API说明

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及以后的版本采用)

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

#### [\*] 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();

#### 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);

## 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();

## 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();

## 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')})

## 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);**

## 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)

## 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.

# 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")

具体可以参见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()

## 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()

## 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()

## 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()

## 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()

## 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()

## 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()

## 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

# 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()6](http://bbs.nodemcu.com/t/nodemcu-api/28" \l "wf_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()20](http://bbs.nodemcu.com/t/nodemcu-api/28" \l "wf_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()12](http://bbs.nodemcu.com/t/nodemcu-api/28" \l "wf_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()5](http://bbs.nodemcu.com/t/nodemcu-api/28" \l "wf_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()12](http://bbs.nodemcu.com/t/nodemcu-api/28" \l "ws_connect)  
****-**** [wifi.sta.disconnect()8](http://bbs.nodemcu.com/t/nodemcu-api/28" \l "ws_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()8](http://bbs.nodemcu.com/t/nodemcu-api/28" \l "ws_disconnect)  
  
****-**** [wifi.sta.config()21](http://bbs.nodemcu.com/t/nodemcu-api/28" \l "ws_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()21](http://bbs.nodemcu.com/t/nodemcu-api/28" \l "ws_config)  
  
****-**** [wifi.sta.connect()12](http://bbs.nodemcu.com/t/nodemcu-api/28" \l "ws_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()21](http://bbs.nodemcu.com/t/nodemcu-api/28" \l "ws_config)  
  
****-**** [wifi.sta.connect()12](http://bbs.nodemcu.com/t/nodemcu-api/28" \l "ws_connect)  
  
****-**** [wifi.sta.disconnect()8](http://bbs.nodemcu.com/t/nodemcu-api/28" \l "ws_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()](http://bbs.nodemcu.com/t/nodemcu-api/28" \l "ws_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()5](http://bbs.nodemcu.com/t/nodemcu-api/28" \l "ws_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()5](http://bbs.nodemcu.com/t/nodemcu-api/28" \l "ws_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()](http://bbs.nodemcu.com/t/nodemcu-api/28" \l "wa_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()3](http://bbs.nodemcu.com/t/nodemcu-api/28" \l "wa_getip)

# timer 模块

## tmr.delay()

#### 描述

延迟us微秒。

#### 语法

tmr.dealy(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()12](http://bbs.nodemcu.com/t/nodemcu-api/28" \l "tm_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()4](http://bbs.nodemcu.com/t/nodemcu-api/28" \l "tm_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()12](http://bbs.nodemcu.com/t/nodemcu-api/28" \l "tm_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()12](http://bbs.nodemcu.com/t/nodemcu-api/28" \l "tm_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()4](http://bbs.nodemcu.com/t/nodemcu-api/28" \l "tm_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()4](http://bbs.nodemcu.com/t/nodemcu-api/28" \l "io_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()11](http://bbs.nodemcu.com/t/nodemcu-api/28" \l "io_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()11](http://bbs.nodemcu.com/t/nodemcu-api/28" \l "io_mode)  
  
****-**** [gpio.read()4](http://bbs.nodemcu.com/t/nodemcu-api/28" \l "io_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()11](http://bbs.nodemcu.com/t/nodemcu-api/28" \l "io_mode)  
  
****-**** [gpio.write()1](http://bbs.nodemcu.com/t/nodemcu-api/28" \l "io_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()7](http://bbs.nodemcu.com/t/nodemcu-api/28" \l "pw_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()7](http://bbs.nodemcu.com/t/nodemcu-api/28" \l "pw_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()1](http://bbs.nodemcu.com/t/nodemcu-api/28" \l "pw_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()7](http://bbs.nodemcu.com/t/nodemcu-api/28" \l "pw_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()](http://bbs.nodemcu.com/t/nodemcu-api/28" \l "pw_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()](http://bbs.nodemcu.com/t/nodemcu-api/28" \l "pw_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()](http://bbs.nodemcu.com/t/nodemcu-api/28" \l "pw_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()3](http://bbs.nodemcu.com/t/nodemcu-api/28" \l "pw_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()12](http://bbs.nodemcu.com/t/nodemcu-api/28" \l "nt_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()35](http://bbs.nodemcu.com/t/nodemcu-api/28" \l "nt_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(sck, pl) print(pl) end)

c:send("hello world")

end)

#### 参见

****-**** [net.createServer()35](http://bbs.nodemcu.com/t/nodemcu-api/28" \l "nt_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()35](http://bbs.nodemcu.com/t/nodemcu-api/28" \l "nt_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()44](http://bbs.nodemcu.com/t/nodemcu-api/28" \l "nk_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()44](http://bbs.nodemcu.com/t/nodemcu-api/28" \l "nk_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()35](http://bbs.nodemcu.com/t/nodemcu-api/28" \l "nt_createServer)

## close()

#### 描述

关闭socket。

#### 语法

close()

#### 参数

nil

#### 返回值

nil

#### 参见

****-**** [net.createServer()35](http://bbs.nodemcu.com/t/nodemcu-api/28" \l "nt_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()35](http://bbs.nodemcu.com/t/nodemcu-api/28" \l "nt_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()5](http://bbs.nodemcu.com/t/nodemcu-api/28" \l "ic_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()5](http://bbs.nodemcu.com/t/nodemcu-api/28" \l "ic_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()5](http://bbs.nodemcu.com/t/nodemcu-api/28" \l "ic_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()5](http://bbs.nodemcu.com/t/nodemcu-api/28" \l "ic_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()5](http://bbs.nodemcu.com/t/nodemcu-api/28" \l "ic_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()](http://bbs.nodemcu.com/t/nodemcu-api/28" \l "ic_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 = 9600, 19200, 38400, 57600, 74880, 115200, 230400, 460800, 921600。115200以上未测试。  
  
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

## CONSTANT

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