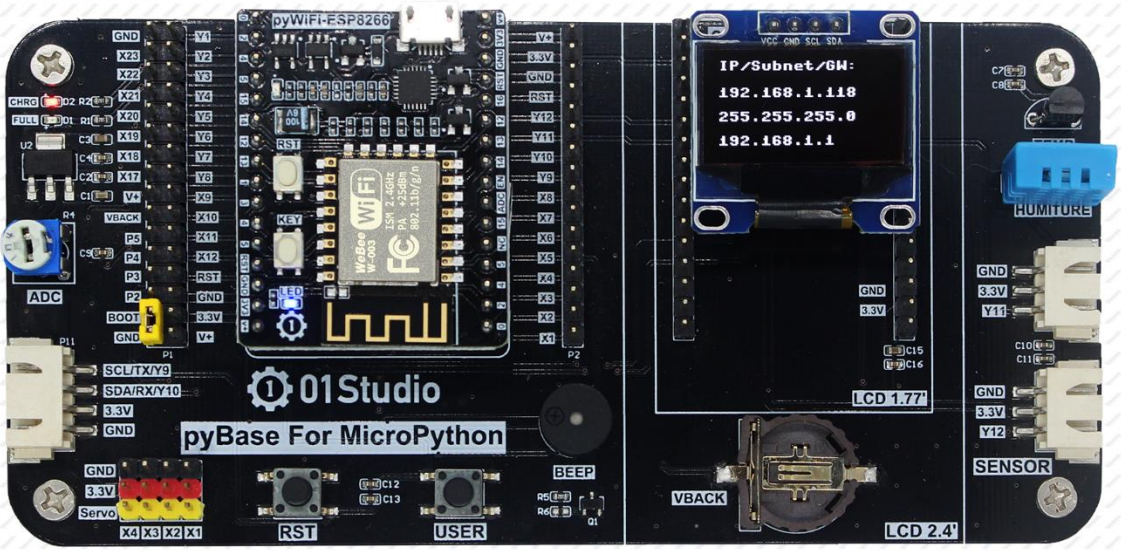
MicroPython 从0到1

用python做嵌入式编程

(基于ESP8266平台)

01Studio团队 编著

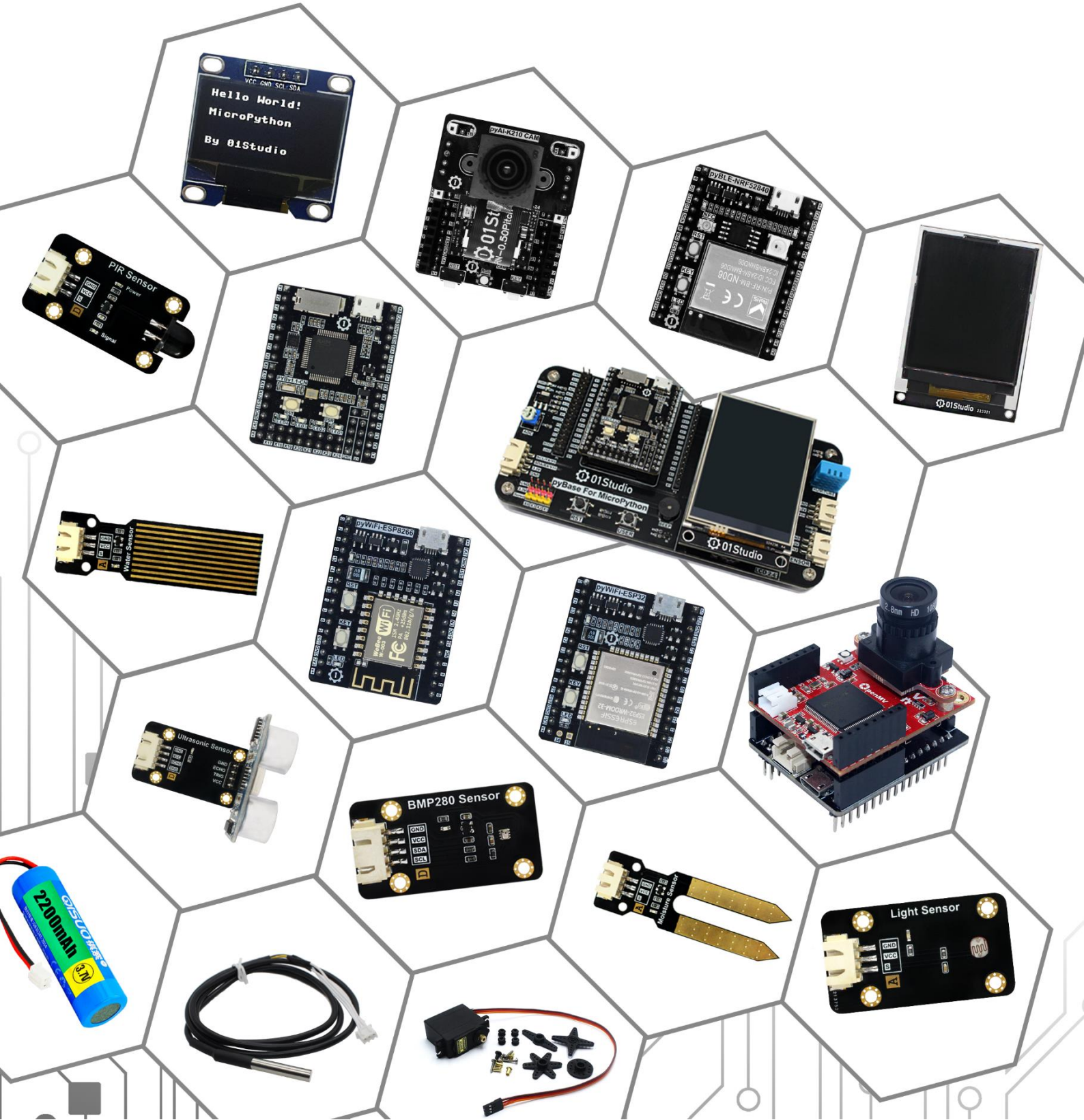


01Studio

-让编程变得简单有趣-

MicroPython

产品家族



前言

为什么学习 MicroPython？

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

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

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

为什么要写《MicroPython 从 0 到 1》教程？

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

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

为什么要打造 MicroPython 学习套件？

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

以用它来完成你的比赛或项目。

为什么要打造 01Studio 社区论坛？

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

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

期待你加入 01Studio 社区大家庭，成为我们的一份子，一起成长。

【社区链接：www.01Studio.org】

【微信公众号：01Studio 社区】

01Studio

2019.4 于深圳

版本说明

版本	日期	作者	更新内容
v1.0	2021-2-5	Jackey	1.第 1 版（重构）

版权声明

目录

第一部分　MicroPython 基础知识

第1章　MicroPython 简介

1.1　MicroPython 是什么

第一次接触 MicroPyhon 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。

MicroPyhon 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，Micropyton 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 ESP8266 平台来进行编写。

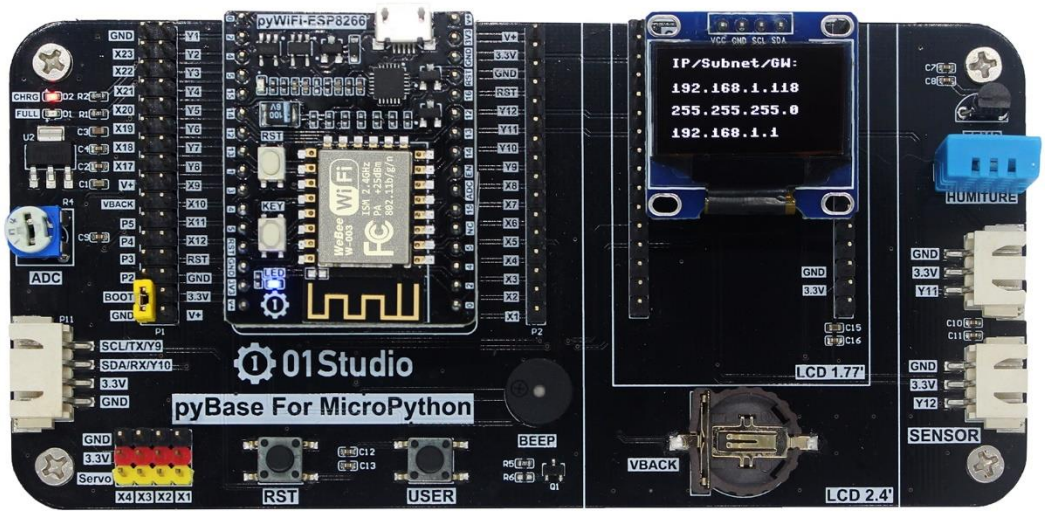


图 1-1 pyWiFi-ESP8266 开发套件

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

以下是 01Studio 其它 micropython 开发平台：

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

图 1-2 01Studio MicroPython 系列开发平台

1.3 MicroPython 相关学习资料

1.3.1 01Studio 社区

【社区网址：www.01Studio.org】

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。

图 1-3 01Studio 社区

1.3.2 MicroPython 文档（中文）

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

MicroPython 文档（中文） 【网址：docs.micropython.01studio.org】

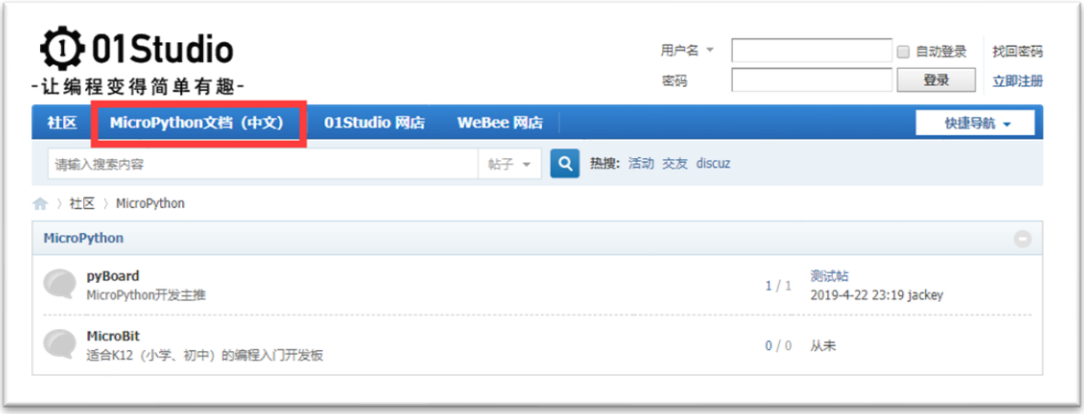


图 1-4 文档在社区的位置

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。

图 1-5 MciroPython 文档（中文）

1.3.3 MicroPython 官方网站

【网址：www.micropython.org】

英文版官网有官方文档（DOCS）和英文论坛，适合比较英语比较好的小伙伴。

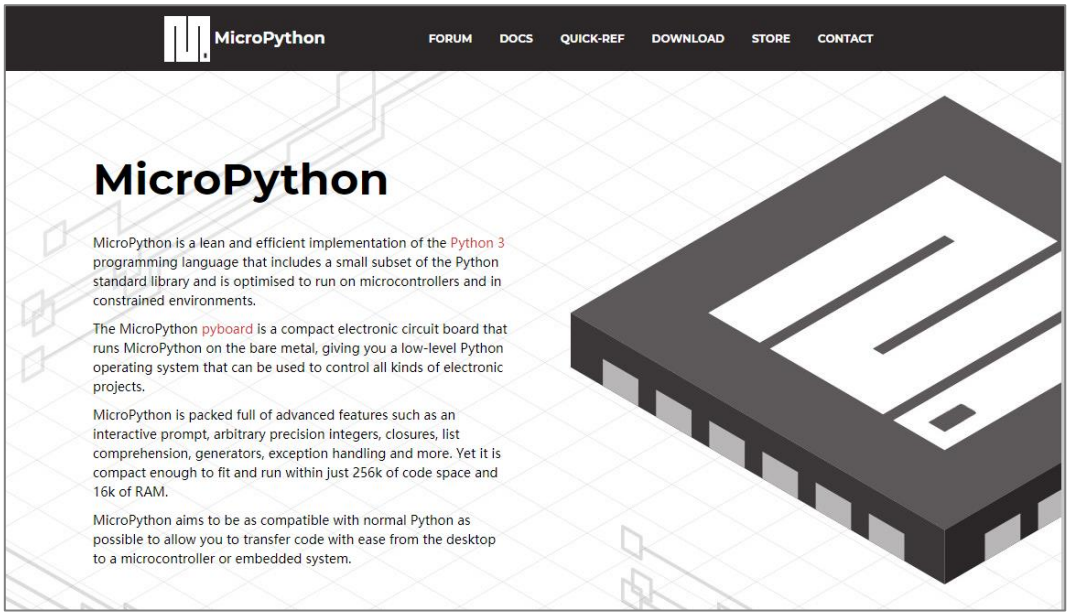


图 1-6 MicroPython 官网

1.4 MicroPython 开发套件介绍

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

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

1.4.1 ESP8266 平台

1.4.1.1 pyWiFi-ESP8266

pyWiFi-ESP8266 是由 01Studio 设计研发，基于 ESP8266 平台，接口兼容 MicroPython 的 pyBoard，主要特点如下：

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

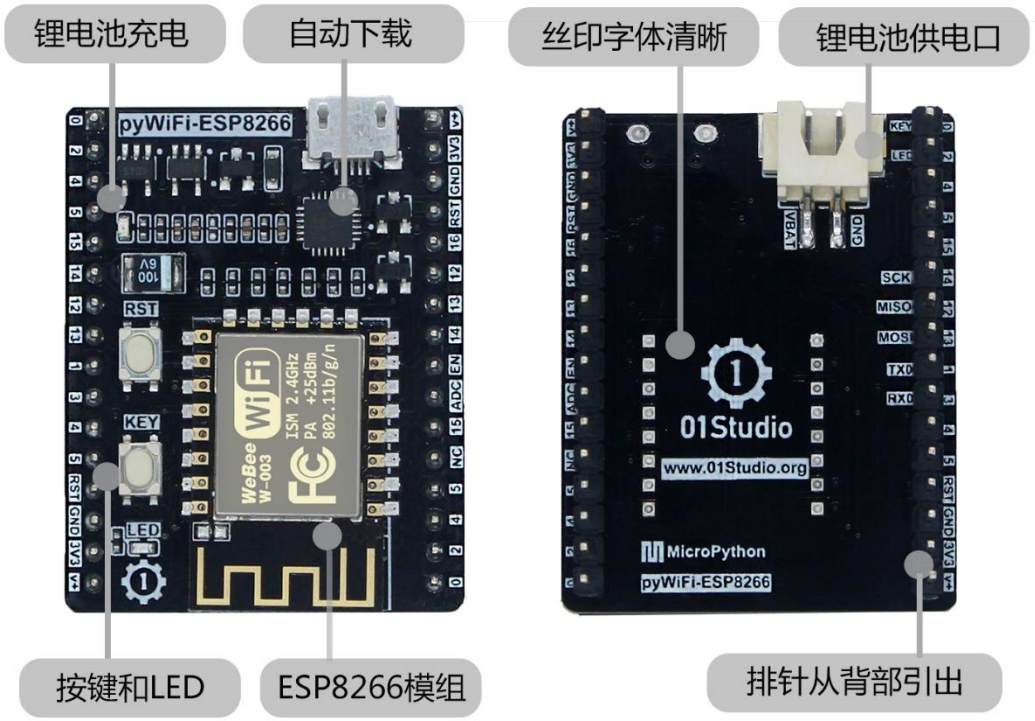


图 1-7 pyWiFi-ESP8266

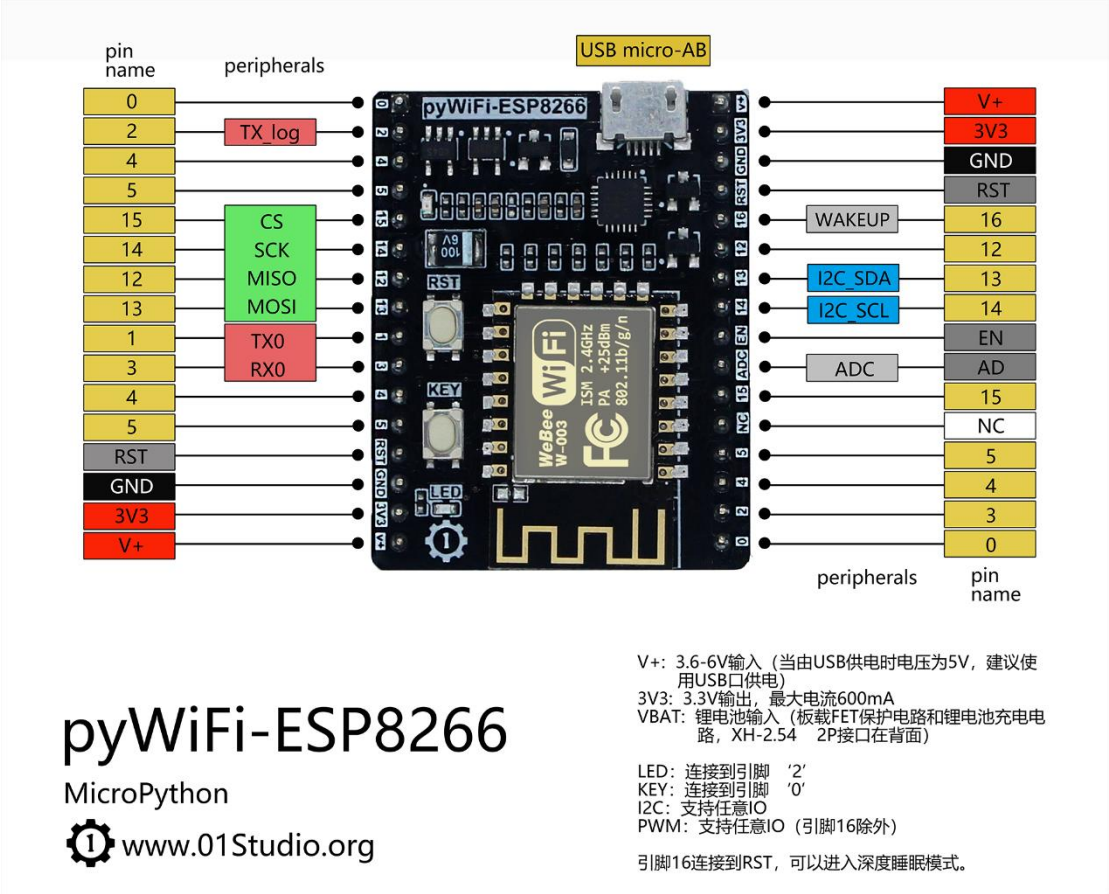


图 1-8 pyWiFi-ESP8266 引脚图

功能参数	
V+	3.6v – 6v 输入（当插入 USB 时候由 USB 供电，电压为 5V，建议使用 USB 口供电）
3V3	3.3v 输出，最大电流 600mA
VBAT	锂电池输入（板载 FET 保护电路和锂电池充电电路，XH-2.54 2P 接口在背面）
LED	连接到引脚 2
KEY	连接到引脚 0
I2C	支持任意 IO
PWM	支持任意 IO（引脚 16 除外）
SLEEP	引脚连接到 RST，可以进入深度睡眠唤醒模式

表 1-1 pyWiFi-ESP8266 功能参数表

1.4.1.2 pyWiFi-ESP8266 开发套件

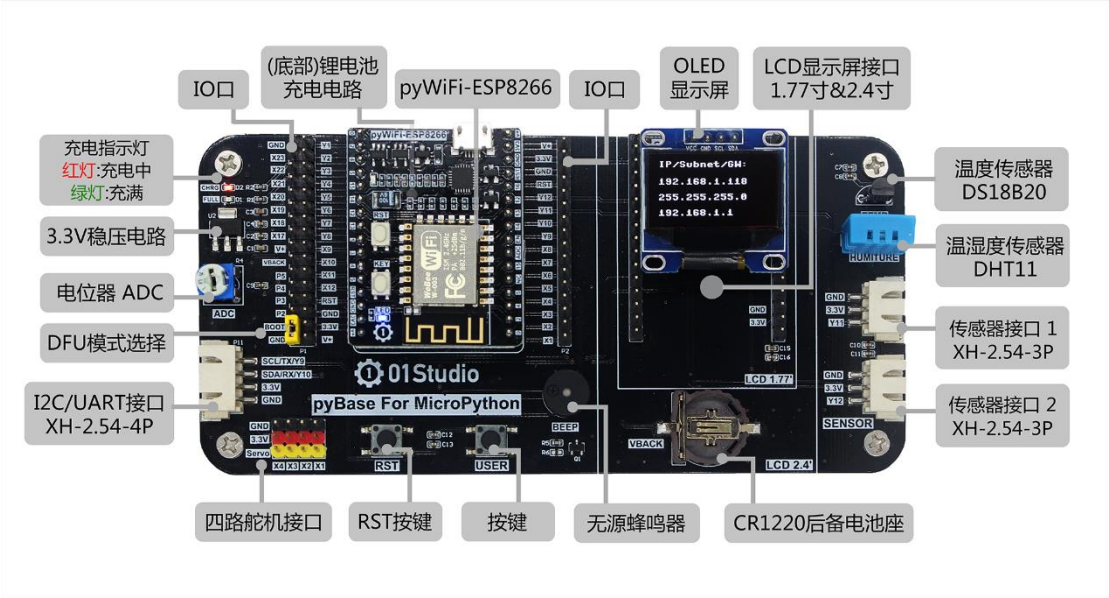


图 1-9 pyWiFi-ESP8266 开发套件

主要配置	
核心板	pyWiFi-ESP8266
底板	pyBase
显示屏	0.9 寸 OLED 显示屏

表 1-2

1.4.2 pyBase

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

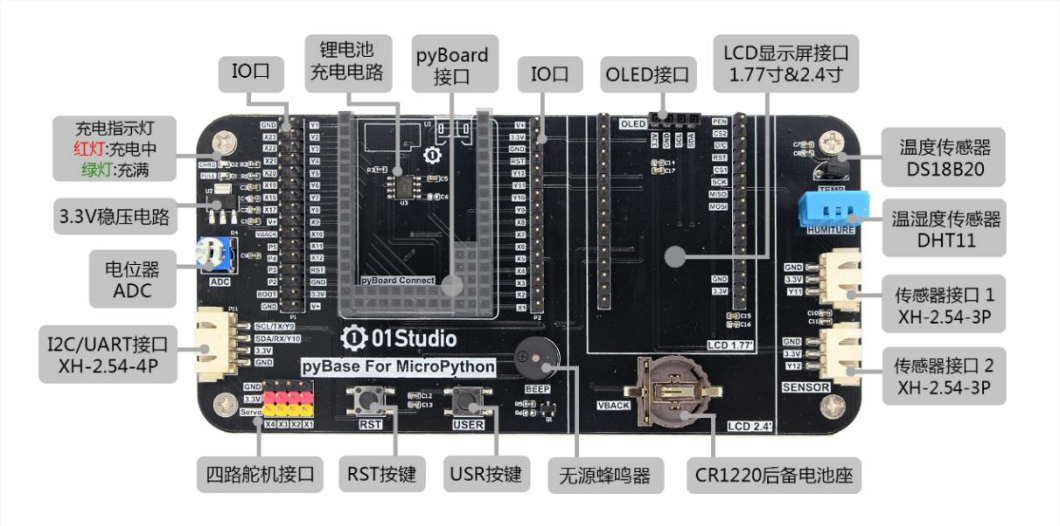


图 1-10 pyBase 功能说明

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

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

表 1-3

1.4.3 传感器模块

1.4.3.1 温度传感器模块（金属探头封装）

图 1-11 金属探头温度传感器

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

表 1-4

1.4.4 拓展配件

1.4.4.1 OLED 显示屏

图 1-12 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 1-5

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 pyhton 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的 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 :(
```

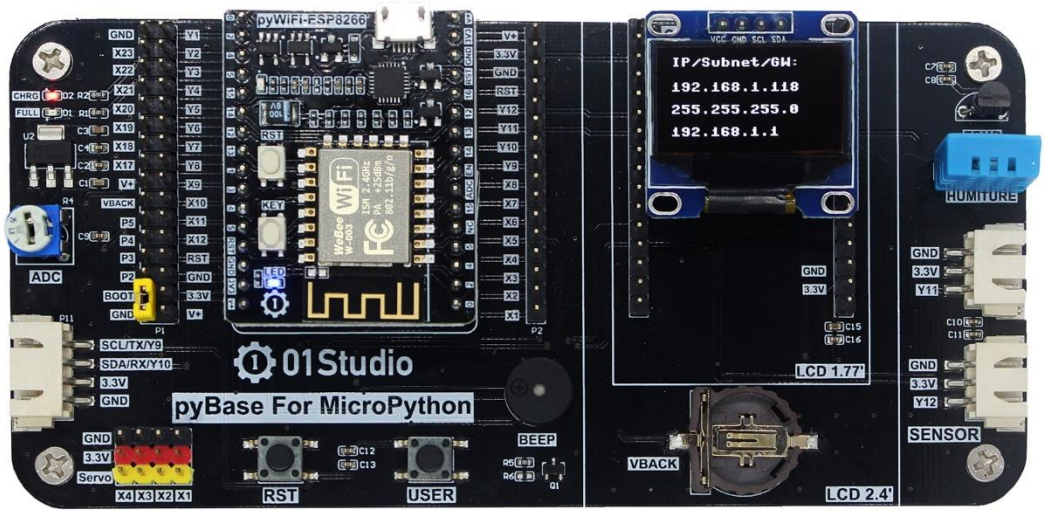
第二部分　基于 ESP8266 平台

ESP8266 相信大部分朋友都听过了，这是上海乐鑫公司推出的一款集成度非常高的 WiFi 无线 SOC，该芯片的特点是集成度非常高，只需要很少的周边外围元器件就能工作，因此市面上的 ESP8266 WIFI 模块也是非常成熟，用户拿到手即可开发。另外极高的性价比为 WIFI 嵌入式系统提供无限可能性。

传统的 ESP8266 开发方式是基于嵌入式 C 语言体系，这为一部分想快速应用物联网的用户增加了门槛，而 MicroPython 对 ESP8266 的兼容性已经非常全面了，我们使用 MicroPython 进行开发，可以快速实现基础功能和联网，非常高效！

如果你已经学习了前面基于 STM32 平台部分内容，那么恭喜你，你需要的只是搭建一下开发环境就可以快速开发了。如果没有学习过前面内容，也没关系，本部分也是会从 0 到 1，带你一步步学会 MicroPython 基于 ESP8266 的编程应用。

本部分内容包含开发环境快速建立、基础实验和 WIFI 项目，由于 ESP 主要是实现物联网 WIFI 无线连接，因此我们会重点讲解 WIFI 联网实验内容。



pyWiFi-ESP8266 开发套件

第3章　开发环境快速建立

3.1　基于 Windows

3.1.1　安装开发软件 Mu

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，我们推荐使用 Mu。

Mu 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\01-开发工具\01-Windows\MicroPython 开发软件(IDE)\Mu** 下获取。

也可以通过 01Studio 社区选择合适自己电脑的版本下载，下载链接：

http://www.01studio.org/forum.php?mod=viewthread&tid=210&extra=page%3D1

图 3-1 Mu 下载

下载安装后，在桌面不会生成图标，这时候可以通过 Windows 搜索栏搜索“mu”关键词找到软件并打开。

图 3-2 搜索“Mu”

打开 Mu，可以看你的界面非常简洁：

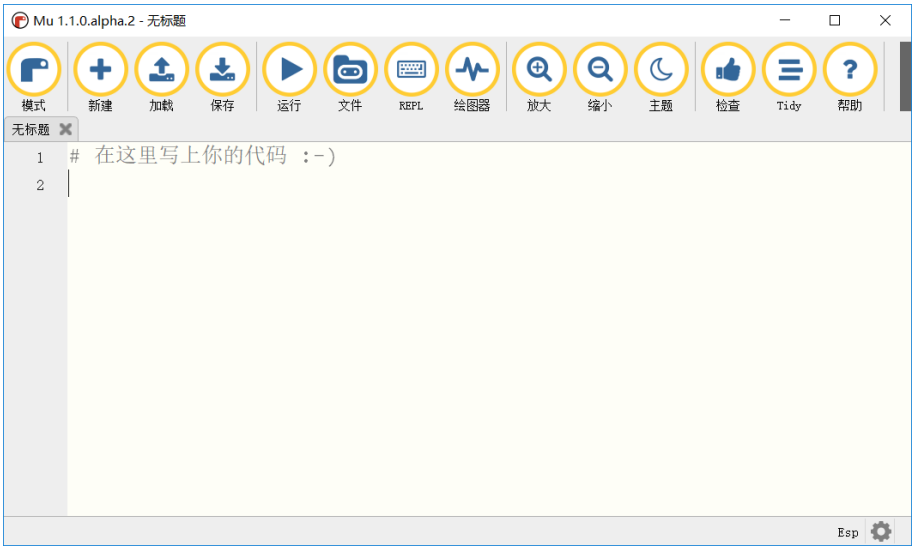


图 3-3 Mu 主界面

点击左上角“模式”，弹出对话框，可以看到支持 MicroBit、ESP 等一系列 MicroPython 产品。通常情况下在 USB 线插入能自动识别设备硬件类型。基于 pyWiFi-ESP8266 开发板选择【ESP MicroPython】进行开发。

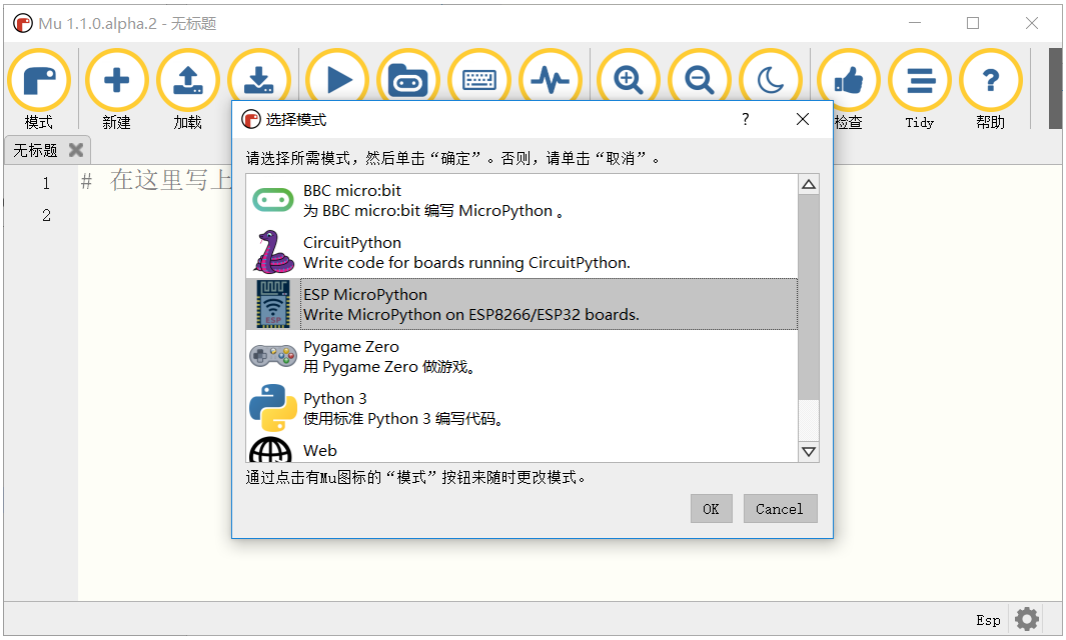


图 3-4

至此，Mu 安装完成。

3.1.2 开发套件使用

3.1.2.1 驱动安装

主要是安装 USB 转串口驱动。我们将 pyWiFi-ESP8266 开发板通过 MicroUSB 数据线连接到电脑：

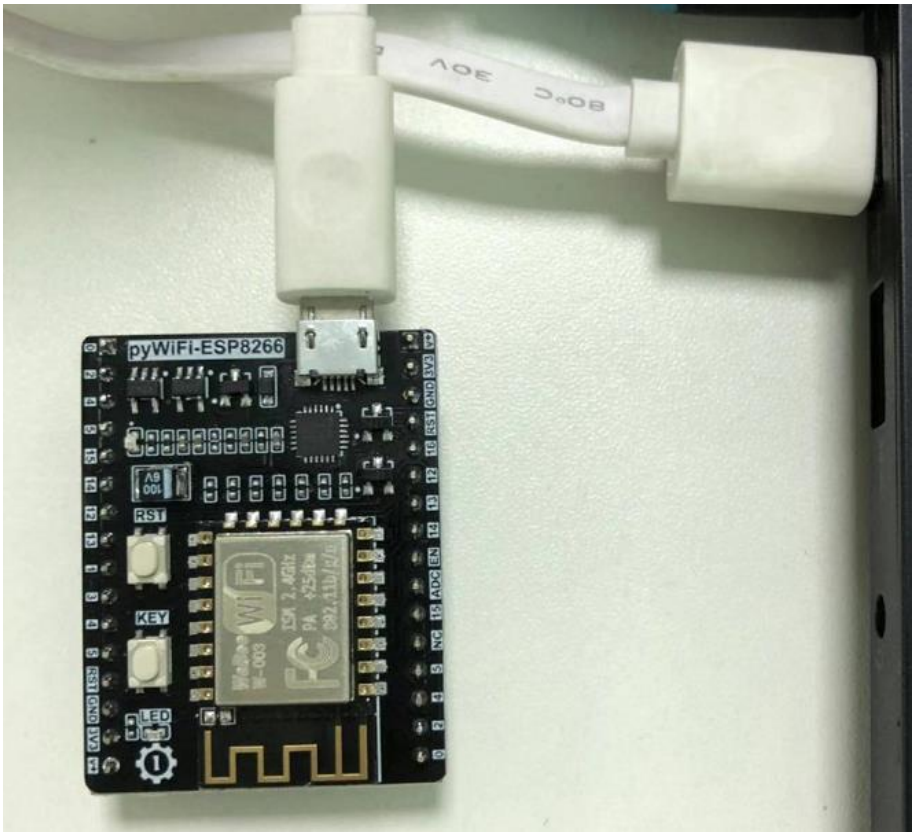


图 3-5 通过 MicroUSB 线连接到电脑

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。

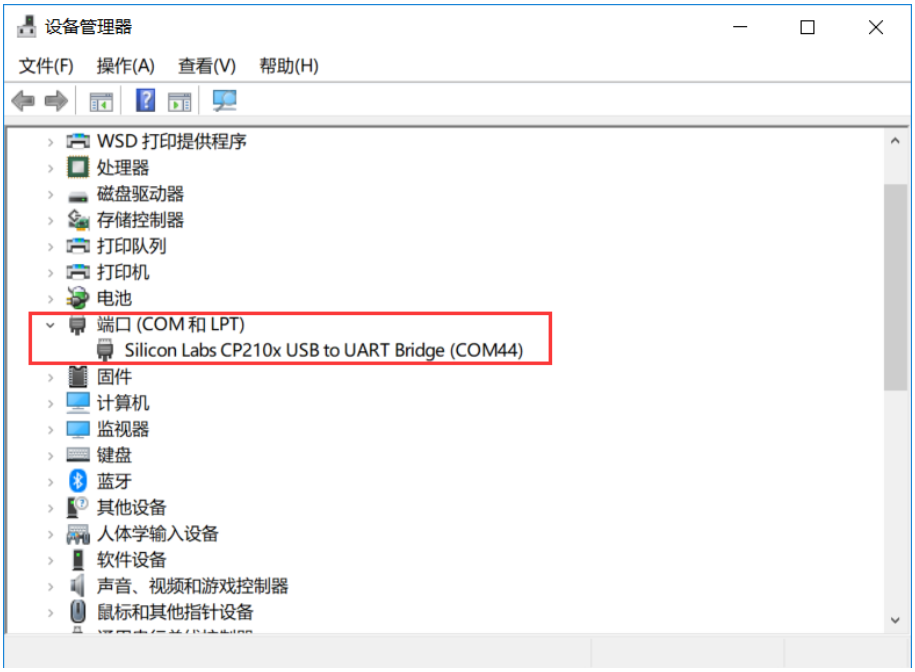


图 3-6 串口驱动成功安装

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”：

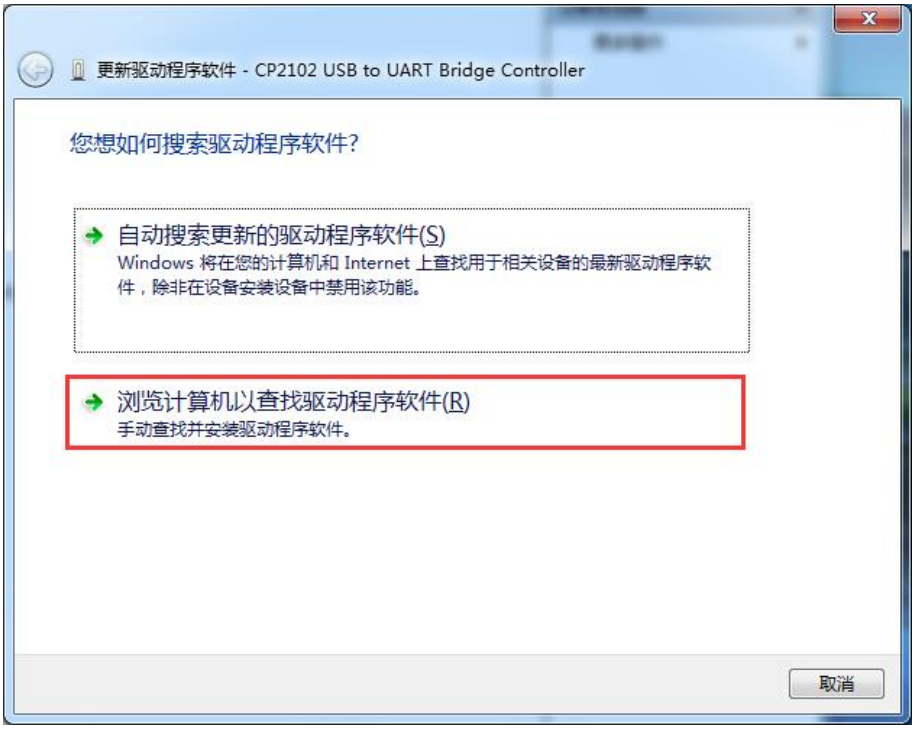


图 3-7

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

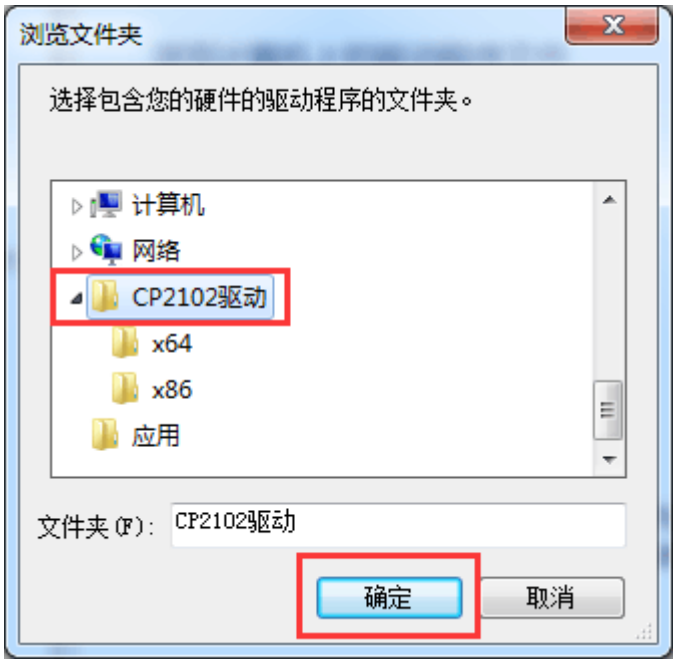


图 3-8

安装成功后如下图：

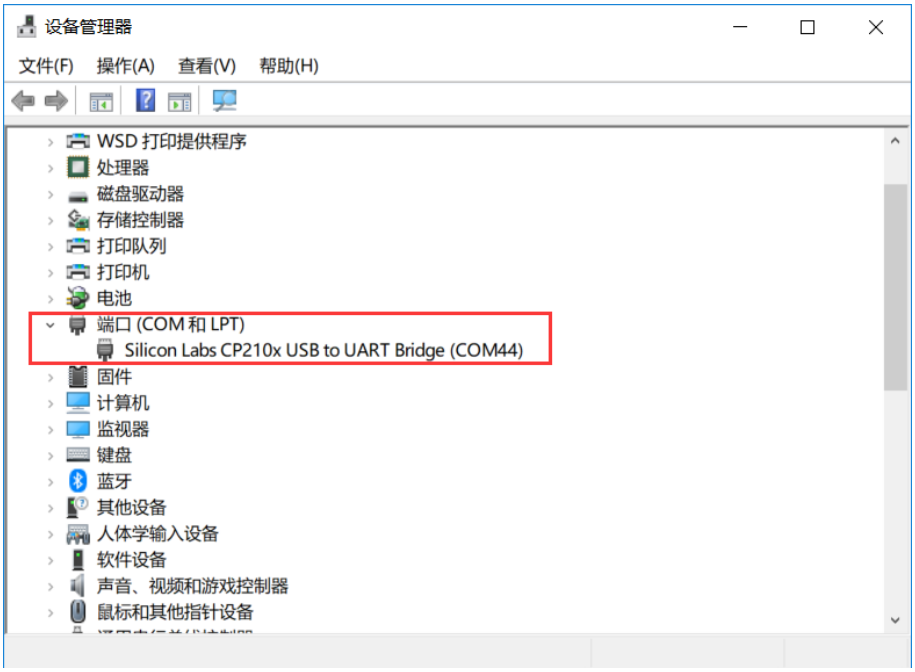


图 3-9 串口驱动安装成功

3.1.2.2 REPL 串口交互调试

MicroPython 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 micropython 开发套件。

打开 Mu 软件，连接 pyWiFi-ESP8266 开发板，点击上方 REPL 按钮，软件下方即出现 REPL 调试界面：

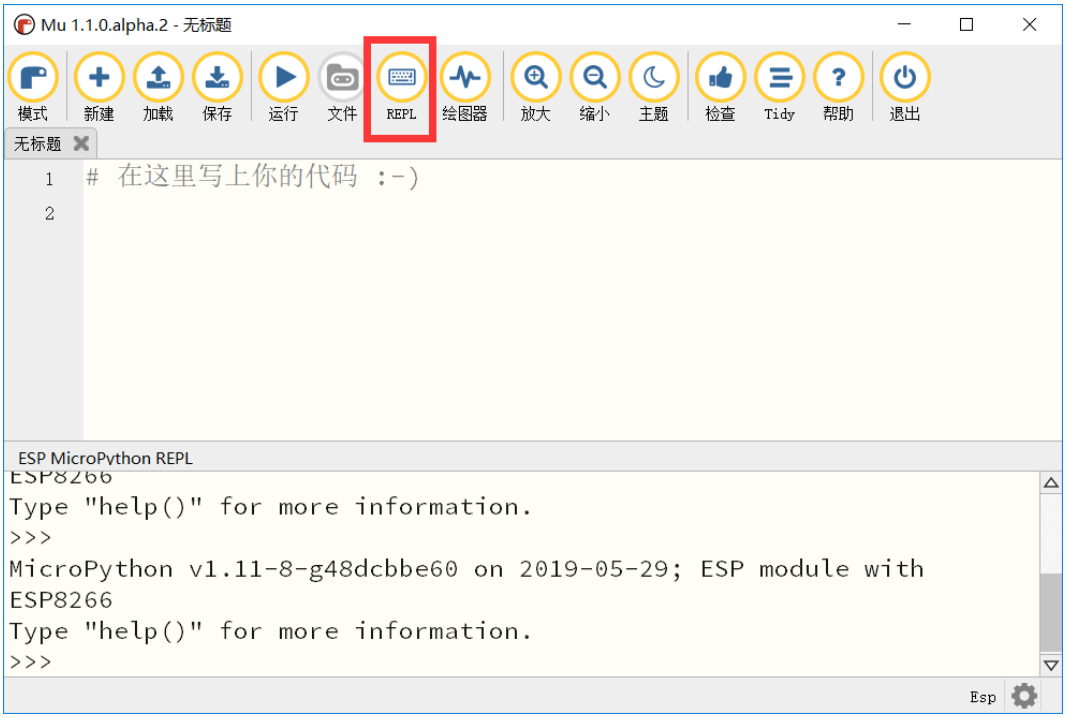


图 3-10

我们可以简单测试一下这个 REPL 交互，输入下面语句后按回车，可以看到终端打印出相关信息：

```
print(“hello 01studio!”)
```

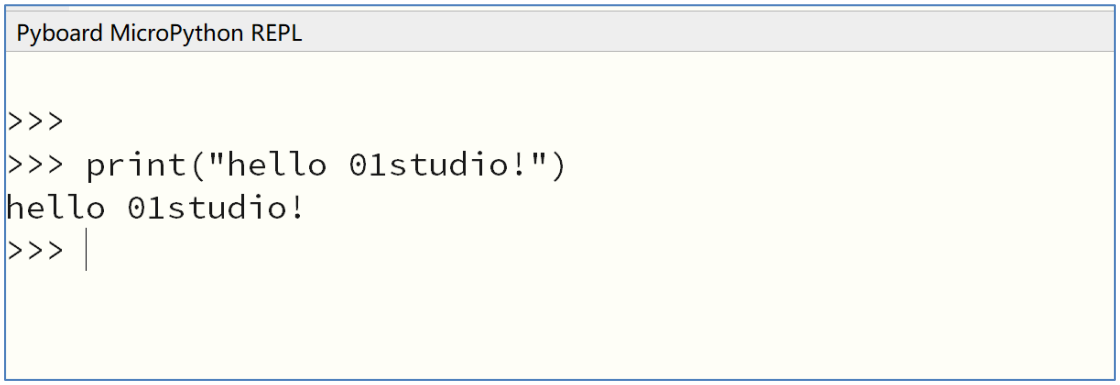


图 3-11

输入 1+1，可以看到出来结果为 2. 这过程就是调用了开发板里面的 micropython 解析器。

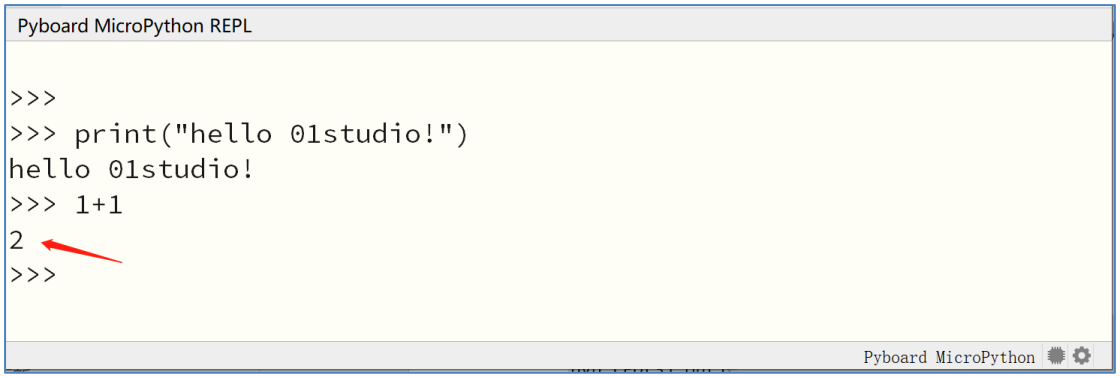


图 3-12

RLPL 终端常用键盘组合键：

1. Ctrl+C：打断正在运行的程序，当代码含 while True: 时，REPL 可能无法响应，这是可以使用此组合键打断正在运行的程序。
2. Ctrl+D：软件复位开发板。

其它串口终端助手：putty 介绍

除了使用 Mu 自带串口 REPL 功能外，你也可以使用串口终端软件来调试，这里介绍一款免费开源的串口终端软件 putty。**（建议直接使用 Mu 的 REPL，这部分内容仅作为补充，可以跳过）。**

将开发板连接到电脑，从我的电脑—属性—设备管理器中找到当前的串口号，这里是 COM4。

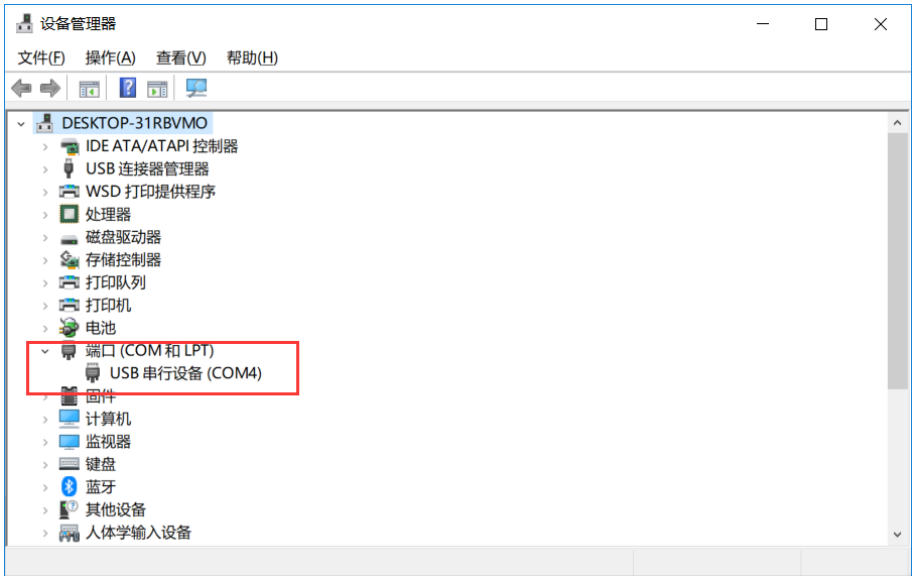


图 3-13 找到串口号

打开 MicroPython 开发套件配套资料\开发工具\串口终端工具\Putty.exe，选择左下角 Serial，配置信息如下：

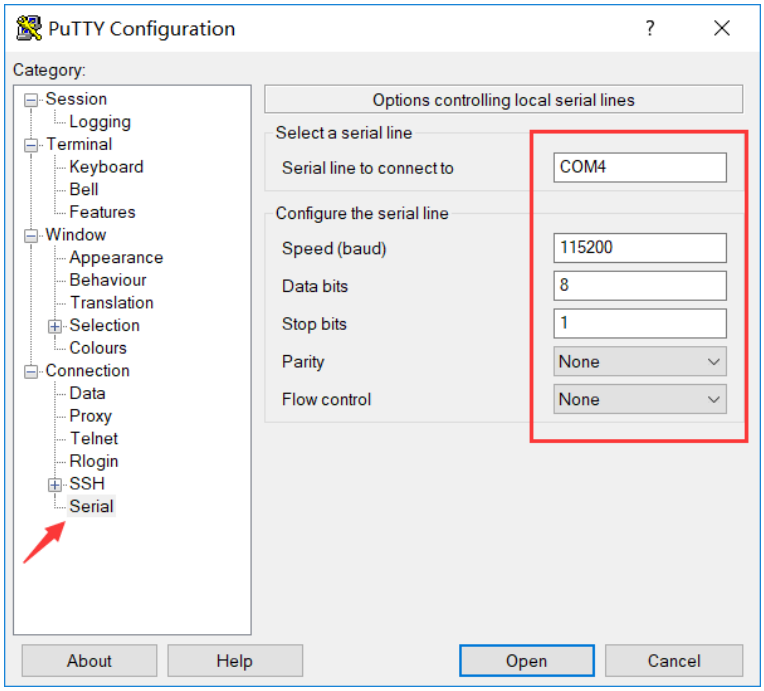


图 3-14 配置串行设备参数

配置好后不是点 open，而是点左边上方 Session，选择 Serial 后可看到刚刚的配置信息。串口号通常不会变化，我们在 Save Session 下方输入 COM4 或者自己喜欢的名称，点右边 Save，在空白框里面就出现 COM4 字样，以后可以直接使

用。设置好后我们点击 Open。

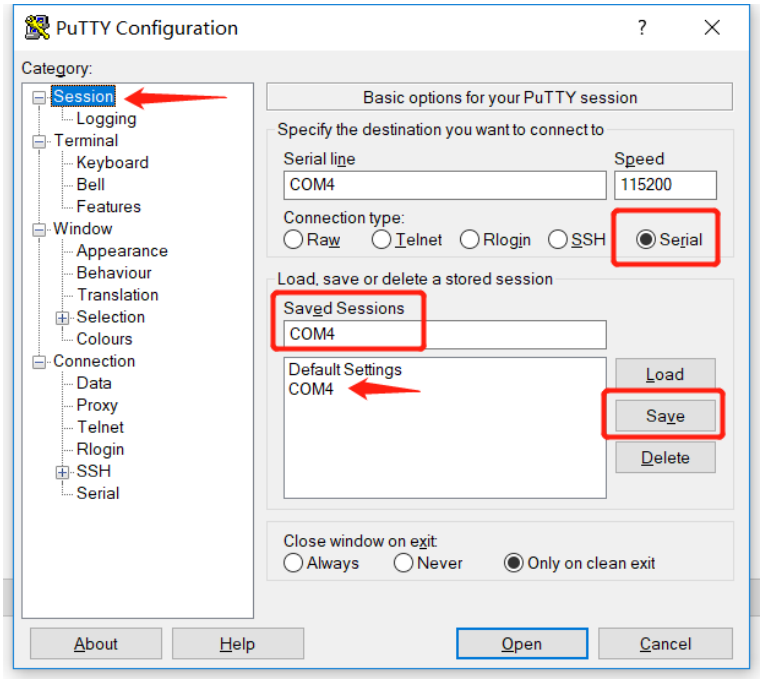


图 3-15

出现下面对话框，是不是似曾相识，没错，跟我们之前在计算机命令行执行 python 指令后出现类似的。上面主要是当前设备固件的版本号。

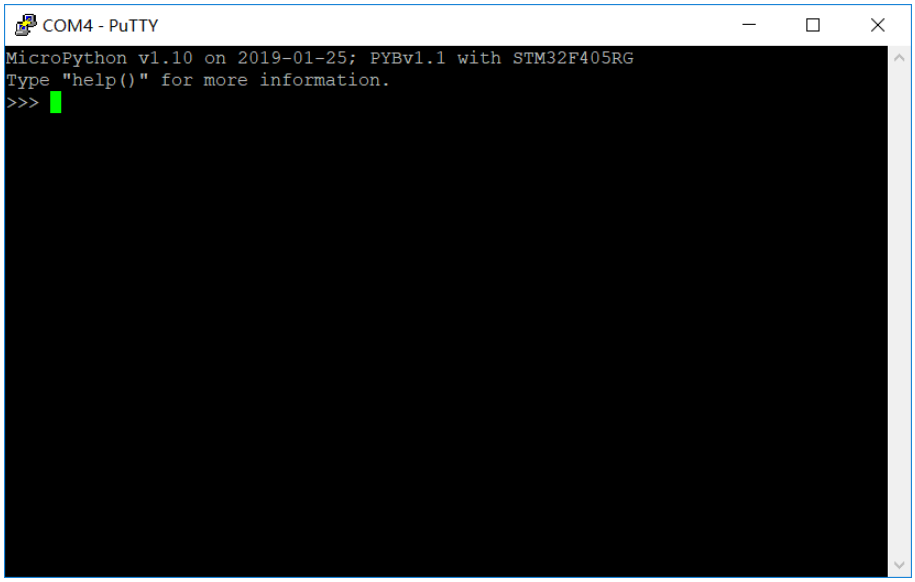


图 3-16

现在对话框相当于连接上了开发板上，由于 pyboard 集成了解析器。我们在这里可以进行调试和简单编程，接下来我们测试一下。在对话框输入下面代码，

按回车，可以看到代码运行情况。

```
print(“Hello 01Studio!”)
```

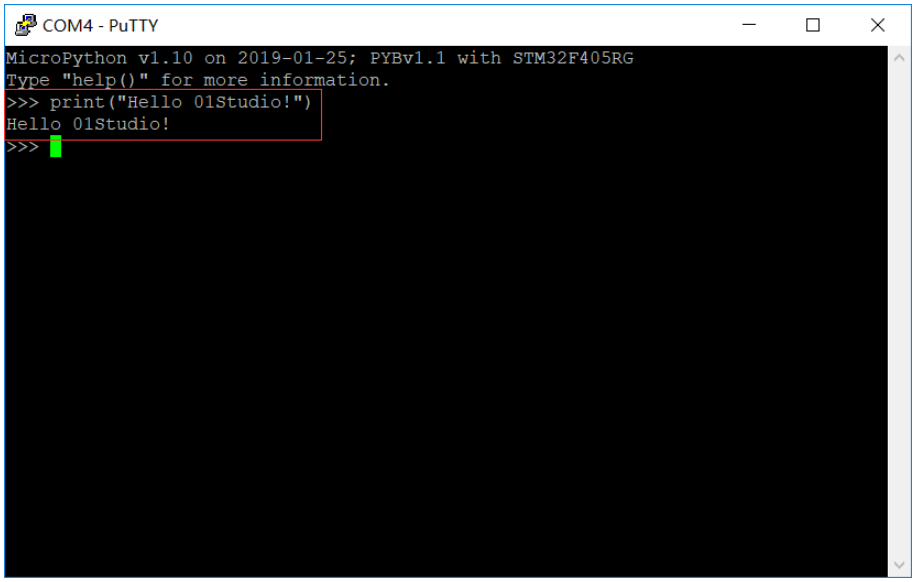


图 3-17

我们再输入 1+1 按回车，得到了计算结果 2。

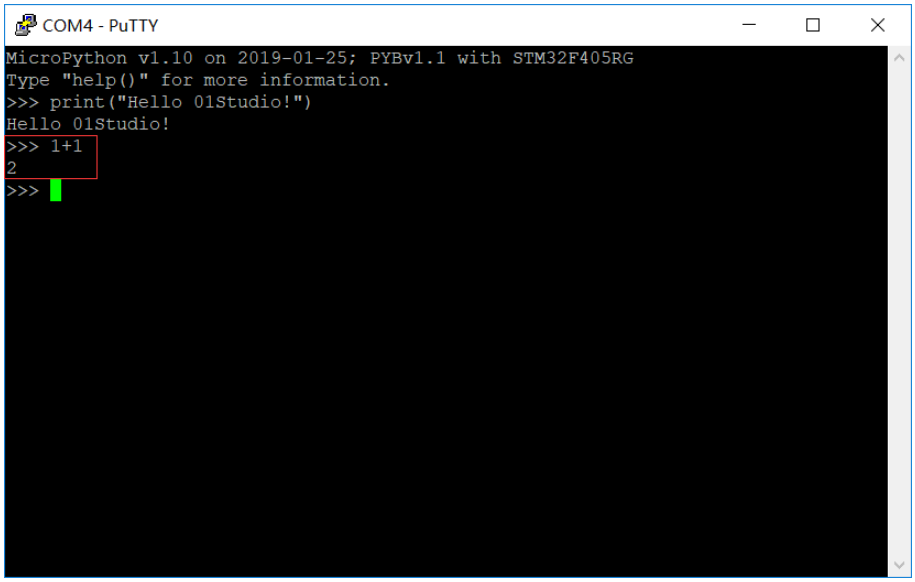


图 3-18

3.1.2.3 文件系统

pyWiFi 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 Mu 非常方便地读取。

连接开发板，打开 Mu，先按一下开发板的复位键，然后点击上方“文件”按钮，可以在下方弹出的方框看到开发板里的文件和本地 PC 的文件。

（由于串口是共用的，因此有些时候可能打不开，这时候要先关掉 Mu 的文件或者 REPL 窗口，按下开发板复位键，再点击打开。）

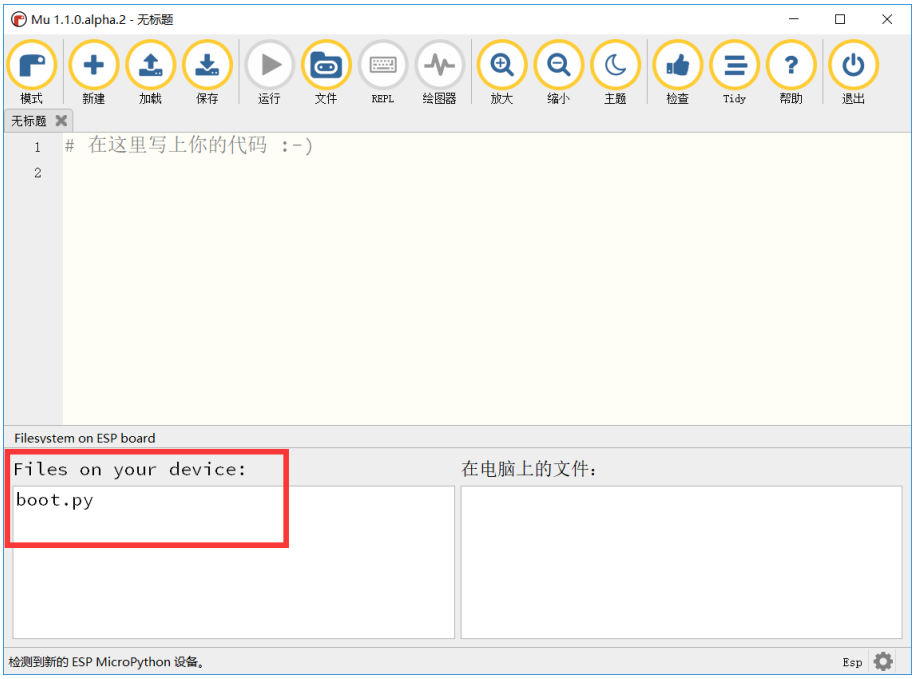


图 3-19

从上图可以看到开发板上的有个“boot.py”文件，这是系统启动后执行的第一个文件，通常做初始化工作。我们尝试将 boot.py 文件拖动到电脑的空白区，拖完后即完成从设备到电脑的文件复制，反过来就是从电脑复制文件到设备，使用非常方便直观。

复制完成后可以点击电脑端文件右键，选择在 Mu 中打开，即可查看和编辑该文件。**（新版固件可能没 boot.py 文件，不影响使用。）**

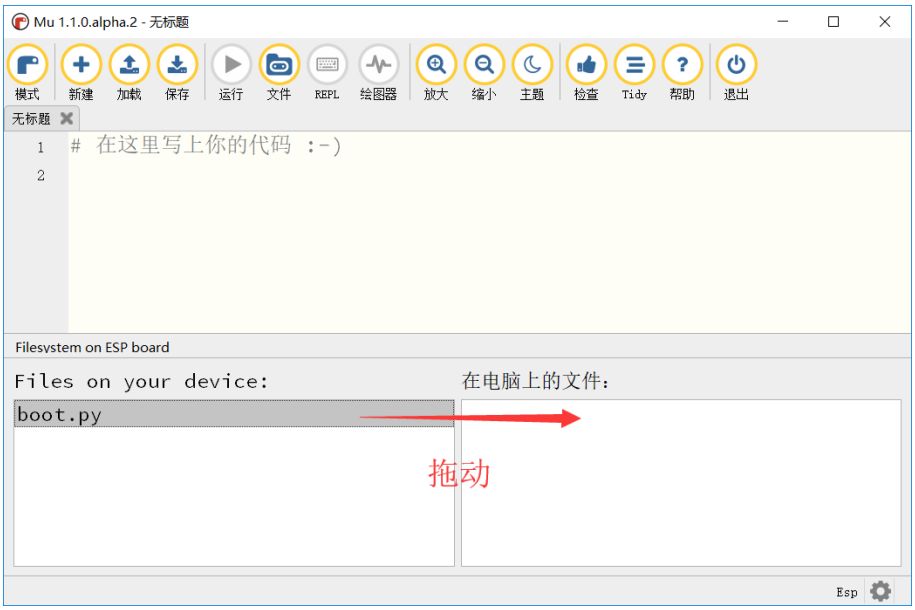


图 3-20

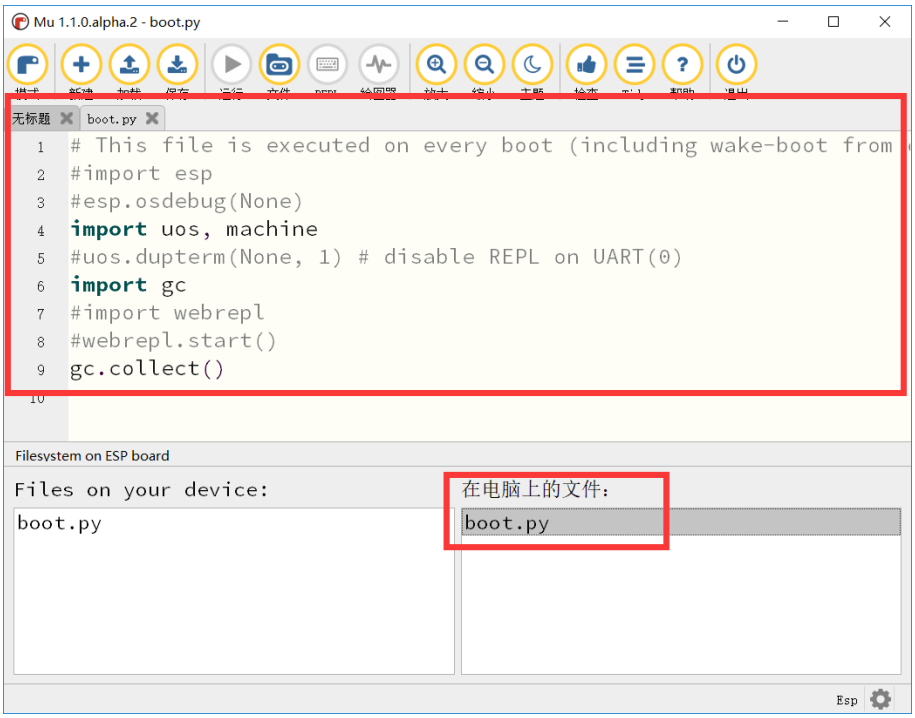


图 3-21 查看本地文件

那么本地文件都放在什么地方呢，可以在 Mu 中**双击**文件上方名称，即可在弹出的对话框中看到该文件的保存位置。

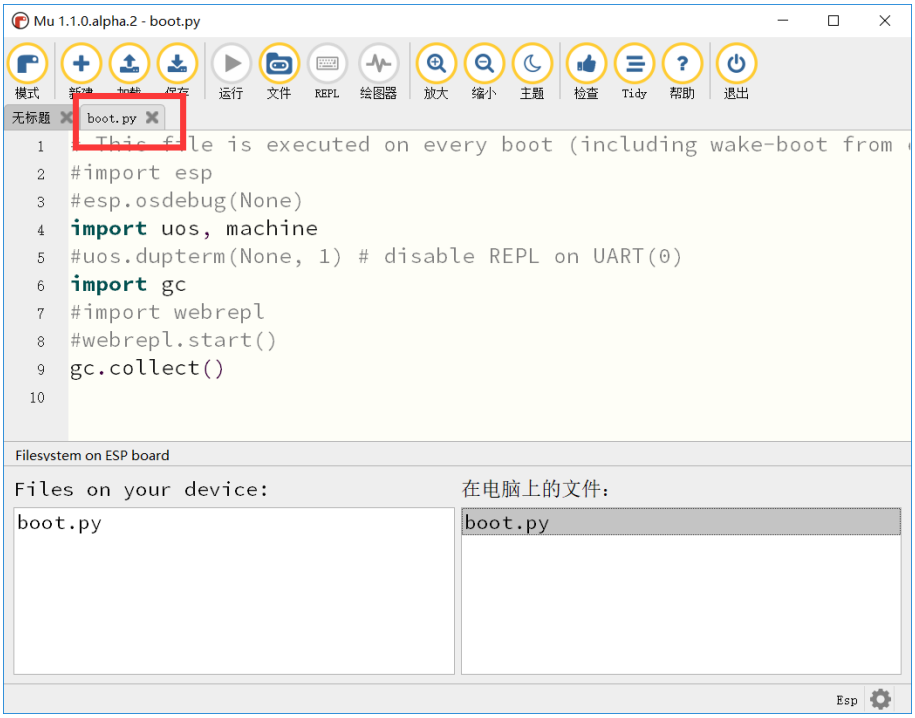


图 3-22 双击文件名

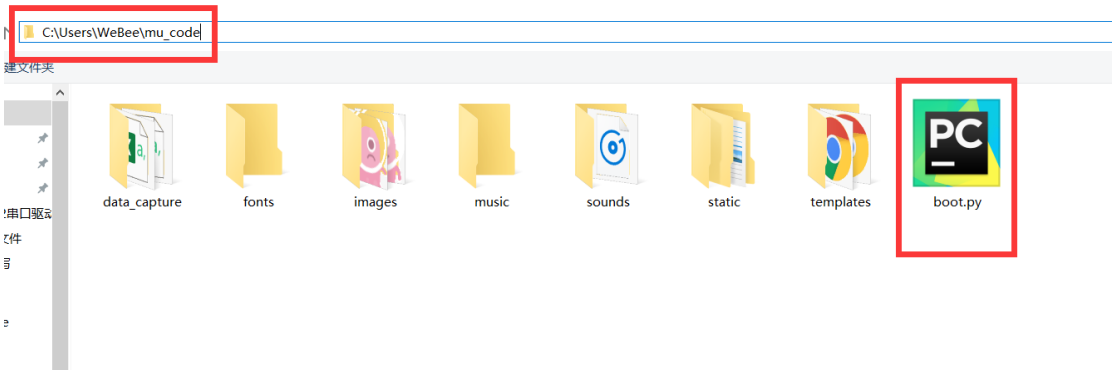


图 3-23 本地文件位置查看

我们在 Mu 中新建一个空白文件，保存名称为 main.py 到刚刚查看到的那个本地文件的路径：

图 3-24

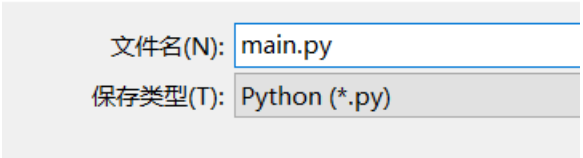


图 3-25

点击文件按钮关闭文件下方方框，再次点击刷新一下，可以发现出现了刚刚保存的 main.py 文件。将 main.py 文件拖动到左边设备栏，可以看到设备栏出现了 main.py 文件，说明复制成功。这就是使用 Mu 实现电脑跟设备文件相互传输的方法。

图 3-26

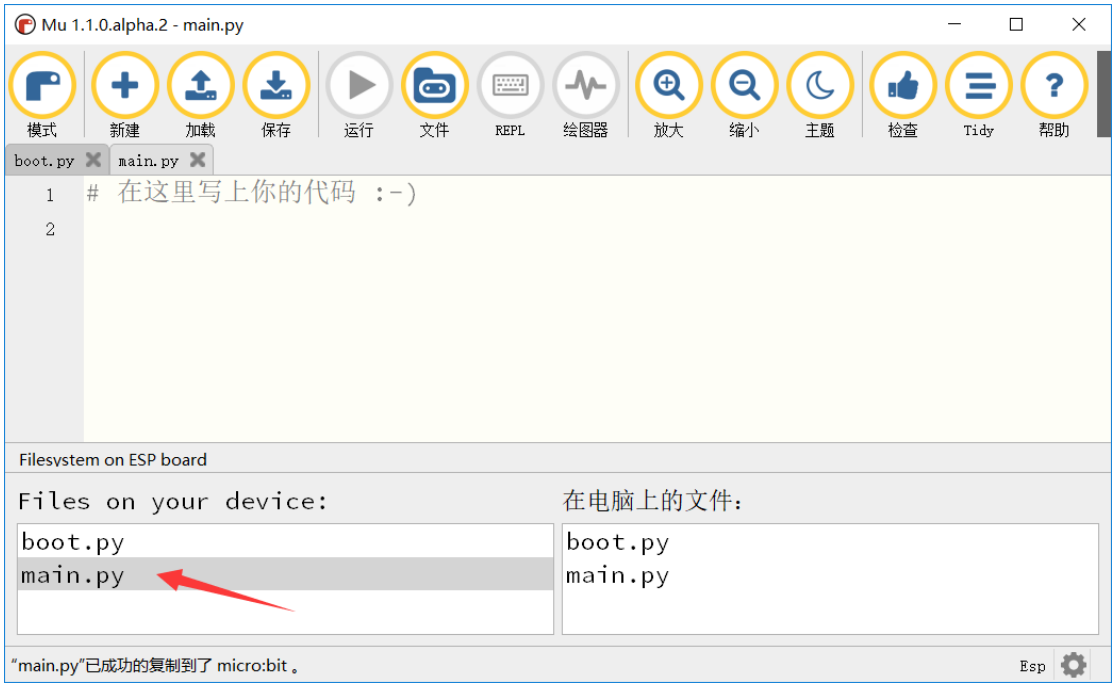


图 3-27 成功复制到设备

main.py 是主文件，是运行完 boot.py 后执行的第一个程序文件，我们主要也是在 main.py 文件里面编程，这在后面的实验会详细讲解。

3.1.2.4 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，烧录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_latest\01-开发工具\01-Windows\固件更新工具\flash_download_tools_v3.6.5** 下的 flash_download_tools_v3.6.5.exe 软件，双击打开。

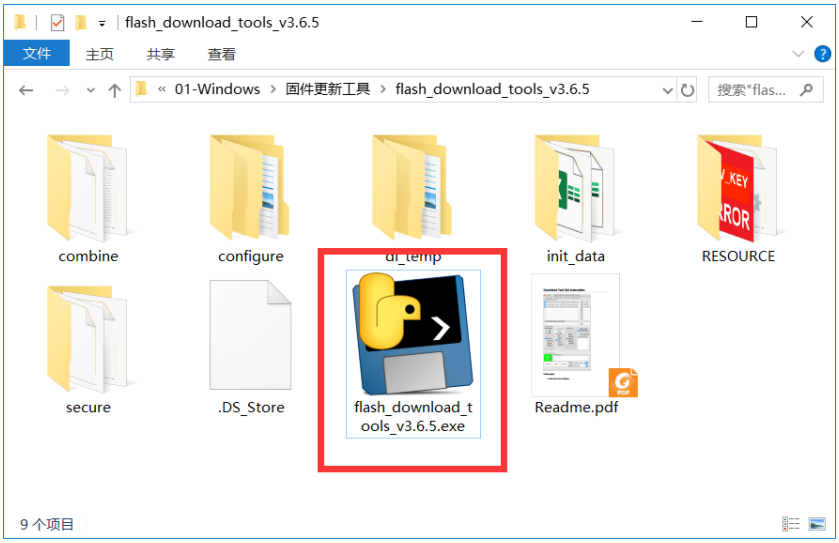


图 3-28

选择 ESP8266 Download tool :

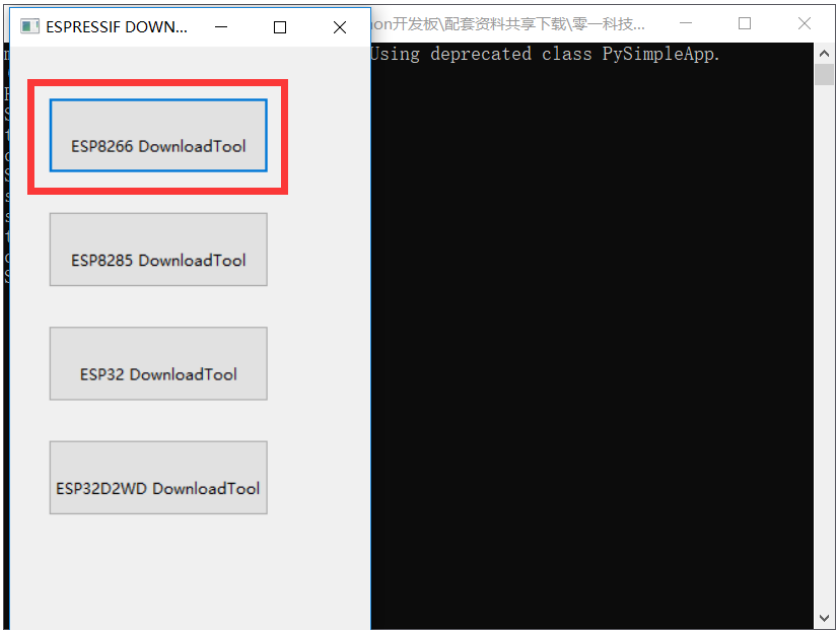


图 3-29 选择 ESP8266

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\01-pyWiFi-ESP8266** 目录下。

其它配置选项也请参考下图：（COM 串口是选择自己的串口，在设备管理器查询。）

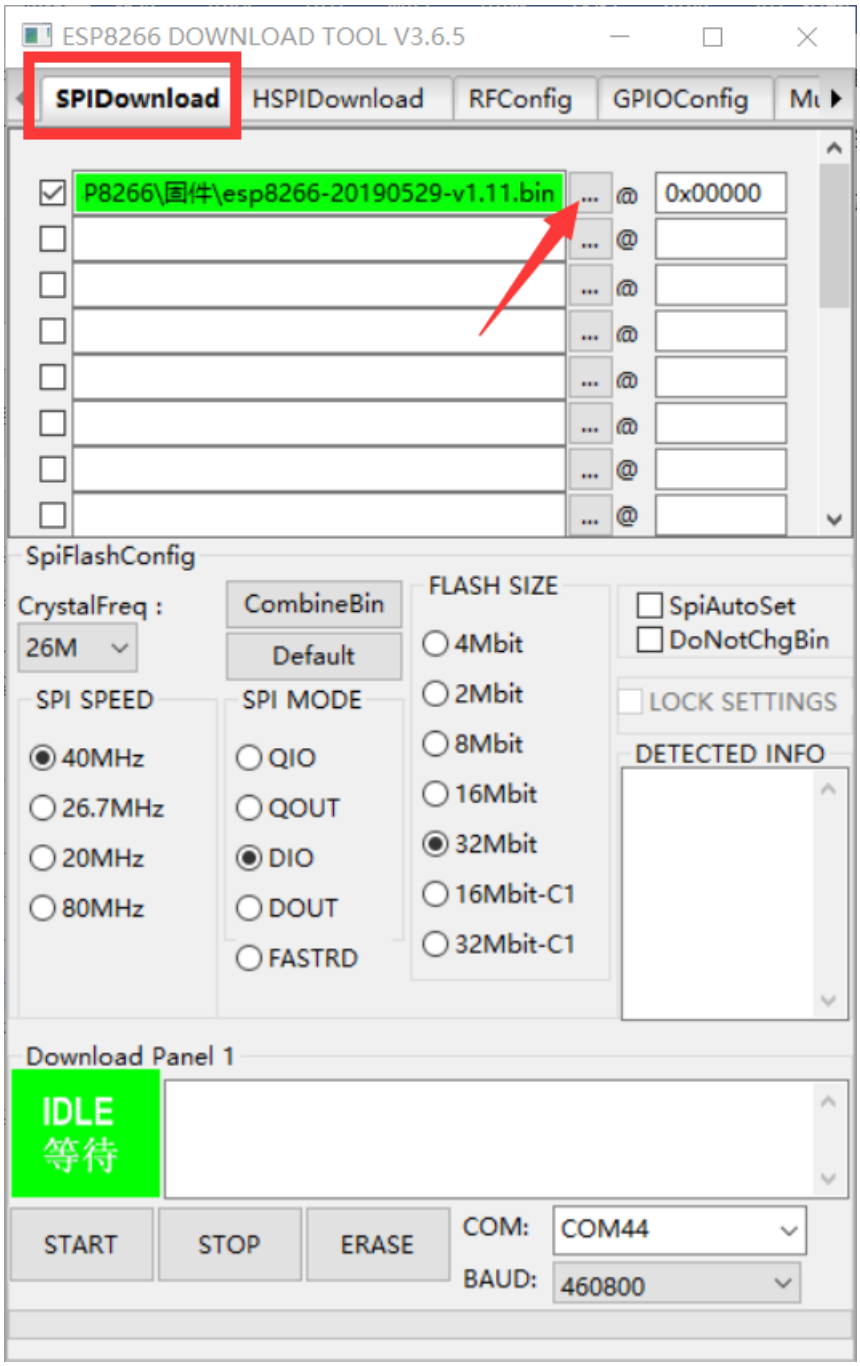


图 3-30 配置

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。

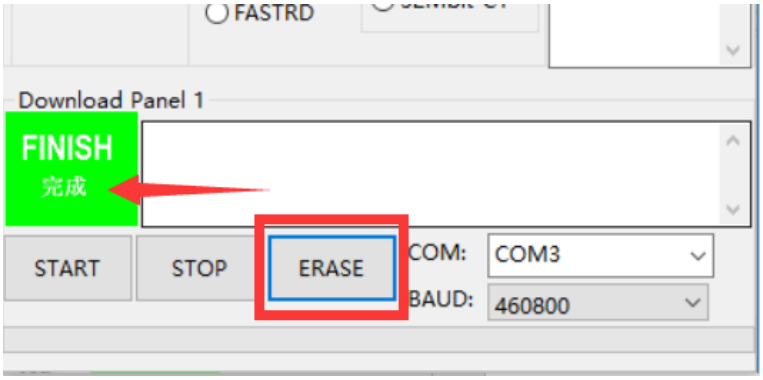


图 3-31 刷除 flash 成功

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。

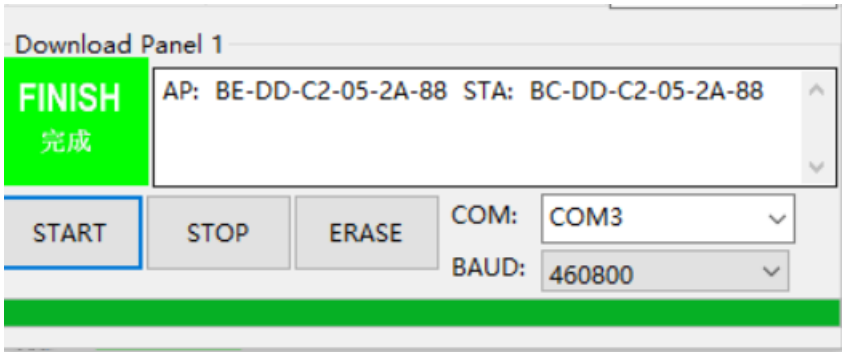


图 3-32 烧录成功

烧写完成后，打开 Mu 的 REPL 串口调试（也可以使用 putty 或其它串口调试工具），发现出现了熟悉的 MicroPython 调试模式。输入以下两句指令，可以检测固件的完整性。

```
import esp

esp.check_fw()
```

当系统返回 True 时候，说明固件是完整的。

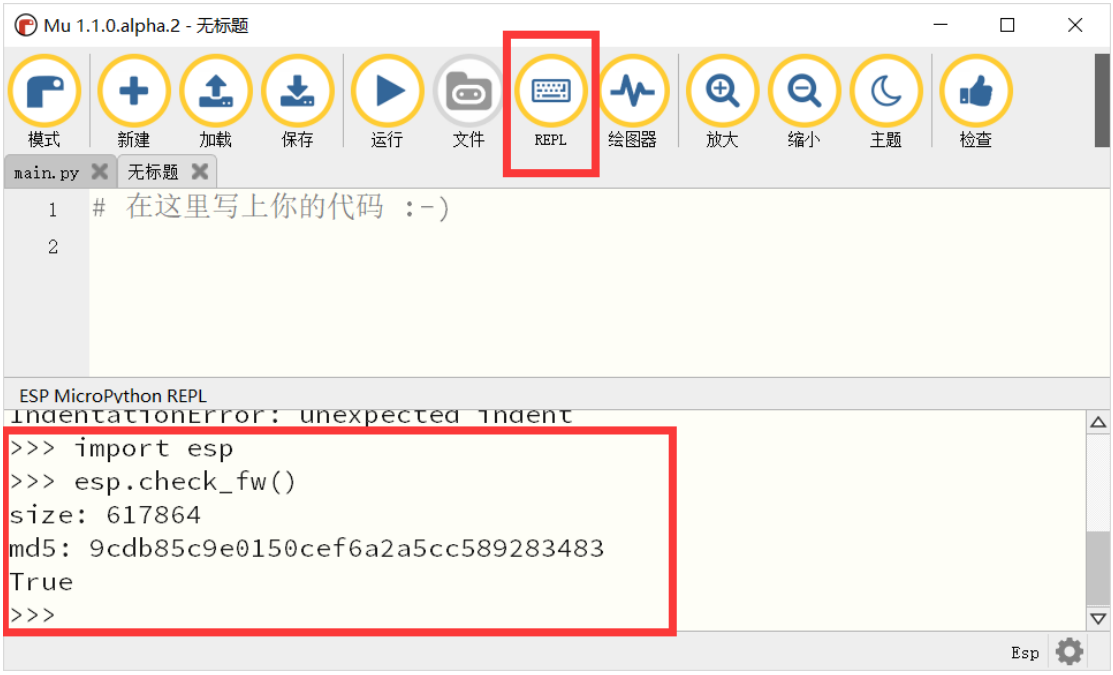


图 3-33 点击上方 REPL 按钮进入调试

图 3-34 固件完整性检测

3.2 基于 Mac OS

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

3.2.1 安装开发软件 Mu

Mu 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 Mu** 章节内容，这里不再重复。

3.2.2 开发套件使用

3.2.2.1 驱动安装

Macbook 自带这个版本的串口驱动，一般不需要再安装，查看方法如下：

将 pyWiFi-ESP8266 通过 USB 线连接到 MacBook：

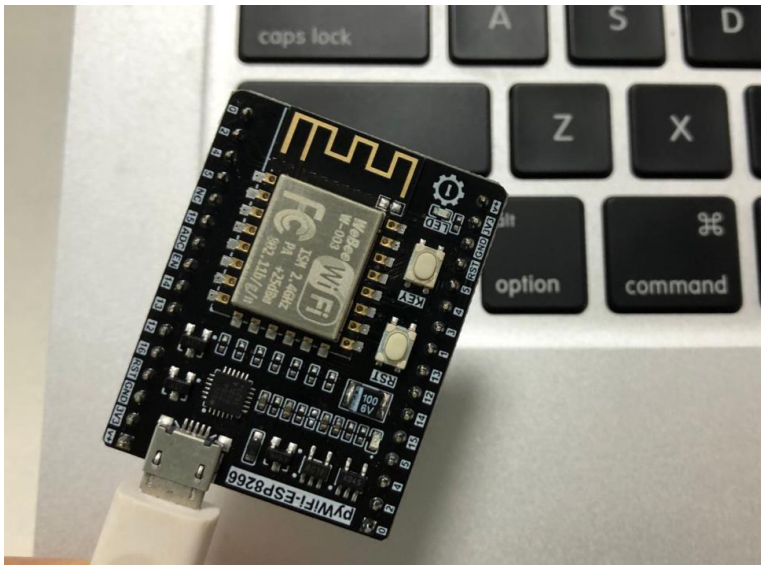


图 3-35

点击左上角苹果图标—关于本机—系统报告—选择 USB。可以看到相关驱动如下图所示：

图 3-36 关于本机

图 3-37　CP210X USB 转串口驱动

3.2.2.2 REPL 串口交互调试

基于 Mac OS 的 REPL 串口交互调试可以使用 Mu 软件自带的 REPL 功能。

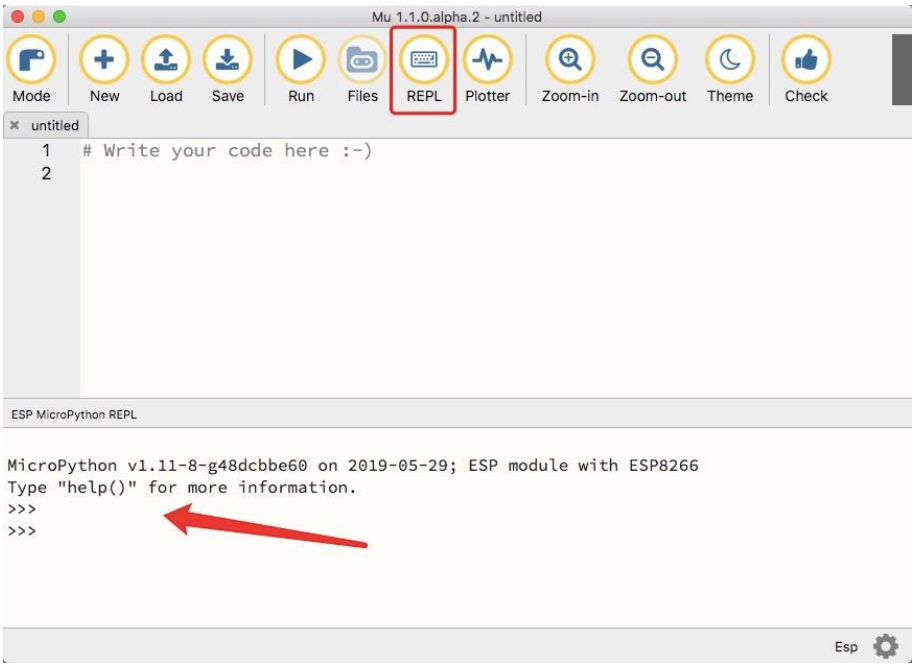


图 3-38 Mu 的 REPL

3.2.2.3 文件系统

这里是指实现电脑与开发板之间的文件传输，具体请参阅：**3.1.2.3 文件系统** 章节内容，这里不再重复

3.2.2.4 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

```
[redacted]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 3-39 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（esp8266-20190529-v1.11.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。）

```
esptool.py --port /dev/tty.SLAB_USBtoUART --baud 460800 write_flash --
flash_size=detect 0 esp8266-20190529-v1.11.bin
```

烧写成功如下图所示：

```
[redacted]MacBook-Air:Desktop jackey$ esptool.py --port /dev/tty.SLAB_USBtoUART --baud 460800 write_flash --flash_size=detect 0 esp8266-20190529-v1.11.bin
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...
Changing baud rate to 460800
Changed.
Configuring flash size...
Auto-detected Flash size: 1MB
Flash params set to 0x0020
Compressed 617880 bytes to 402086...
Wrote 617880 bytes (402086 compressed) at 0x00000000 in 9.7 seconds (effective 510.9 kbit/s)...
Hash of data verified.
```

图 3-40 烧写成功

烧写完成后，打开 Mu 的 REPL 串口调试（也可以使用其它串口调试工具），发现出现了熟悉的 MicroPython 调试模式。输入以下两句指令，可以检测固件的完整性。

```
import esp

esp.check_fw()
```

当系统返回 True 时候，说明固件是完整的。

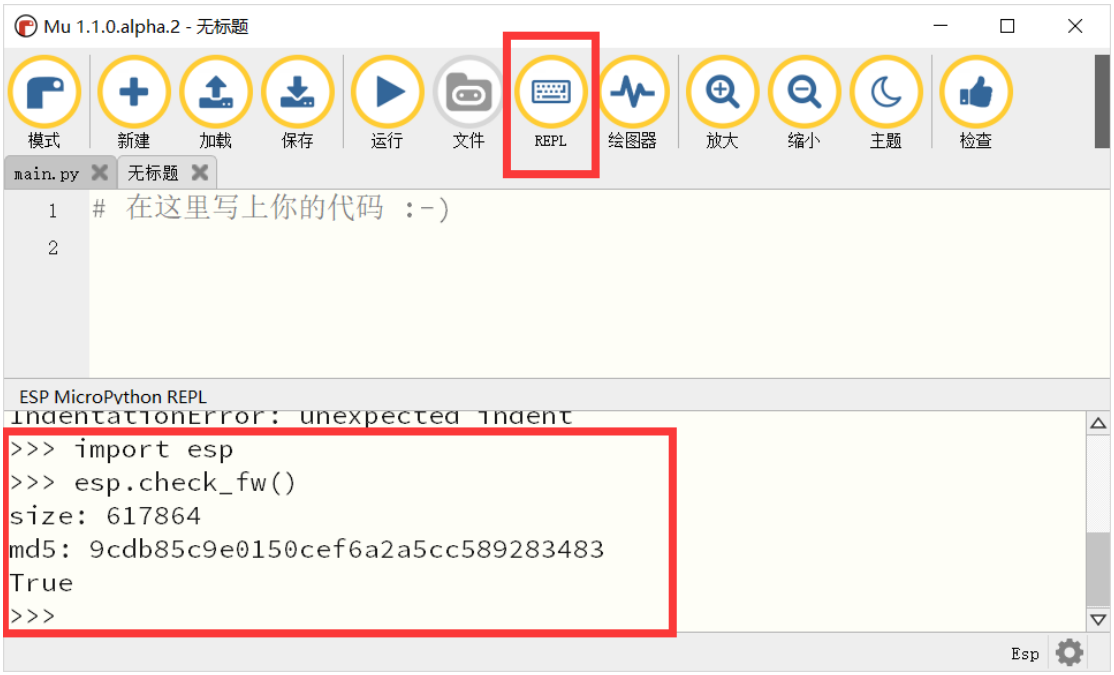


图 3-41 点击上方 REPL 按钮进入调试

图 3-42 固件完整性检测

3.3 基于 Linux

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

3.3.1 开发软件 Mu 安装

安装方法请参考：https://codewith.mu/en/howto/1.0/install_raspberry_pi

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，务求达到快速理解和学习的作用：

（1）　前言：简单介绍这个实验；

（2）　实验平台：实验所用到的开发板、配件和接线说明；

（3）　实验目的：本实验要实现的功能；

（4）　实验讲解：对函数、代码、编程方法以及实验的详细讲解；

（5）　实验结果：记录程序下载到开发板上的图片示例；

（6）　总结:对实验进行总结。

4.1 点亮第一个 LED

- **前言：**

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

- **实验平台：**

pyWiFi-ESP8266。

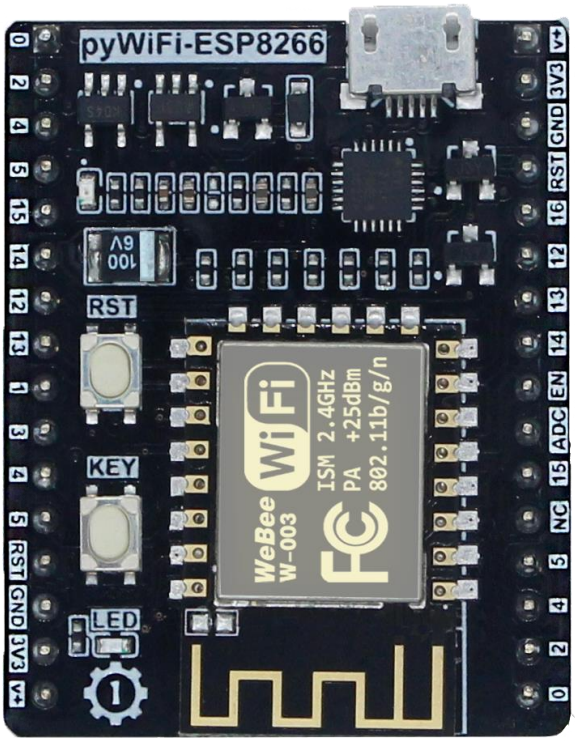


图 4-1 pyWiFi-ESP8266

- **实验目的：**

点亮 LED（蓝灯）。

- **实验讲解：**

pyWiFi-ESP8266 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
led=machine.Pin(id,mode,pull)
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
led.value([x])
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。
led.on()
使引脚输出高电平“1”。
led.off()
使引脚输出低电平“0”。

表 4-1 Pin 对象

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

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

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

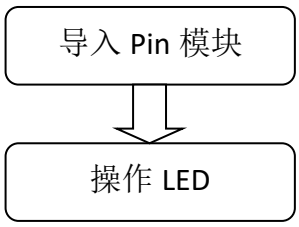


图 4-2 代码编写流程

LED 跟模块引脚 2 相连，通过输出低电平方式点亮。

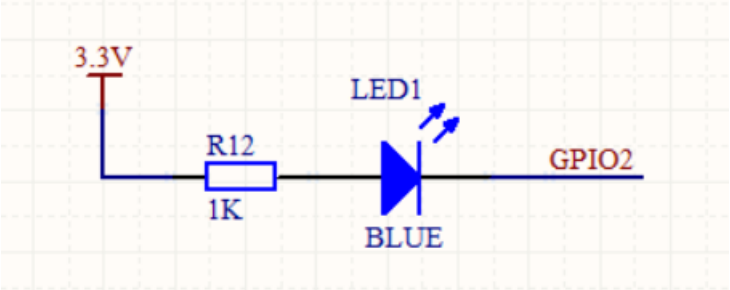


图 4-3 LED 接线图

参考代码如下 ：

```
'''
实验名称：点亮 LED 蓝灯
版本：v1.0
日期：2019.7
作者：01Studio
'''

from machine import Pin #导入 Pin 模块
led=Pin(2,Pin.OUT) #构建 led 对象，GPIO2,输出
led.value(0) #点亮 LED
```

运行程序有两个方法：

方法一：

编写好代码后点击 Mu 上方的“运行”按钮，可以直接观察到代码运行情况。这个方法不会将程序代码保存到 ESP8266 模块的 flash 里面。这注意是方便调试使用。

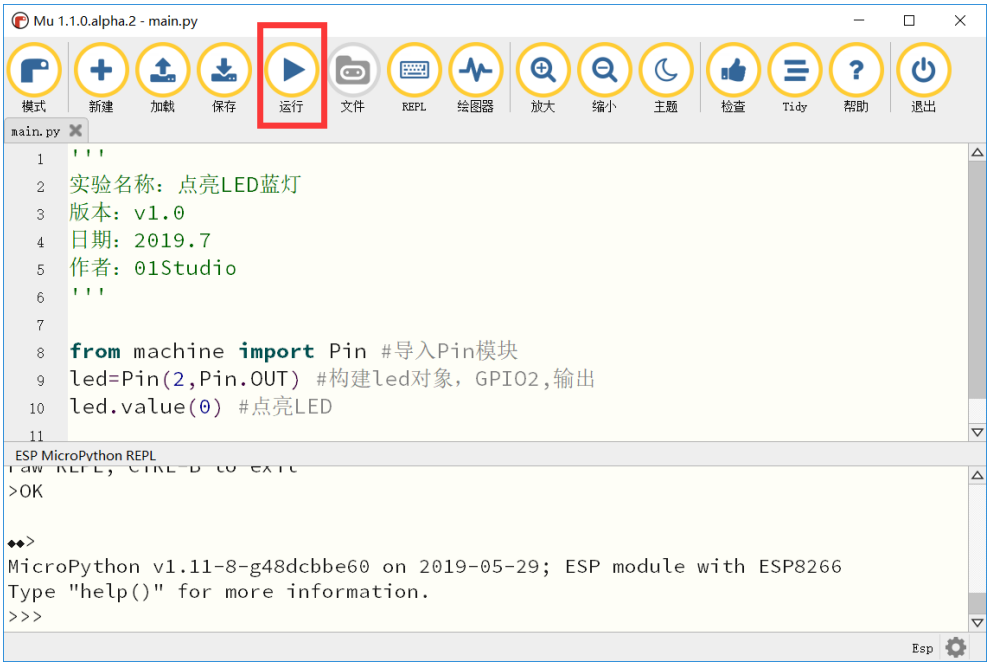


图 4-4 运行仿真

方法二：

将新建的文件保存名称为 “main.py”的 py 文件，使用 Mu 的文件功能，从本地拖动到设备。然后按下复位按键，设备运行相关代码，这个方式相当于将程序烧录到设备 flash。 可以脱机使用。 操作方法参考上一章文件系统章节

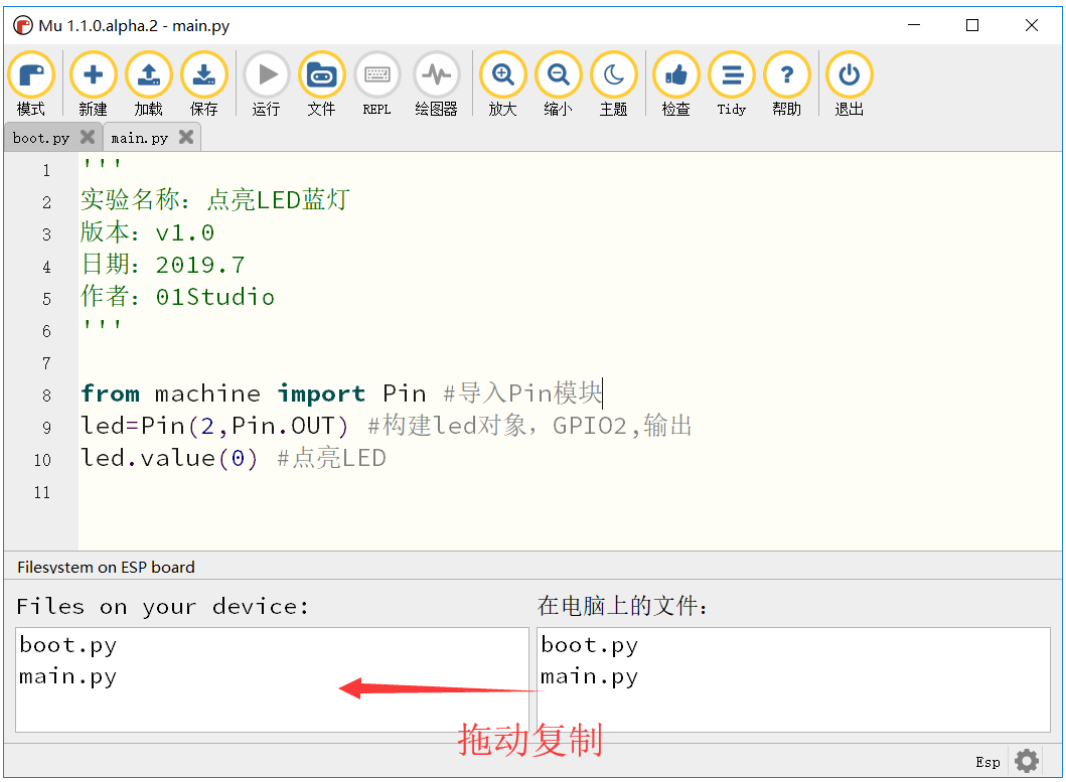


图 4-5 拷贝文件到设备

- **实验结果：**

下载程序，可以看到 LED（蓝灯）成功点亮。

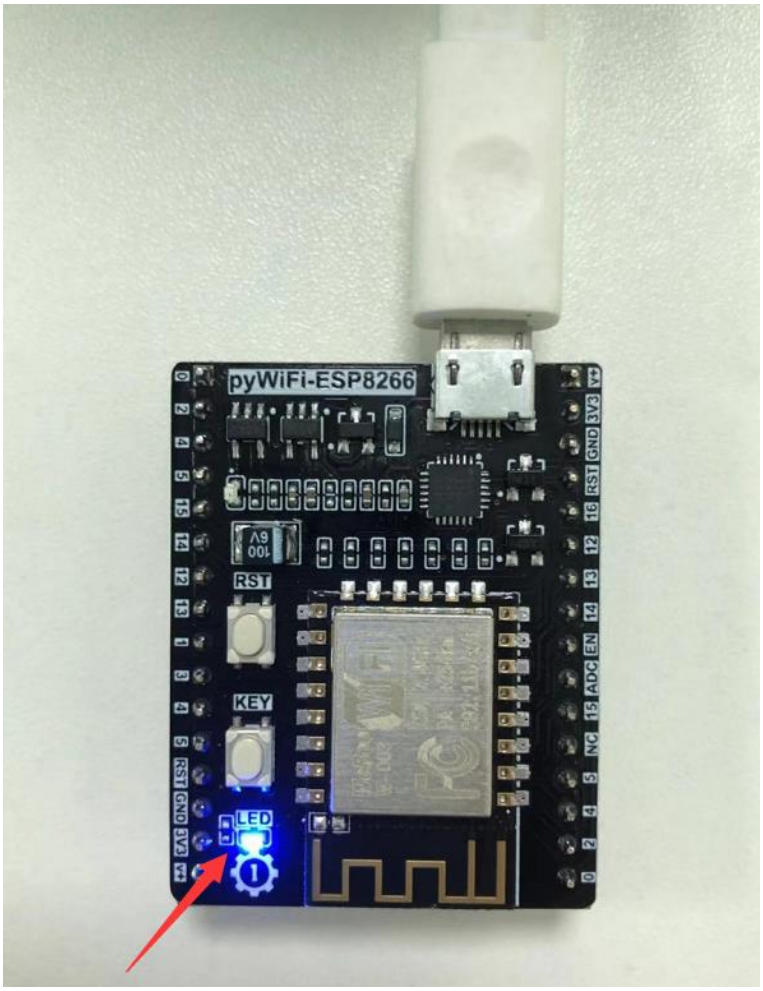


图 4-6　LED 点亮

- **总结：**

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

4.2 按键

- **前言：**

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

- **实验平台：**

pyWiFi-ESP8266。

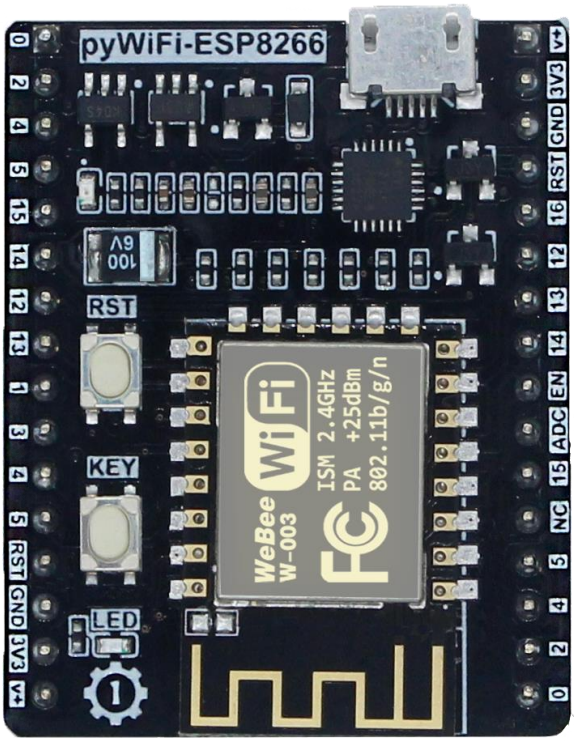


图 4-7 pyWiFi-ESP8266

- **实验目的：**

使用按键功能，通过检测按键被按下后，改变 LED（蓝灯）的亮灭状态。

- **实验讲解：**

pyWiFi-ESP8266 开发板上有 2 个按键，RST 和 KEY，RST 顾名思义是复位用的，所以真正自带可以用的就只有 1 个按键 KEY。

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

构造函数
KEY=machine.Pin(id,mode,pull)
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
KEY.value()
引脚电平值。输入状态：无须参数，返回当前引脚值 0 或者 1。

表 4-2 KEY 对象

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

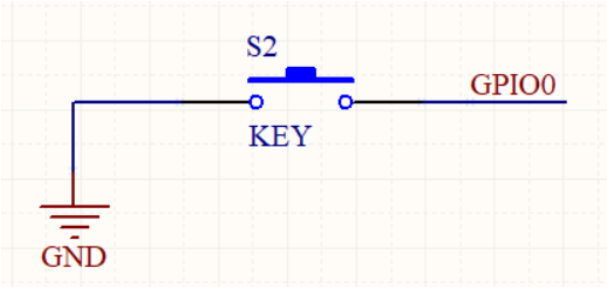


图 4-8 按键原理图

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

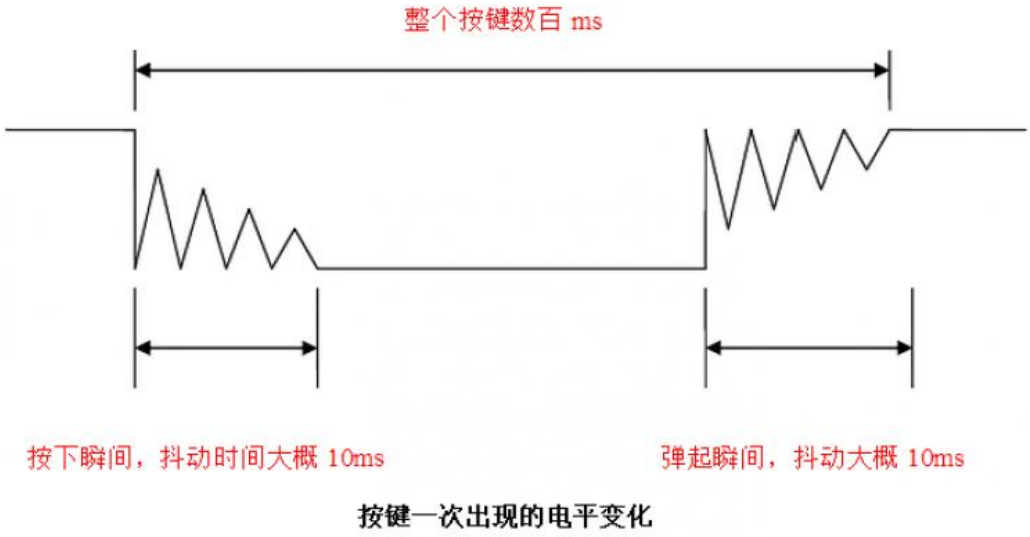


图 4-9 按键按下过程

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

```
import time

time.sleep(1)           # 睡眠 1 秒
time.sleep_ms(500)      # 睡眠 500 毫秒
time.sleep_us(10)       # 睡眠 10 微妙
start = time.ticks_ms() # 获取毫秒计时器开始值

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

编程流程图如下：

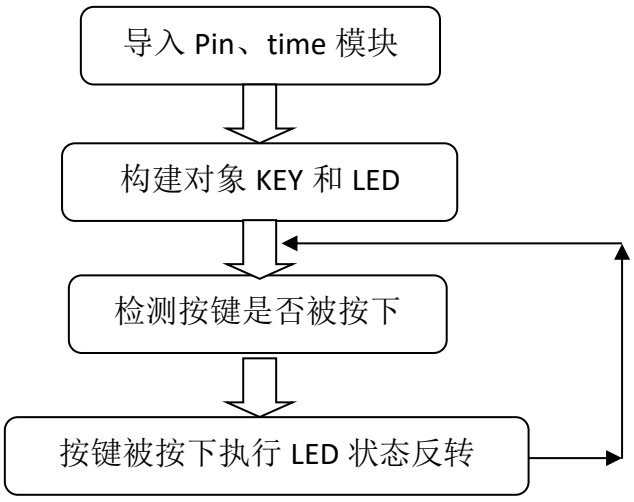


图 4-10 代码编写流程

实验参考代码如下：

```
'''
实验名称：按键
版本：v1.0
日期：2019.7
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,value=1) #构建 LED 对象,开始熄灭
KEY=Pin(0,Pin.IN,Pin.PULL_UP) #构建 KEY 对象
state=1 #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 #改变 state 的值 0,1 切换
            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~ 是数值取反操作，会导致出错。

- **实验结果：**

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

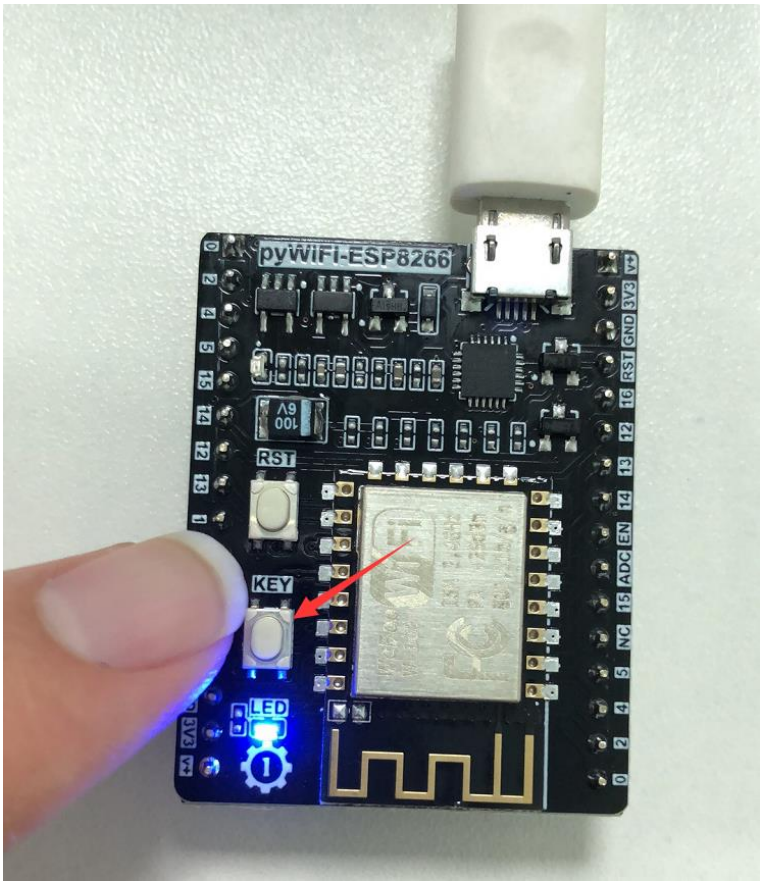


图 4-11

- **总结：**

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

4.3 外部中断

- **前言：**

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

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

- **实验平台：**

pyWiFi-ESP8266。

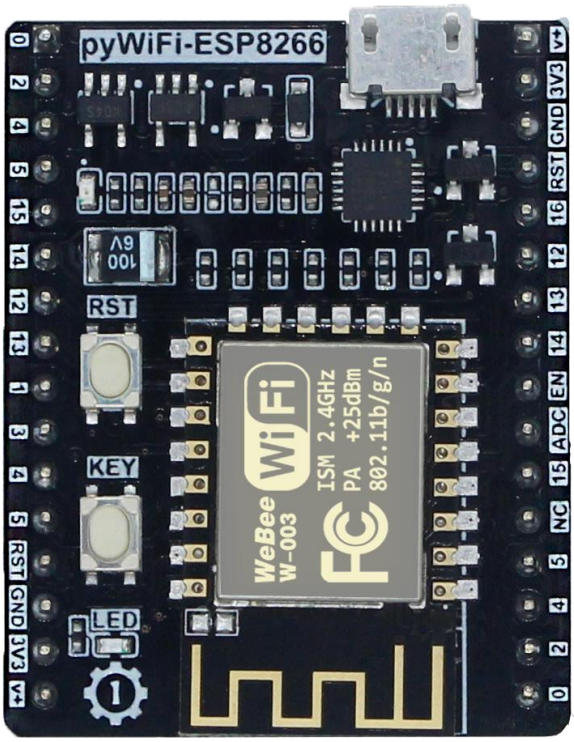


图 4-12　pyWiFi-ESP8266

- **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
KEY=machine.Pin(id,mode,pull)
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
KEY.irq(handler,trigger)
配置中断模式。 handler:中断执行的回调函数； trigger: 触发中断的方式，共 4 种，分别是 Pin.IRQ_FALLING（下降沿触发）、Pin.IRQ_RISING（上升沿触发）、Pin.IRQ_LOW_LEVEL（低电平触发）、Pin.IRQ_HIGH_LEVEL（高电平触发）。

表 4-3 按键对象

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

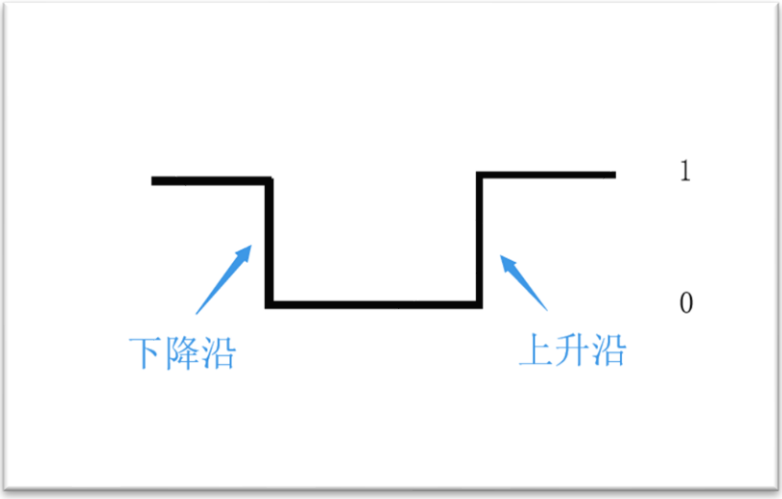


图 4-13 边沿触发

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

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

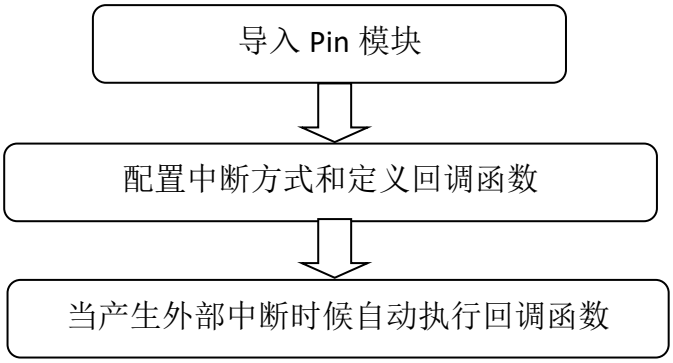


图 4-14 代码编写流程

参考程序代码如下：

```
'''
实验名称：外部中断
版本：v1.0
日期：2019.7
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,value=1) #构建 LED 对象,开始熄灭
KEY=Pin(0,Pin.IN,Pin.PULL_UP) #构建 KEY 对象
state=1  #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，需要将引脚对象 KEY 传递进去。

- **实验结果：**

将 main.py 文件拷贝到设备，复位。

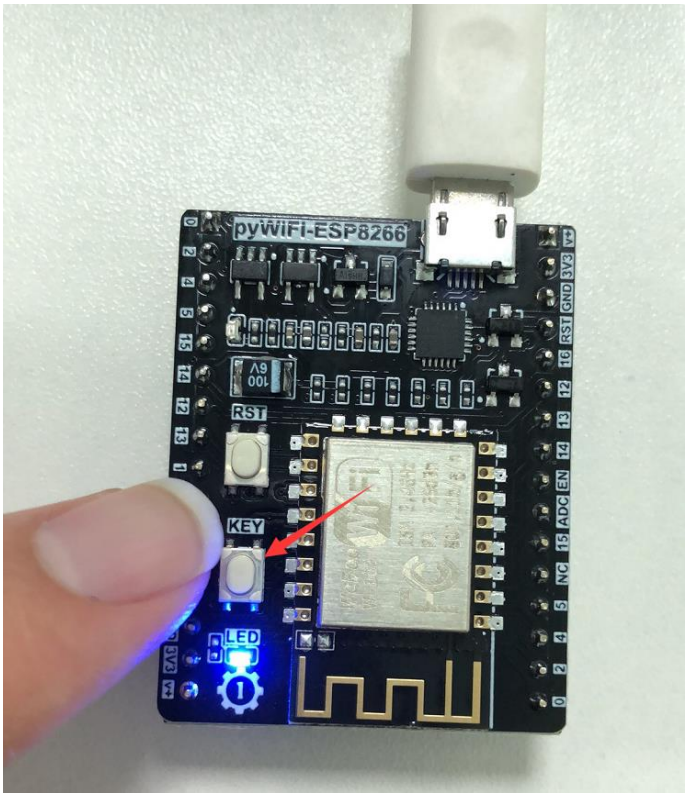


图 4-15 实验现象

- **总结：**

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

4.4 定时器

- **前言：**

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

- **实验平台：**

pyWiFi-ESP8266。

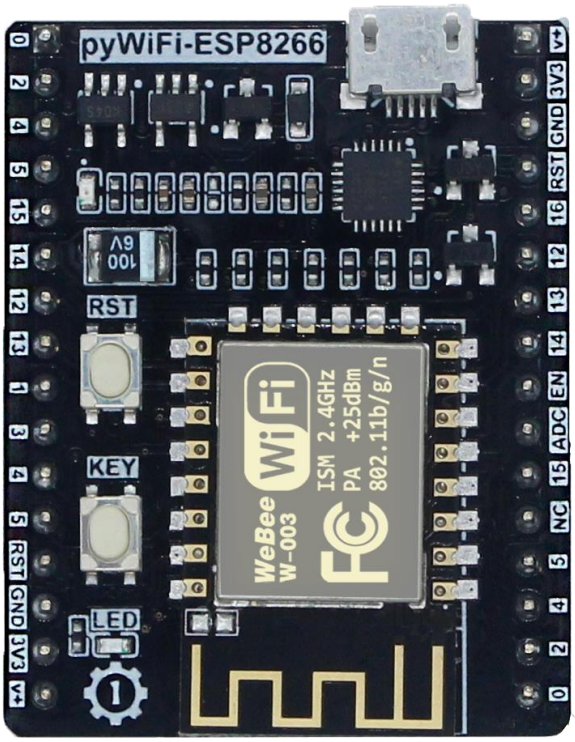


图 4-16 pyWiFi-ESP8266

- **实验目的：**

通过定时器让 LED 周期性每秒闪烁 1 次。

- **实验讲解：**

ESP8266 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
tim=machine.Timer(-1)
构建定时器对象。RTOS 定时器编号为-1；
使用方法
tim.init(period,mode,callback)
定时器初始化。period:单位为 ms；mode：2 种工作模式，Timer.ONE_SHOT（执行一次）、Timer.PERIODIC（周期性）；callback:定时器中断后的回调函数。

表 4-4 定时器对象

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

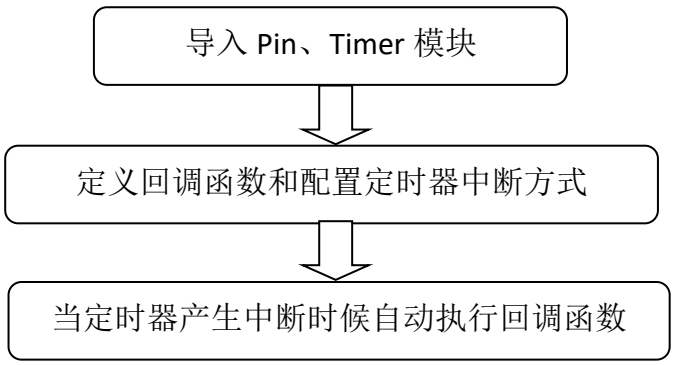


图 4-17 代码编写流程

参考代码如下：

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

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

def fun(tim):

    global Counter
    Counter = Counter + 1
    print(Counter)
    led.value(Counter%2)

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

- 实验结果：

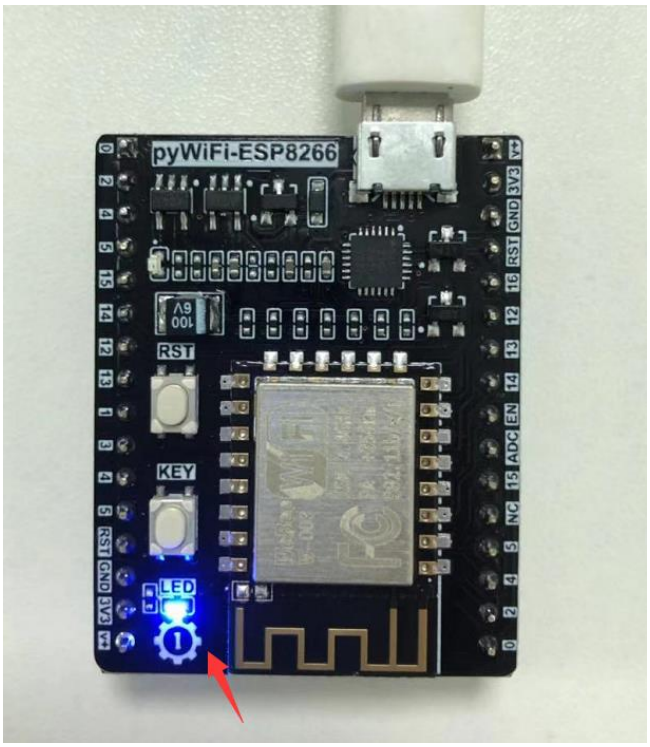


图 4-18　LED 以周期 1 秒的间隔闪烁

- **总结：**

本节实验介绍了 RTOS 定时器的使用方式，有用户可能会认为使用延时延时也可以实现这个功能，但相比于延时函数，定时器的好处就是不占用 CPU 资源。RTOS（实时操作系统）的原理就是利用多个定时器分别执行不同任务方式实现。

有兴趣的用户也可以多定义几个定时器对象 tim2,tim3，定时器标号还是-1，通过不同的周期和参数配置实现多任务的操作。

4.5 I2C 总线（OLED 显示屏）

- **前言：**

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

本章节的OLED显示屏学习，实际上是在使用I2C的总线接口，pyWiFi-ESP8266是通过 I2C 总线与 OLED 显示屏通讯的。这章稍微复杂一点，但我们把它放在了前面来学习，是因为学会了显示屏的使用，那么在后面的实验中可玩性就更强了。

- **实验平台：**

pyWiFi-ESP8266 和 pyBase 开发底板。

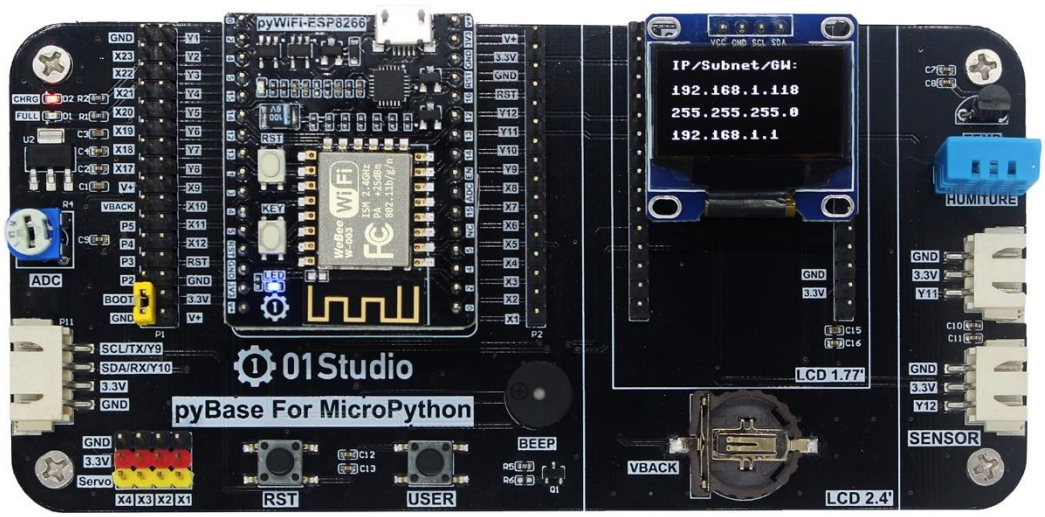


图 4-19 pyWiFi-ESP8266 开发套件

- **实验目的：**

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

- **实验讲解：**

什么是 I2C?

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

什么是 OLED 显示屏？

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

图 4-20　I2C 接口的 OLED

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

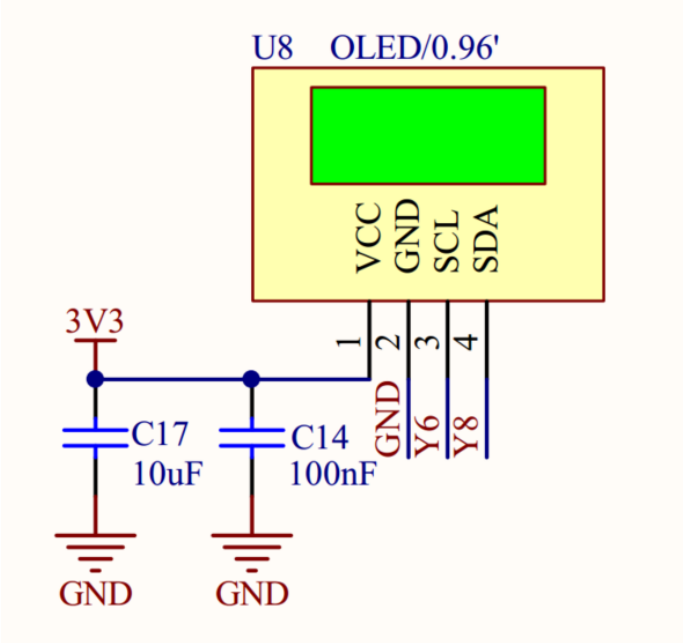


图 4-21　OLED 接口原理图

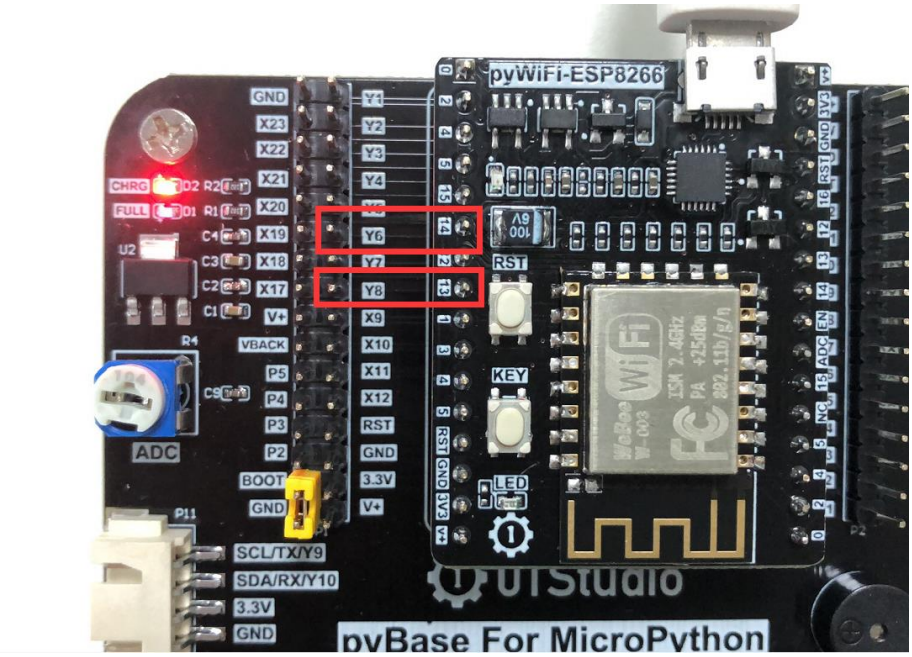


图 4-22　pyWiFi-ESP8266 和 pyBase 的连线图

从上图可见连接到 OLED 的别是 Y6→SCL 和 Y8→SDA。跟 pyWiFi-ESP8266 的 14、13 引脚相连。本实验将使用 MicroPython 的 Machine 模块来定义 Pin 口和 I2C 初始化。具体如下：

构造函数
i2c = machine.I2C(scl,sda)
构建 I2C 对象。scl:时钟引脚；sda:数据引脚。
使用方法
i2c.scan()
扫描 I2C 总线的设备。返回地址，如：0x3c；
i2c.readfrom(addr,nbytes)
从指定地址读数据。addr:指定设备地址；nbytes:读取字节数；
i2c.write(buf)
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：http://docs.micropython.01studio.org/

表 4-5　I2C 对象

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

构造函数
oled = SSD1306_I2C(width, height, i2c, addr)
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
`oled.text(string,x,y)`
将 string 字符写在指定为位置。string：字符；x:横坐标；y:纵坐标。
`oled.show()`
执行显示。
`oled.fill(RGB)`
清屏。RGB：`0` 表示黑色，`1` 表示白色。

图 4-23 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：

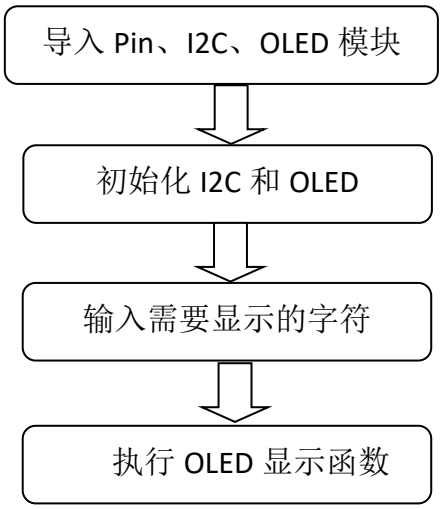


图 4-24 代码编写流程

main.py 文件参考代码如下：

```
'''
实验名称：I2C 总线(OLED 显示屏)
版本：v1.0
日期：2019.7
作者：01Studio
'''

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

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

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

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

oled.show()   #OLED 执行显示
```

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

另外记得将我们提供的示例代码中的 ssd1306.py 驱动文件拷贝到 pyWiFi-ESP8266 的文件系统下，跟 main.py 保持同一个路径。

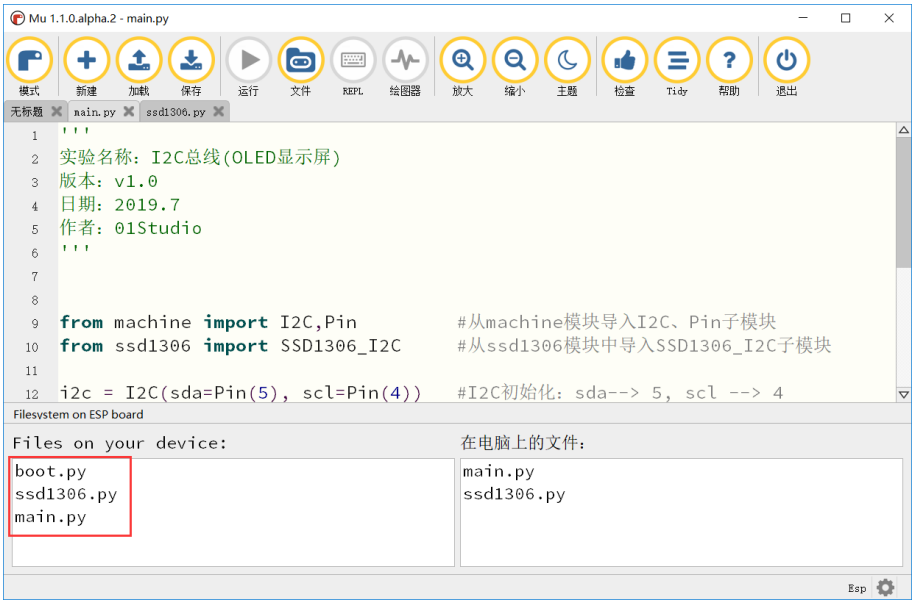


图 4-25 将 main.py、ssd1306.py 拷贝到设备

- **实验结果：**

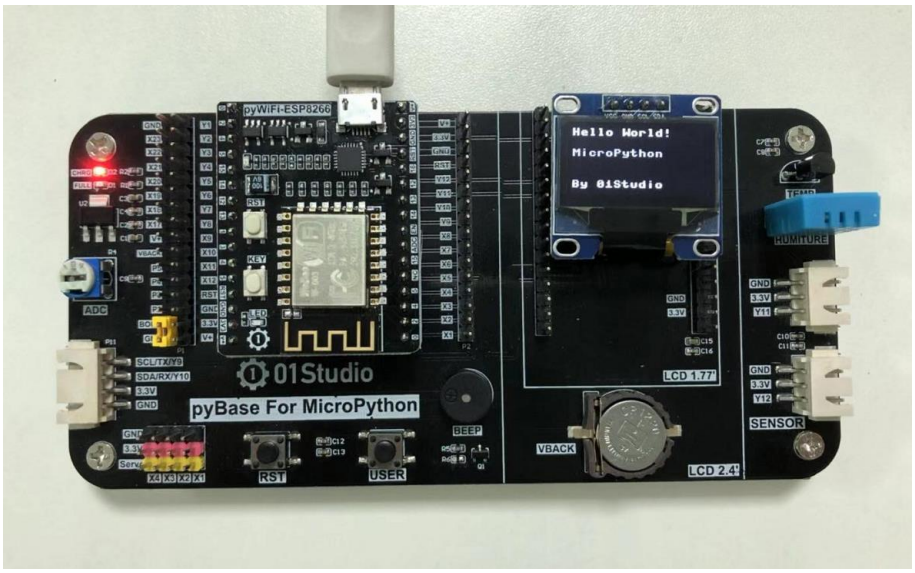


图 4-26 OLED 实验结果

- **总结：**

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

就好。

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

4.6 RTC 实时时钟

- **前言：**

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

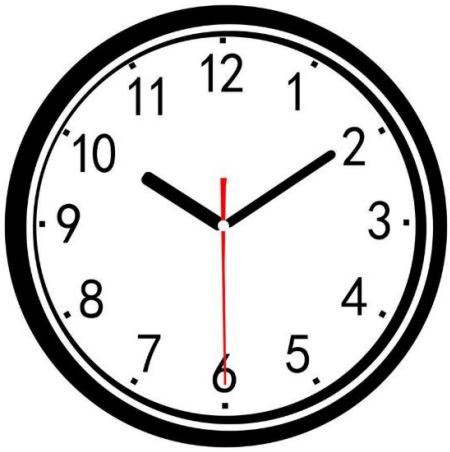


图 4-27 时钟

- **实验平台：**

pyWiFi-ESP8266 和 pyBase 开发底板。

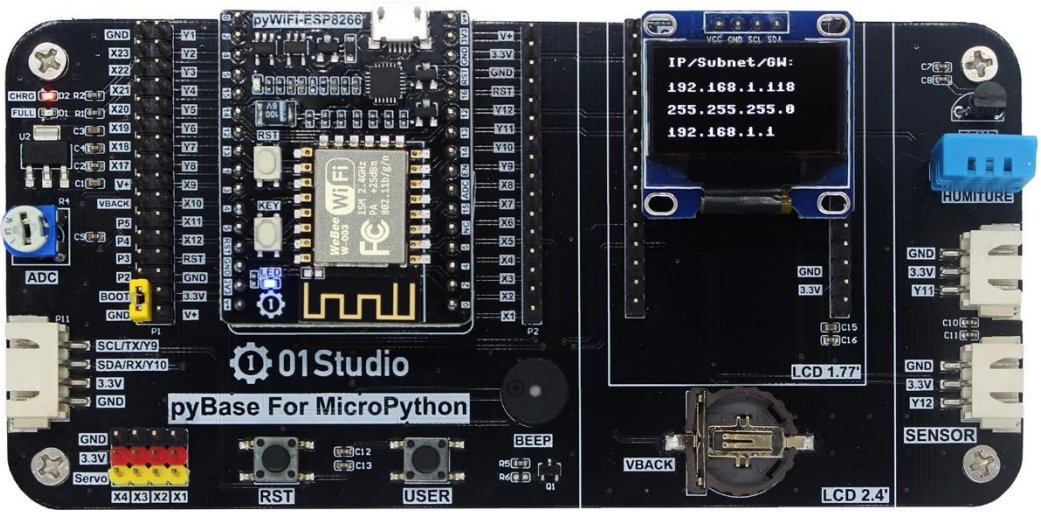


图 4-28 pyWiFi-ESP8266 开发套件

- **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
rtc=machine.RTC()
构建 RTC 对象。
使用方法
`rtc.datetime((2019, 4, 1, 0, 0, 0, 0, 0))`
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。
`rtc.datetime()`
获取当前日期和时间。

表 4-6

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：

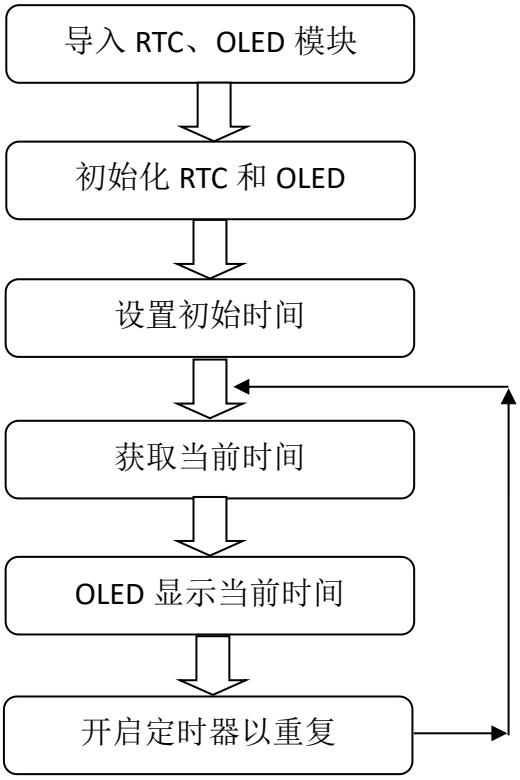


图 4-29 代码编写流程图

实验参考代码如下：

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

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

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

```
time_list = ['', '', '']

# 初始化所有相关对象
i2c = I2C(sda=Pin(13), scl=Pin(14)) #I2C 初始化：sda-->13, scl --> 14
oled = SSD1306_I2C(128, 64, i2c, addr=0x3c)
rtc = RTC()

# 首次上电配置时间，按顺序分别是：年，月，日，星期，时，分，秒，次秒级；这里做
#了一个简单的判断，检查到当前年份不对就修改当前时间，开发者可以根据自己实际情况
#来修改。
if rtc.datetime()[0] != 2019:
    rtc.datetime((2019, 4, 1, 0, 0, 0, 0, 0))

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

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

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

# 显示时间需要判断时、分、秒的值否小于 10，如果小于 10，则在显示前面补“0”以
# 达到较佳的显示效果
    for i in range(4, 7):
        if datetime[i] < 10:
            time_list[i - 4] = "0"
        else:
```

```
            time_list[i - 4] = ""


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


    oled.show()


#开启 RTOS 定时器
tim = Timer(-1)
tim.init(period=300, mode=Timer.PERIODIC, callback=RTC_Run) #周期 300ms
```

由于实验要用到 OLED 显示屏，所以同样别忘了将示例代码该实验文件夹下的 ssd1306.py 文件复制到 pyWiFi-ESP8266 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。

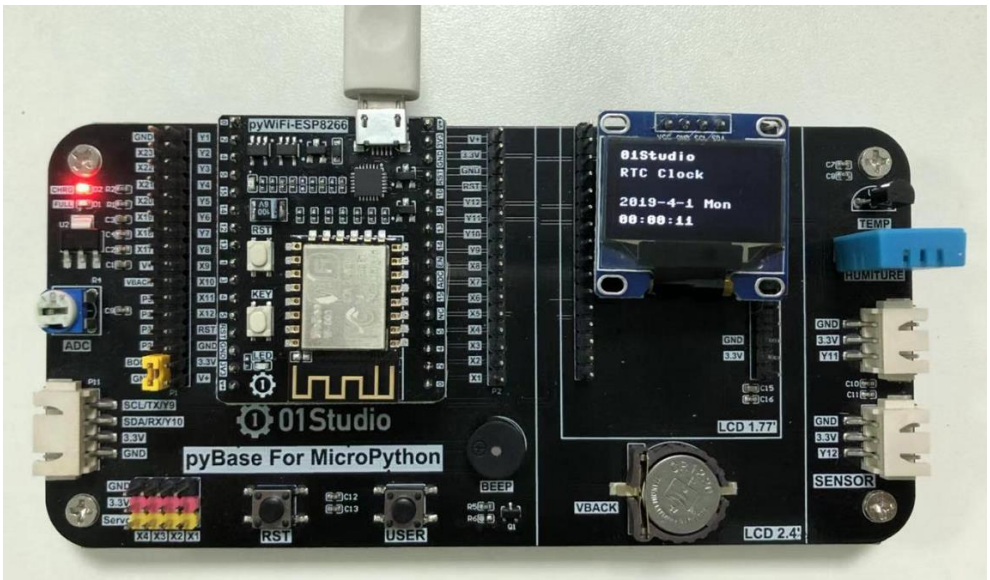


图 4-30

由于 ESP8266 没有后备电池引脚，所以不支持掉电保存。因此 pybase 上面的纽扣电池是不起作用的。

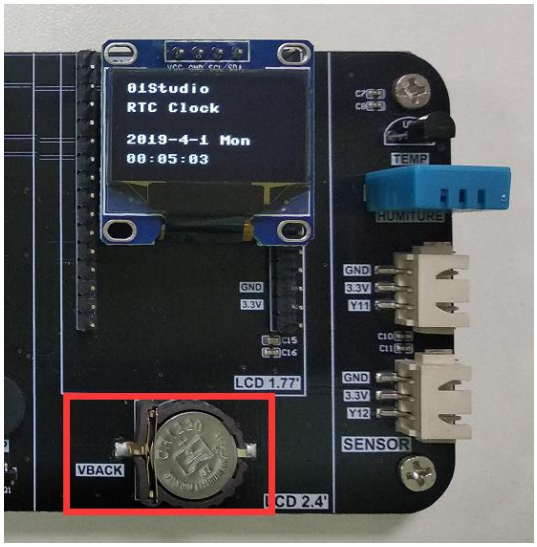


图 4-31 后备电池（VBACK）

- **总结：**

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

4.7 ADC

- **前言：**

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

- **实验平台：**

pyWiFi-ESP8266 和 pyBase 开发底板。

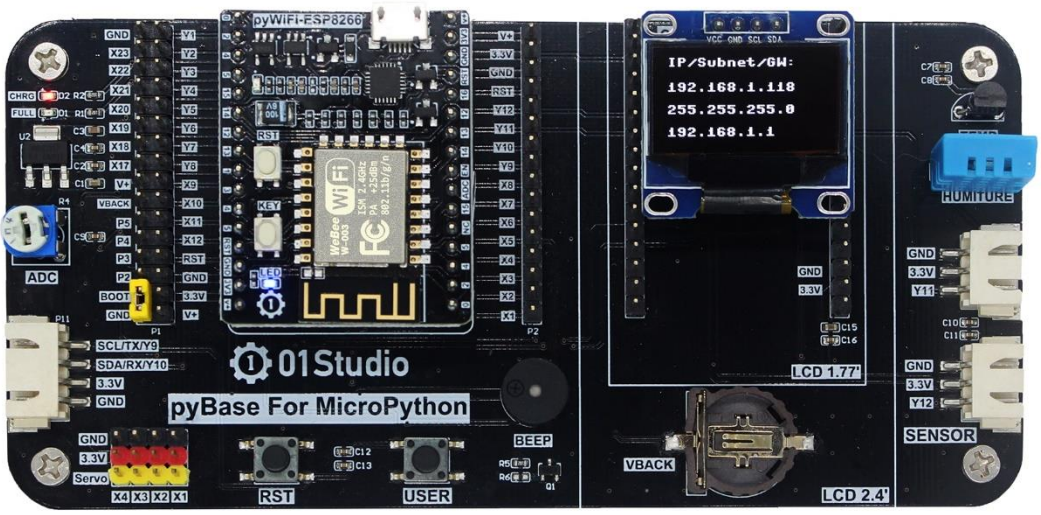


图 4-32 pyWiFi-ESP8266 开发套件

- **实验目的：**

通过编程调用 MicroPython 的内置 ADC 函数，实现测量 0-1V 电压，并显示到屏幕上。

- **实验讲解：**

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

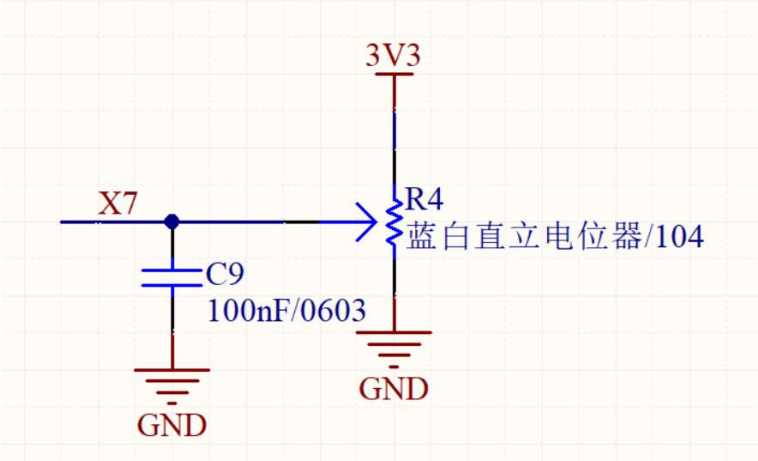


图 4-33 电位器原理图

但由于 ESP8266 的 ADC 输入引脚只能测量 0-1V 的量程，因此本实验的测量电压范围为 0-1V。我们来看看 ADC 模块的构造函数和使用方法。

构造函数
adc=machine.ADC(0)
构建 ADC 对象。0:ESP8266 只有 1 个 ADC
使用方法
adc.read()
获取 ADC 值。测量精度是 10 位，返回 0-1024（表示 0-1V）。

表 4-7 ADC 对象

你没看错，就这么简单。两句函数就可以获得 ADC 数值，让我们来理顺一下编程逻辑。先导入相关模块，然后初始化模块。在循环中不断读取 ADC 的值，转化成电压值后在 OLED 上面显示，每隔 300 毫秒读取一次，具体如下：

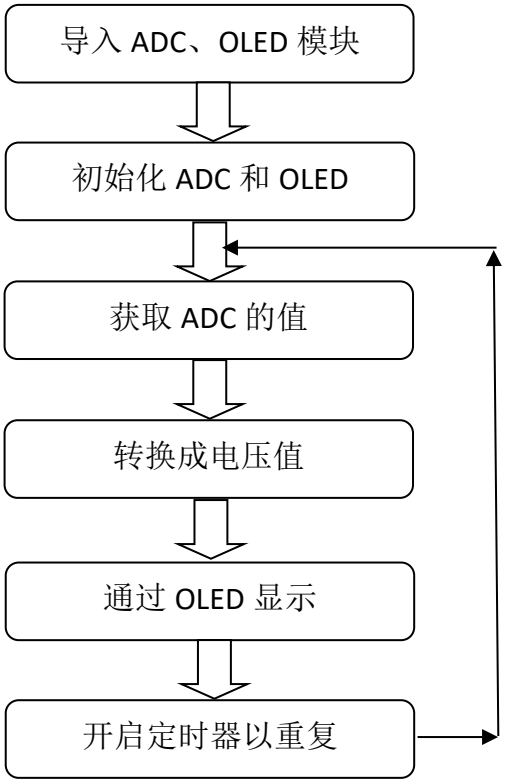


图 4-34 代码编写流程

实验参考代码：

```
'''
实验名称：ADC-电压测量
版本：v1.0
日期：2019.7
作者：01Studio
说明：通过对 ADC 数据采集，转化成电压在显示屏上显示。ADC 精度 10 位，电压 0-1V。
'''

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

```
#初始化相关模块
i2c = I2C(sda=Pin(13), scl=Pin(14))  #I2C 初始化：sda--> 13, scl --> 14
oled = SSD1306_I2C(128, 64, i2c, addr=0x3c)
adc = ADC(0) #Pin='AD'

def ADC_Test(tim):

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

    #获取 ADC 数值
    oled.text(str(adc.read()),0,40)
    oled.text('(1024)',40,40)

    #计算电压值，获得的数据 0-1024 相当于 0-1V，（'%.2f'%）表示保留 2 位小数
    oled.text(str('%.2f'%(adc.read()/1024)),0,55)
    oled.text('V',40,55)

    oled.show()

#开启 RTOS 定时器
tim = Timer(-1)

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

- **实验结果：**

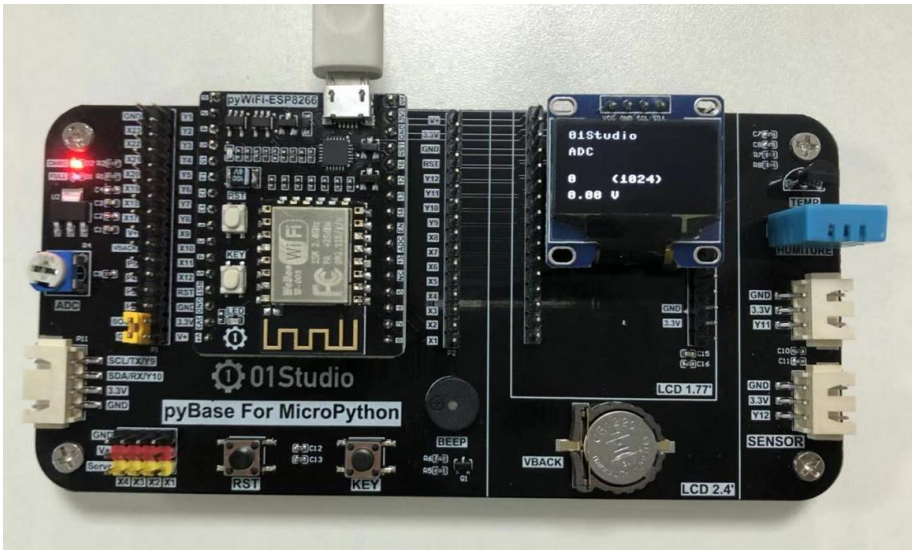


图 4-35 电位器顺时钟拧到尽头是 0V

通过调节电位器，可以发现电压在不断变化。由于 ESP8266 电压测量量程限制，当超出 1V 量程后，始终显示电压在 1V，测量值是 1024。

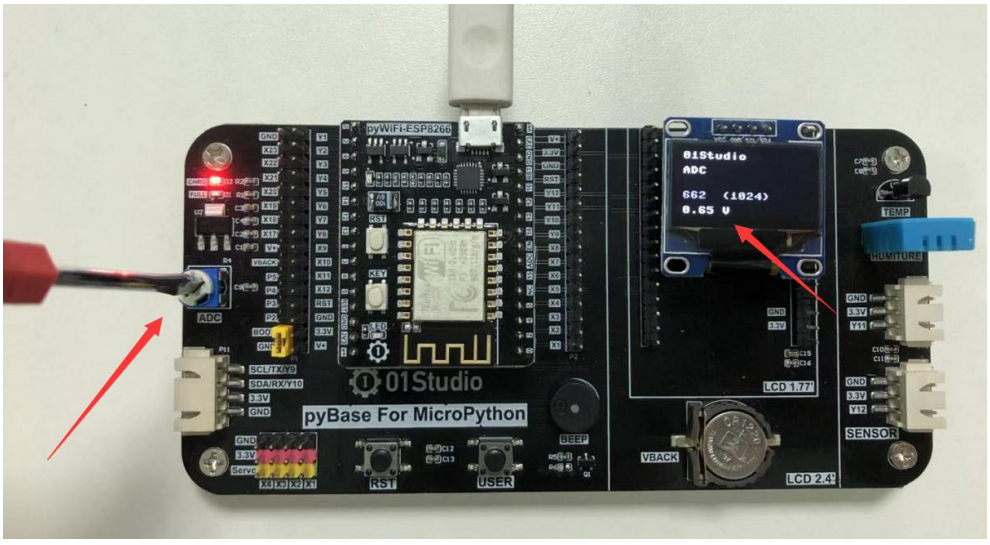


图 4-36 调节电位器来改变电压输入

- **总结：**

这一节我们学习了 ADC 的应用，主要用于电压的测量。

4.8 PWM

- **前言：**

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

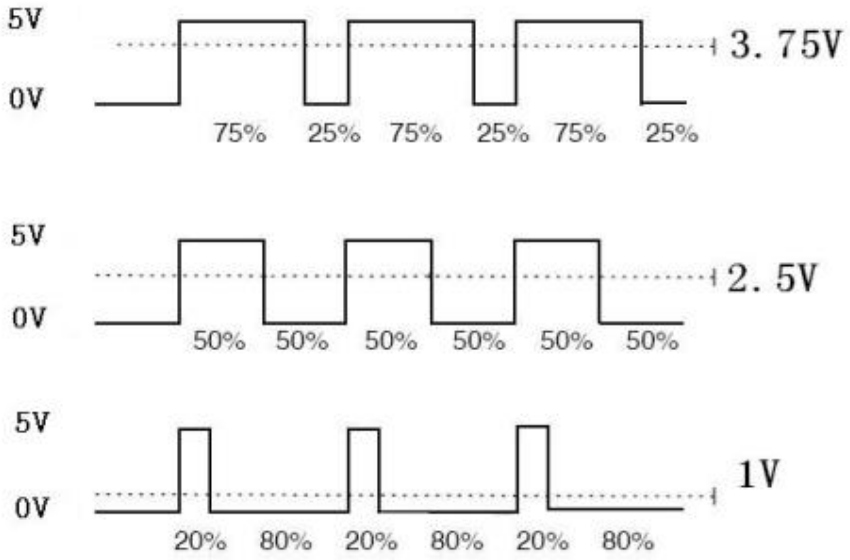


图 4-37 PWM 波形示例

- **实验平台：**

pyWiFi-ESP8266 和 pyBase 开发底板。

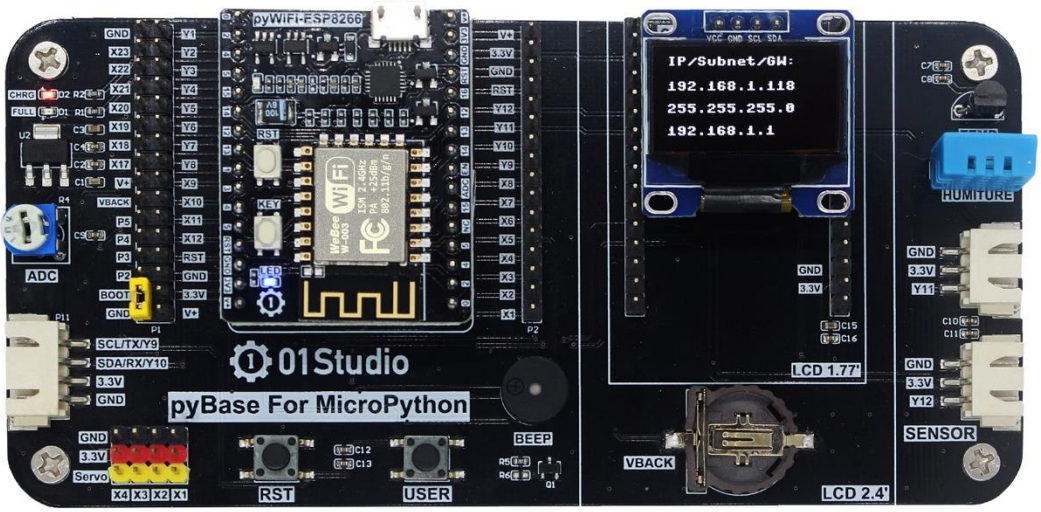


图 4-38 pyWiFi-ESP8266 开发套件

- **实验目的：**

通过不同频率的 PWM 信号输出，驱动无源蜂鸣器发出不同频率的声音。

- **实验讲解：**

蜂鸣器分有源蜂鸣器和无源蜂鸣器，有源蜂鸣器的使用方式非常简单，只需要接上电源，蜂鸣器就发声，断开电源就停止发声。而本实验用到的无源蜂鸣器，是需要给定指定的频率，才能发声的，而且可以通过改变频率来改变蜂鸣器的发声音色，以此来判定 pyWiFi-ESP8266 的 PWM 输出频率是在变化的。

pyBase 开发底板上的无源蜂鸣器连接到引脚 X5。如下图所示：

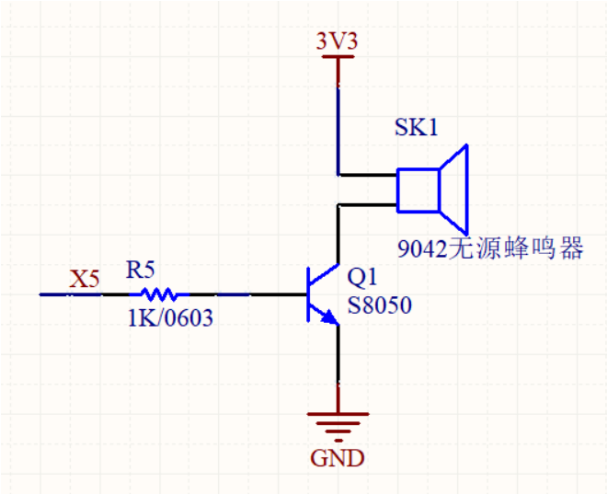


图 4-39 蜂鸣器原理图

PWM 可以通过 ESP8266 所有引脚输出除了 Pin(16). 所有通道都有 1 个特定的频率，从 1 到 1000 之间（单位是 Hz）。占空比的值为 0 至 1023 之间。在本实验中我们用到引脚 15。

先看看 PWM 模块对象：

构造函数
pwm15=machine.PWM(machine.Pin(id),freq,duty)
构建 PWM 对象。id:引脚编号；freq:频率值；duty:占空比；配置完后 PWM 自动生效。
使用方法
`pwm15.freq(freq)`
设置频率。`freq`:频率值在 `1-1000` 之间，`freq` 为空时表示获取当前频率值。

pwm15.duty(duty)
设置占空比。duty:占空比在 0-1023 之间，duty 为空时表示获取当前占空比值。
pwm15.deinit()
关闭 PWM。

表 4-8　PWM 对象

无源蜂鸣器我们可以用特定频率的方波来驱动，方波的原理很简单，就是一定频率的高低电平转换，可以简单理解成占空比为 50%的 PWM 输出。

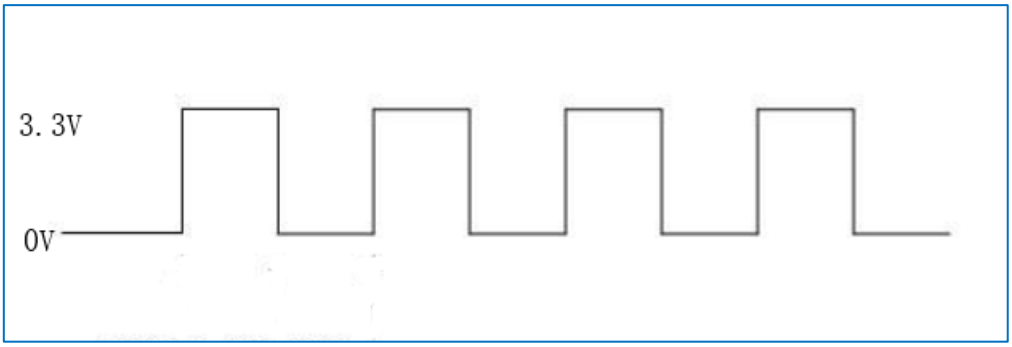


图 4-40　方波信号

结合上述讲解，总结出代码编写流程图如下：

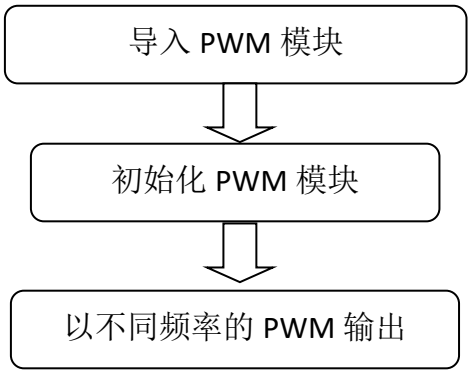


图 4-41 代码编写流程图

实验参考代码：

```
'''
实验名称：PWM
版本：v1.0
日期：2019.7
作者：01Studio
说明：通过不同频率的 PWM 信号输出，驱动无源蜂鸣器发出不同频率的声音。
'''

from machine import Pin, PWM
import time

Beep = PWM(Pin(15), freq=0, duty=512) # 在同一语句下创建和配置 PWM

#蜂鸣器发出频率 200Hz 响声
Beep.freq(200)
time.sleep_ms(1000)

#蜂鸣器发出频率 400Hz 响声
Beep.freq(400)
time.sleep_ms(1000)

#蜂鸣器发出频率 600Hz 响声
Beep.freq(600)
time.sleep_ms(1000)


#蜂鸣器发出频率 800Hz 响声
Beep.freq(800)
time.sleep_ms(1000)
```

```
#蜂鸣器发出频率 1000Hz 响声
Beep.freq(1000)
time.sleep_ms(1000)

#停止
Beep.deinit()
```

- **实验结果：**

由于 pyBase 上的 X5 对应 pyWiFi-ESP8266 的引脚是 NC 悬空引脚。X6 对应 15 引脚。因此我们可以用一个跳线帽将 pybase 开发底板的 X5、X6 引脚短接。以实现引脚 15 连接到无源蜂鸣器。

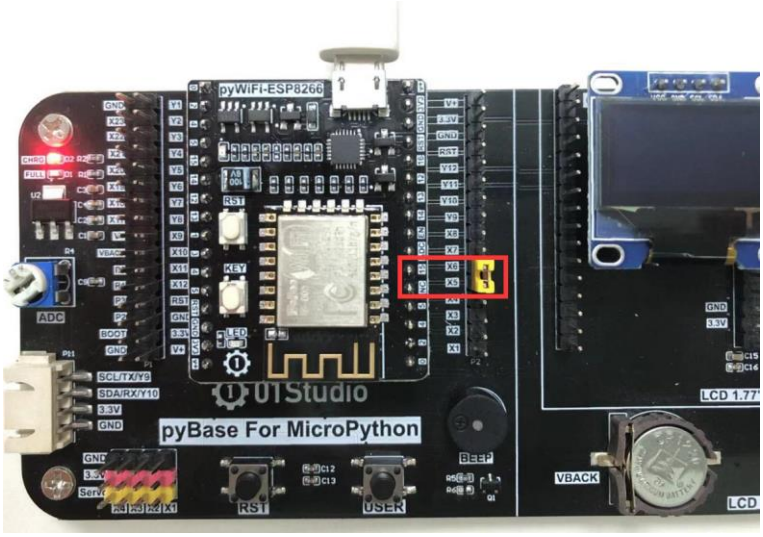


图 4-42 用跳线帽将 pyBase 的 X5、X6 短接

下载程序，复位后可以听到蜂鸣器依次发出不同频率的响声。

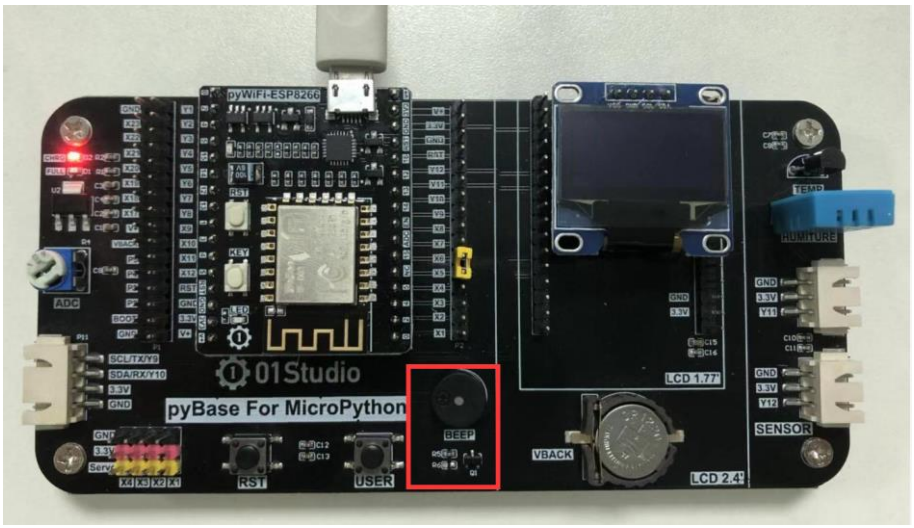


图 4-43 实验现象

有条件的朋友可以使用示波器测量 pyWiFi-ESP8266 的 15 引脚或 pyBase 的 X5 接口，观察信号波形的变化：

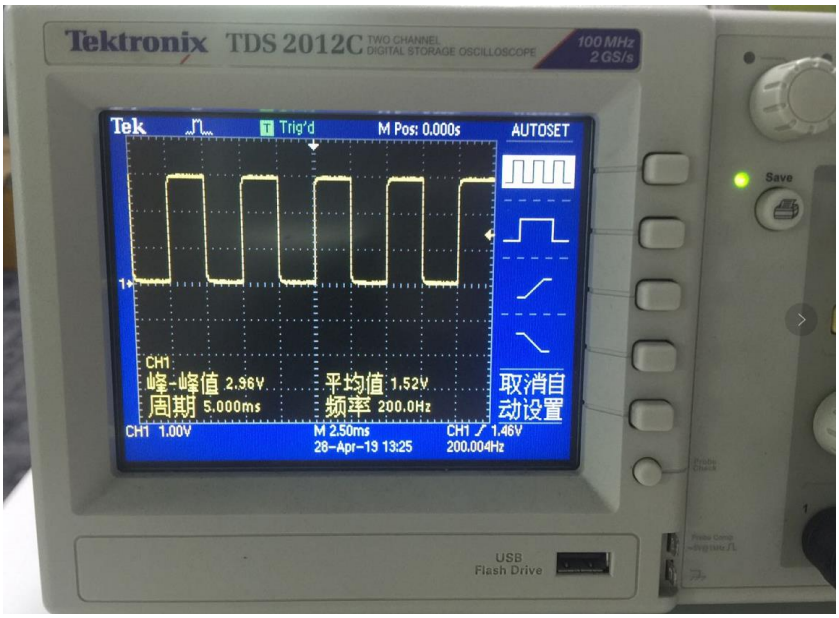


图 4-44 示波器显示波形

- **总结：**

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

第5章　传感器实验

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

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

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

5.1 温度传感器 DS18B20

- **前言：**

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

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

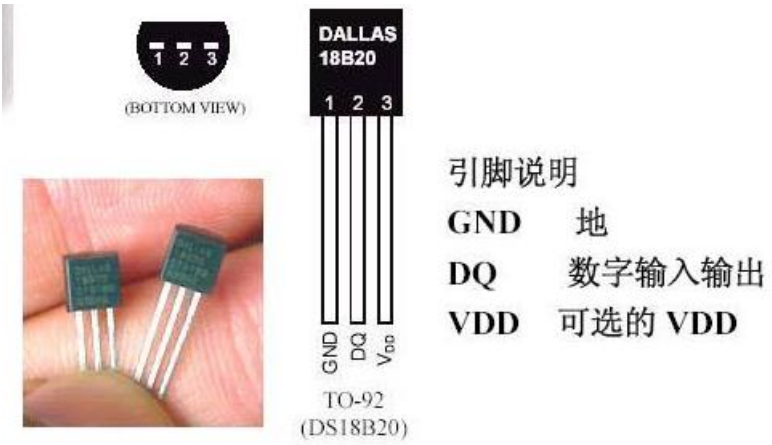


图 5-1 DS18B20 传感器

图 5-2 DS18B20 金属探头封装

- **实验平台：**

pyWiFi-ESP8266 开发套件。DS18B20 温度传感器位于 pyBase 右上方。

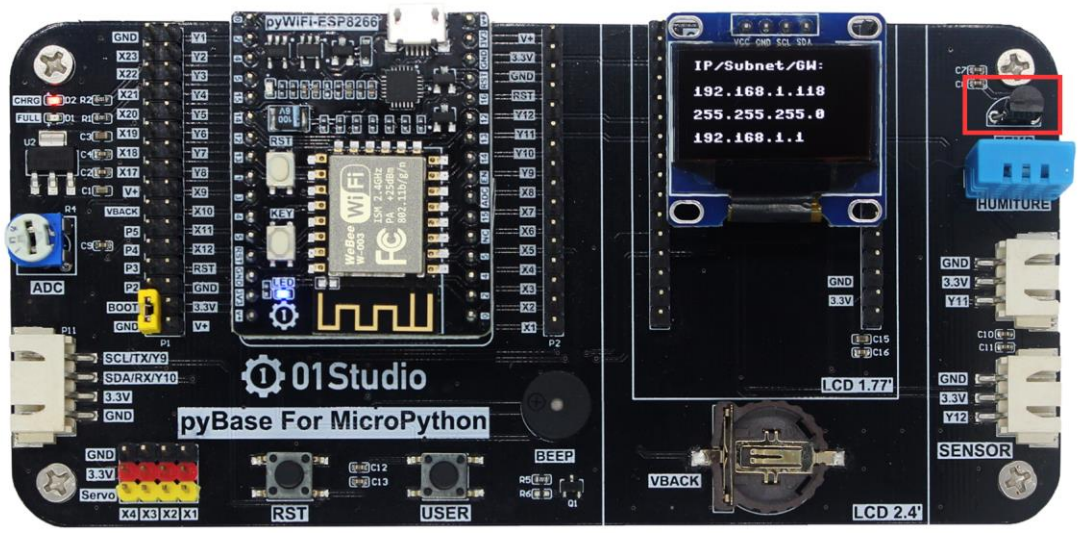


图 5-3　pyWiFi-ESP8266 开发套件

- **实验目的：**

通过编程采集温度数据，并在 OLED 上显示。

- **实验讲解：**

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

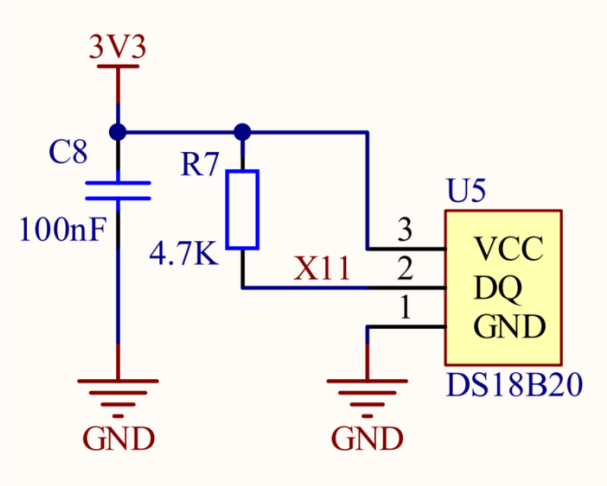


图 5-4 DS18B20 接线原理图

可以看到 DS18B20 传感器连接到了 pyBase 的 X11 引脚上。也就是说连接到 pyWiFi-ESP8266 的引脚 4 上。

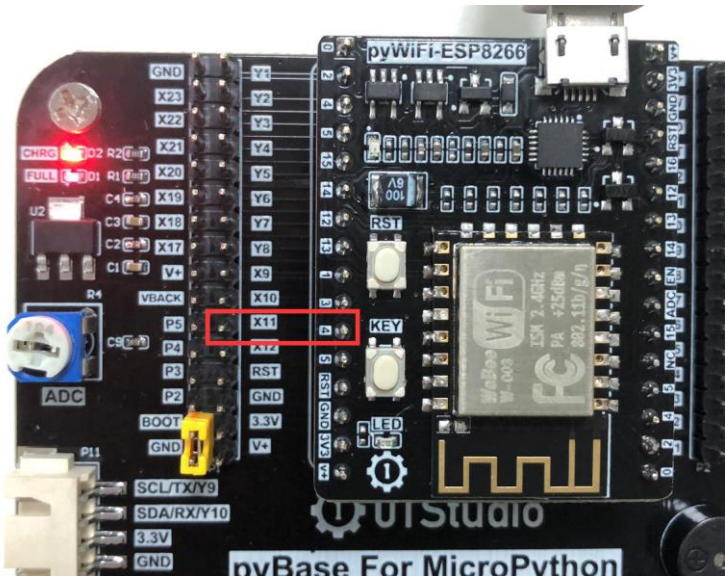


图 5-5 用跳线帽将 pyBase 的 X19、Y6 短接

也就是说我们需要针对引脚 4 编写程序来驱动 DS18B20。那么我们需要自己来编写驱动么？如果你有兴趣的可以自己尝试一下。这部分我们 01Studio 已经收集整理和编写好了，单总线模块文件是：onewire.py，DS18B20 模块的文件是 ds18x20.py。如果你学习过前面基于 STM32 平台应该不陌生。而对于 ESP8266,这两个模块已经集成到了初始化固件中，也就是说我们可以直接在 main.py 导入模块并调用即可！

单总线模块（onewire）说明如下：

构造函数
ow=onewire.OneWire(machine.Pin(id))
构建单总线对象。id:引脚编号；
使用方法
ow.scan()
扫描总线上的设备。返回设备地址，支持多设备同时挂载。
ow.reset()

总线设备复位。
ow.readbyte()
读 1 个字节。
ow.writebyte(0x12)
写入 1 个字节。
ow.write('123')
写入多个字节。
ow.select_rom(b'12345678')
根据 ROM 编号选择总线上指定设备。

表 5-1　单总线对象

ds18x20 模块说明如下：

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

表 5-2　DS18B20 对象

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

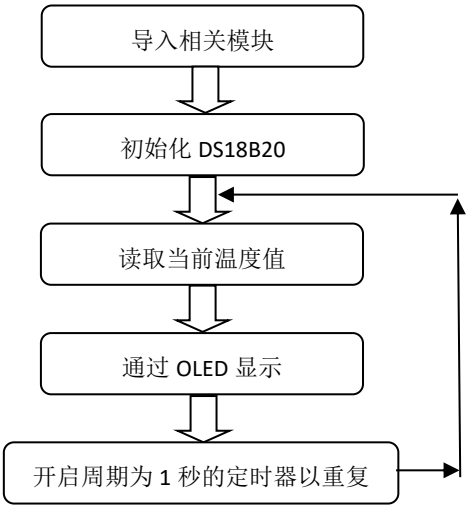


图 5-6 代码编写流程图

实验参考代码：

```
'''
实验名称：温度传感器 DS18B20
版本：v1.0
日期：2019.7
作者：01Studio
说明：通过编程采集温度数据，并在 OLED 上显示。。
'''

#引用相关模块
from machine import Pin,I2C,Timer
from ssd1306 import SSD1306_I2C
import onewire,ds18x20

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

```
#初始化 DS18B20
ow= onewire.OneWire(Pin(4))  #使能单总线
ds = ds18x20.DS18X20(ow)      #传感器是 DS18B20
rom = ds.scan()          #扫描单总线上的传感器地址，支持多个传感器同时连接

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

    #OLED 数据显示
    oled.fill(0)   #清屏背景黑色
    oled.text('MicroPython', 0, 0)
    oled.text('Temp test:',0,20)
    oled.text(str('%.2f'%temp)+' C',0,40) #显示 temp,保留 2 位小数
    oled.show()

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

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

- **实验结果：**

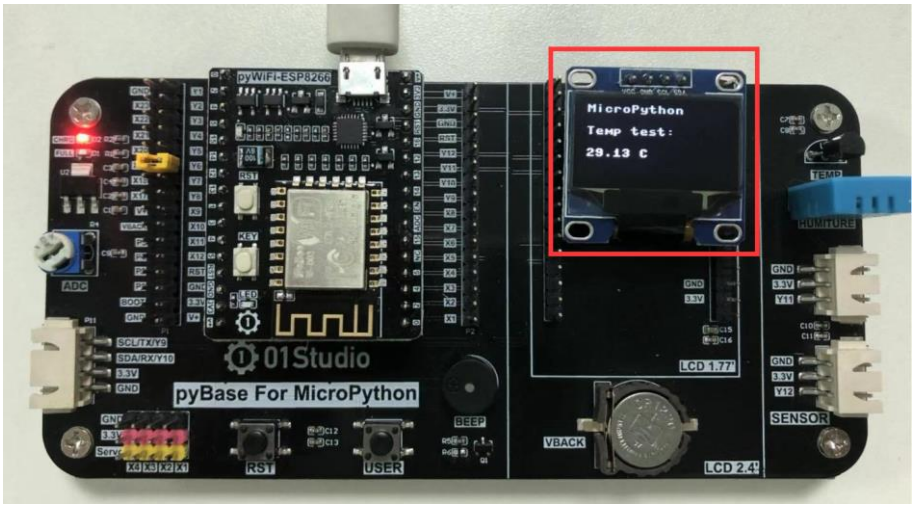


图 5-7 温度检测实验

- **实验拓展：**

pyBase 开发底板预留了外界传感器接口，只要接线正确就可以进行更多的传感器实验。我们将带金属探头的 DS18B20 传感器接到 pyBase 右侧上面的传感器母座，其连接到 pyBase 的“Y11”引脚,也就是对应 pyWiFi-ESP8266 的引脚 12。所以只要将原程序代码的 ow= onewire.OneWire(Pin(4)) 改成 ow= onewire.OneWire(Pin(12))，即可驱动外接的 DS18B20。

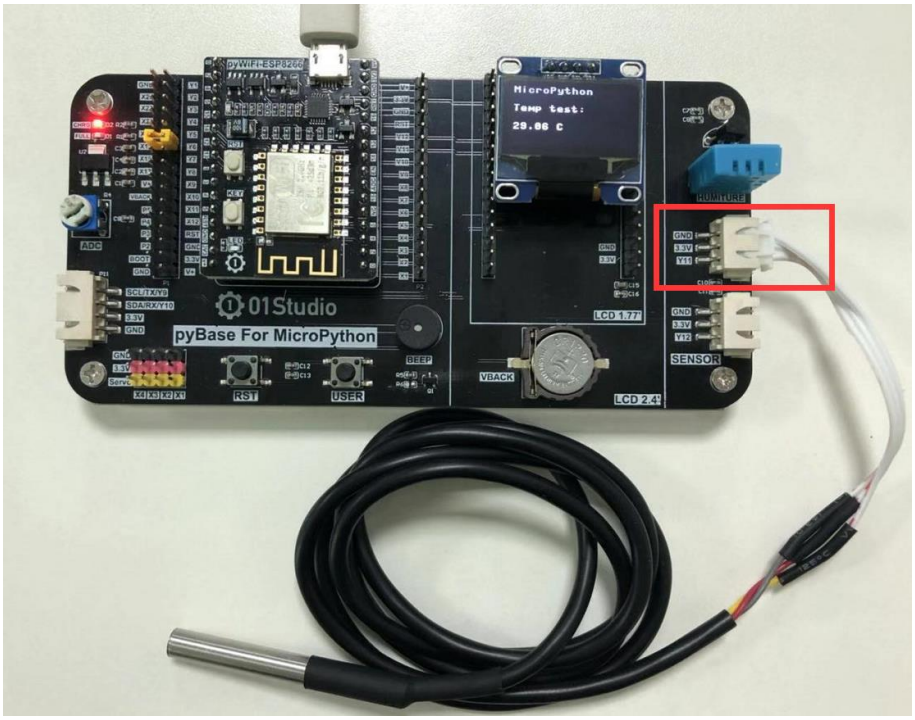


图 5-8 外接温度传感器

我们将金属探头放在水里面，可以见到测试到的温度。

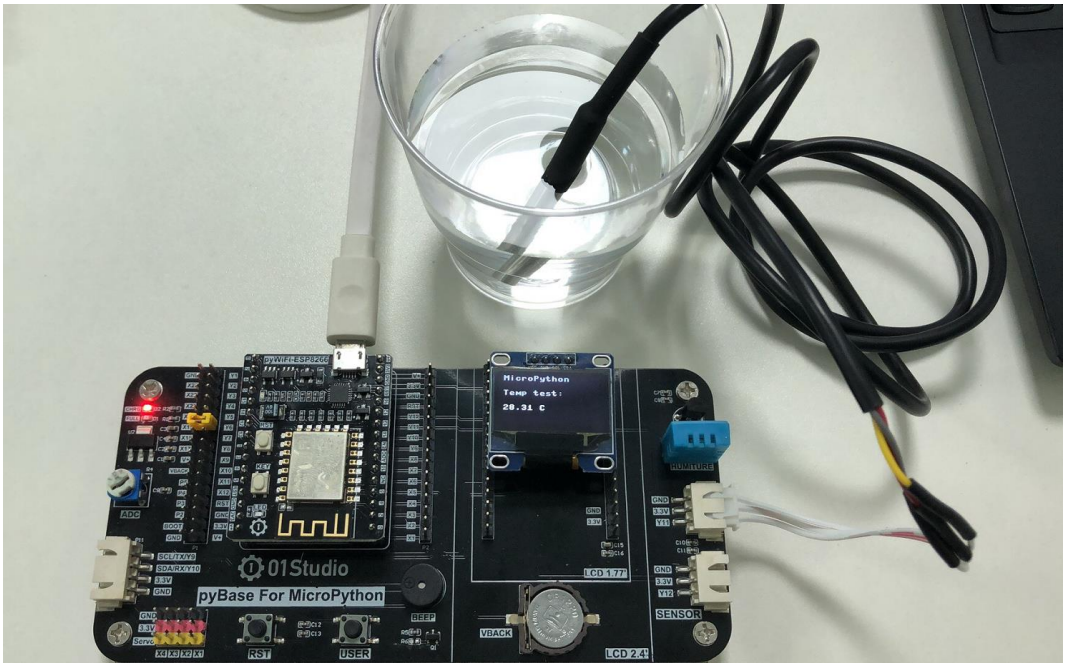


图 5-9 实现金属探头 DS18B20 测量

- **总结**

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

5.2 温湿度传感器 DHT11

- **前言：**

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

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

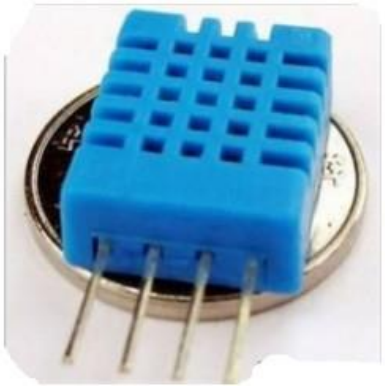


图 5-10 DHT11 温湿度传感器

- **实验平台：**

pyWiFi-ESP8266 和 pyBase 开发底板。DHT11 温湿度传感器位于右上方。

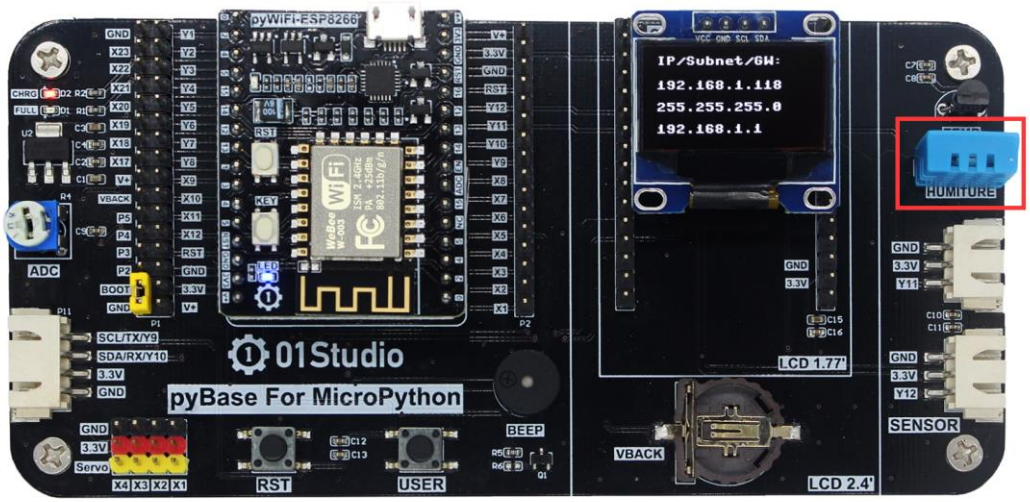


图 5-11 温湿度传感器 DHT11 位于右上方

- **实验目的：**

通过编程采集温湿度数据，并在 OLED 上显示。

- **实验讲解：**

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

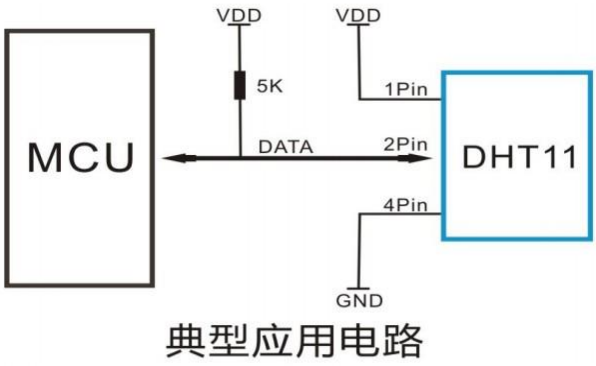


图 5-12　DHT11 应用电路

我们来看看 DHT11 在开发板上的接线图：

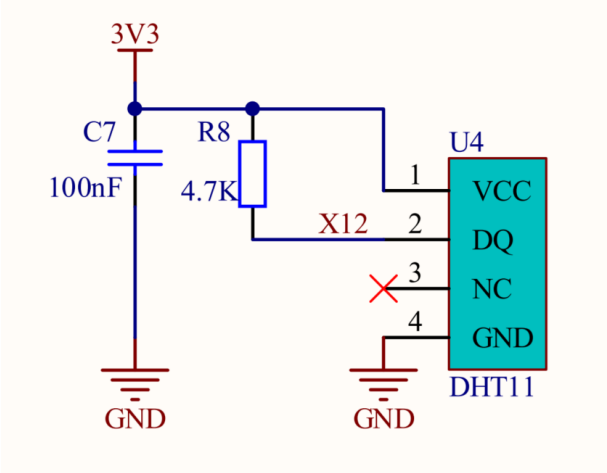


图 5-13　DHT11 接线图

可以看到 DHT11 连接到 pyBase 的‘X12’引脚，也就是连接到 pyWiFi-ESP8266 的引脚 5，如下图：

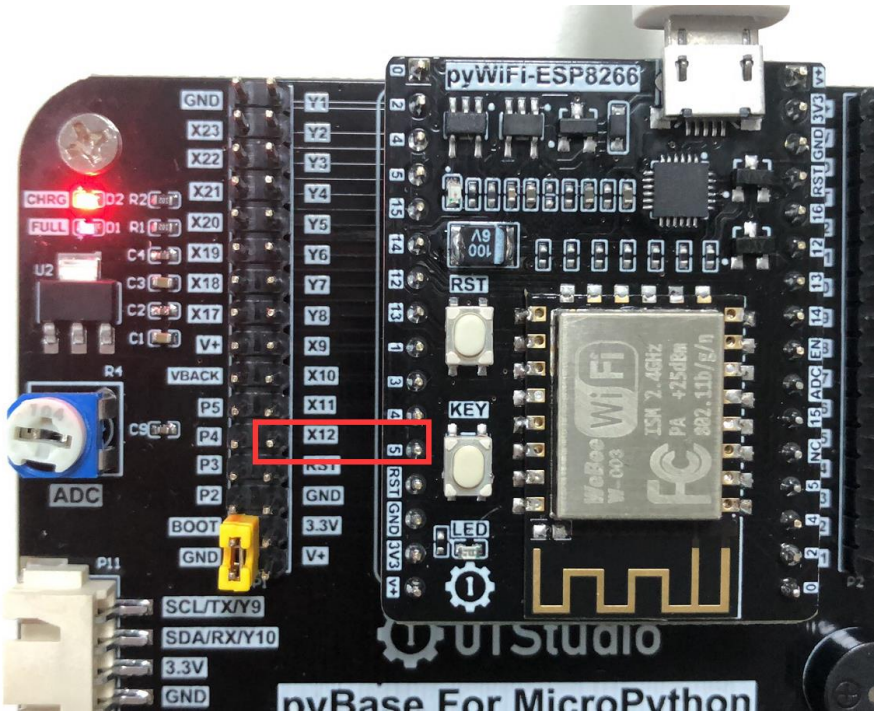


图 5-14 用跳线将 pyBase 的 X20、Y6 短接

也就是针对引脚 5 编程来驱动 DHT11 传感器，模块文件是 dht.py，如果你学习过前面基于 STM32 平台应该不陌生。而对于 ESP8266,这个模块已经集成到了初始化固件中，也就是说我们可以直接在 main.py 导入模块并调用即可。函数模块说明如下：

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

表 5-3 DHT11 对象

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

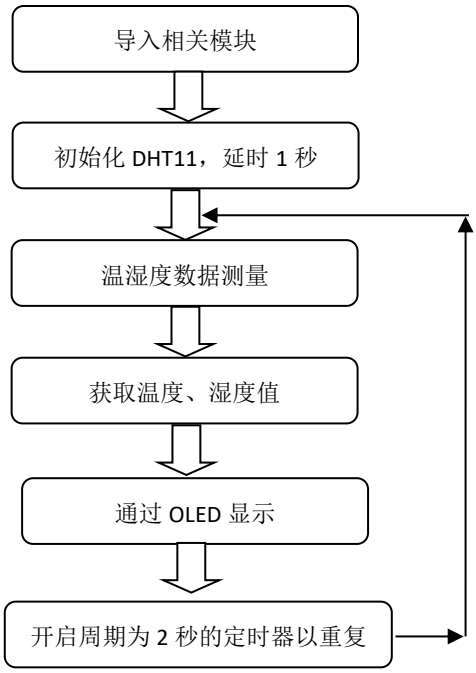


图 5-15 代码编写流程图

实验参考代码：

```
'''
实验名称：温湿度传感器 DHT11
版本：v1.0
日期：2019.7
作者：01Studio
说明：通过编程采集温湿度数据，并在 OLED 上显示。。
'''

#引入相关模块
from machine import Pin,I2C,Timer
from ssd1306 import SSD1306_I2C
import dht,time
```

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

#创建 DTH11 对象
d = dht.DHT11(Pin(5)) #传感器连接到引脚 14
time.sleep(1)   #首次启动停顿 1 秒让传感器稳定

def dht_get(tim):

    d.measure()         #温湿度采集

   #OLED 显示温湿度
    oled.fill(0) #清屏背景黑色
    oled.text('01Studio', 0, 0)
    oled.text('DHT11 test:',0,15)
    oled.text(str(d.temperature() )+' C',0,40)   #温度显示
    oled.text(str(d.humidity())+' %',48,40)  #湿度显示
    oled.show()

#开启 RTOS 定时器，编号为-1，周期 2 秒
tim = Timer(-1)
tim.init(period=2000, mode=Timer.PERIODIC,callback=dht_get)
```

- **实验结果：**

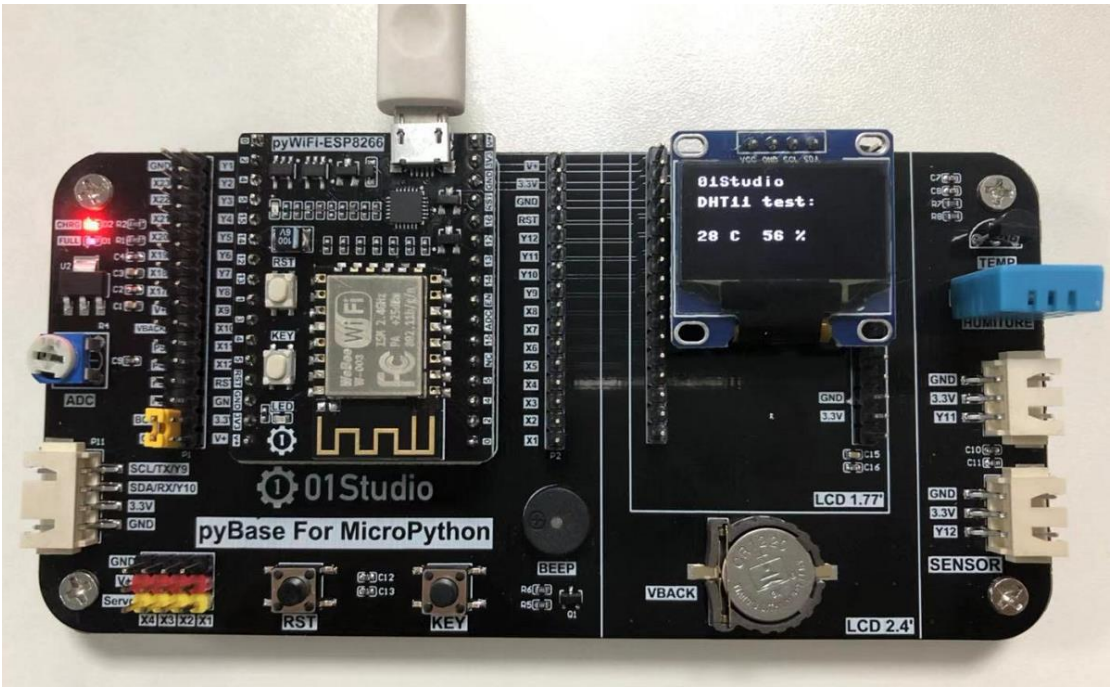


图 5-16　DHT11 实验结果

- **总结：**

通过本节学习我们学会了使用 MicroPython 来驱动 DTH11 温湿度传感器，DHT11 性价比比较高，是很适合学习使用的，但精度和响应速度有点低，需要更高要求应用的用户可以使用 DHT22 或者其他更高级的传感器。

第6章　WiFi 应用

通过前面的实验，我们已经对 ESP8266 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。ESP8266 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。

6.1 连接无线路由器

- **前言：**

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

- **实验平台：**

pyWiFi-ESP8266 和 pyBase 开发底板。

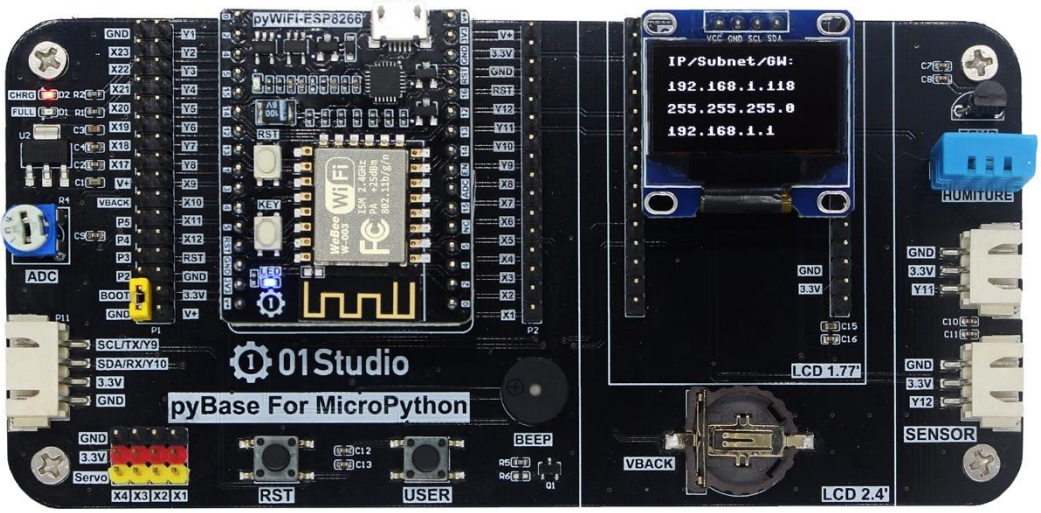


图 6-1 pyWiFi-ESP8266 开发套件

- **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。Ture：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 Ture: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.disconnected()
断开连接。

表 6-1　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 ESP8266 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：

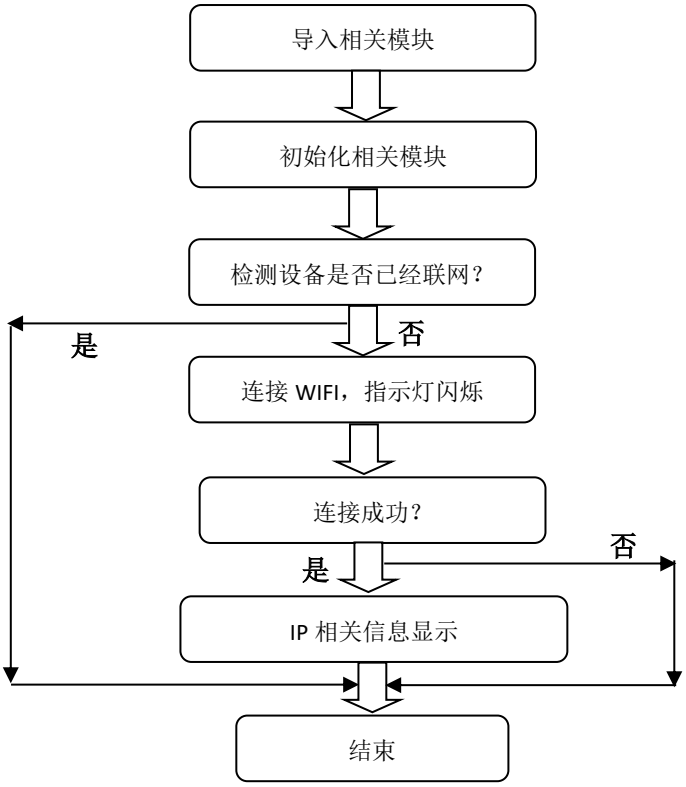


图 6-2 代码编写流程图

实验参考代码如下：

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

import network,time
from machine import I2C,Pin
from ssd1306 import SSD1306_I2C
```

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

#WIFI 连接函数
def WIFI_Connect():

    WIFI_LED=Pin(2, Pin.OUT, value=1) #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(0)
            time.sleep_ms(300)
            WIFI_LED.value(1)
            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(0)

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

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

#执行 WIFI 连接函数
WIFI_Connect()
```

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

- **实验结果：**

下载程序，可以观察到上电后 pyWiFi-ESP8266 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。

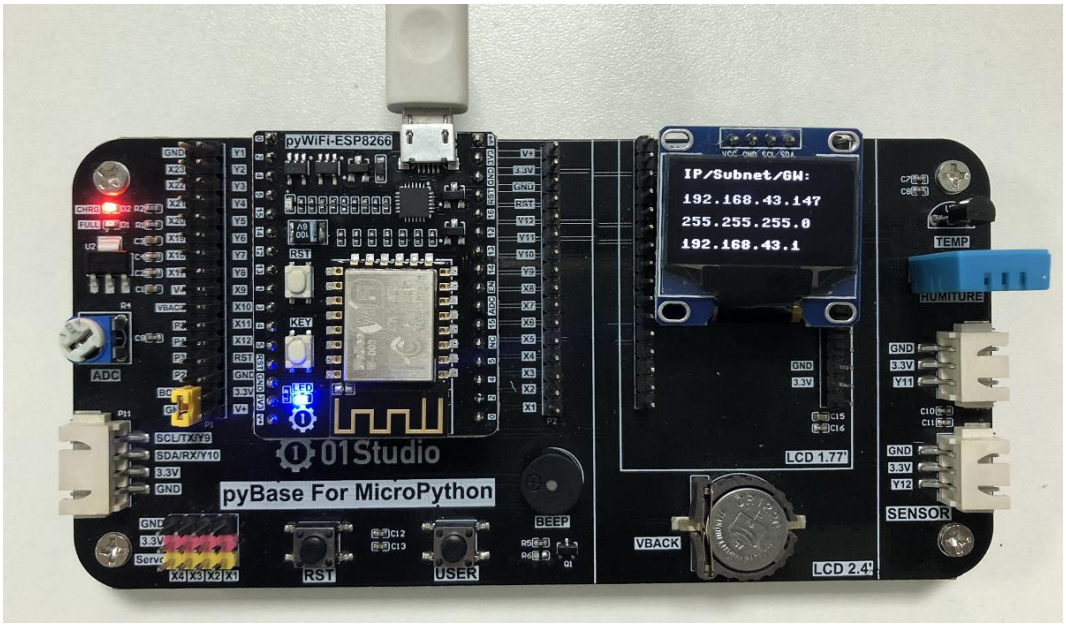


图 6-3 实验现象

- **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。

6.2 Socket 通信

- **前言：**

上一节我们学习了如何通过 MicroPython 编程实现 pyWiFi-ESP8266 模块连接到无线路由器。这一节我们则来学习一下 Socket 通信实验。Socket 几乎是整个互联网通信的基础。

- **实验平台：**

pyWiFi-ESP8266 和 pyBase 开发底板。

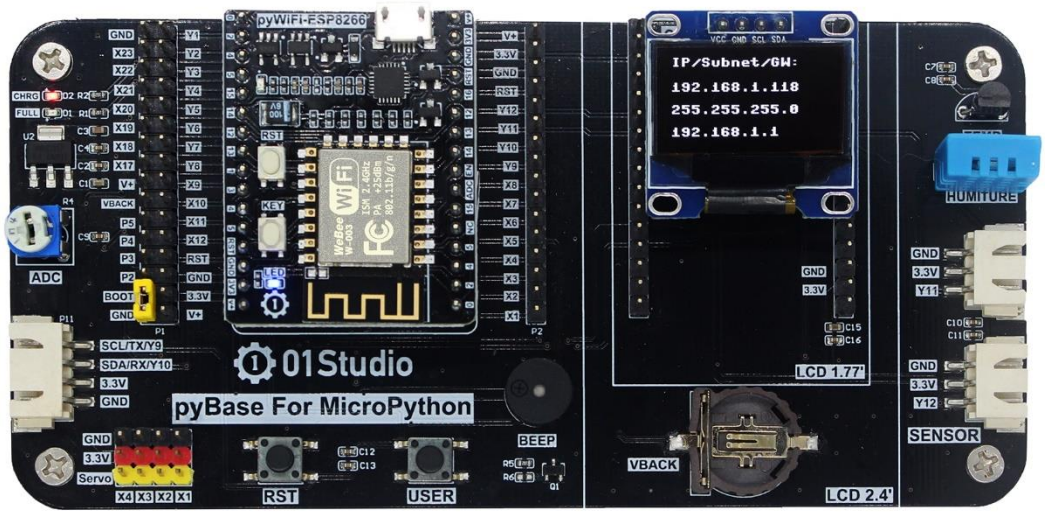


图 6-4 pyWiFi-ESP8266 开发套件

- **实验目的：**

通过 Socket 编程实现 pyWiFi-ESP8266 与电脑服务器助手建立连接，相互收发数据。

- **实验讲解：**

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

我们先来看看网络层级模型图，这是构成网络通信的基础：

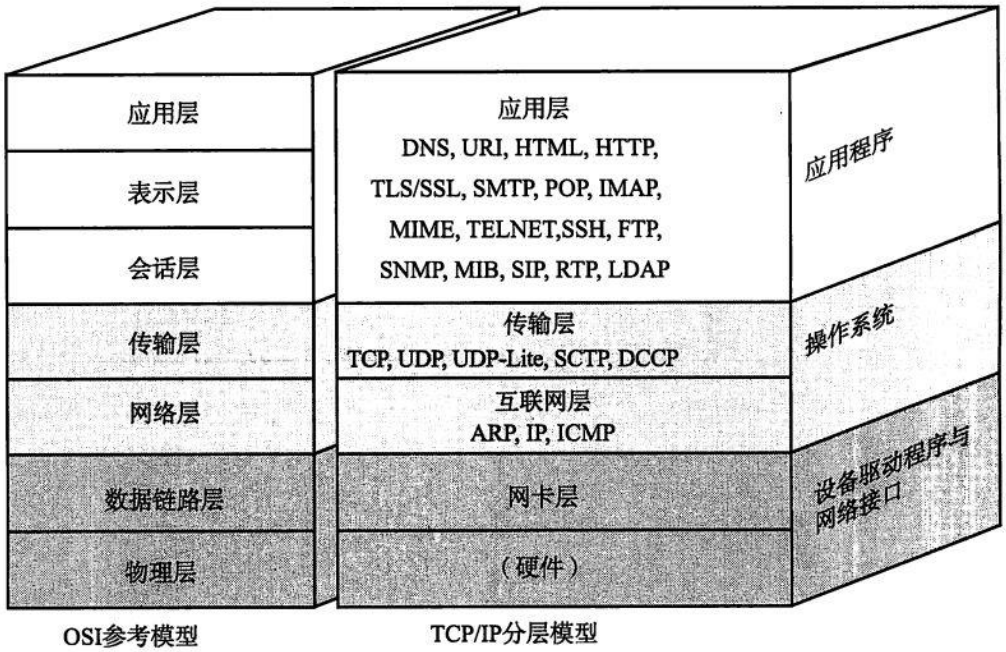


图 6-5 网络层级模型

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。

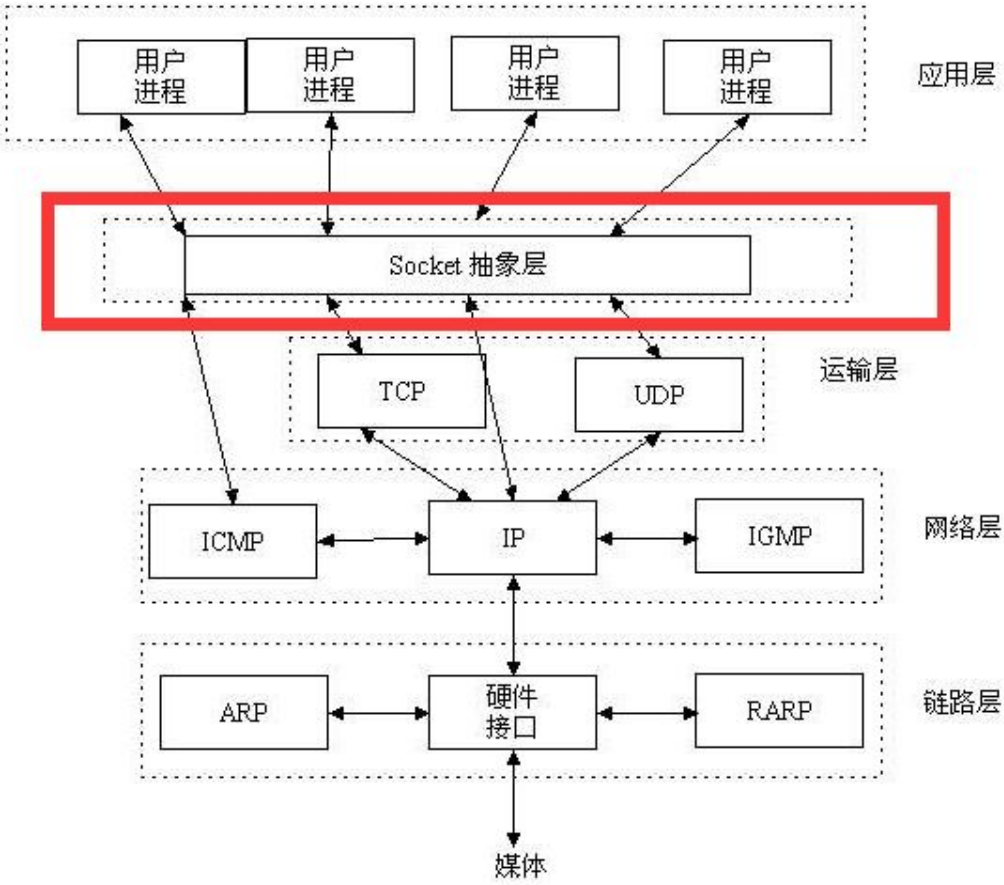


图 6-6　Socket 抽象层

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：

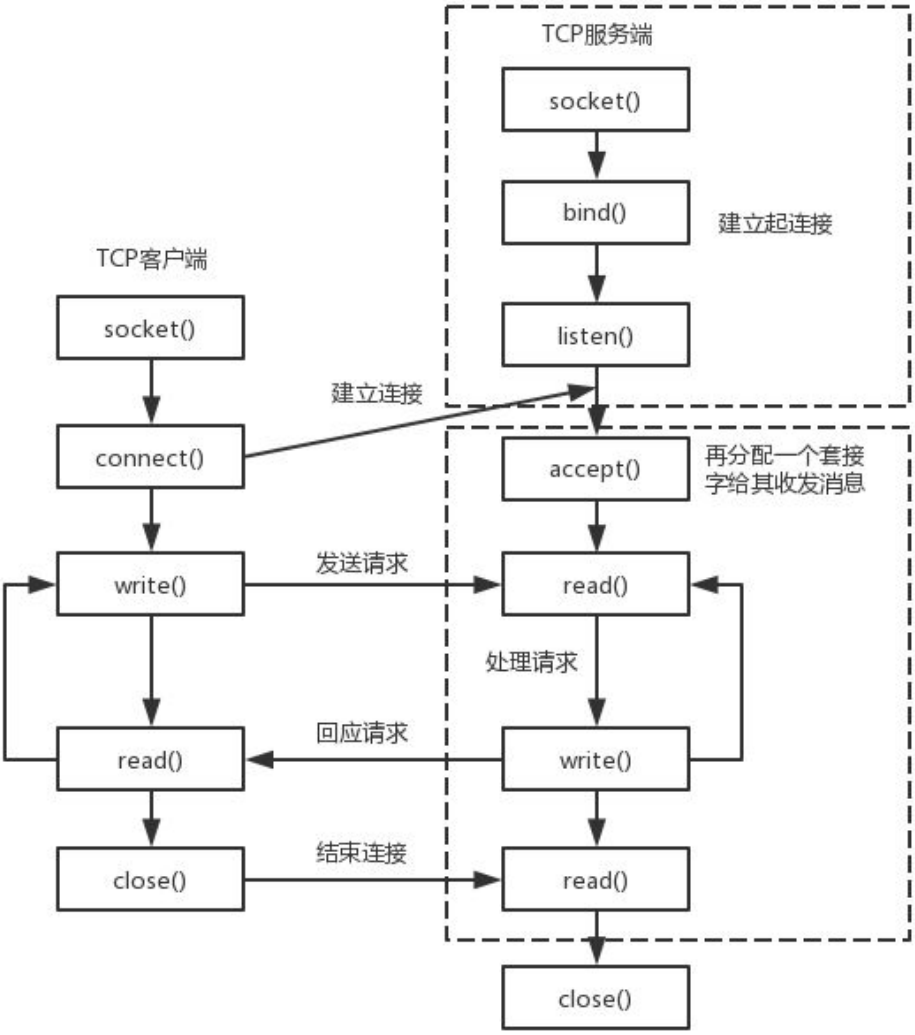


图 6-7　Socket 通信过程

从上图可以看到，建了 Socket 通信需要一个服务器端和一个客户端，以本实验为例，pyWiFi-ESP8266 作为客户端，电脑使用网络调试助手作为服务器端，双方使用 TCP 协议传输。对于客户端，则需要知道电脑端的 IP 和端口即可建立连接。（端口可以自定义，范围在 0~65535，注意不占用常用的 80 等端口即可。）

以上的内容，简单来说就是如果用户面向应用来说，那么 ESP8266 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
s=usocket.socekt(af=AF_INET, type=SOCK_STREAM,proto=IPPROTO_TCP)
构建 usocket 对象。 af: AF_INET→IPV4，AF_INET6 → IPV6； type: SCOK_STREAM→TCP，SOCK_DGRAM→UDP； proto: IPPROTO_TCP→TCP 协议，IPPROTO_UDP→UDP 协议。 （如果要构建 TCP 连接，可以使用默认参数配置，即不输入任何参数。）
使用方法
addr=usocket.getaddrinfo('www.01studio.org', 80)[0][-1]
获取 Socket 通信格式地址。返回：('47.91.208.161',80)
s.connect(address)
创建连接。address:地址格式为 IP+端口。例：('192.168.1.115',10000)
s.send(bytes)
发送。bytes：发送内容格式为字节
s.recv(bufsize)
接收数据。bufsize：单次最大接收字节个数。
s.bind(address)
绑定，用于服务器角色
s.listen([backlog])
监听，用于服务器角色。backlog:允许连接个数，必须大于 0。
s.accept()
接受连接，用于服务器角色。
*其它更多用法请阅读 MicroPython 文档：(搜索:usocket) 中文文档链接：http://docs.micropython.01studio.org/

表 6-2　Socket 对象

本实验中 pyWiFi-ESP8266 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：

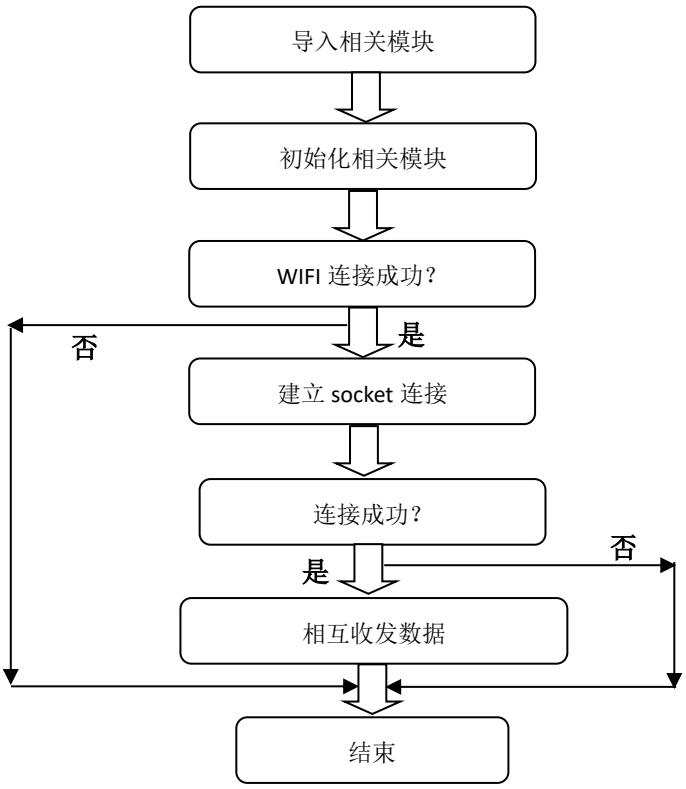


图 6-8 代码编写流程图

实验参考代码：

```
'''
实验名称：连接无线路由器
版本：v1.0
日期：2019.8
作者：01Studio
说明：通过 Socket 编程实现 pyWiFi-ESP8266 与电脑服务器助手建立 TCP 连接，相互收发数据。
'''
#导入相关模块
import network,usocket,time
from machine import I2C,Pin,Timer
from ssd1306 import SSD1306_I2C
```

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

# WIFI 连接函数
def WIFI_Connect():

    WIFI_LED=Pin(2, Pin.OUT, value=1) #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(0)
            time.sleep_ms(300)
            WIFI_LED.value(1)
            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(0)

        #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\网络调试助手目录下的 NetAssist.exe ，直接双击打开即可！

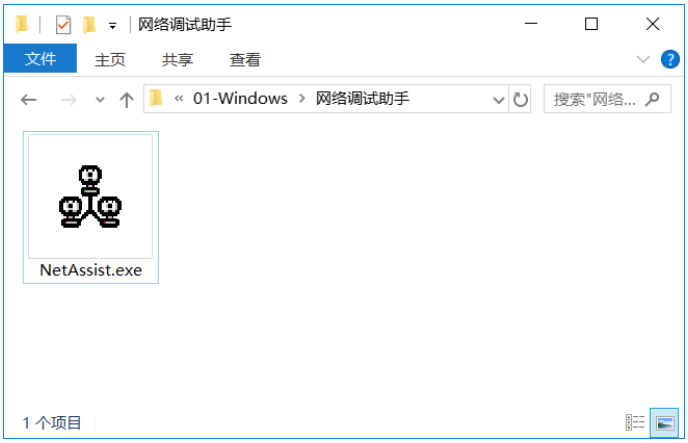


图 6-9 网络调试助手

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 6-10 TCP 服务器配置

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

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

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

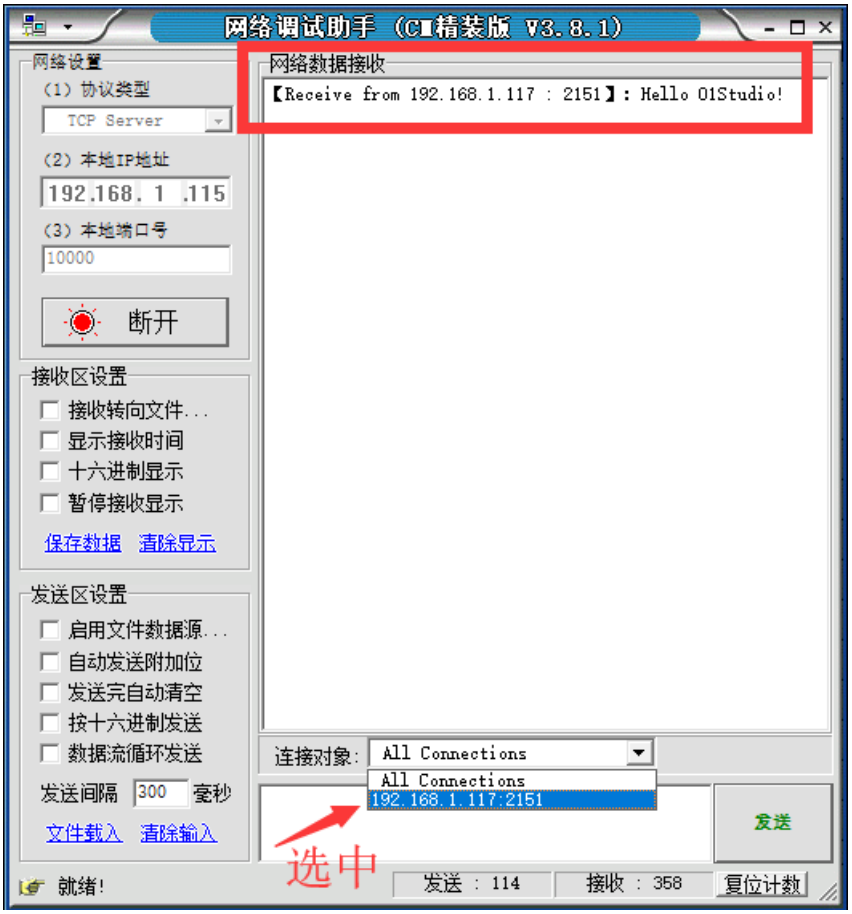


图 6-11 选中连接对象

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-g48dcbbe60 on 2019-05-29; ESP module with ESP8266
Type "help()" for more information.
>>> Warning: Comparison between bytes and str
b'Hi'
```

图 6-12

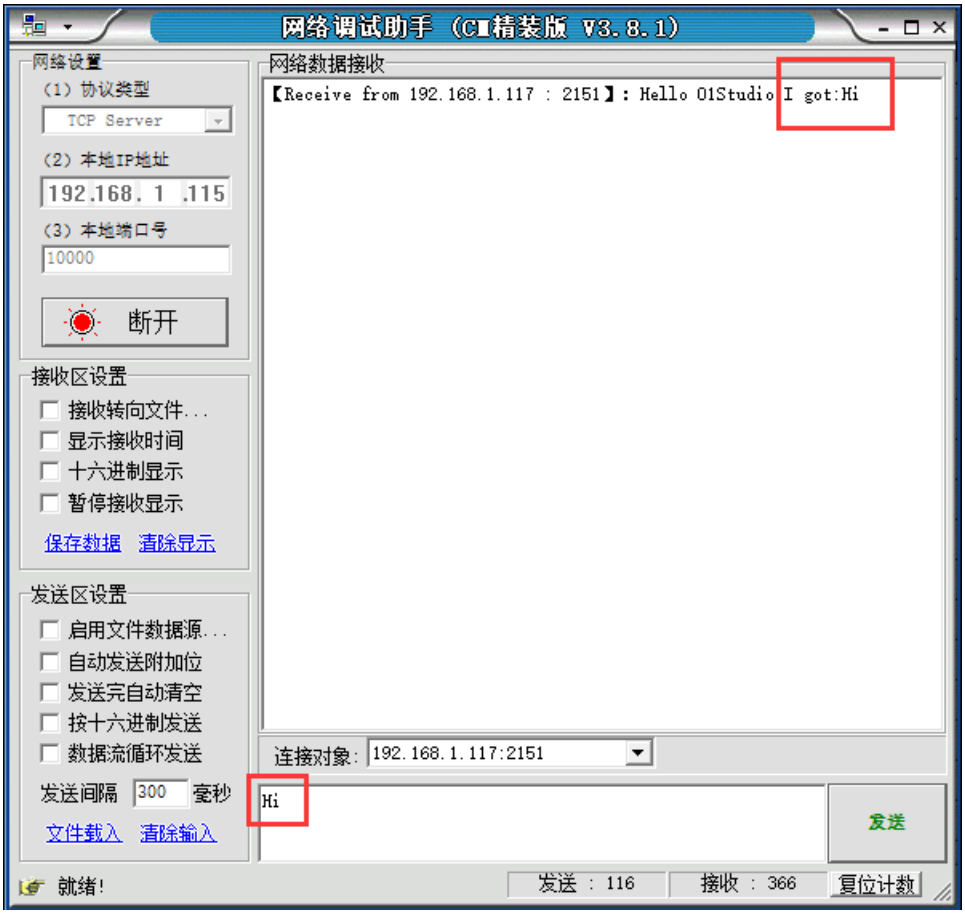


图 6-13 服务器发送数据

- **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。

6.3 MQTT 通信

- **前言：**

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

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

- **实验平台：**

pyWiFi-ESP8266 和 pyBase 开发底板。

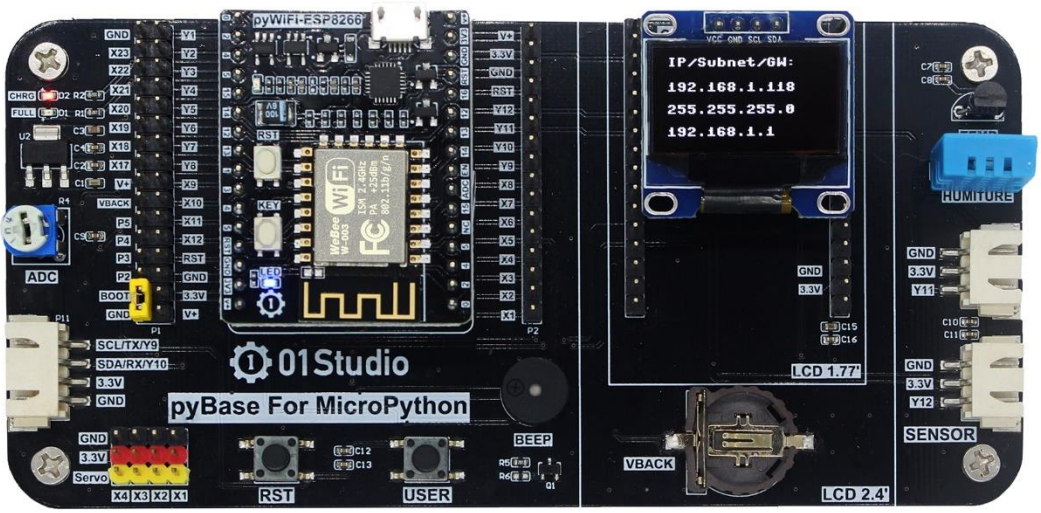


图 6-14 pyWiFi-ESP8266 开发套件

- **实验目的：**

通过编程实现 pyWiFi-ESP8266 实现 MQTT 协议信息的发布和订阅（接收）。

- **实验讲解：**

MQTT 是 IBM 于 1999 年提出的，和 HTTP 一样属于应用层，它工作在 TCP/IP 协议族上，通常还会调用 socket 接口。是一个基于客户端-服务器的消息发布/订

阅传输协议。其特点是协议是轻量、简单、开放和易于实现的，这些特点使它适用范围非常广泛。在很多情况下，包括受限的环境中，如：机器与机器（M2M）通信和物联网（IoT）。其在，通过卫星链路通信传感器、偶尔拨号的医疗设备、智能家居、及一些小型化设备中已广泛使用。

总结下来 MQTT 有如下特性/优势：

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

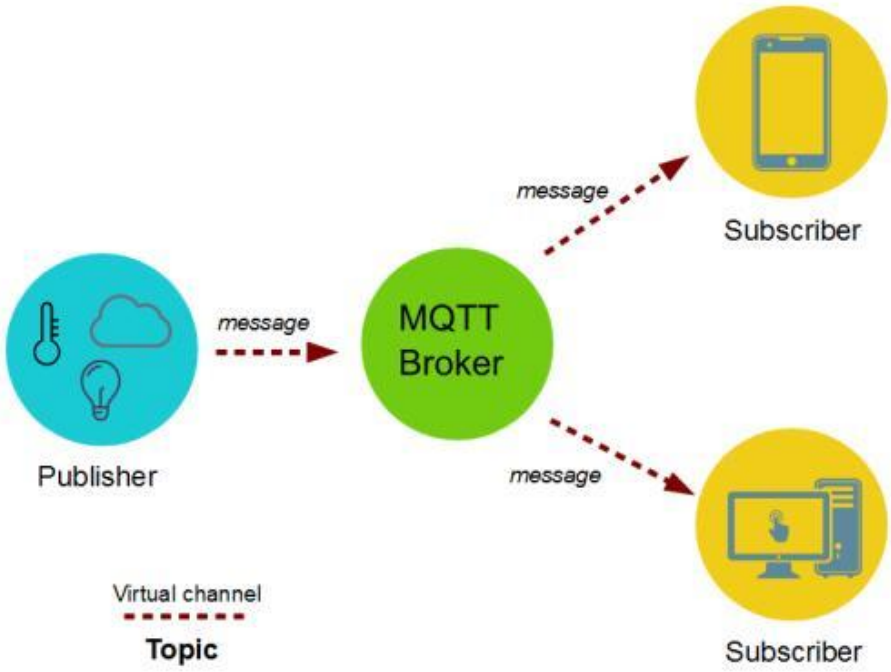


图 6-15 MQTT 通信流程

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：

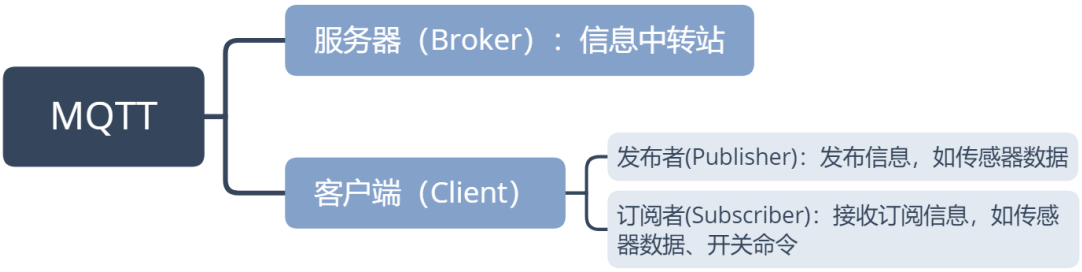


图 6-16　MQTT 角色说明

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

图 6-17 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
client=simple. MQTTClient (client_id, server, port)
构建 MQTT 客户端对象。 client_id: 客户端 ID，具有唯一性； server: 服务器地址，可以是 IP 或者网址； port:服务器端口。（默认是 1883，服务器通常采用的端口，也可以自定义。）
使用方法

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

表 6-3　MQTT 客户端对象

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

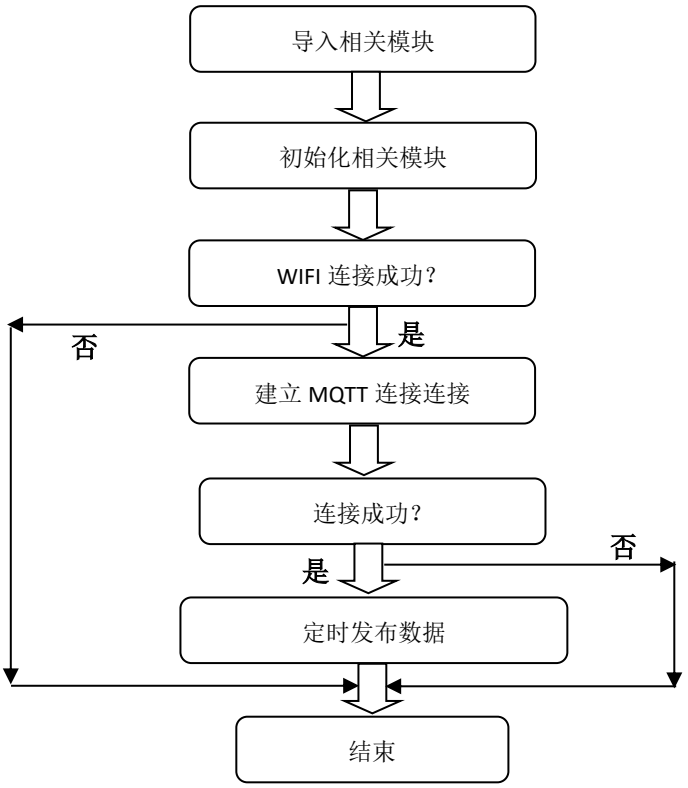


图 6-18 发布者代码编写流程

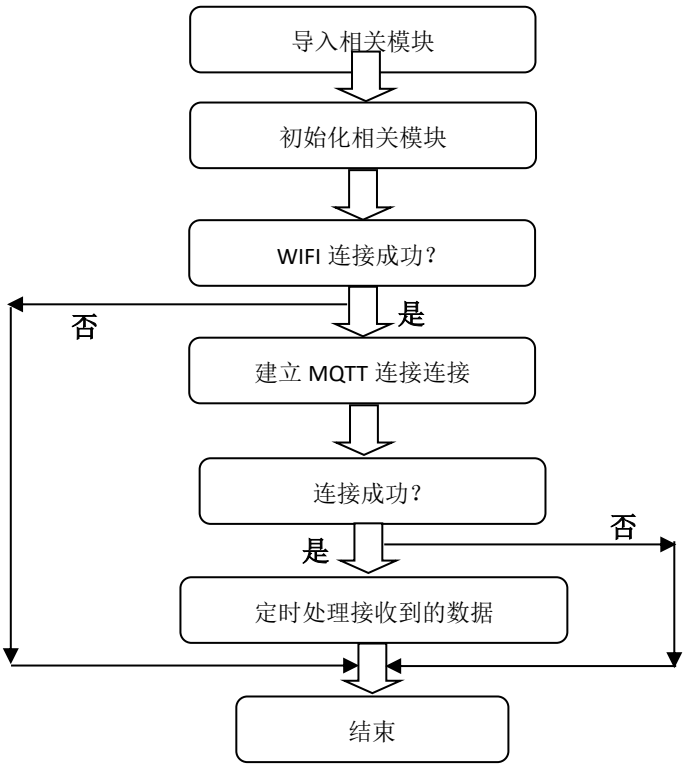


图 6-19 订阅者代码编写流程

发布者（publish）参考代码：

```
'''
实验名称：MQTT 通信
版本：v1.0
日期：2019.8
作者：01Studio
说明：编程实现 MQTT 通信，实现发布数据。
'''
import network,time
from simple import MQTTClient #导入 MQTT 板块
from machine import I2C,Pin,Timer
from ssd1306 import SSD1306_I2C


#初始化相关模块
```

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

#WIFI 连接函数
def WIFI_Connect():

    WIFI_LED=Pin(2, Pin.OUT, value=1) #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(0)
            time.sleep_ms(300)
            WIFI_LED.value(1)
            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(0)

        #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

#发布数据任务
def MQTT_Send(tim):
    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 = 'mqtt.p2hp.com'
    PORT = 1883
    CLIENT_ID = '01Studio-ESP8266' # 客户端 ID
    TOPIC = '/public/01Studio/1' # TOPIC 名称
    client = MQTTClient(CLIENT_ID, SERVER, PORT)
```

```
    client.connect()


    #开启 RTOS 定时器，编号为-1,周期 1000ms，执行 socket 通信接收任务
    tim = Timer(-1)
    tim.init(period=1000, mode=Timer.PERIODIC,callback=MQTT_Send)
```

订阅者（subscribe）参考代码：

```
'''
实验名称：MQTT 通信
版本：v1.0
日期：2019.8
作者：01Studio
说明：编程实现 MQTT 通信，实现订阅（接收）数据。
'''
import network,time
from simple import MQTTClient #导入 MQTT 板块
from machine import I2C,Pin,Timer
from ssd1306 import SSD1306_I2C


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


#WIFI 连接函数
def WIFI_Connect():


    WIFI_LED=Pin(2, Pin.OUT, value=1) #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(0)
        time.sleep_ms(300)
        WIFI_LED.value(1)
        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(0)

    #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 = 'mqtt.p2hp.com'
    PORT = 1883
    CLIENT_ID = '01Studio-ESP8266' # 客户端 ID
    TOPIC = '/public/01Studio/1' # TOPIC 名称

    client = MQTTClient(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 网络助手同一个服务器和端口。

```
SERVER = 'mqtt.p2hp.com'
PORT = 1883
```

- **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://mqtt.p2hp.com/websocket/

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，这里默认即可，点击连接。

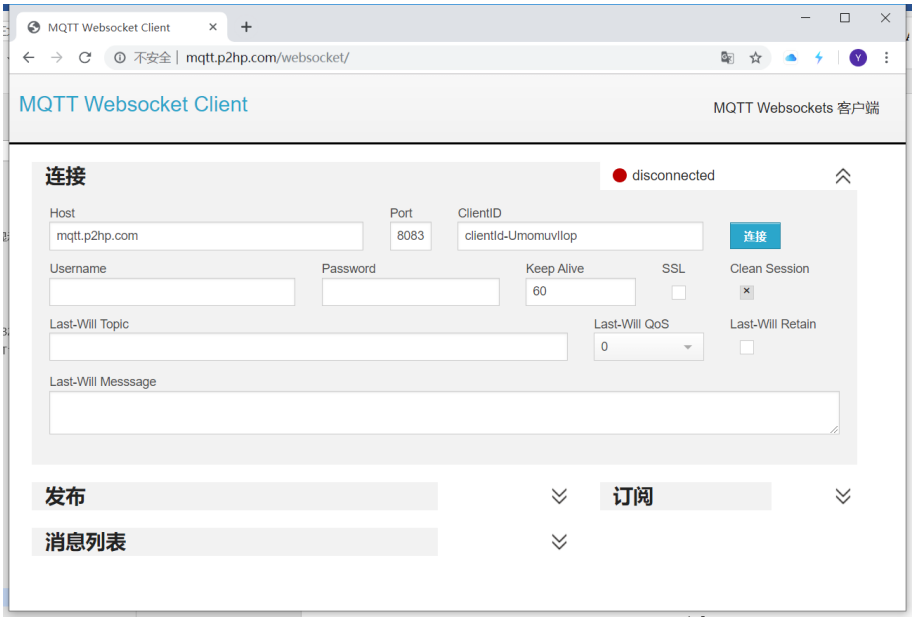


图 6-20 MQTT 助手

连接成功可以看到显示 Connected。

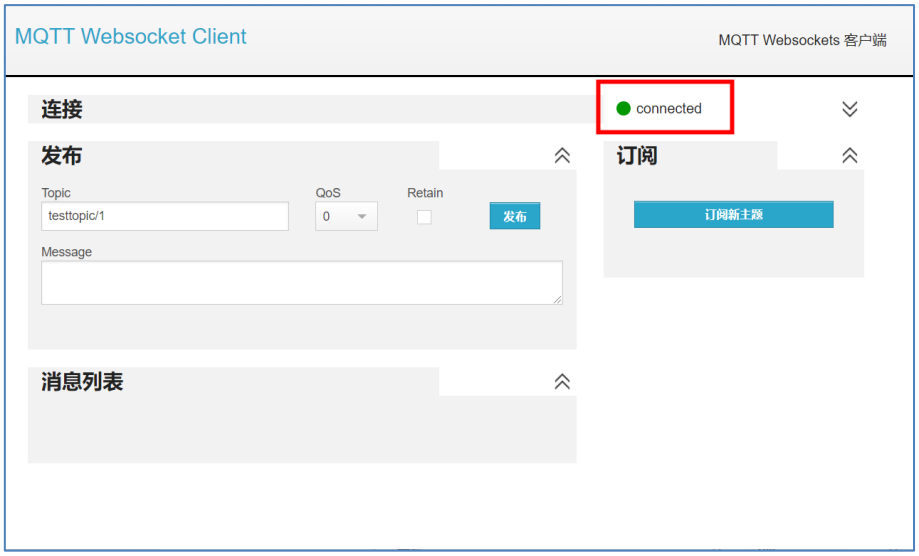


图 6-21 MQTT 助手连接成功

测试“发布者”代码：

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

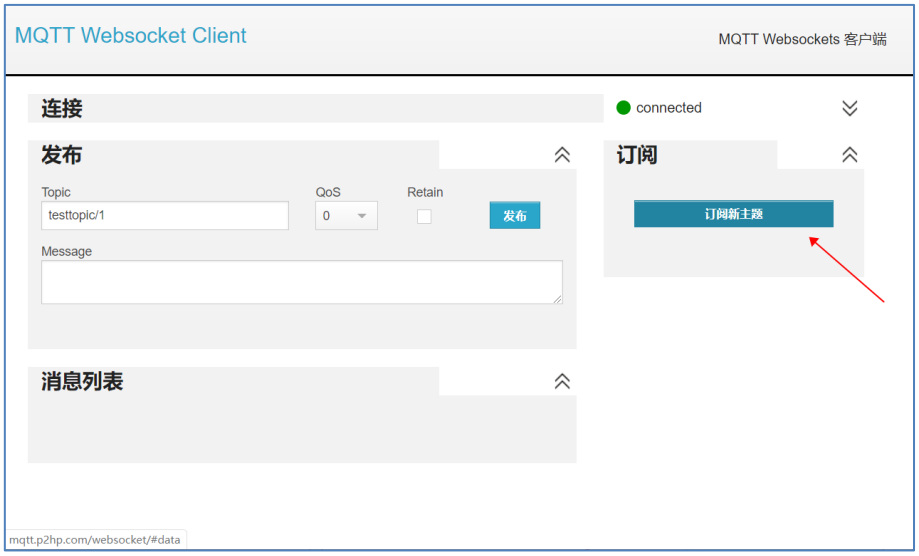


图 6-22 订阅主题

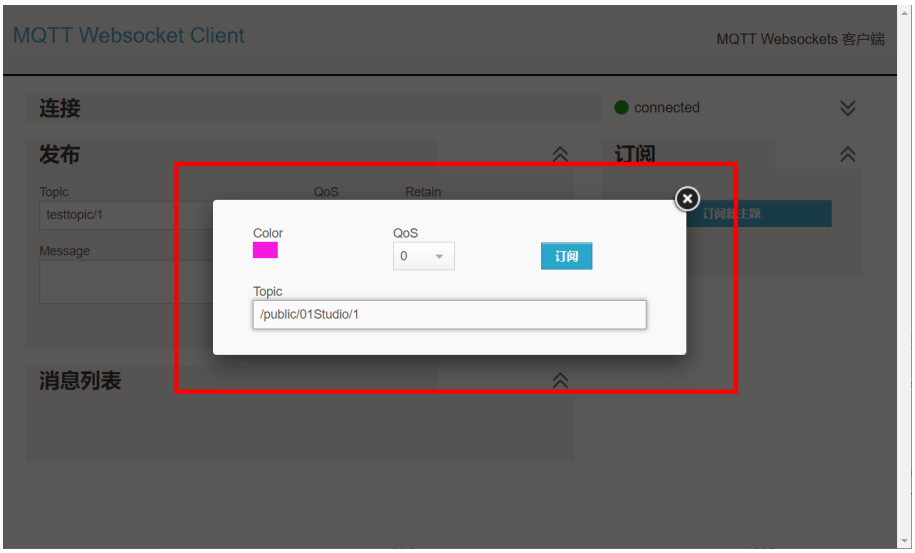


图 6-23

订阅后可以看到接收框收到来自开发板发布的信息。

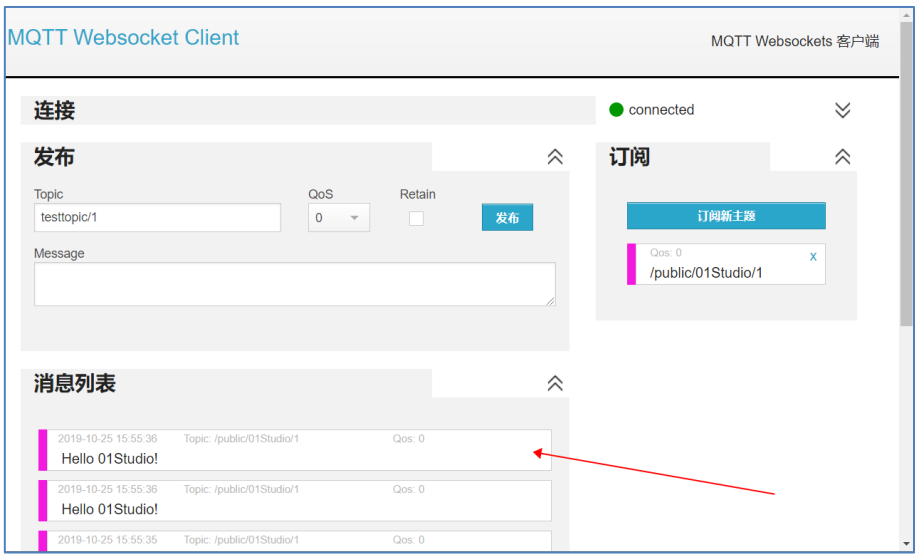


图 6-24 MQTT 助手收到开发板发布的信息

测试“订阅者”代码：

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

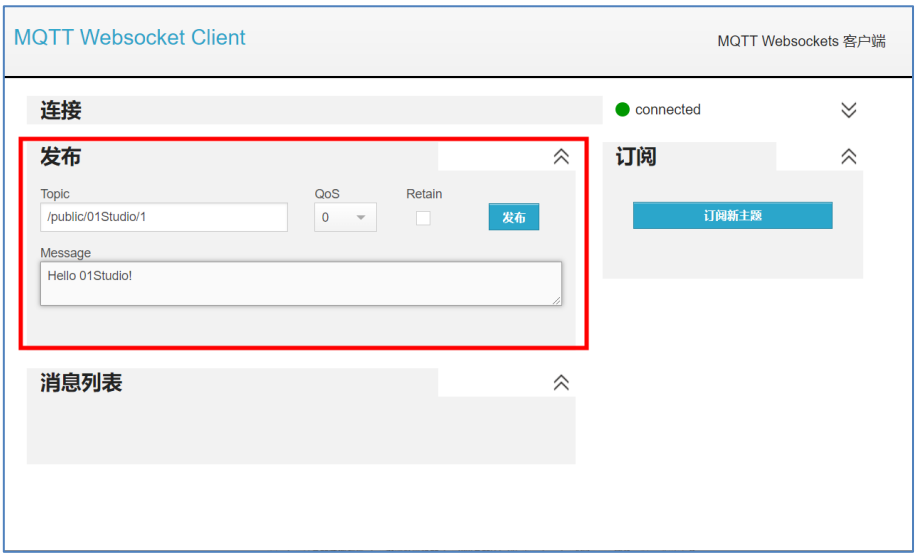


图 6-25 输入即将发布的主题和信息

将“订阅者”代码下载到开发板，成功连接后回到 MQTT 助手点击【发布】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）。

```
ESP MicroPython REPL
network information: ('192.168.1.117', '255.255.255.0', '192.168.1.1', '192.168.1.1')
◆◆>
MicroPython v1.11-8-g48dcbbe60 on 2019-05-29; ESP module with ESP8266
Type "help()" for more information.
>>> topic: b'/public/01Studio/1
msg: b'Hello 01Studio!'
```

图 6-26 开发板接收到相关信息并打印

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

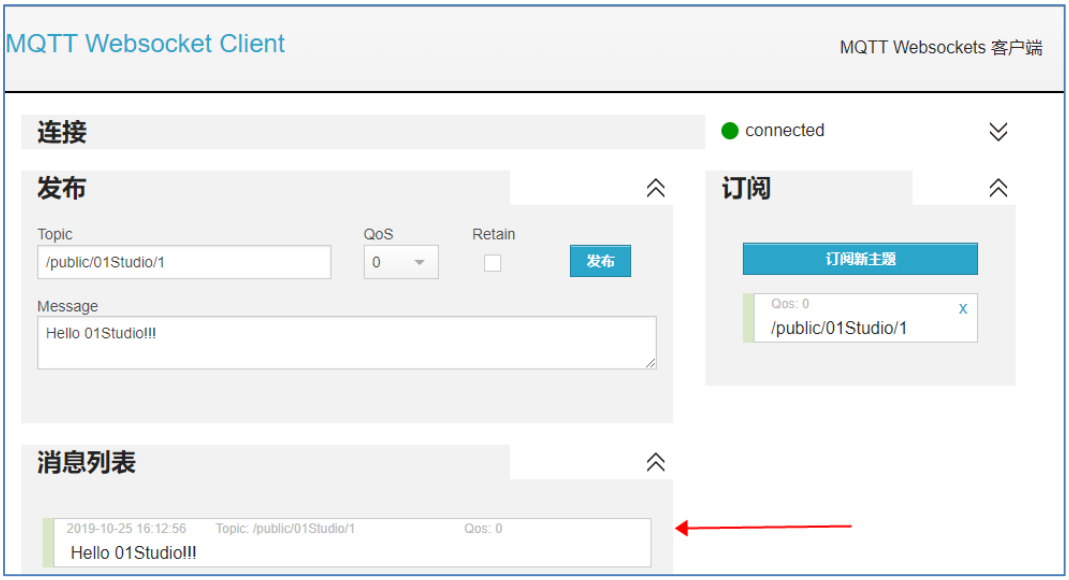


图 6-27 客户端同时发布和订阅信息

- **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。

6.4 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；

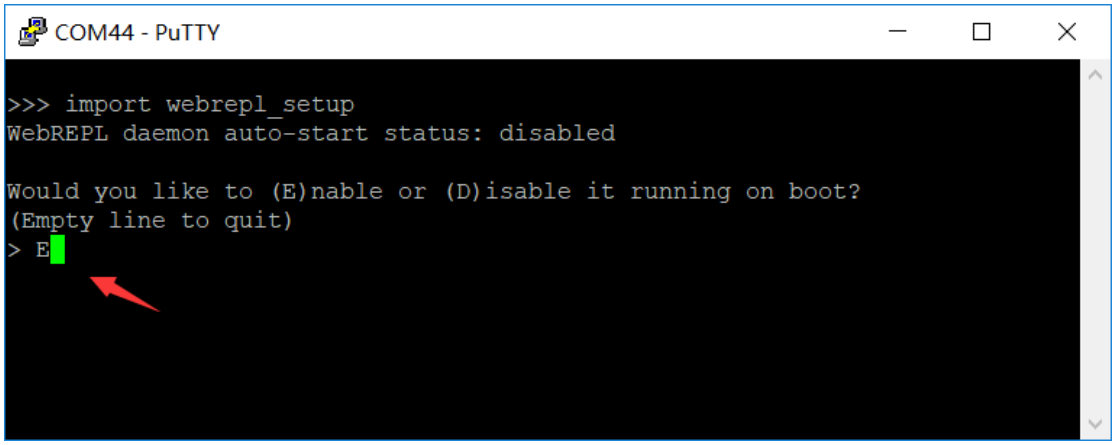


图 6-28

接下来设置密码，设置完成后输入‘y’,选择重启。

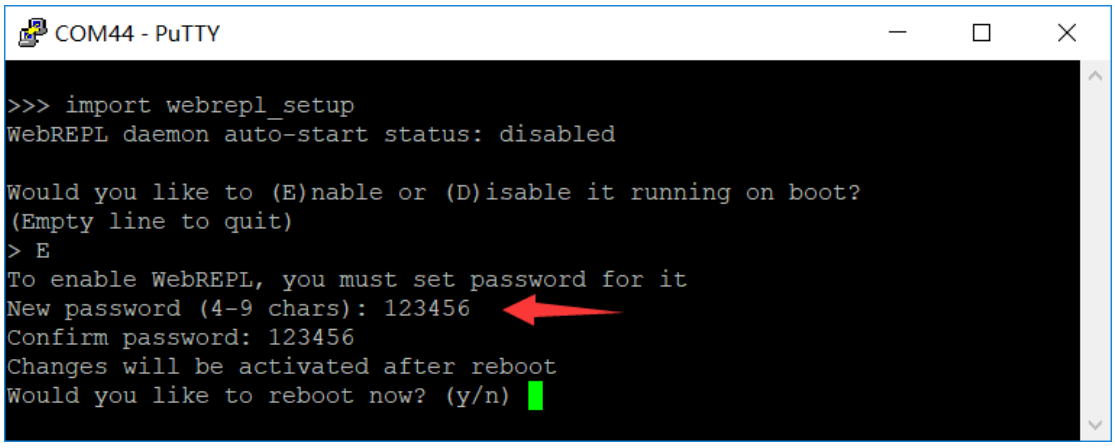


图 6-29

第3步:

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

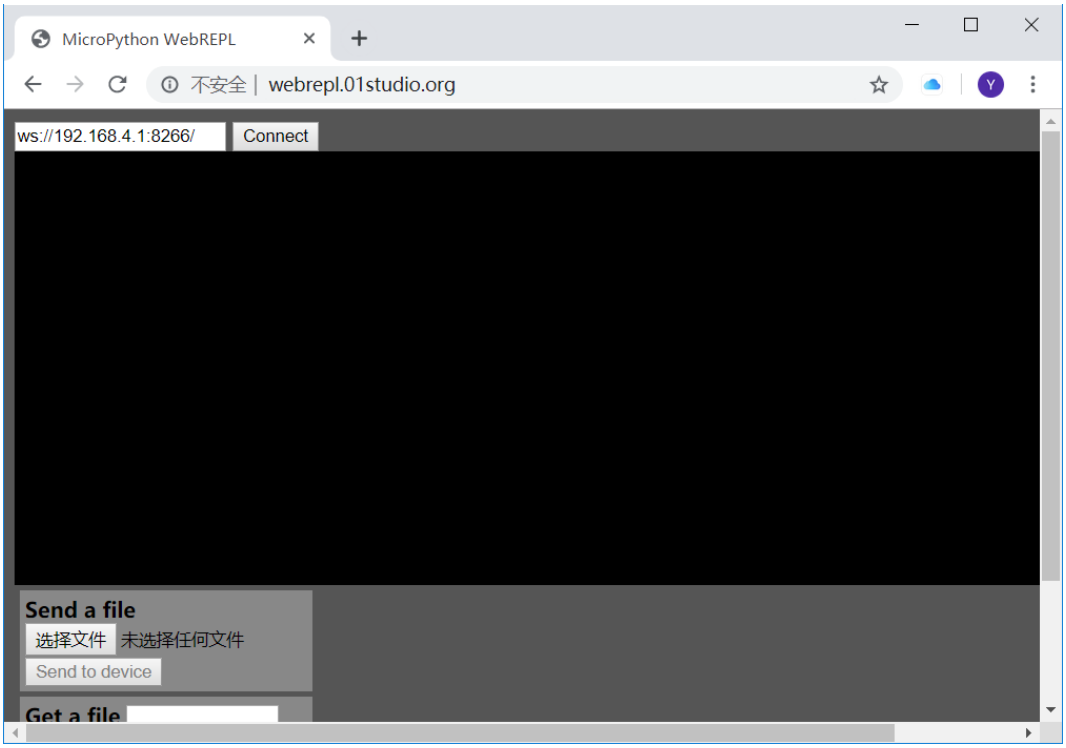


图 6-30 WebREPL

第4步:

在左上角输入 pyWiFi-ESP8266 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：

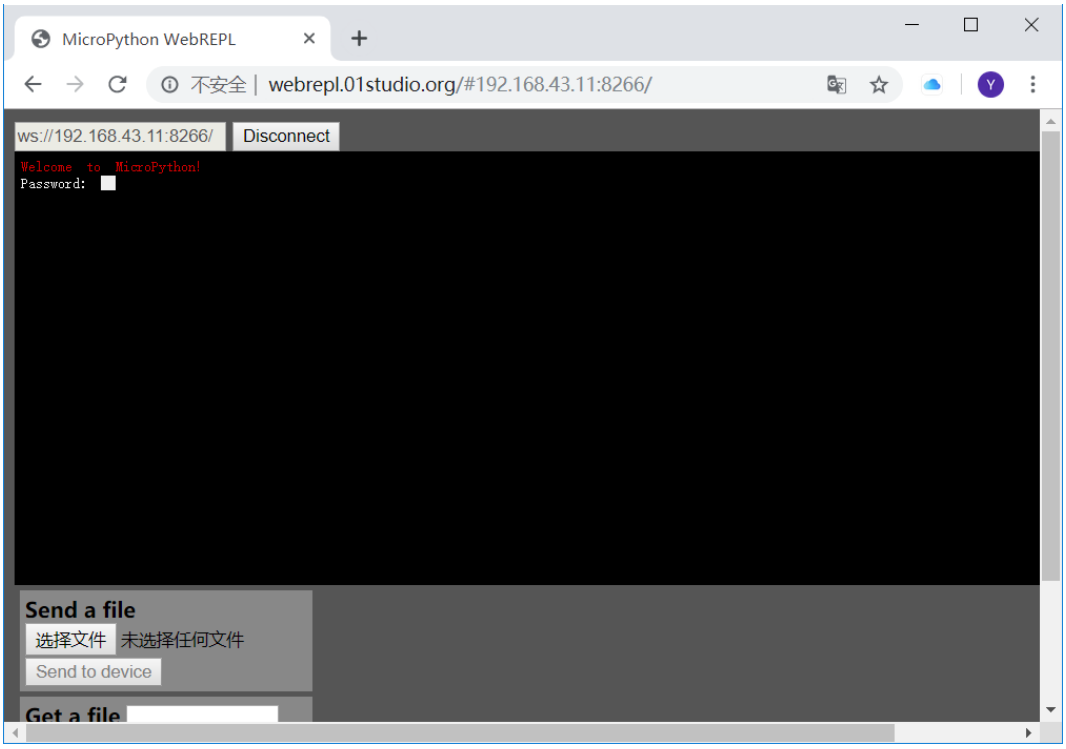


图 6-31

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。

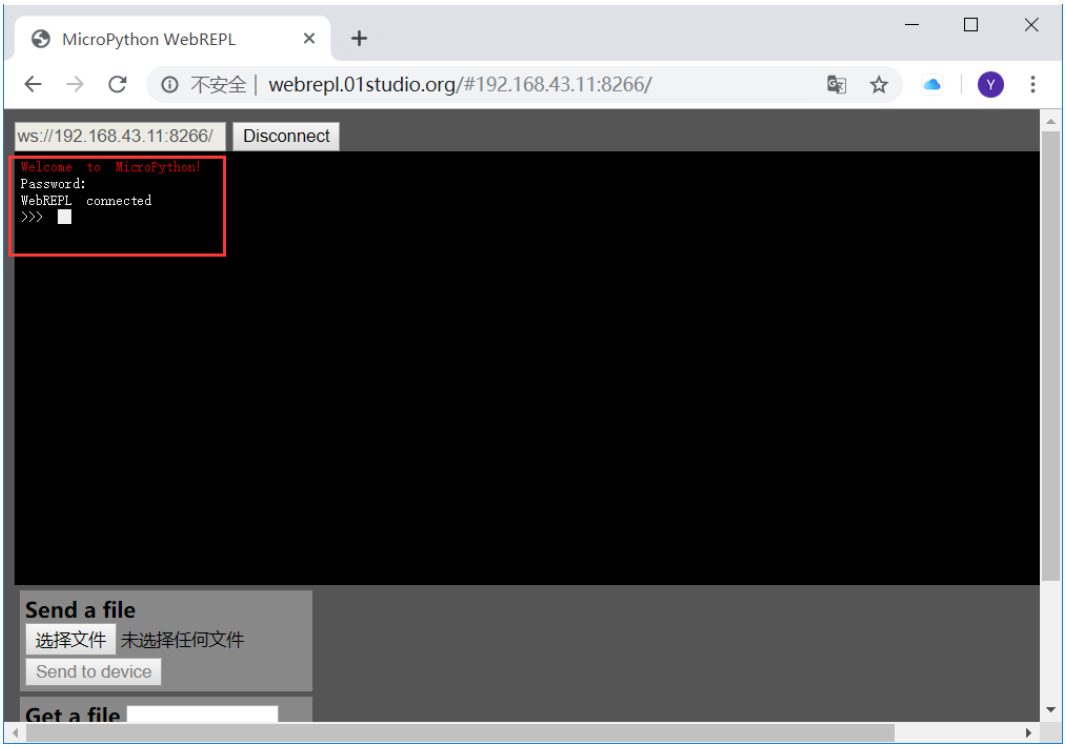


图 6-32　WebREPL 连接成功

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”。

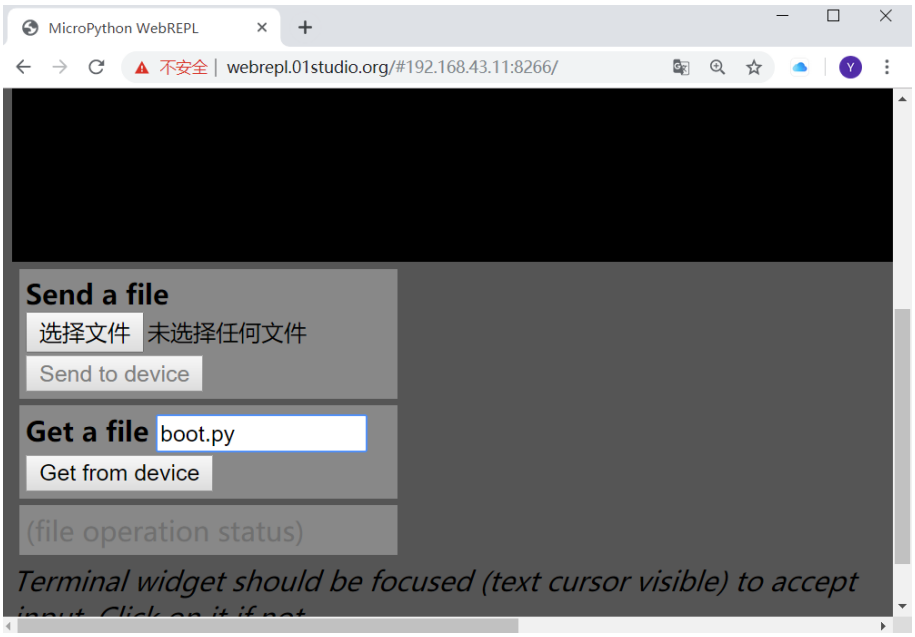


图 6-33 文件收发


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-


基于哥伦布 STM2F407平台
基于K210平台
基于OpenMV4平台
基于NRF62840平台
基于ESP32平台
基于ESP8266平台
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)

关注公众号，免费下载