Arduino嵌入式讲义

西安深蓝软件开发实践培训中心

目录

[实验介绍 3](#_Toc516141483)

[教学目标 4](#_Toc516141484)

[Arduino开发基础 4](#_Toc516141485)

[Arduino简介 4](#_Toc516141486)

[安装驱动 5](#_Toc516141487)

[NodeMcu介绍 7](#_Toc516141488)

[引脚映射 8](#_Toc516141489)

[GPIO序号 9](#_Toc516141490)

[ESP8266开发方式 10](#_Toc516141491)

[Arduino语言说明 19](#_Toc516141492)

[实验1 22](#_Toc516141493)

[点亮蓝灯 22](#_Toc516141494)

[观察串口监视器 23](#_Toc516141495)

[点亮Led灯 25](#_Toc516141496)

[Led灯亮灭 26](#_Toc516141497)

[跑马灯 29](#_Toc516141498)

[观察电位器的值 30](#_Toc516141499)

[电位器控制Led的亮度 33](#_Toc516141500)

[实验2 35](#_Toc516141501)

[7段数码管显示 35](#_Toc516141502)

[例子1：显示数字 37](#_Toc516141503)

[例子2：秒表显示（0-9） 39](#_Toc516141504)

[例子3：红绿灯显示 41](#_Toc516141505)

[4位数码管 43](#_Toc516141506)

[例子1：16进制计数器 44](#_Toc516141507)

[例子2：4位数码管计数器 48](#_Toc516141508)

[例子3：秒表 51](#_Toc516141509)

[LCD显示模块 57](#_Toc516141510)

[实验3 64](#_Toc516141511)

[按键控制小灯 64](#_Toc516141512)

[例1：按下亮，弹起灭 64](#_Toc516141513)

[例2：按一下灭，按一下亮 66](#_Toc516141514)

[按键抢答器 67](#_Toc516141515)

[发声模块 69](#_Toc516141516)

[发声 70](#_Toc516141517)

[播放乐谱 71](#_Toc516141518)

[LED灯伴随音乐闪烁 76](#_Toc516141519)

[实验4 78](#_Toc516141520)

[74HC138 3-8译码器 78](#_Toc516141521)

[RGB 三基色 LED 82](#_Toc516141522)

[例子1 82](#_Toc516141523)

[例子2 83](#_Toc516141524)

[例子3 85](#_Toc516141525)

[舵机 86](#_Toc516141526)

[例子1：180度转舵 88](#_Toc516141527)

[例子2：电位器控制舵机 89](#_Toc516141528)

[实验5 90](#_Toc516141529)

[红外遥控 90](#_Toc516141530)

[红外按键的编码 91](#_Toc516141531)

[红外遥控控制灯亮与灭 94](#_Toc516141532)

[DHT11温湿度传感器 95](#_Toc516141533)

[倾斜开关 101](#_Toc516141534)

[超声波测距 102](#_Toc516141535)

[例子1：测距 105](#_Toc516141536)

[例子2：测距，亮灯提示 108](#_Toc516141537)

[步进电机使用 109](#_Toc516141538)

[代码1 111](#_Toc516141539)

[代码2 112](#_Toc516141540)

[实验6 113](#_Toc516141541)

[通过Wi-Fi + AP + HTTP服务器，实现本地数据发布 113](#_Toc516141542)

[实验7 117](#_Toc516141543)

[通过Wi-Fi + AP + HTTP服务器，实现本地传感器控制 117](#_Toc516141544)

[实验8 122](#_Toc516141545)

[通过MQTT客户端完成与IOT的互联 122](#_Toc516141546)

# 实验介绍

随着物联网的迅猛发展，各类家用、工业设备逐步趋于智能化，为了实现这一点，单片机功不可没。Arduino 平台的基础就是AVR指令集的单片机,Arduino 是一款价格不高、易于使用的电子平台。包括硬件和软件在内的整个平台是完全开源的，并且使用的是松散地基于 C/C++ 的语言,非常容易学习和使用。本次实训选用的就是Arduino 平台。

实训的内容涉及基础部分和项目部分，项目选择目前非常成熟的物联网应用，通过一个基础的气象监测系统，来学习Arduino 平台的使用。气象监测系统是根据对局部气象环境监测而设计的多要素微气象监测装置，将采集到的各种气象参数及其变化状况，通过现代网络技术实时的传送到中心监控分析系统，当出现异常情况时，系统会以多种方式发出预报警信息，提示管理人员应对报警点予以重视或采取必要的预防措施。

实训项目实现真实气象检测系统的部分功能，项目基于开源电子平台Arduino包含硬件(Arduino UNO板)和软件(Arduino IDE)，ESP8266-12E开发板,NodeMCU开发环境，通过使用环境检测的传感器实现数据的采集检测，然后通过WIFI网络,发布实时的气象检测数据。

# 教学目标

采用“基础知识+模块实验+项目实战”的教学方式进行教学，采用灵活的授课手段，在课堂上可以根据实际情况取舍教学内容，动态调整，使学生学习使用不同传感器采集数据、网络传输、显示数据以及控制显示设备等，所需知识涉及计算机硬件及计算机软件等方面，将软件知识与硬件知识相结合，培养学生思维与动手能力。增加学生对物联网的认识，通过具体物联网应用感知物联网技术发展的潮流和方向，学生将进一步加深硬件和软件之间的联系的理解，熟悉解决问题的思想，为实际工作打下良好基础。

# Arduino开发基础

## Arduino简介

Arduino是一款便捷灵活、方便上手的开源电子原型平台。包含硬件(各种型号的Arduino板)和软件(Arduino IDE)。由一个欧洲开发团队于2005年冬季开发。

主要包含两个主要的部分:硬件部分是可以用来做电路连接的Arduino电路板;另外一个则是Arduino IDE，你的计算机中的程序开发环境。你只要在IDE中编写程序代码，将程序上传到Arduino电路板后，程序便会告诉Arduino电路板要做些什么了。

## 安装驱动

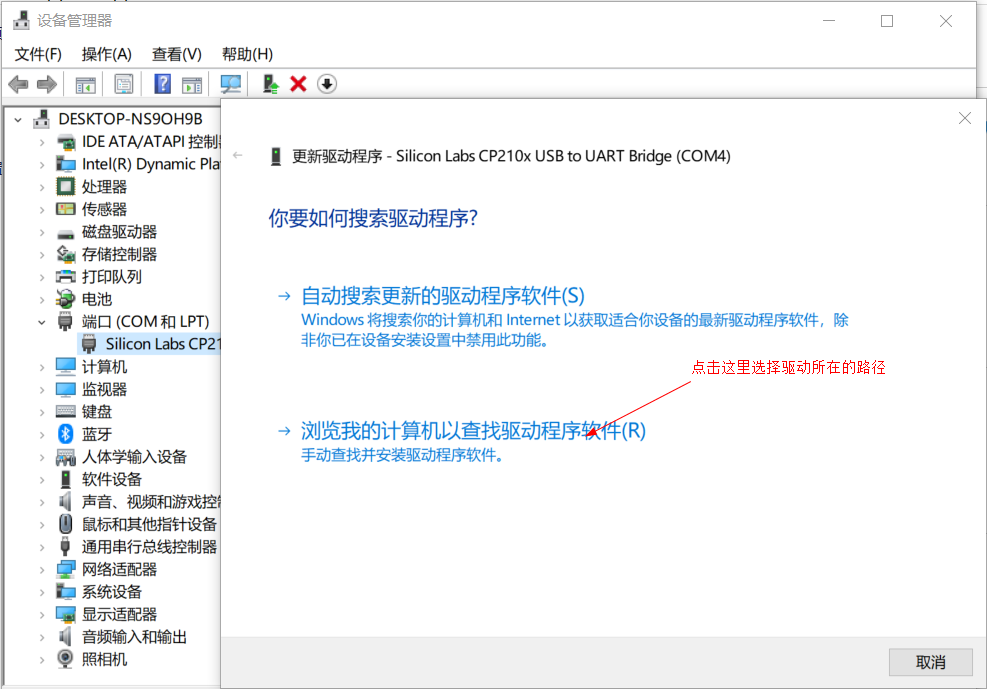
驱动安装：XP系统会提示“新硬件需要安装驱动”，直接关掉，手动安装即可。Win7 以上系统会自动搜索驱动安装，大部分都能正确安装，如果不能，直接手动安装。

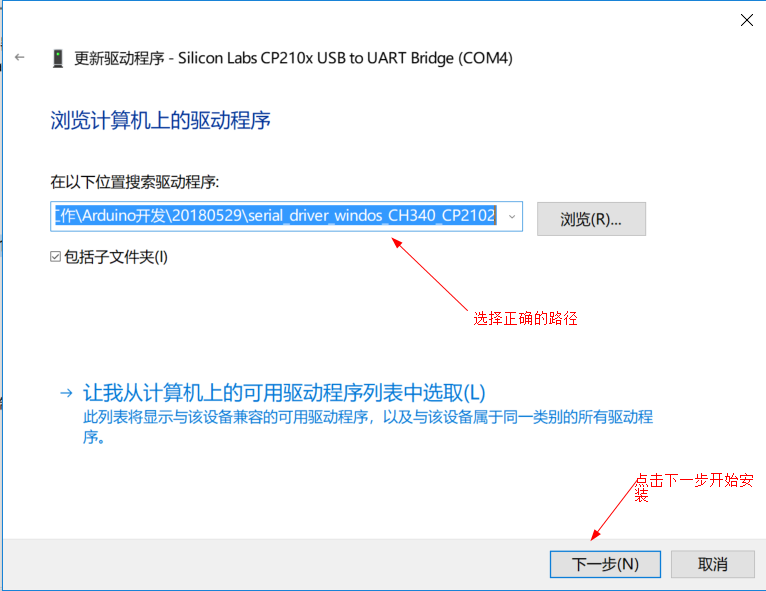
安装驱动方法如下：

Usb 线插入电脑后会提示如上信息，点击“取消”，我们需要手动安装驱动程序手动打开文件夹（驱动程序文件夹内） ，双击安装驱动

XP 系统右击“我的电脑”->“属性”->“硬件”->“设备管理器”

Win10 系统右击“此电脑”->“属性”-> “设备管理器”





安装后：



## NodeMcu介绍

ESP8266是2015年推出的一款Wifi模块，准确来说是集成了Wifi功能的MCU，拉低了目前的市场上Wifi模块价格，甚至一度突破极限，模块的开发语言除了Lua，还有c++，Python（MicroPython）和Java（Smart.js）等。

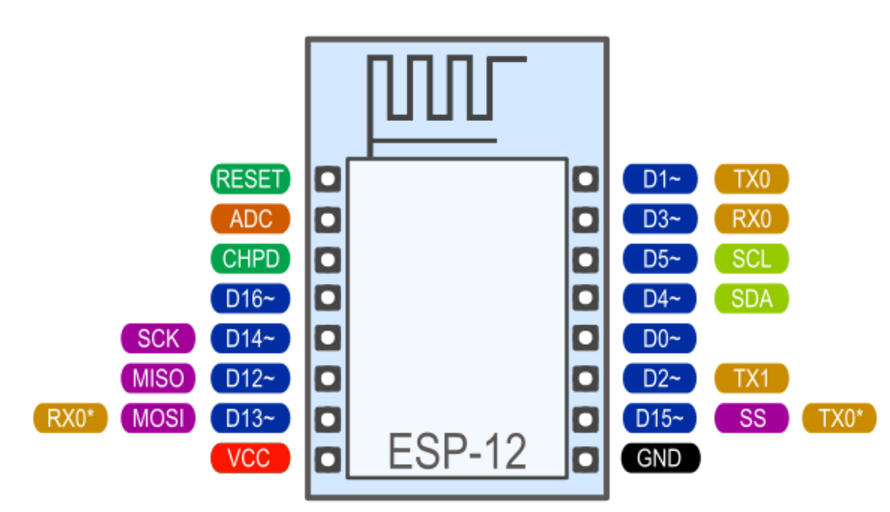
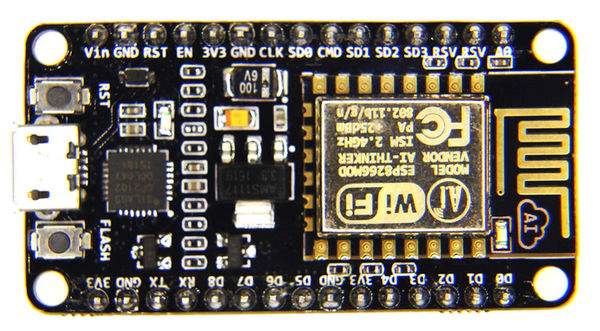


图1：模块引脚与复用功能示意

ESP8266最大峰值+25dbm，实测无阻碍距离300米



**GPIO模式定义语法如下：**

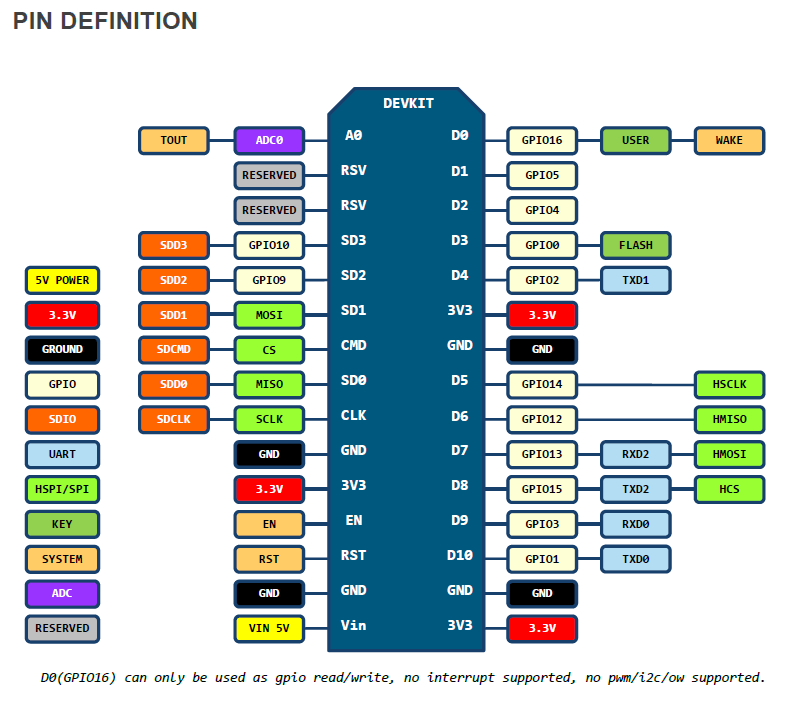
gpio.mode(pin,mode[,pullup])

参数

* pin IO 口序号
* mode 可以是gpio.OUTPUT, gpio.OPENDRAIN, gpio.INPUT, 或者 gpio.INT (中断模式)
* pullup 支持gpio.PULLUP上拉模式；默认为 gpio.FLOAT浮空模式

## 引脚映射

NodeMCU与ESP8266端口对应关系Figure-1



## GPIO序号

GPIO序号与ESP8266引脚对应关系Table-1

| **IO index** | **ESP8266 pin** | **IO index** | **ESP8266 pin** |
| --- | --- | --- | --- |
| 0 [\*] | GPIO16 | 7 | GPIO13 |
| 1 | GPIO5 | 8 | GPIO15 |
| 2 | GPIO4 | 9 | GPIO3 |
| 3 | GPIO0 | 10 | GPIO1 |
| 4 | GPIO2 | 11 | GPIO9 |
| 5 | GPIO14 | 12 | GPIO10 |
| 6 | GPIO12 |  |  |

结合Figure-1与Table-1，我们就能得到NodeMCU硬件引脚对应的IO序号。

## ESP8266开发方式

**AT指令方式** -烧录AT的固件包，使用AT指令与ESP8266交互，执行相应指令，需与单片机相连

**NodeMCU的lua开发** –烧录NodeMCU的固件包，使用Lua语言开发，使用ESP内部资源。

**Arduino IDE下的开发** –相当于直接编写固件，编译之后，烧录进ESP，使用ESP内部资源。

**各自优缺点**

**AT指令开发方式：**

**优点：**开发简单，资料较多。只需知道AT指令集，以及它的通信方式即可。

**缺点：**浪费资源，需要MCU与其通信，不能独立完成某项功能。

**NodeMCU的lua开发：**

NodeMCU本质也是ESP8266，只是它的固件是与lua脚本语言交互。

**优点：**节省资源，开发简单，代码量少。

**缺点：**

1.lua解释器执行效率较低，我最终换成ArduinoIDE开发，就是因为读取传感器数据时，老是漏掉一个数据。2.前期准备比较麻烦。需要准备相应功能的固件，烧录进去，然后使用lua语言和工具与之调试。

**Arduino IDE下的开发方式：**

**优点：**集编程和烧录一体，使用很方便。语言执行效率高，节省资源，开发简单，该有的库也都有。重要的是开发语言，对于我用单片机的人来说，能够很快上手。

**缺点：**Arduino IDE需要写较长的代码时，不是很方便，如果需要查看底层函数或者方法，非常费劲。据说在VisualStudio 中有插件，因为自己电脑的原因，我没有尝试过。

**Arduino开发ESP8266**

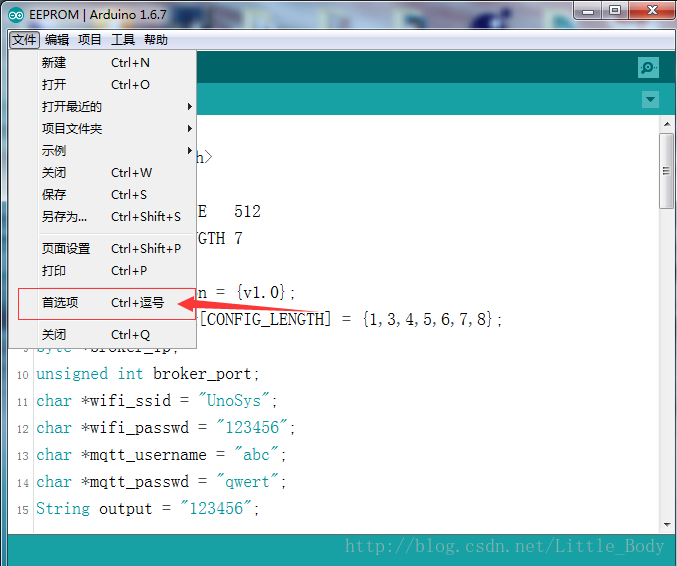
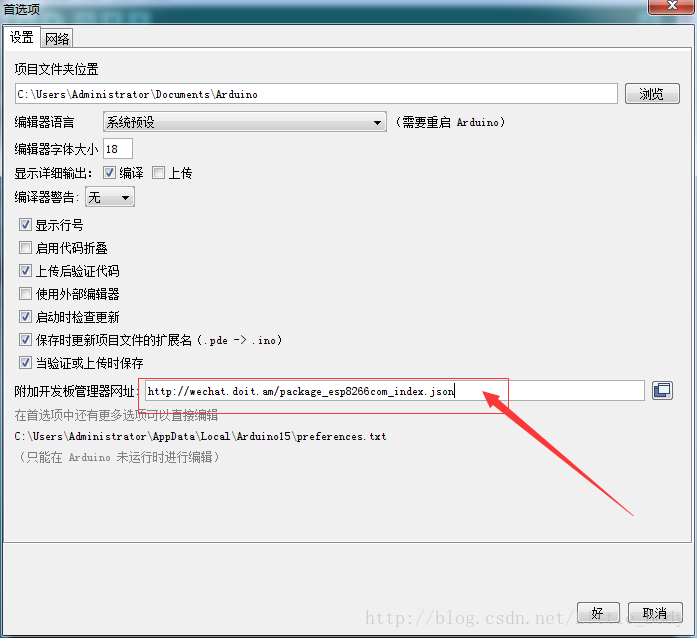
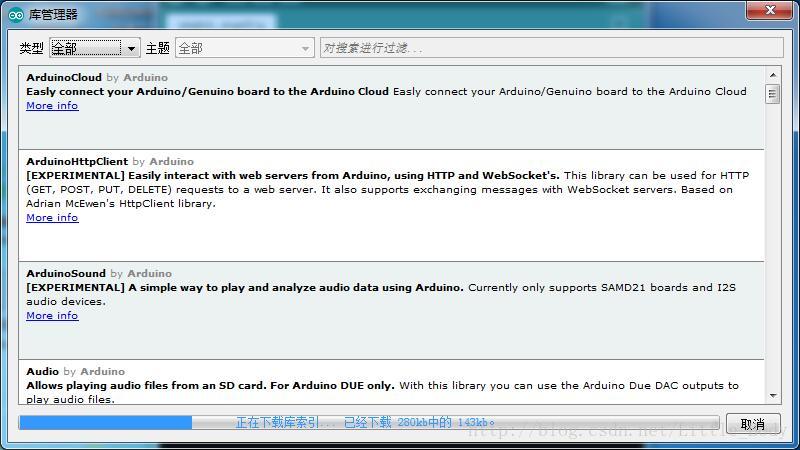
现在介绍一下如何利用Arduino IDE开发8266。

软件环境：Windows 7 64bit

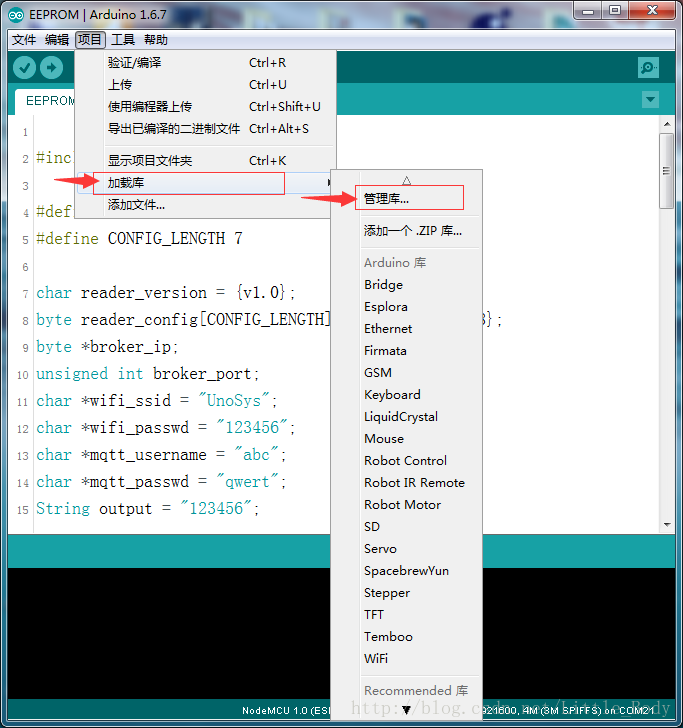
      Arduino 1.6.7 或更高版本

硬件环境：NodeMCU（CH340版）



1. 打开Arduino IDE，文件–>首选项
2. 
3. 附加开发板管理器网址：
4. http://wechat.doit.am/package\_esp8266com\_index.json 或者
5. http://arduino.esp8266.com/stable/package\_esp8266com\_index.json
6. 
7. 项目 –> 加载库 –> 管理库，进入之后，它会自动下载库索引，等待下载完后。（注：如果不能下载，肯定是网络的原因，我使用的是vpn）在搜索栏输入esp8266,点击安装。
8. 



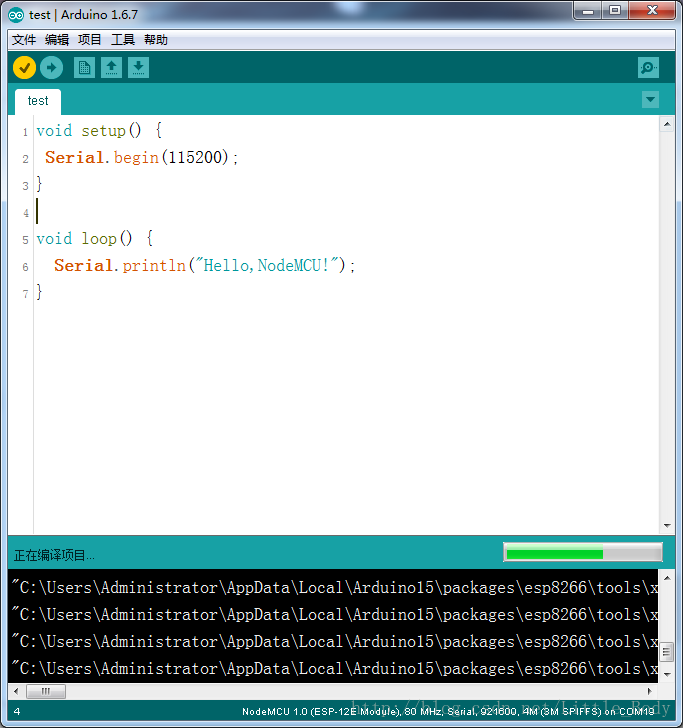
1. 安装完成之后，点击 工具 –> 开发板，会看见esp8266 模块，选择相应的模块。之后，可以看到右下角出现了相应的信息。
2. 
3. 前期准备工作完成，下边开始编写代码，并且烧录。
4. 打开Arduino，点击 文件 –> 新建，点击 文件 –> 保存，输入文件名，确定。
5. 写入代码：
6. void setup() {
7. Serial.begin(115200);
8. }

void loop() {

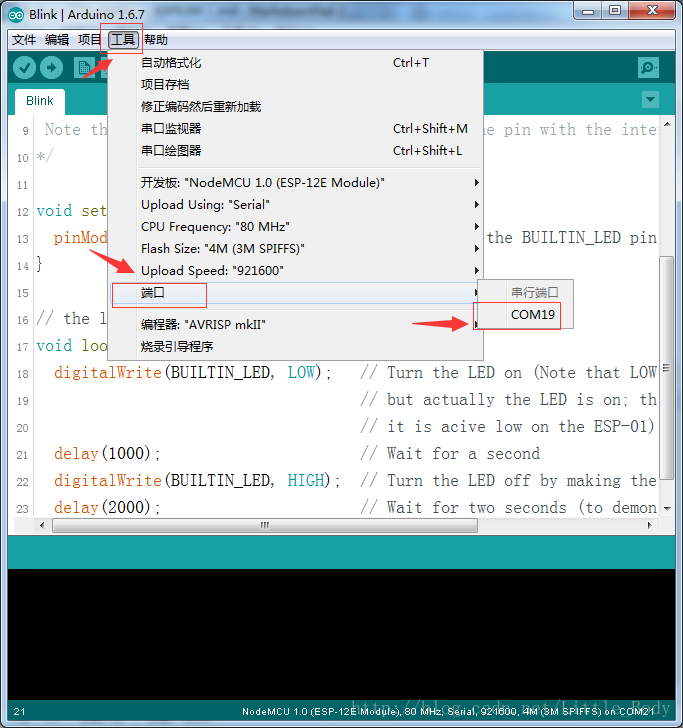
Serial.println("Hello,NodeMCU!");

}

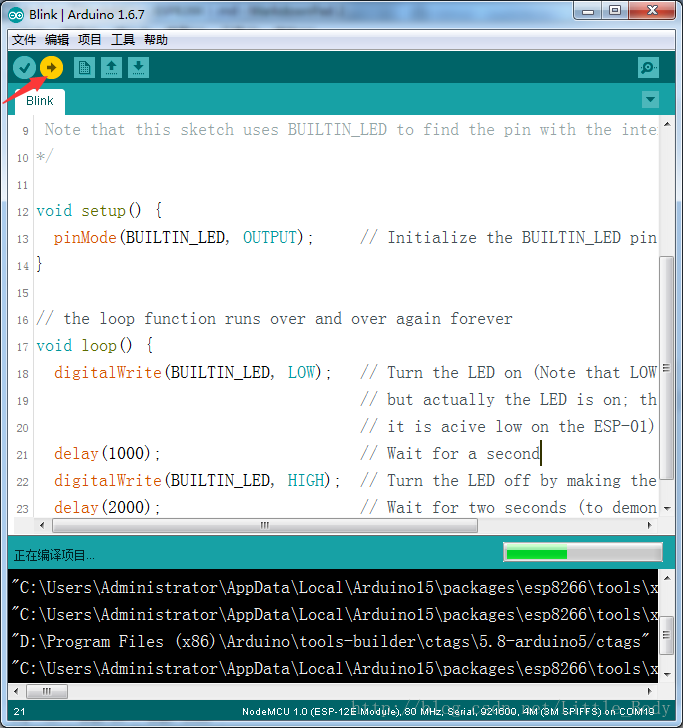
点击 验证，确认没有报错。



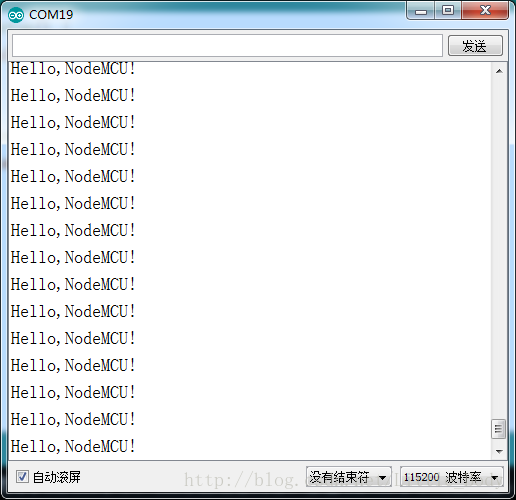
7.连接NodeMCU，安装ch340驱动，点击 工具 –> 端口 –> 端口号。如果驱动安装成功，连接上NodeMCU之后，会显示相应的端口号。若没有显示，请检查驱动或者连接问题。



8.点击 上传，先进行编译，等待上传。上传完成之后，显示上传成功。



9.点击工具 –>串口监视器。可以看见，在不断的输出 Hello，NodeMCU。



到这里，安装完毕，接下来就可以编程了

## Arduino语言说明

**常量：**

[HIGH](http://arduino.cc/en/Reference/Constants) | [LOW](http://arduino.cc/en/Reference/Constants) ：

表示数字IO口的电平，[HIGH](http://arduino.cc/en/Reference/Constants) 表示高电平（1），[LOW](http://arduino.cc/en/Reference/Constants) 表示低电平（0）。

[INPUT](http://arduino.cc/en/Reference/Constants) | [OUTPUT](http://arduino.cc/en/Reference/Constants)：

表示数字IO口的方向，[INPUT](http://arduino.cc/en/Reference/Constants) 表示输入（高阻态），[OUTPUT](http://arduino.cc/en/Reference/Constants) 表示 输出（AVR能提供5V电压 40mA电流）。

**结构：**

void [setup](http://arduino.cc/en/Reference/Setup)() 初始化变量，管脚模式，调用库函数等

void [loop](http://arduino.cc/en/Reference/Loop)() 连续执行函数内的语句

**数字 I/O：**

[pinMode](http://arduino.cc/en/Reference/PinMode)(pin, mode) 数字IO口输入输出模式定义函数，pin表示为0～13， mode表示为INPUT或OUTPUT。

[digitalWrite](http://arduino.cc/en/Reference/DigitalWrite)(pin, value) 数字IO口输出电平定义函数，pin表示为0～13，value表示为HIGH或LOW。比如定义HIGH可以驱动LED。

int [digitalRead](http://arduino.cc/en/Reference/DigitalRead)(pin) 数字IO口读输入电平函数，pin表示为0～13，value表示为HIGH或LOW。比如可以读数字传感器。

**模拟 I/O：**

int [analogRead](http://arduino.cc/en/Reference/AnalogRead)(pin) 模拟IO口读函数，pin表示为0～5（Arduino Diecimila为0～5，Arduino nano为0～7）。比如可以读模拟传感器（10位AD，0～5V表示为0～1023）。

[analogWrite](http://arduino.cc/en/Reference/AnalogWrite)(pin, value) - *PWM* 数字IO口PWM输出函数，Arduino数字IO口标注了PWM的IO口可使用该函数，pin表示3, 5, 6, 9, 10, 11，value表示为0～255。比如可用于电机PWM调速或音乐播放。

**时间函数：**

unsigned long [millis](http://arduino.cc/en/Reference/Millis)() 返回时间函数（单位ms），该函数是指，当程序运行就开始计时并返回记录的参数，该参数溢出大概需要50天时间。

[delay](http://arduino.cc/en/Reference/Delay)(ms) 延时函数（单位ms）。

[delayMicroseconds](http://arduino.cc/en/Reference/DelayMicroseconds)(us) 延时函数（单位us）。

**串口收发函数：**

[Serial.begin](http://arduino.cc/en/Serial/Begin)(speed) 串口定义波特率函数，speed表示波特率，如9600，19200等。

int [Serial.available](http://arduino.cc/en/Serial/Available)() 判断缓冲器状态。

int [Serial.read](http://arduino.cc/en/Serial/Read)() 读串口并返回收到参数。

[Serial.flush](http://arduino.cc/en/Serial/Flush)() 清空缓冲器。

[Serial.print](http://arduino.cc/en/Serial/Print)(data) 串口输出数据。

[Serial.println](http://arduino.cc/en/Serial/Println)(data) 串口输出数据并带回车符。

**数学函数：**

[min](http://arduino.cc/en/Reference/Min)(x, y) 求最小值

[max](http://arduino.cc/en/Reference/Max)(x, y) 求最大值

[abs](http://arduino.cc/en/Reference/Abs)(x) 计算绝对值

[constrain](http://arduino.cc/en/Reference/Constrain)(x, a, b) 约束函数，下限a，上限b，x必须在ab之间才能返回。

[map](http://arduino.cc/en/Reference/Map)(value, fromLow, fromHigh, toLow, toHigh) 约束函数，value必须在fromLow与toLow之间和fromHigh与toHigh之间。

# 实验1

## 点亮蓝灯

/\*

ESP8266 Blink by Simon Peter

Blink the blue LED on the ESP-01 module

This example code is in the public domain

The blue LED on the ESP-01 module is connected to GPIO1

(which is also the TXD pin; so we cannot use Serial.print() at the same time)

Note that this sketch uses BUILTIN\_LED to find the pin with the internal LED

\*/

void setup() {

pinMode(BUILTIN\_LED, OUTPUT); // Initialize the BUILTIN\_LED pin as an output

}

// the loop function runs over and over again forever

void loop() {

digitalWrite(BUILTIN\_LED, LOW); // Turn the LED on (Note that LOW is the voltage level

// but actually the LED is on; this is because

// it is acive low on the ESP-01)

delay(1000); // Wait for a second

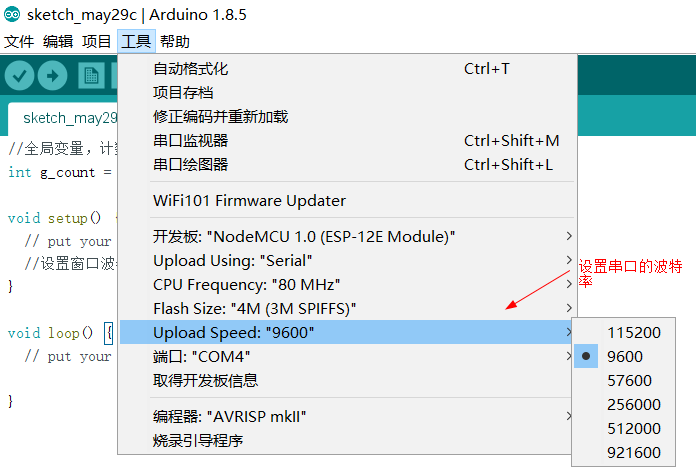
digitalWrite(BUILTIN\_LED, HIGH); // Turn the LED off by making the voltage HIGH

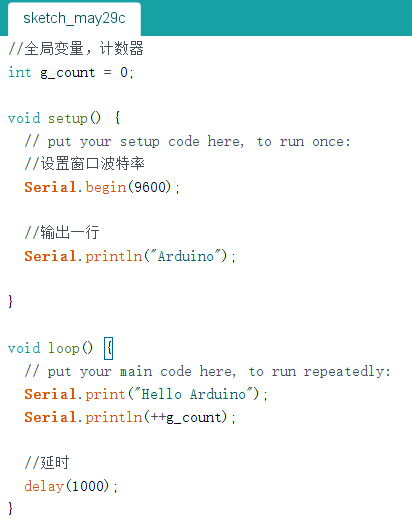
delay(2000); // Wait for two seconds (to demonstrate the active low LED)

}

## 观察串口监视器

**注意这里的波特率最好设置为115200，不然编译速度很慢**





**代码**

//全局变量，计数器

int g\_count = 0;

void setup() {

// put your setup code here, to run once:

//设置窗口波特率

Serial.begin(9600);

//输出一行

Serial.println("Arduino");

}

void loop() {

// put your main code here, to run repeatedly:

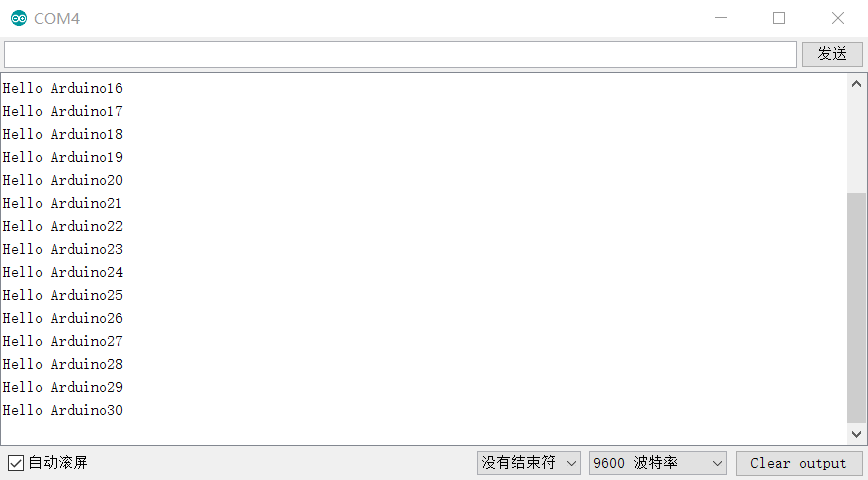
Serial.print("Hello Arduino");

Serial.println(++g\_count);

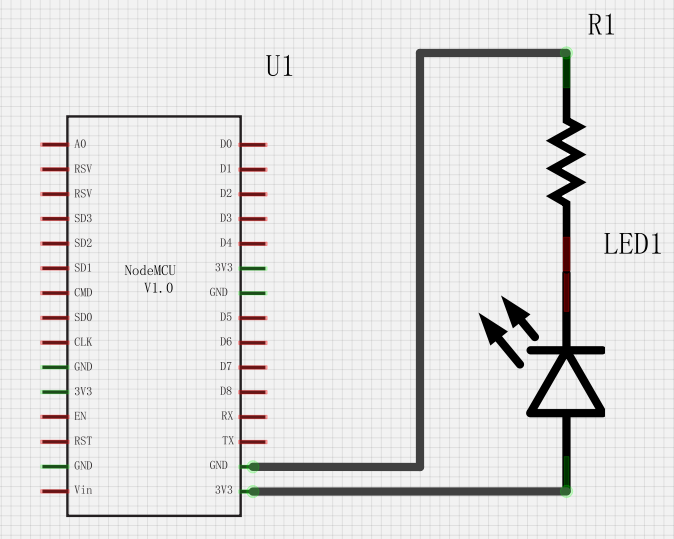
//延时

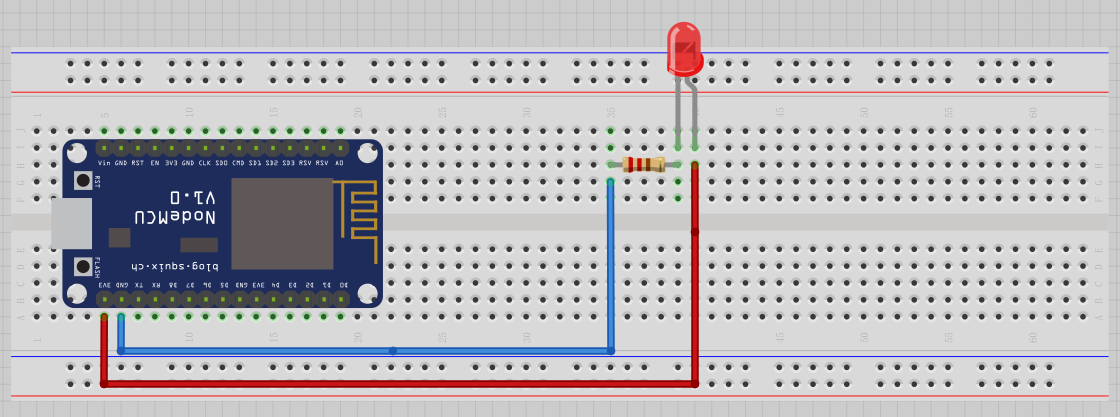
delay(1000);

}



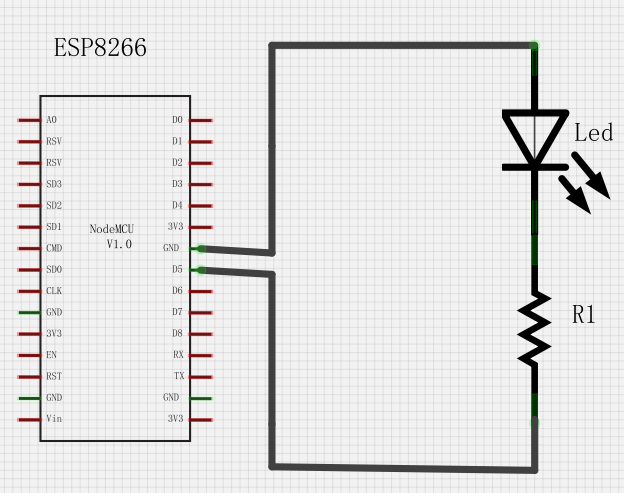
## 点亮Led灯



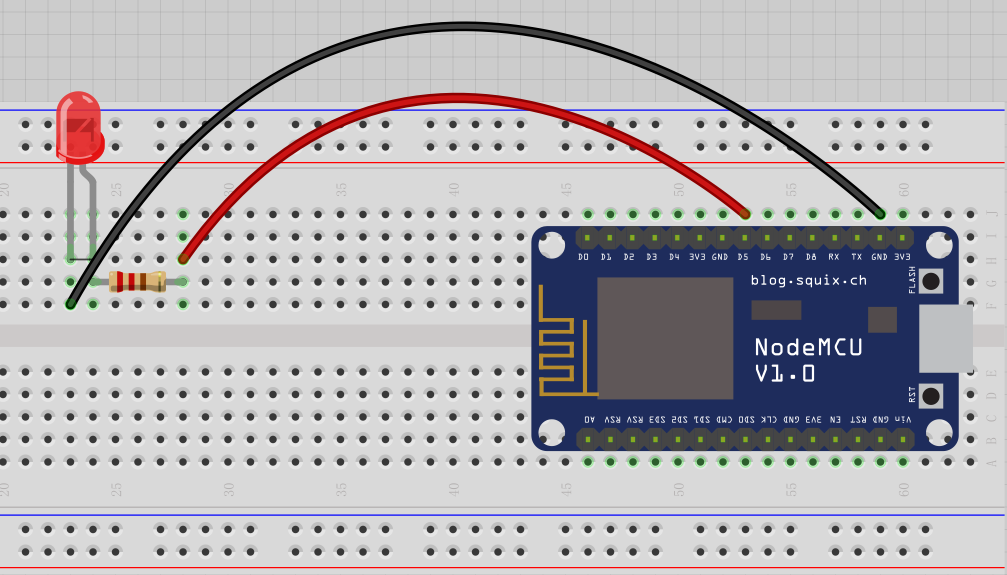


## Led灯亮灭

**原理图**



**实物图**



**接线**

ESP8266上正极为V3.3，负极为GND

3.3V                 <----->    VCC

GND                 <----->   GND

PIN---D5             <----->   OUT



**代码**

//定义引脚

int LED\_BUILTIN = D5;

// the setup function runs once when you press reset or power the board

void setup() {

// initialize digital pin LED\_BUILTIN as an output.

pinMode(LED\_BUILTIN, OUTPUT);

}

// the loop function runs over and over again forever

void loop() {

digitalWrite(LED\_BUILTIN, HIGH); // turn the LED on (HIGH is the voltage level)

delay(1000); // wait for a second

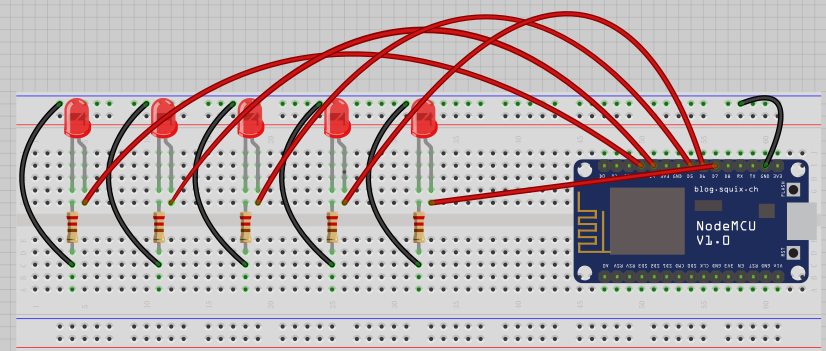
digitalWrite(LED\_BUILTIN, LOW); // turn the LED off by making the voltage LOW

delay(1000); // wait for a second

}

## 跑马灯

**实物图**



//全局变量,记录数字引脚

int LED[] = { D3, D4, D5, D6, D7 };

void setup() {

// put your setup code here, to run once:

for (int i = 0; i <= 4; i++)

{

//设置引脚模式

pinMode(LED[i], OUTPUT);

//给引脚写入高电平

digitalWrite(LED[i], HIGH);

}

}

void loop() {

// put your main code here, to run repeatedly:

for (int i = 0; i <= 4; i++)

{

//给引脚写入低电平

digitalWrite(LED[i], LOW);

//延迟1000毫秒

delay(1000);

//给引脚写入高电平

digitalWrite(LED[i], HIGH);

}

}

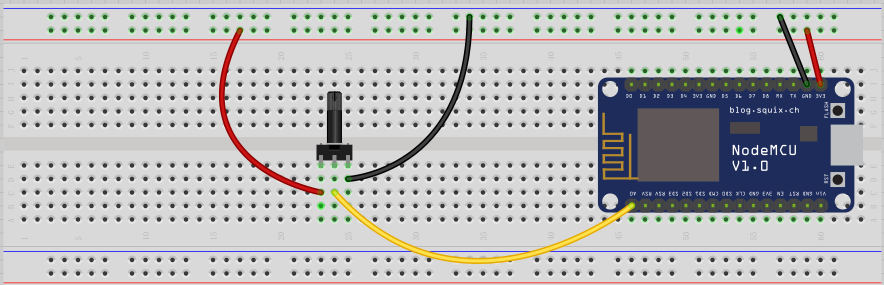
## 观察电位器的值

这个电位器实验，非常简单，

可应用在很多场景，例如音量大小生，一些数值的调控



**实物图**

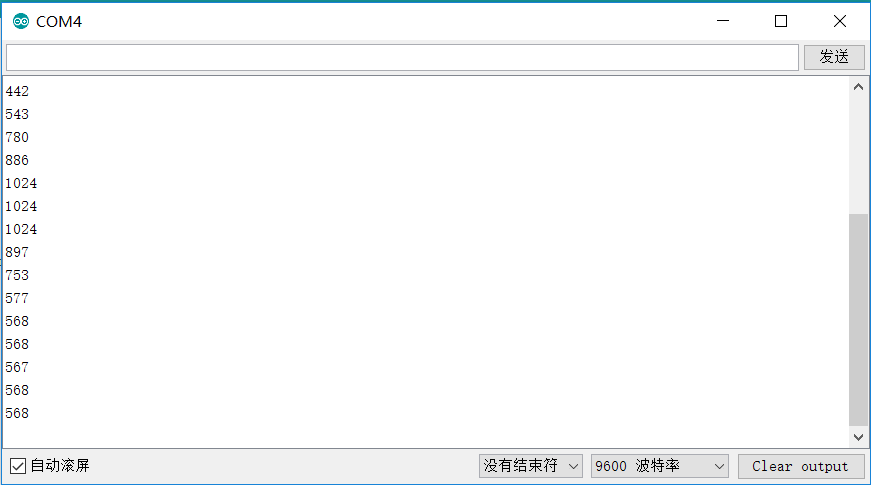


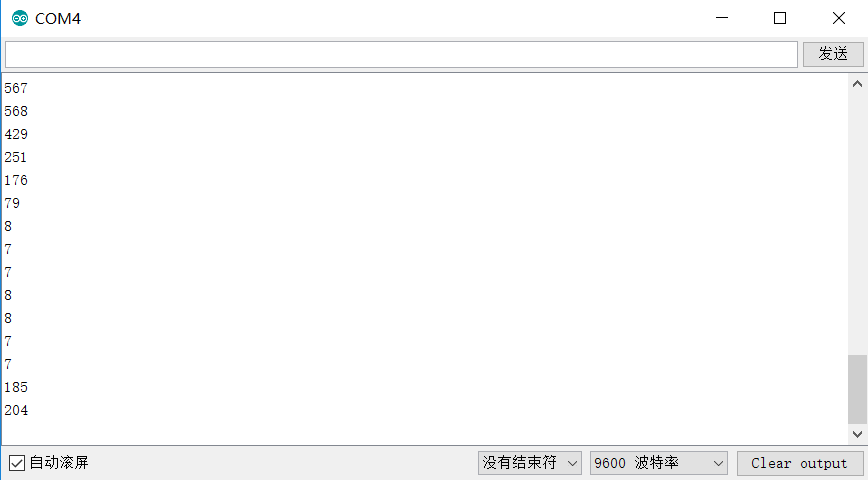
**接线：**

**VCC------------->电源正极**

**OUT-------------> 接到A0**

**GND------------->接地**







**代码**

//观察电位器的值的范围

// the setup function runs once when you press reset or power the board

void setup() {

//设置串口的波特率

Serial.begin(9600);

}

// the loop function runs over and over again forever

void loop() {

//读取模拟口A0的值

int val = analogRead(A0);

//输出值，并换行

Serial.println(val);

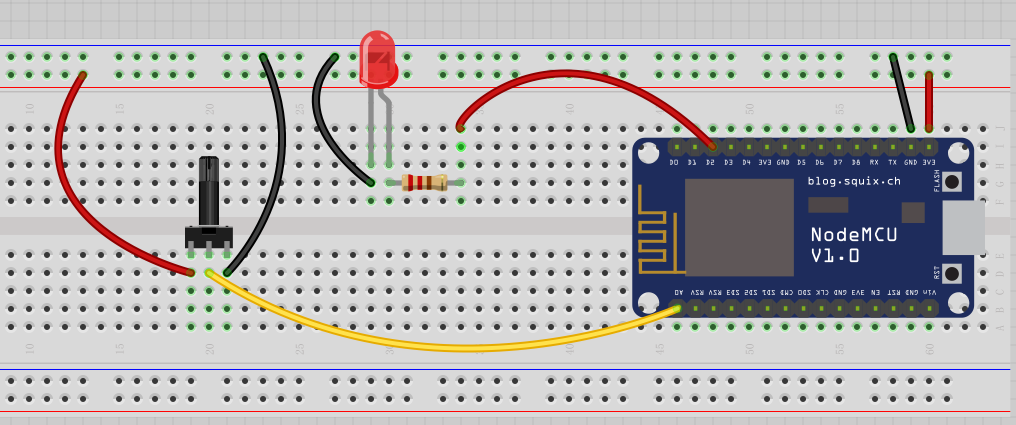
//延时

delay(1000);

}

## 电位器控制Led的亮度

**实物图**



ESP-12E中的D2可以支持模拟的写



**代码**

//定义引脚，连接Led

int led = D2;

void setup()

{

//设置为输出模式

pinMode(led,OUTPUT);

//设置串口的波特率

Serial.begin(9600);

}

void loop()

{

//读取模拟口A0的值

int val = analogRead(A0);

//将读取到的值映射到指定的范围内

val = map(val, 0, 1023, 0, 255);

//输出值

Serial.println(val);

//PWM模拟写

analogWrite(led, val);

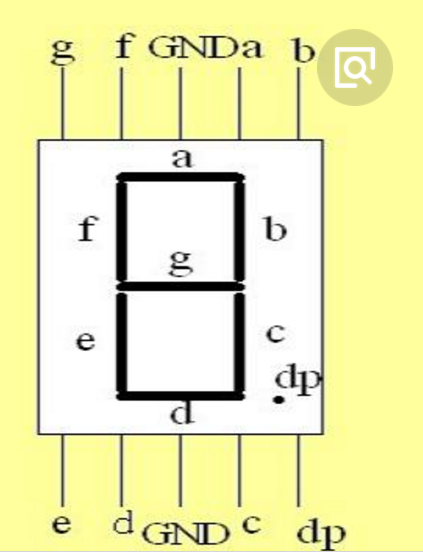
}

# 实验2

## 7段数码管显示

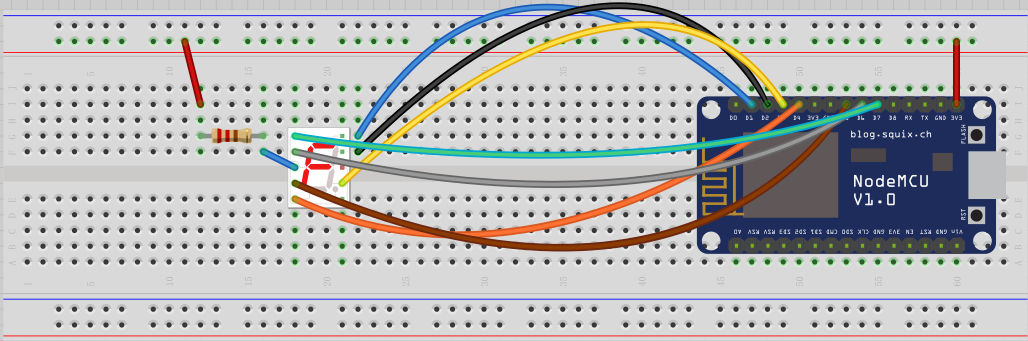
数码管是一种半导体发光器件，其基本单元是发光二极管。数码管按段数分为七段数码管和八段数码管，八段数码管比七段数码管多一个发光二极管单元（多一个小数点显示），本实验所使用的是八段数码管。按发光二极管单元连接方式分为共阳极数码管和共阴极数码管。共阳数码管是指将所有发光二极管的阳极接到一起形成公共阳极的数码管。共阳数码管在应用时应将公共极接到+5V，当某一字段发光二极管的阴极为低电平时，相应字段就点亮。当某一字段的阴极为高电平时，相应字段就不亮。共阴数码管是指将所有发光二极管的阴极接到一起形成公共阴极的数码管。共阴数码管在应用时应将公共极接到地线GND 上，当某一字段发光二极管的阳极为高电平时，相应字段就点亮。当某一字段的阳极为低电平时，相应字段就不亮。

1. **数码管的引脚图：**





**实物图**



**接线:**

a-------->D1

b-------->D2

c-------->D3

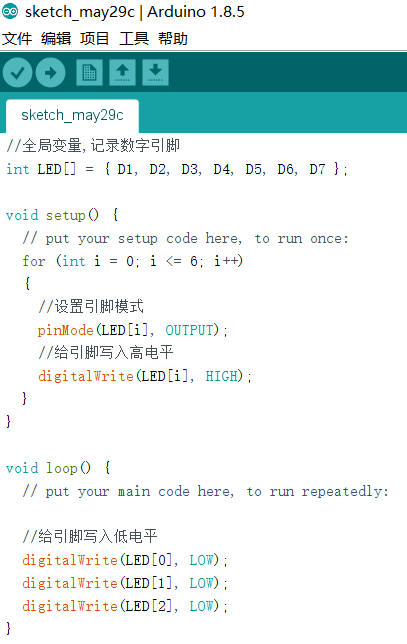
d-------->D4

e-------->D5

f--------->D6

g--------->D7

### 例子1：显示数字



//全局变量,记录数字引脚

int LED[] = { D1, D2, D3, D4, D5, D6, D7 };

void setup() {

// put your setup code here, to run once:

for (int i = 0; i <= 6; i++)

{

//设置引脚模式

pinMode(LED[i], OUTPUT);

//给引脚写入高电平

digitalWrite(LED[i], HIGH);

}

}

void loop() {

// put your main code here, to run repeatedly:

//给引脚写入低电平

digitalWrite(LED[0], LOW);

digitalWrite(LED[1], LOW);

digitalWrite(LED[2], LOW);

}

### 例子2：秒表显示（0-9）

要实现秒表的显示，就是过指定的时间间隔，给数码管中的7个不同的晶体管，设置（高电平或低电平），**我们的接线图是共阳极的**



**因此设置共阳极数码管的数组dofly\_DuanMa中数字的的含义为：**

{0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0xf8,0x80,0x90}

hgfedcba(共阳级，顺序相反，按照上图中abcdefgh的顺序去设置对应的管脚的输出值，设置低电平就导通，对应的数码管会亮起)

---------------------------------------------------------------------------------------------------------------------------------

0xc0 = 1100 0000---->0(hg为高电平)

0xf9 = 1111 1001---->1(bc为低电平)

0xa4 = 1010 0100---->2(abged为低电平)

0xb0 = 1011 0000---->3(abgcd为低电平)

0x99 = 1001 1001---->4(fgbc为低电平)

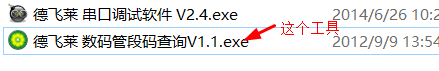
0x92 = 1001 0010---->5(afgcd为低电平)

0x82 = 1000 0010---->6(afedcg为低电平)

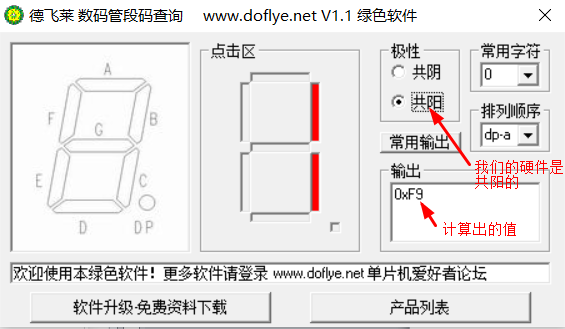
0xf8 = 1111 1000---->7(abc为低电平)

0x80 = 1000 0000---->8(abcdefg为低电平)

0x90 = 1001 0000---->9(afgbcd为低电平)

上面的这些编码的值可以通过，

来自动产生：



//共阳极码段

const unsigned char dofly\_DuanMa[10] = { 0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92, 0x82, 0xf8, 0x80, 0x90 };//const关键字是防止定义的变量的值被以外修改

//全局变量,记录数字引脚

int LEDPins[] = { D1, D2, D3, D4, D5, D6, D7,D8 }; // 对应的 LED 引脚

void setup() {

//循环设置，把对应的 LED 都设置成输出

for (int LED = 0; LED < 8; LED++)

{

pinMode(LEDPins[LED], OUTPUT);

}

}

// 数据处理，把需要处理的 byte 数据写到对应的引脚端口。

void deal(unsigned char value)

{

for (int i = 0; i<8; i++)

{

digitalWrite(LEDPins[i], bitRead(value, i));//使用了 bitRead 函数，非常简单

}

}

//主循环

void loop() {

//循环显示 0-9 数字

for (int i = 0; i<10; i++)

{

deal(dofly\_DuanMa[i]);//读取对应的段码值

delay(1000); //调节延时，2个数字之间的停留间隔

}

}

**然后，编译程序并上传，会看到数字秒表的显示**

### 例子3：红绿灯显示

**注意这里的接线顺序，用排线接到红绿灯模块上，其中红绿灯模块的，**

**G1针脚控制：绿灯,**

**Y1针脚控制：黄灯,**

**R1针脚控制：红灯,**

**COM针脚控制：接地**

**本例中：R1接到8号针脚**

**Y1接到9号针脚**

**G1接到10号针脚**

**黄灯是3秒切换，红灯和绿灯是10秒切换**

然后，在编译器下写入下面的代码：

//针脚的数组

int Pins[] = { D5, D6, D7 };

//标识

int flag = 1;

void setup() {

// put your setup code here, to run once:

for (int i = 0; i < 3; i++)

{

//设置针脚的模式为输出

pinMode(Pins[i], OUTPUT);

}

}

//循环处理

void loop() {

// put your main code here, to run repeatedly:

if (flag == 1)

{

//红灯亮

digitalWrite(Pins[0], HIGH);

//绿灯灭

digitalWrite(Pins[2], LOW);

delay(5000);

//切换标识

flag = 0;

}

else

{

//红等灭

digitalWrite(Pins[0], LOW);

//绿灯亮

digitalWrite(Pins[2], HIGH);

delay(5000);

//切换标识

flag = 1;

}

//红灯灭

digitalWrite(Pins[0], LOW);

//绿灯灭

digitalWrite(Pins[2], LOW);

//黄灯亮

digitalWrite(Pins[1], HIGH);

delay(3000);

//黄灯灭

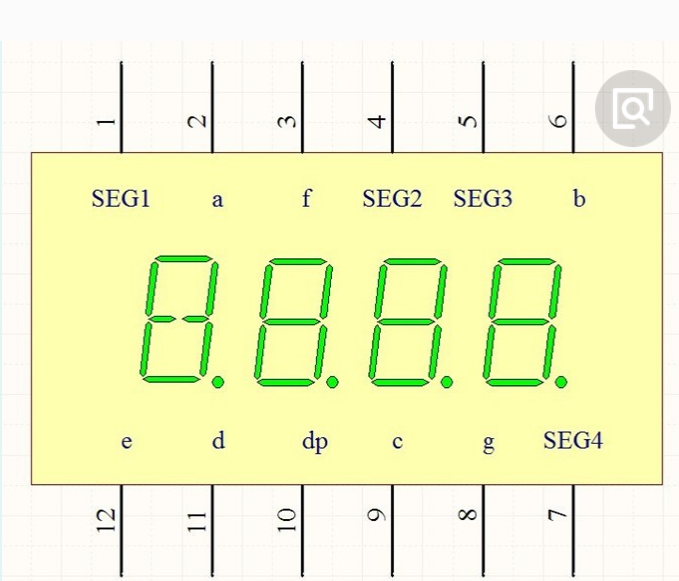
digitalWrite(Pins[1], LOW);

}

## 4位数码管

之前学过 1 位数码管，需要 8 根线与单片机连接，如果是 4 数码管个也用同样的方式连接，则需要 32 个端口，很显然 UNO 主板没有这么多端口，那么最终有什么方法节省端口呢?。就是接通不同com口（给高电平），点亮对应的LED组，实现单个字符控制，再通过短时间的刷新，例如3微秒，如此快闪，人类的肉眼已经看不出是一个一个轮着点亮的。

使用动态的方式解决，首先了解4位数码管的引脚：



a b c d e f g 分别是数码管的引脚，跟之前学的以为数码管一样，dp是4个数码管共同的（阴极或阳极），SEG1，SEG2，SEG3，SEG4分别值的是第一个1位数码管到第四个一位数码管的控制引脚。

//设置阴极接口

a b c d e f g

//设置阳极接口

SEG1，SEG2，SEG3，SEG4

**由于ESP8266NodeMCU开发板，没有像ArduinoUNO开发板那样，直接提供了足够的引脚端口，因此需要参照GPIO引脚映射图，来使用端口。**

### 例子1：16进制计数器

**共阴极**

**本例中, //设置阴极接口**

a 接 D1

b 接 D2

c 接 D3

d 接 D4

e 接 D5

f 接 D6

g 接 D7

dp 接 D8

**//设置阳极接口**

SEG1 接 RX 对应的GPIO是03

SEG2 接 TX 对应的GPIO是01

SEG3 接 SD02 对应的GPIO是09

SEG4 接 SD03 对应的GPIO是10

**a b c d e f g 先挂接电阻，再接数字引脚**

**然后在编译器中写入下面的代码。**

//设置阴极接口

#define d\_a D1

#define d\_b D2

#define d\_c D3

#define d\_d D4

#define d\_e D5

#define d\_f D6

#define d\_g D7

#define d\_h D8

//设置阳极接口

#define COM1 3

#define COM2 1

#define COM3 9

#define COM4 10

//显示数字的数组

unsigned char num[17][8] = {

//a b c d e f g h

{ 1, 1, 1, 1, 1, 1, 0, 0 }, //0

{ 0, 1, 1, 0, 0, 0, 0, 0 }, //1

{ 1, 1, 0, 1, 1, 0, 1, 0 }, //2

{ 1, 1, 1, 1, 0, 0, 1, 0 }, //3

{ 0, 1, 1, 0, 0, 1, 1, 0 }, //4

{ 1, 0, 1, 1, 0, 1, 1, 0 }, //5

{ 1, 0, 1, 1, 1, 1, 1, 0 }, //6

{ 1, 1, 1, 0, 0, 0, 0, 0 }, //7

{ 1, 1, 1, 1, 1, 1, 1, 0 }, //8

{ 1, 1, 1, 1, 0, 1, 1, 0 }, //9

{ 1, 1, 1, 0, 1, 1, 1, 1 }, //A

{ 1, 1, 1, 1, 1, 1, 1, 1 }, //B

{ 1, 0, 0, 1, 1, 1, 0, 1 }, //C

{ 1, 1, 1, 1, 1, 1, 0, 1 }, //D

{ 1, 0, 0, 1, 1, 1, 1, 1 }, //E

{ 1, 0, 0, 0, 1, 1, 1, 1 }, //F

{ 0, 0, 0, 0, 0, 0, 0, 1 }, //.

};

//初始化

void setup()

{

pinMode(d\_a, OUTPUT);

pinMode(d\_b, OUTPUT);

pinMode(d\_c, OUTPUT);

pinMode(d\_d, OUTPUT);

pinMode(d\_e, OUTPUT);

pinMode(d\_f, OUTPUT);

pinMode(d\_g, OUTPUT);

pinMode(d\_h, OUTPUT);

pinMode(COM1, OUTPUT);

pinMode(COM2, OUTPUT);

pinMode(COM3, OUTPUT);

pinMode(COM4, OUTPUT);

}

//循环执行

void loop()

{

for (int l = 0; l < 16; l++)

{

for (int k = 0; k < 16; k++)

{

for (int j = 0; j < 16; j++)

{

for (int i = 0; i < 16; i++)

{

//1000/8=125

for (int q = 0; q<125; q++)

{

Display(1, l);

delay(2);

Display(2, k);

delay(2);

Display(3, j);

delay(2);

Display(4, i);

delay(2);

}

}

}

}

}

}

//显示

void Display(unsigned char com, unsigned char n) {

digitalWrite(d\_a, LOW);

digitalWrite(d\_b, LOW);

digitalWrite(d\_c, LOW);

digitalWrite(d\_d, LOW);

digitalWrite(d\_e, LOW);

digitalWrite(d\_f, LOW);

digitalWrite(d\_g, LOW);

digitalWrite(d\_h, LOW);

switch (com) {

case 1:

digitalWrite(COM1, LOW);

digitalWrite(COM2, HIGH);

digitalWrite(COM3, HIGH);

digitalWrite(COM4, HIGH);

break;

case 2:

digitalWrite(COM1, HIGH);

digitalWrite(COM2, LOW);

digitalWrite(COM3, HIGH);

digitalWrite(COM4, HIGH);

break;

case 3:

digitalWrite(COM1, HIGH);

digitalWrite(COM2, HIGH);

digitalWrite(COM3, LOW);

digitalWrite(COM4, HIGH);

break;

case 4:

digitalWrite(COM1, HIGH);

digitalWrite(COM2, HIGH);

digitalWrite(COM3, HIGH);

digitalWrite(COM4, LOW);

break;

default:break;

}

digitalWrite(d\_a, num[n][0]);

digitalWrite(d\_b, num[n][1]);

digitalWrite(d\_c, num[n][2]);

digitalWrite(d\_d, num[n][3]);

digitalWrite(d\_e, num[n][4]);

digitalWrite(d\_f, num[n][5]);

digitalWrite(d\_g, num[n][6]);

digitalWrite(d\_h, num[n][7]);

}

### 例子2：4位数码管计数器

**共阴极**

#define d\_a D1

#define d\_b D2

#define d\_c D3

#define d\_d D4

#define d\_e D5

#define d\_f D6

#define d\_g D7

#define d\_h D8

#define COM1 3

#define COM2 1

#define COM3 9

#define COM4 10

//数码管0-F码值

unsigned char num[17][8] =

{

//a b c d e f g h

{ 1, 1, 1, 1, 1, 1, 0, 0 }, //0

{ 0, 1, 1, 0, 0, 0, 0, 0 }, //1

{ 1, 1, 0, 1, 1, 0, 1, 0 }, //2

{ 1, 1, 1, 1, 0, 0, 1, 0 }, //3

{ 0, 1, 1, 0, 0, 1, 1, 0 }, //4

{ 1, 0, 1, 1, 0, 1, 1, 0 }, //5

{ 1, 0, 1, 1, 1, 1, 1, 0 }, //6

{ 1, 1, 1, 0, 0, 0, 0, 0 }, //7

{ 1, 1, 1, 1, 1, 1, 1, 0 }, //8

{ 1, 1, 1, 1, 0, 1, 1, 0 }, //9

{ 1, 1, 1, 0, 1, 1, 1, 1 }, //A

{ 1, 1, 1, 1, 1, 1, 1, 1 }, //B

{ 1, 0, 0, 1, 1, 1, 0, 1 }, //C

{ 1, 1, 1, 1, 1, 1, 0, 1 }, //D

{ 1, 0, 0, 1, 1, 1, 1, 1 }, //E

{ 1, 0, 0, 0, 1, 1, 1, 1 }, //F

{ 0, 0, 0, 0, 0, 0, 0, 1 }, //.

};

void setup()

{

pinMode(d\_a, OUTPUT); //设置为输出引脚

pinMode(d\_b, OUTPUT);

pinMode(d\_c, OUTPUT);

pinMode(d\_d, OUTPUT);

pinMode(d\_e, OUTPUT);

pinMode(d\_f, OUTPUT);

pinMode(d\_g, OUTPUT);

pinMode(d\_h, OUTPUT);

pinMode(COM1, OUTPUT);

pinMode(COM2, OUTPUT);

pinMode(COM3, OUTPUT);

pinMode(COM4, OUTPUT);

}

void loop()

{

//累加循环当去到9999自动从0开机计数

for (int l = 0; l < 10; l++)

{

for (int k = 0; k < 10; k++)

{

for (int j = 0; j < 10; j++)

{

for (int i = 0; i < 10; i++)

{

//一秒钟快闪125次，就等于一秒，

//1000/8=125

for (int q = 0; q<125; q++)

{

Display(1, l);//第一位数码管显示l的值

delay(2);

Display(2, k);

delay(2);

Display(3, j);

delay(2);

Display(4, i);

delay(2);

}

}

}

}

}

}

//显示函数，com可选数值范围1-4，num可选数值范围0-9

void Display(unsigned char com, unsigned char n)

{

digitalWrite(d\_a, LOW); //去除余晖

digitalWrite(d\_b, LOW);

digitalWrite(d\_c, LOW);

digitalWrite(d\_d, LOW);

digitalWrite(d\_e, LOW);

digitalWrite(d\_f, LOW);

digitalWrite(d\_g, LOW);

digitalWrite(d\_h, LOW);

//选通位选

switch (com)

{

case 1:

digitalWrite(COM1, LOW); //选择位1

digitalWrite(COM2, HIGH);

digitalWrite(COM3, HIGH);

digitalWrite(COM4, HIGH);

break;

case 2:

digitalWrite(COM1, HIGH);

digitalWrite(COM2, LOW); //选择位2

digitalWrite(COM3, HIGH);

digitalWrite(COM4, HIGH);

break;

case 3:

digitalWrite(COM1, HIGH);

digitalWrite(COM2, HIGH);

digitalWrite(COM3, LOW); //选择位3

digitalWrite(COM4, HIGH);

break;

case 4:

digitalWrite(COM1, HIGH);

digitalWrite(COM2, HIGH);

digitalWrite(COM3, HIGH);

digitalWrite(COM4, LOW); //选择位4

break;

default:break;

}

digitalWrite(d\_a, num[n][0]); //a查询码值表

digitalWrite(d\_b, num[n][1]);

digitalWrite(d\_c, num[n][2]);

digitalWrite(d\_d, num[n][3]);

digitalWrite(d\_e, num[n][4]);

digitalWrite(d\_f, num[n][5]);

digitalWrite(d\_g, num[n][6]);

digitalWrite(d\_h, num[n][7]);

}

### 例子3：秒表

//设置引脚

int a = D1;

int b = D2;

int c = D3;

int d = D4;

int e = D5;

int f = D6;

int g = D7;

int p = D8;

//选择哪个数字位显示的引脚

int d4 = 10;

int d3 = 9;

int d2 = 1;

int d1 = 3;

long n = 0;

int x = 100;

int del = 55;

//初始化执行

void setup()

{

pinMode(a, OUTPUT);

pinMode(b, OUTPUT);

pinMode(c, OUTPUT);

pinMode(d, OUTPUT);

pinMode(e, OUTPUT);

pinMode(f, OUTPUT);

pinMode(g, OUTPUT);

pinMode(p, OUTPUT);

pinMode(d4, OUTPUT);

pinMode(d3, OUTPUT);

pinMode(d2, OUTPUT);

pinMode(d1, OUTPUT);

}

//循环执行

void loop()

{

clearLEDs();

pickDigit(1);

pickNumber((n / x / 1000) % 10);

delayMicroseconds(del);

clearLEDs();

pickDigit(2);

pickNumber((n / x / 100) % 10);

delayMicroseconds(del);

clearLEDs();

pickDigit(3);

dispDec(3);

pickNumber((n / x / 10) % 10);

delayMicroseconds(del);

clearLEDs();

pickDigit(4);

pickNumber(n / x % 10);

delayMicroseconds(del);

n++;

if (digitalRead(3) == LOW)

{

n = 0;

}

}

void pickDigit(int x)

{

digitalWrite(d1, HIGH);

digitalWrite(d2, HIGH);

digitalWrite(d3, HIGH);

digitalWrite(d4, HIGH);

switch (x)

{

case 1:

digitalWrite(d1, LOW);

break;

case 2:

digitalWrite(d2, LOW);

break;

case 3:

digitalWrite(d3, LOW);

break;

default:

digitalWrite(d4, LOW);

break;

}

}

void pickNumber(int x)

{

switch (x)

{

default:

zero();

break;

case 1:

one();

break;

case 2:

two();

break;

case 3:

three();

break;

case 4:

four();

break;

case 5:

five();

break;

case 6:

six();

break;

case 7:

seven();

break;

case 8:

eight();

break;

case 9:

nine();

break;

}

}

void dispDec(int x)

{

digitalWrite(p, HIGH);

}

void clearLEDs()

{

digitalWrite(a, LOW);

digitalWrite(b, LOW);

digitalWrite(c, LOW);

digitalWrite(d, LOW);

digitalWrite(e, LOW);

digitalWrite(f, LOW);

digitalWrite(g, LOW);

digitalWrite(p, LOW);

}

void zero()

{

digitalWrite(a, HIGH);

digitalWrite(b, HIGH);

digitalWrite(c, HIGH);

digitalWrite(d, HIGH);

digitalWrite(e, HIGH);

digitalWrite(f, HIGH);

digitalWrite(g, LOW);

}

void one()

{

digitalWrite(a, LOW);

digitalWrite(b, HIGH);

digitalWrite(c, HIGH);

digitalWrite(d, LOW);

digitalWrite(e, LOW);

digitalWrite(f, LOW);

digitalWrite(g, LOW);

}

void two()

{

digitalWrite(a, HIGH);

digitalWrite(b, HIGH);

digitalWrite(c, LOW);

digitalWrite(d, HIGH);

digitalWrite(e, HIGH);

digitalWrite(f, LOW);

digitalWrite(g, HIGH);

}

void three()

{

digitalWrite(a, HIGH);

digitalWrite(b, HIGH);

digitalWrite(c, HIGH);

digitalWrite(d, HIGH);

digitalWrite(e, LOW);

digitalWrite(f, LOW);

digitalWrite(g, HIGH);

}

void four()

{

digitalWrite(a, LOW);

digitalWrite(b, HIGH);

digitalWrite(c, HIGH);

digitalWrite(d, LOW);

digitalWrite(e, LOW);

digitalWrite(f, HIGH);

digitalWrite(g, HIGH);

}

void five()

{

digitalWrite(a, HIGH);

digitalWrite(b, LOW);

digitalWrite(c, HIGH);

digitalWrite(d, HIGH);

digitalWrite(e, LOW);

digitalWrite(f, HIGH);

digitalWrite(g, HIGH);

}

void six()

{

digitalWrite(a, LOW);

digitalWrite(b, LOW);

digitalWrite(c, HIGH);

digitalWrite(d, HIGH);

digitalWrite(e, HIGH);

digitalWrite(f, HIGH);

digitalWrite(g, HIGH);

}

void seven()

{

digitalWrite(a, HIGH);

digitalWrite(b, HIGH);

digitalWrite(c, HIGH);

digitalWrite(d, LOW);

digitalWrite(e, LOW);

digitalWrite(f, LOW);

digitalWrite(g, LOW);

}

void eight()

{

digitalWrite(a, HIGH);

digitalWrite(b, HIGH);

digitalWrite(c, HIGH);

digitalWrite(d, HIGH);

digitalWrite(e, HIGH);

digitalWrite(f, HIGH);

digitalWrite(g, HIGH);

}

void nine()

{

digitalWrite(a, HIGH);

digitalWrite(b, HIGH);

digitalWrite(c, HIGH);

digitalWrite(d, HIGH);

digitalWrite(e, LOW);

digitalWrite(f, HIGH);

digitalWrite(g, HIGH);

}

## LCD显示模块

LCD1602是一款简易的液晶显示屏，可以**显示**16X2的 字符，包括**符号**，**数字**，**英文**，但不包含中文

**注意在本例中，ESP8266NodeMCU的开发板电压只有3.3V，看到的文字的效果，不太清楚**

本次试验使用arduino直接驱动1602液晶显示文字。1602液晶在应用中非常广泛，最初的1602液晶使用的是HD44780控制器，现在各个厂家的1602模块基本上都是采用了与之兼容的IC，所以特性上基本都是一致的。

**1602LCD主要技术参数**

显示容量为16×2个字符；

芯片工作电压为4.5～5.5V；

工作电流为2.0mA（5.0V）；

模块最佳工作电压为5.0V；

字符尺寸为2.95×4.35（W×H）mm。

**1602液晶接口引脚定义**



LiquidCrystal库的了解：

构造函数，创建一个LiquidCrystal的实例（LiquidCrystal是一个类）。可使用4线或8线方式作为数据线(请注意,还需要指令线).若采用四线方式,将d0-d3悬空不连接.**RW引脚可接地而不用接在Arduino的某个引脚上**;如果这样接,省略在函数中的rw参数.

**语法：**

　　LiquidCrystal lcd(rs, enable, d4, d5, d6, d7)

　　LiquidCrystal lcd(rs, rw, enable, d4, d5, d6, d7)

　　LiquidCrystal lcd(rs, enable, d0, d1, d2, d3, d4, d5, d6, d7)

　　LiquidCrystal lcd(rs, rw, enable, d0, d1, d2, d3, d4, d5, d6, d7)

**参数设置：**

rs: rs连接的Arduino的引脚编号

　　rw: rw连接的Arduino的引脚编号

　　enable:enable连接的Arduino的引脚编号

　　d0, d1, d2, d3, d4, d5, d6, d7: 连接的Arduino的引脚编号

**begin ()指定显示屏的尺寸（宽和高）**

　　　　lcd.begin(cols, rows)

**参数设置：**

　　　　lcd: 液晶类型的名称变量

　　　　cols: 显示器可以显示的列数(1602是16列)

　　　　rows: 显示器可以显示的行数(1602是2行)

**clear () 清除LLCD屏幕上内容，并将光标置于左上角**

　　　　lcd.clear()

　　参数：

　　　　lcd：LiquidCrystal类的对象

**home() 将光标定位在屏幕的左上角。**

　　　　lcd.home()

　　参数设置：

　　　　lcd: LiquidCrystal类的对象

**setCursor() 将光标设置在特定的位置**

　　　　lcd.setCursor(col, row)

　　参数：

　　　　lcd：LiquidCrystal类的对象

　　　　col: 你要显示光标的列 (从0开始计数)

　　　　row: 你要显示光标的行 (从0开始计数)

**write() 向LCD写一个字符**

　　　　lcd.write(data)

　　参数：

　　　　lcd: LiquidCrystal类的对象

　　　　data: 你要显示的字符（仅限英文和数字和你自己定义的字符）。

**print() 将文本内容显示在LCD上**

　　　　lcd.print(data)

　　参数：

　　　　lcd: 液晶类型的名称变量

　　　　data:要显示的数据,可以是char, byte, int, long或者string类型的

**cursor() 显示光标（显示下一个字符将被显示的位置）**

　　　　lcd.cursor()

　　参数设置：

　　　　lcd: 液晶类型的名称变量

**noCursor() 隐藏光标**

　　　　lcd.noCursor()

　　参数：

　　　　lcd: 液晶类型的名称变量

**blink() 显示闪烁的光标**

　　　　lcd.blink()

　　参数设置：

　　　　lcd: 液晶类型的名称变量

**noBlink() 关闭闪烁的光标**

　　　　lcd.noBlink()

　　参数设置：

　　　　lcd: 液晶类型的名称变量

**noDisplay() 关闭液晶显示，但原来显示的内容不会丢失**

　　　　lcd.noDisplay()

　　参数：

　　　　lcd: 液晶类型的名称变量

**display() 恢复使用noDisplay()函数隐藏的内容**

　　　　lcd.display()

　　参数：

　　　　lcd: 液晶类型的名称变量

**scrollDisplayLeft() 使屏幕上的内容向左滚动一个字符**

　　　　lcd.scrollDisplayLeft()

　　参数：

　　　　lcd: 一个LiquidCrystal类的对象

**scrollDisplayRight() 使屏幕上的内容向右滚动一个字符**

　　　　lcd.scrollDisplayRight()

　　参数:

　　　　lcd: 一个LiquidCrystal类的对象

**leftToRight() 将文本从左到右写入屏幕，这意味着后续字符的显示将是从左向右的，但是不会影响先前显示的字符**

　　　　lcd.leftToRight()

　　参数设置：

　　　　lcd: a variable of type LiquidCrystal

**rightToLeft() 将文本从右到左写入屏幕，这意味着后续字符的显示将是从右至左写入，但不影响先前显示的字符**

　　　　lcd.rightToLeft()

　　参数：

　　　　lcd: 一个LiquidCrystal类的对象

**autoscroll() 打开液晶显示屏的自动滚动，将会使得当一个字符输出到LCD时，令先前的文本移动一个位置。**

　　　　lcd.autoscroll()

　　参数：

　　　　lcd: a variable of type LiquidCrystal

**noAutoscroll() 关闭自动滚动功能**

　　　　lcd.noAutoscroll()

　　参数：

　　　　LCD：LiquidCrystal类的对象

**createChar() 创建用户自定义的字符，共课创建8个用户自定义字符，编号从0-7，自定义的字符是由一个5\*7的像素构成，1表示亮，0表示暗**

　　　　lcd.createChar(num, data)

　　参数设置：

　　　　lcd: a variable of type LiquidCrystal

　　　　num: 所创建字符的编号(0-7)

　　　　data: 字符的像素数据

**英文显示**

#include <LiquidCrystal.h>

/\*

\*初始化引脚

\*设置接线的对应引脚编号

\*rs:8

\*enable:7

\*d4:5

\*d5:4

\*d6:3

\*d7:2

\*/

LiquidCrystal lcd(D7, D6, D5, D4, D3, D2);

//初始化

void setup() {

//设置行列值

//指定显示屏的尺寸（宽度和高度）,16列，2行

lcd.begin(16, 2);

//打印字符，将文本显示在LCD上.

lcd.print("hello, DOFLY!");

}

//循环执行

void loop() {

//设置光标到第2行第7列

//(注意：1 表示第2行，从0开始计数):

lcd.setCursor(6, 1);

//打印复位后的运行秒值

lcd.print(millis() / 1000);

lcd.print(" Second");

}

**循环显示**

#include <LiquidCrystal.h>

/\*

\*初始化引脚

\*设置接线的对应引脚编号

\*rs:7

\*enable:6

\*d4:5

\*d5:4

\*d6:3

\*d7:2

\*/

LiquidCrystal lcd(D7, D6, D5, D4, D3, D2);

//显示的内容

char strtitle[] = "hello, DOFLY!";

//内容的长度

int nSize = strlen(strtitle);

//初始化

void setup() {

lcd.begin(16, 2);

}

//循环执行

void loop() {

lcd.setCursor(0, 0);

lcd.print("hello, DOFLY!");

//将光标设置到第1行的最右边

lcd.setCursor(16, 0);

//设置自动滚屏

lcd.autoscroll();

//依次输出单个字符

for (int i = 0; i<nSize; i++)

{

lcd.print(strtitle[i]);

delay(300);

}

//关闭自动滚屏

lcd.noAutoscroll();

}

**左右移动**

#include <LiquidCrystal.h>

/\*

\*初始化引脚

\*设置接线的对应引脚编号

\*rs:7

\*enable:6

\*d4:5

\*d5:4

\*d6:3

\*d7:3

\*/

LiquidCrystal lcd(D7, D6, D5, D4, D3, D2);

//显示的内容

char strtitle[] = "hello, DOFLY!";

//内容的长度

int nSize = strlen(strtitle);

//初始化

void setup() {

lcd.begin(16, 2);

lcd.print("hello, DOFLY!");

}

//循环执行

void loop() {

//lcd.scrollDisplayLeft();

lcd.scrollDisplayRight();

delay(300);

}

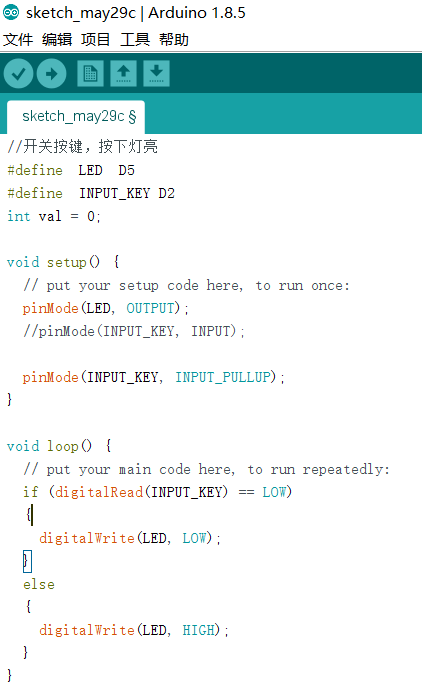
# 实验3

## 按键控制小灯

### 例1：按下亮，弹起灭

Led的正极PIN-----------接线D5

按键开关-----------接线D2



**代码**

#define LED D5

#define INPUT\_KEY D2

int val = 0;

void setup() {

// put your setup code here, to run once:

pinMode(LED, OUTPUT);

//pinMode(INPUT\_KEY, INPUT);

pinMode(INPUT\_KEY, INPUT\_PULLUP);

}

void loop() {

// put your main code here, to run repeatedly:

if (digitalRead(INPUT\_KEY) == LOW)

{

digitalWrite(LED, LOW);

}

else

{

digitalWrite(LED, HIGH);

}

}

### 例2：按一下灭，按一下亮

//按键按一下亮，按一下灭

#define LED D5

#define INPUT\_KEY D2

int val = 0;

void setup() {

// put your setup code here, to run once:

pinMode(LED, OUTPUT);

pinMode(INPUT\_KEY, INPUT\_PULLUP);

digitalWrite(LED, LOW);

}

void loop() {

// put your main code here, to run repeatedly:

ScanKey();

if (val == 1)

{

digitalWrite(LED, !digitalRead(LED));

}

}

//消除抖动,确认按键的处理

void ScanKey()

{

val = 0;

if (digitalRead(INPUT\_KEY) == LOW)

{

delay(20);

if (digitalRead(INPUT\_KEY) == LOW)

{

val = 1;

while (digitalRead(INPUT\_KEY) == LOW)

{

delay(20);

}

}

}

}

## 按键抢答器

**代码**

//LED灯的引脚

int greenled = D4;

int redled = D5;

int yellowled = D6;

//开关的引脚，通过开关控制LED灯

int greenpin = D1;

int redpin = D2;

int yellowpin = D3;

//读取开关的信，来控制LED灯

int red;

int yellow;

int green;

//初始化

void setup()

{

//设置引脚模式为输出

pinMode(redled, OUTPUT);

pinMode(yellowled, OUTPUT);

pinMode(greenled, OUTPUT);

//设置开关的引脚为输入

pinMode(redpin, INPUT);

pinMode(yellowpin, INPUT);

pinMode(greenpin, INPUT);

digitalWrite(greenled, LOW);

digitalWrite(redled, LOW);

digitalWrite(yellowled, LOW);

}

//循环

void loop()

{

//设置为低电平

digitalWrite(greenled, LOW);

digitalWrite(redled, LOW);

digitalWrite(yellowled, LOW);

//读取控制红灯的开关的信号

red = digitalRead(redpin);

//根据控制开关的信，给LED引脚写入电平

if (red == LOW)

{

//给LED引脚写入低电平

digitalWrite(redled, LOW);

}

else

{

//给LED引脚写入高电平

digitalWrite(redled, HIGH);

}

//读取控制黄灯的开关的信号

yellow = digitalRead(yellowpin);

//根据控制开关的信，给LED引脚写入电平

if (yellow == LOW)

{

//给LED引脚写入低电平

digitalWrite(yellowled, LOW);

}

else

{

//给LED引脚写入高电平

digitalWrite(yellowled, HIGH);

}

//读取控制绿灯的开关的信号

green = digitalRead(greenpin);

//根据控制开关的信，给LED引脚写入电平

if (green == LOW)

{

//给LED引脚写入低电平

digitalWrite(greenled, LOW);

}

else

{

//给LED引脚写入高电平

digitalWrite(greenled, HIGH);

}

}

## 发声模块

蜂鸣器和喇叭发声实验

用Arduino 可以完成的互动作品有很多，最常见也最常用的就是声光展示了，前面一直都是在用LED、LCD，本个实验就让大家的电路发出声音，能够发出声音的最常见的元器件就是蜂鸣器和喇叭了。

两者的区别：

蜂鸣器：蜂鸣器直接接固定直流电压，通电即发出固定频率的声音，蜂鸣器只能用固定电压驱动，发生频率出厂固定的。音响、MP3、耳机、手机上都是用的喇叭，声音频率是可以改变的

喇叭：喇叭通过驱动器可以发出各种声音。

无源蜂鸣器：一种一体化结构的电子讯响器，分为有源蜂鸣器与无源蜂鸣器。这里的“源”不是指电源，而是指震荡源，有源蜂鸣器内部带震荡源，所以只要一通电就会响，而无源内部不带震荡源，所以如果仅用直流信号无法令其鸣叫，必须用2K-5K的方波去驱动它。从外观上看，两种蜂鸣器区别不大，但将两种蜂鸣器的引脚都朝上放置时，**可以看出有绿色电路板的一种是无源蜂鸣器，没有电路板而用黑胶封闭的一种是有源蜂鸣器**。如图：



### 发声

int tonepin=D3;//设置控制蜂鸣器的数字D3脚

void setup()

{

pinMode(tonepin,OUTPUT);//设置数字IO脚模式，OUTPUT为输出

}

void loop()

{

unsigned char i,j;

while(1)

{

for(i=0;i<80;i++)//输出一个频率的声音

{

digitalWrite(tonepin,HIGH);//发声音

delay(1);//延时1ms

digitalWrite(tonepin,LOW);//不发声音

delay(1);//延时ms

}

for(i=0;i<100;i++)//输出另一个频率的声音,这里的100与前面的80一样，用来控制频率，可以自己调节

{

digitalWrite(tonepin,HIGH);

delay(2);

digitalWrite(tonepin,LOW);

delay(2);

}

}

}

### 播放乐谱

从上面的实验看，如果我们能够控制好频率和节拍，那就有可能演奏出动听的音乐。因此，我们首先需要搞清楚各音调的频率，具体见下表：

低音：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 音调  音符 | 1# | 2# | 3# | 4# | 5# | 6# | 7# |
| A | 221 | 248 | 278 | 294 | 330 | 371 | 416 |
| B | 248 | 278 | 294 | 330 | 371 | 416 | 467 |
| C | 131 | 147 | 165 | 175 | 196 | 221 | 248 |
| D | 147 | 165 | 175 | 196 | 221 | 248 | 278 |
| E | 165 | 175 | 196 | 221 | 248 | 278 | 312 |
| F | 175 | 196 | 221 | 234 | 262 | 294 | 330 |
| G | 196 | 221 | 234 | 262 | 294 | 330 | 371 |

中音：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 音调  音符 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| A | 441 | 495 | 556 | 589 | 661 | 742 | 833 |
| B | 495 | 556 | 624 | 661 | 742 | 833 | 935 |
| C | 262 | 294 | 330 | 350 | 393 | 441 | 495 |
| D | 294 | 330 | 350 | 393 | 441 | 495 | 556 |
| E | 330 | 350 | 393 | 441 | 495 | 556 | 624 |
| F | 350 | 393 | 441 | 495 | 556 | 624 | 661 |
| G | 393 | 441 | 495 | 556 | 624 | 661 | 742 |

高音：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 音调  音符 | 1# | 2# | 3# | 4# | 5# | 6# | 7# |
| A | 882 | 990 | 1112 | 1178 | 1322 | 1484 | 1665 |
| B | 990 | 1112 | 1178 | 1322 | 1484 | 1665 | 1869 |
| C | 525 | 589 | 661 | 700 | 786 | 882 | 990 |
| D | 589 | 661 | 700 | 786 | 882 | 990 | 1112 |
| E | 661 | 700 | 786 | 882 | 990 | 1112 | 1248 |
| F | 700 | 786 | 882 | 935 | 1049 | 1178 | 1322 |
| G | 786 | 882 | 990 | 1049 | 1178 | 1322 | 1484 |

我们知道了音调的频率后，下一步就是控制音符的演奏时间。每个音符都会播放一定的时间，这样才能构成一首优美的曲子，而不是生硬的一个调的把所有的音符一股脑的都播放出来。音符节奏分为一拍、半拍、1/4拍、1/8拍，我们规定一拍音符的时间为1；半拍为0.5；1/4拍为0.25；1/8拍为0.125……，所以我们可以为每个音符赋予这样的拍子播放出来，音乐就成了。

这里我们具体以《欢乐颂》为例：



        从简谱看，该音乐是D调的，这里的各音符对应的频率对应的是上表中D调的部分。另外，该音乐为四分之四拍，每个对应为1拍。几个特殊音符说明如下：

第一，普通音符。如第一个音符3，对应频率350，占1拍。

第二，带下划线音符，表示0.5拍。

第三，有的音符后带一个点，表示多加0.5拍，即1+0.5

第四，有的音符后带一个—，表示多加1拍，即1+1

第五，有的两个连续的音符上面带弧线，表示连音，可以稍微改下连音后面那个音的频率，比如减少或增加一些数值（需自己调试），这样表现会更流畅，其实不做处理，影响也不大。

下面，看具体代码：

#define NTD0 -1

#define NTD1 294

#define NTD2 330

#define NTD3 350

#define NTD4 393

#define NTD5 441

#define NTD6 495

#define NTD7 556

#define NTDL1 147

#define NTDL2 165

#define NTDL3 175

#define NTDL4 196

#define NTDL5 221

#define NTDL6 248

#define NTDL7 278

#define NTDH1 589

#define NTDH2 661

#define NTDH3 700

#define NTDH4 786

#define NTDH5 882

#define NTDH6 990

#define NTDH7 112

//列出全部D调的频率

#define WHOLE 1

#define HALF 0.5

#define QUARTER 0.25

#define EIGHTH 0.25

#define SIXTEENTH 0.625

//列出所有节拍

int tune[]= //根据简谱列出各频率

{

NTD3,NTD3,NTD4,NTD5,

NTD5,NTD4,NTD3,NTD2,

NTD1,NTD1,NTD2,NTD3,

NTD3,NTD2,NTD2,

NTD3,NTD3,NTD4,NTD5,

NTD5,NTD4,NTD3,NTD2,

NTD1,NTD1,NTD2,NTD3,

NTD2,NTD1,NTD1,

NTD2,NTD2,NTD3,NTD1,

NTD2,NTD3,NTD4,NTD3,NTD1,

NTD2,NTD3,NTD4,NTD3,NTD2,

NTD1,NTD2,NTDL5,NTD0,

NTD3,NTD3,NTD4,NTD5,

NTD5,NTD4,NTD3,NTD4,NTD2,

NTD1,NTD1,NTD2,NTD3,

NTD2,NTD1,NTD1

};

float durt[]= //根据简谱列出各节拍

{

1,1,1,1,

1,1,1,1,

1,1,1,1,

1+0.5,0.5,1+1,

1,1,1,1,

1,1,1,1,

1,1,1,1,

1+0.5,0.5,1+1,

1,1,1,1,

1,0.5,0.5,1,1,

1,0.5,0.5,1,1,

1,1,1,1,

1,1,1,1,

1,1,1,0.5,0.5,

1,1,1,1,

1+0.5,0.5,1+1,

};

int length;

int tonepin=D3; //得用D3号接口

void setup()

{

pinMode(tonepin,OUTPUT);

length=sizeof(tune)/sizeof(tune[0]); //计算长度

}

void loop()

{

for(int x=0;x<length;x++)

{

tone(tonepin,tune[x]);

delay(500\*durt[x]); //这里用来根据节拍调节延时，500这个指数可以自己调整，在该音乐中，我发现用500比较合适。

noTone(tonepin);

}

delay(2000);

}

### LED灯伴随音乐闪烁

**接线：有源蜂鸣器，正极接到Esp8266的vin（5V）口上，负极接到D3口**

**led等正极接到D5口**

#define NTD0 -1

#define NTD1 294

#define NTD2 330

#define NTD3 350

#define NTD4 393

#define NTD5 441

#define NTD6 495

#define NTD7 556

#define NTDL1 147

#define NTDL2 165

#define NTDL3 175

#define NTDL4 196

#define NTDL5 221

#define NTDL6 248

#define NTDL7 278

#define NTDH1 589

#define NTDH2 661

#define NTDH3 700

#define NTDH4 786

#define NTDH5 882

#define NTDH6 990

#define NTDH7 112

//c pinlv

#define WHOLE 1

#define HALF 0.5

#define QUARTER 0.25

#define EIGHTH 0.25

#define SIXTEENTH 0.625

int tune[]=

{

NTD3,NTD3,NTD4,NTD5,

NTD5,NTD4,NTD3,NTD2,

NTD1,NTD1,NTD2,NTD3,

NTD3,NTD2,NTD2,

NTD3,NTD3,NTD4,NTD5,

NTD5,NTD4,NTD3,NTD2,

NTD1,NTD1,NTD2,NTD3,

NTD2,NTD1,NTD1,

NTD2,NTD2,NTD3,NTD1,

NTD2,NTD3,NTD4,NTD3,NTD1,

NTD2,NTD3,NTD4,NTD3,NTD2,

NTD1,NTD2,NTDL5,NTD0,

NTD3,NTD3,NTD4,NTD5,

NTD5,NTD4,NTD3,NTD4,NTD2,

NTD1,NTD1,NTD2,NTD3,

NTD2,NTD1,NTD1

};

float durt[]=

{

1,1,1,1,

1,1,1,1,

1,1,1,1,

1+0.5,0.5,1+1,

1,1,1,1,

1,1,1,1,

1,1,1,1,

1+0.5,0.5,1+1,

1,1,1,1,

1,0.5,0.5,1,1,

1,0.5,0.5,1,1,

1,1,1,1,

1,1,1,1,

1,1,1,0.5,0.5,

1,1,1,1,

1+0.5,0.5,1+1,

};

int length;

int tonepin = D3;

int ledp = D5;

void setup()

{

pinMode(tonepin,OUTPUT);

pinMode(ledp,OUTPUT);

length=sizeof(tune)/sizeof(tune[0]);

}

void loop()

{

for(int x=0;x<length;x++)

{

tone(tonepin,tune[x]);

digitalWrite(ledp, HIGH);

delay(400\*durt[x]);//与前一代码不同之处，这里将原来的500分为了400和100，分别控制led的开与关，对于蜂鸣器来说依然是500.

digitalWrite(ledp, LOW);

delay(100\*durt[x]);

noTone(tonepin);

}

delay(2000);

}

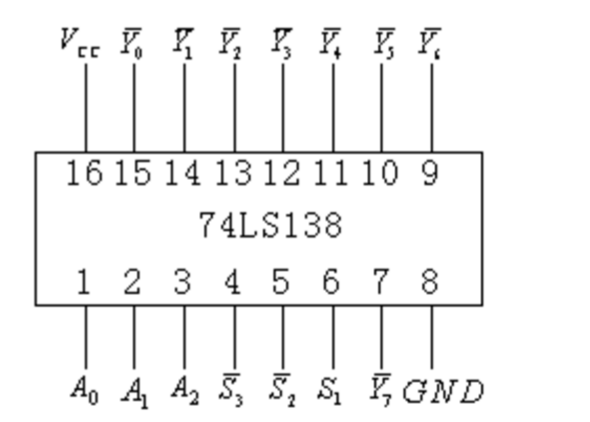
# 实验4

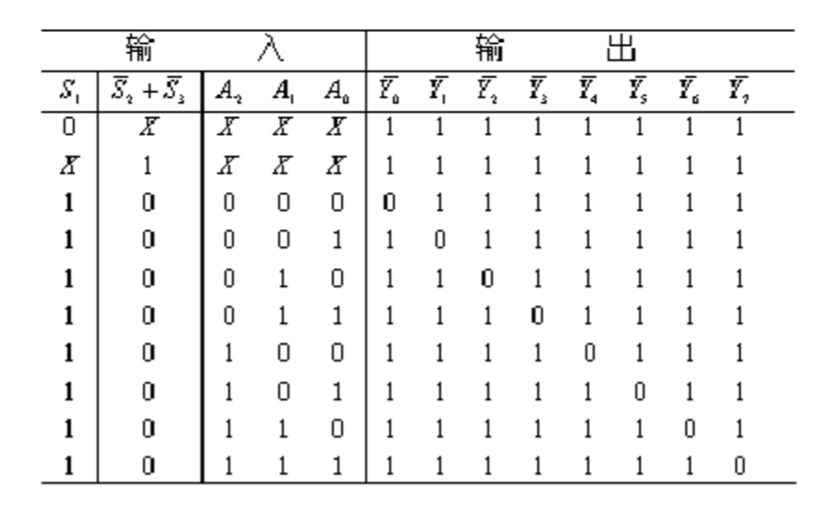
## 74HC138 3-8译码器

译码器也称解码器，译码过程实际上是一种翻译过程,即编码的逆过程。**译码器的输入是n位二值代码**，**输出是m个表征代码原意的状态信号**（或另一种代码）。一般情况下有m小于等于2的n次方，即译码器输入线比输出线要少。

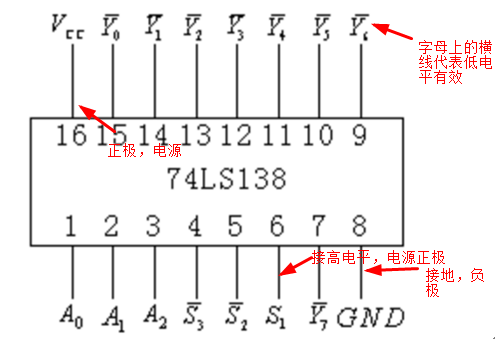
从这个名字来分析，三八译码器，就是把 3 种输入状态翻译成 8 种输出状态。

**3-8译码器的引脚说明：**



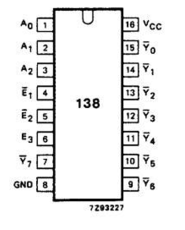
1-表示高电平，0表示低电平，x表示任意电平

从功能表我们都可以看到74HC138的八个输出引脚，任何时刻要么全为高电平1—芯片处于不工作状态，要么只有一个为低电平0(有效)，其余7个输出引脚全为高电平1（无效）。如果出现两个输出引脚同时为0的情况，说明该芯片已经损坏。



实物品图

注意：注：E1,E2必须接地，E3接正极



**(0到7对应的2进制编码)**

0------->00000000

1------->00000001

2------->00000010

3------->00000011

4------->00000100

5------->00000101

6------->00000110

7------->00000111

**然后在编译器中，写入下面的代码。**

//全局变量

int segCount = 3;

int segPins[] = { D2, D3, D4 }; // A0,A1,A2

//初始化

void setup()

{

//循环设置，把对应的端口都设置成输出

for (int thisSeg = 0; thisSeg < segCount; thisSeg++)

{

//设置引脚为输出模式

pinMode(segPins[thisSeg], OUTPUT);

}

}

//数据处理，把需要处理的 byte 数据写到对应的引脚端口。

void deal(unsigned char value)

{

for (int i = 0; i<3; i++)

{

digitalWrite(segPins[i], bitRead(value, i));//使用了 bitRead函数，非常简单

}

}

//循环执行

void loop()

{

//循环输出0-7,输出端是 Y0-Y7 依次为0,任何时刻只有1个脚为低电平。

//只要A0,A1,A2输入，38译码器会转换成对应的数字,然后译码器会让

//Y0-Y7中对应下标的引脚输出低电平,从而电路导通,灯亮起

for (int i = 0; i<8; i++)

{

deal(i);//数据处理

delay(1000);//延时1秒

}

}

## RGB 三基色 LED

三基色是指 RGB 三种颜色，R=Red，G=Green，B=Blue，通过这三种颜色不同比例的混合，可以显示任何颜色。全彩的 LED 屏幕就是通过这种单个的 RGB 灯珠集成在一起做成的，一个 LED 就是一个像素，这个像素可以通过程序控制显示任何颜色、任何灰度。使用 1 个单色 LED，主要功能是亮度变化，如果三种颜色都用 PWM 控制器亮度，就可以混合出不同的比例。看到的彩色图片一般也会用 RGB 格式表示，比如 24 位彩色，RGB 三种颜色各占 8 位。

还是从最基本的说起，既然三基色灯是 3 种颜色灯的组合，那么控制部分也就变得非常明确，1 种颜色用 1 路 PWM，三种颜色就用三路 PWM，很幸运，UNO 有大于 3 路 PWM功能输出，所以还有剩余 PWM 来完成其他功能。下面是电路图和实物连接他。同样的 RGB也分共阳和共阴，LED 一共 4 个有效引脚，1 个公共端，另外 3 个是三种颜色的另外一端。

**注意我们的三基色管是共阴极的（接地），针脚的顺序依次是（红，接地，绿，蓝）**

**本例中红接D7，绿接D6，蓝接D5**

然后在编译器中写入下面的代码：

### 例子1

//全局变量,定义使用的针脚

int led1 = D7;

int led2 = D6;

int led3 = D5;

//初始化

void setup() {

//put your setup code here, to run once:

//设置引脚为输出模式

pinMode(led1, OUTPUT);

pinMode(led2, OUTPUT);

pinMode(led3, OUTPUT);

}

//设置颜色

void setColor(int red, int green, int blue)

{

analogWrite(led1, red);

analogWrite(led2, green);

analogWrite(led3, blue);

}

//循环执行

void loop() {

//put your main code here, to run repeatedly:

setColor(255, 0, 0);

delay(1000);

setColor(0, 255, 0);

delay(1000);

setColor(0, 0, 255);

delay(1000);

setColor(0, 255, 255);

delay(1000);

setColor(255, 0, 255);

delay(1000);

}

### 例子2

#define LED\_R D7

#define LED\_G D6

#define LED\_B D5

enum{ Color\_R, Color\_G, Color\_B, Color\_RG, Color\_RB, Color\_GB, Color\_RGB };

void setup(){

pinMode(LED\_R, OUTPUT);

pinMode(LED\_G, OUTPUT);

pinMode(LED\_B, OUTPUT);

}

void loop(){

unsigned int i;

for (i = 0; i<7; i++)

{

Change\_Color(i);

delay(1000);

}//改变七种色彩，调用change\_Color 函数

}

void Change\_Color(unsigned char data\_color)

{

switch (data\_color)

{

case Color\_R: //红色

digitalWrite(LED\_R, LOW);

digitalWrite(LED\_G, HIGH);

digitalWrite(LED\_B, HIGH);

break;

case Color\_G: //绿色

digitalWrite(LED\_R, HIGH);

digitalWrite(LED\_G, LOW);

digitalWrite(LED\_B, HIGH);

break;

case Color\_B: //蓝色

digitalWrite(LED\_R, HIGH);

digitalWrite(LED\_G, HIGH);

digitalWrite(LED\_B, LOW);

break;

case Color\_RG: //黄色

digitalWrite(LED\_R, LOW);

digitalWrite(LED\_G, LOW);

digitalWrite(LED\_B, HIGH);

break;

case Color\_RB: //紫色

digitalWrite(LED\_R, LOW);

digitalWrite(LED\_G, HIGH);

digitalWrite(LED\_B, LOW);

break;

case Color\_GB: //青色

digitalWrite(LED\_R, HIGH);

digitalWrite(LED\_G, LOW);

digitalWrite(LED\_B, LOW);

break;

case Color\_RGB: //白色

digitalWrite(LED\_R, LOW);

digitalWrite(LED\_G, LOW);

digitalWrite(LED\_B, LOW);

break;

default:

break;

}

}

### 例子3

//模拟红绿灯

#define LED\_R D7

#define LED\_G D6

#define LED\_B D5

enum{ Color\_R, Color\_RG, Color\_G }; //红,黄,绿

void setup()

{

pinMode(LED\_R, OUTPUT);

pinMode(LED\_G, OUTPUT);

pinMode(LED\_B, OUTPUT);

}

void loop()

{

unsigned int i;

for (i = 0; i < 3; i++)

{

Change\_Color(i);

delay(1000);

};

}

void Change\_Color(unsigned char data\_color) //选择颜色函数

{

int j;

switch (data\_color)

{

case Color\_R: //红色

digitalWrite(LED\_R, HIGH);

digitalWrite(LED\_G, LOW);

digitalWrite(LED\_B, LOW);

delay(3000);

break;

case Color\_G: //绿色

digitalWrite(LED\_R, LOW);

digitalWrite(LED\_G, HIGH);

digitalWrite(LED\_B, LOW);

delay(8000);

break;

case Color\_RG:

for (j = 0; j < 5; j++){ //黄色

digitalWrite(LED\_R, HIGH);

digitalWrite(LED\_G, HIGH);

digitalWrite(LED\_B, LOW);

delay(1000);

digitalWrite(LED\_R, LOW);

digitalWrite(LED\_G, LOW);

digitalWrite(LED\_B, LOW);

delay(1000);

}

break;

default:

break;

};

}

## 舵机

顾名思义，舵机是用来控制舵的，比如轮船的方向舵，飞机的方向舵、升降舵等，这些

都需要控制一定的角度，但并非需要连续旋转。所以一般舵机都是只能转动一定的角度，这

里说的舵机主要应用于航模、车模和监控等多种领域。之前提及的舵机一般是正负90度之

间转动（连续旋转舵机除外），舵机内部是有直流电机，位置电位器和驱动反馈电路板组成，

当需要舵机转到一定角度时，输入信号会与标准信号比较，如果反馈位置不是所需要的位置，

电机则会朝向需要的方向转动，直到转到指定位置，电位器反馈信息促使电机停止转动。

舵机的控制信号实际上是PWM信号，周期不变，高电平的时间决定舵机的实际位置。

单片机中常用的PWM产生方式有2种，其一是通过定时器或者延时模拟出PWM信号，

其二是单片机内部包含PWM发生器。

**9g舵机的外型：**



servo类下成员函数

attach()---连接舵机

write()---角度控制

writeMicroseconds()---输入一个值单位为us,来控制舵机转动到相应角度，输入值为1000时舵机轴转动到逆时针最大位置。输入值为2000时舵机轴转动到顺时针最大位置。为1500时舵机轴在中间位置。

read()---读上一次舵机转动角度

attached()---连接舵机接口引脚

detach()---断开舵机连接

**servo类成员函数**

|  |  |
| --- | --- |
| 函数 | 说明 |
| attach() | 设定舵机的接口，只有9或10接口可利用。 |
| write() | 用于设定舵机旋转角度的语句，可设定的角度范围是0°到180°。 |
| writeMicroseconds() | 用于设定舵机旋转角度的语句，直接用微秒作为参数。 |
| read() | 用于读取舵机角度的语句，可理解为读取最后一条write()命令中的值。 |
| attached() | 判断舵机参数是否已发送到舵机所在接口。 |
| detach() | 使舵机与其接口分离，该接口（9或10）可继续被用作PWM接口 |

### 例子1：180度转舵

**接线：红线接VCC正极，橙色线接D6引脚，棕色或黑色的线接GND负极**

#include <Servo.h>

Servo myservo; //定义舵机对象，最多可以控制8路舵机

int pos = 0; //用于存储舵机位置的变量

//初始化

void setup()

{

myservo.attach(D6); //舵机控制信号引脚,连接舵机

Serial.begin(115200);

}

//主循环

void loop()

{

for(pos = 0; pos < 180; pos += 1) //从0度-180度

{ //步进角度1度

myservo.write(pos); //输入对应的角度值，舵机会转到此位置

delay(15); //15ms后进入下一个位置

Serial.println(pos);

}

for(pos = 180; pos>=1; pos-=1) //从180度-0度

{

myservo.write(pos); //输入对应的角度值，舵机会转到此位置

delay(15); //15ms后进入下一个位置

Serial.println(pos);

}

}

### 例子2：电位器控制舵机

#include <Servo.h>

//创建一个舵机控制对象

Servo myservo;

int potpin = A0;//该变量用于存储用电位器的模拟值

int val;//该变量用于存储舵机角度位置

void setup()

{

myservo.attach(D6);//该舵机由ESP8266MCU,D6引脚控制

Serial.begin(115200);

}

void loop()

{

val = analogRead(potpin); //读取电位器控制的模拟值 (范围在0-1023)

val = map(val, 0, 1023, 0, 179); //scale it to use it with the servo (value between 0 and 180)

myservo.write(val); //指定舵机转向的角度

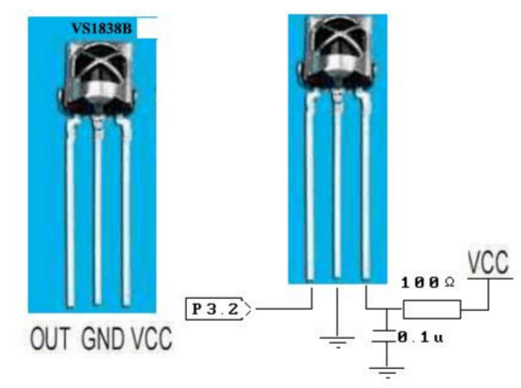
Serial.println(val);

delay(15); //等待15ms让舵机到达指定位置

}

# 实验5

## 红外遥控



* 外形尺寸：7.4X6.2X5.3mm（长X宽X高）
* 工作电压：2.7-5.5V
* 接收距离：20-23M
* 管脚间距：2.54mm
* 脚长：23mm
* 产品介绍
* 采用内、外双屏蔽直插式外形结构，抗光、电磁干扰能力强，宽角度及长距离接收。
* 广泛应该于
* 音响、电视、录影机、碟机、机顶盒等视听产品；冷气机、暧风机、电风扇、灯饰等家用电器。
  + 产品编号：VS1838B
  + 规　　格：7.4X6.2X5.3mm
  + 产品备注：接收距离20-23M

要使用遥控器控制arduino,首先我们得知道遥控器各按键对应的编码，不同的遥控器，不同的按键，不同的协议，对应不同的编码。

红外接收模块中的一体化红外接收头，内部集成了红外接收电路，包括红外检测二极管，放大器，限幅器，带通滤波电容，积分电路，比较器等。能够将接收到的调制波进行解调。**本次实验利用红外遥控器发出的红外载波信号，红外接收模块接收解调红外信号，来相应控制LED通断。**

使用红外接收需要库文件<IRremote>，了解库文件：

**IRrecv irrecv(pin);**

**设置pin定义的端口为红外信号接收端口**

**decode\_results results;**

**定义results变量为红外结果存放位置**

**irrecv.enableIRIn();**

**启动红外解码**

**int irrecv.decode(&results)**

**判断是否已经收到编码，如果成功返回1，数据放入results变量中，失败返回0，**

**irrecv.resume()**

**一旦编码解码成功，resume()函数必须被调用来恢复接收下一次编码**

### 红外按键的编码

#ifndef UNIT\_TEST

#include <Arduino.h>

#endif

#include <IRremoteESP8266.h>

#include <IRrecv.h>

#include <IRutils.h>

//An IR detector/demodulator is connected to GPIO pin 14(D5 on a NodeMCU

//board).

uint16\_t RECV\_PIN = 14;

IRrecv irrecv(RECV\_PIN);

decode\_results results;

void setup() {

Serial.begin(115200);

irrecv.enableIRIn(); //Start the receiver

}

void loop() {

if (irrecv.decode(&results)) {

//print() & println() can't handle printing long longs. (uint64\_t)

serialPrintUint64(results.value, HEX);

Serial.println("");

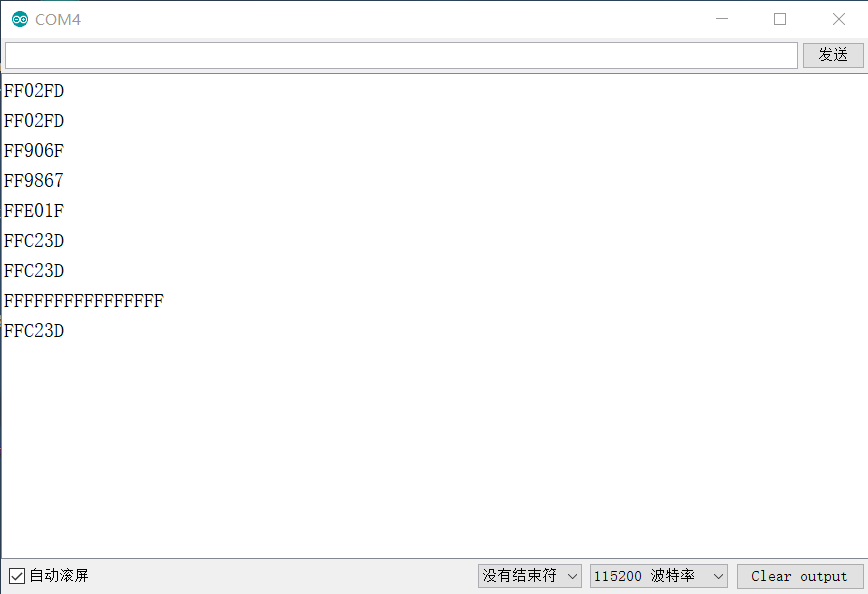
irrecv.resume(); //Receive the next value

}

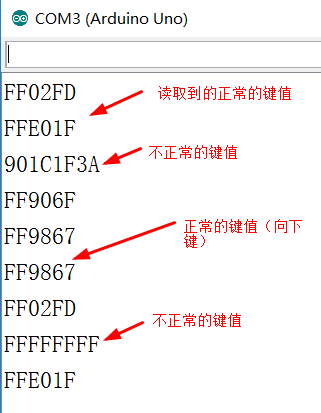
delay(100);

}

运行结果：



编译运行就可以，测试效果



**测试到的键值和自己硬件带的遥控器有关系，可以根据这些键值做操作**

**第1行**

**----------------------**

**FFA25D静音**

**FF629D停止**

**FFE21D播放/暂停**

**第2行**

**----------------------**

**FF22DD设置**

**FF02FD向上**

**FFC23D退出**

**第3行**

**----------------------**

**FFE01F向左**

**FFA857确认**

**FF906F向右**

**第4行**

**----------------------**

**FF9867向下**

### 红外遥控控制灯亮与灭

接线：led灯正极接到D6，其它的和上面的例子相同

#ifndef UNIT\_TEST

#include <Arduino.h>

#endif

#include <IRremoteESP8266.h>

#include <IRrecv.h>

#include <IRutils.h>

//An IR detector/demodulator is connected to GPIO pin 14(D5 on a NodeMCU

//board).

uint16\_t RECV\_PIN = 14;

IRrecv irrecv(RECV\_PIN);

decode\_results results;

//LED Pins

int LEDpin = D6;

void setup() {

Serial.begin(115200);

irrecv.enableIRIn(); //Start the receiver

pinMode(LEDpin, OUTPUT);

digitalWrite(LEDpin, LOW);

}

void loop() {

if (irrecv.decode(&results)) {

//print() & println() can't handle printing long longs. (uint64\_t)

serialPrintUint64(results.value, HEX);

Serial.println("");

if (results.value == 0xFF22DD) //若接收到按键设置按下的指令，打开LED

{

digitalWrite(LEDpin, HIGH);

}

else if (results.value == 0xFFC23D) //接收到退出按键按下，关闭LED

{

digitalWrite(LEDpin, LOW);

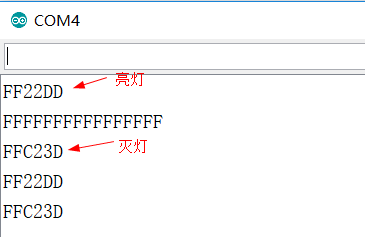
}

irrecv.resume(); //Receive the next value

}

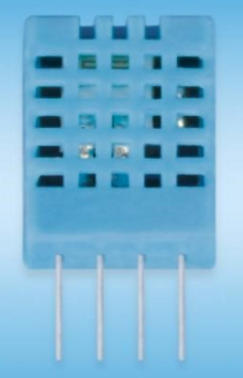
delay(100);

}



## DHT11温湿度传感器

　DHT11数字温湿度传感器是一款含有已校准数字信号输出的温湿度复合传感器。它应用专用的数字模块采集技术和温湿度传感技术，确保产品具有极高的可靠性与卓越的长期稳定性。传感器包括一个电阻式感湿元件和一个NTC测温元件，并与一个高性能8位单片机相连接。因此该产品具有品质卓越、超快响应、抗干扰能力强、性价比极高等优点。每个DHT11传感器都在极为精确的湿度校验室中进行校准。校准系数以程序的形式储存在OTP内存中，传感器内部在检测信号的处理过程中要调用这些校准系数。单线制串行接口，使系统集成变得简易快捷。超小的体积、极低的功耗，信号传输距离可达20米以上，使其成为各类应用甚至最为苛刻的应用场合的最佳选则。产品为 4 针单排引脚封装。连接方便，特殊封装形式可根据用户需求而提供。



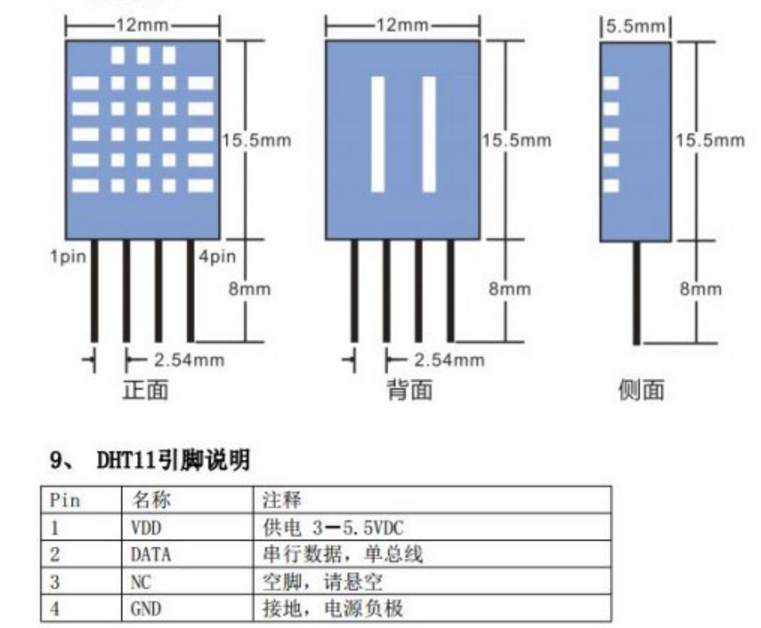
2.2DHT11的应用领域

DHT11广泛应用在一下几个方面：暖通、空调、测试及检测设备、汽车数据记录器、消费品自动控制、气象站、家电、湿度调节器、医疗、除湿器

2.3DHT11的产品特点

| **特点** |
| --- |
| 相对湿度和温度测量 |
| 全部校准，数字输出 |
| 卓越的长期稳定性 |
| 无需额外部件 |
| 超长的信号传输距离 |
| 超低能耗 |
| 4引脚安装 |
| 完全互换 |

2.4DHT11的外形尺寸



**从左**

**PIN1 VDD 3V-5.5V**

**PIN2 DATA 总线**

**PIN3 NC 空地,不接**

**PIN4 GND 地线**

**例子1：观察DHT11的数据**

一、器材

arduino UNO、面包板、DHT11温湿度传感器、连接线

这里说一下DHT11的基本情况：

（1）引脚说明：

1、VDD 供电 3.5V-5.5V DC

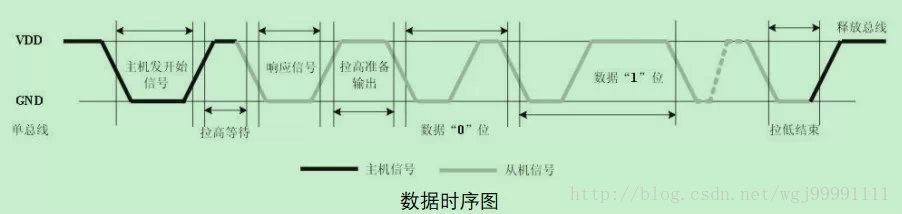
2、DATA 串行数据，单总线

3、GND 接地，电源负极。

（2）DHT11data数据格式：

一次传输40位数据=8bit湿度整数数据 + 8bit湿度小数数据 + 8bint温度整数数据 + 8bit温度小数数据 + 8bit校验位

3、时序图



将DHT11的正极与5V电源接口相连，负极与GND相连，中间的数据接口与2号引脚相连

DHT11.h库了解：

class dht11

{

public:

int read(int pin);

int humidity;

int temperature;

};

dht11.read() 获取输出端口的值

dht11.humidity 获取湿度

dht11.temperature 获取温度

**首先将温湿度需要的库文件，拷贝到编译器的libraries目录下。**



**然后在编译器中，编写下面的代码。**

//温湿度的数据针脚

#define DHT11PIN D5

//温湿度库对应的头文件

#include <dht11.h>

//全局变量

dht11 DHT11;

//数据

String strData;

//初始化函数

void setup()

{

//设置串口的波特率

Serial.begin(115200);

//设置针脚端口的模式

pinMode(DHT11PIN, OUTPUT);

}

//循环执行

void loop()

{

//读取温湿度的数据

int chk = DHT11.read(DHT11PIN);

//温度

float temp = (float)DHT11.temperature;

Serial.print("Tep: ");

Serial.print(temp);

Serial.println("C");

//湿度

float humi = (float)DHT11.humidity;

Serial.print("Hum: ");

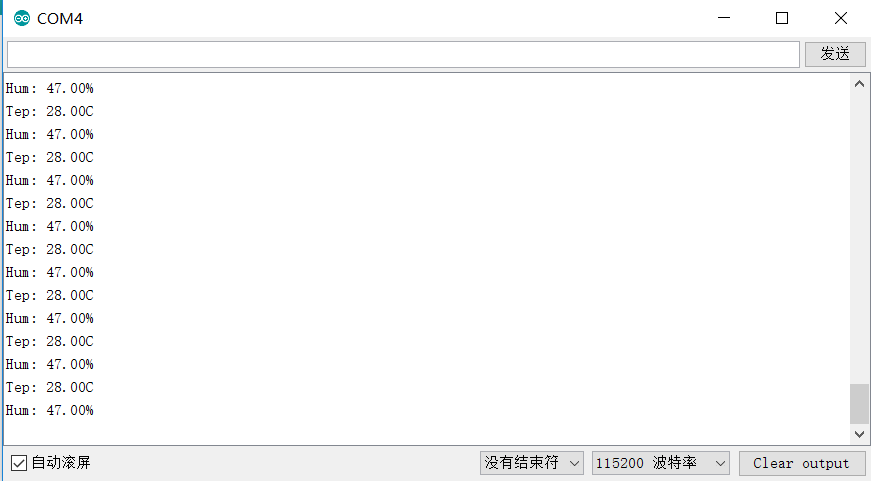
Serial.print((float)DHT11.humidity);

Serial.println("%");

delay(500);

}

运行结果：



## 倾斜开关

倾斜开关是内容带有一个金属滚珠的滚珠倾斜开关。滚珠开关：也叫碰珠开关、摇珠开关、钢珠开关、倾斜开关，倒顺开关、角度传感器。主要是利用滚珠在开关内随不同倾斜角度的变化，达到触发电路的目的。

实验原理 当开关一端低于水平位置倾斜，开关寻通，模拟口电压值为5V左右 （数字二进制表示为1023），点亮led灯。当另一端低于水平位置倾斜 ，开关停止，模拟口电压值为0V左右（数字二进制表示为0），熄灭led 灯。

**实验现象：手拿着面包板，当倾斜到一定程度时，Led等会灭，没有倾斜时，led亮。**

//初始化

void setup()

{

//打开串口

Serial.begin(115200);

//设置数字8引脚为辒出模式

pinMode(D2, OUTPUT);

}

//循环执行

void loop()

{

//定义变量i

int i;

//读取模拟5口电压值

i = analogRead(A0);

Serial.println(i);

if (i>200)//如果大于512（2.5V）

{

digitalWrite(D2, HIGH);//点亮led灯

}

else//否则

{

digitalWrite(D2, LOW);//熄灭led灯

}

delay(500);

}

## 超声波测距

这个模块是国外原创，现在国内应用最多的一种。

原理：

超声波测距原理是在超声波发射装置发出超声波，它的根据是接收器接到超声波时的时间差，与雷达测距原理相似。 超声波发射器向某一方向发射超声波，在发射时刻的同时开始计时，超声波在空气中传播，途中碰到障碍物就立即返回来，超声波接收器收到反射波就立即停止计时。

pulseIn()函数用来读取一个引脚的脉冲（HIGH或LOW）。

例如，如果value是HIGH，pulseIn()会等待引脚变为HIGH，开始计时，再等待引脚变为LOW并停止计时。

**语法：**pulseIn(pin, value)

pulseIn(pin, value, timeout)

**参数：**pin:你要进行脉冲计时的引脚号（int）。

value:要读取的脉冲类型，HIGH或LOW（int）。

**返回值：脉冲发生变化的时间us​**

**距离s(cm) =** pulseIn(pin, value) \*340m/s\*1/2

=pulseIn(pin, value) \*(34000/1000000)\*(1/2)

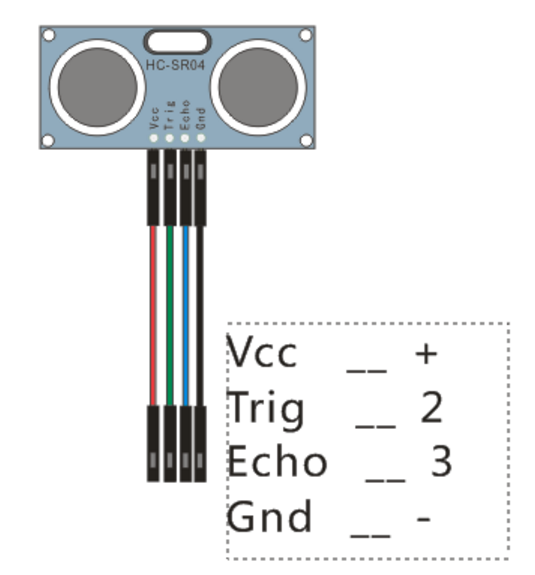
=pulseIn(pin, value) \*0.015

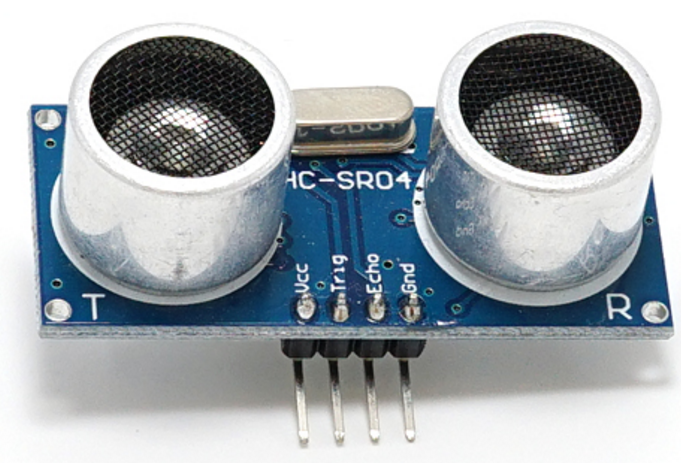
=pulseIn(pin, value) /58.8

**实验器材：**

1. Esp8266 NodeMCU 12E开发板
2. USB下载线
3. 面包板1个
4. LED灯1个
5. 超声波模块
6. 面包板跳线1扎







**主要技术参数**

1：使用电压：DC---5V

2：静态电流：小于2mA

3：电平输出：高5V

4：电平输出：底0V

5：感应角度：不大于15度

6：探测距离：2cm-450cm

7: 高精度 可达0.2cm

**引脚说明**

VCC -- 供5V电源

TRIG -- 触发控制信号输入

ECHO -- 回响信号输出等四个接口端

GND -- 为地线

### 例子1：测距

**代码1**

#define Trig D2 //引脚Tring 连接 IO D2

#define Echo D3 //引脚Echo 连接 IO D3

float cm; //距离变量

float temp; //

void setup() {

Serial.begin(115200);

pinMode(Trig, OUTPUT);

pinMode(Echo, INPUT);

}

void loop() {

//给Trig发送一个低高低的短时间脉冲,触发测距

digitalWrite(Trig, LOW); //给Trig发送一个低电平

delayMicroseconds(2); //等待 2微妙

digitalWrite(Trig, HIGH); //给Trig发送一个高电平

delayMicroseconds(10); //等待 10微妙

digitalWrite(Trig, LOW); //给Trig发送一个低电平

temp = float(pulseIn(Echo, HIGH)); //存储回波等待时间,

//pulseIn函数会等待引脚变为HIGH,开始计算时间,再等待变为LOW并停止计时

//返回脉冲的长度

//声速是:340m/1s 换算成 34000cm / 1000000μs => 34 / 1000

//因为发送到接收,实际是相同距离走了2回,所以要除以2

//距离(厘米) = (回波时间 \* (34 / 1000)) / 2

//简化后的计算公式为 (回波时间 \* 17)/ 1000

cm = (temp \* 17) / 1000; //把回波时间换算成cm

Serial.print("Echo =");

Serial.print(temp);//串口输出等待时间的原始数据

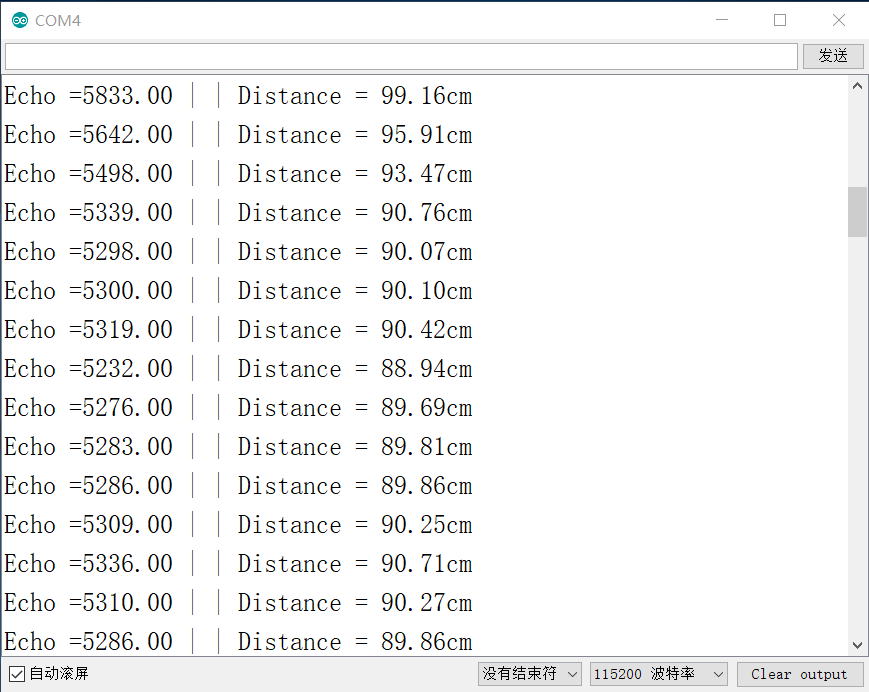
Serial.print(" | | Distance = ");

Serial.print(cm);//串口输出距离换算成cm的结果

Serial.println("cm");

delay(500);

}



**相同的代码**

//引脚定义

const int trig = D2; // 触发信号

const int echo = D3; // 反馈信号

//初始化

void setup() {

//反馈端口设置为输入

pinMode(echo, INPUT);

//触发端口设置为输出

pinMode(trig, OUTPUT);

//打开串口

Serial.begin(115200);

}

//主循环

void loop() {

long IntervalTime = 0; //定义一个时间变量

digitalWrite(trig, 1);//置高电平

delayMicroseconds(15);//延时15us

digitalWrite(trig, 0);//设为低电平

IntervalTime = pulseIn(echo, HIGH);//用自带的函数采样反馈的高电平的宽度，单位us

float S = IntervalTime / 58.00; //使用浮点计算出距离，单位cm

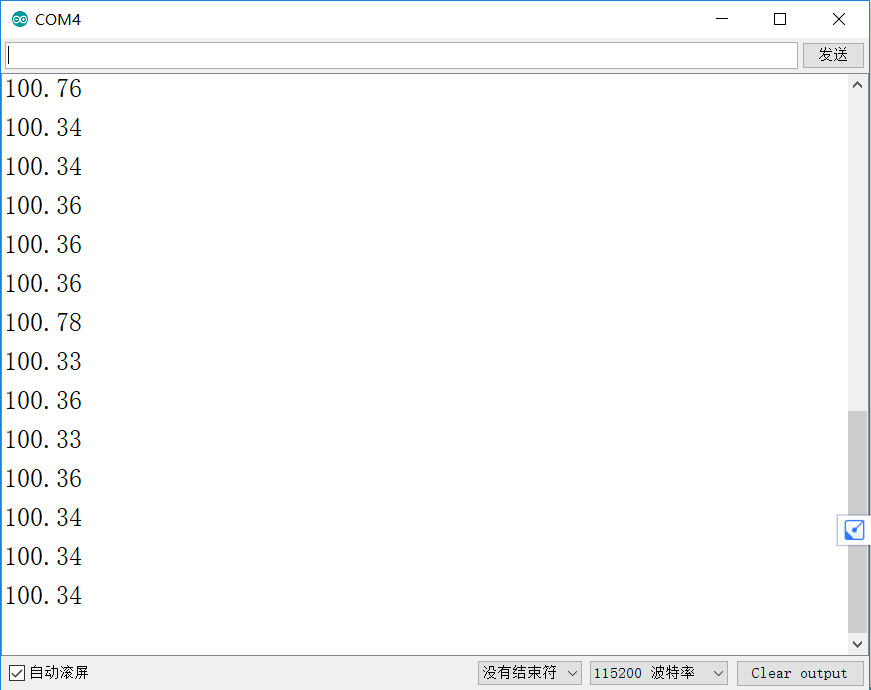
Serial.println(S);//通过串口输出距离数值

S = 0;

IntervalTime = 0;//对应的数值清零。

delay(500);//延时间隔决定采样的频率，根据实际需要变换参数

}



### 例子2：测距，亮灯提示

接线：led串联330欧的电阻，led正极接D5引脚

//引脚定义

const int trig = D2; // 触发信号

const int echo = D3; // 反馈信号

//初始化

void setup() {

//反馈端口设置为输入

pinMode(echo, INPUT);

//触发端口设置为输出

pinMode(trig, OUTPUT);

pinMode(D5,OUTPUT);

digitalWrite(D5,LOW);

//打开串口

Serial.begin(115200);

}

//主循环

void loop() {

long IntervalTime = 0; //定义一个时间变量

digitalWrite(trig, 1);//置高电平

delayMicroseconds(15);//延时15us

digitalWrite(trig, 0);//设为低电平

IntervalTime = pulseIn(echo, HIGH);//用自带的函数采样反馈的高电平的宽度，单位us

float S = IntervalTime / 58.00; //使用浮点计算出距离，单位cm

if(S < 100.00)

{

delay(10);

digitalWrite(D5,HIGH);

}

else

{

delay(10);

digitalWrite(D5,LOW);

}

Serial.println(S);//通过串口输出距离数值

S = 0;

IntervalTime = 0;//对应的数值清零。

delay(500);//延时间隔决定采样的频率，根据实际需要变换参数

}

## 步进电机使用

步进电机是将电脉冲信号转变为角位移或线位移的开环控制元步进电机件。在非超载的

情况下，电机的转速、停止的位置只取决于脉冲信号的频率和脉冲数，而不受负载变化的影

响，当步进驱动器接收到一个脉冲信号，它就驱动步进电机按设定的方向转动一个固定的角

度，称为“步距角”，它的旋转是以固定的角度一步一步运行的。可以通过控制脉冲个数来控制角位移量，从而达到准确定位的目的；同时可以通过控制脉冲频率来控制电机转动的速度和加速度，从而达到调速的目的。

步进电机驱动方式：

1、1相励磁法：每一瞬间只有一个线圈导通，其他线圈休息。其特点是励磁方法简单，

耗电低，精确度良好。但是力矩小、震动大，每次励磁信号走的角度是标称角度。

2、2相励磁法：每一瞬间有两个线圈同时导通，特点是力矩大、震动较小，每次励磁

转动角度是标称角度。

3、1-2相励磁法：1相和2相轮流交替导通，精度较高，且运转平滑。每送一个励磁信

号转动二分之一标称角度。有称为半步驱动。4相电机中，1、2种方式称4相4拍，3种称

4相8拍。

1相励磁、2相励磁和1-2相励磁方式如下表所示。

表2.15-1



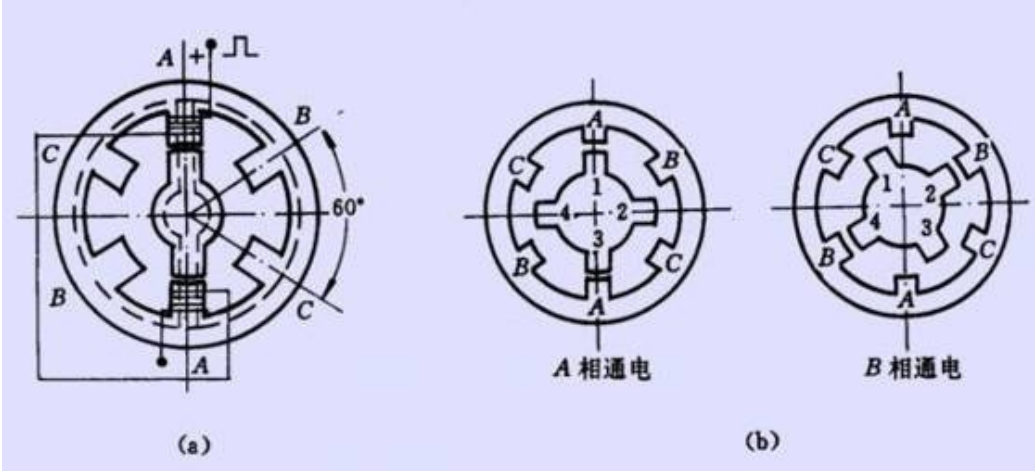


图2.15-1 简单示意图

步进电机种类很多。ULN2003只能驱动单极电机，也就是有中间抽头，电流只有一个

方向流向。双极电机需要H桥驱动，每个线圈可以正反向施加电压。

接线注意：由于正极接线到vin(5V)引脚

### 代码1

#include <Stepper.h>

//change this to the number of steps on your motor

#define STEPS 512

//定义引脚的连接

Stepper stepper(STEPS, D5, D6, D7, D8);

//上一次的读取模拟量

int previous = 0;

//初始化电机转动速度

void setup()

{

stepper.setSpeed(20);

}

//主循环

void loop()

{

//读取AD值，在LY-51S可以直接通过电位器调节

int val = analogRead(A0);

//把速度设置为这次和上次读取的差值

stepper.step(val - previous);

//记下这次读取值，方便下次比较

previous = val;

}

### 代码2

/\*

\* 步进电机跟随电位器旋转

\* (或者其他传感器)使用0号模拟口输入

\* 使用arduino IDE自带的Stepper.h库文件

\*/

#include <Stepper.h>

//这里设置步进电机旋转一圈是多少步

#define STEPS 100

//attached to设置步进电机的步数和引脚

Stepper stepper(STEPS, D5, D6, D7, D8);

// 定义变量用来存储历史读数

int previous = 0;

void setup()

{

//设置电机每分钟的转速为90步

stepper.setSpeed(90);

}

void loop()

{

//获取传感器读数

int val = analogRead(0);

//移动步数为当前读数减去历史读数

stepper.step(val - previous);

//保存历史读数

previous = val;

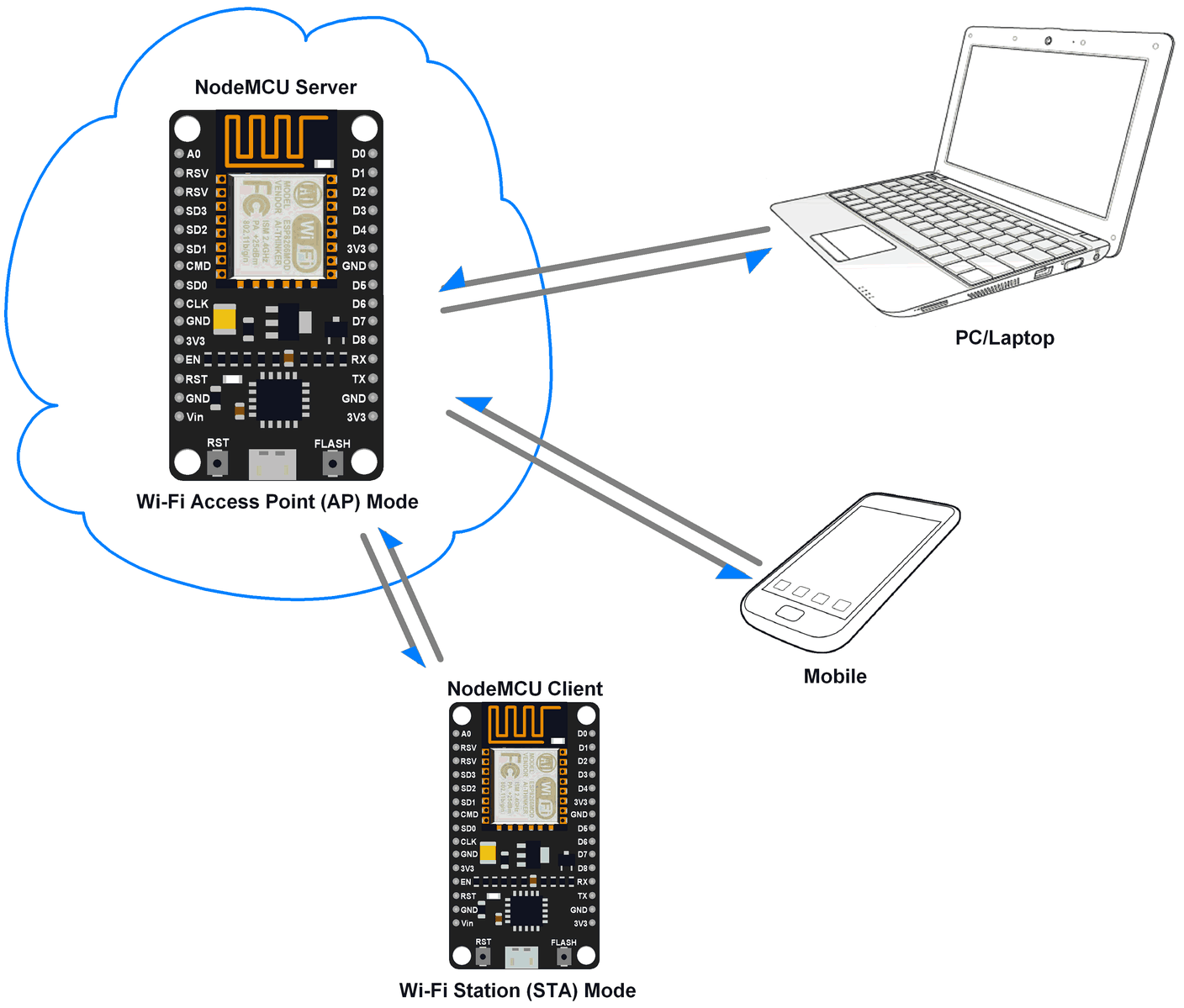
}

# 实验6

## 通过Wi-Fi + AP + HTTP服务器，实现本地数据发布

##### NodeMCU作为Wi-Fi使用AP模式的HTTP Server

NodeMCU(ESP 8266) wi-fi具有接入点（AP）模式，通过它可以创建无线LAN，任何支持Wi-Fi的设备都可以连接，如下图所示。



当然，我们也可以将NodeMCU当成一个HTTP服务器，让所有连接到它之上的Wi-Fi设备通过它获取到相应的信息。

##### HTTP协议

超文本传输​​协议（HTTP）是标准的应用层协议，用作服务器和客户端之间的请求响应协议。

它被广泛应用于物联网（IoT）嵌入式应用，每个传感器都连接到一个服务器，通过HTTP协议，我们可以将传感器所捕获的数据通过网络发送给每一个客户端。

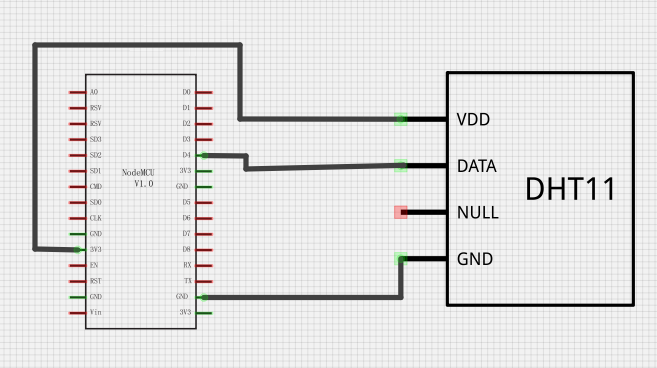
##### 实验：

将采集到的温湿度，通过NodeMCU中的HTTP服务器，发布到每个连接到它的Wi-Fi设备上。

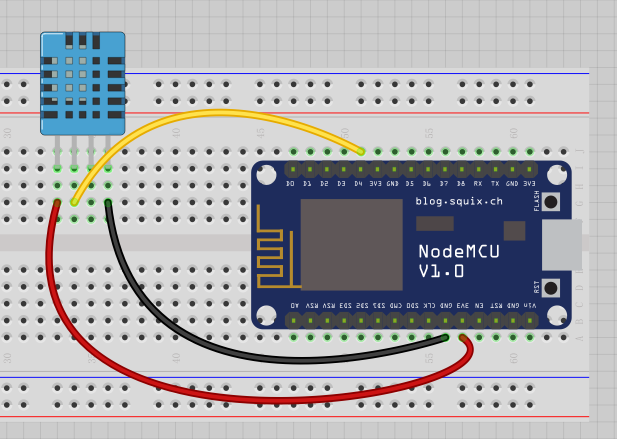
##### 实验器材：

1. NodeMCU板子
2. USB下载线
3. 面包板1个
4. 温度湿度传感器1个
5. 面包板跳线1扎

##### 实验原理图：



##### 实物图：



##### 代码：

|  |
| --- |
| //温湿度库对应的头文件  #include <dht11.h>  //网络服务所需头文件  #include <ESP8266WiFi.h>  #include <WiFiClient.h>  #include <ESP8266WebServer.h>  //设置AP的SSID和密码  const char \*ssid = "DHT11";  const char \*password = "12345678";  //全局对象，用提供web服务  ESP8266WebServer server(80);  //全局对象，用于获取温湿度  dht11 DHT11;  //温湿度的数据针脚  #define DHT11PIN D4  //访问http://192.168.4.1会看到的内容  void handleRoot() {  float h = getHumidity();  float t = getTemperature();  String str = "<head><meta charset=\"utf-8\" /><title>ESP-12E试验</title></head><h1>当前湿度(%):";  str += String(h);  str += "<br>当前温度(℃):";  str += String(t);  str += "</h1>";  server.send(200, "text/html", str);  }  void setup() {  delay(1000);  Serial.begin(115200);  Serial.println();  Serial.print("Configuring access point...");    //启动AP热点  WiFi.softAP(ssid, password);  //显示热点信息  IPAddress myIP = WiFi.softAPIP();  Serial.print("AP IP address: ");  Serial.println(myIP);  //绑定HTTP服务器/目录下所发布的信息，并启动服务  server.on("/", handleRoot);  server.begin();    Serial.println("HTTP server started");  }  //获取湿度  float getHumidity(){  int chk = DHT11.read(DHT11PIN);  return (float)DHT11.humidity;  }  //获取温度  float getTemperature(){  int chk = DHT11.read(DHT11PIN);  return (float)DHT11.temperature;  }  void loop() {  server.handleClient();  } |

编译上传。

##### 运行测试：

当你在串口监视器中看到：“HTTP server started”，以后表明此时NodeMCU的Wi-Fi AP + HTTP服务已经启动。

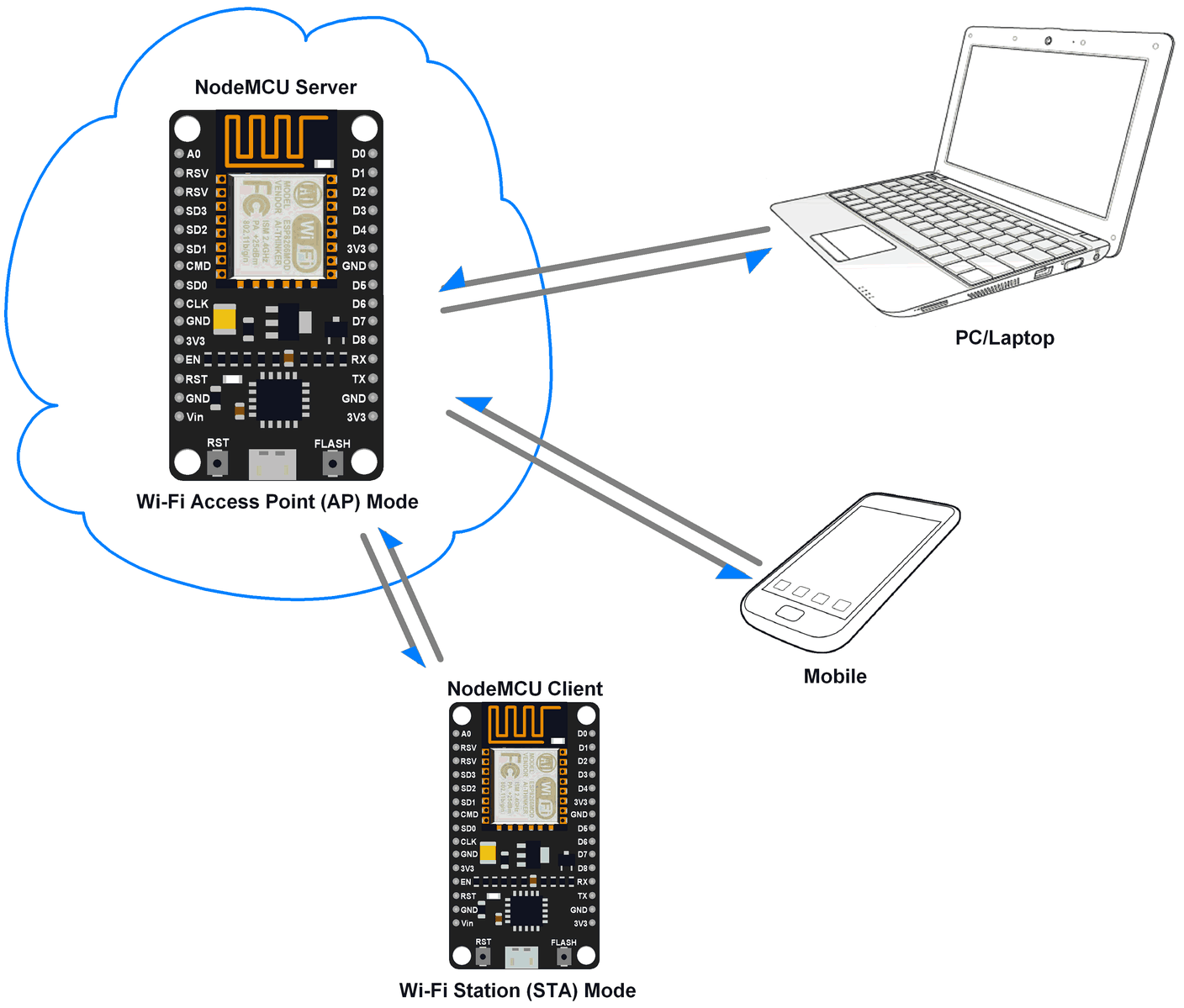
此时，你可以通过在代码中设置的SSID和密码连接到Wi-Fi热点中。打开浏览器，在网址中输入：http://192.168.4.1，就会看到当前DHT11所采集到的温度和湿度值了。

# 实验7

## 通过Wi-Fi + AP + HTTP服务器，实现本地传感器控制

##### NodeMCU作为Wi-Fi使用AP模式的HTTP Server

NodeMCU(ESP 8266) wi-fi具有接入点（AP）模式，通过它可以创建无线LAN，任何支持Wi-Fi的设备都可以连接，如下图所示。



这个网络结构和实验“通过Wi-Fi + AP + HTTP服务器，实现本地数据发布”是一样的。只不过这次，我们要通过连接到NodeMCU的设备，向它发出控制信号，使得NodeMCU可以按照控制信号，对连接在其上的传感器产生作用。

##### HTTP协议

为了使所有连接到NodeMCU的客户端设备，都可以和它进行通讯，我们依旧使用HTTP协议。在所有客户端设备上以浏览器作为统一的用户界面。

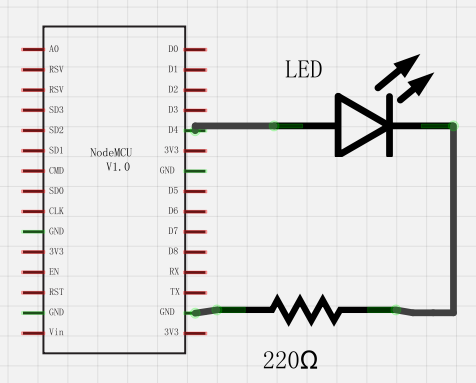
##### 实验：

通过手机或者笔记本等Wi-Fi设备，控制连接在NodeMCU上LED灯的亮灭。

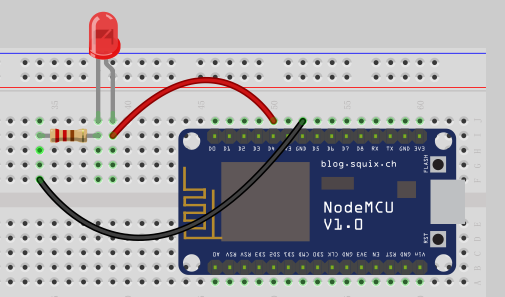
##### 实验器材：

1. NodeMCU板子
2. USB下载线
3. 面包板1个
4. LED灯1个
5. 面包板跳线1扎

##### 实验原理图：



##### 实物图：



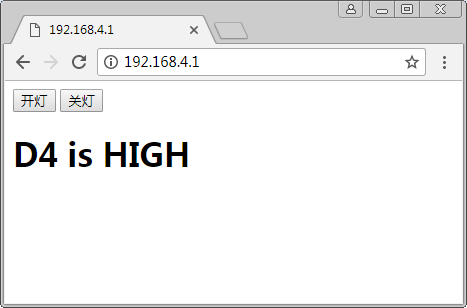
##### 代码：

|  |
| --- |
| //网络服务所需头文件  #include <ESP8266WiFi.h>  #include <WiFiClient.h>  #include <ESP8266WebServer.h>  //设置AP的SSID和密码  const char \*ssid = "NodeMCU";  const char \*password = "12345678";  //全局对象，用提供web服务  ESP8266WebServer server(80);  //全局字符串对象，用以完成网页所需的html脚本  String strHtml1 = "\  <!DOCTYPE html>\  <html>\  <head>\  <meta charset=\"utf-8\" />\  <title></title>\  </head>\  <body>\  <form id=form1>\  <input type=\"button\" value=\"开灯\" onclick=\'location.href=(\"/Digital?value=1\")\' />\  <input type=\"button\" value=\"关灯\" onclick=\'location.href=(\"/Digital?value=0\")\' />\  </form>\  <h1>D4 is ";    //全局字符串对象，用以完成网页所需的html脚本  String strHtml2 ="</h1>\  </body>\  </html>\  ";  //全局变量，表示LED的亮灭  int valueLED = 0;  void setup() {  delay(1000);  Serial.begin(115200);  Serial.println();  Serial.print("Configuring access point...");  //启动AP热点  WiFi.softAP(ssid, password);  //显示热点信息  IPAddress myIP = WiFi.softAPIP();  Serial.print("AP IP address: ");  Serial.println(myIP);  //设置D4端口模式为输出  pinMode(D4, OUTPUT);  digitalWrite(D4, 0);    //绑定HTTP服务器/目录下所发布的信息  server.on("/", handleRoot);  //绑定HTTP服务器/Digital目录下所发布的信息  server.on("/Digital", HTTP\_GET, handleDigital);  //启动服务  server.begin();    Serial.println("HTTP Server started");  }  //用户跳转到http://192.168.4.1/Digital时页面内容  void handleDigital()  {  String str = server.arg("value");  valueLED = str.toInt();  digitalWrite(D4, valueLED);  String strHtml = strHtml1 + String(valueLED?"HIGH" : "LOW") + strHtml2;  server.send(200, "text/html", strHtml);  }  //用户跳转到http://192.168.4.1时页面内容  void handleRoot() {  String strHtml = strHtml1 + String(valueLED?"HIGH" : "LOW") + strHtml2;  server.send(200, "text/html", strHtml);  }  void loop() {  server.handleClient();  } |

##### 运行测试：

当你在串口监视器中看到：“HTTP server started”，以后表明此时NodeMCU的Wi-Fi AP + HTTP服务已经启动。

此时，你可以通过在代码中设置的SSID和密码连接到Wi-Fi热点中。打开浏览器，在网址中输入：http://192.168.4.1，你会到如下图所示页面。点击“开灯”或“关灯”按钮，观察NodeMCU上的LED的亮灭，同时观察不同按钮按下时，网址的变化。



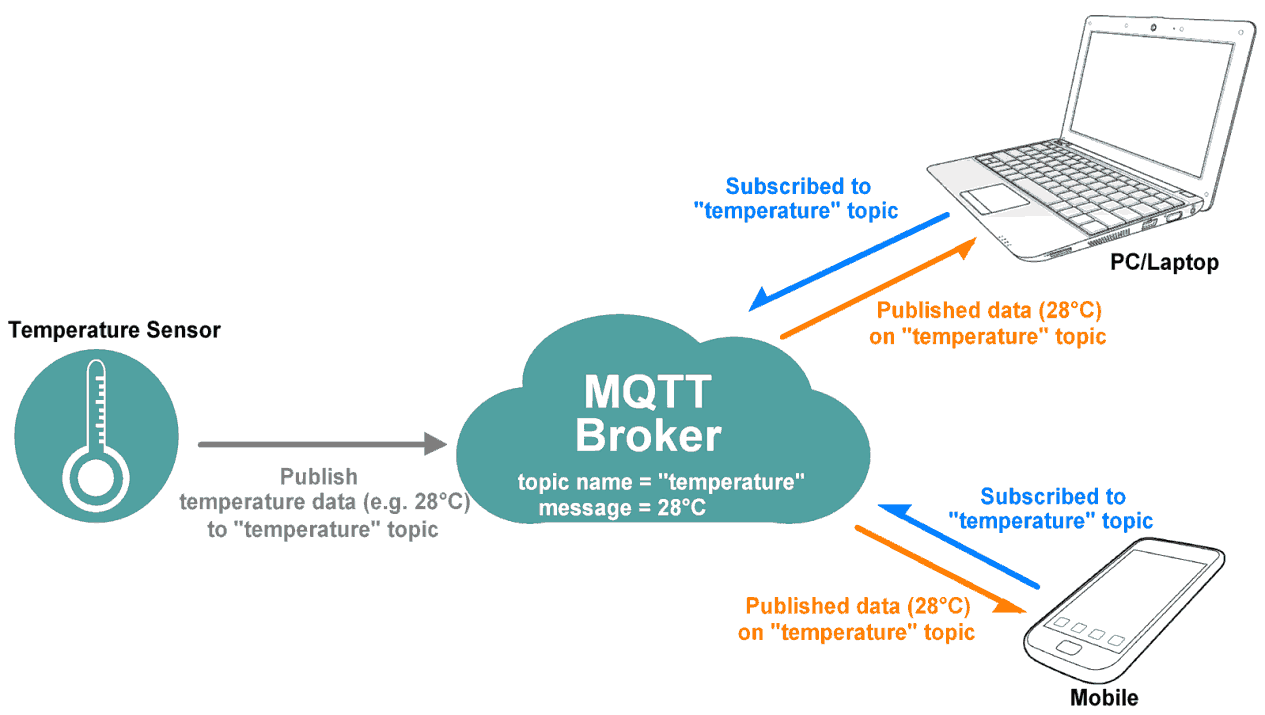
# 实验8

### 通过MQTT客户端完成与IOT的互联

##### MQTT介绍

MQTT（Message Queuing Telemetry Transport，消息队列遥测传输）是IBM开发的基于轻量级的基于发布 - 订阅的消息传递协议。该协议支持所有平台，几乎可以把所有联网物品和外部连接起来，被用来当做传感器和制动器（比如通过Twitter让房屋联网）的通信协议。

* 它比其他基于请求响应的API（如HTTP）更快。
* 它是基于TCP / IP协议开发的。
* 它允许远程定位设备在消息代理的帮助下连接，订阅，发布等到服务器上的特定主题。
* MQTT Broker / Message Broker是发件人和收件人之间的一个模块。这是消息验证，转换和路由的一个元素。
* 经纪人负责将消息分发给感兴趣的客户（订阅客户）。



例如，如果温度传感器发布“温度”主题的温度数据（消息），那么订购了“温度”主题的感兴趣的客户就可以获得如上所示的公布的温度数据。

MQTT广泛应用于物联网（IoT）嵌入式应用，每个传感器都连接到服务器，并且可以通过互联网进行控制。

NodeMCU是一个开源的物联网平台。这是一个运行在Espressif Systems的ESP8266 Wi-Fi SoC上的固件。它具有可用的物联网应用程序易于构建的无线网络。

NodeMCU的MQTT客户端模块与MQTT协议版本3.1.1相同。确保您的代理支持3.1.1版，并且已正确配置。让我们来看NodeMCU上用于MQTT的功能。

##### 实验：

编写Arduino程序将NodeMCU配置为MQTT客户端

1. 控制来自远程位置的LED亮度

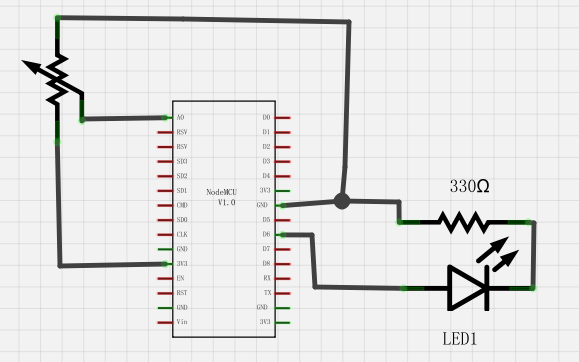
2. 通过POT（电位器）以数字形式发送电压到Adafruit仪表板的远程位置。

这里我们使用Adafruit服务器来进行MQTT客户端演示。

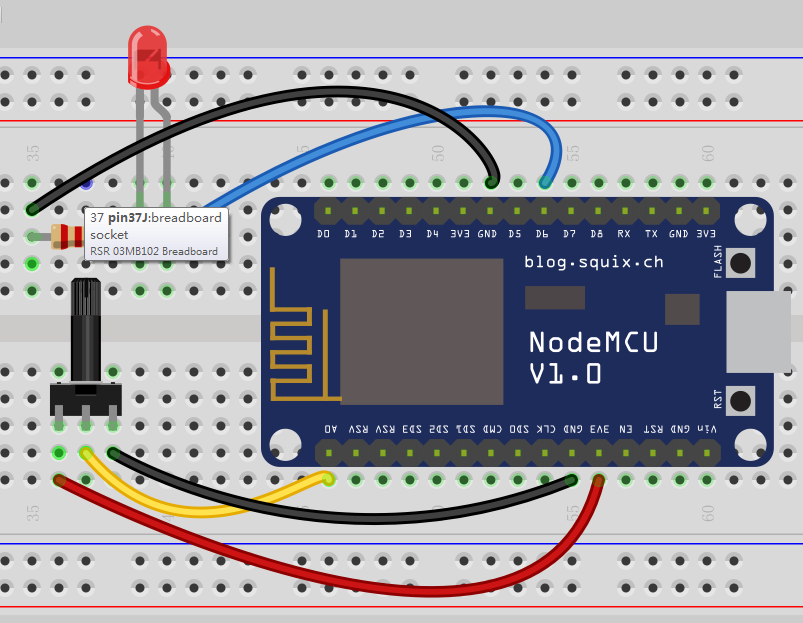
##### 实验器材：

1. NodeMCU板子
2. USB下载线
3. 面包板1个
4. LED灯1个
5. 330Ω电阻1个
6. 电位器1个
7. 面包板跳线1扎

##### 实验原理图：



##### 实物图：



##### 在物联网平台，添加仪表板

打开链接https://io.adafruit.com/，注册用户并登陆。

请从Adafruit IO控制板开始，成功创建仪表板后，我们将获得AIO密钥，稍后用于访问馈送数据。

具体步骤，请参阅链接：

[https://learn.adafruit.com/adafruit-io-basics-dashboards/creating-a-dashboard](https://learn.adafruit.com/adafruit-io-basics-dashboards/creating-a-dashboard" \t "_blank)

一旦我们在Adafruit上创建了仪表盘，我们可以添加各种可用于控制设备的模块以及监视设备的状态。

##### 安装所需的库

我们将需要使用Arduino库管理器来安装**Adafruit IO**，**Adafruit MQTT**和**ArduinoHttpClient**库。

打开Arduino IDE并导航到**项目->加载库->管理库...**



库管理器窗口将弹出。现在在搜索框中输入**Adafruit IO Arduino**，然后点击**Adafruit IO Arduino**库选项上的安装来安装版本2.6.0或更高版本。



用同样的方法安装**Adafruit MQTT(版本0.17.0或更高)**和**ArduinoHttpClient(0.3.0或更高)**库。

##### 代码：

|  |
| --- |
| #include <ESP8266WiFi.h>  #include "Adafruit\_MQTT.h"  #include "Adafruit\_MQTT\_Client.h"  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* WiFi Access Point \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  #define WLAN\_SSID "" //你的WI-FI的SSID  #define WLAN\_PASS "" //你的WI-FI的密码  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Adafruit.io Setup \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  #define AIO\_SERVER "io.adafruit.com"  #define AIO\_SERVERPORT 1883 // use 8883 for SSL  #define AIO\_USERNAME "" //你在io.adafruit.com上注册的用户名  #define AIO\_KEY "" //你在io.adafruit.com所获得的AIO  /\*\*\*\*\*\*\*\*\*\*\*\* Global State (you don't need to change this!) \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  // Create an ESP8266 WiFiClient class to connect to the MQTT server.  WiFiClient client;  // Setup the MQTT client class by passing in the WiFi client and MQTT server and login details.  Adafruit\_MQTT\_Client mqtt(&client, AIO\_SERVER, AIO\_SERVERPORT, AIO\_USERNAME, AIO\_KEY);  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Feeds \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  // Setup a feed called 'photocell' for publishing.  // Notice MQTT paths for AIO follow the form: <username>/feeds/<feedname>  Adafruit\_MQTT\_Publish photocell = Adafruit\_MQTT\_Publish(&mqtt, AIO\_USERNAME "/feeds/test");  // Setup a feed called 'onoff' for subscribing to changes.  Adafruit\_MQTT\_Subscribe onoffbutton = Adafruit\_MQTT\_Subscribe(&mqtt, AIO\_USERNAME "/feeds/Switch");  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Sketch Code \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  // Bug workaround for Arduino 1.6.6, it seems to need a function declaration  // for some reason (only affects ESP8266, likely an arduino-builder bug).  void MQTT\_connect();  void setup() {  Serial.begin(115200);  delay(10);  Serial.println(F("Adafruit MQTT demo"));  // Connect to WiFi access point.  Serial.println(); Serial.println();  Serial.print("Connecting to ");  Serial.println(WLAN\_SSID);  WiFi.begin(WLAN\_SSID, WLAN\_PASS);  while (WiFi.status() != WL\_CONNECTED) {  delay(500);  Serial.print(".");  }  Serial.println();  Serial.println("WiFi connected");  Serial.println("IP address: "); Serial.println(WiFi.localIP());  // Setup MQTT subscription for onoff feed.  mqtt.subscribe(&onoffbutton);  pinMode(D6, OUTPUT);  }  uint32\_t x=0;  void loop() {  // Ensure the connection to the MQTT server is alive (this will make the first  // connection and automatically reconnect when disconnected). See the MQTT\_connect  // function definition further below.  MQTT\_connect();  // this is our 'wait for incoming subscription packets' busy subloop  // try to spend your time here  Adafruit\_MQTT\_Subscribe \*subscription;  while ((subscription = mqtt.readSubscription(5000))) {  if (subscription == &onoffbutton) {  Serial.print(F("Got: "));  Serial.println((char \*)onoffbutton.lastread);  String value = (char \*)onoffbutton.lastread;  Serial.println(value);  if(!value.compareTo("开"))  {  digitalWrite(D6,HIGH);  }  if(!value.compareTo("关"))  {  digitalWrite(D6,LOW);  }    }  }  // Now we can publish stuff!  int value = analogRead(A0);  x = map(value,100,1000,1,100);  Serial.print(F("\nSending photocell val "));  Serial.print(x);  Serial.print("...");  if (! photocell.publish(x++)) {  Serial.println(F("Failed"));  } else {  Serial.println(F("OK!"));  }  // ping the server to keep the mqtt connection alive  // NOT required if you are publishing once every KEEPALIVE seconds  /\*  if(! mqtt.ping()) {  mqtt.disconnect();  }  \*/  }  // Function to connect and reconnect as necessary to the MQTT server.  // Should be called in the loop function and it will take care if connecting.  void MQTT\_connect() {  int8\_t ret;  // Stop if already connected.  if (mqtt.connected()) {  return;  }  Serial.print("Connecting to MQTT... ");  uint8\_t retries = 3;  while ((ret = mqtt.connect()) != 0) { // connect will return 0 for connected  Serial.println(mqtt.connectErrorString(ret));  Serial.println("Retrying MQTT connection in 5 seconds...");  mqtt.disconnect();  delay(5000); // wait 5 seconds  retries--;  if (retries == 0) {  // basically die and wait for WDT to reset me  while (1);  }  }  Serial.println("MQTT Connected!");  } |

编译上传。

##### 运行测试：

当你点击Adafruit IO控制板中的开关时，你可以观察到LED的亮灭变换。当你转动电位器时，你可以观察到控制面板中仪表的变化。