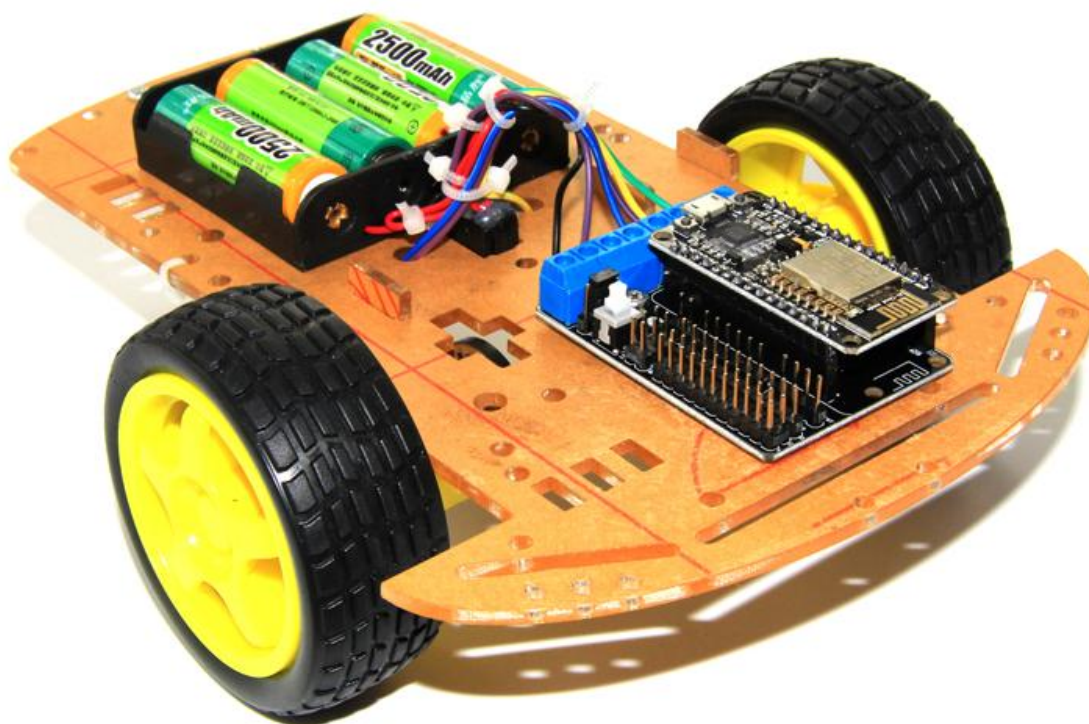




深圳四博智联科技有限公司

基于 ESP12E Dev Kit 的开发教程



二〇一五年五月



目 录

一 ESP12E Dev Kit 开发步骤.....	1
二 烧写固件	1
三 程序调试工具	4
3.1 “Hello World”	4
3.2 GPIO 测试.....	6
3.3 系统功能	7
3.4 WiFi 测试.....	8
3.5 文件操作	9
3.6 程序编辑	12
四 实例代码	13
4.1 init.lua 文件	13
4.2 WiFi 模式.....	14
4.3 PWM 功能	19
五 资料汇总	20



一 ESP12E Dev Kit 开发步骤

如果刚刚拿到 ESP12E Dev Kit，在 Windows 下的开发步骤如下。

- 步骤 1：硬件准备：开发 PC 机一台。MicroUSB 线一根（非 MiniUSB）。
- 步骤 2：软件准备：
 - （a）ESP12E Dev Kit 的核心是基于 NodeMCU 的，最新版本的 NodeMCU 固件下载地址：

<https://github.com/nodemcu/nodemcu-firmware/releases>。

Tips: 固件分为 Float 版本和 Integer 版本。为了最大限度降低运行内存消耗，通常情况下选用 Integer 版本。

（b）固件下载工具

nodemcu_flasher32bit.exe/nodemcu_flasher64bit.exe，请根据电脑系统选择 Win32 或 Win64 版本。最新版下载地址：<https://github.com/nodemcu/nodemcu-flasher>。

（c）程序调试工具

程序调试使用 LuaLoader。最新版下载地址：

<https://github.com/GeoNomad/LuaLoader>。

或者：

<http://benlo.com/esp8266/index.html#LuaLoader>。

（d）其他辅助工具：网络调试工具、串口工具等

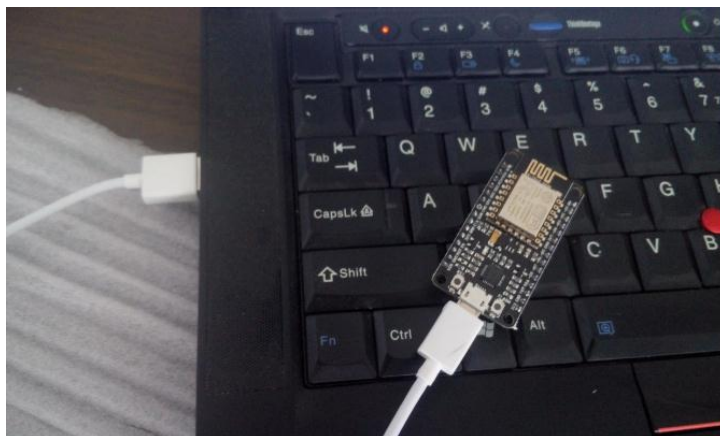
- 步骤 3：按照本章第二节内容烧写固件。
- 步骤 4：按照本章第三节内容直接进行调试。
- 步骤 5：按照本章第四节内容编写 Lua 文件，下载调试。

二 烧写固件

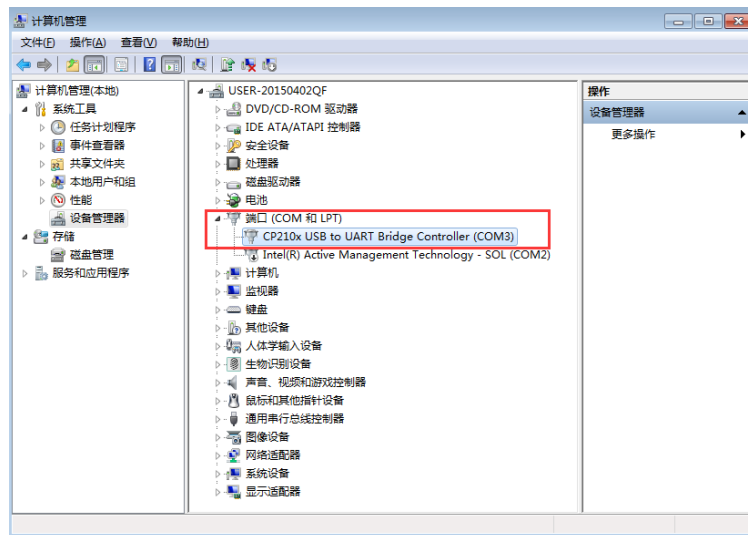
烧写固件通过 ESP8266 厂家提供的 FLASH_DOWNLOAD_TOOLS 工具下载。

步骤一：使用 MICRO USB 连接，USB 线不仅实现通信，还对 NodeMCU 供电。如果第一次连接，需要安装 USB 驱动。ESP12E Dev Kit 的 NodeMCU 使用 ttl 转 USB 芯片为 CP2102。

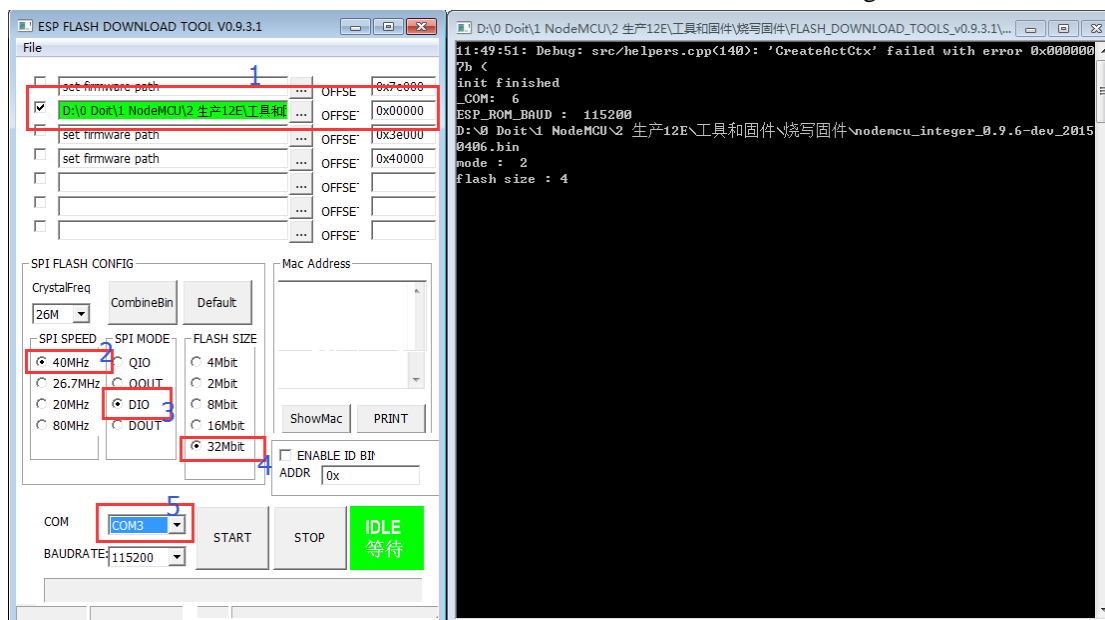
其驱动下载地址：<http://www.ddooo.com/softdown/56219.htm>。



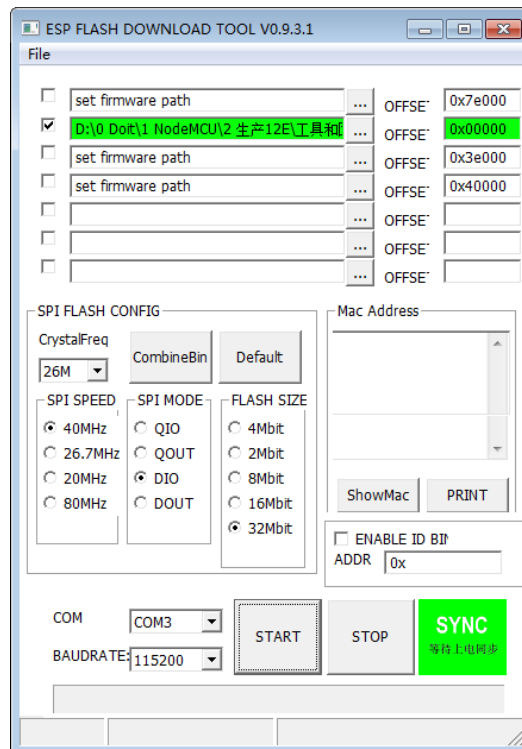
步骤二：连接 USB 后，在 Windows 的设备管理器中可以查看串口号。Win7 下，“计算机”右键->“管理”->“设备管理器”->“端口(COM 和 LPT)”。



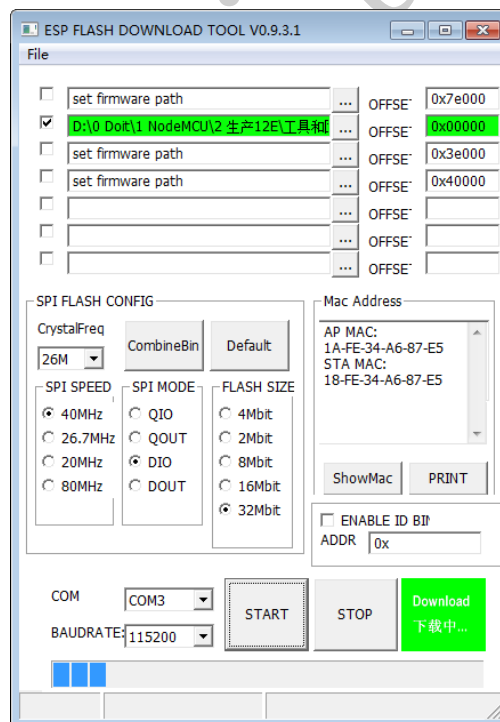
步骤三：打开 FLASH_DOWNLOAD_TOOLS 文件夹中的 frame_test.exe。按照下图的 1~5 步依次选择。其中在第一步选择 NodeMCU 的固件，通常使用 Integer 版本。



步骤四：点击烧写工具上的点 START 按钮，进入等待上电同步。



步骤五：按住模块的 FLASH 键不放，然后再按一下 RST 键，进入烧写状态后，松开后即开始下载



Tips: 下载进度完成后会显示 error 错误提示,可忽略该错误,这是官方烧写工具的一个 BUG。

下载完成后，关闭程序。即可开始 NodeMCU 开发之旅。



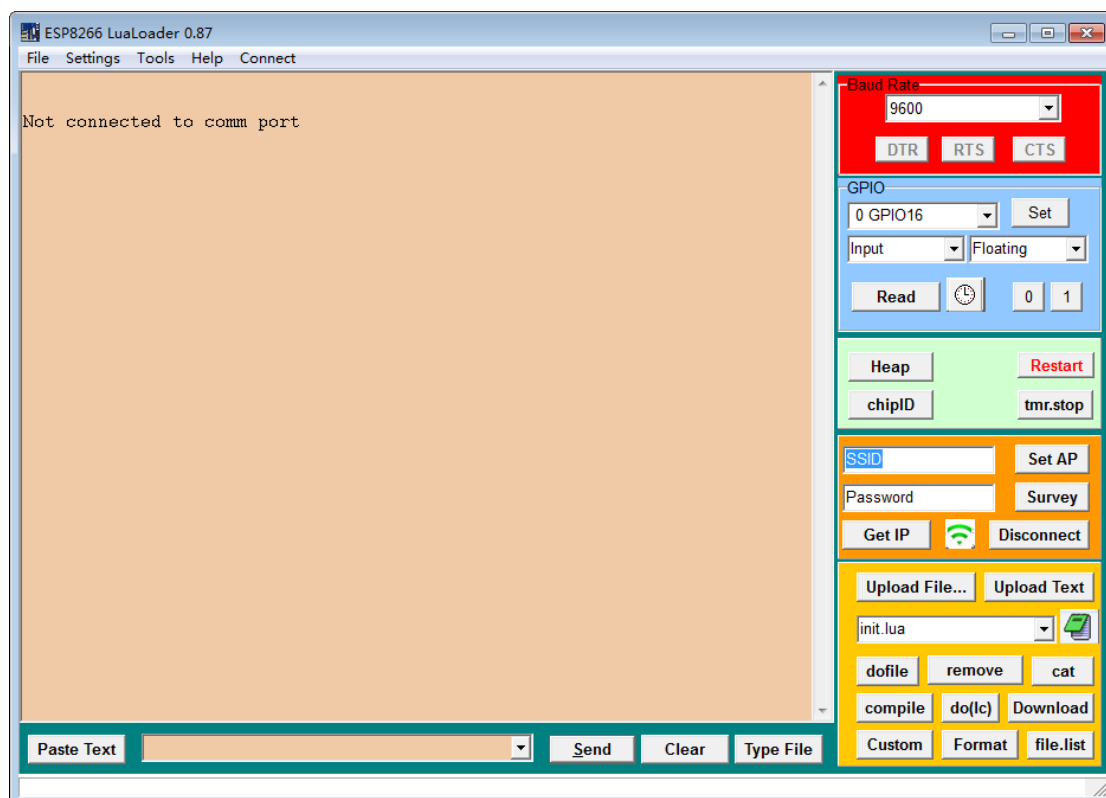
三 程序调试工具

程序调试工具提供基于串口的交互界面，包含基本的运行命令，应用程序编辑下载功能等。针对 NodeMCU 有多款开源工具可以使用，比如 LuaLoader、NodeMCUStudioIDE、Decoda 等。本教程以 LuaLoader 为例说明程序调试步骤。

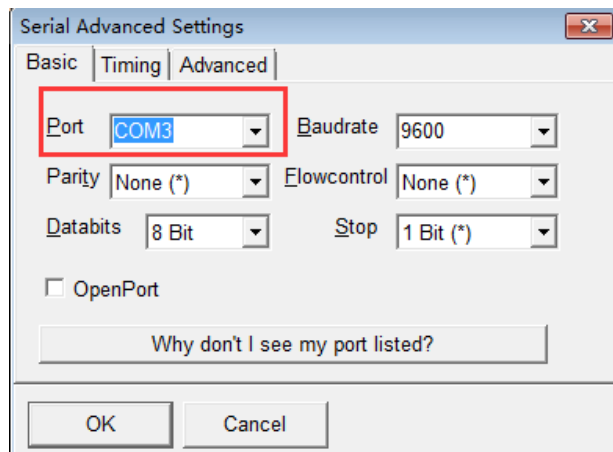
Tips: NodeMCUStudioIDE 下载地址：
<https://github.com/nodemcu/nodemcu-studio-csharp>
Decoda 下载地址：
<https://github.com/unknownworlds/decoda/releases>

3.1 “Hello World”

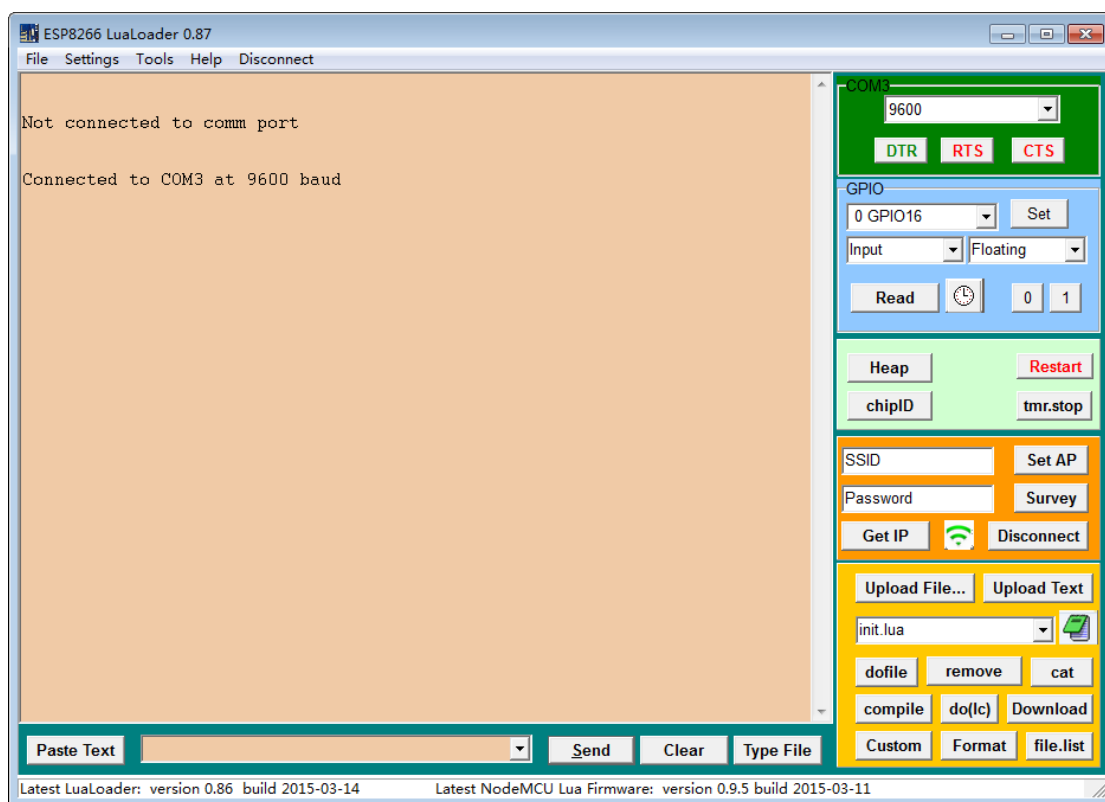
断开 NodeMCU 的 USB 连接线。运行 Lualoader，运行画面如下。



在“Setting”菜单中，选择“Comm Port Settings”，弹出“Serial Advanced Setting”。在“Port”中设置串行端口号，其他为默认即可。



插入 NodeMCU 的 USB 连接线。在主界面菜单栏选择“Connect”，完成连接。



在 Lualoader 下方有命令输入编辑框，可以通过发送命令的方式与 NodeMCU 交互。例如输入如下。



将会得到如下输出：

```
print("Hello World!")  
Hello World!  
>
```

第一个 Lua 代码执行成功，是不是很简单？

Tips:在程序调试过程中，需要查阅 NodeMCU 支持的 API 函数。查看地址：

<http://www.nodemcu.com/docs/node-module/>或

https://github.com/nodemcu/nodemcu-firmware/wiki/nodemcu_api_cn

3.2 GPIO 测试

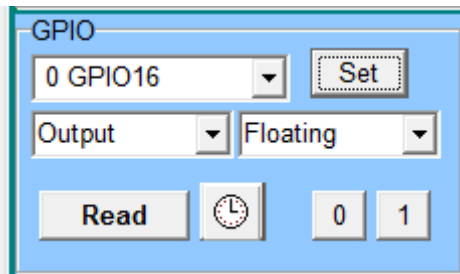
使用 LuaLoader 连接到 NodeMCU 后，可以使用 LuaLoader 自带的快捷功能实现对 NodeMCU 的操作。这些功能与直接使用 Lua 代码执行的效果等同。

根据 NodeMCU API 可以查知中 ESP8266 的 GPIO16 映射到 0 号 IO 口。即使用 Lua 对 0 号 IO 口操作，将会在 GPIO16 上实现。

使用 LuaLoader 右侧“GPIO”组的功能可以完成 GPIO 测试。

例如设置 0 号 IO 口为输出模式。只需要选择“0 GPIO16”，“Output”，然后点击“Set”即可。此时 LuaLoader 向 NodeMCU 写入“gpio.mode(0,gpio.OUTPUT)”设置完成。点击“0”或“1”即可设置该端口为低电平输出或者高电平输出。输出的代码为：“gpio.write(0,gpio.LOW)”或“gpio.write(0,gpio.HIGH)”

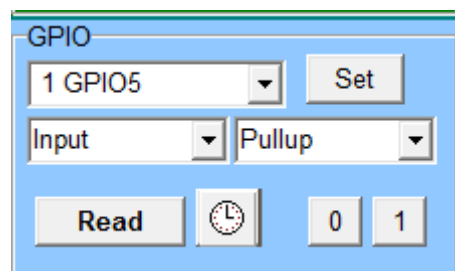
Tips: 由于 0 号端口连接到板载 LED，当将 0 号端口设置为输出模式，输出低电平时，LED 将会被点亮，否则熄灭。



运行效果：

```
gpio.mode(0,gpio.OUTPUT)
> gpio.write(0,gpio.LOW)
> gpio.write(0,gpio.HIGH)
>
```

为了测试 1 号端口的输入功能，使用一根杜邦线连接“1”号端口和 GND。然后选择“1 GPIO5”，“Input”，“Pullup”，点击“Set”，可以完成对该端口的输入设置。点击“Read”可以执行一次读操作，并返回结果“0”。当将杜邦线连接到“1”号端口和 3V3 端口时，使用“Read”将返回“1”。旁边的定时器功能可以周期性读取。



运行效果：

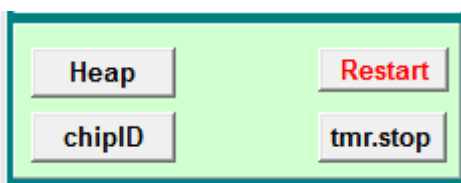
```
gpio.mode(1,gpio.INPUT,gpio.PULLUP)
> = gpio.read(1)
1
> = gpio.read(1)
0
>
```



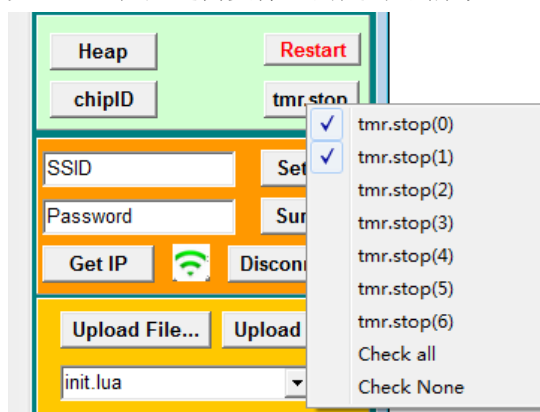

Tips: 将鼠标停留在 LuaLoader 按钮上方, 可以显示该按钮的功能说明。
Tips: 如果在 LuaLoader 主界面上显示的字符太多, 影响阅读, 可以使用软件下方的“Clear”按钮进行清屏。

3.3 系统功能

在 LuaLoader 中提供了 NodeMCU 常见的系统命令。比如: 读取剩余内存容量的 Heap 按钮。读取 ESP8266 芯片 ID 的“chipID”按钮, 软件重启按钮“Restart”以及停止指定定时器的“tmr.stop”按钮。



其中“tmr.stop”可以通过右键点选需要停止的定时器编号。



上述按钮运行效果:

```
= node.heap()
21592
> = node.chipid()
10556583
> tmr.stop(0)
> tmr.stop(1)
>
Soft Restart 2015 年 4 月 19 日 10:28:20
```

```
node.restart()
> ?G??缺)M 猴?}?鼠
```

```
NodeMCU 0.9.6 build 20150406 powered by Lua 5.1.4
lua: cannot open init.lua
>
```

NodeMCU 重启时包含有乱码，是因为 NodeMCU 启动时的波特率是 74880bps，不是 NodeMCU 默认的 9600bps。

“lua:cannot open init.lua”表示固件启动的时候没有找到 init.lua 文件。这是因为 NodeMCU 在启动时预留了 init.lua 作为应用程序入口，如果没有该文件则忽略，如果存在则开始执行该文件。利用这个特性可以在 init.lua 中写入需要执行的代码，以便上电自动运行。

3.4 WiFi 测试

NodeMCU 的 WiFi 具有 AP (Access Point)、STA (STATION)、AP+STA 三种模式。AP 模式下，NodeMCU 作为热点，发出无线信号，接受其他节点连接，相当于一台无线路由器。STA 模式下，NodeMCU 作为一个节点，可以连接到其他热点（智能手机或无线路由器上）。

Lualoader 提供了对 NodeMCU 在 STA 模式进行测试的功能。关于 AP 和 AP+STA 模式的测试请查阅 NodeMCU 文档。在下一章中将给出示例代码。



“SSID”和“Password”编辑框用于输入无线路由器的 SSID 和密码。

“Survey”是搜索 NodeMCU 所在环境的 SSID，并打印出来。等同于下列程序。

```
wifi.setmode(wifi.STATION)
wifi.sta.getap(function(t)
    if t then print("\n\nVisible Access Points:\n")
        for k,v in pairs(t) do l = string.format("%-10s",k) print(l.."  "..v) end
    else
        print("Try again")
    end
end)
```

程序的执行结果如下：

```
> wifi.setmode(wifi.STATION) wifi.sta.getap(function(t) if t then print
("\n\nVisible Access Points:\n") for k,v in pairs(t) do l =
string.format("%-10s",k) print(l.."  "..v) end else print("Try again")
end end)
>
Visible Access Points:
dahuijin    4,-78,5c:63:bf:c5:d6:66,6
CMCC-WEB    0,-90,00:23:89:96:02:00,11
MERCURY_44B6 4,-67,c0:61:18:21:44:b6,11
oplinx_b    3,-76,c0:c1:c0:a0:c9:ad,2
TP-LINK_0A002C 4,-77,c0:61:18:0a:00:2c,6
TP-LINK_222DCC 4,-90,28:2c:b2:22:2d:cc,1
ChinaNet-Dahuijin 4,-66,84:74:2a:8c:17:a3,3
ChinaNet-qCSQ 4,-90,60:bb:0c:2b:96:93,6
Wireless    0,-80,00:25:12:62:a6:3a,6
Beusoft-JE  3,-68,ec:17:2f:5f:9b:f0,1
green-id    4,-86,20:dc:e6:b5:3c:18,1
Beusoft-AE  4,-74,d8:24:bd:77:08:fb,1
ChinaNet-WESG 4,-76,b4:41:7a:b2:c9:54,1
BeUssoft    4,-81,d4:ee:07:0f:00:5c,1
HP-HOTSPOT-F9-LaserJet M1218 2,-87,54:35:30:0c:71:f9,6
CMCC        0,-70,00:23:89:22:98:b1,1
MERCURY_1013 4,-58,14:e6:e4:85:98:7a,6
Tomato      4,-83,8c:28:06:1e:01:54,13
and-Business 0,-93,00:23:89:96:02:02,11
bluebanner-1103 4,-72,14:75:90:b3:08:42,6
seeker      4,-64,ec:26:ca:6c:6b:92,11
ChinaNet-mALi 4,-70,8c:e0:81:30:c1:95,10
```

“Set AP”是使用 SSID 和 Password 的内容连接到无线路由器。



```
> wifi.setmode(wifi.STATION)
> wifi.sta.config("MERCURY_1013","123456789")
```

“Get IP”是获取 STA 模式下的 IP 地址并显示。等同于直接发送代码：`print(wifi.sta.getip())` 在主界面返回结果：

```
> = wifi.sta.getip()
192.168.1.106 255.255.255.0 192.168.1.1
```

“Disconnect”是断开与无线路由器之间的连接。

```
wifi.sta.disconnect()
```

按钮是检查当前 STA 模式的工作状态。

```
> = wifi.sta.status()
0 IDLE
```

3.5 文件操作

NodeMCU 使用文件来保存 Lua 代码，得益于 Lua 的解释性执行特性，Lua 源程序可以在线编译运行，因此文件操作非常重要。

NodeMCU 支持多种文件操作。常用的文件操作有文件上传、删除、编译为二进制文件、运行等。Lualoader 中提供了丰富的文件操作功能。



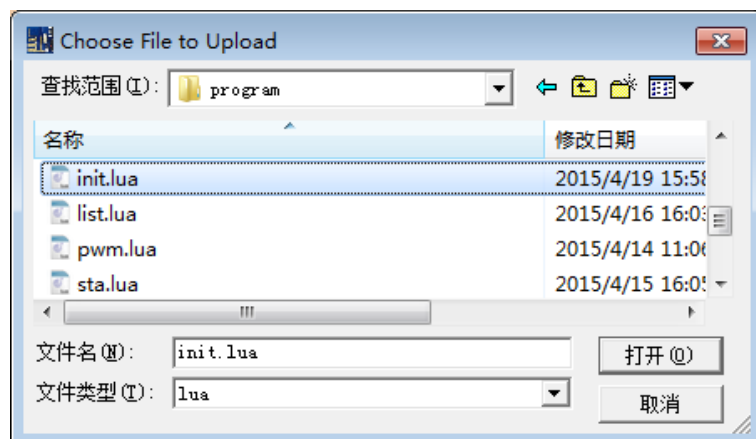
(1) 上传文件

“Upload File”：上传编写好的 lua 源代码文件到 NodeMCU 中。例如上传 init.lua 文件。

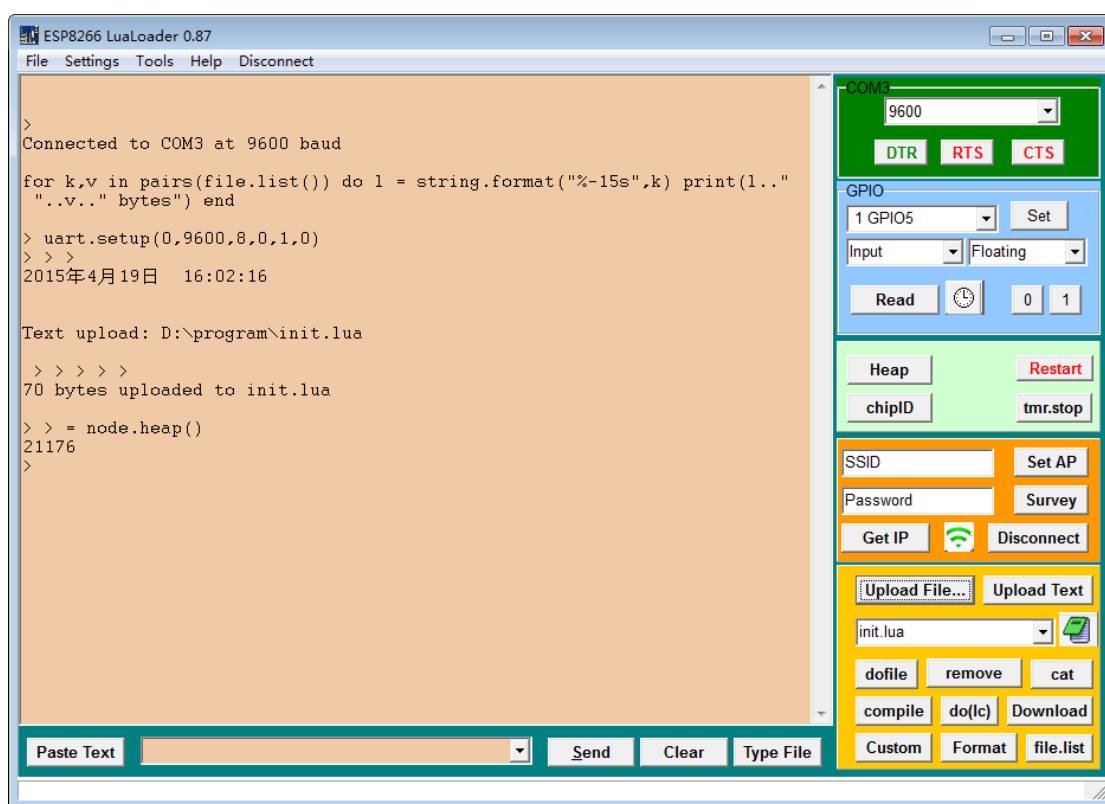
Init.lua 示例程序内容如下：


```
print("\n")
print("NodeMCU Started")
print("Type something to start")
```

点击“Upload File”：



上传完成后:



Tips: 上传完成后，在“Upload File”按钮下方的可编辑下拉列表框中将显示该文件的文件名，后续操作都是基于该选择。如果有多个文件上传，该下拉列表框将记录所有已经上传过的文件名称。旁边的“”按钮是打开 lua 文件编辑器，实现文件编辑。

(2) 查看文件

如果需要查看该文件，可以使用“cat”按钮。点击“cat”按钮后，Lualoader 下载如下程序到 NodeMCU 中。

```
print("\n\ncat init.lua\n")
file.open( "init.lua", "r")
while true
do
    line = file.readline()
    if (line == nil) then break
```



```
end
print(string.sub(line, 1, -2))
tmr.wdclr()
end
file.close()
```

其中 `tmr.wdclr()` 用于清除看门狗，避免触发看门狗。程序运行结果：

```
cat init.lua

print("\n")
print("NodeMCU Started")
print("Type something to start")
>
```

Tips: 一个长时间的循环或者事务，需内部调用 `tmr.wdclr()` 清除看门狗计数器，防止看门狗计数器溢出导致重启。

(3) 运行文件

点击按钮“dofile”，运行该文件，输出执行结果。该命令相当于执行：`dofile("init.lua")`

`dofile(init.lua)` 2015 年 4 月 19 日 16:07:24

```
dofile("init.lua")
```

```
NodeMCU Started
Type something to start
>
```

(4) 文件编译

点击“compile”按钮实现文件的编译，相当于运行：`node.compile()`。编译完成后，自动生成“文件名.lc”文件

```
> node.compile("init.lua")
>
```

如果将 `init.lua` 文件中

```
print("NodeMCU Started")
```

语句改为：

```
print("NodeMCU Started"
```

编译将出现错误，将会给出响应的提示。

```
> node.compile("init.lua")
stdin:1: init.lua:3: ')' expected (to close '(' at line 2) near 'print'
>
```

Tips: Lua 程序的二进制文件为 `.lc` 格式。将 `.lua` 格式的源程序转为二进制文件运行可以提高程序效率，降低内存使用。因此建议在程序调试完成需要运行时，使用 `.lc` 文件运行。

(5) 运行 lc 文件

点击按钮“dolc”，运行 `lc` 文件，输出执行结果。该命令相当于执行：`dofile("init.lc")`，执行效果与 `dofile("init.lc")` 相同。

(6) 列出文件列表



点击按钮“file.list”，该操作相当于执行：

```
for k,v in pairs(file.list()) do l = string.format("%-15s",k) print(l.."    "..v.." bytes") end
```

执行结果如下：

```
init.lua          70 bytes
init.lc           160 bytes
>
```

(7) 删除文件

点击按钮“remove”，该操作相当于执行：file.remove("文件名.lua")。

更多关于 LuaLoader 的功能，请参考：

<http://benlo.com/esp8266/index.html#LuaLoader>。

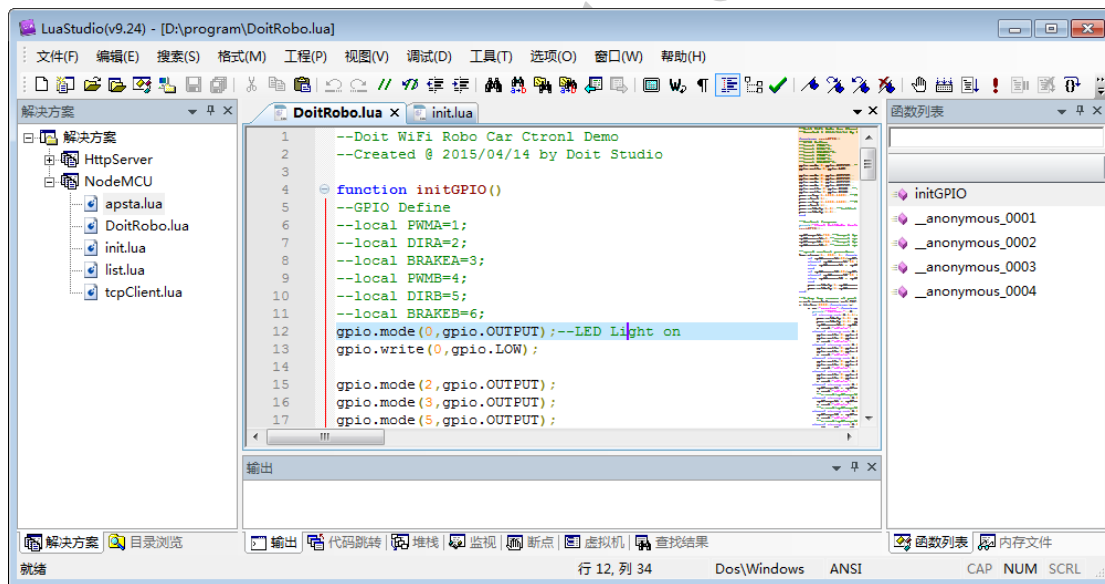
(8) 格式化文件系统

点击按钮“Format”按钮，格式化文件系统，该操作相当于执行：file.format("文件名.lua")。

```
file.format()
format done.
>
```

3.6 程序编辑

Lua 程序编写可以直接使用记事本，也可以使用专用调试编辑 IDE。本教程推荐使用 LuaStudio。软件如下图所示。



在 LuaStudio 中源文件的管理采用工程模式。在菜单“工程”中可以新建工程或者添加现有工程。

右侧“解决方案”选项卡里面可以查看现有工程。在工程名称上使用右键可以对源文件进行管理。例如新建、添加现有文件、删除、重命名等。

在编辑 lua 文件时，LuaStudio 可以高亮关键词、采用不同颜色显示数据类型、进行输入提示等，非常方便上手。

编辑完成的 lua 文件通过 LuaLoader 下载到 NodeMCU 中运行。

Tips: 如果编写的 Lua 程序有 BUG，下载到 NodeMCU 之后运行出现问题导致程序不能通过串口下载或者不能正常执行串口命令(例如 file.formart()语句已经无法下载到 NodeMCU)。



此时需要重新烧写固件。

四 实例代码

4.1 init.lua 文件

NodeMCU 在启动时预留了 `init.lua` 作为应用程序入口，如果没有该文件则忽略，如果存在则开始执行该文件。利用这个特性可以在 `init.lua` 中写入需要执行的代码，以便上电自动运行。

资料包中文件名为“`init.lua`”。

```
1  print("\n")
2  print("ESP8266 Started")
3
4  local luaFile = {"fileName.lua"}
5  for i, f in ipairs(luaFile) do
6      if file.open(f) then
7          file.close()
8          print("Compile File:"..f)
9          node.compile(f)
10         print("Remove File:"..f)
11         file.remove(f)
12     end
13 end
14
15 luaFile = nil
16 collectgarbage()
17
18 dofile("fileName.lua");
```

程序第 1、2 行打印字符信息。

第 4 行定义需要编译的 lua 文件名。其中“`fileName`”需要根据不同的文件进行更改。如果要编译多个文件，在后面继续添加即可。例如：

```
local luaFile = {"fileName1.lua","fileName2.lua"}
```

第 5 行是使用 `for` 循环完成多个文件的操作。

第 6 行判断文件是否存在，如果存在则执行编译。如果不存在则忽略。

第 7 行是关闭已经打开的文件。

第 8~11 行完成编译，自动生成“`filename.lua`”文件。

第 15~16 行是回收内存。

第 17 行是执行刚刚编译完成的二进制文件。

Tips: 在 Lua 程序中，默认变量为全局类型，如果要限定变量只能在本文件中使用，需要在变量前加入关键词“`local`”。在使用完后，需要将其赋值为 `nil`，然后调用 `collectgarbage()` 显示的回收内存。

下载程序，重启 NodeMCU 运行时，如果发现编译出现内存不足的警告。如下所示：



lua: init.lua:8: not enough memory

这是因为 Lua 程序在刚刚启动的时候，内存消耗较大，在启动完成后回到正常。因此可以在启动后，稍微延迟一定时间再进行编译。

4.2 WiFi 模式

如前所述，NodeMCU 的 WiFi 具有 AP（Access Point）、STA（STATION）、AP+STA 三种模式。下面的代码演示了如何通过 Lua 代码设置这三种模式。

4.2.1 AP 模式

资料包中文件名为“ap.lua”。

将“init.lua”文件中“fileName.lua”修改为“ap.lua”，“fileName.lc”修改为“ap.lc”。修改完成后下载。

程序代码：

```
1  print("Ready to start soft ap")
2
3  local str=wifi.ap.getmac();
4  local
    ssidTemp=string.format("%s%s%s",string.sub(str,10,11),string.sub(str,13,14),string.sub(s
tr,16,17));
5
6  cfg={ }
7  cfg.ssid="ESP8266_"..ssidTemp;
8  cfg.pwd="12345678"
9  wifi.ap.config(cfg)
10
11  cfg={ }
12  cfg.ip="192.168.1.1";
13  cfg.netmask="255.255.255.0";
14  cfg.gateway="192.168.1.1";
15  wifi.ap.setip(cfg);
16  wifi.setmode(wifi.SOFTAP)
17
18  str=nil;
19  ssidTemp=nil;
20  collectgarbage();
21
22  print("Soft AP started")
23  print("Heap:(bytes)"..node.heap());
24  print("MAC:"..wifi.ap.getmac().."\r\nIP:"..wifi.ap.getip());
```

程序第 1 行打印字符信息。

第 3、4 行是获取 AP 模式下的 MAC 地址，以 MAC 地址后 6 位为 AP 的 SSID，当然你



也可以使用其他作为 ID，比如通过 node.chipid() 得到 ESP8266 的芯片 ID。

第 6~9 行设置 AP 模式下的 SSID。SSID 格式为“ESP8266_XXXXXX”，其中 XXXXXX 为 MAC 地址后 6 位。

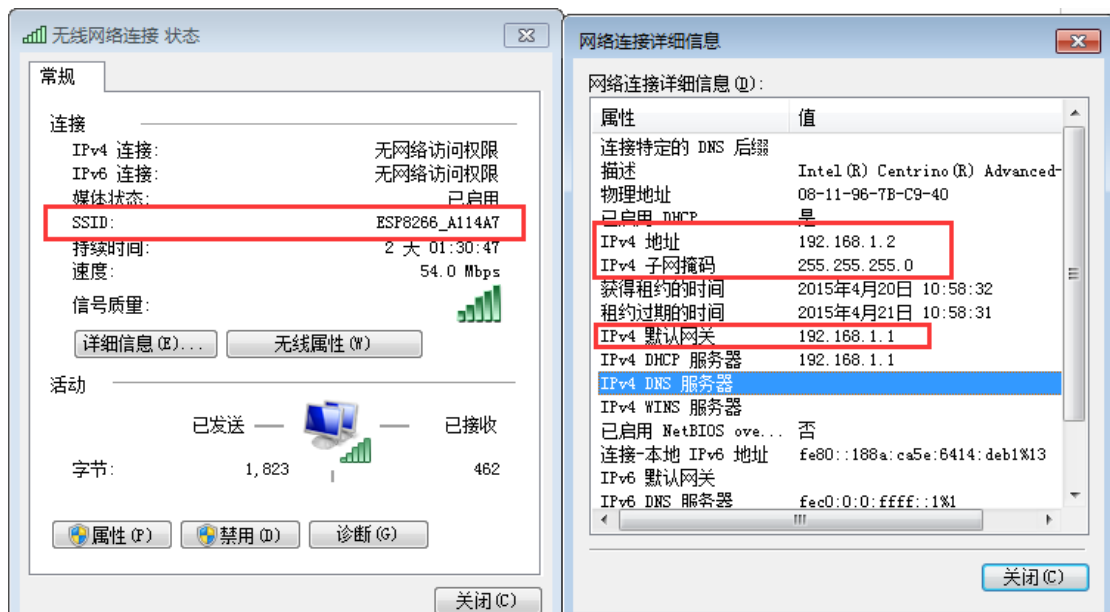
第 11~15 行为设置模块的 IP 地址、子网掩码以及网关地址。

第 16 行调用 wifi.setmode() 函数设置执行。

第 23 行打印当前内存。

第 24 行打印 mac 地址和 ip 地址。

下载该程序，执行结果是使用无线网络设备可以搜索到 NodeMCU 发出来的 AP 信号。通过电脑连接到该 SSID，见下图。



执行程序的 Log 如下：

1 NodeMCU 0.9.6 build 20150406 powered by Lua 5.1.4



```
2
3
4   ESP8266 Started
5   Compile File:ap.lua
6   Remove File:ap.lua
7   Ready to start soft ap
8   Soft AP started
9   Heap:(bytes)15328
10  MAC:1A-FE-34-A1-14-A7
11  IP:192.168.1.1
12  >
```

从上面可以看到 ap.lua 在被编译后删除了。

如果重新启动 NodeMCU，程序 Log 如下：

```
1   NodeMCU 0.9.6 build 20150406   powered by Lua 5.1.4
2
3
4   ESP8266 Started
5   Ready to start soft ap
6   Soft AP started
7   Heap:(bytes)15048
8   MAC:1A-FE-34-A1-14-A7
9   IP:192.168.1.1
10  >
```

关于更多函数的功能请查阅：

https://github.com/nodemcu/nodemcu-firmware/wiki/nodemcu_api_cn。

4.2.2 STA 模式

资料包中文件名为 “sta.lua”。

将“init.lua”文件中“fileName.lua”修改为“sta.lua”，“fileName.lc”修改为“sta.lc”。修改完成后下载。

程序代码：

```
1   print("Ready to Set up wifi mode")
2   wifi.setmode(wifi.STATION)
3
4   wifi.sta.config("MERCURY_1013","123456789")
5   wifi.sta.connect()
6   local cnt = 0
7   tmr.alarm(3, 1000, 1, function()
8       if (wifi.sta.getip() == nil) and (cnt < 20) then
9           print("Trying Connect to Router, Waiting...")
10          cnt = cnt + 1
11      else
```



```
12  tmr.stop(3)
13  if (cnt < 20) then print("Config done, IP is "..wifi.sta.getip())
14  else print("Wifi setup time more than 20s, Please verify wifi.sta.config() function. Then
    re-download the file.")
15  end
16      cnt = nil;
17      collectgarbage();
18  end
19  end)
```

程序第 1 行打印字符信息。

第 2 行设置 WiFi 为 STA 模式。

第 4 行设置 STA 模式下，将要连接的无线路由器 SSID 和密码。

第 5 行是发起连接。

第 6~19 行是启动 3 号定时器，每隔 1000 毫秒检查一次连接，如果超过 20 秒钟没有连接上无线路由器，则提示连接失败。如果在 20 秒内连接成功，则停止 3 号定时器，打印 IP 地址。

执行程序的 Log 如下：

```
1  NodeMCU 0.9.6 build 20150406  powered by Lua 5.1.4
2
3
4  ESP8266 Started
5  Compile File:sta.lua
6  Remove File:sta.lua
7  Ready to Set up wifi mode
8  > Trying Connect to Router, Waiting...
9  Trying Connect to Router, Waiting...
10 Config done, IP is 192.168.1.100
```

可以看到 NodeMCU 获取到 IP 地址为“192.168.1.100”。

4.2.3 AP+STA

资料包中文件名为“apsta.lua”。

将“init.lua”文件中“fileName.lua”修改为“apsta.lua”，“fileName.lc”修改为“apsta.lc”。修改完成后下载。

程序代码：

```
1  print("Ready to start soft ap AND station")
2  local str=wifi.ap.getmac();
3  local
    ssidTemp=string.format("%s%s%s",string.sub(str,10,11),string.sub(str,13,14),string.sub(s
tr,16,17));
4  wifi.setmode(wifi.STATIONAP)
5
6  local cfg={}
7  cfg.ssid="ESP8266_"..ssidTemp;
```



```
8  cfg.pwd="12345678"
9  wifi.ap.config(cfg)
10 cfg={}
11 cfg.ip="192.168.2.1";
12 cfg.netmask="255.255.255.0";
13 cfg.gateway="192.168.2.1";
14 wifi.ap.setip(cfg);
15
16 wifi.sta.config("MERCURY_1013","123456789")
17 wifi.sta.connect()
18
19 local cnt = 0
20 gpio.mode(0,gpio.OUTPUT);
21 tmr.alarm(0, 1000, 1, function()
22   if (wifi.sta.getip() == nil) and (cnt < 20) then
23     print("Trying Connect to Router, Waiting...")
24     cnt = cnt + 1
25     if cnt%2==1 then gpio.write(0,gpio.LOW);
26     else gpio.write(0,gpio.HIGH); end
27   else
28     tmr.stop(0);
29     print("Soft AP started")
30     print("Heap:(bytes)"..node.heap());
31     print("MAC:"..wifi.ap.getmac().."\\r\\nIP:"..wifi.ap.getip());
32     if (cnt < 20) then print("Conected to
Router\\r\\nMAC:"..wifi.sta.getmac().."\\r\\nIP:"..wifi.sta.getip())
33       else print("Conected to Router Timeout")
34     End
35     gpio.write(0,gpio.LOW);
36     cnt = nil;cfg=nil;str=nil;ssidTemp=nil;
37     collectgarbage()
38   end
39 end)
```

程序第 1 行打印字符信息。

第 2~14 行设置 AP；第 16~17 行设置 STA 参数。

第 19~39 行是设置定时器 0，每隔 1 秒钟检查一次模块是否连接到无线路由器。

第 20 行设置 D0 即 GPIO16 为输出模式，改端口连接到 NodeMCU 的 LED 灯，通过第 25~26 行代码指示模块 WiFi 连接情况。

执行程序的 Log 如下：

```
1  NodeMCU 0.9.6 build 20150406  powered by Lua 5.1.4
2
3
4  ESP8266 Started
5  Compile File:apsta.lua
```



```
6 Remove File:apsta.lua
7 Ready to start soft ap AND station
8 > Trying Connect to Router, Waiting...
9 Trying Connect to Router, Waiting...
10 Trying Connect to Router, Waiting...
11 Soft AP started
12 Heap:(bytes)16192
13 MAC:1A-FE-34-A1-14-A7
14 IP:192.168.2.1
15 Conected to Router
16 MAC:18-FE-34-A1-14-A7
17 IP:192.168.1.100
```

4.3 PWM 功能

NodeMCU 的 D1~12 管脚均具有 PWM 功能（除了 D0）。

PWM 频率可以设置范围 1~1000Hz，占空比设置范围为 0~1023（对应 0%~100%）

资料包中文件名为“pwm.lua”。

将“init.lua”文件中“fileName.lua”修改为“pwm.lua”，“fileName.lua”修改为“pwm.lua”。修改完成后下载。

PWM 功能程序代码：

```
1 print("PWM Function test")
2 pwm.setup(1,1000,1023);
3 pwm.start(1);
4
5 local r=512;
6 local flag=1;
7 tmr.alarm(2,100,1,function()
8   pwm.setduty(1,r);
9   if flag==1 then
10    r=r-50;      if r<0 then flag=0 r=0 end
11   else
12    r= r+50; if r>1023 then flag=1 r=1023 end
13   end
14 end)
```

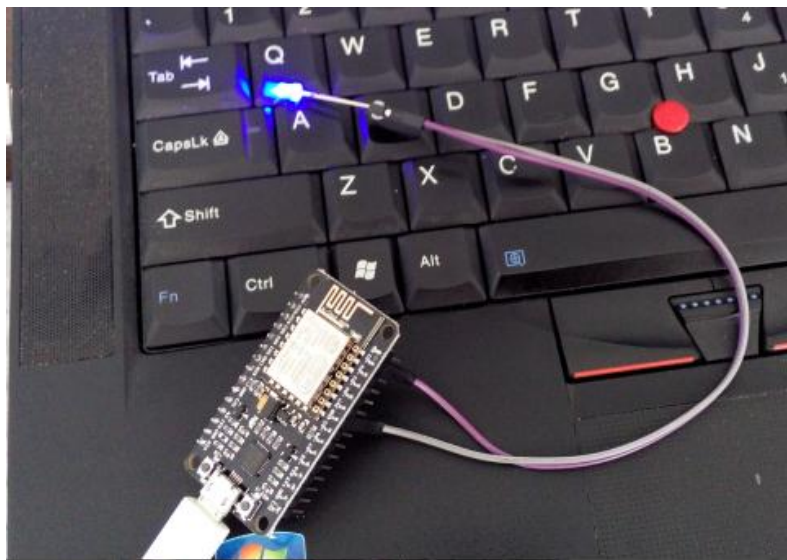
程序第 1 行打印字符信息。

第 2 行设置端口 1 的频率为 1000Hz，占空比为 1023。

第 3 行启动 PWM 输出。

第 5~14 行设置了定时器 2，每隔 100 毫秒，对占空比 r 进行增加或者减小循环操作。通过变量 flag 进行记录。

下载程序，通过万用表或者示波器可以看到 D1 端口的电平输出变化。如果在 D1 端口和 GND 之间接入一个 LED 二极管。可以查看到二极管的亮度循环变化。



Tips: LED 的长管脚为阳极，接 D1；短管脚为阴极，接 GND。

执行程序的 Log 如下：

```
1 NodeMCU 0.9.6 build 20150406 powered by Lua 5.1.4
2
3
4 ESP8266 Started
5 Compile File:pwm.lua
6 Remove File:pwm.lua
7 PWM Function test
8 >
```

五 资料汇总

- 1) 乐鑫 ESP8266 官方网站
<http://espressif.com/zh-hans/>
芯片手册地址：
- 2) Nodemcu:
<https://github.com/nodemcu>
- 3) NodeMCU 的 API:
<https://github.com/nodemcu/nodemcu-firmware>
- 4) Lua:
<http://www.lua.org/>
<http://baike.baidu.com/view/416116.htm>
<http://zh.wikipedia.org/wiki/Lua>
- 5) eLua:
<http://www.eluaproject.net/>
- 6) LuaLoader
<http://benlo.com/esp8266/index.html#LuaLoader>
<https://github.com/GeoNomad/LuaLoader>