AR 语言指令手册

AR 机器人语言

1 AR 语言总览

指令类别	指令符号	指令说明
算术运算符	+	加法运算
	-	减法运算
	*	乘法运算
	/	除法运算
	//	取整除法，即得到不大于结果的最大整数值
	%	取余除法
	^	指数运算
	-	取负运算
	~	异或运算
	&	与运算
	|	或运算
	~	取反运算
	<<	左移运算
	>>	右移运算
关系运算符	==	等于
	~=	不等于
	<=	小于等于
	>=	大于等于
	<	小于
	>	大于
逻辑运算符	or	逻辑或
	not	逻辑非
	and	逻辑与
	false	假 （注：nil 也为 false）
	true	真
一般符号	#	求 table 数组长度
	=	赋值操作
	--	单行注释
	--[[	多行注释开始行
	--]]	多行注释结束行
	( )	用于函数定义、调用，表达式运算
	{ }	用于定义 table 数组
	[ ]	table 数组元素操作符
	::	定义 goto 指令的跳转位置标签
	;	语句结束符，可以省略不写
	,	用于函数参数定义、调用，多变量赋值，table 数组定义等

	.	用于 table 数组元素访问
	..	字符串连接符
	...	定义函数可变参数项
过程控制指令	if then else elseif end	if 条件分支指令
	while do end	while 循环控制指令
	for do end	for 循环控制指令
	repeat until	repeat 循环控制指令
	goto	goto 无条件跳转指令
	function end	用户函数定义指令
一般关键字	break	跳出 for、while、repeat 循环
	local	定义局部变量
	nil	变量为空值
	return	函数调用返回
机器人轴定义	AX	笛卡尔坐标系 X 轴号
	AY	笛卡尔坐标系 Y 轴号
	AZ	笛卡尔坐标系 Z 轴号
	AC	笛卡尔坐标系 C 轴号
	J1	J1 关节
	J2	J2 关节
	J3	J3 关节
	J4	J4 关节
字符串操作	string.find	函数用于在一个给定的目标字符串中搜索一个模式，返回该模式所在的位置
	string.match	用于在一个字符串中搜索一种模式，返回的是目标字符串中与模式相匹配的那部分字符串
	string.sub	截取字符串 s 的从第 i 个字符到第 j 个字符之间的字符串
	string.gsub	将目标字符串中所有出现模式的地方替换为目标字符串
	string.char,string.byte	用于转换字符和对应的数字
	string.format	用来对字符串格式化，与 C 语言中的 printf()函数类似
	tonumber	将数字组成的字符串 string 变量转化成数字 number 类型
机器人全局变量	ON	打开状态
	OFF	关闭状态
	p0~p2999	机器人默认点名称
运动指令	MovL	直线方式运行到笛卡尔坐标系绝对位置的指令
	MovLR	直线方式运行到笛卡尔坐标系相对位置的指令
	MovP	点到点方式运行到笛卡尔坐标系绝对位置的指令
	MovPR	点到点方式运行到笛卡尔坐标系相对位置的指

		令
	MovJ	控制各个关节移动到目标绝对角度的指令
	MArchP	点到点方式控制机器人进行拱形移动的指令
	MArc	从当前位置圆弧插补到笛卡尔坐标系绝对位置的指令
	MCircle	笛卡尔坐标系下的整圆插补指令
	MSpline	样条曲线插补运动指令
	handmove	自动运行状态下实现手动笛卡尔/关节的连续、点动运动
	stoprun	自动运行状态下实现停止手动直线运动
	waitpos	等待运动脉冲发送完成指令（运动 DSP 发送脉冲完成指令）
	waitrealpos	机器人运动到位（停稳）指令（伺服响应脉冲到位指令）
运动参数设置指令	AccJ	设置加速度比例指令，影响 MovJ、MovP、MovPR、MArchP 指令的加速时间
	DecJ	设置减速度比例指令，影响 MovJ、MovP、MovPR、MArchP 指令的减速时间
	SpdJ	设置速度比例指令，影响 MovJ、MovP、MovPR、MArchP 指令的运行速度
	AccL	设置直线运动指令加速度，影响 MovL、MovLR、MArc、MCircle 指令的加速时间。单位：mm/s^2
	DecL	设置直线运动指令减速度，影响 MovL、MovLR、MArc、MCircle 指令的减速时间。单位：mm/s^2
	SpdL	设置直线运动指令速度，影响 MovL、MovLR、MArc、MCircle 指令的运行速度。单位：mm/s
程序管理指令	Delay	延时指令。单位：毫秒
	Exit	程序运行终止
	Pause	暂停程序运行
一般指令	X	创建指定 X 轴笛卡尔坐标绝对位置点的指令
	Y	创建指定 Y 轴笛卡尔坐标绝对位置点的指令
	Z	创建指定 Z 轴笛卡尔坐标绝对位置点的指令
	C	创建指定 C 轴笛卡尔坐标绝对位置点的指令
	XYZC	创建指定 XYZC 轴笛卡尔坐标绝对位置点的指令
输入/输出指令	DI	读取输入端口状态
	DO	输出端口打开或关闭操作
	WDI	读取输入端口状态,直到等待某一信号有效，则继续执行后续的程序
	WDO	读取输出端口状态,直到等待某一信号有效，则继续执行后续的程序
坐标系指令	SetU	设置机器人当前用户坐标系

	WrU	修改用户坐标系的数据
	SetT	设置机器人当前工具坐标系
	WrT	修改工具坐标系的数据
	CacU	新建用户坐标系
	U2U	用户（0~9）之间坐标相互转换
	V2Tool	根据视觉坐标计算工具坐标
	CacT	两点法新建工具坐标系
	getcart	获取当前笛卡尔坐标以及关节坐标
	getjoint	计算目标点对应的某一个轴的关节坐标
	encoderget	获取编码器的脉冲值
	targetok	判断目标位置是否为机器人可到达点位
码垛指令	SetPlt	设置码垛数指令
	GetPlt	取码垛数据点指令
	SET_PLT	设置码垛数指令（用于编程码垛）
	GET_PLT	取码垛数据点指令（用于编程码垛）
	GetPLTPos	获取码盘是否码满、码盘当前码垛的个数以及码垛点位置信息（适用于配置码垛）
	ResetPLT	重置码垛个数
	SetArcPlt	圆形码垛设置
	GetArcPlt	获取圆形码垛当前位置
伺服管理指令	MotOn	伺服所有轴上电使能
	MotOff	关闭伺服所有轴使能
	DragMode	AR 程序实现拖拽功能
通信指令	RecCom	从 RS232 串口接收数据
	ClrCom	清除 RS232 串口接收缓冲区
	sysnetclr	清除网络数据缓冲区
	sysnetget	查询方式读取网络数据
	sysnetsend	从控制器向外围设备发送网络数据
	sysnetcatch	阻塞式读取网络数据
	CloseNet	关闭 TCP 网络连接
	OpenNet	创建 TCP 网络
	ConnectNet	连接TCP网络
	RecvNet	网络接收数据功能函数
	WriteNet	网络发送数据功能函数
	publicread	读取全局表数据值
	publicwrite	写入数据值到全局表
视觉指令	initTCPnet	网络通讯参数初始化
	CCDrecv	接收视觉返回的坐标数据
	CCDtrigger	触发相机拍照
	CCDsent	发送字符串到视觉
	CCDclr	清除网络托管的 IP
	CCDoffset	视觉偏差补偿

	GetDynCCDPos	动态视觉位置计算
	CCDGet	接收视觉返回的字符串类型数据
动态跟随指令	FollowInit	动态跟随参数初始化
	SetDynCatch	开启或关闭跟随抓取任务
	GetCatchSpace	获取物件是否进入抓取区域
	SetCatch	执行跟随
	GetCatchState	获取抓取状态
	SynOover	结束跟随
	GetTrigger	获取触发状态（同一物体拍两次使用）
	SetViewData	发送数据给控制器，存入缓存待跟随
	FastCatch	动态跟随快速抓取，拱形运动到抓取位置
	GetSynPos	获取进入抓取区工件的信息
调试指令	print	打印输出用户调试数据
	Error	终止程序运行，并输出错误信息
点位指令	Point	调用点位表中的点位
	new	将要更新的点位写入到对应工程中的点位表里
	teach	将当前点位示教到对应工程中的点位表里
	ReadTabDat	加载当前工程中的点位数据表
系统指令	syswork	设置系统工作状态
	sysstate	读取系统状态
	sysrate	设置全局速度倍率
	systime	获取系统时钟
Modbus 主站读写指令	ReadRegW	读指定 PLC 寄存器的 16 位字地址
	ReadRegDW	读指定 PLC 寄存器的 32 位双字地址
	WriteRegW	写入 16 位字地址到指定的 PLC 寄存器
	WriteRegDW	写入 32 位字地址到指定的 PLC 寄存器
文件操作指令	fopen	文件打开
	fsize	文件大小
	fwrite	写文件，写数据到文件中
	fread	读文件，从文件中读取数据
	fgets	逐行获取文件中的数据
	fseek	文件定位，把文件指针地址移动到一个指定位置
	feof	文件结束
	fclose	文件关闭
队列操作指令	qexist	判断队列是否存在
	qcreate	创建队列
	qpush	往队列中写入数据
	qpop	删除队列中的首元素
	qfront	取队列中的首元素
	qpopfront	取队列中的首元素然后从队列中删除
	qempty	判断队列是否为空

	qsize	队列大小
	qdestroy	删除队列

◆ 注意：AR 语言是大小写敏感的语言，用户一定按照规定的操作符使用，表中的所有指令符号用户只能根据说明使用，不能重新定义。例如：

```
local X   --定义变量 X
X=10      --赋值给变量 X
```

X 已经定义为系统指令，用户重新定义可能会运行出错。

1.1 算术运算符

指令符号	指令说明
+	加法运算
-	减法运算
*	乘法运算
/	除法运算
//	取整除法，即得到不大于结果的最大整数值
%	取余除法
^	指数运算
-	取负运算
~	异或运算
&	与运算
|	或运算
~	取反运算
<<	左移运算
>>	右移运算

算术运算符提供了各种实数的运算功能，整形数据的位运算功能等。

“+” 加法运算还可以实现两个点位数据的加法运算。

举例说明:

```
local point1=p1
local point2=p2
local point3=p3
```

注：自定义的数据点位不能直接相加。能直接相加的数据点位，必须是机器人系统定义的数据点位（p0~p2999）。

1.2 关系运算符

指令符号	指令说明
==	等于
~=	不等于
<=	小于等于
>=	大于等于

<	小于
>	大于

关系运算符用在流程控制语句的条件判断中。

举例说明：

```
local    a =10
local    b=20
if a == b then
……
end
if a < b then
……
end
if a < 30 then
……
end
```

== 等号可以比较字符串类型变量。

```
local    a = “ROBOT”
if    a == “ROBOT”    then
……
end
```

1.3 逻辑运算符

指令符号	指令说明
or	逻辑或
not	逻辑非
and	逻辑与
false	假（注：nil 也为假）
true	真

逻辑运算符用在流程控制语句的条件判断中。

举例说明：

```
if    5 or 6 then      --真
……
end

local    a = 30
local    b = true
if a and b then      --真
……
end

local a = 0
local b = 1
if a and b then      --真
…..
end

local a = 0
local b = nil
if a and b then      --假
……
end
```

◆ 注意：AR 语言中，只有 nil 和 false 值为假，其它值都为真，包括 0 也为真。

1.4 一般符号

指令符号	指令说明
#	求 table 数组中元素的个数或计算字符串的长度
=	赋值操作
--	单行注释
--[[	多行注释开始行
--]]	多行注释结束行
( )	用于函数定义、调用，表达式运算
{ }	用于定义 table 数组
[ ]	table 数组元素操作符
::	定义 goto 指令的跳转位置标签
;	语句结束符，可以省略不写
,	用于函数参数定义、调用，多变量赋值，table 数组定义等
.	用于 table 数组元素访问
..	字符串连接符
…	定义函数可变参数项

举例说明：

```
1.  print(#("123,456"))          --返回值为 7（字符串）
    print(#{123,456})            --返回值为 2（数组）
2.  local a,b,c = 1,2,3          --用,符号写多变量赋值语句
3.  p.x                          --用.符号访问数组元素
4.  local a={10,20,30}           --用{}符号定义数组
5.  a[1]                         --用[]符号访问数组元素
6.  ::lab::                      --用::符号定义 goto 指令跳转位置标签
```

◆ 注意：table 类型数组变量下标是从 1 开始，如 a[1]的值为 10。

1.5 一般关键字

指令符号	指令说明
break	跳出 for、while、repeat 循环
local	定义局部变量
global.*	同一个工程中，多个 CPU 之间变量的共用
nil	变量为空值
return	函数调用返回

local 符号用来声明局部变量

举例说明：

```
local   a                        --声明局部变量 a ，值为 nil
local   b={10,10}                --声明 table 数组
local   b={{10,20},{30,40}}      --声明 table 二维数组
local   a="ROBOT"                --声明字符串变量 a
```

➢ 介绍局部变量之前，首先来介绍一下 lua 语言中，代码块内全局变量是如何定义的；

全局变量不需要声明，给一个变量赋值后即创建了这个全局变量，访问一个没有初始化的全局变量也不会出错，只不过得到的结果是：nil

```
print(b)    -->nil
b = 10
print(b)    -->10
--如果想删除一个全局变量，只需要将变量赋值为 nil;这样变量 b 就好像从来没被使用过一样，换句话说，当且仅当一个变量不等于 nil 时，这个变量存在。
b =nil
print(b)    -->nil
```

➢ 使用 local 创建一个局部变量，与全局变量不同，局部变量只在被声明的那个代码块内有效。代码块：指一个控制结构内，一个函数体，或者一个 chunk（变量被声明的那个文件或者文本串）。

```
x = 10
local i = 1                -- local to the chunk
while i<=x do
      local x = i*2        -- local to the “while” body
      print(x)             --> 2, 4, 6, 8, ...
      i = i + 1
end
if i > 20 then
      local x              -- local to the “then” body
      x = 20
      print(x +2)          -->22(the local one)
else
      print(x)             --> 10 (the global one)
end
print(x)                   --> 10 (the global one)
```

global.* 根据实际应用工艺需求，一个工程须包含多个 CPU。公共变量（global.*）可解决多个 CPU 之间共用同一个变量的问题。

使用说明：

- **在各个 CPU 中，公共变量定义为 global.*的形式，例如：global.var, global.a,global.b;**
- **公共数组变量里不能再嵌套数组，例如：global={pos={x=0,y=0},a=1,b=2} 非法；**
- **全局（公共）变量仅限于一个工程中多个 CPU 之间变量的共用，单个 CPU 中全局变量的使用可参考 AR 编程手册中的 local 指令的使用；**

1.6 基本函数库

函数名	描述	示例	结果
math.pi	圆周率	math.pi	3.1415926
math.abs	取绝对值	math.abs(-2012)	2012
math.ceil	向上取整	math.ceil(9.1)	10
math.floor	向下取整	math.floor(9.9)	9
math.max	取参数最大值	math.max(2,4,6,8)	8
math.min	取参数最小值	math.min(2,4,6,8)	2
math.pow	计算 x 的 y 次幂	math.pow(2,16)	65536
math.sqrt	开平方	math.sqrt(65536)	256
math.fmod	取模	math.fmode(14,5)	4
math.modf	取整数和小数部分	math.modf(20.12)	20　0.12
math.randomseed	设随机数种子	math.randomseed(os.time())	

math.random	随机坐标系	math.random(5,90)	5~90
math.rad	角度转弧度	math.rad(180)	3.1415926
math.deg	弧度转角度	math.deg(math.pi)	180
math.exp	e 的 x 次方	math.exp(4)	54.5981
math.log	计算 x 的自然对数	math.log(54.5981)	4
math.log10	计算 10 为底，x 的对数	math.log10(1000)	3
math.frexp	将参数拆成 x*（2^y）的形式	math.frexp(160)	0.625　8
math.ldexp	计算 x*(2^y)	math.ldexp(0.625,8)	160
math.sin	正弦	math.sin(math.rad(30))	0.5
math.cos	余弦	math.cos(math.rad(60))	0.5
math.tan	正切	math.tan(math.rad(45))	1
math.asin	反正弦	math.deg(math.asin(0.5))	30
math.acos	反余弦	math.deg(math.acos(0.5))	60
math.atan	反正切	math.deg(math.atan(1))	45

1.7 机器人轴定义

指令符号	指令说明
AX	笛卡尔坐标系 X 轴号
AY	笛卡尔坐标系 Y 轴号
AZ	笛卡尔坐标系 Z 轴号
AC	笛卡尔坐标系 C 轴号
J1	J1 关节
J2	J2 关节
J3	J3 关节
J4	J4 关节

轴号是全局变量，作为运动指令的一些参数使用，用户也不能重新定义。

1.8 机器人全局变量

指令符号	指令说明
ON	打开状态
OFF	关闭状态
p0~p2999	机器人默认点名称

全局变量为系统定义好的变量，有具体的意义，用户不能再重新定义或赋值，否则可能运行出错。

点位变量 p0~p2999 为 table 类型的点变量，定义如下：

```
local     p={x=VALUE1,y=VALUE2,z=VALUE3,c=VALUE4,h=VALUE5}
```

程序中，用户可以以 p.x　p.y 的方式访问点的数据值。

```
local    a = p1                    -- a 为  p1 的引用
a.x = 10                           --那么 p1.x 的值也变为 10
local    a = #p1                   --拷贝 p1 的值到 a
a.x=10                             --p1.x 保持原来的值
```

1.9 过程控制指令

指令符号	指令说明
if then else elseif end	if 条件分支指令
while do end	while 循环控制指令
for do end	for 循环控制指令
repeat until	repeat 循环控制指令
goto	goto 无条件跳转指令
function end	用户函数定义指令

1.9.1 if 条件分支指令

使用说明: if then else elseif end

语法说明:

Case1	Case2	Case3

if conditions then
then-part
end
	if conditions then
	then-part
	else
	else-part
	end
		if conditions then
		then-part
		elseif conditions then
		elseif-part
		else
		else-part
		end

Conditions 表示控制语句的条件，如果为 true 则条件成立，part 为执行的代码部分。

Conditions 条件可以是常量，变量，表达式，函数调用等，程序只判断最终结果为 false 或 true 去选择执行程序。

1.9.2 while 循环指令

使用说明: while do end

语法说明:

```
while condition do
    statements
end
```

conditions 表示控制条件，如果为 true 则条件成立，statements 为执行的代码部分，如果为 false 则条件不成立，不执行 statements 代码部分。

```
a = 15
while a>10 do                --条件判断，如果为 true，则继续执行 a = a-1
    a = a-1
end
```

1.9.3 for 循环指令

使用说明: for….. do …..end

语法说明:

```
for var=exp1,exp2,exp3 do
    loop-part
end
```

for 将用 exp3 作为 step（步进值）从 exp1（初始值）到 exp2（终止值），执行 loop-part（循环体）。其中 exp3 可以省略，默认 step=1

exp 是一个表达式，可以是数值常量，变量，或函数调用的返回值，表达式运算。

有几点需要注意：

1. 三个表达式只会被计算一次，并且是在循环开始前。

```
for i=1,f(x) do              --调用 f(x)函数一次，函数返回值作为循环的终止值
    print(i)
end
for i=10,1,-1do
    print(i)
end
```

2. 控制变量 var 是局部变量自动被声明,并且只在循环内有效.

```
for i=1,10 do
    print(i)
end
max =i                                --错误引用 i，i 是局部变量
```

如果需要保留控制变量的值，需要在循环中将其保存

```
local found = nil
for i=1,a.n do
    if a[i] == value then
        found = i
        break
    end
end
print(found)
```

3. 循环过程中不要改变控制变量的值，那样做的结果是不可预知的。如果要退出循环，使用 break 语句。

1.9.4 repeat–until 循环指令

使用说明: repeat-until
语法说明:

```
repeat
    statements
until conditions
```

while 语句大致相同，只是循环部分 statements 的执行先后不一样，repeat-until 的循环体先执行在判断循环条件，而 while 语句是先判断循环条件。

1.9.5 goto 无条件跳转指令

使用说明 goto
语法说明 goto lab_name
其中 lab_name 用户定义的文件行跳转位置名称，字符串类型，如果没有定义会出错。

```
local a = 0
::lab::                          --定义跳转位置名称 lab
MovL(p0)
MovL(p1)
a = a + 1
print("跳转次数:",a)
goto lab                         --跳转到示例 2 循环执行
```

◆ 注意：goto 指令不能从一个函数跳转到另外一个函数,lab_name 跳转名称不能重复定义。

1.9.6 函数指令

使用说明 function …… end
语法说明 function func_name (arguments-list)

```
function func_name(arguments-list)
    statements-list        --函数体部分
end
```

func_name 是函数名字，根据 AR 语言命名规则进行定义，且函数名不能重复，否则会出错。

arguments-list 为函数的参数列表，列表中的参数可以是任何数据类型的变量，当有多个参数时，通过","分隔。函数还可以没有参数，在定义的时候参数列表为空即可，但是括号不能省略。

statements-list 为函数体部分，函数体就是实现该函数功能的具体程序细节，函数体结束部分可以有返回值，也可以没有返回值，如果有返回值则通过**关键字 return** 说明。

例如：定义一个加法功能函数:

```
function add (a,b)
    return a+b
end
add(1,2)   --函数相加并返回结果 3
```

返回多个值，求相加与相乘函数

```
function addmul(a,b)
    return a+b,a*b
end
addmul(2,5)                          --函数相加相乘，返回 7   10
```

注意 1：

function main 主函数的使用方法，当 AR 程序中包含有 function main 主函数是，子函数既可以放在 function main 函数的前面定义也可以放在后面定义，例如：

```
function addmul(a,b)
    return a+b,a*b
end
function main()
    while true do
        addmul(2,5)                  --函数相加相乘，返回 7   10
    end
end
```

```
function main()
    while true do
        addmul(2,5)                  --函数相加相乘，返回 7   10
    end
end
function addmul(a,b)
    return a+b,a*b
end
```

注意 2：

当 AR 程序中没有主函数时，子函数一定要放在调用前定义，否则程序出错，例如：

```
function addmul(a,b)
    return a+b,a*b
end

while true do
    print("开始运行")
    addmul(2,5)
end
```

运行正常

```
while true do
    print("开始运行")
    addmul(2,5)
end
function addmul(a,b)
    return a+b,a*b
end
```

运行异常

1.10 字符串操作

指令符号	指令说明
string.find	函数用于在一个给定的目标字符串中搜索一个模式，返回该模式所在的位置
string.match	用于在一个字符串中搜索一种模式，返回的是目标字符串中与模式相匹配的那部分字符串
string.sub	函数截取字符串 s 的从第 i 个字符到第 j 个字符之间的串
string.gsub	将目标字符串中所有出现模式的地方替换为目标字符串
string.char,string.byte	用于转换字符和对于的数字之间值
string.format	用来对字符串格式化，与 C 语言中的 printf()函数类似
tonumber	将数字组成的字符串 string 变量转化成数字 number 类型

1. **string.find()** 函数用于在一个给定的目标字符串中搜索一个模式。最简单的模式就是一个单词，它只会匹配与自己完全相同的拷贝。当 find 找到一个模式后，它会返回两个值：匹配到的起始索引和结尾索引；如果没有找到任何匹配，它就返回 nil。 示例代码：

```
local str = "Hello World"
local i,j = string.find(str, "Hello")      --返回 Hello 在 str 中的起始位置和终止位置
print(i,j)      --> 1    5
```

2. **string.match()** 与 string.find()非常相似，它也是用于在一个字符串中搜索一种模式。区别在于，string.match 返回的是目标字符串中与模式相匹配的那部分字符串，并不是该模式

所在的位置。示例代码：

```
local str = “Hello12345World”
local subStr = string.match(str, “%d+”)
print(subStr)      --> 1 2 3 4 5

local i,j = string.find(str, “%d+”)
subStr = string.sub(str,i,j)
print(subStr)      --> 1 2 3 4 5
```

3. **string.sub(s,i,j)** 函数截取字符串 s 的从第 i 个字符到第 j 个字符之间的串. Lua 中，字符串的第一个字符索引从 1 开始。你也可以使用负索引，负索引从字符串的结尾向前计数：-1 指向最后一个字符，-2 指向倒数第二个，以此类推。示例代码：

```
s = “[in brackets]”
b = string.sub(s,2,-2)
print(b)      --> in brackets
```

4. **string.gsub(s, pattern,reps)**有三个参数，给定字符串，匹配模式和替代字符串。作用就是将所有符合匹配模式的地方都替换成替代字符串。并返回替换后的字符串，以及替换次数。示例代码：

```
s = string.gsub("Lua is cute", "cute", "great")
print(s)                  --> Lua is great
```

5. **string.char() &&string.byte()** 用于转换字符和对应的数字之间值。示例代码：

```
i = 97
print(string.char(i, i+1, i+2))    --> abc
print(string.byte("abc"))          --> 97
print(string.byte("abc"), -2)      --> 98
```

6. **string.format()** 的功能就是格式化一个字符串。第一个参数为字符串格式，后面的参数可以任意多个，用于填充第一个参数中的格式控制符，最后返回完整的格式化后的字符串。示例代码:

```
str = string.format(“%0.2f”, 34.2344)
print(str)      --> 34.23
str = string.format(“0x%08X”, 348)
print(str)      --> 0x0000015C
```

7. **tonumber()** 函数的功能是将数字组成的字符串变量转化成数字类型，如果要转换的 string 不能被转为 number，它返回 nil。示例代码：

```
1. Str1 = “12”
   A = tonumber(Str1)
   print(A)          --12
2. Str2=”123,456”
   B=tonumber(Str2)
```

```
    print(B)          --nil
3.  Str3 = “123,a”
    C=tonumber(Str3)
    print(C)          --nil
```

1.11 运动指令

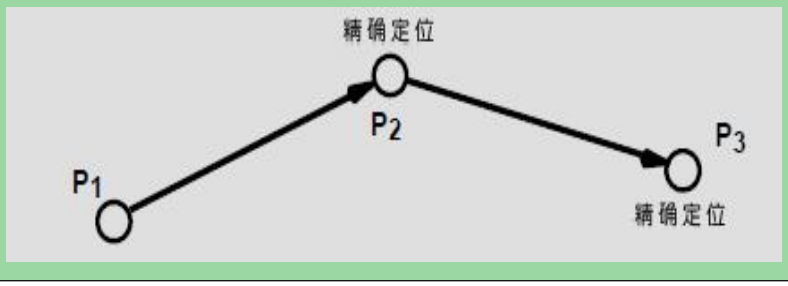
指令符号	指令说明
MovL	直线方式运行到笛卡尔坐标系绝对位置的指令
MovLR	直线方式运行到笛卡尔坐标系相对位置的指令
MovP	点到点方式运行到笛卡尔坐标系绝对位置的指令
MovPR	点到点方式运行到笛卡尔坐标系相对位置的指令
MovJ	控制各个关节移动到目标角度的指令
MArchP	点到点方式控制机器人进行拱形移动的指令
MArc	从当前位置圆弧插补到笛卡尔坐标系绝对位置的指令
MCircle	笛卡尔坐标系下的整圆插补指令
MSpline	样条曲线插补运动指令
handmove	自动运行状态下实现手动笛卡尔/关节的连续、点动运动
stoprun	自动运行状态下实现停止手动直线运动
waitpos	等待运动脉冲发送完成指令（运动 DSP 发送脉冲完成指令）
waitrealpos	机器人运动到位（停稳）指令（伺服响应脉冲到位指令）

MovL

使用说明	以直线的方式运动到笛卡尔坐标系下的目标绝对位置
语法说明	STA = MovL(A, “CP=20 Acc=20 Dec=20 Spd=100 AccC=20 SpdC=20 I=0 In=10 ON/ OFF”)
参数说明	该指令一共两个参数，第一个参数 A 为目标点，第二个参数为可选参数、省略时系统默认全局状态值

输入参数说明

A	笛卡尔坐标目标位置。可以是点的名称 p1~p2999，也可以是点的索引号 1~2999。可选参数，可以指定运动到目标位置的各项参数
CP	可选参数，说明运动到目标点时是否平滑过渡，范围为 0~100
Acc	可选参数，指定运动到目标位置的加速度，单位为 mm/s^2
Dec	可选参数，指定运动到目标位置的减速度，单位为 mm/s^2
Spd	可选参数，指定运动到目标位置的速度，单位为 mm/s
AccC	可选参数，指定运动到目标位置的姿态加速度，单位**度/s^2**
SpdC	可选参数，指定运动到目标位置的姿态速度，单位**度/s**
I	可选参数，第三轴电流设定值，单位 mA。若运动过程中的电流超过了设定值则当前运动指令停止，继续往下执行

		其它指令。
	In	可选参数，输入检测信号。若运动过程中检测到该输入信号则当前的运动停止，继续往下执行其它指令。
	ON/OFF	ON：打开；OFF：关闭
	返回值说明	
	STA	0：指令运行正常
		1：输入信号中断运行
		-1：电流到达阈值中断运行
示意图		
举例说明	1. MovL(p1) --机器人以直线的方式从当前位置运动到 p1 目标点 2. MovL(10) --机器人以直线的方式从当前位置运动到 p10 目标点 3. MovL(10, "Acc=100 Spd=1000") --机器人以直线的方式从当前位置运动到目标点 p10，其中加速度为 100mm/s2，速度为 1000mm/s 4. MovL(p20, "CP=20") --机器人以直线的方式运动到 p20 位置，其中目标位置 p20 以平滑度 20 过渡 5. MovL(p20, "In=10 ON") --机器人以直线的方式运动到 p20 目标位置，在运动过程中如果检测输入信号 In10 打开则当前指令运行结束。 6. MovL(p3, “I = 2000”) --机器人以直线的方式运动到 p3 目标位置，在运动过程中如果检测电流超过 2000mA 则当前指令运行结束。 7. MovL(p10+Z(-10)) --机器人以直线的方式从当前位置运动到 p10 点下方 10mm 的位置 8. MovL(p11+Z(10)) --机器人以直线的方式从当前位置运动到 p10 点上方 10mm 的位置 9. local pos = {x=300,y=100,z=-50,c=60} --用户定义 pos 点变量 MovL(pos+X(-10)) --机器人以直线的方式从当前位置运动到相对于 pos 点 X 负方向 10mm 的位置 10. MovL(12+Y(10)) --机器人以直线的方式从当前位置运动到相对于 p12 点 Y 正方向 10mm 的位置	

注意：

- 可选参数编程时可以根据需要是否设置，其中 Acc、Spd 省略时，系统默认全局值，当设置 In 选项触发停止时，参数 ON/OFF 才会有意义，且只能二选一设置信号有效停止或无效停止。
- 当使用 MovL 走直线偏移（例如，例 7、8、9、10）时，一定要用正号（+）连接轴实现偏移，偏移的正负取决于轴括号内的数字的正负。
- 大写 X 、Y、Z、C 分别表示 X 轴、Y 轴、Z 轴、C 轴方向的轴偏移

MovLR

使用说明	以直线的方式运动到笛卡尔坐标系下的目标相对位置
语法说明	STA = MovLR(A,B,"CP=20 Acc=20 Dec=20 Spd=100 AccC=20 SpdC=20 I = 0 In=10 ON/ OFF")
参数说明	该指令一共三个参数，第一个为笛卡尔坐标轴号，第二个参数为移动的相对距离，第三个参数为可选参数、省略时系统默认全局状态值

输入参数说明

A	AX,AY,AZ,AC 各笛卡尔坐标轴号
B	各轴移动的相对距离
CP	可选参数，说明运动到目标点时是否平滑过渡，范围为 0~100
Acc	可选参数，指定运动到目标位置的直线加速度，单位为 mm/s^2
Dec	可选参数，指定运动到目标位置的直线减速度，单位为 mm/s^2
Spd	可选参数，指定运动到目标位置的直线速度，单位 mm/s
AccC	可选参数，指定运动到目标位置的姿态加速度，单位**度/s^2**
SpdC	可选参数，指定运动到目标位置的姿态速度，单位**度/s**
I	可选参数，第三轴电流设定值，单位 mA。若运动过程中的电流超过了设定值则当前运动停止，继续往下执行。
In	可选参数，输入检测信号。若运动过程中检测到该输入信号则当前的运动停止，继续往下执行。
ON/OFF	ON：打开；OFF：关闭

返回值说明

STA	0：指令运动正常
	1：输入信号中断运行
	-1：电流到达阈值中断运行

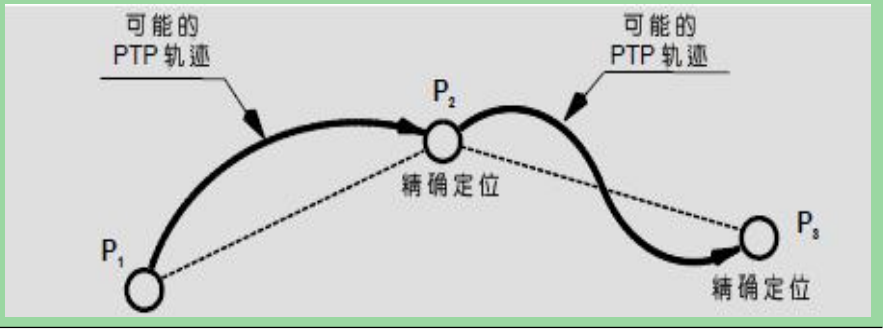
举例说明	1. MovLR(AX,10) --从当前位置往 X 轴正方向移动 10mm 的距离 2. MovLR(AC,10) --从当前位置往 C 轴正方向移动 10 度的角度 3. MovLR(AY,10) --从当前位置往 Y 轴正方向移动 10mm 的距离 4. MovLR(AZ,-10) --从当前位置往 Z 轴负方向移动 10mm 的距离

MovP

使用说明	点到点方式移动到笛卡尔坐标系下的目标绝对位置
语法说明	STA = MovP(A,"CP=20 Acc=20 Dec=20 Spd=100 I=0 In =10 ON/OFF")
参数说明	该指令一共两个参数，第一个参数 A 为目标点，第二个参数为可选参数，省略时系统默认全局状态值

输入参数说明

A	笛卡尔坐标目标位置。可以是点的名称 p1~p2999，也可以是点的索引号 1~2999
CP	可选参数，说明运动到目标点时是否平滑过渡，范围 0~100

	Acc	可选参数，指定运动到目标位置的加速度比例，范围 1~100
	Dec	可选参数，指定运动到目标位置的减速度比例，范围 1~100
	Spd	可选参数，指定运动到目标位置的速度比例，范围 1~100
	I	可选参数，第三轴电流设定值，单位 mA。若运动过程中的电流超过了设定值，则会报警（电流超出报警）终止整个程序运行
	In	可选参数，输入检测信号。若运动过程中检测到该输入信号则当前的运动停止，继续往下执行。
	ON/OFF	ON：打开；OFF：关闭
	返回值说明	
	STA	0: 指令运动正常
		1：输入信号中断运行
		-1：电流到达阈值中断运行
示意图		
举例说明	1. MovP(p1) --机器人以点到点的方式从当前位置运动到 p1 目标点 MovP(10) --机器人以点到点方式从当前位置运动到 p10 目标点 2. MovP(10, "Acc=50 Spd=50") --机器人以点到点方式从当前位置运动到 p10 目标点，其中加速度为 50%的倍率，速度为 50%的倍率 3. MovP(p20, "CP=20") --机器人以点到点方式运动到 p20 位置，其中目标位置 p20 以平滑度 20 过渡 4. local p={x=200,y=10,z=-10,c=30} --用户定义 p 点变量 MovP(p) --运动到 p 点目标位置 5. MovP(p10+Z(-10)) --机器人以点到点的方式从当前位置运动到 p10 点下方 10mm 的位置 6. MovP(p11+Z(10)) --机器人以点到点的方式从当前位置运动到 p10 点上方 10mm 的位置 7. local pos = {x=300,y=100,z=-50,c=60} --用户定义 pos 点变量 MovP(pos+X(-10)) --机器人以点到点的方式从当前位置运动到相对于 pos 点 X 负方向 10mm 的位置 8. MovP(12+Y(10)) --机器人以点到点的方式从当前位置运动到相对于 p12 点 Y 正方向 10mm 的位置	

注：

(1) 当使用 MovP 走点位偏移（例如，例 5、6、7、8）时，一定要用正号（+）连接轴实现偏移，偏移的正负取决于轴括号内的数字的正负。

(2) 大写 X 、Y、Z、C 分别表示 X 轴、Y 轴、Z 轴、C 轴方向的轴偏移

MovPR

使用说明	点到点方式移动到笛卡尔坐标系下的目标相对位置	
语法说明	STA = MovPR(A,B," CP=20 Acc=20 Dec=20 Spd=100 I=0 In =10 ON/OFF ")	
参数说明	**输入参数说明**	
	A	AX,AY,AZ,AC 各笛卡尔坐标轴号
	B	各轴移动的相对距离
	CP	可选参数，说明运动到目标点时是否平滑过渡，范围 0~100
	Acc	可选参数，指定运动到目标位置的加速度比例，范围 1~100
	Dec	可选参数，指定运动到目标位置的减速度比例，范围 1~100
	Spd	可选参数，指定运动到目标位置的速度比例，范围 1~100
	I	可选参数，第三轴电流设定值，单位 mA。若运动过程中的电流超过了设定值则当前运动停止，继续往下执行。
	In	可选参数，输入检测信号。若运动过程中检测到该输入信号则当前的运动停止，继续往下执行。
	ON/OFF	ON：打开；OFF：关闭
	返回值说明	
	STA	0: 指令运动正常
		1：输入信号中断运行
		-1：电流到达阈值中断运行
举例说明	1. MovPR(AX,10) --从当前位置以点到点的方式往 X 轴正方向移动 10mm 的距离 2. MovPR(AC,10) --从当前位置以点到点的方式往 C 轴正方向移动 10 度的角度	

MovJ

使用说明	点到点的方式移动机器人各个关节到指定的角度绝对位置	
语法说明	1. MovJ (A,B," Acc=20 Dec=20 Spd=20") 2. MovJ(A,"Acc=20 Dec=20 Spd=20")	
参数说明	**用法 1：下表各指令参数说明**	
	A	J1,J2,J3,J4 对应机器人各个关节号
	B	各关节移动的目标角度
	Acc	可选参数，指定运动到目标位置的加速度比例，范围 1~100
	Dec	可选参数，指定运动到目标位置的减速度比例，范围 1~100
	Spd	可选参数，指定运动到目标位置的速度比例，范围 1~100
	用法 2：下表各指令参数说明	
	A	每个轴的关节目标位置（数组变量）
	Acc	可选参数，指定运动到目标位置的加速度比例，范围 1~100
	Dec	可选参数，指定运动到目标位置的减速度比例，范围 1~100
	Spd	可选参数，指定运动到目标位置的速度比例，范围 1~100

举例说明	1. MovJ (J1,10) --机器人第一关节移动到 10 度的位置 2. MovJ (J3,-10) --机器人第三关节移动到-10 毫米的位置 3. JointAngle = {x=30,y=40,z=10,c=30} MovJ(JointAngle, “Acc=20 Dec=20 Spd=20”) –关节运动到目标位置点

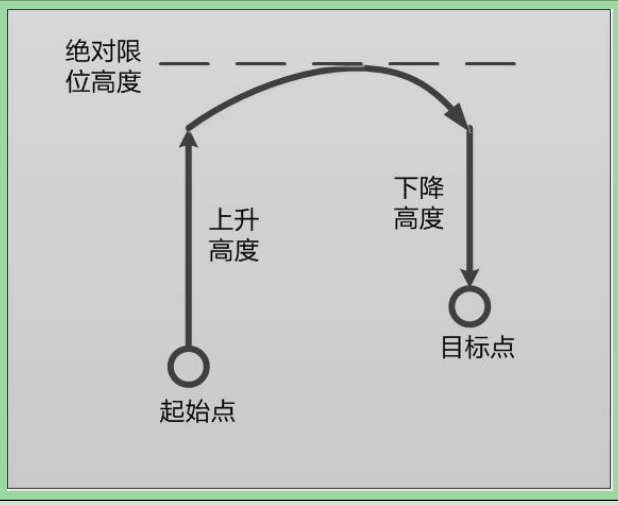
◆ 注意：关节运动时 J3 为毫米单位，其它关节单位为角度。

MArchP

使用说明	机器人以点到点的方式做拱形运动
语法说明	1. STA = MArchP(A,B,C,D," Acc=20 Dec=20 Spd=100 I=0 In=10 ON/OFF”) 2. STA = MArchP(A,B, "Acc=20 Dec=20 Spd=100 I=0 In=10 ON/OFF")
参数说明	**用法 1：输入参数说明**

A	笛卡尔坐标目标位置。可以是点的名称 p1~p2999，也可以是点的索引号 1~2999
B	Z 轴最高限位绝对位置，单位为毫米
C	Z 轴上升的高度，单位为毫米
D	Z 轴下降的高度，单位为毫米
Acc	可选参数，指定拱形运动的加速度比例，范围 1~100
Dec	可选参数，指定拱形运动的减速度比例，范围 1~100
Spd	可选参数，指定拱形运动的速度比例，范围 1~100
I	可选参数，第三轴电流设定值，单位 mA。若运动过程中的电流超过了设定值则当前运动停止，继续往下执行。
In	可选参数，输入检测信号。若运动过程中检测到该输入信号则当前的运动停止，继续往下执行。
ON/OFF	ON：打开；OFF：关闭

示意图



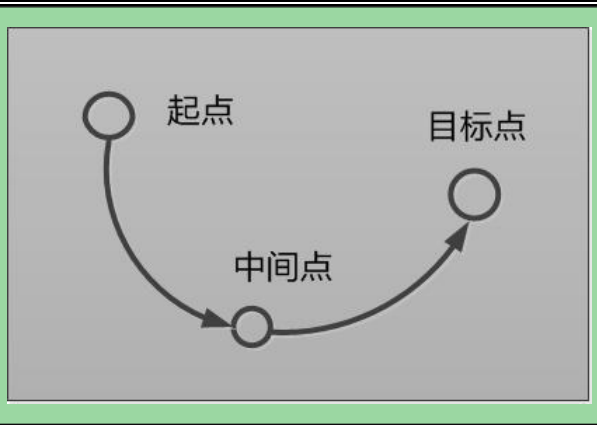
用法 2：输入参数说明

A	笛卡尔坐标目标位置。可以是点的名称 p1~p2999，也可以是点的索引号 1~2999
B	Z 轴最高限位（绝对位置），也是 Z 轴要实际走到的高度，单位为毫米
Acc	可选参数，指定拱形运动的加速度比例，范围 1~100
Dec	可选参数，指定拱形运动的减速度比例，范围 1~100

	Spd	可选参数，指定拱形运动的速度比例，范围 1~100
	I	可选参数，第三轴电流设定值，单位 mA。若运动过程中的电流超过了设定值则当前运动停止，继续往下执行
	In	可选参数，输入检测信号。若运动过程中检测到该输入信号则当前的运动停止，继续往下执行
	ON/OFF	ON：打开；OFF：关闭
	返回值说明	
	STA	0: 指令运动正常
		1：输入信号中断运行
		-1：电流到达阈值中断运行
举例说明	1. MArchP(p1,0,10,5) --机器人从当前位置 Z 轴上升 10mm 距离，然后以点到点的运动方式移动到距离目标位置 5mm 处，Z 轴下降 5mm 到达目标位置（拱形的高度不能超过 Z 轴的最高限 0mm （绝对位置））	
	2. MArchP(p1,0,10,5,"Acc=50 Spd=100") --机器人以 50%的加速度，100%的速度做点到点方式拱形运动	
	3. MArchP(p2,-10) --机器人从当前位置 Z 轴上升到-10mm 位置，然后以点到点的运动方式移动到（p2.x,p2.y,-10,p2.c）位置，最后 Z 轴下降(-10-p2.z)到达目标位置	

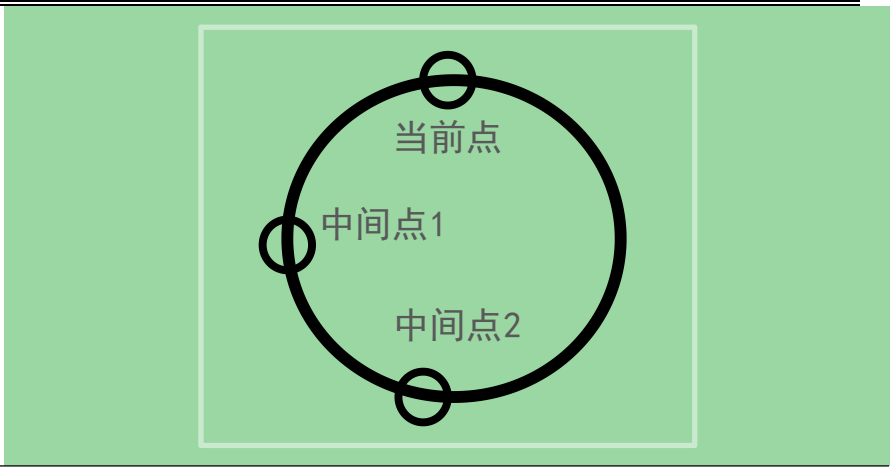
MArc

使用说明	笛卡尔坐标系下的圆弧运动	
语法说明	MArc(A,B,"CP=20 Acc=20 Dec=20 Spd=100 Angle = 360")	
参数说明	**下表各指令参数说明**	
	A	笛卡尔坐标圆弧经过点。可以是点的名称 p1~p2999，也可以是点的索引 1~2999
	B	笛卡尔坐标圆弧终点。可以是点的名称 p1~p2999，也可以是点的索引 1~2999
	CP	可选参数，说明运动到目标点时是否平滑过渡，范围 0~100
	Acc	可选参数，指定运动到目标位置的加速度，单位为 mm/s^2
	Dec	可选参数，指定运动到目标位置的减速度，单位为 mm/s^2
	Spd	可选参数，指定运动到目标位置的速度，单位为 mm/s
	Angle	可选参数，指定圆弧的角度，范围 1~360

示意图	
举例说明	1. MArc(p1,p2) --机器人从当前位置以圆弧的方式经过 p1 点运动到 p2 目标位置 2. MArc(1,2) --机器人从当前位置以圆弧的方式经过 p1 点运动到 p2 目标位置 3. MArc(1,2,"Acc=100 Spd=1000") --机器人从当前位置以圆弧的方式经过 p1 点运动到 p2 目标点，其中加速度为 100mm/s2，速度为 1000mm/s 4. MArc(1,2,"CP=20") --机器人圆弧运动到 p2 点时平滑过渡

MCircle

使用说明	机器人在笛卡尔坐标系下的圆周运动	
语法说明	MCircle(A,B," CP=20 Acc=20 Dec=20 Spd=100 Angle=360 ")	
参数说明	**下表各指令参数说明**	
	A	笛卡尔坐标圆周经过点 1。可以是点的名称 p1~p2999，也可以是点的索引 1~2999
	B	笛卡尔坐标圆周经过点 2。可以是点的名称 p1~p2999，也可以是点的索引 1~2999
	CP	可选参数，说明运动到目标点时是否平滑过渡，范围 0~100
	Acc	可选参数，指定运动到目标位置的加速度，单位为 mm/s^2
	Dec	可选参数，指定运动到目标位置的减速度，单位为 mm/s^2
	Spd	可选参数，指定运动到目标位置的速度，单位为 mm/s
	Angle	指定圆周的角度，范围 1~360

示意图	
举例说明	1. MCircle(p1,p2) --机器人从当前位置经过 p1 点运动到 p2 目标位置的整圆运动 2. MCircle(1,2) --机器人从当前位置经过 p1 点运动到 p2 目标位置的整圆运动。 3. MCircle(1,2,"Acc=100 Spd=1000") --机器人从当前位置经过 p1 点运动到 p2 目标点的圆周运动，其中加速度为 100mm/s2，速度为 1000mm/s。 4. MCircle(1,2,"CP=20") --机器人圆周运动到 p2 点时平滑过渡 p2 点。 5. MCircle (1,2,"Angle=180") --机器人从当前位置往 p1 到 p2 的方向运动 180 度的半圆

MSpline

使用说明	样条曲线插补运动指令
语法说明	MSpline(A,B, “CP=20 Acc=20 Dec=20 Spd=100 AccC=20 SpdC=20”)
参数说明	**下表各指令参数说明**

A	样条曲线起始点（点位表中示教）
B	样条曲线结束点（点位表中示教）
Acc	可选参数，指定运动到目标位置的加速度，单位 mm/s^2
Dec	可选参数，指定运动到目标位置的减速度，单位 mm/s^2
Spd	可选参数，指定运动到目标位置的速度，单位 mm/s
AccC	可选参数，指定运动到目标位置的姿态加速度，单位**度/s^2**
SpdC	可选参数，指定运动到目标位置的姿态速度，单位**度/s**

举例说明	1. MSpline(p1,p10) --p1~p10 点共 10 个点形成的样条曲线轨迹 2. MSpline(p10,p150) --p10~p150 共 141 个点形成的样条曲线轨迹

注：

- 起始点、结束点、以及之间的点位都需要在点位表（DATA.PTS）中示教（必须连续，中间的点位数据不能为空，否则出错）；
- 从起始点到结束点，点位坐标（数据）个数不能少于 3 个，也不能超过 300 个；
- 样条曲线插补运动指令，须结合示教器参数里的**样条曲线参数（默认拆分步长）；**
- 该指令会自动判断起始点与结束点坐标是否为同一个点，若为同一个点，则曲线封闭，否则曲线不封闭。

handmove

使用说明	自动运行状态下实现手动笛卡尔/关节的连续、点动运动
语法说明	handmove(0xAB,C,D,E)
参数说明	**下表各指令参数说明**

A	A=0：非阻塞运动； A=1：阻塞运动	
B	B=0：笛卡尔连续运动； B=1：关节连续运动； B=2：笛卡尔点动运动； B=3：关节点动运动	
C	**笛卡尔运动** C=AX：X 轴； C=AY：Y 轴； C=AZ：Z 轴； C=AC：C 轴	**关节运动** C=J1：J1 轴； C=J2：J2 轴； C=J3：J3 轴； C=J4：J4 轴
D	**连续运动** D=1：正方向； D=-1：负方向	**点动运动** 点动移动的距离（距离的正负代表运动方向）
E	指定机器人运动速度倍率，范围 1~100（-1 指的是默认当前的机器人速度倍率）	

举例说明

1. handmove(0x00,AX,1,20) --机器人以 20%的速度倍率往 X 轴正方向连续运动，若接收到停止信号立即响应
2. handmove(0x01,J2,-1,30) --机器人以 30%的速度倍率往 J2 轴负方向连续运动，若接收到停止信号立即响应
3. handmove(0x12,AZ,-10,-1) --机器人以当前默认的速度倍率往 Z 轴负方向运动 10mm 的距离，直到运动到指定位置才能响应停止信号
4. handmove(0x13,J4,30,-1) --机器人以当前默认的速度倍率往 J4 轴正方向运动 30 度，直到运动到指定位置才能响应停止信号

stoprun

使用说明	自动运行状态下实现停止手动直线运动
语法说明	stoprun()
参数说明	**无参数**

举例说明	function Bit(value,bit)
	if (value & (0x0001<<bit)) == 0 then
	return nil
	else
	return 1
	end
	end

```
function ExtManu()
    local value = publicread(0x100)
    if Bit(value,0) then
        print("X+\n")
        --handmove(0x00,1,1,-1) --X+方向连续运动
        --handmove(0x01,1,1,-1) --J1+方向连续运动
        handmove(0x12,1,5,-1)    --X+方向寸动
        --handmove(0x13,1,5,-1) --J1+方向寸动
    elseif Bit(value,1) then
        print("X-\n")
        --handmove(0x00,1,-1,-1)--X-方向连续运动
        --handmove(0x01,1,-1,-1)--J1-方向连续运动
        handmove(0x12,1,-5,-1) --X-方向寸动
        --handmove(0x13,1,-5,-1)--J1-方向寸动
    elseif Bit(value,2) then
        print("Y+\n")
        --handmove(0x00,2,1,-1)--Y+方向连续运动
        --handmove(0x01,2,1,-1)--J2+方向连续运动
        handmove(0x12,2,5,-1) --Y+方向寸动
        --handmove(0x13,2,-5,-1)--J2+方向寸动
    elseif Bit(value,3) then
        print("Y-\n")
        --handmove(0x00,2,-1,-1)--Y-方向连续运动
        --handmove(0x01,2,-1,-1)--J2-方向连续运动
        handmove(0x12,2,-5,-1) --Y-方向寸动
        --handmove(0x13,2,-5,-1)--J2-方向寸动
    elseif Bit(value,4) then
        print("Z+\n")
        --handmove(0x00,3,1,-1)--Z+方向连续运动
        --handmove(0x01,3,1,-1)--J3+方向连续运动
        handmove(0x12,3,5,-1) --Z+方向寸动
        --handmove(0x13,3,-5,-1)--J3+方向寸动
    elseif Bit(value,5) then
        print("Z-\n")
        --handmove(0x00,3,-1,-1)--Z-方向连续运动
        --handmove(0x01,3,-1,-1)--J3-方向连续运动
        handmove(0x12,3,-5,-1) --Z-方向寸动
        --handmove(0x13,3,-5,-1)--J3-方向寸动
    elseif Bit(value,6) then
        print("C+\n")
        --handmove(0x00,4,1,-1)--C+方向连续运动
        --handmove(0x01,4,1,-1)--J4+方向连续运动
        handmove(0x12,4,5,-1) --C+方向寸动
```

```
            --handmove(0x13,4,-5,-1)--J4+方向寸动
        elseif Bit(value,7) then
            print("C-\n")
            --handmove(0x00,4,-1,-1)--C-方向连续运动
            --handmove(0x01,4,-1,-1)--J4-方向连续运动
            handmove(0x12,4,-5,-1) --C-方向寸动
            --handmove(0x13,4,-5,-1)--J4-方向寸动
        else
            stoprun()
        end
    end
    while 1 do
        if publicread(0x102)==1 then --手动运动
            print("手动模式停止 CPU1")
            syswork(3)                          --停止 CPU1 的运动
            ExtManu()
        end
        if publicread(0x104) ==1 then     --自动运行
            stoprun()
            print("自动模式启动 CPU1")
            syswork(1)
        end
        Delay(10)
    end
```

注： stoprun 指令实现的是停止自动运行状态下的手动运动，故 handmove 指令和 stoprun 指令是配套使用的。

waitpos	
使用说明	等待运动脉冲发送完成指令（运动 DSP 发送脉冲完成指令）
语法说明	waitpos()
参数说明	无
举例说明	1. MovP(p1) MovP(p2) waitpos() --等待运动 DSP 将运动到 P2 点的脉冲全部发送给伺服 DSP a =1 --若无 waitpos()指令， a 的值会提前打印 print(“a”)

waitrealpos	
使用说明	机器人运动到位（停稳）指令（伺服响应脉冲到位指令）
语法说明	waitrealpos()
参数说明	无

举例说明	2. MovP(p1) waitrealpos() --等待机器人到达 p1 点（伺服响应有一定的滞后性） print(“已到达 p1 点”)

1.12 运动参数设置指令

指令符号	指令说明
AccJ	设置加速度比例指令，影响 MovJ、MovJR、MovP、MovPR、MArchP 指令的加速时间
DecJ	设置减速度比例指令，影响 MovJ、MovJR、MovP、MovPR、MArchP 指令的减速时间
SpdJ	设置速度比例指令，影响 MovJ、MovJR、MovP、MovPR、MArchP 指令的运行速度
AccL	设置直线运动指令加速度，影响 MovL、MovLR、MArc、MCircle 指令的加速时间。单位 mm/s^2
DecL	设置直线运动指令减速度，影响 MovL、MovLR、MArc、MCircle 指令的减速时间。单位 mm/s^2
SpdL	设置直线运动指令速度，影响 MovL、MovLR、MArc、MCircle 指令的运行速度。单位 mm/s

AccJ

使用说明	设置点到点运动方式的加速度比例	
语法说明	AccJ(A)	
参数说明	A	百分比，范围 1~100
举例说明	1. AccJ(50) --设置点到点 50%的加速度比例 2. MovP(p2) --机器人以 50%的加速度比例运动到 p2 目标位置	

◆ 注意：设置后机器人一直保持该全局加速度比例为点到点运动的默认加速度比例值，直到下一次更新。

DecJ

使用说明	设置点到点运动方式的减速度比例	
语法说明	DecJ(A)	
参数说明	A	百分比，范围 1~100
举例说明	3. DecJ(50) --设置点到点 50%的减速度比例 4. MovP(p2) --机器人以 50%的减速度运动到 p2 目标位置	

SpdJ

使用说明	设置点到点运动方式的速度比例

语法说明	SpdJ(A)	
参数说明	A	百分比，范围 1~100
举例说明	1. SpdJ(50) --设置点到点 50%的速度比例 2. MovP(p2) --机器人以 50%的速度比例运动到 p2 目标位置	

AccL

使用说明	设置直线运动方式的加速度	
语法说明	AccL(A)	
参数说明	A	实际加速度，单位 mm/s^2，范围 1~10000
举例说明	1. AccL(500) --设置直线运动的加速度为 500 mm/s^2 2. MovL(p2) --机器人以 500 mm/s^2的加速度运动到 p2 目标位置	

DecL

使用说明	设置直线运动方式的减速度	
语法说明	DecL(A)	
参数说明	A	实际减速度，单位 mm/s^2，范围 1~10000
举例说明	DecL(500) --设置直线运动的减速度为 500 mm/s^2 MovL(p2) --机器人以 500 mm/s^2的减速度运动到 p2 目标位置	

SpdL

使用说明	设置直线运动方式的速度	
语法说明	SpdL(A)	
参数说明	A	实际速度，单位 mm/s
举例说明	1. SpdL(500) --设置直线运动的速度为 500mm/s 2. MovL(p2) --机器人以 500mm/s 的速度运动到 p2 目标位置	

1.13 程序管理指令

指令符号	指令说明
Delay	延时指令。单位：毫秒
Exit	程序运行终止
Pause	暂停程序运行

Delay

使用说明	延时指令	
语法说明	Delay(A)	
参数说明	A	延时时间，单位为毫秒,范围 1~100000
举例说明	1. Delay(1000) --程序延时 1000 毫秒	

	2. Delay(1)	--机器人延时 1 毫秒
	3. local time = 1000 Delay(time)	--参数为变量

Exit	
使用说明	退出程序执行指令
语法说明	Exit()
参数说明	该指令无参数
举例说明	Exit() --执行该指令后程序程序停止执行 MovL(1) --该指令不会被执行

Pause	
使用说明	暂停程序执行指令
语法说明	Pause()
参数说明	该指令无参数
举例说明	Pause() --执行该指令后程序暂停执行,按**启动键**后程序继续运行 MovL(p1)

◆ 注意：**暂停后，按启动键后程序会继续从暂停行继续运行。**

1.14 一般指令

指令符号	指令说明
X	创建指定 X 轴笛卡尔坐标绝对位置点的指令
Y	创建指定 Y 轴笛卡尔坐标绝对位置点的指令
Z	创建指定 Z 轴笛卡尔坐标绝对位置点的指令
C	创建指定 C 轴笛卡尔坐标绝对位置点的指令
XYZC	创建指定 XYZC 轴笛卡尔坐标绝对位置点的指令

X/Y/Z/C		
使用说明	创建指定某个轴笛卡尔坐标位置的点	
语法说明	X(A) Y(A) Z(A) C(A)	
参数说明	A	各个轴笛卡尔坐标位置，C 轴为角度，其它轴为 mm
举例说明	1. MovL(p10 + X(20)) --机器人运动到相对于 p10 点往 X 轴正向偏移 20 毫米的位置 2. MovL(p11 + Y(-20)) --机器人运动到相对于 p11 点往 Y 轴负向偏移 20 毫米的位置 3. MovL(p12 + Z(-10)) --机器人运动到相对于 p12 点往 Z 轴负向偏移 10 毫米的位置 4. MovL(p13 +C(30)) --机器人运动到相对于 p13 点往 C 轴正向偏移 30 度的位置	

◆ 注意：该指令是用来生成一个点数据，不会产生运动，一般与点数据运算后作为参数传递给运动指令。

XYZC	
使用说明	创建指定各个轴笛卡尔坐标位置的点
语法说明	XYZC(A,B,C,D)
参数说明	**下表各指令参数说明**
	A 指定 X 轴笛卡尔坐标位置，单位 mm
	B 指定 Y 轴笛卡尔坐标位置，单位 mm
	C 指定 Z 轴笛卡尔坐标位置，单位 mm
	D 指定 C 轴笛卡尔坐标位置，单位 mm
举例说明	MovL(p10 + XYZC(10,20,-5,30)) --机器人运动到相对于 p10 点往 X 轴正向偏移 10 毫米,Y 轴正向偏移 20 毫米、Z 轴负向偏移 5 毫米、C 轴正向偏移 30 度的位置移动

1.15 输入/输出指令

指令符号	指令说明
DI	读取输入端口状态
DO	输出端口打开或关闭操作
WDI	读取输入端口状态,直到等待某一信号有效，则继续执行后续的程序
WDO	读取输出端口状态,直到等待某一信号有效，则继续执行后续的程序

DI	
使用说明	读取输入端口状态
语法说明	**形式 1：** DI(-1)或 DI(-2) **形式 2：** DI(A)
参数说明	**形式 1：** DI(-1)的返回值是一个 32 位的二进制数转化的十进制数值，从低位到高位分别代表输入端口（0~31）的状态值；DI(-2)的返回值也是一个 32 位的二进制数转化的十进制数值，低两位分别代表输入端口（32~33）的状态值。其中 0 代表关闭，1 代表打开。 **形式 2：** 返回值 ON(OFF) 或十进制的值 **下表各指令参数说明** A 读取的输入端口号，范围 0~33
举例说明	1. local input = DI(-1) --获取输入端口（0~31）的状态 if ((input>>0)&0x0001)==1 then --判断输入端口 0 是否打开，若为--1，则输入端口 0 打开 MovP(p1) elseif ((input>>9)&0x0001)==1 then--判断输入端口 9 是否打开，若

--为 1，则输入端口 1 打开

```
    MovP(p2)
end
```

2. if DI(10) == ON then --输入端口 10 有效时运动到 p1 点

```
    MovP(p1)
end
```

3. value = DI({1,2,3}) --读取输入端口 1,2,3 的状态，若返回值 value 为 7（111）则表示输入端口 1,2,3 打开；若 value 为 6（110），则 1,2 打开，3 关闭；若 value 为 5（101），则 1,3 打开，2 关闭….依次类推…

- 注意：DI 指令只是读取输入端口的状态，状态有效或无效都不会死等待。
- 注意：若同时读取某几个输入端口的状态，输入端口一定要按照从小到大或从大到小的顺序排序。

DO

使用说明	输出端口打开或关闭操作
语法说明	用法 1：DO(A,B) 用法 2：DO(A,B,"**Time** = C") 用法 3：DO(A,B, “**F**”) 用法 4：DO(A,B, “**Time** = C **H**”) 用法 5：DO(A,B , “**Time** = C **F**”) 用法 6：DO(A,B, “**POS**=D”)
参数说明	返回值 ON 或 OFF **下表各指令参数说明**

A	读写的输出端口号，范围 0~26，可以是单个端口，也可以是多个端口
B	输出端口的状态值
C	可选参数，时间（ms）
D	位置百分比

举例说明

1. DO(10,ON) --打开 10 号输出端口为 ON
2. DO(10,ON,“Time=1000”) --打开 10 号输出端口为 ON,并保持 1 秒钟，1 秒之后输出 10 切换为 OFF
3. DO({1,2,3,4},{ON,ON,ON,ON}) --打开输出 1,2,3,4 为 ON
4. DO({5,6,7,8},{ON,OFF,ON,OFF}) --打开输出 5,7 为 ON，关闭输出 6,8 为 OFF
5. DO({9,10},{ON,ON},“Time=2000”) --打开输出 9,10,并保持 2 秒钟，2 秒之后输出 9,10 切换为 OFF
6. DO({9,10},{0,0},“Time=2000”) --打开输出 9,10 为 OFF,并保持 2 秒钟，2 秒之后输出 9,10 切换为 ON
7. DO({1,2,3},7) --1 对应的二进制为 111，则表示打开输出端口 1，2,3 都为 ON
8. DO({1,2,3},5) --5 对应的二进制为 101，则表示打开输出端口 1,3

为 ON，关闭输出端口 2 为 OFF

9. MovP(p9)
   DO(1,ON, “F”) --从当前位置运动到 p9 点，到达 p9 点后打开输出端口 1，然后运动到 p10 点，运动连续
   MovP(p10)
10. MovP(p1)
    MovP(p2)
    DO(1,ON, “POS=50”) --从 p1 点运动到 p2 的点的 50%距离时打开输出端口 1
11. MovP(p1)
    MovP(p2)
    DO(1,ON, “Time=100 H”) --从 p1 点运动到 p2 的点，在到达 p2 点前的 100ms 打开输出端口 1
12. MovP(p1)
    MovP(p2)
    DO(1,ON, “Time=50 F”) --从 p1 点运动到 p2 的点，在到达 p2 点后的 50ms 打开输出端口 1

- 可选参数 Time 为输出端口保持时间，超过保持时间后会将该当前的输出端口状态切换为相反状态；
- 若同时打开或关闭某几个输出端口，输出端口一定要按照从小到大或从大到小的顺序排序；
- 缓存 IO 功能不适用于组合 IO 的情况；
- 缓存 IO 功能适用于点到点运动（MovP）、直线运动（MovL）以及拱形运动（MArchP）；
- **Time、POS、F、H** 为 DO 指令中的关键字，不能修改；
- 运动过程中启用缓存 IO 功能，当前的运动不会终止，运动连续。

WDI

使用说明	读取输入端口状态,直到等待某一或多个信号有效，则继续执行后续的程序
语法说明	WDI(A,B) WDI(A,B, “Time=1000”)
参数说明	返回值 1 (ON)或 0 (OFF) **下表各指令参数说明**

A	读取的输入端口号
B	输入端口的状态值，ON 或 OFF
Time	可选参数，等待时间，单位为毫秒

举例说明

1. local flag = WDI(10,ON) --直到等到输入端口 10 状态为 ON 再执行 MovP 指令
   print(flag) --返回值为 1
   MovP(p1)
2. local flag = WDI(10,OFF) --直到等到输入端口 10 状态为 OFF 再执行 MovP 指令

```
print(flag)     --返回值为 0
MovP(p1)
```

3. local flag = WDI(10,ON, “Time=2000”) --等待输入端口 10 状态为 ON，等待时间 2 秒，若 2 秒后输入端口 10 状态仍为 OFF，则继续执行 MovP 指令

```
if flag == 1 then          --在 2 秒内输入端口 10 状态检测为 ON
    MovP(p1)
elseif flag == 0 then    --在 2 秒内输入端口 10 状态一直检测为 OFF
    MovP(p2)
end
```

4. local flag = WDI({1,2,3},{ON,ON,ON}) --直到同时等到输入端口 1,2,3 同时为 ON 再执行 MovP 指令

```
print(flag)     --返回结果为 7（对应二进制的 111）
MovP(p1)
```

5. local flag = WDI({1,2,3},{ON,ON,ON}, “Time=2000”) --等待输入端口 1,2,3 状态为 ON，等待时间为 2 秒，若 2 秒之后输入端口仍 OFF ，仍继续执行 MovP 指令

```
if flag ==7 then       --输入端口 1、2、3 在 2 秒内状态检测到为 ON
    MovP(p1)
elseif flag ==0 then
    MovP(p2)        --输入端口 1、2、3 在 2 秒内状态一直检测为 OFF
end
```

◆ 注意：若同时等待某几个输入端口的状态，输入端口一定要按照从小到大或从大到小的顺序排列。

WDO	
使用说明	读取输出端口状态,等待某一信号有效，则继续执行后续的程序
语法说明	WDO(A,B) WDO(A,B, “Time=1000”)
参数说明	返回值　ON 或 OFF **下表各指令参数说明** A：读取的输出端口号 B：输出端口的状态值，ON 或 OFF Time：可选参数，等待时间，单位为毫秒
举例说明	1. WDO(10,ON) --直到等到输入端口 10 状态为 ON 再执行 MovP 指令 MovP(p2) 2. WDO(10,ON, “Time=2000”) --等待输出端口 10 状态为 ON，等待时间为 2 秒，若 2 秒之后输出端口 10 状态仍为 OFF，则继续执行 MovP MovP(p2)

1.16 坐标系指令

指令符号	指令说明

SetU	设置机器人当前用户坐标系
WrU	修改用户坐标系的数据
SetT	设置机器人当前工具坐标系
WrT	修改工具坐标系的数据
CacU	两点法新建用户坐标系
U2U	用户（0~9）之间坐标相互转换
V2Tool	根据视觉坐标计算工具坐标
CacT	两点法新建工具坐标系
getcart	获取当前笛卡尔坐标
getjoint	计算目标点对应的某一个轴的关节坐标
encoderget	获取编码器的脉冲值
targetok	判断目标位置是否为机器人可到达点位

SetU

使用说明	设置当前用户坐标系	
语法说明	**形式 1：** SetU(A) **形式 2：** SetU(A,B)	
参数说明	A	设置的用户坐标系号，范围 0~9
	B	用户坐标偏移量
举例说明	1. SetU(1) --选择用户坐标系 1 MovL(p1) --运动到用户坐标系 1 中的 p1 位置 2. SetU(1,{ x=10,y=20,z=0,c=30}) --在用户坐标系 1 的基础上增加偏移量，然后设定为新的用户坐标系 MovL(p1) --在新的用户坐标系中运动到 p1 位置	

- 注意：坐标系设置后立即有效并保持到下一次更新。
- 注意：在原有的用户坐标系的基础上设置偏移量生成新的用户坐标，原有的用户坐标没有被修改。

SetT

使用说明	设置当前工具坐标系	
语法说明	SetT(A)	
参数说明	A	设置的工具坐标系号，范围 1~9
举例说明	1. SetT (1) --选择工具坐标系 1 2. MovL(p1) --运动到工具坐标系 1 中的 p1 位置	

WrU

使用说明	修改用户坐标系的数据
语法说明	**形式 1：** WrU(A,B,C) **形式 2：** WrU(A,B)

参数说明	**下表各指令参数说明(形式 1)**	
	A	要修改的用户坐标系号，范围为 1~9
	B	修改的轴号，分别为 AX、AY、AZ、AC
	C	修改的值，X/Y/Z 轴单位为毫米,C 轴单位为角度
	下表各指令参数说明(形式 2)	
	A	要修改的第几组用户坐标系号，范围为 1~9
	B	Table 类型，包含用户坐标系的原点坐标，以及该用户相对于用户 0 旋转的角度
举例说明	1. WrU(1,AX,200) --修改第一组用户坐标系的 X 轴为 200 毫米 2. WrU(1,AC,100) --修改第一组用户坐标系的 C 轴为 100 度 3. WrU(2,{x=200,y=100,z=0,c=90}) --修改用户 2 的原点为(200,100,0),用户 2 相对于用户 0 的旋转角度为 90 度 4. Pos = { x=300,y=100,z=0,c=-90} WrU(3,Pos)	

WrT

使用说明	修改工具坐标系的数据
语法说明	**形式 1：** WrT (A,B,C) **形式 2：** WrT(A,B)
参数说明	**下表各指令参数说明(形式 1)**
	A 要修改的工具坐标系号，范围为 1~9
	B 修改的轴号，分别为 AX、AY、AZ、AC
	C 修改的值，单位为毫米,C 轴为角度
	下表各指令参数说明(形式 2)
	A 要修改的工具坐标系号，范围为 1~9
	B Table 类型，包含要修改的工具末端相对于机器人末端在 X,Y,Z,C 轴方向的偏移量
举例说明	1. WrT(1,AX,20) --修改工具坐标系 1 的 X 轴偏移量为 20 毫米 2. WrT(1,AY,10) --修改工具坐标系 1 的 Y 轴偏移量为 10 毫米 3. WrT(1,{x= 10,y=20,z=0,c=30}) –修改工具 1 相对于机器人末端在 X、Y、Z、C 方向上的偏移量分别为 10mm,20mm,0mm,30 度。 4. Offset = { x=10,y=10,z=0,c=-30} WrT(2,Offset)

CacU

使用说明	新建用户坐标系
语法说明	CacU (pos1,pos2)
参数说明	**下表各指令参数说明**
	pos1 用户坐标系的原点
	Pos2 用户坐标系 X 方向上的一点

举例说明	1. local pos1 = {300,100,0,0} --用户坐标系的原点 2. local pos2 = {300,120,0,0} --用户坐标系 X 轴方向上的一点 3. WrU(1,CacU(pos1,pos2)) --将新建的用户坐标指定为用户坐标系 1

U2U

使用说明	用户（0~9）之间坐标相互转换	
语法说明	Pos = U2U (A,B,C)	
参数说明	**下表各指令参数说明**	
	A	被转换的用户号（0~9）
	B	目标用户号（0~9）
	C	被转换用户号下的坐标
	Pos	目标用户号下的坐标
举例说明	1. local pos = U2U(0,2,p1) --将用户 0 下的 p1 点转换到用户 2 下的坐标 SetU(2) MovP(pos) 2. local pos1= {x= 100,y=40,z=-10,c=30} local pos = U2U(1,3,pos1) –把用户 1 中的 pos1 点的坐标转换到用户 3 下的坐标 SetU(3) MovP(pos)	

V2Tool

使用说明	根据视觉坐标计算工具坐标	
语法说明	V2Tool(A,B)	
参数说明	A	视觉输出的笛卡尔坐标值
	B	工具坐标系号，范围 1~9
举例说明	local vis = {x=300, y=10,z=0,c=0} --视觉坐标 V2Tool(vis,3) --将计算出来的工具写入到工具坐标系 3 SetT(3) --设置当前工具坐标系为 3 号坐标系	

◆ 注意：视觉坐标和当前的机器人坐标必须在同一坐标系下。

CacT

使用说明	两点法新建工具坐标系	
语法说明	CacT(pos1,pos2)	
参数说明	**下表各指令参数说明**	
	pos1	机器人末端笛卡尔坐标
	pos2	夹具末端笛卡尔坐标
举例说明	Pos1= getcart()	

```
pos2 = {x=300,y=100,z=0,C=30}
WrT(1,CacT(pos1,pos2))
```

注：

（1） 夹具末端的笛卡尔坐标通过视觉定位获取；
（2） 夹具末端笛卡尔坐标和机器人末端笛卡尔坐标必须处于同一坐标系；

getcart		
使用说明	获取当前笛卡尔坐标以及关节坐标	
语法说明	pos1,pos2=getcart()	
参数说明	无参数	
返回值	pos1	机器人当前的笛卡尔坐标
	pos2	机器人当前的关节坐标
举例说明	1. local pos1,pos2= getcart()--获取当前笛卡尔坐标以及关节坐标 --pos1.x,pos1.y,pos1.z,pos1.c,pos1.h 分别为笛卡尔 x/y/z/c/h 的值 --pos2.x,pos2.y,pos2.z,pos2.c,pos2.h 为关节 J1/J2/J3/J4 以及手系的值 2. local pos = getcart() --获取当前笛卡尔坐标 -- pos.x,pos.y,pos.z,pos.c,pos.h 分别为笛卡尔 x/y/z/c/h 的值	

getjoint		
使用说明	计算目标点对应的 J1 轴或 J2 轴的关节坐标	
语法说明	Err, Joint = getjoint(pos,A)	
参数说明	pos	目标点坐标
	A	轴关节号 J1/J2
返回值	Err	错误号，用来判断目标点是否是机器人可到达点； Err = 0：目标点能到达 Err = 6: 目标点不能到达
	Joint	J1 或 J2 轴对应的关节值

举例说明	local p1 --基准点
	local pos =CCDrecv(“CAM0”) --目标位置 pos
	pos.h = p1.h
	local Err,Joint1 = getjoint(pos,J1) –计算 pos 点在基准点手系下的 J1 角度
	if Err==0 then --目标点是机器人能到达点
	print(Joint1) --打印目标点 J1 轴的关节坐标
	if joint>190 and joint<283 then
	Delay(1)
	else
	print(“Joint1 的值不在合理范围，请检查位置”)
	Pause()
	end	

```
else
    print("Err 不正确，机器人不能到达这个点",Err)
    Exit()
end
```

- 注意：计算目标点对应的 J1 轴或 J2 轴对应的关节坐标，一定要指定达到目标点的手系，默认是机器人当前点的手系；
- getjoint 函数主要解决用于吊装（倒装）机器人走自定义点位（例如：视觉发送的坐标或码垛点计算的坐标，非示教点位）最优路径的问题。

encoderget

使用说明	获取编码器的脉冲值	
语法说明	Pulse = encoderget(A)	
参数说明	A	编码器号，范围 1~6
返回值	Pulse	返回对应编码器的脉冲值
举例说明	1. Pulse1=encoderget(1) --获取 J1 轴编码器 M1 的脉冲值； 2. Pulse2=encoderget(2) --获取 J2 轴编码器 M2 的脉冲值； 3. Pulse3=encoderget(3) --获取 J3 轴编码器 M3 的脉冲值； 4. Pulse4=encoderget(4) --获取 J4 轴编码器 M4 的脉冲值； 5. Pulse5=encoderget(5) --获取外接编码器 M5 的脉冲值； 6. Pulse6=encoderget(6) --获取外接编码器 M6 的脉冲值；	

targetok

使用说明	判断目标位置是否为机器人可到达点位	
语法说明	sta = targetok(A)	
参数说明	A	目标点位，table 类型数据
返回值	sta	是否为可到达表示位，sta=0 为可到达，sta=1 为不可到达

举例说明

```
1. sta = targetok(p1)     --判断 p1 点是否为机器人可到达点位
   if sta == 0 then
       print("p1 点为机器人可到达点位")
   elseif sta== 1 then
       print("p1 点位为机器人不可到达点位，请检查")
       Pause()     --暂停
   end
2. pos = {x=300,y=10,z=-20,c=30,h=1}
   sta = targetok(pos)     --判断 pos 点是否为机器人可到达点位
   if sta == 0 then
       print("pos 点为机器人可到达点位")
   elseif sta== 1 then
       print("pos 点位为机器人不可到达点位，请检查")
       Pause()    --暂停
   end
```

3. local pos ={ }
local n,data,err=CCDrecv(“CAM1”) --接收视觉 CAM1 的数据
if err == 0 and n ~=nil then
for i=1,n do --n 决定循环的次数
print(i,data[i][1],data[i][2],data[i][3]) --打印每组数据
if data[i][1]~=0 or data[i][2]~=0 then--判断数据不同时为零
pos.x=data[i][1] --数据 data[i][1]赋值给 pos.x
pos.y=data[i][2] --数据 data[i][2]赋值给 pos.y
pos.z=0
pos.c=data[i][3] --数据 data[i][3]赋值给 pos.c
pos.h = 0
sta = targetok(pos)
if sta == 0 then
print(“pos 点为机器人可到达点位”)
MovP(pos) --点到点方式运动到点 pos
elseif sta== 1 then
print(“pos 点位为不可到达点位，请检查”)
Pause() --暂停运动
end
end
end
elseif err==0 and n == nil then
print(“数据格式配置错误”)
elseif err~=0 then
print(“网线断开或 IP 配置错误或网络超时”)
end

注：判断目标点是否为机器人可到达点位时，一定要指定目标点的手系，因为对于四轴机器人来说有些点位只能有一个手系可以到达。

1.17 码垛指令

1.17.1 三点法编程码垛

指令符号	指令说明
SetPlt	设置码垛数指令（用于编程码垛）
GetPlt	取码垛数据点指令（用于编程码垛）

SetPlt	
使用说明	设置码垛参数
语法说明	平面码垛 SetPlt(A,B,C,D,E,F) 三维码垛 SetPlt(A,B,C,D,E,F,G,H)
参数说明	**下表各指令参数说明**

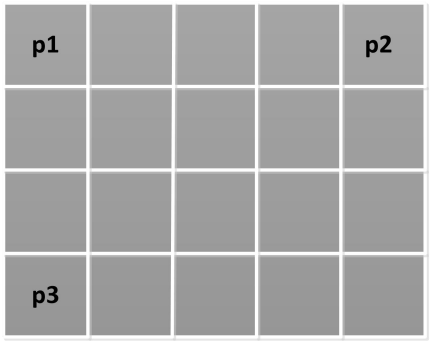
	平面码垛	SetPlt(A,B,C,D,E,F)
	A	码垛号（number 类型数据）
	B	码垛原点位置
	C	码垛行方向的最后一个点
	D	码垛列方向的最后一个点
	E	码垛的行数
	F	码垛的列数
	三维码垛	SetPlt(A,B,C,D,E,F,G,H)
	A	码垛号（number 类型数据）
	B	码垛原点位置
	C	码垛行方向的最后一个点
	D	码垛列方向的最后一个点
	E	码垛层方向的最后一个点
	F	码垛的行数
	G	码垛的列数
	H	码垛的层数
举例说明	1. SetPlt(1,p1,p2,p3,5,11)	--设置 XY 轴码垛
	2. SetPlt(1,p1,p2,p3,p4,11,5,3)	--设置 XYZ 轴码垛

◆ 注意: 码垛设置指令只要在程序开始执行一次，指令自动根据参数的个数判断是 XY 轴码垛或 XYZ 轴码垛。

GetPlt

使用说明	获得码垛数据点	
语法说明	平面码垛	GetPlt(A,B,C)
	三维轴码垛	GetPlt(A,B,C,D)
参数说明	返回值	返回码垛中的各个点位数据
	下表各指令参数说明	
	平面码垛	GetPlt(A,B,C)
	A	码垛号（number 类型数据）
	B	码垛行方向上的某一点，从 1 开始
	C	码垛列方向上的某一点，从 1 开始
	三维码垛	GetPlt(A,B,C,D)
	A	码垛号（number 类型数据）
	B	码垛行方向上某一点，从 1 开始
	C	码垛列方向上某一点，从 1 开始
	D	码垛层方向上某一点，从 1 开始

举例说明




```
SetPlt(1,p1,p2,p3,5,4)              --设置平面码垛
while true do
    i=1
    j=1
    while i <=5 do
        j=1
        while j <=4 do
            pos = GetPlt(1,i,j)          --循环指令中读取码垛点位数据
            j = j +1
            MovP(pos)                    --运动到码垛位置
        end
        i =i+ 1
    end
end                                      --主程序结束
```

- 注意： GetPlt 指令与 SetPlt 指令是组合配套使用，应用于矩形阵列码垛；
- 编程码垛是通过 AR 程序实现的，码垛中涉及到的参数（例如：码垛名、行数、列数等）都是在 AR 程序中设定的，故称作编程码垛；
- 编程码垛须添加码垛库（SetPlt 和 GetPlt 这两个指令是在 pallet.lib 库里面封装定义的）。

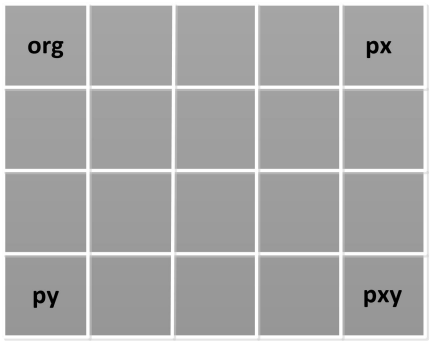
1.17.2 四点法编程码垛

指令符号	指令说明
SET_PLT	设置码垛数指令（用于编程码垛）
GET_PLT	取码垛数据点指令（用于编程码垛）

SET_PLT		
使用说明	设置码垛参数	
语法说明	SET_PLT(No,org,px,py,pxy,pxyz,nx,ny,nz)	
参数说明	**下表各指令参数说明**	
	No	码垛号（number 类型数据）
	org	码垛原点位置
	px	码垛行方向的最后一个点
	py	码垛列方向的最后一个点

	pxy	码垛第一层对角线的点位
	pxyz	码垛最后一层对角线的点位
	nx	码垛的行数
	ny	码垛的列数
	nz	码垛的层数，nz=1 为平面码垛，nz 大于 1 为三维码垛
举例说明	1. SET_PLT(1,p1,p2,p3,p4,p5,5,4,1) --平面码垛 2. SET_PLT(1,p1,p2,p3,p4,p5,5,4,3) --三维码垛	

GET_PLT

使用说明	获得码垛数据点	
语法说明	pos =GET_PLT(No,num)	
参数说明	返回值 返回码垛中的各个码垛点位置数据 **下表各指令参数说明** 平面码垛 GetPlt(A,B,C)	
	No	码垛号（number 类型数据）
	num	当前码垛的序列号
举例说明		

```
local org = p1
local px = p2
local py = p3
local pxy = p4
local pxyz = p4
local nx,ny,nz = 5,4,1
local pos = {}
SET_PLT(1,org, px,py,pz,pxy,pxyz,nx,ny,nz)    --设置平面码垛
while true do
    for i =1, nx*ny*nz do
        pos = GET_PLT(1,i)
        print(pos.x,pos.y,pz.z,pos.c)
        MArchP(pos,0,10,10)
    end
end
```

- 注意： GET_PLT 指令与 SET_PLT 指令是组合配套使用，应用于矩形阵列码垛；
- 编程码垛是通过 AR 程序实现的，码垛中涉及到的参数（例如：码垛名、行数、列

数等）都是在 AR 程序中设定的，故称作编程码垛；

- 编程码垛须添加码垛库（GET_PLT 和 SET_PLT 这两个指令是在 pallet.lib 库里面封装定义的）；
- 四点法码垛相对于三点法码垛须多示教一个码垛点位，但是精度高于三点法；故在某些应用中出现码盘对角线位置附近的某些点出现偏差时须考虑用四点法码垛来替代三点法码垛。

1.17.3 配置码垛

指令符号	指令说明
GetPLTPos	获取码盘是否码满、码盘当前码垛的个数以及码垛点位置信息（适用于配置码垛）
ResetPLT	重置码垛个数（适用于配置码垛）

GetPLTPos

使用说明	获取码盘是否码满、码盘当前码垛的个数以及码垛点位置信息		
语法说明	flag,num,pos=GetPLTPos(“PltName”)		
参数说明	无返回值		
	输入参数说明		
	PltName	码垛名（PLT0~PLT4）	
	flag	码盘是否码满的标志位	
		0	表示没有码满
		1	表示码满
		2	表示最后一个为不码垛点,码满的情况
	num	当前码垛的个数	
	pos	当前码垛点的位置	

注：使用该语句，须在码垛配置中设置关于码垛的一些参数，包括码垛名、码垛顺序、参考坐标系等。

ResetPLT

使用说明	重置码垛个数	
语法说明	ResetPLT(“PltName”,count)	
参数说明	**返回值**	
	无	
	输入参数说明	
	PltName	码垛名（PLT0~PLT4）
	count	码垛个数，可任意设置个数，码垛个数从 1 开始
举例说明	local posReady = p1	--待机点&安全点
	local quliao = p2	--取料点
	local flag	
	local num	

```
local pos ={}
MArchP(posReady,5,1,1)      --待机点&安全点
while true do
    MovP(quliao)            --运动到取料点
    DO(1,ON)
    Delay(150)
    flag,num,pos =GetPLTPos("PLT0")
    if flag == 0 then       --未码完
        MovP(pos)           --运动到码垛点
        DO(1,OFF)
        Delay(150)
    elseif flag ==1 then    --码满
        MovP(pos)           --运动到码垛点
        DO(1,OFF)
        Delay(150)
        ResetPLT("PLT0",1)  --重置码垛
    elseif flag==2 then --码盘的最后一个点为不码垛点，码完的返回值
        ResetPLT("PLT0",1)  --重置码垛
    end
end
```

- 注意：GetPLTPos 指令与 ResetPLT 指令是组合配套使用的。
- 配置码垛是通过在**码垛配置界面**设定码垛工艺用到的参数（例如：码垛名、行数、列数、间隔数等），然后结合 AR 程序实现码垛工艺，故称作配置码垛；
- 配置码垛须添加码垛库（GetPLTPos 与 ResetPLT 这两个指令是在 pallet.lib 库里面封装定义的）。

1.17.4 圆形码垛

指令符号	指令说明
SetArcPlt	圆形码垛设置
GetArcPlt	获取圆形码垛当前位置

SetArcPlt

使用说明	设置圆形码垛参数	
语法说明	平面码垛 SetArcPlt(A,B,C,D,E) 三维码垛 SetArcPlt(A,B,C,D,E,F,G)	
参数说明	**下表各指令参数说明** 平面码垛 SetArcPlt(A,B,C,D,E)	
	A	码垛号（number 类型数据）
	B	码垛原点，即圆形码垛的圆心位置
	C	码垛起点位置
	D	码垛终点位置

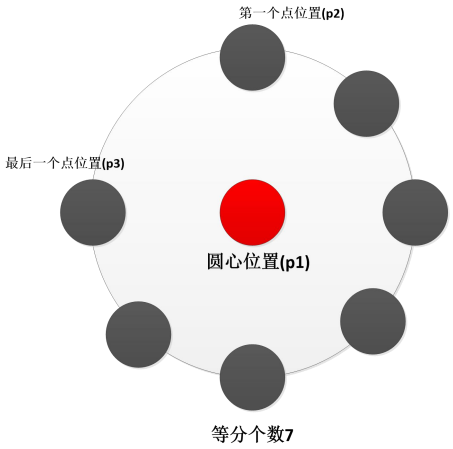
	E	法向方向码垛等分个数（第一层起点与终点之间沿着圆周方向的间隔数）
	三维码垛	SetArcPlt(A,B,C,D,E,F,G)
	A	码垛号（number 类型数据）
	B	码垛原点，即圆形码垛的圆心位置
	C	码垛第一层起点位置
	D	码垛第一层终点位置
	E	Z 方向最后一层起点位置
	F	法向方向码垛等分个数（第一层起点与终点之间沿着圆周方向的间隔数）
	G	空间码垛层数
举例说明	1. SetArcPlt(1,p1,p2,p3,7) --设置 XY 轴码垛，如下图所示 2. SetArcPlt(1,p1,p2, p3, p4,7,3) --设置空间（三维）码垛	

◆ 注意：圆形码垛设置指令只要在程序开始执行一次，指令自动根据参数的个数判断是 XY 轴码垛或 XYZ 轴码垛。

GetArcPlt

使用说明	获取圆形码垛上每个点的位置	
语法说明	GetArcPlt(A,B,C,D)	
参数说明	A	码垛号（number 类型数据）
	B	从第一个位置点起的第几个位置
	C	圆形码垛当前处于的层数（第几层）
	D	法向顺时针或者逆时针，1 为顺时针，0 为逆时针

举例说明




```
SetU(1)          --以圆形码盘上的六个点建立的圆形用户坐标系 1
local posready = p10                --待机点（用户 1 下示教）
local place = p1                    --放料点（用户 1 下示教）
local Arc_org = {x=0,y=0,z=0,c=0}   --码垛圆心位置（用户 1 原点）
local Arc_start = p2          --码垛第一层起点位置（用户 1 下示教）
local Arc_end = p3            --码垛第一层终点位置（用户 1 下示教）
local Arc_layer = p4          --码垛最后一层终点位置（用户 1 下示教）
```

```
local num = 7                    --起点(p2)和终点(p3)之间(顺时针)的等分数 7
local layer = 6                  --圆形码垛层数（p2 点与 p4 点之间的间隔数）
SetArcPlt(1,Arc_org,Arc_start,Arc_end,Arc_layer,num,layer) --设置平面圆形码垛
MArchP(posready,0,10,10)
while true do
    i = 1
    j = 1
    for j =1,layer do
        for i =1,num do
            pos = GetArcPlt(1,i,j,1) --读取码垛点位数据（顺时针）
            MArchP(pos,0,10,10)          --运动到码垛位置
            Delay(5)
            MArchP(place,0,10,10)        --运动到放料点
        end
    end                              --主程序结束
end
```

- 注意：GetArcPlt 指令与 SetArcPlt 指令是组合配套使用，应用于圆形码垛；
- 配置码垛须添加码垛库（GetArcPlt 与 SetArcPlt 这两个指令是在 **pallet.lib** 库里面封装定义的）。
- 圆形码垛一定先以圆形码盘建立一个圆形用户坐标系（六点法）,因此圆形码盘圆心的位置为{x=0,y=0,z=0,c=0}

1.18 伺服管理指令

指令符号	指令说明
MotOn	伺服所有轴打开使能
MotOff	伺服所有轴关闭使能
DragMode	AR 程序中实现拖拽功能

MotOn

使用说明	伺服所有轴使能打开
语法说明	MotOn()
参数说明	无参数
举例说明	MotOn()　　--所有轴使能打开

MotOff

使用说明	伺服所有轴使能关闭
语法说明	MotOff()
参数说明	无参数
举例说明	MotOff()　　--所有轴使能关闭

DragMode	
使用说明	AR 程序实现拖拽工能
语法说明	DragMode()
参数说明	无参数
举例说明	MovP(p1) MotOff()　　--先断开断使能 DragMode()　--再切换到拖拽模式

注：AR 程序中实现自动模式下拖拽模式的切换，一定要在断使能的情况下使用。

1.19 通信指令

指令符号	指令说明
RecCom	从 RS232 串口接收数据
SendCom	往 RS232 串口发送数据
ClrCom	清除 RS232 串口接收缓冲区
sysnetclr	清除网络数据
sysnetget	查询方式读取网络数据
sysnetsend	发送网络数据
sysnetcatch	阻塞式读取网络数据
CloseNet	关闭 TCP 网络连接
OpenNet	创建 TCP 网络
ConnectNet	连接TCP网络
RecvNet	网络接收数据功能函数
WriteNet	网络发送数据功能函数
publicread	读取全局表数据值
publicwrite	写入数据值到全局表

RecCom			
使用说明	无协议 RS232 串口通讯接收数据		
语法说明	Err,RecBuf=RecCom(A, "Time=B")		
参数说明	**指令参数说明**		
	A	通信端口号 1	
	B	接收数据超时时间，单位为毫秒	
	返回值		
	RecBuf	接收数据缓冲区	
	Err	0	表示接收成功
		非 0	表示接收超时
举例说明	local　RecBuf　　--定义接收缓冲区 local　Err　　--定义接收错误号 Err,RecBuf = RecCom(1,"Time=5000")　--串口 1 接收数据，接收超时时间		

为 5 秒
print(RecBuf.buff) --RecBuf 缓冲区中接收到上位机发送的数据
end

注：通过 RecCom 指令接收到的数据存储在缓冲区 RecBuf 的 buff 变量中；调用方法为：**RecBuf.buff**

SendCom

使用说明	无协议 RS232 串口通讯发送数据
语法说明	SendCom(A, “B”)
参数说明	**指令参数说明**

A	通信端口号 1
B	发送的数据，可以是ASCII码类型，也可以使TABLE十六进制类型数据，指令自动判断参数类型进行发送

举例说明

1. local SendBuf={0x01,0x05,0x00,0x1A,0xff,0x00}
   SendCom(1,SendBuf) --串口 1 以十六进制方式发送数据
2. SendCom(1, “ROBOT”) --发送字符串
3. pos =getcart()
   SendCom(1,string.format(“%f,%f,%f,%f;”,pos.x,pos.y.pos.z,pos.c)) --将数值变量 pos.x,pos.y,pos.z,pos.c 以字符串的形式发送
   SendCom(1, “abc,123,34,56”) --发送字符串

ClrCom

使用说明	无协议 RS232 串口通讯清除接收数据缓冲区
语法说明	ClrCom(A)

参数说明	A 通信端口号 1

举例说明
ClrCom(1) --清除端口号 1 中串口数据
Err,RecBuf = RecCom(1,"Time=5000") --清除串口 1 接收缓冲区后再接收

注：串口通讯端口号只能为 1，且示教器里的 7 号参数要设置成“无协议”。

sysnetclr

使用说明	清除网络数据
语法说明	sysnetclr(ipaton{“A”},B)
参数说明	**返回值** 无返回值 **指令参数说明**

A	网络通讯 IP 地址
B	网络通讯端口号

举例说明
local CameraNet={ipaton("192.168.0.100"),8080} -- IP：192.168.0.100；端口号 8080
sysnetclr(CameraNet) --清除网络数据

sysnetget

使用说明	查询方式读取网路数据
语法说明	Err,RecBuf = sysnetget({ipaton("A"),B})
参数说明	**指令参数说明**

A	网络通讯 IP 地址	
B	网络通讯端口号	
返回值		
Err	接收错误号	
	0	表示接收成功
	非 0	表示接收失败
RecBuf	接收数据缓冲区	

举例说明

```
local CameraNet={ipaton("192.168.0.100"),2000}     -- IP： 192.168.0.100；端口号 2000
local Err
local RecBuf
Err,RecBuf =sysnetget(CameraNet)                    --扫描式接收数据
```

注：通过 sysnetget 指令接收到的数据存储在缓冲区 RecBuf 的 buff 变量中；调用方法为：**RecBuf.buff**

sysnetsend

使用说明	发送网络数据
语法说明	sysnetsend({ipaton("A"),B},C)
参数说明	返回值　无返回值

指令参数说明

A	网络通讯 IP 地址
B	网络通讯端口号
C	网络发送的数据

举例说明

local CameraNet={ipaton(“192.168.0.100”),2000}　--IP： 192.168.0.100；端口号 2000

1. local data={0x23,0x11,0x33}
   sysnetsend(CameraNet,data)　　--发送数组数组
2. local data = "trigger"
   sysnetsend(CameraNet,data)　　--发送字符串
   sysnetsend(CameraNet, “start”)　　--发送字符串
3. local　value = 123.67
   sysnetsend(CameraNet,string.format(“%f”,value))　--将 number 变量以字符串发送

sysnetcatch

使用说明	阻塞式读取网路数据

语法说明	Err,RecBuf = sysnetcatch({ipaton("A"),B},C)
参数说明	**指令参数说明**

A	网络通讯 IP 地址	
B	网络通讯端口号	
C	阻塞时间，单位 ms	
返回值		
Err	接收错误号	
	0	表示接收成功
	非 0	表示接收失败
RecBuf	接收数据缓冲区	

举例说明	local CameraNet={ipaton("192.168.0.100"),2000}　--IP：192.168.0.100；端口号 2000 local Err local RecBuf Err,RecBuf = sysnetcatch(CameraNet,2000) --阻塞式接收数据，阻塞时间 2000ms

注：通过 sysnetcatch 指令接收到的数据存储在缓冲区 RecBuf 的 buff 变量中；调用方法为：**RecBuf.buff**

CloseNet

使用说明	关闭 TCP 网络连接
语法说明	CloseNet ({ipaton("A"),B})
参数说明	**指令参数说明**

A	TCP 通讯，RC400 控制器作为客户端时视觉设备的 IP 地址
B	TCP 通讯，RC400 控制器作为客户端时视觉设备的端口号

OpenNet

使用说明	创建 TCP 网络
语法说明	OpenNet({ipaton("A"),B})
参数说明	**指令参数说明**

A	TCP 通讯，RC400 控制器作为客户端时视觉设备的 IP 地址	
B	TCP 通讯，RC400 控制器作为客户端时视觉设备的端口号	
返回值		
Err	接收错误号	
	0	表示 TCP 网络打开成功
	非 0	表示 TCP 网络打开失败

ConnectNet

使用说明	连接 TCP 网络

语法说明	ConnectNet({ipaton(“A”),B})		
参数说明	**指令参数说明**		
	A	TCP 通讯，RC400 控制器作为客户端时视觉设备的 IP 地址	
	B	TCP 通讯，RC400 控制器作为客户端时视觉设备的端口号	
	返回值		
	Err	接收错误号	
		0	表示 TCP 网络连接成功
		非 0	表示 TCP 网络连接失败

RecvNet

使用说明	网络接收数据功能函数		
语法说明	Err_net,RecBuf = RecvNet ({ipaton(“A”),B},C)		
参数说明	**指令参数说明**		
	A	TCP 通讯，RC400 控制器作为客户端时视觉设备的 IP 地址	
	B	TCP 通讯，RC400 控制器作为客户端时视觉设备的端口号	
	C	超时时间，单位毫秒（ms）	
	返回值		
	Err_net	0	表示接收数据成功
		非 0	表示接收数据失败
	RecBuf	接收数据缓冲区	

注：通过 RecvNet 指令接收到的数据存储在缓冲区 RecBuf 的 buff 变量中；调用方法为：**RecBuf.buff**

WriteNet

使用说明	网络发送数据功能函数
语法说明	WriteNet ({ipaton(“A”),B},C)
参数说明	**指令参数说明**
	A　TCP 通讯，RC400 控制器作为客户端时视觉设备的 IP 地址
	B　TCP 通讯，RC400 控制器作为客户端时视觉设备的端口号
	C　发送的网络数据
举例说明	local　ipPort={ipaton(“192.168.0.100”),2000}　　--ip，port CloseNet(ipPort)　　--关闭 TCP 网络 Delay(100) print("关闭 TCP 网络！") if OpenNet(ipPort) == 0 then　　--打开 TCP 网络 　　print("打开 TCP 网络！正在连接...") 　　repeat 　　　　Delay(2) 　　until ConnectNet(ipPort) == 0　　--连接 TCP 网络 　　print("已连接 TCP 网络！")

```
end
while 1 do
    local x,y,z,c
    local err_net
    err_net,RecBuf = RecvNet(ipPort,5000)      --等待接收网络数据,5 秒钟超时
    if err_net == 0 then                       --接收成功
        if RecBuf.buff == "?" then             --判断接收字符串是否为"？"
                WriteNet(ipPort,"OK")          --发送字符串 “OK”
        end
    else
        print("接收失败:",err_net)
    end
    Delay(10)
end
```

publicread

使用说明	读取全局表数据值
语法说明	用法 1：C = publicread(A, “B”) 用法 2：C = publicread(A,len, “B”)
参数说明	**指令参数说明**

A	全局数据表中的地址
B	可选参数，读取数据的方式，整型（可省略）/浮点型/十六进制
len	连续读取地址的个数

返回值

C	读取全局表中地址对应的数据值

举例说明

1. local a = publicread(0x100,"float") --AR 以浮点型方式读地址 0x100 的值
2. local b = publicread(0x102) --AR 以整型方式读地址 0x102 的值
3. local c = publicread(0x100,3) --从 0x100 开始连续读取 3 个地址（0x100,0x102,0x104）的数值(整型),返回的值存储在数组 c 中， c[1]为地址 0x100 里面的值，依次类推
4. local c = publicread(0x100,3,”float”) --从 0x100 开始连续读取 3 个地址（0x100,0x102,0x104）的数值(浮点型),返回的值存储在数组 c 中，c[1]为地址 0x100 里面的值，依次类推

publicwrite

使用说明	写入数据值到全局表
语法说明	用法 1：publicwrite(A,B,"D") 用法 2：publicwrite(A,C, “D”)
参数说明	返回值 无返回值

	指令参数说明	
	A	全局表数据值的地址
	B	写入到对应地址的数据
	C	写入到连续地址中的数组
	D	写入数据的方式，整型（可省略）/浮点型/十六进制
举例说明	1. publicwrite(0x102,10.5,"float") --浮点型方式写入到地址 0x100 的值 2. publicwrite(0x102,5) --以整型方式写入到地址 0x102 的值 3. publicwrite(0x100,{100,200,300}) –将数组{100,200,300} (整型)分别写入到以 0x100 为起始地址的连续 3 个地址中（0x100,0x102,0x104）） 4. publicwrite(0x100,{10.1,200.1,300.1}, “float”) –将数组{10.1,20.1,30.1}(浮点型)分别写入到以 0x100 为起始地址的连续 3 个地址中（0x100,0x102,0x104））	

1.20 视觉指令

指令符号	指令说明
initTCPnet	网络通讯初始化
CCDrecv	接收视觉返回的坐标数据
CCDtrigger	触发相机拍照
CCDsent	发送字符串到视觉
CCDclr	清除网络托管的 IP
CCDoffset	视觉偏差补偿
GetDynCCDPos	计算动态视觉位置
CCDGet	接收视觉返回的字符串类型数据

InitTCPnet

使用说明	网络通讯初始化	
语法说明	initTCPnet(“CamName”)	
参数说明	CamName	视觉名称(字符串类型)，在视觉 UI 配置时生成

举例说明

```
local pos={}
initTCPnet("CAM1")                    --步骤 1：初始化网络 IP 以及端口
while 1 do
    Delay(10)
    local n,data=CCDrecv("CAM1")    --接收相机 CAM1 发送的网络数据
    if data then                    --如果接收数据不为空，则进行 for 循环
        for i=1,n do
            if data[i][1]~=0 or data[i][2]~=0 then
                print(data[i][1],data[i][2],data[i][3])
                --如果接收的视觉数据 X，Y 不为 0，则数据有效
                pos.x=data[i][1]      --data[i][1]赋值给 pos.x
                pos.y=data[i][2]      --data[i][2]赋值给 pos.y
```

```
                    pos.c=data[i][3]      --data[i][3]赋值给 pos.c
                    pos.z =-20
                    MovP(pos)
                end
            end
        end
    end
```

CCDrecv		
使用说明	接收视觉返回的坐标数据	
语法说明	n,data,err=CCDrecv(“CamName”)	
参数说明	CamName	视觉名称(字符串类型)，在视觉 UI 配置时生成
返回值	n	接收视觉数据的组数
	data	接收视觉返回的绝对坐标，可直接用于点位运动
	err	错误号；0 为正常返回，非 0 为异常返回

举例说明

```
local pos ={ }
local n,data,err=CCDrecv(“CAM1”)        --接收视觉 CAM1 的数据
if err == 0 and n ~=nil then
    for i=1,n do                                    --n 决定循环的次数
        print(i,data[i][1],data[i][2],data[i][3])    --打印每组数据
        if data[i][1]~=0 or data[i][2]~=0 then    --判断数据不同时为零
            pos.x=data[i][1]                --数据 data[i][1]赋值给 pos.x
            pos.y=data[i][2]                --数据 data[i][2]赋值给 pos.y
            pos.z=0
            pos.c=data[i][3]                --数据 data[i][3]赋值给 pos.c
            MovP(pos)                       --点到点方式运动到点 pos
        end
    end
elseif err==0 and n == nil then
    print(“数据格式配置错误”)
elseif err~=0 then
    print(“网线断开或 IP 配置错误或网络超时”)
end
```

CCDtrigger		
使用说明	触发相机拍照	
语法说明	CCDtrigger(“CamName”)	
参数说明	CamName	视觉名称(字符串类型)，在视觉 UI 配置时生成，可根据设置自动识别 网络触发 or IO 触发
返回值	无返回值	
举例说明	CCDtrigger(“CAM1”)	

CCDsent

使用说明	发送字符串到视觉	
语法说明	CCDsent(“CamName”,Buff)	
参数说明	CamName	视觉名称(字符串类型)，在视觉 UI 配置时生成
	Buff	发送的字符串
返回值	无返回值	
举例说明	1. CCDsent(“CAM2”, “123”) 2. CCDsent(“CAM1”, “[0]”) 3. CCDsent(“CAM0”, “0,0,0,0;”)	

CCDclr

使用说明	清除网络托管的 IP	
语法说明	CCDclr(“CamName”)	
参数说明	CamName	视觉名称(字符串类型)，在视觉 UI 配置时生成
返回值	无返回值	
举例说明	CCDclr(“CAM2”)	

CCDoffset

使用说明	视觉偏差补偿	
语法说明	pos = CCDoffset(“CamName”,view_pos)	
参数说明	CamName	视觉名称(字符串类型)，在视觉 UI 配置界面生成
	view_pos	接收的视觉坐标
返回值	pos	输出补偿后的绝对坐标
举例说明	local view_pos={ } local pos = { } view_pos = {x=245,y=200,z=0,c=30,h=1} pos = CCDoffset(“CAM0”,view_pos)	

◆ 此函数中涉及到的补偿值是在视觉配置界面设定：dx、dy、dc

GetDynCCDPos

使用说明	动态视觉位置计算	
语法说明	robot_pos= GetDynCCDPos(“CamName”,view_pos)	
参数说明	CamName	视觉名称(字符串类型)，在视觉 UI 配置时生成
	view_pos	动态相机坐标系下的坐标
返回值	robot_pos	输出计算后的绝对坐标（基坐标下或当前用户坐标系下）
举例说明	local view_pos={ } local robot_pos = { }	

	view_pos.x=300 view_pos.y=100 view_pos.z=0 view_pos.c=30 robot_pos=GetDynCCDPos(“CAM4”,view_pos)

CCDGet		
使用说明	接收视觉返回的字符串类型数据	
语法说明	RecBuf=CCDGet(“CamName”)	
参数说明	CamName	视觉名称(字符串类型)，在视觉 UI 配置时生成
返回值	RecBuf	接收视觉返回的数据
举例说明	local RecBuf = CCDGet(“CAM1”) if RecBuf.buff == “OK” then MovP(p1) elseif RecBuf.buff == “NG” then MovP(p2) end	

注:若外部设备（例如，视觉）发送给机器人的是字符串类型的数据，则字符串存储在数组 RecBuf 中的 RecBuf.buff 变量中，字符串的长度储存在 RecBuf.len 变量中。

1.21 动态跟随指令

指令符号	指令说明
FollowInit	动态跟随初始化
SetDynCatch	开启或关闭跟随抓取任务
GetCatchSpace	获取物件是否进入抓取区域
SetCatch	执行跟随
GetCatchState	获取抓取状态
SynOver	结束跟随
GetTrigger	获取触发状态（同一物体拍两次使用）
SetViewData	发送数据给控制器，存入缓存待跟随
FastCatch	动态跟随快速抓取，拱形运动到抓取位置
GetSynPos	获取进入抓取区工件的信息

FollowInit		
使用说明	动态跟随初始化	
语法说明	FollowInit(“CamName”)	
参数说明	CamName	视觉名称(字符串类型)，在 UI 配置时生成

返回值	无返回值
举例说明	1. FollowInit(“CAM1”) --初始化动态跟随视觉 CAM1

SetDynCatch

使用说明	开启或关闭动态抓取任务		
语法说明	SetDynCatch(n)		
参数说明	n	开启或关闭标志(可选参数)，范围 0~1;	
		0	关闭跟随抓取任务
		1	开始跟随抓取任务
举例说明	1. SetDynCatch(0) --关闭跟随任务 2. SetDynCatch(1) --开始跟随任务		

GetCatchSpace

使用说明	获取物件是否进入抓取区域	
语法说明	state=GetCatchSpace()	
参数说明	无参数	
返回值	0	未进入抓取区
	1	已进入抓取区

SetCatch

使用说明	执行跟随 （待物件进入抓取区后可执行此命令进行跟随）
语法说明	SetCatch ()
参数说明	无参数: 默认开始抓取高度为等待抓取高度
	有参数: 设置开始同步抓取高度
返回值	无返回值

GetCatchState

使用说明	获取抓取状态	
语法说明	state=GetCatchState ()	
参数说明	无参数	
返回值	0	抓取结束，即正常抓取结束
	1	开始运动，即从当前点运行到目标点，并同速同位。
	2	进入同步，已经同步成功，已开始同速同位。
	3	错误退出，超出抓取区，无法完成同步抓取。

SynOver

使用说明	结束跟随（待完成同步中工艺后，可执行此命令进行退出同步）
语法说明	SynOver()
参数说明	无参数
返回值	无返回值

GetTrigger

使用说明	获取触发状态，用来分辨同一物体拍两次时的先后顺序	
语法说明	num=GetTrigger()	
参数说明	无参数	
返回值	0	为第一次拍照后到第二次拍照前标志
	1	为第二次拍照后再到第一次拍照前标志

SetViewData

使用说明	机器人控制器将接收到的视觉数据存入到动态抓取缓存队列待跟随	
语法说明	SetViewData (pos)	
参数说明	pos	视觉接收到的点位数据

举例说明	local pos={x=0,y=0,z=0,c=0,h= 0}
	local n,data=CCDrecv("CAM1") --动态抓取接收相机 CAM1 发送的网络数据
	if data then --如果接收数据不为空，则进行 for 循环
	for i=1,n do
	if data[i][1]~=0 and data[i][2]~=0 then
	pos.x=data[i][1] --data[i][1]赋值给 pos.x
	pos.y=data[i][2] --data[i][2]赋值给 pos.y
	pos.c=data[i][3] --data[i][3]赋值给 pos.c
	SetViewData(pos) --发送数据给控制器
	end
	end
	end	

FastCatch

使用说明	动态跟随快速抓取，拱形运动到抓取位置	
语法说明	FastCatch(A,B,C,D)	
参数说明	A	抓取物体时 Z 轴的绝对高度
	B	抓取物体时 Z 轴的限位高度
	C	Z 轴从等待位置上升的高度
	D	Z 轴相对于抓取位置上方下降的高度
举例说明	FastCatch(-80,-10,10,10) -- -80：抓取传送带上物体时 Z 轴的高度为-80mm; -- -10：从待机点到等待点运动过程中 z 轴的绝对限位高度为-10mm;	

-- 10：Z 轴从待机位置(等待抓取位置)上升 10mm；
-- 10：Z 轴从抓取位置上方 10mm 处下降的高度；

GetSynPos		
使用说明	获取进入抓取区工件的信息	
语法说明	Flag, pos = GetSynPos()	
参数说明	无参数	
返回值	Flag	判断物体进入抓取区的标志位，成功返回 0
	pos	进入抓取区工件的信息（数组类型数据）：坐标信息（pos.x\pos.y\pos.z\pos.c\pos.h）以及模板号信息(pos.n)

注：动态跟随这一章节的指令不能单独使用，须组合才能完成动态跟随工艺；

此章节只是对这几个指令做简单的介绍说明，具体应用请参考动态跟随应用 AR 程序或咨询厂家。

1.22 调试指令

指令符号	指令说明
print	打印输出用户调试数据
Error	终止程序运行，并输出错误信息

print	
使用说明	从 RS232 串口输出调试数据
语法说明	print(…)
参数说明	可变参数，参数个数和类型可以任意
举例说明	1. print(12) --从串口输出数据 12 2. print (“Robot”) --从串口输出字符串数据 Robot 3. local a=10 print ("Robot",a) --从串口输出字符串数据 Robot 和数字 10

Error	
使用说明	终止程序运行，并输出错误信息
语法说明	Error(A)
参数说明	用于输出用户定义的错误信息，字符串类型
举例说明	Error(“程序运行错误”) --执行后在界面输出报警信息并终止程序运行

1.23 点位指令

指令符号	指令说明
Point	调用点位表中的点位

new	将要更新的点位写入到对应工程中的点位表里
teach	将当前点位示教到对应工程中的点位表里
ReadTabDat	加载当前工程中的点位数据表

Point

使用说明	调用点位表中的点位	
语法说明	B= Point (A)	
参数说明	A	点位表 DATA 中的点位数据，范围 1~2999
返回值	B	将点位表中点位对出赋值给 B
举例说明	local pos={} pos=Point(10) --调用点位数据表中的 p10 点赋值给 pos print(pos.x,pos.y,pos.z,pos.c)	

注：A 的值只能为 1~2999 中的数字，不能写成 p1~p2999。

new

使用说明	将要更新的点位写入到对应工程中的点位表里	
语法说明	new(A,B,C,D,E,F,G)	
参数说明	A	点位表 DATA 中的点位序号，范围 1~2999
	B	新点位中的 x 坐标值
	C	新点位中的 y 坐标值
	D	新点位中的 z 坐标值
	E	新点位中的 c 坐标值
	F	新点位中的手系 h
	G	可选参数（可填、可不填）。新点位的名字（string 类型）
返回值	无	
举例说明	1. print(pos.x,pos.y,pos.z,pos.c) new(5,300,100,-10,30,0) --将点位（300,100,-10,30）写入到 DATA.PTS 中的 p5 点； 2. new(1,300,100,-10,30,0, “fangliao”) --将点位（300,100,-10,30）写入到 DATA.PTS 中的 p1 点, p1 点注释为 fangliao 点； 3. local pos = getcart() new(10,pos.x,pos.y.pos.z,pos.c,pos.h) --将获取的当前点位写入到 DATA.PTS 中的 p10 点	

teach

使用说明	将当前点位示教到对应工程中的点位表里	
语法说明	teach (A)	
参数说明	A	点位表 DATA 中的点位数据，范围 1~2999
举例说明	teach(10) --将当前点位写入到 DATA.PTS 中的 p10 点；	

ReadTabDat		
使用说明	加载当前工程中的点位数据表	
语法说明	B= ReadTabDat (“A”)	
参数说明	A	数据表的名字（数据表为 **.PTS** 格式的文件）
返回值	B	标志位，B 等于 0 加载成功，B 小于 0 加载失败

举例说明	DataPath = “newdata.PTS” --文件名 newdata 数据表存放在当前工程目录下
	local flag = ReadTabDat(DataPath) --加载 newdata 数据表
	if flag == 0 then
	  print(“数据表加载成功”)
	elseif flag<0 then
	  print(“数据表加载失败”)
	end	

1.24 系统指令

指令符号	指令说明
syswork	设置系统工作状态
sysstate	读取各个轴的电流值或获取系统状态
sysrate	设置全局速度速率
systime	获取系统时钟

syswork		
使用说明	设置系统工作状态	
语法说明	syswork(A)	
参数说明	无返回值 **输入参数说明**	
	A	1 启动程序运行
		2 暂停程序运行
		3 停止程序运行
		4 程序复位
举例说明	MovP(p1) --点到点运动到 p1 点 syswork(2) --程序暂停 Delay(100) --延时 100ms syswork(1) --启动程序	

sysstate		
使用说明	读取各轴的电流值或获取系统状态	
语法说明	state = sysstate(n)	
	n	state

	空	获取机器人机器人报警信息	0：无报警
			1：伺服报警
			2：DSP 报警
			4：操作报警
			8：系统报警
	0	获取机器人的使能状态信息	0：断使能状态
			1：使能状态
	1	获取第一轴的电流值，单位毫安	
	2	获取第二轴的电流值，单位毫安	
	3	获取第三轴的电流值，单位毫安	
	4	获取第四轴的电流值，单位毫安	
	5	获取机器人的运行状态	0：空闲
			1：暂停
			2：运行
	6	获取机器人的系统模式	0：手动状态
			1：自动状态
	7	获取伺服报警号	
	8	获取 DSP 报警号	
	9	获取操作报警号	
	10	获取系统报警号	

举例说明

1.
```
local state1 = sysstate(1)    --获取第一轴的电流值
local state2 = sysstate(2)    --获取第二轴的电流值
local state3 = sysstate(3)    --获取第三轴的电流值
local state4 = sysatate(4)    --获取第四轴的电流值
```
2.
```
local state = sysstate(5)
if state == 0 then
    print(“机器人空闲状态”)
elseif state==1then
    print(“机器人处于暂停状态”)
elseif state==2 then
    print(“机器人处于运行状态”)
end
```
3.
```
local state=sysstate(6)
if state == 0 then
    print(“机器人处于手动运行模式”)
elseif state==1 then
    print(“机器人处于自动运行模式”)
end
```
4.
```
state7 = sysstate(7)      --获取伺服报警号
state8 = sysstate(8)      --获取 DSP 报警号
state9 = sysstate(9)      --获取操作报警号
```

state10 = sysstate(10)　--获取系统报警号

sysrate		
使用说明	获取或设置全局速度倍率	
语法说明	Rate = sysrate(A)	
参数说明	A	Rate
	空	获取全局速度倍率
	1~100	设置全局速度倍率
举例说明	1. sysrate()　--获取当前的全局速度倍率 2. sysrate(50)　--设置全局速度倍率为 50%	

注：　当设置全局速度倍率时，设置的倍率值必须是 1~100 之间的整数值。

systime	
使用说明	获取系统时钟
语法说明	time = systime()
参数说明	无输入 **返回值　time**
	time　系统时钟值
举例说明	local time1=systime()　--获取当前的系统时钟 MovP(p1) MovP(p2) local time2 = systime()-time1　--获取程序运行的时间 print(time2)

1.25 modbus 主站读写指令

指令符号	指令说明
ReadRegW	读指定 PLC 寄存器的 16 位字地址
ReadRegDW	读指定 PLC 寄存器的 32 位双字地址
WriteRegW	写入 16 位字地址到指定的 PLC 寄存器
WriteRegDW	写入 32 位字地址到指定的 PLC 寄存器

ReadRegW		
使用说明	读指定 PLC 寄存器的 16 位字地址	
语法说明	Value = ReadRegW ({A,B},C,D)	
参数说明	A	数据通讯协议，A=1 异步收发器（Mobbus RTU）；A=3 代表网络通讯（Mobus TCP/IP）
	B	站号

	C	寄存器地址
	D	读取变量的个数（可选），默认为 1
返回值	Value	返回读取寄存器对应地址中变量的值
举例说明	1. local plc={1,1} --PLC 通信参数使用 1 为 UART,站号为 1 ReadRegW(plc,250) --读 PLC 寄存器 250 16 位字地址 local a=ReadRegW(plc,250,20) --连续读 PLC 寄存器 250 开始的 20 个 16 位变量 for i=1,20 do print(a[i]) Delay(10) end	

ReadRegDW

使用说明	读指定 PLC 寄存器的 32 位双字地址	
语法说明	Value=ReadRegDW ({A,B},C,D,E)	
参数说明	A	数据通讯协议，A=1 异步收发器（Mobbus RTU）；A=3 代表网络通讯（Mobus TCP/IP）
	B	站号
	C	寄存器地址
	D	读取变量的个数（可选），默认为 1
	E	读取变量的类型（可选），整型和浮点型，默认为整型
返回值	Value	返回读取寄存器对应地址中变量的值

举例说明	1. local plc={1,1} --PLC 通信参数使用 1 为 UART,站号为 1 2. print(ReadRegDW(plc,250)) --读 PLC 寄存器 250 32 位双字地址，整型变量 3. print(ReadRegDW(plc,250,"float")) --读 PLC 寄存器 250 32 位双字地址，浮点型变量 4. local a=ReadRegDW(plc,250,20) --连续读 PLC 寄存器 250 开始的 20 个 32 位整型变量 5. for i=1,20 do print(a[i]) end 6. a = nil 7. local a=ReadRegDW(plc,250,20,"float") --连续读 PLC 寄存器 250 开始的 20 个 32 位浮点变量 8. fori=1,20 do print(a[i]) end 9. ReadRegDW(plc,250,0x100,10) --读 PLC 250 开始的 32 位变量到本地 0x100 变量中，共 10 个变量且为整型值 10. ReadRegDW(plc,250,0x100,10,"float") --读 PLC 250 开始的 32 位变量到本地 0x100 变量中，共 10 个变量且为浮点数 11. ReadRegDW(plc,250,{x=true,y=true,z=true,c=true,n=1},10) -- 读 PLC 250 开始的 32 位变量到 点位表中，共 10 点个变量且为整型 12. ReadRegDW(plc,250,{x=true,y=true,z=true,c=true,n=1},10,"float") --读 PLC250 开始的 32 位变量到 点位表中，共 10 点个变量且为浮点

WriteRegW

使用说明	写入 16 位字地址到指定的 PLC 寄存器	
语法说明	Value =WriteRegW ({A,B},C,D)	
参数说明	A	数据通讯协议，A=1 异步收发器（Mobbus RTU）；A=3 代表网络通讯（Mobus TCP/IP）
	B	站号
	C	寄存器地址
	D	写入地址中的数据
返回值	Value	返回写入到寄存器对应地址中变量的值
举例说明	1. local plc={1,1} --PLC 通信参数使用 1 为 UART,站号为 1 2. WriteRegW(plc,250,1) --写 PLC 寄存器 250 16 位字地址 3. WriteRegW(plc,250,{10,20,30}) --连续写 PLC 寄存器 250 16 位字地址	

WriteRegDW

使用说明	写入 32 位字地址到指定的 PLC 寄存器
语法说明	WriteRegDW ({A,B},C,D,E,F)

参数说明	A	数据通讯协议，A=1 异步收发器（Mobbus RTU）；A=3 代表网络通讯（Mobus TCP/IP）
	B	站号
	C	寄存器地址
	D	本地地址（可选）
	E	写入到寄存器的数据
	F	写入变量的类型（可选），整型和浮点型，默认为整型
举例说明	1. local plc={1,1} --PLC 通信参数使用 1 为 UART,站号为 1 2. WriteRegDW(plc,250,1) --写 PLC 寄存器 250 32 位字地址 3. WriteRegDW(plc,250,0x100,10) --写本地 0x100 变量到 PLC 250 开始的 32 位变量到,共 10 个变量且为整型值 4. WriteRegDW(plc,250,{10,20,30})--连续写 PLC 寄存器 250 32 位字地址 5. WriteRegDW(plc,250,1,"float")--写 PLC 寄存器 250 32 位字地址浮点数 6. WriteRegDW(plc,250,0x100,10,"float") --写本地 0x100 变量到 PLC 250 开始的 32 位变量到,共 10 个变量且为浮点数 7. WriteRegDW(plc,250,{10,20,30},"float") --连续写 PLC 寄存器 250 32 位字地址浮点数	

1.26 文件操作指令

指令符号	指令说明
fopen	文件打开
fsize	文件大小
fwrite	写文件，写数据到文件中
fread	读文件，从文件中读取数据(指定数据长度)
fgets	逐行获取文件中的数据
fseek	文件定位，把文件指针地址移动到一个指定位置
feof	文件结束
fclose	文件关闭

fopen

使用说明	文件打开	
语法说明	Address =fopen(path,mode)	
参数说明	path	**文件路径** 相对路径---不写盘符，文件位置在当前工程目录下，例如当前工程目录为 d:\\projects\\scara, 则文件位置为 d:\\projects\\scara\\test.txt 绝对路径---指定文件的完整路径，例如 d:\\test.txt
	mode	**打开方式** "w" ---只写 "r" ---只读 "a" ---追加方式

		"+" ---读写
返回值	Address	返回文件指针地址,大于 0 表示打开成功,0 为打开失败

fsize

使用说明	文件大小	
语法说明	size = fsize(file)	
参数说明	file	文件指针地址
返回值	size	文件大小

fwrite

使用说明	写文件，写数据到文件中	
语法说明	size = fwrite(file,data)	
参数说明	file	文件指针地址
	data	写入的数据
返回值	size	写入数据的大小,大于 0 成功,0 为失败

fread

使用说明	读文件，从文件中读取数据	
语法说明	data = fread(file,len)	
参数说明	file	文件指针地址
	len	读数据的长度
返回值	data	从文件中读取的数据

fgets

使用说明	逐行获取文件中的数据	
语法说明	data = fgets(file)	
参数说明	file	文件指针地址
返回值	data	逐行从文件中读取的数据(数组类型，包括读取的数据内容以及数据的长度)，其中数据存放在 data.buff 缓存变量中

fseek

使用说明	文件定位，把文件指针地址移动到一个指定位置	
语法说明	Err = fseek(file,offset,whence)	
参数说明	file	文件指针地址
	offset	偏移位置
	whence	"set" ：定位到开始 "cur" ：定位到当前

		"end" ：定位到结尾
返回值	Err	错误号，0 表示成功

feof

使用说明	文件结束	
语法说明	flag= feof(file)	
参数说明	file	文件指针地址
返回值	flag	文件结束标志 0 为未结束,-1 为文件尾

fclose

使用说明	文件关闭	
语法说明	Err = fclose(file)	
参数说明	file	文件指针地址
返回值	Err	错误号 0 表示成功

举例说明

```
function main()
    ----------文件只写----------
    local f=fopen("3.txt","w") --写方式打开文件名为相对路径在当前工程目录下
    if f > 0 then
        local len=fwrite(f,"write test!")
        print(len)      --写入长度
        fclose(f)
    end
    ----------文件只读----------
    local f=fopen("3.txt","r") --写方式打开 文件名为相对路径在当前工程目录下
    if f > 0 then
        print("size",fsize(f))          --文件大小
        local data=fread(f,2)
        print(data.buff,data.len)       --输出读数据 和长度
        fclose(f)
    end
    ----------文件读写----------
    local f=fopen("3.txt","wr") --文件读写方式打开
    if f > 0 then
        local len=fwrite(f,"write test!")
        print(len)      --写入长度
        fseek(f,0,"set") --定位到开始
        local data=fread(f,2)
        print(data.buff,data.len)       --输出读数据 和长度
        fclose(f)
```

```
    end

end
```

1.27 队列操作指令

指令符号	指令说明
qexist	判断队列是否存在
qcreate	创建队列
qpush	往队列中写入数据
qpop	删除队列中的首元素
qfront	取队列中的首元素
qpopfront	取队列中的首元素然后从队列中删除
qempty	判断队列是否为空
qsize	队列大小
qdestroy	删除队列

qexist

使用说明	判断队列是否存在	
语法说明	Flag = qexist(ID)	
参数说明	ID	队列的名字，类型：整形数字
返回值	Flag	Flag 为 true，队列存在；Flag 为 false，队列不存在

qcreate

使用说明	创建队列	
语法说明	qcreate(ID)	
参数说明	ID	队列的名字，类型：整形数字
返回值	无	

qpush

使用说明	往队列里写入数据	
语法说明	qpush(ID,data)	
参数说明	ID	队列的名字，类型：整形数字
	data	写入到队列中的数据，数据类型可以是数字、字符串、数组等，或者组合
返回值	无	

qpop

使用说明	删除队列中的首元素	
语法说明	qpop(ID)	
参数说明	ID	队列的名字，类型：整形数字
返回值	无	

qfront

使用说明	取队列中的首元素	
语法说明	Data=qfront(ID)	
参数说明	ID	队列的名字，类型：整形数字
返回值	Data	队列中第一个元素的值

qpopfront

使用说明	取队列中的首元素然后从队列中删除	
语法说明	Data=qpopfront(ID)	
参数说明	ID	队列的名字，类型：整形数字
返回值	Data	队列中第一个元素的值

qempty

使用说明	判断队列是否为空	
语法说明	Flag =qempty(ID)	
参数说明	ID	队列的名字，类型：整形数字
返回值	Flag	Flag 为 1，队列为空；Flag 不为 1，队列中有数据

qsize

使用说明	队列的大小	
语法说明	Len=qsize(ID)	
参数说明	ID	队列的名字，类型：整形数字
返回值	Len	队列的长度

qdestroy

使用说明	删除队列	
语法说明	qdestroy(ID)	
参数说明	ID	队列的名字，类型：整形数字
返回值	无	
举例	flag1 = qexist(1)　　--判断 1 号队列是否存在 if flag1 == true then 　　print("OK")	

```
elseif flag1 ==false then    --若不存在创建 1 号队列
    qcreate(1)
end
qpush(1,123)                 --将 123 存入到 1 号队列
qpush(1,"abc")               --将字符串 abc 存入到 1 号队列
qpush(1,{x=300,y=100,z=0,c=30})   --将数组存入到 1 号队列
len = qsize(1)               --求取队列的长度（元素的个数）
print(len)                   --3
while qempty(1)~=1 do        --判断 1 号队列是否为空，非 1 表示不为空
    data = qfront(1)         --取 1 号队列中的首元素
    print(data)              --打印
    qpop(1)                  --删除 1 号队列中的首元素
end
qdestroy(1)                  --删除 1 号队列
```