# ESP32 入门学习套件

XINZHILI

深圳市心力科技有限公司

二〇二五年四月

目录

深圳市心力科技有限公司.....1
套件清单.....5
课前准备.....6
1.1 安装驱动.....6
1.1.1 查看端口（Windows 系统）.....6
1.1.2 安装驱动.....7
1.2 Arduino 安装使用说明.....8
1.2.1 软件介绍.....8
1.2.2 下载与安装.....8
1.2.3 离线安装 ESP32.....14
1.2.4 添加 Arduino 库方法.....19
1.2.5 学习使用串口监视器.....22
第 1 章 ESP32 简介.....24
1.1 概述.....24
1.2 ESP32 规格.....25
1.3 组件介绍.....26
1.4 ESP32 引脚布局.....29
1.4.1 仅输入引脚.....31
1.4.2 集成在 ESP-WROOM-32 上的 SPI 闪存.....31
1.4.3 电容式触摸 GPIO.....32
1.4.4 模数转换器 (ADC).....32
1.4.5 数模转换器 (DAC).....34
1.4.6 RTC GPIO.....34
1.4.7 PWM.....35
1.4.8 I2C.....36
1.4.9 SPI.....36
1.4.10 中断.....36
1.4.11 捆绑引脚.....37
1.4.12 启动时为高电平的引脚.....37
1.4.13 启用 (EN).....38
1.4.14 GPIO 所消耗的电流.....38
1.4.15 ESP32 内置霍尔效应传感器.....38
第 2 章 ESP32 输入和输出.....38
2.1 概述.....38
2.2 所需组件.....38
2.3 预备知识.....39
2.3.1 ESP32 控制数字输出.....39
2.4 *ESP32* 读取数字输入.....39
2.4.1 项目实例.....39
2.5 所需部件.....40
2.6 示意图.....40
2.7 代码.....41
2.7.1 代码如何工作.....42

XINZHILI

2.7.2 上传代码 ..... 43
2.8 演示 ..... 44
第 3 章 ESP32 模拟输入 ..... 45
3.1 概述 ..... 45
3.2 模拟输入 (ADC) ..... 45
3.2.1 ADC 是非线性的 ..... 46
3.2.2 analogRead() 函数 ..... 46
3.3 所需元件 ..... 48
3.4 原理图 ..... 48
3.5 代码 ..... 49
3.6 测试示例 ..... 50
**3.7 总结** ..... 51
第 4 章 ESP32 PWM(模拟输出) ..... 52
4.1 概述 ..... 52
4.2 ESP32 LED PWM 控制器 ..... 52
4.3 所需元件 ..... 53
**4.4 电路图** ..... 53
4.5 代码 ..... 54
**4.6 测试示例** ..... 56
第 5 章 ESP32 PIR 运动传感器 ..... 57
5.1 概述 ..... 57
5.2 HC-SR501 人体感应传感器的工作原理 ..... 58
5.2.1 delay() 函数 ..... 59
5.2.2 millis() 函数 ..... 59
5.3 所需元件 ..... 59
5.4 电路图 ..... 60
5.5 代码 ..... 61
5.6 演示 ..... 61
第 6 章 ESP32 网络开关服务器 ..... 62
6.1 概述 ..... 62
6.2 所需元件 ..... 63
6.3 电路图 ..... 63
6.4 代码 ..... 64
6.5 演示 ..... 69
第 7 章 RGB LED 网络服务器 ..... 77
7.1 概述 ..... 77
7.2 原理 ..... 78
7.2.1 RGB LED 的工作原理 ..... 78
7.3 所需元件 ..... 79
7.4 电路图 ..... 80
7.5 演示 ..... 84
第 8 章 ESP32 继电器网络服务器 ..... 86
8.1 概述 ..... 86
8.2 电路图 ..... 90

XINZHILI

8.3 代码 ..... 92
8.4 演示 ..... 93
第 9 章 输出状态_同步_网络服务器 ..... 95
9.1 概述 ..... 95
9.2 原理 ..... 95
9.3 总结 ..... 96
9.4 所需元件 ..... 96
9.5 电路图 ..... 97
9.6 代码 ..... 99
9.7 演示 ..... 102
第 10 章 DHT11 网络服务器 ..... 104
10.1 概述 ..... 104
10.2 异步网络服务器 ..... 105
10.3 所需元件 ..... 105
10.4 电路图 ..... 106
10.5 安装库 ..... 106
10.5.1 安装 DHT 传感器库 ..... 106
10.5.2 安装 Adafruit 统一传感器驱动库 ..... 107
10.5.3 安装 ESPAsyncWebServer 库 ..... 107
10.5.4 为 ESP32 安装 Async TCP 库 ..... 108
10.6 代码 ..... 108
10.7 演示 ..... 111
第 11 章 ESP32 OLED 显示屏 ..... 112
11.1 概述 ..... 112
11.2 0.96 英寸 OLED 显示屏介绍 ..... 112
11.3 电路图 ..... 114
11.4 代码 ..... 115
11.5 代码工作原理 ..... 117

XINZHILI

# 套件清单



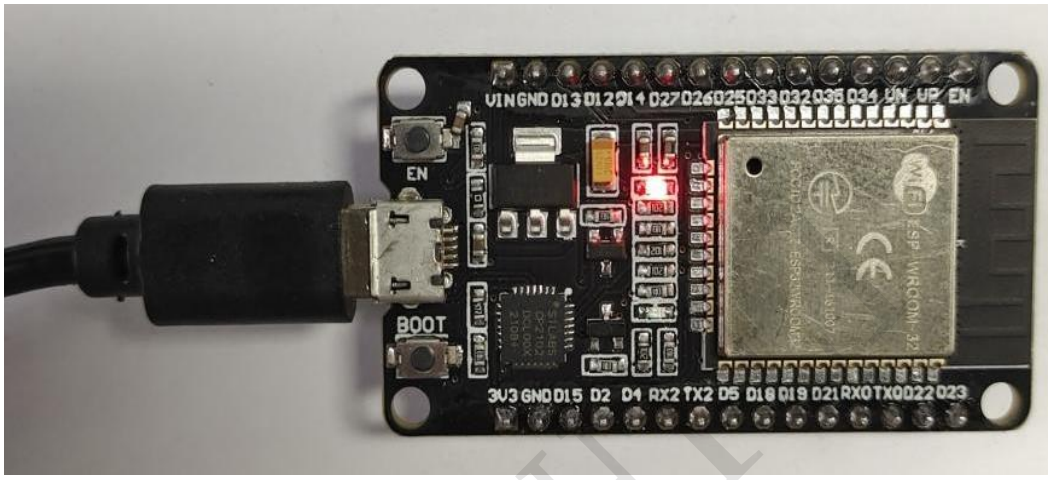

注：如果碰到有技术问题需要加技术人员指导，请+V：ManFishSea

# 课前准备

## 1.1 安装驱动

### 1.1.1 查看端口（Windows 系统）

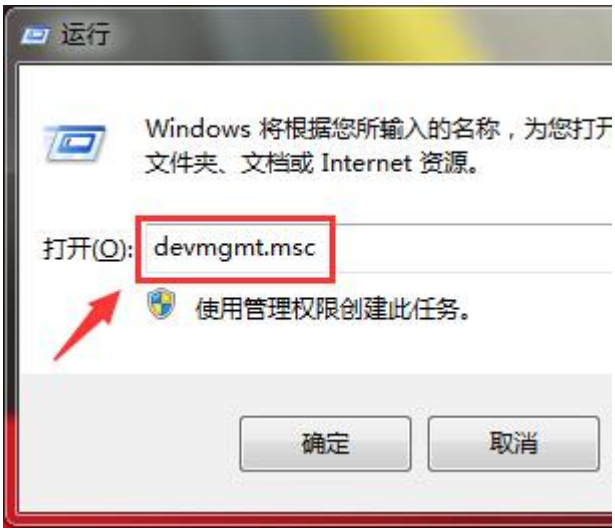
使用 USB 数据线将 ESP32 开发板与计算机连接起来，如下图所示



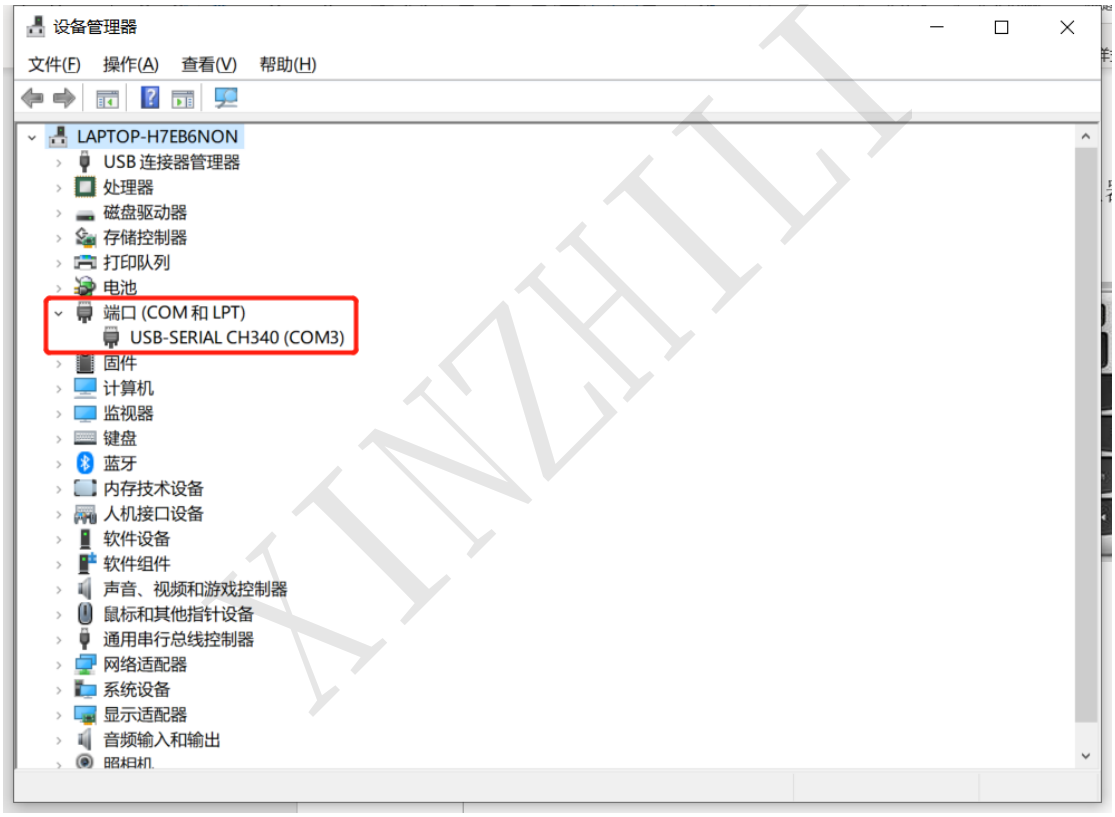

ESP32 开发板与计算机连接好以后，我们需要进入设备管理器查看端口号，首先同时按下 Windows 徽标键+R 键




输入“devmgmt.msc”并点击确定




调出设备管理器后即可看到端口号，如下图所示，记下这个端口号




XINZHILI

### 1.1.2 安装驱动

如果上一步您已经看到了端口，说明您的电脑已经识别了 ESP32 单片机，则可以忽略这一步。如果您的电脑没有识别到端口号，则需要手动安装驱动。进入资料包的“**软件**”文件夹，双击“**CH341SER.EXE**”进行驱动安装。安装成功后，重新插拔一下连接电脑和 ESP32 的数据线。重复上一步的步骤，在电脑的设备管理器查看电脑有没有检测到端口。

注：ESP32 对供电电压要求比较高，请使用质量比较高的数据线。如果是台式机，最好使用机箱背后的 USB 口。当检测不到，通常，换一下 USB 口也是常规操作。

## 1.2 Arduino 安装使用说明

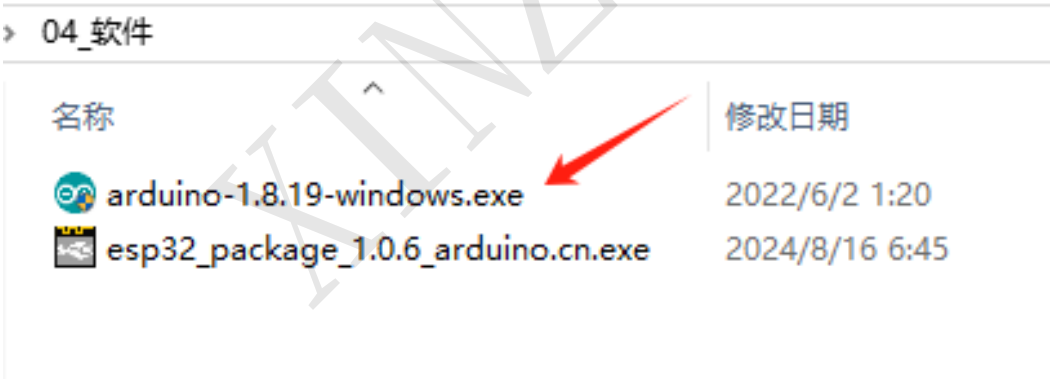
### 1.2.1 软件介绍

Arduino IDE 是一款专业的 Arduino 开发工具，主要用于 Arduino 程序的编写和开发，拥有开放源代码的电路图设计、支持 ISP 在线烧，同时支持 Flash、Max/Msp、VVVV、PD、C、Processing 等多种程序兼容的特点。

### 1.2.2 下载与安装

#### 1.2.2.1 下载（Windows）

进入资料包的“**软件**”文件夹，如下图所示，其中的“arduino-1.8.19-windows.exe”文件是我们已经下载好的 Arduino 安装包，您可以在资料包中找到该文件，直接双击安装即可。

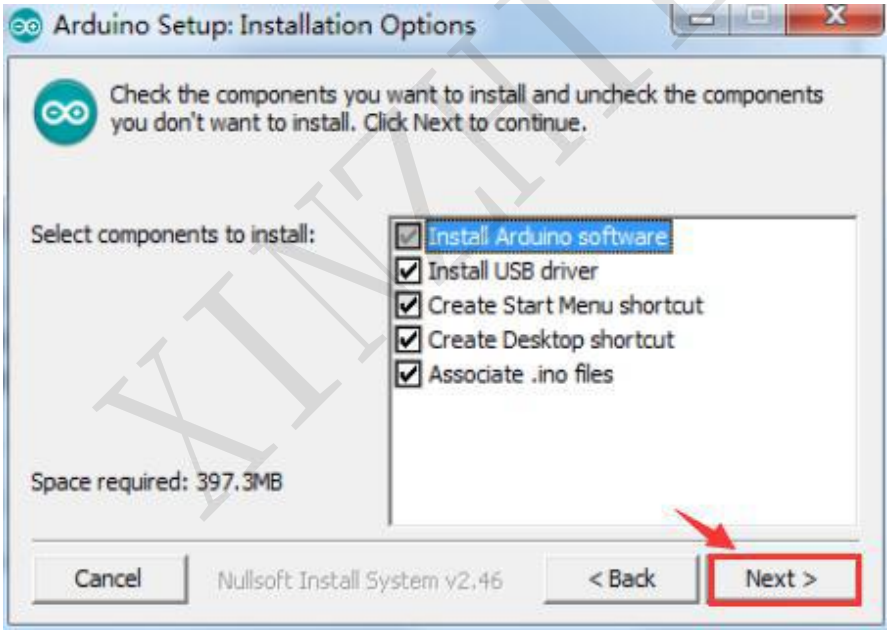



双击 exe 文件，出现如下界面，选择“I Agree（我同意）”。

XINZHILI

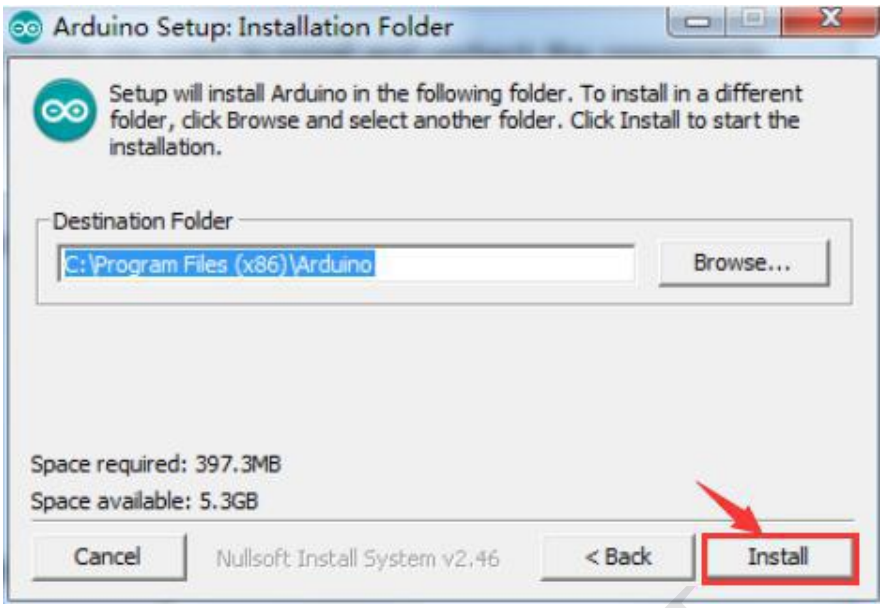



即可看到如下界面，选择“Next（下一步）”。

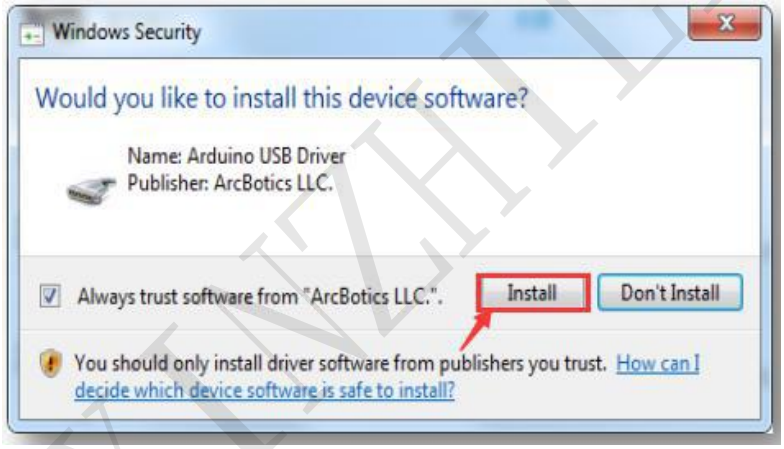



得到如下界面，按“Install（安装）”开始安装。

XINZHILI




最后出现如下界面，请选择“Install”以保证开发环境安装的正确性。

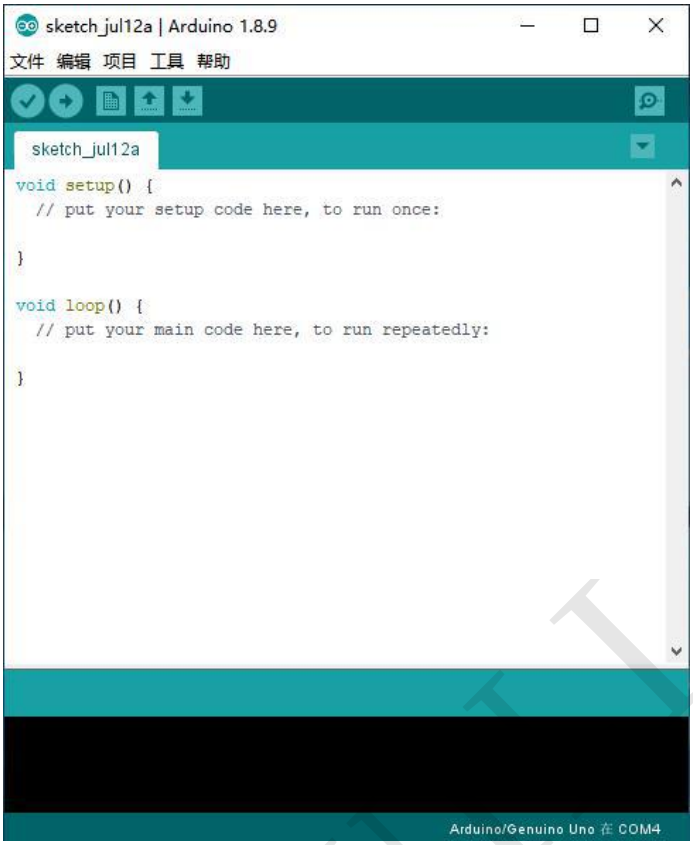



接下来，桌面上会出现以下图标。



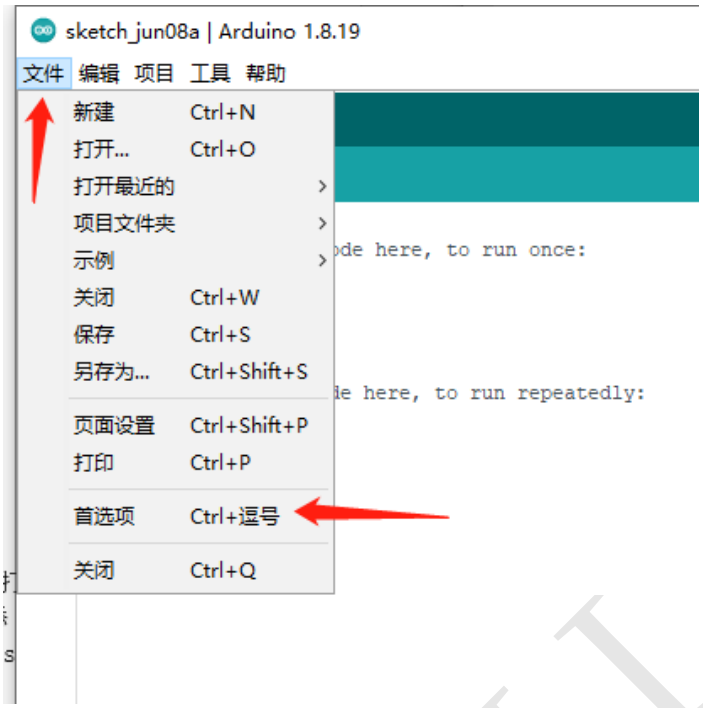

双击进入开发环境。

XINZHILI




我们这里用的主控是 ESP32-WROOM-32，所以需要安装 ESP32 插件，在上一步打开的 Arduino IDE,找到”文件->首选项”。在”附加开发板管理器网址:”中添加以下网址 https://dl.espressif.com/dl/package_esp32_index.json，如下图所示。

XINLI





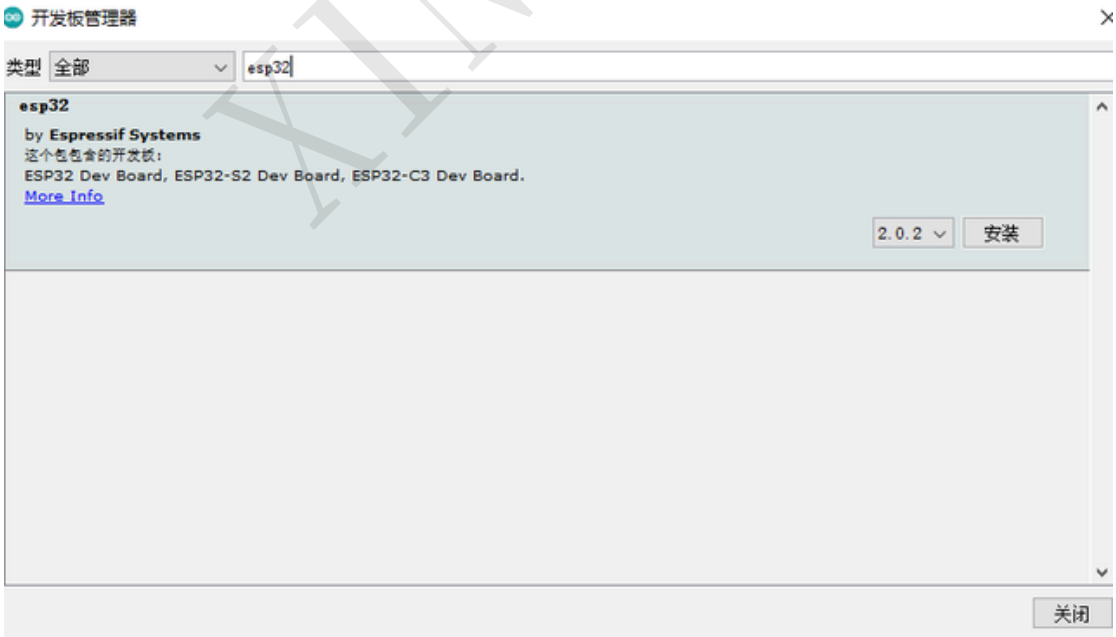


打开”工具->开发板->开发板管理器”

XINZHILI




搜索 esp32，选择版本 1.0.6 进行安装，如下图所示



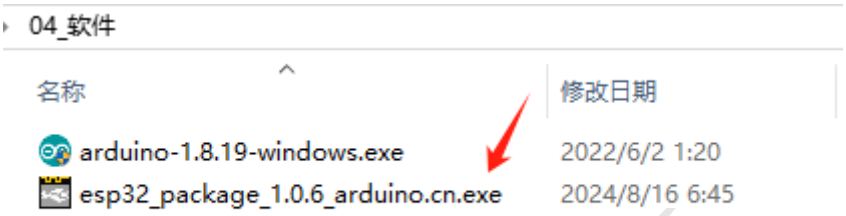

因为下载的时候需要连接 github,你可能需要多点击下载几次。安装成功后

XINZHILI

重启 Arduino IDE。

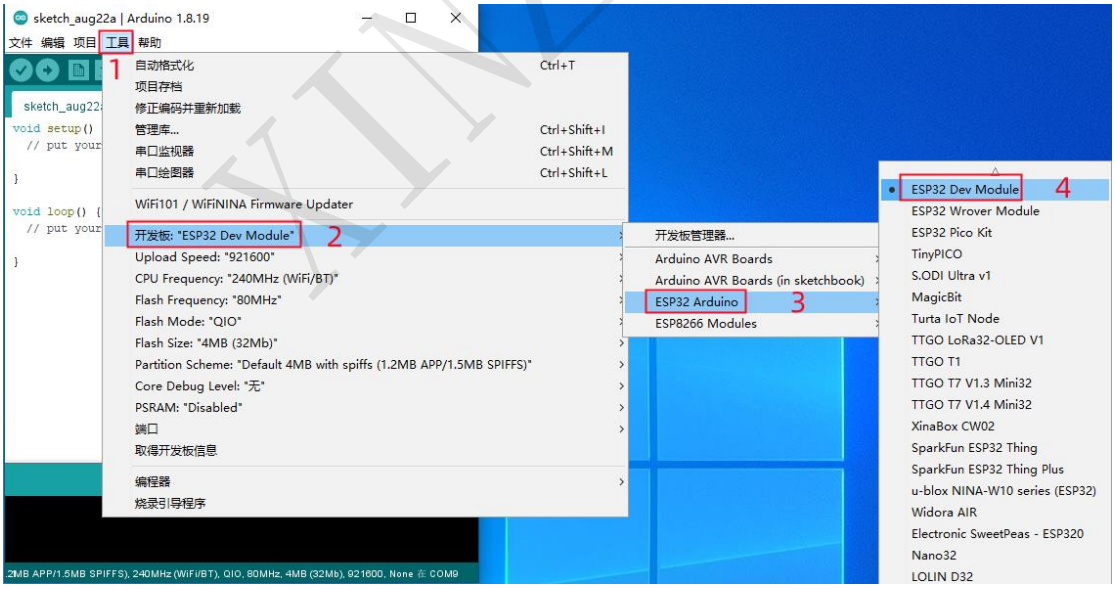
### 1.2.3 离线安装 ESP32

由于在线安装插件需要访问外网，网络可能不稳定，如果一直安装失败，则可以使用网盘资料里面的离线安装包，所在位置如下所示，下载后，双击运行即可。



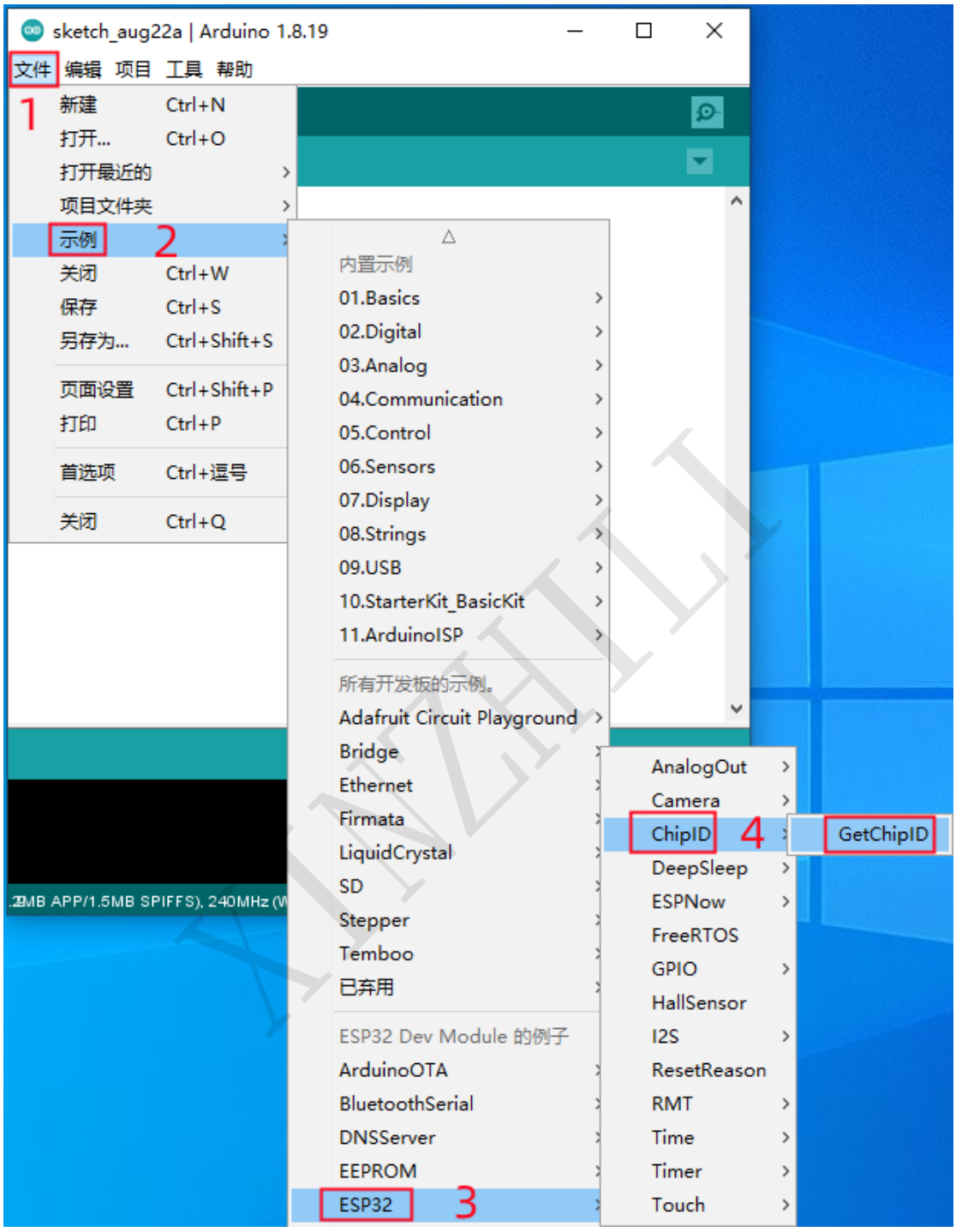

安装成功后重启 Arduino IDE。

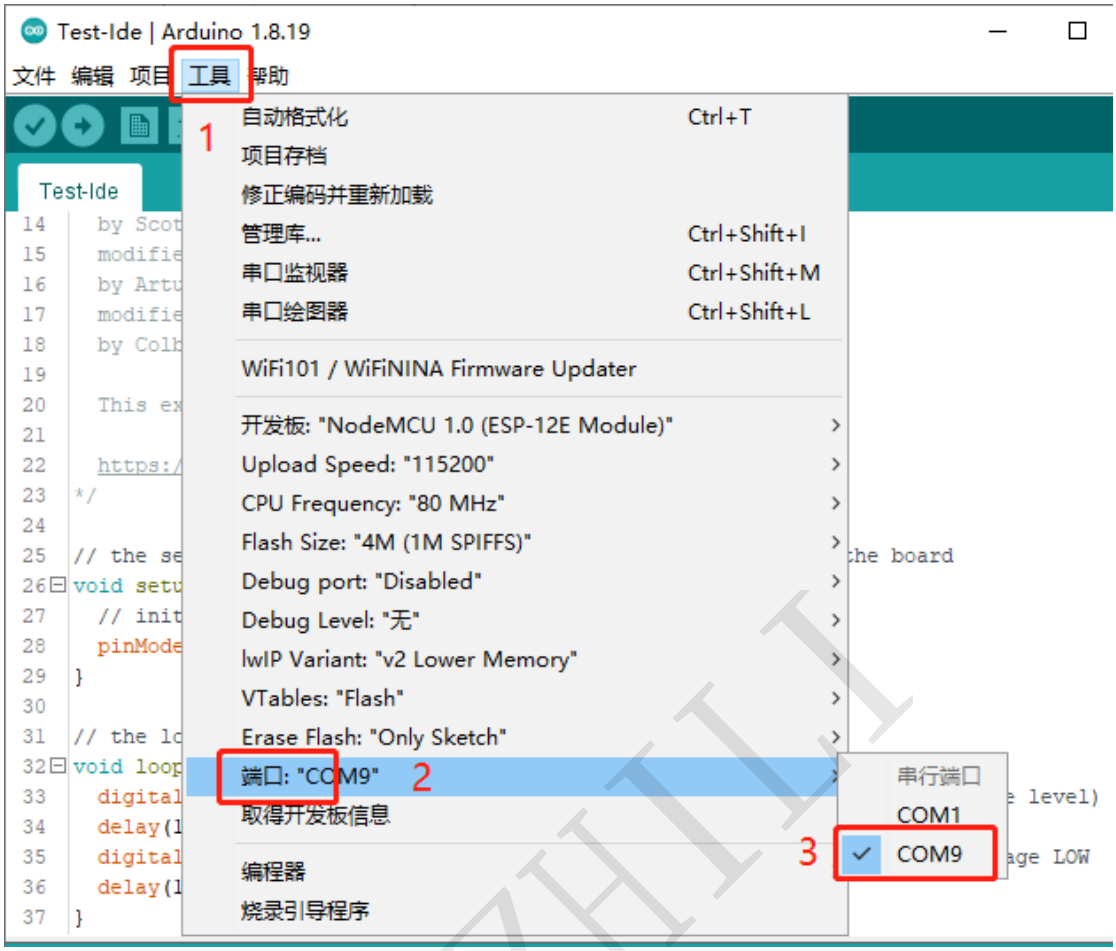
安装好 IDE 软件后，用 USB 数据线连接开发板和计算机的 USB 口。在 Arduino IDE 中选择 **ESP32 Dev Module** 作为开发板：




测试是否安装成功。依次点击 Arduino IDE 的“文件→示例→ESP32→ChipID→GetChipID”。在“工具->端口”选择“COM9”。(在 Arduino 1.8.9 的同一台计算机上，每个 ESP32 开发板都有不同的 COM 编号。 您应该

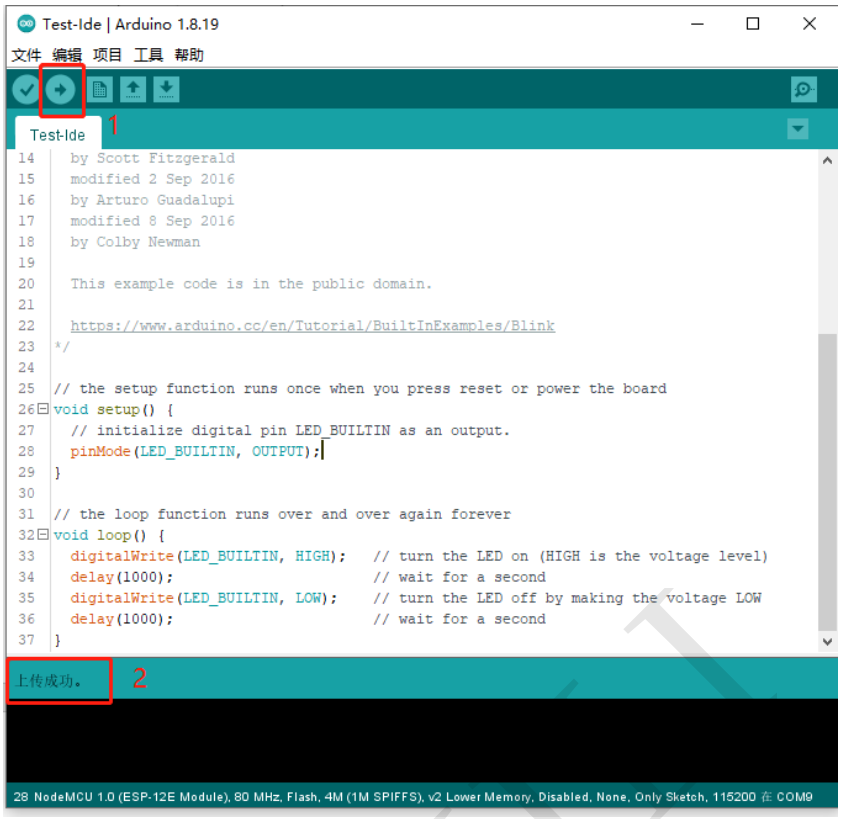
选择实际显示的 COM 号。）






点击下载按钮，将程序下载进开发板中的单片机。如下图所示。

XINZHILI



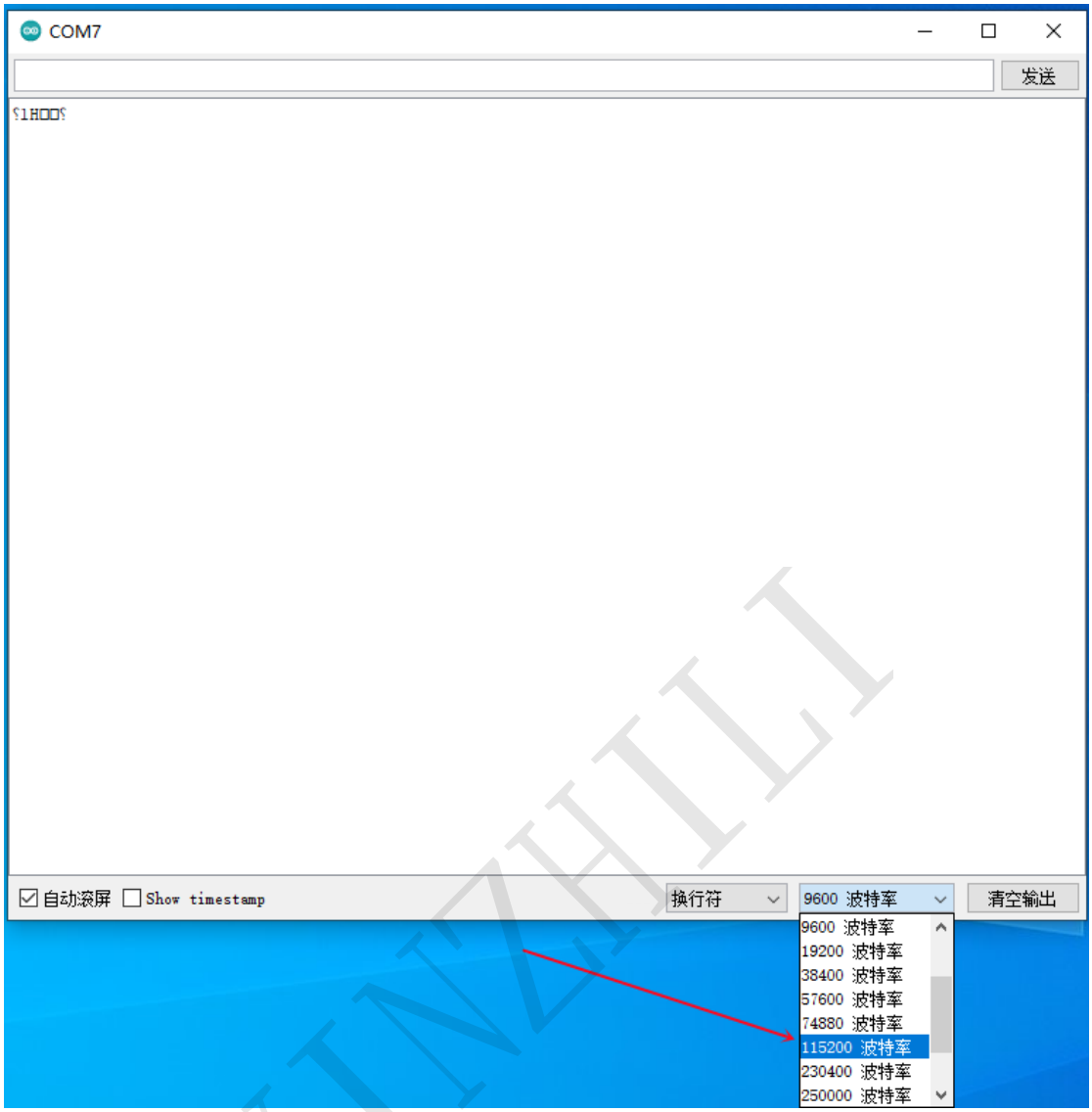

看到提示 2 中所示”上传成功”提示，则完成下载。

点击右上角的串口监视器按钮。如下图所示：

XINZHI




在弹出的串口监视器窗口中选择 115200 的波特率，如下图所示：



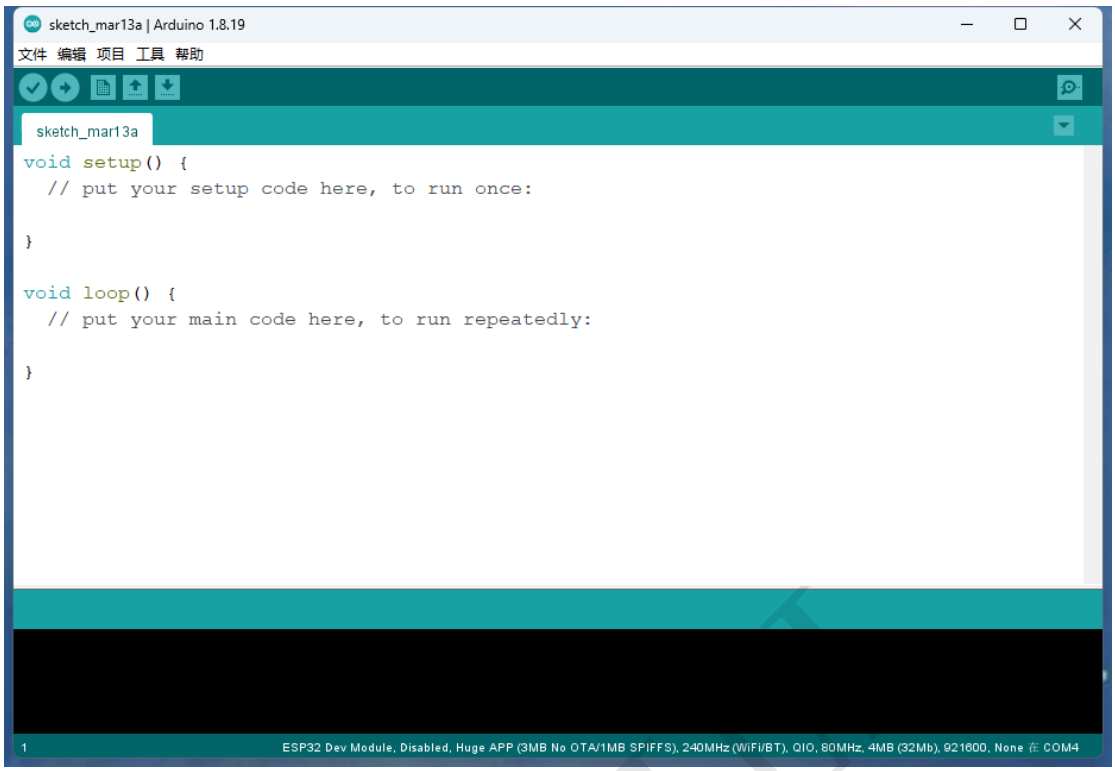

正常情况下可以看到窗口中正确显示芯片的 ID 信息。至此，Arduino IDE 软件安装成功。

### 1.2.4 添加 Arduino 库方法

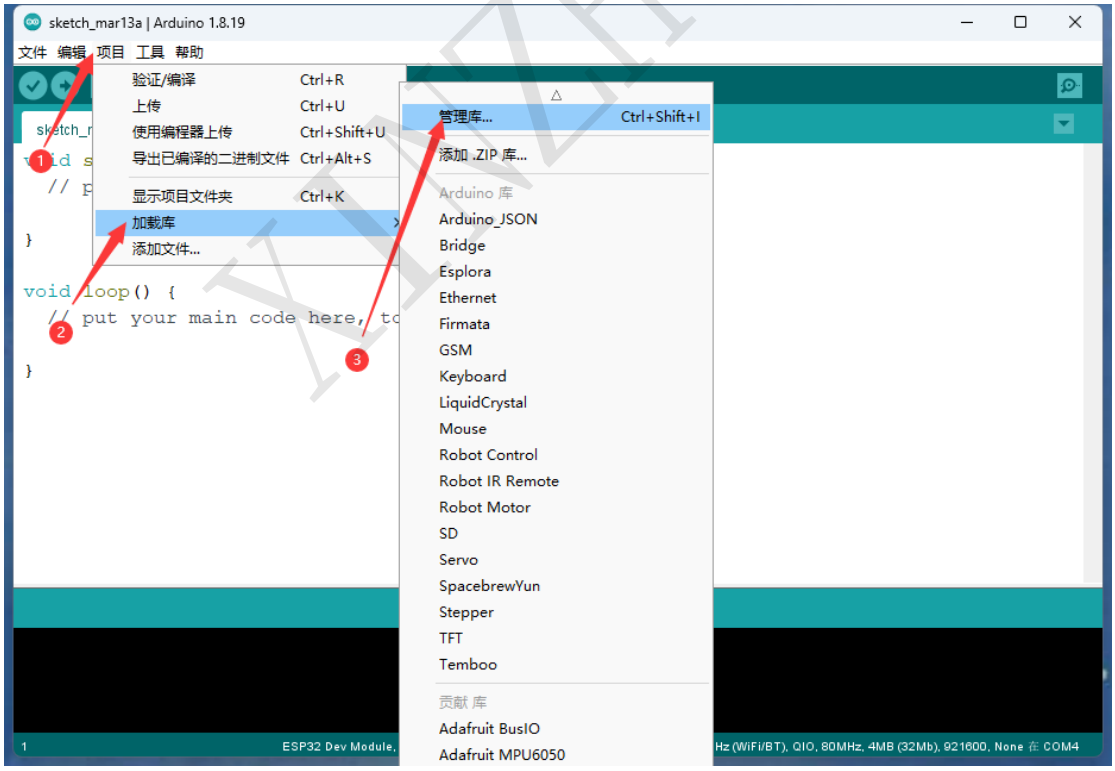
#### 1.2.4.1 在线安装（此安装方式需要网络）

打开 Arduino 软件

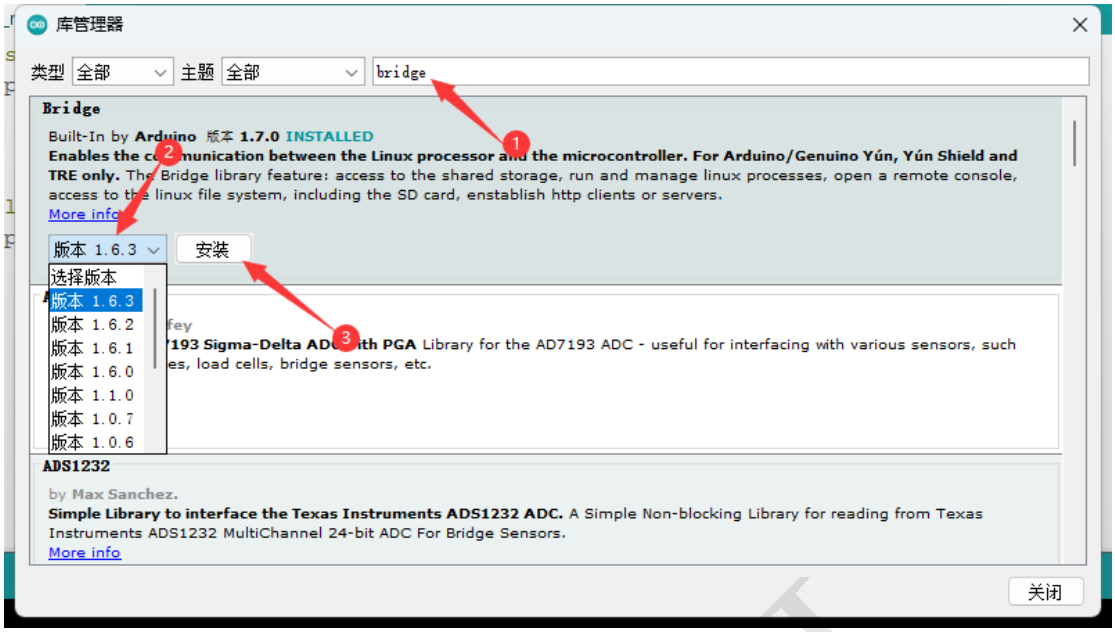
XINHUI




点击左上角“项目”>“加载库”>“管理库”

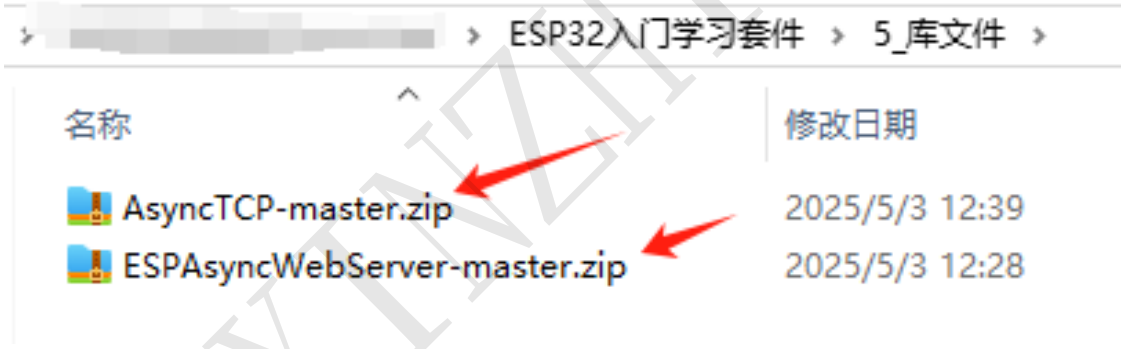



在搜索栏直接搜索需要安装的库，然后选择版本点击安装



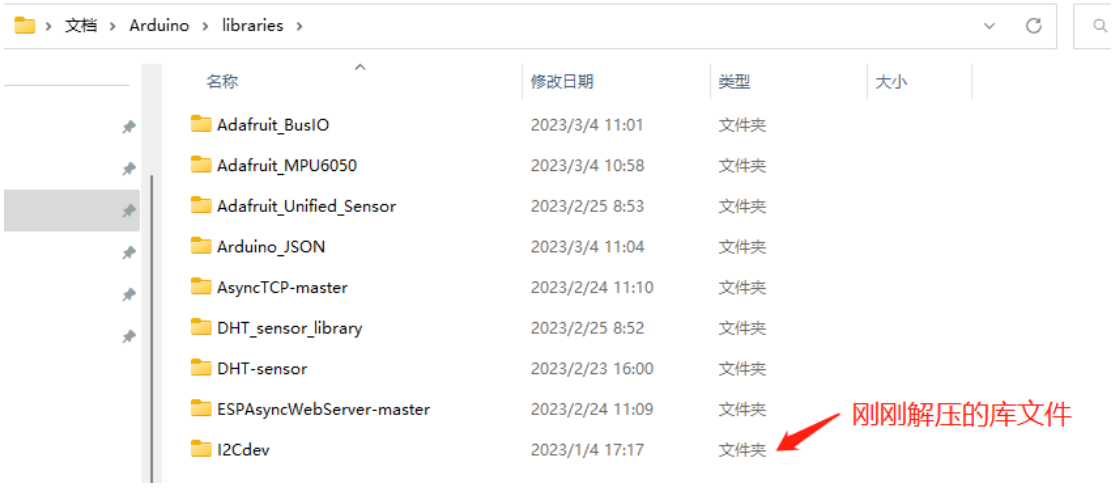

### 1.2.4.2 离线安装

进入资料包的库文件夹，解压.zip 库文件

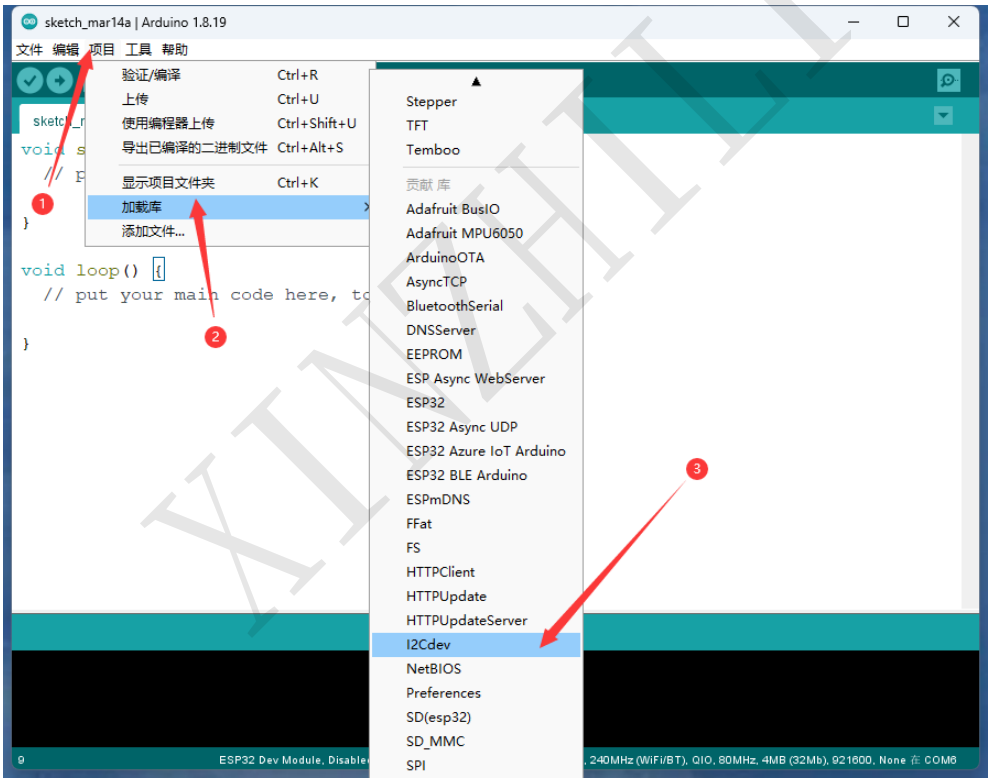



将解压后的库文件夹放入 Arduino 库文件存放地，默认安装的库文件存放路径为：C:/User/用户名/文档/Arduino/libraries，为了方便管理，如果没有 libraries 文件夹，请在路径下创建一个文件夹。

XINZHILI

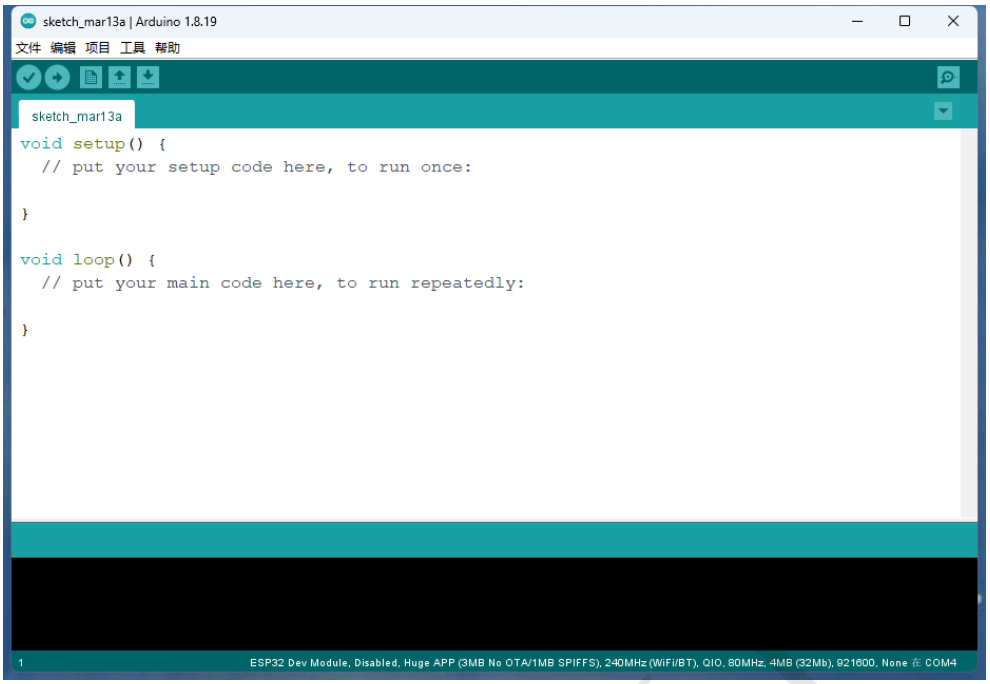



打开 Arduino IDE 软件，点击“项目”>“加载库”，查看贡献库下是否有我们添加的库，有即添加成功。

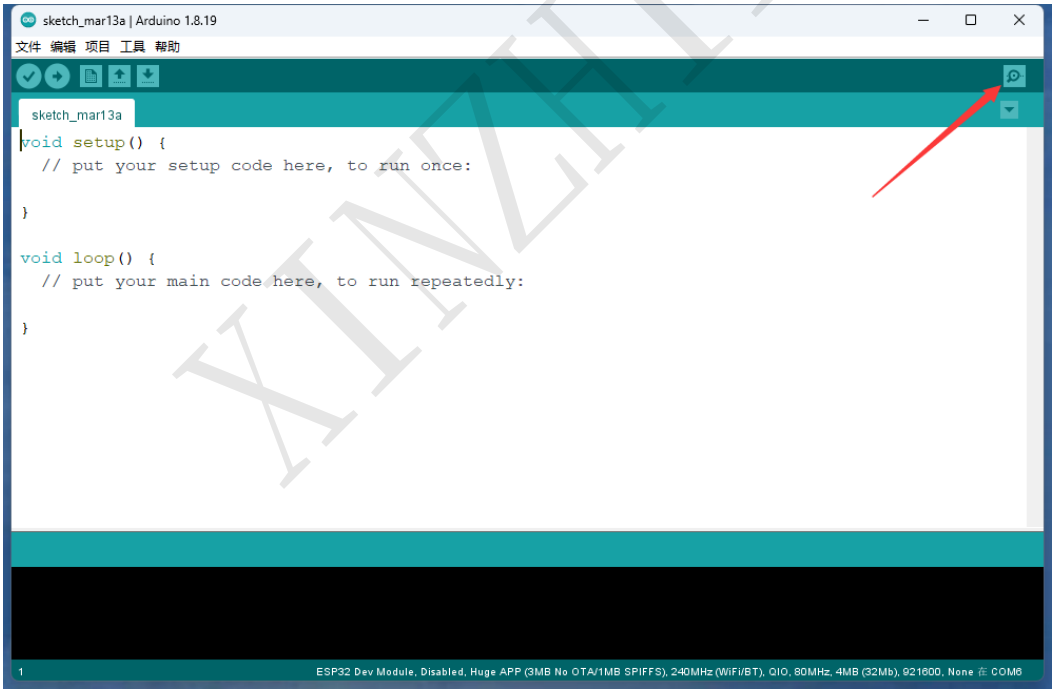



## 1.2.5 学习使用串口监视器

打开 Arduino 软件

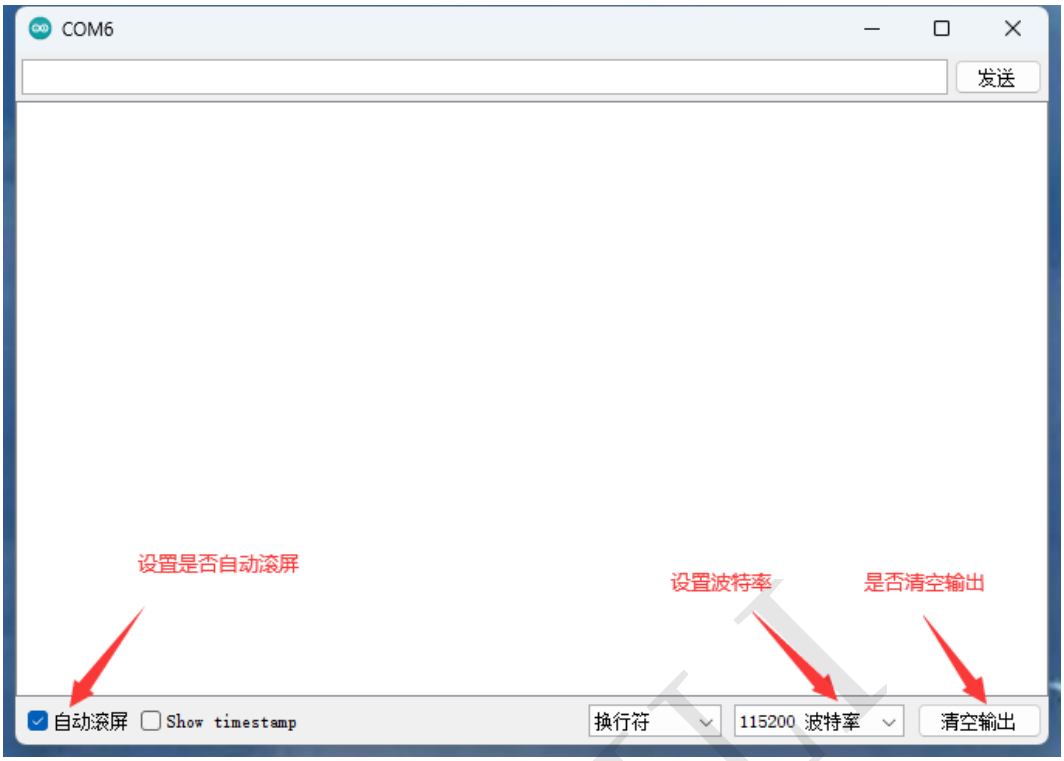



打开串口监视器




串口监视器功能介绍

XINZHILI




# 第1章 ESP32 简介

## 1.1 概述

ESP32 新手？从这里开始！ESP32 是 Espressif 开发的一系列低成本、低功耗片上系统（SoC ）微控制器，具有 Wi-Fi 和蓝牙无线功能以及双核处理器。如果您熟悉 ESP8266，那么 ESP32 就是它的后继者，具有许多新功能。



## 1.2 ESP32 规格

如果您想了解更多技术细节，可以查看以下 ESP32 的详细规格（来源：http://esp32.net/）。如需了解更多详情，请查阅数据表）：

■ **无线连接 WiFi**：150.0 Mbps 数据传输速率，带 HT40

■ **蓝牙**：BLE（低功耗蓝牙）和经典蓝牙

■ **处理器**：Tensilica Xtensa 双核 32 位 LX6 微处理器，运行频率为 160 或 240 MHz

■ **内存**：

- ROM: 448 KB（用于启动和核心功能）
- SRAM: 520 KB（用于数据和指令）
- RTC fas SRAM：8 KB（用于数据存储和 RTC 从深度睡眠模式启动时的主 CPU）
- RTC 慢速 SRAM：8KB（用于在深度睡眠模式下访问协处理器）
- eFuse：1 Kbit（其中 256 位用于系统（MAC 地址和芯片配置），其余 768 位保留给客户应用，包括闪存加密和芯片 ID）

■ **嵌入式闪存**：在 ESP32-D2WD 和 ESP32-PICO-D4 上，闪存通过 IO16、IO17、SD_CMD、SD_CLK、SD_DATA_0 和 SD_DATA_1 内部连接。

- 0 MiB（ESP32-D0WDQ6、ESP32-D0WD 和 ESP32-S0WD 筹码)
- 2 MiB（ESP32-D2WD 芯片）
- 4 MiB（ESP32-PICO-D4 SiP 模块）

■ **低功耗**：确保在深度睡眠等情况下仍能使用 ADC 转换。

■ **外设输入/输出**：

- 带 DMA 的外设接口，包括电容式触摸屏

XINZHILI

- 模数转换器（ADC）
- DAC（数模转换器）
- I²C（集成电路）
- UART（通用异步接收器/发送器）
- SPI（串行外设接口）
- I²S（集成芯片间声音）
- RMII（精简媒体独立接口）
- PWM（脉宽调制）

■ **安全：**AES 和 SSL/TLS 硬件加速器

## 1.3 组件介绍

ESP32 指的是裸露的 ESP32 芯片。不过，"ESP32 "一词也用于指 ESP32 开发板。使用 ESP32 裸芯片并不容易，也不实用，尤其是在学习、测试和原型设计时。大多数情况下，您需要使用 ESP32 开发板。

我们将使用 ESP32 DEVKIT V1 板作为参考。下图显示的是 ESP32 DEVKIT V1 板，该版本有 30 个 GPIO 引脚。

XINZHILI

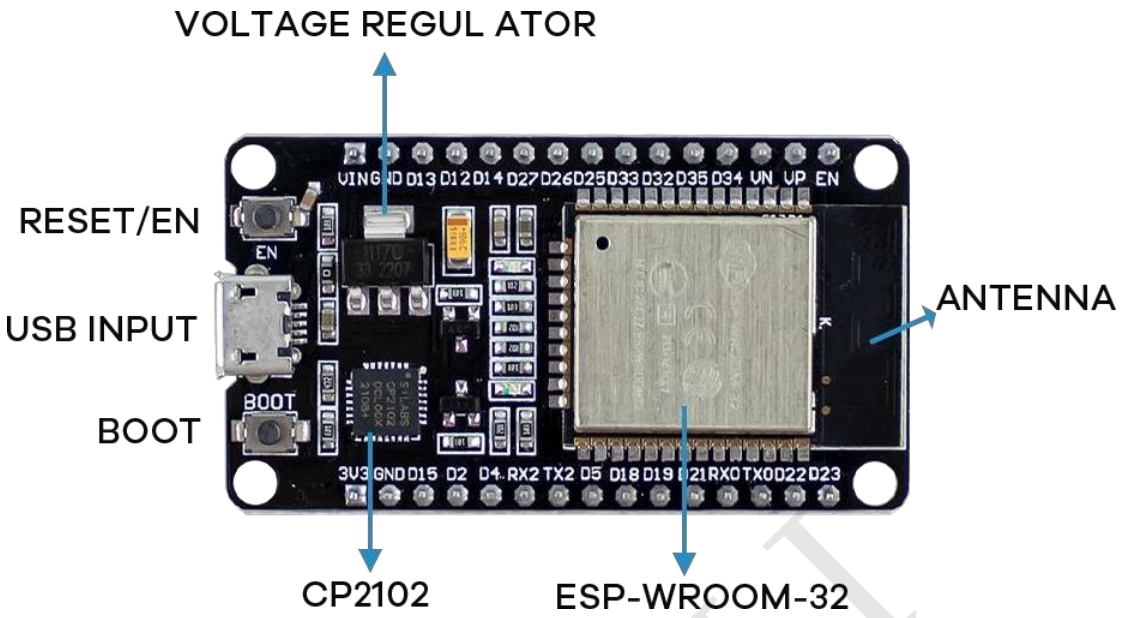



下表概括介绍了 ESP32 DEVKIT V1 DOIT 板的功能和规格：

| | |
|---|---|
| 内核数量 | 2（双核） |
| 无线网络 | 2.4 千兆赫，高达 150 Mbits/s |
| 蓝牙 | BLE（低功耗蓝牙）和传统蓝牙 |
| 架构 | 32 位 |
| 时钟频率 | 高达 240 MHz |
| 内存 | 512 KB |
| 引脚 | 30（视型号而定） |
| 外围设备 | 电容式触摸、ADC（模数转换器）、DAC（数模转换器）、I2C（集成电路间）、UART（通用异步接收器/发送器）、CAN 2.0（控制器区域网）、SPI（串行外设接口）、I2S（集成电路间声音）、RMII（简化的串行外设接口）、SPI（串行外设接口）、SPI（串行外设接口）、I2S（集成电路间声音）、RMII（简化的串行外设接口媒体独立接口）、PWM（脉宽调制）等。 |
| 内置按钮 | 复位和启动按钮 |
| 内置 LED | 内置蓝色 LED，连接到 GPIO2；内置红色 LED<br>，显示电路板正在供电 |

XINLI

| | |
|---|---|
| USB 转 UART 芯片 | CH340 |

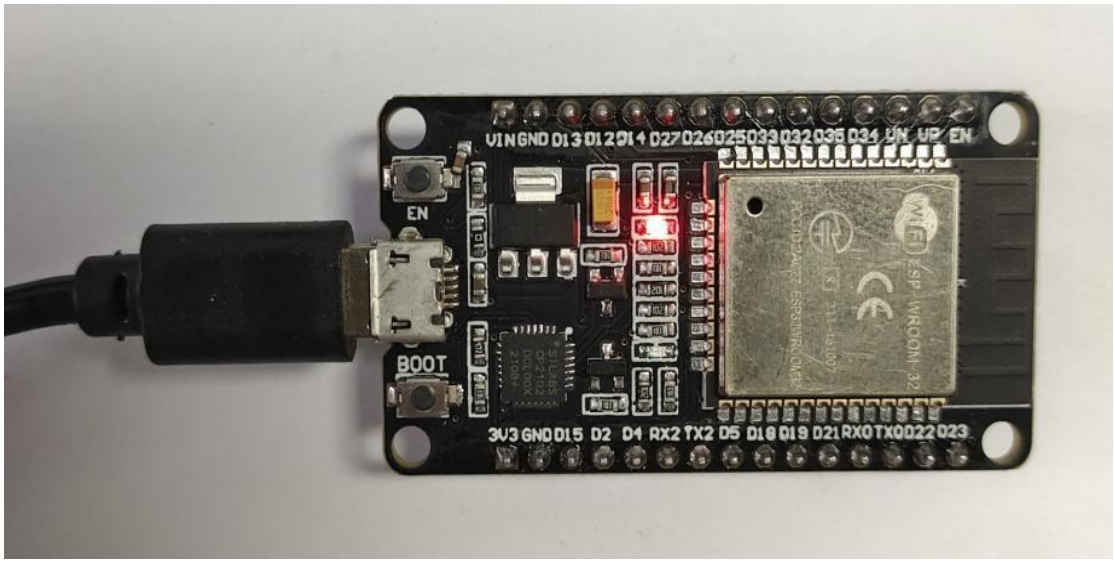



它配有微型 USB 接口，可用于将电路板连接到电脑，上传代码或供电。

它使用 CH340 芯片（USB 转 UART），通过 COM 端口使用串行接口与计算机通信。另一种常用的芯片是 CP2102。请检查电路板上的 USB 转 UART 芯片转换器，因为您需要安装所需的驱动程序，这样计算机才能与电路板通信（本指南稍后将提供更多相关信息）。

该电路板还配有一个 RESET 按钮（可能标有 EN），用于重新启动电路板，以及一个 BOOT 按钮，用于将电路板置于闪烁模式（可用于接收代码）。请注意，有些电路板可能没有 BOOT 按钮。

它还内置一个蓝色 LED，内部连接到 GPIO 2。这个 LED 在调试时非常有用，可以提供某种可视化的物理输出。另外还有一个红色 LED 灯，在为电路板供电时亮起。



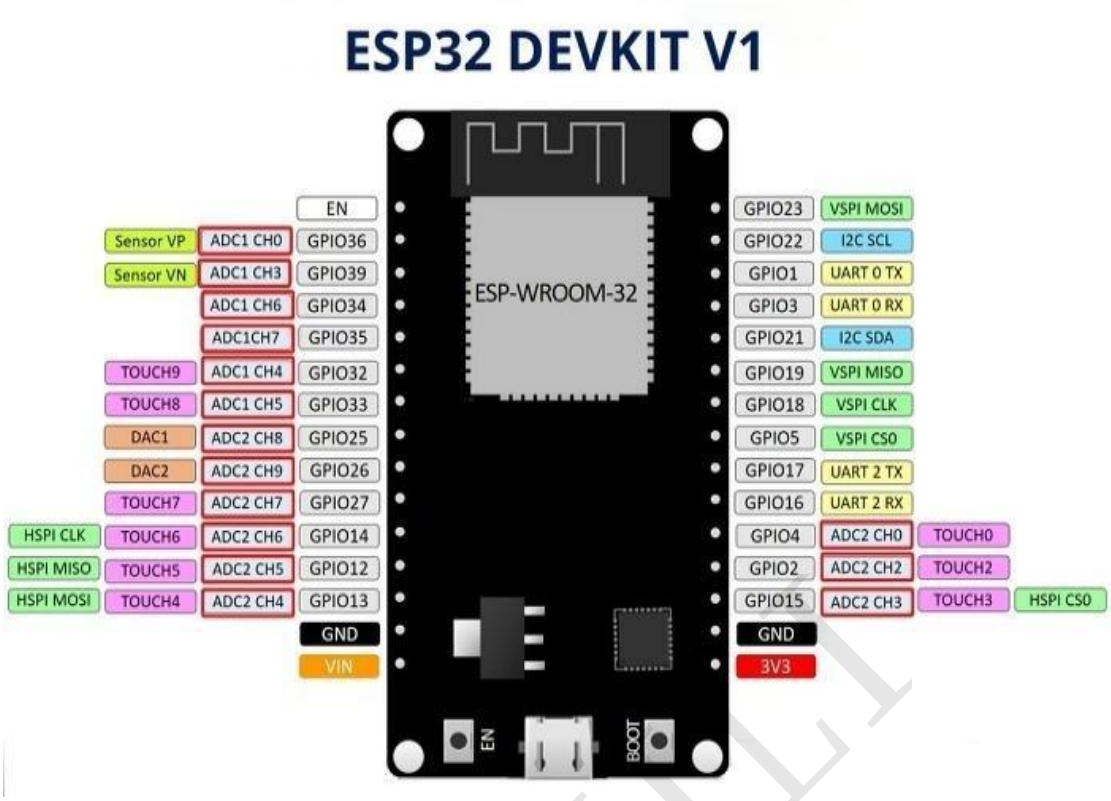

## 1.4 ESP32 引脚布局

ESP32 外围设备包括

- 18 个模数转换器 (ADC) 通道
- 3 个 SPI 接口
- 3 个 UART 接口
- 2 个 I2C 接口
- 16 个 PWM 输出通道
- 2 个数模转换器（DAC）
- 2 个 I2S 接口
- 10 个电容感应 GPIO ADC（模数转换器）和 DAC（数模转换器）功能分配给特定的静态引脚。不过，您可以决定哪些引脚是 UART、I2C、SPI、PWM 等，只需在代码中进行分配即可。这得益于 ESP32 芯片的多路复用功能。

虽然可以在软件上定义引脚属性，但默认情况下会分配引脚，如下图所示

XINZHILI




此外，有些引脚还具有特定的功能，适合或不适合特定的项目。下表显示了哪些引脚最适合用作输入和输出，哪些引脚需要谨慎使用。

绿色标注的引脚可以使用。黄色标注的引脚可以使用，但需要注意，因为它们在启动时可能会出现意外行为。红色标注的引脚不建议用作输入或输出。

| GPIO | 输入 | 输出 | 说明 |
|---|---|---|---|
| 0 | 拉起 | OK | 启动时输出 PWM 信号，必须为低电平才能进入闪烁模式 |
| 1 | TX 引脚 | OK | 启动时的调试输出 |
| 2 | OK | OK | 连接至板载 LED，必须保持浮动或低电平状态才能进入闪烁模式 |
| 3 | OK | RX 引脚 | 启动时为高电平 |
| 4 | OK | OK | |
| 5 | OK | OK | 启动时输出 PWM 信号，绑扎引脚 |
| 12 | OK | OK | 高电平时启动失败，捆绑引脚 |
| 13 | OK | OK | |
| 14 | OK | OK | 启动时输出 PWM 信号 |
| 15 | OK | OK | 启动时输出 PWM 信号，绑扎引脚 |
| 16 | OK | OK | |
| 17 | OK | OK | |
| 18 | OK | OK | |

| | | | |
|---|---|---|---|
| 19 | OK | OK | |
| 14 | OK | OK | |
| 15 | OK | OK | |
| 16 | OK | OK | |
| 17 | OK | OK | |
| 18 | OK | OK | |
| 19 | OK | OK | |
| 21 | OK | OK | |
| 22 | OK | OK | |
| 23 | OK | OK | |
| 25 | OK | OK | |
| 26 | OK | OK | |
| 27 | OK | OK | |
| 32 | OK | OK | |
| 33 | OK | OK | |
| 34 | OK | | 仅输入 |
| 35 | OK | | 仅输入 |
| 36 | OK | | 仅输入 |
| 39 | OK | | 仅输入 |

继续阅读，了解有关 ESP32 GPIO 及其功能的更多细节和深入分析。

### 1.4.1 仅输入引脚

GPIO 34 至 39 为 GPI，仅为输入引脚。这些引脚没有内部上拉或下拉电阻。它们不能用作输出，因此只能用作输入：

- GPIO 34
- GPIO 35
- GPIO 36
- GPIO 39

### 1.4.2 集成在 ESP-WROOM-32 上的 SPI 闪存

在某些 ESP32 开发板中，GPIO 6 至 GPIO 11 是暴露在外的。但是，这些引脚与 ESP-WROOM-32 芯片上的集成 SPI 闪存相连，不建议用于其他用途。因此，请不要在您的项目中使用这些引脚：

- GPIO 6（SCK/CLK）
- GPI 7（SDO/SD0）

XINZHILI

- GPI 8（SDI/SD1）
- GPI 9（SHD/SD2）
- GPI 10（SWP/SD3）
- GPI 11（CSC/CMD）

### 1.4.3 电容式触摸 GPIO

ESP32 内部有 10 个电容式触摸传感器。这些传感器可以感知任何带电荷的物体（如人体皮肤）的变化。因此，它们可以检测到手指触摸 GPIO 时引起的变化。这些引脚可以电容式触摸引脚可轻松集成到电容式焊盘中，取代机械按钮。电容式触摸引脚还可用于将 ESP32 从深度睡眠中唤醒。

这些内部触摸传感器与这些 GPIO 相连：

- T0（GPIO 4）
- T1（GPIO 0）
- T2（GPIO 2）
- T3（GPIO 15）
- T4（GPIO 13）
- T5（GPIO 12）
- T6（GPIO 14）
- T7（GPIO 27）
- T8（GPIO 33）
- T9（GPIO 32）

### 1.4.4 模数转换器 (ADC)

ESP32 有 18 x 12 位 ADC 输入通道（而 ESP8266 只有 1 x 10 位 ADC）。这些 GPIO 可用作 ADC 和相应的通道：

XINZHILI

- adc1_ch0（GPIO 36）
- adc1_ch1（GPIO 37）
- adc1_ch2（GPIO 38）
- adc1_ch3（GPIO 39）
- adc1_ch4（GPIO 32）
- adc1_ch5（GPIO 33）
- adc1_ch6（GPIO 34）
- adc1_ch7（GPIO 35）
- adc2_ch0（GPIO 4）
- adc2_ch1（GPIO 0）
- adc2_ch2（GPIO 2）
- adc2_ch3（GPIO 15）
- adc2_ch4（GPIO 13）
- adc2_ch5（GPIO 12）
- ADC2_CH6（GPIO 14）
- adc2_ch7（GPIO 27）
- adc2_ch8（GPIO 25）
- adc2_ch9（GPIO 26）

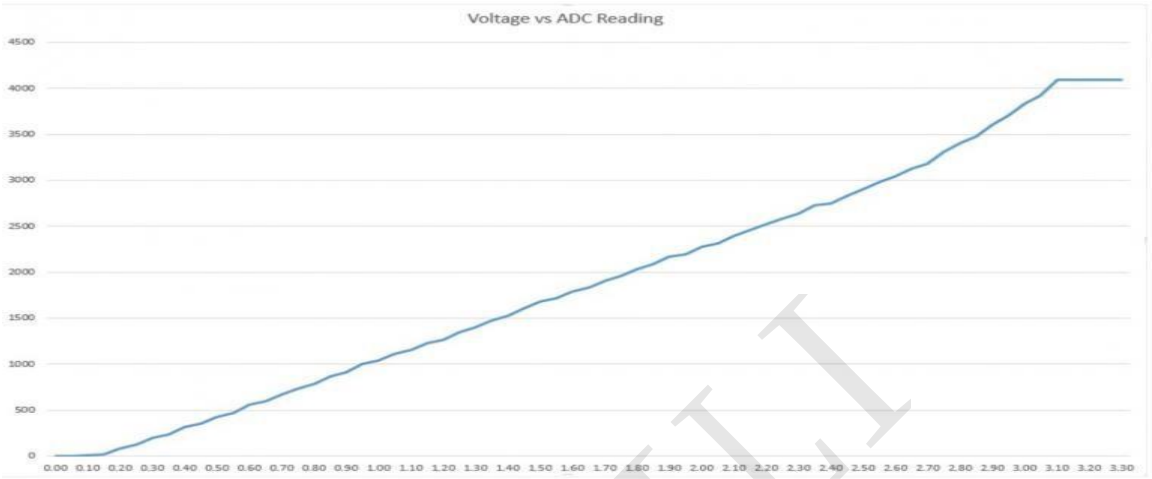
注意：使用 WIFI 时不能使用 ADC2 引脚。因此，如果您在使用 WIFI 时无法从 ADC2 GPIO 获取数值，可以考虑使用 ADC1 GPIO 代替。这应该能解决您的问题。

ADC 输入通道的分辨率为 12 位。这意味着可以获得 0 至 4095 的模拟读

XINZHILI

数，其中 0 对应 0V，4095 对应 3.3V.您还可以在代码和 ADC 范围上设置通道的分辨率。

ESP32 ADC 引脚不是线性的。您可能无法区分 0V 和 0.1V，或 3.2V 和 3.3V。在使用 ADC 引脚时，您需要牢记这一点。您将得到类似下图所示的行为。




XINZHILI

### 1.4.5 数模转换器 (DAC)

ESP32 上有 2 x 8 位 DAC 通道，用于将数字信号转换为模拟电压信号输出。这些就是 DAC 通道：

- DAC1 （GPIO25）
- DAC2 （GPIO26）

### 1.4.6 RTC GPIO

ESP32 支持 RTC GPIO。路由到 RTC 的 GPIO 这些 RTC GPIO 可在 ESP32 处于深度睡眠状态时从深度睡眠状态唤醒 ESP32。这些 RTC GPIO 可用于在超低功耗时将 ESP32 从深度睡眠中唤醒。

电源 (ULP) 协处理器正在运行。以下 GPIO 可用作外部唤醒源。

- RTC_GPIO0（GPIO36）
- RTC_GPIO3（GPIO39）
- RTC_GPIO4（GPIO34）
- RTC_GPIO5（GPIO35）
- RTC_GPIO6（GPIO25）
- RTC_GPIO7（GPIO26）
- rtc_gpio8 (gpio33)
- rtc_gpio9 (gpio32)
- RTC_GPIO10（GPIO4）
- RTC_GPIO11（GPIO0）
- RTC_GPIO12（GPIO2）
- RTC_GPIO13 (GPIO15)
- RTC_GPIO14 (GPIO13)
- RTC_GPIO15（GPIO12）
- rtc_gpio16 (gpio14)
- RTC_GPIO17 (GPIO27)

### 1.4.7 PWM

ESP32 LED PWM 控制器有 16 个独立通道，可配置为产生不同性质的 PWM 信号。所有可以作为输出的引脚都可以用作 PWM 引脚（GPIO 34 至 39 不能产生 PWM）。

要设置 PWM 信号，需要在代码中定义这些参数：

- 信号的频率
- 占空比

XINZHILI

- PWM 通道
- 要输出信号的 GPIO

### 1.4.8 I2C

ESP32 有两个 I2C 通道，任何引脚都可以设置为 SDA 或 SCL。将 ESP32 与 Arduino IDE 结合使用时，默认的 I2C 引脚为

- GPIO 21（SDA）
- GPIO 22（SCL）

如果您想在使用导线库时使用其他引脚，只需调用

```
Wire.begin(SDA, SCL);
```

### 1.4.9 SPI

默认情况下，SPI 的引脚映射为

| SPI | MOSI | MISO | CLK | CS |
|---|---|---|---|---|
| VSPI | GPIO 23 | GPIO 19 | GPIO 18 | GPIO 5 |
| HSPI | GPIO 13 | GPIO 12 | GPIO 14 | GPIO 15 |

### 1.4.10 中断

所有 GPIO 均可配置为中断。

XINZHILI

### 1.4.11 捆绑引脚

ESP32 芯片有以下捆绑引脚：

- GPIO 0（必须为低电平才能进入启动模式）
- GPIO 2（启动时必须为浮动或低电平）
- GPIO 4
- GPIO 5（启动时必须为高电平）
- GPIO 12（启动时必须为低电平）
- GPIO 15（启动时必须为高电平）

这些引脚用于将 ESP32 置入引导程序或闪存模式。在大多数内置 USB/串口的开发板上，您无需担心这些引脚的状态。开发板会将引脚置于闪烁或启动模式的正确状态。有关 ESP32 启动模式选择的更多信息，请点击此处。

但是，如果您有外设连接到这些引脚，您在尝试上传新代码、用新固件闪烁 ESP32 或重置电路板时可能会遇到困难。如果您有一些外设连接到捆绑引脚，而您在上传代码或闪烁 ESP32 时遇到困难，可能是因为这些外设阻止 ESP32 进入正确的模式。阅读

请参阅启动模式选择文档，以获得正确的指导。重置、闪存或启动后，这些引脚都能正常工作。

### 1.4.12 启动时为高电平的引脚

某些 GPIO 在启动或复位时会将状态变为高电平或输出 PWM 信号。这意味着，如果您将输出连接到这些 GPIO，那么当 ESP32 复位或启动时，您可能会得到意想不到的结果。

- GPIO 1
- GPIO 3
- GPIO 5

XINZHILI

- GPIO 6 至 GPIO 11（连接至 ESP32 集成的 SPI 闪存，不建议使用）。
- GPIO 14
- GPIO 15

### 1.4.13 启用 (EN)

使能 (EN) 是 3.3V 稳压器的使能引脚。该引脚被上拉，因此将其接地可禁用 3.3V 稳压器。这意味着您可以将该引脚连接到按钮上，以重启 ESP32 等设备。

### 1.4.14 GPIO 所消耗的电流

根据 ESP32 数据表中的 ″推荐工作条件 ″部分，每个 GPIO 的绝对最大电流为 40mA。

### 1.4.15 ESP32 内置霍尔效应传感器

ESP32 还内置霍尔效应传感器，可检测周围磁场的变化。

# 第2章 ESP32 输入和输出

## 2.1 概述

在本课程中,我们将学习如何使用 ESP32 和 Arduino IDE 读取数字输入（如按钮开关）和控制数字输出（如 LED）。

## 2.2 所需组件

- ESP32 开发板

XINZHILI

## 2.3 预备知识

### 2.3.1 ESP32 控制数字输出

首先，需要将要控制的 GPIO 设置为 OUTPUT。使用 pinMode() 函数如下：

```
pinMode(GPIO, OUTPUT);
```

要控制数字输出，只需使用 digitalWrite() 函数，该函数的参数包括 GPIO（整数）和状态（高或低）。

```
digitalWrite(GPIO, STATE);
```

除了 GPIO 6 至 11（连接到集成 SPI 闪存）和 GPIO 34、35、36 和 39（仅输入 GPIO）外，所有 GPIO 均可用作输出；

## 2.4 *ESP32* 读取数字输入

首先，使用 pinMode() 函数将要读取的 GPIO 设置为 INPUT，如下所示：

```
pinMode(GPIO, INPUT);
```

要读取数字输入（如按钮），需要使用 digitalRead() 函数，该函数接受 GPIO（整数）作为参数。

```
digitalRead(GPIO);
```

GPIO 6 至 11（连接至集成 SPI 闪存）外，所有 ESP32 GPIO 均可用作输入。

### 2.4.1 项目实例

为了向您展示如何使用数字输入和数字输出，我们将用一个按钮和一个 LED 来构建一个简单的项目示例。如下图所示，我们将读取按钮的状态并相应点亮 LED。




## 2.5 所需部件

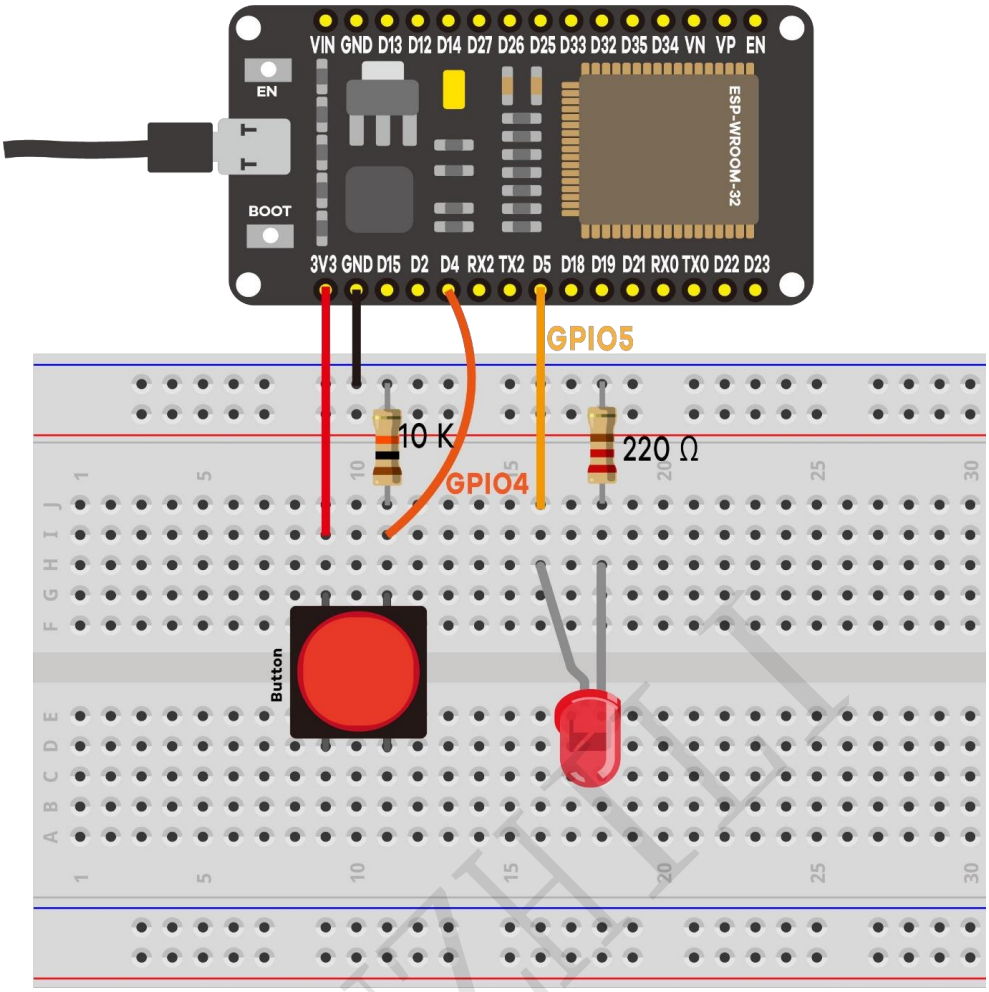
下面是构建电路所需的部件清单：

- esp32 devkit v1
- 5 毫米 LED
- 220 欧姆电阻
- 按钮
- 10k 欧姆电阻
- 面包板
- 跳线

## 2.6 示意图

在继续之前，您需要组装一个带有 LED 和按钮的电路。我们将把 LED 连接到 ，把按钮连接到 。

XINZHILI




## 2.7 代码

在 arduino IDE 中打开资料包中的“**代码 --> Project_1_ESP32_Inputs_Outputs**”文件夹中的 Project_1_ESP32_Inputs_Outputs.ino。

```
// set pin numbers
const int buttonPin = 4; // the number of the pushbutton pin
const int ledPin = 5; // the number of the LED pin
// variable for storing the pushbutton status
int buttonState = 0;

void setup() {
    Serial.begin(115200);
    pinMode(buttonPin, INPUT); // initialize the pushbutton pin as an input
    pinMode(ledPin, OUTPUT); // initialize the LED pin as an output
}
```

```
void loop() {
    // read the state of the pushbutton value
    buttonState = digitalRead(buttonPin);
    Serial.println(buttonState);
    // check if the pushbutton is pressed.
    // if it is, the buttonState is HIGH
    if (buttonState == HIGH) {
        // turn LED on
        digitalWrite(ledPin, HIGH);
    } else {
        // turn LED off
        digitalWrite(ledPin, LOW);
    }
}
```

### 2.7.1 代码如何工作

在下面两行中，您将创建变量来分配引脚：

```
const int buttonPin = 4; // the number of the pushbutton pin
const int ledPin = 5;
```

按键连接到 GPIO4，LED 连接到 GPIO5。当在 Arduino IDE 中使用 ESP32，4 对应 GPIO4，5 对应 GPIO5。

接下来，创建一个变量来保存按钮状态。默认状态为 0（未按下）。

```
int buttonState = 0;
```

在 setup()函数中，你需要将按钮初始化为 输入模式（INPUT），将 LED 初始化为 输出模式（OUTPUT）。为此，你需要使用 pinMode() 函数，该函数接受两个参数：

（1）引脚编号（你指定的引脚，例如 BUTTON_PIN 或 LED_PIN）

（2）模式（INPUT 或 OUTPUT，用于定义引脚的功能方向）。或 OUTPUT。

```
pinMode(buttonPin, INPUT); // initialize the pushbutton pin as an input
pinMode(ledPin, OUTPUT);
```

在 loop() 中，您将读取按钮状态并相应地设置 LED。下一行将读取按钮状

态并保存到 buttonState 变量中。正如我们之前看到的，你需要使用 digitalRead()函数。

```
buttonState = digitalRead(buttonPin);
```

下面的 if 语句检查按钮状态是否为 HIGH。如果是，则使用 digitalWrite()函数打开 LED，该函数接受 ledPin 和 HIGH 状态作为参数。

```
if (buttonState == HIGH) {
    // turn LED on
    digitalWrite(ledPin, HIGH);
}
```

如果按钮状态不是 HIGH，则 LED 关闭。只需将 LOW 设置为 digitalWrite()函数的第二个参数即

```
else {
    // turn LED off
    digitalWrite(ledPin, LOW);
}
```

### 2.7.2 上传代码

点击上传按钮之前，请转到“**工具 > 开发板**”，然后选择电路板：DOIT ESP32 DEVKIT V1。

转到“**工具 > 端口**”，选择 ESP32 所连接的 COM 端口。然后按下“**上传**”按钮，等待 "Done uploading（**上传成功**）"信息。

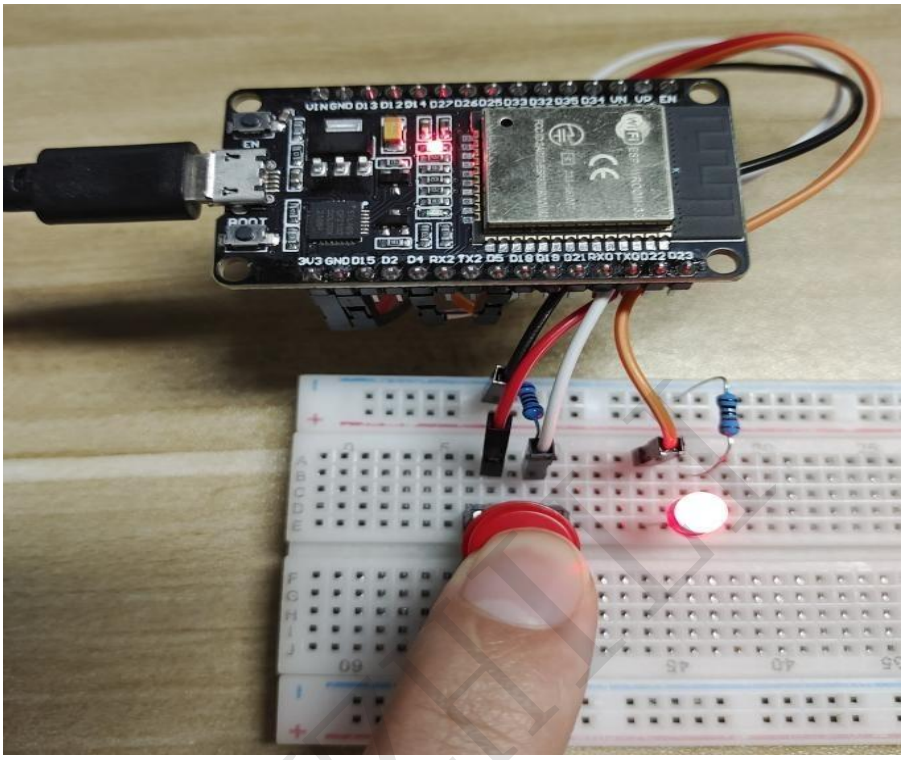


注意：如果在调试窗口中看到大量点状提示（正在连接……）并显示“连接 ESP32 失败：等待数据包头超时”的错误信息，这意味着当点开始出现后，您需要按下 ESP32 板载的 BOOT 按钮。

XINZHILI

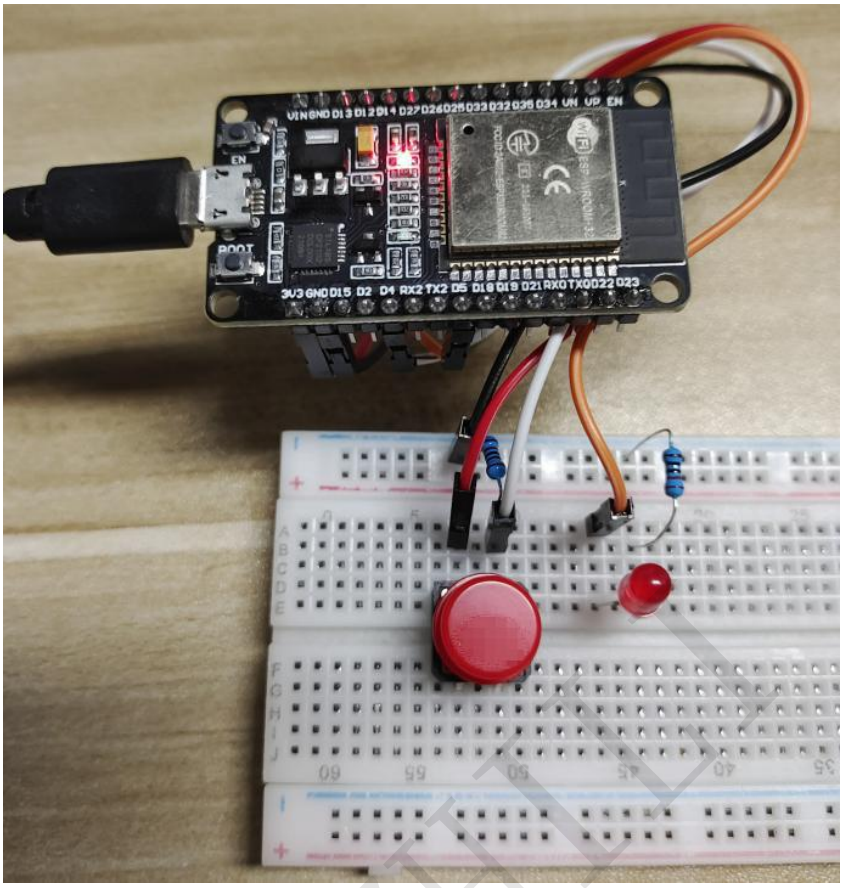
## 2.8 演示

上传代码后，测试电路。按下按钮时，LED 指示灯应亮起：



并在释放时关闭：

XINZHI



# 第3章 ESP32 模拟输入

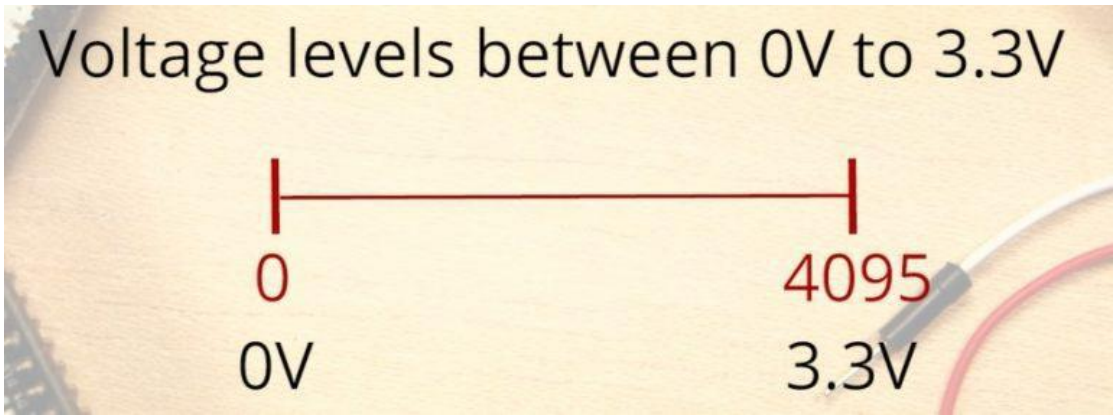
## 3.1 概述

本项目展示了如何通过 Arduino IDE 使用 ESP32 读取模拟输入。模拟读取适用于从可变电阻（如电位器）或模拟传感器中获取数值。

## 3.2 模拟输入 (ADC)

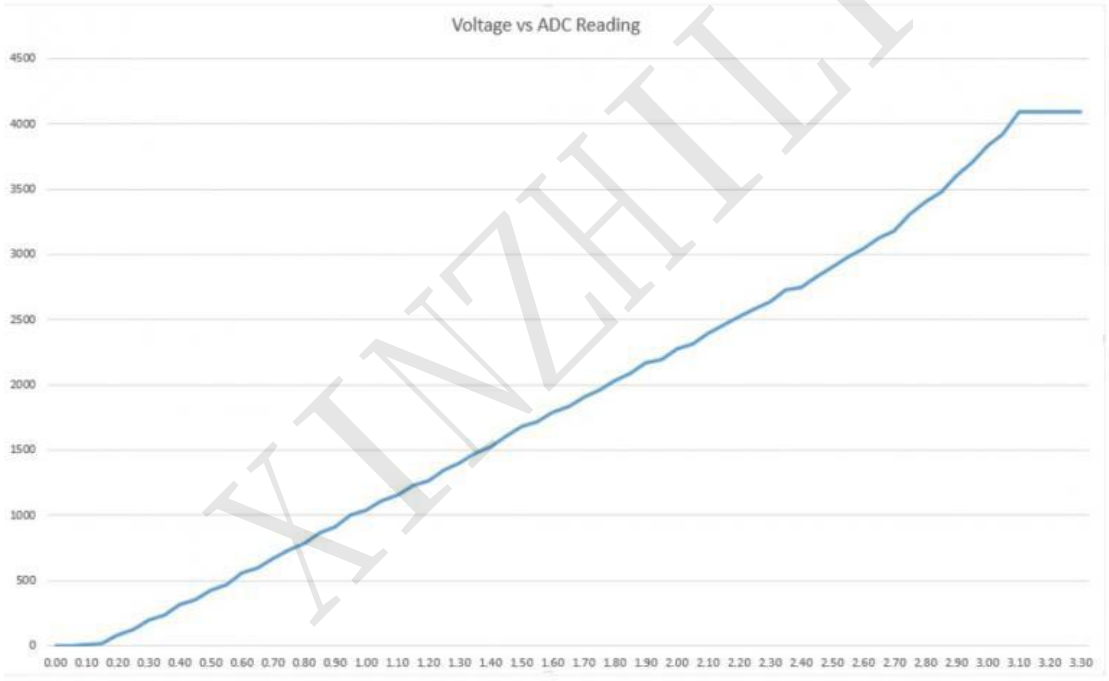
通过ESP32 读取模拟值意味着你可以测量 0 伏至 3.3 伏之间的变化的电压值。

测量到的电压会被映射到 0 至 4095 之间的数值，其中 0 伏对应 0，3.3 伏对应 4095。介于 0 伏和 3.3 伏之间的任何电压值均会对应相应的中间数值。




### 3.2.1 ADC 是非线性的

理想情况下，使用 ESP32 的 ADC 引脚时，您会期望其具有线性行为。然而实际情况并非如此。实际得到的行为如下图所示：




XINZHILI

这种行为意味着你的 ESP32 无法区分 3.3 伏和 3.2 伏的电压。对于这两个电压值，你会得到相同的数值：4095。

同样的情况也出现在极低电压范围：0 伏和 0.1 伏会得到相同的数值：0。在使用 ESP32 的 ADC 引脚时，需牢记这一特性。

### 3.2.2 analogRead() 函数

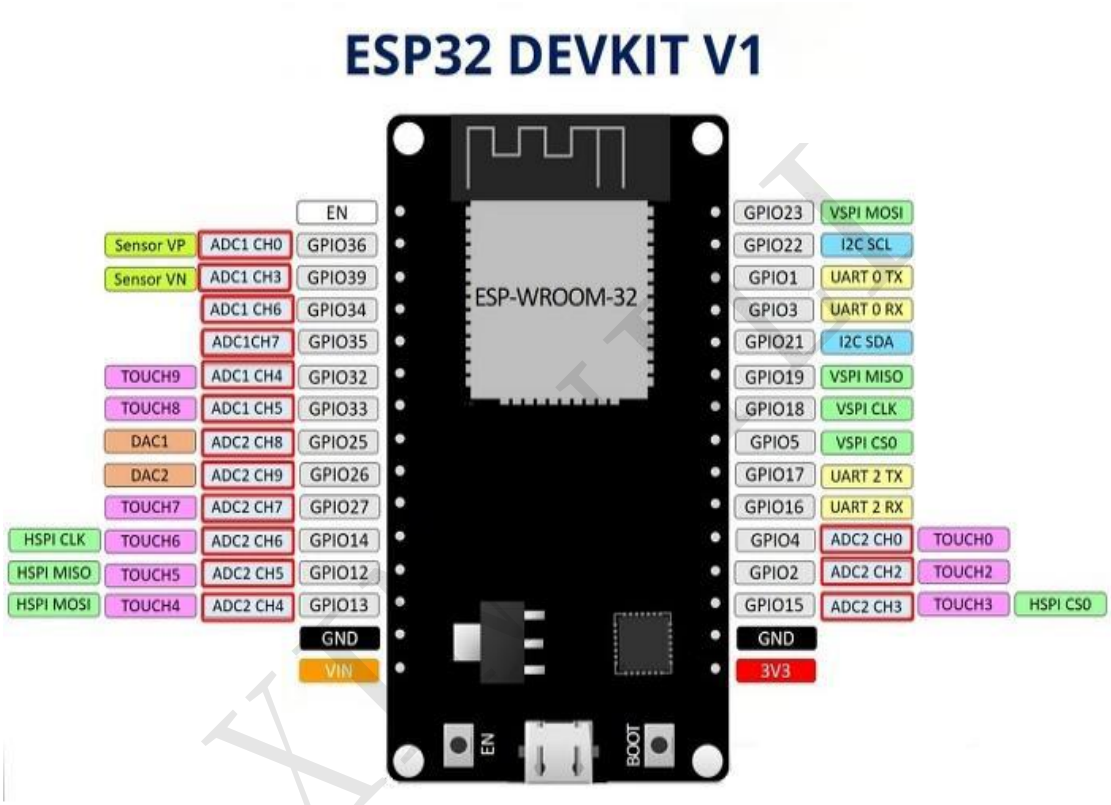
通过 Arduino IDE 使用 ESP32 读取模拟输入非常简单，只需使用

analogRead() 函数。该函数接受一个参数，即需要读取的 GPIO 引脚编号：

```
analogRead(GPIO);
```

在 DEVKIT V1 开发板（30 个 GPIO 引脚的版本）中，仅有 15 个 GPIO 引脚可用。

请查看你的 ESP32 开发板引脚图并定位 ADC 引脚。这些引脚在下图中以红色边框高亮显示。




这些模拟输入引脚具有 12 位分辨率。这意味着当你读取模拟输入时，其数值范围可从 0 到 4095。

注意：使用 Wi-Fi 时，ADC2 引脚不可用。因此，若你正在使用 Wi-Fi 且无法从 ADC2 GPIO 引脚读取数值，可尝试改用 ADC1 GPIO 引脚，此举通常能解决问题。

为展示整体应用方式，我们将通过一个简单示例读取电位器的模拟值。

## 3.3 所需元件

本示例需以下元件：

ESP32 DEVKIT V1 开发板

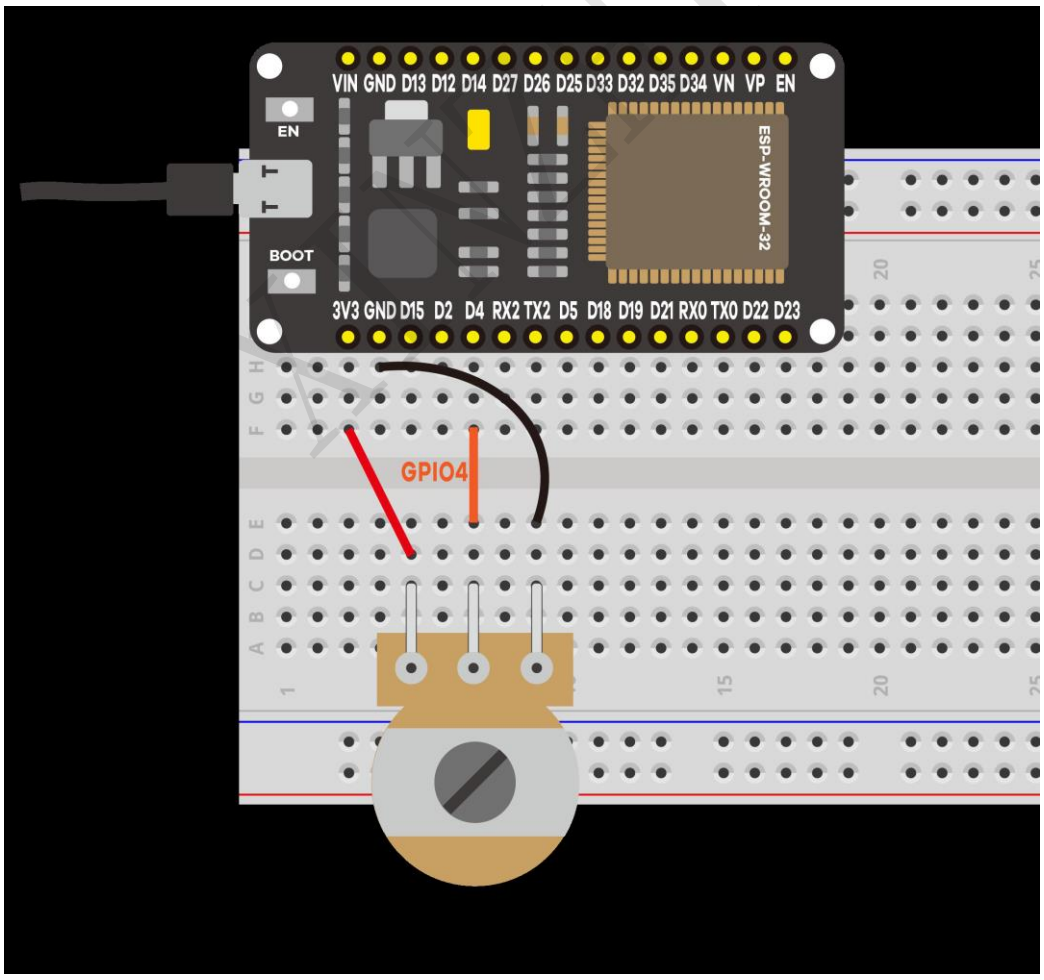
电位器

面包板

跳线

## 3.4 原理图

将电位器连接至 ESP32。电位器的中间引脚应连接到 GPIO 4。可参考以下电路图进行连接：



## 3.5 代码

我们将使用 Arduino IDE 对 ESP32 进行编程，因此请确保在继续操作前已安装 ESP32 插件：（如果已完成此步骤，可跳至下一步。）

在 Arduino IDE 中打开资料包中的“**代码 --> Project_2_ESP32_Analog_Inputs**”文件夹中的代码文件 Project_2_ESP32_Analog_Inputs.ino

```
// Potentiometer is connected to GPIO 4 (Analog ADC2_CH0)
const int potPin = 4;
// variable for storing the potentiometer valueint potValue = 0;
void setup() {
    Serial.begin(115200);
    delay(1000);
}

void loop() {
    // Reading potentiometer value
    potValue = analogRead(potPin);
    Serial.println(potValue);
    delay(500);
}
```

此代码仅读取电位器的数值，并在串行监视器中显示这些数值。

在代码中，首先需定义电位器连接的 GPIO 引脚。本例中使用 GPIO 4。

```
const int potPin = 4;
```

在 setup() 中，初始化波特率为 115200 的串口通信。

```
Serial.begin(115200);
```

在 loop() 中，使用 analogRead() 函数从 potPin 读取模拟输入值。

```
potValue = analogRead(potPin);
```

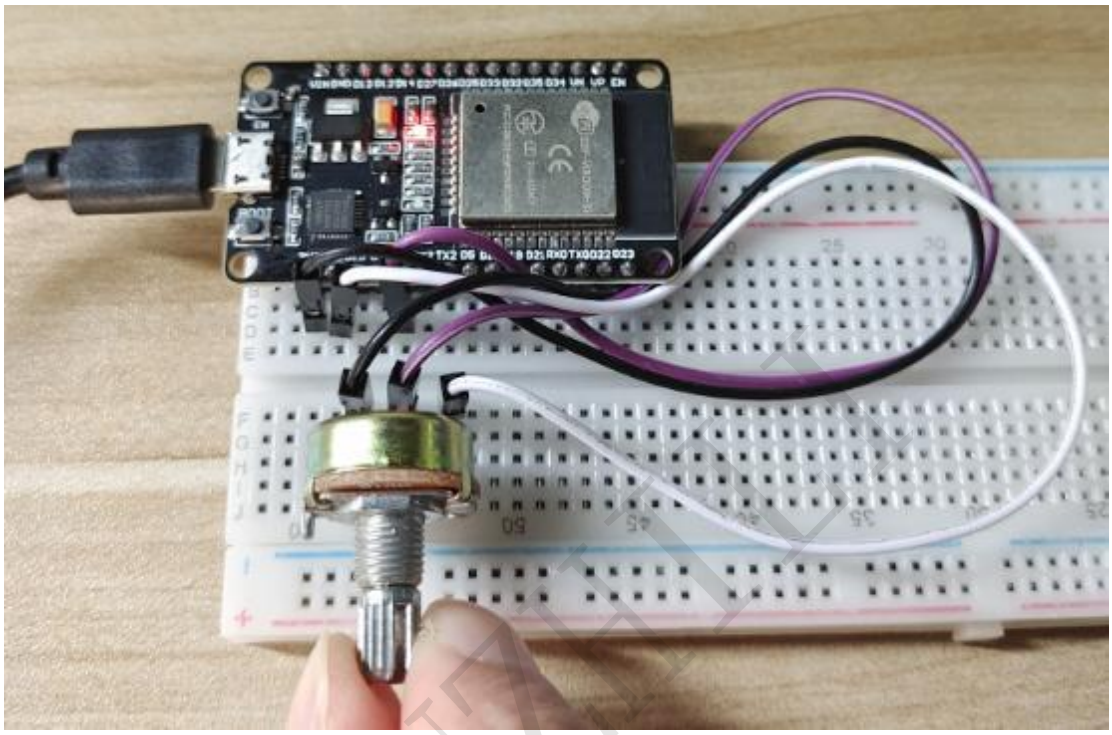
最后，将读取的电位器数值输出到串行监视器。

```
Serial.println(potValue);
```

将提供的代码上传至 ESP32。确保在 工具 菜单中选择了正确的开发板和 COM 端口。

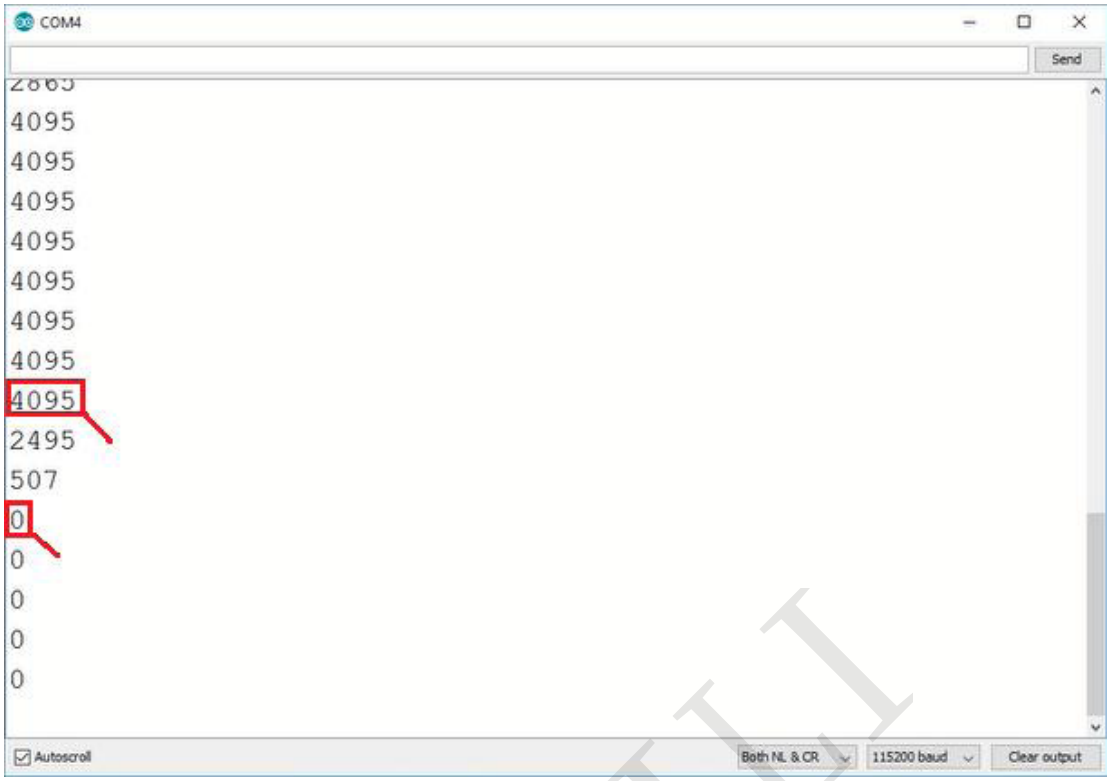
## 3.6 测试示例

上传代码并按下 ESP32 的重置按钮后，打开波特率为 115200 的串行监视器。旋转电位器，观察数值变化。



你将获得的最大数值为 4095，而最小数值为 0。

XINLI




## 3.7 总结

在本文中，你已学习了如何通过 Arduino IDE 使用 ESP32 读取模拟输入。以下是关键要点：

- **ESP32 DEVKIT V1 DOIT 开发板**（30 引脚版本）拥有 **15 个 ADC 引脚**，可用于读取模拟输入。
- 这些引脚具备 **12 位分辨率**，意味着数值范围为 **0 至 4095**。
- 在 Arduino IDE 中读取数值时，只需使用 analogRead() 函数。
- **ESP32 的 ADC 引脚具有非线性特性**。你很可能无法区分 **0 V 与 0.1 V** 或 **3.2 V 与 3.3 V** 的电压。使用 ADC 引脚时需牢记此特性。

XINZHILI

# 第4章 ESP32 PWM(模拟输出)

## 4.1 概述

在本教程中，我们将展示如何通过 Arduino IDE 使用 ESP32 生成 PWM 信号。我们将通过一个简单电路示例，利用 ESP32 的 LED PWM 控制器 调节 LED 的亮度。




## 4.2 ESP32 LED PWM 控制器

ESP32 拥有一个 LED PWM 控制器，包含 16 个独立通道，可配置生成具有不同特性的 PWM 信号。

以下是使用 Arduino IDE 通过 PWM 调节 LED 亮度的步骤：

1. 选择一个 PWM 通道（共 16 个通道，编号为 0 至 15）。

2. 设置 PWM 信号频率。对于 LED，建议频率为 5000 Hz。

3. 还需设置信号的 占空比分辨率：可选范围为 1 至 16 位。本例使用 8 位分辨率，即 LED 亮度可通过 0 至 255 的数值控制。

4.接着，需指定 PWM 信号输出的 GPIO 引脚。为此需调用以下函数：

```
ledcAttachPin(GPIO, channel);
```

此函数接受两个参数：

- 第一个参数为输出信号的 **GPIO 引脚**；
- 第二个参数为生成信号的 **通道号**。

5. 最后，通过以下函数使用 PWM 控制 LED 亮度：

```
ledcWrite(channel, dutycycle);
```

此函数接受两个参数：

- 生成 PWM 信号的 **通道号**；
- 目标 **占空比**（0 至最大值，如 255）。

## 4.3 所需元件

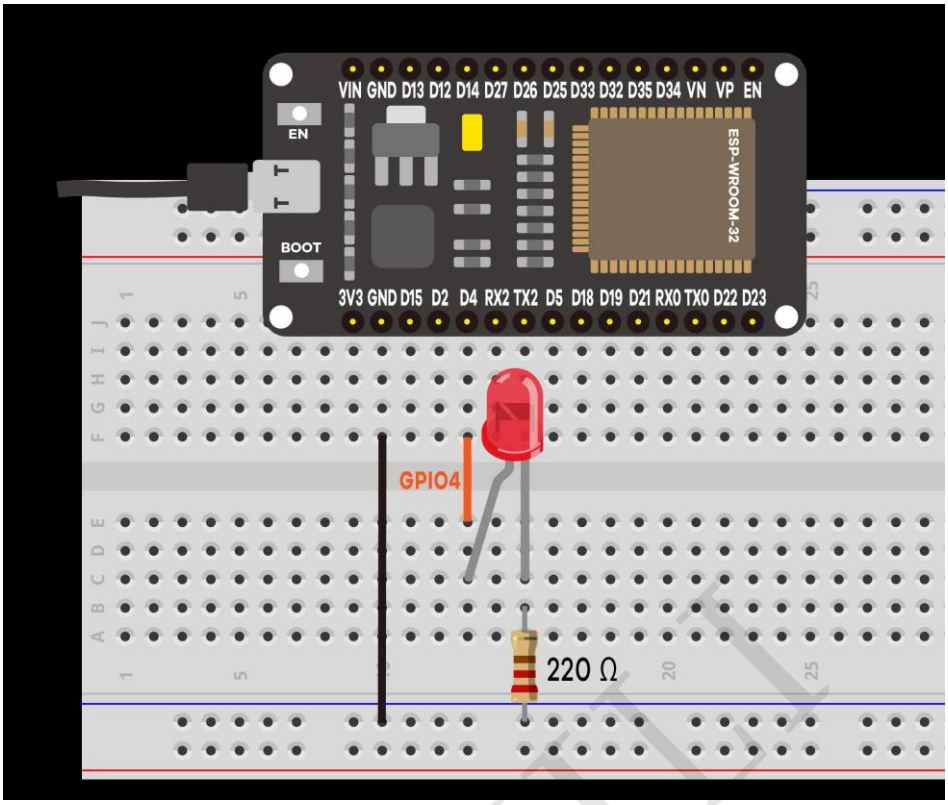
本教程需以下元件：

- ESP32 DEVKIT V1 开发板
- 5 毫米 LED
- 220 欧姆电阻
- 面包板
- 跳线

## 4.4 电路图

按照以下电路图将 LED 连接到你的 ESP32。LED 应连接至 GPIO 4。

XINZHILI




注意：只要引脚可用作输出，你可以使用任意引脚。所有可用作输出的引脚均可作为 PWM 引脚使用。如需详细了解 ESP32 的 GPIO 引脚，请阅读：ESP32 引脚参考指南：应使用哪些 GPIO 引脚？

## 4.5 代码

我们将使用 Arduino IDE 对 ESP32 进行编程，因此请确保在继续操作前已安装 ESP32 插件：（如果已完成此步骤，可跳至下一步。）

在 Arduino IDE 中打开资料包中的“**代码--> Project_3_ESP32_PWM**”文件夹中代码文件 Project_3_ESP32_PWM.ino

```
// the number of the LED pin
const int ledPin = 4; // 4 corresponds to GPIO4
// setting PWM properties
const int freq = 5000;
const int ledChannel = 0;
const int resolution = 8;

void setup(){
```

```
    // configure LED PWM functionalitites
    ledcSetup(ledChannel, freq, resolution);
    // attach the channel to the GPIO to be controlled
    ledcAttachPin(ledPin, ledChannel);
}

void loop(){
    // increase the LED brightness
    for(int dutyCycle = 0; dutyCycle <= 255; dutyCycle++){
        // changing the LED brightness with PWM
        ledcWrite(ledChannel, dutyCycle);
        delay(15);
    }
    // decrease the LED brightness
    for(int dutyCycle = 255; dutyCycle >= 0; dutyCycle--){
        // changing the LED brightness with PWM
        ledcWrite(ledChannel, dutyCycle);
        delay(15);
    }
}
```

首先定义 LED 连接的引脚。本例中 LED 连接至 GPIO 4。

```
const int ledPin = 4;  // 16 对应 GPIO4（注释保留原文逻辑）
```

接下来设置 PWM 信号参数：

- 定义频率为 5000 Hz；
- 选择通道 0 生成信号；
- 设置分辨率为 8 位（可调整参数以生成不同 PWM 信号）。

```
const int freq = 5000;
const int ledChannel = 0;
const int resolution = 8;
```

在 setup() 中，需通过 ledcSetup() 函数配置 LED PWM 参数。该函数接受 通道号、频率 和 分辨率 作为参数：

```
ledcSetup(ledChannel, freq, resolution);
```

然后，指定输出信号的 GPIO 引脚。使用 ledcAttachPin() 函数，参数为 目标 GPIO 和 生成信号的通道。本例中：

- 信号输出至 ledPin（即 GPIO 4）；
- 信号生成通道为 ledChannel（即通道 0）。

```
ledcAttachPin(ledPin, ledChannel);
```

在 loop() 中，通过将 占空比 从 0 到 255 逐步调整，以增加 LED 亮度。

```
for(int dutyCycle = 0; dutyCycle <= 255; dutyCycle++){
  // 通过 PWM 改变 LED 亮度
  ledcWrite(ledChannel, dutyCycle);
  delay(15);
}
```

随后，从 255 到 0 逐步降低亮度：

```
for(int dutyCycle = 255; dutyCycle >= 0; dutyCycle--){
  // 通过 PWM 改变 LED 亮度
  ledcWrite(ledChannel, dutyCycle);
  delay(15);
}
```

要设置 LED 的亮度，只需使用 ledcWrite() 函数。该函数接受两个参数：

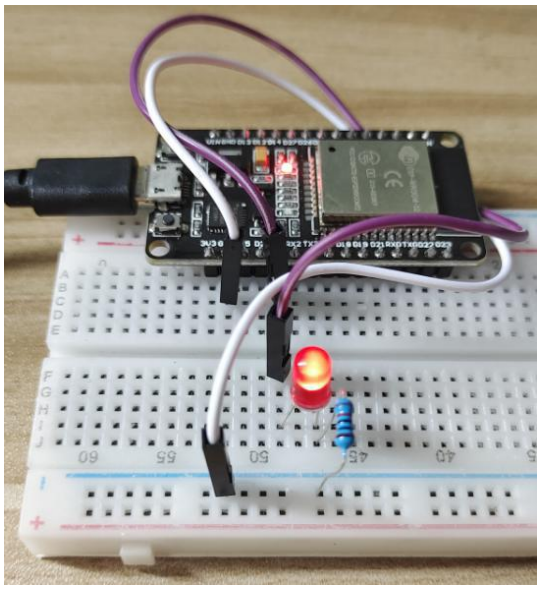
- 生成信号的通道号；
- 目标占空比。

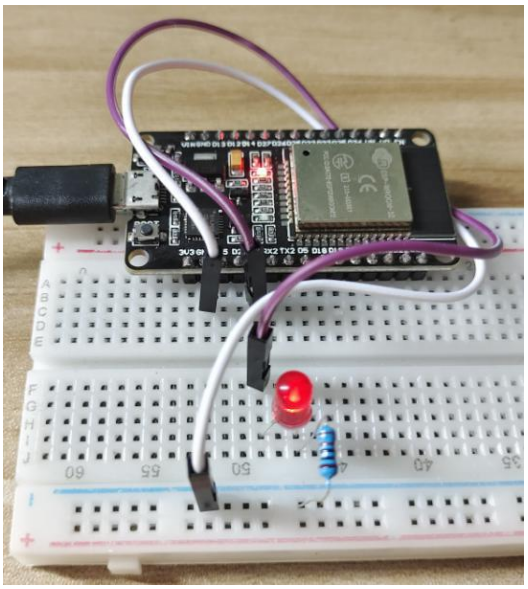
```
ledcWrite(ledChannel, dutyCycle);
```

由于使用 8 位分辨率，占空比范围为 0 至 255。注意：在 ledcWrite() 函数中，使用的是生成信号的 通道号，而非 GPIO 引脚。

## 4.6 测试示例

将代码上传至你的 ESP32。确保已选择正确的开发板和 COM 端口。观察你的电路。此时应能看到 LED 的亮度逐渐增加和减少。





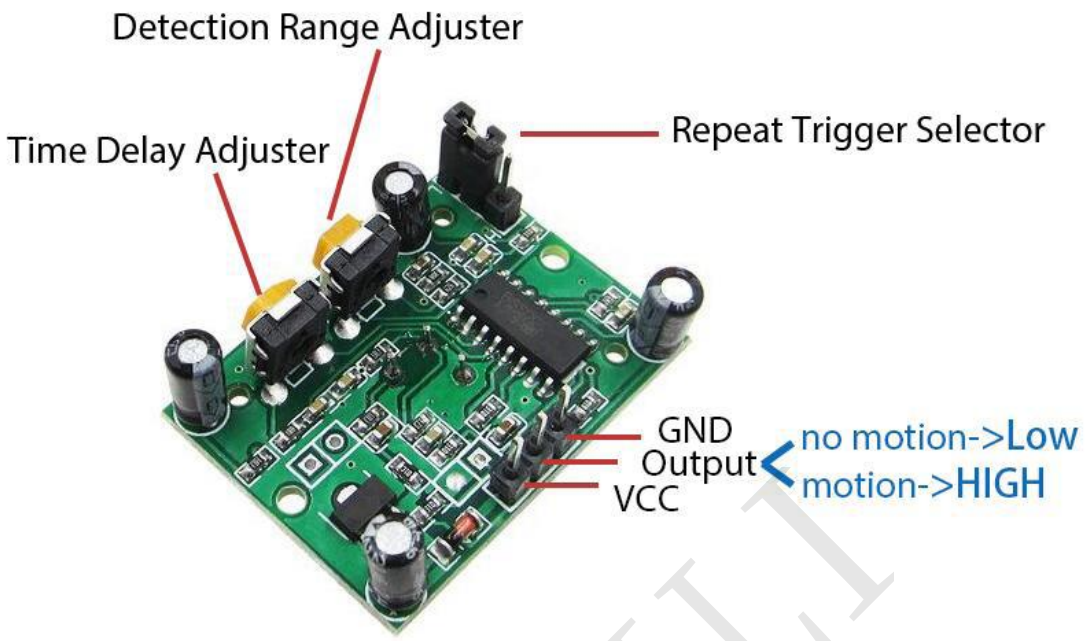
# 第5章 ESP32 PIR 运动传感器

## 5.1 概述

本项目展示了如何通过 ESP32 使用 **PIR 运动传感器** 检测运动。当检测到运动时，蜂鸣器将触发警报；若在预设时间内（例如 4 秒）未检测到运动，警报将自动停止。

XINZHILI

## 5.2 HC-SR501 人体感应传感器的工作原理




HC-SR501 传感器基于移动物体的红外辐射变化进行检测。若物体需要被 HC-SR501 传感器检测到，必须满足以下两个条件：

- **物体需发射红外线；**
- **物体需处于运动或振动状态。**

因此：

1. 若物体发射红外线但**未移动**（例如静止不动的人），传感器将**无法检测**。
2. 若物体移动但**未发射红外线**（例如机器人或车辆），传感器亦**无法检测**。

**引入定时器**

在本示例中，我们将引入定时器。我们希望在检测到运动后，LED 能保持亮起 预设的秒数。为实现此功能，应使用 定时器 而非 delay() 函数，因为 delay() 会在指定时间内阻塞代码执行，导致其他操作无法进行。

XINZHILI



### 5.2.1 delay() 函数

你应该对 delay() 函数较为熟悉，因其被广泛使用。此函数用法非常直观，它接受一个整数作为参数，表示程序需等待的毫秒数，之后才会继续执行下一行代码。

```
delay(时间（毫秒）);
```

例如，调用 delay(1000) 将使程序暂停 1 秒。

delay() 是一个 阻塞函数。阻塞函数会阻止程序执行其他任务，直到当前任务完成。若需同时执行多个任务，则不能使用 delay()。

建议：在大多数项目中，应避免使用 delay()，改用定时器或其他非阻塞方法。

### 5.2.2 millis() 函数

通过名为 millis() 的函数，可以获取程序启动后经过的毫秒数。

```
millis();
```

**该函数为何有用？**

通过数学运算，你可以轻松计算经过的时间，而不会阻塞代码执行。

## 5.3 所需元件

本教程需以下元件：

- **ESP32 DEVKIT V1** 开发板
- **PIR** 运动传感器（**HC-SR501**）

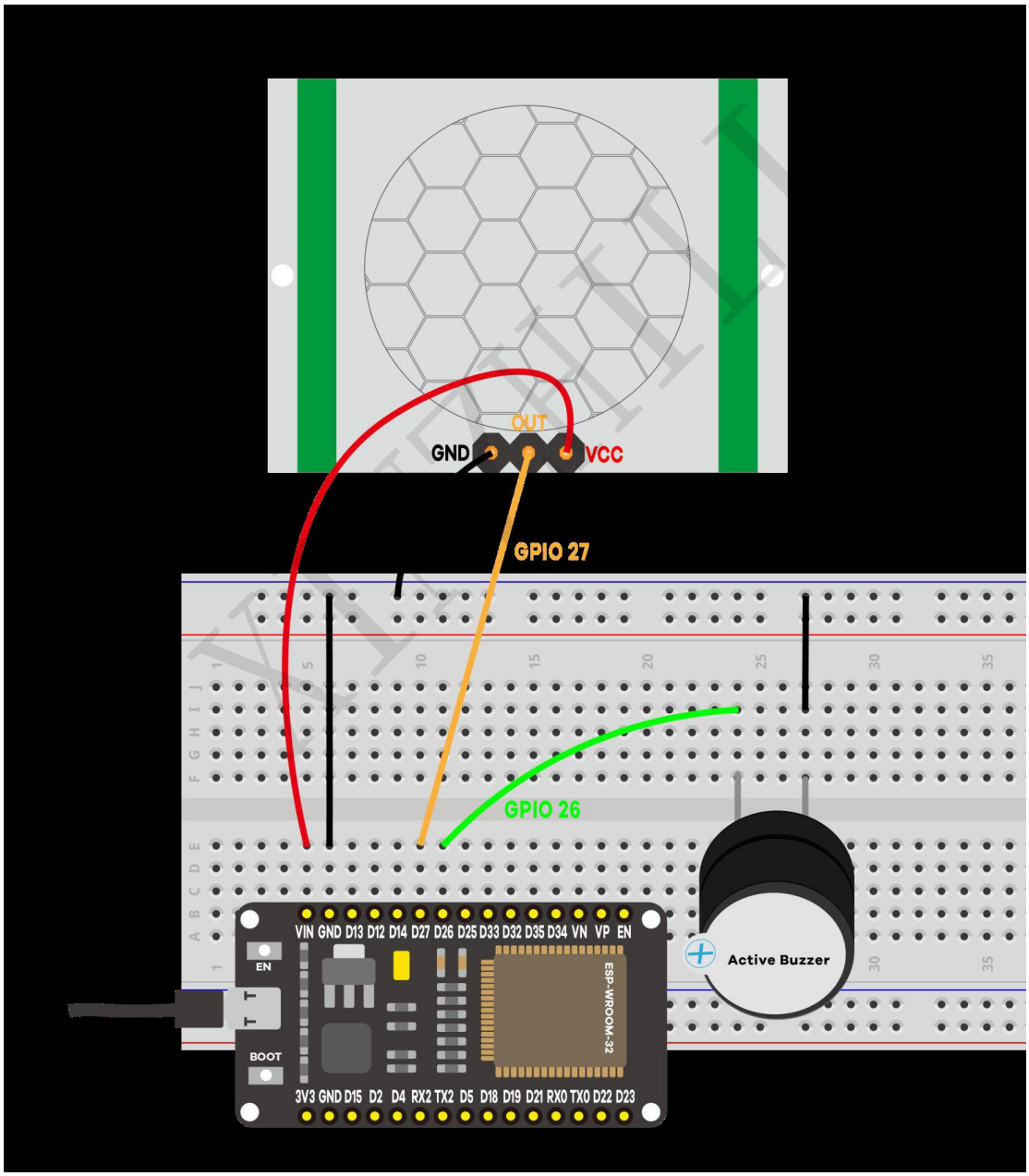
XINZHILI

♣ 有源蜂鸣器

♣ 跳线

♣ 面包板

## 5.4 电路图




注意：HC-SR501 的工作电压为 5V。请使用 Vin 引脚 为其供电。

## 5.5 代码

在继续本教程前，请确保已为 Arduino IDE 安装 ESP32 插件。若未安装，请参考以下教程之一进行安装：（如果已完成此步骤，可跳至下一步。）

在 Arduino IDE 中打开资料包中的“**代码 --> Project_4_ESP32_PIR_Motion_Sensor**”文件夹中代码文件 Project_4_ESP32_PIR_Motion_Sensor.ino。

## 5.6 演示

将代码上传至 ESP32 开发板。确保已选择正确的开发板和 COM 端口。上传步骤参考：

选择开发板：**工具 → 开发板 → ESP32 Dev Module**

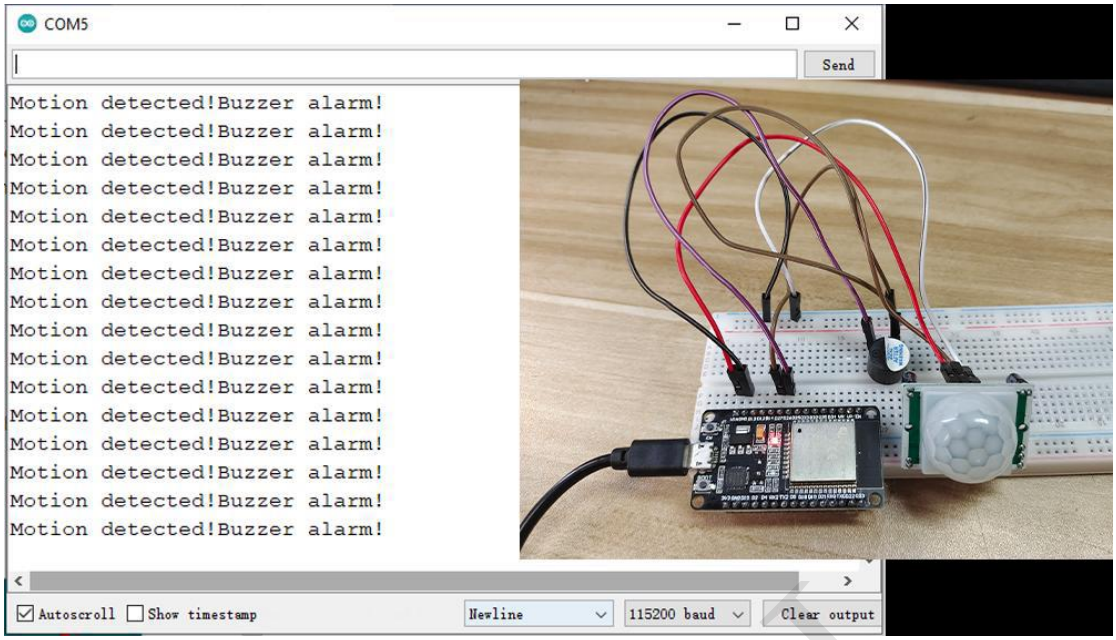
选择端口：**工具 → 端口 → 对应 COM 号**

上传完成后，打开波特率为 115200 的串行监视器。



在 PIR 传感器前移动你的手。蜂鸣器应启动，并在串行监视器中打印消息“检测到运动！蜂鸣器警报”。4 秒后，蜂鸣器应自动关闭。

XINZHILI




# 第6章 ESP32 网络开关服务器

## 6.1 概述

在本项目中，你将使用 ESP32 通过 Arduino IDE 编程环境创建一个独立的网络服务器，用于控制两个输出（LED）。该网络服务器支持移动端响应，可通过本地网络中任何带浏览器的设备访问。我们将逐步展示如何创建该服务器及其代码工作原理。

在直接进入项目前，明确网络服务器的功能目标将有助于后续步骤的理解：

你构建的网络服务器将控制连接至 ESP32 GPIO 26 和 GPIO 27 的两个 LED；

通过在本地网络浏览器中输入 ESP32 的 IP 地址即可访问该服务器；

通过点击网页上的按钮，可即时切换每个 LED 的状态。

XINZHILI

## 6.2 所需元件

本教程需以下元件：

ESP32 DEVKIT V1 开发板

2 个 5 毫米 LED

2 个 200 欧姆电阻

面包板

跳线

## 6.3 电路图

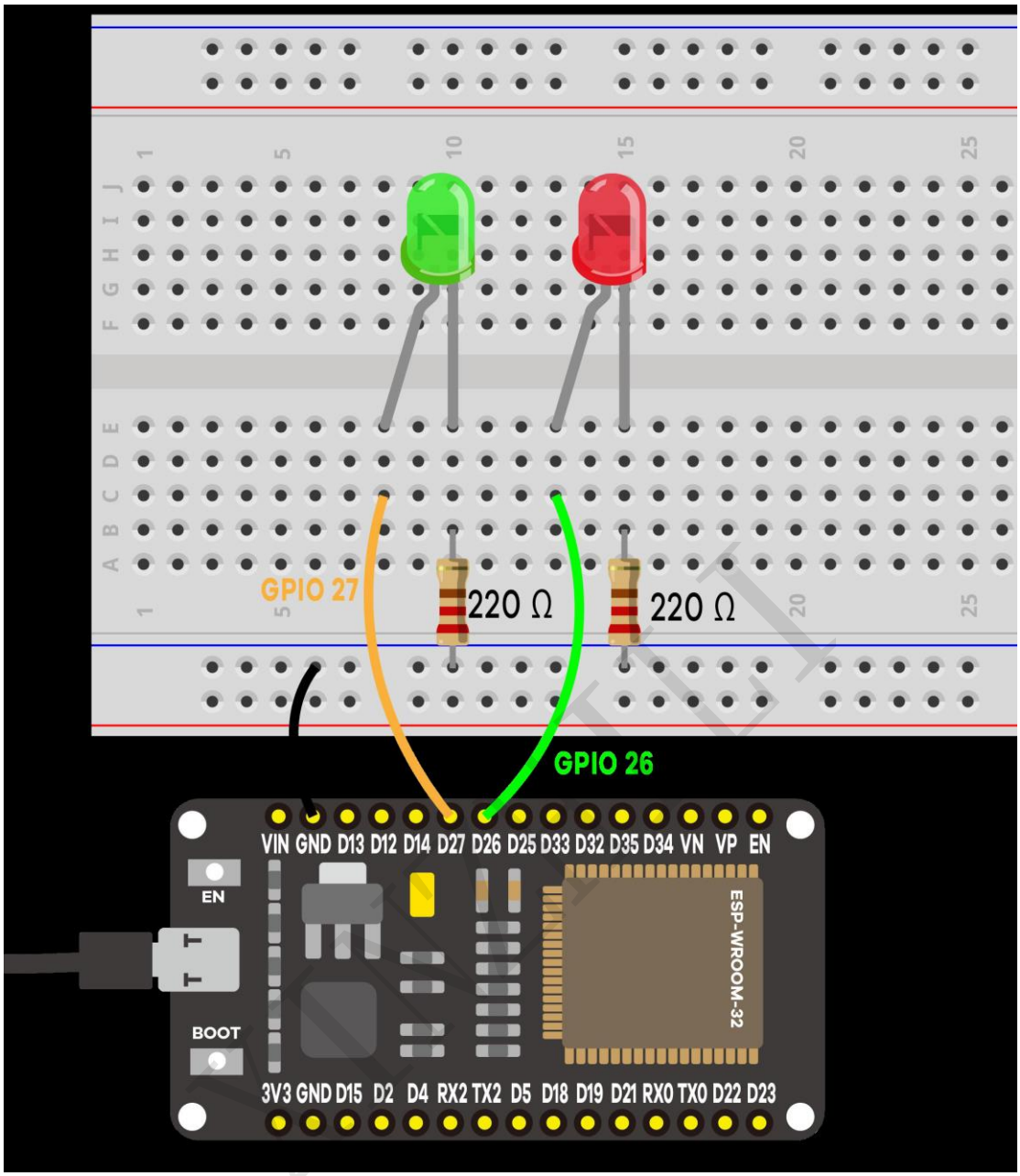
首先构建电路。将两个 LED 连接到 ESP32，如以下电路图所示：

- **LED 1** 连接至 **GPIO 26**；
- **LED 2** 连接至 **GPIO 27**。

**注意：**

- 我们使用的是 **36 引脚 ESP32 DEVKIT DOIT 开发板**。
- 在组装电路前，请核对你所使用开发板的 **引脚排列图**。

XINZHILI




## 6.4 代码

此处提供创建 ESP32 网络服务器的代码。在 Arduino IDE 中打开资料包中的“代码 --> Project_5_ESP32_Switch_Web_Server”文件夹中代码文件 Project_5_ESP32_Switch_Web_Server.ino，但暂不上传。你需要进行部分修改以适配你的环境。

我们将使用 Arduino IDE 对 ESP32 进行编程，因此请确保在继续操作前已安装 ESP32 插件：（如果已完成此步骤，可跳至下一步。）

### 设置网络凭证

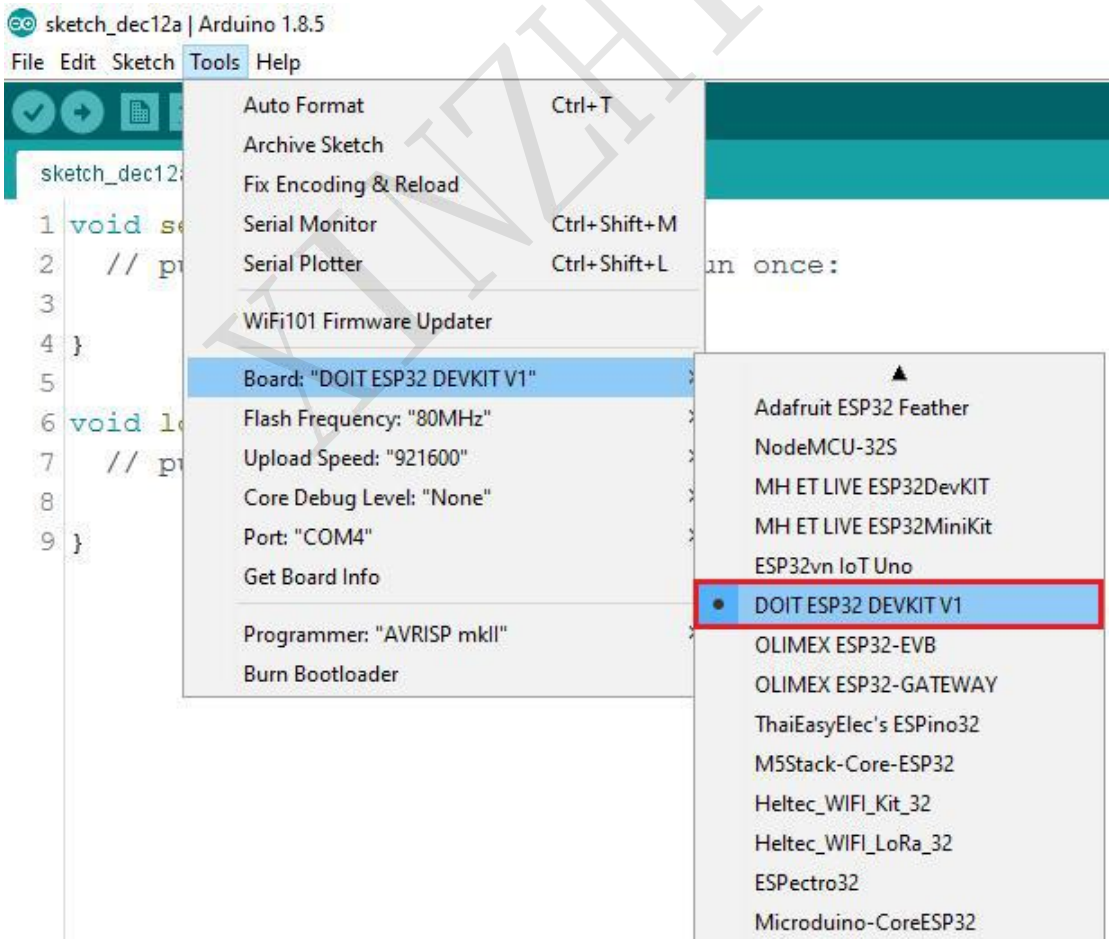
你需要修改以下代码行，填入你的网络凭证（即**路由器的 SSID 和密码**）。代码中已通过注释标出需修改的位置：

```
// 替换为你的网络凭证
const char* ssid = "REPLACE_WITH_YOUR_SSID";
const char* password = "REPLACE_WITH_YOUR_PASSWORD";
```

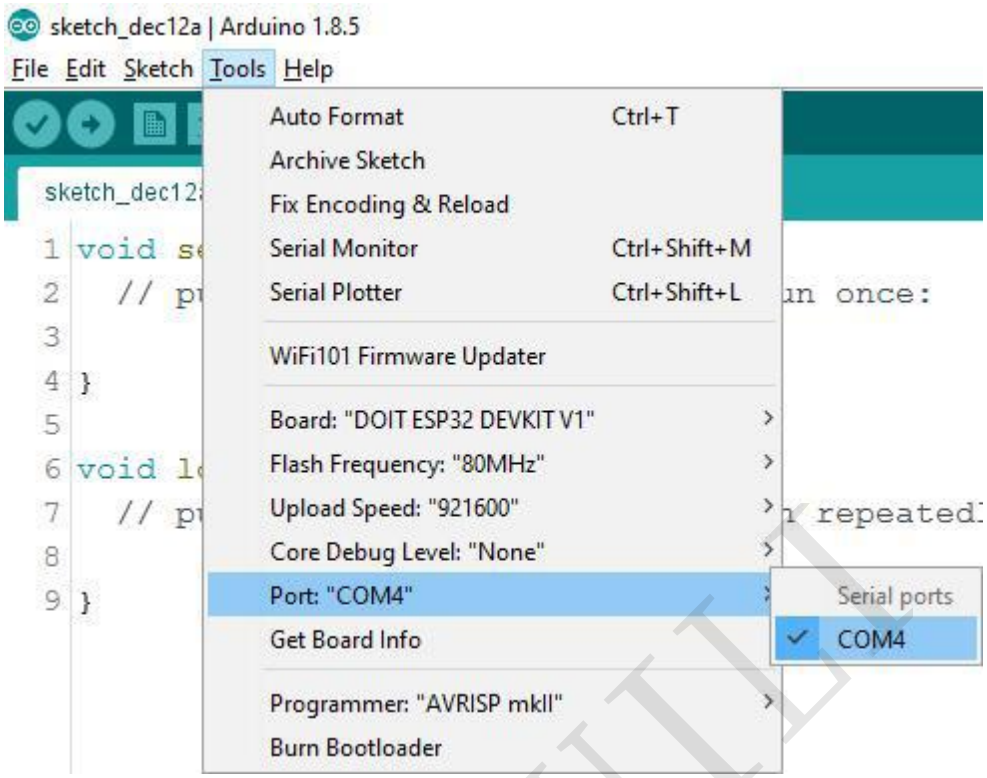
### 上传代码

现在可上传代码，网络服务器将立即运行。按以下步骤将代码上传至 ESP32：

1. 将 ESP32 开发板插入电脑；
2. 在 Arduino IDE 中选择开发板：**工具** → **开发板**（本例中使用 **ESP32 DEVKIT DOIT 开发板**）；



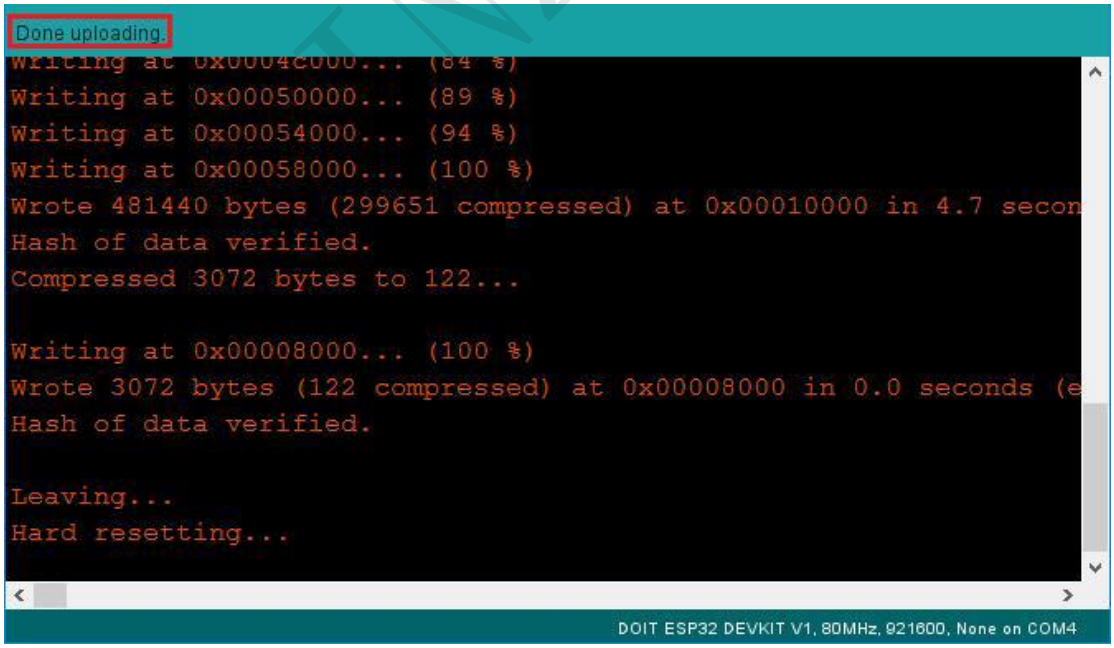
3. 选择 COM 端口：**工具 → 端口**。




4) 在 Arduino IDE 中点击 **上传** 按钮，等待几秒钟，代码将编译并上传至开发板。



5) 等待“上传完成”提示消息

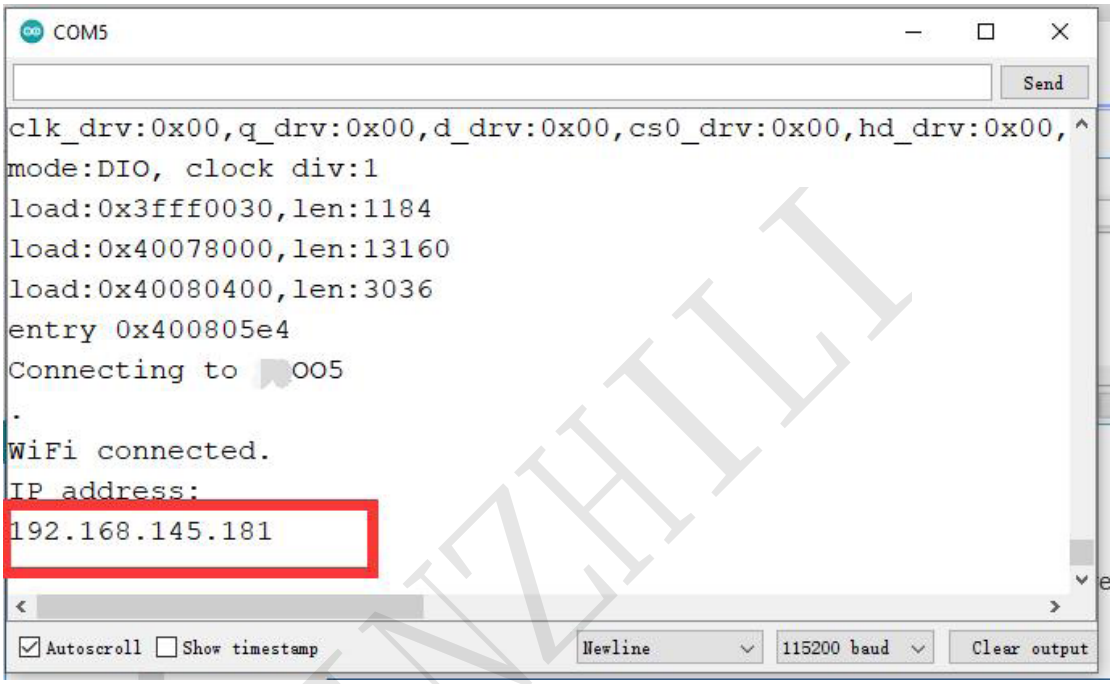



**查找 ESP 的 IP 地址**

代码上传完成后，打开波特率为 115200 的串行监视器。



按下 ESP32 的 EN 按钮（复位）。ESP32 将连接至 Wi-Fi，并在串行监视器中输出其 IP 地址。请复制该 IP 地址，因为你需要用它来访问 ESP32 的网络服务器。

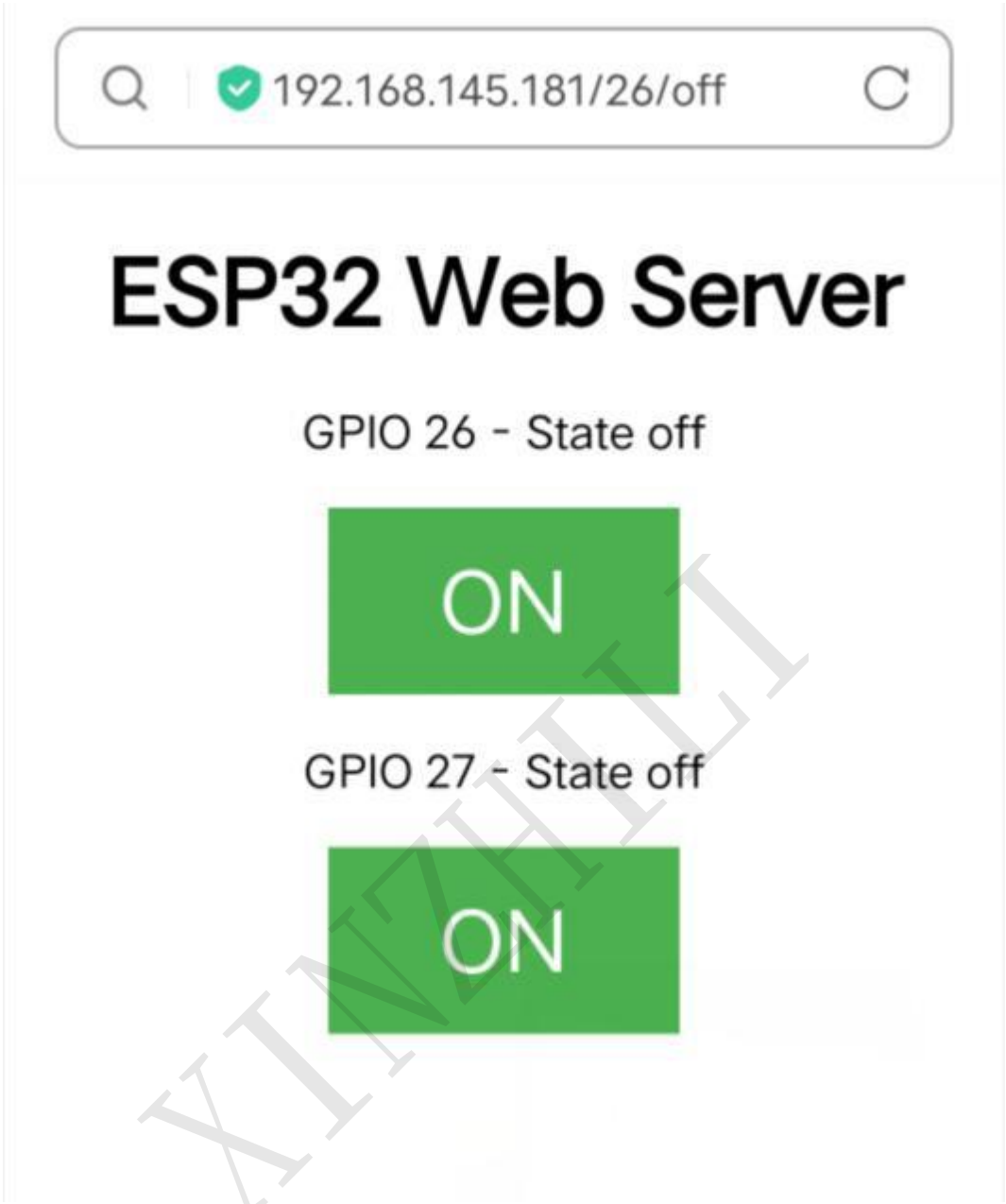



**访问网络服务器**

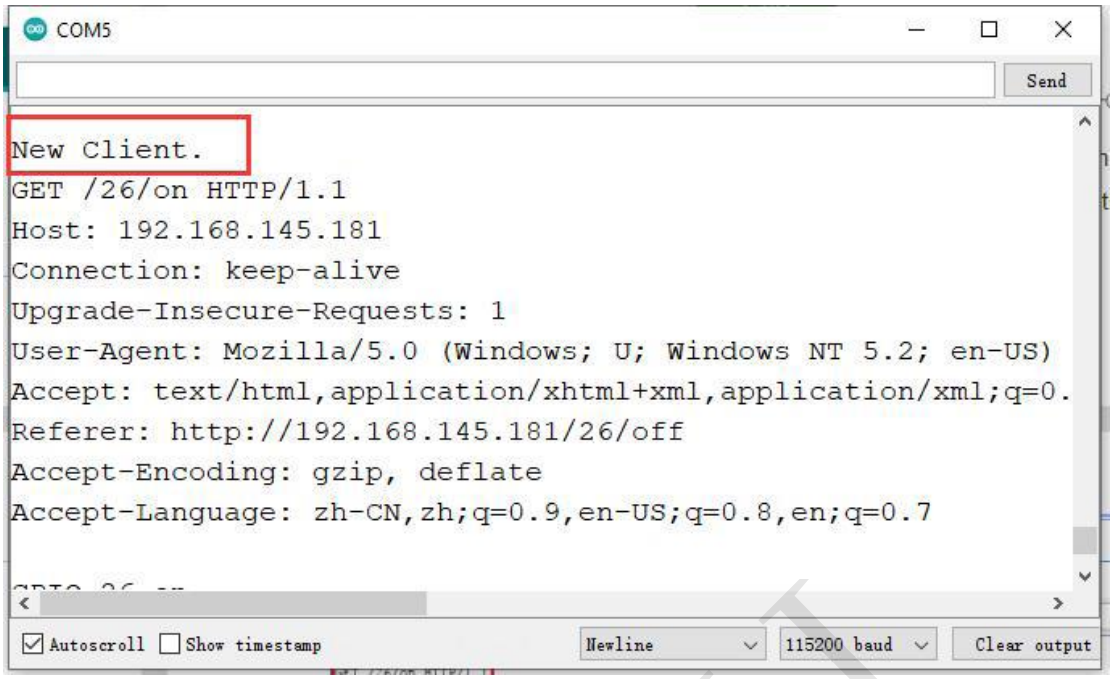
要访问网络服务器，请打开浏览器，粘贴 ESP32 的 IP 地址，您将看到如下页面。

**注意：**

- 你的浏览器与 ESP32 必须连接到 **同一局域网（LAN）**。




若查看串行监视器，可观察到后台运行状态。此时，ESP32 会收到来自新客户端的 HTTP 请求（在此场景中，客户端即你的浏览器）。




你还可以查看与 HTTP 请求相关的其他信息。

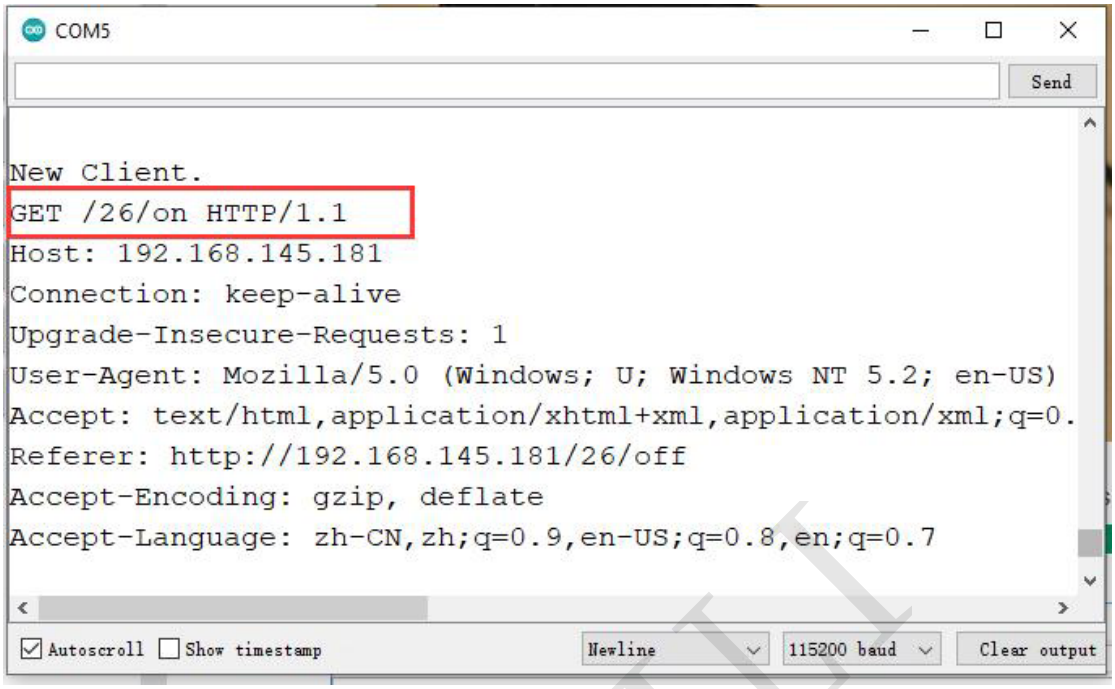
## 6.5 演示

现在，你可以测试网络服务器是否正常工作。点击按钮以控制 LED。




同时，你可以查看串行监视器以观察后台运行状态。例如，当你点击按钮以

开启 GPIO 26 时，ESP32 将在 /26/on 路径接收到 HTTP 请求。

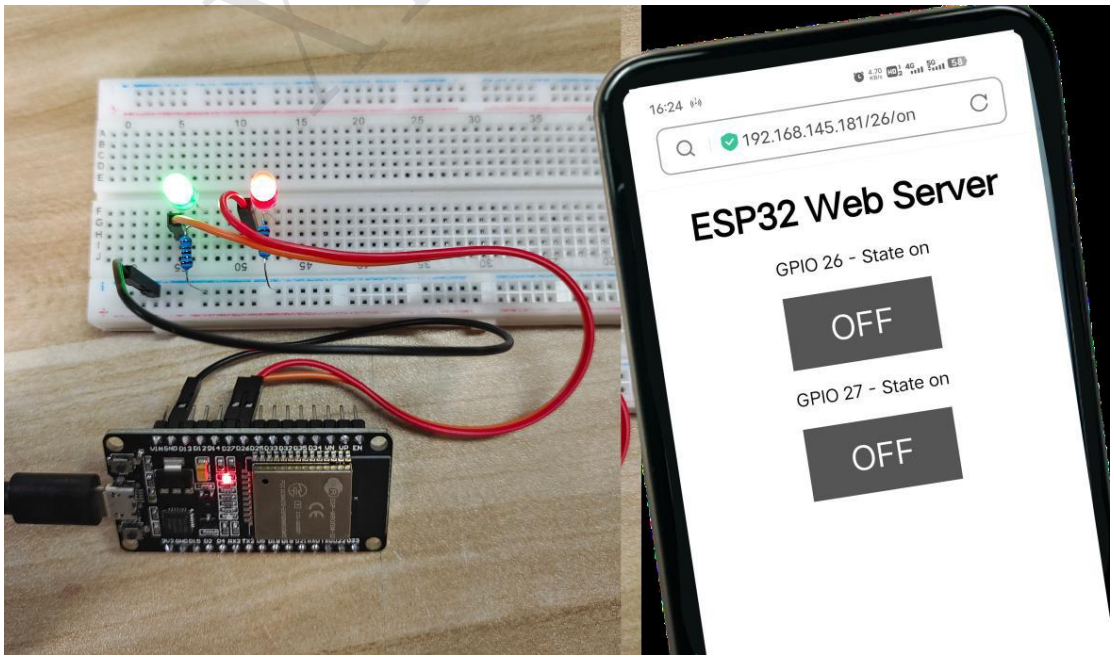



当 ESP32 接收到该请求时，将开启连接至 GPIO 26 的 LED，并在网页上更新其状态。

XINZHILI




GPIO 27 的按钮功能类似。请测试其是否正常工作。



## 代码工作原理

本节将深入解析代码，了解其工作机制。

首先需包含 WiFi 库：

```
#include <WiFi.h>
```

如前所述，你需要在以下代码行的双引号内填入 SSID 和 密码：

```
const char* ssid = "";
const char* password = "";
```

接着，将网络服务器端口设置为 80：

```
WiFiServer server(80);
```

以下代码行创建变量以存储 HTTP 请求头：

```
String header;
```

随后，创建辅助变量以保存输出端口的当前状态。若需添加更多输出并保存其状态，需额外定义变量：

```
String output26State = "off";
String output27State = "off";
```

还需为每个输出分配 GPIO 引脚。本例使用 GPIO 26 和 GPIO 27，但可选择其他适用的 GPIO：

```
const int output26 = 26;
const int output27 = 27;
```

---

### setup() 函数

现在，我们进入 setup() 函数的解析。

首先，初始化波特率为 115200 的串行通信（用于调试）：

```
Serial.begin(115200);
```

接下来，将 GPIO 引脚定义为 输出模式 并初始化为 低电平：

```
// 将输出变量初始化为输出模式
pinMode(output26, OUTPUT);
```

```
pinMode(output27, OUTPUT);

// 将输出设置为低电平
digitalWrite(output26, LOW);
digitalWrite(output27, LOW);
```

以下代码通过 WiFi.begin(ssid, password) 启动 Wi-Fi 连接，等待成功连接后，在串行监视器中打印 ESP32 的 IP 地址：

```
// Connect to Wi-Fi network with SSID and password
Serial.print("Connecting to ");
Serial.println(ssid);
WiFi.begin(ssid, password);while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
}

// Print local IP address and start web server
Serial.println("");
Serial.println("WiFi connected.");
Serial.println("IP address: ");
Serial.println(WiFi.localIP());
server.begin();
```

**loop() 函数**

在 loop() 函数中，我们编写了当新客户端与网络服务器建立连接时的处理逻辑。

ESP32 始终通过以下代码监听传入的客户端连接：

```
WiFiClient client = server.available();   // 监听传入的客户端
```

当接收到客户端的请求时，程序会保存传入的数据。只要客户端保持连接，后续的 while 循环将持续运行。除非你完全理解代码功能，否则不建议修改以下部分：

```
// Connect to Wi-Fi network with SSID and password
Serial.print("Connecting to ");
Serial.println(ssid);
WiFi.begin(ssid, password);while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
```

XINZHILI

```
}

// Print local IP address and start web server
Serial.println("");
Serial.println("WiFi connected.");
Serial.println("IP address: ");
Serial.println(WiFi.localIP());
server.begin();


if (client) { // If a new client connects,
    Serial.println("New Client."); // print a message out in the serial port
    String currentLine = ""; // make a String to hold incoming data from the
client
    while (client.connected()) { // loop while the client's connected
    if (client.available()) { // if there's bytes to read from the client,
    char c = client.read(); // read a byte, then
    Serial.write(c); // print it out the serial monitor
    header += c;
    if (c == '\n') { // if the byte is a newline character
    // if the current line is blank, you got two newline characters in a row.
    // that's the end of the client HTTP request, so send a response:
    if (currentLine.length() == 0) {
    // HTTP headers always start with a response code (e.g.HTTP/1.1 200 OK)
    // and a content-type so the client knows what's coming, then
    a blank line:
    client.println("HTTP/1.1 200 OK");
    client.println("Content-type:text/html");
    client.println("Connection: close");
    client.println();
```

下一部分的 **if 和 else 语句** 将检测网页中哪个按钮被按下，并据此控制输出设备。如之前所述，系统会根据按下的按钮向不同 URL 发送请求，从而触发对应的操作。

```
// turns the GPIOs on and offif (header.indexOf("GET /26/on") >= 0)
{
    Serial.println("GPIO 26 on");
    output26State = "on";
    digitalWrite(output26, HIGH);} else if
(header.indexOf("GET/26/off") >= 0) {
    Serial.println("GPIO 26 off");
    output26State = "off";
    digitalWrite(output26, LOW);} else if (header.indexOf("GET/27/on") >=
0) {
    Serial.println("GPIO 27 on");
    output27State = "on";
```

```
    digitalWrite(output27, HIGH);} else if
(header.indexOf("GET/27/off") >= 0) {
    Serial.println("GPIO 27 off");
    output27State = "off";
    digitalWrite(output27, LOW);}
```

例如，若你按下 GPIO 26 ON 按钮，ESP32 将在 /26/ON 路径接收到请求（可通过串行监视器的 HTTP 请求头查看此信息）。此时，程序会检查请求头是否包含 GET /26/on。若包含，则将 output26State 变量设为 ON，并点亮对应 LED。

其他按钮的功能与此类似。因此，若需添加更多输出设备，需在此部分代码中扩展逻辑。

---

### 生成 HTML 网页

下一步是创建网页。ESP32 将向浏览器发送包含 HTML 代码的响应以构建页面。通过 client.println() 函数向客户端发送网页内容，需将待发送内容作为参数传入。

首先，必须发送以下代码行，声明当前传输的是 HTML 内容：

```
<!DOCTYPE HTML><html>
运行 HTML
```

随后，以下代码使网页适配不同浏览器的响应式布局：

```
client.println("<head><meta name=\"viewport\" content=\"width=device-widt
h, initial-scale=1\">");
```

以下代码用于阻止浏览器请求 favicon（无需关注此行）：

```
client.println("<link rel=\"icon\" href=\"data:,\">");
```

### 网页样式设计

接下来，通过 CSS 代码设置按钮与网页的视觉样式：

- **字体与布局**：选用 Helvetica 字体，定义内容以块级元素内联显示，并居中对齐。

```
client.println("<style>html { font-family: Helvetica; display: inline-block; margin: 0px auto; text-align: center;}");
```

- **按钮样式**：
  - 背景色为 **#4CAF50**（**绿色**），无边框，文字白色，内边距为 **16px 40px**；
  - 无文本装饰，字体大小 30px，外边距 2px，光标悬停为指针样式。

```
client.println(".button { background-color: #4CAF50; border: none; color: white; padding: 16px 40px;");
client.println("text-decoration: none; font-size: 30px; margin: 2px; cursor: pointer;}");
```

- **关闭按钮样式**：继承上述按钮属性，但背景色更改为 **#555555**（**深灰色**）。

```
client.println(".button2 {background-color: #555555;}</style></head>");
```

**设置网页主标题**

在以下代码中设置网页的主标题。此处为“ESP32 Web Server”，但你可根据需求修改文本内容：

```
// 网页主标题
client.println("<h1>ESP32 Web Server</h1>");
```

---

**显示按钮与对应状态**

接下来，编写段落以显示 GPIO 26 的当前状态。此处使用 output26State 变量，因此当变量值变化时，状态会即时更新：

```
client.println("<p>GPIO 26 - 状态 " + output26State + "</p>");
```

随后，根据 GPIO 的当前状态显示 ON 或 OFF 按钮：

- 若 **GPIO 状态为 off**：显示 **ON** 按钮（点击后触发 /26/on 请求）；
- 若 **GPIO 状态为 on**：显示 **OFF** 按钮（点击后触发 /26/off 请求）。

```
if (output26State == "off") {
  client.println("<p><a href=\"/26/on\"><button class=\"button\">ON</button></a></p>");
}
else {
  client.println("<p><a href=\"/26/off\"><button class=\"button button2\">OFF</button></a></p>");
}
```

**GPIO 27** 的按钮逻辑与此相同。

### 关闭连接

当响应结束时，需清空头部变量并通过 client.stop() 终止与客户端的连接：

```
// 清空头部变量
header = "";

// 关闭连接
client.stop();
```

### 总结

在本教程中，我们展示了如何通过 ESP32 构建一个网络服务器。示例以控制两个 LED 为基础，但核心思路是将其替换为继电器或其他需控制的输出设备。

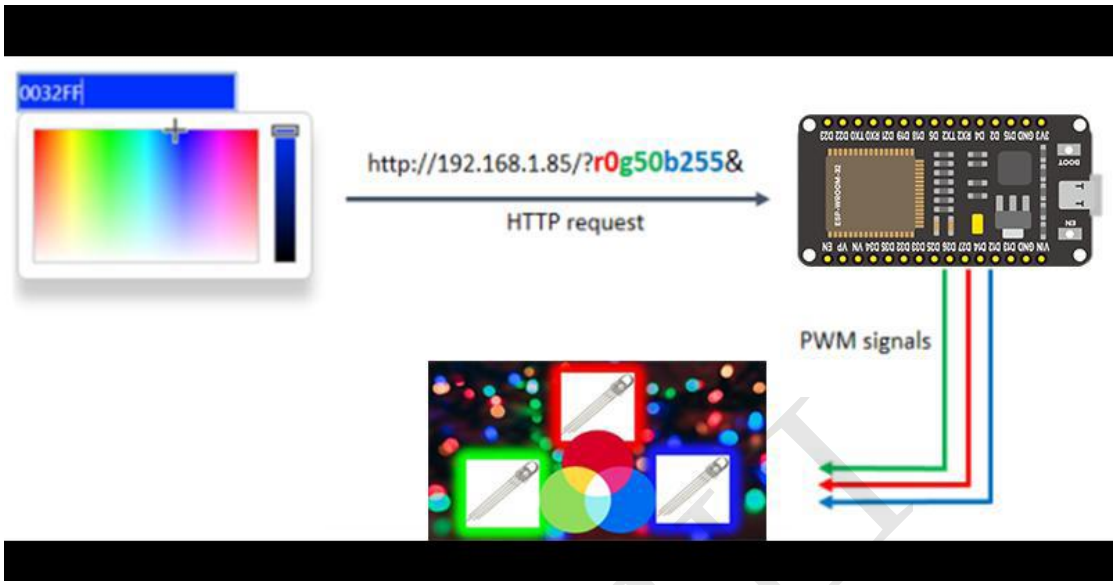
# 第7章 RGB LED 网络服务器

## 7.1 概述

在本项目中，我们将展示如何通过带有 颜色选择器 的网页服务器，使用 ESP32 开发板 远程控制 RGB LED。

## 7.2 原理

在开始之前，让我们先了解本项目的工作原理：




♣ **ESP32 网络服务器** 显示一个 **颜色选择器**。

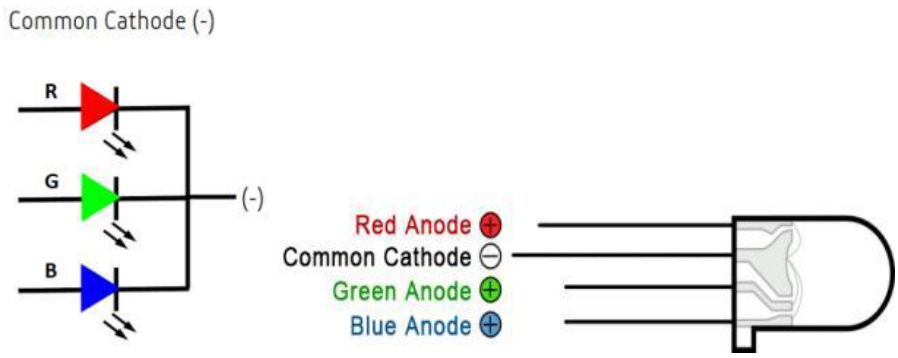
♣ 当你选择颜色时，浏览器会通过 **URL** 发送包含所选颜色的 **R、G、B 参数** 的请求。

♣ **ESP32** 接收请求，并分离各颜色参数的值。

♣ 随后，它向控制 **RGB LED** 的 **GPIO 引脚** 发送携带对应数值的 **PWM 信号**。

### 7.2.1 RGB LED 的工作原理

在 共阴极 RGB LED 中，三个 LED 共享一个负极连接（阴极）。套件中包含的均为 共阴极型 RGB LED。




**如何创建不同颜色？**

通过 RGB LED，你当然可以生成红、绿、蓝光，并通过调节每个 LED 的强度来生成其他颜色。

**示例：**

- **纯蓝光**：将蓝色 LED 设为**最高强度**，绿光和红光 LED 设为**最低强度**。
- **白光**：将所有三个 LED 设为**最高强度**。

---

**颜色混合**

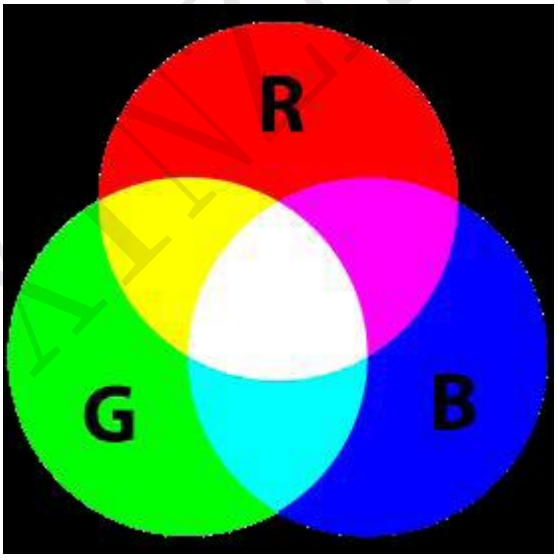
要生成其他颜色，可通过调整三种颜色的强度进行混合。

为调节每个 LED 的强度，可使用 PWM 信号。

由于 LED 灯珠紧密排列，人眼感知到的是颜色混合后的结果，而非单独的红、绿、蓝三色。

要了解颜色混合方法，可参考以下颜色混合图表。

此图表为 最简化的颜色混合示意，但能帮助你理解其工作原理及如何生成不同颜色。




## 7.3 所需元件

本项目需以下元件：

**♣ ESP32 DEVKIT V1 开发板**

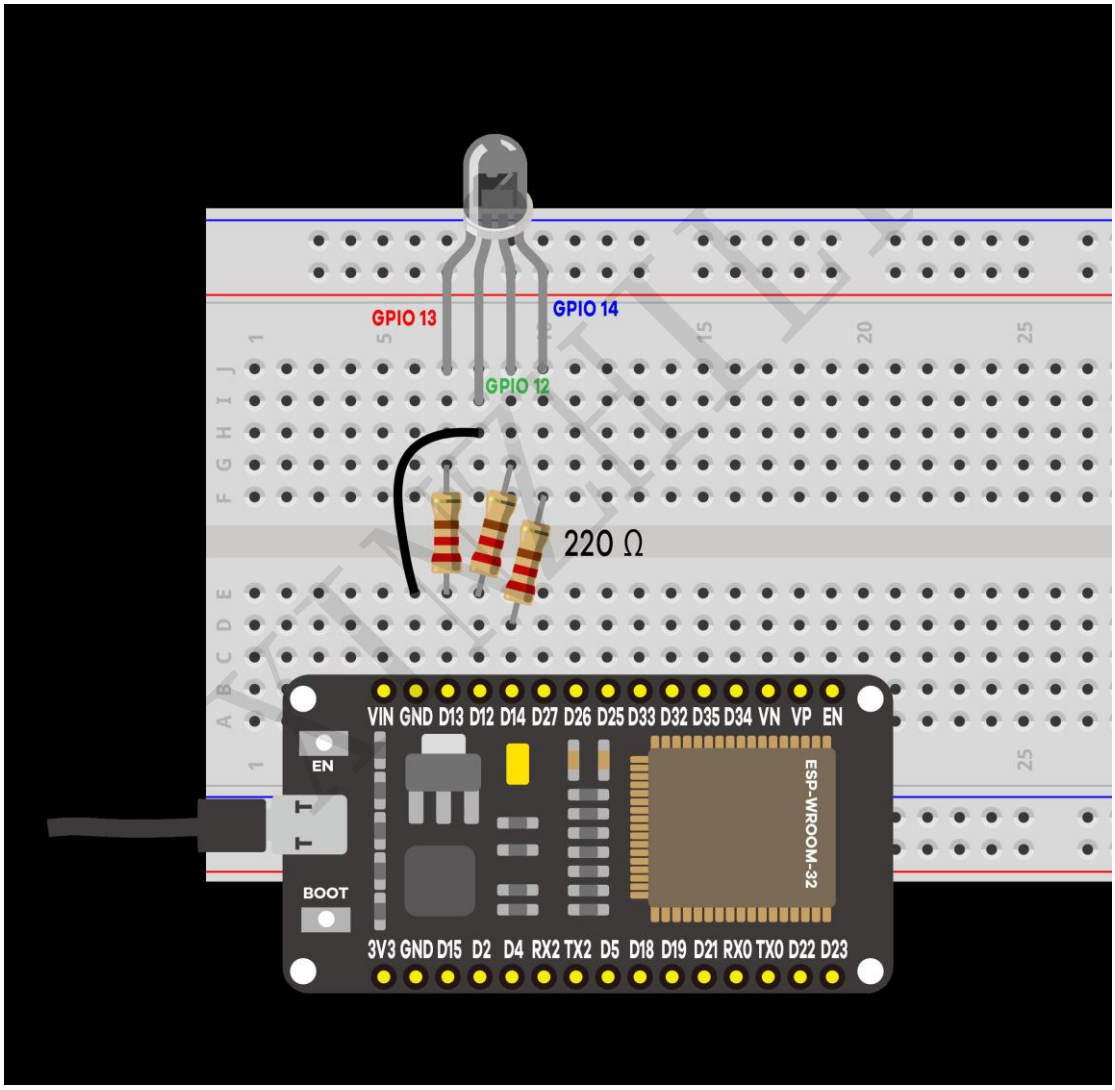
♣ **RGB LED**

♣ **3 个 220 欧姆电阻**

♣ **跳线**

♣ **面包板**

## 7.4 电路图




我们将使用 Arduino IDE 对 ESP32 进行编程，因此请确保在继续操作前已安装 ESP32 插件：（如果已完成此步骤，可跳至下一步。）

## 7.5 代码

组装电路后，在 Arduino IDE 中打开资料包中的“**代码 --> Project_6_RGB_LED_Web_Server**”文件夹中代码文件 Project_6_RGB_LED_Web_Server.ino。上传代码前，请勿忘记填入你的网络凭证，以便 ESP32 连接到本地网络：

```
const char* ssid = "REPLACE_WITH_YOUR_SSID";
const char* password = "REPLACE_WITH_YOUR_SSID";
```

**代码工作原理**

此 ESP32 代码使用 WiFi.h 库：

```
#include <WiFi.h>
```

以下代码行定义字符串变量以存储请求中的 R、G、B 参数：

```
String redString = "0";
String greenString = "0";
String blueString = "0";
```

接下来四个变量用于后续解析 HTTP 请求：

```
int pos1 = 0;
int pos2 = 0;
int pos3 = 0;
int pos4 = 0;
```

定义三个变量对应控制 R、G、B 的 GPIO 引脚。本例使用 GPIO 13、GPIO 12 和 GPIO 14：

```
const int redPin = 13;
const int greenPin = 12;
const int bluePin = 14;
```

这些 GPIO 引脚需输出 PWM 信号，因此需先配置 PWM 参数：

- **设置 PWM 信号频率为 5000 Hz**；
- 为每个颜色分配独立的 **PWM 通道**。

```
const int freq = 5000;
const int redChannel = 0;
```

```
const int greenChannel = 1;
const int blueChannel = 2;
```

最后，将 PWM 通道的分辨率设为 8 位：

```
const int resolution = 8;
```

在 setup() 中，为每个 PWM 通道配置属性：

```
ledcSetup(redChannel, freq, resolution);
ledcSetup(greenChannel, freq, resolution);
ledcSetup(blueChannel, freq, resolution);
```

将 PWM 通道绑定至对应的 GPIO 引脚：

```
ledcAttachPin(redPin, redChannel);
ledcAttachPin(greenPin, greenChannel);
ledcAttachPin(bluePin, blueChannel);
```

以下代码段在网页中显示 颜色选择器，并根据所选颜色发送请求：

```
    client.println("<!DOCTYPE html><html>");
    client.println("<head><meta
name=\"viewport\"content=\"width=device-width, initial-scale=1\">");
    client.println("<link rel=\"icon\" href=\"data:,\">");
    client.println("<link
rel=\"stylesheet\"href=\"https://stackpath.bootstrapcdn.com/bootstrap/4
.3.1/css/boo tstrap.min.css\">");
    client.println("<script
src=\"https://cdnjs.cloudflare.com/ajax/libs/jscolor/2.0.4/jscolor.min.
js\"></script>");
    client.println("</head><body><div
class=\"container\"><divclass=\"row\"><h1>ESP Color Picker</h1></div>");
    client.println("<a class=\"btn btn-primary btn-lg\"
href=\"#\"id=\"change_color\" role=\"button\">Change Color</a> ");
    client.println("<input
class=\"jscolor{onFineChange:'update(this)'}\" id=\"rgb\"></div>");
    client.println("<script>function
update(picker){document.getElementById('rgb').innerHTML
=Math.round(picker.rgb[0]) + ', ' + Math.round(picker.rgb[1]) + ',' +
Math.round(picker.rgb[2]);");
    client.println("document.getElementById(\"change_color\").href=\"?r\
" + Math.round(picker.rgb[0]) + \"g\" + Math.round(picker.rgb[1])+ \"b\"
```

```
+ Math.round(picker.rgb[2]) +\"&\";}</script></body></html>");// The HTTP
response ends with
    another blank line
    client.println();
```

当选择颜色时，你将收到如下格式的请求：

```
/?r201g32b255&
```

因此，需分割该字符串以提取 R、G、B 参数。参数将保存至 redString、greenString 和 blueString 变量中，取值范围为 0 至 255。

```
pos1 = header.indexOf('r');  // 定位红色参数起始位置
pos2 = header.indexOf('g');  // 定位绿色参数起始位置
pos3 = header.indexOf('b');  // 定位蓝色参数起始位置
pos4 = header.indexOf('&');  // 定位参数结束位置

redString   = header.substring(pos1+1, pos2);  // 提取红色值（如 "201"）

greenString = header.substring(pos2+1, pos3);  // 提取绿色值（如 "32"）
blueString  = header.substring(pos3+1, pos4);  // 提取蓝色值（如 "255"）
```

要使用 ESP32 控制灯带，可通过 ledcWrite() 函数生成基于 HTTP 请求解析值的 PWM 信号：

```
ledcWrite(redChannel, redString.toInt());     // 红色通道 PWM
ledcWrite(greenChannel, greenString.toInt()); // 绿色通道 PWM
ledcWrite(blueChannel, blueString.toInt());   // 蓝色通道 PWM
```

注意：有关 ESP32 PWM 的更多信息，请参考 项目 3：ESP32 PWM（模拟输出）。

对于 ESP8266，需使用 analogWrite() 函数生成 PWM 信号：

```
analogWrite(redPin, redString.toInt());     // 红色引脚 PWM
analogWrite(greenPin, greenString.toInt()); // 绿色引脚 PWM
analogWrite(bluePin, blueString.toInt());   // 蓝色引脚 PWM
```

由于参数值以字符串形式接收，需通过 toInt() 方法将其转换为整数。

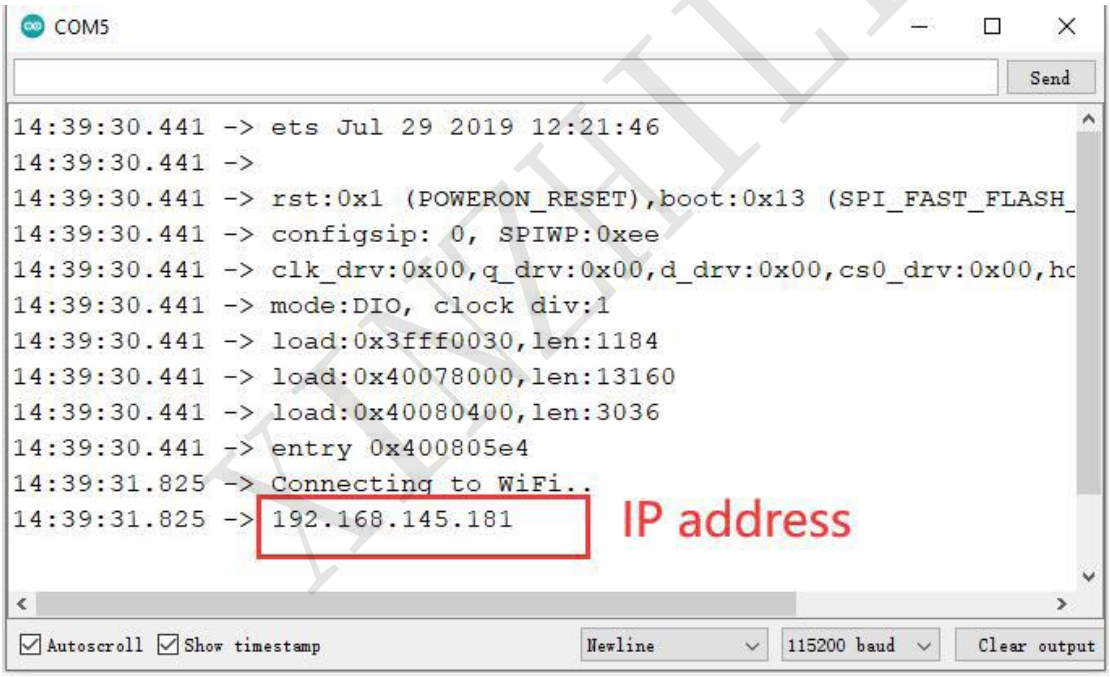
## 7.6 演示

填入网络凭证后，按以下步骤上传代码至 ESP32：

1. 选择正确的开发板：**工具** → **开发板** → **ESP32 Dev Module**；
2. 选择正确的 COM 端口：**工具** → **端口** → **对应 COM 号**；
3. 点击 **上传** 按钮。

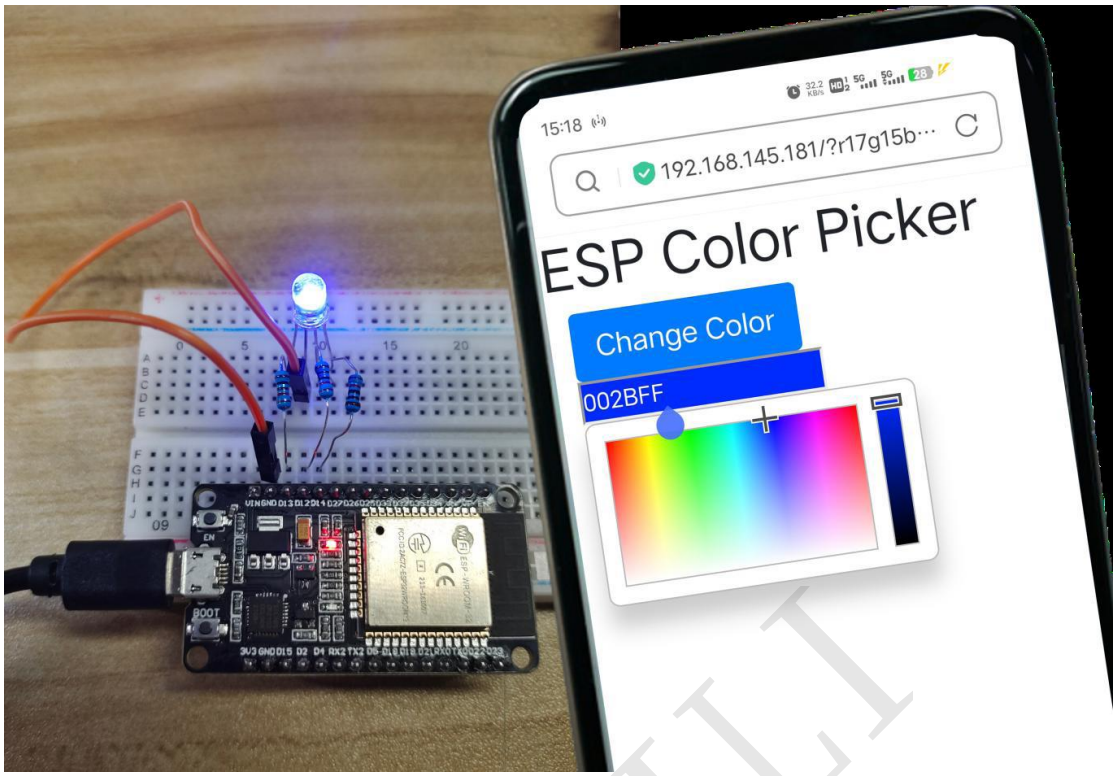
代码上传完成后，打开波特率为 115200 的串行监视器，并按下 ESP32 的 EN 复位按钮。此时，串行监视器将显示开发板的 IP 地址。

注意：您的浏览器与 ESP32 必须连接到 同一局域网（LAN）。




打开浏览器并输入 ESP32 的 IP 地址。此时，使用 颜色选择器 为 RGB LED 选择颜色。

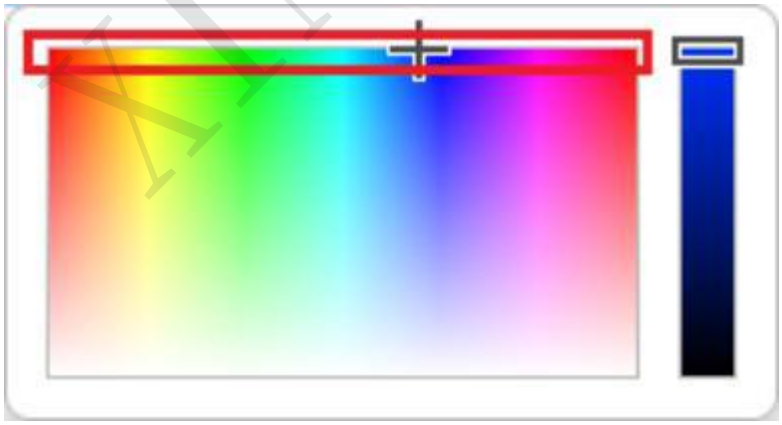
随后，需点击 “Change Color” 按钮以使颜色生效。




要关闭 RGB LED，请选择 黑色。

## 颜色选择提示

颜色选择器中位于顶部的 高饱和度颜色 通常能呈现更佳效果。



XINZHILI

# 第8章 ESP32 继电器网络服务器

## 8.1 概述

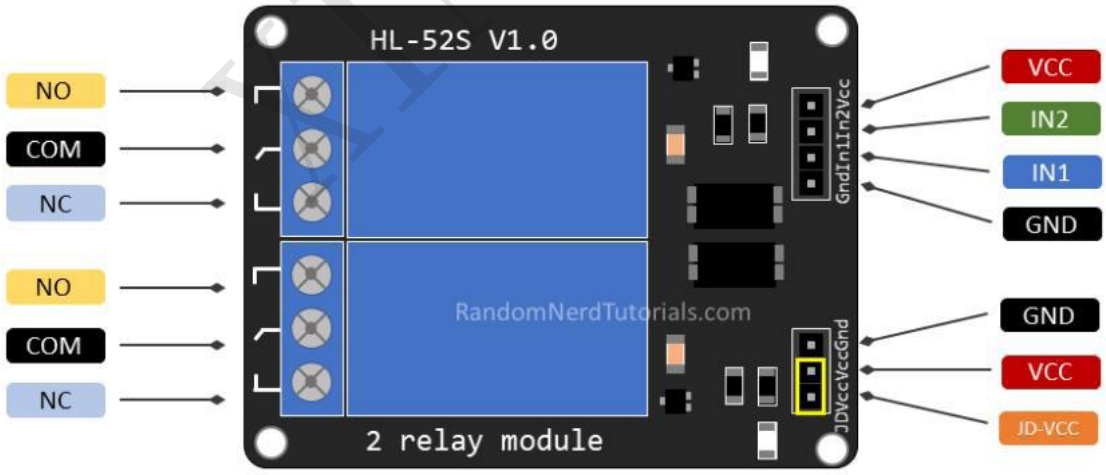
使用 ESP32 控制继电器模块 是实现 远程控制交流家用电器 的绝佳方案。本教程将详细说明如何通过 ESP32 控制继电器模块，包括：

1. **继电器模块的工作原理；**
2. **如何将继电器连接至 ESP32；**
3. **构建远程控制继电器的网络服务器。**

### 继电器简介

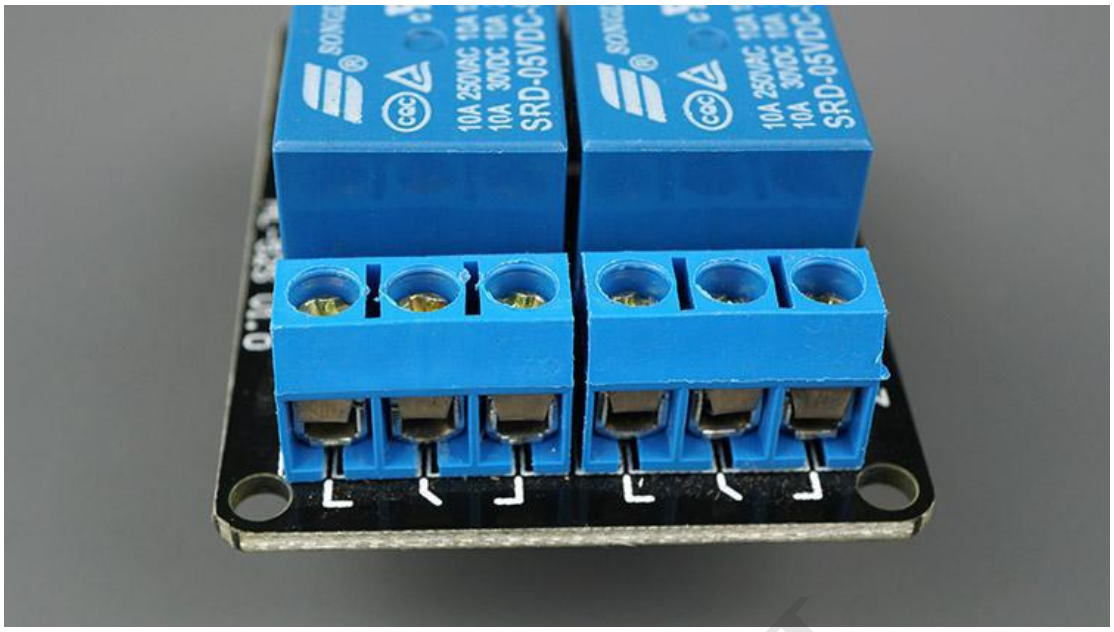
继电器是一种 电控开关，可通过控制信号开启或关闭，从而导通或切断电流。其核心优势在于：

- **低电压控制**：可用 **3.3V**（ESP32 GPIO 引脚输出电压）控制；
- **高电压负载**：支持 **12V、24V** 甚至市电电压(欧洲为 **230V**，美国为 **120V**)。




在左侧，设有 两组三个插座 用于连接 高电压；右侧的 针脚（低电压） 则连接至 ESP32 的 GPIO 引脚。

**市电电压连接**




上图中展示的继电器模块包含　两个连接器，每个连接器有三个接口：

- **公共端（COM）**
- **常闭端（NC）**
- **常开端（NO）**

**功能说明：**

♣ **COM**：连接需控制的 **市电电压**（如交流 220V/120V）。

♣ **NC（常闭端）：**

- **默认状态**：COM 与 NC 连通，电流流通；
- **触发操作**：当 ESP32 发送信号时，继电器断开电路，停止电流。
- ♣ **NO（常开端）：**
- **默认状态**：COM 与 NO 断开，电路无电流；
- **触发操作**：当 ESP32 发送信号时，继电器闭合电路，导通电流。

**控制引脚**

XINZHILI




低电压侧包含 两组引脚：

- **第一组**：
  - **VCC** 与 **GND**：为模块供电；
  - **输入 1（IN1）** 与 **输入 2（IN2）**：分别控制底部和顶部的继电器。

若继电器模块仅有 单通道，则仅有一个 IN 引脚；若为四通道，则有四个 IN 引脚，依此类推。

发送至 IN 引脚 的信号决定了继电器是否激活。当输入电压低于 2V 时，继电器触发。具体场景如下：

**♣ 常闭配置（NC）：**

- **高电平信号（HIGH）**：电流流通；
- **低电平信号（LOW）**：电流断开。

**♣ 常开配置（NO）：**

- **高电平信号（HIGH）**：电流断开；
- **低电平信号（LOW）**：电流流通。

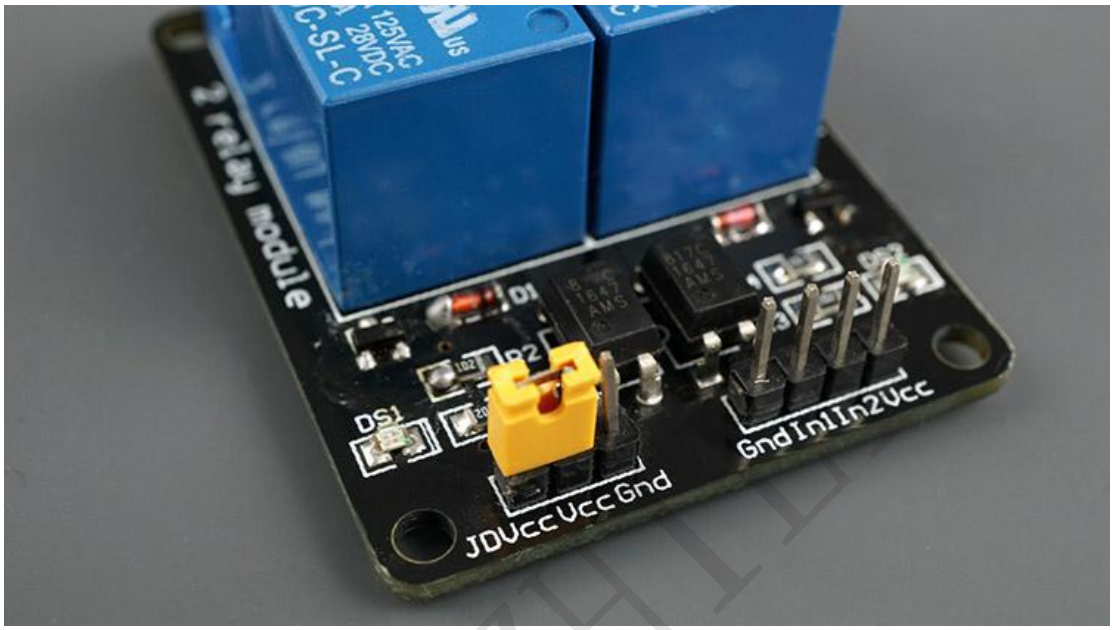
**配置选择建议**

- **常闭配置（NC）适用场景：**

XINZHILI

- 流需在 **大多数时间流通**，仅需偶尔断开时（例如紧急断电）。
- **常开配置（NO）适用场景：**
- 当电流需 **偶尔流通** 时（例如定时开启灯具）。

**电源选择**




**第二组引脚**

第二组引脚包含 **GND**、**VCC** 和 **JD-VCC**。

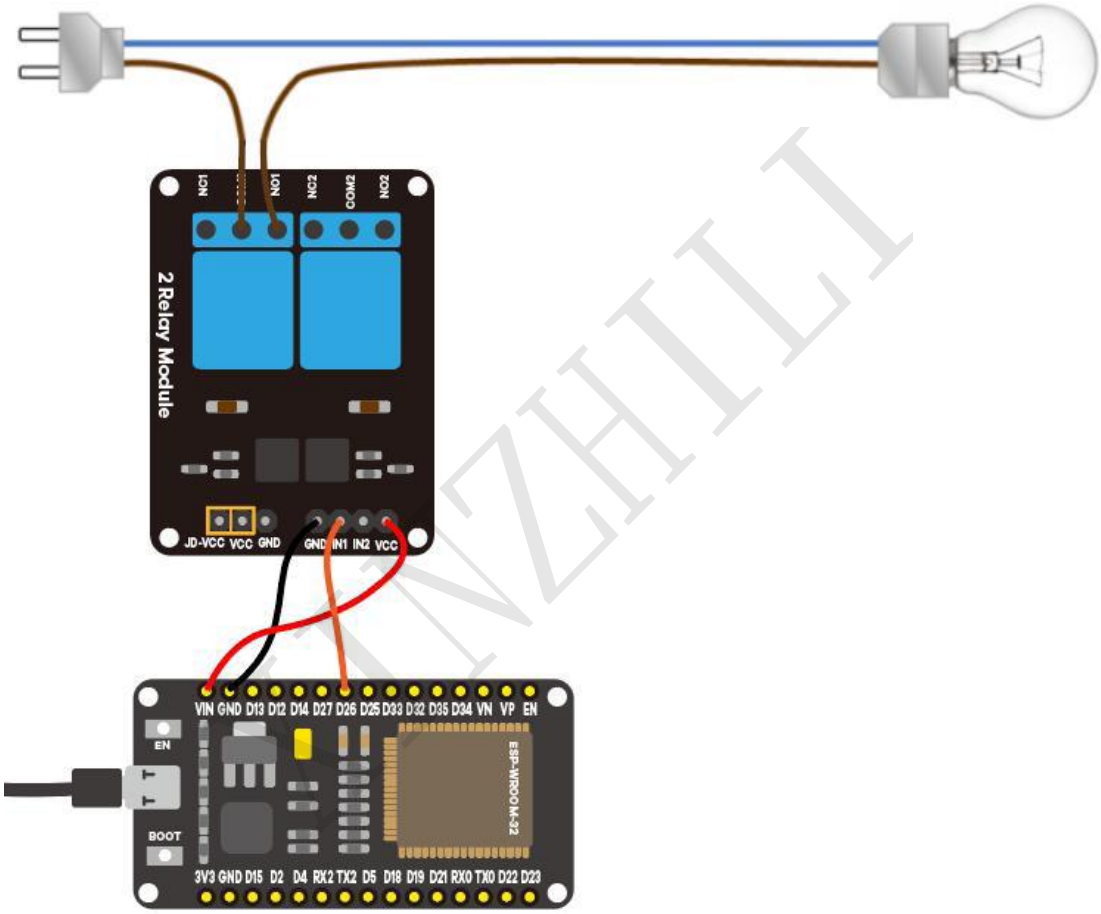
- **JD-VCC 引脚：**为继电器的 **电磁铁** 供电。
- **跳线帽：**模块上有一个连接 **VCC** 和 **JD-VCC** 的跳线帽（图中为黄色，实际颜色可能不同）。

**跳线帽连接状态**

1. **跳线帽安装时：**
    - **VCC** 与 **JD-VCC** 导通，电磁铁直接由 **ESP32 电源引脚** 供电。
    - **后果：**继电器模块与 ESP32 电路 **未物理隔离**。
2. **跳线帽移除时：**
    - 需通过 **JD-VCC** 引脚外接独立电源为电磁铁供电。
    - **优势：**

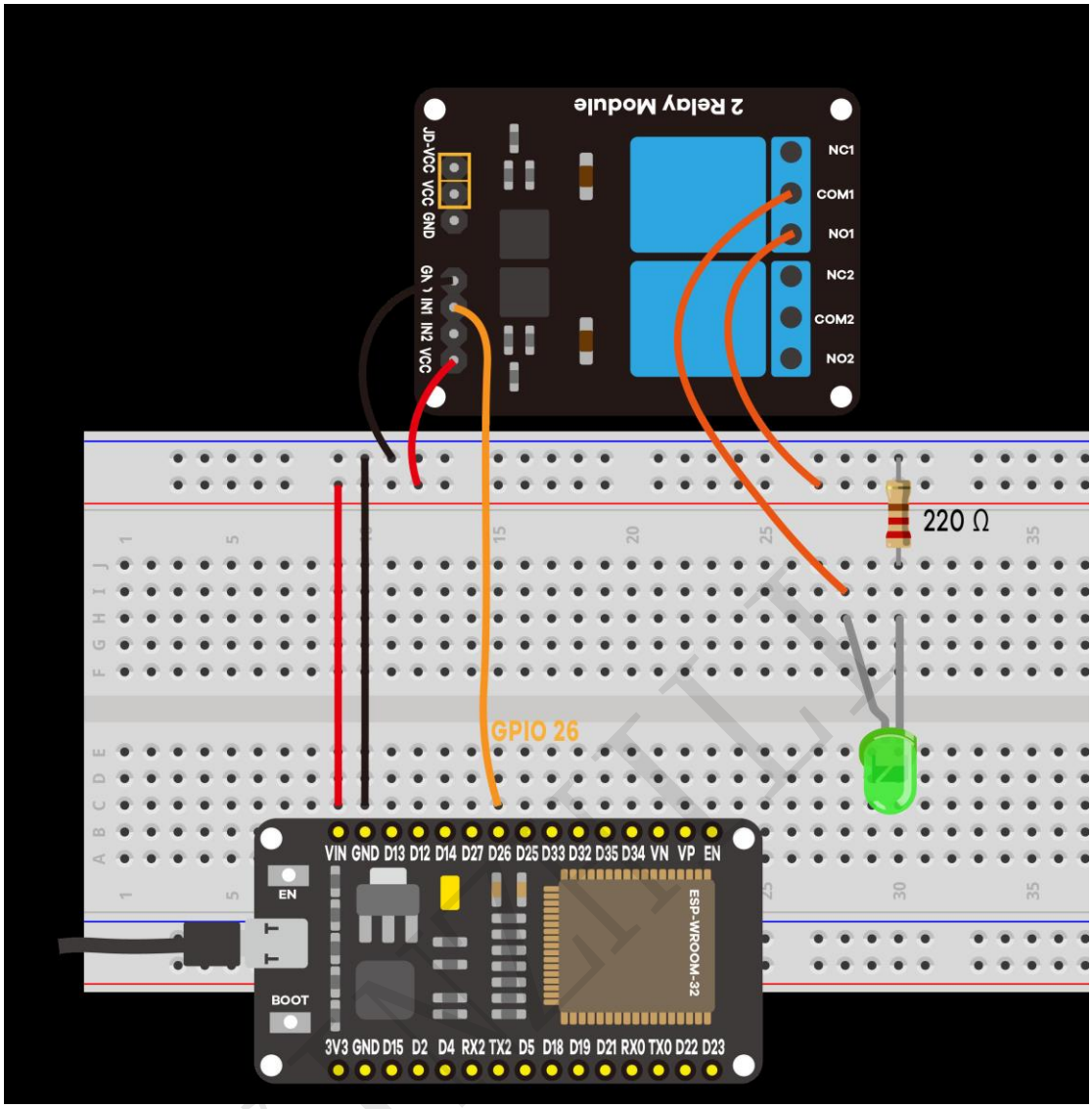
- 模块内置 **光耦** 实现继电器与 ESP32 的 **物理隔离**；
- 防止电涌等异常电压损坏 ESP32。

## 8.2 电路图




**警告：使用高电压电源可能导致严重伤害。**

因此，实验中用 5mm LED 代替高电压灯泡。若你不熟悉市电电压操作，请寻求熟悉者协助。在编程 ESP 或连接电路时，请确保所有设备已与市电电压断开。




**为 ESP32 安装库**

要构建此网络服务器，我们使用 ESPAsyncWebServer 库 和 AsyncTCP 库。

**安装 ESPAsyncWebServer 库**

按以下步骤安装 ESPAsyncWebServer 库：

1. 点击此处下载 ESPAsyncWebServer 库。你的下载文件夹中应该会有一个 **.zip** 文件夹。（**注：如果网络不好下载有问题也可以直接使用我们下好在资料包中的库文件夹**）
2. 解压 **.zip** 文件夹，你将得到 **ESPAsyncWebServer-master** 文件夹。

3. 将文件夹名称从 **ESPAsyncWebServer-master** 重命名为 **ESPAsyncWebServer**。
4. 将 **ESPAsyncWebServer** 文件夹移动到 Arduino IDE 安装目录的 **libraries** 文件夹中。

或者，在 Arduino IDE 中，你可以转到 项目 > 加载库 > 添加.ZIP 库…，然后选择你刚刚下载的库。

**为 ESP32 安装 AsyncTCP 库**

ESPAsyncWebServer 库需要 AsyncTCP 库 才能工作。按以下步骤安装该库：

1. 点击此处下载 AsyncTCP 库。你的下载文件夹中应该会有一个 **.zip** 文件夹。（**注：如果网络不好下载有问题也可以直接使用我们下好在资料包中的库文件夹**）
2. 解压 **.zip** 文件夹，你将得到 **AsyncTCP-master** 文件夹。
3. 将文件夹名称从 **AsyncTCP-master** 重命名为 **AsyncTCP**。
4. 将 **AsyncTCP** 文件夹移动到 Arduino IDE 安装目录的 **libraries** 文件夹中。
5. 最后，重新打开 Arduino IDE。

或者，在 Arduino IDE 中，你可以转到 项目 > 加载库 > 添加.ZIP 库…，然后选择你刚刚下载的库。

## 8.3 代码

我们将使用 Arduino IDE 对 ESP32 进行编程，因此请确保在继续操作前已安装 ESP32 插件：（如果已完成此步骤，可跳至下一步。）

**♣ 在 Arduino IDE 中安装 ESP32 插件**

安装所需库后，在 Arduino IDE 中打开资料包中的“**代码-->**

Project_7_ESP32_Relay_Web_Server” 文件夹中代码文件 Project_7_ESP32_Relay_Web_Server.ino。

上传代码前，请勿忘记填入你的网络凭证，以便 ESP32 连接到本地网络：

```
const char* ssid = "REPLACE_WITH_YOUR_SSID";
const char* password = "REPLACE_WITH_YOUR_SSID";
```

## 8.4 演示

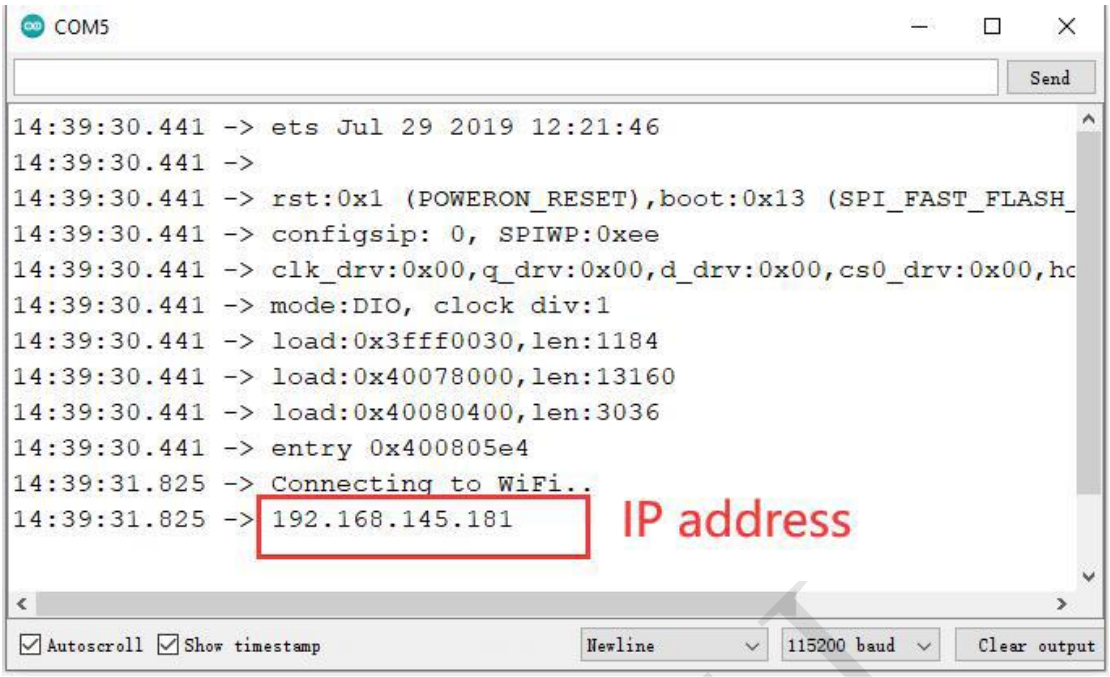
完成必要修改后，将代码上传至 ESP32。上传步骤参考：

1. 选择开发板：**工具 → 开发板 → ESP32 Dev Module**；
2. 选择端口：**工具 → 端口 → 对应 COM 号**；
3. 点击 **上传** 按钮。

上传完成后：

1. 打开波特率为 **115200** 的串行监视器，按下 **ESP32 EN 按钮** 以获取其 IP 地址；
2. 在本地网络的浏览器中输入 ESP32 的 IP 地址，即可访问网络服务器。

XINZHILI




注意：你的浏览器和 ESP32 必须连接到同一局域网（LAN）。

你将看到如下页面，其中包含 与代码中定义的继电器数量相同 的按钮（例如，若定义了两个继电器，则显示两个按钮）。



现在，你可以使用按钮通过智能手机控制你的继电器。



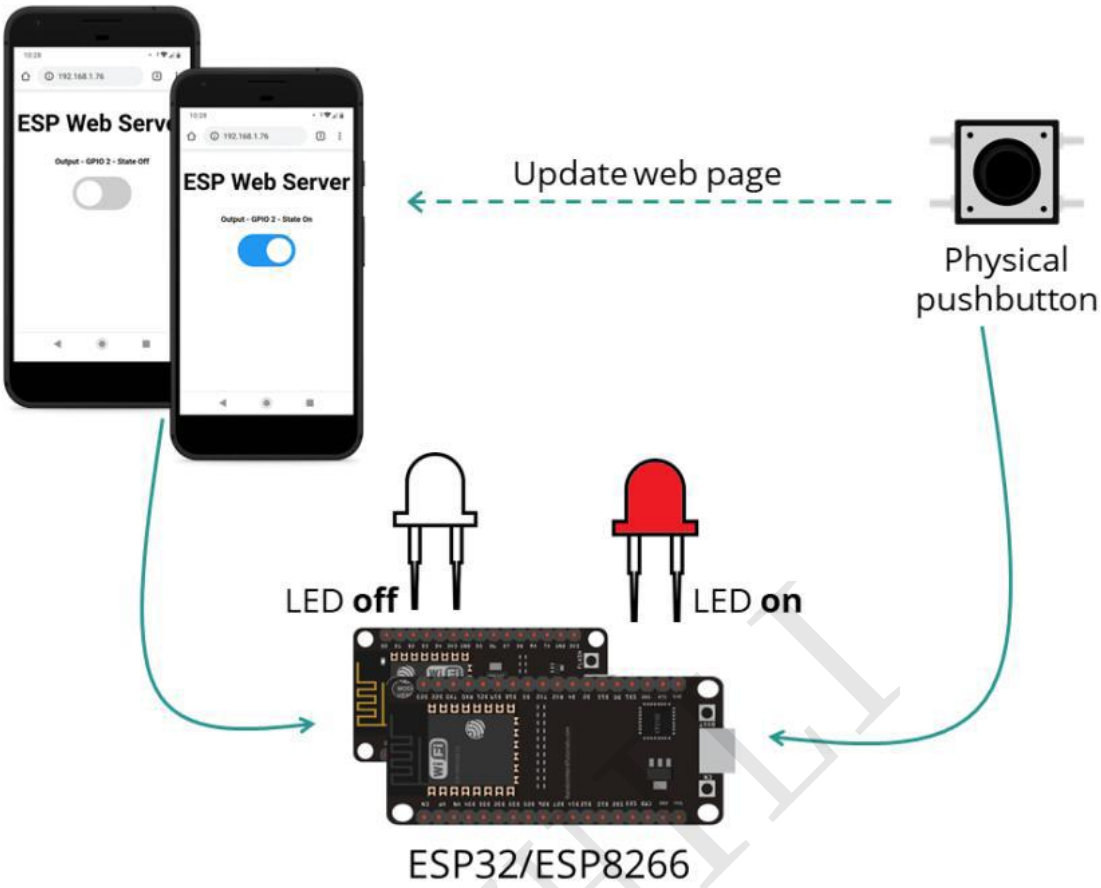

# 第9章 输出状态_同步_网络服务器

## 9.1 概述

本项目展示如何通过网页服务器与物理按钮同时控制 ESP32 或 ESP8266 的输出。无论通过物理按钮还是网页服务器更改输出状态，网页上的状态均会同步更新。

## 9.2 原理

让我们快速了解项目的工作原理：




♣ **ESP32 或 ESP8266** 托管一个网页服务器，允许控制输出状态；

♣ 当前输出状态显示在网页服务器上；

♣ ESP 同时连接物理按钮以控制同一输出；

♣ 若通过物理按钮更改输出状态，其当前状态也会在网页服务器上同步更新。

## 9.3 总结

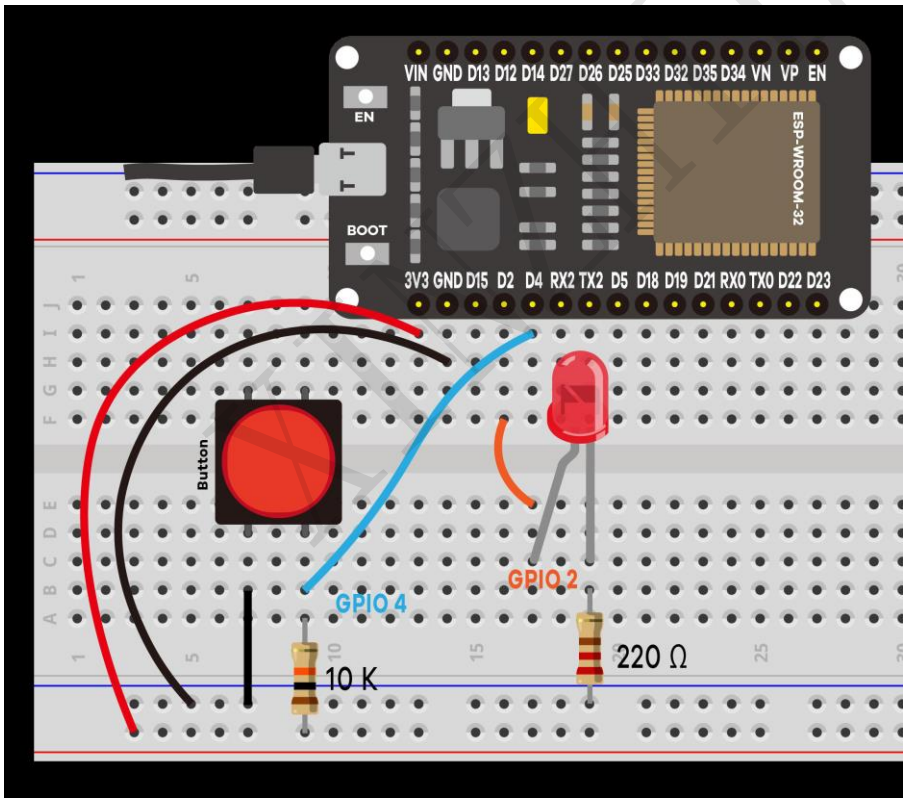
本项目允许通过 网页服务器 和 物理按钮 同步控制同一输出。无论输出状态如何变化，网页服务器均会实时更新。

## 9.4 所需元件

构建电路需以下元件：

- ESP32 DEVKIT V1 开发板
- 5 毫米 LED
- 220 欧姆电阻
- 按钮开关
- 10k 欧姆电阻
- 面包板
- 跳线

## 9.5 电路图




### 为 ESP32 安装库

要构建此网络服务器，我们使用 ESPAsyncWebServer 库和 AsyncTCP 库。（如果已完成此步骤，可跳至下一步。）

## 安装 ESPAsyncWebServer 库

按以下步骤安装 ESPAsyncWebServer 库：

1. 点击此处下载 ESPAsyncWebServer 库。你的下载文件夹中应该会有一个 **.zip** 文件夹。
2. 解压 **.zip** 文件夹，你将得到 **ESPAsyncWebServer-master** 文件夹。
3. 将文件夹名称从 **ESPAsyncWebServer-master** 重命名为 **ESPAsyncWebServer**。
4. 将 **ESPAsyncWebServer** 文件夹移动到 Arduino IDE 安装目录的 **libraries** 文件夹中。

或者，在 Arduino IDE 中，你可以转到 **项目 > 加载库 > 添加.ZIP 库…**，然后选择你刚刚下载的库。

## 为 ESP32 安装 AsyncTCP 库

ESPAsyncWebServer 库需要 AsyncTCP 库 才能工作。按以下步骤安装该库：

1. 点击此处下载 AsyncTCP 库。你的下载文件夹中应该会有一个 **.zip** 文件夹。
2. 解压 **.zip** 文件夹，你将得到 **AsyncTCP-master** 文件夹。
3. 将文件夹名称从 **AsyncTCP-master** 重命名为 **AsyncTCP**。
4. 将 **AsyncTCP** 文件夹移动到 Arduino IDE 安装目录的 **libraries** 文件夹中。
5. 最后，重新打开 Arduino IDE。

或者，在 Arduino IDE 中，你可以转到 **项目 > 加载库 > 添加.ZIP 库…**，

XINZHILI

然后选择你刚刚下载的库。

## 9.6 代码

我们将使用 Arduino IDE 对 ESP32 进行编程，因此请确保在继续操作前已安装 ESP32 插件：（如果已完成此步骤，可跳至下一步。）

### ♣ 在 Arduino IDE 中安装 ESP32 插件

安装所需库后，在 Arduino IDE 中打开资料包中的“**代码-->Project_8_Output_State_Synchronization_Web_Server**”文件夹中代码文件 Project_8_Output_State_Synchronization_Web_Server.ino。

上传代码前，请勿忘记填入你的网络凭证，以便 ESP32 连接到本地网络：

```
const char* ssid = "REPLACE_WITH_YOUR_SSID";
const char* password = "REPLACE_WITH_YOUR_SSID";
```

**代码工作原理**

**按钮状态与输出状态**

ledState 变量保存 LED 的输出状态。默认情况下，当网络服务器启动时，其状态为 LOW。

```
int ledState = LOW; // 输出引脚的当前状态
```

buttonState 和 lastButtonState 用于检测按钮是否被按下：

```
int buttonState;        // 输入引脚的当前读数
int lastButtonState = HIGH; // 输入引脚的上一次读数
```

**按钮（网页服务器）**

我们未在 index_html 变量中包含创建按钮的 HTML 代码。这是因为需要根据 LED 状态（可通过物理按钮更改）动态调整按钮显示。

因此，在代码中创建了占位符 %BUTTONPLACEHOLDER%，后续将通过 HTML 文本替换为实际按钮（此操作在 processor() 函数中完成）：

```
<h2>ESP Web Server</h2>
%BUTTONPLACEHOLDER%
```

运行 HTML

### processor() 函数

processor() 函数将 HTML 文本中的占位符替换为实际值。首先检查 HTML 是否包含 %BUTTONPLACEHOLDER% 占位符：

```
if(var == "BUTTONPLACEHOLDER"){
String outputStateValue = outputState();
```

随后，利用该值生成显示按钮状态的 HTML 文本：

```
buttons += "<h4>Output - GPIO 2 - State <span id=\"outputState\"><span></h4><label class=\"switch\"><input type=\"checkbox\" onchange=\"toggleCheckbox(this)\" id=\"output\" " + outputStateValue + "><span class=\"slider\"></span></label>";
```

运行 HTML

### 通过 HTTP GET 请求更改输出状态（JavaScript）

按下按钮时，调用 toggleCheckbox() 函数。该函数将向不同 URL 发送请求以开关 LED：

```
function toggleCheckbox(element) {
  var xhr = new XMLHttpRequest();
  if(element.checked) { xhr.open("GET", "/update?state=1", true); }
  else { xhr.open("GET", "/update?state=0", true); }
  xhr.send();
}
```

要开启 LED，需向 /update?state=1 URL 发送请求：

```
if(element.checked){ xhr.open("GET", "/update?state=1", true); }
else { xhr.open("GET", "/update?state=0", true); }
```

否则，它将向 /update?state=0 URL 发送请求。

### 通过 HTTP GET 请求更新状态（JavaScript）

为使网页服务器实时更新输出状态，需每秒向 /state URL 发送请求：

```
setInterval(function ( ) {
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
      var inputChecked;
      var outputStateM;
      if( this.responseText == 1){
        inputChecked = true;
        outputStateM = "On";
      }
      else {
        inputChecked = false;
        outputStateM = "Off";
      }
      document.getElementById("output").checked = inputChecked;
      document.getElementById("outputState").innerHTML = outputState
M;
    }
  };
  xhttp.open("GET", "/state", true);
  xhttp.send();
}, 1000 );
```

**请求处理**

接下来，需定义 ESP32/ESP8266 收到请求时的处理逻辑：

当收到根路径 / 的请求时，发送 HTML 页面并调用 processor 函数：

```
server.on("/", HTTP_GET, [](AsyncWebServerRequest *request){
  request->send_P(200, "text/html", index_html, processor);
});
```

当收到 /update?state=1 或 /update?state=0 URL 的请求时，以下代码检查请求并相应更新 ledState：

```
server.on("/update", HTTP_GET, [] (AsyncWebServerRequest *request) {

  String inputMessage;
  String inputParam;
  // 在 <ESP_IP>/update?state=<inputMessage> 获取 input1 的值
  if (request->hasParam(PARAM_INPUT_1)) {
```

```
    inputMessage = request->getParam(PARAM_INPUT_1)->value();
    inputParam = PARAM_INPUT_1;
    digitalWrite(output, inputMessage.toInt());
    ledState = !ledState;
  }
  else {
    inputMessage = "未发送消息";
    **114/137**
    inputParam = "无";
  }
  Serial.println(inputMessage);
  request->send(200, "text/plain", "OK");
});
```

当收到 /state URL 的请求时，发送当前输出状态：

```
server.on("/state", HTTP_GET, [] (AsyncWebServerRequest *request) {
  request->send(200, "text/plain", String(digitalRead(output)).c_str());
});
```

**loop() 函数**

在 loop() 中，对按钮进行防抖动处理，并根据 ledState 变量的值控制 LED 开关：

```
• digitalWrite(output, ledState);
```

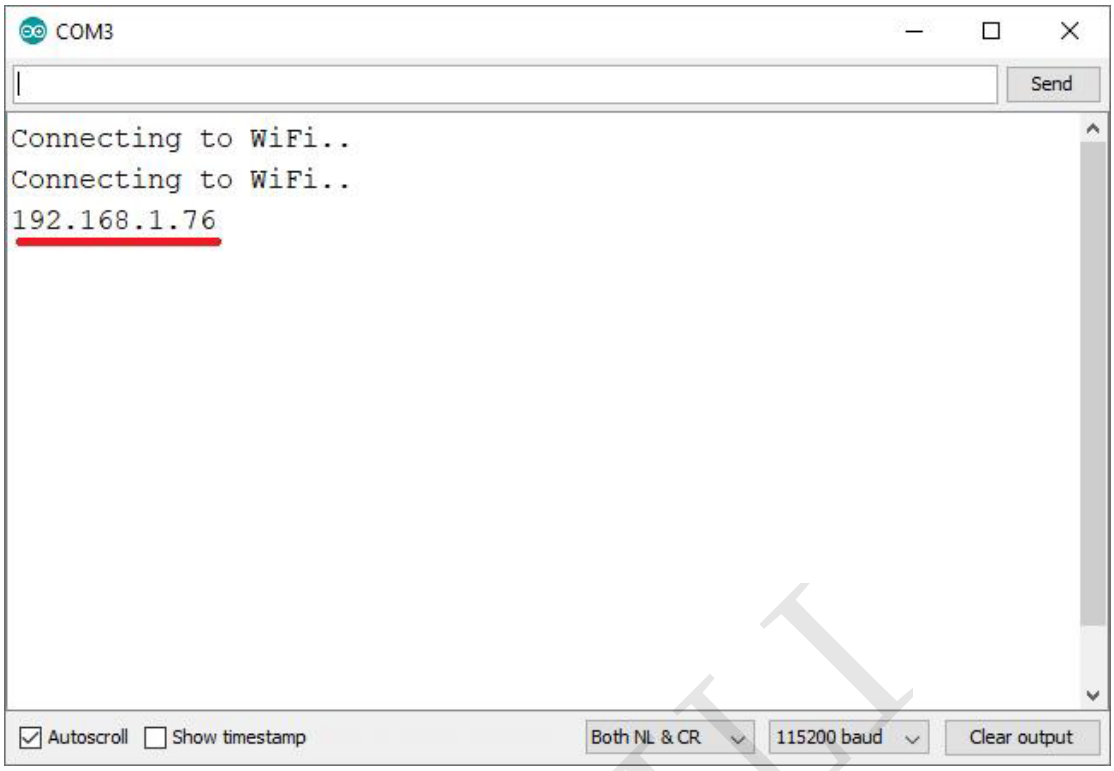
## 9.7 演示

将代码上传至你的 ESP32 开发板。上传步骤参考：

1. 选择开发板：**工具 → 开发板 → ESP32 Dev Module**；
2. 选择端口：**工具 → 端口 → 对应 COM 号**；
3. 点击 **上传** 按钮。

上传完成后：

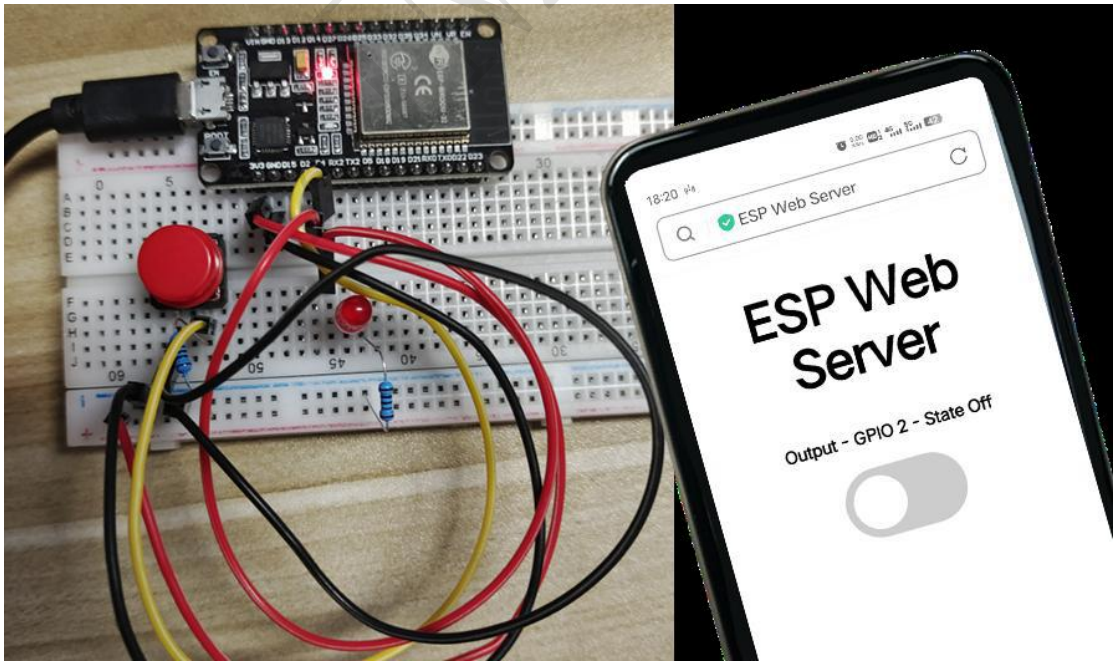
1. 打开波特率为 **115200** 的串行监视器；
2. 按下开发板上的 **EN/RST 按钮** 以获取其 **IP 地址**。




打开本地网络中的浏览器，输入 ESP32 的 IP 地址。

你将能够访问如下所示的网页服务器。

注意：你的浏览器与 ESP32 必须连接到同一局域网（LAN）。




你可以切换网页服务器上的按钮以打开 LED。




你也可以通过物理按钮控制同一个 LED。其状态将始终在网页服务器上自动更新。

# 第10章　DHT11 网络服务器

## 10.1 概述

在本项目中，你将学习如何使用 Arduino IDE 构建基于 DHT11 的异步 ESP32 网络服务器，以显示温湿度数据。

我们将构建的网络服务器能够　自动更新读数，无需手动刷新页面。

通过本项目，你将学习：

- ♣ 如何从 DHT 传感器读取温湿度；
- ♣ 使用 **ESPAsyncWebServer 库** 构建异步网络服务器；
- ♣ 实现传感器读数自动更新，无需页面刷新。

XINZHILI

## 10.2 异步网络服务器

为构建此服务器，我们将使用 ESPAsyncWebServer 库，该库提供了便捷的异步服务器搭建方式。异步服务器的优势包括（摘自库的 GitHub 页面）：

♣ “同时处理多个连接”；

♣ “发送响应后立即准备处理新连接，服务器在后台处理响应传输”；

♣ “内置简易模板处理引擎”；

## 10.3 所需元件

完成本教程需以下元件：

♣ **ESP32 开发板**

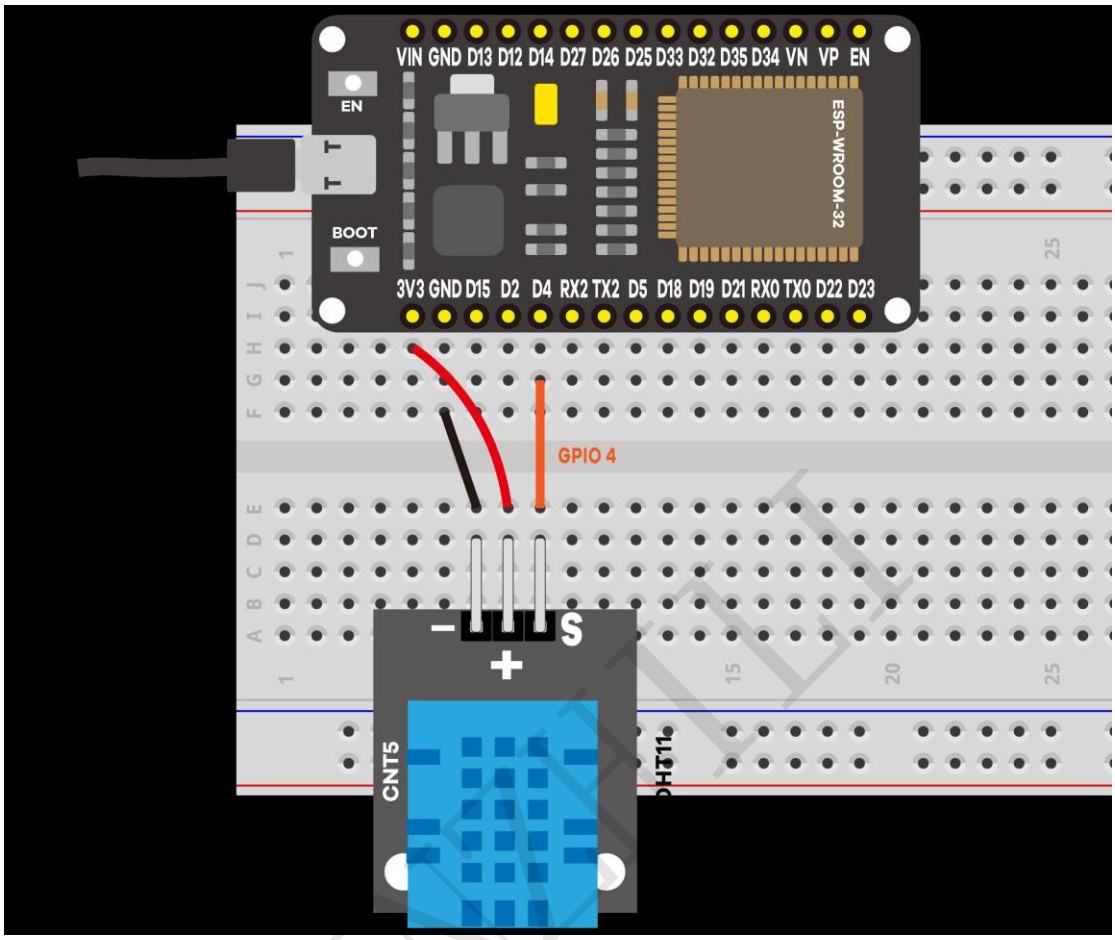
♣ **DHT11 模块**

♣ **面包板**

♣ **跳线**

XINZHILI

## 10.4 电路图




## 10.5 安装库

本项目需安装以下库：

♣ **DHT 传感器库** 与 **Adafruit 统一传感器驱动库**，用于读取 DHT 传感器数据；

♣ **ESPAsyncWebServer 库** 与 **Async TCP 库**，用于构建异步网络服务器。

按以下步骤安装这些库：

### 10.5.1 安装 DHT 传感器库

通过 Arduino IDE 读取 DHT 传感器数据需安装 DHT 传感器库。步骤如下：

1. 点击此处下载 DHT 传感器库。你的下载文件夹中将出现 **.zip** 文件；
2. 解压 **.zip** 文件，得到 **DHT-sensor-library-master** 文件夹；
3. 将文件夹重命名为 **DHT_sensor**；
4. 将 **DHT_sensor** 文件夹移动到 Arduino IDE 安装目录的 **libraries** 文件夹中；
5. 最后，重新打开 Arduino IDE。

### 10.5.2 安装 Adafruit 统一传感器驱动库

使用 DHT 传感器还需安装 Adafruit 统一传感器驱动库。步骤如下：

1. 点击此处下载 Adafruit 统一传感器库。你的下载文件夹中将出现 **.zip** 文件；
2. 解压 **.zip** 文件，得到 **Adafruit_sensor-master** 文件夹；
3. 将文件夹重命名为 **Adafruit_sensor**；
4. 将 **Adafruit_sensor** 文件夹移动到 Arduino IDE 安装目录的 **libraries** 文件夹中；
5. 最后，重新打开 Arduino IDE。

### 10.5.3 安装 ESPAsyncWebServer 库

按以下步骤安装 ESPAsyncWebServer 库：

1. 点击此处下载 ESPAsyncWebServer 库。你的下载文件夹中将出现 **.zip** 文件；
2. 解压 **.zip** 文件，得到 **ESPAsyncWebServer-master** 文件夹；
3. 将文件夹重命名为 **ESPAsyncWebServer**；
4. 将 **ESPAsyncWebServer** 文件夹移动到 Arduino IDE 安装目录的 **libraries** 文件夹中。

XINZHILI

### 10.5.4 为 ESP32 安装 Async TCP 库

ESPAsyncWebServer 库需 Async TCP 库 支持。步骤如下：

1. 点击此处下载 Async TCP 库。你的下载文件夹中将出现 **.zip** 文件；
2. 解压 **.zip** 文件，得到 **AsyncTCP-master** 文件夹；
3. 将文件夹重命名为 **AsyncTCP**；
4. 将 **AsyncTCP** 文件夹移动到 Arduino IDE 安装目录的 **libraries** 文件夹中；
5. 最后，重新打开 Arduino IDE。

## 10.6 代码

我们将使用 Arduino IDE 对 ESP32 进行编程，因此请确保在继续操作前已安装 ESP32 插件：（如果已完成此步骤，可跳至下一步。）

♣ 在 **Arduino IDE** 中安装 **ESP32** 插件

安装所需库后，在 Arduino IDE 中打开资料包中的“**代码 --> Project_9_ESP32_DHT11_Web_Server**” 文件夹中代码文件 Project_9_ESP32_DHT11_Web_Server.ino。

上传代码前，请勿忘记填入你的网络凭证，以便 ESP32 连接到本地网络：

```
const char* ssid = "REPLACE_WITH_YOUR_SSID";
const char* password = "REPLACE_WITH_YOUR_SSID";
```

**代码工作原理**

以下段落将解释代码的运行机制。如需了解更多细节，请继续阅读；若想直接查看演示结果，可跳转至 演示 部分。

**导入库**

首先导入必要的库：

- **WiFi**、**ESPAsyncWebServer** 和 **ESPAsyncTCP** 用于构建网络服务器；
- **Adafruit_Sensor** 和 **DHT** 用于读取 DHT11/DHT22 传感器数据。

```
#include "WiFi.h"
**122/137**
#include "ESPAsyncWebServer.h"
#include <ESPAsyncTCP.h>
#include <Adafruit_Sensor.h>
#include <DHT.h>
```

变量定义

定义 DHT 数据引脚连接的 GPIO（本例为 GPIO 4）：

```
#define DHTPIN 4 // 连接到 DHT 传感器的数字引脚
```

选择使用的 DHT 传感器类型（本例为 DHT11）。若使用其他型号，需取消对应注释并注释其他型号：

```
#define DHTTYPE DHT11 // DHT 11
```

实例化 DHT 对象（基于已定义的引脚和传感器类型）：

```
DHT dht(DHTPIN, DHTTYPE);
```

创建端口 80 的 AsyncWebServer 对象：

```
AsyncWebServer server(80);
```

温湿度读取函数

我们创建了两个函数：一个用于读取温度（readDHTTemperature()），另一个用于读取湿度（readDHTHumidity()）。

```
String readDHTTemperature() {
  // 传感器读数可能延迟最多 2 秒（DHT 为低速传感器）
  // 读取摄氏温度（默认）
  float t = dht.readTemperature();
  // 读取华氏温度（设置 isFahrenheit = true）
  //float t = dht.readTemperature(true);

  // 检测读取失败并提前返回（以便重试）
  if (isnan(t)) {
    Serial.println("无法从 DHT 传感器读取数据！");
```

```
    return "--";
  }
  else {
    Serial.println(t);
    return String(t);
  }
}
```

获取传感器读数只需调用 dht 对象的 readTemperature() 和 readHumidity() 方法：

```
float t = dht.readTemperature();
float h = dht.readHumidity();
```

若传感器读取失败，函数将返回双横线 --：

```
if (isnan(t)) {
  Serial.println("无法从 DHT 传感器读取数据！");
  return "--";
}
```

读数以字符串形式返回。使用 String() 函数将浮点数转换为字符串：

```
return String(t);
```

默认读取摄氏温度。如需读取华氏温度，请注释掉摄氏部分并取消华氏部分的注释，如下所示：

```
float t = dht.readTemperature();
// 读取华氏温度（设置 isFahrenheit = true）
//float t = dht.readTemperature(true);
```

---

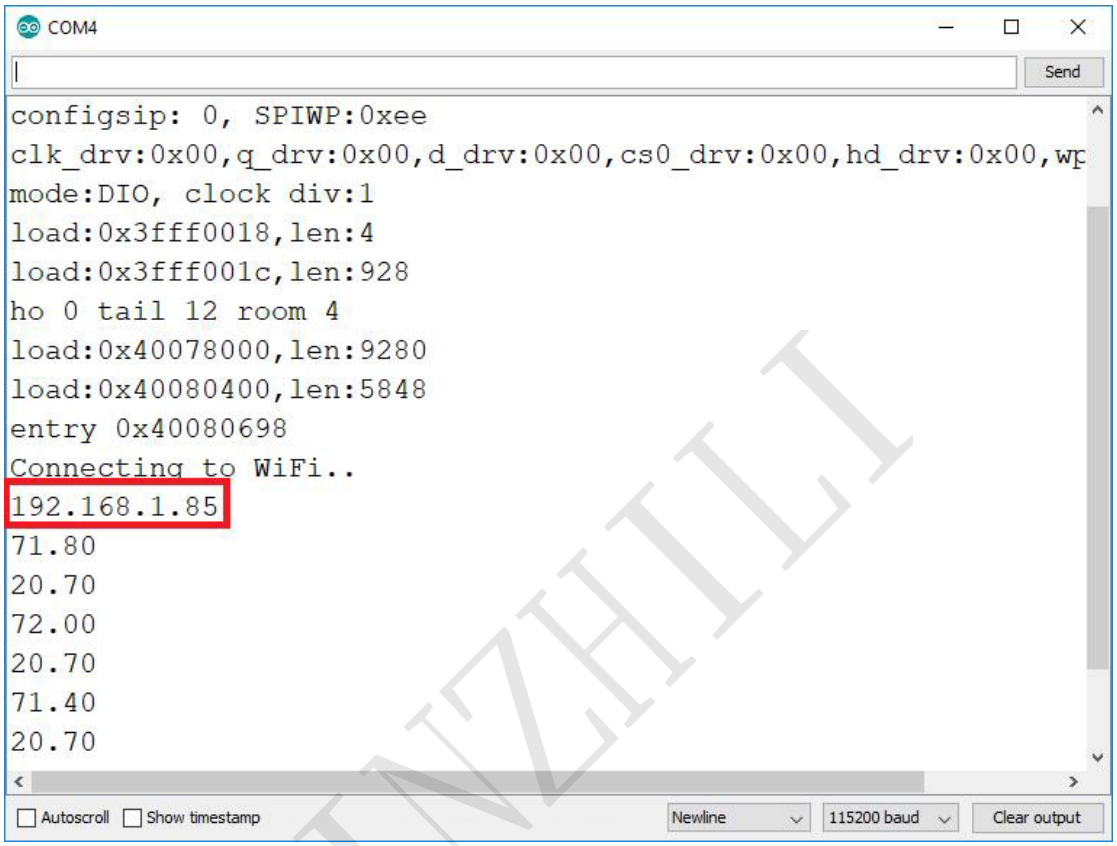
## 上传代码

现在，将代码上传至你的 ESP32。确保选择正确的开发板和 COM 端口。上传步骤参考：

1. 选择开发板：**工具 → 开发板 → ESP32 Dev Module**；
2. 选择端口：**工具 → 端口 → 对应 COM 号**；
3. 点击 **上传** 按钮。

上传完成后：

1. 打开波特率为 **115200** 的串行监视器；
2. 按下 **ESP32 复位按钮**，其 **IP 地址** 将显示在串行监视器中。



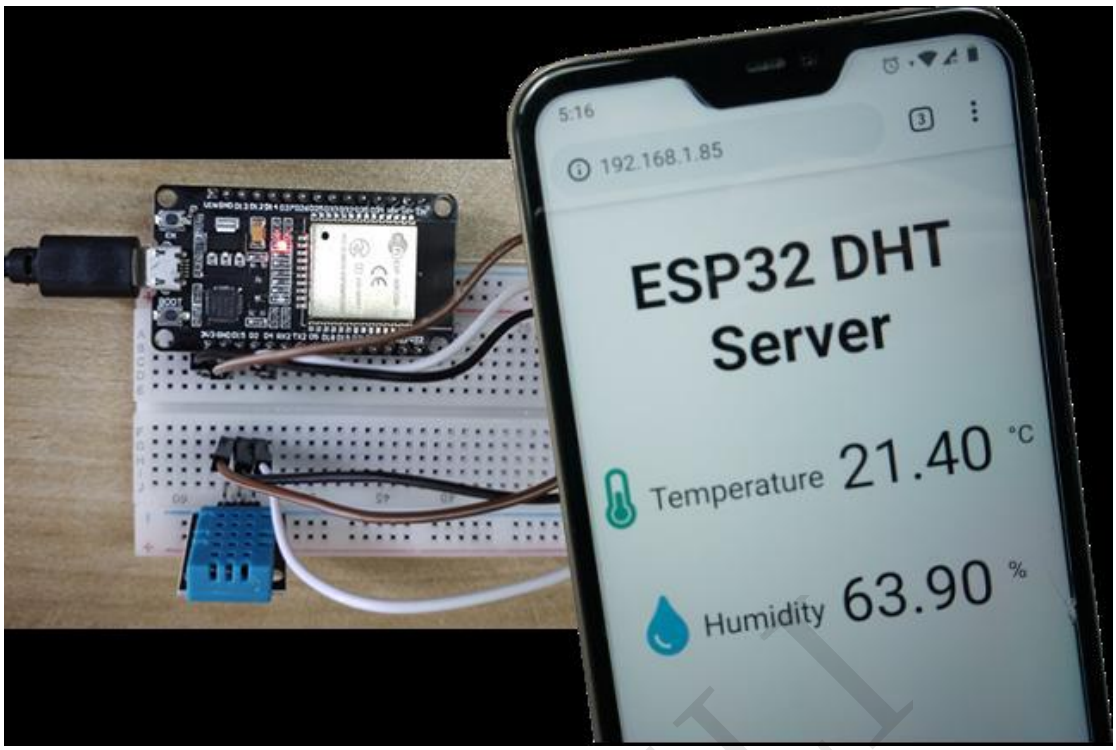

XINZHILI

## 10.7 演示

打开浏览器并输入 ESP32 的 IP 地址。网页服务器将显示最新的传感器读数。

注意：你的浏览器与 ESP32 必须连接到同一局域网（LAN）。

请注意，温度和湿度读数会自动更新，无需手动刷新网页。




# 第11章　ESP32 OLED 显示屏

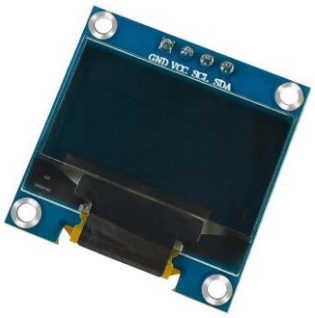
## 11.1 概述

本项目展示如何通过 Arduino IDE 使用 ESP32 驱动 0.96 英寸 SSD1306 OLED 显示屏。

## 11.2 0.96 英寸 OLED 显示屏介绍

本教程使用的 OLED 显示屏型号为 SSD1306：这是一款 单色、0.96 英寸、128×64 像素 的显示屏，如下图所示。

XINZHILI



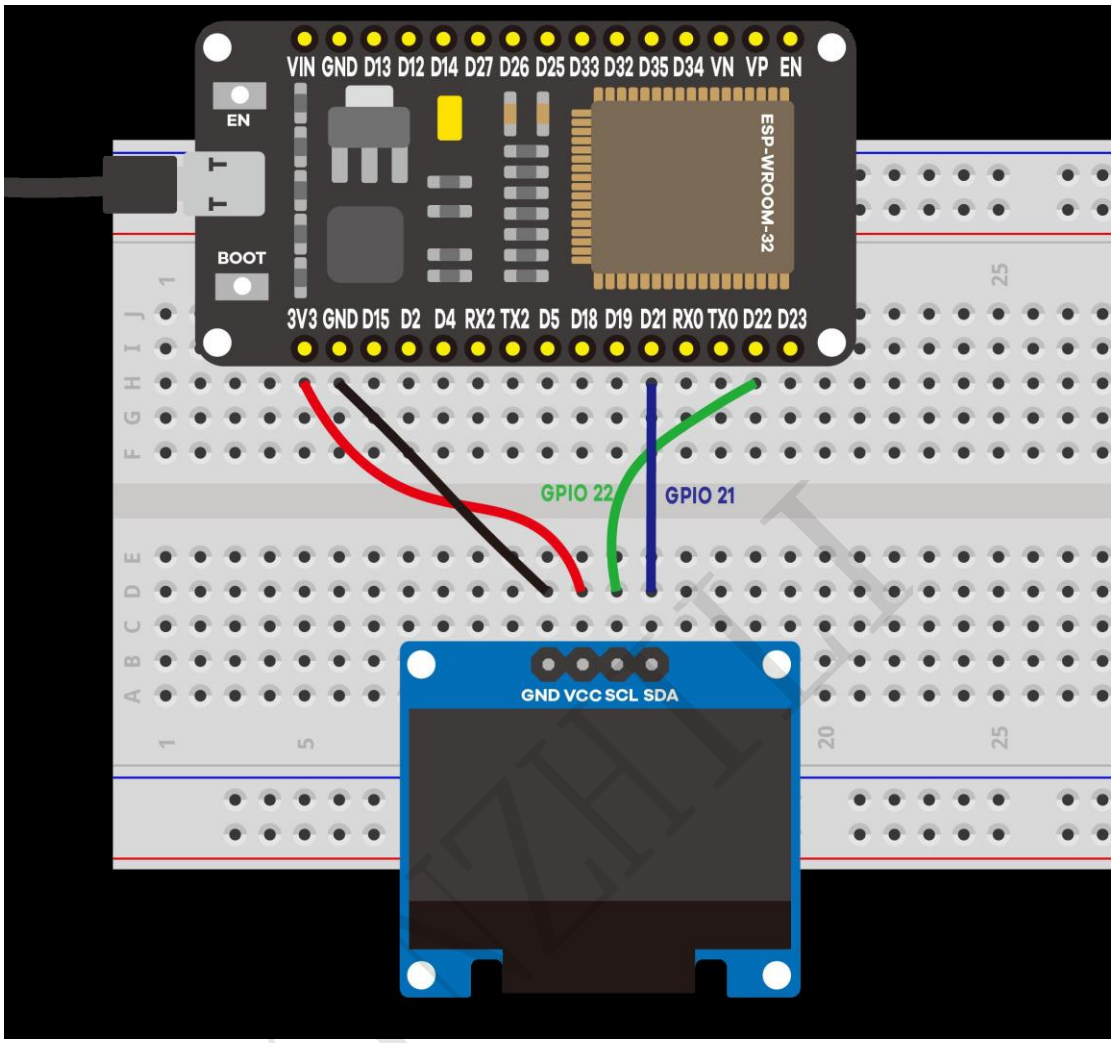

OLED 显示屏无需背光，因此在暗光环境下具有出色的对比度。此外，其像素仅在点亮时消耗电能，因此与其他显示屏相比，OLED 的功耗更低。

由于 OLED 显示屏采用 I2C 通信协议，接线非常简单。您可参考以下表格进行连接：

| OLED Pin | ESP32 |
| --- | --- |
| Vin | 3.3V |
| GND | GND |
| SCL | GPIO 22 |
| SDA | GPIO 21 |

XINZHILI

## 11.3 电路图



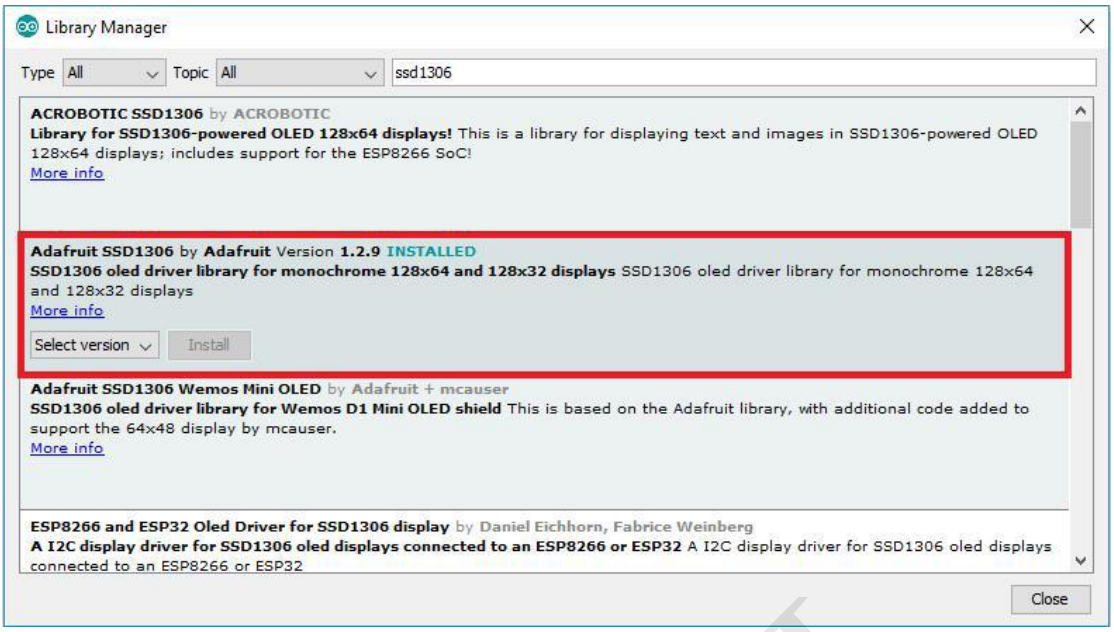

**为 ESP32 安装 SSD1306 OLED 库**

有多个库可用于通过 ESP32 控制 OLED 显示屏。本教程使用以下两个 Adafruit 库：

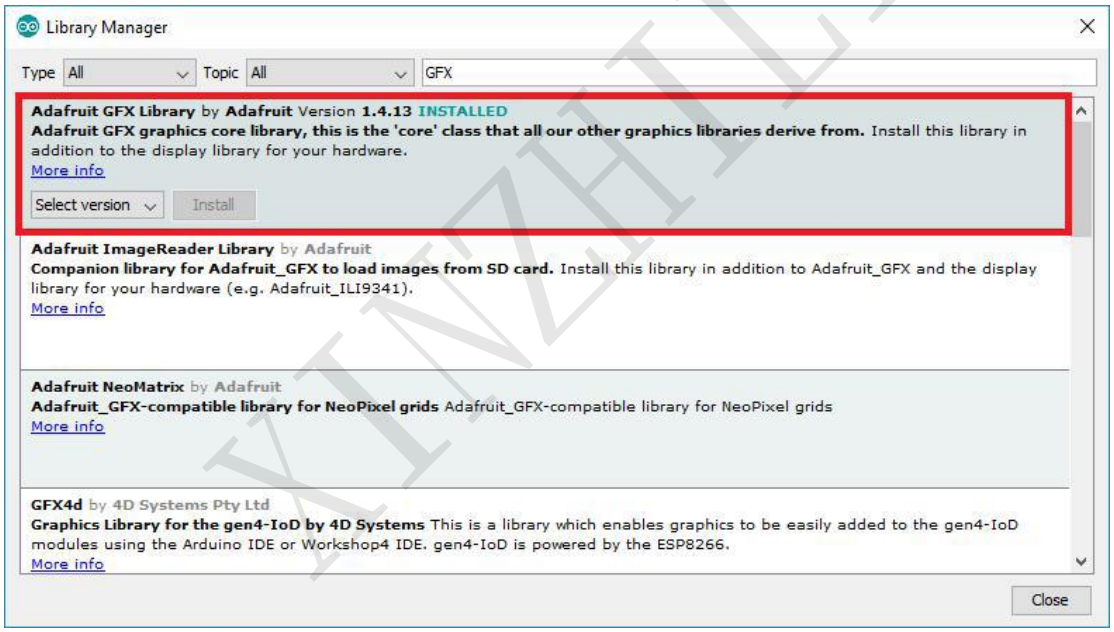
- **Adafruit_SSD1306 库**
- **Adafruit_GFX 库**

按以下步骤安装这些库：

1. 打开 Arduino IDE，转到 **项目 → 加载库 → 管理库**，库管理器将打开。
2. 在搜索框中输入 **“SSD1306”**，安装 Adafruit 提供的 **SSD1306 库**




**3. 安装 Adafruit 的 SSD1306 库后，在搜索框中输入“GFX”并安装该库。**




**4. 安装库后，重启你的 Arduino IDE。**

## 11.4 代码

安装所需库后，在 Arduino IDE 中打开资料包中的“**代码-->Project_10_ESP32_OLED_Display**”文件夹中代码文件 Project_10_ESP32_OLED_Display.ino。

我们将使用 Arduino IDE 对 ESP32 进行编程，因此请确保在继续操作前已安装 ESP32 插件：（如果已完成此步骤，可跳至下一步。）

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#define SCREEN_WIDTH 128 // OLED display width, in pixels
#define SCREEN_HEIGHT 64 // OLED display height, in pixels
// Declaration for an SSD1306 display connected to I2C (SDA, SCL pins)
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire,-1);
void setup() {
    Serial.begin(115200);
    if(!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) { // Address 0x3D for
128x64
        Serial.println(F("SSD1306 allocation failed"));
        for(;;);
    }
    delay(2000);
    display.clearDisplay();
    display.setTextSize(2);
    display.setTextColor(WHITE);
    display.setCursor(0, 30);
    // Display static text
    display.println("LAFVIN");
    display.display();
    delay(100);
}

void loop() {
    // Scroll in various directions, pausing in-between:
    133/137
    display.startscrollright(0x00, 0x0F);
    delay(7000);
    display.stopscroll();
    delay(1000);
    display.startscrollleft(0x00, 0x0F);
    delay(7000);
    display.stopscroll();
    delay(1000);
}
```

## 11.5 代码工作原理

### 导入库

首先需导入必要的库：

- **Wire** 库用于 I2C 通信；
- **Adafruit** 库（Adafruit_GFX 和 Adafruit_SSD1306）用于控制显示屏。

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
```

### 初始化 OLED 显示屏

定义 OLED 的宽度与高度（本例为 128×64 像素）。若使用其他尺寸，需修改 SCREEN_WIDTH 和 SCREEN_HEIGHT：

```
#define SCREEN_WIDTH 128 // OLED 显示屏宽度（像素）
#define SCREEN_HEIGHT 64 // OLED 显示屏高度（像素）
```

通过 I2C 协议初始化显示屏对象（&Wire）：

```
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, -1);
```

- **参数说明：**
  - -1 表示 OLED 无复位引脚。若有复位引脚，需将其连接至 GPIO 并将引脚号作为参数传入。

在 setup() 中，初始化波特率为 115200 的串行监视器（用于调试）：

```
Serial.begin(115200);
```

调用 begin() 方法初始化 OLED 显示屏：

```
if(!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {
  **135/137**
  Serial.println("SSD1306 分配失败");
  for(;;); // 停止执行，无限循环
}
```

- **OLED 地址**：若使用其他型号显示屏，需修改地址（本例为 **0x3C**）。

初始化完成后，延迟 2 秒 以确保显示屏就绪：

```
delay(2000);
```

清除显示、设置字体大小、颜色和写入文本

初始化后，清除显示缓冲区：

```
display.clearDisplay();
```

设置文本属性：

1. 字体大小：

```
display.setTextSize(1);
```

2. 字体颜色（白色字体，黑色背景）：

```
display.setTextColor(WHITE);
```

3. 文本起始位置（左上角坐标 (0,0)）：

```
display.setCursor(0, 0);
```

写入文本并显示：

```
display.println("Hello, world!");
display.display();
```

**滚动文本功能：**

♣ startscrollright(0x00, 0x0F)：**从左向右滚动**

♣ startscrollleft(0x00, 0x0F)：**从右向左滚动**

♣ startscrolldiagright(0x00, 0x07)：**左下角向右上角滚动**

♣ startscrolldiagleft(0x00, 0x07)：**右下角向左上角滚动**

**上传代码**

现在将代码上传至 ESP32。上传步骤参考：

1. 选择开发板：**工具 → 开发板 → ESP32 Dev Module**；

2. 选择端口：**工具 → 端口 → 对应 COM 号**；

3. 点击 **上传** 按钮。

上传代码后，OLED 将显示滚动文本。

