
MICROPYTHON 用户手册

RT-THREAD 文档中心

上海睿赛德电子科技有限公司版权 ©2019



WWW.RT-THREAD.ORG

Tuesday 2nd July, 2019

版本和修订

Date	Version	Author	Note
2013-05-14	v1.0.0	bernard	初始版本
2018-12-29	v2.0.0	yangjie	内核内容修订
2018-12.29	v2.0.0	misonyo	组件及设备内容修订

目录

版本和修订	i
目录	ii
1 RT-Thread MicroPython 开发手册介绍	1
1.1 主要特性	1
1.2 MicroPython 的优势	1
1.3 MicroPython 的应用领域	1
1.3.1 产品原型验证	2
1.3.2 硬件测试	2
1.3.3 教育	2
1.3.4 创客 DIY	2
1.4 MicroPython 开发资源	3
2 RT-Thread MicroPython 快速上手	4
2.1 开始使用 MicroPython	4
2.1.1 选择合适的 BSP 平台	4
2.1.2 MicroPython 软件包的安装	4
2.1.3 选择开发环境	5
2.1.4 运行 MicroPython	5
2.2 MicroPython 基本功能	6
2.2.1 Python 语法与内建函数	6
2.2.1.1 使用 python 交互命令行	6
2.2.1.2 交互命令行的粘贴模式	6
2.2.2 MicroPython 内建模块	7
2.3 MicroPython 例程	7
2.3.1 闪烁灯	7
2.3.2 按键灯	8

3	RT-Thread MicroPython 基础知识	9
3.1	运行 python 文件	9
3.2	术语表	9
3.2.1	board	9
3.2.2	CPython	9
3.2.3	GPIO	9
3.2.4	interned string	10
3.2.5	MCU	10
3.2.6	micropython-lib	10
3.2.7	stream	10
3.2.8	upip	10
4	MicroPython 模块	11
4.1	Python 标准库和微型库	11
4.2	RT-Thread MicroPython 模块	11
4.2.1	常用内建模块	11
4.2.2	硬件模块	12
4.2.3	网络模块	12
4.2.4	常用第三方模块	12
4.3	rtthread – 系统相关函数	12
4.3.1	函数	13
4.3.1.1	rtthread.current_tid()	13
4.3.1.2	rtthread.is_preempt_thread()	13
4.3.1.3	rtthread.stacks_analyze()	13
4.3.2	示例	13
4.4	utime – 时间相关函数	13
4.4.1	函数	14
4.4.1.1	utime.localtime ([secs])	14
4.4.1.2	utime.mktime ()	14
4.4.1.3	utime.sleep (seconds)	14
4.4.1.4	utime.sleep_ms (ms)	14
4.4.1.5	utime.sleep_us (us)	14
4.4.1.6	utime.ticks_ms ()	14

4.4.1.7	<code>utime.ticks_us()</code>	15
4.4.1.8	<code>utime.ticks_cpu()</code>	15
4.4.1.9	<code>utime.ticks_add(ticks, delta)</code>	15
4.4.1.10	<code>utime.ticks_diff(ticks1, ticks2)</code>	15
4.4.1.11	<code>utime.time()</code>	15
4.4.2	示例	16
4.5	sys – 系统特有功能函数	16
4.5.1	函数	16
4.5.1.1	<code>sys.exit(retval=0)</code>	16
4.5.2	常数	16
4.5.2.1	<code>sys.argv</code>	16
4.5.2.2	<code>sys.byteorder</code>	17
4.5.2.3	<code>sys.implementation</code>	17
4.5.2.4	<code>sys.modules</code>	17
4.5.2.5	<code>sys.path</code>	17
4.5.2.6	<code>sys.platform</code>	17
4.5.2.7	<code>sys.stderr</code>	17
4.5.2.8	<code>sys.stdin</code>	17
4.5.2.9	<code>sys.stdout</code>	17
4.5.2.10	<code>sys.version</code>	17
4.5.2.11	<code>sys.version_info</code>	17
4.5.3	示例	18
4.6	math – 数学函数	18
4.6.1	常数	18
4.6.1.1	<code>math.e</code>	18
4.6.1.2	<code>math.pi</code>	18
4.6.2	函数	19
4.6.2.1	<code>math.acos(x)</code>	19
4.6.2.2	<code>math.acosh(x)</code>	19
4.6.2.3	<code>math.asin(x)</code>	19
4.6.2.4	<code>math.asinh(x)</code>	19
4.6.2.5	<code>math.atan(x)</code>	19
4.6.2.6	<code>math.atan2(y, x)</code>	19

4.6.2.7	<code>math.atanh(x)</code>	19
4.6.2.8	<code>math.ceil(x)</code>	19
4.6.2.9	<code>math.copysign(x, y)</code>	19
4.6.2.10	<code>math.cos(x)</code>	20
4.6.2.11	<code>math.cosh(x)</code>	20
4.6.2.12	<code>math.degrees(x)</code>	20
4.6.2.13	<code>math.erf(x)</code>	20
4.6.2.14	<code>math.erfc(x)</code>	20
4.6.2.15	<code>math.exp(x)</code>	20
4.6.2.16	<code>math.expm1(x)</code>	20
4.6.2.17	<code>math.fabs(x)</code>	20
4.6.2.18	<code>math.floor(x)</code>	21
4.6.2.19	<code>math.fmod(x, y)</code>	21
4.6.2.20	<code>math.frexp(x)</code>	21
4.6.2.21	<code>math.gamma(x)</code>	21
4.6.2.22	<code>math.isfinite(x)</code>	21
4.6.2.23	<code>math.isinf(x)</code>	22
4.6.2.24	<code>math.isnan(x)</code>	22
4.6.2.25	<code>math.ldexp(x, exp)</code>	22
4.6.2.26	<code>math.lgamma(x)</code>	22
4.6.2.27	<code>math.log(x)</code>	22
4.6.2.28	<code>math.log10(x)</code>	22
4.6.2.29	<code>math.log2(x)</code>	22
4.6.2.30	<code>math.modf(x)</code>	23
4.6.2.31	<code>math.pow(x, y)</code>	23
4.6.2.32	<code>math.radians(x)</code>	23
4.6.2.33	<code>math.sin(x)</code>	23
4.6.2.34	<code>math.sinh(x)</code>	23
4.6.2.35	<code>math.sqrt(x)</code>	23
4.6.2.36	<code>math.tan(x)</code>	23
4.6.2.37	<code>math.tanh(x)</code>	24
4.6.2.38	<code>math.trunc(x)</code>	24

4.7	<code>uio</code> – 输入/输出流	24
-----	---------------------------	----

4.7.1	函数	24
4.7.1.1	<code>uio.open(name, mode='r', **kwargs)</code>	24
4.7.2	类	24
4.7.2.1	<code>class uio.FileIO(...)</code>	24
4.7.2.2	<code>class uio.TextIOWrapper(...)</code>	24
4.7.2.3	<code>class uio.StringIO([string])</code>	24
4.7.2.4	<code>class uio.BytesIO([string])</code>	24
4.7.2.5	<code>getvalue()</code>	25
4.8	ucollections – 收集和容器类型	25
4.8.1	类	25
4.8.1.1	<code>ucollections.namedtuple(name, fields)</code>	25
4.8.1.2	<code>ucollections.OrderedDict(...)</code>	25
4.9	ustruct – 打包和解包原始数据类型	26
4.9.1	函数	26
4.9.1.1	<code>ustruct.calcsize(fmt)</code>	26
4.9.1.2	<code>ustruct.pack(fmt, v1, v2, ...)</code>	26
4.9.1.3	<code>ustruct.unpack(fmt, data)</code>	27
4.9.1.4	<code>ustruct.pack_into(fmt, buffer, offset, v1, v2, ...)</code>	27
4.9.1.5	<code>ustruct.unpack_from(fmt, data, offset=0)</code>	27
4.10	array – 数字数据数组	27
4.10.1	构造函数	27
4.10.1.1	<code>class array.array(typecode[, iterable])</code>	27
4.10.2	方法	28
4.10.2.1	<code>array.append(val)</code>	28
4.10.2.2	<code>array.extend(iterable)</code>	28
4.11	gc – 控制垃圾回收	28
4.11.1	函数	29
4.11.1.1	<code>gc.enable()</code>	29
4.11.1.2	<code>gc.disable()</code>	29
4.11.1.3	<code>gc.collect()</code>	29
4.11.1.4	<code>gc.mem_alloc()</code>	29
4.11.1.5	<code>gc.mem_free()</code>	29
4.12	machine – 与硬件相关的功能	29

4.12.1	函数	29
4.12.1.1	复位相关函数	29
4.12.1.1.1	machine.info()	29
4.12.1.1.2	machine.rest()	29
4.12.1.1.3	machine.reset_cause()	29
4.12.1.2	中断相关函数	30
4.12.1.2.1	machine.disable_irq()	30
4.12.1.2.2	machine.enable_irq(state)	30
4.12.1.3	功耗相关函数	30
4.12.1.3.1	machine.freq()	30
4.12.1.3.2	machine.idle()	30
4.12.1.3.3	machine.sleep()	30
4.12.1.3.4	machine.deepsleep()	30
4.12.2	常数	30
4.12.2.1	machine.IDLE	30
4.12.2.2	machine.SLEEP	30
4.12.2.3	machine.DEEPSLEEP	30
4.12.2.4	machine.PWRON_RESET	30
4.12.2.5	machine.HARD_RESET	30
4.12.2.6	machine.WDT_RESET	30
4.12.2.7	machine.DEEPSLEEP_RESET	30
4.12.2.8	machine.SOFT_RESET	30
4.12.2.9	machine.WLAN_WAKE	31
4.12.2.10	machine.PIN_WAKE	31
4.12.2.11	machine.RTC_WAKE	31
4.12.3	类	31
4.12.3.1	class Pin - 控制 I/O 引脚	31
4.12.3.2	class I2C - I2C 协议	31
4.12.3.3	class SPI - SPI 协议	31
4.12.3.4	class UART - 串口	31
4.12.3.5	class LCD - LCD	31
4.12.3.6	class RTC - RTC	31
4.12.3.7	class PWM - PWM	31

4.12.3.8	class ADC - ADC	31
4.12.3.9	class WDT - 看门狗	31
4.12.3.10	class TIMER - 定时器	31
4.12.4	示例	31
4.13	machine.Pin	32
4.13.1	构造函数	32
4.13.1.1	class machine.Pin(id, mode = -1, pull = -1, value)	32
4.13.2	方法	33
4.13.2.1	Pin.init(mode= -1, pull= -1, *, value, drive, alt)	33
4.13.2.2	Pin.value([x])	33
4.13.2.3	Pin.name()	33
4.13.3	常量	33
4.13.3.1	选择引脚模式:	33
4.13.3.1.1	Pin.IN	33
4.13.3.1.2	Pin.OUT	33
4.13.3.1.3	Pin.OPEN_DRAIN	33
4.13.3.2	选择上/下拉模式:	33
4.13.3.2.1	Pin.PULL_UP	33
4.13.3.2.2	Pin.PULL_DOWN	33
4.13.3.2.3	None	33
4.13.4	示例	34
4.14	machine.I2C	34
4.14.1	构造函数	34
4.14.1.1	class machine.I2C(id= -1, scl, sda, freq=400000)	34
4.14.2	方法	35
4.14.2.1	I2C.init(scl, sda, freq=400000)	35
4.14.2.2	I2C.deinit()	35
4.14.2.3	I2C.scan()	35
4.14.3	I2C 基础方法	35
4.14.3.1	I2C.start()	35
4.14.3.2	I2C.stop()	35
4.14.3.3	I2C.readinto(buf, nack=True)	35
4.14.3.4	I2C.write(buf)	35

4.14.4	I2C 标准总线操作	35
4.14.4.1	I2C.readfrom (addr, nbytes, stop=True)	36
4.14.4.2	I2C.readfrom__into (addr, buf, stop=True)	36
4.14.4.3	I2C.writeto (addr, buf, stop=True)	36
4.14.5	内存操作	36
4.14.5.1	I2C.readfrom__mem (addr, memaddr, nbytes, *, addrsize=8)	36
4.14.5.2	I2C.readfrom__mem__into (addr, memaddr, buf, *, addrsize=8)	36
4.14.5.3	I2C.writeto__mem (addr, memaddr, buf, *, addrsize=8)	36
4.14.6	示例	36
4.14.6.1	软件模拟 I2C	36
4.14.6.2	硬件 I2C	37
4.15	machine.SPI	37
4.15.1	构造函数	37
4.15.1.1	class machine.SPI (id, ...)	37
4.15.2	方法	38
4.15.2.1	SPI.init (baudrate=1000000, *, polarity=0, phase=0, bits=8, first-bit=SPI.MSB, sck=None, mosi=None, miso=None)	38
4.15.2.2	SPI.deinit ()	38
4.15.2.3	SPI.read (nbytes, write=0x00)	38
4.15.2.4	SPI.readinto (buf, write=0x00)	38
4.15.2.5	SPI.write (buf)	38
4.15.2.6	SPI.write__readinto (write_buf, read_buf)	38
4.15.3	常量	39
4.15.3.1	SPI.MASTER	39
4.15.3.2	SPI.MSB	39
4.15.3.3	SPI.LSB	39
4.15.4	示例	39
4.15.4.1	软件模拟 SPI	39
4.15.4.2	硬件 SPI	39
4.16	machine.UART	40
4.16.1	构造函数	40
4.16.1.1	class machine.UART (id, ...)	40
4.16.2	方法	40

4.16.2.1	UART.init (baudrate = 9600, bits=8, parity=None, stop=1)	40
4.16.2.2	UART.deinit ()	40
4.16.2.3	UART.read ([nbytes])	40
4.16.2.4	UART.readinto (buf[, nbytes])	40
4.16.2.5	UART.readline ()	41
4.16.2.6	UART.write (buf)	41
4.16.3	示例	41
4.17	machine.LCD	41
4.17.1	构造函数	41
4.17.1.1	class machine.LCD ()	41
4.17.2	方法	41
4.17.2.1	LCD.light (value)	41
4.17.2.2	LCD.fill (color)	42
4.17.2.3	LCD.pixel (x, y, color)	42
4.17.2.4	LCD.text (str, x, y, size)	42
4.17.2.5	LCD.line (x1, y1, x2, y2)	42
4.17.2.6	LCD.rectangle (x1, y1, x2, y2)	42
4.17.2.7	LCD.circle (x1, y1, r)	42
4.17.3	示例	42
4.18	machine.RTC	43
4.18.1	构造函数	43
4.18.1.1	class machine.RTC ()	43
4.18.2	方法	43
4.18.2.1	RTC.init (datetime)	43
4.18.2.2	RTC.deinit ()	43
4.18.2.3	RTC.now ()	44
4.18.3	示例	44
4.19	machine.PWM	44
4.19.1	构造函数	44
4.19.1.1	class machine.PWM (id, channel, freq, duty)	44
4.19.2	方法	45
4.19.2.1	PWM.init (channel, freq, duty)	45
4.19.2.2	PWM.deinit ()	45

4.19.2.3	PWM.freq(freq)	45
4.19.2.4	PWM.duty(duty)	45
4.19.3	示例	45
4.20	machine.ADC	45
4.20.1	构造函数	46
4.20.1.1	class machine.ADC(id, channel)	46
4.20.2	方法	46
4.20.2.1	ADC.init(channel)	46
4.20.2.2	ADC.deinit()	46
4.20.2.3	ADC.read()	46
4.20.3	示例	46
4.21	machine.WDT	47
4.21.1	构造函数	47
4.21.1.1	class machine.WDT(timeout=5)	47
4.21.2	方法	47
4.21.2.1	WDT.feed()	47
4.21.3	示例	47
4.22	machine.Timer	48
4.22.1	构造函数	48
4.22.1.1	class machine.Timer(id)	48
4.22.2	方法	48
4.22.2.1	Timer.init(mode = Timer.PERIODIC, period = 0, callback = None) . . .	48
4.22.2.2	Timer.deinit()	48
4.22.3	常量	49
4.22.3.1	选择定时器模式:	49
4.22.3.1.1	Timer.PERIODIC	49
4.22.3.1.2	Timer.ONE_SHOT	49
4.22.4	示例	49
4.23	uos – 基本的操作系统服务	49
4.23.1	函数	49
4.23.1.1	uos.chdir(path)	49
4.23.1.2	uos.getcwd()	49
4.23.1.3	uos.listdir([dir])	50

4.23.1.4	uos.mkdir (path)	50
4.23.1.5	uos.remove (path)	50
4.23.1.6	uos.rmdir (path)	50
4.23.1.7	uos.rename (old_path, new_path)	50
4.23.1.8	uos.stat (path)	50
4.23.1.9	uos.sync ()	50
4.23.2	示例	50
4.24	uselect – 等待流事件	51
4.24.1	常数	51
4.24.1.1	select.POLLIN - 读取可用数据	51
4.24.1.2	select.POLLOUT - 写入更多数据	51
4.24.1.3	select.POLLERR - 发生错误	51
4.24.1.4	select.POLLHUP - 流结束/连接终止检测	51
4.24.2	函数	51
4.24.2.1	select.select (rlist, wlist, xlist[, timeout])	51
4.24.3	Poll 类	52
4.24.3.1	select.poll ()	52
4.24.3.2	poll.register (obj[, eventmask])	52
4.24.3.3	poll.unregister (obj)	52
4.24.3.4	poll.modify (obj, eventmask)	52
4.24.3.5	poll.poll ([timeout])	53
4.25	uctypes – 以结构化的方式访问二进制数据	53
4.25.1	常量	53
4.25.2	构造函数	53
4.25.2.1	class uctypes.struct (addr, descriptor, type)	53
4.25.3	方法	54
4.25.3.1	uctypes.sizeof (struct)	54
4.25.3.2	uctypes.addressof (obj)	54
4.25.3.3	uctypes.bytes_at (addr, size)	54
4.25.3.4	uctypes bytearray_at (addr, size)	54
4.26	uerrno – 系统错误码模块	54
4.26.1	示例	55
4.27	_thread – 多线程支持	55

4.27.1 示例	55
4.28 cmath – 复数的数学函数	55
4.28.1 函数	56
4.28.1.1 cmath.cos (z)	56
4.28.1.2 cmath.exp (z)	56
4.28.1.3 cmath.log (z)	56
4.28.1.4 cmath.log10 (z)	56
4.28.1.5 cmath.phase (z)	56
4.28.1.6 cmath.polar (z)	56
4.28.1.7 cmath.rect (r, phi)	56
4.28.1.8 cmath.sin (z)	56
4.28.1.9 cmath.sqrt (z)	56
4.28.2 常数	56
4.28.2.1 cmath.e	56
4.28.2.2 cmath.pi	57
4.29 ubinscii – 二进制/ ASCII 转换	57
4.29.1 函数	57
4.29.1.1 ubinscii.hexlify (data[, sep])	57
4.29.1.2 ubinscii.unhexlify (data)	57
4.29.1.3 ubinscii.a2b_base64 (data)	57
4.29.1.4 ubinscii.b2a_base64 (data)	58
4.30 uhashlib – 哈希算法	58
4.30.1 算法功能	58
4.30.1.1 SHA256	58
4.30.1.2 SHA1	58
4.30.1.3 MD5	58
4.30.2 函数	58
4.30.2.1 class hashlib.sha256 ([data])	58
4.30.2.2 class hashlib.sha1 ([data])	58
4.30.2.3 class hashlib.md5 ([data])	58
4.30.2.4 hash.update (data)	58
4.30.2.5 hash.digest ()	59
4.30.2.6 hash.hexdigest ()	59

4.31	uheapq – 堆排序算法	59
4.31.1	函数	59
4.31.1.1	uheapq.heappush (heap, item)	59
4.31.1.2	uheapq.heappop (heap)	59
4.31.1.3	uheapq.heapify (x)	59
4.32	ujson – JSON 编码与解码	59
4.32.1	函数	59
4.32.1.1	ujson.dumps (obj)	59
4.32.1.2	ujson.loads (str)	60
4.33	ure – 正则表达式	60
4.33.1	匹配字符集	60
4.33.1.1	匹配任意字符	60
4.33.1.2	匹配字符集合, 支持单个字符和一个范围	60
4.33.1.3	支持多种匹配元字符	60
4.33.2	函数	61
4.33.2.1	ure.compile (regex)	61
4.33.2.2	ure.match (regex, string)	61
4.33.2.3	ure.search (regex, string)	61
4.33.2.4	ure.DEBUG	61
4.33.3	正则表达式对象:	61
4.33.3.1	regex.match (string)	61
4.33.3.2	regex.search (string)	61
4.33.3.3	regex.split (string, max_split=-1)	61
4.33.4	匹配对象:	61
4.33.4.1	match.group ([index])	61
4.34	uzlib – zlib 解压缩	61
4.34.1	函数	62
4.34.1.1	uzlib.decompress (data)	62
4.35	urandom - 随机数生成模块	62
4.35.1	函数	62
4.35.1.1	urandom.choice (obj)	62
4.35.1.2	urandom.getrandbits (size)	62
4.35.1.3	urandom.randint (start, end)	63

4.35.1.4	urandom.random()	63
4.35.1.5	urandom.randrange (start, end, step)	63
4.35.1.6	urandom.seed (sed)	64
4.35.1.7	urandom.uniform (start, end)	65
4.36	usocket – 套接字模块	65
4.36.1	常数	65
4.36.1.1	地址簇	65
4.36.1.2	套接字类型	65
4.36.1.3	IP 协议号	65
4.36.1.4	套接字选项级别	65
4.36.2	函数	66
4.36.2.1	socket.socket	66
4.36.2.2	socket.getaddrinfo (host, port)	66
4.36.2.3	socket.close()	66
4.36.2.4	socket.bind (address)	66
4.36.2.5	socket.listen ([backlog])	66
4.36.2.6	socket.accept()	66
4.36.2.7	socket.connect (address)	66
4.36.2.8	socket.send (bytes)	67
4.36.2.9	socket.recv (bufsize)	67
4.36.2.10	socket.sendto (bytes, address)	67
4.36.2.11	socket.recvfrom (bufsize)	67
4.36.2.12	socket.setsockopt (level, optname, value)	67
4.36.2.13	socket.settimeout (value)	68
4.36.2.14	socket.setblocking (flag)	68
4.36.2.15	socket.read ([size])	68
4.36.2.16	socket.readinto (buf[, nbytes])	68
4.36.2.17	socket.readline()	68
4.36.2.18	socket.write (buf)	68
4.36.3	示例	68
4.36.3.1	TCP Server example	68
4.36.3.2	TCP Client example	69
4.37	network – 网络配置	69

4.37.1	专用的网络类配置	69
4.38	class WLAN – 控制内置的 WiFi 网络接口	69
4.38.1	构造函数	69
4.38.1.1	class network.WLAN(interface_id)	70
4.38.2	方法	70
4.38.2.1	WLAN.active([is_active])	70
4.38.2.2	WLAN.connect(ssid, password)	70
4.38.2.3	WLAN.disconnect()	70
4.38.2.4	WLAN.scan()	70
4.38.2.5	WLAN.status([param])	70
4.38.2.6	WLAN.isconnected()	70
4.38.2.7	WLAN.ifconfig([(ip, subnet, gateway, dns)])	71
4.38.2.8	WLAN.config('param')	71
4.38.2.9	WLAN.config(param=value, ...)	71
4.38.3	示例	71
5	RT-Thread MicroPython 包管理	73
5.1	1. 使用 micropython-lib 源代码	73
5.1.1	1.1 从 GitHub 上克隆/下载 micropython-lib 的源代码到本地	73
5.1.2	1.2 使用扩展包	74
5.2	2. upip 包管理器的使用	74
6	RT-Thread MicroPython 网络编程指南	77
6.1	预备知识	77
6.2	HttpClient	77
6.2.1	获取并安装 urequests 模块	77
6.2.2	urequests 模块的使用	78
6.3	HttpServer	78
6.3.1	获取并安装 MicroWebSrv 模块	78
6.3.2	MicroWebSrv 模块的使用	79
6.3.3	服务器功能的修改	82
6.4	MQTT	83
6.4.1	获取并安装 umqtt.simple 模块	83
6.4.2	umqtt.simple 模块的使用	83

6.4.2.1	MQTT 订阅功能	83
6.4.2.2	MQTT 发布功能	84
6.5	OneNET	85
6.5.1	准备工作	85
6.5.2	产品创建	85
6.5.2.1	用户注册	85
6.5.2.2	产品创建	85
6.5.3	硬件接入	86
6.5.3.1	设备的注册和接入	86
6.5.3.2	云平台向设备发送命令	87
6.5.3.3	设备向云平台上传数据	88
6.5.3.4	添加独立应用	90
6.5.4	代码讲解	90
6.5.4.1	附录示例代码	91

第 1 章

RT-Thread MicroPython 开发手册介绍

本手册介绍了 RT-Thread MicroPython 的基础知识、常用模块，以及开发新模块的流程。带领读者了解 MicroPython，并学会使用 MicroPython 进行开发。

1.1 主要特性

- MicroPython 是 Python 3 编程语言的一种精简而高效的实现，它包含 Python 标准库的一个子集，并被优化为在微控制器和受限环境中运行。
- RT-Thread MicroPython 可以运行在任何搭载了 RT-Thread 操作系统并且有一定资源的嵌入式平台上。
- MicroPython 可以运行在有一定资源的开发板上，给你一个低层次的 Python 操作系统，可以用来控制各种电子系统。
- MicroPython 富有各种高级特性，比如交互式提示、任意精度整数、闭包函数、列表解析、生成器、异常处理等等。
- MicroPython 的目标是尽可能与普通 Python 兼容，使开发者能够轻松地将代码从桌面端转移到微控制器或嵌入式系统。程序可移植性很强，因为不需要考虑底层驱动，所以程序移植变得轻松和容易。

1.2 MicroPython 的优势

- Python 是一款容易上手的脚本语言，同时具有强大的功能，语法优雅简单。使用 MicroPython 编程可以降低嵌入式的开发门槛，让更多的人体验嵌入式的乐趣。
- 通过 MicroPython 实现硬件底层的访问和控制，不需要了解底层寄存器、数据手册、厂家的库函数等，即可轻松控制硬件。
- 外设与常用功能都有相应的模块，降低开发难度，使开发和移植变得容易和快速。

1.3 MicroPython 的应用领域

- MicroPython 在嵌入式系统上完整实现了 Python3 的核心功能，可以在产品开发的各个阶段给开发者带来便利。

- 通过 MicroPython 提供的库和函数，开发者可以快速控制 LED、液晶、舵机、多种传感器、SD、UART、I2C 等，实现各种功能，而不用再去研究底层硬件模块的使用方法，翻看寄存器手册。这样不但降低了开发难度，而且减少了重复开发工作，可以加快开发速度，提高开发效率。以前需要较高水平的嵌入式工程师花费数天甚至数周才能完成的功能，现在普通的嵌入式开发者用几个小时就能实现类似的功能。
- 随着半导体技术的不断发展，芯片的功能、内部的存储器容量和资源不断增加，成本不断降低，可以使用 MicroPython 来进行开发设计的应用领域也会越来越多。

1.3.1 产品原型验证

- 众所周知，在开发新产品时，原型设计是一个非常重要的环节，这个环节需要以最快速的方式设计出产品的大致模型，并验证业务流程或者技术点。与传统开发方法相比，使用 MicroPython 对于原型验证非常有用，让原型验证过程变得轻松，加速原型验证过程。
- 在进行一些物联网功能开发时，网络功能也是 MicroPython 的长处，可以利用现成的众多 MicroPython 网络模块，节省开发时间。而这些功能如果使用 C/C++ 来完成，会耗费几倍的时间。

1.3.2 硬件测试

- 嵌入式产品在开发时，一般会分为硬件开发及软件开发。硬件工程师并不一定都擅长软件开发，所以在测试新硬件时，经常需要软件工程师参与。这就导致软件工程师可能会耗费很多时间帮助硬件工程师查找设计或者焊接问题。有了 MicroPython 后，将 MicroPython 固件烧入待测试的新硬件，在检查焊接、连线等问题时，只需使用简单的 Python 命令即可测试。这样，硬件工程师一人即可搞定，再也不用麻烦别人了。

1.3.3 教育

- MicroPython 使用简单、方便，非常适合于编程入门。在校学生或者业余爱好者都可以通过 MicroPython 快速的开发一些好玩的项目，在开发的过程中学习编程思想，提高自己的动手能力。
- 下面是一些 MicroPython 教育项目：
 - 从 [TurnipBit](#) 开始完成编程启蒙
 - [MicroBit](#) 创意编程

1.3.4 创客 DIY

- MicroPython 无需复杂的设置，不需要安装特别的软件环境和额外的硬件，使用任何文本编辑器就可以进行编程。大部分硬件功能，使用一个命令就能驱动，不用了解硬件底层就能快速开发。这些特性使得 MicroPython 非常适合创客使用来开发一些有创意的项目。
- 下面是使用 MicroPython 制作的一些 DIY 项目：
 - [显示温湿度的 WIFI 时钟](#)
 - [OpenMV 智能摄像头](#)
 - [手机遥控车](#)
 - [搭建 MQTT 服务器](#)

1.4 MicroPython 开发资源

- [RT-Thread MicroPython 开发手册](#)
- [RT-Thread MicroPython 源码](#)
- [RT-Thread MicroPython 论坛](#)
- [MicroPython 官方网站](#)
- [官方在线文档](#)
- [MicroPython 在线演示](#)
- [MicroPython 源码](#)
- [MicroPython 官方论坛](#)
- [MicroPython 中文社区](#)

第 2 章

RT-Thread MicroPython 快速上手

2.1 开始使用 MicroPython

注意：RT-Thread MicroPython 需要运行在 **RT-Thread 3.0** 版本以上。

2.1.1 选择合适的 BSP 平台

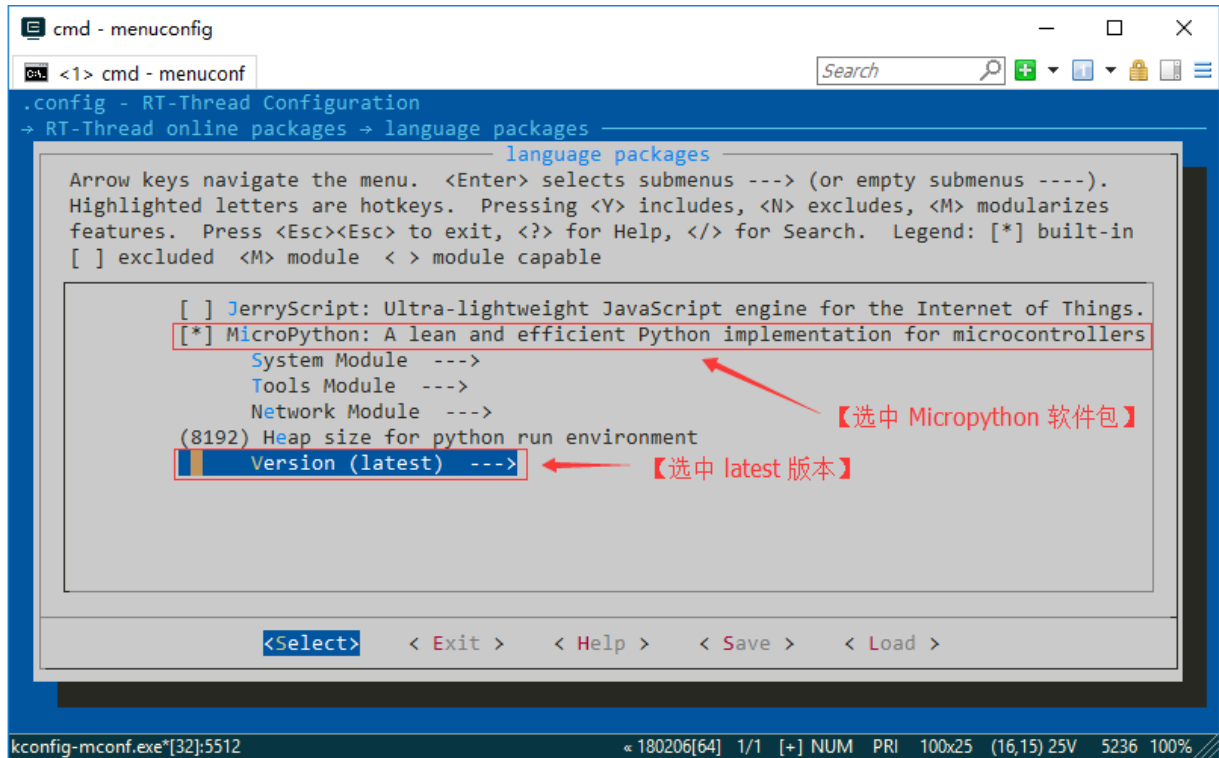
RT-Thread MicroPython mini 版本占用资源最大不超过：

- 1、ROM : 190KB
- 2、RAM : 8KB

常见的许多开发板都可以运行 MicroPython，如 `stm32f10x`、`stm32f40x`、`stm32f429-apollo`、`imxrt1052-evk`、`iot-camera` 等。

2.1.2 MicroPython 软件包的安装

- MicroPython 软件包可以通过 `env` 工具在线下载获得。在下载 MicroPython 软件包前，建议使用 `pkgs --upgrade` 命令更新软件包列表，并且在配置版本时勾选 `latest` 版本，如图：

图 2.1: *elect_micropytho*

- 使用 env 下载 MicroPython 软件包的方法请参考: [RT-Thread env 工具使用手册](#)

2.1.3 选择开发环境

- 目前 MicroPython 支持三种开发环境，分别为 MDK / IAR / GCC，选择合适的开发环境，使用 env 工具将 MicroPython 软件包开启后，需重新生成工程，再进行编译、下载。

2.1.4 运行 MicroPython

- 在 Finsh/MSH 命令行内输入 `python` 即可进入 MicroPython 的交互命令行 REPL(Read-Evaluate-Print-Loop)，可在终端看到如下界面：

```

\ | /
- RT -   Thread Operating System
/ | \   3.0.3 build Mar 22 2018
2006 - 2018 Copyright by rt-thread team
lwIP-2.0.2 initialized!
using iccarm, version: 8020001
build time: Mar 22 2018 18:50:15
msh />[PHY] wait autonegotiation complete...
sdcard init fail or timeout: -2!
[PHY] wait autonegotiation complete...

msh />python

MicroPython v1.9.3-477-g7b0a020-dirty on 2018-03-21; Universal python platform with RT-Thread
Type "help()" for more information.
>>> 

```

图 2.2: *elect_micropytho*

使用 `Ctrl-D` 或输入 `quit()` 以及 `exit()` 即可退出 REPL，回到 RT-Thread Finsh/MSH。

2.2 MicroPython 基本功能

2.2.1 Python 语法与内建函数

2.2.1.1 使用 python 交互命令行

- MicroPython 是 Python 3 编程语言的一种精简而高效的实现，语法和 Python 3 相同，并拥有丰富的内建函数，使用 MicroPython 交互命令行即可运行 Python 代码：

```
>>> print("hello RT-Thread!")
hello RT-Thread!
>>>
>>>
>>> █
```

图 2.3: *elect_micropytho*

2.2.1.2 交互命令行的粘贴模式

- MicroPython 比一般的 python 交互环境多了一个特别的粘贴模式，可以一次粘贴输入多行 python 代码。
- 在命令行提示符状态下，按下 `Ctrl-E` 组合键，就会出现提示：`paste mode; Ctrl-C to cancel, Ctrl-D to finish`。粘贴需要运行的代码后，按下 `Ctrl-D` 即可退出粘贴模式，同时输入的代码也会自动执行。
- 程序正在执行时，如果想取消，可以使用 `Ctrl-C`。

输入代码：

```
for i in range(1,10):
    print(i)
```

执行效果如下：

```
paste mode; Ctrl-C to cancel, Ctrl-D to finish
=== for i in range(1,10):
===     print(i)
1
2
3
4
5
6
7
8
9
>>> █
```

图 2.4: *elect_micropytho*

2.2.2 MicroPython 内建模块

- MicroPython 提供丰富的内建模块用来完成相关的程序功能。同时 RT-Thread 也提供了 `rtthread` 模块用来返回系统运行相关的信息。
- 以 `rtthread` 和 `time` 模块为例，调用方式如下：

```
>>> import rtthread
GC: total: 7936, used: 704, free: 7232
No. of 1-blocks: 11, 2-blocks: 6, max blk sz: 8, max free sz: 123
>>> rtthread.is_preempt_thread()
True
>>> rtthread.current_tid()
536935332
>>> rtthread.stacks_analyze()
thread      pri  status      sp      stack size max used left tick error
-----
tshell      20  ready  0x000001e8 0x000001000 36% 0x00000002 000
ntp_sync    26  suspend 0x0000007c 0x000000600 29% 0x00000001 000
phy         30  suspend 0x00000090 0x000000200 46% 0x00000002 000
tcpip       10  suspend 0x000000c4 0x000000800 16% 0x00000003 000
etx         12  suspend 0x00000098 0x000000400 14% 0x00000010 000
erx         12  suspend 0x000000a0 0x000000400 43% 0x00000008 000
mmcsd_detect 22  suspend 0x000000a0 0x000000400 47% 0x00000012 000
tidle       31  ready  0x0000005c 0x000000100 35% 0x00000015 000
main        10  suspend 0x0000008c 0x000000800 24% 0x00000009 000
>>> import time
>>> time.sleep(1)
>>> time.sleep_
sleep_ms    sleep_us
>>> time.sleep_ms(500)
>>> time.
__class__   __name__   sleep      sleep_ms
sleep_us    ticks_add  ticks_cpu  ticks_diff
ticks_ms    ticks_us   time
>>> time.sleep_us(10)
>>>
```

图 2.5: *elect_micropytho*

!!! tip “提示” 默认下载的 MicroPython 软件包为 mini 版本，为 RT-Thread 推出的最小版本的 MicroPython，如果想使用更多的 MicroPython 模块，可以在 `menuconfig` 配置项中打开更多的模块选项。

2.3 MicroPython 例程

通过 MicroPython 可以用非常简单的方式来控制开发板的硬件资源，下面用两个例子来说明：

以下例程运行在 **i.MX RT1050** 开发板上，运行之前需要开启 RT-Thread **Pin** 设备功能。

2.3.1 闪烁灯

- i.MX RT1050 开发板中: 第 52 号 pin 为 LED D18，与 phy 复位引脚共用

```
import time
from machine import Pin

LED = Pin(("LED1", 52), Pin.OUT_PP)  #将第52号 Pin 设备设置为输出模式
while True:
    LED.value(1)
```

```
time.sleep_ms(500)
LED.value(0)
time.sleep_ms(500)
```

针对自己的开发板修改引脚号，将以上脚本使用 3.1.2 章节介绍的**粘贴模式**输入，即可看到 LED 灯按照指定的频率闪烁。使用 **Ctrl-C** 可以取消当前正在运行程序。

2.3.2 按键灯

- i.MX RT1050 开发板中: [第 125 号 pin](#) 为 SW8

```
from machine import Pin

led = Pin(("LED1", 52), Pin.OUT_PP)
key = Pin(("KEY", 125), Pin.IN, Pin.PULL_UP) #将第125号 Pin 设备设置为上拉输入模式
while True:
    if key.value():
        led.value(0)
    else:
        led.value(1)
```

针对自己的开发板修改引脚号，使用**粘贴模式**输入以上脚本，即可通过按键 KEY 控制 LED 灯的亮灭。

第 3 章

RT-Thread MicroPython 基础知识

3.1 运行 python 文件

在 MicroPython 上运行 Python 文件有以下要求：

- 系统内使用了 `rt-thread` 的文件系统。
- 开启 `msh` 功能。

符合以上两点，就可以使用 `msh` 命令行中的 `python` 命令加上 `*.py` 文件名来执行一个 Python 文件了。

3.2 术语表

3.2.1 board

开发板，通常这个术语用来表示以一个特定的 MCU 为核心的开发板。它也可以被用来表示移植 MicroPython 到一个特定的开发板上，也可以表示像 Unix 移植这样没有开发板的移植。

3.2.2 CPython

CPython 是 Python 编程语言的一种实现，是最被人们所熟知和使用的一种。然而它只是许多种实现中的一种（其中包括 Jython、IronPython、PyPy、MicroPython 等）。由于没有正式的 Python 语言规范，只有 CPython 文档，在 Python 语言本身和 Cpython 这种实现之间画出一条界限并不容易。这同时也给其他方式的实现留下了更多的自由。比如 MicroPython 做了许多和 Cpython 不一样的事情，同时仍然成为 Python 语言的一种实现。

3.2.3 GPIO

通用输入/输出。控制电子信号最简单的方法。通过 GPIO 用户可以配置硬件信号引脚为输入或者输出，并设置或者获取其数字信号值（逻辑 ‘0’ 或 ‘1’）。MicroPython 使用抽象类 `machine.Pin` 来访问 GPIO。

3.2.4 interned string

由其唯一的标识而不是其地址引用的字符串。因此，可以用他们的标识符而不是通过内容快速地比较内部字符串。`interned` 字符串的缺点是插入操作需要时间（与现有 `interned` 字符串的数量成正比，也就是说会随着时间的推移耗时越来越久），而用于插入 `interned` 字符串的空间是不可回收的。当某个 `interned` 字符串被应用需求（作为关键字参数）或者对系统有益（可以减少查找的时间）时，就会被 MicroPython 的编译器和运行环境自动生成。由于上面的缺点，大多数字符串和输入/输出操作都不会产生 `interned` 字符串。

3.2.5 MCU

微控制器，也称单片机，通常资源比成熟的计算机系统要少的多，但是体积更小，也更便宜，功耗更低。MicroPython 被设计的足够小，并且优化到可以运行在一个普通的现代微控制器上。

3.2.6 micropython-lib

MicroPython 通常是单个的可执行/二进制文件，只有很少的内置模块。没有广泛的标准库可以用来和 CPython 相比。与 CPython 不同的是 MicroPython 有一个相关但是独立的项目 `micropython-lib`，它提供了来自 CPython 标准库的许多模块的实现。然而这些模块大部分需要类似于 POSIX 的环境，只能在 MicroPython 的 Unix 移植上工作。安装方法与 CPython 也不同，需要使用手动复制或者使用 `upip` 来安装。由于 RT-Thread 操作系统提供了很好的 POSIX 标准支持，所以 `micropython-lib` 中很多模块可以在 RT-Thread MicroPython 上运行。

3.2.7 stream

也被称为文件类对象，一种对底层数据提供顺序读写访问的对象。`stream` 对象实现了对应的接口，包括 `read()`, `write()`, `readinto()`, `seek()`, `flush()`, `close()` 等方法。在 MicroPython 中，流是一个重要的概念，许多输入/输出对象都实现了流接口，因此可以在不同的上下文中一致地使用。更多关于 MicroPython 流的信息，可以参考 `uio`。

3.2.8 upip

字面意思是微型的 `pip` 工具。由 CPython 启发而开发的 MicroPython 包管理程序，但它要小的多，功能也更少。`upip` 可以在 MicroPython 的 Unix 移植上运行。由于 RT-Thread 操作系统提供了很好的 POSIX 标准支持，所以 `upip` 也可以运行在 RT-Thread MicroPython 上。使用 `upip` 工具可以在线下载 MicroPython 的扩展模块，并且自动下载其依赖的模块，为用户扩展 MicroPython 功能提供了很大的便利。

第 4 章

MicroPython 模块

- MicroPython 提供丰富的模块，每个模块提供特定的功能。了解开发的过程中一些常用的模块的使用方式，可以让你很好的使用 MicroPython 的功能。
- 这些模块可以通过 env 工具的 menuconfig 功能来开启和关闭，如果你需要使用特定的模块，在 menuconfig 中选中模块名，保存退出后，重新编译运行即可。

4.1 Python 标准库和微型库

Python 的标准库被“微型化”后，就是 MicroPython 标准库，也称 MicroPython 模块。它们仅仅提供了该模块的核心功能，用来替代 Python 标准库。一些模块使用 Python 标准库的名字，但是加上了前缀“u”，例如 `ujson` 代替 `json`。也就是说 MicroPython 的标准库 (微型库)，只实现了一部分模块功能。通过给这些库以不同的方式命名，用户可以写一个 Python 级的模块来扩展微型库的功能，以便于兼容 CPython 的标准库（这项工作就是 [micropython-lib](#) 项目的正在做的）。

在一些嵌入式平台上，可添加 Python 级别封装库从而实现命名兼容 CPython，使用 MicroPython 标准库既可使用他们的 u-name，也可以使用 non-u-name。使用 non-u-name 的模块可以被库路径文件夹里面的同名模块所覆盖。

例如，当 `import json` 时，首先会在库路径文件夹中搜索一个 `json.py` 文件或 `json` 目录进行加载。如果没有找到，它才会去加载内置 `ujson` 模块。

4.2 RT-Thread MicroPython 模块

4.2.1 常用内建模块

- `rtthread` – RT-Thread 系统相关函数
- `utime` – 时间相关函数
- `sys` – 系统特有功能函数
- `math` – 数学函数
- `uio` – 输入/输出流
- `ucollections` – 提供有用的集合类

- [ustruct](#) – 打包和解包原始数据类型
- [array](#) – 数字数据数组
- [gc](#) – 控制垃圾回收
- [uos](#) – 基本的“操作系统”服务
- [select](#) – 等待流事件
- [uctypes](#) – 以结构化的方式访问二进制数据
- [uerrno](#) – 系统错误码模块
- [_thread](#) – 多线程支持

4.2.2 硬件模块

- [machine](#) – 与硬件相关的功能
- [machine.Pin](#) - Pin 引脚控制类
- [machine.UART](#) - UART 外设控制类
- [machine.I2C](#) - I2C 外设控制类
- [machine.SPI](#) - SPI 外设控制类
- [machine.RTC](#) - RTC 外设控制类
- [machine.PWM](#) - PWM 外设控制类
- [machine.ADC](#) - ADC 外设控制类
- [machine.LCD](#) - LCD 外设控制类

4.2.3 网络模块

- [usocket](#) – 网络套接字模块
- [network](#) – 网络连接控制模块
- [network.WLAN](#) – WiFi 连接控制类

4.2.4 常用第三方模块

- [cmath](#) – 复数的数学函数
- [ubinascii](#) – 二进制/ ASCII 转换
- [uhashlib](#) – 哈希算法
- [uheapq](#) – 堆排序算法
- [ujson](#) – JSON 编码与解码
- [ure](#) – 正则表达式
- [uzlib](#) – zlib 解压缩
- [urandom](#) – 随机数生成模块

4.3 rtthread – 系统相关函数

rtthread 模块提供了与 RT-Thread 操作系统相关的功能，如查看栈使用情况等。

4.3.1 函数

4.3.1.1 rtthread.current_tid()

返回当前线程的 id 。

4.3.1.2 rtthread.is_preempt_thread()

返回是否是可抢占线程。

4.3.1.3 rtthread.stacks_analyze()

返回当前系统线程和栈使用信息。

4.3.2 示例

```
>>> import rtthread
>>>
>>> rtthread.is_preempt_thread()      # determine if code is running in a
    preemptible thread
True
>>> rtthread.current_tid()            # current thread id
268464956
>>> rtthread.stacks_analyze()         # show thread information
thread      pri  status      sp      stack size max used left tick  error
-----
eelog_async 31  suspend 0x000000a8 0x00000400    26%  0x00000003 000
tshell      20  ready   0x00000260 0x00001000    39%  0x00000003 000
tidle       31  ready   0x00000070 0x00000100    51%  0x0000000f 000
SysMonitor  30  suspend 0x000000a4 0x00000200    32%  0x00000005 000
timer       4   suspend 0x00000080 0x00000200    25%  0x00000009 000
>>>
```

4.4 utime – 时间相关函数

utime 模块提供获取当前时间和日期、测量时间间隔和延迟的功能。

初始时刻: Unix 使用 POSIX 系统标准，从 1970-01-01 00:00:00 UTC 开始。嵌入式程序从 2000-01-01 00:00:00 UTC 开始。

保持实际日历日期/时间: 需要一个实时时钟 (RTC)。在底层系统 (包括一些 RTOS 中)，RTC 已经包含在其中。设置时间是通过 OS/RTOS 而不是 MicroPython 完成，查询日期/时间也需要通过系统 API。对于裸板系统时钟依赖于 `machine.RTC()` 对象。设置时间通过 `machine.RTC().datetime(tuple)` 函数，并通过下面方式维持：

- 后备电池 (可能是选件、扩展板等)。

- 使用网络时间协议 (需要用户设置)。
- 每次上电时手工设置 (大部分只是在硬复位时需要设置, 少部分每次复位都需要设置)。

如果实际时间不是通过系统 / MicroPython RTC 维持, 那么下面函数结果可能不是和预期的相同。

4.4.1 函数

4.4.1.1 utime.localtime([secs])

从初始时间的秒转换为元组: (年, 月, 日, 时, 分, 秒, 星期, `yearday`)。如果 `secs` 是空或者 `None`, 那么使用当前时间。

- `year` 年份包括世纪 (例如 2014)。
- `month` 范围 1-12
- `day` 范围 1-31
- `hour` 范围 0-23
- `minute` 范围 0-59
- `second` 范围 0-59
- `weekday` 范围 0-6 对应周一到周日
- `yearday` 范围 1-366

4.4.1.2 utime.mktime()

时间的反函数, 它的参数是完整 8 参数的元组, 返回值一个整数自 2000 年 1 月 1 日以来的秒数。

4.4.1.3 utime.sleep(seconds)

休眠指定的时间 (秒), `Seconds` 可以是浮点数。注意有些版本的 MicroPython 不支持浮点数, 为了兼容可以使用 `sleep_ms()` 和 `sleep_us()` 函数。

4.4.1.4 utime.sleep_ms(ms)

延时指定毫秒, 参数不能小于 0。

4.4.1.5 utime.sleep_us(us)

延时指定微秒, 参数不能小于 0。

4.4.1.6 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()`。注: 直接在这些值上执行标准数学运算 (+, -) 或关系运算符 (<, >, >=) 会导致无效结果。执行数学运算然后传递结果作为参数给 `ticks_diff()` 或 `ticks_add()` 也将导致函数产生无效结果。

4.4.1.7 utime.ticks_us()

和上面 `ticks_ms()` 类似，只是返回微秒。

4.4.1.8 utime.ticks_cpu()

与 `ticks_ms()` 和 `ticks_us()` 类似，具有更高精度 (使用 CPU 时钟)，并非每个端口都实现此功能。

4.4.1.9 utime.ticks_add(ticks, delta)

给定一个数字作为节拍的偏移值 `delta`，这个数字的值是正数或者负数都可以。给定一个 `ticks` 节拍值，本函数允许根据节拍的模算数定义来计算给定节拍值之前或者之后 `delta` 个节拍的节拍值。`ticks` 参数必须是 `ticks_ms()`、`ticks_us()`，或 `ticks_cpu()` 函数的直接返回值。然而，`delta` 可以是一个任意整数或者是数字表达式。`ticks_add` 函数对计算事件/任务的截至时间很有用。(注意：必须使用 `ticksdiff()` 函数来处理最后期限)。

代码示例：

```
## 查找 100ms 之前的节拍值
print(utime.ticks_add(utime.ticks_ms(), -100))

## 计算操作的截止时间然后进行测试
deadline = utime.ticks_add(utime.ticks_ms(), 200)
while utime.ticks_diff(deadline, utime.ticks_ms()) > 0:
    do_a_little_of_something()

## 找出本次移植节拍值的最大值
print(utime.ticks_add(0, -1))
```

4.4.1.10 utime.ticks_diff(ticks1, ticks2)

计算两次调用 `ticksms()`、`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：

代码示例：

```
## 等待 GPIO 引脚有效，但是最多等待500微秒
start = time.ticks_us()
while pin.value() == 0:
    if time.ticks_diff(time.ticks_us(), start) > 500:
        raise TimeoutError
```

4.4.1.11 utime.time()

返回从开始时间的秒数 (整数)，假设 RTC 已经按照前面方法设置好。如果 RTC 没有设置，函数将返回参考点开始计算的秒数 (对于 RTC 没有后备电池的板子，上电或复位后的情况)。如果你开发便携版的

MicroPython 应用程序，你不要依赖函数来提供超过秒级的精度。如果需要高精度，使用 `ticks_ms()` 和 `ticks_us()` 函数。如果需要日历时间，使用不带参数的 `localtime()` 是更好选择。

!!! tip “与 CPython 的区别” 在 CPython 中，这个函数用浮点数返回从 Unix 开始时间（1970-01-01 00:00 UTC）的秒数，通常是毫秒级的精度。在 MicroPython 中，只有 Unix 版才使用相同开始时间，如果允许浮点精度，将返回亚秒精度。嵌入式硬件通常没有用浮点数表示长时间访问和亚秒精度，所以返回值是整数。一些嵌入式系统硬件不支持 RTC 电池供电方式，所以返回的秒数是从最后上电、或相对某个时间、以及特定硬件时间（如复位）。

4.4.2 示例

```
>>> import utime
>>> utime.sleep(1)           # sleep for 1 second
>>> utime.sleep_ms(500)      # sleep for 500 milliseconds
>>> utime.sleep_us(10)       # sleep for 10 microseconds
>>> start = utime.ticks_ms() # get value of millisecond counter
>>> delta = utime.ticks_diff(utime.ticks_ms(), start) # compute time difference
>>> delta
6928
>>> print(utime.ticks_add(utime.ticks_ms(), -100))
1140718
>>> print(utime.ticks_add(0, -1))
1073741823
```

更多内容可参考 `utime`。

4.5 sys – 系统特有功能函数

`sys` 模块提供系统特有的功能。

4.5.1 函数

4.5.1.1 sys.exit(retval=0)

终止当前程序给定的退出代码。函数会抛出 `SystemExit` 异常。#### `sys.print_exception(exc, file=sys.stdout)` 打印异常与追踪到一个类似文件的对象 `file` (或者缺省 `sys.stdout`)。

提示：这是 CPython 中回溯模块的简化版本。不同于 `traceback.print_exception()`，这个函数用异常值代替了异常类型、异常参数和回溯对象。文件参数在对应位置，不支持更多参数。CPython 兼容回溯模块在 `micropython-lib`。

4.5.2 常数

4.5.2.1 sys.argv

当前程序启动时参数的可变列表。

4.5.2.2 sys.byteorder

系统字节顺序 (“little” or “big”).

4.5.2.3 sys.implementation

关于当前 Python 实现的信息，对于 MicroPython 来说，有以下属性： - 名称 - ‘micropython’ - 版本 - 元组（主要，次要，小），比如（1, 9, 3）

4.5.2.4 sys.modules

已加载模块的字典。在一部分移植中，它可能不包含内置模块。

4.5.2.5 sys.path

用来搜索导入模块地址的列表。

4.5.2.6 sys.platform

返回当前平台的信息。

4.5.2.7 sys.stderr

标准错误流。

4.5.2.8 sys.stdin

标准输入流。

4.5.2.9 sys.stdout

标准输出流。

4.5.2.10 sys.version

符合的 Python 语言版本，如字符串。

4.5.2.11 sys.version_info

本次实现使用的 Python 语言版本，用一个元组的方式表示。

4.5.3 示例

```
>>> import sys
>>> sys.version
'3.4.0'
>>> sys.version_info
(3, 4, 0)
>>> sys.path
['', '/libs/mpy/']
>>> sys.__name__
'sys'
>>> sys.platform
'rt-thread'
>>> sys.byteorder
'little'
```

更多内容可参考 [sys](#)。

4.6 math – 数学函数

math 模块提供了对 C 标准定义的数学函数的访问。

注意：本模块需要带有硬件 FPU，精度是 32 位，这个模块需要浮点功能支持。

4.6.1 常数

4.6.1.1 math.e

自然对数的底数。

示例：

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

4.6.1.2 math.pi

圆周长与直径的比值。

示例：

```
>>> print(math.pi)
3.141593
```

4.6.2 函数

4.6.2.1 math.acos(x)

传入弧度值，计算 $\cos(x)$ 的反三角函数。

4.6.2.2 math.acosh(x)

返回 x 的逆双曲余弦。

4.6.2.3 math.asin(x)

传入弧度值，计算 $\sin(x)$ 的反三角函数。示例：

```
>>> x = math.asin(0.5)
>>> print(x)
0.5235988
```

4.6.2.4 math.asinh(x)

返回 x 的逆双曲正弦。

4.6.2.5 math.atan(x)

返回 x 的逆切线。

4.6.2.6 math.atan2(y, x)

Return the principal value of the inverse tangent of y/x .

4.6.2.7 math.atanh(x)

Return the inverse hyperbolic tangent of x .

4.6.2.8 math.ceil(x)

向上取整。示例：

```
>>> x = math.ceil(5.6454)
>>> print(x)
6
```

4.6.2.9 math.copysign(x, y)

Return x with the sign of y .

4.6.2.10 math.cos(x)

传入弧度值，计算余弦。示例：计算 $\cos 60^\circ$

```
>>> math.cos(math.radians(60))
0.5
```

4.6.2.11 math.cosh(x)

Return the hyperbolic cosine of x.

4.6.2.12 math.degrees(x)

弧度转化为角度。示例：

```
>>> x = math.degrees(1.047198)
>>> print(x)
60.00002
```

4.6.2.13 math.erf(x)

Return the error function of x.

4.6.2.14 math.erfc(x)

Return the complementary error function of x.

4.6.2.15 math.exp(x)

计算 e 的 x 次方（幂）。示例：

```
>>> x = math.exp(2)
>>> print(x)
7.389056
```

4.6.2.16 math.expm1(x)

计算 $\text{math.exp}(x) - 1$ 。

4.6.2.17 math.fabs(x)

计算绝对值。示例：

```
>>> x = math.fabs(-5)
>>> print(x)
5.0
>>> y = math.fabs(5.0)
>>> print(y)
5.0
```

4.6.2.18 math.floor(x)

向下取整。示例：

```
>>> x = math.floor(2.99)
>>> print(x)
2
>>> y = math.floor(-2.34)
>>> print(y)
-3
```

4.6.2.19 math.fmod(x, y)

取 x 除以 y 的模。示例：

```
>>> x = math.fmod(4, 5)
>>> print(x)
4.0
```

4.6.2.20 math.frexp(x)

Decomposes a floating-point number into its mantissa and exponent. The returned value is the tuple (m, e) such that $x == m * 2^e$ exactly. If $x == 0$ then the function returns (0.0, 0), otherwise the relation $0.5 \leq \text{abs}(m) < 1$ holds.

4.6.2.21 math.gamma(x)

返回伽马函数。示例：

```
>>> x = math.gamma(5.21)
>>> print(x)
33.08715
```

4.6.2.22 math.isfinite(x)

Return True if x is finite.

4.6.2.23 math.isinf(x)

Return True if x is infinite.

4.6.2.24 math.isnan(x)

Return True if x is not-a-number

4.6.2.25 math.ldexp(x, exp)

Return $x * (2^{**exp})$.

4.6.2.26 math.lgamma(x)

返回伽马函数的自然对数。示例：

```
>>> x = math.lgamma(5.21)
>>> print(x)
3.499145
```

4.6.2.27 math.log(x)

计算以 e 为底的 x 的对数。示例：

```
>>> x = math.log(10)
>>> print(x)
2.302585
```

4.6.2.28 math.log10(x)

计算以 10 为底的 x 的对数。示例：

```
>>> x = math.log10(10)
>>> print(x)
1.0
```

4.6.2.29 math.log2(x)

计算以 2 为底的 x 的对数。示例：

```
>>> x = math.log2(8)
>>> print(x)
3.0
```


4.6.2.30 math.modf(x)

Return a tuple of two floats, being the fractional and integral parts of x. Both return values have the same sign as x.

4.6.2.31 math.pow(x, y)

计算 x 的 y 次方（幂）。示例：

```
>>> x = math.pow(2, 3)
>>> print(x)
8.0
```

4.6.2.32 math.radians(x)

角度转化为弧度。示例：

```
>>> x = math.radians(60)
>>> print(x)
1.047198
```

4.6.2.33 math.sin(x)

传入弧度值，计算正弦。示例：计算 $\sin 90^\circ$

```
>>> math.sin(math.radians(90))
1.0
```

4.6.2.34 math.sinh(x)

Return the hyperbolic sine of x.

4.6.2.35 math.sqrt(x)

计算平方根。示例：

```
>>> x = math.sqrt(9)
>>> print(x)
3.0
```

4.6.2.36 math.tan(x)

传入弧度值，计算正切。示例：计算 $\tan 60^\circ$

```
>>> math.tan(math.radians(60))
1.732051
```

4.6.2.37 math.tanh(x)

Return the hyperbolic tangent of x.

4.6.2.38 math.trunc(x)

取整。示例：

```
>>> x = math.trunc(5.12)
>>> print(x)
5
>>> y = math.trunc(-6.8)
>>> print(y)
-6
```

更多内容可参考 [math](#)。

4.7 uio – 输入/输出流

uio 模块包含流类型 (类似文件) 对象和帮助函数。

4.7.1 函数

4.7.1.1 uio.open(name, mode='r', **kwargs)

打开一个文件，关联到内建函数`open()`。所有端口 (用于访问文件系统) 需要支持模式参数，但支持其他参数不同的端口。

4.7.2 类

4.7.2.1 class uio.FileIO(...)

这个文件类型用二进制方式打开文件，等于使用`open(name, "rb")`。不应直接使用这个实例。

4.7.2.2 class uio.TextIOWrapper(...)

这个类型以文本方式打开文件，等同于使用`open(name, "rt")`不应直接使用这个实例。

4.7.2.3 class uio.StringIO([string])

4.7.2.4 class uio.BytesIO([string])

内存文件对象。`StringIO` 用于文本模式 I/O (用“t”打开文件)，`BytesIO` 用于二进制方式 (用“b”方式)。文件对象的初始内容可以用字符串参数指定 (`stringio`用普通字符串，`bytesio`用`bytes`对象)。所有的文件方法，如 `read()`，`write()`，`seek()`，`flush()`，`close()` 都可以用在这些对象上，包括下面方法：

4.7.2.5 getvalue()

获取缓存区内容。

更多内容可参考 [uio](#)。

4.8 ucollections – 收集和容器类型

ucollections 模块实现了专门的容器数据类型，它提供了 Python 的通用内置容器的替代方案，包括了字典、列表、集合和元组。

4.8.1 类

4.8.1.1 ucollections.namedtuple(name, fields)

这是工厂函数创建一个新的 **namedtuple** 型与一个特定的字段名称和集合。**namedtuple** 是元组允许子类要访问它的字段不仅是数字索引，而且还具有属性使用符号字段名访问语法。字段是字符串序列指定字段名称。为了兼容的实现也可以用空间分隔的字符串命名的字段（但效率较低）。

代码示例：

```
from ucollections import namedtuple

MyTuple = namedtuple("MyTuple", ("id", "name"))
t1 = MyTuple(1, "foo")
t2 = MyTuple(2, "bar")
print(t1.name)
assert t2.name == t2[1]
ucollections.OrderedDict(...)
```

4.8.1.2 ucollections.OrderedDict(...)

字典类型的子类，会记住并保留键/值的追加顺序。当有序的字典被迭代输出时，键/值会按照他们被添加的顺序返回：

```
from ucollections import OrderedDict

# To make benefit of ordered keys, OrderedDict should be initialized
# from sequence of (key, value) pairs.
d = OrderedDict([("z", 1), ("a", 2)])
# More items can be added as usual
d["w"] = 5
d["b"] = 3
for k, v in d.items():
    print(k, v)
```

输出：

z 1 a 2 w 5 b 3

更多的内容可参考 [ucollections](#) 。

4.9 `ustruct` – 打包和解包原始数据类型

`ustruct` 模块在 Python 值和以 Python 字节对象表示的 C 结构之间执行转换。

- 支持 size/byte 的前缀: @, <, >, !.
- 支持的格式代码: b, B, h, H, i, I, l, L, q, Q, s, P, f, d (最后 2 个需要支持浮点数).

4.9.1 函数

4.9.1.1 `ustruct.calcsize(fmt)`

返回存放某一类型数据 `fmt` 需要的字节数。

`fmt`: 数据类型

- `b` — 字节型
- `B` — 无符号字节型
- `h` — 短整型
- `H` — 无符号短整型
- `i` — 整型
- `I` — 无符号整型
- `l` — 整型
- `L` — 无符号整型
- `q` — 长整型
- `Q` — 无符号长整型
- `f` — 浮点型
- `d` — 双精度浮点型
- `P` — 无符号型

示例:

```
>>> print(struct.calcsize("i"))
4
>>> print(struct.calcsize("B"))
1
```

4.9.1.2 `ustruct.pack(fmt, v1, v2, ...)`

按照格式字符串 `fmt` 打包参数 `v1, v2, ...`。返回值是参数打包后的字节对象。

`fmt`: 同上

示例:

```
>>> struct.pack("ii", 3, 2)
b'\x03\x00\x00\x00\x02\x00\x00\x00'
```

4.9.1.3 struct.unpack(fmt, data)

从 `fmt` 中解包数据。返回值是解包后参数的元组。

data: 要解压的字节对象

示例:

```
>>> buf = struct.pack("bb", 1, 2)
>>> print(buf)
b'\x01\x02'
>>> print(struct.unpack("bb", buf))
(1, 2)
```

4.9.1.4 struct.pack_into(fmt, buffer, offset, v1, v2, ...)

按照格式字符串 `fmt` 压缩参数 `v1, v2, ...` 到缓冲区 `buffer`, 开始位置是 `offset`。当 `offset` 为负数时, 从缓冲区末尾开始计数。

4.9.1.5 struct.unpack_from(fmt, data, offset=0)

以 `fmt` 作为规则从 `data` 的 `offset` 位置开始解包数据, 如果 `offset` 是负数就是从缓冲区末尾开始计算。返回值是解包后的参数元组。

```
>>> buf = struct.pack("bb", 1, 2)
>>> print(struct.unpack("bb", buf))
(1, 2)
>>> print(struct.unpack_from("b", buf, 1))
(2,)
```

更多的内容可参考 [struct](#)。

4.10 array – 数字数据数组

`array` 模块定义了一个对象类型, 它可以简洁地表示基本值的数组: 字符、整数、浮点数。支持代码格式: `b, B, h, H, i, I, l, L, q, Q, f, d` (最后 2 个需要支持浮点数)。

4.10.1 构造函数

4.10.1.1 class array.array(typecode[, iterable])

用给定类型的元素创建数组。数组的初始内容由 `iterable` 提供, 如果没有提供, 则创建一个空数组。

`typecode`: 数组的类型
`iterable`: 数组初始内容

示例:

```
>>> import array
>>> a = array.array('i', [2, 4, 1, 5])
>>> b = array.array('f')
>>> print(a)
array('i', [2, 4, 1, 5])
>>> print(b)
array('f')
```

4.10.2 方法

4.10.2.1 `array.append(val)`

将一个新元素追加到数组的末尾。

示例:

```
>>> a = array.array('f', [3, 6])
>>> print(a)
array('f', [3.0, 6.0])
>>> a.append(7.0)
>>> print(a)
array('f', [3.0, 6.0, 7.0])
```

4.10.2.2 `array.extend(iterable)`

将一个新的数组追加到数组的末尾，注意追加的数组和原来数组的数据类型要保持一致。

示例:

```
>>> a = array.array('i', [1, 2, 3])
>>> b = array.array('i', [4, 5])
>>> a.extend(b)
>>> print(a)
array('i', [1, 2, 3, 4, 5])
```

更多内容可参考 [array](#)。

4.11 gc – 控制垃圾回收

`gc` 模块提供了垃圾收集器的控制接口。

4.11.1 函数

4.11.1.1 `gc.enable()`

允许自动回收内存碎片。

4.11.1.2 `gc.disable()`

禁止自动回收，但可以通过 `collect()` 函数进行手动回收内存碎片。

4.11.1.3 `gc.collect()`

运行一次垃圾回收。

4.11.1.4 `gc.mem_alloc()`

返回已分配的内存数量。

4.11.1.5 `gc.mem_free()`

返回剩余的内存数量。

更多内容可参考 [gc](#)。

4.12 `machine` – 与硬件相关的功能

`machine` 模块包含与特定开发板上的硬件相关的特定函数。在这个模块中的大多数功能允许实现直接和不受限制地访问和控制系统上的硬件块（如 CPU，定时器，总线等）。如果使用不当，会导致故障，死机，崩溃，在极端的情况下，硬件会损坏。

需要注意的是，由于不同开发板的硬件资源不同，MicroPython 移植所能控制的硬件也是不一样的。因此对于控制硬件的例程来说，在使用前需要修改相关的配置参数来适配不同的开发板，或者直接运行已经对某一开发板适配好的 MicroPython 示例程序。本文档中的例程都是基于 RT-Thread IoT Board 潘多拉开发板而讲解的。

4.12.1 函数

4.12.1.1 复位相关函数

4.12.1.1.1 `machine.info()` 显示关于系统介绍和内存占用等信息。

4.12.1.1.2 `machine.reset()` 重启设备，类似于按下复位按钮。

4.12.1.1.3 `machine.reset_cause()` 获得复位的原因，查看可能的返回值的常量。

4.12.1.2 中断相关函数

4.12.1.2.1 machine.disable_irq() 禁用中断请求。返回先前的 `IRQ` 状态，该状态应该被认为是一个未知的值。这个返回值应该在 `disable_irq` 函数被调用之前被传给 `enable_irq` 函数来重置中断到初始状态。

4.12.1.2.2 machine.enable_irq(state) 重新使能中断请求。状态参数应该是从最近一次禁用功能的调用中返回的值。

4.12.1.3 功耗相关函数

4.12.1.3.1 machine.freq() 返回 `CPU` 的运行频率。

4.12.1.3.2 machine.idle() 阻断给 `CPU` 的时钟信号，在较短或者较长的周期里减少功耗。当中断发生时，外设将继续工作。

4.12.1.3.3 machine.sleep() 停止 `CPU` 并禁止除了 `WLAN` 之外的所有外设。系统会从睡眠请求的地方重新恢复工作。为了确保唤醒一定会发生，应当首先配置中断源。

4.12.1.3.4 machine.deepsleep() 停止 `CPU` 和所有外设（包括网络接口）。执行从主函数中恢复，就像被复位一样。复位的原因可以检查 `machine.DEEPSLEEP` 参数获得。为了确保唤醒一定会发生，应该首先配置中断源，比如一个引脚的变换或者 `RTC` 的超时。

4.12.2 常数

4.12.2.1 machine.IDLE

4.12.2.2 machine.SLEEP

4.12.2.3 machine.DEEPSLEEP

`IRQ` 的唤醒值。

4.12.2.4 machine.PWRON_RESET

4.12.2.5 machine.HARD_RESET

4.12.2.6 machine.WDT_RESET

4.12.2.7 machine.DEEPSLEEP_RESET

4.12.2.8 machine.SOFT_RESET

复位的原因。

4.12.2.9 machine.WLAN_WAKE**4.12.2.10 machine.PIN_WAKE****4.12.2.11 machine.RTC_WAKE**

唤醒的原因。

4.12.3 类**4.12.3.1 class Pin - 控制 I/O 引脚****4.12.3.2 class I2C - I2C 协议****4.12.3.3 class SPI - SPI 协议****4.12.3.4 class UART - 串口****4.12.3.5 class LCD - LCD****4.12.3.6 class RTC - RTC****4.12.3.7 class PWM - PWM****4.12.3.8 class ADC - ADC****4.12.3.9 class WDT - 看门狗****4.12.3.10 class TIMER - 定时器****4.12.4 示例**

```
>>> import machine
>>>
>>> machine.info()           # show information about the board
-----
RT-Thread
-----
total memory: 131048
used memory : 4920
maximum allocated memory: 5836
thread      pri  status      sp      stack size max used left tick  error
-----
elog_async  31  suspend 0x000000a8 0x00000400      26% 0x00000003 000
tshell      20  ready  0x0000019c 0x00001000      39% 0x00000006 000
tidle       31  ready  0x0000006c 0x00000100      50% 0x0000000b 000
SysMonitor  30  suspend 0x000000a8 0x00000200      32% 0x00000005 000
timer       4   suspend 0x0000007c 0x00000200      24% 0x00000009 000
-----
```

```

qstr:
  n_pool=0
  n_qstr=0
  n_str_data_bytes=0
  n_total_bytes=0
-----
GC:
  16064 total
  464 : 15600
  1=14 2=6 m=3
>>> machine.enable_irq()          # enable interrupt
>>> machine.disable_irq()         # disable interrupt, WARNING: this operation is
    dangerous
>>> machine.reset()               # hard reset, like push RESET button

```

更多内容可参考 [machine](#) 。

4.13 machine.Pin

machine.Pin 类是 `machine` 模块下面的一个硬件类，用于对引脚的配置和控制，提供对 **Pin** 设备的操作方法。

Pin 对象用于控制输入/输出引脚（也称为 **GPIO**）。**Pin** 对象通常与一个物理引脚相关联，他可以驱动输出电压和读取输入电压。**Pin** 类中有设置引脚模式（输入/输出）的方法，也有获取和设置数字逻辑（0 或 1）的方法。

一个 **Pin** 对象是通过一个标识符来构造的，它明确地指定了一个特定的输入输出。标识符的形式和物理引脚的映射是特定于一次移植的。标识符可以是整数，字符串或者是一个带有端口和引脚号码的元组。在 RT-Thread MicroPython 中，引脚标识符是一个由代号和引脚号组成的元组，如 `Pin(("PB15", 31), Pin.OUT_PP)` 中的 `("PB15", 31)`。

4.13.1 构造函数

在 RT-Thread MicroPython 中 **Pin** 对象的构造函数如下：

4.13.1.1 class machine.Pin(id, mode = -1, pull = -1, value)

- **id**：由用户自定义的引脚名和 **Pin** 设备引脚号组成，如 `("PB15", 31)`，“PB15”为用户自定义的引脚名，31 为 RT-Thread **Pin** 设备驱动在本次移植中的引脚号。
- **mode**：指定引脚模式，可以是以下几种：
 - **Pin.IN**：输入模式
 - **Pin.OUT**：输出模式
 - **Pin.OPEN_DRAIN**：开漏模式
- **pull**：如果指定的引脚连接了上拉下拉电阻，那么可以配置成下面的状态：

- **None** : 没有上拉或者下拉电阻。
 - **Pin.PULL_UP** : 使能上拉电阻。
 - **Pin.PULL_DOWN** : 使能下拉电阻。
- **value** : **value** 的值只对输出模式和开漏输出模式有效, 用来设置初始输出值。

4.13.2 方法

4.13.2.1 Pin.init(mode= -1, pull= -1, *, value, drive, alt)

根据输入的参数重新初始化引脚。只有那些被指定的参数才会被设置, 其余引脚的状态将保持不变, 详细的参数可以参考上面的构造函数。

4.13.2.2 Pin.value([x])

如果没有给定参数 **x**, 这个方法可以获得引脚的值。如果给定参数 **x**, 如 **0** 或 **1**, 那么设置引脚的值为逻辑 **0** 或逻辑 **1**。

4.13.2.3 Pin.name()

返回引脚对象在构造时用户自定义的引脚名。

4.13.3 常量

下面的常量用来配置 **Pin** 对象。

4.13.3.1 选择引脚模式:

4.13.3.1.1 Pin.IN

4.13.3.1.2 Pin.OUT

4.13.3.1.3 Pin.OPEN_DRAIN

4.13.3.2 选择上/下拉模式:

4.13.3.2.1 Pin.PULL_UP

4.13.3.2.2 Pin.PULL_DOWN

4.13.3.2.3 None 使用值 **None** 代表不进行上下拉。

4.13.4 示例

```
from machine import Pin

PIN_OUT = 31
PIN_IN  = 58

p_out = Pin(("PB15", PIN_OUT), Pin.OUT_PP)
p_out.value(1)          # set io high
p_out.value(0)          # set io low

p_in = Pin(("key_0", PIN_IN), Pin.IN, Pin.PULL_UP)
print(p_in.value() )    # get value, 0 or 1
```

更多内容可参考 [machine.Pin](#) 。

4.14 machine.I2C

machine.I2C 类是 **machine** 模块下面的一个硬件类，用于对 **I2C** 的配置和控制，提供对 **I2C** 设备的操作方法。

- **I2C** 是一种用于设备间通信的两线协议。在物理层上，它由两根线组成：**SCL** 和 **SDA**，即时钟和数据线。
- **I2C** 对象被创建到一个特定的总线上，它们可以在创建时被初始化，也可以之后再初始化。
- 打印 **I2C** 对象会打印出配置时的信息。

4.14.1 构造函数

在 RT-Thread MicroPython 中 **I2C** 对象的构造函数如下：

4.14.1.1 class machine.I2C(id= -1, scl, sda, freq=400000)

使用下面的参数构造并返回一个新的 **I2C** 对象：

- **id**：标识特定的 **I2C** 外设。如果填入 **id = -1**，即选择软件模拟的方式实现 **I2C**，这时可以使用任意引脚来模拟 **I2C** 总线，这样在初始化时就必须指定 **scl** 和 **sda**。软件 **I2C** 的初始化方式可参考软件 **I2C** 示例。硬件 **I2C** 的初始化方式可参考硬件 **I2C** 示例。
- **scl**：应该是一个 **Pin** 对象，指定为一个用于 **scl** 的 **Pin** 对象。
- **sda**：应该是一个 **Pin** 对象，指定为一个用于 **sda** 的 **Pin** 对象。
- **freq**：应该是为 **scl** 设置的最大频率。

4.14.2 方法

4.14.2.1 I2C.init(scl, sda, freq=400000)

初始化 I2C 总线，参数介绍可以参考构造函数中的参数。

4.14.2.2 I2C.deinit()

关闭 I2C 总线。

4.14.2.3 I2C.scan()

扫描所有 0x08 和 0x77 之间的 I2C 地址，然后返回一个有响应地址的列表。如果一个设备在总线上收到了他的地址，就会通过拉低 SDA 的方式来响应。

4.14.3 I2C 基础方法

下面的方法实现了基本的 I2C 总线操作，可以组合成任何的 I2C 通信操作，如果需要对总线进行更多的控制，可以使用他们，否则可以使用后面介绍的标准使用方法。

4.14.3.1 I2C.start()

在总线上产生一个启动信号。（SCL 为高时，SDA 转换为低）

4.14.3.2 I2C.stop()

在总线上产生一个停止信号。（SCL 为高时，SDA 转换为高）

4.14.3.3 I2C.readinto(buf, nack=True)

从总线上读取字节并将他们存储到 buf 中，读取的字节数时 buf 的长度。在收到最后一个字节以外的所有内容后，将在总线上发送 ACK。在收到最后一个字节之后，如果 NACK 是正确的，那么就会发送一个 NACK，否则将会发送 ACK。

4.14.3.4 I2C.write(buf)

将 buf 中的数据接入到总线，检查每个字节之后是否收到 ACK，并在收到 NACK 时停止传输剩余的字节。这个函数返回接收到的 ACK 的数量。

4.14.4 I2C 标准总线操作

下面的方法实现了标准 I2C 主设备对一个给定从设备的读写操作。

4.14.4.1 I2C.readfrom(addr, nbytes, stop=True)

从 `addr` 指定的从设备中读取 `n` 个字节，如果 `stop = True`，那么在传输结束时会产生一个停止信号。函数会返回一个存储着读到数据的字节对象。

4.14.4.2 I2C.readfrom_into(addr, buf, stop=True)

从 `addr` 指定的从设备中读取数据存储在 `buf` 中，读取的字节数将是 `buf` 的长度，如果 `stop = True`，那么在传输结束时会产生一个停止信号。这个方法没有返回值。

4.14.4.3 I2C.writeto(addr, buf, stop=True)

将 `buf` 中的数据写入到 `addr` 指定的从设备中，如果在写的过程中收到了 `NACK` 信号，那么就不会发送剩余的字节。如果 `stop = True`，那么在传输结束时会产生一个停止信号，即使收到一个 `NACK`。这个函数返回接收到的 `ACK` 的数量。

4.14.5 内存操作

一些 `I2C` 设备充当一个内存设备，可以读取和写入。在这种情况下，有两个与 `I2C` 相关的地址，从机地址和内存地址。下面的方法是与这些设备进行通信的便利函数。

4.14.5.1 I2C.readfrom_mem(addr, memaddr, nbytes, *, addrsize=8)

从 `addr` 指定的从设备中 `memaddr` 地址开始读取 `n` 个字节。`addrsize` 参数指定地址的长度。返回一个存储读取数据的字节对象。

4.14.5.2 I2C.readfrom_mem_into(addr, memaddr, buf, *, addrsize=8)

从 `addr` 指定的从设备中 `memaddr` 地址读取数据到 `buf` 中，读取的字节数是 `buf` 的长度。这个方法没有返回值。

4.14.5.3 I2C.writeto_mem(addr, memaddr, buf, *, addrsize=8)

将 `buf` 里的数据写入 `addr` 指定的从机的 `memaddr` 地址中。这个方法没有返回值。

4.14.6 示例

4.14.6.1 软件模拟 I2C

```
>>> from machine import Pin, I2C
>>> clk = Pin(("clk", 29), Pin.OUT_OD) # Select the 29 pin device as the clock
>>> sda = Pin(("sda", 30), Pin.OUT_OD) # Select the 30 pin device as the data line
>>> i2c = I2C(-1, clk, sda, freq=100000) # create I2C peripheral at frequency of 100
kHz
```

```
>>> i2c.scan()                                # scan for slaves, returning a list of 7-bit
      addresses
[81]                                           # Decimal representation
>>> i2c.writeto(0x51, b'123')                 # write 3 bytes to slave with 7-bit address 42
3
>>> i2c.readfrom(0x51, 4)                     # read 4 bytes from slave with 7-bit address
      42
b'\xf8\xc0\xc0\xc0'
>>> i2c.readfrom_mem(0x51, 0x02, 1)          # read 1 bytes from memory of slave 0x51(7-bit
      ),
b'\x12'                                       # starting at memory-address 8 in the slave
>>> i2c.writeto_mem(0x51, 2, b'\x10')        # write 1 byte to memory of slave 42,
      # starting at address 2 in the slave
```

4.14.6.2 硬件 I2C

需要先开启 I2C 设备驱动，查找设备可以在 `msh` 中输入 `list_device` 命令。在构造函数的第一个参数传入 `0`，系统就会搜索名为 `i2c0` 的设备，找到之后使用这个设备来构建 `I2C` 对象：

```
>>> from machine import Pin, I2C
>>> i2c = I2C(0)                             # create I2C peripheral at frequency of 100kHz
>>> i2c.scan()                                # scan for slaves, returning a list of 7-bit
      addresses
[81]                                           # Decimal representation
```

更多内容可参考 [machine.I2C](#)。

4.15 machine.SPI

`machine.SPI` 类是 `machine` 模块下面的一个硬件类，用于对 SPI 的配置和控制，提供对 SPI 设备的操作方法。

- SPI 是一个由主机驱动的同步串行协议。在物理层，总线有三根：`SCK`、`MOSI`、`MISO`。多个设备可以共享同一总线，每个设备都由一个单独的信号 `SS` 来选中，也称片选信号。
- 主机通过片选信号选定一个设备进行通信。`SS` 信号的管理应该由用户代码负责。（通过 `machine.Pin`）

4.15.1 构造函数

在 RT-Thread MicroPython 中 SPI 对象的构造函数如下：

4.15.1.1 class machine.SPI(id, ...)

在给定总线上构造一个 SPI 对象，`id` 取决于特定的移植。

如果想要使用软件 SPI，即使用引脚模拟 SPI 总线，那么初始化的第一个参数需要设置为 `-1`，可参考软件 SPI 示例。

使用硬件 SPI 在初始化时只需传入 SPI 设备的编号即可，如 '50' 表示 SPI5 总线上的第 0 个设备。初始化方式可参考硬件 SPI 示例。

如果没有额外的参数，SPI 对象会被创建，但是不会被初始化，如果给出额外的参数，那么总线将被初始化，初始化参数可以参考下面的 SPI.init 方法。

4.15.2 方法

4.15.2.1 SPI.init(baudrate=1000000, *, polarity=0, phase=0, bits=8, firstbit=SPI.MSB, sck=None, mosi=None, miso=None)

用给定的参数初始化SPI总线：

- **baudrate** : SCK 时钟频率。
- **polarity** : 极性可以是 0 或 1，是时钟空闲时所处的电平。
- **phase** : 相位可以是 0 或 1，分别在第一个或者第二个时钟边缘采集数据。
- **bits** : 每次传输的数据长度，一般是 8 位。
- **firstbit** : 传输数据从高位开始还是从低位开始，可以是 SPI.MSB 或者 SPI.LSB。
- **sck** : 用于 sck 的 machine.Pin 对象。
- **mosi** : 用于 mosi 的 machine.Pin 对象。
- **miso** : 用于miso 的 machine.Pin 对象。

4.15.2.2 SPI.deinit()

关闭 SPI 总线。

4.15.2.3 SPI.read(nbytes, write=0x00)

读出 n 字节的同时不断的写入 write 给定的单字节。返回一个存放着读出数据的字节对象。

4.15.2.4 SPI.readinto(buf, write=0x00)

读出 n 字节到 buf 的同时不断地写入 write 给定的单字节。这个方法返回读入的字节数。

4.15.2.5 SPI.write(buf)

写入 buf 中包含的字节。返回None。

4.15.2.6 SPI.write_readinto(write_buf, read_buf)

在读出数据到 readbuf 时，从 writebuf 中写入数据。缓冲区可以是相同的或不同，但是两个缓冲区必须具有相同的长度。返回 None。

4.15.3 常量

4.15.3.1 SPI.MASTER

用于初始化 SPI 总线为主机。

4.15.3.2 SPI.MSB

设置从高位开始传输数据。

4.15.3.3 SPI.LSB

设置从低位开始传输数据。

4.15.4 示例

4.15.4.1 软件模拟 SPI

```
>>> from machine import Pin, SPI
>>> clk = Pin(("clk", 26), Pin.OUT_PP)
>>> mosi = Pin(("mosi", 27), Pin.OUT_PP)
>>> miso = Pin(("miso", 28), Pin.IN)
>>> spi = SPI(-1, 500000, polarity = 0, phase = 0, bits = 8, firstbit = 0, sck = clk
, mosi = mosi, miso = miso)
>>> print(spi)
SoftSPI(baudrate=500000, polarity=0, phase=0, sck=clk, mosi=mosi, miso=miso)
>>> spi.write("hello rt-thread!")
>>> spi.read(10)
b'\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00'
```

4.15.4.2 硬件 SPI

需要先开启 SPI 设备驱动，查找设备可以在 msh 中输入 `list_device` 命令。在构造函数的第一个参数传入 50，系统就会搜索名为 `spi50` 的设备，找到之后使用这个设备来构建 SPI 对象：

```
>>> from machine import SPI
>>> spi = SPI(50)
>>> print(spi)
SPI(device port : spi50)
>>> spi.write(b'\x9f')
>>> spi.read(5)
b'\xff\xff\xff\xff'
>>> buf = bytearray(1)
>>> spi.write_readinto(b"\x9f",buf)
>>> buf
bytearray(b'\xef')
>>> spi.init(100000,0,0,8,1)      # Resetting SPI parameter
```

更多内容可参考 [machine.SPI](#)。

4.16 machine.UART

machine.UART 类是 `machine` 模块下面的一个硬件类，用于对 UART 的配置和控制，提供对 UART 设备的操作方法。

UART 实现了标准的 `uart/usart` 双工串行通信协议，在物理层上，他由两根数据线组成：**RX** 和 **TX**。通信单元是一个字符，它可以是 8 或 9 位宽。

4.16.1 构造函数

在 RT-Thread MicroPython 中 **UART** 对象的构造函数如下：

4.16.1.1 class machine.UART(id, ...)

在给定总线上构造一个 **UART** 对象，**id** 取决于特定的移植。初始化参数可以参考下面的 **UART.init** 方法。

使用硬件 UART 在初始化时只需传入 **UART** 设备的编号即可，如传入 **1** 表示 `uart1` 设备。初始化方式可参考示例。

4.16.2 方法

4.16.2.1 UART.init(baudrate = 9600, bits=8, parity=None, stop=1)

- **baudrate** : `SCK` 时钟频率。
- **bits** : 每次发送数据的长度。
- **parity** : 校验方式。
- **stop** : 停止位的长度。

4.16.2.2 UART.deinit()

关闭串口总线。

4.16.2.3 UART.read([nbytes])

读取字符，如果指定读 `n` 个字节，那么最多读取 `n` 个字节，否则就会读取尽可能多的数据。返回值：一个包含读入数据的字节对象。如果如果超时则返回 `None`。

4.16.2.4 UART.readinto(buf[, nbytes])

读取字符到 `buf` 中，如果指定读 `n` 个字节，那么最多读取 `n` 个字节，否则就读取尽可能多的数据。另外读取数据的长度不超过 `buf` 的长度。返回值：读取和存储到 `buf` 中的字节数。如果超时则返回 `None`。

4.16.2.5 UART.readline()

读一行数据，以换行符结尾。返回值：读入的行数，如果超时则返回 `None`。

4.16.2.6 UART.write(buf)

将 `buf` 中的数据写入总线。返回值：写入的字节数，如果超时则返回 `None`。

4.16.3 示例

在构造函数的第一个参数传入1，系统就会搜索名为 `uart1` 的设备，找到之后使用这个设备来构建 `UART` 对象：

```
from machine import UART
uart = UART(1, 115200) # init with given baudrate
uart.init(115200, bits=8, parity=None, stop=1) # init with given parameters
uart.read(10) # read 10 characters, returns a bytes object
uart.read() # read all available characters
uart.readline() # read a line
uart.readinto(buf) # read and store into the given buffer
uart.write('abc') # write the 3 characters
```

更多内容可参考 `machine.UART`。

4.17 machine.LCD

`machine.LCD` 类是 `machine` 模块下面的一个硬件类，用于对 LCD 的配置和控制，提供对 LCD 设备的操作方法。

IoT board 板载一块 1.3 寸，分辨率为 240*240 的 LCD 显示屏，因此对该屏幕操作时，(x, y) 坐标的范围是 0 - 239。

4.17.1 构造函数

在 RT-Thread MicroPython 中 `LCD` 对象的构造函数如下：

4.17.1.1 class machine.LCD()

在给定总线上构造一个 `LCD` 对象，无入参，初始化的对象取决于特定硬件，初始化方式可参考示例。

4.17.2 方法

4.17.2.1 LCD.light(value)

控制是否开启 LCD 背光，入参为 `True` 则打开 LCD 背光，入参为 `False` 则关闭 LCD 背光。

4.17.2.2 LCD.fill(color)

根据给定的颜色填充整个屏幕，支持多种颜色，可以传入的参数有：

WHITE BLACK BLUE BRED GRED GBLUE RED MAGENTA GREEN CYAN YELLOW BROWN BRRED GRAY
GRAY175 GRAY151 GRAY240

详细的使用方法可参考示例。

4.17.2.3 LCD.pixel(x, y, color)

向指定的位置 (x, y) 画点，点的颜色为 color 指定的颜色，可指定的颜色和上一个功能相同。

注意：(x, y) 坐标的范围是 0 - 239，使用下面的方法对坐标进行操作时同样需要遵循此限制。

4.17.2.4 LCD.text(str, x, y, size)

在指定的位置 (x,y) 写入字符串，字符串由 str 指定，字体的大小由 size 指定，size 的大小可为 16, 24, 32。

4.17.2.5 LCD.line(x1, y1, x2, y2)

在 LCD 上画一条直线，起始地址为 (x1, y1)，终点为 (x2, y2)。

4.17.2.6 LCD.rectangle(x1, y1, x2, y2)

在 LCD 上画一个矩形，左上角的位置为 (x1, y1)，右下角的位置为 (x2, y2)。

4.17.2.7 LCD.circle(x1, y1, r)

在 LCD 上画一个圆形，圆心的位置为 (x1, y1)，半径长度为 r。

4.17.3 示例

```
from machine import LCD      # 从 machine 导入 LCD 类
lcd = LCD()                  # 创建一个 lcd 对象
lcd.light(False)             # 关闭背光
lcd.light(True)              # 打开背光
lcd.fill(lcd.BLACK)          # 将整个 LCD 填充为黑色
lcd.fill(lcd.RED)            # 将整个 LCD 填充为红色
lcd.fill(lcd.GRAY)           # 将整个 LCD 填充为灰色
lcd.fill(lcd.WHITE)          # 将整个 LCD 填充为白色
lcd.pixel(50, 50, lcd.BLUE)  # 将 (50,50) 位置的像素填充为蓝色
lcd.text("hello RT-Thread", 0, 0, 16)  # 在 (0, 0) 位置以 16 字号打印字符串
lcd.text("hello RT-Thread", 0, 16, 24)  # 在 (0, 16) 位置以 24 字号打印字符串
lcd.text("hello RT-Thread", 0, 48, 32)  # 在 (0, 48) 位置以 32 字号打印字符串
lcd.line(0, 50, 239, 50)     # 以起点 (0, 50)，终点 (239, 50) 画一条线
```

```

lcd.line(0, 50, 239, 50)    # 以起点 (0, 50), 终点 (239, 50) 画一条线
lcd.rectangle(100, 100, 200, 200) # 以左上角为 (100,100), 右下角 (200,200) 画矩形
lcd.circle(150, 150, 80)    # 以圆心位置 (150,150), 半径为 80 画圆

```

4.18 machine.RTC

machine.RTC 类是 **machine** 模块下面的一个硬件类，用于对指定 RTC 设备的配置和控制，提供对 RTC 设备的操作方法。

- RTC (Real-Time Clock) 实时时钟可以提供精确的实时时间，它可以用于产生年、月、日、时、分、秒等信息。

4.18.1 构造函数

在 RT-Thread MicroPython 中 **RTC** 对象的构造函数如下：

4.18.1.1 class machine.RTC()

所以在给定的总线上构造一个 **RTC** 对象，无入参对象，使用方式可参考示例。

4.18.2 方法

4.18.2.1 RTC.init(datetime)

根据传入的参数初始化 RTC 设备起始时间。入参 **datetime** 为一个时间元组，格式如下：

```
(year, month, day, wday, hour, minute, second, yday)
```

参数介绍如下所示：

- **year**: 年份；
- **month**: 月份，范围 [1, 12]；
- **day**: 日期，范围 [1, 31]；
- **wday**: 星期，范围 [0, 6]，0 表示星期一，以此类推；
- **hour**: 小时，范围 [0, 23]；
- **minute**: 分钟，范围 [0, 59]；
- **second**: 秒，范围 [0, 59]；
- **yday**: 从当前年份 1 月 1 日开始的天数，范围 [0, 365]，一般置位 0 未实现。

使用的方式可参考示例。

4.18.2.2 RTC.deinit()

重置 RTC 设备时间到 2015 年 1 月 1 日，重新运行 RTC 设备。

4.18.2.3 RTC.now()

获取当前时间，返回值为上述 `datetime` 时间元组格式。

4.18.3 示例

```
>>> from machine import RTC
>>> rtc = RTC()                # 创建 RTC 设备对象
>>> rtc.init((2019,6,5,2,10,22,30,0)) # 设置初始化时间
>>> rtc.now()                  # 获取当前时间
(2019, 6, 5, 2, 10, 22, 40, 0)
>>> rtc.deinit()              # 重置时间到2015年1月1日
>>> rtc.now()                  # 获取当前时间
(2015, 1, 1, 3, 0, 0, 1, 0)
```

4.19 machine.PWM

`machine.PWM` 类是 `machine` 模块下的一个硬件类，用于指定 PWM 设备的配置和控制，提供对 PWM 设备的操作方法。

- PWM (Pulse Width Modulation，脉冲宽度调制) 是一种对模拟信号电平进行数字编码的方式；
- PWM 设备可以通过调节有效电平在一个周期信号中的比例时间来操作设备；
- PWM 设备两个重要的参数：频率 (`freq`) 和占空比 (`duty`)；
 - 频率：从一个上升沿（下降沿）到下一个上升沿（下降沿）的时间周期，单位为 Hz；
 - 占空比：有效电平（通常为电平）在一个周期内的时间比例；

4.19.1 构造函数

在 RT-Thread MicroPython 中 `PWM` 对象的构造函数如下：

4.19.1.1 class machine.PWM(id, channel, freq, duty)

在给定的总线上构建一个 `PWM` 对象，参数介绍如下：

- `id`：使用的 PWM 设备编号，如 `id = 1` 表示编号为 1 的 PWM 设备；
- `channel`：使用的 PWM 设备通道号，每个 PWM 设备包含多个通道，范围为 [0, 4]；
- `freq`：初始化频率，范围 [1, 156250]；
- `duty`：初始化占空比数值，范围 [0 255]；

例如：`PWM(1,4,100,100)` 表示当前使用编号为 1 的 PWM 设备的 4 通道，初始化频率为 1000 Hz，初始化占空比的数值为 100。

4.19.2 方法

4.19.2.1 PWM.init(channel, freq, duty)

根据输入的参数初始化 PWM 对象，参数说明同上。

4.19.2.2 PWM.deinit()

用于关闭 PWM 对象，对象 deinit 之后需要重新 init 才能使用。

4.19.2.3 PWM.freq(freq)

用于获取或者设置 PWM 对象的频率，频率的范围为 [1, 156250]。如果参数为空，返回当前 PWM 对象的频率；如果参数非空，则使用该参数设置当前 PWM 对象的频率。

4.19.2.4 PWM.duty(duty)

用于获取或者设置 PWM 对象的占空比数值，占空比数值的范围为 [0, 255]，例如 `duty = 100`，表示当前设备占空比为 $100/255 = 39.22\%$ 。如果参数为空，返回当前 PWM 对象的占空比数值；如果参数非空，则使用该参数设置当前 PWM 对象的占空比数值。

4.19.3 示例

```
>>> from machine import PWM      # 从 machine 导入 PWM 类
>>> pwm = PWM(3, 3, 1000, 100)  # 创建 PWM 对象，当前使用编号为 1 的 PWM 设备的 4 通
                                # 道，初始化的频率为 1000Hz，占空比数值为 100（占空比为 100/255 = 39.22%）
>>> pwm.freq(2000)              # 设置 PWM 对象频率
>>> pwm.freq()                  # 获取 PWM 对象频率
2000
>>> pwm.duty(200)                # 设置 PWM 对象占空比数值
>>> pwm.duty()                  # 获取 PWM 对象占空比数值
200
>>> pwm.deinit()                # 关闭 PWM 对象
>>> pwm.init(3, 1000, 100)      # 开启并重新配置 PWM 对象
```

4.20 machine.ADC

`machine.ADC` 类是 `machine` 模块下的一个硬件类，用于指定 ADC 设备的配置和控制，提供对 ADC 设备的操作方法。

- ADC（Analog-to-Digital Converter，模数转换器），用于将连续变化的模拟信号转化为离散的数字信号。
- ADC 设备两个重要参数：采样值、分辨率；
 - 采样值：当前时间由模拟信号转化的数值信号的数值；

- 分辨率：以二进制（或十进制）数的位数来表示，一般有 8 位、10 位、12 位、16 位等，它说明模数转换器对输入信号的分辨能力，位数越多，表示分辨率越高，采样值会更精确。

4.20.1 构造函数

在 RT-Thread MicroPython 中 ADC 对象的构造函数如下：

4.20.1.1 class machine.ADC(id, channel)

- **id**: 使用的 ADC 设备编号，**id = 1** 表示编号为 1 的 ADC 设备；
- **channel**: 使用的 ADC 设备通道号，每个 ADC 设备对应多个通道；

例如：**ADC(1,4)** 表示当前使用编号为 1 的 ADC 设备的 4 通道。

4.20.2 方法

4.20.2.1 ADC.init(channel)

根据输入的层参数初始化 ADC 对象，入参为使用的 ADC 对象通道号；

4.20.2.2 ADC.deinit()

用于关闭 ADC 对象，ADC 对象 deinit 之后需要重新 init 才能使用。

4.20.2.3 ADC.read()

用于获取并返回当前 ADC 对象的采样值。例如当前采样值为 2048，对应设备的分辨率为 12 位，当前设备参考电压为 3.3V，则该 ADC 对象通道上实际电压值的计算公式为：**采样值 * 参考电压 / (1 << 分辨率位数)**，即 **vol = 2048 / 4096 * 3.3 V = 1.15V**。

4.20.3 示例

```
>>> from machine import ADC      # 从 machine 导入 ADC 类
>>> adc = ADC(1, 13)           # 创建 ADC 对象，当前使用编号为 1 的 ADC 设备的 13
    通道
>>> adc.read()                 # 获取 ADC 对象采样值
4095
>>> adc.deinit()               # 关闭 ADC 对象
>>> adc.init(13)               # 开启并重新配置 ADC 对象
```


4.21 machine.WDT

machine.WDT 类是 **machine** 模块下的一个硬件类，用于 WDT 设备的配置和控制，提供对 WDT 设备的操作方法。

如下为 WDT 设备基本介绍：

- WDT (WatchDog Timer, 硬件看门狗)，是一个定时器设备，用于系统程序结束或出错导致系统进入不可恢复状态时重启系统。
- WDT 启动之后，计数器开始计数，在计数器溢出前没有被复位，会对 CPU 产生一个复位信号使设备重启（简称“被狗咬”）；
- 系统正常运行时，需要在 WDT 设备允许的时间间隔内对看门狗计数清零（简称“喂狗”），WDT 设备一旦启动，需要定时“喂狗”以确保设备正常运行。

4.21.1 构造函数

在 RT-Thread MicroPython 中 **WDT** 对象的构造函数如下：

4.21.1.1 class machine.WDT(timeout=5)

- **timeout**：设置看门狗超时时间，单位：秒 (s)；

用于创建一个 WDT 对象并且启动 WDT 功能。一旦启动，设置的超时时间无法改动，WDT 功能无法停止。

如果该函数入参为空，则设置超时时间为 5 秒；如果入参非空，则使用该入参设置 WDT 超时时间，超时时间最小设置为 1 秒。

4.21.2 方法

4.21.2.1 WDT.feed()

用于执行“喂狗”操作，清空看门狗设备计数。应用程序应该合理的周期性调用该函数，以防止系统重启。

4.21.3 示例

```
>>> from machine import WDT      # 从 machine 导入 WDT 类
>>> wdt = WDT()                 # 创建 WDT 对象，默认超时时间为 5 秒
>>> wdt = WDT(10)               # 创建 WDT 对象，设置超时时间为 10 秒
>>> wdt.feed()                  # 在 10 秒超时时间内需要执行“喂狗”操作，清空看门狗
                                设备计数，否则系统将重启
```

更多内容可参考 [machine.WDT](#)。

4.22 machine.Timer

machine.Timer 类是 **machine** 模块下的一个硬件类，用于 Timer 设备的配置和控制，提供对 Timer 设备的操作方法。

- Timer（硬件定时器），是一种用于处理周期性和定时性事件的设备。
- Timer 硬件定时器主要通过内部计数器模块对脉冲信号进行计数，实现周期性设备控制的功能。
- Timer 硬件定时器可以自定义**超时时间**和**超时回调函数**，并且提供两种**定时器模式**：
 - **ONE_SHOT**：定时器只执行一次设置的回调函数；
 - **PERIOD**：定时器会周期性执行设置的回调函数；
- 打印 Timer 对象会打印出配置的信息。

4.22.1 构造函数

在 RT-Thread MicroPython 中 **Timer** 对象的构造函数如下：

4.22.1.1 class machine.Timer(id)

- **id**：使用的 Timer 设备编号，**id = 1** 表示编号为 1 的 Timer 设备；

该函数主要用于通过设备编号创建 Timer 设备对象。

4.22.2 方法

4.22.2.1 Timer.init(mode = Timer.PERIODIC, period = 0, callback = None)

- **mode**：设置 Timer 定时器模式，可以设置两种模式：**ONE_SHOT**（执行一次）、**PERIOD**（周期性执行），默认设置的模式为 **PERIOD** 模式；
- **period**：设置 Timer 定时器定时周期，单位：毫秒（ms）
- **callback**：设置 Timer 定义器超时回调函数，默认设置的函数为 None 空函数，设置的函数格式如下所示：

```
def callback_test(device):          # 回调函数有且只有一个入参，为创建的 Timer 对象
    print("Timer callback test")
    print(device)                  # 打印 Timer 对象配置信息
```

该函数使用方式如下示例所示：

```
timer.init(wdt.PERIOD, 5000, callback_test)  # 设置定时器模式为周期性执行，超时时间
                                             为 5 秒，超时函数为 callback_test
```

4.22.2.2 Timer.deinit()

该函数用于停止并关闭 Timer 设备。

4.22.3 常量

下面的常量用来配置 `Timer` 对象。

4.22.3.1 选择定时器模式：

4.22.3.1.1 `Timer.PERIODIC`

4.22.3.1.2 `Timer.ONE_SHOT`

4.22.4 示例

```
>>> from machine import Timer                # 从 machine 导入 Timer 类
>>> timer = Timer(15)                        # 创建 Timer 对象，当前设备编号
      为 11
>>>                                         # 进入粘贴模式
paste mode; Ctrl-C to cancel, Ctrl-D to finish
=== def callback_test(device):              # 定义超时回调函数
===     print("Timer callback test")
>>> timer.init(timer.PERIODIC, 5000, callback_test) # 初始化 Timer 对象，设置定时器
      模式为循环执行，超时时间为 5 秒，超时回调函数 callback_test
>>> Timer callback test                      # 5 秒超时循环执行回调函数，打印
      日志
>>> Timer callback test
>>> Timer callback test
>>> timer.init(timer.ONE_SHOT, 5000, callback_test) # 设置定时器模式为只执行一次，超
      时时间为 5 秒，超时回调函数为 callback_test
>>> Timer callback test                      # 5 秒超时后执行一次回调函数，打
      印日志
>>> timer.deinit()                          # 停止并关闭 Timer 定时器
```

更多内容可参考 [machine.Timer](#)。

4.23 uos – 基本的操作系统服务

`uos` 模块包含了对文件系统的访问操作，是对应 CPython 模块的一个子集。

4.23.1 函数

4.23.1.1 `uos.chdir(path)`

更改当前目录。

4.23.1.2 `uos.getcwd()`

获取当前目录。

4.23.1.3 uos.listdir([dir])

没有参数就列出当前目录，否则列出给定目录。

4.23.1.4 uos.mkdir(path)

创建一个目录。

4.23.1.5 uos.remove(path)

删除文件。

4.23.1.6 uos.rmdir(path)

删除目录。

4.23.1.7 uos.rename(old_path, new_path)

重命名文件或者文件夹。

4.23.1.8 uos.stat(path)

获取文件或目录的状态。

4.23.1.9 uos.sync()

同步所有的文件系统。

4.23.2 示例

```
>>> import uos
>>> uos.
__name__      uname      chdir      getcwd
listdir       mkdir      remove     rmdir
stat          unlink     mount      umount
>>> uos.mkdir("rtthread")
>>> uos.getcwd()
 '/'
>>> uos.chdir("rtthread")
>>> uos.getcwd()
 '/rtthread'
>>> uos.listdir()
 ['web_root', 'rtthread', '11']
>>> uos.rmdir("11")
>>> uos.listdir()
 ['web_root', 'rtthread']
```

```
>>>
```

更多内容可参考 [uos](#) 。

4.24 uselect – 等待流事件

`uselect` 模块提供了等待数据流的事件功能。

4.24.1 常数

4.24.1.1 `select.POLLIN` - 读取可用数据

4.24.1.2 `select.POLLOUT` - 写入更多数据

4.24.1.3 `select.POLLERR` - 发生错误

4.24.1.4 `select.POLLHUP` - 流结束/连接终止检测

4.24.2 函数

4.24.2.1 `select.select(rlist, wlist, xlist[, timeout])`

监控对象何时可读或可写，一旦监控的对象状态改变，返回结果（阻塞线程）。这个函数是为了兼容，效率不高，推荐用 `poll` 函数。

`rlist`: 等待读就绪的文件描述符数组
`wlist`: 等待写就绪的文件描述符数组
`xlist`: 等待异常的数组
`timeout`: 等待时间（单位：秒）

示例：

```
def selectTest():
    global s
    rs, ws, es = select.select([s,], [], [])
    #程序会在此等待直到对象s可读
    print(rs)
    for i in rs:
        if i == s:
            print("s can read now")
            data,addr=s.recvfrom(1024)
            print('received:',data,'from',addr)
```

4.24.3 Poll 类

4.24.3.1 select.poll()

创建 poll 实例。

示例：

```
>>>poller = select.poll()
>>>print(poller)
<poll>
```

4.24.3.2 poll.register(obj[, eventmask])

注册一个用以监控的对象，并设置被监控对象的监控标志位 flag。

obj: 被监控的对象
flag: 被监控的标志
select.POLLIN — 可读
select.POLLHUP — 已挂断
select.POLLERR — 出错
select.POLLOUT — 可写

4.24.3.3 poll.unregister(obj)

解除监控的对象的注册。

obj: 注册过的对象

示例：

```
>>>READ_ONLY = select.POLLIN | select.POLLHUP | select.POLLERR
>>>READ_WRITE = select.POLLOUT | READ_ONLY
>>>poller.register(s, READ_WRITE)
>>>poller.unregister(s)
```

4.24.3.4 poll.modify(obj, eventmask)

修改已注册的对象监控标志。

obj: 已注册的被监控对象
flag: 修改为的监控标志

示例：

```
>>>READ_ONLY = select.POLLIN | select.POLLHUP | select.POLLERR
>>>READ_WRITE = select.POLLOUT | READ_ONLY
>>>poller.register(s, READ_WRITE)
>>>poller.modify(s, READ_ONLY)
```

4.24.3.5 poll.poll([timeout])

等待至少一个已注册的对象准备就绪。返回 (obj, event, ...) 元组, event 元素指定了一个流发生的事件, 是上面所描述的 `select.POLL*` 常量组合。根据平台和版本的不同, 在元组中可能有其他元素, 所以不要假定元组的大小是 2。如果超时, 则返回空列表。

更多内容可参考 [uselect](#)。

4.25 ctypes – 以结构化的方式访问二进制数据

ctypes 模块用来访问二进制数据结构, 它提供 C 兼容的数据类型。

4.25.1 常量

- `ctypes.LITTLE_ENDIAN` — 小端压缩结构。
- `ctypes.BIG_ENDIAN` — 大端压缩结构类型。
- `NATIVE` — mricopython 本地的存储类型

4.25.2 构造函数

4.25.2.1 class ctypes.struct(addr, descriptor, type)

将内存中以 c 形式打包的结构体或联合体转换为字典, 并返回该字典。

addr: 开始转换的地址
descriptor: 转换描述符
 格式: `"field_name": offset | ctypes.UINT32`
offset: 偏移量,
 单位: 字节、VOID、UINT8、INT8、UINT16、INT16、UINT32、INT32、UINT64、INT64、BFUINT8、BFINT8、BFUINT16、BFINT16、BFUINT32、BFINT32、BF_POS、BF_LEN、FLOAT32、FLOAT64、PTR、ARRAY
type: c 结构体或联合体存储类型, 默认为本地存储类型

示例:

```
>>> a = b"0123"
>>> s = ctypes.struct(ctypes.addressof(a), {"a": ctypes.UINT8 | 0, "b": ctypes.UINT16 | 1}, ctypes.LITTLE_ENDIAN)
>>> print(s)
<struct STRUCT 3ffc7360>
>>> print(s.a)
48
>>> s.a = 49
>>> print(a)
b'1123'
```

4.25.3 方法

4.25.3.1 ctypes.sizeof(struct)

按字节返回数据的大小。参数可以是类或者数据对象 (或集合)。示例：

```
>>> a = b"0123"
>>> b = ctypes.struct(ctypes.addressof(a), {"a": ctypes.UINT8 | 0, "b": ctypes.
    UINT16 | 1}, ctypes.LITTLE_ENDIAN)
>>> b.a
48
>>> print(ctypes.sizeof(b))
3
```

4.25.3.2 ctypes.addressof(obj)

返回对象地址。参数需要是 bytes, bytearray 。示例：

```
>>> a = b"0123"
>>> print(ctypes.addressof(a))
1073504048
```

4.25.3.3 ctypes.bytes_at(addr, size)

捕捉从 addr 开始到 size 个地址偏移量结束的内存数据为 bytearray 对象并返回。示例：

```
>>> a = b"0123"
>>> print(ctypes.bytes_at(ctypes.addressof(a), 4))
b'0123'
```

4.25.3.4 ctypes bytearray_at(addr, size)

捕捉给定大小和地址内存为 bytearray 对象。与 bytes_at() 函数不同的是，它可以被再次写入，可以访问给定地址的参数。示例：

```
>>> a = b"0123"
>>> print(ctypes.bytearray_at(ctypes.addressof(a), 2))
bytearray(b'01')
```

更多内容可参考 [ctypes](#) 。

4.26 uerrno – 系统错误码模块

uerrno 模块提供了标准的 errno 系统符号，每个符号都有对应的整数值。

4.26.1 示例

```
try:
    uos.mkdir("my_dir")
except OSError as exc:
    if exc.args[0] == uerrno.EEXIST:
        print("Directory already exists")
    uerrno.errorcode
Dictionary mapping numeric error codes to strings with symbolic error code (see
    above):

>>> print(uerrno.errorcode[uerrno.EEXIST])
EEXIST
```

更多内容可参考 [uerrno](#)。

4.27 `_thread` – 多线程支持

`_thread` 模块提供了用于处理多线程的基本方法——多个控制线程共享它们的全局数据空间。为了实现同步，提供了简单的锁（也称为互斥锁或二进制信号量）。

4.27.1 示例

```
import _thread
import time
def testThread():
    while True:
        print("Hello from thread")
        time.sleep(2)

_thread.start_new_thread(testThread, ())
while True:
    pass
```

输出结果（启动新的线程，每隔两秒打印字符）：

Hello from thread Hello from thread Hello from thread Hello from thread Hello from thread

更多内容可参考 [_thread](#)。

4.28 `cmath` – 复数的数学函数

`cmath` 模块提供了对复数的数学函数的访问。这个模块中的函数接受整数、浮点数或复数作为参数。他们还将接受任何有复数或浮点方法的 Python 对象：这些方法分别用于将对象转换成复数或浮点数，然后将该函数应用到转换的结果中。

4.28.1 函数

4.28.1.1 cmath.cos(z)

返回 z 的余弦。

4.28.1.2 cmath.exp(z)

返回 z 的指数。

4.28.1.3 cmath.log(z)

返回 z 的对数。

4.28.1.4 cmath.log10(z)

返回 z 的常用对数。

4.28.1.5 cmath.phase(z)

返回 z 的相位, 范围是 $(-\pi, +\pi]$, 以弧度表示。

4.28.1.6 cmath.polar(z)

返回 z 的极坐标。

4.28.1.7 cmath.rect(r, phi)

返回模量 r 和相位 ϕ 的复数。

4.28.1.8 cmath.sin(z)

返回 z 的正弦。

4.28.1.9 cmath.sqrt(z)

返回 z 的平方根。

4.28.2 常数

4.28.2.1 cmath.e

自然对数的指数。

4.28.2.2 cmath.pi

圆周率。

更多内容可参考 [cmath](#)。

4.29 ubinascii – 二进制/ ASCII 转换

`ubinascii` 模块包含许多在二进制和各种 `ascii` 编码的二进制表示之间转换的方法。

4.29.1 函数

4.29.1.1 ubinascii.hexlify(data[, sep])

将字符串转换为十六进制表示的字符串。

示例：

```
>>> ubinascii.hexlify('hello RT-Thread')
b'68656c6c6f2052542d546872656164'
>>> ubinascii.hexlify('summer')
b'73756d6d6572'
```

如果指定了第二个参数 `sep`，它将用于分隔两个十六进制数。

示例：

```
如果指定了第二个参数sep，它将用于分隔两个十六进制数。
示例：
>>> ubinascii.hexlify('hello RT-Thread', " ")
b'68 65 6c 6c 6f 20 52 54 2d 54 68 72 65 61 64'
>>> ubinascii.hexlify('hello RT-Thread', ",")
b'68,65,6c,6c,6f,20,52,54,2d,54,68,72,65,61,64'
```

4.29.1.2 ubinascii.unhexlify(data)

转换十六进制字符串为二进制字符串，功能和 `hexlify` 相反。

示例：

```
>>> ubinascii.unhexlify('73756d6d6572')
b'summer'
```

4.29.1.3 ubinascii.a2b_base64(data)

Base64 编码的数据转换为二进制表示。返回字节串。

4.29.1.4 ubinascii.b2a_base64(data)

编码 base64 格式的二进制数据。返回的字符串。

更多内容可参考 [ubinascii](#)。

4.30 uhashlib – 哈希算法

`uhashlib` 模块实现了二进制数据哈希算法。

4.30.1 算法功能

4.30.1.1 SHA256

当代的散列算法（SHA2 系列），它适用于密码安全的目的。被包含在 MicroPython 内核中，除非有特定的代码大小限制，否则推荐任何开发板都支持这个功能。

4.30.1.2 SHA1

上一代的算法，不推荐新的应用使用这种算法，但是 SHA1 算法是互联网标准和现有应用程序的一部分，所以针对网络连接便利的开发板会提供这种功能。

4.30.1.3 MD5

一种遗留下来的算法，作为密码使用被认为是不安全的。只有特定的开发板，为了兼容老的应用才会提供这种算法。

4.30.2 函数

4.30.2.1 class uhashlib.sha256([data])

创建一个 SHA256 哈希对象并提供 data 赋值。

4.30.2.2 class uhashlib.sha1([data])

创建一个 SHA1 哈希对象并提供 data 赋值。

4.30.2.3 class uhashlib.md5([data])

创建一个 MD5 哈希对象并提供 data 赋值。

4.30.2.4 hash.update(data)

将更多二进制数据放入哈希表中。

4.30.2.5 hash.digest()

返回字节对象哈希的所有数据。调用此方法后，将无法将更多数据送入哈希。

4.30.2.6 hash.hexdigest()

此方法没有实现，使用 `ubinascii.hexlify(hash.digest())` 达到类似效果。

更多内容可参考 [uhashlib](#)。

4.31 uheapq – 堆排序算法

`uheapq` 模块提供了堆排序相关算法，堆队列是一个列表，它的元素以特定的方式存储。

4.31.1 函数

4.31.1.1 uheapq.heappush(heap, item)

将对象压入堆中。

4.31.1.2 uheapq.heappop(heap)

从 `heap` 弹出第一个元素并返回。如果是堆时空的会抛出 `IndexError`。

4.31.1.3 uheapq.heapify(x)

将列表 `x` 转换成堆。

更多内容可参考 [uheapq](#)。

4.32 ujson – JSON 编码与解码

`ujson` 模块提供 Python 对象到 JSON（JavaScript Object Notation）数据格式的转换。

4.32.1 函数

4.32.1.1 ujson.dumps(obj)

将 `dict` 类型转换成 `str`。

`obj`: 要转换的对象

示例：

```
>>> obj = {1:2, 3:4, "a":6}
>>> print(type(obj), obj) #原来为dict类型
<class 'dict'> {3: 4, 1: 2, 'a': 6}
>>> jsonObj = json.dumps(obj) #将dict类型转换成str
>>> print(type(jsonObj), jsonObj)
<class 'str'> {3: 4, 1: 2, "a": 6}
```

4.32.1.2 ujson.loads(str)

解析 JSON 字符串并返回对象。如果字符串格式错误将引发 ValueError 异常。示例：

```
>>> obj = {1:2, 3:4, "a":6}
>>> jsDumps = json.dumps(obj)
>>> jsLoads = json.loads(jsDumps)
>>> print(type(obj), obj)
<class 'dict'> {3: 4, 1: 2, 'a': 6}
>>> print(type(jsDumps), jsDumps)
<class 'str'> {3: 4, 1: 2, "a": 6}
>>> print(type(jsLoads), jsLoads)
<class 'dict'> {'a': 6, 1: 2, 3: 4}
```

更多内容可参考 [ujson](#) 。

4.33 ure – 正则表达式

ure 模块用于测试字符串的某个模式，执行正则表达式操作。

4.33.1 匹配字符集

4.33.1.1 匹配任意字符

```
'.'
```

4.33.1.2 匹配字符集合，支持单个字符和一个范围

```
'[]'
```

4.33.1.3 支持多种匹配元字符

```
'^' '$' '?' '*' '+' '??' '*?' '+?' '{m,n}'
```

4.33.2 函数

4.33.2.1 ure.compile(regex)

编译正则表达式，返回 regex 对象。

4.33.2.2 ure.match(regex, string)

用 string 匹配 regex，匹配总是从字符串的开始匹配。

4.33.2.3 ure.search(regex, string)

在 string 中搜索 regex。不同于匹配，它搜索第一个匹配位置的正则表达式字符串 (结果可能会是 0)。

4.33.2.4 ure.DEBUG

标志值，显示表达式的调试信息。

4.33.3 正则表达式对象:

编译正则表达式，使用 `ure.compile()` 创建实例。

4.33.3.1 regex.match(string)

4.33.3.2 regex.search(string)

4.33.3.3 regex.split(string, max_split=-1)

4.33.4 匹配对象:

匹配对象是 `match()` 和 `search()` 方法的返回值。

4.33.4.1 match.group([index])

只支持数字组。

更多内容可参考 [ure](#)。

4.34 uzlib – zlib 解压缩

`uzlib` 模块实现了使用 DEFLATE 算法解压缩二进制数据 (常用的 `zlib` 库和 `gzip` 文档)。目前不支持压缩。

4.34.1 函数

4.34.1.1 uzlib.decompress(data)

返回解压后的 bytes 数据。

更多内容可参考 [uzlib](#)。

4.35 urandom - 随机数生成模块

`urandom` 模块实现了伪随机数生成器。

4.35.1 函数

4.35.1.1 urandom.choice(obj)

随机生成对象 `obj` 中的元数。

`obj`: 元数列表

示例:

```
>>> print(random.choice("DFRobot"))
R
>>> print(random.choice("DFRobot"))
D
>>> print(random.choice([0, 2, 4, 3]))
3
>>> print(random.choice([0, 2, 4, 3]))
3
>>> print(random.choice([0, 2, 4, 3]))
2
```

4.35.1.2 urandom.getrandbits(size)

随机生成 0 到 `size` 个位二进制数范围内的正整数。比如:

- `size = 4`, 那么便是从 0 到 0b1111 中随机一个正整数。
- `size = 8`, 那么便是从 0 到 0b11111111 中随机一个正整数。

`size`: 位大小

示例:

```
>>> print( random.getrandbits(1))  #1位二进制位, 范围为0~1 (十进制: 0~1)
1
```



```
>>> print(random.getrandbits(1))
0
>>> print(random.getrandbits(8)) #8位二进制位，范围为0000 0000~1111 1111（十进制：
    0~255）
224
>>> print(random.getrandbits(8))
155
```

4.35.1.3 urandom.randint(start, end)

随机生成一个 start 到 end 之间的整数。

start: 指定范围内的开始值，包含在范围内
end: 指定范围内的结束值，包含在范围内

示例：

```
>>> import random
>>> print(random.randint(1, 4))
4
>>> print(random.randint(1, 4))
2
```

4.35.1.4 urandom.random()

随机生成一个 0 到 1 之间的浮点数。示例：

```
>>> print(random.random())
0.7111824
>>> print(random.random())
0.3168149
```

4.35.1.5 urandom.randrange(start, end, step)

随机生成 start 到 end 并且递增为 step 的范围内的正整数。例如，randrange(0, 8, 2) 中，随机生成的数为 0、2、4、6 中任一个。

start: 指定范围内的开始值，包含在范围内
end: 指定范围内的结束值，包含在范围内
step: 递增基数

示例：

```
>>> print(random.randrange(2, 8, 2))
4
>>> print(random.randrange(2, 8, 2))
6
```

```
>>> print(random.randrange(2, 8, 2))
2
```

4.35.1.6 urandom.seed(sed)

指定随机数种子，通常和其他随机数生成函数搭配使用。**注意：**MicroPython 中的随机数其实是一个稳定算法得出的稳定结果序列，而不是一个随机序列。sed 就是这个算法开始计算的第一个值。所以就会出现只要 sed 是一样的，那么后续所有“随机”结果和顺序也都完全一致。

sed: 随机数种子

示例：

```
import random

for j in range(0, 2):
    random.seed(13) #指定随机数种子
    for i in range(0, 10): #生成0到10范围内的随机序列
        print(random.randint(1, 10))
    print("end")
```

运行结果：

```
5
2
3
2
3
4
2
5
8
2
end
5
2
3
2
3
4
2
5
8
2
end
```

从上面可以看到生成两个随机数列表是一样的，你也可以多生成几个随机数列表查看结果。

4.35.1.7 urandom.uniform(start, end)

随机生成 start 到 end 之间的浮点数。

start: 指定范围内的开始值，包含在范围内
stop: 指定范围内的结束值，包含在范围内

示例：

```
>>> print(random.uniform(2, 4))
2.021441
>>> print(random.uniform(2, 4))
3.998012
```

更多内容可参考 [urandom](#)。

4.36 usocket – 套接字模块

`usocket` 模块提供对 BSD 套接字接口的访问。

4.36.1 常数

4.36.1.1 地址簇

- `socket.AF_INET = 2` — TCP/IP – IPv4
- `socket.AF_INET6 = 10` — TCP/IP – IPv6

4.36.1.2 套接字类型

- `socket.SOCK_STREAM = 1` — TCP 流
- `socket.SOCK_DGRAM = 2` — UDP 数据报
- `socket.SOCK_RAW = 3` — 原始套接字
- `socket.SO_REUSEADDR = 4` — socket 可重用

4.36.1.3 IP 协议号

- `socket.IPPROTO_TCP = 16`
- `socket.IPPROTO_UDP = 17`

4.36.1.4 套接字选项级别

- `socket.SOL_SOCKET = 4095`

4.36.2 函数

4.36.2.1 socket.socket

```
socket.socket(socket.AF_INET, socket.SOCK_STREAM, socket.IPPROTO_TCP)
```

创建新的套接字，使用指定的地址、类型和协议号。

4.36.2.2 socket.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。元组列表的结构如下：

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

示例：

```
>>> info = socket.getaddrinfo("rt-thread.org", 10000)
>>> print(info)
[(2, 1, 0, '', ('118.31.15.152', 10000))]
```

4.36.2.3 socket.close()

关闭套接字。一旦关闭后，套接字所有的功能都将失效。远端将接收不到任何数据（清理队列数据后）。虽然在垃圾回收时套接字会自动关闭，但还是推荐在必要时用 close() 去关闭。

4.36.2.4 socket.bind(address)

将套接字绑定到地址，套接字不能是已经绑定的。

4.36.2.5 socket.listen([backlog])

监听套接字，使服务器能够接收连接。

backlog：接受套接字的最大个数，至少为0，如果没有指定，则默认一个合理值。

4.36.2.6 socket.accept()

接收连接请求。**注意**：只能在绑定地址端口号和监听后调用，返回 conn 和 address。

conn：新的套接字对象，可以用来收发消息
address：连接到服务器的客户端地址

4.36.2.7 socket.connect(address)

连接服务器。

address：服务器地址和端口号的元组或列表

4.36.2.8 socket.send(bytes)

发送数据，并返回成功发送的字节数，返回字节数可能比发送的数据长度少。

bytes: bytes 类型数据

4.36.2.9 socket.recv(bufsize)

接收数据，返回接收到的数据对象。

bufsize: 指定一次接收的最大数据量

示例:

```
data = conn.recv(1024)
```

4.36.2.10 socket.sendto(bytes, address)

发送数据，目标由 address 决定，常用于 UDP 通信，返回发送的数据大小。

bytes: bytes 类型数据

address: 目标地址和端口号的元组

示例:

```
data = sendto("hello RT-Thread", ("192.168.10.110", 100))
```

4.36.2.11 socket.recvfrom(bufsize)

接收数据，常用于 UDP 通信，并返回接收到的数据对象和对象的地址。

bufsize: 指定一次接收的最大数据量

示例:

```
data, addr = fd.recvfrom(1024)
```

4.36.2.12 socket.setsockopt(level, optname, value)

根据选项值设置套接字。

level: 套接字选项级别

optname: 套接字的选项

value: 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:

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

4.36.2.13 socket.settimeout(value)

设置超时时间，单位：秒。示例：

```
s.settimeout(2)
```

4.36.2.14 socket.setblocking(flag)

设置阻塞或非阻塞模式：如果 flag 是 false，设置非阻塞模式。

4.36.2.15 socket.read([size])

Read up to size bytes from the socket. Return a bytes object. If size is not given, it reads all data available from the socket until EOF; as such the method will not return until the socket is closed. This function tries to read as much data as requested (no “short reads”). This may be not possible with non-blocking socket though, and then less data will be returned.

4.36.2.16 socket.readinto(buf[, nbytes])

Read bytes into the buf. If nbytes is specified then read at most that many bytes. Otherwise, read at most len(buf) bytes. Just as read(), this method follows “no short reads” policy. Return value: number of bytes read and stored into buf.

4.36.2.17 socket.readline()

接收一行数据，遇换行符结束，并返回接收数据的对象。

4.36.2.18 socket.write(buf)

将字节类型数据写入套接字，并返回写入成功的数据大小。

4.36.3 示例

4.36.3.1 TCP Server example

```
>>> import usocket
>>> s = usocket.socket(usocket.AF_INET, usocket.SOCK_STREAM) # Create STREAM TCP
    socket
>>> s.bind(('192.168.12.32', 6001))
>>> s.listen(5)
>>> s.setblocking(True)
>>> sock, addr = s.accept()
>>> sock.recv(10)
b'rt-thread\r'
>>> s.close()
```

4.36.3.2 TCP Client example

```
>>> import usocket
>>> s = usocket.socket(usocket.AF_INET, usocket.SOCK_STREAM)
>>> s.connect(("192.168.10.110", 6000))
>>> s.send("micropython")
11
>>> s.close()
```

connect to a web site example:

```
s = socket.socket()
s.connect(socket.getaddrinfo('www.micropython.org', 80)[0][-1])
```

更多的内容可参考 [usocket](#) 。

4.37 network – 网络配置

此模块提供网络驱动程序和路由配置。特定硬件的网络驱动程序在此模块中可用，用于配置硬件网络接口。然后，配置接口提供的网络服务可以通过 [usocket](#) 模块使用。

4.37.1 专用的网络类配置

下面具体的类实现了抽象网卡的接口，并提供了一种控制各种网络接口的方法。

- class WLAN – control built-in WiFi interfaces

4.38 class WLAN – 控制内置的 WiFi 网络接口

该类为 WiFi 网络处理器提供一个驱动程序。使用示例:

```
import network
# enable station interface and connect to WiFi access point
nic = network.WLAN(network.STA_IF)
nic.active(True)
nic.connect('your-ssid', 'your-password')
# now use sockets as usual
```

4.38.1 构造函数

在 RT-Thread MicroPython 中 [WLAN](#) 对象的构造函数如下:

4.38.1.1 class network.WLAN(interface_id)

创建一个 WLAN 网络接口对象。支持的接口是 `network.STA_IF` (STA 模式, 可以连接到上游的 WiFi 热点上) 和 `network.AP_IF` (AP 模式, 允许其他 WiFi 客户端连接到自身的热点)。下面方法的可用性取决于接口的类型。例如, 只有 STA 接口可以使用 `WLAN.connect()` 方法连接到 AP 热点上。

4.38.2 方法

4.38.2.1 WLAN.active([is_active])

如果向该方法传入布尔数值, 传入 `True` 则使能卡, 传入 `False` 则禁止网卡。否则, 如果不传入参数, 则查询当前网卡的状态。

4.38.2.2 WLAN.connect(ssid, password)

使用指定的账号和密码链接指定的无线热点。

4.38.2.3 WLAN.disconnect()

从当前链接的无线网络中断开。

4.38.2.4 WLAN.scan()

扫描当前可以连接的无线网络。

只能在 STA 模式下进行扫描, 使用元组列表的形式返回 WiFi 接入点的相关信息。

(ssid, bssid, channel, rssi, authmode, hidden)

4.38.2.5 WLAN.status([param])

返回当前无线连接的状态。

当调用该方法时没有附带参数, 就会返回值描述当前网络连接的状态。如果还没有从热点连接中获得 IP 地址, 此时的状态为 `STATION_IDLE`。如果已经从连接的无线网络中获得 IP 地址, 此时的状态为 `STAT_GOT_IP`。

当调用该函数使用的参数为 `rssi` 时, 则返回 `rssi` 的值, 该函数目前只支持这一个参数。

4.38.2.6 WLAN.isconnected()

在 STA 模式时, 如果已经连接到 WiFi 网络, 并且获得了 IP 地址, 则返回 `True`。如果处在 AP 模式, 此时已经与客户端建立连接, 则返回 `True`。其他情况下都返回 `False`。

4.38.2.7 WLAN.ifconfig([(ip, subnet, gateway, dns)])

获取或者设置网络接口的参数，IP 地址，子网掩码，网关，DNS 服务器。当调用该方法不附带参数时，该方法会返回一个包含四个元素的元组来描述上面的信息。想要设置上面的值，传入一个包含上述四个元素的元组，例如：

```
nic.ifconfig(('192.168.0.4', '255.255.255.0', '192.168.0.1', '8.8.8.8'))
```

4.38.2.8 WLAN.config('param')

4.38.2.9 WLAN.config(param=value, ...)

获取或者设置一般网络接口参数，这些方法允许处理标准的 ip 配置之外的其他参数，如 `WLAN.ifconfig()` 函数处理的参数。这些参数包括特定网络和特定硬件的参数。对于参数的设置，应该使用关键字的语法，可以一次性设置多个参数。

当查询参数时，参数名称的引用应该为字符串，且每次只能查询一个参数。

```
# Set WiFi access point name (formally known as ESSID) and WiFi password
ap.config(essid='My_AP', password="88888888")
# Query params one by one
print(ap.config('essid'))
print(ap.config('channel'))
```

下面是目前支持的参数：

Parameter	Description
mac	MAC address (bytes)
essid	WiFi access point name (string)
channel	WiFi channel (integer)
hidden	Whether ESSID is hidden (boolean)
password	Access password (string)

4.38.3 示例

STA 模式下：

```
import network
wlan = network.WLAN(network.STA_IF)
wlan.scan()
wlan.connect("rtthread", "021888888888")
wlan.isconnected()
```

AP 模式下：

```
import network
ap = network.WLAN(network.AP_IF)
ap.config(essid="hello_rt-thread", password="88888888")
ap.active(True)
ap.config("essid")
```

第 5 章

RT-Thread MicroPython 包管理

MicroPython 自身不像 CPython 那样拥有广泛的标准库。但是 MicroPython 有一个相关且独立的项目 micropython-lib，它提供了来自 CPython 标准库的许多模块的实现。由于 RT-Thread 操作系统提供了很好的 POSIX 标准支持，所以 micropython-lib 中很多模块可以在 RT-Thread MicroPython 上直接运行。这篇文章将介绍如何利用这些扩展模块来增强 MicroPython 的功能。

5.1 1. 使用 micropython-lib 源代码

5.1.1 1.1 从 GitHub 上克隆/下载 micropython-lib 的源代码到本地

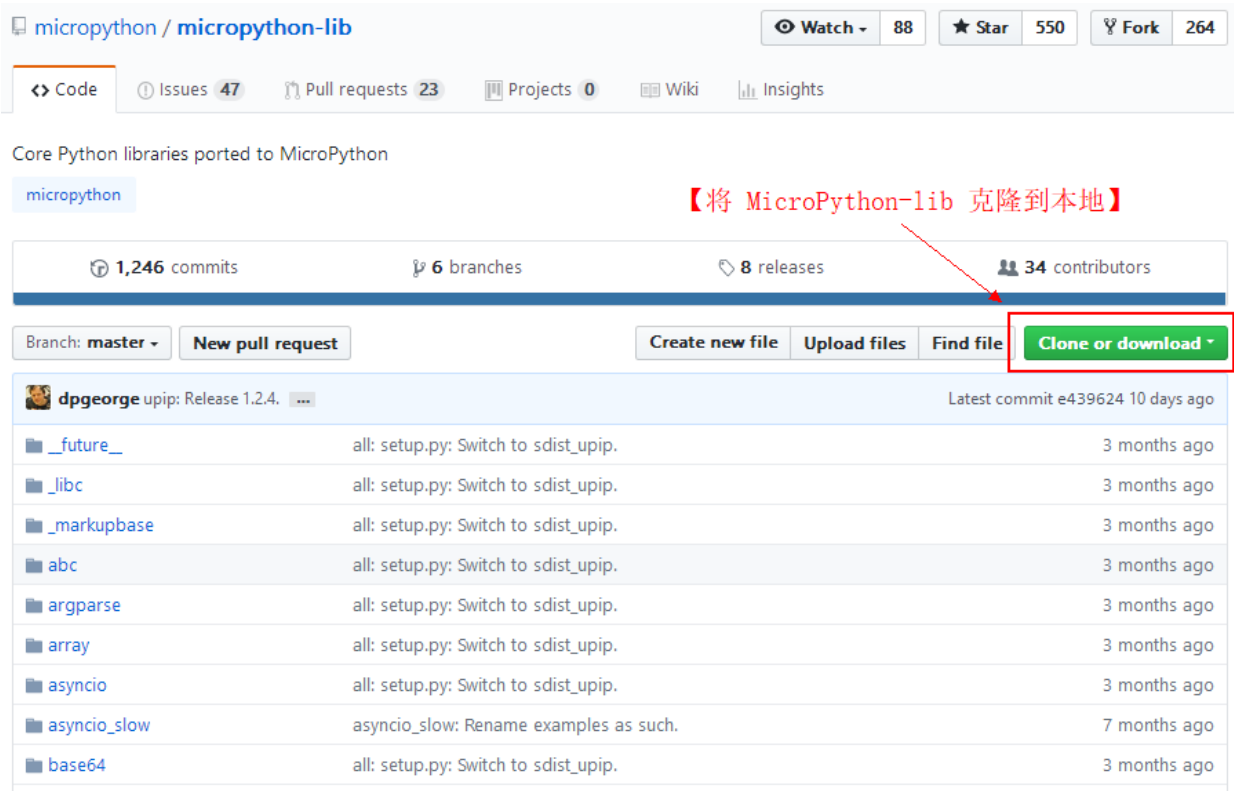


图 5.1: 1525330162579

5.1.2 1.2 使用扩展包

如使用 abc 扩展包，就可以将 abc 文件夹中除了 metadata.txt 和 setup.py 的内容 (不同模块有所不同，如果想避免出错，可以直接将 abc 文件夹中的内容全部复制过去即可) 复制到 SD 卡 (文件系统) 的 /libs/mpy 目录下：

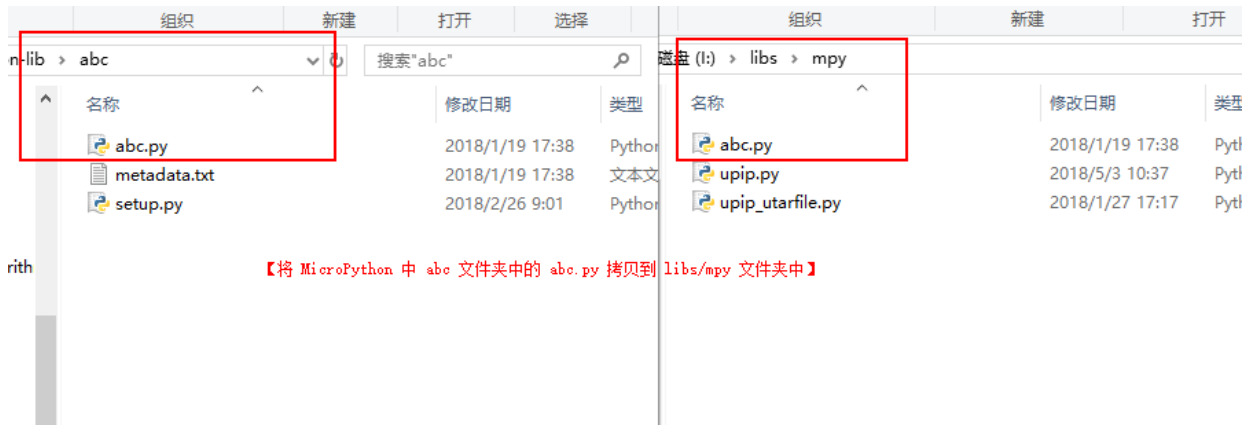


图 5.2: 1525333595133

现在就可以在 MicroPython 交互命令行中导入 abc 模块了。

```
MicroPython v1.9.3-477-g7b0a020-dirty on 2018-03-21; Universal python platform with RT-Thread
Type "help()" for more information.
>>> import abc
>>> abc.
__class__      __file__      __name__      abstractmethod
>>> abc.abstractmethod("Hello RT-Thread!")
'Hello RT-Thread!'
>>>
```

图 5.3: 1525340341541

!!! note “注意” 当安装的模块依赖于其他模块的功能时，需要先安装依赖的模块。这时候就需要搞清楚依赖的模块有哪些，然后依次安装依赖的模块。

5.2 2. upip 包管理器的使用

upip 是 MicroPython 的包管理程序，由于 RT-Thread 操作系统提供了很好的 POSIX 标准支持，所以 upip 可以很好地运行在 RT-Thread MicroPython 上。使用 upip 工具可以在线下载 MicroPython 的扩展模块，并且自动下载其依赖的模块，为用户扩展 MicroPython 功能提供了很大的便利。

upip 包管理器对系统资源的占用较大，推荐在有较多系统资源的开发板上使用 upip。使用 upip 可以按照下面的步骤来进行配置：

- 将 micropython-lib 中 upip 文件夹下的所需内容拷贝到 SD 卡 (文件系统) 的 /libs/mpy 目录下：

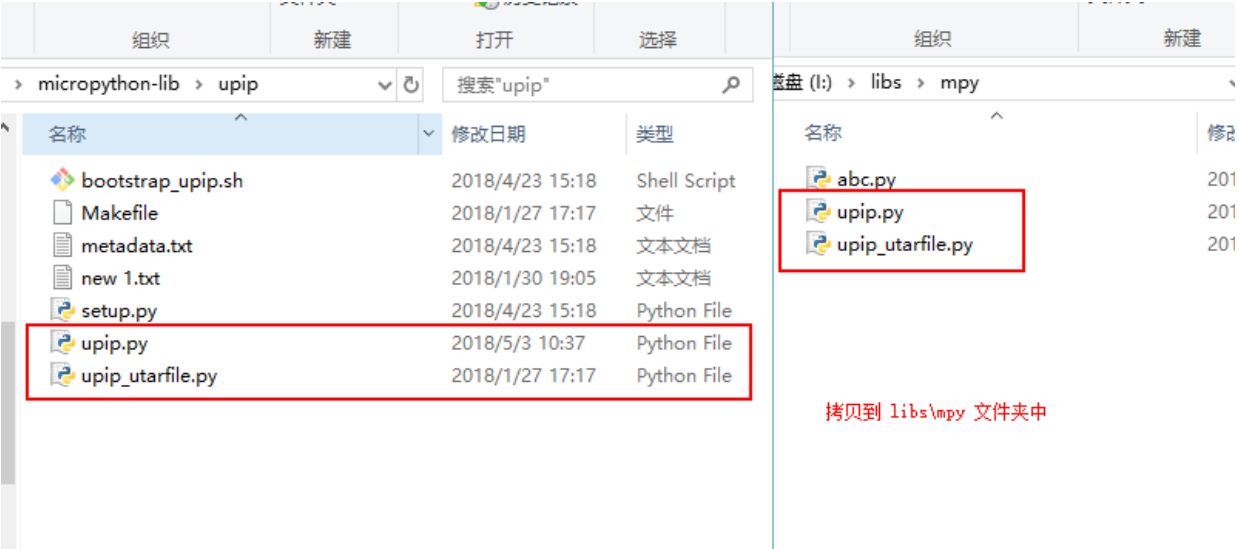


图 5.4: 1525336695038

- 在 env 中开启 MicroPython 网络模块中的 ussl 模块。

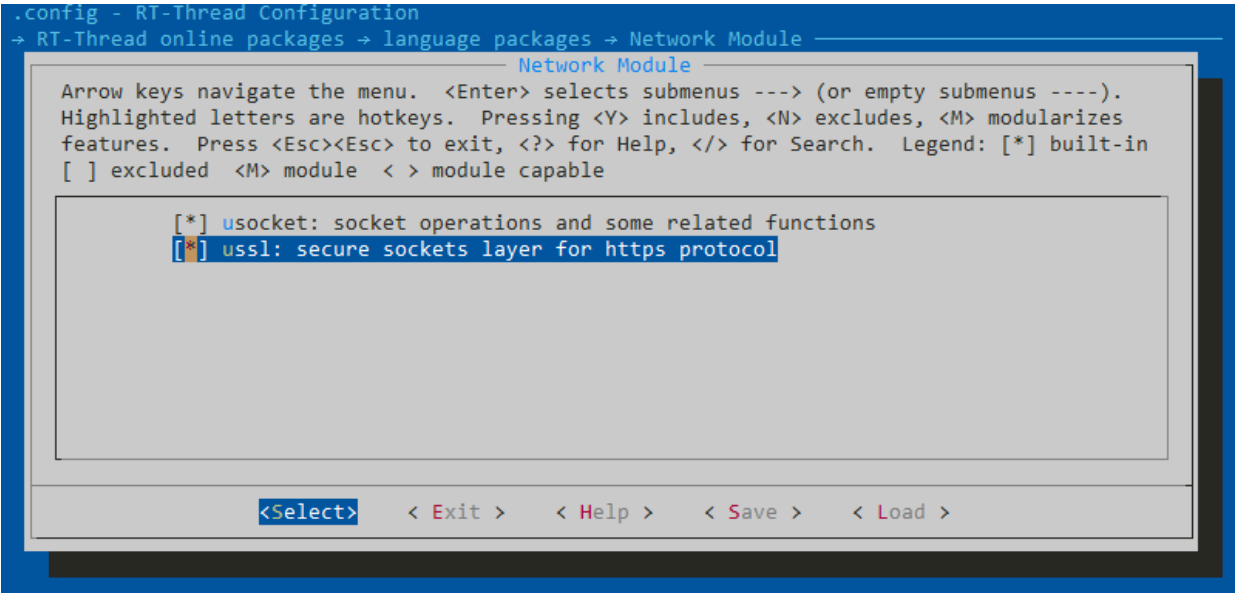


图 5.5: 1525337170291

- 修改软件包中安全类的 mbedtls 软件包 Maxium fragmenty length in bytes 参数为 5120。

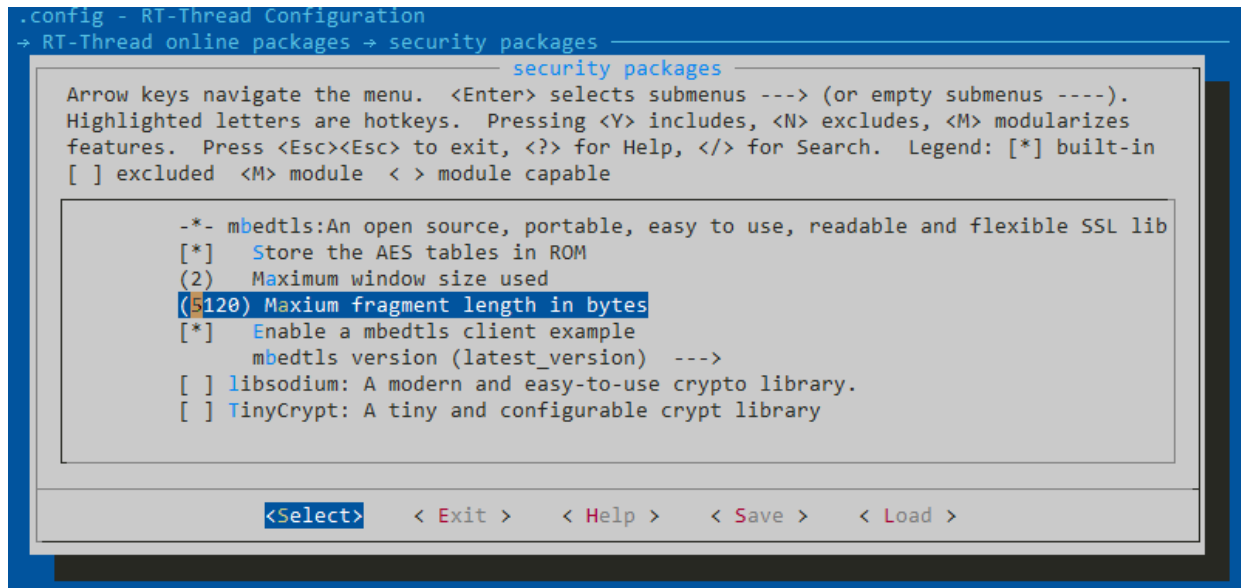


图 5.6: 1525337315573

- 更新软件包并重新生成工程，重新编译并下载固件到开发板然后再运行。至此，模块就被成功下载到 /libs/mpy 文件夹下，可以直接使用了。

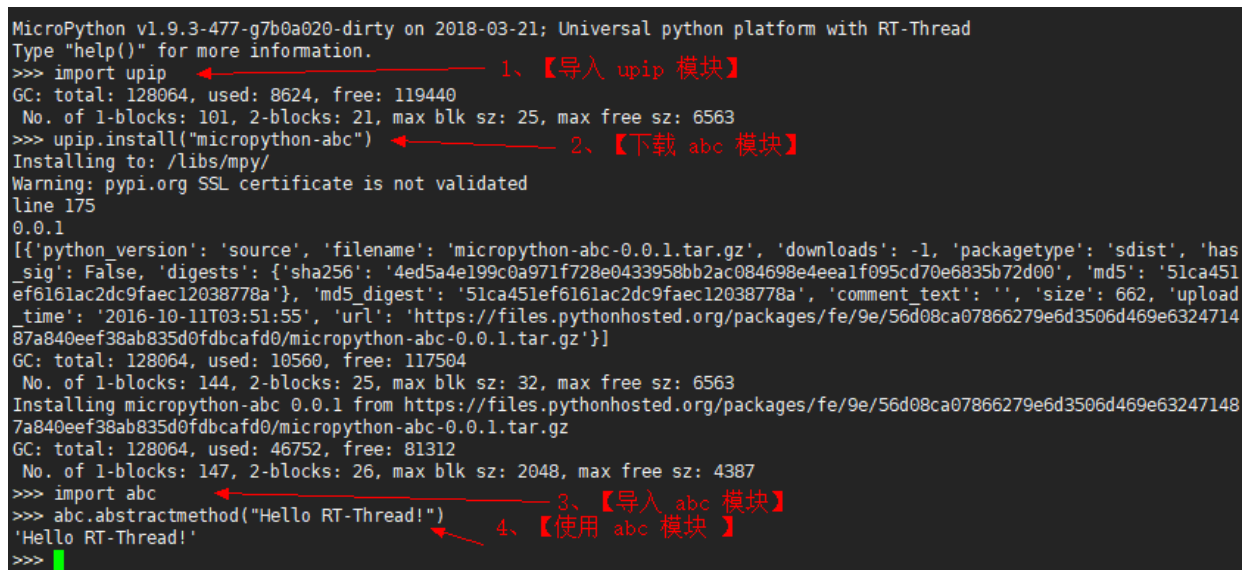


图 5.7: 1525340694594

除了 micropython-lib 中提供的之外，更多 MicroPython 模块可以到 [pypi 搜索](#) 中查找。

!!! note “注意” 下载指定模块时，需要在模块前加上 micropython- 前缀。如想要安装 json 模块，使用 `upip.install("micropython-json")` 命令。

第 6 章

RT-Thread MicroPython 网络编程指南

MicroPython 提供丰富的网络功能，可以加快物联网项目的开发进程，本章介绍常用的网络功能以及相关模块的使用方法。了解网络功能之后，就可以将产品轻松的接入网络，实现更多物联网功能。

6.1 预备知识

- 在阅读本网络编程指南之前，需要先行了解 MicroPython 模块的网络模块章节，了解基本网络连接模块的使用方法。
- 如果想要使用较为复杂的网络功能，需要在 menuconfig 中将工具模块中的模块都打开，并且必须开启网络模块中的 `usocket` 模块。

6.2 HttpClient

本节介绍如何在 RT-Thread MicroPython 上使用 Http Client 功能，本章主要使用的模块为 `urequests`。

6.2.1 获取并安装 `urequests` 模块

获取该模块有两种方式，详细操作可参考包管理章节：

- 方法 1：使用 `upip` 包管理工具下载，这里使用 `upip.install("micropython-urequests")` 命令，`upip` 工具将自动下载并安装 `urequests` 模块，下载过程如图所示：

```
>>> import upip
GC: total: 128064, used: 8480, free: 119584
No. of 1-blocks: 101, 2-blocks: 21, max blk sz: 25, max free sz: 6580
>>> upip.install("micropython-urequests")
Installing to: /libs/mpy/
Warning: pypi.org SSL certificate is not validated
GC: total: 128064, used: 10576, free: 117488
No. of 1-blocks: 143, 2-blocks: 24, max blk sz: 32, max free sz: 5998
Installing micropython-urequests 0.6 from https://files.pythonhosted.org/packages/c9/0c/3dd3d54ea3bd70ae3173209d
ee5c4aa75edbf5f91abc79168ed367d9b1e/micropython-urequests-0.6.tar.gz
GC: total: 128064, used: 46912, free: 81152
No. of 1-blocks: 146, 2-blocks: 24, max blk sz: 2048, max free sz: 3950
```

图 6.1: 1525690379859

- 方法 2: 从 MicroPython-lib 中复制到开发板上文件系统的 `/libs/upy` 目录下。

接下来 `urequests` 模块就可以被导入使用了。

6.2.2 urequests 模块的使用

下面示例程序使用 `get` 命令来抓取 `http://www.baidu.com/` 的首页信息，并格式化输出：

```
try:
    import urequests as requests
except ImportError:
    import requests

r = requests.get("http://www.baidu.com/")
print(r)
print(r.content)
print(r.text)
r.close()
```

6.3 HttpServer

本章介绍如何使用 RT-Thread MicroPython 搭建一个 Web 服务器，需要使用到的模块为 `MicroWebSrv` 模块。

6.3.1 获取并安装 MicroWebSrv 模块

- 首先从 <https://github.com/jczic/MicroWebSrv.git> 将相关文件克隆到本地。
- 将 `www` 文件夹拷贝到文件系统的根目录 (这里将 SD 卡作为开发板文件系统的根目录)。

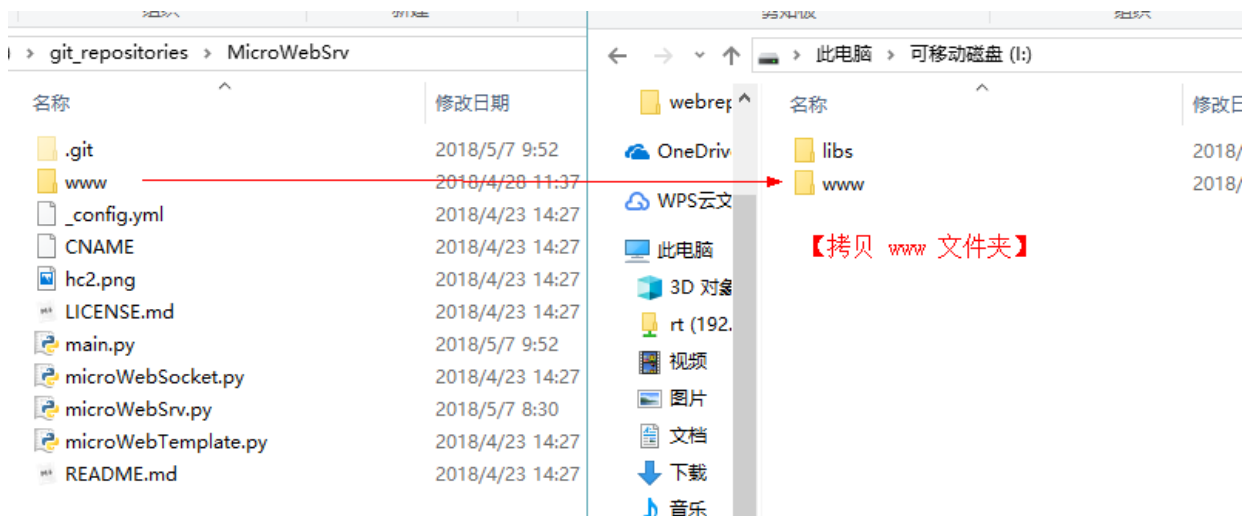


图 6.2: 1525674983856

- 把其他文件拷贝到 `/libs/mpy/` 文件夹中。

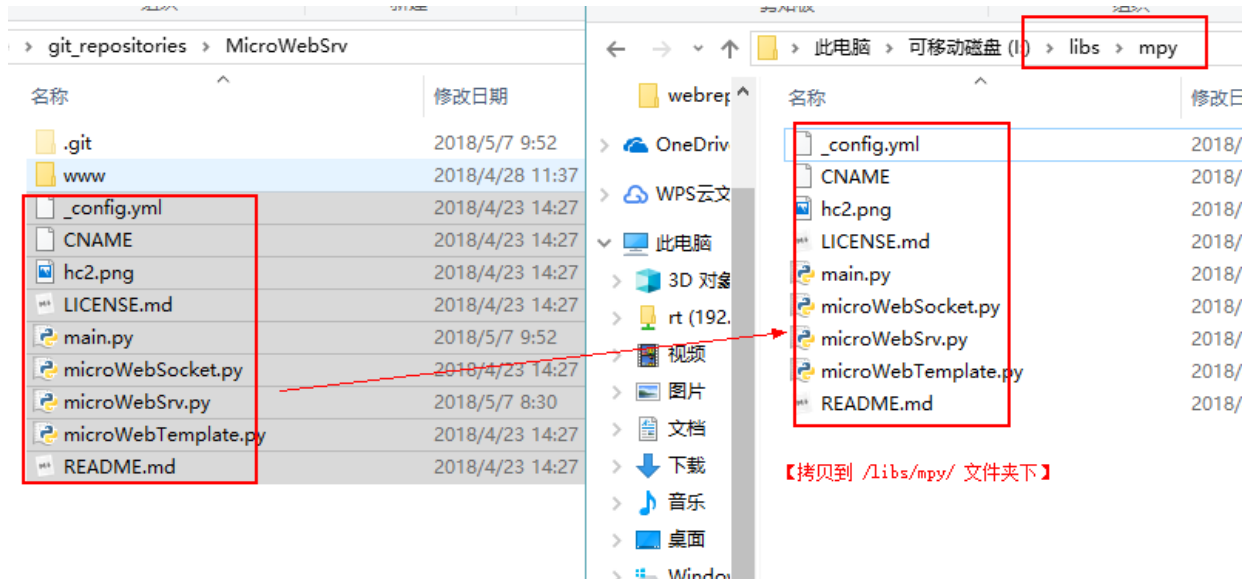


图 6.3: 1525675205931

- 这样 MicroWebSrv 模块就安装好了，可以在 MicroPython 交互命令行中直接使用 `import` 命令导入了。

6.3.2 MicroWebSrv 模块的使用

- 在 MSH 中，使用 `ifconfig` 命令查看开发板 IP 地址。
- 输入 `python` 命令，进入 MicroPython 交互命令行。
- 使用 `import main` 命令，启动 Web 服务器。

```
msh />ifconfig
network interface: e0 (Default)
MTU: 1500
MAC: 00 04 9f 05 43 72
FLAGS: UP LINK_UP ETHARP IGMP
ip address: 192.168.12.23
gw address: 192.168.10.1
net mask : 255.255.0.0
dns server #0: 192.168.10.1
dns server #1: 0.0.0.0
msh />python

MicroPython v1.9.3-477-g7b0a020-dirty on 2018-03-21; Universal python platform with RT-Thread
Type "help()" for more information.
>>> import main
```

图 6.4: 1525659036361

- 打开浏览器，在地址栏输入开发板 IP 地址并回车，即可看到 Web 页面。



图 6.5: 1525659139123

- 输入网址 ip/test 使用表格填写示例。



图 6.6: 1525659204069

如下代码完成了这个表格的获取功能：

```

@MicroWebSrv.route('/test')
def _httpHandlerTestGet(httpClient, httpResponse) :
    content = """\
<!DOCTYPE html>
<html lang=en>
  <head>
    <meta charset="UTF-8" />
    <title>TEST GET</title>
  </head>
  <body>
    <h1>TEST GET</h1>
    Client IP address = %s
    <br />
    <form action="/test" method="post" accept-charset="ISO-8859-1">
      First name: <input type="text" name="firstname"><br />
      Last name: <input type="text" name="lastname"><br />
      <input type="submit" value="Submit">
    </form>
  </body>
</html>
""" % httpClient.GetIPAddr()
    httpResponse.WriteResponseOk( headers = None,
                                   contentType = "text/html",
                                   contentCharset = "UTF-8",
                                   content = content )

```

图 6.7: 1525770427295

- 点击 Submit，服务器返回你填写的信息。



图 6.8: 1525659232565

如下代码完成了数据的推送功能：

```

@MicroWebSrv.route('/test', 'POST')
def _httpHandlerTestPost(httpClient, httpResponse) :
    formData = httpClient.ReadRequestPostedFormData()
    firstname = formData["firstname"]
    lastname = formData["lastname"]
    content = """\
<!DOCTYPE html>
<html lang=en>
  <head>
    <meta charset="UTF-8" />
    <title>TEST POST</title>
  </head>
  <body>
    <h1>TEST POST</h1>
    Firstname = %s<br />
    Lastname = %s<br />
  </body>
</html>
""" % ( MicroWebSrv.HTMLEscape(firstname),
        MicroWebSrv.HTMLEscape(lastname) )
    httpResponse.WriteResponseOk( headers = None,
                                   contentType = "text/html",
                                   contentCharset = "UTF-8",
                                   content = content )

```

图 6.9: 1525770467078

6.3.3 服务器功能的修改

- 如果想通过服务器实现自己所需的功能，可以修改 main.py 文件，导入更多模块，使用 Python 语言来添加更多功能。
- 在网页中展示加速度计和磁力计的例程中，下面的代码完成了这些数据的返回功能，可以参考 WebServer 的例子来对 main.py 进行修改，以达到自己想要完成的功能。

```

@MicroWebSrv.route('/sysdata')
def _httpHandlerTestGet(httpClient, httpResponse) :
    ax, ay, az = sensor.accelerometer
    mx, my, mz = sensor.magnetometer
    cpu_usage = machine.get_cpu_usage()
    cpu_value = cpu_usage[0] + cpu_usage[1] * 0.1
    ip = httpClient.GetIPAddr();

    content = {
        'status'      : 200,
        'body'        : {
            'status'   : 1,
            'result'   : {
                "versions": "3.0.3",
                "getTime": "1497594033",
                "cpuUtilization": cpu_value,
                "presentUtilization": 1,
                "ipAddress": ip,
                "key": 1,
                "Acceleration": { "accel_x" : ax, "accel_y" : ay, "accel_z" : az},
                "Magnetometer": { "mag_x" : mx, "mag_y" : my, "mag_z" : mz}
            }
        }
    }

    headers = {
        'Access-Control-Allow-Origin': '*',
        'Access-Control-Allow-Methods': 'POST,GET',
        'Access-Control-Allow-Headers': 'Content-Type, Authorization, Content-Length, X-Requested-With'
    }

    json_content = json.dumps(content)
    json_headers = json.dumps(headers)
    httpResponse.WriteResponseOk( headers = headers,
                                   contentType = "text/json",
                                   contentCharset = "UTF-8",
                                   content = json_content)

```

图 6.10: 1525770559437

6.4 MQTT

MQTT 是一种基于发布/订阅（publish/subscribe）模式的“轻量级”通讯协议。本章介绍如何在 RT-Thread MicroPython 上使用 MQTT 功能，使用到的模块为 `umqtt.simple` 模块。

6.4.1 获取并安装 `umqtt.simple` 模块

同样的可以使用包管理中的两种方式来获取，使用 `upip` 安装的方式可使用 `upip.install("micropython-umqtt.simple")` 如图：

```

>>> upip.install("micropython-umqtt.simple")
Installing to: /libs/mpy/
Warning: pypi.org SSL certificate is not validated
GC: total: 128064, used: 10592, free: 117472
No. of 1-blocks: 143, 2-blocks: 24, max blk sz: 32, max free sz: 5790
Installing micropython-umqtt.simple 1.3.4 from https://files.pythonhosted.org/packages/bd/cf/697e3418b2f44222b3e848078b1e33ee76aedca9b6c2430calblaecld/micropython-umqtt.simple-1.3.4.tar.gz
GC: total: 128064, used: 46928, free: 81136
No. of 1-blocks: 146, 2-blocks: 24, max blk sz: 2048, max free sz: 3742

```

图 6.11: 1525690229174

6.4.2 `umqtt.simple` 模块的使用

6.4.2.1 MQTT 订阅功能

- 使用 `iot.eclipse.org` 作为测试服务器

```

import time
from umqtt.simple import MQTTClient

# Publish test messages e.g. with:
# mosquitto_pub -t foo_topic -m hello

# Received messages from subscriptions will be delivered to this callback
def sub_cb(topic, msg):
    print((topic, msg))

def main(server="iot.eclipse.org"): # 测试 server 为 iot.eclipse.org
    c = MQTTClient("RT-Thread", server)
    c.set_callback(sub_cb)
    c.connect()
    c.subscribe(b"foo_topic") # 订阅 foo_topic 主题
    while True:
        if True:
            # Blocking wait for message
            c.wait_msg()
        else:
            # Non-blocking wait for message
            c.check_msg()
            # Then need to sleep to avoid 100% CPU usage (in a real
            # app other useful actions would be performed instead)
            time.sleep(1)

    c.disconnect()

if __name__ == "__main__":
    main()

```

- 使用 python 命令执行上述代码文件，就会连接上 MQTT 服务器，可收到我们从另一个客户端发布的以 `foo_topic` 为主题的内容

```

msh /libs/mpy>python example_sub.py
(b'foo_topic', b'Hello RT-Thread !!!')
(b'foo_topic', b'Today is a sunnyday !!!')

```

图 6.12: 1525665942426

6.4.2.2 MQTT 发布功能

- 执行下面的代码后将向 MQTT 服务器发布以 `foo_topic` 为主题的信息

```

from umqtt.simple import MQTTClient

# Test reception e.g. with:

```

```
# mosquitto_sub -t foo_topic

def main(server="iot.eclipse.org"):
    c = MQTTClient("SummerGift", server)
    c.connect()
    c.publish(b"foo_topic", b"Hello RT-Thread !!!")
    c.disconnect()

if __name__ == "__main__":
    main()
```

6.5 OneNET

本节介绍如何使用 RT-Thread MicroPython 来将设备接入 OneNET 云平台，本次示例使用的接入协议为 MQTT。

6.5.1 准备工作

- 首先需要安装 `urequests` 模块和 `umqtt.simple` 模块，安装方法参考 `HttpClient` 和 `MQTT` 章节。
- 本章实例代码在最后一节的附录中，可以在添加必要的注册信息后复制到 `main.py` 文件中在 MSH 中使用 `python` 命令来执行。

6.5.2 产品创建

- 想要将开发板接入 OneNET 云平台，首先要进行产品的创建，创建分为两步，第一步是注册一个用户账号，第二步是创建一个基于特定协议的产品。

6.5.2.1 用户注册

- 为了使用 OneNET 设备云的强大功能，首先要在 OneNET 上[注册开发者账号](#)，来创建专属的“开发者中心”。

6.5.2.2 产品创建

- 接下来需要在 OneNET 平台上[创建产品](#)。这里要注意的是在最后选择设备接入方式和设备接入协议时，因为本次示例使用的是 MQTT 协议，所以要在设备接入方式中选择公开协议，设备接入协议选择 MQTT。



设备接入方式：

☒ 公开协议 ☐ 私有协议(RGMP)

联网方式：

☒ wifi ☐ 移动蜂窝网络 ☐ NB-IoT

模组选择：

☒ 其他 ☐ 安信可 ESP8266-XX系列

☐ 庆科 EMW3081

设备接入协议：

MQTT

MQTT 协议基本功能介绍：
1、上报传感器数据点
2、订阅自定义TOPIC
3、接收实时消息或离线消息

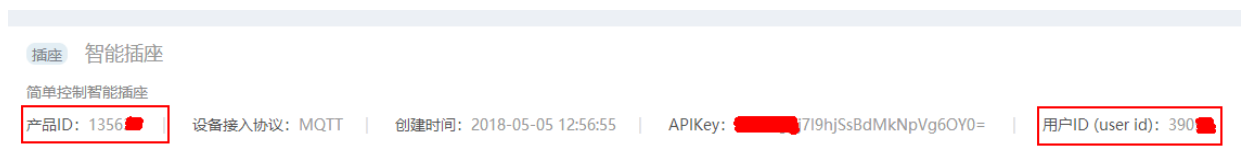
图 6.13: 1525764833130

6.5.3 硬件接入

本章节将介绍如何将设备接入到 OneNET 云平台上。并且演示一个云平台向设备发送命令，设备向云平台返回命令发送次数的示例。

6.5.3.1 设备的注册和接入

- 成功创建设备之后，将得到的产品 ID 记录下来供后面推送数据使用。



插座 智能插座

简单控制智能插座

产品ID: 1356 [REDACTED] | 设备接入协议: MQTT | 创建时间: 2018-05-05 12:56:55 | APIKey: [REDACTED]7I9hjSsBdMkNpVg6OY0= | 用户ID (user id): 390 [REDACTED]

图 6.14: 1525765493792

- 将设备的正式环境注册码记录下来用于注册新的设备。

接入设备

在接入设备时，请将以下注册码写入到设备中，只用于设备注册

正式环境注册码：3uy7LZsqblMe



图 6.15: 1525765209683

- 接下来打开例程中的 main.py，修改 sn 为设备唯一标识码，product_id 为上面得到的 6 位产品 ID，regKey 为上面记录下来的正式环境注册码。

```
def main():
    sn = 'RT_Thread_Test_Product'      #1、填入设备唯一标识码
    title = 'Device' + sn
    product_id = '5636'                #2、填入创建设备时获得的产品 ID
    regKey = '3uy7LZsqblMe'           #3、填入正式环境注册码
    url = 'http://api.heclouds.com/register_de?register_code=' + regKey
    reg = register.Register(url=url, title=title, sn=sn)      #根据上面的信息注册设备，如果已经注册不再重复注册
    if reg.regist()==0:
        mq = mqtt.mqtt(client_id=reg.device_id, username=product_id, password=reg.key) #开启 MQTT 服务
        mq.connect()
```

图 6.16: 1525766961043

- 在开发板中运行 main.py 即可在 OneNET 上看到我们注册的设备。

```
msh /libs/mpy>python main.py
Connected to 183.230.40.39, subscribed to topic_sub topic.
```

图 6.17: 1525767092149

- 名为 DeviceRT_Thread_Test_Product 的设备已经注册完毕并且上线。


全部	全部	关联应用数
	<p>● DeviceRT_Thread_Test_Product</p> <p>设备ID: 30102200</p> <p>创建时间: 2018年05月08日 16:11:18</p>	0个

图 6.18: 1525767167244

6.5.3.2 云平台向设备发送命令

- 可以通过发送命令功能来给开发板发送几组命令。

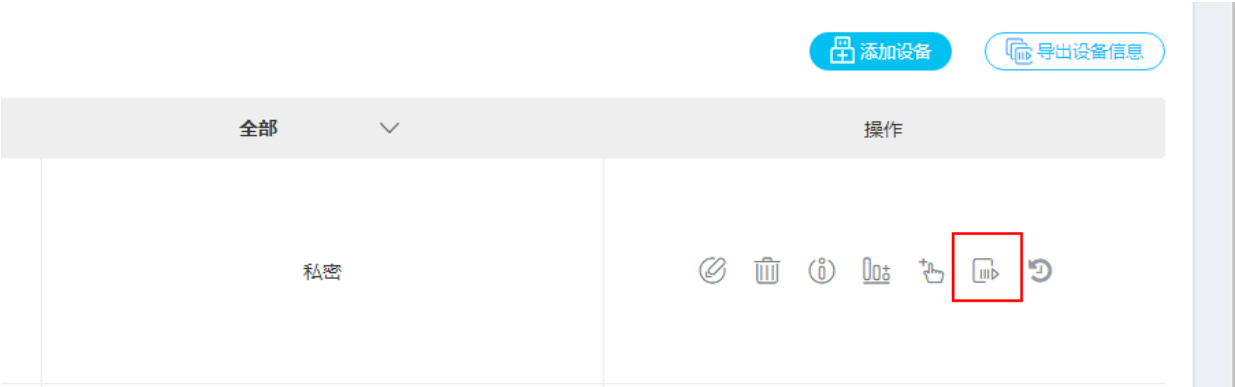


图 6.19: 1525767264155



图 6.20: 1525767369050

- 可以在设备端看到云平台下发的数据，同时设备端会上传命令发送次数的数据

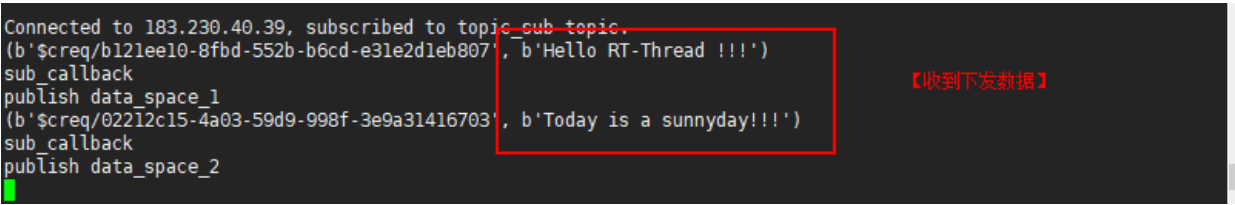


图 6.21: 1525767520609

6.5.3.3 设备向云平台上传数据

- 点击数据流管理功能来查看设备端上传的数据

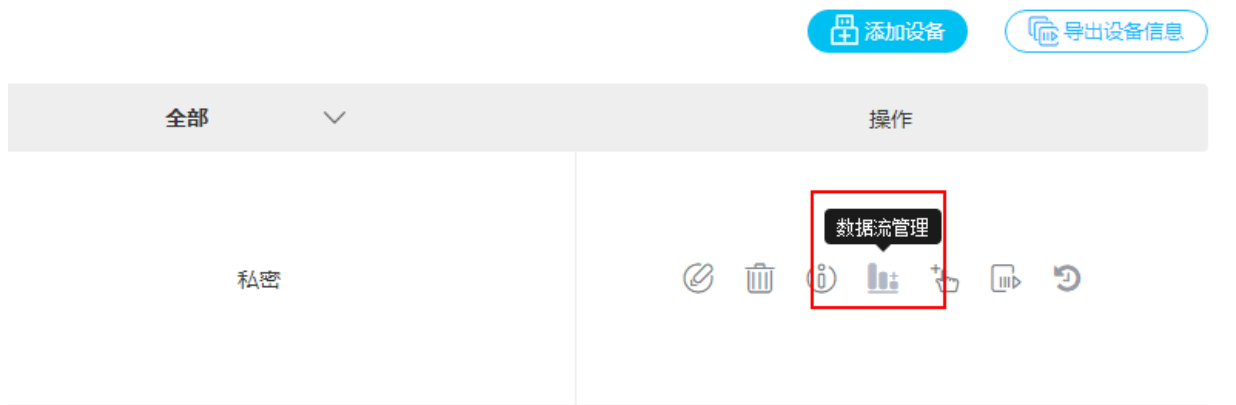


图 6.22: 1525767621708

- 可以在数据流管理中看到设备上传的命令发送次数 switch

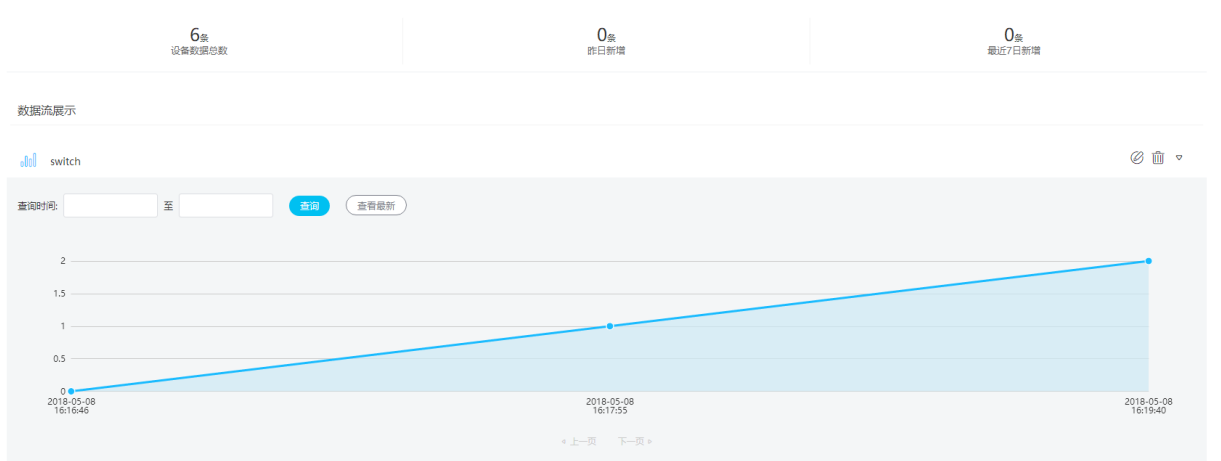


图 6.23: 1525767740215

- 这里的 switch 数据是在 mqtt.py 的 pubData 函数里面修改的，可以通过 value 对象的内容来给云平台上传不同的数据。

```
def pubData(self, t):
    value = {'datastreams': [{"id": "switch", "datapoints": [{"value": self.cmd_times}]}], {"id": "humi"}
    jdata = json.dumps(value)
    jlen = len(jdata)
    bdata = bytearray(jlen+3)
    bdata[0] = 1 # publish data in type of json
    bdata[1] = int(jlen / 256) # data lenght
    bdata[2] = jlen % 256 # data lenght
    bdata[3:jlen+4] = jdata.encode('ascii') # json data
    print('publish data', str(self.pid + 1))
    try:
        self.mqttClient.publish('$dp', bdata) # $dp 为特殊系统 topic, 可以通过这个 topic 给系统推送数据
        self.cmd_times += 1
```

图 6.24: 1525774755758

- 至此设备和 OneNET 云平台就对接好了。

6.5.3.4 添加独立应用

- 为了方便使用还可以给设备添加独立的应用，效果如下图：

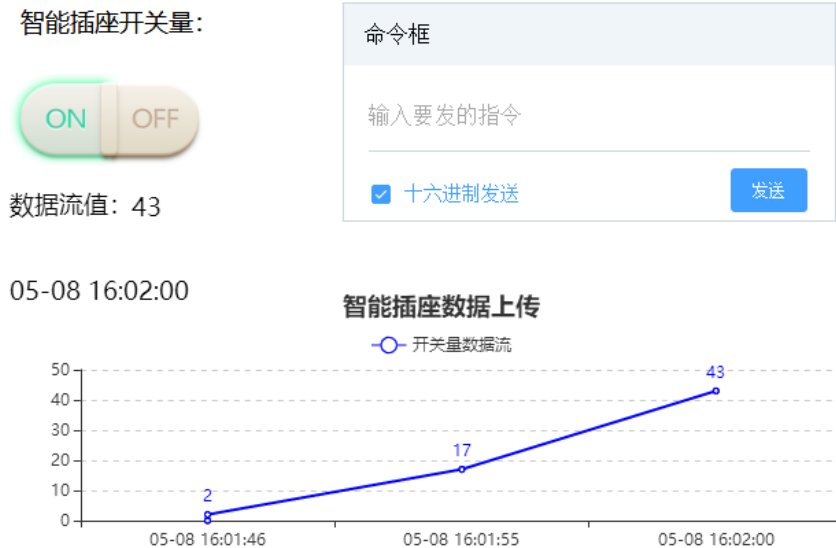


图 6.25: 1525768143233

6.5.4 代码讲解

- 通过修改 value 对象来修改向服务器发送的数据，这里是发送到特殊的系统 topic \$dp

```
def pubData(self, t):
    value = {'datastreams':[{"id":"switch","datapoints":[{"value": self.cmd_times}], {"id":"humi","datapoints":[{"value":0}]}]}
    jdata = json.dumps(value)
    jlen = len(jdata)
    bdata = bytearray(jlen+3)
    bdata[0] = 1 # publish data in type of json
    bdata[1] = int(jlen / 256) # data length
    bdata[2] = jlen % 256 # data length
    bdata[3:jlen+4] = jdata.encode('ascii') # json data
    print('publish data', str(self.pid + 1))
    try:
        self.mqttClient.publish('$dp', bdata) # $dp 为特殊系统 topic, 可以通过这个 topic 给系统推送信息, 但是不能订阅这个 topic
        self.cmd_times += 1
        self.failed_count = 0
    except Exception as ex:
        self.failed_count += 1
        print('publish failed:', ex.message())
        if self.failed_count >= 3:
            print('publish failed three times, esp resetting...')
            reset()
```

图 6.26: 1525774792209

- 给服务器发送数据的触发条件是收到服务器下发的命令，这样就没有保证会一直有数据发送到服务器，所以在没有数据交换的一段时间后，MQTT 连接有可能断开。

```
def connect(self):
    self.mqttClient.set_callback(self.sub_callback)
    self.mqttClient.connect()
    self.mqttClient.subscribe(self.topic)
    print("Connected to %s, subscribed to %s topic." % (self.server, self.topic))
    try:
        while 1:
            #self.mqttClient.wait_msg()
            self.mqttClient.check_msg()
            self.pubData('x')
```

图 6.27: 1525768433046

6.5.4.1 附录示例代码

```
import urequests as requests
from umqtt.simple import MQTTClient
import ujson as json
import time

class Register():
    def __init__(self, url='', title='', sn='', mac=''):
        self.url = url
        self.title = title
        self.sn = sn
        self.mac = mac
        self.sock = None
        self.tjson = {}
        self.erron = 0
        self.key = ''
        self.device_id = ''

    def regist(self):
        assert self.url is not None, "Url is not set"
        _, _, host, path = self.url.split('/', 3)
        if host == '':
            return
        device = {"mac":self.mac} if self.sn == '' else {"sn":self.sn}
        if self.title != '':
            device['title'] = self.title
        jdata = json.dumps(device)

        resp = requests.post(self.url, data=jdata)
        if resp:
            self.tjson = resp.json()
            if self.tjson['errno'] == 0:
                self.key = self.tjson['data']['key']
                self.device_id = self.tjson['data']['device_id']
            return 0
        else:
```

```

        return -1

class OneNetMqtt:
    failed_count = 0

    def __init__(self, client_id='', username='', password=''):
        self.server = "183.230.40.39"
        self.client_id = client_id
        self.username = username
        self.password = password
        self.topic = "topic_sub"          # 填入测试 topic
        self.mqttClient = MQTTClient(self.client_id, self.server, 6002, self.username,
                                     self.password)
        self.cmd_times = 0                # publish count

    def pubData(self, t):
        value = {'datastreams': [{"id": "switch", "datapoints": [{"value": self.
            cmd_times }]}]}
        jdata = json.dumps(value)
        jlen = len(jdata)
        bdata = bytearray(jlen+3)
        bdata[0] = 1                      # publish data in type of json
        bdata[1] = int(jlen / 256)         # data lenght
        bdata[2] = jlen % 256             # data lenght
        bdata[3:jlen+4] = jdata.encode('ascii') # json data
        print('publish data', str(self.cmd_times + 1))
        try:
            self.mqttClient.publish('$dp', bdata) # $dp 为特殊系统 topic, 可以通过
            这个 topic 给系统推送信息, 但是不能订阅这个 topic
            self.cmd_times += 1
            self.failed_count = 0
        except Exception as ex:
            self.failed_count += 1
            print('publish failed:', ex.message())
            if self.failed_count >= 3:
                print('publish failed three times, esp resetting...')
                reset()

    def sub_callback(self, topic, msg):
        print((topic, msg))
        cmd = msg.decode('ascii').split(" ")
        print('sub_callback')

    def connect(self):
        self.mqttClient.set_callback(self.sub_callback)
        self.mqttClient.connect()
        self.mqttClient.subscribe(self.topic)
        print("Connected to %s, subscribed to %s topic." % (self.server, self.topic)
        )

```

```

    try:
        while True:
            self.mqttClient.check_msg()
            print("pubdata")
            self.pubData('x')

    finally:
        self.mqttClient.disconnect()
        print('MQTT closed')

def main():
    sn = 'RT_Thread_Test_Product'           #1、填入设备唯一标识符
    title = 'Device' + sn
    product_id = 'XXXXX'                   #2、填入创建设备时获得的产品 ID
    regKey = 'XXXXXXXXX'                   #3、填入正式环境注册码
    url = 'http://api.heclouds.com/register_de?register_code=' + regKey

    reg = Register(url=url, title=title, sn=sn)    #根据上面的信息注册设备，如果已
    #注册不再重复注册
    if reg.regist()==0:
        MQTT = OneNetMqtt(client_id=reg.device_id, username=product_id, password=reg
            .key) #开启 MQTT 服务
        MQTT.connect()
    else:
        print('Error: No Client ID!')

if __name__ == "__main__":
    main()

```