NodeMCU-API 中文说明

Version 0.9.5 build 2015-02-13

flash 错误

注意：有些模块在烧写之后启动，串口输出 ERROR in flash_read: r=。。。

这是因为模块原来的 flash 内部没有擦除。

可使用 blank512k.bin，

内容为全 0xFF，从 0x00000 开始烧入。
烧入之后可以正常运行。

概述

- 快速、自动连接无线路由器
- 基于 Lua 5.1.4，使用者需了解最简单的 Lua 语法：采用事件驱动的编程模型
- 内置 file, timer, pwm, i2c, net, gpio, wifi, uart, adc 模块
- 串口波特率:9600-8N1
- 对模块的引脚进行编号；gpio，i2c，pwm 等模块需要使用引脚编号进行索引

目前的编号对应表格:新 Gpio 索引 (20141219)

IO index	ESP8266 pin	IO index	ESP8266 pin	IO index	ESP8266 pin
0 [*]	GPIO16	5	GPIO14	9	GPIO3

1	GPIO5	6	GPIO12	10	GPIO1
2	GPIO4	7	GPIO13	11	GPIO9
3	GPIO0	8	GPIO15	12	GPIO10
4	GPIO2				

[*] D0(GPIO16) 只能用作 **gpio** 读写，不支持中断，**i2c/pwm/ow**

node 模块

node.restart()

描述：重新启动

语法：node.restart()

参数：nil

返回值：nil

示例：

```
node.restart();
```

参见：-

node.dsleep()

描述：进入睡眠模式，计时时间之后唤醒

语法：node.dsleep(us)

-注意: 如需使用此功能，需要将 esp8266 的 PIN32(RST)和 PIN8(XPD_DCDC)短接。

参数：us: 睡眠时间，单位：us

返回值：nil

示例：

```
node.dsleep(us);
```

参见：-

node.info()

描述：返回 NodeMCU 版本信息，包括：chipid, flashid, flash size, flash mode, flash speed.

语法：node.info()

参数：`nil`

返回值：`majorVer` (number)

`minorVer` (number)

`devVer` (number)

`chipid` (number)

`flashid` (number)

`flashsize` (number)

`flashmode` (number)

`flashspeed` (number)

示例：

```
majorVer, minorVer, devVer, chipid, flashid, flashsize, flashmode, flashspeed = node.info();
   print("NodeMCU "..majorVer.."."..minorVer.."."..devVer)
```

参见：-

node.chipid()

描述：返回芯片 ID

语法：node.chipid()

参数：nil

返回值：number:芯片 ID

```
示例：    id = node.chipid();
```

参见：-

node.flashid()

描述：返回 flashid

语法：node.flashid()

参数：nil

返回值：number:flash ID

```
示例：    flashid = node.flashid();
```

参见：-

node.heap()

描述：返回当前系统剩余内存大小，单位：字节

语法：node.heap()

参数：nil

返回值：number: 系统剩余内存字节数

示例：
```
heap_size = node.heap();
```

参见：-

node.key()

描述：定义按键的功能函数，按键与 GPIO16 相连。

语法：node.key(type, function())

参数：type: type 取字符串"long"或者"short". long:按下按键持续 3s 以上, short: 短按按键(时间短于 3s)

function(): 用户自定义的按键回调函数。 如果为 nil，则取消用户定义的回调函数。

默认函数：long：改变 LED 闪烁频率，short：重新启动。

返回值：nil

示例：
```
node.key("long", function() print('hello world') end)
```

参见：-

node.led()

描述：设置 LED 的亮/暗时间, LED 连接到 GPIO16，与 node.key()复用。

语法：node.led(low, high)

参数：Low: LED 关闭时间，如设置为 0，则 LED 处于常亮状态。单位：毫秒，时间分辨率：80~100ms

High: LED 打开时间，单位：毫秒，时间分辨率：80~100ms

返回值：nil

示例：
```
-- LED 常亮.
node.led(0);
```

参见：-

node.input()

描述：接收字符串并将字符串传入 lua 解释器。

功能同 pcall(loadstring(str))，增加了支持多行输入的功能。

语法：node.input(str)

参数：str: Lua 代码段

返回值：nil

示例：
```
-- 注意：该函数不支持在命令行中使用。
sk:on("receive", function(conn, payload) node.input(payload) end)
```

参见：-

node.output()

描述：将 lua 解释器输出重定向于回调函数。

语法：node.output(function(str), serial_debug)

参数：function(str): 接收 lua 解释器输出的 str 作为输入，可以将该输出通过 socket 发送。

serial_debug: 1：将输出送至串口； 0：输出不送至串口

返回值： nil

示例：
```
function tonet(str)
  sk:send(str)
  -- print(str) 错误!!! 千万不要在此函数中再使用 print 函数
  -- 因为这样会导致函数的嵌套调用！！
end
node.ouput(tonet, 1)  -- serial also get the lua output.
```

参见： -

node.readvdd33()

描述： 读取 vdd33 管脚电压.

语法： node.readvdd33()

参数： nil

返回值： 电压数值，单位：毫伏.

示例：
```
print(node.readvdd33())
```

output

3345

```
v = node.readvdd33() / 1000
print(v)
v=nil
```

output

3.315

参见： -

node.compile()

描述：将.lua 文本文件编译为字节码文件，并保存为.lc 文件.

语法：node.compile("file.lua")

参数：字符串：lua 文件名.

返回值：nil

示例：
```
file.open("hello.lua","w+")
file.writeline([[print("hello nodemcu")]])
file.writeline([[print(node.heap())]])
file.close()

node.compile("hello.lua")
dofile("hello.lua")
dofile("hello.lc")
```

参见：-

file 模块

file.remove()

描述：删除文件。

语法：file.remove(filename)

参数：filename: 需要删除的文件。

返回值：nil

示例：
```
-- 删除 foo.lua 文件
file.remove("foo.lua")
```

参见： - file.open()

- file.close()

file.open()

描述： 打开文件。

语法： file.open(filename, mode)

参数： filename: 需要打开的文件，不支持文件夹。

mode:

"r": read mode (the default)

"w": write mode

"a": append mode

"r+": update mode, 文件内的数据保留

"w+": update mode, 文件内的数据清除

"a+": append update mode, 文件内的数据保留，要写入的数据仅能增加在文件最后。

返回值： nil: 文件打开失败，不存在 true: 文件打开成功

示例：
```
-- 打开'init.lua'，并打印文件的第一行。
file.open("init.lua", "r")
print(file.readline())
file.close()
```

参见： - file.close()

- file.readline()

file.close()

描述：关闭文件。

语法：file.close()

参数：nil

返回值：nil

示例：
```
-- 打开'init.lua'，并打印文件的第一行，然后关闭文件。
file.open("init.lua", "r")
print(file.readline())
file.close()
```

参见：- file.open()

- file.readline()

file.readline()

描述：读取文件的一行。

语法：file.readline()

参数：nil

返回值：逐行返回文件内容。返回值：末尾包含 EOL('\n')

如果读到 EOF 返回 nil。

示例：
```
-- 打开'init.lua'，读取并打印文件的第一行，然后关闭文件。
file.open("init.lua", "r")
print(file.readline())
file.close()
```

参见：- file.open()

- file.close()

file.writeline()

描述：向文件写入一行，行末尾增加'\n'。

语法：file.writeline(string)

参数：string: 需要写入的字符串

返回值：true: 写入成功

nil: 写入失败

示例：

```
    -- 以'a+'的模式打开'init.lua'
    file.open("init.lua", "a+")
    -- 将'foo bar'写到文件的末尾
    file.writeline('foo bar')
    file.close()
```

参见：- file.open()

- file.write()

file.write()

描述：向文件写入字符串。

语法：file.write(string)

参数：string: 需要写入的字符串

返回值：true: 写入成功

nil: 写入失败

示例：

```
    -- 以'a+'的模式打开'init.lua'
    file.open("init.lua", "a+")
    -- 将'foo bar'写到文件的末尾
```

```
    file.writeline('foo bar')
    file.close()
```

参见： - file.open()

- file.writeline()

file.flush()

描述：清空缓存写入文件。

语法：file.flush()

参数：nil

返回值：nil

示例：
```
    -- 以'a+'的模式打开'init.lua'
    file.open("init.lua", "a+")
    -- 将'foo bar'写到文件的末尾
    file.write('foo bar')
    file.flush()
    file.close()
```

参见： - file.open()

- file.writeline()

file.seek()

描述：设置或者读取文件的读写位置，位置等于 whence 加上 offset 的值。

语法：file.seek(whence, offset)

参数：whence:

"set": base is position 0 (beginning of the file);

"cur": base is current position;(default value)

"end": base is end of file;

offset: default 0

返回值：成功: 返回当前的文件读写位置

失败: 返回 nil

示例：

```
-- 以'a+'的模式打开'init.lua'
file.open("init.lua", "a+")
-- 将'foo bar'写到文件的末尾
file.write('foo bar')
file.flush()
--将文件读写位置设置在文件开始
file.seek("set")
--读取并打印文件的第一行
print(file.readline())
file.close()
```

参见：- file.open()

- file.writeline()

file.list()

描述：显示所有文件。

语法：file.list()

参数：nil

返回值：返回包含{文件名：文件大小}的 lua table

示例：

```
l = file.list();
for k,v in pairs(l) do
  print("name:"..k..", size:"..v)
end
```

参见：- file.remove()

wifi 模块

常量： wifi.STATION, wifi.SOFTAP, wifi.STATIONAP

wifi.setmode(mode)

描述：设置 wifi 的工作模式。

语法：wifi.setmode(mode)

参数：mode: 取值为：wifi.STATION, wifi.SOFTAP or wifi.STATIONAP

返回值：返回设置之后的 mode 值

示例：
```
wifi.setmode(wifi.STATION)
```
参见：- wifi.getmode()

wifi.getmode(mode)

描述：获取 wifi 的工作模式。

语法：wifi.getmode()

参数：nil

返回值：返回 wifi 的工作模式

示例：
```
print(wifi.getmode())
```

参见：- wifi.setmode()

wifi.startsmart()

描述：开始自动配置，如果配置成功自动设置 ssid 和密码。

语法：wifi.startsmart(channel, function succeed_callback())

参数：channel: 1~13, 启动寻找的初始频段，如果为 nil 默认值为 6 频段。每个频段搜寻 20s。

succeed_callback: 配置成功的回调函数，配置成功并连接至 AP 后调用此函数。

返回值：nil

示例：

```
wifi.startsmart(6, function() end)
```

参见：- wifi.stopsmart()

wifi.stopsmart()

描述：停止配置。

语法：wifi.stopsmart()

参数：nil

返回值：nil

示例：

```
wifi.stopsmart()
```

参见：- wifi.startsmart()

wifi.sta 子模块

wifi.sta.config()

描述：设置 station 模式下的 ssid 和 password。

语法：wifi.sta.config(ssid, password)

参数：ssid: 字符串，长度小于 32 字节。

password: 字符串，长度小于 64 字节。

返回值：nil

示例：

```
wifi.sta.config("myssid","mypassword")
```

参见：- wifi.sta.connect()

- wifi.sta.disconnect()

wifi.sta.connect()

描述：station 模式下连接 AP。

语法：wifi.sta.connect()

参数：nil

返回值：nil

示例：

```
wifi.sta.connect()
```

参见：- wifi.sta.disconnect()

- wifi.sta.config()

wifi.sta.disconnect()

描述：station 模式下与 AP 断开连接。

语法：wifi.sta.disconnect()

参数：nil

返回值：nil

示例：
```
wifi.sta.disconnect()
```
参见：- wifi.sta.config()

- wifi.sta.connect()

wifi.sta.autoconnect()

描述：station 模式下自动连接。

语法：wifi.sta.autoconnect(auto)

参数：auto: 0：取消自动连接，1：使能自动连接。

返回值：nil

示例：
```
wifi.sta.autoconnect()
```

参见： - wifi.sta.config()

- wifi.sta.connect()

- wifi.sta.disconnect()

wifi.sta.getip()

描述： station 模式下获取 ip

语法： wifi.sta.getip()

参数： nil

返回值： ip 地址字符串，如:"192.168.0.111"

若 ip 地址为 0.0.0.0，则返回 nil

示例：

```
-- print current ip
print(wifi.sta.getip())
```

参见： - wifi.sta.getmac()

wifi.sta.getmac()

描述： station 模式下获取 mac 地址。

语法： wifi.sta.getmac()

参数： nil

返回值： mac 地址字符串，如:"18-33-44-FE-55-BB"

示例：

```
-- 打印当前的 mac 地址
print(wifi.sta.getmac())
```

参见：- wifi.sta.getip()

wifi.sta.getap()

描述：扫描并列出 ap，结果以一个 lua table 为参数：传递给回调函数。

语法：wifi.sta.getap(function(table))

参数：function(table): 当扫描结束时，调用此回调函数

扫描结果是一个 lua table，key 为 ap 的 ssid，value 为其他信息，格式：

authmode,rssi,bssid,channel

返回值：nil

示例：

```
-- print ap list
function listap(t)
  for k,v in pairs(t) do
    print(k.." : "..v)
  end
end
wifi.sta.getap(listap)
```

参见：- wifi.sta.getip()

wifi.sta.status()

描述：station 模式下获取当前连接状态。

语法：wifi.sta.status()

参数：nil

返回值：number： 0~5 0: STATION_IDLE, 1: STATION_CONNECTING, 2: STATION_WRONG_PASSWORD, 3: STATION_NO_AP_FOUND, 4: STATION_CONNECT_FAIL, 5: STATION_GOT_IP.

参见：-

wifi.ap 子模块

wifi.ap.config()

描述：设置 ap 模式下的 ssid 和 password

语法：wifi.ap.config(cfg)

参数：cfg: 设置 AP 的 lua table

示例：:

```
    cfg={}
    cfg.ssid="myssid"
    cfg.pwd="mypwd"
    wifi.ap.config(cfg)
```

返回值：nil

示例： `wifi.ap.config(ssid, 'password')`

参见：-

wifi.ap.getip()

描述：ap 模式下获取 ip

语法：wifi.ap.getip()

参数：nil

返回值：ip 地址字符串，如:"192.168.0.111"

若 ip 地址为 0.0.0.0，则返回 nil

示例：
```
wifi.ap.getip()
```
参见：- wifi.ap.getmac()

wifi.ap.getmac()

描述：ap 模式下获取 mac 地址。

语法：wifi.ap.getmac()

参数：nil

返回值：mac 地址字符串，如:"1A-33-44-FE-55-BB"

示例：
```
wifi.ap.getmac()
```
参见：- wifi.ap.getip()

timer 模块

tmr.delay()

描述：延迟 us 微秒。

语法：tmr.delay(us)

参数：us: 延迟时间，单位：微秒

返回值：nil

示例：
```
-- delay 100us
tmr.delay(100)
```

参见：- tmr.now()

tmr.now()

描述：返回系统计数器的当前值，uint31，单位：us。

语法：tmr.now()

参数：nil

返回值：uint31: value of counter

示例：
```
-- 打印计数器的当前值。
print(tmr.now())
```

参见：- tmr.delay()

tmr.alarm()

描述：闹钟函数。

语法：tmr.alarm(id, interval, repeat, function do())

参数：id: 定时器的 id，0~6. Interval: 定时时间，单位：毫秒。

repeat: 0：一次性闹钟；1：重复闹钟。

function do(): 定时器到时回调函数。

返回值：nil

```
示例：    -- 每 1000ms 输出一个 hello world
    tmr.alarm(0, 1000, 1, function() print("hello world") end )
```

参见：- tmr.now()

tmr.stop()

描述：停止闹钟功能。

语法：tmr.stop(id)

参数：id: 定时器的 id，0~6.

返回值：nil

```
示例：    -- 每隔 1000ms 打印 hello world
    tmr.alarm(1, 1000, 1, function() print("hello world") end )

    -- 其它代码

    -- 停止闹钟
    tmr.stop(1)
```

参见：- tmr.now()

tmr.wdclr()

描述：清除看门狗计数器。

语法：tmr.wdclr()

参数：nil.

返回值：nil

示例：

```
for i=1,10000 do
    print(i)
    tmr.wdclr()   -- 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，防止重启。
  end
```

参见： - tmr.delay()

GPIO 模块

常量： gpio.OUTPUT, gpio.INPUT, gpio.INT, gpio.HIGH, gpio.LOW

gpio.mode()

描述：将 pin 初始化为 GPIO 并设置输入输出模式，内部上拉方式。

语法：gpio.mode(pin, mode, pullup)

参数：pin: 0~12, IO 编号

mode: 取值为：gpio.OUTPUT or gpio.INPUT, or gpio.INT(中断模式) pullup: 取值为：gpio.PULLUP or gpio.FLOAT，默认为 gpio.FLOAT

返回值：nil

示例：

```
-- 将 GPIO0 设置为输出模式
gpio.mode(0, gpio.OUTPUT)
```

参见： - gpio.read()

gpio.read()

描述：读取管脚电平高低。

语法：gpio.read(pin)

参数：pin: 0~12, IO 编号

返回值：number:0：低电平, 1：高电平。

```
示例：    -- 读取 GPIO0 的电平
    gpio.read(0)
```

参见：- gpio.mode()

gpio.write()

描述：设置管脚电平

语法：gpio.write(pin)

参数：pin: 0~12, IO 编号

level: gpio.HIGH or gpio.LOW

返回值：nil

```
示例：    -- 设置 GPIO 1 为输出模式，并将输出电平设置为高
    pin=1
    gpio.mode(pin, gpio.OUTPUT)
    gpio.write(pin, gpio.HIGH)
```

参见：- gpio.mode()

- gpio.read()

gpio.trig()

描述：设置管脚中断模式的回调函数。

语法：gpio.trig(pin, type, function(level))

参数：pin: **1~12**, IO 编号。注意 pin0 不支持中断。

type: 取值为"up", "down", "both", "low", "high", 分别代表上升沿、下降沿、双边沿、低电平、高电平触发方式。
function(level): 中断触发的回调函数，GPIO 的电平作为输入参数：。如果此处没有定义函数，则使用之前定义的回调函数。

返回值：nil

示例：
```
    -- 使用 GPIO0 检测输入脉冲宽度
    pulse1 = 0
    du = 0
    gpio.mode(1,gpio.INT)
    function pin1cb(level)
     du = tmr.now() - pulse1
     print(du)
     pulse1 = tmr.now()
     if level == 1 then gpio.trig(1, "down ") else gpio.trig(1, "up ") end
    end
    gpio.trig(1, "down ",pin1cb)
```

参见：- gpio.mode()

- gpio.write()

PWM 模块

pwm.setup()

描述：设置管脚为 pwm 模式，最多支持 6 个 pwm。

语法：pwm.setup(pin, clock, duty)

参数：pin: 1~12, IO 编号

clock: 1~1000, pwm 频率

duty: 0~1023, pwm 占空比，最大 1023（10bit）。

返回值：nil

示例：
```
-- 将管脚 1 设置为 pwm 输出模式，频率 100Hz，占空比 50-50
pwm.setup(1, 100, 512)
```

参见：- pwm.start()

pwm.close()

描述：退出 pwm 模式。

语法：pwm.close(pin)

参数：pin: 1~12, IO 编号

返回值：nil

示例：
```
pwm.close(1)
```

参见：- pwm.start()

pwm.start()

描述：pwm 启动，可以在对应的 GPIO 检测到波形。

语法：pwm.start(pin)

参数：pin: 1~12, IO 编号

返回值：nil

```
示例：    pwm.start(1)
```

参见：- pwm.stop()

pwm.stop()

描述：暂停 pwm 输出波形。

语法：pwm.stop(pin)

参数：pin: 1~12, IO 编号

返回值：nil

```
示例：    pwm.stop(1)
```

参见：- pwm.start()

pwm.setclock()

描述：设置 pwm 的频率

-Note: 设置 pwm 频率将会同步改变其他 pwm 输出的频率，当前版本的所有 pwm 仅支持同一频率输出。

语法：pwm.setclock(pin, clock)

参数：pin: 1~12, IO 编号

clock: 1~1000, pwm 周期

返回值：nil

示例：

```
pwm.setclock(1, 100)
```

参见： - pwm.getclock()

pwm.getclock()

描述： 获取 pin 的 pwm 工作频率

语法： pwm.getclock(pin)

参数： pin: 1~12, IO 编号

返回值： number:pin 的 pwm 工作频率

示例：

```
print(pwm.getclock(1))
```

参见： - pwm.setclock()

pwm.setduty()

描述： 设置 pin 的占空比。

语法： pwm.setduty(pin, duty)

参数： pin: 1~12, IO 编号

duty: 0~1023, pwm 的占空比，最大为 1023.

返回值： nil

示例：

```
pwm.setduty(1, 512)
```

参见： - pwm.getduty()

pwm.getduty()

描述：获取 pin 的 pwm 占空比。

语法：pwm.getduty(pin)

参数：pin: 1~12, IO 编号

返回值：number: 该 pin 的 pwm 占空比，最大为 1023.

示例：

```
    -- D1 连接绿色 led
    -- D2 连接蓝色 led
    -- D3 连接红色 led
    pwm.setup(1,500,512)
    pwm.setup(2,500,512)
    pwm.setup(3,500,512)
    pwm.start(1)
    pwm.start(2)
    pwm.start(3)
    function led(r,g,b)
      pwm.setduty(1,g)
      pwm.setduty(2,b)
      pwm.setduty(3,r)
    end
    led(512,0,0) --  led 显示红色
    led(0,0,512) -- led 显示蓝色
```

参见：- pwm.setduty()

net 模块

常量： net.TCP, net.UDP

net.createServer()

描述：创建一个 server。

语法：net.createServer(type, timeout)

参数：type: 取值为：net.TCP 或者 net.UDP

timeout: 1~28800, 当为 tcp 服务器时，客户端的超时时间设置。

返回值：net.server 子模块

示例：
```
net.createServer(net.TCP, 30)
```
参见：- net.createConnection()

net.createConnection()

描述：创建一个 client。

语法：net.createConnection(type, secure)

参数：type: 取值为：net.TCP 或者 net.UDP

secure: 设置为 1 或者 0, 1 代表安全连接，0 代表普通连接。

返回值：net.server 子模块

示例：
```
net.createConnection(net.UDP, 0)
```
参见：- net.createServer()

net.server 子模块

listen()

描述：侦听指定 ip 地址的端口。

语法：net.server.listen(port,[ip],function(net.socket))

参数：port: 端口号

ip:ip 地址字符串，可以省略

function(net.socket): 连接创建成功的回调函数，可以作为参数：传给调用函数。

返回值：nil

示例：

```
    -- 创建一个 server
    sv=net.createServer(net.TCP, 30)  -- 30s 超时
    -- server 侦听端口 80，如果收到数据将数据打印至控制台，并向远端发送‘hello world’
    sv:listen(80,function(c)
      c:on("receive", function(c, pl) print(pl) end)
      c:send("hello world")
      end)
```

参见：- net.createServer()

close()

描述：关闭 server

语法：net.server.close()

参数：nil

返回值：nil

示例：

```
    -- 创建 server
    sv=net.createServer(net.TCP, 5)
    -- 关闭 server
```

```
sv:close()
```

参见： - net.createServer()

net.socket 子模块

connect()

描述： 连接至远端。

语法： connect(port, ip/domain)

参数： port: 端口号

ip: ip 地址或者是域名字符串

返回值： nil

参见： - net.socket:on()

send()

描述： 通过连接向远端发送数据。

语法： send(string, function(sent))

参数： string: 待发送的字符串

function(sent): 发送字符串后的回调函数。

返回值： nil

参见： - net.socket:on()

on()

描述：向事件注册回调函数。

语法：on(event, function cb())

参数：event: 字符串，取值为: "connection", "reconnection", "disconnection", "receive", "sent"

function cb(net.socket, [string]): 回调函数。第一个参数：是 socket.

如果事件是"receive", 第二个参数：则为接收到的字符串。

返回值：nil

示例：

```
    sk=net.createConnection(net.TCP, 0)
    sk:on("receive", function(sck, c) print(c) end )
    sk:connect(80,"192.168.0.66")
    sk:send("GET / HTTP/1.1\r\nHost: 192.168.0.66\r\nConnection: keep-alive\r\nAccept: */*\r\n\r\n")
```

参见：- net.createServer()

close()

描述：关闭 socket。

语法：close()

参数：nil

返回值：nil

参见：- net.createServer()

dns()

描述：获取当前域的 ip

语法：dns(domain, function(net.socket, ip))

参数：domain: 当前域的名称

function (net.socket, ip): 回调函数。第一个参数：是 socket，第二个参数：是当前域的 ip 字符串。

返回值：nil

示例：
```
    sk=net.createConnection(net.TCP, 0)
    sk:dns("www.nodemcu.com",function(conn,ip) print(ip) end)
    sk = nil
```

参见：- net.createServer()

i2c 模块

常量： i2c.SLOW, i2c.TRANSMITTER, i2c. RECEIVER. FAST（400k）模式目前不支持。

i2c.setup()

描述：初始化 i2c。

语法：i2c.setup(id, pinSDA, pinSCL, speed)

参数：id = 0

pinSDA: 1~12, IO 编号

pinSCL: 1~12, IO 编号

speed: i2c.SLOW

返回值：返回设置的速度

参见：- i2c.read()

i2c.start()

描述：启动 i2c 传输。

语法：i2c.start(id)

参数：id = 0

返回值：nil

参见：- i2c.read()

i2c.stop()

描述：停止 i2c 传输。

语法：i2c.stop(id)

参数：id = 0

返回值：nil

参见：- i2c.read()

i2c.address()

描述：设置 i2c 地址以及读写模式。

语法：i2c.address(id, device_addr, direction)

参数：id=0

device_addr: 设备地址。

direction: i2c.TRANSMITTER：写模式；i2c. RECEIVER：读模式。

返回值：true: 收到 ack false：没有收到 ack

参见：- i2c.read()

i2c.write()

描述：向 i2c 写数据。数据可以是多个数字，字符串或者 lua table。

语法：i2c.write(id, data1, data2,...)

参数：id=0

data: 数据可以是多个数字，字符串或者 lua table。

返回值：number：成功写入的字节个数

示例：
```
i2c.write(0, "hello", "world")
```
参见：- i2c.read()

i2c.read()

描述：读取 len 个字节的数据。

语法：i2c.read(id, len)

参数：id=0

len: 数据长度。

返回值：string:接收到的数据。

示例：

```
id=0
sda=1
scl=2

-- 初始化 i2c，将 pin1 设置为 sda，将 pin2 设置为 scl
i2c.setup(id,sda,scl,i2c.SLOW)

-- 用户定义函数:读取地址 dev_addr 的寄存器 reg_addr 中的内容。
function read_reg(dev_addr, reg_addr)
  i2c.start(id)
  i2c.address(id, dev_addr ,i2c.TRANSMITTER)
  i2c.write(id,reg_addr)
  i2c.stop(id)
  i2c.start(id)
  i2c.address(id, dev_addr,i2c.RECEIVER)
  c=i2c.read(id,1)
  i2c.stop(id)
  return c
end

-- 读取 0x77 的寄存器 0xAA 中的内容。
reg = read_reg(0x77, 0xAA)
print(string.byte(reg))
```

参见：- i2c.write()

adc 模块

常量： 无

adc.read()

描述：读取 adc 的值，esp8266 只有一个 10bit adc，id 为 0，引脚为 TOUT，最大值 1024

语法：adc.read(id)

参数：id = 0

返回值：adc 值 10bit，最大 1024.

参见：-

uart 模块

常量： 无

uart.setup()

描述：设置 uart 的波特率，字节长度，校验，停止位，是否 echo。

语法：uart.setup(id, baud, databits, parity, stopbits, echo)

参数：id = 0, 只支持一个串口

baud = 300, 600, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 74880, 115200, 230400, 460800, 921600, 1843200, 2686400

databits = 5, 6, 7, 8。表示字节长度。

parity = 0(none)。

stopbits = 1(1 stopbit), 2(2 stopbit).

echo = 0(关闭回显)。

返回值：返回波特率

参见：-

uart.on()

描述：设置 uart 的事件回调函数，目前支持"data"事件，表示 uart 收到了数据，以行为单位。

语法：uart.on(method, function, [run_input])

参数：method = "data"，表示 uart 接收到了数据

function 为回调函数，"data" 的回调函数签名为 function(data) end

run_input: 0 或 1，0 表示从 uart 输入的 data 不经过 lua 解释器执行，1 表示输入的行会被送到 lua 解释器执行。

返回值：nil

示例：

```
    uart.on("data",
    function(data)
      print("receive from uart:", data)
      if data=="quit" then
        uart.on("data")
      end
  end, 0)
```

参见：-

uart.write()

描述：向串口写入数据。

语法：uart.write(id, string1, string2...)

参数：id = 0, 只支持一个串口

string1：需要写入的字符串。

返回值：nil

参见：-

onewire 模块

常量： 无

ow.setup()

描述：将 pin 设置为 one wire 模式。

语法：ow.setup(pin)

参数：pin: 1~12, IO 编号。

返回值：nil

参见：-

ow.reset()

描述：执行一次 1-wire 复位操作。

语法：ow.reset(pin)

参数：pin: 1~12, IO 编号

返回值：number: 如果有器件响应返回 1，如果没有器件响应或者总线被拉低超过 250us 返回 0。

参见：-

ow.skip()

描述：发送一个 1-wire“忽略 rom”的命令，可以用来寻址总线上的所有器件。

语法：ow.skip(pin)

参数：pin: 1~12, IO 编号

返回值：nil

参见：-

ow.select()

描述：发送一个 1-Wire“选择 rom”的命令，执行此函数前请务必先执行 ow.reset()函数。

语法：ow.select(pin,rom)

参数：pin: 1~12, IO 编号

rom: 包含 slave 器件 rom 内容的 8 个字节长度的 string。

返回值：nil

示例：

```
-- 18b20 Example
pin = 9
ow.setup(pin)
count = 0
repeat
  count = count + 1
  addr = ow.reset_search(pin)
  addr = ow.search(pin)
  tmr.wdclr()
until((addr ~= nil) or (count > 100))
if (addr == nil) then
  print("No more addresses.")
else
  print(addr:byte(1,8))
  crc = ow.crc8(string.sub(addr,1,7))
  if (crc == addr:byte(8)) then
    if ((addr:byte(1) == 0x10) or (addr:byte(1) == 0x28)) then
      print("Device is a DS18S20 family device.")
        repeat
          ow.reset(pin)
          ow.select(pin, addr)
          ow.write(pin, 0x44, 1)
          tmr.delay(1000000)
          present = ow.reset(pin)
          ow.select(pin, addr)
          ow.write(pin,0xBE,1)
          print("P="..present)
          data = nil
          data = string.char(ow.read(pin))
          for i = 1, 8 do
            data = data .. string.char(ow.read(pin))
          end
          print(data:byte(1,9))
          crc = ow.crc8(string.sub(data,1,8))
```

```
          print("CRC="..crc)
          if (crc == data:byte(9)) then
             t = (data:byte(1) + data:byte(2) * 256) * 625
             t1 = t / 10000
             t2 = t % 10000
             print("Temperature="..t1.."."..t2.."Centigrade")
          end
          tmr.wdclr()
        until false
    else
      print("Device family is not recognized.")
    end
  else
    print("CRC is not valid!")
  end
end
```

参见：-

ow.write()

描述：向选定的 slave 写一个字节。

语法：ow.write(pin, v, power)

参数：pin: 1~12, IO 编号

v: 向 slave 器件发送的字节

power: 1，用于向寄生供电器件供电；0，不需要寄生供电。注意：请务必调用 ow.depower()或者发起新的读写操作来取消寄生供电。

返回值：nil

参见：-

ow.write_bytes()

描述：向选定的 slave 写多个字节。

语法：ow.write_bytes(pin, buf, power)

参数：pin: 1~12, IO 编号

buf: 向 slave 发送的多个字节的字符串

power: 1，用于向寄生供电器件供电；0，不需要寄生供电。注意：请务必调用ow.depower()或者发起新的读写操作来取消寄生供电。

返回值：nil

参见：-

ow.read()

描述：从选定的 slave 读取一个字节。

语法：ow.read(pin)

参数：pin: 1~12, IO 编号

返回值：从 slave 读取的一个字节。

参见：-

ow.read_bytes()

描述：从选定的 slave 读取多个字节。

语法：ow.read_bytes(pin, size)

参数：pin: 1~12, IO 编号

size: 需要从 slave 读取的字节的个数

返回值：从 slave 返回的多个字节的字符串。

参见：-

ow.depower()

描述：取消向总线供电。仅需在 ow.write()或者 ow.write_bytes()中的'power=1'且 不再进行读写 slave 的情况下使用。

语法：ow.depower(pin)

参数：pin: 1~12, IO 编号

返回值：nil

参见：-

ow.reset_search()

描述：清除查找状态用于重新开始进行查找操作。

语法：ow.reset_search(pin)

参数：pin: 1~12, IO 编号

返回值：nil

参见：-

ow.target_search()

描述：设置查找选项'family_code'，用于在下次调用 ow.search()时查找该'family_code'的器件。

语法：ow.target_search(pin, family_code)

参数：pin: 1~12, IO 编号

family_code: family_code 字节

返回值：nil

参见：-

ow.search()

描述：寻找下一个 slave 器件。

语法：ow.search(pin)

参数：pin: 1~12, IO 编号

返回值：查找成功则返回 slave 器件的 8 个字节的 rom code 字符串；

查找失败则返回 nil

参见：-

ow.crc8()

描述：计算 Dallas Semiconductor 的 8 位 CRC, 用于与 ROM 或者暂存器中的内容进行比较。

语法：ow.crc8(buf)

参数：buf: 需要进行 crc8 计算的字符串

返回值：crc 结果字节

参见：-

ow.check_crc16()

描述：计算 1-Wire 的 CRC16 并与接收的 CRC 结果进行比较。

语法：ow.check_crc16(buf, inverted_crc0, inverted_crc1, crc)

参数：buf: 需要进行 crc8 计算的字符串

inverted_crc0: 接收到的 CRC 结果的低字节

inverted_crc1: 接收到的 CRC 结果的高字节

crc: crc 初始值 (可选)

返回值：布尔值: true, crc 结果相符; false，crc 结果不相符。

参见：-

ow.crc16()

描述：计算 Dallas Semiconductor 的 16 位的 CRC 值。用于 1-wire 总线中多器件通信的数据完整性校验。请注意：这里的 CRC 计算结果并不一定是 1-wire 总线中获得的 CRC，原因如下：

1) 1-wire 总线传输的 CRC 是低位先传输的。

2) 另外由于因处理器而异的字节顺序，ow.crc16()返回结果的 MSB 和 LSB 顺序可能不同于 1-wire 总线中获取的 MSB 和 LSB 顺序。

语法：ow.crc16(buf, crc)

参数：buf: 需要进行 crc8 计算的字符串

crc: crc 初始值 (可选)

返回值：返回 16 位的 Dallas Semiconductor CRC 计算结果

参见：-

bit 模块

常量： none

bit.bnot()

描述：按位取反，相当于 C 语言中的'~value'。

语法：bit.bnot(value)

参数：value: 取反操作数

返回值：number: 按位取反后的结果

参见：-

bit.band()

描述：按位与，相当于 C 语言中的'val1 & val2 & ... & valn'。

语法：bit.band(val1, val2, ... valn)

参数：val1: 第一个'与'操作数

val2: 第二个'与'操作数

valn: 第 n 个'与'操作数

返回值：number: 所有操作数按位'与'操作的结果

参见：-

bit.bor()

描述：按位或，相当于 C 语言中的 val1 | val2 | ... | valn。

语法：bit.bor(val1, val2, ... valn)

参数：val1: 第一个'或'操作数

val2: 第二个'或'操作数

valn: 第 n 个'或'操作数

返回值：number: 所有操作数按位'或'操作的结果

参见：-

bit.bxor()

描述：按位异或，相当于 C 语言中的 val1 ^ val2 ^ ... ^ valn。

语法：bit.bxor(val1, val2, ... valn)

参数：val1: 第一个'异或'操作数

val2: 第二个'异或'操作数

valn: 第 n 个'异或'操作数

返回值：number: 所有操作数按位'异或'操作的结果

参见：-

bit.lshift()

描述：按位左移一个操作数，相当于 C 语言中的 value<

语法：bit.lshift(value, shift)

参数：value: 按位左移的操作数

shift: 左移的偏移量

返回值：number: 按位左移的结果

参见：-

bit.rshift()

描述：逻辑右移一个操作数，相当于 C 语言中的无符号数 value>>shift。

语法：bit.rshift(value, shift)

参数：value: 按位右移的操作数

shift: 右移的偏移量

返回值：number: 按位右移的结果（按无符号数处理）

参见：-

bit.arshift()

描述：算术右移一个操作数，相当于 C 语言中的 value>>shift。

语法：bit.arshift(value, shift)

参数：value: 按位右移的操作数

shift: 右移的偏移量

返回值：number: 按位右移的结果（算术右移）

参见：-

bit.bit()

描述：将某一个位设置为 1，相当于 C 语言中的 1 << position。

语法：bit.bit(position)

参数：position: 需要设置为 1 的位序。

返回值：number: 某位设置为 1 的结果 (其余位设为 0)

参见：-

bit.set()

描述：将某些位设置为 1。

语法：bit.set(value, pos1, pos2, ..., posn)

参数：value: 操作数

pos1: 第一个需要设置为 1 的位序

pos2: 第二个需要设置为 1 的位序

posn: 第 n 个需要设置为 1 的位序

返回值：number: 将特定位设置为 1 的结果

参见：-

bit.clear()

描述：将某些位设置为 0。

语法：bit.clear(value, pos1, pos2, ..., posn)

参数：value: 操作数

pos1: 第一个需要设置为 0 的位序

pos2: 第二个需要设置为 0 的位序

posn: 第 n 个需要设置为 0 的位序

返回值：number: 将特定位设置为 0 的结果

参见：-

bit.isset()

描述：测试特定位是否为 1。

语法：bit.isset(value, position)

参数：value: 需要测试的操作数

position: 需要测试的位序

返回值：boolean: 如果指定位序为 1，返回 true，否则返回 false

参见：-

bit.isclear()

描述：测试特定位是否为 0。

语法：bit.isclear(value, position)

参数：value: 需要测试的操作数

position: 需要测试的位序

返回值：boolean: 如果指定位序为 0，返回 true，否则返回 false

参见：-

spi 模块

常量： MASTER, SLAVE, CPHA_LOW, CPHA_HIGH, CPOL_LOW, CPOL_HIGH, DATABITS_8, DATABITS_16

spi.setup()

描述：配置 spi.

语法：spi.setup(id, mode, cpol, cpha, databits, clock)

参数：id: spi id 号.

mode: MASTER 或者 SLAVE(目前不支持).

cpol: CPOL_LOW 或者 CPOL_HIGH, 时钟极性.

cpha: CPHA_HIGH 或者 CPHA_LOW, 时钟相位.

databits: DATABITS_8 或者 DATABITS_16.

clock: spi 时钟 (目前不支持).

返回值：number: 1.

示例：参见：-

- Back to Index

spi.send()

描述：向 spi 设备发送数据.

语法：wrote = spi.send(id, data1, [data2], ..., [datan])

参数：id: spi id 号.

data: data 可以是字符串、Lua table 或者 8 位数值.

返回值：number: 发送数据的字节数.

示例：参见：-

- Back to Index

spi.recv()

描述：从 spi 设备接收数据.

语法：read = spi.recv(id, size)

参数：id: spi id 号.

size: 需要读取数据的字节数.

返回值：string: 读取的字符串（字节形式）.

示例：参见：-

- Back to Index

mqtt 模块

常量：mqtt.Client()

描述：创建一个 mqtt client.

语法：mqtt.Client(clientid, keepalive, user, pass)

参数：clientid: mqtt 客户端 id.

keepalive: 保持连接的时间，单位：秒.

user: 用户名，字符串.

pass: 密码，字符串.

返回值：mqtt 客户端.

示例：
```
-- 创建一个 mqtt client，保持连接包时间 120s.
m = mqtt.Client("clientid", 120, "user", "password")

--设置 Last Will 和 Testament (可选).
--如果 mqtt client 不发送保持连接包，服务器会向标题"/lwt"发送一个 qos = 0, retain = 0, data = "offline"的消息.
m:lwt("/lwt", "offline", 0, 0)

m:on("connect", function(con) print ("connected") end)
m:on("offline", function(con) print ("offline") end)

-- 接收到消息事件
m:on("message", function(conn, topic, data)
  print(topic .. ":" )
```

```
  if data ~= nil then
    print(data)
  end
end)

-- 如果需要安全连接，则 m:connect("192.168.11.118", 1880, 1)
m:connect("192.168.11.118", 1880, 0, function(conn) print("connected") end)

-- 订阅"/topic"消息，qos = 0
m:subscribe("/topic",0, function(conn) print("subscribe success") end)

-- 向"/topic"标题发送消息，消息设置：data = hello, QoS = 0, retain = 0
m:publish("/topic","hello",0,0, function(conn) print("sent") end)

m:close();
-- 或者可以再次调用 m:connect()
```

参见：-

- Back to Index

mqtt client 子模块

mqtt:lwt()

描述：设置 Last Will 和 Testament (可选)

如果 mqtt client 不发送保持连接包，服务器会向标题"/lwt"发送一个 qos = 0, retain = 0, data = "offline"的消息.

语法：mqtt:lwt(topic, message, qos, retain)

参数：topic: 需要发布消息的标题，字符串类型.

message: 需要发布的消息， Buffer 或者字符串.

qos: qos 值， 默认值为 0.

retain: 保留标志，默认值为 0.

返回值：nil.

示例：参见：-

- Back to Index

mqtt:connect()

描述：连接到 mqtt 服务器.

语法：mqtt:connect(host, port, secure, function(client))

参数：host: 主机域名或者 ip 地址，字符串类型.

port: 服务器端口号.

secure: 0 或者 1, 默认值为 0.

function(client): 连接成功的回调函数.

返回值：nil.

示例：参见：-

- Back to Index

mqtt:close()

描述：关闭 mqtt 连接.

语法：mqtt:close()

参数：nil

返回值：nil.

示例：参见：-

- Back to Index

mqtt:publish()

描述：发布一个消息.

语法：mqtt:publish(topic, payload, qos, retain, function(client))

参数：topic: 需要发布消息的标题，字符串类型.

message: 需要发布的消息，字符串类型.

qos: qos 值，默认值为 0.

retain: 保留标志，默认值为 0.

function(client): 发送成功的回调函数，如果接收到 PUBACK 回调函数解除.

返回值：nil.

示例：参见：-

- Back to Index

mqtt:subscribe()

描述：订阅一个或者多个标题的消息.

语法：mqtt:subscribe(topic, qos, function(client, topic, message))

参数：topic: 需要订阅消息的标题.

qos: 订阅消息的 qos 值, 默认值为 0

function(client, topic, message): 接收消息的回调函数，接收后即解除.

返回值：nil.

示例：参见：-

- Back to Index

mqtt:on()

描述：注册 mqtt 事件的回调函数.

语法：mqtt:on(event, function(client, [topic], [message]))

参数：event: 字符串，取值为: "connect", "message", "offline"

function cb(client, [topic], [message]): 事件触发的回调函数. 第一个参数：是 mqtt client.

如果事件是"message", 第二个和第三个参数：分别是标题和消息内容，字符串类型.

返回值：nil.

示例：参见：-

WS2812 模块

常量：无

ws2812.writergb()

描述：将 RGB 编码成 8bit 数据发送至 WS2812

语法：ws2812.writeegb(pin, string.char(R1,G1,B1(,R2,G2,B2...)))

参数：pin = 支持所有 PIN(0,1,2...)

R1 = 级联的第一个 WS2812 的红色通道（0-255）

G1 = 级联的第一个 WS2812 的绿色通道（0-255）

B1 = 级联的第一个 WS2812 的蓝色通道（0-255）

... 可几乎无限级联,R2,G2,B2 为下一个级联的 LED 的红绿蓝参数；**返回值：**nil

参见：-