

# 普中 ESP32 开发攻略--基于 MicroPython



开源共享

携手共进

普中

普中 ESP32 开发攻略--基于 MicroPython .....	1
第 1 章 如何使用本攻略 .....	1
产品购买地址: .....	2
第 2 章 开发板功能及使用介绍 .....	4
2.1 开发板功能介绍 .....	5
2.1.1 核心板功能介绍 .....	5
2.1.2 底板功能介绍 .....	8
2.2 开发板使用方法 .....	9
2.2.1 开发软件 Thonny 安装 .....	9
2.2.2 CH340 驱动安装 .....	13
2.2.3 REPL 串口交互调试 .....	16
2.2.4 文件系统 .....	20
2.2.5 程序下载运行 .....	22
2.2.6 更新固件 .....	25
课后作业 .....	28
第 3 章 Python 基础 .....	29
3.1 编程基础 .....	30
3.1.1 标识符 .....	30
3.1.2 关键字 .....	31
3.1.3 变量 .....	32
3.1.4 语句 .....	33
3.1.5 代码注释 .....	33
3.1.6 模块 .....	34
3.2 数字类型的数据 .....	35
3.2.1 数据类型 .....	35
3.2.2 整数类型 .....	35
3.2.3 浮点类型 .....	36
3.2.4 复数类型 .....	36
3.2.5 布尔类型 .....	37
3.2.6 数字类型相互转换 .....	37
3.3 运算符 .....	39
3.3.1 算术运算符 .....	39
3.3.2 比较运算符 .....	39
3.3.3 逻辑运算符 .....	39
3.3.4 位运算符 .....	40
3.3.5 赋值运算符 .....	42
3.4 程序流程控制 .....	42
3.4.1 分支语句 .....	42
3.4.2 循环语句 .....	45
3.4.3 跳转语句 .....	47
3.4.4 pass 语句 .....	48
3.5 容器类型数据 .....	48
3.5.1 序列 .....	48

3.5.2 列表 .....	51
3.5.3 元组 .....	54
3.5.4 集合 .....	55
3.5.5 字典 .....	58
3.6 字符串 .....	62
3.6.1 字符串的表示方式 .....	62
3.6.2 字符串与数字相互转换 .....	64
3.6.3 格式化字符串 .....	65
3.6.4 操作字符串 .....	68
3.7 函数 .....	70
3.7.1 定义函数 .....	70
3.7.2 调用函数 .....	71
3.7.3 函数参数传递 .....	71
3.7.4 变量作用域 .....	74
3.7.5 函数特殊形式 .....	75
3.7.6 常用内置函数 .....	76
3.8 类与对象 .....	77
3.8.1 面向对象 .....	77
3.8.2 类与对象 .....	78
3.8.3 构造方法与析构方法 .....	81
3.8.4 类方法和静态方法 .....	83
3.8.5 继承 .....	85
3.8.6 多态 .....	89
3.9 异常处理 .....	90
3.9.1 除零异常 .....	90
3.9.2 捕获异常 .....	91
3.9.3 抛出异常 .....	96
3.9.4 自定义异常 .....	98
3.10 模块 .....	99
3.10.1 模块的概念 .....	99
3.10.2 模块的导入方式 .....	100
3.10.3 自定义模块 .....	101
3.10.4 模块的导入特性 .....	102
3.10.5 Python 中的包 .....	104
课后作业 .....	105
第 4 章 LED 实验 .....	106
4.1 实验介绍 .....	107
4.1.1 实验简介 .....	107
4.1.2 实验目的 .....	107
4.1.3 MicroPython 函数使用 .....	107
4.2 硬件设计 .....	108
4.3 软件设计 .....	109
4.3.1 点亮第一个 LED 实验 .....	109
4.3.2 LED 闪烁实验 .....	110

4.3.3 LED 流水灯实验 .....	111
4.4 实验现象 .....	113
4.4.1 点亮第一个 LED 实验 .....	113
4.4.2 LED 闪烁实验 .....	114
4.4.3 LED 流水灯实验 .....	115
课后作业 .....	116
第 5 章 蜂鸣器实验 .....	117
5.1 实验介绍 .....	118
5.1.1 实验简介 .....	118
5.1.2 实验目的 .....	118
5.1.3 MicroPython 函数使用 .....	118
5.2 硬件设计 .....	118
5.3 软件设计 .....	120
5.4 实验现象 .....	120
课后作业 .....	121
第 6 章 继电器实验 .....	122
6.1 实验介绍 .....	123
6.1.1 实验简介 .....	123
6.1.2 实验目的 .....	123
6.1.3 MicroPython 函数使用 .....	123
6.2 硬件设计 .....	124
6.3 软件设计 .....	125
6.4 实验现象 .....	126
课后作业 .....	127
第 7 章 按键控制实验 .....	128
7.1 实验介绍 .....	129
7.1.1 实验简介 .....	129
7.1.2 实验目的 .....	129
7.1.3 MicroPython 函数使用 .....	129
7.2 硬件设计 .....	130
7.3 软件设计 .....	131
7.4 实验现象 .....	133
课后作业 .....	134
第 8 章 直流电机实验 .....	135
8.1 实验介绍 .....	136
8.1.1 实验简介 .....	136
8.1.2 实验目的 .....	137
8.1.3 MicroPython 函数使用 .....	137
8.2 硬件设计 .....	137
8.3 软件设计 .....	139
8.4 实验现象 .....	140
课后作业 .....	140
第 9 章 步进电机实验 .....	141
9.1 实验介绍 .....	142

9.1.1 实验简介 .....	142
9.1.2 实验目的 .....	144
9.1.3 MicroPython 函数使用 .....	144
9.2 硬件设计 .....	144
9.3 软件设计 .....	145
9.4 实验现象 .....	148
课后作业 .....	149
第 10 章 外部中断实验 .....	149
10.1 实验介绍 .....	150
10.1.1 实验简介 .....	150
10.1.2 实验目的 .....	151
10.1.3 MicroPython 函数使用 .....	151
10.2 硬件设计 .....	151
10.3 软件设计 .....	152
10.4 实验现象 .....	154
课后作业 .....	155
第 11 章 定时器中断实验 .....	155
11.1 实验介绍 .....	156
11.1.1 实验简介 .....	156
11.1.2 实验目的 .....	156
11.1.3 MicroPython 函数使用 .....	156
11.2 硬件设计 .....	157
11.3 软件设计 .....	158
11.4 实验现象 .....	159
课后作业 .....	159
第 12 章 PWM 呼吸灯实验 .....	160
12.1 实验介绍 .....	161
12.1.1 实验简介 .....	161
12.1.2 实验目的 .....	161
12.1.3 MicroPython 函数使用 .....	162
12.2 硬件设计 .....	163
12.3 软件设计 .....	163
12.4 实验现象 .....	164
课后作业 .....	165
第 13 章 串口通信实验 .....	165
13.1 实验介绍 .....	166
13.1.1 实验简介 .....	166
13.1.2 实验目的 .....	167
13.1.3 MicroPython 函数使用 .....	167
13.2 硬件设计 .....	168
13.3 软件设计 .....	169
13.4 实验现象 .....	170
课后作业 .....	171
第 14 章 ADC 实验 .....	172

14.1 实验介绍 .....	173
14.1.1 实验简介 .....	173
14.1.2 实验目的 .....	173
14.1.3 MicroPython 函数使用 .....	173
14.2 硬件设计 .....	174
14.3 软件设计 .....	175
14.4 实验现象 .....	176
课后作业 .....	177
第 15 章 RGB 彩灯实验 .....	178
15.1 实验介绍 .....	179
15.1.1 实验简介 .....	179
15.1.2 实验目的 .....	180
15.1.3 MicroPython 函数使用 .....	180
15.2 硬件设计 .....	180
15.3 软件设计 .....	181
15.4 实验现象 .....	182
课后作业 .....	183
第 16 章 数码管显示实验 .....	184
16.1 实验介绍 .....	185
16.1.1 实验简介 .....	185
16.1.2 实验目的 .....	186
16.1.3 MicroPython 函数使用 .....	186
16.2 硬件设计 .....	187
16.3 软件设计 .....	189
16.4 实验现象 .....	189
课后作业 .....	191
第 17 章 RTC 实时时钟实验 .....	192
17.1 实验介绍 .....	193
17.1.1 实验简介 .....	193
17.1.2 实验目的 .....	193
17.1.3 MicroPython 函数使用 .....	193
17.2 硬件设计 .....	194
17.3 软件设计 .....	194
17.4 实验现象 .....	194
课后作业 .....	195
第 18 章 DS1302 实时时钟实验 .....	196
18.1 实验介绍 .....	197
18.1.1 实验简介 .....	197
18.1.2 实验目的 .....	197
18.1.3 MicroPython 函数使用 .....	197
18.2 硬件设计 .....	198
18.3 软件设计 .....	199
18.4 实验现象 .....	200
课后作业 .....	201

第 19 章 DS18B20 温度传感器实验 .....	202
19.1 实验介绍 .....	203
19.1.1 实验简介 .....	203
19.1.2 实验目的 .....	204
19.1.3 MicroPython 函数使用 .....	204
19.2 硬件设计 .....	206
19.3 软件设计 .....	207
19.4 实验现象 .....	208
课后作业 .....	209
第 20 章 DHT11 湿湿度传感器实验 .....	209
20.1 实验介绍 .....	210
20.1.1 实验简介 .....	210
20.1.2 实验目的 .....	211
20.1.3 MicroPython 函数使用 .....	211
20.2 硬件设计 .....	212
20.3 软件设计 .....	213
20.4 实验现象 .....	214
课后作业 .....	215
第 21 章 超声波测距实验 .....	215
21.1 实验介绍 .....	216
21.1.1 实验简介 .....	216
21.1.2 实验目的 .....	217
21.1.3 MicroPython 函数使用 .....	217
21.2 硬件设计 .....	218
21.3 软件设计 .....	219
21.4 实验现象 .....	220
课后作业 .....	221
第 22 章 红外遥控实验 .....	222
22.1 实验介绍 .....	223
22.1.1 实验简介 .....	223
22.1.2 实验目的 .....	225
22.1.3 MicroPython 函数使用 .....	225
22.2 硬件设计 .....	225
22.3 软件设计 .....	226
22.4 实验现象 .....	227
课后作业 .....	229
第 23 章 舵机实验 .....	230
23.1 实验介绍 .....	231
23.1.1 实验简介 .....	231
23.1.2 实验目的 .....	232
23.1.3 MicroPython 函数使用 .....	233
23.2 硬件设计 .....	233
23.3 软件设计 .....	234
23.4 实验现象 .....	235

课后作业 .....	237
第 24 章 OLED 液晶显示实验 .....	238
24.1 实验介绍 .....	239
24.1.1 实验简介 .....	239
24.1.2 实验目的 .....	240
24.1.3 MicroPython 函数使用 .....	240
24.2 硬件设计 .....	242
24.3 软件设计 .....	243
24.4 实验现象 .....	245
课后作业 .....	246
第 25 章 SD 卡实验 .....	247
25.1 实验介绍 .....	248
25.1.1 实验简介 .....	248
25.1.2 实验目的 .....	249
25.1.3 MicroPython 函数使用 .....	249
25.2 硬件设计 .....	250
25.3 软件设计 .....	251
25.4 实验现象 .....	252
课后作业 .....	253
第 26 章 WIFI 实验-连接路由器 .....	254
26.1 实验介绍 .....	255
26.1.1 实验简介 .....	255
26.1.2 实验目的 .....	255
26.1.3 MicroPython 函数使用 .....	255
26.2 硬件设计 .....	257
26.3 软件设计 .....	257
26.4 实验现象 .....	258
课后作业 .....	258
第 27 章 WIFI 实验-Socket 通信 .....	259
27.1 实验介绍 .....	260
27.1.1 实验简介 .....	260
27.1.2 实验目的 .....	262
27.1.3 MicroPython 函数使用 .....	262
27.2 硬件设计 .....	263
27.3 软件设计 .....	264
27.4 实验现象 .....	265
课后作业 .....	268
第 28 章 WIFI 实验-MQTT 通信 .....	269
28.1 实验介绍 .....	270
28.1.1 实验简介 .....	270
28.1.2 实验目的 .....	271
28.1.3 MicroPython 函数使用 .....	271
28.2 硬件设计 .....	272
28.3 软件设计 .....	272

28.3.1 发布者 .....	272
28.3.2 订阅者 .....	274
28.4 实验现象 .....	275
28.4.1 发布者 .....	275
28.4.2 订阅者 .....	278
课后作业 .....	281
第 29 章 WIFI 实验-手机控制 LED .....	282
29.1 实验介绍 .....	283
29.1.1 实验简介 .....	283
29.1.2 实验目的 .....	283
29.1.3 MicroPython 函数使用 .....	283
29.2 硬件设计 .....	284
29.3 软件设计 .....	285
29.4 实验现象 .....	287
课后作业 .....	288

普中PRECHIN

# 第 1 章 如何使用本攻略

学习本开发攻略主要参考的文档有《Quick reference for the ESP32》，这是 MicroPython 官方手册，里面包含了使用 MicroPython 操控 ESP32 单片机内部资源介绍与范例，非常详细。大家在学习 ESP32 的时候可以参考这个文档，也可直接在网页版浏览，网址：

<http://docs.micropython.org/en/latest/esp32/quickref.html>

可惜官方提供的是英文版本，不过网上有很多翻译中文版本的，大家可以自行搜索查阅。当然本攻略已将 MicroPython 在 ESP32 的使用讲解非常详细，可直接参考本攻略即可。

本攻略编写风格是：

- (1) 实验介绍
- (2) MicroPython 模块函数讲解
- (3) 硬件讲解
- (4) 软件分析
- (5) 实验现象

通过上述几大块的介绍让您快速掌握 ESP32 使用 MicroPython 开发。

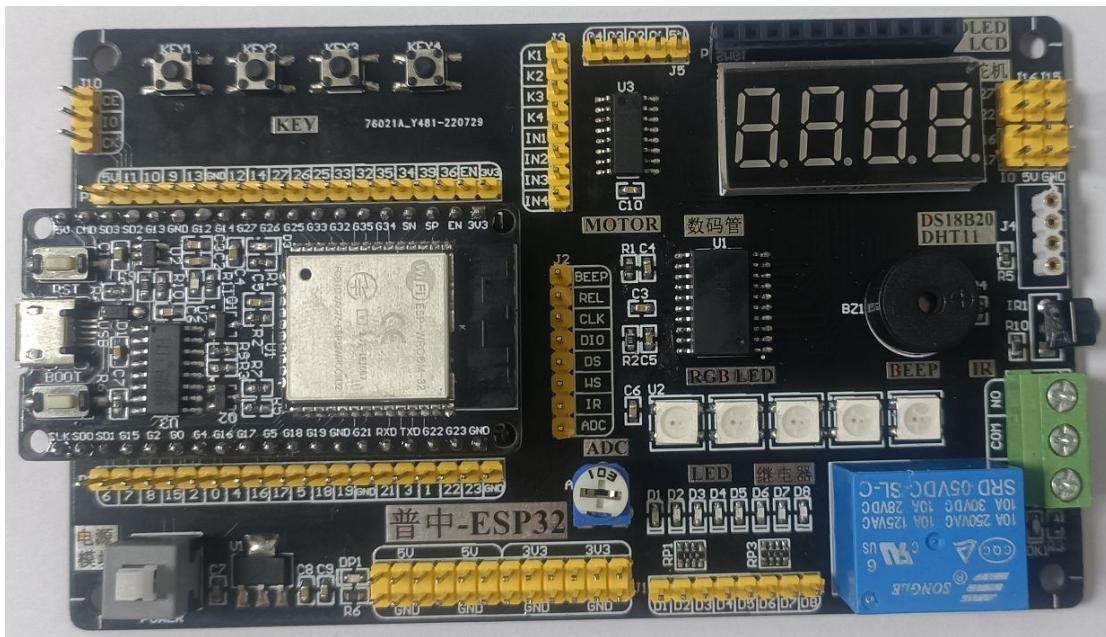
那么问题来了，为什么要学习 MicroPython？Micropython 的由来，这得益于 Damien George 这位伟大的计算机工程师，Damien George 每天都会使用 Python 工作，他有一天冒出一个大胆的想法：能否用 Python 来控制单片机，从而实现对机器人的操作呢？Python 语言本身就是一款简单上手的脚本语言，一些非计算机专业的爱好者都选 Python 语言作为入门语言，但是美中不足的是，它不能实现对一些底层的操作，在硬件领域毫不起眼。至于 Python 强大之处相信不用给大家介绍也都听过他的威名。所以，Damien George 利用 6 个月时间打造了 MicroPython，这就是 MicroPyhton 的由来。

MicroPython 它基于 ANSIC，语法跟 Python3 基本一致，拥有独立的解析器、编译器、虚拟机和类库等，所以可以在所支持的硬件平台上使用 Python 语言对硬件控制。目前他支持基于 32-bit 的 ARM 处理器，比如说 STM32F401、STM32F405、STM32F407、ESP32 等，现如今支持的处理器更加丰富，比如 ESP8266 等，有兴趣的朋友可以去官网看看。

趣的可以去官网了解下。

本开发攻略配套的实验平台为：普中-ESP32，这款开发板出厂搭配的是ESP32-WROOM-32 模组，学习的时候如果配套该硬件平台做实验，必会达到事半功倍的效果，可以省去中间移植时遇到的各种问题。

普中-ESP32 外观图如下：（此图没有采用任何包装拍摄）



在学习的过程中，如果遇到什么问题，可以到我们技术论坛：[www.prechin.net](http://www.prechin.net) 发帖交流，也可联系我们技术电话：0755-21509063，我们共同进步。

鉴于作者水平有限，难免会有纰漏，还请热心的读者指正并发到论坛，好让我们改进，祝您生活学习愉快。在 ESP32 的学习过程中，我们与您同行！

## 产品购买地址：

(1) 购买地址（普中授权店铺）

<http://www.prechin.net/forum.php?mod=viewthread&tid=38746&extra=>

(2) 资料下载

<http://prechin.net/forum.php?mod=viewthread&tid=35264&extra=page%3D1>

(3) 技术支持

普中官网：[www.prechin.cn](http://www.prechin.cn)

普中论坛: [www.prechin.net](http://www.prechin.net)

技术电话: 0755-21509063 (转技术)

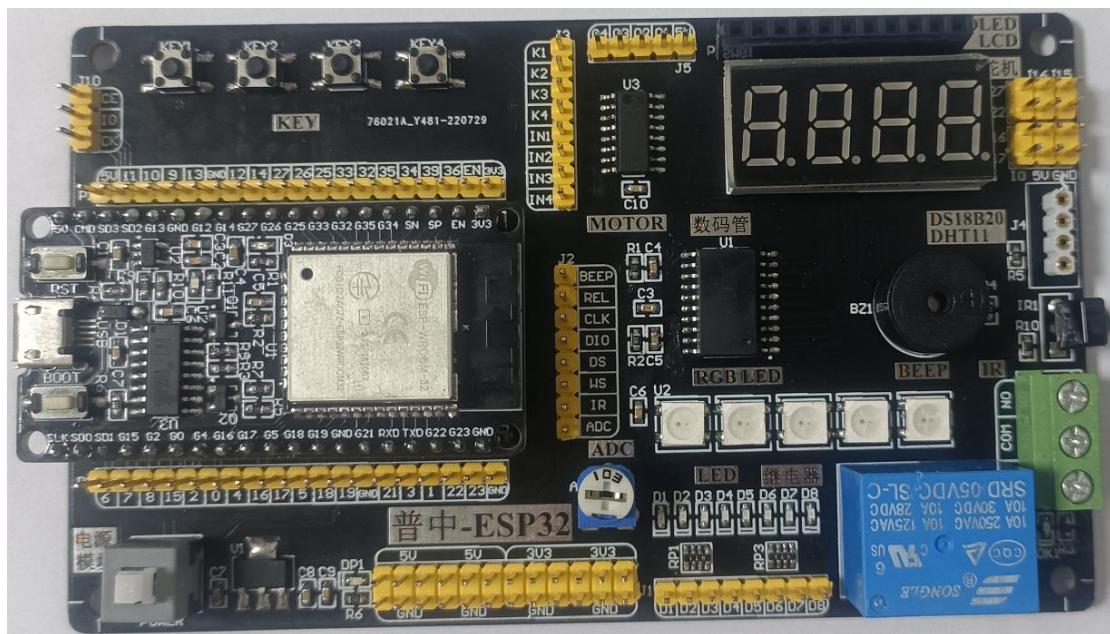
普中 PRECHIN

## 第 2 章 开发板功能及使用介绍

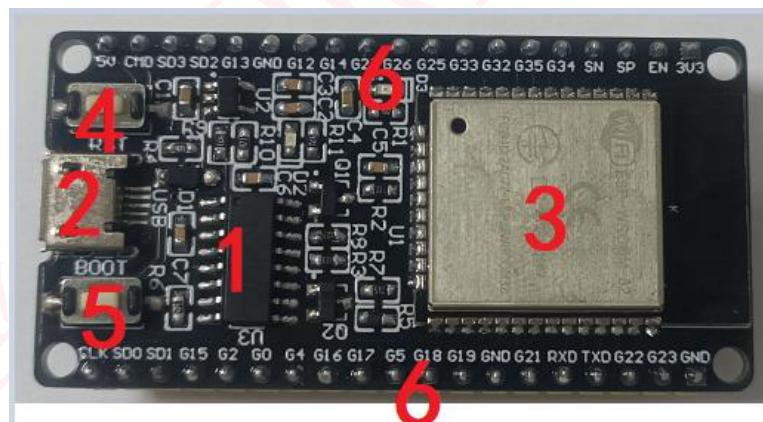
本章将向大家介绍普中-ESP32 开发板(以下简称 ESP32)的功能及使用方法，通过本章的学习，让大家能快速上手开发板的学习。本章分为如下几部分内容：

- 2.1 开发板功能介绍
- 2.2 开发板使用方法

## 2.1 开发板功能介绍

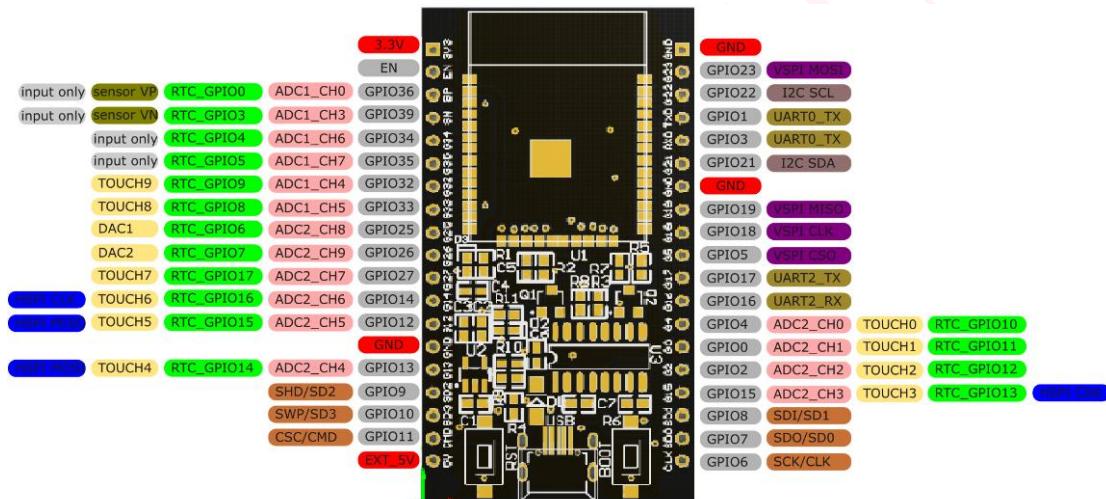


### 2.1.1 核心板功能介绍



ESP32核心板模块资源	
1	CH340模块，用于USB转串口与ESP32对接
2	MicroUSB接口，用于程序下载、固件升级和电源输入等
3	ESP32-WROOM-32模组
4	复位按键
5	BOOT按键
6	ESP32模组引出GPIO

核心板引脚如下：



下表显示了哪些管脚最适合用作输入和输出，哪些管脚需要谨慎使用。

GPIO	Input	Output	Notes
0 pulled up	OK		Strapping 管脚
1 TX Pin	OK		debug output at boot
2 OK	OK		Strapping 管脚
3 OK	RX Pin		HIGH at boot
4 OK	OK		
5 OK	OK		Strapping 管脚
6 ✗	✗		模组集成 SPI flash
7 ✗	✗		模组集成 SPI flash
8 ✗	✗		模组集成 SPI flash
9 ✗	✗		模组集成 SPI flash
10 ✗	✗		模组集成 SPI flash
11 ✗	✗		模组集成 SPI flash
12 OK	OK		Strapping 管脚
13 OK	OK		
14 OK	OK		
15 OK	OK		Strapping 管脚
16 OK	OK		
17 OK	OK		
18 OK	OK		
19 OK	OK		
21 OK	OK		
22 OK	OK		
23 OK	OK		
25 OK	OK		
26 OK	OK		
27 OK	OK		
32 OK	OK		
33 OK	OK		
34 OK			输入引脚
35 OK			输入引脚
36 OK			输入引脚
39 OK			输入引脚

绿色突出显示的管脚可以使用。黄色突出显示的可以使用，但需要注意，因为它们可能在启动时有意外行为。不建议将红色突出显示的管脚用作输入或输出，因为被模组中 SPI-FLASH 占用。

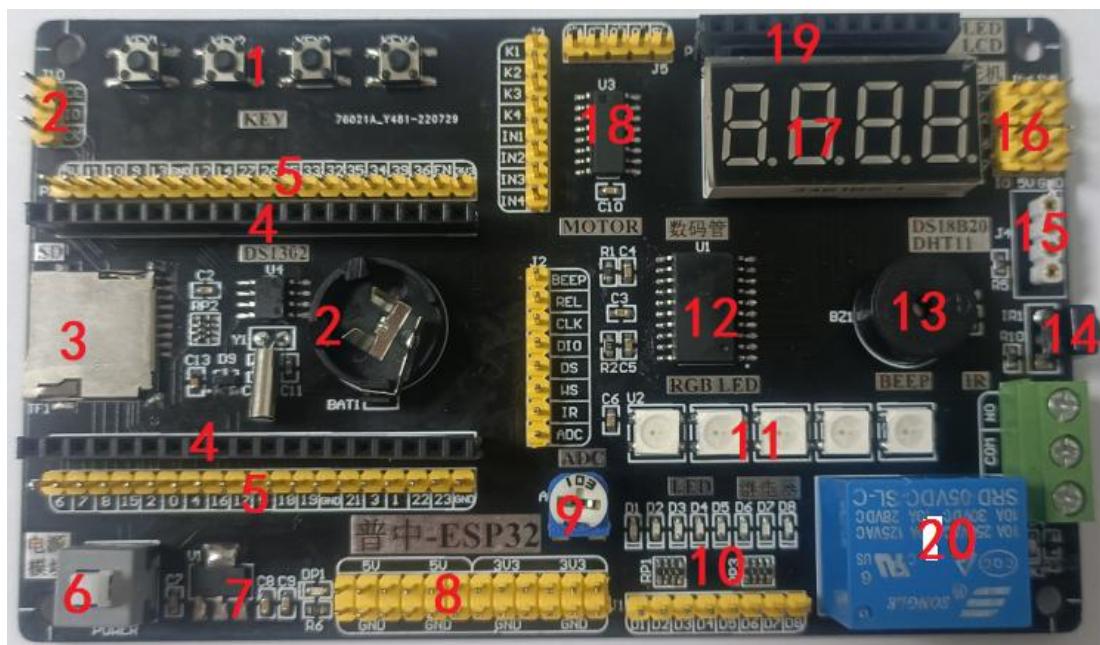
注意：使用 Wi-Fi 时不能使用 ADC2 管脚。因此，如果您使用 Wi-Fi，并且无法从 ADC2 GPIO 获取值，则可以考虑改用 ADC1 GPIO，这应该可以解决您的问题。

启用 (EN) 是 3.3V 调节器的启用引脚。它被拉起来了，所以接地使 3.3V 调节器失效。这意味着您可以使用连接到按钮的该管脚来重新启动 ESP32。

关于 ESP32 管脚详细功能介绍，大家可以参考文档 “\7--ESP32 相关资料 \esp\_wroom\_32\_datasheet\_cn.pdf”，也可以直接查看这个网址：

<https://lingshunlab.com/book/esp32/esp32-pinout-reference>

## 2.1.2 底板功能介绍



ESP32底板资源	
1	按键模块: 4个
2	DS1302时钟模块
3	TF卡座
4	ESP32核心板接口
5	ESP32核心板引出IO
6	底板电源开关
7	3.3V稳压模块
8	5V&3.3V电源输入输出口
9	ADC电位器
10	LED模块: 8个
11	RGB彩灯: 5个
12	数码管驱动模块: TM1637芯片
13	无源蜂鸣器
14	红外接收头
15	DS18B20&DHT11传感器接口
16	SG90舵机接口: 4路
17	共阳数码管
18	电机驱动模块: ULN2003芯片
19	OLED&LCD液晶接口
20	继电器模块

## 2.2 开发板使用方法

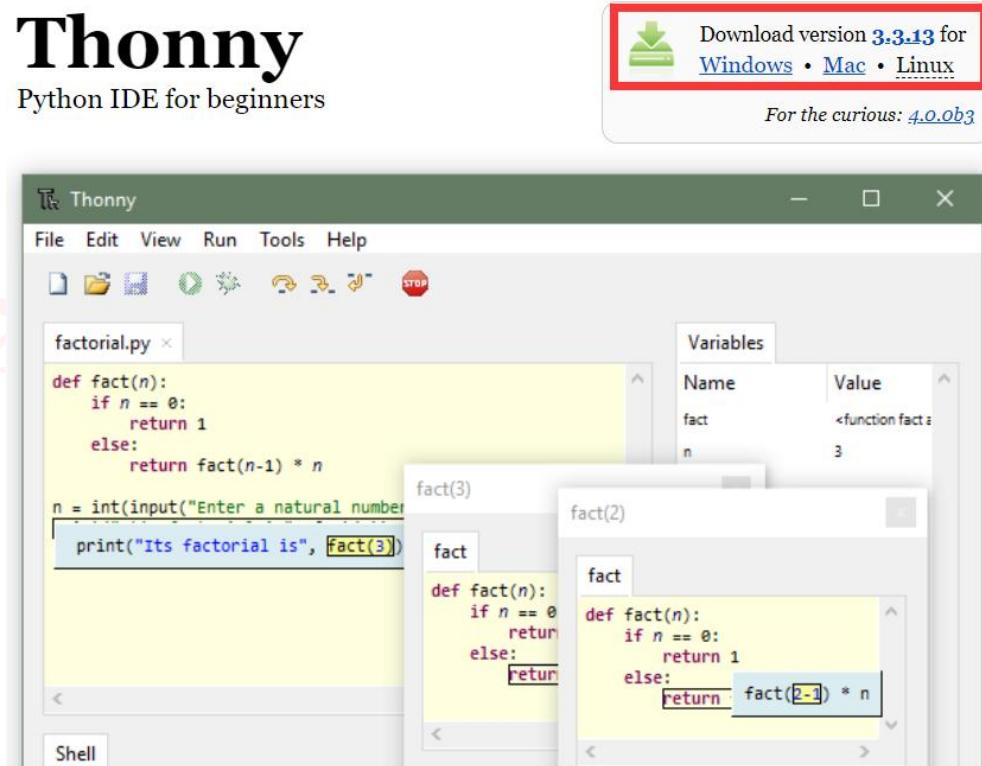
### 2.2.1 开发软件 Thonny 安装

#### 2.2.1.1 获取 Thonny 软件

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。使用 Thonny 还有一个方便之处，可直接在该软件中实现程序开发和下载。

要在电脑上成功安装 Thonny，首先必须要有安装包，我们可以在 Thonny 官网下载：<https://thonny.org/>，打开界面如下图所示。



可以根据电脑系统下载对应版本，例如作者使用 Windows 系统，则直接选择“Windows”下载。

我们资料内已给大家提供好安装包，在资料“\5--开发工具\1-MicroPython 开发工具”内，大家直接使用即可，省去了查找下载的时间。我们使用的 Thonny 是 3.3.11 版本，如果后面出了更高的版本选择性升级即可，不过也没有必要使用最新的，用习惯了一个软件就行。

使用我们提供下载好的软件包，其内部含有如下图所示文件。

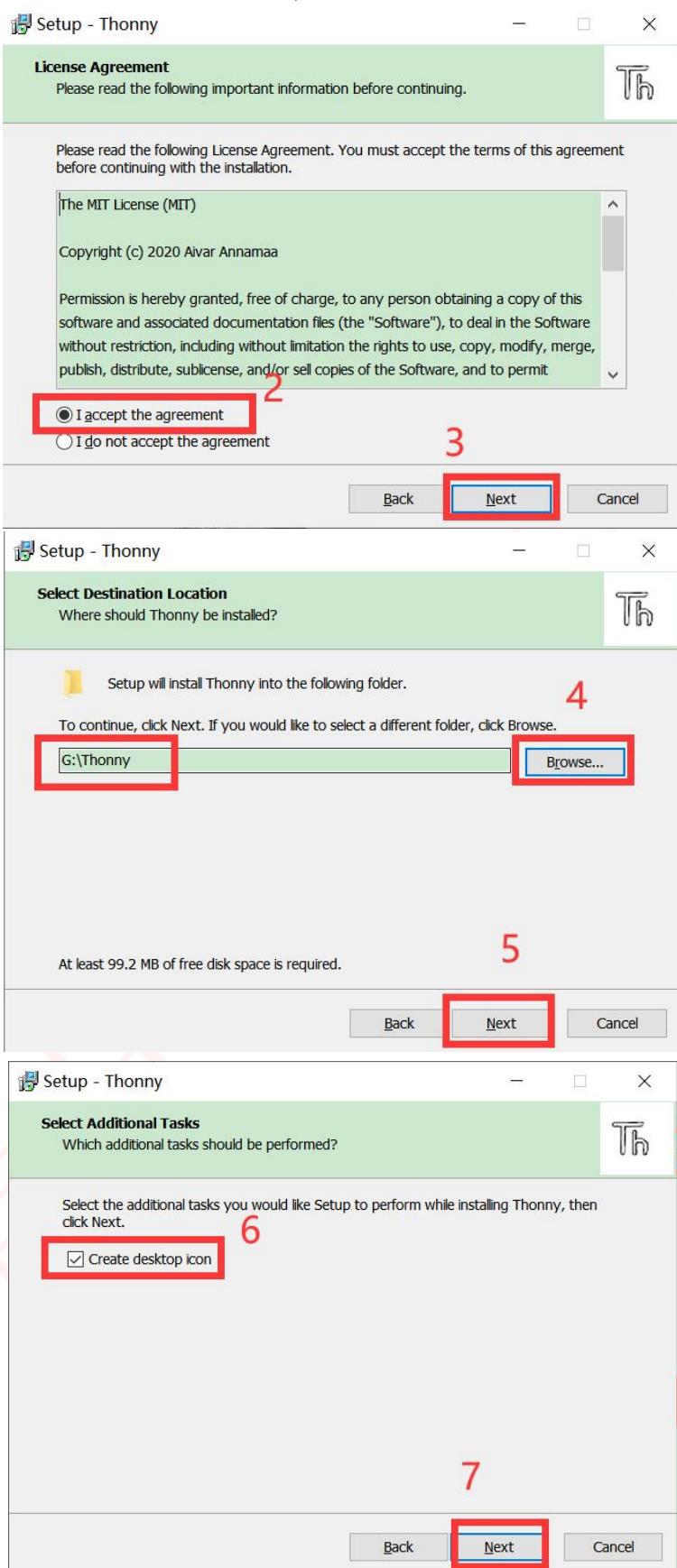
5--开发工具 > 1-MicroPython 开发工具

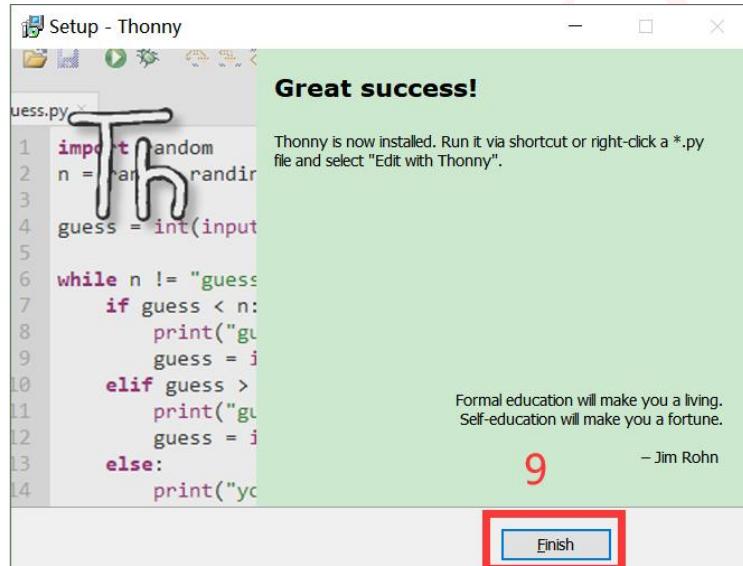
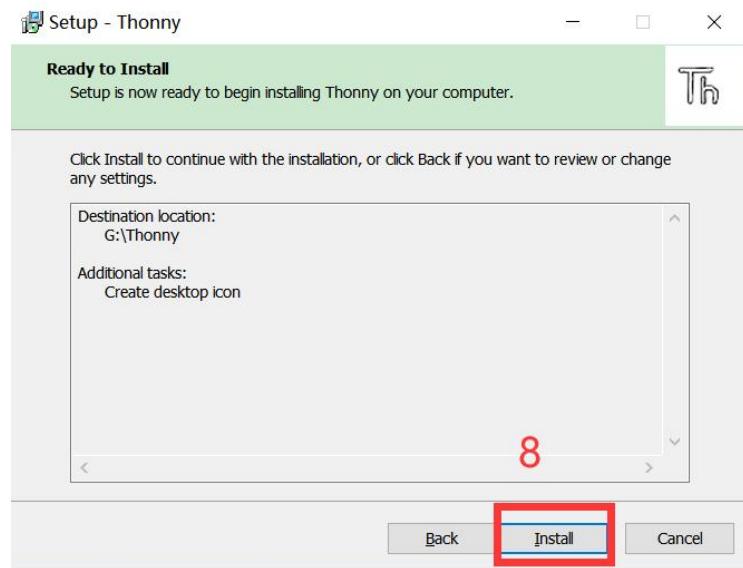


### 2.2.1.2 Thonny 软件安装

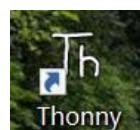
鼠标右键“thonny-3.3.11.exe”这个应用程序，使用管理员模式运行/打开，弹出如下所示对话框，点击 Next，选择好存放路径即可，注意：**存放路径不能出现中文或特殊字符**，详细操作步骤如下：



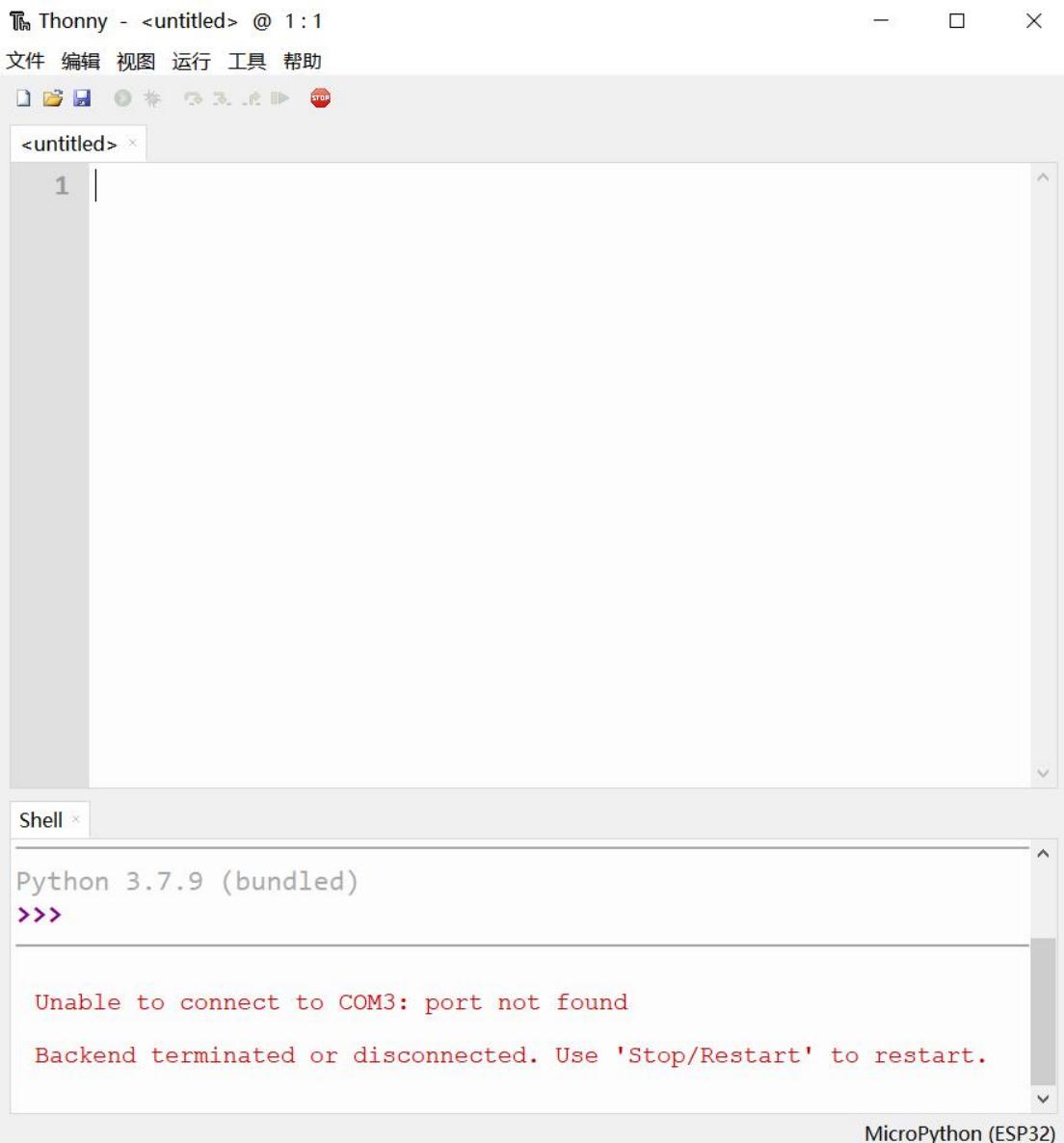




安装完成后，在电脑桌面上可以看到一个快捷图标，如下所示：



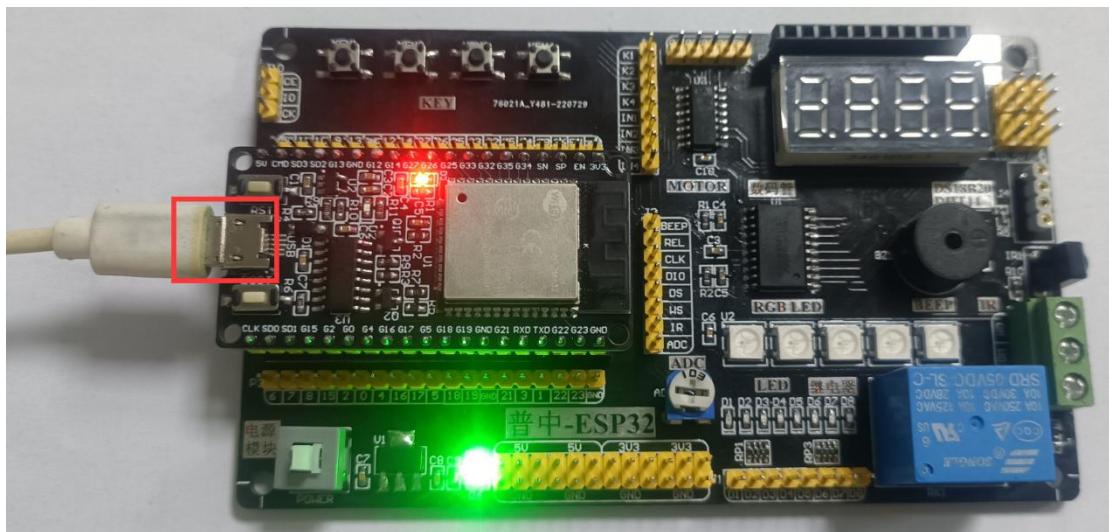
双击打开该软件界面如下所示：



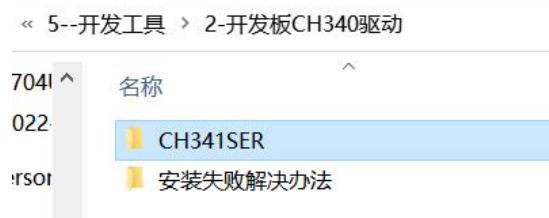
## 2.2.2 CH340 驱动安装

上一节，我们已经介绍了开发板各个模块的功能，下面我们看看如何使用这款开发板。

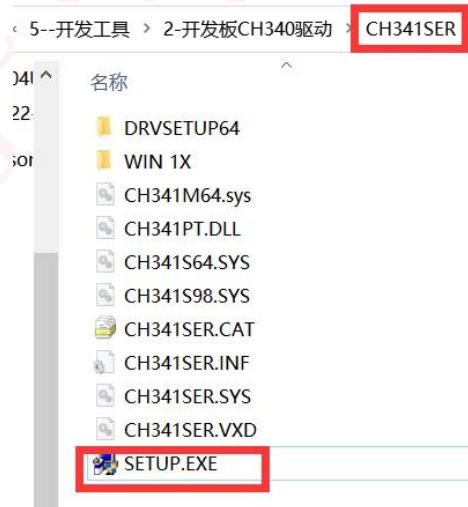
首先，拿到开发板后，要安装 USB 转串口 CH340 驱动，对于大多数电脑系统，**将 USB 线连接电脑和开发板的 USB 接口后会自动检测安装 CH340 驱动**，连接如下：



如果您的电脑没有自动安装 CH340 驱动，没关系，可以手动安装，打开资料目录“\5--开发工具\2-开发板 CH340 驱动”，如下：



打开“CH341SER”文件夹，如下：



双击 SETUP. EXE 应用程序，出现如下界面，点击安装即可。



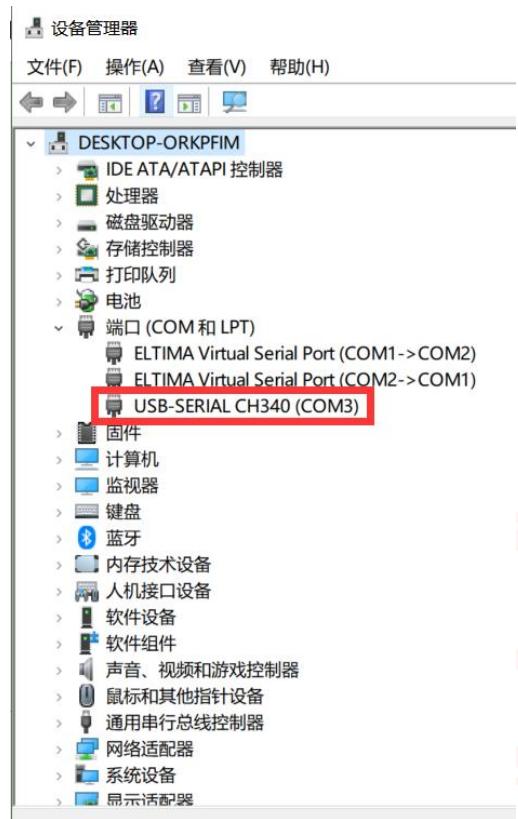
一段时间后，如果安装成功会显示如下界面：（前提：必须使用 USB 线将电脑 USB 口和开发板 USB 接口连接）



如果显示“驱动预安装成功”或者“驱动安装失败”等提示信息，表明驱动安装不成功。这时可以打开资料目录“\5--开发工具\2-开发板 CH340 驱动\安装失败解决办法”，安装对应的驱动。如果还是安装失败，可以重新换条 USB 线（支持安卓手机数据线）再次安装测试；如果依然安装失败，可以手动将“\5--开发工具\2-开发板 CH340 驱动\安装失败解决办法”对应自己系统的文件夹内 serenum.sys 和 serial.sys 这两个文件，拷贝到 C:\Windows\System32\drivers 文件夹下。如果该文件夹下本来就有这两个文件，提示无法替换，那么请先删除这两个原有的文件，再拷贝过去即可。然后再试试能否成功安装驱动，通过上述操作一般就可以解决串口无法安装/使用的问题了。

如果还安装失败，你的电脑系统是 WIN8 以上，可以试试关闭电脑数字签名，具体方法请百度“数字签名如何关闭”。假如还是安装失败，请联系我们技术电话：0755-21509063，或者到我们公司论坛：[www.prechin.net](http://www.prechin.net) 发帖咨询，我们技术看到后会第一时间给您处理。

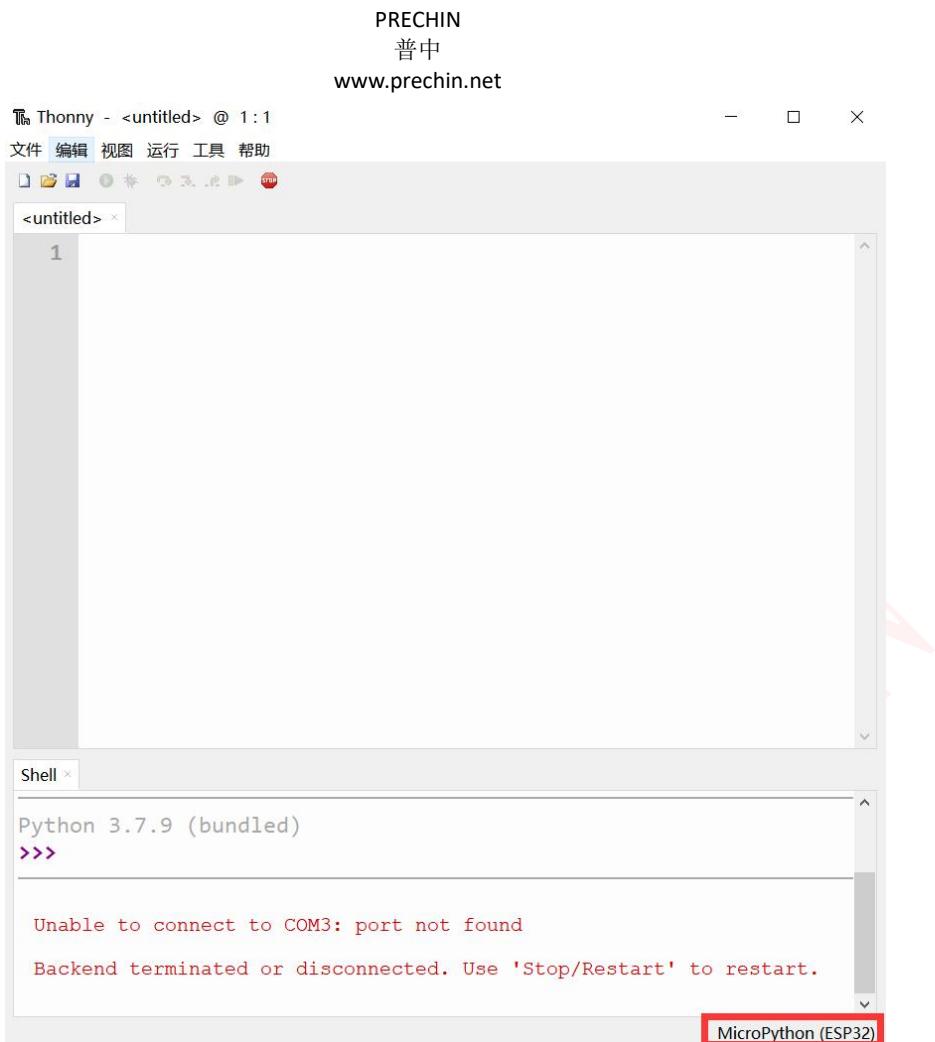
驱动安装成功后，可打开电脑设备管理器，检查是否有 CH340 端口显示，如下：（不同电脑识别的串口号可能不同，COM3 是作者电脑识别的）



### 2.2.3 REPL 串口交互调试

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

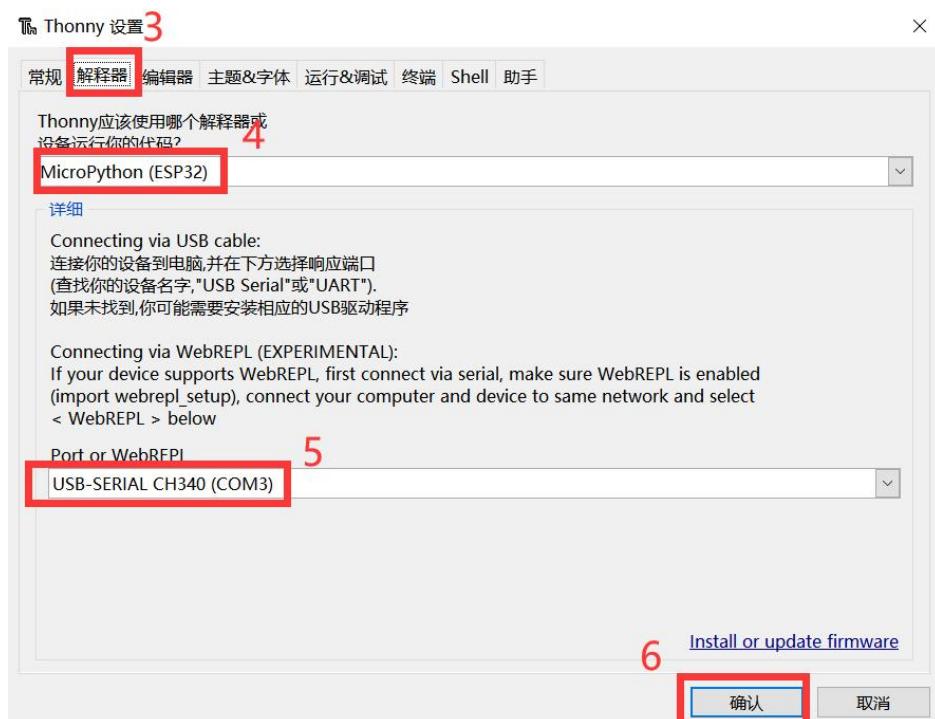
我们打开 Thonny 软件，将开发板连接到电脑，点击右下角，如下所示：



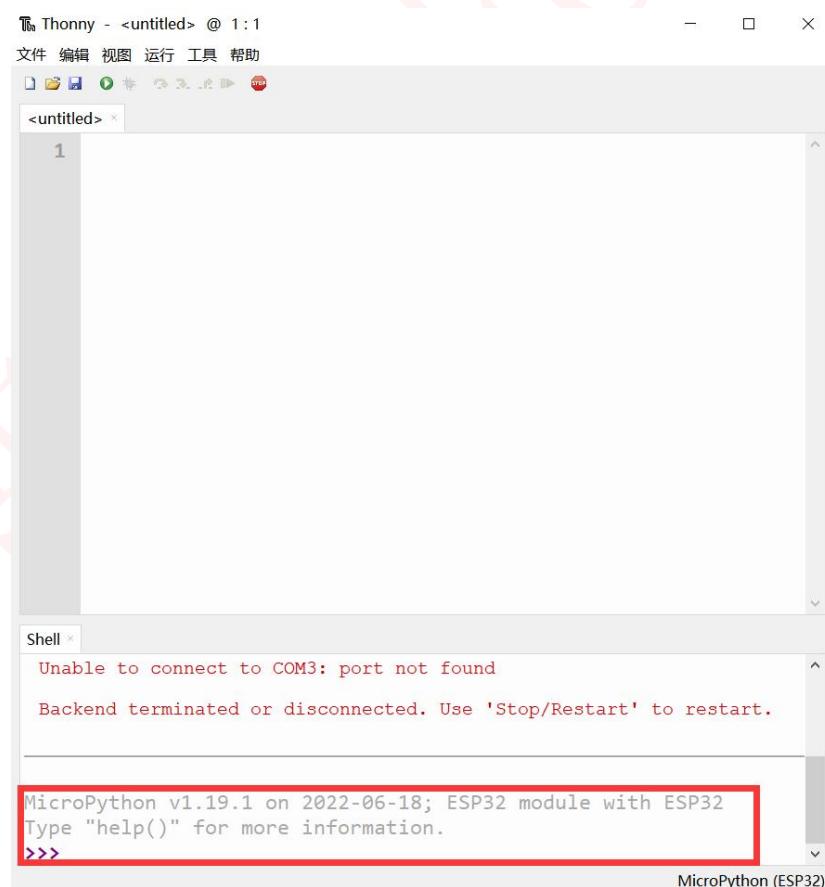
在弹出的列表选择：Configure interpreter，如下：



在“解释器”选项卡中选择“MicroPython（ESP32）”和开发板对应的串口号，点击确认，如下：



连接成功后可以在 shell (串口终端) 看到固件的相关信息，如下：



我们在 Shell 里面输入 print("Hello World!"), 按回车, 可以看到打印出 Hello World! 字符：（注意：一定要是英文双引号，否则运行报错）

PRECHIN  
普中  
www.prechin.net

```
Shell < Type "help()" for more information.
>>> print("Hello World!")
Traceback (most recent call last):
  File "<stdin>", line 1
    SyntaxError: invalid syntax
>>> print("Hello World!")
Hello World!
>>> |
```

MicroPython (ESP32)

再输入  $1+1$ , 按回车:

```
Shell <
  File "<stdin>", line 1
    SyntaxError: invalid syntax
>>> print("Hello World!")
Hello World!
>>> 1+1
2
>>> |
```

MicroPython (ESP32)

REPL 还有一个强大的功能就是输出代码错误信息，在后面我们编写代码运行时，如果程序出错，出错信息将通过 REPL 打印。例如我们将前面 print 中的双引号改写成中文双引号看报错信息，如下：

Thonny - H:\普中-ESP32开发板资料\4--实验程序\-----调试程序\test.py @ 1:1

文件 编辑 视图 运行 工具 帮助

test.py <

```
1 print("Hello World!")
```

Shell <

```
>>> %Run -c $EDITOR_CONTENT
Traceback (most recent call last):
  File "<stdin>", line 1
    SyntaxError: invalid syntax
>>> |
```

MicroPython (ESP32)

然后根据输出错误信息快速定位并解决。

REPL 终端常用键盘按键:

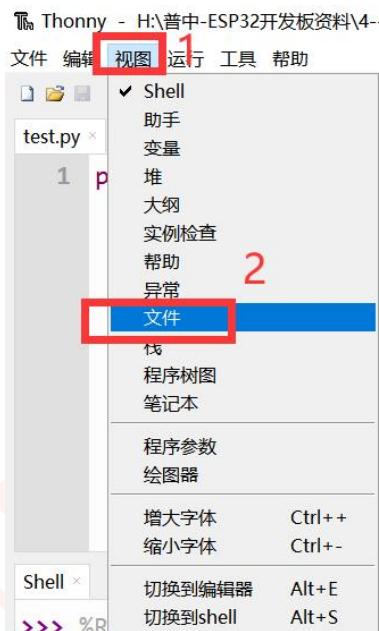
**Ctrl + C** : 打断正在运行的程序（特别是含 While True: 的代码）；

**Ctrl + D** : 软件复位开发板。

## 2.2.4 文件系统

MicroPython 固件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。

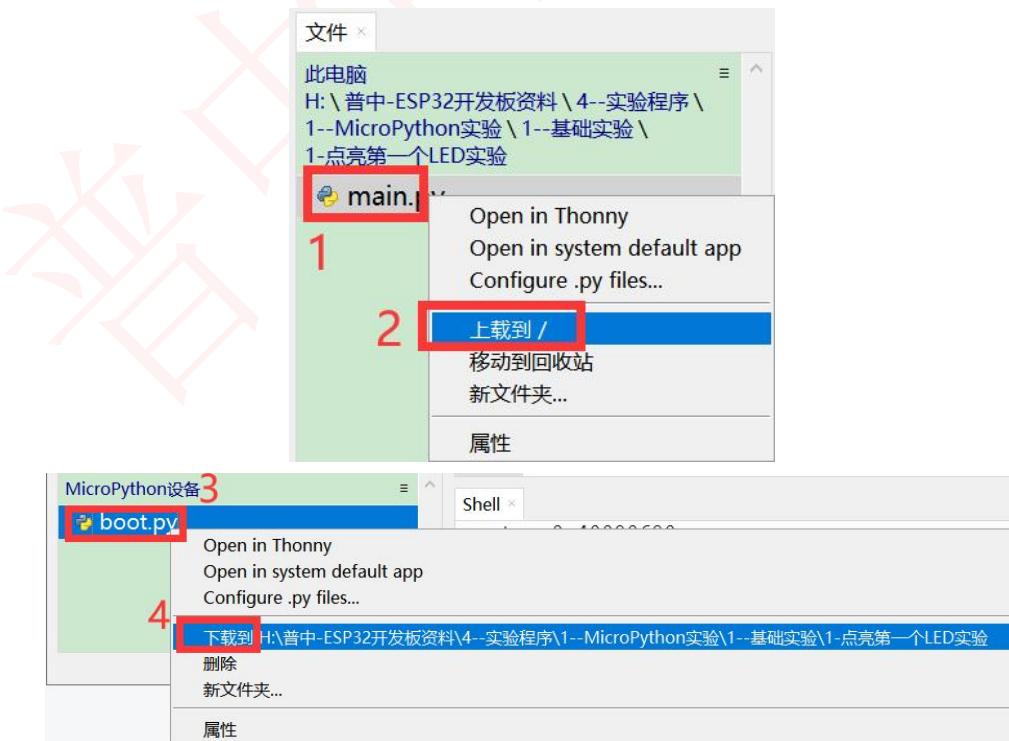
打开 Thonny 软件，点击“视图”选择“文件”，如下：



可以看到左边出现本地和开发板的实时文件浏览窗口：



在电脑文件点击鼠标右键，选择“上载到/”即可将电脑中相关文件发送到开发板内，同样的也可以将开发板上的文件下载到电脑内，非常方便。

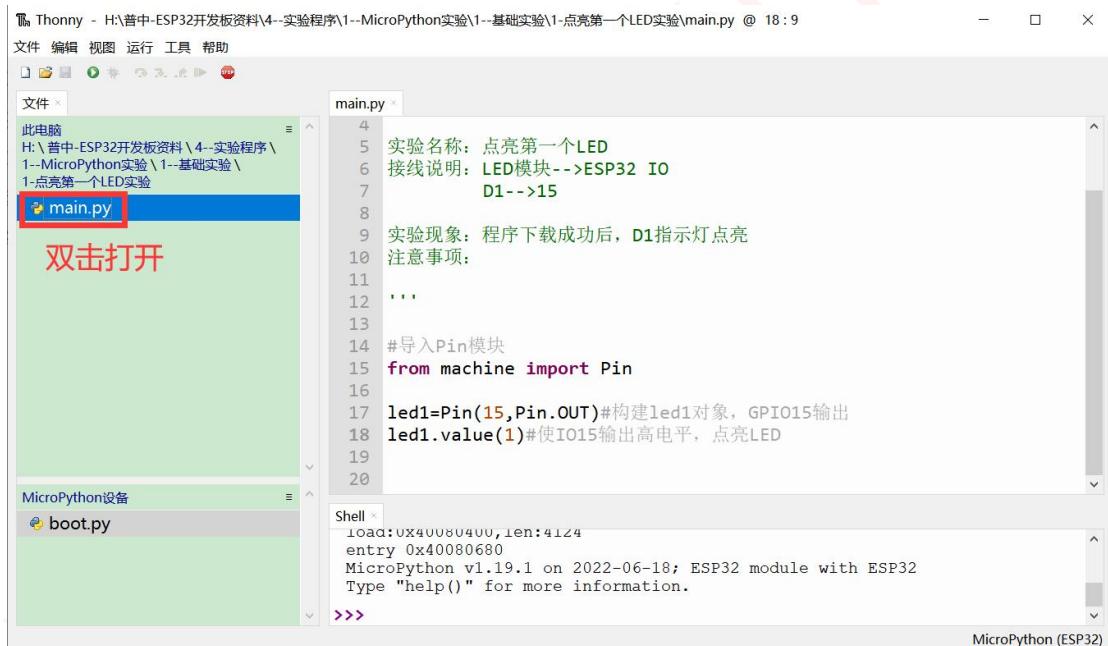


如果实验例程内有多个.py文件，则同样方法全部上传到开发板内即可。

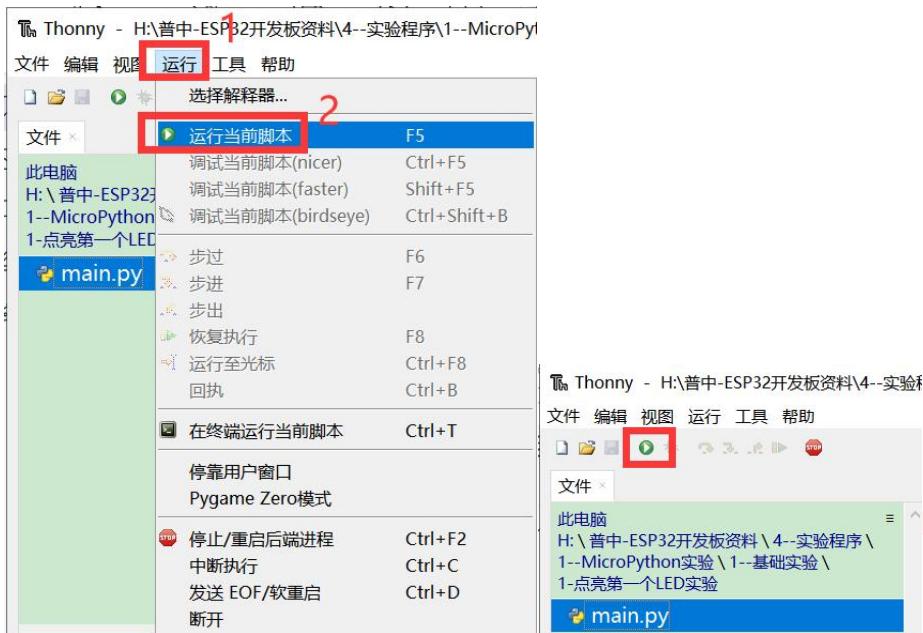
## 2.2.5 程序下载运行

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 的实验，大家暂时先不用理解代码意思，后面章节会有讲解。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：

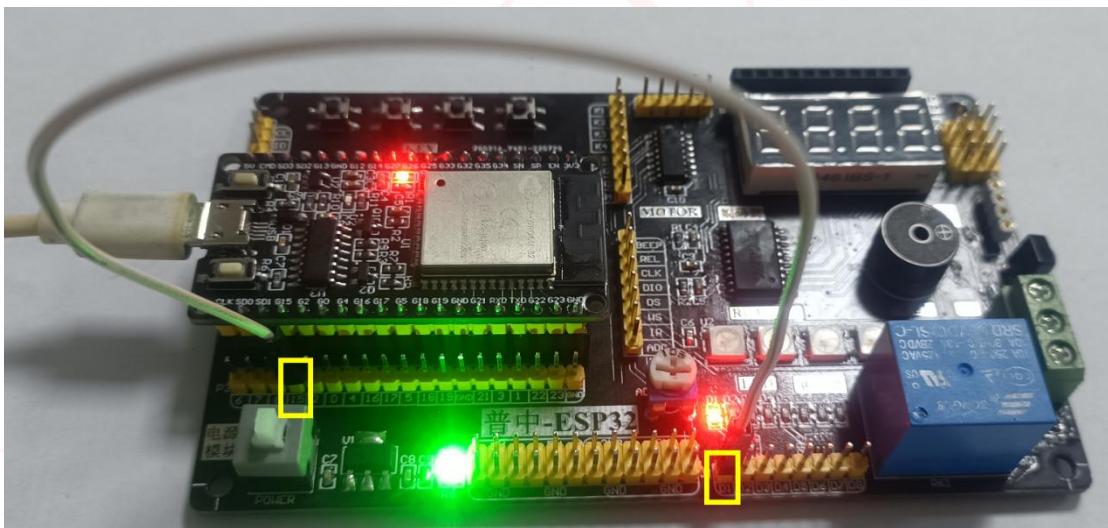
将开发板与电脑连接，打开 Thonny 软件，在 Thonny 左上角电脑文件区域找到“\普中-ESP32 开发板资料\4--实验程序\1--MicroPython 实验\1--基础实验\1-点亮第一个 LED 实验”的 main.py 文件，双击打开后看到右边编程区出现相关代码。



点击“运行”，选择“运行当前脚本”或者直接在工具栏中点绿色按钮图标：



此时程序已在板子中运行，可预先使用一根杜邦线将 P3 端子上的 15 号脚与 LED 模块的 D1 脚连接，可以看到开发板上的 D1 指示灯被点亮，如下：



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：

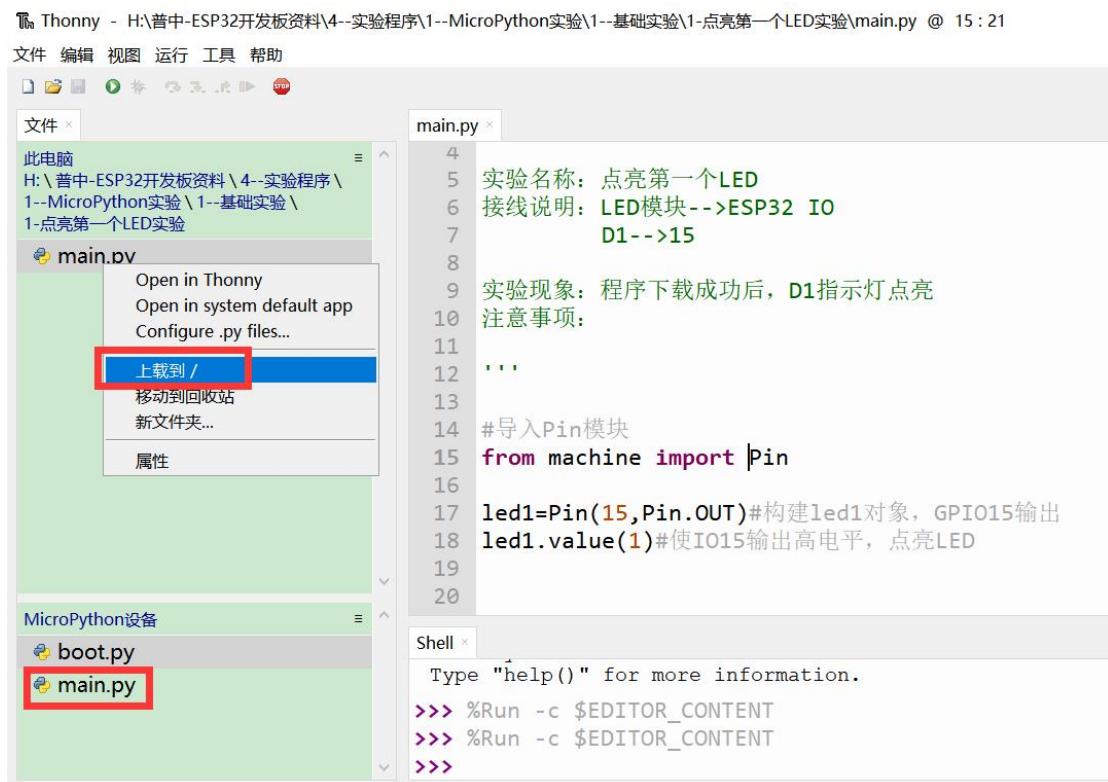
Micropython 上电默认先运行名字为 boot.py 文件，然后在运行 main.py 文件，如果没有 boot.py 那么直接运行 main.py。

**boot.py：一般用于配置初始化参数；**

**main.py：主程序**

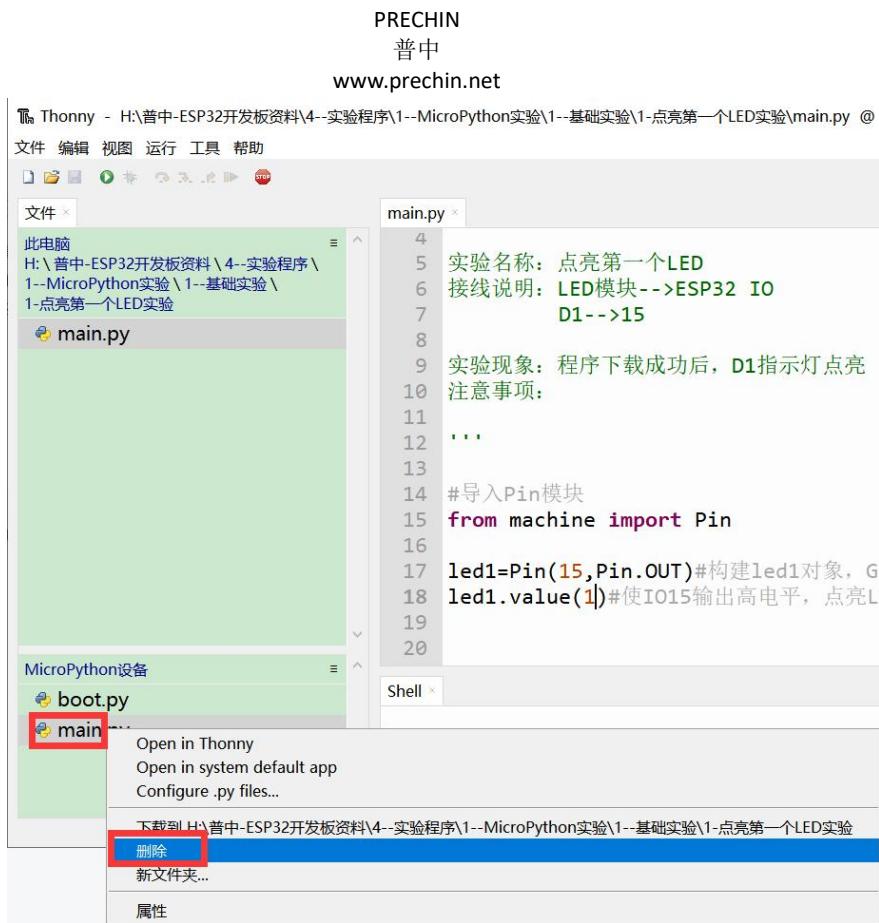
我们只需要将代码以 main.py 文件发送到开发板，此时程序文件就存放在开发板的 FLASH 内，断电不丢失，那么开发板就可以实现上电运行相关程序。

我们将点亮第一个 LED 程序的 main.py 发送到开发板，如下：



按下开发板上的复位键 RST，可以看到 D1 指示灯亮，并且断电重启后依然可以运行点亮。

如果需要删除开发板中的文件，可鼠标右键直接删除。



**注意:** 若程序正在运行, 在删除文件时, 可能会出现一直等待情况, 此时可直接选择取消, 软件已删除完, 只需重新插下 USB 接口, 检查所需删除的文件是否清除。

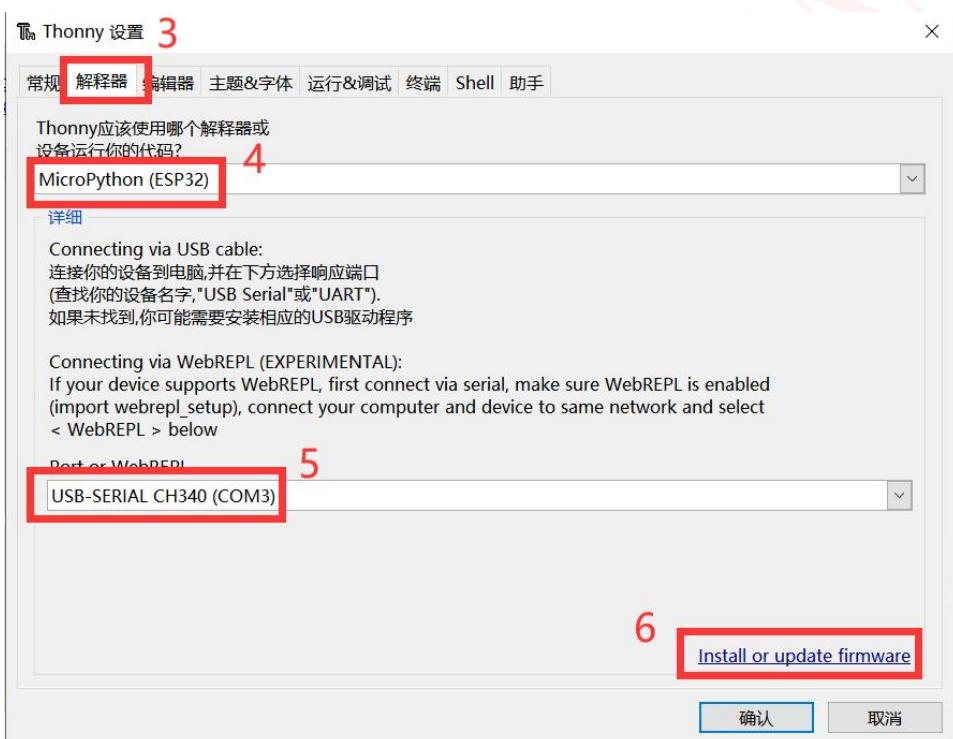
## 2.2.6 更新固件

我们开发板出厂已经烧录好固件, 更新固件是指重新烧写开发板的出厂文件或者是升级的 MicroPython 固件。有 2 种方法可更新开发板固件: ①使用 FLASH 下载工具 “[flash\\_download\\_tool\\_v3.8.8.exe](#)” ; ②使用 Thonny 软件更新。第一种使用 FLASH 下载工具, 大家可以打开资料 “[\5--开发工具\5-ESP32-FLASH 下载工具](#)”里面有软件下载操作界面图片。我们推荐使用 Thonny 软件直接更新固件。操作方法如下:

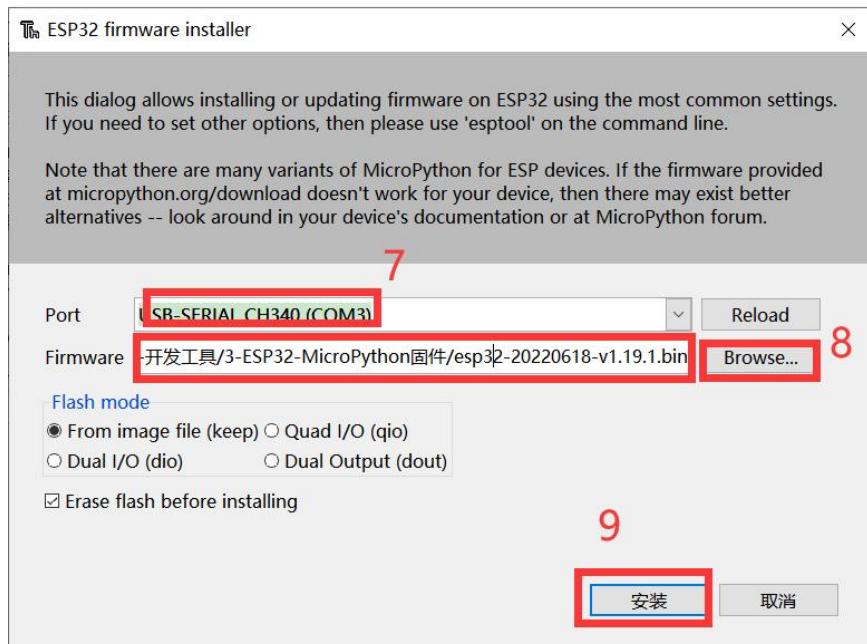
(1) 打开 Thonny 软件, 点击“工具”, 选择“设置...”, 如下:



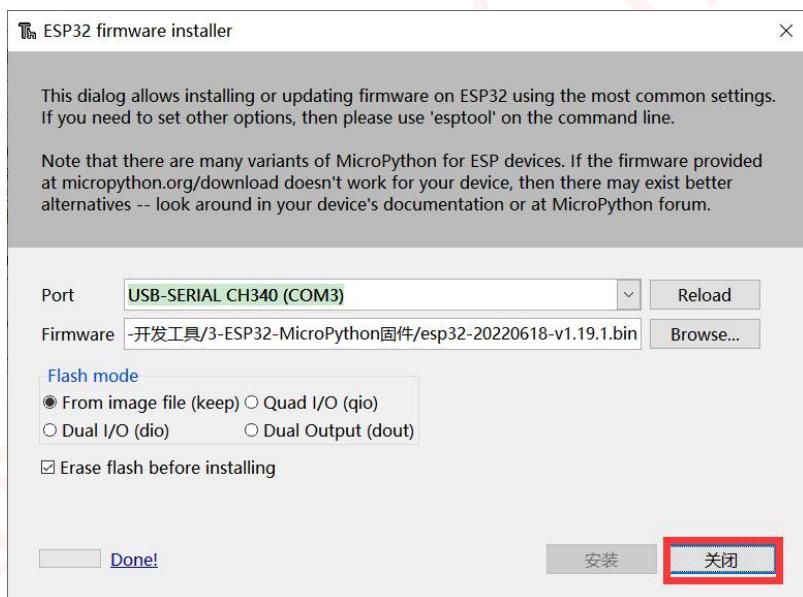
(2) 在设置界面中，点击“解释器”选项卡，选择“MicroPython(ESP32)”和电脑识别的端口，点击“Install or update firmware”，如下：



(3) 选择端口号和要更新的固件文件，点击安装。



(4) 更新完成后，点击关闭即可。



然后在 Shell 中就会显示当前更新固件的版本信息，如下：

The screenshot shows the MicroPython shell output:

```
Shell > MicroPython v1.19.1 on 2022-06-18; ESP32 module with ESP32
Type "help()" for more information.
MicroPython v1.19.1 on 2022-06-18; ESP32 module with ESP32
Type "help()" for more information.
>>>
```

A red arrow points to the second line of text in the shell output, which displays the updated MicroPython version information.

## 课后作业

普中 PRECHIN

## 第 3 章 Python 基础

要学习 MicroPython，首先要有 Python3 语言基础，前面我们已经说过他俩语法基本一致。为了照顾没有 Pyhton 基础的朋友，我们一直在思考应该给大家提供怎么样的 Python 基础教程。Python 相关的书籍和学习资料相信大家都能在网上找到不少，所以本章给大家整理了 Python3 快速学习资料，甚至可以使用它来当作 Python 字典查阅。本章分为如下几部分内容：

- 3.1 编程基础
- 3.2 数字类型的数据
- 3.3 运算符
- 3.4 程序流程控制
- 3.5 容器类型数据
- 3.6 字符串
- 3.7 函数
- 3.8 类与对象
- 3.9 异常处理
- 3.10 模块

## 3.1 编程基础

### 3.1.1 标识符

标识符是变量、函数、类、模块和其他对象的名称。Python 中标识符的命名不是随意的，而是要遵守一定的命名规则，比如说：

- 1、标识符是由字母（A~Z 和 a~z）、下划线和数字组成，但第一个字符不能是数字。
- 2、标识符不能和 Python 中的保留字相同。有关保留字，后续章节会详细介绍。
- 3、Python 中的标识符中，不能包含空格、@、% 以及 \$ 等特殊字符。

例如：下面所列举的标识符是合法的：

UserID

name

mode12

user\_age

以下命名的标识符不合法：

4word #不能以数字开头

try #try 是保留字，不能作为标识符

\$money #不能包含特殊字符

4、在 Python 中，标识符中的字母是严格区分大小写的，也就是说，两个同样的单词，如果大小格式不一样，代表的意义也是完全不同的。比如说下面这 3 个变量之间，就是完全独立、毫无关系的，它们彼此之间是相互独立的个体。

number = 0

Number = 0

NUMBER = 0

5、Python 语言中，以下划线开头的标识符有特殊含义，例如：

- 以单下划线开头的标识符（如 \_width），表示不能直接访问的类属性，

其无法通过 `from.. import*` 的方式导入；

- 以双下划线开头的标识符（如`_add`）表示类的私有成员；
- 以双下划线作为开头和结尾的标识符（如`__init__`），是专用标识符。

因此，除非特定场景需要，应避免使用以下划线开头的标识符。

另外需要注意的是，Python 允许使用汉字作为标识符，例如：

C 语言中文网 = "hello world!"

但我们应尽量避免使用汉字作为标识符，这会避免遇到很多奇葩的错误。

标识符的命名，除了要遵守以上这几条规则外，不同场景中的标识符，其名称也有一定的规范可循，例如：

- 当标识符用作模块名时，应尽量短小，并且全部使用小写字母，可以使用下划线分割多个字母，例如`game_mian`、`game_register` 等。
- 当标识符用作包的名称时，应尽量短小，也全部使用小写字母，不推荐使用下划线，例如`com.mr`、`com.mr.book` 等。
- 当标识符用作类名时，应采用单词首字母大写的形式。例如，定义一个图书类，可以命名为`Book`。
- 模块内部的类名，可以采用“下划线+首字母大写”的形式，如`_Book`；
- 函数名、类中的属性名和方法名，应全部使用小写字母，多个单词之间可以用下划线分割；
- 常量命名应全部使用大写字母，单词之间可以用下划线分割；

有朋友可能会问，如果不遵守这些规范，会怎么样呢？答案是程序照样可以运行，但遵循以上规范的好处是，可以更加直观地了解代码所代表的含义，以`Book` 类为例，我们可以很容易就猜到此类与书有关，虽然将类名改为`a`（或其它）不会影响程序运行，但通常不这么做。

### 3.1.2 关键字

关键字是 python 内置的具有特殊意义的单词，在命名变量名时不要用关键字命名，否则会出现不同原因的报错。Python 中保留关键字如下：

```
False  class  from  or  None  continue  global  pass
True   def  if  raise  and  del  import  return  as  elif
in  try  assert  else  is  while  async  except  lambda
with  await  finally  nonlocal  yield  break  for  not
```

例如：使用关键字 def 作为变量名，此时编译即会报错。

```
1 #coding: utf-8
2
3 name = '小沉'
4
5 def = 'ok'
6
7 if __name__ == '__main__':
8     print(def)
```

### 3.1.3 变量

何为变量？在 Python 中，变量其实严格意义上称作为“名字”。也可以理解为标签。

当我们把一个值赋给一个变量的时候（如：把“学习 Python 可以为我们减少很多工作量”赋值给 python），python 就称之为变量。现在大部分的编程语言中都把这种行为叫做“把值存储在变量中”。意思就是说在计算机的内存中开辟一块空间用来存储这个字符串“学习 Python 可以为我们较少很多工作量”，你不需要知道他存放到哪里。只需要告诉 Python 这个字符串的名字。通过这个名字就能找到字符串序列了。

在 Python 中为一个变量赋值的同时就声明了该变量，该变量的数据类型就是赋值数据所属的类型，该变量还可以接收其他类型的数据。变量的命名必须是一个有效的标识符。例如：

PRECHIN  
普中  
www.prechin.net

```
Shell < >>> greet="Hello World"
>>> greet
'Hello World'
>>> score=95.5
>>> score
95.5
>>> x=10
>>> x
10
>>> x=30
>>> x
30
>>> x=True
>>> x
True
>>>
```

### 3.1.4 语句

Python 代码是由关键字、标识符、表达式和语句等构成的，语句是代码的重要组成部分。在 Python 中，一行代码表示一条语句，在一般情况下语句结束时不加分号，当然也可以在语句结束时加分号，但不符合 Python 编程规范。例如：

```
Shell < >>> greet="Hello World"
>>> score=95.5;
>>> a=b=c=10
>>> b
10
>>> score
95.5
>>>
```

### 3.1.5 代码注释

使用#（井号）时，#位于注释行的开头，#后面有一个空格（也可省略），接着是注释的内容，这是单行注释。多行注释可使用英文字符三个单引号或三个双引号，并在其中添加注释内容。例如：

```
#这是一个整数变量
score=95
print(score) #打印score变量
```

方式1  
多行注释  
多行注释  
...

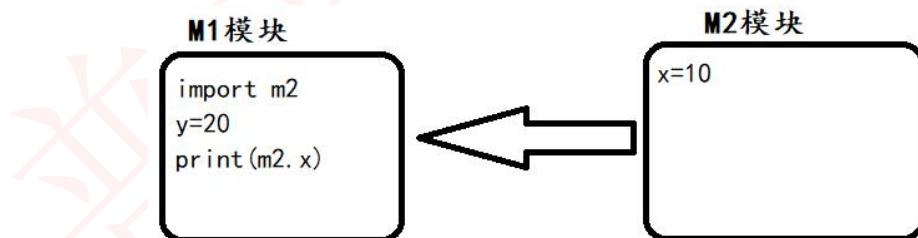
方式2  
多行注释  
多行注释  
...

### 3.1.6 模块

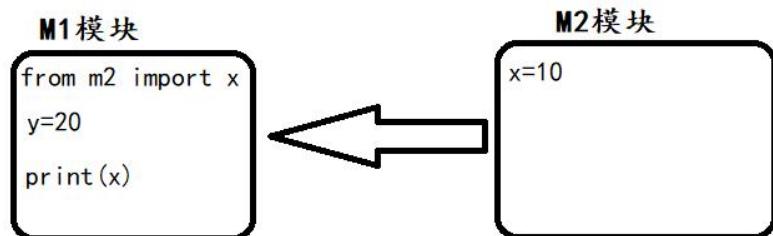
模块就是一个包含了 Python 定义和声明的 “.py” 文件。从模块中导入语句有 3 种形式，如下：

```
import <模块名>
from <模块名> import <代码元素>
from <模块名> import <代码元素> as <代码元素别名>
```

①import <模块名>：通过这种方式会导入 M2 模块的所有代码元素，在访问时需要加前缀“m2.”。

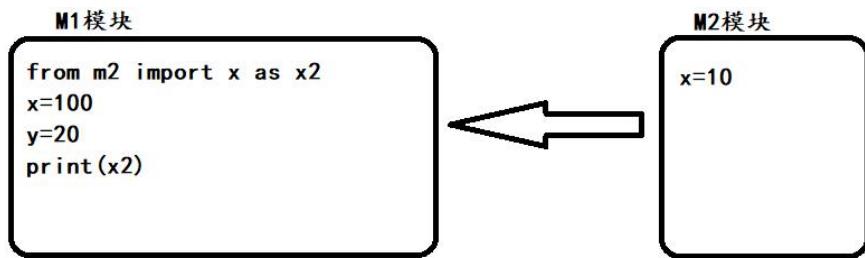


②from <模块名> import <代码元素>：通过这种方式会导入 M2 中的 x 变量，在访问时不需要加前缀“m2.”。



③from <模块名> import <代码元素> as <代码元素别名>：与②类似，在

当前 m1 模块的代码元素（x 变量）与要导入的 m2 模块的代码元素（x 变量）名称有冲突时，可以给要导入的代码元素起别名（x2）。



## 3.2 数字类型的数据

### 3.2.1 数据类型

在编程中，数据类型是一个重要的概念。变量可以存储不同类型的数据，并且不同类型可以执行不同的操作。Python 中有 6 种主要的内置数据类型：数字、字符串、列表、元组、集合和字典。Python 中有 4 种数字类型：整数类型、浮点类型、复数类型和布尔类型。

### 3.2.2 整数类型

类似 -2、-1、0、1、2 这样的数据称为整型数据（简称整数）。在 Python 中可以使用 4 种进制表示整型，分别为二进制（以 0B 或 0b 开头）、八进制（以 0o 或 00 开头）、十进制（默认表示方式）和十六进制（以 0x 或 0X 开头）。

例如：整数 10 各种表示方法。

```
Shell > 
>>> 10
10
>>> 0x0A
10
>>> 0b1010
10
>>> type(0x0a)
<class 'int'>
>>>
```

可使用 type 方法返回数据类型。Python 中整数类型为 int 类。

### 3.2.3 浮点类型

类似 1.1、0.5、-1.4 这样的数据被称为浮点型数据。浮点型数据用于保存带有小数点的数值，Python 的浮点数一般以十进制形式表示，对于较大或较小的浮点数可使用科学计数法表示。Python 中浮点类型为 float 类。

```
Shell >>> 3.14
3.14
>>> type(3.14)
<class 'float'>
>>> 3.36e2
336.0
>>> 1.56e-2
0.0156
>>>
```

### 3.2.4 复数类型

复数在数学中被表示为： $a+bi$ ，其中 a 为实部，b 为虚部，i 为虚数单位。  
在 Python 中虚数单位为 j 或者 J。

在 Python 中有两种方创建复数的方式：一种是按照复数的一般形式直接创建；另一种是通过内置函数 complex() 创建。例如：

```
Shell >>> c=3+2j
>>> c
(3+2j)
>>> type(c)
<class 'complex'>
>>> a=complex(3,2)
>>> a
(3+2j)
>>>
```

### 3.2.5 布尔类型

Python 中的布尔类型为 `bool` 类，它只有两个值：`True` 和 `False`。实际上布尔类型是一种特殊的整型，其中 `True` 对应 1，`False` 对应 0。Python 中任何对象都可以转换为布尔类型，若要转换，符合以下条件的数据都会被转换为 `False`。

- `None`
- 任何为 0 的数字类型，如 0、0.0、0j。
- 任何空序列，如 ""、()、[]。
- 任何空字典，如 {}。
- 用户定义的类实例，如类中定义了 `__bool__()` 或者 `__len__()`。

除以上对象外，其他的对象都会被转换为 `True`。可以使用 `bool()` 函数检测对象的布尔值。

```
Shell > 
>>> bool(None)
False
>>> bool(0)
False
>>> bool([])
False
>>> bool(2)
True
>>> bool("")
False
>>> bool(" ")
True
```

### 3.2.6 数字类型相互转换

#### 3.2.6.1 隐式类型的转换

数字之间可以进行数学计算，在进行数学计算时若数字类型不同，则会发生隐式类型的转换。例如：

```
Shell >>> a=1+True
>>> a
2
>>> a=1.0+1
>>> a
2.0
>>> a=1.0+False
>>> a
1.0
```

### 3.2.6.2 显式类型的转换

表达式  $1.0+1$  中的整数 1 被隐式转换为浮点数 1.0，但在很多情况下希望浮点数被转换为整数 1，此时该怎么办呢？这种情况下就需要使用转换函数进行显式转换了。除复数外，三种数字类型如整数、浮点数和布尔都有自己的转换函数，分别是 `int()`、`float()` 和 `bool()` 函数，`bool()` 函数前面已介绍。

```
>>> a=int(1.0)+1
>>> a
2
>>> int(False)
0
>>> int(True)
1
>>> int(0.3)
0
>>> float(3)
3.0
>>> float(True)
1.0
```

## 3.3 运算符

### 3.3.1 算术运算符

操作符	描述	示例
+	加：两个对象相加	20+10=30
-	减：一个数减去另外一个数	20-10=10
*	乘：两数相乘或返回一个被重复若干次的字符串	20*10=200
/	除：两个数相除	20/10=2
%	取模：返回两个数相除的余数	20%10=0
**	幂：返回某一个数的若干次幂	2**1=2, 2**2=4
//	取整：返回两数相除后所得商的整数部分	7//3=2, 7.0//2.0=3.0

### 3.3.2 比较运算符

运算符	名称	例子	说明
==	等于	a == b	a等于b时返回True, 否则返回False
!=	不等于	a != b	与==相反
>	大于	a > b	a大于b时返回True, 否则返回False
<	小于	a < b	a小于b时返回True, 否则返回False
>=	大于等于	a >= b	a大于等于b时返回True, 否则返回False
<=	小于等于	a <= b	a小于等于b时返回True, 否则返回False

### 3.3.3 逻辑运算符

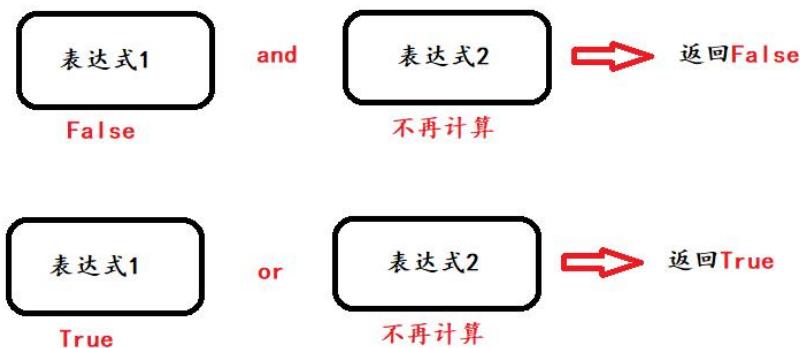
逻辑运算符	含义	基本格式	说明
and	逻辑与运算，等价于数学中的“且”	a and b	当 a 和 b 两个表达式都为真时，a and b 的结果才为真，否则为假。
or	逻辑或运算，等价于数学中的“或”	a or b	当 a 和 b 两个表达式都为假时，a or b 的结果才是假，否则为真。
not	逻辑非运算，等价于数学中的“非”	not a	如果 a 为真，那么 not a 的结果为假；如果 a 为假，那么 not a 的结果为真。相当于对 a 取反。

逻辑运算符一般和关系运算符结合使用，例如：

14>6 and 45.6 > 90

14>6 结果为 True，成立，45.6>90 结果为 False，不成立，所以整个表达

式的结果为 False，也即不成立。逻辑运算中 and 和 or 有如下特点：



例如：

```
>>> a=1
>>> b=0
>>> def f1():
    print("进入f1函数")
    return True

>>> (a>b) or f1()
True
>>> (a<b) or f1()
进入f1函数
True
>>> (a<b) and f1()
False
>>> (a>b) and f1()
进入f1函数
True
>>>
```

### 3.3.4 位运算符

运算符	名称	例子	说明
~	位反	$\sim x$	将x的值按位取反
&	位与	$x \& y$	将x与y按位进行位与运算
	位或	$x   y$	将x与y按位进行位或运算
^	位异或	$x ^ y$	将x与y按位进行位异或运算
>>	右移	$x >> a$	将x右移a位，高位采用符号位补位
<<	左移	$x << a$	将x左移a位，低位用0补位

例如：

```
>>> a=0b10110010
>>> b=0b01011110
>>> a|b
254
>>> a&b
18
>>> a^b
236
>>> ~a
-179
>>> a>>2
44
>>> a<<2
712
>>> c=-20
>>> ~c
19
```

在按位取反运算中涉及到原码、补码、反码运算，比较麻烦。这里提供一个好用的公式： $\sim a = (a+1) * (-1)$ ，如果 a 为十进制 178，则 $\sim a$  为十进制数-179；如果 a 为十进制数-20，则 $\sim a$  为十进制数 19。

### 3.3.5 赋值运算符

运算符	说明	用法举例	等价形式
=	最基本的赋值运算	x = y	x = y
+=	加赋值	x += y	x = x + y
-=	减赋值	x -= y	x = x - y
*=	乘赋值	x *= y	x = x * y
/=	除赋值	x /= y	x = x / y
%=	取余数赋值	x %= y	x = x % y
**=	幂赋值	x **= y	x = x ** y
//=	取整数赋值	x // y	x = x // y
&=	按位与赋值	x &= y	x = x & y
=	按位或赋值	x  = y	x = x   y
^=	按位异或赋值	x ^= y	x = x ^ y
<<=	左移赋值	x <<= y	x = x << y, 这里的 y 指的是左移的位数
>>=	右移赋值	x >>= y	x = x >> y, 这里的 y 指的是右移的位数

例如：

```
>>> x=100
>>> x-=10
>>> x
90
>>> y=2
>>> y*=x-10
>>> y
160
```

### 3.4 程序流程控制

程序中的语句默认是自上而下顺序执行。程序流程控制指的是在程序执行时，通过一些特定的指令更改程序中语句的执行顺序，使程序产生跳跃、回溯等现象。本节将对分支语句、循环语句和跳转语句介绍。

#### 3.4.1 分支语句

if 语句可使程序产生分支，if 语句可分为单分支 if 语句、双分支 if-else 语句和多分支 if-elif-else 语句，而且 if 语句还可以嵌套。

### 3.4.1.1 if 语句

单分支结构是最简单的一种选择结构，语法结构如下：

1 | **if** 条件表达式:  
2 |      语句块

当条件表达式成立的时候，执行语句块，不成立则不执行。例如：

```
1 age=5
2 if age>=3:
3     print("可以上幼儿园了")
```

### 3.4.1.2 if-else 语句

二分支结构在单分支结构的基础上增加了 else 语句，当 if 条件不成立时，执行 else 语句，语法结构如下：

1 | **if** 条件表达式:  
2 |      语句块1  
3 | **else:**  
4 |      语句块2

二分支结构是二选一的结构，语句块 1 和语句块 2 有且只有一个一定会被执行到。例如：

```
1 u_name=input("请输入用户名: ")
2 pwd=input("请输入密码: ")
3 if u_name=="admin" and pwd=="123":
4     print("登入成功! 即将进入主界面...")
5 else:
6     print("您输入的用户名或密码错误, 请重新输入...")
```

### 3.4.1.3 if-elif-else 语句

多分支结构是二分支结构的扩展，即多选一的情况，其中 else 语句是可选的，当 else 存在时，有且只有一个分支会被执行到。其语法结构如下：

```
1 | if 条件表达式1:  
2 |   语句块1  
3 | elif 条件表达式2:  
4 |   语句块2  
5 | ...  
6 | elif 条件表达式n:  
7 |   语句块n  
8 | else:  
9 |   语句块n+1
```

如果条件表达式 1 结果为 True，则执行语句块 1；如果条件表达式 2 结果为 True，则执行语句块 2；依次类推。如果 else 前面的条件表达式均为 False，则执行语句代码块 n+1。例如：

```
1 score=int(input("请输入您的会员积分: "))  
2 if score==0:  
3     print("注册会员")  
4 elif 0<score<=200:  
5     print("铜牌会员")  
6 elif 200<score<=1000:  
7     print("银牌会员")  
8 elif 1000<score<=300:  
9     print("金牌会员")  
10 else:  
11     print("钻石会员")
```

#### 3.4.1.4 嵌套分支结构

在分支语句中如果要做进一步的条件判断，就会用到嵌套的分支结构。嵌套也可以有多层，通过缩进来表示其包含关系。代表性语法结构如下：

```
1 | if 条件表达式1:  
2 | ...  
3 |   if 条件表达式2:  
4 |     语句块1  
5 |   else:  
6 |     语句块2  
7 | else:  
8 |   语句块3
```

例如：在做身份证号的性别判断时，如果要先对输入的身份证号合法性做基本检查，判断其位数是不是正确，则需要嵌套分支：

```
1 id_code = input('请输入身份证号码：')
2 if len(id_code) == 18:
3     number = int(id_code[-2])
4     if number%2 == 0:
5         print("女性")
6     else:
7         print("男性")
8 else:
9     print("输入不合法")
```

### 3.4.2 循环语句

简单理解，循环就是反复的去做某一件事情。生活中的例子：比如我们听歌的时候，在歌曲的页面就会出现单曲循环、列表循环播放等。

Python 中循环语句的逻辑：执行一个语句/一段代码块多次。Python 的循环语句有 while 和 for 循环。

#### 3.4.2.1 while 循环

while 语句用于循环执行程序，即在某条件下，循环执行某段程序，以处理需要重复处理的相同任务。语法如下：

```
1 | while 判断条件:
2 |   语句
```

①判断条件可以是表达式，也可以是字符。当判断条件成立时，则进入循环体执行语句，否则退出循环，不再执行语句。

例如：

```
1 # 输出1-100的所有数的和
2 count = 0
3 num = 0
4 while count < 100:
5     count = count + 1
6     num = num + count
7 print("1-100的所有数的总和为：和", num)
```

②当判断条件永远为 True 是，则 while 是一个无限循环，若要在无限循环中退出，可使用键盘 Ctrl+C。例如：

```
1 # 无限循环
2 while True:
3     num=int(input("请输入一个数字: "))
4     print("您输入的数字是: ", num)
```

③while 循环可使用 else 语句: while ... else。在条件语句为 false 时执行 else 的语句块。例如:

```
1 count = 0
2 while count < 5:
3     print(count, " 小于 5")
4     count = count + 1
5 else:
6     print(count, " 大于或等于 5")
```

### 3.4.2.2 for 循环

python 中的 for 循环可以针对数据类型元组、字符串、列表、字典进行遍历，也可以针对某一个区间的数据范围进行遍历。for 循环的使用语法格式如下：

```
1 for 临时变量 in 可迭代对象:
2     执行语句1
3     执行语句2
4     ...
5     ...
```

每执行一次循环，临时变量都会被赋值为可迭代对象的当前元素，提供给执行语句使用。

for 循环主要通过遍历对象来控制循环次数，对象的数据遍历完之后，循环就结束了。例如：

```
1 languages = ["C", "C++", "Python", "Java", "C#"]
2 for x in languages:
3     print("当前语言是: ", x)
```

如果需要遍历数字序列，可以使用内置 range() 函数，它会生成数列。例如：

```
1 for i in range(6):
2     print("当前的数列为: ", i)
3
4 for i in range(1,3): # 范围从1开始，不包含最后一个数字
5     print(i)
```

for 循环也可支持嵌套循环，最常见的用法就是打印 9\*9 乘法表，例如：

```
1 for i in range(1, 10): #遍历9次，打印9行
2     for j in range(1, 10): #遍历9次，打印9列的数据
3         if j <= i: # 当列数<=行数的时候，就可以打印乘法公式
4             print(f'{i}*{j}={i*j}'.format(i, j), end='\t')
```

### 3.4.3 跳转语句

循环语句一般会执行完所有的情况后自然结束，但是有些情况下，需要停止当前正在执行的循环，即跳出循环。Python 支持使用 break 语句跳出整个循环，使用 continue 语句跳出本次循环。

#### 3.4.3.1 break 语句

跳出 for 和 while 的循环体。如果从 for 或 while 循环中终止，任何对应的循环 else 块将不执行。例如：

```
1 for s in 'Hello':
2     if s == 'l':
3         break
4     print('当前字母为：', s)

>>> for n in range(2,10):
    for x in range(2,n):
        if n%x==0:
            print(n,'等于',x,'*',n//x)
            break
    else:
        print(n,'是质数')

2 是质数
3 是质数
4 等于 2 * 2
5 是质数
6 等于 2 * 3
7 是质数
8 等于 2 * 4
9 等于 3 * 3
```

break 语句一般会结合 if 语句进行搭配使用，表示在某种条件之下，跳出循环。

### 3.4.3.2 continue 语句

continue 语句用于跳出当前循环，继续执行下一次循环。当执行到 continue 语句时，程序会忽略当前循环中剩余的代码，重新开始执行下一次循环。

例如：从列表中找出所有的正数。

```
1 for x in [0,-2,5,7,-10]:  
2     if x<=0:  
3         continue  
4     print(x)
```

Break 和 continue 语句只能用于循环中，不能单独使用。

### 3.4.4 pass 语句

pass 是 python 中的保留字，pass 语句又称为空语句或占位语句。程序运行到 pass 语句时，什么也不做，直接跳过，运行后面的语句。pass 的作用就是为了保持程序结构的完整性。有时候程序需要占一个位置，或者放一条语句，但又不希望这条语句做任何事情，此时就可以通过 pass 语句来实现。使用 pass 语句比使用注释更加优雅。例如：

```
1 if x<=0:  
2     pass
```

## 3.5 容器类型数据

### 3.5.1 序列

所谓序列，指的是一块可存放多个值的连续内存空间，这些值按一定顺序排列，可通过每个值所在位置的编号（称为索引）访问它们。

在 Python 中，序列类型包括字符串、列表、元组、集合和字典，这些序列支持以下几种通用的操作，但比较特殊的是，集合和字典不支持索引、切片、相加和相乘操作。

### 3.5.1.1 序列索引操作

序列中，每个元素都有属于自己的编号（索引）。从起始元素开始，索引值从 0 开始递增。如下：



除此之外，Python 还支持索引值是负数，此类索引是从右向左计数，换句话说，从最后一个元素开始计数，从索引值 -1 开始，如下：



我们是通过下标运算符访问序列中的元素的，下标运算符是跟在容器数据后的一对中括号（[]），中括号带有参数，这个参数即为索引值。例如：

```
>>>
>>> a="Hello"
>>> a[0]
'H'
>>> a[-5]
'H'
>>> a[5]
Traceback (most recent call last):
  File "<pyshell>", line 1, in <module>
IndexError: string index out of range
>>> max(a)
'o'
>>> min(a)
'H'
>>> len(a)
5
```

注意：若索引超出范围，则发生错误。`max` 函数用于返回最后一个元素，`min` 函数用于返回第一个元素，`len` 函数用于获取序列的长度。

### 3.5.1.2 加和乘操作

Python 中，支持两种类型相同的序列使用“+”运算符做相加操作，它会将两个序列进行连接，但不会去除重复的元素。这里所说的“类型相同”，指的是“+”运算符的两侧序列要么都是列表类型，要么都是元组类型，要么都是字符串。例如：

```
>>> a='hello'  
>>> print(a+'world')  
helloworld  
>>> print(a+' world')  
hello world  
>>> print(a+' world'+'你好')  
hello world你好
```

Python 中，使用数字 n 乘以一个序列会生成新的序列，其内容为原来序列被重复 n 次的结果。例如：

```
>>> a='hello'  
>>> print(a*2)  
hellohello
```

### 3.5.1.3 切片操作

切片操作是访问序列中元素的另一种方法，它可以访问一定范围内的元素，通过切片操作，可以生成一个新的序列。

序列实现切片操作的语法格式如下：

```
sname[start : end : step]
```

其中，各个参数的含义分别是：

sname：表示序列的名称；

start：表示切片的开始索引位置（包括该位置），此参数也可以不指定，会默认为 0，也就是从序列的开头进行切片；

end：表示切片的结束索引位置（不包括该位置），如果不指定，则默认为序列的长度；

step：表示在切片过程中，隔几个存储位置（包含当前位置）取一次元素，

也就是说，如果 step 的值大于 1，则在进行切片去序列元素时，会“跳跃式”的取元素。如果省略设置 step 的值，则最后一个冒号就可以省略。

例如：

```
>>> str='Hello world'  
>>> print(str[:2])  
He  
>>> print(str[::2])  
Hlowrd  
>>> print(str[:])  
Hello world
```

### 3.5.1.4 成员测试

成员测试运算符有两个：in 和 not in。in 用于测试是否包含某一个元素，not in 用于测试是否不包含某一个元素。例如：

```
>>> str='Hello'  
>>> 'e' in str  
True  
>>> 'E' not in str  
True
```

## 3.5.2 列表

列表是 Python 中最灵活的有序序列，它可以存储任意类型的元素，开发人员可以对列表中的元素进行添加、删除、修改等操作。

### 3.5.2.1 创建列表

创建列表的方式非常简单，既可以使用中括号“[]”创建，也可以使用内置的 list() 函数快速创建。

#### ① 使用中括号“[]”创建列表

使用中括号[]创建列表时，只需要在中括号[]中使用逗号分隔每个元素即可。例如：

```
1 list_one=[] #空列表
2 list_two=['p','y','t','h','o','n'] #列表中元素类型均为字符串类型
3 list_three=[1,'a','&',2.3] #列表中元素类型不同
```

## ②使用 list() 函数创建列表

使用 list() 函数同样可以创建列表，需要注意的是该函数接收的参数必须是一个可迭代类型的数据。例如：

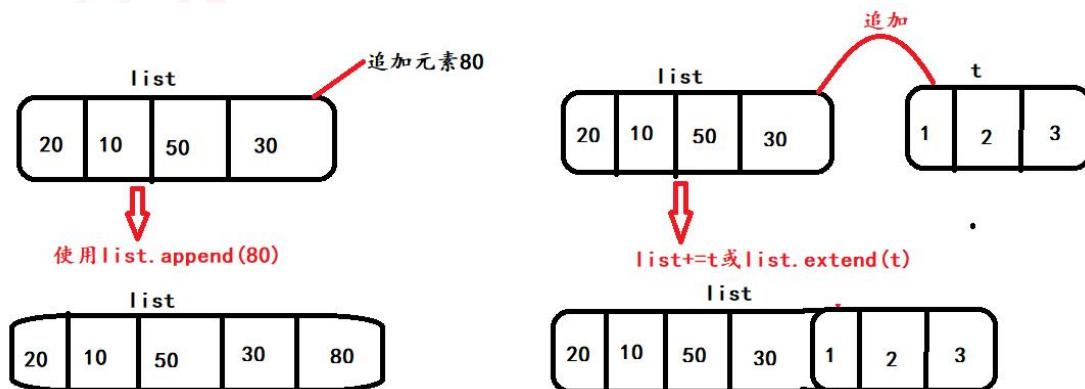
```
>>> list_one=list(1)
Traceback (most recent call last):
  File "<pyshell>", line 1, in <module>
TypeError: 'int' object is not iterable
>>> list_two=list('python')
>>> list_three=list([1,'python'])
>>> type(list_three)
<class 'list'>
```

可直接使用 for 循环的对象成为可迭代对象。比如字符串、列表、元组、字典和集合。如何判断一个对象是否为可迭代对象呢？可使用 isinstance() 函数进行判断，例如：

```
>>> from collections import Iterable
>>> print(isinstance([],Iterable))
True
>>>
```

### 3.5.2.2 追加元素

在列表中追加单个元素时，可以使用列表的 append(x) 方法。在列表中追加多个元素时，可以使用加 (+) 运算符或列表的 extend(t) 方法。例如：



### 3.5.2.3 插入元素

想要向列表中插入元素时，可以使用列表的 `list.insert(i, x)` 方法。其中 `i` 表示插入到列表的位置，`x` 为插入元素。例如：

```
>>> names=['张三', '李四', '王五']
>>> names.insert(2, '小强')
>>> print(names)
['张三', '李四', '小强', '王五']
>>>
```

### 3.5.2.4 替换元素

想要替换列表中的元素时，就是通过索引获取元素并对该元素重新赋值。例如：

```
>>> names=['张三', '李四', '王五']
>>> names[1]='小强'
>>> print(names)
['张三', '小强', '王五']
```

### 3.5.2.5 删除元素

删除列表元素的常用方式有 `del` 语句、`remove()` 方法和 `pop()` 方法，具体介绍如下：

#### ①del 语句

`del` 语句用于删除列表中指定位置的元素。如：

```
>>> names=['张三', '李四', '王五']
>>> print(names)
['张三', '李四', '王五']
>>> del names[0]
>>> print(names)
['李四', '王五']
```

#### ②remove()方法

`remove()` 方法用于移除列表中的某个元素，若列表中有多个匹配的元素，只

会移除匹配到的第一个元素。例如：

```
>>> chars=['h','e','l','l','o']
>>> chars.remove('e')
>>> print(chars)
['h', 'l', 'l', 'o']
```

### ③pop()方法

pop()方法用于移除列表中的某个元素，如果不指定具体元素，那么移除列表中的最后一个元素。例如：

```
>>> numbers=[1,2,3,4,5]
>>> print(numbers.pop())
5
>>> print(numbers.pop(2))
3
>>> print(numbers)
[1, 2, 4]
```

## 3.5.3 元组

元组（tuple）是一种不可变序列类型。元组中的元素是不允许修改的，除非在元组中包含可变类型的数据。

### 3.5.3.1 创建元组

元组的创建方式与列表的创建方式相似，可以通过圆括号“()”或内置的tuple()函数快速创建。

#### ①使用圆括号“()”创建元组

使用圆括号“()”创建元组，并将元组中的元素用逗号进行分隔。例如：

```
1 tuple_one=() #空元组
2 tuple_two=('t','u','p','l','e') #元组中元素类型均为字符串类型
3 tuple_three=(0.3,1,'python','&') #元组中元素类型不同
```

注意：当使用圆括号()创建元组时，如果元组中只包含一个元素，则需要在该元素的后面添加逗号，保证Python解释能够识别其为元组类型。

#### ②使用tuple()函数创建元组

使用tuple()函数创建元组时，如果不传入任何数据，就会创建一个空元组；

如果要创建包含元素的元组，就必须要传入可迭代类型的数据。例如：

```
>>> tuple_null=tuple()  
>>> print(tuple_null)  
()  
>>> tuple_str=tuple('abc')  
>>> print(tuple_str)  
('a', 'b', 'c')  
>>> tuple_list=tuple([1,2,3])  
>>> print(tuple_list)  
(1, 2, 3)
```

### 3.5.3.2 元组拆包

元组的拆包就是将元组内部的每个元素按照位置，对应的赋值给不同变量。可以用于：变量赋值，变量值交换，函数参数赋值，获取元组中特定位置的元素值等。此外，Python 函数 return 多个对象，默认就是以 tuple 形式返回。例如：

```
>>> val=(10, 20)  
>>> a,b=val  
>>> print(a)  
10  
>>> print(b)  
20
```

不仅是元组，在 python 中任何序列或可迭代对象（如：列表、元组、字符串、文件对象、迭代器和生成器等），皆可通过类似这样的简单赋值语句拆包给多个变量。唯一的要求就是变量必须跟序列元素的数量一致，否则会抛出 ValueError 的异常。

### 3.5.4 集合

集合集合（set）是一种可迭代的、无序的、不能包含重复元素的容器类型的数据。

### 3.5.4.1 创建集合

Python 中的集合分为可变集合与不可变集合，可变集合由 `set()` 函数创建，集合中的元素可以动态的增加或删除；不可变集合由 `frozenset()` 函数创建，集合中的元素不可改变。格式如下：

```
1 set([iterable])
2 frozenset([iterable])
3
```

上述参数中 `iterable` 是一个可迭代对象，若没有指定可迭代对象，则返回一个空的集合。

#### ①可变集合的创建

使用 `set()` 函数创建可变集合。此外，还可以直接使用花括号 {} 创建可变集合，花括号中的多个元素以逗号分隔。例如：

```
>>> set_one=set([1,2,1,3])
>>> set_two=set((1,2,3))
>>> type(set_one)
<class 'set'>
>>> set_one
{1, 2, 3}
>>> set_three={1,21,3,44,6,21}
>>> set_three
{1, 3, 6, 44, 21}
```

#### ②不可变集合的创建

使用 `frozenset()` 函数创建的集合是不可变集合。例如：

```
>>> frozenset_one=frozenset((1,2,3,4))
>>> frozenset_two=frozenset([1,2,3,4])
>>> type(frozenset_one)
<class 'frozenset'>
>>> frozenset_two
frozenset({1, 2, 3, 4})
```

### 3.5.4.2 修改集合

Python 中可变集合支持添加、删除和清空元素这些基本操作。

#### ①添加元素

add() 方法只能添加一个元素，而 update() 方法可以添加多个元素，例如：

```
>>> set_one=set()  
>>> set_one  
set()  
>>> set_one.add('py')  
>>> set_one  
{'py'}  
>>> set_one.update('thon')  
>>> set_one  
{'t', 'o', 'n', 'h', 'py'}
```

#### ②删除元素

remove() 方法：用于删除可变集合中的指定元素。注意，指定的元素不在集合中，则会出现 KeyError 错误。

discard() 方法：也可以删除指定的元素，但若指定的元素不存在，该方法不执行任何操作。

pop() 方法：用于删除可变集合中的随机元素。

例如：

```
>>> set_one={'张三','李四','王五','小强'}  
>>> set_one.remove('张三')  
>>> print(set_one)  
{'李四', '王五', '小强'}  
>>> set_one.remove('张三')  
Traceback (most recent call last):  
  File "<pyshell>", line 1, in <module>  
KeyError: '张三'  
>>> set_one.discard('张三')  
>>> set_one.discard('李四')  
>>> set_one  
{'王五', '小强'}  
>>> set_one.pop()  
'王五'  
>>> set_one  
{'小强'}
```

### ③清空元素

如果需要清空可变集合中的元素，可以使用 clear() 方法。例如：

```
>>> set_one={'张三','李四','王五','小强'}  
>>> set_one.clear()  
>>> set_one  
set()
```

## 3.5.5 字典

字典（dict）是可迭代的、通过键（key）来访问元素的可变的容器类型的数据。与列表一样是一个可变序列，以键值对的方式存储数据，字典是一个无序的序列。键视图不能包含重复的元素，值视图可重复。在键视图中，键和值是成对出现的。

### 3.5.5.1 创建字典

Python 可以使用花括号 {} 包含多个键值对（Key-Value）的形式创建字典，也可以使用内置的 dict() 函数创建字典。

#### ①使用花括号 {} 创建字典

使用花括号{}创建字典时，字典中的键（Key）和值（Value）使用冒号连接，每个键值对之间使用逗号分隔。格式如下：

```
{键 1:值 1, 键 2:值 2...}
```

例如：

```
>>> scores={'张三':100,'李四':98,'王五':45}  
>>> scores  
{'张三': 100, '李四': 98, '王五': 45}  
>>> scores_null={}
>>> type(scores)
<class 'dict'>
>>> scores_null
{}
>>> type(scores_null)
<class 'dict'>
```

## ②使用内置的 dict() 函数创建字典

使用 dict() 函数创建字典时，键和值使用“=”进行连接，其语法格式如下：

```
dict(键 1=值 1, 键 2=值 2...)
```

例如：

```
>>> dict_one=dict(name='张三',age=20)
>>> dict_one
{'name': '张三', 'age': 20}
>>> dict_two=dict({100:'张三',101:'李四',102:'王五'})
>>> dict_two
{100: '张三', 101: '李四', 102: '王五'}
>>> dict_three=dict(((100,'张三'),(101,'李四'),(102,'王五')))
>>> dict_three
{100: '张三', 101: '李四', 102: '王五'}
>>> dict_four=dict([(100,'张三'),(101,'李四'),(102,'王五')])
>>> dict_four
{100: '张三', 101: '李四', 102: '王五'}
>>> dict_five=dict([(100,'张三'),(100,'李四'),(102,'王五')])
>>> dict_five
{100: '李四', 102: '王五'}
```

### 3.5.5.2 修改字典

字典可以被修改，但都是针对键和值同时操作的，对字典的修改包括添加、替换和删除。

字典支持使用 `update()` 方法或通过指定的键添加元素或修改元素。

#### ①添加和修改字典元素

```
>>> add_dict={'stu1':'张三'}  
>>> add_dict.update(stu2='李四')  
>>> add_dict  
{'stu1': '张三', 'stu2': '李四'}  
>>> add_dict['stu3']='王五'  
>>> add_dict  
{'stu1': '张三', 'stu2': '李四', 'stu3': '王五'}  
>>> add_dict['stu1']='小强'  
>>> add_dict  
{'stu1': '小强', 'stu2': '李四', 'stu3': '王五'}
```

#### ②删除字典元素

Python 支持通过 `pop()`、`popitem()` 和 `clear()` 方法删除字典中的元素。

`pop()` 方法：可根据指定键值删除字典中的指定元素，若删除成功，该方法返回目标元素的值。

`popitem()` 方法：可以随机删除字典中的元素。实际上 `popitem()` 之所以能删除随机元素，是因为字典元素本身是无序的，没有所谓的第一项、最后一项。若删除成功，`popitem()` 方法返回目标元素。

`clear()` 方法：用于清空字典中的元素。

例如：

```
>>> dist_one={'001':'张三','002':'李四','003':'王五','004':'小强'}  
>>> dist_one.pop('001')  
'张三'  
>>> dist_one  
{'002': '李四', '003': '王五', '004': '小强'}  
>>> dist_one.popitem()  
('004', '小强')  
>>> dist_one  
{'002': '李四', '003': '王五'}  
>>> dist_one.clear()  
>>> dist_one  
{}
```

### 3.5.5.3 访问字典

items()方法：使用该方法可以查看字典的所有元素。items()方法会返回一个dict\_items对象，该对象支持迭代操作，通过for循环遍历dict\_items对象中的数据并以(key, value)的形式显示。

keys()方法：使用该方法可以查看字典中的所有键。keys方法会返回一个dict\_keys对象，该对象也支持迭代操作，通过for循环遍历输出字典中所有的键。

values()方法返回字典中所有的值。values()方法会返回一个dict\_values对象，该对象支持迭代操作，使用for循环遍历输出字典中所有的值。

例如：

```
>>> info={'001':'张三','002':'李四','003':'王五'}
>>> print(info.items())
dict_items([('001', '张三'), ('002', '李四'), ('003', '王五')])
>>> for i in info.items():
    print(i)

('001', '张三')
('002', '李四')
('003', '王五')

>>> print(info.keys())
dict_keys(['001', '002', '003'])
>>> for i in info.keys():
    print(i)

001
002
003

>>> print(info.values())
dict_values(['张三', '李四', '王五'])
>>> for i in info.values():
    print(i)

张三
李四
王五
```

## 3.6 字符串

字符串是一种用来表示文本的数据类型，它是由符号或者数值组成的一个连续序列，Python 中的字符串是不可变的，字符串一旦创建便不可修改。

### 3.6.1 字符串的表示方式

#### 3.6.1.1 普通字符串

普通字符串指用单引号('')或双引号("")括起来的字符串。例如：'Hello' 或"Hello"。

如果想要在字符串中包含一些特殊的字符，例如换行符、制表符等，在普通字符串中就需要转义，前面要加上反斜杠(\)，这叫作字符转义。常用的转义符

如下：

字符表示	Unicode编码	说明
\t	\u0009	水平制表符
\n	\u000a	换行
\r	\u000d	回车
\"	\u0022	双引号
'	\u0027	单引号
\	\u005c	反斜线

例如：

```
>>> 'Hello'
'Hello'
>>> "Hello"
'Hello'
>>> s='\u0048\u0065\u006c\u006c\u006f'
>>> s
'Hello'
>>> "Hello'world"
"Hello'world"
>>> 'Hello"world'
'Hello"world'
```

### 3.6.1.2 原始字符串

在 Python 中，以字母 r 或者 R 作为前缀的字符串，例如 r' ... ' 和 R' ... '，被称为原始字符串。与常规字符串不同，原始字符串中的反斜线 (\) 是一个普通字符，不具有转义功能。

原始字符串通常用于处理字符串中存在多个反斜线的情况，例如正则表达式和 Windows 目录路径。

例如：

PRECHIN  
普中  
www.prechin.net

```
>>> s = 'hello\world\nPython\t3'
>>> print(s)
hello    world
Python    3
>>> s=r'hello\world\nPython\t3'
>>> print(s)
hello\world\nPython\t3
```

### 3.6.1.3 长字符串

如果要使用字符串表示一篇文章，其中包含了换行、缩进等排版字符，则可以使用长字符串表示。对于长字符串，要使用三个单引号(''')或三个双引号("}")括起来。格式如下：

1	'''XXXXXX
2	XXXXXXX
3	XXXXXXX'''
4	
5	"""\XXXXXX
6	XXXXXXX
7	XXXXXXX"""

例如：

```
>>> s='''<<静夜思>>--李白
床前明月光，疑是地上霜。
举头望明月，低头思故乡。
...
>>> print(s)

<<静夜思>>--李白
床前明月光，疑是地上霜。
举头望明月，低头思故乡。
```

### 3.6.2 字符串与数字相互转换

#### 3.6.2.1 将字符串转换为数字

将字符串转换为数字，可以使用 `int()` 和 `float()` 实现，如果成功则返回数字，否则引发异常。

```
>>> int("10")
10
>>> int("10.2")
Traceback (most recent call last):
  File "<pyshell>", line 1, in <module>
ValueError: invalid literal for int() with base 10: '10.2'
>>> float("10.2")
10.2
>>> int("ab")
Traceback (most recent call last):
  File "<pyshell>", line 1, in <module>
ValueError: invalid literal for int() with base 10: 'ab'
>>> int("ab",16)
171
```

### 3.6.2.2 将数字转换为字符串

将数字转换为字符串，可以使用 str() 函数，str() 函数可以将很多类型的数据都转换为字符串。

```
>>> str(12)
'12'
>>> str(12.3)
'12.3'
>>> str(True)
'True'
```

### 3.6.3 格式化字符串

Python 的字符串可通过占位符%、format() 方法和 f-strings 三种方式实现格式化输出。

#### 3.6.3.1 占位符%

利用占位符%对字符串进行格式化时，Python 会使用一个带有格式符的字符串作为模板，这个格式符用于为真实值预留位置，并说明真实值应该呈现的格式。

例如：

```
>>> name='小强'  
>>> '你好，我叫%s' % name  
'你好，我叫小强'  
>>> '你好，我叫%s' %name  
'你好，我叫小强'  
>>> age=20  
>>> '你好，我叫%s, 今年%d岁' % (name,age)  
'你好，我叫小强，今年20岁'  
>>> age='20'  
>>> '你好，我叫%s, 今年%d岁' % (name,age)  
Traceback (most recent call last):  
  File "<pyshell>", line 1, in <module>  
TypeError: %d format: a number is required, not str
```

如果需要对多个变量进行格式化输出，可以使用()将多个变量存储起来。

常用的占位符如下所示：

```
%s: 字符串  
%d: 十进制整数  
%o: 八进制整数  
%x: 十六进制整数 (a~f 小写) , %X: 十六进制整数 (A~F 大写)  
%e: 指数 (底写为 e)  
%f: 浮点数
```

使用占位符%时需要注意变量的类型，类型与占位符不匹配则产生异常。

### 3.6.3.2 format()方法

format()方法同样可以对字符串进行格式化输出，与占位符%不同的是，使用format()方法不需要关注变量的类型。基本格式如下：

```
<字符串>.format(<参数列表>)
```

在format()方法中使用{}为变量预留位置。例如：

```
>>> name='小强'  
>>> age=20  
>>> '你好，我叫{}，今年我{}岁了。'.format(name,age)  
'你好，我叫小强，今年我20岁了。  
>>> '你好，我叫{1}，今年我{0}岁了。'.format(age,name)  
'你好，我叫小强，今年我20岁了。'
```

如果字符串中包含多个{}，并且{}内没有指定任何序号（从0开始编号），那么默认按照{}出现的顺序分别用format()方法中的参数进行替换；如果字符串的{}中明确指定了序号，那么按照序号对应的format()方法的参数进行替换。

format()方法还可以对数字进行格式化，包括保留n位小数、数字补齐和显示百分比。

- ①保留n位小数。格式为“{:.nf}”。
- ②数字补齐。格式为“{:m>nd}”，其中m表示补齐的数字，n表示补齐后数字的长度，>表示在原数字左侧进行补充。
- ③显示百分比。可以将数字以百分比形式显示。其格式为“{:.n%}”，其中n表示保留的小数位。

例如：

```
>>> pi=3.1415  
>>> '{:.2f}'.format(pi)  
'3.14'  
  
>>> num=1  
>>> '{:0>3d}'.format(num)  
'001'  
  
>>> num  
1  
  
>>> num=0.1  
>>> '{:.0%}'.format(num)  
'10%'
```

### 3.6.3.3 f-strings

f-strings是从Python3.6版本开始加入Python标准库的内容，它提供了一种更为简洁的格式化字符串方法。

f-strings 在格式上以 f 或 F 引领字符串，字符串中使用 {} 标明被格式化的变量。使用 f-strings 不需要关注变量的类型，但是仍然需要关注变量传入的位置。例如：

```
>>> addr='广州'  
>>> f'欢迎来到{addr}'  
'欢迎来到广州'  
>>> name='小强'  
>>> age=20  
>>> gender='男'  
>>> f'我的名字是{name}，今年{age}岁，性别{gender}。'  
'我的名字是小强，今年20岁，性别男。'
```

## 3.6.4 操作字符串

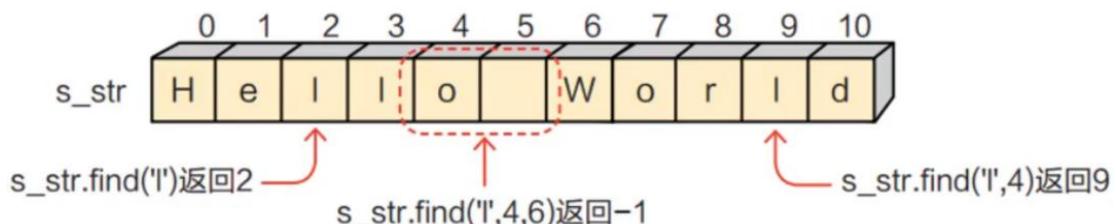
### 3.6.4.1 字符串拼接

字符串的拼接可直接使用“+”符号实现。例如：

```
>>> str_one='人生苦短，'  
>>> str_two='我用Python。'  
>>> str_one+str_two  
'人生苦短，我用Python。'
```

### 3.6.4.2 字符串查找

`str.find(sub[, start[, end]])` 查找子字符串，在索引 start 到 end 之间查找子字符串 sub，如果找到，则返回最左端位置的索引；如果没有找到，则返回 -1。例如：



### 3.6.4.3 字符串替换

`str.replace(old, new, count=None)`字符串替换, new 子字符串替换 old 子字符串。count 参数指定了替换 old 子字符串的个数, count 被省略, 则替换所有 old 子字符串。例如:

```
>>> word='我是小强, 我今年20岁'  
>>> word.replace('我','他')  
'他是小强, 他今年20岁'  
>>> word.replace('我','他',1)  
'他是小强, 我今年20岁'  
>>> word.replace('他','你')  
'我是小强, 我今年20岁'
```

### 3.6.4.4 字符串分割

`str.split(sep=None, maxsplit=-1)`, 使用 sep 子字符串分割字符串 str, 返回一个分割以后的字符串列表。maxsplit 是最大分割次数, 如果 maxsplit 被省略, 则表示不限制分割次数。例如:

```
>>> word="1 2 3 4 5"  
>>> word.split()  
['1', '2', '3', '4', '5']  
>>> word="a,b,c,d,e"  
>>> word.split(',')  
['a', 'b', 'c', 'd', 'e']  
>>> word.split(',',3)  
['a', 'b', 'c', 'd,e']
```

### 3.6.4.5 去除字符串两侧空格

`str.strip(chars=None)`, 一般用于去除字符串两侧的空格, 参数 chars 用于设置要去除的字符, 默认要去除的字符为空格。例如:

PRECHIN  
普中  
www.prechin.net

```
>>> word=' strip '
>>> word.strip()
'strip'
>>> word='**strip*'
>>> word.strip('*')
'strip'
```

## 3.7 函数

当程序实现的功能较为复杂时，开发人员通常会将其中的功能性代码定义为一个函数，提高代码复用性、降低代码冗余、使程序结构更加清晰。函数指被封装起来的、实现某种功能的一段代码，它可以被其它函数调用。

### 3.7.1 定义函数

在 Python 中，使用关键字 def 定义函数，其语法格式如下：

```
def 函数名([参数列表]):
    [”函数文档字符串”]
    函数体
    [return 语句]
```

格式介绍如下：

- (1) def 关键字：函数以 def 关键字开头，其后跟函数名和圆括号 ()。
- (2) 函数名：用于标识函数的名称，遵循标识符命名规则。
- (3) 参数列表：用于接收传入函数中的数据，可以为空。
- (4) 冒号：英文字符冒号，用于标识函数体的开始。
- (5) 函数文档字符串：用于注释，函数的说明信息，可以省略。
- (6) 函数体：实现函数功能的具体代码。
- (7) return 语句：用于将函数的处理结果返回给函数调用者，若函数没有返回值，return 语句可以省略。

例如：

```
>>> def rect_area():
    print("*" * 13)
    print('长方形面积: %d' %(12*2))
    print("*" * 13)

>>> rect_area
<function rect_area at 0x03217E40>
>>> rect_area()
*****
长方形面积: 24
*****


>>> def rect_area(width,height):
    area=width*height
    return area

>>> rect_area(10*2)
Traceback (most recent call last):
  File "<pyshell>", line 1, in <module>
TypeError: rect_area() missing 1 required positional argument: 'height'
>>> print(rect_area(10,2))
```

20

函数定义时可以设置参数列表，以实现更灵活的功能。

### 3.7.2 调用函数

函数调用格式如下：

函数名([参数列表])

定义好的函数直到被程序调用才会执行。例如前面小节的 rect\_area()。运行结果如前面小节截图。

调用带有参数的函数时需要传入参数，传入的参数称为实际参数，实际参数是程序执行过程中真正会使用的参数。例如前面的 rect\_area(10, 2)。调用该函数时为其传入 2 个参数，这些参数在函数体被执行时代替了形式参数。

### 3.7.3 函数参数传递

函数的参数传递是指将实际参数传递给形式参数的过程，根据不同的传递形式，函数的参数可分为位置参数、关键字参数、默认值参数、不定长参数。

### 3.7.3.1 位置参数

调用函数时，编译器会将函数的实际参数按照位置顺序依次传递给形式参数，即将第 1 个实参传给第 1 个形参，将第 2 个实参传递给第 2 个形参，依此类推。例如：

```
>>> def division(num_one,num_two):
    print(num_one/num_two)

>>> division(6,2)
3.0
```

### 3.7.3.2 关键字参数

使用位置参数传值时，如果函数中存在多个参数，记住每个参数的位置及其含义并不是一件容易的事，此时可以使用关键字参数进行传递。关键字参数传递通过“形式参数=实际参数”的格式将实际参数与形式参数相关联，根据形式参数的名称进行参数传递。例如：

```
>>> def info(name,age,addr):
    print(f'姓名: {name}')
    print(f'年龄: {age}')
    print(f'地址: {addr}')

>>> info(name='小王',addr='深圳',age=20)
姓名:小王
年龄:20
地址:深圳
```

### 3.7.3.3 默认参数

定义函数时可以指定形式参数的默认值，调用函数时，若没有给带有默认值的形参传值，则直接使用参数默认值；若给带有默认值的形参传值，则实际参数的值会覆盖默认值。例如：

```
>>> def info(name,age,sex='男'):
    print(f'姓名: {name}')
    print(f'年龄: {age}')
    print(f'性别: {sex}')

>>> info(name='小王',age=20)
姓名: 小王
年龄: 20
性别: 男

>>> info(name='小王',age=20,sex='女')
姓名: 小王
年龄: 20
性别: 女
```

注意：若函数中包含默认参数，该默认参数必须放在参数最后位置。

```
>>> def info(name,sex='男',age):
    print(f'姓名: {name}')
    print(f'年龄: {age}')
    print(f'性别: {sex}')

File "<pyshell>", line 1
SyntaxError: non-default argument follows default argument
```

### 3.7.3.4 不定长参数

若要传入函数中的参数的个数不确定，可以使用不定长参数。不定长参数也称可变参数，此种参数接收参数的数量可以任意改变。语法格式如下：

```
1 def 函数名([formal_args,] *args, **kwargs):
2     ["函数文档字符串"]
3     函数体
4     [return 语句]
```

上述参数`*args` 和参数`**kwargs` 都是不定长参数，这两个参数可搭配使用，也可单独使用。

#### ①`*args`

不定长参数`*args` 用于接收不定数量的位置参数，调用函数时传入的所有参数被`*args` 接收后以元组形式保存。定义一个包含参数`*args` 的函数如下：

```
>>> def test(*args):  
    print(args)  
  
>>> test(1,2,3,'a','b','c')  
(1, 2, 3, 'a', 'b', 'c')
```

## ②\*\*kwargs

不定长参数\*\*kwargs 用于接收不定数量的关键字参数，调用函数时传入的所有参数被\*\*kwargs 接收后以字典形式保存。定义一个包含\*\*kwargs 的函数如下：

```
>>> def test(**kwargs):  
    print(kwargs)  
  
>>> test(a=1,b=2,c=3,d=4)  
{'a': 1, 'b': 2, 'c': 3, 'd': 4}
```

## 3.7.4 变量作用域

变量作用域指变量的作用范围。根据作用范围，Python 中的变量分为局部变量和全局变量。

### 3.7.4.1 局部变量

局部变量在函数内定义的变量，只在定义它的函数内生效。函数外访问局部变量将报错。例如：

```
>>> def use_var():  
    name='Python'  
    print(name)  
  
>>> use_var()  
Python  
>>> print(name)  
Traceback (most recent call last):  
  File "<pyshell>", line 1, in <module>  
NameError: name 'name' is not defined
```

### 3.7.4.2 全局变量

全局变量是在函数外定义的变量，它在程序中任何位置都可以被访问。注意：

函数中只能访问全局变量，但不能修改全局变量。若要在函数内修改全局变量的值，需首先在函数内使用关键字 global 进行声明。例如：

```
>>> count=10
>>> def use_var():
    print(count)

>>> use_var()
10
>>> print(count)
10
>>> def use_var():
    global count
    count+=10
    print(count)

>>> use_var()
20
>>> def use_var():
    count+=10
    print(count)

>>> use_var()
Traceback (most recent call last):
  File "<pyshell>", line 1, in <module>
    File "<pyshell>", line 2, in use_var
      UnboundLocalError: local variable 'count' referenced before assignment
```

### 3.7.5 函数特殊形式

除了前面介绍的函数外，Python 还支持两种特殊形式的函数，即匿名函数和递归函数。

#### 3.7.5.1 匿名函数

匿名函数是无需函数名标识的函数，它的函数体只能是单个表达式。Python 中使用关键字 lambda 定义匿名函数，匿名函数的语法格式如下：

```
lambda [arg1 [, arg2,... argn]]:expression
```

上述格式中，“[arg1 [, arg2,... argn]]”表示匿名函数的形参，“expression”是一个表达式。

匿名函数与普通函数主要有以下不同：

- ①普通函数需要使用函数名进行标识，匿名函数不需要使用函数名标识。
- ②普通函数的函数体中可以有多条语句，匿名函数只能是一个表达式。
- ③普通函数可以实现比较复杂的功能，匿名函数只能实现比较单一的功能。
- ④普通函数可以被其他程序使用，匿名函数不能被其他程序使用。

为了方便使用匿名函数，通常使用变量来记录这个函数，例如：

```
>>> area=lambda a,h:(a*h)*0.5
>>> print(area)
<function <lambda> at 0x04F17F18>
>>> print(area(3,4))
6.0
```

### 3.7.5.2 递归函数

递归是一个函数过程在定义中直接或间接调用自身的一种方法，它通常把一个大型的复杂问题层层转化为一个与原问题相似，但规模较小的问题进行求解。如果一个函数中调用了函数本身，这个函数就是递归函数。

函数递归调用时，需要确定两点：一是递归公式；二是边界调节。递归公式是递归求解过程中的归纳项，用于处理问题以及与原问题规律相同的子问题；边界调节即终止条件，用于终止递归。

阶乘是利用递归方式求解的经典问题，例如：

```
>>> def factorial(num):
    if num==1:
        return 1
    else:
        return num*factorial(num-1)

>>> factorial(5)
120
```

### 3.7.6 常用内置函数

Python 内置了一些实现特定功能的函数，这些函数无须由用户重新定义，可直接使用。常用的内置函数如下：

- `abs()`: 计算绝对值，其参数必须是数字类型。
- `len()`: 返回序列对象（字符串、列表、元组等）的长度。
- `map()`: 根据提供的函数对指定的序列做映射。
- `help()`: 用于查看函数或模块的使用说明。
- `ord()`: 用于返回 Unicode 字符对应的码值。
- `chr()`: 与 `ord()` 功能相反，用于返回码值对应的 Unicode 字符。
- `filter()`: 用于过滤序列，返回由符合条件的元素组成的新列表。

例如：

```
>>> abs(-5)
5
>>> abs(3.14)
3.14
>>> len('Python')
6
>>> ord('A')
65
>>> chr(65)
'A'
>>> help(len)
Help on built-in function len in module builtins:
len(obj, /)
    Return the number of items in a container.
```

## 3.8 类与对象

### 3.8.1 面向对象

面向对象是程序开发领域中的重要思想，这种思想模拟了人类认识客观世界的逻辑，是当前计算机软件工程学的主流方法；类是面向对象的实现手段。Python 在设计之初就已经是一门面向对象语言。面向对象的三大基本特征：封装、继承、多态。

## 3.8.2 类与对象

### 3.8.2.1 类与对象的关系

面向对象编程思想力求在程序中对事物的描述与该事物在现实中的形态保持一致。为此面向对象的思想中提出了两个概念：类与对象。类是对多个对象共同特征的抽象描述，是对象的模板；对象用于描述现实中的个体，它是类的实例。

例如：汽车是人类出行所需的交通工具之一，厂商在生产汽车之前会先分析用户需求，设计汽车模型，制作设计图样。设计图样描述了汽车的各种属性和功能，例如汽车应该有方向盘、发动机、加速器等部件，也应该有执行制动、加速、倒车等操作。设计图通过之后工厂再依照图纸批量生产汽车。

其中汽车设计的图纸可以视为一个类，批量生产的汽车可以视为对象，由于按照同一图纸生产，这些汽车对象具有很多共性。

### 3.8.2.2 类的定义与访问

在程序中创建对象之前需要先定义类。类是对象的抽象，是一种自定义数据类型，它用于描述一组对象的共同特征和行为。类中可以定义数据成员和成员函数，数据成员用于描述对象的特征，成员函数用于描述对象的行为，其中数据成员也被称为属性，成员函数被称为方法。

类的定义格式如下：

```
1 class 类名:  
2     属性名=属性值  
3     def 方法名(self):  
4         方法体
```

上述格式中的 `class` 是定义类的关键字，其后的类名是类的标识符，类名首字母一般为大写。类名后的冒号（`:`）必不可少，之后的属性和方法都是类的成员，其中属性类似于前面学习的变量，方法类似于前面学习的函数，但需要注意，方法中有一个指向对象的默认参数 `self`。

下面定义一个 `Car` 类，代码如下：

```
1 class Car:  
2     wheels=4  
3     def drive(self):  
4         print('开车方法')  
5     def stop(self):  
6         print('停车方法')
```

### 3.8.2.3 对象的创建与使用

类定义完成后不能直接使用，这就好比画好一张房屋设计图纸，此图纸只能帮助人们了解房屋的结构，但不能提供居住场所，需要根据房屋设计图纸搭建实际的房屋。同理，程序中的类需要实例化为对象才能实现其意义。

#### ①对象的创建

创建对象格式如下：

对象名=类名()

例如：创建前面定义好的 Car 对象 my\_car，如下：

my\_car=Car()

#### ②访问对象成员

若想在程序中真正的使用对象，需掌握访问对象成员方式，对象成员分属性和方法，访问格式如下：

对象名. 属性

对象名. 方法()

例如：访问上述 Car 类对象 my\_car 成员，如下：

```
1 class Car:  
2     wheels=4  
3     def drive(self):  
4         print('开车方法')  
5     def stop(self):  
6         print('停车方法')  
7  
8     my_car=Car()  
9     print(my_car.wheels)  
10    my_car.drive()
```

### 3.8.2.4 访问限制

类中定义的属性和方法默认为公有属性和方法，该类的对象可以任意访问类的公有成员，但考虑到封装思想，类中的代码不应被外部代码轻易访问。为了契合封装原则，Python 支持将类中的成员设置为私有成员，在一定程度上限制对象对类中的成员访问。

#### ①定义私有成员

Python 通过在类成员名之前添加双下划线（\_\_）来限制成员的访问权限，语法格式如下：

\_\_属性名  
\_\_方法名

例如：

```
1 class PersonInfo:  
2     __weight=55  
3     def __info(self):  
4         print(f'我的体重是: {__weight}')
```

#### ②私有成员的访问

创建 PersonInfo 类的对象 person，通过该对象访问类的私有属性，具体代码如下：

```
>>> class PersonInfo:  
...     __weight=55  
...     def __info(self):  
...         print(f'我的体重是: {__weight}')  
  
>>> person=PersonInfo()  
>>> person.__weight  
Traceback (most recent call last):  
  File "<pyshell>", line 1, in <module>  
AttributeError: 'PersonInfo' object has no attribute '__weight'  
  
>>> person.__info()  
Traceback (most recent call last):  
  File "<pyshell>", line 1, in <module>  
AttributeError: 'PersonInfo' object has no attribute '__info'
```

对象无法直接访问类的私有成员。下面分别介绍如何在类内部访问私有属性和私有方法：

(1) 访问私有属性。私有属性可在公有方法中通过指代类本身默认参数 self 访问，类外部可通过公有方法间接获取类的私有属性。例如：

```
>>> class PersonInfo:  
    __weight=55  
    def get_weight(self):  
        print(f'体重: {self.__weight}kg')  
  
>>> person=PersonInfo()  
>>> person.get_weight()  
体重: 55kg
```

(2) 访问私有方法。私有方法同样在公有方法中通过参数 self 访问，修改 PersonInfo 类，在私有方法 \_\_info() 中通过 self 参数访问私有属性 \_\_weight，并在公有方法 get\_weight() 中通过 self 参数访问私有方法 \_\_info()，代码如下：

```
>>> class PersonInfo:  
    __weight=55  
    def __info(self):  
        print(f'我的体重是: {self.__weight}')  
    def get_weight(self):  
        print(f'体重: {self.__weight}kg')  
        self.__info()  
  
>>> person=PersonInfo()  
>>> person.get_weight()  
体重: 55kg  
我的体重是: 55
```

### 3.8.3 构造方法与析构方法

类中有两个特殊的方法：构造方法 \_\_init\_\_() 和析构方法 \_\_del\_\_()。这两个方法分别在类创建和销毁时自动调用。

#### 3.8.3.1 构造方法

每个类都有一个默认的 \_\_init\_\_() 方法，如果在定义类时显式地定义了 \_\_init\_\_() 方法，则创建对象时 Python 解释器会调用显式定义的 \_\_init\_\_() 方法；如果定义类时没有显示地定义 \_\_init\_\_() 方法，则 Python 解释器会调用默认的 \_\_init\_\_() 方法。

`__init__()`方法按照参数的有无(`self`除外)可分为有参构造方法和无参构造方法，无参构造方法中可以为属性设置初始值，此时使用该方法创建的所有对象都具有相同的初始值。若希望每次创建的对象都有不同的初始值，则可以使用有参构造方法实现。

例如：

```
>>> class Inforamtion(object):
    def __init__(self, name, sex):
        self.name=name
        self.sex=sex
    def info(self):
        print(f'姓名: {self.name}')
        print(f'性别: {self.sex}')
```

上述代码中首先定义了一个包含3个参数的构造方法，然后通过参数`name`和`sex`为属性`name`和`sex`进行赋值，最后在`info()`方法中访问属性的值。

注意：在构造方法中定义的属性是实例属性，只能通过对对象进行访问。前面在类中定义的属性是类属性，可通过类或对象访问。

因为定义的构造方法中需要接收两个实参，所以实例化对象时需要传入2个参数，如下：

```
>>> infomat=Inforamtion('小王','女')
>>> infomat.info()
姓名:小王
性别:女
```

### 3.8.3.2 析构方法

在创建对象时，系统自动调用`__init__()`方法，在对象被清理时，系统也会自动调用一个`__del__()`方法，这个方法就是类的析构方法。

在介绍析构方法之前，先了解下Python的垃圾回收机制。Python中的垃圾回收主要采用的是引用计数。引用计数是一种内存管理计数，它通过引用计数器记录所有对象的引用数量，当对象的引用计数器数值为0时，就会将该对象视为垃圾进行回收。`getrefcount()`函数是`sys`模块中用于统计对象引用数量的函数，其返回结果通常比预期的结果大1，这是因为`getrefcount()`函数也会统计临时对象的引用。

当一个对象的引用计数器数值为 0 时，就会调用`__del__()`方法，例如：

```
1 import sys
2 class Destruction:
3     def __init__(self):
4         print('对象被创建')
5     def __del__(self):
6         print('对象被释放')
7
8 des=Destruction()
9 print(sys.getrefcount(des))
10 del(des)
11 |
```

Shell >

```
>>> %Run main.py
对象被创建
2
对象被释放
```

从结果可以看出，对象被创建以后，其引用计数器的值变为 2，由于返回引用计数器的值会整加一个临时引用，因此对象引用计数器的值实际为 1。

### 3.8.4 类方法和静态方法

类中的方法可以有三种定义形式，直接定义只比普通函数多了一个`self`参数的方法是类最基本的方法，这种方法称为实例方法，它只能通过类实例化的对象调用。除此之外，Python 中的类还可定义使用`@classmethod`修饰的类方法和使用`@staticmethod`修饰的静态方法，下面分别介绍这两种方法。

#### 3.8.4.1 类方法

类方法与实例方法有以下不同：

- (1) 类方法使用装饰器`@classmethod`修饰。
- (2) 类方法的第一个参数为`cls`而非`self`，它代表类本身。
- (3) 类方法即可由对象调用，亦可直接由类调用。
- (4) 类方法可以修改类属性，实例方法无法修改类属性。

## ①定义类方法

类方法可以通过类名或对象名进行调用，其语法格式如下：

类名. 类方法  
对象名. 类方法

例如：

```
>>> class Test:  
    @classmethod  
    def use_classmet(cls):  
        print('我是类方法')  
  
>>> test=Test()  
>>> test.use_classmet()  
我是类方法  
>>> Test.use_classmet()  
我是类方法
```

使用类名或对象名均可调用类方法。

## ②修改类属性

在实例方法中无法修改类属性的值，但在类方法中可以将类属性的值进行修改。例如：

```
>>> class Apple():  
    count=0  
    def add_one(self):  
        self.count=1  
    @classmethod  
    def add_two(cls):  
        cls.count=2  
  
>>> apple=Apple()  
>>> apple.add_one()  
>>> print(Apple.count)  
0  
>>> Apple.add_two()  
>>> print(Apple.count)  
2
```

从输出结果中可以看出，调用 add\_one() 后访问 count 的值为 0，说明属性 count 的值并没有被修改；调用类方法 add\_two() 后再次访问 count 的值为 2，说明类属性 count 的值被修改成功。

可能大家会存在这样的疑惑，在实例方法 add\_one() 中明明通过“self.count=1”重新为 count 赋值，为什么 count 的值仍然为 0 呢？这是因为通过“self.count=1”只是创建一个与类属性同名的实例属性 count 并将其赋值为 1，而非对类属性重新赋值。

### 3.8.4.2 静态方法

静态方法与实例方法有以下不同：

- (1) 静态方法没有 self 参数，它需要使用@staticmethod 修饰。
- (2) 静态方法中需要以“类名. 方法/属性名”的形式访问类的成员。
- (3) 静态方法即可由对象调用，亦可直接由类调用。

例如：

```
>>> class Example:  
    num=10  
    @staticmethod  
    def static_method():  
        print(f'类属性的值为: {Example.num}')  
        print('----静态方法')  
  
>>> example=Example()  
>>> example.static_method()  
类属性的值为: 10  
----静态方法  
  
>>> Example.static_method()  
类属性的值为: 10  
----静态方法
```

### 3.8.5 继承

“龙生龙，凤生凤，老鼠的儿子会打洞”，这句话将动物界中的继承关系表现的淋漓尽致。在 Python 中，类与类之间也具有继承关系，其中被继承的类称为父类或几类，派生的类称为子类或派生类。子类在继承父类时，会自动拥有父类中的方法和属性。

### 3.8.5.1 单继承

单继承指的是子类只继承一个父类，其语法格式如下：

```
class 子类(父类):
```

例如：

```
1 class Animal:
2     name='动物'
3     def features(self):
4         print("Animal类方法")
5
6 class Dog(Animal):
7     def feed(self):
8         print(f'狗是{self.name}')
9         print('狗吃骨头')
10
11 dog=Dog()
12 print(dog.name)
13 dog.features()
14 dog.feed()
15
```

Shell ×

```
Python 3.7.9 (bundled)
>>> %Run main.py
>>> %Run main.py
```

```
动物
Animal类方法
狗是动物
狗吃骨头
```

上述代码中定义了 Animal 类，其包含属性 name 和实例方法 features()，Dog 类继承 Animal 类并定义了自己的方法 feed()。

从输出结果看，子类继承父类之后，就拥有父类继承的属性和方法，它既可以调用自己的方法，又可以调用从父类继承的方法。

Python 中提供了两个与继承相关的函数，分别是 isinstace() 函数和 issubclass() 函数。

isinstace(o, t) 函数用于检查对象的类型，它有 2 个参数，第 1 个参数是要判断类型的对象(o)，第 2 个参数是类型(t)，如果 o 是 t 类型的对象，则函数返回 True，否则返回 False。例如：

```
>>> isinstance(dog,Dog)
True
```

函数 `issubclass(cls, classinfo)` 用于检查类的继承关系，它有 2 个参数，第 1 个参数是要判断的子类类型 (`cls`)；第 2 个参数是要判断的父类类型 (`classinfo`)。如果 `cls` 类型是 `classinfo` 类型的子类，则函数返回 `True`，否则返回 `False`。例如：

```
>>> issubclass(Dog,Animal)
True
```

### 3.8.5.2 多继承

多继承是一个子类继承多个父类，语法格式如下：

```
class 子类(父类 A, 父类 B):
```

例如：

```
1  class English:
2      def eng_know(self):
3          print('具备英语知识')
4  class Math:
5      def math_know(self):
6          print('具备数学知识')
7
8  class Student(English,Math):
9      def study(self):
10         print('学生的任务是学习')
11
12 s=Student()
13 s.eng_know()
14 s.math_know()
15 s.study()
```

Shell ×

```
>>> %Run main.py
```

```
具备英语知识
具备数学知识
学生的任务是学习
```

### 3.8.5.3 方法的重写

子类可以继承父类的属性和方法，若父类的方法不能满足子类的要求，子类可以重写父类的方法，以实现理想的功能。例如：

```
1 class Felines:
2     def feature(self):
3         print('猫科动物特长是爬树')
4 class Cat(Felines):
5     name='猫'
6     def feature(self):
7         print(f'{self.name}会抓老鼠')
8         print(f'{self.name}会爬树')
9
10 cat=Cat()
11 cat.feature()
```

Shell >>> %Run main.py  
猫会抓老鼠  
猫会爬树

### 3.8.5.4 super()函数

如果子类重写了父类的方法，但仍希望调用父类中的方法，该如何实现？Python 提供了一个 super() 函数，使用该函数可以调用父类中的方法。使用方法如下：

super().方法名()

例如：

```
1 class Felines:
2     def feature(self):
3         print('猫科动物特长是爬树')
4 class Cat(Felines):
5     name='猫'
6     def feature(self):
7         print(f'{self.name}会抓老鼠')
8         print(f'{self.name}会爬树')
9         print('-'*20)
10    super().feature()
11
12
13 cat=Cat()
14 cat.feature()
15
```

Shell >

```
>>> %Run main.py
猫会抓老鼠
猫会爬树
-----
猫科动物特长是爬树
```

### 3.8.6 多态

在 Python 中，多态指在不考虑对象类型的情况下使用对象。相比于强类型，Python 更推崇“鸭子类型”。“鸭子类型”是这样推断的：如果一只生物走起路来像鸭子，游起泳来像鸭子，叫起来也像鸭子，那么它就可以被当做鸭子。也就是说“鸭子类型”不关注对象的类型，而是关注对象具有的行为。

Python 中并不需要显式指定对象的类型，只要对象具有预期的方法和表达式操作符，就可以使用对象。也可以说，只要对象支持所预期的“接口”，就可以使用，从而实现多态。一个体现多态特性的示例如下：

```
1 class Animal(object):
2     def move(self):
3         pass
4
5 class Rabbit(Animal):
6     def move(self):
7         print('兔子蹦蹦跳跳')
8
9 class Snail(Animal):
10    def move(self):
11        print('蜗牛缓慢爬行')
12
13 def test(obj):
14     obj.move()
15
16 rabbit=Rabbit()
17 test(rabbit)
18 snail=Snail()
19 test(snail)
20
```

Shell ×

```
>>> %Run main.py
```

```
兔子蹦蹦跳跳
蜗牛缓慢爬行
```

同一个函数会根据参数的类型去调用不同的方法，从而产生不同的结果。

## 3.9 异常处理

现实生活中并不是一帆风顺的，总会遇到各种突发情况，例如：飞机延误、火车晚点、公交车堵车等，这些情况可能会导致上班迟到、约会赶不上。在程序中也会遇到各种各样的问题，例如除数为 0 时会引发除零异常。

为避免因各种异常状况导致程序崩溃，程序开发中引入了异常处理机制，以处理或修复程序中可能出现的错误，提供诊断信息，帮助开发人员尽快解决问题，恢复程序的正常运行。

### 3.9.1 除零异常

除零异常 ZeroDivisionError，例如：

```
1 i=input('请输入数字: ')
2 n=12
3 result=n/int(i)
4 print(result)
5 print(f'{n}除以{i}等于{result}')

Shell < 
>>> %Run calc.py
请输入数字: 2
6.0
12除以2等于6.0
>>> %Run calc.py
请输入数字: 0
Traceback (most recent call last):
  File "H:\普中-ESP32开发板资料\4--实验程序\1--MicroPython实验\1--基础实验\calc.py", line 3, in <module>
    result=n/int(i)
ZeroDivisionError: division by zero
```

### 3.9.2 捕获异常

Python 程序在运行时出现的异常会导致程序崩溃，因此开发人员需要用一种友好的方式处理程序运行时的异常。在 Python 中可使用 try...except 语句捕获一次，try...except 还可以与 else、finally 组合使用实现更强大的异常处理功能。

#### 3.9.2.1 try...except 语句

try...except 语句用于捕获程序运行时的异常，语法格式如下：

```
try:
    可能出错的代码
    ...
except [异常类型]:
    错误处理语句
    ...
```

try...except 语句的执行过程如下：

- ①先执行 try 子句，即 try 与 except 之间的代码。
- ②若 try 子句中没有产生异常，则忽略 except 子句中的代码。
- ③若 try 子句产生异常，则忽略 try 子句的剩余代码，执行 except 子句中的代码。

例如：将除零异常代码进行改进，如下：

```
1 i=input('请输入数字: ')
2 n=12
3 try:
4     result=n/int(i)
5     print(result)
6     print(f'{n}除以{i}等于{result}')
7 except:
8     print('除数不能为0! ')
```

Shell ×

```
>>> %Run calc.py
```

```
请输入数字: 2
6.0
12除以2等于6.0
```

```
>>> %Run calc.py
```

```
请输入数字: 0
除数不能为0!
```

### 3.9.2.2 捕获异常信息

try...except 语句可以捕获和处理程序运行时的单个异常、多个异常、所有异常，也可以在 except 子句中使用关键字 as 获取系统反馈的异常的具体信息。

#### ①捕获程序运行时的单个异常

使用 try...except 语句捕获和处理单个异常时，需要在 except 子句的后面指定具体的异常类，例如：

```
1 i=input('请输入数字: ')
2 n=12
3 try:
4     result=n/int(i)
5     print(result)
6     print(f'{n}除以{i}等于{result}')
7 except ZeroDivisionError as e:
8     print('除数不能为0! ')
9     print(f'异常原因: {e}')
```

Shell ×

```
>>> %Run calc.py
```

```
请输入数字: 0
除数不能为0!
异常原因: division by zero
```

注意：如果指定的异常与程序产生的异常不一致，程序运行时仍会崩溃。

### ②捕获程序运行时的多个异常

一段代码中可能产生多个异常，此时可以将多个具体的异常类组成元组放在 except 语句后处理，也可以联合使用多个 except 语句。例如：

The screenshot shows two code cells in a Jupyter Notebook environment. The first cell contains Python code with a single except clause:

```
1 try:  
2     print(count)  
3     demo_list=['Python','Java','C','C++']  
4     print(demo_list[4])  
5 except (NameError,IndexError) as e:  
6     print(f'异常错误原因: {e}')  
7
```

The second cell shows the output of running the code:

```
Shell >  
>>> %Run calc.py  
异常错误原因: name 'count' is not defined
```

The second cell contains Python code with multiple except clauses:

```
1 try:  
2     print(count)  
3     demo_list=['Python','Java','C','C++']  
4     print(demo_list[4])  
5 except NameError as e:  
6     print(f'异常错误原因: {e}')  
7 except IndexError as e:  
8     print(f'异常错误原因: {e}')  
9
```

The second cell shows the output of running the code:

```
Shell >  
Python 3.7.9 (bundled)  
>>> %Run calc.py  
异常错误原因: name 'count' is not defined
```

### ③捕获程序运行时的所有异常

在 Python 中，使用 try..except 语句捕获所有异常有两种方式：指定异常类为 Exception 类和省略异常类。例如：

```
1 try:  
2     print(count)  
3     demo_list=['Python','Java','C','C++']  
4     print(demo_list[4])  
5 except Exception as e:  
6     print(f'异常错误原因: {e}')  
7  
8
```

Shell ×

```
>>> %Run calc.py  
异常错误原因: name 'count' is not defined
```

```
1 try:  
2     print(count)  
3     demo_list=['Python','Java','C','C++']  
4     print(demo_list[4])  
5 except :  
6     print('程序出现异常, 原因未知')  
7  
8
```

Shell ×

```
>>> %Run calc.py  
程序出现异常, 原因未知
```

Exception 类是常见异常类的父类，因此它可以指代所有常见的异常类。使用省略异常类的方式也能捕获所有常见的异常，但这种方式不能获取异常的具体信息。

### 3.9.2.3 else 子句

异常处理的主要目的是防止因外部环境的变化导致程序产生无法控制的错误，而不是处理程序设计的错误。因此将所有代码都用 try 子句包含起来的做法是不推荐的，try 子句因尽量只包含可能产生异常的代码。Python 中 try...except 还可以与 else 子句联合使用，该子句放在 except 语句之后，当 try 子句没有出现错误时应执行 else 语句中的代码。格式如下：

```
1 try: 可能出错的语句
2 ...
3 ...
4 except: 出错后的执行语句
5
6 else: 未出错时的执行语句
7
8
```

例如：某程序的分页显示数据功能可以根据用户输入控制每页显示多少条数据，但要求用户输入的数据为整型数据，如果输入的数据符合输入要求，每页显示用户指定条数的数据；如果输入的数据不符合要求，则显示默认条数的数据。如下：

The screenshot shows a Jupyter Notebook interface with two sections:

- Code Cell:** Contains Python code for reading user input and displaying data per page. It uses a try-except block to handle non-integer inputs and a default value of 10 if no input is provided.

```
1 num=input('请输入每页显示多少条数据: ')
2 try:
3     page_num=int(num)
4 except Exception as e:
5     page_num=10
6     print(f'当前页面显示{page_num}条数据')
7 else:
8     print(f'当前页面显示{num}条数据')
9
```

- Shell:** Shows the execution of the code and its output. Three runs are shown:

  - First run: Input '8', Output '当前页面显示8条数据'
  - Second run: Input '20', Output '当前页面显示20条数据'
  - Third run: Input 'test', Output '当前页面显示10条数据'

```
Python 3.7.9 (bundled)
>>> %Run calc.py
请输入每页显示多少条数据: 8
当前页面显示8条数据

>>> %Run calc.py
请输入每页显示多少条数据: 20
当前页面显示20条数据

>>> %Run calc.py
请输入每页显示多少条数据: test
当前页面显示10条数据
```

### 3.9.2.4 finally 子句

finally 子句与 try...except 语句连用时，无论 try...except 是否捕获到异常，finally 子句后的代码都要执行，语法格式如下：

```
1 try:          可能出错的语句
2     ...
3
4 except:       出错后的执行语句
5     ...
6
7 finally:      无论是否错误都会执行的语句
8
9
```

### 3.9.3 抛出异常

Python 程序中的异常不仅可以由系统抛出，还可以由开发人员使用关键字 `raise` 主动抛出。只要异常没有被处理，异常就会向上传递，直至最顶级也未处理，则会使用系统默认的方式处理（程序崩溃）。

#### 3.9.3.1 `raise` 语句

`raise` 语句用于引发特定的异常，其使用方式大致可分为 3 种：

- (1) 由异常类名引发异常。
- (2) 由异常对象引发异常。
- (3) 由程序中出现过的异常引发异常。

##### ① 使用类名引发异常

语法格式如下：

```
raise 异常类名
```

当 `raise` 语句指定了异常的类名时，Python 解释器会自动创建该异常类的对象，进而引发异常。例如：

```
>>> raise NameError
Traceback (most recent call last):
  File "<pyshell>", line 1, in <module>
NameError
```

##### ② 使用异常对象引发异常

使用异常对象引发相应异常，语法格式如下：

```
raise 异常对象
```

例如：

```
1 name_error=NameError()
2 raise name_error
3
Shell >
>>> %Run calc.py
Traceback (most recent call last):
File "H:\普中-ESP32开发板资料\4--实验程序\1--MicroPython实验\1--基础实验\calc.py", line 2, in <module>
    raise name_error
NameError
```

### ③由异常引发异常

仅使用 `raise` 关键字可重新引发刚才发生的异常，语法格式如下：

```
raise
```

例如：

```
1 try:
2     num
3 except NameError as e:
4     raise
5
6
Shell >
>>> %Run calc.py
Traceback (most recent call last):
File "H:\普中-ESP32开发板资料\4--实验程序\1--MicroPython实验\1--基础实验\calc.py", line 2, in <module>
    num
NameError: name 'num' is not defined
```

#### 3.9.3.2 异常的传递

如果程序中的异常没有被处理，默认情况下会将该异常传递给上一级，如果上一级仍然没有处理，会继续向上传递，直至异常被处理或程序崩溃。

例如：

```
1 def get_width():
2     print('get_width开始执行')
3     num=int(input('请输入除数: '))
4     width_len=10/num
5     print('get_width执行结束')
6     return width_len
7 def calc_area():
8     print('calc_area开始执行')
9     width_len=get_width()
10    print('calc_area执行结束')
11    return width_len*width_len
12 def show_area():
13     try:
14         print('show_area开始执行')
15         area_val=calc_area()
16         print(f'正方形的面积是: {area_val}')
17         print('show_area执行结束')
18     except ZeroDivisionError as e:
19         print(f'捕获到异常: {e}')
20 if __name__=='__main__':
21     show_area()
```

Shell ×

```
>>> %Run calc.py
show_area开始执行
calc_area开始执行
get_width开始执行
请输入除数: 0
捕获到异常: division by zero
```

### 3.9.4 自定义异常

Python 中定义了大量的异常类，虽然这些异常类可以描述编程时出现的绝大部分情况，但仍难以涵盖所有可能出现的异常。Python 允许程序开发人员自定义异常。自定义异常的方法很简单，只需创建一个类，让它继承 Exception 类或其他的异常类即可。

定义一个继承自异常类 Exception 的类 CustomError，例如：

```
1 class CustomError(Exception):
2     pass
```

```
1 class CustomError(Exception):
2     pass
3
4     try:
5         pass
6         raise CustomError('出现错误')
7     except CustomError as e:
8         print(e)
```

Shell >

```
>>> %Run calc.py
出现错误
```

自定义异常类与普通类一样，也可以包含属性和方法，但一般情况下不添加或只为其添加几个用于描述异常的详细信息的属性即可。

## 3.10 模块

模块（Module）是一个扩展名为. py 的 Python 文件，这个文件中包含许多功能函数或类，多个模块可以通过包组织。

### 3.10.1 模块的概念

在 Python 程序中，每个. py 文件都可以视为一个模块，通过在当前. py 文件导入其他. py 文件，可以使用被导入文件中定义的内容，如类、变量、函数等。

Python 中的模块可分为三类：内置模块、第三方模块和自定义模块。

①内置模块是 Python 内置的标准库中的模块，也是 Python 的官方模块，可直接导入使用。

②第三方模块是有非官方制作发布、供大众使用的 Python 模块，在使用前需要安装。

③自定义模块是开发人员在程序编写过程中自行编写的，存放功能性代码的. py 文件。

### 3.10.2 模块的导入方式

Python 模块的导入方式分为使用 import 导入和使用 from...import... 导入两种。

#### ① 使用 import 导入

使用 import 导入格式如下：

```
import 模块 1, 模块 2, ...
```

例如：

```
1 import time  
2 import random, sys
```

模块导入后便可以通过“.”使用模块中的函数或类，语法格式如下：

```
模块名. 函数名() / 类名
```

例如：

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

如果在开发过程中需要导入一些名称较长的模块，可使用 as 为这些模块起别名，格式如下：

```
import 模块名 as 别名
```

#### ② 使用 from...import... 导入

使用该方式导入模块之后，无需添加前缀，格式如下：

```
from 模块名 import 函数/类/变量
```

该方式也支持一次导入多个函数、类或变量，使用逗号隔开。例如：导入 time 模块中的 sleep() 函数和 time() 函数，具体代码如下：

```
from time import sleep, time
```

利用通配符“\*”可使用该方式导入模块中的全部内容，格式如下：

```
from 模块名 import *
```

该方式也支持为模块或模块中的函数起别名，格式如下：

```
from 模块名 import 函数名 as 别名
```

例如：

```
from time import sleep as sl
```

### 3.10.3 自定义模块

一般在进行程序开发时，不会将所有代码都放在一个文件中，而是将耦合度较低的多个功能写入不同的文件中，制作成模块，并在其它文件中以导入模块的方式使用自定义模块中的内容。

Python 中每个文件都可以作为一个模块存在，文件名即为模块名。例如：有一个 main 的 Python 文件，内容如下：

```
main.py
1 age=13
2 def introduce():
3     print(f'我的名字是小王, 今年{age}岁')
4
```

main 文件便可视为一个模块，该模块中 age 变量和 introduce() 函数都可以在导入该模块的程序中使用。与标准模块相同，自定义模块也可通过 import 语句和 from... import... 语句导入。例如：

```
main.py
1 import main
2 main.introduce()
3 print(main.age)
4

Shell
>>> %Run test.py
我的名字是小王, 今年13岁
13

main.py
1 from main import introduce
2 introduce()
```

```
Shell
>>> %Run test.py
我的名字是小王, 今年13岁
```

## 3.10.4 模块的导入特性

### 3.10.4.1 \_\_all\_\_ 属性

Python 模块的开头通常会定义一个`__all__`属性，该属性实际上是一个列表，该列表中包含的元素决定了在使用`from... import *`语句导入模块内容时通配符“\*”所包含的内容。如果`__all__`中只包含模块的部分内容，那么“`from... import *`”语句只会将`__all__`中包含的部分内容导入程序。

例如：当前有一个自定义模块`calc.py`，该模块中包含计算两个数的四则运算函数，代码如下：

The screenshot shows a code editor with two tabs: `calc.py` and `test.py`. The `calc.py` tab is active, displaying the following code:

```
1 __all__=['add','subtract']
2
3 def add(a,b):
4     return a+b
5 def subtract(a,b):
6     return a-b
7 def multiply(a,b):
8     return a*b
9 def divide(a,b):
10    if b:
11        return a/b
12    else:
13        print('error')
```

The `test.py` tab is also visible, showing the following code:

```
1 from calc import *
2
3 print(add(2,3))
4 print(subtract(2,3))
5 print(multiply(2,3))
6 print(divide(2,3))
```

Below the code editor is a terminal window titled "Shell" with the following output:

```
>>> %Run test.py
5
-1
Traceback (most recent call last):
  File "H:\普中-ESP32开发板资料\4--实验程序\1--MicroPython实验\1--基础实验\test.py", line 5, in <module>
    print(multiply(2,3))
NameError: name 'multiply' is not defined
```

### 3.10.4.2 \_\_name\_\_ 属性

在较大型的项目开发中，一个项目通常由多名开发人员共同进行，每名开发人员负责不同的模块。为了保证自己编写的程序在整合后可以正常运行，开发人员通常需要在整合前额外编写测试代码，对自己负责的模块进行测试。然而，对整个项目而言，这些测试代码是无用的。为了避免项目运行时执行这些测试代码，Python 中增加了 \_\_name\_\_ 属性。

\_\_name\_\_ 属性通常与 if 条件语句一起使用，若当前模块是启动模块，则其 \_\_name\_\_ 的值为 “\_\_main\_\_”；若该模块被其它程序导入，则 \_\_name\_\_ 的值为文件名。

```
calc.py x test.py x
1  __all__=['add','subtract']
2
3  def add(a,b):
4      return a+b
5  def subtract(a,b):
6      return a-b
7  def multiply(a,b):
8      return a*b
9  def divide(a,b):
10     if b:
11         return a/b
12     else:
13         print('error')
14
15 if __name__=='__main__':
16     print(add(2,3))
17     print(subtract(2,3))
18     print(multiply(2,3))
19     print(divide(2,3))

Shell x
>>> %Run calc.py
5
-1
6
0.6666666666666666
```

## 3.10.5 Python 中的包

### 3.10.5.1 包的结构

为了更好的组织 Python 代码，开发人员通常会根据不同业务将模块进行归类划分，并将功能相近的模块放在一个目录下。如果想要导入该目录下的模块，就需要先导入包。

Python 中的包是一个包含 `__init__.py` 文件的目录，该目录下还包含一些模块和子包。例如：

```
package
|---__init__.py
|---module_a1.py
|---module_a2.py
|---package_b
|   |---__init__.py
|   |---module_b.py
```

包的存在使整个项目更富有层次，也可在一定程度上避免合作开发中模块重名的问题。包中的 `__init__.py` 文件可以为空，但必须存在，否则包将退化为一个普通目录。

值得一提的是，`__init__.py` 文件有两个作用，第一个作用是标识当前目录是一个 Python 包；第二个作用是模糊导入。如果 `__init__.py` 文件中没有声明 `__all__` 属性，那么使用 “`from... import *`” 导入的内容为空。

### 3.10.5.2 包的导入

包的导入与模块的导入方法大致相同，也是使用 `import` 或 `from... import...` 实现。

假设现有一个包 `package_demo`，该包中包含模块 `module_demo`，模块中有一个 `add()` 函数，代码如下：

```
1 def add(num1,num2):
2     print(num1+num2)
```

## ①使用 import 导入

使用 import 导入包中的模块时，需要在模块名的前面加上包名，格式为“包名. 模块名”。若要使用已导入模块中的函数，需要通过“包名. 模块名. 函数名”实现。

例如：使用 import 方式导入包 package\_demo，并使用 module\_demo 模块中的 add() 函数，代码如下：

```
1 import package_demo.module_demo
2 package_demo.module_demo.add(1,3)
```

## ②使用 from... import... 导入

通过 from... import... 导入包中模块包含的内容时，若需要使用导入模块中的函数，需要通过“模块. 函数”实现。

例如：

```
1 from package_demo import module_demo
2 module_demo.add(1,3)
```

# 课后作业

## 第 4 章 LED 实验

不论学习什么单片机，最简单的外设莫过于 IO 口的高低电平控制 LED，本章将向大家介绍如何使用 MicroPython 控制 ESP32 的 GPIO 输出。通过本章的学习，让大家对 MicroPython 的程序架构有一定的认识，为以后大型项目程序学习打下基础，增强信心。本章分为如下几部分内容：

- 4. 1 实验介绍
- 4. 2 硬件设计
- 4. 3 软件设计
- 4. 4 实验现象

## 4.1 实验介绍

### 4.1.1 实验简介

相信大部分人开始学习嵌入式单片机编程时都会从点亮 LED 开始，我们在学习 ESP32 使用 MicroPython 的编程也不例外，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。

### 4.1.2 实验目的

点亮 LED 模块中的一个 LED 灯，即让 GPIO 输出高或低电平。

### 4.1.3 MicroPython 函数使用

MicroPython 中可使用 machine 模块中的 Pin 模块对 GPIO 输出控制。其构造方法和使用方法如下：

构造函数
<code>led=machine.Pin(id,mode,pull)</code>
构建 led 对象。 <code>id</code> :引脚编号； <code>mode</code> :输入输出方式； <code>pull</code> :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： <code>x=0</code> 表示低电平， <code>x=1</code> 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”。

图中对 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。本章实验中使用方式 2 导入模块, 代码更直观方便。

Pin 模块的使用方法如下:

```
from machine import Pin

p0 = Pin(0, Pin.OUT)      # 创建对象p0, 对应GPIO0口输出
p0.on()                   # 设置引脚为 "on" (1)高电平
p0.off()                  # 设置引脚为 "off" (0)低电平
p0.value(1)                # 设置引脚为 "on" (1)高电平

p2 = Pin(2, Pin.IN)       # 创建对象p2, 对应GPIO2口输入
print(p2.value())          # 获取引脚输入值, 0 (低电平) 或者 1 (高电平)

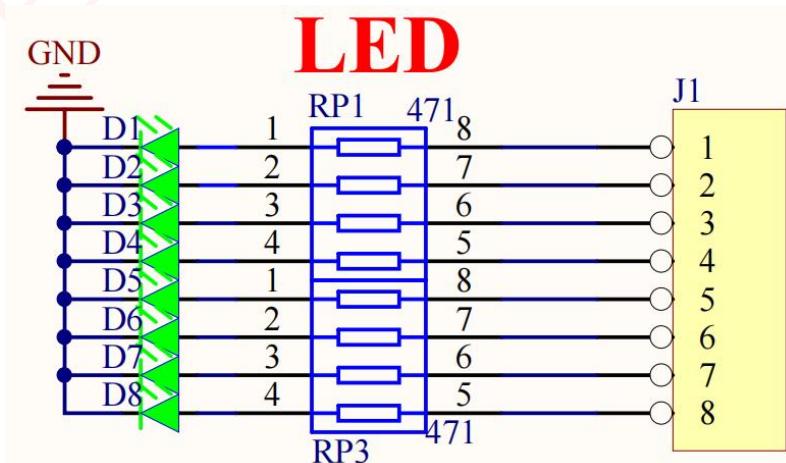
p4 = Pin(4, Pin.IN, Pin.PULL_UP) # 打开内部上拉电阻
p5 = Pin(5, Pin.OUT, value=1) # 初始化时候设置引脚的值为 1 (高电平)
```

## 4.2 硬件设计

本实验使用到硬件资源如下:

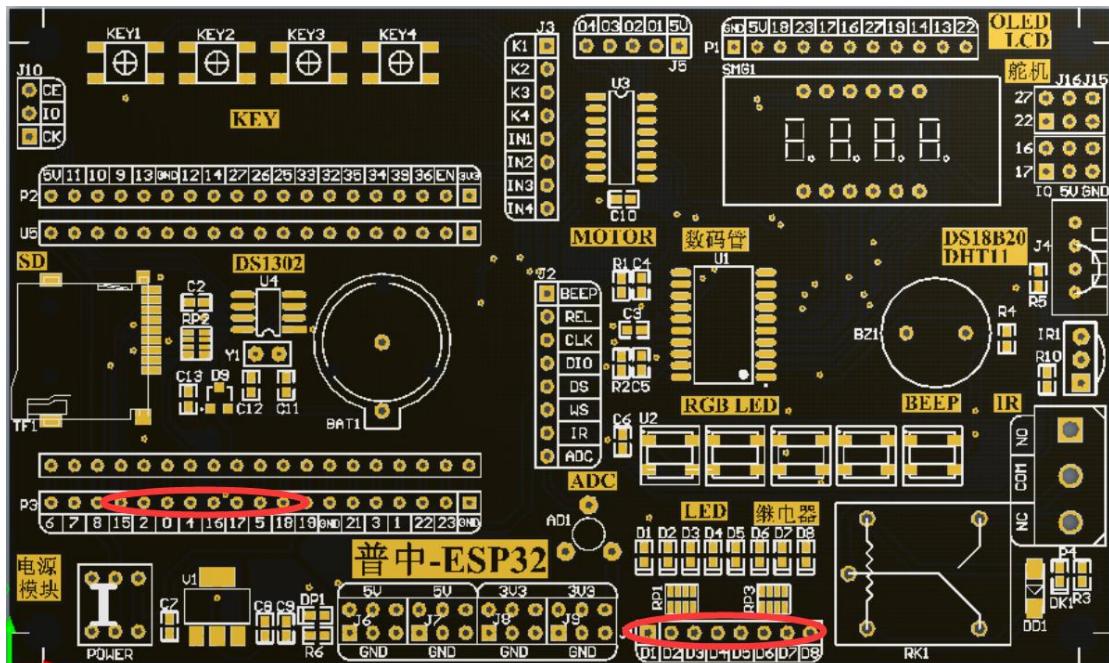
- (1) LED 模块中 D1 指示灯
- (2) ESP32 GPIO

LED 模块电路如下所示:



由图可知，J1 端子为 LED 的控制端，要使 LED 点亮，只需给 J1 端子一个高电平，因此可使用导线将 ESP32 的 IO 口与 J1 端子连接，通过 ESP32 的 GPIO 输出高电平即可控制 LED 点亮。

本章实验使用 ESP32 的 I015, 2, 0, 4, 16, 17, 5, 18 引脚，接线如下所示：



接线说明：

LED 模块-->ESP32 IO

(D1-D8)-->(15, 2, 0, 4, 16, 17, 5, 18)

## 4.3 软件设计

### 4.3.1 点亮第一个 LED 实验

下面我们打开“\4--实验程序\1--MicroPython 实验\1--基础实验\1-点亮第一个 LED 实验”程序，控制代码全部都在 main.py 中，代码如下：

```
main.py x
1 ...
2 深圳市普中科技有限公司 (PRECHIN 普中)
3 技术支持: www.prechin.net
4
5 实验名称: 点亮第一个LED
6 接线说明: LED模块-->ESP32 IO
7 D1-->15
8
9 实验现象: 程序下载成功后, D1指示灯点亮
10 注意事项:
11 ...
12 ...
13
14 #导入Pin模块
15 from machine import Pin
16
17 led1=Pin(15,Pin.OUT)#构建led1对象, GPIO15输出
18 led1.value(1)#使IO15输出高电平, 点亮LED
19
20
```

### 4.3.2 LED 闪烁实验

下面我们打开“\4--实验程序\1--MicroPython 实验\1--基础实验\2-LED 闪烁实验”程序，控制代码全部都在 main.py 中，代码如下：

```
main.py x
1 ...
2 深圳市普中科技有限公司 (PRECHIN 普中)
3 技术支持: www.prechin.net
4
5 实验名称: LED闪烁实验
6 接线说明: LED模块-->ESP32 IO
7 D1-->15
8
9 实验现象: 程序下载成功后, D1指示灯闪烁
10 注意事项:
11 ...
12 |
13 |
14
15 from machine import Pin #导入Pin模块
16 import time #导入time模块
17
18 led1=Pin(15,Pin.OUT)#构建led1对象, GPIO15输出
19
20 #循环
21 while True:
22     led1.value(1)#使IO15输出高电平, 点亮LED
23     time.sleep(0.5)#延时0.5秒
24     led1.value(0)#使IO15输出低电平, 熄灭LED
25     time.sleep(0.5)
26
```

Python 中用到延时，可使用 time 模块，time 模块中常用的几个延时函数使用如下：

```
import time

time.sleep(1)          # 延时 1 秒
time.sleep(0.5)        # 延时 0.5 秒
time.sleep_ms(500)      # 延时 500 毫秒
time.sleep_us(500)      # 延时 500 微秒

start=time.ticks_ms()    # 获取毫秒计时器开始值
# 计算从上电开始到当前时间的差值
delta=time.ticks_diff(time.ticks_ms(),start)
```

### 4.3.3 LED 流水灯实验

下面我们打开“\4--实验程序\1--MicroPython 实验\1--基础实验\3-LED 流水灯实验”程序，控制代码全部都在 main.py 中，代码如下：

```
main.py < ...>
1 ...
2 深圳市普中科技有限公司 (PRECHIN 普中)
3 技术支持: www.prechin.net
4
5 实验名称: LED流水灯实验
6 接线说明: LED模块-->ESP32 IO
7 (D1-D8)-->(15,2,0,4,16,17,5,18)
8
9 实验现象: 程序下载成功后, LED模块D1-D8指示灯依次点亮后依次熄灭
10 注意事项:
11 ...
12 ...
13 #导入Pin模块
14 from machine import Pin
15 import time
16
17 led_pin=[15,2,0,4,16,17,5,18] #定义LED控制引脚
18 leds=[] #定义leds列表, 保存LED管脚配置对象
19 for i in range(8): #循环8次, 0-7
20     leds.append(Pin(led_pin[i],Pin.OUT)) #给leds列表添加对象
21 # leds=[Pin(led_pin[i],Pin.OUT) for i in range(0,8)]
22
23
24 #程序入口
25 if __name__=="__main__":
26     #LED全熄灭
27     for n in range(8):
28         leds[n].value(0)
29
30     while True:
31         #LED逐个点亮
32         for n in range(8):
33             leds[n].value(1)
34             time.sleep(0.05)
35         #LED逐个熄灭
36         for n in range(8):
37             leds[n].value(0)
38             time.sleep(0.05)
39
40
```

range() 函数是 Python 内置函数，创建一个整数列表，一般用于 for 循环当中。

函数原型: range(start, end, scan)

参数含义:

start: 计数从 start 开始，默认是从 0 开始。例如 range (5) 等价于 range (0, 5);

end: 计数到 end 结束, 但不包括 end。例如: range (0, 5) 是[0, 1, 2, 3, 4]没有 5;

scan: 每次跳跃的间距, 默认为 1。例如: range (0, 5) 等价于 range(0, 5, 1)。

在代码中, 可以看到有这么一条语句:

```
if __name__ == "__main__":
```

这句话是什么意思呢? 只要你创建了一个模块 (一个.py 文件), 这个模块就有一个内置属性 name 生成, 该模块的 name 的值取决于如何应用这个模块。

简单来说就是, 如果你直接运行该模块, 那么 \_\_name\_\_ == "\_\_main\_\_";

如果你 import 一个模块, 那么模块 name 的值通常为模块文件名。

如果模块是被直接运行的, 则代码块被运行, 如果模块被 import, 则代码块不被运行。

通俗的理解也就是 \_\_name\_\_ == '\_\_main\_\_': 假如你叫小明.py, 在朋友眼中, 你是小明 (\_\_name\_\_ == '小明'); 在你自己眼中, 你是你自己 (\_\_name\_\_ == '\_\_main\_\_')。

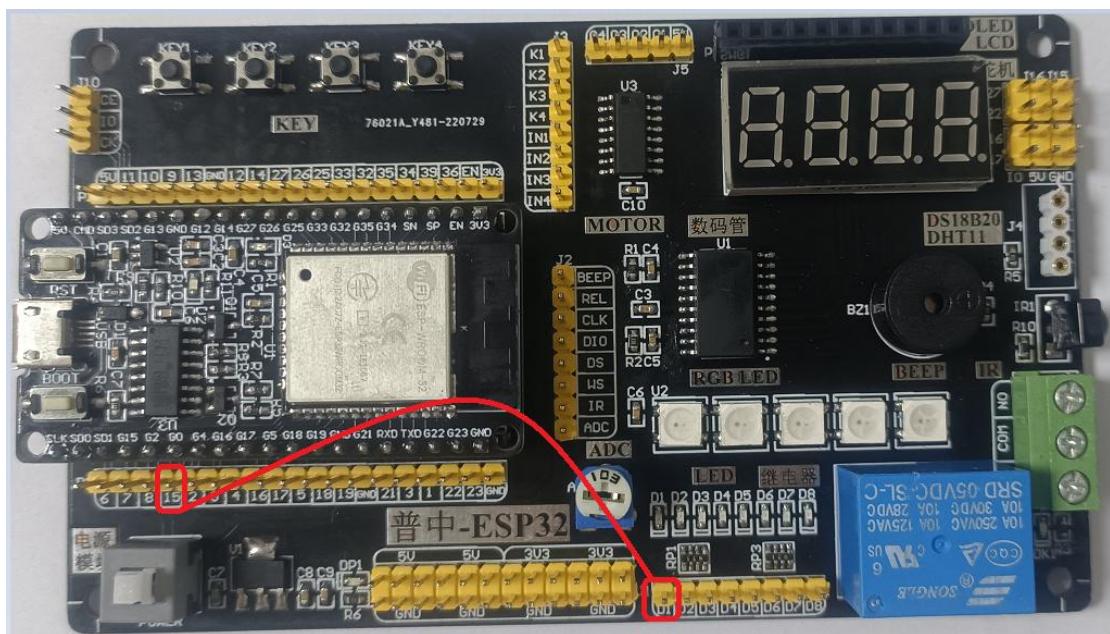
if \_\_name\_\_ == '\_\_main\_\_' 的意思是: 当.py 文件被直接运行时, if \_\_name\_\_ == '\_\_main\_\_' 之下的代码块将被运行; 当.py 文件以模块形式被导入时, if \_\_name\_\_ == '\_\_main\_\_' 之下的代码块不被运行。

通常我们习惯把这个作为程序入口函数, 就像 C、C++、JAVA 等语言一样, 从 main 函数开始执行。

## 4.4 实验现象

### 4.4.1 点亮第一个 LED 实验

下载程序前, 按照如下接线:



**接线说明：**

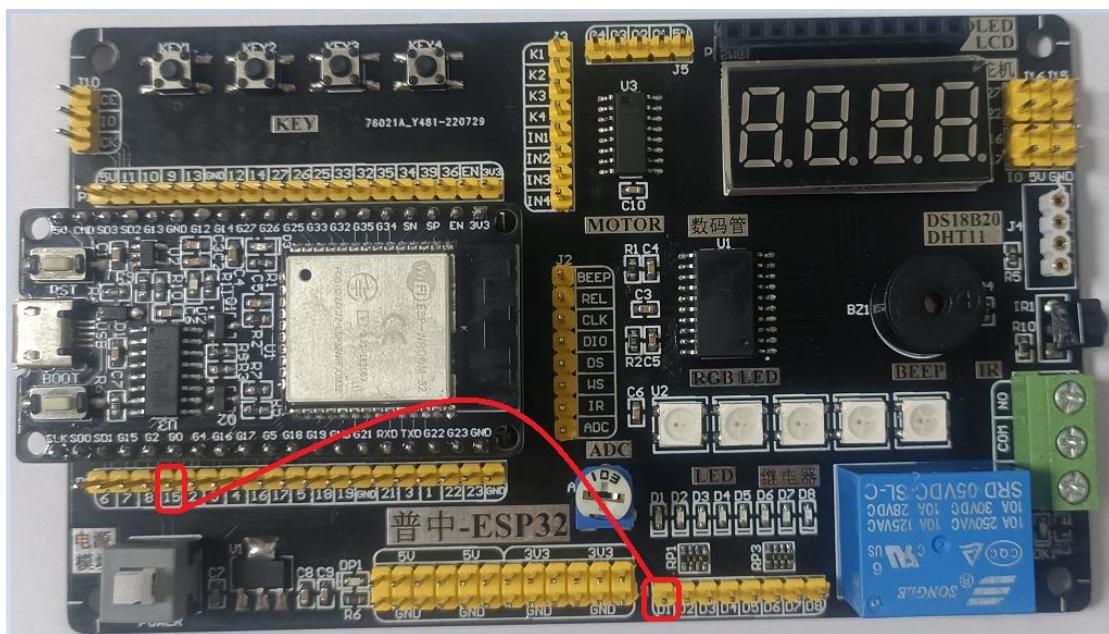
**LED模块-->ESP32 10**

**D1-->15**

将程序下载到开发板内（可参考“2.2.5 程序下载运行”章节），可以看到 LED 模块中 D1 指示灯点亮。

#### 4.4.2 LED 闪烁实验

下载程序前，按照如下接线：



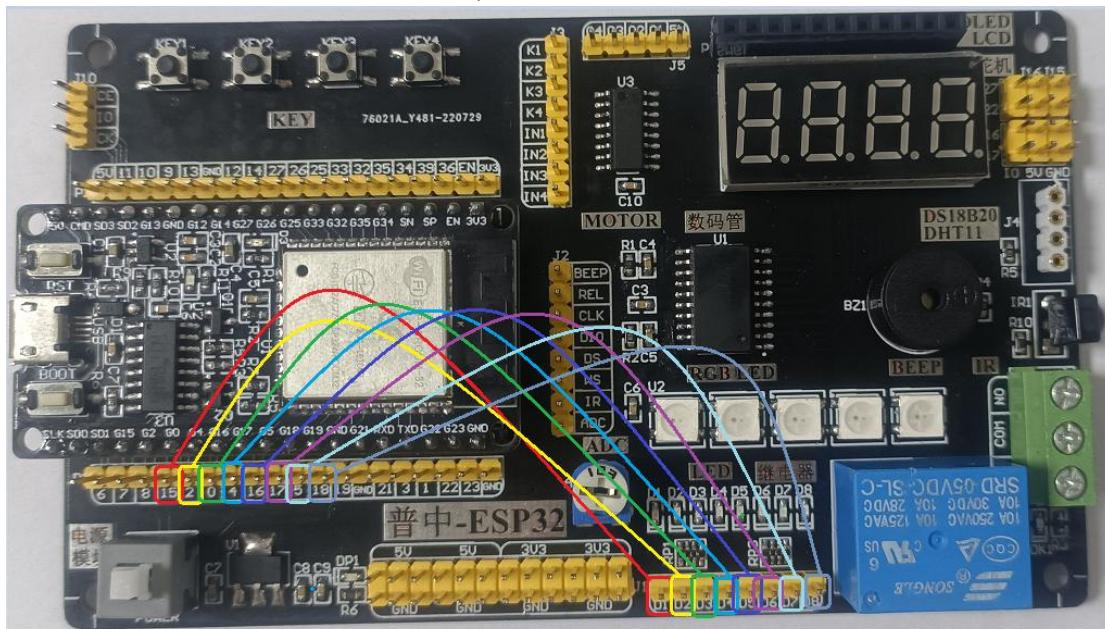
**接线说明：**

LED模块-->ESP32 10  
D1-->15

将程序下载到开发板内（可参考“2.2.5 程序下载运行”章节），可以看到 LED 模块中 D1 指示灯闪烁。

#### 4.4.3 LED 流水灯实验

下载程序前，按照如下接线：



接线说明：

LED模块-->ESP32 IO

D1-D8-->15, 2, 0, 4, 16, 17, 5, 18

将程序下载到开发板内（可参考“2.2.5 程序下载运行”章节），可以看到 LED 模块 D1-D8 指示灯依次点亮后依次熄灭。

## 课后作业

## 第 5 章 蜂鸣器实验

前面章节我们已经介绍了使用 MicroPython 控制 ESP32 的 IO 口输出高低电平，本章我们通过另外一个实验来讲述 ESP32 单片机 IO 口的输出。通过单片机的一个 IO 口控制板载无源蜂鸣器，实现蜂鸣器控制。本章分为如下几部分内容：

- 5.1 实验介绍
- 5.2 硬件设计
- 5.3 软件设计
- 5.4 实验现象

## 5.1 实验介绍

### 5.1.1 实验简介

蜂鸣器分有源和无源，有源蜂鸣器控制相对简单，只需得电即可发出声音；而无源蜂鸣器需要连续输出一定频率的脉冲信号才能使蜂鸣器发出声音，即循环让 IO 口输出高低电平，最佳发声频率在 1.5K-5KHz 之间，当然其它频率值也能发出声音。改变脉冲信号的频率和占空比可改变声音的音调和音量。这样我们就可以使用板载无源蜂鸣器发出各种美妙的音乐。后期也可以制作自己的音乐盒。

### 5.1.2 实验目的

让 ESP32 的 IO 口输出一个 2KHz 频率的脉冲信号控制板载无源蜂鸣器发声。

### 5.1.3 MicroPython 函数使用

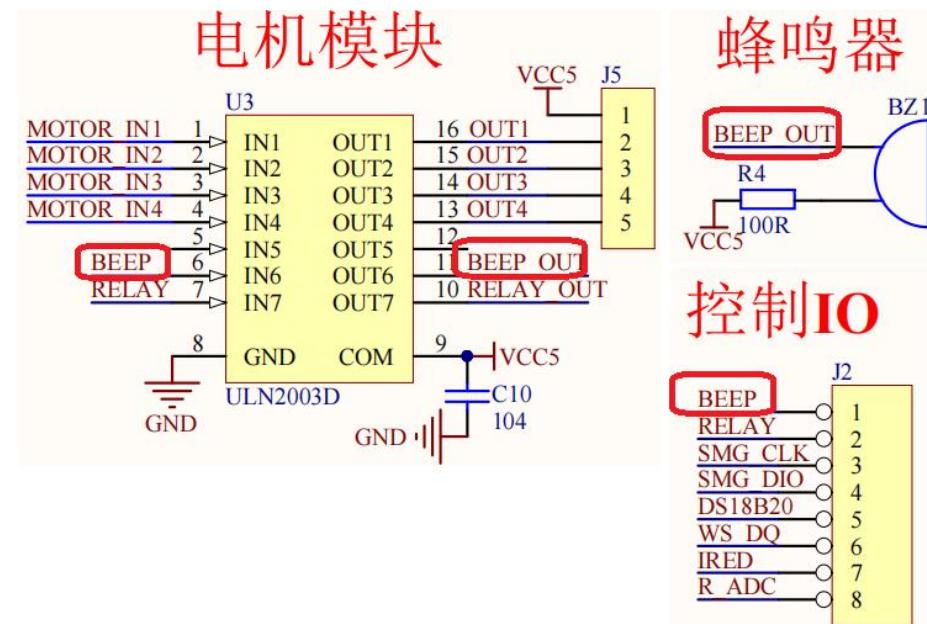
MicroPython 中可使用 `machine` 模块中的 `Pin` 模块对 GPIO 输出控制。相关函数使用方法在前面章节已介绍，此处省略。

## 5.2 硬件设计

本实验使用到硬件资源如下：

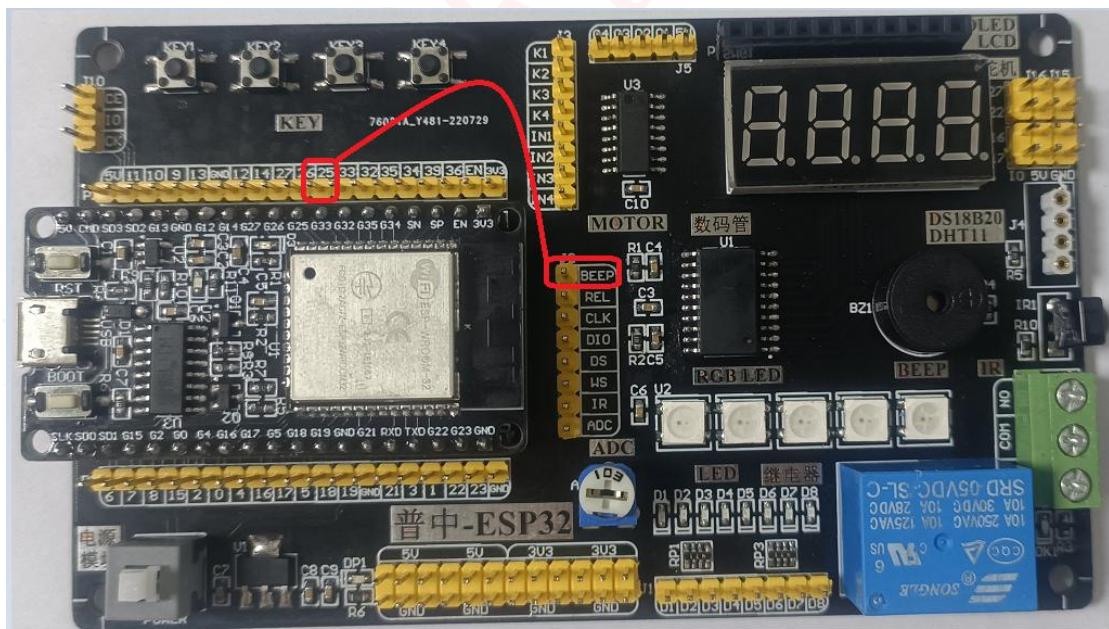
- (1) 蜂鸣器模块
- (2) ESP32 GPIO

蜂鸣器模块电路如下所示：



由图可知，J2 端子的 BEEP 脚为蜂鸣器控制端，要使蜂鸣器发声，只需给 J2 端子的 BEEP 脚输出一定频率脉冲信号，因此可使用导线将 ESP32 的 IO 口与 J2 端子的 BEEP 脚连接，通过 ESP32 的 GPIO 输出一定频率的脉冲信号即可。

本章实验使用 ESP32 的 IO25 引脚，接线如下所示：



#### 接线说明：

蜂鸣器模块-->ESP32 IO

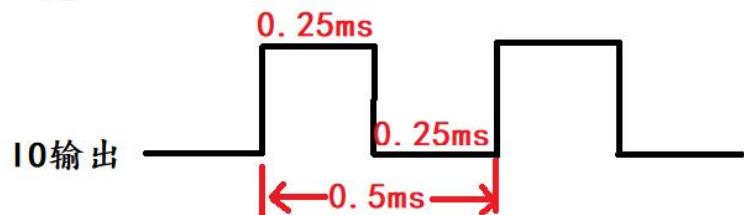
BEEP-->25

## 5.3 软件设计

下面我们打开“\4--实验程序\1--MicroPython 实验\1--基础实验\4-蜂鸣器实验”程序，控制代码全部都在 main.py 中，代码如下：

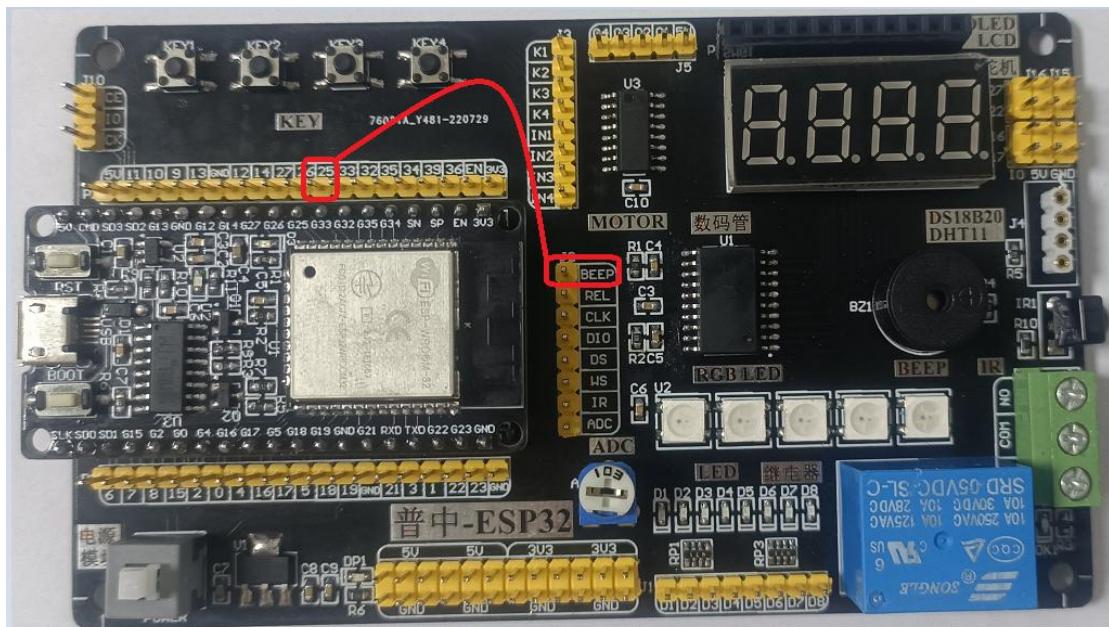
```
main.py x
1 ...
2 深圳市普中科技有限公司 (PRECHIN 普中)
3 技术支持: www.prechin.net
4
5 实验名称: 蜂鸣器实验
6 接线说明: 蜂鸣器模块-->ESP32 IO
7 (BEEP)-->(25)
8
9 实验现象: 程序下载成功后, BEEP模块发出声音
10 注意事项:
11 ...
12 ...
13
14 #导入Pin模块
15 from machine import Pin
16 import time
17
18 #定义蜂鸣器控制对象
19 beep=Pin(25,Pin.OUT)
20
21 #程序入口
22 if __name__=="__main__":
23     i=0
24     while True:
25         i=not i #非运算
26         beep.value(i)
27         time.sleep_us(250) #脉冲频率为2KHz
```

频率为 2KHz，则周期为 0.5ms，若占空比为 50%，则高低电平分别输出时间为 0.25ms，即 250us，如下所示：



## 5.4 实验现象

下载程序前，按照如下接线：



接线说明：

蜂鸣器模块-->ESP32 IO

BEEP-->25

将程序下载到开发板内（可参考“2.2.5 程序下载运行”章节），可以听到蜂鸣器发出“滴...”声音。

## 课后作业

## 第 6 章 继电器实验

前面章节我们已经介绍了使用 MicroPython 控制 ESP32 的 IO 口输出高低电平，本章我们通过另外一个实验来讲述 ESP32 单片机 IO 口的输出。通过单片机的一个 IO 口控制板载继电器，实现继电器的开和断控制。本章分为如下几部分内容：

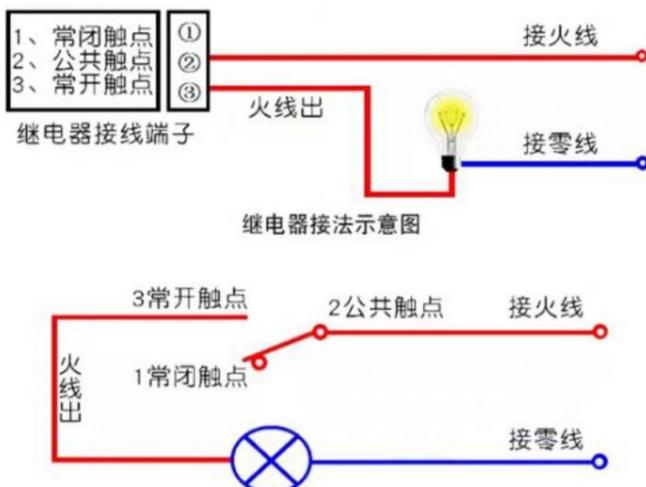
- 6.1 实验介绍
- 6.2 硬件设计
- 6.3 软件设计
- 6.4 实验现象

## 6.1 实验介绍

### 6.1.1 实验简介

继电器是一种电子控制器件，相当于一个开关，接在任意线上，断开状态下线就断开，没导通，闭合状态下线就接在一起，导通。通常使用继电器可实现小电流控制大电流，或者低压控制高压等设备。比如 ESP32 的 GPIO 输出电平是 3.3V，要控制 24V 或者 220V 高压设备，可选择继电器。

继电器接线示意图如下所示：



默认情况下，常闭触点和公共触点是导通连接在一起的，当继电器工作后，公共触点和常闭触点就断开，然后常开触点和公共触点导通连接在一起，这就像开关一样。

### 6.1.2 实验目的

将 D1 指示灯连接到继电器的常开触点上，间隔 1S 控制继电器开和断，从而让 D1 指示灯间隔 1S 亮和灭。

### 6.1.3 MicroPython 函数使用

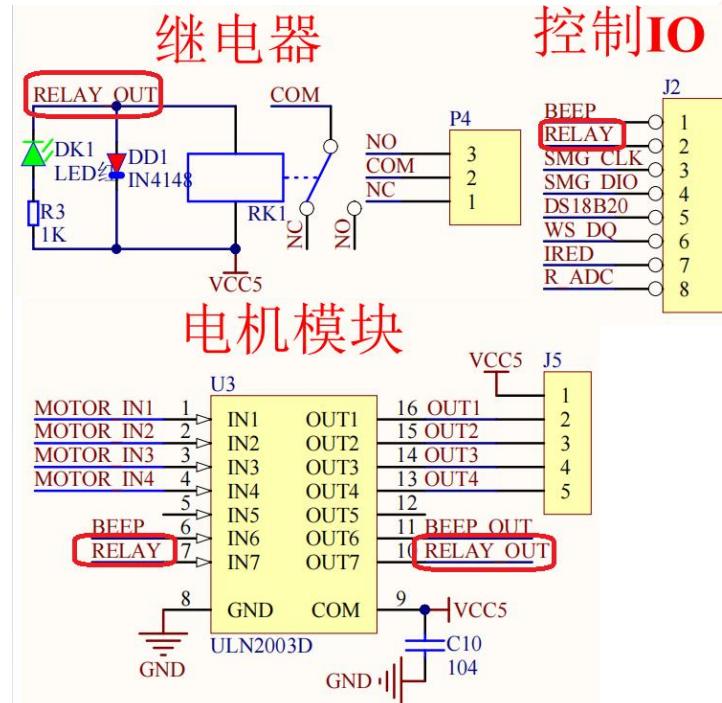
MicroPython 中可使用 machine 模块中的 Pin 模块对 GPIO 输出控制。相关函数使用方法在前面章节已介绍，此处省略。

## 6.2 硬件设计

本实验使用到硬件资源如下：

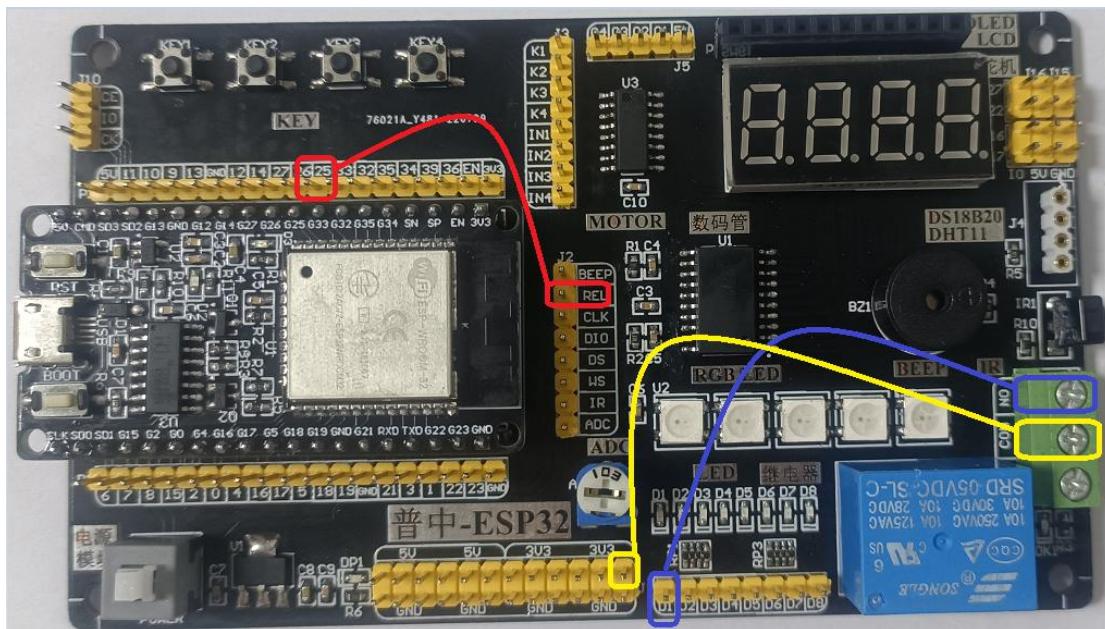
- (1) 继电器模块
- (2) ESP32 GPIO
- (3) D1 指示灯

继电器模块电路如下所示：



由图可知，J2 端子的 RELAY 脚为继电器控制端，要使继电器工作，只需给 J2 端子的 RELAY 脚输出一个高电平，因此可使用导线将 ESP32 的 IO 口与 J2 端子的 RELAY 脚连接，通过 ESP32 的 GPIO 输出高电平即可，若要继电器不工作，则输出低电平。P4 端子为继电器接线端子，COM 为公共触点、NO 为常开触点、NC 为常闭触点。

本章实验使用 ESP32 的 IO25 引脚，接线如下所示：



接线说明：

继电器模块—>ESP32 10

REL—>25

继电器模块输出—>LED模块

COM—>3V3

NO—>D1

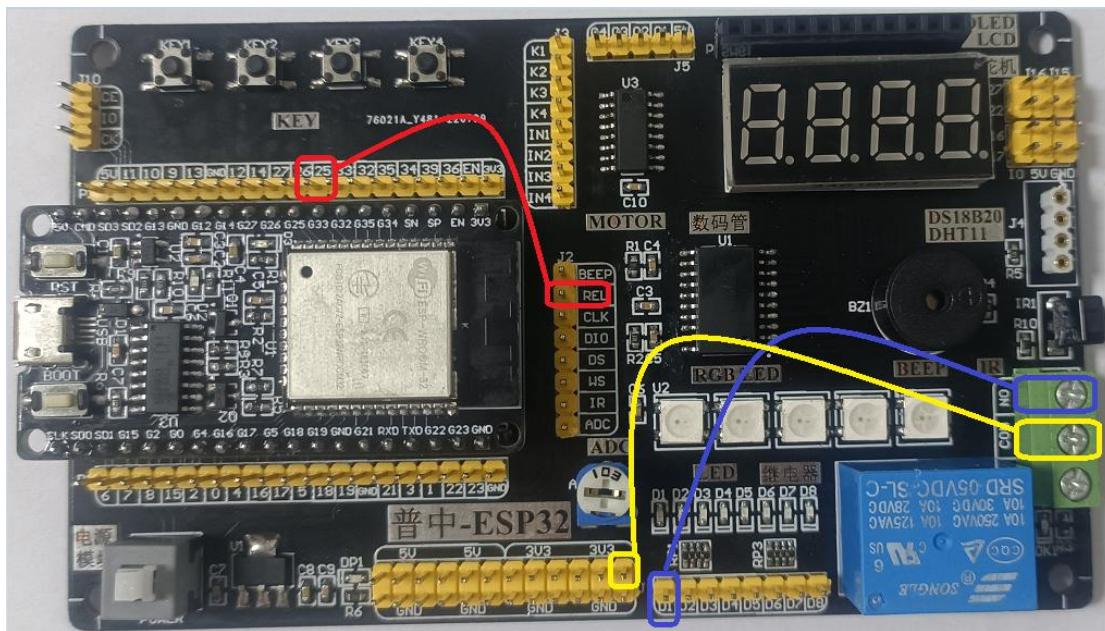
## 6.3 软件设计

下面我们打开“\4--实验程序\1--MicroPython 实验\1--基础实验\5-继电器实验”程序，控制代码全部都在 main.py 中，代码如下：

```
main.py x
1  '''
2  深圳市普中科技有限公司 (PRECHIN 普中)
3  技术支持: www.prechin.net
4
5  实验名称: 继电器实验
6  接线说明: 继电器模块-->ESP32 IO
7      (REL)-->(25)
8
9      继电器模块输出-->LED模块
10     (COM)-->(3V3)
11     (NO)-->(D1)
12
13 实验现象: 程序下载成功后, 继电器模块间隔一定时间吸合断开, 吸合时D1指示灯亮, 断开时灭
14 注意事项: |
15
16 ...
17
18 #导入Pin模块
19 from machine import Pin
20 import time
21
22 #定义继电器控制对象
23 relay=Pin(25,Pin.OUT)
24
25 #程序入口
26 if __name__=="__main__":
27     i=0
28     while True:
29         i=not i
30         relay.value(i)
31         time.sleep(1)
32
```

## 6.4 实验现象

下载程序前, 按照如下接线:



接线说明：

继电器模块—>ESP32 10

REL—>25

继电器模块输出—>LED模块

COM—>3V3

NO—>D1

将程序下载到开发板内（可参考“2.2.5 程序下载运行”章节），可以看到  
D1 指示灯间隔 1S 亮灭，并且继电器吸合断开时也能听到声音。

## 课后作业

## 第 7 章 按键控制实验

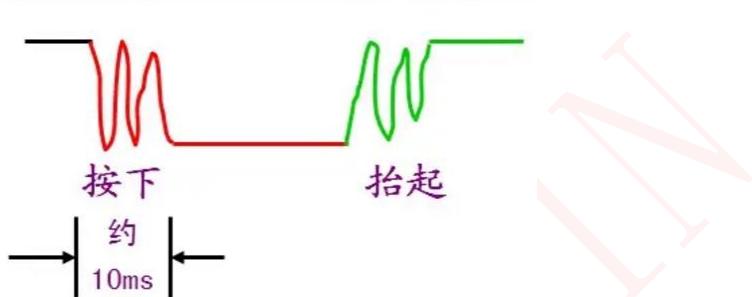
前几章介绍的都是 IO 口输出的使用，本章我们通过按键实验来介绍 IO 口作为输入的使用。本章分为如下几部分内容：

- 7. 1 实验介绍
- 7. 2 硬件设计
- 7. 3 软件设计
- 7. 4 实验现象

## 7.1 实验介绍

### 7.1.1 实验简介

按键是一种电子开关，使用时轻轻按开关按钮就可使开关接通，当松开手时，开关断开。一般机械按键按下和松开时存在抖动情况，如下：



这种抖动可能会影响程序误判，造成严重后果，通常会使用软件延时 10ms 来消抖。例如，当按键按下后，引脚为低电平；所以首先读取引脚电平，若引脚为低电平，则延时 10ms 后再次读取引脚电平，若为低电平，则证明按键已按下。

有了按键输入功能，我们就可以做很多好玩的东西了。

### 7.1.2 实验目的

通过板载按键 K1-K4 控制 LED 模块中的 D1-D4 指示灯亮灭。

### 7.1.3 MicroPython 函数使用

MicroPython 中可使用 machine 模块中的 Pin 模块对 GPIO 输入检测。其构造方法和使用方法如下：

构造函数
<code>KEY=machine.Pin(id,mode,pull)</code>
构建按键对象。 <code>id</code> :引脚编号； <code>mode</code> :输入输出方式； <code>pull</code> :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。

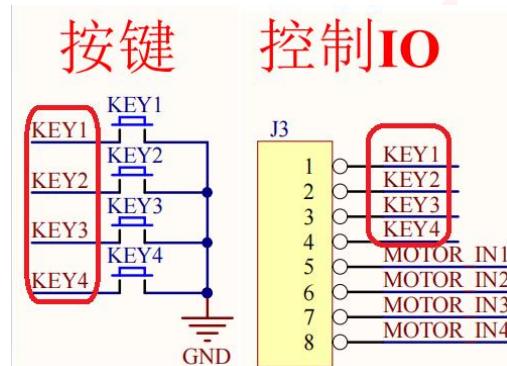
GPIO 输入操作函数和输出是类似的，仅在 mode 参数配置为输入方式和 pull 设置上下拉电阻。

## 7.2 硬件设计

本实验使用到硬件资源如下：

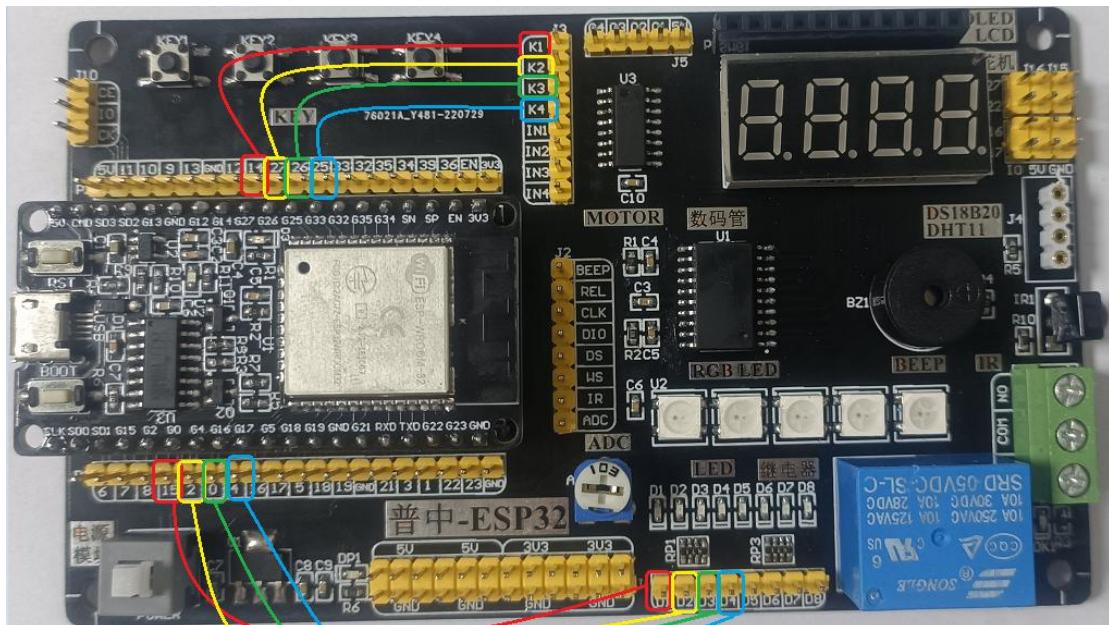
- (1) 按键模块
- (2) ESP32 GPIO
- (3) LED 模块

按键模块电路如下所示：



由图可知，J3 端子的 KEY1–KEY4 脚为按键控制端，要检测按键是否被按下，只需读取 J3 端子的 KEY1–KEY4 脚是否为低电平，因此可使用导线将 ESP32 的 IO 口与 J2 端子的 KEY1–KEY4 脚连接。

本章实验使用 ESP32 的 I014、27、26、25 引脚，接线如下所示：



#### 接线说明:

按键模块-->ESP32 IO  
K1-K4-->14, 27, 26, 25

LED模块-->ESP32 IO  
D1-D4-->15, 2, 0, 4

## 7.3 软件设计

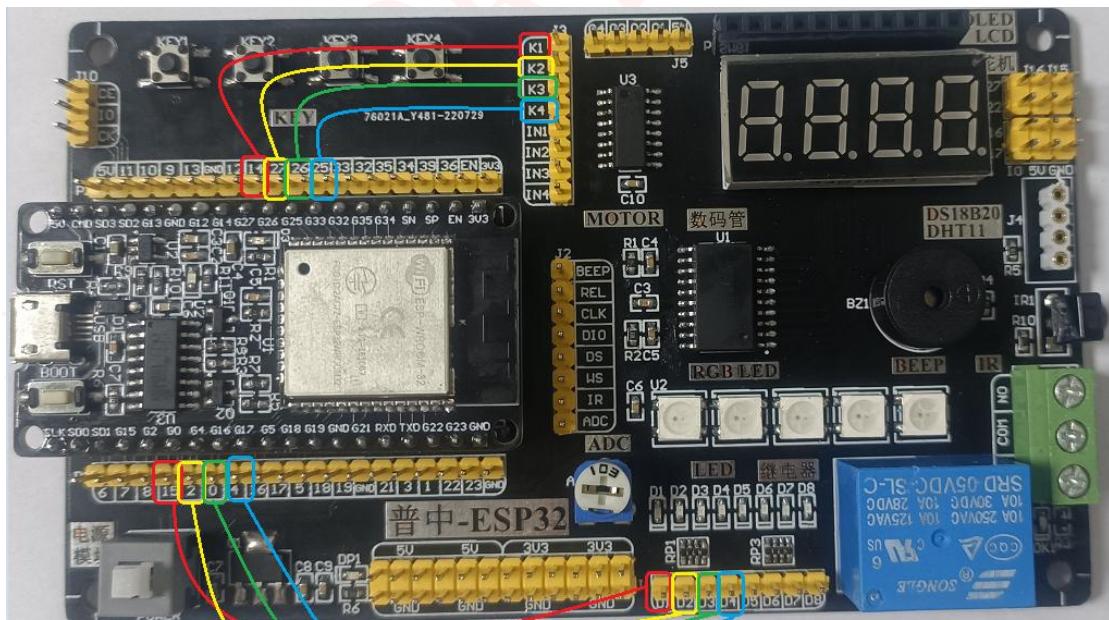
下面我们打开“\4--实验程序\1--MicroPython 实验\1--基础实验\6-按键控制实验”程序，控制代码全部都在 main.py 中，代码如下：

```
main.py x
1  ...
2  深圳市普中科技有限公司（PRECHIN 普中）
3  技术支持: www.prechin.net
4
5  实验名称: 按键控制实验
6  接线说明: 按键模块-->ESP32 IO
7      (K1-K4)-->(14,27,26,25)
8
9      LED模块-->ESP32 IO
10     (D1-D4)-->(15,2,0,4)
11
12  实验现象: 程序下载成功后, 操作K1键控制D1指示灯亮灭; 操作K2键控制D2指示灯亮灭;
13      操作K3键控制D3指示灯亮灭; 操作K4键控制D4指示灯亮灭;
14  注意事项:
15
16  ...
17
18 #导入Pin模块
19 from machine import Pin
20 import time
21
22 #定义按键控制对象
23 key1=Pin(14,Pin.IN,Pin.PULL_UP)
24 key2=Pin(27,Pin.IN,Pin.PULL_UP)
25 key3=Pin(26,Pin.IN,Pin.PULL_UP)
26 key4=Pin(25,Pin.IN,Pin.PULL_UP)
27
28 #定义LED控制对象
29 led1=Pin(15,Pin.OUT)
30 led2=Pin(2,Pin.OUT)
31 led3=Pin(0,Pin.OUT)
32 led4=Pin(4,Pin.OUT)
33
34 #定义按键键值
35 KEY1_PRESS,KEY2_PRESS,KEY3_PRESS,KEY4_PRESS=1,2,3,4
36 key_en=1
37 #按键扫描函数
38 def key_scan():
39     global key_en  #全局变量
40     if key_en==1 and (key1.value()==0 or key2.value()==0 or
41                         key3.value()==0 or key4.value()==0):
42         time.sleep_ms(10) #消抖
43         key_en=0
44         if key1.value()==0:
45             return KEY1_PRESS
46         elif key2.value()==0:
47             return KEY2_PRESS
48         elif key3.value()==0:
49             return KEY3_PRESS
50         elif key4.value()==0:
51             return KEY4_PRESS
```

```
52     elif key1.value()==1 and key2.value()==1 and key3.value()==1 and key4.value()==1:  
53         key_en=1  
54     return 0  
55  
56 #程序入口  
57 if __name__=="__main__":  
58     key=0  
59     i_led1,i_led2,i_led3,i_led4=0,0,0,0 #定义变量，用于LED状态翻转  
60     led1.value(i_led1) #初始化LED，熄灭状态  
61     led2.value(i_led2)  
62     led3.value(i_led3)  
63     led4.value(i_led4)  
64     while True:  
65         key=key_scan() #按键扫描  
66         if key==KEY1_PRESS: #K1键按下  
67             i_led1=not i_led1  
68             led1.value(i_led1)  
69         elif key==KEY2_PRESS: #K2键按下  
70             i_led2=not i_led2  
71             led2.value(i_led2)  
72         elif key==KEY3_PRESS: #K3键按下  
73             i_led3=not i_led3  
74             led3.value(i_led3)  
75         elif key==KEY4_PRESS: #K4键按下  
76             i_led4=not i_led4  
77             led4.value(i_led4)
```

## 7.4 实验现象

下载程序前，按照如下接线：



接线说明：

按键模块-->ESP32 IO  
K1-K4-->14, 27, 26, 25

LED模块-->ESP32 IO  
D1-D4-->15, 2, 0, 4

将程序下载到开发板内（**可参考“2.2.5 程序下载运行”章节**），可以操作 K1 键控制 D1 指示灯亮灭；操作 K2 键控制 D2 指示灯亮灭；操作 K3 键控制 D3 指示灯亮灭；操作 K4 键控制 D4 指示灯亮灭。

## 课后作业

## 第 8 章 直流电机实验

在 ESP32 应用中，电机控制的应用也非常多，比如使用 ESP32 制作智能小车等，本章我们学习使用 ESP32 控制直流电机旋转和停止，至于调速等到后面学习 PWM 即可实现。本章分为如下几部分内容：

- 8.1 实验介绍
- 8.2 硬件设计
- 8.3 软件设计
- 8.4 实验现象

## 8.1 实验介绍

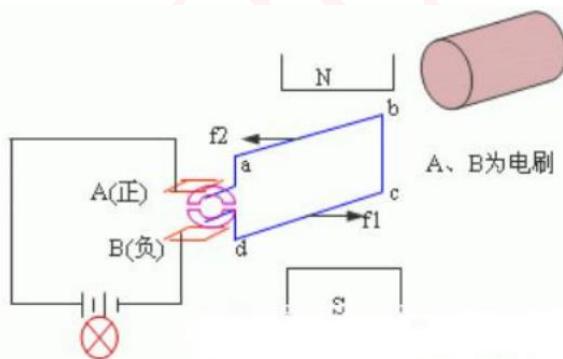
### 8.1.1 实验简介

直流电机是指能将直流电能转换成机械能（直流电动机）或将机械能转换成直流电能（直流发电机）的旋转电机。

直流电机没有正负之分，在两端加上直流电就能工作。开发板配置的直流电机为 5V 直流电机，如下：

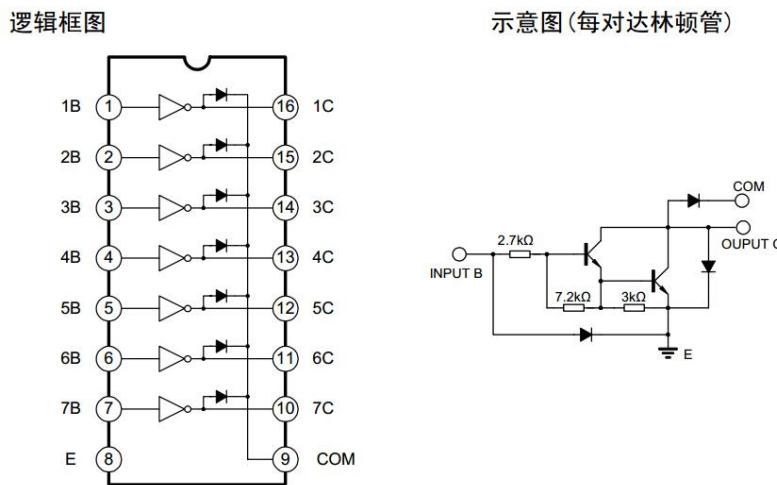


其内部结构如下图所示：



要让直流电机旋转，直接使用 ESP32 的 GPIO 连接肯定是不行的，需要借助驱动模块，比如板载的 ULN2003 模块。它可以让 ESP32 的 GPIO 提供很小的电流就能驱动大电流的设备。

ULN2003 使用非常简单，可以简单理解为一个非门，即输入为高电平，输出则为低电平，输入为低电平，输出则为高电平。ULN2003 结构图如下：



从图中可知，1B 输入对应 1C 输出，2B 输入对应 2C 输出，因此类推。注意：因为 ULN2003 的输出是集电极开路，ULN2003 要输出高电平，必须在输出口外接上拉电阻。这也就能解释在后面连接直流电机时为什么不能直接将 ULN2003 的 2 个输出口接电机线，而必须一根线接电源，另一个才接 ULN2003 输出口。

### 8.1.2 实验目的

通过板载 ULN2003 驱动直流电机旋转 3S 后停止。

### 8.1.3 MicroPython 函数使用

MicroPython 中可使用 `machine` 模块中的 `Pin` 模块对 GPIO 输出控制。相关函数使用方法在前面章节已介绍，此处省略。

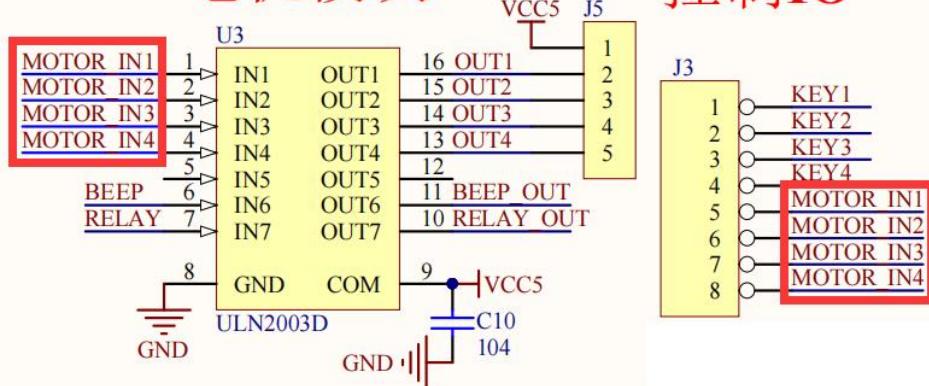
## 8.2 硬件设计

本实验使用到硬件资源如下：

- (1) ULN2003 模块
- (2) ESP32 GPIO
- (3) 5V 直流电机

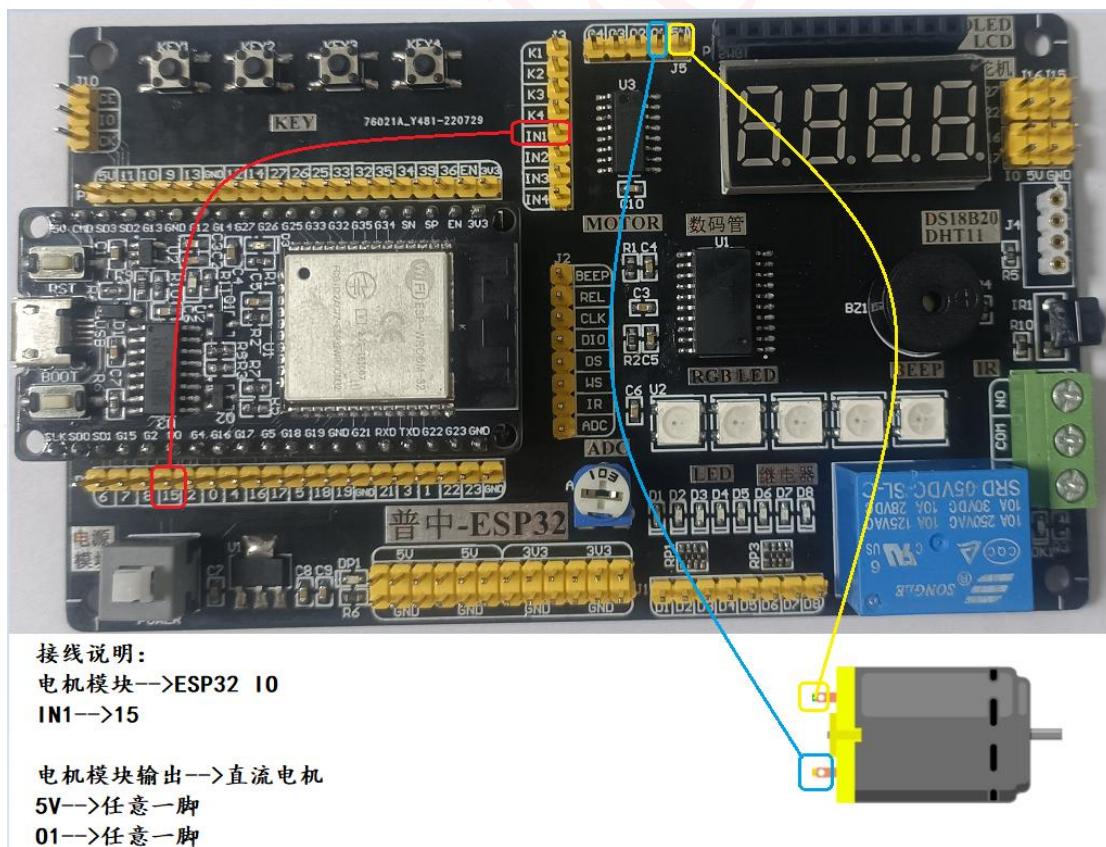
ULN2003 模块电路如下所示：

## 电机模块 控制IO



由图可知，J3 端子的 MOTOR\_IN1–MOTOR\_IN4 脚为 ULN2003 控制端，J5 端子为 ULN2003 的输出端，本实验直流电机的一脚接在 J5 端子的 VCC5，另一脚接在 J5 端子的 OUT1。要使直流电机旋转，只需给 MOTOR\_IN1 脚输出高电平，如输出低电平，则电机停止。因此可使用导线将 ESP32 的 IO 口与 MOTOR\_IN1 脚连接。

本章实验使用 ESP32 的 IO15 引脚，接线如下所示：



## 8.3 软件设计

下面我们打开“\4--实验程序\1--MicroPython 实验\1--基础实验\7-直流电机实验”程序，控制代码全部都在 main.py 中，代码如下：

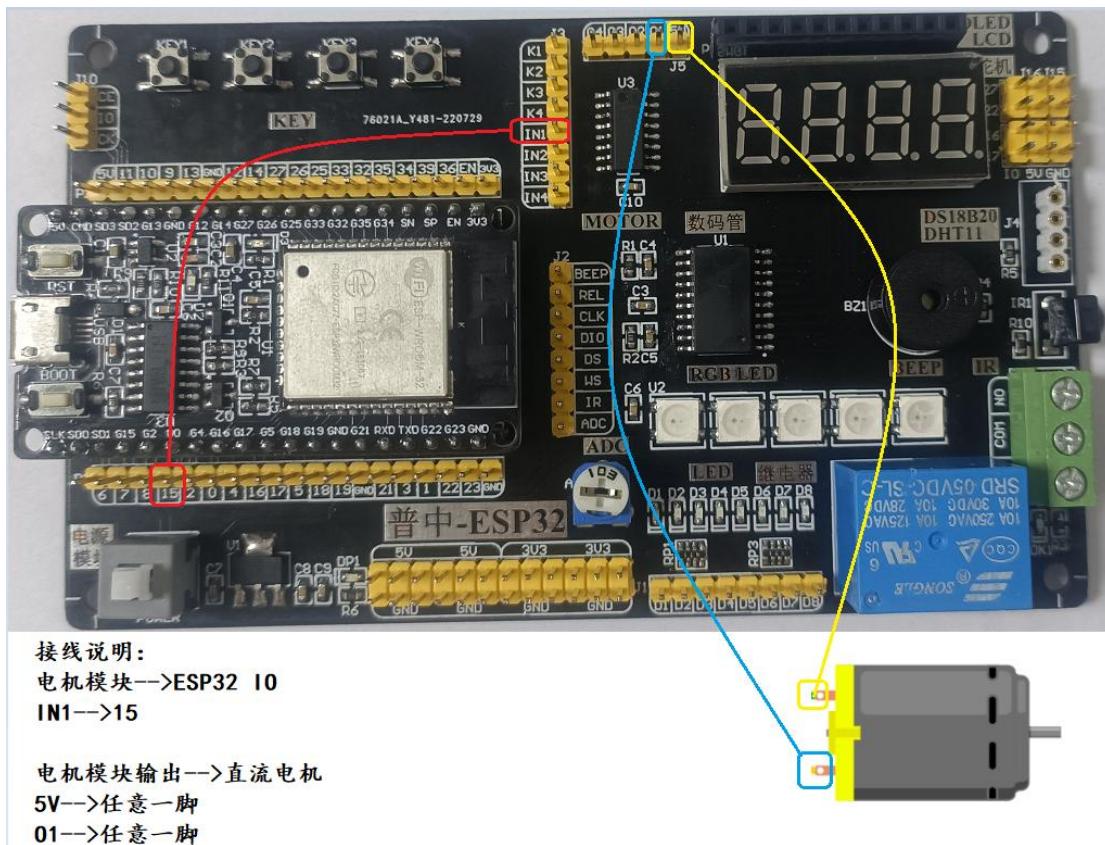
```
main.py x
1 ...
2 深圳市普中科技有限公司 (PRECHIN 普中)
3 技术支持: www.prechin.net
4
5 实验名称: 直流电机实验
6 接线说明: 电机模块-->ESP32 IO
7 (IN1)-->(15)
8
9     电机模块输出-->直流电机
10    (5V)-->任意一脚
11    (01)-->任意一脚
12
13 实验现象: 程序下载成功后, 直流电机旋转3S后停止
14 注意事项:
15 ...
16 ...
17
18 #导入Pin模块
19 from machine import Pin
20 import time
21
22 #定义直流电机控制对象
23 dc_motor=Pin(15,Pin.OUT,Pin.PULL_DOWN)
24
25
26 #程序入口
27 if __name__=="__main__":
28     dc_motor.value(1) #电机开启
29     time.sleep(3)
30     dc_motor.value(0) #电机停止
31     while True:
32         pass
```

在配置 IO 上下拉时，为了初始状态让电机停止，此时配置为下拉，即 ULN2003 输入为低，输出则为高组态，电机停止。

在 Python 中，pass 是一个空语句，为了保持程序结构的完整性。一般情况下，pass 不做任何事情，被用做占位符。

## 8.4 实验现象

下载程序前，按照如下接线：



### 接线说明：

电机模块—>ESP32 IO

IN1—>15

电机模块输出—>直流电机

5V—>任意一脚

01—>任意一脚

将程序下载到开发板内（[可参考“2.2.5 程序下载运行”章节](#)），可以看到直流电机运行 3S 后停止。直流电机没有方向，若此种接线电机反转，可调换接线管脚即可变为正转方向。

**注意：在电机运行过程中，电机电流消耗过大，可能导致 ESP32 串口无法识别，若要下载其它程序，可重新插拔 USB 接口数据线。**

## 课后作业

## 第 9 章 步进电机实验

前面章节，我们介绍了直流电机的控制，本章将向大家介绍步进电机，步进电机是将电脉冲信号转变为角位移或线位移的开环控制元件。本章我们学习使用 ESP32 控制 28BYJ48 步进电机电机旋转方向和速度。本章分为如下几部分内容：

- 9.1 实验介绍
- 9.2 硬件设计
- 9.3 软件设计
- 9.4 实验现象

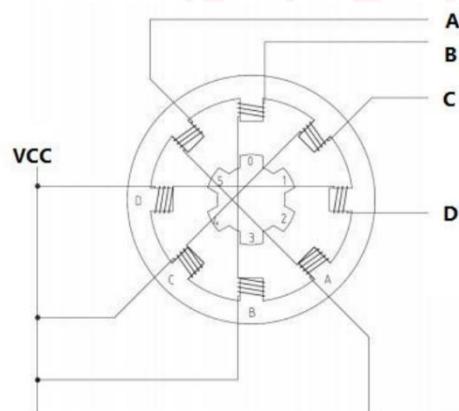
## 9.1 实验介绍

### 9.1.1 实验简介

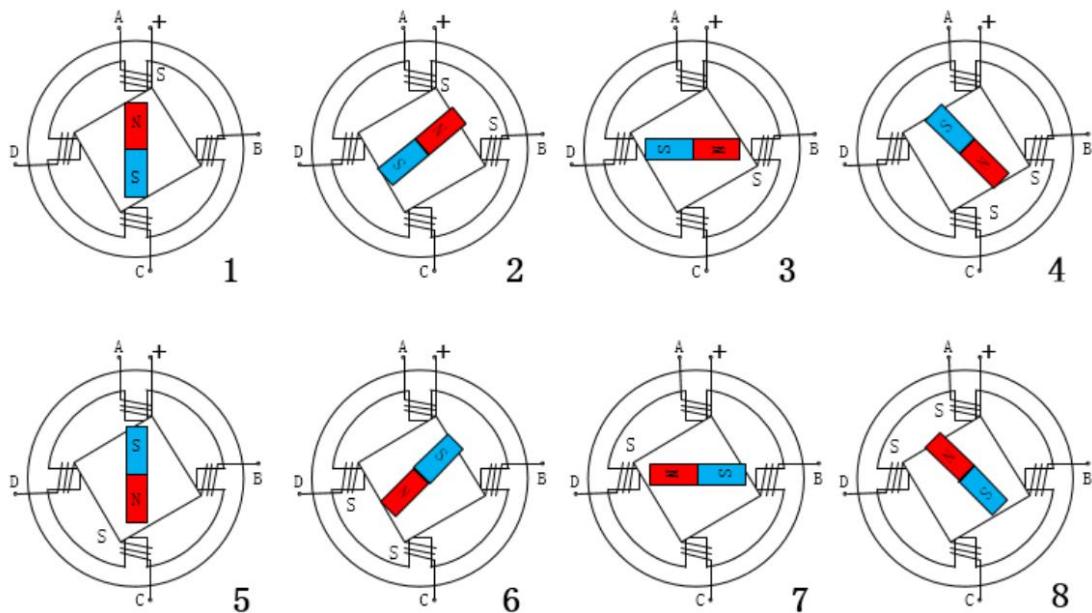
28BYJ48 步进电机自带减速器，为五线四相步进电机，直径为 28mm，实物如下所示：



28BYJ48 电机内部结构等效图如下所示：



步进电机旋转控制图：



28BYJ48 步进电机旋转驱动方式如下表:

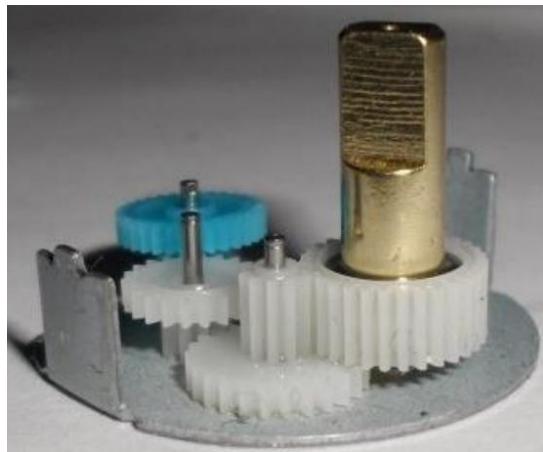
导线颜色	第1步	第2步	第3步	第4步	第5步	第6步	第7步	第8步
VCC 红	5V							
A 蓝	GND	GND						GND
B 粉		GND	GND	GND				
C 黄				GND	GND	GND		
D 橙						GND	GND	GND

28BYJ48 步进电机主要参数如下所示:

电机型号	电压 V	相数	相电阻 ±10%	步进角度	减速比	起动转矩 100PPS g. cm	起动频率 PPS	定位转矩 g. cm	绝缘耐压 VAC/s
28BYJ48	5	4	300欧	5.625/64	1:64	≥300	≥500	≥300	600

在上图中 28BYJ48 步进电机主要参数中可以看到有一个减速比: 1:64, 步进角为 5.625/64 度, 如果需要转动一圈, 那么需要  $360/5.625*64=4096$  个脉冲信号。

减速比这个和之前介绍的直流减速电机有点类似, 所以 28BYJ48 步进电机实际上是: 减速齿轮+步进电机组成, 28BYJ48 步进电机减速齿轮实物图如下所示:



减速齿轮计算方法如下所示：

$$\frac{9}{32} \times \frac{11}{22} \times \frac{9}{26} \times \frac{10}{31} = \frac{8910}{567424} \approx \frac{1}{64}$$



### 9.1.2 实验目的

使用按键控制 28BYJ48 步进电机正反转和加减速。

### 9.1.3 MicroPython 函数使用

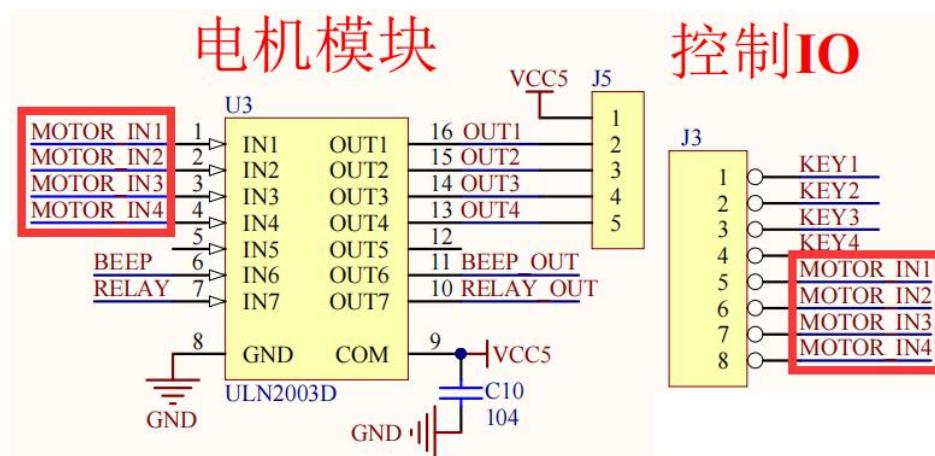
MicroPython 中可使用 machine 模块中的 Pin 模块对 GPIO 输出控制。相关函数使用方法在前面章节已介绍，此处省略。

## 9.2 硬件设计

本实验使用到硬件资源如下：

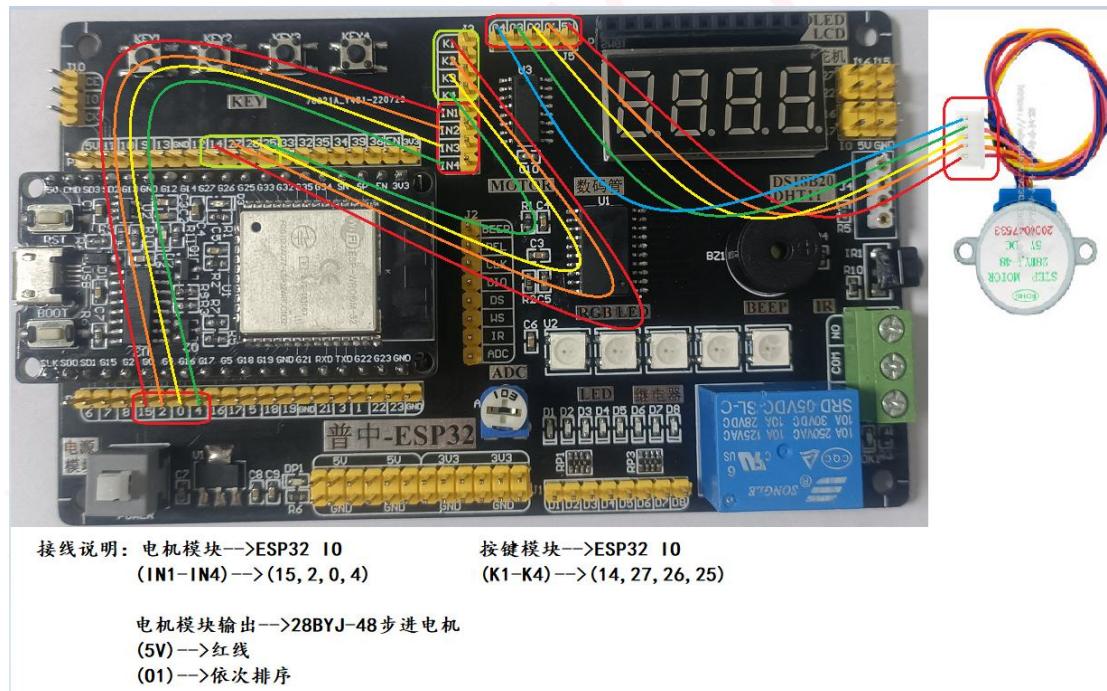
- (1) ULN2003 模块
- (2) ESP32 GPIO
- (3) 28BYJ48 步进电机

ULN2003 模块电路如下所示：



该电路在直流电机实验章节已介绍，此处不再重复。本实验使用 J5 端子与 28BYJ48 步进电机连接。J3 端子的 IN1-IN4 与 ESP32 IO 连接。

本章实验使用 ESP32 的 I015、2、0、14 引脚，接线如下所示：



### 9.3 软件设计

下面我们打开“\4--实验程序\1--MicroPython 实验\1--基础实验\8-步进电机实验”程序，控制代码全部都在 main.py 中，代码如下：

```
main.py x
1 ...
2 深圳市普中科技有限公司（PRECHIN 普中）
3 技术支持: www.prechin.net
4
5 实验名称: 步进电机实验
6 接线说明: 电机模块-->ESP32 IO
7 (IN1-IN4)-->(15,2,0,4)
8
9 电机模块输出-->28BYJ-48步进电机
10 (5V)-->红线
11 (01)-->依次排序
12
13 按键模块-->ESP32 IO
14 (K1-K4)-->(14,27,26,25)
15
16 实验现象: 程序下载成功后, 当按下KEY1键可调节电机旋转方向; 当按下KEY2键, 电机加速;
17 当按下KEY3键, 电机减速;
18 注意事项:
19 ...
20 ...
21
22 #导入Pin模块
23 from machine import Pin
24 import time

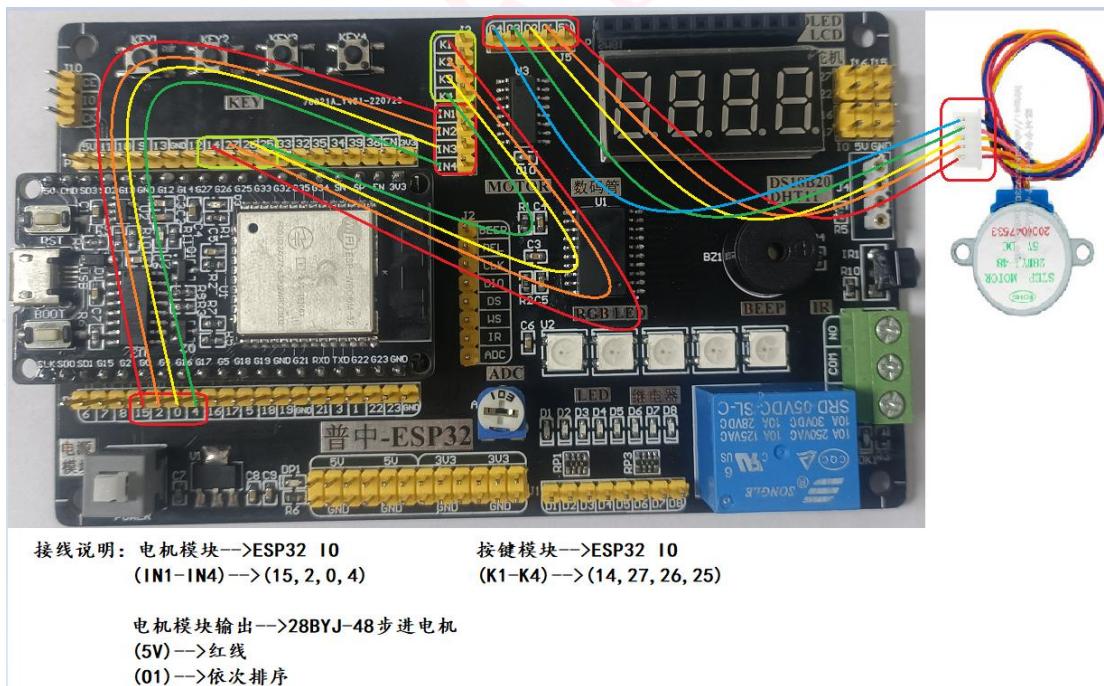
25 #定义按键控制对象
26 key1=Pin(14,Pin.IN,Pin.PULL_UP)
27 key2=Pin(27,Pin.IN,Pin.PULL_UP)
28 key3=Pin(26,Pin.IN,Pin.PULL_UP)
29 key4=Pin(25,Pin.IN,Pin.PULL_UP)
30
31 #定义步进电机控制对象
32 motor_a=Pin(15,Pin.OUT,Pin.PULL_DOWN)
33 motor_b=Pin(2,Pin.OUT,Pin.PULL_DOWN)
34 motor_c=Pin(0,Pin.OUT,Pin.PULL_DOWN)
35 motor_d=Pin(4,Pin.OUT,Pin.PULL_DOWN)
36
37
38 #定义按键键值
39 KEY1_PRESS,KEY2_PRESS,KEY3_PRESS,KEY4_PRESS=1,2,3,4
40 key_en=1
41 #按键扫描函数
42 def key_scan():
43     global key_en
44     if key_en==1 and (key1.value()==0 or key2.value()==0 or
45                         key3.value()==0 or key4.value()==0 ):
46         time.sleep_ms(10)
47         key_en=0
48         if key1.value()==0:
49             return KEY1_PRESS
```

```
51     elif key2.value()==0:
52         return KEY2_PRESS
53     elif key3.value()==0:
54         return KEY3_PRESS
55     elif key4.value()==0:
56         return KEY4_PRESS
57     elif key1.value()==1 and key2.value()==1 and key3.value()==1 and key4.value()==1:
58         key_en=1
59     return 0
60
61
62 #步进电机发送脉冲函数
63 def step_motor_send_pulse(step,fx):
64     temp=step
65     if fx==0:
66         temp=7-step
67     if temp==0:
68         motor_a.value(1)
69         motor_b.value(0)
70         motor_c.value(0)
71         motor_d.value(0)
72     elif temp==1:
73         motor_a.value(1)
74         motor_b.value(1)
75         motor_c.value(0)
76         motor_d.value(0)
77
78     elif temp==2:
79         motor_a.value(0)
80         motor_b.value(1)
81         motor_c.value(0)
82         motor_d.value(0)
83     elif temp==3:
84         motor_a.value(0)
85         motor_b.value(1)
86         motor_c.value(1)
87         motor_d.value(0)
88     elif temp==4:
89         motor_a.value(0)
90         motor_b.value(0)
91         motor_c.value(1)
92         motor_d.value(0)
93     elif temp==5:
94         motor_a.value(0)
95         motor_b.value(0)
96         motor_c.value(1)
97         motor_d.value(1)
98     elif temp==6:
99         motor_a.value(0)
100        motor_b.value(0)
101        motor_c.value(0)
102        motor_d.value(1)
103    elif temp==7:
```

```
103     motor_a.value(1)
104     motor_b.value(0)
105     motor_c.value(0)
106     motor_d.value(1)
107
108 #程序入口
109 if __name__=="__main__":
110     key=0
111     fx1=1
112     STEP_MAXSPEED=1
113     STEP_MINSPEED=5
114     speed1=STEP_MAXSPEED
115     step1=0
116
117     while True:
118         key=key_scan()
119         if key==KEY1_PRESS:
120             fx1=not fx1
121         elif key==KEY2_PRESS:
122             if speed1>STEP_MAXSPEED:
123                 speed1-=1
124             elif key==KEY3_PRESS:
125                 if speed1<STEP_MINSPEED:
126                     speed1+=1
127             step_motor_send_pulse(step1,fx1)
128             step1+=1
129             if step1==8:
130                 step1=0
131
132             time.sleep_ms(speed1)
```

## 9.4 实验现象

下载程序前，按照如下接线：



将程序下载到开发板内（[可参考“2.2.5 程序下载运行”章节](#)），可以按下 KEY1 键调节电机旋转方向；当按下 KEY2 键，电机加速；当按下 KEY3 键，电机减速。

## 课后作业

# 第 10 章 外部中断实验

前面章节我们学习了 ESP32 的按键控制，当时通过查询 GPIO 输入电平来判断按键状态，此种方法占用 CPU 资源，效率不高。本章我们学习外部中断，通过外部中断实现 LED 控制。本章分为如下几部分内容：

- 10.1 实验介绍
- 10.2 硬件设计
- 10.3 软件设计
- 10.4 实验现象

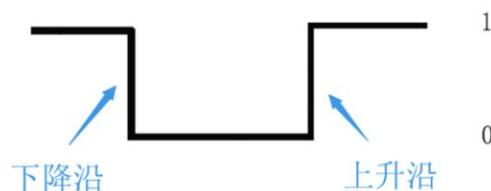
## 10.1 实验介绍

### 10.1.1 实验简介

前面我们在做按键控制实验时，虽然能实现 IO 口输入功能，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，才去执行相关功能。这大大节省了 CPU 的资源，因此中断在实际项目中应用非常普遍。

ESP32 的外部中断有上升沿、下降沿、低电平、高电平触发模式。上升沿和下降沿触发如下：



若将按键对应 IO 配置为下降沿触发，当按键按下后即触发中断，然后在中

断回调函数内执行对应功能。

### 10.1.2 实验目的

使用外部中断功能实现按键控制 LED 亮灭。

### 10.1.3 MicroPython 函数使用

外部中断也是通过 `machine` 模块的 `Pin` 子模块来配置，先来看构造函数和使用方法：

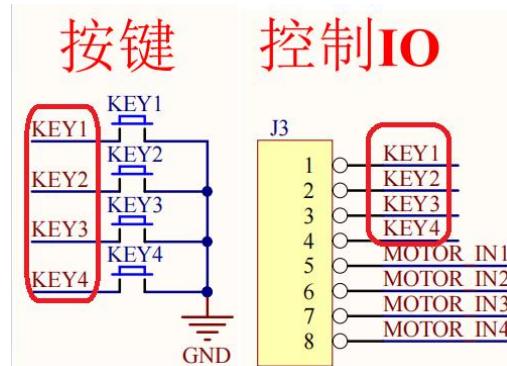
构造函数
<code>KEY=machine.Pin(id,mode,pull)</code>
构建按键对象。 <code>id</code> :引脚编号; <code>mode</code> :输入输出方式; <code>pull</code> :上下拉电阻配置。
使用方法
<code>KEY.irq(handler,trigger)</code>
配置中断模式。 <code>handler</code> :中断执行的回调函数; <code>trigger</code> :触发中断的方式, 共 4 种, 分别是 <code>Pin.IRQ_FALLING</code> (下降沿触发)、 <code>Pin.IRQ_RISING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

## 10.2 硬件设计

本实验使用到硬件资源如下：

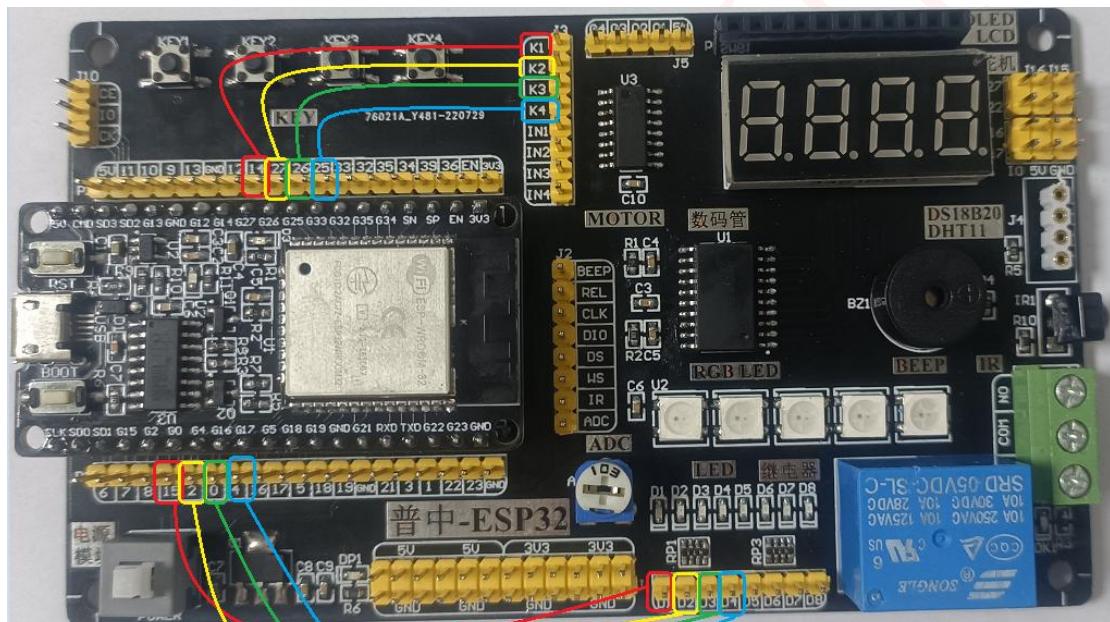
- (1) 按键模块
- (2) ESP32 GPIO
- (3) LED 模块

按键模块电路如下所示：



由图可知，J3 端子的 KEY1–KEY4 脚为按键控制端，要检测按键是否被按下，只需读取 J3 端子的 KEY1–KEY4 脚是否为低电平，因此可使用导线将 ESP32 的 IO 口与 J2 端子的 KEY1–KEY4 脚连接。

本章实验使用 ESP32 的 I014、27、26、25 引脚，接线如下所示：



接线说明：

按键模块-->ESP32 IO  
K1-K4-->14, 27, 26, 25

LED模块-->ESP32 IO  
D1-D4-->15, 2, 0, 4

### 10.3 软件设计

下面我们打开“\4--实验程序\1--MicroPython 实验\1--基础实验\9-外部中断实验”程序，控制代码全部都在 main.py 中，代码如下：

```
main.py *  
1  ...  
2  深圳市普中科技有限公司 (PRECHIN 普中)  
3  技术支持: www.prechin.net  
4  
5  实验名称: 外部中断实验  
6  接线说明: 按键模块-->ESP32 IO  
7      (K1-K4)-->(14,27,26,25)  
8  
9  LED模块-->ESP32 IO  
10     (D1-D4)-->(15,2,0,4)  
11  
12 实验现象: 程序下载成功后, 操作K1键控制D1指示灯亮灭; 操作K2键控制D2指示灯亮灭;  
13 操作K3键控制D3指示灯亮灭; 操作K4键控制D4指示灯亮灭;  
14 注意事项:  
15  
16  ...  
17  
18 #导入Pin模块  
19 from machine import Pin  
20 import time  
21  
22 #定义按键控制对象  
23 key1=Pin(14,Pin.IN,Pin.PULL_UP)  
24 key2=Pin(27,Pin.IN,Pin.PULL_UP)  
25 key3=Pin(26,Pin.IN,Pin.PULL_UP)  
26 key4=Pin(25,Pin.IN,Pin.PULL_UP)  
27  
28 #定义LED控制对象  
29 led1=Pin(15,Pin.OUT)  
30 led2=Pin(2,Pin.OUT)  
31 led3=Pin(0,Pin.OUT)  
32 led4=Pin(4,Pin.OUT)  
33  
34  
35 #定义LED状态  
36 led1_state,led2_state,led3_state,led4_state=1,1,1,1  
37  
38 #KEY1外部中断函数  
39 def key1_irq(key1):  
40     global led1_state  
41     time.sleep_ms(10)  
42     if key1.value()==0:  
43         led1_state=not led1_state  
44         led1.value(led1_state)  
45  
46 #KEY2外部中断函数  
47 def key2_irq(key2):  
48     global led2_state  
49     time.sleep_ms(10)  
50     if key2.value()==0:  
51         led2_state=not led2_state  
52         led2.value(led2_state)  
53  
54 #KEY3外部中断函数  
55 def key3_irq(key3):  
56     global led3_state  
57     time.sleep_ms(10)  
58     if key3.value()==0:  
59         led3_state=not led3_state  
60         led3.value(led3_state)  
61  
62 #KEY4外部中断函数  
63 def key4_irq(key4):  
64     global led4_state  
65     time.sleep_ms(10)  
66     if key4.value()==0:  
67         led4_state=not led4_state  
68         led4.value(led4_state)  
69  
70 #程序入口  
71 if __name__=="__main__":  
72     led1.value(led1_state) #初始化LED, 熄灭状态  
73     led2.value(led2_state)  
74     led3.value(led3_state)  
75     led4.value(led4_state)  
76
```

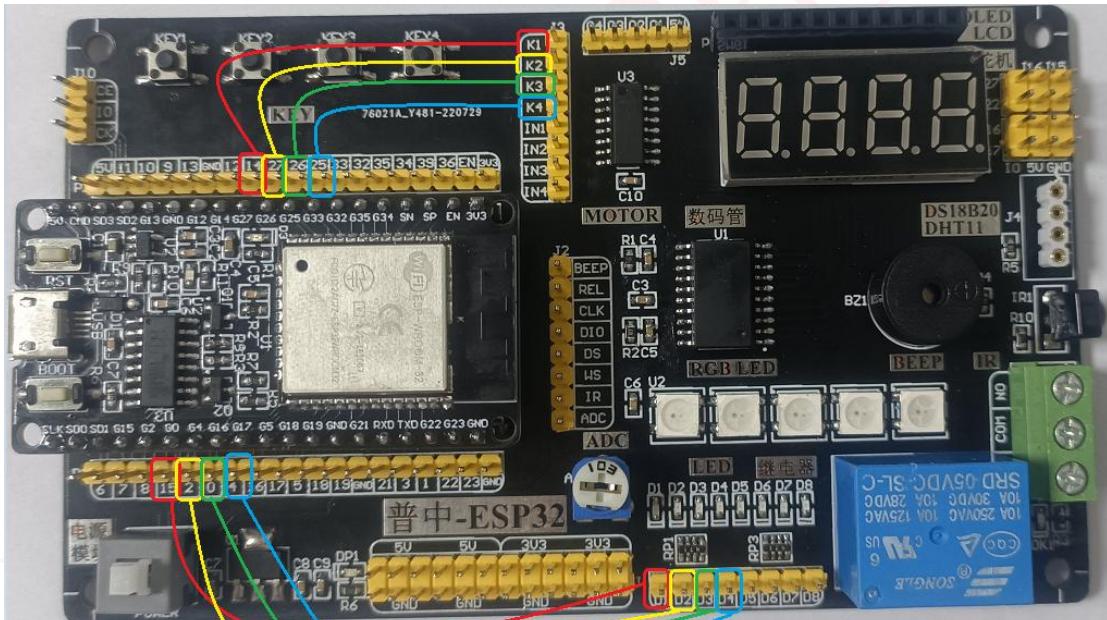
```
77     key1.irq(key1_irq,Pin.IRQ_FALLING) #配置key1外部中断, 下降沿触发
78     key2.irq(key2_irq,Pin.IRQ_FALLING) #配置key2外部中断, 下降沿触发
79     key3.irq(key3_irq,Pin.IRQ_FALLING) #配置key3外部中断, 下降沿触发
80     key4.irq(key4_irq,Pin.IRQ_FALLING) #配置key4外部中断, 下降沿触发
81
82     while True:
83         pass
84
```

以上代码中需要注意的地方：

- 1、`ledx_state` 是全局变量，因此在 `fun` 函数里面用该变量必须添加 `global ledx_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 `keyx_irq` 的时候，需要将 `Pin` 对象 `KEY` 传递进去。

## 10.4 实验现象

下载程序前，按照如下接线：



接线说明：

按键模块—>ESP32 IO

K1-K4—>14, 27, 26, 25

LED模块—>ESP32 IO

D1-D4—>15, 2, 0, 4

将程序下载到开发板内（可参考“2.2.5 程序下载运行”章节），可以操作 K1 键控制 D1 指示灯亮灭；操作 K2 键控制 D2 指示灯亮灭；操作 K3 键控制 D3 指示灯亮灭；操作 K4 键控制 D4 指示灯亮灭。

## 课后作业

# 第 11 章 定时器中断实验

前面章节我们介绍了 ESP32 的外部中断使用，本章继续介绍 ESP32 的定时器功能。本章分为如下几部分内容：

- 11.1 实验介绍
- 11.2 硬件设计
- 11.3 软件设计
- 11.4 实验现象

## 11.1 实验介绍

### 11.1.1 实验简介

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。ESP32 也是这样，通过定时器可以完成各种预设好的任务。ESP32 定时器到达指定时间后也会产生中断，然后在回调函数内执行所需功能，这个和外部中断类似。

### 11.1.2 实验目的

通过定时器让 LED 周期性每秒闪烁 1 次。

### 11.1.3 MicroPython 函数使用

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

ESP32 拥有 4 个定时器。使用 `machine.Timer` 类通过设置 timer ID 号为 0-3。使用方法如下:

```
from machine import Timer

tim0 = Timer(0)
tim0.init(period=5000, mode=Timer.ONE_SHOT, callback=lambda t:print(0))

tim1 = Timer(1)
tim1.init(period=2000, mode=Timer.PERIODIC, callback=lambda t:print(1))
```

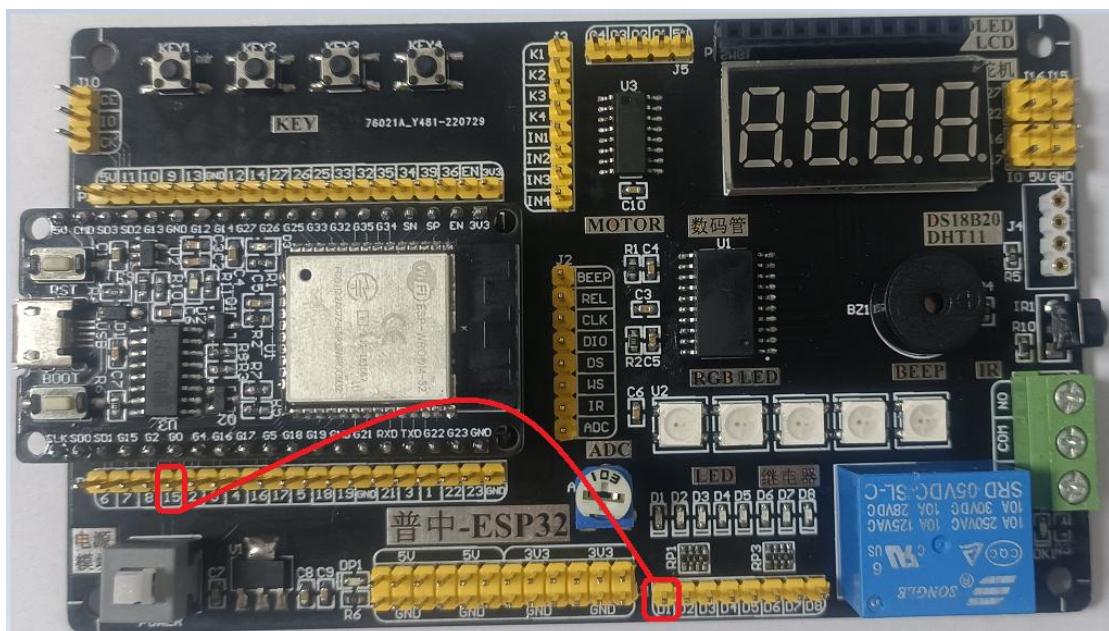
## 11.2 硬件设计

本实验使用到硬件资源如下:

- (1) LED 模块
- (2) ESP32 GPIO

LED 模块电路前面已介绍, 此处不再重复。

本章实验使用 ESP32 的 I015 引脚, 接线如下所示:



接线说明：

LED模块-->ESP32 IO

D1-->15

### 11.3 软件设计

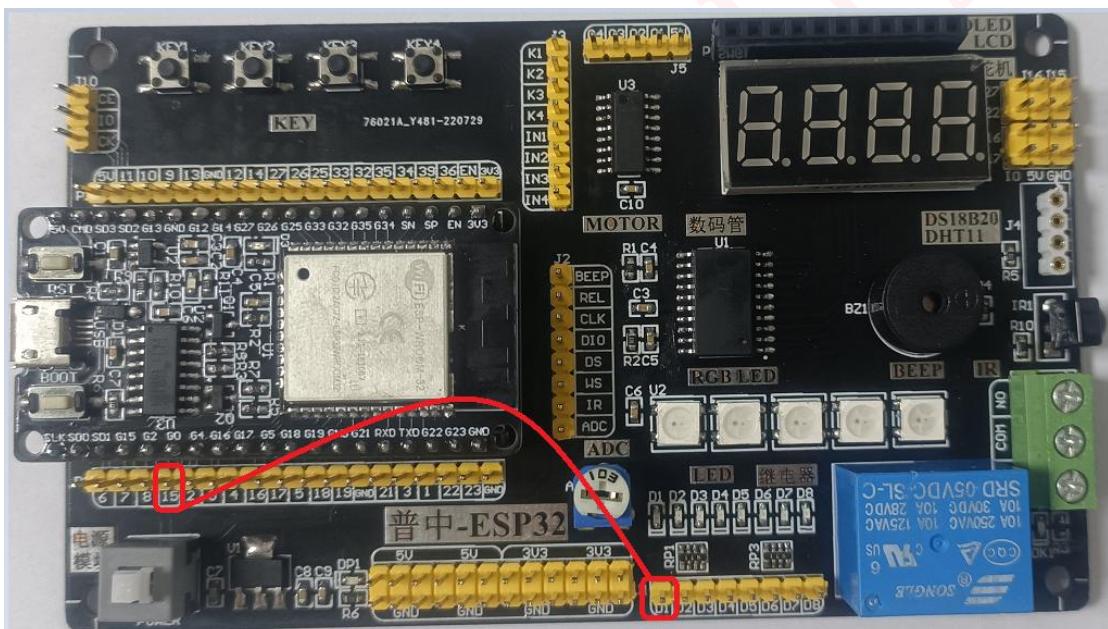
下面我们打开“\4--实验程序\1--MicroPython 实验\1--基础实验\10-定时器中断实验”程序，控制代码全部都在 main.py 中，代码如下：

```
main.py x
1 ...
2 深圳市普中科技有限公司（PRECHIN 普中）
3 技术支持: www.prechin.net
4
5 实验名称: 定时器中断实验
6 接线说明: LED模块-->ESP32 IO
7 (D1)-->(15)
8
9 实验现象: 程序下载成功后, D1指示灯间隔0.5s状态翻转
10 注意事项:
11 ...
12 ...
13
14 #导入Pin模块
15 from machine import Pin
16 from machine import Timer
17
18 #定义LED控制对象
19 led1=Pin(15,Pin.OUT)
20
21 #定义LED状态
22 led1_state=0
23
24
```

```
25 #定时器0中断函数
26 def time0_irq(time0):
27     global led1_state
28     led1_state=not led1_state
29     led1.value(led1_state)
30
31 #程序入口
32 if __name__=="__main__":
33     led1.value(led1_state) #初始化LED, 熄灭状态
34
35 time0=Timer(0) #创建time0定时器对象
36 time0.init(period=500,mode=Timer.PERIODIC,callback=time0_irq)
37
38 while True:
39     pass
40
```

## 11.4 实验现象

下载程序前，按照如下接线：



接线说明：

LED模块-->ESP32 10

D1-->15

将程序下载到开发板内（可参考“2.2.5 程序下载运行”章节），可以看到D1指示灯间隔1S闪烁。

## 课后作业

## 第 12 章 PWM 呼吸灯实验

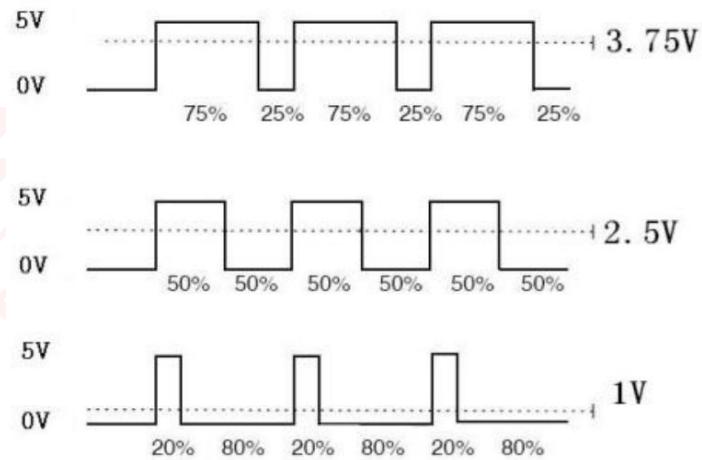
前面我们介绍直流电机时说过可通过 PWM 调速，即使用 PWM 可输出不同频率占空比的脉冲信号，等效为不同的电压输出值。本章通过一个 LED 灯的亮度变化来验证 PWM 不同电压的输出。本章分为如下几部分内容：

- 12.1 实验介绍
- 12.2 硬件设计
- 12.3 软件设计
- 12.4 实验现象

## 12.1 实验介绍

### 12.1.1 实验简介

PWM 是脉冲宽度调制，简称脉宽调制。它是利用微处理器的数字输出来对模拟电路进行控制的一种非常有效的技术。PWM 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。频率固定，改变占空比可改变输出电压，如下所示：



### 12.1.2 实验目的

通过调节 PWM 占空比，控制 LED 指示灯亮度变化，实现呼吸灯效果，即由暗

变亮，再由亮变暗的循环过程。

### 12.1.3 MicroPython 函数使用

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。

PWM 在 machine 的 PWM 模块中，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
<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1-40MHz 之间， freq 为空时表示获取当前频率值。
<pre>pwm25.duty(duty)</pre>
设置占空比。duty:占空比在 0-1023 之间， duty 为空时表示获取当前占空比值。
<pre>pwm25.deinit()</pre>
关闭 PWM。

PWM 使用方法如下：

```
from machine import Pin, PWM

pwm0 = PWM(Pin(0))          # create PWM object from a pin
freq = pwm0.freq()           # get current frequency (default 5kHz)
pwm0.freq(1000)              # set PWM frequency from 1Hz to 40MHz

duty = pwm0.duty()           # get current duty cycle, range 0-1023 (default 512, 50%)
pwm0.duty(256)                # set duty cycle from 0 to 1023 as a ratio duty/1023, (now 25%)

duty_u16 = pwm0.duty_u16()    # get current duty cycle, range 0-65535
pwm0.duty_u16(2**16*3//4)     # set duty cycle from 0 to 65535 as a ratio duty_u16/65535, (now 75%)

duty_ns = pwm0.duty_ns()      # get current pulse width in ns
pwm0.duty_ns(250_000)         # set pulse width in nanoseconds from 0 to 1_000_000_000/freq, (now 25%)

pwm0.deinit()                 # turn off PWM on the pin

pwm2 = PWM(Pin(2), freq=20000, duty=512) # create and configure in one go
print(pwm2)                      # view PWM settings
```

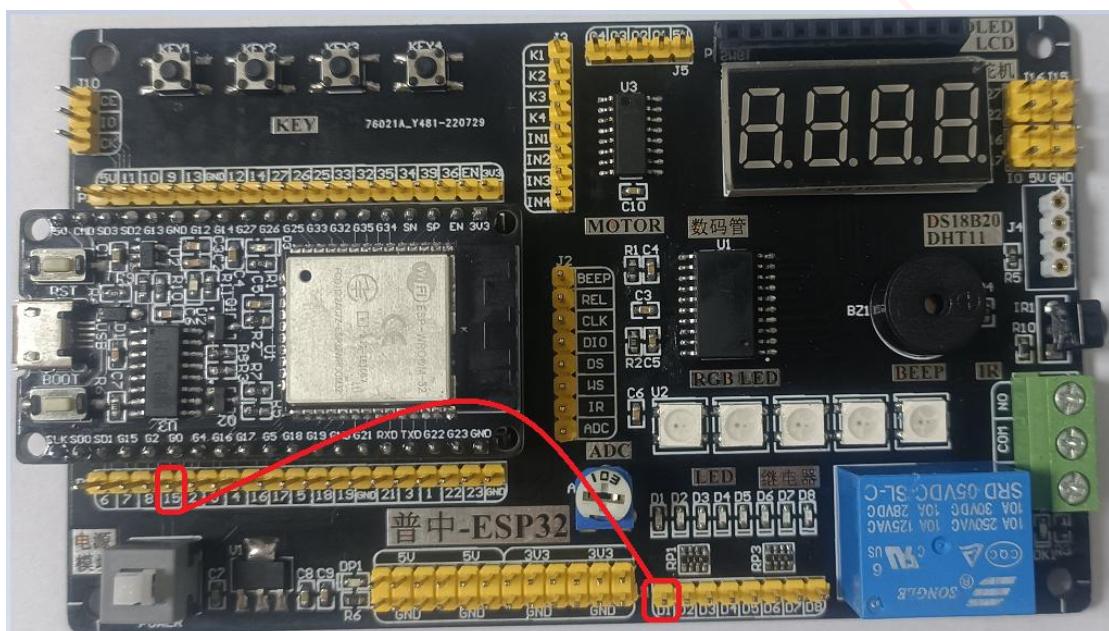
## 12.2 硬件设计

本实验使用到硬件资源如下：

- (1) LED 模块
- (2) ESP32 GPIO

LED 模块电路前面已介绍，此处不再重复。

本章实验使用 ESP32 的 IO15 引脚，接线如下所示：



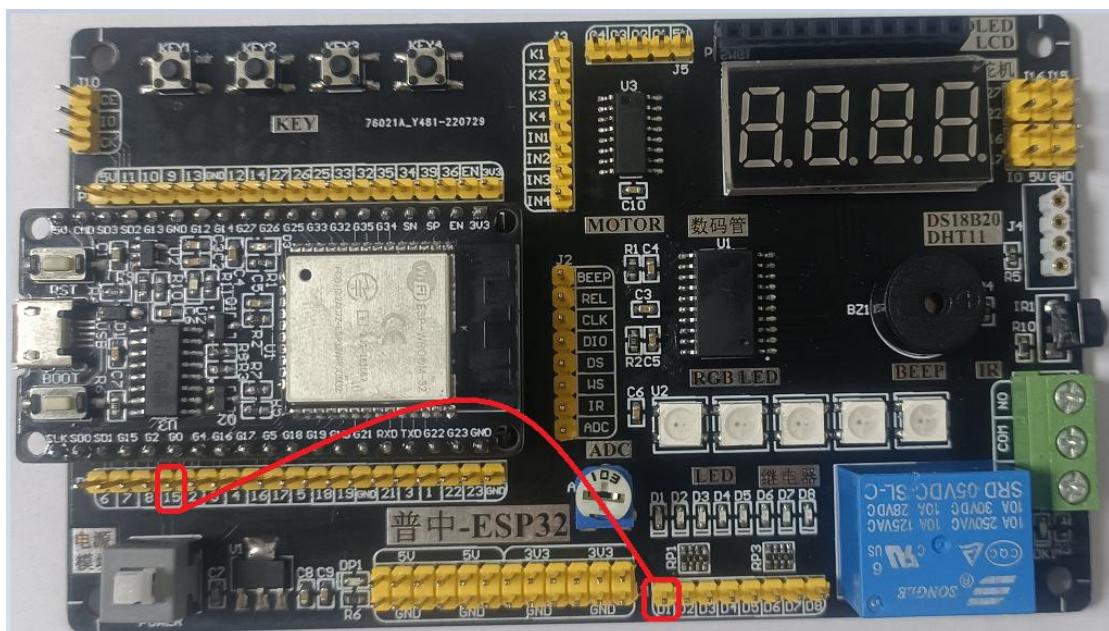
## 12.3 软件设计

下面我们打开“\4--实验程序\1--MicroPython 实验\1--基础实验\11--PWM 呼吸灯实验”程序，控制代码全部都在 main.py 中，代码如下：

```
main.py x
1 ...
2 深圳市普中科技有限公司 (PRECHIN 普中)
3 技术支持: www.prechin.net
4
5 实验名称: PWM呼吸灯实验
6 接线说明: LED模块-->ESP32 IO
7 (D1)-->(15)
8
9 实验现象: 程序下载成功后, D1指示灯呈现呼吸灯效果, 由暗变亮, 再由亮变暗
10 注意事项:
11 ...
12 ...
13 ...
14 #导入Pin模块
15 from machine import Pin
16 from machine import PWM
17 import time
18
19 #定义LED1控制对象
20 led1=PWM(Pin(15),freq=1000,duty=0)
21
22
23 #程序入口
24 if __name__=="__main__":
25     duty_value=0
26     fx=1
27     while True:
28
29         if fx==1:
30             duty_value+=10
31             if duty_value>1010:
32                 fx=0
33             else:
34                 duty_value-=10
35                 if duty_value<10:
36                     fx=1
37             led1.duty(duty_value)
38             time.sleep_ms(10)
```

## 12.4 实验现象

下载程序前, 按照如下接线:



**接线说明：**

LED模块-->ESP32 10

D1-->15

将程序下载到开发板内（可参考“2.2.5 程序下载运行”章节），可以看到D1指示灯由暗变亮，再由亮变暗。

## 课后作业

## 第 13 章 串口通信实验

串口几乎在任何单片机中都是具备的资源外设，使用它可实现程序下载，串口通信等。由于串口通信的简单方便，现如今越来越多支持串口通信功能的设备和模块，比如触摸屏、WIFI/蓝牙/GPS 等，让开发工作变得越来越简单且高效。本章我们来学习如何使用 MicroPython 控制 ESP32 的串口实现数据收发。本章分为如下几部分内容：

13.1 实验介绍

13.2 硬件设计

13.3 软件设计

13.4 实验现象

## 13.1 实验介绍

### 13.1.1 实验简介

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。

ESP32 有三个硬件 UART：UART0、UART1 和 UART2。它们每个都分配有默认的 GPIO，如下：

	UART0	UART1	UART2
tx	1	10	17
rx	3	9	16

UART0 用于下载和 REPL 调试，UART1 用于模块内部连接 Flash，通常也不使用，因此可以使用 UART2 与外部串口设备通信。

### 13.1.2 实验目的

使用 UART2 串口实现数据收发。

### 13.1.3 MicroPython 函数使用

UART 在 `machine` 的 `UART` 模块中，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>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
...
<b>特别说明：</b> 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
<code>uart.read([nbytes])</code>
【nbytes】 读取字节数
<code>UART.readline()</code>
读行
<code>UART.write(buf)</code>
【buf】 串口 TX 写数据

使用方法如下：

```
from machine import UART

uart1 = UART(1, baudrate=9600, tx=33, rx=32)
uart1.write('hello') # write 5 bytes
uart1.read(5)        # read up to 5 bytes
```

## 13.2 硬件设计

本实验使用到硬件资源如下：

(1) USB 转 TTL 模块

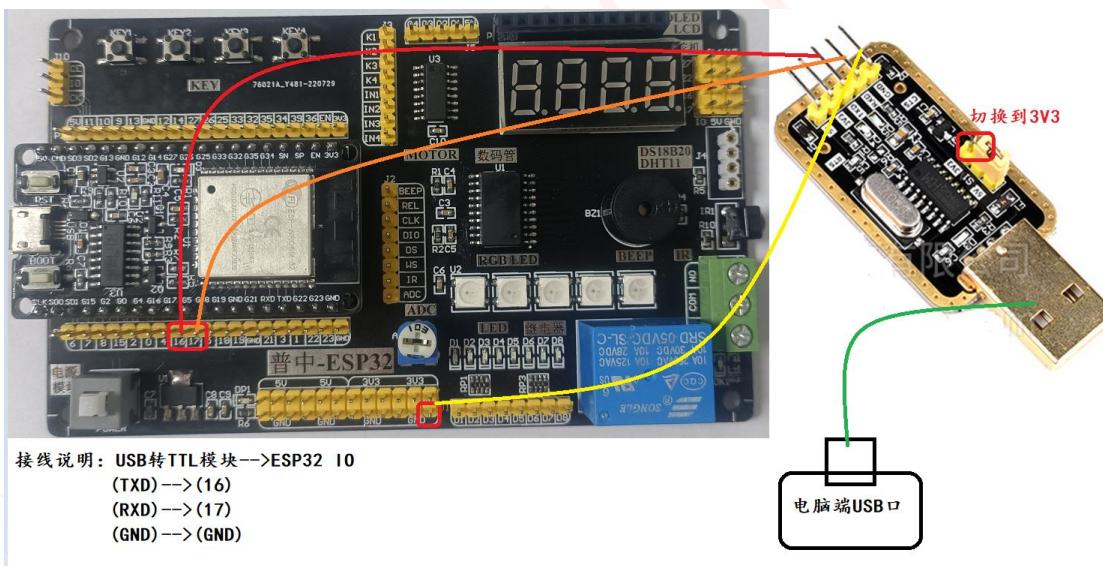
## (2) ESP32 GPIO

本实验使用 ESP32 开发板的 UART2 与电脑端串口通信，而电脑端是没有串口的，因此需要用到一个 USB 转 TTL 模块，配合电脑端串口调试助手实现通信。

USB 转 TTL 模块如下所示：



注意要使用 3.3V 电平的 USB 转串口 TTL 模块，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：



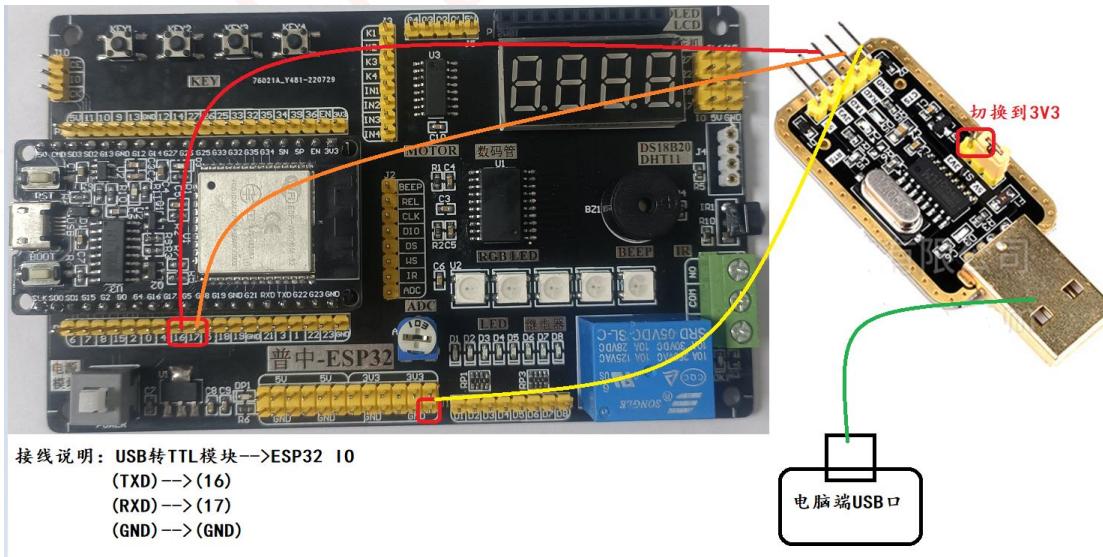
## 13.3 软件设计

下面我们打开 “\4--实验程序\1--MicroPython 实验\1--基础实验\12-串口通信实验” 程序，控制代码全部都在 main.py 中，代码如下：

```
main.py x
1 /**
2  深圳市普中科技有限公司 (PRECHIN 普中)
3  技术支持: www.prechin.net
4
5  实验名称: 串口通信实验
6  接线说明: USB转TTL模块-->ESP32 IO
7      (TXD)-->(16)
8      (RXD)-->(17)
9      (GND)-->(GND)
10
11 实验现象: 程序下载成功后, 打开串口调试助手, 选择好串口、波特率115200参数等, 在串口助手上发送字符数据,
12     ESP32串口接收后原封不动返回到串口助手显示
13
14 注意事项: USB转TTL模块上将电源切换为3.3V
15
16 ...
17
18 #导入Pin模块
19 from machine import Pin
20 from machine import UART
21 import time
22
23 #定义UART控制对象
24 uart=UART(2,115200,rx=16,tx=17)
25
26
27 #程序入口
28 if __name__=="__main__":
29     uart.write("Hello World!")
30     while True:
31         if uart.any():
32             text=uart.read(128)
33             uart.write(text)
34
```

## 13.4 实验现象

下载程序前, 按照如下接线:



打开串口调试助手, 在资料“\5--开发工具\4-串口调试助手\串口调试助手(丁丁)\sscom5.13.1.exe”, 选择 USB 转 TTL 模块的串口, 波特率设置为 115200

等参数，打开串口，将程序下载到开发板内（可参考“2.2.5 程序下载运行”章节），实验现象：上位机串口助手发送数据给ESP32串口，然后ESP32串口原封不动发送给上位机串口，实现串口收发。



## 课后作业

## 第 14 章 ADC 实验

前面我们介绍使用 PWM 输出不同电压值，类似于模拟信号输出。如果需要 ESP32 检测外部输入的模拟信号该怎么办？这就是本章要介绍的 ADC，通过 ADC 将模拟信号转换为数字信号给 ESP32 处理。本章就来学习使用 MicroPython 控制 ESP32 的 ADC，实现模拟信号的处理。本章分为如下几部分内容：

- 14.1 实验介绍
- 14.2 硬件设计
- 14.3 软件设计
- 14.4 实验现象

## 14.1 实验介绍

### 14.1.1 实验简介

ADC (analog to digital converter) 即模数转换器，它可以将模拟信号转换为数字信号。由于单片机只能识别二进制数字，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。

ADC 功能在 ESP32 引脚 32–39 上可用。请注意，使用默认配置时，ADC 引脚上的输入电压必须介于 0.0v 和 1.0v 之间（任何高于 1.0v 的值都将读为 4095）。如果需要增加测量范围，需要配置衰减器。

### 14.1.2 实验目的

使用 ADC 采集电位器 0–3.3V 电压，并输出在软件 Shell 控制台显示。

### 14.1.3 MicroPython 函数使用

ADC 在 `machine` 的 ADC 模块中，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	
adc=machine.ADC(Pin(id))	构建 ADC 对象。id: ESP32 的 ADC 在 32-39 引脚上。
使用方法	
adc.read()	获取 ADC 值。测量精度是 12 位，返回 0-4095 (表示 0-1V)。
adc.attenu(attenuation)	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。 attenuation: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。

使用方法如下：

```
from machine import ADC

adc = ADC(Pin(32))          # 在ADC引脚上创建ADC对象
adc.read()                   # 读取测量值，0-4095 表示电压从 0.0v - 1.0v

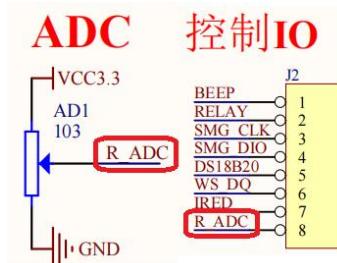
adc.atten(ADC.ATTN_11DB)    # 设置 11dB 衰减输入 (测量电压大致从 0.0v - 3.6v)
adc.width(ADC.WIDTH_9BIT)   # 设置 9位 精度输出 (返回值 0-511)
adc.read()                   # 获取重新配置后的测量值
```

## 14.2 硬件设计

本实验使用到硬件资源如下：

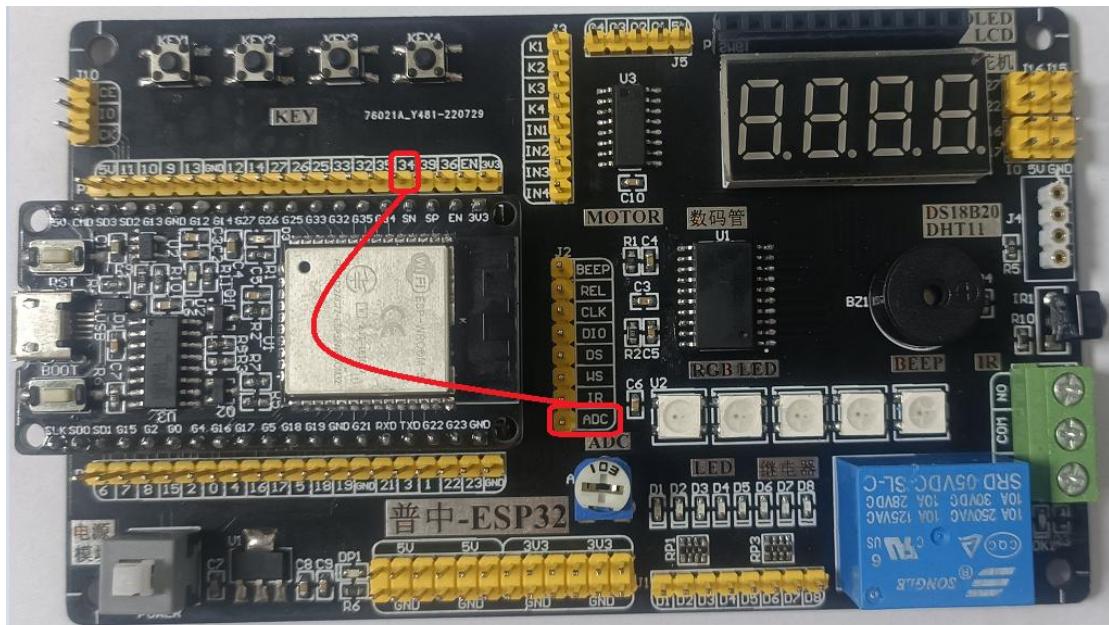
- (1) 电位器
- (2) ESP32 GPIO

ADC 电位器电路如下：



由图可知，J2 端子的 R\_ADC 脚为电位器电压输出端，可将该引脚与 ESP32 的 ADC 脚连接即可采集。

本章实验使用 ESP32 的 IO34 引脚，接线如下所示：



接线说明：ADC电位器-->ESP32 IO  
ADC-->(34)

### 14.3 软件设计

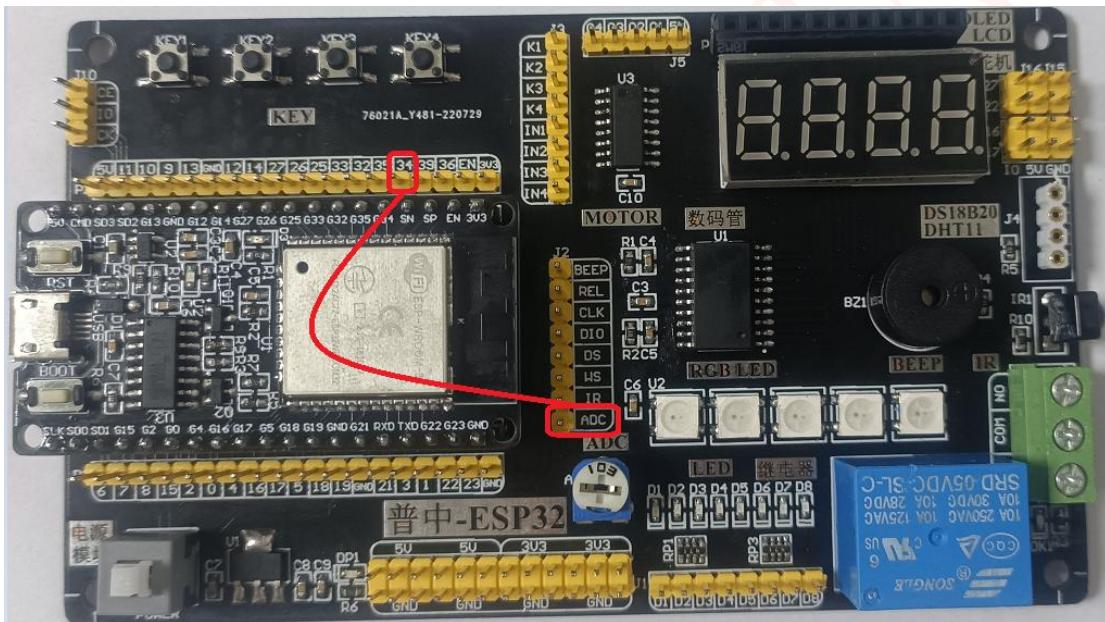
下面我们打开“\4--实验程序\1--MicroPython 实验\1--基础实验\13-ADC 实验”程序，控制代码全部都在 main.py 中，代码如下：

```
main.py x
1 ...
2 深圳市普中科技有限公司（PRECHIN 普中）
3 技术支持: www.prechin.net
4
5 实验名称: ADC实验
6 接线说明: ADC电位器-->ESP32 IO
7     ADC-->(34)
8
9 实验现象: 程序下载成功后, 会在软件shell控制台上输出ADC检测电压值, 调节电位器可改变检测电压
10
11 注意事项:
12 ...
13 ...
14 #导入Pin模块
15 from machine import Pin
16 from machine import ADC
17 from machine import Timer
18
19
20
21 #定义ADC控制对象
22 adc=ADC(Pin(34))
```

```
23 adc.atten(ADC.ATTN_11DB) #开启衰减，量程增大到3.3V
24
25 #定时器0中断函数
26 def time0_irq(time0):
27     adc_vol=3.3*adc.read()/4095
28     print("ADC检测电压: %.2fV" %adc_vol)
29
30 #程序入口
31 if __name__=="__main__":
32     time0=Timer(0) #创建time0定时器对象
33     time0.init(period=500,mode=Timer.PERIODIC,callback=time0_irq)
34     while True:
35         pass
```

## 14.4 实验现象

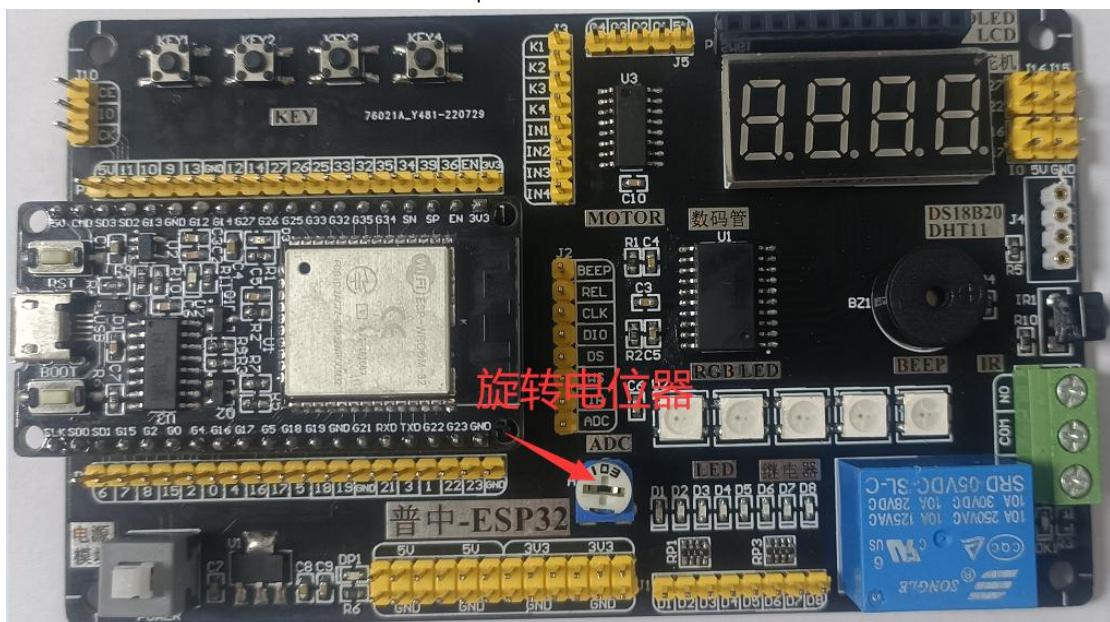
下载程序前，按照如下接线：



接线说明：ADC电位器-->ESP32 IO  
ADC-->(34)

将程序下载到开发板内（可参考“2.2.5 程序下载运行”章节），实验现象：

软件 Shell 控制台输出 ADC 检测电压，可旋转电位器改变电压值。



Shell ×

```
ADC检测电压: 2.39V
ADC检测电压: 2.39V
ADC检测电压: 1.71V
ADC检测电压: 0.79V
ADC检测电压: 2.93V
ADC检测电压: 1.98V
ADC检测电压: 0.43V
ADC检测电压: 1.10V
ADC检测电压: 2.13V
ADC检测电压: 2.13V
```

课后作业

## 第 15 章 RGB 彩灯实验

本章来学习使用 MicroPython 控制 WS2812B RGB 彩灯，实现任意颜色显示。

本章分为如下几部分内容：

15.1 实验介绍

15.2 硬件设计

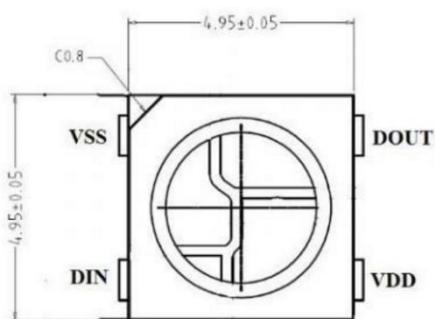
15.3 软件设计

15.4 实验现象

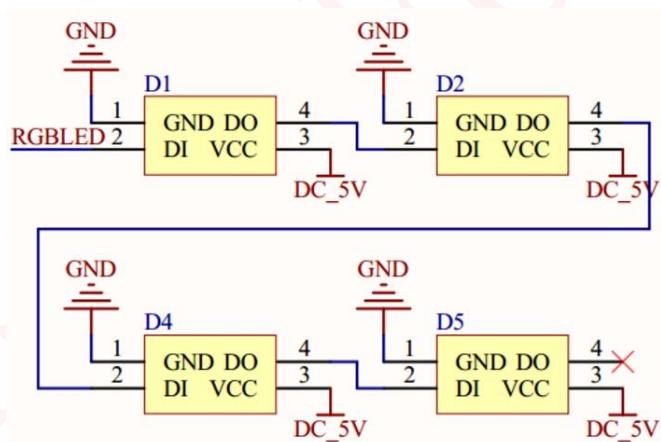
## 15.1 实验介绍

### 15.1.1 实验简介

WS2812B 是一款智能控制 LED 光源，控制电路和 RGB 芯片集成在一个 5050 组件的封装中。内部包括智能数字端口数据锁存器和信号整形放大驱动电路。其管脚图如下：



可将多个 RGB 灯珠级联，市面上的 RGB 彩灯带也是这样级联的，如下所示：



这样 ESP32 只需要通过 1 个 GPIO 口就可以控制数十上百的灯珠。注意：级联的灯珠越多，所需电流就越大，如果数量很多的情况，建议外接 5V 电源，然后与开发板共 GND 即可。

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 WS2812B RGB 彩灯。

## 15.1.2 实验目的

使用 ESP32 控制 RGB 彩灯循环点亮，且按照红、橙、黄、绿、蓝、靛、紫颜色循环变化。

## 15.1.3 MicroPython 函数使用

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。

使用方法如下：

```
from machine import Pin
from neopixel import NeoPixel

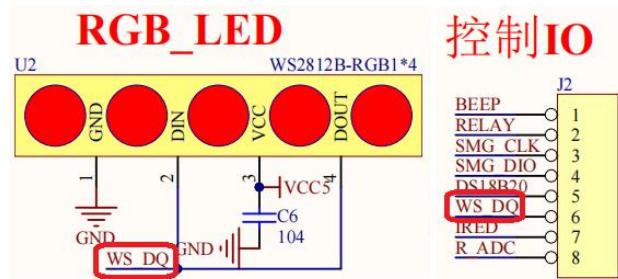
pin = Pin(0, Pin.OUT)    # 设置引脚GPIO0来驱动 NeoPixels
np = NeoPixel(pin, 8)    # 在GPIO0上创建一个 NeoPixel 对象，包含8个灯珠
np[0] = (255, 255, 255) # 设置第一个灯珠显示数据为白色
np.write()               # 写入数据
r, g, b = np[0]          # 获取第一个灯珠的颜色
```

## 15.2 硬件设计

本实验使用到硬件资源如下：

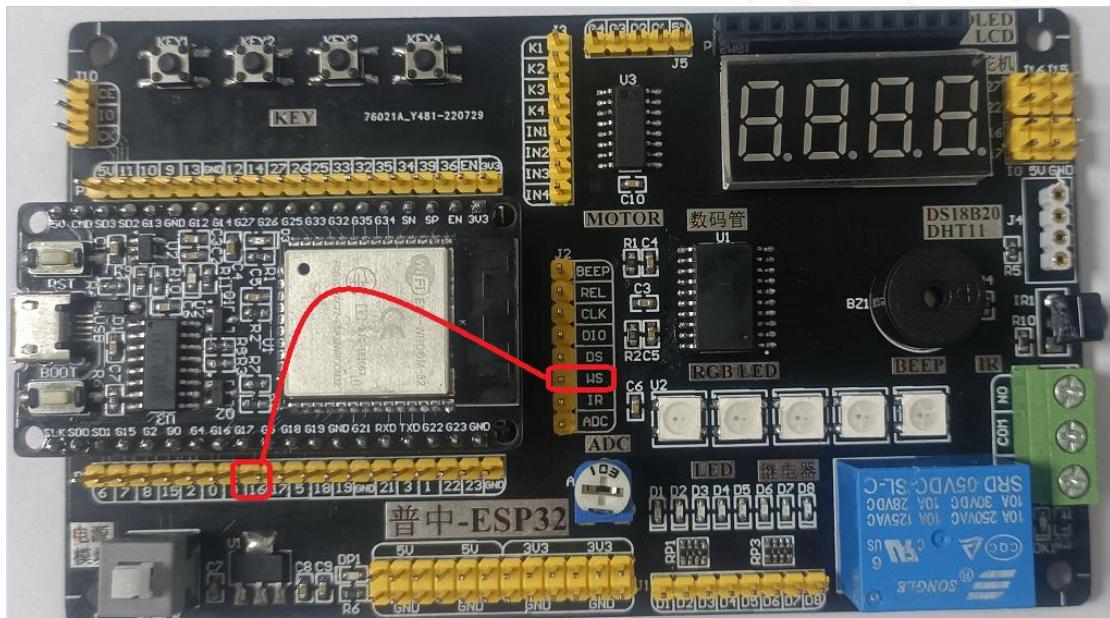
- (1) RGB 彩灯
- (2) ESP32 GPIO

RGB 彩灯电路如下：



图中已将 WS2812B 级联过程全部封装，因此没有展示其内部连接，只提供控制引脚。由图可知，J2 端子的 WS\_DQ 脚为 RGB 彩灯控制口，可将该引脚与 ESP32 的 GPIO 连接。

本章实验使用 ESP32 的 IO16 引脚，接线如下所示：



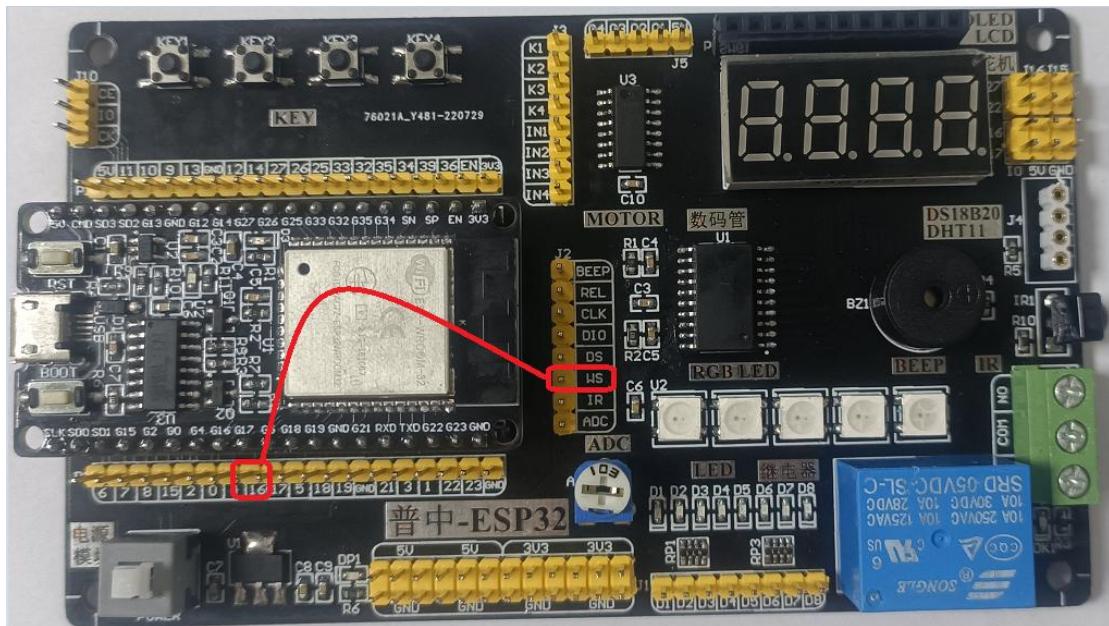
### 15.3 软件设计

下面我们打开 “\4--实验程序\1--MicroPython 实验\1--基础实验\14--RGB 彩灯实验” 程序，控制代码全部都在 main.py 中，代码如下：

```
main.py x
1  ...
2 深圳市普中科技有限公司 (PRECHIN 普中)
3 技术支持: www.prechin.net
4
5 实验名称: RGB彩灯实验
6 接线说明: RGB彩灯模块-->ESP32 IO
7   WS-->(16)
8
9 实验现象: 程序下载成功后, RGB彩灯循环点亮且循环变化颜色
10
11 注意事项:
12
13 ...
14
15 #导入Pin模块
16 from machine import Pin
17 from neopixel import NeoPixel
18 import time
19
20
21 #定义RGB控制对象
22 #控制引脚为16, RGB灯串联5个
23 pin=16
24 rgb_num=5
25 rgb_led=NeoPixel(Pin(pin,Pin.OUT),rgb_num)
26
27 #定义RGB颜色
28 RED = (255, 0, 0)
29 ORANGE = (255, 165, 0)
30 YELLOW = (255, 150, 0)
31 GREEN = (0, 255, 0)
32 BLUE = (0, 0, 255)
33 INDIGO = (75, 0, 130)
34 VIOLET = (138, 43, 226)
35 COLORS = (RED, ORANGE, YELLOW, GREEN, BLUE, INDIGO, VIOLET)
36
37 #程序入口
38 if __name__=="__main__":
39
40     while True:
41         for color in COLORS:
42             for i in range(rgb_num):
43                 rgb_led[i]=(color[0], color[1], color[2])
44                 rgb_led.write()
45                 time.sleep_ms(100)
46                 time.sleep_ms(1000)
47
```

## 15.4 实验现象

下载程序前, 按照如下接线:



接线说明：RGB彩灯模块—>ESP32 IO  
WS—>(16)

将程序下载到开发板内（**可参考“2.2.5 程序下载运行”章节**），实验现象：  
RGB 彩灯循环点亮，且按照红、橙、黄、绿、蓝、靛、紫颜色循环变化。

## 课后作业

## 第 16 章 数码管显示实验

本章来学习使用 MicroPython 控制 TM1637 驱动数码管显示，让数码管显示数字信息。本章分为如下几部分内容：

16.1 实验介绍

16.2 硬件设计

16.3 软件设计

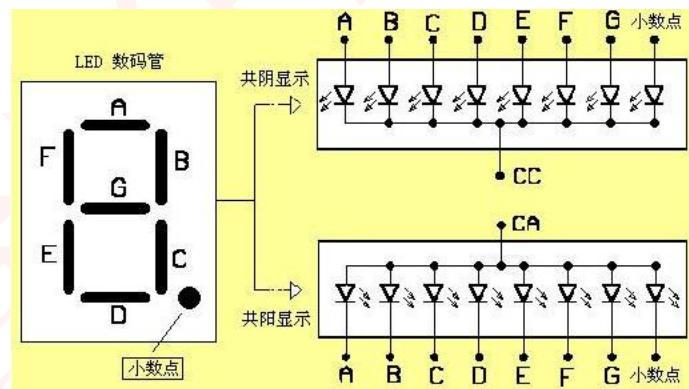
16.4 实验现象

## 16.1 实验介绍

### 16.1.1 实验简介

数码管是一种半导体发光器件，其基本单元是发光二极管。数码管也称 LED 数码管。数码管按段数可分为七段数码管和八段数码管，八段数码管比七段数码管多一个发光二极管单元，也就是多一个小数点 (DP)，这个小数点可以更精确的表示数码管想要显示的内容。

按发光二极管单元连接方式可分为共阳极数码管和共阴极数码管。共阳数码管是指将所有发光二极管的阳极接到一起形成公共阳极 (COM) 的数码管，共阴数码管是指将所有发光二极管的阴极接到一起形成公共阴极 (COM) 的数码管。不同位数的数码管实物图如下所示：



TM1637 是一种带键盘扫描接口的 LED(发光二极管显示器)驱动控制专用电路，内部集成有 MCU 数字接口、数据锁存器、LED 高压驱动、键盘扫描等电路。支持共阳数码管输出，辉度调节电路（占空比 8 级可调）。主要应用于电磁炉、微波炉及小家电产品的显示屏驱动。

TM1637 管脚图如下所示：

1	GND	K2	20
2	SG1/KS1	K1	19
3	SG1/KS2	CLK	18
4	SG1/KS3	DIO	17
5	SG1/KS4	VDD	16
6	SG1/KS5	GRID1	15
7	SG1/KS6	GRID2	14
8	SG1/KS7	GRID3	13
9	SG1/KS8	GRID4	12
10	GRID6	GRID5	11

符号	管脚名称	管脚号	说明
DIO	数据输入/输出	17	串行数据输入/输出，输入数据在 SCLK 的低电平变化，在 SCLK 的高电平被传输，每传输一个字节芯片内部都将在第九个时钟产生一个 ACK
CLK	时钟输入	18	在上升沿输入/输出数据
K1~K2	键扫数据输入	19-20	输入该脚的数据在显示周期结束后被锁存
SG1~SG8	输出（段）	2-9	段输出（也用作键扫描），N 管开漏输出
GRID6~GRID1	输出（位）	10-15	位输出，P 管开漏输出
VDD	逻辑电源	16	5V±10%
VSS	逻辑地	1	接系统地

### 16.1.2 实验目的

间隔 1S 计数，将计数值显示在数码管上。

### 16.1.3 MicroPython 函数使用

MicroPython 固件库内并没有集成 TM1637 模块，因此需要我们自己实现，对于不了解 TM1637 底层寄存器命令和时序的用户来说，要编写出驱动是困难的。MicroPython 拥有着庞大的用户群，自然 TM1637 模块也有开源的代码，直接拿过来使用即可，这就是使用 MicroPython 开发的高效之处，市面上常见的模块在网上几乎都可以找到相应的模块代码，大家一定要善于在网上搜索资源。我们已将 TM1637 模块驱动程序 tm1637.py 下载好，放入工程中。使用方法如下：

构造函数
smg=tm1637.TM1637(clk=Pin(16), dio=Pin(17))
构建tm1637数码管对象。clk为TM1637时钟管脚对象，dio为TM1637数据管脚对象。
使用方法
smg.number(num)
数码管显示十进制整数，显示位数受数码管位数限制
smg.numbers(num1, num2)
数码管显示小数，小数点后保留2位
smg.hex(val)
将十进制数转换十六进制显示
smg.brightness(val)
亮度调节：0-7参数有效
smg.temperature(num)
显示带温度符号°C，整数温度值
smg.show(string)
字符串显示，显示整数
smg.scroll(string, delay)
string: 字符串滚动显示，delay: 速度调节

使用方法如下：

```
import tm1637

#定义数码管控制对象
smg=tm1637.TM1637(clk=Pin(16),dio=Pin(17))

smg.number(12) #显示整数12
smg.numbers(1,24) #显示小数01.24
smg.hex(123) #将十进制数转换十六进制显示
smg.brightness(0) #亮度调节
smg.temperature(25) #显示带温度符号°C，整数温度值
smg.show("1314") #字符串显示，显示整数
smg.scroll("1314-520",500) #字符串滚动显示，速度调节
```

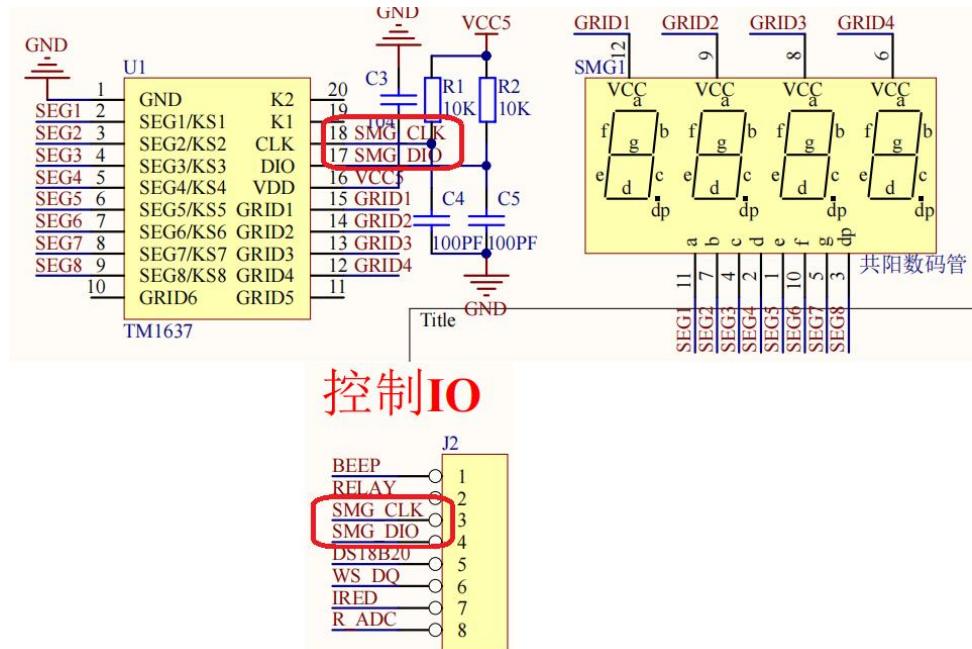
## 16.2 硬件设计

本实验使用到硬件资源如下：

- (1) TM1637 数码管模块

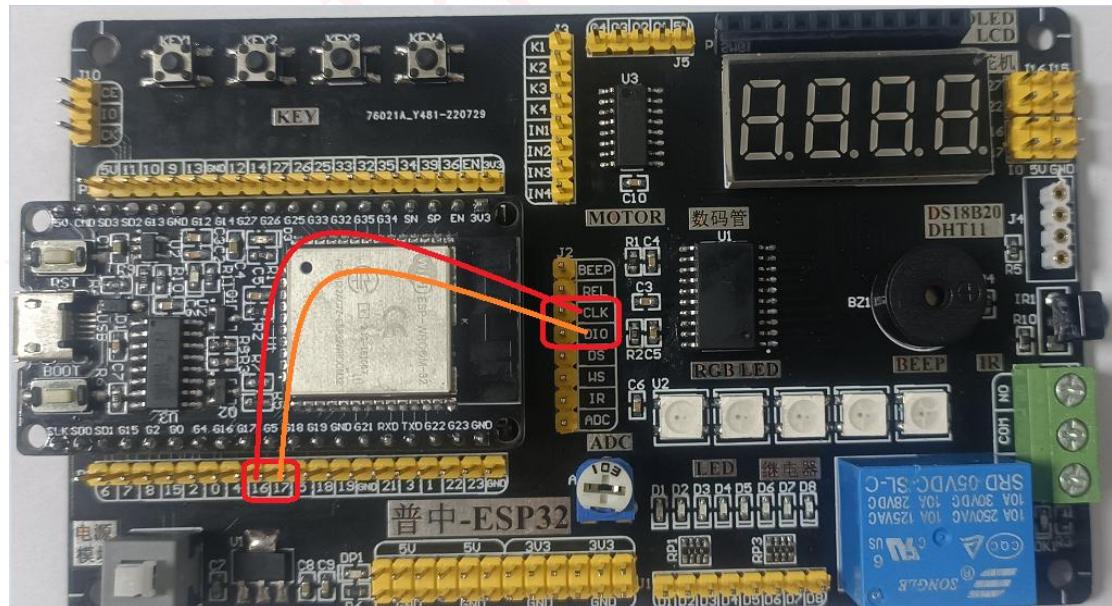
## (2) ESP32 GPIO

TM1637 数码管模块电路如下：



由图可知，J2 端子的 CLK、DIO 脚为 TM1637 控制口，可将该引脚与 ESP32 的 GPIO 连接。

本章实验使用 ESP32 的 IO16、17 引脚，接线如下所示：



接线说明：数码管模块-->ESP32 IO

CLK-->(16)

DIO-->(17)

## 16.3 软件设计

下面我们打开“\4--实验程序\1--MicroPython 实验\1--基础实验\15-数码管显示实验”程序，TM1637 驱动在 tm1637.py 中，该代码不展示，用户可打开工程查看。控制代码全部都在 main.py 中，代码如下：

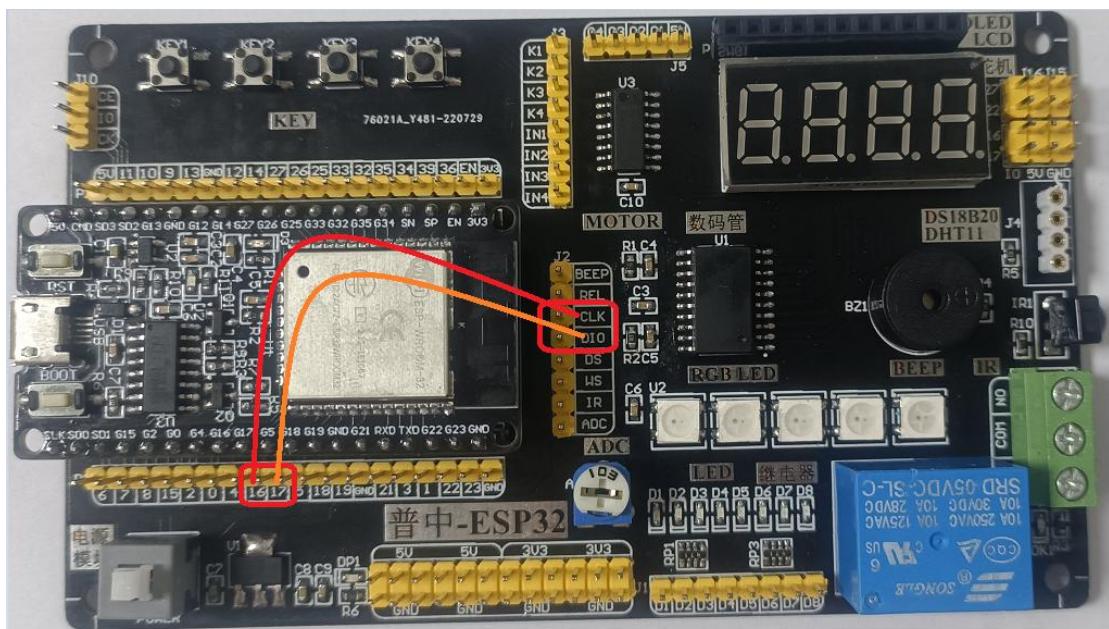
```
main.py x tm1637.py
1  ...
2 深圳市普中科技有限公司（PRECHIN 普中）
3 技术支持: www.prechin.net
4
5 实验名称: 数码管显示实验
6 接线说明: 数码管模块--->ESP32 IO
7     CLK-->(16)
8     DIO-->(17)
9
10 实验现象: 程序下载成功后, 数码管间隔1s从0开始计数显示
11
12 注意事项:
13
14 ...
15
16 #导入Pin模块
17 from machine import Pin
18 import time
19 import tm1637
20
21 #定义数码管控制对象
22 smg=tm1637.TM1637(clk=Pin(16),dio=Pin(17))
23
24 #程序入口
25 if __name__=="__main__":
26     #smg.numbers(1,24) #显示小数01.24
27     #smg.hex(123) #将十进制数转换十六进制显示
28     #smg.brightness(0) #亮度调节
29     #smg.temperature(25) #显示带温度符号°C, 整数温度值
30     #smg.show("1214") #字符串显示, 显示整数
31     #smg.scroll("1314-520",500) #字符串滚动显示, 速度调节
32     #time.sleep(5)
33     n=0
34     while True:
35         smg.number(n)
36         n+=1
37         time.sleep(1)
38
```

Pin 构造方法有几种，本实验中使用的是只传递引脚号，至于引脚使用输入还是输出模式和是否使用上下拉电阻此处不设置，而在 tm1637.py 构造方法中已设置该引脚输出模式，如下所示：

51	self.clk.init(Pin.OUT, value=0)
52	self.dio.init(Pin.OUT, value=0)

## 16.4 实验现象

下载程序前，按照如下接线：



接线说明：数码管模块-->ESP32 IO

CLK-->(16)

DIO-->(17)

将程序下载到开发板内（可参考“2.2.5 程序下载运行”章节），注意：因为本实验有 2 个程序模块，即 2 个.py 文件，而程序是从 main.py 开始运行的，其中又调用了 tm1637.py 内容，因此要先将该文件上载开发板中，然后可在 main.py 文件下点击“运行”或选择“运行当前脚本”。如下所示：

在main.py文件下，点击运行

```
PRECHIN
普中
www.prechin.net

文件 编辑 视图 运行 工具 帮助
①
②
main.py
此电脑
H:\普中-ESP32开发板资料\4-实验程序\1-MicroPython实验\1--基础实验\15-数码管显示实验
main.py
tm1637.py

深圳市普中科技有限公司（PRECHIN 普中）
技术支持: www.prechin.net

实验名称: 数码管显示实验
接线说明: 数码管模块-->ESP32 IO
    CLK-->(16)
    DIO-->(17)

实验现象: 程序下载成功后, 数码管间隔1s从0开始计数显示
注意事项:
...
#导入Pin模块
from machine import Pin
import time
import tm1637
|
#define数码管控制对象
smg=tm1637.TM1637(clk=Pin(16),dio=Pin(17))

#程序入口
if __name__=="__main__":
    #smg.numbers(1,24) #显示小数01.24
    #smg.show() #显示十进制数0123456789
    smg.show() #显示十六进制数0123456789ABCDEF

Shell < />
MicroPython v1.19.1 on 2022-06-18; ESP32 module with ESP32
Type "help()" for more information.
>>> %Run -c $EDITOR_CONTENT
```

实验现象：数码管显示计数值，从 0 开始，每秒加 1。

## 课后作业

## 第 17 章 RTC 实时时钟实验

本章来学习使用 MicroPython 中的 RTC 模块，使用该模块制作一个 RTC 实时时钟，通过软件 Shell 控制台输出日期与时间。本章分为如下几部分内容：

- 17.1 实验介绍
- 17.2 硬件设计
- 17.3 软件设计
- 17.4 实验现象

## 17.1 实验介绍

### 17.1.1 实验简介

时钟可以说是我们日常生活中最常用的东西了，手表、电脑、手机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，接下来我们就用开发板来制作一个属于自己的电子时钟。

### 17.1.2 实验目的

使用 RTC 模块让 ESP32 在软件 Shell 控制台上输出日期和时间。

### 17.1.3 MicroPython 函数使用

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。

使用方法如下：

```
from machine import RTC

rtc = RTC()
rtc.datetime((2017, 8, 23, 1, 12, 48, 0, 0)) # 设置时间(年, 月, 日, 星期, 时, 分, 秒, 微秒)
                                                # 其中星期使用0-6表示星期一至星期日。
rtc.datetime() # 获取当前日期和时间
```

## 17.2 硬件设计

由于 RTC 模块为 MicroPython 固件所含有的功能，且在 Shell 控制台输出，因此只需 ESP32 开发板即可实现。

## 17.3 软件设计

下面我们打开 “\4--实验程序\1--MicroPython 实验\1--基础实验\16-RTC 实时时钟实验” 程序，控制代码全部都在 main.py 中，代码如下：



```
main.py
1 ...
2 深圳市普中科技有限公司（PRECHIN 普中）
3 技术支持: www.prechin.net
4
5 实验名称: RTC实时时钟实验
6 接线说明:
7
8 实验现象: 程序下载成功后, 软件shell控制台间隔1S输出RTC实时时钟年月日时分秒星期
9
10 注意事项:
11
12 ...
13
14 #导入Pin模块
15 from machine import Pin
16 from machine import RTC
17 import time
18
19
20 #定义RTC控制对象
21 rtc=RTC()
22
23 #定义星期
24 week=("星期一","星期二","星期三","星期四","星期五","星期六","星期天")
25
26 #程序入口
27 if __name__=="__main__":
28     if rtc.datetime()[0]!=2022:
29         rtc.datetime((2022,8,10,2,10,58,0))
30     while True:
31         date_time=rtc.datetime()
32         print("%d-%d-%d %t %02d:%02d:%02d %t %s" %(date_time[0],date_time[1],date_time[2],
33                                         date_time[4],date_time[5],date_time[6],
34                                         week[date_time[3]]))
35         time.sleep(1)
36
```

进入程序入口内，首先读取日期和时间，通过比较年份是否为 2022，选择初始化当前日期和时间。初始化一次后，当前年份即为 2022，等待下次程序运行，即不再重新初始化时间。

## 17.4 实验现象

将程序下载到开发板内（可参考“2.2.5 程序下载运行”章节），实验现象：

软件 Shell 控制台输出日期和时间信息，如下所示：

```
Shell ✘
2022-8-10      10:43:40    星期二
2022-8-10      10:43:41    星期三
2022-8-10      10:43:42    星期三
2022-8-10      10:43:43    星期三
2022-8-10      10:43:44    星期三
```

注意：由于 ESP32 没有后备电池引脚，所以该 RTC 模块不支持掉电保存。

## 课后作业

## 第 18 章 DS1302 实时时钟实验

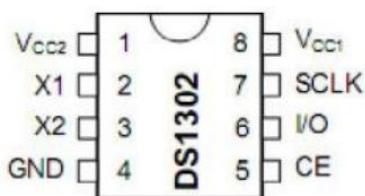
前面章节我们已经介绍使用 MicroPython 的 RTC 模块输出日期和时间，但无法实现断电保存，因此它并不是一个优秀的电子时钟。本章我们来学习使用 MicroPython 控制 DS1302，实现一个可断电保存的电子时钟。本章分为如下几部分内容：

- 18.1 实验介绍
- 18.2 硬件设计
- 18.3 软件设计
- 18.4 实验现象

## 18.1 实验介绍

### 18.1.1 实验简介

DS1302 是 DALLAS 公司推出的涓流充电时钟芯片，内含有一个实时时钟/日历和 31 字节静态 RAM，通过简单的串行接口与单片机进行通信。实时时钟/日历电路提供秒、分、时、日、周、月、年的信息，每月的天数和闰年的天数可自动调整。时钟操作可通过 AM/PM 指示决定采用 24 或 12 小时格式。



- 1, VCC2: 主电源引脚
- 2, X1、X2: DS1302 外部晶振引脚，通常需外接 32.768K 晶振
- 3, GND: 电源地
- 4, CE: 使能引脚，也是复位引脚。
- 5, I/O: 串行数据引脚，数据输出或者输入都从这个引脚
- 6, SCLK: 串行时钟引脚
- 7, VCC1: 备用电源

### 18.1.2 实验目的

使用 ESP32 控制 DS1302，在 Shell 控制台显示日期和时间。

### 18.1.3 MicroPython 函数使用

MicroPython 固件库内并没有集成 DS1302 模块，因此需要我们自己实现，对于不了解 DS1302 底层寄存器命令和时序的用户来说，要编写出驱动是困难的。MicroPython 拥有着庞大的用户群，自然 DS1302 模块也有开源的代码，直接拿过来使用即可，这就是使用 MicroPython 开发的高效之处，市面上常见的模块在网上几乎都可以找到相应的模块代码，大家一定要善于在网上搜索资源。我们已

将 DS1302 模块驱动程序 DS1302.py 下载好，放入工程中。使用方法如下：

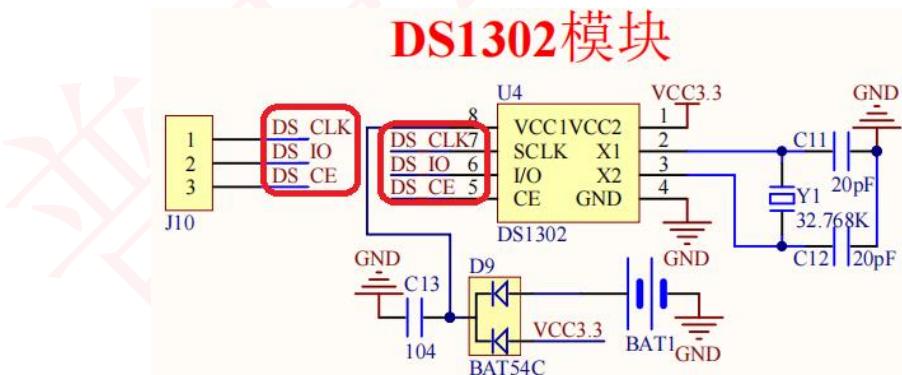
构造函数
ds1302=DS1302.DS1302(clk=Pin(18), dio=Pin(19), cs=Pin(23))
构建DS1302数码管对象。clk为DS1302时钟管脚对象，dio为数据管脚对象，cs为片选管脚对象。
使用方法
ds1302.DateTime([2022, 8, 10, 2, 11, 3, 58])
设置日期和时间。按顺序分别是：[年，月，日，星期，时，分，秒]， 其中星期使用 0-6 表示周一至周日
date_time=ds1302.DateTime()
获取当前日期和时间
ds1302.start()
DS1302开启，当使用stop()方法停止DS1302时可使用
ds1302.stop()
DS1302停止

## 18.2 硬件设计

本实验使用到硬件资源如下：

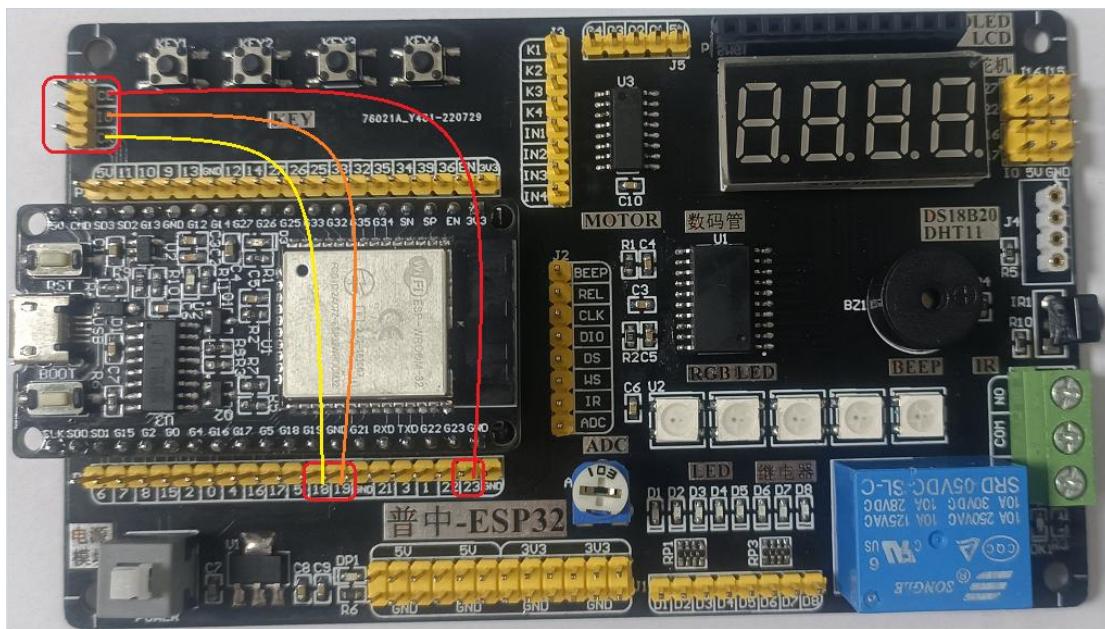
- (1) DS1302 模块
- (2) ESP32 GPIO

DS1302 模块电路如下：



由图可知，J10 端子的 CLK、IO、CE 脚为 DS1302 控制口，可将该引脚与 ESP32 的 GPIO 连接。

本章实验使用 ESP32 的 IO18、19、23 引脚，接线如下所示：



接线说明：DS1302时钟模块-->ESP32 IO

- (CE)-->(23)
- (IO)-->(19)
- (CK)-->(18)

### 18.3 软件设计

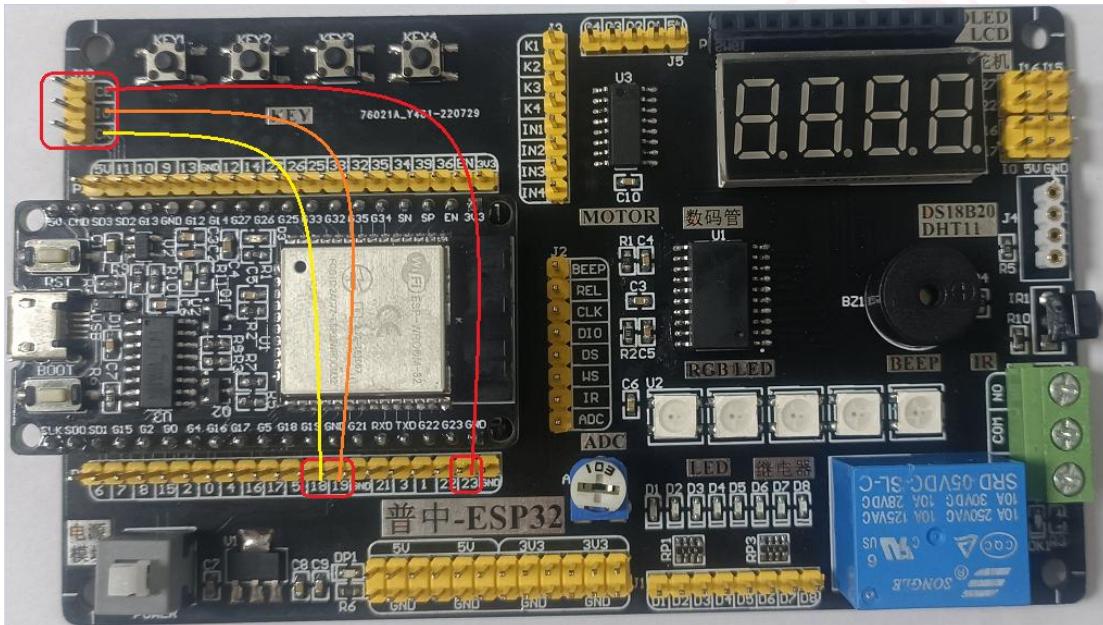
下面我们打开“\4--实验程序\1--MicroPython 实验\1--基础实验\17-DS1302 实时时钟实验”程序，DS1302 驱动在 DS1302.py 中，该代码不展示，用户可打开工程查看。控制代码全部都在 main.py 中，代码如下：

```
main.py x DS1302.py
1 ...
2 深圳市普中科技有限公司 (PRECHIN 普中)
3 技术支持: www.prechin.net
4
5 实验名称: DS1302实时时钟实验
6 接线说明: DS1302时钟模块-->ESP32 IO
7     (CE)-->(23)
8     (IO)-->(19)
9     (CK)-->(18)
10
11 实验现象: 程序下载成功后, 软件shell控制台间隔1S输出DS1302实时时钟年月日时分秒星期
12
13 注意事项:
14 ...
15
16
17 #导入Pin模块
18 from machine import Pin
19 import time
20 import DS1302
21
22 #定义DS1302控制对象
23 ds1302=DS1302.DS1302(clk=Pin(18),dio=Pin(19),cs=Pin(23))
24
25 #定义星期
26 week=("星期一","星期二","星期三","星期四","星期五","星期六","星期天")
```

```
27
28 #程序入口
29 if __name__=="__main__":
30     if ds1302.DateTime()[0]!=2022:
31         ds1302.DateTime([2022,8,10,2,11,3,58])
32     while True:
33         date_time=ds1302.DateTime()
34         print("%d-%d-%d %t %02d:%02d:%02d %t %s" %(date_time[0],date_time[1],date_time[2],
35                                         date_time[4],date_time[5],date_time[6],
36                                         week[date_time[3]]))
37         time.sleep(1)
38
```

## 18.4 实验现象

下载程序前，按照如下接线：



接线说明：DS1302时钟模块-->ESP32 10

- (CE)-->(23)
- (IO)-->(19)
- (CK)-->(18)

将程序下载到开发板内（可参考“2.2.5 程序下载运行”章节），注意：因为本实验有2个程序模块，即2个.py文件，而程序是从main.py开始运行的，其中又调用了DS1302.py内容，因此要先将该文件上载开发板中，然后可在main.py文件下点击“运行”或选择“运行当前脚本”。如下所示：

The screenshot shows the MicroPython code editor interface. At the top, there is a toolbar with icons for file operations. Below the toolbar, the file tree on the left shows two files: 'DS1302.py' and 'main.py'. A red circle labeled ① points to the 'main.py' file. A red circle labeled ② points to the green 'Run' button icon. The main workspace displays the code for 'main.py'. The code initializes a DS1302 module, defines a week array, and prints the date and time from the module's DateTime() method every second. The bottom section shows the 'Shell' output window with five lines of text, each containing a date (2022-8-10), a time (e.g., 11:25:46, 11:25:47, etc.), and a day of the week (星期二, 星期三, etc.).

```
注意事项:  
#导入Pin模块  
from machine import Pin  
import time  
import DS1302  
  
#定义DS1302控制对象  
ds1302=DS1302.DS1302(clk=Pin(18),dio=Pin(19),cs=Pin(23))  
  
#定义星期  
week=("星期一","星期二","星期三","星期四","星期五","星期六","星期天")  
  
#程序入口  
if __name__=="__main__":  
    if ds1302.DateTime()[0]!=2022:  
        ds1302.DateTime([2022,8,10,2,11,3,58])  
    while True:  
        date_time=ds1302.DateTime()  
        print("%d-%d-%d \t %02d:%02d:%02d \t %s" %(date_time[0],date_time[1],date_time[2],  
            date_time[4],date_time[5],date_time[6],  
            week[date_time[3]]))  
        time.sleep(1)  
  
Shell x  
2022-8-10 11:25:46 星期二  
2022-8-10 11:25:47 星期三  
2022-8-10 11:25:48 星期三  
2022-8-10 11:25:49 星期三  
2022-8-10 11:25:50 星期三
```

实验现象：软件 Shell 控制台输出日期和时间信息，如下所示：

The screenshot shows the MicroPython Shell window. It contains five lines of text, each representing a timestamp and the corresponding day of the week. The output is identical to the one shown in the previous screenshot.

```
2022-8-10 11:28:35 星期二  
2022-8-10 11:28:36 星期三  
2022-8-10 11:28:37 星期三  
2022-8-10 11:28:38 星期三  
2022-8-10 11:28:39 星期三
```

注意：由于 DS1302 有后备电池，当系统电源断电时，DS1302 依然可以走时，若不能在断电前时间继续走时，请更换纽扣电池。长时间放置，DS1302 依然会消耗纽扣电池电量。

## 课后作业

## 第 19 章 DS18B20 温度传感器实验

生活中，温度测量应用随处可见，小到数字温度计，大到工业设备温度测量等。本章来学习使用 MicroPython 控制 DS18B20 数字温度传感器，实时读取环境温度。本章分为如下几部分内容：

- 19.1 实验介绍
- 19.2 硬件设计
- 19.3 软件设计
- 19.4 实验现象

## 19.1 实验介绍

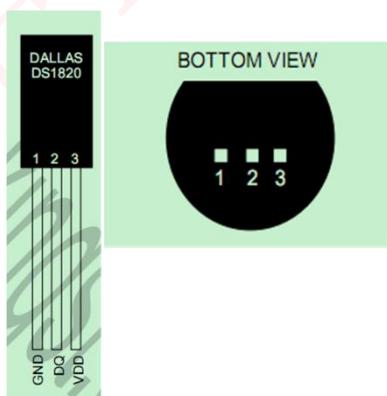
### 19.1.1 实验简介

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，多点组网的特点。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。

温度范围 $-55^{\circ}\text{C} \sim +125^{\circ}\text{C}$ ，在 $-10^{\circ}\text{C} \sim +85^{\circ}\text{C}$ 时精度为 $\pm 0.5^{\circ}\text{C}$ ；电压范围：3.0~5.5V；DS18B20 支持多点组网功能，多个 DS18B20 可以并联在唯一的三线上，实现组网多点测温。

DS18B20 不同封装外观实物如下图所示：





### 19.1.2 实验目的

读取 DS18B20 温度数据，并在软件 Shell 控制台上显示。

### 19.1.3 MicroPython 函数使用

ESP32 的 MicroPython 固件集成了单总线模块 onewire 和 DS18B20 模块 ds18x20，因此我们可以直接使用。说明如下：

构造函数
<code>ow=onewire.OneWire(machine.Pin(id))</code>
构建单总线对象。id: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。

构造函数
<code>ds=ds18x20.DS18X20(ow)</code>
构建 DS18B20 传感器对象。ow: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。rom: 表示对应的设备号。

使用方法如下：

```
from machine import Pin
import onewire

ow = onewire.OneWire(Pin(12)) # 在引脚 GPIO12 创建单总线对象ow
ow.scan()                      # 扫描设备, 返回设备编号列表
ow.reset()                      # 复位总线
ow.readbyte()                   # 读取1字节
ow.writebyte(0x12)              # 写入1个字节 (0x12)
ow.write('123')                 # 写入多个字节 ('123')
ow.select_rom(b'12345678')     # 根据ROM编号选择总线上的指定设备

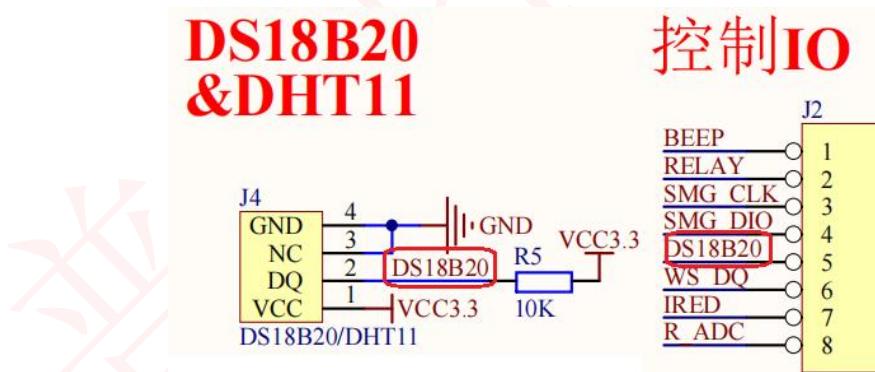
import time, ds18x20
ds = ds18x20.DS18X20(ow)
roms = ds.scan()
ds.convert_temp()
time.sleep_ms(750)
for rom in roms:
    print(ds.read_temp(rom))
```

## 19.2 硬件设计

本实验使用到硬件资源如下：

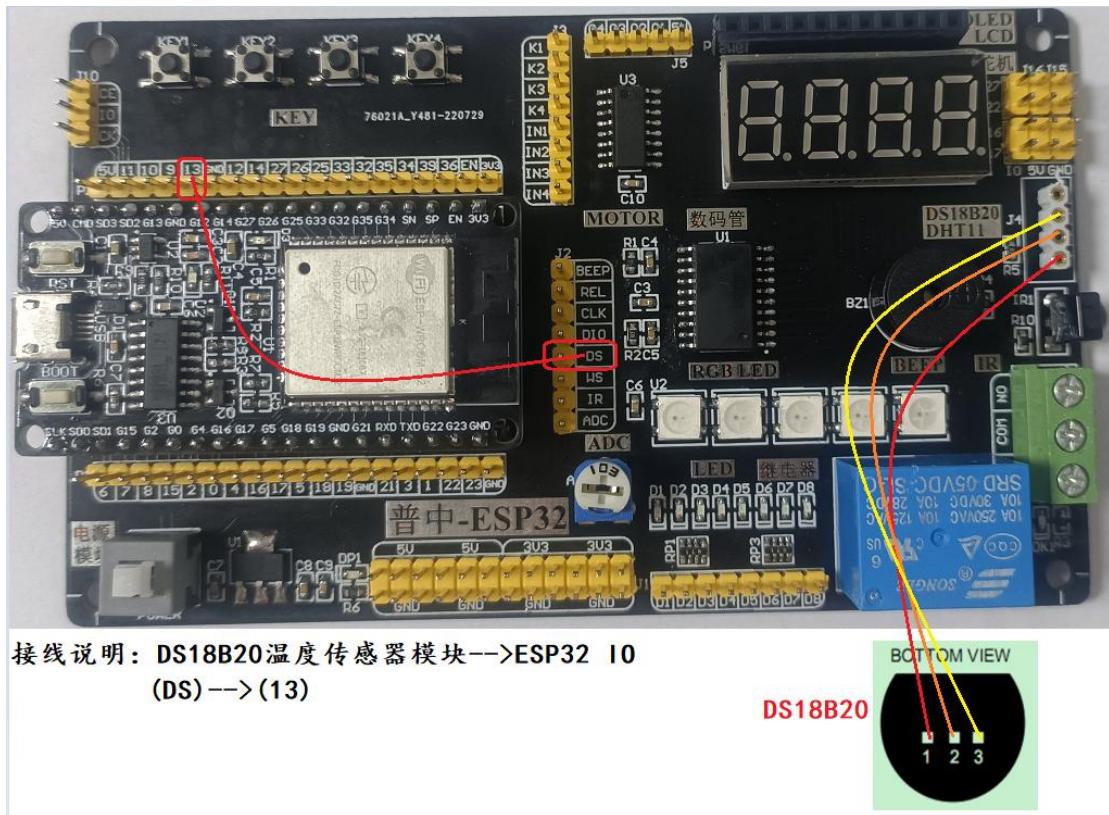
- (1) DS18B20 温度传感器
- (2) ESP32 GPIO

DS18B20 模块电路如下：



由图可知，J2 端子的 DS18B20 脚为 DS18B20 温度传感器控制口，可将该引脚与 ESP32 的 GPIO 连接。

本章实验使用 ESP32 的 IO13 引脚，接线如下所示：



接线说明：DS18B20温度传感器模块-->ESP32 IO  
(DS)-->(13)

DS18B20

### 19.3 软件设计

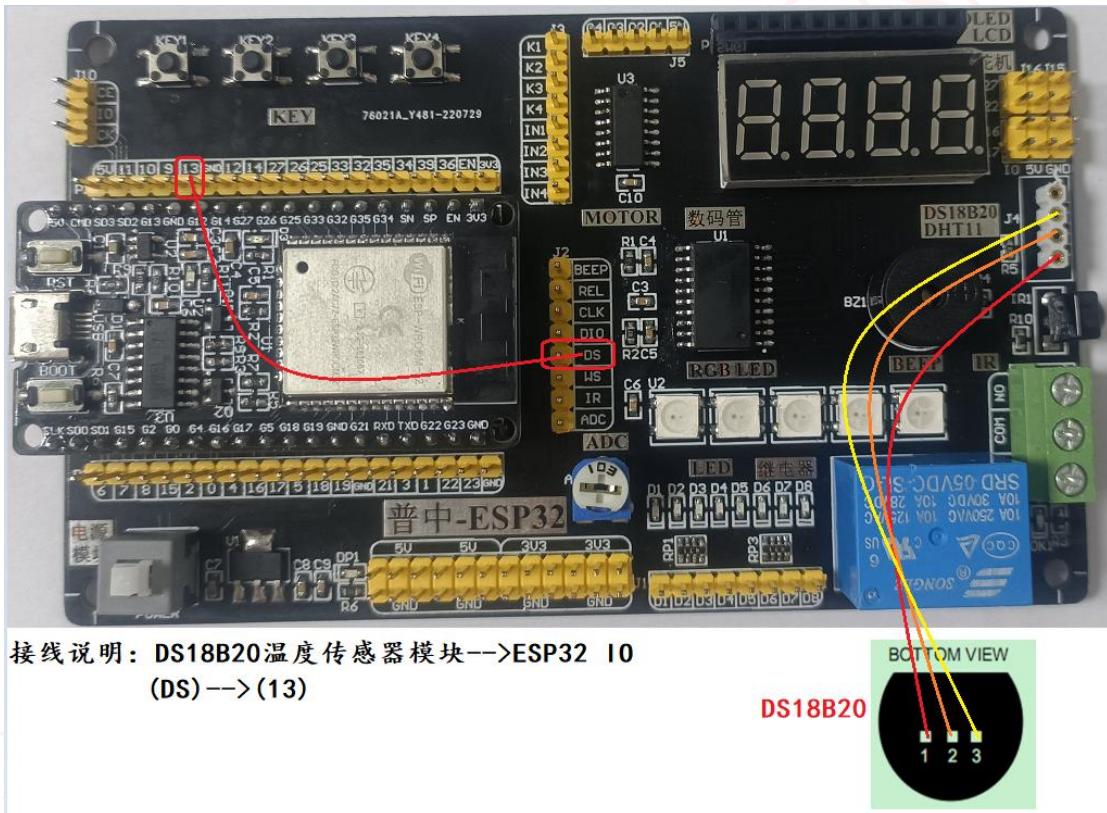
下面我们打开“\4--实验程序\1--MicroPython 实验\1--基础实验\18-DS18B20 温度传感器实验”程序，控制代码全部都在 main.py 中，代码如下：

```
main.py
1 ...
2 深圳市普中科技有限公司 (PRECHIN 普中)
3 技术支持: www.prechin.net
4
5 实验名称: DS18B20温度传感器实验
6 接线说明: DS18B20温度传感器模块-->ESP32 IO
7 (DS)-->(13)
8
9 实验现象: 程序下载成功后, 软件shell控制台间隔1S输出DS18B20温度传感器采集的温度
10
11 注意事项:
12 ...
13
14
15 #导入Pin模块
16 from machine import Pin
17 import time
18 import onewire
19 import ds18x20
20
21 #定义DS18B20控制对象
22 ds18b20=ds18x20.DS18X20(onewire.OneWire(Pin(13)))
```

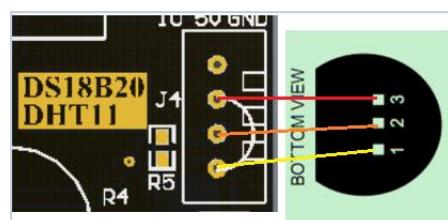
```
25 #程序入口
26 if __name__=="__main__":
27     roms = ds18b20.scan() #扫描是否存在DS18B20设备
28     print("DS18B20初始化成功!")
29     while True:
30         ds18b20.convert_temp()
31         time.sleep(1)
32         for rom in roms:
33             print("DS18B20检测温度: %.2f°C" %ds18b20.read_temp(rom))
34
35
```

## 19.4 实验现象

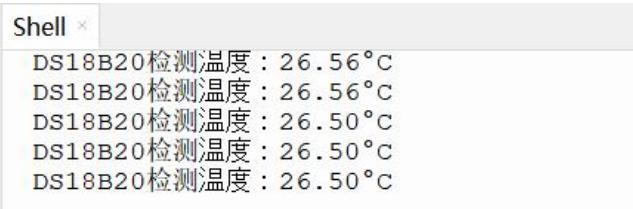
下载程序前，按照如下接线：



注意：DS18B20 温度传感器不要插反，否则电源短路，容易烧坏开发板。方向参考下图：



将程序下载到开发板内（可参考“2.2.5 程序下载运行”章节），实验现象：软件 Shell 控制台上间隔 1S 显示 DS18B20 传感器采集的温度值。如下：



```
Shell ×
DS18B20检测温度 : 26.56°C
DS18B20检测温度 : 26.56°C
DS18B20检测温度 : 26.50°C
DS18B20检测温度 : 26.50°C
DS18B20检测温度 : 26.50°C
```

## 课后作业

# 第 20 章 DHT11 温湿度传感器实验

生活中，除了温度指标要求测量外，还有一个非常重要的指标是湿度，温湿度测量应用随处可见，例如智能家居系统中温湿度的测量、农业大棚内温湿度测量等。本章来学习使用 MicroPython 控制 DHT11 温湿度传感器，实时读取环境温度和湿度。本章分为如下几部分内容：

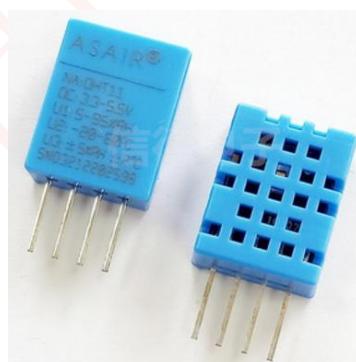
- 20.1 实验介绍
- 20.2 硬件设计
- 20.3 软件设计
- 20.4 实验现象

## 20.1 实验介绍

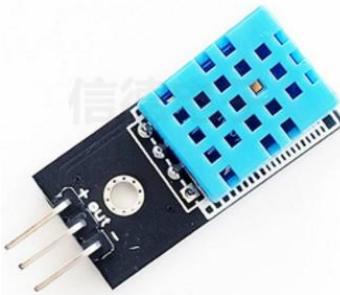
### 20.1.1 实验简介

温湿度也是我们日常常见的指标，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。

DHT11 具有小体积、极低的功耗，与 DS18B20 一样是单总线接口，为 4 针单排引脚封装，连接方便。如下：



这种封装的可直接插入我们开发板 DHT11 传感器接口上使用，当然也可使用 DHT11 模块，通过导线连接 IO 口。DHT11 模块如下：



+：电源正极（3-5.5V），OUT：数据口，-：电源负极。

湿度测量范围：20~90%RH，精度±5%RH；

温度测量范围：-20~+60°C，精度±2°C；

DHT11 性价比较高，很适合学习使用，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 或者其它更高级的传感器。

### 20.1.2 实验目的

控制 DHT11 温湿度传感器，间隔 2S 读取环境温度和湿度，在软件 Shell 控制台输出。

### 20.1.3 MicroPython 函数使用

ESP32 的 MicroPython 固件集成了 dht11 模块，因此我们可以直接使用。

说明如下：

构造函数
<code>d = dht.DHT11(machine.Pin(id))</code>
构建 DHT11 传感器对象。id: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。

使用方法如下：

```
import dht
import machine

d = dht.DHT11(machine.Pin(4))
d.measure()
d.temperature() # eg. 23 (°C)
d.humidity()    # eg. 41 (% RH)

d = dht.DHT22(machine.Pin(4))
d.measure()
d.temperature() # eg. 23.6 (°C)
d.humidity()    # eg. 41.3 (% RH)
```

## 20.2 硬件设计

本实验使用到硬件资源如下：

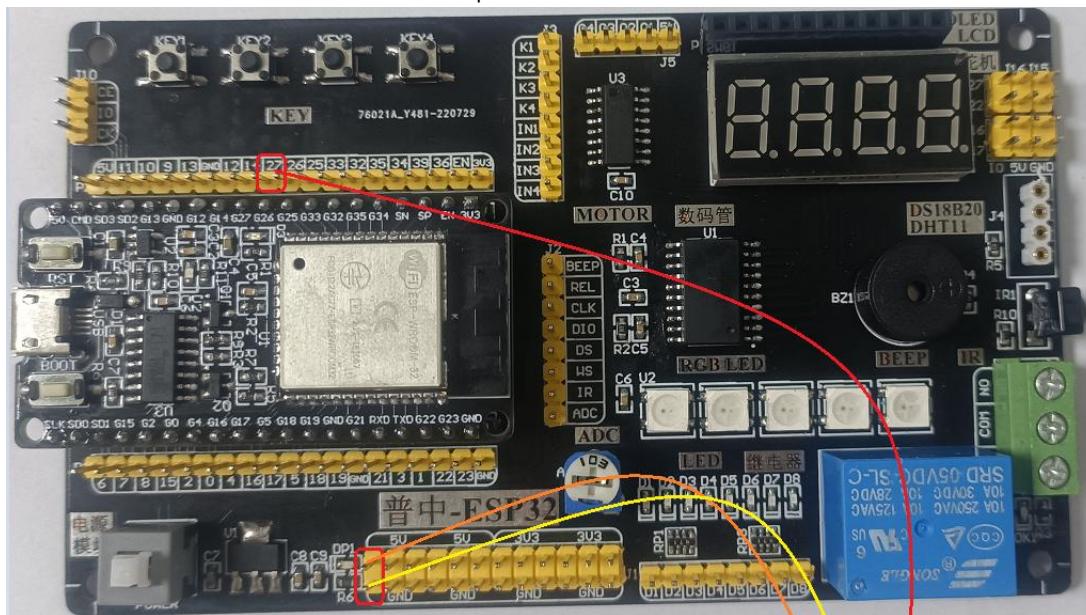
- (1) DHT11 温湿度传感器
- (2) ESP32 GPIO

DHT11 模块电路如下：



由图可知，DHT11 和 DS18B20 可共用一个接口，J2 端子的 DS18B20 脚为 DS18B20 和 DHT11 传感器控制口，可将该引脚与 ESP32 的 GPIO 连接。如果使用 DHT11 模块的可直接将模块通信口与 ESP32 连接，本章实验使用 DHT11 模块连接。

本章实验使用 ESP32 的 IO27 引脚，接线如下所示：



接线说明: DHT11温湿度传感器模块-->ESP32 IO

(VCC)-->(5V)

(DATA)-->(27)

(GND)-->(GND)

DHT11模块

## 20.3 软件设计

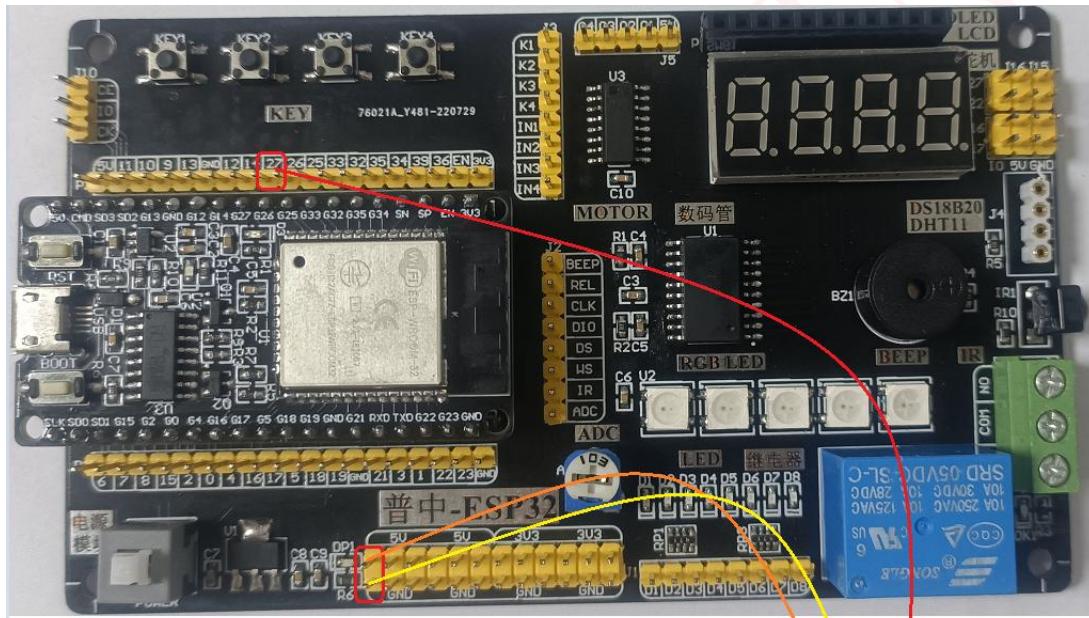
下面我们打开“\4—实验程序\1—MicroPython 实验\1—基础实验\19-DHT11温湿度传感器实验”程序，控制代码全部都在 main.py 中，代码如下：

```
main.py *
1 ...
2 深圳市普中科技有限公司 (PRECHIN 普中)
3 技术支持: www.prechin.net
4
5 实验名称: DHT11温湿度传感器实验
6 接线说明: DHT11温湿度传感器模块-->ESP32 IO
7     (VCC)-->(5V)
8     (DATA)-->(27)
9     (GND)-->(GND)
10
11 实验现象: 程序下载成功后, 软件shell控制台间隔2S输出DHT11温湿度传感器采集的温度和湿度
12
13 注意事项:
14 ...
15
16 #导入Pin模块
17 from machine import Pin
18 import time
19 import dht
20
21 #定义DHT11控制对象
22 dht11=dht.DHT11(Pin(27))
23
24
25
```

```
26 #程序入口
27 if __name__=="__main__":
28     time.sleep(1) #首次启动间隔1S让传感器稳定
29     while True:
30         dht11.measure() #调用DHT类库中测量数据的函数
31         temp = dht11.temperature()
32         humi = dht11.humidity()
33         if temp==None:
34             print("DHT11传感器检测失败!")
35         else:
36             print("temp=%d°C humi=%dRH" %(temp,humi))
37         time.sleep(2) #如果延时时间过短, DHT11温湿度传感器不工作
38
```

## 20.4 实验现象

下载程序前，按照如下接线：



接线说明：DHT11温湿度传感器模块-->ESP32 10

(VCC)-->(5V)

(DATA)-->(27)

(GND)-->(GND)

将程序下载到开发板内（[可参考“2.2.5 程序下载运行”章节](#)），实验现象：

软件 Shell 控制台上间隔 2S 显示 DHT11 传感器采集的温湿度值。如下：

```
Shell x
temp=25 °C  humi=56RH
temp=25 °C  humi=57RH
temp=25 °C  humi=57RH
temp=25 °C  humi=57RH
temp=25 °C  humi=57RH
```

## 课后作业

# 第 21 章 超声波测距实验

喜欢电子制作的朋友，相信你们心中都有过一个小目标，那就是制作一台智能小车。超声波避障是智能小车中一个基础功能。本章来学习使用 MicroPython 控制 HC-SR04 超声波模块，实现距离检测。本章分为如下几部分内容：

- 21.1 实验介绍
- 21.2 硬件设计
- 21.3 软件设计
- 21.4 实验现象

## 21.1 实验介绍

### 21.1.1 实验简介

HC-SR04 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。

模块外形如下所示：



管脚功能定义：

VCC：供电电源； Trig：触发信号； Echo：反馈信号； GND：电源地

模块主要参数：

●工作电压：3-5.5V 宽电压供电

●工作电流：<20mA

● 测量距离：2cm~450cm

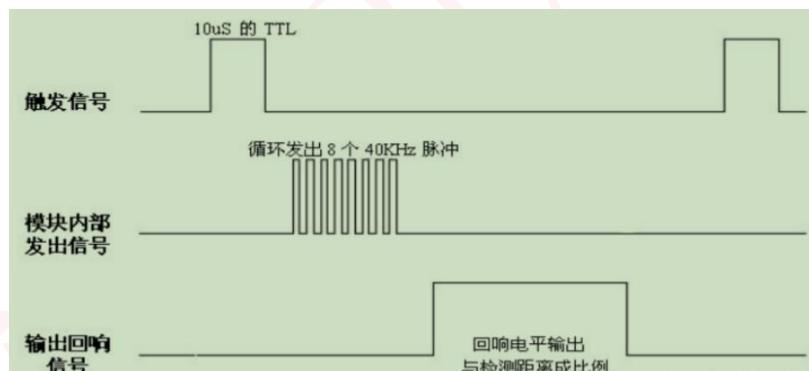
● 测量精度：0.5cm

● 通信接口：IO 数字接口

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 10us 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。

HC-SR04 超声波模块工作时序图如下：



### 21.1.2 实验目的

控制 HC-SR04 超声波模块，实现距离检测，在软件 Shell 控制台上显示。

### 21.1.3 MicroPython 函数使用

MicroPython 固件库内并没有集成 HC-SR04 模块，因此需要我们自己实现，对于不了解超声波测距原理及时序的用户来说，要编写出驱动是困难的。

MicroPython 拥有着庞大的用户群，自然 HC-SR04 模块也有开源的代码，直接拿

过来使用即可，这就是使用 MicroPython 开发的高效之处，市面上常见的模块在网上几乎都可以找到相应的模块代码，大家一定要善于在网上搜索资源。我们已将 HC-SR04 模块驱动程序 hcsr04.py 下载好，放入工程中。使用方法如下：

构造函数
hcsr04=hCSR04(trigger_pin=4, echo_pin=27)
构建HCSR04对象。定义传感器连接的trig和echo引脚
使用方法
hcsr04.distance_mm()
获取距离数据，单位mm
hcsr04.distance_cm()
获取距离数据，单位cm

使用方法如下：

```
from hcsr04 import HCSR04

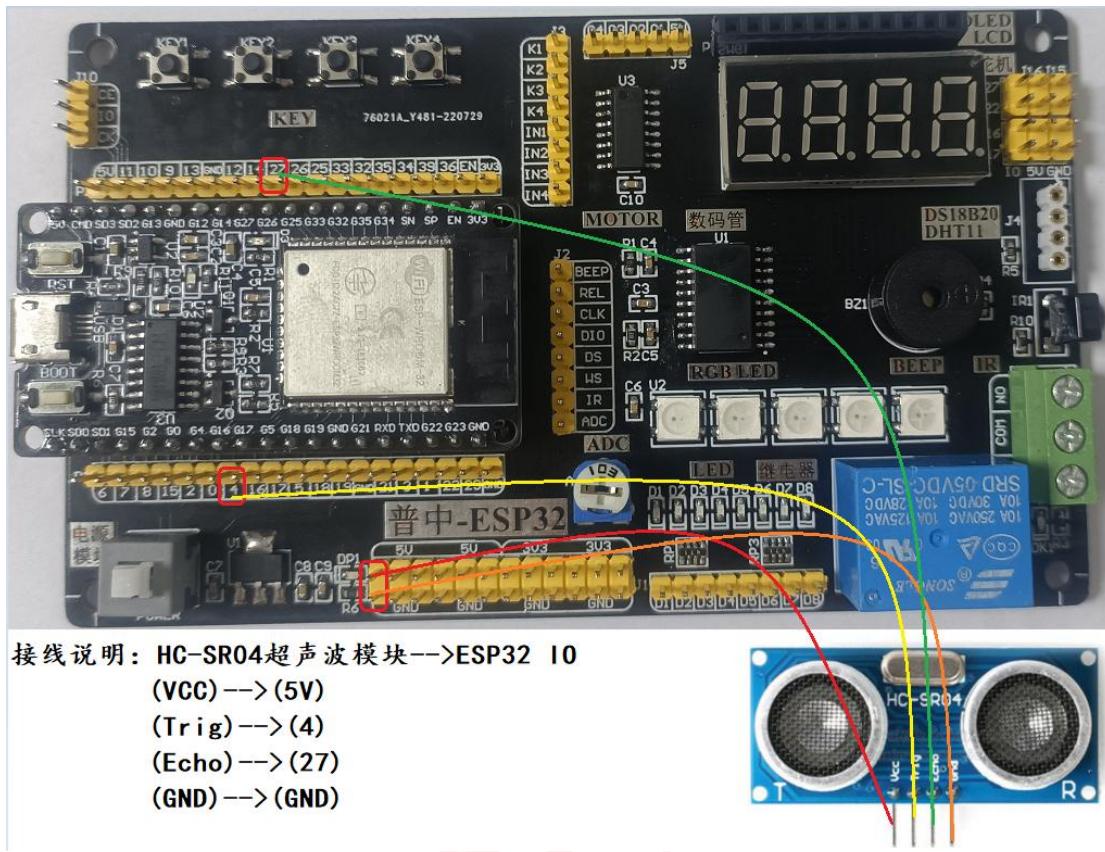
#定义HCSR04控制对象
hCSR04=HCSR04(trigger_pin=4, echo_pin=27)
distance=hCSR04.distance_cm()
print("distance=%.2fCM" %distance)
```

## 21.2 硬件设计

本实验使用到硬件资源如下：

- (1) HC-SR04 模块
- (2) ESP32 GPIO

本章实验使用 ESP32 的 I04、27 引脚，接线如下所示：



## 21.3 软件设计

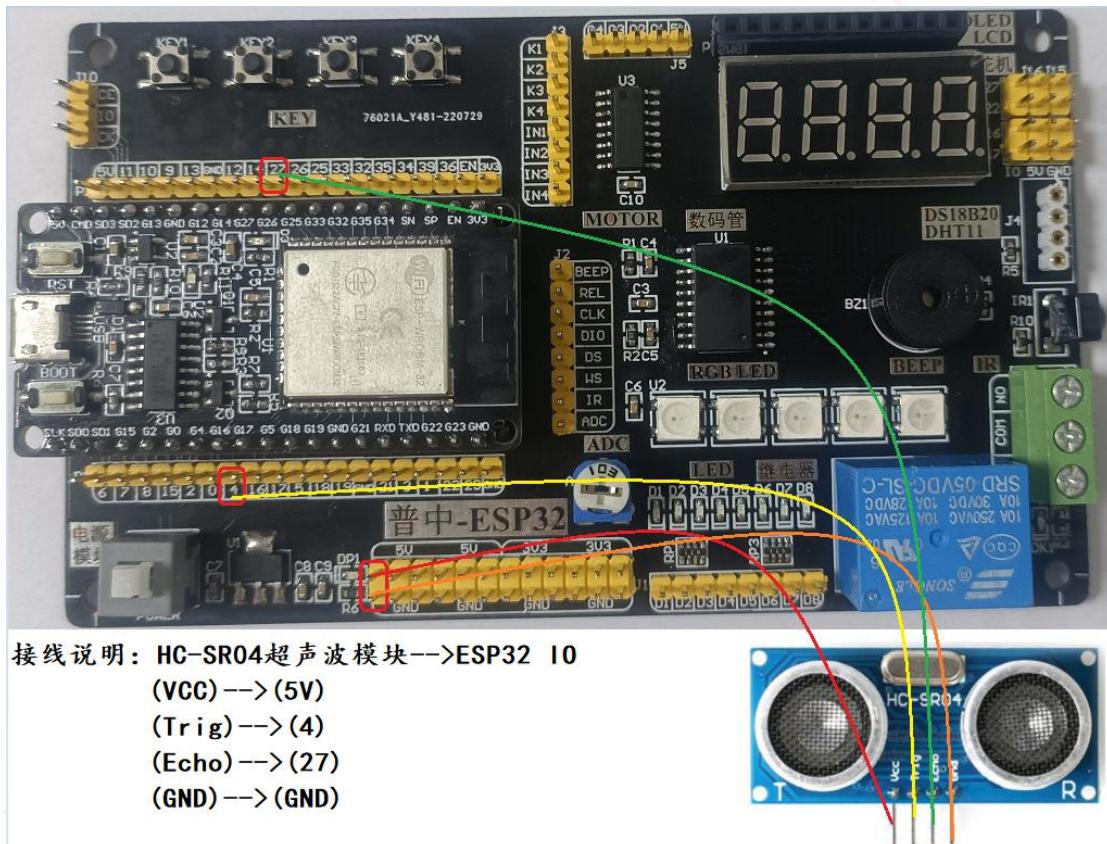
下面我们打开“\4--实验程序\1--MicroPython 实验\1--基础实验\20-超声波测距实验”程序，HC-SR04 驱动在 hcsr04.py 中，该代码不展示，用户可打开工程查看。控制代码全部都在 main.py 中，代码如下：

```
main.py < hcsr04.py <
1 ...
2 深圳市普中科技有限公司 (PRECHIN 普中)
3 技术支持: www.prechin.net
4
5 实验名称: 超声波测距实验
6 接线说明: HC-SR04超声波模块-->ESP32 IO
7     (VCC)-->(5V)
8     (Trig)-->(4)
9     (Echo)-->(27)
10    (GND)-->(GND)
11
12 实验现象: 程序下载成功后, 软件shell控制台间隔一段时间输出超声波模块测量距离
13
14 注意事项:
15
16 ...
17
18 #导入Pin模块
19 from machine import Pin
20 import time
21 from hcsr04 import HCSR04
22
23 #定义HCSR04控制对象
24 hcsr04=HCSR04(trigger_pin=4, echo_pin=27)
25
```

```
26 #程序入口
27 if __name__=="__main__":
28
29     while True:
30         distance=hcsr04.distance_cm()
31         print("distance=%2fCM" %distance)
32         time.sleep(0.5)
33
```

## 21.4 实验现象

下载程序前，按照如下接线：



将程序下载到开发板内（可参考“2.2.5 程序下载运行”章节），注意：因为本实验有2个程序模块，即2个.py文件，而程序是从main.py开始运行的，其中又调用了hcsr04.py内容，因此要先将该文件上载开发板中，然后可在main.py文件下点击“运行”或选择“运行当前脚本”。如下所示：

The screenshot shows the PRECHIN software interface. On the left, there's a file browser window titled '此电脑' (This Computer) showing a directory structure. A red arrow labeled ① points from the 'hcsr04.py' file in the browser to the same file in the main code editor window. Another red arrow labeled ② points from the 'main.py' file in the code editor to the green circular icon at the top of the software window. The code editor window has tabs for 'main.py' and 'hcsr04.py'. The 'main.py' tab is active, displaying Python code for an ESP32 project involving an HCSR04 ultrasonic sensor. The 'hcsr04.py' tab also contains some code. Below the code editor is a 'Shell' window showing the output of the program, which is printing distance measurements in centimeters.

```
PRECHIN
普中
www.prechin.net

文件 编辑 视图 运行 工具 帮助
②在main.py文件下运行
文件 x
此电脑
H:\普中-ESP32开发板资料\4--实验程序\1--MicroPython实验\1--基础实验\20-超声波测距实验
①上传到开发板内
hcsr04.py
main.py

MicroPython设备
boot.py
hcsr04.py

main.py hcsr04.py

1
2 深圳市普中科技有限公司 (PRECHIN 普中)
3 技术支持: www.prechin.net
4
5 实验名称: 超声波测距实验
6 接线说明: HC-SR04超声波模块-->ESP32 IO
7 (VCC)-->(5V)
8 (Trig)-->(4)
9 (Echo)-->(27)
10 (GND)-->(GND)
11
12 实验现象: 程序下载成功后, 软件shell控制台间隔一段时间输出超声波模块测量距离
13
14 注意事项:
15 ...
16
17
18 #导入Pin模块
19 from machine import Pin
20 import time
21 from hcsr04 import HCSR04
22
23 #定义HCSR04控制对象
24 hcsr04=HCSR04(trigger_pin=4, echo_pin=27)
25
26 #程序入口
27 if __name__ == "__main__":
Shell x
distance=195.70CM
distance=195.72CM
distance=195.29CM
distance=196.19CM
distance=195.29CM
```

实验现象：软件 Shell 控制台输出超声波检测的距离，如下所示：

The screenshot shows the 'Shell' window of the software. It displays a series of distance measurements in centimeters, indicating the output of the ultrasonic sensor. The text is repeated twice for each measurement.

```
Shell x
distance=6.24CM
distance=6.32CM
distance=6.03CM
distance=194.85CM
distance=194.83CM
```

## 课后作业

## 第 22 章 红外遥控实验

红外遥控在电子产品中非常常见，比如电视、空调、冰箱、智能小车等都有应用，因此学习红外遥控对后续完成电子制作非常有意义。本章学习使用 MicroPython 对红外接收数据解码。本章分为如下几部分内容：

- 22.1 实验介绍
- 22.2 硬件设计
- 22.3 软件设计
- 22.4 实验现象

## 22.1 实验介绍

### 22.1.1 实验简介

红外遥控是一种无线、非接触控制技术，具有抗干扰能力强，信息传输可靠，功耗低，成本低，易实现等显著优点。由于红外线遥控不具有像无线电遥控那样穿过障碍物去控制被控对象的能力，所以不用担心串码问题。

红外遥控通信系统一般由红外发射装置和红外接收设备两大部分组成。

#### (1) 红外发射装置

红外发射装置，也就是通常我们说的红外遥控器，如下所示：



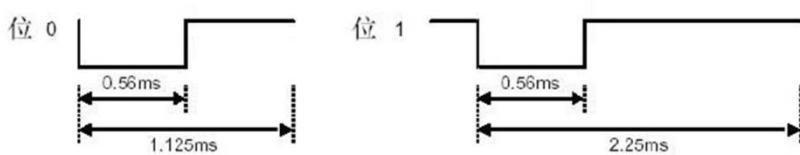
通常的红外遥控器是将遥控信号（二进制脉冲码）调制在 38KHz 的载波上，经缓冲放大后送至红外发光二极管，转化为红外信号发射出去的。

二进制脉冲码的形式有多种，其中最为常用的是 NEC Protocol 的 PWM 码（脉冲宽度调制）和 Philips RC-5 Protocol 的 PPM 码（脉冲位置调制码，脉冲串之间的时间间隔来实现信号调制）。如果要开发红外接收设备，一定要知道红外遥控器的编码方式和载波频率，我们才可以选取一体化红外接收头和制定解码方案。我们配套的红外遥控器使用的是 NEC 协议，其特征如下：

- 1、8 位地址和 8 位指令长度；
- 2、地址和命令 2 次传输（确保可靠性）
- 3、PWM 脉冲位置调制，以发射红外载波的占空比代表“0”和“1”；
- 4、载波频率为 38Khz；
- 5、位时间为 1.125ms 或 2.25ms；

NEC 码的位定义：一个脉冲对应 560us 的连续载波，一个逻辑 1 传输需要 2.25ms (560us 脉冲+1680us 低电平)，一个逻辑 0 的传输需要 1.125ms (560us

脉冲+560us 低电平）。而红外接收头在收到脉冲的时候为低电平，在没有脉冲的时候为高电平，这样，我们在接收头端收到的信号为：逻辑 1 应该是 560us 低+1680us 高，逻辑 0 应该是 560us 低+560us 高。所以可以通过计算高电平时间判断接收到的数据是 0 还是 1。NEC 码位定义时序图如下图所示：



NEC 遥控指令的数据格式为：引导码、地址码、地址反码、控制码、控制反码。引导码由一个 9ms 的低电平和一个 4.5ms 的高电平组成，地址码、地址反码、控制码、控制反码均是 8 位数据格式。按照低位在前，高位在后的顺序发送。采用反码是为了增加传输的可靠性（可用于校验）。数据格式如下：



NEC 码还规定了连发码(由 9ms 低电平+2.5m 高电平+0.56ms 低电平 +97.94ms 高电平组成)，如果在一帧数据发送完毕之后，红外遥控器按键仍然没有放开，则发射连发码，可以通过统计连发码的次数来标记按键按下的长短或次数。

## (2) 红外接收设备

红外遥控接收器的主要作用是将遥控发射器发来的红外光信号转换成电信号，再放大、限幅、检波、整形，形成遥控指令脉冲，输出至遥控微处理器。近几年不论是业余制作还是正式产品，大多都采用成品红外接收头。成品红外接收头的封装大致有两种：一种采用铁皮屏蔽；一种是塑料封装。均有三只引脚，即电源正（VDD）、电源负（GND）和数据输出（VOUT）。其外观实物图如下图所示：



正对接收头的凸起处看，从左至右，管脚依次是 1: VOUT, 2: GND, 3: VDD。

由于红外接收头在没有脉冲的时候为高电平，当收到脉冲的时候为低电平，所以可以通过外部中断的下降沿触发中断，在中断内通过计算高电平时间来判断接收到的数据是 0 还是 1。

## 22.1.2 实验目的

解码红外遥控数据，并将控制码通过软件 Shell 控制台输出。

## 22.1.3 MicroPython 函数使用

本实验通过外部中断下降沿触发对红外遥控解码，因此外部中断模块使用可参考前面章节。

外部中断也是通过 machine 模块的 Pin 子模块来配置，先来看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。 <code>id</code> :引脚编号； <code>mode</code> :输入输出方式； <code>pull</code> :上下拉电阻配置。
使用方法
<code>KEY.irq(handler,trigger)</code>
配置中断模式。 <code>handler</code> :中断执行的回调函数； <code>trigger</code> :触发中断的方式，共 4 种，分别是 <code>Pin.IRQ_FALLING</code> （下降沿触发）、 <code>Pin.IRQ_RISING</code> （上升沿触发）、 <code>Pin.IRQ_LOW_LEVEL</code> （低电平触发）、 <code>Pin.IRQ_HIGH_LEVEL</code> （高电平触发）。

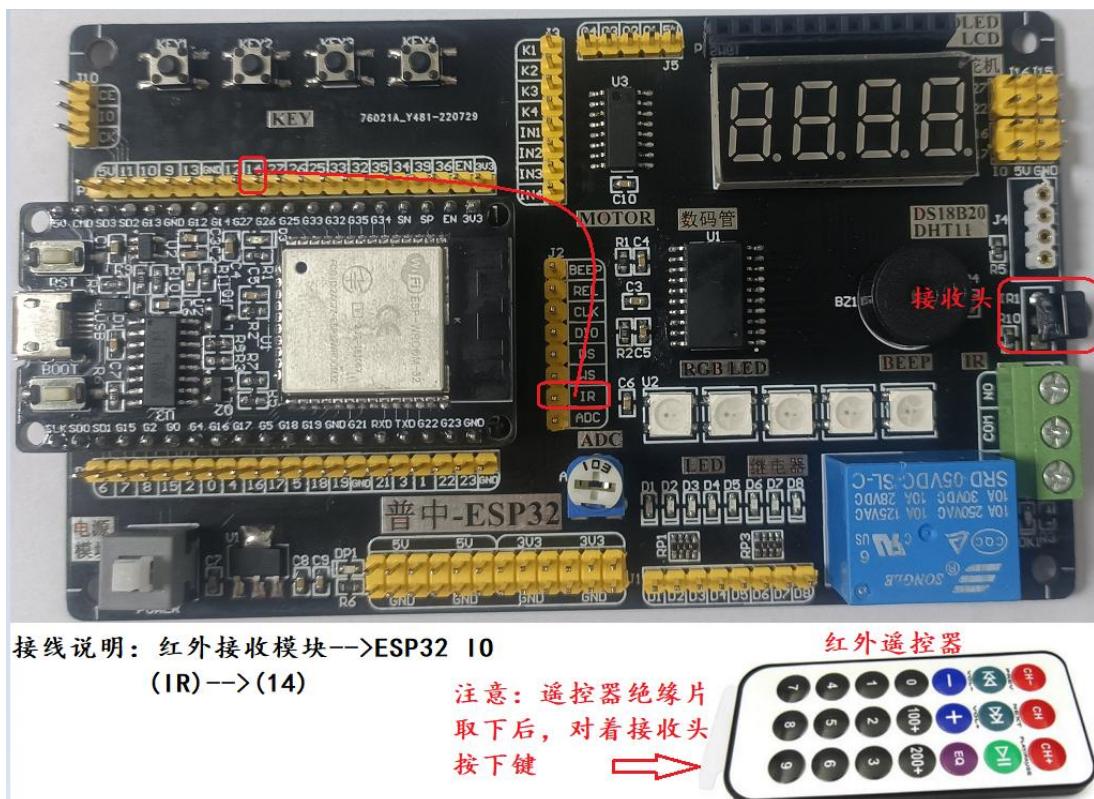
## 22.2 硬件设计

本实验使用到硬件资源如下：

- (1) 红外遥控和红外接收头模块

## (2) ESP32 GPIO

本章实验使用 ESP32 的 IO14 引脚，接线如下所示：



## 22.3 软件设计

下面我们打开“\4--实验程序\1--MicroPython 实验\1--基础实验\21-红外遥控实验”程序，控制代码全部都在 main.py 中，代码如下：

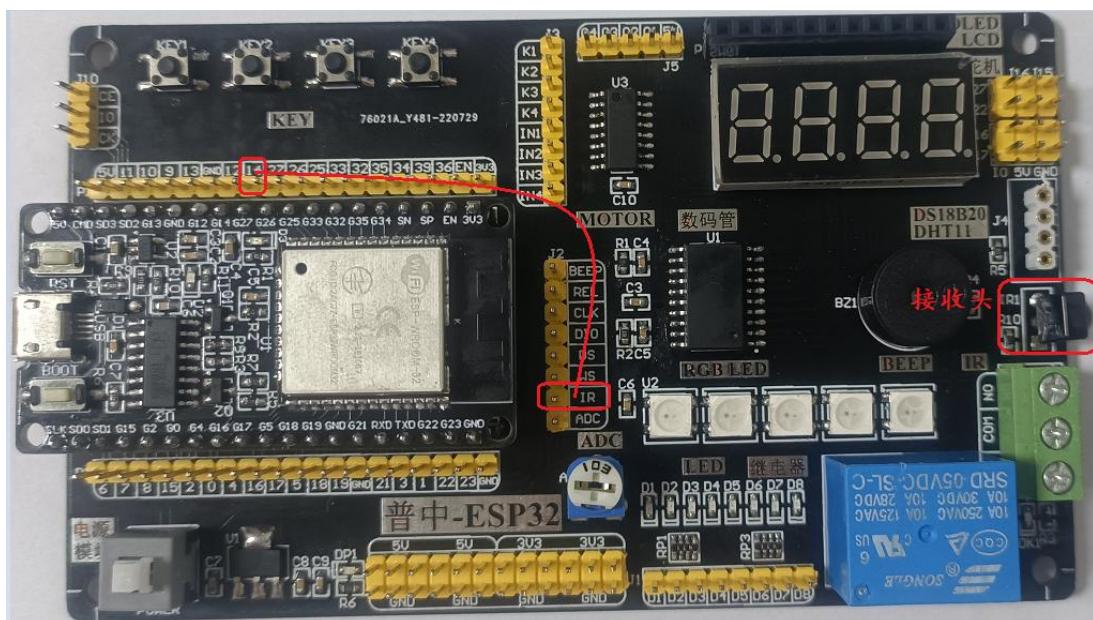
```
main.py
1  ...
2  深圳市普中科技有限公司 (PRECHIN 普中)
3  技术支持: www.prechin.net
4
5  实验名称: 红外遥控实验
6  接线说明: 红外接收模块-->ESP32 IO
7  (IR)-->(14)
8
9  实验现象: 程序下载成功后, 当按下遥控器键时, 软件shell控制台输出红外遥控器控制码 (十六进制数)
10
11 注意事项:
12
13  ...
14
15 #导入Pin模块
16 from machine import Pin
17 import time
18
19 #定义IRED控制对象
20 ired=Pin(14,Pin.IN,Pin.PULL_UP)
21
22 #存储红外遥控器键值
23 gired_data=[0,0,0,0]
24
```

```
25 #外部中断函数
26 def ired_irq(ired):
27     ired_high_time=0 #保存高电平时间，鉴别数据1还是0
28
29     if ired.value()==0:
30         time_cnt=1000
31         while (not ired.value()) and time_cnt: #等待引导信号9ms低电平结束，若超过10ms强制退出
32             time.sleep_us(10)
33             time_cnt-=1
34             if time_cnt==0:
35                 return
36
37     if ired.value()==1: #引导信号9ms低电平已过，进入4.5ms高电平
38         time_cnt=500
39         while ired.value() and time_cnt: #等待引导信号4.5ms高电平结束，若超过5ms强制退出
40             time.sleep_us(10)
41             time_cnt-=1
42             if time_cnt==0:
43                 return
44     for i in range(4): #循环4次，读取4个字节数据
45         for j in range(8): #循环8次读取每位数据即一个字节
46             time_cnt=600
47             while (ired.value()==0) and time_cnt: #等待数据1或0前面的0.56ms结束，若超过6ms强制退出
48                 time.sleep_us(10)
49                 time_cnt-=1
50                 if time_cnt==0:
51
52                     return
53     time_cnt=20
54     while ired.value()==1: #等待数据1或0后面的高电平结束，若超过2ms强制退出
55         time.sleep_us(10)
56         ired_high_time+=1
57         if ired_high_time>20:
58             return
59     gired_data[i]>>=1 #先读取的为低位，然后是高位
60     if ired_high_time>=8: #如果高电平时间大于0.8ms，数据则为1，否则为0
61         gired_data[i]|=0x80
62     ired_high_time=0 #重新清零，等待下一次计算时间
63     if gired_data[2]!~gired_data[3]: #校验控制码与反码，错误则返回
64         for i in range(4):
65             gired_data[i]=0
66         return
67
68     print("红外遥控器操作码: 0x%02X" % gired_data[2])
69
70 #程序入口
71 if __name__=="__main__":
72     ired.irq(ired_irq,Pin.IRQ_FALLING)
73
74     while True:
75         pass
76
```

通过外部中断下降沿触发，对红外遥控解码，按照 NEC 时序数据格式解码。

## 22.4 实验现象

下载程序前，按照如下接线：



接线说明：红外接收模块-->ESP32 IO

(IR)-->(14)

注意：遥控器绝缘片  
取下后，对着接收头  
按下键

红外遥控器



将程序下载到开发板内（可参考“2.2.5 程序下载运行”章节），实验现象：

软件 Shell 控制台上显示红外遥控器按下控制码。如下：

```
Shell
红外遥控器操作码: 0x40
红外遥控器操作码: 0x46
红外遥控器操作码: 0x46
红外遥控器操作码: 0x46
红外遥控器操作码: 0x46
```

红外遥控器从上到下，从左到右对应键值表：

0X45	0X46	0X47
0X44	0X40	0X43
0X07	0X15	0X09
0X16	0X19	0X0D
0X0C	0X18	0X5E
0X08	0X1C	0X5A
0X42	0X52	0X4A

## 课后作业

© PRECHIN

## 第 23 章 舵机实验

舵机在电子产品中非常常见，比如机器人、智能小车等都有应用，因此学习舵机对后续完成电子制作非常有意义。本章学习使用 MicroPython 对 SG90 舵机旋转角度控制。本章分为如下几部分内容：

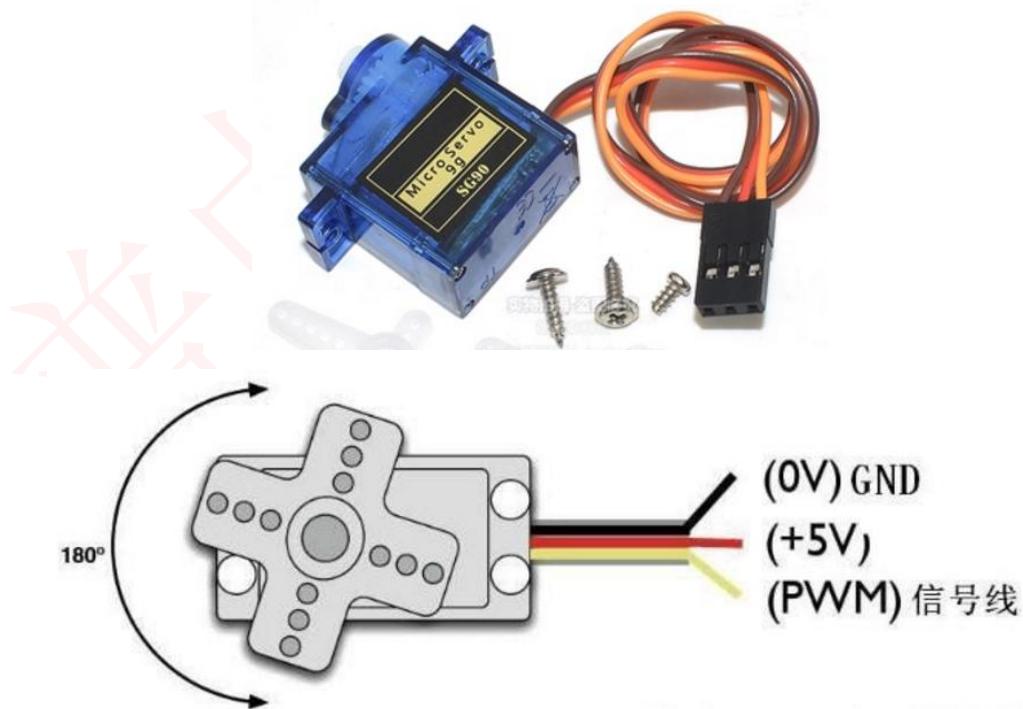
- 23.1 实验介绍
- 23.2 硬件设计
- 23.3 软件设计
- 23.4 实验现象

## 23.1 实验介绍

### 23.1.1 实验简介

舵机是一种位置（角度）伺服的驱动器，适用那些需要角度不断变化并可以保持的控制系统。舵机只是一种通俗的叫法，其实质是一个伺服马达。舵机主要分为模拟舵机和数字舵机。模拟舵机：需要不断的发送目的地 PWM 信号，才能旋转到指定位置。例如：我现在让它旋转 90 度，我就需要不断的发送 90 度的 PWM 信号直到到达指定位置才能停止。数字舵机：只需给一个目的地 PWM 信号，即可旋转到指定位置。例如：我现在让它旋转 90 度，我只需要发送一次 90 度的 PWM 信号，它就可以旋转到 90 度。

SG90 模拟舵机在市面上十分常见，价格也比较便宜。常用于航模，机器人或智能小车等。如下图所示：



一个舵机有三条线：VCC（红线）、GND（棕色线）和信号线（橙色线）。只要通过信号线给予规定的控制信号即可实现舵机码盘的转动。

#### （1）模块主要电气参数

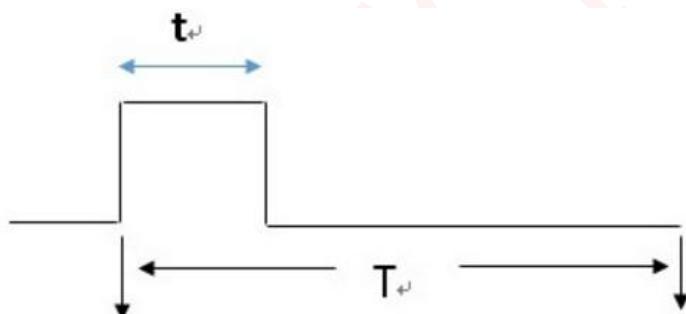
- 使用电压: 4.8V-6V
- 尺寸: 221.5mmX11.8mmX22.7mm
- 重量: 9g
- 角度范围: 0-180°

## (2) 工作原理

舵机内部有一个基准电压，微处理器产生的 PWM 信号通过信号线进入舵机产生直流偏置电压，与舵机内部的基准电压作比较，获得电压差输出。电压差的正负输出到电机驱动芯片上，从而决定正反转。当舵机开始旋转的时候，舵机内部通过级联减速齿轮带动电位器旋转，使得电压差为零，电机停止转动。

我们无需了解其内部构造，只需知道如何通过 PWM 控制其转动即可。

①我们需要使用 PWM 产生周期为 20ms，高电平  $t$  等于 0.5ms-2.5ms 之间的这样一个方波。可以使用 ESP32 的 PWM 功能产生这样的方波。波形如下所示：



高电平在一个周期 (20ms) 的持续时间对应的舵机角度，如下图所示：

高电平 $t$ 占整个周期 $T$ (20ms) 的时间	舵机旋转的角度
0.5ms	0度
1ms	45度
1.5ms	90度
2ms	135度
2.5ms	180度

### 23.1.2 实验目的

控制 SG90 舵机以 45° 步进循环从 0° 旋转到 180°。

### 23.1.3 MicroPython 函数使用

MicroPython 固件库内并没有集成 SG90 模块，因此需要我们自己实现，对于不了解 SG90 工作原理的用户来说，要编写出驱动是困难的。MicroPython 拥有着庞大的用户群，自然 SG90 模块也有开源的代码，直接拿过来使用即可，这就是使用 MicroPython 开发的高效之处，市面上常见的模块在网上几乎都可以找到相应的模块代码，大家一定要善于在网上搜索资源。我们已将 SG90 舵机模块驱动程序 servo.py 下载好，放入工程中。使用方法如下：

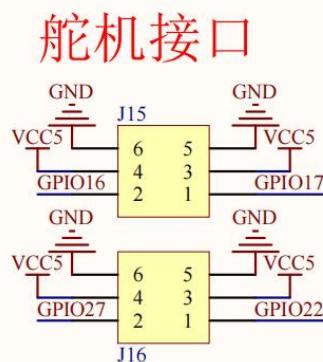
构造函数
my_servo = servo.Servo(Pin(17))
构建舵机对象。定义舵机连接的信号引脚
使用方法
my_servo.write_angle(degrees)
控制舵机旋转角度，范围0-180

## 23.2 硬件设计

本实验使用到硬件资源如下：

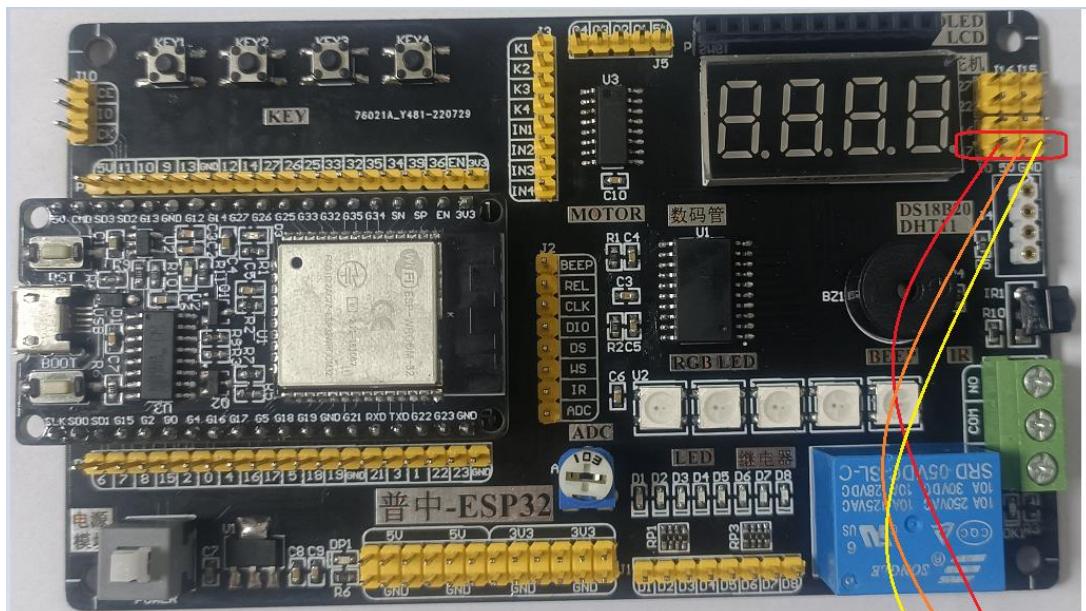
- (1) SG90 舵机模块
- (2) ESP32 GPIO

舵机接口电路如下：



从上图可知，板载 4 路舵机接口均连接到指定 IO。

本章实验使用 ESP32 的 IO17 引脚，接线如下所示：



接线说明: SG90舵机模块-->ESP32 IO

- 橙色(信号线)-->(17)
- 红色(电源正)-->(5V)
- 褐色(电源负)-->(GND)

SG90舵机



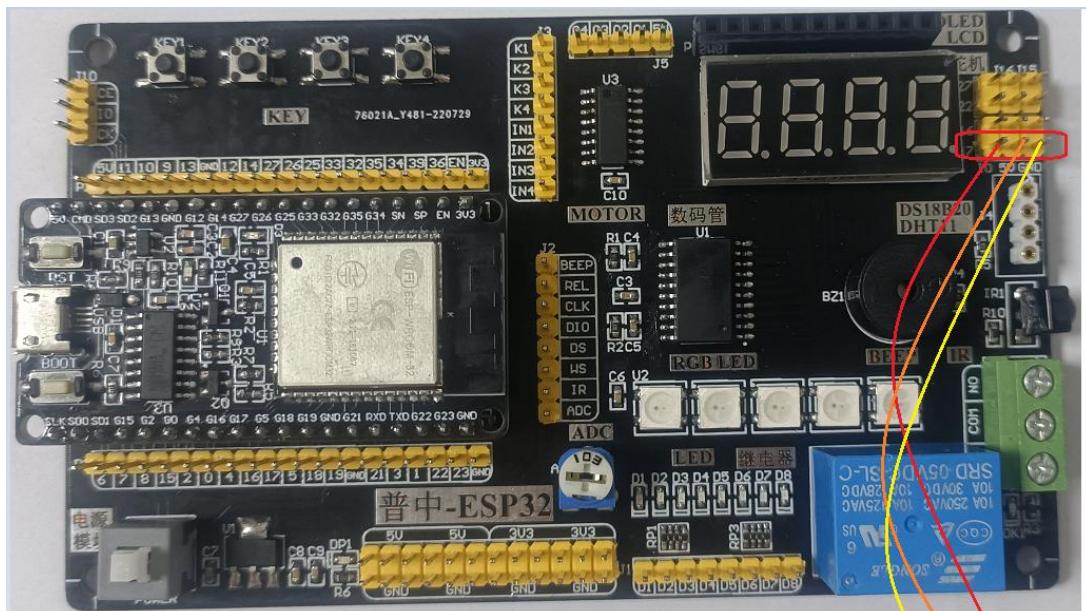
### 23.3 软件设计

下面我们打开“\4--实验程序\1--MicroPython 实验\1--基础实验\22-舵机实验”程序，舵机驱动在 servo.py 中，该代码不展示，用户可打开工程查看。控制代码全部都在 main.py 中，代码如下：

```
main.py * servo.py
1 ...
2 深圳市普中科技有限公司 (PRECHIN 普中)
3 技术支持: www.prechin.net
4
5 实验名称: 舵机实验
6 接线说明: SG90舵机模块-->ESP32 IO
7 橙色(信号线)-->(17)
8 红色(电源正)-->(5V)
9 褐色(电源负)-->(GND)
10
11 实验现象: 程序下载成功后, SG90舵机循环以45°步进从0°旋转到180°
12
13 注意事项:
14 ...
15 ...
16
17 #导入Pin模块
18 from machine import Pin
19 import time
20 from servo import Servo
21
22 #定义SG90舵机控制对象
23 my_servo = Servo(Pin(17))
24
25 #程序入口
26 if __name__=="__main__":
27
28     while True:
29         my_servo.write_angle(0)  #角度0°
30         time.sleep(0.5)
31         my_servo.write_angle(45) #角度45°
32         time.sleep(0.5)
33         my_servo.write_angle(90) #角度90°
34         time.sleep(0.5)
35         my_servo.write_angle(135) #角度135°
36         time.sleep(0.5)
37         my_servo.write_angle(180) #角度180°
38         time.sleep(0.5)
39
```

## 23.4 实验现象

下载程序前, 按照如下接线:



接线说明：SG90舵机模块-->ESP32 IO

橙色(信号线)-->(17)

红色(电源正)-->(5V)

褐色(电源负)-->(GND)

SG90舵机



将程序下载到开发板内（可参考“2.2.5 程序下载运行”章节），注意：因为本实验有 2 个程序模块，即 2 个 .py 文件，而程序是从 main.py 开始运行的，其中又调用了 servo.py 内容，因此要先将该文件上载开发板中，然后可在 main.py 文件下点击“运行”或选择“运行当前脚本”。如下所示：

The screenshot shows a MicroPython code editor interface. On the left, there are two panes: '此电脑' (This Computer) and 'MicroPython设备' (MicroPython Device). In the '此电脑' pane, the file 'main.py' is selected and highlighted with a red circle. In the 'MicroPython设备' pane, the file 'servo.py' is selected and highlighted with a red circle. A red arrow points from the 'servo.py' file in the device pane to the text '①上载到开发板内' (Upload to development board) located in the center of the screen. At the top of the editor, there is a toolbar with icons for file operations, and a status bar at the bottom indicates 'MicroPython v1.19.1 on 2022-06-18; ESP32 module with ESP32'.

②在main.py文件下运行

①上载到开发板内

```
1  ...
2 深圳市普中科技有限公司 (PRECHIN 普中)
3 技术支持: www.prechin.net
4
5 实验名称: 舵机实验
6 接线说明: SG90舵机模块-->ESP32 IO
7     橙色(信号线)-->(17)
8     红色(电源正)-->(5V)
9     褐色(电源负)-->(GND)
10
11 实验现象: 程序下载成功后, SG90舵机循环以45°步进从0°旋转到180°
12
13 注意事项:
14
15 ...
16
17 #导入Pin模块
18 from machine import Pin
19 import time
20 from servo import Servo
21
22 #定义SG90舵机控制对象
23 my_servo = Servo(Pin(17))
24
25 #程序入口
26 if __name__=="__main__":
27
```

Shell <

```
MicroPython v1.19.1 on 2022-06-18; ESP32 module with ESP32
Type "help()" for more information.
>>> %Run -c $EDITOR_CONTENT
```

实验现象: SG90 舵机以 45° 步进循环从 0° 旋转到 180° 。

## 课后作业

## 第 24 章 OLED 液晶显示实验

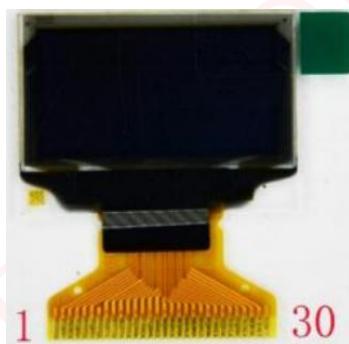
前面我们学习过数码管显示，但显示数据内容和数量很少，面对需要多种数据字符显示的应用显然不适合，因此就引出本章 OLED 液晶显示。本章学习使用 MicroPython 的 I2C 控制 OLED 液晶显示字符、点、线、矩形、填充矩形及滚动。通过本章学习，大家可以尝试将之前在 Shell 控制台显示的数据信息放在 OLED 上显示。本章分为如下几部分内容：

- 24.1 实验介绍
- 24.2 硬件设计
- 24.3 软件设计
- 24.4 实验现象

## 24.1 实验介绍

### 24.1.1 实验简介

OLED，即有机发光二极管（Organic Light Emitting Diode）。OLED 由于同时具备自发光，不需背光源、对比度高、厚度薄、视角广、反应速度快、可用于挠曲性面板、使用温度范围广、构造及制程较简单等优异之特性，被认为是下一代的平面显示器新兴应用技术。LCD 都需要背光，而 OLED 不需要，因为它是自发光的。这样同样的显示 OLED 效果要来得更好一些。以目前的技术，OLED 的尺寸还难以大型化，但是分辨率确可以做到很高。我们使用的是 0.96 寸 OLED 显示屏，内部驱动芯片是 SSD1306，如下图所示：



该屏有以下特点：

- 0.96 寸 OLED 有黄蓝，白，蓝三种颜色可选；其中黄蓝是屏上 1/4 部分为黄光，下 3/4 为蓝；而且是固定区域显示固定颜色，颜色和显示区域均不能修改；白光则为纯白，也就是黑底白字；蓝色则为纯蓝，也就是黑底蓝字。
- 分辨率为 128\*64
- 多种接口方式；OLED 裸屏总共种接口包括：6800、8080 两种并行接口方式、3 线或 4 线的串行 SPI 接口方式、IIC 接口方式（只需要 2 根线就可以控制 OLED），这五种接口是通过屏上的 BS0-BS2 来配置的。

本教程使用的是 0.96 寸 OLED（IIC 接口）模块，如下所示：



管脚功能介绍：

GND：电源地；VDD：电源正（3-5.5V）；SCK：I2C 时钟管脚；SDA：I2C 数据管脚；

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。也就是说不同设备间通过这两根线就可以进行通信。

ESP32 有 2 硬件 I2C 接口和 N 个软件 I2C 接口，硬件 I2C 总线默认的 IO 如下：

	I2C(0)	I2C(1)
scl	18	25
sda	19	26

硬件 I2C 接口通过配置可在任意 IO 口使用。软件 I2C 接口可通过配置在任意 IO 口使用，相当于使用 IO 口模拟 I2C 时序。本实验使用软件 I2C 来与 OLED 通信。

### 24.1.2 实验目的

使用 MicroPython 的软件 I2C 控制 OLED 液晶显示字符、点、线、矩形、填充矩形及滚动。

### 24.1.3 MicroPython 函数使用

本实验需要使用到 MicroPython 的 machine 模块来定义 Pin 口和 I2C 初始化。具体如下：

构造函数
i2c = SoftI2C(sda=Pin(23), scl=Pin(18))
构建软件I2C对象。定义时钟和数据引脚
i2c = I2C(0, sda=Pin(19), scl=Pin(18), freq=400000)
构建硬件I2C对象。定义I2C序号和时钟、数据引脚，默认频率为400000HZ，可省略
使用方法
i2c.scan()
扫描 I2C 总线的设备。返回地址，如：0x3c；
i2c.readfrom(addr, nbytes)
从指定地址读数据。addr:指定设备地址; nbytes:读取字节数;
i2c.writeto(addr, buf)
从指定地址写数据。addr:指定设备地址; buf:写入数据;
*其它更多用法请阅读 MicroPython 文档： <a href="http://docs.micropython.org/en/latest/library/machine.I2C.html?highlight=i2c#machine.I2C">http://docs.micropython.org/en/latest/library/machine.I2C.html?highlight=i2c#machine.I2C</a>

其它更多用法请阅读 MicroPython 文档：

<http://docs.micropython.org/en/latest/library/machine.I2C.html?highlight=i2c#machine.I2C>

定义好 I2C 后，还需要驱动一下 OLED。

MicroPython 固件库内并没有集成 SSD1306 驱动模块，因此需要我们自己实现，对于不了解 I2C 总线时序和 SSD1306 命令的用户来说，要编写出驱动是困难的。MicroPython 拥有着庞大的用户群，自然 SSD1306 模块也有开源的代码，直接拿过来使用即可，这就是使用 MicroPython 开发的高效之处，市面上常见的模块在网上几乎都可以找到相应的模块代码，大家一定要善于在网上搜索资源。我们已将 SSD1306 模块驱动程序 ssd1306.py 下载好，放入工程中。使用方法如下：

构造函数
oled = SSD1306_I2C(width, height, i2c, addr)
构建OLED 显示屏对象。width:屏幕宽像素; height: 屏幕高像素; i2c:定义好的I2C 对象; addr:显示屏设备地址（默认设置，可省略）。
使用方法
oled.text(s, x, y, c)
将字符写在指定为位置。s: 字符; x:横坐标; y:纵坐标; c: 颜色（1白色, 0黑色）。
oled.show()
执行显示
oled.fill(c)
清屏。RGB: 0 表示黑色, 1 表示白色。
oled.pixel(x, y, c)
指定位置画点。x, y横纵坐标; c: 颜色（1白色, 0黑色）
oled.hline(x, y, w, c)
指定长度画横线。x, y横纵坐标; w: 长度; c: 颜色（1白色, 0黑色）
oled.vline(x, y, h, c)
指定长度画竖线。x, y横纵坐标; h: 长度; c: 颜色（1白色, 0黑色）
oled.line(x1, y1, x2, y2, c)
指定2点坐标画直线。x1, y1坐标1; x2, y2坐标2; c: 颜色（1白色, 0黑色）
oled.rect(x, y, w, h, c)
指定坐标画矩形。x, y坐标; w, h矩形宽和高; c: 颜色（1白色, 0黑色）
oled.fill_rect(x, y, w, h, c)
指定坐标画填充矩形。x, y坐标; w, h矩形宽和高; c: 颜色（1白色, 0黑色）
oled.scroll(xstep, ystep)
指定x和y值移动。xstep, ystep移动位置

其它更多用法请阅读 MicroPython 文档：

<http://docs.micropython.org/en/latest/library/framebuf.html?highlight=framebuf#module-framebuf>

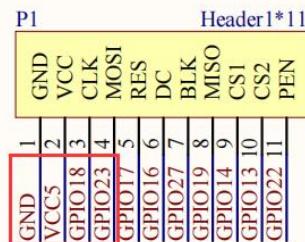
## 24.2 硬件设计

本实验使用到硬件资源如下：

- (1) 0.96 寸 IIC 接口 OLED 模块
- (2) ESP32 GPIO

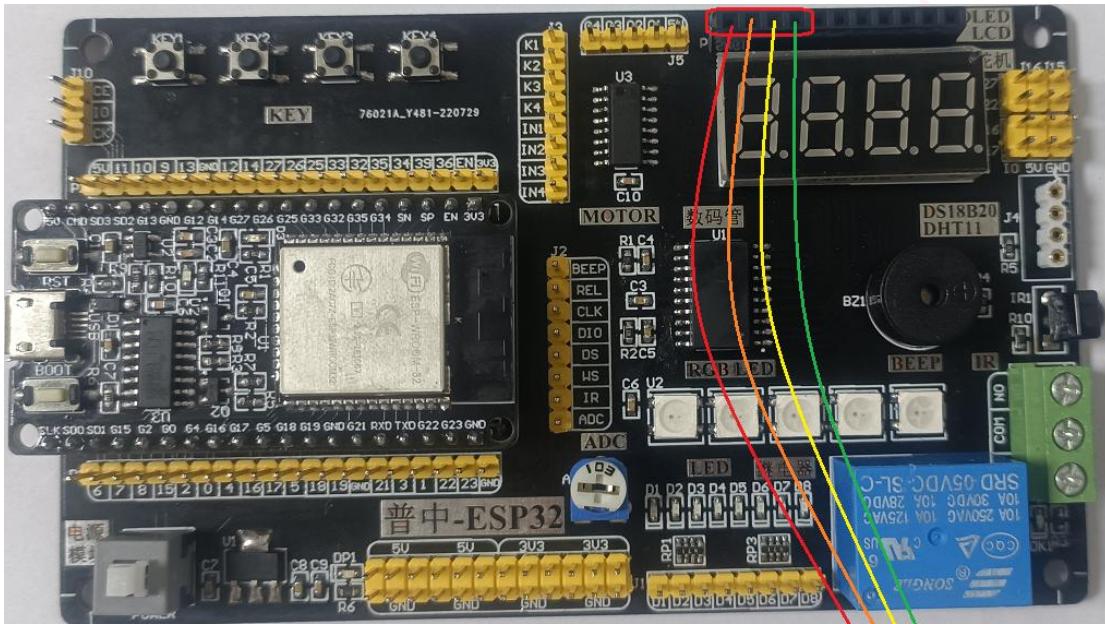
OLED 接口电路如下：

## TFTLCD&OLED



从上图可知，板载 4 路舵机接口均连接到指定 IO。

本章实验使用 ESP32 的 IO18、23 引脚，接线如下所示：



接线说明：OLED (I<sup>2</sup>C) 液晶模块-->ESP32 IO

- GND-->(GND)
- VCC-->(5V)
- SCL-->(18)
- SDA-->(23)



### 24.3 软件设计

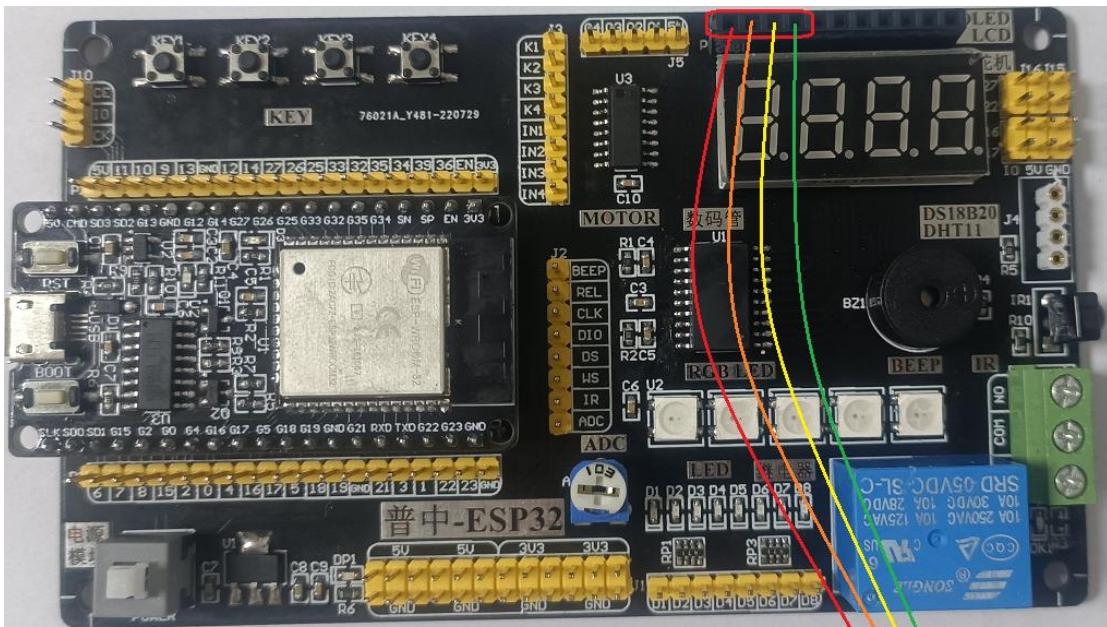
下面我们打开“\4--实验程序\1--MicroPython 实验\1--基础实验\23-OLED 液晶显示实验”程序，OLED 驱动在 ssd1306.py 中，该代码不展示，用户可打开工程查看。控制代码全部都在 main.py 中，代码如下：

```
main.py x ssd1306.py x
1 ...
2 深圳市普中科技有限公司 (PRECHIN 普中)
3 技术支持: www.prechin.net
4
5 实验名称: 航机实验
6 接线说明: OLED(IIC)液晶模块-->ESP32 IO
7     GND-->(GND)
8     VCC-->(5V)
9     SCL-->(18)
10    SDA-->(23)
11
12 实验现象: 程序下载成功后, OLED液晶屏显示字符信息
13
14 注意事项:
15
16 ...
17
18 #导入Pin模块
19 from machine import Pin
20 import time
21 from machine import SoftI2C
22 from ssd1306 import SSD1306_I2C #I2C的oled选该方法
23
24 #创建硬件I2C对象
25 #i2c=I2C(0,sda=Pin(19), scl=Pin(18), freq=400000)
26
27 #创建软件I2C对象
28 i2c = SoftI2C(sda=Pin(23), scl=Pin(18))
29 #创建OLED对象, OLED分辨率、I2C接口
30 oled = SSD1306_I2C(128, 64, i2c)
31
32 #程序入口
33 if __name__=="__main__":
34     oled.fill(0) #清空屏幕
35     oled.show() #执行显示
36
37     oled.text("Hello World!",0,0,1) #显示字符串
38     oled.show() #执行显示
39
40     oled.pixel(10,20,1) #显示一个像素点
41     oled.hline(0,10,100,1) #画横线
42     oled.vline(120,0,30,1) #画竖线
43     oled.line(10,40,100,60,1) #画指定坐标直线
44     oled.rect(50,20,50,30,1) #画矩形
45     oled.fill_rect(60,30,30,20,1) #画填充矩形
46     oled.show() #执行显示
47
48     time.sleep(2)
49     oled.fill(0) #清空屏幕
50     oled.text("Hello World!",0,0,1)
51     oled.show() #执行显示
52
53     time.sleep(1)
54
55     oled.scroll(10,0) #指定像素X轴移动
56     oled.fill_rect(0,0,10,8,0) #清除移动前显示区
57     oled.show() #执行显示
58     time.sleep(1)
59
60     oled.scroll(0,10) #指定像素Y轴移动
61     oled.fill_rect(0,0,128,10,0) #清除移动前显示区
62     oled.show() #执行显示
63     while True:
64         pass
```

上述代码中 OLED 的 I2C 地址默认是 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。

## 24.4 实验现象

下载程序前，按照如下接线：



接线说明：OLED(IIC)液晶模块-->ESP32 IO

GND-->(GND)

VCC-->(5V)

SCL-->(18)

SDA-->(23)



将程序下载到开发板内（[可参考“2.2.5 程序下载运行”章节](#)），注意：因为本实验有2个程序模块，即2个.py文件，而程序是从main.py开始运行的，其中又调用了ssd1306.py内容，因此要先将该文件上载开发板中，然后可在main.py文件下点击“运行”或选择“运行当前脚本”。如下所示：

①上载到开发板内

②在main.py文件下运行

```
1 深圳市普中科技有限公司（PRECHIN 普中）
2 技术支持: www.prechin.net
3
4 实验名称: 舵机实验
5 接线说明: OLED(IIC)液晶模块-->ESP32 IO
6 GND-->(GND)
7 VCC-->(5V)
8 SCL-->(18)
9 SDA-->(23)
10
11 实验现象: 程序下载成功后, OLED液晶屏显示字符信息
12
13 注意事项:
14 ...
15 ...
16
17 #导入Pin模块
18 from machine import Pin
19 import time
20 from machine import SoftI2C
21 from ssd1306 import SSD1306_I2C #I2C的oled选该方法
22
23 #创建硬件I2C对象
24 #i2c=I2C(0,sda=Pin(19), scl=Pin(18), freq=400000)
25
26 #创建软件I2C对象
27
```

Shell <

```
MicroPython v1.19.1 on 2022-06-18; ESP32 module with ESP32
Type "help()" for more information.
>>> %Run -c $EDITOR_CONTENT
```

实验现象: OLED 液晶屏上显示字符、点、线、矩形、填充矩形及滚动。

## 课后作业

## 第 25 章 SD 卡实验

在嵌入式系统中，经常需要存储大量的数据或图片，使用 SD 卡作为大容量存储设备是一个非常好的选择，我们开发板自带一个 SD 卡接口（可直接插 TF 卡）。本章学习使用 MicroPython 控制 SD 卡读写文件。本章分为如下几部分内容：

- 25.1 实验介绍
- 25.2 硬件设计
- 25.3 软件设计
- 25.4 实验现象

## 25.1 实验介绍

### 25.1.1 实验简介

在嵌入式系统中，经常需要存储大量的数据和图片，使用 SD 卡作为大容量存储设备是一个非常好的选择。比如系统想要存储一天采集的温湿度数据，ESP32 自带的存储容量是很小的，无法保存大容量数据，此时可使用外部 SD 卡，通过 ESP32 控制将采集的温湿度数据写入到 SD 卡文件内。又比如，使用 ESP32 制作一个电子相册或 MP3，需要读取图片或音乐文件，依靠 ESP32 自身内存是实现不了的，此时可使用外部 SD 卡，通过 ESP32 将图片或音频数据写入到彩屏上显示。

开发板已板载 TF 卡座，可使用 TF 卡插入使用。如下：（SD 卡容量大小可任意，本实验测试使用的是 4G 内存）



ESP32-WROOM-32 可使用 SPI 口与 SD 卡通信，这在 MicroPython 已集成 SPI 模块可直接使用。

SPI 的全称是“Serial Peripheral Interface”，意为串行外围接口。SPI 是一种高速的，全双工，同步的通信总线，并且在芯片的管脚上只占用四根线，节约了芯片的管脚，同时为 PCB 的布局上节省空间，提供方便，正是出于这种简单易用的特性，如今越来越多的芯片集成了这种通信协议。

SPI 接口一般使用 4 条线通信，事实上只需 3 条线也可以进行 SPI 通信（单向传输时），其中 3 条为 SPI 总线（MISO、MOSI、SCLK），一条为 SPI 片选信号线（CS）。

ESP32 有两个硬件 SPI 通道允许更高速率传输（到达 80MHz）。也可以配置成任意引脚，但相关引脚要符合输入输出的方向性。通过自定义引脚而非默认引脚，会降低传输速度，上限为 40MHz。以下是硬件 SPI 总线默认引脚：

	HSPI (id=1)	VSPI (id=2)
sck	14	18
mosi	13	23
miso	12	19

id=1 的 HSPI 口已连接到模块 SPI-FLASH，不要去使用。因此可使用 id=2 的 SPI 与 SD 通信。

ESP32 还可以使用软件 SPI，可配置到所有可用引脚。有关 SPI 使用可参考 MicroPython 文档：

<http://docs.micropython.org/en/latest/library/machine.SPI.html#machine.SPI>

### 25.1.2 实验目的

对 SD 卡指定路径文件进行读写，并将读取信息在软件 Shell 控制台输出。

### 25.1.3 MicroPython 函数使用

本实验需要使用到 MicroPython 的 machine 模块来定义 Pin 口和 SPI 初始化。具体如下：

构造函数
<code>spi = SPI(id, baudrate=500000, polarity=0, phase=0, bits=8, firstbit=MSB, sck=None, mosi=None, miso=None)</code>
构建硬件SPI对象。id: 选择SPI号; baudrate/polarity/phase/bits/firstbit: 为SPI通信参数, 已有默认值, 可省略。 sck、mosi、miso: SPI通信引脚;
<code>spi = SPI(baudrate=500000, polarity=0, phase=0, bits=8, firstbit=MSB, sck=None, mosi=None, miso=None)</code>
构建软件SPI对象。baudrate/polarity/phase/bits/firstbit: 为SPI通信参数, 已有默认值, 可省略。 sck、mosi、miso: SPI通信引脚, 可任意脚;
使用方法
<code>spi.read(nbytes, write)</code>
读取指定nbytes字节数据, 同时输出数据write
<code>spi.write(buf)</code>
写数据

其它更多用法请阅读 MicroPython 文档：

<http://docs.micropython.org/en/latest/library/machine.SPI.html#machine.SPI>

定义好 SPI 后，还需要驱动一下 SD 卡。

MicroPython 固件库内集成的 SD 卡模块为固定 IO，无法修改，考虑到片选 CS 管脚能够更灵活使用，因此需要我们自己实现，对于不了解 SD 卡底层和命令的用户来说，要编写出驱动是困难的。MicroPython 拥有着庞大的用户群，自然 SD 模块也有开源的代码，直接拿过来使用即可，这就是使用 MicroPython 开发的高效之处，市面上常见的模块在网上几乎都可以找到相应的模块代码，大家一定要善于在网上搜索资源。我们已将 SD 模块驱动程序 `sdcards.py` 下载好，放入工程中。使用方法如下：

构造函数
<code>sd = sdcards.SDCard(spi, cs)</code>
构建SD卡对象。spi对象，cs片选管脚
使用方法
<code>os.mount(sd, "/sd")</code>
挂载SD卡
<code>os.listdir("/sd")</code>
输出SD卡根目录下文件夹
<code>os.umount("/sd")</code>
卸载SD卡

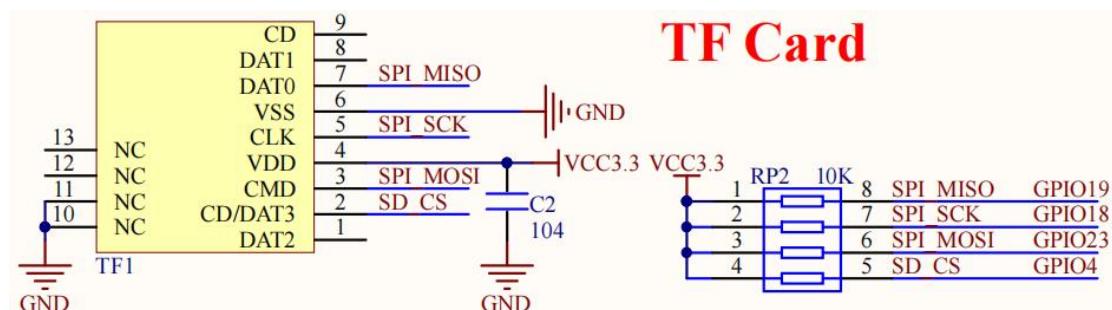
操作 SD 卡，可以按照 Python 文件相关方法来操作。

## 25.2 硬件设计

本实验使用到硬件资源如下：

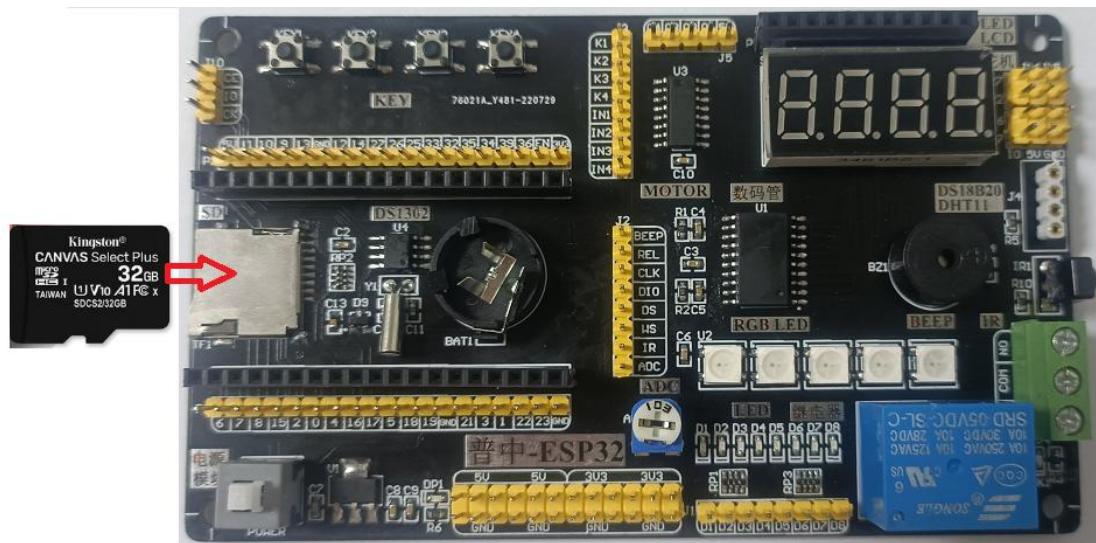
- (1) TF 卡模块
- (2) ESP32 GPIO

SD 卡接口电路如下：



从上图可知，SD 卡接口均连接到指定 IO。

本章实验使用 ESP32 的 I018、19、23 和 4 引脚，将预先准备好的 SD 卡进行格式化为 FAT32 格式，然后在 SD 卡根目录下新建一个文件夹“EBOOK”，在该文件夹下在新建一个文本文件“text.txt”。然后插到开发板 TF 卡座上即可。



### 25.3 软件设计

下面我们打开“\4--实验程序\1--MicroPython 实验\1--基础实验\24-SD 卡实验”程序，SD 卡驱动在 sdcards.py 中，该代码不展示，用户可打开工程查看。控制代码全部都在 main.py 中，代码如下：

```
main.py x sdcards.py x
1  """
2  深圳市普中科技有限公司 (PRECHIN 普中)
3  技术支持: www.prechin.net
4
5  实验名称: SD卡实验
6  接线说明: SD卡模块-->ESP32 IO
7      (DAT0)(MISO)-->(19)
8      (CMD)(MOSI)-->(23)
9      (CLK)(SCK)-->(18)
10     (CD/DAT3)(CS)-->(4)
11
12 实验现象: 程序下载成功后, 在SD卡指定目录下新建text.txt文件, 写入信息, 然后读取文件, 在Shell
13 控制台上输出读取文件中的字符信息。
14
15 注意事项: 程序中指定的SD卡路径为: /EBOOK/text.txt
16
17 ...
18
19 #导入Pin模块
20 import machine, sdcards, os
21 from machine import SPI, Pin
22
23 sd = sdcards.SDCard(SPI(2, sck=Pin(18), mosi=Pin(23), miso=Pin(19)), Pin(4))
24 os.mount(sd, "/sd")
25
26 print(os.listdir("/sd"))
27
```

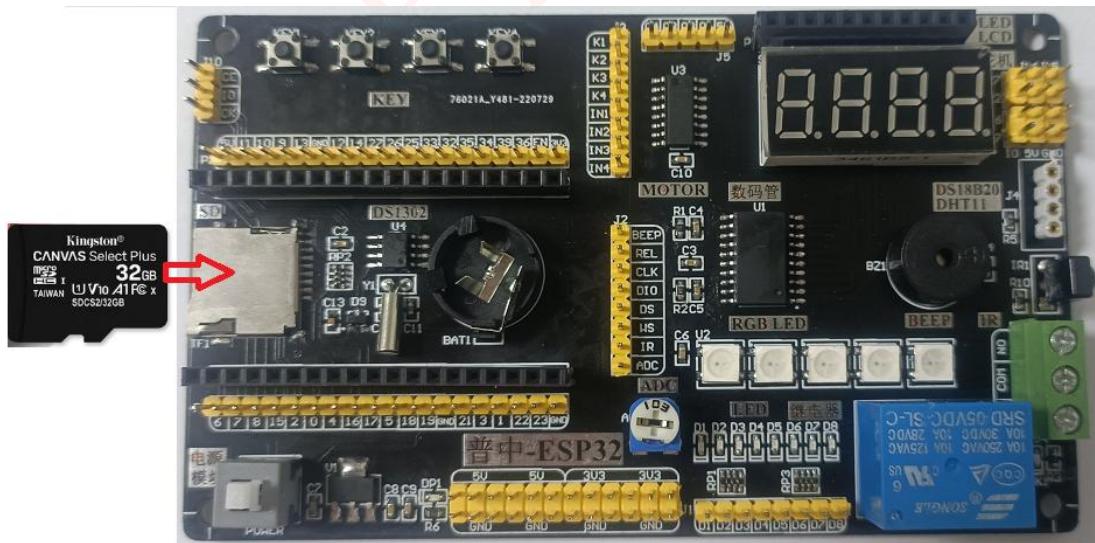
```
28 file_name="/sd/EBOOK/text.txt"
29
30 f=open(file_name,"w") #打开文件, 进行写操作
31 write_txt="大家好, 欢迎使用普中-ESP32开发板, 人生苦短, 我选Python和MicroPython"
32 print(f.write(write_txt)) #写入
33 f.close() #关闭文件
34
35 f=open(file_name) #打开文件, 进行读操作
36 read_txt=f.read()
37 print(read_txt)
38 f.close() #关闭文件
39
40
41 #程序入口
42 if __name__=="__main__":
43
44     while True:
45         pass
46
```

## 25.4 实验现象

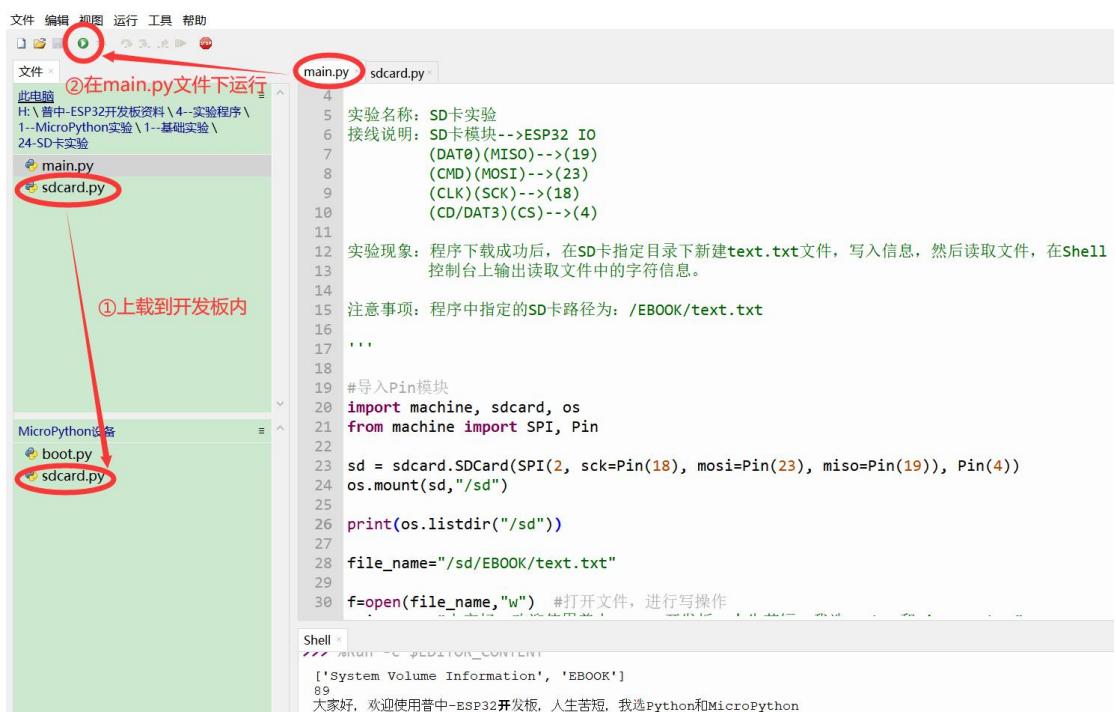
下载程序前, 准备好一张已格式化为 FAT32 的 SD 卡, 且在 SD 卡根目录下创建好 EBOOK 文件夹, 在该文件夹下已新建文本文件 text.txt, 如下所示:



插入到 TF 卡座上:



将程序下载到开发板内 (可参考“[2.2.5 程序下载运行](#)”章节), 注意: 因为本实验有 2 个程序模块, 即 2 个 .py 文件, 而程序是从 main.py 开始运行的, 其中又调用了 sdcard.py 内容, 因此要先将该文件上载开发板中, 然后可在 main.py 文件下点击“运行”或选择“运行当前脚本”。如下所示:



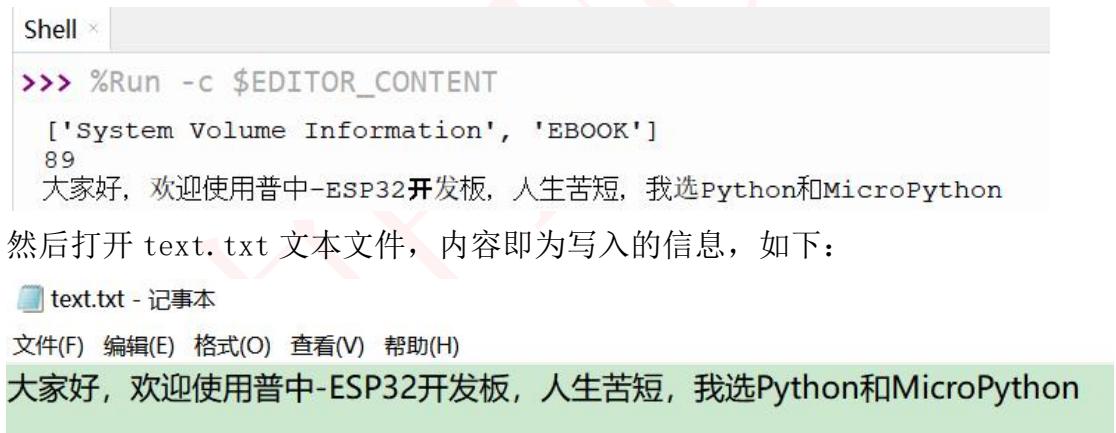
①上载到开发板内

②在main.py文件下运行

```
4 实验名称: SD卡实验
5 接线说明: SD卡模块-->ESP32 IO
6 (DAT0) (MISO)-->(19)
7 (CMD) (MOSI)-->(23)
8 (CLK) (SCK)-->(18)
9 (CD/DAT3) (CS)-->(4)
10
11 实验现象: 程序下载成功后, 在SD卡指定目录下新建text.txt文件, 写入信息, 然后读取文件, 在Shell控制台上输出读取文件中的字符信息。
12 注意事项: 程序中指定的SD卡路径为: /EBOOK/text.txt
13
14 ...
15
16
17
18 #导入Pin模块
19 import machine, sdcard, os
20 from machine import SPI, Pin
21
22 sd = sdcard.SDCard(SPI(2, sck=Pin(18), mosi=Pin(23), miso=Pin(19)), Pin(4))
23 os.mount(sd,"/sd")
24
25 print(os.listdir("/sd"))
26
27 file_name="/sd/EBOOK/text.txt"
28
29 f=open(file_name,"w") #打开文件, 进行写操作
30
```

Shell < /> %Run -c \$EDITOR\_CONTENT  
['System Volume Information', 'EBOOK']  
大家好, 欢迎使用普中-ESP32开发板, 人生苦短, 我选Python和MicroPython

实验现象: 可在 Shell 控制台输出读取的文件内容, 如下:



```
>>> %Run -c $EDITOR_CONTENT
['System Volume Information', 'EBOOK']
大家好, 欢迎使用普中-ESP32开发板, 人生苦短, 我选Python和MicroPython
```

## 课后作业

## 第 26 章 WIFI 实验-连接路由器

如今物联网市场异常火爆，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，通过 WIFI 接入到互联网，成了智能家居产品普遍的选择。ESP32 内部已集成 WIFI 功能，可以说它就是为 WIFI 无线连接而生的。本章来学习 ESP32 的 WIFI，使用 MicroPython 开发 WIFI 是非常简单而美妙的，让大家在学习物联网中变得简单有趣。本章分为如下几部分内容：

- 26.1 实验介绍
- 26.2 硬件设计
- 26.3 软件设计
- 26.4 实验现象

## 26.1 实验介绍

### 26.1.1 实验简介

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，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。本章我们就来学习如何通过 MicroPython 编程连上路由器。

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。

### 26.1.2 实验目的

连接无线路由器，将 ESP32 的 IP 地址等相关信息通过软件 Shell 控制台输出显示。

### 26.1.3 MicroPython 函数使用

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。这时我们只需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。

模块包含热点 AP 模式和客户端 STA 模式，热点 AP 是指电脑或手机端直接连接 ESP32 发出的热点实现连接，如果电脑连接模块 AP 热点，这样电脑就不能上网，因此在使用电脑端和模块进行网络通信时，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。

使用方法如下：

```
import network

wlan = network.WLAN(network.STA_IF) # 创建 station 接口
wlan.active(True)      # 激活接口
wlan.scan()           # 扫描允许访问的SSID
wlan.isconnected()    # 检查创建的station是否连已经接到AP
wlan.connect('essid', 'password') # 连接到指定ESSID网络
wlan.config('mac')     # 获取接口的MAC地址
wlan.ifconfig()        # 获取接口的 IP/netmask(子网掩码)/gw(网关)/DNS 地址

ap = network.WLAN(network.AP_IF) # 创建一个AP热点接口
ap.config(essid='ESP-AP') # 激活接口
ap.config(max_clients=10) # 设置热点允许连接数量
ap.active(True)          # 设置AP的ESSID名称
```

PRECHIN  
普中  
www.prechin.net

```

def do_connect():
    import network
    wlan = network.WLAN(network.STA_IF)
    wlan.active(True)
    if not wlan.isconnected():
        print('connecting to network...')
        wlan.connect('essid', 'password')
        while not wlan.isconnected():
            pass
    print('network config:', wlan.ifconfig())

```

## 26.2 硬件设计

由于 ESP32 内置 WIFI 功能，所以直接在开发板上使用即可，无需额外连接。

## 26.3 软件设计

下面我们打开 “\4--实验程序\1--MicroPython 实验\1--基础实验\25-WIFI 实验-连接无线路由器” 程序。控制代码全部都在 main.py 中，代码如下：

```

main.py x
1 ...
2 深圳市普中科技有限公司 (PRECHIN 普中)
3 技术支持: www.prechin.net
4
5 实验名称: WIFI实验-连接无线路由器
6 接线说明: LED模块-->ESP32 IO
7 (D1)-->(15)
8
9 实验现象: 程序下载成功后, 软件shell控制台输出当前IP、子网掩码、网关的地址信息
10
11 注意事项: ESP32 WIFI作为客户端连接路由器热点, 然后电脑也连接路由器, 此时可连接成功输出信息
12
13 ...
14
15 #导入Pin模块
16 from machine import Pin
17 import time
18 import network
19
20 #定义LED控制对象
21 led1=Pin(15,Pin.OUT)
22
23 #路由器WIFI账号和密码
24 ssid="puzhong88"
25 password="PUZHONG88"
26

```

```
27 #WIFI连接
28 def wifi_connect():
29     wlan=network.WLAN(network.STA_IF) #STA模式
30     wlan.active(True) #激活
31     start_time=time.time() #记录时间做超时判断
32
33     if not wlan.isconnected():
34         print("conneting to network...")
35         wlan.connect(ssid,password) #输入 WIFI 账号密码
36
37         while not wlan.isconnected():
38             led1.value(1)
39             time.sleep_ms(300)
40             led1.value(0)
41             time.sleep_ms(300)
42
43         #超时判断,15 秒没连接成功判定为超时
44         if time.time()-start_time>15:
45             print("WIFI Connect Timeout!")
46             break
47
48     else:
49         led1.value(0)
50         print("network information:", wlan.ifconfig())
51
52 #程序入口
53 if __name__=="__main__":
54     wifi_connect()
55
56
57
```

程序中 ssid 和 password 为所要连接路由器的 WIFI 账号和密码，按照自己路由器账号密码自行修改即可。

## 26.4 实验现象

将程序下载到开发板内（[可参考“2.2.5 程序下载运行”章节](#)），实验现象：软件 Shell 控制台上输出当前模块的 IP、子网掩码、网关和 DNS 域地址。如下：

```
Shell x
>>> %Run -c $EDITOR_CONTENT
network information: ('192.168.1.20', '255.255.255.0', '192.168.1.1', '192.168.1.1')
```

成功连接上无线路由器后，就可以做 socket 等相关网络通信的应用了。

注意：程序中 ssid 和 password 为所要连接路由器的 WIFI 账号和密码，按照自己路由器账号密码自行修改即可。如下所示：

```
26 #路由器WIFI账号和密码
27 ssid="puzhong88"
28 password="PUZHONG88"
```

## 课后作业

## 第 27 章 WIFI 实验-Socket 通信

前面章节已学习如何通过 MicroPython 编程实现 ESP32 开发板连接到无线路由器。本章我们来学习使用 MicroPython 实现 Socket 通信。Socket 几乎是整个互联网通信的基础。本章分为如下几部分内容：

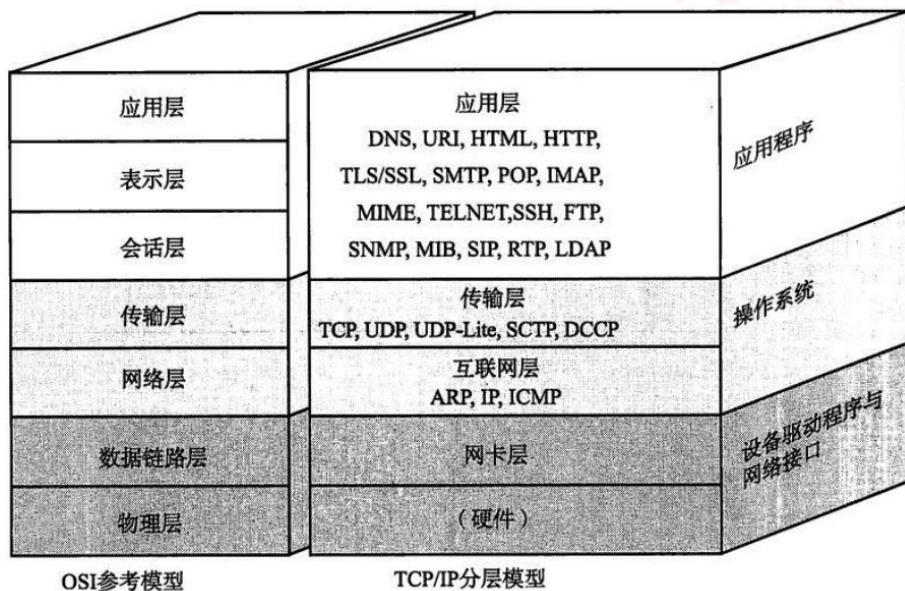
- 27.1 实验介绍
- 27.2 硬件设计
- 27.3 软件设计
- 27.4 实验现象

## 27.1 实验介绍

### 27.1.1 实验简介

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：



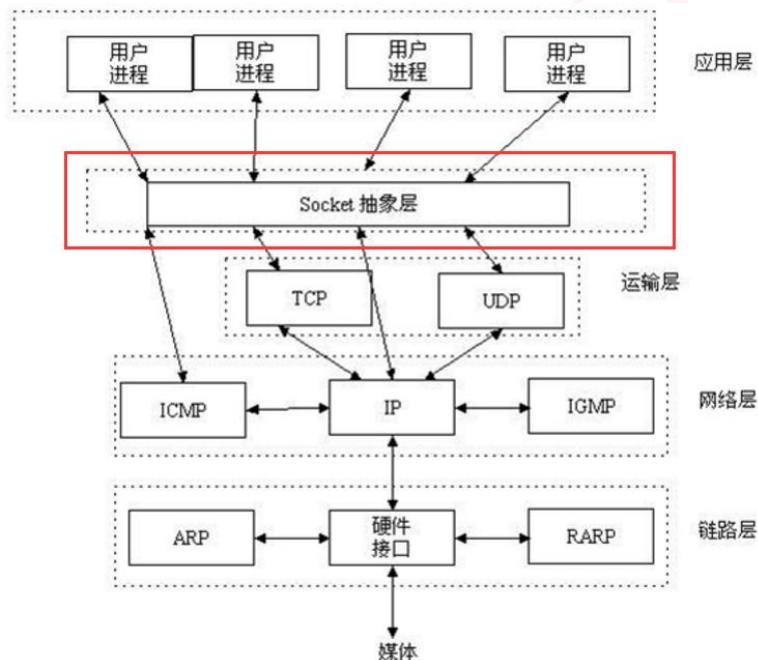
我们看看 TCP/IP 模型的传输层和应用层，传输层比较熟悉的概念是 TCP 和 UDP，UDP 协议基本没有对 IP 层的数据进行任何的处理了。而 TCP 协议还加入了更加复杂的传输控制，比如滑动的数据发送窗口（Slice Window），以及接收确认和重发机制，以达到数据的可靠传送。应用层中网页常用的则是 HTTP。那么我们先来解析一下这 TCP 和 HTTP 两者的关系。

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。

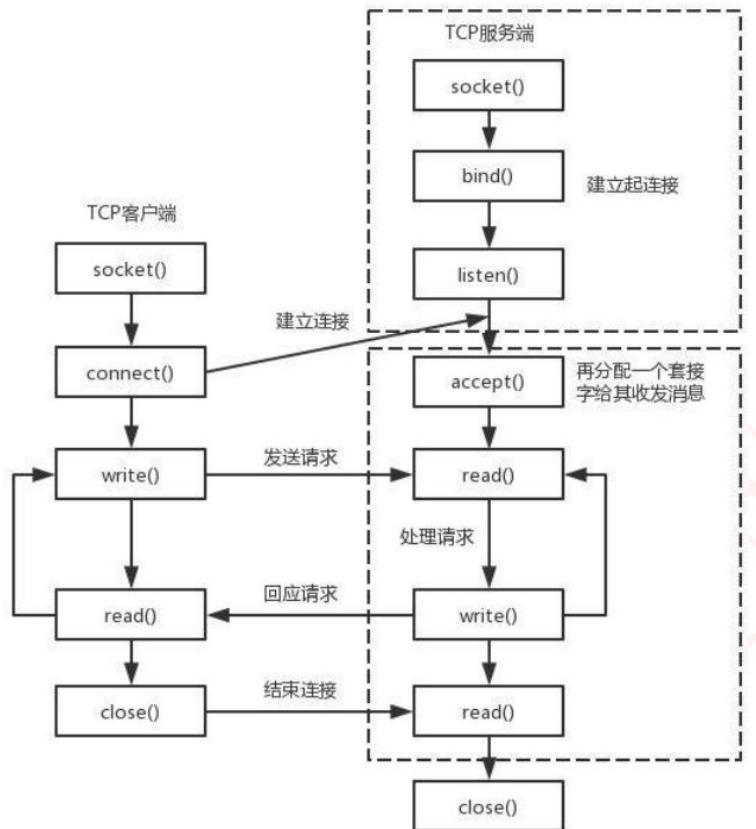
也就是应用层的 HTTP 通过传输层进行数据通信时，TCP 会遇到同时为多个应用程序进程提供并发服务的问题。多个 TCP 连接或多个应用程序进程可能需要通过同一个 TCP 协议端口传输数据。为了区别不同的应用程序进程和连接，许多计算机操作系统为应用程序与 TCP / IP 协议交互提供了套接字(Socket)接口。应用层可以和传输层通过 Socket 接口，区分来自不同应用程序进程或网络连接的通信，实现数据传输的并发服务。

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。



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：



从上图可以看到，建立 Socket 通信需要一个服务器端和一个客户端，以本实验为例，ESP32 开发板作为客户端，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。

### 27.1.2 实验目的

使用 Socket 编程实现 ESP32 开发板与电脑网咯调试助手建立连接，相互收发数据。

### 27.1.3 MicroPython 函数使用

MicroPython 已经封装好相关模块 usocket，它与传统的 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
s=usocket. socket (af=AF_INET, type=SOCK_STREAM, proto=IPPROTO_TCP)
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
addr=usocket. getaddrinfo ('www. prechin. cn', 80) [0] [-1]
获取 Socket 通信格式地址。返回: ('123. 56. 131. 110', 80)
s. connect (address)
创建连接。address: 地址格式为 IP+端口。例: ('192. 168. 1. 10', 10000)
s. send (bytes)
发送。bytes: 发送内容格式为字节
s. recv (bufsize)
接收数据。bufsize: 单次最大接收字节个数。
s. bind (address)
绑定, 用于服务器角色
s. listen ([backlog])
监听, 用于服务器角色。backlog: 允许连接个数, 必须大于 0。
s. accept ()
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:socket)

<http://docs.micropython.org/en/latest/library/socket.html?highlight=socket>

## 27.2 硬件设计

由于 ESP32 内置 WIFI 功能, 所以直接在开发板上使用即可, 无需额外连接。

## 27.3 软件设计

下面我们打开“\4--实验程序\1--MicroPython 实验\1--基础实验\26-WIFI 实验-Socket 通信”程序。控制代码全部都在 main.py 中，代码如下：

```
main.py x
1 ...
2 深圳市普中科技有限公司 (PRECHIN 普中)
3 技术支持: www.prechin.net
4
5 实验名称: WIFI实验-Socket通信
6 接线说明: LED模块-->ESP32 IO
7 (D1)-->(15)
8
9 实验现象: 程序下载成功后, 软件shell控制台输出当前IP、子网掩码、网关的地址信息, 在网络调试助手上
10 连接成功后, 可数据收发。
11
12 注意事项: ESP32 WIFI作为客户端连接路由器热点, 然后电脑也连接路由器, 此时可在电脑端使用网络调试助手,
13 设置: 协议类型: TCP Server, 本机主机地址: 电脑端IP地址, 本机主机端口: 10000
14
15 ...
16
17 #导入Pin模块
18 from machine import Pin
19 import time
20 import network
21 import usocket
22
23 #定义LED控制对象
24 led1=Pin(15,Pin.OUT)
25

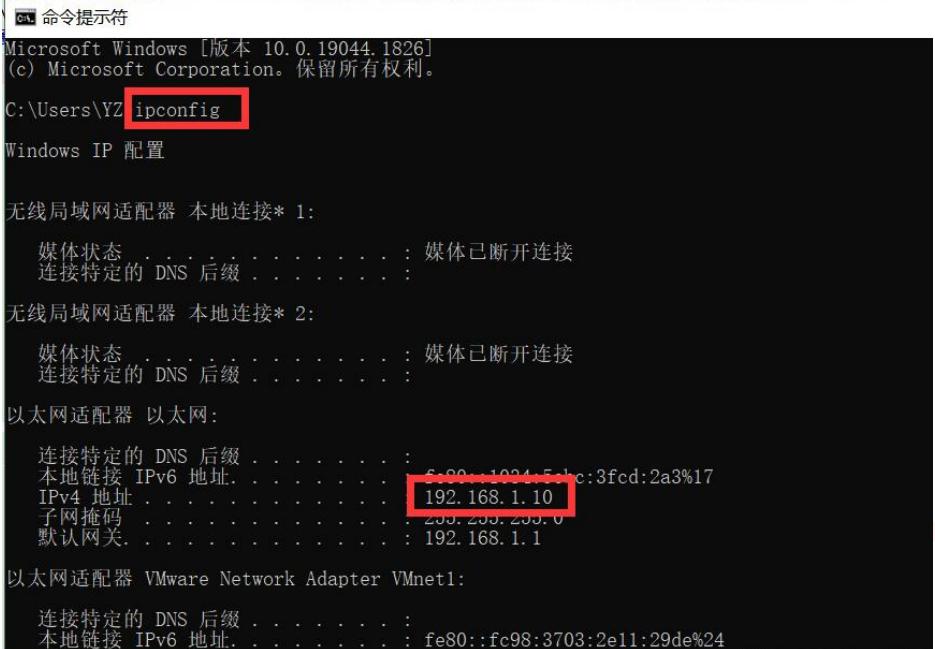
26 #路由器WIFI账号和密码
27 ssid="puzhong88"
28 password="PUZHONG88"
29
30 #服务器地址和端口
31 dest_ip="192.168.1.10"
32 dest_port=10000
33
34 #WIFI连接
35 def wifi_connect():
36     wlan=network.WLAN(network.STA_IF) #STA模式
37     wlan.active(True) #激活
38     start_time=time.time() #记录时间做超时判断
39
40     if not wlan.isconnected():
41         print("conneting to network...")
42         wlan.connect(ssid,password) #输入WIFI账号和密码
43
44         while not wlan.isconnected():
45             led1.value(1)
46             time.sleep_ms(300)
47             led1.value(0)
48             time.sleep_ms(300)
49
50         #超时判断,15 秒没连接成功判定为超时
51         if time.time()-start_time>15:
52             print("WIFI Connect Timeout!")
```

```
53         break
54     return False
55
56 else:
57     led1.value(0)
58     print("network information:", wlan.ifconfig())
59     return True
60
61
62 #程序入口
63 if __name__=="__main__":
64
65     if wifi_connect():
66         socket=usocket.socket() #创建socket连接
67         addr=(dest_ip,dest_port) #服务器IP地址和端口
68         socket.connect(addr)
69         socket.send("Hello PRECHIN")
70
71     while True:
72         text=socket.recv(128) #单次最多接收128字节
73         if text==None:
74             pass
75         else:
76             print(text)
77             socket.send("I get:"+text.decode("utf-8"))
78             time.sleep_ms(300)
```

WIFI 连接代码在前面已经讲解，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 PRECHIN’ 信息到服务器。另外每间隔 300ms 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。

## 27.4 实验现象

首先要确认电脑网络与 ESP32 模块连接的 WIFI 网络是否是同一个路由器，如果不是请连接一个网络。然后获取电脑本机 IP 地址，可打开电脑“命令提示符”窗口，输入命令： ipconfig 后回车，如下所示：



```
C:\命令提示符
Microsoft Windows [版本 10.0.19044.1826]
(c) Microsoft Corporation。保留所有权利。

C:\Users\YZ ipconfig

Windows IP 配置

无线局域网适配器 本地连接* 1:
    媒体状态 . . . . . : 媒体已断开连接
    连接特定的 DNS 后缀 . . . . . :

无线局域网适配器 本地连接* 2:
    媒体状态 . . . . . : 媒体已断开连接
    连接特定的 DNS 后缀 . . . . . :

以太网适配器 以太网:
    连接特定的 DNS 后缀 . . . . . :
    本地链接 IPv6 地址 . . . . . : 5001:1001:51c:3fcd:2a3%17
    IPv4 地址 . . . . . : 192.168.1.10
    子网掩码 . . . . . : 255.255.255.0
    默认网关. . . . . : 192.168.1.1

以太网适配器 VMware Network Adapter VMnet1:
    连接特定的 DNS 后缀 . . . . . :
    本地链接 IPv6 地址 . . . . . : fe80::fc98:3703:2e11:29de%24
```

从命令提示符窗口输出信息可知，我电脑本机 IP 地址是 192.168.1.10，因此在程序中，要修改该地址，使程序中的服务器地址与电脑本机地址一致。如下所示：



```
main.py x
11 注意事项: ESP32 WIFI作为客户端连接路由器热点, 然后电脑也连接路由器, 此时可在电脑端使用网络调试助手,
12     设置: 协议类型: TCP Server, 本机主机地址: 电脑端IP地址, 本机主机端口: 10000
13
14 ...
15
16
17 #导入Pin模块
18 from machine import Pin
19 import time
20 import network
21 import usocket
22
23 #定义LED控制对象
24 led1=Pin(15,Pin.OUT)
25
26 #路由器WIFI账号和密码
27 ssid="puzhong88"
28 password="PUZHONG88"
29
30 #服务器地址和端口
31 dest_ip="192.168.1.10"
32 dest_port=10000
33
```

并且注意：程序中 ssid 和 password 为所要连接路由器的 WIFI 账号和密码，按照自己路由器账号密码自行修改即可。如下所示：

```
26 #路由器WIFI账号和密码
27 ssid="puzhong88"
28 password="PUZHONG88"
```

然后打开资料 “\5--开发工具\6-网络调试助手\NetAssist.exe”，设置网络调试助手参数，注意端口号要与程序中设置保持一致。打开界面如下：



将程序下载到开发板内（可参考“2.2.5 程序下载运行”章节），实验现象：软件 Shell 控制台上输出当前模块的 IP、子网掩码、网关和 DNS 域地址。如下：

```
Shell >>> %Run -c $EDITOR_CONTENT
network information: ('192.168.1.22', '255.255.255.0', '192.168.1.1', '192.168.1.1')
b'Hello PRECHIN'
```

从输出模块 IP 地址可知，该模块所分配 IP 地址与服务器 IP 地址处于同一网段。然后可在网络调试助手上发送数据，并可看到 ESP32 模块发送的数据，如下所示：



## 课后作业

## 第 28 章 WIFI 实验-MQTT 通信

前面我们学习了 Socket 通信，当服务器和客户端建立起连接时，就可以相互通信。在互联网应用中，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。因此就诞生出 MQTT（消息队列遥测传输）协议。本章来学习使用 ESP32 实现 MQTT 协议信息的发布和订阅。本章分为如下几部分内容：

- 28.1 实验介绍
- 28.2 硬件设计
- 28.3 软件设计
- 28.4 实验现象

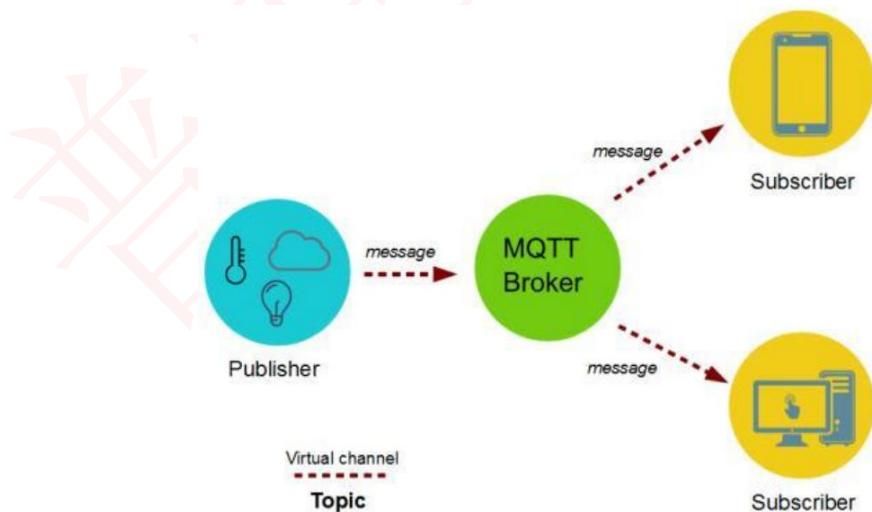
## 28.1 实验介绍

### 28.1.1 实验简介

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



从上图可看到，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：



确定了角色后，数据是如何传输的呢？下图是 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。

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

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。

### 28.1.2 实验目的

使用 ESP32 实现 MQTT 协议信息的发布和订阅，与通信猫 MQTT 调试助手实现数据收发。

### 28.1.3 MicroPython 函数使用

MicroPython 已经封装好 MQTT 客户端模块，这使得 MQTT 应用开发变得简单。MQTT 模块文件在例程文件夹里面的 simple.py 文件，使用方法如下：

构造函数
client=simple.MQTTClient (client_id, server, port)
构建 MQTT 客户端对象。 client_id: 客户端 ID, 具有唯一性; server: 服务器地址, 可以是 IP 或者网址; port: 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
client.connect()
连接到服务器。
client.publish(TOPIC, message)
发布。TOPIC: 主题编号; message: 信息内容, 例: 'Hello PRECHIN'
client.subscribe(TOPIC)
订阅。TOPIC: 主题编号。
client.set_callback(callback)
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。
client.check_msg()
检查订阅信息。如收到信息就执行设置过的回调函数 callback。

## 28.2 硬件设计

由于 ESP32 内置 WIFI 功能, 所以直接在开发板上使用即可, 无需额外连接。

## 28.3 软件设计

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, 本实验分开两个程序来编程, 分别为发布者和订阅者。

### 28.3.1 发布者

下面我们打开 “\4--实验程序\1--MicroPython 实验\1--基础实验\27-WIFI 实验-MQTT 通信\发布者” 程序, MQTT 客户端模块在 simple.py 中, 该代码不展

示，用户可打开工程查看。控制代码全部都在 main.py 中，代码如下：

```
main.py x
1 ...
2 深圳市普中科技有限公司 (PRECHIN 普中)
3 技术支持: www.prechin.net
4
5 实验名称: WIFI实验-MQTT通信 (发布者)
6 接线说明: LED模块-->ESP32 IO
7 (D1)-->(15)
8
9 实验现象: 程序下载成功后, 软件shell控制台输出当前IP、子网掩码、网关的地址信息,
10      MQTT在线助手, 输入网址: http://www.tongxinmao.com/txm/webmqtt.php#
11      进入, 按默认设置, 选择连接, 然后在Subscribe订阅主题中选择程序中设置的Topic主题:
12      /public/pz_esp32/1
13
14 注意事项: ESP32 WIFI作为客户端连接路由器热点, 然后电脑也连接路由器, 此时可连接成功输出信息
15 ...
16 ...
17
18 #导入Pin模块
19 from machine import Pin
20 import time
21 import network
22 from simple import MQTTClient
23
24 #定义LED控制对象
25 led1=Pin(15,Pin.OUT)
26
27 #路由器WIFI账号和密码
28 ssid="puzhong88"
29 password="PUZHONG88"
30
31 #WIFI连接
32 def wifi_connect():
33     wlan=network.WLAN(network.STA_IF) #STA模式
34     wlan.active(True) #激活
35     start_time=time.time() #记录时间做超时判断
36
37     if not wlan.isconnected():
38         print("conneting to network...")
39         wlan.connect(ssid,password) #输入WIFI账号和密码
40
41         while not wlan.isconnected():
42             led1.value(1)
43             time.sleep_ms(300)
44             led1.value(0)
45             time.sleep_ms(300)
46
47             #超时判断,15 秒没连接成功判定为超时
48             if time.time()-start_time>150:
49                 print("WIFI Connect Timeout!")
50                 break
51
52     return False
```

```
53     else:
54         led1.value(0)
55         print("network information:", wlan.ifconfig())
56         return True
57
58 #发布数据任务
59 def mqtt_send(tim):
60     client.publish(TOPIC, "Hello PRECHIN")
61
62 #程序入口
63 if __name__=="__main__":
64
65     if wifi_connect():
66         SERVER="mq.tongxinmao.com"
67         PORT=18830
68         CLIENT_ID="PZ-ESP32" #客户端ID
69         TOPIC="/public/pz_esp32/1" #TOPIC名称
70         client = MQTTClient(CLIENT_ID, SERVER, PORT)
71         client.connect()
72
73         #开启RTOS定时器,周期 1000ms, 执行MQTT通信接收任务
74         tim = Timer(0)
75         tim.init(period=1000, mode=Timer.PERIODIC,callback=mqtt_send)
76
```

### 28.3.2 订阅者

下面我们打开“\4--实验程序\1--MicroPython 实验\1--基础实验\27-WIFI 实验-MQTT 通信\订阅者”程序，MQTT 客户端模块在 simple.py 中，该代码不展示，用户可打开工程查看。控制代码全部都在 main.py 中，代码如下：

```
main.py x
1 ...
2 深圳市普中科技有限公司（PRECHIN 普中）
3 技术支持: www.prechin.net
4
5 实验名称: WIFI实验-MQTT通信（订阅者）
6 接线说明: LED模块-->ESP32 IO
7 (D1)-->(15)
8
9 实验现象: 程序下载成功后, 软件shell控制台输出当前IP、子网掩码、网关的地址信息,
10      MQTT在线助手, 输入网址: http://www.tongxinmao.com/txm/webmqtt.php#
11      进入, 按默认设置, 选择连接, 然后在Publish发布主题中选择程序中设置的Topic主题:
12      /public/pz_esp32/1
13      , 设置要发送的信息, 点击发布, 在Shell控制台即可输出主题和消息
14
15 注意事项: ESP32 WIFI作为客户端连接路由器热点, 然后电脑也连接路由器, 此时可连接成功输出信息
16
17 ...
18
19 #导入Pin模块
20 from machine import Pin
21 from machine import Timer
22 import time
23 import network
24 from simple import MQTTClient
25
26 #定义LED控制对象
27 led1=Pin(15,Pin.OUT)
```

```
28 #路由器WIFI账号和密码
29 ssid="puzhong88"
30 password="PUZHONG88"
31
32 #WIFI连接
33 def wifi_connect():
34     wlan=network.WLAN(network.STA_IF) #STA模式
35     wlan.active(True) #激活
36     start_time=time.time() #记录时间做超时判断
37
38 if not wlan.isconnected():
39     print("conneting to network...")
40     wlan.connect(ssid,password) #输入WIFI账号和密码
41
42     while not wlan.isconnected():
43         led1.value(1)
44         time.sleep_ms(300)
45         led1.value(0)
46         time.sleep_ms(300)
47
48     #超时判断,15 秒没连接成功判定为超时
49     if time.time()-start_time>150:
50         print("WIFI Connect Timeout!")
51         break
52
53     return False
54
55 else:
56     led1.value(0)
57     print("network information:", wlan.ifconfig())
58     return True
59
60 #设置 MQTT 回调函数,有信息时候执行
61 def mqtt_callback(topic,msg):
62     print("topic: {}".format(topic))
63     print("msg: {}".format(msg))
64
65 #接收数据任务
66 def mqtt_recv(tim):
67     client.check_msg()
68
69
70 #程序入口
71 if __name__=="__main__":
72
73     if wifi_connect():
74         SERVER="mq.tongxinmao.com"
75         PORT=18830
76         CLIENT_ID="PZ-ESP32" #客户端ID
77         TOPIC="/public/pz_esp32/1" #TOPIC名称
78         client = MQTTClient(CLIENT_ID, SERVER, PORT) #建立客户端
79         client.set_callback(mqtt_callback) #配置回调函数
80
81         client.connect()
82         client.subscribe(TOPIC) #订阅主题
83
84     #开启RTOS定时器,周期 300ms, 执行MQTT通信接收任务
85     tim = Timer(0)
86     tim.init(period=300, mode=Timer.PERIODIC,callback=mqtt_recv)
```

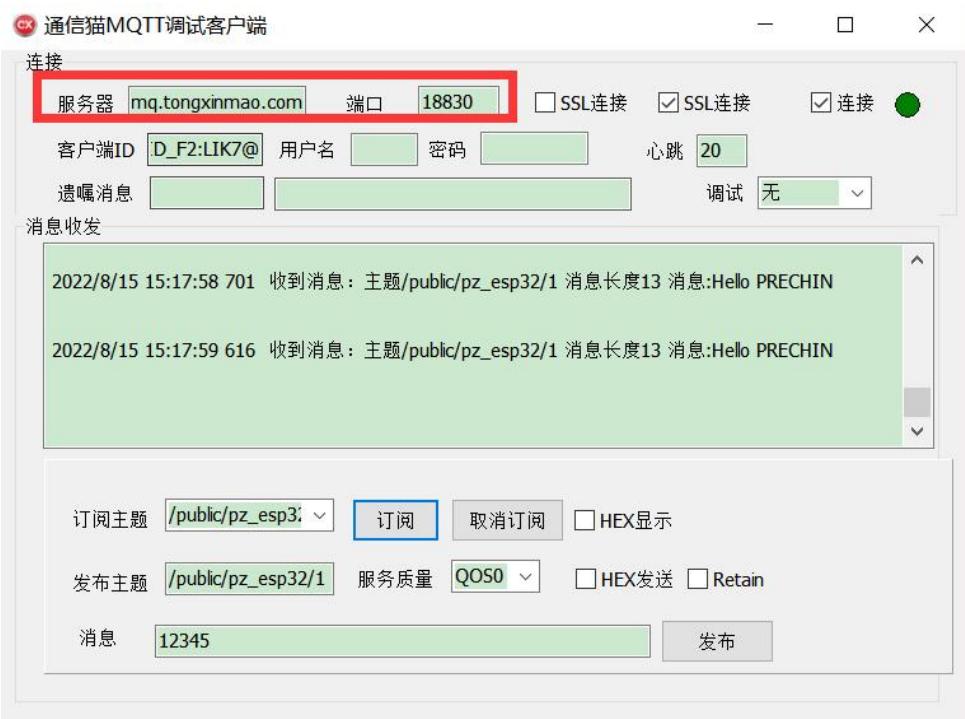
## 28.4 实验现象

### 28.4.1 发布者

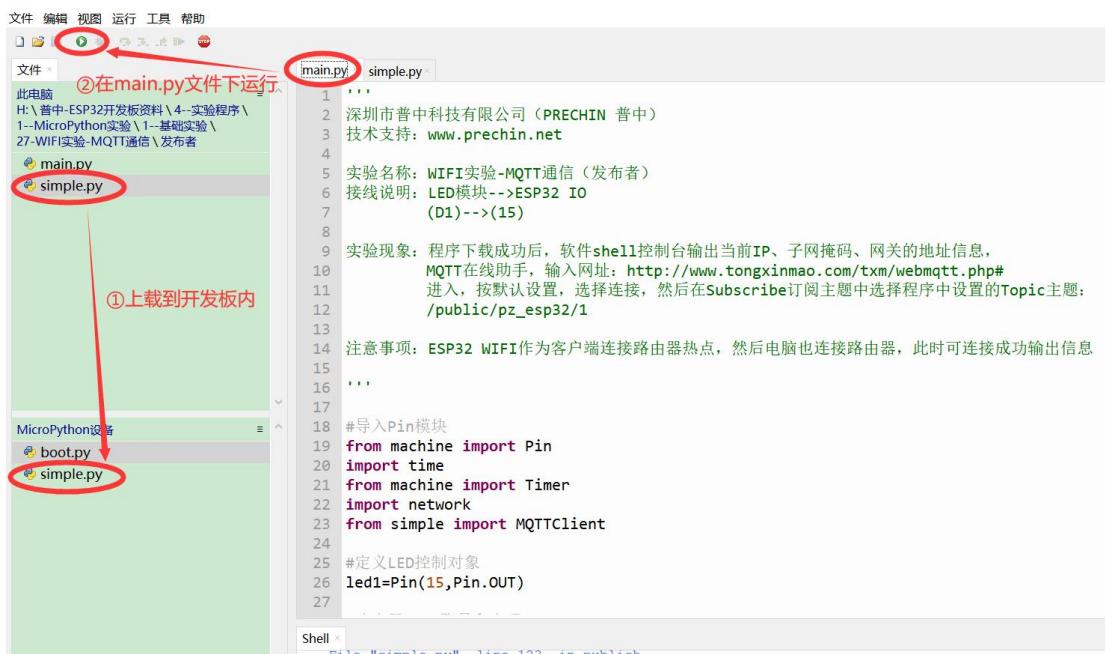
首先要确认 ESP32 模块已成功连接上路由器，不同用户路由器账号密码不一样，可在代码对应位置修改，如下：

```
28 #路由器WIFI账号和密码
29 ssid="puzhong88"
30 password="PUZHONG88"
```

然后打开资料“\5--开发工具\7-通信猫 MQTT 调试客户端\MQTT 调试客户端.exe”，设置网络调试助手参数，注意服务器地址和端口号要与程序中设置保持一致。打开界面如下：



注意：因为本实验有 2 个程序模块，即 2 个.py 文件，而程序是从 main.py 开始运行的，其中又调用了 simple.py 内容，因此要先将该文件上载开发板中，然后可在 main.py 文件下点击“运行”或选择“运行当前脚本”。如下所示：



将程序下载到开发板内（可参考“2.2.5 程序下载运行”章节），实验现象：

软件 Shell 控制台上输出当前模块的 IP、子网掩码、网关和 DNS 域地址。说明模块已成功连接无线路由器，如下：

```
Shell x
File "simple.py", line 123, in publish
    OSError: [Errno 128] ENOTCONN%Run -c $EDITOR_CONTENT
    network information: ('192.168.4.5', '255.255.255.0', '192.168.4.15', '192.168.4.15')
>>>
```

然后可在网络调试助手，设置好订阅主题”/public/pz\_esp32/1”，注意：订阅主题必须与程序中一致，如下所示：

```
63 #程序入口
64 if __name__=="__main__":
65
66     if wifi_connect():
67         SERVER="mq.tongxinmao.com"
68         PORT=18830
69         CLIENT_ID="PZ-ESP32" #客户端ID
70         TOPIC="/public/pz_esp32/1" #TOPIC名称
71         client = MQTTClient(CLIENT_ID, SERVER, PORT)
72         client.connect()
73
```

点击订阅按钮，在调试助手上即可显示 ESP32 模块发送该主题的信息，如下所示：



## 28.4.2 订阅者

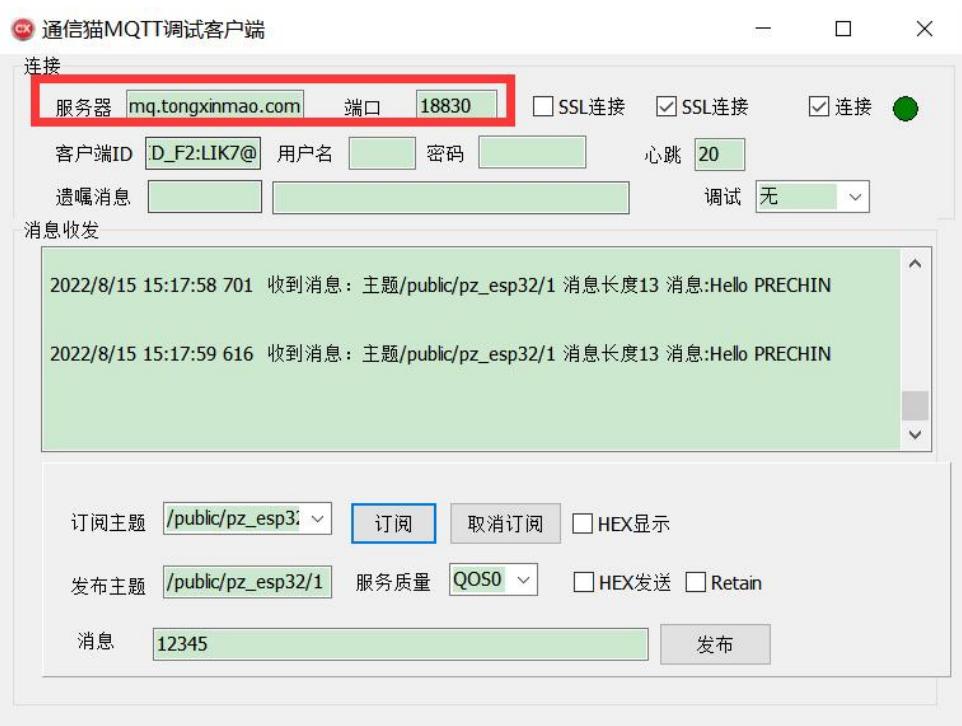
首先要确认 ESP32 模块已成功连接上路由器，不同用户路由器账号密码不一样，可在代码对应位置修改，如下：

```

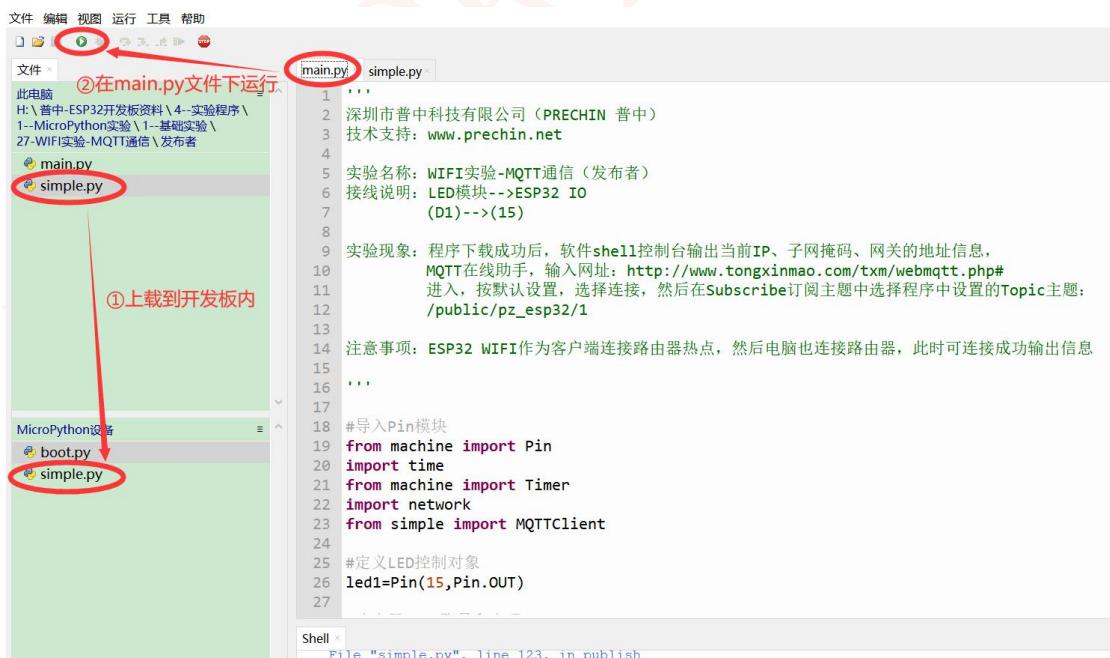
28 #路由器WIFI账号和密码
29 ssid="puzhong88"
30 password="PUZHONG88"

```

然后打开资料“\5--开发工具\7-通信猫 MQTT 调试客户端\MQTT 调试客户  
端.exe”，设置网络调试助手参数，注意服务器地址和端口号要与程序中设置保  
持一致。打开界面如下：



注意：因为本实验有 2 个程序模块，即 2 个 .py 文件，而程序是从 main.py 开始运行的，其中又调用了 simple.py 内容，因此要先将该文件上载开发板中，然后可在 main.py 文件下点击“运行”或选择“运行当前脚本”。如下所示：



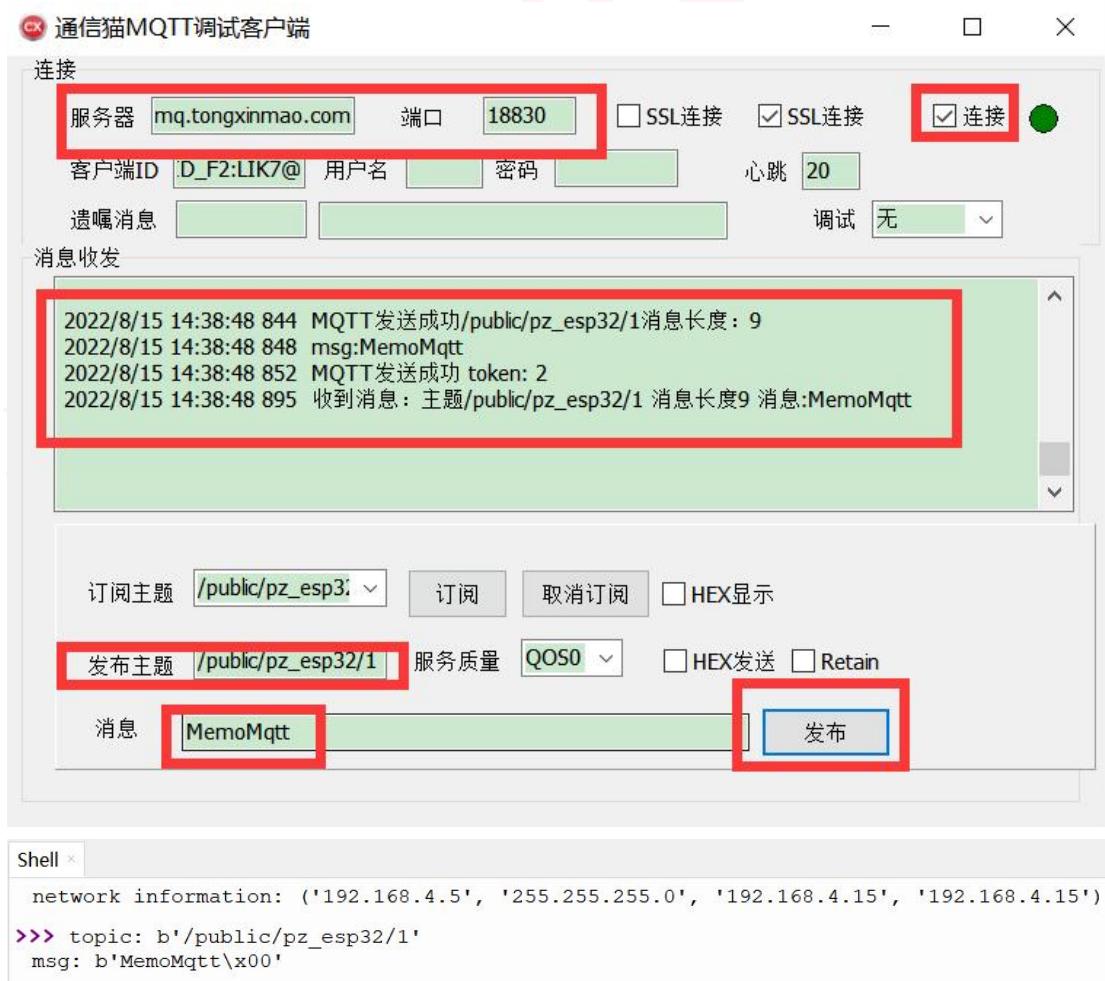
将程序下载到开发板内（**可参考“2.2.5 程序下载运行”章节**），实验现象：软件 Shell 控制台上输出当前模块的 IP、子网掩码、网关和 DNS 域地址。说明模块已成功连接无线路由器，如下：

```
Shell x
  File "simple.py", line 123, in publish
    OSError: [Errno 128] ENOTCONN%Run -c $EDITOR_CONTENT
    network information: ('192.168.4.5', '255.255.255.0', '192.168.4.15', '192.168.4.15')
>>>
```

然后可在网络调试助手，设置好发布主题”/public/pz\_esp32/1”，注意：发布主题必须与订阅者程序中一致，如下所示：

```
63 #程序入口
64 if __name__=="__main__":
65
66     if wifi_connect():
67         SERVER="mq.tongxinmao.com"
68         PORT=18830
69         CLIENT_ID="PZ-ESP32" #客户端ID
70         TOPIC="/public/pz_esp32/1" #TOPIC名称
71         client = MQTTClient(CLIENT_ID, SERVER, PORT)
72         client.connect()
73
```

点击发布按钮，在调试助手上即可显示发送该主题的信息，并且在 ESP32 开发板软件 Shell 控制台输出成功接收到的订阅主题信息，如下所示：



## 课后作业

普中 PRECHIN

## 第 29 章 WIFI 实验-手机控制 LED

前面我们已经学习了 ESP32 的 WIFI 使用，包括连接路由器、Socket 通信和 MQTT 通信，本章就来应用下 Socket，实现手机 WIFI 连接控制板载 LED。本章分为如下几部分内容：

- 29.1 实验介绍
- 29.2 硬件设计
- 29.3 软件设计
- 29.4 实验现象

## 29.1 实验介绍

### 29.1.1 实验简介

前面我们已经学习了 ESP32 的 WIFI 常用通信方法，要实现远程控制，可使用 MQTT 通信与云端连接进行数据传输，然后远端设备同样连接到云端收发 ESP32 数据。但有的场景需要在局域内实现无线控制，比如利用 WIFI 连接控制小车或家居内智能设备等，此时可使用 Socket 通信。本实验就使用 Socket 通信让手机或电脑控制 ESP32 开发板上 LED。

### 29.1.2 实验目的

使用 Socket 通信让手机或电脑控制 ESP32 开发板上 LED。

### 29.1.3 MicroPython 函数使用

MicroPython 已经封装好相关模块 usocket，它与传统的 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
s=usocket. socket (af=AF_INET, type=SOCK_STREAM, proto=IPPROTO_TCP)
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
addr=usocket.getaddrinfo('www.prechin.cn', 80)[0][-1]
获取 Socket 通信格式地址。返回: ('123.56.131.110', 80)
s.connect(address)
创建连接。address: 地址格式为 IP+端口。例: ('192.168.1.10', 10000)
s.send(bytes)
发送。bytes: 发送内容格式为字节
s.recv(bufsize)
接收数据。bufsize: 单次最大接收字节个数。
s.bind(address)
绑定, 用于服务器角色
s.listen([backlog])
监听, 用于服务器角色。backlog: 允许连接个数, 必须大于 0。
s.accept()
接受连接, 用于服务器角色。

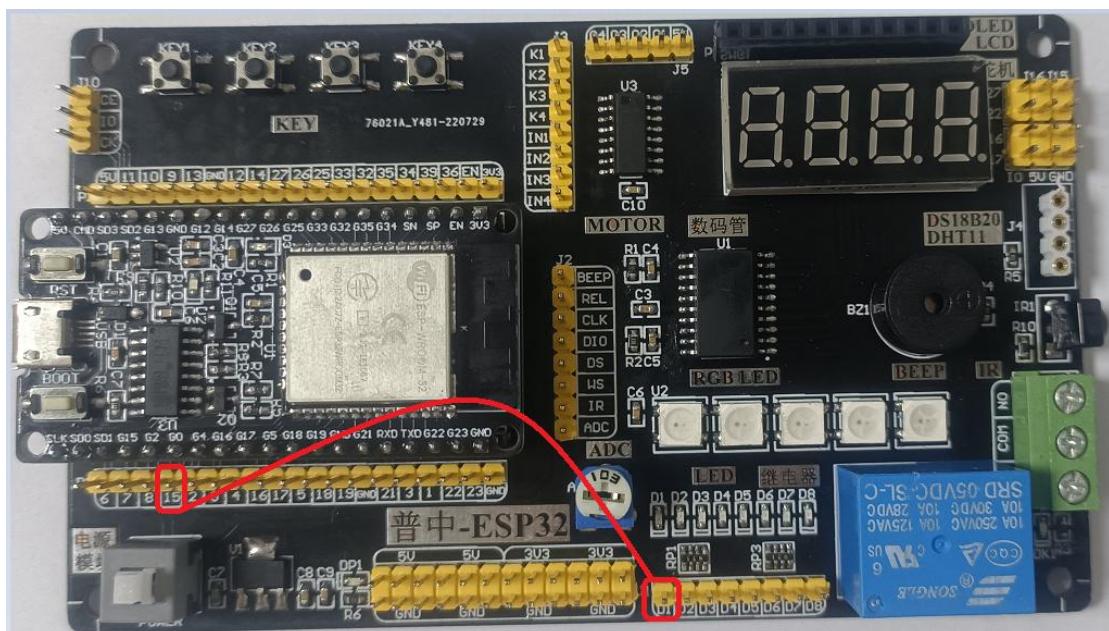
其它更多用法请阅读 MicroPython 文档: (搜索:socket)

<http://docs.micropython.org/en/latest/library/socket.html?highlight=socket>

ht=socket

## 29.2 硬件设计

由于 ESP32 内置 WIFI 功能, 所以直接在开发板上使用即可, 无需额外连接。



接线说明：

LED模块-->ESP32 IO

D1-->15

## 29.3 软件设计

下面我们打开“\4--实验程序\1--MicroPython 实验\1--基础实验\28-WIFI 实验-手机控制 LED”程序。控制代码全部都在 main.py 中，代码如下：

```
main.py <...>
1  ...
2  深圳市普中科技有限公司 (PRECHIN 普中)
3  技术支持: www.prechin.net
4
5  实验名称: WIFI实验-手机控制LED
6  接线说明: LED模块-->ESP32 IO
7      (D1)-->(15)
8
9  实验现象: 程序下载成功后, 手机连接的WIFI需和ESP32连接的WIFI处于同一频段 (比如192.168.1.xx) ,
10  然后在手机网页输入Shell控制台输出的本机IP地址即可进入手机端网页控制板子LED
11
12 注意事项: ESP32作为服务器, 手机或电脑作为客户端
13
14 ...
15
16 #导入Pin模块
17 from machine import Pin
18 import time
19 import network
20 import socket
21
22 #定义LED控制对象
23 led1=Pin(15,Pin.OUT,Pin.PULL_DOWN)
24
25 #连接的WIFI账号和密码
26 ssid = "puzhong88"
27 password = "PUZHONG88"
```

```
28 #WIFI连接
29 def wifi_connect():
30     wlan=network.WLAN(network.STA_IF) #STA模式
31     wlan.active(True) #激活
32
33     if not wlan.isconnected():
34         print("conneting to network...")
35         wlan.connect(ssid,password) #输入 WIFI 账号密码
36
37     while not wlan.isconnected():
38         led1.value(1)
39         time.sleep_ms(300)
40         led1.value(0)
41         time.sleep_ms(300)
42     led1.value(0)
43     return False
44
45 else:
46     led1.value(0)
47     print("network information:", wlan.ifconfig())
48     return True
49

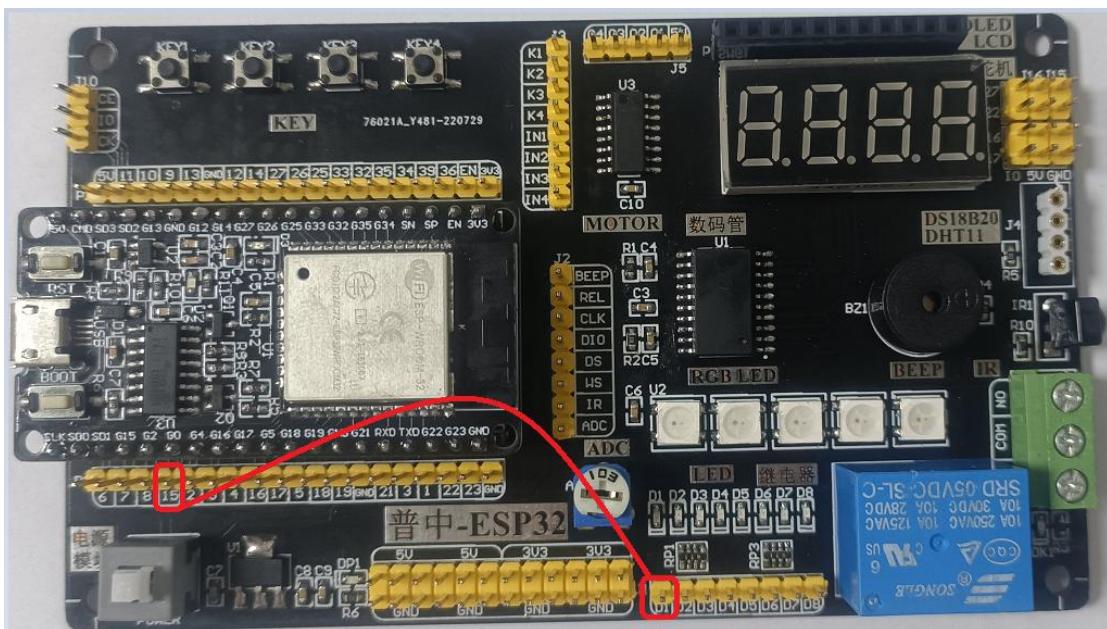
50 #网页数据
51 def web_page():
52     if led1.value() == 0:
53         gpio_state="OFF"
54     else:
55         gpio_state="ON"
56
57     html = """<html><head> <title>ESP32 LED control</title> <meta name="viewport" content="width=device-width, initial-
58     <link rel="icon" href="data:, "> <style>html{font-family: Helvetica; display:inline-block; margin: 0px auto; te
59     h1{color: #0F3376; padding: 2vh;}p{font-size: 1.5rem;}.button{display: inline-block; background-color: #e7bd3b;
60     border-radius: 4px; color: white; padding: 16px 40px; text-decoration: none; font-size: 30px; margin: 2px; cur
61     .button2{background-color: #4286f4;}</style></head><body> <h1>ESP32 LED control</h1>
62     <p>GPIO state: <strong>"""+ gpio_state + """/strong></p><p><a href="/?led=on"><button class="button">ON</but
63     <p><a href="/?led=off"><button class="button button2">OFF</button></a></p></body></html>"""
64     return html
65
66 #程序入口
67 if __name__=="__main__":
68
69     if wifi_connect():
70         #SOCK_STREAM表示的是TCP协议, SOCK_DGRAM表示的是UDP协议
71         my_socket=socket.socket(socket.AF_INET, socket.SOCK_STREAM) #创建socket连接
72         # 将socket对象绑定ip地址和端口号
73         my_socket.bind(('', 80))

74     # 相当于电话的开机 括号里的参数表示可以同时接收5个请求
75     my_socket.listen(5)
76
77     while True:
78         # 进入监听状态, 等待别人链接过来, 有两个返回值,
79         # 一个是对方的socket对象, 一个是对方的ip以及端口
80         client, addr = my_socket.accept()
81         print('Got a connection from %s' % str(addr))
82         # recv表示接收, 括号里是最大接收字节
83         request = client.recv(1024)
84         request = str(request)
85         print('Content = %s' % request)
86         led_on = request.find('/?led=on')
87         led_off = request.find('/?led=off')
88         if led_on == 6:
89             print('LED ON')
90             led1.value(1)
91         if led_off == 6:
92             print('LED OFF')
93             led1.value(0)
94         response = web_page()
95         client.send('HTTP/1.1 200 OK\n')
96         client.send('Content-Type: text/html\n')
97         client.send('Connection: close\n\n')
98         client.sendall(response)
99         client.close()
100
```

有关网页编程及格式大家可上网百度，网上有详细教程。

## 29.4 实验现象

下载程序前，按照如下接线：



接线说明：

LED模块—>ESP32 10

D1—>15

首先要确认 ESP32 连接的 WIFI 网络与手机端或电脑端同一个，比如 ESP32 模块连接路由器 WIFI，然后手机或者电脑也连接同一路由器。

注意：程序中 ssid 和 password 为所要连接路由器的 WIFI 账号和密码，按照自己路由器账号密码自行修改即可。如下所示：

```
26 #路由器WIFI账号和密码
27 ssid="puzhong88"
28 password="PUZHONG88"
```

将程序下载到开发板内（**可参考“2.2.5 程序下载运行”章节**），实验现象：  
软件 Shell 控制台上输出当前模块的 IP 地址和端口。

```
Shell x
LED OFF
Got a connection from ('192.168.149.223', 59986)
```

可打开手机浏览器或电脑浏览器输入对应 IP，然后即可显示控制页面，通过点击页面中的 ON 或 OFF 按钮控制板子 LED。

16:20 | 1.2K/s 4G 71%

ESP32 LED control

## ESP32 LED control

GPIO state: OFF

ON

OFF

16:20 | 0.0K/s 4G 71%

ESP32 LED control

## ESP32 LED control

GPIO state: ON

ON

OFF

课后作业

論著