



ELITE ROBOTS  
艾利特机器人



# 艾利特机器人 CS 系列 CS脚本手册

苏州艾利特机器人有限公司

2022-11-24

版本:2.1.0



## 使用前请仔细阅读本手册

此版本手册对应产品版本信息请见用户手册版本信息章节，使用前请仔细核对实际产品版本信息，确保一致。

本手册会定期进行检查和修正,更新后的内容将出现在新版本中。本手册中的内容或信息如有变更,恕不另行通知。

苏州艾利特机器人有限公司对本手册中可能出现的任何错误概不负责。

苏州艾利特机器人有限公司对因使用本手册及其中所述产品而引起的意外或间接伤害概不负责。

安装、使用产品前,请阅读本手册。

请保管好本手册,以便可以随时阅读和参考。

本说明书图片仅供参考,请以收到的实物为准。



## 目录

<b>1 EliteScript 编程语言</b>	<b>1</b>
1.1 简介 .....	1
1.2 表述约定 .....	1
1.3 EliteScript 脚本 .....	1
1.3.1 数据类型 .....	2
1.3.2 变量 .....	7
1.3.3 条件与循环语句 .....	8
1.3.4 函数定义 .....	11
1.3.5 线程使用 .....	12
<b>2 运动指令</b>	<b>13</b>
2.1 MOVE 相关指令 .....	13
2.1.1 movej 指令 .....	13
2.1.2 movec 指令 .....	14
2.1.3 movel 指令 .....	15
2.1.4 get_actual_joint_positions 指令 .....	15
2.1.5 get_actual_joint_speeds 指令 .....	16
2.1.6 get_actual_tcp_pose 指令 .....	16
2.1.7 get_actual_tcp_speed 指令 .....	17
2.1.8 get_controller_temperature 指令 .....	17
2.1.9 get_joint_temperatures 指令 .....	17
2.1.10 get_joint_torques 指令 .....	18
2.1.11 get_target_joint_positions 指令 .....	18
2.1.12 get_target_joint_speeds 指令 .....	19
2.1.13 get_target_tcp_pose 指令 .....	19
2.1.14 get_target_tcp_speed 指令 .....	20

---

2.1.15 set_gravity 指令.....	20
2.1.16 set_payload 指令 .....	21
2.1.17 get_target_payload_cog 指令.....	21
2.1.18 get_target_payload_mass 指令 .....	22
2.1.19 set_tcp 指令 .....	22
2.1.20 encoder_enable_set_tick_count 指令 .....	23
2.1.21 encoder_get_tick_count 指令 .....	23
2.1.22 encoder_set_tick_count 指令.....	24
2.1.23 encoder_unwind_delta_tick_count 指令 .....	24
2.1.24 stop_conveyor_tracking 指令 .....	25
2.1.25 speedj 指令 .....	25
2.1.26 speedl 指令 .....	26
2.1.27 stopj 指令 .....	26
2.1.28 stopl 指令 .....	27
2.1.29 track_conveyor_circular 指令.....	27
2.1.30 track_conveyor_linear 指令 .....	28
2.1.31 get_actual_tool_flange_pose 指令 .....	28
2.1.32 get_target_waypoint 指令 .....	29
2.1.33 get_tcp_offset 指令 .....	29
2.1.34 load_micro_line_file 指令 .....	30
2.1.35 micro_line_set_pcs 指令 .....	30
2.1.36 get_micro_line_values_by_index 指令.....	31
2.1.37 moveml 指令 .....	31
2.2 MATH 相关指令 .....	32
2.2.1 get_inverse_kin 指令 .....	32
2.2.2 get_forward_kin 指令 .....	33

---

2.2.3 binary_list_to_integer 指令 .....	33
2.2.4 integer_to_binary_list 指令 .....	34
2.2.5 get_list_length 指令.....	34
2.2.6 length 指令.....	35
2.2.7 acos 指令 .....	35
2.2.8 asin 指令.....	36
2.2.9 atan 指令.....	36
2.2.10 atan2 指令.....	37
2.2.11 cos 指令 .....	37
2.2.12 sin 指令 .....	38
2.2.13 tan 指令.....	38
2.2.14 d2r 指令 .....	39
2.2.15 r2d 指令 .....	39
2.2.16 ceil 指令 .....	40
2.2.17 floor 指令.....	40
2.2.18 log 指令.....	41
2.2.19 sqrt 指令 .....	41
2.2.20 pow 指令 .....	42
2.2.21 norm 指令 .....	42
2.2.22 normalize 指令 .....	43
2.2.23 point_dist 指令 .....	43
2.2.24 pose_add 指令.....	44
2.2.25 pose_dist 指令 .....	45
2.2.26 pose_inv 指令.....	45
2.2.27 pose_sub 指令 .....	46
2.2.28 pose_trans 指令.....	47

---

2.2.29 rotvec2rpy 指令.....	48
2.2.30 rpy2rotvec 指令.....	48
2.2.31 interpolate_pose 指令.....	49
2.2.32 random 指令.....	49
<b>3 IO 指令</b>	<b>50</b>
3.1 get_configurable_digital_in 指令.....	50
3.2 get_configurable_digital_out 指令.....	50
3.3 get_standard_analog_in 指令.....	51
3.4 get_standard_analog_out 指令.....	51
3.5 get_standard_digital_in 指令.....	52
3.6 get_standard_digital_out 指令.....	52
3.7 get_tool_analog_in 指令.....	53
3.8 get_tool_analog_out 指令.....	53
3.9 set_standard_analog_output_domain 指令.....	54
3.10 set_configurable_digital_out 指令.....	54
3.11 set_standard_analog_input_domain 指令.....	55
3.12 set_standard_analog_out 指令.....	55
3.13 set_standard_digital_out 指令.....	56
3.14 set_tool_analog_input_domain 指令.....	56
3.15 set_tool_analog_output_domain 指令.....	57
3.16 set_tool_digital 指令.....	57
3.17 get_tool_digital 指令.....	58
3.18 set_runstate_configurable_digital_output_to_value 指令.....	59
3.19 set_runstate_gp_boolean_output_to_value 指令.....	60
3.20 set_runstate_standard_analog_output_to_value 指令.....	61
3.21 set_runstate_tool_analog_output_to_value 指令.....	62



---

3.22 set_runstate_standard_digital_output_to_value 指令 .....	63
3.23 set_runstate_tool_digital_output_to_value 指令 .....	64
3.24 set_input_actions_to_default 指令 .....	65
3.25 set_configurable_digital_input_action 指令 .....	66
3.26 set_standard_digital_input_action 指令 .....	67
3.27 set_tool_digital_input_action 指令 .....	68
3.28 set_gp_boolean_input_action 指令 .....	69
3.29 tool_serial_is_open 指令 .....	69
3.30 tool_serial_config 指令 .....	70
3.31 tool_serial_read 指令 .....	71
3.32 tool_serial_write 指令 .....	71
3.33 tool_modbus_read_registers 指令 .....	72
3.34 tool_modbus_read_bits 指令 .....	72
3.35 tool_modbus_write_registers 指令 .....	73
3.36 tool_modbus_write_bits 指令 .....	73
3.37 tool_serial_mode 指令 .....	74
3.38 set_tool_voltage 指令 .....	74
3.39 serial_open 指令 .....	75
3.40 serial_close 指令 .....	75
3.41 serial_config 指令 .....	76
3.42 serial_read 指令 .....	77
3.43 serial_write 指令 .....	78
3.44 serial_is_open 指令 .....	78
3.45 serial_open_list 指令 .....	79
<b>4 总线指令</b> .....	<b>80</b>
4.1 Modbus .....	80

---

4.1.1 modbus_add_signal 指令 .....	80
4.1.2 modbus_delete_signal 指令 .....	81
4.1.3 modbus_get_signal_status 指令 .....	81
4.1.4 modbus_send_custom_command 指令 .....	82
4.1.5 modbus_set_output_register 指令 .....	82
4.1.6 modbus_set_output_signal 指令 .....	83
4.1.7 modbus_set_runstate_dependent_choice 指令 .....	83
4.1.8 modbus_set_signal_update_frequency 指令 .....	83
4.1.9 read_port_bit 指令 .....	84
4.1.10 read_port_register 指令 .....	84
4.1.11 write_port_bit 指令 .....	85
4.1.12 write_port_register 指令 .....	85
4.1.13 read_input_boolean_register 指令 .....	86
4.1.14 read_input_float_register 指令 .....	86
4.1.15 read_input_integer_register 指令 .....	87
4.1.16 read_output_boolean_register 指令 .....	87
4.1.17 read_output_float_register 指令 .....	88
4.1.18 read_output_integer_register 指令 .....	88
4.1.19 write_output_boolean_register 指令 .....	89
4.1.20 write_output_float_register 指令 .....	89
4.1.21 write_output_integer_register 指令 .....	90
<b>5 通讯指令</b> .....	<b>91</b>
5.1 SOCKET .....	91
5.1.1 socket_open 指令 .....	91
5.1.2 socket_close 指令 .....	91
5.1.3 socket_get_var 指令 .....	92

---

5.1.4 socket_read_ascii_float 指令.....	93
5.1.5 socket_read_binary_integer 指令 .....	94
5.1.6 socket_read_byte_list 指令.....	95
5.1.7 socket_read_string 指令.....	96
5.1.8 socket_send_byte 指令.....	97
5.1.9 socket_send_int 指令 .....	98
5.1.10 socket_send_line 指令.....	98
5.1.11 socket_send_string 指令.....	99
5.1.12 socket_set_var 指令.....	99
5.2 RPC .....	100
5.2.1 rpc_factory 指令 .....	100
<b>6 系统指令</b>	<b>101</b>
6.1 线程 .....	101
6.1.1 start_thread 指令.....	101
6.1.2 stop_thread 指令.....	101
6.2 提示警告 .....	102
6.2.1 textmsg 指令 .....	102
6.2.2 popup 指令 .....	102
6.3 字符 .....	103
6.3.1 str_at 指令.....	103
6.3.2 str_cat 指令.....	104
6.3.3 str_empty 指令.....	104
6.3.4 str_find 指令.....	105
6.3.5 str_len 指令 .....	105
6.3.6 str_sub 指令 .....	106
6.3.7 to_num 指令 .....	107

6.3.8 to_str 指令 .....	108
6.4 其他 .....	109
6.4.1 sleep 指令 .....	109
6.4.2 set_flag 指令 .....	109
6.4.3 get_flag 指令 .....	110
6.4.4 get_steptime 指令.....	110

# 1 EliteScript 编程语言

## 1.1 简介

艾利特机器人提供两种编程方式：图形化编程和脚本编程。图形化编程采用艾利特机器人示教器界面完成，快速实现点位的示教与编辑；脚本编程则是底层的编程语言，既可通过图形化界面编程也可以通过文本编辑器编程。图形化程序在发送给控制器执行之前，会先解析成脚本语言，进而由控制器解释执行；图形化程序可以包含脚本语言，反之则不行。

EliServer 是控制器中的底层控制程序。用户界面 EliRobot 通过 TCP/IP 协议与 EliServer 连接，并将图形界面生成的脚本程序发送给 EliServer 来运行。对于外部设备中的脚本程序也可以通过 TCP/IP 接口连接到 EliServer 上，从而控制机器人的运行（需开启远程模式）。

EliteScript 是艾利特机器人脚本编程语言，像任何其他编程语言一样，EliteScript 也有变量、类型、控制流语句、函数等。此外，EliteScript 有许多内置变量和函数，可监视并控制机器人的输入/输出和运动。

## 1.2 表述约定

```
instruction(para1, para2=default_value2,..., <paraM=valuem, paraL=valuel | paraK=valueK>, paraN= default_valueN)
```

其中：

paraX=default\_valueX 表示这是可选参数，若未指定则用默认值，若 default\_valueX 为单引号引用的斜体(Italic)，则代表是特殊含义，例如 'current\_joint\_positions' 代表当前关节角；

|代表是互斥的参数。

## 1.3 EliteScript 脚本

EliteScript 与 Python 语言相似，但也有一些与机器人相关的指令和特定数据类型以及使用方法，本章节中选取了部分较为常用的变量类型、语法规则以及 EliteScript 中定制化的语法、关键字、线程等功能的介绍。如果在本章节未涉及的数据类型、语法规则等编程规则，均可以参考 Python 语言默认的编程规则直接进行使用。

## 1.3.1 数据类型

EliteScript 支持 Number,String,List,Tuple,Dictionary 通用 Python 数据类型,同时也支持 Joint 以及 Pose 等定制的数据类型。

### 1.3.1.1 Number (数字)

#### 不可改变的数据类型

当其类型被改变时,将会赋值给一个新的对象。当对变量赋予了数值后,这个对象就会被创建,可通过 del 语句删除对这些对象的引用。

#### 脚本支持的数字类型

表 1-1 脚本支持的数字类型

类型	备注
int	有符号整型,如 0x69, 10
long	长整型[也可以代表八进制和十六进制],如-4721885298529L,数字后面的 L 表示长整型
float	浮点型,如 70.2E-12
complex	复数,如 4.53e-7j

### 1.3.1.2 String (字符串)

字符串由数字、字母、下划线组成。

#### 字符串截取

脚本中字符串从左至右截取:索引范围 (0, 长度-1),从右至左截取 (-1, 字符串开头)。

#### 脚本中不存在单字符

脚本中即使有单字符,也会被当作字符串处理。

## 脚本转义字符

表 1-2 脚本转义字符

字符	描述
\	出现在行尾时表现为续行符，出现在行中时，用于“翻译”特殊字符表示特殊含义，如下面选项所示
\\	反斜杠符号
\'	单引号
\"	双引号
\a	响铃
\b	退格(Backspace)
\e	转义
\000	空
\n	换行
\v	纵向制表符
\t	横向制表符
\r	回车
\f	换页
\oyy	八进制数，yy 代表的字符，例如：\o12 代表换行
\xyy	十六进制数，yy 代表的字符，例如：\x0a 代表换行
\other	其它的字符以普通格式输出

### 1.3.1.3 List (列表)

列表数据结构使用非常频繁，支持数字、字符、字符串甚至列表的集合结构。

#### 增加或删除列表元素

直接重新赋值给根据索引值取出的值，或通过 `append ()` 函数来添加；

通过 `del` 语句删除列表项，如：`dellist1[2]`。

## 列表的脚本操作符

和对字符串的操作类似。

**表 1-3** 列表的脚本操作符

方法	结果	描述
<code>len([1,2,3])</code>	3	长度
<code>[1,2,3] + [4,5,6]</code>	<code>[1,2,3,4,5,6]</code>	组合
<code>['Hi!'] * 4</code>	<code>['Hi!','Hi!','Hi!','Hi!']</code>	重复
<code>3 in [1,2,3]</code>	True	判断元素是否存在于列表中
<code>for x in [1,2,3]: print x</code>	1 2 3	迭代

## 列表的截取

**表 1-4** 列表的截取

方法	结果	描述
<code>L[2]</code>	'CS612'	读取列表中第三个元素
<code>L[-2]</code>	'CS66'	读取列表中倒数第二个元素
<code>L[1:]</code>	<code>['CS66','CS612']</code>	从第二个元素开始截取列表

注: L 列表['CS63','CS66','CS612']

## 脚本中列表的函数及方法

脚本包含以下函数:

**表 1-5** 脚本中列表的函数

方法名称	描述
<code>cmp(list1,list2)</code>	比较两个列表的元素
<code>len(list)</code>	列表元素个数
<code>max(list)</code>	返回列表元素最大值
<code>min(list)</code>	返回列表元素最小值
<code>list(seq)</code>	将元组转换为列表



脚本包含以下方法：

表 1-6 脚本中列表的方法

方法名称	描述
list.append(obj)	在列表末尾添加新的对象
list.count(obj)	统计某个元素在列表中出现的次数
list.extend(seq)	在列表末尾一次性追加另一个序列中的多个值（用新列表扩展原来的列表）
list.index(obj)	从列表中找出某个值第一个匹配项的索引位置
list.insert(index,obj)	将对象插入列表
list.pop(obj=list[-1])	移除列表中的一个元素（默认最后一个元素），并且返回该元素的值
list.remove(obj)	移除列表中某个值的第一个匹配项
list.reverse()	反向列表中元素
list.sort([func])	对原列表进行排序

### 1.3.1.4 Tuple (元组)

#### 与列表的区别

类似列表，但列表用[]标识，元组用()标识，并且列表元素可二次赋值，但元组元素不能。

#### 元组的创建

创建空元组：tuple ()。

创建只有一个元素的元组：tuple (a,)，必须要在元素后加逗号。

#### 元素的访问

虽然创建时用 () 包含，但是在访问单个元素时，与列表一样，通过[索引号]来访问。

#### 删除元组

元组中的单个元素不能被删除，但是元组可以通过 del 语句整个删除。

## 元组运算符 (同列表)

任意无符号的对象, 以逗号隔开, 默认为元组 (无关闭分隔符)

## 元组内置函数

表 1-7 元组内置函数

方法	描述
<code>cmp(tuple1,tuple2)</code>	比较两个元组元素。
<code>len(tuple)</code>	计算元组元素个数。
<code>max(tuple)</code>	返回元组中元素最大值。
<code>min(tuple)</code>	返回元组中元素最小值。
<code>tuple(seq)</code>	将列表转换为元组。

### 1.3.1.5 Dictionary (字典)

#### 与列表的差别

列表是有序对象集合, 字典是无序对象结合。字典中的元素通过 Key 来获取, 而列表中的元素通过位移来获取。

#### 字典的定义

下面是两种定义字典的方法, 两种方法都与列表的定义方法类似。

```
dict={}
dict['one']="This is one"
dict[2]="This is two"
tinydict={'name':'john','code':6734,'dept':'sales'}
```

## 数据类型的转换

表 1-8 数据类型的转换

方法	描述
int(x [,base])	将 x 转换为一个整数
long(x [,base])	将 x 转换为一个长整数
float(x)	将 x 转换到一个浮点数
complex(real [,imag])	创建一个复数
str(x)	将对象 x 转换为字符串
repr(x)	将对象 x 转换为表达式字符串
eval(str)	用来计算在字符串中的有效 Python 表达式,并返回一个对象
tuple(s)	将序列 s 转换为一个元组
list(s)	将序列 s 转换为一个列表
set(s)	转换为可变集合
dict(d)	创建一个字典。d 必须是一个序列 (key,value)元组。
frozenset(s)	转换为不可变集合
chr(x)	将一个整数转换为一个字符
unichr(x)	将一个整数转换为 Unicode 字符
ord(x)	将一个字符转换为它的整数值
hex(x)	将一个整数转换为一个十六进制字符串
oct(x)	将一个整数转换为一个八进制字符串

### 1.3.2 变量

EliteScript 脚本语言, 根据变量的作用域不同, 将变量区分为局部变量与全局变量两种类型, 脚本中定义的带有 global 关键字的全局变量会出现在 EliRobot 的变量监视页面。

#### 1.3.2.1 全局变量

定义全局变量时, 请用 global 关键字进行定义。

#### 1.3.2.2 局部变量

定义在函数内则为局部变量, 只能在相应的代码块内使用。

## 1 EliteScript 编程语言

```
# 定义全局变量
global total
total=0

# 可写函数说明
def sum(arg1,arg2): #返回 2 个参数的和
    total=arg1 + arg2 # total 在这里是局部变量
    print("函数内是局部变量 :",total)
    return(total)

#调用 sum 函数
sum(10,20);
print("函数外是全局变量 :",total)
```

在上面的例子中，在函数内部，total 是局部变量，而在外部 total 是全局变量，局部变量的改变不会改变全局变量的值，因此第一个打印结果是 30，而第二个是 0。

### 1.3.3 条件与循环语句

#### 1.3.3.1 条件语句

脚本语言不支持 switch 语句，因此判断结果对应多种执行方式时，只能用 elif 或多层嵌套 if 语句来实现。

```
num=5
if num==3: #判断 num 的值
    print('boss')
elif num==2:
    print('user')
elif num==1:
    print('worker')
```

#### 1.3.3.2 循环语句

脚本语言中没有 do while 循环，支持的循环语句如下所示。

**表 1-9 循环语句**

循环	描述
while 循环	在给定的判断条件为 true 时执行循环体，否则退出循环体。
for 循环	重复执行语句
嵌套循环	可以在 while 循环体中嵌套 for 循环 (for 中也可以嵌套 for)

**循环控制语句：****表 1-10** 循环控制语句

控制语句	描述
break 语句	在语句块执行过程中终止循环，并且跳出整个循环
continue 语句	在语句块执行过程中终止当前循环，跳出该次循环，执行下一次循环。
pass 语句	pass 是空语句，是为了保持程序结构的完整性。

**pass 语句在函数中的作用**

在编写一个程序时，如果执行语句部分思路还没有完成，这时可以用 pass 语句来占位，也可以当做是一个标记，是要过后来完成的代码。比如下面这样：

```
def test():  
    pass  
end
```

定义一个函数 test，但函数体部分暂时还没有完成，又不能空着不写内容，因此可以用 pass 来替代占个位置。

**pass 语句在循环中的作用**

pass 也常用于为复合语句编写一个空的主体，比如想在一个 while 语句的无限循环，使之每次迭代不需要任何操作，可以这样写：

```
while True:  
    pass  
end
```

以上只是举个例子，现实中最好不要写这样的代码，因为执行代码块为 pass 也就是空什么也不做，这时 python 会进入死循环。

**pass 语句用法总结**

- 1、空语句，什么也不做；
- 2、在特别的时候用来保证格式或是语义的完整性。

## While 循环（可在循环中使用 else 语句）

\# continue 和 break 用法:

```
i=1
while i<10:
    i+=1
    if i%2>0: # 非双数时跳过输出
        continue
    print(i) # 输出双数 2、4、6、8、10
end

i=1
while 1: # 循环条件为 1 必定成立
    print(i) # 输出 1~10
    i+=1
    if i>10: # 当 i 大于 10 时跳出循环
        break
end
```

在循环中使用 else 语句，即当条件不满足之后，结束循环，执行 else 语句。

```
count=0
while count <5:
    print(count," is less than 5")
    count=count +1
else:
    print(count," is not less than 5")
```

## for 循环（可在循环中使用 else 语句）

可以通过直接取值迭代，也可以通过序列索引迭代。

### 取值迭代:

```
for letter in 'Python': # 逐个输出字符串中的字符，每次输出的字符都被临时定义为 letter
    (定义名称可以换成其他任意名称)
    print('当前字母:',letter)

fruits=['banana','apple','mango']
for fruit in fruits: # 逐个输出列表中的元素，每次输出的元素都被临时定义为 fruit（定义
    名称可以换成其他任意名称）
    print('当前水果:',fruit)
print("Good bye!")
```

### 索引迭代:

```
fruits=['banana','apple','mango']
#通过 len () 函数获得列表的长度，通过 range () 函数获得了一个值不超过长度-1 的索引序列
for index in range(len(fruits)):
    print('当前水果:',fruits[index])
    print("Good bye!")
```

## 1.3.4 函数定义

### 1.3.4.1 函数定义

```
# 标准函数定义
def mult(a,b):
    return a*b
end
```

```
# 带有默认参数的函数定义
def add(a=3,b=3)
    return a+b
end
```

# 注意: end 为 EliteScript 的特殊关键字, 代表的块结束, 但并不是必要的关键字, 使用过程中, 也可以使用标准的 Python 语法定义函数。

### 1.3.4.2 函数调用方法

```
ret_mult=mult(2,4)

#调用带有默认参数的函数定义, 可以不进行传参
ret_add=add()
```

### 1.3.4.3 调用函数时的参数使用规则

#### 必备参数

必须与函数声明时一致的顺序来传递参数。

#### 关键字参数

传参数时可与函数声明的顺序不一样, 因为脚本解释器可以用参数名来匹配参数值。

#### 缺省参数

传入参数时未给参数赋值, 则保持默认值。

```
#可写函数说明
def printinfo(name,age=35):
    print("Name: ",name) # 打印任何传入的字符串
    print("Age ",age)
    return;

#调用 printinfo 函数
printinfo(age=50,name="miki")# 此处使用了关键字参数, 所以可以与声明的顺序不同
printinfo(name="miki") # 此处 age 没有输入值, 则输出默认值 35。
```

## 不定长参数（也就是包含非必备参数的参数定义）

当不确定会传入参数的个数时，可以对不需要输入的参数名前面加 “\*” 号，按顺序输入时进行对应即可。

```
def printinfo(arg1,*vartuple):
print("输出: ",arg1) #"打印任何传入的参数"
for var in vartuple:
print var
return

# 调用 printinfo 函数
printinfo(10)
printinfo(70,60,50)
```

### 1.3.5 线程使用

脚本语言支持用户创建线程的功能，用户可以通过创建线程实现复杂的机器人控制逻辑。

#### 1.3.5.1 创建和关闭线程的方法

```
# 定义线程函数，定义方式与普通函数相同
def thread_add_func(a,b):
c=a + b
print(c)
return c
end

# 运行线程函数，此线程函数会被异步执行。start_thread 函数的返回值为线程句柄，可以在停止线程运行时使用。
thread_handler=start_thread(thread_add_func,(1,3))

# 停止线程函数运行，stop_thread 函数的参数值为运行线程时返回的句柄。
stop_thread(thread_handler)
```

#### 1.3.5.2 使用线程功能时的注意事项

- 系统中支持同时运行的最大线程数为 15 个；
- 主线程内创建的线程会在主线程停止运行时自动被结束；
- 启动线程函数时，可以传递参数；
- 线程函数内部的返回值，无法从线程外部访问到；
- 线程内部调度遵循 C Python API 标准调度；
- 子线程或主线程内的循环语句，请避免无 sleep 的死循环，此种操作可能会影响系统整体性能。



## 2 运动指令

### 2.1 MOVE 相关指令

这一部分主要介绍 MOVE 相关的指令。

#### 2.1.1 movej 指令

```
movej(q, <a=0, v=0|t=0>, r=0)
```

- **功能:**

该指令用于关节运动，即通过该指令可以将机器人移动到位置  $q$ ，使用该指令时，机器人必须处于停止状态，或者是 movej 和 movel 的转接状态。

- **参数:**

$q$ : 目标点的关节位置 (可使用逆运动学函数将笛卡尔空间坐标系转换为关节空间坐标系后输入), 格式为[Base,Shoulder,Elbow,Wrist1,Wrist2,Wrist3], list 型数据, 单位为 rad;

$a$ : 关节加速度, 单位  $\text{rad/s}^2$ , float 型数据;

$v$ : 关节速度, 单位  $\text{rad/s}$ , float 型数据;

$t$ : 时间, 单位  $s$ , float 型数据;

$r$ : 交融半径, 单位  $m$ , float 型数据。

- **返回值:** 无

- **示例:**

```
movej([0.57636, -1.01469, -2.04816, -1.29723, 1.5708, -0], a=1.4, v=1.05, t=0, r=0)
```

## 2.1.2 movec 指令

```
movec(p_via, p_to, a=0, v=0, r=0, mode=0)
```

- **功能:**

该指令用于圆弧轨迹运动指令，即从当前位置起始经过 p\_via 中间点到达 p\_to 目标点。当加速到恒定的工具速度 v 后，然后以此速度运动。

- **参数:**

p\_via: 中间点的工具位姿（可使用正运动学函数将关节空间坐标转化为笛卡尔空间坐标后输入），格式为 [x,y,z,Rx,Ry,Rz]，list 型数据，其中 x、y、z 单位为 m，Rx、Ry、Rz 单位为 rad；

p\_to: 目标点的工具位姿（可使用正运动学函数将关节空间坐标转化为笛卡尔空间坐标后输入），格式为 [x,y,z,Rx,Ry,Rz]，list 型数据，其中 x、y、z 单位为 m，Rx、Ry、Rz 单位为 rad；

a: 工具加速度，单位  $m/s^2$ ，float 型数据；

v: 工具速度，单位 m/s，float 型数据；

r: 交融半径，单位 m，float 型数据；

mode: 圆弧模式（0 为固定模式，其他值为无约束模式，不写默认为 0），integer 型数据。

- **返回值:** 无

- **示例:**

```
movec([0.41951, -0.16, 0.25745, -3.11757, -0, -1.5708],  
[0.41951, -0.00562, 0.25745, -3.11757, 0, -1.5708], a=1.4, v=1.05, r=0, mode=0)
```

### 2.1.3 movel 指令

```
movel(p, <a=0, v=0|t=0>, r=0)
```

- **功能:**

该指令用于直线运动指令，即通过该指令可以将机器人移动到位置  $p$ ，使用该指令时，机器人必须处于停止状态，或者是 `movej` 和 `movel` 的转接状态。

- **参数:**

$p$ : 目标点的工具位姿 (可使用正运动学函数将关节空间坐标转化为笛卡尔空间坐标后输入)，格式为  $[x,y,z,Rx,Ry,Rz]$ ，list 型数据，其中  $x$ 、 $y$ 、 $z$  单位为  $m$ ， $Rx$ 、 $Ry$ 、 $Rz$  单位为  $rad$ ；

$a$ : 工具加速度，单位  $m/s^2$ ，float 型数据；

$v$ : 工具速度，单位  $m/s$ ，float 型数据；

$t$ : 时间，单位  $s$ ，float 型数据；

$r$ : 交融半径，单位  $m$ ，float 型数据。

- **返回值:** 无

- **示例:**

```
movel([0.41951, -0.16, 0.25745, -3.11757, -0, -1.5708], a=1, v=1.05, t=0, r=0.03)
```

### 2.1.4 get\_actual\_joint\_positions 指令

```
get_actual_joint_positions()
```

- **功能:**

该指令用于获取所有关节的实际关节角位置。实际关节角位置以弧度为单位，并以长度为 6 的矢量返回。

- **参数:** 无

- **返回值:**

list 型数据:  $[Base, Shoulder, Elbow, Wrist1, Wrist2, Wrist3]$ ，以弧度为单位的当前实际关节角位置。

- **示例:**

```
global j
j=get_actual_joint_positions()
print(j)
```

返回值: 当前关节角位置

- **注意:**

输出可能不同于 `get_target_joint_positions()` 的输出，尤其是加速和重负载时。

## 2.1.5 get\_actual\_joint\_speeds 指令

```
get_actual_joint_speeds()
```

- **功能:**

该指令用于获取所有关节的实际关节速度。实际关节速度以 rad/s 为单位, 并以长度为 6 的矢量返回。

- **参数:** 无

- **返回值:**

list 型数据: [Base,Shoulder,Elbow,Wrist1,Wrist2,Wrist3], 以 rad/s 为单位的当前实际关节速度。

- **示例:**

```
global b
b=get_actual_joint_speeds()
print(b)
```

返回值: 当前关节速度

- **注意:**

输出可能不同于 get\_target\_joint\_speeds() 的输出, 尤其是加速和重负载时。

## 2.1.6 get\_actual\_tcp\_pose 指令

```
get_actual_tcp_pose()
```

- **功能:**

该指令用于获取当前 TCP 位姿。返回 6D 位姿——6D 位姿表示基架中规定的工具位置和方位。此姿态的计算基于实际机器人编码器读数。

- **参数:** 无

- **返回值:**

当前实际 TCP 的位姿, 格式为 [x,y,z,Rx,Ry,Rz], list 型数据, 其中 x、y、z 单位为 m, Rx、Ry、Rz 单位为 rad。

- **示例:**

```
global b
b=get_actual_tcp_pose()
print(b)
```

返回值: 当前工具 TCP 位姿

## 2.1.7 get\_actual\_tcp\_speed 指令

```
get_actual_tcp_speed()
```

- **功能:**

该指令用于获取当前 TCP 速度。返回 6D 速度——6D 速度的列表前三个数值是沿 x、y、z 轴的线性速度，后三个数值用于确定当前旋转轴 RVx、RVy、RVz，并且向量 [RVz,RVy,RVz] 的长度用于确定以 rad/s 为单位的角速度。

- **参数:** 无

- **返回值:**

当前实际 TCP 速度矢量，格式为 [Vx,Vy,Vz,RVx,RVy,RVz]，list 型数据，其中 Vx、Vy、Vz 单位为 m/s，RVx、RVy、RVz 单位为 rad/s。

- **示例:**

```
get_actual_tcp_speed()
```

返回值：当前实际 TCP 速度矢量

## 2.1.8 get\_controller\_temperature 指令

```
get_controller_temperature()
```

- **功能:**

该指令用于获取控制柜主板温度。

- **参数:** 无

- **返回值:**

float 型数据，控制柜主板温度，单位：摄氏度。

- **示例:**

```
get_controller_temperature()
```

返回值：控制柜主板温度，单位：摄氏度

## 2.1.9 get\_joint\_temperatures 指令

```
get_joint_temperatures()
```

- **功能:**

该指令用于获取机器人所有关节的温度。

- **参数:** 无

- **返回值:**

list 型数据: [Base,Shoulder,Elbow,Wrist1,Wrist2,Wrist3]，所有关节的温度，单位：摄氏度。

- **示例:**

```
get_joint_temperatures()
```

返回值：所有关节的温度，单位：摄氏度

## 2.1.10 get\_joint\_torques 指令

```
get_joint_torques()
```

- **功能:**

该指令用于获取所有关节的力矩, 单位: Nm;

关节力矩——通过所需的力矩校正使机器人自身移动(重力、摩擦力等), 以长度为 6 的矢量返回。

- **参数:** 无

- **返回值:**

list 型数据, 关节力矩矢量。

- **示例:**

```
global l
l=get_joint_torques()
print(l)
返回值: 1 到 6 轴的关节力矩
```

## 2.1.11 get\_target\_joint\_positions 指令

```
get_target_joint_positions()
```

- **功能:**

该指令用于获取所有关节目标关节角位置; 目标关节角位置以 rad 为单位, 并以长度为 6 的矢量返回。

- **参数:** 无

- **返回值:**

list 型数据, 以 rad 为单位的当前目标关节角位置:[Base,Shoulder,Elbow,Wrist1,Wrist2,Wrist3]。

- **示例:**

```
get_target_joint_positions()
返回值: 1 到 6 轴的关节目标关节角位置
```

- **注意:**

输出可能不同于 get\_actual\_joint\_positions() 的输出, 尤其是加速和重负载时。

## 2.1.12 get\_target\_joint\_speeds 指令

```
get_target_joint_speeds()
```

- **功能:**

该指令用于获取所有关节的目标关节速度。目标关节速度以 rad/s 为单位, 并以长度为 6 的矢量返回。

- **参数:** 无

- **返回值:**

list 型数据: [Base,Shoulder,Elbow,Wrist1,Wrist2,Wrist3], 以 rad/s 为单位的目标关节速度。

- **示例:**

```
get_target_joint_speeds()
```

返回值: 目标关节速度

## 2.1.13 get\_target\_tcp\_pose 指令

```
get_target_tcp_pose()
```

- **功能:**

该指令用于获取当前目标工具姿态。返回 6D 姿态, 6D 姿态表示基架中规定的工具位置和方位。此姿态的计算基于当前目标关节位置。

- **参数:** 无

- **返回值:**

当前目标 TCP 矢量: [x,y,z,Rx,Ry,Rz], list 型数据, 其中 x、y、z 单位为 m, Rx、Ry、Rz 单位为 rad。

- **示例:**

```
get_target_tcp_pose()
```

返回值: 当前目标 TCP 矢量

## 2.1.14 get\_target\_tcp\_speed 指令

```
get_target_tcp_speed()
```

- **功能:**

该指令用于获取目标 TCP 速度。返回 6D 速度——6D 速度的列表前三个数值是沿 x、y、z 轴的线性速度，后三个数值用于确定当前旋转轴 RVx、RVy、RVz，并且向量 [RVx,RVy,RVz] 的长度用于确定以 rad/s 为单位的角速度。

- **参数:** 无

- **返回值:**

目标 TCP 速度矢量，格式为 [Vx,Vy,Vz,RVx,RVy,RVz]，list 型数据，其中 Vx、Vy、Vz 单位为 m/s，RVx、RVy、RVz 单位为 rad/s。

- **示例:**

```
get_target_tcp_speed()
```

返回值：目标 TCP 速度矢量

## 2.1.15 set\_gravity 指令

```
set_gravity(d)
```

- **功能:**

该指令用于设置机器人经历的加速方向。机器人安装座固定好时，这对应于远离地面中心的加速度。

- **参数:**

d: 3D 矢量，描述了相对于机器人基座的重力方向，list 型数据。

- **返回值:** 无

- **示例:**

```
set_gravity([0,9.82 * sin(theta),9.82 * cos(theta)])
```

说明：设置机器人重力加速度方向为绕机器人基坐标系 X 轴旋转 theta 弧度。



## 2.1.16 set\_payload 指令

```
set_payload(m, CoG, inertia=[0,0,0,0,0,0])
```

- **功能:**

该指令用于设置机器人有效负载的质量、重心和转动惯量；

机器人的有效负载发生明显的改变时（比如拿起或放下较重的工件），需要调用该函数设置新的有效负载信息。

- **参数:**

m: 有效负载的质量, float 型数据: 范围不小于 0, 不大于当前型号机器人最大标称负载, 单位: kg;

CoG: 有效负载的重心坐标 (相对于法兰盘坐标系), list 型数据: [CoGx,CoGy,CoGz], 单位: m;

inertia: 有效负载的转动惯量, 可选参数, list 型数据: [lxx,lyy,lzz,lxy,lxz,lyz], 单位: kg/m<sup>2</sup>。

- **返回值:** 无

- **示例:**

```
set_payload(1,[0,0,0.12],[0,0,0,0,0,0])
```

## 2.1.17 get\_target\_payload\_cog 指令

```
get_target_payload_cog()
```

- **功能:**

该指令用于获取机器人当前负载重心坐标, 单位: m。

- **参数:** 无

- **返回值:**

list 型数据, [CoGx,CoGy,CoGz]。

- **示例:**

```
get_target_payload_cog()
```

返回值: 重心坐标

## 2.1.18 get\_target\_payload\_mass 指令

```
get_target_payload_mass()
```

- **功能:**

该指令用于获取机器人当前有效负载的质量。

- **参数:** 无

- **返回值:**

float 型数据, 以 kg 为单位的质量。

- **示例:**

```
get_target_payload_mass()
```

返回值: 机器人当前有效负载的质量

## 2.1.19 set\_tcp 指令

```
set_tcp(pose)
```

- **功能:**

该指令用于设置机器人工具中心点的坐标和姿态。

- **参数:**

pose: 描述变换的一种姿态。[x,y,z,Rx,Ry,Rz], list 型数据, 其中 x、y、z 单位为 m, Rx、Ry、Rz 单位为 rad。

- **返回值:** 无

- **示例:**

```
set_tcp([0,0,0.05,0,0.1,0.1])
```

## 2.1.20 encoder\_enable\_set\_tick\_count 指令

```
encoder_enable_set_tick_count(encoder_index, range_id)
```

- **功能:**

该指令用于设置一个能够通过 encoder\_set\_tick\_count 函数更新滴答计数的编码器。

- **参数:**

encoder\_index: 编码器的序号, 必须是 0 或者 1, integer 型数据;

range\_id: 译码器的序号: 用于指定编码器的范围, integer 型数据。需要为编码器选择合适的范围:

0 是一个 32 位有符号的译码器, 范围是[-2147483648 : 2147483647];

1 是一个 8 位无符号的译码器, 范围是[0 : 255];

2 是一个 16 位无符号的译码器, 范围是[0 : 65535];

3 是一个 24 位无符号的译码器, 范围是[0 : 16777215];

4 是一个 32 位无符号的译码器, 范围是[0 : 4294967295], integer 型数据。

- **返回值:** 无

- **示例:**

```
encoder_enable_set_tick_count(0,0)
```

## 2.1.21 encoder\_get\_tick\_count 指令

```
encoder_get_tick_count(encoder_index)
```

- **功能:**

该指令用于返回选定编码器的滴答计数。

- **参数:**

encoder\_index: 被访问的编码器序号, 序号值必须是 0 或者 1, integer 型数据。

- **返回值:**

float 型数据, 传送带编码器的滴答计数。

- **示例:**

```
encoder_get_tick_count(0)
```

返回值: 传送带编码器的滴答计数

## 2.1.22 encoder\_set\_tick\_count 指令

```
encoder_set_tick_count(encoder_index, count)
```

- **功能:**

该指令用于告诉机器人控制器当前编码器的嘀嗒计数。该函数可在绝对编码器条件下使用(e.g. MODBUS);

另请参阅: 2.1.20 encoder\_enable\_set\_tick\_count()指令。

- **参数:**

encoder\_index: 编码器序号, 序号值必须是 0 或者 1, integer 型数据;

count: 所设置的嘀嗒计数。必须在编码器设定的范围内, float 型数据。

- **返回值:** 无

- **示例:**

```
encoder_set_tick_count(0,1234)
```

## 2.1.23 encoder\_unwind\_delta\_tick\_count 指令

```
encoder_unwind_delta_tick_count(encoder_index, delta_tick_count)
```

- **功能:**

该指令用于: 如果编码器发生了计数越界行为, 则返回重构后的 delta\_tick\_count。如果编码器未发生计数越界行为, 则不对 delta\_tick\_count 进行任何修改, 直接返回。

- **参数:**

encoder\_index: 被访问的编码器序号, 序号值必须是 0 或者 1, integer 型数据;

delta\_tick\_count: 需要被重构的两次滴答计数的差值, float 型数据。

- **返回值:**

float 型数据, 重构后的 delta\_tick\_count。

- **示例:**

```
encoder_unwind_delta_tick_count(0,-64666)
```

说明:

假定序号为 0 的编码器, 其范围为[0 : 65535]

假定初次使用 encoder\_get\_tick\_count(0), 获取 tick\_count 为 65530

假定现在使用 encoder\_get\_tick\_count(0), 获取 tick\_count 为 864

此时 delta\_tick\_count=-64666 (864 - 65530=-64666)

返回值: 870

## 2.1.24 stop\_conveyor\_tracking 指令

```
stop_conveyor_tracking()
```

- **功能:**

该指令用于停止由 track\_conveyor\_linear()或 track\_conveyor\_circular()触发的传送带跟踪功能,同时将工具速度降为 0。

- **参数:** 无

- **返回值:** 无

- **示例:**

```
stop_conveyor_tracking()
```

## 2.1.25 speedj 指令

```
speedj(qd, a, t)
```

- **功能:**

该指令用于将每个关节加速到 qd, 并以此速度移动。时间参数 t 可选, 如不指定 t, 函数将在达到目标速度时返回; 指定 t 时, 函数将在 t 秒后返回, 无论是否已经达到目标速度。该指令受速度百分比影响, 当且仅当速度百分比为 100%时符合预期参数效果。

- **参数:**

qd: 每个关节的速度 (rad/s), list 型数据: 长度为 6, 每个元素依次指代了关节目标速度;

a: (主轴的) 关节加速度 (rad/s<sup>2</sup>), float 型数据;

t: 函数返回的最短时间 (s), float 型数据。

- **返回值:** 无

- **示例:**

```
speedj([0.5,1,1,1,1,1],0.5,1)
```

## 2.1.26 speedl 指令

```
speedl(xd, a, t)
```

- **功能:**

该指令用于工具端加速到恒定的速度  $xd$ ，并以此速度移动。时间参数  $t$  可选，如不指定  $t$ ，函数将在达到目标速度时返回；指定  $t$  时，函数将在  $t$  秒后返回，无论是否已经达到目标速度。该指令受速度百分比影响，当且仅当速度百分比为 100% 时符合预期参数效果。

- **参数:**

$xd$ : 工具速度 (m/s)，格式为  $[Vx, Vy, Vz, RVx, RVy, RVz]$ ，list 型数据，其中  $Vx$ 、 $Vy$ 、 $Vz$  单位为 m/s， $RVx$ 、 $RVy$ 、 $RVz$  单位为 rad/s；

$a$ : 工具加速度 ( $m/s^2$ )，float 型数据；

$t$ : 函数返回前的最短时间，float 型数据。

- **返回值:** 无

- **示例:**

```
speedl([-1.4E-4, -0.04, -0.02, 0.0, 0.0, -0.0], 1.2, 100.0)
```

## 2.1.27 stopj 指令

```
stopj(a)
```

- **功能:**

该指令用于将关节速度减速到零。

- **参数:**

$a$ : 关节加速度, 单位为  $rad/s^2$ , float 型数据。

- **返回值:** 无

- **示例:**

```
stopj(0.1)
```

## 2.1.28 stopl 指令

```
stopl(a)
```

- **功能:**

该指令用于将工具末端速度减速到零。

- **参数:**

a: 关节加速度, 单位为  $m/s^2$ 。

- **返回值:** 无

- **示例:**

```
stopl(0.5)
```

## 2.1.29 track\_conveyor\_circular 指令

```
track_conveyor_circular(center, ticks_per_revolution, rotate_tool=False,  
encoder_index=0)
```

- **功能:**

该指令用于使机器人运动 (movej() 等) 追踪圆形传送带。

- **参数:**

center: 姿态矢量, 用于确定机器人基座坐标系中传送带的中心位置, list 型数据;

ticks\_per\_revolution: 传送带移动一圈时编码器变化的刻度数, float 型数据;

rotate\_tool: 工具应当与传送带一起旋转, 或者停留在轨迹 (movel() 等) 规定的方位。不指定布尔值时, 使用默认布尔值为 False, boolean 型数据 (可选参数);

encoder\_index: 与传送带跟踪相联系的编码器序号, 序号值必须为 0 或者 1。不指定序号值时, 使用默认序号值为 0, integer 型数据 (可选参数)。

- **返回值:** 无

- **示例:**

```
track_conveyor_circular([0.5,0.5,0,0,0,0],500.0,False)
```

参数:

```
center=[0.5,0.5,0,0,0,0]
```

```
ticks_per_revolution=500.0
```

```
rotate_tool=False
```

```
encoder_index=0, 默认值
```

### 2.1.30 track\_conveyor\_linear 指令

```
track_conveyor_linear(direction, ticks_per_meter, encoder_index=0)
```

- **功能:**

该指令用于使机器人运动 (movej() 等) 追踪线性传送带。

- **参数:**

direction: 姿态矢量, 用于确定机器人基座坐标系中传送带的方向, list 型数据;

ticks\_per\_meter: 传送带移动一米时编码器变化的刻度数, float 型数据;

encoder\_index: 与传送带跟踪相联系的编码器序号, 序号值必须为 0 或者 1。不指定序号值时, 使用默认序号值为 0, integer 型数据 (可选参数)。

- **返回值:** 无

- **示例:**

```
track_conveyor_linear([1,0,0,0,0,0],1000.0)
```

参数:

```
direction=[1,0,0,0,0,0]
```

```
ticks_per_meter=1000.0
```

```
encoder_index=0, 默认值
```

### 2.1.31 get\_actual\_tool\_flange\_pose 指令

```
get_actual_tool_flange_pose()
```

- **功能:**

该指令用于获得当前工具法兰位姿。

- **参数:** 无

- **返回值:**

当前测量的工具法兰位姿, 格式为 [x,y,z,Rx,Ry,Rz], list 型数据, 其中 x、y、z 单位为 m, Rx、Ry、Rz 单位为 rad。

- **示例:**

```
get_actual_tool_flange_pose()
```

返回值: 当前的工具法兰位姿



## 2.1.32 get\_target\_waypoint 指令

```
get_target_waypoint()
```

- **功能:**

该指令用于获得当前移动的目标路径点。

- **参数:** 无

- **返回值:**

当前移动的目标路径点, 格式为  $[x,y,z,Rx,Ry,Rz]$ , list 型数据, 其中  $x$ 、 $y$ 、 $z$  单位为  $m$ ,  $Rx$ 、 $Ry$ 、 $Rz$  单位为  $rad$ 。

- **示例:**

```
get_target_waypoint()
```

返回值: 当前移动的目标路径点

## 2.1.33 get\_tcp\_offset 指令

```
get_tcp_offset()
```

- **功能:**

该指令用于获得当前目标工具偏移量。

- **参数:** 无

- **返回值:**

当前目标工具偏移量, 格式为  $[x,y,z,Rx,Ry,Rz]$ , list 型数据, 其中  $x$ 、 $y$ 、 $z$  单位为  $m$ ,  $Rx$ 、 $Ry$ 、 $Rz$  单位为  $rad$ 。

- **示例:**

```
get_tcp_offset()
```

返回值: 当前目标工具偏移量

### 2.1.34 load\_micro\_line\_file 指令

```
load_micro_line_file(file_name, interval_time, pos_type=0, unit_type=0)
```

- **功能:**

该指令用于通过微段文件名称, 设置微段相邻两点间的时间间隔、微段文件数据类型、微段文件单位类型参数加载微段文件, 生成并返回唯一的标识序号。

- **参数:**

file\_name: 微段文件名称, string 型数据;

interval\_time: 微段文件中两个点位之间的时间间隔, 单位为 ms, integer 型数据;

pos\_type: 微段文件数据类型, integer 型数据: 0: 关节空间数据, 1: 笛卡尔空间数据 (可选参数);

unit\_type: 微段文件单位类型, integer 型数据: 0: 关节空间数据为弧度、笛卡尔空间数据为米以及弧度, 1: 关节空间数据为度、笛卡尔空间数据为毫米以及弧度 (可选参数)。

- **返回值:**

integer 型数据, 当前微段文件以及相应配置下加载的微段序号。

- **示例:**

```
load_micro_line_file("new_pose1", 2)
```

参数:

```
new_pose1="new_pose1"
```

```
interval_time=2
```

返回值: 微段序号

### 2.1.35 micro\_line\_set\_pcs 指令

```
micro_line_set_pcs(micro_line_index, part_coordinate_system)
```

- **功能:**

该指令用于设置当前微段序号对应微段文件的局部坐标;

另请参阅: 2.1.34 load\_micro\_line\_file() 指令, micro\_line\_index 通常由 load\_micro\_line\_file() 获得。

- **参数:**

micro\_line\_index: 微段序号, integer 型数据;

part\_coordinate\_system: 局部坐标, list 型数据。

- **返回值:** 无

- **示例:**

```
micro_line_set_pcs(0, [1, 0, 0, 0, 0, 0])
```

### 2.1.36 get\_micro\_line\_values\_by\_index 指令

```
get_micro_line_values_by_index(micro_line_index, pos_type, line_index)
```

- **功能:**

该指令用于以关节空间数据或笛卡尔空间数据的方式返回微段序号对应微段文件对应行的数据;

另请参阅: 2.1.34 load\_micro\_line\_file()指令, micro\_line\_index 通常由 load\_micro\_line\_file() 获得。

- **参数:**

micro\_line\_index: 微段序号, integer 型数据;

pos\_type: 获取的数据类型, integer 型数据: 0: 关节空间数据, 1: 笛卡尔空间数据 (可选参数);

line\_index: 微段文件行序号, integer 型数据。

- **返回值:**

list 型数据, 关节空间数据或笛卡尔空间数据。

- **示例:**

```
get_micro_line_values_by_index(0,0,1)
```

返回值: 序号 1 对应微段文件第一行的关节角

### 2.1.37 moveml 指令

```
moveml(micro_line_index, start_line=0, end_line=0)
```

- **功能:**

该指令用于运行微段序号对应的微段文件, 从设定的起始行运行至设定的终止行;

另请参阅: 2.1.34 load\_micro\_line\_file()指令, micro\_line\_index 通常由 load\_micro\_line\_file() 获得。

- **参数:**

micro\_line\_index: 微段序号, integer 型数据;

start\_line: 运行的起始行, 默认为 0 代表从微段文件首行运行, integer 型数据 (可选参数) ;

end\_line: 运行的终止行, 默认为 0 代表运行至微段文件末行, integer 型数据 (可选参数)。

- **返回值:** 无

- **示例:**

```
moveml(0)
```

## 2.2 MATH 相关指令

这一部分主要介绍计算相关的指令。

### 2.2.1 get\_inverse\_kin 指令

```
get_inverse_kin(p, qnear='', tcp='active tcp')
```

- **功能:**

该指令用于运动学逆解计算，即从笛卡尔空间到关节空间的转换。如果定义了 qnear，则返回最接近 qnear 的解。否则，将返回最接近当前关节位置的解。如果未提供 tcp，则将使用控制器的当前激活 tcp。

- **参数:**

p: 位姿，格式为 [x,y,z,Rx,Ry,Rz]，list 型数据，其中 x、y、z 单位为 m，Rx、Ry、Rz 单位为 rad；

qnear: 最接近的关节角度，缺省值为当前机器人关节角，list 型数据（可选参数）；

tcp: 工具形状（可选参数）。

- **返回值:**

list 型数据，关节角度，单位：弧度。

- **示例:**

```
global v
po=[0.4,-0.2,0.4,-3.1,2.8,-1.8]
qn=[-0.2,-1.6,-1.4,-1.5,1.5,-1.6]
v=get_inverse_kin(po,qnear=qn)
参数:
p=po=[0.4,-0.2,0.4,-3.1,2.8,-1.8]
qnear=qn=[-0.2,-1.6,-1.4,-1.5,1.5,-1.6]
```

## 2.2.2 get\_forward\_kin 指令

```
get_forward_kin(q='', tcp='active tcp')
```

- **功能:**

该指令用于正运动学计算，即通过关节角度获取位姿，使用当前系统激活的 TCP 数据。如果未提供关节位置矢量，则将使用机器人本体的当前关节角度。如果未提供 tcp，则将使用控制器的当前激活 tcp。

- **参数:**

q: 机器人的关节角度，缺省值为当前机器人关节角度，单位为弧度，list 型数据（可选参数）。

tcp: 工具形状（可选参数）。

- **返回值:**

机器人位姿，格式为 [x,y,z,Rx,Ry,Rz]，list 型数据，其中 x、y、z 单位为 m，Rx、Ry、Rz 单位为 rad。

- **示例:**

```
global p_p
p_j=[-0.2,-1.6,-1.4,-1.5,1.5,0]
p_p=get_forward_kin(p_j)
参数:
q=p_j=[-0.2,-1.6,-1.4,-1.5,1.5,0]
```

## 2.2.3 binary\_list\_to\_integer 指令

```
binary_list_to_integer(l)
```

- **功能:**

该指令用于计算并返回列表 l 中包含的布尔值代表的整数值。

- **参数:**

l: 要转换为整数的布尔值列表。索引 0 处的布尔值作为最低有效位。False 代表 0，True 代表 1。如果列表是空的，此函数返回 0。如果列表含有超过 32 个布尔值，此函数返回列表中前 32 个布尔值所转换的有符号整数值，list 型数据。

- **返回值:**

integer 型数据，二进制列表所代表的整数值。

- **示例:**

```
binary_list_to_integer([True,4 <5,9==10,9%3==0])
返回值: 13
```

## 2.2.4 integer\_to\_binary\_list 指令

```
integer_to_binary_list(x)
```

- **功能:**

该指令用于计算并返回布尔值列表作为有符号整数值  $x$  的二进制数。

- **参数:**

$x$ : 要转换为二进制列表的整数值, 返回值含有 32 个布尔值的列表, 其中 False 代表 0, True 代表 1, list 型数据。

- **返回值:**

list 型数据, 二进制布尔值列表, 列表索引 0 处的值为输入值  $x$  的最低位。

- **示例:**

```
integer_to_binary_list(2)
```

```
返回值: [False, True, ..., False, False, False] (长度为 32)
```

## 2.2.5 get\_list\_length 指令

```
get_list_length(v)
```

- **功能:**

该指令用于计算并返回列表变量的长度, 列表长度是列表包含的条目数。

- **参数:**

$V$ : 列表变量, list 型数据。

- **返回值:**

integer 型数据, 用于规定既定列表长度的一个整数。

- **示例:**

```
list=[666,666,666,666,666,666,666,666,666,666]
```

```
list_length=get_list_length(list)
```

```
返回值: 10
```

## 2.2.6 length 指令

length(v)

- **功能:**

该指令用于计算并返回一个列表变量或字符串变量的长度。

- **参数:**

v: 一个列表或字符串变量, string 型数据或 list 型数据。

- **返回值:**

integer 型数据, 列表或字符串长度。

- **示例:**

```
length([1,2,3,4,5])  
返回值: 5  
length("hello world")  
返回值: 11
```

## 2.2.7 acos 指令

acos(f)

- **功能:**

该指令用于计算并返回参数 f 的反余弦值, 返回值单位为 rad。

- **参数:**

f: 待计算角度的余弦值, 定义域为: [-1, 1], float 型数据。

- **返回值:**

float 型数据, f 的反余弦值, 单位为 rad。

- **示例:**

```
acos(0.866)  
返回值: 0.5235 rad
```

## 2.2.8 asin 指令

`asin(f)`

- **功能:**

该指令用于计算并返回参数  $f$  的反正弦值，返回值单位为 rad。

- **参数:**

$f$ : 待计算角度的正弦值，定义域为:  $[-1, 1]$ ，float 型数据。

- **返回值:**

float 型数据， $f$  的反正弦值，单位为 rad。

- **示例:**

```
asin(0.5)
```

```
返回值: 0.5235 rad
```

## 2.2.9 atan 指令

`atan(f)`

- **功能:**

该指令用于计算并返回参数  $f$  的反正切值，返回值单位为 rad。

- **参数:**

$f$ : 待计算角度的正切值，float 型数据。

- **返回值:**

float 型数据， $f$  的反正切值，单位为 rad。

- **示例:**

```
atan(1.0)
```

```
返回值: 0.785 rad
```



## 2.2.10 atan2 指令

atan2(x, y)

- **功能:**

该指令用于计算并返回  $x/y$  的反正切值，返回值单位为 rad， $x$  和  $y$  值的符号决定了正确的象限。

- **参数:**

$x$ : 用于计算的值，float 型数据；

$y$ : 用于计算的值，float 型数据。

- **返回值:**

float 型数据， $x/y$  的反正切值，单位为 rad。

- **示例:**

```
atan2(1.0,1.0)
```

参数:

$x=1.0$ ，向量在  $x$  轴正方向上的投影

$y=1.0$ ，向量在  $y$  轴正方向上的投影

返回值: 0.785 rad

## 2.2.11 cos 指令

cos(f)

- **功能:**

该指令用于计算并返回参数  $f$  弧度角的余弦。

- **参数:**

$f$ : 待计算的值，float 型数据。

- **返回值:**

float 型数据， $f$  的余弦值。

- **示例:**

```
cos(3.1415926)
```

返回值: -1

## 2.2.12 sin 指令

`sin(f)`

- **功能：**  
该指令用于计算并返回参数  $f$  弧度角的正弦。
- **参数：**  
 $f$ : 待计算数值, float 型数据。
- **返回值：**  
float 型数据,  $f$  的正弦值。
- **示例：**

```
sin(3.1415926)  
返回值: 0
```

## 2.2.13 tan 指令

`tan(f)`

- **功能：**  
该指令用于计算并返回参数  $f$  弧度角的正切值。
- **参数：**  
 $f$ : 待计算数值, float 型数据。
- **返回值：**  
float 型数据,  $f$  的正切值。
- **示例：**

```
tan(0)  
返回值: 0
```

## 2.2.14 d2r 指令

d2r(d)

- **功能:**

该指令用于角度单位换算，将参数 d 的角度值转换成弧度值。

- **参数:**

d: 以 degree 为单位的角度，float 型数据。

- **返回值:**

float 型数据，以 rad 为单位的角度。

- **示例:**

d2r(90)

返回值: 1.5707963267949

## 2.2.15 r2d 指令

r2d(r)

- **功能:**

该指令用于角度单位换算，将参数 r 的弧度值转换成角度值。

- **参数:**

r: 以 rad 为单位的弧度，float 型数据。

- **返回值:**

float 型数据，以 degree 为单位的角度。

- **示例:**

r2d(3.1415926)

返回值: 180

## 2.2.16 ceil 指令

ceil(f)

- **功能:**

该指令用于计算并返回不小于参数 f 的最小整数值，向上取整。

- **参数:**

f: 待计算的值，float 型数据。

- **返回值:**

integer 型数据，向上取整后的整数。

- **示例:**

```
ceil(-8.8)
```

```
返回值: -8
```

## 2.2.17 floor 指令

floor(f)

- **功能:**

该指令用于计算并返回不大于 f 的最大整数，向下取整。

- **参数:**

f: 输入的浮点值，float 型数据。

- **返回值:**

integer 型数据，f 向下取整的整数。

- **示例:**

```
floor(8.8)
```

```
返回值: 8
```

## 2.2.18 log 指令

`log(b, f)`

- **功能:**

该指令用于计算并返回以 b 为底 f 的对数，如果 b 或 f 为负数，运行时会提示错误。

- **参数:**

b: 底数, float 型数据;

f: 真数, float 型数据。

- **返回值:**

float 型数据, 以 b 为底 f 的对数。

- **示例:**

`log(2, 3)`

返回值: 1.584

## 2.2.19 sqrt 指令

`sqrt(f)`

- **功能:**

该指令用于计算并返回参数 f 的平方根。如果 f 为负，运行时会提示错误。

- **参数:**

f: 待计算数值, float 型数据。

- **返回值:**

float 型数据, f 的平方根。

- **示例:**

`sqrt(4)`

返回值: 2

## 2.2.20 pow 指令

```
pow(base, exponent)
```

- **功能:**

该指令用于计算并返回以 base 为底数，exponent 为指数进行幂运算的值。

- **参数:**

base: 底数值，float 型数据；

exponent: 指数值，float 型数据。

- **返回值:**

float 型数据，幂运算的结果。

- **示例:**

```
pow(2, 3)
```

```
返回值: 8
```

## 2.2.21 norm 指令

```
norm(a)
```

- **功能:**

该指令用于计算并返回自变量的范数；

自变量可以是四种不同类型的其中一种：

姿态：在这种情况下，返回姿态的欧几里德范数；

浮点数：在这种情况下，返回绝对值 (a)；

整数：在这种情况下，返回绝对值 (a)；

列表：在这种情况下，返回列表的欧几里德范数，列表元素必须是数字。

- **参数:**

a: 姿态、浮点数、整数或列表，list 型数据、integer 型数据或 float 型数据。

- **返回值:**

float 型数据，a 的范数。

- **示例:**

```
norm(-10)
```

```
返回值: 10
```

## 2.2.22 normalize 指令

normalize(v)

- **功能:**

该指令用于对给定的由浮点数组成的列表内的数据进行归一化处理。若数组内所有数据均为 0，运行时提示触发无效参数异常。

- **参数:**

v: 给定的浮点数列表，list 型数据。

- **返回值:**

list 型数据，对所有数据单位化后的数组。

- **示例:**

```
如 normalize([0,1,0])  
返回值: [0,1,0]  
如 normalize([2,0,0])  
返回值: [1,0,0]  
如 normalize([1,1])  
返回值: [0.707,0.707]
```

## 2.2.23 point\_dist 指令

point\_dist(p\_from, p\_to)

- **功能:**

该指令用于计算点 p\_from 到点 p\_to 的距离。

- **参数:**

p\_from: 工具起始位姿，格式为 [x,y,z,Rx,Ry,Rz]，list 型数据，其中 x、y、z 单位为 m，Rx、Ry、Rz 单位为 rad；

p\_to: 工具目标位姿，格式为 [x,y,z,Rx,Ry,Rz]，list 型数据，其中 x、y、z 单位为 m，Rx、Ry、Rz 单位为 rad。

- **返回值:**

float 型数据，两个工具位置之间的距离（不考虑旋转）。

- **示例:**

```
global p_dist  
pf=[0.4,-0.2,0.4,-3.1,2.8,-2.3]  
pt=[-0.01,-1.5,-0.002,-1.5,1.5,-2.3]  
pd=point_dist(p_from, p_to)  
参数:  
p_from=pf=[0.4,-0.2,0.4,-3.1,2.8,-2.3]  
p_to=pt=[-0.01,-1.5,-0.002,-1.5,1.5,-2.3]  
返回值: 1.42116
```

## 2.2.24 pose\_add 指令

```
pose_add(p_1, p_2)
```

- **功能:**

该指令用于点 p\_1 和点 p\_2 位置和姿态的求和运算。p\_1 和 p\_2 均含有三个位置参数 (x、y、z)，合称为 P，以及三个旋转参数 (Rx、Ry、Rz)，合称为 R。此函数用于计算既定姿态相加的结果 p\_3，如下所示：

$$p_3.P = p_1.P + p_2.P$$

$$p_3.R = p_1.R * p_2.R$$

- **参数:**

p\_1: 工具位姿 1，格式为[x,y,z,Rx,Ry,Rz]，list 型数据，其中 x、y、z 单位为 m，Rx、Ry、Rz 单位为 rad；

p\_2: 工具位姿 2，格式为[x,y,z,Rx,Ry,Rz]，list 型数据，其中 x、y、z 单位为 m，Rx、Ry、Rz 单位为 rad。

- **返回值:**

位置部分与旋转部分乘积之和的列表，格式为 [x,y,z,Rx,Ry,Rz]，list 型数据，其中 x、y、z 单位为 m，Rx、Ry、Rz 单位为 rad。

- **示例:**

```
global add_pose
p1=[0.4, -0.2, 0.4, -3.1, 2.8, -2.3]
p2=[-0.01, -1.5, -0.002, -1.5, 1.5, -2.3]
add_pose=pose_add(p1,p2)
返回值: [0.39, -1.7, 0.398, -2.3149, 1.1819, -2.27039]
```



## 2.2.25 pose\_dist 指令

pose\_dist(p\_from, p\_to)

- **功能:**

该指令用于计算并返回点 p\_from 到点 p\_to 之间的姿态距离。

- **参数:**

p\_from: 工具起始姿态, 格式为[x,y,z,Rx,Ry,Rz], list 型数据, 其中 x、y、z 单位为 m, Rx、Ry、Rz 单位为 rad;

p\_to: 工具目标姿态, 格式为[x,y,z,Rx,Ry,Rz], list 型数据, 其中 x、y、z 单位为 m, Rx、Ry、Rz 单位为 rad。

- **返回值:**

float 型数据, 两个姿态之间的距离, 包括旋转。

- **示例:**

```
global p_dist
pf=[0.4,-0.2,0.4,-3.1,2.8,-2.3]
pt=[-0.01,-1.5,-0.002,-1.5,1.5,-2.3]
p_dist=pose_dist(pf,pt)
参数:
p_from=pf=[0.4,-0.2,0.4,-3.1,2.8,-2.3]
p_to=pt=[-0.01,-1.5,-0.002,-1.5,1.5,-2.3]
返回值: 2.50394
```

## 2.2.26 pose\_inv 指令

pose\_inv(p\_from)

- **功能:**

该指令用于计算并返回位姿的逆。

- **参数:**

p\_from: 工具位姿, 格式为 [x,y,z,Rx,Ry,Rz], list 型数据, 其中 x、y、z 单位为 m, Rx、Ry、Rz 单位为 rad。

- **返回值:**

p\_from 的逆, 格式为 [x,y,z,Rx,Ry,Rz], list 型数据, 其中 x、y、z 单位为 m, Rx、Ry、Rz 单位为 rad。

- **示例:**

```
global inv_pose
pf=[0.4,-0.2,0.4,-3.1,2.8,-2.3]
inv_pose=pose_inv(pf)
返回值: [0.023407, 0.41385, -0.43378, 0.23146, -0.25682, -0.86443]
```

## 2.2.27 pose\_sub 指令

```
pose_sub(p_to, p_from)
```

- **功能:**

该指令用于点 p\_to 和点 p\_from 的两点位姿相减。

- **参数:**

p\_to: 工具位姿, 格式为 [x,y,z,Rx,Ry,Rz], list 型数据, 其中 x、y、z 单位为 m, Rx、Ry、Rz 单位为 rad;

p\_from: 工具位姿, 格式为 [x,y,z,Rx,Ry,Rz], list 型数据, 其中 x、y、z 单位为 m, Rx、Ry、Rz 单位为 rad。

- **返回值:**

工具姿态变换, 格式为 [x,y,z,Rx,Ry,Rz], list 型数据, 其中 x、y、z 单位为 m, Rx、Ry、Rz 单位为 rad。

- **示例:**

```
global sub_pose
pt=[0.4, -0.2, 0.4, -3.1, 2.8, -2.3]
pf=[-0.01, -1.5, -0.002, -1.5, 1.5, -2.3]
sub_pose=pose_sub(pt,pf)
返回值: [0.41, 1.3, 0.402, 1.48725, 1.2969, -0.14867]
```

## 2.2.28 pose\_trans 指令

pose\_trans(p\_from, p\_from\_to)

- **功能:**

该指令用于姿态变换。第一个自变量 p\_from 用于变换第二个自变量 p\_from\_to, 然后返回结果。这意味着结果是从 p\_from 坐标系开始, 然后坐标系移动 p\_from\_to 生成的姿态。如果姿态被视为变换矩阵, 则如下所示:

$$T_{\text{world} \rightarrow \text{to}} = T_{\text{world} \rightarrow \text{from}} * T_{\text{from} \rightarrow \text{to}}$$

$$T_{\text{x} \rightarrow \text{to}} = T_{\text{x} \rightarrow \text{from}} * T_{\text{from} \rightarrow \text{to}}$$

- **参数:**

p\_fom: 起始姿态, 格式为[x,y,z,Rx,Ry,Rz], list 型数据, 其中 x、y、z 单位为 m, Rx、Ry、Rz 单位为 rad;

p\_from\_to: 相对于起始姿态的姿态变化, 格式为[x,y,z,Rx,Ry,Rz], list 型数据, 其中 x、y、z 单位为 m, Rx、Ry、Rz 单位为 rad。

- **返回值:**

生成的姿态, 格式为 [x,y,z,Rx,Ry,Rz], list 型数据, 其中 x、y、z 单位为 m, Rx、Ry、Rz 单位为 rad。

- **示例:**

```
global tp
pf=[0.4,-0.2,0.4,-3.1,2.8,-2.3]
pft=[-0.01,-1.5,-0.002,-1.5,1.5,-2.3]
tp=pose_trans(pf,pft)
返回值: [1.4,-1.2,0.3,-2.3,1.1,-2.2]
```

## 2.2.29 rotvec2rpy 指令

```
rotvec2rpy(rotation_vector)
```

- **功能:**

该指令用于计算与旋转向量 `rotation_vector` 相对应的 RPY 向量。

- **参数:**

`rotation_vector`: 旋转向量 (Vector3d) 弧度, 也称为轴角向量 (单位旋转轴乘以弧度为单位的旋转角度), list 型数据。

- **返回值:**

list 型数据, 以弧度表示的 RPY 向量 (Vector3d), 描述了一个围绕 X-Y-Z 的外在旋转的滚转-俯仰-偏航序列轴, (相对于绕 Z-Y' -X" 的固有旋转轴)。矩阵形式中, RPY 向量定义为  $R_{rpy} = R_z(\text{yaw})R_y(\text{pitch})R_x(\text{roll})$ 。

- **示例:**

```
rotvec2rpy([3.14,1.57,0])  
返回值: [-2.8, -0.1, 0.9]
```

## 2.2.30 rpy2rotvec 指令

```
rpy2rotvec(rpy_vector)
```

- **功能:**

该指令用于计算并返回与参数 `rpy_vector` 相对应的旋转向量。

- **参数:**

`rpy_vector`: 以弧度表示的 RPY 向量 (Vector3d), 描述了一个围绕 X-Y-Z 的外在旋转的滚转-俯仰-偏航序列轴, (相对于绕 Z-Y' -X" 的固有旋转轴)。矩阵形式中, RPY 向量定义为  $R_{rpy} = R_z(\text{yaw})R_y(\text{pitch})R_x(\text{roll})$ , list 型数据。

- **返回值:**

list 型数据, 以弧度为单位的旋转矢量 (Vector3d), 也称为 Axis-Angle 向量 (单位旋转轴乘以弧度为单位的旋转角度)。

- **示例:**

```
rpy2rotvec([3.14,1.57,0])  
返回值: [2.2, 0.001, -2.2]
```

### 2.2.31 interpolate\_pose 指令

```
interpolate_pose(p_from, p_to, alpha)
```

- **功能:**

该指令用于工具位置和方位的线性插值;

alpha 是 0 时, 返回 p\_from。alpha 是 1 时, 返回 p\_to;

alpha 从 0 变为 1 时, 返回从 p\_from 到 p\_to 的直线姿态 (以及大地定向变化);

如果 alpha 小于 0, 返回直线上 p\_from 之前的点;

如果 alpha 大于 1, 返回直线上 p\_to 之后的姿态。

- **参数:**

p\_from: 工具起始位姿, 格式为 [x,y,z,Rx,Ry,Rz], list 型数据, 其中 x、y、z 单位为 m, Rx、Ry、Rz 单位为 rad;

p\_to: 工具目标位姿, 格式为 [x,y,z,Rx,Ry,Rz], list 型数据, 其中 x、y、z 单位为 m, Rx、Ry、Rz 单位为 rad;

alpha: 浮点数, float 型数据。

- **返回值:**

插值姿态, 格式为 [x,y,z,Rx,Ry,Rz], list 型数据, 其中 x、y、z 单位为 m, Rx、Ry、Rz 单位为 rad。

- **示例:**

```
interpolate_pose(pf,pt,al)
pf=[0.4,-0.2,0.4,-3.1,2.8,-2.3]
pt=[-0.01,-1.5,-0.002,-1.5,1.5,-2.3]
al=0
返回值: [0.4,-0.2,0.4,0.041,0.3,0.8]
```

### 2.2.32 random 指令

```
random()
```

- **功能:**

该指令用于产生一个值域为[0,1]的随机数。

- **参数:** 无

- **返回值:**

float 型数据, 0 到 1 之间的伪随机数。

- **示例:**

```
random()
返回值: 0 到 1 之间的随机数
```

## 3 IO 指令

### 3.1 get\_configurable\_digital\_in 指令

```
get_configurable_digital_in(n)
```

- **功能:**

该指令用于获取可配置数字输入信号电平;

另请参阅: 3.5 get\_standard\_digital\_in 指令。

- **参数:**

n: 输入的索引, integer 型数据: [0:7]。

- **返回值:**

信号电平, boolean 型数据, True 或 False。

- **示例:**

```
get_configurable_digital_in(0)
```

返回值: 信号电平, True 或 False;

### 3.2 get\_configurable\_digital\_out 指令

```
get_configurable_digital_out(n)
```

- **功能:**

该指令用于获取可配置数字输出信号电平;

另请参阅: 3.6 get\_standard\_digital\_out 指令。

- **参数:**

n: 输出的索引, integer 型数据: [0:7]。

- **返回值:**

信号电平, boolean 型数据, True 或 False。

- **示例:**

```
get_configurable_digital_out(0)
```

返回值: 信号电平, True 或 False;

### 3.3 get\_standard\_analog\_in 指令

```
get_standard_analog_in(n)
```

- **功能:**  
该指令用于获取标准模拟输入信号电平;  
另请参阅: 3.7 get\_tool\_analog\_in 指令。
- **参数:**  
n: 输入的索引, integer 型数据: [0:1]。
- **返回值:**  
模拟信号电平, float 型数据, 单位为 A 或者 V。
- **示例:**

```
get_standard_analog_in(1)
```

返回值: 模拟信号电平, 单位为 A 或者 V;

### 3.4 get\_standard\_analog\_out 指令

```
get_standard_analog_out(n)
```

- **功能:**  
该指令用于获取标准模拟输出信号电平;  
另请参阅: 3.8 get\_tool\_analog\_out 指令。
- **参数:**  
n: 输出的索引, integer 型数据: [0:1]。
- **返回值:**  
模拟信号电平, float 型数据, 单位为 A 或者 V。
- **示例:**

```
get_standard_analog_out(1)
```

返回值: 模拟信号电平, 单位为 A 或者 V;

## 3.5 get\_standard\_digital\_in 指令

```
get_standard_digital_in(n)
```

- **功能:**  
该指令用于获取标准数字输入信号电平;  
另请参阅: 3.1 get\_configurable\_digital\_in 指令。
- **参数:**  
n: 输入的索引, integer 型数据: [0:15]。
- **返回值:**  
信号电平, boolean 型数据, True 或 False。
- **示例:**

```
get_standard_digital_in(0)
```

返回值: 信号电平, True 或 False;

## 3.6 get\_standard\_digital\_out 指令

```
get_standard_digital_out(n)
```

- **功能:**  
该指令用于获取标准数字输出信号电平;  
另请参阅: 3.2 get\_configurable\_digital\_out 指令。
- **参数:**  
n: 输出的索引, integer 型数据: [0:15]。
- **返回值:**  
信号电平, boolean 型数据, True 或 False。
- **示例:**

```
get_standard_digital_out(0)
```

返回值: 信号电平, True 或 False;



## 3.7 get\_tool\_analog\_in 指令

```
get_tool_analog_in()
```

- **功能:**  
该指令用于获取工具模拟输入电平;  
另请参阅: 3.3 get\_standard\_analog\_in 指令。
- **参数:** 无
- **返回值:**  
模拟输入电平, float 型数据, 单位为 A 或 V。
- **示例:**

```
get_tool_analog_in()
```

返回值: 工具模拟输入电平, 单位为 A 或者 V

## 3.8 get\_tool\_analog\_out 指令

```
get_tool_analog_out()
```

- **功能:**  
该指令用于获取工具模拟输出电平;  
另请参阅: 3.4 get\_standard\_analog\_out 指令。
- **参数:** 无
- **返回值:**  
模拟输出电平, float 型数据, 单位为 A 或者 V。
- **示例:**

```
get_tool_analog_out()
```

返回值: 工具模拟输出电平, 单位为 A 或者 V

### 3.9 set\_standard\_analog\_output\_domain 指令

```
set_standard_analog_output_domain(port, domain)
```

- **功能:**

该指令用于设置模拟输出的域;

另请参阅: 3.15 set\_tool\_analog\_output\_domain 指令。

- **参数:**

port: 输出的索引, integer 型数据: [0:1];

domain: 模拟输出域, integer 型数据: 0: 电流模式 (4-20mA), 1: 电压模式 (0-10V)。

- **返回值:** 无

- **示例:**

```
set_standard_analog_output_domain(0,0)
```

### 3.10 set\_configurable\_digital\_out 指令

```
set_configurable_digital_out(n, b)
```

- **功能:**

该指令用于设置可配置的数字输出信号电平;

另请参阅: 3.13 set\_standard\_digital\_out 指令。

- **参数:**

n: 输出的索引, integer 型数据: [0:7];

b: 信号电平, boolean 型数据, True 或 False。

- **返回值:** 无

- **示例:**

```
set_configurable_digital_out(0,True)
```

### 3.11 set\_standard\_analog\_input\_domain 指令

```
set_standard_analog_input_domain(port, domain)
```

- **功能:**

该指令用于设置控制箱内标准模拟输入的域;

另请参阅: 3.14 set\_tool\_analog\_input\_domain 指令。

- **参数:**

port: 输入的索引, integer 型数据: [0:1];

domain: 模拟输入域, integer 型数据: 0 代表电流模式 (4-20mA), 1 代表电压模式 (0-10V)。

- **返回值:** 无

- **示例:**

```
set_standard_analog_input_domain(0,0)
```

### 3.12 set\_standard\_analog\_out 指令

```
set_standard_analog_out(n, f)
```

- **功能:**

该指令用于设置标准模拟输出 IO 的相对信号电平值。

- **参数:**

n: 输出的索引, integer 型数据: [0:1];

f: 相对模拟信号电平 (模拟输出范围百分比), float 型数据: [0:1], 给定的值若大于 1, 则设置为 1, 小于 0, 则设置为 0。

- **返回值:** 无

- **示例:**

```
set_standard_analog_out(0,1.0)
```

参数:

n=0

f=1.0, 将标准模拟输出值设置 10V 或者 20mA (取决于模拟 IO 模式设置)

### 3.13 set\_standard\_digital\_out 指令

```
set_standard_digital_out(n, b)
```

- **功能:**  
该指令用于设置标准数字输出 IO 信号电平;  
另请参阅: 3.10 set\_configurable\_digital\_out 指令。
- **参数:**  
n: 输出的索引, integer 型数据: [0:15];  
b: 信号电平, boolean 型数据, True 或 False。
- **返回值:** 无
- **示例:**

```
set_standard_digital_out(0,True)
```

### 3.14 set\_tool\_analog\_input\_domain 指令

```
set_tool_analog_input_domain(domain)
```

- **功能:**  
该指令用于设置工具末端模拟输入的域;  
有关控制箱输入, 另请参阅 3.11 set\_standard\_analog\_input\_domain 指令。
- **参数:**  
domain: 模拟输入域, integer 型数据: 0 代表电流模式 (4-20mA), 1 代表电压模式 (0-10V)。
- **返回值:** 无
- **示例:**

```
set_tool_analog_input_domain(0)
```

## 3.15 set\_tool\_analog\_output\_domain 指令

```
set_tool_analog_output_domain(domain)
```

- **功能:**  
该指令用于设置工具末端模拟输出的域;  
有关控制箱输出, 另请参阅 3.9 set\_standard\_analog\_output\_domain 指令。
- **参数:**  
domain: 模拟输出域, integer 型数据: 0 代表电流模式 (4-20mA), 1 代表电压模式 (0-10V)。
- **返回值:** 无
- **示例:**

```
set_tool_analog_output_domain(0)
```

## 3.16 set\_tool\_digital 指令

```
set_tool_digital(n, b)
```

- **功能:**  
该指令用于设置给定的索引对应工具数字 IO 电平。
- **参数:**  
n: 工具数字 IO 的索引, integer 型数据: [0:3];  
b: 信号电平, boolean 型数据, True 或 False。
- **返回值:** 无
- **示例:**

```
set_tool_digital(1,True)
```

- **注意:**  
若当前索引所对应的数字 IO 为禁用或者非输出类型时, 使用此接口设置后, 会触发 IO 类型错误的运行时报警。

## 3.17 get\_tool\_digital 指令

```
get_tool_digital(n)
```

- **功能:**  
该指令用于获取给定的索引对应的工具数字 IO 信号电平。
- **参数:**  
n: 工具数字 IO 的索引, integer 型数据: [0:3]。
- **返回值:**  
信号电平, boolean 型数据, True 或 False。
- **示例:**

```
get_tool_digital(1)  
返回值: True 或 False
```

## 3.18 set\_runstate\_configurable\_digital\_output\_to\_value 指令

```
set_runstate_configurable_digital_output_to_value(port, runstate)
```

- **功能:**

该指令用于设置输出信号电平跟随任务运行状态改变（运行或停止）。

另请参阅:

3.19 set\_runstate\_gp\_boolean\_output\_to\_value 指令

3.22 set\_runstate\_standard\_digital\_output\_to\_value 指令

3.23 set\_runstate\_tool\_digital\_output\_to\_value 指令

- **参数:**

port: 输出的索引, integer 型数据: [0:7];

runstate: integer 型数据: [0:3]:

0: 维持信号电平不变;

1: 任务在非运行状态时信号电平为低, 任务运行状态信号电平不变;

2: 任务在非运行状态时信号电平为高, 任务运行状态信号电平不变;

3: 任务在非运行状态时信号电平为低, 任务运行状态信号电平为高。

- **返回值:** 无

- **示例:**

```
set_runstate_configurable_digital_output_to_value(1,2)
```

参数:

port=1

runstate=2, 任务在非运行状态时, 设置输出 1 的信号电平为高, 任务运行状态信号电平不变;

## 3.19 set\_runstate\_gp\_boolean\_output\_to\_value 指令

```
set_runstate_gp_boolean_output_to_value(port, runstate)
```

- **功能:**

该指令用于设置输出信号电平跟随任务运行状态改变（运行或停止）；

另请参阅：

3.18 set\_runstate\_configurable\_digital\_output\_to\_value 指令

3.22 set\_runstate\_standard\_digital\_output\_to\_value 指令

3.23 set\_runstate\_tool\_digital\_output\_to\_value 指令

- **参数:**

port: 输出的索引, integer 型数据: [0:127];

runstate: integer 型数据: [0:3]:

0: 维持信号电平不变;

1: 任务在非运行状态时信号电平为低, 任务运行状态信号电平不变;

2: 任务在非运行状态时信号电平为高, 任务运行状态信号电平不变;

3: 任务在非运行状态时信号电平为低, 任务运行状态信号电平为高。

- **返回值:** 无

- **示例:**

```
set_runstate_gp_boolean_output_to_value(1,2)
```

参数:

port=1

runstate=2, 任务在非运行状态时, 设置输出 1 的信号电平为高, 任务运行状态信号电平不变



## 3.20 set\_runstate\_standard\_analog\_output\_to\_value 指令

```
set_runstate_standard_analog_output_to_value(port, runstate)
```

- **功能:**

该指令用于设置输出模拟信号电平跟随任务运行状态改变（运行或停止）。
- **参数:**

port: 输出的索引, integer 型数据: [0:1];

runstate: integer 型数据: [0:3]:

0: 维持模拟信号电平不变;

1: 任务在非运行状态时模拟信号电平最低值, 任务运行状态模拟信号电平不变;

2: 任务在非运行状态时模拟信号电平为最高值, 任务运行状态模拟信号电平不变;

3: 任务在非运行状态时模拟信号电平为最低值, 任务运行状态模拟信号电平为最高值。
- **返回值:** 无
- **示例:**

```
set_runstate_standard_analog_output_to_value(1,2)
```

参数:

port=1

runstate=2, 任务在非运行状态时, 设置模拟输出 1 的信号电平为最高值, 任务运行状态信号电平不变

## 3.21 set\_runstate\_tool\_analog\_output\_to\_value 指令

```
set_runstate_tool_analog_output_to_value(runstate)
```

- **功能:**

该指令用于设置工具模拟输出模拟信号电平跟随任务运行状态改变（运行或停止）。

- **参数:**

runstate: integer 型数据: [0:3]:

0: 维持模拟信号电平不变;

1: 任务在非运行状态时模拟信号电平最低值, 任务运行状态模拟信号电平不变;

2: 任务在非运行状态时模拟信号电平为最高值, 任务运行状态模拟信号电平不变;

3: 任务在非运行状态时模拟信号电平为最低值, 任务运行状态模拟信号电平为最高值。

- **返回值:** 无

- **示例:**

```
set_runstate_tool_analog_output_to_value(2)
```

参数:

runstate=2, 任务在非运行状态时, 设置模拟输出 1 的信号电平为最高值, 任务运行状态信号电平不变

## 3.22 set\_runstate\_standard\_digital\_output\_to\_value 指令

```
set_runstate_standard_digital_output_to_value(port, runstate)
```

- **功能:**

该指令用于设置输出信号电平跟随任务运行状态改变（运行或停止）；

另请参阅：

3.18 set\_runstate\_configurable\_digital\_output\_to\_value 指令

3.19 set\_runstate\_gp\_boolean\_output\_to\_value 指令

3.23 set\_runstate\_tool\_digital\_output\_to\_value 指令

- **参数:**

port: 输出的索引, integer 型数据: [0:15];

runstate: integer 型数据: [0:3]:

0: 维持信号电平不变;

1: 任务在非运行状态时信号电平为低, 任务运行状态信号电平不变;

2: 任务在非运行状态时信号电平为高, 任务运行状态信号电平不变;

3: 任务在非运行状态时信号电平为低, 任务运行状态信号电平为高。

- **返回值:** 无

- **示例:**

```
set_runstate_standard_digital_output_to_value(1,2)
```

参数:

port=1

runstate=2, 任务在非运行状态时, 设置标准数字输出 1 的信号电平为高, 任务运行状态信号电平不变

## 3.23 set\_runstate\_tool\_digital\_output\_to\_value 指令

```
set_runstate_tool_digital_output_to_value(port, runstate)
```

- **功能:**

该指令用于设置输出信号电平跟随任务运行状态改变（运行或停止）；

另请参阅：

3.18 set\_runstate\_configurable\_digital\_output\_to\_value 指令

3.19 set\_runstate\_gp\_boolean\_output\_to\_value 指令

3.22 set\_runstate\_standard\_digital\_output\_to\_value 指令

- **参数:**

port: 输出的索引, integer 型数据: [0:3];

runstate: integer 型数据: [0:3]:

0: 维持信号电平不变;

1: 任务在非运行状态时信号电平为低, 任务运行状态信号电平不变;

2: 任务在非运行状态时信号电平为高, 任务运行状态信号电平不变;

3: 任务在非运行状态时信号电平为低, 任务运行状态信号电平为高。

- **返回值:** 无

- **示例:**

```
set_runstate_tool_digital_output_to_value(1,2)
```

参数:

port=1

runstate=2, 任务在非运行状态时, 设置输出 1 的信号电平为高, 任务运行状态信号电平不变

## 3.24 set\_input\_actions\_to\_default 指令

```
set_input_actions_to_default()
```

- **功能:**

该指令用于将所有标准数字输入、可配置数字输入、工具数字输入以及通用布尔输入寄存器的 IO 触发动作设置为 “defalut”;

另请参阅:

3.26 set\_standard\_digital\_input\_action 指令

3.25 set\_configurable\_digital\_input\_action 指令

3.27 set\_tool\_digital\_input\_action 指令

3.28 set\_gp\_boolean\_input\_action 指令

- **参数:** 无
- **返回值:** 无
- **示例:**

```
set_input_actions_to_default()
```

说明: 设置所有数字输入以及通用布尔输入寄存器的 IO 触发动作为“defalut ”。

## 3.25 set\_configurable\_digital\_input\_action 指令

```
set_configurable_digital_input_action(port, action)
```

- **功能:**

该指令用于将可配置数字输入绑定触发动作，包括默认触发动作 (“default”) 和拖动模式触发动作 (“freedrive”), 并在绑定的输入信号电平为高时触发相应动作;

另请参阅:

3.24 set\_input\_actions\_to\_default 指令

3.26 set\_standard\_digital\_input\_action 指令

3.27 set\_tool\_digital\_input\_action 指令

3.28 set\_gp\_boolean\_input\_action 指令

- **参数:**

port: 输入的索引, integer 型数据: [0:7];

action: 触发动作类型 “default” 或 “freedrive” , string 型数据。

- **返回值:** 无

- **示例:**

```
set_configurable_digital_input_action(0,"freedrive")
```

参数:

port=0

action="freedrive", 当可配置输入 0 的信号电平为高时, 会打开拖动模式

## 3.26 set\_standard\_digital\_input\_action 指令

```
set_standard_digital_input_action(port, action)
```

- **功能:**

该指令用于将标准数字输入绑定触发动作，包括默认触发动作 (“default”) 和拖动模式触发动作 (“freedrive”)，并在绑定的输入信号电平为高时触发相应动作；

另请参阅：

3.24 set\_input\_actions\_to\_default 指令

3.25 set\_configurable\_digital\_input\_action 指令

3.27 set\_tool\_digital\_input\_action 指令

3.28 set\_gp\_boolean\_input\_action 指令

- **参数:**

port: 输入的索引, integer 型数据: [0:15];

action: 触发动作类型 “default” 或 “freedrive” , string 型数据。

- **返回值:** 无

- **示例:**

```
set_standard_digital_input_action(0,"freedrive")
```

参数:

port=0

action="freedrive", 当标准输入 0 的信号电平为高时, 会打开拖动模式

## 3.27 set\_tool\_digital\_input\_action 指令

```
set_tool_digital_input_action(port, action)
```

- **功能:**

该指令用于将工具数字输入绑定触发动作，包括默认触发动作 (“default”) 和拖动模式触发动作 (“freedrive”)，并在绑定的输入信号电平为高时触发相应动作；

另请参阅：

3.24 set\_input\_actions\_to\_default 指令

3.25 set\_configurable\_digital\_input\_action 指令

3.26 set\_standard\_digital\_input\_action 指令

3.28 set\_gp\_boolean\_input\_action 指令

- **参数:**

port: 输入的索引, integer 型数据: [0:3];

action: 触发动作类型 “default” 或 “freedrive” , string 型数据。

- **返回值:** 无

- **示例:**

```
set_tool_digital_input_action(0,"freedrive")
```

参数:

port=0

action="freedrive", 当工具数字输入 0 的信号电平为高时，会打开拖动模式；



## 3.28 set\_gp\_boolean\_input\_action 指令

```
set_gp_boolean_input_action(port, action)
```

- **功能:**

该指令用于将通用布尔输入寄存器绑定触发动作，包括默认触发动作 (“default”) 和拖动模式触发动作 (“freedrive”), 并在绑定的输入信号电平为高时触发相应动作;

另请参阅:

3.24 set\_input\_actions\_to\_default 指令

3.25 set\_configurable\_digital\_input\_action 指令

3.26 set\_standard\_digital\_input\_action 指令

3.27 set\_tool\_digital\_input\_action 指令

- **参数:**

port: 输入的索引, integer 型数据: [0:127];

action: 触发动作类型 “default” 或 “freedrive”, string 型数据。

- **返回值:** 无

- **示例:**

```
set_gp_boolean_input_action(0,"freedrive")
```

参数:

port=0

action="freedrive", 当通用布尔输入寄存器 0 的信号电平为高时, 会打开拖动模式

## 3.29 tool\_serial\_is\_open 指令

```
tool_serial_is_open()
```

- **功能:**

该指令用于获取工具串口通信功能的使能打开状态。

- **参数:** 无

- **返回值:**

串口通信的打开状态, boolean 型数据, True 为打开状态, False 为关闭状态。

- **示例:**

```
tool_serial_is_open()
```

返回值: True 或 False

### 3.30 tool\_serial\_config 指令

```
tool_serial_config(enabled, baud_rate, parity, stop_bits, size=8,  
modbus_rtu=False)
```

- **功能:**

该指令用于打开或者关闭工具串口功能，并对串口进行配置。

- **参数:**

enabled: 工具串口通信的使能状态, boolean 型数据;

baud\_rate: 串口波特率, integer 型数据: 9600,19200,38400,57600,115200,1000000,2000000;

parity: 奇偶校验位, integer 型数据:0(none),1(odd),2(even);

stop\_bits: 停止位, integer 型数据:1,2;

size: 串口的位宽。不指定时, 使用默认参数 8, 若使能 modbus-rtu 则该参数自动设置为 8;

modbus\_rtu: 是否启用 modbus-rtu 模式, 指定为 True 时为 Modbus-rtu 模式, 指定为 False 时为 RS485 模式, 不指定时使用默认参数 False, boolean 类型数据。

- **返回值:**

串口通信的打开状态, boolean 型数据, True 为打开状态, False 为关闭状态。

- **示例:**

```
tool_serial_config(True,115200,1,2)
```

参数:

enabled=True

baud\_rate=115200

parity=1

stop\_bits=2

返回值: True 或 False

- **注意:**

通信模式为 Modbus-rtu 时, 可使用下列 tool\_modbus\_read\_registers、tool\_modbus\_read\_bits、tool\_modbus\_write\_registers、tool\_modbus\_write\_bits 等指令;

通信模式为 RS485 时, 可使用下列指令 tool\_serial\_read、tool\_serial\_write。

### 3.31 tool\_serial\_read 指令

```
tool_serial_read(read_count, is_number, time_out)
```

- **功能:**

该指令用于从工具串口读取指定长度的字节数据。
- **参数:**

read\_count: 读取数据长度, integer 型数据: 大于等于 0;

is\_number: 是否按照整数格式返回字节数据, boolean 型数据;

time\_out: 超时时间, 单位: 秒, 若小于等于 0, 则等待超时时间为无限长, float 型数据。
- **返回值:**

bytearray 或 list, 若 is\_number 参数 True, 则返回值为整数类型的 list, 若 is\_number 参数 False, 则返回值为 bytes array, 若发生超时, 则返回所有已经读取到的数据。
- **示例:**

```
tool_serial_read(3,True,0.1)
```

返回值: 读取的整数数据 list, 类似 [255,1,0]

```
tool_serial_read(3,False,0.1)
```

返回值: 读取的字节数据, 类似 b"a/x90/x33"

### 3.32 tool\_serial\_write 指令

```
tool_serial_write(write_data)
```

- **功能:**

该指令用于向工具串口发送给定的字符串、整数或者整数列表数据。
- **参数:**

write\_data: 待发送的字节数据, string 型数据、integer 型数据或整数 list 型数据, 若为 integer 或整数 list 型数据, 会写入一个或多个整数数据, 数据值若超过 [0-255] 范围, 会自动按照内存类型转换或内存截断的方式强制转换至范围内。
- **返回值:**

integer 型数据, 成功发送的字节数据长度, 小于等于 0 时, 表示发送失败。
- **示例:**

```
tool_serial_write("hello world")
```

返回值: 成功发送的数据长度

```
tool_serial_write(255)
```

返回值: 成功发送的数据长度

```
tool_serial_write([255,0,10])
```

返回值: 成功发送的数据长度

### 3.33 tool\_modbus\_read\_registers 指令

```
tool_modbus_read_registers(slave, addr, len, time_out)
```

- **功能:**

该指令用于工具串口读 MODBUS-RTU 从站寄存器。

- **参数:**

slave: 从站, 范围为 1 - 247, integer 型数据;

addr: 地址, 范围为大于等于 0, integer 型数据;

len: 要读取数据长度, 范围为 1 - 125, 当 len 大于 1 时, 将会从 slave 的 addr 开始顺序向下读取, integer 型数据;

time\_out: 超时时间, 单位: 秒, 若小于等于 0, 则等待超时时间为无限长, float 型数据。

- **返回值:**

list 型数据, 读取的数据列表。

- **示例:**

```
read_data=tool_modbus_read_registers (1,1,1,0)
```

返回值: 读取的寄存器值;

### 3.34 tool\_modbus\_read\_bits 指令

```
tool_modbus_read_bits(slave, addr, len, time_out)
```

- **功能:**

该指令用于工具串口读 MODBUS-RTU 从站线圈。

- **参数:**

slave: 从站, 范围为 1 - 247, integer 型数据;

addr: 地址, 范围为大于等于 0, integer 型数据;

len: 要读取数据长度, 范围为 1 - 125, 当 len 大于 1 时, 将会从 slave 的 addr 开始顺序向下读取, integer 型数据;

time\_out: 超时时间, 单位: 秒, 若小于等于 0, 则等待超时时间为无限长, float 型数据。

- **返回值:**

list 型数据, 读取的数据列表。

- **示例:**

```
read_data=tool_modbus_read_bits (1,1,1,0)
```

返回值: 读取的线圈值;

### 3.35 tool\_modbus\_write\_registers 指令

```
tool_modbus_write_registers(slave, addr, data)
```

- **功能:**

该指令用于工具串口写 MODBUS-RTU 从站寄存器。

- **参数:**

slave: 从站, 范围为 1 - 247, integer 型数据;

addr: 地址, 范围为大于等于 0, integer 型数据;

data: 待写入的数据, integer、string、list 型数据 (列表的每个元素需为 integer)。当数据为 string 或 list 时, 将会向 slave 的 addr 地址处顺序向下写。

- **返回值:**

boolean 型数据, 写成功返回 True, 否则为 False。

- **示例:**

```
tool_modbus_write_registers (1,1,1)
```

```
返回值: True;
```

### 3.36 tool\_modbus\_write\_bits 指令

```
tool_modbus_write_bits(slave, addr, len, time_out)
```

- **功能:**

该指令用于工具串口写 MODBUS-RTU 从站线圈。

- **参数:**

slave: 从站, 范围为 1 - 247, integer 型数据;

addr: 地址, 范围为大于等于 0, integer 型数据;

data: 待写入的数据, integer、string、list 型数据 (列表的每个元素需为 integer)。当数据为 string 或 list 时, 将会向 slave 的 addr 地址处顺序向下写。当数据类型为 string 时, 字符串的内容必须为数字, 每个数字都会被转换成十进制的 09。注意, 所有的非 0 元素都会被隐式地转换为 1。

- **返回值:**

boolean 型数据, 写成功返回 True, 否则为 False。

- **示例:**

```
tool_modbus_write_bits (1,1,1)
```

```
返回值: True;
```

### 3.37 tool\_serial\_mode 指令

```
tool_serial_mode()
```

- **功能:**  
该指令用于获取末端串口处于哪种模式。
- **参数:** 无
- **返回值:**  
string 型数据, “MODBUS-RTU” 或 “RS485”, 若二者都未使能则为 “CLOSE”。
- **示例:**

```
tool_serial_mode()
```

```
返回值: “RS-485”
```

### 3.38 set\_tool\_voltage 指令

```
set_tool_voltage(voltage)
```

- **功能:**  
该指令用于设置机器人工具法兰的外部工具接头提供电源的电压, 电压可以为 0、12 或 24 伏。
- **参数:**  
voltage: 外部工具接头提供电源的电压, 单位: 伏, integer 型数据: 0、12 或 24。
- **返回值:** 无
- **示例:**

```
set_tool_voltage(24)
```

### 3.39 serial\_open 指令

```
serial_open(port="/dev/ttymx0")
```

- **功能:**

该指令用于打开指定的串口设备。
- **参数:**

port: 串口的端口名, 目前仅支持:" /dev/ttymx0" 。不指定名称时, 使用默认名称 "/dev/ttymx0" , string 型数据 (可选参数)。
- **返回值:**

如果打开成功, 返回 True, 如果打开失败, 返回 False, boolean 型数据。
- **示例:**

```
serial_open("/dev/ttymx0")
```

返回值: True 或 False

### 3.40 serial\_close 指令

```
serial_close(port="/dev/ttymx0")
```

- **功能:**

该指令用于关闭指定的串口设备。
- **参数:**

port: 串口的端口名, 目前仅支持:" /dev/ttymx0" 。不指定名称时, 使用默认名称 "/dev/ttymx0" 。如果想要关闭所有的串口可以使用" all" , string 型数据 (可选参数)。
- **返回值:**

如果关闭成功, 返回 True, 如果关闭失败, 返回 False, boolean 型数据。
- **示例:**

```
serial_close("/dev/ttymx0")
```

返回值: True 或 False

## 3.41 serial\_config 指令

```
serial_config(baud, parity_bit, stop_bit, size=8, port="/dev/ttymx0")
```

- **功能:**

该指令用于配置指定的串口设备。

- **参数:**

baud: 波特率, integer 型数据: 2400、4800、9600、19200、38400、57600、115200、230400、460800、921600、500000、1000000;

parity\_bit: 串口奇偶校验, integer 型数据: 0 (不校验)、1 (奇校验)、2 (偶校验);

stop\_bit: 串口停止位, integer 型数据: 1、2;

size: 串口的位宽。不指定时, 使用默认参数 8, integer 型数据: 8 或 9 (可选参数);

port: 串口的端口名, 目前仅支持: "/dev/ttymx0"。不指定名称时, 使用默认名称 "/dev/ttymx0", string 型数据 (可选参数)。

- **返回值:**

如果配置成功, 返回 True, 如果配置失败, 返回 False, boolean 型数据。

- **示例:**

```
serial_config(115200,0,1)
```

参数:

baud=115200

parity\_bit=0

stop\_bit=1

size=8, 默认位宽

port="/dev/ttymx0", 默认串口设备名称

返回值: True 或 False



## 3.42 serial\_read 指令

```
serial_read(len, is_number=True, time_out=0.1, port="/dev/ttymx0")
```

- **功能:**

该指令用于从指定的串口设备读取数据。

- **参数:**

len: 要读取的字节数, 注意假设要读取的字节数为 5, 但实际只接收到 2 字节数据, 最后读出的长度为 2 字节;

is\_number: 是否按照整数格式返回字节数据。若不指定, 使用默认参数 True, boolean 型数据 (可选参数);

time\_out: 超时时间, 单位: 秒, 若小于等于 0, 则等待超时时间为无限长, 不指定时, 使用默认值 0.1, float 型数据 (可选参数);

port: 串口的端口名, 目前仅支持: "/dev/ttymx0"。不指定名称时, 使用默认名称 "/dev/ttymx0", string 型数据 (可选参数)。

- **返回值:**

bytearray 或 list 型数据, 若 is\_number 参数 True, 则返回值为整数类型的 list, 若 is\_number 参数 False, 则返回值为 bytes array, 若发生超时, 则返回所有已经读取到的数据。

- **示例:**

```
serial_read(4)
参数:
len=4
is_number=True 默认值
time_out=0.1, 默认值
port="/dev/ttymx0", 默认串口设备名称
返回值: 读取数据的列表, [12,99,255,1]
serial_read(3,False, 0.1)
参数:
len=3
is_number=False
time_out=0.1
返回值: 读取的字节数据, 类似 b"a/x90/x33"
```

### 3.43 serial\_write 指令

```
serial_write(data, port="/dev/ttymx0")
```

- **功能:**

该指令用于向指定的串口设备写入数据。

- **参数:**

data: 待发送的数据, string、integer、float、list 型数据 (注: list 里数据类型只能为 integer)

port: 串口的端口名, 目前仅支持: "/dev/ttymx0"。不指定名称时, 使用默认名称 "/dev/ttymx0", string (可选参数)。

- **返回值:**

如果发送成功, 返回 True, 如果发送失败, 返回 False, boolean 型数据。

- **示例:**

```
serial_write([123,56,444])
```

参数:

data=[123,56,444], 其中 444 这个数将会被拆成 188,1,0,0

port="/dev/ttymx0", 默认串口设备名称

返回值: True 或 False

### 3.44 serial\_is\_open 指令

```
serial_is_open(port="/dev/ttymx0")
```

- **功能:**

该指令用于判断指定的串口设备是否已经打开。

- **参数:**

port: 串口的端口名, 目前仅支持: "/dev/ttymx0"。不指定名称时, 使用默认名称 "/dev/ttymx0", string 型数据 (可选参数)。

- **返回值:**

如果打开, 返回 True, 如果没打开, 返回 False, boolean 型数据。

- **示例:**

```
serial_is_open()
```

参数:

port="/dev/ttymx0", 默认串口设备名称

返回值: True 或 False

## 3.45 serial\_open\_list 指令

```
serial_open_list()
```

- **功能:**  
该指令用于返回所有已经打开的串口端口名称列表。
- **参数:** 无
- **返回值:**  
list 型数据, 已打开串口名称列表。
- **示例:**

```
serial_open_list()
```

```
返回值: ['/dev/ttymx0 ', '/dev/ttyUSB0 ']
```

## 4 总线指令

### 4.1 Modbus

#### 4.1.1 modbus\_add\_signal 指令

```
modbus_add_signal(IP, slave_number, signal_address, signal_type, signal_name)
```

- **功能:**

该指令用于为控制器添加一个新的 modbus 信号用于监督; 预期无响应。

- **参数:**

IP: 用于指定 modbus 信号所接 modbus 装置的 IP 地址。string 型数据;

slave\_number: 通常设置为 255, 可在 [0: 247]之间自由选择, 设置为 0 时为广播, 设置为 [248: 254] 之间时会被设置为 255。Integer 型数据: [0: 247, 255]

signal\_address: 用于指定此新信号应当反映的线圈或寄存器的地址。有关此信息, 请查阅 modbus 装置的配置。integer 型数据: [0: 65535];

signal\_type: 用于指定要添加的信号类型。0=数字输入, 1=数字输出, 2 = 寄存器输入, 3 = 寄存器输出。integer 型数据: [0: 3];

signal\_name: 专用于识别信号。如果提供一个与已添加的信号相同的字符串, 新的信号将代替旧的信号。string 型数据。

- **返回值:** 无

- **示例:**

```
modbus_add_signal("172.140.17.11",255,5,1,"output1")
```

## 4.1.2 modbus\_delete\_signal 指令

```
modbus_delete_signal(signal_name)
```

- **功能:**  
该指令用于删除所提供的信号名称识别的信号。
- **参数:**  
singal\_name: 信号名, 与应当删除的信号名称相同。string 型数据。
- **返回值:** 无
- **示例:**

```
modbus_delete_signal("output1")
```

## 4.1.3 modbus\_get\_signal\_status 指令

```
modbus_get_signal_status(signal_name)
```

- **功能:**  
该指令用于读取特定信号的当前值。
- **参数:**  
singal\_name: 信号名, 与应当获取数值的信号名称相同。string 型数据。
- **返回值:**  
对于数字信号: True 或 False, boolean 型数据;  
对于寄存器信号: 以无符号整数显示的寄存器数值, integer 型数据。
- **示例:**

```
modbus_get_signal_status("output1")  
返回值: True, False 或者一个无符号整数
```

### 4.1.4 modbus\_send\_custom\_command 指令

```
modbus_send_custom_command(IP, slave_number, function_code, data)
```

- **功能:**

该指令用于将用户指定的命令发送到位于指定 IP 地址的 modbus 装置。无法用于索取数据，因为不会收到响应。用户负责提供对所提供的函数代码有意义的的数据。内建函数注意构建 modbus 框架，因此用户无需担心命令的长度。

- **参数:**

IP: 用于指定自定义命令应当发送到的 modbus 装置的 IP 地址。string 型数据;

slave\_number: 通常不使用，并且通常设为 255，可在 [0: 247]之间自由选择，设置为 0 时为广播，设置为 [248: 254] 之间时会被设置为 255。Integer 型数据: [0: 247, 255]

function\_code: 用于指定自定义命令所用的函数代码,目前只支持 6(设置单个寄存器)。integer 型数据;

data: 一个整数列表，每个条目必须是有效的字节 [0:255] 值。integer list 型数据。

- **返回值:** 无

- **示例:**

```
modbus_send_custom_command ("172.140.17.11",103,6,[17,32,2,88])
```

说明: Backhoff BK9050 上的电子狗超时设为 600ms。

这是如下所述完成的: 使用 modbus 函数代码 6 (预设单个寄存器)，然后在数据阵列的前两个字节 ([17,32]=[0x1120]) 提供寄存器地址，并且在后两个字节 ([2,88]=[0x0258]=dec 600) 提供所需寄存器内容。

### 4.1.5 modbus\_set\_output\_register 指令

```
modbus_set_output_register(signal_name, register_value)
```

- **功能:**

该指令用于将既定名称识别的输出寄存器信号设为既定值。

- **参数:**

signal\_name: 信号名，用于识别已事先添加的输出寄存器信号。string 型数据;

register\_value: 必须是有效的字符值。若超出范围，会自动按照内存类型转换或内存截断的方式强制转换至范围内，integer 型数据: [0: 65535]。

- **返回值:** 无

- **示例:**

```
modbus_set_output_register("output1",300)
```

参数:

```
signal_name="output1"
```

```
register_value=300
```

### 4.1.6 modbus\_set\_output\_signal 指令

```
modbus_set_output_signal(signal_name, digital_value)
```

- **功能:**  
该指令用于将既定名称识别的输出数字信号设为既定值。
- **参数:**  
signal\_name: 信号名, 用于识别已事先添加的输出数字信号。string 型数据;  
digital\_value: 信号将被设置为此值。boolean 型数据。
- **返回值:** 无
- **示例:**

```
modbus_set_output_signal("output2", True)
```

### 4.1.7 modbus\_set\_runstate\_dependent\_choice 指令

```
modbus_set_runstate_dependent_choice(signal_name, runstate_choice)
```

- **功能:**  
该指令用于设置输出信号是必须保持来自程序的状态, 还是程序不运行时设高或设低。
- **参数:**  
signal\_name: 信号名, 用于识别已事先添加的输出数字信号。string 型数据;  
runstate\_choice: 0=保持程序状态; 1=程序不运行时设低; 2=程序不运行时设高; 3=程序停止时输出低电平, 程序运行时输出高电平。integer 型数据: [0: 3]。
- **返回值:** 无
- **示例:**

```
modbus_set_runstate_dependent_choice("output2", 1)
```

### 4.1.8 modbus\_set\_signal\_update\_frequency 指令

```
modbus_set_signal_update_frequency(signal_name, update_frequency)
```

- **功能:**  
该指令用于设置机器人将请求发送给 Modbus 控制器读取或写入信号值的频率。
- **参数:**  
signal\_name: 信号名, 用于识别已事先添加的输出数字信号。string 型数据;  
update\_frequency: 用于指定更新频率, 以 Hz 为单位。integer 型数据: [0: 250]。
- **返回值:** 无
- **示例:**

```
modbus_set_signal_update_frequency("output2", 20)
```

### 4.1.9 read\_port\_bit 指令

```
read_port_bit(address)
```

- **功能:**

该指令用于读取本机 Modbus 从站数字（线圈）端口的值。

- **参数:**

address: 端口的地址 (请参阅支持网站“使用 Modbus 服务器”页面上的端口映射器)。integer 型数据: [0:127]。

- **返回值:**

端口保存的数值, boolean 型数据。

- **示例:**

```
boolval=read_port_bit(3)  
返回值: True 或者 False
```

### 4.1.10 read\_port\_register 指令

```
read_port_register(address)
```

- **功能:**

该指令用于读取 Modbus 客户端也可访问的其中一个端口 (读取从站寄存器)。

- **参数:**

address: 端口的地址, integer 型数据。

- **返回值:**

integer 型数据, 端口保存的有符号整数值[-32768:32767]或[0:65535]。

- **示例:**

```
intval=read_port_register(3)  
返回值: 有符号整数
```



### 4.1.11 write\_port\_bit 指令

```
write_port_bit(address, value)
```

- **功能:**

该指令用于写入 Modbus 客户端也可访问的其中一个端口 (写从站线圈)。

- **参数:**

address: 端口的地址 (请参阅支持网站“使用 Modbus 服务器”页面上的端口映射器) integer 型数据;

value: 寄存器中要设置的数值。boolean 型数据。

- **返回值:** 无

- **示例:**

```
write_port_bit(3,True)
```

### 4.1.12 write\_port\_register 指令

```
write_port_register(address, value)
```

- **功能:**

该指令用于写入 Modbus 客户端也可访问的其中一个端口 (写从站寄存器)。

- **参数:**

address: 端口的地址。integer 型数据;

value: 端口中要设置的数值 (0 : 65536) 或 (-32768 : 32767)。integer 型数据。

- **返回值:** 无

- **示例:**

```
write_port_register(3,100)
```

### 4.1.13 read\_input\_boolean\_register 指令

```
read_input_boolean_register(address)
```

- **功能:**

该指令用于从输入寄存器中读取布尔值,也可以通过现场总线访问。该寄存器存储在系统内存空间。

- **参数:**

address: 寄存器地址。integer 型数据: [0: 127]。

- **返回值:**

boolean 型数据, 选定输入寄存器的值。

- **示例:**

```
read_input_boolean_register(88)
```

返回值: True 或者 False

### 4.1.14 read\_input\_float\_register 指令

```
read_input_float_register(address)
```

- **功能:**

该指令用于从输入寄存器中读取浮点数,也可以通过现场总线访问。该寄存器存储在系统内存空间。

- **参数:**

address: 寄存器地址。integer 型数据: [0: 47]。

- **返回值:**

float 型数据, 选定输入寄存器的值。

- **示例:**

```
read_input_float_register(2)
```

返回值: 浮点数

### 4.1.15 read\_input\_integer\_register 指令

```
read_input_integer_register(address)
```

- **功能:**

该指令用于从输入寄存器中读取整数,也可以通过现场总线访问。该寄存器存储在系统内存空间。

- **参数:**

address: 寄存器地址。integer 型数据: [0: 47]。

- **返回值:**

integer 型数据, 选定输入寄存器的值。

- **示例:**

```
read_input_integer_register(2)
```

返回值: 整数

### 4.1.16 read\_output\_boolean\_register 指令

```
read_output_boolean_register(address)
```

- **功能:**

该指令用于从输出寄存器中读取布尔值,也可以通过现场总线访问。该寄存器存储在系统内存空间。

- **参数:**

address: 寄存器地址。integer 型数据: [0: 127]。

- **返回值:**

boolean 型数据, 选定输出寄存器的值。

- **示例:**

```
read_output_boolean_register(88)
```

返回值: True 或者 False

### 4.1.17 read\_output\_float\_register 指令

```
read_output_float_register(address)
```

- **功能:**

该指令用于从输出寄存器中读取浮点数,也可以通过现场总线访问。该寄存器存储在系统内存空间。

- **参数:**

address: 寄存器地址。integer 型数据: [0: 47]。

- **返回值:**

float 型数据, 选定输出寄存器的值。

- **示例:**

```
read_output_float_register(2)
```

返回值: 浮点数

### 4.1.18 read\_output\_integer\_register 指令

```
read_output_integer_register(address)
```

- **功能:**

该指令用于从输出寄存器中读取整数,也可以通过现场总线访问。该寄存器存储在系统内存空间。

- **参数:**

address: 寄存器地址。integer 型数据: [0: 47]。

- **返回值:**

integer 型数据, 选定输出寄存器的值。

- **示例:**

```
read_output_integer_register(2)
```

返回值: 整数

### 4.1.19 write\_output\_boolean\_register 指令

```
write_output_boolean_register(address, value)
```

- **功能:**

该指令用于将布尔值写入一个输出寄存器,也可以通过现场总线访问。该寄存器存储在系统内存空间。

- **参数:**

address: 寄存器地址。integer 型数据: [0: 127];

value: 需要写入寄存器的值, boolean 型数据。

- **返回值:** 无

- **示例:**

```
write_output_boolean_register(3,True)
```

### 4.1.20 write\_output\_float\_register 指令

```
write_output_float_register(address, value)
```

- **功能:**

该指令用于将浮点数写入一个输出寄存器,也可以通过现场总线访问。该寄存器存储在系统内存空间。

- **参数:**

address: 寄存器地址。integer 型数据: [0: 47];

value: 需要写入寄存器的值。float 型数据。

- **返回值:** 无

- **示例:**

```
write_output_float_register(3,37.68)
```

### 4.1.21 write\_output\_integer\_register 指令

```
write_output_integer_register(address, value)
```

- **功能:**

该指令用于将整数写入一个输出寄存器,也可以通过现场总线访问。该寄存器存储在系统内存空间。

- **参数:**

address: 寄存器地址。integer 型数据: [0: 47];

value: 需要写入寄存器的值。integer 型数据。

- **返回值:** 无

- **示例:**

```
write_output_integer_register(3,37)
```

## 5 通讯指令

### 5.1 SOCKET

#### 5.1.1 socket\_open 指令

```
socket_open(address, port, socket_name="socket_0")
```

- **功能:**

该指令用于建立 TCP/IP 网络通信, 连接超时时间为 2 秒。

- **参数:**

address: 服务器地址, string 型数据;

port: 端口号, integer 型数据;

socket\_name: 服务器名称, 不指定名称或指定的名称为空时, 使用默认名称 "socket\_0", string 型数据 (可选参数)。

- **返回值:**

boolean 型数据, 如果连接失败, 返回 False, 如果连接成功, 返回 True。

- **示例:**

```
socket_open("192.168.1.119", 23333, "socket_1")
```

返回值: True 或 False

#### 5.1.2 socket\_close 指令

```
socket_close(socket_name="socket_0")
```

- **功能:**

该指令用于关闭 TCP/IP 网络通信。

- **参数:**

socket\_name: 服务器名称, 不指定名称或指定的名称为空时, 使用默认名称 "socket\_0", string 型数据 (可选参数)。

- **返回值:** 无

- **示例:**

```
socket_close("socket_3")
```

### 5.1.3 socket\_get\_var 指令

```
socket_get_var(var_name, socket_name="socket_0")
```

- **功能:**

该指令用于从服务器读取一个整数数据;

该接口会向指定服务器发送“GET <var\_name>\n”消息,并等待服务器回复格式为“<var\_name><int>\n”的内容,等待回复的超时时间为 2 秒,若发生超时或回复格式不满足协议需求,返回 0。

- **参数:**

var\_name: 变量名称, string 型数据;

socket\_name: 服务器名称,不指定名称或指定的名称为空时,使用默认名称“socket\_0”, string 型数据(可选参数)。

- **返回值:**

integer 型数据,服务器返回的整数值,若为 0,则可能是发生了错误(除非服务器本身为 0),返回值范围为[-2147483648,2147483647],超过该范围会返回最大值或最小值。

- **示例:**

```
joint_0_pos=socket_get_var("JOINT_0")
```

说明:发送字符串“GET JOINT\_0\n”至服务器 socket\_0,并等待服务器回复格式类似“JOINT\_0 1000\n”的内容,等待超时时间为 2 秒

返回值:从服务器获取的整数值



## 5.1.4 socket\_read\_ascii\_float 指令

```
socket_read_ascii_float(number, socket_name="socket_0", timeout=2)
```

- **功能:**

该指令用于从连接的 TCP/IP 服务器读取多个 ASCII 格式的 float 型数据;

服务器传输过来的数据需要使用括号括起来, 数据之间使用逗号 “,” 或者空格 “ ” 分开, 类似 “(3.14,2.562,8.24,3.4434)”;

如果传输过来的数据格式类似 “(3.14,,,2.562,,8.24,3.4434)”, 仍会正确读取四个数据。

- **参数:**

number: 需要读取的数据个数, integer 型数据: [1:30];

socket\_name: 服务器名称, 不指定名称或指定的名称为空时, 使用默认名称 “socket\_0”, string 型数据 (可选参数);

timeout: 超时时间, 单位: 秒, 若小于等于 0, 则等待超时时间为无限长, float 型数据 (可选参数)。

- **返回值:**

list 型数据, 读取的数据列表, 列表的第一个元素为所有成功读取的数据个数, 第二个元素开始为读取的 float 型数据;

若读取失败, 则列表第一个元素为 0, 其他元素为 nan, 类似[0,nan,nan];

若读取成功, 会返回数据列表, 内容类似[3,1.444,2.234,3.444];

若接收的数据长度不满足 <number> 的数量, 返回的数据列表包括成功读取的数据长度以及数据, 未读取到的数据为 nan, [2,1.444,2.234,nan]。

- **示例:**

```
list_of_three_floats=socket_read_ascii_float(3)
```

参数:

number=3

socket\_name=socket\_0, 默认名称

timeout=2, 默认超时时间

返回值: 读取的数据列表, [3,3.12,3.11,8.24]、[2,3.12,3.11,nan]或[0,nan,nan,nan]

### 5.1.5 socket\_read\_binary\_integer 指令

```
socket_read_binary_integer(number, socket_name="socket_0", timeout=2)
```

- **功能:**

该指令用于从连接的 TCP/IP 服务器读取多个 32 位整数数据，数据格式为网络字节序 (networkbyte order)。

- **参数:**

number: 需要读取的数据个数, integer 型数据: [1:30];

socket\_name: 服务器名称, 不指定名称或指定的名称为空时, 使用默认名称 "socket\_0", string 型数据 (可选参数);

timeout: 超时时间, 单位: 秒, 若小于等于 0, 则等待超时时间为无限长, float 型数据 (可选参数)。

- **返回值:**

list 型数据, 读取的数据列表, 列表的第一个元素为成功读取的数据个数, 第二个元素开始为读取的 integer 数据;

若读取失败, 则列表第一个元素为 0, 其他元素为-1, 类似 [0,-1,-1,-1];

若读取成功, 会返回数据列表, 内容类似 [2,100,200];

若接收的数据长度不满足 <number> 的数量, 返回的数据列表包括成功读取的数据长度以及数据, 未读取到的数据为-1, [2,100,200,-1]。

- **示例:**

```
list_of_three_ints=socket_read_binary_integer(3)
```

参数:

number=3

socket\_name=socket\_0, 默认名称

timeout=2, 默认超时时间

返回值: 读取的数据列表, [3,100,200,300]或[0,-1,-1,-1]

## 5.1.6 socket\_read\_byte\_list 指令

```
socket_read_byte_list(number, socket_name="socket_0", timeout=2)
```

- **功能:**

该指令用于从连接的 TCP/IP 服务器读取多个字节数据。

- **参数:**

number: 需要读取的数据个数, integer 型数据: [1:30];

socket\_name: 服务器名称, 不指定名称或指定的名称为空时, 使用默认名称 "socket\_0", string 型数据 (可选参数);

timeout: 超时时间, 单位: 秒, 若小于等于 0, 则等待超时时间为无限长, float 型数据 (可选参数)。

- **返回值:**

list 型数据, 读取的数据列表, 列表的第一个元素为成功读取的数据个数, 第二个元素开始为读取的字节数据;

若读取失败, 则列表第一个元素为 0, 其他元素为-1, 类似 [0,-1,-1,-1];

若读取成功, 会返回数据列表, 内容类似 [2,100,200];

若接收的数据长度不满足 <number> 的数量, 返回的数据列表包括成功读取的数据长度以及数据, 未读取到的数据为-1, [2,100,200,-1]。

- **示例:**

```
list_of_three_byts=socket_read_byte_list(3)
```

参数:

number=3

socket\_name=socket\_0, 默认名称

timeout=2, 默认超时时间

返回值: 读取的数据列表, [3,100,200,123]或[0,-1,-1,-1]

## 5.1.7 socket\_read\_string 指令

```
socket_read_string(socket_name="socket_0", prefix="", suffix="", timeout=2)
```

- **功能:**

该指令用于从连接的 TCP/IP 服务器读取全部数据，并根据 <prefix> 与 <suffix> 的设置，返回符合规则的字符串数据。

- **参数:**

socket\_name: 服务器名称, 不指定名称或指定的名称为空时, 使用默认名称 "socket\_0", string 型数据 (可选参数);

prefix: 字符串匹配前缀, string 型数据 (可选参数);

suffix: 字符串匹配后缀, string 型数据 (可选参数);

timeout: 超时时间, 单位: 秒, 若小于等于 0, 则等待超时时间为无限长, float 型数据 (可选参数)。

- **返回值:**

string 型数据, 读取到的字符串变量。未指定 prefix 和 suffix 时, 返回读取的全部字符串数据; 只指定了 prefix 时, 会返回 prefix 后所有读取到的字符串数据, 不包括 prefix 字符串本身; 只指定了 suffix 时, 会返回 suffix 前所有读取到的字符串数据, 不包括 suffix 字符串本身。prefix 和 suffix 都指定了时, 返回 prefix 和 suffix 之间的字符串, 不包括 prefix 和 suffix 字符串本身。本次未被读取的数据, 会保留在 socket 中, 可以进行循环读取。读取超时, 返回空字符串。

- **示例:**

```
while(True):  
    string_from_server=socket_read_string(prefix="<",suffix=">")  
    if(!string_from_server)  
        break  
    textmsg(string_from_server)
```

说明: 假设 socket\_0 服务器发送的字符串为 "<111><222><333><444>"

返回值: 会依次返回 "111"、"222"、"333"、"444"

- **注意:**

可以使用 socket\_read\_string(suffix=" \n" ) 脚本, 实现对服务器的字符串数据按行读取;

该指令单次获取的最大数据量为 2047。

## 5.1.8 socket\_send\_byte 指令

```
socket_send_byte(value, socket_name="socket_0")
```

- **功能:**

该指令用于向连接的 TCP/IP 服务器发送一个字节数据，可以用于发送 ASCII 字符数据：如 10 代表“\n”。

- **参数:**

value: 需要发送的字节数据，string 型数据或 integer 型数据，若为 string 型数据，长度不能超过 1，若为 integer 型数据，范围若超过[0-255]，会自动按照内存类型转换或内存截断的方式强制转换至范围内；

socket\_name: 服务器名称，不指定名称或指定的名称为空时，使用默认名称“socket\_0”，string 型数据（可选参数）。

- **返回值:**

boolean 型数据，如果发送成功，返回 True，如果发送失败，返回 False。

- **示例:**

```
socket_send_byte(10)
```

参数:

value=10, ASCII 字符数据，10 代表“\n”

socket\_name=socket\_0, 默认名称

返回值: True 或 False

```
socket_send_byte("a")
```

参数:

value="a", string 字符数据

socket\_name=socket\_0, 默认名称

返回值: True 或 False

### 5.1.9 socket\_send\_int 指令

```
socket_send_int(value, socket_name="socket_0")
```

- **功能:**

该指令用于向连接的 TCP/IP 服务器发送整数 (int32\_t) 数据, 发送格式为网络字节序 (network byte order)。

- **参数:**

value: 需要发送的字节数据, integer 型数据;

socket\_name: 服务器名称, 不指定名称或指定的名称为空时, 使用默认名称 "socket\_0", string 型数据 (可选参数)。

- **返回值:**

boolean 型数据, 如果发送成功, 返回 True, 如果发送失败, 返回 False。

- **示例:**

```
socket_send_int(10)
```

返回值: True 或 False

### 5.1.10 socket\_send\_line 指令

```
socket_send_line(str, socket_name="socket_0")
```

- **功能:**

该指令用于向连接的 TCP/IP 服务器以 ASCII 编码发送字符串数据, 发送时会在字符串数据 <str> 的末尾增加 " \n" 换行符。

- **参数:**

str: 要发送的字符串, string 型数据;

socket\_name: 服务器名称, 不指定名称或指定的名称为空时, 使用默认名称 "socket\_0", string 型数据 (可选参数)。

- **返回值:**

boolean 型数据, 如果发送成功, 返回 True, 如果发送失败, 返回 False。

- **示例:**

```
socket_send_line("Hello world")
```

说明: 发送 "Hello world\n" 至服务器 socket\_0

返回值: True 或 False

### 5.1.11 socket\_send\_string 指令

```
socket_send_string(str, socket_name="socket_0")
```

- **功能:**

该指令用于向连接的 TCP/IP 服务器以 ASCII 编码发送字符串数据。

- **参数:**

str: 支持 Python 内置数据类型: string、list、boolean、integer、float 等数据类型;

socket\_name: 服务器名称, 不指定名称或指定的名称为空时, 使用默认名称 "socket\_0", string 型数据 (可选参数)。

- **返回值:**

boolean 型数据, 如果发送成功, 返回 True, 如果发送失败, 返回 False。

- **示例:**

```
socket_send_string("Hello world")
```

说明: 发送“Hello world”至服务器 socket\_0

返回值: True 或 False

### 5.1.12 socket\_set\_var 指令

```
socket_set_var(name, value, socket_name="socket_0")
```

- **功能:**

该指令用于向连接的 TCP/IP 服务器发送内容为 “SET <name> <value>\n” 的数据, 用于实现设置服务器整型变量的值的功能。

- **参数:**

name: 变量名称, string 型数据;

value: 变量值, integer 型数据;

socket\_name: 服务器名称, 不指定名称或指定的名称为空时, 使用默认名称 "socket\_0", string 型数据 (可选参数)。

- **返回值:** 无

- **示例:**

```
socket_set_var("JOINT_0",1000)
```

说明: 发送字符串“SET JOINT\_0 1000\n”至服务器 socket\_0

## 5.2 RPC

### 5.2.1 rpc\_factory 指令

```
rpc_factory(type, url)
```

- **功能:**

该指令用于创建一个远程调用 (RPC) 句柄, 用于实现远程调用功能。

- **参数:**

type: RPC 类型, 目前仅支持 “xmlrpc” 类型, string 型数据;

url : RPC 服务器地址, 格式为 “<ip-address>:<port>” 或 “http://<ip address>:<port>/<path>”, 如 “127.0.0.1:5050” 或 “http://127.0.0.1:5050/RPC2”, string 型数据。

- **返回值:**

Object, RPC 句柄。

- **示例:**

```
camera=rpc_factory("xmlrpc","http://127.0.0.1:5050/RPC2")  
camera.capture()
```

说明: 创建相机的 RPC 远程调用句柄, 并调用相机的方法进行拍照操作  
返回值: RPC 句柄



## 6 系统指令

### 6.1 线程

#### 6.1.1 start\_thread 指令

```
start_thread(fundef, arg)
```

- **功能:**  
该指令用于启动线程。
- **参数:**  
fundef: 定义的函数, function arg: 传入 fundef 的参数, tuple。
- **返回值:**  
线程 ID。
- **示例:**

```
def thread_test(a,b):  
    sum=a+b;  
    for i in range(0,5):  
        if i \% 2==0:  
            sum +=sum  
    return sum  
global test_thread_id  
test_thread_id=start_thread(thread_test, (1,2))  
参数:  
fundef=thread_test  
返回值: 线程 ID
```

#### 6.1.2 stop\_thread 指令

```
stop_thread(thread_id)
```

- **功能:**  
该指令用于关闭线程; 线程外关闭线程。
- **参数:**  
thread\_id: 启动线程返回的线程 ID。
- **返回值:** 无
- **示例:**

```
id=start_thread(thread_test,(1,2))  
stop_thread(id)
```

## 6.2 提示警告

### 6.2.1 textmsg 指令

```
textmsg(s1, s2='')
```

- **功能:**

该指令用于发送 s1 和 s2 连接后的字符串到 EliRobot, 并显示在 EliRobot 的日志栏中。

- **参数:**

s1: 消息字符串, 也可发送其他类型的变量, integer,boolean,list 型数据等;

s2: 消息字符串, 也可发送其他类型的变量, integer,boolean,list 型数据等 (可选参数, 默认为空字符串)。

- **返回值:** 无

- **示例:**

```
textmsg("value=",3)
```

- **注意:**

不建议高频率并且大量的调用该脚本函数, 否则可能导致软件性能问题。

### 6.2.2 popup 指令

```
popup(s,title='Popup', warning=False, error=False, blocking=False)
```

- **功能:**

该指令用于弹出窗口并显示消息。

- **参数:**

s: 消息字符串, string 型数据;

title: 标题字符串, string 型数据;

warning: 警告消息, boolean 型数据;

error: 错误消息, boolean 型数据;

blocking: 等待, boolean 型数据。

- **返回值:** 无

- **示例:**

```
popup(s="here I am",title="Popup#1",blocking=True)
```

## 6.3 字符

### 6.3.1 str\_at 指令

```
str_at(src, index)
```

- **功能:**

该指令用于提供对字符串字节的直接访问。此接口返回一个字符串，该字符串包含源字符串中与指定索引对应的位置处的字符。

- **参数:**

src: 源字符串, string 型数据;

index: 指定字符串中的位置, integer 型数据。

- **返回值:**

string 型数据, 索引号对应的字符, 如果索引无效, 则会引发异常。

- **示例:**

```
str_at("Hello",0)
返回值: 'H'
str_at("Hello",1)
返回值: 'e'
str_at("Hello",10)
返回值: 报警索引超出范围
str_at("",0)
返回值: 给定的源字符串为空
```

### 6.3.2 str\_cat 指令

```
str_cat(object1, object2)
```

- **功能:**

该指令用于实现将输入的两个数据按照一定规则转换为字符串并进行连接,输入的两个数据可以为任意 Python 内置数据类型。其中浮点数类型将被格式化为 6 位小数,并删除末尾无效的 0。连接后的字符串最大长度为 1023 个字符,超过该长度会触发运行时异常。

- **参数:**

object1: 第一个数据, string、boolean、integer、float、list 型数据;

object2: 第二个数据, string、boolean、integer、float、list 型数据。

- **返回值:**

string 型数据, 拼接后的字符串。

- **示例:**

```
str_cat("Hello"," World!")
返回值: "Hello World!"
str_cat("Integer ",1)
返回值: "Integer 1"
str_cat("",[1.0,2.0,3.0,4.0,5.0,6.0])
返回值: "[1,2,3,4,5,6]"
str_cat([True,False, True],[1,0,1])
返回值: "[True,False, True][1,0,1]"
str_cat(str_cat("",str_cat("11","22")),str_cat("Three","Four"))
返回值: "1122ThreeFour"
```

### 6.3.3 str\_empty 指令

```
str_empty(str)
```

- **功能:**

该指令用于 str 为空时返回 true, 否则返回 false。

- **参数:**

str: 源字符串, string 型数据。

- **返回值:**

boolean 型数据, 如果字符串为空则为 True, 否则为 False。

- **示例:**

```
str_empty("")
返回值: True
str_empty("Hello")
返回值: False
```

### 6.3.4 str\_find 指令

```
str_find(src, target, start_from=0)
```

- **功能:**

该指令用于在 src 中查找子字符串目标的第一个匹配项，字符串为零索引。

- **参数:**

src: 源字符串, string 型数据;

target: 要搜索的子字符串, string 型数据;

start\_from: 可选起始位置 (默认为 0), integer 型数据。

- **返回值:**

integer 型数据, 如果在 src 中找到了目标字符串, 则返回目标字符串的位置。如果 src 中找不到目标字符串, 则为-1。

- **示例:**

```
str_find("Hello World!","o")  
返回值: 4  
str_find("Hello World!","lo")  
返回值: 3  
str_find("Hello World!","o",5)  
返回值: 7  
str_find("abc","z")  
返回值: -1
```

### 6.3.5 str\_len 指令

```
str_len(str)
```

- **功能:**

该指令用于返回字符串中的字节数。

- **参数:**

str: 源字符串, string 型数据。

- **返回值:**

integer 型数据, 输入字符串中的字节数。

- **示例:**

```
str_len("Hello")  
返回值: 5  
str_len("")  
返回值: 0
```

### 6.3.6 str\_sub 指令

```
str_sub(src, index, len)
```

- **功能:**

该指令用于返回 src 的子字符串。结果是 src 的子字符串，从 index 指定的字节开始，长度最多为 len 字节。如果请求的子字符串超出了原始字符串的末尾（即  $index + len > src \text{ length}$ ），则结果子字符串的长度将限制为 src 的大小。如果 index 或 len 不在范围内，则引发异常。字符串为零索引。

- **参数:**

src: 源字符串，string 型数据；

index: 整型值，指定范围  $[0, src \text{ length}]$  中的初始字节，integer 型数据；

len: 子字符串的长度。如果未指定 len，则为范围  $[索引, src \text{ 长度}]$  中的字符串，integer 型数据（可选参数）。

- **返回值:**

string 型数据，src 中从索引开始并截取 len 个字符的部分。

- **示例:**

```
str_sub("0123456789abcdefghij",2,0)
返回值: "" (len 是 0)
str_sub("abcde",2,50)
返回值: "cde"
str_sub("abcde",-5,50)
返回值: 报警索引超出范围
str_sub("0123456789abcdefghij",5,3)
返回值:"567"
str_sub("0123456789abcdefghij",10)
返回值:"abcdefghij"
```

### 6.3.7 to\_num 指令

to\_num(str)

- **功能:**

该指令用于将字符串转换为整型或浮点型数字。‘.’ 小数点是区分二者的关键,但注意的是科学计数法默认为浮点型。合法的字符串包含可选的开头“+”/“-”号后跟上下列字串(不区分大小写):

(1)由十进制数字组成的十进制数(如,40),拥有‘.’的浮点数(如,10.1,-2.,.3),采用科学计数法表示的浮点数(如,10E1,1.5E4)。

(2)以‘0X’开头的十六进制数(如,0XABC,0x4a)。

(3)用“INF”或“INFINITY”代表的无穷大。

(4)用“NAN”代表的无意义数。

如果字符串所代表的数字过大将会出现异常(如,1.18973e+4932)。

- **参数:**

str: 要转换的字符串, string 型数据。

- **返回值:**

float 型数据或 integer 型数据, 字符串所代表的数值。

- **示例:**

```
to_num("10")
返回值: 10, integer
to_num("0xce110")
返回值: 844048, integer
to_num("3.14")
返回值: 3.14, float 型数据
to_num("NAN")
返回值: nan
to_num("123abc")
返回值: 报警非法的字符串输入
```

### 6.3.8 to\_str 指令

to\_str(val)

- **功能:**

该指令用于获取值的字符串表示形式;

结果字符串不能超过 1023 个字符。浮点数的格式为 6 位小数, 尾随的 0 将被移除。

- **参数:**

val: 要转换的值, float 型数据或 integer 型数据。

- **返回值:**

string 型数据, 给定值的字符串表示形式。

- **示例:**

```
to_str(10)
返回值: "10"
to_str(3.123456123456)
返回值: "3.123456"
to_str([1.0,2.0,3.0,4.0,5.0,6.0])
返回值: "[1,2,3,4,5,6]"
to_str([True,False, True])
返回值: "[True,False, True]"
```



## 6.4 其他

### 6.4.1 sleep 指令

```
sleep(t)
```

- **功能:**  
该指令用于休眠一段时间。
- **参数:**  
t: 时间[s],float 型数据。
- **返回值:** 无
- **示例:**

```
sleep(0.5)
```

### 6.4.2 set\_flag 指令

```
set_flag(n, b)
```

- **功能:**  
该指令用于设置系统内部标记位的值，类似虚拟数字输出 IO，用于在不同程序运行时保存数据信息。
- **参数:**  
n: 标记的索引, integer 型数据: [0:31];  
b: 标记值, boolean 型数据。
- **返回值:** 无
- **示例:**

```
set_flag(1,True)
```

### 6.4.3 get\_flag 指令

```
get_flag(n)
```

- **功能:**

该指令用于获取系统内部标记位的值，类似虚拟数字输出 IO，用于在不同程序运行时保存数据信息。

- **参数:**

n: 标记的索引, integer 型数据: [0:31]。

- **返回值:**

boolean 型数据, 内部标记位的值。

- **示例:**

```
get_flag(0)
```

返回值: 内部标记位的值, True 或 False

### 6.4.4 get\_steptime 指令

```
get_steptime()
```

- **功能:**

该指令用于返回机器人时间步长的持续时间 (以秒为单位)。在每个时间步长中, 机器人控制器会从机器人接收测量的关节位置和速度, 并将所需的关节位置和速度发送回机器人。上述过程以预定的频率在有规律的间隔发生。该间隔长度是机器人的时间步长。

- **参数:** 无

- **返回值:**

float 型数据, 机器人步长的持续时间。

- **示例:**

```
get_steptime()
```

返回值: 机器人步长的持续时间





ELITE ROBOTS  
艾利特机器人

# ALWAYS EASIER THAN BEFORE

[www.elibot.com](http://www.elibot.com)



公众号



视频号



抖音



B站

## 联系我们

商务合作: [market@elibot.cn](mailto:market@elibot.cn)

技术咨询: [tech@elibot.cn](mailto:tech@elibot.cn)

### 上海艾利特机器人有限公司

上海市浦东新区张江科学城学林路36弄18号楼

### 苏州艾利特机器人有限公司

苏州市工业园区长阳街259号中新钟园工业坊4栋1F

### 北京艾利特科技有限公司

北京市经济技术开发区荣华南路2号院6号楼1102室

### 德国分公司

Hersbrucker Weg 5, 85290, Geisenfeld, Germany

### 美国分公司

10521 Research Dr., Suite 104, 37932, Knoxville (TN), United States

### 日本分公司

愛知県名古屋市中村区名駅4-4-38 愛知県産業労働センター18FJetro