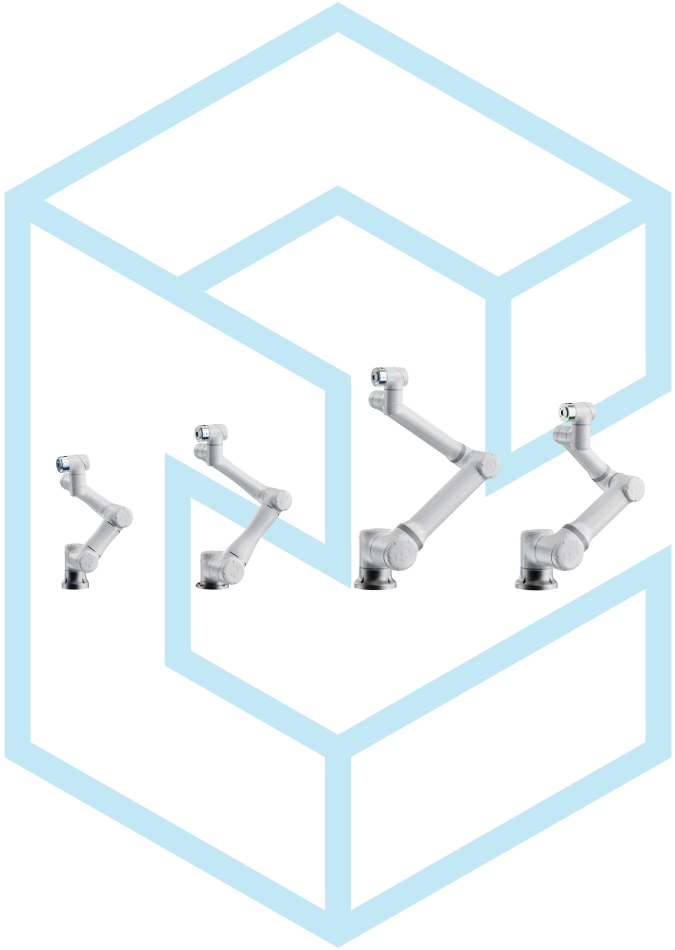
ELITE ROBOTS
艾利特机器人

ELITE ROBOTS EC系列
编程手册



Lua 脚本手册

艾利特智能机器人股份有限公司

2025-12-31

版本：Ver3.22.2

目录

第 1 章 语言定义

1.1 标识符与变量

标识符可以由非数字打头的任意字母，下划线和数字构成。可用于对变量，函数等命名。下列关键字是保留的，不可用于标识符命名:

and not or break return do then if else elseif end true false for while repeat until in function goto local

示例:

1. 标识符命名合法的：a _abc a_123
2. 数字常量是合法的：1 123 0xff 0xABCD
3. 合法的浮点常量：1.0 3.141592 1.6e-2 100e1
4. 字符串变量用""表示，例如："abcdef"
5. 数组变量以 {} 表示，例如：a={1.0,0.2,3.0}

全局变量与局部变量的作用域不同，声明局部变量前加上 local 关键字，例如 local a=5 ，否则就是一个全局变量。

EliteScript 中表达式语法是非常标准的，如下:

1. 算术表达式

   6+2-3

   5*2/3

   (2+3)*4/(5-6)
2. 逻辑表达式

   true or false and (2==3) and (1 ＞ 2)

   or (3~=4) or (5 ＜-6) not (9 ＞ =10)

   and 100 ＜ =50
3. 变量赋值

   A = 100

   bar = true

   PI = 3.1415

   name = "Lily"

   position = {0.1,-1.0,0.2,1.0,0.4,0.5}

EliteScript 中的变量类型不需在前面修饰，而是由变量的第一个赋值推导出的。比如上例中，A 是一个整数，bar 是一个布尔值，PI 是一个浮点数，name 是一个字符串，position 是一

个数组。

EliteScript 中的基本变量类型有：数字，布尔，字符串，数组。

1.2 控制流语句

程序的控制流可以通过 if, while, repeat, for 这些控制结构来实现，在 EliteScript 中，它们的语法规则都符合通常定义，语法为：

1.2.1 分支语句

if(exp1) then

——[[exp1 表达式为 true 时执行该语句块]]

else if(exp2) then

——[[exp2 表达式为 true 时执行该语句块]]

else

——[[满足其他条件时执行该语句块]]

end

例如：

```
a= -2
if(a<0) then
          print("a<0")
elseif(a>0) then
         print("a>0") else
         print("a=0");
end
```

1.2.2 while 循环

while(exp) do

—— [[exp 表达式为 true 时循环执行该语句块]]

end

例如：

```
while(true) do
        print("A");
end
```

1.2.3 repeat 循环

repeat

——[[exp 表达式为 false 时循环执行该语句块]]

until(exp)

如下程序打印：10 11 12 13 14 15

```
A = 10
repeat
        print(A)
        A = A+1
until(A>15)
```

1.2.4 for 循环

for var=exp1,exp2,exp3 do

——[[执行的语句块]]

end

如下程序打印：10 9 8 7 6 5 4 3 2 1

```
for i=10,1,-1 do
        print(i)
end
```

可以使用 break 停止某个循环, 使用 goto 语句实现跳转，可以通过 return 直接返回。

特别注意，EliteScript 不支持 continue 语句，但可以使用 goto 间接实现。

1.3 操作符

数字操作符	含义
+	加法
*	乘法
–	减法
/	浮点除法
//	向下取整除法
%	取模
^	乘方
–	取负

位操作符	含义
&	按位与
|	按位或
~	按位异或
>>	右移
<<	左移
~	按位非

比较操作符	含义
==	等于
~=	不等于
<	小于
>	大于
< =	小于等于
> =	大于等于

逻辑操作符	含义
and	与
or	或
not	非

字符串连接符：写作两个点..，如 12..34，等价 1234。

取长度操作符：为 #，字符串的长度是它的字节数。

优先级定义（由低向高）：

or

and

<，>，<=，>=，~=，==

|

~

&

«，»

..

+，-

*，/，//，%

#，—(unary)，not

^

可以用括号来改变运算次序。连接操作符.. 和乘方操作 ^ 是从右至左的。其它所有的操作都是从左至右。

1.4 函数

1.4.1 函数定义语法

函数定义的语法如下：

```
function MyFunc(param)
--Do something
end
```

1.4.2 自定义函数

自定义加函数：

function add(a,b)

return (a+b)

end

函数调用：

x= add(3,4)

函数可以无返回值，也可以有一个或多个返回值，例如:

```
function add(a,b)
        return a,b,(a+b)
end
```

函数调用：

x,y,z=add(a,b)

1.4.3 分割字符串

string.sub(strccd, 起始位置, 终止位置)

-- 获取指定位置长度的字符串

string.len(目标字符串)

-- 获取字符串的长度函数调用

function string.split(str, delimiter)

-- 分割字符串，带分隔字符串，返回数组，str 要分离的字符串，delimiter 分隔符

```lua
function string.split(str, delimiter)
        if str==nil or str=='' or delimiter==nil then
                return nil
        end
        local result = {}
        for match in (str..delimiter):gmatch("(.-)"..delimiter) do
                table.insert(result, match)
        end
                return result
end
```

内在有 string:split 进行字符串分隔

1.4.4 string 字符串

string.len(s) 返回字符串 s 的长度

string.lower(s) 将 s 中的大写字母转换成小写

string.upper(s) 将 s 中的小写字母转换成大写

string.sub(s,i,j) 函数截取字符串 s 的从第 i 个字符到第 j 个字符之间的串。

string.char() 字符转换成数字

string.byte() 数字转换成字符

1.4.4.1 字符串格式化

string.format() 函数来生成具有特定格式的字符串，函数的第一个参数是格式，之后是对应格式中每个代号的各种数据。

以下实例演示了如何对字符串进行格式化操作：

格式字符串可能包含以下的转义码:

%c - 接受一个数字, 并将其转化为 ASCII 码表中对应的字符

string.format(“%c”, 83) -- 输出 S

%d, %i - 接受一个数字并将其转化为有符号的整数格式

string.format(“%+d”, 17.0) -- 输出 +17

%o - 接受一个数字并将其转化为八进制数格式

string.format(“%o”, 17) -- 输出 21

%u - 接受一个数字并将其转化为无符号整数格式

string.format(“%u”, 3.14) -- 输出 3

%x - 接受一个数字并将其转化为十六进制数格式, 使用小写字母

string.format(“%x”, 13) -- 输出 d

%X - 接受一个数字并将其转化为十六进制数格式, 使用大写字母

string.format(“%X”, 13) -- 输出 D

%e - 接受一个数字并将其转化为科学记数法格式, 使用小写字母 e

string.format(“%e”, 1000) -- 输出 1.000000e+03

%E - 接受一个数字并将其转化为科学记数法格式, 使用大写字母 E

string.format(“%E”, 1000) -- 输出 1.000000E+03

%f - 接受一个数字并将其转化为浮点数格式

string.format(“%6.3f”, 13) -- 输出 13.000

%g(%G) - 接受一个数字并将其转化为%e(%E, 对应%G) 及%f 中较短的一种格式

%q - 接受一个字符串并将其转化为可安全被 Lua 编译器读入的格式

string.format(“%q”, “One\nTwo”) -- 输出 “One\Two”

%s - 接受一个字符串并按照给定的参数格式化该字符串

string.format(“%s”, “monkey”) -- 输出 monkey

string.format(“%10s”, “monkey”) -- 输出 monkey

1.5 http client 支持

```lua
local http = require("socket.http")
local ltn12 = require("ltn12")

-- 编辑请求体
local reqbody = '{"hello":"elite", "new":"cs"}'
-- 编辑请求头
local headers = {
    -- 发送内容类型["Content-Type"] = "application/x-www-form-urlencoded";
    ["Content-Type"] = "application/json",
    -- POST必须包含该字
    ["Content-Length"] = #reqbody;
}
-- 响应体
local respbody = {}

--------------------------------------------------------------------------------

-- 发送GET请求
-- res:发送结果 code:状态码 response_headers:接收请求头
local res, code, response_headers = http.request({
    url = "http://192.168.11.11:6680/",
    method = "GET",
    headers = headers,
    sink = ltn12.sink.table(respbody)
})

elite_print("发送结果: ", res)
elite_print("状态码:", code)
-- 输出接收的请求头
for index, value in pairs(response_headers) do
    elite_print(index, value)
end
-- 输出响应体
elite_print(table.concat(respbody))

--------------------------------------------------------------------------------

-- 发送POST请求
-- res:发送结果 code:状态码 response_headers:接收请求头
```

```lua
local res, code, response_headers = http.request({
    url = "http://172.16.11.251:6680/api/api",
    source = ltn12.source.string(reqbody),
    method = "POST",
    headers = headers,
    sink = ltn12.sink.table(respbody)
})

elite_print("发送结果: ", res)
elite_print("状态码:", code)
-- 输出接收的请求头
for index, value in pairs(response_headers) do
    elite_print(index, value)
end
-- 输出响应体
elite_print(table.concat(respbody))
```

第 2 章 内部方法列表

2.1 延时

```
sleep(second)
```

功能：

sleep(second) 用于睡眠等待

参数：

second：

等待时间，double类型，单位：秒

返回值：

nil

备注：

无

示例：

```
sleep(0.1)
```

2.2 设置机械臂本体 I/O 状态

```
set_robot_io_status(name,value)
```

功能：

set_robot_io_status(name,value) 用于设置机械臂本体 I/O 状态，可以设置 Y、M、AO 变量的状态。

参数：

name：

I/O 名称，string类型

value：

I/O 状态值，int 类型（对应 Y，M）或 double 类型（对应 AO）

返回值：

nil

备注：

设置 Y 变量的值时，参数 name 为“o**”，例如“o0”表示“Y0”

设置模拟量输出值时，参数 name 为“a**”，比如“a0”表示“AO001”，“a1”表示“AO002”，以此类推。

设置虚拟量输出值时，参数name为“M**”，比如“M528”表示“VOUT528”。

本命令适用于v2.9.3及以上版本。

示例：

```
set_robot_io_status("M528",1)
```

2.3 获取机器人臂本体 I/O 状态

```
ret get_robot_io_status(name)
```

功能：

get_robot_io_status(name) 用于获取机械臂本体 I/O 状态，可以获取 X、Y、M、AI 变量的状态

参数：

name：

I/O 名称, string类型

返回值：

ret：

对应 I/O 的状态值

备注：

1. 本命令适用于v2.9.3及以上版本。

2. 若需获取X变量（数字输入）的状态，参数name需设置为“i**”，如“i0”表示X0；若需获取Y变量（数字输出）的状态，参数name需设置为“o**”，如“o0”表示Y0；若需获取AI变量（模拟输入）的状态，参数name需设置为“a**”。例如“a0”表示AI001，以此类推。

示例：

```
--获取M变量（虚拟输入/输出），如获取M20变量
ret=get_robot_io_status("M20")
--获取X变量，如获取X1变量
ret=get_robot_io_status("i1")
--获取Y变量，如获取Y2变量
ret=get_robot_io_status("o2")
--获取AI变量，如获取AI003变量
ret=get_robot_io_status("a2")
```

2.4 获取示教器全局变量值

```
ret get_global_variable(varName)
```

功能：

get_global_variable(varName) 用于获取示教器全局变量值，支持获取 B、I、D、P、V 变量

参数：

varName：

变量名称, string类型

返回值：

ret：

对应变量名称的变量值，返回值类型取决于变量的类型

备注：

获取 I 变量时，范围为 0~8255。

示例：

```
--获取B、I、D变量，如获取B变量
ret=get_global_variable("B20")
elite_print(ret)
--获取B、I、D变量，如获取P变量
a1,a2,a3,a4,a5,a6=get_global_variable("P20")
elite_print(a1,a2,a3,a4,a5,a6)
```

2.5 设置示教器全局变量值

```
set_global_variable(varName,varValue)
```

功能：

set_global_variable(varName,varValue) 用于设置示教器全局变量值，支持获取 B、I、D、P、V 变量

参数：

varName：

变量名称，string类型

varValue：

变量值，variant类型

返回值：

nil

备注：

设置 I 变量，范围为 0~8255。

示例：

```
--设置B、I、D变量，如设置B变量
set_global_variable ("B20", 1)
--设置P、V变量，如设置P变量
set_global_variable("P20",90,0,-90,80,30,60)
```

2.6 设置是否调试模式

```
set_debug(debug)
```

功能：

set_debug(debug) 用于设置是否调试模式

参数：

debug：

int类型

0：不输出 INFO 信息

1：输出 INFO 信息

返回值：

nil

备注：

该函数为 1 时，输出内部模块的调试信息，不影响 elite_print 的功能。

示例：

```
set_debug(1)
```

2.7 打印信息至示教器界面

```
elite_print(var1,var2....)
```

功能：

elite_print(var1,var2....) 用于打印指定信息到示教器中信息提示窗口

参数：

var1：

string类型

返回值：

nil

备注：

无

示例：

```
num=10
elite_print("test",tostring(10))
```

2.8 逆解函数

```
ret get_inv_kinematics(var1,var2)
```

功能：

get_inv_kinematics(var1,var2) 用于逆解函数

参数：

var1：

pose（目标点位姿）, table 类型

var2：

joint（参考点关节角度，参考点需接近目标点。若不写则视为参考当前点），table类型

返回值：

ret：

逆解结果无/

table:joint（目标点关节角度）

备注：

pose={x,y,z,rx,ry,rz}，size 为 6 的数组（注意：pose 中的 rx,ry,rz 均为弧度，下同）

joint={j1,j2,j3,j4,j5,j6}size 为 6 的数组

示例：

```
pose={378.538,212.504,134.055,-2.712,-0.791,2.553}
joint={10.081,-75.007,105.449,-70.694,98.434,89.481}
ret=get_inv_kinematics(pose,joint)
```

2.9 正解函数

```
ret get_fwd_kinematics(var1)
```

功能：

get_fwd_kinematics(var1) 用于正解函数

参数：

var1：

joint（目标点关节角度）, table 类型

返回值：

ret：

正解结果空/

table:pose（目标点位姿）

备注：

pose={x,y,z,rx,ry,rz}size 为 6 的数组
joint={j1,j2,j3,j4,j5,j6}size 为 6的数组

示例：

```
joint={10.081,-75.007,105.449,-70.694,98.434,89.481}
ret=get_fwd_kinematics(joint)
```

2.10 位姿的逆

```
ret pose_inv(var1)
```

功能：

pose_inv(var1) 用于返回 p 位姿的逆

参数：

var1：

pose位姿数据，table类型

返回值：

ret：

结果空/

table:pose

备注：

pose={x,y,z,rx,ry,rz}size 为 6 的数组

示例：

```
ret=pose_inv({0,1,2,3,4,5})
```

2.11 位姿的乘

```
ret pose_mul(var1,var2)
```

功能：

pose_mul(var1,var2) 用于返回位姿的乘

参数：

var1：

pose位姿数据，table类型

var2：

pose位姿数据，table类型

返回值：

ret：

结果空/

table:pose

备注：

pose={x,y,z,rx,ry,rz}size 为 6 的数组

示例：

```
ret=pose_mul({0,1,2,3,4,5},{0,1,2,3,4,5})
```

2.12 获取机器人模式

```
ret get_robot_mode()
```

功能：

get_robot_mode() 用于获取机器人模式

参数：

无

返回值：

ret：

0：示教模式

1：自动模式

2：远程模式

备注：

本命令适用于v2.9.3及以上版本。

示例：

```
ret=get_robot_mode()
```

2.13 获取机器人 JBI 运行模式

```
ret get_cycle_mode()
```

功能：

get_cycle_mode() 用于获取机器人 JBI 运行模式

参数：

无

返回值：

ret：

0：单步

1：单循环

2：连续循环

备注：

本命令适用于v2.9.3及以上版本。

示例：

```
ret=get_cycle_mode()
```

2.14 获取机器人伺服使能状态

```
ret get_servo_status()
```

功能：

get_servo_status() 用于获取机器人伺服使能状态

参数：

无

返回值：

ret：

0：伺服使能关闭

1：伺服使能打开

备注：

本命令适用于v2.9.3及以上版本。

示例：

```
ret=get_servo_status()
```

2.15 获取机器人运行状态

```
ret get_robot_state()
```

功能：

get_robot_state() 用于获取机器人运行状态

参数：

无

返回值：

ret：

0：停止
1：暂停
3：运行中
4：报警
5：碰撞

备注：

无

示例：

```
ret=get_robot_state()
```

2.16 获取机器人坐标系

```
ret get_current_coord()
```

功能：

get_current_coord() 用于获取机器人坐标系

参数：

无

返回值：

ret：

0：关节坐标系
1：基座坐标系
2：工具坐标系
3：用户坐标系
4：圆柱坐标系

备注：

本命令适用于v2.9.3及以上版本。

示例：

```
ret=get_current_coord()
```

2.17 获取机器人位姿

```
ret get_robot_pose()
```

功能：

get_robot_pose() 用于获取机器人位姿

参数：

无

返回值：

ret：

机器人当前位姿 [x,y,z,rx,ry,rz]

注：rx、ry、rz 为弧度

备注：

本命令适用于v2.9.3及以上版本。

本命令不推荐使用。

示例：

```
ret=get_robot_pose()
```

2.18 获取机器人关节角度

```
ret get_robot_joint()
```

功能：

get_robot_joint() 用于获取机器人关节角度（以角度返回）

参数：

无

返回值：

ret：

机器人当前关节角度 [joint1,joint2,joint3,joint4,joint5,joint6]

备注：

本命令适用于v2.9.3及以上版本。

示例：

```
ret=get_robot_joint()
```

2.19 获取机器人力矩信息

```
ret get_robot_torque()
```

功能：

get_robot_torque() 用于获取机器人力矩信息

参数：

无

返回值：

ret：

机器人当前位姿各关节的力矩 [torque1,torque2,torque3,torque4,torque5,torque6]

备注：

本命令适用于v2.9.3及以上版本。

示例：

```
ret=get_robot_torque()
```

2.20 获取当前tcp坐标系下的“外部”力和力矩

```
ret get_tcp_force(ref_tcp)
```

功能：

get_tcp_force(ref_tcp) 用于获取当前tcp或基坐标系的“外部”力和力矩

参数：

ref_tcp：

参考坐标系,int[0,7], 可选参数, 0代表基坐标系，1代表tcp坐标系，不写默认为tcp坐标系。

返回值：

ret：

当机器人处于奇异位置时，则返回null，当机器人处于非奇异位置时，则返回当前tcp或基坐标系的“外部”力和力矩

备注：

本命令适用于v3.4.2及以上版本。

示例：

```
ret=get_tcp_force()
```

2.21 获取当前关节力矩

```
ret get_joint_torques()
```

功能：

get_joint_torques() 用于获取当前关节力矩

参数：

无

返回值：

ret：

机器人当前关节力矩 [J1_torq,J2_torq,J3_torq,J4_torq,J5_torq,J6_torq]

备注：

关节力矩为电机力矩减去“自身驱动所需力矩”，反应的是“外部”力矩。本命令适用于v3.4.2及以上版本。

示例：

```
ret=get_joint_torques()
```

2.22 获取真实的末端位姿数据

```
ret get_actual_tcp(int tool_num,int user_num)
```

功能：

get_actual_tcp(int tool_num, int user_num) 用于获取基坐标系或者指定用户坐标系下的真实末端位姿数据

参数：

tool_num：

工具坐标编号，可选，int[-1,7]，若将tool_num设置为-1，则表示当前工具号，否则为指定工具号

user_num：

用户坐标编号，可选，int[-1,15]，若将user_num设置为-1，则表示获取的位姿是在基坐标系，否则为指定用户坐标系下的位姿

返回值：

ret：

table位姿数据

备注：

本命令适用于v3.5.2及以上版本。

示例：

```
ret=get_actual_tcp(0,0)
```

2.23 获取目标插补末端位姿数据

```
ret get_target_tcp(int tool_num,int user_num)
```

功能：

get_target_tcp(int tool_num, int user_num) 用于获取基坐标系或者指定用户坐标系下的目标插补末端位姿数据

参数：

tool_num：

工具坐标编号，可选，int[-1,7]，若将tool_num设置为-1，则表示当前工具号，否则为指定工具号

user_num：

用户坐标编号，可选，int[-1,15]，若将user_num设置为-1，则表示获取的位姿是在基坐标系，否则为指定用户坐标系下的位姿

返回值：

nil

备注：

本命令适用于v3.5.2及以上版本。

示例：

```
ret=get_target_tcp(0,0)
```

2.24 获取线性插补位姿数据

```
ret get_interp_pose(table var1,table var2,double ratio)
```

功能：

get_interp_pose(table var1, table var2, double ratio) 用于获取指定的两个位姿之间的线性插补位姿数据

参数：

var1：

位姿数据，double pose[6]，前三位代表位置，单位x、 y、 z为毫米，范围是[-∞，+∞]，后三位代表姿态，单位Rx、 Ry、 Rz为弧度，范围是[-π, π]

var2：

位姿数据，double pose[6]，前三位代表位置，单位x、 y、 z为毫米，范围是[-∞，+∞]，后三位代表姿态，单位Rx、 Ry、 Rz为弧度，范围是[-π, π]

ratio：

浮点型数据，代表比例值，范围是[0,1]，当数值=0时，机器人返回第一个位姿，当数值=1时，机器人返回第二个位姿

返回值：

ret：

table位姿数据

示例：

```
pose1={371.534, 101.636, 3.038, 0, -0.174, 2.861}
pose2={346.312, -256.945, -91.131, -0.142, 0.521, 1.903}
interp_pose=get_interp_pose(pose1,pose2,0.1)
```

2.25 获取真实的关节数据

```
ret get_actual_joint()
```

功能：

get_actual_joint() 用于获取当前真实的关节数据

参数：

无

返回值：

ret：

table关节数据

备注：

本命令适用于v3.5.2及以上版本。

示例：

```
ret=get_actual_joint()
```

2.26 获取目标插补关节数据

```
ret get_target_joint()
```

功能：

get_target_joint() 用于获取当前目标插补关节数据

参数：

无

返回值：

ret：

table关节数据

备注：

本命令适用于v3.5.2及以上版本。

示例：

```
ret=get_target_joint()
```

2.27 获取工具坐标系位姿

```
ret get_tool_frame(num)
```

功能：

get_tool_frame(num) 用于获取toolframe中的数据

参数：

num：

工具坐标编号,int[0,7], 可选参数, 如果 num 未设置，那么获取当前的工具坐标系位姿。

返回值：

ret：

tool[x,y,z,rx,ry,rz]

注：rx、ry、rz 为弧度

备注：

本命令适用于v2.9.4及以上版本。

示例：

```
ret=get_tool_frame(0)
```

2.28 获取用户坐标系位姿

```
ret get_user_frame(num)
```

功能：

get_user_frame(num) 用于获取 userframe 中的数据

参数：

num：

用户坐标编号,int[0,7], 可选参数, 如果 num 未设置，那么获取当前的用户坐标系位姿。

返回值：

ret：

user[x,y,z,rx,ry,rz]

注：rx、ry、rz 为弧度

备注：

本命令适用于v2.9.4及以上版本。

示例：

```
ret=get_user_frame(0)
```

2.29 获取工具坐标系编号

```
ret get_tool_no()
```

功能：

get_tool_no() 用于获取当前的工具坐标系编号

参数：

无

返回值：

ret：

当前机器人工具坐标系编号

备注：

本命令适用于v2.17.0及以上版本。

示例：

```
ret=get_tool_no()
```

2.30 获取用户坐标系编号

```
ret get_user_no()
```

功能：

get_user_no() 用于获取当前的用户坐标系编号

参数：

无

返回值：

ret：

当前机器人用户坐标系编号

备注：

本命令适用于v2.17.0及以上版本。

示例：

```
ret=get_user_no()
```

2.31 检查工具启用状态

```
ret check_tool_frame_enable(num)
```

功能：

check_tool_frame_enable(num) 用于检查指定工具是否被启用

参数：

num：

工具坐标编号，int[0,7]

返回值：

ret：

0：当前工具号未启用

1：当前工具号启用

备注：

本命令适用于v2.9.4及以上版本

示例：

```
ret=check_tool_frame_enable(0)
```

2.32 检查用户坐标启用状态

```
ret check_user_frame_enable(num)
```

功能：

check_user_frame_enable(num) 用于检查指定用户坐标是否被启用

参数：

num：

用户坐标编号，int[0,15]

返回值：

ret：

0：当前用户坐标号未启用

1：当前用户坐标号启用

备注：

本命令适用于v2.9.4及以上版本

示例：

```
ret=check_user_frame_enable(0)
```

2.33 获取 M 变量寄存器接口 (0-191)

```
recv,ret get_robot_register(index)
```

功能：

get_robot_register(index) 用于获取 M 变量的寄存器接口

参数：

index：

字节定义，int[0,191]

返回值：

recv：

寄存器的接口值

ret：

0：成功

-1：失败

备注：

本命令适用于v2.16.2及以上版本。

示例：

```
recv ,ret=get_robot_register(77)
```

2.34 设置 M 变量寄存器接口 (66-191)

```
set_robot_register(index,size,value)
```

功能：

set_robot_register(index,size,value) 用于设置 M 变量的八位寄存器接口

参数：

index：

寄存器地址，int类型[66,191]

size：

单位是字节，int类型[1,128]，且与 index 值的和需小于等于 192

value：

寄存器值，hex 字符串或unsgined int类型

注：当value类型为unsgined int时，size的范围为[1,4]，value的范围为[0,2^(8*size)-1]。

返回值：

nil

备注：

本命令适用于v2.16.2及以上版本。

示例：

```
set_robot_register (66,1,"01") -执行结果，M528-M535数据依次为"10000000"
```

2.35 获取 M 变量寄存器接口（192-575）

```
recv,ret get_robot_extra_register(index)
```

功能：

get_robot_extra_register(index) 用于获取 M 变量的寄存器接口

参数：

index：

字节定义，int类型[192,575]

返回值：

recv：

寄存器的接口值

ret：

0：成功

-1：失败

备注：

本命令适用于v2.16.2及以上版本。

示例：

```
recv,ret=get_robot_extra_register(192)
```

2.36 设置 M 变量寄存器接口（300-447）

```
set_robot_extra_register(index,size,value)
```

功能：

set_robot_extra_register(index,size,value) 用于设置 M 变量的十六位寄存器接口

参数：

index：

寄存器地址，int类型[300,447]

size：

int类型，单位是字节，且与 index 值的和需小于等于 448

value：

寄存器值，hex 字符串或unsgined int类型

注：当value类型为unsgined int时，size的范围为[1,4]，value的范围为[0,2^(8*size)-1]。

返回值：

nil

备注：

系统占用的寄存器不可修改，详细接口请参见通信协议手册。

本命令适用于v2.16.2及以上版本。

示例：

`set_robot_extra_register (300,2,"0F0E")`。此时`get_robot_extra_register(300)`，返回值为"35999 (0E0F)"（寄存器存储模式为小端模式，数据高位对应地址高8位，数据低位对应地址低8位）。

2.37 获取当前 tcp 速度

```
ret get_current_tcp_spd()
```

功能：

get_current_tcp_spd() 用于获取当前 tcp 速度

参数：

无

返回值：

ret：

当前 tcp 速度，单位：mm/s

备注：

本命令适用于v2.16.2及以上版本。

示例：

```
ret=get_current_tcp_spd()
```

2.38 获取法兰中心在当前基坐标系下的位姿

```
ret get_flange_pose_inbase()
```

功能：

get_flange_pose_inbase() 用于获取法兰中心在当前基坐标系下的位姿数据

参数：

无

返回值：

ret：

机器人当前位姿 [x,y,z,rx,ry,rz]

注：rx、ry、rz 为弧度

备注：

本命令适用于v2.17.0及以上版本。

示例：

```
ret=get_flange_pose_inbase()
```

2.39 获取法兰中心在当前用户坐标系下的位姿

```
ret get_flange_pose_inuser()
```

功能：

get_flange_pose_inuser() 用于获取法兰中心在当前用户坐标系下的位姿

参数：

无

返回值：

ret：

机器人当前用户坐标系位姿 [x,y,z,rx,ry,rz]

注：rx、ry、rz 为弧度

备注：

本命令适用于v2.17.0及以上版本。

示例：

```
ret=get_flange_pose_inuser()
```

2.40 获取机器人位姿

```
ret get_tcp_pose()
```

功能：

get_tcp_pose() 用于获取机器人位姿

参数：

无

返回值：

ret：

机器人当前位姿 [x,y,z,rx,ry,rz]

注：rx、ry、rz 为弧度

备注：

本命令适用于v2.17.0及以上版本。

示例：

```
ret=get_tcp_pose()
```

2.41 获取当前 tcp 在当前用户坐标系下的位姿

```
ret get_tcp_pose_inuser()
```

功能：

get_tcp_pose_inuser() 用于获取当前 tcp 在当前用户坐标系下的位姿

参数：

无

返回值：

ret：

机器人当前 tcp 在当前用户坐标系下的位姿 [x,y,z,rx,ry,rz]

注：rx、ry、rz 为弧度

备注：

本命令适用于v2.17.0及以上版本。

示例：

```
ret=get_tcp_pose_inuser()
```

2.42 获取末端 485 模式

```
ret get_terminal_485_mode()
```

功能：

get_terminal_485_mode() 用于获取末端 485 模式

参数：

无

返回值：

ret：

0：初始模式

1：tci 通讯模式

2：modbusRTU 主站模式

备注：

本命令适用于v2.17.0及以上版本。

示例：

```
ctx=modbus_new_rtu(2,9600,78,8,1)
sleep(2)
ret=get_terminal_485_mode()
elite_print(ret)
```

2.43 获取两点距离

```
ret get_point_distance(table var1, table var2)
```

功能：

get_point_distance (table var1, table var2) 用于获取笛卡尔空间下两点的直线距离

参数：

var1:

位姿数据，double pose [6], 前三位代表位置，单位x、y、z为毫米，范围是[-∞，+∞]，后三位代表姿态，单位 Rx、Ry、Rz为弧度，范围是[-π, π]

var2:

位姿数据，double pose [6]，前三位代表位置，单位x 、y、z为毫米，范围是[-∞，+∞]，后三位代表姿态，单位 Rx、Ry、Rz为弧度，范围是[-π, π]

返回值：

ret：

两点距离，数据类型为浮点型，单位为毫米

备注：

无

示例：

```
table1={499,19,423,-3.14,0,-1.57}
table2={500,122,454,-3.14,0,-1.57}
dist=get_point_distance(table1,table2)
elite_print("dist", tostring(dist))
```

2.44 获取关节温度

```
ret get_joint_temp()
```

功能：

get_joint_temp() 用于获取关节角度

参数：

无

返回值：

ret：

double joint_temp[6]

示例：

```
ret=get_joint_temp()
elite_print(ret[1],ret[2],ret[3],ret[4],ret[5],ret[6])
```

2.45 计算位姿变化量

```
ret pose_sub(var1,var2)
```

功能：

pose_sub(var1,var2) 用于计算位姿变化量

参数：

var1：

double pose[6]：位姿

var2：

double pose[6]：位姿

返回值：

ret：

空：失败

double pose[6]位姿变化量：成功

备注：

pose前三位代表位置，X，Y，Z的单位为毫米，后三位代表姿态，Rx，Ry，Rz的单位为弧度。
本命令适用于v3.7.2及以上版本。

示例：

```
pose1={495,122,419,3.14,0,-1.57}
pose2={554,122,301,1.57,1,-1.57}
ret=pose_sub(pose1,pose2)
```

2.46 位姿转化为齐次矩阵

```
ret pose_to_matrix(var1)
```

功能：

pose_to_matrix(var1) 用于将位姿转化为齐次矩阵

参数：

var1：

double pose[6]

返回值：

ret：

空：失败

double matrix[4][4]，齐次矩阵：成功

备注：

pose前三位代表位置，X，Y，Z的单位为毫米，后三位代表姿态，Rx，Ry，Rz的单位为弧度。
本命令适用于v3.7.2及以上版本。

示例：

```
pose1={50,0,-30,0.5,1,-0.5}
ret=pose_to_matrix(pose1)
```

2.47 齐次矩阵转化为位姿

```
ret matrix_to_pose(var1)
```

功能：

matrix_to_pose(var1) 用于将齐次矩阵转化为位姿

参数：

var1：

double matrix[4][4]

注：此处是通过二维数组表示的齐次矩阵。

返回值：

ret：

空：失败

double pose[6]：成功

备注：

pose前三位代表位置，X，Y，Z的单位为毫米，后三位代表姿态，Rx，Ry，Rz的单位为弧度。

本命令适用于v3.7.2及以上版本。

示例：

```
matrix1={{1.0,0.0,0.0,100.0},{0.0,0.87758256189037,-0.4794255386042,
200.0},{0.0,0.4794255386042,0.87758256189037,50.0},{0.0,0.0,0.0,1.0}}
ret=matrix_to_pose(matrix1)
```

2.48 位置和旋转矢量转化为齐次矩阵

```
ret pos_rot_vector_to_matrix(var1)
```

功能：

pose_rot_vector_to_matrix(var1) 用于将位置和旋转矢量转化为齐次矩阵

参数：

var1：

double Pos[6]

返回值：

ret：

空：失败

double matrix[4][4]：成功

注：此处是用二维数组表示的齐次矩阵。

备注：

前三位代表位置，X，Y，Z的单位为毫米，后三位代表旋转矢量。

本命令适用于v3.7.2及以上版本。

示例：

```
pos={543.146,116.884,382.408,2.216,-2.216,-0.012}
ret=pos_rot_vector_to_matrix(pos)
```

2.49 齐次矩阵转化为位置和旋转矢量

```
ret matrix_to_pos_rot_vector(var1)
```

功能：

matrix_to_pos_rot_vector(var1) 用于将齐次矩阵转化为位置和旋转矢量

参数：

var1：

double matrix[4][4]：齐次矩阵

返回值：

ret：

空：失败

double PoseAngleAxis[6]：成功

备注：

前三位代表位置，X，Y，Z的单位为毫米，后三位代表旋转矢量。
本命令适用于v3.7.2及以上版本。

示例：

```
matrix1={{0,1,-0.01,-543.147},{-1,0,0,116.884},{0,0.01,1.0,382.408},
{0.0,0.0,0.0,1.0}}
ret=matrix_to_pose(matrix1)
```

2.50 位置和四元数转化为齐次矩阵

```
ret quaternion_to_matrix(var1)
```

功能：

quaternion_to_matrix(var1) 用于获将位置和四元数转化为齐次矩阵

参数：

var1：

double pos[7]

返回值：

ret：

空：失败

double matrix[4][4]：成功

注：此处是用二维数组表示的齐次矩阵。

备注：

前三位代表位置，X，Y，Z的单位为毫米，后四位代表四元数。
本命令适用于v3.7.2及以上版本。

示例：

```
pos={543.146, 116.884, 382.408,0.0004,0.707,-0.707,-0.0037}
ret=quaternion_to_matrix(pos)
```

2.51 齐次矩阵转化为位置和四元数

```
ret matrix_to_quaternion(var1)
```

功能：

matrix_to_quaternion(var1) 用于将齐次矩阵转化为位置和四元数

参数：

var1：

double matrix[4][4]

注：此处是用二维数组表示的齐次矩阵。

返回值：

ret：

空：失败

double pos[7]：成功

备注：

前三位代表位置，X，Y，Z的单位为毫米，后四位代表四元数。
本命令适用于v3.7.2及以上版本。

示例：

```
matrix1={{0,1,-0.01,-543.147},{-1,0,0,116.884},{0,0.01,1.0,382.408},
{0.0,0.0,0.0,1.0}}
ret=matrix_to_quaternion(matrix1)
```

2.52 用户坐标系下的位姿转化为直角坐标系下的位姿

```
ret convert_pose_from_user_to_cart(var1,var2)
```

功能：

convert_pose_from_user_to_cart(var1,var2) 用于将用户坐标系下的位姿转化为直角坐标系下的位姿

参数：

var1：

double pose[6]：用户坐标系下的位姿

var2：

double user_frame[6]：用户坐标系的数据

返回值：

ret：

空：失败

double pose[6]，直角坐标系下的位姿：成功

备注：

位姿前三位代表位置，X，Y，Z的单位为毫米，后三位代表姿态，Rx，Ry，Rz的单位为弧度。
本命令适用于v3.7.2及以上版本。

示例：

```
pose1={495,122,419,3.14,0,-1.57}
user_frame={30,50,30,-1.57,0.1,1}
ret=convert_pose_from_user_to_cart(var1,var2)
```

2.53 直角坐标系下的位姿转化为用户坐标系下的位姿

```
ret convert_pose_from_cart_to_user(var1,var2)
```

功能：

convert_pose_from_cart_to_user(var1,var2) 用于将直角坐标系下的位姿转化为用户坐标系下的位姿

参数：

var1：

double pose[6]：直角坐标系下的位姿

var2：

double user_frame[6]：用户坐标系的数据

返回值：

ret：

空：失败

double pose[6]，用户坐标系下的位姿：成功

备注：

直角坐标系下的位姿，前三位代表位置，X，Y，Z的单位为毫米，后三位代表姿态，Rx，Ry，Rz的单位为弧度。
本命令适用于v3.7.2及以上版本。

示例：

```
pose1={495,122,419,3.14,0,-1.57}
user_frame={30,50,30,-1.57,0.1,1}
ret=convert_pose_from_cart_to_user(var1,var2)
```

2.54 矩阵相乘

```
ret matrix_mul(var1,var2)
```

功能：

matrix_mul(var1,var2) 用于矩阵相乘

参数：

var1：

double matrix[m][n]：二维数组表示的m*n阶矩阵

var2：

double matrix[l][m]：二维数组表示的l*m阶矩阵

注：m，n，l的范围为[2,6]。

返回值：

ret：

空：失败

double matrix[l][n]：成功

备注：

本命令适用于v3.7.2及以上版本。

示例：

```
matrix1={{1,0},{2.5,1},{3.5,-2}}
matrix2={{-1,-1,2,4},{-1,-0.5,-3,1}}
ret=matrix_mul(matrix1,matrix2)
```

2.55 方阵求逆

```
ret matrix_inverse(var1)
```

功能：

matrix_inverse(var1) 用于方阵求逆

参数：

var1：

double matrix[m][m]：二维数组表示的m阶方阵

注：m的范围为[2,6]。

返回值：

ret：

空：失败

double matrix[m][m]：成功

备注：

本命令适用于v3.7.2及以上版本。

示例：

```
matrix1={{1.5,2,-1},{3,-0.5,-2},{-1,1,1}}
ret=matrix_inverse(var1)
```

2.56 计算用户坐标系数据

```
ret calculate_user_frame(var1,var2,var3)
```

功能：

calculate_user_frame(var1,var2,var3) 用于计算用户坐标系数据

参数：

var1：

double pose[6]：标定点工具末端位姿

var2：

double pos[6]：标定点工具末端位姿

var3：

double pos[6]：标定点工具末端位姿

返回值：

ret：

空：失败

double user_frame[6]：成功

备注：

位姿前三位代表位置，X，Y，Z的单位为毫米，后三位代表姿态，Rx，Ry，Rz的单位为弧度。
本命令适用于v3.7.2及以上版本。

示例：

```
pose1={223.707,111.695,769.511,-1.680,-0.109,-1.676}
pose2={110.357,224.367,769.513,-1.680,-0.109,-1.025}
pose3={-114.241,53.414,803.271,-1.279,-0.059,-1.059}
ret=calculate_user_frame(pose1,pose2,pose3)
```

2.57 计算工具TCP的位置

```
ret calculate_tool_position(var1,var2,var3,var4)
```

功能：

calculate_tool_position(var1,var2,var3,var4) 用于计算工具TCP的位置

参数：

var1：

double pose[6]：标定点法兰盘位姿

var2：

double pose[6]：标定点法兰盘位姿

var3：

double pose[6]：标定点法兰盘位姿

var4：

double pose[6]：标定点法兰盘位姿

返回值：

ret：

空：失败

double tool_pos[3]：成功

备注：

位姿前三位代表位置，X，Y，Z的单位为毫米，后三位代表姿态，Rx，Ry，Rz的单位为弧度。
本命令适用于v3.7.2及以上版本。

示例：

```
pose1={530.732,-75.490,648.495,-2.018,-0.941,-1.622}
pose2={384.792,-14.637,700.131,-0.276,-0.385,-2.851}
pose3={351.995,-69.002,677.506,-0.879,-0.024,-3.089}
pose4={396.092,39.977,682.950,0.284,-0.756,-2.852}
ret=calculate_tool_position(pose1,pose2,pose3,pose4)
```

2.58 设置末端指示灯控制模式

```
ret set_ending_pilot_lamp_ctrl_mode
```

功能：

set_ending_pilot_lamp_ctrl_mode用于设置末端指示灯控制模式

参数：

var1：

mode：0：常亮模式，1：自定义模式

返回值：

ret：

-1：失败

0：成功

备注：

本命令适用于V3.11.2及以上版本。

示例：

```
sleep(2)
while 1 do
set_ending_pilot_lamp_ctrl_mode(1)
sleep(2)
ret=get_ending_pilot_lamp_ctrl_mode()
elite_print(ret)
end
```

2.59 获取当前机器人末端指示灯控制模式

```
ret get_ending_pilot_lamp_ctrl_mode
```

功能：

get_ending_pilot_lamp_ctrl_mode 用于获取当前机器人末端指示灯控制模式

参数：

无

返回值：

ret：

0：常亮模式

1：自定义模式

备注：

本命令适用于V3.11.2及以上版本。

示例：

```
ret=get_ending_pilot_lamp_ctrl_mode()
```

2.60 获取自动运行速度百分比

```
get_auto_run_speed
```

功能：

get_auto_run_speed 用于获取自动运行速度的百分比

参数：

无

返回值：

自动运行速度百分比

备注：

本命令适用于v3.15.2及以上版本。

示例：

```
ret=get_auto_run_speed()
```

2.61 设置当前模式下的全局速度万分比

```
set_speed(speed)
```

功能：

设置当前模式下的全局速度万分比

参数：

speed: 全局速度万分比, int类型，范围 [0,10000]

返回值：

-1：失败， 1：成功

示例：

```
ret=set_speed(300)
```

2.62 设置末端扫描按钮模式

```
set_scan_button_mode
```

功能：

设置末端扫描按钮模式

参数：

mode：扫描按钮模式，0：常规模式，1：扫描按钮模式，2：末端摇杆按钮输入模式

返回值：

-1：失败， 0：成功

示例：

```
ret=set_scan_button_mode(0)
```

2.63 获取末端扫描按钮模式

```
get_scan_button_mode
```

功能：

获取末端扫描按钮模式

参数：

无

返回值：

mode：扫描按钮模式，0：常规模式，1：扫描按钮模式，2：末端摇杆按钮输入模式

示例：

```
ret=get_scan_button_mode(1)
```

2.64 映射摇杆按钮到X46~X49

```
deal_end_joystick_input_io
```

功能：

映射摇杆按钮到X46~X49

参数：

value：读取的末端摇杆寄存器1的值，类型：unit16_t

返回值：

-1：失败， 0：成功

示例：

```
ret=deal_end_joystick_input_io(0)
```

第 3 章 TCP 通信

3.1 TCP 服务器

3.1.1 初始化 TCP 服务器

```
init_tcp_server(port)
```

功能：

init_tcp_server(port) 用于初始化 TCP 服务器

参数：

port：

端口号，int类型

返回值：

nil

备注：

控制器已占用22、80、6680、8055、8056、8058、8059、502、1502、5900等端口号，请尽量避免使用。

示例：

```
init_tcp_server(8888)
```

3.1.2 判断客户端是否连接了服务器

```
ret is_client_connected(IP,port)
```

功能：

is_client_connected(IP,port) 用于判断客户端是否连接了服务器

参数：

IP：

客户端 IP 地址，string类型

port：

可选参数，TCP 服务器端口号，int类型

返回值：

ret：

1：参数中的 IP 地址已连接 tcp 服务器

-1：参数中的 IP 地址未连接 tcp 服务器

备注：

port 参数仅适用于 v2.15.2 及以上版本。

示例：

```
ret=is_client_connected("192.168.1.100",8888)
```

3.1.3 接收客户端数据

```
ret,recv server_recv_data(IP,hex,port,recv_timeout)
```

功能：

server_recv_data(IP,hex,port,recv_timeout) 用于接收客户端数据

参数：

IP：

客户端 IP 地址，string类型

hex：

可选参数，指定的服务器发送到客户端的数据格式，0：为字符串，1：为16进制，默认为0，int类型

port：

可选参数，TCP 服务器端口号，int类型

recv_timeout：

可选参数，double类型，超时时间，单位：秒

返回值：

ret：

接收到的数据数量

-1：接收错误

recv：

接收到的数据

备注：

port 参数仅适用于 v2.15.2 及以上版本。

示例：

```
ret,recv=server_recv_data("192.168.1.100",0,888,0.5)
```

3.1.4 向客户端发送数据

```
ret server_send_data(IP,msg,hex,port)
```

功能：

server_send_data(IP,msg,hex,port) 用于向客户端发送数据

参数：

IP：

客户端 IP 地址，string类型

msg：

发送的消息，string类型

hex：

可选参数，int类型，是否为 16 进制数，1 的发送数据为 16 进制数（默认为 0）

port：

可选参数，TCP 服务器端口号，int类型

返回值：

ret：

发送数据数量

-1：发送失败

备注：

port 参数仅适用于 v2.15.2 及以上版本。

示例：

```
ret=server_send_data("192.168.1.100","msg",0,8888)
```

3.1.5 Example

Example：

```
port = 6666
ip ="192.168.1.100"
init_tcp_server(port)
sleep(5)
while(1)do
        ret=is_client_connected(ip)
        if(ret==1)then
                server_send_data(ip,"1")
                recv="1"
                while(recv ~= "2") do
                        sleep(1)
                        Ret,recv=server_recv_data(ip,0,6666,0.5)
                        print(recv,Ret)
                end
        end
end
```

3.2 TCP 客户端

3.2.1 连接服务器

```
ret connect_tcp_server(IP,port)
```

功能：

connect_tcp_server(IP,port) 用于连接指定 IP 和 port 的 TCP 服务器

参数：

IP：

服务器 IP 地址，string类型

port：

端口号，int类型

返回值：

ret：

1：连接成功

0：未连接

备注：

无

示例：

```
ret=connect_tcp_server("192.168.1.100",7777)
```

3.2.2 接收服务器数据

```
ret,recv client_recv_data(IP,recv_timeout,hex,port)
```

功能：

client_recv_data(IP,recv_timeout,hex,port) 用于客户端接收指定 IP 和端口号的服务器发送来的数据

参数：

IP：

服务器ip地址，string类型

recv_time：

接收超时时间,可选参数（该参数不填时，可以通过client_set_recv_timeout设置全局超时时间,否则默认使用4s为超时时间），double类型，单位：秒

hex：

指定的服务器发送到客户端的数据格式，0：为字符串，1：为16进制。可选参数(默认为0), int类型

port：

服务器的端口号（可以通过指定该参数接收不同端口号的数据），可选参数，int类型，端口号

返回值：

ret：

返回 recv 的数量

-1：接收失败需要重新连接

recv：

从服务器发送来的数据

备注：

1：当recv_time参数缺省，后续可选参数必须缺省。

2：当使用多条设置全局超时时间指令和通过接收服务器数据指定超时时间后，如在此之后的接收服务器数据使用缺省超时时间时，此时超时时间为最后指定的超时时间。

port 参数仅适用于 v2.15.2 及以上版本。

本命令适用于v2.9.4 及以上版本。

示例：

```
ret,recv=client_recv_data ("192.168.1.100",0.1,0,7777)
注：recv_timeout参数可不写，而用下文函数设置超时时间。
```

3.2.3 整体设置超时时间

```
ret client_set_recv_timeout(IP,recv_timeout)
```

功能：

client_set_recv_timeout(IP,recv_timeout) 用于整体设置 recv_timeout 时间与上文中的 recv 组合使用

参数：

IP：

IP 地址，srting类型

recv_timeout：

超时时间，double类型，单位：秒

返回值：

ret：

1：设置成功

-1：设置失败

备注：

无

示例：

```
ret=client_set_recv_timeout("192.168.1.100",0.1)
```

3.2.4 清除客户端缓冲区

```
client_flush(string IP, int port)
```

功能：

client_flush(string IP, int port) 用于清除指定TCP客户端缓冲区

参数：

IP：

服务器IP 地址，string类型

port：

服务器端口号（由连接的端口号来指定对应的客户端），可选参数，int类型

返回值：

nil

备注：

无

示例：

```
client_flush("192 .168.1.100" ,7777)
```

3.2.5 向服务器发送数据

```
ret client_send_data(IP,msg,hex,port)
```

功能：

client_send_data(IP,msg,hex,port) 用于客户端向指定 IP 和端口号的服务器发送数据 msg

参数：

IP：

IP 地址，string类型

msg：

向服务器发送的数据，string类型

hex：

是否为 16 进制数,1 发送数据为 16 进制字符格式（默认为 0），int类型

port：

可选参数，端口号，int类型

返回值：

ret：

服务器发送的数据长度

-1：发送失败需要重连

备注：

port 参数仅适用于 v2.15.2 及以上版本。

本命令适用于v2.9.4 及以上版本。

示例：

```
ret=client_send_data("127.0.0.1","OK",0,7777)
```

3.2.6 断开 TCP 连接

```
disconnect_tcp_server(IP,port)
```

功能：

disconnect_tcp_server(IP,port) 用于断开客户端与服务器的连接

参数：

IP：

IP 地址，string类型

port：

可选参数，端口号（不写则默认断开所有 TCP 连接），int类型

返回值：

nil

备注：

port 参数仅适用于 v2.15.2 及以上版本。

示例：

```
disconnect_tcp_server("192.168.1.200",7777)
```

3.2.7 Example

Example：

```
port = 7777
ip = "192.168.1.100"
connect_tcp_server(ip,port)
sleep(1)
client_send_data(ip,"OK")
recv="1"
while(recv ~= "2") do
        sleep(1)
        Ret,recv=client_recv_data(ip,2)
        elite_print(Ret,recv)
end
disconnect_tcp_server(ip)
```

第 4 章 UDP 通信

本章节命令适用于v2.16及以上版本

4.1 UDP 服务器

4.1.1 初始化 UDP 服务器

```
ret init_udp_server(port)
```

功能：

init_udp_server(port) 用于初始化 UDP 服务器

参数：

port：

端口号，int类型

返回值：

ret：

1：成功

-1：失败

备注：

无

示例：

```
ret=init_udp_server(8888)
```

4.1.2 接收 UDP 客户端数据

```
ret,recvbuff,client_ip,client_port udp_server_recv_data(port,timeout,hex
    )
```

功能：

udp_server_recv_data(port,timeout,hex) 用于接收来自指定 IP 和端口号的客户端的数据

参数：

port：

服务器端口号，int类型

timeout：

超时时间，单位：s，double类型

hex：

可选参数，指定的服务器发送到客户端的数据格式，0：为字符串，1：为16进制，默认为0，int类型

返回值：

ret：

大于等于 0：成功

-1：失败

recvbuff：

接收到的数据

client_ip：

源客户端 IP 地址

client_port：

源客户端端口号

备注：

无

示例：

```
ret,recvbuff,client_ip,client_port=udp_server_recv_data(777,0.1
    ,0)
```

4.1.3 向 UDP 客户端发送数据

```
ret udp_server_send_data(server_port,msg,client_port,client_ip,hex)
```

功能：

udp_server_send_data(server_port,msg,client_port,client_ip,hex) 用于服务器向指定端口号和 IP 地址的客户端发送数据

参数：

server_port：

服务器端口号，int类型

msg：

向客户端发送的数据，string类型

client_port：

目标客户端端口号，int类型

client_ip：

目标客户端 IP，string类型

hex：

可选参数，是否为 16 进制数，1 的发送数据为 16 进制数（默认为 0），int类型

返回值：

ret：

大于等于 0：成功

-1：发送失败

备注：

无

示例：

```
ret=udp_server_send_data(777,"test",10000,"192.168.1.100")
```

4.1.4 关闭 udp 服务器

```
ret close_udp_server(port)
```

功能：

close_udp_server(port) 用于关闭 udp 服务器

参数：

port：

可选参数，服务器端口号，不填关闭所有 udp 服务器，int类型

返回值：

ret：

大于等于 0：成功
小于0：发送失败

备注：

无

示例：

```
ret=close_udp_server(777)
```

4.2 UDP 客户端

4.2.1 连接 UDP 服务器

```
ret connect_udp_server(port,IP)
```

功能：

connect_udp_server(port,IP) 用于客户端连接指定端口号和 IP 地址的服务器

参数：

port：

目标服务器端口，int类型

IP：

目标服务器 IP 地址，string类型

返回值：

ret：

大于等于 0：成功

小于 0：未连接

备注：

无

示例：

```
ret=connect_udp_server(777,"192.168.1.100")
```

4.2.2 接收 UDP 服务器数据

```
ret,recv udp_client_recv_data(port,IP,timeout,hex)
```

功能：

udp_client_recv_data(port,IP,timeout,hex) 用于接收来自指定端口号和 IP 地址的服务器的数据

参数：

port：

目标服务器端口，int类型

IP：

目标服务器 IP 地址，string类型

recv_timeout：

超时时间，单位：秒，double类型

hex：

可选参数，指定的服务器发送到客户端的数据格式，0：为字符串，1：为16进制，默认为0，int类型

返回值：

ret：

大于等于 0：成功

小于 0：失败

recv：

接收到的数据

备注：

UDP 采取无连接模式，因此必须为客户端先发送数据，服务器后响应数据。

示例：

```
ret,recv=udp_client_recv_data(777,"192.168.1.6",0.1)
```

4.2.3 向 UDP 服务器发送数据

```
ret udp_client_send_data(port,IP,msg,hex)
```

功能：

udp_client_send_data(port,IP,msg,hex) 用于向指定端口号的服务器发送数据

参数：

port：

目标服务器端口，int类型

IP：

目标服务器 IP 地址，string类型

msg：

向服务器发送的数据，string类型

hex：

可选参数，是否为 16 进制数,1 发送数据为 16 进制数（默认为 0），int类型

返回值：

ret：

大于等于 0：成功

小于 0：失败

备注：

无

示例：

```
ret=udp_client_send_data(777,"192.168.1.100","test")
```

4.2.4 取消连接 udp 服务器

```
ret disconnect_udp_server(port,IP)
```

功能：

disconnect_udp_server(port,IP) 用于断开客户端与服务器的连接

参数：

port：

可选参数，端口号（不写则默认断开所有 UDP 连接），int类型

IP：

可选参数，目标服务器 IP 地址，string类型

返回值：

ret：

大于等于 0：成功

小于 0：失败

备注：

无

示例：

```
ret=disconnect_udp_server(777,"192.168.1.100")
```

第 5 章 485 通信

5.1 打开 485 接口

```
ret rs485_open()
```

功能：

rs485_open 用于打开 485 接口

参数：

无

返回值：

ret：

大于等于 0：打开成功

-1：打开失败

备注：

无

示例：

```
ret=rs485_open()
```

5.2 设置 485 串口配置

```
ret rs485_setopt(speed,bits,event,stop)
```

功能：

rs485_setopt 用于设置 485 串口的配置

参数：

speed：

波特率，int类型

bits：

数据长度 7/8，int类型

event：

奇偶校验“O”，“N”，“E”，int类型

stop：

停止位 1/2，int类型

返回值：

ret：

大于等于 0：设置成功

-1：设置失败

备注：

无

示例：

```
ret=rs485_setopt(9600,8,"N",1)
```

5.3 接收数据

```
ret,recv_buff rs485_recv(time_out,hex,len)
```

功能：

rs485_recv 用于 485 的读操作

参数：

time_out：

超时时间，单位：ms，int类型

hex：

可选参数，是否为 16 进制数，默认为0，int类型
1 表示接收到的数据为 16 进制字符格式，0或其它数值表示接收到的数据非16进制

len：

可选参数，想要获取的长度，在超过 1024 情况下，会自动被设置成 1024，int类型

返回值：

ret：

读到的长度（都是转化为字符长度）

0，-1：读取失败

recv_buff：

获取数据

备注：

无

示例：

```
ret,recv_buff=rs485_recv(100,0,512)
```

5.4 发送数据

```
ret rs485_send(buff,hex)
```

功能：

rs485_send 用于 485 的发送操作

参数：

buff：

需要发送的字符，string类型

hex：

是否为 16 进制数,1 的发送的数据为 16 进制字符格式，int类型

返回值：

ret：

1：发送成功

-1：发送失败

备注：

无

示例：

```
ret=rs485_send("test",0)
```

5.5 关闭 485 接口

```
ret rs485_close()
```

功能：

rs485_close 用于关闭 485 接口

参数：

无

返回值：

ret：

大于等于 0：关闭成功

-1：关闭失败

备注：

无

示例：

```
ret=rs485_close()
```

5.6 清空 485 缓冲区

```
rs485_flush()
```

功能：

rs485_flush 用于清空 485 的缓冲区

参数：

无

返回值：

nil

备注：

无

示例：

```
ret=rs485_flush()
```

5.7 Example

Example:

```
open = rs485_open()
if(open >= 0) then
        set = rs485_setopt(9600,8,"N",1)
        elite_print("set = ", set)
        if(set >= 0) then
                while(1) do
                        repeat
                        ret,recv_buff=rs485_recv(500,0)
                        until(ret~=0)
                elite_print("receive data :",recv_buff)
                rs485_send(recv_buff)
                end
        end
end
rs485_close()
```

第 6 章 232 通信

6.1 打开 232 接口

```
ret rs232_open()
```

功能：

rs232_open() 用于打开 232 接口

参数：

无

返回值：

ret：

大于等于 0：打开成功

-1：打开失败

备注：

无

示例：

```
ret=rs232_open()
```

6.2 设置 232 串口配置

```
ret rs232_setopt(speed,bits,event,stop)
```

功能：

rs232_setopt(speed,bits,event,stop) 用于设置 232 串口的配置

参数：

speed：

波特率，int类型

bits：

数据长度 7/8，int类型

event：

奇偶校验“O”，“N”，“E”，string类型

stop：

停止位 1/2，int类型

返回值：

ret：

大于等于 0：设置成功

-1：设置失败

备注：

无

示例：

```
ret=rs232_setopt(9600,8,"N",1)
```

6.3 接收数据

```
ret,recv_buff rs232_recv(time_out,hex,len)
```

功能：

rs232_recv(time_out,hex,len) 用于 232 的读操作

参数：

time_out：

超时时间，单位：ms，int类型

hex：

是否为 16 进制数,1 的接收到的数据为 16 进制字符格式（默认为 0），int类型

len：

可选参数，想要获取的长度，在超过 1024 情况下，会自动被设置成 1024，int类型

返回值：

ret：

读到的长度（都是转化为字符长度）

0，-1：读取失败

recv_buff：

获取数据

备注：

无

示例：

```
ret,recv_buff=rs232_recv(100,0,512)
```

6.4 发送数据

```
ret rs232_send(buff,hex)
```

功能：

rs232_send(buff,hex) 用于 232 的发送操作

参数：

buff：

需要发送的字符，string类型

hex：

是否为 16 进制数,1 的发送的数据为 16 进制字符格式，int类型

返回值：

ret：

1：发送成功

-1：发送失败

备注：

无

示例：

```
ret=rs232_send("test",0)
```

6.5 关闭 232 接口

```
ret rs232_close()
```

功能：

rs232_close() 用于关闭 232 接口

参数：

无

返回值：

ret：

大于等于 0：关闭成功

-1：关闭失败

备注：

无

示例：

```
ret=rs232_close()
```

上述命令适用于2.12.0及以上版本!

6.6 清空 232 缓冲区

```
rs232_flush()
```

功能：

rs232_flush() 用于清空 232 的缓冲区

参数：

无

返回值：

nil

备注：

本命令适用于v2.16.2 及以上版本。

示例：

```
ret=rs232_flush()
```

6.7 Example

```
open = rs232_open()
if(open >= 0) then
        set = rs232_setopt(9600,8,"N",1)
        elite_print("set = ", set)
        if(set >= 0) then
                while(1) do
                        repeat
                        ret,recv_buff=rs232_recv(500,0)
                                until(ret~=0)
                                elite_print("receive data :",recv_buff)

                                rs232_send(recv_buff)
                end
        end
end
rs232_close()
```

第 7 章 TCI 通信

7.1 打开末端 485 接口

```
ret tci_open()
```

功能：

tci_open() 用于打开末端 485 接口

参数：

无

返回值：

ret：

大于等于 0：打开成功

-1：打开失败

备注：

无

示例：

```
ret=tci_open()
```

7.2 设置 TCI 串口的配置

```
ret tci_setopt(speed,bits,event,stop)
```

功能：

tci_setopt(speed,bits,event,stop) 用于设置 TCI 串口的配置

参数：

speed：

波特率，int类型

bits：

数据长度 8，int类型

event：

奇偶校验“O”，“N”，“E”，string类型

stop：

停止位 1/2，int类型

返回值：

ret：

大于等于 0：设置成功

-1：设置失败

备注：

无

示例：

```
ret=tci_setopt(9600,8,"N",1)
```

7.3 接收数据

```
ret,recv_buff tci_recv(time_out,hex,len)
```

功能：

tci_recv(time_out,hex,len) 用于 TCI 的读操作

参数：

time_out：

超时时间，单位：ms，int类型

hex：

可选参数，是否为 16 进制数，默认为0，int类型
1 表示接收到的数据为 16 进制字符格式，0或其它数值表示接收到的数据非16进制

len：

可选参数，想要获取的长度，在超过 1024 情况下，会自动被设置成 1024，int类型

返回值：

ret：

读到的长度（都是转化为字符长度）

0，-1：读取失败

recv_buff：

获取数据

备注：

无

示例：

```
ret,recv_buff=tci_recv(100,0,512)
```

7.4 发送数据

```
ret tci_send(buff,hex)
```

功能：

tci_send(buff,hex) 用于 TCI 的发送操作

参数：

buff：

需要发送的字符，string类型

hex：

是否为 16 进制数,1 的发送的数据为 16 进制字符格式，int类型

返回值：

ret：

1：发送成功

-1：发送失败

备注：

无

示例：

```
ret=tci_send("test",0)
```

7.5 关闭 TCI 接口

```
ret tci_close()
```

功能：

tci_close() 用于关闭 TCI 接口

参数：

无

返回值：

ret：

大于等于 0：关闭成功

-1：关闭失败

备注：

无

示例：

```
ret=tci_close()
```

7.6 清空 TCI 缓冲区

```
tci_flush()
```

功能：

tci_flush() 用于清空 TCI 的缓冲区

参数：

无

返回值：

nil

备注：

无

示例：

```
ret=tci_flush()
```

7.7 Example

Example:

```
sleep(5)
local open = tci_open()
if (open >= 0) then
        local set = tci_setopt(9600,8,"N",1)
        if (set >= 0) then
                sleep(1)
                tci_send("Testing TCI (testing firmware: 20190826)")
                while (1) do
                        ret,recv_buff=tci_recv(500,0)
                        sleep(1)
                        if(ret>0) then
                                elite_print(recv_buff)
                                tci_send(recv_buff)
                        end
                end
        else
                elite_print("set tci failed.")
        end
else
        elite_print("open tci failed.")
end
tci_close()
```

第 8 章 公版扩展

HTTP 参考：http://w3.impa.br/ diego/software/luasocket/http.html

Xml 解析参考：第 11.1 节 附录 1：lua 脚本

第 11.2 节 附录 2：xml

JSON 解析参考：第 11.3 节 附录 3：lua 脚本

Xmlrpc 客户端参考：第 11.4 节 附录 4

第 9 章 MODBUS MASTER

9.1 获取 modbusrtu 句柄

```
ctx modbus_new_rtu(choose,baud,parity,data_bit,stop_bit)
```

功能：

modbus_new_rtu(choose,baud,parity,data_bit,stop_bit) 用于获取 modbusrtu 的句柄

参数：

choose：

0/2, 0 为主板 485 口，2为TCI 485 口， int类型

baud：

4800，9600... 可选参数，不写默认为 4800，最高支持115200，int类型

parity：

‘E’，‘N’，‘O’的 ASCII 码值 69,78,79，可选参数，不写默认为 78，int类型

data_bit：

8，可选参数，不写默认 8，int类型

stop_bit：

1/2，可选参数，不写默认 1，int类型

返回值：

ctx：

modbus 句柄

备注：

本命令适用于2.9.3及以上版本。

示例：

```
ctx=modbus_new_rtu(2,9600,78,8,1)
```

9.2 获取 modbustcp 句柄

```
ctx modbus_new_tcp(IP,port)
```

功能：

modbus_new_tcp(IP,port) 用于获取 modbustcp 的句柄

参数：

IP：

modbusslave 的 IP 地址，string类型

port：

modbusslave 的端口号，int类型

返回值：

ctx：

modbus 句柄

备注：

本命令适用于2.9.3及以上版本。

示例：

```
ctx=modbus_new_tcp("192.168.1.15",502)
```

9.3 连接 modbus 句柄

```
ret modbus_connect(ctx)
```

功能：

modbus_connect(ctx) 用于连接 modbustcp 的句柄

参数：

ctx:：

前面 modbus_new_rtu, modbus_new_tcp 创建的句柄

返回值：

ret：

-1：连接失败

其他：连接正常

备注：

返回值不作为连接 rtu 句柄连接成功与否的标志

本命令适用于2.9.3及以上版本。

示例：

```
ret=modbus_connect(ctx)
```

9.4 关闭 modbus 句柄

```
modbus_close(ctx)
```

功能：

modbus_close(ctx) 用于关闭 modbustcp 的句柄

参数：

ctx:：

前面 modbus_new_rtu, modbus_new_tcp 创建的句柄

返回值：

nil

备注：

本命令适用于2.9.3及以上版本。

示例：

```
modbus_close(ctx)
```

9.5 设置 slave 地址

```
ret modbus_set_slave(ctx,slave_id)
```

功能：

modbus_set_slave(ctx,slave_id) 用于设置 slave 地址

参数：

ctx:：

前面 modbus_new_rtu, modbus_new_tcp 创建的句柄

slave_id：

slave 地址，int类型

返回值：

ret：

-1：出错

其他：正常

备注：

本命令适用于2.9.3及以上版本。

示例：

```
ret=modbus_set_slave(ctx,0x2)
```

9.6 modbus 往指定寄存器地址写入值

```
ret modbus_write_register(ctx,reg,value)
```

功能：

modbus_write_register(ctx,reg,value) 用于 modbus 往指定寄存器地址写入值

参数：

ctx:：

前面 modbus_new_rtu, modbus_new_tcp 创建的句柄

reg：

寄存器地址（为十进制），int类型

value：

值，int类型

返回值：

ret：

-1：出错

其他：正常

备注：

本命令适用于2.9.3及以上版本。

示例：

```
ret=modbus_write_register(ctx, 771, 1)
```

9.7 modbus 从指定寄存器读取信号值

```
ret,value modbus_read_register(ctx,reg)
```

功能：

modbus_read_register(ctx,reg) 用于 modbus 从指定寄存器读取信号值

参数：

ctx:：

前面 modbus_new_rtu, modbus_new_tcp 创建的句柄

reg：

寄存器地址（为十进制），int类型

返回值：

ret：

读写成功与否，-1为失败，

value：

返回读到的值

备注：

本命令适用于2.9.3及以上版本。

示例：

```
ret,value=modbus_read_register(ctx,771)
```

9.8 modbus 从指定线圈读取信号值

```
ret modbus_read_bits(ctx,reg,len)
```

功能：

modbus_read_bits(ctx,reg,len) 用于 modbus 从指定线圈读取信号值

参数：

ctx:：

前面 modbus_new_rtu, modbus_new_tcp 创建的句柄

reg：

线圈地址，int类型

len：

线圈个数 (<128)，int类型

返回值：

ret：

table

备注：

本命令适用于2.12.0及以上版本。

示例：

```
ret=modbus_read_bits(ctx, 771,1)
```

9.9 modbus 往指定多个线圈写入值

```
ret modbus_write_bits(ctx,reg,size,value)
```

功能：

modbus_write_bits(ctx,reg,size,value) 用于 modbus 往指定多个线圈写入值

参数：

ctx:：

前面 modbus_new_rtu, modbus_new_tcp 创建的句柄

reg：

线圈地址，int类型

size：

数量，int类型

value：

写入线圈的数据，table类型，table中的数据为0或1

返回值：

ret：

-1：出错

其他：正常

备注：

Modbus 不能对 M0-M527，M1472-M1536 进行写操作。

本命令适用于v2.12.0及以上版本。

示例：

```
value_array = {1,1,1}
ret=modbus_write_bits(ctx,1,3,value_array)
```

9.10 modbus 往指定线圈写入值

```
ret modbus_write_bit(ctx,reg,value)
```

功能：

modbus_write_bit(ctx,reg,value) 用于 modbus 往指定线圈写入值

参数：

ctx:：

前面 modbus_new_rtu, modbus_new_tcp 创建的句柄

reg：

线圈地址，int类型

value：

写入线圈的数据 1 或 0，int类型

返回值：

ret：

-1：出错

0，1：正常

备注：

Modbus 不能对 M0-M527，M1472-M1536 进行写操作。

本命令适用于v2.12.0及以上版本。

示例：

```
ret=modbus_write_bit(ctx,1,1)
```

9.11 modbus 读 input 寄存器值

```
ret,reg modbus_read_input_register(ctx,addr)
```

功能：

modbus_read_input_register(ctx,addr) 用于 modbus 读 input 寄存器值

参数：

ctx:：

前面 modbus_new_rtu, modbus_new_tcp 创建的句柄

addr：

input register 地址，int类型

返回值：

ret：

-1：出错

0，1：正常

reg：

返回的寄存器值

备注：

无

示例：

```
ret,reg=modbus_read_input_register(ctx, 1)
```

9.12 modbus 往指定多个寄存器地址写入值

```
ret modbus_write_registers(ctx,reg,size,value)
```

功能：

modbus_write_registers(ctx,reg,size,value) 用于 modbus 往指定寄存器地址写入值

参数：

ctx:：

前面 modbus_new_rtu, modbus_new_tcp 创建的句柄

reg：

寄存器地址，int类型

size：

数量，int类型

value：

写入寄存器的数据，table类型

返回值：

ret：

-1：出错

其他：正常

备注：

无

示例：

```
ret=modbus_write_registers(ctx,1,3,{0x3444,0x3333,0x4444})
```

9.13 读取多个寄存器的值

```
ret modbus_read_registers(ctx,addr,len)
```

功能：

modbus_read_registers(ctx,addr,len) 用于读取多个寄存器的值

参数：

ctx：

创建的句柄

addr：

寄存器地址，int类型

len：

寄存器个数 [1,125]，int类型

返回值：

ret：

空：失败

寄存器值列表：成功

备注：

无

示例：

```
sleep(5)
ctx=modbus_new_rtu(2,9600,78,8,1)
sleep(1)
modbus_set_slave(ctx,0x1)
table_c=modbus_read_registerS(ctx,1,12)
sleep(1)
i=1
for i,v in pairs(table_c) do
    elite_print(v)
end
```

9.14 获取输入线圈状态

```
ret modbus_read_input_bits(ctx,reg,len)
```

功能：

modbus_read_input_bits(ctx,reg,len) 用于获取单个或多个输入线圈的状态

参数：

ctx：

前面 modbus_new_rtu，modbus_new_tcp 创建的句柄

reg：

线圈地址，int类型

len：

线圈个数 (<=128)，int类型

返回值：

ret：

单个或多个输入线圈的状态

备注：

无

示例：

```
ret=modbus_read_input_bits(ctx, 0, 1)
```

9.15 Example

9.15.1 RTU Example

```
sleep(5)
ctx=modbus_new_rtu(0,9600,78,8,1)
modbus_connect(ctx)
ret1=modbus_set_slave(ctx,0x3)
if(ret1==-1)then
        elite_print("Wrong address")
end
ret2=modbus_write_register(ctx,771,1)
if(ret2==-1)then
        elite_print("Write error")
end
ret3,value=modbus_read_register(ctx,771)
if(ret3==-1)then
        elite_print("Read error")
end
elite_print("value is:",value)
modbus_close(ctx)
```

9.15.2 TCP Example

```
sleep(5)
ip="192.168.1.7"
port=502
ctx=modbus_new_tcp(ip,port)
repeat
        ret=modbus_connect(ctx)
until(ret~=-1)
ret2=modbus_write_register(ctx,771,1)
if(ret2==-1)then
        elite_print("Write error")
end
ret3,value=modbus_read_register(ctx,771)
if(ret3==-1)then
        elite_print("Read error")
```

```
15 end
16 elite_print("value is:",value)
17 modbus_close(ctx)
```

第 10 章 Profinet

10.1 获取 int 型输入寄存器的值

```
ret get_profinet_int_input_registers(addr,size)
```

功能：

get_profinet_int_input_registers(addr,size) 用于获取 profinet int 型输入寄存器的值

参数：

addr：

寄存器起始地址，范围：[0,31]，int类型

size：

寄存器个数，范围：[1,32]，int类型

注：addr 与 size 的和应小于等于 32

返回值：

ret：

空：失败

寄存器值列表：成功

备注：

无

示例：

```
ret=get_profinet_int_input_registers(0,2)
```

10.2 获取 int 型输出寄存器的值

```
ret get_profinet_int_output_registers(addr,size)
```

功能：

get_profinet_int_output_registers(addr,size) 用于获取 profinet int 型输出寄存器的值

参数：

addr：

寄存器起始地址，范围：[0,31]，int类型

size：

寄存器个数，范围：[1,32]，int类型

注：addr 与 size 的和应小于等于 32

返回值：

ret：

空：失败

寄存器值列表：成功

备注：

无

示例：

```
ret=get_profinet_int_output_registers(0,2)
```

10.3 获取 float 型输入寄存器的值

```
ret get_profinet_float_input_registers(addr,size)
```

功能：

get_profinet_float_input_registers(addr,size) 用于获取 profinet float 型输入寄存器的值

参数：

addr：

寄存器起始地址，范围：[0,31]，int类型

size：

寄存器个数，范围：[1,32]，int类型

注：addr 与 size 的和应小于等于 32

返回值：

ret：

空：失败

寄存器值列表：成功

备注：

无

示例：

```
ret=get_profinet_float_input_registers(0,2)
```

10.4 获取 float 型输出寄存器的值

```
ret get_profinet_float_output_registers(addr,size)
```

功能：

get_profinet_float_output_registers(addr,size) 用于获取 profinet float 型输出寄存器的值

参数：

addr：

寄存器起始地址，范围：[0,31]，int类型

size：

寄存器个数，范围：[1,32]，int类型

注：addr 与 size 的和应小于等于 32

返回值：

ret：

空：失败

寄存器值列表：成功

备注：

无

示例：

```
ret=get_profinet_float_output_registers(0,2)
```

10.5 设置 int 型输出寄存器的值

```
ret set_profinet_int_output_registers(addr,size,value)
```

功能：

set_profinet_int_output_registers(addr,size,value) 用于设置 profinet int 型输出寄存器的值

参数：

addr：

寄存器起始地址，范围：[0,31]，int类型

size：

寄存器个数，范围：[1,32]，int类型

注：addr 与 size 的和应小于等于 32

value：

寄存器值列表，table类型，列表元素类型，int[-2147483648,2147483647]

返回值：

ret：

-1：失败

1：成功

备注：

无

示例：

```
ret=set_profinet_int_output_registers(0,2,{123,-123})
```

10.6 设置 float 型输出寄存器的值

```
ret set_profinet_float_output_registers(addr,size,value)
```

功能：

set_profinet_float_output_registers(addr,size,value) 用于设置 profinet float 型输出寄存器的值

参数：

addr：

寄存器起始地址，范围：[0,31]，int类型

size：

寄存器个数，范围：[1,32]，int类型

注：addr 与 size 的和应小于等于 32

value：

寄存器值列表，table类型，列表元素类型，number[-3.40E+38,3.40E+38]

返回值：

ret：

-1：失败

1：成功

备注：

无

示例：

```
ret=set_profinet_float_output_registers(0,2,{2.33,-2.33})
```

第 11 章 Ethernet/IP

11.1 获取 int 型输入寄存器的值

```
ret get_eip_int_input_registers(addr,size)
```

功能：

get_eip_int_input_registers(addr, size) 用于获取Ethernet/IP int型输入寄存器的值

参数：

addr：寄存器起始地址，范围为[0,31]
size：寄存器个数，范围为[1,32]

返回值：

ret：
空：失败
寄存器值列表：成功

备注：

addr与size的和应小于等于32，本命令适用于v3.5.2及以上版本。

示例：

```
ret=get_eip_int_input_registers(0,2)
```

11.2 获取 int 型输出寄存器的值

```
ret get_eip_int_output_registers(addr,size)
```

功能：

get_eip_int_output_registers(addr, size) 用于获取Ethernet/IP int型输出寄存器的值

参数：

addr：寄存器起始地址，范围为[0,31]
size：寄存器个数，范围为[1,32]

返回值：

ret：
空：失败
寄存器值列表：成功

备注：

addr与size的和应小于等于32，本命令适用于v3.5.2及以上版本。

示例：

```
ret=get_eip_int_output_registers(0,2)
```

11.3 获取 float 型输入寄存器的值

```
ret get_eip_float_input_registers(addr,size)
```

功能：

get_eip_float_input_registers(addr, size) 用于获取Ethernet/IP float型输入寄存器的值

参数：

addr：寄存器起始地址，范围为[0,31]

size：寄存器个数，范围为[1,32]

返回值：

ret：

空：失败

寄存器值列表：成功

备注：

addr与size的和应小于等于32，本命令适用于v3.5.2及以上版本。

示例：

```
ret=get_eip_float_input_registers(0,2)
```

11.4 获取 float 型输出寄存器的值

```
ret get_eip_float_output_registers(addr,size)
```

功能：

get_eip_float_output_registers(addr, size) 用于获取Ethernet/IP float型输出寄存器的值

参数：

addr：寄存器起始地址，范围为[0,31]

size：寄存器个数，范围为[1,32]

返回值：

ret：

空：失败

寄存器值列表：成功

备注：

addr与size的和应小于等于32，本命令适用于v3.5.2及以上版本。

示例：

```
ret=get_eip_float_output_registers(0,2)
```

11.5 设置 int 型输出寄存器的值

```
ret set_eip_int_output_registers(addr,size,value)
```

功能：

set_eip_int_output_registers(addr, size, value) 用于设置Ethernet/IP int型输出寄存器的值

参数：

addr：寄存器起始地址，范围为[0,31]
size：寄存器个数，范围为[1,32]
value：寄存器值列表，列表元素类型，int[-2147483648,2147483647]

返回值：

ret：
-1：失败
1：成功

备注：

addr与size的和应小于等于32，本命令适用于v3.5.2及以上版本。

示例：

```
ret=set_eip_int_output_registers(0,2,{123, -123})
```

11.6 设置 float 型输出寄存器的值

```
ret set_eip_float_output_registers(addr,size,value)
```

功能：

set_eip_float_output_registers(addr, size, value) 用于设置Ethernet/IP float型输出寄存器的值

参数：

addr：寄存器起始地址，范围为[0,31]
size：寄存器个数，范围为[1,32]
value：寄存器值列表，列表元素类型，number[-3.40E+38,3.40E+38]

返回值：

ret：
-1：失败
1：成功

备注：

addr与size的和应小于等于32，本命令适用于v3.5.2及以上版本。

示例：

```
ret=set_eip_float_output_registers(0,2,{2.33, -2.33})
```

第 12 章 外部力传感器

12.1 标记力矩数据传输开始

```
ret start_push_force()
```

功能：

start_push_force() 用于标记力矩数据传输开始

参数：

无

返回值：

ret：

0：成功
-1：失败

备注：

本命令适用于v3.5.2及以上版本。

示例：

```
ret=start_push_force()
```

12.2 传递力矩数据

```
ret push_external_force(index,torque_arry)
```

功能：

push_external_force(index, torque_arry) 用于传递力矩数据

参数：

index：

序列号，指示传输的顺序，int型，范围为int[0,65535]

*参数index的值应在一次完整的传输过程开始后，从0开始逐一向上递增，达到最大值65535后，从0开始计数，循环往复，通过这种方式标记torque_arry是最新的数据。

torque_arry*：

力矩数据数组，double torques[6]

*参数torque_arry的定义具体如下：数据名称包含X方向力、Y方向力、Z方向力、X轴扭矩、Y轴力矩、Z轴力矩，当数据类型为double时，则依次表示的是力传感器输出坐标系下的X方向力、Y方向力、Z方向力、X轴力矩、Y轴力矩和Z轴力矩。方向力的单位为kg，扭矩的单位为kgM。若原始数据的单位与坐标系和定义不一致，请对参数进行转换再传入。

返回值：

ret：

0：成功
-1：失败

备注：

使用前需要先调用start_push_force标记外部力矩数据开始传递。本命令适用于v3.5.2及以上版本。

示例：

```
ret=start_push_force()
index=0
if(ret==0)then
    while(true)do
        ret=push_external_force(index,{0,1,2,3,4,5})
        index=index+1
        if(index>=65535)then
            index=0
```

```
        end
    end
end
```

12.3 结束当前力矩数据的传递

```
ret stop_push_force()
```

功能：

stop_push_force() 用于结束当前力矩数据的传递

参数：

无

返回值：

ret：

0：成功
-1：失败

备注：

本命令适用于v3.5.2及以上版本。

示例：

```
ret=stop_push_force()
```

12.4 获取当前力矩数据的来源

```
ret get_force_ctrl_mode()
```

功能：

get_force_crrl_mode() 用于获取当前力矩数据的来源

参数：

无

返回值：

ret：

int[0,4]，0和1表示力矩数据来自内部，2表示力矩数据来自SDK，3表示力矩数据来自LUA，4表示力矩数据来自末端

备注：

本命令适用于v3.5.2及以上版本。

示例：

```
ret=get_force_ctrl_mode()
```

12.5 获取基于外部力传感器TCP力及力矩信息

```
ret get_tcp_force_by_sensor()
```

功能：

get_tcp_force_by_sensor() 用于获取基于外部力传感器TCP力及力矩信息

参数：

无

返回值：

ret：

失败：空

成功：double force[6]，返回TCP力及力矩信息，前三位代表力，后三位代表力矩

备注：

本命令仅支持在外部力传感器处于连接状态时使用。本命令适用于v3.9.2及以上版本。

示例：

```
ret=get_tcp_force_by_sensor()
```

12.6 Example

Example：

```
function hex2float(hexString)
  if hexString == nil then
      return 0
  end
  local t = type(hexString)
  if t == "string" then
      hexString = tonumber(hexString,16)
  end

  local hexNums = hexString

  local sign = math.modf(hexNums/(2^31))

  local exponent = hexNums%(2^31)
  exponent = math.modf(exponent/(2^23)) -127

  local mantissa = hexNums%(2^23)

  for i = 1,23 do
      mantissa = mantissa/2
  end
  mantissa = 1+mantissa
  local result = (-1)^sign * mantissa * 2^exponent
  return result
end

function str2hex(str)
  -- 判断输入类型
  if (type(str) ~= "string") then
      return nil,"str2hex invalid input type"
  end
  -- 滤掉分隔符
  str = str:gsub("[%s%p]",""):upper()
  -- 检查内容是否合法
  if (str:find("[^0-9A-Fa-f]") ~= nil) then
      return nil,"str2hex invalid input content"
  end
  -- 检查字符串长度
  if (str:len()%2 ~= 0) then
      return nil,"str2hex invalid input length"
  end
```

```
  -- 拼接字符串
  local index = 1
  local ret = ""
  for index = 1,str:len(),2 do
      ret = ret..string.char(tonumber(str:sub(index,index+1),16))
  end

  return ret
end

function hex2str(hex)
  -- 判断输入类型
  if (type(hex) ~= "string") then
      return nil,"hex2str invalid input type"
  end
  -- 拼接字符串
  local index = 1
  local ret = ""
  for index = 1,hex:len() do
      ret = ret..string.format("%02X",hex:sub(index):byte())
  end

  return ret
end

function convert(hex )
  -- body
  hex = string.reverse(hex)
  local hex_str = hex2str(hex)
  return hex2float(hex_str)
end
-- rs485 parameter
spd = 460800
bits = 8
event = "N"
stop = 1
set_global_variable("B4",0)
sleep(1)
open = rs485_open()

if open >= 0 then
    local set = rs485_setopt(spd,bits,event,stop)
    sleep(1)
    -- 开始前清除传输标记
    stop_push_force()
```

```
87
88      -- 控制传感器持续数据发送
89      repeat
90          rs485_flush()
91          local lenth = rs485_send("48AA0D0A",1)
92          ret,recv_buff = rs485_recv(2,1)
93      until(ret ~= 0)
94
95      -- 设置传输开始标志
96      elite_print("Transition start.")
97      ret = start_push_force()
98      if (ret ~= 0) then
99          elite_print("Transition not ready.")
100         exit()
101     end
102
103     index = 0
104     err_cnt = 0
105     tm_last, tm_tmp = 0, 0
106     last_call_tm, curr_call_tm = 0, 0
107     torque = {0, 0, 0, 0, 0, 0}
108     F1 = 0
109     F2 = 0
110     F3 = 0
111     F4 = 0
112     F5 = 0
113     F6 = 0
114     repeat
115         rs485_flush()
116         ret_485, recv_buff = rs485_recv(2,1)
117         begin_idx, end_idx = 0,0
118         begin_idx, end_idx = string.find(recv_buff, '48AA.*0D0A', 1, false)
119         sleep(0.001)
120         if (begin_idx and end_idx and ((end_idx - begin_idx) >= 55)) then
121             end_idx_tmp, end_idx = string.find(recv_buff, '0D0A', begin_idx)
122             if end_idx - begin_idx == 55 then
123                 valid_data = string.sub(recv_buff, begin_idx, end_idx)
124                 local hex = str2hex(valid_data)
125                 local f1 = string.sub(hex,3,7)
126                 local f2 = string.sub(hex,7,11)
127                 local f3 = string.sub(hex,11,15)
128                 local f4 = string.sub(hex,15,19)
129                 local f5 = string.sub(hex,19,23)
130                 local f6 = string.sub(hex,23,27)
```

```
                F1 = convert(f1)
                F2 = convert(f2)
                F3 = convert(f3)
                F4 = convert(f4)
                F5 = convert(f5)
                F6 = convert(f6)
                torque = {F1,F2,F3,F4,F5,F6}
                err_cnt = 0
            end
        else
            err_cnt = err_cnt + 1
            -- 多次没有接收到正确的数据则认为是通讯断开或者其它问题
            if err_cnt > 100 then
                elite_print("Failed to receive data from sensor.")
            end
        end

        tm_tmp = os.clock()
        once_elapse = tm_tmp - tm_last
        tm_last = tm_tmp
        set_global_variable("D7",once_elapse)
        set_global_variable("D8",last_call_tm)
        set_global_variable("D9",curr_call_tm)
        ret = push_external_force(index, torque)
        if (ret == -1) then
            curr_call_tm = os.clock() - tm_last
            elite_print("ret = ",ret, "push_external_force failed.")
            break
        end

        last_call_tm = os.clock() - tm_last
        set_global_variable("D1",F1)
        set_global_variable("D2",F2)
        set_global_variable("D3",F3)
        set_global_variable("D4",F4)
        set_global_variable("D5",F5)
        set_global_variable("D6",F6)

        if index < 65535 then
            index = index+1
        else
            index = 1
        end

```

```
    until(false)

else
    elite_print("open dead")
end
stop_push_force()
rs485_close()
```

第 13 章 附录

13.1 附录 1

```
xml = require('LuaXml')
-- load XML data from file "test.xml" into local table
local xfile = xml.load("test.xml")
-- search for substatement having the tag "scene" local
xscene = xfile:find("scene")
if xscene ~= nil then
        print(xscene)
-- print tag, attribute id and first
substatementprint( xscene:tag(), xscene.id, xscene[1] )
end
xfile:save"t.xml" print("---\nREADY.")
```

13.2 附录 2

```
<?xml version="1.0" encoding="iso-8859-1"?>
<XperiML version="2.0">
<!-- application info -->
<applicationInfo>
        <string id="version">0.5.0</string>
        <string id="date">2004-11-23 - 2006-08-18 </string>
        <string id="shortDescr"> a demonstration application for LuaXML
            scripting </string>
        <string id="usage">[ -i(ini.xml) ]</string>
</applicationInfo>
<!-- initialization of the window -->
<deviceWindow id="0" name="window" deviceContainer="input">
        <string id="winTitle"> my window </string>
        <float id="mouseRelative"> 0 </float>
        <float id="mouseNeutral"> 0.0 </float>
        <bool id="mouseVisible"> 0 </bool>
        <bool id="fullScreen"> 0 </bool>
```

```
17          <int id="zBufferBits"> 24 </int>
18          <int id="stencilBufferBits"> 0 </int>
19          <int id="winSizeX"> 800 </int>
20          <int id="winSizeY"> 600 </int>
21          <floatArray id="bgColor"> 0.5 0.6 0.7 </floatArray>
22          <!-- 2d visualization / overlay initialization -->
23          <overlay>
24                  <floatArray id="bgNormalColor"> 1.0 1.0 1.0 0.2 </
                       floatArray>
25                  <floatArray id="fgNormalColor"> 1.0 0.0 1.0 0.7 </
                       floatArray>
26                  <floatArray id="bgSelectColor"> 0.0 0.5 1.0 0.7 </
                       floatArray>
27                  <floatArray id="fgSelectColor"> 1.0 1.0 1.0 1.0 </
                       floatArray>
28                  <!-- overlay plane minimum and maximum coordinates -->
29                  <float id="minX"> -1.0 </float>
30                  <float id="minY"> -1.0 </float>
31                  <float id="maxX"> 1.0 </float>
32                  <float id="maxY"> 1.0 </float>
33          </overlay>
34          <!-- X  Y  Z  H  P  R (output axes) -->
35          <axesInputMapping> 0 1 2 3 4 5 </axesInputMapping>
36          <axesInputScale> 1 1 1 1 1 1 </axesInputScale>
37          <axesInputShift> 0 0 0 0 0 0 </axesInputShift>
38  </deviceWindow>
39  <deviceGraphics d="0">
40          <float id="nearClipping"> 0.1 </float>
41          <float id="farClipping"> 2000 </float>
42          <float id="frustumLeft"> -1.0 </float>
43          <float id="frustumRight"> 1.0 </float>
44          <float id="frustumBottom"> -.75 </float>
45          <float id="frustumTop"> .75 </float>
46          <bool id="backFaceCulling"> 1 </bool>
47          <!-- camera initialization -->
48          <camera object="observer" pos="0 -3 1.6 0 0 0"/>
49  </deviceGraphics>
50  <!-- audio device initialization -->
51  <deviceAudio id="0" class="deviceAudioAL">
```

```
52          <!-- specific global settings for OpenAL -->
53          <int id="distanceModel"> 1 </int>
54          <!-- valid arguments are NONE=0, INVERSE\_DISTANCE=1 (default),
                INVERSE\_DISTANCE\_CLAMPED=2, see OpenAL ref. man. p.21 -->
55          <float id="dopplerVelocity"> 330.0 </float>
56          <!-- corresponds to sonic speed for doppler effect calculations
                -->
57          <float id="dopplerFactor"> 1.0 </float>
58          <!-- additional scaling factor for doppler effect calculations
               -->
59  </deviceAudio>
60  <!-- internal script definitions -->
61  <scripts>
62          <script name="main" mime="application/x-lua">
63                  <![CDATA[
64                  -- initialization
65                  if nFrames==nil then
66                          nFrames=0;  tLastFrames=ve.now();
67                  end
68                  -- termination
69                  if obj.getFlag(INPUT,27) then
70                          ve.exit();
71                  end
72                  -- update info line:
73                  if ve.now()>=tLastFrames+1.0 then
74                          tLastFrames=ve.now()
75                          ovl.text(1,-1,ALIGN\_RIGHT,ALIGN\_BOTTOM,
                               nFrames," fps");
76                          nFrames=0;
77                  end
78                  nFrames=nFrames+1;
79                  ]]>
80          </script>
81  </scripts>
82  <!-- resource definition -->
83  <resources>
84          <resource mime="model/vrml" id="1" name="virtualab"
85          url="resources/virtualab.wrl"/>
86          <resource mime="model/vrml" id="2" name="surface"
```

```
87          url="resources/virtualab\_surface.wrl"/>
88          <resource mime="model/vrml" id="3" name="cube"
89          url="resources/box.wrl"/>
90          <resource mime="model/vrml" id="5" name="ball"
91          rl="resources/ball01.wrl"/>
92          <resource mime="image/png" id="6" name="veRner"
93          url="resources/veRner\_small.png" sizeX="6" sizeY="3"/>
94          <resource mime="image/png" id="7" name="explo"
95          url="resources/explo.png" sizeX="3" sizeY="3" timeSpan="1.0"
               loop="true" tileX="4" tileY="4" usePitch="true"/>
96          <resource mime="font/txf" id="10" name="default"
97          url="default.txf" size="24"/>
98          <resource mime="audio/wav" id="11" loop="true"
99          url="resources/ding.wav"/>
100         <resource  mime="audio/wav" id="12" pitch="1.0" loop="true"
               attenuationDist="3.0"
101         url="resources/riff01.wav"/>
102         <resource mime="audio/wav" id="13" pitch="1.0" loop="true"
103         url="resources/river.wav"/>
104         <container id="20" name="box" shape="3" sound="12"/>
105         <container id="21" name="ufo" shape="5" sound="13"/>
106 </resources>
107 <!-- scene and simulation initialization -->
108 <scene id="0" script="main">
109         <object id="0" name="observer" script="camera.lua" input="
               window"/>
110         <object id="1" shape="1" surface="2" pos="0 0 0"/>
111         <object id="3" shape="20" sound="20" pos="-5 5 0 225 0 0"/>
112         <object id="5" shape="21" sound="21" pos="2 2 2" speed="0 2 0
               30 0 0"/>
113         <object id="6" shape="6" pos="0 7 5"/>
114         <object id="7" shape="7" pos="-4.5 4.5 1.7"/>
115         <object id="11" sound="11" pos="10 -10 0"/>
116         <light id="0" enabled="1" position="-.5 -.75 1.0 0.0"
117         ambient="0.3 0.3 0.3 1.0" diffuse="0.7 0.7 0.7 1.0" specular="1
               .0 1.0 1.0 1.0"/>
118 </scene>
119 <cdata_test>
120         <chars><![CDATA[x<works>]]></chars>
```

```
        <tagged><![CDATA[<works>]]></tagged>
        <open><![CDATA[<]]></open>
        <close><![CDATA[>]]></close>
        <empty><![CDATA[]]></empty>
</cdata_test>
</XperiML>
```

13.3 附录 3

```
-- Additional path that may be required
require("json")  local testStrings = {
        [[{1:[1213.3e12, 123 , 123, "hello", [12, 2], {1:true /*test
           */}]}]],
        [[{"username":"demo1","message":null,"password":""}]],
        [[{"challenge":"b64d-fnNQ6bRZ7CYiNIKwmdHoNgl9JR9MIYtz
        jBhpQzYXCFrgARt9mNmgUuO7
   FoODGr1NieT9yTeB2SLztGkvIA4NXmN9Bi27hqx1ybJIQq6S2
        L-AjQ3VTDClSmCsYFPOm9EMVZDZ0j hBX1fXw3o9VYj1j9KzSY5VCSAzGqYo-
   cBPY\n.b64","cert":"
        b64MIIGyjCCBbKgAwIBAgIKFAC1ZgAAAAAUYzANBgkqh
        kiG9w0BAQUFADBZMRUwEwYKCZImiZP
        yLGQBGRYFbG9tp8uQuFjWGS_KxTHXz9v
        kLNFjOoZY2bOwzsdEpshuYSdvX-9
   bRvHTQcoMNz8Q9nXG1aMl5x1nbV5byQNTCJlz4gzMJeNfe
        KGci pdCj7B6e_VpF-n2P-dFZizUHjxMksCVZ3nTr51x3Uw\n.b64",
        "key":"D79B30BA7954DF520B44897A 6FF58919"}]],
        [[{"key":"D79B30BA7954DF520B44897A6FF58919"}]],
        [[{"val":undefined}]],
        [[{
        "Image": {
                "Width": 800,
                "Height": 600,
                "Title": "View from 15th Floor",
                 "Thumbnail": {
                    "Url":
                    "http://www.example.com/image/481989943",
                     "Height": 125,
                 "Width": "100"
```

```
                                    },
                    "IDs": [116, 943, 234, 38793]
                  }
              }]], [[ [
         {
            "precision": "zip",
        "Latitude": 37.7668,
            "Longitude": -122.3959,
        "Address": "",
        "City": "SAN FRANCISCO",
        "State": "CA",
            "Zip":   "94107",
        "Country": "US"
         },
         {
            "precision": "zip",
            "Latitude": 37.371991,
            "Longitude": -122.026020,
        "Address":"",
            "City": "SUNNYVALE",
        "State":"CA",
            "Zip":"94085",
            "Country":"US"
         }
           ] ]],
         [[[null,true,[1,2,3],"hello\"],[world!"] ]],
         [[ [{"0":"tan\\\\","model\\\\":"sedan"},{"0":"red","model":"sp
            orts"}] ]],
         [[ {"1":"one","2":"two","5":"five"} ]],
         [=[ [[[[[[[[[[[[[[[[[["Not too deep"]]]]]]]]]]]]]]]]] ]=]
}
for i, v in ipairs(testStrings) do
         print("Testing: #" .. i)
         local dec = json.decode(v)
         json.util.printValue(dec, "JSONVALUE")
         local reenc = json.encode(dec)
         print("RE_ENC: ", reenc)
         local redec = json.decode(reenc)
         json.util.printValue(redec,"REDECJSONVALUE")
```

```
end
local testValues = {
        {[300] = {nil, true, 1,2,3, nil, 3}}
}
for i, v in ipairs(testValues) do
        local ret = json.encode(v)
        print(ret)
        local dec = json.decode(ret)
json.util.printValue(dec, "Encoded value")
print("Re-encoded", json.encode(dec))
end
```

13.4 附录 4

```
sleep(0.5)
local xmlrpchttp=require("xmlrpc.http")
-- xmlrpchttp.call: 参数1: xmlrpc服务器地址, 参数2: 为请求的方法名, 其他参数: 为对应方法需要输入的参数 (有几个写几个)
local ok, res = xmlrpchttp.call("http://172.19.0.102:8080/RPC2", "dataSum", 1, 2)
elite_print(tostring(ok))
elite_print("Result: ",res)
```

ELITE ROBOTS
艾利特机器人

明天比今天更简单一点

联系我们

商务合作：market@elibot.cn
技术咨询：technical@elibot.cn

- **苏州公司（生产基地）**
  苏州市工业园区长阳街 259 号中新钟园工业坊 4 栋
  +86-400-189-9358
- **北京公司**
  北京市经济技术开发区荣华南路 2 号院 6 号楼 1102 室
- **上海公司（研创中心）**
  上海市浦东新区张江科学城学林路 36 弄 18 号
- **深圳公司**
  深圳市宝安区航空路泰华梧桐岛科技创新园 1A 栋 202 室
- **美国公司**
  10521 Research Dr., Ste. 104, 37932, Knoxville, TN (USA)
- **德国公司**
  Münchener Str. 53, 85290, Geisenfeld, Bavaria (Germany)
- **日本公司**
  TOSHIN Hirokoji Honmachi Bldg., 1F, 2-4-3 Sakae, Naka-ku, 460-0008, Nagoya (Japan)
- **墨西哥公司**
  Calzada del pedregal 523, fraccionamiento el pedregal

关注公众号了解更多