Quectel QuecPython 类库API说明

MicroPython标准库和微库

Python基础属性

内置函数

Python 解释器内置了很多函数和类型，您可以在任何时候使用它们，以下按字母表顺序列出它们。更多信息请参阅CPython文档：Built-in Functions

		内置函数		
abs()	all()	any()	bin()	callable()
chr()	classmethod()	compile()	delattr()	dir()
divmod()	enumerate()	eval()	exec()	filter()
getattr()	globals()	hasattr()	hash()	hex()
id()	input()	isinstance()	issubclass()	iter()
len()	locals()	map()	max()	min()
next()	oct()	open()	ord()	pow()
print()	property()	range()	repr()	reversed()
round()	setattr()	sorted()	staticmothed()	sum()
super()	type()	zip()		

内置常量

有少数的常量存在与内置命名空间中，如下表所示。更多信息请参阅阅CPython文档：Built-in Constants

内置常量	
False	bool 类型的假值
True	bool 类型的真值
None	Nonetype类型的唯一值
__debug__	Python没有以 -O 选项启动，则此常量为真值
Ellipsis	与用户定义的容器数据类型的扩展切片语法结合使用
NotImplemented	二进制特殊方法应返回的特殊值（例如，**eq()**、**lt()**等）表示操作没有针对其他类型实现

内置类型

下表列出内置的数据类型，更多详情请参阅阅CPython文档：Built-in Types

内置类型	
int	整数，数值类型
float	浮点数，数值类型
complex	复数，数值类型
bool	bool，数值类型
list	列表，序列类型
tuple	元组，序列类型
range	range对象，序列类型
str	字符串，序列类型
bytes	单个字节构成的不可变序列，序列类型
bytearray	bytes对象的可变对应物，序列类型
memoryview	二进制序列
dict	字典，映射类型
set	集合
frozenset	集合，不可修改，具有哈希值
object	对象，python3.x后class默认的基类
slice	函数，切片

标准库

uos - 基本系统服务

uos模块包含文件系统访问和挂载构建，该模块实现了CPython模块相应模块的子集。更多信息请参阅阅CPython文档：os

> **uos.remove(path)**

删除文件。path表示文件名。

> **uos.chdir(path)**

改变当前目录。path表示目录名。

> **uos.getcwd()**

获取当前路径。

> **uos.listdir([dir])**

没有参数列出当前目录文件，否则列出给定目录的文件。dir为可选参数，表示目录名，默认为 '/' 目录。

示例：

```
>>> uos.listdir()
['file1', 'read.txt', 'demo.py']
```

> **uos.mkdir(path)**

创建一个新的目录。path表示准备创建的目录名。

示例：

```
>>> uos.mkdir('testdir')
>>> uos.listdir()
['file1', 'read.txt', 'demo.py', 'testdir']
```

> **uos.rename(old_path, new_path)**

重命名文件。old_path表示旧文件或目录名，new_path表示新文件或目录名。

示例：

```
>>> uos.rename('testdir', 'testdir1')
```

> **uos.rmdir(path)**

删除指定目录。path表示目录名。

示例：

```
>>> uos.rmdir('testdir')
>>> uos.listdir()
['file1', 'read.txt', 'demo.py']
```

> **uos.ilistdir([dir])**

该函数返回一个迭代器，该迭代器会生成所列出条目对应的3元组。dir为可选参数，表示目录名，没有参数时，默认列出当前目录，有参数时，则列出dir参数指定的目录。元组的形式为 `(name, type, inode[, size])`：

- name 是条目的名称，字符串类型，如果dir是字节对象，则名称为字节;
- type 是条目的类型，整型数，0x4000表示目录，0x8000表示常规文件;
- 是一个与文件的索引节点相对应的整数，对于没有这种概念的文件系统来说，可能为0;

- 一些平台可能会返回一个4元组，其中包含条目的size。对于文件条目，size表示文件大小的整数，如果未知，则为-1。对于目录项，其含义目前尚未定义。

uos.stat(path)

获取文件或目录的状态。path表示文件或目录名。返回值是一个元组，返回值形式为：

`(mode, ino, dev, nlink, uid, gid, size, atime, mtime, ctime)`

- `mode` – inode保护模式
- `ino` – inode节点号
- `dev` – inode驻留的设备
- `nlink` – inode的链接数
- `uid` – 所有者的用户ID
- `gid` – 所有者的组ID
- `size` – 文件大小，单位字节
- `atime` – 上次访问的时间
- `mtime` – 最后一次修改的时间
- `ctime` – 操作系统报告的“ctime”，在某些系统上是最新的元数据更改的时间，在其它系统上是创建时间，详细信息参见平台文档

uos.statvfs(path)

获取文件系统状态信息。path表示文件或目录名。返回一个包含文件系统信息的元组：

`(f_bsize, f_frsize, f_blocks, f_bfree, f_bavail, f_files, f_ffree, f_favail, f_flag, f_namemax)`

- `f_bsize` – 文件系统块大小，单位字节
- `f_frsize` – 分栈大小，单位字节
- `f_blocks` – 文件系统数据块总数
- `f_bfree` – 可用块数
- `f_bavai` – 非超级用户可获取的块数
- `f_files` – 文件结点总数
- `f_ffree` – 可用文件结点数
- `f_favail` – 超级用户的可用文件结点数
- `f_flag` – 挂载标记
- `f_namemax` – 最大文件长度，单位字节

示例：

```
>>> import uos
>>> res = uos.statvfs("main.py")
>>> print(res)
(4096, 4096, 256, 249, 249, 0, 0, 0, 0, 255)
```

uos.uname()

获取关于底层信息或其操作系统的信息。返回一个元组，形式为：

`(sysname, nodename, release, version, machine)`

- `sysname` – 底层系统的名称，string类型
- `nodename` – 网络名称(可以与 sysname 相同)，string类型
- `release` – 底层系统的版本，string类型
- `version` – MicroPython版本和构建日期，string类型
- `machine` – 底层硬件(如主板、CPU)的标识符，string类型

示例：

```
>>> import uos
>>> uos.uname()
(sysname='EC100Y', nodename='EC100Y', release='1.12.0', version='v1.12 on 2020-06-23', machine='EC100Y with QUECTEL')
```

> **uos.urandom(n)**

返回具有*n*个随机字节的bytes对象，只要有可能，它就会由硬件随机数生成器生成。

示例：

```
>>> import uos
>>> uos.urandom(5)
b'\xb3\xc9Y\x1b\xe9'
```

gc - 内存碎片回收

gc 模块实现内存垃圾回收机制，该模块实现了CPython模块相应模块的子集。更多信息请参阅阅CPython文档：gc

> **gc.enable()**

启用自动回收内存碎片机制。

> **gc.disable()**

禁用自动回收机制。

> **gc.collect()**

回收内存碎片。

> **gc.mem_alloc()**

返回分配的堆RAM的字节数。此功能是MicroPython扩展。

> **gc.mem_free()**

返回可用堆RAM的字节数，如果此数量未知，则返回-1。此功能是MicroPython扩展。

ubinascii - 二进制与ASCII转换

ubinascii 模块实现了二进制数据与各种ASCII编码之间的转换(双向)，该模块实现了CPython模块相应模块的子集。更多信息请参阅阅CPython文档：binascii

> **ubinascii.a2b_base64(data)**

解码base64编码的数据，会自动忽略输入中的无效字符，返回 bytes 对象。

> ubinascii.b2a_base64(data)

以base64格式编码二进制数据，返回编码数据。后面跟换行符，作为 bytes 对象。

> **ubinascii.hexlify(data, [sep])**

将二进制数据转换为十六进制字符串表示。

示例：

```
>>> import ubinascii
# 没有sep参数
>>> ubinascii.hexlify('\x11\x22123')
b'1122313233'
>>> ubinascii.hexlify('abcdfg')
b'616263646667'
# 指定了第二个参数sep，它将用于分隔两个十六进制数
>>> ubinascii.hexlify('\x11\x22123', ' ')
b'11 22 31 32 33'
>>> ubinascii.hexlify('\x11\x22123', ',')
b'11,22,31,32,33'
```

> **ubinascii.unhexlify(data)**

将十六进制形式的字符串转换成二进制形式的字符串表示。

示例：

```
>>> import ubinascii
>>> ubinascii.unhexlify('313222')
b'12"'
```

ucollections - 集合和容器类型

ucollections 模块用于创建一个新的容器类型，用于保存各种对象。该模块实现了CPython模块相应模块的子集。更多信息请参阅阅CPython文档：collections

> **mytuple = ucollections.namedtuple(name, fields)**

创建一个具有特定名称和一组字段的新namedtuple容器类型，namedtuple是元组的子类，允许通过索引来访问它的字段。

参数

参数	参数类型	参数说明
name	str	新创建容器的类型名称
fields	tuple	新创建容器类型包含子类型的字段

示例：

```
>>> import ucollections
>>> mytuple = ucollections.namedtuple("mytuple", ("id", "name"))
>>> t1 = mytuple(1, "foo")
>>> t2 = mytuple(2, "bar")
>>> print(t1.name)
foo
```

> **dq = ucollections.deque(iterable, maxlen, flag)**

创建deque双向队列

- 参数

参数	参数类型	参数说明
iterable	tuple	iterable必须是空元组
maxlen	int	指定maxlen并将双端队列限制为此最大长度
flag	int	可选参数；0(默认)：不检查队列是否溢出，达到最大长度时继续append会丢弃之前的值，1：当队列达到最大设定长度会抛出IndexError: full

- 返回值

deque对象

deque对象方法

> **dq.append(data)**

往队列中插入值。

- 参数

参数	参数类型	参数说明
data	基本数据类型	需要添加到队列的数值

- 返回值

无

> **dq.popleft()**

从deque的左侧移除并返回移除的数据。如果没有deque为空，会引起索引错误

- 参数

无

- 返回值

返回pop出的值

使用示例

```
from ucollections import deque

dq = deque((),5)
dq.append(1)
dq.append(["a"])
dq.append("a")

dq.popleft()  # 1
dq.popleft()  # ["a"]
dq.popleft()  # a
```

urandom - 生成随机数

urandom 模块提供了生成随机数的工具。

> **urandom.choice(obj)**

随机生成对象 obj 中的元素，obj 类型 string。

示例：

```
>>> import urandom
>>> urandom.choice("QuecPython")
't'
```

> **urandom.getrandbits(k)**

随机产生一个k比特长的随机整数。

示例：

```
>>> import urandom
>>> urandom.getrandbits(1)  #1位二进制位，范围为0~1（十进制：0~1）
1
>>> urandom.getrandbits(1)
0
>>> urandom.getrandbits(8)  #8位二进制位，范围为0000 0000~1111 11111（十进制：0~255）
224
```

> **urandom.randint(start, end)**

随机生成一个 start 到 end 之间的整数。

示例：

```
>>> import urandom
>>> urandom.randint(1, 4)
4
>>> urandom.randint(1, 4)
2
```

> **urandom.random()**

随机生成一个 0 到 1 之间的浮点数。

示例：

```
>>> import urandom
>>> urandom.random()
0.8465231
```

> **urandom.randrange(start, end, step)**

随机生成 start 到 end 间并且递增为 step 的正整数。

示例：

```
>>> import urandom
>>> urandom.randrange(0, 8, 2)
0
>>> urandom.randrange(0, 8, 2)
6
```

> **urandom.seed(sed)**

指定随机数种子，通常和其它随机数生成函数搭配使用。

示例：

```
>>> import urandom
>>> urandom.seed(20)  #指定随机数种子
>>> for i in range(0, 15): #生成0~15范围内的随机序列
...     print(urandom.randint(1, 10))
...
8
10
9
10
2
1
9
3
2
```

```
2
6
1
10
9
6
```

> **urandom.uniform(start, end)**

随机生成 start 到 end 范围内的浮点数。

示例：

```
>>> import urandom
>>> urandom.uniform(3, 5)
3.219261
>>> urandom.uniform(3, 5)
4.00403
```

math - 数学运算

math 模块提供数学运算函数。该模块实现相应CPython模块的子集。更多信息请参阅阅CPython文档：[math]

> **math.pow(x, y)**

返回x的y次方，返回值是浮点数。

示例：

```
>>> import math
>>> math.pow(2, 3)
8.0
```

> **math.acos(x)**

返回x的反余弦弧度值，返回值为浮点数。x是-1~1之间的数，包括-1和1，如果小于-1或者大于1，会产生错误。

示例：

```
>>> import math
>>> math.acos(0.6)
0.9272952
```

> **math.asin(x)**

返回x的反正弦弧度值，返回值为浮点数。x是-1~1之间的数，包括-1和1，如果小于-1或者大于1，会产生错误。

示例：

```
>>> import math
>>> math.asin(-1)
-1.570796
```

math.atan(x)

返回x的反正切弧度值，返回值为浮点数。

示例：

```
>>> import math
>>> math.atan(-8)
-1.446441
>>> math.atan(6.4)
1.4158
```

math.atan2(x, y)

返回给定的 X 及 Y 坐标值的反正切值，返回值为浮点数。

示例：

```
>>> import math
>>> math.atan2(-0.50,0.48)
-0.8058035
>>> math.atan2(7, 9)
0.6610432
```

math.ceil(x)

返回数字的上入整数。

示例：

```
>>> import math
>>> math.ceil(4.1)
5
```

math.copysign(x, y)

把y的正负号加到x前面，可以使用0，返回值为浮点数。

示例：

```
>>> import math
>>> math.copysign(5, 0)
5.0
>>> math.copysign(5, -4)
-5.0
>>> math.copysign(5, 9)
5.0
```

math.cos(x)

返回x的弧度的余弦值，范围再-1~1之间，返回值为浮点数。

示例：

```
>>> import math
>>> math.cos(3)
-0.9899925
```

math.degrees(x)

将弧度转换为角度，返回值为浮点数。

示例：

```
>>> import math
>>> math.degrees(5)
286.4789
>>> math.degrees(math.pi/2)
90.0
```

math.e

数学常量`e`，`e`即自然常数。

math.exp(x)

返回e的x次幂，返回值为浮点数。

示例：

```
>>> import math
>>> math.exp(1)
2.718282
>>> print(math.e)
2.718282
```

math.fabs(x)

返回数字的绝对值，返回值为浮点数。

示例：

```
>>> import math
>>> math.fabs(-3.88)
3.88
```

math.floor(x)

返回数字的下舍整数。

示例：

```
>>> import math
>>> math.floor(8.7)
8
>>> math.floor(9)
9
>>> math.floor(-7.6)
-8
```

math.fmod(x, y)

返回x/y的余数，返回值为浮点数。

示例：

```
>>> import math
>>> math.fmod(15, 4)
3.0
>>> math.fmod(15, 3)
0.0
```

math.modf(x)

返回由x的小数部分和整数部分组成的元组。

示例：

```
>>> import math
>>> math.modf(17.592)
(0.5919991, 17.0)
```

math.frexp(x)

返回一个元组(m,e),其计算方式为：x分别除0.5和1,得到一个值的范围，2e的值在这个范围内，e取符合要求的最大整数值,然后x/(2e)，得到m的值。如果x等于0，则m和e的值都为0，m的绝对值的范围为(0.5,1)之间，不包括0.5和1。

示例：

```
>>> import math
>>> math.frexp(52)
(0.8125, 6)
```

math.isfinite(x)

判断 x 是否为有限数，是则返回True，否则返回False。

示例：

```
>>> import math
>>> math.isfinite(8)
True
```

math.isinf(x)

如果x是正无穷大或负无穷大，则返回True,否则返回False。

示例：

```
>>> import math
>>> math.isinf(123)
False
```

math.isnan(x)

如果x不是数字True,否则返回False。

示例：

```
>>> import math
>>> math.isnan(23)
False
```

math.ldexp(x, exp)

返回x*(2**i)的值。

示例：

```
>>> import math
>>> math.ldexp(2, 1)
4.0
```

math.log(x)

返回x的自然对数，x > 0，小于0会报错。

示例：

```
>>> import math
>>> math.log(2)
0.6931472
```

math.pi

数学常量 pi（圆周率，一般以π来表示）。

math.radians(x)

将角度转换为弧度，返回值为浮点数。

示例：

```
>>> import math
>>> math.radians(90)
1.570796
```

math.sin(x)

返回x弧度的正弦值，数值在 -1 到 1 之间。

示例：

```
>>> import math
>>> math.sin(-18)
0.7509873
>>> math.sin(50)
-0.2623749
```

math.sqrt(x)

返回数字x的平方根，返回值为浮点数。

示例：

```
>>> import math
>>> math.sqrt(4)
2.0
>>> math.sqrt(7)
2.645751
```

math.tan(x)

返回 x 弧度的正切值，数值在 -1 到 1 之间，为浮点数。

示例：

```
>>> import math
>>> math.tan(9)
-0.4523157
```

> **math.trunc(x)**

返回x的整数部分。

示例：

```
>>> import math
>>> math.trunc(7.123)
7
```

usocket - socket模块

usocket 模块提供对BSD套接字接口的访问。该模块实现相应CPython模块的子集。更多信息请参阅阅CPython文档：socket

> **usocket.socket(af=AF_INET, type=SOCK_STREAM, proto=IPPROTO_TCP)**

根据给定的地址族、套接字类型以及协议类型参数，创建一个新的套接字。注意，在大多数情况下不需要指定*proto*，也不建议这样做，因为某些MicroPython端口可能会省略 `IPPROTO_*` 常量。

常量说明

af - 地址族

- usocket.AF_INET： IPV4
- usocket.AF_INET6： IPV6

type - socket类型

- usocket.SOCK_STREAM： 对应TCP的流式套接字
- usocket.SOCK_DGRAM： 对应UDP的数据包套接字
- usocket.SOCK_RAW： 原始套接字

proto - 协议号

- usocket.IPPROTO_TCP
- usocket.IPPROTO_UDP

其他

- usocket.SOL_SOCKET - 套接字选项级别，
- usocket.SO_REUSEADDR - 允许绑定地址快速重用

示例：

```
import usocket
# 创建基于TCP的流式套接字
socket = usocket.socket(usocket.AF_INET, usocket.SOCK_STREAM)
# 创建基于UDP的数据报套接字
socket = usocket.socket(usocket.AF_INET, usocket.SOCK_DGRAM)
```

> **usocket.getaddrinfo(host, port)**

将主机域名（host）和端口（port）转换为用于创建套接字的5元组序列，元组结构如下：

`(family, type, proto, canonname, sockaddr)`

socket类的方法

> **socket.bind(address)**

绑定地址address。在此之前，socket必须没有绑定过。

- `address`：由地址端口号组成的列表或者元组

示例：

```
addr = ('127.0.0.1', 6000)
socket.bind(addr)
```

> **socket.listen(backlog)**

允许服务端接受连接，可指定最大连接数。

- `backlog`：接受套接字的最大个数，至少为0。

> **socket.accept()**

接受连接请求，返回元组，包含新的套接字和客户端地址，形式为：`(conn, address)`

- `conn`：新的套接字对象，可以用来发送和接收数据
- `address`：连接到服务器的客户端地址

> **socket.connect(address)**

连接到指定地址address的服务器。

- `address`：包含地址和端口号的元组或列表

> **socket.read([size])**

从套接字中读取size字节数据，返回一个字节对象。如果没有指定size，则会从套接字读取所有可读数据，直到读取到数据结束，此时作用和 `socket.readall()` 相同。

> **socket.readinto(buf, [, nbytes])**

将字节读取到缓冲区buf中。如果指定了nbytes，则最多读取nbytes数量的字节；如果没有指定nbytes，则最多读取len(buf)字节。返回值是实际读取的字节数。

> **socket.readline()**

按行读取数据，遇到换行符结束，返回读取的数据行。

socket.write(buf)

写入缓冲区的数据，buf为待写入的数据，返回实际写入的字节数。

socket.send(bytes)

发送数据，返回实际发送的字节数。

- `bytes`： bytes型数据

socket.sendall(bytes)

将所有数据都发送到套接字。与`send()`方法不同的是，此方法将尝试通过依次逐块发送数据来发送所有数据。

注意：该方法再非阻塞套接字上的行为是不确定的，建议再MicroPython中，使用`write()`方法，该方法具有相同的“禁止短写”策略来阻塞套接字，并且将返回在非阻塞套接字上发送的字节数。

- `bytes`： bytes型数据

socket.sendto(bytes, address)

将数据发送到套接字。该套接字不应连接到远程套接字，因为目标套接字是由*address*指定的。

- `bytes`： bytes型数据
- `address`： 包含地址和端口号的元组或列表

socket.recv(bufsize)

从套接字接收数据。返回值是一个字节对象，表示接收到的数据。一次接收的最大数据量由bufsize指定。

- `bufsize`： 一次接收的最大数据量

socket.close()

将套接字标记为关闭并释放所有资源。

socket.recvfrom(bufsize)

从套接字接收数据。返回一个元组，包含字节对象和地址。

返回值形式为：`(bytes, address)`

- bytes： 接收数据的字节对象
- address： 发送数据的套接字的地址

socket.setsockopt(level, optname, value)

设置套接字选项的值。

- `level`：套接字选项级别
- `optname`：socket选项
- `value`：既可以是一个整数，也可以是一个表示缓冲区的bytes类对象

示例：

```
socket.setsockopt(usocket.SOL_SOCKET, usocket.SO_REUSEADDR, 1)
```

> **socket.setblocking(flag)**

设置套接字为阻塞模式或者非阻塞模式。如果标志为false，则将套接字设置为非阻塞，否则设置为阻塞模式。

该方法是某些 `settimeout()` 调用的简写：

`socket.setblocking(True)` 相当于 `socket.settimeout(None)`

`socket.setblocking(False)` 相当于 `socket.settimeout(0)`

> **socket.settimeout(value)**

设置套接字的超时时间，单位秒。

- `value`：可以是表示秒的非负浮点数，也可以是None。如果给出一个非零值，则 `OSError` 在该操作完成之前已超过超时时间值，则随后的套接字操作将引发异常。如果给定零，则将套接字置于非阻塞模式。如果未指定，则套接字将处于阻塞模式。

> **socket.makefile(mode='rb')**

返回与套接字关联的文件对象，返回值类型与指定的参数有关。仅支持二进制模式 (rb和wb)。

socket通信示例：

```
# 客户端示例
import usocket
client = usocket.socket(usocket.AF_INET, usocket.SOCK_STREAM)
sockaddr = usocket.getaddrinfo('www.tongxinmao.com',80)[0][-1]
client.connect(sockaddr)
while True:
    re_data = input()
    client.send(re_data.encode("utf8"))
    data = client.recv(1024)
    print(data.decode("utf8"))
```

uio - 输入输出流

uio 模块包含其他类型的stream（类文件）对象和辅助函数。该模块实现相应CPython模块的子集。更多信息请参阅阅CPython文档：io

> **fd = uio.open(name, mode='r', **kwarg)**

打开文件，内置 `open()` 函数是该函数的别名。

- `name` ： 文件名
- `mode` ： 打开模式
- r 只读模式打开文件
  - w 写入模式打开文件，每次写入会覆盖上次写入数据
  - a 只写追加模式打开文件，可连续写入文件数据而不是覆盖数据
- `**kwarg`：可变长参数列表

> **fd.close()**

关闭打开的文件。

ustruct - 打包和解压原始数据类型

该模块实现相应CPython模块的子集。更多信息请参阅阅CPython文档：struct

字节顺序，大小和对齐方式

默认情况下，C类型以机器的本机格式和字节顺序表示，并在必要时通过跳过填充字节来正确对齐（根据C编译器使用的规则）。根据下表，格式字符串的第一个字符可用于指示打包数据的字节顺序，大小和对齐方式：

Character	Byte order	Size	Alignment
`@`	native	native	native
`=`	native	standard	none
`<`	little-endian	standard	none
`>`	big-endian	standard	none
`!`	network (= big-endian)	standard	none

格式化字符表

Format	C Type	Python type	Standard size
`x`	pad byte	no value	
`c`	`char`	bytes of length 1	1
`b`	`signed char`	integer	1
`B`	`unsigned char`	integer	1
`?`	`_Bool`	bool	1
`h`	`short`	integer	2
`H`	`unsigned short`	integer	2
`i`	`int`	integer	4
`I`	`unsigned int`	integer	4
`l`	`long`	integer	4
`L`	`unsigned long`	integer	4
`q`	`long long`	integer	8
`Q`	`unsigned long long`	integer	8
`n`	`ssize_t`	integer	
`N`	`size_t`	integer	
`f`	`float`	float	4
`d`	`double`	float	8
`s`	`char[]`	bytes	
`p`	`char[]`	bytes	
`P`	`void *`	integer	

默认情况下，C类型以机器的本机格式和字节顺序表示，并在必要时通过跳过填充字节来正确对齐（根据C编译器使用的规则）

> **ustruct.calcsize(fmt)**

返回存放 fmt 需要的字节数。

- `fmt`：格式字符的类型，详情见上文格化式字符表

示例：

```
>>> import ustruct
>>> ustruct.calcsize('i')
4
>>> ustruct.calcsize('f')
4
>>> ustruct.calcsize('d')
8
```

> **ustruct.pack(fmt, v1, v2, ...)**

按照格式字符串 fmt 压缩参数v1、 v2、...返回值是参数编码后的字节对象。

- `fmt`： 格式字符的类型，详情见上文格化式字符表

> **unstrcut.unpack(fmt, data)**

根据格式化字符串 fmt 对数据进行解压，返回值为一个元组。

示例：

```
>>> import ustruct
>>> ustruct.pack('ii', 7, 9)  #打包2两个整数
b'\x07\x00\x00\x00\t\x00\x00\x00'
>>> ustruct.unpack('ii', b'\x07\x00\x00\x00\t\x00\x00\x00')  #解压两个整数
(7, 9)
```

> **ustruct.pack_info(fmt, buffer, offset, v1, v2, ...)**

根据格式字符串fmt将值v1、v2、 ...打包到从`offset`开始的缓冲区中。从缓冲区的末尾算起，`offset`可能为负。

- `fmt`： 格式字符的类型，详情见上文格化式字符表

> **unstruct.unpack_from(fmt, data, offset=0)**

根据格式化字符串 `fmt` 解析从 `offest` 开始的数据解压，从缓冲区末尾开始计数的偏移量可能为负值。返回值是解压值的元组。

ujson - JSON编码和解码

ujson 模块实现在Python数据对象和JSON数据格式之间进行转换的功能。该模块实现相应CPython模块的子集。更多信息请参阅阅CPython文档：json

> **ujson.dump(obj, stream)**

将 `obj` 数据对象转化成 JSON字符串，将其写入到给定的 `stream` 中。

> **ujson.dumps(dict)**

将 `dict` 类型的数据转换成str。

> **ujson.load(stream)**

解析给定的数据 `stream`，将其解释为JSON字符串并反序列化成Python对象。

> **ujson.loads(str)**

解析JSON字符串并返回 `obj` 对象

示例：

```
>>> import ujson
>>> msg = ['foo',{'bar':('baz',None,1,2)}]
>>> print(msg)
['foo', {'bar': ('baz', None, 1, 2)}]
>>> s = ujson.dumps(msg)
>>> print(s)
["foo", {"bar": ["baz", null, 1, 2]}]
>>> s1 = ujson.loads(s)
>>> print(s1)
['foo', {'bar': ['baz', None, 1, 2]}]
```

utime - 与时间相关功能

utime 模块用于获取当前时间和日期、测量时间间隔和延迟。该模块实现相应CPython模块的子集。更多信息请参阅阅CPython文档：time

> **utime.localtime([secs])**

该函数用来将一个以秒表示的时间转换为一个元组，元组包含了了年、月、日、时、分、秒、星期、一年中第几天；如果没有给定参数sec，则使用RTC时间。返回值形式如下：

`(year, month, mday, hour, minute, second, weekday, yearday)`

- `year`：年份，int型
- `month`：月份，1~12，int型
- `mday`：日，当月多少号，1~31，int型
- `hour`：小时，0~23，int型
- `minute`：分钟，0~59，int型
- `second`：秒，0~59，int型
- `weekday`：星期，周一到周日是0~6，int型
- `yearday`：一年中的第多少天，int型

示例：

```
>>> import utime
>>> utime.localtime()
(2020, 9, 29, 8, 54, 42, 1, 273)
>>> utime.localtime(646898736)
(2020, 7, 1, 6, 5, 36, 2, 183)
```

utime.mktime(date)

该函数作用与locatime()相反，它将一个存放在元组中的时间转换为以秒计的时间戳。

示例：

```
>>> import utime
>>> date = (2020, 9, 29, 8, 54, 42, 1, 273)
>>> utime.mktime(date)
1601340882
```

utime.sleep(seconds)

休眠给定秒数的时间。

注意：sleep()函数的调用会导致程序休眠阻塞。

utime.sleep_ms(ms)

休眠给定毫秒数的时间。

注意：sleep_ms()函数的调用会导致程序休眠阻塞。

utime.sleep_us(us)

休眠给定微秒的时间。

注意：sleep_us()函数的调用会导致程序休眠阻塞。

utime.ticks_ms()

返回不断递增的毫秒计数器，在某些值后会重新计数(未指定)。计数值本身无特定意义，只适合用在`ticks_diff()`函数中。

注意：sleep_us()函数的调用会导致程序休眠阻塞。

utime.ticks_us()

和`ticks_ms()`类似，只是返回微秒计数器。

utime.ticks_cpu()

和 ticks_ms/ticks_us 类似，具有更高精度 (使用 CPU 时钟)。

utime.ticks_diff(old, new)

计算两次调用`ticks_ms()`， `ticks_us()`，或`ticks_cpu()`之间的时间。因为这些函数的计数值可能会回绕，所以不能直接相减，需要使用 ticks_diff() 函数。“旧” 时间需要在 “新” 时间之前，否则结果无法确定。这个函数不要用在计算很长的时间 (因为 ticks_*() 函数会回绕，通常周期不是很长)。通常用法是在带超时的轮询事件中调用。

示例：

```
import utime
start = utime.ticks_us()
while pin.value() == 0:
    if utime.ticks_diff(time.ticks_us(), start) > 500:
        raise TimeoutError
```

> **utime.time()**

返回自纪元以来的秒数（以整数形式）。如果未设置RTC，则此函数返回自特定于端口的参考时间点以来的秒数（对于不具有电池后备RTC的嵌入式板，通常是由于加电或复位）。如果要开发可移植的MicroPython应用程序，则不应依赖此功能提供高于秒的精度。如果需要更高的精度，请使用`ticks_ms()`和`ticks_us()`函数，如果需要日历时间，则`localtime()`不带参数会更好。

sys - 系统相关功能

sys 模块中提供了与QuecPython运行环境有关的函数和变量。该模块实现相应CPython模块的子集。更多信息请参阅阅CPython文档：sys

常数说明

> **sys.argv**

当前程序启动的可变参数列表。

> **sys.byteorder**

字节顺序 ('little' - 小端， 'big' - 大端)。

> **sys.implementation**

返回当前microPython版本信息。对于MicroPython，它具有以下属性：

- name - 字符串" micropython"
- version - 元组（主要，次要，微型），例如（1、7、0）

建议使用此对象来将MicroPython与其他Python实现区分开。

> **sys.maxsize**

本机整数类型可以在当前平台上保留的最大值，如果它小于平台最大值，则为MicroPython整数类型表示的最大值（对于不支持长整型的MicroPython端口就是这种情况）。

> **sys.modules**

已载入模块的字典。

> **sys.platform**

MicroPython运行的平台。

> **sys.stdin**

标准输入（默认是USB虚拟串口，可选其他串口）。

> **sys.stdout**

标准输出（默认是USB虚拟串口，可选其他串口）。

> **sys.version**

MicroPython 语言版本，字符串格式。

> **sys.version_info**

MicroPython 语言版本，整数元组格式。

方法

> **sys.exit(retval=0)**

使用给定的参数退出当前程序。与此同时，该函数会引发`SystemExit`退出。如果给定了参数，则将其值作为参数赋值给`SystemExit`。

> **sys.print_exception(exc, file=sys.stdout)**

打印异常到文件对象，默认是 sys.stdout，即输出异常信息的标准输出。

uzlib - zlib解压缩

uzlib 模块解压缩用DEFLATE算法压缩的二进制数据 （通常在zlib库和gzip存档器中使用），压缩尚未实现。该模块实现相应CPython模块的子集。更多信息请参阅阅CPython文档：zlib

注意：解压缩前，应检查模块内可使用的空间，确保有足够空间解压文件。

> **uzlib.decompress(data, wbits=0, bufsize=0)**

返回解压后的 bytes 对象。`wbits`是解压时使用的DEFLATE字典窗口大小（8-15，字典大小是`wbits`值的2的幂）。如果该值为正，则假定`data`为zlib流（带有zlib标头），如果为负，则假定为原始的DEFLATE流。`bufsize`参数是为了与CPython兼容，将被忽略。

> **class uzlib.DecompIO(stream, wbits=0)**

创建一个`stream`装饰器，该装饰器允许在另一个流中透明地压缩数据。这允许处理数据大于可用堆大小的压缩流。wbits的值除了上面所述的值以外，还可以取值24..31（16 + 8..15），这表示输入流具有gzip标头。

_thread - 多线程

_thread 模块提供创建新线程的方法，并提供互斥锁。

> **_thread.get_ident()**

获取当前线程号。

> **_thread.get_heap_size()**

获取系统剩余内存大小。

> **_thread.stack_size(size)**

设置创建新线程使用的栈大小（以字节为单位），默认为8k。

> **_thread.start_new_thread(function, args)**

创建一个新线程，接收执行函数和被执行函数参数。

> **_thread.allocate_lock()**

创建一个互斥锁对象。

示例：

```
import _thread
lock = _thread.allocate_lock()
```

> **lock.acquire()**

获取锁，成功返回True，否则返回False。

> **lock.release()**

释放锁。

> **lock.locked()**

返回锁的状态，True表示被某个线程获取，False则表示没有。

_thread使用示例

```
import _thread
a = 0
lock = _thread.allocate_lock()
def th_func(delay, id):
    global a
    while True:
        lock.acquire()  # 获取锁
        if a >= 10:
            print('thread %d exit' % id)
            lock.release() # 释放锁
            break
        a+=1
        print('[thread %d] a is %d' % (id, a))
        lock.release()
for i in range(2):
    _thread.start_new_thread(th_func, (i + 1, i))
```

uhashlib - 哈希算法

模块功能: 实现二进制数据散列算法,目前支持sha256，sha1，MD5。

> **hash_obj = uhashlib.sha256(bytes)**

创建一个SHA256哈希对象

- 参数

参数	参数类型	参数说明
bytes	bytes	可选参数，可在创建时传入bytes数据，也可通过update方法

- 返回值

SHA256哈希对象

> **hash_obj = uhashlib.sha1(bytes)**

创建一个SHA1哈希对象

- 参数

参数	参数类型	参数说明
bytes	bytes	可选参数，可在创建时传入bytes数据，也可通过update方法

- 返回值

SHA1哈希对象

> **hash_obj = uhashlib.md5(bytes)**

创建一个MD5哈希对象

- 参数

参数	参数类型	参数说明
bytes	bytes	可选参数，可在创建时传入bytes数据，也可通过update方法

- 返回值

MD5哈希对象

哈希对象方法

> **hash_obj .update(bytes)**

将更多的bytes数据加到散列

- 参数

参数	参数类型	参数说明
bytes	bytes	需要被加密的数据

- 返回值

无

> hash_obj .digest()

返回通过哈希传递的所有数据的散列，数据为字节类型。调用此方法后，无法再将更多的数据送入散列。

- 参数

无

- 返回值

返回加密后字节类型的数据

使用实例

```
import uhashlib
import ubinascii

hash_obj  = uhashlib.sha256()  # 创建hash对象
hash_obj.update(b"QuecPython")
res = hash_obj.digest()
#
b"\x1e\xc6gq\xb3\xa9\xac>\xa4\xc4O\x00\x9eTW\x97\xd4.\x9e}Bo\xff\x82u\x89Th\xfe'
\xc6\xcd"
# 转成十六进制表示
hex_msg = ubinascii.hexlify(res)
# b'1ec66771b3a9ac3ea4c44f009e545797d42e9e7d426fff8275895468fe27c6cd'
```

QuecPython类库

example - 执行python脚本

模块功能：提供方法让用户可以在命令行或者代码中执行python脚本。

> example.exec(filename)

执行指定的python脚本文件。

- 参数

参数	参数类型	参数说明
filename	string	要执行python脚本的文件名

- 返回值

无

- 示例

```
# 假设有文件test.py,内容如下

def myprint():
    count = 10
    while count > 0:
        count -= 1
        print('##### test #####')

myprint()

#将test.py文件上传到模块中，进入命令行执行如下代码
>>> uos.listdir()
['apn_cfg.json', 'test.py']
>>> import example
>>> example.exec('test.py')
# 执行结果如下

##### test #####
##### test #####
##### test #####
##### test #####
##### test #####
##### test #####
##### test #####
##### test #####
##### test #####
##### test #####
```

dataCall - 数据拨号

模块功能：提供数据拨号相关接口。

> **dataCall.start(profileIdx, ipType, apn, username, password, authType)**

启动拨号，进行数据链路激活。

- 参数

参数	参数类型	参数说明
profileIdx	int	PDP索引，取值1-8，一般设置为1，设置其他值可能需要专用apn与密码才能设置成功
ipType	int	IP类型，0-IPV4，1-IPV6，2-IPV4和IPV6
apn	string	apn名称，可为空
username	string	apn用户名，可为空
password	string	apn密码，可为空
authType	int	加密方式，0-不加密，1-PAP，2-CHAP，3-PAP或CHAP

- 返回值

成功返回整型值0，失败返回整型值-1。

- 示例

```
>>> import dataCall
>>> dataCall.start(1, 0, "3gnet.mnc001.mcc460.gprs", "", "", 0)
0
```

> **dataCall.setApn(profileIdx, ipType, apn, username, password, authType)**

用户apn信息配置接口，用户调用该接口后，会在用户分区目录下创建user_apn.json文件，用于保存用户apn信息，并使用该apn信息启动拨号，进行数据链路激活。

- 参数

参数	参数类型	参数说明
profileIdx	int	PDP索引，取值1-8，一般设置为1，设置其他值可能需要专用apn与密码才能设置成功
ipType	int	IP类型，0-IPV4，1-IPV6，2-IPV4和IPV6
apn	string	apn名称，可为空
username	string	apn用户名，可为空
password	string	apn密码，可为空
authType	int	加密方式，0-不加密，1-PAP，2-CHAP，3-PAP或CHAP

- 返回值

成功返回整型值0，失败返回整型值-1。

- 示例

```
>>> import dataCall
>>> dataCall.setApn(1, 0, "3gnet.mnc001.mcc460.gprs", "", "", 0)
0
```

dataCall.setCallback(usrFun)

注册用户回调函数，当网络状态发生变化，比如断线、上线时，会通过该回调函数通知用户。

- 参数

参数	参数类型	参数说明
usrFun	function	用户回调函数，函数形式见示例

- 返回值

注册失败返回整型-1，成功返回整型0。

- 示例

```
>>> import dataCall
>>> import net

>>> def nw_cb(args):
        pdp = args[0]
        nw_sta = args[1]
        if nw_sta == 1:
            print("*** network %d connected! ***" % pdp)
        else:
            print("*** network %d not connected! ***" % pdp)

>>> dataCall.setCallback(nw_cb)
0
>>> net.setModemFun(4)  # 进入飞行模式
0
>>> *** network 1 not connected! *** # 进入飞行模式导致断网，通过回调告知用户
>>> net.setModemFun(1)  # 退出飞行模式
0
>>> *** network 1 connected! *** # 退出飞行模式，自动拨号，等待联网成功，通过回调告知用户
```

dataCall.getInfo(profileIdx, ipType)

获取数据拨号信息，包括连接状态、IP地址、DNS等。

- 参数

参数	参数类型	参数说明
profileIdx	int	PDP索引，取值1-8
ipType	int	IP类型，0-IPV4，1-IPV6，2-IPV4和IPV6

- 返回值

错误返回整型-1，成功返回拨号信息，返回格式根据ipType的不同而有所区别：
ipType =0，返回值格式如下：

`(profileIdx, ipType, [nwState, reconnect, ipv4Addr, priDns, secDns])`

`profileIdx`：PDP索引，取值1-8

`ipType`：IP类型，0-IPV4，1-IPV6，2-IPV4和IPV6

`nwState`：拨号结果，0-失败，1-成功

`reconnect`：重拨标志

`ipv4Addr`：ipv4地址

`priDns`：dns信息

`secDns`：dns信息

ipType =1，返回值格式如下：

`(profileIdx, ipType, [nwState, reconnect, ipv6Addr, priDns, secDns])`

`profileIdx`：PDP索引，取值1-8

`ipType`：IP类型，0-IPV4，1-IPV6，2-IPV4和IPV6

`nwState`：拨号结果，0-失败，1-成功

`reconnect`：重拨标志

`ipv6Addr`：ipv6地址

`priDns`：dns信息

`secDns`：dns信息

ipType =2，返回值格式如下：

`(profileIdx, ipType, [nwState, reconnect, ipv4Addr, priDns, secDns], [nwState, reconnect, ipv6Addr, priDns, secDns])`

- 示例

```
>>> import dataCall
>>> dataCall.getInfo(1, 0)
(1, 0, [1, 0, '10.91.44.177', '58.242.2.2', '218.104.78.2'])
```

注：返回值`(1, 0, [0, 0, '0.0.0.0', '0.0.0.0', '0.0.0.0'])`表示当前没有拨号或者拨号没有成功。

cellLocator - 基站定位

模块功能：提供基站定位接口，获取坐标信息。

> **cellLocator.getLocation(serverAddr, port, token, timeout, profileID)**

获取基站坐标信息。

- 参数

参数	参数类型	参数说明
serverAddr	string	服务器域名，长度必须小于255 bytes，目前仅支持 “www.queclocator.com”
port	int	服务器端口，目前仅支持 80 端口
token	string	密钥，16位字符组成，需要申请
timeout	int	设置超时时间，范围1-300s，默认300s
profileID	int	PDP索引，范围1-8

- 返回值

功返回度格式经纬度坐标信息，返回格式：`(latitude, longtitude, accuracy)`，`(0.0, 0.0, 0)`表示未获取到有效坐标信息；失败返回错误码说明如下：

-1 – 初始化失败

-2 – 服务器地址过长（超过255字节）

-3 – 密钥长度错误，必须为16字节

-4 – 超时时长超出范围，支持的范围（1~300）s

-5 – 指定的PDP网络未连接，请确认PDP是否正确

-6 – 获取坐标出错

- 示例

```
>>> import cellLocator
>>> cellLocator.getLocation("www.queclocator.com", 80, "1111111122222222", 8, 1)
(117.1138, 31.82279, 550)
# 上面使用的密钥仅为测试密钥
```

sim - SIM卡

模块功能：提供sim卡操作相关API，如查询sim卡状态、iccid、imsi等。

注意：能成功获取IMSI、ICCID、电话号码的前提是SIM卡状态为1，可通过sim.getStatus()查询。

sim.getImsi()

获取sim卡的imsi。

- 参数

无

- 返回值

成功返回string类型的imsi，失败返回整型-1。

- 示例

```
>>> import sim
>>> sim.getImsi()
'460185466870381'
```

sim.getIccid()

获取sim卡的iccid。

- 参数

无

- 返回值

成功返回string类型的iccid，失败返回整型-1。

- 示例

```
>>> sim.getIccid()
'89860390845513443049'
```

sim.getPhoneNumber()

获取sim卡的电话号码。

- 参数

无

- 返回值

成功返回string类型的phone number，失败返回整型-1。

- 示例

```
>>> sim.getPhoneNumber()
'+8618166328752'
```

sim.getStatus()

获取sim卡的状态。

- 参数

无

- 返回值

返回值	说明
0	SIM was removed.
1	SIM is ready.
2	Expecting the universal PIN./SIM is locked, waiting for a CHV1 password.
3	Expecting code to unblock the universal PIN./SIM is blocked, CHV1 unblocking password is required.
4	SIM is locked due to a SIM/USIM personalization check failure.
5	SIM is blocked due to an incorrect PCK; an MEP unblocking password is required.
6	Expecting key for hidden phone book entries.
7	Expecting code to unblock the hidden key.
8	SIM is locked; waiting for a CHV2 password.
9	SIM is blocked; CHV2 unblocking password is required.
10	SIM is locked due to a network personalization check failure.
11	SIM is blocked due to an incorrect NCK; an MEP unblocking password is required.
12	SIM is locked due to a network subset personalization check failure.
13	SIM is blocked due to an incorrect NSCK; an MEP unblocking password is required.
14	SIM is locked due to a service provider personalization check failure.
15	SIM is blocked due to an incorrect SPCK; an MEP unblocking password is required.
16	SIM is locked due to a corporate personalization check failure.
17	SIM is blocked due to an incorrect CCK; an MEP unblocking password is required.
18	SIM is being initialized; waiting for completion.
19	Use of CHV1/CHV2/universal PIN/code to unblock the CHV1/code to unblock the CHV2/code to unblock the universal PIN/ is blocked.
20	Unknow status.

sim.enablePin(pin)

启用sim卡PIN码验证，开启后需要输入正确的PIN验证成功后，sim卡才能正常使用。只有3次输入PIN码机会，3次都错误，sim卡被锁定，需要PUK来解锁。

- 参数

参数	参数类型	参数说明
pin	string	PIN码，一般默认是'1234'

- 返回值

成功返回整型0，失败返回整型-1。

- 示例

```
>>> sim.enablePin("1234")
0
```

sim.disablePin(pin)

关闭sim卡PIN码验证。

- 参数

参数	参数类型	参数说明
pin	string	PIN码，一般默认是'1234'

- 返回值

成功返回整型0，失败返回整型-1。

- 示例

```
>>> sim.disablePin("1234")
0
```

sim.verifyPin(pin)

sim卡PIN码验证。需要在调用sim.enablePin(pin)成功之后，才能进行验证，验证成功后，sim卡才能正常使用。

- 参数

参数	参数类型	参数说明
pin	string	PIN码，一般默认是'1234'

- 返回值

验证成功返回整型0，验证失败返回整型-1。

- 示例

```
>>> sim.verifyPin("1234")
0
```

sim.unblockPin(puk, newPin)

sim卡解锁。当多次错误输入 PIN/PIN2 码后，SIM 卡状态为请求 PUK/PUK2 时，输入 PUK/PUK2 码和新的 PIN/PIN2 码进行解锁，puk码输入10次错误，SIM卡将被永久锁定自动报废。

- 参数

参数	参数类型	参数说明
puk	string	PUK码，长度8位数字
newPin	string	新PIN码

- 返回值

解锁成功返回整型0，解锁失败返回整型-1。

- 示例

```
>>> sim.unblockPin("12345678", "0000")
0
```

sim.changePin(oldPin, newPin)

更改sim卡PIN码。

- 参数

参数	参数类型	参数说明
oldPin	string	旧的PIN码
newPin	string	新的PIN码

- 返回值

更改成功返回整型0，更改失败返回整型-1。

- 示例

```
>>> sim.changePin("1234", "4321")
0
```

sim.readPhonebook(storage, start, end, username)

获取 SIM 卡上指定电话本中的一条或多条电话号码记录。

- 参数

参数	参数类型	参数说明
storage	int	需要读取电话号码记录的电话本存储位置，可选参数如下： 0 – DC，1 – EN，2 – FD，3 – LD，4 – MC，5 – ME，6 – MT，7 – ON， 8 – RC，9 – SM，10 – AP，11 – MBDN，12 – MN，13 – SDN，14 – ICI，15 - OCI
start	int	需要读取电话号码记录的起始编号，start为 0 表示不使用编号获取电话号码记
end	int	需要读取电话号码记录的结束编号
username	string	当 start为 0 时有效，电话号码中的用户名

- 返回值

读取失败返回整型-1，成功返回一个元组，包含读取记录，格式如下：

`(record_number, [(index, username, phone_number), ... , (index, username, phone_number)])`

返回值参数说明：

`record_number` – 读取的记录数量，整型

`index` – 在电话簿中的索引位置，整型

`username` – 姓名，string类型

`phone_number` – 电话号码，string类型

- 示例

```
>>> sim.readPhonebook(9, 1, 3, "")
(3,[(1,'zhangsan','15544272539'),(2,'lisi','15544272539'),
(3,'wangwu','18144786859')])
```

> **sim. writePhonebook(storage, index, username, number)**

写入一条电话号码记录。

- 参数

参数	参数类型	参数说明
storage	int	需要读取电话号码记录的电话本存储位置，可选参数如下： 0 – DC，1 – EN，2 – FD，3 – LD，4 – MC，5 – ME，6 – MT，7 – ON， 8 – RC，9 – SM，10 – AP，11 – MBDN，12 – MN，13 – SDN，14 – ICI，15 - OCI
index	int	需要写入电话号码记录的在电话簿中的编号
username	string	电话号码的用户名
number	string	电话号码

- 返回值

写入成功返回整型0，写入失败返回整型-1。

net - 网络相关功能

模块功能：该模块提供配置和查询网络模式信息等接口。

> **net.csqQueryPoll()**

获取csq信号强度。

- 参数

无

- 返回值

成功返回整型的csq信号强度值，失败返回整型值-1，返回值为99表示异常；

信号强度值范围0~31，值越大表示信号强度越好。

- 示例

```
>>> import net
>>> net.csqQueryPoll()
31
```

> **net.getCellInfo()**

获取邻近 CELL 的信息。

- 参数

无

- 返回值

失败返回整型值-1，成功返回包含三种网络系统（GSM、UMTS、LTE）的信息的list，如果对应网络系统信息为空，则返回空的List。返回值格式如下：

`([(flag, cid, mcc, mnc, lac, arfcn, bsic, rssi)], [(flag, cid, licd, mcc, mnc, lac, arfcn, bsic, rssi)], [(flag, cid, mcc, mnc, pci, tac, earfcn, rssi)])`

GSM网络系统返回值说明

参数	参数意义
flag	返回 0 - 2， 0：present，1：inter，2：intra
cid	返回cid信息，0则为空
mcc	移动设备国家代码
mnc	移动设备网络代码
lac	位置区码
arfcn	无线频道编号
bsic	基站识别码
rssi	接收的信号强度

UMTS网络系统返回值说明

参数	参数意义
flag	返回 0 - 2， 0：present，1：inter，2：intra
cid	返回cid信息，0则为空
licd	区域标识号
mcc	移动设备国家代码
mnc	移动设备网络代码
lac	位置区码
arfcn	无线频道编号
bsic	基站识别码
rssi	接收的信号强度

LTE网络系统返回值说明

参数	参数意义
flag	返回 0 - 2， 0：present，1：inter，2：intra
cid	返回cid信息，0则为空
mcc	移动设备国家代码
mnc	移动设备网络代码
pci	小区标识
tac	Tracing area code
earfcn	无线频道编号 范围: 0 - 65535
rssi	接收的信号强度

- 示例

```
>>> net.getCellInfo()
([], [], [(0, 14071232, 1120, 0, 21771, 123, 1300)])
```

net.getConfig()

获取当前网络模式、漫游配置。

- 参数

无

- 返回值

失败返回整型值-1，成功返回一个元组，包含当前首选的网络制式与漫游打开状态。

网络制式

值	网络制式
0	GSM
1	UMTS . not supported in EC100Y
2	GSM_UMTS, auto. not supported in EC100Y and EC200S
3	GSM_UMTS, GSM preferred. not supported in EC100Y and EC200S
4	SM_UMTS, UMTS preferred. not supported in EC100Y and EC200S
5	LTE
6	GSM_LTE, auto, single link
7	GSM_LTE, GSM preferred, single link
8	GSM_LTE, LTE preferred, single link
9	UMTS_LTE, auto, single link. not supported in EC100Y and EC200S
10	UMTS_LTE, UMTS preferred, single link. not supported in EC100Y and EC200S
11	UMTS_LTE, LTE preferred, single link . not supported in EC100Y and EC200S
12	GSM_UMTS_LTE, auto, single link. not supported in EC100Y and EC200S
13	GSM_UMTS_LTE, GSM preferred, single link. not supported in EC100Y and EC200S
14	GSM_UMTS_LTE, UMTS preferred, single link. not supported in EC100Y and EC200S
15	GSM_UMTS_LTE, LTE preferred, single link. not supported in EC100Y and EC200S
16	GSM_LTE, dual link
17	UMTS_LTE, dual link. not supported in EC100Y and EC200S
18	GSM_UMTS_LTE, dual link. not supported in EC100Y and EC200S

- 示例

```
>>>net.getConfig ()
(8, False)
```

net.setConfig(mode, roaming)

设置网络模式、漫游配置。

- 参数

参数	参数类型	参数说明
mode	int	网络制式(详见上图)
roaming	int	漫游开关(0：关闭， 1：开启)

- 返回值

设置成功返回整型值0，设置失败返回整型值-1。

net.getNetMode()

获取网络配置模式。

- 参数

无

- 返回值

失败返回整型值-1，成功返回一个元组，格式为：`(selection_mode, mcc, mnc, act)`

返回值参数说明：
`selection_mode`：方式，0 - 自动，1 - 手动
`mcc`：移动设备国家代码
`mnc`：移动设备网络代码
`act`：首选网络的ACT模式

ACT模式

值	ACT模式
0	GSM
1	COMPACT
2	UTRAN
3	GSM wEGPRS
4	UTRAN wHSDPA
5	UTRAN wHSUPA
6	UTRAN wHSDPA HSUPA
7	E UTRAN
8	UTRAN HSPAP
9	E TRAN A
10	NONE

- 示例

```
>>> net.getNetMode()
(0, '460', '46', 7)
```

net.getSignal()

获取详细信号强度。

- 参数

无

- 返回值

失败返回整型值-1，成功返回一个元组，包含两个List(GW 、LTE)，返回值格式如下：

`([rssi, bitErrorRate, rscp, ecno], [rssi, rsrp, rsrq, cqi])`

返回值参数说明：

GW list：

`rssi`：接收的信号强度

`bitErrorRate`：误码率

`rscp`：接收信号码功率

`ecno`：导频信道

LTE list：

`rssi`：接收的信号强度

`rsrp`：下行参考信号的接收功率

`rsrq`：下行特定小区参考信号的接收质量

`cqi`：信道质量

- 示例

```
>>>net.getSignal()
([99, 99, 255, 255], [-51, -76, -5, 255])
```

net.nitzTime()

获取当前基站时间。

- 参数

无

- 返回值

失败返回整型值-1，成功返回一个元组，包含基站时间与对应时间戳与闰秒数（0表示不可用），格式为：

`(date, abs_time, leap_sec)`

`date`：基站时间，string类型

`abs_time`：基站时间的绝对秒数表示，整型

`leap_sec`：闰秒数，整型

- 示例

```
>>> net.nitzTime()
('20/11/26 02:13:25 +8 0', 1606356805, 0)
```

net.operatorName()

获取当前注网的运营商信息。

- 参数

无

- 返回值

失败返回整型值-1，成功返回一个元组，包含注网的运营商信息，格式为：

`(long_eons, short_eons, mcc, mnc)`

`long_eons`：运营商信息全称，string类型

`short_eons`：运营商信息简称，string类型

`mcc`：移动设备国家代码，string类型

`mnc`：移动设备网络代码，string类型

- 示例

```
>>> net.operatorName()
('CHN-UNICOM', 'UNICOM', '460', '01')
```

net.getState()

获取当前网络注册信息。

失败返回整型值-1，成功返回一个元组，包含注网的网络注册信息，格式为：

`([voice_state, voice_lac, voice_cid, voice_rat, voice_reject_cause, voice_psc], [data_state, data _lac, data _cid, data _rat, data _reject_cause, data _psc])`

返回值参数说明：

`state`：网络注册状态

`lac`：位置区码

`cid`：int类型id信息

`act`：注网制式

`reject_cause`：注册被拒绝的原因

`psc`：Primary Scrambling Code

网络注册状态

值	状态说明
0	not registered, MT is not currently searching an operator to register to
1	registered, home network
2	not registered, but MT is currently trying to attach or searching an operator to register to
3	registration denied
4	unknown
5	registered, roaming
6	egistered for “SMS only”, home network (not applicable)
7	registered for “SMS only”, roaming (not applicable)
8	attached for emergency bearer services only
9	registered for “CSFB not preferred”, home network (not applicable)
10	registered for “CSFB not preferred”, roaming (not applicable)
11	emergency bearer services only

- 示例

```
>>> getState()
([11, 26909, 232301323, 7, 0, 466], [0, 26909, 232301323, 7, 0, 0])
```

net.getCi()

获取附近小区ID。

- 参数

无

- 返回值

成功返回一个list类型的数组，包含小区id，格式为：`[id, ……, id]`。数组成员数量并非固定不变，位置不同、信号强弱不同等都可能导致获取的结果不一样。

失败返回整型值-1。

- 示例

```
>>> net.getCi()
[14071232, 0]
```

net.getMnc()

获取附近小区的mnc。

- 参数

无

- 返回值

成功返回一个list类型的数组，包含小区mnc，格式为：`[mnc，……，mnc]`。数组成员数量并非固定不变，位置不同、信号强弱不同等都可能导致获取的结果不一样。

失败返回整型值-1。

- 示例

```
>>> net.getMnc()
[0, 0]
```

net.getMcc()

获取附近小区的mcc。

- 参数

无

- 返回值

成功返回一个list类型的数组，包含小区mcc，格式为：`[mcc，……，mcc]`。数组成员数量并非固定不变，位置不同、信号强弱不同等都可能导致获取的结果不一样。

失败返回整型值-1。

- 示例

```
>>> net.getMcc()
[1120, 0]
```

net.getLac()

获取附近小区的Lac。

- 参数

无

- 返回值

成功返回一个list类型的数组，包含小区lac，格式为：`[lac，……，lac]`。数组成员数量并非固定不变，位置不同、信号强弱不同等都可能导致获取的结果不一样。

失败返回整型值-1。

- 示例

```
>>> net.getLac()
[21771, 0]
```

net.getModemFun()

获取当前SIM模式。

- 参数

无

- 返回值

成功返回当前SIM模式：

0： 全功能关闭

1： 全功能开启（默认）

4： 飞行模式

失败返回整型值-1。

- 示例

```
>>> net.getModemFun()
1
```

> **net.setModemFun(function, rst)**

设置当前SIM模式。

- 参数

参数	参数类型	参数说明
function	int	设置SIM卡模式，0 - 全功能关闭， 1 - 全功能开启， 4 - 飞行模式
rst	int	可选参数，0 - 设置立即生效（默认为0），1 - 设置完重启

- 返回值

设置成功返回整型值0，设置失败返回整型值-1。

- 示例

```
>>> net.setModemFun(4)
0
```

fota - 固件升级

模块功能：固件升级。

创建fota对象

> **import fota**
>
> **fota_obj = fota()**

> **fota_obj.write(bytesData, file_size)**

写入升级包数据流。

- 参数

参数	参数类型	参数说明
bytesData	bytes	升级包文件数据
file_size	int	升级包文件总大小(单位：字节)

- 返回值

写入成功返回整型值0，写入失败返回值整型值-1。

> **fota_obj.verify()**

数据校验。

- 参数

无

- 返回值

检验成功返回整型值0，校验失败返回整型值-1。

- 示例

```
>>> fota_obj.verify()
0
```

audio - 音频播放

模块功能：音频播放，支持TTS、mp3以及AMR文件播放。

TTS

创建TTS对象

> **import audio**
> **tts = audio.TTS(device)**

- 参数

`device`：设备类型，0 - 话筒，1 - 耳机，2 - 喇叭。

- 示例

```
>>> import audio
>>> tts = audio.TTS(1)
```

> **tts.close()**

关闭TTS功能。

- 参数

无

- 返回值

成功返回整型0，失败返回整型-1。

tts.play(priority, breakin, mode, str)

语音播放，支持优先级0~4，数字越大优先级越高，每个优先级组可同时最多加入10个播放任务；播放策略说明如下：

1. 如果当前正在播放任务A，并且允许被打断，此时有高优先级播放任务B，那么会打断当前低优先级播放任务A，直接播放高优先级任务B；
2. 如果当前正在播放任务A，并且不允许被打断，此时有高优先级播放任务B，那么B播放任务将会加入到播放队列中合适的位置，等待A播放完成，再依次从队列中按照优先级从高到低播放其他任务；
3. 如果当前正在播放任务A，且不允许被打断，此时来了一个同优先级播放任务B，那么B会被加入到该优先级组播放队列队尾，等待A播放完成，再依次从队列中按照优先级从高到低播放其他任务；
4. 如果当前正在播放任务A，且允许被打断，此时来了一个同优先级播放任务B，那么会打断当前播放任务A，直接播放任务B；
5. 如果当前正在播放任务A，且任务A的优先级组播放队列中已经有几个播放任务存在，且该优先级组播放队列最后一个任务N是允许被打断的，此时如果来了一个同样优先级的播放任务B，那么任务B会直接覆盖掉任务N；也就是说，某个优先级组，只有最后一个元素是允许被打断的，即breakin为1，其他任务都是不允许被打断的；
6. 如果当前正在播放任务A，不管任务A是否允许被打断，此时来了一个优先级低于任务A的请求B，那么将B加入到B对应优先级组播放队列。

- 参数

参数	参数类型	参数说明
priority	int	播放优先级，支持优先级0~4，数值越大优先级越高
breakin	int	打断模式，0表示不允许被打断，1表示允许被打断
mode	int	编码模式，1 - UNICODE16(Size end conversion)，2 - UTF-8，3 - UNICODE16(Don't convert)
str	string	待播放字符串

- 返回值

播放成功返回整型0；

播放失败返回整型-1；

无法立即播放，加入播放队列，返回整型1；

无法立即播放，且该请求的优先级组队列任务已达上限，无法加入播放队列，返回整型-2。

- 示例

```
>>> import audio
>>> tts = audio.TTS(1)
#正在播放任务A，且A允许被打断，此时来了任务B，且优先级高于任务A，那么A会被#打断，直接播放B
>>> tts.play(1, 1, 2, '1111111111111111')  #任务A
```

```
0
>>> tts.play(2, 0, 2, '2222222222222222')  #任务B
0

#正在播放任务A，且A不允许被打断，此时来了任务B，且优先级高于任务A，那么B会#被加入播放队列，等待A播放完成播放B（假设播放队列之前为空）
>>> tts.play(1, 0, 2, '1111111111111111')  #任务A
0
>>> tts.play(2, 0, 2, '2222222222222222')  #任务B
1

#正在播放任务A，且A允许被打断，此时来了任务B，且优先级和A优先级一样，那么A
#会被打断，直接播放B
>>> tts.play(2, 1, 2, '22222222222222222')  #任务A
0
>>> tts.play(2, 0, 2, '3333333333333333')  #任务B
0

#正在播放任务A，且A不允许被打断，此时来了任务B，且优先级和A优先级一样，那么#B会被加入播放队列，等待A播放完成播放B（假设播放队列之前为空）
>>> tts.play(2, 0, 2, '22222222222222222')  #任务A
0
>>> tts.play(2, 0, 2, '3333333333333333')  #任务B
1

#正在播放A，且A不允许被打断，此时来了任务B，且任务B允许被打断，优先级与A相同，那么任务B会被加入到播放队列中，此时又来了一个任务C，且优先级和A、B相同，那么C会被加入播放队列中，且直接覆盖率任务B，所以A播放完成下一个播放的是C（假设播放队列之前为空）
>>> tts.play(2, 0, 2, '22222222222222222')  #任务A
0
>>> tts.play(2, 1, 2, '3333333333333333')  #任务B
1
>>> tts.play(2, 0, 2, '4444444444444444')  #任务C
1
```

tts播放中文示例：

注意，python文件开头需要加上“*# -- coding: UTF-8 --*”，如果播放的中文中有标点符号，要用英文的标点符号。

```
# -*- coding: UTF-8 -*-
import audio

tts = audio.TTS(1)
str1 = '移联万物,志高行远' #这里的逗号是英文的逗号
tts.play(4, 0, 2, str1)
```

tts.setCallback(usrFun)

注册用户的回调函数，用于通知用户TTS播放状态。注意，该回调函数中不要进行耗时以及阻塞性的操作，建议只进行简单、耗时短的操作。

- 参数

参数	参数类型	参数说明
usrFun	function	用户回调函数，函数形式见示例

- 返回值

注册成功返回整型0，失败返回整型-1。

- 示例

```
import audio

def tts_cb(event):
    if event == 2:
        print('TTS-play start.')
    elif event == 4:
        print('TTS-play finish.')

tts = audio.TTS(1)
tts.setCallback(tts_cb)
tts.play(1, 0, 2, 'QuecPython')
```

关于TTS播放回调函数参数event的几种状态值说明：

event	表示状态
2	开始播放
3	停止播放
4	播放完成
5	播放失败

tts.getVolume()

获取当前播放音量大小，音量值为0~9，0表示静音。

- 参数

无

- 返回值

成功返回整型音量大小值，失败返回整型-1。

- 示例

```
>>> tts.getVolume()
4
```

tts.setVolume(vol)

设置播放音量大小。

- 参数

参数	参数类型	参数说明
vol	int	音量值，音量值为0~9，0表示静音

- 返回值

成功返回整型音量值，失败返回整型-1。

- 示例

```
>>> tts.setVolume(6)
0
```

tts.getSpeed()

获取当前播放速度，速度值为0~9，值越大，速度越快。

- 参数

无

- 返回值

成功返回当前播放速度，失败返回整型-1。

- 示例

```
>>> tts.getSpeed()
4
```

tts.setSpeed(speed)

设置TTS播放速度。

- 参数

参数	参数类型	参数说明
speed	int	速度值，速度值为0~9，值越大，速度越快

- 返回值

成功返回整型0，失败返回整型-1。

- 示例

```
>>> tts.setSpeed(6)
0
```

tts.getState()

获取tts状态。

- 参数

无

- 返回值

0 – 整型值，表示当前无tts播放；

1 – 整型值，表示当前有tts正在播放。

- 示例

```
>>> tts1 = audio.TTS(1)
>>> tts1.getState()
0
>>> tts1.play(1, 0, 2, '8787878787878787')
0
>>> tts1.getState() #在上面tts播放过程中执行这句
1
```

> **tts.stop()**

停止TTS播放。

- 参数

无

- 返回值

成功返回整型0，失败返回整型-1。

Audio

创建一个对象

> **import audio**
>
> **aud = audio.Audio(device)**

- 参数

`device`： 设备类型，0 - 话筒，1 - 耳机，2 - 喇叭。

- 示例

```
>>> import audio
>>> aud = audio.Audio(1)
```

> **aud.play(priority, breakin, filename)**

音频文件播放，支持mp3和amr文件播放。支持优先级0~4，数字越大优先级越高，每个优先级组可同时最多加入10个播放任务，与TTS播放共用同一个播放队列。

- 参数

参数	参数类型	参数说明
priority	int	播放优先级，支持优先级0~4，数值越大优先级越高
breakin	int	打断模式，0表示不允许被打断，1表示允许被打断
filename	string	待播放的文件名称，包含文件存放路径

- 返回值

播放成功返回整型0；

播放失败返回整型-1；

无法立即播放，加入播放队列，返回整型1；

无法立即播放，且该请求的优先级组队列任务已达上限，无法加入播放队列，返回整型-2。

- 示例

```
>>> import audio
>>> a = audio.Audio(1)

>>> a.play(2, 1, 'U:/music.mp3')  #文件名前面要加上路径
0
```

关于文件播放路径的说明：

用户分区路径固定为'U:/'开头，表示用户分区的根目录，如果用户在根目录下新建audio目录，并将音频文件存放在根目录下的audio目录，那么播放接口中，传入的路径参数应该是：'U:/audio/music.mp3'。

- 说明

由于TTS和音频文件播放共用同一个播放队列，所以TTS中设置的播放优先级、打断模式不仅仅是和其他TTS播放任务比较，还会和音频文件播放任务的优先级和打断模式比较，反之，音频文件播放中设置的播放优先级与打断模式对TTS任务同样是有效的。

aud.stop()

停止音频文件播放。

- 参数

无

- 返回值

成功返回整型0，失败返回整型-1。

aud.setCallback(usrFun)

注册用户的回调函数，用于通知用户音频文件播放状态。注意，该回调函数中不要进行耗时以及阻塞性的操作，建议只进行简单、耗时短的操作。

- 参数

参数	参数类型	参数说明
usrFun	function	用户回调函数，函数形式见示例

- 返回值

注册成功返回整型0，失败返回整型-1。

- 示例

```
import audio

def audio_cb(event):
    if event == 0:
        print('audio-play start.')
    elif event == 7:
        print('audio-play finish.')

aud = audio.Audio(1)
aud.setCallback(audio_cb)
aud.play(1, 0, 'U:/test.mp3')
```

关于audio播放回调函数参数event的几种状态值说明：

event	表示状态
-1	播放错误
0	开始播放
7	播放完成

aud.getState()

获取audio初始化状态。

- 参数

无

- 返回值

audio初始化未完成返回整型值-1，初始化完成返回整型值0。

aud.getVolume()

获取audio音量大小。

- 参数

无

- 返回值

返回整型音量值。

> **aud.setVolume(vol)**

设置audio音量大小。

- 参数

参数	参数类型	参数说明
vol	int	音量等级，范围（1~11），数值越大，音量越大

- 返回值

设置成功返回整型0，失败返回整型-1。

- 示例

```
>>> aud.setVolume(6)
0
>>> aud.getVolume()
6
```

misc - 其他

模块功能：提供关机、软件重启、PWM以及ADC相关功能。

Power

关机以及软件重启。

使用前导入该类：from misc import Power

> **Power.powerDown()**

模块关机。

- 参数

无

- 返回值

无

> **Power.powerRestart()**

模块重启。

- 参数

无

- 返回值

无

> **Power. powerOnReason()**

获取模块启动原因。

- 参数

无

- 返回值

返回int数值，解释如下：

1：正常电源开机

2：重启

3：VBAT

4：RTC定时开机

5：Fault

6：VBUS

0：未知

> **Power. powerDownReason()**

获取模块上次关机原因。

- 参数

无

- 返回值

1：正常电源关机

2：电压过高

3：电压偏低

4：超温

5：WDT

6：VRTC 偏低

0：未知

> **Power. getVbatt()**

获取电池电压，单位mV。

- 参数

无

- 返回值

int类型电压值。

- 示例

```
>>> Power.getVbatt()
3590
```

PWM

常量说明

常量	说明
PWM.PWM0	PWM0
PWM.PWM1	PWM1
PWM.PWM2	PWM2
PWM.PWM3	PWM3
PWM.PWM4	PWM4
PWM.PWM5	PWM5

创建一个pwm对象

> **from misc import PWM**
>
> **pwm = PWM(PWM.PWMn, highTime, cycleTime)**

- 参数

参数	参数类型	参数说明
PWMn	int	PWM号 注：EC100YCN平台，支持PWM0-PWM5，对应引脚如下： PWM0 – 引脚号19 PWM1 – 引脚号18 PWM2 – 引脚号16 PWM3 – 引脚号17 PWM4 – 引脚号23 PWM5 – 引脚号22 注：EC600SCN平台，支持PWM0-PWM5，对应引脚如下： PWM0 – 引脚号52 PWM1 – 引脚号53 PWM2 – 引脚号57 PWM3 – 引脚号56 PWM4 – 引脚号70 PWM5 – 引脚号69
highTime	int	高电平时间，单位ms
cycleTime	int	pwm一个周期时间，单位ms

- 示例

```
>>> from misc import PWM
>>> pwm4 = PWM(PWM.PWM4, 100, 200)
```

> **pwm.open()**

开启PWM输出。

- 参数

无

- 返回值

成功返回整型0，失败返回整型-1。

> **pwm.close()**

关闭PWM输出。

- 参数

无

- 返回值

成功返回整型0，失败返回整型-1。

ADC

常量说明

常量	说明
ADC.ADC0	ADC通道0
ADC.ADC1	ADC通道1

创建一个ADC对象

> **from misc import ADC**
>
> **adc = ADC()**

- 示例

```
>>> from misc import ADC
>>> adc = ADC()
```

> **adc.open()**

ADC功能初始化。

- 参数

无

- 返回值

成功返回整型0，失败返回整型-1。

> **adc.read(ADCn)**

读取指定通道的电压值，单位mV。

- 参数

参数	参数类型	参数说明
ADCn	int	ADC通道 注：EC100YCN平台支持ADC0，ADC1，对应引脚如下 ADC0 – 引脚号39 ADC1 – 引脚号81

- 返回值

成功返回指定通道电压值，错误返回整型-1。

- 示例

```
>>>adc.read(ADC.ADC0)  #读取ADC通道0电压值
613
>>>adc.read(ADC.ADC1)  #读取ADC通道1电压值
605
```

> **adc.close()**

关闭ADC。

- 参数

无

- 返回值

0关闭成功，-1关闭失败。

modem - 设备相关

模块功能：设备信息获取。

> **modem.getDevImei()**

获取设备的IMEI。

- 参数

无

返回值

成功返回string类型设备的IMEI，失败返回整型值-1。

- 示例

```
>>> import modem
>>> modem.getDevImei()
'866327040830317'
```

modem.getDevModel()

获取设备型号。

- 参数

无

- 返回值

成功返回string类型设备型号，失败返回整型值-1。

- 示例

```
>>> modem.getDevModel()
'EC100Y'
```

modem.getDevSN()

获取设备序列号。

- 参数

无

- 返回值

成功返回string类型设备序列号，失败返回整型值-1。

- 示例

```
>>> modem.getDevSN()
'D1Q20GM050038341P'
```

modem.getDevFwVersion()

获取设备固件版本号。

- 参数

无

- 返回值

成功返回string类型固件版本号，失败返回整型值-1。

- 示例

```
>>> modem.getDevFwVersion()
'EC100YCNAAR01A01M16_OCPU_PY'
```

> **modem.getDevProductId()**

获取设备的制造商ID。

- 参数

无

- 返回值

成功返回设备制造商ID，失败返回整型值-1。

- 示例

```
>>> modem.getDevProductId()
'Quectel'
```

machine - 硬件相关功能

模块功能：包含与特定电路板上的硬件相关的特定功能。该模块中的大多数功能允许直接和不受限制地访问和控制系统上的硬件。

Pin

类功能：GPIO读写操作。

常量说明

常量	说明
Pin.GPIO1（EC600S / EC100Y）	GPIO1
Pin.GPIO2（EC600S / EC100Y）	GPIO2
Pin.GPIO3（EC600S / EC100Y）	GPIO3
Pin.GPIO4（EC600S / EC100Y）	GPIO4
Pin.GPIO5（EC600S / EC100Y）	GPIO5
Pin.GPIO6（EC600S）	GPIO6
Pin.GPIO7（EC600S）	GPIO7
Pin.GPIO8（EC600S）	GPIO8
Pin.GPIO9（EC600S）	GPIO9
Pin.GPIO10（EC600S）	GPIO10
Pin.IN	输入模式
Pin.OUT	输出模式
Pin.PULL_DISABLE	浮空模式
Pin.PULL_PU	上拉模式
Pin.PULL_PD	下拉模式

创建gpio对象

> **gpio = Pin(GPIOn, direction, pullMode, level)**

- 参数

参数	类型	说明
GPIOn	int	引脚号 EC100YCN平台引脚对应关系如下： GPIO1 – 引脚号22 GPIO2 – 引脚号23 GPIO3 – 引脚号178 GPIO4 – 引脚号199 GPIO5 – 引脚号204 EC600SCN平台引脚对应关系如下： GPIO1 – 引脚号10 GPIO2 – 引脚号11 GPIO3 – 引脚号12 GPIO4 – 引脚号13 GPIO5 – 引脚号14 GPIO6 – 引脚号15 GPIO7 – 引脚号16 GPIO8 – 引脚号39 GPIO9 – 引脚号40 GPIO10 – 引脚号48
direction	int	IN – 输入模式，OUT – 输出模式
pullMode	int	PULL_DISABLE – 浮空模式 PULL_PU – 上拉模式 PULL_PD – 下拉模式
level	int	0 - 设置引脚为低电平, 1- 设置引脚为高电平

- 示例

```
from machine import Pin
gpio1 = Pin(Pin.GPIO1, Pin.OUT, Pin.PULL_DISABLE, 0)
```

Pin.read()

获取PIN脚电平。

- 参数

无

- 返回值

PIN脚电平，0-低电平，1-高电平。

Pin.write(value)

设置PIN脚电平。

- 参数

参数	类型	说明
value	int	0 - 当PIN脚为输出模式时，设置当前PIN脚输出低; 1 - 当PIN脚为输出模式时，设置当前PIN脚输出高

- 返回值

设置成功返回整型值0，设置失败返回整型值-1。

- 示例

```
>>> from machine import Pin
>>> gpio1 = Pin(Pin.GPIO1, Pin.OUT, Pin.PULL_DISABLE, 0)
>>> gpio1.write(1)
0
>>> gpio1.read()
1
```

UART

类功能：uart串口数据传输。

常量说明

常量	说明
UART.UART0	UART0
UART.UART1	UART1
UART.UART2	UART2
UART.UART3	UART3

创建uart对象

> **uart = UART(UART.UARTn, buadrate, databits, parity, stopbits, flowctl)**

- 参数

参数	类型	说明
UARTn	int	端口号 EC100YCN平台与EC600SCN平台,UARTn作用如下: UART0 - DEBUG PORT UART1 – BT PORT UART2 – MAIN PORT UART3 – USB CDC PORT
buadrate	int	波特率，常用波特率都支持，如4800、9600、19200、38400、57600、115200、230400等
databits	int	数据位（5~8）
parity	int	奇偶校验（0 – NONE，1 – EVEN，2 - ODD）
stopbits	int	停止位（1~2）
flowctl	int	硬件控制流（0 – FC_NONE， 1 – FC_HW）

- 示例

```
>>> from machine import UART
>>> uart1 = UART(UART.UART1, 115200, 8, 0, 1, 0)
```

uart.any()

返回接收缓存器中有多少字节的数据未读。

- 参数

无

- 返回值

返回接收缓存器中有多少字节的数据未读。

- 示例

```
>>> uart.any()
20 #表示接收缓冲区中有20字节数据未读
```

uart.read(nbytes)

从串口读取数据。

- 参数

参数	类型	说明
nbytes	int	要读取的字节数

- 返回值

返回读取的数据。

> **uart.write(data)**

发送数据到串口。

- 参数

参数	类型	说明
data	string	发送的数据

- 返回值

返回发送的字节数。

> **uart.close()**

关闭串口。

- 参数

无

- 返回值

成功返回整型0，失败返回整型-1。

UART使用示例

```
>>> from machine import UART
>>> uart1 = UART(UART.UART1, 115200, 8, 0, 1, 0) #串口1
>>> uart1.any()
10
>>> uart1.read(5)
b'12345'
>>> uart1.any()
5
```

Timer

类功能：硬件定时器。

常量说明

常量	说明
Timer.Timer0	定时器0
Timer.Timer1	定时器1
Timer.Timer2	定时器2
Timer.Timer3	定时器3
Timer.ONE_SHOT	单次模式，定时器只执行一次
Timer.PERIODIC	周期模式，定时器循环执行

创建Timer对象

> **timer = Timer(Timern)**

创建Timer对象。

- 参数

参数	类型	说明
Timern	int	定时器号 EC100YCN支持定时器Timer0~Timer3

- 示例

```
>>> from machine import Timer
>>> timer1 = Timer(Timer.Timer1)
```

> **timer.start(period, mode, callback)**

启动定时器。

- 参数

参数	类型	说明
period	int	中断周期，单位毫秒
mode	int	运行模式 Timer.ONE_SHOT 单次模式，定时器只执行一次 Timer.PERIODIC 周期模式，循环执行
callback	function	定时器执行函数

- 返回值

启动成功返回整型值0，失败返回整型值-1。

- 示例

```
>>> def fun(args):
        print("###timer callback function###")
>>> timer.start(period=1000, mode=timer.PERIODIC, callback=fun)
0
###timer callback function###
###timer callback function###
###timer callback function###
……
```

> **timer.stop()**

关闭定时器。

- 参数

无

- 返回值

成功返回整型值0，失败返回整型值-1。

Timer使用示例

```
from machine import Timer
timer1 = Timer(Timer.Timer1) #定时器1
def run(t):
    print("run count")
timer1.start(period=1000, mode=timer1.PERIODIC, callback=run)
while 1:
    pass
```

ExtInt

类功能：用于配置I/O引脚在发生外部事件时中断。

常量说明

常量	说明
ExtInt.GPIO1（EC600S / EC100Y）	GPIO1
ExtInt.GPIO2（EC600S / EC100Y）	GPIO2
ExtInt.GPIO3（EC600S / EC100Y）	GPIO3
ExtInt.GPIO4（EC600S / EC100Y）	GPIO4
ExtInt.GPIO5（EC600S / EC100Y）	GPIO5
ExtInt.GPIO6（EC600S）	GPIO6
ExtInt.GPIO7（EC600S）	GPIO7
ExtInt.GPIO8（EC600S）	GPIO8
ExtInt.GPIO9（EC600S）	GPIO9
ExtInt.GPIO10（EC600S）	GPIO10
ExtInt.IRQ_RISING	上升沿触发
ExtInt.IRQ_FALLING	下降沿触发
ExtInt.IRQ_RISING_FALLING	上升和下降沿触发
ExtInt.PULL_DISABLE	浮空模式
ExtInt.PULL_PU	上拉模式
ExtInt.PULL_PD	下拉模式

创建ExtInt对象

> **extint = ExtInt(GPIOn, mode, pull, callback)**

- 参数

参数	类型	说明
GPIOn	int	引脚号 EC100YCN平台引脚对应关系如下： GPIO1 – 引脚号22 GPIO2 – 引脚号23 GPIO3 – 引脚号178 GPIO4 – 引脚号199 GPIO5 – 引脚号204 EC600SCN平台引脚对应关系如下： GPIO1 – 引脚号10 GPIO2 – 引脚号11 GPIO3 – 引脚号12 GPIO4 – 引脚号13 GPIO5 – 引脚号14 GPIO6 – 引脚号15 GPIO7 – 引脚号16 GPIO8 – 引脚号39 GPIO9 – 引脚号40 GPIO10 – 引脚号48
mode	int	设置触发方式 IRQ_RISING – 上升沿触发 IRQ_FALLING – 下降沿触发 IRQ_RISING_FALLING – 上升和下降沿触发
pull	int	PULL_DISABLE – 浮空模式 PULL_PU – 上拉模式 PULL_PD – 下拉模式
callback	function	中断触发回调函数

- 示例

```
>>> from machine import ExtInt
>>> def fun(args):
        print("###interrupt  %d ###" %args)
>>> extint = ExtInt(ExtInt.GPIO1, ExtInt.IRQ_FALLING, ExtInt.PULL_PU, fun)
```

extint.enable()

使能extint对象外部中断，当中断引脚收到上升沿或者下降沿信号时，会调用callback执行 。

- 参数

无

- 返回值

使能成功返回整型值0，使能失败返回整型值-1。

extint.disable()

禁用与extint对象关联的中断 。

- 参数

无

- 返回值

使能成功返回整型值0，使能失败返回整型值-1。

> **extint.line()**

返回引脚映射的行号。

- 参数

无

- 返回值

引脚映射的行号。

- 示例

```
>>> extint = ExtInt(ExtInt.GPIO1, ExtInt.IRQ_FALLING, ExtInt.PULL_PU, fun)
>>> ext.line()
32
```

RTC

类功能：提供获取设置rtc时间方法。

创建RTC对象

> **from machine import RTC**
>
> **rtc = RTC()**

> **rtc.datetime([year, month, day, week, hour, minute, second, microsecond])**

设置和获取RTC时间，不带参数时，则用于获取时间，带参数则是设置时间；设置时间的时候，参数week不参于设置，microsecond参数保留，暂未使用，默认是0。

- 参数

参数	类型	说明
year	int	年
month	int	月
day	int	日
week	int	星期，设置时间时，该参数不起作用，保留；获取时间时该参数有效
hour	int	时
minute	int	分
second	int	秒
microsecond	int	微秒，保留参数，暂未使用，设置时间时该参数写0即可

- 返回值

获取时间时，返回一个元组，包含日期时间，格式如下：

`[year, month, day, week, hour, minute, second, microsecond]`

设置时间时，设置成功返回整型值0，设置失败返回整型值-1 。

- 示例

```
>>> from machine import RTC
>>> rtc = RTC()
>>> rtc.datetime()
(2020, 9, 11, 5, 15, 43, 23, 0)
>>> rtc.datetime([2020, 3, 12, 1, 12, 12, 12, 0])
0
>>> rtc.datetime()
(2020, 3, 12, 4, 12, 12, 14, 0)
```

I2C

类功能：用于设备之间通信的双线协议。

常量说明

常量	
I2C.I2C0	i2c 通路索引号: 0
I2C.I2C1	i2c 通路索引号: 1
I2C.STANDARD_MODE	标准模式
I2C.FAST_MODE	快速模式

创建I2C对象

> **from machine import I2C**
>
> **i2c_obj = I2C(I2Cn，MODE)**

参数说明

参数	类型	说明
I2Cn	int	i2c 通路索引号: I2C.I2C0 : 0 （EC100Y） I2C.I2C0 : 1 （EC600S）
MODE	int	i2c 的工作模式: I2C.STANDARD_MODE : 0 标准模式 I2C.FAST_MODE： 1 快速模式

- 示例

```
from machine import I2C

i2c_obj = I2C(I2C.I2C0, I2C.STANDARD_MODE)  # 返回i2c对象
```

> **I2C.read(slaveaddress, addr,addr_len, r_data, datalen, delay)**

从 I2C 总线中读取数据。

参数说明

参数	类型	说明
slaveaddress	16进制	i2c 设备地址
addr	16进制	i2c 寄存器地址
addr_len	int	寄存器地址长度
r_data	bytearray	接收数据的字节数组
datalen	int	字节数组的长度
delay	int	延时，数据转换缓冲时间（单位ms）

- 返回值

成功返回整型值0，失败返回整型值-1。

> **I2C.write(slaveaddress, addr, addr_len, data, datalen)**

从 I2C 总线中写入数据。

参数说明

参数	类型	说明
slaveaddress	16进制	i2c 设备地址
addr	16进制	i2c 寄存器地址
addr_len	int	寄存器地址长度
data	bytearray	写入的数据
datalen	int	写入数据的长度

- 返回值

成功返回整型值0，失败返回整型值-1。

使用示例

需要连接设备使用！

```
from machine import I2C

I2C_SLAVE_ADDR = 0x1B  # i2c 设备地址
WHO_AM_I= bytearray({0x02, 0})   # i2c 寄存器地址，以buff的方式传入，取第一个值，计算一个值的长度

data = bytearray({0x12, 0})   # 输入对应指令
i2c_obj = I2C(I2C.I2C0, I2C.STANDARD_MODE)  # 返回i2c对象
i2c_obj.write(I2C_SLAVE_ADDR, WHO_AM_I, 1, data, 2) # 写入data

r_data = bytearray(2)  # 创建长度为2的字节数组接收
i2c_obj.read(I2C_SLAVE_ADDR, WHO_AM_I, 1, r_data, 2, 0)   # read
i2c_log.info(r_data[0])
i2c_log.info(r_data[1])
```

WDT

模块功能：APP应用程序发生异常不执行时进行系统重启操作

> **wdt = WDT(period)**

创建软狗对象。

- 参数

参数	类型	说明
period	int	设置软狗检测时间，单位(s)

- 返回值

返回软狗对象

> **wdt.feed()**

喂狗

- 参数

无

- 返回值

无

> **wdt.stop()**

关闭软狗功能

- 参数

无

- 返回值

无

使用示例

```
from machine import WDT
from machine import Timer


timer1 = Timer(Timer.Timer1)

def feed(t):
    wdt.feed()

if __name__ == '__main__':
    wdt = WDT(20)  # 启动看门狗，间隔时长
    timer1.start(period=15000, mode=timer1.PERIODIC, callback=feed)  # 使用定时器喂狗
    # wdt.stop()
```

pm - 低功耗

模块功能：在无业务处理时使系统进入休眠状态，进入低功耗模式。

> **lpm_fd = pm.create_wakelock(lock_name, name_size)**

创建wake_lock锁

- 参数

参数	类型	说明
lock_name	string	自定义lock名
name_size	int	lock_name的长度

- 返回值

成功返回wakelock的标识号，否则返回-1。

> **pm.delete_wakelock(lpm_fd)**

删除wake_lock锁

- 参数

参数	类型	说明
lpm_fd	int	需要删除的锁对应标识id

- 返回值

成功返回0。

> **pm.wakelock_lock(lpm_fd)**

加锁

- 参数

参数	类型	说明
lpm_fd	int	需要执行加锁操作的wakelock标识id

- 返回值

成功返回0，否则返回-1。

> **pm.wakelock_unlock(lpm_fd)**

释放锁

- 参数

参数	类型	说明
lpm_fd	int	需要执行释放锁操作的wakelock标识id

- 返回值

成功返回0，否则返回-1。

> **pm.autosleep(sleep_flag)**

自动休眠模式控制

- 参数

参数	类型	说明
sleep_flag	int	0，关闭自动休眠；1开启自动休眠

- 返回值

成功返回0。

> pm.get_wakelock_num()

获取已创建的锁数量

- 参数

无

- 返回值

返回已创建wakelock锁的数量。

使用示例

模拟测试，实际开发请根据业务场景选择使用！

```
import pm
import utime

# 创建wakelock锁
lpm_fd = pm.create_wakelock("test_lock", len("test_lock"))
# 设置自动休眠模式
pm.autosleep(1)

# 模拟测试，实际开发请根据业务场景选择使用
while 1:
    utime.sleep(20)  # 休眠
    res = pm.wakelock_lock(lpm_fd)
    print("ql_lpm_idlelock_lock, g_c1_axi_fd = %d" %lpm_fd)
    print("unlock  sleep")
    utime.sleep(20)
    res = pm.wakelock_unlock(lpm_fd)
    print(res)
    print("ql_lpm_idlelock_unlock, g_c1_axi_fd = %d" % lpm_fd)
    num = pm.get_wakelock_num()  # 获取已创建锁的数量
    print(num)
```

ure - 正则

模块功能：提供通过正则表达式匹配数据（ps：此re模块目前支持的操作符较少，部分操作符暂不支持）

支持操作符：

字符	说明
'.'	匹配任意字符
'[]'	匹配字符集合，支持单个字符和一个范围，包括负集
'^'	匹配字符串的开头。
'$'	匹配字符串的结尾。
'?'	匹配零个或前面的子模式之一。
'*'	匹配零个或多个先前的子模式。
'+'	匹配一个或多个先前的子模式。
'??'	非贪婪版本的？，匹配0或1
'*?'	非贪婪版本的*，匹配零个或多个
'+?'	非贪婪版本的+，匹配一个或多个
'|'	匹配该操作符的左侧子模式或右侧子模式。
'\d'	数字匹配
'\D'	非数字匹配
'\s'	匹配空格
'\S'	匹配非空格
'\w'	匹配"单词字符" (仅限ASCII)
'\W'	匹配非"单词字符"（仅限ASCII）

不支持：

- 重复次数（`{m,n}`）
- 命名组（`(?P<name>...)`）
- 非捕获组（`(?:...)`）
- 更高级的断言（`\b`，`\B`）
- 特殊字符转义，如 `\r`，`\n` - 改用Python自己的转义。

> **ure.compile(regex)**

compile 函数用于编译正则表达式，生成一个正则表达式（ Pattern ）对象，供 match() 和 search() 这两个函数使用。

- 参数

参数	类型	说明
regex	string	正则表达式

- 返回值

返回 regex 对象

> **ure.match(regex, string)**

将正则表达式对象 与 string 匹配，匹配通常从字符串的起始位置进行

- 参数

参数	类型	说明
regex	string	正则表达式
string	string	需要匹配的字符串数据

- 返回值

匹配成功返回一个匹配的对象，否则返回None。

> **ure.search(regex, string)**

re.search 扫描整个字符串并返回第一个成功的匹配。

- 参数

参数	类型	说明
regex	string	正则表达式
string	string	需要匹配的字符串数据

- 返回值

匹配成功返回一个匹配的对象，否则返回None。

Match 对象

匹配由 match() 和 serach 方法返回的对象

> **match.group(index)**

匹配的整个表达式的字符串

- 参数

参数	类型	说明
index	int	正则表达式中，group()用来提出分组截获的字符串, index=0返回整体，根据编写的正则表达式进行获取，当分组不存在时会抛出异常

- 返回值

返回匹配的整个表达式的字符串

> **match.groups()**

匹配的整个表达式的字符串

- 参数

无

- 返回值

返回一个包含该匹配组的所有子字符串的元组。

match.start(index)

返回匹配的子字符串组的起始原始字符串中的索引。

- 参数

参数	类型	说明
index	int	index 默认为整个组，否则将选择一个组

- 返回值

返回匹配的子字符串组的起始原始字符串中的索引。

match.end(index)

返回匹配的子字符串组的结束原始字符串中的索引。

- 参数

参数	类型	说明
index	int	index 默认为整个组，否则将选择一个组

- 返回值

返回匹配的子字符串组的结束原始字符串中的索引。

使用示例

```
import ure

res = '''
$GNRMC,133648.00,A,3149.2969,N,11706.9027,E,0.055,,311020,,,A,V*18
$GNGGA,133648.00,3149.2969,N,11706.9027,E,1,24,1.03,88.9,M,,M,,*6C
$GNGLL,3149.2969,N,11706.9027,E,133648.00,A,A*7A
$GNGSA,A,3,31,26,11,194,27,195,08,09,03,193,04,16,1.41,1.03,0.97,1*31
'''

r = ure.search("GNGGA(.+?)M", res)
print(r.group(0))
```

第三方库

aLiYun - 阿里云服务

模块功能：阿里云物联网套件客户端功能,目前的产品节点类型仅支持“设备”，设备认证方式支持“一机一密和“一型一密”。

> **aLiYun(productKey, productSecret, DeviceName, DeviceSecret)**

配置阿里云物联网套件的产品信息和设备信息。

- 参数

参数	类型	说明
productKey	string	产品标识
productSecret	string	可选参数，默认为None，productSecret，产品密钥 一机一密认证方案时，此参数传入None 一型一密认证方案时，此参数传入真实的产品密钥
DeviceName	string	设备名称
DeviceSecret	string	可选参数,默认为Non，设备密钥（一型一密认证方案时此参数传入None）

- 返回值

返回aLiYun连接对象。

> **aLiYun.setMqtt(clientID, clean_session, keepAlive)**

设置MQTT数据通道的参数

- 参数

参数	类型	说明
clientID	string	自定义阿里云连接id
clean_session	bool	可选参数，一个决定客户端类型的布尔值。 如果为True，那么代理将在其断开连接时删除有关此客户端的所有信息。 如果为False，则客户端是持久客户端，当客户端断开连接时，订阅信息和排队消息将被保留。默认为False
keepAlive	int	通信之间允许的最长时间段（以秒为单位）,默认为300，范围（60-1200）

- 返回值

成功返回整型值0，失败返回整型值-1。

> **aLiYun.setCallback(sub_cb)**

注册回调函数。

- 参数

参数	类型	说明
sub_cb	function	回调函数

- 返回值

无

aLiYun.subscribe(topic,qos)

订阅mqtt主题。

- 参数

参数	类型	说明
topic	string	topic
qos	int	MQTT消息服务质量（默认0，可选择0或1）0：发送者只发送一次消息，不进行重试 1：发送者最少发送一次消息，确保消息到达Broker

- 返回值

成功返回整型值0，失败返回整型值-1。

aLiYun.publish(topic,msg, qos=0)

发布消息。

- 参数

参数	类型	说明
topic	string	topic
msg	string	需要发送的数据
qos	int	MQTT消息服务质量（默认0，可选择0或1）0：发送者只发送一次消息，不进行重试 1：发送者最少发送一次消息，确保消息到达Broker

- 返回值

成功返回整型值0，失败返回整型值-1。

aLiYun.start()

运行连接。

- 参数

无

- 返回值

无

使用示例

```
from aLiYun import aLiYun

productKey = ""     # 产品标识
productSecret = None  # 产品密钥（一机一密认证此参数传入None）
DeviceName = ""     # 设备名称
DeviceSecret = None   # 设备密钥（一型一密认证此参数传入None）
# 创建aliyun连接对象
ali = aLiYun(productKey, productSecret, DeviceName, DeviceSecret)
# 设置mqtt连接属性
clientID = "mqttText"  # 自定义字符（不超过64）
ali.setMqtt(clientID, clean_session=False, keepAlive=300)
# 回调函数
def sub_cb(topic, msg):
    print("subscribe recv:")
    print(topic, msg)
# 设置回调函数
ali.setCallback(sub_cb)
topic = ""  # 主题
# 订阅主题
ali.subscribe(topic)
# 发布消息
ali.publish(topic, "hello world")
# 运行
ali.start()
```

TenCentYun- 腾讯云服务

模块功能：腾讯云物联网套件客户端功能,目前的产品节点类型仅支持“设备”，设备认证方式支持“一机一密和“动态注册认证”。

> **TXyun(productID, devicename, devicePsk, ProductSecret)**

配置阿里云物联网套件的产品信息和设备信息。

- 参数

参数	类型	说明
productID	string	产品标识（唯一ID）
ProductSecret	string	可选参数，默认为None，productSecret，产品密钥 一机一密认证方案时，此参数传入None 一型一密认证方案时，此参数传入真实的产品密钥
devicename	string	设备名称
devicePsk	string	可选参数,默认为Non，设备密钥（一型一密认证方案时此参数传入None）

- 返回值

返回TXyun连接对象。

TXyun.setMqtt(clean_session, keepAlive)

设置MQTT数据通道的参数

- 参数

参数	类型	说明
clean_session	bool	可选参数，一个决定客户端类型的布尔值。 如果为True，那么代理将在其断开连接时删除有关此客户端的所有信息。 如果为False，则客户端是持久客户端，当客户端断开连接时，订阅信息和排队消息将被保留。默认为False
keepAlive	int	通信之间允许的最长时间段（以秒为单位）,默认为300，范围（60-1000），建议300以上

- 返回值

成功返回整型值0，失败返回整型值-1。

TXyun.setCallback(sub_cb)

注册回调函数。

- 参数

参数	类型	说明
sub_cb	function	设置消息回调函数，当服务端响应时触发该方法

- 返回值

无

TXyun.subscribe(topic,qos)

订阅mqtt主题。

- 参数

参数	类型	说明
topic	string	topic
qos	int	MQTT消息服务质量（默认0，可选择0或1）MQTT消息服务质量（默认0，可选择0或1）0：发送者只发送一次消息，不进行重试 1：发送者最少发送一次消息，确保消息到达Broker

- 返回值

成功返回整型值0，失败返回整型值-1。

> **TXyun.publish(topic,msg, qos=0)**

发布消息。

- 参数

参数	类型	说明
topic	string	topic
msg	string	需要发送的数据
qos	int	MQTT消息服务质量（默认0，可选择0或1）MQTT消息服务质量（默认0，可选择0或1）0：发送者只发送一次消息，不进行重试 1：发送者最少发送一次消息，确保消息到达Broker

- 返回值

成功返回整型值0，失败返回整型值-1。

> **TXyun.start()**

运行连接。

- 参数

无

- 返回值

无

使用示例

```
from TenCentYun import TXyun


productID = ""  # 产品标识
devicename = ""   # 设备名称
devicePsk = ""   # 设备密钥（一型一密认证此参数传入None）
ProductSecret = None   # 产品密钥（一机一密认证此参数传入None）

tenxun = TXyun(productID, devicename, devicePsk, ProductSecret)  # 创建连接对象

def sub_cb(topic, msg):   # 云端消息响应回调函数
    print("subscribe recv:")
    print(topic, msg)

tenxun.setMqtt()
tenxun.setCallback(sub_cb)
topic = ""  # 输入自定义的Topic
tenxun.subscribe(topic)
tenxun.publish(topic, "hello world")
tenxun.start()
```

request - HTTP

模块功能：HTTP客户端的相关功能函数。

> **request.get(url, data, headers)**

发送GET请求。

- 参数

参数	类型	说明
url	string	网址
data	json	（可选参数）附加到请求的正文，json类型，默认为None
headers	dict	（可选参数）请求头，默认为None

- 示例

```
import request
url = "http://httpbin.org/get"
response = request.get(url)
```

> **request.post(url, data, headers)**

发送POST请求。

- 参数

参数	类型	说明
url	string	网址
data	json	（可选参数）附加到请求的正文，json类型，默认为None
headers	dict	（可选参数）请求头，默认为None

- 示例

```
import request
url = "http://httpbin.org/post"
response = request.post(url)
```

> **request.put(url, data, headers)**

发送PUT请求。

- 参数

参数	类型	说明
url	string	网址
data	json	（可选参数）附加到请求的正文，json类型，默认为None
headers	dict	（可选参数）请求头，默认为None

- 示例

```
import request
url = "http://httpbin.org/put"
response = request.put(url)
```

request.head(url, data, headers)

发送HEAD请求。

- 参数

参数	类型	说明
url	string	网址
data	json	（可选参数）附加到请求的正文，json类型，默认为None
headers	dict	（可选参数）请求头，默认为None

- 示例

```
import request
url = "http://httpbin.org/head"
response = request.head(url)
```

request.patch(url, data, headers)

发送PATCH请求。

- 参数

参数	类型	说明
url	string	网址
data	json	（可选参数）附加到请求的正文，json类型，默认为None
headers	dict	（可选参数）请求头，默认为None

- 示例

```
import request
url = "http://httpbin.org/patch"
response = request.patch(url)
```

> request.delete(url, data, headers)

发送DELETE请求。

- 参数

参数	类型	说明
url	string	网址
data	json	（可选参数）附加到请求的正文，json类型，默认为None
headers	dict	（可选参数）请求头，默认为None

- 示例

```
import request
url = "http://httpbin.org/delete"
response = request.delete(url)
```

Response类方法说明

> response =request.get(url)

方法	说明
response.content	返回响应的内容，以字节为单位
response.text	以文本方式返回响应的内容，编码为unicode
response.json()	返回响应的json编码内容并转为dict类型
response.close()	关闭socket

request使用示例

```
import request
import ujson

url = "http://httpbin.org/post"
data = {"key1": "value1", "key2": "value2", "key3": "value3"}

# POST请求
response = request.post(url, data=ujson.dumps(data))
print(response.text)
```

log - 日志

模块功能：系统日志记录,分级别日志工具。

log.basicConfig(level)

设置日记输出级别，设置日志输出级别, 默认为log.INFO，系统只会输出 level 数值大于或等于该 level 的的日志结果。

- 参数

参数	参数类型	说明
CRITICAL	常量	日志记录级别的数值 50
ERROR	常量	日志记录级别的数值 40
WARNING	常量	日志记录级别的数值 30
INFO	常量	日志记录级别的数值 20
DEBUG	常量	日志记录级别的数值 10
NOTSET	常量	日志记录级别的数值 0

- 示例

```
import log
log.basicConfig(level=log.INFO)
```

log.getLogger(name)

获取logger对象，如果不指定name则返回root对象，多次使用相同的name调用getLogger方法返回同一个logger对象。

- 参数

参数	参数类型	说明
name	string	日志主题

- 返回值

log对象。

- 示例

```
import log
Testlog = log.getLogger (“TestLog”)
```

log.debug(tag, msg)

输出debug级别的日志。

- 参数

参数	参数类型	说明
tag	string	模块或功能名称，作为日志前缀
msg	string	可变参数，日志内容

- 返回值

无

- 示例

```
import log
Testlog = log.getLogger (“TestLog”)
Testlog.debug("Test message: %d(%s)", 100, "foobar")
```

log.info(tag,msg)

输出info级别的日志。

- 参数

参数	参数类型	说明
tag	string	模块或功能名称，作为日志前缀
msg	string	可变参数，日志内容

- 返回值

无

- 示例

```
import log
Testlog = log.getLogger (“TestLog”)
Testlog.info("Test message: %d(%s)", 100, "foobar")
```

log.warning(tag,msg)

输出warning级别的日志。

- 参数

参数	参数类型	说明
tag	string	模块或功能名称，作为日志前缀
msg	string	可变参数，日志内容

- 返回值

无

- 示例

```
import log
Testlog = log.getLogger (“TestLog”)
Testlog.warning("Test message: %d(%s)", 100, "foobar")
```

log.error(tag,msg)

输出error级别的日志。

- 参数

参数	参数类型	说明
tag	string	模块或功能名称，作为日志前缀
msg	string	可变参数，日志内容

- 返回值

无

- 示例

```
import log
Testlog = log.getLogger (“TestLog”)
Testlog.error("Test message: %d(%s)", 100, "foobar")
```

log.critical(tag,msg)

输出critical级别的日志。

- 参数

参数	参数类型	说明
tag	string	模块或功能名称，作为日志前缀
msg	string	可变参数，日志内容

- 返回值

无

- 示例

```
import log
Testlog = log.getLogger (“TestLog”)
Testlog.critical("Test message: %d(%s)", 100, "foobar")
```

log使用示例

```
import log

log.basicConfig(level=log.INFO)    # 设置日志输出级别

# 获取logger对象，如果不指定name则返回root对象，多次使用相同的name调用getLogger方法返回同一个logger对象
log_obj = log.getLogger("test")

log_obj.debug("Test message: %d(%s)", 100, "foobar")
log_obj.info("Test message2: %d(%s)", 100, "foobar")
log_obj.warning("Test message3: %d(%s)")
log_obj.error("Test message4")
log_obj.critical("Test message5")
```

umqtt - MQTT

模块功能:提供创建MQTT客户端发布订阅功能。

```
QoS级别说明
在MQTT协议中，定义了三个级别的QoS，分别是：
QoS0 – 最多一次，是最低级别；发送者发送完消息之后，并不关心消息是否已经到达接收方；
QoS1 – 至少一次，是中间级别；发送者保证消息至少送达到接收方一次；
QoS2 – 有且仅有一次，是最高级别；保证消息送达且仅送达一次。
```

> **MQTTClient(client_id, server, port=0, user=None, password=None, keepalive=0, ssl=False, ssl_params={})**

构建mqtt连接对象。

- 参数

参数	参数类型	说明
client_id	string	客户端 ID，具有唯一性
server	string	服务端地址，可以是 IP 或者域名
port	int	服务器端口（可选）。 默认为1883，请注意，MQTT over SSL/TLS的默认端口是8883
user	string	（可选) 在服务器上注册的用户名
password	string	（可选) 在服务器上注册的密码
keepalive	int	（可选）客户端的keepalive超时值。 默认为60秒，范围（60~1200）s
ssl	bool	（可选）是否使能 SSL/TLS 支持
ssl_params	string	（可选）SSL/TLS 参数

- 返回值

mqtt对象。

MQTTClient.set_callback(callback)

设置回调函数，收到消息时会被调用。

- 参数

参数	参数类型	说明
callback	function	消息回调函数

- 返回值

无

MQTTClient.set_last_will(topic,msg,retain=False,qos=0)

设置要发送给服务器的遗嘱，客户端没有调用disconnect()异常断开，则发送通知到客户端。

- 参数

参数	参数类型	说明
topic	string	遗嘱主题
msg	string	遗嘱的内容
retain	bool	retain = True boker会一直保留消息，默认False
qos	int	消息服务质量(0~2)

- 返回值

无

MQTTClient.connect(clean_session=True)

与服务器建立连接，连接失败会导致MQTTException异常。

- 参数

参数	参数类型	说明
clean_session	bool	可选参数，一个决定客户端类型的布尔值。 如果为True，那么代理将在其断开连接时删除有关此客户端的所有信息。 如果为False，则客户端是持久客户端，当客户端断开连接时，订阅信息和排队消息将被保留。默认为False

- 返回值

无

MQTTClient.disconnect()

与服务器断开连接。

- 参数

无

- 返回值

无

MQTTClient.ping()

当keepalive不为0且在时限内没有通讯活动，会主动向服务器发送ping包,检测保持连通性，keepalive为0则不开启。

- 参数

无

- 返回值

无

MQTTClient.publish(topic,msg,qos)

发布消息。

- 参数

参数	类型	说明
topic	string	消息主题
msg	string	需要发送的数据
qos	int	MQTT消息服务质量（默认0，可选择0或1）0：发送者只发送一次消息，不进行重试 1：发送者最少发送一次消息，确保消息到达Broker

- 返回值

无

MQTTClient.subscribe(topic,qos)

订阅mqtt主题。

- 参数

参数	类型	说明
topic	string	topic
qos	int	MQTT消息服务质量（默认0，可选择0或1）0：发送者只发送一次消息，不进行重试 1：发送者最少发送一次消息，确保消息到达Broker

- 返回值

无

MQTTClient.check_msg()

检查服务器是否有待处理消息。

- 参数

无

- 返回值

无

MQTTClient.wait_msg()

阻塞等待服务器消息响应。

- 参数

无

- 返回值

无

示例代码

```
import utime
from umqtt import MQTTClient
state = 0

def sub_cb(topic, msg):
    '''
    消息回调函数
    '''
    global state
    mqtt_log.info("Subscribe Recv: Topic={},Msg={}".format(topic.decode(),
msg.decode()))
    state = 1

c = MQTTClient("umqtt_client", "mq.tongxinmao.com", 18830)
c.set_callback(sub_cb)
c.connect()
c.subscribe(b"/public/TEST/quecpython")
c.publish(b"/public/TEST/quecpython", b"my name is Quecpython!")

while True:
    c.wait_msg()  # 阻塞函数，监听消息
    if state == 1:
        break

# 关闭连接
c.disconnect()
```

ntptime - NTP对时

模块功能：该模块用于时间同步。

> **ntptime.host**

返回当前的ntp服务器，默认为"ntp.aliyun.com"。

> **ntptime.sethost(host)**

设置ntp服务器。

- 参数

参数	类型	说明
host	string	ntp服务器地址

- 返回值

成功返回整型值0，失败返回整型值-1。

> **ntptime.settime()**

同步ntp时间。

- 参数

无

- 返回值

成功返回整型值0，失败返回整型值-1。

ntptime使用示例

```
import ntptime

ntptime.sethost('pool.ntp.org')   # set the ntp service
ntptime.settime()   # sync the local time
```