# Python4IoTs2

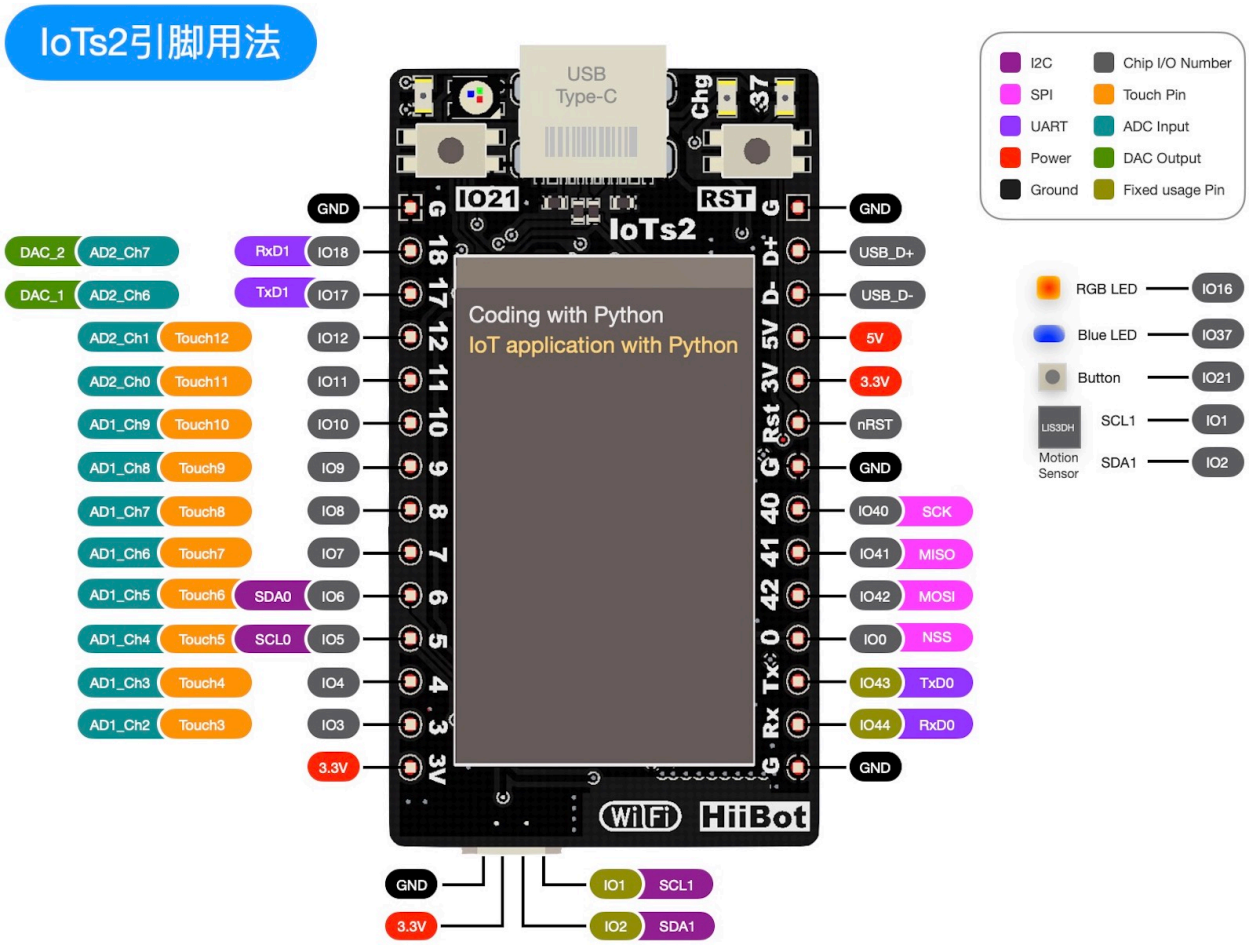
***Release 1.0.0***

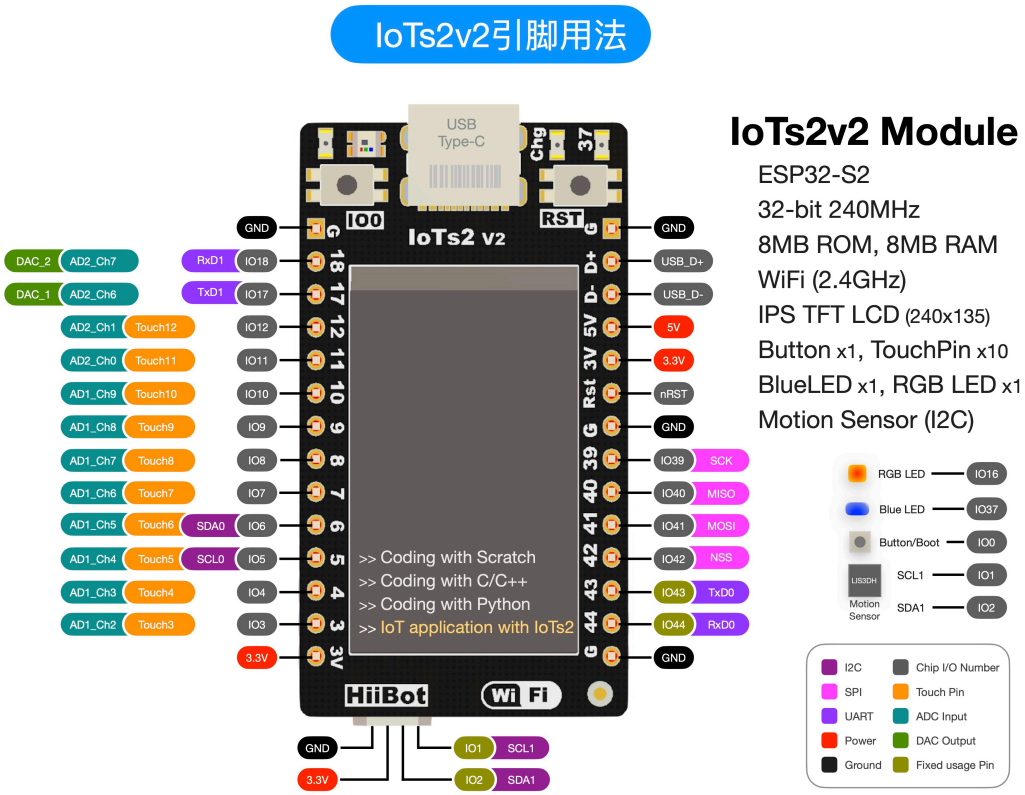
**LeBan**

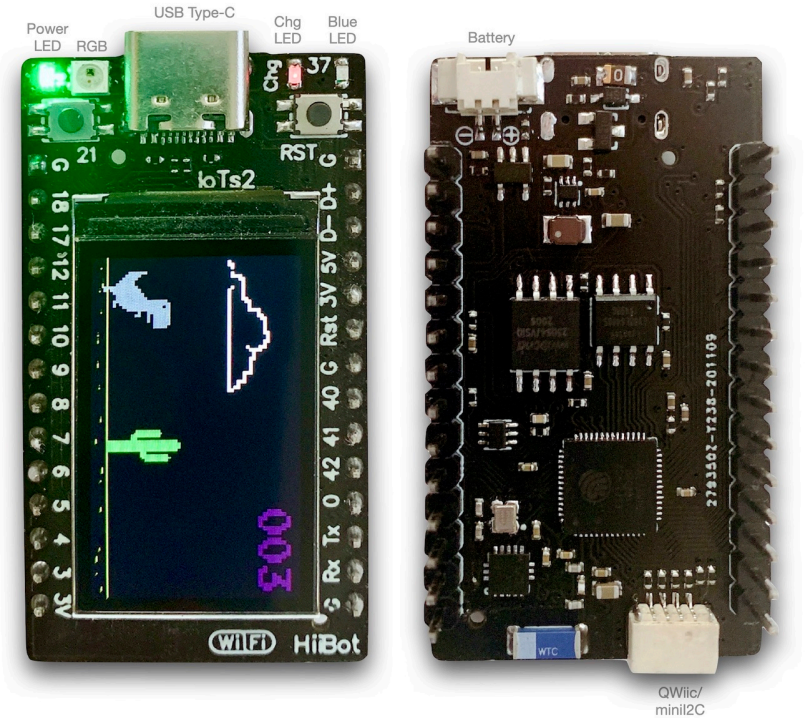
**Oct 22, 2021**

# CONTENTS

IoTs2 采用 ESP32-S2(Xtensa 32-bit LX7 单核 CPU，主频达 240MHz) WiFi SoC，板上带有 8MB FlashROM 和 8MB RAM、1.13 寸彩色 LCD(135x240 点阵)、3 轴加速度传感器 (LIS3DH)，以及 1 颗可编程 RGB 和 1 颗可编程蓝色 LED、1 个可编程按钮等资源。板载大功率开关型 DC-DC(输出 3.3V@1.2A) 可为功能拓展板供电，支持单节锂电池供电输入。IoTs2 采用双列 (2x14) 拓展接口和 4P 的 mini-I2C(QWiic/STEMMA QT) 拓展接口，即可将 IoTs2 嵌入系统拓展板内使用，也可以使用 I2C 总线拓展系统功能。IoTs2 使用 USB Type-C 接口与程序开发主机 (电脑) 连接，IoTs2 支持 Scratch 图形化编程语言和 Python 脚本编程语言，采用“拖放文件即下载程序”的操作模式，不依赖任何特定的软件环境。




IoTs2v2引脚用法
IoTs2v2 Module
ESP32-S2
32-bit 240MHz
8MB ROM, 8MB RAM
WiFi (2.4GHz)
IPS TFT LCD (240x135)
Button x1, TouchPin x10
BlueLED x1, RGB LED x1
Motion Sensor (I2C)
USB Type-C
IoTs2 v2
>> Coding with Scratch
>> Coding with C/C++
>> Coding with Python
>> IoT application with IoTs2
HiiBot
Wi Fi
RGB LED IO16
Blue LED IO37
Button/Boot IO0
SCL1 IO1
SDA1 IO2
Motion Sensor
I2C
SPI
UART
Power
Ground
Chip I/O Number
Touch Pin
ADC Input
DAC Output
Fixed usage Pin




合作咨询请联系：400-666-8152

CHAPTER

ONE

# IOTS2 简介

## 1.1 IoTs2 简介

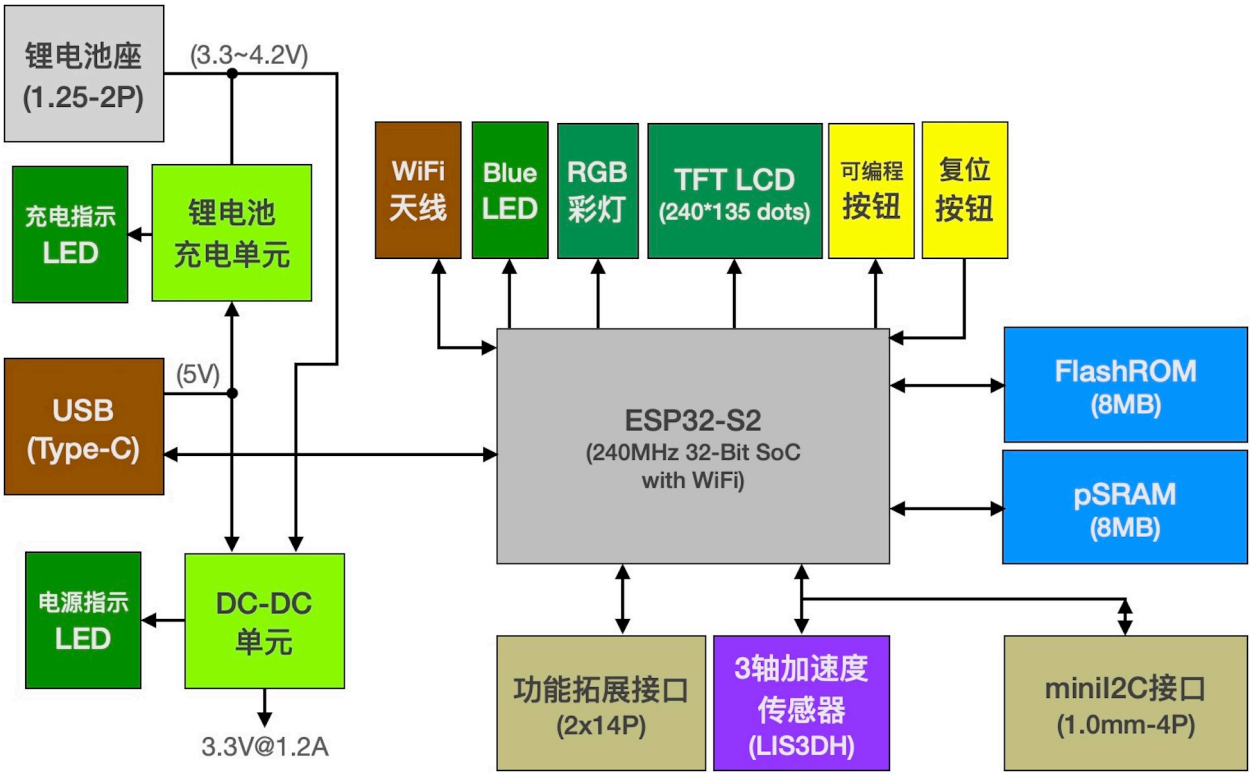
从这里开始认识并学习和使用 IoTs2。

### 1.1.1 IoTs2 是什么？

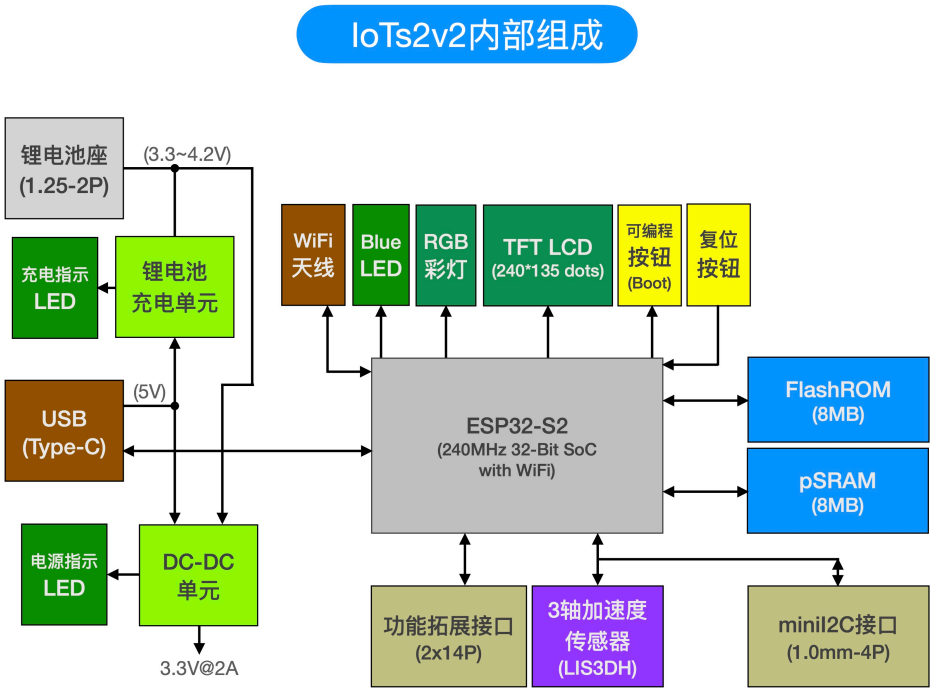
IoT(Internet of Things) 是近 10 年持续的热词，从概念到落地、普及应用，刚过去的数年内的很多种成功的创新商业模式几乎都有 IoT 技术，譬如共享 xx、智能 xx 等。我们想要了解和学习 IoT 技术，是否有容易的、低成本的方法呢？IoTs2 正是为此目的而设计，几十元的成本、几十个小时的验证即可掌握 IoT 的核心技术细节。

IoTs2 是一种小体积的、低功耗的、低成本的、高计算性能的 IoT 型嵌入式系统。IoTs2 不仅支持 UART(或 RS485)、CANBus 等有线的低速实时网络接口，以及 USB 接口，还支持 WiFi(2.4GHz) 高速无线网络接口。除了板载的可编程按钮、指示灯、RGB 彩灯、视网膜级彩色 LCD、加速度传感器等资源，IoTs2 具有 18 个可编程 I/O(呈 DIP 排列)。

为了方便我们了解 IoTs2 的资源，下图给出 IoTs2 内部单元的组成框图。


锂电池座
(1.25-2P)
(3.3~4.2V)
充电指示
LED
锂电池
充电单元
WiFi
天线
Blue
LED
RGB
彩灯
TFT LCD
(240*135 dots)
可编程
按钮
复位
按钮
FlashROM
(8MB)
(5V)
USB
(Type-C)
ESP32-S2
(240MHz 32-Bit SoC
with WiFi)
pSRAM
(8MB)
电源指示
LED
DC-DC
单元
3.3V@1.2A
功能拓展接口
(2x14P)
3轴加速度
传感器
(LIS3DH)
miniI2C接口
(1.0mm-4P)

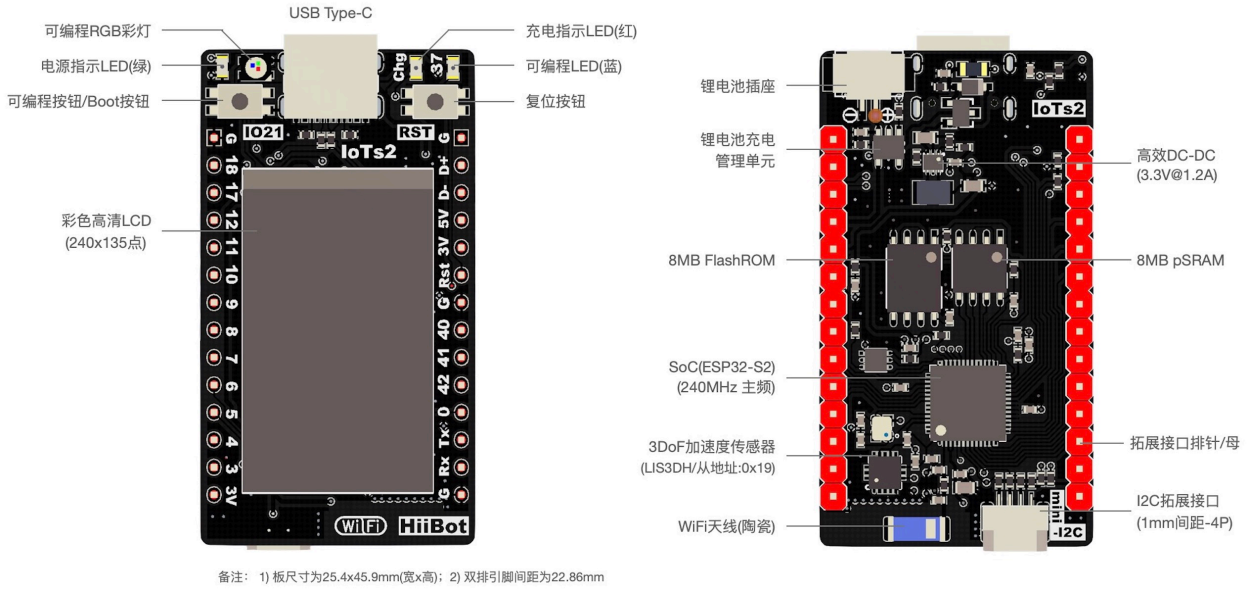
(IoTs2_v1)




(IoTs2_v2)

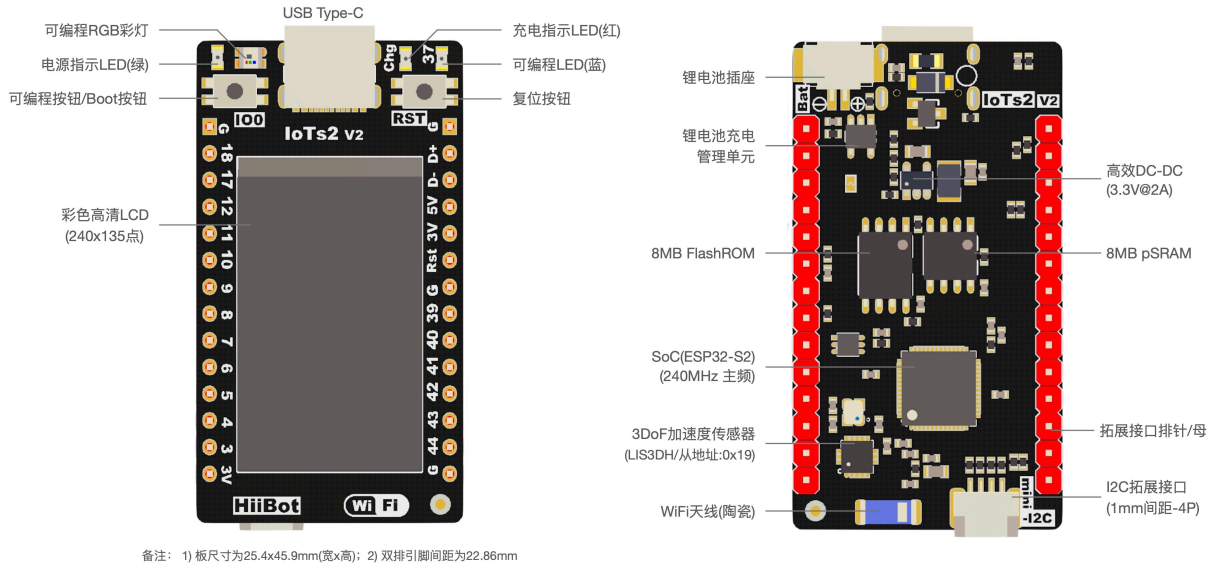
### 1.1.2 IoTs2 板载资源

现在可以进一步认识 IoTs2，了解 IoTs2 的板载资源。IoTs2 具有丰富的板载资源，如下图所示。




(IoTs2_v1)

## IoTs2v2的板载资源




* 开源硬件(在线向导中提供硬件电路)，开源软件(可在CircuitPython官网下载最新版固件及源码)
* 支持Python(推荐的编程语言)，支持Scratch(易造云在线编程)，兼容Arduino，支持ESP IDF(乐鑫)

(IoTs2_v2)

**主控制器 (SoC)**：240MHz 32-Bit Soc，片内带有 2.4GHz WiFi(支持 IEEE 802.11 b/g/n，最高数据速率可达 150M)。

**存储器资源**：SoC 片内 320KB SRAM 和 128KB FlashROM，片外扩展 (QSPI)8MB pSRAM(伪静态 RAM) 和 8MB FlashROM。

**传感器与输入资源**：3 轴加速度传感器 (运动姿态、摇晃和敲击等运动感知)，1 个可编程按钮输入，10 个人体触摸输入。

**显示器和输出资源**：240*135 点阵近视网膜级高清彩色 LCD 显示器，1 个可编程 RGB 彩灯，1 个可编程蓝色 LED。

**板载供电单元**：电池插座，单节锂电池充电管理单元，高效的开关型 DC-DC(IoTs2_v1: 3.3V@1.2A 输出，约 4W 功率; IoTs2_v2: 3.3V@2A 输出，约 6.5W)，板载充电 LED，电源 LED。

**功能拓展接口**：带有 1.0mm 间距的 4P miniI2C 接口 (或称 qwIIC)，2.54 间距的 2*14P 拓展接口。

**通讯接口**：USB Type-C 接口 (可编程为 HID[鼠标/键盘类设备]、CDC[虚拟串口类设备] 和 MSC[大容量存储器/可移动磁盘类设备] 等三类常用 USB 通讯类)，WiFi(支持 IEEE 802.11 b/g/n，板载高效能的陶瓷天线)，2xI2C、4xSPI、2xUART、1xCANBus/TWI、1xI2S 等串行通讯接口。

**编程语言**：支持三种编程语言和 4 种环境。Scratch 图形化语言，易造云 (https://www.ezaoyun.com) 在线编程；C/C++ 语言，Arduino IDE(安装和环境搭建详见 https://github.com/espressif/arduino-esp32) 和 ESP-IDF；Python 语言，建议使用 MU 编辑器 (安装详见 https://codewith.mu/)，板上带有数十个 Python 示例程序 (Python API 接口详见 https://circuitpython.readthedocs.io/en/latest/docs/index.html)

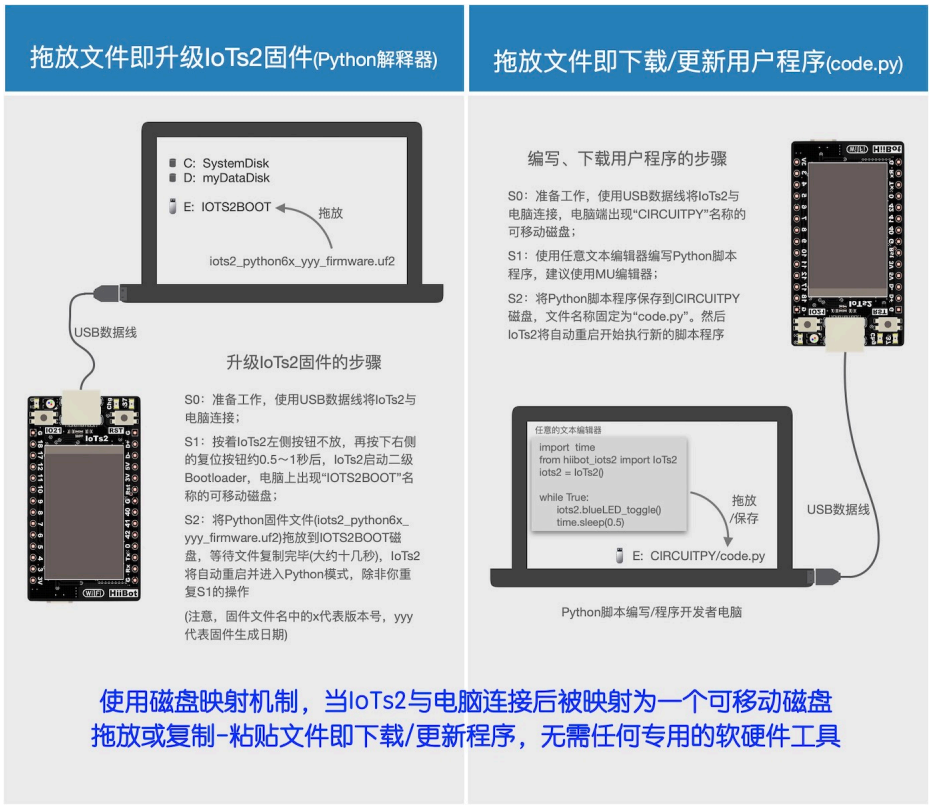
### 1.1.3 使用 Python 对 IoTs2 编程

Python 是过去的几年内普及速度最快的一种编程语言，尤其在网络编程应用领域，Python 几乎是独角兽。目前，Python 作为嵌入式系统的编程语言的相关技术发展速度很快，嵌入式 C++ 编程的普及得益于活跃的 Arduino 开源社区，嵌入式 Python 编程技术大有赶超 C++ 的趋势，不仅少了编译的步骤，脚本 (文本) 文件 ? 二进制文件的下载在工具成本和速度等方面也极具优势。很显然，从网络编程、程序编译和下载等方面考虑，使用 Python 编程语言学习 IoT 技术是最优的选择。

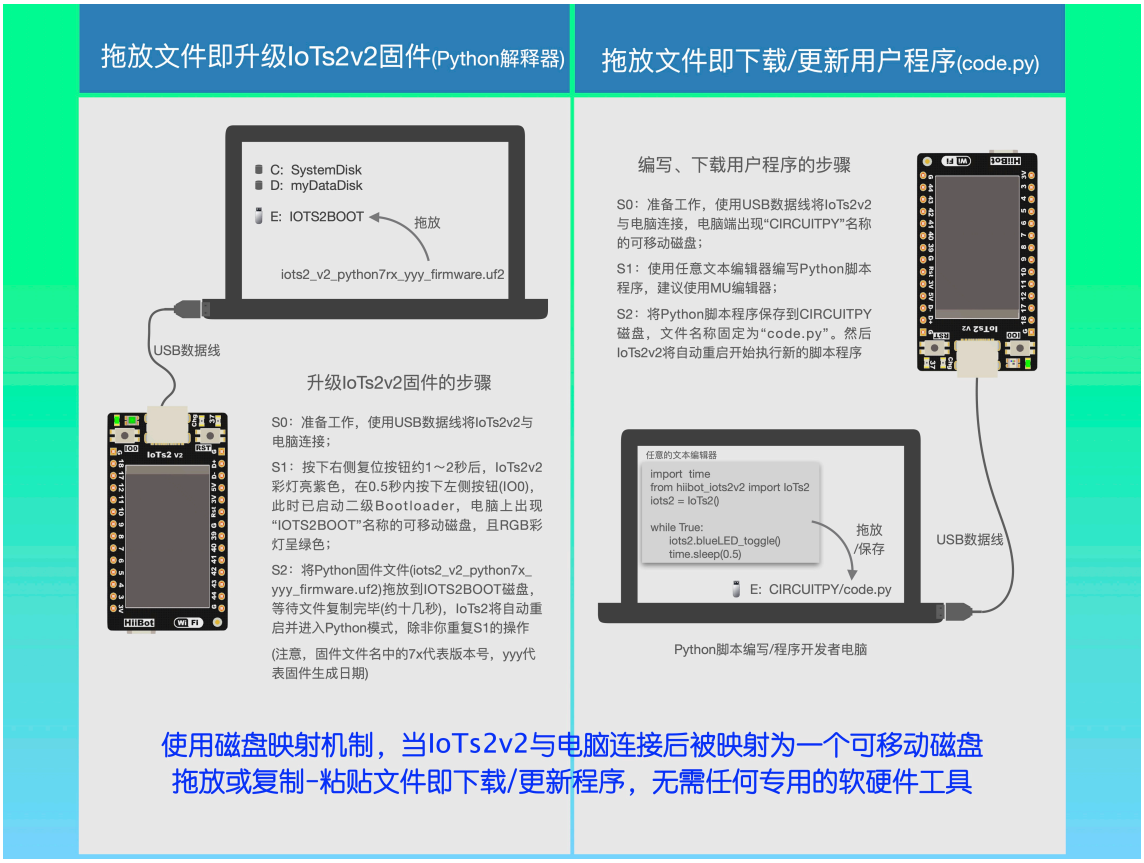
对 IoTs2 编程应用时，首选的编程语言是 Python。IoTs2 默认的编程环境和编程语言是 Python，当 IoTs2 出厂时已默认安装 Python 解释器，使用 USB 数据线将 IoTs2 与电脑连接后只需要修改并保存 CIRCUITPY 磁盘上的 code.py 脚本 (文本) 文件即可改变 IoTs2 的功能。

啊哈！嵌入式系统编程已变得如此简单！必须归功于嵌入式 Python 编程语言相关的技术发展。当我们使用 USB 数据线将 IoTs2 与电脑正确连接后，IoTs2 是一个可移动磁盘 (默认的磁盘名称为 CIRCUITPY)，添加或删除 Python 库文件、维护图片和声音等资源文件、修改用户的脚本程序文件 (code.py) 都如同我们在 Windows、macOS、Linux 系统文件资源管理器中的操作一样。这是你记忆中的嵌入式系统开发环境吗？xxIDE 等庞大的专用的嵌入式系统软件开发环境，JTAG、ICE 等专用的硬件下载工具，使用 Python 对 IoTs2 编程时这些统统不需要了，一个文本编辑器的应用程序 (这是所有桌面 OS 自带的 App) 即可。

IoTs2 的磁盘映射分为 2 种：BootLoader 磁盘和 Python 解释器磁盘。进入两种磁盘的方法和两种磁盘的使用方法如下图所示：


拖放文件即升级IoTs2固件(Python解释器)
拖放文件即下载/更新用户程序(code.py)
C: SystemDisk
D: myDataDisk
E: IOTS2BOOT
拖放
iots2_python6x_yyy_firmware.uf2
USB数据线
升级IoTs2固件的步骤
S0：准备工作，使用USB数据线将IoTs2与电脑连接；
S1：按着IoTs2左侧按钮不放，再按下右侧的复位按钮约0.5～1秒后，IoTs2启动二级Bootloader，电脑上出现"IOTS2BOOT"名称的可移动磁盘；
S2：将Python固件文件(iots2_python6x_yyy_firmware.uf2)拖放到IOTS2BOOT磁盘，等待文件复制完毕(大约十几秒)，IoTs2将自动重启并进入Python模式，除非你重复S1的操作
(注意，固件文件名中的x代表版本号，yyy代表固件生成日期)
编写、下载用户程序的步骤
S0：准备工作，使用USB数据线将IoTs2与电脑连接，电脑端出现"CIRCUITPY"名称的可移动磁盘；
S1：使用任意文本编辑器编写Python脚本程序，建议使用MU编辑器；
S2：将Python脚本程序保存到CIRCUITPY磁盘，文件名称固定为"code.py"。然后IoTs2将自动重启开始执行新的脚本程序
任意的文本编辑器
import time
from hiibot_iots2 import IoTs2
iots2 = IoTs2()
while True:
iots2.blueLED_toggle()
time.sleep(0.5)
拖放
/保存
E: CIRCUITPY/code.py
USB数据线
Python脚本编写/程序开发者电脑
使用磁盘映射机制，当IoTs2与电脑连接后被映射为一个可移动磁盘
拖放或复制-粘贴文件即下载/更新程序，无需任何专用的软硬件工具

(IoTs2_v1)

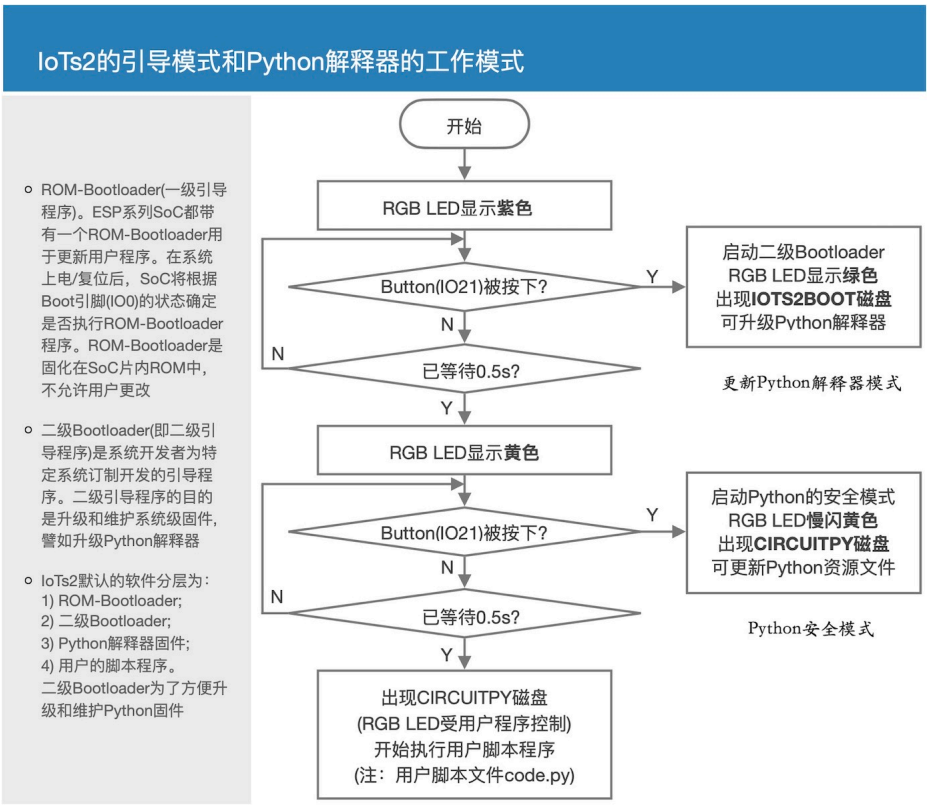



(IoTs2_v2)

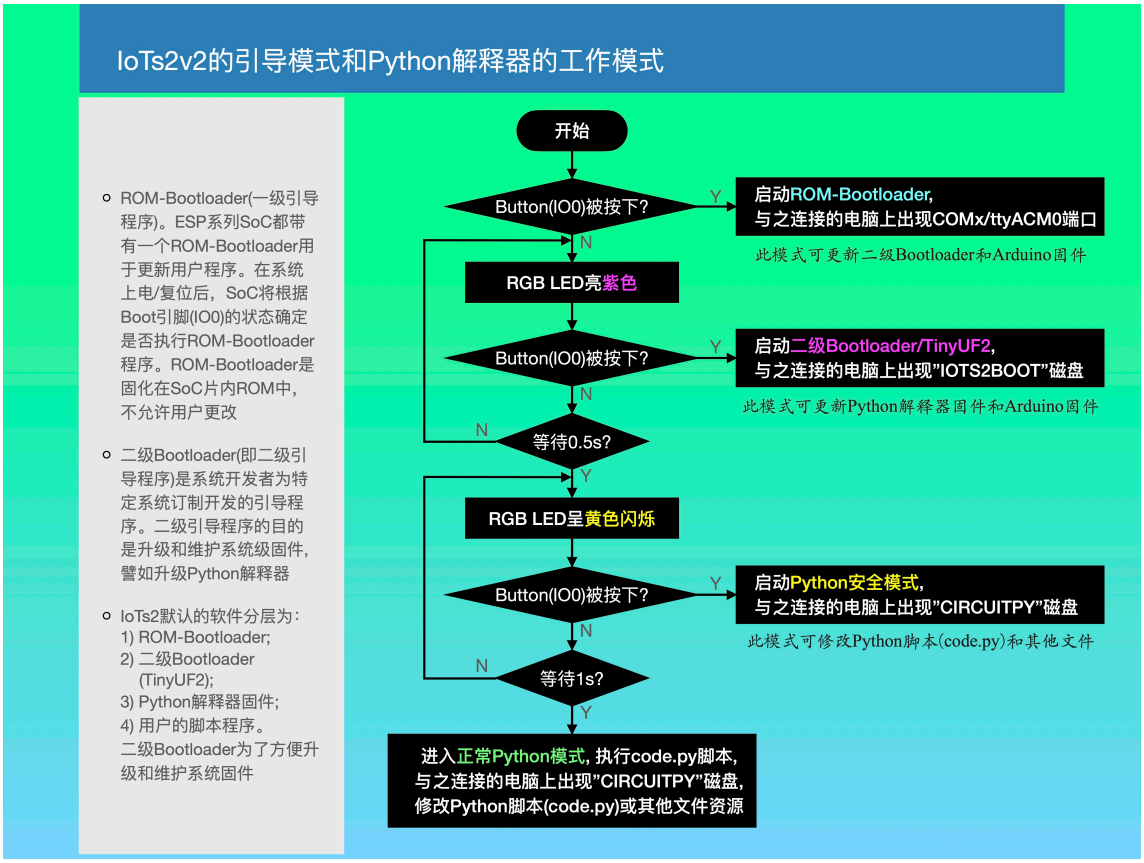
注意：IoTs2v2 升级 Python 固件的方法略有改变，主要原因是二级 Bootloader 已经改变。

BootLoader 磁盘的名称为 IOTS2BOOT，将 IoTs2 的固件文件拖放到该磁盘即可更新/升级 Python 解释器；Python 解释器磁盘的名称为 CIRCUITPY，将我们编写的 Python 脚本程序保存到该磁盘根目录的 code.py 文件即为下载/更新应用程序。

IoTs2 默认带有 Python 解释器固件，使用 IoTs2 板载的可编程按钮和复位按钮可以让 IoTs2 进入三种不同的模式：BootLoader 模式 (可更新 Python 解释器固件)、Python 的正常模式 (立即执行 code.py)、Python 的安全模式 (不执行 code.py)。进入这些模式的操作流程如下图所示。


IoTs2的引导模式和Python解释器的工作模式
ROM-Bootloader(一级引导程序)。ESP系列SoC都带有一个ROM-Bootloader用于更新用户程序。在系统上电/复位后，SoC将根据Boot引脚(IO0)的状态确定是否执行ROM-Bootloader程序。ROM-Bootloader是固化在SoC片内ROM中，不允许用户更改
二级Bootloader(即二级引导程序)是系统开发者为特定系统订制开发的引导程序。二级引导程序的目的是升级和维护系统级固件，譬如升级Python解释器
IoTs2默认的软件分层为：
1) ROM-Bootloader;
2) 二级Bootloader;
3) Python解释器固件;
4) 用户的脚本程序。
二级Bootloader为了方便升级和维护Python固件
开始
RGB LED显示紫色
Button(IO21)被按下?
Y
N
已等待0.5s?
N
Y
启动二级Bootloader
RGB LED显示绿色
出现IOTS2BOOT磁盘
可升级Python解释器
更新Python解释器模式
RGB LED显示黄色
Button(IO21)被按下?
Y
N
已等待0.5s?
N
Y
启动Python的安全模式
RGB LED慢闪黄色
出现CIRCUITPY磁盘
可更新Python资源文件
Python安全模式
出现CIRCUITPY磁盘
(RGB LED受用户程序控制)
开始执行用户脚本程序
(注：用户脚本文件code.py)

(IoTs2_v1)




(IoTs2_v2)

注意：IoTs2v2 的工作模式也稍有改变，ROM-Bootloader 模式可更新二级 Bootloader 和 Aruduino 固件；启动二级 Booloader 的方法也稍有改变；等待进入 Python 安全模式的黄色指示灯是闪烁状态 (在 v1 板上不闪烁)。

为什么需要 Python 的安全模式呢？当我们编写的某些 Python 脚本程序并保存到 Python 磁盘 (即 CIRCUITPY 磁盘) 之后或许引起某些严重问题，Python 的安全模式将不执行“问题”程序让我们有机会修改他们。

IoTs2 的 Python 脚本程序是什么样的呢？譬如下面的代码是 Python 脚本程序

```
1 import time
2 from hiibot_iots2 import IoTs2  # IoTs2
3 iots2 = IoTs2()
4 iots2.blueLED_bright = 1.0
5 while True:
6     iots2.blueLED_toggle()
7     time.sleep(0.5)
```

当我们使用 USB 数据线 (注意！USB 数据线，不是 USB 充电线) 将 IoTs2 与电脑连接好之后，大约 1 ~ 2 秒后 IoTs2 将进入 Python 的正常模式，通过复制-粘贴的操作并用上面的代码覆盖/CIRCUITPY/code.py 文件中的程序代码，然后保存该文件。然后，我们会观察到 IoTs2 上的蓝色 LED 呈闪烁状态。修改和更新 IoTs2 的 Python 脚本程序就像这样的“修改-保存”文本文件一样地简单。为什么 IoTs2 的蓝色 LED 会闪烁呢？上面的程序中，前两行分别导入 time 库、IoTs2 库，第 3 行代码将 IoTs2 类实例化为名叫“iots2”的对象，第 4 行代

码则设置蓝色 LED 的亮度属性为 1.0(即最亮)，在无穷循环程序中重复执行 1) 切换蓝色 LED 的状态；2) 睡眠 0.5 秒。

### 1.1.4 IoTs2 支持的其他编程语言

除了 Python 编程语言之外，IoTs2 还支持 C/C++ 等编译型编程语言和 Scratch 等图形化编程语言。

Scratch 图形化编程语言本身是另一种脚本语言——Javascript(看名字像是 Java 语言，事实上与 Java 语言无任何关系)，但是 IoTs2 并不能直接执行 Javascript 程序，我们将 Scratch 图形化程序自动转换为 Python 脚本程序，这样的“语言转换器”是一种代码自动生成器。因此，本质上使用 Scratch 图形化编程语言编写的程序是一种图形化形式的 Python 脚本程序。与直接编辑 Python 脚本代码的形式相比，使用 Scratch 图形化编程语言编写程序时不会出现拼写关键词、变量名等困难，所以 Scratch 图形化编程语言非常适合青少年或初学编程者使用。

如果需要使用 Scratch 图形化编程语言来编写 IoTs2 的应用程序，请使用 Chrome 等浏览器打开 https://www.ezaoyun.com/ 网址，并进入“在线创作”页面，点击“cppBlockly(Scratch)”按钮，即可进入在线版 Scratch 图形化编程环境，在该窗口的右下角“硬件”标签页选择 IoTs2 即可。

C/C++ 等编译型编程语言程序代码必须经过专用的编译器、链接器等软件工具转换成机器码，这意味着我们在使用 C/C++ 对 IoTs2 编程之前必须先在电脑上安装这些工具软件。譬如 Arduino IDE 和 ESP IDF 等应用程序都支持 IoTs2 的 C/C++ 开发。安装 C/C++ 编程语言的开发环境并不是一件难事儿，我们可以根据乐鑫官网 (https://www.espressif.com/) 的相关文档安装 C/C++ 编程语言的开发环境所需要的工具软件或应用程序。乐鑫官方的 Arduino 软件包网址：https://github.com/espressif/arduino-esp32，请参照此链接的说明搭建 Arduino IDE 的开发环境。

---

## 1.2 使用 IoTs2 前的准备工作

使用 IoTs2 之前，你需要了解的相关知识、操作、软件工具等都在这里..

### 1.2.1 安装 MU 编辑器

几乎所有的嵌入式系统或单板机都没有桌面计算机的键盘、鼠标和大屏幕显示器等标准外设，对这些系统编程或开发时都必须借助于桌面计算机。通常，我们把用于开发嵌入式系统或单板机的桌面计算机称作“宿主计算机”，待开发和编程的嵌入式系统或单板机称作“目标计算机”，两者通过 USB 等标准的通讯接口连接。并且，宿主计算机系统必须预先安装必要的编程软件、编译软件和程序下载工具软件。同时，目标计算机系统也应该具备一个小型的软件系统能够与宿主计算机系统连接，下载用户程序。

(备注：上图中以 ARM Cortex-M 系列处理器的单板机开发为例)

IoTs2 已经尽可能地简化宿主计算机上所需要的软件环境，仅使用任何桌面计算机系统都具备的文本编辑器就可以对 IoTs2 编程、下载，宿主计算机上无需任何特殊的编程软件、编译软件和下载工具软件。使用文本

编辑器编辑 Python 代码，保存为 code.py 文件，并将 code.py 文件拖放至 IoTs2 磁盘 (磁盘名称：CIRCUITPY) 即可。

欲善其事必先利其器！如果我们能够借助于更合适的工具，Python 代码编程的效率也必将提升。尤其程序员们最喜爱某些编程环境的语法高亮、输入记忆和自动填充、代码块自动缩进等功能都是简化代码编辑工作，提升编程效率 (减少错误，减少调试时间) 的利器。

随着 Python 语言在全球的流行，大多数桌面计算机系统都已经安装各种各样的 Python 代码编辑器，如 Pycharm、Visual Studio、Sublime Text 等，这些软件可用于 IoTs2 的代码编辑、保存文件到 IoTs2 磁盘。

面向嵌入式系统或单板机 Python 编程的开源软件工具——MU，不仅支持 Python3(适合桌面计算机的数值计算和网络等) 编程，还支持硬件编程。强烈推荐你使用开源的 MU 编辑器作为 IoTs2 的代编编辑、下载和调试工具。

#### 1. 下载 MU 软件包

在浏览器中打开开源 MU 编辑器的官网 (https://codewith.mu)，点击”Download”进入下载页面，根据自己的宿主计算机系统选择下载合适的 MU 软件包。

如果你觉得自己没有固定的电脑可用，甚至于可以下载一个 U 盘上可运行的 MU 软件包。

#### 2. 安装 MU 编辑器

1) 安装到 Windows 电脑

(此动画暂缺)

针对 Windows 系统的详细安装向导请遵照 MU 官网的说明 https://codewith.mu/en/howto/1.0/install_windows

2) 安装到 Apple macOS 电脑

macOS 电脑的 MU 安装包的文件名为 xxx.dmg，具体名称取决于你下载、保存文件时的选择。双击 macOS 系统的 xxx.dmg(App 软件包)，出现一个名称为“mu-editor”可卸载磁盘，打开这个磁盘，你将会看到：

将”mu-editor”图标拖放至”Application”文件夹即可。熟悉 macOS 电脑的人都知道，Apple 推荐使用自己的 App Store 安装 App。显然，这样安装的 MU 编辑器与 App Store 安装的 App 不同，macOS 会提醒你这个 App 涉及安全隐私。开源的 MU 编辑器不涉及电脑的非安全因素，安装和运行 MU 时不必担心他会损坏或影响你的电脑安全。

首次运行 MU 编辑器时，请打开 Application 文件夹，选中“MU”应用程序，点击鼠标右键，并选择“运行”，当 macOS 提示你是否继续执行时，选择信任和继续。第二次及以后再启动 MU 编辑器应用程序，你就可以直接双击或单击 MU 图标即可。

针对 macOS 系统的详细安装向导请遵照 MU 官网的说明 https://codewith.mu/en/howto/1.0/install_macos

#### 3) 制作运行 MU 编辑器的 U 盘

详见 https://codewith.mu/en/howto/1.0/use_portamu

### 1.2.2 将 IoTs2 连接到 MU 编辑器

绝大多数情况，将 IoTs2 连接到宿主计算机系统都是为了更新/下载程序。与其他嵌入式系统或单板机不同的时，当我们使用 USB 数据线将 IoTs2 插入自己的电脑时，IoTs2 被映射成一个“移动磁盘”。根据 IoTs2 的工作状态，这个可移动磁盘使用两个固定的名称：IOTS2BOOT 和 CIRCUITPY。

---

我们会在以下几种情况使用 IoTs2 磁盘：

- 更新 IoTs2 固件 (Python 脚本解释器)
  - 按着 Boot 按钮 (IO21) 同时按下复位按钮并释放复位按下，当 IoTs2 的 RGB 彩灯变为绿色时，释放 Boot 按钮，出现 IOTS2BOOT 磁盘
  - 下载最新的 IoTs2 固件
  - 拖放 IoTs2 固件 (Python 脚本解释器) 到 IOTS2BOOT 磁盘
- 更新 IoTs2v2 固件 (Python 脚本解释器)
  - 按下复位按钮约 1 ~ 2 秒并释放复位按钮，当 IoTs2v2 彩灯亮紫色的 0.5 秒内按下 Boot 按钮 (IO0)，等待 IoTs2v2 的 RGB 彩灯变为绿色时，与之连接的电脑上出现 IOTS2BOOT 磁盘
  - 下载最新的 IoTs2v2 固件
  - 拖放 IoTs2v2 固件 (Python 脚本解释器) 到 IOTS2BOOT 磁盘
- 更新用户程序
  - 将 IoTs2/IoTs2v2 插入电脑 USB 端口，出现 CIRCUITPY 磁盘
  - 将用户程序文件 (必须使用 main.py 或 code.py) 拖放至 CIRCUITPY 磁盘

---

使用 USB 数据线将 IoTs2 与电脑连接好，按着 Boot 按钮 (IO21) 同时按下复位按钮并释放复位按下，当 IoTs2 的 RGB 彩灯变为绿色时，释放 Boot 按钮，出现 IOTS2BOOT 磁盘。在这个状态下，你可以拖放新版固件文件 (必须是 UF2 格式) 到 IOTS2BOOT 磁盘，即可升级固件.

使用 USB 数据线将 IoTs2v2 与电脑连接好，按下复位按钮约 1~2 秒并释放复位按钮，当 IoTs2v2 彩灯亮紫色时按下 Boot 按钮 (Io0)，当 IoTs2 的 RGB 彩灯变为绿色时，出现 IOTS2BOOT 磁盘。在这个状态下，你可以拖放新版固件文件 (必须是 UF2 格式) 到 IOTS2BOOT 磁盘，即可升级固件.

当 IoTs2 被复位或正常通电后，使用 USB 数据线将 IoTs2 与电脑连接好，如果不按 Boot 按钮 (IO21)，IoTs2 自动执行用户的 Python 程序，并在电脑上出现一个名为 CIRCUITPY 的磁盘。在这个状态下，LCD 显示器会显示程序的执行结果，除非用户程序有意将 LCD 屏幕关闭或使用显示器显示其他内容。

当 IoTs2v2 被复位或正常通电后，使用 USB 数据线将 IoTs2v2 与电脑连接好，当 IoTs2v2 彩灯亮紫色期间 (约 0.5 秒), 如果不按 Boot 按钮 (IO0), IoTs2v2 自动执行用户的 Python 程序, 并在电脑上出现一个名为 CIRCUITPY 的磁盘。在这个状态下，LCD 显示器会显示程序的执行结果，除非用户程序有意将 LCD 屏幕关闭或使用显示器显示其他内容。

在出现 CIRCUITPY 磁盘时，IoTs2 上的全部资源都受用户程序控制，LCD 的显示内容、彩灯状态等都与用户程序有关。

---

如果系统未按预期出现相应的磁盘, 请首先检查 USB 数据线和连接的可靠性。如果 IoTs2 的电源指示灯 (IoTs2 正面的左上角的绿色 LED) 都不能正常工作，请更换到其他 USB 端口，电脑的某些 USB 端口或许已经不可靠的或损坏!

---

### 1. 当 IoTs2 与 MU 编辑器成功连接时

如果你的电脑上已经安装了 MU 编辑器程序 (具体安装步骤见前一小节)，请首先打开 MU 编辑器，当 IoTs2 通过 USB 数据线与宿主计算机成功连接时，MU 编辑器会自动侦测 IoTs2，并提示 “检测到新的 CircuitPython 设备”，并提醒你是否切换到 “CircuitPython 模式”。

MU 编辑器 (V1.1 及以后) 支持 6 种不同的模式，其中 4 种类别的硬件编程，MU 编辑器会根据侦测到的硬件类型自动切换到对应模式。

IoTs2 使用 CircuitPython 编程语言，这是一种与 Python3 几乎完全相同的、支持单板机的脚本编程语言。CircuitPython 由美国知名开源开源硬件供应商——Adafruit 发起，并得到全球很多人的维护和支持，同时支持很多种不同的嵌入式硬件系统，方便我们使用流行的 Python 语言开发和设计各种嵌入式计算机系统或单板机。

### 2. CIRCUITPY 磁盘

当 IoTs2 与电脑连接后，自动执行用户程序。用户程序的文件名必须使用 “code.py” 或” main.py”，而且只能选择使用其中一个。用户程序 code.py 文件保存在一个名叫” CIRCUITPY” 的磁盘上。如果你熟悉 Python 编程语言，使用任意的文本编辑器修改” /CIRCUITPY/code.py” 程序文件，然后保存该文件，你会发现 IoTs2 自动执行你修改好的程序。

IoTs2 自动检测用户程序文件是否被修改，如果发现被修改，自动执行软复位 (soft reset)，并开始执行修改后的程序。任何时候，你只要打开/保存” CIRCUITPY” 磁盘上的文件，IoTs2 都将自动执行软复位，重新执行用户程序 (CIRCUITPY 磁盘上名称为 code.py 或 main.py)。

建议你使用 MU 编辑器编写自己的 Python 程序，程序代码编辑完成后，点击 “保存” 按钮，请选择保存到” CIRCUITPY” 磁盘的根目录中，而且程序名称必须使用 code.py 或 main.py(只能选择其中一种)。每次修改程序并保存后，IoTs2 都会自动执行软复位，重新执行新保存的程序。使用 MU 编辑器将给你带来极大的方便。

有 C 语言编程经历的人都知道，C 程序需要专门的软件工具把我们的 C 程序代码转成目标计算机系统的指令，这个过程叫做编译和连接，软件工具叫做工具链。然而，Python 脚本程序可以被许多计算机系统直接执行，无需编译，这种便捷性或许正式 Python 语言在全球流行的关键原因之一。IoTs2 不仅支持 Python 脚本语言，而且采用 USB MSC(大容量存储设备类) 模式，与电脑连接后是一个可移动磁盘，而且磁盘卷标/名称固定为 CIECUITPY，你的 Python 程序文件只需要拖放到 CIRCUITPY 磁盘即可，无需任何专用的软件下载工具。

### 3. 问题及其解决

你在使用过程中，或许会遇到以下问题，这里列举每一种问题的解决方法：

1) IoTs2 与电脑连接后，未出现名为 “CIRCUITPY” 的磁盘，出现 “NONAME” 或其他名称的磁盘

- 电脑的磁盘名称或卷标是可以修改的，你可以将” NONAME” 卷标名称修改为” CIRCUITPY” 即可。

2) IoTs2 与电脑连接后，未出现名为 “CIRCUITPY” 的磁盘，出现名为” IOTS2BOOT” 的磁盘

- 如果按下 Boot 按钮同时按下 IoTs2 的复位按钮然后释放复位按钮，IoTs2 将自动进入 Bootloader 模式。拔掉 USB 数据线，重新插入电脑即可。

3) IoTs2v2 与电脑连接后，未出现名为 “CIRCUITPY” 的磁盘，出现名为” IOTS2BOOT” 的磁盘

- 如果 IoTs2v2 被复位后板上彩灯亮紫色期间 (约 0.5 秒) 按下 Boot 按钮 (IO0)，IoTs2v2 将自动进入二级 Bootloader 模式。拔掉 USB 数据线，重新插入电脑即可。

4) IoTs2v2 与电脑连接后，未出现名为 “CIRCUITPY” 或 “IOTS2BOOT” 的磁盘，但设备管理器中有新增的虚拟串口 COMx/cu.xx/ttyACM0 等

- 如果 IoTs2v2 被复位时并按着 Boot 按钮 (IO0)，IoTs2v2 将自动进入 ROM-Bootloader 模式。拔掉 USB 数据线，重新插入电脑即可。

5) IoTs2 与电脑连接后，未出现任何 (增加的可移动) 磁盘

- 首先确保 IoTs2 正常启动，左上角绿色电源指示灯亮，而且 LCD 屏幕有字符或其他信息。如果 IoTs2 没有正常供电和正常启动，不会出现 CIRCUITPY 磁盘，线检查供电是否正常。
- 首先检查使用的 USB 是否是数据线，市面上很多 USB 供电线，并不是 USB 数据线。更换为 USB 数据线即可。
- 更换电脑的 USB 端口，确保 USB 端口未损坏，且接触良好。

6) IoTs2 需要专用驱动程序吗?

- Windonws7 及之后的系统都无需驱动
- macOS 和 Linux 系统无需驱动

### 1.2.3 保存 code.py 到 IoTs2

IoTs2 要求用户的程序必文件须使用 code.py 或 main.py(两者只能选用一种) 作为文件名！在电脑的任何地方或任意文本编辑器编写好代码文件，保存为 code.py 或 main.py，然后将该文件拖放/拷贝-粘贴到 IoTs2 的 CIRCUITPY 磁盘即可。一旦这个文件被修改，IoTs2 会自动执行软复位，并执行修改后的 code.py 或 main.py。

你或许会担心：程序功能很强大，程序文件自然很大，全部都放在一个脚本程序中，不容易维护。事实上，在 Python 编程世界里，模块 (module) 是一种最科学代码编写和维护的方法。将很大的程序分割为多个小模块，分别实施、调试、测试后，在主程序中导入 (import) 所用到的模块，在各个模块程序中导入自己所用的模块。这种模块化设计是实施大工程和大系统的科学方法，分而治之不仅容易维护，代码也更好管理。

事实上，Python 语言之所以在全球快速流行也正是归功于“模块化”和“导入模块”等特性。很多模块是别人设计好的，你只需要一行导入语句就可以把一个功能强大的模块 (或许这个模块又导入了别人设计的很多模块) 导入到自己的代码中，日趋活跃的开源社区涌现越来越多的模块，用简单几行 Python 脚本程序就可以实现强大功能的程序。如此容易就能“站在众人肩膀上”，著名的 Python 段子“人生苦短，我用 Python”是在称颂 Python 的模块化和导入模块的理念。

如何设计 Python 的程序模块，将在“IoTs2 教程”的“项目级应用教程”中说明。

---

#### 在 MU 编辑器中保存 code.py 到 IoTs2

我们推荐使用 MU 编辑器来编写 IoTs2 的 Python 脚本程序，这个软件工具不仅支持 Python 语法高亮、自动填充使用过的变量名、自动填充 Tab 键保持 Python 程序块结构完整等。在 MU 编辑器中，当我们把脚本程序代码编辑完成后，如果 IoTs2 已经与宿主计算机连接好，点击“保存”按钮，并指定保存文件的名称和路径，将 code.py 或 main.py 文件保存到 CIRCUITPY 磁盘根目录，IoTs2 将立即执行软复位，开始执行新保存的/新修改的这个 code.py 或 main.py。

#### 在其他文本编辑器中保存 code.py 到 IoTs2

其他文本编辑器，如 Pycharm、Visual Studio、Sublime Text 等，都支持 Python 语法高亮、自动填充使用过的变量名、自动填充 Tab 键保持 Python 程序块结构的完整性，也适合于编写 IoTs2 的 Python 脚本程序。如果你使用这些代码编辑器编写好 Python 脚本程序之后，点击文件菜单中的“保存”，并在弹出的窗口中选择输入文件名为“code.py”或”main.py”，保存文件的路径/位置选择“CIRCUITPY”磁盘根目录即可将你的程序文件保存到 IoTs2，IoTs2 将立即执行软复位，开始执行新保存的/新修改的这个 code.py 或 main.py。

你是否有疑问：既然有这么多种支持 Python 代码编程的文本编辑器，为啥推荐使用 MU 编辑器呢? 下一节给出推荐的理由。

### 1.2.4 使用 REPL

认识和使用 REPL 之前，我们需要首先了解一点脚本语言的特性。Python 是最流行的脚本语言之一，我们每天都在使用 JavaScript 脚本语言 (简称 JS) 浏览网页和某些 App，除了 Python 和 JS 之外，PHP、Ruby、Lua、Perl 等都是脚本语言。

与脚本语言相对的编译型语言，如 C/C++ 和 Java 语言。使用编译型语言编写的程序，必须“编译-链接”才能被目标计算机执行。然而，使用脚本语言编程可以实现 REPL 的效果。在正式解释 REPL 之前，你可以这么想象 REPL：输入一行代码，立即执行并给出结果。输入一行代码，执行这一行代码，如此重复。

用脚本语言编写的程序，几乎就是如上所述的“逐行执行，并立即输出执行结果”，而且无需“编译-链接”。

---

#### 1. REPL 是什么？

如果你现在使用英汉词典 (近期出版的)、搜索引擎、wiki 百科等工具查询“REPL”，必然会得到答复。REPL 的正式解释是

- Read-Eval-Print Loop (“读取-求值-输出”的循环)

进一步解释，REPL 就是

- “读取”你输入的一行脚本程序，“求值”是执行该程序语句，并“输出”执行结果，如此循环。

所谓的脚本程序，如 Python 或 JS 等脚本程序都是逐行地“REPL”，计算机自动连续执行这些脚本程序即可实现编程者的意图/任务。REPL 几乎是所有脚本编程语言都支持的一种程序执行模式，编程者输入一行脚本程序，计算机立即执行这个程序语句，并输出执行结果。显然，REPL 是调试脚本程序的模式。如果你调试过 C/C++ 等编译型语言的程序，你肯定喜欢断点 (break point)、单步 (single step) 等功能，这些功能本质上是帮助我们单步执行程序并给出每一步的节行结果，我们从每一步的结果中很快就能发现 bug。

如果你认为自己编写 Python 程序的过程中无需调试，的确不需要 REPL。否则，REPL 就是每一位 Python 编程者的调试工具。

#### 2. 在 MU 编辑器中使用 REPL

REPL 是一种很重要的脚本程序调试工具，每一种脚本语言的 REPL 都有自己的协议和规则。MU 编辑器支持 Python 脚本语言的 REPL，这就是我们推荐使用 MU 编辑器编写 IoTs2 的 Python 脚本程序的理由。

当 MU 编辑器在宿主计算机上运行时，使用 USB 数据线将 IoTs2 与宿主计算机连接，MU 编辑器会自动侦测和切换为 IoTs2 编程模式，点击 MU 编辑器的“串口”按钮，MU 编辑器下方将出现一个特殊的窗口，习惯上把这个窗口称作“串口控制台 (console)”。在 MU 编辑器的控制台中，输入“ctrl+C”，IoTs2 将立即停止执行 code.py 或 main.py 程序，并进入 REPL 模式，按下回车键，MU 编辑器的控制台将出现“>>”提示符。这就是 MU 编辑器区别于 Pycharm、Visual Studio 和 Sublime Text 等文本编辑器的关键，MU 编辑器天生支持 REPL。

MU 编辑器的 REPL 使用控制台实现人-机交互 (你输入一个脚本程序语句供 REPL 读取，然后执行，并输出执行结果。然后你再输入下一个脚本程序语句，重复这一过程 (loop))。

如上图所示，在控制台的”>>”提示符后面输入“import random”，即导入 Python 内建的”random”模块 (随机数发生器模块)；然后输入“r=random.randint(0, 100)”脚本语句并按回车键表示输入完毕，REPL 将立即执行该语句，即生成一个 0~100 之间的一个随机的整数，并把这个数据赋于一个名叫“r”的变量；最后输入“r”并按回车键，REPL 立即将这个随机数输出到控制台。

当你需要调试 Python 程序时，REPL 就是“单步执行”Python 程序的模式；REPL 就是调试 Python 程序最佳工具。

#### 3. REPL 的妙用 1——help(“modules”)

如果你想了解 IoTs2 支持多少种内建的 (built-in) 模块，在“>>”提示符后面输入“help(“modules”)”，你将会看到以下的执行结果输出：

你是否发现前面用过的“random”在其中吗?

#### 4. REPL 的妙用 2——help(random)

如果你想了解一个模块所支持的全部 API 接口，尝试使用“help(modulename)”。以 Python 内建的”random”为例，在 REPL 模式首先输入“import random”并按回车键，即导入 random 模块；然后输入“help(random)”，IoTs2 将会想控制台输出内建的”random 模块”所支持的所有接口都列举出来，如下图所示：

#### 5. REPL 的妙用 3——dir(random)

在导入”random”模块之后，使用“help(random)”语句将会给我们列举 random 模块所支持的全部类 (class)、变量和方法 (function)。仍以 random 模块为例，使用”dir(random)”将以列表 (list) 格式显示出 random 模块所支持的类、变量和方法名称。如下图所示:

#### 6. REPL 的妙用 4——random. (+tab 键)

如果你曾经使用过一些支持面向对象编程的编辑软件，在编辑程序时，“输入一个对象名称和点，然后按 Tab 键，编辑软件会立即把这个对象的所有接口方法都列举出来”这种辅助式交互可以让程序员不必记住一个对象的全部接口，用到时输入“ClassName.”并按 Tab 键，编辑软件会帮你列出所有方法，选择即可。Python 的 REPL 也具备这一辅助功能。以 random 模块为例，导入 random 模块后，再“>>”提示符后面输入“random.”并按 Tab 键，REPL 会把 random 模块支持的所有类、变量和方法全部列出来。

使用 IoTs2 学习 Python 编程，你不必记住每一个模块的全部接口，当你需要了解一个模块有哪些具体接口 (包括类、变量、方法等) 时，在串口控制台按”Ctrl+C”终止当前正在执行的程序，进入 REPL 模式，输入”

import modulename”并按回车，然后输入“modulename.”并按 Tab 键，你将看到这个模块所支持的接口。以 random 模块为例，如下图：

---

**Tip:**

- **进入 REPL 的方法：** 在串口控制台 (鼠标停留在控制台窗口，点击鼠标左键)，同时按下“Ctrl+C”键，即可进入 REPL 模式，出现 REPL“>>”提示符
- **退出 REPLD 的方法：** 在串口控制台 (鼠标停留在控制台窗口，点击鼠标左键)，同时按下“Ctrl+D”键，即可退出 REPL 模式，IoTs2 立即重新开始执行 code.py 或 main.py 程序

---

### 1.2.5 使用 IoTs2 的 LCD 显示器

IoTs2 配置有一个 240x135 点阵的 1.14 英寸彩色 LCD 显示器，这个小显示屏的点阵密度几乎接近视网膜屏，即使采用很小的字体的情况下仍保持字符和图案的清晰度。IoTs2 处于 Bootloader 模式时，显示屏自动关闭。进入 Python 模式时，执行 code.py 或 main.py 程序时，IoTs2 的 LCD 显示器由用户程序掌控，停止 Python 代码执行或进入 REPL 模式，这个显示屏与串口控制台几乎完全同步。

IoTs2 的 LCD 显示器主要有以下几种用法：

---

#### 1. 作为用户程序的一种输出设备

这个模式下，只要 code.py 或 main.py 程序启动时，屏幕会显示一行提示：“code.py output:” (或“main.py output:”)，提醒我们屏幕的内容都是由 code.py 或 main.py 执行结果。譬如，我们编写一个最简单的程序：

```
1 print("hello world")
2 print("i'am IoTs2")
```

将这个输出两行文本到控制台的脚本程序保存到/CIRCUITPY/code.py 文件，如果 IoTs2 已经与宿主计算机上 MU 编辑器链接好，点击“串口”按钮，你在串口控制台的窗口中将会看到以下提示信息：

> Auto-reload is on. Simply save files over USB to run them or enter REPL to disable.
>
> code.py output:
>
> hello world
>
> I’ m IoTs2 将会想控制台输出内建的
>
> Press any key to enter the REPL. Use CTRL-D to reload.

“Auto-reload is on. Simply save files over USB to run them or enter REPL to disable.”这一行信息提示我们，IoTs2 开始自动加载用户程序 (code.py)，禁用 REPL；“code.py output:”这一行信息表示，开始执行 code.py 程序，启

动 code.py 程序时，IoTs2 的 Python 解释器自动给出的提示信息；“hello world” 和 “I’ m IoTs2” 是我们编写的 code.py 程序中的两个语句所输出的结果；“Press any key to enter the REPL. Use CTRL-D to reload.” 这一行信息表示，code.py 程序已经执行完毕，按任意键进入 REPL 模式，按 “Ctrl+D” 键重新执行 code.py 程序。

你在 IoTs2 的 LCD 屏幕上也能看到这些信息，而且完全相同。通过这个示例我们看到，IoTs2 的 LCD 显示器相当于一个小型控制台，用户程序 “print( “xxx” )” 的输出都被作为一整行显示出来，如果文本内容超过一行，并自动换行继续显示。当开始启动用户程序或执行完毕后，系统的提示信息也被显示在 LCD 上。

我们再来看另外一个示例，Python 脚本程序如下：

```
1  # 从 adafruit_turtle 中导入 turtle 画笔模块和颜色模块
2  from adafruit_turtle import Color, turtle
3  # 从 hiibot_iots2 模块导入 IoTs2 类，该类的"screen" 接口即为显示屏
4  from hiibot_iots2 import IoTs2
5  # 实例化 Screen 模块类为 screen
6  iots2 = IoTs2()
7  screen = iots2.screen
8  colors = [Color.ORANGE, Color.PURPLE]
9  # 实例化 turtle 画笔为 turtle, 并使用 screen 作为显示器
10 turtle = turtle(screen)
11 # 落笔
12 turtle.pendown()
13 # 循环 200 次
14 for x in range(200):
15     turtle.pencolor(colors[x % 2])
16     turtle.forward(x)
17     turtle.left(91)
18
19 while True:
20     pass
```

看起来这个示例程序蛮长，其实并不复杂，尤其你如果使用过 Scratch 的画笔模块、Python 的 turtle 模块，这里所用的 turtle 模块用法几乎与他们完全相同。当然这个时候我们并不关心程序的具体细节。将程序代码复制到 MU 编辑器中，并保存到/CIRCUITPY/code.py 文件中，等待几秒看 IoTs2 的 LCD 屏幕输出就可以。

在这个示例中，我们看不到系统的提示信息，只是看到 code.py 程序的执行结果：在 LCD 屏幕上绘制了一个彩色图案。这是因为本示例的第 6 行程序将 IoTs2 的 LCD 显示器当作 turtle 画笔的输出设备使用，画笔程序绘制的几何图案被显示在 LCD 屏幕上，但系统提示等文本信息就不会显示在这里。

如果你将程序的最后两行 (第 18 行和第 19 行) 删除了，包保存到/CIRCUITPY/code.py 文件，再观察执行结果的屏幕显示是什么内容？你认为这是什么原因？

### 2. 输出脚本程序的错误信息

脚本程序的执行过程是“逐行执行并输出结果”的串行模式。如果我们编写的一个 10 行的脚本程序，前 5 行程序没有任何错误，第 6 行程序有一个拼写错误。执行这个脚本程序的效果是，执行到第 6 行包含拼写错误的语句时，脚本解释器无法执行，于是直接终止程序执行，并输出错误信息的提示，无论第 7 ~ 10 行程序是否正确都不会被执行。

以下面程序为例，复制下面代码到 MU 编辑器中，并保存到/CIRCUITPY/code.py 文件：

```
from adafruit_turtle import Color, turtle
from hiibot_iots2 import IoTs2
iots2 = IoTs2()
screen = iots2.screen
colors = [Color.ORANGE, Color.PURPLE]
turtle = turtle(screen)
turtle.pen()
for x in range(200):
    turtle.pencolor(colors[x % 2])
    turtle.forward(x)
    turtle.left(91)
```

第 6 行程序的正确拼写为“turtle.pendown()”，但是上面程序中被我们错误拼写为“turtle.pen()”。IoTs2 在执行这个包含错误拼写的程序时，会出现以下提示：

> Auto-reload is on. Simply save files over USB to run them or enter REPL to disable.
>
> code.py output:
>
> Traceback (most recent call last):
>
> > File “code.py”, line 6, in <module>
> >
> > AttributeError: ‘turtle’ object has no attribute ‘pen’
>
> Press any key to enter the REPL. Use CTRL-D to reload.

显然，第 4 和第 5 行提示信息中已经明确地告诉我们错误的语句是第几行，错误的原因是 turtle 对象没有 pen 这个属性。

IoTs2 的 LCD 显示器虽然很小，但是作用很大！尤其执行 Python 脚本程序的过程中，动态地加载和执行一些存在错误的模块，终止脚本程序的执行，并提示发声错误的位置和错误原因，很容易帮助我们排查错误。如果没有这样的显示和提示机制，程序被终止后编程者就很难定位问题。任何时候，只要用代码编写程序，各种各样的错误是难免的，如果能快速地定位问题，编写代码、调试程序的效率将大大地提升。

### 1.2.6 软件库

很快就来了!!

---

## 1.3 入门级教程

从这里开始使用，你将开始使用 IoTs2 学习 Python 编程及应用。学习进度安排由简单到复杂，只需要你具备能够用电脑编辑文本、保存文件到指定路径和基本的编程思维能力。你能想象得出计算机执行以下每一条短语时的行为?

- if .. else ..
- while ..
- for ..
- sum+=100
- print( “hello world” )

---

学习内容如下

### 1.3.1 正在工作中..

LED 指示灯是最简单的一种输出设备，常用于指示计算机系统的内部状态。在 IoTs2 上有一个可编程控制的蓝色 LED 指示灯。所谓可编程控制的 LED，就是我们可以用程序控制其亮度，以及亮或灭。

首先看一个示例:

```
import time
from hiibot_iots2 import IoTs2
iots2 = IoTs2()
while True:
    iots2.blueLED_bright = 1.0
    time.sleep(0.5)
    iots2.blueLED_bright = 0.0
    time.sleep(0.5)
```

打开 MU 编辑器，点击“新建”按钮，并将本示例代码复制-粘贴的 MU 编辑器中，然后点击“保存”按钮，并在弹出的窗口中输入文件名为“code.py”，保存文件的磁盘为“CIRCUITPY”，路径为该磁盘的根目录。一旦将 code.py 文件保存到 IoTs2 的 CIRCUITPY 磁盘上，IoTs2 立即开始执行这个脚本程序。

这个示例程序的执行效果是: IoTs2 的蓝色 LED 指示灯不停地闪烁。闪烁的 LED 指示灯常用于指示计算机系统正在工作中，如果程序一旦停止，指示灯也停止闪烁。如果你打开 MU 编辑器的串口控制台，按下“Ctrl+C”键让 IoTs2 终止执行 code.py 程序，进入 REPL 模式，你会发现蓝色 LED 指示灯不再闪烁。

下面我们逐行来分析每行脚本程序的效果，就像 REPL 一样地执行程序。

示例代码分析：

- 第 1 行，导入一个 Python 内建的模块“time”
- 第 2 行，从“/CIRCUITPY/lib/hiibot_iots2.py”模块中导入一个名叫“IoTs2”的类
- 第 3 行，将导入的“IoTs2”类实例化为一个实体对象，名叫“iots2”
- 第 4 行，一个无穷循环的程序块
- 第 5 行 (无穷循环程序块的第 1 行)，iots2 实体对象的 blueLED_bright 属性 (蓝色 LED 指示灯的亮度) 设置为 1.0
- 第 6 行 (无穷循环程序块的第 2 行)，执行 time 的 sleep 方法，参数为 0.5 秒
- 第 7 行 (无穷循环程序块的第 3 行)，iots2 实体对象的 blueLED_bright 属性 (蓝色 LED 指示灯的亮度) 设置为 0.0
- 第 8 行 (无穷循环程序块的第 4 行)，执行 time 的 sleep 方法，参数为 0.5 秒

其中第 1 行和第 2 行都是导入 Python 的模块，为什么有两种不同的写法呢? 第 1 行的导入方法，目的是将整个 time 模块导入到 code.py 中；第 2 行的导入方法仅仅从 hiibot_iots2.py 模块中导入 IoTs2 类，或许 hiibot_iots2.py 模块中还有其他类。事实上，Python 的导入 (import) 模块的方法远不止这两种，如果需要深入了解 import 的其他方法，可以使用网络引擎搜索“python import”查阅更多的介绍。

第 3 行是本示例的重点，执行这个语句的目的是将 IoTs2 类实例化。Python 是一种面向对象编程 (OOP) 的机制，IoTs2 类包含有多种接口或方法，通过类的实例化对象即可访问该类的接口或方法。

第 5 行和第 7 行也是本示例的重点，从两个语句中等号右边的值我们可以想象得出，一个语句是让蓝色 LED 亮度为最大，另一个语句是让蓝色 LED 亮度为 0(即 LED 灭)。我们可以借助于 REPL 模式分别单步执行其中一个语句，并观察蓝色 LED 指示灯的状态。在 MU 编辑器的串口控制台窗口，按下“Ctrl+C”键，让 IoTs2 进入 REPL 模式，然后在“>>”提示符后依次输入以下语句并按“Enter”键执行每一个语句：

```
>> from hiibot_iots2 import IoTs2
>> iots2 = IoTs2()
>> iots2.blueLED_bright = 1.0
>>
```

然后，观察执行“iots2.blueLED_bright = 1.0”之后蓝色 LED 的状态；然后再输入下面语句并按“Enter”键，观察蓝色 LED 的状态：

```
>> iots2.blueLED_bright = 0.0
>>
```

我们在“IoTs2 简介”的内容中已经遇到过下面的示例代码：

```
import time
from hiibot_iots2 import IoTs2
```

(continues on next page)

(continued from previous page)

```
3 iots2 = IoTs2()
4 iots2.blueLED_bright = 1.0
5 while True:
6     iots2.blueLED_toggle()
7     time.sleep(0.5)
```

现在可以仔细来理解每一个语句的作用和执行效果。打开 MU 编辑器，点击“新建”按钮，并将本示例代码复制-粘贴的 MU 编辑器中，然后点击“保存”按钮，并在弹出的窗口中输入文件名为“code.py”，保存文件的磁盘为“CIRCUITPY”，路径为该磁盘的根目录。一旦将 code.py 文件保存到 IoTs2 的 CIRCUITPY 磁盘上，IoTs2 立即开始执行这个脚本程序。

这个示例程序的执行效果：IoTs2 的蓝色 LED 指示灯不停地闪烁。程序执行效果与上面的示例相同！但代码看起来有很多区别，这是为什么？显然，同一种任务可以采用不同的程序软件来实现！

下面我们逐行来分析每行脚本程序的效果。

示例代码分析：

- 第 1 行，导入一个 Python 内建的模块“time”
- 第 2 行，从“/CIRCUITPY/lib/hiibot_iots2.py”模块中导入一个名叫“IoTs2”的类
- 第 3 行，将导入的“IoTs2”类实例化为一个实体对象，名叫“iots2”
- 第 4 行，将 iots2 实体对象的 blueLED_bright 属性 (蓝色 LED 指示灯的亮度) 设置为 1.0
- 第 5 行，一个无穷循环的程序块
- 第 6 行 (无穷循环程序块的第 1 行)，调用 iots2 实体对象的接口函数 blueLED_toggle()
- 第 7 行 (无穷循环程序块的第 2 行)，执行 time 的 sleep 方法，参数为 0.5 秒

从代码分析看，实现 IoTs2 上蓝色 LED 闪烁的关键语句是第 6 行，即调用 iots2 实体对象的接口函数 blueLED_toggle()，这个接口函数可以实现蓝色 LED 的亮/灭状态切换。现在我们尝试将第 4 行代码中“iots2 实体对象的 blueLED_bright 属性值”修改为 0.1，再观察蓝色 LED 的闪烁效果。或者修改为其他值，再观察效果。注意，blueLED_bright 属性值的有效范围：0.0~1.0。

通过反复修改示例中第 4 行的 blueLED_bright 属性值，我们将会发现闪烁时蓝色 LED 的亮度不同。

---

---

**总结：**

- Python 的程序块使用相同的行缩进空格数来界定
- Python 的 import 有很多种用法，本节我们使用过两种方法
- Python 中的导入的类，使用前必须先实例化, iots2=IoTs2()
- 实体对象的属性赋值
  - iots2.blueLED_bright=1.0 # 蓝色 LED 亮度的属性值

- 实体对象的函数调用
  - iots2.blueLED_toggle() # 切换蓝色色 LED 的状态
- 本节中，你总计完成了 8 行代码的编写工作

---

**Important: IoTs2 类的 blueLED 属性和接口**

- blueLED_bright (属性, 可读可写, 有效值：0.0~1.0), IoTs2 蓝色 LED 的亮度
- blueLED_toggle (函数，无参数), 切换 IoTs2 蓝色 LED 的状态

---

### 1.3.2 呼吸灯

前一节中我们已经掌握从 hiibot_iots.py 模块中导入 IoTs2 类，并实例化为 iots2，然后对 blueLED_bright 属性赋不同的值以设置 IoTs2 蓝色 LED 亮度，或者调用函数：blueLED_toggle() 控制 IoTs2 蓝色 LED 的状态切换，也能达到指示灯闪烁的目的。

这一节我们将通过编程改变 LED 亮度实现呼吸效果。首先看下面的示例：

```
1  import time
2  from hiibot_iots2 import IoTs2
3  iots2 = IoTs2()
4  b=1.0
5  while True:
6      iots2.blueLED_bright = b
7      b -= 0.05
8      if b<0.0:
9          b = 1.0
10     time.sleep(0.1)
```

打开 MU 编辑器，点击“新建”按钮，并将本示例代码复制-粘贴的 MU 编辑器中，然后点击“保存”按钮，并在弹出的窗口中输入文件名为“code.py”，保存文件的磁盘为“CIRCUITPY”，路径为该磁盘的根目录。一旦将 code.py 文件保存到 IoTs2 的 CIRCUITPY 磁盘上，IoTs2 立即开始执行这个脚本程序。

仔细观察本示例程序的执行效果，感觉到 IoTs2 在打盹儿吗？蓝色 LED 灯的亮度逐渐减弱 (像人的眼睛慢慢闭上)，然后立即变为最亮 (像突然睁开眼睛)，再逐渐变暗，如此往复。为什么有这种效果？下面我们逐行来分析每行脚本程序的效果。

示例代码分析：

- 第 1 行，导入一个 Python 内建的模块“time”
- 第 2 行，从“/CIRCUITPY/lib/hiibot_iots2.py”模块中导入“IoTs2”类
- 第 3 行，将导入的“IoTs2”类实例化为一个实体对象，名叫“iots2”

- 第 4 行，声明一个变量 “b”，并赋值 1.0
- 第 5 行，开始一个无穷循环的程序块
- 第 6 行 (无穷循环程序块的第 1 行)，将 iots2 对象的 blueLED_bright 属性值设置为变量 b 的值
- 第 7 行 (无穷循环程序块的第 2 行)，将变量 b 的值减小 0.05
- 第 8 行 (无穷循环程序块的第 3 行)，判断变量 b 的值是否小于 0.0
- 第 9 行 (无穷循环程序块的第 4 行)，如果 b<0.0，则 b=1.0
- 第 10 行 (无穷循环程序块的第 5 行)，执行 time 的 sleep 方法，参数为 0.1 秒 (即 100ms)

这个示例中，我们在无穷循环的程序块中不停地将变量 b 减小 0.05，一直减到到 b<0.0 之后再把 b 重新赋值为 1.0，如此重复，而且每重复一次都会把 b 的值赋给 iots2 对象的 blueLED_bright 属性。

变量，允许在程序中改变的量！在本示例中，变量 b 的值顺序地取 {1.0, 0.95, 0.90, .., 0.0} 数据集中的一个值，并把这个数值当作蓝色 LED 的亮度赋值给 “iots2.blueLED_bright” 属性。蓝色 LED 亮度的变化规律正好与数据集中的数值变化规律一致。

下面的示例程序执行结果具有特殊的医学效果：催眠。运行本示例程序时，切勿直视 IoTs2 的蓝色 LED，直视蓝色 LED 太久，我们可能会被催眠！

```
1  import time
2  from hiibot_iots2 import IoTs2
3  iots2 = IoTs2()
4  b = 1.0
5  dir = 0
6  while True:
7      iots2.blueLED_bright = b
8      b += 0.05 if dir==1 else -0.05
9      if b>1.0:
10         b = 1.0
11         dir = 0
12     if b<0.0:
13         b = 0.0
14         dir = 1
15     time.sleep(0.05)
```

你被催眠了吗？这个示例程序的执行效果俗称 “呼吸灯”。蓝色 LED 的亮度从灭逐渐变为最亮，然后又逐渐灭掉，如此重复。这样的周期如果正好与你的呼气-吸气的周期一致，据说很容易把人催眠。

这段程序能够让蓝色 LED 亮度随着我们呼吸节奏改变亮度，其中的关键之处是变量 b 的变化规律。第 8 ~ 14 行程序都是在增加或减少 b 变量的值。你能列举出变量 b 取值的完全数据集？{}

你能用一句既贴切又合适的话来概括变量 b 的变化规律?

---

**总结：**

- 实体对象的属性赋值
- 变量
- 变量赋值
- 变量自增/自减
- 本节中，你总计完成了 15 行代码的编写工作

---

**Important: IoTs2 类的 blueLED 属性和接口**

- blueLED_bright (属性, 可读可写, 有效值：0.0~1.0), IoTs2 蓝色 LED 的亮度
- blueLED_toggle (函数，无参数), 切换 IoTs2 蓝色 LED 的状态

---

### 1.3.3 用按钮给计算机发指令

按钮 (Button) 是计算机系统最简单的一种输入设备，计算机键盘上有 100 个左右的按键 (不同键盘的按键个数不同)，每一个按键就是一个按钮，每一个按钮赋予惟一的编号，如 A 按键的编号为 0x41，B 按键的编号为 0x42，当我们按下按键的按钮时，键盘将对应按键的惟一编号发送给电脑主机，实现人-机交互的输入。

绝大多数嵌入式系统的按键输入比较少，如电梯召唤按钮、轿箱内楼层按钮等都仅有几个或几十个按钮，这些按钮用于人向计算机系统发出指令，按下电梯某楼层的召唤按钮告知电梯控制的计算机系统我需要乘坐电梯，电梯系统收到召唤后即让召唤按钮背后指示灯亮起，表示收到召唤，启动响应。

IoTs2 具有两个按钮，分别是可编程按钮和复位按钮，分别位于 USB 座的左右两侧，我们可以使用这两个按钮向 IoTs2 发指令，譬如开启蓝色 LED、关闭蓝色 LED、复位系统等。当然，复位按钮的优先级最高，任何时候按下复位按钮都可以终止 CPU 执行程序并重新启动。本节重点了解可编程按钮。

---

#### 可编程按钮的输入状态

按钮仅有两个状态：被按下和未被按下。如果被按下时的状态记为“1”，未被按下的状态记为“0”。也可以使用逻辑变量值来表示，按下时的状态记为“True”，未被按下时的状态记为“False”。

我们使用以下程序，将 IoTs2 上按钮的状态显示在屏幕 (和控制台) 上：

```
1 import time
2 from hiibot_iots2 import IoTs2
3 iots2 = IoTs2()
4 while True:
5     print("button:{:d} ".format(iots2.button_state))
6     time.sleep(0.5)
```

打开 MU 编辑器，点击“新建”按钮，并将本示例代码复制-粘贴的 MU 编辑器中，然后点击“保存”按钮，并在弹出的窗口中输入文件名为“code.py”，保存文件的磁盘为“CIRCUITPY”，路径为该磁盘的根目录。一旦将 code.py 文件保存到 IoTs2 的 CIRCUITPY 磁盘上，IoTs2 立即开始执行这个脚本程序。你将会看到 LCD 屏幕 (和控制台) 上看到下面的显示效果：

```
button:0
button:0
button:1
button:1
button:0
```

当我们按下 IoTs2 的可编程按钮时，屏幕上显示“button:1”信息；当 IoTs2 的可编程按钮被释放时，屏幕上显示“button:0”信息。

示例代码分析：

- 第 1 行，导入一个 Python 内建的模块“time”
- 第 2 行，从“/CIRCUITPY/lib/hiibot_iots2.py”模块中导入 IoTs2 类
- 第 3 行，将导入的“IoTs2”类实例化为一个实体对象，名叫“iots2”
- 第 4 行，一个无穷循环的程序块
- 第 5 行 (无穷循环程序块的第 1 行)，将 IoTs2 可编程的按钮的状态值格式化显示到 LCD 屏幕 (控制台) 上
- 第 6 行 (无穷循环程序块的第 2 行)，执行 time 的 sleep 方法，参数为 0.5 秒

第 5 行程序是重点，将变量值 val 格式化为一个字符串：”{:d}” .format(val)。其中，“{}”内的“:d”表示将 val 值显示十进制形式。print( “hello” ) 是将字符串“hello”显示在 LCD 屏幕 (控制台) 上。如果将本示例程序稍作修改，你将会看到另外一种显示效果：

```
import time
from hiibot_iots2 import IoTs2
iots2 = IoTs2()
while True:
    print("button:{}".format(iots2.button_state))
    time.sleep(0.5)
```

修改后的程序只是去掉“{}”内的“:d”，即使用默认的格式化 (系统根据变量 val 的类型自动决定格式化的输出形式)。执行这个示例程序时屏幕上将显示“button:True”(当按钮被按下时)、“button:False”(当按钮未被按下时)。即，

```
button:False
button:True
button:True
button:False
```

### 用按钮开关蓝色 LED

下面这个示例，我们使用 IoTs2 的可编程按钮来切换蓝色 LED 的亮和灭：

```
1 import time
2 from hiibot_iots2 import IoTs2
3 iots2 = IoTs2()
4 while True:
5     iots2.button_update()
6     if iots2.button_wasPressed:
7         iots2.blueLED_toggle()
```

打开 MU 编辑器，点击“新建”按钮，并将本示例代码复制-粘贴的 MU 编辑器中，然后点击“保存”按钮，并在弹出的窗口中输入文件名为“code.py”，保存文件的磁盘为“CIRCUITPY”，路径为该磁盘的根目录。一旦将 code.py 文件保存到 IoTs2 的 CIRCUITPY 磁盘上，IoTs2 立即开始执行这个脚本程序。运行本示例程序时，你会发现程序的效果：每按下可编程按钮一次，蓝色 LED 状态就被切换一次。这个效果像是一个被轻触开关控制的照明灯。

示例代码分析：

- 第 1 行，导入一个 Python 内建的模块“time”
- 第 2 行，从“/CIRCUITPY/lib/hiibot_iots2.py”模块中导入 IoTs2 类
- 第 3 行，将导入的“IoTs2”类实例化为一个实体对象，名叫“iots2”
- 第 4 行，一个无穷循环的程序块
- 第 5 行 (无穷循环程序块的第 1 行)，更新 IoTs2 上的按钮状态
- 第 6 行 (无穷循环程序块的第 2 行，判断条件为 True 时的程序块)，判断 IoTs2 的可编程按钮是否已被按下
- 第 7 行 (无穷循环程序块的第 3 行，条件为 True 时的程序块的第 1 行)，如果按钮已被按下，切换 IoTs2 蓝色 LED 的状态

第 6 行和第 6 行是一个简单的逻辑判断和逻辑程序块，当“iots2.button_wasPressed”为 True 时，执行“iots2.blueLED_toggle()”。

### 用按钮调节蓝色 LED 的亮度

我们将上面的程序稍作修改，即可实现“使用可编程按钮调节蓝色 LED 的亮度”，程序代码如下：

```
1   import time
2   from hiibot_iots2 import IoTs2
3   iots2 = IoTs2()
4   b=0.2
5   dir=1
```

(continues on next page)

(continued from previous page)

```
6  while True:
7      iots2.blueLED_bright=b
8      iots2.button_update()
9      if iots2.button_wasPressed:
10         b += 0.2 if dir==1 else -0.2
11         if b>1.0:
12             b=1.0
13             dir=0
14         if b<0.0:
15             b=0.0
16             dir=1
```

将上面的示例程序保存到 IoTs2 的 CIRCUITPY 磁盘的 code.y 文件，当 IoTs2 执行该程序时，试一试按下 IoTs2 的可编程按钮，你将观察到蓝色 LED 的亮度变化。

示例代码分析：

- 第 1 行，导入一个 Python 内建的模块“time”
- 第 2 行，从“/CIRCUITPY/lib/hiibot_iots2.py”模块中导入 IoTs2 类
- 第 3 行，将导入的“IoTs2”类实例化为一个实体对象，名叫“iots2”
- 第 4 行，声明一个变量 b，并赋初始值为 0.5
- 第 5 行，声明一个变量 dir，并赋初始值为 1
- 第 6 行，一个无穷循环的程序块
- 第 7 行 (无穷循环程序块的第 1 行)，用变量 b 的值更新实体对象 iots2 的 blueLED_bright 属性 (即蓝色 LED 的亮度属性)
- 第 8 行 (无穷循环程序块的第 2 行)，更新 IoTs2 的可编程按钮的状态
- 第 9 行 (无穷循环程序块的第 3 行，判断条件为 True 时的程序块)，判断 IoTs2 的可编程按钮是否已被按下
- 第 10 行 (无穷循环程序块的第 4 行，条件为 True 时的程序块的第 1 行)，如果按钮已被按下，变量 b 的值增加 0.2(如果变量 dir 等于 1) 或-0.2(如果变量 dir 等于 0)
- 第 11 行 (无穷循环程序块的第 5 行，判断条件为 True 时的程序块)，判断变量 b 的值是否大于 1.0
- 第 12 行 (无穷循环程序块的第 6 行，条件为 True 时的程序块的第 1 行)，如果变量 b 大于 1.0，变量 b 设为 1.0
- 第 13 行 (无穷循环程序块的第 7 行，条件为 True 时的程序块的第 2 行)，如果变量 b 大于 1.0，变量 dir 设为 0
- 第 14 行 (无穷循环程序块的第 8 行，判断条件为 True 时的程序块)，判断变量 b 的值是否小于 0.0

- 第 15 行 (无穷循环程序块的第 9 行，条件为 True 时的程序块的第 1 行)，如果变量 b 的值小于 0.0，变量 b 设为 0.0
- 第 16 行 (无穷循环程序块的第 10 行，条件为 True 时的程序块的第 2 行)，如果变量 b 的值小于 0.0，变量 dir 设为 1

你能确定变量 b 的有效数据集吗？根据每个循环中变量 b 的取值列出来即可。上面示例程序的关键每次侦测到按钮被按下后都会调整一次 IoTs2 蓝色 LED 的亮度。如果我们使用 Python 的列表 (list) 将蓝色 LED 亮度值列举出来，也能实现同样的执行效果，但是程序代码更短。代码如下：

```
import time
from hiibot_iots2 import IoTs2
iots2 = IoTs2()
cnt=1
bl = [0.0, 0.2, 0.4, 0.6, 0.8, 1.0, 0.8, 0.6, 0.4, 0.2]
while True:
    iots2.blueLED_bright=bl[cnt%10]
    iots2.button_update()
    if iots2.button_wasPressed:
        cnt += 1
```

将上面的示例程序保存到 IoTs2 的 CIRCUITPY 磁盘的 code.y 文件，当 IoTs2 执行该程序时，试一试按下 IoTs2 的可编程按钮，你将观察到蓝色 LED 的亮度变化与前一个示例代码的执行效果有何区别。

### 按钮的短按和长按

当你一直按着桌面计算机的某个按键时，相当于快速输入很多个相同的字母或数字，IoTs2 的按钮也有相同的效果吗?

为了验证这一设想，我们可以修改前一个示例程序，如果发现长按 IoTs2 的可编程按钮时则直接让变量 b 的值变为 1.0(最大亮度) 或 0.0(最小亮度/灭) 根据当前亮度的增加方向，如果短按该按钮时仍以 0.2 的步长增减变量 b 的值。修改后的程序如下：

```
import time
from hiibot_iots2 import IoTs2
iots2 = IoTs2()
b = 1.0
dir = 0
while True:
    iots2.blueLED_bright = b
    iots2.button_update()
    if iots2.button_wasPressed:        # 是否已被短按
        b += 0.2 if dir==1 else -0.2
        if b>1.0:
            b=1.0
```

(continues on next page)

(continued from previous page)

```
13              dir=0
14          if b<0.0:
15              b=0.0
16              dir=1
17      if iots2.button_pressedFor(2.0):  # 是否已被长按并超过 2.0s
18          b = 1.0 if dir==1 else 0.0
19          dir = 0 if dir==1 else 1
```

修改后的程序仅仅增加最后的 3 行，即第 17 ~ 19 行。第 17 行是条件判断，条件是 IoTs2 的可编程按钮是否已按下超过 2s？如果条件成立则执行第 18 行和第 19 行，第 18 行将变量 b 设为 1.0(如果 dir 等于 1) 或 0.0(如果 dir 不等于 1)，第 19 行将变量 dir 设为 0(如果 dir 等于 1) 或 1(如果 dir 不等于 1)。其他程序语句与前一示例程序完全相同，此处不再赘述。

请在 IoTs2 上测试本示例，检验程序的执行效果是否达到设想：短按 IoTs2 的可编程按钮时蓝色 LED 的亮度将分别减小或增加，长按按钮时蓝色 LED 亮度直接变为 0.0 或 1.0。然后试一试修改第 17 的长按时间参数，观察执行效果，并思考为什么是这样的效果。

---

**总结：**

- 按钮输入
- 实体对象的属性的状态
- 变量
- 变量赋值
- 变量自增/自减
- 逻辑判断和逻辑程序块
- 本节中，你总计完成了 19 行代码的编写工作

---

**Important: IoTs2 类的 button 属性和接口**

- button_state (属性, 只读, 有效值：0/False 或 1/True), IoTs2 的可编程按钮的状态
- button_wasPressed (属性, 只读, 有效值：0/False 或 1/True), IoTs2 的可编程按钮是否已被按下
- button_wasReleased (属性, 只读, 有效值：0/False 或 1/True), IoTs2 的可编程按钮是否已被释放
- button_pressedFor (函数, 输入参数: 时长, 返回值:0/False 或 1/True), IoTs2 的可编程按钮是否被长按超过指定的时长
- button_update (函数, 无参数, 无返回值), 更新 IoTs2 的按钮状态, 必须放在循环体内调用

---

### 1.3.4 触摸输入 (无声的按钮)

触摸输入是一种特殊的输入接口，功能几乎与按钮一样，但是没有按钮操作的机械声音，所以触摸输入被称作无声的按钮。

IoTs2 的 2x14P 扩展接口上具有 8 个人体触摸输入引脚，这些引脚是一种特殊的输入接口。这些触摸引脚不仅能感知你触摸他们，还允许你使用鳄鱼夹电线将触摸引脚与某些导体或相当于导体的材料相连接，如锡箔纸、导电不干胶、水果、蔬菜、盛有水的杯子等，用手触摸这些导体或材料时，等同于触摸到 IoTs2 的触摸引脚。

本节我们来了解这些触摸输入的用法。先看第一个示例：

```
import time
import board
from touchio import TouchIn
# list of the Touched Pins
touchPins = [TouchIn(board.IO3),  TouchIn(board.IO4),
             TouchIn(board.IO7),  TouchIn(board.IO7),
             TouchIn(board.IO9),  TouchIn(board.IO10),
             TouchIn(board.IO11), TouchIn(board.IO12)]
# set their threshold (default threshold be equal to 300 + "initial raw_value")
for i in range(8):
    threshold = touchPins[i].raw_value
    touchPins[i].threshold = threshold+300
while True:
    for i in range(8):  # scan all touchPins
        if touchPins[i].value:
            print("touchPin-{} be touched".format(i))
            time.sleep(0.2)
```

将本示例的程序复制-粘贴到 MU 编辑器，并保存到 IoTs2 的/CIRCUITPY/code.py 文件，IoTs2 将立即执行该程序，你可以用手指去触摸 IoTs2 的 IO3、IO4、IO7、IO8、IO9、IO10、IO11 或 IO12 引脚，观察 LCD 屏幕 (控制台) 上的显示信息。如果手指触摸时会影响到相邻触摸引脚，我们可以手握水笔或铅笔并用笔尖触碰这些引脚时，将会观察到同样的效果，而且不影响相邻引脚的状态。控制台输出的信息如下：

```
Auto-reload is on. Simply save files over USB to run them or enter REPL to disable.

code.py output:
touchPin-0 be touched
touchPin-1 be touched
touchPin-7 be touched
touchPin-6 be touched
touchPin-5 be touched
touchPin-4 be touched
touchPin-3 be touched
```

(continues on next page)

(continued from previous page)

```
11 touchPin-4 be touched
```

根据你所看到的程序运行和试验结果，思考本示例的程序语句与测试结果之间的关联关系。

示例代码分析：

- 第 1 行，导入一个 Python 内建的模块“time”
- 第 2 行，导入一个 Python 内建的模块“board”
- 第 3 行，从“touchio”模块中导入 TouchIn 类
- 第 4 行，注释语句
- 第 5 ~ 8 行，声明一个名叫 touchPins 的列表，列表中的各项分别是“TouchIn(board.Px) “，即 IoTs2 的可用触摸输入引脚的实例化对象
- 第 9 行，注释语句
- 第 10 ~ 12 行，使用一个 8 次的循环体分别读取触摸引脚当前状态的原始值，并加上 300 后作为该触摸引脚的阈值
- 第 13 行，一个无穷循环的程序块
- 第 14 行 (无穷循环程序块的第 1 行)，定义一个 8 次的循环程序块
- 第 15 行 (无穷循环程序块的第 2 行，8 次循环程序块的第 1 行)，检测第 i 个触摸引脚是否被触摸
- 第 16 行 (无穷循环程序块的第 3 行，条件判断为 True 时的程序块第 1 行)，检测第 i 个触摸引脚被触摸，则向 LCD 屏幕 (控制台) 输出“touchPin-i be touched”
- 第 17 行 (无穷循环程序块的第 4 行，条件判断为 True 时的程序块第 2 行)，延迟 0.2 秒

请注意，执行本示例程序时，当 IoTs2 初始上电或复位期间不要触摸任何引脚或与触摸引脚相连的电极，第 10 ~ 12 行语句是系统初始化操作的一部分，即使用当前每个触摸引脚的原始值“+300”后当作对应触摸引脚的阈值。这种操作是电容型触摸输入的常规思路。

---

**总结：**

- 人体触摸输入
- 实体对象的属性的状态
- 逻辑判断和逻辑程序块
- 本节中，你总计完成了 17 行代码的编写工作

---

**Important: IoTs2 的 TouchIn 类的接口**

- TouchIn(board.px) (将 board.px 引脚实例化为人体触摸输入引脚)
- value (属性, 只读, 有效值：0/False 或 1/True), 人体触摸引脚的状态 (是否被触摸)
- raw_value (属性, 只读, 有效值：0~65535), 人体触摸输入引脚的原始值 (若大于阈值则视为被触摸，否则未被触摸)
- threshold (属性, 可读可写的, 有效值：0 ~ 65535), 触摸引脚的阈值 (非常重要，初始化触摸引脚时被设置为合适的值)

### 1.3.5 改变 RGB 灯珠的颜色

RGB 像素灯珠串是非常有趣的一种输出，不仅可以用于照明和产生彩光，还能输出动态的流光溢彩的光效。

IoTs2 上仅有 1 颗 RGB 彩色灯珠，虽然不能产生彩色灯珠串那种流光溢彩的光效，但仍可以利用这个灯珠产生有趣的光效。

如果你已经使用 IoTs2 一段时间了，相信你一定注意过 IoTs2 的 RGB 彩色灯珠，在 IoTs2 每次启动时该灯珠的颜色都会切换几次，这些颜色代表不同的 IoTs2 状态，譬如显示紫色时此时按下 Boot 按钮将迫使 IoTs2 进入二级 BootLoader 状态 (可更新 Python 固件)，当显示黄色时按下 Boot 按钮将迫使 IoTs2 进入 Python 的安全模式 (不执行我们的 code.py 程序)。本节我们将掌握如何通过编程控制这个灯珠编程产生特殊的颜色。

第一个示例的代码如下：

```
import time
from hiibot_iots2 import IoTs2
iots2 = IoTs2()
iots2.pixels.auto_write = True
iots2.pixels.brightness = 0.1
colorsList = [(255,0,0),(192,63,0),(127,127,0),(0,255,0),
            (0,127,127), (0,0,255),(127,0,127)]
index = 0
while True:
    iots2.pixels[0] = colorsList[index%len(colorsList)]
    time.sleep(0.5)
    index += 1
```

打开 MU 编辑器，点击“新建”按钮，并将本示例代码复制-粘贴的 MU 编辑器中，然后点击“保存”按钮，并在弹出的窗口中输入文件名为“code.py”，保存文件的磁盘为“CIRCUITPY”，路径为该磁盘的根目录。一旦将 code.py 文件保存到 IoTs2 的 CIRCUITPY 磁盘上，IoTs2 立即开始执行这个脚本程序。运行这个示例程序时，如果你的环境光较暗，可以调低 RGB 像素灯珠的亮度，即减小第 5 行代码的等号右边的值，避免眼睛盯着非常明亮的光源。

示例代码分析：

- 第 1 行，导入一个 Python 内建的模块“time”

- 第 2 行，从“/CIRCUITPY/lib/hiibot_iots2.py”模块中导入一个名叫“IoTs2”的类
- 第 3 行，将导入的“IoTs2”类实例化为一个实体对象，名叫“iots2”
- 第 4 行，设置 iots2 对象的 pixels 的 auto_write 属性 (即 RGB 像素彩灯的自动刷新) 为 True
- 第 5 行，设置 iots2 对象的 pixels 的 brightness 属性 (即 RGB 像素彩灯的整体亮度) 为 0.1，合理取值范围：0.05(亮度最小)~1.0(亮度最大)
- 第 6~7 行，定一个名叫“colorsList”颜色列表，列表中包含 7 种颜色 (即彩虹的颜色) 的 RGB 值
- 第 8 行，定义一个变量 index，并设置其初始值为 0
- 第 9 行，开始一个无穷循环的程序块
- 第 10 行 (无穷循环程序块的第 1 行)，设置 iots2 对象的 pixels[0] 像素灯珠的颜色为 colorsList[index%len(colorsList)]，即颜色列表 colorsList[] 种的某一项
- 第 11 行 (无穷循环程序块的第 2 行)，执行 time 的 sleep 方法，参数为 0.5 秒，即系统空操作 0.5 秒
- 第 12 行 (无穷循环程序块的第 3 行)，将变量 index 增加 1

对照每一行代码的作用，你能根据观察到的本示例程序的执行效果与示例代码逐行地对照上吗?

---

**总结:**

- RGB 像素灯珠
- RGB 三基色
- 子类
- 变量赋值
- 变量自增/自减
- 逻辑判断和逻辑程序块
- 本节中，你总计完成了 12 行代码的编写工作

---

---

**Important: IoTs2 类的 pixel 属性和接口**

- pixels (子类), BlueFi 的 NeoPixel 子类
- pixels.n (属性, 只读, 有效值：1 或更多), IoTs2 的 RGB 像素灯珠的个数 (默认为 1)
- pixels.brightness (属性, 可读可写, 有效值：0.0 ~ 1.0), IoTs2 的 RGB 像素灯珠的亮度
- pixels.auto_write (属性, 可读可写, 有效值：0/False 或 1/True), 自动更新 IoTs2 RGB 像素灯珠的状态

- pixels[0] (属性, 可读可写, 有效值：tuple 型 (R,G,B) 三基色信息), IoTs2 的 RGB 像素灯珠颜色
- pixles.fill( (R,G,B) ) (函数, 输入参数：三基色分量的元组, 无返回值), 填充 IoTs2 的所有 RGB 像素灯珠为指定的颜色
- pixels.show() (函数, 无输入参数, 无返回值), 更新 IoTs2 的 RGB 像素灯珠的状态 (当 pixles.auto_write 为 False 时，改变 pixels 的任何属性值后必须调用此方法更新像素的状态)

---

### 1.3.6 使用 LCD 显示器显示文本

IoTs2 有一个 1.14 寸的彩色 LCD 屏幕，分辨率为 240x135 点阵，点间距和像素点都很小，这个屏幕几何达到视网膜级别 (据说 iPhone 和 iPad 都采用视网膜级别的显示器)，视网膜级显示器上显示文字或图案时非常细腻。在准备阶段我们已经介绍过 IoTs2 的 LCD 屏幕的用途，他是我们的控制台，无论任何时候只要脚本程序遇到错误停止执行时，详细的错误提示信息都会显示在这个屏幕上，方便我们快速排查问题所在，这一功能在执行 Python 等脚本程序的计算机相同中尤为重要。

控制台只接受“print()”方法输出的信息和相同的提示信息等，如果用户编程需要使用 LCD 屏幕显示自己订制的信息，那就需要进一步了解 BlueFi 的 LCD 屏幕的用法。本节仅介绍如何显示简单的文本，虽然只是文本显示，但是颜色、字体大小和位置等都是可编程的。

---

#### 把文字放大显示

用本节的第一个示例程序来回复“BlueFi 的显示文字太小”，当我们把 LCD 屏当作控制台使用使用时，其显示的相同提示等信息尽可能多，采用越大的字体意味着整屏能显示的信息就更少，当你嫌控制台信息的字体过小时，那就不用“print()”来 show 自己的程序输出，参考本示例的方法，相信可以满足你的“显示大文字”需求。示例程序如下：

```
import time
import displayio
import terminalio
from adafruit_display_text import label
from hiibot_iots2 import IoTs2
iots2 = IoTs2()
text_group = displayio.Group(scale=2)
text0 = label.Label(terminalio.FONT, x=0, y=0,  text="", max_glyphs=24,  color=(255,0,
↪0))
text1 = label.Label(terminalio.FONT, x=0, y=9,  text="", max_glyphs=24, color=(192,63,
↪0))
text2 = label.Label(terminalio.FONT, x=0, y=18, text="", max_glyphs=24,  color=(127,
↪127,0))
text3 = label.Label(terminalio.FONT, x=0, y=27, text="", max_glyphs=24,  color=(0,255,
↪0))
```

(continues on next page)

(continued from previous page)

```
12 text4 = label.Label(terminalio.FONT, x=0, y=36, text="", max_glyphs=24,  color=(0,127,
   ↪127))
13 text5 = label.Label(terminalio.FONT, x=0, y=45, text="", max_glyphs=24,  color=(0,0,
   ↪255))
14 text6 = label.Label(terminalio.FONT, x=0, y=54, text="", max_glyphs=24,  color=(127,0,
   ↪127))
15 text7 = label.Label(terminalio.FONT, x=0, y=63, text="", max_glyphs=24,  color=(255,0,
   ↪0))
16 labelList = [text0, text1, text2, text3, text4, text5, text6, text7]
17 for i in range(8):
18     text_group.append(labelList[i])
19 iots2.screen.show(text_group)
20 stateList = ['Released', 'Pressed']
21 while True:
22     stateBtn = iots2.button_state
23     str = "button: {:d} / {}".format(stateBtn, stateList[stateBtn])
24     for i in range(8):
25         labelList[i].text = str
26     time.sleep(0.02)
```

将本示例程序保存到 IoTs2 到/CIRCUITPY/code.py 文件，执行程序的显示效果与控制台的显示效果相对比，你会发现文本字体明显变大，这是因为本示例程序的第 7 行中“scale”参数为 2，该参数的有效值是 1 或 2 的整数倍，如果我们将该参数修改为 1，你会发现什么现象？

下面来分析本示例程序。示例代码分析：

- 第 1 行，导入一个 Python 内建的模块“time”
- 第 2 行，导入一个 Python 内建的模块“displayio”
- 第 3 行，导入一个 Python 内建的模块“terminalio”
- 第 4 行，从“adafruit_display_text”模块中导入“label”类
- 第 5 行，从“/CIRCUITPY/lib/hiibot_iots2.py”模块中导入“IoTs2”类
- 第 6 行，将导入的“IoTs2”类实例化为一个实体对象，名叫“iots2”
- 第 7 行，创建一个“displayio”的“Group”类的实体对象，即一组显示内容，名叫“text_group”，该组最多有 8 个显示内容，且字体放大 2 倍
- 第 8 ~ 15 行，分别定义一个文本标签 (Label)，所有文本标签的字体都是“terminalio.FONT”类型、最多字符个数为 24，每个标签的坐标 (x,y) 和颜色分别指定
- 第 16 行，将定义好的 8 个文本标签组成一个列表便于后面使用遍历列表的方法来访问每个标签
- 第 17 ~ 18 行，定义一个 8 次的循环，遍历标签列表将列表中的文本标签逐个添加到“text_group”中
- 第 19 行，将“text_group”作为显示对象传递给“iots2.screen.show”接口，目的是将“text_group”显示在 IoTs2 的 LCD 屏幕上

- 第 20 行，定义一个字符串列表用来存储按钮的两个状态对应的字符串
- 第 21 行，一个无穷循环的程序块
- 第 22 行 (无穷循环程序块的第 1 行)，将 IoTs2 的可编程按钮的当前状态保存在变量“stateBtn”中
- 第 23 行 (无穷循环程序块的第 2 行)，将按钮状态和描述该状态的字符串格式化成一个字符串，并赋给变量“str”
- 第 24 ~ 25 行 (无穷循环程序块的第 3 ~ 4 行)，定义一个 8 次的循环，将“str”当作标签的文本内容赋给标签列表中的每一项
- 第 26 行 (无穷循环程序块的第 5 行)，执行 time 的 sleep 方法，参数为 0.02 秒

在本示例中，“displayio”模块是最重要的 Python 内建库，使用这个库方便设计屏幕上多种显示内容的布局，虽然本示例仅仅显示多行文本。“displayio”库的基本思路是，将一屏内要显示的所有内容称作一个“group”，一个显示屏 (如本示例中的 iots2.screen) 只能显示一个“group”内容，因此我们需要把多种显示内容添加到一个“group”中 (如本示例中将 8 个文本行逐个添加到“text_group”中 [第 17 ~ 18 行])。当然“displayio”库支持“group”中包含其他“group”(准确地说是“sub-group”)，这样我们就可以把文本、图案、图片等按“group”形式分别组织好，然后再添加到一个大的“group”中，最后再交给一个显示器上 show 出来。关于“displayio”更多的用法，我们将在高级向导中专门介绍。

上面示例的 8 个文本行都是一样的层次，有时候我们需要显示文本标题，该如何处理呢？文本标题和普通文本的主要区别是字体、字体大小等。我们对上面示例代码稍作修改，即可实现文本标题的显示效果，我们假设文本标签“text0”是标题，标题字体是其他文本的字体的 2 倍。注意，前面示例中所有文本的字体大小都是标准字体的 2 倍，如果标题字体大小是其他文本行字体大小的 2 倍，那么标题字体是标准字体的 4 倍，意味着标题行占用更多的屏幕空间。修改后的代码如下：

```
1  import time
2  import displayio
3  import terminalio
4  from adafruit_display_text import label
5  from hiibot_iots2 import IoTs2
6  iots2 = IoTs2()
7  text_group = displayio.Group(scale=2)
8  text0 = label.Label(terminalio.FONT, x=8, y=6,  text="Hi, IoTs2", scale=2, max_
   ↪glyphs=24,  color=(255,0,0))
9  text1 = label.Label(terminalio.FONT, x=0, y=20, text="", max_glyphs=24, color=(192,63,
   ↪0))
10 text2 = label.Label(terminalio.FONT, x=0, y=29, text="", max_glyphs=24,  color=(127,
   ↪127,0))
11 text3 = label.Label(terminalio.FONT, x=0, y=38, text="", max_glyphs=24,  color=(0,255,
   ↪0))
12 text4 = label.Label(terminalio.FONT, x=0, y=47, text="", max_glyphs=24,  color=(0,127,
   ↪127))
13 text5 = label.Label(terminalio.FONT, x=0, y=56, text="", max_glyphs=24,  color=(0,0,
   ↪255))
```

(continues on next page)

(continued from previous page)

```
text6 = label.Label(terminalio.FONT, x=0, y=65, text="", max_glyphs=24,  color=(127,0,
↪127))
testList = [text0, text1, text2, text3, text4, text5, text6]
for i in range(7):
    text_group.append(testList[i])
iots2.screen.show(text_group)
stateList = ['Released', 'Pressed']
while True:
    stateBtn = iots2.button_state
    str = "button: {:d} / {}".format(stateBtn, stateList[stateBtn])
    for i in range(1,7):
        testList[i].text = str
    time.sleep(0.02)
```

修改的示例程序中，文本标签 “text0” 的 (x,y) 坐标、缺省的文本内容、字体放大倍数为 2(再乘以整个 “group” 的放大倍数 2，即为标准字体的 4 倍) 等做了修改。此外，由于 “text0” 占用更大的屏幕空间，我们去掉前示例中的 “text7” (即第 8 行) 文本标签。

修改前后的 IoTs2 显示效果如下图所示：


对于 “adafruit_display_text” 模块的 “label” 类，其详细的属性和参数见 “https://circuitpython.readthedocs.io/projects/display_text/en/latest/api.html?highlight=label”。在这里需要稍作说明，便于理解前面的示例代码。“label” 类的原型和默认参数为：

```
class adafruit_display_text.label.Label(font, *, x=0, y=0, text='', max_glyphs=None,␣
↪color=16777215, background_color=None, line_spacing=1.25, background_tight=False,␣
↪padding_top=0, padding_bottom=0, padding_left=0, padding_right=0, anchor_point=None,
↪ anchored_position=None, scale=1, base_alignment=False, **kwargs)
```

参数说明如下：

- **font**，字体参数。示例中使用 Python 内建的 “terminalio.FONT” 字体，即 “print()” 使用的控制台字体
- **x, y**，分别为文本标签的 x,y 坐标。注意，x 是文本显示的起始横坐标，y 是文本中间的纵坐标 (不是左上角，也不是左下角，而是两者的中间)
- **text**，文本标签的文字内容，字符串类型。注意，文字内容的最大字符个数 (字节数) 不能超过 “max_glyphs” 指定的个数
- **max_glyphs**，指定文本标签的文字内容允许的最大字符个数，即字符串 “text” 参数的最大长度。前面示例中设为 24

- **color**，指定文本标签的文字颜色，即前景的颜色，默认为白色。前面示例中每一个文本标签的颜色都是单独指定的
- **background_color**，指定文本标签的背景颜色，默认为无色 (与屏幕当前的背景色保持一致)。前面示例中使用默认值
- **line_spacing**，文本标签的行间隔，默认为 1.25 倍。前面示例中使用默认值
- **background_tight**，文本标签的背景框是否需要紧紧地围绕着文本，默认值为 False。如果这个参数设置为 True，后面的 4 个参数将被忽略。前面示例中使用默认值
- **padding_top**，围绕着文本标签的背景框与文字内容的上边界之间需要额外的空白像素个数，默认值为 0。前面示例中使用默认值
- **padding_bottom**，围绕着文本标签的背景框与文字内容的下边界之间需要额外的空白像素个数，默认值为 0。前面示例中使用默认值
- **padding_left**，围绕着文本标签的背景框与文字内容的左边界之间需要额外的空白像素个数，默认值为 0。前面示例中使用默认值
- **padding_right**，围绕着文本标签的背景框与文字内容的右边界之间需要额外的空白像素个数，默认值为 0。前面示例中使用默认值
- **anchor_point**，指定文本标签的卯接点参数，卯接点参数是 tuple 型 (x,y) 分别指定横向和纵向的卯接位置，有效值范围：0.0~1.0。(0.0,0.0) 是左上角，(1.0,1.0) 是右下角。默认值为 (0.0, 0.5)，前面示例中使用此默认值
- **anchored_position**，指定问呗标签的卯接点在屏幕上的坐标，该参数是 tuple 型 (x,y) 分别指定横坐标和纵坐标。前面示例中未使用此参数
- **scale**，指定文本标签的字体放大倍数，该参数必须是整数，默认值为 1，即不放大。前面示例中仅标题“text0”文本标签使用该参数
- **base_alignment**，指定文本标签的背景框是否与基线对齐，默认为 False。前面示例中未使用该参数

此外，文本标签“label”类还具有另外几个重要的属性，包括：

- **hidden**，指定文本标签隐藏或显示的属性，正常显示时该属性为 False，如果设置该属性为 True，文本标签将被隐藏 (不显示出来)
- **height**, **width**，只读属性，返回文本标签当前的高度和宽度 (像素个数)
- **bounding_box**，只读属性，返回文本标签当前的背景框的左上角顶点坐标、宽度和高度，返回值的 4 个参数是 tueple 型 (x,y,width,height)

在前面的示例中，无穷循环程序块中仅改变某个文本标签的“text”参数即可刷新屏幕，这说明“displayio”库和“label”类能够自动刷新。那么，我们改变文本标签的 x,y 坐标等其他参数时会发生什么变化呢？改变 x,y 坐标后，自动刷新屏幕后文本标签的位置将会发生改变，譬如下面示例代码：

```
1 import time
2 import random
```

(continues on next page)

(continued from previous page)

```
import displayio
import terminalio
from adafruit_display_text import label
from hiibot_iots2 import IoTs2
iots2 = IoTs2()
text_group = displayio.Group(scale=2)
textLabel = label.Label(
                    terminalio.FONT,     # font of the text label
                    x=20, y=30,          # initial position
                    text="Hello IoTs2", # text content of the label
                    max_glyphs=24,       # the maximal length of the text content
                    color=(255,0,0)      # text color
                    )
text_group.append(textLabel)
iots2.screen.show(text_group)
xb = textLabel.x
yb = textLabel.y
while True:
    xp=random.randint(0, 60)
    yp=random.randint(6, 60)
    steps = max( abs(xp-xb), abs(yp-yb) )
    xdelta = float(xp-xb)/steps
    ydelta = float(yp-yb)/steps
    for i in range(steps):
        textLabel.x = int(xb+(xdelta*i))
        textLabel.y = int(yb+(ydelta*i))
        time.sleep(0.2)
    xb = textLabel.x
    xb = textLabel.y
```

将上面的代码保存到 IoTs2 的/CIRCUITPY/code.py 文件，IoTs2 执行这个示例程序的效果与电脑“文字移动”的屏保效果很相似：红色字符串“Hello IoTs2”将在 IoTs2 的 LCD 屏幕上随机地移动。虽然这个示例代码看起来比较多，初始化部分的代码与前面的示例是相同的，只是这个示例仅使用一个文本标签“textLabel”；在无穷循环程序块中，我们首先产生 2 个随机整数作为本次循环后文本标签的新坐标位置，然后计算每次移动一个像素时需要移动的步数，再用一个有限次循环逐步移动文本标签，移动的方法仅仅是改变文本标签的 x 和 y 坐标，循环最后再用变量将当前坐标位置保存下来。

### 调整屏幕亮度和屏保

如果你想了解 “label” 类的接口都有那些，最快速的查找方法是使用 IoTs2 Python 解释器的 REPL 模式。在 MU 编辑器上，点击 “串口” 按钮，再用鼠标在 MU 编辑器底部弹出的新窗口，按下 “ctrl+c” 键将终止 code.py 程序的执行，立即进入 Python 解释器的通过导入模块或类，使用 “help(className)”、“dir(className)”、“className.” 并按 Tab 键来了解 className 类的接口。

前面的示例中用到的 “iots2” 类的 “screen” 子类也包含多种属性和接口，现在我们以这个子类为例，用 Python 解释器的 REPL 来了解其属性和接口函数。在 MU 编辑器的串口控制台窗口按下 “ctrl+c” 键后，IoTs2 的 Python 解释器将立即终止 code.py 程序，并在控制台输出以下信息 (第一个 “>>>” 之前)：

```
Press any key to enter the REPL. Use CTRL-D to reload.

Adafruit CircuitPython 6.2.0-beta.2-105-gb19e7c914-dirty on 2021-03-02; IoTs2 with␣
↪ESP32S2
>>> from hiibot_iots2 import IoTs2
>>> iots2 = IoTs2()
>>> help(iots2.screen)
object <Display> is of type Display
  show -- <function>
  refresh -- <function>
  fill_row -- <function>
  auto_refresh -- <property>
  brightness -- <property>
  auto_brightness -- <property>
  width -- <property>
  height -- <property>
  rotation -- <property>
  bus -- <property>
```

在 “>>>” 提示符后输入一行 Python 脚本然后按回车键，Pythn 解释器将立即执行这个语句并输出执行结果。当我们输入 “from hiibot_iots2 import IoTs2” 并回车，再输入 “iots2 = IoTs2()” 再回车，此时 REPL 空间已经有一个名叫 “iots2” 的 “IoTs2” 类实体对象，然后再输入 “help(iots2.screen)” 并回车，Python 解释器将会把 “iots2.screen” 子类的接口函数 (function) 和属性 (property) 都列举出来，如上所示。可以看出，“iots2.screen” 子类有 3 个接口函数和 7 种属性。

“iots2.screen” 子类接口函数和属性的说明如下：

- **show()**，将一组 ( “group” ) 显示内容传递给该接口，此组的内容将会显示在屏幕上
- **refresh()**，无参数函数，用于刷新显示器。注意，如果 “auto_refresh” 属性设为 “True”，则无需使用此接口
- **fill_row()**，将一个 “buffer” 型参数传递给该接口以填充屏幕的一行空间。“buffer” 是字节数组 (存储有图案信息)
- **auto_refresh**，可读可写的属性，用于指定屏幕自动刷新操作，默认值是 “True”，即自动刷新

- **brightness**，可读可写的属性，用于指定屏幕的亮度，有效值范围：0.0~1.0，默认是 1.0(最亮)
- **auto_brightness**，可读可写的属性，用于配置自动根据环境光亮度调节屏幕亮度，默认是“False”
- **width**，只读属性，返回屏幕的宽度 (像素个数)。IoTs2 横屏显示时，屏幕宽度像素数固定为 240
- **height**，只读属性，返回屏幕的高度 (像素个数)。IoTs2 横屏显示时，屏幕高度像素数固定为 135
- **rotation**，可读可写的属性，指定屏幕旋转角度，有效值的集为 {0，90，180，270}。IotS2 的默认值是 90，即横屏显示
- **bus**，只读属性，返回显示器使用的接口总线

了解这些属性和接口之后，我们再来看看下面的示例代码：

```
import time
import random
import displayio
import terminalio
from adafruit_display_text import label
from hiibot_bluebox5 import BlueBox5
iots2 = BlueBox5()
text_group = displayio.Group(scale=2)
textLabel = label.Label(
                    terminalio.FONT,     # font of the text label
                    x=20, y=30,          # initial position
                    text="Hello IoTs2", # text content of the label
                    max_glyphs=24,       # the maximal length of the text content
                    color=(255,0,0)      # text color
                    )
text_group.append(textLabel)
iots2.screen.show(text_group)
xb = textLabel.x
yb = textLabel.y
cntDelay = 0
while True:
    xp=random.randint(0, 60)
    yp=random.randint(6, 60)
    steps = max( abs(xp-xb), abs(yp-yb) )
    xdelta = float(xp-xb)/steps
    ydelta = float(yp-yb)/steps
    for i in range(steps):
        textLabel.x = int(xb+(xdelta*i))
        textLabel.y = int(yb+(ydelta*i))
        time.sleep(0.1)
        # quit Screen saver if button be pressed
        if iots2.button_state:
            cntDelay = 0
```

(continues on next page)

(continued from previous page)

```
    xb = textLabel.x
    xb = textLabel.y
    # Screen saver
    cntDelay += 1
    if cntDelay>12:
        iots2.screen.brightness = 0.0
    elif cntDelay>8:
        iots2.screen.brightness = 0.2
    elif cntDelay>6:
        iots2.screen.brightness = 0.5
    else :
        iots2.screen.brightness = 1.0
```

这个示例代码是在前一个示例的基础上稍作修改得到的，增加的代码主要包括注释语句“# Screen saver”下面的部分，以及无穷循环程序块中的有限次循环代码。将本示例代码保存到 IoTs2 的/CIRCUITPY/code.py 文件中，IoTs2 执行该示例程序时，你将会看到更接近电脑“移动文字”屏保的效果：屏幕亮度逐渐变暗直到屏幕被关闭，按下 IoTs2 的可编程按钮后屏幕再次开启显示。

LCD 显示屏的是一种被动显示器，屏幕本身不会发光，必须借助于外接光源才能看到屏幕上的字。因此，绝大多数 LCD 屏幕都会带着一个背光板，一种面积跟 LCD 屏幕完全相同的平面光源，光源被置于 LCD 屏幕后面。LCD 屏背光板是功耗较大的电子元件，而且寿命也比较短，尤其是随着使用时间的增加，背光板的亮度将逐渐降低。为了节能，大部分时间我们不需要看 LCD 显示屏的内容时，我们应该关闭 LCD 屏幕后面的背光板。

本示例程序中，我们仅仅是通过调节“iots2.screen.brightness”属性值来实现屏保效果。

---

**总结：**

- LCD 显示器
- LCD 显示器背光板的亮度和屏保
- 多行文本显示的数据结构
- 文本字体的缩放
- 函数及其定义和调用
- 全局变量和局部变量
- 本节中，你总计完成了 45 行代码的编写工作

### 1.3.7 使用 LCD 显示器画几何图形

本节种我们将继续探索 LCD 屏幕显示的功能，掌握如何在 IoTs2 的彩色 LCD 屏幕上绘制基本几何图形，包括直线、三角形、任意多边形、矩形、圆角矩形和圆，并掌握边框线径、填充与透明等处理。利用这些绘制基本几何图形的方法，我们能够创意出各种各样的几何图案。

---

#### 绘制几何图案并形成“糖葫芦”动画效果

相信你现在能明白本节最后面所列举的基本几何图形接口及其说明，下面的第一个示例程序是非常有趣的动画效果——“吃糖葫芦”。我们知道糖葫芦是由几颗涂满红色糖浆的水果和一根串水果的木棒组成，即包含 2 种几个图案：填充圆和直线。接着上一节的多种显示内容以“group”形式来组织的思维，下面的第一个示例种仅增加绘制基本几何图形的程序。示例程序如下：

```
import time
import displayio
import terminalio
from adafruit_display_shapes.line import Line
from adafruit_display_shapes.triangle import Triangle
from adafruit_display_shapes.polygon import Polygon
from adafruit_display_shapes.rect import Rect
from adafruit_display_shapes.roundrect import RoundRect
from adafruit_display_shapes.circle import Circle
from adafruit_display_text.label import Label
from hiibot_iots2 import IoTs2
iots2 = IoTs2()
iots2.screen.rotation = 180
shape_group = displayio.Group() # 3 sub-groups
# Color variables available
y_initial = 25
RED, GREEN, GOLD = (255, 0, 0), (0, 255, 0), (255, 222, 30)
lines, circles = [], []
# draw 3 lines and append on the lines_subgroup
lines_subgroup = displayio.Group()
for i in range(3):
    lines.append( Line(66+i, 8, 66+i, 230, color=GOLD) )
    lines_subgroup.append(lines[i])
shape_group.append(lines_subgroup)
# draw 6 circles and append on the circle_subgroup
circle_subgroup = displayio.Group()
for i in range( 6 ):
    circles.append( Circle(67,y_initial+(i*20), 16, fill=RED, outline=RED) )
    circle_subgroup.append(circles[i])
```

(continues on next page)

(continued from previous page)

```
shape_group.append(circle_subgroup)
# create a text label
textlabel_subgroup = displayio.Group(scale=2)
textlabel = Label(terminalio.FONT,x=18,y=78,text="",max_glyphs=24, color=GREEN)
textlabel_subgroup.append(textlabel)
shape_group.append(textlabel_subgroup)
# show those content on the IoTs2 screen
iots2.screen.show(shape_group)
while True:
    textlabel.text = 'eat it'
    time.sleep(0.5)
    textlabel.text = 'eating'
    for i in range(len(circles)):
        jy = circles[i].y//3
        for j in range(jy+3):
            circles[i].y -= 3
            time.sleep(0.02)
        circles[i].hidden = True
        time.sleep(0.2)
    lines_subgroup.hidden = True
    textlabel.text = 'again'
    time.sleep(0.5)
    for i in range(len(circles)):
        circles[i].y = y_initial+(i*20)
    lines_subgroup.hidden = False
    for i in range(len(circles)):
        circles[i].hidden = False
```

将上面的代码保存到 IoTs2 的 CIRCUITPY/code.py 文件中，IoTs2 的屏幕上将会呈现一种动画的效果：一串糖葫芦逐个被吃掉，然后又出现一串新的糖葫芦，再被逐个吃掉，如此重复。在这个示例中，我们将用到 adafruit_display_shapes 模块，运行示例程序前，请确认这个库模块已经保存在 CIRCUITPY/lib/文件夹中，参见 IoTs2 的库及其下载向导，自行下载和复制文件到该文件夹中。

示例代码分析：

- 第 1 行，导入一个 Python 内建的模块“time”
- 第 2 行，导入一个 Python 内建的模块“displayio”
- 第 3 行，导入一个 Python 内建的模块“terminalio”
- 第 4 行，从“/CIRCUITPY/lib/adafruit_display_shapes/line.py”模块中导入 Line 类
- 第 5 行，从“/CIRCUITPY/lib/adafruit_display_shapes/triangle.py”模块中导入 Triangle 类
- 第 6 行，从“/CIRCUITPY/lib/adafruit_display_shapes/polygon.py”模块中导入 Polygon 类
- 第 7 行，从“/CIRCUITPY/lib/adafruit_display_shapes/rect.py”模块中导入 Rect 类

- 第 8 行，从 “/CIRCUITPY/lib/adafruit_display_shapes/roundrect.py” 模块中导入 RoundRect 类
- 第 9 行，从 “/CIRCUITPY/lib/adafruit_display_shapes/circle.py” 模块中导入 Circle 类
- 第 10 行，从 “/CIRCUITPY/lib/adafruit_display_text/label.py” 模块中导入 Label 类
- 第 11 行，从 “/CIRCUITPY/lib/hhiibot_iots2.py” 模块中导入 IoTs2 类
- 第 12 行，将导入的 “IoTs2” 类实例化为一个实体对象，名叫 “iots2”，本示例将使用 “iots2.screen” 来访问彩色 LCD 屏幕
- 第 13 行，将 IoTs2 的屏幕方向调整为竖屏且 USB 插座在屏幕上方，即屏幕旋转 180 度
- 第 14 行，声明一个 displayio 类显示元素群 Group 型变量，名叫 “shape_group”
- 第 15 行，注释
- 第 16 ~ 18 行，定义数值变量 y_initial，元组变量 RED、GREEN 和 GOLD，以及列表 lines 和 circles
- 第 19 行，注释
- 第 20 行，定义一个名叫 lines_subgroup 的 displayio 类 Group 子类的实体对象
- 第 21 ~ 23 行，使用一个 3 次的循环绘制 3 个直线，并将每个直线都添加到 lines_subgroup 中
- 第 24 行，将 lines_subgroup 添加到 shape_group 中
- 第 25 ~ 30 行，定义 displayio 类 Group 子类的实体对象 circle_subgroup，并添加 6 个红色填充圆 (6 个糖葫芦的果子)，最后把 circle_subgroup 添加到 shape_group 中
- 第 31 ~ 35 行，定义 displayio 类 Group 子类的实体对象 textlabel_subgroup，并添加一个文本标签 textlabel 到该组，然后把该组添加到 shape_group 中
- 第 36 行，注释。注意，至此我们已经将 “糖葫芦” 的几何元素绘制完毕，并逐个添加在 shape_group 中
- 第 37 行，将 shape_group 作为 iots2 的 screen 的显示对象传递给 “iots2.screen.show()” 函数
- 第 38 行，定义一个无穷循环程序块
- 第 39 行 (无穷循环程序块的第 1 行)，指定 textlabel 的 text 属性值为 “eat it”
- 第 40 行 (无穷循环程序块的第 2 行)，程序暂停 0.5 秒
- 第 41 行 (无穷循环程序块的第 3 行)，指定 textlabel 的 text 属性值为 “eating”
- 第 42~48 行 (无穷循环程序块的第 4~10 行)，用一个嵌套循环让一个 “糖葫芦的果子” 缓慢地移到屏幕顶端并隐藏。外循环是 6 次，即 6 个果子每个果子一次；内循环次数是可变的，由每个果子的 y 坐标确定
- 第 49 行 (无穷循环程序块的第 11 行)，隐藏 lines_subgroup
- 第 50 行 (无穷循环程序块的第 12 行)，指定 textlabel 的 text 属性值为 “again”
- 第 51 行 (无穷循环程序块的第 13 行)，程序暂停 0.5 秒
- 第 52~53 行 (无穷循环程序块的第 14~15 行)，用一个循环将糖葫芦的 6 个果子分别移到原始位置
- 第 54 行 (无穷循环程序块的第 16 行)，显示 lines_subgroup

- 第 55~56 行 (无穷循环程序块的第 17~18 行)，用一个循环将糖葫芦的 6 个果子分别显示出来

虽然示例程序看起来有 50+ 行，但关键的程序功能是绘制直线当作串起“糖葫芦”的木棒，绘制红色填充圆当作“糖葫芦”的果子，并在无穷循环中控制每个果子移动-隐藏，然后再将果子移到初始位置并显示出来，重复这两步就可以形成动画效果。

### 用按钮交互实现“吃糖葫芦”游戏

游戏和动画相比较，游戏增加了交互设计，增加游戏输入和游戏的规则，即可将动画变成游戏。下面的示例是将前一个“糖葫芦”动画效果的示例代码修改为“吃糖葫芦”游戏，游戏的输入仅有一个按钮，即 IoTs2 的板载按钮 (IO21) 或 IoTs2v2 的板载按钮 (IO0)。游戏规则也十分地简单：当按下按钮一次即吃掉一个果子，6 个果子全部吃掉后显示吃果子的总耗时。示例代码如下：

```
import time
import terminalio
import displayio
from adafruit_display_shapes.line import Line
from adafruit_display_shapes.triangle import Triangle
from adafruit_display_shapes.polygon import Polygon
from adafruit_display_shapes.rect import Rect
from adafruit_display_shapes.roundrect import RoundRect
from adafruit_display_shapes.circle import Circle
from adafruit_display_text.label import Label
from hiibot_iots2 import IoTs2
iots2 = IoTs2()
iots2.screen.rotation = 180
shape_group = displayio.Group() # 3 sub-groups
# Color variables available for import.
RED, GREEN, GOLD = (255, 0, 0), (0, 255, 0), (255, 222, 30)
lines, circles = [], []
# draw 3 lines and append on the lines_subgroup
lines_subgroup = displayio.Group()
for i in range(3):
    lines.append( Line(66+i, 8, 66+i, 230, color=GOLD) )
    lines_subgroup.append(lines[i])
shape_group.append(lines_subgroup)
# draw 6 circles and append on the circle_subgroup
y_initial = 25
circle_subgroup = displayio.Group()
for i in range( 6 ):
    circles.append( Circle(67,y_initial+(i*20), 16, fill=RED, outline=RED) )
    circle_subgroup.append(circles[i])
shape_group.append(circle_subgroup)
# create a text label
```

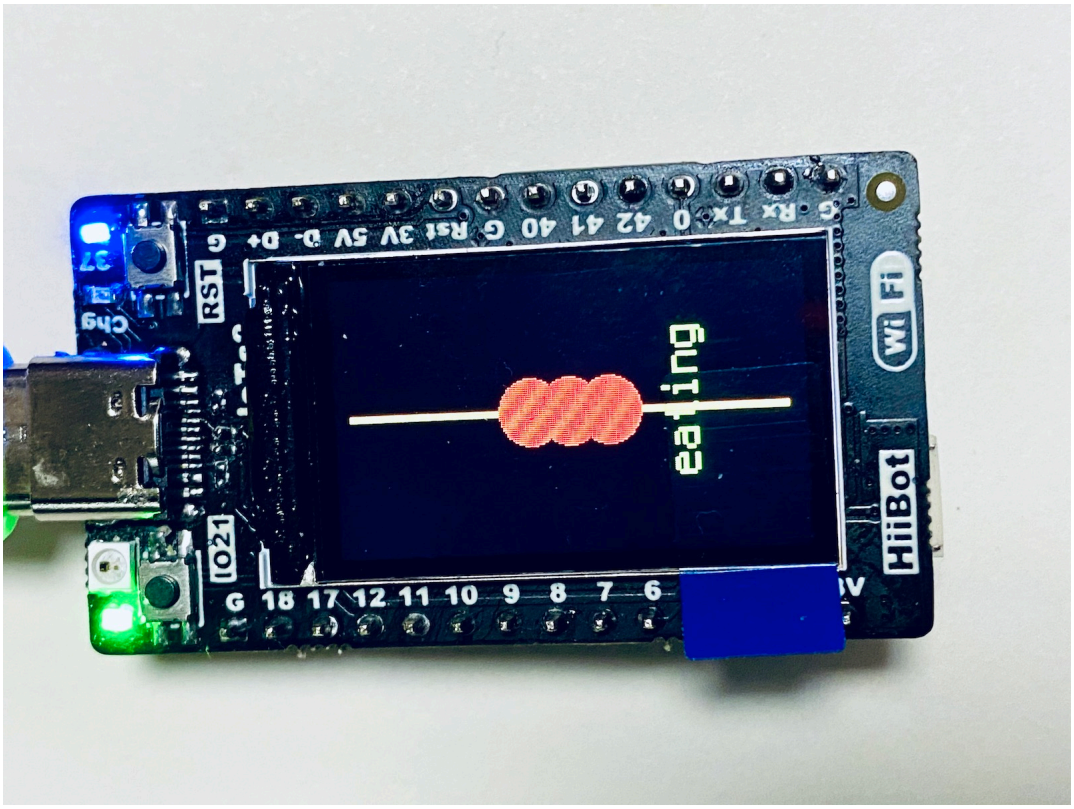
(continues on next page)

(continued from previous page)

```
textlabel_subgroup = displayio.Group(scale=2)
textlabel = Label(terminalio.FONT,x=18,y=88,text="",max_glyphs=24, color=GREEN)
textlabel_subgroup.append(textlabel)
shape_group.append(textlabel_subgroup)
# show those content on the IoTs2 screen
iots2.screen.show(shape_group)
eat_index = 0
textlabel.text = 'eating'
s_point = time.monotonic()
while True:
    if iots2.button_state:
        jy = circles[eat_index].y//3
        for j in range(jy+3):
            circles[eat_index].y -= 3
            time.sleep(0.02)
        circles[eat_index].hidden = True
        eat_index += 1
        if eat_index>=len(circles):
            tl = time.monotonic() - s_point
            eat_index = 0
            for i in range(20):
                lines_subgroup.y += 5
                time.sleep(0.01)
            lines_subgroup.hidden = True
            lines_subgroup.y = 0
            textlabel.text = '{:.3f}'.format(tl)
            time.sleep(0.5)
            for i in range(len(circles)):
                circles[i].y = y_initial+(i*20)
            lines_subgroup.hidden = False
            for i in range(len(circles)):
                circles[i].hidden = False
            time.sleep(0.5)
            textlabel.text = 'eating'
            s_point = time.monotonic()
```

程序的具体细节不必赘述，绝大多数与前一个示例的代码相同，在无穷循环中仅增加一个条件判断——IoTs2 的按钮是否被按下，如果未被按下则什么都不做；如果按钮被按下，模拟吃掉“糖葫芦”的一个果子；没吃掉一个果子还要判断果子是否已全部吃掉，如果是则显示总耗时并再显示一个完整的“糖葫芦”。

本示例程序在 IoTs2 上的执行结果如下图：




如果你把本示例代码保存到 IoTs2 的/CIRCUITPY/code.py 文件中，当你看到 IoTs2 执行该程序的效果时，请你立即按下 IoTs2 的按钮 (IO21) 或 IoTs2v2 的按钮 (IO0)，将会看到一个果子移出“糖葫芦”的木棒 (模拟被吃掉的效果)，再按下按钮后另外一个果子也会被“吃掉”，继续重复，然后看自己总耗时是多久，跟其他人比一比看谁吃得快。这是一中非常有趣的小游戏。

使用本节所 get 的技能，你一定能设计出更有趣的图案、动画或游戏。

---

---

**总结：**

- 基本几何形状及其参数
- 填充与透明
- 颜色的十六进制表示
- 绝对坐标
- 本节中，你总计完成了 108 行代码的编写工作

---

---

---

**Important: adafruit_display_shapes 接口**

- Line (子类), line = Line(x0, y0, x1, y1, color)
  - x0, y0: 起点坐标

  - x1, y1: 终点坐标
  - color: 线颜色，一个十六进制的颜色值，如 0xFF0000

- Triangle (子类), triangle = Triangle(x0, y0, x1, y1, x2, y2, fill=None, outline=None)
  - x0, y0, x1, y1, x2, y2: 顶点坐标
  - fill: 填充颜色选项，=None: 透明/无填充；=0xFF0000: 填充为红色，或其他颜色的十六进制数
  - outline: 外框线颜色，=None: 使用默认的前景色；=0xFF0000: 外框线为红色，或其他颜色的十六进制数

- Polygon (子类), polygon = Polygon(listPoints, outline=None)
  - listPoints: 顶点坐标的列表，如 [(x0, y0), (x1,y1), .., (xm, ym)]
  - outline: 线颜色，=None: 使用默认前景色；=0xFF0000: 外框线为红色，或其他颜色的十六进制数

- Rect (子类), rect = Rect(x, y, width, height, fill=None, outline=None, stroke=1)
  - x, y, width, height: 左上顶点坐标、宽度和高度
  - fill: 填充颜色选项，=None: 透明/无填充；=0xFF0000: 填充为红色，或其他颜色的十六进制数
  - outline: 外框线颜色，=None: 使用默认的前景色；=0xFF0000: 外框线为红色，或其他颜色的十六进制数
  - stroke: 线径 (粗细)，默认为 1，可设定其他值

- RoundRect (子类), rrect = RoundRect(x, y, width, height, r, fill=None, outline=None, stroke=1)
  - x, y, width, height: 左上顶点坐标、宽度和高度
  - r: 圆角半径，最小值为 0，最大值 =min(width//2, height//2)
  - fill: 填充颜色选项，=None: 透明/无填充；=0xFF0000: 填充为红色，或其他颜色的十六进制数
  - outline: 外框线颜色，=None: 使用默认的前景色；=0xFF0000: 外框线为红色，或其他颜色的十六进制数
  - stroke: 线径 (粗细)，默认为 1，可设定其他值

- Circle (子类), circle = Circle(cx, cy, r, fill=None, outline=0xFF0000)
  - cy, cy，r: 圆心坐标和半径
  - fill: 填充颜色选项，=None: 透明/无填充；=0xFF0000: 填充为红色，或其他颜色的十六进制数
  - outline: 外框线颜色，=None: 使用默认的前景色；=0xFF0000: 外框线为红色，或其他颜色的十六进制数

### 1.3.8 加速度传感器

加速度传感器是惯性导航装置中最重要的一个组件，这是一种利于地球重力场的物理特性的传感器，并组合敏感元件、特殊设计的机械结构。加速度计、地磁计、海拔计和陀螺仪都是惯性导航的关键元件，本节内容中仅涉及加速度计。由于加速度传感器是基于地球重力场的物理特性，无法用于地外太空领域。

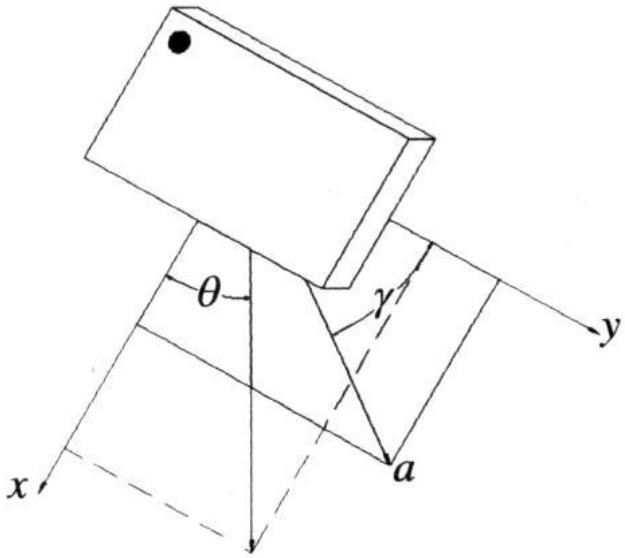
IoTs2 的陶瓷型 WiFi 天线的附近就是加速度计元件，这是一种 3-DoF/3 轴角速度传感器，其体积非常小是因为近些年才发展起来的 MEMS(微机电系统) 技术，以及今天的半导体技术使得这些惯性元件的体积非常小，但在原理上几乎与数十年前使用的体积庞大的加速度传感器相同。

---

#### 重力加速度和姿态感知

今天的惯性测量装置几乎是所有飞行器的关键元件，没有这些元件我们的飞机无法准确地抵达目的地，无人机无法稳定停留在空中帮你拍照。加速度计使用近地空间不变的重力场方向和重力常数 (g)，借助于压电效应、容变、热变等敏感元件、特定机械结构和测量电路，根据牛顿第三定律当物体姿态发生改变时的加速度变化对传感器内部的质量块/热气腔产生反作用力，敏感元件和测量电路能够精确地测量力的大小，从而得到加速度的变化。

根据加速度计的设计，加速度计分为动态型和动静态型两类。采用压电效应的加速度传感器只能感知加速度的“动态”变化，无法感知确定静态时的物体姿态，此类传感器必须借助于姿态估算算法和动态加速度变化信息来确定物体的当前姿态；采用容变和热变的加速度传感器，特殊的机械结构、敏感元件和测量电路设计使得他们不仅能感知加速度的动态变化，还能测量静态的姿态。基于中学的物理知识 (力的分量和合成)，结合下图我们就能想象得出加速度计的基本原理：




此图仅展示一个方向的加速度变化和重力分量。事实上，我们在描述物体的加速度时总会先构建一个三维坐标系，加速度计会输出三个方向的加速度分量。当然也有少部分加速度传感器只能给出 x-、y-两个方向的加速度分量。所以加速度传感器又分为 1D-、2D、3D 的。

IoTs2 采用 3D 的动静态型加速度计，即使在静止状态，IoTs2 的加速度传感器也能给出自己的准确姿势。下面我们使用一个简单示例来观察 IoTs2 给出的加速度变化信息。示例代码如下：

```
import time
import terminalio
import displayio
from adafruit_display_text.label import Label
from hiibot_iots2 import IoTs2
iots2 = IoTs2()
iots2.screen.rotation = 180
# create a text label
textlabel_group = displayio.Group(scale=2)
textLabel_title = Label(terminalio.FONT, x=10, y=10, text="Acce", max_glyphs=10,
↪scale=2, color=(255,0,0))
textLabel_x = Label(terminalio.FONT, x=10, y=32, text="", max_glyphs=24, color=(255,0,
↪0))
textLabel_y = Label(terminalio.FONT, x=10, y=48, text="", max_glyphs=24, color=(0,255,
↪0))
textLabel_z = Label(terminalio.FONT, x=10, y=64, text="", max_glyphs=24, color=(0,0,
↪255))
textlabel_group.append(textLabel_title)
textlabel_group.append(textLabel_x)
textlabel_group.append(textLabel_y)
textlabel_group.append(textLabel_z)
# show those content on the IoTs2 screen
iots2.screen.show(textlabel_group)
while True:
    x, y, z = iots2.Accele_ms2 # Accele_Gs
    textLabel_x.text = "X: {:.2f}".format(x)
    textLabel_y.text = "Y: {:.2f}".format(y)
    textLabel_z.text = "Z: {:.2f}".format(z)
    time.sleep(0.1)
```

将本示例代码保存到 IoTs2 的/CIRCUITPY/code.py 文件中，IoTs2 在执行示例程序时，你可以尝试让 IoTs2 平躺、竖立、倒立、侧立等姿势，并观察三个加速度分量与姿态之间关系，并找出加速度每一个分量的最小值和最大值 (加速度每一个分量的范围)。

示例代码分析：

- 第 1 行，导入一个 Python 内建的模块“time”
- 第 2 行，导入一个 Python 内建的模块“terminalio”
- 第 3 行，导入一个 Python 内建的模块“displayio”
- 第 4 行，从“/CIRCUITPY/lib/adafruit_display_text/label.py”模块中导入一个名叫“Label”的类，即文本标签类

- 第 5 行，从“/CIRCUITPY/lib/hiibot_iots2.py”模块中导入一个名叫“IoTs2”的类，即 IoTs2 模块类
- 第 6 行，将导入的“IoTs2”类实例化为一个实体对象，名叫“iots2”
- 第 7 行，将 IoTs2 的 LCD 屏幕旋转为 180 度，即 USB 座朝上
- 第 8 行，注释
- 第 9 ~ 17 行，定义一个文本标签组 textlabel_group，然后定义 4 个文本标签并添加到 textlabel_group 中，这些文本标签分别为 textLabel_title、textLabel_x、textLabel_y、textLabel_z
- 第 18 行，注释
- 第 19 行，将 textlabel_group 对象当作参数传递给“iots2.screen.show()”函数，即让上述文本标签的内容显示在 IoTs2 的 LCD 显示器上
- 第 20 行，一个无穷循环的程序块
- 第 21 行 (无穷循环程序块的第 1 行)，用”iots2.Accele_ms2”更新变量 x, y, z (iots2.Accele_ms2 是采用 m/s^2 为量纲的加速度值)
- 第 22~24 行 (无穷循环程序块的第 2~4 行)，设置多行文本显示子类的第 2 行的文本内容为 acce[0] 的值
- 第 25 行 (无穷循环程序块的第 5 行)，执行 time 的 sleep 方法，参数为 0.1 秒

建议你设计数据记录表，将本示例给出的加速度三分量的值与姿态之间关系记录下来，然后根据这一关系确定姿态和加速度数据之间变化规律。如果你想要以“相对加速度”(相对于地球重力场的加速度) 的单位的结果，修改上述代码的第 21 行，即用“iots2.Accele_Gs”属性值更新 x, y, z 并观察结果。

我们还可以使用“iots2.angle_RollPitch”属性值获得 IoTs2 当前的姿态角：Roll(翻滚角) 和 Pitch(俯仰角)。修改上述示例代码，增加 2 个文本标签，然后将“iots2.angle_RollPitch”属性值 (2 个元素组成的元组) 当作 2 个标签的 text 属性显示在屏幕上。修改后的代码如下：

```
1  import time
2  import terminalio
3  import displayio
4  from adafruit_display_text.label import Label
5  from hiibot_iots2 import IoTs2
6  iots2 = IoTs2()
7  iots2.screen.rotation = 180
8  # create a text label
9  textlabel_group = displayio.Group(scale=2)
10 textLabel_title = Label(terminalio.FONT, x=9, y=10, text="Acce", max_glyphs=10,␣
   ↪scale=2, color=(255,0,0))
11 textLabel_x = Label(terminalio.FONT, x=1, y=32, text="", max_glyphs=24, color=(255,0,
   ↪0))
12 textLabel_y = Label(terminalio.FONT, x=1, y=48, text="", max_glyphs=24, color=(0,255,
   ↪0))
13 textLabel_z = Label(terminalio.FONT, x=1, y=64, text="", max_glyphs=24, color=(0,0,
   ↪255))
```

(continues on next page)

(continued from previous page)

```
textLabel_roll  = Label(terminalio.FONT, x=1, y=80, text="", max_glyphs=24,␣
↪color=(255,0,255))
textLabel_pitch = Label(terminalio.FONT, x=1, y=96, text="", max_glyphs=24, color=(0,
↪255,255))
textlabel_group.append(textLabel_title)
textlabel_group.append(textLabel_x)
textlabel_group.append(textLabel_y)
textlabel_group.append(textLabel_z)
textlabel_group.append(textLabel_roll)
textlabel_group.append(textLabel_pitch)
# show those content on the IoTs2 screen angle_RollPitch
iots2.screen.show(textlabel_group)
while True:
    x, y, z = iots2.Accele_ms2 # Accele_Gs
    roll, pitch = iots2.angle_RollPitch # angle of roll and pitch
    textLabel_x.text = "x: {:.2f}".format(x)
    textLabel_y.text = "y: {:.2f}".format(y)
    textLabel_z.text = "z: {:.2f}".format(z)
    textLabel_roll.text = "roll: {}".format(roll)
    textLabel_pitch.text = "pitch: {}".format(pitch)
    time.sleep(0.1)
```

修改前后的两种示例代码的执行效果如下图所示：


修改前的效果(仅显示m/s^2的加速度三个分量)


修改后的效果(增加翻滚角(roll)和俯仰角(pitch)的显示)

或许你会问“为什么只有翻滚角和俯仰角?”除了翻滚角和俯仰角，3D 空间物体的姿态角还有航向角 (Yaw angle)，即绕着垂直于地面方向的直线的旋转角。事实上，仅依赖 IoTs2 的 3 轴角速度传感器无法获取航向角信息。

### 用 RGB 像素颜色来表示加速度变化

IoTs2 的 RGB 像素彩灯是一种特殊的显示元件，响应速度快能呈现动感效果。在前一个示例中我们把加速度的三个分量的数值显示在 LCD 屏幕上，当你旋转 IoTs2 改变其姿态时，观察屏幕的数值并不方便。那我们就想到其他的显示方式，譬如我们用 IoTs2 的 RGB 像素灯珠发出的光颜色来指示加速度的三个分量。巧合的是，IoTs2 加速度传感器能给出三个分量值，而 RGB 像素灯珠的颜色也正好是三个分量 RGB。下面的示例程序中，我们将 IoTs2 的加速度传感器三个分量分别映射为像素灯珠的 RGB 三基色的三个分量。示例代码如下:

```
import time
from hiibot_iots2 import IoTs2
iots2 = IoTs2()
# map acceleration values (-10.24~10.24) into 0~255
def map(v):
    return  abs( int((v/10.24)*255.0) )

while True:
    x, y, z = iots2.Accele_ms2 # unit with ms^2
```

(continues on next page)

(continued from previous page)

```
10      iots2.pixels[0] = ( map(x), map(y), map(z) )
11      time.sleep(0.1)
```

很酷！这么短一点代码就能实现如此酷的效果。这个示例的程序结构已经被我们在前几节中反复使用过，我们定义来一个函数来处理数据映射：把加速度的某个分量值 (范围-10.24 ~ +10.24) 映射成 RGB 三基色某个分量 (范围 0~255)。下面是该示例程序的代码分析：

示例代码分析：

- 第 1 行，导入一个 Python 内建的模块“time”
- 第 2 行，从“/CIRCUITPY/lib/hiibot_iots2.py”模块中导入一个名叫“IoTs2”的类
- 第 3 行，将导入的“IoTs2”类实例化为一个实体对象，名叫“iots2”
- 第 4 行，注释
- 第 5 行，定义一个名叫“map”的函数，输入参数是变量 v
- 第 6 行，函数 map 的程序块，直接返回“abs( int((v/10.24)*255.0) )”
- 第 8 行，开始一个无穷循环
- 第 9 行 (无穷循环程序块的第 1 行)，将元组型 sensor.acceleration 加速度传感器的三分量分别赋给变量 x,y,z
- 第 10 行 (无穷循环程序块的第 2 行)，分别将 x,y,z 映射为 RGB 三基色分量，并用这个三基色设置 IoTs2 的 RGB 彩灯颜色
- 第 11 行 (无穷循环程序块的第 3 行)，执行 time 的 sleep 方法，参数为 0.1 秒

将本示例代码保存到 IoTs2 的/CIRCUITPY/code.py 文件中，当 IoTs2 运行示例代码期间，试着改变 IoTs2 的姿态，你发现 IoTs2 的 RGB 像素的颜色与 IoTs2 姿态之间什么关系？

正面朝上时，为什么是蓝色？根据本示例代码，以及加速度传感器三分量、姿态之间关系，请你说明这个原因。

---

**总结：**

- 地球重力场和方向
- 地球重力常数
- 加速度计
- 姿态感知和加速度
- 姿态估算和加速度动态变化
- 多行文本显示的数据结构
- 文本字体的缩放

- 本节中，你总计完成了 56 行代码的编写工作

---

---

---

**Important: IoTs2 类的加速度传感器属性和接口**

- Accele_Range (属性值，可读可写的，有效值为 {0, 1, 2, 3} 分别代表 2G、4G、8G 和 16G 的加速度范围 (G——地球重力加速度))
- Accele_DataRate (属性值，可读可写的，有效值为 0~9 的整数，分别代表掉电、1Hz、10Hz、25Hz、50HZ、100Hz、200Hz、400Hz、1.6KHz、1.334KHz 的加速度数据采样率
- Accele_ms2 (属性, 元组类型, 只读, 每个分量的有效值: -10.24~+10.24), IoTs2 的 Accele_ms2 属性采用 m/s^2 为加速度量纲, 加速度传感器的三个分量值
  - Accele_ms2[0]: x 方向分量
  - Accele_ms2[1]: y 方向分量
  - Accele_ms2[2]: z 方向分量
- Accele_Gs (属性, 元组类型, 只读, 每个分量的有效值: -1.0~+1.0), IoTs2 的 Accele_Gs 属性是相对加速度值，相对于地球标准重力场的加速度值, 加速度传感器的三个分量值
  - Accele_Gs[0]: x 方向分量
  - Accele_Gs[1]: y 方向分量
  - Accele_Gs[2]: z 方向分量
- angle_RollPitch (属性, 元组类型, 只读, 每个分量的有效值: -180~+180), IoTs2 的 angle_RollPitch 属性是翻滚角和俯仰角，angle_RollPitch 的两个分量值
  - angle_RollPitch[0]: 翻滚角 (roll)
  - angle_RollPitch[1]: 俯仰角 (pitch)
- Shake (函数, 输入参数: 阈值, 采样次数, 延迟时间, 返回值: 0/False 或 1/True), IoTs2 的加速度传感器的晃动检测, 返回 1/True: 有晃动，0/False: 无晃动

---

### 1.3.9 扫描周围的 WiFi 热点并连接到指定的 AP

这一节教程中，我们将初步体验 IoTs2 的 WiFi。WiFi 是一种广泛应用于固定的或移动范围较小的设备，为此类设备提供高速的高吞吐量的无线通讯接口。在手机、平板电脑、桌面电脑、个人数字终端等设备上 WiFi 是标配的功能单元，IoTs2 也具有此类设备完全相同的 WiFi 通讯接口。

近些年，IoT 设备的 NFC(近场通讯) 技术只能解决 2 厘米以内的无线电波通讯，IoT 的最后几米是蓝牙，那么 IoT 的最后几十米是 WiFi。据业内人士的专业预测，新兴的 WiFi6 技术将与 5G 技术将分庭抗争室外和室内的无线电波通讯技术应用领域，各自都会有巨大的市场潜力。近两年的 4G 资费急剧下降，室外的甚至部分

室内的无线电波通讯方案已经让我们可以放弃 WiFi，但是室内尤其非工业现场的住宅和办公区域仍是 WiFi 的天下，WiFi6 是非常有竞争力的方案。

目前，WiFi 是几十米到数百米距离内的无线电波通讯技术的最佳选择，虽然还有其他的选择，但综合考虑设备、流量和资费等成本，WiFi 的确是无可争议的无线电波通讯方案。那么 WiFi 的工作原理是啥样？从本节开始，我们通过一系列的教程帮助你逐步理解这一重要的 IoT 技术原理和实现方法。

本节先使用 IoTs2 的 WiFi 通道让你初步体验“scan 周围 WiFi 热点并连接到指定的热点”，任何 WiFi 设备，只要具备交互能力，scan 周围热点是其最重要的基础功能之一。示例代码如下：

```
import time
import wifi
from hiibot_iots2 import IoTs2
iots2 = IoTs2()
iots2.screen.rotation = 270  # rotate IoTs2 screen
iots2.pixels[0] = (255,0,0)  # show RED color
avaliableAP_list = []  # to save the avaliable APs
print('start to scan APs')
for network in wifi.radio.start_scanning_networks():
    time.sleep(0.05)
    print("\t'%s'\t\tRSSI: %d\tChannel: %d" % (network.ssid, network.rssi, network.
↪channel))
    avaliableAP_list.append( network.ssid ) # append the AP to the list
wifi.radio.stop_scanning_networks()  # stop scan
iots2.pixels[0] = (127,127,0) # show YELLOW color
print('AP scanning Done')
try:
    from secrets import secrets # import secrets.py module
except ImportError:
    print("WiFi secrets are kept in secrets.py, please add them there!")
    raise
theAP = None
for ap in avaliableAP_list:  # check avaliable APs
    if ap==secrets["ssid"]:
        theAP = ap
        break
if theAP==None:
    print("Not this AP, please to modify the secrects.py")
    iots2.pixels[0] = (255,0,0)  # show RED color
    while True:
        pass
elif not wifi.radio.ipv4_address:
    macaddr = 'My MAC Address: 0x'
    for addr in wifi.radio.mac_address:
        macaddr += '{:02X}'.format(addr)
```

(continues on next page)

(continued from previous page)

```
35      print( macaddr )
36      print("Connecting to WiFi AP (SSID: %s) ..." % secrets["ssid"])
37      print("Connecting to %s" % secrets["ssid"])
38      wifi.radio.connect(secrets["ssid"], secrets["password"])
39  print("Connected to %s!" % secrets["ssid"])
40  iots2.pixels[0] = (0,0,255)
41  print("My IP address is {}".format(wifi.radio.ipv4_address))
42  while True:
43      time.sleep(10)
```

本示例程序只实现 IoTs2 发现 (扫描) 周围的 WiFi 热点 (AP)，并按信号强度有高到低的顺序地输出到 LCD 屏幕或 Python 的串口控制台。请将本示例程序保存到 IoTs2 的/CIRCUITPY/code.py 文件中，IoTs2 将开启内部 WiFi 通道，并开始扫描周围 WiFi 热点，最后把 scan 到的热点按信号强度顺序地输出到屏幕并保存在一个列表中，然后与/CIRCUITPY/secrets.py 文件中的" ssid" 项的值对比，如果附近热点的列表中有与之匹配的 SSID，则检测 WiFi 是否已经连接到热点，如果未连接则将 WiFi 连接到该热点，并输出 IoTs2 的 IP 地址。

示例代码分析：

- 第 1 行，导入一个 Python 内建的模块"time"
- 第 2 行，导入一个 Python 内建的模块"wifi"
- 第 3 行，从"/CIRCUITPY/lib/hiibot_iots2.py"模块中导入一个名叫"IoTs2"的类，即 IoTs2 模块类
- 第 4 行，将导入的"IoTs2"类实例化为一个实体对象，名叫"iots2"
- 第 5 行，将 IoTs2 的 LCD 屏幕旋转为 90/270 度，即横屏显示效果
- 第 6 行，IoTs2 的 RGB 彩灯显示红色
- 第 7 行，定义一个空的列表，名称为 avaliableAP_list
- 第 8 行，向 LCD 屏幕或串口控制台输出字符串"start to scan APs"，提示开始扫描附近的 WiFi AP
- 第 9 ~ 12 行，用一个有限次的循环，逐个地将扫描到的附近热点的名称 (SSID)、信号强度、所用信道等信息打印到 LCD 屏幕或串口控制台上，并将所有 SSID 保存在列表 avaliableAP_list 中
- 第 13 行，停止 WiFi 的扫描操作
- 第 14 行，IoTs2 的 RGB 彩灯显示黄色
- 第 15 行，向 LCD 屏幕或串口控制台输出字符串"AP scanning Done"，提示扫描结束
- 第 16 ~ 20 行，尝试导入"/CIRCUITPY/secrets.py"文件，该文件中保存有 IoTs2 的联网配置信息
- 第 21 ~ 25 行，根据"secrets.py"文件的" ssid" 项的值，判断前面扫描到的周围热点中是否有该热点
- 第 26 ~ 30 行，如果"secrets.py"文件的" ssid" 项的值所给定的 SSID 与周围热点名称都不匹配，则让 IoTs2 的 RGB 彩灯亮红色，输出提示信息后让程序进入死循环
- 第 31 ~ 38 行，如果 WiFi 的 IP 地址无效则说明未连接到指定的 AP，首先打印 MAC 信息，然后尝试连接到指定的 AP。注意，这一连接过程是阻塞式的，除非出现错误，否则一直等到连接成功

- 第 39 行，提示已经连接到指定的 AP，并输出该 AP 的 SSID 名称
- 第 40 行，让 IoTs2 的 RGB 彩灯显示蓝色，提示已经连接成功
- 第 41 行，将本机的 IP 地址输出到 LCD 屏幕或串口控制台
- 第 42 行，一个无穷循环的程序块
- 第 43 行 (无穷循环程序块的第 1 行)，执行 time 的 sleep 方法，参数为 10 秒

---

---

**总结：**

- IoT 的无线电波通讯
- WiFi
- 扫描周围的可用热点
- 连接到一个指定的 WiFi AP
- MAC 地址
- IP 地址 (IPv4 的地址)
- 本节中，你总计完成了 43 行代码的编写工作

---

---

---

**Important: WiFi 类的属性和接口**

- radio (wifi 的子类，用于访问 IoTs2 的 WiFi 通讯接口)
- radio.enable (属性值，可读可写的，有效值为 False 和 True 分别为使能和禁止 IoTs2 的 WiFi 接口
- radio.hostname (属性值，只读的，固定为 “HiiBot_IoTs2”，当 IoTs2 连接到一个 AP 后，在 AP 的客户端列表中将会看到该模块的名称字符串)
- radio.mac_address (属性值，只读的，每个 IoTs2 的 MAC 地址都是固定的，这是一个 6 项的列表数据)
- radio.ipv4_address (属性值，只读的，当 IoTs2 与一个 AP 连接后，AP 会自动为 IoTs2 分配一个 IPv4 地址，形式为这是一个 6 项的列表数据。未连接到 AP 时，该属性值为 None)
- radio.ipv4_dns (属性值，只读的，当 IoTs2 与一个 AP 连接后自动获取 dns 的 IPv4 地址。未连接到 AP 时，该属性值为 None)
- radio.ipv4_gateway (属性值，只读的，当 IoTs2 与一个 AP 连接后自动获取 GateWay 的 IPv4 地址。未连接到 AP 时，该属性值为 None)
- radio.ipv4_subnet (属性值，只读的，当 IoTs2 与一个 AP 连接后自动获取 SubNet 的 IPv4 地址。未连接到 AP 时，该属性值为 None)
- radio.ping() (函数，输入参数：xx.xx.xx.xx 形式的 IPv4 地址), 该接口的用法示例见文末的示例程序
- radion.start_scanning_networks() (函数，无输入参数，用法见上面的示例)

- radio.stop_scanning_networks() (函数，无输入参数，无返回值，停止正在执行的 AP 扫描)
- radio.connect() (函数，两个字符串型输入参数：’ssid’ 和’password’，无返回值。这是一个阻塞式接口函数，除非出错将立即退出，否则将一直等待到连接到指定的 AP 后才会返回)
- radio.ap_info (wifi.Network 子类，用于访问 IoTs2 的 WiFi 接口的状态属性，使用 “wifi.radio.ap_info.” 或许以下的 IoTs2 WiFi 接口状态的属性值：
  - authmode: 只读的，返回’WPA_WPA2_PSK’等类型的安全认证模式的字符串
  - bssid: 只读的，返回 bytearray 型
  - ssid: 只读的，返回已经连接的 AP 名称，未连接到 AP 时该属性值未 None
  - rssi: 只读的，返回当前的信号强度
  - chennel: 只读的，返回当前连接 AP 所用的信号编号 (有效值范围是 0~11)
  - country: 只读的，返回当前连接的 AP 所属的国家，如 “CN” 表示中国

关于 wifi.radio.ping() 接口函数的用法，示例代码如下：

```
import wifi
import ipaddress
wifi.radio.connect('your_ap_name', 'your_ap_password')
dest_ip = ipaddress.ip_address('192.168.1.1')
wifi.radio.ping(dest_ip)
```

建议使用 REPL 模式执行上面的示例代码。

### 1.3.10 联网获取本地时间

前一个向导中我们已经认识了 IoTs2 的 WiFi，启动 WiFi 扫描周围 AP(WiFi 热点) 并连接到指定的 AP。本向导帮助我们如何将 IoTs2 的 WiFi 连接到周围的一个可用的 AP，如果这个 AP 与互联网是连通的，我们就可以使用网络时间服务校准本地的日期和时间。

所谓可用的 AP，IoTs2 必须能够扫描到，而且需要你知道这个 AP 的密码。网络时间服务是一种公益性的网络服务，我国已经建有十余个开放性的网络时间服务器。网络时间服务器使用网络时间服务协议 (NTP, Network Time Protocol) 向服务请求方提供标准的时间校准服务。NTP 的诞生完全是为网络设备提供时间同步服务，传统的计时器需要定期手工校准时间，譬如利用电台的半点或整点报时来手工校准，今天的大多数智能设备都能够连接到互联网，并使用 NTP 服务器自动校准时间。

本向导的目的是回答两个问题：1) 如何让 IoTs2 连接到互联网？2) 如何使用 NTP 服务器校准本地计时器？

### 让 IoTs2 接入互联网

电脑、Pad、手机等设备可以通过 WiFi 与无线路由器连接，并通过无线路由器连接到互联网，我们就可以使用搜索引擎找到自己需要的信息或者观看视频等。IoTs2 几乎与智能手机或平板有完全相同的联网和使用网络的方法，因为 IoTs2 也有一个内置的“无线网络设备/网卡”。

根据前一个向导的经验，如果 IoTs2 的 WiFi 能够扫描到周围的一些 AP，而且你已经知道其中某些 AP 的密码，我们下一步就是将这个 AP 的名称和密码告诉 IoTs2，并编程让 IoTs2 连接到这个 AP。如果这个 AP 与互联网是连通的，我们的 IoTs2 即可通过这个 AP 接入互联网。

如何将 AP 的名称（ssid) 和密码 (password) 告诉 IoTs2 呢? 我们有两种方法：

- 第 1 种方法是将这个 AP 的 ssid 和 password 两个字符串分别保存在“/CIRCUITPY/secrets.py”文件的对应位置，IoTs2 需要联网时自动去这个文件中读取这些信息
- 第 2 种方法是将这个 AP 的 ssid 和 password 两个字符串直接用 Python 程序接口传给 IoTs2 的 wifi.radio 类

虽然两种方法是等价的，建议使用第 1 种方法，即便是你分享自己的代码给其他人时，你的 AP 信息不会泄露给别人。secrets.py 文件的格式如下：

```
secrets = {
  "ssid": "your_ap_name",
  "password": "your_ap_password",
  "timezone": "Asia/Shanghai",  # Check http://worldtimeapi.org/timezones
  "broker":"www.hiibotiot.com",
  "hiibotiot_user":"anyone",
  "hiibotiot_password":"12345678"
}
```

这是一个 JSON 格式化的文本型“key:value”信息对 (即字典)，也可以用 Python 字典型数据结构来访问。每一个“:”前的字符串是“key”，“:”后的字符串是这个“key”对应的“value”。

下面我们使用第 2 种方法设计一个让 BlueFi 连接到互联网的程序示例。程序代码如下：

```
import wifi
from hiibot_iots2 import IoTs2
iots2 = IoTs2()
iots2.screen.rotation = 270
wifi.radio.enabled = True
while not wifi.radio.ipv4_address:
    # ConnectionError: No network with that ssid?
    wifi.radio.connect('your_ap_name', 'your_ap_password')
print("Connected to", str(wifi.radio.ap_info.ssid, "utf-8"), "\tRSSI: {}".format(wifi.
↪radio.ap_info.rssi) )
print("My IP address is {}".format(wifi.radio.ipv4_address))
wifi.radio.enabled = False
```

(continues on next page)

(continued from previous page)

```
12 while True:
13       pass
```

根据前一节的内容，我们不难理解上面的程序代码，如果不修改第 8 行代码中的字符串‘your_ap_name’和‘your_ap_password’，直接将程序保存到 IoTs2 的/CIRCUITPY/code.py 文件中，我们将会看到 “ConnectionError: No network with that ssid” 错误提示。你的 AP 名称和连接密码正好是‘your_ap_name’和‘your_ap_password’的概率极低，将你的 IoTs2 附近可用的 AP 和密码分别填入这两个字符串中，我们将会看到正确的结果。

第 9~10 行将已经连接到的 AP 名称、当前的信号强度和 IP 地址输出到 LCD 屏幕和串口控制台上。第 11 行将 WiFi 关闭，关闭 WiFi 的目的是可以降低 IoTs2 的功耗，本示例仅仅是为了演示连接 WiFi 热点而已。

与第 1 种设置 WiFi 名称和密码相比，第 2 种方法使用本示例第 8 行的 Python 接口程序将 AP 名称和密码作为参数直接传递给该接口。如果你分享这个示例代码给别人时，你的 AP 名称和密码也同时泄露出去了。

下面示例采用第 1 种方法设置 AP 的名称和密码：

```
1  import wifi
2  # Get wifi details and more from a /CIRCUITPY/secrets.py file
3  try:
4      from secrets import secrets
5  except ImportError:
6      print("WiFi secrets are kept in secrets.py, please add them there!")
7      raise
8  from hiibot_iots2 import IoTs2
9  iots2 = IoTs2()
10 iots2.screen.rotation = 270
11 wifi.radio.enabled = True
12 while not wifi.radio.ipv4_address:
13     wifi.radio.connect(secrets["ssid"], secrets["password"])
14 print("Connected to", str(wifi.radio.ap_info.ssid, "utf-8"), "\tRSSI: {}".format(wifi.
   ↪radio.ap_info.rssi) )
15 print("My IP address is {}".format(wifi.radio.ipv4_address))
16 wifi.radio.enabled = False
17 while True:
18       pass
```

与前一个示例相比，该示例的第 13 行代码，即用于连接 AP 的 Python 接口程序，我们并没有传递参数，执行该程序语句时会自动到 “/CIRCUITPY/secrets.py” 文件中读取 AP 的名称和密码，并自动连接该 AP。

如果你分享这个程序代码时，记得提醒代码使用者需要自己修改 “/CIRCUITPY/secrets.py” 文件中的 “ssid” 和 “password” 两个 key 的 value。

### 用互联网同步本地时间

当我们搞清楚如何让 IoTs2 的 WiFi 连接到互联网之后，我们就可以使用 NTP 服务校准/同步本地的日期和时间。什么是 NTP? 请自行使用搜索引擎查阅相关资料，NTP 是 TCP/IP 协议栈中的一种应用层协议。

下面我们使用国际时间 NTP 服务器 (域名：http://worldtimeapi.org/) 来校准本地时间，这个服务器提供多种 NTP 服务接口，本示例使用“按照本地的 IP 地址返回当地的日期和时间信息”，这个 NTP 服务器的服务接口：

- http://worldtimeapi.org/api/timezone 返回所有时区的当前日期和时间 (如果你需要设计一个五星级酒店大堂使用的数字计时器)
- http://worldtimeapi.org/api/timezone/:Asia/Shanghai 返回上海时区 (中国时间) 的当前日期和时间
- http://worldtimeapi.org/api/ip/:ipv4_addr 返回指定 IP 所在地区的当前日期和时间

本示例程序的代码如下：

```
import time
import rtc
import wifi
import ipaddress
import ssl
import wifi
import socketpool
import adafruit_requests
from hiibot_iots2 import IoTs2
iots2 = IoTs2()
iots2.screen.rotation = 180
iots2.pixels[0] = (255,0,0)
the_rtc = rtc.RTC()
response = None
weekDayAbbr = ['Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun']
try:
    from secrets import secrets
except ImportError:
    print("WiFi secrets are kept in secrets.py, please add them there!")
    raise
print("Connecting to %s" % secrets["ssid"])
wifi.radio.connect(secrets["ssid"], secrets["password"])
print("Connected to %s!" % secrets["ssid"])
iots2.pixels[0] = (0,0,255)
print("My IP address is {}".format(wifi.radio.ipv4_address))
pool = socketpool.SocketPool(wifi.radio)
requests = adafruit_requests.Session(pool, ssl.create_default_context())
response = requests.get("http://worldtimeapi.org/api/timezone/Asia/Shanghai",␣
↪timeout=60.0)
```

(continues on next page)

(continued from previous page)

```
iots2.pixels[0] = (0,255,255)
rgb_gright = 0.1
######### fade IoTs2 RGB pixels #########
def fadeRGB() :
    global rgb_gright
    rgb_gright += 0.005
    if rgb_gright>0.1:
        rgb_gright = 0.0
    iots2.pixels.brightness = rgb_gright
    iots2.pixels.show()
    time.sleep(0.1)
######### Parse Date&Time from JSON #########
if response.status_code == 200:
    print("We got a NTP server")
    iots2.pixels[0] = (0,255,0)
    json = response.json()
    print(json)  # print all message
    current_time = json["datetime"]
    the_date, the_time = current_time.split("T")
    print(the_date)
    year, month, mday = [int(x) for x in the_date.split("-")]
    the_time = the_time.split(".")[0]
    print(the_time)
    hours, minutes, seconds = [int(x) for x in the_time.split(":")]
    # We can also fill in these extra nice things
    year_day = json["day_of_year"]
    week_day = json["day_of_week"]
    # Daylight Saving Time (夏令时)?
    is_dst = json["dst"]
    now = time.struct_time(
        (year, month, mday, hours, minutes, seconds+1, week_day, year_day, is_dst) )
    the_rtc.datetime = now
    while True:
        print( "  {}-{}-{}".format(
            the_rtc.datetime.tm_year,
            the_rtc.datetime.tm_mon,
            the_rtc.datetime.tm_mday, )
            )
        print( "  " + weekDayAbbr[the_rtc.datetime.tm_wday] )
        print( "  {}:{}:{}".format(
            the_rtc.datetime.tm_hour,
            the_rtc.datetime.tm_min,
            the_rtc.datetime.tm_sec, )
```

(continues on next page)

(continued from previous page)

```
72              )
73          ipv4 = ipaddress.ip_address('182.61.200.6')
74          print('  ping time:', wifi.radio.ping(ipv4))
75          for _ in range(520):
76              fadeRGB()
77  else:
78      print("Getting time failed")
```

你首先将本示例代码保存到 IoTs2 的/CIRCUITPY/code.py 文件中，务必记得修改 secrets.py 文件中的 ssid 和 password 两个选项的值，当 IoTs2 执行本示例程序时，如果与 AP 成功连接后调用 “http://worldtimeapi.org/” 的时间服务接口，即第 28 行代码，从 NTP 服务器请求本地时间，并将请求结果保存在 response 变量中。

如果你用 web 浏览器打开 http://worldtimeapi.org/页面时，将会看到该服务的返回结果说明。根据说明我们可以知道，reponse 是一个 JSON 格式化的文本字符串信息，本示例程序的第 42 ~ 55 行通过解析这个 JSON 格式化的信息流确定本地的日期和时间，并分别保存在 year, month, mday, hours, minutes, seconds,year_day 和 week_day 等变量中。

在本示例程序的最后一个程序块 (无穷循环程序块) 中，读取本地 RTC 的日期和时间，并格式化后输出到 IoTs2 的 LCD 屏幕和串口控制台上，你会发现 “秒” 数据的不断变化。同时，为了更好滴理解 “wifi.radio.ping()” 接口函数的用法，无穷循环中也不断地你用该接口 ping 百度服务器，即’ 182.61.200.6’ 地址，并输出 ping 操作的耗时。

你或许会问，这样的方法同步本地时间，是否存在误差？当然存在，受你的无线网络状况、执行 NTP 服务的 CPU 速度等因素影响，这种方法校准的本地时间与国际时间相差几十到几百毫秒。

如果这个误差太大，不能满足你的应用，你觉得如何减少这一误差呢？

---

**总结：**

- 将 AP 的名称和密码告知 IoTs2
- 让 IoTs2 连接到互联网
- 从互联网的 NTP 服务器获取 IoTs2 本地的当前日期和时间

---

### 1.3.11 物联网 (IoT) 和 MQTT

物联网 (IoT) 已经彻底改变我们今天的生活方式。如果没有 IoT 技术做支撑，共享单车、共享汽车等便捷的生活方式几乎不可能实现。当然，IoT 的应用远不止这些。

QQ、wechat(微信) 是人-人互联的即时通软件，MQTT(消息队列遥测传输，Message Queuing Telemetry Transport)) 是实现物-物互联的即时通协议。MQTT 已经成为全球公认的 IoT 标准。

虽然 QQ 和 MQTT 都诞生于上世纪末，但 QQ 很早就被全世界华人社区所接受并逐步淘汰传统的电话通讯，QQ 即时通软件的成功成就了腾讯。然而，MQTT 直到近 10 年才被广泛接受。虽然他们的技术方面几乎相

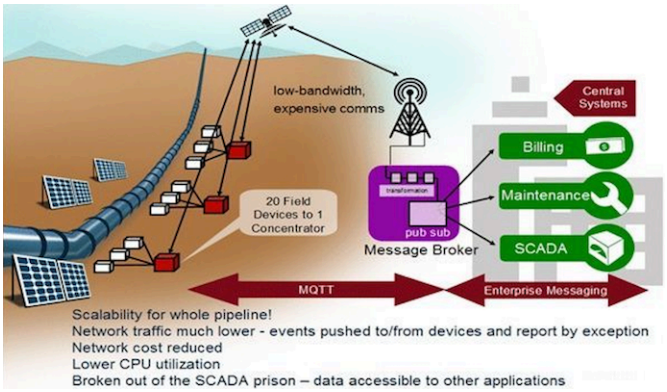
近，但两种即时通所面向的业务领域却完全不同，MQTT 是随着 IoT 技术的发展日趋成熟才逐步走进我们的视野。

本向导中我们将认识 MQTT，并掌握 MQTT 的简单应用。

### 关于 MQTT

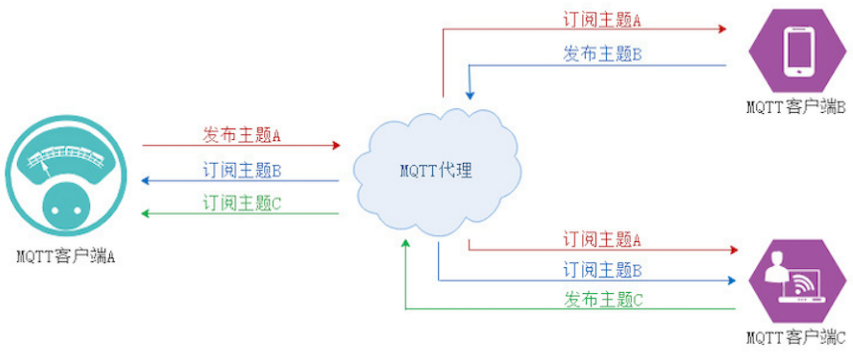
MQTT 是构建在 TCP/IP 协议栈之上的基于客户端-服务器的消息发布/订阅 (publish/subscribe) 传输协议，属于轻量级的即时通讯协议，可以以极少的代码和有限的网络带宽的条件下为连接远程设备提供实时可靠的消息服务，在物联网、小型设备、移动应用等方面已有广泛的应用。按照七层 OSI 模型，MQTT 与 HTTP 等一样属于 TCP/IP 的应用层协议。

这里的几个关键词需要稍作解释：1) 构建在 TCP/IP 协议栈之上，根据 MQTT 所使用的 TCP/IP 协议栈的类型 (如 TCP, TLS(具备安全层传输控制的 TCP), WS(web-socket) 和 WSS(具备安全加密传输的 WS))，又将 MQTT 分为 4 种；2) 基于客户端-服务器，说明 MQTT 系统的设备分为两种角色：客户端和服务器，一般来说消息的发布者和订阅者都是客户端，服务器仅仅负责消息的路由和分发；3) 消息发布/订阅，这是 MQTT 协议的基本工作模式，与传统的网络通讯相比，这种消息发布/订阅模式可以极大地降低网络带宽的需求，传统的消息发布和接收是一对一或一对多 (广播)，一对一必定会增加网络带宽需求，而 MQTT 的消息发布者并不关心消息发个谁，只有订阅该消息的订阅者才会自动收到 MQTT 服务器转发的消息。



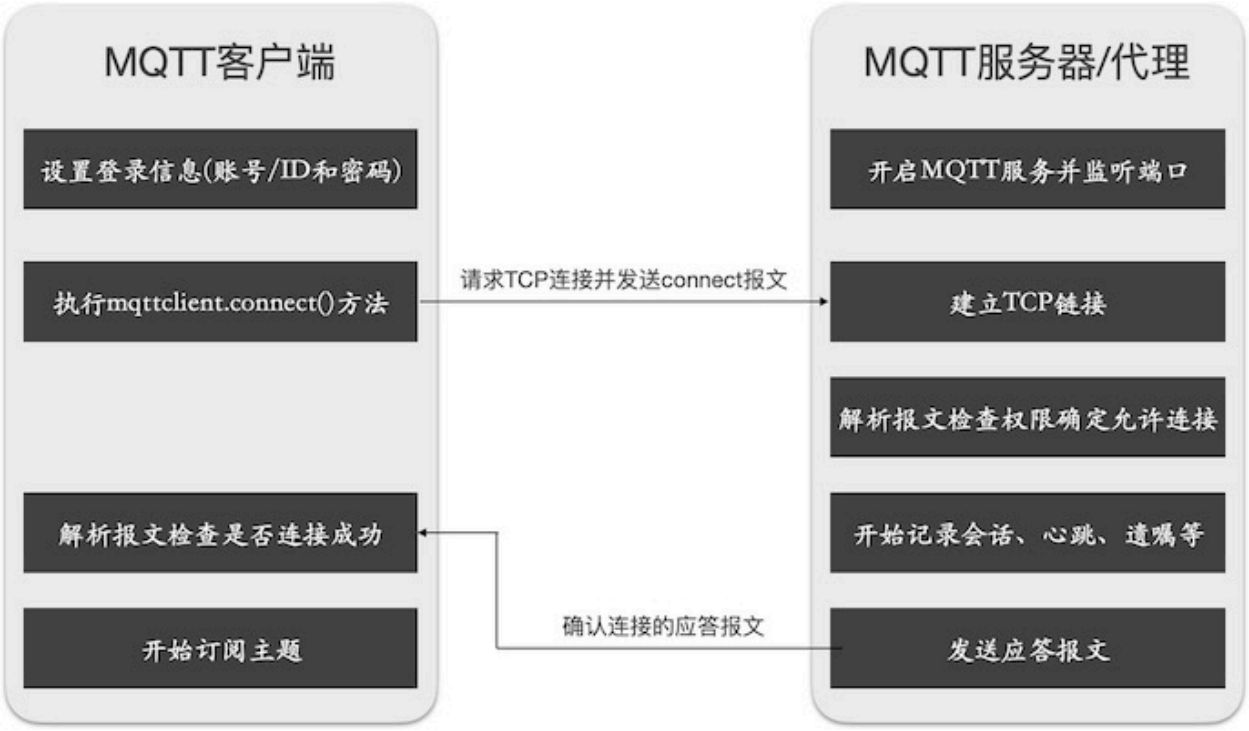

据考证，MQTT 协议是上世纪末 IBM 在帮助石油和天然气公司客户设计有效的数据传输协议时首次提出，当时，为了实现数千英里长的石油和天然气管道的无人值守监控，采取的设计方案是将管道上的传感器数据通过卫星通信传输到监控中心。如上图所示。今天我们所看到的 MQTT 协议的特征恰好满足当年石油和天然气公司向 IBM 提出的需求。

MQTT 的消息发布者/订阅者之间关系示意，如下图所示：

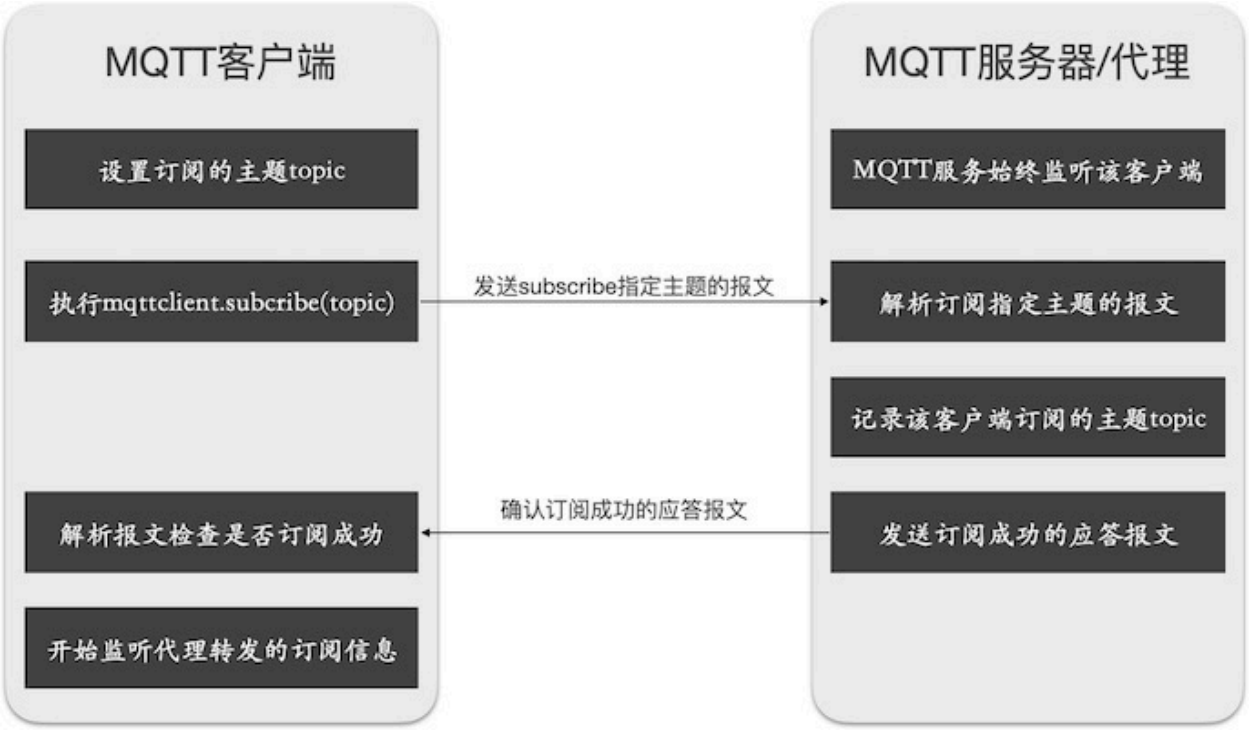



### MQTT 的工作流程

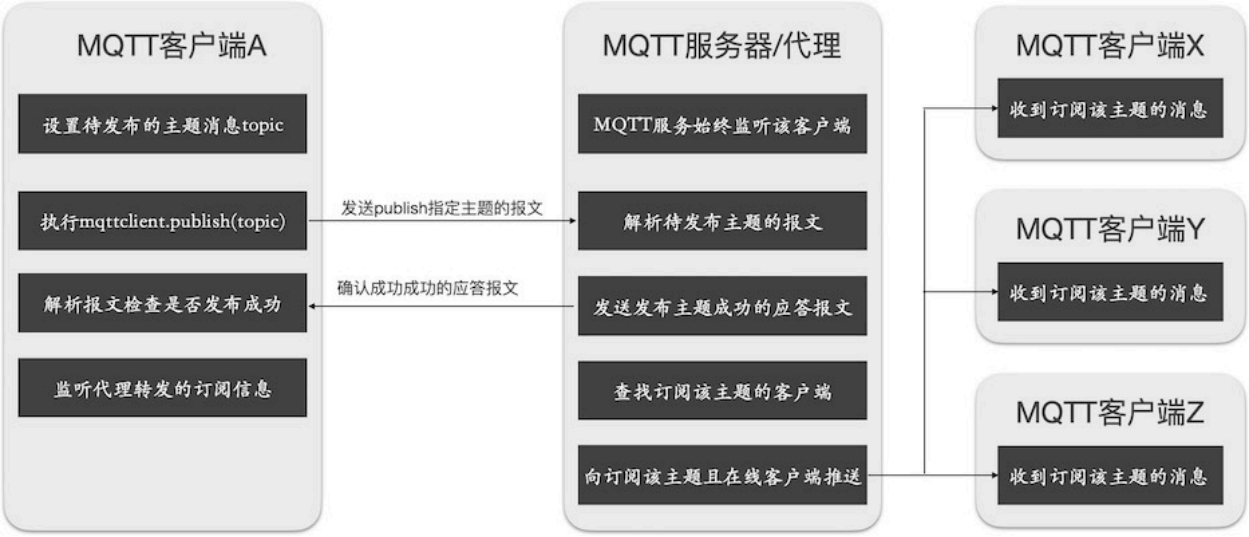
下面我们用三张图来简要说明 MQTT 的连接服务器、消息订阅、消息发布和推送的基本流程，先了解这些流程对于后面如何使用 MQTT 协议非常有益。




(某个客户端使用 TCP 协议连接到 MQTT 服务器/消息代理的流程)




(某个客户端订阅指定的主题 topic 消息的流程)




(客户端 A 发布一个主题消息并由 MQTT 服务器/代理推送给该消息的订阅者的流程)

### 将 IoTs2 连接到 MQTT 服务器

当我们初步了解 MQTT 协议和工作流程之后，我们开着手让 IoTs2 连接到 MQTT 服务器，为了简化问题，我们首先是使用匿名方式登录 MQTT 服务器免去注册获取 ID 和密码的过程。

我们用一个示例程序来掌握如何让 IoTs2 连接到 MQTT 服务器并订阅和发布消息。具体的代码如下：

```
1  import time
2  from hiibot_iots2 import IoTs2
3  from hiibot_iots2_mqtt import MQTTClient
4  iots2 = IoTs2()
5  iots2.pixels[0] = (255, 63, 0)
6  iots2.screen.rotation = 180
7  mqttClient = MQTTClient()
8  msgTopic1 = "/test/topic1"
9  msgTopic2 = "/test/topic2"
10 msgTopic3 = "/test/topic3"
11 def testTopic1(message):
12     print('New message (topic="{}", message="{}")'.format(msgTopic1, message))
13     mqttClient.publishMessage(msgTopic2, message)
14 def testTopic2(message):
15     print('New message (topic="{}", message="{}")'.format(msgTopic2, message))
16     mqttClient.publishMessage(msgTopic3, message)
17 mqttClient.subscribeTopic(msgTopic1, testTopic1)
18 mqttClient.subscribeTopic(msgTopic2, testTopic2)
19 mqttClient.connect()
20 while True:
21     mqttClient.loop()
22     time.sleep(0.005)
```

根据前两个向导，IoTs2 的 WiFi 必须首先连接到一个指定的 AP，只要该 AP 与互联网是连通的，那么 IoTs2 就可以与 MQTT 服务器 (消息转发代理) 连接，即可订阅、发布特定主题的消息。虽然上面的代码中并没有 WiFi 及其连接相关的代码，我们使用“/CIRCUITPY/lib/hiibot_iots2_mqtt.py”模块中的 MQTTClient 类，在该类中会根据联网的需要适时地让 IoTs2 的 WiFi 连接到 AP，并连接到 MQTT 代理，这些操作都在第 7 行代码中完成，

示例程序包含有两个函数 cb_testTopic1 和 cb_testTopic2。你会不会觉得奇怪？这两个函数并没有被其他程序调用。这两个函数属于“发生特定事件后响应该事件的回调函数”，你可以把他们想象成 Scratch 中的事件。示例程序的第 25 和 26 行分别从 MQTT 服务器订阅了两个主题消息，并指定 cb_testTopic1 函数作为收到“/test/topic1”主题消息的事件响应，指定 cb_testTopic2 函数作为收到“/test/topic2”主题消息的事件响应。

该示例程序的最关键的程序语句是第 24 行和第 27 行。第 24 行是实例化 MQTTClient 类 (MQTT 的 client 类)，传入的网络参数包括：wifi，即连接 MQTT 服务器的网络；sever，即 MQTT 服务器的网址。mqttClient 是 MQTTClient 类的实例化变量。执行第 27 行语句才是真正连接到指定的 MQTT 服务器/代理。

在最后的无穷循环程序块中，调用 MQTTClient 类的 loop() 方法，与 MQTT 服务器持续不断地联系 (发送心

跳、接收订阅消息、侦测并更新网络连接等)。

你把本示例代码保存到 BlueFi 的/CIRCUITPY/code.py 文件中，根据 BlueFi 屏幕或串口控制台提示的信息，你可以确定其连网状态、是否与 MQTT 服务器已经成功连接等。

### BlueFi 和电脑互推消息

如果你只有一个 BlueFi，如何体验 MQTT 的消息发布/订阅机制？可以借助于电脑端的 MQTT 客户端应用程序，这种客户端应用程序非常多，而且都是免费使用的。推荐你使用“MQTTBox 软件”，点击此处 打开 MQTTBox 网站并下载 MQTTBox 软件 该软件支持 Linux、maxOS 和 Windows 三种平台，选择适合自己系统的软件点击下载并安装 (如果安装过程需要向导，请参考该网页的相关文档)，然后你就可以使用这个 MQTT 客户端软件发布或订阅 MQTT 的主题消息。

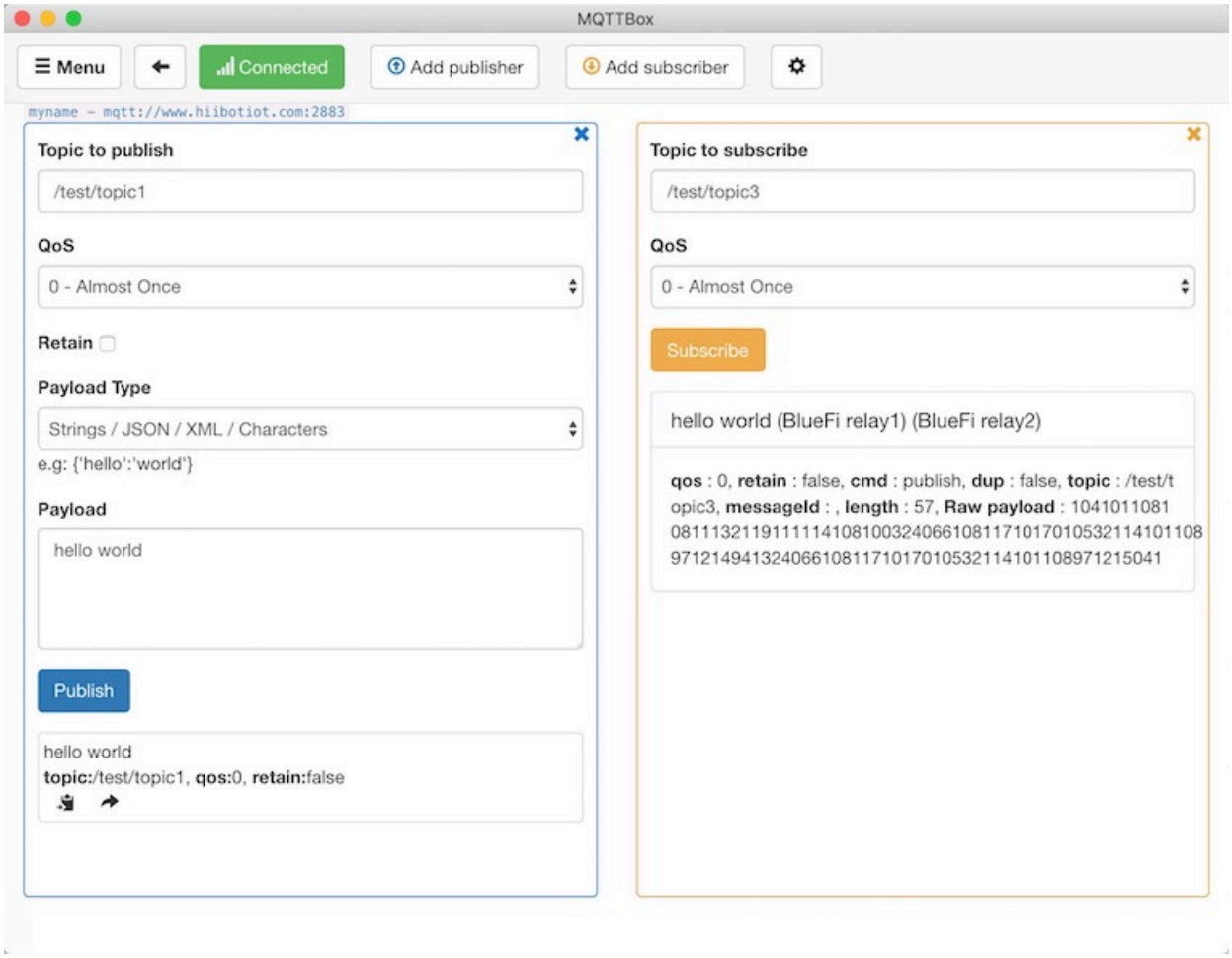
下图中演示如何使用 MQTTBox 软件创建新的 MQTT 客户端、订阅指定主题的消息、发布特定主题的消息。

(此动画暂缺)

其中的关键步骤如下：

- 点击“Creat MQTT Client”按钮，创建一个 MQTT 客户端
- 在弹出的窗口中填写 MQTT 客户端的主要参数选项值，包括“MQTT Client Name”(随意输入都可以)、“Host”(www.hiibotiot.com:2883)，并展开“Protocol”选项选择“mtqq/tcp”，最后点击“save”按钮
- 当 MQTT 客户端的窗口上方的出现绿色“Connect”按钮后，表明你创建的 MQTT 客户端已经与服务器连接上
- 在“Topic to subscribe”下方的第一个输入框中输入订阅的主题“/test/topic3”
- 在“Topic to public”下方的第一个输入框中输入待发布的主题“/test/topic1”，并在“Payload”下方输入框中输入消息内容 (随意输入)

然后点击“Public”按钮，你看到下图的消息了吗？




现在可以确认你的电脑和 BlueFi 通过 MQTT 服务器 (www.hiibotiot.com:2883) 相互订阅消息，当我们通过电脑发布一个主题为“/test/topic1”消息为“hello world”之后，根据本示例的程序代码，BlueFi 已经订阅了该主题消息，当 MQTT 服务器将电脑发布的这个消息推送给 BlueFi 之后，在 cb_testTopic1 回调函数中将这条消息打印到串口控制台和 BlueFi 的 LCD 屏幕上，然后将此消息内容尾部添加“(BlueFi relay1)”并以“/test/topic2”作为主题将该消息发布出去。然后会发生什么？因为 BlueFi 已订阅“/test/topic2”主题消息，这个主题消息虽然是 BlueFi 发布的，自己又订阅该主题消息，这个消息会被 MQTT 服务器再推送给 BlueFi，在 cb_testTopic2 回调函数中将这条消息打印到串口控制台和 BlueFi 的 LCD 屏幕上，然后将此消息内容尾部添加“(BlueFi relay2)”并以“/test/topic3”作为主题将该消息发布出去。你电脑端的 MQTTBox 软件创建的 MQTT 客户端已订阅“/test/topic3”主题消息，所以你在电脑上看到“hello world (BlueFi relay1) (BlueFi relay2)”消息，应该就很容易明白了。

通过这个示例，我们初步掌握 MQTT 的消息发布/订阅机制，并初步了解如何使用电脑搭建 MQTT 客户端，以及如何用 BlueFi 实现 MQTT 客户端，通过订阅/发布消息，电脑和 BlueFi 之间可以相互发送 IoT 信息。

假设 BlueFi 是 MQTT 客户端，如果麦克风侦测到很大的声音，让 BlueFi 自动发布一个主题为“/security/home”消息为“Someone broke into”，在手机或电脑上执行 MQTT 客户端软件，并确保已经连接到 MQTT 服务器，并订阅“/security/home”主题消息，当你手机或电脑端看到该消息时，这代表着某种特殊意义。看到这里，你是否觉得用 BlueFi 设计一个家庭安全警报系统很容易？

**IoT 和 MQTT**

- MQTT 是一种应用层协议，实现物-物互联的即时通讯协议
- MQTT 采用客户端和服务器架构，客户端发布/订阅指定主题消息，服务器管理消息并向订阅者推送新发布的主题消息
- BlueFi 实现 MQTT 客户端，必须先让 BlueFi 与互联网连接，然后与 MQTT 服务器连接
- BlueFi 和电脑之间能够通过 MQTT 服务器和消息的发布/订阅机制相互推送消息

## 1.4 进阶教程

从这里开始你将进入高级阶段，使用 IoTs2 和 Python 编程语言实现一些较为复杂的 idea，并深入掌握 Python 的数值计算、列表、元组、字典、迭代器和生成器等概念，以及计算机算法和物联传感方面的知识。

学习内容如下

## 1.5 项目级应用教程

这里向导或示例都属于项目级别，实现功能的程序代码比较长，或者需要其他一些辅材才能实现的应用。

学习内容如下

## 1.6 基本 I/O 功能拓展

学习内容如下

## 1.7 高级 I/O 功能拓展

学习内容如下

## 1.8 项目级应用教程

这里向导或示例都属于项目级别，实现功能的程序代码比较长，或者需要其他一些辅材才能实现的应用。

---

学习内容如下

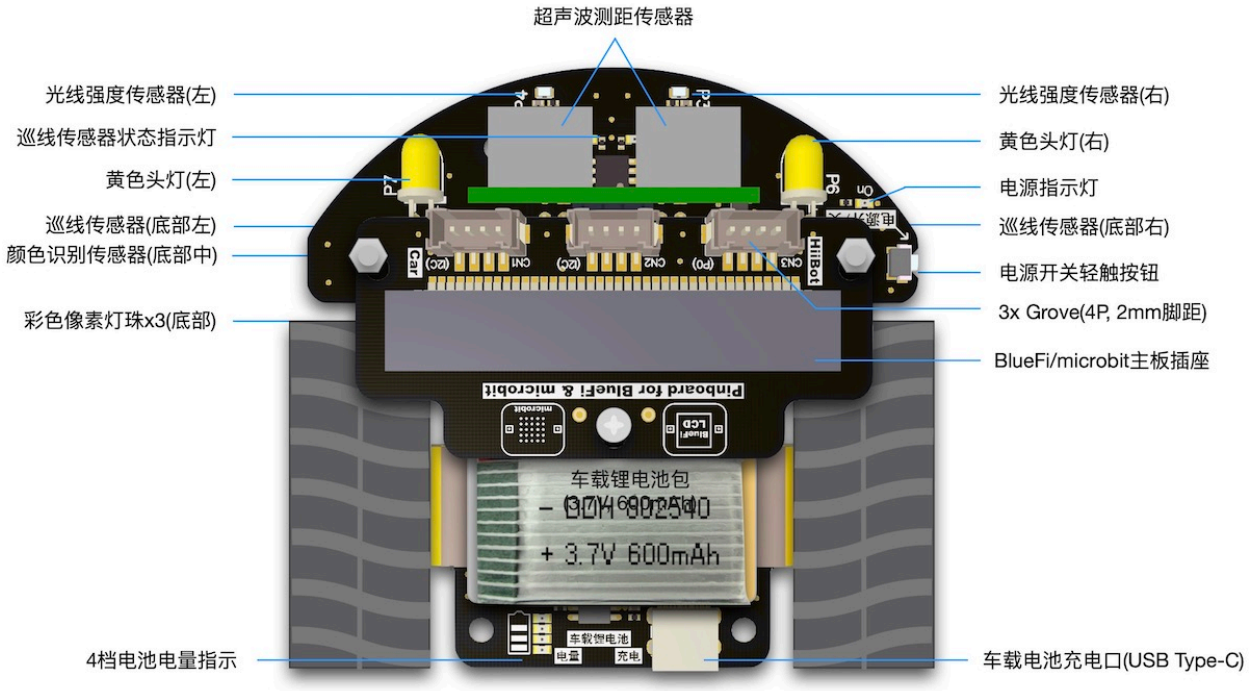
### 1.8.1 智能小车——RunGo

跑酷小车 (RunGo) 由 HiiBot 推出的智能小车底盘，包含以下资源：

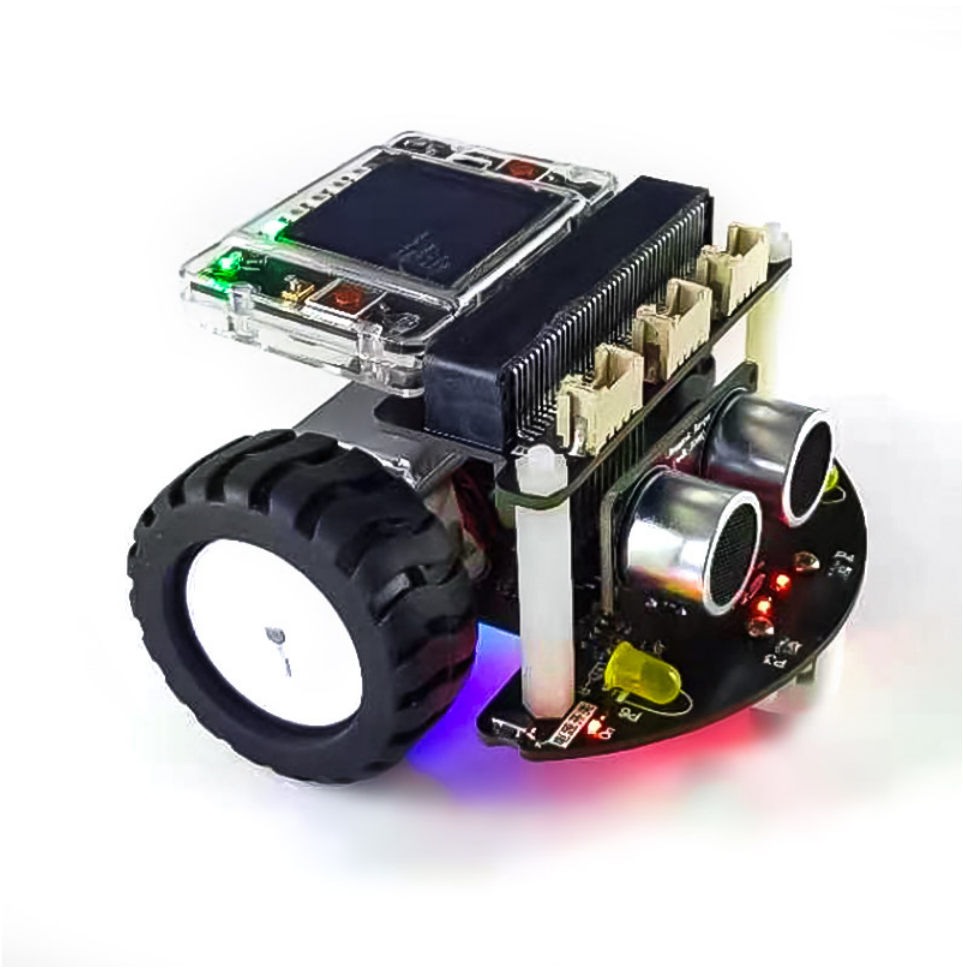
- 3x 轻触按钮 (1 个是 IoTs2 模块自带的按钮使用 IO21 引脚)，A 按钮使用 IoTs2 的 IO43 引脚，B 按钮使用 IoTs2 的 IO44 引脚
- 3x 底盘像素彩灯 (位于小车底盘的底部/面向地面, pixels)，使用 IoTs2 的 IO0 引脚
- 1x 颜色识别传感器 (位于小车底盘的底部，识别地面上色块的颜色, groundColor)，使用 IoTs2 的 IO7(A4) 引脚
- 2x 光线强度传感器 (位于小车前部)，使用 IoTs2 的 IO8(小车左前部光强度, leftLightSensor) 和 IO9(小车右前部光强度, rightLightSensor) 引脚
- 2x 黄色头灯 (位于小车前部)，使用 IoTs2 的 IO17(小车右前灯, rightHeadLED) 和 IO18(小车左前灯, leftHeadLED) 引脚
- 1x 超声波避障传感器 (与前方障碍物之间的距离, distance)，使用 IoTs2 的 IO10(Echo) 和 IO11(Trig) 引脚
- 2x 光电反射型循迹传感器，使用 IoTs2 的 IO3(leftTracker) 和 IO4(rightTracker) 引脚
- 2x 直流有刷电机 (N20 型)，使用 IoTs2 的 IO12(IN1) 和 IO40(IN2)(左马达)、IO41(IN1) 和 IO42(IN2)(右马达) 引脚
- 1x 车载锂电池包 (3.7V,600mAh) 及其充电和电量指示单元，使用 USB Type C 接口实现快速充电 (20~30 分钟即可充满)，满电后可连续使用 1 小时以上
- 1x 高效的车载电源管理单元，小车底盘右前方的轻触按钮可开启/关闭小车电源，

---

RunGo 小车底盘的传感器和分布 (示意图)，如下图所示：




IoTs2+RunGo 组成智能小车 (实物图)，如下图所示。



### 使用前准备工作：添加 RunGo 库模块

使用 RunGo 小车前，请点击“下载按钮”下载 RunGo 的 Python 库文件 `hiibot_iots2_rungo.py` 到本地，然后将该文件保存到 IoTs2 的”/CIRCUITPY/lib/”文件夹中。

注意，请使用 USB 数据线将 IotS2 与你的电脑连接在一起，电脑的资源管理器中出现“CIRCUITPY”磁盘，然后将 RunGo 的 Python 库文件复制到”/CIRCUITPY/lib/”文件夹即可。注意，部分 IoTs2 的 lib 文件夹中已有 hiibot_iots2_rungo.py 库文件的，请忽略此步骤。检查方法：使用 USB 数据线将 IoTs2 与电脑连接，出现 CIRCUITPY 磁盘后，展开”/CIRCUITPY/lib/”文件夹，检查是否已有“hiibot_iots2_rungo.py”或“hiibot_iots2_rungo.mpy”，其中“py”格式是 Python 脚本源码库，“mpy”格式是压缩的 Python 脚本源码库，前者是文本格式具有可读性但文件较大，后者是二进制格式无可读性但文件较小。

**Important: RunGo+IoTs2 模块的 USB 接口和操作顺序**

- RunGo 底盘的 USB Type-C 的充电接口仅仅用于对车载锂电池充电，不能用于其他功能。**充电用的 USB 插座位于 RunGo 底盘的尾部**

- IoTs2 模块使用 USB Type-C 供电和通讯，下载程序或更新固件必须使用这个 USB 端口。**下载程序和通讯用的 USB 插座位于 IoTs2 模块上，在小车顶部**
- 正常开机顺序：1) 按下 RunGo 右前方的电源开关即可开机，再次按下则关机；2) 任何时候必须首先开启 RunGo 的电源才能进行其他操作
- 下载程序的顺序：1) 打开 RunGo 电源 (见前一步)；2) 等待 IoTs2 启动完毕 (LCD 屏幕上将会出现提示文字)；3) 使用 USB 数据线将 IoTs2 模块与电脑连接 (不必取下模块)；4) 电脑的资源管理器中将出现一个名叫“CIRCUITPY”的可移动磁盘，然后可以通过复制-粘贴等操作下载程序
- 关机顺序：1) 首先将连接电脑和 IoTs2 模块的 USB 数据线拔掉；2) 按下 RunGo 电源开关即可。注意，当我们使用 USB 数据线将 IoTs2 模块与电脑连接时，按下 RunGo 电源开关将不能关闭电源

---

### 让 RunGo 动起来

当你拿到 RunGo 和 IoTs2 之后，并做好前述的准备工作做，我们首先让 RunGo 小车动起来。实现的效果：小车前进一段距离；然后开启右转灯并开始右转，然后关闭右转灯并停止右转；如此循环。

示例程序代码如下：

```
import time
from hiibot_iots2 import IoTs2  # IoTs2
from hiibot_iots2_rungo import RunGo
iots2 = IoTs2()
iots2.screen.rotation = 180
print('Run Go!')
car = RunGo()
carspeed = 100
go = False
cnt = 0
fsm = 0
print('Button-A -> run')
print('Button-B -> stop')
while True:
    car.pixelsRotation()
    car.buttons_update()
    if car.button_A_wasPressed:
        go = True
        print('running')
    if car.button_B_wasPressed:
        go = False
        cnt = 0
        fsm = 0
```

(continues on next page)

(continued from previous page)

```
        car.rightHeadLED = 0    # turn off right head lamp
        car.stop()
        print('stoping')
    time.sleep(0.01) # 10ms
    if go:
        if fsm==0:
            car.motor(carspeed, carspeed)
            cnt += 1
            if cnt>50:
                cnt = 0
                car.stop()
                fsm = 1
        elif fsm==1:
            car.rightHeadLED = 1    # turn on right head lamp
            car.motor(carspeed//2, -carspeed//2)
            cnt += 1
            if cnt>65:
                car.rightHeadLED = 0    # turn off right head lamp
                car.stop()
                cnt = 0
                fsm = 0
    else:
        pass
```

请首先打开 RunGo 电源，当 IoTs2 屏幕上有显示内容出现后，使用 USB 数据线将 IoTs2 模块与电脑连接，然后我们电脑资源管理器中将出现一个名叫“CIRCUITPY”磁盘，将上述示例代码保存到 IoTs2 的/CIRCUITPY/code.py 文件，请观察 RunGo 小车的动作是否达到我们的预期效果。

请注意：下载程序时务必使用 IoTs2 模块的 USB 插座，该插座位于 RunGo+IoTs2 的顶部!

上述示例程序虽然看起来很长，但非常容易理解。前三行语句是导入 Python 模块；第 4 行程序是将 IoTs2 类实例化为“iots2”，第 5 行则使用实体对象“iots2”设置其属性将 IoTs2 模块的 LCD 屏幕旋转为 180 度，即竖屏显示；第 7 行程序语句是将 RunGo 类实例化为“car”，后面的程序中将使用实体对象“car”的属性或接口来控制 RunGo 小车运动。

在无穷循环程序块中，我们使用“car.pixelsRotation()”函数控制 RunGo 底部的 3 颗彩灯形成旋转光效；并调用“car.buttons_update()”接口更新 RunGo 的 A 和 B 按钮的状态；当 A 按钮按下时则启动 RunGo 进入运动状态，按下 B 按钮则停止 RunGo 运动；当 RunGo 处于运动状态时，我们调用“car.motor(左轮速度和方向, 右轮速度和方向)”接口控制小车前进、后退、左转和右转，该接口的两个参数分别代表左轮速度和方向、右轮速度和方向，负数表示反转，正数表示正转，数值绝对值的大小代表速度，速度取值 0 ~ 255。

此外，本示例也展示如何控制 RunGo 小车的左/右头灯。

### 识别地面颜色 (色块)

RunGo 小车的底部有一个颜色识别传感器，可用于识别地面的颜色或色块，有效识别区域是巡线传感器的区域 (颜色识别传感器在标注 “P2” 文字的地方)。本示例程序实现的效果：让小车置于白、红、黄、绿、青、蓝或紫色的地面或贴有这些颜色色块之上，按下 A 按钮后，IoTs2 的 RGB 彩灯将显示对应的地面颜色，并在 LCD 屏幕上显示颜色的名称字符串 (White、Red、Yellow、Green、Cyan、Blue 或 Purple)。

示例程序代码如下：

```
from hiibot_iots2 import IoTs2  # IoTs2
from hiibot_iots2_rungo import RunGo
iots2 = IoTs2()
car = RunGo()
iots2.pixels.brightness = 0.3
iots2.pixels[0] = (0,0,0)
car.stop() # stop motors
print("Press Button-A to sense ground color")
car.pixels.fill(0)
car.pixels.show()
while True:
    car.buttons_update()
    if car.button_A_wasPressed:
        cid = car.groundColorID # get ground color id (0~6)
        print(car.groundColor_name[cid])
        iots2.pixels[0] = car.groundColor_list[cid]
```

请将上述示例代码保存到 IoTs2 的/CIRCUITPY/code.py 文件，每次按下 RunGo 的 A 按钮即可执行一次 “地面颜色” 识别，并将识别出来的颜色名字字符串显示到 LCD 屏幕上，同时 IoTs2 模块上的 RGB 彩灯 (USB 插座旁边) 也显示出同样的颜色。

上述示例程序的无穷循环程序块中，我们使用 “car.buttons_update()” 接口检测 A 按钮是否被按下，如果被按下则开始识别地面颜色并返回颜色识别结果 (颜色 ID)，使用 “car.groundColor_name[color_id]” 列表返回该颜色 ID 对应的颜色名称 (字符串) 并打印到屏幕上，然后使用 “car.groundColor_list[id]” 列表返回该颜色 ID 对应的颜色的 RGB 分量值 (元组类型)，并让 IoTs2 的 RGB 彩灯显示这种颜色。

---

### 电子围栏

前面的示例程序中逻辑问题都是非常简单的，下面我们来实现一个稍微复杂一点的程序逻辑和动作效果：地上画个黑色圆作为电子围栏的边界，RunGo 小车就在围栏内随意行驶。准备工作：在白色地面或纸上贴上宽度大于 1 公分以上的黑色胶带或不干胶，确保黑色胶带围成一个封闭的图案，并将 RunGo 小车放在图案内。图案可以参考下图所示：


执行下面的示例代码，你会看到 RunGo 小车在电子围栏内随意地行驶，但始终不会跑出围栏。

```
import time
# import RunGo module from hiibot_iots2_rungo.py
from hiibot_iots2_rungo import RunGo
car = RunGo()
print("Run Go!")
# speed=100, 0, forward; 1, backward; 2, rotate-left; 3, rotate-right
car.stop() # stop motors
print("press Button-A")
car.rightHeadLED = 0
car.leftHeadLED = 0
carSpeed_fast = 100
carSpeed_slow = 70
carrun = False
idleCnt = 0
while True:
    idleCnt+=1
    if idleCnt>=50000:
```

(continues on next page)

(continued from previous page)

```
        for i in range(3):
            car.pixels[i] = (0,0,0)
        car.pixels.show()
    else:
        car.pixelsRotation()
    car.buttons_update()
    if car.button_A_wasPressed:
        carrun = True
        idleCnt = 0
        print("running")
    if car.button_B_wasPressed:
        car.stop()
        idleCnt = 0
        print("stop")
        carrun = False
    lt = car.leftTracker   # left sensor
    rt = car.rightTracker  # right sensor
    if carrun:
        idleCnt = 0
        if lt ==1 and rt ==1 :  # dual sensor above back-line
            car.stop()
            car.move(1, 0-carSpeed_fast)  # backward
            time.sleep(0.2)
            car.stop()
            car.move(2, carSpeed_fast)  # turn left
            time.sleep(0.2)
            car.stop()
        elif lt ==1 :  # left sensor above back-line only
            car.stop()
            car.rightHeadLED = 1
            car.move(3, carSpeed_fast)  # turn right
            time.sleep(0.2)
            car.stop()
            car.rightHeadLED = 0
        elif rt ==1 :   # right sensor above back-line only
            car.stop()
            car.leftHeadLED = 1
            car.move(2, carSpeed_fast)  # turn left
            time.sleep(0.2)
            car.stop()
            car.leftHeadLED = 0
        else:
            car.move(0, carSpeed_slow)  # forward
```

(continues on next page)

(continued from previous page)

```
61              time.sleep(0.02)
62      pass
```

将示例程序保存到 IoTs2 的/CIRCUITPY/code.py 文件中，等待我们的程序正式开始运行后，按下 RunGo 的 A 按钮，并将整个小车放在黑色胶带围成的封闭图案内，你将看到 RunGo 小车始终在围栏内行驶。当你想要让 RunGo 小车停下时，请按下 RunGo 的 B 按钮即可，或者直接关闭 RunGo 电源。

为什么 RunGo 小车不会越过黑色胶带围成的“围栏边界”呢？我们使用 RunGo 小车底部的一对循迹传感器来侦测小车是否到达“围栏边界”，如果遇到边界则根据这对传感器的状态来调整行驶方向：如果两个传感器都侦测到黑色边界，则先后退一段距离再左转；如果只有左侧传感器侦测到黑色边界则右转；如果右侧传感器侦测到黑色边界则左转；如果传感器都未侦测到黑色边界则继续前进。

这是本示例程序的无穷循环程序块中的关键逻辑，或者说这就是实现“电子围栏”效果的关键逻辑。本示例中增加 2 个按钮做交互实现开始行驶和停止行驶的功能，也属于无穷循环程序块的一部分逻辑。

为了达到更好的视觉效果，我们可以使用 RunGo 小车底盘的 3 颗彩灯来指示行驶、停车状态：在围栏内行驶期间 3 颗彩灯的颜色不断地转动；当停车时彩灯颜色全部保持白色。

你可以根据本向导底部的接口库介绍来掌握 RunGo 小车的控制接口，然后设计更加有趣的示例。

---

### 循迹小车

AGV(Automatic Guided Vehicle，无人搬运车) 小车已经是很多现代车间里最重要的物料“搬运工”！沿着预先规划好的路线能够无人且自动驾驶的货车能够将仓库的物料自动地运送到指定工位，并从指定工位将产品自动运送会成品仓库。这些曾经依靠人力或依靠司机开着货车来完成的工作，现在逐步被 AGV 代替。

AGV 如何实现“沿着规定路线行驶到指定停靠点”呢？有很多种方法可以实现 AGV 的功能，本向导给出一种循迹的方法。使用循迹传感器反馈的状态信号控制 RunGo 小车运动来模拟 AGV。

我们采用地面贴黑色胶带或黑色不干胶来“指定路线”，编程控制 RunGo 小车沿着该路线行驶 (允许弯曲的路线)，到达路线末端后自动调头并原路返回。为了更好地理解循迹的程序逻辑，我们先分析下图的三种情况：


根据上图所示，容易回答以下问题：如果小车向右偏离路线我们应该如何纠偏呢？向左偏离时又如何纠偏呢？此外，当我们达到道路末端时循迹传感器的状态是怎样？如何让 RunGo 小车绕自身中心调头呢？

**道路末端，原地调头** **沿道路前进**

简单地分析这几种特殊情况将有助于掌握下面的示例程序中的关键逻辑和代码。本示例的准备工作非常简单，使用前示例所用的黑色胶带围成的封闭边界作为本次循迹的“指定路线”。

循迹小车的示例程序如下：

```
import time
import random
from hiibot_iots2 import IoTs2  # IoTs2
from hiibot_iots2_rungo import RunGo
iots2 = IoTs2()
iots2.screen.rotation = 180
car = RunGo()
car.stop()  # stop car one second
```

(continues on next page)

(continued from previous page)

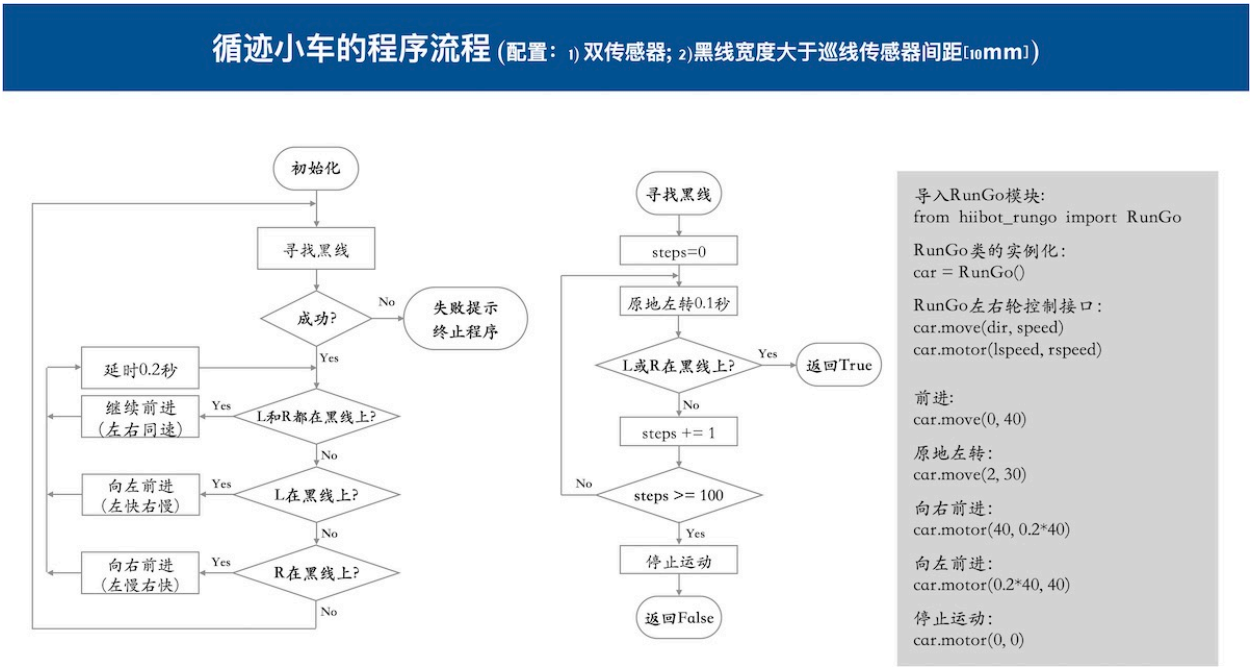
```
carspeed = 80
time.sleep(1)
running = False
def  searchBackLine():
    global car
    for steps in range(360):
        rdir = random.randint(0, 2)
        if rdir==0:
            car.move(2, 60)
        else:
            car.move(3, 60)
        time.sleep(0.005)
        if not car.rightTracker or not car.leftTracker:
            # backlin be searched by any sensor
            car.stop()
            return True
    car.stop()
    return False
while True:
    car.pixelsRotation()
    car.buttons_update()
    if car.button_A_wasPressed:
        running = True
        print("running")
    if car.button_B_wasPressed:
        car.stop()
        print("stop")
        running = False
    lt = car.leftTracker    # left sensor
    rt = car.rightTracker  # right sensor
    if running:
        # two sensors is above backline, go on
        if lt and rt:
            car.motor(carspeed, carspeed)
        # left sensor is above backline, but right sensor missed backline, thus turn␣
↪left
        elif lt:
            car.motor(carspeed//2, carspeed)
        # right sensor is above backline, but left sensor missed backline, thus turn␣
↪right
        elif rt:
            car.motor(carspeed, carspeed//2)
        # two sensors missed backline, thus stop car and search backline
```

(continues on next page)

(continued from previous page)

```
        else:
            car.stop()
            print("black line is missing, need to search the black line")
            if not searchBackLine():
                break
        time.sleep(0.01)
    else:
        pass
```

看起来程序代码很长！为了帮助你理解程序语句的作用，请分析下面的流程图，并对照程序代码、执行程序时 RunGo 小车的行为。

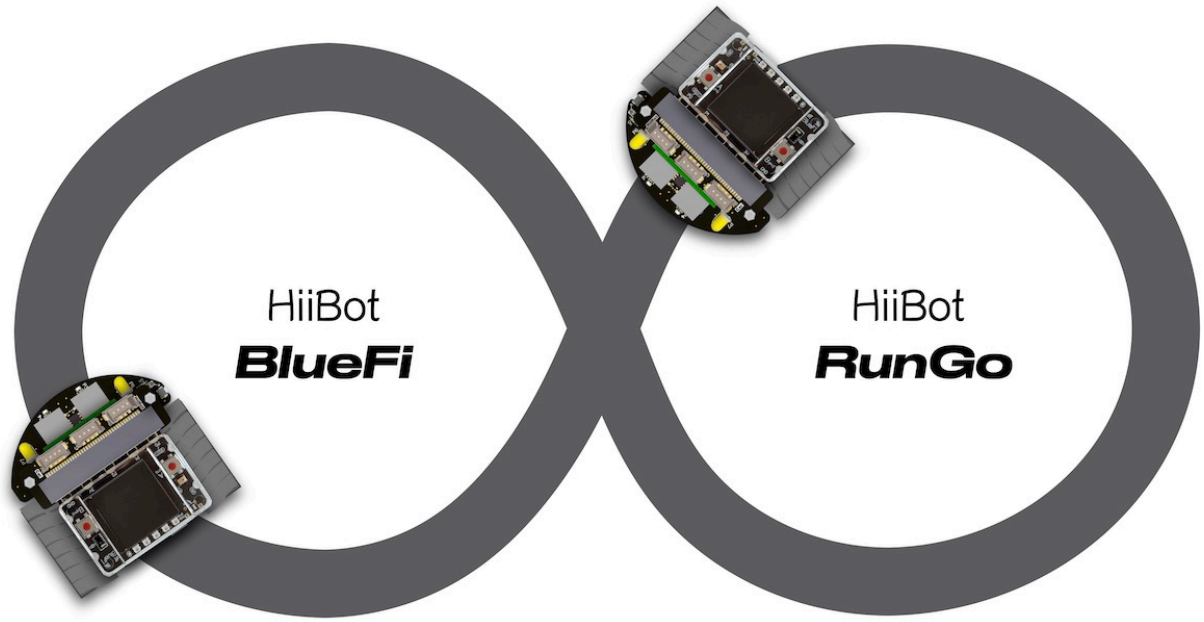



将上面的示例程序保存到 IoTs2 的/CIRCUITPY/code.py 文件中，然后将 RunGo 小车放在黑色胶带上方，等待我们的程序正式开始运行后，观察程序的执行效果。如果你想要让 RunGo 小车停下来，按下 B 按钮即可。如果想要 RunGo 小车继续巡线行驶，按下 A 按钮即可。

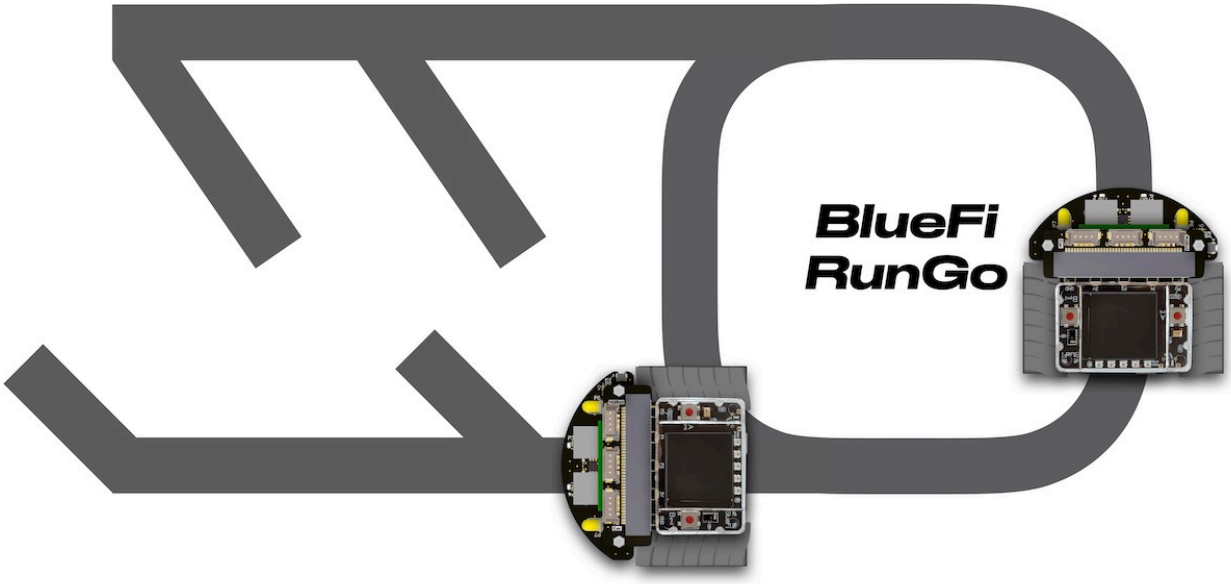
虽然本示例程序看起来很长，真正的循迹控制逻辑只是在无穷的循环体中。

此外，本示例程序中包含一个容错处理，被定义成子程序 searchBackLine。该子程序可以实现：当 RunGo 小车的两个循迹传感器都未检测到“指定路线”的黑色道路时，小车将自动开始绕自身中心旋转，找到黑色道路后再继续沿路行驶。如果你未将小车放在黑色道路上方，该容错程序将控制 RunGo 小车原地打转几圈来尝试找黑色道路，如果尝试几圈都未找到黑线则自动停车。

你也可以试一试如下图所示的“指定路线”，你能预测自己的 RunGo 小车会如何行驶?




事实上，企业车间的仓库分为原料仓库、半成品仓库、成品仓库等多种，生产工位较多，如何实现多点物料搬运？需要我们去探索，下图是多点物料搬运问题的抽象图例，你可以使用黑色胶带或不干胶绘制这些图中的“指定路线”，编程实现沿着这些“指定路线”自动搬运物料的小车。




或许你觉得单纯使用巡线传感器的信息并不足以实现自己的想法，RunGo 小车底盘带有颜色识别传感器，可以用来识别地面的颜色，如果我们在道路的分叉口的地面贴上一些特殊颜色，譬如红、黄、绿、青、蓝和紫色等，每种颜色代表不同的旋转方向，或许实现上图的多点之间货物运输会变得非常简单。动手试一试吧。

### AGV 避障

如果 AGV 行驶过程中遇到障碍物怎么办？譬如有人正好站在 AGV 行驶路线上，此时 AGV 绝对不能直接撞上去。问题是，AGV 如何知道前方有人？RunGo 小车带有一个超声波传感器，能够检测 2 公分到 4 米距离内的障碍物。下面我们修改前一个示例实现这一功能：当 RunGo 小车的行驶方向有障碍物时，让 RunGo 小车自动停下来，直到障碍物被移除。

---

### 让 RunGo 配合你扮演“气功大师”

武林高手能隔山打牛、隔空取物，气功大师能用气击倒对手。本示例的执行效果：让 RunGo 当个“托儿”帮助我们表演气功大师的绝招。气功大师不仅能用手掌“发气”隔空推动 RanGo 小车后退，还能用手掌隔空“吸引”RunGo 小车，其中的奥秘是什么呢？

请注意，本示例程序需要使用超声波传感器，请将超声波传感器模块正确地插在 RunGo 小车上。

本示例程序的代码如下：

```
import time
from hiibot_iots2 import IoTs2  # IoTs2
from hiibot_iots2_rungo import RunGo
iots2 = IoTs2()
car = RunGo()
car.stop()
iots2.screen.rotation = 180
print('Run Go!')
minDistance = 8.0
maxDistance = 15.0
carMaxSpeed = 80
speedsList = [carMaxSpeed, carMaxSpeed-10, carMaxSpeed-20, carMaxSpeed-30]
running = False
idleCnt = 0
while True:
    idleCnt += 1
    if idleCnt>50000:
        for i in range(3):
            car.pixels[i] = (0,0,0)
        car.pixels.show()
    else:
        car.pixelsRotation()
    car.buttons_update()
    if car.button_A_wasPressed:
        running = True
        idleCnt = 0
```

(continues on next page)

(continued from previous page)

```
        print('running')
    if car.button_B_wasPressed:
        running = False
        car.stop()
        idleCnt = 0
        print('stopping')
    if running:
        idleCnt = 0
        ds = car.distance
        if ds<minDistance:
            si = int(minDistance-ds)
            if si<len(speedsList):
                s = speedsList[si]
            else:
                s = speedsList[3]
            car.motor(-s, -s)
        elif ds>maxDistance:
            si = int(ds-maxDistance)
            if si<len(speedsList):
                s = speedsList[3-si]
            else:
                s = speedsList[0]
            car.motor(s, s)
        else:
            car.stop()
        time.sleep(0.01)
    else:
        pass
```

将示例程序保存到 IoTs2 的/CIRCUITPY/code.py 文件中，等待我们的程序正式开始运行后，按下 RunGo 的 A 按钮，然后用手掌靠近或远离 RunGo 小车的超声波，观察程序的执行效果是否有“武林高手”、“气功大师”隔空推车、隔空取物等效果。

---

### 帮助 RunGo 走出“巨石阵”

三国演义中诸葛亮在长江边摆的“巨石阵”让诸多敌人有进无出。你能使用今天的科技手段帮助 RunGo 走出纸杯模拟的“诸葛巨石阵”吗?

示例程序代码如下:

```
import time
import random
```

(continues on next page)

(continued from previous page)

```
from hiibot_iots2 import IoTs2  # IoTs2
from hiibot_iots2_rungo import RunGo
car = RunGo()
iots2 = IoTs2()
car.stop()
iots2.screen.rotation = 180
print('Run Go!')
minDistance = 8.0
maxDistance = 15.0
badDistance = 440.00
carMaxSpeed = 80
carMinSpeed = 60
pdt = [0, 0, 0]
running = False
idleCnt = 0
# 检查是否堵住 (堵住时连续的距离变化非常小)
def stallCheck(dt) :
    dif0 = abs(pdt[0] - pdt[1])
    dif1 = abs(pdt[1] - pdt[2])
    dif2 = abs(pdt[2] - dt)
    pdt[0] = pdt[1]
    pdt[1] = pdt[2]
    pdt[2] = dt
    if 0.4>max(dif0, dif1, dif2):
        return True
    else:
        return False
# 随机转弯
def randomTurn():
    global car, carMinSpeed, carMaxSpeed
    car.stop()
    time.sleep(0.01)
    dir = random.randint(0,2)
    if dir==1:
        car.motor(carMinSpeed, -carMinSpeed)
    else:
        car.motor(-carMinSpeed, carMinSpeed)
    time.sleep(0.5)
    car.motor(carMaxSpeed, carMaxSpeed)
# 先后退一段距离再随机转弯
def backThenRandomRurn():
    global car, carMaxSpeed
    car.stop()
```

(continues on next page)

(continued from previous page)

```
    time.sleep(0.01)
    car.motor(-carMaxSpeed, -carMaxSpeed)
    time.sleep(0.4)
    randomTurn()
# 主循环：检查是否待机，待机则关闭彩灯；检查启动 (A) 或停止 (B)；
# 启动后检测障碍物距离并决定前进/随机转弯/先后退再随机转弯等 3 种行为
while True:
    idleCnt += 1
    if idleCnt>50000:
        for i in range(3):
            car.pixels[i] = (0,0,0)
        car.pixels.show()
    else:
        car.pixelsRotation()
    car.buttons_update()
    if car.button_A_wasPressed:
        running = True
        idleCnt = 0
        print('running')
    if car.button_B_wasPressed:
        running = False
        car.stop()
        idleCnt = 0
        print('stopping')
    if running:
        time.sleep(0.01)
        idleCnt = 0
        dt = car.distance
        if stallCheck(dt):
            backThenRandomRurn()
            continue
        if dt<minDistance or dt>badDistance:
            backThenRandomRurn()
        elif dt<maxDistance:
            randomTurn()
        else:
            car.motor(carMaxSpeed, carMaxSpeed)
    else:
        pass
```

确保 IoTs2 模块正确地插在 RunGo 小车上，并开启 RunGo 电源 (RunGo 的电源开关旁边有一颗红色 LED 的亮/灭指示电源状态)，使用 USB 数据线将 IoTs2 模块与电脑连接好，当电脑资源管理器中出现 CIRCUITPY 磁盘后，将上述的示例代码保存到 IoTs2 的/CIRCUITPY/code.py 文件，当程序执行时按下 RunGo 的 A 按钮，并将 RunGo 小车放在“巨石阵”中，观察 RunGo 如何走出我们的“巨石阵”。

考虑到纸杯或其他物品组成的“巨石阵”中障碍物之间的距离，建议根据测试结果修改上述示例程序的第 10 行和第 11 行程序代码等号右边的数值，这两行语句常数值大小决定 RunGo 距离障碍物多远就开始随机转弯 (为什么是“随机”?) 多远则先后退再随机转弯。如果你设计的“巨石阵”中障碍物之间距离较小则将两行程序等号右边的数值也随之修改为更小的值，反之亦然。

本示例程序的关键逻辑包括：判断 RunGo 是否被堵、判断前方障碍物距离是否小于允许的最小距离或不大于最大距离等几种情况的识别，我们根据每一种情况来确定 RunGo 的下一步行为：继续前进、随机转弯、先后退再随机转弯等。

此外，我们使用 RunGo 的按钮 A 和 B 控制 RunGo 是否开始“闯阵”或停止，在程序运行期间保持 RunGo 底部的 3 个 RG 彩灯不断地旋转，如果停止时间超过 25 秒则关闭这些彩灯以节能。

---

### RunGo 的“趋光性”

RunGo 的前部带有一对光线强度传感器能够识别前方光线的方向 (哪个方向的光线更亮)

```
import time
from hiibot_iots2 import IoTs2  # IoTs2
from hiibot_iots2_rungo import RunGo
iots2 = IoTs2()
car = RunGo()
car.stop()
iots2.screen.rotation = 180
print('Run Go!')
minDistance = 8.0
maxDistance = 15.0
carMaxSpeed = 100
speedsList = [carMaxSpeed-30, carMaxSpeed-20, carMaxSpeed-10, carMaxSpeed]
running = False
idleCnt = 0
diff = 200
preDiff = 0
df_scale = 300
def checkDirection():
    global car, diff, preDiff, df_scale
    ls, rs = car.leftLightSensor, car.rightLightSensor
    preDiff = diff
    diff = abs(ls-rs)
    ediff = abs(diff-preDiff)
    if diff<df_scale:
        if diff>preDiff and ediff>100:
            return 3
        else:
```

(continues on next page)

(continued from previous page)

```
            return 0
    elif ls>rs:
        return 1
    else:
        return 2
while True:
    idleCnt += 1
    if idleCnt>50000:
        for i in range(3):
            car.pixels[i] = (0,0,0)
        car.pixels.show()
    else:
        car.pixelsRotation()
    car.buttons_update()
    if car.button_A_wasPressed:
        running = True
        idleCnt = 0
        print('running')
    if car.button_B_wasPressed:
        running = False
        car.stop()
        idleCnt = 0
        print('stopping')
    if running:
        idleCnt = 0
        dir = checkDirection()
        if dir==0:
            car.stop()
        elif dir==1:
            scale = int(diff/df_scale)
            if scale<len(speedsList):
                s = speedsList[scale]
            else:
                s = speedsList[3]
            car.motor(int(s*0.4), s)
        elif dir==2:
            scale = int(diff/df_scale)
            if scale<len(speedsList):
                s = speedsList[scale]
            else:
                s = speedsList[3]
            car.motor(s, int(s*0.4))
        else:
```

(continues on next page)

(continued from previous page)

```
71              car.motor(carMaxSpeed, carMaxSpeed)
72          time.sleep(0.01)
73      else:
74          pass
```

**总结:**

- 按钮输入
- 实体对象的属性的状态
- 变量
- 变量赋值
- 变量自增/自减
- 逻辑组合
- 逻辑判断和逻辑程序块
- 循环和嵌套循环
- 函数及其定义
- RGB 彩灯及其接口与光效控制
- 小车方向、速度
- 小车转弯 (差速)
- 颜色识别
- 光电反射传感器
- 巡线/循迹传感器
- 超声波测距传感器
- 本节中，你总计完成了 84 行代码的编写工作

**Important: RunGo 类 (hiibot_iots2_rungo.py 模块) 的属性和接口**

- car (自定义的 RunGo 类实例化对象):
  - from hiibot_iots2_rungo import RunGo # 从 hiibot_iots2_rungo.py 模块导入 RunGo 类
  - car = RunGo() # 将 RunGo 类实例化，实体对象”car”可以自定义为其他名称
- pixels (底盘像素彩灯子类，默认 3 颗 RGB(兼容 WS2812B)/50% 亮度/GRB 模式) 支持的接口方法和属性包括:

  - car.pixels.fill( (R,G,B) ): 填充全部像素为设定颜色
  - car.pixels.show(): 刷新全部像素
  - car.pixels.brightness: 全部像素的亮度属性值 (可读可写的)，属性值范围：0.0(灭)~1.0(最亮)
  - car.pixels[index]: 指定某个像素的颜色属性 (可读可写的), index 有效值范围：0~2；属性值为 (R, G, B)

- A 和 B 两个按钮支持的接口方法和属性包括：
  - car.button_A (属性, 只读的, 有效值：0/False(释放时) 或 1/True(按下时)), RunGo 的可编程按钮 A 的状态
  - car.button_B (属性, 只读的, 有效值：0/False(释放时) 或 1/True(按下时)), RunGo 的可编程按钮 B 的状态
  - car.buttons_update() (函数，无输入参数，无输出参数)，必须在 RunGo 控制程序的无穷循环中调用该函数，目的是更新按钮的状态和去抖动操作
  - car.button_A_wasPressed (属性, 只读的, 有效值：0/False(未被按下) 或 1/True(已被按下)), 检测 RunGo 的可编程按钮 A 是否已被按下
  - car.button_A_wasReleased (属性, 只读的, 有效值：0/False(未被释放) 或 1/True(已被释放)), 检测 RunGo 的可编程按钮 A 是否已被释放
  - car.button_A_pressedFor(dt) (函数，输入参数：被长按的时间阈值；返回值：0/False(未被长按超过指定时间) or 1/True(已被按下且超过指定时长)), 检测 RunGo 的可编程按钮 A 释放被长按超过指定的时间
  - car.button_B_wasPressed (属性, 只读的, 有效值：0/False(未被按下) 或 1/True(已被按下)), 检测 RunGo 的可编程按钮 B 是否已被按下
  - car.button_B_wasReleased (属性, 只读的, 有效值：0/False(未被释放) 或 1/True(已被释放)), 检测 RunGo 的可编程按钮 B 是否已被释放
  - car.button_B_pressedFor(dt) (函数，输入参数：被长按的时间阈值；返回值：0/False(未被长按超过指定时间) or 1/True(已被按下且超过指定时长)), 检测 RunGo 的可编程按钮 B 释放被长按超过指定的时间

- groundColor (地面颜色传感器) 支持的接口方法和属性包括：
  - car.groundColorID: 地面颜色 ID 属性值 (只读的)，地面颜色 ID 属性值有效范围：0~6
  - car.groundColorValue: 地面颜色的 RGB 值属性 (只读的)，该属性值为“元组型”颜色分量值：(R, G, B)
  - car.groundColor: 地面颜色的名称属性 (只读的)，地面颜色的名称有效值为：’white’ (ID=0)，‘Red’，‘Yellow’，‘Green’，‘Cyan’，’Blue’，’Purple’ (ID=6)

- LightSensor (小车前部光线强度传感器) 支持的接口方法和属性包括：
  - car.rightLightSensor: 右前部光线强度的属性值 (只读的)，该属性值有效范围：0~65535

  - car.leftLightSensor: 右前部光线强度的属性值 (只读的)，该属性值有效范围：0~65535

- HeadLED (小车 (黄色) 前灯) 支持的接口方法和属性包括：
  - car.rightHeadLED: 右 (黄色) 前灯的属性值 (可读可写的)，该属性值有效范围：1 或 0, True 或 False；1 或 True: On, 0 或 False: Off
  - car.leftHeadLED: 左 (黄色) 前灯的属性值 (可读可写的)，该属性值有效范围：1 或 0, True 或 False；1 或 True: On, 0 或 False: Off
- distance (超声波测距传感器) 获取的小车与障碍物之间的距离属性值 (只读的)，有效值范围：2.0~400.0, 量纲为 cm(厘米)
- Tracker (小车底盘的巡线传感器) 支持的接口方法和属性包括：
  - car.rightTracker: 右前部巡线传感器的状态属性值 (只读的)，该属性值有效范围：1 或 0, True 或 False；1 或 True: 黑线, 0 或 False: 非黑线
  - car.leftTracker: 左前部巡线传感器的状态属性值 (只读的)，该属性值有效范围：1 或 0, True 或 False；1 或 True: 黑线, 0 或 False: 非黑线
  - car.tracking(mode): 巡线传感器对儿的状态属性值 (只读的)，该属性值有效范围：1 或 0, True 或 False；1 或 True: 小车在线上, 0 或 False: 小车偏离线；mode 有效值：0: 使用较宽 (线宽大于两个巡线传感器的间距 [1cm]) 的黑色线，左右巡线传感器同时在黑线上；1: 使用较窄 (线宽小于两个巡线传感器的间距 [1cm]) 的黑色线，仅左巡线传感器在黑线上；2: 使用较窄 (线宽小于两个巡线传感器的间距 [1cm]) 的黑色线，仅右巡线传感器在黑线上；3: 使用较宽 (线宽大于两个巡线传感器的间距 [1cm]) 的白色线，左右巡线传感器同时在白线上

---

### 1.8.2 机器人扩展板

机器人扩展板带有 2 个直流马达控制接口、16 路 PWM/Servo(舵机) 控制接口，板载锂电池充放电管理单元，以及放电电流高达 6A@5V 的高效 DC-DC 单元。由于机器人扩展板的工作电流较大，电源的开关采用极低导通电阻的大功率 P-MOS 和轻触开关等，有效地避免物理开关的打火等不安全因素。机器人拓展板使用单节 3.7V 的锂电池 (满电电压 4.2V)，要求锂电池能支持 10 倍率的放电电流。

---

---

**总结：**

- 直流马达控制
- PWM 信号
- Servo(舵机) 控制
- 机器人关节
- 机械动力
- 本节中，你总计完成了 190 行代码的编写工作

**Important: RobotModule 类的属性和接口**

- motor (函数，输入参数：马达 1 的速度，马达 2 的速度)

## 1.9 UART 通讯

UART(通用异步收发器) 是 IoT 系统的基本功能组件，使用 2 个信号线即可实现两个 IoT 系统之间的全双工通讯，这两个信号分别称作 RxD(异步串行数据输入)、TxD(异步串行数据输出)。异步通讯，意味着通讯双方不使用专用的同步时钟信号，譬如 I2C 和 SPI 等串行通讯接口都有专用的同步时钟信号。全双工通讯指的是通讯双方可以同时发送和接收数据。

事实上，我们一直在使用 UART，譬如使用 USB 数据线将 IoTs2 与电脑连接后，在 MU 编辑器中使用 REPL，即输入一行完整的 Python 脚本程序并按下回车键 (Enter)，IoTs2 的 Python 解释器立即执行该程序语句并产生执行结果。使用 REPL 期间，IoTs2 的 Python 解释器与电脑上的 MU 编辑器之间使用 UART 通讯，当然现在的桌面计算机几乎都没有标准的 UART 接口，电脑端使用的是“USB-Serial”桥虚拟串口。目前仍有很多小设备在使用 UART，譬如 GPS 模块、NBIoT 等蜂窝网通讯模块、WiFi 和 BlueTooth 通讯模块。

IoTs2 具有 2 个标准的 UART 功能单元，意味着我们可以使用 UART 接口同时连接至少 2 个独立的 UART 接口型外部功能模块，譬如 GPS、NBIoT 等。但是需要注意的是，IoTs2 的其中一个 UART 也被一级 BootLoader 用作固件更新和下载端口使用，当 IoTs2 的复位后到二级 BootLoader 启动前这段时间内该 UART 也用作 CPU 启动信息输出使用。因此，使用 IoTs2 的 Rx 和 Tx 两个引脚 (在 IoTs2 板上有清晰地标示) 当作 UART 的信号引脚时务必注意复位后 Tx 引脚将会输出一些字符串信息，请确保这些信息不会影响外部接口的功能。

IoTs2 的另外一个 UART 的 2 个信号可以使用任一 I/O 引脚！除了已固定用作 UART 的 Rx 和 Tx 两个引脚外，IoTs2 总计有 16 个 GPIO，你可以使用其中的任意 2 个引脚当作 UART 的 2 个信号接口。

### 1.9.1 单个 IoTs2 的“数字接龙”

UART 的应用场景大多数是 2 个系统之间的互联通讯，当我们手上仅有一个 IoTs2 时，除了使用“USB-Serial”桥虚拟串口与电脑通讯之外，仍可以实现“自发自收”功能来测试 UART 通讯，掌握 UART 通讯接口的编程。

我们的第一个示例程序的执行效果如下图：

执行该示例程序之前，请使用一根两母头的杜邦线 (或跳线帽) 将 IoTs2 的 IO17 和 IO18 引脚短接，示例程序初始启动后会在 LCD 屏幕上提示“Press the button to start game”，即按下 IoTs2 的按钮 (IO21 位置的按钮，不

是 RST 按钮！) 即可开始游戏。一旦按下 IoTs2 的按钮，我们将会在 LCD 屏幕上看到“数字接龙”游戏的效果。

本示例程序的代码如下：

```
import time                 # time moudle
import board, busio         # board and busio modules
from hiibot_iots2 import  IoTs2      # use the Button and BlueLED of IoTs2

iots2 = IoTs2()
iots2.blueLED_bright = 1.0          # turn on BlueLED
uart = busio.UART(
        board.IO17, board.IO18,     # two Pins(TxD, RxD)
        baudrate=115200,            # baudrate: 9600 is default
        timeout=0.01,               # waiting time(s) for read(nBytes), and
↪readinto(buf, nBytes)
        receiver_buffer_size=1)     # size of buffer
uart.reset_input_buffer()           # clear input buffer
outBuf = bytearray(1)               # out buffer, uart.write(buf, nBytes)
print("Press the button to start game")
while True:
    iots2.button_update()         # update the Button State
    if iots2.button_wasPressed : # start our game, send a digit
        outBuf[0] = 0             # any digit: 0~255
        uart.write(outBuf, 1)     # send the first digit
        print("Go it! the first was sent")
    if uart.in_waiting>0:         # check input buffer isn't empty
        inBuf = uart.read(1)      # return a bytearray type
        if inBuf is not None :    # inBuf isn't None
            iots2.blueLED_toggle()       # toggle BlueLED (turn off)
            time.sleep(0.1)
            print( 'Rec: {}'.format(inBuf[0]) )
            if int(inBuf[0]) < 255:
                outBuf[0] = inBuf[0]+1  # increment and send this digit
            else:
                outBuf[0] = 0           # rollback: .., 254, 255, next 0
            uart.write(outBuf, 1)
            iots2.blueLED_toggle()       # toggle BlueLED (turn on)
            time.sleep(0.1)
```

将上面的示例代码复制-粘贴并保存到 IoTs2 的/CIRCUITPY/code.py 文件中，然后 IoTs2 将立即开始执行该程序。请将 IO17 和 IO18 两个 I/O 引脚短接，我们将会看到上图的执行效果。

程序代码的详细注释如下：

- 第 1 行，导入 time 模块

- 第 2 行，导入 board 和 busio 两个 IoTs2 的 Python 解释器内建的模块。board 模块定义 IoTs2 的所有 GPIO 引脚名，busio 模块的 UART 子类是本示例的核心
- 第 3 行和第 4 行，从/lib/hiibot_iots2.py 模块中导入 IoTs2 类
- 第 5 行，将 IoTs2 类实例化为 iots2 对象
- 第 6 行，将 iots2 对象的 blueLED_bright 属性 (即 BlueLED 的亮度) 设置为 1.0(即最亮)
- 第 7 ~ 11 行，将 busio.UART 实例化为 uart 对象，并指定该对象的 TxD 和 RxD 引脚分别为 board.IO17 和 board.IO18，波特率为 115200，超时等待时间为 0.01 秒，接收缓冲区仅 1 个字节
- 第 12 行，清空 uart 对象的接收缓冲区，丢弃以前的接收数据
- 第 13 行，定义一个名叫 outBuf 的字节数组 (即 bytearray 型) 当作发送缓冲区，本示例每次仅发送一个字节，因此该数组的长度为 1
- 第 14 行，向 Python 解释器的串口控制台 (如 MU 编辑器的串口控制台) 和 LCD 屏幕输出提示信息
- 第 15 行，定义一个无穷循环
- 第 16 行，(无穷循环的第 1 行) 更新 iots2 对象的按钮状态
- 第 17 行，(无穷循环的第 2 行) 检查 iots2 对象的按钮是否已被点击，如果被点击则执行以下三行语句
- 第 18 ~ 20 行，(无穷循环的第 3 ~ 5 行) 当前一句 if 条件满足时，首先设置 outBuf[0] 为 0，然后调用 uart 对象的 write 接口将 outBuf 中的这个数据发送出去，最后向控制台/LCD 输出提示信息
- 第 21 行，(无穷循环的第 6 行) 判断 uart 对象的 in_waiting 属性值是否大于 0，如果大于 0 则执行下面的 12 行语句，否则继续下一个无穷循环
- 第 22 ~ 23 行，(无穷循环的第 7 ~ 8 行) 如果 uart.in_waiting 属性值大于 0，则调用 uart 对象的 read 接口将接收缓冲区的数据读到 inBuf 数组，如果 inBuf 数组是非空的则执行下面的 10 行语句，否则继续下一个无穷循环
- 第 24 行，(无穷循环的第 9 行) 切换 iots2 对象的 BlueLED 状态
- 第 25 行，(无穷循环的第 10 行) 延迟 0.1 秒
- 第 26 行，(无穷循环的第 11 行) 向控制台或 LCD 屏幕输出提示信息，其中包含接收到的数据的十进制表示
- 第 27 ~ 30 行，(无穷循环的第 12 ~ 15 行) 将 outBuf[0] 设置为 inBuf[0]+1，如果加 1 后的值大于 255 则设置 outBuf[0]=0(即从 255 回滚到 0)。注意：这 4 个语句是“数字接龙”游戏的关键
- 第 31 行，(无穷循环的第 16 行) 使用 uart 对象的 write 接口将 outBuf[0] 发送出去
- 第 32 行，(无穷循环的第 17 行) 切换 iots2 对象的 BlueLED 状态
- 第 33 行，(无穷循环的第 18 行) 延迟 0.1 秒

请注意本示例程序的第 6 ~ 10 行定义 uart 对象的方法，如果我们打算使用 IoTs2 的其他 I/O 引脚当作 uart 对象的 TxD 和 RxD，则修改第 7 行中的 I/O 引脚编号即可。此外，baudrate 参数是保障通讯双方正确通讯的基本参数，如果两个系统的此参数不一致将会导致无法正常通讯，务必将通讯双方的该参数配置为相同的值。

UART 子类的更多属性和接口方法请参见本文底部。

### 1.9.2 多个 IoTs2 的“数字接龙”

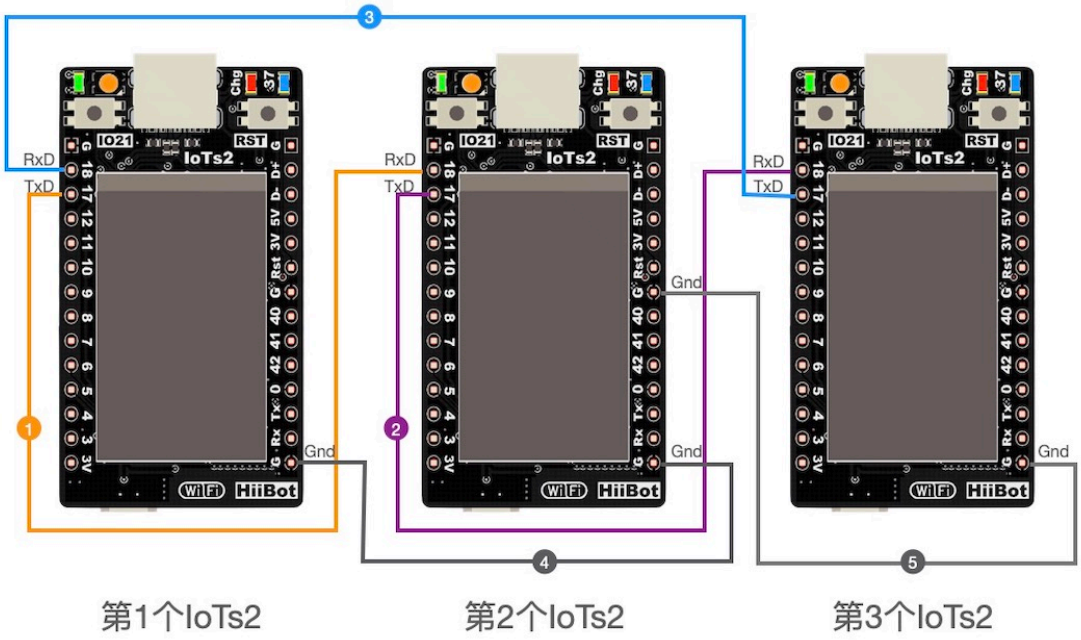
仅使用一个 IoTs2 实现的数字接龙非常好理解，TxD 和 RxD 两个信号短接，TxD 引脚发送出去的数据被 RxD 接收到，然后加 1 并再发送出去，期间惟一要注意的是“如果加 1 后大于 255”则回滚到 0，前面动图中看到的效果就不难理解。为啥我们发送的数据不能大于 255 呢？你能说出其中的缘由吗？

如果我们有多个 IoTs2 玩“数字接龙”游戏或许更好玩，甚至还会出现某些意外，找出这些意外的根源所在，能够帮助我们更好地理解 UART 及其编程应用。首先将 3 个 IoTs2 使用 UART 通讯接口连接在一起，当然更多个 IoTs2 的连接思路是相同的，你可以称这种连接为“菊花链”形式：

- 第 1 个 IoTs2 的 TxD 与第 2 个 IoTs2 的 RxD 连接
- 第 2 个 IoTs2 的 TxD 与第 3 个 IoTs2 的 RxD 连接
- 
- 第 n-1 个 IoTs2 的 TxD 与第 n 个 IoTs2 的 RxD 连接
- 第 n 个 IoTs2 的 TxD 与第 1 个 IoTs2 的 RxD 连接

连接他们需要的材料仅仅是杜邦线。然后将上面的示例程序保存到所有的 IoTs2 的 code.py 文件中，无需任何修改，给所有 IoTs2 通上电，随意按下某个 IoTs2 的按钮，我们观察所有 IoTs2 的 LCD 屏幕，你会发现连续的数字在多个 IoTs2 之间传递，“数字接龙”游戏难道不是这样吗？

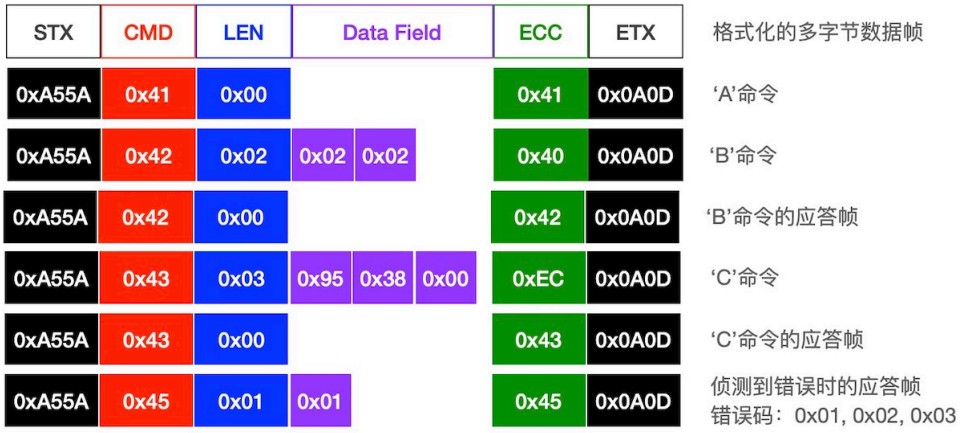
3 个 IoTs2 的连接方法如下示意图，需要使用 5 根双母头的杜邦线，按下图连接即可。



除了 TxD 和 RxD 两个异步串行通讯的信号之外，多个 IoTs2 之间使用 UART 通讯时，务必将这些 IoTs2 的 Gnd 引脚连接在一起。上图中的第 (4) 和第 (5) 根线将 3 个 IoTs2 俩俩连接，由于 IoTs2 的双排插针上的所有 Gnd 引脚是内部连通的，上图的连接方法实质上是将 3 个 IoTs2 的 Gnd 连接在一起。

### 1.9.3 数据帧的收发操作

前面的示例中每次仅发送/接收单个字节，在实际的异步串行通讯应用中往往需要发送/接收多个字节组成的特定格式的数据帧，譬如下图所示的数据帧：




虽然上图所示的数据帧仅仅是一个示例，但实际应用中的数据帧绝大多数都是这种形式。通讯双方定义特定格式的数据帧及其每一个数据域中每个字节的含义，这就是通讯协议的重要组成部分，目的是遵循此协议的收发双方能够明确每个数据帧代表的具体信息。

下面的示例程序采用上图的通讯协议来接收数据帧，并针对不同数据帧发送不同的应答，当接收者发现错误时则发送错误所对应的应答。

程序代码如下：

```
import time                 # time moudle
import random
import board, busio         # board and busio modules
from hiibot_iots2 import  IoTs2      # use the Button and BlueLED of IoTs2
iots2 = IoTs2()
iots2.blueLED_bright = 1.0          # turn on BlueLED
uart = busio.UART(
        board.IO3, board.IO4,       # two Pins(Tx, RxD)
        baudrate=115200,            # baudrate: 9600 is default
        timeout=0.01,               # waiting time(s) for read(nBytes), and
↪readinto(buf, nBytes)
        receiver_buffer_size=10)    # size of buffer
uart.reset_input_buffer()           # clear input buffer
outBuf = bytearray(1)               # out buffer, uart.write(buf, nBytes)
```

(continues on next page)

(continued from previous page)

```
sFrame = (  bytearray(b'\x41\x00'),
            bytearray(b'\x42\x02\x02\x45'),
            bytearray(b'\x43\x03\x12\x34\x56') )
nFrame = (2, 4, 5)
print("Press the button to send a Cmd")
# send the formated data frame: STX, CMD, Len, Data Field, ECC, ETX
# given CMD, LEN, Data Field, then calculate ECC and append it on the frame
def send(bytes, nBytes):
    global uart
    if nBytes<2:
        return
    if nBytes>5:
        nBytes = 5
    sbuf = bytearray(b'\xA5\x5A')
    ecc = 0x0
    for i in range(nBytes):
        sbuf.append( bytes[i] )  # append a byte on the sbuf
        ecc ^= bytes[i]
    sbuf.append(ecc)
    sbuf.append(0x0A)
    sbuf.append(0x0D)
    uart.write(sbuf, nBytes+5)
# send a response
def sendAck(cmd):
    sb = bytearray(b'\x00\x00')
    sb[0] = cmd
    send(sb, 2)
# send a error response
def sendErrorAck(errorCode):
    sb = bytearray(b'\x45\x01\x00')
    sb[2] += errorCode
    send(sb, 3)
    print('Recv failed{}'.format(errorCode))
# check the readable bytes of received buffer, then read and resolve it
def recv():
    global uart
    rbuf = bytearray(5)
    if uart.in_waiting<1:
        return None
    else:
        tryTimes = 0
        while True:
            time.sleep(0.1)
```

(continues on next page)

(continued from previous page)

```
            if uart.in_waiting>=7:
                break
            tryTimes += 1
            if tryTimes>3:
                uart.reset_input_buffer()
                return None
        time.sleep(0.02)
        rBytes = uart.in_waiting   # check the valid bytes
        rb = uart.read(rBytes)     # read the data frame
        if rb[0]!=0xA5 or rb[1]!=0x5A or rb[-2]!=0x0A or rb[-1]!=0x0D:
            sendErrorAck(0x01)
            return None
        ecc = 0x00
        for i in range(rBytes-5):
            rbuf[i] = rb[2+i]
            ecc ^= rbuf[i]
        if rb[-3]!=ecc:
            sendErrorAck(0x02)
            return None
        if rbuf[0]!=0x41 and rbuf[0]!=0x42 and rbuf[0]!=0x43 and rbuf[0]!=0x45:
            sendErrorAck(0x03)
            return None
        return rbuf
# resolve received data frame, and execute some action
def resolving(rbuf):
    nBytes = len(rbuf)
    if nBytes>=2:
        if rbuf[0]==0x41:
            sendAck(0x41)
            print('Recv Cmd 0x{:02X}, send Ack'.format(rbuf[0]))
        elif rbuf[0]==0x42:
            if rbuf[1]==0x02:
                sendAck(0x42)
                print('Recv Cmd 0x{:02X}, send Ack'.format(rbuf[0]))
            else:
                print('this 0x{:02X} response'.format(rbuf[0]))
        else: # rbuf[0]==0x43
            if rbuf[1]==0x03:
                sendAck(0x43)
                print('Recv Cmd 0x{:02X}, send Ack'.format(rbuf[0]))
            else:
                print('this 0x{:02X} response'.format(rbuf[0]))
    else:
```

(continues on next page)

(continued from previous page)

```
        pass
while True:
    iots2.button_update()         # update the Button State
    if iots2.button_wasPressed : # start our game, send a digit
        uart.reset_input_buffer()
        rd = random.randint(0,2) # only {0, 1, 2}
        send(sFrame[rd], nFrame[rd])
        print('send Cmd 0x{:02X} ok'.format(sFrame[rd][0]))
    recbuf = recv()
    if recbuf is not None:
        resolving(recbuf)
    time.sleep(0.01)
```

虽然这个示例程序代码看起来很多，112 行！但从程序架构角度看，非常好理解。整个示例代码分为 3 个部分：初始化部分，函数声明 (共 5 个函数) 部分，无穷循环部分。其中初始化部分仅 14 行代码，无穷循环仅 10 行代码，其他的代码用于声明 5 个 UART 通讯接口的功能函数。5 个 UART 通讯接口的功能函数包括，发送一个完整数据帧 (参见上图)，发送一个应答数据帧，发送一个接收错误的对应应答数据帧，接收一个完整数据帧 (期间并检查是否存在接收错误，如果发现接收错误则立即给出错误应答帧)，解析接收到的数据帧并给出应答。

将示例代码保存到 IoTs2 的/CIRCUITPY/code.py 文件后，将 IO3 和 IO4 短接，IoTs2 执行示例代码时，按下 IoTs2 的按钮，我们将会看到以下信息：

```
code.py output:
Press the button to send a Cmd
send Cmd 0x43 ok
Recv Cmd 0x43, send Ack
this 0x43 response
send Cmd 0x41 ok
Recv Cmd 0x41, send Ack
Recv Cmd 0x41, send Ack
Recv Cmd 0x41, send Ack
Recv Cmd 0x41, send Ack
Recv Cmd 0x41, send Ack
Recv Cmd 0x41, send Ack
..
send Cmd 0x42 ok
Recv Cmd 0x42, send Ack
this 0x42 response
```

按下 IoTs2 的按钮时，如果提示“send Cmd 0x42 ok”，或“send Cmd 0x43 ok”，然后将会再看到 2 行提示后程序将等待你再次按下按钮，但是如果提示“send Cmd 0x41 ok”之后，我们将会看到一连串的“Recv Cmd 0x41, send Ack”提示，除非你再次按下 IoTs2 的按钮，否则此过程将无穷无尽地持续下去。根据“resolving”函数的陈旭逻辑不难发现上述现象的缘由。

如果我们使用 2 个 IoTs2，请找来 3 根杜邦线将两个 IoTs2 连接起来：

- 第 1 个 IoTs2 的 IO3 与第 2 个 IoTs2 的 IO4 连接
- 第 2 个 IoTs2 的 IO3 与第 1 个 IoTs2 的 IO4 连接
- 两个 IoTs2 的 Gnd 连接在一起

并将此示例程序分别保存在两个 IoTs2 的 code.py 文件，无需任何修改，2 个 IoTs2 都通上电之后，按下任何一个 IoTs2 的按钮，此 IoTs2 的 LCD 屏幕将提示“send Cmd 0x42 ok”或“send Cmd 0x43 ok”，另一个 IoTs2 的 CD 屏幕上将提示“Recv Cmd 0x42, send Ack”或“Recv Cmd 0x43, send Ack”，然后前一个 IoTs2 的屏幕将提示“this 0x42 response”或“this 0x43 response”，接着程序进入等待，等待你再次按下按钮。

或者某次按下某个 IoTs2 的按钮时提示“send Cmd 0x41 ok”，接着两个 IoTs2 都将连续地提示“Recv Cmd 0x41, send Ack”，并无穷无尽地持续下去，除非我们再次按下某个 IoTs2 的按钮。

UART 是一种面向字节编码的串行通讯接口，即单次传送数据的最小单位是一个字节 (虽然该字节的实际二进制位数是可编程的，但习惯上仍称之为一个字节)，所以我们在示例程序中使用字节数组 (bytearray 型) 来交换发送和接收的数据。Python 的 bytearray 中每个数据单元是单个字节，取值范围为 0~255，字节数组支持“append()”接口将一个字节 (int 型) 数据添加到数组尾部，因此我们称之为字节数组，但该数据容器的数据个数支持动态地改变。此外，在 Python 语言中，字符串、字节数组、列表、元组等数据集都支持反向下标的索引，譬如示例程序中的第 66 行和第 73 行语句中，我们使用 rb[-1]、rb[-2]、rb[-3] 分别来访问字节数组的最后 3 个数据单元。

---

---

**总结：**

- UART 及其接口
- busio.UART 类

---

---

**Important: busio 类的 UART 子类的属性和接口**

- busio.UART(tx, rx, baudrate=9600, bits=8, parity=None, stop=1, timeout=1.0, receiver_buffer_size=64, flow=None) 参数如下：
  - tx, rx: 必须使用“board.IOx”来指定这两个引脚
  - baudrate: 波特率参数，默认值为 9600
  - bits: 数据位个数，默认值为 8
  - parity: 奇偶校验参数，默认值为 None(即不启用校验位)；另外两个有效值：busio.UART.Parity.EVEN 和 busio.UART.Parity.ODD
  - stop: 停止位个数，默认值为 1；有效值: 1, 1.5, 2
  - timeout: 阻塞式接收过程的最大等待时间参数，单位是秒
  - receiver_buffer_size: 接收缓冲区的字节个数

  - flow: 是否启用硬件流控，默认值为 None(即不启用硬件流控)

- in_waiting (属性, 只读, 有效值：0~receiver_buffer_size), 用于检查接收缓冲区内可读的字节数
- timeout (属性, 可读可写的), 指定 read()/readinto()/readline() 等接口的最大等待时间，单位是秒
- reset_input_buffer() (函数，无输入、输出参数和返回值) 清空接收缓冲区
- write(buf, nBytes) (函数, 输入参数: 待发送的字节数据 (必须是 bytearray 型)、待发送的字节个数, 返回值：发送成功的字节个数), 将 buf 中的前 nBytes 个字节数据从 TxD 引脚发送出去
- read(nBytes) (函数, 输入参数：待读取的字节个数, 返回值：bytearray 型 inBuf), 等待并读取从 RxD 引脚接收到的数据，当收到 nBytes 指定的字节个数则立即返回，否则一直等待到超时后返回 (返回值为 None，或实际读取到的数据)
- readinto(inBuf, nBytes) (函数, 输入参数：保存接收数据的字节数组，待读取的字节个数, 返回值：实际读取到的字节个数)，等待并读取从 RxD 引脚接收到的数据，数据保存在 inBuf 中，返回值为实际读取到的数据字节个数，返回值为 0 则 inBuf 为空
- readline() (函数, 无输入参数, 返回值：bytearray 型 inBuf), 等待并读取 RxD 引脚接收到的数据，直到出现“换行键”字符出现则立即返回，返回值为已读取到的数据；或者超时返回，返回值保存有实际读取到的数据

---

## 1.10 CAN(车载网络) 通讯

## 1.11 WiFi 接口与通讯

## 1.12 MQTT:IoT 消息订阅和发布

## 1.13 使用易造云的 IoT 平台

---

## 1.14 IoTs2 固件

IoTs2 的研发工作是持续的，我们将不断地更新固件和维护开源库。固件的下载链接、更新方法都在本向导中。

## 1.15 IoTs2 开源库

IoTs2 的研发工作是持续的，我们将不断地更新固件和维护开源库。开源库的下载链接和使用方法都在本向导中。

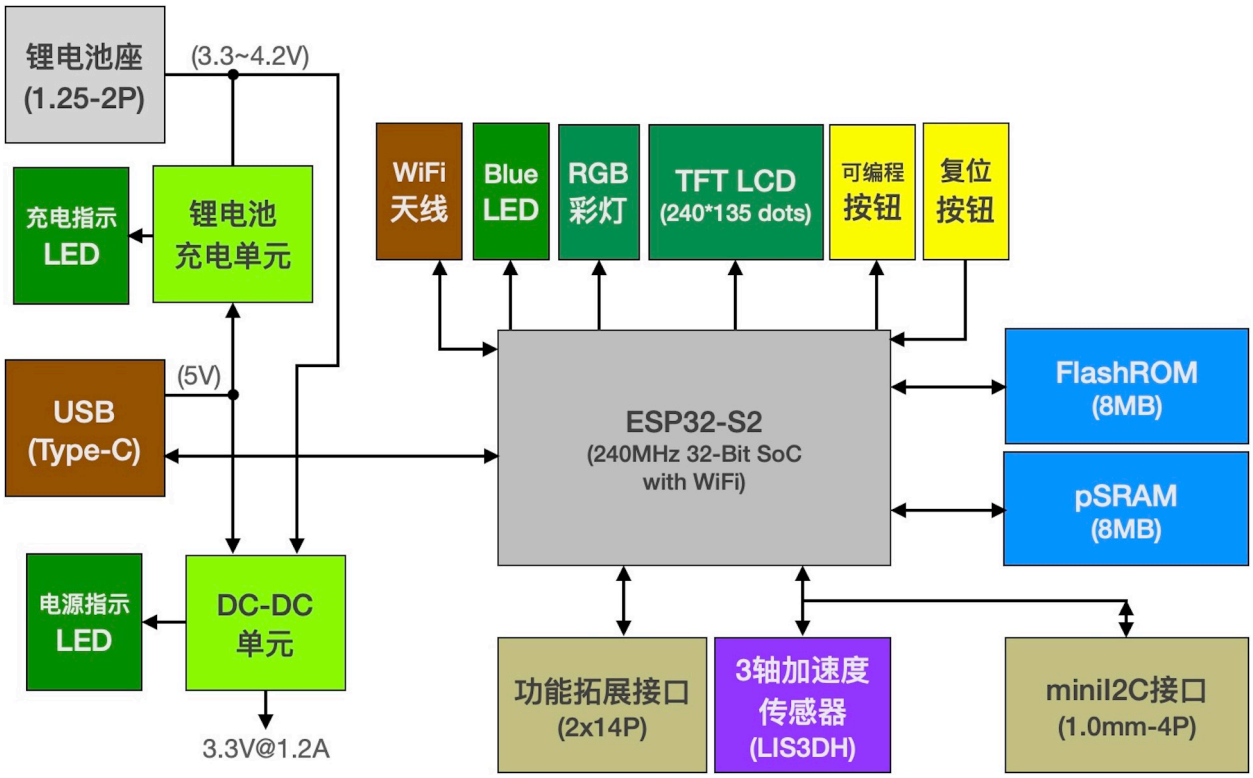
## 1.16 IoTs2 API (Python)

IoTs2 的 Python 版 API 简要说明及其应用实例

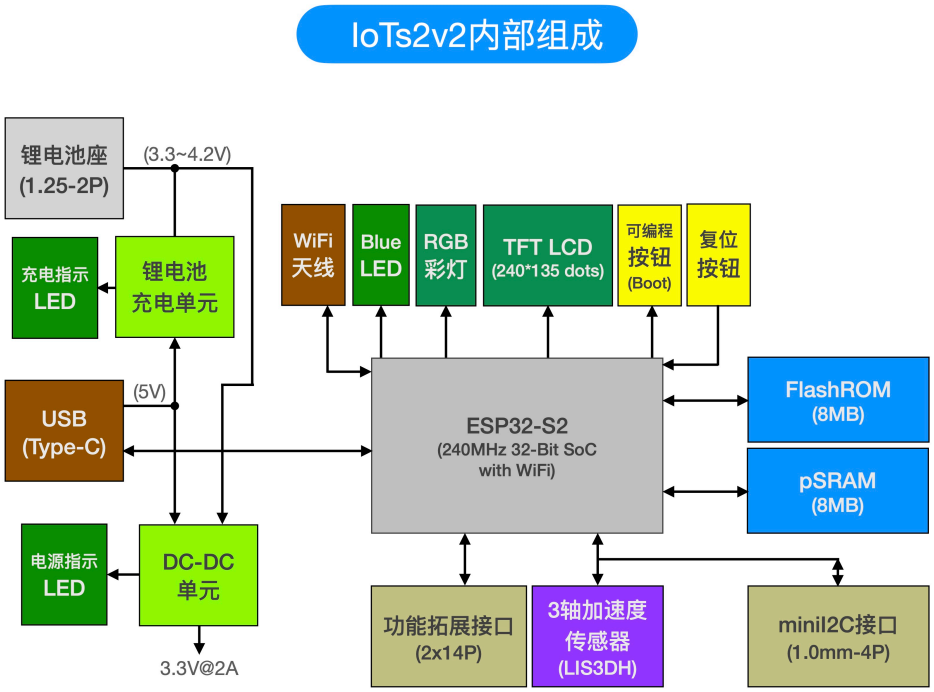
---

## 1.17 IoTs2 内部组成和电路原理图

IoTs2 的内部组成单元如下图所示：

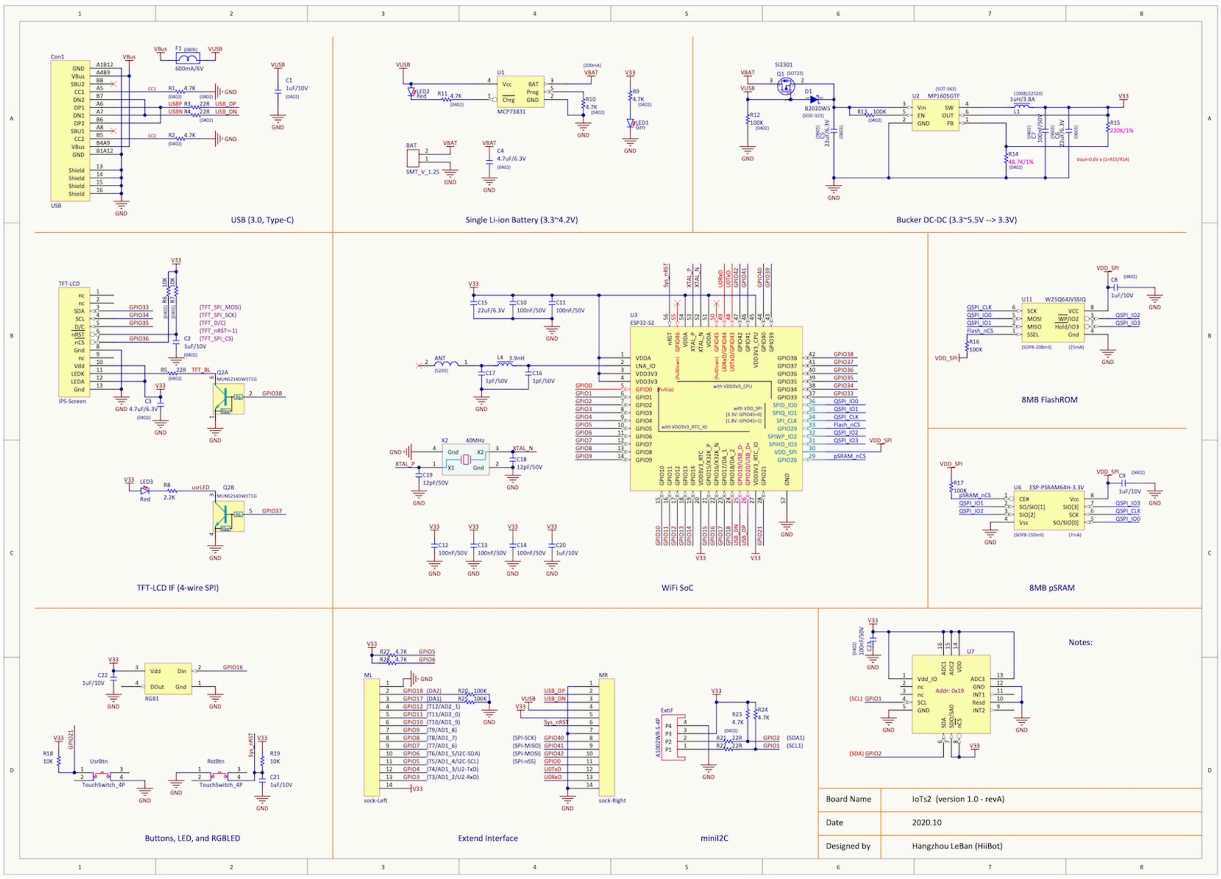



(IoTs2_v1)

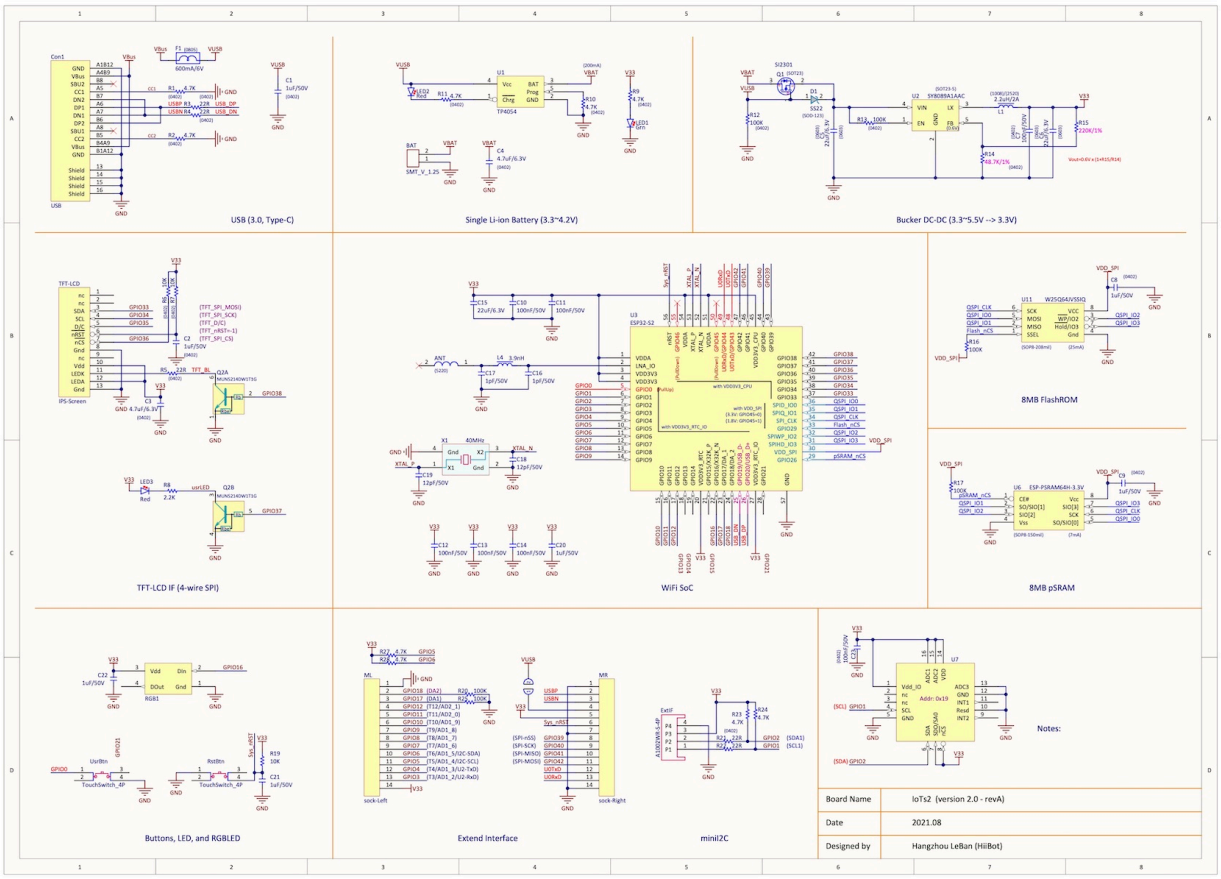



(IoTs2_v2)

IoTs2 的电路原理如下图所示：

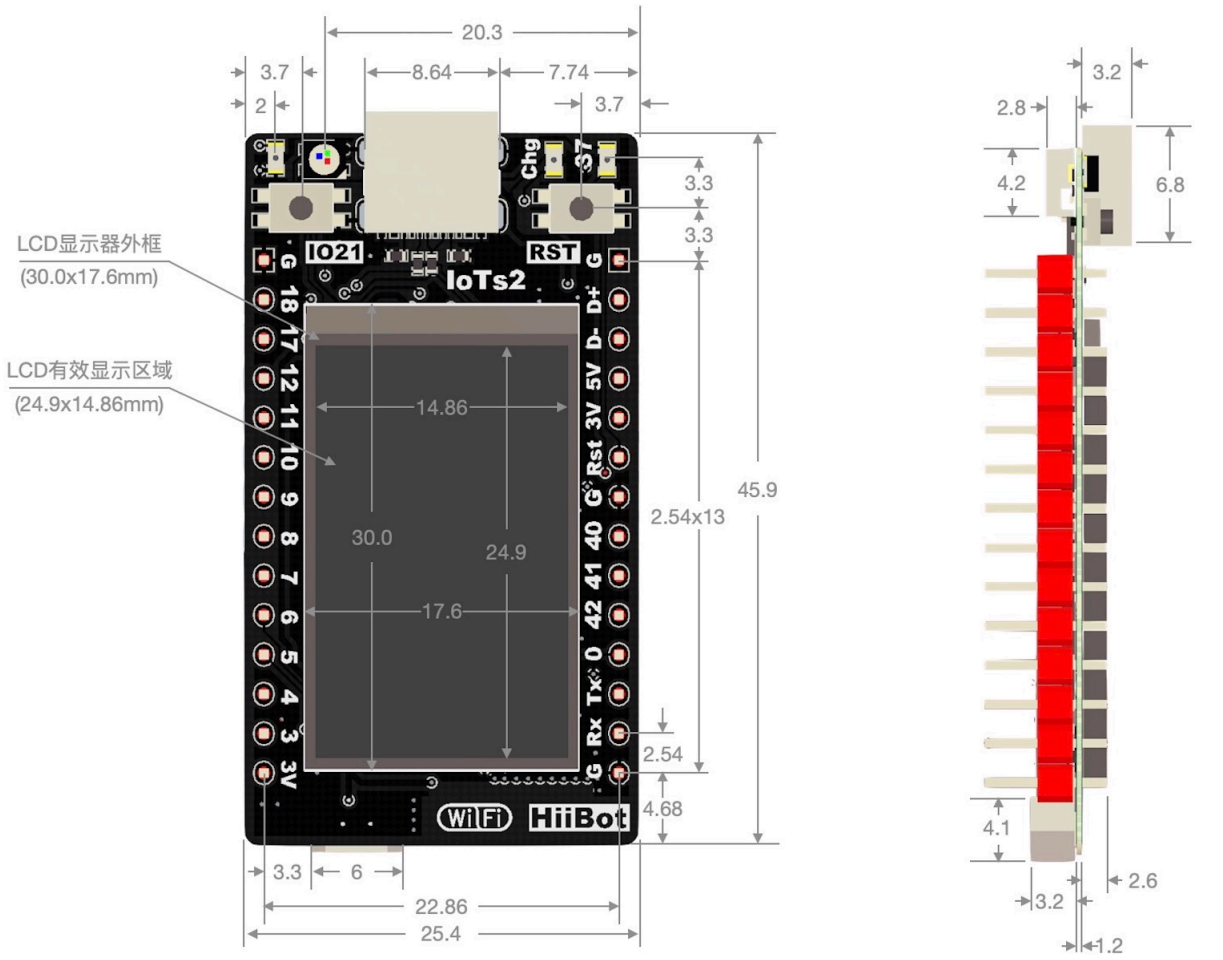

USB (3.0, Type-C)
Single Li-ion Battery (3.3~4.2V)
Bucker DC-DC (3.3~5.5V --> 3.3V)
TFT-LCD IF (4-wire SPI)
WiFi SoC
8MB FlashROM
8MB pSRAM
Buttons, LED, and RGBLED
Extend Interface
miniI2C
Notes:
Board Name
IoTs2 (version 1.0 - revA)
Date
2020.10
Designed by
Hangzhou LeBan (HiiBot)

(IoTs2_v1)

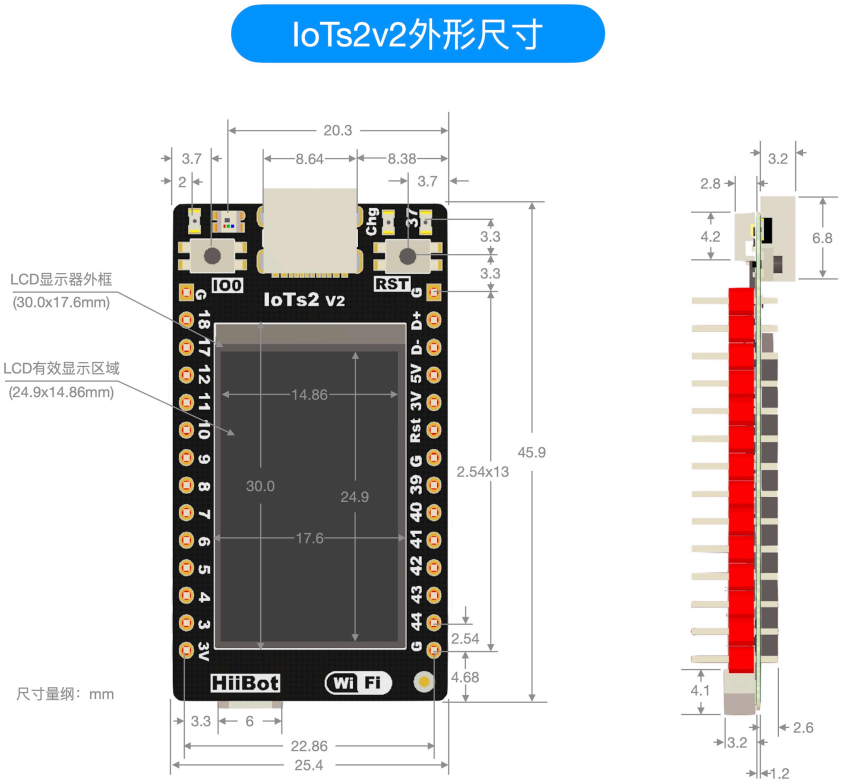



(IoTs2_v2)

IoTs2 的尺寸规格如下图所示：


20.3
3.7
8.64
7.74
2
3.7
3.3
3.3
LCD显示器外框
(30.0x17.6mm)
LCD有效显示区域
(24.9x14.86mm)
IO21
RST
IoTs2
Chg
14.86
30.0
24.9
17.6
45.9
2.54x13
2.54
4.68
WiFi
HiiBot
3.3
6
22.86
25.4
3.2
2.8
4.2
6.8
4.1
2.6
3.2
1.2

(IoTs2_v1)




(IoTs2_v2)

上图的尺寸量纲为 mm。