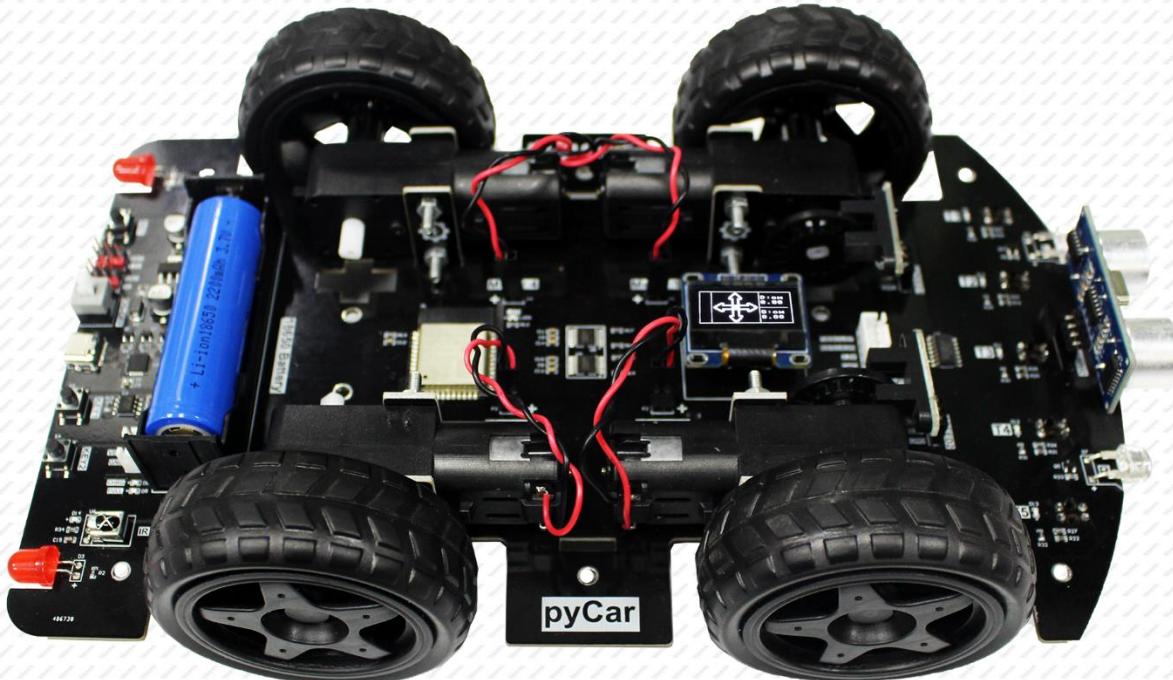


# MicroPython 从0到1

用python做嵌入式编程

(基于pyCar平台)

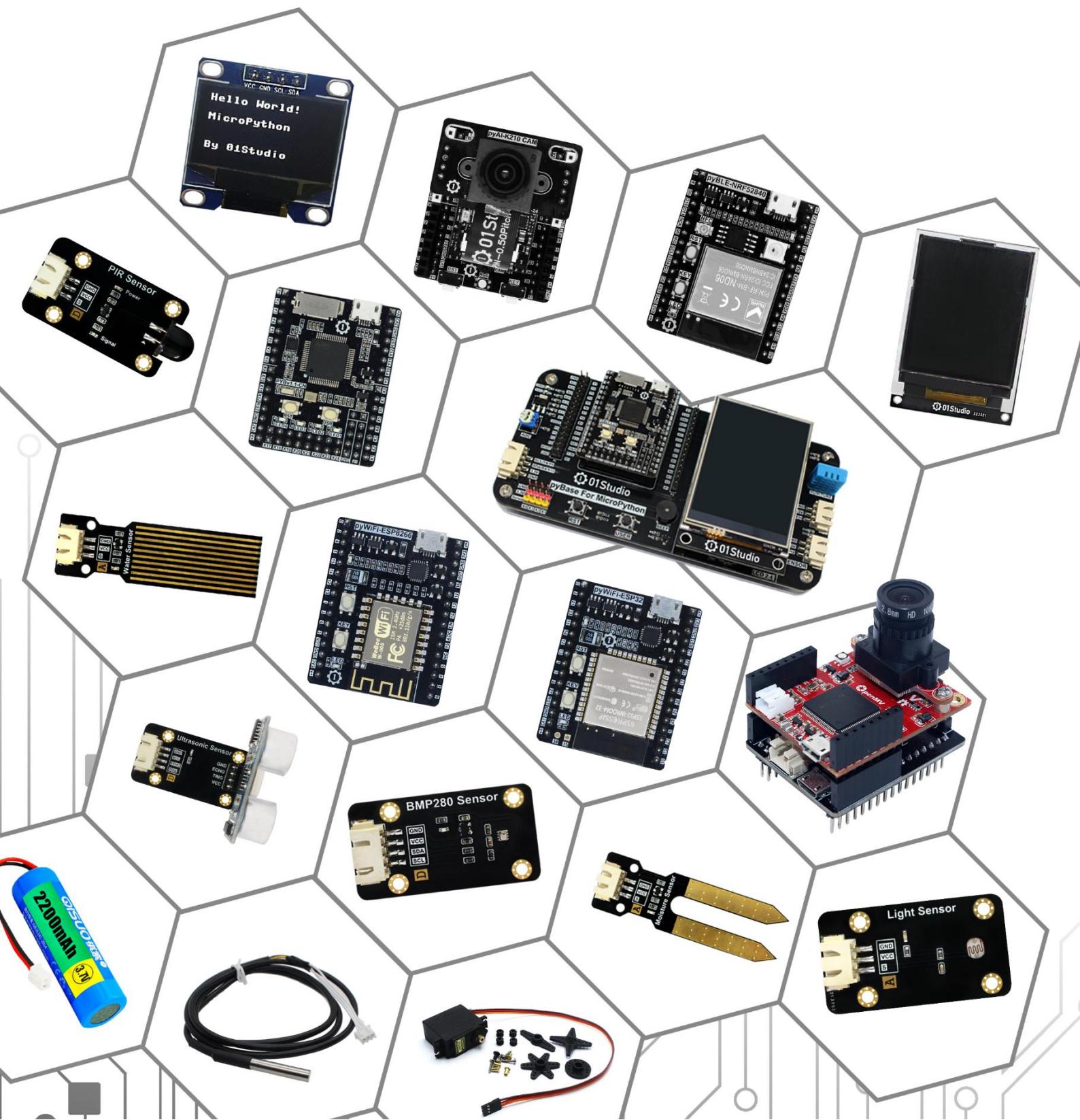
01Studio团队 编著



 01Studio  
-专注Python嵌入式编程-

# MicroPython

## 产品家族



# 前言

## 为什么学习 MicroPython?

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

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

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

## 为什么要写《MicroPython 从 0 到 1》教程？

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

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

## 为什么要打造 MicroPython 学习套件？

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

以用它来完成你的比赛或项目。

### 为什么要打造 01Studio 社区论坛？

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

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

期待你加入 01Studio 社区大家庭，成为我们的一份子，一起成长。

【社区链接: [www.01Studio.cc](http://www.01Studio.cc)】

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

【微信公众号: 01Studio 社区】

01Studio

2021.12 于深圳

## 版本说明

版本	日期	作者	更新内容
v1. 0	2021-12-16	Jackey	1. 第 1 版发布。
v1. 1	2021-12-17	Jackey	1. 增加 pyCar 组装教程。

## 版权声明

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

## 目录

第 1 章 MicroPython 简介 .....	7
1.1 MicroPython 是什么 .....	7
1.2 MicroPython 支持的微控制器平台 .....	8
1.3 MicroPython 相关学习资料 .....	10
1.3.1 pyCar 官方文档 .....	10
1.3.2 Github 开源 .....	11
1.3.3 O1Studio 技术论坛 .....	12
1.3.4 MicroPython 库文档 .....	13
1.3.5 MicroPython 官方网站 .....	14
1.4 pyCar 开发套件介绍与组装 .....	15
1.4.1 pyCar 硬件资源 .....	16
1.4.2 底盘 .....	17
1.4.3 电机 .....	18
1.4.4 轮胎 .....	19
1.4.5 OLED 显示屏 .....	20
1.4.6 超声波传感器 .....	21
1.4.7 光电测速传感器 .....	22
1.4.8 锂电池 .....	22
1.4.9 红外遥控器 .....	23
1.4.10 Type-C 数据线 .....	24
1.4.11 其它组裝件 .....	24
1.4.12 pyCar 组裝 .....	25
第 2 章 Python 基础知识 .....	36
2.1 原始数据类型和运算符 .....	36
2.2 变量和集合 .....	41
2.3 流程控制和迭代器 .....	47
2.4 函数 .....	51
2.5 类 .....	54
2.6 模块 .....	56
2.7 高级用法 .....	57
第 3 章 开发环境快速建立 .....	59
3.1 基于 Windows .....	60
3.1.1 安装开发软件 Thonny .....	60
3.1.2 开发套件使用 .....	62
3.2 基于 Mac OS .....	79
3.2.1 安装开发软件 Thonny .....	79
3.2.2 开发套件使用 .....	79
3.3 基于 Linux (树莓派) .....	82
3.3.1 安装开发软件 Thonny .....	82
3.3.2 开发套件使用 .....	82
第 4 章 基础实验 .....	83
4.1 pyCar 引脚简表 .....	84

4.2 点亮第一个 LED .....	85
4.3 按键 .....	90
4.4 外部中断 .....	95
4.5 定时器 .....	100
4.6 I2C 总线（OLED 显示屏） .....	104
4.7 RTC 实时时钟 .....	110
4.8 UART（串口通信） .....	116
第 5 章 WiFi 应用 .....	123
5.1 连接无线路由器 .....	124
5.2 Socket 通信 .....	130
5.3 MQTT 通信 .....	142
5.4 WebREPL .....	156
第 6 章 pyCar 子模块实验 .....	160
6.1 车头灯 .....	161
6.2 动作 .....	164
6.3 超声波测距 .....	168
6.4 码盘测路程 .....	174
6.5 红外遥控器 .....	180
第 7 章 综合实验 .....	185
7.1 红外遥控车 .....	186
7.2 避障小车 .....	191
7.3 巡线小车 .....	195

# 第1章 MicroPython 简介

## 1.1 MicroPython 是什么

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

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

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

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

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

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

## 1.2 MicroPython 支持的微控制器平台

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

MicroPython 最早支持的硬件平台是 STM32，开发板名称叫 pyboard。使用的芯片型号是：STM32F405RGT6，该芯片具备 1MB flash 和 196k SRAM，168MHZ 主频。

除此之外，上海乐鑫的 WIFI 芯片 ESP8266/ESP32 也非常成熟。用户使用 MicroPython 可以快速开发物联网相关应用，实现 WIFI 无线连接。本书主要是围绕 pyCar（ESP32 平台）来进行编写。

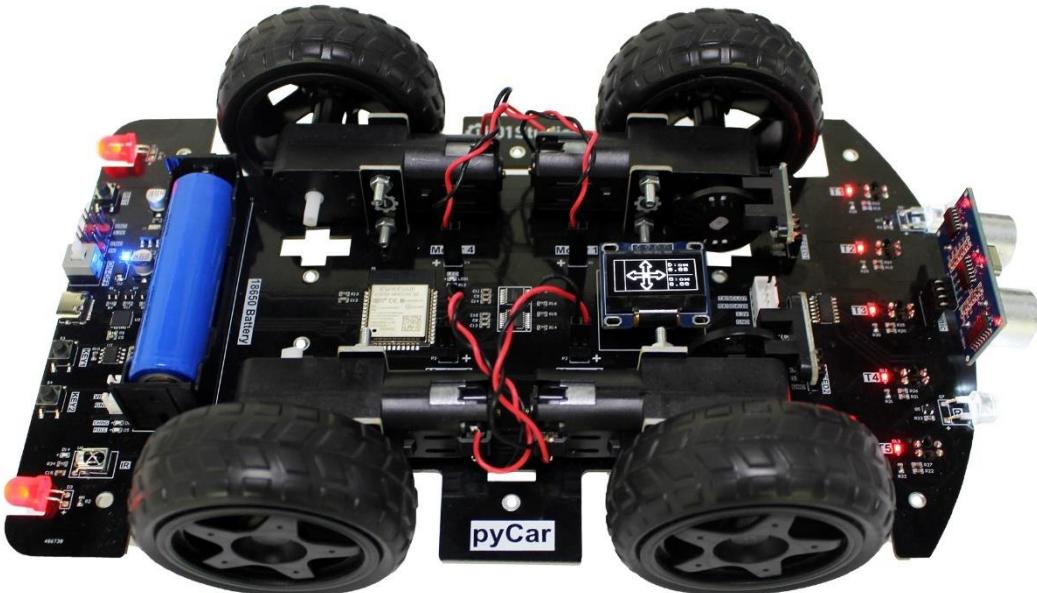


图 1-1 pyCar 开发套件

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

以下是 O1Studio 其它 micropython 开发平台：

01Studio MicroPython全系列开发板			
达芬奇 (TKM32F499)	达芬奇TKM32F499开发板 		
哥伦布 (STM32F407)	哥伦布STM32F407开发板 	pyBoard Pro 	pyBoard Plus 
pyBoard (STM32F405)	Micro-Dython开发板 	pyboard v1.1-CN 	pyBoard Mini 
pyPico (RP2040)	PVPico开发套件 		
pyWiFi-ESP8266	pyWiFi-ESP8266开发套件 	pyWiFi-ESP8266 	
pyWiFi-ESP32-C3	pyWiFi-ESP32-C3开发套件 	pyWiFi-ESP32-C3 	
pyWiFi-ESP32	pyWiFi-ESP32开发套件 	pyWiFi-ESP32 	pyWiFi-ESP32D 
pyWiFi-ESP32-S2	pyWiFi-ESP32-S2D开发套件 		
pyBLE-NRF52840	pyBLE-NRF52840开发套件 	pyBLE-NRF52840 	
py4G-EC600	py4G-EC600开发套件 		
pyAI-OpenMV4	pyAI-MV4开发套件 	pyAI-MV4 Plus 	
pyAI-K210	pyAI-K210开发套件 	pyAI-K210核心板 	

## 1.3 MicroPython 相关学习资料

### 1.3.1 pyCar 官方文档

<https://pycar.01studio.cc/> , 包含 pyCar 项目介绍和 micropython 库使用说明。



图 1-2

### 1.3.2 Github 开源

<https://github.com/01studio-lab/pyCar> 项目开源代码，包含资料、例程代码、库代码。

01studio-lab / pyCar Public

Code Issues Pull requests 1 Actions Projects Wiki Security Insights Set

main 1 branch 0 tags Go to file Add file Code

Commit	Message	Time
CaptainJackey add IR control	add IR control	8 days ago
code	add IR control	8 days ago
docs	发布	9 days ago
firmware	发布	9 days ago
hardware	发布	9 days ago
.gitignore	Initial commit	24 days ago
LICENSE	Initial commit	24 days ago
README.md	add docs link	9 days ago
update.md	发布	9 days ago

图 1-3

最新的示例优先发布到 Githua，欢迎关注和贡献。

01studio-lab / pyCar Public

Code Issues Pull requests 1 Actions Projects Wiki Security

main pyCar / code /

CaptainJackey add Ultrasound\_avoidance example

..

examples 实例

ble.py ble

car.py Update car.py

ssd1306.py add screen

图 1-4

### 1.3.3 01Studio 技术论坛

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

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

论坛开设开源项目--**pyCar** 专区，用于玩车交流分享。

The screenshot shows the 01Studio forum homepage. On the left, there's a sidebar with categories like 'MicroPython开发板' and 'Linux Python开发板'. The main area displays a post by user 'oyyyy' with the title '计数器' and a post by user 'Jackey' with the title '【腾讯云物联网应用】MicroPython ESP32温湿度传感器DHT11数据采集'. The right side features a '推荐内容' sidebar with several recommended posts.

图 1-5 01Studio 技术论坛

### 1.3.4 MicroPython 库文档

限于篇幅，本教程部分实验只介绍 ESP32 和 pyCar 关键的函数和模块应用，如果在学习过程中希望深入了解所有 MicroPython 函数和模块，请查阅：

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



图 1-6 MicroPython 库文档（中文）

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

### 1.3.5 MicroPython 官方网站

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

英文版官网有官方文档(DOCS)和英文论坛，适合比较英语比较好的小伙伴。

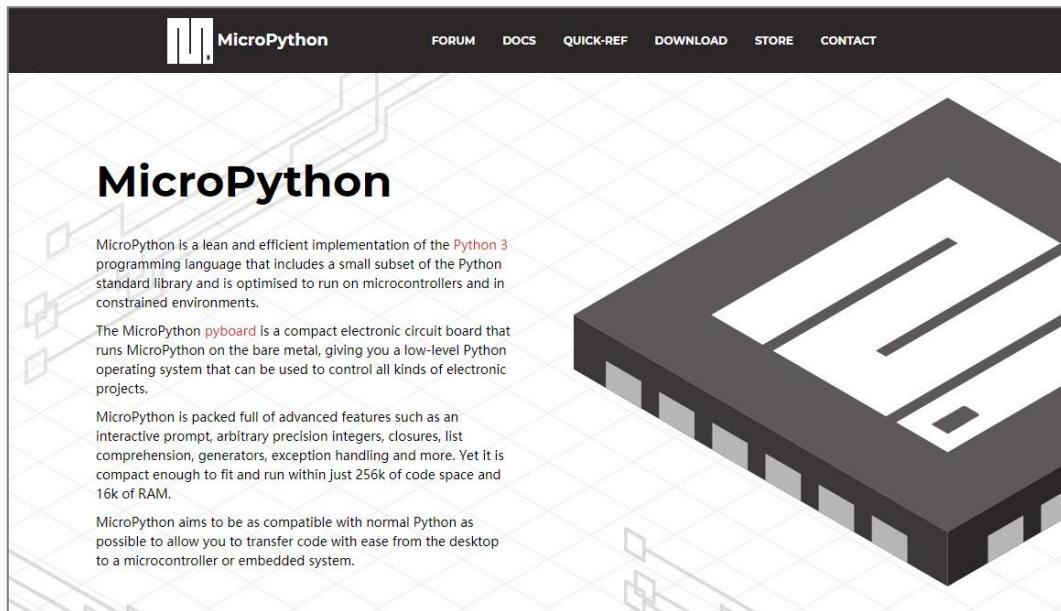


图 1-7 MicroPython 官网

## 1.4 pyCar 开发套件介绍与组装

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

而 pyCar 则是基于 ESP32 平台（考虑到对 WiFi 和蓝牙的支持），是一个全开源项目，旨在让用户能基于 micropython 做出好玩的应用，从而进一步普及 micropython。

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

### 1.4.1 pyCar 硬件资源

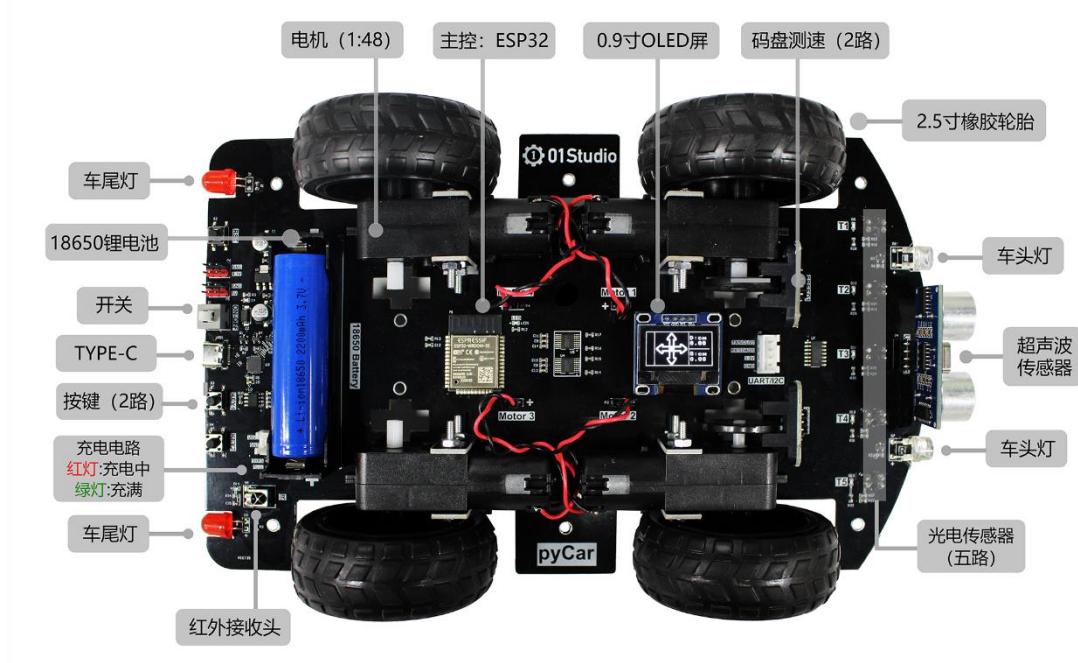


图 1-8 pyCar 资源描述

- 主控: ESP32-WROOM-32 (Flash:4MBytes) 支持 WiFi/BLE
- 4 x TT 马达。支持 PWM 调速
- 2 x 编码盘测速
- 5 x 光电传感器, 用于巡线
- 1 x 超声波传感器, 用于避障
- 1 x 0.96 寸 OLED 显示屏
- 1 x 红外遥控
- 2 x 车头草帽灯 (可控制)
- 2 x 车尾灯 (常亮)
- 2 x 按键
- 1 x LED
- 1 x 锂电池座 18650 (带充电电路)

### 1.4.2 底盘

pyCar 底盘由 01Studio 设计研发，基于 ESP32 平台，一体化 PCB 板设计，简单易用。

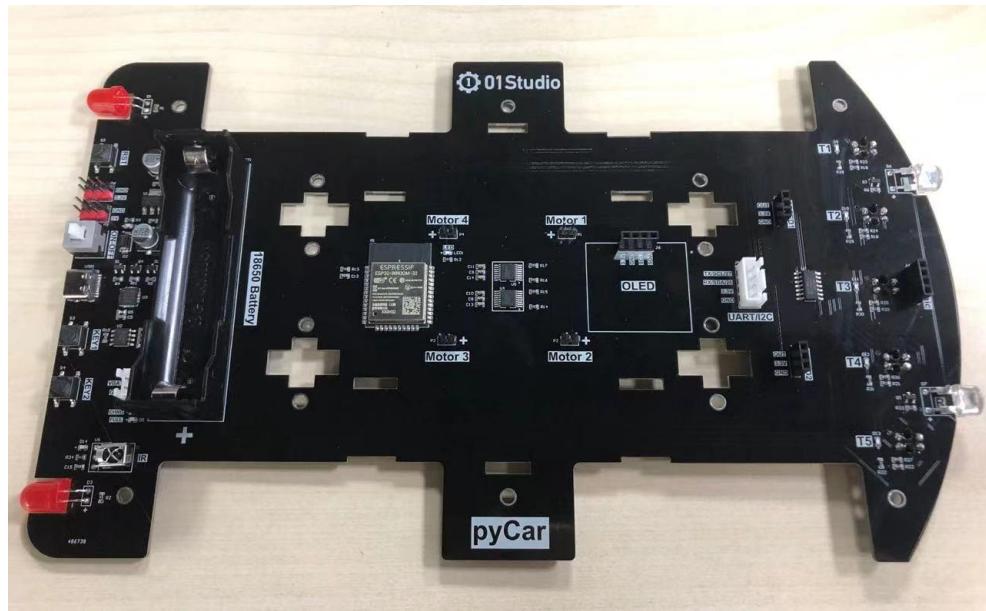


图 1-9 pyCar 底盘

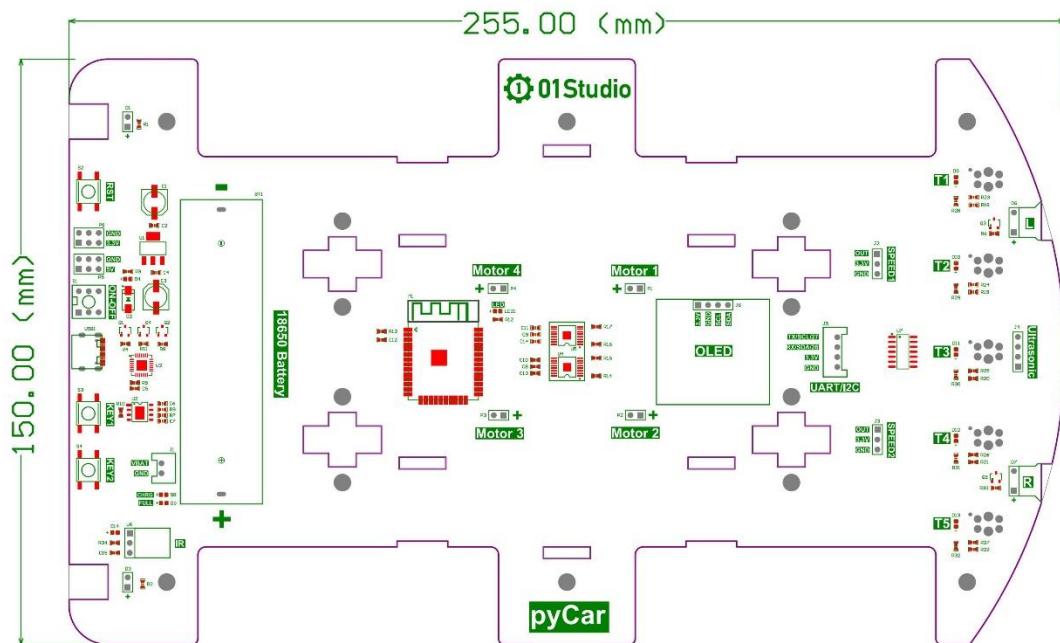


图 1-10 pyCar 尺寸图

### 1.4.3 电机

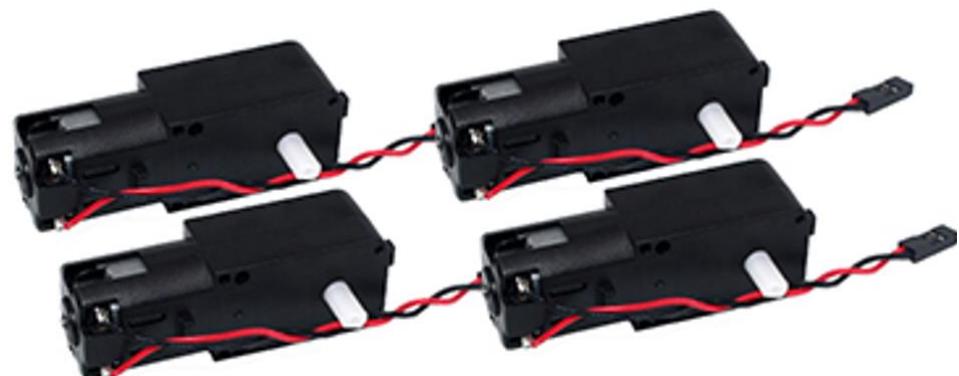


图 1-11 直流减速电机

功能参数	
工作电压	3-6V
减速比	1:48
无负载转速	6V 200RPM±10%
通讯方式	I2C 总线

图 1-12

#### 1.4.4 轮胎



图 1-13 轮胎

功能参数	
尺寸	直径: 65mm , 宽度: 27mm
材质	橡胶
重量	33g

图 1-14

#### 1.4.5 OLED 显示屏



图 1-15 OLED 显示屏

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

表 1-1

#### 1.4.6 超声波传感器



图 1-16 超声波传感器模块

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

表 1-2

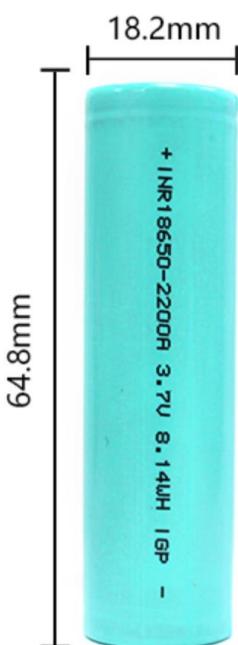
#### 1.4.7 光电测速传感器



图 1-17 光电测速模块

#### 1.4.8 锂电池

标准 18650 锂电池，容量 2200mAh。



- 型号：18650锂电池
- 电压：3.7V
- 直径： $18.2 \pm 0.1$  mm
- 高度： $64.8 \pm 0.2$  mm

图 1-18 18650 锂电池

#### 1.4.9 红外遥控器



图 1-19 红外遥控器

#### 1.4.10 Type-C 数据线

用于开发、测试、充电多合一。



图 1-20

#### 1.4.11 其它组装件

含码盘（测速用）、电机固定柱、螺丝和螺母。

码盘 2个	电机固定柱 8个	M3*30螺丝 8个	M3螺母 10个

图 1-21 pyCar 组装件

### 1.4.12 pyCar 组装

拿到套件后先检查 pyCar 默认配置清单。



图 1-22 pyCar 清单

pyCar 采用 PCB 一体化设计，无需外接杜邦线，安装非常简单，以下是详细的安装步骤：

将电机固定柱从底盘底部插入。

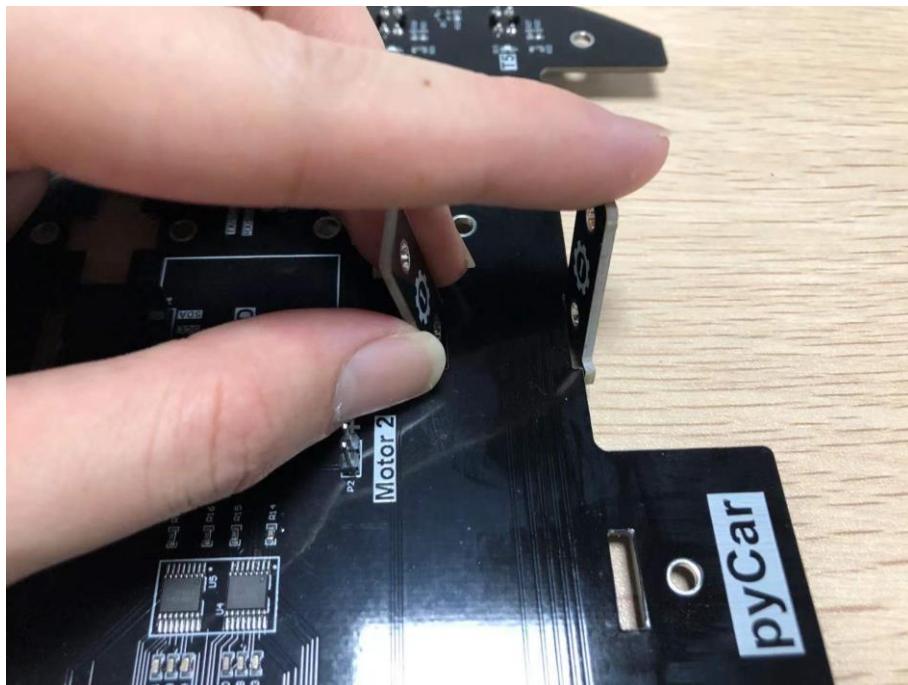


图 1-23

放进电机，注意电机引线焊接处朝外。

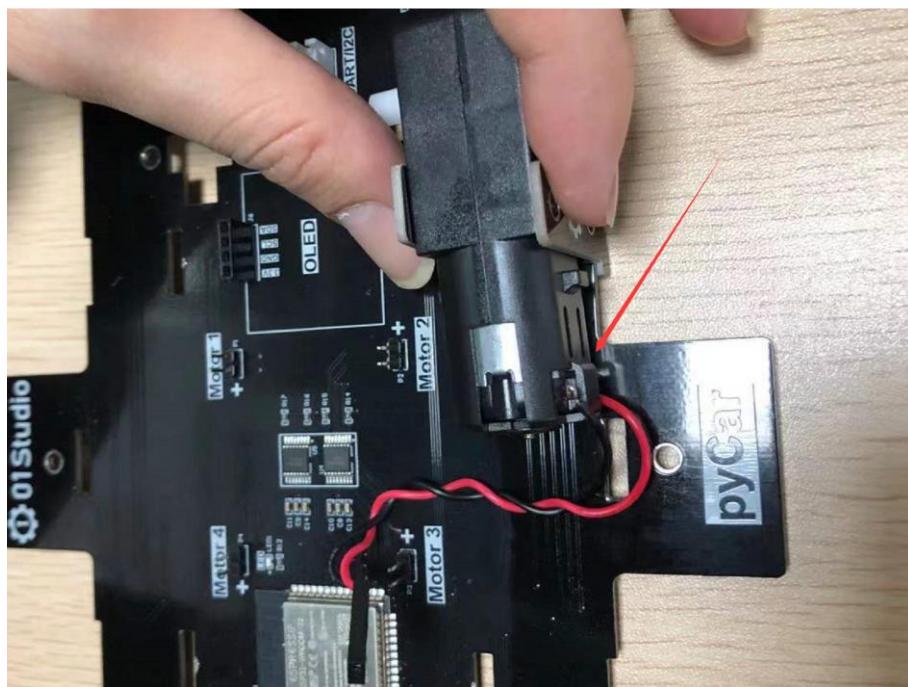


图 1-24

将 2 个长螺丝从外侧插入，穿过电机固定柱和电机孔。

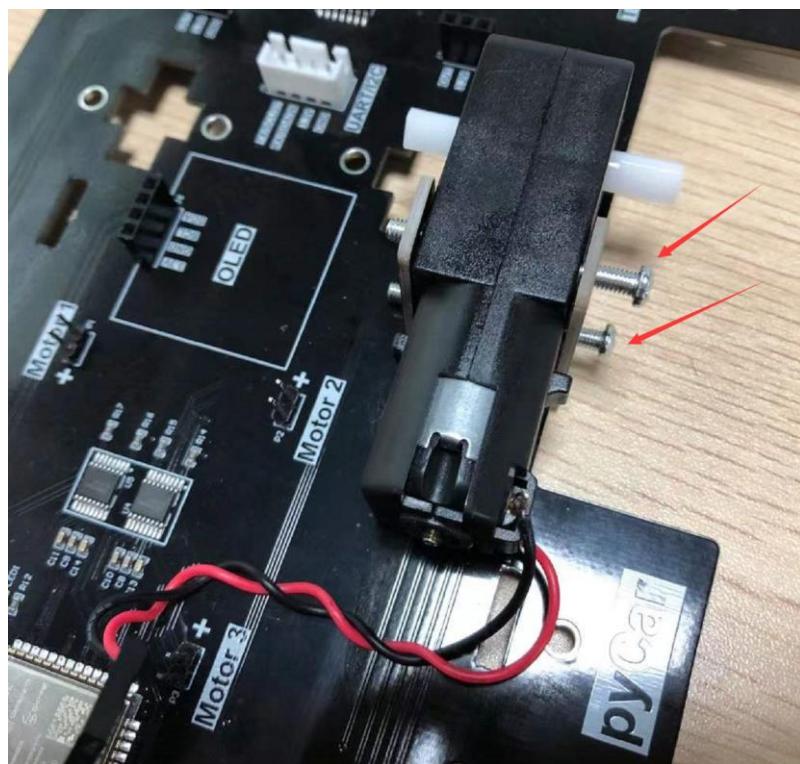


图 1-25

使用螺母拧紧固定。

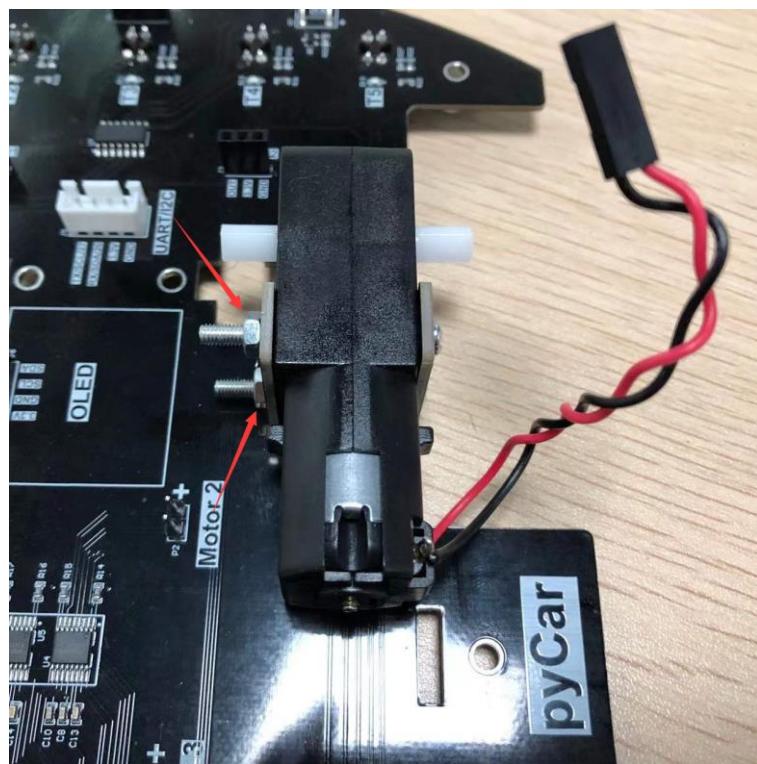


图 1-26

安装电机后如感觉有点松动属于正常现象，不影响行驶使用，想紧一点的话可以在电机和底板接触面垫个纸片或贴个胶布再装电机。

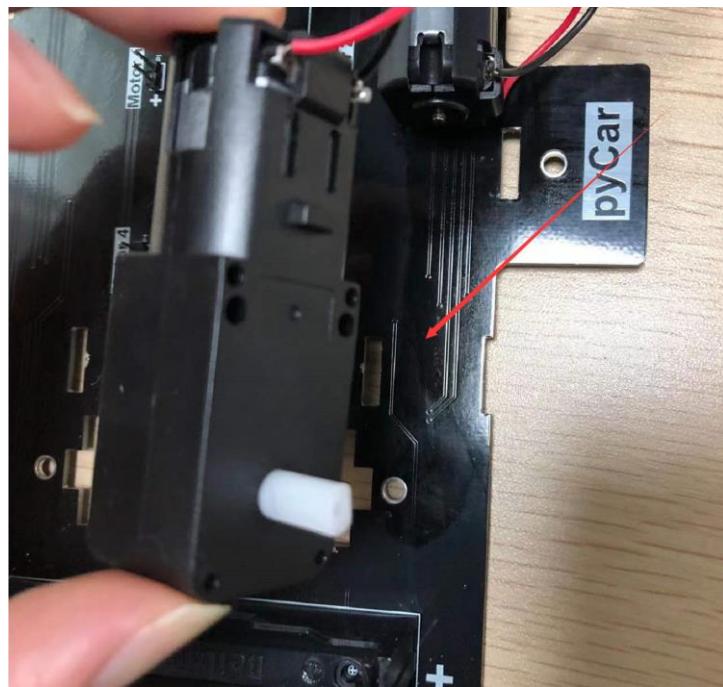


图 1-27

使用同样的方法将 4 路电机组装好，注意引线焊接处均朝外。

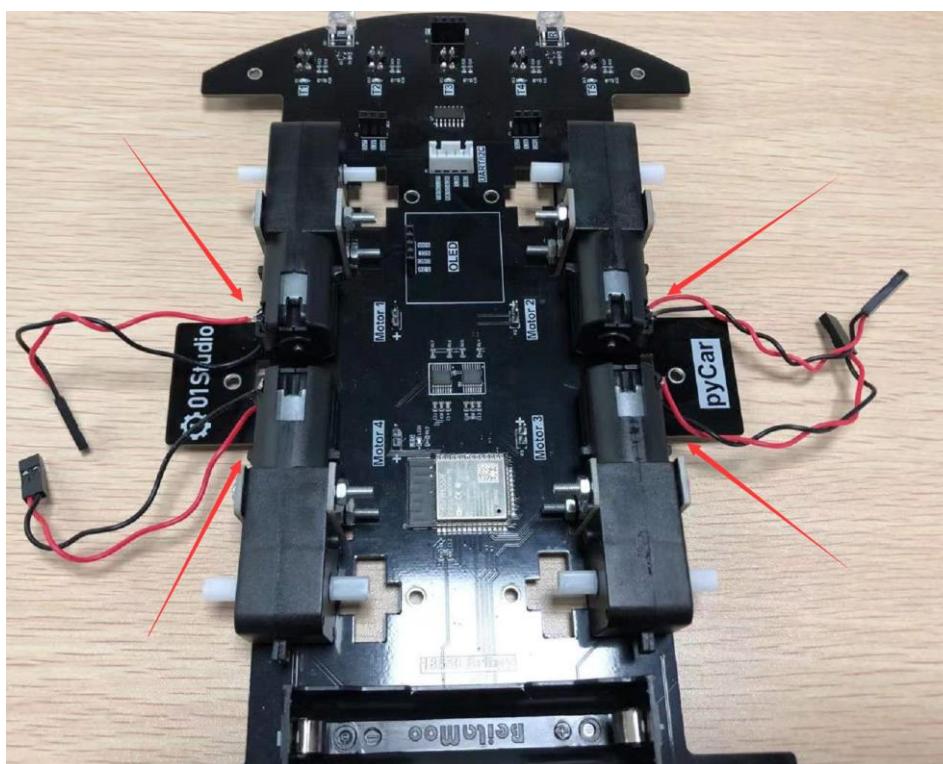


图 1-28

接下来安装轮胎，电机和轮子是长条形 TT 接口，务必对准后方可插入。

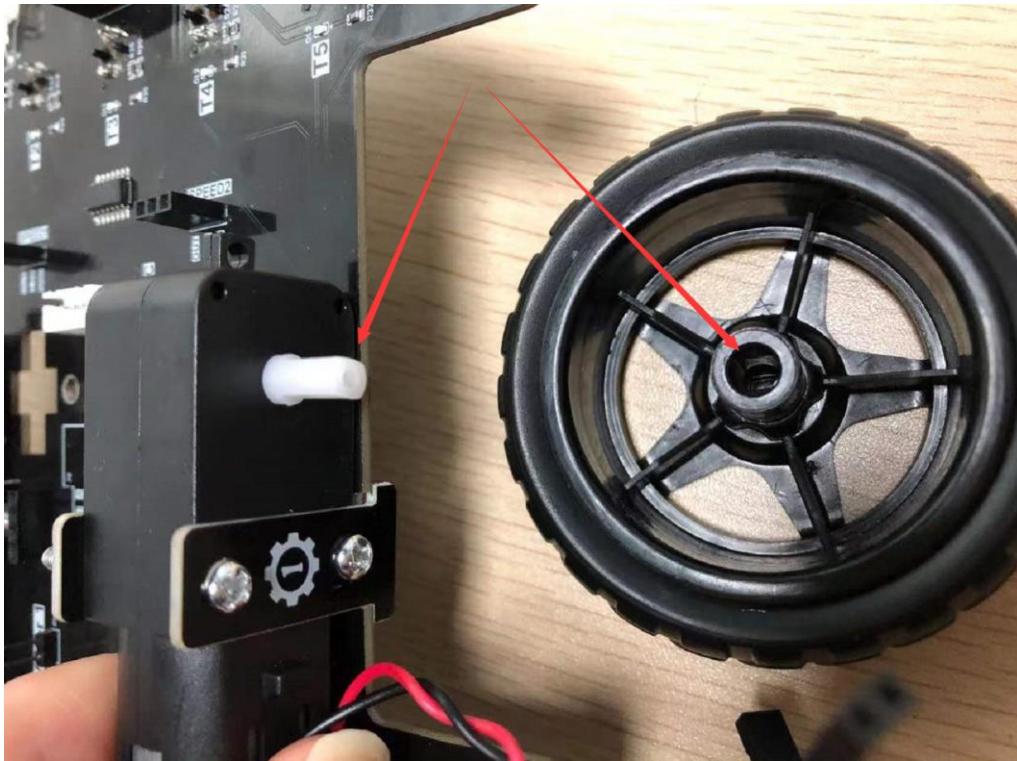


图 1-29

轮胎安装时必须用手将电机和底盘固定，避免插入过程损坏固定柱导致松动。

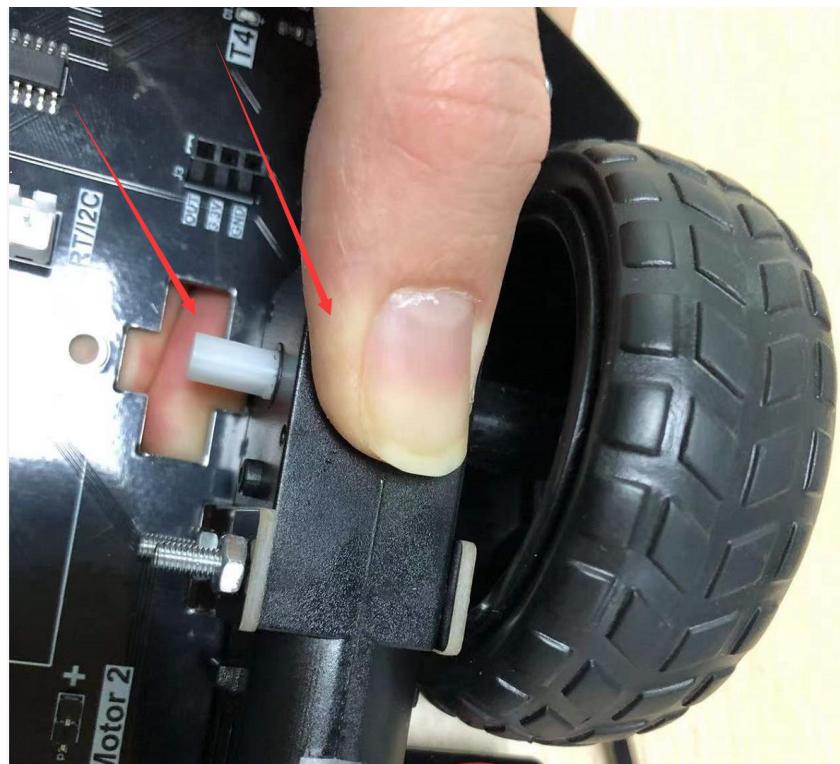


图 1-30

用同样方法将 4 个论坛组装好。

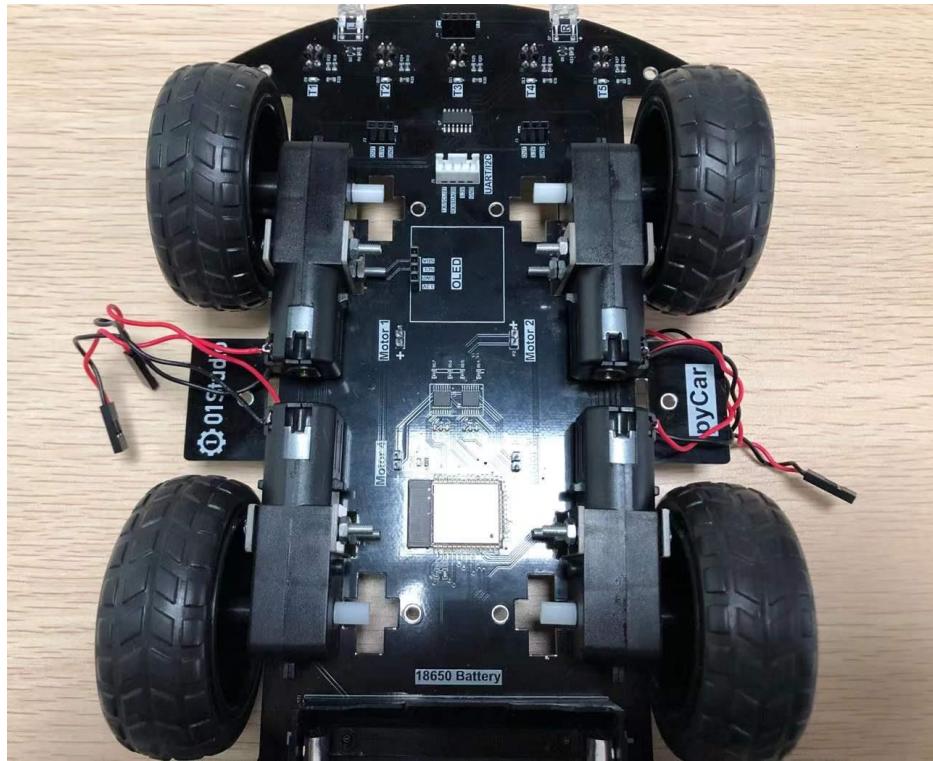


图 1-31

将码盘安装到前轮位置：

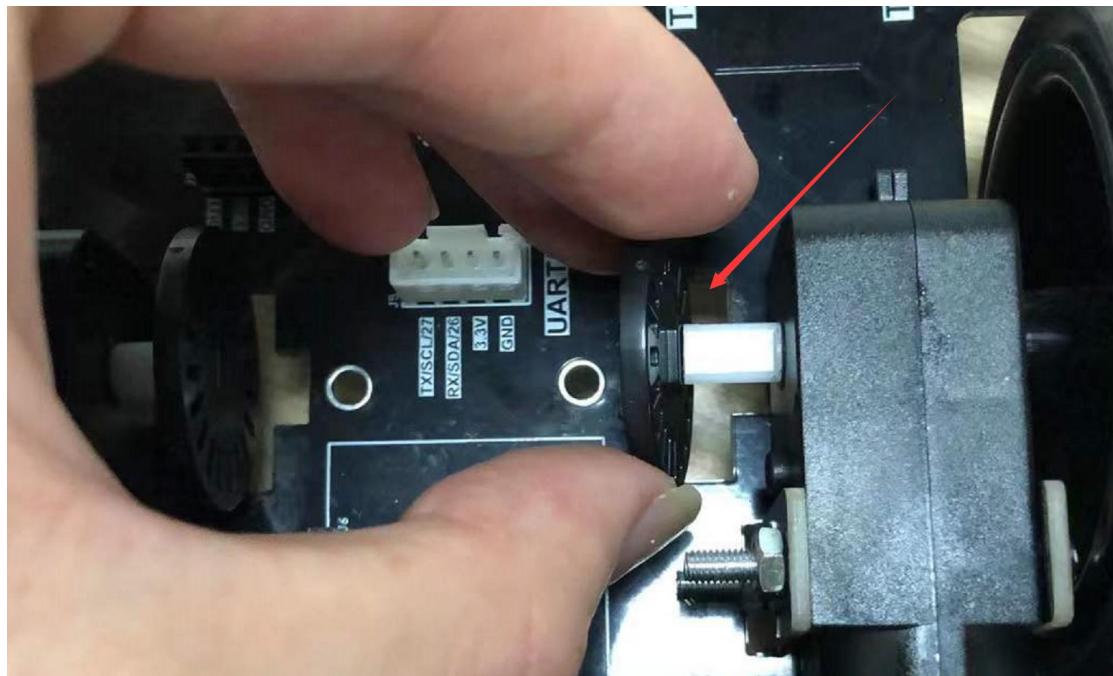


图 1-32

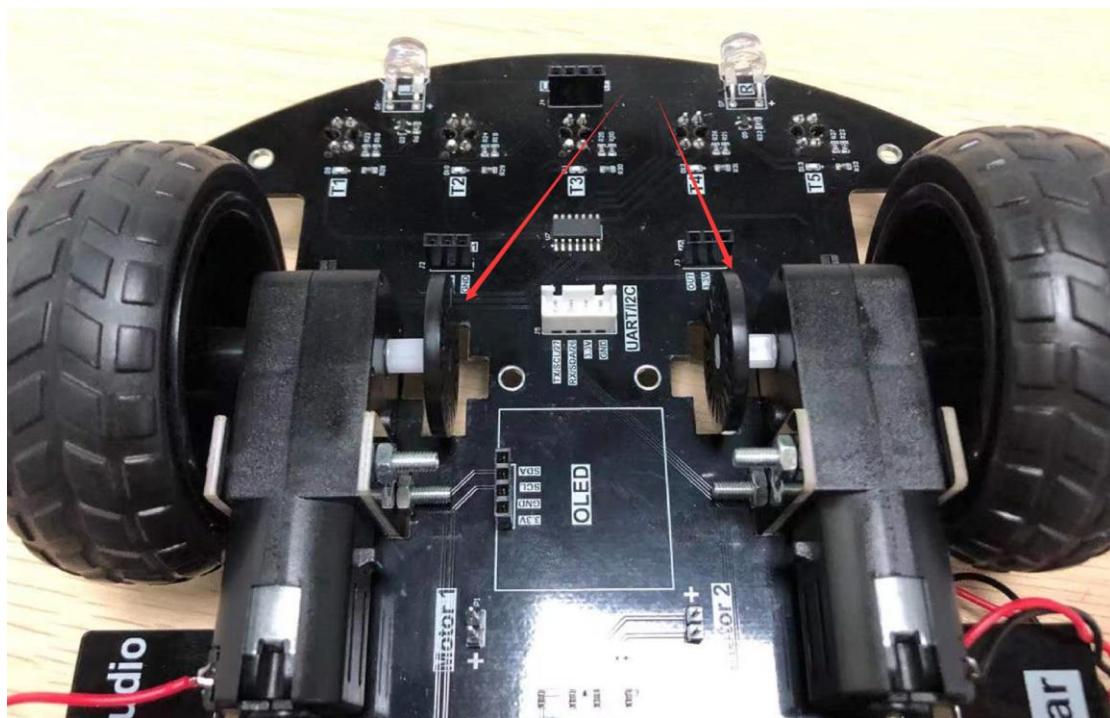


图 1-33

按图示位置插入 2 个测速传感器。

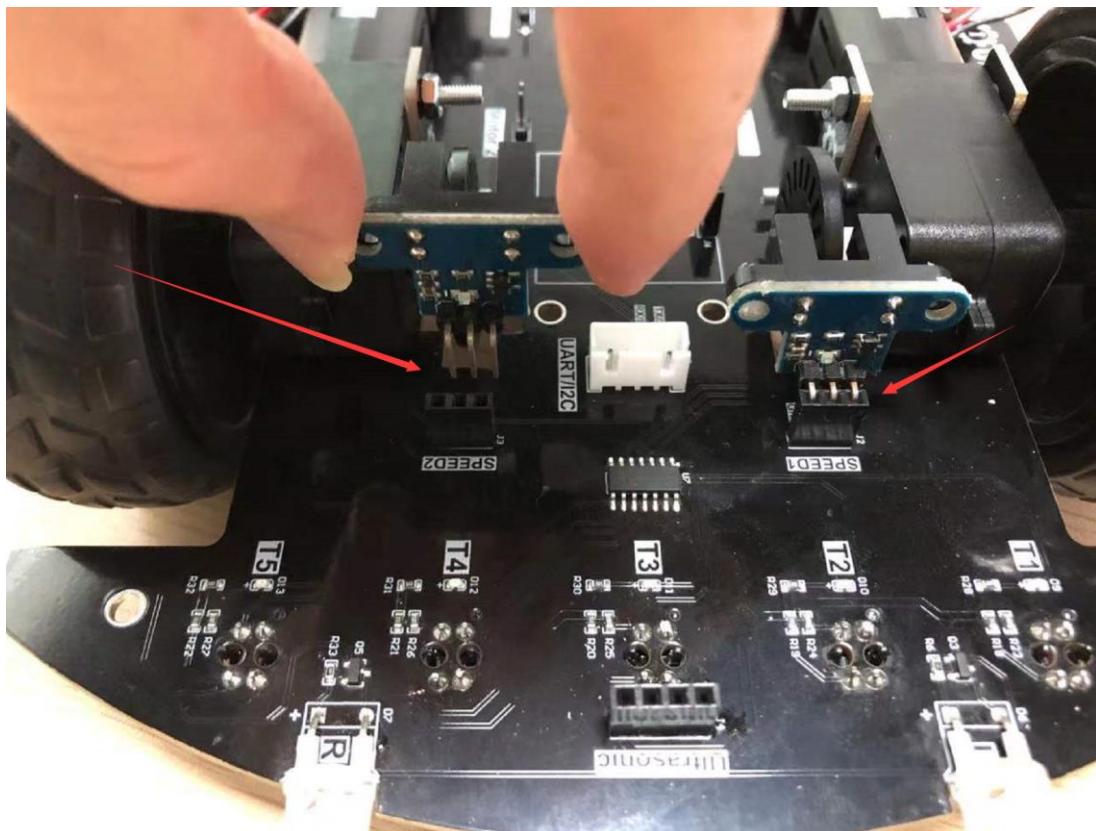


图 1-34

插上 OLED 显示屏：

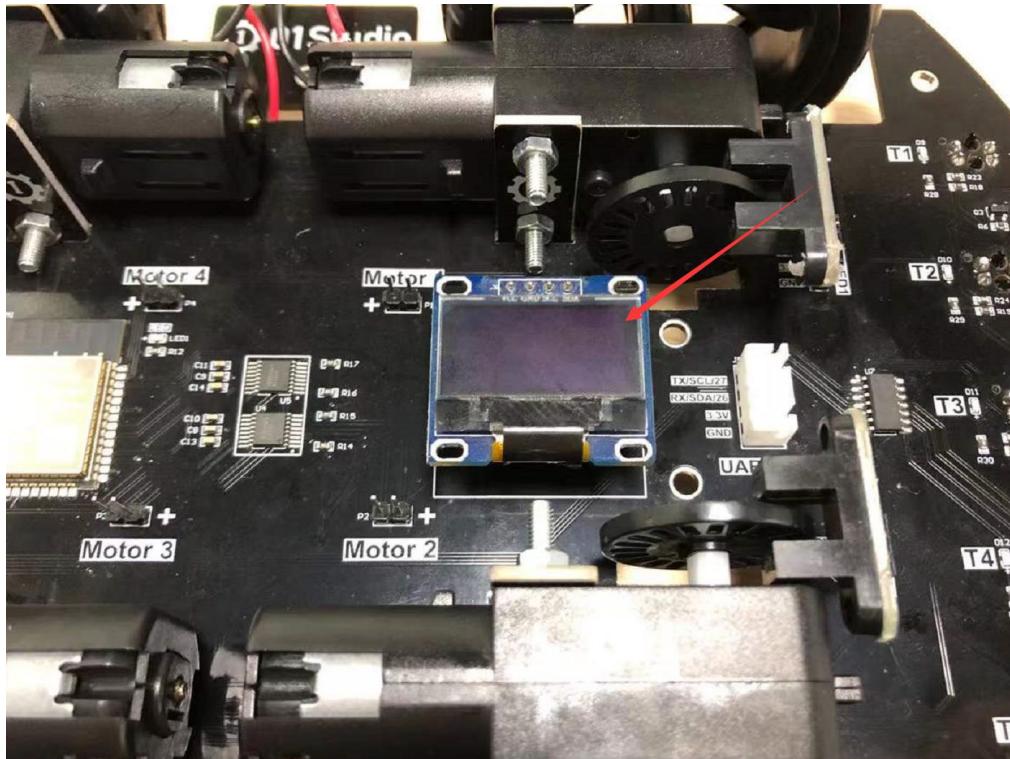


图 1-35

车头处插上超声波传感器。

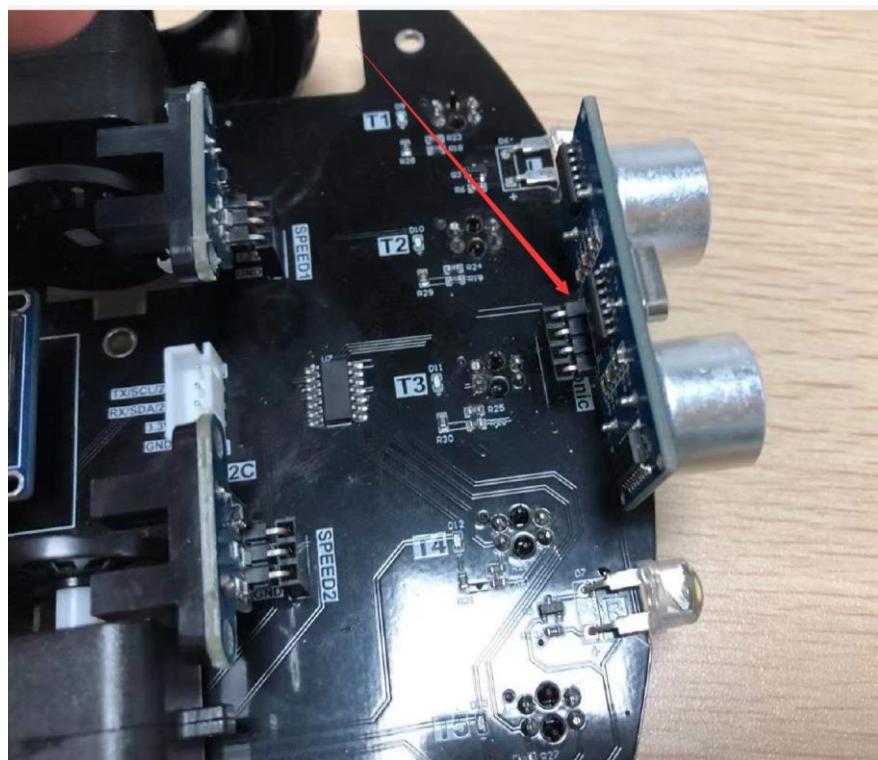


图 1-36

然后将电机引线接到底盘排针，注意红色接正极（开发板丝印“+”）。

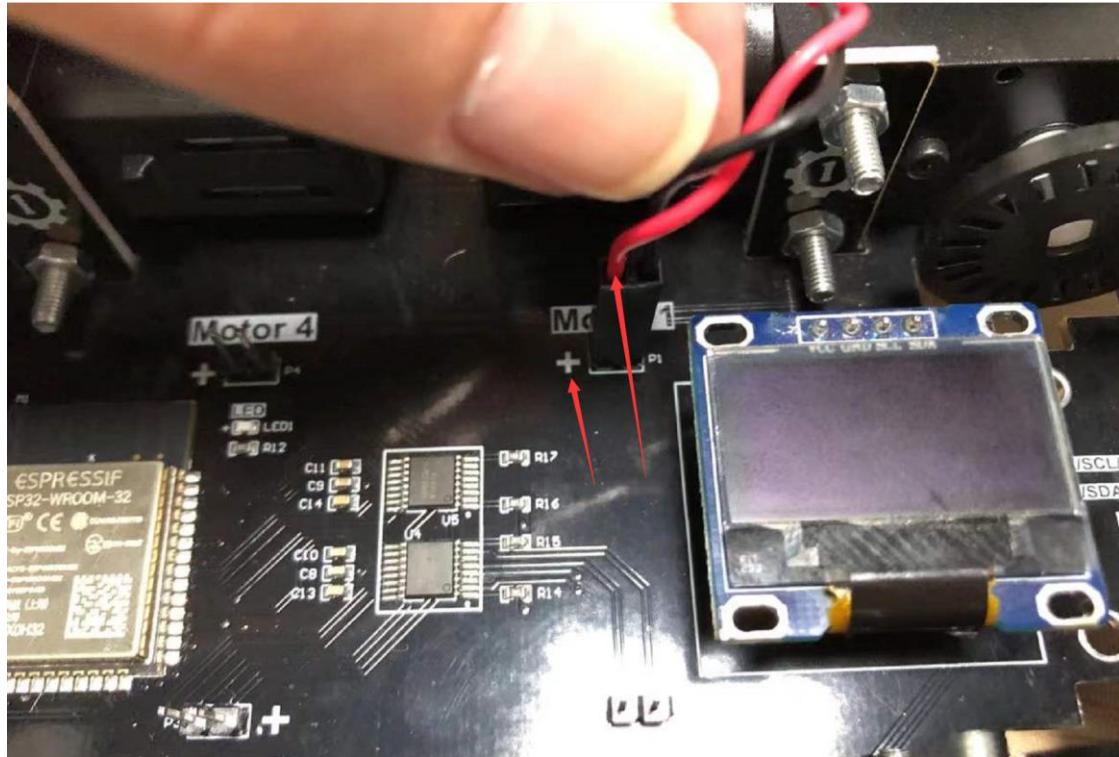


图 1-37

同样方法将 4 个电机引线接好。

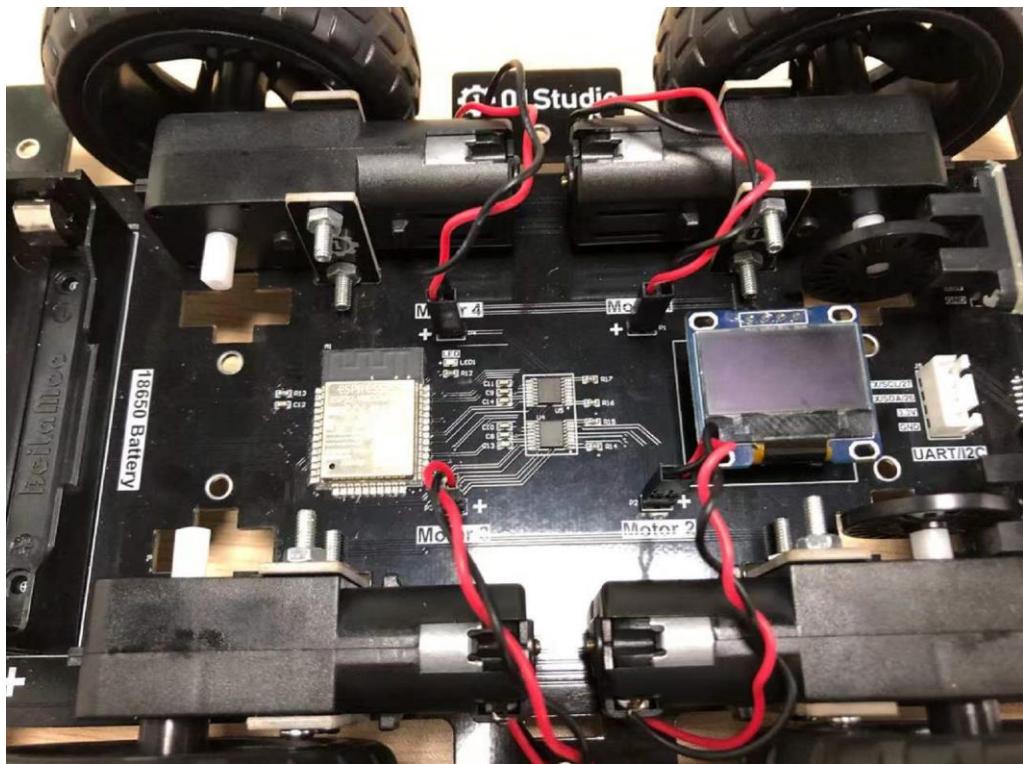


图 1-38

最后安装锂电池，注意电池正负极要与底盘丝印方向一致。

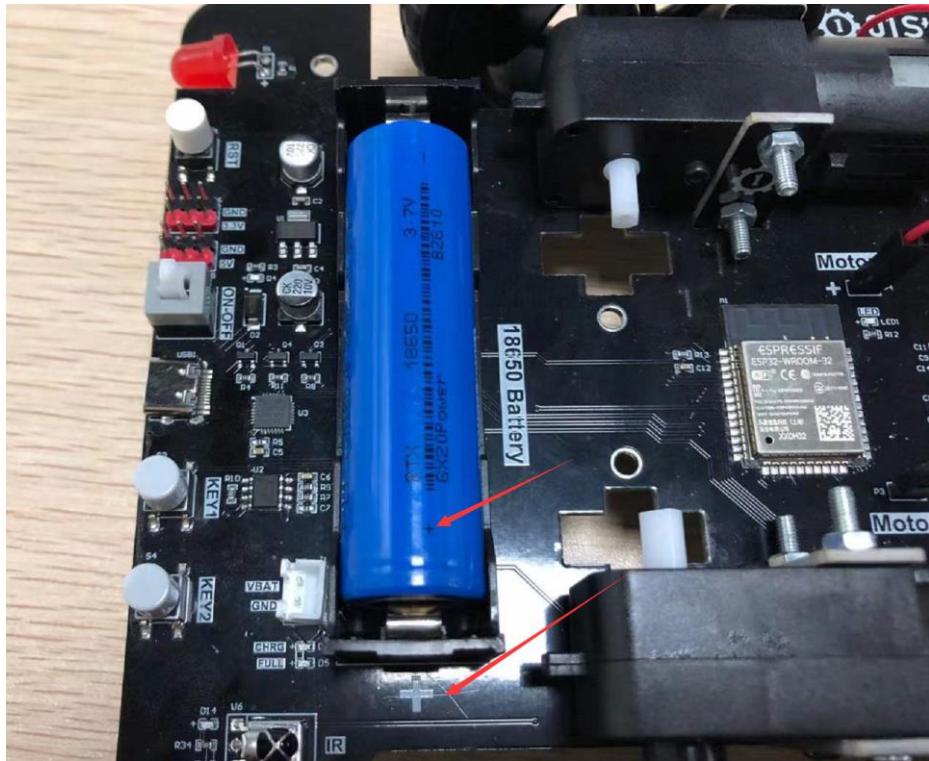


图 1-39

按下车尾的开关，灯亮说明供电正常。

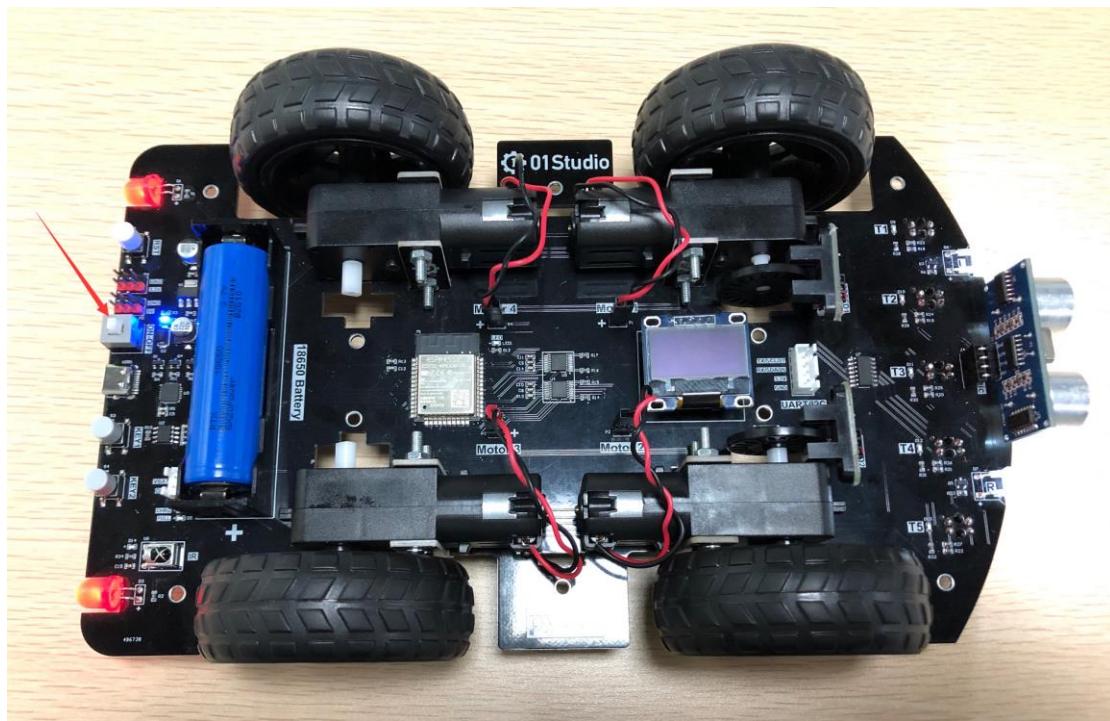


图 1-40

转动 2 个前轮，测速器上的红色灯有闪亮说明检测到转动，测速正常。若没有可以前后掰动传感器适当调整位置。

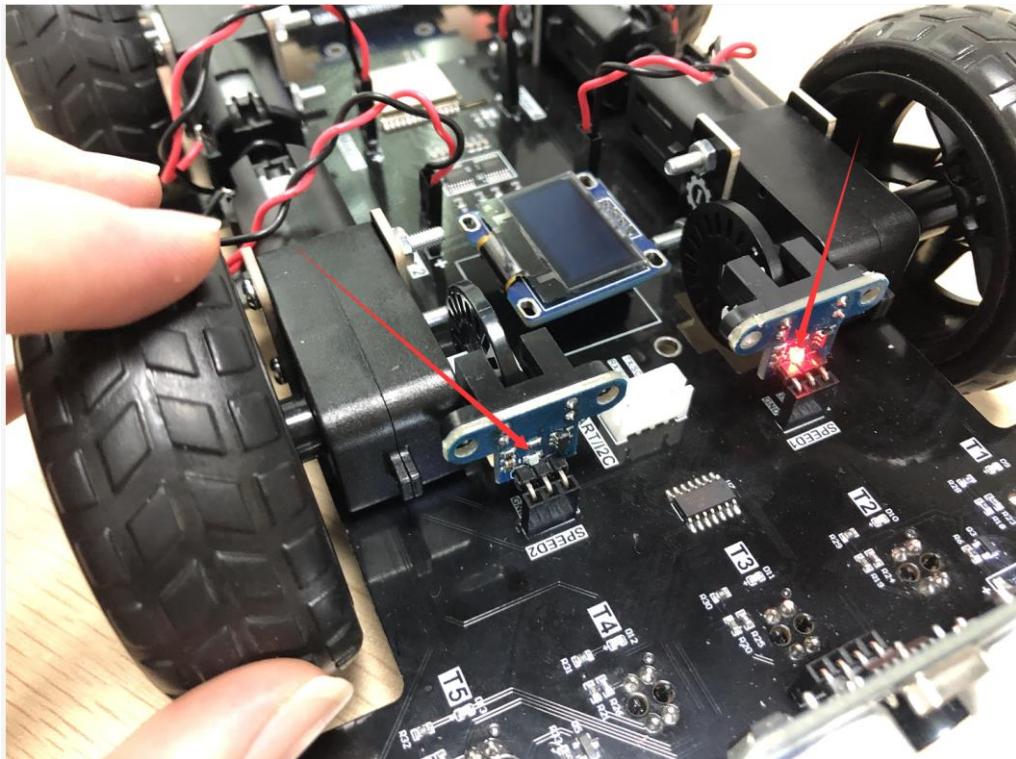


图 1-41

至此，pyCar 组装完成：

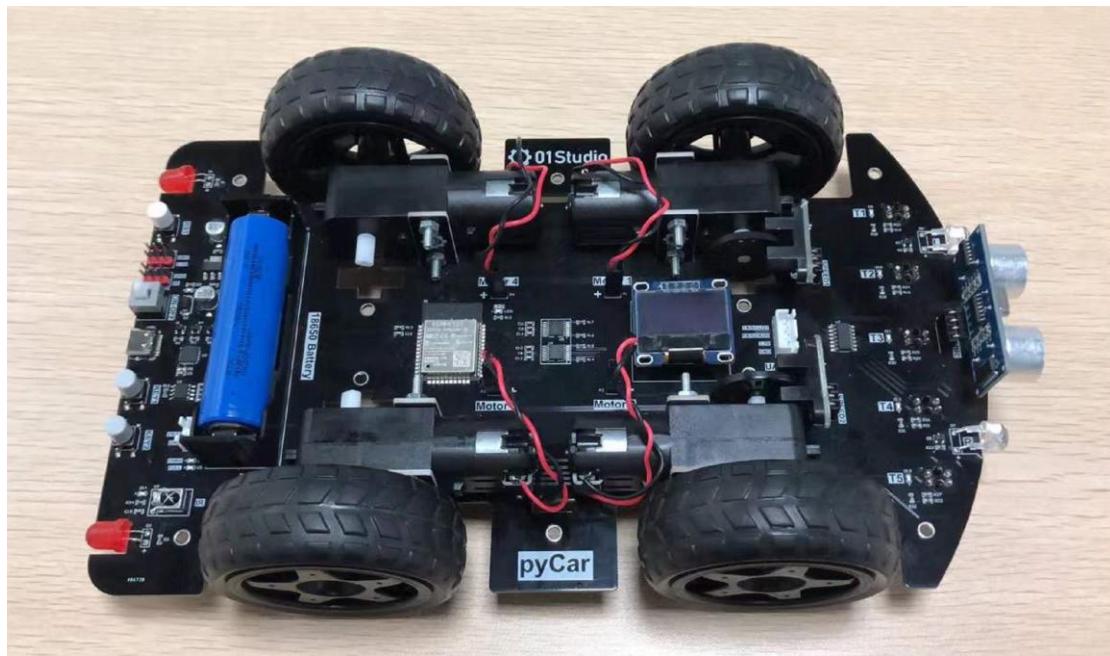


图 1-42 组装后的 pyCar

## 第2章 Python 基础知识

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

【原著：Louie Dinh，翻译：Geoff Liu】

```
# 用井字符开头的是单行注释
```

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

### 2.1 原始数据类型和运算符

```
#####
## 1. 原始数据类型和运算符
#####

# 整数
3 # => 3

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

```
# 但是除法例外，会自动转换成浮点数
35 / 5 # => 7.0
5 / 3 # => 1.6666666666666667

# 整数除法的结果都是向下取整
5 // 3      # => 1
5.0 // 3.0 # => 1.0 # 浮点数也可以
-5 // 3    # => -2
-5.0 // 3.0 # => -2.0

# 浮点数的运算结果也是浮点数
3 * 2.0 # => 6.0

# 模除
7 % 3 # => 1

# x 的 y 次方
2**4 # => 16

# 用括号决定优先级
(1 + 3) * 2 # => 8

# 布尔值
True
False

# 用 not 取非
not True # => False
not False # => True
```

```
# 逻辑运算符，注意 and 和 or 都是小写
```

```
True and False # => False
```

```
False or True # => True
```

```
# 整数也可以当作布尔值
```

```
0 and 2 # => 0
```

```
-5 or 0 # => -5
```

```
0 == False # => True
```

```
2 == True # => False
```

```
1 == True # => True
```

```
# 用==判断相等
```

```
1 == 1 # => True
```

```
2 == 1 # => False
```

```
# 用!=判断不等
```

```
1 != 1 # => False
```

```
2 != 1 # => True
```

```
# 比较大小
```

```
1 < 10 # => True
```

```
1 > 10 # => False
```

```
2 <= 2 # => True
```

```
2 >= 2 # => True
```

```
# 大小比较可以连起来！
```

```
1 < 2 < 3 # => True
```

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

```
# 字符串用单引双引都可以
"这是个字符串"
'这也是个字符串'

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

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

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

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

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

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

# None 是一个对象
None # => None

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

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

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

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

## 2.2 变量和集合

```
#####
## 2. 变量和集合
#####

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

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

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

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

# 用 append 在列表最后追加元素
li.append(1)    # li 现在是[1]
li.append(2)    # li 现在是[1, 2]
li.append(4)    # li 现在是[1, 2, 4]
li.append(3)    # li 现在是[1, 2, 4, 3]
# 用 pop 从列表尾部删除
```

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

li.append(3)    # li 变回[1, 2, 4, 3]

# 列表存取跟数组一样

li[0]  # => 1
# 取出最后一个元素

li[-1] # => 3

# 越界存取会造成 IndexError

li[4] # 抛出 IndexError

# 列表有切割语法

li[1:3] # => [2, 4]
# 取尾

li[2:] # => [4, 3]
# 取头

li[:3] # => [1, 2, 4]
# 隔一个取一个

li[::-2] # => [1, 4]
# 倒排列表

li[::-1] # => [3, 4, 2, 1]
# 可以用三个参数的任何组合来构建切割

# li[始:终:步伐]

# 用 del 删除任何一个元素

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

# 列表可以相加

# 注意: li 和 other_li 的值都不变
```

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

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

# 用 in 测试列表是否包含值
1 in li  # => True

# 用 len 取列表长度
len(li)  # => 6

# 元组是不可改变的序列
tup = (1, 2, 3)
tup[0]  # => 1
tup[0] = 3  # 抛出 TypeError

# 列表允许的操作元组大都可以
len(tup)  # => 3
tup + (4, 5, 6)  # => (1, 2, 3, 4, 5, 6)
tup[:2]  # => (1, 2)
2 in tup  # => True

# 可以把元组合成列表解包，赋值给变量
a, b, c = (1, 2, 3)  # 现在 a 是 1, b 是 2, c 是 3
# 元组周围的括号是可以省略的
d, e, f = 4, 5, 6
# 交换两个变量的值就这么简单
e, d = d, e  # 现在 d 是 5, e 是 4
```

```
# 用字典表达映射关系

empty_dict = {}

# 初始化的字典

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

# 用[]取值

filled_dict["one"] # => 1

# 用 keys 获得所有的键。

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

# 注意：字典键的顺序是不定的，你得到的结果可能和以下不同。

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

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

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

# 用 in 测试一个字典是否包含一个键

"one" in filled_dict # => True

1 in filled_dict # => False

# 访问不存在的键会导致 KeyError

filled_dict["four"] # KeyError

# 用 get 来避免 KeyError

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

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

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

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

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

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

# 可以把集合赋值于变量
filled_set = some_set

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

# & 取交集
```

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

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

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

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

## 2.3 流程控制和迭代器

```
#####
## 3. 流程控制和迭代器
#####

# 先随便定义一个变量
some_var = 5

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

"""

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

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

"""


```

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

```
打印：
```

```
0  
1  
2  
3  
....  
  
for i in range(4):  
    print(i)
```

```
....
```

```
while 循环直到条件不满足
```

```
打印：
```

```
0  
1  
2  
3  
....  
  
x = 0  
  
while x < 4:  
    print(x)  
    x += 1 # x = x + 1 的简写
```

```
# 用 try/except 块处理异常状况
```

```
try:
```

```
    # 用 raise 抛出异常  
    raise IndexError("This is an index error")  
  
except IndexError as e:  
    pass # pass 是无操作，但是应该在这里处理错误  
  
except (TypeError, NameError):
```

```
pass      # 可以同时处理不同类的错误

else:    # else 语句是可选的，必须在所有的 except 之后
    print("All good!")  # 只有当 try 运行完没有错误的时候这句才会运行

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

# 的对象。比如说上面 range 返回的对象就是可迭代的。

filled_dict = {"one": 1, "two": 2, "three": 3}
our_iterable = filled_dict.keys()

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

# 可迭代对象可以遍历

for i in our_iterable:
    print(i)    # 打印 one, two, three

# 但是不可以随机访问

our_iterable[1] # 抛出 TypeError

# 可迭代对象知道怎么生成迭代器

our_iterator = iter(our_iterable)

# 迭代器是一个可以记住遍历的位置的对象

# 用 __next__ 可以取得下一个元素

our_iterator.__next__() # => "one"

# 再一次调取 __next__ 时会记得位置

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

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

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

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

## 2.4 函数

```
#####
## 4. 函数
#####

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

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

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

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

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

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

```

# 我们来看看结果是什么:

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


# 这两种可变参数可以混着用

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

# 调用可变参数函数时可以做跟上面相反的，用*展开序列，用**展开字典。

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


# 函数作用域

x = 5


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

```

```
def setGlobalX(num):  
    global x  
    print (x) # => 5  
    x = num # 现在全局域的 x 被赋值  
    print (x) # => 6  
  
setX(43)  
setGlobalX(6)  
  
# 函数在 Python 是一等公民  
def create_adder(x):  
    def adder(y):  
        return x + y  
    return adder  
  
add_10 = create_adder(10)  
add_10(3) # => 13  
  
# 也有匿名函数  
(lambda x: x > 2)(3) # => True  
  
# 内置的高阶函数  
map(add_10, [1, 2, 3]) # => [11, 12, 13]  
filter(lambda x: x > 5, [3, 4, 5, 6, 7]) # => [6, 7]  
  
# 用列表推导式可以简化映射和过滤。列表推导式的返回值是另一个列表。  
[add_10(i) for i in [1, 2, 3]] # => [11, 12, 13]  
[x for x in [3, 4, 5, 6, 7] if x > 5] # => [6, 7]
```

## 2.5 类

```
#####
## 5. 类
#####

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

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

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

    # 种格式。

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

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

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

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

## 2.6 模块

```
#####
## 6. 模块
#####

# 用 import 导入模块

import math

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


# 也可以从模块中导入个别值

from math import ceil, floor

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


# 可以导入一个模块中所有值

# 警告：不建议这么做

from math import *


# 如此缩写模块名字

import math as m

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


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

# 模块的名字就是文件的名字。

# 你可以这样列出一个模块里所有的值

import math

dir(math)
```

## 2.7 高级用法

```
#####
## 7. 高级用法
#####

# 用生成器(generators)方便地写惰性运算
def double_numbers(iterable):
    for i in iterable:
        yield i + i

# 生成器只有在需要时才计算下一个值。它们每一次循环只生成一个值，而不是把所有的
# 值全部算好。
#
# range 的返回值也是一个生成器，不然一个 1 到 900000000 的列表会花很多时间和内
# 存。
#
# 如果你想用一个 Python 的关键字当作变量名，可以加一个下划线来区分。
range_ = range(1, 900000000)
# 当找到一个 >=30 的结果就会停
# 这意味着 `double_numbers` 不会生成大于 30 的数。
for i in double_numbers(range_):
    print(i)
    if i >= 30:
        break

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

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

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

    return wrapper

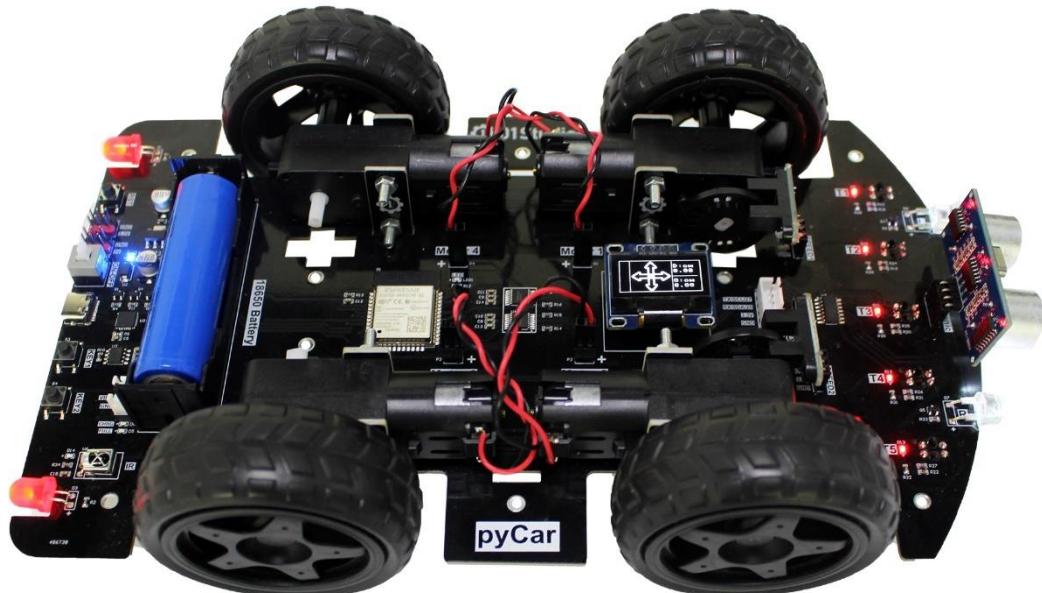
@beg
def say(say_please=False):
    msg = "Can you buy me a beer?"
    return msg, say_please

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

## 第3章 开发环境快速建立

pyCar 是基于上海乐鑫 ESP32 平台，ESP32 芯片的资源非常丰富，除了拥有更多的引脚外设外，还同时支持 WIFI、蓝牙功能。

如果你已经学习过 MicroPython 平，那么恭喜你，本部分内容的学习将会变得轻松，因为它们的开发环境和编程方式都非常相似。



pyCar 开发套件

### 3.1 基于 Windows

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

#### 3.1.1 安装开发软件 Thonny

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

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

该软件可以在 零一科技（01Studio）MicroPython 开发套件配套资料\01-开发工具\01-Windows\MicroPython 开发软件(IDE)\Thonny 下获取。默认提供 Windows 的安装包。当然你也可以在在 <https://thonny.org/> 下载最新版，选择自己的开发平台进行下载安装即可(这里选择 Windows！)：



图 3-1 Thonny Python IDE 下载

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

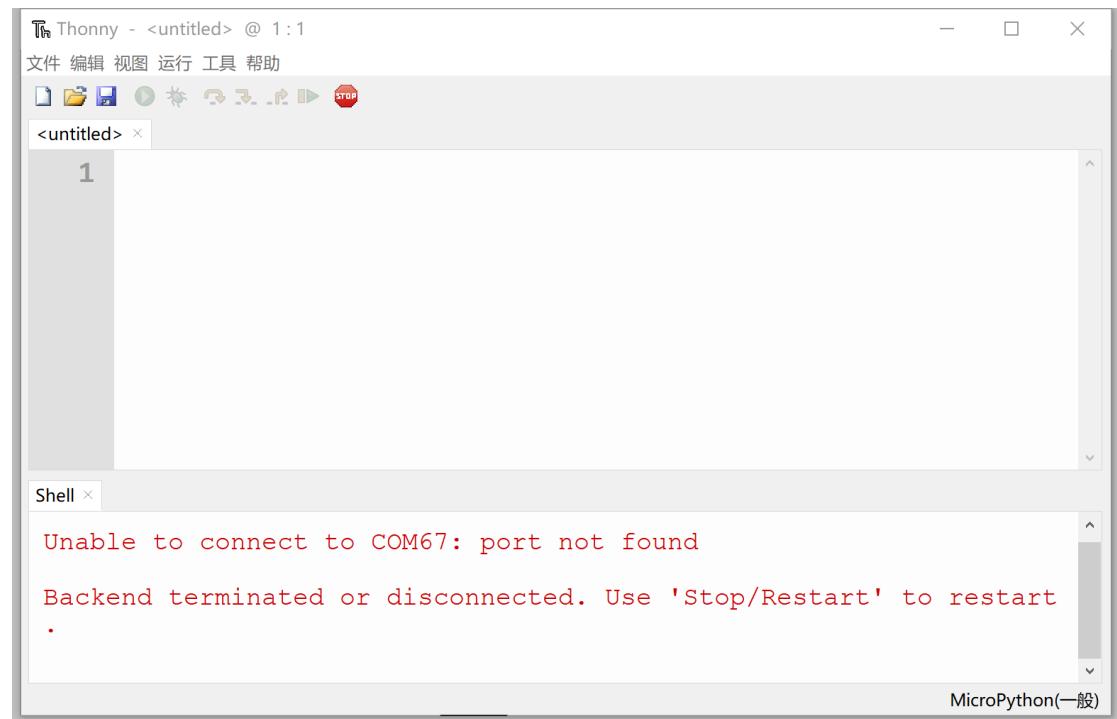


图 3-2 安装完成

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

### 3.1.2 开发套件使用

#### 3.1.2.1 驱动安装

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

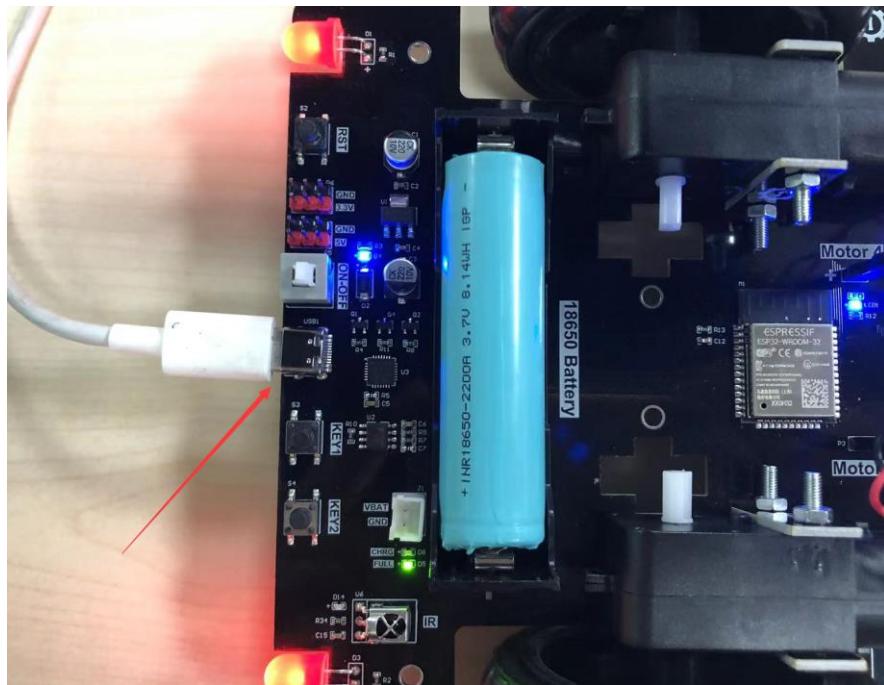


图 3-3 通过 MicroUSB 线连接到电脑

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

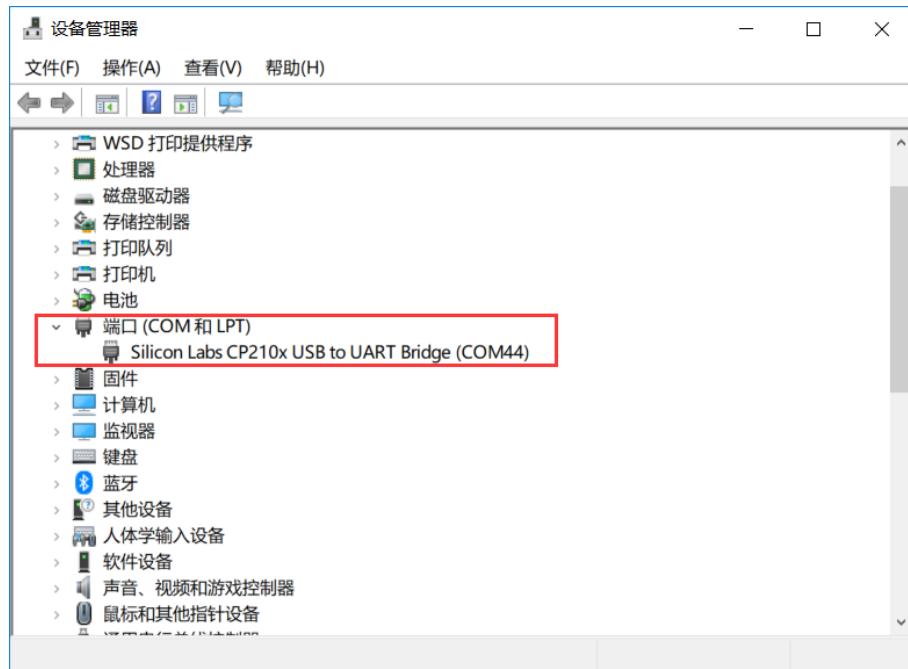


图 3-4 串口驱动成功安装

如果无法安装，请手动安装驱动，方法如下：

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

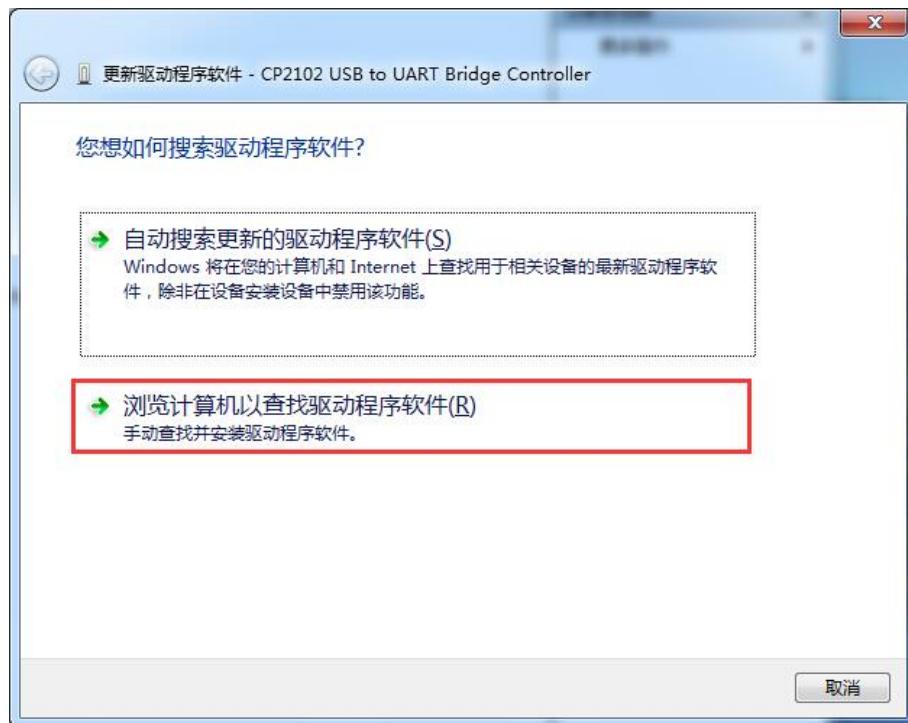


图 3-5

驱动路径选择：零一科技（01Studio）MicroPython 开发套件配套资料\01-开发工具\01-Windows\串口终端工具\CP210x 驱动，**注意查看文件夹里面的文件如果没解压要先解压**。点击确认后即可自动安装：

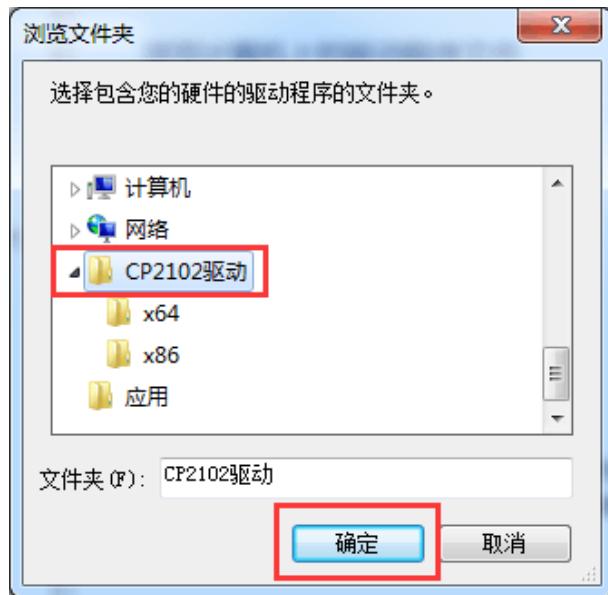


图 3-6

安装成功后如下图：

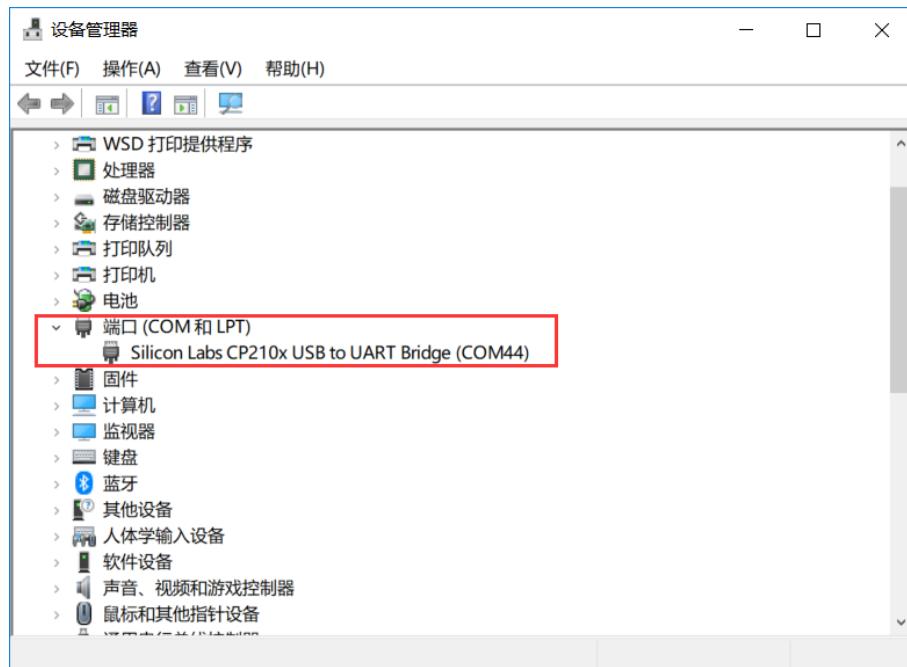


图 3-7 串口驱动安装成功

### 3.1.2.2 REPL 串口交互调试

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

我们打开 Thonny，将开发板连接到电脑。点击右下角：

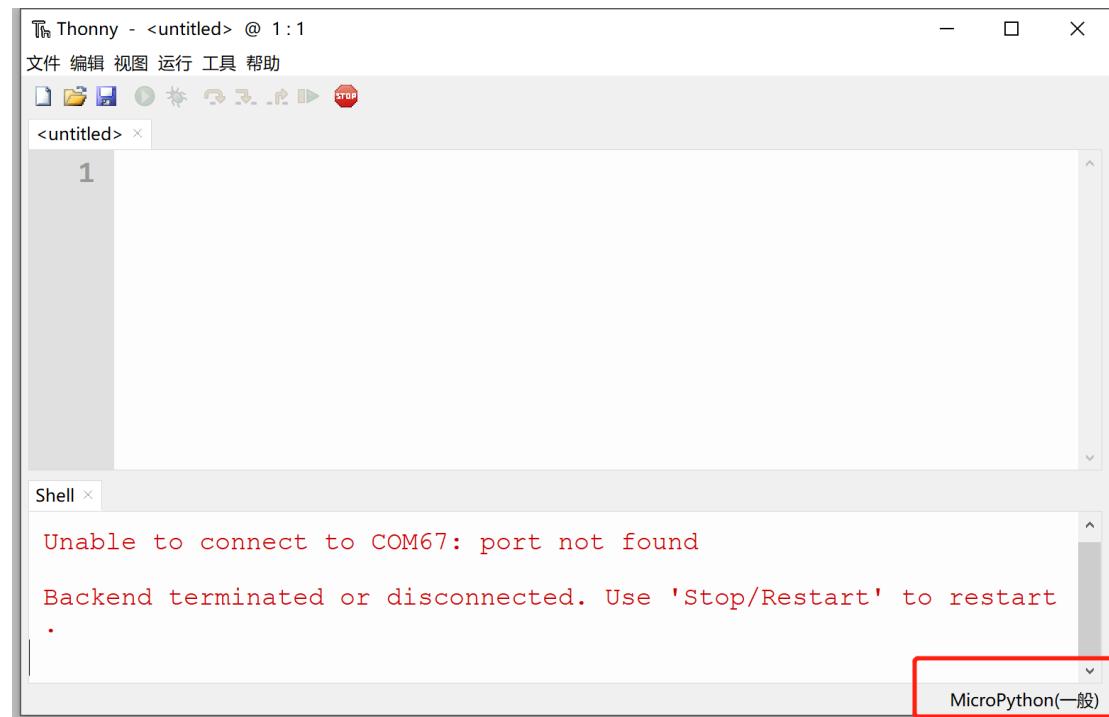


图 3-8 选择要连接的设备

在弹出的列表选择：Configure interpreter

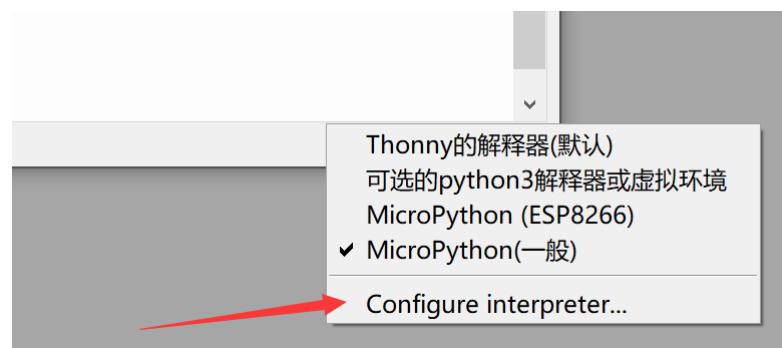


图 3-9

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

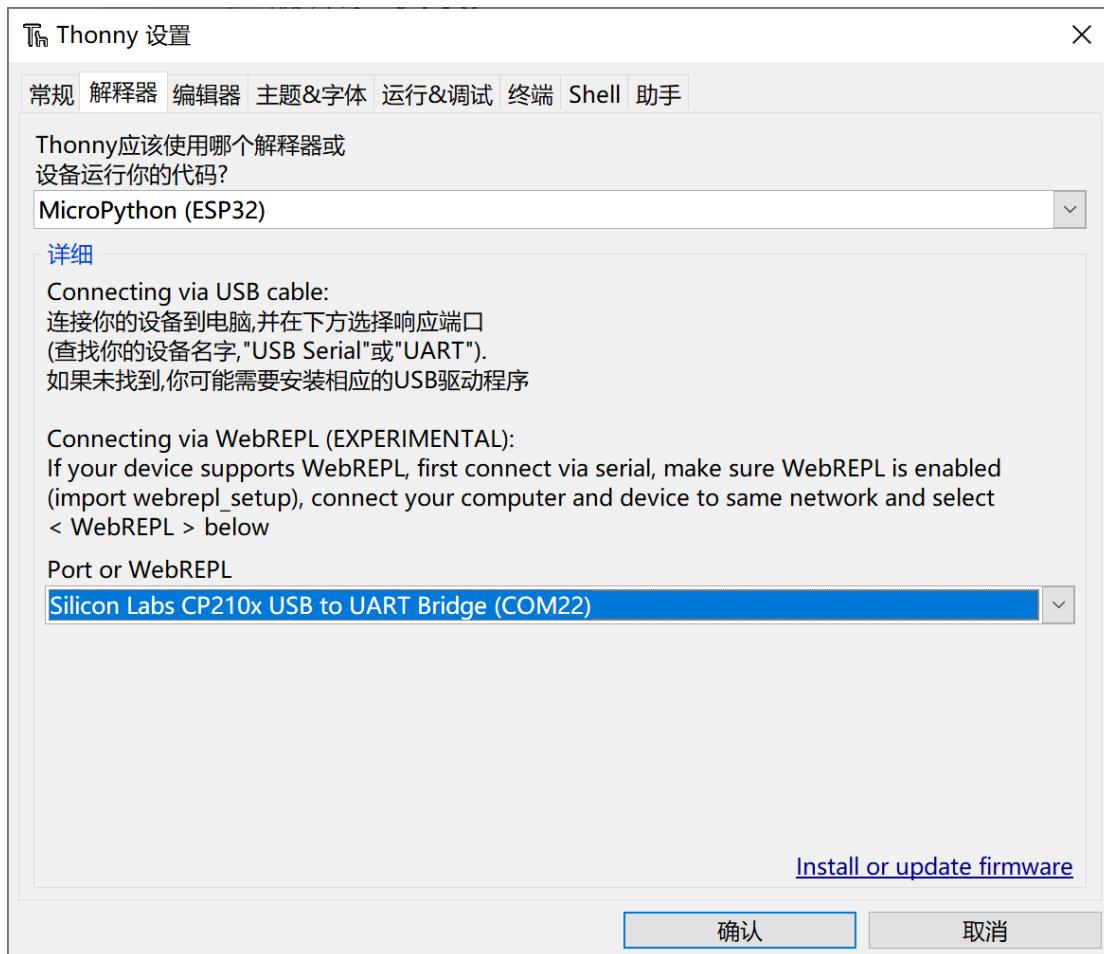


图 3-10 选择开发板类型

连接成功后可以在 shell（串口终端）看到固件的相关信息：

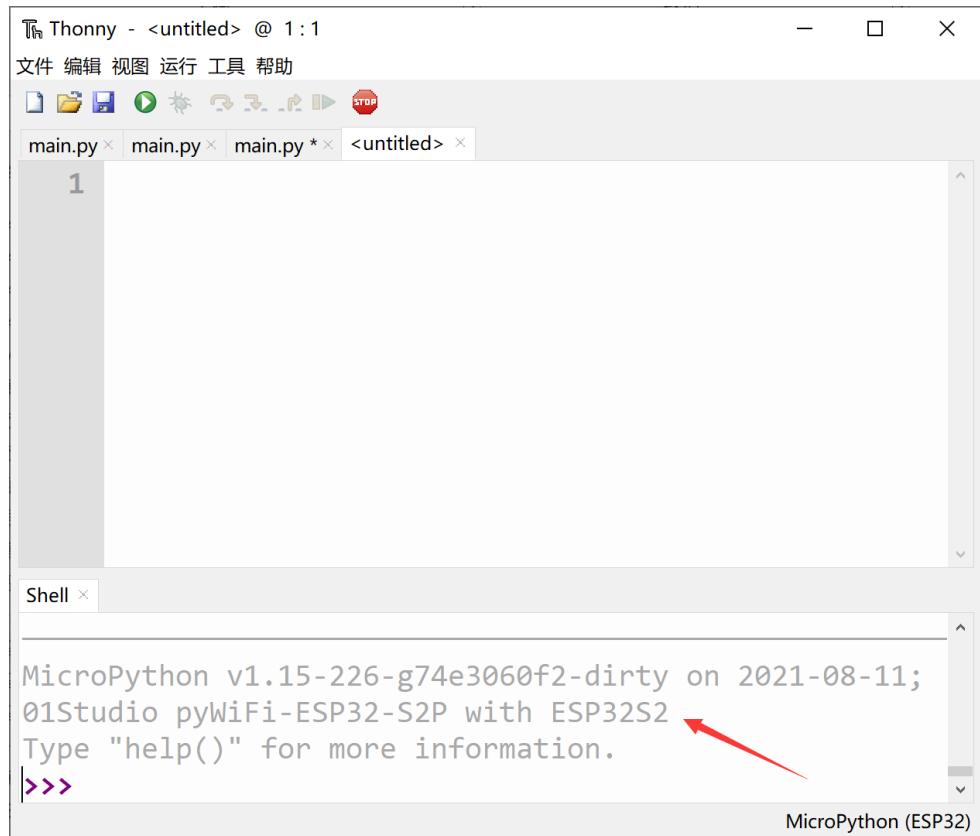


图 3-11 连接成功

我们在 Shell 里面输入 `print("Hello 01Studio!")`，按回车，可以看到打印出 Hello 01Studio 字符：



图 3-12

再输入 `1+1`，按回车：



图 3-13

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

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

Shell < Type "help()" for more information.  
>>> from machine import Pin  
>>> led=Pin(2,Pin.OUT)  
>>> led.value(1)  
>>>

图 3-14 逐行输入

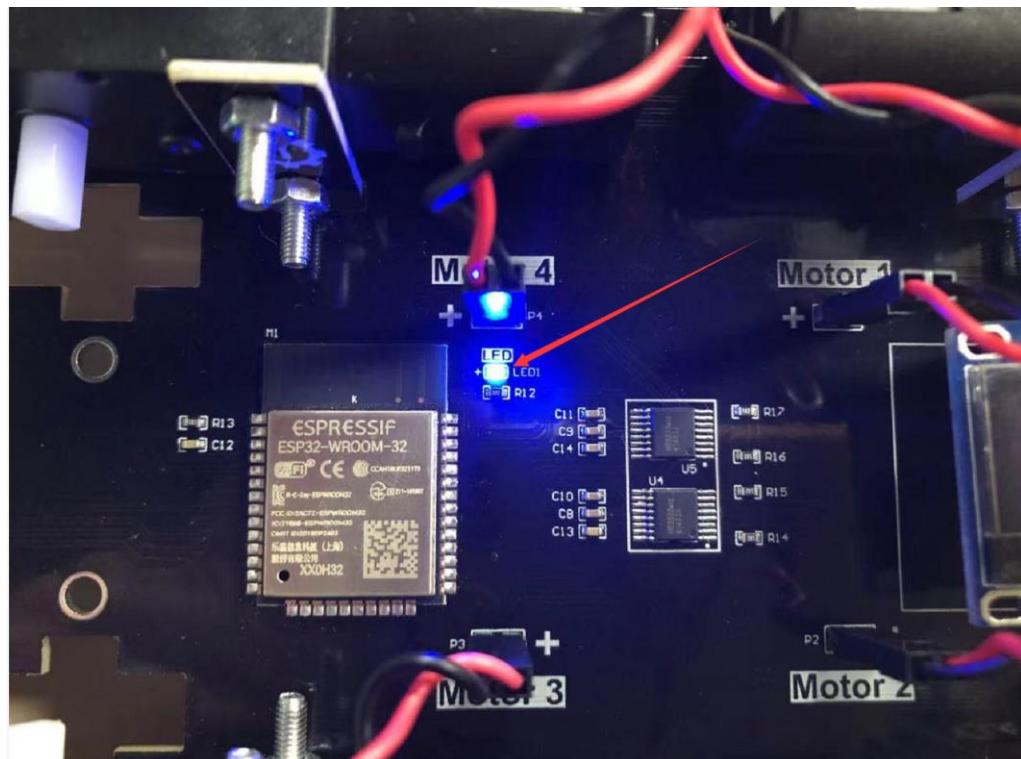


图 3-15 LED 被点亮

REPL 还有一个强大的功能就是打印错误的代码来调试程序，在后面代码运行时候，如果程序出错，出错信息将通过 REPL 打印。

The screenshot shows a software interface for MicroPython development. At the top is a menu bar with '文件' (File), '编辑' (Edit), '视图' (View), '运行' (Run), '工具' (Tools), and '帮助' (Help). Below the menu is a toolbar with icons for file operations like Open, Save, and Run. The main area has two tabs: 'main.py' and 'Shell'. The 'main.py' tab contains Python code for a project, including comments and imports. The 'Shell' tab shows the REPL interaction where the user runs the code and receives an error message about a missing attribute 'valu'.

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

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

图 3-16 错误打印

### REPL 终端常用键盘按键:

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

Ctrl + D : 软件复位开发板。

### 3.1.2.3 文件系统

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

点击 视图--文件：

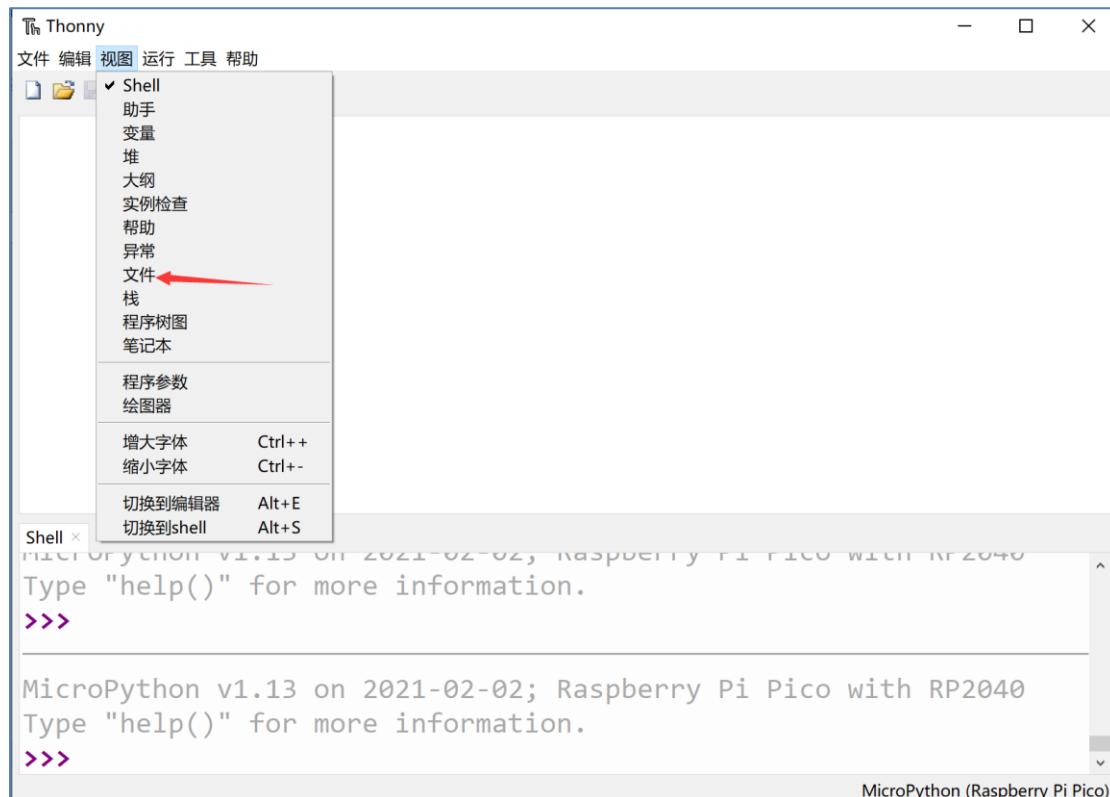


图 3-17

可以看到左边出现本地和开发板的实时文件浏览窗口：



图 3-18

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

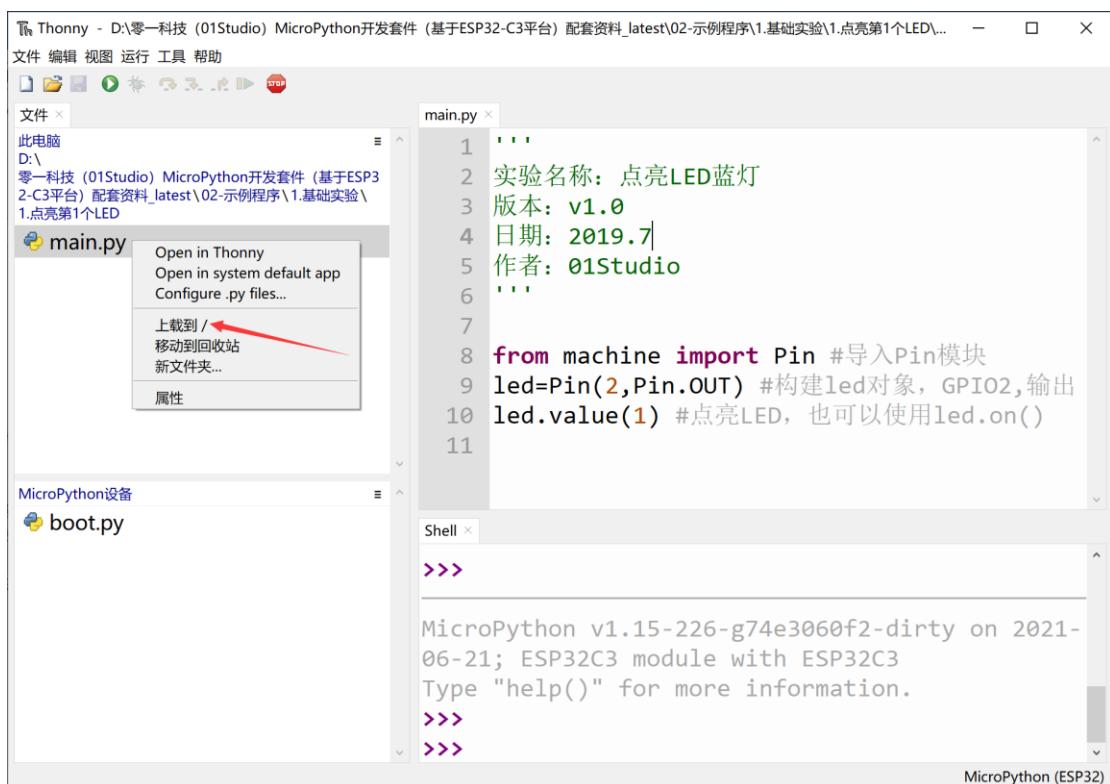


图 3-19 发送文件

### 3.1.2.4 代码测试

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

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

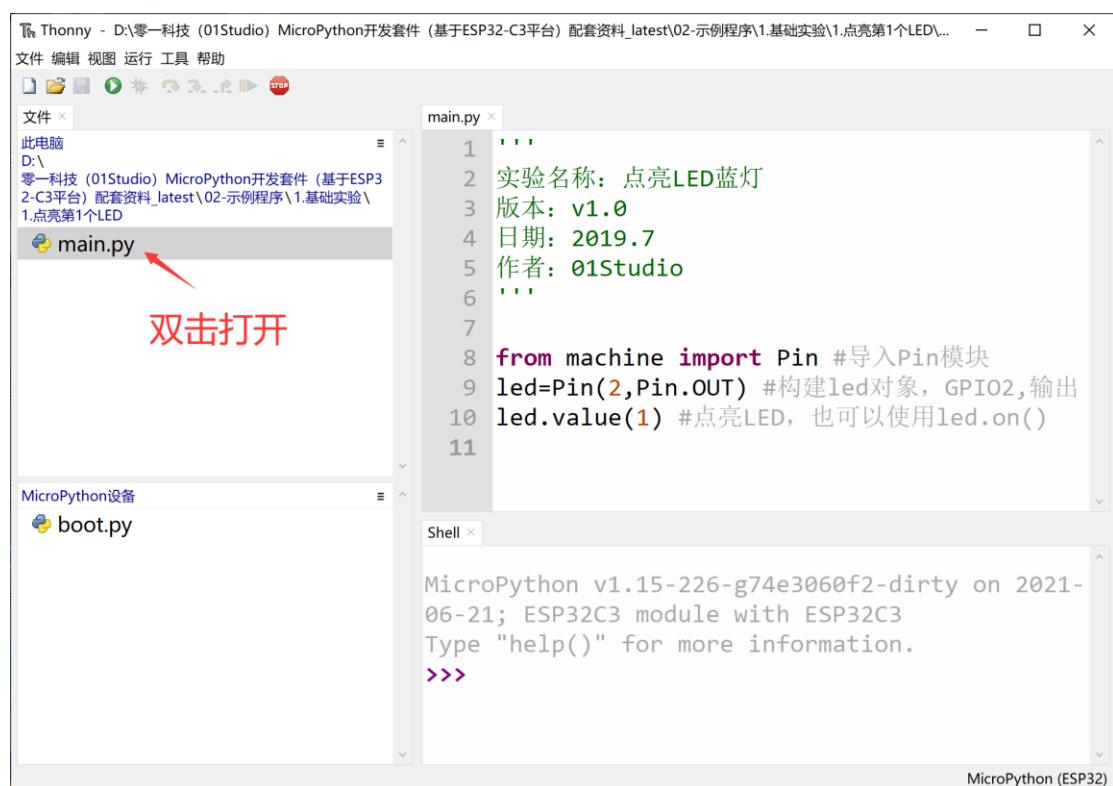


图 3-20

点击 运行—运行当前脚本 或者直接点绿色按钮：



图 3-21 运行例程

这时候可以看到开发板上的蓝灯被点亮：

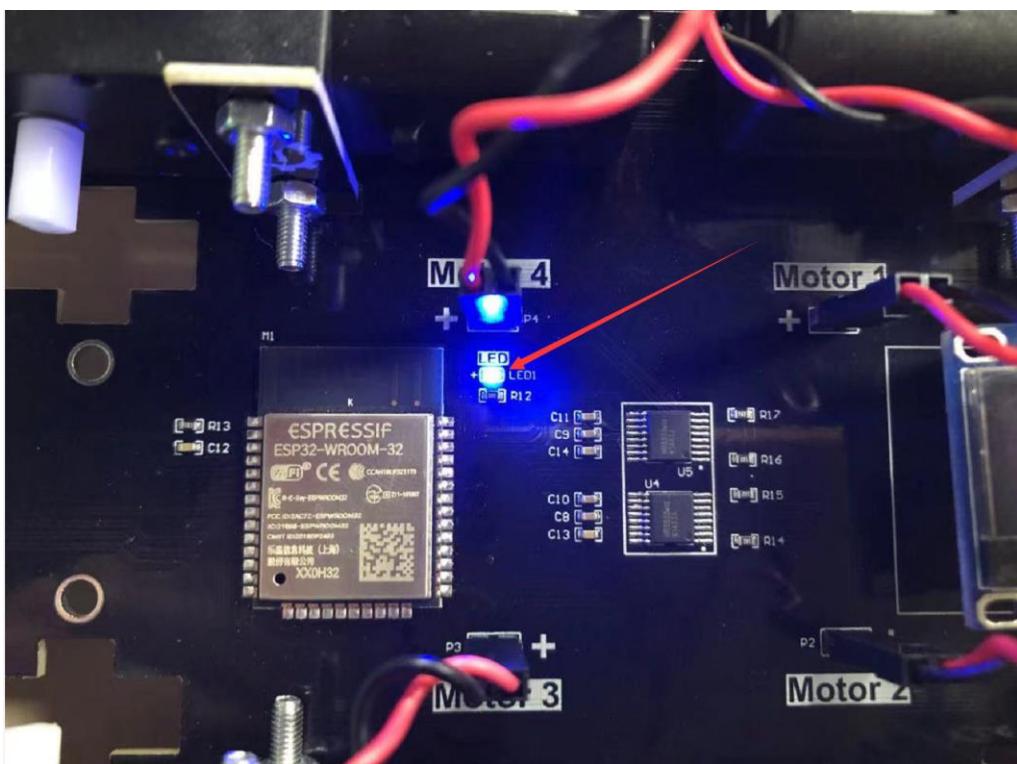


图 3-22 蓝灯被点亮

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

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

**boot.py:** 一般用于配置初始化参数；

**main.py:** 主程序

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

我们将 LED 例程的 `main.py` 发送到开发板

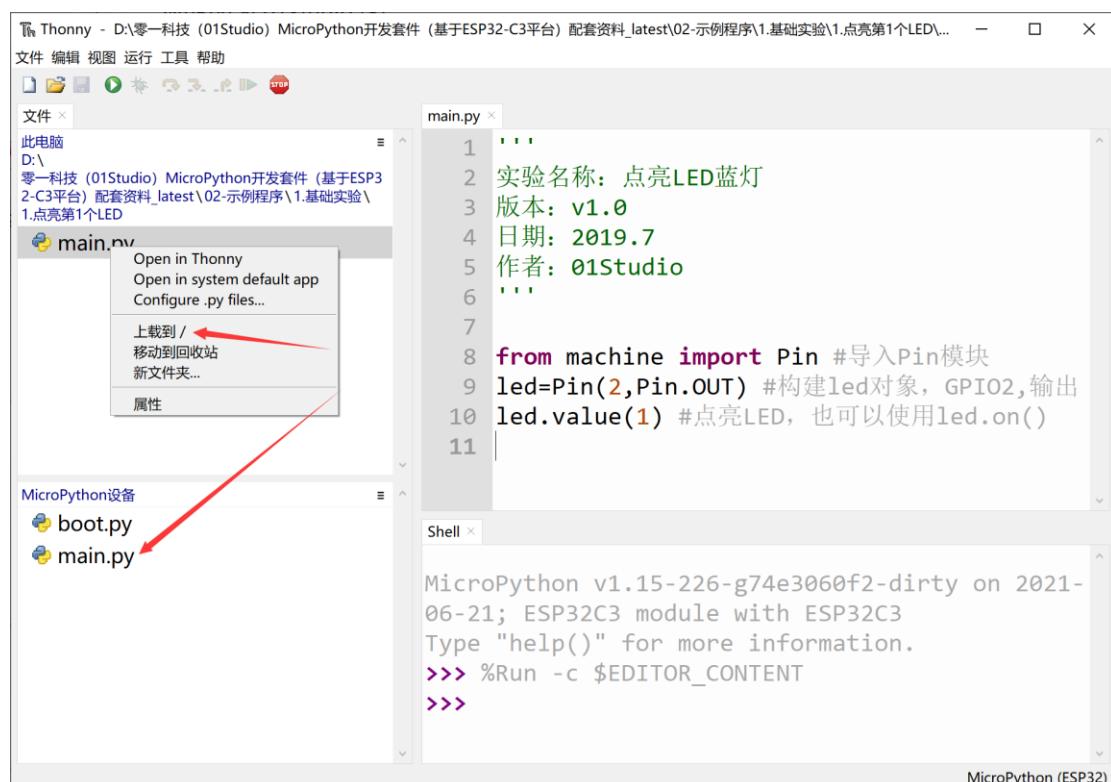


图 3-23 将 LED 例程的 `main.py` 发送到开发板

按下开发板的复位键，可以看到 LED 蓝灯被点亮：

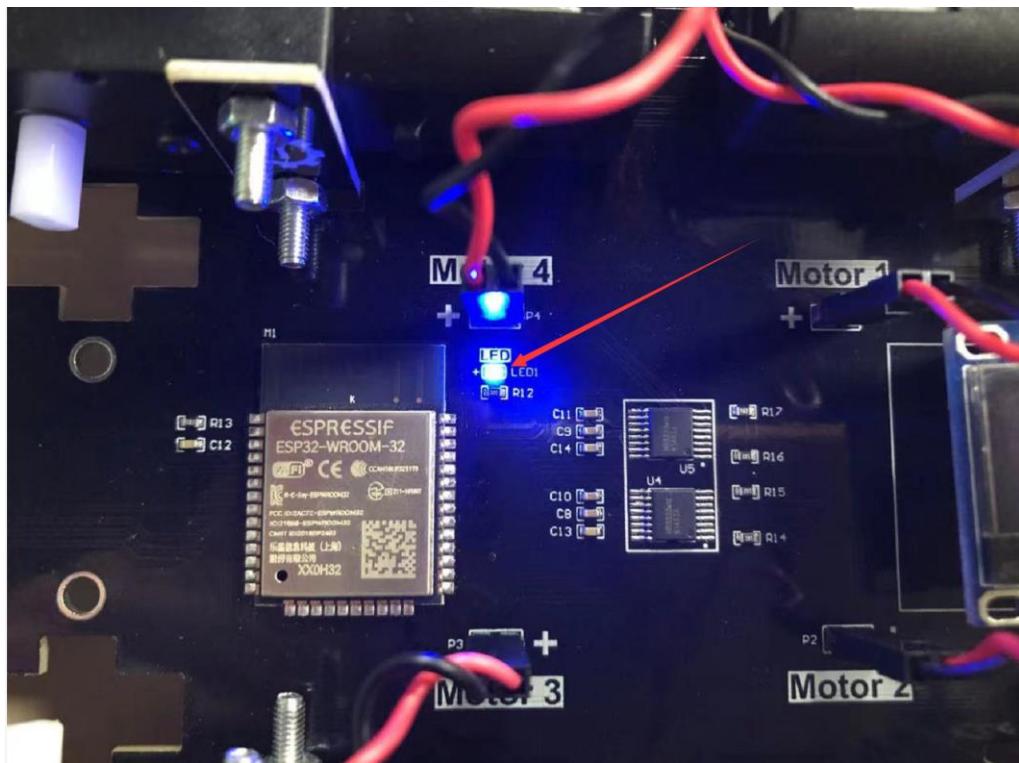


图 3-24 代码离线运行

### 3.1.2.5 固件更新

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

找到路径：[零一科技（01Studio）MicroPython 开发套件配套资料\01-开发工具\01-Windows\固件更新工具\flash\\_download\\_tools\\_v3.8.8](#) 下的 flash\_download\_tools\_v3.8.8.exe 软件，双击打开。

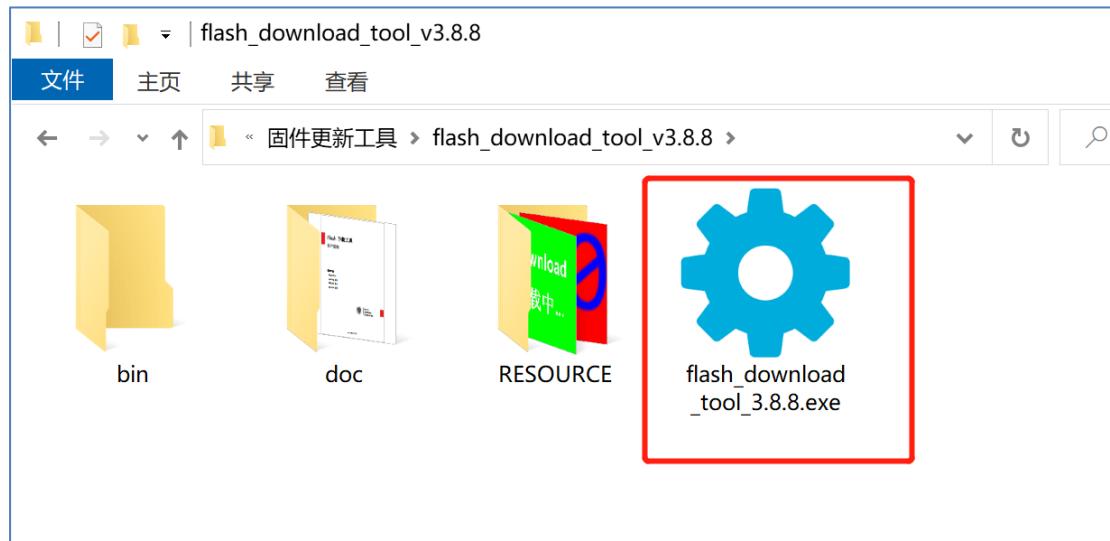


图 3-25

芯片这里选择 ESP32，develop 开发者模式，然后点击 OK：

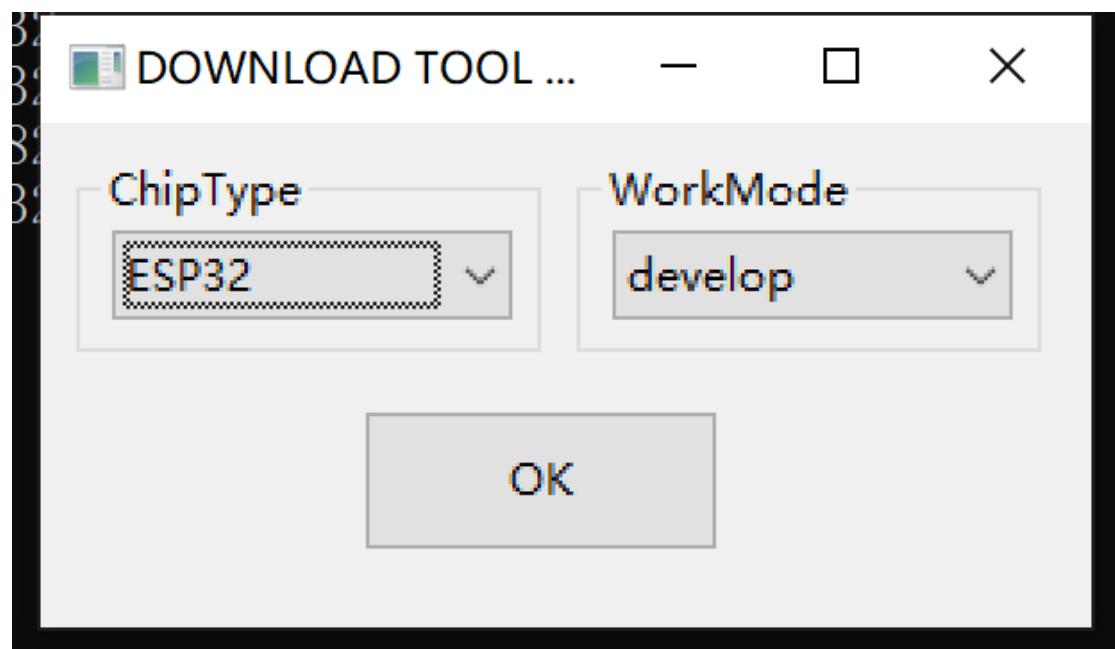


图 3-26 选择 ESP32

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

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

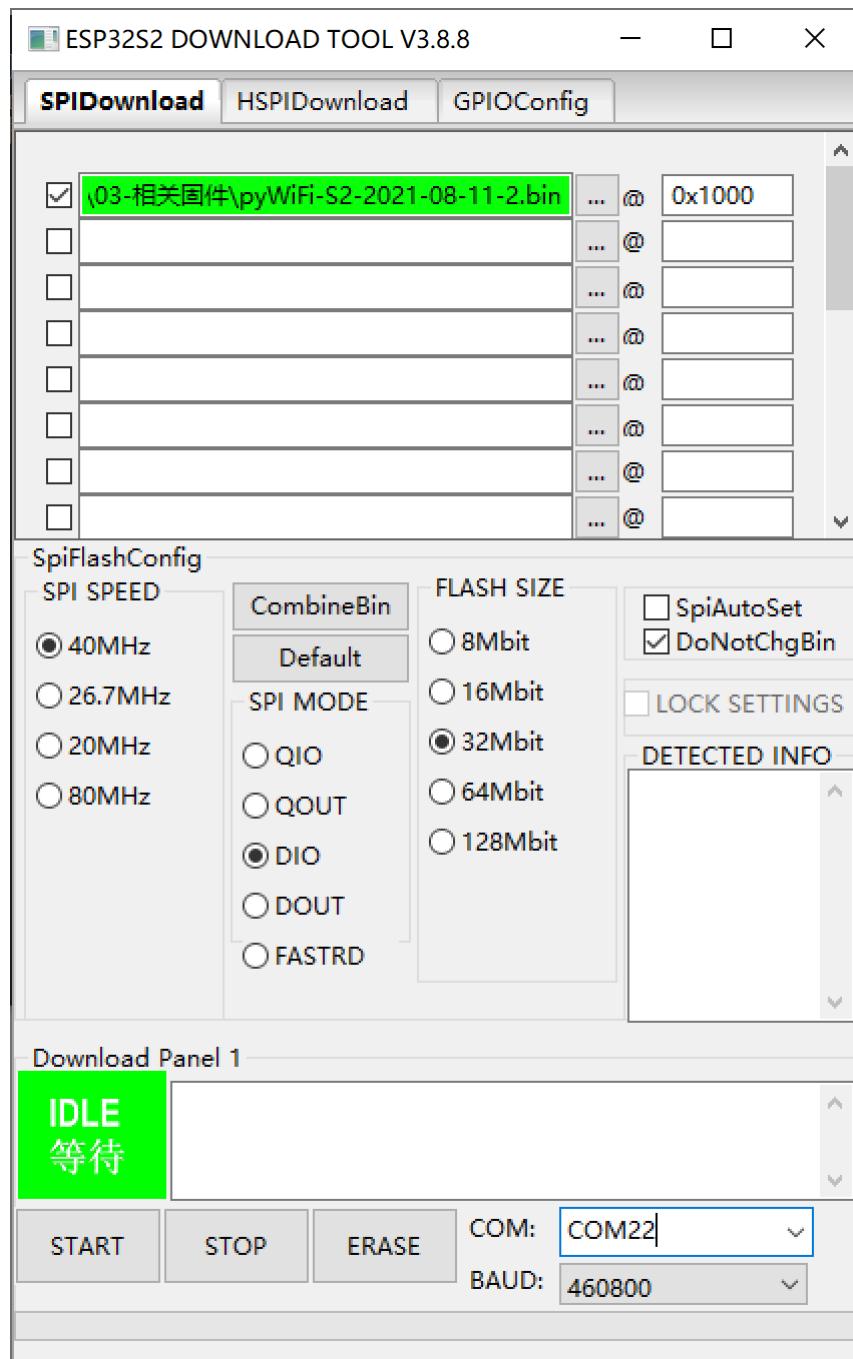


图 3-27 配置

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

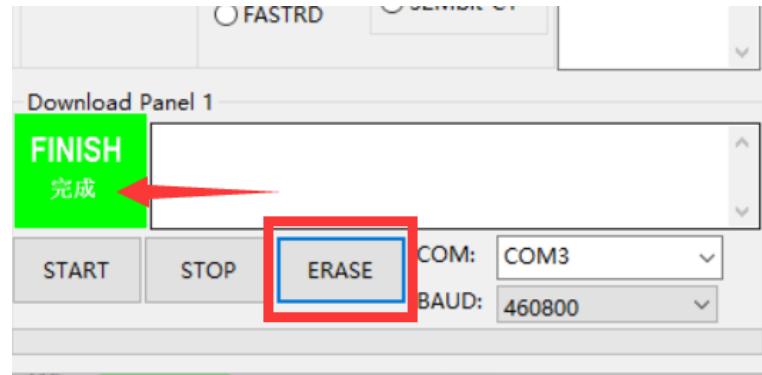


图 3-28 刷除 flash 成功

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

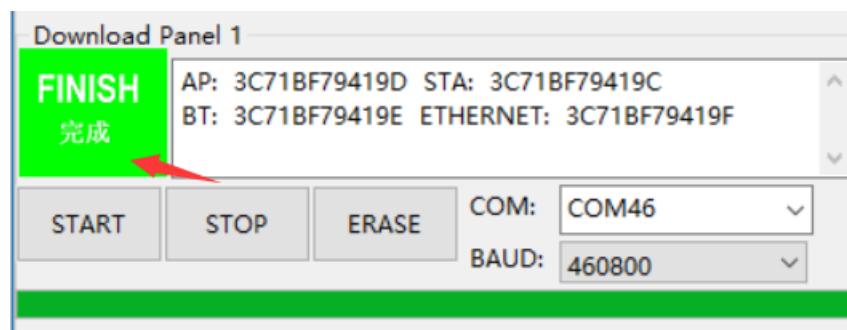


图 3-29 烧录成功

## 3.2 基于 Mac OS

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

### 3.2.1 安装开发软件 Thonny

在 <https://thonny.org/> 选择 Mac 版本下载：

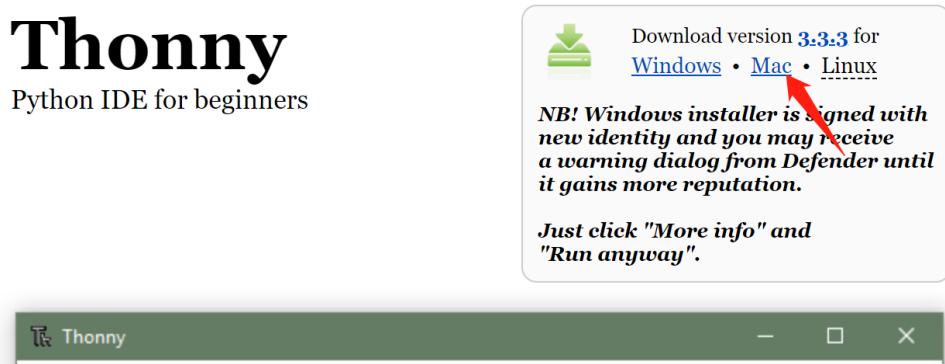


图 3-30

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

### 3.2.2 开发套件使用

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

### 3.2.2.1 固件更新

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

打开终端，输入以下命令安装 esptool：

```
pip install esptool
```

安装完成后即可使用 esptool 工具来烧写固件，步骤如下：

#### 第一步：刷除 Flash：

在终端输入下面命令按回车。

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

刷除成功后出现以下信息

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

图 3-31 刷除成功提示

#### 第二步：烧写固件：

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

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

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

烧录成功如下图：

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

图 3-32 固件烧录成功

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

### 3.3 基于 Linux (树莓派)

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

#### 3.3.1 安装开发软件 Thonny

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

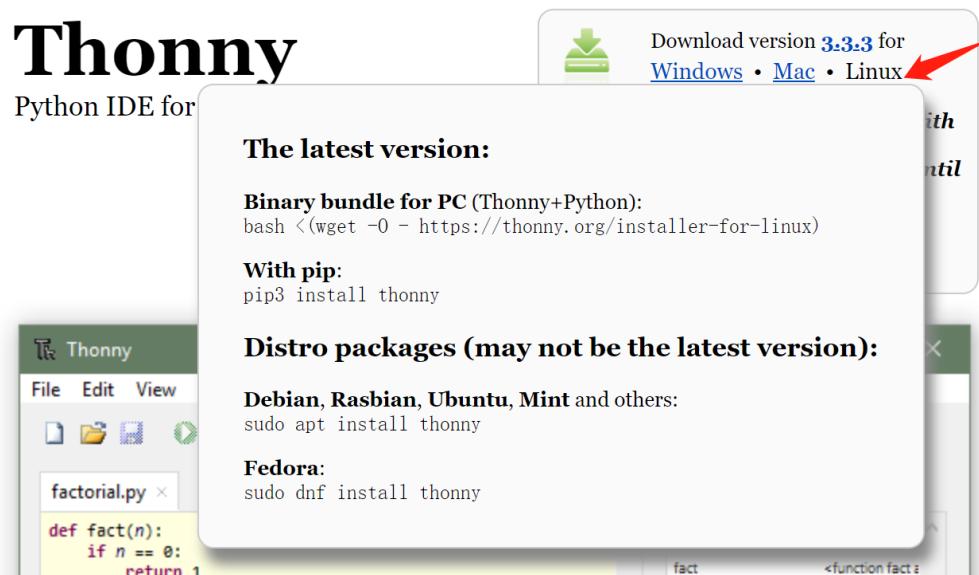


图 3-33

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

#### 3.3.2 开发套件使用

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

## 第4章 基础实验

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

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

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

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

## 4.1 pyCar 引脚简表

pyCar 使用 ESP32 做主控，电路连接方式可以参考原理图，这里为了更方便快速查看，制作了一个简表。其中 26 和 27 引脚复用扩展接口。GPIO26—RX—SCL, GPIO27—TX—SDA。

ESP32-WROOM-32		
GPIO	功能1	功能1说明
0	KEY	KEY1/BOOT
1	REPL	TX0
2	LED	LED1
3	REPL	RX0
4	测速	编码盘1
5	车头灯	控制
12	KEY	KEY2
13	测速	编码盘2
14	电机	电机1+
15	电机	电机1-
16	电机	电机2+
17	电机	电机2-
18	电机	电机3+
19	电机	电机3-
21	电机	电机4+
22	电机	电机4-
23	OLED	SCL
25	OLED	SDA
26	超声波	ECHO
27	超声波	TRIG
32	红外	红外接收头
33	巡线	光电传感器1
34	巡线	光电传感器2
35	巡线	光电传感器3
36	巡线	光电传感器4
37	停用	没引出
38	停用	没引出
39	巡线	光电传感器5

图 4-1

## 4.2 点亮第一个 LED

- **前言:**

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

- **实验平台:**

pyCar。



图 4-2 pyCar

- **实验目的:**

点亮 LED（蓝灯）。

- **实验讲解:**

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

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

表 4-1 Pin 对象

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

**方式 1 是：** import machine，然后通过 machine.Pin 来操作；

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

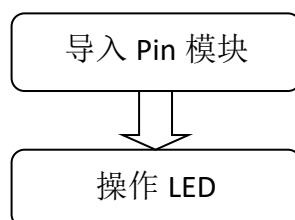


图 4-3 代码编写流程

LED 跟模块引脚 2 相连，通过输出高电平方式点亮。

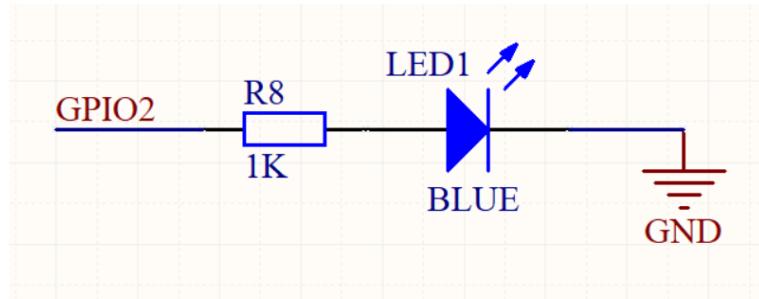


图 4-4 LED 接线图

参考代码如下：

```

...
实验名称: 点亮 LED 蓝灯
版本: v1.0
日期: 2021.11
作者: 01Studio
...
from machine import Pin #导入 Pin 模块
led=Pin(2,Pin.OUT) #构建 led 对象, GPIO2, 输出
led.value(1) #点亮 LED, 也可以使用 led.on()

```

运行程序有两个方法：

#### 方法一：

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

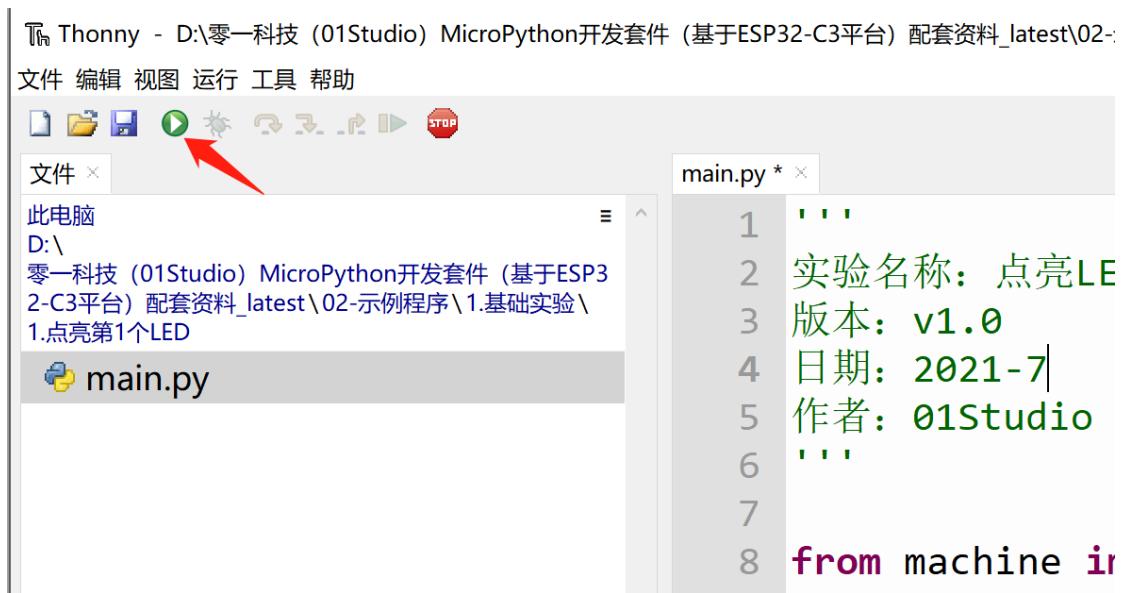


图 4-5 运行仿真

## 方法二：

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

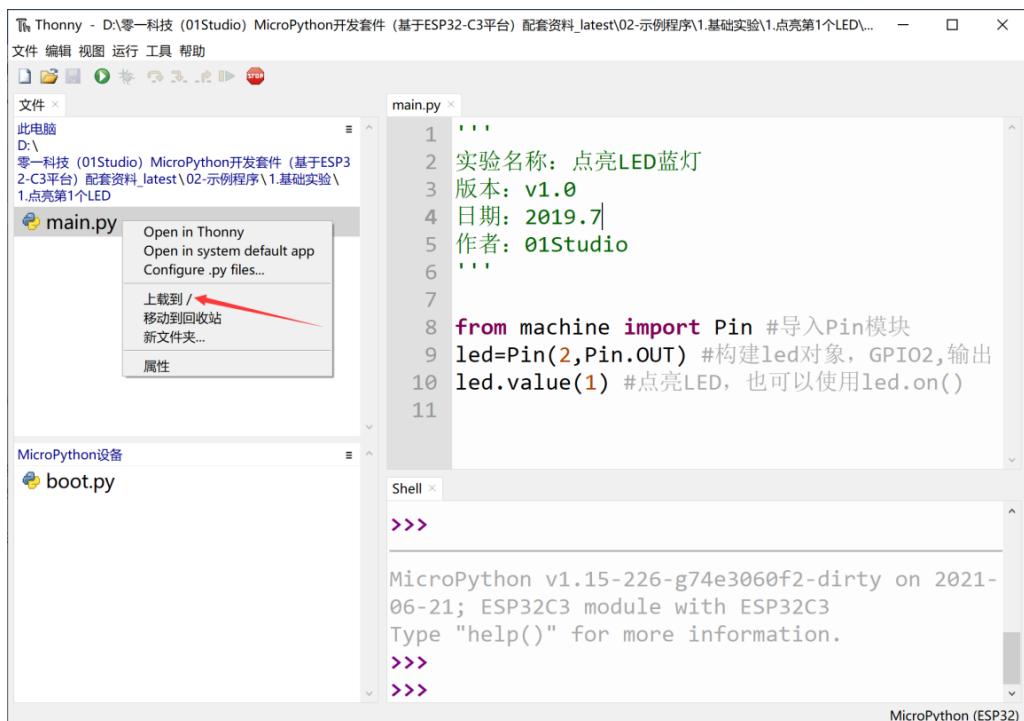


图 4-6 拷贝文件到设备

- **实验结果：**

下载程序，可以看到 LED（蓝灯）成功点亮。

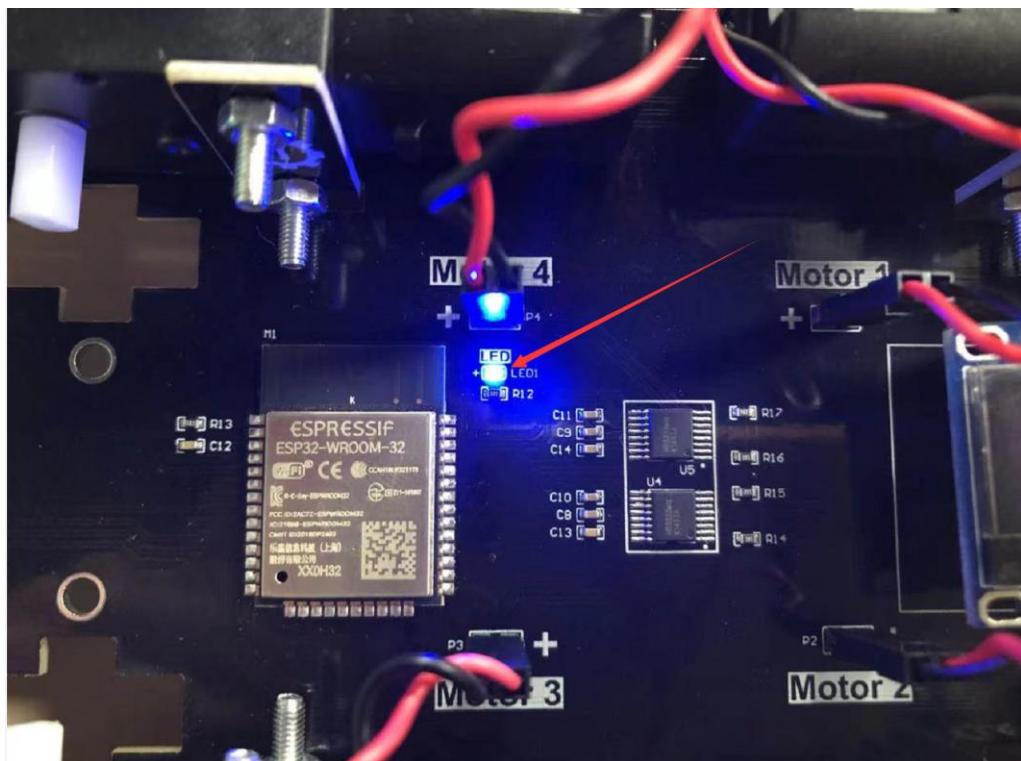


图 4-7

- **总结：**

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

### 4.3 按键

- **前言:**

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

- **实验平台:**

pyCar。

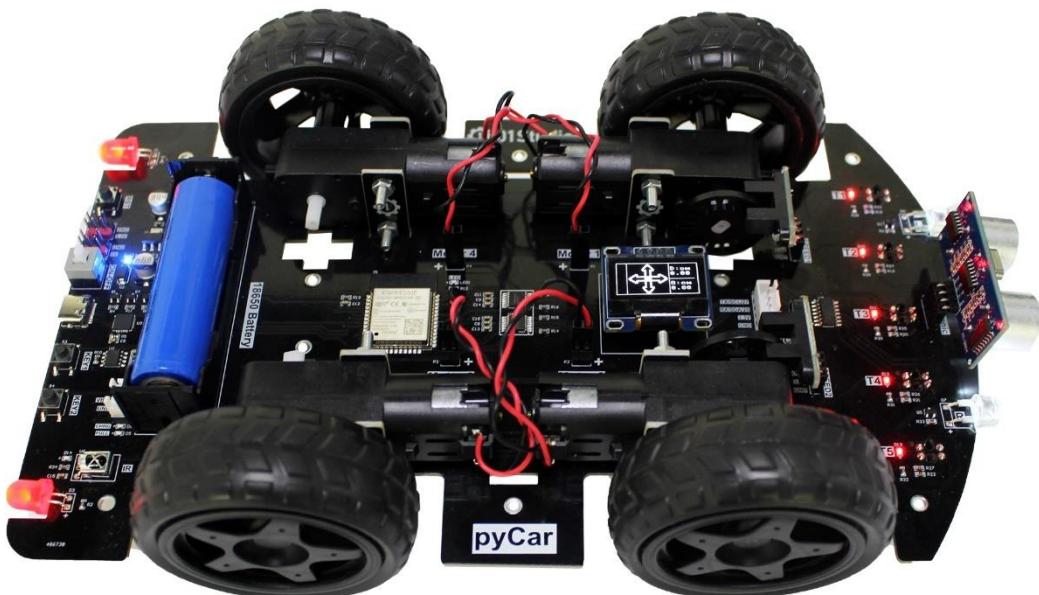


图 4-8 pyCar

- **实验目的:**

使用按键功能，通过检测按键被按下后，改变 LED（蓝灯）的亮灭状态。

- **实验讲解:**

pyCar 开发板上有 3 个按键，RST、KEY1 和 KEY2，RST 顾名思义是复位用的，所以真正自带可以用的就有 2 个功能按键。

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

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

表 4-2 KEY 对象

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

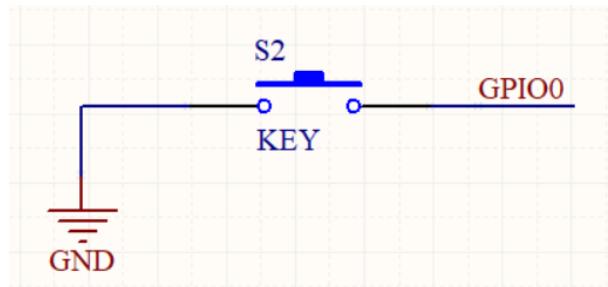


图 4-9 pyCar 按键原理图

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

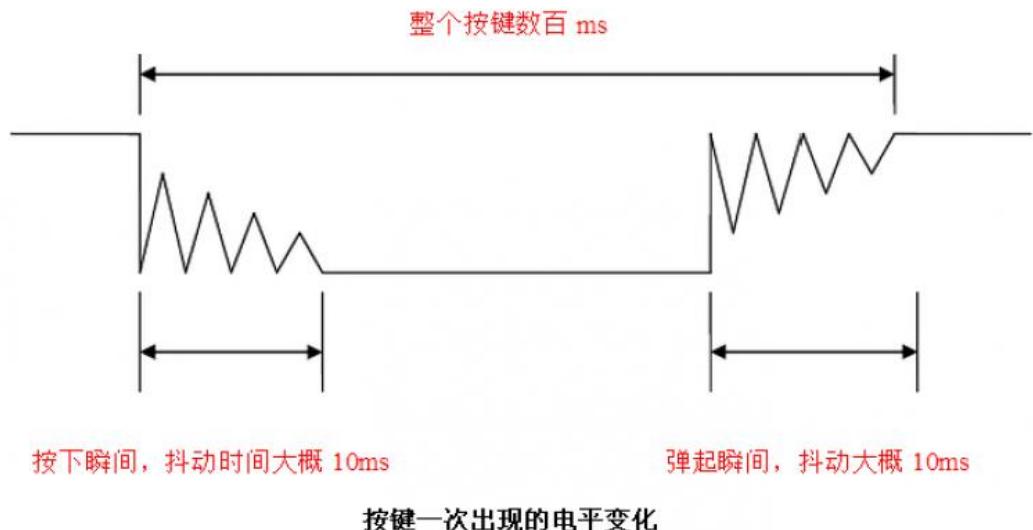


图 4-10 按键按下过程

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

```
import time

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

start = time.ticks_ms() # 获取毫秒计时器开始值

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

编程流程图如下：

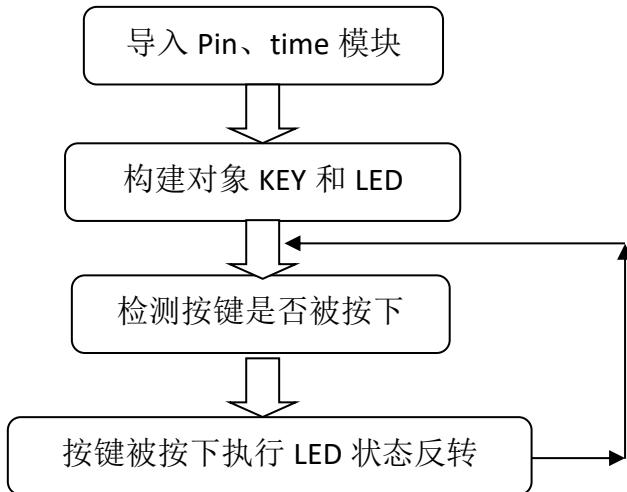


图 4-11 代码编写流程

实验参考代码如下：

```
'''  
实验名称: 按键  
版本: v1.0  
日期: 2021.12  
作者: 01Studio  
说明: 通过按键改变 LED 的亮灭状态  
...  
  
from machine import Pin  
import time  
  
  
LED=Pin(2,Pin.OUT) #构建 LED 对象,开始熄灭  
KEY=Pin(0,Pin.IN,Pin.PULL_UP) #构建 KEY 对象  
state=0 #LED 引脚状态  
  
  
while True:  
    if KEY.value()==0:    #按键被按下  
        time.sleep_ms(10) #消除抖动  
        if KEY.value()==0: #确认按键被按下  
            state=not state #使用 not 语句而非~语句  
            LED.value(state) #LED 状态翻转  
            while not KEY.value(): #检测按键是否松开  
                pass
```

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

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

- **实验结果：**

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

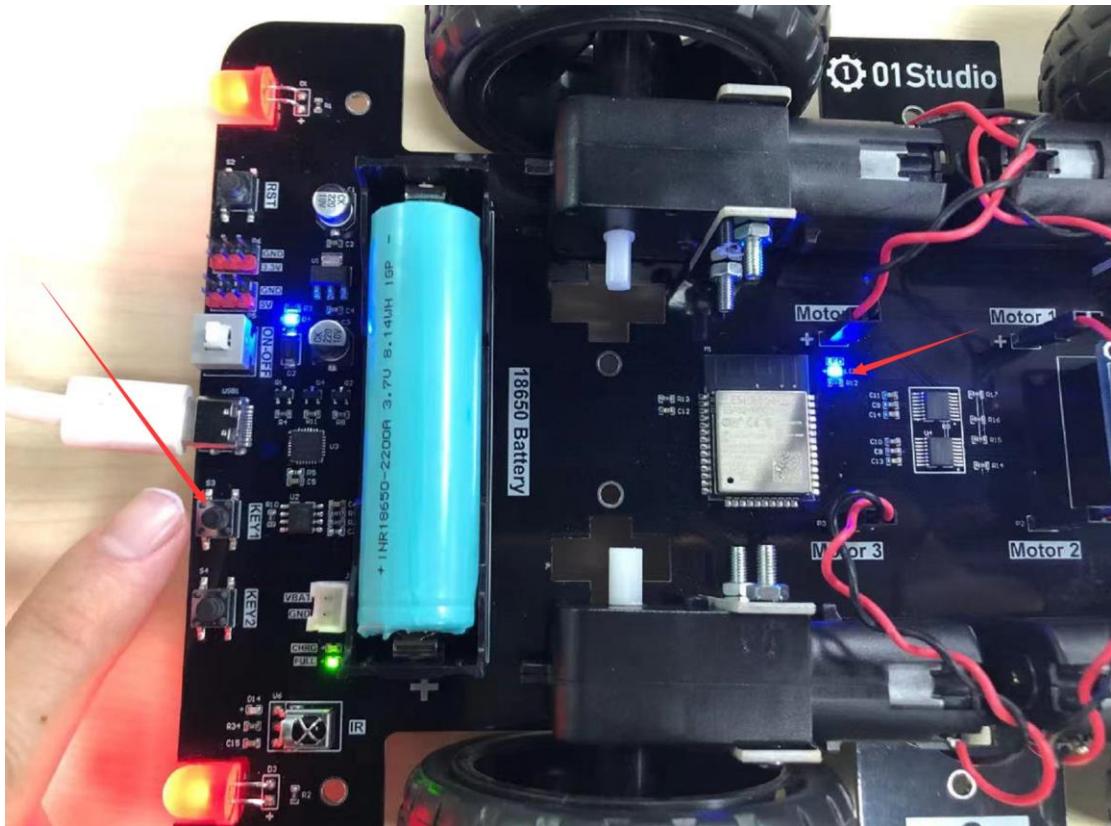


图 4-12

- **总结：**

按键作为我们学习的第一个输入设备，有了输入设备我们就可以跟硬件做人机交互了，这对后面的学习非常有意义。可以看到按键在 MicroPython 下开发也很简单。

## 4.4 外部中断

- **前言：**

前面我们在做普通的 GPIO 时候，虽然能实现 IO 口输入输出功能，但代码是一直在检测 IO 输入口的变化，因此效率不高，特别是在一些特定的场合，比如某个按键，可能 1 天才按下一次去执行相关功能，这样我们就浪费大量时间来实时检测按键的情况。

为了解决这样的问题，我们引入外部中断概念，顾名思义，就是当按键被按下(产生中断)时，我们才去执行相关功能。这大大节省了 CPU 的资源，因此中断的在实际项目的应用非常普遍。

- **实验平台：**

pyCar。



图 4-13 pyCar

- **实验目的：**

利用中断方式来检查按键 KEY 状态，被按键被按下(产生外部中断)后使 LED 的亮灭状态翻转。

### ● 实验讲解：

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

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

表 4-3 按键对象

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

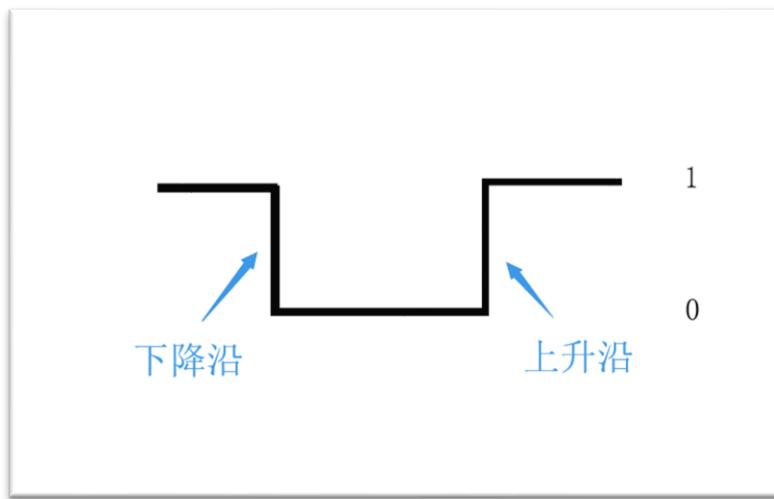


图 4-14 边沿触发

由此可见，我们可以选择下降沿方式触发外部中断，也就是当按键被按下的时候立即产生中断。

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

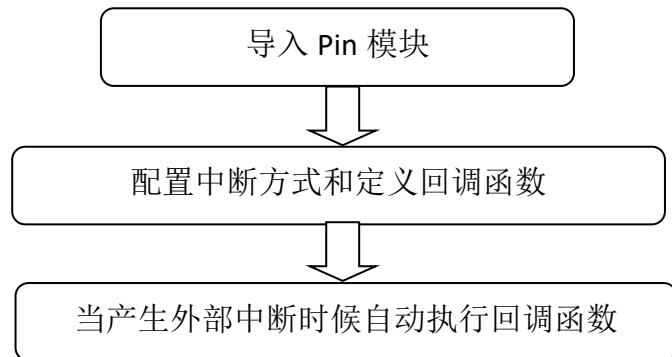


图 4-15 代码编写流程

参考程序代码如下：

```
''''''''''''''''''''''
实验名称：外部中断
版本：v1.0
日期：2021.12
作者：01Studio
说明：通过按键改变 LED 的亮灭状态（外部中断方式）
'''''''''''''''''''''''

from machine import Pin
import time

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

#LED 状态翻转函数
def fun(KEY):
```

```

global state

time.sleep_ms(10) #消除抖动

if KEY.value()==0: #确认按键被按下

    state = not state

    LED.value(state)

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

```

以上代码中需要注意的地方：

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

### ● 实验结果：

将 main.py 文件拷贝到设备，复位。

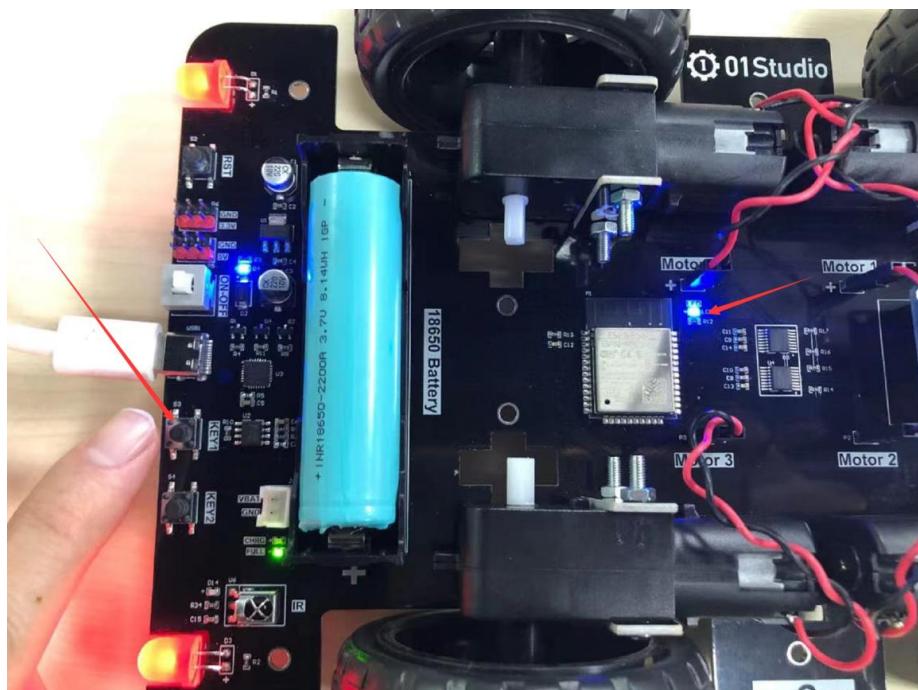


图 4-16 实验现象

- **总结：**

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

## 4.5 定时器

- **前言：**

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

- **实验平台：**

pyCar。



图 4-17 pyCar

- **实验目的：**

通过定时器让 LED 周期性每秒闪烁 1 次。

- **实验讲解：**

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

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

表 4-4 定时器对象

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

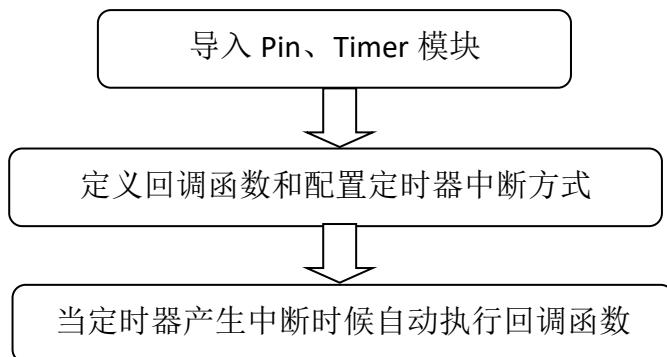


图 4-18 代码编写流程

参考代码如下：

```

...
实验名称: 定时器
版本: v1.0
日期: 2021.12
作者: 01Studio
说明: 通过定时器让 LED 周期性每秒闪烁 1 次
...
  
```

```

from machine import Pin,Timer

led=Pin(2,Pin.OUT)

Counter = 0

Fun_Num = 0

def fun(tim):

    global Counter

    Counter = Counter + 1

    print(Counter)

    led.value(Counter%2)

#开启 RTOS 定时器，编号为-1

tim = Timer(-1)

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

```

● 实验结果：

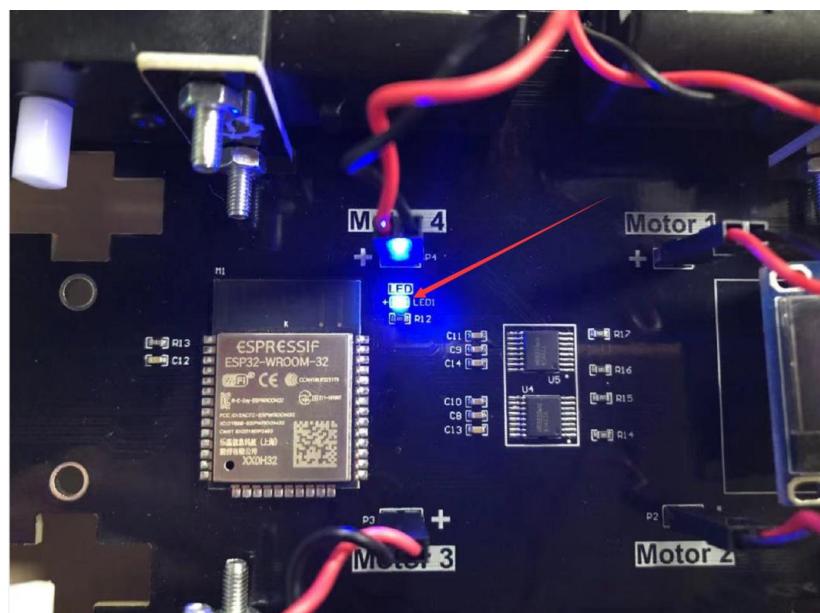


图 4-19 LED 以周期 1 秒的间隔闪烁

- **总结：**

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

## 4.6 I2C 总线（OLED 显示屏）

- **前言：**

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

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

- **实验平台：**

pyCar。

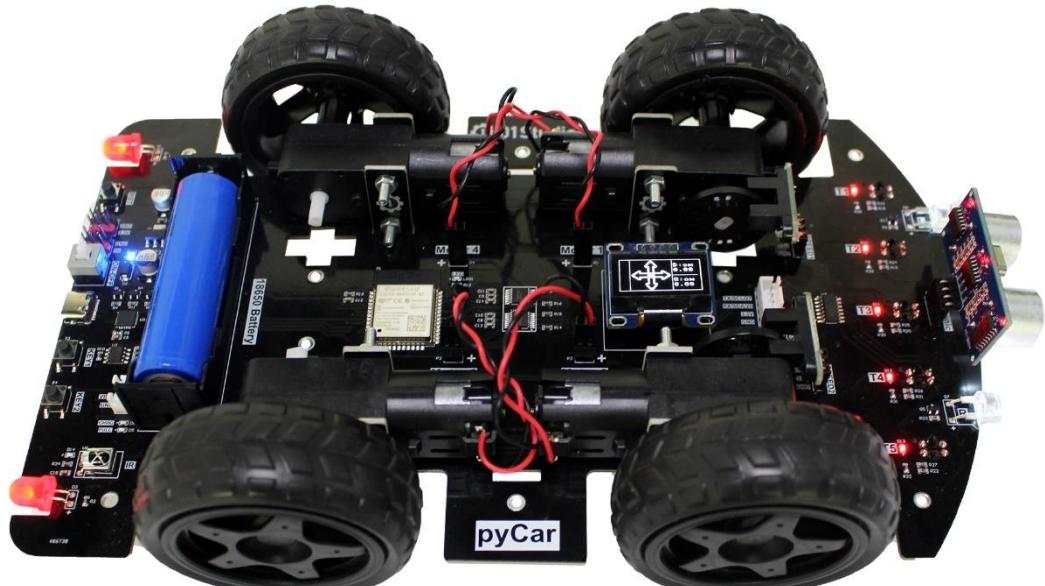


图 4-20 pyCar 开发套件

- **实验目的：**

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

- **实验讲解：**

**什么是 I2C？**

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

信。

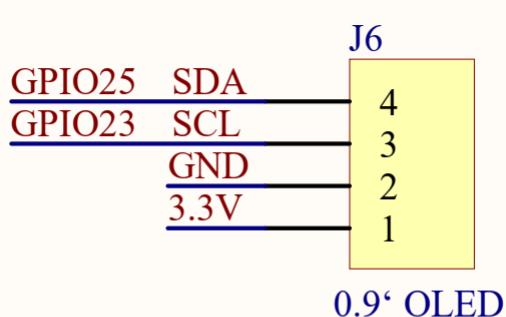
### 什么是 OLED 显示屏？

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



图 4-21 I2C 接口的 OLED

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



0.9' OLED

图 4-22 OLED 接口原理图

从上图可见连接到 OLED 的别是 GPIO23→SCL 和 GPIO25→SDA。本实验将使用 MicroPython 的 Machine 模块来定义 Pin 口和 I2C 初始化。具体如下：

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

表 4-5 I2C 对象

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

构造函数
<code>oled = SSD1306_I2C(width, height, i2c, addr)</code>
构 OLED 显示屏对象。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。
使用方法
<code>oled.text(string,x,y)</code>
将 string 字符写在指定为位置。string: 字符; x:横坐标; y:纵坐标。

<code>oled.show()</code>
执行显示。
<code>oled.fill(RGB)</code>
清屏。RGB: 0 表示黑色, 1 表示白色。

表 4-6 OLED 对象

学习了 I2C、OLED 对象用法后我们通过编程流程图来理顺一下思路：

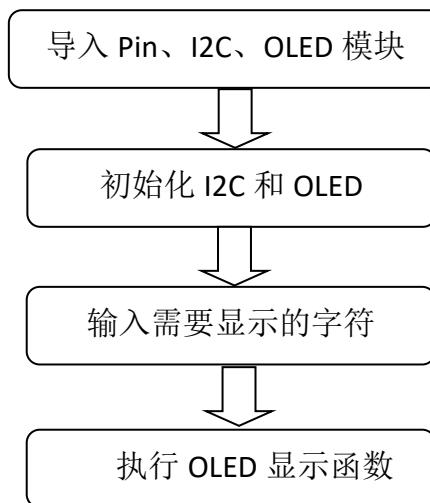


图 4-23 代码编写流程

main.py 文件参考代码如下：

```

...
实验名称: I2C 总线(OLED 显示屏)
版本: v1.0
日期: 2021.12
作者: 01Studio
...

from machine import SoftI2C,Pin      #从 machine 模块导入 SoftI2C、Pin 子模块
from ssd1306 import SSD1306_I2C    #从 ssd1306 模块中导入 SSD1306_I2C 子模块
  
```

```

i2c = SoftI2C(sda=Pin(25), scl=Pin(23))

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

oled.text("Hello World!", 0, 0)      #写入第 1 行内容
oled.text("MicroPython", 0, 20)       #写入第 2 行内容
oled.text("By 01Studio", 0, 50)       #写入第 3 行内容

oled.show()  #OLED 执行显示

```

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

另外记得将我们提供的示例代码中的 ssd1306.py 驱动文件拷贝到 pyCar 的文件系统下。

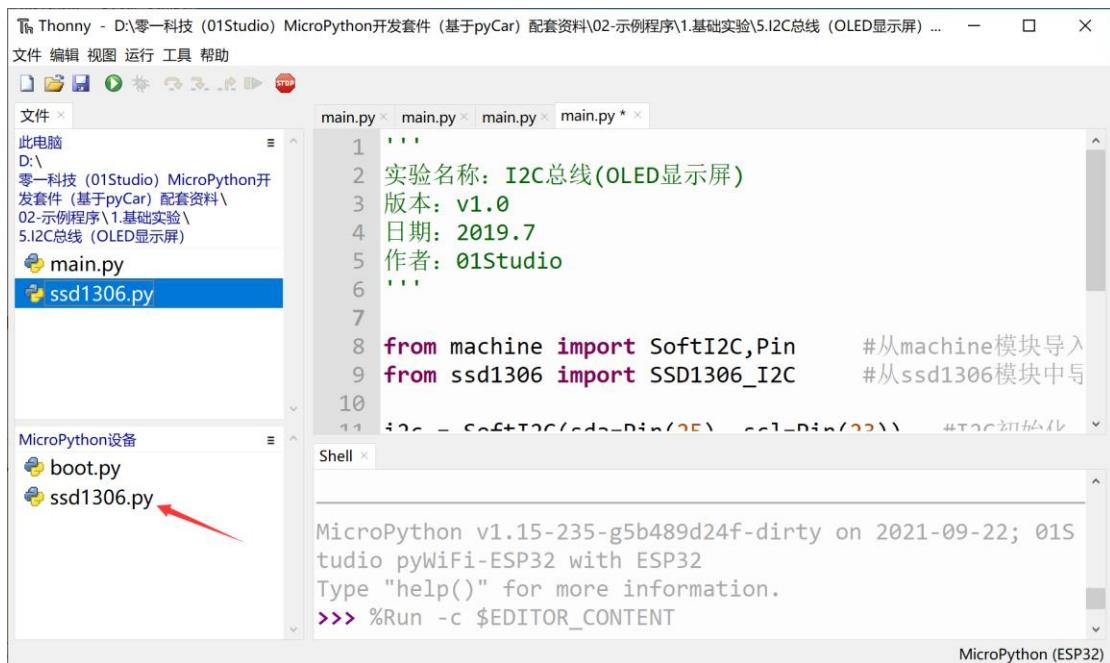


图 4-24 将 main.py、ssd1306.py 拷贝到设备

- 实验结果：

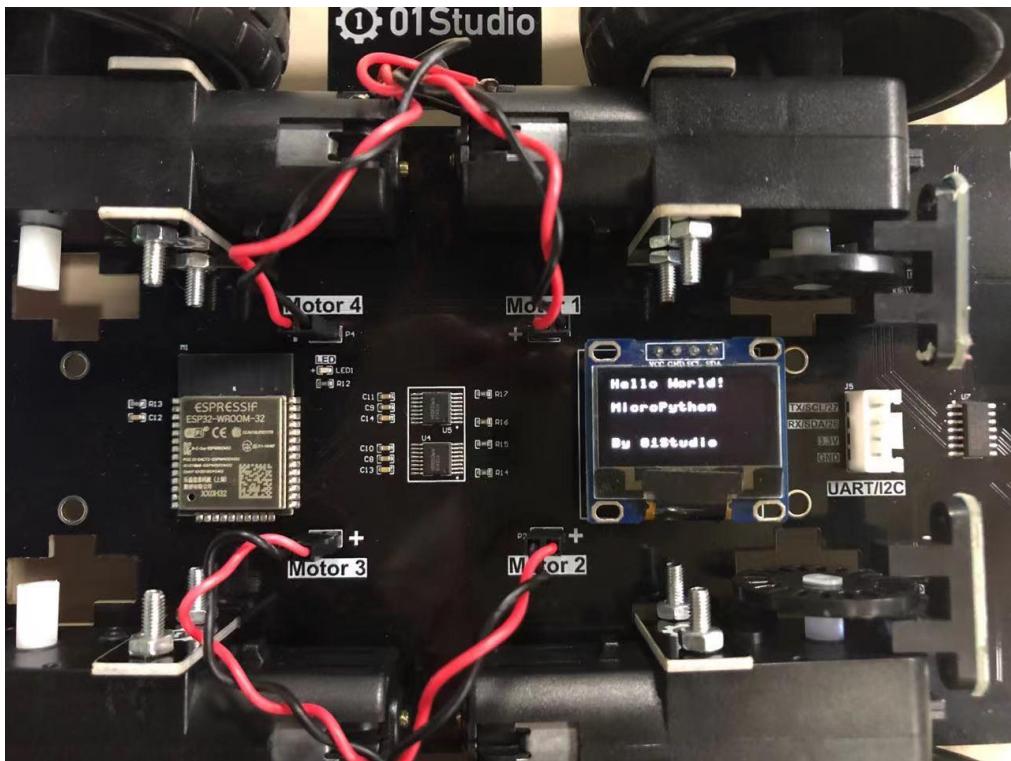


图 4-25 OLED 实验结果

- 总结：

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

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

## 4.7 RTC 实时时钟

- 前言：

时钟可以说我们日常最常用的东西了，手表、电脑、手机等等无时无刻不显示当前的时间。可以说每一个电子爱好者心中都希望拥有属于自己制作的一个电子时钟，接下来我们就用 MicroPython 开发板来制作一个属于自己的电子时钟。



图 4-26 时钟

- 实验平台：

pyCar。



图 4-27 pyCar 开发套件

- **实验目的:**

学习 RTC 编程和制作电子时钟，使用 OLED 显示。

- **实验讲解:**

实验的原理是读取 RTC 数据，然后通过 OLED 显示。毫无疑问，强大的 MicroPython 已经集成了内置时钟函数模块。位于 `machine` 的 RTC 模块中，具体介绍如下：

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

表 4-7

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

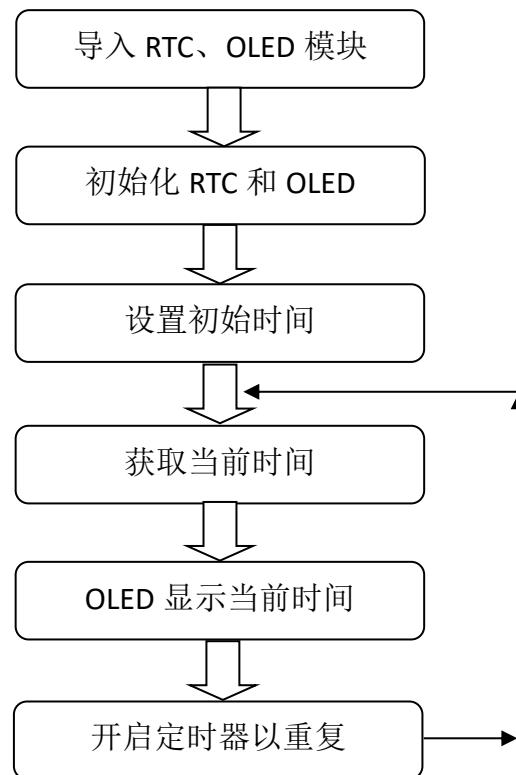


图 4-28

实验参考代码如下：

```

...
实验名称: RTC 实时时钟
版本: v1.0
日期: 2021.12
作者: 01Studio
...

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

# 定义星期和时间（时分秒）显示字符列表
week = ['Mon', 'Tues', 'Wed', 'Thur', 'Fri', 'Sat', 'Sun']
time_list = ['', '', '']


```

```

# 初始化所有相关对象

i2c = SoftI2C(sda=Pin(25), scl=Pin(23)) #I2C: sda--> 25, scl --> 23
oled = SSD1306_I2C(128, 64, i2c, addr=0x3c)
rtc = RTC()

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

if rtc.datetime()[0] != 2021:
    rtc.datetime((2021, 4, 1, 0, 0, 0, 0))

def RTC_Run(tim):
    datetime = rtc.datetime() # 获取当前时间

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

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

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

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

```

```

# 显示时间

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

oled.show()

#开启 RTOS 定时器

tim = Timer(-1)

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

```

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

### ● 实验结果：

烧录程序复位后，可以看到时钟开始跑起来。

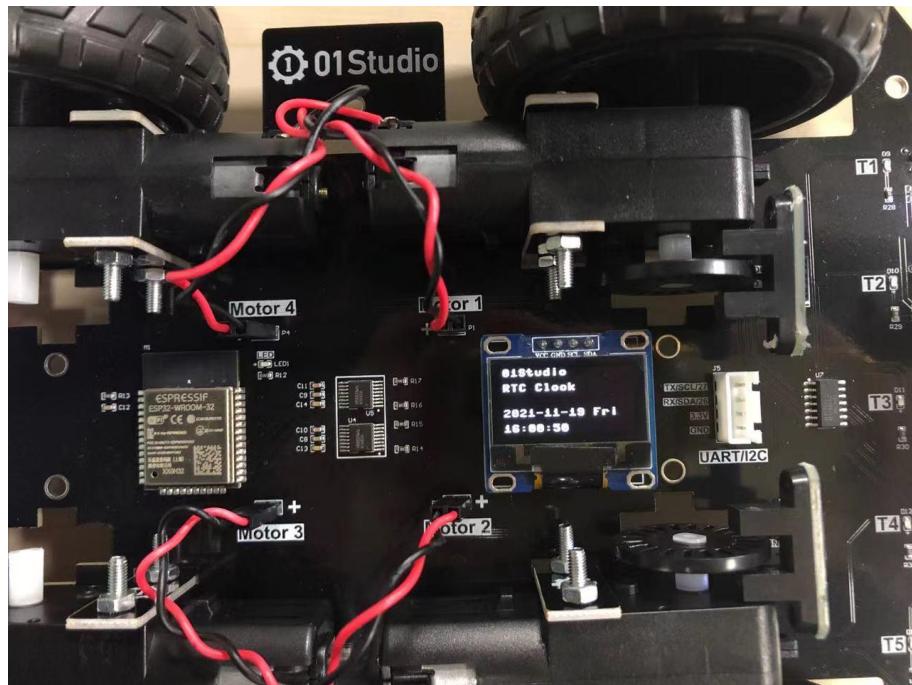


图 4-29 RTC 实时时钟实验

- **总结：**

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

## 4.8 UART（串口通信）

- **前言：**

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

- **实验平台：**

pyCar 开发套件，需要使用 USB 转 TTL 工具。



图 4-30pyCar 开发板

- **实验目的：**

编程实现串口收发数据。

- **实验讲解：**

pyCar 开发板一共有 3 个串口，编号是 0-2，如下表：

UART(0)	TX	1
	RX	3
UART(1)	TX	10
	RX	9

<b>UART(2)</b>	<b>TX</b>	<b>17</b>
	<b>RX</b>	<b>16</b>

表 4-8 ESP32 串口引脚资源

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

我们来了解一下串口对象的构造函数和使用方法：

构造函数
<code>uart=machine.UART(id,baudrate,tx=None,rx=None,bits=8,parity=None,stop=1,...)</code>
创建 UART 对象。
<b>【id】</b> 0-2
<b>【baudrate】</b> 波特率，常用 115200、9600
<b>【tx】</b> 自定义 IO
<b>【rx】</b> 自定义 IO
<b>【bits】</b> 数据位
<b>【parity】</b> 校验；默认 None, 0(偶校验), 1(奇校验)
<b>【stop】</b> 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=27,rx=26 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>
<b>【nbytes】</b> 读取字节数

UART.readline()
读行
UART.write( <i>buf</i> )
【 <i>buf</i> 】串口 TX 写数据
*更多使用说明请阅读官方文档： <a href="http://docs.01studio.org/esp32/quickref.html#uart-serial-bus">http://docs.01studio.org/esp32/quickref.html#uart-serial-bus</a>

表 4-9 machine 的 UART 对象

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

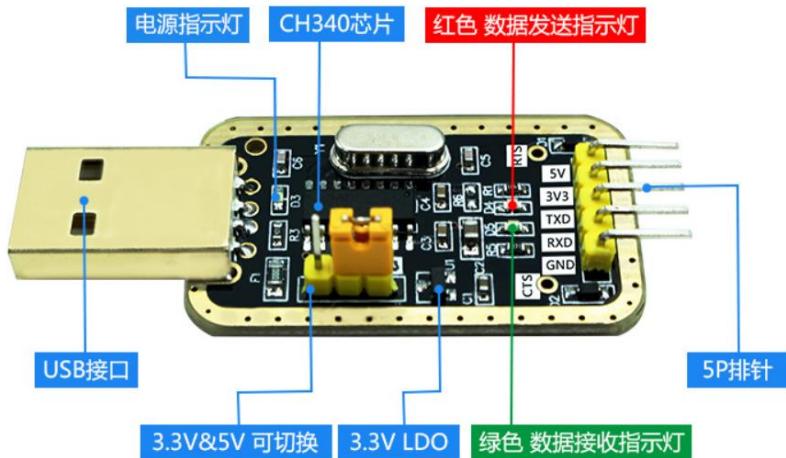


图 4-31 常用 USB 转串口工具

注意要使用 3.3V 电平的 USB 转串口 TTL 工具，本实验我们使用串口 2，也就是 27 (TX) 和 26 (RX)，接线示意图如下：



图 4-32 串口接线图

在本实验中我们可以先初始化串口，然后给串口发去一条信息，这样 PC 机的串口助手就会在接收区显示出来，然后进入循环，当检测到有数据可以接收时候就将数据接收并打印，并通过 REPL 打印显示。代码编写流程图如下：

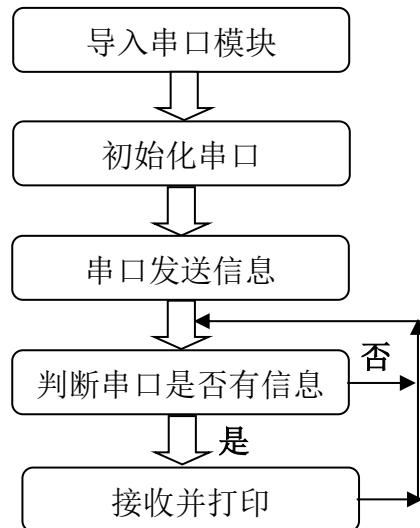


图 4-33 代码编程流程图

参考代码：

```
...
实验名称: 串口通信
版本: v1.0
日期: 2021.12
作者: 01Studio
说明: 通过编程实现串口通信，跟电脑串口助手实现数据收发。
平台: pyCar
参考: http://docs.01studio.org/esp32/quickref.html#uart-serial-bus
...
#导入串口模块
from machine import UART

uart=UART(2,115200,tx=27, rx=26) #设置串口号 2 和波特率,TX--27,RX--26
```

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

while True:

    #判断有无收到信息

    if uart.any():

        text=uart.read(128) #接收 128 个字符

        print(text) #通过 REPL 打印串口 3 接收的数据
```

### ● 实验结果：

我们按照上述方式将 USB 转 TTL 的 TX 接到开发板的 RX (26), USB 转 TTL 的 RX 接到开发板的 TX (27)。GND 接一起，3.3V 可以选择接或不接。**(特别注意超声波传感器不能接。)**

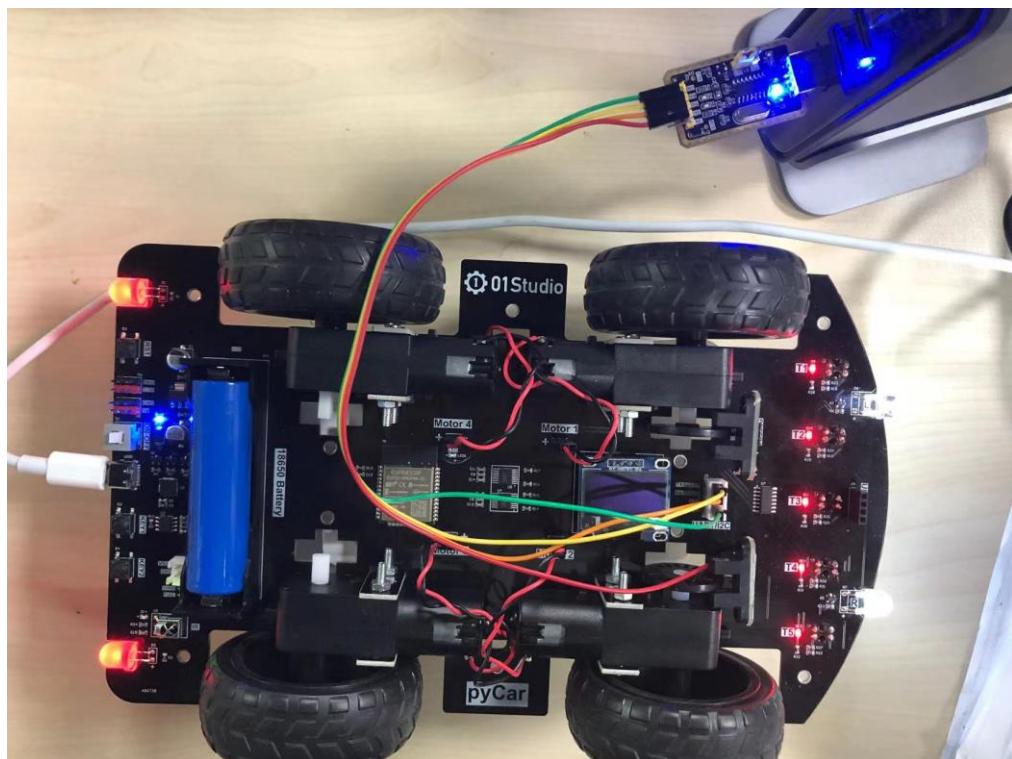


图 4-34 将 pyCar 和 usb ttl 工具同时接入电脑

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



图 4-35

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



图 4-36

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

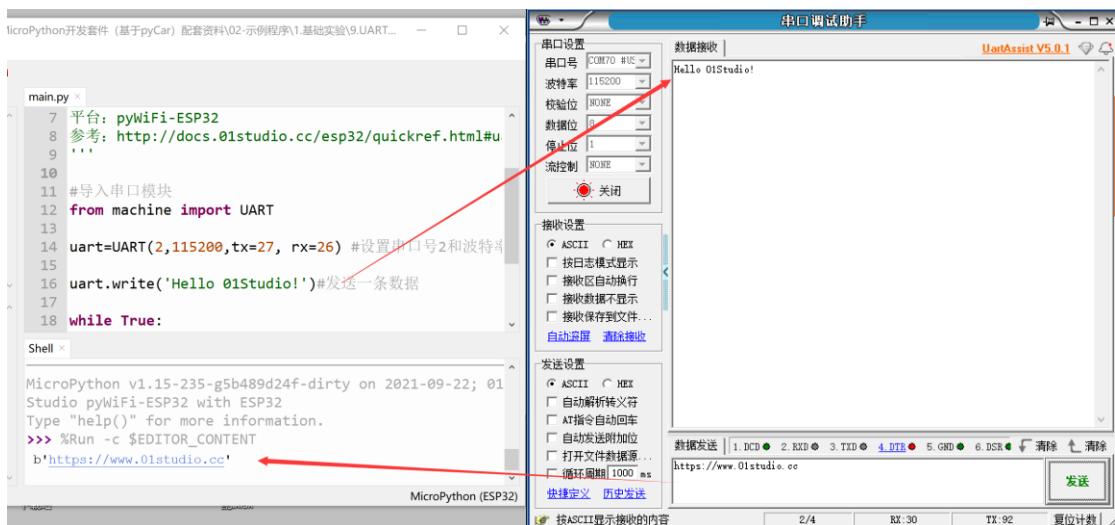


图 4-37 串口收发实验

### ● 总结：

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

## 第5章 WiFi 应用

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

## 5.1 连接无线路由器

- **前言：**

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

- **实验平台：**

pyCar。

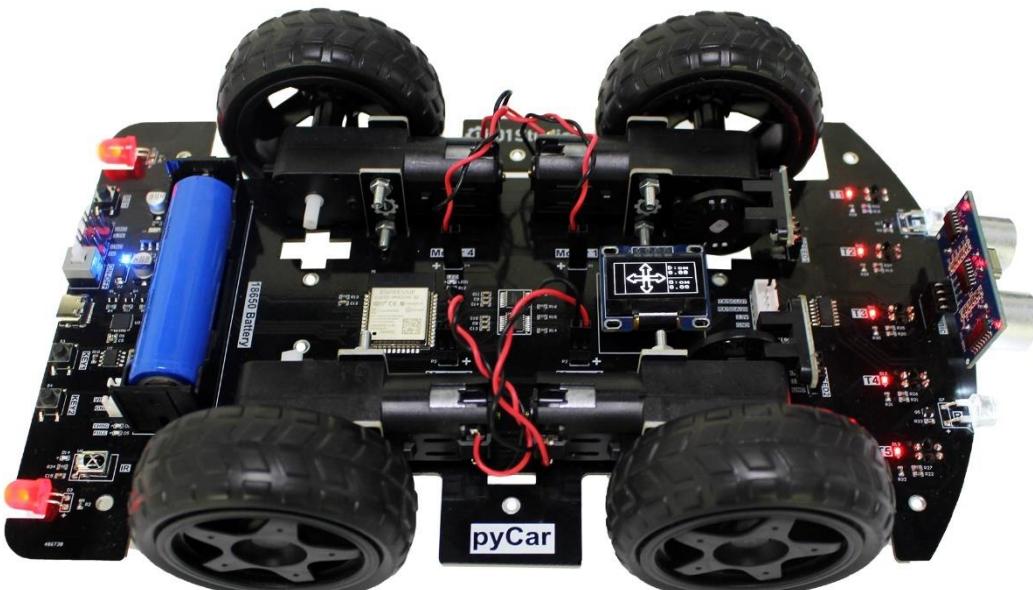


图 5-1 pyCar 开发套件

- **实验目的：**

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

- **实验讲解：**

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

MicroPython 已经集成了 network 模块，开发者使用内置的 network 模块函

数可以非常方便地连接上路由器。但往往也有各种连接失败的情况，如密码不正确等。这时候我们只需要再加上一些简单的判断机制，避免陷入连接失败的死循环即可！

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

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

表 5-1 WLAN 对象

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

模块上电后可以先判断是否已经连接到网络，如果是则无需再次连接，否的话则进入 WiFi 连接状态，指示灯闪烁，连接成功后指示灯常亮，IP 等相关信息通过 OLED 显示和串口打印。另外需要配置超时 15 秒还没连接成功时执行取消连接，避免因无法连接而陷入死循环。代码编写流程如下：

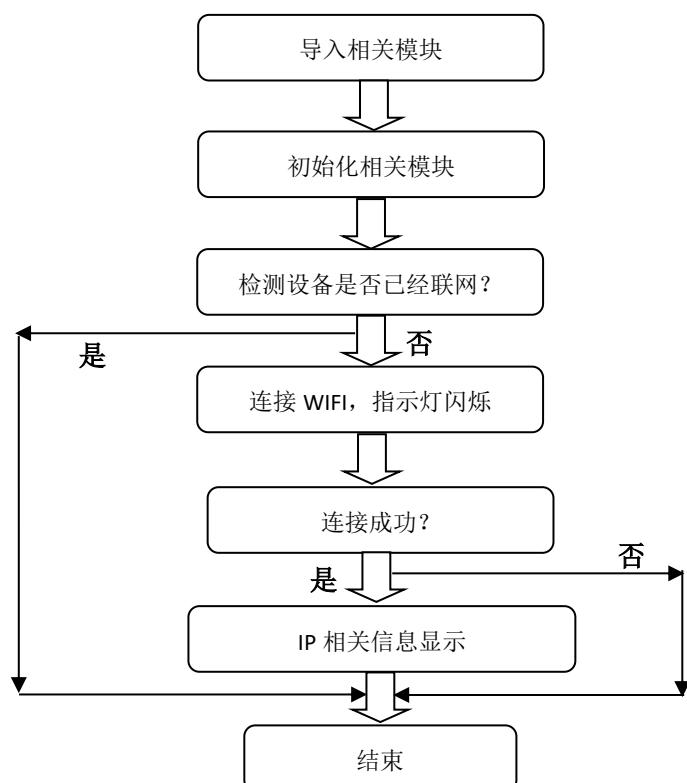


图 5-2 代码编写流程图

实验参考代码如下：

```
...  
实验名称: 连接无线路由器  
版本: v1.0  
日期: 2021.12  
作者: 01Studio  
说明: 编程实现连接路由器, 将 IP 地址等相关信息通过 OLED 显示 (只支持 2.4G 网络)。  
...  
import network,time
```

```

from machine import SoftI2C,Pin
from ssd1306 import SSD1306_I2C

#初始化相关模块
i2c = SoftI2C(sda=Pin(25), scl=Pin(23))
oled = SSD1306_I2C(128, 64, i2c, addr=0x3c)

#WIFI 连接函数
def WIFI_Connect():

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

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

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

    while not wlan.isconnected():

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

    #超时判断,15 秒没连接成功判定为超时
    if time.time()-start_time > 15 :

```

```
    print('WIFI Connected Timeout!')  
    break  
  
if wlan.isconnected():  
    #LED 点亮  
    WIFI_LED.value(1)  
  
    #串口打印信息  
    print('network information:', wlan.ifconfig())  
  
    #OLED 数据显示  
    oled.fill(0)    #清屏背景黑色  
    oled.text('IP/Subnet/GW:', 0, 0)  
    oled.text(wlan.ifconfig()[0], 0, 20)  
    oled.text(wlan.ifconfig()[1], 0, 38)  
    oled.text(wlan.ifconfig()[2], 0, 56)  
    oled.show()  
  
#执行 WIFI 连接函数  
WIFI_Connect()
```

上面代码在 `main.py` 文件中，将 `WIFI` 连接封装成了子函数形式，我们也可以新建一个 `WIFI.py` 文件，然后通过模块引用方式来供 `main.py` 调用，以提高程序的灵活性。

### ● 实验结果：

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

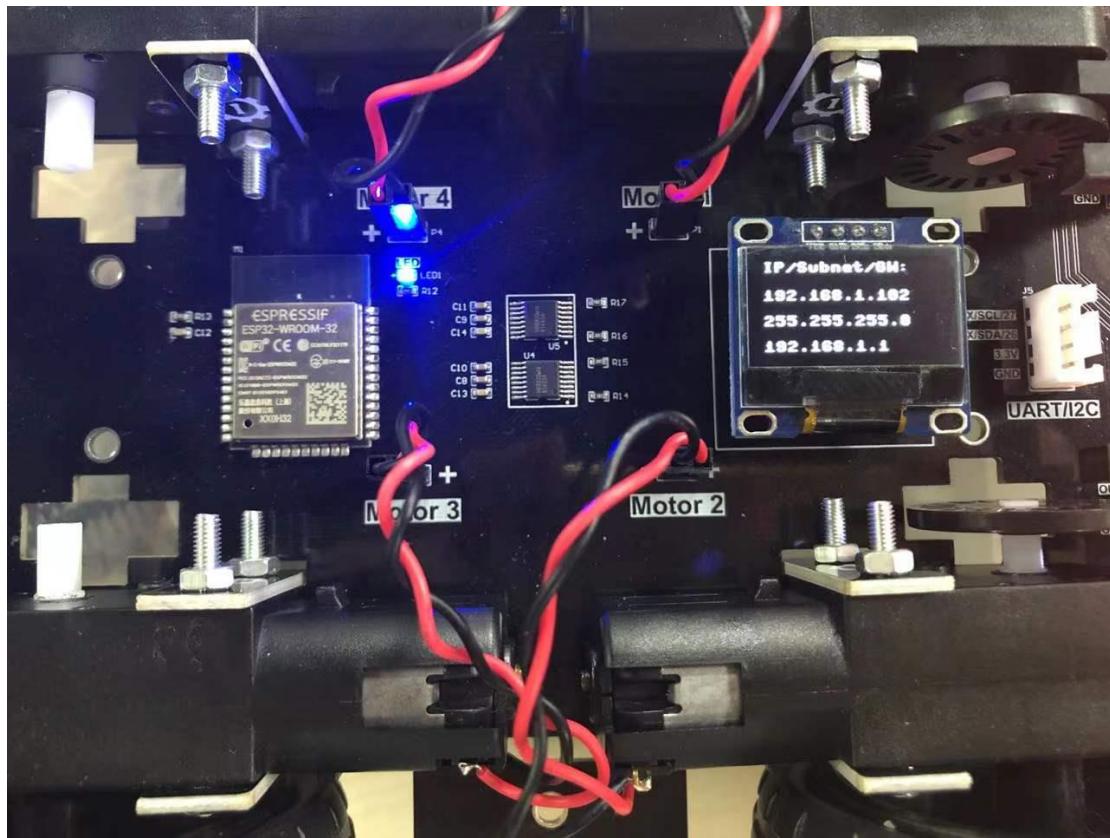


图 5-3

● 总结：

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

## 5.2 Socket 通信

- 前言：

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

- 实验平台：

pyCar。

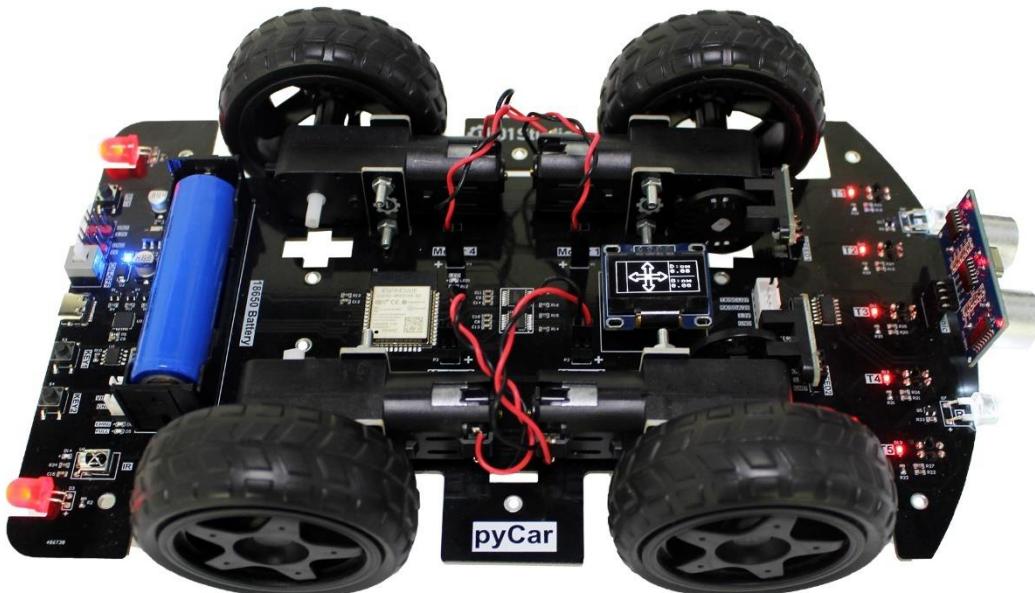


图 5-4 pyCar 开发套件

- 实验目的：

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

- 实验讲解：

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

我们先来看看网络层级模型图，这是构成网络通信的基础：

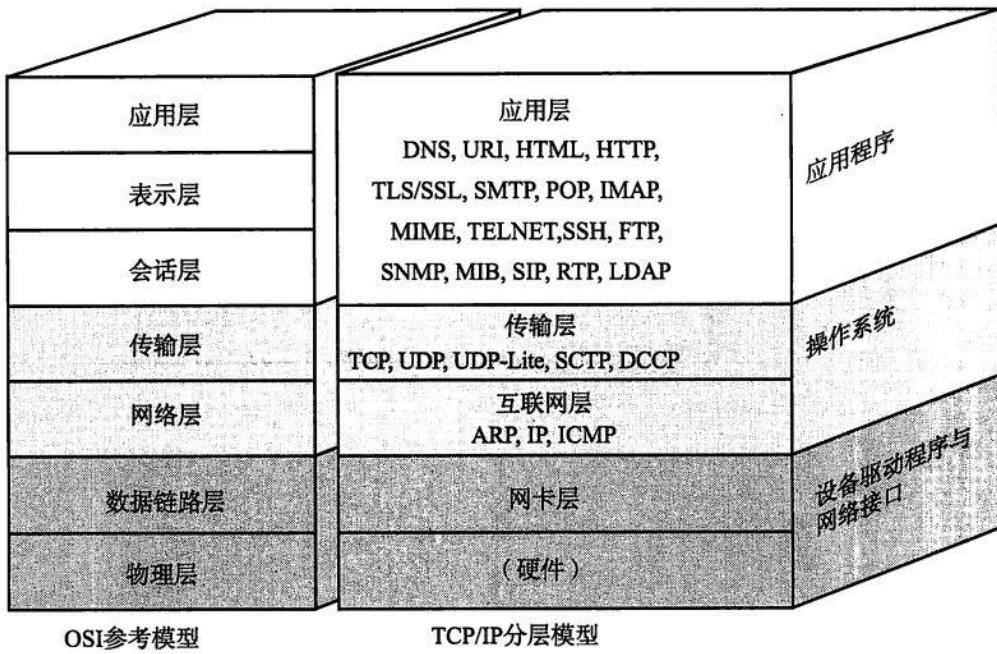


图 5-5 网络层级模型

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

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

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

现数据传输的并发服务。

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

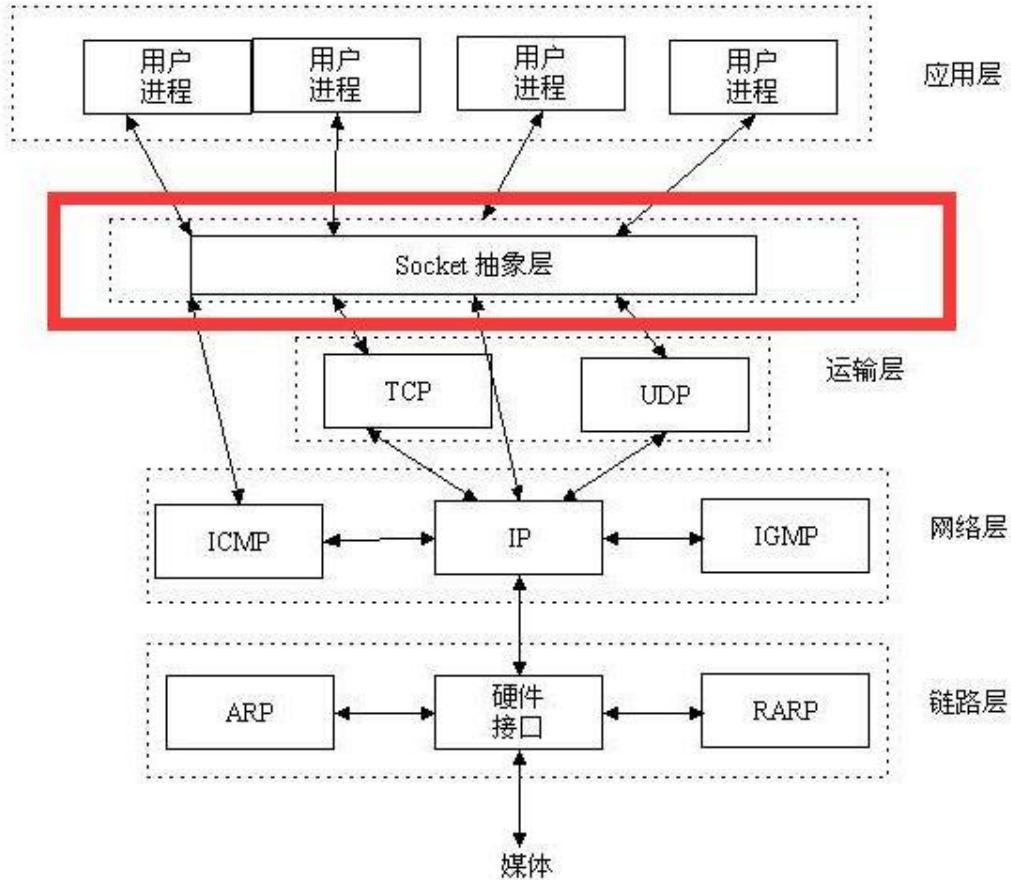


图 5-6 Socket 抽象层

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

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

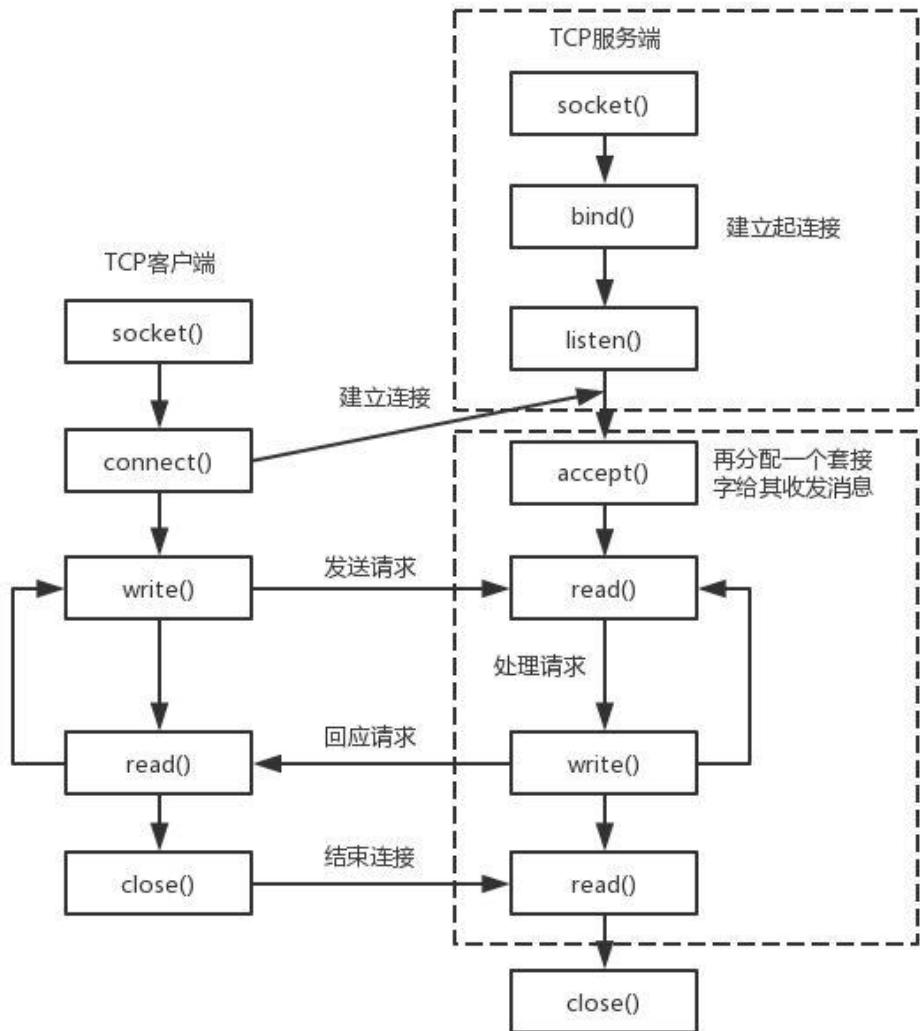


图 5-7 Socket 通信过程

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

以上的内容，简单来说就是如果用户面向应用来说，那么 ESP32 只需要知道通讯协议是 TCP 或 UDP、服务器的 IP 和端口号这 3 个信息，即可向服务器发起连接和发送信息。就这么简单。

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

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

表 5-2 Socket 对象

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

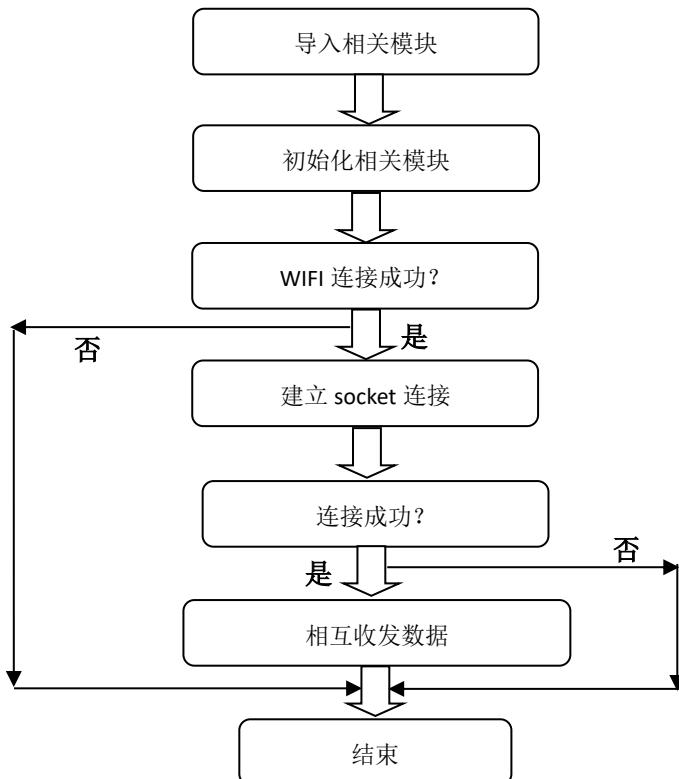


图 5-8 代码编写流程图

实验参考代码：

```

...
实验名称: 连接无线路由器
版本: v1.0
日期: 2021.12
作者: 01Studio
说明: 通过 Socket 编程实现 pyCar 与电脑服务器助手建立 TCP 连接, 相互收发数据。
...
#导入相关模块

import network,usocket,time
from machine import I2C,Pin,Timer
from ssd1306 import SSD1306_I2C

#初始化相关模块

```

```

i2c = I2C(sda=Pin(25), scl=Pin(23))
oled = SSD1306_I2C(128, 64, i2c, addr=0x3c)

# WIFI 连接函数

def WIFI_Connect():

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

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

    if not wlan.isconnected():
        print('connecting to network...')
        wlan.connect('01Studio', '88888888') #输入 WIFI 账号密码
        while not wlan.isconnected():
            #LED 闪烁提示
            WIFI_LED.value(1)
            time.sleep_ms(300)
            WIFI_LED.value(0)
            time.sleep_ms(300)

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

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

```

```
#串口打印信息

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


#OLED 数据显示

oled.fill(0)    #清屏背景黑色

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

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

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

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

oled.show()

return True


else:

    return False


def Socket_fun(tim):

    text=s.recv(128) #单次最多接收 128 字节

    if text == '':

        pass

    else: #打印接收到的信息为字节，可以通过 decode('utf-8')转成字符串

        print(text)

        s.send('I got:' +text.decode('utf-8'))


#判断 WIFI 是否连接成功

if WIFI_Connect():

    #创建 socket 连接 TCP 类似，连接成功后发送“Hello 01Studio!”给服务器。
```

```

s=usocket.socket()

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

s.connect(addr)

s.send('Hello 01Studio!')

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

tim = Timer(-1)

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

```

WIFI 连接代码在上一节已经讲解，这里不再重复，WIFI 连接成功后返回 True，否则返回 False。程序在返回连接成功后建了 Socket 连接，连接成功发送 ‘Hello 01Studio!’ 信息到服务器。另外 RTOS 定时器设定了每 300ms 处理从服务器接收到的数据。将接收到数据通过串口打印和发送给服务器。

### ● 实验结果：

先在电脑端打开网络调试助手并建立服务器，软件在 零一科技（01Studio）MicroPython 开发套件配套资料\_latest\01-开发工具\01-Windows\网络调试助手下的 NetAssist.exe ，直接双击打开即可！

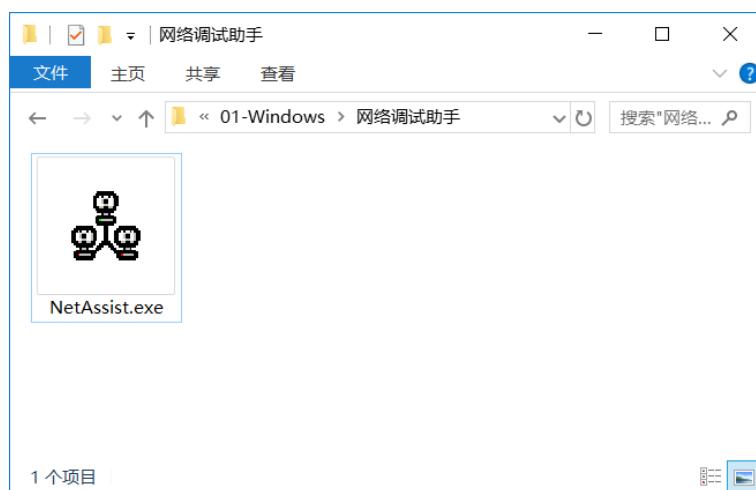


图 5-9 网络调试助手

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



图 5-10 TCP 服务器配置

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

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

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

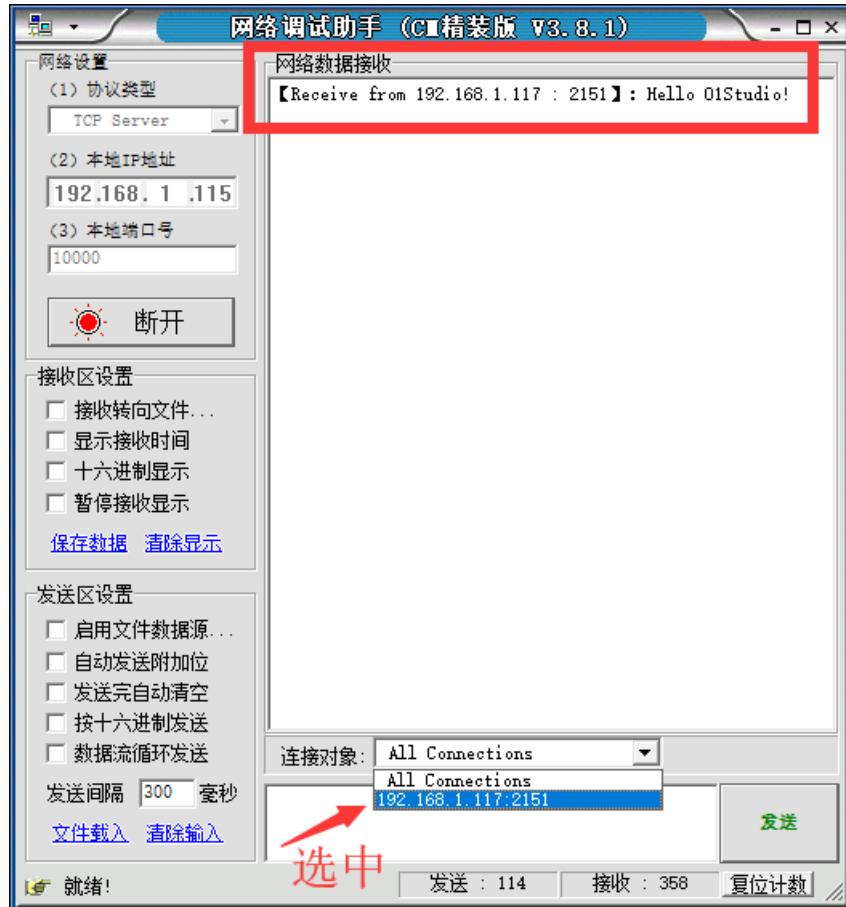


图 5-11 选中连接对象

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

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

图 5-12

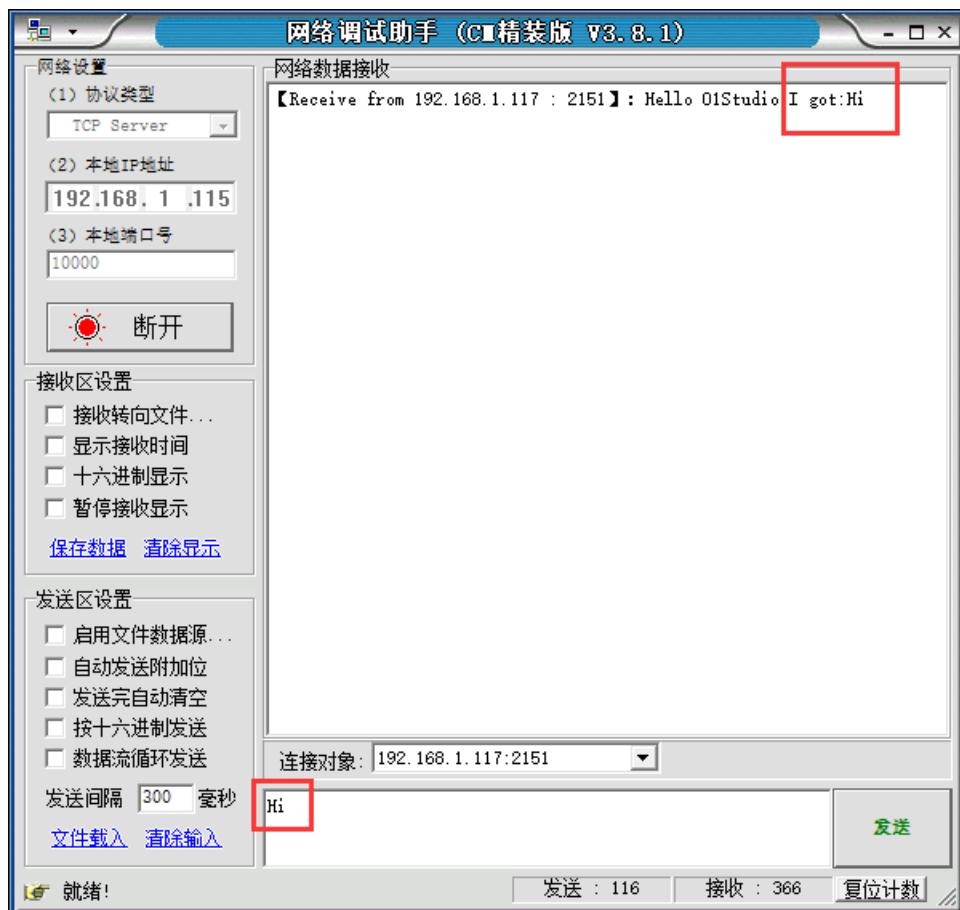


图 5-13 服务器发送数据

### ● 总结：

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

## 5.3 MQTT 通信

- **前言：**

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

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

- **实验平台：**

pyCar。



图 5-14 pyCar 开发套件

- **实验目的：**

通过编程实现 pyCar 实现 MQTT 协议信息的发布和订阅（接收）。

- **实验讲解：**

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

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

总结下来 MQTT 有如下特性/优势：

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

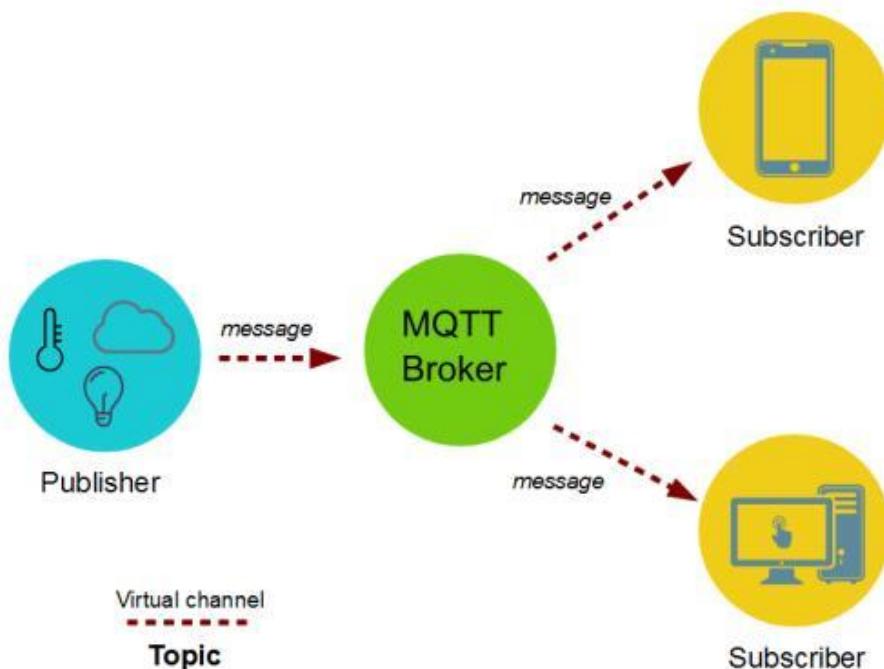


图 5-15 MQTT 通信流程

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

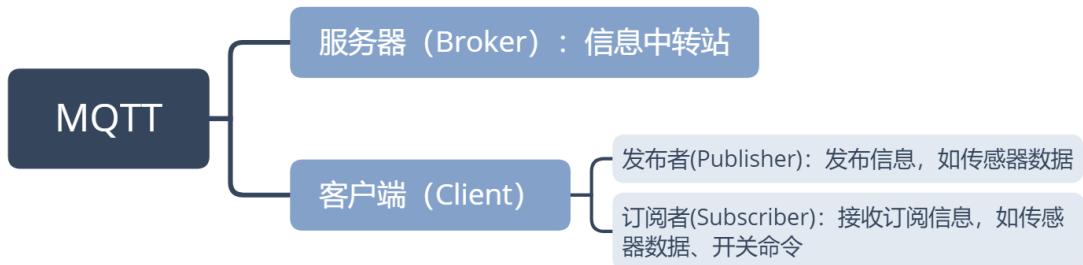


图 5-16 MQTT 角色说明

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

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

表 5-3 MQTT 数据格式

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

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

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

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

表 5-4 MQTT 客户端对象

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

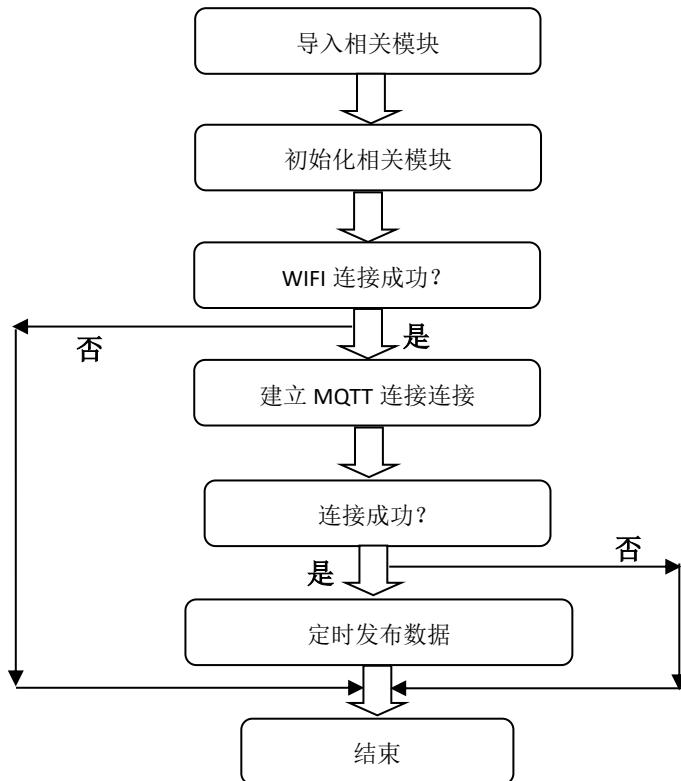


图 5-17 发布者代码编写流程

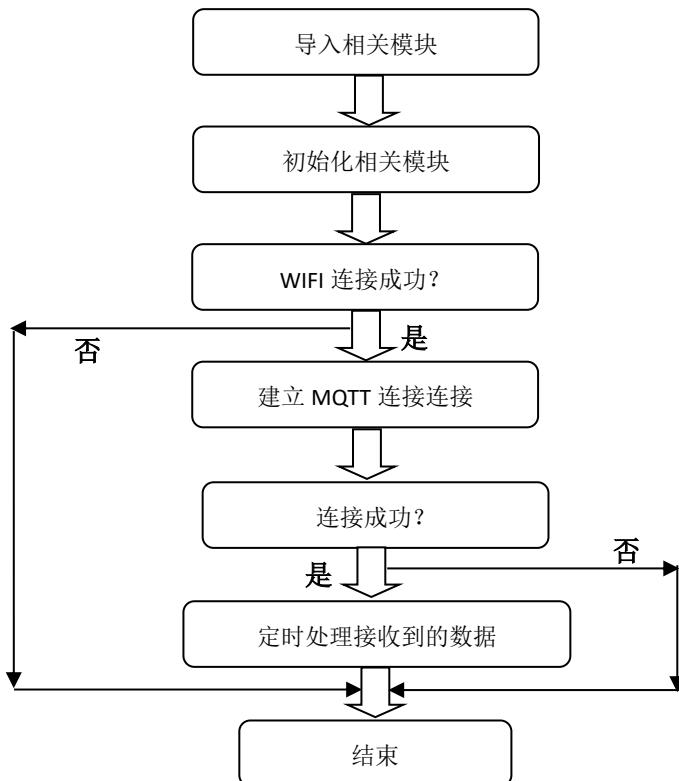


图 5-18 订阅者代码编写流程

发布者（publish）参考代码：

```

...
实验名称: MQTT 通信
版本: v1.0
日期: 2021.12
作者: 01Studio
说明: 编程实现 MQTT 通信, 实现发布数据。
...

import network,time
from simple import MQTTClient #导入 MQTT 板块
from machine import SoftI2C,Pin,Timer
from ssd1306 import SSD1306_I2C
#初始化相关模块

```

```

i2c = SoftI2C(sda=Pin(25), scl=Pin(23))
oled = SSD1306_I2C(128, 64, i2c, addr=0x3c)

#WIFI 连接函数

def WIFI_Connect():

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

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

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

        while not wlan.isconnected():

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

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

    if wlan.isconnected():

```

```

#LED 点亮

WIFI_LED.value(1)


#串口打印信息

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


#OLED 显示数据（如果没接 OLED， 请将下面代码屏蔽）

oled.fill(0)    #清屏背景黑色

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

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

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

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

oled.show()

return True


else:

    return False


#发布数据任务

def MQTT_Send(topic):

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


#执行 WIFI 连接函数并判断是否已经连接成功

if WIFI_Connect():

    SERVER = 'mq.tongxinmao.com'

    PORT = 18830

    CLIENT_ID = 'ESP-32' # 客户端 ID

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

    client = MQTTClient(CLIENT_ID, SERVER, PORT)

```

```
client.connect()

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

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

订阅者（subscribe）参考代码：

```
'''

实验名称：MQTT 通信

版本：v1.0

日期：2019.8

作者：01Studio

说明：编程实现 MQTT 通信，实现订阅（接收）数据。

'''

import network,time

from simple import MQTTClient #导入 MQTT 板块

from machine import SoftI2C,Pin,Timer

from ssd1306 import SSD1306_I2C


#初始化相关模块

i2c = SoftI2C(sda=Pin(25), scl=Pin(23))

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


#WIFI 连接函数

def WIFI_Connect():

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


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

```

wlan.active(True) #激活接口

start_time=time.time() #记录时间做超时判断


if not wlan.isconnected():

    print('connecting to network...')

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


    while not wlan.isconnected():

        #LED 闪烁提示

        WIFI_LED.value(1)

        time.sleep_ms(300)

        WIFI_LED.value(0)

        time.sleep_ms(300)

#超时判断,15 秒没连接成功判定为超时

if time.time()-start_time > 15 :

    print('WIFI Connected Timeout!')

    break


if wlan.isconnected():

    #LED 点亮

    WIFI_LED.value(1)


    #串口打印信息

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


#OLED 数据显示（如果没接 OLED, 请将下面代码屏蔽）

oled.fill(0) #清屏背景黑色

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

```

```

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

    else:
        return False

#设置 MQTT 回调函数,有信息时候执行
def MQTT_callback(topic, msg):
    print('topic: {}'.format(topic))
    print('msg: {}'.format(msg))

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

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

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

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

```

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

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

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

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

### ● 实验结果：

为了方便测试，我们可以使用 MQTT 网络助手进行调试。这里推荐一个在线 MQTT 网络调试助手：<http://www.tongxinmao.com/txm/webmqtt.php#collapseOne>

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

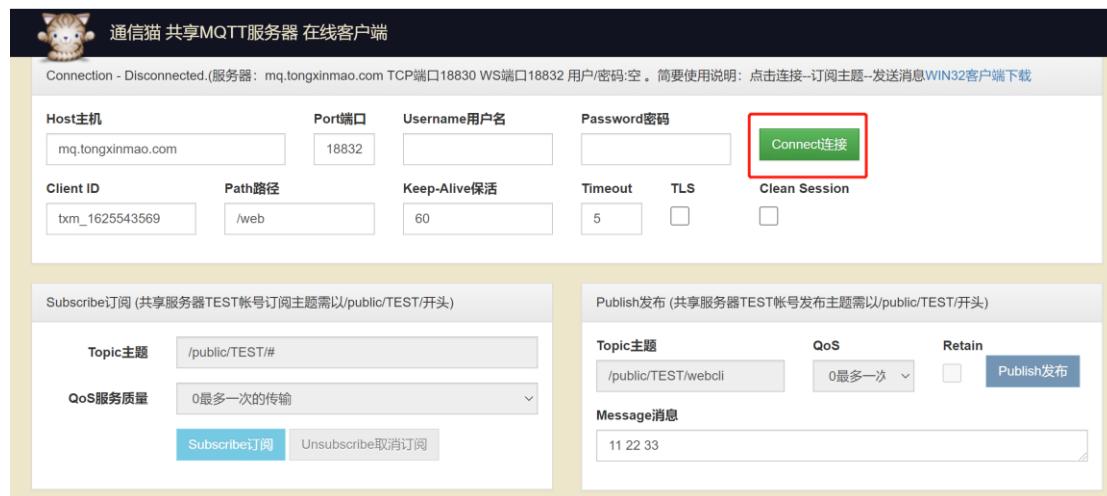


图 5-19 MQTT 在线助手

连接成功可以看到显示。



图 5-20 MQTT 助手连接成功

### 测试“发布者”代码：

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

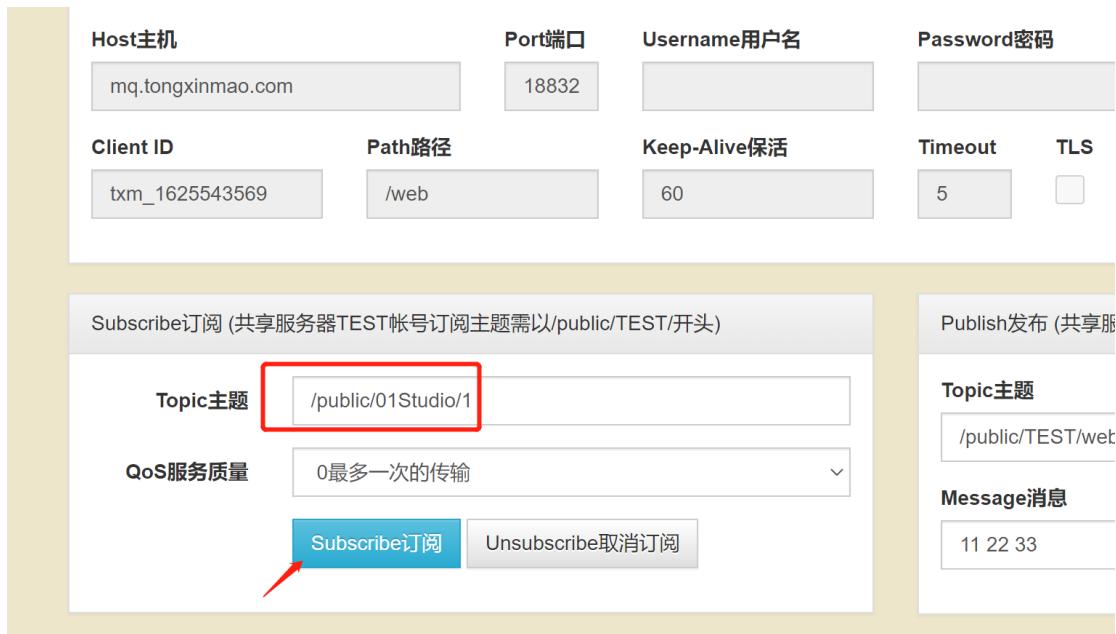


图 5-21 订阅主题

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

板发布的信息。

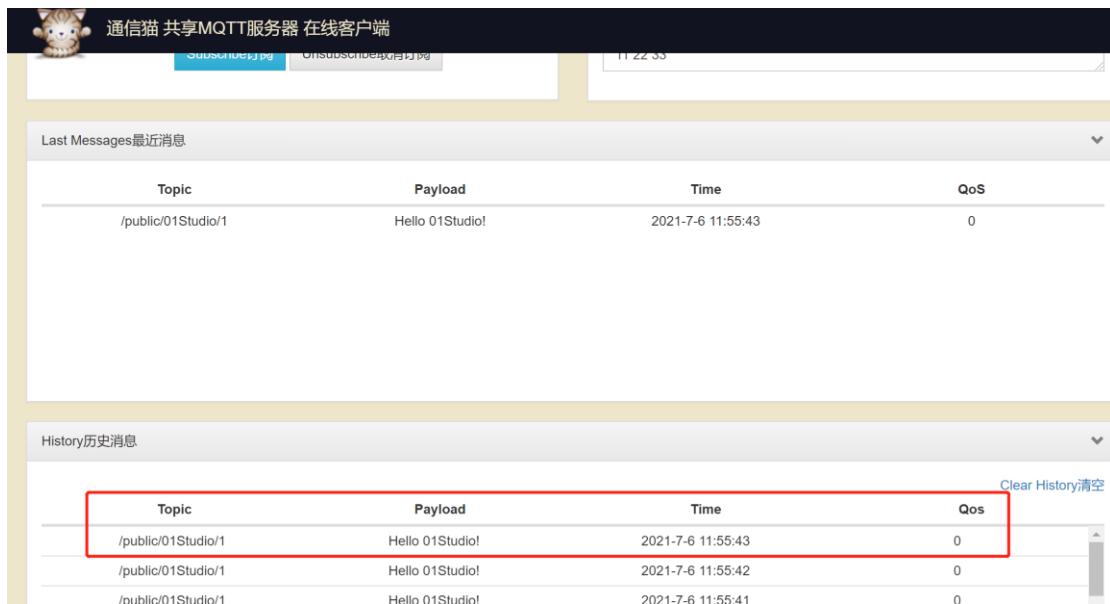


图 5-22 MQTT 助手收到开发板发布的信息

### 测试“订阅者”代码：

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

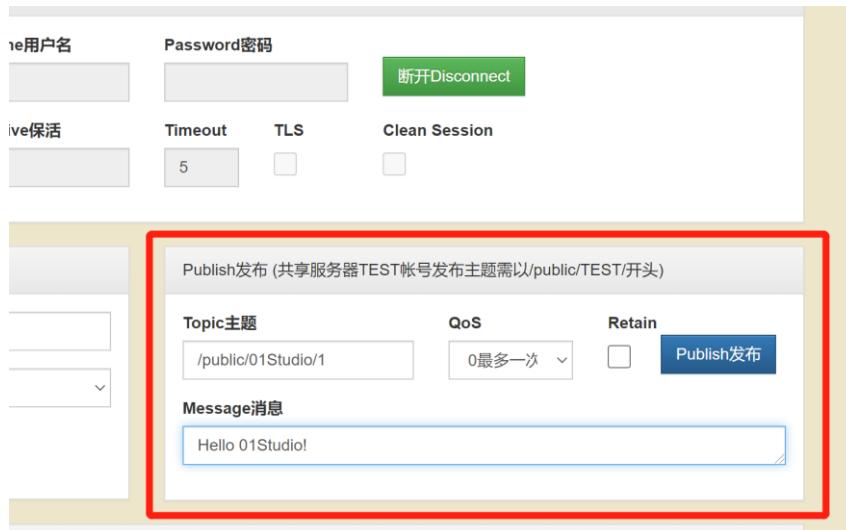


图 5-23 输入即将发布的主题和信息

开发板运行“订阅者”代码，成功连接后回到 MQTT 助手点击【发布】按钮

发布信息，可以在开发板的 REPL 看到接收到的订阅信息“Hello 01Studio!”被打印出来了（数据格式为字节数据）。

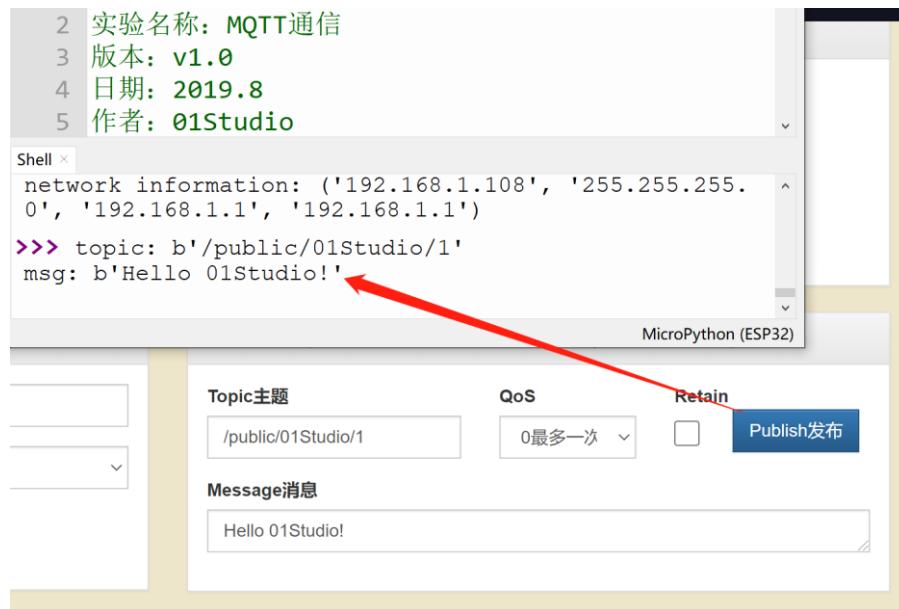


图 5-24 开发板接收到相关信息并打印

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



图 5-25 客户端同时发布和订阅信息

### ● 总结：

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

## 5.4 WebREPL

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

### 第1步:

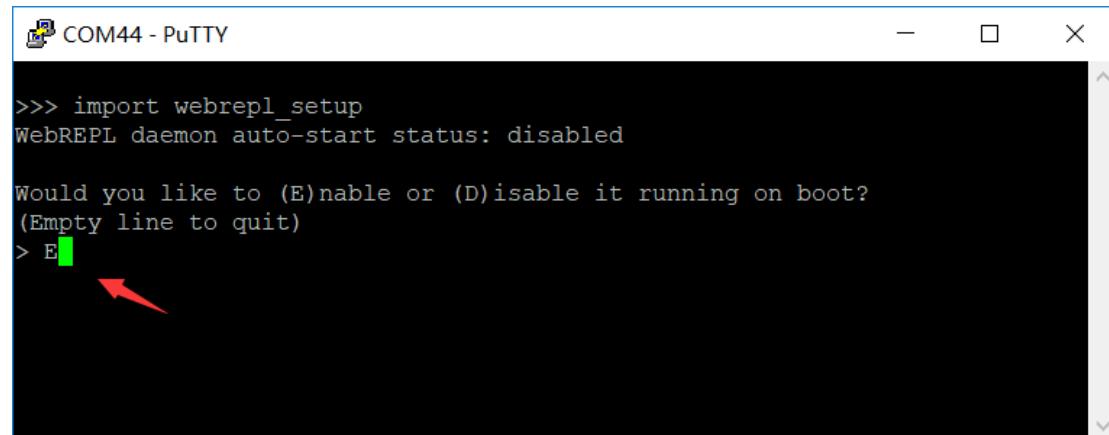
确保你的电脑和设备已经连接到同一个网络上（通常指同一个网段，如 192.168.1.X）。具体可以参考本章连接无线路由器章节实验。

### 第2步: 配置开发板

在串口终端输入以下命令：

```
import webrepl_setup
```

在弹出的串口提示框中输入'E'，按回车，使能 WebREPL 相关功能；

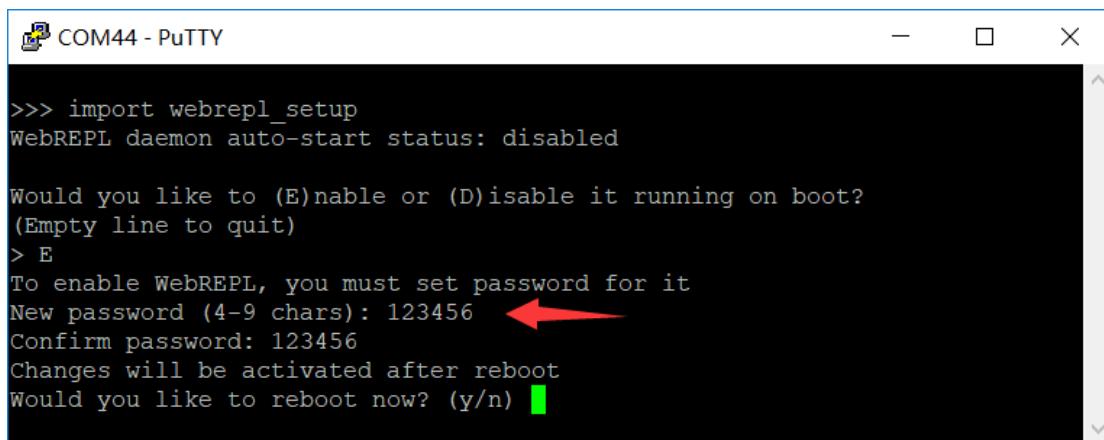


```
>>> import webrepl_setup
WebREPL daemon auto-start status: disabled

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

图 5-26

接下来设置密码，设置完成后输入‘y’，选择重启。



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

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

图 5-27

### 第3步:

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

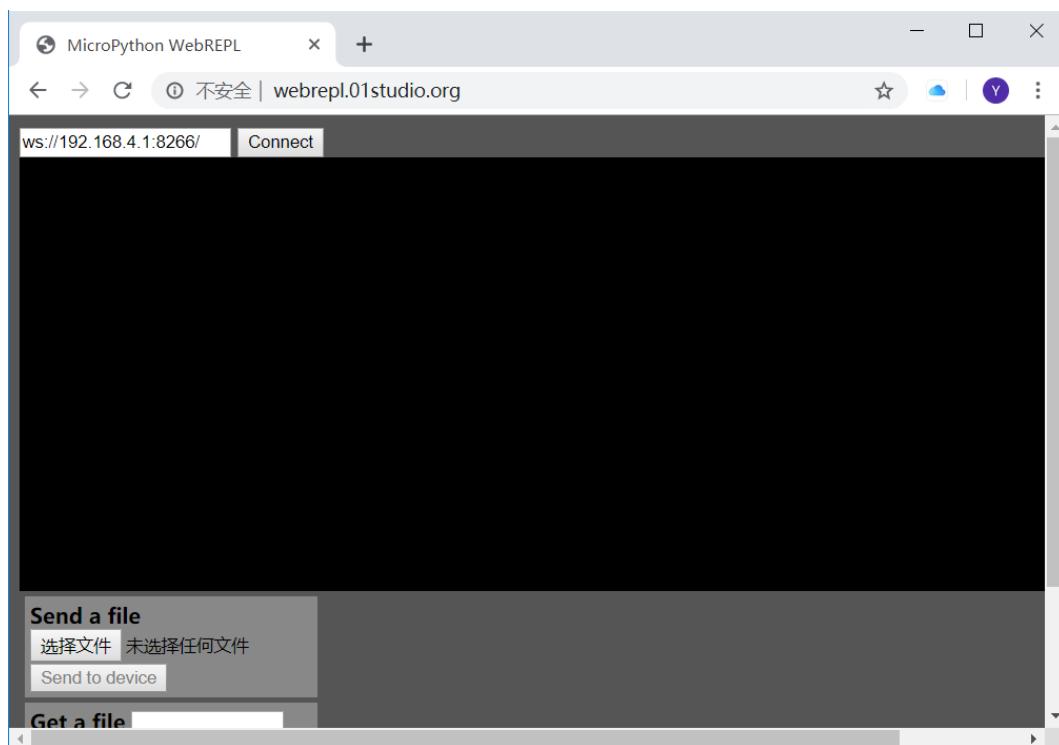


图 5-28 WebREPL

### 第4步:

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

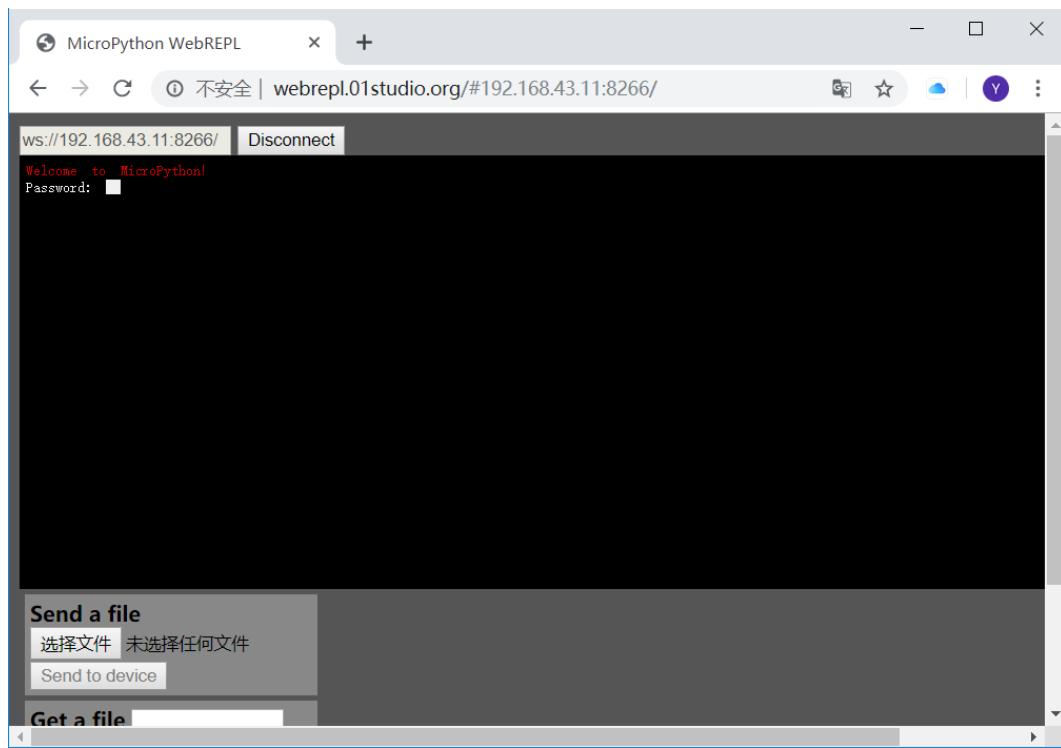


图 5-29

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

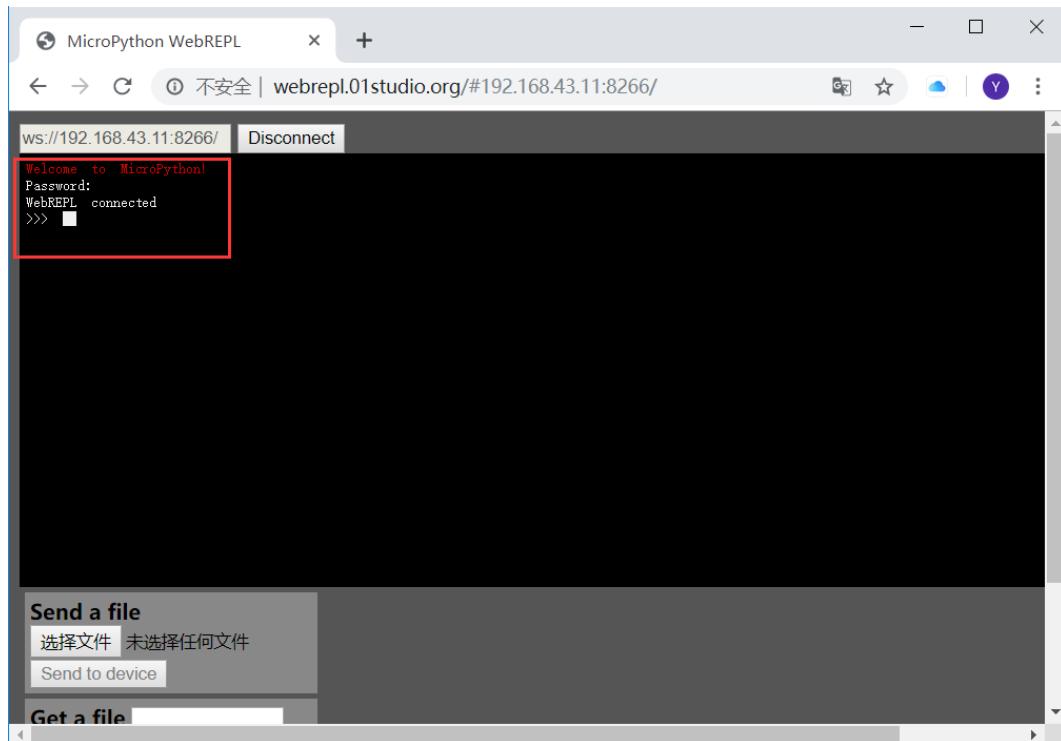


图 5-30 WebREPL 连接成功

## 第5步：文件传输

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

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

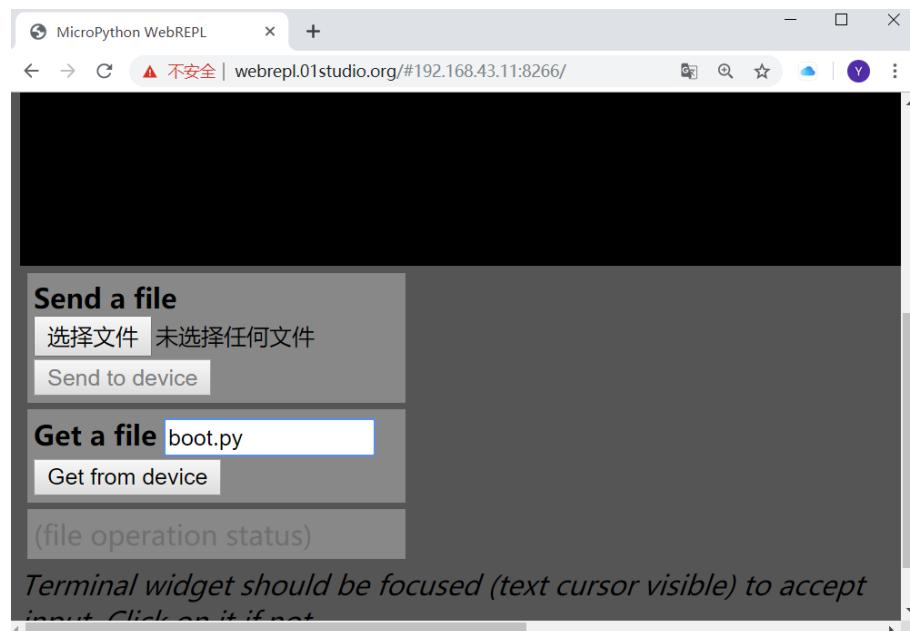


图 5-31 文件收发

## 第6章 pyCar 子模块实验

基础实验让我们对 MicroPython 的操作和编程有了比较全面的了解，但其魅力绝不限于此。前面我们也使用了不少的 `micropython` 库，本章我们来使用以下 `pyCar` 的相关库，`pyCar` 相关库已经将车灯、电机控制、红外遥控等功能封装成库，用户只需简单的调用即可使用。

`pyCar` 的库代码完全开源，依赖于用户不断完善。

<https://github.com/01studio-lab/pyCar>

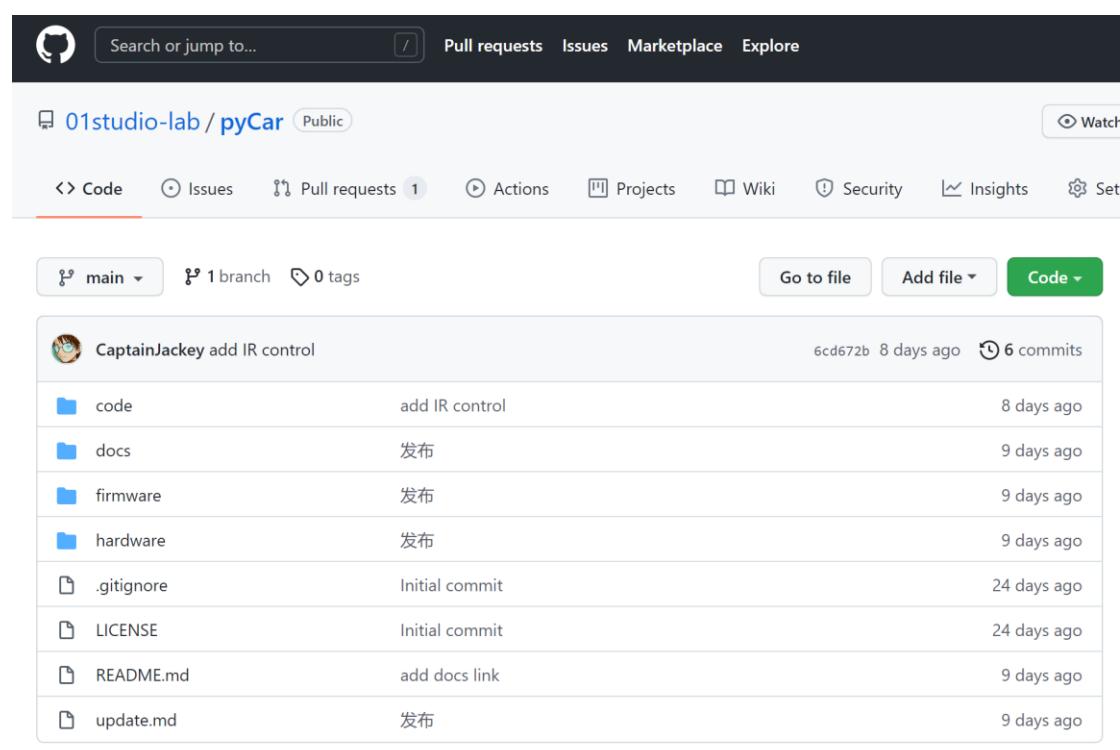


图 6-1

## 6.1 车头灯

- **前言:**

pyCar 带有 2 个车头草帽灯，这 2 个车头灯是可以控制的。本节我们就来学习如何通过 pyCar 的库文件实现最简单的车头灯控制。

- **实验平台:**

pyCar。

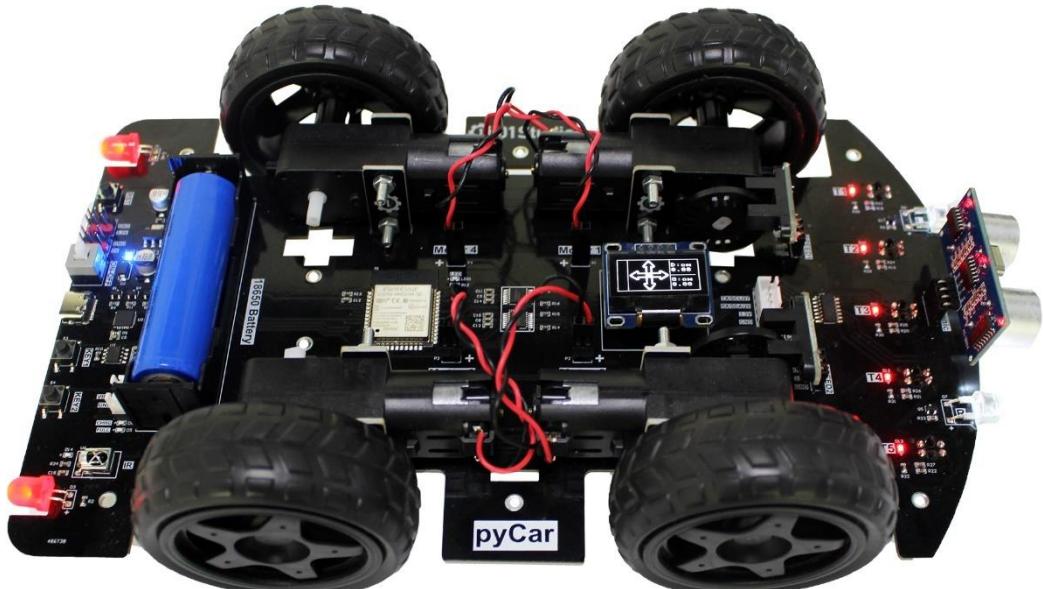


图 6-2 pyCar

- **实验目的:**

点亮车头灯。

- **实验讲解:**

前面我们学习过点亮 LED 灯内容，那是通过 `machine` 的 `GPIO` 库实现，这里我们直接通过 `car.py` 里面的模块来实现，底层实现原理一样，但是这样再封装了一层，使用会更直观一点。

构造函数
<code>Car=car.CAR()</code>
构造 pyCar 对象。
使用方法
<code>Car.light_on()</code>
打开车头灯。
<code>Car.light_off()</code>
关闭车头灯。
<code>Car.light(value=0)</code>
车头灯开关设置：
【value】 0:关闭； 1： 打开。
更详细内容请看官方文档: <a href="https://pycar.01studio.cc/">https://pycar.01studio.cc/</a>

表 6-1 pyCar 对象

上表对 car.py 里面的车头灯使用方法做了详细说明，我们只需要导入 car.py，构建对象后即可直接控制车头灯。代码编写流程如下：

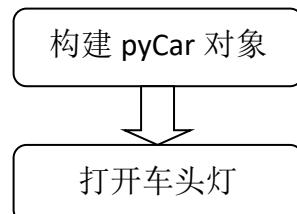


图 6-3 代码编写流程

参考代码如下：

```

...
实验名称: 点亮车头灯
版本: v1.0
日期: 2021.12
作者: 01Studio
...

```

```
from car import CAR  
  
Car = CAR() #构建 pyCar 对象  
  
Car.light_on() #点亮车头灯
```

- **实验结果：**

下载程序，可以看到车头灯成功点亮。



图 6-4

- **总结：**

从第一个车头灯实验不难看到，pyCar 库的封装让其变动简单易用。接下来我们将学习更多的功能应用。

## 6.2 动作

- **前言:**

pyCar 带有 4 路直流电机，每路可以单独控制。本节我们学习小车的各种移动动作。

- **实验平台:**

pyCar。

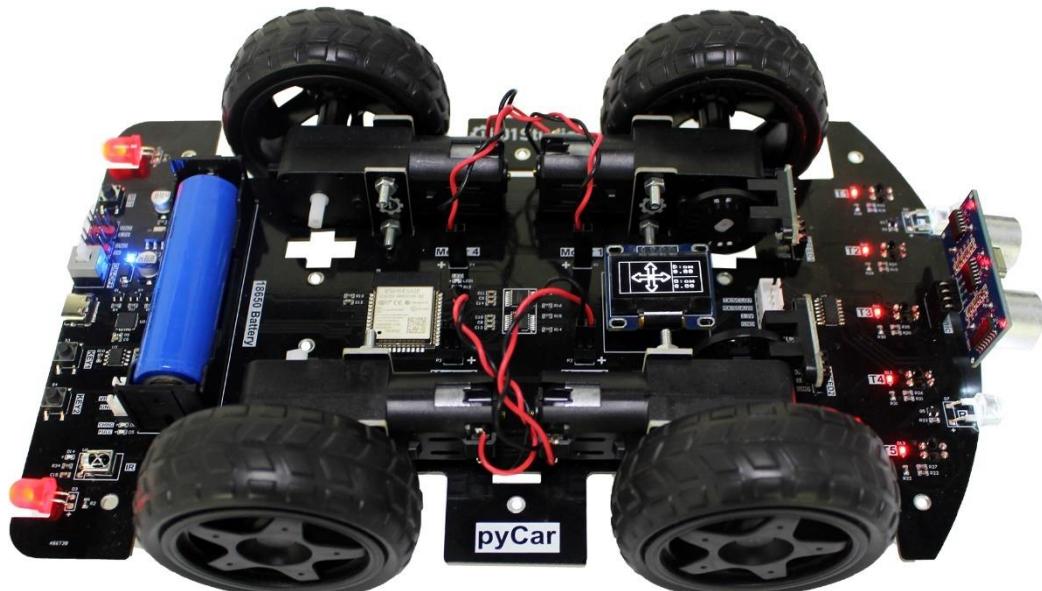


图 6-5 pyCar

- **实验目的:**

编程实现小车前进、后台、左右转向各种动作。

- **实验讲解:**

上一节我们感受到 car.py 封装后的易用性，这节还是一样，构建小车是一样的，只是调用不同的函数来实现各种动作。具体说明如下：

构造函数
<code>Car=car.CAR()</code>
构造 pyCar 对象。
使用方法
<code>Car.forward()</code>
前进。
<code>Car.backward()</code>
后退。
<code>Car.turn_left(mode=0)</code>
左转。  【mode】转动模式  0: 小幅度转动，单排轮子工作； 1: 大幅度转动，双排轮子同时工作，可实现原地旋转。
<code>Car.turn_right(mode=0)</code>
右转。  【mode】转动模式  0: 小幅度转动，单排轮子工作； 1: 大幅度转动，双排轮子同时工作，可实现原地旋转。
<code>Car.stop()</code>
停止。
更详细内容请看官方文档: <a href="https://pycar.01studio.cc/">https://pycar.01studio.cc/</a>

表 6-2 pyCar 对象

通过上表我们了解了小车的动作函数后，可以通过编程轻松实现相关功能。  
代码编写流程如下：

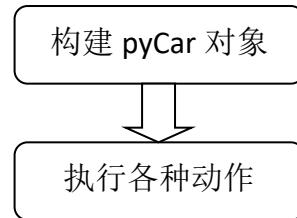


图 6-6 代码编写流程

参考代码如下：

```
...  
实验名称: pyCar 各种移动动作  
版本: v1.0  
日期: 2021.12  
作者: 01Studio  
...  
  
from car import CAR  
import time  
  
Car = CAR() #构建 pyCar 对象  
  
Car.forward() #前进  
time.sleep(1)  
  
Car.backward() #后退  
time.sleep(1)  
  
Car.turn_left() #左转  
time.sleep(1)  
  
Car.turn_right() #右转
```

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

```
Car.stop() #停止
```

- **实验结果：**

下载程序，可以看到车头灯成功点亮。

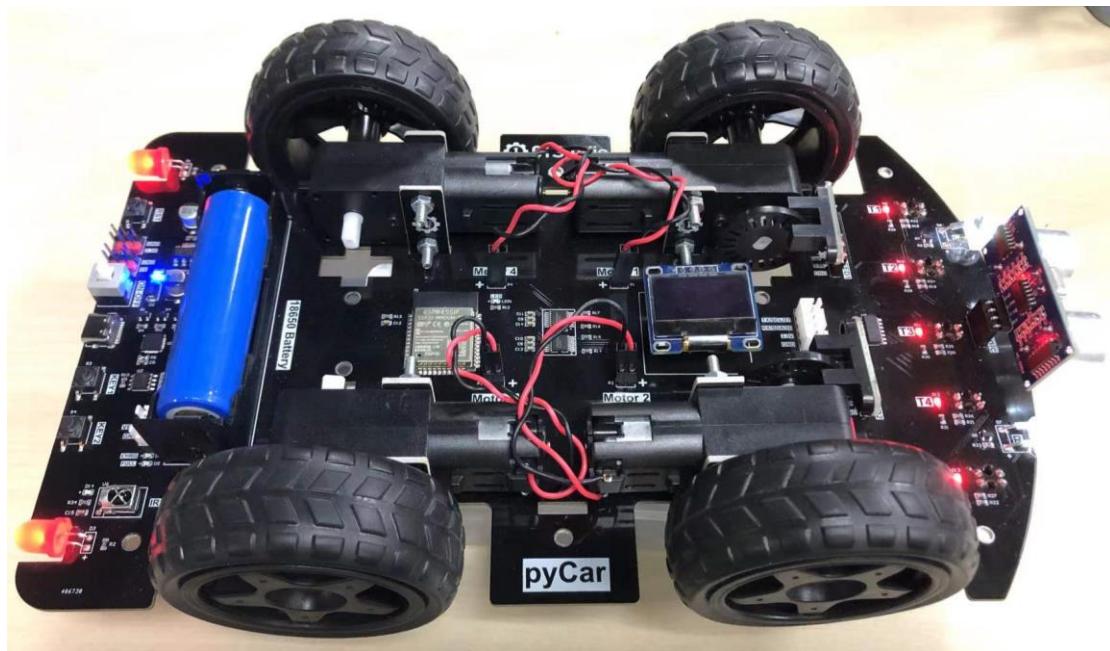


图 6-7

- **总结：**

从第一个车头灯实验不难看到，pyCar 库的封装让其变动简单易用。接下来我们将学习更多的功能应用。

## 6.3 超声波测距

- **前言：**

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

- **实验平台：**

pyCar。

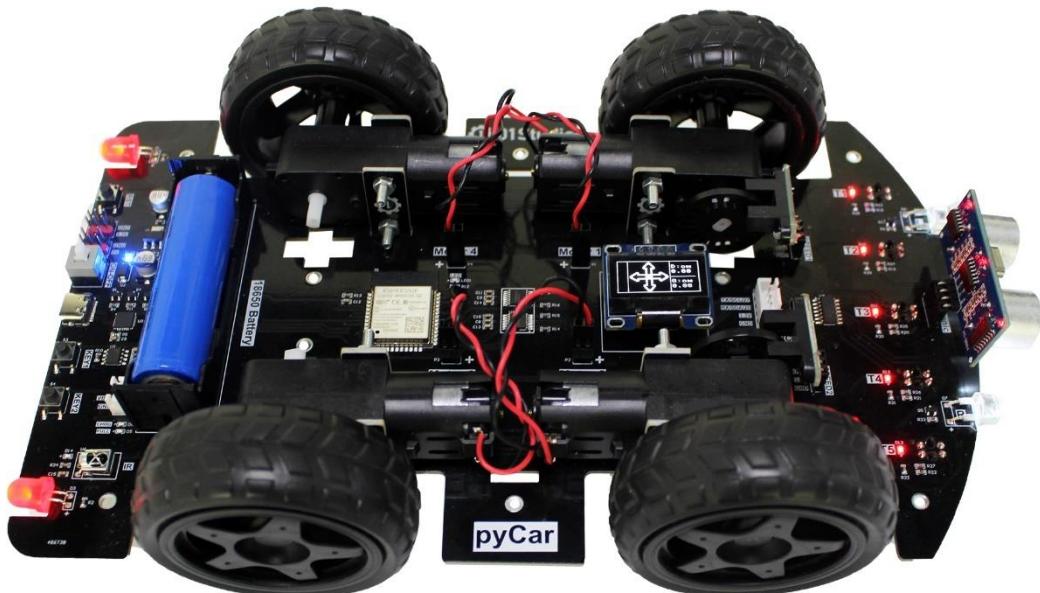


图 6-8 开发套件与传感器连接

- **实验目的：**

测量车头与障碍物之间的距离。

- **实验讲解：**

我们先来看看超声波传感器模块的介绍：



图 6-9 超声波模块

功能参数	
传感器型号	HCSR04
供电电压	3.3-5V
工作电流	<20 mA
工作温度	-20 至 85°C
接口定义	【VCC、TRIG 、 ECHO、 GND】
通讯信号	IO 数字接口
测量距离	2-450 cm
测量精度	0.5 cm

表 6-3 pyCar 超声波传感器参数

超声波传感器模块使用两个 IO 口分别控制超声波发送和接收，工作原理如下：

1. 给超声波模块接入电源和地；
2. 给脉冲触发引脚（trig）输入一个长为 20us 的高电平方波；
3. 输入方波后，模块会自动发射 8 个 40KHz 的声波，与此同时回波引脚（echo）端的电平会由 0 变为 1；（此时应该启动定时器计时）
4. 当超声波返回被模块接收到时，回波引脚端的电平会由 1 变为 0；（此时应该停止定时器计数），定时器记下的这个时间即为超声波由发射到返回的总时长；

5. 根据声音在空气中的速度为 340 米/秒，即可计算出所测的距离。

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

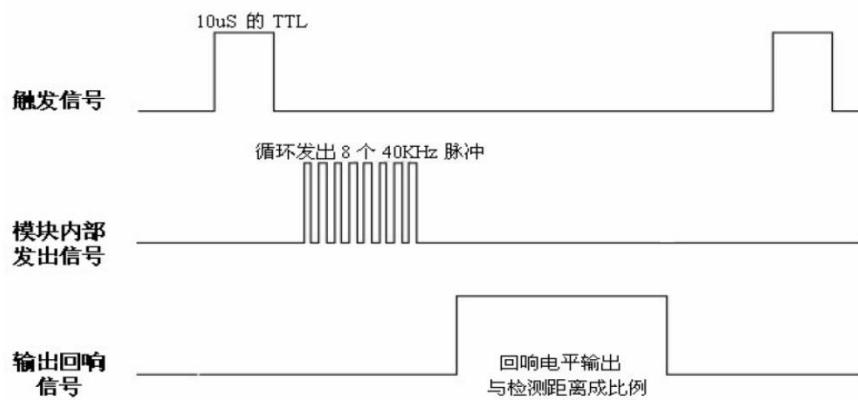


图 6-10 超声波传感器 HCSR04 时序图

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

构造函数
<code>Car=car.CAR()</code>
构造 pyCar 对象。
使用方法
<code>Car.getDistance()</code>
获取车头超声波传感器距离值，单位 cm。
更详细内容请看官方文档： <a href="https://pycar.01studio.cc/">https://pycar.01studio.cc/</a>

表 6-4 超声波传感器使用方法

从上表看到，只需要 1 行代码即可实现超声波距离值的测量，代码编写流程如下：

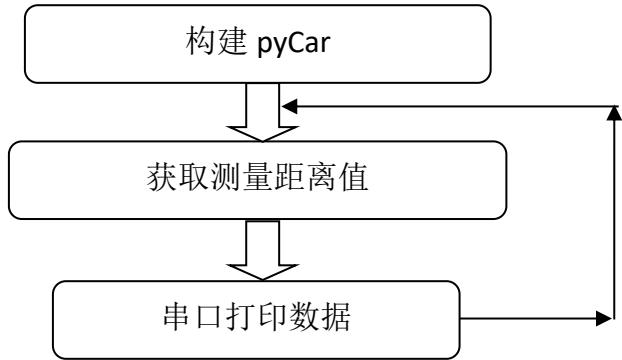


图 6-11 代码编写流程

主程序 main.py 参考代码如下：

```
...
实验名称: pyCar 超声波测距
版本: v1.0
日期: 2021.12
作者: 01Studio
...

from car import CAR
import time

aCar = CAR() #构建 pyCar 对象

while True:

    value = Car.getDistance()
    print(str(value) + ' cm')
    time.sleep(1)
```

## ● 实验结果：

用手掌或障碍物在车头的超声波传感器模块移动，可以看到 REPL 打印的距离变化：

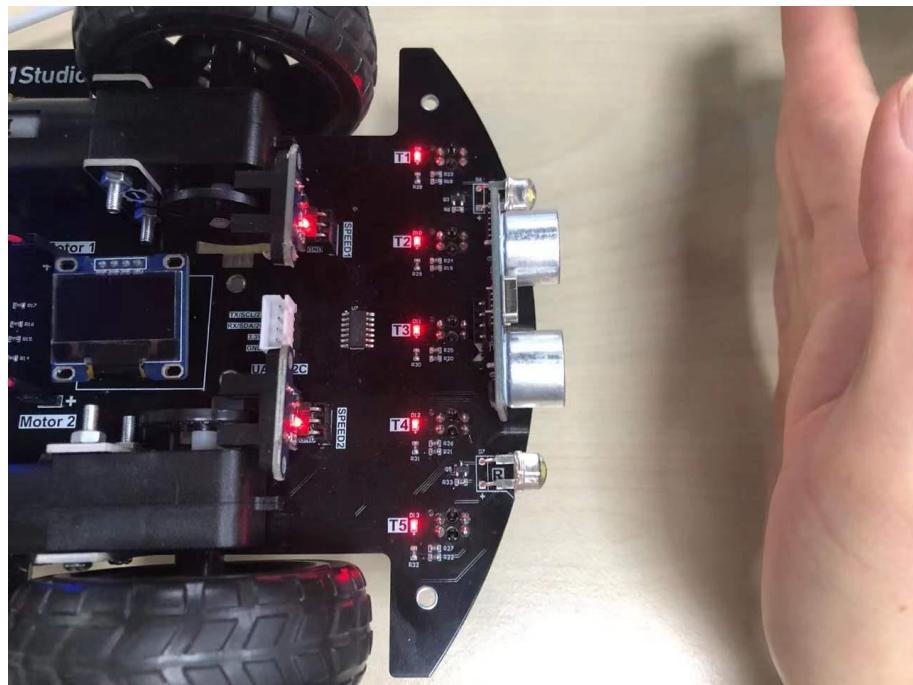


图 6-12 障碍物测试

The screenshot shows the Thonny IDE interface. On the left, there is a file browser with files like ble.py, car.py, and ssd1306.py. The main window shows the code for main.py:

```
1  #!/usr/bin/python  
2  # 实验名称: pyCar超声波测距  
3  # 版本: v1.0  
4  # 日期: 2021.12  
5  # 作者: 01Studio  
6  #  
7  
8  from car import CAR  
9  import time
```

The Shell tab at the bottom displays the output of the script, which shows distance measurements in centimeters:

```
26.35 cm  
24.718 cm  
26.35 cm  
6.035 cm  
7.446 cm  
7.089 cm  
6.63 cm  
6.562 cm  
40.613 cm
```

A red arrow points to the last line of the shell output, "40.613 cm".

图 6-13 实验结果

### ● 总结：

通过 `micropython` 库模块非常简单就实现了对超声波传感器测距的应用。这让我们再一次感受到了 `MicroPython` 的魅力。赶快动手制作自己的避障小车和其他好玩的创作吧。

## 6.4 码盘测路程

- **前言:**

本节来学习一下如何使用小车上的码盘测试小车行驶路程。

- **实验平台:**

pyCar。

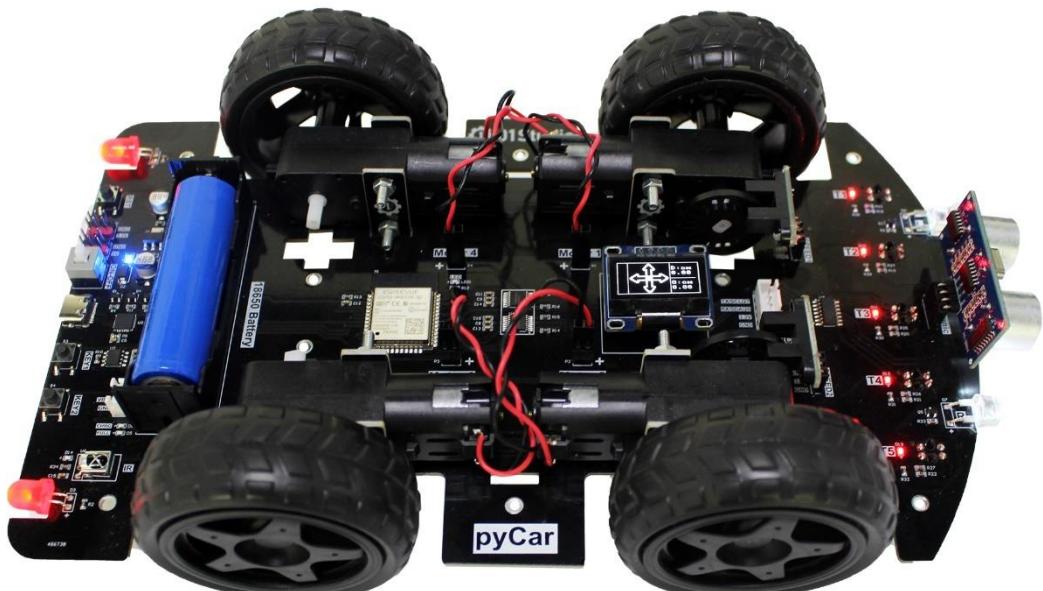


图 6-14 开发套件与传感器连接

- **实验目的:**

测量 pyCar 行驶路程。

- **实验讲解:**

我们先来看看码盘测速原理，pyCar 前轮 2 路电机安装了编码盘和配套的光电测速模块。

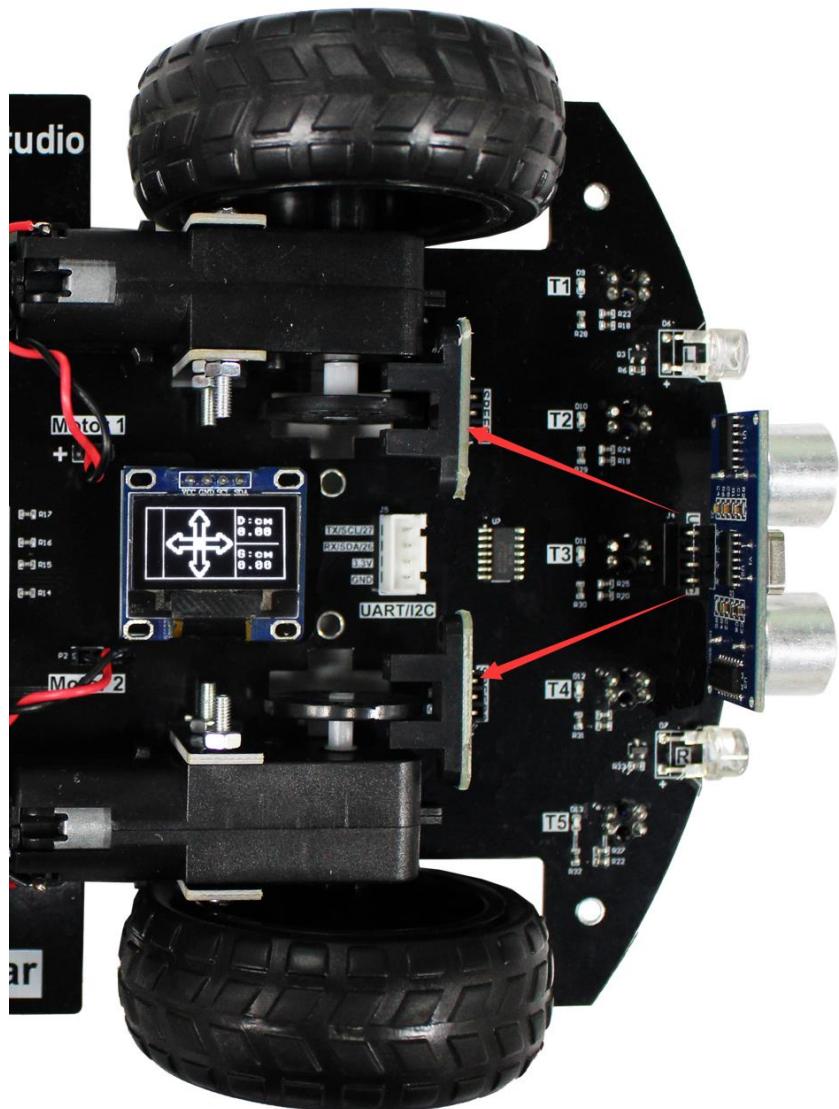


图 6-15

码盘的原理很简单，硬件看下图，就是一个个等距离的孔，电机带动码盘转动，光电对管便产生一系列方波，通过计算方波信号的数量便可知道码盘转动了多少圈，再结合车轮的直径，便可计算路程。

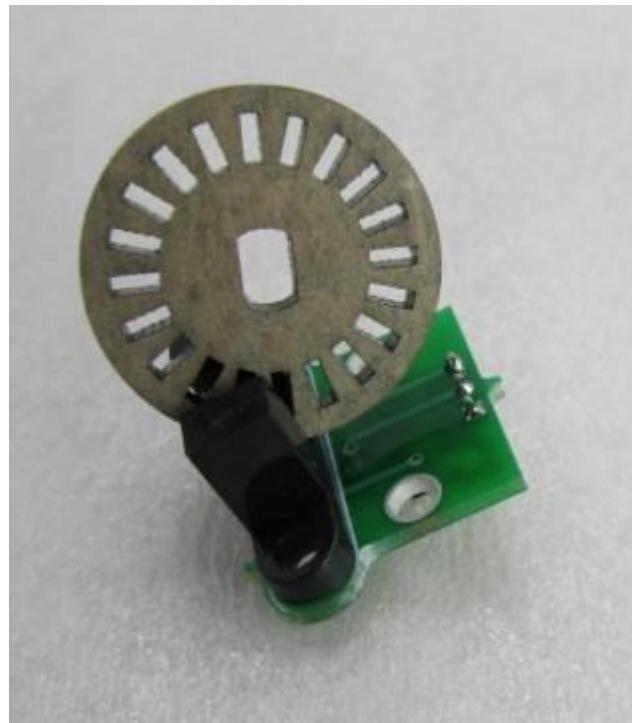


图 6-16 码盘和光电对管

然而计算方法已经集成在 car.py 文件，如想了解代码原理可以打开 car.py 文件查看，使用方法如下：

构造函数
<code>Car=car.CAR()</code>
构造 pyCar 对象。
使用方法
<code>Car.getJourney()</code>
返回行驶路程，单位 m。
更详细内容请看官方文档： <a href="https://pycar.01studio.cc/">https://pycar.01studio.cc/</a>

表 6-5 路程测量库函数

从上表看到，只需要 1 行代码即可实现行驶路程值的测量，代码编写流程如下：

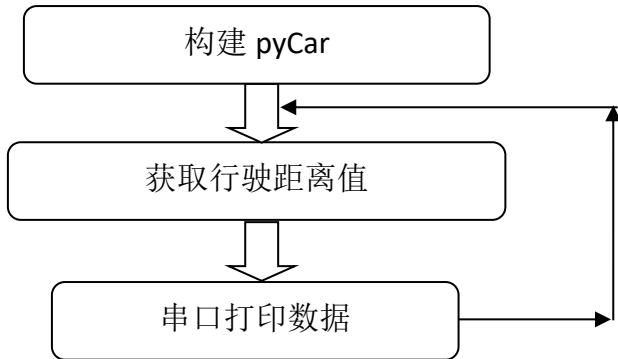


图 6-17 代码编写流程

主程序 main.py 参考代码如下：

```
...
实验名称: pyCar 行驶路程
版本: v1.0
日期: 2021.12
作者: 01Studio
...

from car import CAR
import time

Car = CAR() #构建 pyCar 对象

while True:

    value = Car.getJourney()
    print(str(value) + ' m')
    time.sleep(1)
```

## ● 实验结果：

运行代码，转动前轮，可以看到行驶路程的变化：

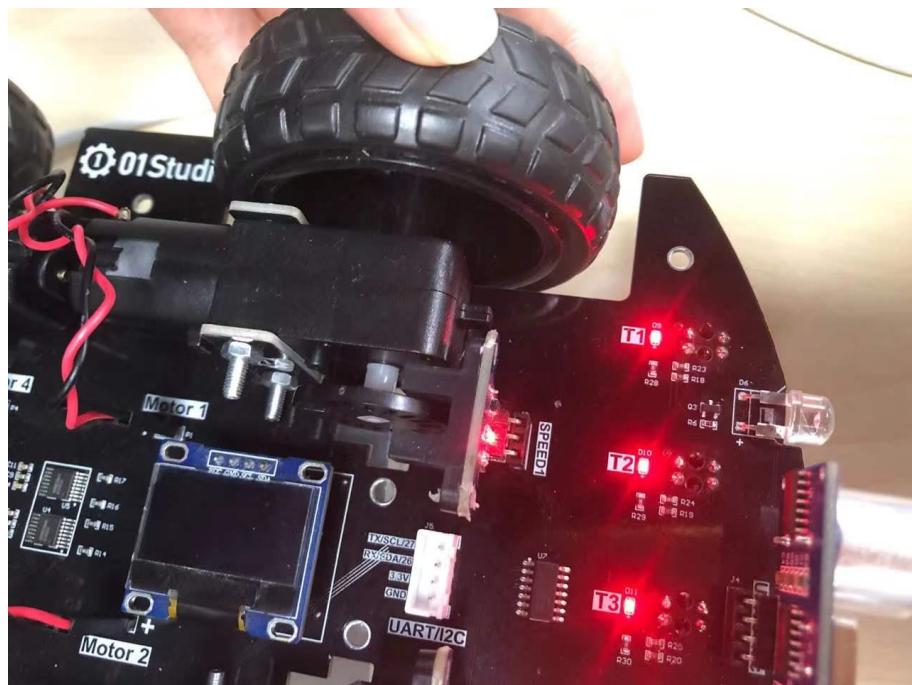


图 6-18 障碍物测试

```
实验名称: pyCar行驶路程
版本: v1.0
日期: 2021.12
作者: 01Studio
...
from car import CAR
import time

('0.08', '0.00') m
('0.15', '0.00') m
('0.21', '0.00') m
('0.25', '0.00') m
('0.33', '0.00') m
('0.40', '0.00') m
('0.45', '0.00') m
('0.53', '0.00') m
('0.56', '0.00') m
```

图 6-19 实验结果

- **总结：**

通过 `micropython` 库模块非常简单就实现了行驶路程的应用。接口返回的是 2 路轮子行驶路程值，实际使用过程中只需要使用其中一路或者将两路值求平均即可。

## 6.5 红外遥控器

- **前言:**

本节来学习一下如何使用小车接收红外遥控器数据。

- **实验平台:**

pyCar。

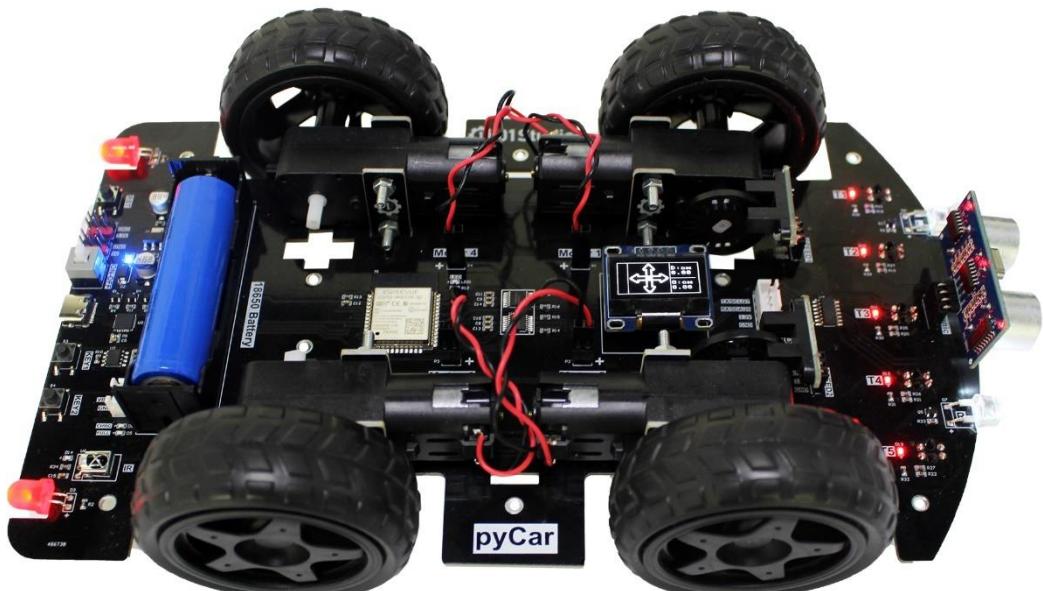


图 6-20 开发套件与传感器连接

- **实验目的:**

获取红外遥控器发出的编码。

- **实验讲解:**

pyCar 的右后轮位置集成了红外一体接收头，可以接收市面上大部分的红外遥控器发出的信号。

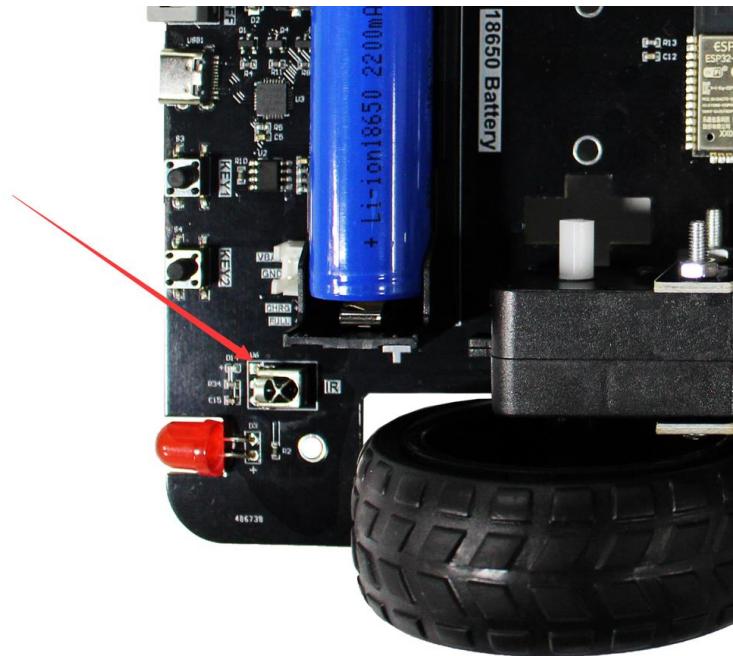


图 6-21

我们可以使用配套的红外遥控器进行本实验，遥控器每个按键对应的码如下图，16 进制。



图 6-22 红外遥控器

如果用 C 来开发，需要使用指定时序读取红外信号，然后再解码。编程会非常繁琐，而使用 `micropython` 库编程，直接调用相关函数即可（其实也少不了解码的步骤，只是已经封装好了库，我们直接使用即可），使用方法如下：

构造函数
<code>Car=car.CAR()</code>
构造 <code>pyCar</code> 对象。
使用方法
<code>Car.getIR()</code>
返回红外解码值，连续按下的情况会在返回 1 次值后一直返回 ‘REPEAT’ 字符。当没有信号时候返回 <code>None</code> 。
更详细内容请看官方文档： <a href="https://pycar.01studio.cc/">https://pycar.01studio.cc/</a>

表 6-6 红外解码

红外解码的使用方法也非常简单，编程流程如下：

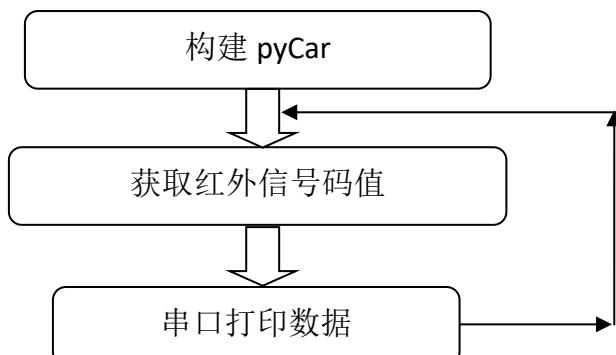


图 6-23 代码编写流程

主程序 `main.py` 参考代码如下：

```
...
实验名称: pyCar 红外遥控解码
版本: v1.0
日期: 2021.12
作者: 01Studio
```

```
'''  
  
from car import CAR  
  
import time  
  
  
Car = CAR() #构建 pyCar 对象  
  
  
while True:  
  
    value = Car.getIR()  
  
    if value != None:  
  
        print(value)
```

### ● 实验结果：

运行代码，用遥控器对着 pyCar 红外接收头按下按键，可以看到红外接收头旁边的指示灯闪亮，表示有接收到信号：

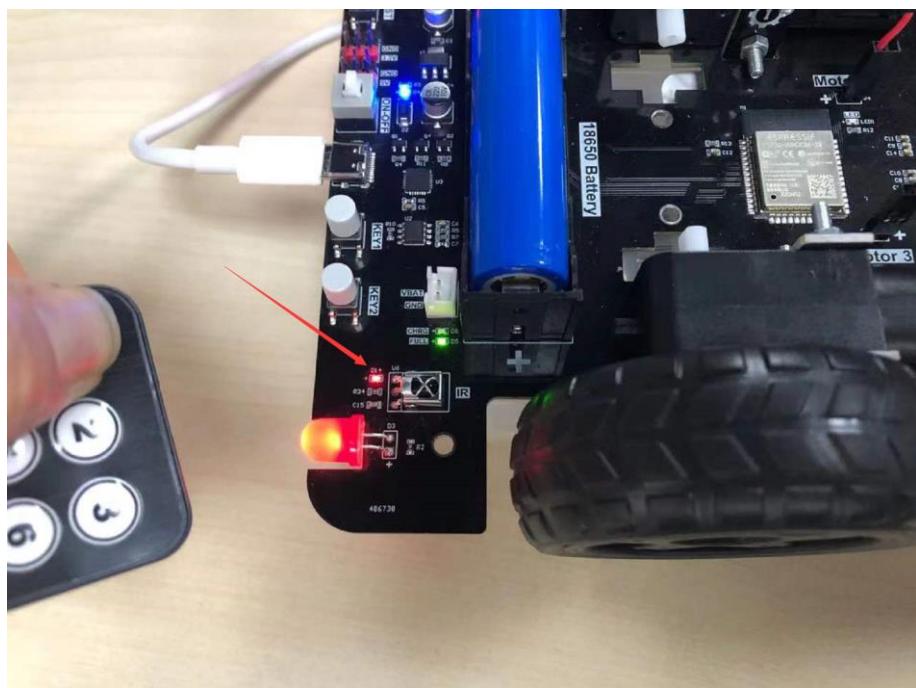


图 6-24 障碍物测试

The screenshot shows the Thonny IDE interface. On the left, there's a file browser with files like 'main.py', 'ble.py', 'car.py', and 'ssd1306.py'. Below it is a 'MicroPython设备' (MicroPython Device) list with the same four files. In the center, there's a code editor window titled 'main.py' containing the following Python code:

```
11 Car = CAR() #构建pyCar对象
12
13 while True:
14
15     value = Car.getIR()
16     if value != None:
17         print(value)
18
19
```

To the right of the code editor is a 'Shell' window showing the output of the script. The output consists of several '69' characters, followed by the word 'REPEAT', then another '69', and finally another 'REPEAT' with a red arrow pointing to it. The entire interface is labeled 'MicroPython (ESP32)' at the bottom.

图 6-25 实验结果

### ● 总结：

学习了本节后，便可通过检测红外按键来控制小车各项功能，在后面综合实验中会涉及红外遥控车相关内容。

## 第7章 综合实验

在前面学习了 MicroPython 内容后，相信我们已经对其编程方法有了一定的认识，那么除了较为简单的基础实验和 pyCar 子模块实验外，我们可以将例程代码适当整合，实现小车的综合应用实验。

## 7.1 红外遥控车

- 前言：

我们在前面学习了小车的移动动作和红外遥控器解码实验，本节我们就来整合一下，打造一辆红外遥控小车！

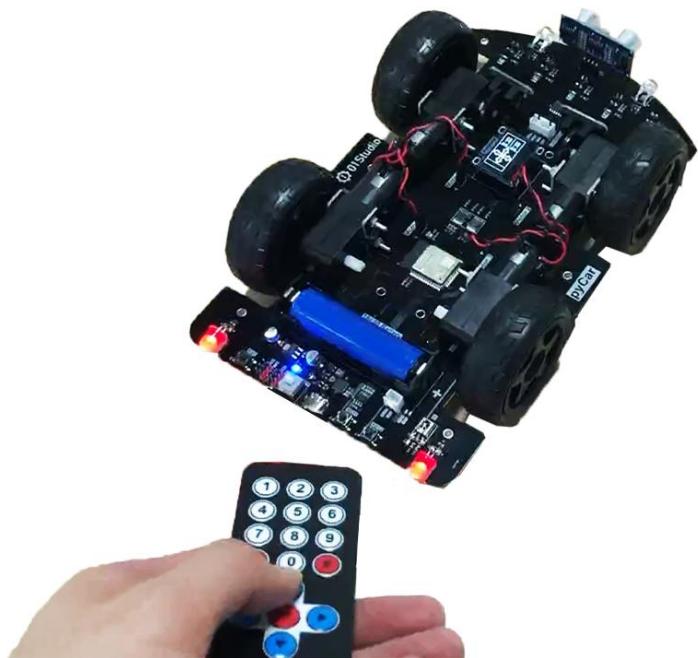


图 7-1 红外遥控车

- 实验平台：

pyCar。



图 7-2 pyCar

● 实验目的：

编程实现红外遥控器控制小车的各种移动。

● 实验讲解：

关于小车移动和红外遥控实验前面章节已经有详细讲解，这里不再重复，具体参考：

移动动作参阅：[6.2 动作](#) 章节内容；

红外遥控器参阅：[6.5 红外遥控器](#) 章节内容。

本节我们使用遥控器的上下左右和 OK 键，分别代表 pyCar 的前进、后台、左转、右转已经开关车头灯。

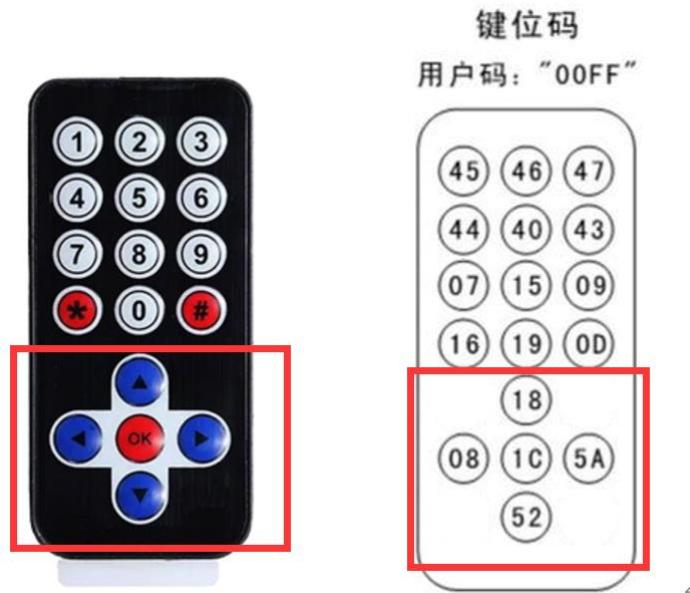


图 7-3 红外遥控码表

有了前面学习的积累，可以通过编程轻松实现相关功能。代码编写流程如下：

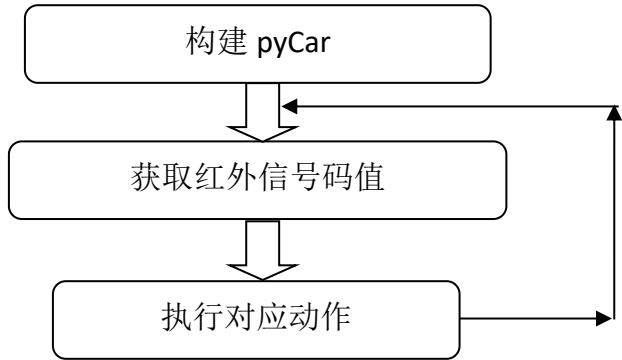


图 7-4 代码编写流程

参考代码如下：

```

...
实验名称: 红外遥控车
版本: v1.0
日期: 2021.12
作者: CaptainJackey
说明: 主函数文件改成 main.py 发送到 pyCar 可离线运行
...

#导入相关模块
from car import CAR
import time

#初始化 pyCar
Car = CAR()
time.sleep_ms(300) #等待稳定

#车灯状态
light_state = 0

while True:

```

```
key = Car.getIR() #读取红外传感器

if key != None: #有按键按下

    #按键上， 前进
    if key == 0x18:

        Car.forward()
        time.sleep(1)

        Car.stop()

    #按键下， 后退
    if key == 0x52:

        Car.backward()
        time.sleep(1)

        Car.stop()

    #按键左， 左转
    if key == 0x8:

        Car.turn_left(mode=1)
        time.sleep_ms(250)

        Car.stop()

    #按键右， 右转
    if key == 0x5A:

        Car.turn_right(mode=1)
        time.sleep_ms(250)
```

```
Car.stop()
```

```
#按键OK，车灯开关  
if key == 0x1C:  
  
    light_state = not light_state  
    Car.light(light_state)
```

- **实验结果：**

下载程序，通过遥控器便可控制小车各种动作。

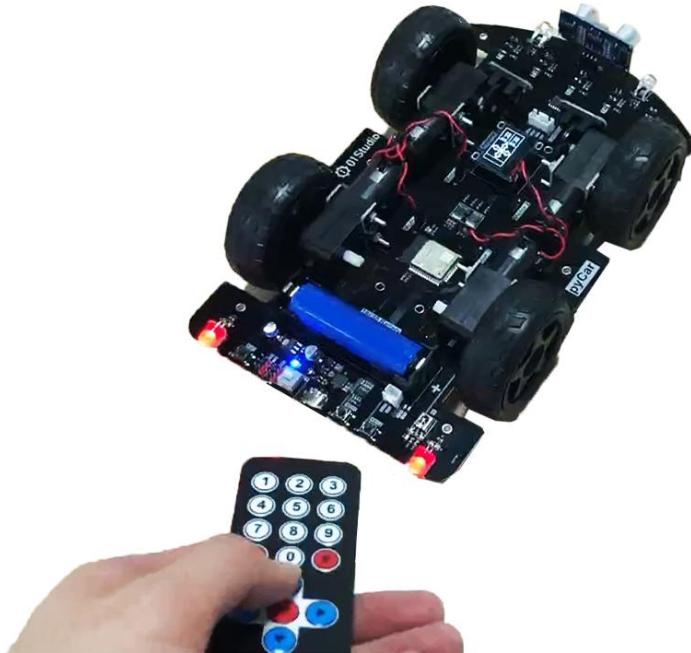


图 7-5

- **总结：**

使用 micropython 结合 car.py 库非常轻松就打造一台红外遥控车，给家里的小朋友玩，估计可以玩一整天。有需要的用户可以自行编程，使用更多的按键实现更多的功能。

## 7.2 避障小车

- 前言：

想知道你的扫地机器人是怎么避开障碍物么？当它遇到墙、桌子会自动转向，pyCar 车头配备了超声波传感器，也可以实现这个功能。

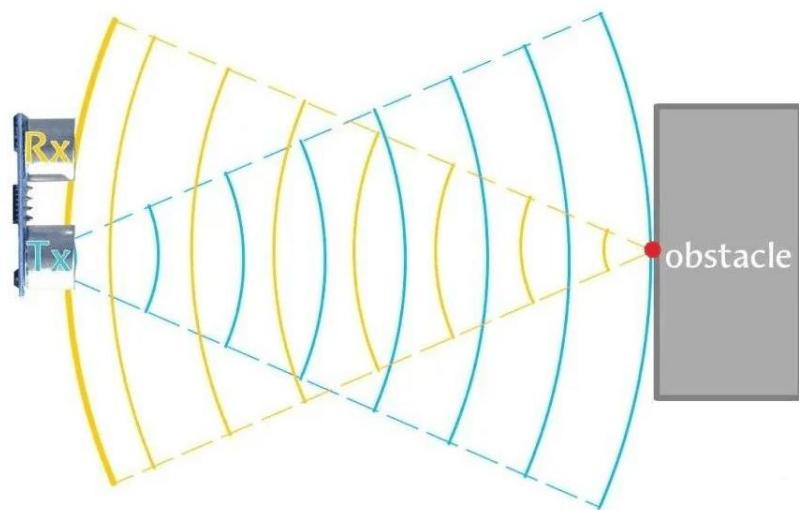


图 7-6 超声波避障

- 实验平台：

pyCar。



图 7-7 pyCar

### ● 实验目的:

编程实现小车行驶避障。

### ● 实验讲解:

关于小车移动和超声波实验前面章节已经有详细讲解，这里不再重复，具体参考：

移动动作参阅：[6.2 动作](#) 章节内容；

红外遥控器参阅：[6.3 超声波测距](#) 章节内容。

本节我们使用一个最简单的控制方式，那就是当小车实时检测超声波距离值，当距离少于某个设定值时，小车转向，大于此设定值则正常行走。

有了前面学习的积累，可以通过编程轻松实现相关功能。代码编写流程如下：

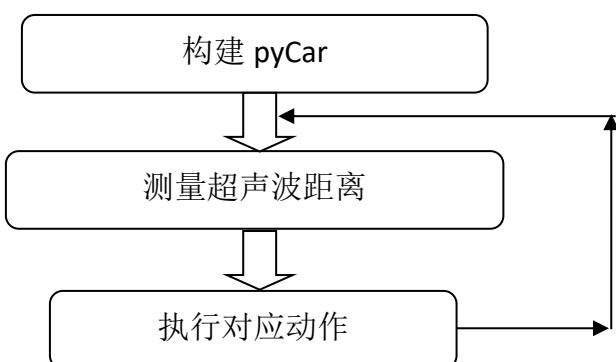


图 7-8 代码编写流程

参考代码如下：

```
...  
实验名称: pyCar 超声波避障  
版本: v1.0  
日期: 2021.12  
作者: CaptainJackey  
说明: 将文件改成 main.py 发送到 pyCar 可离线运行
```

```
'''  
  
#导入相关模块  
  
from car import CAR  
  
import time,random  
  
  
#初始化 pyCar  
  
Car = CAR()  
  
time.sleep_ms(300) #等待稳定  
  
  
turn_node = 1  
  
  
while True:  
  
  
  
  
    #距离少于 50cm 就转弯  
  
    if 0 < int(Car.getDistance()) < 50 :  
  
  
        #从 0 和 1 中随机生成一个数决定左右转向  
  
        if turn_node == 1:  
  
  
            turn_direct = random.randint(0,1)  
            turn_node = 0  
  
  
        #1 右转， 0 左转。  
  
        if turn_direct:  
  
  
            Car.turn_right(mode=1)  
  
  
        else:  
            Car.turn_left(mode=1)
```

```
Car.turn_left(mode=1)

else: #走直线

    Car.forward()
    turn_node = 1

time.sleep_ms(50) #适当延时调整响应速度
```

上述代码加入了一个随机转向设置，实现小车遇到障碍物时候随机左转或右转。

### ● 实验结果：

将资料包的代码发送至 pyCar 文件系统，便可实现上电运行小车避障功能。

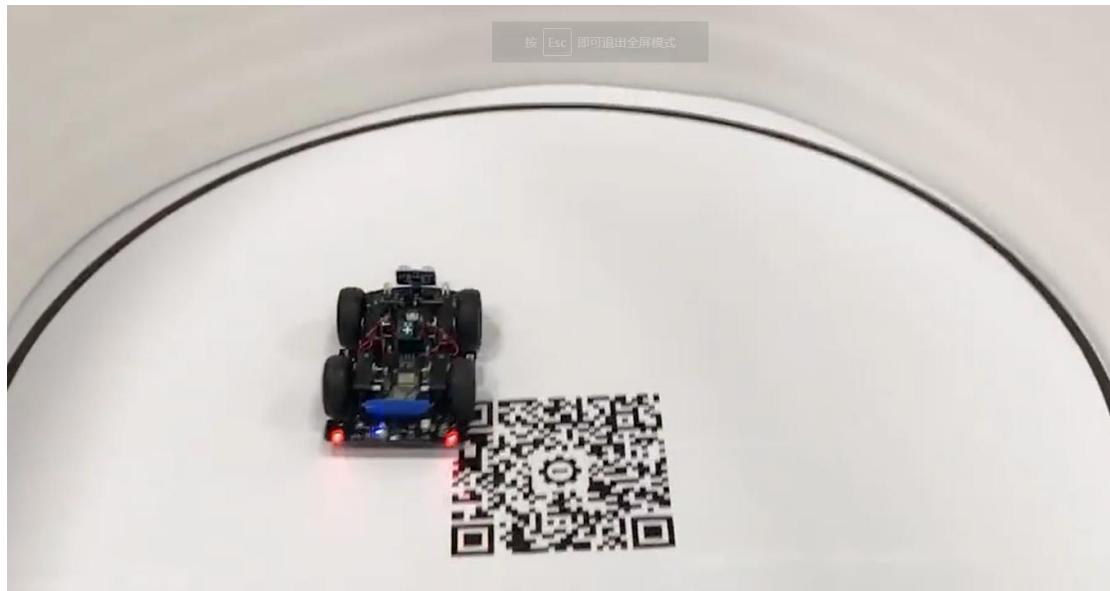


图 7-9

### ● 总结：

小车避障的原理非常简单，有兴趣的用户可以调整代码中的参数，打造一台适合自己的避障小车。

### 7.3 巡线小车

- 前言：

相信大家都有在网上看到过可以跟着黑线行走的小车，也就是我们说的巡线小车，本节我们就实验 pyCar 编程实现巡线行驶。



图 7-10 巡线小车

- 实验平台：

pyCar。

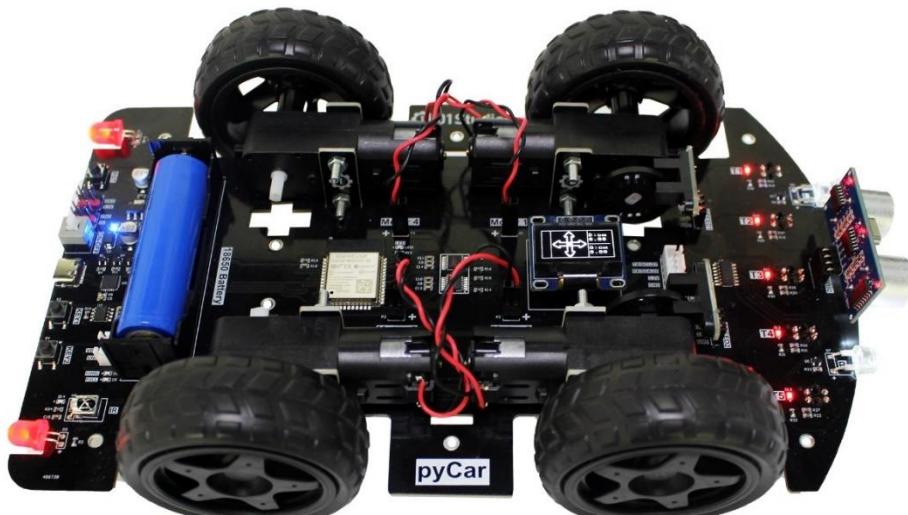


图 7-11 pyCar

- 实验目的:

编程实现小车巡线行驶。

- 实验讲解:

pyCar 的车头有 5 路光电传感器，当车在白色路面行驶时（反光性比较好），全部指示灯熄灭，输出高电平。当遇到黑色赛道（无反光），对应的传感器指示灯亮，输出低电平。

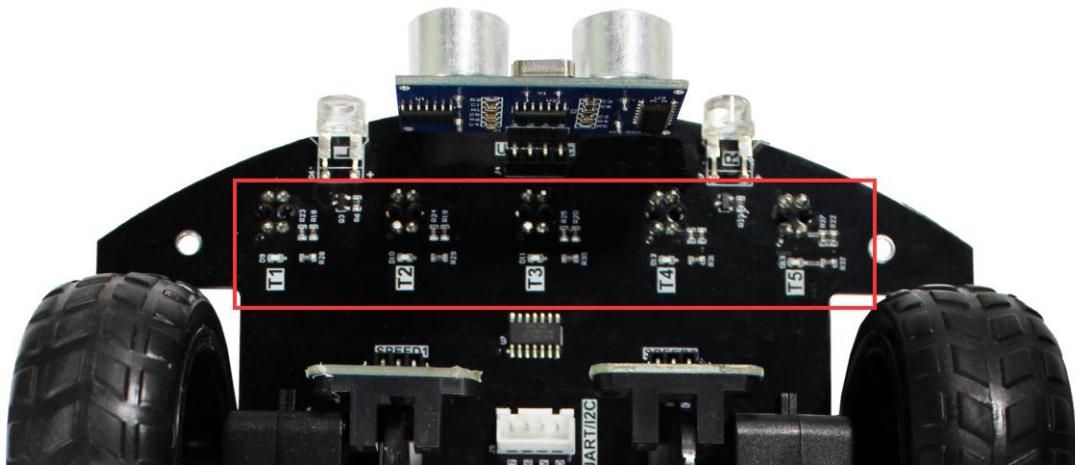


图 7-12 五路光电传感器

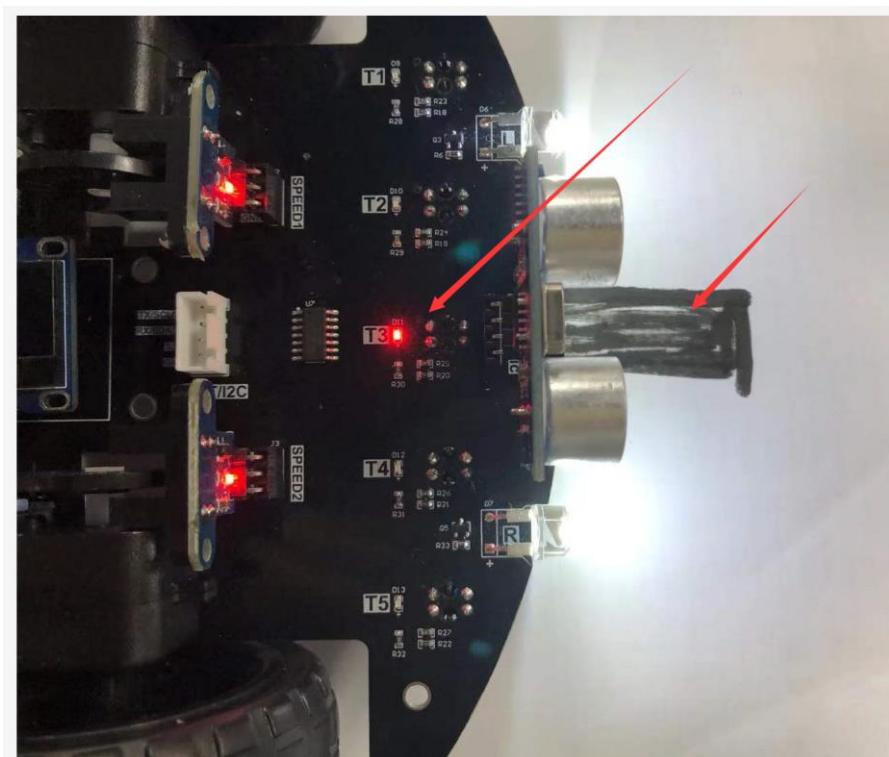


图 7-13 检测黑色线

因此我们可以根据五路光电传感器的输出值来判断小车的位置，从而通过转向让小车始终跟着白线行驶。光电传感器的使用方法如下：

构造函数	
Car=car.CAR()	
构造 pyCar 对象。	
使用方法	
Car. T1()	
返回光电传感器 T1 值，布尔类型。	
Car. T2()	
返回光电传感器 T2 值，布尔类型。	
Car. T3()	
返回光电传感器 T3 值，布尔类型。	
Car. T4()	
返回光电传感器 T4 值，布尔类型。	
Car. T5()	
返回光电传感器 T5 值，布尔类型。	
更详细内容请看官方文档： <a href="https://pycar.01studio.cc/">https://pycar.01studio.cc/</a>	

表 7-1 超声波传感器使用方法

正常情况小车走直线，当小车偏左时候右转调整，当小车偏右时候左转调整，代码编写流程如下：

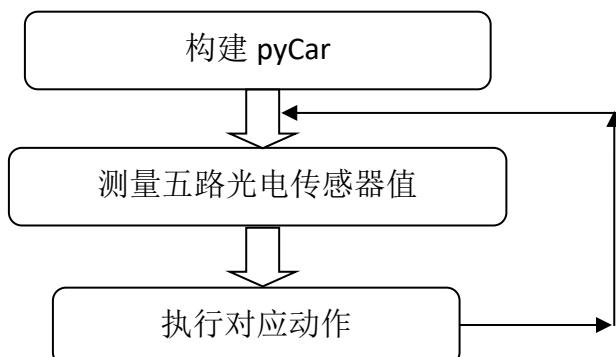


图 7-14 代码编写流程

参考代码如下：

```
...
实验名称: pyCar 巡线例程
版本: v1.0
日期: 2021-11
作者: CaptainJackey
说明: 使用 pyCar 的光电传感器实现黑线巡线
...

#导入相关模块
from car import CAR
import time

#初始化 pyCar
Car = CAR()
time.sleep_ms(300) #等待稳定

...
#####巡线检测#####
T1-T2-T3-T4-T5 光电传感器
11011 直线
10111 偏右, 往左调整
01111 严重偏右, 往左调整
11101 偏左, 往右调整
11110 严重偏左, 往右调整
...
while True:
    #偏右, 往左调整
```

```

if Car.T1()==0 or Car.T2()==0:

    time.sleep_ms(10) #消抖

    if Car.T1()==0 or Car.T2()==0 :

        Car.turn_left()

        while Car.T4()==1 and Car.T5()==1:

            if Car.T3() == 0:

                time.sleep_ms(10)

                if Car.T3() == 0:

                    break

            Car.forward()

#偏左，往右调整

elif Car.T4()==0 or Car.T5()==0 :

    time.sleep_ms(10)

    if Car.T4()==0 or Car.T5()==0 :

        Car.turn_right()

        while Car.T1()==1 and Car.T2()==1:

            if Car.T3() == 0:

                time.sleep_ms(10)

                if Car.T3() == 0:

                    break

            Car.forward()

#普通情况走直线

else:

    Car.forward()

```

- 实验结果：

将示例程序的代码发送至 pyCar 文件系统，便可实现上电运行小车巡线功能。



图 7-15

- 总结：

小车巡线原理简单，有兴趣用户可以深入研究，实现更快的行驶速度。

# MicroBit 从0到1

用方块开始你的编程之旅

01Studio团队 编著



01Studio  
-让编程变得简单有趣-

# MicroPython 从0到1

用python做嵌入式编程

(基于pyBoard STM32F405平台)

01Studio团队 编著



01Studio  
-让编程变得简单有趣-

# 树莓派从0到1

(基于树莓派4B平台)

01Studio团队 编著



01Studio  
-让编程变得简单有趣-

# 基于pyBoard STM32F405平台

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



## 基于ESP8266平台

### 基于ESP32平台

### 基于NRF62840平台

### 基于OpenMV4平台

### 基于K210平台

### 基于哥伦布 STM32F407平台



关注公众号，免费下载