()[1],0,38)
        oled.text(wlan.ifconfig()[2],0,56)
        oled.show()
        return True

    else:
        return False

#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))

#接收数据任务
def MQTT_Rev(tim):
    client.check_msg()

#执行 WIFI 连接函数并判断是否已经连接成功
if WIFI_Connect():

    SERVER = 'mq.tongxinmao.com'
    PORT = 18830
    CLIENT_ID = '01Studio-ESP32' # 客户端 ID
    TOPIC = '/public/01Studio/1' # TOPIC 名称

    client = MQTTCClient(CLIENT_ID, SERVER, PORT) #建立客户端对象
    client.set_callback(MQTT_callback) #配置回调函数
    client.connect()
    client.subscribe(TOPIC) #订阅主题

```

```
#开启 RTOS 定时器，编号为-1，周期 300ms，执行 socket 通信接收任务
tim = Timer(-1)

tim.init(period=300, mode=Timer.PERIODIC,callback=MQTT_Rev)
```

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：

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。

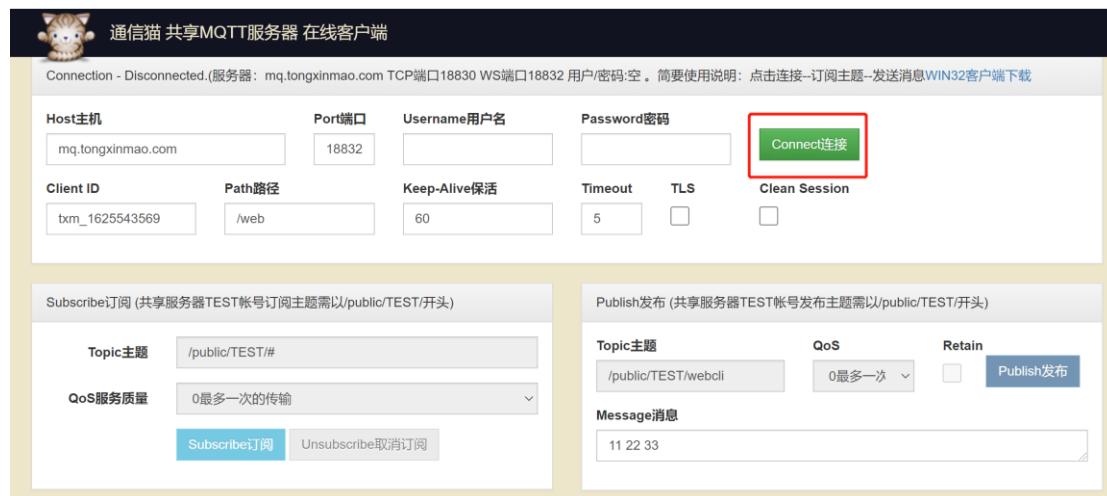


图 7-19 MQTT 在线助手

连接成功可以看到显示。



图 7-20 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 7-21 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。

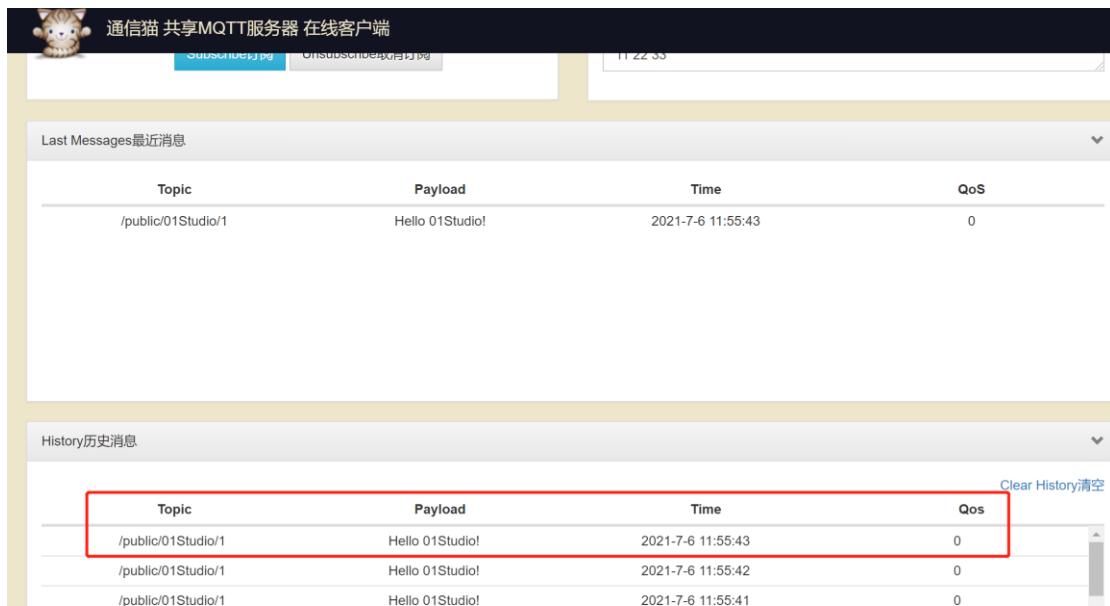


图 7-22 MQTT 助手收到开发板发布的信息

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!”。

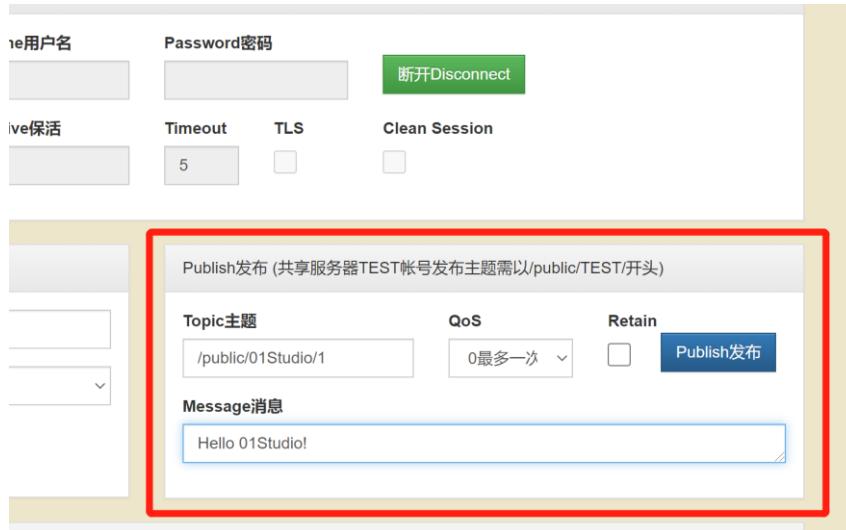


图 7-23 输入即将发布的主题和信息

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 7-24 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 7-25 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。

7.4 WebREPL

串口的 REPL 【读取(Read)-运算(Eval)-输出(Print)-循环(Loop)】我们已经非常熟悉了使用了。而针对 WIFI 类设备，在有些时候设备已经部署好后不方便通过串口线连接，那么我们就可以通过局域网的方式来调试或者传输文件。以更加便捷的方式调试产品。例如我们完成了一个小车产品，那么就可以通过 WebREPL 方式来调试。以下是具体方法介绍。

第1步:

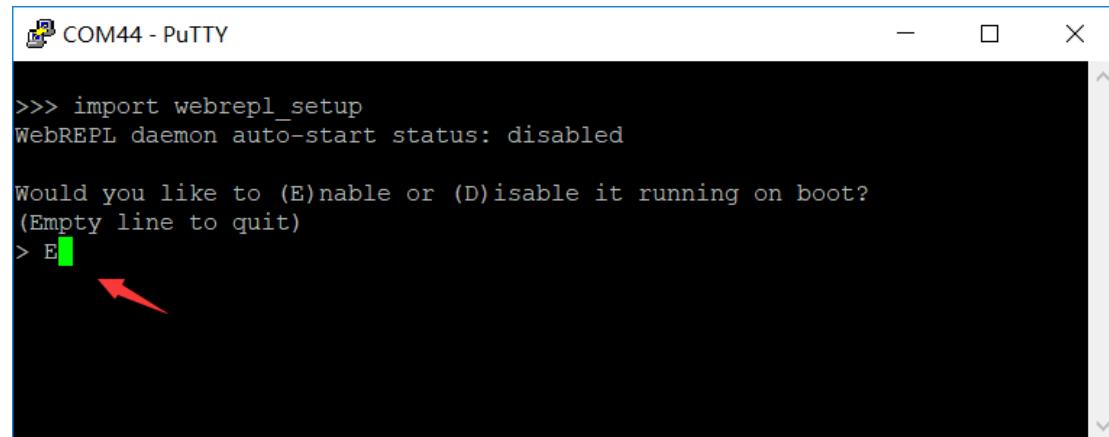
确保你的电脑和设备已经连接到同一个网络上（通常指同一个网段，如 192.168.1.X）。具体可以参考本章连接无线路由器章节实验。

第2步: 配置开发板

在串口终端输入以下命令：

```
import webrepl_setup
```

在弹出的串口提示框中输入'E'，按回车，使能 WebREPL 相关功能；



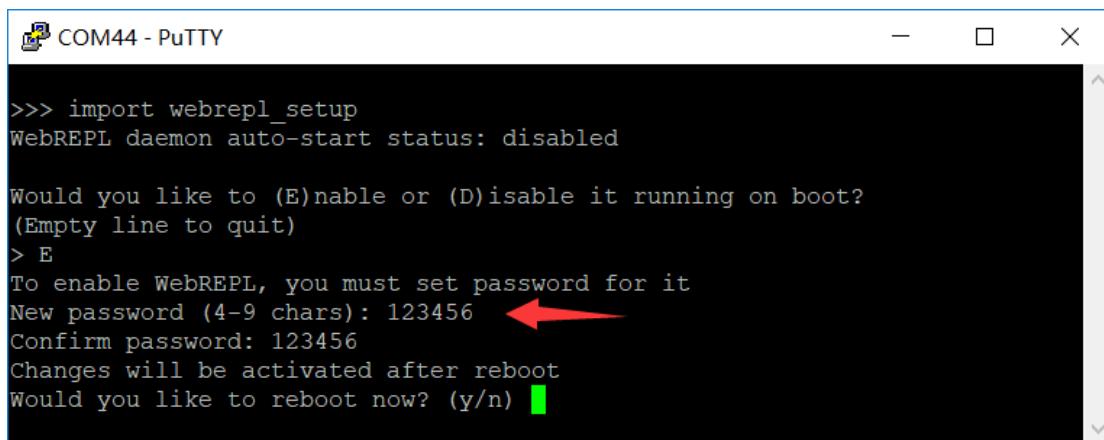
```
COM44 - PuTTY

>>> import webrepl_setup
WebREPL daemon auto-start status: disabled

Would you like to (E)nable or (D)isable it running on boot?
(Empty line to quit)
> E
```

图 7-26

接下来设置密码，设置完成后输入‘y’，选择重启。



```
>>> import webrepl_setup
WebREPL daemon auto-start status: disabled

Would you like to (E)nable or (D)isable it running on boot?
(Empty line to quit)
> E
To enable WebREPL, you must set password for it
New password (4-9 chars): 123456 ←
Confirm password: 123456
Changes will be activated after reboot
Would you like to reboot now? (y/n) [
```

图 7-27

第3步:

打开浏览器，输入 <http://webrepl.01studio.org/> 进入，可以见到 WebREPL 界面。

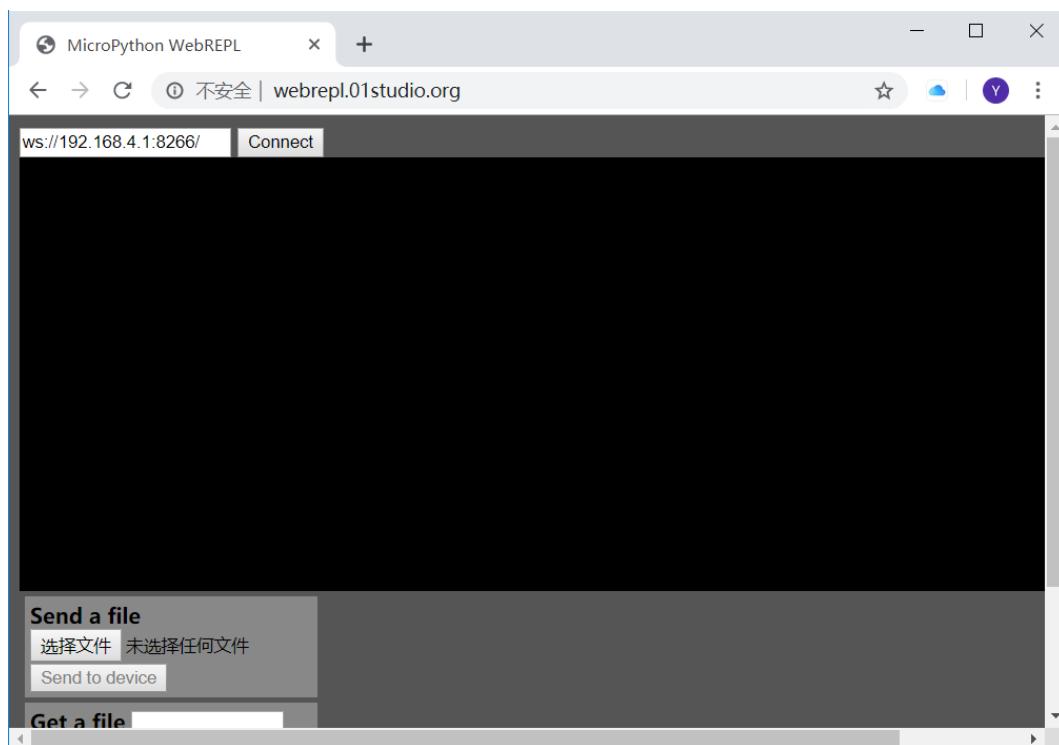


图 7-28 WebREPL

第4步:

在左上角输入 pyWiFi-ESP32 当前的 IP 地址，端口为 8266。点击 Connect 连接。看到出现密码输入提示：

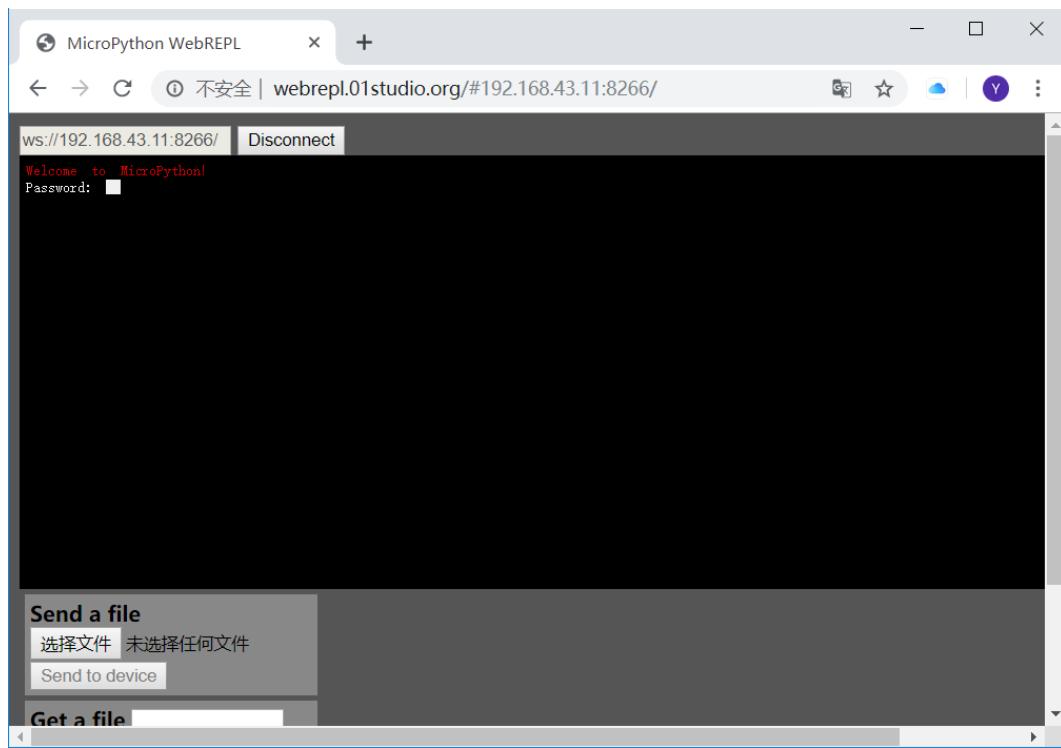


图 7-29

输入之前设置的密码（密码不显示），按回车。连接成功后出现 REPL 交互提示的功能。这是即可以使用跟 REPL 一样的功能。

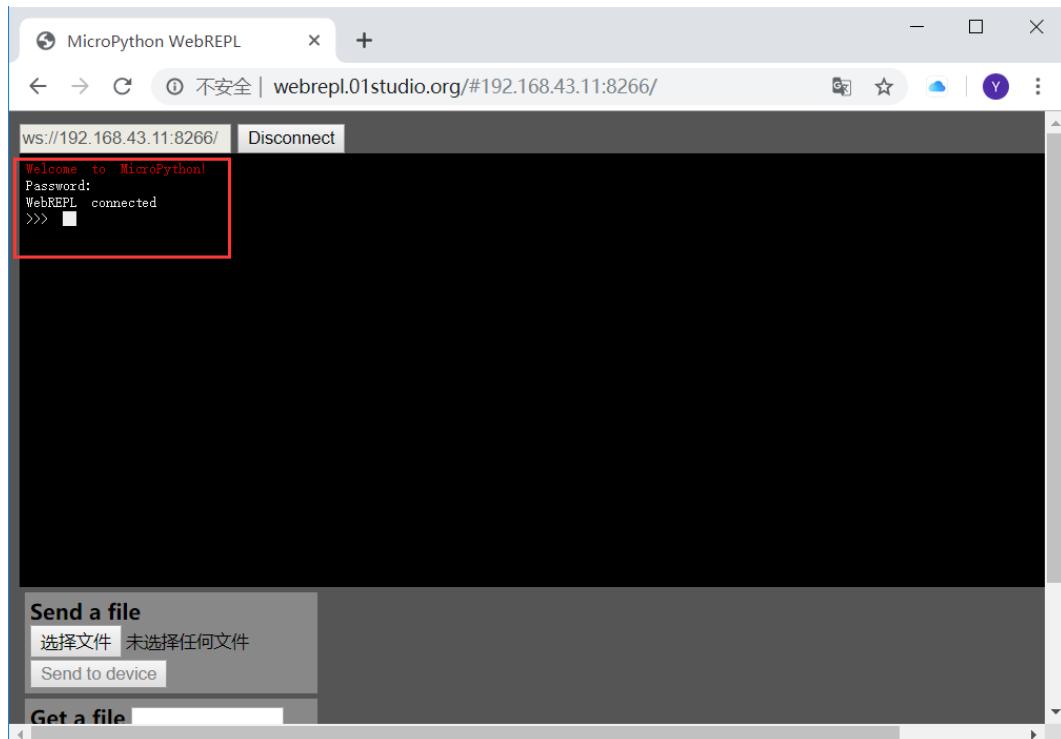


图 7-30 WebREPL 连接成功

第5步：文件传输

使用 WebREPL 网页下方提供文件传输功能，需要给设备发送文件时候，选择文件并发送。

除此之外也可以使用 Get a file 功能从设备中读取文件，注意要求输入文件名称完整。如“boot.py”。

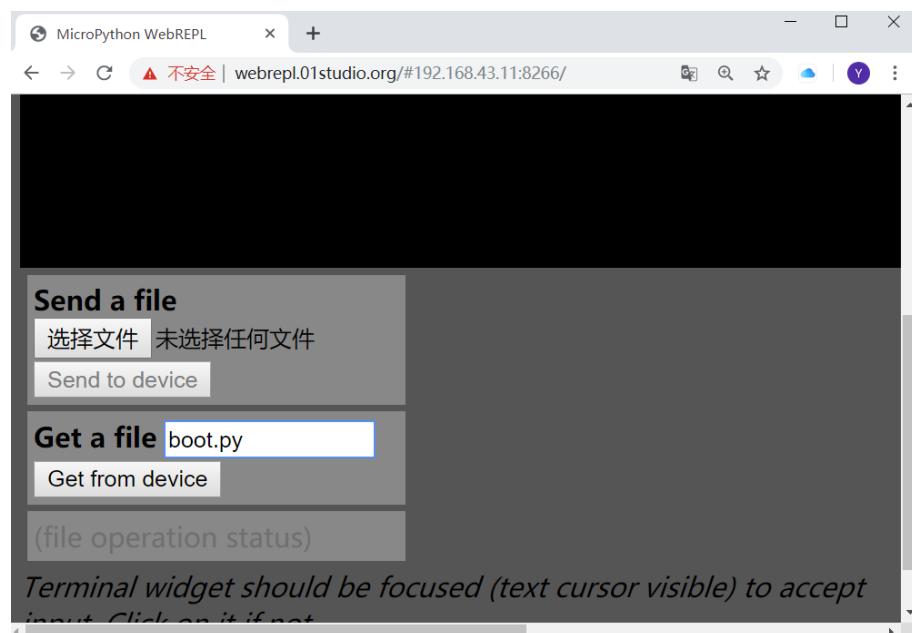


图 7-31 文件收发

第8章 LVGL (LittleVGL)

嵌入式设备开发离不开 GUI (图形用户界面)，漂亮的界面能让你的产品设备倍增光彩，然而在 MicroPython 下有一款漂亮、控件齐全且简单易用的 GUI 嵌入式开源库正在日益成熟，那就是 LVGL。

本章以应用为主，着重介绍 LVGL 常规控件的使用方法，让用户可以轻松学习并将控件部署到自己的产品中去。

本实验平台使用 pyWiFi-ESP32 或 pyWiFi-ESP32P，我们更推荐使用 pyWiFi-ESP32P，因为它搭载的主控模块是 ESP32-WROVER-B，外置 8M 的 PSRAM，有足够的内存来应对更复杂的 UI。

LCD 可以使用 O1Studio 的 2.4 寸或者 3.2 寸 LCD，这两款 LCD 均使用 ILI9341(液晶驱动)和 XPT2046(触摸驱动)，接口兼容，只是尺寸大小的差异。

为了达到更好的演示效果，本章使用 pyWiFi-ESP32P + 3.2 寸 LCD(电阻触摸)来进行实验讲解。



图 8-1 LVGL 官方图片

8.1 LVGL（LittleVGL）简介

LittlevGL 是一个免费的开源图形库，提供了创建嵌入式 GUI 所需的一切，具有易于使用的图形元素、漂亮的视觉效果和低内存占用。

- 占用资源少

64Kb Flash 和 8kB RAM 足以满足简单的用户界面需求。

- 多种窗口小部件

可从 30 多种小部件中进行选择，并轻松定义和使用。

- 多平台

可以在众多平台使用 LVGL，例如：STM32, NXP, PIC, Arduino, ESP32, Raspberry 等。

- 支持多种显示器

可驱动单色、OLED、TFT 显示器和其它显示器。

- 支持 MicroPython

可在 MicroPython 中使用 LVGL 轻松创建你的 UI。

- 开源、免费

LVGL 由 MIT 许可的方式托管在 GitHub 上，真正的开源免费使用。

英文官网：<https://lvgl.io/>

中文网站：<https://littlevgl.cn/>

官方文档（Docs）：<https://docs.lvgl.io/latest/en/html/index.html>

8.2 LCD 和触摸屏校准

- 前言：

LVGL 对每个平台的通用性是建立在已经定义好了 LCD 和触摸屏的参数，不同的 LCD 和触摸有不同的定义，我们通过本节实现了 LCD 显示和触摸校准后，就可以轻松进行后面的 UI 实验了。

- 实验平台：

pyWiFi-ESP32P + 3.2 寸 LCD 开发套件。



图 8-2 pyWiFi-ESP32P + 3.2 寸 LCD 开发套件

- 实验目的：

编程实现 LCD 的参数初始化以及触摸屏校准。

- 实验讲解：

LVGL 需要烧录专门的固件，固件路径位于零一科技 (01Studio) MicroPython 开发套件配套资料\latest\03-相关固件\03-pyWiFi-ESP32\02-LVGL 版(带 LittleVGL)目录下，烧录固件方法请参考：0

固件更新_章节内容，这里不再重复。

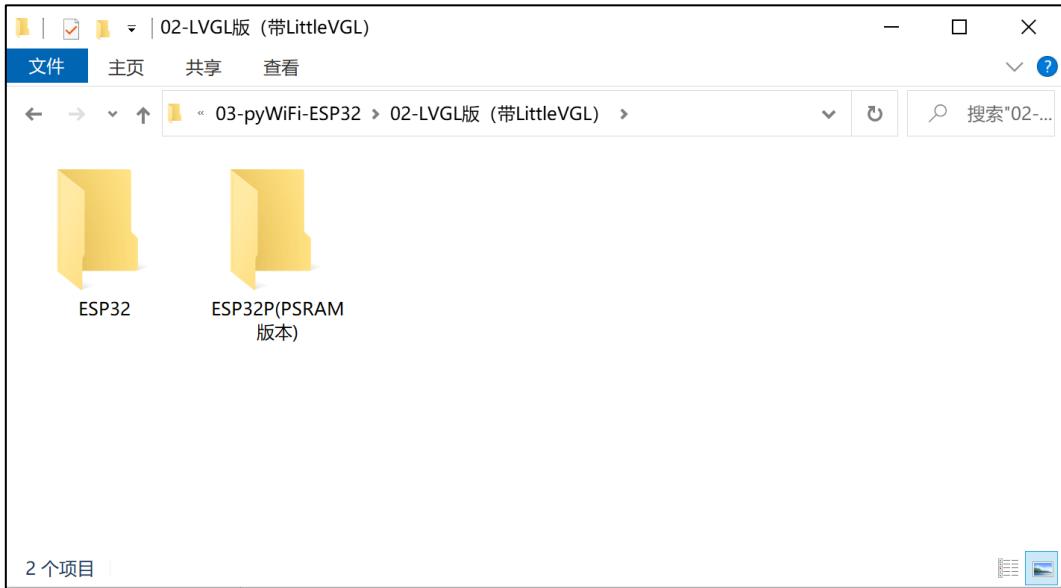


图 8-3 带 LVGL 的固件

01Studio 的 3.2 寸 LCD 电阻屏使用 ili9341+XPT2406(触摸)方案，我们先来看一下 LCD 和触摸屏的初始化函数用法。

LCD 初始化函数
disp = ili9341(miso, mosi, clk, cs, rst, width, height, rot)
LCD 初始化；
【miso】 SPI 引脚；
【mosi】 SPI 引脚；
【clk】 SPI 引脚；
【cs】 SPI 引脚，片选；
【rst】 复位
【width】 LCD 宽度像素
【height】 LCD 高度像素
【rot】 LCD 方向； ili9341.PORTRAIT：垂直， ili9341.LANDSCAPE：水平

表 8-1 LCD 初始化函数

触摸屏初始化函数

```
touch = xpt2046(cs, transpose, cal_x0, cal_x1, cal_y0, cal_y1)
```

触摸屏初始化。

【cs】 SPI 引脚，片选；

【transpose】 方向。1：竖屏，0：横屏。

【cal_x0, cal_x1, cal_y0, cal_y1】 触摸屏校准参数。

表 8-2 触摸屏初始化函数

在实际使用过程中我们并非每次都需要对触摸屏进行校准，正确的方式是将首次校准的参数保存为文件（json）存放在开发板中。以后上电时调用即可，同时应该支持横屏和竖屏的校准。

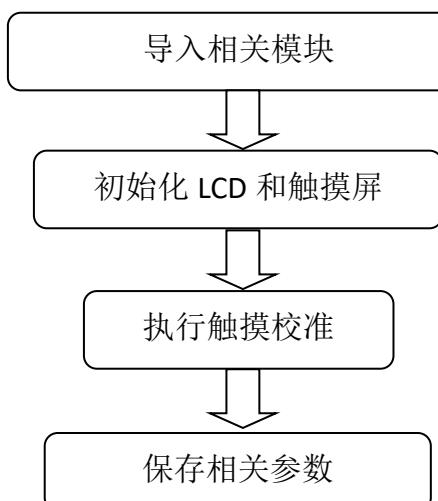


图 8-4 编程流程图

参考代码：

```
...
```

实验名称：LCD 初始化和触摸屏校准

版本：v1.0

日期：2020.7

作者: 01Studio 【www.01Studio.org】

...

```
import lvgl as lv
import time, ujson

from ili9341 import ili9341
from xpt2046 import xpt2046

TFT_IS_PORTRAIT = 1 #竖屏: 1 , 横屏: 0 ;
TOUCH_READY = 0 #用于检测触摸屏是否已经校准过;

#LCD ili9341 初始化
disp = ili9341(
    miso=12,
    mosi=13,
    clk=14,
    cs=15,
    dc=21,
    rst=33,
    power=50, #硬件不支持, 随便配一个参数
    backlight=51, #硬件不支持, 随便配一个参数
    backlight_on= 1,
    power_on= 1,
    width=240 if TFT_IS_PORTRAIT else 320,
    height=320 if TFT_IS_PORTRAIT else 240,
    rot=ili9341.PORTRAIT if TFT_IS_PORTRAIT else ili9341.LANDSCAPE
    #垂直方向 PORTRAIT ; 水平方向: LANDSCAPE
)
```

```
#触摸屏设置校准

TOUCH_CS = 2 #触摸屏 CS 片选引脚

TOUCH_INTERRUPT=0 #横屏


if TFT_IS_PORTRAIT:

    TOUCH_CALI_FILE = "touch_cali_PORTRAIT.json" #保存为竖屏触摸参数

else:

    TOUCH_CALI_FILE = "touch_cali_LANDSCAPE.json" #保存为横屏触摸参数


#从没做过触摸校准

if TOUCH_CALI_FILE not in uos.listdir():

    touch = xpt2046(
        cs=TOUCH_CS,
        transpose=TFT_IS_PORTRAIT,
    )

    from touch_cali import TouchCali

    touch_cali = TouchCali(touch, TOUCH_CALI_FILE)
    touch_cali.start()


#已经做过触摸校准，直接调用触摸参数文件

else:

    with open(TOUCH_CALI_FILE, 'r') as f:

        param = ujson.load(f)

        touch_x0 = param['cal_x0']
        touch_x1 = param['cal_x1']
        touch_y0 = param['cal_y0']
        touch_y1 = param['cal_y1']
```

```

touch = xpt2046(
    cs=TOUCH_CS,
    transpose=TFT_IS_PORTRAIT,
    cal_x0=touch_x0,
    cal_x1=touch_x1,
    cal_y0=touch_y0,
    cal_y1=touch_y1,
)

TOUCH_READY = 1 #表示已经配置好触摸参数

#####
##### 简单的触摸测试 #####
#####

if TOUCH_READY:

    print("LCD and TOUCH is ready!")

#简单的矩形块触摸拖动测试

obj = lv.obj(lv.scr_act())
obj.set_size(100,50)
obj.align(None, lv.ALIGN.CENTER, 0, 0)
obj.set_drag(True)

```

● 实验结果：

记得将例程中的 touch_cali.py 文件拷贝到开发板文件系统中，然后运行主函数代码，可以看到 LCD 提示触摸校准，我们按提示一次点击“+”标志来校准。

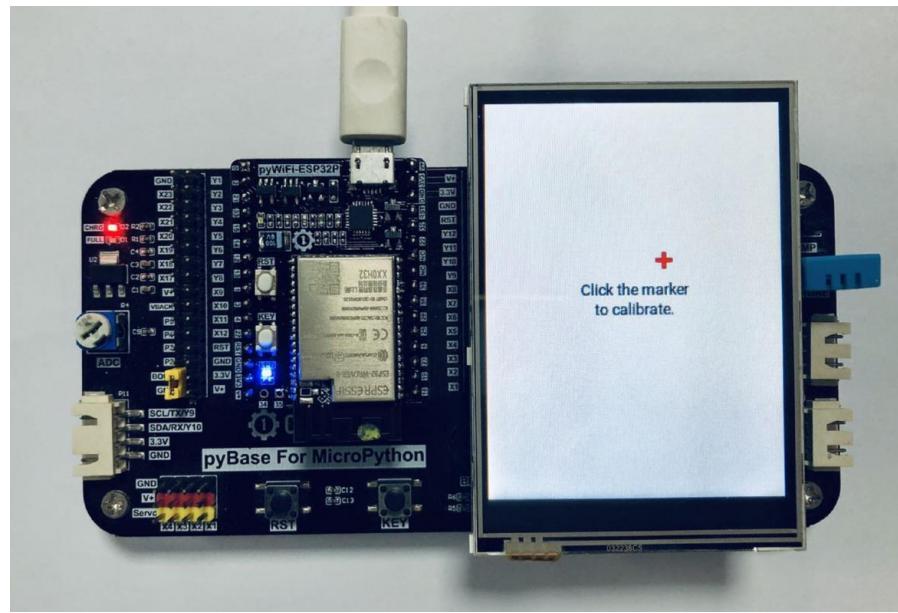


图 8-5 校准界面

校准完成后点击屏幕，开发板自动复位。（也可以直接按开发板 RST），这是我们在 Mu 中打开文件系统，可以看到多了一个“touch_cali_PORTRAIT.json”文件，这个就是刚刚记录的触摸参数文件，以后每次上电都会自动读取参数。

```

Mu 1.1.0.alpha.2 - main.py
无标题 x main.py x
1 """
2 实验名称: LCD初始化和触摸屏校准
3 版本: v1.0
4 日期: 2020.7
5 作者: 01Studio 【www.01Studio.org】
6 """
7
8 import lvgl as lv
Filesystem on ESP board
Files on your device:          在电脑上的文件:
boot.py                         main.py
touch_cali.py                    touch_cali.py
touch_cali_PORTRAIT.json         touch_cali_PORTRAIT.json

```

图 8-6 触摸参数文件

同时重新运行主程序后我们可以看到 LCD 出现一个方块，可以拖动用户测试触摸屏。

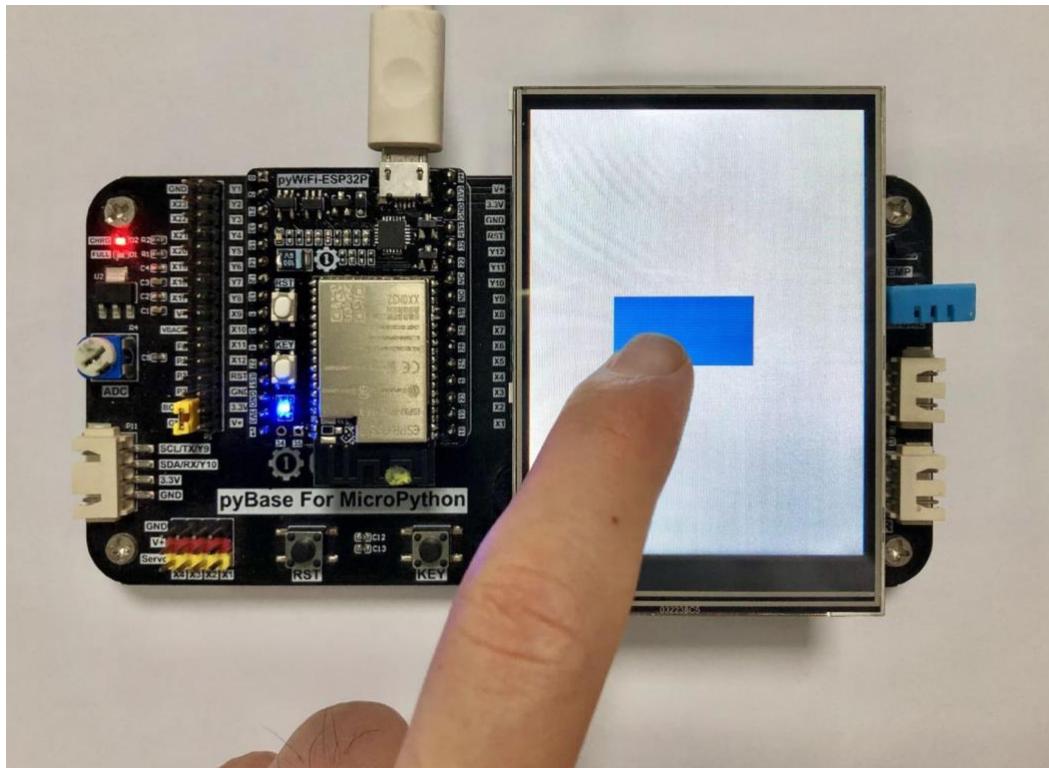


图 8-7 可以触摸拖动的方块

● 总结：

本节我们学习了最基础但最重要的 LCD 初始化和触摸校准，这取决于我们采用的硬件平台和液晶屏参数。不同初始化成功后，后面 LVGL 的使用方式和代码都是完全一样的，可移植性非常强。

8.3 小部件实验

一个完整的 UI 可以拆分成多个小部件（Widgets），如按钮、开关、菜单、列表等等，LVGL 目前拥有数十个小部件，本节将对每个小部件的功能和编程方法进行讲解。

8.3.1 Basic object (基础对象)

- 前言:

基础对象是最基础的部件，但它体现了小部件的一些基本属性，如坐标、父对象继承、风格、可点击和拖动等。

- 实验平台:

pyWiFi-ESP32P + 3.2 寸 LCD 开发套件。

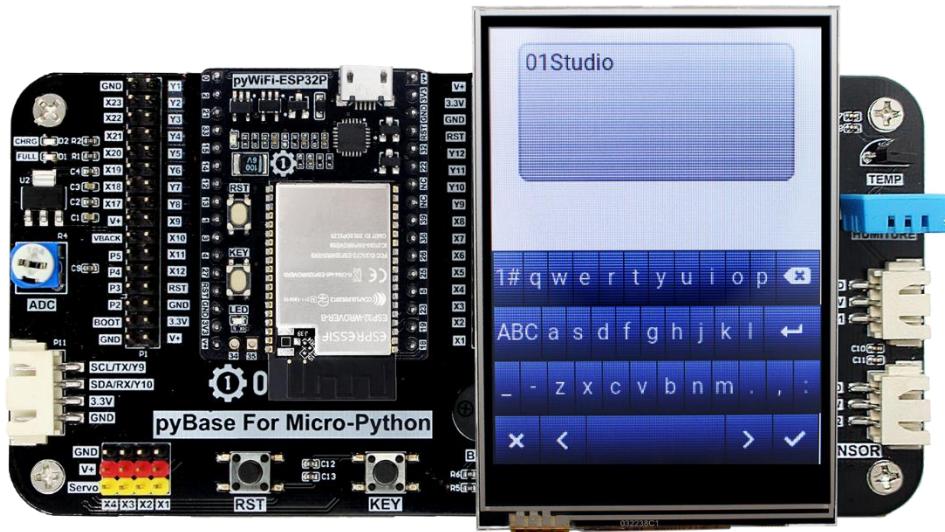


图 8-8 pyWiFi-ESP32P + 3.2 寸 LCD 开发套件

- 实验目的:

学习创建不同的基础对象。

- 实验讲解:

基础对象默认的就是矩形方块形状，可以通过编程赋予不同的显示风格和相关功能。

构造函数
<pre>obj = lv.obj(lv.scr_act())</pre>
构建 obj 对象； lv.scr_act() : 表示在当前屏幕创建；

使用方法	
<code>obj.set_size()</code>	设置尺寸；
<code>obj.set_style()</code>	设置风格；
<code>obj.align()</code>	设置对齐方式；
<code>obj.set_drag()</code>	设置可拖动；
更多用法请参阅小部件说明文档：（或在 REPL 中构建对象然后通过 tab 补全功能来查看。）	
https://docs.lvgl.io/latest/en/html/widgets/obj.html	

表 8-3 object 对象

编程思路如下：

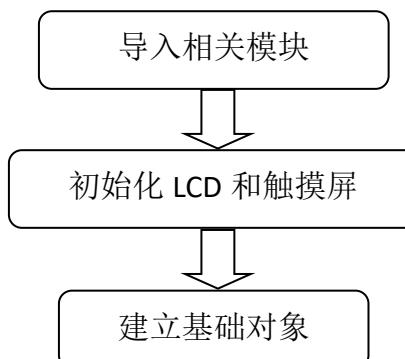


图 8-9 编程流程图

参考代码：（LCD 和触摸校准代码上一节已讲述，这里省略，完整代码参考配套资料包文件夹里面的示例程序）。

```

...
实验名称: Base object(基础对象)
版本: v1.0
日期: 2020.7
作者: 01Studio 【www.01Studio.org】
  
```

```
...
# 省略 LCD 初始化和触摸屏校准代码，详情请参考例程代码文件

#####
##### Basic object #####
#####

if TOUCH_READY:

    #基础对象 1
    obj1 = lv.obj(lv.scr_act())
    obj1.set_size(100,50)
    obj1.set_style(lv.style_plain_color)
    obj1.align(None, lv.ALIGN.CENTER, -60, -30)

    #基础对象 2，继承于基础对象 1，设置为可拖动
    obj2 = lv.obj(lv.scr_act(),obj1)
    obj2.set_style(lv.style_pretty_color)
    obj2.align(None, lv.ALIGN.CENTER, 0, 0)
    obj2.set_drag(True)

    #基础对象 3，继承于基础对象 2，新增风格
    style_shadow = lv.style_t()
    lv.style_copy(style_shadow, lv.style_pretty)
    style_shadow.body.shadow.width = 6
    style_shadow.body.radius = 800 # large enough to make it round

    obj3 = lv.obj(lv.scr_act(),obj2)
    obj3.set_style(style_shadow)
    obj3.align(None, lv.ALIGN.CENTER, 60, 30)
```

本例程创建了 3 个基础对象，obj2 继承于 obj1，obj3 继承于 obj2，其中 obj2 和 obj3 均可拖动。

● 实验结果：

将示例代码的 py 库文件拷贝到开发板文件系统（例程提供了触摸参数文件，可以直接使用，也可以自行触摸校准来生成）。

然后通过运行或者拷贝 main.py 到开发板方式运行程序。

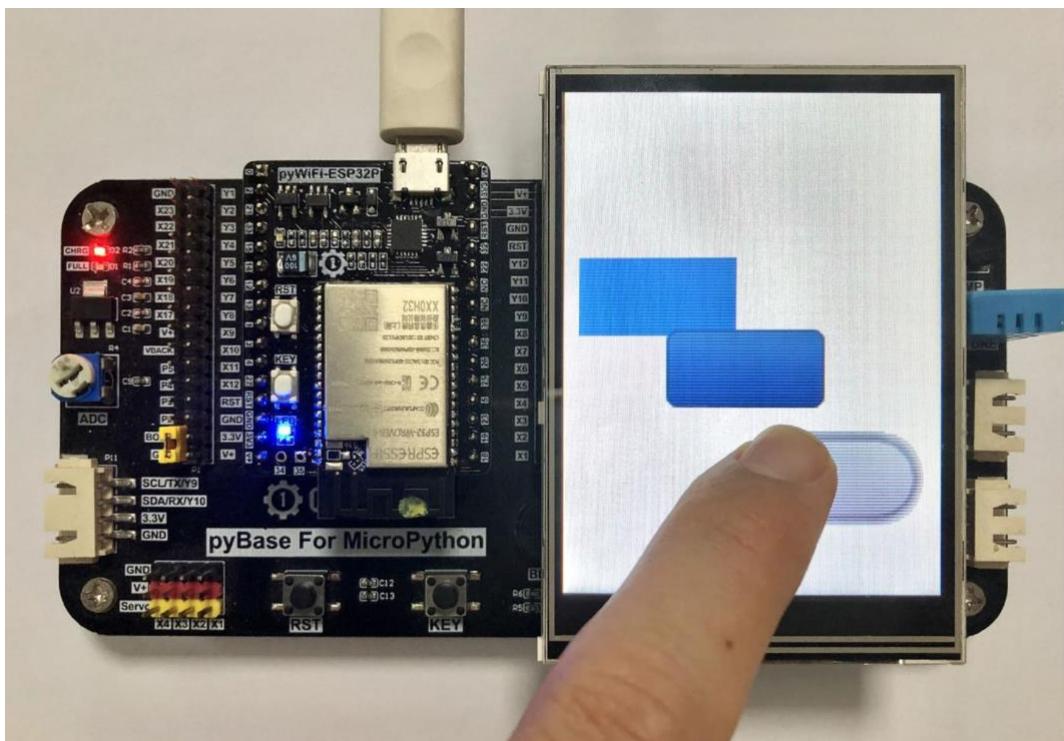


图 8-10 实验结果

● 总结

基础对象是最基础的小部件，通过本节的学习，让我们对 LVGL 的编程方法有了一定的了解。

8.3.2 Bar (进度条)

- 前言:

进度条常用于显示加载进度。

- 实验平台:

pyWiFi-ESP32P + 3.2 寸 LCD 开发套件。



图 8-11 pyWiFi-ESP32P + 3.2 寸 LCD 开发套件

- 实验目的:

创建进度条。

- 实验讲解:

我们来看看进度条对象。

构造函数

```
bar = lv.bar(lv.scr_act())
```

构建 bar 对象；

lv.scr_act() : 表示在当前屏幕创建；

使用方法

```
bar.set_size()
```

设置尺寸；
bar.align()
设置对齐方式；
bar.set_anim_time()
设置进度条加载时间；
bar.set_value(value,anim)
【value】 范围： 0-100， 表示进度条最大值；
【anim】
lv.ANIM.ON： 用动画设置值；
lv.ANIM.OFF： 立即更改值。
更多用法请参阅基小部件说明文档：（或在 REPL 中构建对象然后通过 tab 补全功能来查看。）
https://docs.lvgl.io/latest/en/html/widgets/bar.html

表 8-4 进度条对象

编程思路如下：

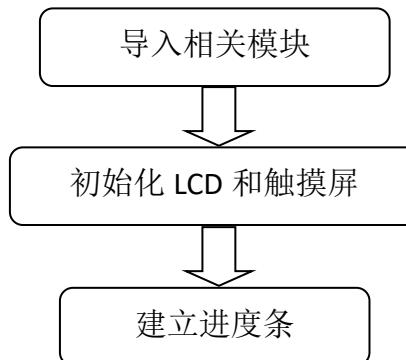


图 8-12 编程流程图

参考代码如下：

```

...
实验名称: Bar(进度条)
版本: v1.0
日期: 2020.7

```

作者: 01Studio 【www.01Studio.org】

```
...
# 省略 LCD 初始化和触摸屏校准代码, 详情请参考例程代码文件

#####
##### Bar #####
#####

if TOUCH_READY:

    bar = lv.bar(lv.scr_act()) #构建进度条对象
    bar.set_size(200, 30) #设置尺寸
    bar.align(None, lv.ALIGN.CENTER, 0, 0) #居中
    bar.set_anim_time(1000) #进度条加载时间
    bar.set_value(100, lv.ANIM.ON)
    #进度条最大值 100 (满), lv.ANIM.ON 表示动画效果。
```

本例程创建了 1 个进度条, 进度条值为满 100%, 动画效果。

● 实验结果:

运行代码, 可以看到 LCD 显示一个进度条。

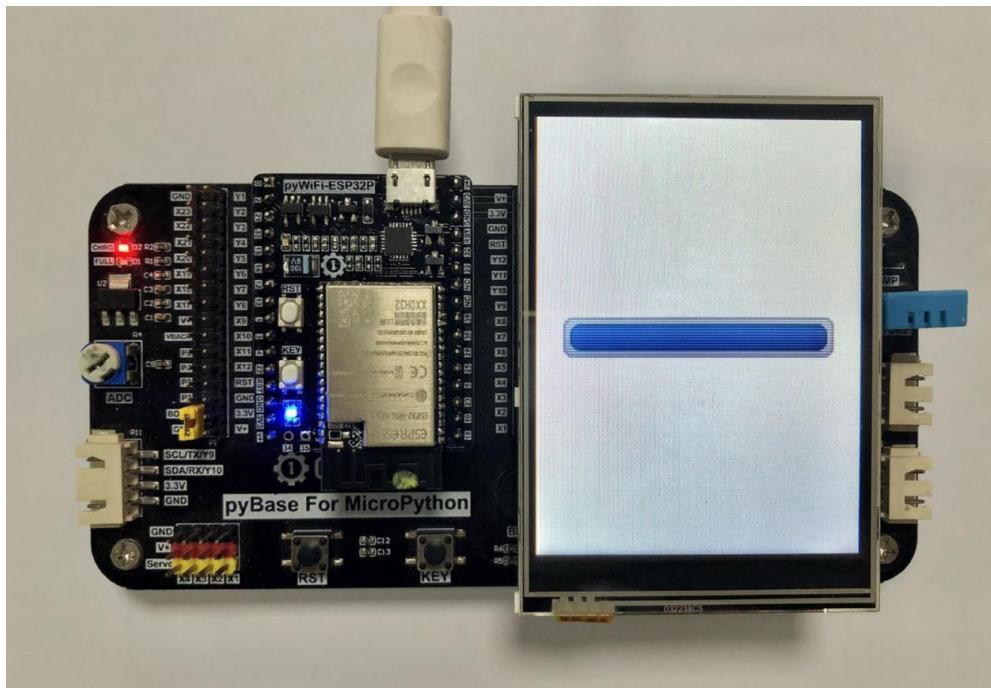


图 8-13 进度条实验结果

● 总结

本节主要讲解了 LVGL 中进度条的编程使用方法。

8.3.3 Button (按钮)

- 前言:

按钮是非常常用的交互工具部件。

- 实验平台:

pyWiFi-ESP32P + 3.2 寸 LCD 开发套件。

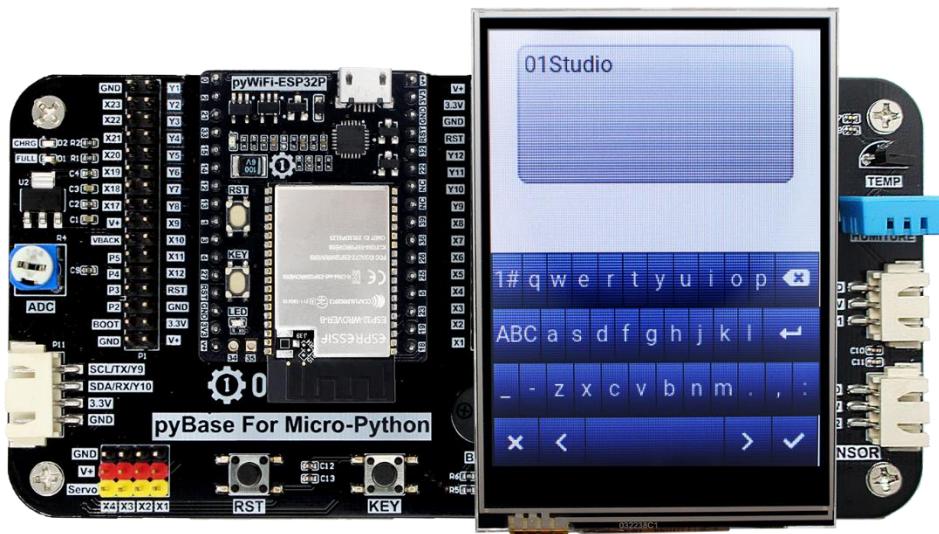


图 8-14 pyWiFi-ESP32P + 3.2寸LCD 开发套件

- 实验目的:

创建按钮。

- 实验讲解:

我们来看看按钮条对象。

构造函数

```
btn = lv.btn(lv.scr_act())
```

构建按钮对象；

lv.scr_act() : 表示在当前屏幕创建；

使用方法

```
btn.set_event_cb(callback)
```

定义按钮事件对应的回调函数;
<code>btn.align()</code>
设置对齐方式;
<code>btn.set_toggle(True)</code>
设置为状态型按钮;
更多用法请参阅小部件说明文档: (或在 REPL 中构建对象然后通过 tab 补全功能来查看。) https://docs.lvgl.io/latest/en/html/widgets(btn.html

表 8-5 按钮对象

编程思路如下:

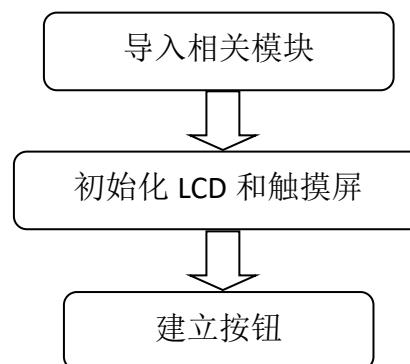


图 8-15 编程流程图

参考代码如下:

```

...
实验名称: Button(按钮)
版本: v1.0
日期: 2020.7
作者: 01Studio 【www.01Studio.org】
...
# 省略 LCD 初始化和触摸屏校准代码, 详情请参考例程代码文件
#####

```

```
##### Button #####
#####

#按钮回调函数

def event_handler(obj, event):
    if event == lv.EVENT.CLICKED:
        print("Clicked")

if TOUCH_READY:

    ##### 普通按钮 #####
    btn1 = lv.btn(lv.scr_act())
    btn1.set_event_cb(event_handler) #定义按钮触发的回调函数
    btn1.align(None, lv.ALIGN.CENTER, 0, -40)

    label = lv.label(btn1) #为 btn1 增加 label
    label.set_text("Button")

    ##### 开关状态型按钮 #####
    btn2 = lv.btn(lv.scr_act())
    # lambda 实现回调函数:
    btn2.set_event_cb(lambda obj, event: print("Toggled") if event ==
                      lv.EVENT.VALUE_CHANGED else None)
    btn2.align(None, lv.ALIGN.CENTER, 0, 40)
    btn2.set_toggle(True) #设置为状态按钮

    label = lv.label(btn2) #为 btn2 增加 label
    label.set_text("Toggled")
```

本例程创建了 2 个按钮，点击后可以在 REPL 打印出按钮操作相关信息。

● 实验结果：

运行代码，可以看到屏幕显示 2 个按钮。

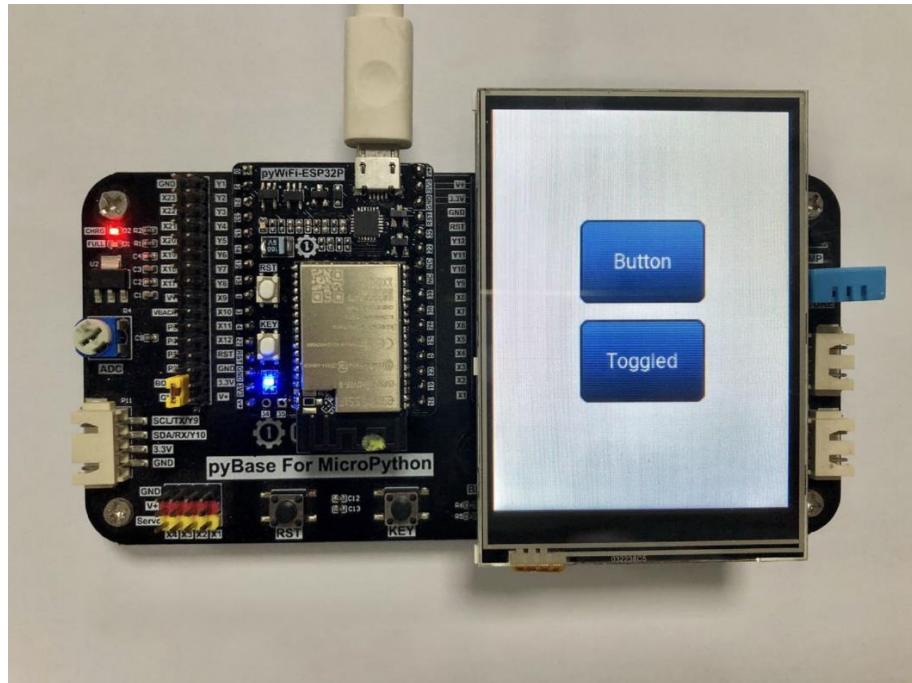


图 8-16 按钮显示

分别按 2 个按钮，可以看到串口 REPL 显示 click 和 toggle 提示信息：

```
1 """
2 实验名称: Button (按钮)
3 版本: v1.0
4 日期: 2020.7
5 作者: 01Studio 【www.01Studio.org】
6 """
7
8 import lvgl as lv
9 import time, ujson
10
11 from ili9341 import ili9341
ESP MicroPython REPL
MicroPython v1.9.4-2072-gf89c8c48c-dirty on 2020-01-17; ESP32 module
(spiram) with ESP32
Type "help()" for more information.
>>> Clicked
Toggled
```

● 总结

本节主要讲解了 LVGL 中按钮的编程使用方法。

8.3.4 Button Matrix (矩阵键盘)

- 前言:

矩阵键盘是一系列按钮的集合，如计算器、电脑键盘等。在 LVGL 下定义和使用非常方便。

- 实验平台:

pyWiFi-ESP32P + 3.2 寸 LCD 开发套件。

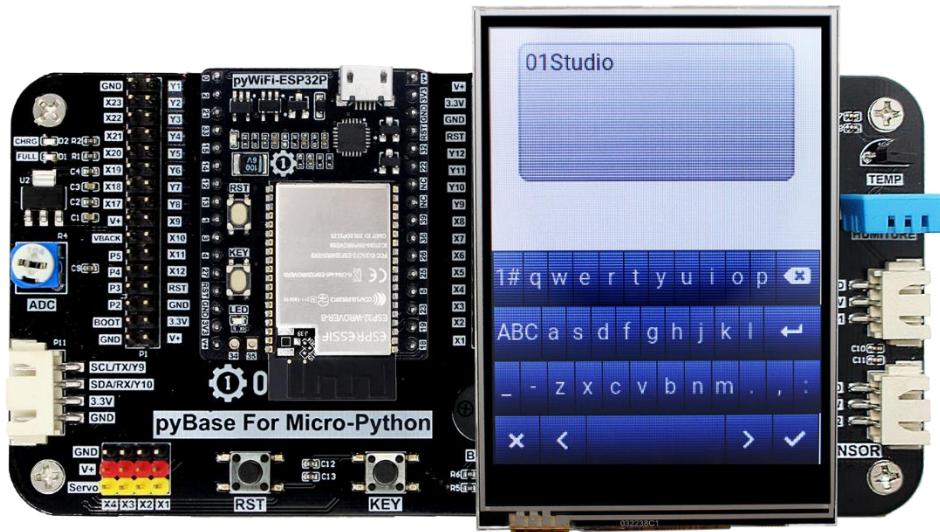


图 8-17 pyWiFi-ESP32P + 3.2 寸 LCD 开发套件

- 实验目的:

创建矩阵键盘。

- 实验讲解:

我们来看看矩阵键盘对象。

构造函数
<pre>btm = lv.btm(lv.scr_act())</pre>
构建矩阵键盘对象； lv.scr_act() : 表示在当前屏幕创建；
使用方法

<code>bbtnm.set_event_cb(callback)</code>
定义按钮事件对应的回调函数；
<code>bbtnm.align()</code>
设置对齐方式；
<code>bbtnm.set_map(bbtnm_map)</code>
定义矩阵键盘所有按钮；
<code>bbtnm.set_width()</code>
设置宽度
<code>bbtnm.set_btn_width(num,size)</code>
设置指定按钮的大小。 【num】第 num 个按钮，例如 10 表示第 10 个按钮； 【size】默认 1，例如 2 表示是默认的 2 倍尺寸
更多用法请参阅小部件说明文档：（或在 REPL 中构建对象然后通过 tab 补全功能来查看。） https://docs.lvgl.io/latest/en/html/widgets/btnmatrix.html

表 8-6 矩阵键盘对象

编程思路如下：

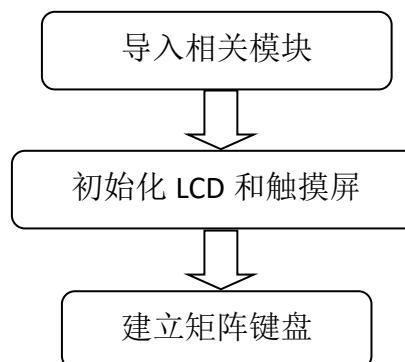


图 8-18 编程流程图

参考代码如下：

```

...
实验名称: Bar(进度条)
版本: v1.0
  
```

日期: 2020.7

作者: 01Studio 【www.01Studio.org】

...

```
# 省略 LCD 初始化和触摸屏校准代码, 详情请参考例程代码文件

#####
##### Button Matrix #####
#####

#矩阵键盘回调函数

def event_handler(obj, event):

    if event == lv.EVENT.VALUE_CHANGED:

        txt = obj.get_active_btn_text()

        print("%s was pressed" % txt)

if TOUCH_READY:

    #定义矩阵键盘, "\n"表示换行

    btnm_map = ["1", "2", "3", "4", "5", "\n",
                "6", "7", "8", "9", "0", "\n",
                "Action1", "Action2", ""]

    btnm1 = lv.btnm(lv.scr_act())

    btnm1.set_map(btnm_map)

    btnm1.set_btn_width(10, 2)

    # Make "Action1" twice as wide as "Action2"

    btnm1.set_width(240)

    btnm1.align(None, lv.ALIGN.CENTER, 0, 0)

    btnm1.set_event_cb(event_handler) #定义按钮事件回调函数
```

本例程创建了1个矩阵键盘, 点击后可以在REPL打印出按钮操作相关信息。

● 实验结果：

运行代码，可以看到屏幕显示 2 个按钮。

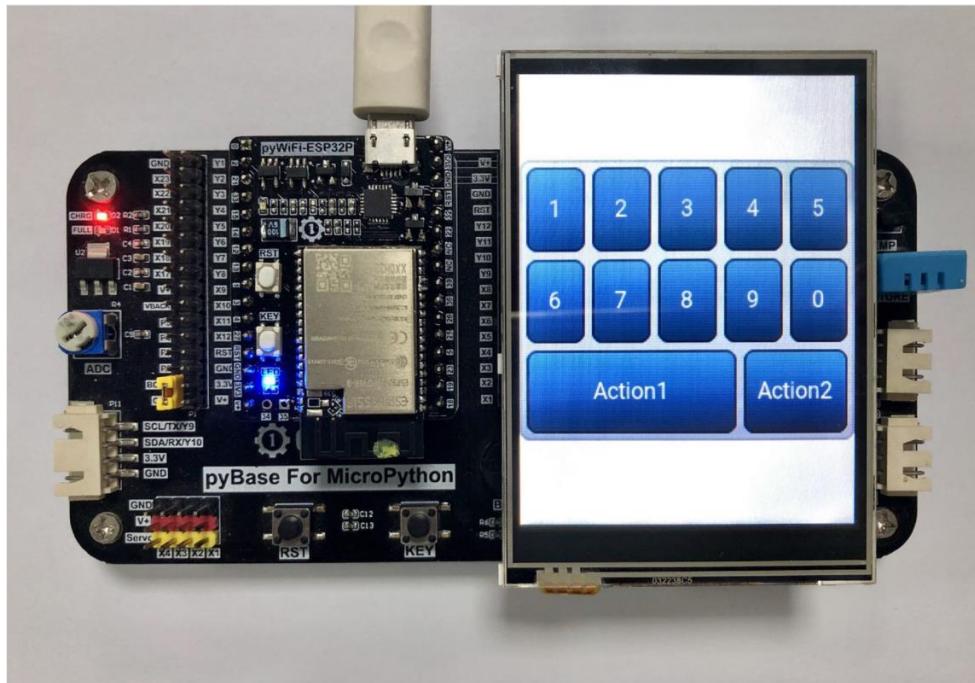


图 8-19 矩阵键盘显示

分别按 2 个按钮，可以看到串口 REPL 显示 click 和 toggle 提示信息：

```
43 #从没做过触摸校准
44 if TOUCH_CALI_FILE not in uos.listdir():
45     touch = xpt2046(
46         cs=TOUCH_CS,
47         transpose=TFT_IS_PORTRAIT,
48     )
49
50 from touch_cali import TouchCali
ESP MicroPython REPL
Type "help()" for more information.
>>> 1 was pressed
2 was pressed
4 was pressed
5 was pressed
6 was pressed
7 was pressed
8 was pressed
9 was pressed
0 was pressed
Action1 was pressed
Action2 was pressed
```

图 8-20 REPL 显示按钮出发信息

● 总结

本节主要讲解了 LVGL 中矩阵键盘的编程使用方法。

8.3.5 Calendar (日历)

- 前言:

日历就是常说的万年历，显示年、月、日、星期信息。

- 实验平台:

pyWiFi-ESP32P + 3.2 寸 LCD 开发套件。

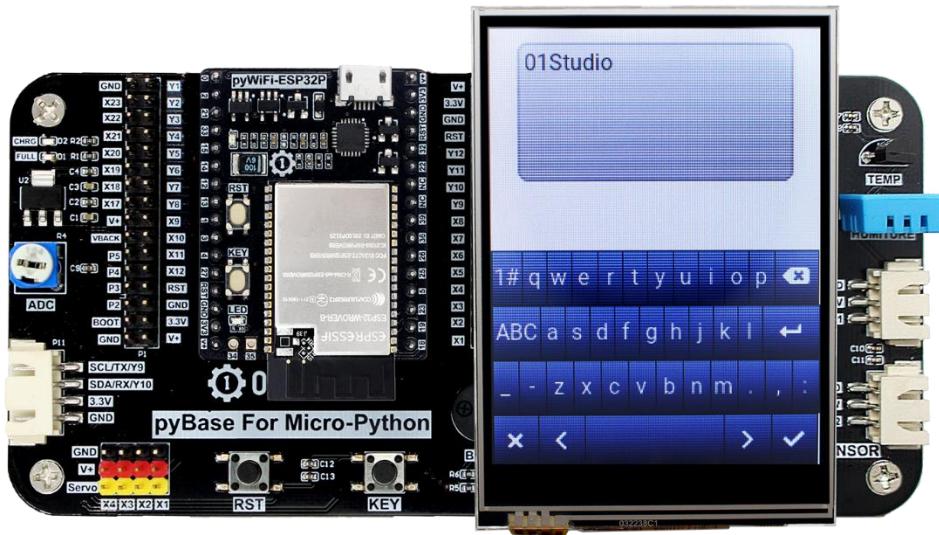


图 8-21 pyWiFi-ESP32P + 3.2寸LCD 开发套件

- 实验目的:

创建日历。

- 实验讲解:

我们来看看日历对象。

构造函数

```
calendar = lv.calendar(lv.scr_act())
```

构建日历对象；

lv.scr_act() : 表示在当前屏幕创建；

使用方法

```
calendar.set_event_cb(callback)
```

定义触摸事件对应的回调函数；
calendar.align()
设置对齐方式；
calendar.set_size(width,height)
定义显示尺寸。
calendar.set_today_date(today)
设置日期；需要构建 today 对象； 例：定义 2020 年 7 月 1 日 <pre>today = lv.calendar_date_t() today.year = 2020 today.month = 7 today.day = 1</pre>
calendar.set_showed_date(today)
显示指定日期。
calendar.set_highlighted_dates(highlighted_days, len(highlighted_days))
特定日期高亮状态显示。
更多用法请参阅小部件说明文档：（或在 REPL 中构建对象然后通过 tab 补全功能来查看。） https://docs.lvgl.io/latest/en/html/widgets/calendar.html

表 8-7 日期对象

编程思路如下：

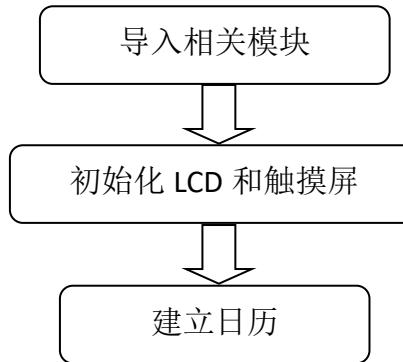


图 8-22 编程流程图

参考代码如下：

```
...
实验名称: Calendar(日历)
版本: v1.0
日期: 2020.7
作者: 01Studio 【www.01Studio.org】
...

# 省略 LCD 初始化和触摸屏校准代码，详情请参考例程代码文件
#####
##### Calendar(日历) #####
#####

#日历触摸回调函数

def event_handler(obj, event):
    if event == lv.EVENT.CLICKED:
        date = obj.get_pressed_date()
        if date is not None:
            obj.set_today_date(date) #显示按下选中的日期

    if TOUCH_READY:

        calendar = lv.calendar(lv.scr_act())
        calendar.set_size(230, 230)
        calendar.align(None, lv.ALIGN.CENTER, 0, 0)
        calendar.set_event_cb(event_handler)

    #设置当前日期
    today = lv.calendar_date_t() #构建日期对象
    today.year = 2020
    today.month = 7
    today.day = 1
```

```

calendar.set_today_date(today)

calendar.set_showed_date(today) #设置当前显示日期

#设置显示 3 个高亮日期

highlihted_days = [
    lv.calendar_date_t({'year':2020, 'month':7, 'day':10}),
    lv.calendar_date_t({'year':2020, 'month':7, 'day':15}),
    lv.calendar_date_t({'year':2020, 'month':7, 'day':20})
]

calendar.set_highlighted_dates(highlihted_days,len(highlihted_days))

```

● 实验结果：

运行代码，可以看到屏幕显示 2 个按钮。

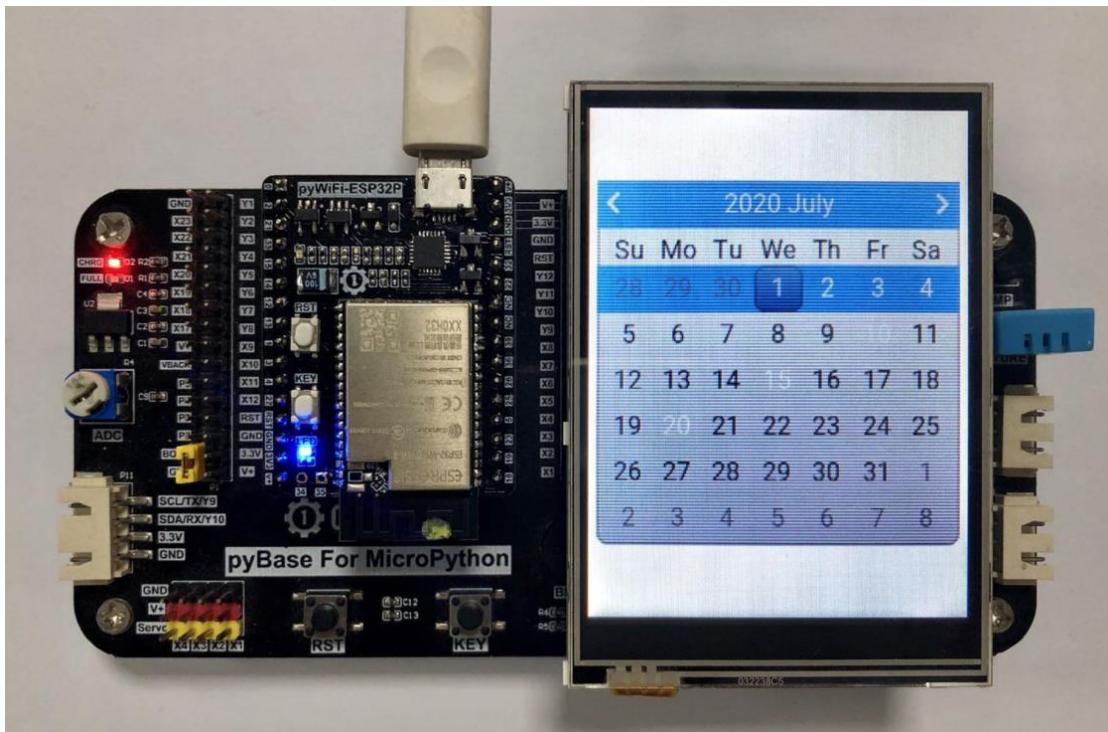


图 8-23 日期显示

● 总结

本节主要讲解了 LVGL 中日历小部件的编程使用方法。

8.3.6 CheckBox (复选框)

- 前言:

在做调查问卷中我们遇到多项选择题每一个选项就是一个复选框。

- 实验平台:

pyWiFi-ESP32P + 3.2 寸 LCD 开发套件。



图 8-24 pyWiFi-ESP32P + 3.2 寸 LCD 开发套件

- 实验目的:

创建复选框。

- 实验讲解:

我们来看看复选框对象。

构造函数
cb= lv.cb(lv.scr_act())
构建复选框对象； lv.scr_act() : 表示在当前屏幕创建；
使用方法
cb.set_event_cb(callback)

定义触摸事件对应的回调函数;
<code>cb.align()</code>
设置对齐方式;
<code>cb.set_text(string)</code>
设置复选框文本;
更多用法请参阅小部件说明文档: (或在 REPL 中构建对象然后通过 tab 补全功能来查看。) https://docs.lvgl.io/latest/en/html/widgets/checkbox.html

表 8-8 复选框对象

编程思路如下:

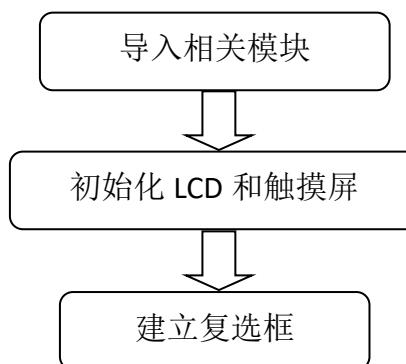


图 8-25 编程流程图

参考代码如下:

```

...
实验名称: CheckBox(复选框)
版本: v1.0
日期: 2020.7
作者: 01Studio 【www.01Studio.org】
...
# 省略 LCD 初始化和触摸屏校准代码, 详情请参考例程代码文件
#####
##### Checkbox #####
#####

```

```

def event_handler(obj, event):
    if event == lv.EVENT.VALUE_CHANGED:
        print("State: %s" % ("Checked" if obj.is_checked() else
"Unchecked"))

if TOUCH_READY:

    #构建复选框对象
    cb = lv.cb(lv.scr_act())
    cb.set_text("I agree !")
    cb.align(None, lv.ALIGN.CENTER, 0, 0)
    cb.set_event_cb(event_handler) #定义选中回调函数

```

● 实验结果：

运行代码，可以看到屏幕显示 2 个按钮。

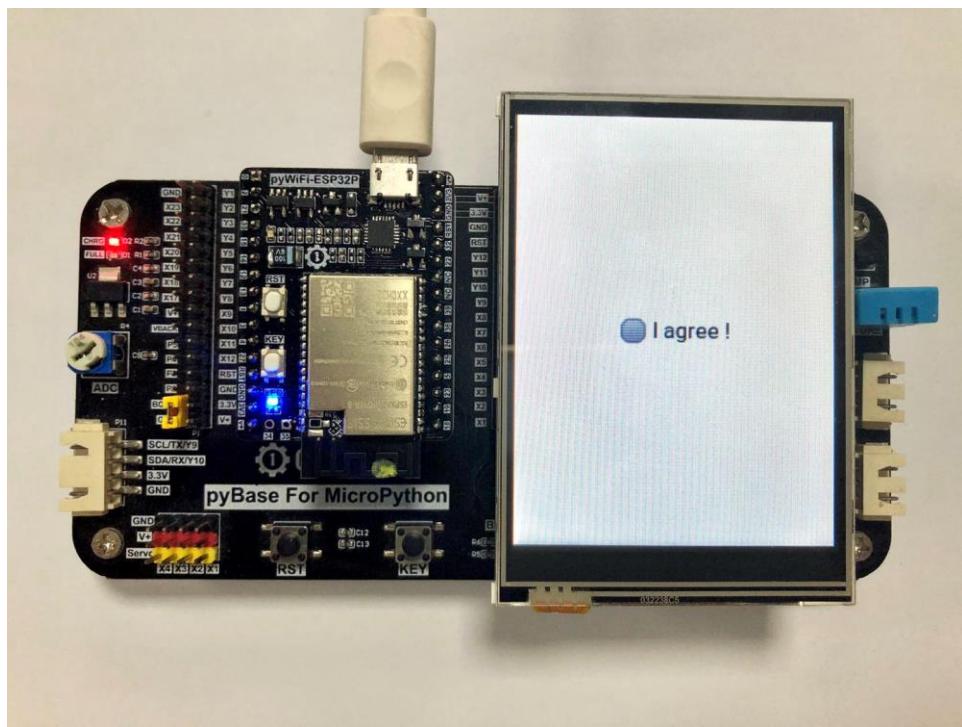


图 8-26 实验结果

● 总结

本节主要讲解了 LVGL 中复选框小部件的编程使用方法。

8.3.7 Chart (图表)

- 前言:

图表是常用的功能，如柱状图、折线图等绘制都用到图表。

- 实验平台:

pyWiFi-ESP32P + 3.2 寸 LCD 开发套件。

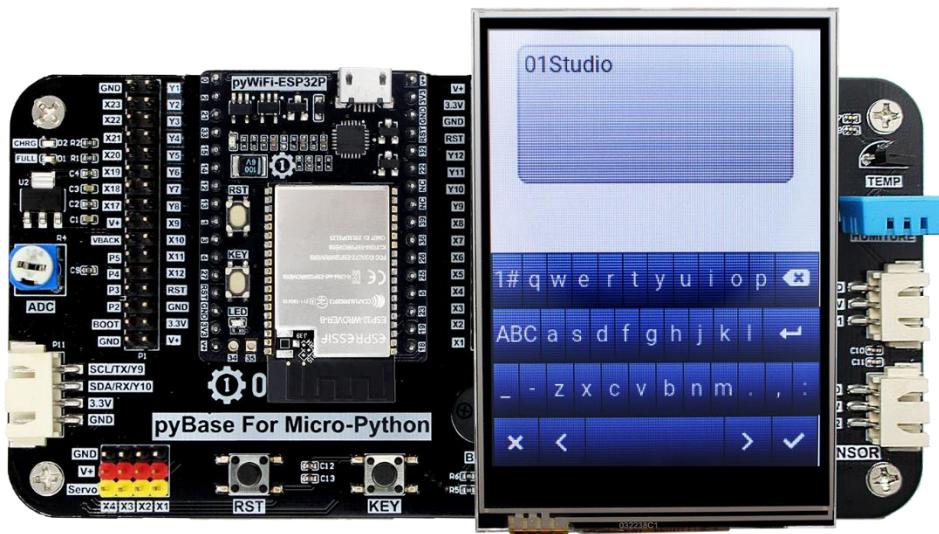


图 8-27 pyWiFi-ESP32P + 3.2 寸 LCD 开发套件

- 实验目的:

创建图表。

- 实验讲解:

我们来看看图表对象。

构造函数

```
chart= lv.chart(lv.scr_act())
```

构建图表对象；

lv.scr_act() : 表示在当前屏幕创建；

使用方法

```
chart.set_size (width,height)
```

定义尺寸。
chart.align()
设置对齐方式；
chart.set_type(lv.chart.TYPE.POINT lv.chart.TYPE.LINE)
设置图表显示类型；
例：lv.chart.TYPE.POINT lv.chart.TYPE.LINE 表示同时显示线和点。
chart.set_series_width(value)
设置线宽大小；
chart.set_range(0, 100)
设置纵坐标范围：0-100
ser1 = chart.add_series(lv.color_make(0xFF,0,0))
新建一个项目并设置颜色：lv.color_make(0xFF,0,0) 表示红色。
chart.set_points(ser1, [10, 10, 10, 10, 10, 10, 10, 10, 30, 70, 90])
设置项目的曲线值。
更多用法请参阅小部件说明文档：（或在 REPL 中构建对象然后通过 tab 补全功能来查看。）
https://docs.lvgl.io/latest/en/html/widgets/chart.html

表 8-9 图表对象

编程思路如下：

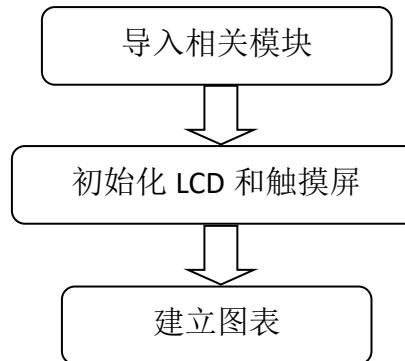


图 8-28 编程流程图

参考代码如下：

```
'''  
实验名称: Chart(图表)  
版本: v1.0  
日期: 2020.7  
作者: 01Studio 【www.01Studio.org】  
'''  
  
# 省略 LCD 初始化和触摸屏校准代码, 详情请参考例程代码文件  
#####  
##### Chart #####  
#####  
  
if TOUCH_READY:  
  
    # 创建图表  
  
        chart = lv.chart(lv.scr_act())  
        chart.set_size(200, 150)  
        chart.align(None, lv.ALIGN.CENTER, 0, 0)  
        # 同时显示线和点  
        chart.set_type(lv.chart.TYPE.POINT | lv.chart.TYPE.LINE)  
        chart.set_series_width(4)    # 线宽  
  
        chart.set_range(0, 100) #纵坐标范围  
  
        # Add two data series  
        ser1 = chart.add_series(lv.color_make(0xFF,0,0))  
        ser2 = chart.add_series(lv.color_make(0,0x80,0))  
  
        # Set points on 'd11'  
        chart.set_points(ser1, [10, 10, 10, 10, 10, 10, 10, 30, 70, 90])
```

```
# Set points on 'dl2'  
chart.set_points(ser2, [90, 70, 65, 65, 65, 65, 65, 65, 65, 65])
```

● 实验结果：

运行代码，可以看到屏幕显示图表出现 2 个曲线。

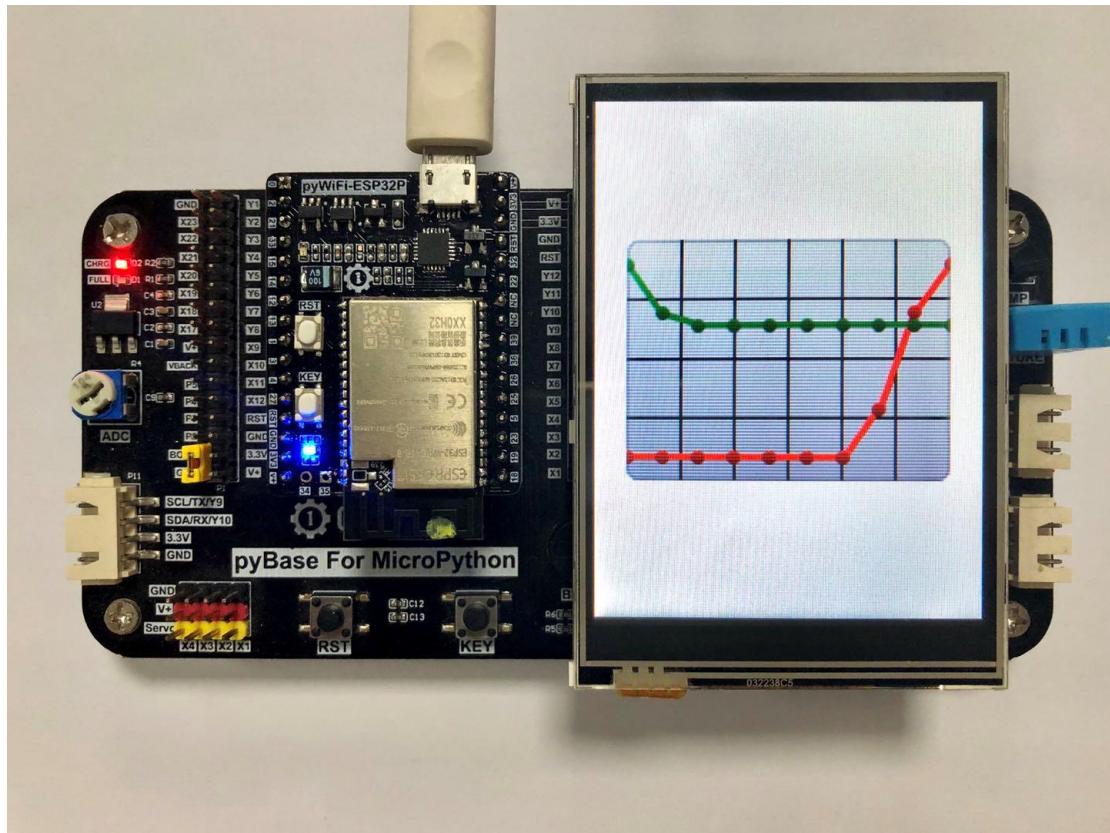


图 8-29 实验结果

● 总结

本节主要讲解了 LVGL 中图表的编程使用方法。

8.3.8 Container (容器)

- 前言:

容器可以用于存放多个文本或者小部件。

- 实验平台:

pyWiFi-ESP32P + 3.2 寸 LCD 开发套件。

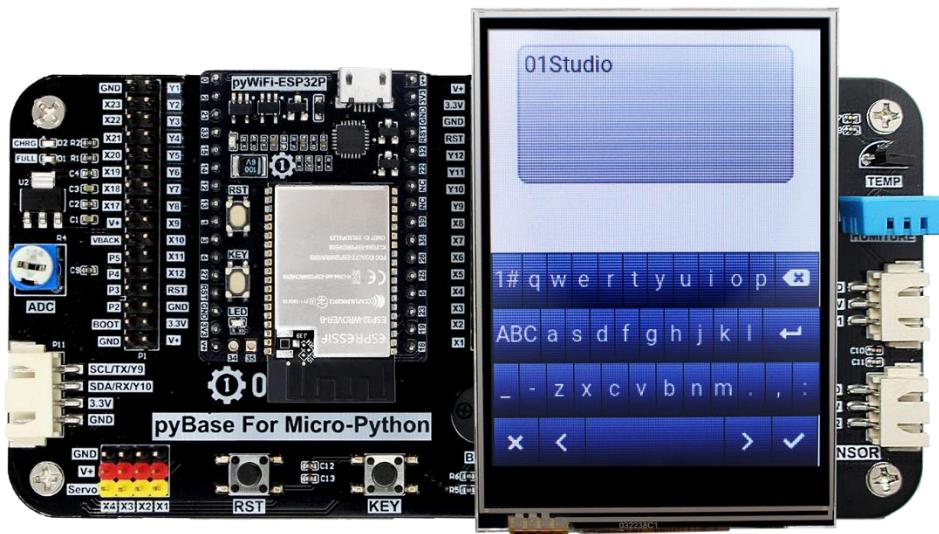


图 8-30 pyWiFi-ESP32P + 3.2 寸 LCD 开发套件

- 实验目的:

创建容器。

- 实验讲解:

我们来看看容器对象。

构造函数

```
cont = lv.cont(lv.scr_act())
```

构建容器对象；

lv.scr_act() : 表示在当前屏幕创建；

使用方法

```
cont.set_auto_realign(True)
```

自动调整尺寸
cont.align_origo()
设置对齐方式:
cont.set_fit(lv.FIT.TIGHT)
自动调整窗口大小:
cont.set_layout(lv.LAYOUT.COL_M)
自动调整容器内容布局;
更多用法请参阅小部件说明文档: (或在 REPL 中构建对象然后通过 tab 补全功能来查看。) https://docs.lvgl.io/latest/en/html/widgets/cont.html

表 8-10 容器对象

编程思路如下:

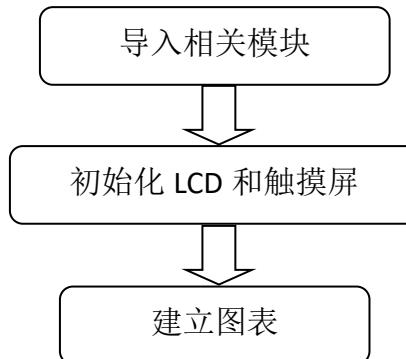


图 8-31 编程流程图

参考代码如下:

```

...
实验名称: Container(容器)
版本: v1.0
日期: 2020.7
作者: 01Studio 【www.01Studio.org】
...
# 省略 LCD 初始化和触摸屏校准代码, 详情请参考例程代码文件
  
```

```
#####
##### Cont #####
#####

if TOUCH_READY:

    cont = lv.cont(lv.scr_act())

    cont.set_auto_realign(True) # Auto realign when the size changes
    cont.align_origo(None, lv.ALIGN.CENTER, 0, 0)
    # This parameters will be used when realigned

    cont.set_fit(lv.FIT.TIGHT) #自动调整窗口大小
    cont.set_layout(lv.LAYOUT.COL_M) #自动调整容器内容布局

    label = lv.label(cont)
    label.set_text("Short text")

    label = lv.label(cont)
    label.set_text("It is a long text")

    label = lv.label(cont)
    label.set_text("Here is an even longer text")
```

● 实验结果：

运行代码，可以看到 3 个 label 信息都位于容器框内。



图 8-32 实验结果

● 总结

本节主要讲解了 LVGL 中容器的编程使用方法。

8.3.9 Drawdown List (下拉列表)

- 前言:

下拉列表可以提供非常方便的多项选择功能。

- 实验平台:

pyWiFi-ESP32P + 3.2 寸 LCD 开发套件。

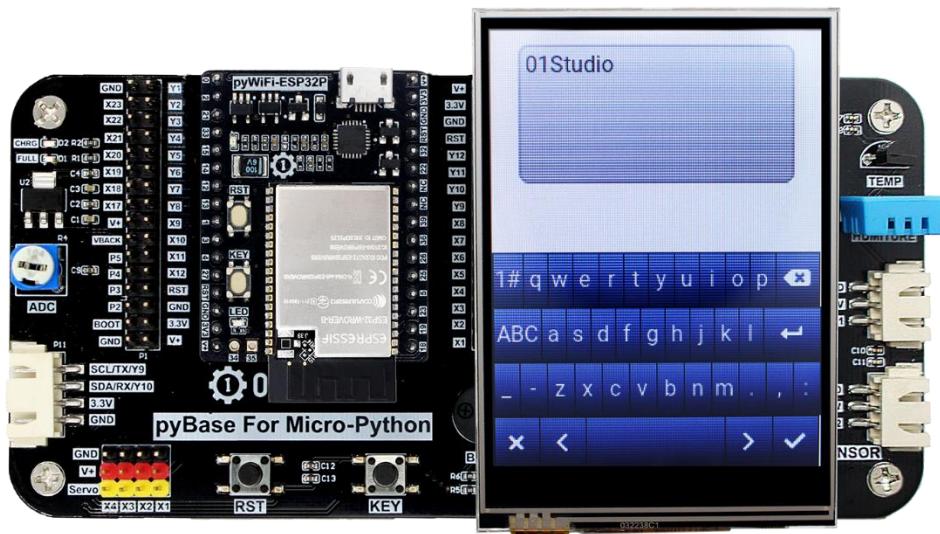


图 8-33 pyWiFi-ESP32P + 3.2 寸 LCD 开发套件

- 实验目的:

创建下拉列表。

- 实验讲解:

我们来看看下拉列表对象。

构造函数
<pre>ddlist = lv.ddlist(lv.scr_act())</pre>
构建下拉列表对象; lv.scr_act() : 表示在当前屏幕创建;
使用方法
<pre>ddlist.set_options()</pre>

设置下拉列表的选项;
<code>ddlist.set_draw_arrow(True)</code>
设置下拉列表箭头显示。
更多用法请参阅小部件说明文档：（或在 REPL 中构建对象然后通过 tab 补全功能来查看。）
https://docs.lvgl.io/latest/en/html/widgets/dropdown.html

表 8-11 下拉列表对象

编程思路如下：

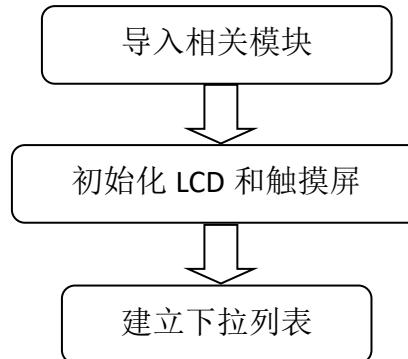


图 8-34 编程流程图

参考代码如下：

```

...
实验名称: Drawdown List(下拉列表)
版本: v1.0
日期: 2020.7
作者: 01Studio 【www.01Studio.org】
...
# 省略 LCD 初始化和触摸屏校准代码, 详情请参考例程代码文件
#####
##### Drawdown List #####
#####

```

```
def event_handler(obj, event):
    if event == lv.EVENT.VALUE_CHANGED:
        option = " "*10 # should be large enough to store the option
        obj.get_selected_str(option, len(option))
        # .strip() removes trailing spaces
        print("Option: \"%s\"" % option.strip())

if TOUCH_READY:

    # Create a drop down list
    ddlist = lv.ddlist(lv.scr_act())
    ddlist.set_options("\n".join([
        "Apple",
        "Banana",
        "Orange",
        "Melon",
        "Grape",
        "Raspberry"]))
    ddlist.set_fix_width(150)
    ddlist.set_draw_arrow(True)
    ddlist.align(None, lv.ALIGN.IN_TOP_MID, 0, 20)
    ddlist.set_event_cb(event_handler)
```

● 实验结果：

运行代码，可以看到 LCD 出现下拉列表。

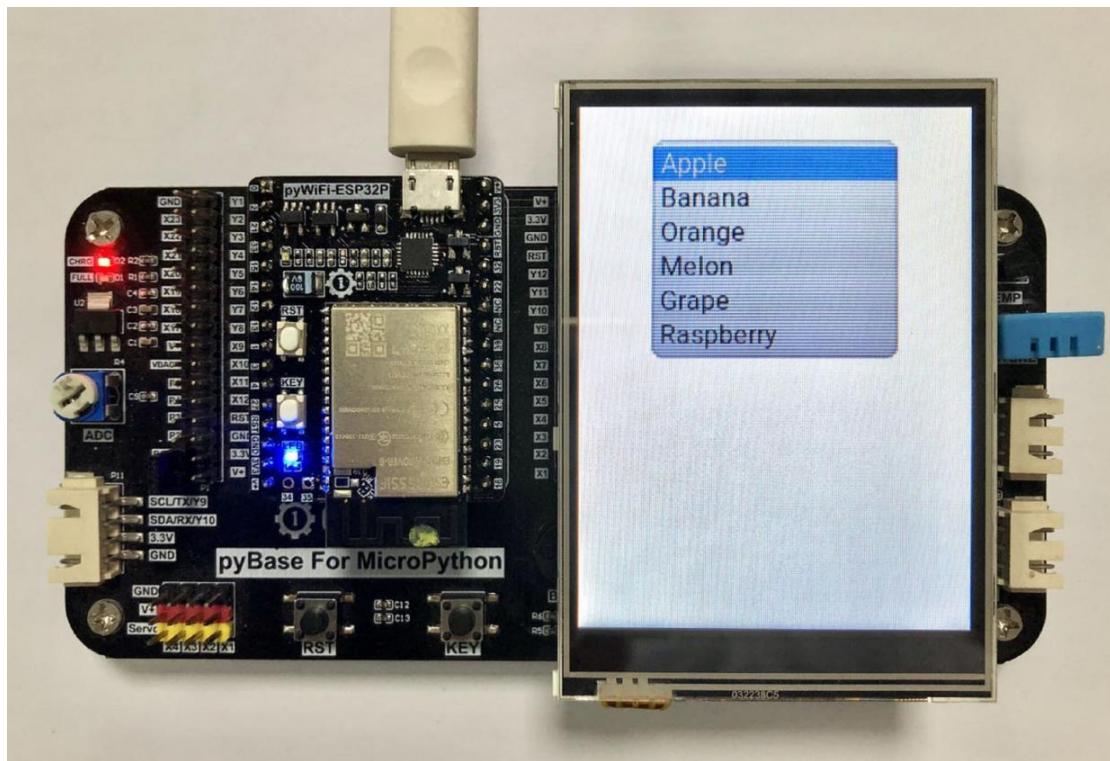


图 8-35 实验结果

● 总结

本节主要讲解了 LVGL 中下拉列表的编程使用方法。

8.3.10 Gauge (计量器)

- 前言:

计量器可以用于仪器仪表盘数据展示。

- 实验平台:

pyWiFi-ESP32P + 3.2 寸 LCD 开发套件。

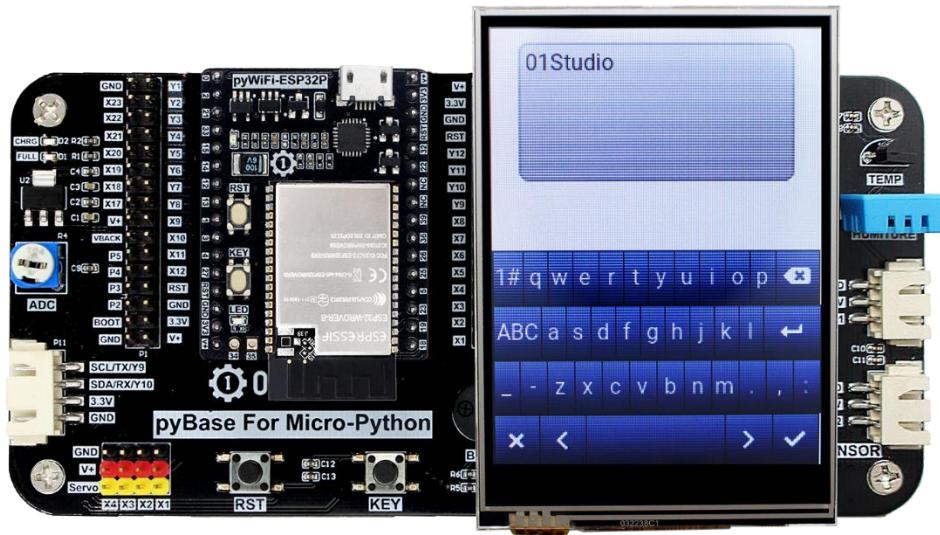


图 8-36 pyWiFi-ESP32P + 3.2 寸 LCD 开发套件

- 实验目的:

创建计量器。

- 实验讲解:

我们来看看计量器对象。

构造函数

```
gauge1 = lv.gauge(lv.scr_act())
```

构建计量器对象；

lv.scr_act() : 表示在当前屏幕创建；

使用方法

```
gauge1.set_style()
```

设置风格:
<code>gauge1.set_needle_count()</code>
设置指针数量和颜色;
<code>gauge1.set_value(num,value)</code>
设置指针值。 【num】指针编号; 【value】值。
更多用法请参阅小部件说明文档: (或在 REPL 中构建对象然后通过 tab 补全功能来查看。) https://docs.lvgl.io/latest/en/html/widgets/gauge.html

表 8-12 计量器对象

编程思路如下:

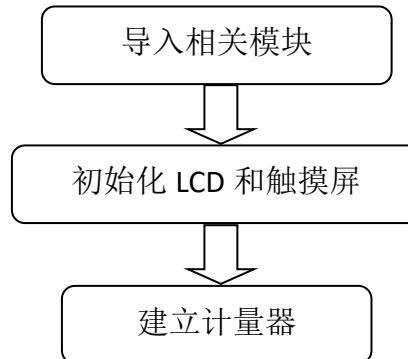


图 8-37 编程流程图

参考代码如下:

```

...
实验名称: Gauge(计量器)
版本: v1.0
日期: 2020.7
作者: 01Studio 【www.01Studio.org】
...
# 省略 LCD 初始化和触摸屏校准代码, 详情请参考例程代码文件
  
```

```

#####
##### Gauge #####
#####

if TOUCH_READY:

    # Create a style

    style = lv.style_t()

    lv.style_copy(style, lv.style_pretty_color)

    # Line color at the beginning

    style.body.main_color = lv.color_hex3(0x666)

    # Line color at the end

    style.body.grad_color = lv.color_hex3(0x666)

    style.body.padding.left = 10          # Scale line length

    style.body.padding.inner = 8         # Scale label padding

    # Needle middle circle color

    style.body.border.color = lv.color_hex3(0x333)

    style.line.width = 3

    style.text.color = lv.color_hex3(0x333)

    # Line color after the critical value

    style.line.color = lv.color_hex3(0xF00)

    # Describe the color for the needles

    needle_colors = [

        lv.color_make(0x00, 0x00, 0xFF),

        lv.color_make(0xFF, 0xA5, 0x00),

        lv.color_make(0x80, 0x00, 0x80)

    ]

    # Create a gauge

    gauge1 = lv.gauge(lv.scr_act())

    gauge1.set_style(lv.gauge.STYLE.MAIN, style)

```

```
gauge1.set_needle_count(len(needle_colors), needle_colors)
gauge1.set_size(150, 150)
gauge1.align(None, lv.ALIGN.CENTER, 0, 20)

# Set the values
gauge1.set_value(0, 10)
gauge1.set_value(1, 20)
gauge1.set_value(2, 30)
```

● 实验结果：

运行代码，可以看到计量器相关部件信息。

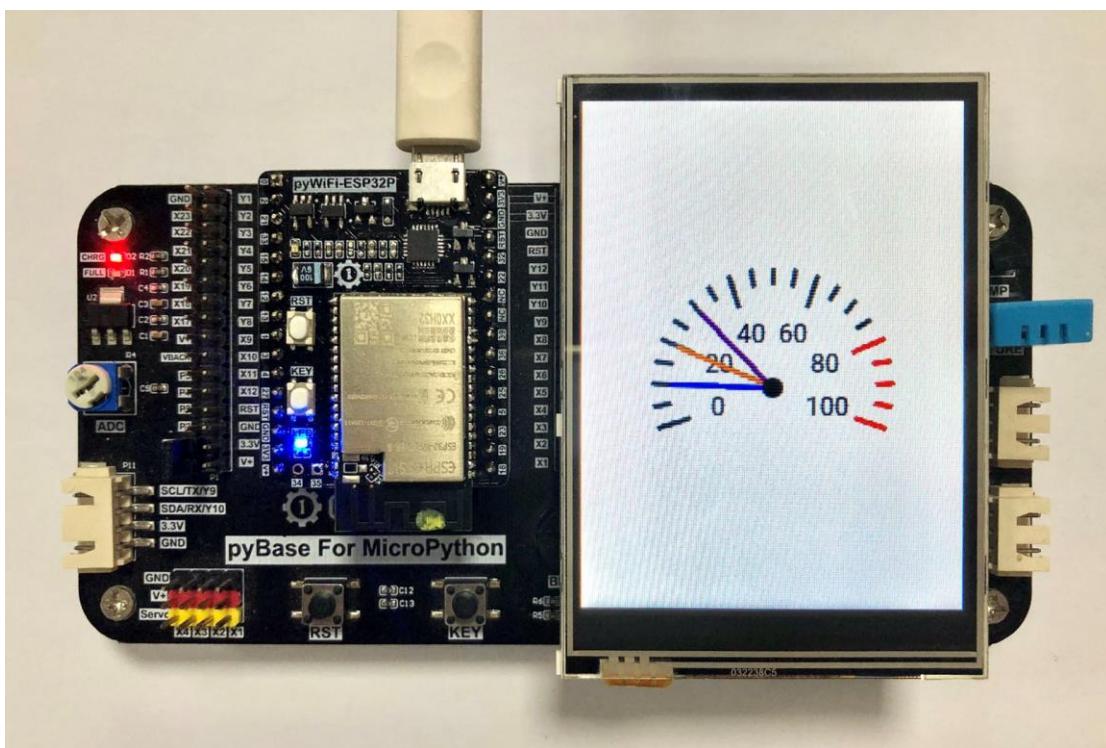


图 8-38 实验结果

● 总结

本节主要讲解了 LVGL 中计量器使用方法。

8.3.11 Keyboard (键盘)

- 前言：

键盘是最常见的输入交互工具。

- 实验平台：

pyWiFi-ESP32P + 3.2 寸 LCD 开发套件。



图 8-39 pyWiFi-ESP32P + 3.2 寸 LCD 开发套件

- 实验目的：

创建键盘。

- 实验讲解：

我们来看看键盘对象。

构造函数

```
kb = lv_kb(lv.scr_act())
```

构建键盘对象；

lv.scr_act() : 表示在当前屏幕创建；

使用方法

```
kb.set_cursor_manage(True)
```

设置光标闪烁;
<code>kb.set_style()</code>
设置键盘风格;
<code>kb.set_ta(ta)</code>
设置输入字符区域。 【ta】text area 字符区域对象;
更多用法请参阅小部件说明文档：（或在 REPL 中构建对象然后通过 tab 补全功能来查看。） https://docs.lvgl.io/latest/en/html/widgets/keyboard.html

表 8-13 键盘对象

编程思路如下：

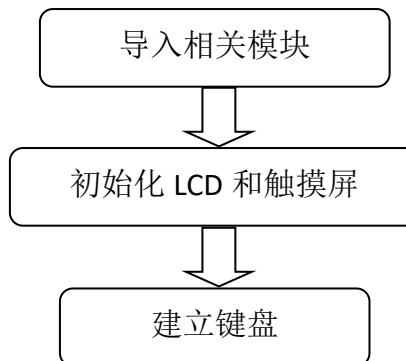


图 8-40 编程流程图

参考代码如下：

```

...
实验名称: Keyboard(键盘)
版本: v1.0
日期: 2020.7
作者: 01Studio 【www.01Studio.org】
...
# 省略 LCD 初始化和触摸屏校准代码, 详情请参考例程代码文件
#####

```

```

#####
# Keyboard #####
#####

if TOUCH_READY:

    # Create styles for the keyboard

    rel_style = lv.style_t()
    pr_style = lv.style_t()

    lv.style_copy(rel_style, lv.style_btn_rel)
    rel_style.body.radius = 0
    rel_style.body.border.width = 1

    lv.style_copy(pr_style, lv.style_btn_pr)
    pr_style.body.radius = 0
    pr_style.body.border.width = 1

    # Create a keyboard and apply the styles
    kb = lv.kb(lv.scr_act())
    kb.set_cursor_manage(True)
    kb.set_style(lv.kb.STYLE.BG, lv.style_transp_tight)
    kb.set_style(lv.kb.STYLEBTN_REL, rel_style)
    kb.set_style(lv.kb.STYLE_BTN_PR, pr_style)

    # Create a text area. The keyboard will write here
    ta = lv.ta(lv.scr_act())
    ta.align(None, lv.ALIGN_IN_TOP_MID, 0, 10)
    ta.set_text("")

    # Assign the text area to the keyboard
    kb.set_ta(ta)

```

- 实验结果：

运行代码，可以看到出现键盘部件，输入信息在文本框显示。



图 8-41 实验结果

- 总结

本节主要讲解了 LVGL 中键盘使用方法。

8.3.12 Label (标签)

- 前言:

标签可以理解为简单的标题或文字。

- 实验平台:

pyWiFi-ESP32P + 3.2 寸 LCD 开发套件。

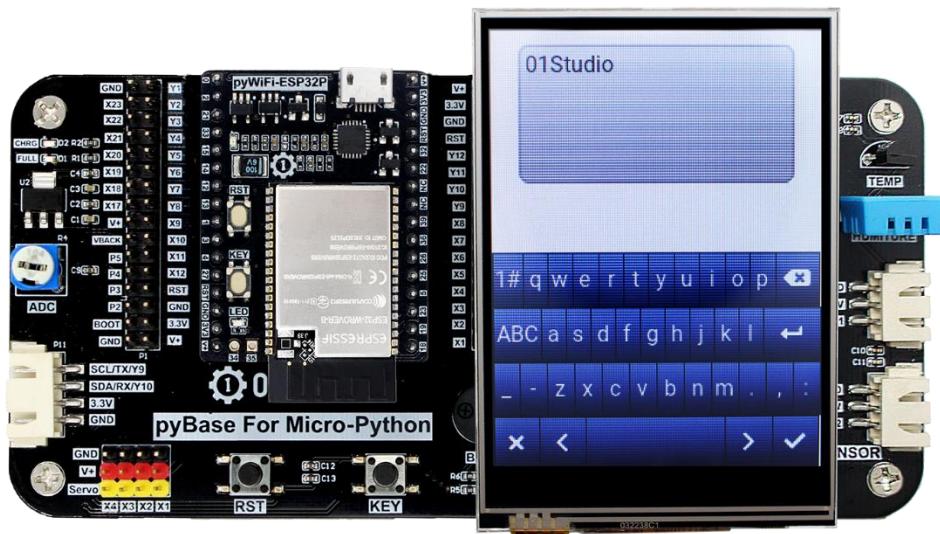


图 8-42 pyWiFi-ESP32P + 3.2 寸 LCD 开发套件

- 实验目的:

创建标签。

- 实验讲解:

我们来看看标签对象。

构造函数

```
label = lv.label(lv.scr_act())
```

构建标签对象；

lv.scr_act() : 表示在当前屏幕创建；

使用方法

```
label.set_long_mode(mode)
```

长文本模式；

【mode】

lv.label.LONG.BREAK：保持对象宽度，文本自动折断换行；

lv.label.LONG.SROLL_CIRC：循环滚动标签；

label.set_recolor(True)

允许通过文本指令修改颜色；

label.set_text(str)

设置标签显示的文本。当允许文本指令修改字符颜色时，可通过下面方法来设置文本颜色，如：`#0000ff Re-color#` 表示将 Re-color 设置成蓝色并显示。

label.align()

设置对齐方式。

更多用法请参阅小部件说明文档：（或在 REPL 中构建对象然后通过 tab 补全功能来查看。）

<https://docs.lvgl.io/latest/en/html/widgets/label.html>

表 8-14 标签对象

编程思路如下：

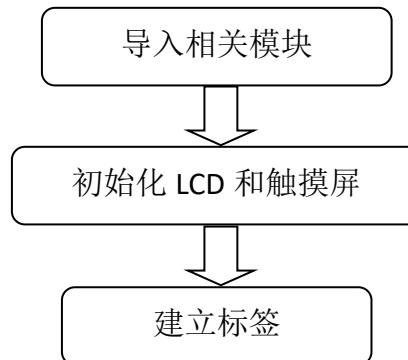


图 8-43 编程流程图

参考代码如下：

...

实验名称：Label(标签)

版本：v1.0

```
日期: 2020.7

作者: 01Studio 【www.01Studio.org】

```
省略 LCD 初始化和触摸屏校准代码, 详情请参考例程代码文件
#####
Label
#####

if TOUCH_READY:

 label1 = lv.label(lv.scr_act())
 label1.set_long_mode(lv.label.LONG.BREAK) # Break the long lines
 # Enable re-coloring by commands in the text
 label1.set_recolor(True)
 label1.set_align(lv.label.ALIGN.CENTER) # Center aligned lines
 label1.set_text("#0000ff Re-color# #00ff00 words# #ff0000 of a#
 label " + "and wrap long text automatically.")
 label1.set_width(150)
 label1.align(None, lv.ALIGN.CENTER, 0, -30)

 label2 = lv.label(lv.scr_act())
 label2.set_long_mode(lv.label.LONG.SROLL_CIRC) # Circular scroll
 label2.set_width(150)
 label2.set_text("It is a circularly scrolling text. ")
 label2.align(None, lv.ALIGN.CENTER, 0, 30)
```

## ● 实验结果:

运行代码, 可以看到 LCD 出现编程实现的标签小部件。

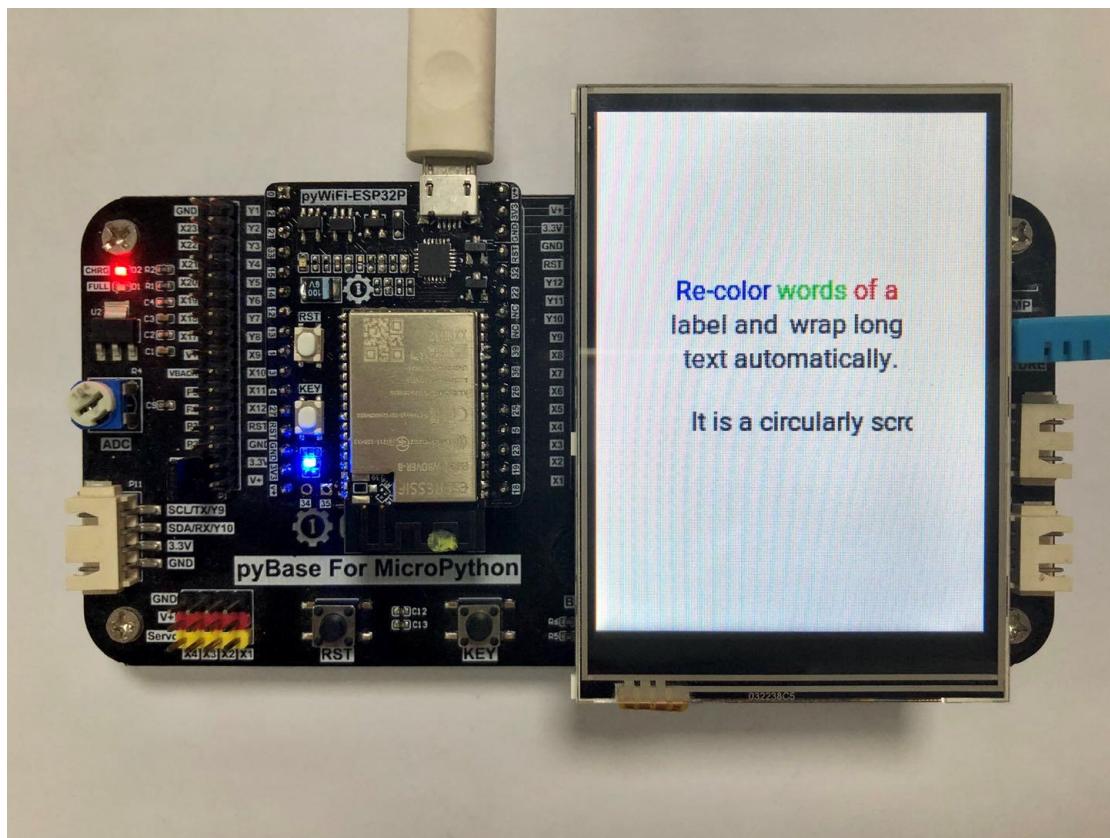


图 8-44 实验结果

### ● 总结

本节主要讲解了 LVGL 中标签使用方法。

### 8.3.13 Line (线)

- 前言:

线部件可以通过定义各个点来绘制折线图。

- 实验平台:

pyWiFi-ESP32P + 3.2 寸 LCD 开发套件。



图 8-45 pyWiFi-ESP32P + 3.2 寸 LCD 开发套件

- 实验目的:

创建线部件。

- 实验讲解:

我们来看看线对象。

#### 构造函数

```
line = lv.line(lv.scr_act())
```

构建线对象；

lv.scr\_act() : 表示在当前屏幕创建；

#### 使用方法

```
line.set_points(line_points, len(line_points))
```

## 设置线段

【line\_points】设置线经过的点，例如经过 2 个点：

```
line_points=[{"x":5, "y":5},
 {"x":10, "y":10}]
```

【len(line\_points)】对应点的数量。

line.set\_style()

设置风格；

line.align()

设置对齐方式；

更多用法请参阅小部件说明文档：（或在 REPL 中构建对象然后通过 tab 补全功能来查看。）

<https://docs.lvgl.io/latest/en/html/widgets/line.html>

表 8-15 线对象

编程思路如下：

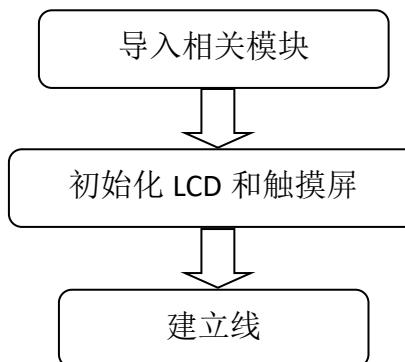


图 8-46 编程流程图

参考代码如下：

```
...
实验名称: Line(线)
版本: v1.0
日期: 2020.7
作者: 01Studio 【www.01Studio.org】
```

```
'''
省略 LCD 初始化和触摸屏校准代码，详情请参考例程代码文件
Line #####

if TOUCH_READY:

 # Create an array for the points of the line
 line_points = [{"x":5, "y":5},
 {"x":70, "y":70},
 {"x":120, "y":10},
 {"x":180, "y":60},
 {"x":240, "y":10}]

 # Create new style (thick dark blue)
 style_line = lv.style_t()
 lv.style_copy(style_line, lv.style_plain)
 style_line.line.color = lv.color_make(0x00, 0x3b, 0x75)
 style_line.line.width = 3
 style_line.line.rounded = 1

 # Copy the previous line and apply the new style
 line1 = lv.line(lv.scr_act())
 line1.set_points(line_points, len(line_points)) # Set the points
 line1.set_style(lv.line.STYLE.MAIN, style_line)
 line1.align(None, lv.ALIGN.CENTER, 0, 0)
```

## ● 实验结果：

运行代码，可以看到 LCD 出现线小部件。

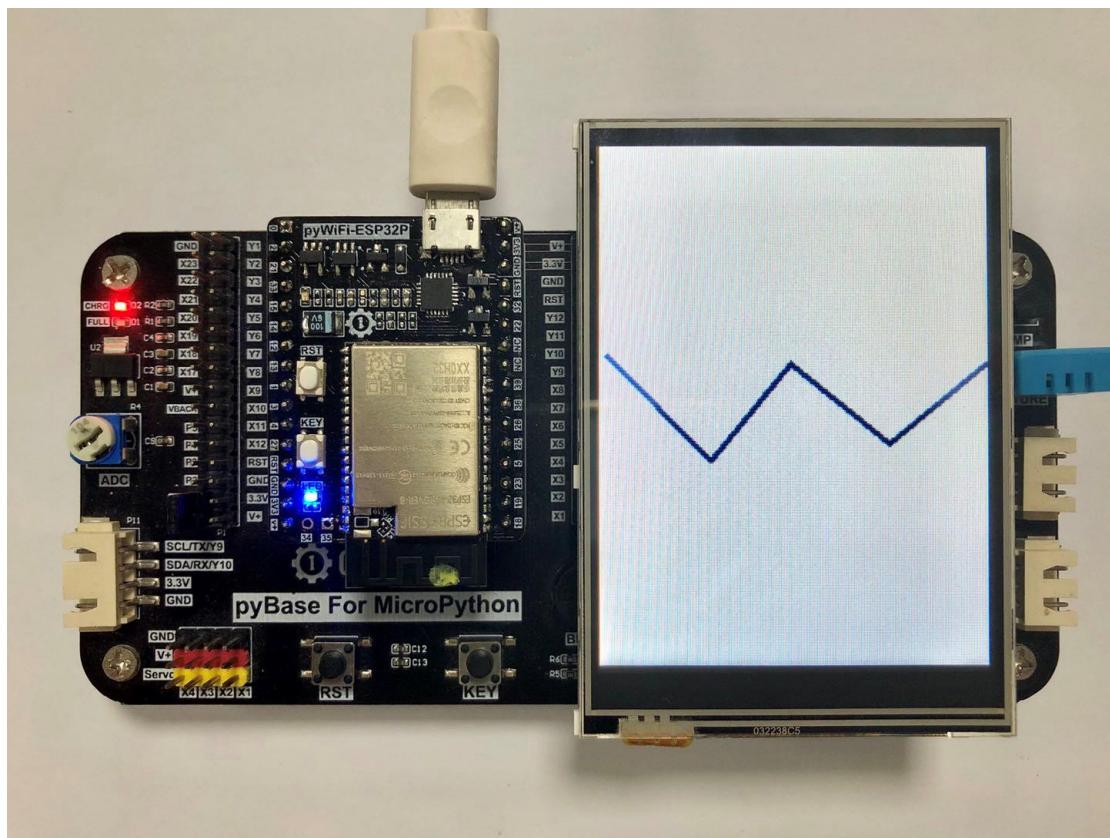


图 8-47 实验结果

### ● 总结

本节主要讲解了 LVGL 中线的使用方法。

### 8.3.14 List (菜单)

- 前言:

菜单可以完成方便快捷的功能选择。

- 实验平台:

pyWiFi-ESP32P + 3.2 寸 LCD 开发套件。



图 8-48 pyWiFi-ESP32P + 3.2 寸 LCD 开发套件

- 实验目的:

创建菜单。

- 实验讲解:

我们来看看菜单对象。

#### 构造函数

```
list = lv.list(lv.scr_act())
```

构建菜单对象；

lv.scr\_act() : 表示在当前屏幕创建；

#### 使用方法

```
list.set_size(width, height)
```

|                                                                                                                                                                               |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 设置菜单尺寸:<br>【width】宽;<br>【height】高。                                                                                                                                            |
| list.aline()                                                                                                                                                                  |
| 设置对齐方式;                                                                                                                                                                       |
| list_btn = list.add_btn()                                                                                                                                                     |
| 为菜单增加按钮                                                                                                                                                                       |
| list_btn.set_event_cb(callback)                                                                                                                                               |
| 新增按钮按下的回调函数                                                                                                                                                                   |
| 更多用法请参阅小部件说明文档: (或在 REPL 中构建对象然后通过 tab 补全功能来查看。)<br><a href="https://docs.lvgl.io/latest/en/html/widgets/list.html">https://docs.lvgl.io/latest/en/html/widgets/list.html</a> |

表 8-16 菜单对象

编程思路如下:

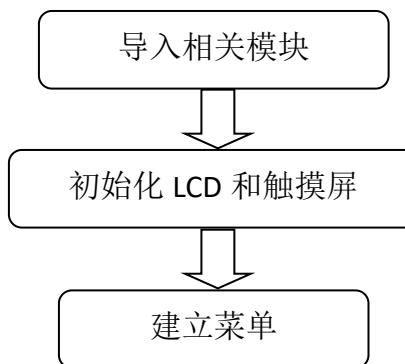


图 8-49 编程流程图

参考代码如下:

```

...
实验名称: List(菜单)
版本: v1.0
日期: 2020.7
作者: 01Studio 【www.01Studio.org】

```

```
'''

省略 LCD 初始化和触摸屏校准代码，详情请参考例程代码文件

List #####

def event_handler(obj, event):

 if event == lv.EVENT.CLICKED:

 print("Clicked: %s" % lv.list.get_btn_text(obj))

if TOUCH_READY:

 # Create a list

 list1 = lv.list(lv.scr_act())

 list1.set_size(160, 200)

 list1.align(None, lv.ALIGN.CENTER, 0, 0)

 # Add buttons to the list

 list_btn = list1.add_btn(lv.SYMBOL.FILE, "New")

 list_btn.set_event_cb(event_handler)

 list_btn = list1.add_btn(lv.SYMBOL.DIRECTORY, "Open")

 list_btn.set_event_cb(event_handler)

 list_btn = list1.add_btn(lv.SYMBOL.CLOSE, "Delete")

 list_btn.set_event_cb(event_handler)

 list_btn = list1.add_btn(lv.SYMBOL.EDIT, "Edit")

 list_btn.set_event_cb(event_handler)

 list_btn = list1.add_btn(lv.SYMBOL.SAVE, "Save")

list_btn.set_event_cb(event_handler)
```

- **实验结果：**

运行代码，可以看到 LCD 出现菜单。

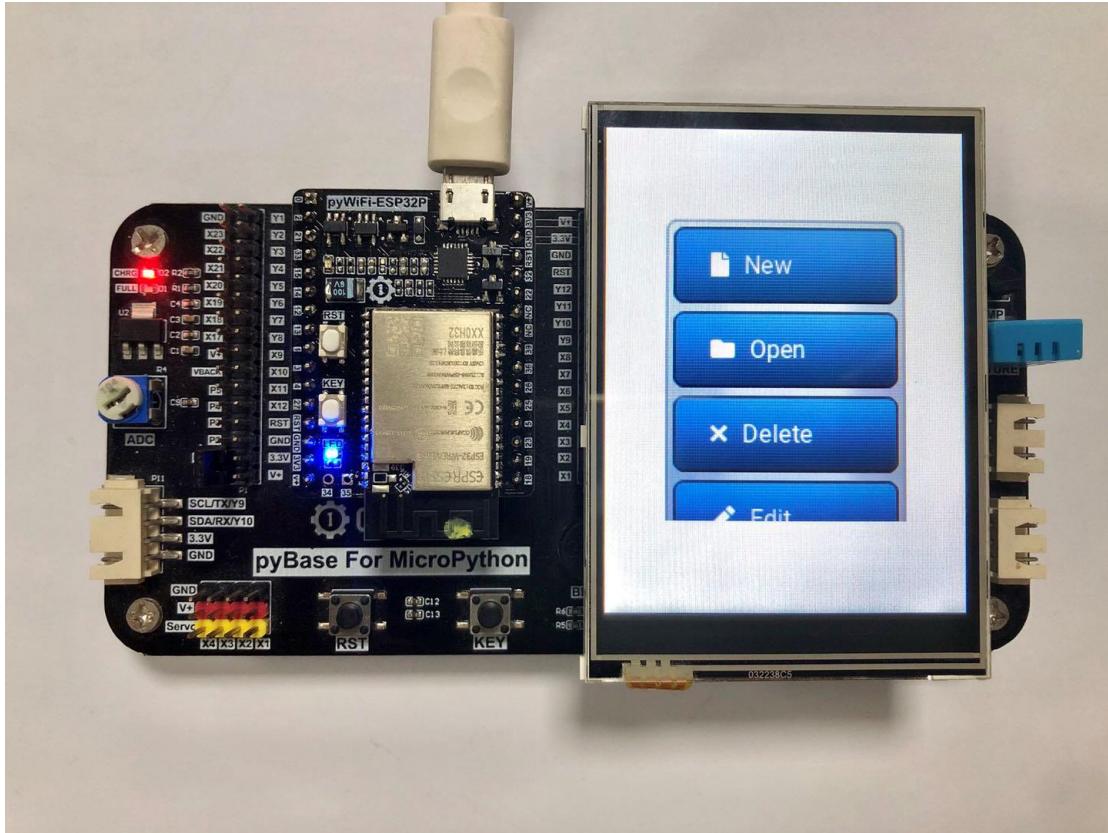


图 8-50 实验结果

- **总结**

本节主要讲解了 LVGL 中菜单的编程和使用方法。

### 8.3.15 Message box (消息框)

- 前言:

消息框可以用于弹出信息或者选择框。

- 实验平台:

pyWiFi-ESP32P + 3.2 寸 LCD 开发套件。

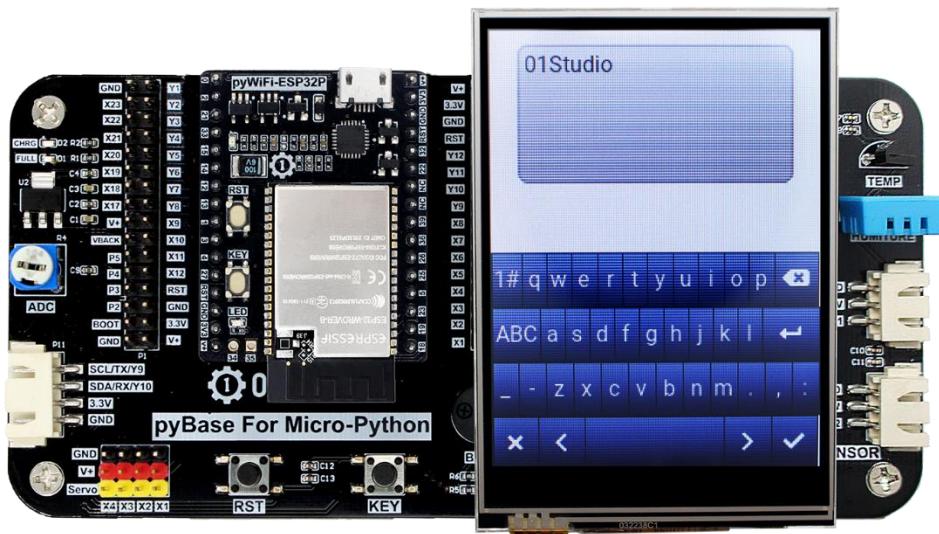


图 8-51 pyWiFi-ESP32P + 3.2 寸 LCD 开发套件

- 实验目的:

创建消息框。

- 实验讲解:

我们来看看消息框对象。

#### 构造函数

```
mbox = lv.mbox(lv.scr_act())
```

构建菜单对象；

lv.scr\_act() : 表示在当前屏幕创建；

#### 使用方法

```
mbox.set_text(str)
```

|                                                                                                                               |
|-------------------------------------------------------------------------------------------------------------------------------|
| 设置消息框显示文本;                                                                                                                    |
| mbox.add_btns()                                                                                                               |
| 为消息框增加按钮;                                                                                                                     |
| mbox = list.add_btn()                                                                                                         |
| 为菜单增加按钮                                                                                                                       |
| mbox.set_event_cb(callback)                                                                                                   |
| 新增按钮按下的回调函数                                                                                                                   |
| 更多用法请参阅小部件说明文档: (或在 REPL 中构建对象然后通过 tab 补全功能来查看。)                                                                              |
| <a href="https://docs.lvgl.io/latest/en/html/widgets/msgbox.html">https://docs.lvgl.io/latest/en/html/widgets/msgbox.html</a> |

表 8-17 消息框对象

编程思路如下：

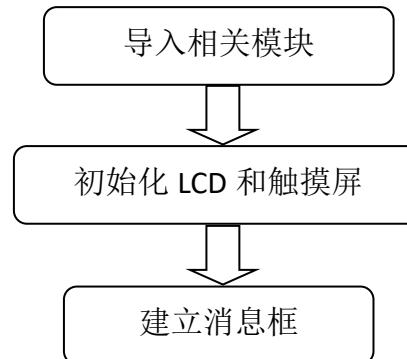


图 8-52 编程流程图

参考代码如下：

```

...
实验名称: Message box(消息框)
版本: v1.0
日期: 2020.7
作者: 01Studio 【www.01Studio.org】
...
省略 LCD 初始化和触摸屏校准代码, 详情请参考例程代码文件

```

```
#####
Message box
#####

def event_handler(obj, event):
 if event == lv.EVENT.VALUE_CHANGED:
 print("Button: %s" % lv.mbox.get_active_btn_text(obj))

if TOUCH_READY:

 btns = ["Apply", "Close", ""]

 mbox1 = lv.mbox(lv.scr_act())
 mbox1.set_text("A message box with two buttons.");
 mbox1.add_btns(btns)
 mbox1.set_width(200)
 mbox1.set_event_cb(event_handler)
 mbox1.align(None, lv.ALIGN.CENTER, 0, 0) # Align to the corner
```

### ● 实验结果：

运行代码，可以看到出现消息框和里面的 2 个按钮。

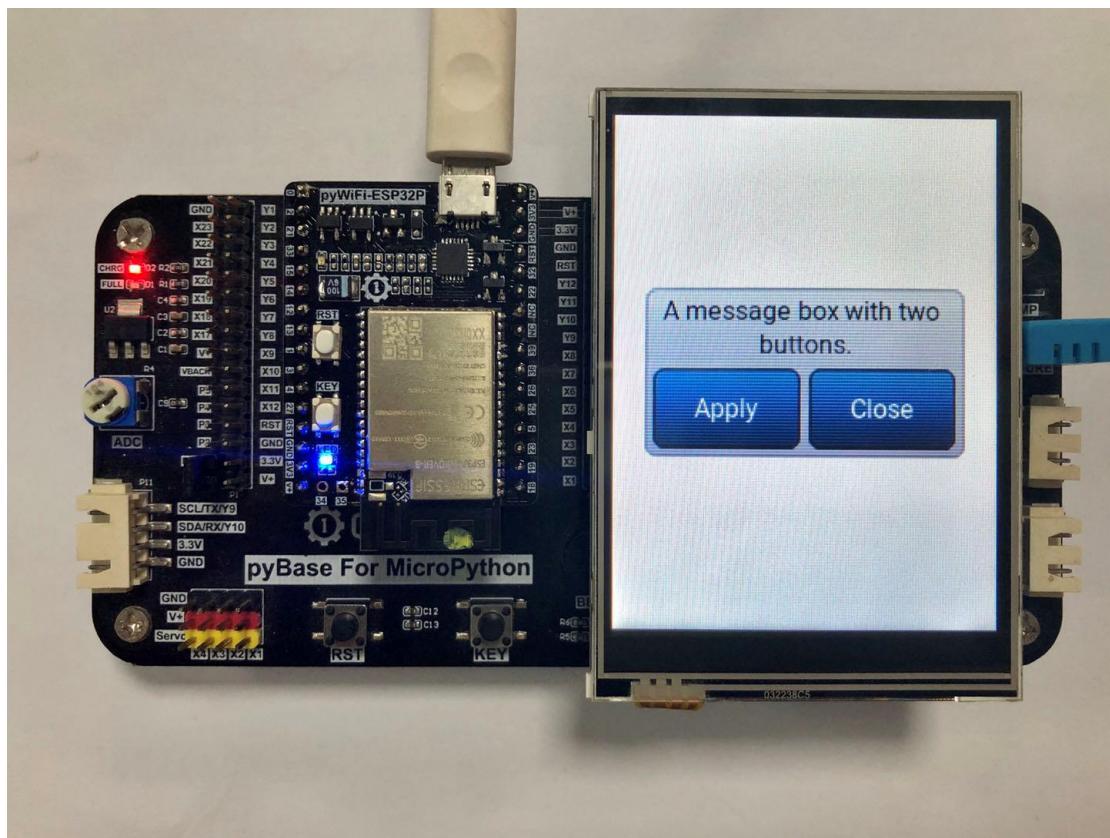


图 8-53 实验结果

### ● 总结

本节主要讲解了 LVGL 中消息框编程和使用方法。

### 8.3.16 Page (页)

- 前言:

页相当于文本阅读器，可以存放大量文字信息(label)。

- 实验平台:

pyWiFi-ESP32P + 3.2 寸 LCD 开发套件。



图 8-54 pyWiFi-ESP32P + 3.2寸LCD 开发套件

- 实验目的:

创建页。

- 实验讲解:

我们来看看页对象。

#### 构造函数

```
page = lv.page(lv.scr_act())
```

构建菜单对象；

lv.scr\_act() : 表示在当前屏幕创建；

#### 使用方法

```
page.set_size(width, height)
```

设置页的尺寸；

【width】宽度；

【height】高度。

page.align()

对齐方式：

label = lv.label(page)

在页中加入 label

更多用法请参阅小部件说明文档：（或在 REPL 中构建对象然后通过 tab 补全功能来查看。）

<https://docs.lvgl.io/latest/en/html/widgets/page.html>

表 8-18 页对象

编程思路如下：

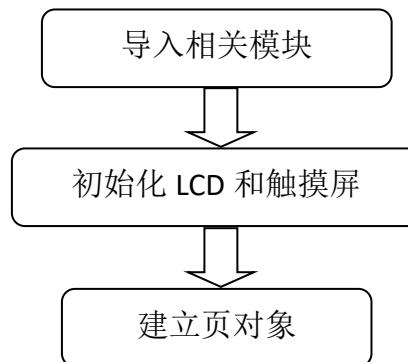


图 8-55 编程流程图

参考代码如下：

```
...
实验名称: Page(页)
版本: v1.0
日期: 2020.7
作者: 01Studio 【www.01Studio.org】
...
省略 LCD 初始化和触摸屏校准代码, 详情请参考例程代码文件
```

```

#####
Page
#####

if TOUCH_READY:

 #Create a scroll bar style

 style_sb = lv.style_t()

 lv.style_copy(style_sb, lv.style_plain)

 style_sb.body.main_color = lv.color_make(0,0,0)
 style_sb.body.grad_color = lv.color_make(0,0,0)
 style_sb.body.border.color = lv.color_make(0xff,0xff,0xff)
 style_sb.body.border.width = 1
 style_sb.body.border.opa = lv.OPA._70
 style_sb.body.radius = 800 # large enough to make a circle
 style_sb.body.opa = lv.OPA._60
 style_sb.body.padding.right = 3
 style_sb.body.padding.bottom = 3
 style_sb.body.padding.inner = 8 # Scrollbar width

 # Create a page

 page = lv.page(lv.scr_act())

 page.set_size(240, 300)
 page.align(None, lv.ALIGN.CENTER, 0, 0)
 page.set_style(lv.page.STYLE.SB, style_sb) #Set the scrollbar style

 # Create a label on the page

 label = lv.label(page)

 #Automatically break long lines

 label.set_long_mode(lv.label.LONG.BREAK)

 # Set the label width to max value to not show hor. scroll bars

 label.set_width(page.get_fit_width())

```

```
label.set_text("""Lorem ipsum dolor sit amet, consectetur adipiscing
elit,
sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.
Ut enim ad minim veniam, quis nostrud exercitation ullamco
laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure
dolor in reprehenderit in voluptate velit esse cillum dolore
eu fugiat nulla pariatur.
Excepteur sint occaecat cupidatat non proident, sunt in culpa
qui officia deserunt mollit anim id est laborum.""")
```

### ● 实验结果：

运行代码，可以看到出现页面以及文字。

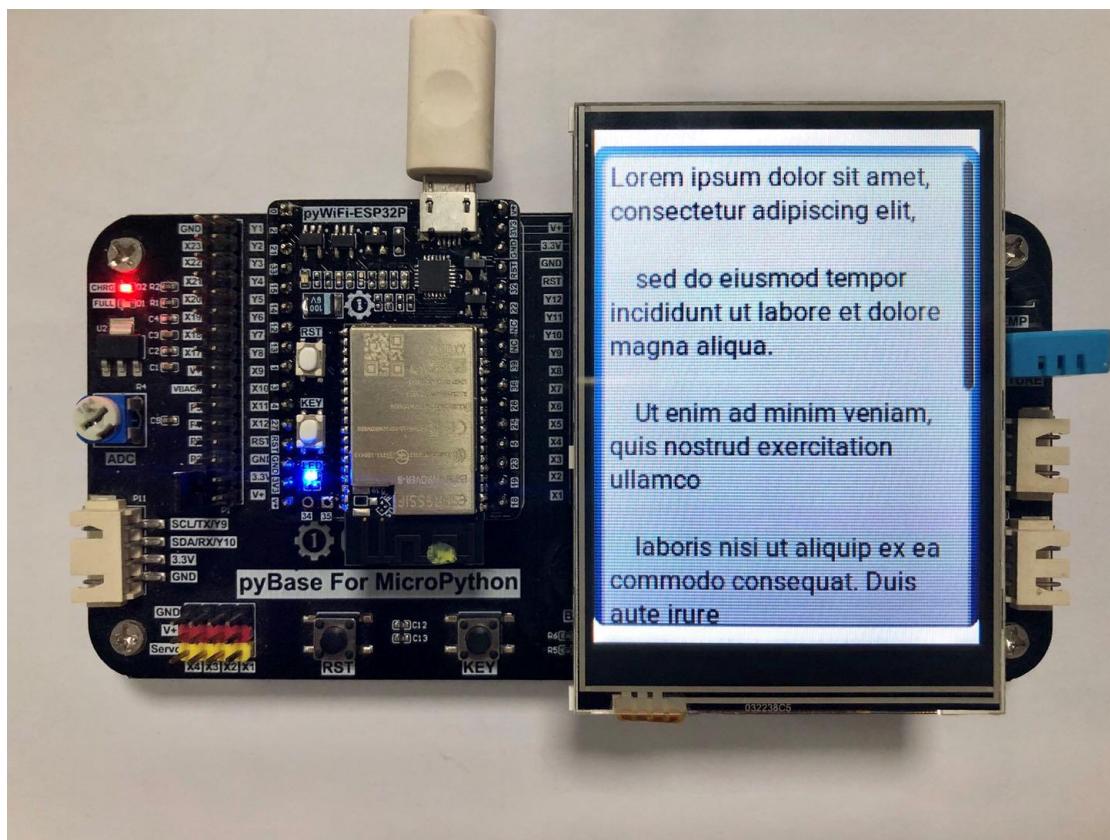


图 8-56 实验结果

### ● 总结

本节主要讲解了 LVGL 中页面的使用方法。

### 8.3.17 Roller (滚筒)

- 前言:

滚筒可以用于多个选项的快速选择，例如时间和日期。

- 实验平台:

pyWiFi-ESP32P + 3.2 寸 LCD 开发套件。



图 8-57 pyWiFi-ESP32P + 3.2 寸 LCD 开发套件

- 实验目的:

创建滚筒。

- 实验讲解:

我们来看看滚筒对象。

#### 构造函数

```
roller = lv.roller(lv.scr_act())
```

构建滚筒对象；

lv.scr\_act() : 表示在当前屏幕创建；

#### 使用方法

```
roller.set_options()
```

|                                                                                                                                                                                  |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 设置选项；设置方法参考例程代码。                                                                                                                                                                 |
| <code>roller.set_visible_row_count(num)</code>                                                                                                                                   |
| 滚筒窗口可视数量个数；<br>【num】可视选项数量。                                                                                                                                                      |
| <code>roller.align()</code>                                                                                                                                                      |
| 对齐方式；                                                                                                                                                                            |
| <code>roller.set_event_cb(callback)</code>                                                                                                                                       |
| 设置回调函数；                                                                                                                                                                          |
| 更多用法请参阅小部件说明文档：（或在 REPL 中构建对象然后通过 tab 补全功能来查看。）<br><a href="https://docs.lvgl.io/latest/en/html/widgets/roller.html">https://docs.lvgl.io/latest/en/html/widgets/roller.html</a> |

表 8-19 滚筒对象

编程思路如下：

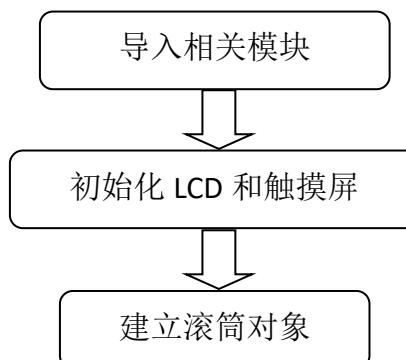


图 8-58 编程流程图

参考代码如下：

```

...
实验名称: Roller(滚筒)
版本: v1.0
日期: 2020.7
作者: 01Studio 【www.01Studio.org】
...

```

```
省略 LCD 初始化和触摸屏校准代码，详情请参考例程代码文件

#####
Roller
#####

def event_handler(obj, event):

 if event == lv.EVENT.VALUE_CHANGED:

 option = " "*10

 obj.get_selected_str(option, len(option))

 print("Selected month: %s" % option.strip())

if TOUCH_READY:

 roller1 = lv.roller(lv.scr_act())

 roller1.set_options("\n".join([
 "January",
 "February",
 "March",
 "April",
 "May",
 "June",
 "July",
 "August",
 "September",
 "October",
 "November",
 "December"]), lv.ROLLER.MODE.INFINITE)

 roller1.set_visible_row_count(4)

 roller1.align(None, lv.ALIGN.CENTER, 0, 0)
```

```
roller1.set_event_cb(event_handler)
```

- 实验结果：

运行代码，可以看到出现滚筒和对应的选项。

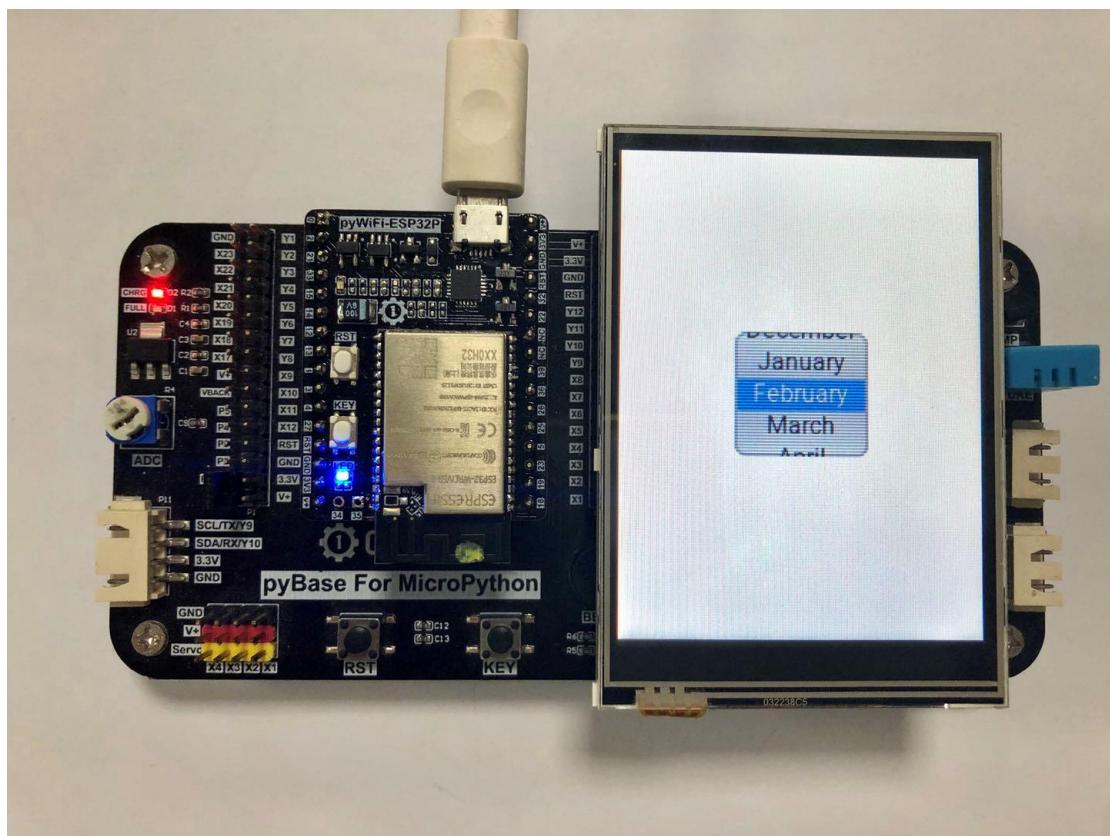


图 8-59 实验结果

- 总结

本节主要讲解了 LVGL 中滚筒的使用方法。

### 8.3.18 Slider (滑动条)

- 前言:

滑动条可以用于进度的选择。

- 实验平台:

pyWiFi-ESP32P + 3.2 寸 LCD 开发套件。



图 8-60 pyWiFi-ESP32P + 3.2 寸 LCD 开发套件

- 实验目的:

创建滑动条。

- 实验讲解:

我们来看看滑动条对象。

#### 构造函数

```
slider = lv.slider(lv.scr_act())
```

构建滑动条对象；

lv.scr\_act() : 表示在当前屏幕创建；

#### 使用方法

```
slider.set_width(200)
```

|                                                                                                                               |
|-------------------------------------------------------------------------------------------------------------------------------|
| 设置宽度。                                                                                                                         |
| <code>slider.set_range(0,100)</code>                                                                                          |
| 设置滑动条最大值和最小值; (0,100)表示从 0 到 100。                                                                                             |
| <code>slider.align()</code>                                                                                                   |
| 对齐方式;                                                                                                                         |
| <code>slider.set_event_cb(callback)</code>                                                                                    |
| 设置回调函数;                                                                                                                       |
| 更多用法请参阅小部件说明文档: (或在 REPL 中构建对象然后通过 tab 补全功能来查看。)                                                                              |
| <a href="https://docs.lvgl.io/latest/en/html/widgets/slider.html">https://docs.lvgl.io/latest/en/html/widgets/slider.html</a> |

表 8-20 滑动条对象

编程思路如下：

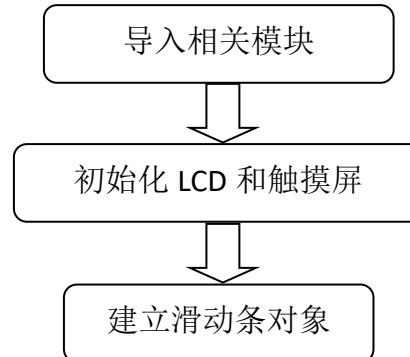


图 8-61 编程流程图

参考代码如下：

```

...
实验名称: Slider(滑动条)
版本: v1.0
日期: 2020.7
作者: 01Studio 【www.01Studio.org】
...
省略 LCD 初始化和触摸屏校准代码, 详情请参考例程代码文件

```

```
#####
Slider
#####

def slider_event_cb(slider, event):
 if event == lv.EVENT.VALUE_CHANGED:
 slider_label.set_text("%u" % slider.get_value())

if TOUCH_READY:

 # Create a slider in the center of the display
 slider = lv.slider(lv.scr_act())
 slider.set_width(200)
 slider.align(None, lv.ALIGN.CENTER, 0, 0)
 slider.set_event_cb(slider_event_cb)
 slider.set_range(0, 100)

 # Create a label below the slider
 slider_label = lv.label(lv.scr_act())
 slider_label.set_text("0")
 slider_label.set_auto_realign(True)
 slider_label.align(slider, lv.ALIGN.OUT_BOTTOM_MID, 0, 10)
```

### ● 实验结果：

运行代码，可以看 LCD 出现滑动条，拖动滑动条，数字发生变化。

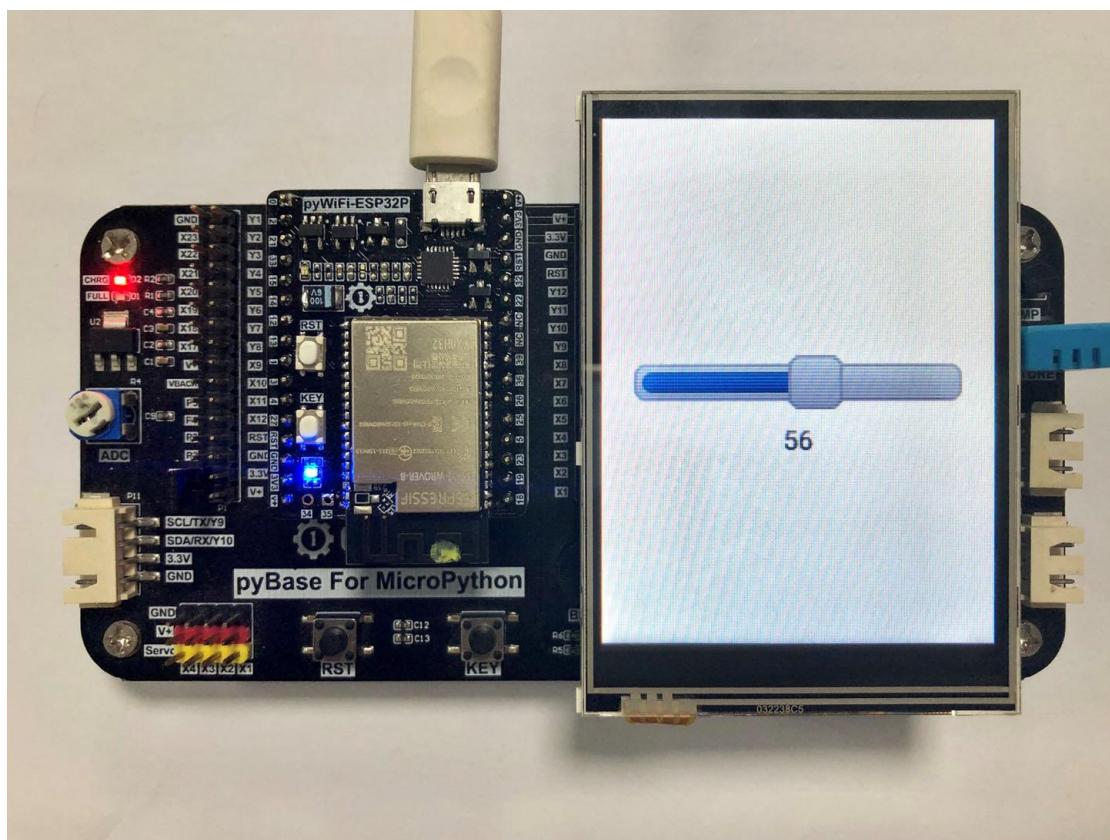


图 8-62 实验结果

### ● 总结

本节主要讲解了 LVGL 中滚动条的使用方法。

### 8.3.19 Spinner (旋转加载条)

- 前言:

旋转加载条用于等待加载指示。

- 实验平台:

pyWiFi-ESP32P + 3.2 寸 LCD 开发套件。

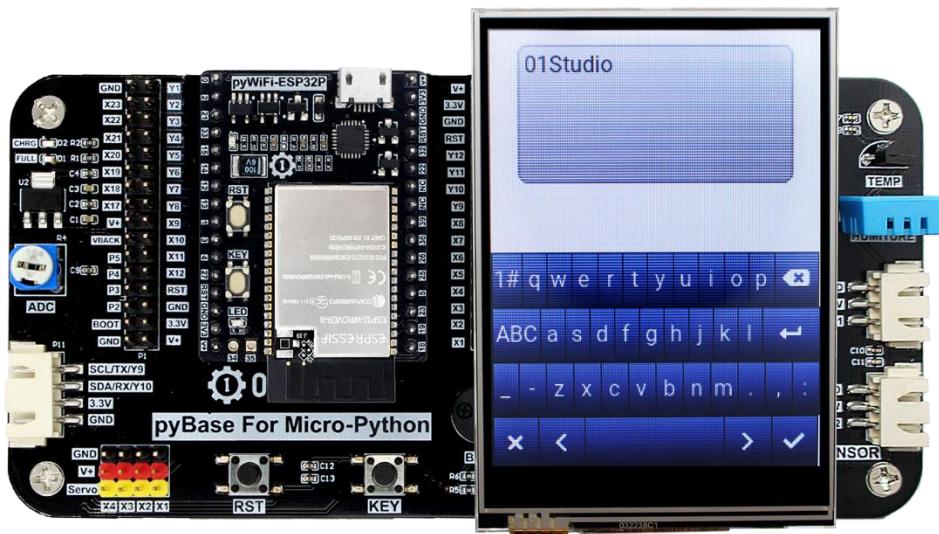


图 8-63 pyWiFi-ESP32P + 3.2 寸 LCD 开发套件

- 实验目的:

创建旋转加载条。

- 实验讲解:

我们来看看旋转加载条对象。

#### 构造函数

```
preload = lv.preload(lv.scr_act())
```

构建旋转加载条对象；

lv.scr\_act() : 表示在当前屏幕创建；

#### 使用方法

```
preload.set_size(width,height)
```

设置尺寸。

【width】宽度；

【height】高度。

preload.align()

对齐方式：

preload.set\_style()

设置风格；

更多用法请参阅小部件说明文档：（或在 REPL 中构建对象然后通过 tab 补全功能来查看。）

<https://docs.lvgl.io/latest/en/html/widgets/spinner.html>

表 8-21 旋转加载条对象

编程思路如下：

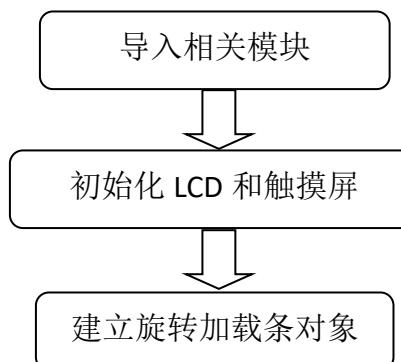


图 8-64 编程流程图

参考代码如下：

```
...
实验名称: Spinner(旋转加载条)
版本: v1.0
日期: 2020.7
作者: 01Studio 【www.01Studio.org】
...
省略 LCD 初始化和触摸屏校准代码, 详情请参考例程代码文件
```

```
#####
Spinner
#####

if TOUCH_READY:

 # Create a style for the Preloader
 style = lv.style_t()
 lv.style_copy(style, lv.style_plain)
 style.line.width = 10 # 10 px thick arc
 style.line.color = lv.color_hex3(0x258) # Blueish arc color

 # Gray background color
 style.body.border.color = lv.color_hex3(0xBBB)
 style.body.border.width = 10
 style.body.padding.left = 0

 # Create a Preloader object
 preload = lv.preload(lv.scr_act())
 preload.set_size(100, 100)
 preload.align(None, lv.ALIGN.CENTER, 0, 0)
 preload.set_style(lv.preload.STYLE.MAIN, style)
```

### ● 实验结果：

运行代码，可以看 LCD 出现旋转加载条。

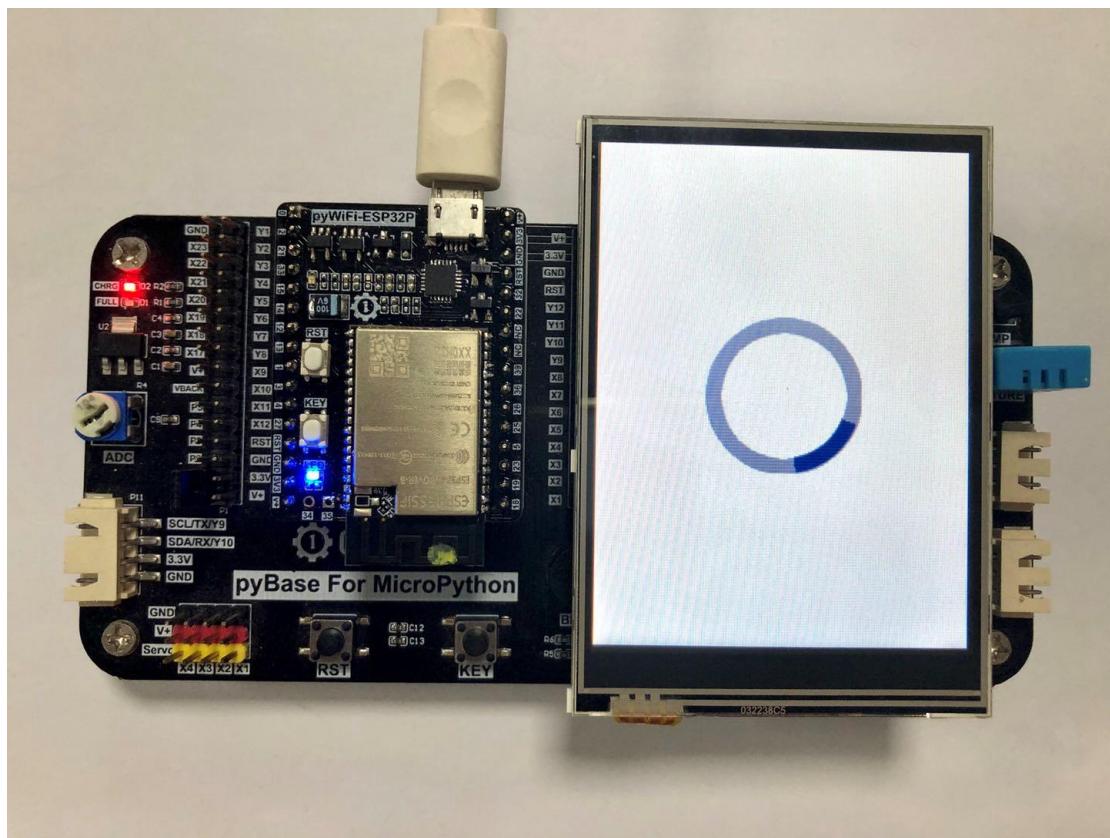


图 8-65 实验结果

### ● 总结

本节主要讲解了 LVGL 中旋转加载条的使用方法。

### 8.3.20 Swtich (开关)

- 前言:

开关是非常常见和常用的小部件。

- 实验平台:

pyWiFi-ESP32P + 3.2 寸 LCD 开发套件。

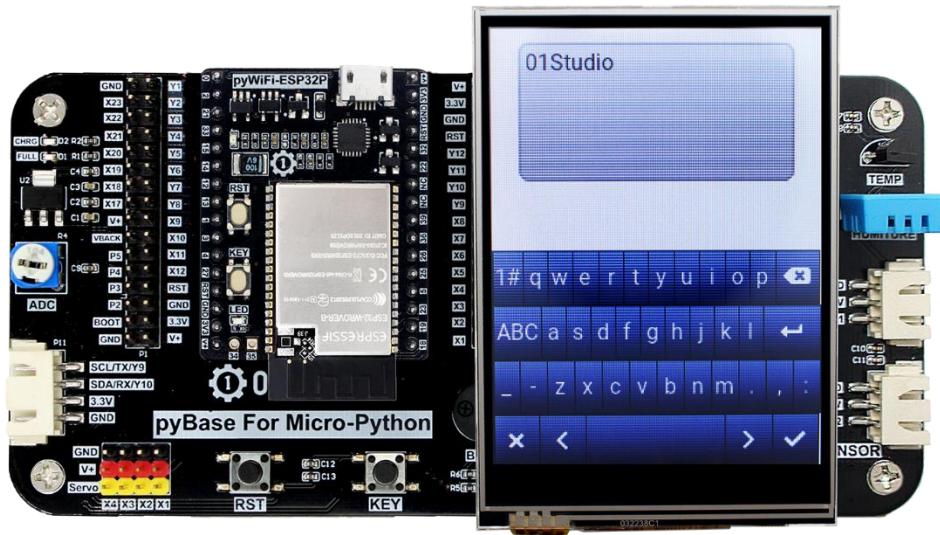


图 8-66 pyWiFi-ESP32P + 3.2 寸 LCD 开发套件

- 实验目的:

创建开关。

- 实验讲解:

我们来看看开关对象。

| 构造函数                                 |
|--------------------------------------|
| <pre>sw = lv.sw(lv.scr_act())</pre>  |
| 构建开关对象;<br>lv.scr_act() : 表示在当前屏幕创建; |
| 使用方法                                 |
| <pre>sw.set_style()</pre>            |

|                                                                                                                               |
|-------------------------------------------------------------------------------------------------------------------------------|
| 设置风格。                                                                                                                         |
| <code>sw.align()</code>                                                                                                       |
| 对齐方式;                                                                                                                         |
| <code>sw.set_event_cb(callback)</code>                                                                                        |
| 设置开关回调函数;                                                                                                                     |
| 更多用法请参阅小部件说明文档: (或在 REPL 中构建对象然后通过 tab 补全功能来查看。)                                                                              |
| <a href="https://docs.lvgl.io/latest/en/html/widgets/switch.html">https://docs.lvgl.io/latest/en/html/widgets/switch.html</a> |

表 8-22 开关对象

编程思路如下:

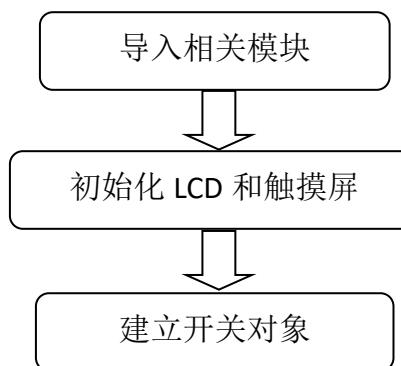


图 8-67 编程流程图

参考代码如下:

```

...
实验名称: Switch(开关)
版本: v1.0
日期: 2020.7
作者: 01Studio 【www.01Studio.org】
...
省略 LCD 初始化和触摸屏校准代码, 详情请参考例程代码文件
#####
Switch
#####

```

```

def event_handler(obj, event):
 if event == lv.EVENT.VALUE_CHANGED:
 print("State: %s" % ("On" if obj.get_state() else "Off"))

if TOUCH_READY:

 # Create styles for the switch

 bg_style = lv.style_t()
 indic_style = lv.style_t()
 knob_on_style = lv.style_t()
 knob_off_style = lv.style_t()

 lv.style_copy(bg_style, lv.style_pretty)
 bg_style.body.radius = 800
 bg_style.body.padding.top = 6
 bg_style.body.padding.bottom = 6

 lv.style_copy(indic_style, lv.style_pretty_color)
 indic_style.body.radius = 800
 indic_style.body.main_color = lv.color_hex(0xfc8ef)
 indic_style.body.grad_color = lv.color_hex(0x9fc8ef)
 indic_style.body.padding.left = 0
 indic_style.body.padding.right = 0
 indic_style.body.padding.top = 0
 indic_style.body.padding.bottom = 0

 lv.style_copy(knob_off_style, lv.style_pretty)
 knob_off_style.body.radius = 800
 knob_off_style.body.shadow.width = 4
 knob_off_style.body.shadow.type = lv.SHADOW.BOTTOM

```

```
lv.style_copy(knob_on_style, lv.style_pretty_color)

knob_on_style.body.radius = 800
knob_on_style.body.shadow.width = 4
knob_on_style.body.shadow.type = lv.SHADOW.BOTTOM

Create a switch and apply the styles
sw1 = lv.sw(lv.scr_act())
sw1.set_style(lv.sw.STYLE.BG, bg_style)
sw1.set_style(lv.sw.STYLE.INDIC, indic_style)
sw1.set_style(lv.sw.STYLE.KNOB_ON, knob_on_style)
sw1.set_style(lv.sw.STYLE.KNOB_OFF, knob_off_style)
sw1.align(None, lv.ALIGN.CENTER, 0, -50)
sw1.set_event_cb(event_handler)

Copy the first switch and turn it ON
sw2 = lv.sw(lv.scr_act(), sw1)
sw2.on(lv.ANIM.ON)
sw2.align(None, lv.ALIGN.CENTER, 0, 50)
sw2.set_event_cb(lambda o,e: None)
```

### ● 实验结果：

运行代码，可以看出现 2 个开关，打开和关闭上面的开关，可以在 REPL 看到状态变化提示信息。

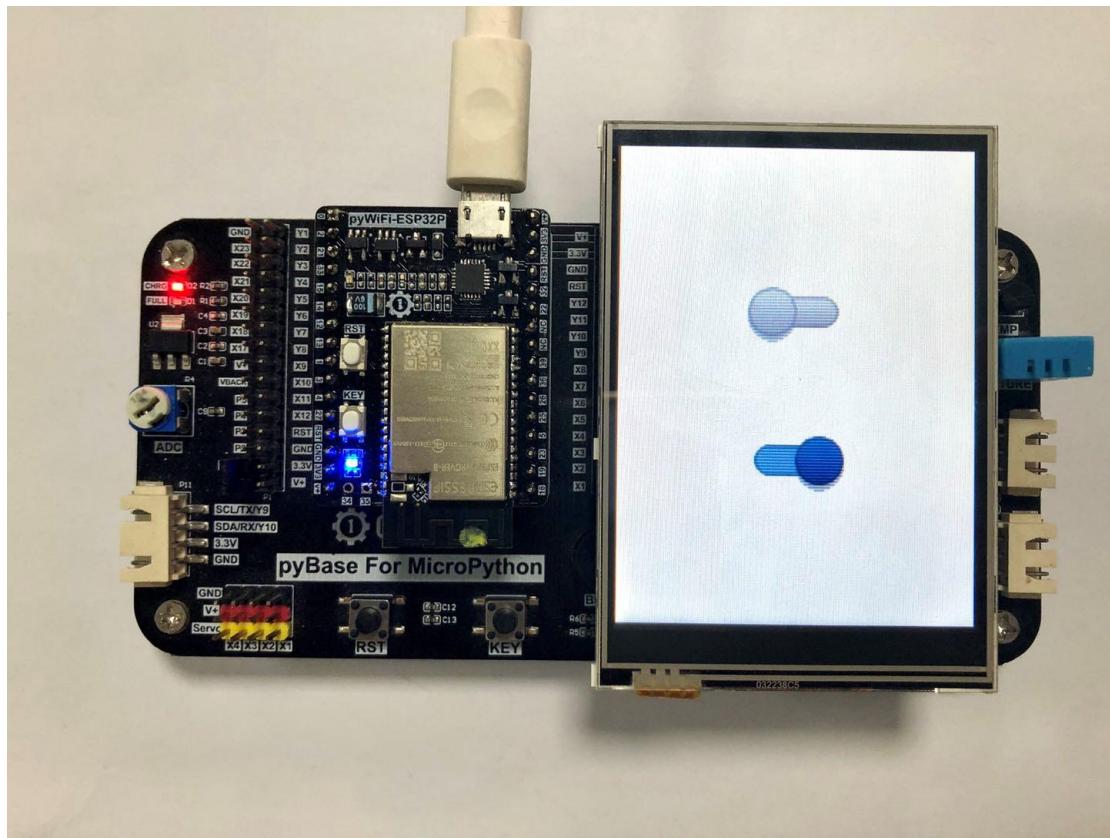


图 8-68 实验结果

```
ESP MicroPython REPL
Enable backlight
Double buffer
MicroPython v1.9.4-2072-gf89c8c48c-dirty on 2020-01-17; ESP32 module (spiram) with ESP32
Type "help()" for more information.
>>> State: On
State: Off
```

图 8-69 REPL 提示

## ● 总结

本节主要讲解了 LVGL 中开关的使用方法。

### 8.3.21 Table (表格)

- 前言:

表格是非常常用的小部件。

- 实验平台:

pyWiFi-ESP32P + 3.2 寸 LCD 开发套件。

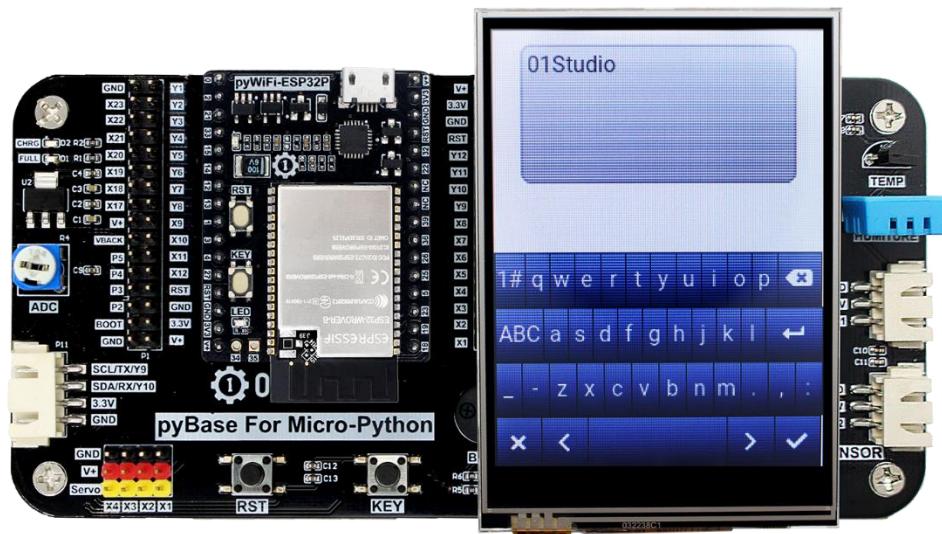


图 8-70 pyWiFi-ESP32P + 3.2 寸 LCD 开发套件

- 实验目的:

创建表格。

- 实验讲解:

我们来看看表格对象。

#### 构造函数

```
table = lv.table(lv.scr_act())
```

构建表格对象；

lv.scr\_act() : 表示在当前屏幕创建；

#### 使用方法

```
table.set_style()
```

|                                                                                                                                                                                |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 设置风格。                                                                                                                                                                          |
| <code>table.set_col_cnt(num)</code>                                                                                                                                            |
| 设置列数；                                                                                                                                                                          |
| <code>table.set_row_cnt(num)</code>                                                                                                                                            |
| 设置行数；                                                                                                                                                                          |
| <code>table.align()</code>                                                                                                                                                     |
| 表格对齐方式；                                                                                                                                                                        |
| <code>table.set_cell_align(row, col, lv.label.ALIGN.CENTER)</code>                                                                                                             |
| 设置某各单元格对齐方式；<br>【row】单元格行；<br>【col】单元格列；<br>【lv.label.ALIGN.CENTER】居中对齐。                                                                                                       |
| <code>table.set_cell_value(row, col, str)</code>                                                                                                                               |
| 设置某个单元格值；<br>【row】单元格行；<br>【col】单元格列；<br>【str】单元格值；                                                                                                                            |
| 更多用法请参阅小部件说明文档：（或在 REPL 中构建对象然后通过 tab 补全功能来查看。）<br><a href="https://docs.lvgl.io/latest/en/html/widgets/table.html">https://docs.lvgl.io/latest/en/html/widgets/table.html</a> |

表 8-23 表格对象

编程思路如下：

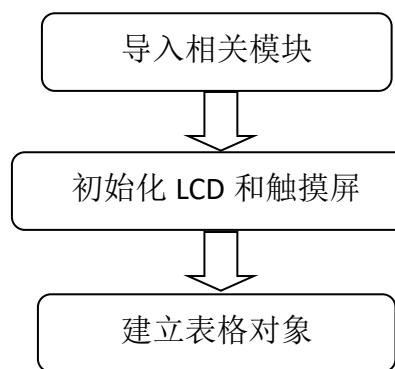


图 8-71 编程流程图

参考代码如下：

```
...
实验名称: Table(表格)
版本: v1.0
日期: 2020.7
作者: 01Studio 【www.01Studio.org】
...

省略 LCD 初始化和触摸屏校准代码, 详情请参考例程代码文件
#####
Table
#####

if TOUCH_READY:

 # Create a normal cell style
 style_cell1 = lv.style_t()
 lv.style_copy(style_cell1, lv.style_plain)
 style_cell1.body.border.width = 1
 style_cell1.body.border.color = lv.color_make(0,0,0)

 # Create a header cell style
 style_cell2 = lv.style_t()
 lv.style_copy(style_cell2, lv.style_plain)
 style_cell2.body.border.width = 1
 style_cell2.body.border.color = lv.color_make(0,0,0)
 style_cell2.body.main_color = lv.color_make(0xC0, 0xC0, 0xC0)
 style_cell2.body.grad_color = lv.color_make(0xC0, 0xC0, 0xC0)

 table = lv.table(lv.scr_act())
 table.set_style(lv.table.STYLE.CELL1, style_cell1)
 table.set_style(lv.table.STYLE.CELL2, style_cell2)
```

```
table.set_style(lv.table.STYLE.BG, lv.style_transp_tight)

table.set_col_cnt(2)

table.set_row_cnt(4)

table.align(None, lv.ALIGN.CENTER, 0, 0)

Make the cells of the first row center aligned

table.set_cell_align(0, 0, lv.label.ALIGN.CENTER)

table.set_cell_align(0, 1, lv.label.ALIGN.CENTER)

Make the cells of the first row TYPE = 2 (use `style_cell2`)

table.set_cell_type(0, 0, 2)

table.set_cell_type(0, 1, 2)

Fill the first column

table.set_cell_value(0, 0, "Name")

table.set_cell_value(1, 0, "Apple")

table.set_cell_value(2, 0, "Banana")

table.set_cell_value(3, 0, "Citron")

Fill the second column

table.set_cell_value(0, 1, "Price")

table.set_cell_value(1, 1, "$7")

table.set_cell_value(2, 1, "$4")

table.set_cell_value(3, 1, "$6")
```

## ● 实验结果：

运行代码，可以看见编程实现的表格内容。

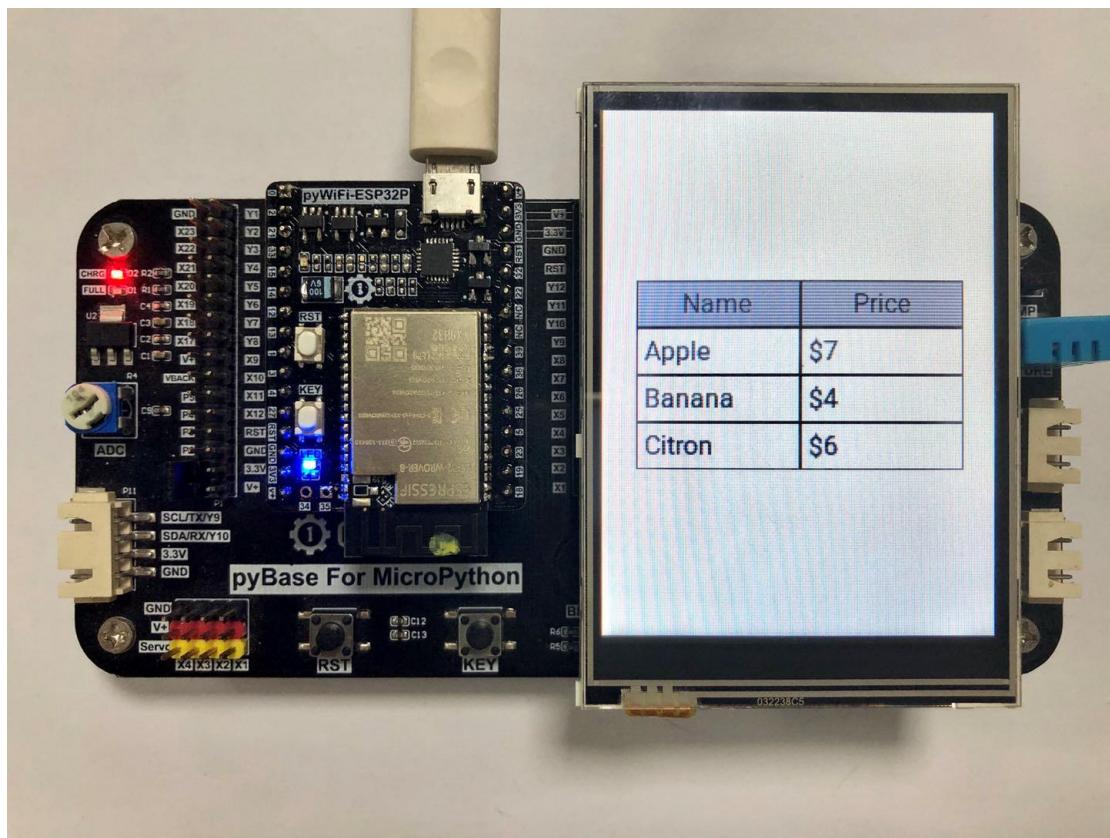


图 8-72 实验结果

### ● 总结

本节主要讲解了 LVGL 中表格的使用方法。

### 8.3.22 Tabview（选项卡视图）

- 前言：

选项卡视图可以用于切换不同的界面。

- 实验平台：

pyWiFi-ESP32P + 3.2 寸 LCD 开发套件。



图 8-73 pyWiFi-ESP32P + 3.2 寸 LCD 开发套件

- 实验目的：

创建选项卡视图。

- 实验讲解：

我们来看看选项卡视图对象。

#### 构造函数

```
tabview = lv.tabview(lv.scr_act())
```

构建选项卡视图对象；

lv.scr\_act() : 表示在当前屏幕创建；

#### 使用方法

```
tab1 = tabview.add_tab("Tab 1")
```

增加一个选项卡

```
label = lv.label(tab1)
```

在选项卡加入标签（label）

更多用法请参阅小部件说明文档：（或在 REPL 中构建对象然后通过 tab 补全功能来查看。）

<https://docs.lvgl.io/latest/en/html/widgets/tabview.html>

表 8-24 选项卡视图对象

编程思路如下：

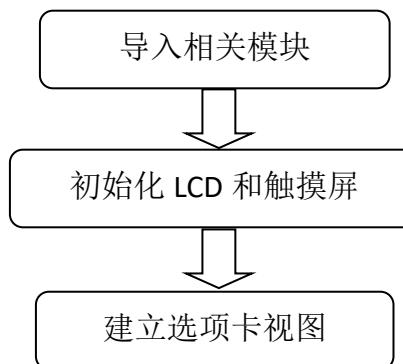


图 8-74 编程流程图

参考代码如下：

```
...
实验名称: Tabview(选项卡视图)
版本: v1.0
日期: 2020.7
作者: 01Studio 【www.01Studio.org】
...

省略 LCD 初始化和触摸屏校准代码, 详情请参考例程代码文件
#####
Tabview
#####

if TOUCH_READY:

 # Create a Tab view object
```

```
tabview = lv.tabview(lv.scr_act())

Add 3 tabs (the tabs are page (lv_page) and can be scrolled
tab1 = tabview.add_tab("Tab 1")
tab2 = tabview.add_tab("Tab 2")
tab3 = tabview.add_tab("Tab 3")

Add content to the tabs
label = lv.label(tab1)
label.set_text(""" This the first tab
If the content
of a tab
become too long
the it
automatically
become
scrollable.""")

label = lv.label(tab2)
label.set_text("Second tab")

label = lv.label(tab3)
label.set_text("Third tab")
```

### ● 实验结果：

运行代码，可以看见编程实现的 3 个选项卡视图内容。

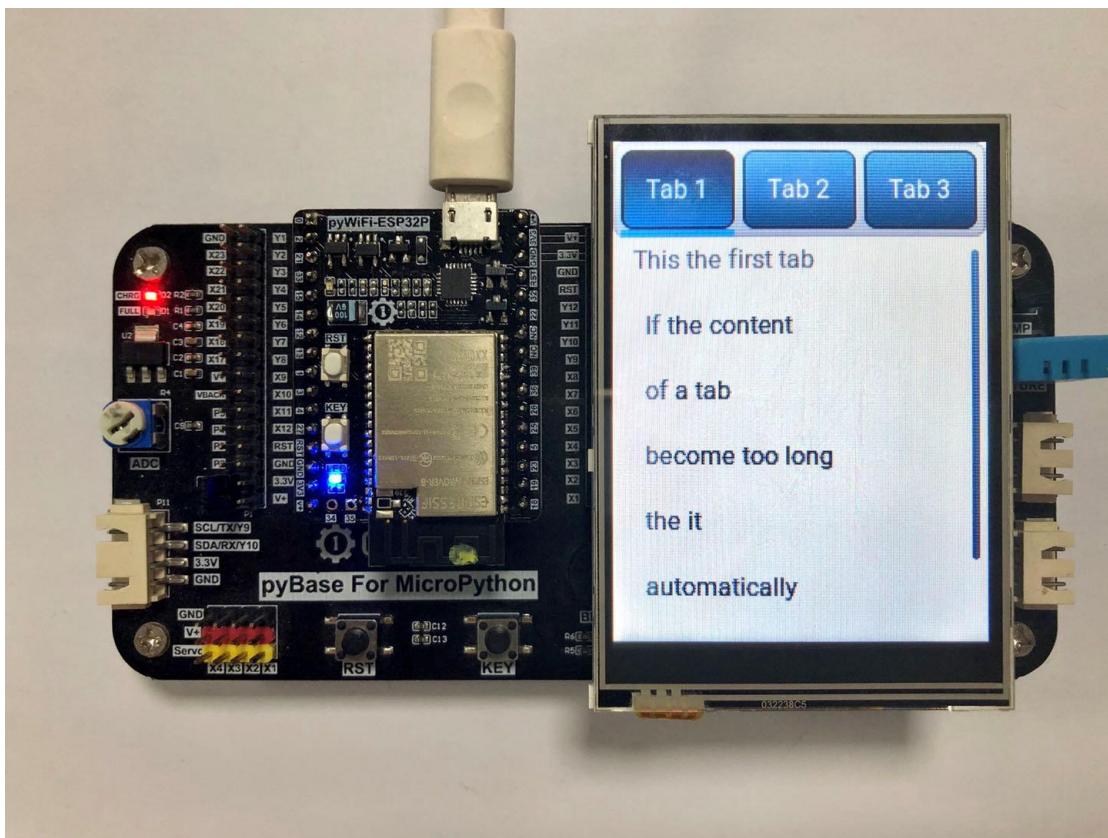


图 8-75 实验结果

## ● 总结

本节主要讲解了 LVGL 中选项卡视图的使用方法。

### 8.3.23 Textarea (文本区域)

- 前言:

文本区域用于展示文本。

- 实验平台:

pyWiFi-ESP32P + 3.2 寸 LCD 开发套件。



图 8-76 pyWiFi-ESP32P + 3.2 寸 LCD 开发套件

- 实验目的:

创建文本区域。

- 实验讲解:

我们来看看文本区域对象。

| 构造函数                                                |
|-----------------------------------------------------|
| <pre>ta = lv.ta(lv.scr_act())</pre>                 |
| 构建文本区域对象;<br><code>lv.scr_act()</code> : 表示在当前屏幕创建; |
| 使用方法                                                |
| <pre>ta.set_size(width, height)</pre>               |

设置尺寸；

【width】宽度；

【height】高度。

ta.align()

对齐方式；

ta.set\_text(str)

为文本区域添加文本。

更多用法请参阅小部件说明文档：（或在 REPL 中构建对象然后通过 tab 补全功能来查看。）

<https://docs.lvgl.io/latest/en/html/widgets/textarea.html>

表 8-25 文本区域对象

编程思路如下：

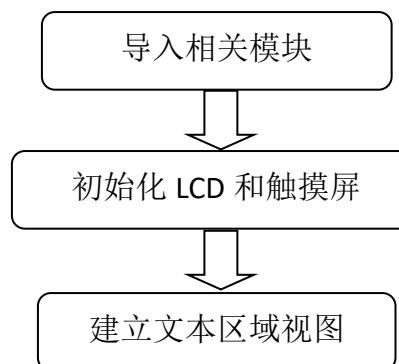


图 8-77 编程流程图

参考代码如下：

```
...
实验名称: Textarea(文本区域)
版本: v1.0
日期: 2020.7
作者: 01Studio 【www.01Studio.org】
...
省略 LCD 初始化和触摸屏校准代码, 详情请参考例程代码文件
#####
Textarea
#####
```

```
#####
if TOUCH_READY:

 ta1 = lv.ta(lv.scr_act())

 ta1.set_size(200, 100)

 ta1.align(None, lv.ALIGN.CENTER, 0, 0)

 ta1.set_cursor_type(lv.CURSOR.BLOCK)

 ta1.set_text("A text in a Text Area") # Set an initial text
```

### ● 实验结果：

运行代码，可以看见编程实现的文本区域小部件。



图 8-78 实验结果

### ● 总结

本节主要讲解了 LVGL 中文本区域的使用方法。

### 8.3.24 Window (窗口)

- 前言:

窗口除了展示内容还可以添加关闭、设置等等按钮。

- 实验平台:

pyWiFi-ESP32P + 3.2 寸 LCD 开发套件。



图 8-79 pyWiFi-ESP32P + 3.2 寸 LCD 开发套件

- 实验目的:

创建窗口。

- 实验讲解:

我们来看看窗口对象。

#### 构造函数

```
win = lv.win(lv.scr_act())
```

构建文本区域对象；

lv.scr\_act() : 表示在当前屏幕创建；

#### 使用方法

```
win.set_title(str)
```

设置窗口标题;

【str】标题内容;

```
close_btn = win.add_btn(lv.SYMBOL CLOSE)
```

为窗口添加关闭按钮

更多用法请参阅小部件说明文档: (或在 REPL 中构建对象然后通过 tab 补全功能来查看。)

<https://docs.lvgl.io/latest/en/html/widgets/win.html>

表 8-26 窗口对象

编程思路如下:

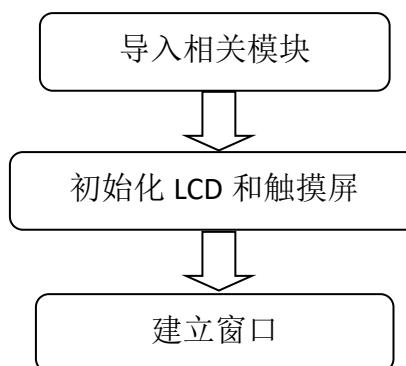


图 8-80 编程流程图

参考代码如下:

```
...
实验名称: Window(窗口)
版本: v1.0
日期: 2020.7
作者: 01Studio 【www.01Studio.org】
...
省略 LCD 初始化和触摸屏校准代码, 详情请参考例程代码文件
#####
Window
#####
if TOUCH_READY:
```

```
Create a window
win = lv.win(lv.scr_act())
win.set_title("Window title") # Set the title

Add control button to the header
close_btn = win.add_btn(lv.SYMBOL.CLOSE) # Add close button and
use built-in close action
close_btn.set_event_cb(lv.win.close_event_cb)
win.add_btn(lv.SYMBOL.SETTINGS) # Add a setup button

Add some dummy content
txt = lv.label(win)
txt.set_text(
"""This is the content of the window
You can add control buttons to
the window header
The content area becomes automatically
scrollable if it's large enough.
You can scroll the content
See the scroll bar on the right!"""
)
```

### ● 实验结果：

运行代码，可以看见编程实现的窗口。

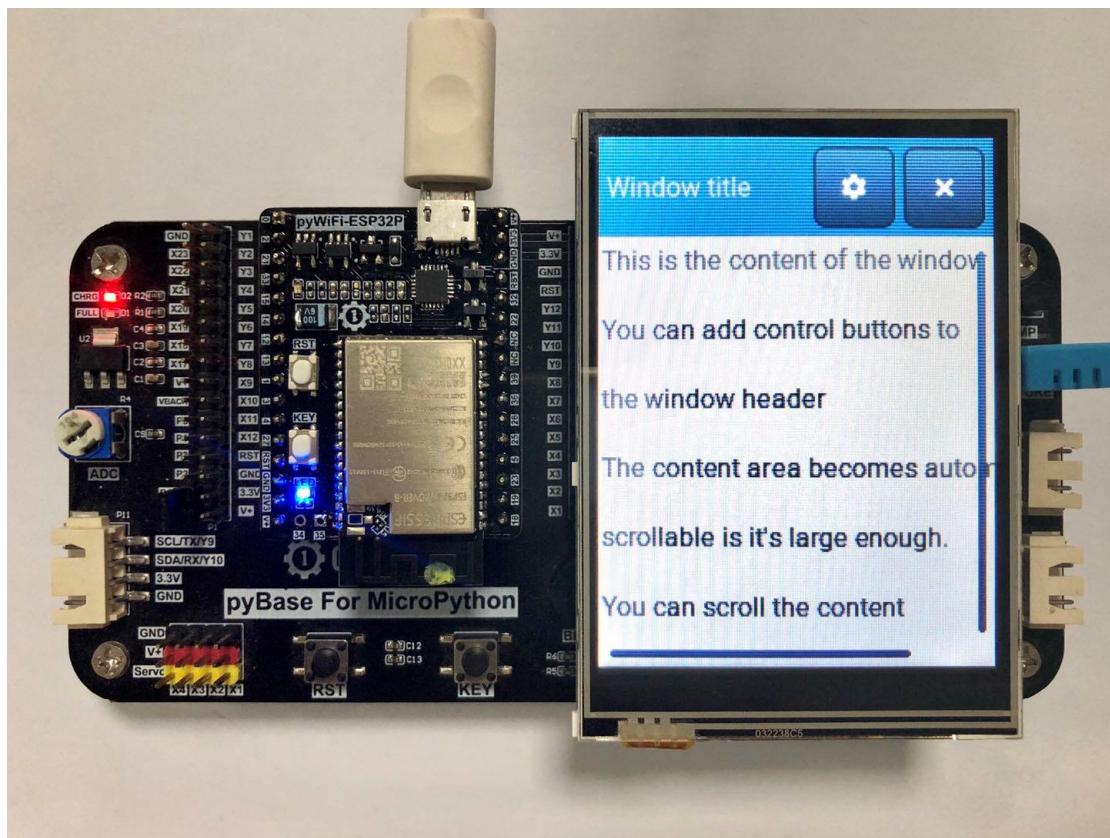


图 8-81 实验结果

### ● 总结

本节主要讲解了 LVGL 中窗口的使用方法。

## 8.4 项目应用

### 8.4.1 网络时钟

- 前言：

日常生活中我们的时钟走着走着可能会出现偏差，这时候最方便的方法就是通过互联网获取当前时间信息，从而更新本地时间。这用到了 NTP 服务器【Network Time Protocol (NTP)】是用来使计算机时间同步化的一种协议，它可以使计算机对其服务器或时钟源（如石英钟，GPS 等等）做同步化，它可以提供高精准度的时间校正。



图 8-82 时钟

- 实验平台：

pyWiFi-ESP32P + 3.2 寸 LCD 开发套件。

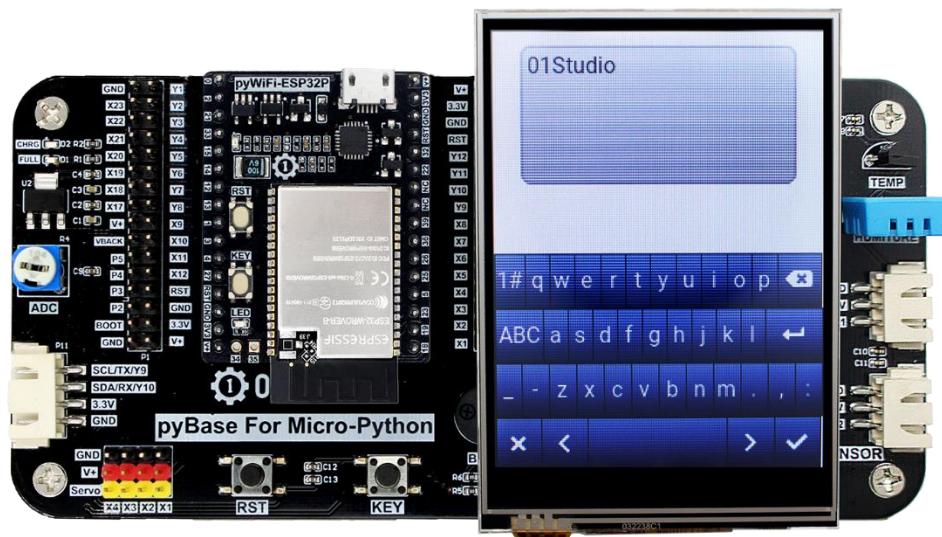


图 8-83 pyWiFi-ESP32P + 3.2 寸 LCD 开发套件

### ● 实验目的：

编程实现 NTP 时钟，并显示温湿度数据。

### ● 实验讲解：

Micropython 基于 ESP32 平台的库自带了 RTC 实时时钟以及 NTP 功能，也就是说我们可以结合之前的内容，可以轻松编程实现在线时钟同步。Wifi 连接的相关对象用法这里不再重复。

先来看看 NTP 的用法：

| 构造函数                           |
|--------------------------------|
| <code>import ntptime</code>    |
| 直接导入模块                         |
| 使用方法                           |
| <code>ntptime.settime()</code> |
| 将当前 RTC 时间设置成 NTP 获取的时间。       |

表 8-27 NTP 对象

编程思路如下：

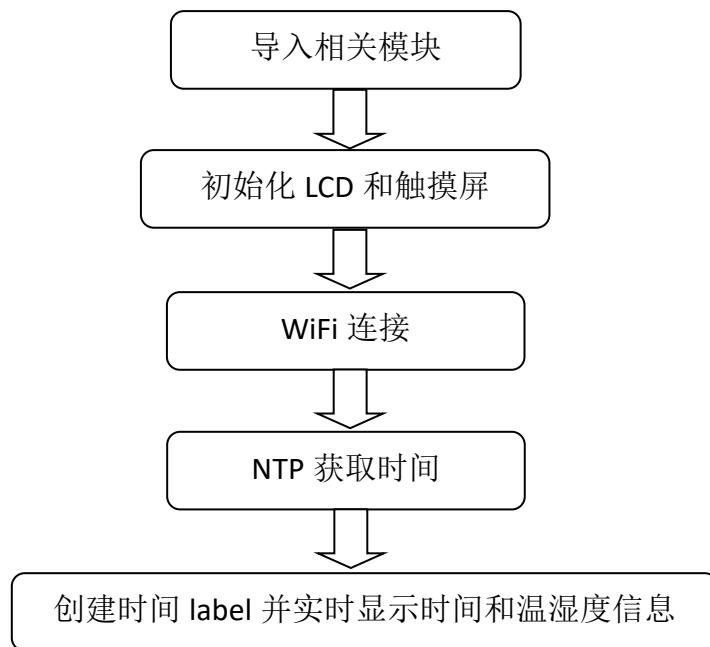


图 8-84 编程流程图

参考代码如下：

```

...
实验名称: NTP(网络时钟)

版本: v1.0

日期: 2020.7

作者: 01Studio 【www.01Studio.org】

...

#####
省略 LCD 初始化、触摸屏校准代码, 详情请参考例程代码文件
#####

#RTC 初始化

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_list = ['', '', '']

rtc = RTC()

#WIFI 连接函数,连接成功后更新时间

def WIFI_Connect():

 wlan = network.WLAN(network.STA_IF) #STA 模式

 wlan.active(True) #激活接口

 start_time=time.time() #记录时间做超时判断

 if not wlan.isconnected():

 print('connecting to network...')

 #输入 WIFI 账号密码

 wlan.connect('01Studio', '88888888')

 while not wlan.isconnected():

 #超时判断,15 秒没连接成功判定为超时

```

```
 if time.time()-start_time > 15 :

 print('WIFI Connected Timeout!')

 break

if wlan.isconnected():

 #串口打印信息

 print('network information:', wlan.ifconfig())

for i in range(5): #最多尝试获取 5 次时间

 try:

 ntptime.settime()

 print(rtc.datetime())

 time.sleep_ms(500)

 return None

 except:

 print("Can not get time!")

#####
NTP Clock
#####

if TOUCH_READY:

 WIFI_Connect() #连接 WiFi

#Create a style based on style_plain
mystyle = lv.style_t(lv.style_plain)
#mystyle.text.font = roboto_80 # font roboto 80 pixel
mystyle.text.font = lv.font_roboto_28
```

```

background top color (main), 0xRRGGBB
mystyle.body.main_color = lv.color_hex(0x000000)

background bottom color (gradient), 0xRRGGBB
mystyle.body.grad_color = lv.color_hex(0x000000)
mystyle.text.color = lv.color_hex(0xffffffff) # text-colour, 0xRRGGBB

#Create screen and labels
scr = lv.obj()
scr.set_style(mystyle)

LOGO = lv.label(scr)
LOGO.set_pos(0,0)
LOGO.set_drag(True)
LOGO.set_text("01Studio")

WIFI_ICON = lv.label(scr)
WIFI_ICON.set_pos(200,0)
WIFI_ICON.set_drag(True)
WIFI_ICON.set_text(lv.SYMBOL.WIFI)

DATE = lv.label(scr)
DATE.set_pos(0,64)
DATE.set_drag(True)

TIME = lv.label(scr)
TIME.set_pos(0,128)
TIME.set_drag(True)

TEMP = lv.label(scr)
TEMP.set_pos(0,192)

```

```

TEMP.set_drag(True)

HUMI = lv.label(scr)
HUMI.set_pos(0,256)
HUMI.set_drag(True)

a=0

#Create content

while(1):

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

 #北京时间，月、日、星期需要适当调整
 datetime[4]=datetime[4]+8 #北京时间，东八区

 if datetime[4] >= 24:
 datetime[4]=datetime[4]%24
 if datetime[1] in [1,3,5,7,8,10,12]: #大月
 datetime[2] = (datetime[2]+1)%32
 else: datetime[2] = (datetime[2]+1)%31
 datetime[3] = (datetime[3]+1)%8

 #显示日期
 DATE.set_text(str(datetime[0]) + '-' + str(datetime[1]) + '-' +
 str(datetime[2]) + ' ' + week[datetime[3]])

 # 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”以达到较佳的显示效果

 for i in range(4, 7):
 if datetime[i] < 10:

```

```

 time_list[i - 4] = "0"
else:
 time_list[i - 4] = ""

TIME.set_text(time_list[0] + str(datetime[4]) + ':' +
 time_list[1] + str(datetime[5]) + ':' + time_list[2] +
 str(datetime[6]))

a=a+1 #控制 DHT11 采集时间，大于 2 秒间隔

if a ==7:
 d.measure() #温湿度采集
 a=0
 TEMP.set_text(str(d.temperature())+' C')
 HUMI.set_text(str(d.humidity())+' %')

300ms 刷屏一次

lv.scr_load(scr)
time.sleep_ms(300)

```

### ● 实验结果：

运行代码，可以看见当成功获取 NTP 时间后，LCD 显示时钟以及采集到的温湿度数据。Label 标签被定义成了可拖动，所以用户可以根据自己的喜好自行对显示进行布局。



图 8-85 实验结果

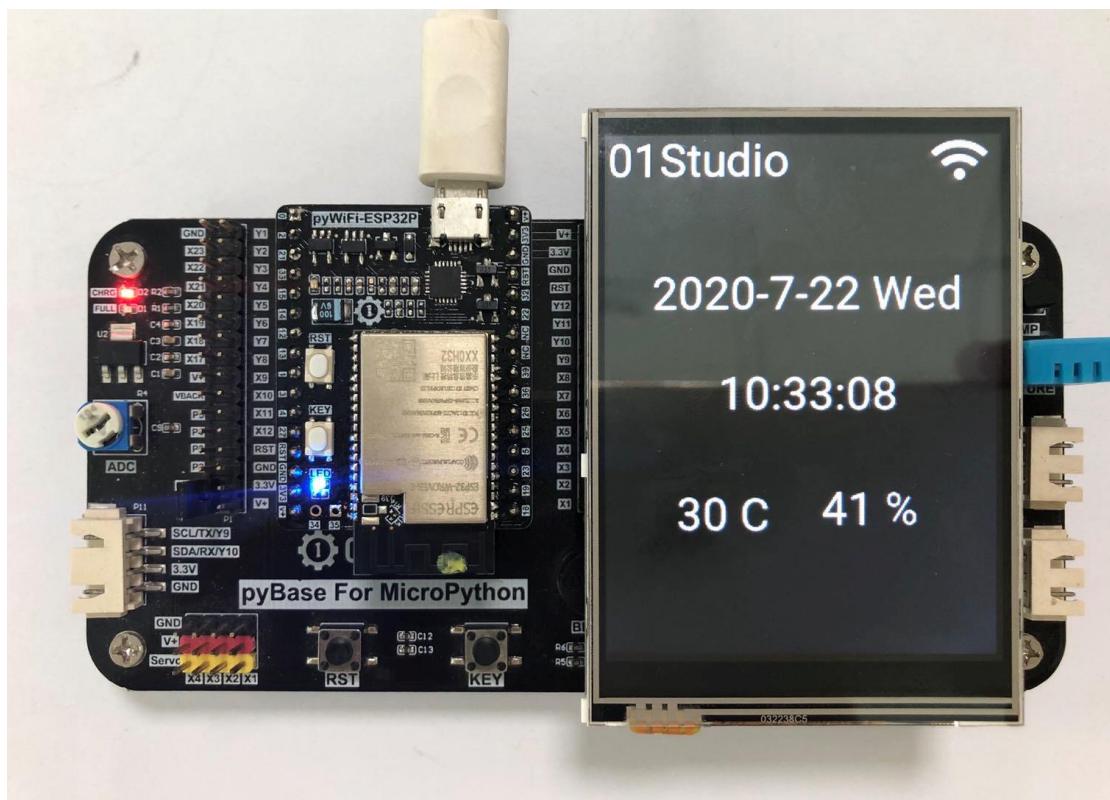


图 8-86 可拖动 label 实现自己喜欢的布局

## ● 总结

本节结合了 NTP、WiFi、RTC 以及 LVGL 内容，实现了一个简易的实时时钟和温湿度计，但 UI 明显是比较简洁的，有兴趣的用户可以结合前面 LVGL 小部件内容，尝试实现自己精美的 UI 时钟界面。

# 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

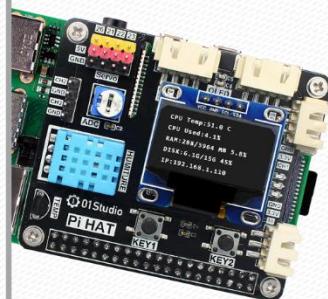


01Studio  
-让编程变得简单有趣-

# 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