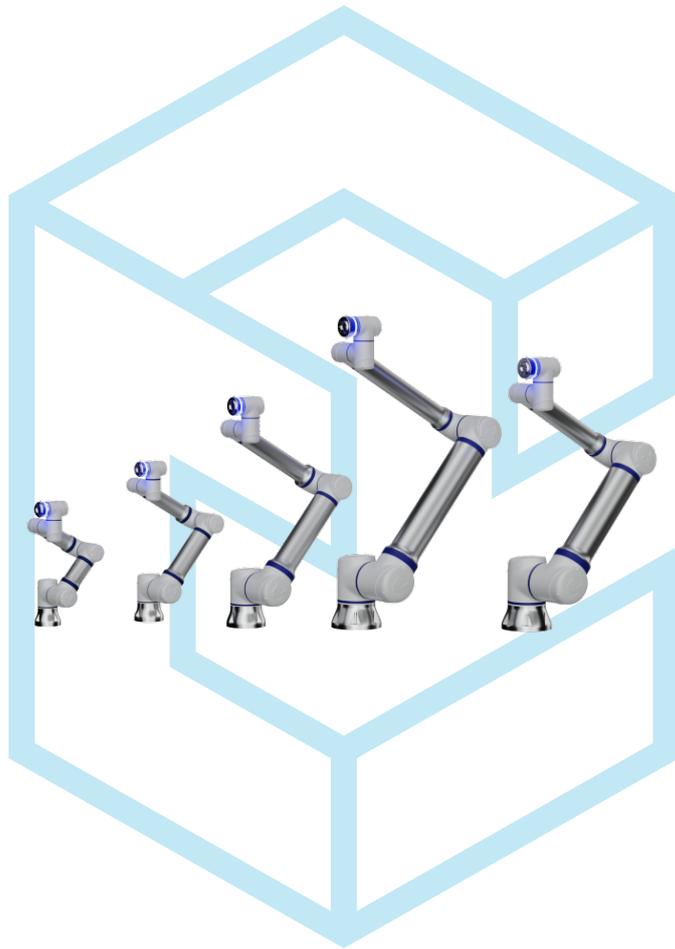


ELITE ROBOTS

脚本手册



脚本手册

艾利特智能机器人股份有限公司

2026-01-19

版本：Ver2.15.0

使用前请仔细阅读本手册

此版本用户手册对应产品版本信息请见本手册版本信息章节，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。

艾利特智能机器人股份有限公司对本手册中可能出现的任何错误概不负责。

艾利特智能机器人股份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

1 EliteScript 编程语言	1
1.1 表述约定	1
1.2 EliteScript 脚本	1
1.2.1 数据类型	2
1.2.2 变量	7
1.2.3 条件与循环语句	8
1.2.4 函数定义	11
1.2.5 线程使用	13
1.2.6 次任务介绍	14
2 解释器模式	15
2.1 使用规范	15
2.1.1 脚本要求	15
2.1.2 函数调用要求	15
2.2 解释器模式回复	16
2.2.1 确认接收 (ack)	16
2.2.2 跳过执行 (discard)	16
2.3 进入解释器模式	17
2.4 关闭解释器模式	18
2.5 清除注入的函数	19
2.6 解释器模式关键字命令	19
2.7 解释器模式日志管理	20
3 运动指令	21
3.1 MOVE 相关指令	21
3.1.1 关节运动	21

3.1.2	圆弧运动	22
3.1.3	直线运动	23
3.1.4	获取实际关节角位置	24
3.1.5	获取实际关节速度	25
3.1.6	控制机器人的关节位置	25
3.1.7	获取当前 TCP 位姿	26
3.1.8	获取当前 TCP 速度	27
3.1.9	获取控制柜主板温度	27
3.1.10	获取机器人关节温度	28
3.1.11	获取关节力矩	28
3.1.12	获取目标关节角位置	28
3.1.13	获取目标关节速度	29
3.1.14	获取当前目标工具姿态	29
3.1.15	获取目标 TCP 速度	30
3.1.16	设置加速方向	30
3.1.17	设置有效负载的质量、重心及转动惯量	30
3.1.18	获取当前有效负载的重心坐标	31
3.1.19	获取当前有效负载的质量	31
3.1.20	设置工具中心点坐标和姿态	32
3.1.21	设置编码器	32
3.1.22	获取编码器滴答计数	32
3.1.23	显示当前编码器计数	33
3.1.24	编码器计数越界	33
3.1.25	关节加速	34
3.1.26	关节减速	35
3.1.27	工具末端加速	35
3.1.28	工具末端减速	36
3.1.29	追踪圆形传送带	36

3.1.30 追踪线性传送带	37
3.1.31 停止传送带跟踪	37
3.1.32 设置传送带增量式编码器	38
3.1.33 获取当前工具法兰位姿	38
3.1.34 获取目标路径点	39
3.1.35 检测目标点位是否在安全范围内	39
3.1.36 获取目标工具偏移量	41
3.1.37 加载微段文件	41
3.1.38 设置微段文件的局部坐标	42
3.1.39 返回微段文件对应行的数据	42
3.1.40 运行微段序号对应的微段文件	43
3.1.41 开启拖动模式	43
3.1.42 关闭拖动模式	44
3.1.43 开启/关闭碰撞检测	44
3.1.44 获取当前碰撞检测启用状态	45
3.1.45 获取碰撞检测灵敏度	45
3.1.46 设置碰撞检测灵敏度	45
3.2 MATH 相关指令	46
3.2.1 逆解计算	46
3.2.2 查看逆解是否有解	46
3.2.3 获取全部逆解数据	47
3.2.4 正解计算	48
3.2.5 二进制列表转换为整数	49
3.2.6 整数转换为二进制列表	49
3.2.7 获取列表长度	49
3.2.8 获取列表或字符串的长度	50
3.2.9 反余弦运算	50
3.2.10 反正弦运算	50

3.2.11 反正切运算	50
3.2.12 判断像限的反正切运算	51
3.2.13 余弦运算	51
3.2.14 正弦运算	52
3.2.15 正切运算	52
3.2.16 角度转换为弧度	52
3.2.17 弧度转换为角度	52
3.2.18 向上取整	53
3.2.19 向下取整	53
3.2.20 对数运算	53
3.2.21 平方根运算	54
3.2.22 幂运算	54
3.2.23 计算自变量的范数	54
3.2.24 将由浮点数组成的列表内的数据归一	55
3.2.25 计算两个工具位置之间的距离	55
3.2.26 求和运算	56
3.2.27 姿态距离	57
3.2.28 位姿求逆	57
3.2.29 位姿相减	57
3.2.30 位姿变换	58
3.2.31 计算 RPY 向量	59
3.2.32 计算旋转向量	59
3.2.33 返回工具位置和方位的线性插值姿态	60
3.2.34 生成随机数	60
4 力控指令	63
4.1 开启力控模式	63
4.2 设置力控模式的增益系数	64
4.3 设置力控模式的阻尼系数	64

4.4	力/力矩传感器读数清零	65
4.5	获取施加在工具 TCP 上的力	65
4.6	获取施加在工具 TCP 上的力/力矩矢量	66
4.7	关闭力控模式	66
4.8	坐标系变换	67
5	I/O 指令	69
5.1	获取可配置 I/O 输入电平	69
5.2	获取可配置 I/O 输出电平	69
5.3	获取标准模拟输入	69
5.4	获取标准模拟输出	70
5.5	获取标准数字 I/O 输入	70
5.6	获取标准数字 I/O 输出	70
5.7	获取工具的模拟输入	71
5.8	获取工具的模拟输出	71
5.9	设置标准模拟输出的域	72
5.10	设置可配置 I/O 电平	72
5.11	设置标准模拟输入的域	72
5.12	设置标准的模拟输出	73
5.13	设置标准的数字 I/O 电平	73
5.14	设置工具末端模拟输入的域	74
5.15	设置工具末端模拟输出的域	74
5.16	设置工具数字 I/O 电平	74
5.17	获取工具数字 I/O 电平	75
5.18	设置工具模拟 I/O 工作模式	75
5.19	配置工具数字 I/O	76
5.20	设置可配置输出的运行时状态	76
5.21	设置输出布尔寄存器的运行时状态	77
5.22	设置模拟输出的运行时状态	78

5.23 设置工具模拟输出的运行时状态	79
5.24 设置标准数字输出的运行时状态	80
5.25 设置工具数字输出的运行时状态	80
5.26 设置所有输入为默认	81
5.27 设置可配置 I/O 触发动作	82
5.28 设置标准 I/O 触发动作	82
5.29 设置工具末端 I/O 触发动作	83
5.30 设置通用布尔寄存器的触发动作	84
5.31 返回工具末端的通信状态	84
5.32 配置工具末端的通信	84
5.33 读取工具末端的通信数据	85
5.34 写入工具末端的通信数据	86
5.35 设置工具法兰外部工具接头的电源电压	86
5.36 获取任务存储目录	87
5.37 读取输入寄存器中的布尔值	87
5.38 读取输入寄存器中的浮点数	87
5.39 读取输入寄存器中的整数	88
5.40 读取输出寄存器中的布尔值	88
5.41 读取输出寄存器中的浮点数	88
5.42 读取输出寄存器中的整数	89
5.43 将布尔值写入输出寄存器	89
5.44 将浮点数写入输出寄存器	90
5.45 将整数写入输出寄存器	90
6 总线指令	91
6.1 Modbus	91
6.1.1 新增信号	91
6.1.2 删除信号	92
6.1.3 新增读写信号	92

6.1.4	读取信号值	93
6.1.5	发送指定命令至 MODBUS 装置	94
6.1.6	设置从站寄存器	94
6.1.7	设置从站线圈	95
6.1.8	设置信号运行时状态	95
6.1.9	设置读写信号频率	96
6.1.10	设置有效信号的看门狗时间	96
6.1.11	读取本机线圈值	97
6.1.12	读取本机寄存器值	97
6.1.13	向本机线圈写入值	97
6.1.14	向本机寄存器写入值	98
6.1.15	返回错误码	98
6.1.16	获取无效信号距上一次读、写成功的时间间隔	100
6.1.17	重置链接	100

7 通讯指令 101

7.1	SOCKET	101
7.1.1	建立 TCP/IP 网络通信	101
7.1.2	关闭 TCP/IP 网络通信	101
7.1.3	获取 TCP/IP 网络通信状态	101
7.1.4	读取整数	102
7.1.5	读取浮点数	102
7.1.6	读取 32 位整数	103
7.1.7	读取字节流	104
7.1.8	发送字节流	105
7.1.9	读取字符串	105
7.1.10	发送一个字节	106
7.1.11	发送 32 位整数	107
7.1.12	发送一行字符串	107

7.1.13	发送字符串	108
7.1.14	发送设置数据	108
7.2	RPC	109
7.2.1	创建远程调用句柄	109
8	串口通信指令	111
8.1	RS-485	111
8.1.1	查询工具末端 RS-485 模式	111
8.1.2	控制柜 RS-485 通信配置	111
8.1.3	读取控制柜 RS-485 通信数据	112
8.1.4	写入控制柜 RS-485 通信数据	113
8.1.5	查询控制柜 RS-485 模式	113
8.1.6	刷新控制柜 RS-485 数据	113
8.2	MODBUS-RTU	114
8.2.1	工具末端读取 MODBUS-RTU 从站线圈	114
8.2.2	工具末端读取 MODBUS-RTU 从站输入线圈	114
8.2.3	工具末端读取 MODBUS-RTU 从站保持寄存器	115
8.2.4	工具末端读取 MODBUS-RTU 从站输入寄存器	115
8.2.5	工具末端写入单个 MODBUS-RTU 从站线圈	116
8.2.6	工具末端写入多个 MODBUS-RTU 从站线圈	116
8.2.7	工具末端写入单个 MODBUS-RTU 从站寄存器	117
8.2.8	工具末端写入多个 MODBUS-RTU 从站保持寄存器	117
8.2.9	控制柜读取 MODBUS-RTU 从站线圈	118
8.2.10	控制柜读取 MODBUS-RTU 从站输入线圈	118
8.2.11	控制柜读取 MODBUS-RTU 从站保持寄存器	119
8.2.12	控制柜读取 MODBUS-RTU 从站输入寄存器	119
8.2.13	控制柜写入单个 MODBUS-RTU 从站线圈	120
8.2.14	控制柜写入多个 MODBUS-RTU 从站线圈	120
8.2.15	控制柜写入单个 MODBUS-RTU 从站寄存器	121

8.2.16 控制柜写入多个 MODBUS-RTU 从站保持寄存器 121

9 系统指令 123

9.1 线程 123

9.1.1 启动线程 123

9.1.2 等待线程结束 123

9.1.3 关闭线程 124

9.2 任务运行 124

9.2.1 暂停任务运行 124

9.2.2 停止任务运行 125

9.3 提示警告 125

9.3.1 发送字符串至 EliRobot 125

9.3.2 弹框 125

9.4 字符 126

9.4.1 字符串索引 126

9.4.2 字符串拼接 126

9.4.3 空字符串判断 127

9.4.4 字符串匹配 127

9.4.5 字符串长度 128

9.4.6 获取子字符串 128

9.4.7 字符串转数字 129

9.4.8 数字转字符串 130

9.5 其他 131

9.5.1 休眠 131

9.5.2 设置系统内部标记位的值 131

9.5.3 获取系统内部标记位的值 132

9.5.4 获取机器人步长时间 132

9.5.5 获取控制器系统时间戳 132

9.5.6 启用看门狗功能 133

9.5.7 关闭电源	133
------------------	-----

第 1 章 EliteScript 编程语言

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。

1.1 表述约定

```
1 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。

1.2 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。

1.2.1 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. 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
boolean	布尔型，如 True、False
float	浮点型，如 70.2E-12
complex	复数，如 4.53e-7j

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

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

列表的截取

表 1-4. 列表的截取

方法	结果	描述
L[2]	'CS612'	读取列表中第三个元素
L[-2]	'CS66'	读取列表中倒数第二个元素
L[1:]	['CS66','CS612']	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
len(list)	列表元素个数
max(list)	返回列表元素最大值
min(list)	返回列表元素最小值
list(seq)	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
<code>list.append(obj)</code>	在列表末尾添加新的对象
<code>list.count(obj)</code>	统计某个元素在列表中出现的次数
<code>list.extend(seq)</code>	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）
<code>list.index(obj)</code>	从列表中找出某个值第一个匹配项的索引位置
<code>list.insert(index,obj)</code>	将对象插入列表
<code>list.pop(obj=list[-1])</code>	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
<code>list.remove(obj)</code>	移除列表中某个值的第一个匹配项
<code>list.reverse()</code>	反向列表中元素
<code>list.sort([func])</code>	对原列表进行排序

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
len(tuple)	计算元组元素个数。
max(tuple)	返回元组中元素最大值。
min(tuple)	返回元组中元素最小值。
tuple(seq)	将列表转换为元组。

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。

```
1 dict={}
2 dict['one']="This is one"
3 dict[2]="This is two"
4 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
<code>int(x [,base])</code>	将 x 转换为一个整数
<code>float(x)</code>	将 x 转换到一个浮点数
<code>complex(real [,imag])</code>	创建一个复数
<code>str(x)</code>	将对象 x 转换为字符串
<code>repr(x)</code>	将对象 x 转换为表达式字符串
<code>eval(str)</code>	用来计算在字符串中的有效 Python 表达式，并返回一个对象
<code>tuple(s)</code>	将序列 s 转换为一个元组
<code>list(s)</code>	将序列 s 转换为一个列表
<code>set(s)</code>	转换为可变集合
<code>dict(d)</code>	创建一个字典。d 必须是一个序列 (key,value) 元组。
<code>frozenset(s)</code>	转换为不可变集合
<code>chr(x)</code>	将一个整数转换为一个字符
<code>unichr(x)</code>	将一个整数转换为 Unicode 字符
<code>ord(x)</code>	将一个字符转换为它的整数值
<code>hex(x)</code>	将一个整数转换为一个十六进制字符串
<code>oct(x)</code>	将一个整数转换为一个八进制字符串

1.2.2 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. 全局变量

定义全局变量时，请用 `global` 关键字进行定义。

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  # 定义全局变量
2  global total
3  total=0
4
5
6  # 可写函数说明
7  def sum(arg1,arg2): #返回2 个参数的和
8      total=arg1 + arg2 # total 在这里是局部变量
9      print("函数内是局部变量 : ",total)
10     return(total)
11
12
13 #调用sum 函数
14 sum(10,20);
15 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。

1.2.3 条件与循环语句

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。

```

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

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：

```
1 def test():
2     pass
3 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：

```

1 while True:
2     pass
3 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：

```

1 i=1
2 while i<10:
3     i+=1
4     if i%2>0: # 非双数时跳过输出
5         continue
6     print(i) # 输出双数2、4、6、8、10
7 end
8
9
10 i=1
11 while 1: # 循环条件为1 必定成立
12     print(i) # 输出1~10
13     i+=1
14     if i>10: # 当i 大于10 时跳出循环
15         break
16 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。

```

1 count=0
2 while count <5:
3     print(count," is less than 5")
4     count=count +1
5 else:
6     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：

```
1 for letter in 'Python': # 逐个输出字符串中的字符，每次输出的字符都被
2 临时定义为letter（定义名称可以换成其他任意名称）
3     print('当前字母：',letter)
4
5
6 fruits=['banana','apple','mango']
7 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
8 为fruit（定义名称可以换成其他任意名称）
9     print('当前水果：',fruit)
10 print("Good bye!")
```

索引迭代：

```
1 fruits=['banana','apple','mango']
2 #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
3 for index in range(len(fruits)):
4     print('当前水果：',fruits[index])
5     print("Good bye!")
```

1.2.4 函数定义

1. 函数定义

```
1 # 标准函数定义
2 def mult(a,b):
3     return a*b
4 end
5
6
7 # 带有默认参数的函数定义
8 def add(a=3,b=3):
9     return a+b
10 end
11
12
13 # 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
14 键字，使用过程中，也可以使用标准的Python 语法定义函数。
```

2. 函数调用方法

```

1 ret_mult=mult(2,4)
2
3
4 #调用带有默认参数的函数定义，可以不进行传参
5 ret_add=add()
  
```

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。

```

1 #可写函数说明
2 def printinfo(name,age=35):
3     print("Name: ",name) # 打印任何传入的字符串
4     print("Age ",age)
5     return;
6
7
8 #调用printinfo 函数
9 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
10 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。

```
1 def printinfo(arg1,*vartuple):
2 print("输出: ",arg1) #"打印任何传入的参数"
3     for var in vartuple:
4         print var
5     return
6
7
8 # 调用printinfo 函数
9 printinfo(10)
10 printinfo(70,60,50)
```

1.2.5 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. 创建和关闭线程的方法

```
1 # 定义线程函数，定义方式与普通函数相同
2 def thread_add_func(a,b):
3     c=a + b
4     print(c)
5     return c
6 end
7
8
9 # 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。
10 thread_handler=start_thread(thread_add_func,(1,3))
11
12
13 # 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。
14 stop_thread(thread_handler)
```

2. 使用线程功能时的注意事项

- 任务停止运行后，所有创建的线程都将自动结束运行；

- 启动线程函数时，可以传递参数；
- 子线程或主线程内的循环语句，请严格避免无耗时操作的死循环，此操作可能影响系统整体性能。耗时操作主要为 sleep()、sync()、移动指令等。

1.2.6 次任务介绍

用户可以在任务运行过程中，链接 30001 端口，通过向控制系统发送 sec 脚本指令，动态地设置和获取当前运行任务中存在的全局变量数据。如遇任务运行异常，将触发报警，报警内容将以 [SEC] Exception: 为开头，并以警告日志的形式出现在控制器的日志栏中，方便用户查看。

sec 脚本表示次任务，必须以 sec 字符开头，其用法与 Python 中的 def 关键字相同。sec 脚本中存在主任务脚本程序，与 def 脚本相同，可通过 TCP/IP 连接，直接发送至 30001 端口。sec 脚本可以与主任务脚本同时执行，其脚本格式与主任务脚本格式相同。sec 脚本不支持类似 sleep、socket、与串口相关的超时参数。

用户需注意以下事项：

- 运行异常并不会停止正在运行的任务；
- 执行 sec 脚本时，请勿超时操作、长时间运行等，否则可能会出现无法预料的后果。

```
1 sec setDigitalOut():  
2     set_configurable_digital_out(1, True)  
3 end
```



第 2 章 解释器模式

Interpreter Mode（解释器模式）突破 Python 脚本动态插队限制，支持任务运行时通过 30020 端口注入实时 Python 脚本语句。进入解释器模式后，用户可将有效的 Python 语句实时下发至控制器队列中，并按队列顺序执行（最大容量 8500 条），在不中断主程序的前提下实现任务干预。

核心功能：

- 实时脚本注入：任务运行中动态插入指令；
- 函数注入：支持函数注入，且注入的函数可跨脚本调用。

2.1 使用规范

2.1.1 脚本要求

表述约定

1. 脚本语句须为单行语句，并以\n 作为结束符，例如：

```
1 set_tcp([0,0,0,0,0,0])\n
```

2. 如需下发函数及相对复杂的控制脚本，应使用分号，单行下发。整体的空格对齐方式遵循 Python 格式要求。格式如下：

```
1 def test_func():; textmsg("interpreter test"); popup("interpreter  
popup");end;
```

2.1.2 函数调用要求

1. 解释器模式不支持定义新的全局变量，但可使用和修改主程序中的全局变量。
2. 下发至解释器中的定义的函数 test_func，可在其他地方调用，该函数将在解释器执行 clear_interpreter 后被清除。
3. 解释器模式目前不支持 start_thread 函数调用。下发的脚本中，若包含了 start_thread 函数调用，系统将跳过执行（discard）此函数的调用。

4. 解释器模式在运行非运动类的耗时脚本（如 sleep 指令）时，会在等待 sleep 命令执行结束后，才可以自动退出，中途即使调用了 end_interpreter() 命令，也不会强制停止耗时脚本运行。

提示

解释器模式下，任务运行状态的控制操作（如暂停/继续/停止等）同步生效。

2.2 解释器模式回复

2.2.1 确认接收 (ack)

解释器模式接收到可执行的脚本语句时，若编译成功且已添加到脚本队列中，将返回以下格式信息，告知客户端：

```
ack: <id>: <statements>
```

其中，id 为接受的语句被分配的唯一 id，后续的一些状态命令字返回值，将以该 id 信息返回。

2.2.2 跳过执行 (discard)

若接收的脚本未被正常地压入解释器队列，解释器将跳过执行并返回抛弃信息及原因，格式如下：

```
discard: <reason>: <statement>
```

其中，reason 为抛弃原因，statement 为返回的抛弃信息。

目前内部可触发的 discard 跳过执行的常见原因有四种，详见表 2-1，具体触发原因可根据返回报文信息进行判断。

表 2-1. Discard 跳过执行的原因

类型	触发原因	返回报文信息
编译错误	一般为脚本格式错误，须检查下发的脚本格式是否为正常的 Python 格式的代码段	discard: Compile error:<trace info> : <statement>
任务状态错误	当前任务未处于运行状态	discard: Task is in an invalid state: <statement>
脚本队列长度超限	当前脚本队列长度超过最大值 (8500)	discard: Too many interpreted messages: <statement>
执行解释器前队列清除	由于 clearQueueOnEnter 参数设置为 True，在打开解释器模式之前发送到解释器队列的脚本，会在打开时（即调用 interpreter_mode 脚本）进行 discard	discard: Cleaned up before interpretation: <statement>

2.3 进入解释器模式

```
interpreter_mode(clearQueueOnEnter = True, clearOnEnd = True)
```

功能： 该指令用于进入解释器模式。这是一个阻塞函数，仅支持在主线程调用，不支持嵌套调用。

参数： clearQueueOnEnter:

True = 进入解释器模式时，清空进入前接收的脚本队列；

False = 进入解释器模式时，不清除现有队列，并立即开始执行队列中已存在的语句。

clearOnEnd:

True = 退出解释器模式时，清除解释器模式注入的所有函数（执行 clear_interpreter 命令）。

False = 退出解释器模式时，保留所有注入的语句。注入的语句将成为程序的一部分，持续有效直至程序执行结束。后续再次调用 interpreter_mode() 进入解释器模式时，这些语句仍可以被调用或访问。

返回值： 无

示例：

```
interpreter_mode(): 以默认方式启动解释器模式
interpreter_mode(clearQueueOnEnter = True, clearOnEnd = True): 启动解释器模式（进入时清除队列，退出时清除注入内容）
interpreter_mode(clearQueueOnEnter = False): 启动解释器模式（进入时不清除队列）
interpreter_mode(clearOnEnd = False): 启动解释器模式（退出时保留所有注入的语句）
```

2.4 关闭解释器模式

```
end_interpreter()
```

功能： 该指令用于关闭解释器模式，同时退出被阻塞的 interpreter_mode 函数。该函数可在任务的子线程中调用，也可通过 30020 端口发送至解释器模式，在脚本队列中等待调用。

结束时将清除所有已解释的内容，但以下内容会保留改动：机器人状态、外围作用域（enclosing scope）中局部变量的修改、以及全局变量的变更。解释器线程在此操作完成后将进入闲置状态。

参数： 无

返回值： 无

示例：

```
end_interpreter()
```

2.5 清除注入的函数

```
clear_interpreter()
```

功能： 该指令用于清除解释器注入的函数。被清除注入的函数无法在解释器中继续调用。脚本队列中，若 `clear_interpreter()` 函数被确认 (ack)，则系统会停止确认接收 (ack) 新的脚本，此时新的脚本处于等待状态。该指令执行后，处于等待状态的新脚本才会被确认接收 (ack)。

参数： 无

返回值： 无

示例： `clear_interpreter()`

2.6 解释器模式关键字命令

以下命令在接收到后将立即被执行，不会进入脚本队列等待执行：

表 2-2. 关键字命令

命令	功能	返回格式
skipbuffer	跳过已接收到但还未执行的所有脚本命令。需要注意的是，正在执行的命令会继续运行至命令结束。	-
statelastexecuted	返回最后一次执行过（或正在执行）的命令 id	state: <id>: statelastexecuted
statelastinterpreted	返回最新接收的已解析完成的命令 id	state: <id>: statelastinterpreted

statelastcleared	返回最后被清除的命令（指调用 clear_interpreter 或解释器模式退出时被清除的命令）。	state: <id>: statelastinterpreted
stateunexecuted	返回当前队列中还未执行的命令（包含等待被跳过的命令）。	state: <#unexecuted>: stateunexecuted
abort	结束当前程序中正在运行的运动指令（仅支持结束 movej 与 movel 命令）。即使该运动指令并非通过解释器模式触发，返回值可以忽略。	-

2.7 解释器模式日志管理

解释器模式使用单独的日志文件记录数据。在触发安全报警、打包飞行记录时，系统将该日志导出至飞行记录的 EliServer/controller/eli_log 文件夹中。

日志数据存储的默认路径如下：

- 1 `"/tmp/log/EliServer/interpreter.log"` # 解释器脚本日志，记录接收到的解释器脚本、discard 信息等。
- 2 `"/tmp/log/EliServer/interpreter.0.log"` # 解释器脚本日志历史日志
- 3 `"/tmp/log/EliServer/interpreter.execute.log"` # 解释器执行日志，记录解释器的执行过程，比如正在执行的脚本以及对应 id 信息等。
- 4 `"/tmp/log/EliServer/interpreter.execute.0.log"` # 解释器执行日志历史日志



第 3 章 运动指令

本章主要介绍所有运动指令的功能、参数等相关信息。

3.1 MOVE 相关指令

本节主要介绍 MOVE 相关指令。

提示



- 在 CS 系列机器人的所有运动控制脚本命令中，控制速度会受到机器人速度百分比影响。
- 使用关节运动和直线运动指令时，机器人必须处于停止状态，或者是 movej 和 movel 的转接状态。

3.1.1 关节运动

```
movej(q, a=1.396263, v=1.047197, t=0, r=0)
```

功能： 该指令为关节运动指令，用于移动机器人至位置 q 。使用该指令时，机器人须处于停止状态，或是 `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，单位为 rad/s^2 ，float 型数据；取值范围大于 0，默认值为 $1.396263\text{rad}/s^2$ （即 $80^\circ/s^2$ ）。若设置的数值过大，则系统会根据各个关节最大加速能力进行限制，加速值上限会根据轴的类型以及负载情况发生变化。

v : 关节速度，单位为 rad/s ，float 型数据；取值范围大于 0，默认值为 $1.047197\text{rad}/s$ （即 $60^\circ/s$ ）。若设置的数值过大，则系统会根据各个关节最大运动能力进行限制，速度值上限会根据轴的类型以及负载情况发生变化。

t : 时间，单位为 s，float 型数据；默认值为 0s，此时系统会根据给定的参数 a 和 v 进行规划。 t 大于 0s 时，关节加速度和关节速度被忽略，系统按照给定时间进行规划运动。

r : 交融半径，单位为 m，float 型数据。默认值为 0m，最大值与相邻路点间的距离有关。

返回值： 无

示例： `movej([0.57636,-1.01469,-2.04816,-1.29723,1.5708,0],a=1.396263,v=1.047197,t=0,r=0)`

示例指令参数：

`q=[0.57636,-1.01469,-2.04816,-1.29723,1.5708,0]` -> 目标点的关节位置设置为：基座 0.57636rad 、肩部 -1.01469rad 、肘部 -2.04816rad 、手腕1为 -1.29723rad 、手腕2为 1.5708rad 、手腕3为 0rad

`a=1.396263` -> 关节加速度为 $1.396263\text{rad}/s^2$

`v=1.047197` -> 关节速度为 $1.047197\text{rad}/s$

`t=0` -> 未指定运动时间值

`r=0` -> 交融半径为 0m

3.1.2 圆弧运动

```
movec(pose_via, pose_to, a=1.2, v=0.25, 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 pose_via 中间点到达 pose_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： pose_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；

pose_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，单位为 m/s^2 ，float 型数据；取值范围大于 0，默认值为 $1.2m/s^2$ 。若设置的数值过大，则系统会根据最大加速能力进行限制，加速值上限会根据负载情况等发生变化。

v: 工具速度，单位为 m/s，float 型数据；取值范围大于 0，默认值为 0.25m/s。若设置的数值过大，则系统会根据最大运动能力进行限制，速度值上限会根据负载情况等发生变化。

r: 交融半径，单位为 m，float 型数据。默认值为 0m，最大值与相邻路点间的距离有关。

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=1.2,v=0.25,r=0,mode=0)`

3.1.3 直线运动

```
move1(pose,a=1.2,v=0.25,t=0,r=0)
```

功能： 该指令为直线运动指令，用于移动机器人至目标位姿 p。使用该指令时，机器人须处于停止状态，或是 movej 和 move1 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，单位为 m/s^2 ，float 型数据；取值范围大于 0，默认值为 $1.2m/s^2$ 。若设置的数值过大，则系统会根据最大加速能力进行限制，加速值上限会根据负载情况等发生变化。

v: 工具速度，单位为 m/s，float 型数据；取值范围大于 0，默认值为 $0.25m/s$ 。若设置的数值过大，则系统会根据最大运动能力进行限制，速度值上限会根据负载情况等发生变化。

t: 时间，单位为 s，float 型数据；默认值为 0s，此时系统会根据给定的参数 a 和 v 进行规划。t 大于 0s 时，工具加速度和工具速度被忽略，系统按照给定时间进行规划运动。

r: 交融半径，单位为 m，float 型数据。默认值为 0m，最大值与相邻路点间的距离有关。

返回值： 无

示例： `move1([0.41951,-0.16,0.25745,-3.11757,0,-1.5708],a=1.2,v=0.25,t=0,r=0)`

示例指令参数：

p=[0.41951,-0.16,0.25745,-3.11757,0,-1.5708] -> 目标点的工具位姿设置为：x=0.41951m, y=-0.16m, z=0.25745m, Rx=-3.11757rad, Ry=0rad, Rz=-1.5708 rad

a=1.2 -> 工具加速度为 $1.2m/s^2$

v=0.25 -> 工具速度为 $0.25m/s$

t=0 -> 未指定运动时间值

r=0 -> 交融半径为 0m

3.1.4 获取实际关节角位置

```
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。

3.1.5 获取实际关节速度

```
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。

3.1.6 控制机器人的关节位置

```
servoj (q, t=0.004, lookahead_time=0.1, gain=300)
```

功能： 该指令用于实时控制机器人的关节位置，系统会在前瞻时间内，根据设定的时间间隔处理接收到的关节角度，先进行均值滤波，再将滤波的数据进行样条拟合，从而得到实时控制所需的关节位置。

参数： q: 目标关节位置 (单位：弧度)，list 型数据。
t: 当前指令对机器人的控制时间 (单位：秒)。默认值为 0.004，执行时会阻塞该时间，float 型数据 (可选参数)。

lookahead_time: 运动前瞻时间 (单位：秒)，用于计算当前位置的前馈位置。值越小，跟踪响应越快；值越大，跟踪时的超调量越小。范围为 [0.03, 0.2]，默认值为 0.1，float 型数据 (可选参数)。

gain: 控制增益值，用于控制跟踪时的速度调整力度，数值越高，响应越快。默认值为 300，范围为 [100, 2000]，integer 型数据 (可选参数)。

返回值： 无

示例：

```
servoj([0.0,1.57,-1.57,0,0,3.14], 0.004, 0.1, 300)
q = [0.0,1.57,-1.57,0,0,3.14] -> 机器人目标关节位置
t = 0.004 -> 机器人控制时间，当前命令会阻塞0.004秒
lookahead_time = 0.1 -> 前瞻时间
gain = 300 -> 控制增益
```

提示



- 过高的 gain 参数或过低的 lookahead_time 参数可能引起机器人抖动，影响运动平滑性，在目标关节位置噪声较大或更新频率较低时尤为明显。
- 建议在应用场景允许的情况下，优先使用较低的 t 参数值 (如默认值 0.004s)。

3.1.7 获取当前 TCP 位姿

```
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

3.1.8 获取当前 TCP 速度

```
get_actual_tcp_speed()
```

功能： 该指令用于获取当前 TCP 速度。返回长度为 6 的速度列表，列表前三个数值是 TCP 的 x、y、z 三个方向上的速度分量，后三个数值表示当前 TCP 姿态的旋转轴方向。

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 表示 TCP 的 x、y、z 三个方向上的速度分量，单位为 m/s，RVx、RVy、RVz 表示当前 TCP 姿态的旋转轴方向（基坐标系下），其模长 |rvx, rvy, rvz| 表示绕旋转轴的速度，单位为 rad/s。

示例：

```
get_actual_tcp_speed()
```

返回值：当前实际TCP 速度矢量

3.1.9 获取控制柜主板温度

```
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

3.1.10 获取机器人关节温度

```
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

3.1.11 获取关节力矩

```
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

3.1.12 获取目标关节角位置

```
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。

3.1.13 获取目标关节速度

```
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

3.1.14 获取当前目标工具姿态

```
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

3.1.15 获取目标 TCP 速度

```
get_target_tcp_speed()
```

功能： 该指令用于获取目标 TCP 速度。返回长度为 6 的速度列表，列表前三个数值是 TCP 的 x、y、z 三个方向上的速度分量，后三个数值表示当前 TCP 姿态的旋转轴方向。

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 表示 TCP 的 x、y、z 三个方向上的速度分量，单位为 m/s，RVx、RVy、RVz 表示当前 TCP 姿态的旋转轴方向（基坐标系下），其模长 |rvx, rvy, rvz| 表示绕旋转轴的速度，单位为 rad/s。

示例：

```
get_target_tcp_speed()
```


返回值：目标TCP速度矢量

3.1.16 设置加速方向

```
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。

3.1.17 设置有效负载的质量、重心及转动惯量

```
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/m²。

返回值： 无

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

3.1.18 获取当前有效负载的重心坐标

```
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

3.1.19 获取当前有效负载的质量

```
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 为单位的的质量。

示例： `get_target_payload_mass()`
返回值：机器人当前有效负载的质量

3.1.20 设置工具中心点坐标和姿态

```
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])`

3.1.21 设置编码器

```
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)`

3.1.22 获取编码器滴答计数

```
encoder_get_tick_count(encoder_index)
```

功能： 该指令用于返回选定编码器的嘀嗒计数。

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，传送带编码器的嘀嗒计数。

示例：
encoder_get_tick_count(0)
返回值：传送带编码器的嘀嗒计数

3.1.23 显示当前编码器计数

```
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)

3.1.24 编码器计数越界

```
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

提示


正交模式下，通道 A 和通道 B 必须设置为不同的高速 I/O 信号。

3.1.25 关节加速

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

功能： 该指令用于将每个关节加速到目标速度 qd，并以此速度持续移动。

参数： qd: 关节目标角速度 (rad/s)，list 型数据，包含 6 个数值，依次对应 J1-J6 关节。正值表示关节正向旋转，负值表示反向旋转。

a: 关节加速度 (rad/s^2)，用于从当前速度加速到目标值 qd，float 型数据，须大于 0。

t: 函数的返回时间 (s)，可选参数，float 型数据。

不指定 t 时，函数将在所有关节达到目标速度时返回，程序继续执行下一条指令，适用于后续指令须严格在目标速度条件下执行的场景（如遥控控制）。

指定 t 时，若在 t 秒内提前达到目标速度，则机器人将按照目标速度保持匀速运动，直到 t 秒结束；若 t 秒结束时仍未达到目标速度，函数将直接返回，程序继续执行下一条指令。在实际操作中，指定合适的 t 可获得一段稳定的匀速运动，同时也可避免目标速度无法达到而导致的程序死锁。

返回值： 无

示例： `speedj([0.5,1,1,1,1,1],0.5,1)`

用于将机器人本体各关节以 0.5 rad/s^2 的加速度加速到以下速度：

J1: 0.5 rad/s ; J2-J6: 1 rad/s 。

程序运行到该指令时会等待 1 秒。在此期间，各关节尝试达到目标速度。

若在 1s 内提前达到目标速度，则机器人按照目标速度保持匀速运动直到 1 秒结束；若 1 秒结束时未达到目标速度，程序仍将执行下一条指令。

提示


该指令受“速度百分比”影响。只有在速度百分比为 100% 时可达到目标速度。例如，若设置的速度百分比为 50%，实际速度会减半。

3.1.26 关节减速

```
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)`

3.1.27 工具末端加速

```
speedl(xd, a, t, aRot='a')
```

功能： 该指令用于使机器人工具末端（TCP）加速到目标速度 x_d ，并以此速度持续移动。

参数： x_d ： 工具末端的速度 (m/s)，是一个包含 6 个数值的列表 [V_x , V_y , V_z , RV_x , RV_y , RV_z]，list 型数据。正值表示沿坐标系正方向运动，负值表示沿反方向运动。
 V_x 、 V_y 、 V_z 表示工具在机器人的基坐标系下 X、Y、Z 轴方向上的线速度(m/s)； RV_x 、 RV_y 、 RV_z 表示工具在机器人的基坐标系下绕 X、Y、Z 轴旋转的角速度 (rad/s)。
a： 线加速度 (m/s^2)，float 型数据，须大于 0，用于从当前速度加速到目标值 x_d 。
t： 函数的返回时间 (s)，可选参数，float 型数据。

未指定 t 时，函数将在达到目标速度时返回，程序继续执行下一条指令，适用于后续指令须严格在目标速度条件下执行的场景（如遥控控制）；

指定 t 时，若在 t 秒内提前达到目标速度，则机器人将按照目标速度保持匀速运动，直到 t 秒结束；若 t 秒结束时仍未达到目标速度，函数将直接返回，程序继续执行下一条指令。在实际操作中，指定合适的 t 可获得一段稳定的匀速运动，同时也可避免目标速度无法达到而导致的程序死锁。

aRot：角加速度 (rad/s^2)，即工具末端角速度的加速度。默认等于线加速度 a。t 参数值指定后，aRot 参数值的设置才有效。

返回值： 无

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

用于使工具末端以 1.2 m/s^2 的加速度加速到以下速度：

沿机器人基坐标系的X轴负方向以 0.00014 m/s 移动；沿Y轴负方向以 0.04 m/s 移动；沿Z轴负方向以 0.02 m/s 移动；无旋转运动。

程序运行到该指令时会等待 100 s 。在此期间，机器人尝试加速到目标速度。

若提前达到目标速度，则机器人将按照目标速度保持匀速运动直到 100 s 结束；若 100 s 结束时未达到目标速度，程序仍将执行下一条指令。

提示

该指令同样受“速度百分比”设置影响。仅在速度百分比为 100% 时可达到目标速度。例如，若设置的速度百分比为 50%，实际速度会减半。

3.1.28 工具末端减速

```
stop1(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 ；float 型数据。

返回值： 无

示例： `stop1(0.5)`

3.1.29 追踪圆形传送带

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

功能： 该指令用于启用传送带跟踪功能，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（move() 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

3.1.30 追踪线性传送带

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

功能： 该指令用于启用传送带跟踪功能，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

3.1.31 停止传送带跟踪

```
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()`

3.1.32 设置传送带增量式编码器

```
encoder_enable_pulse_decode(encoder_index, decoder_type, A, B)
```

功能： 该指令用于配置传送带增量式编码器的模式和高速 I/O 信号。

参数： encoder_index：被访问的编码器序号。0：传送带 0；1：传送带 1。integer 型数据；

decoder_type：设置编码器模式。1：正交模式；2：升降模式；3：上升模式；4：下降模式。

A：设置通道 A 使用的高速 I/O 信号。16：高速 I/O 16；17：高速 I/O 17。

B：设置通道 B 使用的高速 I/O 信号。若编码器模式设置为正交模式，则需配置此参数。若编码器模式设置为其它模式，则只需一个高速 I/O 信号，此参数可设置为 0。

返回值： 无

示例： `encoder_enable_pulse_decode(0, 1, 16, 17)`

3.1.33 获取当前工具法兰位姿

```
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

3.1.34 获取目标路径点

```
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

3.1.35 检测目标点位是否在安全范围内

```
is_within_safety_limits(p, qNear='', ciPair='') 或 is_within_safety_limits(q='',  
qNear='', ciPair='')
```

功能： 该指令用于检测指定的位姿或关节角度是否可达，同时判断给定参数是否在当前激活的机器人安全限制内，返回布尔类型的检测结果，可在机器人运动前进行安全预检。

目标点位有两种指定位置的方式，即位姿和关节角度，二者选一。

检测范围包括：机器人关节限制、机器人安全平面限制、机器人安全工具限制、机器人正逆解限制。

参数： p：末端 TCP 位姿，格式为 [x, y, z, Rx, Ry, Rz]，list 型数据（单位：x/y/z 为 m，Rx/Ry/Rz 为 rad）。

q：关节角度，格式为 [Base, Shoulder, Elbow, Wrist1, Wrist2, Wrist3]，list 型数据（单位：rad）。需带“q=”关键词。

qnear：可选参数，最接近的关节角度，格式为 [Base, Shoulder, Elbow, Wrist1, Wrist2, Wrist3]，list 型数据，单位为 rad；默认值为机器人实时的关节角，list 型数据，仅当第一个参数是位姿时才会使用该参数进行逆运动学计算。

ciPair：可选参数，安全平面可配置输入 io 对的设置状态，格式为 [1,1,1,1]，长度为 4 的 list 型数据，每位代表一个安全开关组，值为 0 或 1。

第一位为 1 = 检查配置成 CI0 && CI1 状态且安全平面有效的安全平面；0 = 不检测；

第二位为 1 = 检查配置成 CI2 && CI3 状态且安全平面有效的安全平面；0 = 不检测；

第三位为 1 = 检查配置成 CI4 && CI5 状态且安全平面有效的安全平面；0 = 不检测；

第四位为 1 = 检查配置成 CI6 && CI7 状态且安全平面有效的安全平面；0 = 不检测；

该参数为空时，则根据实际 CI 对的高低状态以及安全平面有效性进行安全平面检测。

当安全平面有效，且未配置任何 CI 对的情况下，安全平面默认进行检测。

返回值： True = 在安全范围内；False = 超出安全限制。

示例： `is_within_safety_limits(p, qNear='', ciPair='')`

返回值：True

提示



相同数据格式的位姿和关节角度参数需通过以下方式区分：

1. 带“q=” 关键字时表示关节角度，如 q=[Base, Shoulder, Elbow, Wrist1, Wrist2, Wrist3];
2. 无关键词时默认表示位姿，如 [x, y, z, Rx, Ry, Rz]。

3.1.36 获取目标工具偏移量

```
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

3.1.37 加载微段文件

```
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
```

3.1.38 设置微段文件的局部坐标

```
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])
```

3.1.39 返回微段文件对应的行的数据

```
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

3.1.40 运行微段序号对应的微段文件

```
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)`

3.1.41 开启拖动模式

```
freedrive_mode(freeAxes=[1,1,1,1,1,1], frame=[0,0,0,0,0,0])
```

功能： 该指令用于开启拖动模式，该模式下可以自由拖动机器人。默认参数状态下，机器人将进入完全自由拖动模式，通过修改参数也可以进入锁轴拖动模式。

参数： freeAxes: 长度为 6 的数据，每位数据为 0 或 1，0 表示不能拖动，1 表示可以拖动，前三个数据表示沿笛卡尔空间的 x、y、z 三个方向拖动的使能状态，后三位表示绕笛卡尔空间的 rx、ry、rz 轴旋转的拖动使能状态。该参数表示在选择的坐标系空间内的状态。该参数为可选参数。

frame: RPY 位姿数据，定义了拖动模式的坐标系数据（相对于机器人坐标系），该参数也可以使用 base 与 tool 表示机器人坐标系与动态的 TCP 坐标系。该参数为可选参数。

返回值： 无

示例： `freedrive_mode(freeAxes=[0,1,0,0,0,0], frame="tool")`

参数：

freeAxes = [0,1,0,0,0,0] -> 锁定机器人沿y轴进行拖动

frame = "tool" -> 基于激活的实时TCP坐标系

3.1.42 关闭拖动模式

```
end_freedrive_mode()
```

功能： 该指令用于关闭拖动模式，使机器人进入位置模式，任务运行结束后也会自动退出拖动模式。

参数： 无

返回值： 无

示例： `end_freedrive_mode()`

关闭拖动模式

3.1.43 开启/关闭碰撞检测

```
set_collision_detect_enabled()
```

功能： 该指令用于开启/关闭碰撞检测开关。

参数： boolean 型数据，True = 开启；False = 关闭。

返回值： 无

示例：
`set_collision_detect_enabled(True)`
开启碰撞检测开关
`set_collision_detect_enabled(False)`
关闭碰撞检测开关

3.1.44 获取当前碰撞检测启用状态

```
is_collision_detect_enabled()
```

功能： 该指令用于获取当前碰撞检测是否启用。

参数： 无

返回值： boolean 型数据，True = 已开启；False = 处于关闭状态。

示例：
`is_collision_detect_enabled()`
返回值： True 或 False

3.1.45 获取碰撞检测灵敏度

```
get_dyn_collision_force_min_ratio()
```

功能： 该指令用于获取当前碰撞检测灵敏度。

参数： 无

返回值： integer 型数据，单位为百分比。

示例：
`get_dyn_collision_force_min_ratio()`
返回值： 50 （表示当前碰撞检测灵敏度为50%）

3.1.46 设置碰撞检测灵敏度

```
set_dyn_collision_force_min_ratio(int ratio)
```

功能： 该指令用于设置碰撞检测灵敏度。

参数： int ratio: 碰撞检测灵敏度，范围为 [10, 100]，单位为百分比。

返回值： 无

示例：
`set_dyn_collision_force_min_ratio(20)`
设置碰撞检测灵敏度为20%

3.2 MATH 相关指令

这一部分主要介绍计算相关的指令。

3.2.1 逆解计算

```
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]`

3.2.2 查看逆解是否有解

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

功能： 该指令用于检查 `get_inverse_kin` 是否有解，并返回布尔类型的结果（True 或 False），可避免 `get_inverse_kin` 无解时触发运行时报警。

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，缺省值为系统当前的激活工具（可选参数）。

返回值： 是否有解，boolean 型数据，True 或 False。

```
示例：  
global v  
p=[0.41951,-0.16,0.25745,-3.11757,0,-1.5708]  
v=get_inverse_kin_has_solution(p)  
print(v)  
参数：  
p=po=[ 0.41951,-0.16,0.25745,-3.11757,0,-1.5708]  
返回值：True
```

3.2.3 获取全部逆解数据

```
get_all_inverse_kin(p, tcp='active_tcp')
```

功能： 该指令用于获取全部逆解数据。

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`；

`tcp`: 工具形状（可选参数）。

返回值： list 型数据，list 成员为关节角度的 list 对象，关节角度单位：弧度，范围 $[-\pi, \pi]$ ，返回值长度表示逆解结果数量。

示例：

```

global v
v = get_all_inverse_kin([0.4771486056398243,0.009603682884449335,
0.5119841047483048,-3.111929473888252,4.519127222985093E-6,
-1.570799425431062], tcp=[0,0,0,0,0,0])
global ret_count
ret_count = len(v)

返回值：
v: [0.336318, -3.013889, 1.620656, 2.992033, 1.561013, 0.336185],
    [0.336318, -1.487028, -1.620656, -1.576702, 1.561013, 0.336185],
    [0.336318, -2.965255, 1.062955, 0.359507, -1.561013, -2.805408],
    [0.336318, -1.954786, -1.062955, 1.474948, -1.561013, -2.805408],
    [2.845753, -1.186798, 1.06285, 1.666371, 1.562145, 2.845879],
    [2.845753, -0.176428, -1.06285, 2.7817, 1.562145, 2.845879],
    [2.845753, -1.654565, 1.620748, -1.565353, -1.562145, -0.295714],
    [2.845753, -0.127621, -1.620748, 0.149199, -1.562145, -0.295714]
ret_count : 8 表示有八组逆解结果
  
```

注意： 如果逆解失败，v 则为空 list，ret_count 为 0。

3.2.4 正解计算

```
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]
  
```

3.2.5 二进制列表转换为整数

```
bit_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：

```
bit_list_to_integer([True , 4<5, 9==10 , 9%3==0])
```


参数说明: l = [True , 4<5, 9==10 , 9%3==0] -> 用于转换的列表数据
返回值: 11

3.2.6 整数转换为二进制列表

```
integer_to_bit_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_bit_list(11)
```


参数说明: x = 11 -> 用于转换的整数
返回值: [True, True, False, True, ..., False, False] (长度为32)

3.2.7 获取列表长度

```
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

3.2.8 获取列表或字符串的长度

```
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
```

3.2.9 反余弦运算

```
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
```

3.2.10 反正弦运算

```
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
```

3.2.11 反正切运算

`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

3.2.12 判断像限的反正切运算

`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

3.2.13 余弦运算

`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

3.2.14 正弦运算

`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

3.2.15 正切运算

`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

3.2.16 角度转换为弧度

`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

3.2.17 弧度转换为角度

`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

3.2.18 向上取整

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

3.2.19 向下取整

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

3.2.20 对数运算

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

3.2.21 平方根运算

```
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

3.2.22 幂运算

```
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

3.2.23 计算自变量的范数

```
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
```

### 3.2.24 将由浮点数组成的列表内的数据归一

```
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]
```

3.2.25 计算两个工具位置之间的距离

```
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]
p_dist=point_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: 1.42116
```

3.2.26 求和运算

```
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]
```

3.2.27 姿态距离

```
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
```

3.2.28 位姿求逆

```
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]
```

3.2.29 位姿相减

```
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。`p_to` 和 `p_from` 均含有三个位置参数 (`x, y, z`)，称为 `P`，以及三个旋转参数 (`Rx, Ry, Rz`)，称为 `R`。此函数用于计算给定姿态相减的结果 `p_result`，如下所示：

$$p_result.P = p_to.P - p_from.P$$

$$p_result.R = (p_from.R)^{-1} * p_to.R$$

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, -1.54817, 0.06821, 1.29935]
```

3.2.30 位姿变换

```
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：

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

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

参数： `p_from`：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]
```

3.2.31 计算 RPY 向量

```
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]
```

3.2.32 计算旋转向量

```
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]

3.2.33 返回工具位置和方位的线性插值姿态

```
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]

3.2.34 生成随机数

```
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

第 4 章 力控指令

4.1 开启力控模式

```
force_mode(reference_frame, selection_vector, target_wrench, mode, speed_limits)
```

功能： 该指令用于开启力控模式，机器人将被控制于力控模式下。

参数： `reference_frame`：定义力控参考坐标系的位姿矢量，该位姿矢量是相对于基座坐标系定义的。格式为 $[X,Y,Z,Rx,Ry,Rz]$ ，其中 X 、 Y 、 Z 表示位置，单位为 m ； Rx 、 Ry 、 Rz 表示位姿，单位为 rad 。 Rx 、 Ry 、 Rz 采用标准 RPY 角定义。
`selection_vector`：由 0 和 1 组成的六维矢量，用于定义力控坐标系中的力控轴，1 表示力控轴，0 表示非力控轴。

`target_wrench`：机器人施加于环境的目标力/力矩。机器人将沿/绕力控轴调整位姿以便达到指定的目标力/力矩。格式为 $[Fx,Fy,Fz,Mx,My,Mz]$ ，其中 Fx 、 Fy 、 Fz 表示沿力控轴方向施加的力，单位为 N ； Mx 、 My 、 Mz 表示绕力控轴方向施加的力矩，单位为 Nm 。该值对非力控轴无效。由于关节安全限制，实际施加的力/力矩可能低于设置的目标力/力矩。在单独的线程中使用 `get_tcp_force` 脚本指令可读取实际施加于环境的力/力矩。

`mode`：力控模式参数，integer 型数据，范围为 0 到 3，用于定义力控模式，即：力控坐标系如何定义或者如何由力控参考坐标系变换获得。

0：固定模式。力控坐标系为力控参考坐标系。

1：点模式。力控坐标系的 Y 轴由机器人 TCP 原点指向力控参考坐标系的原点。

2：运动模式。力控坐标系的 X 轴为 TCP 移动方向矢量在力控参考坐标系的 X - Y 平面内的投影。

3：TCP 模式。力控坐标系为机器人的 TCP 坐标系。

`speed_limits`：速度限制参数，六维矢量，float 型数据。格式为 $[Vx,Vy,Vz,\omega_x,\omega_y,\omega_z]$ ，其中 Vx 、 Vy 、 Vz 表示沿该轴允许的最大 TCP 速度，单位为 m/s ； ω_x 、 ω_y 、 ω_z 表示绕该轴允许的最大 TCP 速度，单位为 rad/s 。该速度限制参数对非力控轴无效，非力控轴仍执行该轴上的原始轨迹。

返回值： 无

示例：`force_mode([0.5,0.8,0,0,0,1.57], [0,0,1,0,0,0], [0,0,50,0,0,0], 0, [0,0,0.1,0,0,0])`

参数：

Reference frame=[0.5,0.8,0,0,0,1.57] 力控参考坐标系由基座坐标系在X 轴方向上偏移 500mm、在Y轴方向上偏移800mm、在RZ轴方向上旋转90°获得。

Selection Vector=[0,0,1,0,0,0] 力控坐标系的Z 轴为力控轴，其余轴为非力控轴。

Target wrench=[0,0,50,0,0,0] 机器人将在力控坐标系的Z轴方向上施加50N的力。

Mode=0 力控模式为固定模式，力控坐标系为用户给定的力控参考坐标系。

Speed_limits = [0,0,0.1,0,0,0] 沿力控轴Z方向的最大运动速度为100mm/s。

- 注意：
1. 进入力控模式之前，请避免平行于力控轴方向的运动和高减速度的运动。推荐进入力控模式之前插入至少 0.02 秒的 sleep 脚本指令解决该问题。
 2. 在力控模式中，请避免高加速度运动，否则会降低力控精度。

4.2 设置力控模式的增益系数

```
force_mode_set_gain_scaling(scoring)
```

功能：该指令用于设置力控模式过程的增益系数。

参数：scoring：增益系数，该系数介于 0 到 2 之间，默认值为 0.2。

增益系数越小，力跟踪的响应时间就越慢，超调量就越小；反之，增益系数越大，力跟踪的响应时间就越快，超调量就越大，增益系数过大可能会导致抖动。例如，在发生碰撞或挤压坚硬表面的情况下，若增益系数大于 1，力控模式将处于不稳定状态。

返回值：无

示例：`force_mode_set_gain_scaling(0.5)`

设置力控模式过程的增益系数为0.5。

注意：该增益系数会保留至再次调用该脚本指令。

请在程序起始处或进入力控模式之前添加该指令，以确保其在进入力控模式之前被调用；否则，关机重启后，该增益系数将恢复为默认值。

4.3 设置力控模式的阻尼系数

```
force_mode_set_damping(damping)
```

功能： 该指令用于设置力控模式过程的阻尼系数。

参数： damping： 阻尼系数，该系数介于 0 到 1 之间，默认值为 0。
该阻尼系数设置为 1 时为全阻尼状态，若无外力作用，机器人将快速减速。该阻尼系数设置为 0 时为无阻尼状态，机器人将保持当前速度运动。
阻尼系数越小，机器人在力控方向上的阻尼就越小，力跟踪的震荡抑制能力就越弱，力跟踪响应就越快；反之，阻尼系数越大，机器人在力控方向上的阻尼就越大，力跟踪的震荡抑制能力就越强，力跟踪响应就越慢。

返回值： 无

示例： `force_mode_set_damping(0.5)`
设置力控模式过程的阻尼系数为 0.5。

注意： 该阻尼系数会保留至再次调用该脚本指令。
请在程序起始处或进入力控模式之前添加该指令，以确保其在进入力控模式之前被调用；否则，关机重启后，该阻尼系数将恢复为默认值。

4.4 力/力矩传感器读数清零

```
zero_ftsensor()
```

功能： 该指令用于将力/力矩传感器测量的施加在工具 TCP 上的力/力矩值清零（去皮），所述力/力矩值为 `get_tcp_force(True)` 脚本指令获取的施加在工具 TCP 上的力/力矩矢量，该矢量已进行负载补偿等处理。
该指令执行后，当前的力/力矩测量值会被作为力/力矩参考值保存，后续所有的力/力矩测量值都会减去该力/力矩参考值（去皮）。
请注意，上述力/力矩参考值会在该指令执行时更新，在控制器重启后将重置为 0。

参数： 无

返回值： 无

示例： `zero_ftsensor()`
将当前获取的施加在工具 TCP 上的已进行负载补偿等处理的力/力矩矢量清零（去皮），后续获取的所有力/力矩值都会减去当前的参考力/力矩值。

4.5 获取施加在工具 TCP 上的力

```
force()
```

- 功能：** 该指令用于获取施加在工具 TCP 上的外力。
获取当前施加在工具 TCP 上的外力，该力的值为使用 `get_tcp_force(True)` 脚本指令获取的 F_x 、 F_y 和 F_z 的范数。
有关清零力/力矩传感器读数，另请参阅：3.4 `zero_ftsensor` 指令。
- 参数：** 无
- 返回值：** 力，单位为 N，float 型数据。
- 示例：**
- ```
fe=force()
获取施加在工具TCP上的力。
```

## 4.6 获取施加在工具 TCP 上的力/力矩矢量

```
get_tcp_force(enable_compensation)
```

- 功能：** 该指令用于获取施加在工具 TCP 上的力/力矩矢量。  
有关清零力/力矩传感器读数，另请参阅：3.4 `zero_ftsensor` 指令。
- 参数：** `enable_compensation`: boolean 型数据。True 表示返回值进行负载补偿等处理，False 表示返回值为力/力矩传感器的原始读数，不做任何处理。
- 返回值：** 力/力矩矢量，格式为  $[F_x(N), F_y(N), F_z(N), TR_x(Nm), TR_y(Nm), TR_z(Nm)]$ 。  $F_x$ 、 $F_y$  和  $F_z$  表示力，单位为 N； $TR_x$ 、 $TR_y$  和  $TR_z$  表示力矩，单位为 Nm。该力/力矩矢量是以工具 TCP 坐标系为参考坐标系表示的。
- 示例：**

```
ft_comp=get_tcp_force(True)
获取施加在工具TCP上的力/力矩矢量，该矢量已进行负载补偿等处理。
ft_orig=get_tcp_force(False)
获取力/力矩传感器的原始力/力矩矢量，该矢量为传感器的原始读数，不做任何处理。
```

## 4.7 关闭力控模式

```
end_force_mode()
```

功能： 该指令用于关闭力控模式。程序结束时也会执行此命令。

参数： 无

返回值： 无

示例：  
`end_force_mode()`  
关闭力控模式。

## 4.8 坐标系变换

```
w_to=wrench_trans(T_from_to, w_from)
```

功能： 该指令用于将初始坐标系中定义的力/力矩（wrench，包括力和力矩）变换到目标坐标系。

参数： T\_from\_to：坐标系变换矩阵对应的 RPY 位姿，格式为 [X, Y, Z, Rx, Ry, Rz]，其中，前三位的单位为 m，后三位的单位为 rad。

w\_from：初始坐标系中定义的力/力矩（wrench），格式为 [Fx, Fy, Fz, Mx, My, Mz]，其中，Fx、Fy 和 Fz 表示力，单位为 N；Mx、My 和 Mz 表示力矩，单位为 Nm。

w\_to：变换后的力/力矩（wrench），格式为 [Fx, Fy, Fz, Mx, My, Mz]，其中，Fx、Fy 和 Fz 表示力，单位为 N；Mx、My 和 Mz 表示力矩，单位为 Nm。

返回值： 变换后的力/力矩矢量，即 w\_to，格式为 [Fx, Fy, Fz, Mx, My, Mz]。

示例：  
`global w1`  
`w1=wrench_trans([0.1, 0, 0, 0, 0, 1.5708], [10, 0, 0, 0, 0, 0])`  
返回结果： [0, 10, 0, 0, 0, 1]



## 第 5 章 I/O 指令

本章主要介绍设置、获取 I/O 信号电平等相关指令。

### 5.1 获取可配置 I/O 输入电平

```
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：5.5 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；

### 5.2 获取可配置 I/O 输出电平

```
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：5.6 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；

### 5.3 获取标准模拟输入

```
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：5.7 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；

## 5.4 获取标准模拟输出

```
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：5.8 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；

## 5.5 获取标准数字 I/O 输入

```
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：5.1 get\_configurable\_digital\_in 指令。
- 参数：** n：输入的索引，integer 型数据；范围为 [0:15]（MINI 控制柜范围 [0:7]）。
- 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；

## 5.6 获取标准数字 I/O 输出

```
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：5.2 get\_configurable\_digital\_out 指令。

参数： n：输出的索引，integer 型数据；范围为 [0:15]（MINI 控制柜范围 [0:7]）。

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；

## 5.7 获取工具的模拟输入

```
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：5.3 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

## 5.8 获取工具的模拟输出

```
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：5.4 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 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

## 5.9 设置标准模拟输出的域

```
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： 5.15 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)`

## 5.10 设置可配置 I/O 电平

```
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： 5.13 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)`

## 5.11 设置标准模拟输入的域

```
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：5.14 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)`

## 5.12 设置标准的模拟输出

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

- 功能：** 该指令用于设置标准模拟输出 I/O 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I/O模式设置）

## 5.13 设置标准的数字 I/O 电平

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

- 功能：** 该指令用于设置标准数字输出 I/O 信号电平；  
另请参阅：5.10 set\_configurable\_digital\_out 指令。
- 参数：** n: 输出的索引，integer 型数据；范围为 [0:15]（MINI 控制柜范围为 [0:7]）；  
b: 信号电平，boolean 型数据，True 或 False。
- 返回值：** 无
- 示例：** `set_standard_digital_out(0,True)`

## 5.14 设置工具末端模拟输入的域

```
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 5.11 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)`

## 5.15 设置工具末端模拟输出的域

```
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 5.9 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)`

## 5.16 设置工具数字 I/O 电平

```
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 I/O 电平。

参数： n: 工具数字 I/O 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 I/O 为禁用或者非输出类型时, 使用此接口设置后, 会触发 I/O 类型错误的运行时报警。

## 5.17 获取工具数字 I/O 电平

```
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 I/O 信号电平。

参数： n: 工具数字 I/O 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

## 5.18 设置工具模拟 I/O 工作模式

```
set_tool_analog_io_work_mode(domain)
```

功能： 该指令用于设置工具模拟 I/O 的工作模式。

参数： domain: 工作模式。integer 型数据。  
0: USART 模式。  
1: 模拟输入有效。  
2: 模拟输出有效。  
3: 模拟输入及输出均有效。

返回值： 无

示例： 无

**提示**


ES 系列仅支持 USART 模式。

## 5.19 配置工具数字 I/O

```
set_tool_digital_io_config(index, mode, enable, is_out)
```

**功能：** 该指令用于配置工具数字 I/O。

**参数：** index: 工具数字 I/O 的引脚序号。integer 型数据。

mode: 数字 I/O 的工作模式。integer 型数据。

0: 推/拉。

1: 灌电流 PNP。

2: 吸电流 NPN。

enable: 是否使能。bool 型数据。True 为使能，False 为失能。

is\_out: 是否为输出。bool 型数据。True 为输出，False 为输入。

**返回值：** 无

**示例：** 无

**提示**


ES 系列仅支持“单针模式”（数字 I/O 全部启用，数字 I/O[0], [1] 为输出，输出模式为 PNP；数字 I/O[2], [3] 为输入，输入模式为 PNP）。

## 5.20 设置可配置输出的运行时状态

```
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：

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

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

5.23 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；

## 5.21 设置输出布尔寄存器的运行时状态

```
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：

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

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

5.23 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

## 5.22 设置模拟输出的运行时状态

```
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

## 5.23 设置工具模拟输出的运行时状态

```
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

## 5.24 设置标准数字输出的运行时状态

```
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：

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

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

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

**参数：** port: 输出的索引, integer 型数据; 范围为 [0:15] (MINI 控制柜范围为 [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_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

## 5.25 设置工具数字输出的运行时状态

```
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：

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

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

5.22 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

## 5.26 设置所有输入为默认

```
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 I/O 触发动作设置为 “defalut”；

另请参阅：

5.26 set\_standard\_digital\_input\_action 指令

5.25 set\_configurable\_digital\_input\_action 指令

5.27 set\_tool\_digital\_input\_action 指令

5.28 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 I/O 触发动作为 “defalut ”。

## 5.27 设置可配置 I/O 触发动作

```
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：

5.24 set\_input\_actions\_to\_default 指令

5.26 set\_standard\_digital\_input\_action 指令

5.27 set\_tool\_digital\_input\_action 指令

5.28 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

## 5.28 设置标准 I/O 触发动作

```
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：

5.24 set\_input\_actions\_to\_default 指令

5.25 set\_configurable\_digital\_input\_action 指令

5.27 set\_tool\_digital\_input\_action 指令

5.28 set\_gp\_boolean\_input\_action 指令

**参数：** port: 输入的索引，integer 型数据，范围为 [0:15]（MINI 控制柜范围 [0:7]）；  
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

## 5.29 设置工具末端 I/O 触发动作

```
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：

5.24 set\_input\_actions\_to\_default 指令

5.25 set\_configurable\_digital\_input\_action 指令

5.26 set\_standard\_digital\_input\_action 指令

5.28 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；

## 5.30 设置通用布尔寄存器的触发动作

```
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：

5.24 set\_input\_actions\_to\_default 指令

5.25 set\_configurable\_digital\_input\_action 指令

5.26 set\_standard\_digital\_input\_action 指令

5.27 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

## 5.31 返回工具末端的通信状态

```
tool_serial_is_open()
```

**功能：** 该指令用于返回机器人工具末端 RS-485 通信功能是否打开。

**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

## 5.32 配置工具末端的通信

```
tool_serial_config(enabled, baud_rate, parity, stop_bits, size=8, modbus_rtu=False, usage=0)
```

**功能：** 该指令用于打开或者关闭机器人工具末端 RS-485 通信功能，并对其进行配置。

**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;  
`usage`: 可选参数, int 型数据: 0 (脚本模式), 1 (Daemon 模式), 不指定时, 使用默认值 0, 即脚本模式。脚本模式指的是通过脚本接口访问串口, 具体可参阅相关的工具接口 (即 `tool_serial`、`tool_modbus`)。Daemon 模式指的是通过 ELITECO SDK Daemon 插件访问串口 `ttyTCIO`, 并对其进行读写操作, 若此时通过脚本接口进行访问, 则会触发运行时异常报警。

**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
usage=0
返回值: True 或False
```

**注意：** 1. 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;  
2. 通信模式为 RS485 时, 可使用下列指令 `tool_serial_read`、`tool_serial_write`。

## 5.33 读取工具末端的通信数据

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

**功能：** 该指令用于读取机器人工具末端 RS-485 通信的数据。

**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"

## 5.34 写入工具末端的通信数据

```
tool_serial_write(write_data)
```

**功能：** 该指令用于机器人工具末端 RS-485 发送数据。

**参数：** write\_data: 待发送的字节数据, string 型数据、integer 型数据、整数 list 型数据或 bytearray 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

## 5.35 设置工具法兰外部工具接头的电源电压

```
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)`

## 5.36 获取任务存储目录

```
get_task_path()
```

功能： 该指令用于返回 EliRobot 的任务存储目录。

参数： 无

返回值： string 型数据，EliRobot 的任务存储目录。

示例： `task_path=get_task_path()`

## 5.37 读取输入寄存器中的布尔值

```
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

## 5.38 读取输入寄存器中的浮点数

```
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

## 5.39 读取输入寄存器中的整数

```
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

## 5.40 读取输出寄存器中的布尔值

```
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

## 5.41 读取输出寄存器中的浮点数

```
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

## 5.42 读取输出寄存器中的整数

```
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

## 5.43 将布尔值写入输出寄存器

```
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。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)

## 5.44 将浮点数写入输出寄存器

```
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。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)`

## 5.45 将整数写入输出寄存器

```
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。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)`

## 第 6 章 总线指令

### 6.1 Modbus

#### 6.1.1 新增信号

```
modbus_add_signal(IP, slave_number, signal_address, signal_type, signal_name,
is_sequential = False, registers_count = 1, port = 502)
```

**功能：** 该指令用于在控制器中添加一个 MODBUS TCP 信号；此信号在控制器中会被持续监视与控制；在当前版本中，modbus 可添加最多 200 个信号；对于输出信号，在第一次调用 modbus\_set\_output\_signal/register() 之前会使用 MODBUS 的功能码 1/3 持续读值，调用之后使用 MODBUS 的功能码 5/6/15/16 来持续写值。**表 6-1**展示了不同情况下后台对功能码的使用。

**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。  
is\_sequential: 是否顺序更新，如果为 true，将按照信号添加的顺序来更新信号。bool 型数据（可选参数）。  
registers\_count: 信号的连续读写线圈和寄存器的数量。int 型数据: [1: 123]（可选参数）。  
port: 从站的端口，Integer 型数据。（可选参数，默认为 502）。

**返回值：** 无

**示例：**

```
modbus_add_signal("192.168.1.100", 255, 0, 3, "output1", False, 10,
502)
```

表 6-1. 各信号类型的功能码

| 信号类型      | Read |     | Write |      |
|-----------|------|-----|-------|------|
|           | 单线圈  | 多线圈 | 单寄存器  | 多寄存器 |
| 0 = 数字输入  | 2    | 2   | /     | /    |
| 1 = 数字输出  | 1    | 1   | 5     | 15   |
| 2 = 寄存器输入 | 4    | 4   | /     | /    |
| 3 = 寄存器输出 | 3    | 3   | 6     | 16   |

### 6.1.2 删除信号

```
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： signal\_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")`

### 6.1.3 新增读写信号

```
modbus_add_rw_signal(IP, slave_number, read_address, read_register_count,
 write_address, write_register_count, signal_name, is_sequential = False, port
 = 502)
```

**功能：** 该指令用于在控制器中添加一个 MODBUS TCP 信号，功能码为 23。读写地址可以重叠。在第一次调用 `modbus_set_output_register()` 指令之前，使用的功能码为 3，调用后功能码为 23。

**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]。  
read\_address: 读取寄存器的起始地址。Integer 型数据；  
read\_register\_count: 读取寄存器的数量。Integer 型数据 [1: 123]；  
write\_address: 写入寄存器的起始地址。Integer 型数据；  
write\_register\_count: 写入寄存器的起始地址。Integer 型数据；  
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；  
is\_sequential: 是否顺序更新，如果为 true，将按照信号添加的顺序来更新信号。bool 型数据（可选参数）。  
port: 从站的端口，Integer 型数据。（可选参数，默认为 502）。

**返回值：** 无

**示例：** `modbus_add_rw_signal("172.140.17.11", 255, 0, 10, 2, 10, "output1", False, 502)`

### 6.1.4 读取信号值

```
modbus_get_signal_status(signal_name, is_secondary_program=False)
```

**功能：** 该指令用于读取特定信号的当前值。

**参数：** signal\_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：** 对于单个数字信号：True 或 False，boolean 型数据；  
对于单个寄存器信号：以无符号整数显示的寄存器数值，integer 型数据；  
对于多个数字信号：读取的数字信号列表，list 型数据；  
对于多个寄存器信号：读取的寄存器信号列表，list 型数据。

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

### 6.1.5 发送指定命令至 MODBUS 装置

```
modbus_send_custom_command(IP, slave_number, function_code, data, oneshot = True
, port = 502)
```

**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 设置。由于无法处理 modbus 设备的响应，该指令无法用于接收数据，因此用户所提供的数据须是有意义的，指令会根据 modbus 协议构建报文。

**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: 从站地址。Integer 型数据: [0: 255]；  
function\_code: 用于指定自定义命令所用的功能码。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。list 型数据；  
oneshot: 是否只发送一次请求。boolean 型数据。(可选参数，默认为 True)。  
port: 从站的端口，Integer 型数据。(可选参数，默认为 502)。

**返回值：** 无

**示例：** `modbus_send_custom_command ("172.140.17.11",103,6,[17,32,2,88],True  
,502)`

说明: Backhoff BK9050 上的 watchdog 超时设为 600ms。

这是如下所述完成的: 使用 modbus 功能码 6 (预设单个寄存器), 然后在数据数组的前两个字节 ([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。

### 6.1.6 设置从站寄存器

```
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`: 待写入的寄存器值。如果是一个整数类型的值，范围在 [0:65535]，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；如果为列表，列表长度范围最小为 1，并且必须小于等于添加信号时声明的寄存器数量。如果较小长度的列表输入，那么 MODBUS 地址较高的寄存器将会保留原来的值。integer 或 list 型数据。

**返回值：** 无

**示例：**

```
modbus_set_output_register("output1",300)
参数：
signal_name="output1"
register_value=300
```

### 6.1.7 设置从站线圈

```
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`: 待写入的线圈值。值的类型为布尔值或布尔值列表；如果为列表，列表长度范围最小为 1，并且必须小于等于添加信号时声明的线圈数量。如果较小长度的列表输入，那么 MODBUS 地址较高的线圈将会保留原来的值。boolean 或 list 型数据。

**返回值：** 无

**示例：**

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

### 6.1.8 设置信号运行时状态

```
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)`

### 6.1.9 设置读写信号频率

```
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)`

### 6.1.10 设置有效信号的看门狗时间

```
modbus_set_signal_watchdog(signal_name, timeout)
```

功能： 该指令用于设置 modbus 有效信号的看门狗时间，即信号在出现错误之后的看门狗时间内，使用脚本读写信号将不会抛出异常。可以使用 `modbus_get_error()` 和 `modbus_get_time_since_signal_invalid()` 指令在读写前获取信号是否异常。如未设置，默认的看门狗时间为 5 秒。

参数： signal\_name: 信号名称。string 型；  
timeout: 看门狗超时时间，范围： [0, 300]。单位：秒。float 型。

返回值： 无

示例： `modbus_set_signal_watchdog("MODBUS", 120)`

### 6.1.11 读取本机线圈值

```
read_port_bit(address)
```

功能： 该指令用于读取本机 Modbus 从站数字（线圈）地址的值。

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

### 6.1.12 读取本机寄存器值

```
read_port_register(address, is_signed=False)
```

功能： 该指令用于读取本机 Modbus 从站寄存器地址的值。

参数： address：端口的地址，integer 型数据。

is\_signed： 是否是有符号的数据，默认为 False，可选参数，boolean 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(256,True)`  
返回值： 有符号整数

### 6.1.13 向本机线圈写入值

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

功能： 该指令用于向本机 Modbus 从站数字（线圈）地址写入值。

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；

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)`

### 6.1.14 向本机寄存器写入值

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

功能： 该指令用于向本机 Modbus 从站寄存器写入值。

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)`

### 6.1.15 返回错误码

```
modbus_get_error(signal_name, error_source = 0)
```

功能： 该指令用于返回信号当前的错误码。

参数： signal\_name: 信号名，与应当获取数值的信号名称相同。string 型数据；  
error\_source: 错误码的源。Integer 型数据，0 = 复合，1 = 设备，2 = 连接。(可选参数，默认为 0)。

返回值： 错误码，list 型数据。设备错误码见表 6-2。连接错误码的值的范围为 [1 : 113]，相关说明请参考[维基百科相关页面](#)和[IBM 官网 TCP/IP errors 主页](#)。

示例： `ec = modbus_get_error("MODBUS_1")`

表 6-2. 设备错误码

| 数值 | 含义                                       |
|----|------------------------------------------|
| -2 | 未连接                                      |
| -1 | 连接中                                      |
| 0  | 无错误                                      |
| 1  | 错误的功能码 = ILLEGAL FUNCTION CODE           |
| 2  | 错误的设备内存地址 = ILLEGAL DATA ADDRESS         |
| 3  | 请求包含一个非法的数据值 = ILLEGAL DATA VALUE        |
| 4  | 从机故障 = SLAVE OR SERVER FAILURE           |
| 5  | 已经接收到主站请求，但需要更多的时间来完成请求的操作 = ACKNOWLEDGE |
| 6  | 从站忙 = SLAVE OR SERVER BUSY               |
| 7  | 从站拒绝处理主机的请求 = NEGATIVE ACKNOWLEDGE       |
| 8  | 存储器奇偶校验错误 = MEMORY PARITY                |
| 9  | 未定义的异常 = NOT DEFINED                     |
| 10 | 网关路径不可用 = GATEWAY PROBLEM                |
| 11 | 网关目标设备响应失败 = GATEWAY TARGET FAILURE      |

提示



1. 若错误来源是复合的，将会返回长度为 2 的列表，列表第一个元素为设备错误码，第二个为连接错误码。
2. 若错误来源是设备或连接，则返回的列表长度为 1，数据内容就是对应的错误码。

### 6.1.16 获取无效信号距上一次读、写成功的时间间隔

```
modbus_get_time_since_signal_invalid(signal_name)
```

**功能：** 对于无效信号（如设备或通讯错误），该指令用于获取无效信号距离上一次读、写成功的时间间隔值（单位：秒）。对于有效信号，返回 0。

**参数：** signal\_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：** 距离上一次读写成功过了多少秒。float 型数据。

**示例：** `dleta = modbus_get_time_since_signal_invalid("MODBUS_1")`

### 6.1.17 重置链接

```
modbus_reset_connection(connection_id, is_blocking = True, port = 502)
```

**功能：** 该指令用于重置链接。

**参数：** connection\_id: MODBUS 设备的 IP 地址。string 型数据。

is\_blocking: 是否阻塞。boolean 型数据。（可选参数，默认为 True）。

port: 从站的端口，Integer 型数据。（可选参数，默认为 502）。

**返回值：** 无

**示例：** `modbus_reset_connection("192.168.1.123", True, 502)`

## 第 7 章 通讯指令

### 7.1 SOCKET

#### 7.1.1 建立 TCP/IP 网络通信

```
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。首次调用：若连接成功返回 true，失败返回 false；重复调用：若处于连接状态返回 true，若处于未连接状态将会关闭当前 socket 并重新连接，重连成功返回 true，失败返回 false。

示例：  
socket\_open("192.168.1.119", 23333, "socket\_1")  
返回值： True 或 False

#### 7.1.2 关闭 TCP/IP 网络通信

```
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")

#### 7.1.3 获取 TCP/IP 网络通信状态

```
socket_is_connected(socket_name="socket_0")
```

**功能：** 该指令用于获取 TCP/IP 网络通信状态，判断是否连接到服务器。

**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：** boolean 型数据，若未连接，返回 False，若已连接，返回 True。

**示例：** `socket_is_connected(socket_name="socket_3")`  
返回值: True 或 False

### 7.1.4 读取整数

```
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\_01000\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

### 7.1.5 读取浮点数

```
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]
```

## 7.1.6 读取 32 位整数

```
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]
```

### 7.1.7 读取字节流

```
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: [1:512];  
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]
```

## 7.1.8 发送字节流

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

**功能：** 该指令用于向连接的 TCP/IP 服务器发送一组字节数据。

**参数：** value: 需要发送的字节数据, 整数列表, list 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, 返回成功发送的字节长度。

**示例：**

```
socket_send_byte_list([1, 0x3, 0x2E, 3])
参数:
value=[1, 0x3, 0x2E, 3]
socket_name=socket_0, 默认名称
返回值: 若发送成功, 则返回4
```

## 7.1.9 读取字符串

```
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 len(string_from_server)==0:
 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。

### 7.1.10 发送一个字节

```
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
```

### 7.1.11 发送 32 位整数

```
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
```

### 7.1.12 发送一行字符串

```
socket_send_line(str, linefeed_type=0, 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> 的末尾增加用户指定的换行符。

**参数：** str: 要发送的字符串，string 型数据；  
linefeed\_type: 换行符类型，integer 型数据，0: 末尾增加 “\n” 换行符，1: 末尾增加 “\r” 换行符，2: 末尾增加 “\r\n” 换行符，默认类型为 0（可选参数）；  
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：** 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

### 7.1.13 发送字符串

```
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

### 7.1.14 发送设置数据

```
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

## 7.2 RPC

### 7.2.1 创建远程调用句柄

```
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



## 第 8 章 串口通信指令

本章主要介绍 MODBUS-RTU 及 RS-485 串口通信相关的一些指令。

### 8.1 RS-485

本节主要介绍查询、读取或写入工具末端和控制柜 RS-485 通信状态的一些指令。

#### 8.1.1 查询工具末端 RS-485 模式

```
tool_serial_mode()
```

功能： 该指令用于查询机器人工具末端 RS-485 处于哪种模式下。

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”

#### 8.1.2 控制柜 RS-485 通信配置

```
masterboard_serial_config(enable, baud = 115200, parity = 1, stop_bit = 1, size
= 8, modbus_rtu = False)
```

**功能：** 该指令用于打开/关闭机器人控制柜 RS-485 通信功能，并对其进行配置。

**参数：** enable: 串口通信的使能状态, boolean 型数据。

baud: 波特率, integer 型数据: 2400、4800、9600、19200、38400、57600、115200、230400、460800、921600、500000。不指定时使用默认值 115200。

parity\_bit: 串口奇偶校验, integer 型数据: 0 (不校验)、1 (奇校验)、2 (偶校验)。不指定时使用默认值 1 (奇校验)。

stop\_bit: 串口停止位, integer 型数据: 1、2。

size: 串口的位宽。不指定时, 使用默认值 8, integer 型数据: 7 或 8 (可选参数)。若为 MODBUS-RTU 模式, 则强制为 8;

modbus\_rtu: 是否启用 MODBUS-RTU 模式, boolean 类型数据。True = 启用 MODBUS-RTU 模式; False = 启用 RS485 模式。不指定时使用默认值 False。

**返回值：** boolean 型数据。True = 配置成功; False = 配置失败。

**示例：** `masterboard_serial_config(True, 115200, 1, 1, 8, False)`  
返回值: True 或 False

### 8.1.3 读取控制柜 RS-485 通信数据

```
masterboard_serial_read(read_count, is_number = True, timeout = 0.1)
```

**功能：** 该指令用于读取机器人控制柜 RS-485 通信的数据。

**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。若不指定, 使用默认值 True, boolean 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, 不指定时, 使用默认值 0.1, float 型数据 (可选参数)。

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

**示例：** `masterboard_serial_read(4, True, 0.1)`

参数:

read\_count=4

is\_number=True 默认值

timeout=0.1, 默认值

返回值: 读取数据的列表, [12, 99, 255, 1]

### 8.1.4 写入控制柜 RS-485 通信数据

```
masterboard_serial_write(data)
```

功能： 该指令用于机器人控制柜 RS-485 发送数据。

参数： data: 待发送的数据, string 型数据、integer 型数据、整数 list 型数据、bytes 型数据或 bytearray 型数据。若整数超出 [0 255] 范围, 会自动按字节规则截断, 只保留其最低 8 位。

返回值: boolean 型数据。True = 发送成功; False = 发送失败。

示例: `masterboard_serial_write([123,56,0])`

参数:

`data=[123,56,0]`

返回值: True 或 False

### 8.1.5 查询控制柜 RS-485 模式

```
masterboard_serial_mode()
```

功能： 该指令用于查询机器人控制柜 RS-485 处于何种模式下。

参数： 无。

返回值: integer 型数据。0 = 串口关闭; 1 = RS485 模式; 2 = MODBUS-RTU 模式。

示例: `mode=masterboard_serial_mode()`

返回值: 2

### 8.1.6 刷新控制柜 RS-485 数据

```
masterboard_serial_flush()
```

功能： 该指令用于刷新控制柜 RS-485 的数据。

参数： 无

返回值: 无

示例: `masterboard_serial_flush()`

## 8.2 MODBUS-RTU

本节主要介绍由工具末端、控制柜读取或写入 MODBUS-RTU 从站线圈或寄存器相关的一些指令。

### 8.2.1 工具末端读取 MODBUS-RTU 从站线圈

```
tool_modbus_read_bits(slave, addr, len, time_out=5)
```

**功能：** 该指令用于机器人工具末端 RS-485 读取 MODBUS-RTU 从站线圈，MODBUS 功能码为 0x01。

**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

### 8.2.2 工具末端读取 MODBUS-RTU 从站输入线圈

```
tool_modbus_read_input_bits(slave, addr, len, timeout=5)
```

**功能：** 该指令用于机器人工具末端 RS-485 读取 MODBUS-RTU 从站输入线圈，MODBUS 功能码为 0x02。

**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：待读取的数据长度，范围为 1-125，当 len 大于 1 时，将会向 slave 的 addr 地址处顺序向下读，integer 型数据；  
timeout：超时时间，单位：秒，若小于等于 0，则等待时间为无限长，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_input_bits(1,1,1,0)`  
返回值：读取的从站1 线圈1 的值

### 8.2.3 工具末端读取 MODBUS-RTU 从站保持寄存器

```
tool_modbus_read_registers(slave, addr, len, time_out=5)
```

**功能：** 该指令用于机器人工具末端 RS-485 读取 MODBUS-RTU 从站保持寄存器，MODBUS 功能码为 0x03。

**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

### 8.2.4 工具末端读取 MODBUS-RTU 从站输入寄存器

```
tool_modbus_read_input_registers(slave, addr, len, timeout=5)
```

**功能：** 该指令用于机器人工具末端 RS-485 读取 MODBUS-RTU 从站输入寄存器，MODBUS 功能码为 0x04。

**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：待读取的数据长度，范围为 1-125，当 len 大于 1 时，将会向 slave 的 addr 地址处顺序向下读，integer 型数据；  
timeout：超时时间，单位：秒，若小于等于 0，则等待时间为无限长，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_input_registers (1,1,1,0)`  
返回值：读取的从站1寄存器1的值

### 8.2.5 工具末端写入单个 MODBUS-RTU 从站线圈

```
tool_modbus_write_single_bit(slave, addr,data, timeout=5)
```

**功能：** 该指令用于机器人工具末端 RS-485 写入单个 MODBUS-RTU 从站线圈，MODBUS 功能码为 0x05。

**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 型数据；  
timeout：超时时间，单位为秒，若小于等于 0，则等待超时时间为无限长，float 型数据。

**返回值：** Integer 型数据，写入成功的数据数量，即 1，小于等于 0 为写入失败。

**示例：** `read_data=tool_modbus_write_single_bit(1,1,1)`  
返回值：1

### 8.2.6 工具末端写入多个 MODBUS-RTU 从站线圈

```
tool_modbus_write_bits(slave, addr, data, timeout=5)
```

**功能：** 该指令用于机器人工具末端 RS-485 写入多个 MODBUS-RTU 从站线圈，MODBUS 功能码为 0x0F。

**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，boolean、integer（0 会被隐式转换为 False，非 0 元素会被隐式转换为 True）、list 型数据（列表的每个元素需为 integer 或 boolean）。当数据为 list 时，将会向 slave 的 addr 地址处顺序向下写；  
timeout：超时时间，单位为秒，若小于等于 0，则等待超时时间为无限长，float 型数据。

**返回值：** 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

### 8.2.7 工具末端写入单个 MODBUS-RTU 从站寄存器

```
tool_modbus_write_single_register(slave, addr,data, timeout=5)
```

**功能：** 该指令用于机器人工具末端 RS-485 写入单个 MODBUS-RTU 从站寄存器，MODBUS 功能码为 0x06。

**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 型数据；  
timeout：超时时间，单位为秒，若小于等于 0，则等待超时时间为无限长，float 型数据。

**返回值：** Integer 型数据，写入成功的数据数量，即 1，小于等于 0 为写入失败。

**示例：** `read_data=tool_modbus_write_single_register(1,1,1)`  
返回值：1

### 8.2.8 工具末端写入多个 MODBUS-RTU 从站保持寄存器

```
tool_modbus_write_registers(slave, address, data, timeout=5)
```

**功能：** 该指令用于机器人工具末端 RS-485 写入多个 MODBUS-RTU 从站保持寄存器，MODBUS 功能码为 0x10。

**参数：** slave: 从站，范围为 1-247，integer 型数据；  
address: 地址，范围为大于等于 0，integer 型数据；  
data: 待写入的数据，integer[0, 65535]、list 型数据（列表的每个元素需为 integer[0, 65535]）。当数据为 list 时，将会向 slave 的 addr 地址处顺序向下写；  
timeout: 超时时间，单位为秒，若小于等于 0，则等待超时时间为无限长，float 型数据。

**返回值：** 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

### 8.2.9 控制柜读取 MODBUS-RTU 从站线圈

```
masterboard_modbus_rtu_read_bits(slave, address, read_count, timeout = 3.0)
```

**功能：** 该指令用于机器人控制柜 RS-485 读取 MODBUS-RTU 从站线圈，MODBUS 功能码为 0x01。

**参数：** slave: 从站，范围为 1-247，integer 型数据。  
address: 地址，大于等于 0，integer 型数据。  
read\_count: 待读取的数据长度，范围为 1-125，integer 型数据。大于 1 时，将会从 slave 的 addr 开始依次向下读取。  
timeout: 超时时间，单位为秒，若小于等于 0，则等待超时时间为无限长，默认为 5 秒，float 型数据（可选参数）。

**返回值：** list 型数据，读取的数据列表。

**示例：** `masterboard_modbus_rtu_read_bits(1, 0, 2, 1)`  
返回值: [0, 0]

### 8.2.10 控制柜读取 MODBUS-RTU 从站输入线圈

```
masterboard_modbus_rtu_read_input_bits(slave, address, read_count, timeout = 3.0)
```

**功能：** 该指令用于机器人控制柜 RS-485 读取 MODBUS-RTU 从站输入线圈，MODBUS 功能码为 0x02。

**参数：** slave：从站，范围为 1-247，integer 型数据。  
address：地址，大于等于 0，integer 型数据。  
read\_count：待读取的数据长度，范围为 1-125，integer 型数据。大于 1 时，将会从 slave 的 addr 开始依次向下读取。  
timeout：超时时间，单位：秒，若小于等于 0，则等待时间为无限长，默认为 5 秒，float 型数据（可选参数）。

**返回值：** list 型数据，读取的数据列表。

**示例：** `read_data=masterboard_modbus_rtu_read_input_bits(1, 1, 1, 0)`  
返回值：读取的从站1线圈1的值

### 8.2.11 控制柜读取 MODBUS-RTU 从站保持寄存器

```
masterboard_modbus_rtu_read_registers(slave, address, read_count, timeout = 3.0)
```

**功能：** 该指令用于机器人控制柜 RS-485 读取 MODBUS-RTU 从站保持寄存器，MODBUS 功能码为 0x03。

**参数：** slave：从站，范围为 1-247，integer 型数据。  
address：地址，范围为大于等于 0，integer 型数据。  
read\_count：待读取的数据长度，范围为 1-125，integer 型数据。大于 1 时，将会从 slave 的 addr 开始依次向下读取。  
timeout：超时时间，单位为秒。若小于等于 0，则等待超时时间为无限长，默认为 5 秒，float 型数据（可选参数）。

**返回值：** list 型数据，读取的数据列表。

**示例：** `masterboard_modbus_rtu_read_registers(1, 0, 2, 1)`  
返回值：[0, 0]

### 8.2.12 控制柜读取 MODBUS-RTU 从站输入寄存器

```
masterboard_modbus_rtu_read_input_registers(slave, address, read_count, timeout = 3.0)
```

**功能：** 该指令用于机器人控制柜 RS-485 读取 MODBUS-RTU 从站输入寄存器，MODBUS 功能码为 0x04。

**参数：**  
slave: 从站，范围为 1-247，integer 型数据。  
address: 地址，大于等于 0，integer 型数据。  
read\_count: 待读取的数据长度，范围为 1-125。大于 1 时，将会从 slave 的 addr 开始依次向下读取，integer 型数据。  
timeout: 超时时间，单位：秒。若小于等于 0，则等待时间为无限长，默认为 5 秒，float 型数据（可选参数）。

**返回值：** list 型数据，读取的数据列表。

**示例：**  
`read_data=masterboard_modbus_rtu_read_input_registers(1, 1, 1, 0)`  
返回值：读取的从站1寄存器1的值

### 8.2.13 控制柜写入单个 MODBUS-RTU 从站线圈

```
masterboard_modbus_rtu_write_single_bit(slave, address, data, timeout = 3)
```

**功能：** 该指令用于机器人控制柜 RS-485 写入单个 MODBUS-RTU 从站线圈，MODBUS 功能码为 0x05。

**参数：**  
slave: 从站，范围为 1-247，integer 型数据。  
address: 地址，大于等于 0，integer 型数据。  
data: 待写入的数据，integer 型数据。  
timeout: 超时时间，单位为秒。若小于等于 0，则等待超时时间为无限长，默认为 5 秒，float 型数据（可选参数）。

**返回值：** integer 型数据，成功写入寄存器的数量，小于等于 0 表示写入失败。

**示例：**  
`read_data=masterboard_modbus_rtu_write_single_bit(1, 1, 1, 3)`  
返回值：1

### 8.2.14 控制柜写入多个 MODBUS-RTU 从站线圈

```
masterboard_modbus_rtu_write_bits(slave, address, data, timeout = 3)
```

**功能：** 该指令用于机器人控制柜 RS-485 写入多个 MODBUS-RTU 从站线圈，MODBUS 功能码为 0x0F。

**参数：**  
slave: 从站，范围为 1-247，integer 型数据。  
address: 地址，大于等于 0，integer 型数据。  
data: 待写入的数据，boolean、integer (0 会被隐式转换为 False，非 0 元素会被隐式转换为 True)、list 型数据 (列表的每个元素需为 integer 或 boolean)。当数据为 list 时，将会向 slave 的 addr 地址处顺序向下写；  
timeout: 超时时间，单位为秒。若小于等于 0，则等待超时时间为无限长，默认为 3 秒，float 型数据 (可选参数)。

**返回值：** integer 型数据，成功写入线圈的数量，小于等于 0 表示写入失败。

**示例：**  
`masterboard_modbus_rtu_write_bits(1, 0, [1], 3)`  
返回值: 1

### 8.2.15 控制柜写入单个 MODBUS-RTU 从站寄存器

```
masterboard_modbus_rtu_write_single_register(slave, address, data, timeout = 3)
```

**功能：** 该指令用于机器人控制柜 RS-485 写入单个 MODBUS-RTU 从站寄存器，MODBUS 功能码为 0x06。

**参数：**  
slave: 从站，范围为 1-247，integer 型数据。  
address: 地址，大于等于 0，integer 型数据。  
data: 待写入的数据，integer 型数据。  
timeout: 超时时间，单位为秒。若小于等于 0，则等待超时时间为无限长，默认为 5 秒，float 型数据 (可选参数)。

**返回值：** integer 型数据，成功写入寄存器的数量，小于等于 0 表示写入失败。

**示例：**  
`read_data=masterboard_modbus_rtu_write_single_register(1, 1, [1], 3)`  
返回值: 1

### 8.2.16 控制柜写入多个 MODBUS-RTU 从站保持寄存器

```
masterboard_modbus_rtu_write_registers(slave, address, data, timeout = 3)
```

**功能：** 该指令用于机器人控制柜 RS-485 写入多个 MODBUS-RTU 从站保持寄存器，MODBUS 功能码为 0x10。

**参数：** slave：从站，范围为 1-247，integer 型数据。

address：地址，大于等于 0，integer 型数据。

data：待写入的数据，integer[0, 65535]、list 型数据（列表的每个元素需为 integer[0, 65535]）。当数据为 list 时，将会向 slave 的 addr 地址处顺序向下写。

timeout：超时时间，单位为秒，若小于等于 0，则等待超时时间为无限长，默认为 5 秒，float 型数据（可选参数）。

**返回值：** integer 型数据，成功写入寄存器的数量，小于等于 0 表示写入失败。

**示例：** `masterboard_modbus_rtu_write_registers(1, 0, [12], 3)`

返回值：1

## 第 9 章 系统指令

### 9.1 线程

#### 9.1.1 启动线程

```
start_thread(function, args=(), detach_mode=True)
```

功能： 该指令用于启动线程。

参数： function：线程执行的函数。callable 型数据。

arg：传递至 function 的参数。tuple 型数据。该参数为可选参数，默认为'()' 无参数。

detach\_mode：设置线程启动模式(True: 分离模式, False: 非分离模式)。boolean 型数据，该参数为可选参数，默认为 True。

返回值：新创建的线程 ID，该 ID 可用于作为 stop\_thread 与 join\_thread 函数参数，实现停止线程与等待线程停止功能。

示例：

```
def thread_test(a, b):
 textmsg("Run thread: ", a+b)
 sleep(1)
 textmsg("Thread end.")
global thread_id
thread_id = start_thread(thread_test, (1, 2), False)
sleep(2)
参数：
function = thread_test -> 以线程的形式执行'thread_test'函数
args = (1, 2) -> 传递参数(1, 2)至'thread_test'函数
detach_mode = False -> 以非分离模式运行
```

#### 9.1.2 等待线程结束

```
join_thread(thread_id)
```

功能： 该指令用于等待线程 ID 为给定参数的线程结束。

参数： thread\_id: 启动线程返回的线程 ID。

返回值： 无

```
示例：
def thread_test(a, b):
 textmsg("Run thread: ", a+b)
 sleep(1)
 textmsg("Thread end.")
global thread_id
thread_id = start_thread(thread_test, (1, 2), False)
join_thread(thread_id)
```

### 9.1.3 关闭线程

```
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)
```

## 9.2 任务运行

### 9.2.1 暂停任务运行

```
pause()
```

功能： 该指令用于暂停任务运行。

参数： 无

返回值： 无

```
示例：
pause()
暂停任务运行
```

## 9.2.2 停止任务运行

```
halt()
```

功能： 该指令用于停止任务运行。

参数： 无

返回值： 无

示例：  
`halt()`  
停止任务运行

## 9.3 提示警告

### 9.3.1 发送字符串至 EliRobot

```
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。

### 9.3.2 弹框

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

title: 标题字符串，支持 Python 内置数据类型：string、list、boolean、integer、float 等数据类型；

warning: 警告消息，boolean 型数据；

error: 错误消息，boolean 型数据；

blocking: 是否为阻塞式弹出窗口（sec 函数中强制为非阻塞式弹出窗口）。

返回值： 无

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

## 9.4 字符

### 9.4.1 字符串索引

```
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
```

### 9.4.2 字符串拼接

```
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"
```

### 9.4.3 空字符串判断

```
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
```

### 9.4.4 字符串匹配

```
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
```

### 9.4.5 字符串长度

```
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
```

### 9.4.6 获取子字符串

```
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  $index+len > src\ length$ ），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  $[0, src\ length]$  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  $[索引, src\ 长度]$  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"
```

## 9.4.7 字符串转数字

```
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。需要注意的是，字符串开头和结尾的空白字符 (“\r”, “\n”, “\t” 以及空格) 会被无视。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
```

### 9.4.8 数字转字符串

```
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]"
```

## 9.5 其他

### 9.5.1 休眠

```
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)

### 9.5.2 设置系统内部标记位的值

```
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 I/O，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)`

### 9.5.3 获取系统内部标记位的值

`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 I/O，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

### 9.5.4 获取机器人步长时间

`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

### 9.5.5 获取控制器系统时间戳

```
controller_timestamp()
```

功能： 该指令用于获取控制器系统时间戳（单位：s）。

参数： 无

返回值： 控制器系统时间戳。

示例：

```
import time
获取控制器时间戳的接口
timestamp = controller_timestamp()
转换为本地时间结构
time_local = time.localtime(timestamp)
格式化并输出到示教器日志
textmsg(time.strftime("%D %H:%M:%S", time_local))
```

## 9.5.6 启用看门狗功能

```
rtsi_set_watchdog(variable, frequency, action)
```

功能： 该指令用于启用 RTSI 的看门狗功能，可监视 RTSI 输入变量的设置频率。若其监视到的频率小于设置频率，运行中的程序将会触发“无操作”、“暂停”或“停止”动作。一旦程序停止，所有被监视的变量将不再被监视。

参数： variable：待监视的变量名，字符串数据，例如：“speed\_slider\_mask”；

frequency：指定的频率；float 型数据。

action：触发的动作，字符串数据，例如：“ignore”、“pause”、“stop”。

返回值： 无

示例： `rtsi_set_watchdog("speed_slider_mask", 10, "pause")`

## 9.5.7 关闭电源

```
powerdown ()
```

功能： 该指令用于停止机器人运行并关闭机器人和控制柜的电源。

参数： 无

返回值： 无

示例： `powerdown()`





# 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 10 号院 5 号楼 611 室

## - 上海公司 (研创中心)

上海市浦东新区张江人工智能岛川和路 55 弄 20 号楼 3 层

## - 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