NodeMCU-API 中文说明

Version 2.2.1 build 2019-08-19

**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”)))