NodeMCU-API 中文说明

Version 2.2.1 build 2019-08-19 By:dreamofTaotao

**概述**

1. 易编程无线节点/接入点。
2. 基于Lua5.1.4（没有debug&os模块）。
3. 装载异步事件驱动编程。
4. 超过65内置模块，但是该参考手册暂时只有17个常用内置模块，
5. 固件支持floating模式（integer仅支持小内存）。
6. 英文参考文献地址：[https://nodemcu.readthedocs.io](https://nodemcu.readthedocs.io/)。
7. 博主英文水平有限，欢迎指正错误。
8. 持续更新……

**ADC模块**

ADC模块提供了接入内置ADC。

在ESP8266中，仅有一个频道是电池电压有多路复用。依据设置在“esp init data”（107bit），其中一个也能用ADC去进行读取扩展电压，或者读取系统电压（VDD-3.3V），但是需要注意的是不能同时读取。

adc.force\_init\_mode()函数可以进行配置ADC的模式。注意在从一个系统到另一个系统需要重新开始（例如，电源重新连接、重置按钮或者是node.restart()），这个是必要的在更改生效前。

**函数表：**

|  |  |
| --- | --- |
| adc.force\_init\_mode() | 检查并在必要时重置在ESP数据初始化块中（init）中的ADC模式设置 |
| adc.read() | ADC读取采样 |
| adc.readvdd33() | 读取系统电压 |

**adc.force\_iniy\_mode()**

检查并在必要时重置在ESP数据初始化块中（init）中的ADC模式设置。

**使用语法：**

adc.force\_init\_mode(mode\_value)

**参数介绍：**

mode\_value：adc.INIT\_ADC或者adc.INIT\_VDD33其中之一。

**返回值：**

如果这个函数不得不改变当前模式时返回值为：true

如果当前模式已经配置完成时返回值为：flase

当返回值为：true时ESP需要需要重新启动使其生效，重启方式参考：电源重新连接、重置按钮或者是node.restart()。

**Example：**

-- in you init.lua:

**if** adc.force\_init\_mode(adc.INIT\_VDD33)

**then**

node.restart()

**return** -- don't bother continuing, the restart is scheduled

**end**

print("System voltage (mV):", adc.readvdd33(0))

**adc.read()**

ADC读取采样

**使用语法：**

adc.read(channel)

**参数介绍：**

channel：在ESP8266中总是0

**返回值：**

例子中的值（number）

如果ESP8266已经配置使用ADC读取系统电压，这个函数将总是返回65535。这个是硬件或者说是SDK的限制。

**Example：**

val=adc.read(0)

**adc.readvdd33()**

读取系统电压

**使用语法：**

adc.readvdd33()

**参数介绍：**

none

**返回值：**

系统电压，单位为毫伏（number）

如果ESP8266已经配置使用ADC实例扩展引脚进行采样，这个函数将总是返回65535。这个是硬件或者说是SDK的限制。

**cryto模块**

这个crypto模块为使用加密算法提供了多种函数。

接下来的加密解密算法模式被支持：

<1>“AES-ECB”：对于128bit的AES在ECB模式中（该种模式不推荐使用）

<2> “AES-ECB”：对于128bit的AES在CBC模式中。

下面hash（哈希）算法被支持：

<1>MD2（默认不可用，必须在app/include/user\_config.h文件中进行使能操作）

<2>MD5、SHA1、SHA256、SHA384、SHA512（在app/include /user\_config.h文件不使能）。

**函数表：**

|  |  |
| --- | --- |
| crypto.encrypt() | Lua字符串加密 |
| crypto.decrypt() | 解密之前加密的数据（字符串） |
| crypto.fhash() | 计算文件的加密哈希 |
| crypto.hash() | 计算一个Lua字符串的加密哈希 |
| crypto.new\_hash() | 创建可以添加任何数量字符串的哈希对象 |
| crypto.hmac() | 计算一个HMAC（哈希信息验证代码）的签名对于一个Lua字符串 |
| crypto.new\_hmac() | 创建可以添加任何数量字符串HMAC对象 |
| crypto.mask | 使用XOR（或非门）掩码应用于Lua字符串加密 |
| crypto.toBase64() | 提供二进制Lua字符串的Base64表示 |
| crypto.toHex() | 提供二进制Lua字符串的ASCII十六进制表示 |

**crypto.encrypt()**

Lua字符串加密。

**使用语法：**

crypto.encrypt(algo,key,plain[,iv])

**参数介绍：**

algo：将要在代码中支持使用的加密算法的名称。

key：加密的键设置为字符串；如果使用AES加密，那么这个字长必须设置成16 bytes。（解释：就是把需要加密的字符串变成字符串的样式）。

plain：需要加密的字符串；如果必要的话，将会自动的将0填充至为16-byte的边界。

iv：如果使用AES-CBC，初始化向量；如果没有赋值的话，默认为0。

**返回值：**

以二进制字符串形式加密的数据。对于AES返回值将总会是16字节长度的倍数。

**Example：**

print(crypto.toHex(crypto.encrypt(“AES-ECB”,”1234567890abcdef”,”Hi,I’m secret”)))

**crypto.decrypt()**

解密之前加密的数据。

**使用语法：**

crypto.decrypt(algo,key,cipher[,iv])

**参数介绍：**

algo：将要在代码中使用被支持的加密算法的名称。

key：加密的键设置为字符串；如果使用AES加密，那么这个字长必须设置成16 bytes。（解释：就是把需要加密的字符串变成字符串的样式）。

plain：要解密的密码文本（需要从crypto.encrypt()中获取） 。

iv：如果使用AES-CBC，初始化向量；如果没有赋值的话，默认为0。

**返回值：**

解密后的字符串.

注意解密后的字符串可能包括额外的0字节的填充在字符串结尾。一种剔除这个边缘值得方法是在解密字符串中加入：

“:match(“(.-)%z\*$”)”。另外还需要注意的是如果是在二进制下进行操作要小心，这个真实的字节长度可能需要被编码在数据中，并且“sub(1,n)”可以被用于去剔除填充值。

**Example：**

key = "1234567890abcdef"

cipher = crypto.encrypt("AES-ECB", key, "Hi, I'm secret!")

print(crypto.toHex(cipher))

print(crypto.decrypt("AES-ECB", key, cipher))

**crypto.fhash()**

计算文件的加密哈希

**使用语法：**

hash=crypto.fhash(algp,filename)

**参数介绍：**

algo：将要在代码中使用被支持的加密算法的名称，不区分大小写字符。

filename：需要哈希加密的文件路径。

**返回值：**

包括信息概要的二进制字符串。需要转化成文字版本（ASCII码十六进制字符）请参考使用函数：crypto.toHex()。

**Example：**

print(crypto.toHex(crypto.fhash(“SHA1”,”init.lua”)))

**crypto. hash()**

计算Lua字符串的加密哈希。

**使用语法：**

hash=crypto.hash(algo,str)

**参数介绍：**

algo：将要在代码中使用被支持的加密算法的名称，不区分大小写字符。

str：需要进行哈希算法的str字符串。

**返回值：**

包括信息概要的二进制字符串。需要转化成文字版本（ASCII码十六进制字符）请参考使用函数：crypto.toHex()。

**Example：**

print(crypto.toHex(crypto.hash(“SHA1”,”abc”)))

**crypto. new\_hash()**

创建一个能够添加任意数量字符串的哈希对象。对象有更新和完成的函数。

**使用语法：**

hashobj=crypto.new\_hash(algo)

**参数介绍：**

algo：将要在代码中使用被支持的加密算法的名称，不区分大小写字符。

**返回值：**

具有可用于更新和完成函数的UserData（用户数据）的对象。

**Example：**

hashobj = crypto.new\_hash("SHA1")

hashobj:update("FirstString")

hashobj:update("SecondString")

digest = hashobj:finalize()

print(crypto.toHex(digest))

**crypto. hmac()**

对一个Lua字符串计算一个HMAC（哈希信息验证代码）的签名。

**使用语法：**

signature = crypto .hmac(algo, str, key)

**参数介绍：**

algo：将要在代码中使用被支持的加密算法的名称，不区分大小写字符。

str：需要计算哈希的数据（字符串）。

key：用于签名的关键字（密钥），可能是二进制字符串。

**返回值：**

二进制字符串包含HMAC的签名。使用ctrypto.toHex()函数进行十六进制转换。

**Example：**

print(crypto.toHex(crypto.hmac(“SHA1”,”abc”,”mysecret”)))

**crypto.new\_hmac()**

创建一个可以添加任意数量字符串HMAC对象。对象具有更新和完成的函数。

**使用语法：**

hmacobj = crypto.new\_hmac(algo,key)

**参数介绍：**

algo：将要在代码中使用被支持的加密算法的名称，不区分大小写字符。

key：用于签名的关键字（密钥），可能是二进制字符串。

**返回值：**

具有可用于更新和完成函数的UserData（用户数据）的对象。

**Example：**

hashobj = crypto.new\_hmac("SHA1",”s3kr3t”)

hashobj:update("FirstString")

hashobj:update("SecondString")

digest = hashobj:finalize()

print(crypto.toHex(digesr))

**crypto.new\_hmac()**

使用XOR（或非门）掩码应用于Lua字符串加密.注意这不是一个适当的加密机制，然而有一些协议会使用他=它。

**使用语法：**

crypto.mask(message,mask)

**参数介绍：**

message：需要掩码的信息。

mask：应用于掩码，如果长度少于信息则进行重复。

**返回值：**

这个掩码信息是一个二进制字符串。可以使用crypto.toHex()获取（ASCII十六进制）文本格式。

**Example：**

print(crypto.toHex(crypto.mask(“some message to obscure”,”X0Y7”)))

**crypto.toBase64()**

提供一个二进制Lua字符串的Base64表示形式。

**使用语法：**

b64 = crypto.toBase64(binary)

**参数介绍：**

binary：输入字符串进行Base64编码。

**返回值：**

一个Base64数据形式的编码字符串。

**Example：**

print(crypto.toBase64(crypto.hash(“SHA1”,”abc”)))

**crypto.toBase64()**

提供一种二进制Lua字符串的ASCII十六进制表示形式。每一个被输入的字节被表示成两个十六进制的字符进行输出。

**使用语法：**

hexstr = crypto.toHex(binary)

**参数介绍：**

binary：输入字符串进行十六进制编码表示。

**返回值：**

一个ASCII十六进制字符串。

**Example：**

print(crypto.toHex(crypto.hash(“SHA1”,”abc”)))

**file模块**

file模块给文件系统和单个文件提供了接口。

file系统是平面文件系统，没有任何的文件分支（也就是文件夹的概念不存在）。

除了SPIFFS文件系统在内部的flash中，如果FatFS被使能允许的话，这个模块也可以在外置拓展的SD卡中接入FAT分区。例如，下面的代码：

*-- open file in flash:--打开在flash中文件*

if file.open("init.lua") t**hen**

print(file.read())

file.close()

**end**

*-- or with full pathspec—使用全路径打开相关文件*

file.open("/FLASH/init.lua")

*-- open file on SD card—打开SD卡中的文件*

if file.open("/SD0/somefile.txt") t**hen**

print(file.read())

file.close()

**end**

**函数表：**

|  |  |
| --- | --- |
| file.chdir() | 改变当前文件路径（和驱动器） |
| file.exists() | 确定指定的文件是否存在 |
| file.format() | 格式化文件系统 |
| file.fscfg() | 返回flash地址和文件系统的物理空间大小（字节） |
| file.fsinfo() | 返回文件系统的大小信息 |
| file.list() | 列出在文件系统中的所有文件 |
| file.mount() | 在SD卡中安装一个FatFs卷 |
| file.on() | 注册回调函数 |
| file.open() | 打开一个访问的文件，可能会创建它（用来写入模式） |
| file.remove() | 从文件系统中移除目标文件 |
| file.rename() | 文件重命名 |
| file.stat() | 获得表中目标文件或者目录属性 |
| Basic model | 在basic 模型中有最多只能有一个文件打开 |
| Object model | 文件被被文件创建的文件对象表示 |
| file.close()  file.obj:close() | 无条件关闭当前打开文件 |
| file.flush()  file.obj:close() | 刷新对文件系统的任何挂起写入操作，以防重新启动系统时丢失数据 |
| file.read()  file.obj:read() | 读取打开文件中的相关内容 |
| file.readline()  file.obj:readline() | 读取打开文件的下一行 |
| file.seek()  file.obj:seek() | 设置并且获取目标文件的位置（从文件开头开始测量），该位置由偏移量加上字符串从何处指定的基给出 |
| file.write()  file.obj:write() | 在打开文件中写字符串 |
| file.writeline()  file.obj:writeline() | 在打开文件中写字符串并在结尾处加”\n” |

**file.chdir()**

改变当前的文件目录和驱动。这被用在没有驱动或者目录。

在系统开始执行之后当前的目录默认由内部SPIFFS(Flash)的根目录。

注意：注意这个函数只有在烧录固件的时候有FatFS才可以使用。

**使用语法：**

file.chdir(dir)

**参数介绍：**

dir:文件的名称-/FLASH，/SD，/SD1等等。

**返回值：**

当成功时返回true，其他情况返回值为false。

**file.exists()**

确定指定的文件是否存在。

**使用语法：**

file.exists(filename)

**参数介绍：**

filename：需要被查看的文件名

**返回值：**

如果文件存在即使文件大小只有0字节，那么返回true；

当文件不存在时返回false。

**Example：**

files = file.list()

**i**f files["device.config"] then

print("Config file exists")

**end**

**if** file.exists("device.config") **then**

print("Config file exists")

**end**

**file.format()**

格式化文件系统。完全擦除之前的文件系统并且写一个新的。依靠在ESP中flash芯片的大小，这过程中可能会花费一些时间。

注意：这个函数不支持SD卡的格式化。

**使用语法：**

file.format()

**参数介绍：**

none（无）

**返回值：**

nil（无）

**file.fscfg()**

返回flash的地址和文件系统区的物理大小（字节大小）。

注意：该函数不支持SD卡。

**使用语法：**

file.fscfg()

**参数介绍：**

none（无）

**返回值：**

flash address：flash地址（数字类型）

size：大小 （数字类型）

**Example：**

print(string.format("0x%x", file.fscfg()))

**file.fsinfo()**

返回文件系统的大小信息。对于SPIFFS是字节型数据的，对于FatFS是kb型数据。

**使用语法：**

file.fsinfo()

**参数介绍：**

none（无）

**返回值：**

remaining：剩余大小 （数字类型）

used ：已使用空间（数字类型）

total：总空间大小 （数字类型）

**Example：**

*-- get file system info*

remaining, used, total=file.fsinfo()

print("\nFile system info:\nTotal : "..total.." (k)Bytes\nUsed : "..used.." (k)Bytes\nRemain: "..remaining.." (k)Bytes\n"·

**file.fsinfo()**

列出文件系统中的所有文件。

**使用语法：**

file.list([pattern])

**参数介绍：**

none（无）

**返回值：**

Lua语法中的table（表格），如果没有模式被给的话，包含所有的{“文件名”:”文件大小”}。如果有模式被给，仅仅返回与该模式匹配的文件名（解释为传统的Lua模式，而不是Unix shell glob），file.list()将会抛出模式匹配期间遇到的错误。

**Example：**

**l = file.list();**

**for k,v in** pairs**(l) do**

print**(**"name:"**..k..**", size:"**..v)**

**end**

**file.mount()**

在SD卡中安装一个FatFs卷。

注意：该函数只在FatFS的相关文件烧录到固件中才会被支持，并且该函数不在内部flash中被支持。

**使用语法：**

file.mount(ldrv[, pin])

**参数介绍：**

ldrv：逻辑驱动器的名称，/SD0，/SD1等等。

pin：1~12，ss/cs的IO索引，如果省略，默认为8。

**返回值：**

卷对象

**Example：**

**vol = file.mount(**"/SD0"**)**

**vol:umount()**

**file.on()**

注册回调函数。

触发事件包括：

rtc：传递当前的日期和时间给文件系统。函数希望返回一个(table)表类型，其中包括当前日期和时间的year,mon,day,hour,min,sec。不支持内部flash。

**使用语法：**

file.on(event[, function()])

**参数介绍：**

event：字符串

function()：回调函数。如果省略的话将不会注册回调函数。

**返回值：**

nil

**Example：**

sntp.sync(server\_ip,

**function**()

print("sntp time sync ok")

file.on("rtc",

**function()**

**return** rtctime.epoch2cal(rtctime.get())

**end**)

**end**)

**file.open()**

打开一个访问的文件，可能会创建一个用来写模式。

当处理完文件的时候，必须使用file.close()关闭当前文件。

**使用语法：**

file.open(filename, mode)

**参数介绍：**

filename：将要被打开的文件的文件名。

mode：

“r”：读取模式（默认）

“w”：写模式

“a”：添加模式

“r+”：更新模式，所有之前的数据被保存

“w+”：更新模式，所有之前的数据被擦除

“a+”：添加更新模式，之前的数据被保存，但是只能在文件末尾进行写操作。

**返回值：**

如果文件被打开成功则返回文件对象。如果文件没有被打开或者不存在（读取模式）则返回nil。

**Example：（基础(basic)模式）**

*-- open 'init.lua', print the first line.*

**if** file.open("init.lua", "r") **then**

print(file.readline())

file.close()

**end**

**Example：（基础(object)模式）**

*-- open 'init.lua', print the first line.*

fd = file.open("init.lua", "r")

**if** fd **then**

print(fd:readline())

fd:close(); fd = **nil**

**end**

**file.remove()**

从文件系统中移除目标文件。但是，这个文件不能被打开。

**使用语法：**

file.remove(filename)

**参数介绍：**

filename：将会被移除的文件的文件名。

**返回值：**

nil

**Example：**

*-- remove "foo.lua" from file system.*

file.remove("foo.lua")

**file.rename()**

重命名文件。如果这个文件被打开，那么他将会先被关闭。

**使用语法：**

file.rename(oldname, newname)

**参数介绍：**

oldname：原文件名

newname：新文件名

**返回值：**

更改成功则返回true，更改失败则返回失败。

**Example：**

*-- rename file 'temp.lua' to 'init.lua'.*

file.rename("temp.lua","init.lua")

**file.stat()**

获取表中文件或目录属性。表的元素包括：

size：文件的大小（返回值为字节型）

name：文件名

time：表中时间戳信息。默认为1970-01-01 00:00:00，在SPIFFS中时间戳不被允许。year-mon-day-hour-min-sec

is\_dir：如果该项目是一个目录的话标志位为true，否则为false。

is\_rdonly：如果该项目是只读文件，那么标志位为true，否则为false。

is\_hidden：如果该项目被隐藏，那么标志位为true，否则为false。

is\_sys：如果该项目是系统属性，那么标志位为true，否则为false。

is\_arch：如果该项目为存档文件，那么标志位为true，否则为false。

**使用语法：**

file.stat(filename)

**参数介绍：**

filename：目标文件名

**返回值：**

包含文件属性的表

**Example：**

s = file.stat("/SD0/myfile")

print("name: " .. s.name)

print("size: " .. s.size)

t = s.time

print(string.format("%02d:%02d:%02d", t.hour, t.min, t.sec))

print(string.format("%04d-%02d-%02d", t.year, t.mon, t.day))

**if** s.is\_dir **then** print("is directory") **else** print("is file") **end**

**if** s.is\_rdonly **then** print("is read-only") **else** print("is writable") **end**

**if** s.is\_hidden **then** print("is hidden") **else** print("is not hidden") **end**

**if** s.is\_sys **then** print("is system") **else** print("is not system") **end**

**if** s.is\_arch then print("is archive") **else** print("is not archive") **end**

s = **nil**

t = **nil**

**File access functions**

这个文件模块提供了多个函数在文件使用file.open()打开的文件后接入文件的内容。这些被用在了basic模块或者是object模块。

**Basic model：**

在basic模块中同时最多只可以打开一个文件。默认时，文件访问功能对该文件进行操作。如果另一个文件被打开，那么之前默认的文件需要在操作前被打开。

*-- open 'init.lua', print the first line.*

**if** file.open("init.lua", "r") **then**

print(file.readline())

file.close()

**end**

**Object model：**

文件被file.open()创建的文件对象表示。文件访问函数可用的作这个对象的方法，并且多个文件的对象能同时存在。

src = file.open("init.lua", "r")

**if** src **then**

dest = file.open("copy.lua", "w")

**if** dest **then**

**local** line

**repeat**

line = src:read()

**if** line **then**

dest:write(line)

**end**

**until** line == **nil**

dest:close(); dest = **nil**

**end**

src:close(); dest = **nil**

**end**

特别注意：在一个应用中建议使用单个模块。如果同时使用两个模块的话将会出现不可预知的行为：从Basic模块中将会关闭默认文件的文件对象。从Object模块中关闭一个文件（如果两个是相同的文件也将关闭默认的文件）。

注意：在SPIFFS中打开文件的最大数量在编译时被

user\_config.h文件中SPIFFS\_MAX\_OPEN\_FILES所确定。

**file.close(),file.obj:close()**

关闭打开中的文件，强制执行。

**使用语法：**

file.close()

fd:close()

**参数介绍：**

none

**返回值：**

nil

**file.flush(),file.obj:flush()**

刷新任何被挂起的写入，确保再重启时没有数据丢失。用file.close()/fd:close()关闭正在打开的文件也会执行隐式刷新。

**使用语法：**

file.flush()

fd:flush()

**参数介绍：**

none（无）

**返回值：**

nil（无）

**Example：**

*-- open 'init.lua' in 'a+' mode*

**if** file.open("init.lua", "a+") **then**

*-- write 'foo bar' to the end of the file*

file.write('foo bar')

file.flush()

*-- write 'baz' too*

file.write('baz')

file.close()

**end**

**file.read(),file.obj:read()**

在正在打开中的文件中读取文本。

注意：这个函数在堆上临时分配2\*（请求的字节数），用于缓冲和处理读取的数据。默认块大小（文件读取块）为1024字节，被认为是安全的。将此值推高4倍或者更高可能会导致堆溢出，具体取决与应用程序。选择n\_or\_char参数时需要考虑这一点。

**使用语法：**

file.read([n\_or\_char])

fd:read([n\_or\_char])

**参数介绍：**

n\_or\_char：

<1>如果没有值被传入，那么读取到达FILE\_READ\_CHUNK（块字节）（bytes字节），或者全部的文件（以最小为准）。

<2>如果传入一个数字为n，那么将读取n个字节或者整个文件（不论有多小）。

<3>如果传入了一个字符串包括单独一个字符char，那么读取直到char出现在文件中，文件块字节已经被读，或者达到了EOF。

**返回值：**

文件的内容作为一个字符串，或者当文件为EOF是为nil。

**Example：（basic model）**

*-- print the first line of 'init.lua'*

**if** file.open("init.lua", "r") **then**

print(file.read('\n'))

file.close()

**end**

file.write('baz')

**Example：（object model）**

*-- print the first 5 bytes of 'init.lua'*

fd = file.open("init.lua", "r")

**if** fd **then**

print(fd:read(5))

fd:close(); fd = n**il**

**end**

**file.readline(),file.obj:readline()**

读取已打开文件的下一行。行被定义成0或者更多的字节在结尾的时候有EOF(‘\n’)字节。如果下一行超过了1024字节，这个函数进京返回第一个1024字节。

**使用语法：**

file.readline()

fd:readline()

**参数介绍：**

none（无）

**返回值：**

文件字符串形式的内容，一行一行的，包括EOF(‘\n’)->换行符。当遇到换行符EOF(‘\n’)时返回nil。

**Example：（object model）**

*-- print the first line of 'init.lua'*

**if** file.open("init.lua", "r") **then**

print(fd:readline())

file.close()

**end**

**file.seek(),file.obj:seek()**

设置并且获取文件的位置，从文件开头的位置开始测量，该位置由偏移量加上字符串指定的基得出。

**使用语法：**

file.seek([whence [, offset]])

fd:seek([whence [, offset]])

**参数介绍：**

whence：”set”:基位置是0（文件开始的地方）

“cur”:当前的位置，（默认值）

“end”:文件末尾的位置

offset：偏移量默认为0

如果没有给函数体内传入任何的参数，这个函数就会返回当前文件的偏移量。

**返回值：**

结果文件位置，或出错时为零

**Example：（basic model）**

**if** file.open("init.lua", "r") **then**

*-- seek the first 5 bytes of 'init.lua'*

file.seek(“set”,5)

print(file.readline())

file.close()

**end**

**file.write(),file.obj:write()**

给正在打开的文件写一个字符串。

**使用语法：**

file.write(string)

fd:write(string)

**参数介绍：**

string：给文件将要写的字符串。

**返回值：**

如果文件被写成功，则返回true，反之则返回值为0（非零即为真）。

**Example：（basic model）**

*-- open 'init.lua' in 'a+' mode*

**if** file.open("init.lua", "a+") **then**

*-- write 'foo bar' to the end of the file*

file.write('foo bar')

file.close()

**end**

**Example：（object model）**

*-- open 'init.lua' in 'a+' mode*

fd = file.open("init.lua", "a+")

**if** fd **then**

*-- write 'foo bar' to the end of the file*

fd:write('foo bar')

fd:close()

**end**

**file.writeline(),file.obj:writeline()**

给一个以打开的文件写一个字符串，并且在文件的末尾处添加换行符’\n’。

**使用语法：**

file.writeline(string)

fd:writeline(string)

**参数介绍：**

string：将要给文件写的字符串。

**返回值：**

如果被写成功则返回true，否则（错误）返回nil。

**Example：（basic model）**

*-- open 'init.lua' in 'a+' mode*

**if** file.open("init.lua", "a+") **then**

*-- write 'foo bar' to the end of the file*

file.writeline('foo bar')

file.close()

**end**