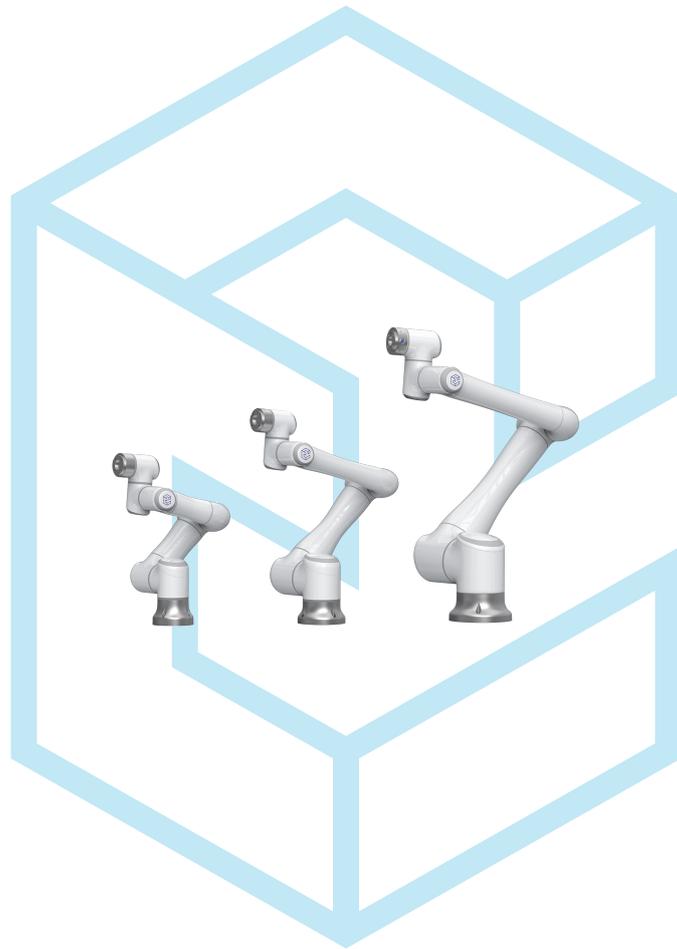


ELITE ROBOT EC系列

编程手册



JBI 脚本手册

苏州艾利特机器人有限公司

2021-12-24

版本：6.0.59

目录

1 示教器交互式编程	2
2 JBI 脚本编程	5
2.1 JBI 脚本关键字	5
2.2 JBI 脚本格式	6
2.3 JBI 流程控制语句	7
2.3.1 IF 指令	7
2.3.2 WHILE 指令	7
2.3.3 CALL 和 JUMP 指令	8
2.3.4 暂停指令	9
2.3.5 等待指令	9
2.3.6 延时指令	10
2.3.7 打印指令	10
2.3.8 指定坐标系指令	10
2.3.9 临时取消碰撞	11
2.4 运动指令	11
2.4.1 关节运动指令	12
2.4.2 直线运动指令	13
2.4.3 圆弧运动指令	14
2.4.4 拖动复现运动指令	15
2.4.5 微段插补指令	15
2.5 运算指令	16
2.5.1 清除指令	16
2.5.2 递增指令	17
2.5.3 递减指令	17
2.5.4 加运算指令	18
2.5.5 减运算指令	18
2.5.6 乘运算指令	19
2.5.7 除运算指令	19
2.5.8 求余运算指令	19
2.5.9 位与运算指令	20
2.5.10 位或运算指令	20
2.5.11 取反运算指令	21
2.5.12 异或运算指令	21

2.5.13	建立用户坐标系	22
2.5.14	计算两点距离	22
2.6	赋值指令	23
2.6.1	变量赋值指令	23
2.6.2	关节位置赋值指令	23
2.6.3	空间位置赋值指令	24
2.6.4	获取当前关节坐标位置	24
2.6.5	获取工具坐标数据	25
2.6.6	设置用户坐标	25
2.6.7	获取用户坐标数据	25
2.6.8	设置 V 变量到工具	26
2.7	输入输出指令	26
2.7.1	数字信号输出指令	26
2.7.2	读取输入信号	27
2.7.3	模拟量信号输出指令	28
2.7.4	获取模拟量输入口电压	28
2.7.5	虚拟信号输出指令	29
2.7.6	读取指定的 M 信号	29
2.7.7	脉冲操作	30
2.8	JBI 特殊指令	30
2.8.1	位姿相乘指令	30
2.8.2	逆解指令	31
2.8.3	正解指令	31
2.8.4	透传初始化	31
2.8.5	透传记录当前位置	32
2.8.6	透传增加目标位置	32
2.8.7	透传结束	32
2.8.8	设置作业文件指令	33
2.8.9	位姿求逆	33
2.8.10	加载作业文件	33
2.8.11	卸载作业文件	34
2.8.12	透传增加目标位置	34
2.9	其他指令	34
2.9.1	保存 P 变量指令	34
2.9.2	打印指令	35
2.9.3	设置负载	35
2.10	脚本指令	36

2.10.1	运行指定脚本文件	36
2.10.2	停止脚本文件	36
2.10.3	重新加载脚本	36
2.10.4	获取脚本状态	37
3	JBI 脚本示例	38
3.1	子程序调用	38
3.2	逻辑门运算	38
3.3	逻辑运算优先级	40
3.4	正逆解计算	40
3.5	坐标系变换计算	41

艾利特协作机器人的 **JBI** 脚本编辑可以采用示教器交互编程方式，也可以采用基于文本的编辑方式。

示教器交互编程方式实质是生成一定格式的脚本程序、无需编程人员记住指令以及参数的类型；基于文本的编辑方式则可以提高编程的效率，比较适合对于 **JBI** 指令比较熟悉的编程人员。

艾利特机器人控制器采用 **Linux** 操作系统，并且自带 **sFTP** 服务端功能，通过 **FTP** 软件（例如 **FileZilla**），用户名:root, 密码:elite2014, 端口号 22, 登录到/rbctrl 文件夹即可查看后缀名为.jbi 的文本格式文件。一些免费的文本编辑器，例如 notepad++，自带 **sFTP** 插件，可以实现远程在线编辑。

第 1 章 示教器交互式编程

1. 将模式开关切换到示教（TEACH）模式，切换到程序列表界面，如图 1-1 所示，新建或打开已有程序。



图 1-1：程序列表界面

2. 点击主菜单中的编辑指令或者子菜单栏的运动/逻辑指令快捷键，选用相应的指令，如图所示。

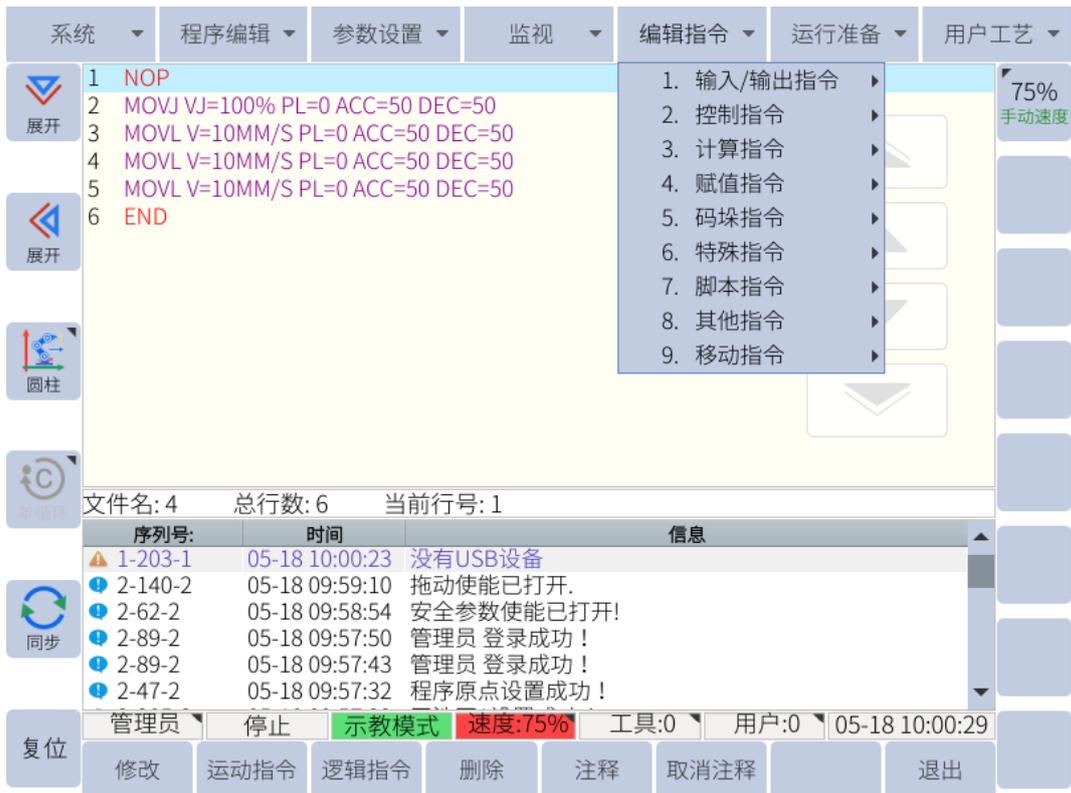


图 1-2：编辑指令



图 1-3：运动/逻辑指令

3. 输入正确的指令参数后，点击“确定”键；点击“取消”键取消指令的输入；对于运动指令的选择和输入，都必须保证伺服处于励磁状态（伺服开关为三档开关，完全松开为

一档，中间为一档，完全按下为一档，其中开关处于中间档代表伺服处于励磁状态)。参数的设置如图 1-4 所示，灰色为可选项，向下的三角符号则表示存在下拉选择。

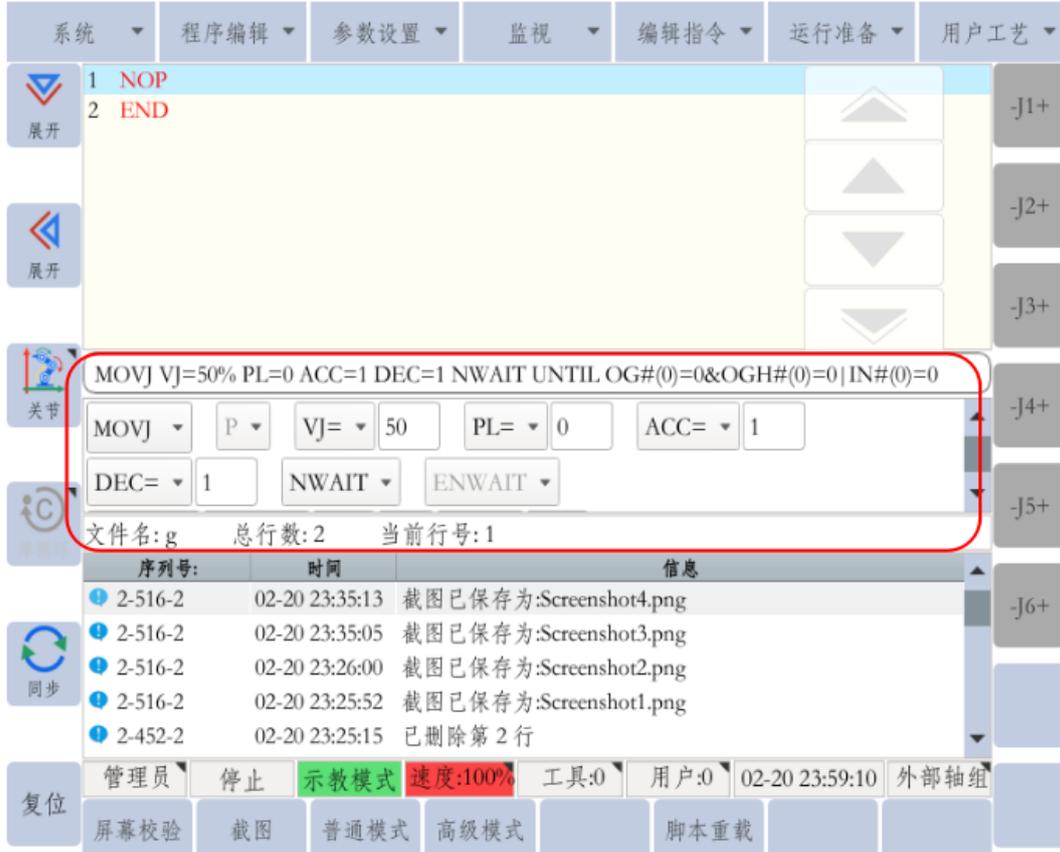


图 1-4：指令参数设置

第 2 章 JBI 脚本编程

艾利特协作机器人使用的脚本语言叫 JBI，由艾利特独立开发。区别于 Lua 语言，JBI 脚本是唯一拥有机器人运动控制指令的语言，也包括变量、逻辑、循环、运算、IO 监控等指令；JBI 脚本能够与软 PLC 以及 Lua 脚本通过 M 寄存器、全局变量等实现快速的数据共享。JBI 脚本存储在一个后缀名为“.jbi”的文本文件中，称之为一个 JOB。

本文约定：

1. 脚本指令包含：脚本和参数，而参数则包含参数名和参数的值，例如：CALL JOB:1 [IF B001=2]，其中“[]”表示这个是可选参数，斜体字为指令程序；
2. 指令说明中的小写字母表示对应类型的名称，例如 job 代表这个参数是一个 JOB，例如名为 1.jbi 的 JOB 为 JOB:1，label 代表一个标签，参数二选一则用“|”连接，例如 JUMP job|label 代表 JUMP 后面既可以是 job 也可以是 label，joint#3 代表第 3 个 joint 类型变量，详见第 2.1 节；
3. {一级菜单}->{二级菜单} 表示示教器上的选项，{} 则表示说明，例如={等于} 表示“=”作用是等于。

2.1 JBI 脚本关键字

JBI 脚本的数据类型有 int{整型}，uint{正整数型}，double{浮点型}，joint{关节位置}，pose{位姿}，bool{布尔型}，string{字符串}，bit{位}。

内置全局变量 (variable) 包括：

B{uint}，正整数型变量，范围 0~2147483647，数量 0~255，例如 B001；

I{int}，整数型变量，范围-32767~32767，数量 0~255，例如 I001；

D{double}，浮点数变量，范围-1e+09~1e+09，数量 0~255，例如 D001；

P{joint}，关节位置变量，J1,J2,J3,J4,J5,J6,0,0，Jx 对应每一个关节的角度值，单位 °，数量 0-255，例如 P001；

V{pose}，位姿变量,x,y,z,Rx,Ry,Rz，x,y,z 为 tcp 的位置、单位 mm，Rx,Ry,Rz 为旋转矢量、单位 rad，例如 V001，旋转矢量的单位矢量为旋转轴，模为旋转角度。

其中 B,I,D,P,V 变量均可以通过示教器 {监视}->{变量} 进行查看和设置。B,I,D,P,V 变量为全局变量，采用变量类型 + 序号的方式 (例如 B001) 的方式进行访问，在 Lua 脚本中也可以进行访问；在各自前面加入 L(local)，则表示局部变量，如图 2-1 所示。例如 LB001，Lua 脚本不能进行访问。所有全局变量均支持掉电保持。

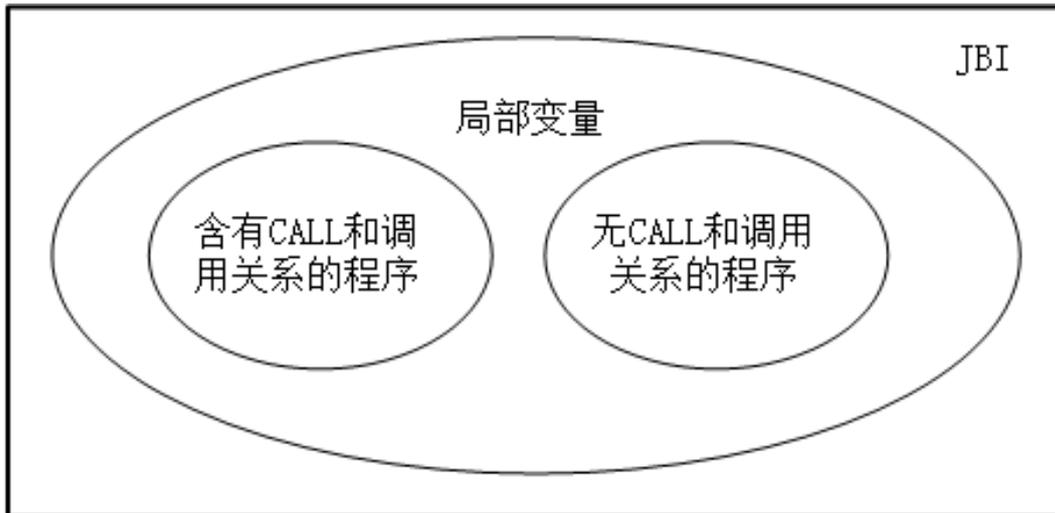


图 2-1：局部变量

M{bit} 虚拟线圈，范围 M000~M1471。单个 M 变量为线圈，多个连续的 M 虚拟线圈则组成寄存器：M#(uint){1 位寄存器}，MGH#(uint){ 4 位寄存器}，MG#(uint){ 8 位寄存器}，MGD#(uint){16 位寄存器}。虚拟输入，范围：M000~M399；虚拟输出，范围：M400~M1023；其中 M400~M527 为系统占用，M528~M799 可由用户改变状态值；M800~M1023 可作为 PLC 的中间继电器；

- 4 位寄存器，最多有 $1024/4=256$ 个，寄存器地址为每连续 4 位线圈的首个线圈地址；
- 8 位寄存器，最多有 $1024/8=128$ 个，寄存器地址为每连续 8 位线圈的首个线圈地址；
- Modbus 是 16 位寄存器，最多有 $1024/16=64$ 个，寄存器地址为连续 16 位线圈的首个线圈地址；

注：用户可以改变状态的只有线圈 M528~M799 任意连续 16 个一组对应的寄存器， $(799-527)/16=17$ ，共 17 个 16 位寄存器。

逻辑运算符：|{or},&{and}, 1=1{true}, 1=0{false}；

比较符：={等于}, <>{不等于}, >{大于}, <{小于}, >={大于等于}, <={小于等于}；

数字量 {switch}:ON{开}, OFF{关}；

程序结构：NOP{程序开始}, END{程序结束}；

坐标系 {frame}：CART{笛卡尔坐标}, JOINT{关节坐标}, USER#(数字){用户坐标系数字：0-7,}; TOOL#(数字){工具坐标系数字：0-7,}

输入输出 {io}：OT#(uint){1 位数字输出口}, OGH#(uint){ 4 位数字输出口}, OG#(uint){8 位数字输出}; IN#(uint){1 位数字输入}, IGH#(uint){ 4 位数字输入}, IG#(uint){8 位数字输入}; AO#(uint){模拟输出}, AI#(uint){模拟输入};

2.2 JBI 脚本格式

JBI 脚本是大小写敏感型，所有的指令及参数均是大写；对缩进没有要求，不同指令需处于不同行。JBI 脚本的常用格式如下，指令均需包围在关键字 NOP 和 END 之间，NOP 之前的数据

是软件自动生成的关节位置数据，对应于使用非变量类型的 MOVE 指令，例如下面代码的第 5、6 行。

```
1 //固定点位的存储（//是注释）
2 C00000=0.0000,-90.0000,0.0000,0.0000,90.0000,0.0000,0.0000,0.0000
3 C00001=0.0000,-90.0000,0.0000,0.0000,90.0000,0.0000,0.0000,0.0000
4 NOP
5 MOVJ VJ=100% PL=0
6 MOVL AV=10.0 MM/S PL=0
7 END
```

2.3 JBI 流程控制语句

2.3.1 IF 指令

IF-ELSEIF-ELSE 语句：

一个 if 必须对应一个 end if，else if 不需要对应 end if；if 嵌套 if，其中的判断语句相同时不会报错。

```
1 IF D000=0&1=1 THEN
2 //添加代码
3 ELSEIF D000=1| 1=0 THEN
4 //添加代码
5 ELSE
6 //添加代码
7 ENDIF
```

2.3.2 WHILE 指令

WHILE-CONTINUE-BREAK 语句：

一个 while 对应一个 end while；单循环模式下程序中只有一组 while 判断语句，不包含 break 且判断条件成立，则程序一直循环。

```
1 WHILE LB000=0 DO
2 //添加代码
3 IF D000=0 THEN
4 BREAK
5 ELSE
6 CONTINUE
```

```

7  ENDIF
8  ENDWHILE
9  //循环模式为单循环时，当LB000=0时开始循环，当D000=0时，while...DO指令后
   //至IF...THEN指令前的代码执行一次；
10 //当LB000=0时开始循环，当D000不等于0时，while...DO指令后至IF...THEN指令
   //前的代码执行多次

```

2.3.3 CALL 和 JUMP 指令

JUMP-LABEL 跳转语句，JUMP job|label [IF true]:

JUMP 和 CALL 均支持程序间的循环嵌套。

```

1  //LABEL后面标签名首字母需为“*”
2  LABEL *p1
3  //添加代码
4  //JUMP后面可以是标签名或者程序名
5  JUMP JOB:1 IF B000=0|B002=1
6  //添加代码
7  JUMP *gy IF B000=1

```

CALL-RET 语句，CALL job|label [IF true], RET[IF true]:

```

1  //LABEL后面标签名首字母需为“*”
2  LABEL *p1
3  //添加代码
4  //CALL后面可以是标签名或者程序名
5  CALL JOB:1 IF B000=0|B002=1
6  //添加代码
7  CALL *gy IF B000=1

```

JUMP 和 CALL 均支持程序间的循环嵌套。

JUMP 和 CALL 都可以调用 JOB；区别在于 CALL 调用的 JOB 带有 RET 语句，返回后继续运行，CALL 最多调用 10 层，所有程序的 CALL 不能超过 128 个（同一名称的算一个）。RET 只能与 CALL 配合使用，JUMP 不能与 RET 配合使用。

JUMP 指令调用文件，运行结束后停止，界面显示跳到的程序页面；JUMP 指令后的指令不会继续运行；CALL 指令调用文件，运行结束后停止，界面显示 call 指令所在的程序界面。

提醒



1. 程序 A 中有 CALL 指令和局部变量，被 CALL 的程序 B 中含有同名的局部变量，程序 B 中的同名的局部变量计算结果不会影响程序 A 中同名的局部变量的值。
2. 程序 C 和程序 D 中没有 CALL 指令，含有同名的局部变量，运行 C 程序后，再运行 D 程序，程序 C 中同名的局部变量计算结果会影响程序 D 中同名的局部变量的值。

2.3.4 暂停指令

```
PAUSE [IF true]
```

功能：暂停指令。机器人执行此指令时程序将立刻暂停

参数：[IF true]：可选的判断条件。

通过逻辑运算符对任意变量或常量进行状态判断，若条件成立 (true) 则程序暂停。多项条件可通过逻辑运算符 `{or}` 和 `&{and}` 进行串联叠加。

示例：

```
PAUSE IF B001=2
//如果全局变量B001的值为2，则程序暂停。
PAUSE IF OT# (003) =0 | D002>5
//如果数字量输出Y003为0或全局变量D002的值大于5，则程序暂停
```

2.3.5 等待指令

```
WAIT bool [T=∞]
```

功能：等待指令，在条件不成立时，机器人执行此指令在此处等待时间 T

参数：bool：可选的等待条件。

条件是任意变量或常量的状态判断，当条件成立时继续运行，机器人状态为运行中，当不满足条件时，则程序等待时间 T。

[T=∞.]：可选的最长等待时间（单位：秒）。不满足条件程序时候等待的时间，若条件满足，则不等待。若不赋值则默认为一直等待。

示例：

```
WAIT M# (600) =1
//
WAIT D001>0.3 T=3
//等待D001大于0.3，条件成立或超时3秒后，继续执行后续指令。
WAIT LI>8 & OGH# (14)T=1.2
//
```

2.3.6 延时指令

```
TIMER T=double
```

功能： 延时指令。机器人执行此指令时程序在此处停留指定的时间，随后继续运行。

参数： T=double： 延时时间，范围在 0~10000，接受整数或小数赋值，单位为秒 (s)；

示例：

```
TIMER T=3  
// 延时3秒钟
```

注意： 与 WAIT 区别为 TIMER 是等待固定时间，无判断条件。

2.3.7 打印指令

```
TPWRITE string
```

功能： 打印指令。机器人执行此指令时将指定内容记录到 log 文档并显示在信息栏内

参数： string： 打印内容。由数字和字母组成的字符串。

示例：

```
TPWRITE Completed  
//打印“Completed”字符串到log窗口。
```

注意： 汉字最多输入 42 个。

2.3.8 指定坐标系指令

```
CCOOD frame
```

功能：指定坐标系指令。JBI 脚本程序中使用此指令时，该指令所在行之后的运算将在指定坐标系下进行。

参数：frame：坐标系类型。

CART= 笛卡尔坐标，JOINT= 关节坐标，USER#()= 用户坐标系数字 (范围 0-7)，
 TOOL#()= 工具坐标系序号 (范围 0-7)

示例：

```
SET P001(2) 60.00
//将P001位姿的3轴角度设为60.00度
CCOOD CART
//指定为直角坐标系
SET P001(2) 100.00
//将P001位姿在基座坐标系下Z方向的偏移距离设置为100.00mm。
```

注意：示教的固定点在不同坐标系下实际位置不发生改变，此指令影响的只是在该指定坐标系指令行后到下个指定坐标系指令行前的内容中相关变量的计算和赋值。
 指定工具坐标系适用于 2.13.0 及以上版本。

2.3.9 临时取消碰撞

```
MCWAIT variable#1 variable#2 T=doulue S=int
```

功能：暂时关闭碰撞报警，直到退出条件满足或程序停止

参数：variable#1: 变量 #1，可选用整数，浮点数，OT#(), OG#(), OGH#(), IN#(), IG#(), IGH#(), M#(), MG#(), MGH#(), B, LB, I, LI, D, LD, OT#(*), OG#(*), OGH#(*), IN#(*), IG#(*), IGH#(*), B[*], LB[*], I[*], LI[*], D[*], LD[*]

variable#2: 变量 #2，可选用整数，浮点数，OT#(), OG#(), OGH#(), IN#(), IG#(), IGH#(), M#(), MG#(), MGH#(), B, LB, I, LI, D, LD, OT#(*), OG#(*), OGH#(*), IN#(*), IG#(*), IGH#(*), B[*], LB[*], I[*], LI[*], D[*], LD[*]

T: 时间

S: 灵敏度

示例：

```
MCWAIT *IN#() 1=ON
```

注意：本命令适用于 2.15.0 及以上版本。

2.4 运动指令

关于艾利特系统移动指令以及移动指令速度特别说明：

1. 手动模式下移动示教的点位，艾利特系统对移动指令的处理都是按 MOVJ 插补来处理的；
2. 手动模式下移动示教的点位，机器人移动的速度只跟手动模式下整体速度有关；
3. 自动模式下，当速度类型是 VJ、V、VR 时，机器人自动运行速度 = (速度类型的数据值) * (自动模式整体速度百分比)；
4. 自动模式下，当速度类型是 AV、AVR 时，机器人自动运行速度 = 速度类型的数据值

2.4.1 关节运动指令

MOVJ [joint] VJ=int PL=int [ACC=int] [DEC=int] [UNTIL bool]

功能： 关节运动指令，机器人执行该指令时将以关节插补方式移动

普通模式 参数： int./B：速度参数，范围是 1%~100%，速度的值可以直接输入数值（百分比）或者使用 B 变量设置

PL=int.：轨迹交融等级，数值范围 0~7。

数值越大，轨迹融合程度越高。未选时默认为 PL=0。

高级模式 参数： [joint]：关节角度变量。当有该可选参数时表示使用指定的 P 变量，没有时表示使用示教的固定点位；

[ACC=int]：加速度百分比，范围为 0~100；

[DEC=int]：减速度百分比，范围为 0~100；

[UNTIL bool]：该条运动指令执行的条件，对某个变量或常量进行判断。

条件成立时，此点位不运行，不成立则运行此点位。多项条件可通过逻辑运算符 |{or} 和 &{and} 进行串联叠加

示例：

```
MOVJ VJ=100% PL=0
//以关节插补、100%的速度、轨迹融合等级为0运动到示教的固定点位
MOVJ P000 VJ=100% PL=0
//以关节插补、100%的速度、轨迹融合等级为0运动到P000点位
SET B000 30
MOVJ VJ=B000 PL=7
//以关节插补、B000的速度、轨迹融合等级为7运动到示教的固定点位
MOVJ P001VJ=60% ACC=75% DEC=40% UNTIL M#(442)=ON
//以60%的速度，75%的加速度，40%的减速度运动到P001点直到M#(442)
//干涉区信号被触发，则运动停止
MOVJ P002 VJ=100% PL=0
//当 M#(442)=ON满足时结束P001点的移动，以100%的速度运动到P002
//点
```

2.4.2 直线运动指令

MOVL [joint] V=int PL=int [ACC=int] [DEC=int] [UNTIL bool]

功能： 直线运动指令，机器人执行该指令时将以直线插补方式移动

普通模式 V=int： 速度参数，可设置为 V, VR, AV, AVR 其中一种。

参数： V= 直线速度，范围 1~3000MM/S；

VR= 旋转角速度： 1~300°/S；

AV= 为绝对直线速度： 1~3000MM/S；

AVR= 为绝对旋转角速度： 1~300°/S；

PL=int.： 轨迹交融等级，数值范围 0~7。

数值越大，轨迹融合程度越高。未选时默认为 PL=0。

高级模式 [joint]： 关节角度变量。当有该可选参数时表示使用指定的 P 变量，没有时表示使用示教的固定点位；

[ACC=int]： 加速度百分比，范围为 0~100；

[DEC=int]： 减速度百分比，范围为 0~100；

[UNTIL bool]： 该条运动指令执行的条件，对某个变量或常量进行判断。

条件成立则运动立刻暂停，不成立则继续运动。多项条件可通过逻辑运算符 `{or}` 和 `{and}` 进行串联叠加

示例：

```

MOVL AV=400 MM/S PL=7
//以直线轨迹插补方式、400.0MM/S的速度，等级为7的交融程度运动到
    固定示教点位置
MOVL P001 AV=400.0 MM/S PL=0
//以直线轨迹插补方式、400.0MM/S的速度，运动到P001点。
SET I000 100
MOVL AV=I000
//以直线插补、速度I000 MM/S 运动到示教的固定点位
MOVL AV=100 MM/S UNTIL IN#(1)=1
//以直线插补、速度100MM/S运动到示教的固定点位直到外部数字量输入
    X1有效，则运动停止
SET I000 100
MOVL P000 AV=I000 PL=7 UNTIL IN#(1)=1|M#(528)=1&D000
    =10.145
//以直线插补、速度I000 MM/S、轨迹平衡等级为7运动到P000点位直到
    外部数字量输入X1有效或虚拟输出M528有效并且变量D000=10.145
    时，则运动停止
    
```

2.4.3 圆弧运动指令

MOV C [joint] V=int PL=int FPT [ACC=int] [DEC=int] [UNTIL bool]

功能： 圆弧运动指令；机器人执行该指令时将通过圆弧插补示教的 3 个点完成一段圆弧轨迹移动。

注： 此指令必须以 3 个为一组应用在程序中，否则圆弧轨迹无法实现，系统报错。此外，PLAY 模式下单步运行该指令时，机器人以关节插补的方式运动。

普通模式 V=int：速度参数，可设置为 V, VR, AV, AVR 其中一种。

参数： V= 直线速度，范围 1~3000MM/S；

VR= 旋转角速度：1~300°/S；

AV= 为绝对直线速度：1~3000MM/S；

AVR= 为绝对旋转角速度：1~300°/S；

PL=int：轨迹交融等级，数值范围 0~7。

数值越大，轨迹融合程度越高。未选时默认为 PL=0。

FPT：确定一段圆弧的终点

高级模式 [joint]：关节角度变量。当有该可选参数时表示使用指定的 P 变量，没有时表示使用示教的固定点位；

[ACC=int]：加速度百分比，范围为 0~100；

[DEC=int]：减速度百分比，范围为 0~100；

[UNTIL bool]：条件成立时，此点位不运行，不成立则运行此点位

示例：

```

MOV C V=500 MM/S PL=1
MOV C V=500 MM/S PL=1
MOV C V=500 MM/S PL=1
//以圆弧插补、速度V=500MM/S、轨迹平衡等级为1运动示教的三个固定
    点位，
MOV C P000 AV=100 MM/S
MOV C P001 AV=100 MM/S
MOV C P002 AV=100 MM/S
//以圆弧插补、速度AV=100MM/S依次运动到P000、P001、P002点位
SET I000 100
MOV C P000 AV=I000 PL=7
MOV C P001 AV=I000 PL=7
MOV C P002 AV=I000 PL=7
//以圆弧插补、速度AV=I000 MM/S、、轨迹平衡等级为7依次运动到
    P000、P001、P002点位
    
```

2.4.4 拖动复现运动指令

```
MOVDRAG VJ=int DRAG=int JOB: dragfile
```

功能：拖动复现运动指令，机器人执行该指令时复现指定拖拽文件中的运动轨迹。

参数：VJ=int：以关节插补的方式从当前位置运动到拖动轨迹起点的速度百分比，范围为0%~100%；

DRAGV=int：机器人运行拖动轨迹时运动速度百分比，范围为0%~100%；

JOB: dragfile：拖动轨迹文件，是一个特殊的jbi文件，无法通过示教器直接打开编辑；

示例：

```
MOVDRAG VJ=60% DRAGV=25% JOB: run
```

```
//在当前点，以60%的速度运行到文件run.jbi中记轨迹的起始点，再以25%的速度复现该轨迹。
```

2.4.5 微段插补指令

```
MOVML VJ = int type#[addr] JOB: filename
```

功能：微段插补指令，用于拖动轨迹的复现

参数：VJ =int：关节运动速度，范围是 1%~100%；

type#[addr]：关节运动到离线文件的起始点后等待该信号有效

type：可选择的信号类型，IN 数字量输入或 M 虚拟量；

addr：地址，int 型

JOB: filename：离线文件文件名，只能识别文件头为 dragfile、trajfile、pathfile 或 loadfile 的 JBI 文件。

示例：NOP

```

LOADML JOB:dragfile1
MOVML VJ=100% IN#(4) JOB:dragfile1
UNLOADML JOB:dragfile1
END

//离线文件格式：
///pathfile //文件头，指示机器人这是个路径文件
version2.15.x //2.15.0及以上版本支持用新格式（添加IO,坐标系）
interval [2]ms //采样时间间隔，支持大于1ms的整数
//第一个点的关节角度或者第一个位姿的逆解参考值，单位：度
refJointPos [-119.95,-167.84,-55.02,-175.47,87.41,-24.31]
length [3729] //后面数据长度（数据行数）
postype [joint] //数据类型关节角还是位姿 joint/pose
outputNumber [23,25] #输出的IO的序号，最多支持定义6个，至少需要写
    一个IO。若不启用，则不使用output命令。
refFrame [0,0,0,0,0,0] #数据基于的坐标系，如果是用基坐标系，则必
    须全为0.
[-119.952,-167.839,-55.0197,-175.474,87.4069,-24.3066]
[-119.902,-167.789,-55.0697,-175.424,87.4569,-24.3566]
output [0,1] #对应设置outputNumber定义序号的IO状态，最多支持6
    个，若不设置状态，上述的IO不使用。
[-119.852,-167.739,-55.1197,-175.374,87.5069,-24.4066]
[-119.802,-167.689,-55.1697,-175.324,87.5569,-24.4066]
....
[-120,-170,-50.2404,-183.597,74.8063,-38.711]
[-120,-170,-50.3607,-183.597,74.8063,-38.6572]
    
```

2.5 运算指令

2.5.1 清除指令

```
CLEAR variable int
```

功能：清除变量值并置 0 的指令

参数：variable: 操作对象变量的起始地址, 可选用 B, I, D, P, LB, LI, LD, LP, B[*], LB[*], I[*], LI[*], D[*], LD[*], P[*], LP[*] 其中一种, 详见第 2.1 节。

int: 变量的个数, 可为整数或 ALL, ALL 表示变量起始地址以后该类型的所有变量

示例:

```
CLEAR B000 3
//清除变量B000, B001, B002的值, 全部置零。
SET P020(4) 0
//使用SET指令清除P变量轴数据
```

注意：P, V 变量时不能清除某一轴数据；变量个数应大于 0, 负数不报错

2.5.2 递增指令

```
INC variable
```

功能：变量值递增（加 1）指令

参数：variable: 操作对象变量, 可选用 B, I, D, LB, LI, LD, B[*], LB[*], I[*], LI[*], D[*], LD[*] 其中一种, 详见第 2.1 节。

示例:

```
INC LB000
//LB000变量的值增1
```

2.5.3 递减指令

```
DEC variable
```

功能：变量值递减（减 1）指令

参数：variable: 操作对象变量, 可选用 B, I, D, LB, LI, LD, B[*], LB[*], I[*], LI[*], D[*], LD[*] 其中一种, 详见第 2.1 节。

示例:

```
DEC LI002
//LI002变量的值减1
DEC B[B5]
//若B005=7, 则DEC B[B5]等同于DEC B007
```

2.5.4 加运算指令

```
ADD variable#1 variable#2
```

功能：加运算指令，变量 #1 与变量 #2 相加，相加后的结果存入变量 #1。

参数：variable#1：变量 #1, 可选用 B, I, D, P, LB, LI, LD, LP, B[*], LB[*], I[*], LI[*], D[*], LD[*], P[*], LP[*] 其中一种；详见第 2.1 节。

variable#2：变量 #2, 可选用整数, 小数, B, I, D, P, LB, LI, LD, LP, B[*], LB[*], I[*], LI[*], D[*], LD[*], P[*], LP[*] 其中一种；详见第 2.1 节。

注意：整数类型变量与浮点数类型变量进行此运算时，若结果保存在整数类型变量中，则只保存计算结果的整数部分（不进行四舍五入，而是将小数部分完全舍弃）。

示例：

```
ADD B001 5
//把整数5与变量B001的值相加后，结果存入变量B001中；
ADD D001 I002
//把变量D001和变量I002的值相加后，结果存入变量D001中；
ADD P001(2) 25
//在变量P001的joint_3的值上增加25度
```

2.5.5 减运算指令

```
SUB variable#1 variable#2
```

功能：减运算指令，变量 #1 减去变量 #2，结果存入变量 #1。

参数：variable#1：变量 #1, 可选用 B, I, D, P, LB, LI, LD, LP, B[*], LB[*], I[*], LI[*], D[*], LD[*], P[*], LP[*] 其中一种；详见第 2.1 节。

variable#2：变量 #2, 可选用整数, 小数, B, I, D, P, LB, LI, LD, LP, B[*], LB[*], I[*], LI[*], D[*], LD[*], P[*], LP[*] 其中一种；详见第 2.1 节。

注意：整数类型变量与浮点数类型变量进行此运算时，若结果保存在整数类型变量中，则只保存计算结果的整数部分（不进行四舍五入，而是将小数部分完全舍弃）。

示例：

```
SUB B001 5.8
//变量B001的值和小数5.8相减后，结果存入变量B001中；
SUB I003 LD004
//变量I003的值与LD004的值相减后，取整数部分保存至I003中
SUB P006(2) 90
//在变量P006的joint_3的值上减去90°
```

注意：若计算结果小于 0 且需要保存至 B 变量，则保存至 B 变量的值为 0

2.5.6 乘运算指令

```
MUL variable#1 variable#2
```

功能：乘运算指令，变量 #1 与变量 #2 相乘，把相乘后的结果存入变量 #1 中。

参数：variable#1：变量 #1, 可选用 B, I, D, P, LB, LI, LD, LP, B[*], LB[*], I[*], LI[*], D[*], LD[*], P[*], LP[*] 其中一种；详见第 2.1 节。

variable#2：变量 #2, 可选用整数, 小数, B, I, D, P, LB, LI, LD, LP, B[*], LB[*], I[*], LI[*], D[*], LD[*], P[*], LP[*] 其中一种；详见第 2.1 节。

注意：整数类型变量与浮点数类型变量进行此运算时，若结果保存在整数类型变量中，则只保存计算结果的整数部分（不进行四舍五入，而是将小数部分完全舍弃）。

示例：

```
MUL B001 I002
//变量B001的值与变量I002的值相乘后，结果存入变量B001中
```

2.5.7 除运算指令

```
DIV variable#1 variable#2
```

功能：除运算指令，变量 #1 除以变量 #2，把相除后的结果存入变量 #1。

参数：variable#1：变量 #1, 可选用 B, I, D, P, LB, LI, LD, LP, B[*], LB[*], I[*], LI[*], D[*], LD[*], P[*], LP[*] 其中一种；详见第 2.1 节。

variable#2：变量 #2, 可选用整数, 小数, B, I, D, P, LB, LI, LD, LP, B[*], LB[*], I[*], LI[*], D[*], LD[*], P[*], LP[*] 其中一种；详见第 2.1 节。

注意：整数类型变量与浮点数类型变量进行此运算时，若结果保存在整数类型变量中，则只保存计算结果的整数部分（不进行四舍五入，而是将小数部分完全舍弃）。

示例：

```
DIV B001 50.1
//若B001=100，B001除整数50.1后，结果取整为1，不四舍五入，最后
B001的值为1
```

注意：当除运算指令中除数设置为 0 时，机器人报错除零错误，被除的变量依旧变为 0

2.5.8 求余运算指令

```
MOD variable#1 variable#2
```

功能：求余运算指令，变量 #1 除以变量 #2，把相除后的余数存入变量 #1 中。

参数：variable#1：变量 #1，可选用 B, I, D, P, LB, LI, LD, LP, B[*], LB[*], I[*], LI[*], D[*], LD[*], P[*], LP[*] 其中一种；详见第 2.1 节。

variable#2：变量 #2，可选用整数, 小数, B, I, D, P, LB, LI, LD, LP, B[*], LB[*], I[*], LI[*], D[*], LD[*], P[*], LP[*] 其中一种；详见第 2.1 节。

注意：此运算不支持小数及浮点数，这两种类型的变量或常数参与运算时，系统只取整数部分（不是四舍五入）进行运算。系统不报错。

示例：

```
MOD B001 50.98
//若B001=51，B001除50.98后结果取整，小数点后有数字都进1，B001
  的值为1
```

2.5.9 位与运算指令

```
AND variable#1 variable#2
```

功能：位与运算指令，取变量 #1 与变量 #2 的值转换为二进制之后进行与逻辑运算，再将结果转换成十进制整数存入变量 #1。

参数：variable#1：变量 #1，可选用 B, I, LB, LI, B[*], LB[*], I[*], LI[*] 其中一种；详见第 2.1 节。

variable#2：变量 #2，可选用整数, 小数, B, I, LB, LI, B[*], LB[*], I[*], LI[*] 其中一种；详见第 2.1 节。

注意：若参与运算的变量值为负数或小数，系统将取该变量整数部分的绝对值，并转换成二进制后带入运算。

示例：

```
AND B001 B002
// B001的值与B002的值的逻辑与之后，结果存入变量B001
```

2.5.10 位或运算指令

```
OR variable#1 variable#2
```

功能：位或运算指令。取变量 #1 与变量 #2 的值转换为二进制之后进行或逻辑运算，将结果转换成十进制整数存入变量 #1。

参数：variable#1：变量 #1，可选用 B, I, LB, LI, B[*], LB[*], I[*], LI[*] 其中一种；详见第 2.1 节。

variable#2：变量 #2，可选用整数, 小数, B, I, LB, LI, B[*], LB[*], I[*], LI[*] 其中一种；详见第 2.1 节。

注意：若参与运算的变量值为负数或小数，系统将取该变量整数部分的绝对值，并转换成二进制后带入运算。

示例：

```
OR B001 B002
// B001 的值与 B002 的值的逻辑或之后，结果存入变量 B001
```

2.5.11 取反运算指令

```
NOT variable#1 variable#2
```

功能：取反运算指令。将变量 #2 的值转换成二进制之后进行非逻辑运算，再将结果转换成十进制整数存入变量 #1。

参数：variable#1：变量 #1，可选用 B, I, LB, LI, B[*], LB[*], I[*], LI[*] 其中一种；详见第 2.1 节。

variable#2：变量 #2，可选用整数, 小数, B, I, LB, LI, B[*], LB[*], I[*], LI[*] 其中一种；详见第 2.1 节。

注意：若参与运算的变量值为负数或小数，系统将取该变量整数部分的绝对值，并转换成二进制后带入运算。

示例：

```
NOT B001 B002
//取B001的值与B002的值的非逻辑之后，结果存入变量B001。若B002
  的值小于等于225，则将B002的值转换成八位二进制数之后进行取
  反运算，并转换为十进制数赋值给B001。
//若B002的值大于255，则转换成多位二进制数（超过八位）进行取反
  计算，并转换为十进制数赋值给B001。
```

2.5.12 异或运算指令

```
XOR variable#1 variable#2
```

功能：异或运算指令。取变量 #1 与变量 #2 的值转换为二进制之后进行异或逻辑运算，将结果转换成十进制整数存入变量 #1。

参数：variable#1：变量 #1，可选用 B, I, LB, LI, B[*], LB[*], I[*], LI[*] 其中一种；详见第 2.1 节。

variable#2：变量 #2，可选用整数, 小数, B, I, LB, LI, B[*], LB[*], I[*], LI[*] 其中一种；详见第 2.1 节。

注意：若参与运算的变量值为负数或小数，系统将取该变量整数部分的绝对值，并转换成二进制后带入运算。

示例：

```
XOR B001 B002
//取B001的值与B002的值的异或逻辑之后，结果存入B001
```

2.5.13 建立用户坐标系

```
MFRAME USER# ( ) joint#1 joint#2 joint#3
```

功能：建立用户坐标系指令。通过已知给定 3 个点的位置数据作为定义点，创建用户坐标。

参数：UESR#(): 自定义用户坐标，范围为 0~7，共 8 个；

joint#1：关节位置变量 P，用户坐标系的原点，用以确定原点，范围为 0~255。

joint#2：关节位置变量 P，用户坐标系 X 轴正方向上的点，用以确定 X 轴正方向，范围为 0~255。

joint#3：关节位置变量 P，用户坐标系平面上的点，用以确定 Y 轴正方向，范围为 0~255。

注：同样的 P001、P002、P003 三个点，其顺序不同，通过 MFRAME 计算的用户坐标系不同。

当前软件版本支持单轴数据（例如，P001(2)）进行计算，系统不报错但计算结果无效。

示例：

```
MFRAME USER# (1) P001 P002 P003
//以P001为原点，P001指向P002为X轴正方向，P003为XY界面第一象限上除P001和P002以外任意一点。
```

2.5.14 计算两点距离

```
DIST D#(int) joint#1 joint#2
```

功能： 计算两点之间距离的指令

参数： D#(): 保存计算结果的变量, 可选用 D, LD, D[*], LD[*] 其中一种, 详见第 2.1 节。
 joint#1, joint#2: 用于计算的位置变量, 可选用 P, LP, P[*], LP[*] 其中一种, 详见第 2.1 节。

当前软件版本支持单轴数据 (例如, P001(2)) 进行计算, 系统不报错但计算结果无效。计算的结果依旧会写入对应的变量中。

示例:

```
DIST D000 P000 LP001
//取P000和LP001两点之间的距离结果存入D000
```

2.6 赋值指令

2.6.1 变量赋值指令

```
SET variable#1 variable#2
```

功能： 变量赋值指令, 把变量 #2 的值赋值给变量 #1, 保存新的变量 #1

参数： variable#1 : 变量 #1, 可选用 B, I, D, P, LB, LI, LD, LP, B[*], LB[*], I[*], LI[*], D[*], LD[*], P[*], LP[*] 其中一种; 详见第 2.1 节。

variable#2 : 变量 #2, 可选用整数, 小数, B, I, D, P, LB, LI, LD, LP, B[*], LB[*], I[*], LI[*], D[*], LD[*], P[*], LP[*] 其中一种; 详见第 2.1 节。

注意： 将小数或浮点数类型变量赋值给整数类型变量时, 则只取整数部分 (不进行四舍五入, 而是将小数部分完全舍弃) 进行赋值。将负数赋值给 B 变量时, 系统取零。此外, 不支持 P 变量以数组形式赋值, 必须细分到单元素进行赋值

示例:

```
SET D001 8.88
//设D001的值为8.88
SET B001 6
//设置B001的值为6
SET B003 D001
//把D001的值取整, 不四舍五入, 赋值给B003
SET P020(4) 100
//将P020位置5轴的旋转角度设置为100度
```

2.6.2 关节位置赋值指令

```
SETJOINT joint double, double, double, double, double, double
,0.0000,0.0000
```

功能： 关节位置赋值指令，把机器人当前 8 个轴 (6 个关节) 的角度值以关节坐标值的形式保存在指定的位置变量中

参数： joint: 关节位置变量 P, 范围 0~255;
 double, double, double, double, double, double, 0.0000, 0.0000: 关节坐标值 J1, J2, J3, J4, J5, J6, J7, J8, 由 8 个轴的角度值组成 (6 个轴, 加 2 个附加轴 J7, J8, 默认为 0), 不可手动修改, 自动根据当前机器人位置生成。

示例：

```
SETJOINT P002 1.1111,2.2222,3.3333,4.4444,
5.5555,6.6666,0.0000,0.0000
//把机器人当前8个关节的坐标角度值保存在指定的P002变量中。可通过“监视”栏点击查看P002的值。
//注：P关节位置型变量保留小数点后面3位，按四舍五入计算。
```

注意： 必须在 servo on 状态时才能插入；set 时必须在 P 变量打开状态下，未打开时关节赋值无效。

2.6.3 空间位置赋值指令

```
SETPOSE pose double,double, double, double, double, double
```

功能： 空间位置赋值指令，把机器人当前位姿以空间坐标值的形式保存在指定的位置变量中

参数： pose: 空间位置变量 V, 范围 0~255;
 double,double, double, double, double, double: 基座坐标系下, 空间坐标值 x,y,z,rx,ry,rz, 由六个维度的偏移和旋转值组成。不可手动修改, 自动根据当前机器人位置生成。

示例：

```
SETJOINT V002 10.00,20.00,30.00,40.00,50.00,60.00
//把机器人当前的空间坐标值保存在指定的V002变量中。
注：V空间位置型变量保留小数点后面2位，按四舍五入计算。
```

2.6.4 获取当前关节坐标位置

```
GETPOS joint
```

功能： 获取当前关节坐标位置指令，把当前关节坐标 J1,J2,J3,J4,J5,J6,J7,J8 赋值到 P 变量中

参数： joint: 关节位置变量 P, 范围为 0~255。

示例：

```
GETPOS P000
//为获取当前关节坐标J1,J2,J3,J4,J5,J6,J7,J8，并赋值到P000变量中
```

2.6.5 获取工具坐标数据

```
GETTOOLFRAME pose TOOL#()
```

功能：获取工具坐标数据到 V 变量中

注：此处获取的工具坐标数据非当前工具坐标位姿，而是工具坐标相应的校验数据，使用前需将程序中用于存储数据的 V 变量打开。

参数：pose：空间位置变量 V，范围 0~255；

TOOL#()：用户指定要获取的工具坐标号，范围 0~7

示例：

```
GETTOOLFRAME V006 TOOL#(0)
```

2.6.6 设置用户坐标

```
SETUSERFRAME USER#() pose
```

功能：设置用户坐标到 V 变量中

参数：USER# ()：用户要指定设置的坐标号，范围 0~7

pose：空间位置变量 V，范围 0~255

示例：

```
SETUSERFRAME USER#(1) V001
```

2.6.7 获取用户坐标数据

```
GETUSERFRAME pose USER#()
```

功能：获取用户坐标数据到 V 变量中

注：此处获取的用户坐标数据非当前用户坐标位姿，而是用户坐标相应的校验数据

参数：pose：空间位置变量 V，范围 0~255；

USER#()：用户指定要获取的用户坐标号，范围 0~7

示例：

```
GETUSERFRAME V009 USER#(0)
```

2.6.8 设置 V 变量到工具

```
SETTOOLFRAME TOOL#() V#()
```

功能：将 V 变量的值设置到工具中，V 变量的值设置为工具坐标系的 X，Y，Z，Rx，Ry，Rz，表示工具的位置和姿态。

参数：TOOL#()：选择工具号；

V#()：全局变量

示例：

```
SETTOOLFRAME TOOL(0) V000
```

2.7 输入输出指令

2.7.1 数字信号输出指令

```
DOUT OT# () switch|int|variable
```

功能：数字信号输出指令，将指定状态赋值给特定的 Y 信号。

参数：OT#(): 数字量输出信号，对应监视列表里 Y 变量。可选用 OT#(), OGH#(), OG#() 三者之一；OT#() 取末位，OGH#() 取末四位，OG#() 取末八位。范围为 Y0~Y063；switch|int|variable: 任意信号状态, 支持的类型有: 整数, B, I, D, LB, LI, LD, B[*], LB[*], I[*], LI[*], D[*], LD[*], ON= 有效, OFF= 无效
 当前软件版本支持对数字输出信号进行小数或浮点数赋值，系统不报错，赋值逻辑是，将输入的值进行四舍五入后，转换为二进制，再逐位赋值给数字输出信号；对数字信号进行负数赋值时，系统不报错，赋值逻辑是，取负数的绝对值（小数，浮点数同样进行四舍五入），再进行取反和对末位加一的操作，最后逐位赋值给数字输出信号；OT#() 取末位，OGH#() 取末四位，OG#() 取末八位。

示例：

```

DOUT OT# (1) ON
//把数字输出Y001置为ON（有效）输出。
DOUT OT# (2) 9.5
//将9.5四舍五入并转换成二进制数字即10=01010000，取末位赋值给
    OT#(2)即Y002=0
DOUT OGH#(1) 14
//取14的值转换成二进制数即01110000，取末四位赋值给OGH#(1)即
    Y004=0，Y005=1，Y006=1，Y007=1
DOUT OGH#(1) -9
//取-9的值转换成二进制数即11100000，取末四位赋值给OGH#(1)即
    Y004=1，Y005=1，Y006=1，Y007=0
DOUT OG# (2) -67.44
//将-67.44四舍五入并转换成二进制数字即-67=00111101，取末八位
    赋值给OG#(2)即Y016=1，Y017=0，Y018 -Y021=1，Y022=0
    ，Y023=1
    
```

2.7.2 读取输入信号

DIN variable IN# ()

功能：读取指定的输入信号（X 变量）并将其从二进制数字转换为十进制整数后赋值给相应变量

参数：variable: 支持全局和局部变量 B, I, D, LB, LI, LD, B[*], LB[*], I[*], LI[*], D[*], LD[*]; 详见第 2.1 节。

IN#(): 数字量输入信号，对应监视列表里 X 变量，可选用 OT#(),OG#(), OGH#(), IN#(), IG#(), IGH#() 六者之一，范围 000~063。详见第 2.1 节。

示例：

```
DIN B001 IN# (001)
//将数字输入X001的值(0或1)读取到变量B001中。
DIN I002 IGH#(002)
//将数字输入信号X008~X011的信号逐个读取后，转换为十进制的整数并赋值给变量I002.例如X008~X011=1010，则I002=5
DIN D003 IG#(003)
//将数字输入X024~X031的信号逐个读取后，转换为十进制的整数并赋值给变量D003.例如M024~M031=11010100，则D003=43
```

2.7.3 模拟量信号输出指令

```
AOUT AO# () double
```

功能：模拟量信号输出指令，将指定数值赋值给特定的 AO 信号

参数：AO#(): 模拟量输出信号，详见第 2.1 节。

double: 指定输出的电压值；范围为-10~10

注意：AO#(5) 时，double 范围为 0~10

示例：

```
AOUT AO# (1) 6.6
//模拟输出A0001的输出电压为6.6V
```

2.7.4 获取模拟量输入电压

```
AIN D001 AI# ()
```

功能：获取模拟量输入电压数值并赋值给相应变量

参数：D001: 支持全局及局部变量 D, LD, D[*], LD[*], 详见第 2.1 节。

AI#(): 指定读取的模拟量输入接口，范围 1~3；详见第 2.1 节。

示例：

```
AIN D004 AI# (1)
//获取模拟量输入AI001的电压到D004
```

2.7.5 虚拟信号输出指令

```
MOUT M# () switch|int|variable
```

功能：虚拟信号输出指令，将指定状态赋值给特定的 M 信号。

参数：[M#()]: 虚拟信号 M, 范围为 M528~M1471, 可选用 M#()(范围 528~1471), MG#()(范围 66~183), MGH#()(范围 132~367), MGD#()(范围 300~447) 四者之一；详见第 2.1 节。

switch|int|variable：任意信号状态, 支持的类型有：整数, B, I, D, LB, LI, LD, B[*], LB[*], I[*], LI[*], D[*], LD[*], ON= 有效, OFF= 无效

当前软件版本支持对 M 信号进行小数或浮点数赋值, 系统不报错。赋值逻辑是, 输入的值进行四舍五入后, 转换为二进制, 再逐位赋值给 M 信号；对 M 信号进行负数赋值时, 系统不报错, 赋值逻辑是, 取负数的绝对值 (小数, 浮点数同样进行四舍五入), 再进行取反和对末位加一的操作, 最后逐位赋值给数字输出信号；M#() 取末位, MGH#() 取末四位, MG#() 取末八位。

示例：

```
MOUT M# (528) ON
//把虚拟输出M528置为ON (有效) 输出。
MOUT M# (600) 11
//11转换成二进制数字即11010000, 取末位赋值给M# (600) 即M600=1
MOUT MGH#(135) -28
// -28转换成二进制数字即-28=00100111, 取末四位赋值给MGH#(135)
    即M540=0, M541=0, M542=1, M543=0
MOUT MG# (68) 135.59
//将135.59四舍五入并转换成二进制数字即136=00010001, 取末八位
    赋值给MG#(68)即M544-M546=0, M547=1, M548-M550=0, M551=1
```

2.7.6 读取指定的 M 信号

```
MIN variable M# ()
```

功能： 读取指定的 M 信号并将其从二进制数字转换为十进制整数后赋值给相应变量

参数： variable: 支持全局和局部变量 B, I, D, LB, LI, LD, B[*], LB[*], I[*], LI[*], D[*], LD[*]; 详见第 2.1 节。

M#(): 虚拟信号 M, 可选用 M#()(范围 0~1535), MG#()(范围 0~191), MGH#()(范围 0~67) 三者之一; 详见第 2.1 节。

示例:

```

MIN B001 M# (399)
//将虚拟输入M399的值(0或1)读取到变量B001中。
MIN I002 MGH#(171)
//将虚拟输入信号M684~M687的信号逐个读取后,转换为十进制的整数
并赋值给变量I002.例如M684~M687=0110,则I002=6
MIN D003 MG#(56)
//将虚拟输入信号M448~M455的信号逐个读取后,转换为十进制的整数
并赋值给变量D003.例如M684~M687=01011010,则D003=90
    
```

2.7.7 脉冲操作

```
PULSE OT# () T=double
```

功能： 对特定信号进行脉冲操作

参数： OT#(): 数字量输出信号, 对应监视列表里 Y 变量。可选用 OT#(), OG#(), OGH#() 三者之一; 参数为 OG# () 时八位均置 1, 参数为 OGH# () 时四位均置 1。范围为 Y0~Y063;

T=double: 脉冲间隔时间 (单位: 秒), 接受浮点数赋值

示例:

```

PULSE OT#(1) T=0.3
//把数字输出Y001置为ON(有效),0.3秒以后置OFF
    
```

2.8 JBI 特殊指令

2.8.1 位姿相乘指令

```
POSEMUL pose#1 pose#2
```

功能：位姿相乘指令，pose1 作为旋转中心，pose2 作为偏移的位姿矩阵，对 pose1 进行位姿偏移。

参数：pose#1：空间位置变量 V，范围为 0~255。

pose#2：空间位置变量 V，范围为 0~255。

示例：

```
POSEMUL V001 V002
//将V001按照V002的偏移量在空间中移动相应的方向与距离
```

2.8.2 逆解指令

```
POSETOJOINT pose#1 joint#1
```

功能：逆解指令，把空间坐标转换成关节坐标

参数：pose#1：空间位置变量 V，范围为 0~255。需要被转换的空间坐标变量

joint#1：关节位置变量 P，范围为 0~255。用于保存逆解计算得出的关节坐标的变量

示例：

```
POSETOJOINT V001 P001
//把变量V001的空间坐标转换成关节坐标，并将结果存入变量P001
```

2.8.3 正解指令

```
JOINTTOPOSE joint#1 pose#1
```

功能：正解指令，把关节坐标转换成空间坐标

参数：joint#1：关节位置变量 P，范围为 0~255。需要被转换的关节坐标变量

pose#1：空间位置变量 V，范围为 0~255。用于保存正解计算得出的空间坐标的变量

示例：

```
JOINTTOPOSE P005 V005
//将变量P005的关节坐标转换成空间坐标，结果存入变量V005
```

2.8.4 透传初始化

```
TTINIT T=double LOOKAHEAD=double SMOOTHNESS=double
```

功能：透传初始化。

参数：T：采样时间，范围为 0.002~0.1，单位为 s；

LOOKAHEAD：前瞻时间，范围为 0.01~1，单位为 s；

SMOOTHNESS：轨迹平滑指数，范围 0~1。

注：LOOKAHEAD 前瞻时间必须大于采样时间

示例：

```
TTINIT T=0.004S LOOKAHEAD=0.1S SMOOTHNESS=0.2
//采样时间为0.004s，前瞻时间为0.1s，轨迹平滑度为0.2
```

2.8.5 透传记录当前位置

```
TTSTARTJOINT joint
```

功能：透传记录当前位置

参数：joint：关节位置，P 变量；

示例：

```
TTSTARTJOINT P001
//记录当前P001点
```

2.8.6 透传增加目标位置

```
TTTARGETJOINT joint
```

功能：透传增加目标位置

参数：joint：关节位置，P 变量；

示例：

```
TTTARGETJOINT P001
//增加目标点P001
```

2.8.7 透传结束

```
TTSTOP
```

功能：透传结束

参数：无

示例：

```
TTSTOP
//透传结束
```

2.8.8 设置作业文件指令

```
SETMLFILE joint#1 job#1
```

功能：设置作业文件指令，将直角坐标位姿点位变换为关节坐标数据，供 MOVML 指令使用。

参数：joint#1：关节位置变量 P，范围为 0~255。程序文件第一个位姿的逆解参考点。
job#1：待运行直角坐标位姿点位文件。点位数据为间隔 10ms 的点位位姿数据。

示例：

```
NOPI  
GETPOS P0  
SETMLFILE P0 JOB:pose  
MOVML VJ=100% JOB:pose  
END  
  
//将pose.jbi文件内容进行转换，并通过MOVML指令运行
```

注意：本命令适用于 v2.10.0 及以上版本。

2.8.9 位姿求逆

```
POSEINV pose
```

功能：进行位姿的求逆计算，将 V1 求逆之后再赋值给 V1。

参数：pose: 空间位置变量 V 范围 0~255

示例：

```
POSEINV V1
```

注意：本命令适用于 v2.11.0 及以上版本。

2.8.10 加载作业文件

```
LOADML JOB:filename
```

功能：将执行的文件内容预加载到内存中，在使用 MOVML 指令运行文件时可以减少时间

参数：JOB: filename: 离线文件文件名，只能识别文件头为 dragfile、trajfile、pathfile 或 loadfile 的 JBI 文件。

示例：

```
LOADML JOB: path_path
```

注意：系统重启后，内存的文件会被删除。内存中最多存放 32 个文件，运行第 33 个文件时会替换掉第 1 个文件，依此类推。
本命令适用于 v2.11.0 及以上版本。

2.8.11 卸载作业文件

```
UNLOADML JOB:filename
```

功能：将内存中的执行文件删除

参数：JOB: filename: 离线文件文件名, 只能识别文件头为 dragfile、trajfile、pathfile 或 loadfile 的 JBI 文件。

示例：

```
UNLOADML JOB:path_path
```

注意：本命令适用于 v2.11.0 及以上版本。

2.8.12 透传增加目标位置

```
TTTARGETJOINT joint 或 TTTARGETJOINT pose
```

功能：透传增加目标位置

参数：joint: 关节位置, P 变量。

pose: 空间位姿, V 变量

示例：

```
//增加目标点P001

TTTARGETJOINT P001

//增加目标点V001

TTTARGETJOINT V001
```

2.9 其他指令

2.9.1 保存 P 变量指令

```
SAVEVARP
```

功能：保存 P 变量指令。

注：P 变量在从自动模式切换到其他模式、急停、自动运行结束、监视界面修改 P 变量的值这几种情况下也会将其保存到文件中。

参数：无

```

示例：
    SETJOINT P001 -55.0671,-76.7208,87.6881,-84.3410,93.0787,
        -0.4583,0.0000,0.0000
    SETJOINT P002 20.2756,-71.7441,64.7525,-85.7218,93.0779,
        -0.4606,0.0000,0.0000
    SAVEVARP
    //保存P变量
    
```

2.9.2 打印指令

```
TPWRITE string
```

功能：打印指令。机器人执行此指令时将指定内容记录到 log 文档并显示在信息栏内。

参数：string：打印内容。由数字和字母组成的字符串

```

示例：
    TPWRITE Completed
    //打印“Completed”字符串到log窗口
    
```

2.9.3 设置负载

```
SETPAYLOAD TOOL#(int) M=double X=double Y=double Z=double
```

功能：设置机器人工具负载的质量和质心。参数中的小数保留小数点后三位

参数：TOOL#(): 工具坐标系数数字，范围：0~7 M：负载公斤数，范围：0~12 X, Y,Z：质心偏移的空间位置，范围：-5000~5000

```

示例：
    NOP
    SETPAYLOAD TOOL #(1) M=2.0KG X=10.0MM Y=20.0MM Z=30.0MM
    //设置机器人工具号1的负载质量为2.0KG，质心X=10mm,Y=20mm,Z=30
    mm。
    TIMER T=2.0
    END
    
```

注意：本命令适用于 v2.13.2 及以上版本。

2.10 脚本指令

2.10.1 运行指定脚本文件

```
STARTLUA INDEX=int
```

功能：运行指定的脚本文件。使用该指令时须先在脚本界面设置好脚本文件。

参数：INDEX= 选择第几个脚本，范围 1~8

示例：

```
STARTLUA INDEX=1
//运行脚本1
```

2.10.2 停止脚本文件

```
STOPLUA INDEX=int
```

功能：使运行中的脚本文件停止运行。

参数：INDEX= 选择第几个脚本，范围 1~8

示例：

```
STOPLUA INDEX=1
//停止脚本1
```

2.10.3 重新加载脚本

```
RESTARTLUA INDEX=int
```

功能：重新加载脚本。

参数：INDEX= 选择第几个脚本，范围 1~8

示例：

```
RESTARTLUA INDEX=1
//重新加载脚本1
```

2.10.4 获取脚本状态

```
GETLUASTATE B#(int) INDEX=int
```

功能：获取脚本状态。0 代表停止，1 代表运行

参数：INDEX= 选择第几个脚本，范围 1~8

B#(): 将获取的状态值存放于变量 B 中

示例：

```
GETLUASTATE B000 INDEX=1  
//获取脚本1的运行状态存放于B000中
```

第 3 章 JBI 脚本示例

3.1 子程序调用

```
1 //子程序:task01.jbi
2 NOP
3 MOVJ P001 VJ=10% PL=0
4 END
5 //子程序:task02.jbi
6 NOP
7 MOVJ P002 VJ=10% PL=0
8 END
9 //主程序:main.jbi
10 NOP
11 IF IN#(4)=1 THEN
12 //如果数字输入信号4等于1,
13 CALL JOB:task01
14 //调用子程序task01
15 ELSEIF IN#(5)=1 THEN
16 //如果数字输入信号5等于1
17 CALL JOB:task02
18 //调用子程序task02
19 ELSE
20 DOUT OT#(6) ON
21 //把数字输出信号6置为ON
22 WAIT IN#(6)=1
23 //等待输入信号6为1
24 ENDIF
25 END
```



3.2 逻辑门运算

```
1 NOP
2 SET B000 1
3 SET B001 2
4 SET B002 3
5 SET B003 4
```



```
6  TIMER T=3.0
7  AND B000 B001
8  //B0=0
9  AND B001 B002
10 //B1=2
11 AND B002 B003
12 //B2=0
13 TIMER T=3.0
14 SET B000 1
15 SET B001 2
16 SET B002 3
17 SET B003 4
18 TIMER T=3.0
19 OR B000 B001
20 //B0=3
21 OR B001 B002
22 //B1=3
23 OR B002 B003
24 //B2=7
25 TIMER T=3.0
26 SET B000 1
27 SET B001 2
28 SET B002 3
29 SET B003 4
30 TIMER T=3.0
31 NOT B000 B001
32 NOT B001 B002
33 NOT B002 B003
34 TIMER T=3.0
35 SET B000 1
36 SET B001 2
37 SET B002 3
38 SET B003 4
39 TIMER T=3.0
40 XOR B000 B001
41 //B0=3
42 XOR B001 B002
43 //B1=1
44 XOR B002 B007
```

```
45 TIMER T=3.0
46 END
```

3.3 逻辑运算优先级

经测试，由逻辑运算符 `&` 和 `|` 连接的多项条件，在任何应用场景下一律遵循从左到右的判断原则。

例如：`A&B|C = A and B or C` 取条件 A 与 B 的与逻辑，再将结果与条件 C 进行或逻辑运算，得出最终判断结果。

`A|B&C|D&E = ((A B) C) D) E`：取条件 A 与 B 的或逻辑，再将结果与条件 C 进行与逻辑运算，再将结果与条件 D 进行或逻辑运算，再将结果与条件 E 进行与逻辑运算，得出最终判断结果。

提醒



当前软件版本中，所有的 IF 和 UNTIL 后(包括 PAUSEIF, WAITIF, MOVJUNTIL 等等)，如果运算符两边变量类型不相同，判断时并不会根据变量类型对原有值进行任何处理（类似四舍五入、取整、取零等），系统也不会进行任何报错。

例如：

IF I001=D003 中，如此时 I001=7，D003=7.18，则此判断不成立。

IFB001=D002 中，如此时 B001=0，D004=-4，则此判断不成立

IF OT#(5)=17 中，如此时 OT#(5) 为 ON，则此判断成立。若此时 OT#(5) 为 OFF，则此判断不成立。

3.4 正逆解计算

正解：关节坐标转化为直角坐标；逆解：直角坐标转化为关节坐标。

```
1 NOP
2 //设置关节角
3 SETJOINT P002 45,56,23,90,0,9,0.0000,0.0000
4 //转换成位姿存入V001中，正解计算
5 JOINTTOPOSE P002 V001
6 //位姿Z轴上加10mm位移
7 ADD V001(2) 10
8 //将位姿转换成关节角（最靠近P2的当前值）并存入P002,逆运动学
9 POSETOJOINT V001 P002
10 //运行到P002点位
```



```
11 MOVJ P002 VJ=100% PL=0
12 END
```

3.5 坐标系变换计算

如图 3-1 所示，坐标系 2 至坐标系 1 的变换矩阵为 V_2 ，坐标系 3 至坐标系 2 的变换矩阵为 V_1 ，则可计算出坐标系 3 至坐标系 1 的变换矩阵为 V_3 。

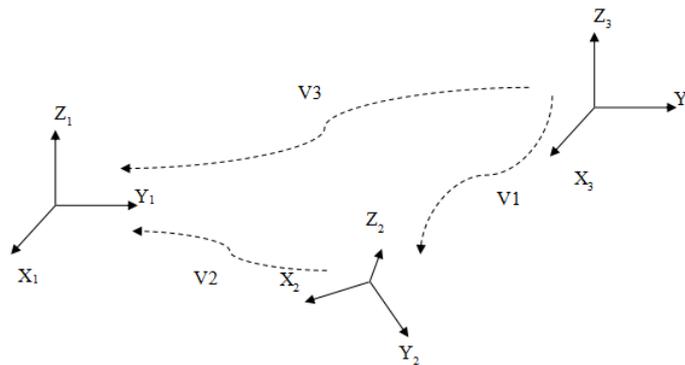


图 3-1：坐标系变换

```
1 NOP
2 //设置关节角
3 SETJOINT P002 45,56,23,90,0,9,0.0000,0.0000
4 //转换成位姿存入V002中，正解计算
5 JOINTTOPOSE P002 V002
6 //将V002数值赋值给V003
7 SET V003 V002
8 //设置V1变量
9 SETPOSE V001 0,0,200.00,0,0,1.5707
10 //坐标变换，结果存入V003变量
11 POSEMUL V003 V001
12 //将位姿转换成关节角（最靠近P002的当前值）并存入P002，逆运动学
13 POSETOJOINT V003 P002
14 //运行到P002点位
15 MOVJ P002 VJ=100% PL=0
16 END
```

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: tech@elibot.cn

- 苏州公司 (生产研发基地)

苏州市工业园区长阳街 259 号中新钟园工业坊 4 号楼 1F (总部)

+86-400-189-9358

+86-0512-83951898

- 北京分公司

北京市北京经济技术开发区荣华南路 2 号院 6 号楼 1102 室

- 上海分公司

上海市浦东新区学林路 36 弄 18 号楼

- 深圳技术服务中心

深圳市宝安区航空路泰华梧桐岛科技创新园 1A 栋 202 室



关注公众号了解更多