



网页版

材料

硬件

- ES8266 nodeMCU 开发板
- 舵机

软件

- Arduino IDE

一句话原理

十二月 17 星期三 将 ESP8266 NodeMCU 看成一个可以联网的单片机，可以通过所编写 html 网页的按钮来控制单片机。

代码


1) 网络

十二月 17 星期三 电脑和 NodeMCU 所连接为一个网络（在此为自己的手机热点）

```
1  #include <ESP8266WiFi.h>           // 本程序使用 ESP8266WiFi库
2  #include <ESP8266WiFiMulti.h>      // ESP8266WiFiMulti库
3  #include <ESP8266WebServer.h>      // ESP8266WebServer 库
4
5  ESP8266WiFiMulti wifiMulti;        // 建立 ESP8266WiFiMulti对象,对象名称是 'wifiMulti'
6  ESP8266WebServer esp8266_server(80); // 建立网络服务器对象，该对象用于响应 HTTP 请求。监听端口
   (80)
7  char ssid[] = "TX";                //wifi名称
8  char pswd[] = "20020603wz";        //wifi密码
9  char html[] = "<form action=\"/LED\" method=\"POST\"><input type=\"submit\" value=\"Toggle LED\"></form>"; //所实现网页html代码
10
11 void setup(void){
12     Serial.begin(115200);           // 启动串口通讯
13     /*****在此开始你的初始化*****/
14
15     /*****在此结束你的初始化*****/
16
17     wifiMulti.addAP(ssid, pswd);    // 将需要连接的一系列WiFi ID和密码输入这里
18     wifiMulti.addAP(ssid, pswd);    // ESP8266-NodeMCU再启动后会扫描当前网络
19
20     Serial.println("Connecting ..."); // 则尝试使用此处存储的密码
   进行连接。
21
22     int i = 0;
23     while (wifiMulti.run() != WL_CONNECTED) { // 此处的wifiMulti.run()是重点。通过wifiMulti.run(), NodeMCU将会在当前
24         delay(1000);                          // 环境中搜索 addAP 函数所存储的 WiFi。如果搜到
   多个存储的 WiFi 那么 NodeMCU
25         Serial.print(i++); Serial.print(' '); // 将会连接信号最强的那一个 WiFi 信号。
```

```
26     } // 一旦连接WiFi成功，wifiMulti.run()将会返回“WL_CONNECTED”。这也是
27     // 此处while循环判断是否跳出循环的条件。
28
29     // WiFi连接成功后将通过串口监视器输出连接成功信息
30     Serial.println('\n');
31     Serial.print("Connected to ");
32     Serial.println(WiFi.SSID()); // 通过串口监视器输出连接的WiFi名称
33     Serial.print("IP address:\t");
34     Serial.println(WiFi.localIP()); // 通过串口监视器输出 ESP8266-NodeMCU 的 IP
35
36     esp8266_server.begin(); // 启动网站服务
37     esp8266_server.on("/", HTTP_GET, handleRoot); // 设置服务器根目录即 '/' 的函数 'handleRoot'
38     esp8266_server.on("/LED", HTTP_POST, handleLED); // 设置处理 LED 控制请求的函数 'handleLED'
39     esp8266_server.onNotFound(handleNotFound); // 设置处理 404 情况的函数 'handleNotFound'
40
41     Serial.println("HTTP esp8266_server started");// 告知用户ESP8266网络服务功能已经启动
42 }
43
44 void loop(void){
45     esp8266_server.handleClient(); // 检查http服务器访问
46 }
47
48 /*设置服务器根目录即 '/' 的函数 'handleRoot'
49 该函数的作用是每当有客户端访问NodeMCU服务器根目录时，
50 NodeMCU都会向访问设备发送 HTTP 状态 200（Ok）这是 send 函数的第一个参数。
51 同时NodeMCU还会向浏览器发送 HTML 代码，以下示例中 send 函数中第三个参数，
52 也就是双引号中的内容就是NodeMCU发送的HTML代码。该代码可在网页中产生LED控制按钮。
53 当用户按下按钮时，浏览器将会向NodeMCU的/LED页面发送HTTP请求，请求方式为POST。
54 NodeMCU接收到此请求后将会执行 handleLED 函数内容*/
55 void handleRoot() {
56     /*****在此开始你的html*****/
57     esp8266_server.send(200, "text/html", html);
58     /*****在此结束你的html*****/
59 }
60
61 //处理 LED 控制请求的函数 'handleLED'
62 void handleLED() {
63     /*****在此开始你的代码*****/
64
65     /*****在此结束你的代码*****/
66     esp8266_server.sendHeader("Location", "/"); // 跳转回页面根目录
67     esp8266_server.send(303); // 发送 Http 相应代码 303 跳转
68 }
69
70 // 设置处理 404 情况的函数 'handleNotFound'
71 void handleNotFound(){
72     esp8266_server.send(404, "text/plain", "404: Not found"); // 发送 HTTP 状态 404（未找到页面）并向浏览器发送文字 "404: Not found"
73 }
```

2) 控制

 在此用 servo.h 库

```
1 #include <Servo.h>
2 Servo myservo; //定义舵机
3
4 void setup(void){
5     pinMode(LED_BUILTIN, OUTPUT); //设置内置LED引脚为输出模式以便控制LED
6     myservo.attach(2); //舵机的IO口，nodemcu的D4口
7     myservo.write(0); //上电时舵机归零垂直（0°）
8 }
9
10 void handleLED() {
11     if(flag == 0)
12     {
13         myservo.write(170); //收到“on”的指令后舵机旋转170度
```

```

14   delay(1000); //延时1秒
15   myservo.write(0);      //舵机归零，回到垂直状态，不要修改！
16   flag = 1;
17 }
18
19 else if(flag == 1)
20 {
21   myservo.write(170);    //收到“off”的指令后舵机旋转170度
22   delay(1000); //延时1秒
23   myservo.write(0);      //舵机归零，回到垂直状态，不要修改！
24   flag = 0;
25 }
26 digitalWrite(LED_BUILTIN, !digitalRead(LED_BUILTIN)); // 改变LED的点亮或者熄灭状态
27 esp8266_server.sendHeader("Location", "/");           // 跳转回页面根目录
28 esp8266_server.send(303);                             // 发送Http相应代码303 跳转
29 }

```

C

3) 总体代码

```

1  #include <ESP8266WiFi.h>          // 本程序使用 ESP8266WiFi库
2  #include <ESP8266WiFiMulti.h>    //  ESP8266WiFiMulti库
3  #include <ESP8266WebServer.h>    //  ESP8266WebServer库
4  #include <Servo.h>
5
6  ESP8266WiFiMulti wifiMulti;      // 建立ESP8266WiFiMulti对象,对象名称是 'wifiMulti'
7  ESP8266WebServer esp8266_server(80); // 建立网络服务器对象，该对象用于响应HTTP请求。监听端口
   (80)
8
9  char ssid[] = "TX";              //wifi名称
10 char pswd[] = "20020603wz";      //wifi密码
11 char html[] = "<form action=\"/LED\" method=\"POST\"><input type=\"submit\" value=\"Toggle LED\"></form>"; //所实现网页html代码
12
13 int flag = 0;
14 Servo myservo; //定义舵机
15
16 void setup(void){
17   Serial.begin(115200);           // 启动串口通讯
18   /*****在此开始你的初始化*****/
19   pinMode(LED_BUILTIN, OUTPUT); //设置内置LED引脚为输出模式以便控制LED
20   myservo.attach(2); //舵机的IO口，nodemcu的D4口
21   myservo.write(0); //上电时舵机归零垂直（0°）
22   /*****在此结束你的初始化*****/
23
24   wifiMulti.addAP(ssid, pswd); // 将需要连接的一系列WiFi ID和密码输入这里
25   wifiMulti.addAP(ssid, pswd); // ESP8266-NodeMCU再启动后会扫描当前网络
26
27   Serial.println("Connecting ..."); // 则尝试使用此处存储的密码
   进行连接。
28
29   int i = 0;
30   while (wifiMulti.run() != WL_CONNECTED) { // 此处的wifiMulti.run()是重点。通过wifiMulti.run(), NodeMCU将会在当前
   环境中搜索addAP函数所存储的WiFi。如果搜到
31     delay(1000); // 环境中搜索addAP函数所存储的WiFi。如果搜到
   多个存储的WiFi那么NodeMCU
32     Serial.print(i++); Serial.print(' '); // 将会连接信号最强的那一个WiFi信号。
33   } // 一旦连接WiFi成功，wifiMulti.run()将会返回“WL_CONNECTED”。这也是
   // 此处while循环判断是否跳出循环的条件。
34
35
36   // WiFi连接成功后将通过串口监视器输出连接成功信息
37   Serial.println('\n');
38   Serial.print("Connected to ");
39   Serial.println(WiFi.SSID()); // 通过串口监视器输出连接的WiFi名称
40   Serial.print("IP address:\t");
41   Serial.println(WiFi.localIP()); // 通过串口监视器输出ESP8266-NodeMCU的IP
42
43   esp8266_server.begin(); // 启动网站服务
44   esp8266_server.on("/", HTTP_GET, handleRoot); // 设置服务器根目录即 '/' 的函数 'handleRoot'
45   esp8266_server.on("/LED", HTTP_POST, handleLED); // 设置处理LED控制请求的函数 'handleLED'
46   esp8266_server.onNotFound(handleNotFound); // 设置处理404情况的函数 'handleNotFound'

```

```
47   und'  
48  
49   Serial.println("HTTP esp8266_server started");// 告知用户ESP8266网络服务功能已经启动  
50 }  
51  
52 void loop(void){  
53     esp8266_server.handleClient();           // 检查http服务器访问  
54 }  
55  
56 /*设置服务器根目录即 '/' 的函数 'handleRoot'  
57 该函数的作用是每当有客户端访问NodeMCU服务器根目录时，  
58 NodeMCU都会向访问设备发送 HTTP 状态 200（Ok）这是 send函数的第一个参数。  
59 同时NodeMCU还会向浏览器发送 HTML 代码，以下示例中 send函数中第三个参数，  
60 也就是双引号中的内容就是NodeMCU发送的HTML代码。该代码可在网页中产生LED控制按钮。  
61 当用户按下按钮时，浏览器将会向NodeMCU的/LED页面发送HTTP请求，请求方式为POST。  
62 NodeMCU接收到此请求后将会执行 handleLED函数内容*/  
63 void handleRoot() {  
64     /*****在此开始你的html*****/  
65     esp8266_server.send(200, "text/html", html);  
66     /*****在此结束你的html*****/  
67 }  
68 //处理LED控制请求的函数 'handleLED'  
69 void handleLED() {  
70     /*****在此开始你的代码*****/  
71     if(flag == 0){  
72         myservo.write(170);      //收到“on”的指令后舵机旋转170度  
73         delay(1000); //延时1秒  
74         myservo.write(0);        //舵机归零，回到垂直状态，不要修改！  
75         flag = 1;  
76     }  
77     else if(flag == 1){  
78         myservo.write(170);      //收到“off”的指令后舵机旋转170度  
79         delay(1000); //延时1秒  
80         myservo.write(0);        //舵机归零，回到垂直状态，不要修改！  
81         flag = 0;  
82     }  
83     digitalWrite(LED_BUILTIN, !digitalRead(LED_BUILTIN)); // 改变LED的点亮或者熄灭状态  
84     /*****在此结束你的代码*****/  
85     esp8266_server.sendHeader("Location", "/");           // 跳转回页面根目录  
86     esp8266_server.send(303);                             // 发送Http相应代码303 跳转  
87 }  
88  
89 // 设置处理404情况的函数 'handleNotFound'  
90 void handleNotFound(){  
91     esp8266_server.send(404, "text/plain", "404: Not found"); // 发送 HTTP 状态 404（未找到  
92     页面）并向浏览器发送文字 "404: Not found"  
93 }
```

C ∨

效果展示

1) upload

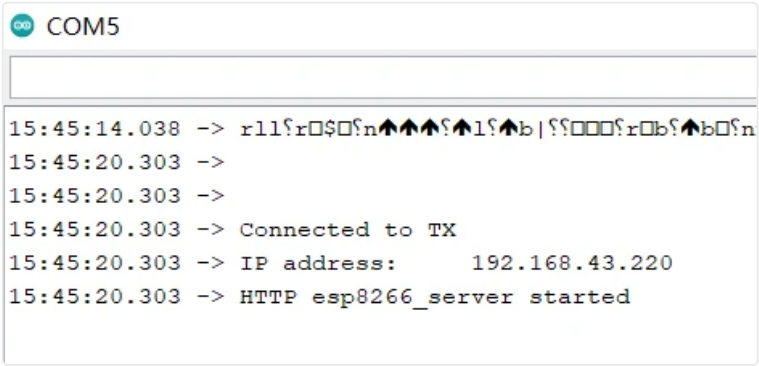
```
Executable segment sizes:  
ICACHE : 32768      - flash instruction cache  
IROM   : 273596     - code in flash      (default or ICACHE_FLASH_ATTR)  
IRAM   : 28369 / 32768 - code in IRAM      (IRAM_ATTR, ISRs...)  
DATA   : 1592 )      - initialized variables (global, static) in RAM/HEAP  
RODATA : 1080 ) / 81920 - constants          (global, static) in RAM/HEAP  
BSS    : 26072 )      - zeroed variables      (global, static) in RAM/HEAP  
  
Sketch uses 304637 bytes (29%) of program storage space. Maximum is 1044464 bytes.  
Global variables use 28744 bytes (35%) of dynamic memory, leaving 53176 bytes for local variables. Maximum is 81920 bytes.  
esptool.py v3.0  
Serial port COM5  
Connecting....  
Chip is ESP8266EX  
Features: WiFi  
Crystal is 26MHz  
MAC: 30:83:98:a2:cd:d6  
Uploading stub...  
Running stub...  
Stub running...  
Configuring flash size...  
Auto-detected Flash size: 4MB
```



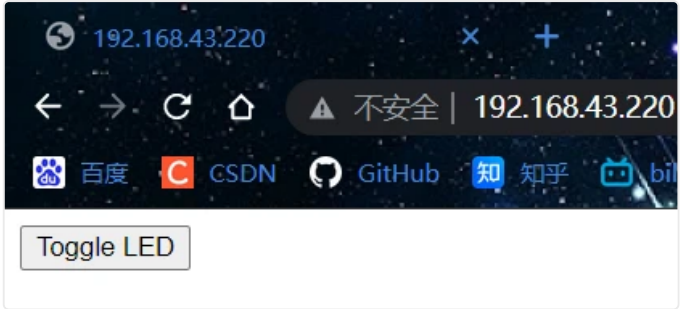
```
Compressed 308784 bytes to 224168...
Writing at 0x00000000... (7 %)
Writing at 0x00004000... (14 %)
Writing at 0x00008000... (21 %)
Writing at 0x0000c000... (28 %)
Writing at 0x00010000... (35 %)
Writing at 0x00014000... (42 %)
Writing at 0x00018000... (50 %)
Writing at 0x0001c000... (57 %)
Writing at 0x00020000... (64 %)
Writing at 0x00024000... (71 %)
Writing at 0x00028000... (78 %)
Writing at 0x0002c000... (85 %)
Writing at 0x00030000... (92 %)
Writing at 0x00034000... (100 %)
Wrote 308784 bytes (224168 compressed) at 0x00000000 in 19.8 seconds (effective 124.5 kbit/s)...
Hash of data verified.

Leaving...
Hard resetting via RTS pin...
```

2) serial monitor



3) webpage



4) Servo

