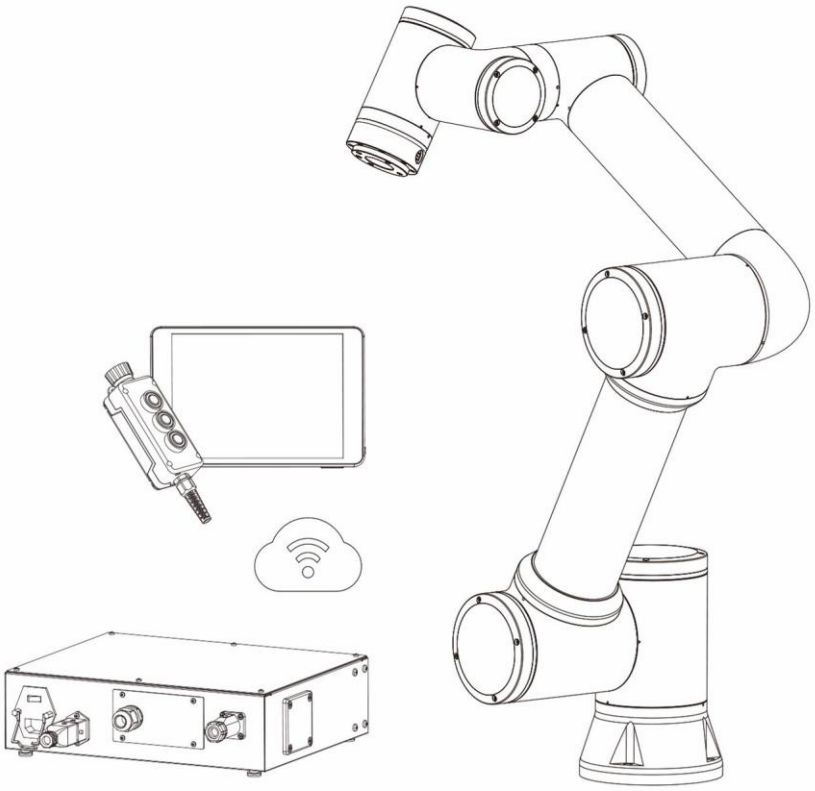
FAIR INNOVATION

FRLua 编程脚本

用户手册



法奥意威（苏州）机器人系统有限公司

资料编码 20200310

目 录

修订记录

日 期	版 本	描 述	作 者
2024/09/30	V1.0	用户手册首次编写发布	法奥

1 概述

欢迎使用法奥协作机器人 FRLua 编程脚本用户手册。本手册基于软件版本 v3.7.4 进行撰写，手册旨在为用户提供全面的指南，帮助您熟练掌握如何在法奥协作机器人上使用 FRLua 脚本进行编程。通过 FRLua 脚本，用户可以灵活地控制机器人执行各种任务。

2 FRLua 编程脚本基础

2.1 基础语法

2.1.1 FRLua 注释

FRLua 脚本中的注释分为单行注释和多行注释。

FRlua 中的单行注释，使用 -- 开头，后面跟随的内容将被解释器忽略。单行注释通常用于对代码的简短解释或标记。

FRlua 中多行注释，使用 --[[ ]] 进行注释，以 --[[开始，并以]] 结束，适用于需要对大段代码进行说明的情况。

FRLua 不支持嵌套注释，这意味着在一个多行注释内不能再嵌套另一个多行注释，FRLua 注释说明说明如下。

代码 2-1 FRLua 注释说明

```
--这是一个单行注释

--[[
    这是一个多行注释
    可以包含多行文本
]]

--[[
    这是一个多行注释
    --[[
    嵌套的多行注释（这是非法操作）
    ]]
    ]]
```

2.1.2 FRLua 关键字

下面是 FRLua 中被保留的关键词，它们不能作为名称使用，FRLua 关键字如下所示：

代码 2-2 FRLua 关键字

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

2.1.3 变量

在 FRLua 环境中，变量是用于存储数据的标识符，它们可以保存包括函数和表在内的多种数据类型。变量名由字母、数字和下划线组成，必须以字母或下划线开始，FRLua 区分大小写。

1）变量类型

FRLua 中包含三种变量类型：

全局变量：默认情况下，所有变量均为全局变量，除非通过 local 显式声明为局部变量。

局部变量：通过 local 关键字声明，作用域从声明位置开始直到当前语句块结束。

表中的域：表是 FRLua 中非常重要的数据结构，表中的域也可作为变量，FRLua 变量类型示例如下所示。

代码 2-3 FRLua 变量类型示例

```
1    --全局变量
2    globalVar = 10
3    --局部变量
4    local localVar = 20
5    --表中的域
6    local tableVar = {key = 30}
```

2）赋值与多值赋值

赋值语句：用于改变变量或表域的值，通过“=”，右侧的值会依次赋给左侧的变量，FRLua 赋值示例如下所示。

代码 2-4 FRLua 赋值示例如下所示

```
1    local a = 5
2    local b = 10
3    a = b    -- a 的值现在是 10
```

多值赋值：FRLua 支持同时对多个变量赋值，通常用于交换变量的值或将函数返回值赋给多个变量, FRLua 多值赋值示例如下所示

代码 2-5 FRLua 多值赋值示例

```
1    -- 交换变量值
2    local x = 1
3    local y = 2
4    x, y = y, x    -- x 现在是 2, y 现在是 1
5    -- 函数返回多个值
6    local function getValues()
7        return 5, 10
8    end
9
10   local a, b = getValues()    -- a 现在是 5, b 现在是 10
```

2.1.4 数据类型

FRLua 支持八种基本数据类型：nil、布尔值、数字、字符串、用户数据、函数、线程和表，基本数据类型描述如表 2-1 所示。

表 2-1 FRLua 数据类型描述

数值类型	描述
nil	表示无效值，变量没有赋值时的默认值。
布尔值（boolean）	仅包含 true 和 false 两个值，通常用于条件判断。
数字（number）	包含整数和浮点数，RFLua 使用双精度浮点数表示数字。
字符串（string）	用于存储文本，可以用单引号、双引号或长字符串表示。
用户数据（userdata）	用于存储由 C 语言生成的外部数据。
函数（function）	由 C 或 RFLua 编写的函数，可赋值、传递和返回，是逻辑实现的基础。
线程（thread）	线程作为协程实现，允许并发执行，每个线程拥有独立的执行栈，同时共享全局环境和状态。

在 FRLua 中可以使用 type 函数测试给定变量或者值的类型，查看变量或值的类型示例如下所示

代码 2-6 查看变量或值的类型示例

```
1  print(type("Hello world"))      --> string
2  print(type(10.4*3))             --> number
3  print(type(print))              --> function
4  print(type(type))               --> function
5  print(type(true))               --> boolean
6  print(type(nil))                --> nil
7  print(type(type(X)))            --> string
```

FRLua 基本数据类型的使用示例

1）nil 数据类型

表示“无效”或“空值”，是默认的未初始化值。通常用来表示没有值或一个变量的值未定义，示例如下：

代码 2-7 nil 数据类型示例

```
1.  local a = nil
2.  print(a)  -- 输出: nil
```

2）布尔（boolean）

在 FRLua 中，布尔类型只有两个值：true 和 false。除了 false 和 nil，其他所有值（包括数字 0）都被视为 true，示例如下：

代码 2-8 布尔数据类型示例

```
1.  --定义布尔值
2.  local a = true
3.  local b = false
4.  --直接输出布尔值
5.  print(a)  -- 输出: true
6.  print(b)  -- 输出: false
7.  --布尔条件判断
8.  if a then
9.      print("a 是 true")  -- 输出: a 是 true
10. end
11. if not b then
12.     print("b 是 false")  -- 输出: b 是 false
13. end
14. --注意：数字 0 也被视为 true
15. local c = 0
```

代码 2-8（续）

```
if c then
    print("数字 0 也被视为 true")  -- 输出: 数字 0 也被视为 true
end
--false 和 nil 会被视为 false
local d = nil
if not d then
    print("nil 被视为 false")  -- 输出: nil 被视为 false
end
```

3）数字（number）

数字类型用于存储整数或浮点数。FRLua 使用双精度浮点数表示 number 类型，因此可以精确表示大范围的整数和小数，示例如下：

代码 2-9 数字数据类型示例

```
local x = 10          -- 整数
local y = 3.14        -- 浮点数
local z = 1e3         -- 科学计数法，表示 1000
print(type(x))        --输出 number
print(x)              -- 输出: 10
print(type(y))        --输出 number
print(y)              -- 输出: 3.14
print(type(z))        --输出 number
print(z)              -- 输出: 1000.0
```

4）字符串（string）

字符串用于存储文本数据。字符串可以用单引号、双引号或者 [[]] 来表示多行字符串，使用..可以链接两个字符串示例如下：

代码 2-10 字符串数据类型示例

```
--示例 1：使用单引号和双引号
local str1 = 'Hello, FR!'        -- 使用单引号
local str2 = " Hello, FR! "      -- 使用双引号
print(str1)   -- 输出: Hello, FR!
print(str2)   -- 输出: Hello, FR!
--示例 2：[[ ]],当字符串较长或者需要跨多行时，可以使用 [[ ]] 来表示多行字符串。
local multi_line_str = [[
我是法奥协作机器人。
感谢您的信任。
请问有什么需要我帮您！
]]
```

代码 2-10（续）

```
print(multi_line_str)
-- 输出:
-- 我是法奥协作机器人。
-- 感谢您的信任。
-- 请问有什么需要我帮您！

--示例 3：字符串连接
--FRLua 使用 .. 来连接多个字符串。
local part1 = "Hello"
local part2 = "FR!"
local combined = part1 .. " " .. part2   -- 用 .. 连接字符串
print(combined)   -- 输出: Hello FR!
```

5）用户数据（userdata）

用户数据是一种特殊类型，用于表示由 C/C++ 代码创建的数据。在 FRLua 中，它通常用于与外部程序或库进行交互。

6）函数（function）

FRLua 中的函数也是一种数据类型，可以被赋值给变量或作为参数传递。函数可以是命名的或匿名的（lambda 函数），示例如下：

代码 2-11 函数数据类型示例

```
local function greet()
    print("Hello，FR!")
end
greet()   -- 输出: Hello，FR！
```

7）线程（thread）

FRLua 中线程用于表示协程实现，协程允许在不同的代码块之间进行非抢占式多任务处理，类似于轻量级的线程，示例如下：

代码 2-12 函数线程数据类型示例

```
local co = coroutine.create(function()
print("Running coroutine") end)
coroutine.resume(co)   -- 输出: Running coroutine
```

8）表（table）

FRLua 的核心数据结构是 table，通过 { } 创建空表。它作为关联数组，支持数字和字符串索引，提供数据组织的灵活性。table 索引从 1 开始，长度随内

容自动扩展，未赋值元素默认为 nil。

创建和使用 table 的基本语法，示例如下：

代码 2-13 table 的基本语法

```
local FR= {}  -- 创建一个空表
local USER= {"Hello", "FR", "!"} -- 直接初始化表
```

table 的数字索引（类似数组）示例如下：

代码 2-14 table 的数字索引

```
-- 数字索引的表
local numbers = {10, 20, 30, 40}
-- 访问表中的值
print(numbers[1])  -- 输出: 10
print(numbers[2])  -- 输出: 20
-- 表的长度会自动扩展
numbers[5] = 50
print(numbers[5])  -- 输出: 50
```

table 的字符串索引（类似字典）示例如下：

代码 2-15 table 的字符串索引

```
-- 字符串索引的表
local person = {
    name = "FR",
    age = FR,
    city = "China"
}
-- 访问表中的值
print(person.name)  -- 输出: FR
print(person["age"])  -- 输出: FR
```

table 的混合索引，FRLua 的表可以同时使用数字和字符串索引，示例如下：

代码 2-16 table 的混合索引

```
-- 创建一个混合索引的表。
local fr_robots = {
    "FR3",
    "FR5",
    "FR10",
    company = "FR",
    founded = 2019
}
```

table 的动态增长，示例如下：

代码 2-17 table 的动态增长

```
-- 创建一个空的表用于存储机器人型号
local fr_models = {}
-- 动态添加机器人产品型号
fr_models[1] = "FR3"
fr_models[2] = "FR5"
fr_models[3] = "FR10"

-- 动态添加更多信息
fr_models["total_models"] = 3
fr_models["latest_model"] = "FR10"

-- 访问表中的数据
print(fr_models[1])    -- 输出: FR3
print(fr_models["total_models"])    -- 输出: 3
print(fr_models["latest_model"])    -- 输出: FR10
```

2.1.5 运算符

1）算术运算符

FRLua 中常用的算术运算符，主要有加（+）、减（-）、乘（*）、除（/）、取余（%）、乘幂（^）、符号（-）、整除（//）。常用算术运算符的使用示例如下：

代码 2-18 FRLua 中的常用算术运算符使用示例

```
-- 变量定义
local FR_1 = 10
local FR_2 = 20

-- 加法
local addition = FR_1 + FR_2
print("加法 (FR_1 + FR_2):", addition)    -- 输出: 加法 (FR_1 + FR_2): 30

-- 减法
local subtraction = FR_1 - FR_2
print("减法 (FR_1 - FR_2):", subtraction)    -- 输出: 减法 (FR_1 - FR_2): -10
-- 乘法
local multiplication = FR_1 * FR_2
```

代码 2-18（续）

```
print("乘法 (FR_1 * FR_2):", multiplication)  -- 输出: 乘法 (FR_1 * FR_2): 200

-- 除法
local division = FR_2 / FR_1
print("除法 (FR_2 / FR_1):", division)  -- 输出: 除法 (FR_2 / FR_1): 2.0

-- 取余
local modulo = FR_2 % FR_1
print("取余 (FR_2 % FR_1):", modulo)  -- 输出: 取余 (FR_2 % FR_1): 0

-- 乘幂
local power = FR_1 ^ 2
print("乘幂 (FR_1 ^ 2):", power)  -- 输出: 乘幂 (FR_1 ^ 2): 100

-- 负号
local negative = -FR_1
print("负号 (-FR_1):", negative)  -- 输出: 负号 (-FR_1): -10

-- 整除
local integerDivision = 5 // 2
print("整除 (5 // 2):", integerDivision)  -- 输出: 整除 (5 // 2): 2
```

2）关系运算符

FRLua 中常用的关系运算符，主要有等于（==）、不等于（~=）、大于（>）、小于（<）、大于等于（>=）和小于等于（<=）。常用关系运算符的使用示例如下：

代码 2-19 FRLua 中常用关系运算符使用示例

```
-- 变量定义
local FR_1 = 10
local FR_2 = 20
-- 等于
local isEqual = (FR_1 == FR_2)
print("等于 (FR_1 == FR_2):", isEqual)  -- 输出: 等于 (FR_1 == FR_2): false
-- 不等于
local isNotEqual = (FR_1 ~= FR_2)
print("不等于 (FR_1 ~= FR_2):", isNotEqual)  -- 输出: 不等于 (FR_1 ~= FR_2): true
-- 大于
local isGreaterThan = (FR_1 > FR_2)
print("大于 (FR_1 > FR_2):", isGreaterThan)  -- 输出: 大于 (FR_1 > FR_2): false
```

代码 2-19（续）

```
-- 小于
local isLessThan = (FR_1 < FR_2)
print("小于 (FR_1 < FR_2):", isLessThan)  -- 输出: 小于 (FR_1 < FR_2): true
-- 大于等于
local isGreaterOrEqual = (FR_1 >= FR_2)
print("大于等于 (FR_1 >= FR_2):", isGreaterOrEqual)  -- 输出: 大于等于 (FR_1 >= FR_2): false
-- 小于等于
local isLessOrEqual = (FR_1 <= FR_2)
print("小于等于 (FR_1 <= FR_2):", isLessOrEqual)  -- 输出: 小于等于 (FR_1 <= FR_2): true
```

3）逻辑运算符

RLua 中常用的逻辑运算符，主要有与（==）、或（~=）和非（>）。常用逻辑运算符的使用示例如下：

代码 2-120 FRLua 中常用逻辑运算符使用示例

```
--示例：
local FR = true
local NFR = false
local result1 = FR and NFR
print("逻辑与 (FR and NFR):", result1)  -- 输出: 逻辑与 (FR and NFR): false
local result2 = FR or NFR
print("逻辑或 (FR or NFR):", result2) -- 输出: 逻辑或 (FR or NFR): true
local result3 = not FR
print("逻辑非 (not FR):", result3) -- 输出: 逻辑非 (not FR): false
```

4）其他运算符

RLua 中常用的其他运算符，主要有连接运算符（..）、表索引（[ ]）、赋值（=）、表构造器（{ }）和变量长度运算符（#）。常用其他运算符的使用示例如下：

代码 2-21 FRLua 中常用其他运算符使用示例

```
--其他运算符使用示例：
local str1 = "Hello"
local str2 = "FR!"
local result = str1 .. " " .. str2
print("连接运算符 (str1 .. ' ' .. str2):", result)  -- 输出: Hello FR!
local myTable = {a = 1, b = 2}
local valueA = myTable["a"]
```

代码 2-21（续）

```
print("表索引 (myTable['a']):", valueA)  -- 输出: 1
local x = 5
print("赋值 (x = 5):", x)  -- 输出: 5
local myTable = {1, 2, 3, 4}
print("表构造器 (myTable[1]):", myTable[1])  -- 输出: 1
print("变量长度运算符 (#myTable):", length) -- 输出: 4
```

2.2 控制结构

2.2.1 条件语句

FRLua 中使用 if、elseif 和 else 关键字来执行不同的代码块，根据条件的真假决定执行路径。以下是 FRLua 条件语句的工作机制：

if 条件语句: 用于指定一个条件。如果条件为 true，则执行该条件块中的代码。

elseif 条件语句: 在 if 条件为 false 时，可以提供另一个条件进行检查。

else 条件语句: 如果所有的 if 和 elseif 条件都为 false，则执行 else 块中的代码。

条件的评估：FRLua 中假定布尔值 true 和非 nil 值为真。布尔值为 false 或 nil 会被认为是假。

注意: 在 FRLua 中，0 被视为 true，与某些其他编程语言不同。

条件语句的基本结构：

代码 2-22 FRLua 中条件语句的基本结构

```
if condition1 then
    -- 当 condition1 为 true 时执行的代码块

elseif condition2 then
    -- 当 condition1 为 false，condition2 为 true 时执行的代码块

else
    -- 当所有条件都为 false 时执行的代码块

end
```

以下为条件语句使用示例：

代码 2-23 条件语句使用示例

```
1.  --示例 1: 判断数字正负
2.  local number = 10
3.
4.  if number > 0 then
5.      print("数字是正数")
6.  elseif number < 0 then
7.      print("数字是负数")
8.  else
9.      print("数字是零")
10.
11. end
12. --输出结果：数字是正数
13.
14. --示例 2: 检查变量是否为 nil
15. local value = nil
16.
17. if value then
18.     print("变量不为 nil")
19. else
20.     print("变量为 nil")
21.
22. end·
23. --输出结果：变量为 nil
24.
25. --示例 3: FRLua 中 0 被视为 true
26. local num = 0
27.
28. if num then
29.     print("0 被视为 true")
30. else
31.     print("0 被视为 false")
32.
33. end
34.     --输出结果：0 被视为 true
```

2.2.2 循环语句

在 FRLua 编程中，很多时候需要重复执行某些代码段，这种重复操作称为循环。循环由两部分组成：循环体和循环的终止条件。FRLua 提供了多种循环控

制结构，用来在满足条件时反复执行某段代码。

循环由循环体和终止条件组成，其中循环体是指被重复执行的一组语句；终止条件是指决定循环是否继续的条件，当条件为 false 时，循环结束。

1）while 循环

while 循环会在指定的条件为 true 时重复执行代码块，直到条件为 false 才会终止。while 循环的基本结构与示例如下：

代码 2-24 FRLua 中 while 循环的基本结构

```
while condition do
        -- 循环体

end
```

代码 2-25 while 的使用示例

```
---示例: 计算 1 到 5 的和
local sum = 0
local i = 1
while i <= 5 do
    sum = sum + i
    i = i + 1
end
print("1 到 5 的和:", sum)   -- 输出: 1 到 5 的和: 15
```

2.for 数值循环：for 循环用于在一个范围内的数字进行迭代，并执行循环体。for 数值循环的基本结构与示例如下：

代码 2-26 FRLua 中 for 数值循环的基本结构

```
for i = start, end, step do
        -- 循环体

end
```

代码 2-27 for 数值循环的使用示例

```
示例: 输出 1 到 5 的数字
for i = 1, 5 do
        print(i)

end
 --[[输出：
        1
        2
```

代码 2-27（续）

```
9.          3
10.         4
11.         5
12.  ]]
```

3） for 泛型循环

for 泛型循环用于遍历表或迭代器。for 泛型循环的基本结构与示例如下:

代码 2-28 FRLua 中 for 泛型循环的基本结构

```
1.   for key, value in pairs(table) do
2.           -- 循环体
3.
4.   end
```

代码 2-29 for 泛型循环的使用示例

```
1.   --示例: 遍历表中的键和值
2.   local myTable = {a = 1, b = 2, c = 3}
3.   for key, value in pairs(myTable) do
4.       print(key, value)
5.   end
6.   --[[输出:
7.   a 1
8.   b 2
9.   c 3
10.  ]]
```

4）repeat...until 循环

repeat...until 循环与 while 循环类似，但它会先执行循环体，再检查条件。只有条件为 false 时，才会继续执行。repeat...until 循环的基本结构与示例如下:

代码 2-30 FRLua 中 repeat...until 循环的基本结构

```
1.   repeat
2.           -- 循环体
3.
4.   until condition
```

代码 2-31 repeat...until 循环的使用示例

```
--示例: 计算 1 到 5 的和
local sum = 0
local i = 1
repeat
    sum = sum + i
    i = i + 1
until i > 5
print("1 到 5 的和:", sum)
-- 输出: 1 到 5 的和: 15
```

5）嵌套循环

嵌套循环是指一个循环结构包含在另一个循环结构中。这通常用于处理多维数据或需要重复执行多组相似操作的场景。在嵌套循环中，外层循环控制大的操作，如行的数量，内层循环控制小的操作，如每行的具体内容。

代码 2-32 嵌套循环示例

```
--示例: 打印一个 5x5 的星号矩阵。
for i = 1, 5 do   -- 外层循环，控制行
    for j = 1, 5 do   -- 内层循环，控制列
        io.write("* ")   -- 输出星号，保持在同一行

    end
    print()   -- 每完成一行的输出后换行

end
for i = 1, 5 do   -- 外层循环，控制行
    for j = 1, 5 do   -- 内层循环，控制列
        io.write("* ")   -- 输出星号，保持在同一行
    end
    print()   -- 每完成一行的输出后换行

end
--[[输出：
 * * * * *
 * * * * *
 * * * * *
 * * * * *
 * * * * *
]]
```

2.2.3 控制语句

FRLua 提供了两种控制循环的特殊语句，分别为 break 和 goto。

break 语句：用于提前跳出当前的循环。当循环遇到 break 语句时，会立即结束循环，跳出当前循环体，不再执行后续的迭代。可以避免不必要的循环，提升效率。

代码 2-33 FRLua 中 break 语句示例

```
--示例: 当计数器达到 3 时退出循环
for i = 1, 5 do
    if i == 3 then   -- 当 i 等于 3 时，退出循环
        break   -- 提前终止循环

    end
    print(i)
end

--[[输出
1
2]]
```

goto 语句：可以无条件跳转到指定的标签位置。可以在某些复杂情况下可以简化代码逻辑。

代码 2-34 FRLua 中 goto 语句示例

```
--示例: 使用 goto 跳过某些语句
local i = 1
::loop_start::   -- 定义一个标签
print(i)
i = i + 1
if i <= 5 then
    goto loop_start   -- 跳回标签，继续循环
end
--[[
输出:
1
2
3
4
5
]]
```

代码 2-35 基础语法与控制结构结合的简单使用

```
-- 法奥协作机器人基本数据类型
local robotModel = "FR5"    -- 机器人型号
local payloadCapacity = 5    -- 负载能力 (kg)
local isOperational = true    -- 机器人是否正在工作
local errorCode = nil    -- 错误码 (初始为空)

-- 变量赋值
local operationHours = 100    -- 机器人运行时间 (小时)
operationHours = operationHours + 10    -- 增加 10 小时运行时间

-- 条件语句
if payloadCapacity > 10 then
    print("FR5 机器人负载超过 10kg")
elseif payloadCapacity < 3 then
    print("FR5 机器人负载小于 3kg")
else
    print("FR5 机器人的负载在 3 到 10kg 之间")
end

-- 循环语句：显示机器人运行的前 5 小时
for hour = 1, 5 do
    print("FR5 已运行小时数: " .. hour)
end

-- while 循环：检查是否达到满负荷运行
local currentLoad = 0
while currentLoad < payloadCapacity do
    print("当前负载: " .. currentLoad .. "kg, 继续增加...")
    currentLoad = currentLoad + 1
end

-- repeat 循环：减少负载，直到负载为 0
repeat
    print("减轻负载中，当前负载: " .. currentLoad .. "kg")
    currentLoad = currentLoad - 1
until currentLoad == 0

print("FR5 负载已清零，准备停止")
```

2.3 函数

函数是程序中一组指令的抽象，用于执行特定的任务或计算并返回结果。FRLua 语言中的函数在编写和使用时非常灵活，可以有参数、无参数，可以返回一个值或多个值。

2.3.1 FRLua 函数的定义与使用

FRLua 中的函数是编程的重要组成部分，支持封装重复代码、实现模块化编程、以及处理复杂的逻辑操作，FRLua 函数的基本结构如下：

代码 2-36 FRLua 函数的基本结构

```
optional_function_scope function function_name(argument1, argument2, ...)
        -- 函数体：函数的操作
        return result_params_comma_separated
end
```

函数结构解析：

optional_function_scope: 可选的作用域设置。如果需要函数仅在特定模块或块中使用，可以用 local 来定义局部函数，未设置时默认为全局函数。

- function_name: 函数名称，用于标识函数，便于调用。
- argument1, argument2, ...: 函数的参数，传递到函数中的数据。
- function_body: 函数体，包含需要执行的具体代码。
- return result_params_comma_separated: 函数的返回值，可以返回多个值。

函数的两种主要用途：

完成任务：如在协作机器人中调用函数执行移动、抓取等操作，函数作为调用语句使用。

计算并返回值：如计算负载或坐标时，函数作为赋值语句的表达式使用。

2.3.2 匿名函数与闭包

在 FRLua 中，匿名函数（Aaonymous Function）闭包（Closures）是函数编程中两个重要的概念。它们允许在函数中创建动态的行为，非常适用于需要灵活处理的场景。

匿名函数指的是没有名字的函数。匿名函数可以直接作为参数传递给其他函

数，或者赋值给一个变量。

代码 2-37 FRLua 匿名函数的基本结构与调用

```
1.  -- 创建一个匿名函数并赋值给变量
2.  local greet = function(name)
3.      return "Hello, " .. name
4.  end
5.
6.  -- 调用匿名函数
7.  print(greet("FR "))  -- 输出: Hello, FR
8.  --匿名函数通常用于回调函数或一次性使用的场景。
9.  示例：匿名函数作为参数
10. -- 定义一个执行函数，接受另一个函数作为参数
11. local function execute(func, value)
12.     return func(value)
13. end
14.
15. -- 使用匿名函数作为参数传递
16. print(execute(function(name) return "Hi, " .. name end, "FR5"))
17. -- 输出: Hi, FR5
```

闭包是 FRLua 中的一种强大特性。闭包是一个函数，它不仅仅包含函数的代码，还包含了它外部环境的变量。换句话说，闭包允许函数访问它定义时所在的环境，即使这个环境已经不再存在。

闭包的主要特点：可以捕获并存储外部变量的状态，即使在外部函数已经执行完后，捕获的变量仍然可以被访问。这使得闭包可以用于创建工厂函数、延迟计算等应用场景。

代码 2-38 FRLua 闭包的基本结构与调用

```
1.  -- 创建一个闭包，返回一个函数
2.  local function create_counter()
3.      local count = 0   -- 外部变量
4.      return function()
5.          count = count + 1
6.          return count
7.      end
8.  end
9.
10. -- 创建两个独立的计数器
11. local counter1 = create_counter()
12. local counter2 = create_counter()
```

代码 2-38（续）

```
13.
14.  -- 调用计数器闭包
15.  print(counter1())   -- 输出: 1
16.  print(counter1())   -- 输出: 2
17.
18.  print(counter2())   -- 输出: 1   （不同的闭包有独立的状态）
19.  print(counter2())   -- 输出: 2
```

2.3.3 函数参数

FRLua 函数支持不同类型的参数传递方式，允许传递基本数据类型、表、以及函数等。

1）传递基本数据类型 可以将数值、字符串、布尔值作为参数传递：

代码 2-39 函数传参示例

```
1.  function set_speed(speed)
2.      print("设置速度为:", speed)
3.  end
4.  set_speed(15)
```

2）传递表 (Table) 作为参数 表作为复杂数据结构，常用于传递多个相关值：

代码 2-40 函数以表作为参数传入

```
1.  function set_position(pos)
2.      print("机器人移动到位置:", pos.x, pos.y, pos.z)
3.  end
4.
5.  local position = {x = 100, y = 200, z = 300}
6.  set_position(position)
```

3）可变参数（Varargs）, FRLua 支持可变参数，即函数可以接受任意数量的参数。通过 ... 语法实现：

代码 2-41 函数以表作为参数传入

```
1.  function print_args(...)
2.      local args = {...}   -- 将所有参数打包为一个 table
3.      for i, v in ipairs(args) do
4.          print("参数" .. i .. ": " .. v)
5.      end
6.  end
```

代码 2-41（续）

```
print_args("Hello，FR!", 123, true)
--[[
输出:
参数 1: Hello，FR!
参数 2: 123
参数 3: true
]]
```

2.3.4 函数的返回值

FRLua 函数可以返回任意类型的值，包括单个值、多个值或表。返回值用于向调用者提供操作结果。

1）返回单个值：函数可以返回一个计算结果或状态信息：

代码 2-42 返回单个值

```
function square(x)
    return x * x
end

local result = square(5)  -- 返回 25
print(result)  -- 输出: 25
```

2）返回多个值 如果函数需要返回多个结果，可以使用逗号分隔多个 return：

代码 2-43 返回多个值

```
function calculate(a, b)
    local sum = a + b
    local diff = a - b
    return sum, diff
end

local sum_result, diff_result = calculate(10, 5)
print("和:", sum_result, "差:", diff_result)
--[[
输出
和: 15  差: 5
]]
```

3）返回表：复杂的数据可以通过表进行返回，特别是在需要传递多个值时：

代码 2-44 返回表

```
function get_robot_status()
    return {speed = 10, position = {x = 100, y = 200, z = 300}}
end

local status = get_robot_status()
print("速度:", status.speed)
print("位置:", status.position.x, status.position.y, status.position.z)
--[[
输出
速度: 10
位置: 100 200 300
]]
```

2.3.5 函数作为参数和返回值

FRLua 是一门函数式语言，允许函数作为参数传递给其他函数，也允许函数返回其他函数。这种特性可以用于创建灵活的回调机制和高阶函数。

1）函数作为参数：可以将一个函数作为参数传递给另一个函数，实现回调：

代码 2-45 函数作为参数传入

```
function operate_on_numbers(a, b, operation)
    return operation(a, b)
end

local result = operate_on_numbers(5, 10, function(x, y)
    return x * y
end)
print(result)    -- 输出: 50
```

2）函数作为返回值：函数还可以返回另一个函数，这在创建定制逻辑时非常有用：

代码 2-46 函数作为参数传入

```
function multiplier(factor)
    return function(x)
        return x * factor
    end
end
```

代码 2-46（续）

```
local double = multiplier(2)
local triple = multiplier(3)

print(double(5))  -- 输出: 10
print(triple(5))  -- 输出: 15
```

2.3.6 递归函数

FRLua 支持递归，即函数调用自身。递归函数常用于处理分解问题、遍历树状结构等任务。

递归函数的基本结构通常由两个部分组成：

基准条件（Base Case）： 这是递归的终止条件，防止函数无限递归。

递归调用（Recursive Case）： 函数通过递归调用自身，逐步缩小问题的规模，直到符合基准条件。

代码 2-47 递归函数的基本结构

```
function recursive_function(param)
    if 基准条件 then
        -- 终止递归，返回结果
        return 结果
    else
        -- 递归调用
        return recursive_function(缩小后的参数)
    end
end
```

简单示例：阶乘计算，阶乘是递归函数的经典例子。阶乘的定义是：n! = n * (n-1) * (n-2) * ... * 1，并且 0! = 1。我们可以通过递归函数来计算阶乘。

阶乘递归公式：n! = n * (n - 1)!

基准条件是 0! = 1

代码 2-48 递归实现阶乘

```
function factorial(n)
    if n == 0 then
        return 1  -- 基准条件：0 的阶乘为 1
    else
        return n * factorial(n - 1)  -- 递归调用：n * (n-1)!
```

代码 2-48（续）

```
6.          end
7.    end
8.
9.    print(factorial(5))   -- 输出：120
10.   --[[
11.   执行过程：
12.   factorial(5) 计算 5 * factorial(4)
13.   factorial(4) 计算 4 * factorial(3)
14.   直到 factorial(0) 返回 1，递归逐级返回结果。
15.   ]]
```

2.4 字符串

在 FRLua 语言中，字符串是一种基本的数据类型，用于存储文本数据。FRLua 中的字符串可以包含各种字符，包括但不限于字母、数字、符号、空格以及其他特殊字符。

2.4.1 字符串定义

在 FRLua 中，字符串可以用单引号 '、双引号 " 或者长方括号 [[ ]] 来表示。长方括号常用于表示多行字符串。

代码 2-49 字符串定义示例

```
1.    -- 使用单引号定义字符串
2.    local str1 = 'Hello FR3'
3.    -- 使用双引号定义字符串
4.    local str2 = "Welcome to FR "
5.    -- 使用长方括号定义多行字符串
6.     local str3 = [[
7.    This is a multi-line
8.    string for FR5 product.
9.    ]]
10.   -- 输出字符串
11.   print(str1)   -- 输出: Hello FR3
12.   print(str2)   -- 输出: Welcome to FR
13.   print(str3)   -- 输出: This is a multi-line
14.                 --          string for FR5 product.
```

2.4.2 转义字符

Lua 支持使用反斜杠 \ 来表示转义字符。转义字符和所对应的意义：

表 2-2 转义字符和所对应的意义

转义字符	意义	ASCII 码值（十进制）
\a	响铃(BEL)	007
\b	退格(BS) ，将当前位置移到前一列	008
\f	换页(FF)，将当前位置移到下页开头	012
\n	换行(LF) ，将当前位置移到下一行开头	010
\r	回车(CR) ，将当前位置移到本行开头	013
\t	水平制表(HT) （跳到下一个 TAB 位置）	009
\v	垂直制表(VT)	011
\\	代表一个反斜线字符"\'	092
\'	代表一个单引号（撇号）字符	039
\"	代表一个双引号字符	034
\0	空字符(NULL)	000
\ddd	1 到 3 位八进制数所代表的任意字符	三位八进制
\xhh	1 到 2 位十六进制所代表的任意字符	二位十六进制

2.4.3 字符串操作

Lua 提供了多种用于字符串操作的内置函数，下面为一些常用函数：

表 2-3 字符串操作常用函数

序号	方法 & 用途
1	string.upper (argument)将字符串全部转为大写字母。
2	string.lower (argument)将字符串全部转为小写字母。
3	string.gsub (mainString, findString, replaceString，num)在字符串中替换指定字符。
4	string.find (str, substr, [init, [plain]])在指定的字符串中查找子串，返回子串

表 2-3（续表）

序号	方法 & 用途
4	的起始和结束索引。
5	string.reverse (arg)将字符串反转。
6	string.format (...)格式化字符串。
7	string.char(arg) 和 string.byte(arg) string.char：将整型数字转为字符。 string.byte：将字符转为整数值。
8	string.len(arg)计算字符串长度。
9	string.rep(string, n))返回字符串的 n 个拷贝。
10	string.match(str, pattern, init)从指定的字符串 str 中寻找与模式 pattern 匹配的第一个子串。
11	string.sub(s, i [, j])进行字符串截取操作。

1）string.upper：将小写字母转换成大写字母

表 2-4 string.upper 的详细参数

属性	说明
原型	local upper_str = string.upper(argument)
描述	将字符串全部转为大写字母
参数	• argument：待转换的字符串
返回值	• upper_str：转换后输出的字符串

代码 2-50 string.upper 示例

```
1.  local original_str = "Hello, FR!"
2.
3.  local upper_str = string.upper(original_str)
4.
5.  print(upper_str)  -- 输出 "HELLO, FR!"
```

2） string.lower：将小写字母转换成大写字母

表 2-5 string.lower 的详细参数

属性	说明
原型	local lower_str = string.lower(str)
描述	将字符串中的所有大写字母转换为小写字母
参数	• str：待转换的字符串
返回值	• lower_str：转换后输出的字符串

代码 2-51 string.upper 示例

```
1.  local original_str = "HELLO, FR!"
2.  local lower_str = string.lower(original_str)
3.  print(lower_str)  -- 输出 "hello, fr!"
```

3） string.gsub：字符串中进行全局替换

表 2-6 string.gsub 的详细参数

属性	说明
原型	local newString = string.gsub(mainString, findString, replaceString, num)
描述	用于在字符串中进行全局替换，即替换所有匹配的子串。
参数	• mainString：要进行替换操作的原始字符串； • findString：要查找的子串或模式； • .（点）：匹配任意单个字符。 • %：转义特殊字符或 Lua 模式。例如，%. 表示字面意义上的点字符。 • %a：匹配任意字母（[A-Za-z]）。 • %c：匹配任意控制字符。 • %d：匹配任意数字（[0-9]）。 • %l：匹配任意小写字母。 • %u：匹配任意大写字母。 • %x：匹配任意十六进制数字（[0-9A-Fa-f]）。 • %p：匹配任意标点符号。 • %s：匹配任意空白字符（空格、制表符、换行符等）。 • %w：匹配任意字母数字字符（相当于 %a%d）。 • %b：匹配任意单词边界。 • %f：匹配任意文件名字符。 • %[：匹配任意字符类。 • %]：结束字符类。 • %*：表示前面的字符或子模式可以出现零次或多次。 • %+：表示前面的字符或子模式至少出现一次。 • %-：表示前面的字符或子模式出现零次或一次。 • %?：表示前面的字符或子模式出现零次或一次。 • %n：表示前面第 n 个捕获的子模式，n 是一个数字。 • %%：匹配百分号 % 本身。 • replaceString：用于替换找到的子串的字符串；
返回值	• newString：替换后的字符串。

代码 2-52 string.gsub 示例

```
1.  local mainString = "Hello, FR! Hello, Lua!"
2.  local findString = "Hello"
3.  local replaceString = "Hi"
```

代码 2-52（续）

```
4.  -- 替换所有匹配的子串
5.  local newString = string.gsub(mainString, findString, replaceString)
6.  print(newString)  -- 输出 "Hi, FR! Hi, Lua!"
```

4）string.find：在字符串中搜索子串。

表 2-7 string. find 的详细参数

属性	说明
原型	local start, end_ = string.find (str, substr, init, plain)
描述	在字符串中搜索子串，并返回子串的起始和结束索引。如果找到子串，它返回子串在字符串中的起始和结束位置；如果没有找到，它返回 nil
参数	• str：要搜索的字符串; • substr：要查找的子串。 • init：搜索的起始位置，默认为 1。如果指定，搜索将从这个位置开始。 • plain：如果设置为 true，则搜索将使用普通字符串比较，而不是模式匹配。默认为 false。
返回值	start：子串在字符串中的起始索引（基于 1 的索引）; end_：子串在字符串中的结束索引（基于 1 的索引）。

代码 2-53 string.find 示例

```
1.  local str = "Hello, FR
2.  local substr = "world"
3.
4.  -- 查找子串 "FR" 的位置
5.  local start, end_ = string.find(str, substr)
6.  print(start, end_)  -- 输出 8 9
7.
8.  -- 从指定位置开始查找
9.  local start, end_ = string.find(str, substr, 6)
10. print(start, end_)  -- 输出 8 9
11.
12. -- 使用普通字符串比较
13. local start, end_ = string.find(str, "FR", 1, true)
14. print(start, end_)  -- 输出 8 9
```

5）string.reverse：将字符串反转。

表 2-8 string.reverse 的详细参数

属性	说明
原型	local reversed_str = string.reverse (arg)
描述	用于将字符串反转
参数	• arg：需要反转的字符串;
返回值	reversed_str：反转后的字符串

代码 2-54 string.reverse 示例

```
1.  local original_str = "Hello, World!"
2.  local reversed_str = string.reverse(original_str)
3.  print(reversed_str)  -- 输出 "!dlroW ,olleH"
```

6） string.format：函数用于创建格式化的字符串。

表 2-9 string.format 的详细参数

属性	说明
原型	local reversed_str = string.format(format, arg1, arg2, ...)
描述	用于创建格式化的字符串
参数	• format：一个包含格式化指令的字符串，这些指令由 % 符号开始，后跟一个或多个字符来指定格式; • %d 或 %i：整数; • %f：浮点数; • %g：根据数值的大小自动选择 %f 或 %e; • %e 或 %E：科学计数法表示的浮点数; • %x 或 %X：十六进制整数; • %o：八进制整数; • %p：指针（通常显示为十六进制数）; • %s：字符串; • %q：双引号包围的字符串，用于程序输出; • %c：字符; • %b：二进制数; • %%：输出 % 符号; • arg1, arg2, ...：要插入到格式化字符串中的参数。
返回值	• reversed_str：反转后的字符串。

代码 2-55 string.format 示例

```
1.  -- 格式化数字
2.  local num = 123
3.  local formatted_num = string.format("Number: %d", num)
4.  print(formatted_num)  -- 输出 "Number: 123"
5.
```

代码 2-55（续）

```
-- 格式化浮点数
local pi = 3.14159
local formatted_pi = string.format("Pi: %.2f", pi)
print(formatted_pi)  -- 输出 "Pi: 3.14"

-- 格式化字符串
local name = "Kimi"
local greeting = string.format("Hello, %s!", name)
print(greeting)  -- 输出 "Hello, Kimi!"

-- 格式化多个值
local name = "Kimi"
local age = 30
local greeting = string.format("Hello, %s. You are %d years old.", name, age)
print(greeting)  -- 输出 "Hello, Kimi. You are 30 years old."

-- 格式化为十六进制
local num = 255
local formatted_hex = string.format("Hex: %x", num)
print(formatted_hex)  -- 输出 "Hex: ff"

-- 格式化为科学计数法
local large_num = 123456789
local formatted_scientific = string.format("Scientific: %e", large_num)
print(formatted_scientific)  -- 输出 "Scientific: 1.234568e+8"
```

7）string.char：将一个或多个整数参数转换为对应的字符串

表 2-10 string.char 的详细参数

属性	说明
原型	local str = string.char(arg1, arg2, ...)
描述	将一个或多个整数参数转换为对应的字符串，其中每个整数都表示一个字符的 ASCII 或 Unicode 编码
参数	• arg1, arg2, ...：要转换为字符的整数序列;
返回值	• str：由转换后的字符组成的字符串。

代码 2-56 string.char 示例

```
local str = string.char(72, 101, 108, 108, 111)
print(str)  -- 输出 "Hello"
```

8）string.byte：将字符串中的一个或多个字符转换整数。

表 2-11 string.byte 的详细参数

属性	说明
原型	local byte1, byte2 = string.byte (s, i, j)
描述	将字符串中的一个或多个字符转换为它们对应的 ASCII 或 Unicode 编码的整数。
参数	• s：要转换的字符串。 • i：字符串中要转换的第一个字符的位置，默认为 1； • j：字符串中要转换的最后一个字符的位置，默认为 i。
返回值	• byte1, byte2：由转换后的字符对应的编码值。

代码 2-57 string.byte 示例

```
1.  local str = "Hello"
2.  local byte1 = string.byte(str, 1)
3.  print(byte1)  -- 输出 72，即 'H' 的 ASCII 码
4.  local bytes = string.byte(str, 1, 5)
5.  for i, v in ipairs(bytes) do
6.      print(i, v)  -- 输出字符 'H', 'e', 'l', 'l', 'o' 的 ASCII 码
7.
8.  end
```

9）string.len：计算字符串的长度。

表 2-12 string.len 的详细参数

属性	说明
原型	local length = string.len (arg)
描述	函数用于计算字符串的长度，即字符串中包含的字符数。
参数	• arg：要计算长度的字符串。
返回值	• length：字符串的长度，即字符串中字符的数量。

代码 2-58 string.len 示例

```
1.  local str = "Hello, FR!"
2.  local length = string.len(str)
3.  print(length)
4.  -- 输出 11
```

10） string.rep：复制字符串。

表 2-13 string.rep 的详细参数

属性	说明
原型	local repeated_str = string.rep (string, n)
描述	用于重复字符串指定的次数，即将一个字符串复制多次并连接起来。

表 2-13（续表）

属性	说明
参数	• string：要重复的原始字符串； • n：重复的次数，即原始字符串需要被复制并连接的次数。
返回值	repeated_str：重复后的字符串。

代码 2-59 string.rep 示例

```
1.  local str = "FR "
2.  local n = 3
3.  local repeated_str = string.rep(str, n)
4.  print(repeated_str)  -- 输出 "FR FR FR "
```

11） string.match：在字符串 str 中搜索与指定模式 pattern 相匹配的子串。

表 2-14 string.match 的详细参数

属性	说明
原型	local match_result = string.match (str, pattern，init)
描述	函数用于在给定的字符串 str 中搜索与指定模式 pattern 相匹配的子串。
参数	• str：要搜索的字符串； • pattern：定义搜索模式的字符串，可以包含特殊的模式匹配字符； • init：搜索的起始位置，默认为 1。如果指定，搜索将从此位置开始。
返回值	match_result：如果找到匹配的子串，返回匹配的字符串。如果模式中定义了捕获（用括号括起来的子模式），则返回这些捕获的值。如果没有找到匹配，返回 nil。

代码 2-60 string.match 示例

```
1.  local text = "Hello, 1234 world!"
2.  local pattern = "%d+"  -- 匹配一个或多个数字
3.  local start = 1
4.  -- 匹配数字
5.  local match = string.match(text, pattern, start)
6.  print(match)  -- 输出 "1234"
7.  -- 匹配并返回
8.  local pattern_with_capture = "(%d+) world"
9.  local match, number = string.match(text, pattern_with_capture, start)
10. print(match)  -- 输出 "1234 world"
11. print(number)  -- 输出 "1234"
```

12） string.sub：从一个字符串中提取子字符串。

表 2-15 string.sub 的详细参数

属性	说明
原型	local sub_result = string.sub (s, i, j)
描述	从给定的字符串 s 中截取子串。它根据指定的起始位置 i 和可选的结束位置 j 来确定要截取的子串范围。
参数	• s：要截取子串的原始字符串； • i：子串开始的索引位置可以为负数，表示从字符串的末尾开始计算； • j：结束位置（可选），也可以为负数，表示从字符串的末尾开始计算。如果省略 j，则默认截取到字符串的末尾。
返回值	sub_result：截取的子串。

代码 2-61 string.sub 示例

```
1.  --简单截取
2.  local text = "FR3 Robotics"
3.  local subText = string.sub(text, 1, 3)
4.  print(subText)  -- 输出: FR3
5.
6.  --从起始位置截取
7.  local text = "FRs FR5"
8.  local subText = string.sub(text, 5)
9.  print(subText)  -- 输出: FR5
10.
11. --使用负数索引从末尾截取
12. local text = "Welcome to FR10"
13. local subText = string.sub(text, -4, -1)
14. print(subText)  -- 输出: FR10
```

2.5 数组

在 FRLua 中，数组是通过 table 类型来实现的，实际上，FRLua 中并没有专门的数组类型，但可以使用 table 作为数组来处理元素。数组的索引一般从 1 开始，而不是像其他语言中从 0 开始。你可以用 {}来创建一个空的数组，并可以在其中存放各种类型的元素。

2.5.1 一维数组

1）创建数组

代码 2-62 创建数组示例

```
1.  -- 创建一个空数组
2.  local array = {}
3.  -- 初始化数组
4.  local robotModels = {"FR3", "FR5", "FR10", "FR20"}
```

2） 访问数组元素:通过索引来访问数组元素，索引从 1 开始。

代码 2-63 访问数组元素示例

```
1.  -- 访问数组元素
2.  print(robotModels [1])
3.  -- 输出: FR3
4.  print(robotModels [3])
5.  -- 输出: FR10
```

3）修改数组元素:可以通过索引来修改数组中的元素。

代码 2-64 修改数组元素示例

```
1.  -- 修改数组中的某个元素
2.  robotModels [2] = "FE5_1"
3.  print(robotModels [2])  -- 输出: FE5_1
```

4）数组的长度：FRLua 提供了 # 操作符来获取数组的长度。

代码 2-65 获取数组长度的示例

```
1.  -- 获取数组长度
2.  print(#robotModels)
3.  -- 输出: 4
```

5） 数组动态增长

FRLua 的 table（数组）长度不是固定的，可以随时向其中添加新元素，数组会自动增长。

代码 2-66 数组动态增长示例

```
1.  -- 动态添加新元素
2.  robotModels[#robotModels + 1] = "FR20_1"
3.  print(robotModels[5])  -- 输出: FR20_1
```

1.5.2 多维数组

在 FRLua 中，多维数组是通过嵌套的 table 来实现的，也就是数组中的每个元素本身也是一个数组。通过这种方式，可以创建二维数组、三维数组，甚至更高维度的数组。

1）创建多维数组

要创建一个二维数组，可以将每一行的数据存储在一个独立的 table 中，然后将这些行的 table 存储在一个更大的 table 中。

下面是一个简单的二维数组的示例，它存储了不同的机器人型号与其参数。

代码 2-67 二维数组示例

```
-- 创建一个二维数组
local robots = {
    {"FR3", 3, "Lightweight"},
    {"FR5", 5, "Standard"},
    {"FR10", 10, "Heavy-duty"}
}
-- 访问二维数组中的元素
print(robots[1][1])  -- 输出: FR3
print(robots[2][2])  -- 输出: 5
print(robots[3][3])  -- 输出: Heavy-duty
--[[
robots[1] 表示二维数组的第一行，即 { "FR3", 3, "Lightweight" }。
robots[1][1] 表示二维数组的第一行的第一列，即 "FR3"。
robots[2][2] 表示二维数组的第二行的第二列，即 5。
]]
```

2）遍历二维数组

可以使用嵌套循环来遍历多维数组。以下示例演示如何遍历 robots 这个二维数组。

代码 2-68 遍历二维数组示例

```
for i = 1, #robots do
    for j = 1, #robots[i] do
        print(robots[i][j])
    end
end
--[[
输出：
FR3
3
Lightweight
FR5
5
Standard
FR10
10
Heavy-duty]]
```

3）多维数组的动态增长

在 FRLua 中可以动态地向多维数组中添加新的行或列。

代码 2-69 在二维数组中添加新行示例

```
-- 添加新的机器人型号
table.insert(robots, {"FR20", 20, "Super Heavy-duty"})
print(robots[4][1])  -- 输出: FR20

--示例：向现有行添加新列

-- 向每一行添加一个新的元素
for i = 1, #robots do
    table.insert(robots[i], "Available")
end

print(robots[1][4])  -- 输出: Available
print(robots[4][4])  -- 输出: Available
```

三维或更高维度的数组也可以通过进一步嵌套 table 来实现。

2.6 表

在 FRLua 中，table 是一种强大且灵活的数据结构。它可以用来表示数组、字典、集合以及其他复杂的数据类型。由于 FRLua 中没有内置的数组或对象系统，table 是 FRLua 最核心的数据结构之一。

2.6.1 Table 的基本用法

table 是一个关联数组（Associative Array），它使用任意类型的键（但不能是 nil）来索引。也就是说，既可以用数字，也可以用字符串等类型作为键值。

代码 2-70 table 索引示例

```
--示例 1：使用数字作为索引
local fruits = {"FR3", "FR5", "FR10"}
print(fruits[1])  -- 输出: FR3

--示例 2：使用字符串作为索引
local person = {name = "FR", age = 5}
print(person["name"])  -- 输出: FR
print(person.age)  -- 输出: 30 （等价写法）
```

2.6.2 Table 操作函数

在 FRLua 中，table 模块提供了一些常用的函数来操作表格数据。

表 2-16 table 模块常用函数详细参数

序号	方法 & 用途
1	table.concat (table , sep , start , end): 从 start 位置到 end 位置的所有元素,以指定的分隔符(sep)隔开，并重新连接。
2	table.insert (table, pos, value): 在 table 的数组部分指定位置插入值元素
3	table.remove (table , pos) 返回 table 数组部分位于 pos 位置的元素
4	table.sort (table , comp) 对给定的 table 进行升序排序。

下面是这些函数的详细说明和使用示例：

1）table.concat：连接表中的字符串

表 2-17 table.concat 的详细参数

属性	说明
原型	local concatenated = table.concat (table , sep , start , end)
描述	此函数用于连接表中的字符串，并可以指定分隔符 sep，以及从表的第 start 项到第 end 项进行连接。
参数	• table：一个包含要连接的字符串元素的表； • sep：连接字符串时使用的分隔符。如果未提供或为 nil，则不使用 • 分隔符。默认值为 nil。 • start：指定从表中哪个索引开始连接。默认值为 1，即从表的第一个元素开始。 • end：指定连接到表中的哪个索引结束。如果未提供或为 nil，则连接到表的末尾。
返回值	concatenated：连接后的字符串。

代码 2-71 table.concat 示例

```
1.  local FRuser = {" Welcome ", " to ", " FR "}
2.
3.  local result = table.concat(FRuser, " ")
4.
5.  print(result)
6.  -- 输出: Welcome to FR
```

2） table.insert：在表（table）的指定位置插入一个值

表 2-18 table.insert 的详细参数

属性	说明
原型	table.insert (table, pos, value)
描述	用于在表（table）的指定位置插入一个值
参数	• table：要插入新元素的表； • pos：插入新元素的位置的索引。如果 pos 等于表的长度加一，新元素将被添加到表的末尾。如果 pos 大于表的长度，新元素将被插入到表的末尾，并且表的长度将增加； • value：要插入的新元素的值。
返回值	无

代码 2-72 table.insert 示例

```
1.  local robots = {"FR3", "FR5"}
2.
3.  -- 在第二个位置插入
4.  table.insert(robots, 2, "FR10")
5.  print(robots [2])  -- 输出: FR10
```

3） table.remove：从表中删除的元素

表 2-19 table.remove 的详细参数

属性	说明
原型	local removed = table.remove (table, pos)
描述	从表中删除指定位置的元素，若不指定位置 pos，删除最后一个元素。
参数	• table：要移除元素的表； • pos：要移除元素的位置索引。默认值为 nil，表示移除最后一个元素。如果 pos 大于表的长度，将返回 nil 且表不会改变。
返回值	removed：被移除的元素的值。如果没有指定 pos 或 pos 超出范围，则返回 nil。

代码 2-73 table.remove 示例

```
1.  local robots = {"FR3", "FR5", "FR10"}
2.  table.remove(robots, 2)
3.
4.   -- 移除第二个位置的元素
5.  print(robots [2])
6.  -- 输出: FR10
```

4）table.sort：对表中的元素进行排序。

表 2-20 table.sort 的详细参数

属性	说明
原型	table.sort (table, comp)
描述	对表中的元素进行排序。如果提供了 comp 函数，则使用自定义的比较函数来确定排序顺序。
参数	• table：要排序的表； • comp：一个比较函数，用于确定两个元素的顺序。这个函数接收两个参数（通常是来自表的元素），并返回一个布尔值。如果第一个参数应该在第二个参数之前，则返回 true；否则返回 false。
返回值	无

代码 2-74 table.sort 示例

```
1.  local numbers = {5, 2, 9, 1, 7}
2.  table.sort(numbers)
3.  for i, v in ipairs(numbers) do
4.      print(v)
5.  end
6.  -- 输出: 1 2 5 7 9
7.
8.  --示例：自定义排序
9.  local numbers = {5, 2, 9, 1, 7}
10. table.sort(numbers, function(a, b) return a > b end)  -- 降序排序
11. for i, v in ipairs(numbers) do
12.     print(v)
13. end
14. -- 输出: 9 7 5 2 1
```

2.7 协同程序

在 FRLua 中协同程序是一种与线程类似的编程结构，但与传统的线程不同，协同程序拥有更精细的控制流。它允许在一个函数的执行过程中暂停并在稍后恢复。协同程序具有独立的堆栈、局部变量和指令指针，但它们共享全局变量和其它资源。因此，协同程序在非抢占式多任务处理的场景中非常适用。

协同程序的特性：

独立性：每个协同程序都有独立的执行环境，具有自己独立的局部变量和堆栈。

共享性：协同程序之间共享全局变量。

非抢占式：协同程序的执行可以通过手动控制，如暂停 (yield) 和恢复 (resume)。

灵活的多任务处理：协同程序可以在合适的时候暂停自己并恢复其他协同程序。

协同程序的相关操作由 coroutine 模块提供支持，表 2-21 为协同程序常用函数：

表 2-21 协同程序常用函数

序号	方法 & 用途
1	coroutine.create(f) 创建一个新的协同程序，f 是启动该协同程序时执行的函数。
2	coroutine.resume (co [, val1, ...]) 恢复协程 co 并传递参数（如果有）。
3	coroutine.yield (...) 暂停协同程序的执行，返回到调用它的地方
4	coroutine.status (co) 查询协同程序的状态
5	coroutine.wrap(f) 创建协同程序，返回一个函数，调用该函数会运行协同程序
6	coroutine.running() 返回正在运行的协同程序

示例 1：创建与恢复协同程序

代码 2-75 创建与恢复协同程序

```
1.  -- 创建一个协同程序，打印"协同程序开始"并使用 yield 暂停
2.  co = coroutine.create(function()
3.      print("协同程序开始")
4.      coroutine.yield()  -- 暂停协同程序
5.      print("协同程序继续")
6.  end)
7.
8.  -- 启动协同程序
9.  coroutine.resume(co)    -- 输出: 协同程序开始
10.
11. -- 恢复协同程序
12. coroutine.resume(co)    -- 输出: 协同程序继续
```

示例 2：协同程序间传递数据

代码 2-76 协同程序间传递数据

```
co = coroutine.create(function(a, b)
        print("协同程序执行", a, b)
        local x = coroutine.yield(a + b)  -- 暂停，并返回结果
        print("恢复协同程序", x)
end)
-- 启动协同程序，传入参数 3 和 7，输出: 协同程序执行 3 7
-- yield 返回 a + b 的结果：10
print(coroutine.resume(co, 3, 7))  -- 输出: true 10
-- 恢复协同程序并传入新参数 15，输出: 恢复协同程序 15
coroutine.resume(co, 15)
```

示例 3：使用 coroutine.wrap 简化调用

代码 2-77 使用 coroutine.wrap 简化调用

```
-- wrap 返回一个函数来直接调用协同程序
wrapped = coroutine.wrap(function()
        print("协同程序开始")
        coroutine.yield()
        print("协同程序继续")
end)
-- 调用 wrapped 相当于 coroutine.resume(co)
wrapped()  -- 输出: 协同程序开始
wrapped()  -- 输出: 协同程序继续
```

示例 4：协同程序的状态，使用 coroutine.status(co) 可以查询协同程序的当前状态：

running: 协同程序正在运行。

suspended: 协同程序已暂停或从未启动。

dead: 协同程序执行完毕或运行时发生错误。

代码 2-78 协同程序的状态

```
--状态查询
co = coroutine.create(function()
        print("协同程序")
end)
print(coroutine.status(co))     -- 输出: suspended
coroutine.resume(co)            -- 启动协同程序
print(coroutine.status(co))     -- 输出: dead
```

2.8 文件操作

FRLua 中文件 I/O 用于读取和操作文件。文件 I/O 可以分为两种模式：简单模式和完全模式。

2.8.1 简单模式

简单模式类似于 C 语言的文件 I/O 操作，它拥有一个当前的输入文件和一个当前的输出文件，并提供针对这些文件的操作。适用于简单的文件操作

利用 io.open 函数打开文件其中 mode 的值如表 2-22 所示。

表 2-22 mode 的值

模式	描述
r	以只读方式打开文件，该文件必须存在。
w	打开只写文件，若文件存在则文件长度清为 0，即该文件内容会消失。若文件不存在则建立该文件。
a	以附加的方式打开只写文件。若文件不存在，则会建立该文件，如果文件存在，写入的数据会被加到文件尾，即文件原先的内容会被保留。（EOF 符保留）
r+	以可读写方式打开文件，该文件必须存在。
w+	打开可读写文件，若文件存在则文件长度清为零，即该文件内容会消失。若文件不存在则建立该文件。
a+	与 a 类似，但此文件可读可写
b	二进制模式，如果文件是二进制文件，可以加上 b
+	号表示对文件既可以读也可以写

简单模式使用标准的 I/O 或使用一个当前输入文件和一个当前输出文件。

例如有一个文件名为 example.txt，进行读取文件操作

代码 2-79 对文件进行读取操作

```
1.  -- 以只读方式打开文件
2.  file = io.open("example.txt", "r")
3.
4.  -- 设置默认输入文件为 example.txt
5.  io.input(file)
6.
7.  -- 输出文件第一行
8.  print(io.read())
9.
10. -- 关闭打开的文件
```

代码 2-79（续）

```
11.  io.close(file)
12.
13.  -- 以附加的方式打开只写文件
14.  file = io.open("example.txt", "a")
15.
16.  -- 设置默认输出文件为 example.txt
17.  io.output(file)
18.
19.  -- 在文件最后一行添加 Lua 注释
20.  io.write("--  example.txt 文件末尾注释")
21.
22.  -- 关闭打开的文件
23.  io.close(file)
```

在以上实例中使用了 io."x" 方法，其中 io.read() 中没有带参数，参数可以是表 2-23 中的一个：

表 2-23 io.read()的参数

模式	描述
"*n"	读取一个数字并返回它。例：file.read("*n")
"*a"	从当前位置读取整个文件。例：file.read("*a")
"*l"（默认）	读取下一行，在文件尾 (EOF) 处返回 nil。例：file.read("*l")
number	返回一个指定字符个数的字符串，或在 EOF 时返回 nil。例：file.read(5)

其他的 io 方法有：

io.tmpfile():返回一个临时文件句柄，该文件以更新模式打开，程序结束时自动删除

io.type(file): 检测 obj 是否一个可用的文件句柄

io.flush(): 向文件写入缓冲中的所有数据

io.lines(optional file name): 返回一个迭代函数，每次调用将获得文件中的一行内容，当到文件尾时，将返回 nil，但不关闭文件。

2.8.2 完全模式

完全模式使用文件句柄（file handle）来进行文件操作，以面向对象的形式将所有文件操作定义为文件句柄的方法。适合进行更复杂的文件操作，例如同时读取多个文件。同样以 example.txt 文件为例进行文件操作。

代码 2-80 完全模式对文件进行操作

```
--1. 读取整个文件内容：
local file = io.open("example.txt", "r")
local content = file:read("*a")  -- 读取整个文件内容
print(content)
file:close()

--2.逐行读取文件：
local file = io.open("example.txt", "r")
for line in file:lines() do
    print(line)
end
file:close()

--3.读取一行
local file = io.open("example.txt", "r")
local line = file:read("*l")  -- 读取一行
print(line)
file:close()

--对 example.txt 文件，进行写入文件操作
--1. 写入字符串：
local file = io.open("example.txt", "w")
file:write("Hello, Lua!\n")
file:close()

--2.追加写入，向文件末尾追加内容：
local file = io.open("example.txt", "a")
file:write("This is appended text.\n")
file:close()
```

2.9 错误处理

FRLua 提供了一种简单且灵活的错误处理机制。通过适当的错误处理机制，程序可以捕获并处理错误，而不是在遇到错误时直接终止执行。

2.9.1 语法错误

语法错误是在程序编写时发生的错误，它们违反了编程语言的语法规则。这类错误通常在代码编译或解释执行之前被识别出来。对于解释型语言如 FRLua，

这类错误通常在代码执行时立即被检测到。

常见的语法错误包括：

拼写错误：变量名、函数名或关键字的拼写错误。

括号不匹配：例如，缺少闭合的括号、大括号或方括号。

错误的符号使用：如使用错误的运算符或在不适当的上下文中使用运算符。

语法结构错误：如在不适当的位置使用语句，或者语句的格式不正确。

代码 2-81 FRLua 错误错误代码示例

```
--拼写错误：变量名、函数名或关键字的拼写错误。
local varialble = "This is a typo in the variable name"  -- 应该是 variable 而不是 varial

--括号不匹配：例如，缺少闭合的括号、大括号或方括号。
 local array = {1, 2, 3  -- 缺少一个闭合的花括号

-- 错误的符号使用：如使用错误的运算符或在不适当的上下文中使用运算符。
local sum = 1 + {2, 3}  -- 括号内是数组，不能与数字直接相加

--语法结构错误：如在不适当的位置使用语句，或者语句的格式不正确。
functon myFunction()  -- 应该是 function 而不是 functon
    print("Hello, world!")
end

myFunction )  --缺少调用函数的括号，应该是 myFunction()

--错误的控制流结构
while true  --缺少 do 关键字
    print("Infinite loop!")
end
// 错误的注释---这是一个错误的注释，因为它没有正确关闭
```

代码 2-82 FRLua 正确代码示例

```
-- 正确的 Lua 代码

local variable = "This is a correct variable name"

local array = {1, 2, 3}

local sum = 1 + 2 + 3  -- 正确的运算符使用
```

代码 2-82（续）

```
8.
9.    function myFunction()
10.         print("Hello, world!")
11.   end
12.
13.   myFunction()   -- 正确的函数调用
14.   while true do
15.      print("Infinite loop!")
16.   end
17.
18.   -- 正确的注释
```

2.9.2 运行时错误

运行时错误是在程序执行过程中发生的错误，通常是因为程序的逻辑问题或外部因素（如资源不可用）。这类错误只有在程序运行时才会显现，因为它们与程序的执行状态和输入数据有关。

常见的运行时错误包括：

逻辑错误：程序按照预期执行，但结果不是预期的，通常是因为算法或逻辑处理不正确。

类型错误：尝试对错误类型的数据执行操作，例如尝试将字符串传递给需要数字的函数。

资源错误：如文件不存在、网络连接失败或内存不足。

访问违规：如访问数组的非法索引或尝试修改常量。

编写正确但会出现运行错误的代码示例如代码 2-80 所示：

代码 2-83 FRLua 运行错误代码示例

```
1.    -- 假设这是一段 Lua 代码
2.
3.    -- 逻辑错误：程序按照预期执行，但结果不是预期的
4.    function calculateArea(width, height)
5.        return width * height    -- 错误地将矩形的周长计算为面积
6.    end
7.    local area = calculateArea(10, 5)
8.    print("Area:", area)    -- 期望得到面积，但得到的是周长
9.
```

代码 2-83（续）

```
-- 类型错误：尝试对错误类型的数据执行操作
function addNumbers(a, b)
    return a + b
end
local result = addNumbers("10", 5)  -- 期望数字相加，但第一个参数是字符串

-- 资源错误：如文件不存在
function readFile(filename)
    local file = io.open(filename, "r")
    if not file then
        print("Error: File does not exist.")
        return
    end
    local content = file:read("*a")
    file:close()
    return content
end
local content = readFile("nonexistentfile.txt")

-- 访问违规：如访问数组的非法索引
local array = {1, 2, 3}
local value = array[4]  -- 尝试访问不存在的数组索引

-- 尝试修改常量
local constant = 10
constant = 20  -- 尝试修改常量值
```

代码 2-84 FRLua 运行错误代码修改成正确后的代码示例

```
-- 正确的 Lua 代码

-- 逻辑错误修正：正确计算面积
function calculateArea(width, height)
    return width * height
end
local area = calculateArea(10, 5)
print("Area:", area)  -- 正确得到面积

-- 类型错误修正：确保参数类型正确
function addNumbers(a, b)
    assert(tonumber(a) and tonumber(b), "Both arguments must be numbers")
    return tonumber(a) + tonumber(b)
```

代码 2-84（续）

```
14.  end
15.  local result = addNumbers(10, 5)   -- 正确地将两个数字相加
16.
17.  -- 资源错误处理：正确处理文件不存在的情况
18.  function readFile(filename)
19.      local file = io.open(filename, "r")
20.      if not file then
21.          print("Error: File does not exist.")
22.          return nil
23.      end
24.      local content = file:read("*a")
25.      file:close()
26.      return content
27.  end
28.  local content = readFile("nonexistentfile.txt") or "File content not available"
29.
30.  -- 访问违规修正：正确处理数组索引
31.  local array = {1, 2, 3}
32.  local value = array[1]   -- 访问存在的数组索引
33.  -- 常量保护：不修改常量值
34.  local constant = 10
35.  -- constant = 20   -- 不修改常量值
```

2.9.3 错误处理

FRLua 中的错误处理主要通过以下两种机制来实现：

error() 函数：用于手动抛出错误。

pcall() 和 xpcall() 函数：用于捕获和处理错误。

1）使用 error() 抛出错误

FRLua 提供了 error() 函数，用于在程序中手动抛出一个错误。当 error() 被调用时，程序会终止当前函数的执行，并返回给调用者一个错误信息。

代码 2-85 FRLua 使用 error() 抛出错误代码示例

```
1.  function divide(a, b)
2.      if b == 0 then
3.          error("除数不能为零")
4.      else
5.          return a / b
```

代码 2-85（续）

```
6.          end
7.    end
8.
9.    print(divide(10, 2))    -- 输出: 5
10.   print(divide(10, 0))    -- 错误: 除数不能为零
```

2）使用 pcall() 捕获错误

pcall()（protected call）用于捕获在 FRLua 代码块中发生的错误。pcall() 函数会执行给定的函数，如果没有错误发生，它返回 true 和函数的返回值；如果有错误发生，它返回 false 和错误信息。

代码 2-86 FRLua 使用 pcall() 捕获错误代码示例

```
1.    function divide(a, b)
2.          if b == 0 then
3.                error("除数不能为零")
4.          else
5.                return a / b
6.          end
7.    end
8.
9.    print(divide(10, 2))    -- 输出: 5
10.   print(divide(10, 0))    -- 错误: 除数不能为零 function divide(a, b)
11.   if b == 0 then
12.         error("除数不能为零")
13.   else
14.         return a / b
15.   end
16.
17.   -- 使用 pcall 捕获错误
18.   status, result = pcall(divide, 10, 2)
19.   print(status, result)    -- 输出: true 5
20.
21.   status, result = pcall(divide, 10, 0)
22.   print(status, result)    -- 输出: false 除数不能为零
```

3）使用 xpcall() 捕获错误并指定错误处理函数

xpcall() 是 pcall() 的扩展，允许在捕获到错误时指定一个错误处理函数。这样可以自定义错误处理逻辑。

代码 2-87 FRLua xpcall() 捕获错误并指定错误处理代码示例

```
function divide(a, b)
    if b == 0 then
        error("除数不能为零")
    else
        return a / b
    end
end

-- 定义一个错误处理函数
function errorHandler(err)
    print("错误发生: " .. err)
end

-- 使用 xpcall 捕获错误并调用错误处理函数
status = xpcall(divide, errorHandler, 10, 0)  -- 输出: 错误发生: 除数不能为零
```

2.10 模块

模块是 FRLua 中用于组织代码的一种机制，提供了更好地封装和重用代码的方式。在大型应用程序中，使用模块可以让代码结构更加清晰，也方便维护和扩展。

2.10.1 创建模块

FRLua 中使用表来创建模块，直接定义一个 table，并返回这个 table 作为模块。这个方法更加直观和清晰。

代码 2-88 使用表创建模块

```
-- 使用表创建模块
-- robot_module.lua
local robot_module = {}

robot_module.version = "1.0"

function robot_module.greet()
    print("欢迎使用法奥协作机器人")
end

return robot_module
```

2.10.2 模块调用

FRLua 使用 require 函数来加载模块。require 会执行模块文件并返回模块的表。模块通常会保存在与 FRLua 脚本相同的目录中，或者配置在 LUA_PATH 环境变量指定的路径中。

require 函数只会在第一次加载模块时执行一次，并将结果缓存起来。如果你多次调用 require 同一个模块，它只会返回第一次加载的模块的表，而不会重新加载该模块。这种行为有助于提高性能，避免重复加载和执行。

代码 2-89 记载已创建的模块

```
1.  -- 加载模块
2.  local robot = require "robot_module"
3.
4.  -- 使用模块中的函数
5.  robot.greet()-- 输出：欢迎使用法奥协作机器人
```

2.10.3 搜索路径

FRLua 使用一个搜索路径来查找模块文件。这个路径可以通过 package.path 变量来设置。路径是一个包含模式的字符串，FRLua 会在这些模式指定的目录中查找模块文件。可以在脚本中设置 package.path 来指定模块的搜索路径：

代码 2-90 记载已创建的模块

```
1.  package.path = package.path .. ";/path/to/modules/?.lua"
2.  local mathlib = require("robot_module ")
```

3 FRLua 脚本预置函数

3.1 逻辑指令

3.1.1 循环

详情请见 2.2.2 节。

3.1.2 等待

WaitMs：等待指定时间

表 3-1 WaitMs 详细参数

属性	说明
原型	WaitMs(t_ms)
描述	等待指定时间
参数	• t_ms:单位[ms]。
返回值	无

WaitDI：等待控制箱数字量输入

表 3-2 WaitDI 详细参数

属性	说明
原型	WaitDI (id, status, maxtime, opt)
描述	等待控制箱数字量输入
参数	• id:控制箱 DI 端口编号，0-7 控制箱 DI0-DI7，8-15 控制箱 CI0-CI7； • status:0-False，1-True； • maxtime:最大等待时间，单位[ms]； • opt:超时后策略，0-程序停止并提示超时，1-忽略超时提示程序继续执行，2-一直等待。
返回值	无

WaitToolDI：等待工具数字量输入

表 3-3 WaitToolDI 详细参数

属性	说明
原型	WaitToolDI (id, status, maxtime, opt)
描述	等待控制箱数字量输入

表 3-3（续表）

属性	说明
参数	• id:工具 DI 端口编号，0 - End-DI0，1 - End-DI1； • status:0-False，1-True； • maxtime:最大等待时间，单位[ms]； • opt:超时后策略，0-程序停止并提示超时，1-忽略超时提示程序继续执行，2-一直等待。
返回值	无

WaitMultiDI：等待控制箱多路数字量输入

表 3-4 WaitMultiDI 详细参数

属性	说明
原型	WaitMultiDI (mode, id, status, maxtime, opt)
描述	等待控制箱多路数字量输入
参数	• mode:[0]-多路与，[1]-多路或； • id:io 编号，bit0~bit7 对应 DI0~DI7，bit8~bit15 对应 CI0~CI7； • status:bit0~bit7 对应 DI0~DI7 状态，bit8~bit15 对应 CI0~CI7 状态:0-False，1-True； • maxtime:最大等待时间，单位[ms]； • opt:超时后策略，0-程序停止并提示超时，1-忽略超时提示程序继续执行，2-一直等待。
返回值	无

WaitAI：等待控制箱模拟量输入

表 3-5 WaitAI 详细参数

属性	说明
原型	WaitAI (id, sign, value, maxtime, opt)
描述	等待控制箱模拟量输入
参数	• id:io 编号，范围[0~1]； • sign:0-大于，1-小于 • value:输入电流或电压值百分比，范围[0~100]对应电流值[0~20mA]或电压[0~10V]； • maxtime:最大等待时间，单位[ms]； • opt:超时后策略，0-程序停止并提示超时，1-忽略超时提示程序继续执行，2-一直等待。
返回值	无

WaitToolAI：等待工具模拟量输入

表 3-6 WaitToolAI 详细参数

属性	说明
原型	WaitToolAI (id, sign, value, maxtime, opt)
描述	等待工具模拟量输入
参数	• id:io 编号，0 - End-AI0； • sign:0 - 大于，1 - 小于; • value:输入电流或电压值百分比，范围[0~100]对应电流值[0~20mA]或电压[0~10V]； • maxtime:最大等待时间，单位[ms]； • opt:超时后策略，0-程序停止并提示超时，1-忽略超时提示程序继续执行，2-一直等待。
返回值	无

代码 3-1 等待指令示例

```
1.  --等待
2.  WaitMs(1000)--等待指定时间 1000ms
3.
4.  --等待控制箱数字量输入
5.  WaitDI(1,1,0,1)--端口号：Ctrl-DI1，状态：true(开)，最大等待时间：1000ms，等待超时后策略：忽略超时提示程序继续执行
6.
7.  --等待工具数字量输入
8.  WaitToolDI(1,1,0,1)--端口号：End-DI0, 状态：true(开)，最大等待时间：1000ms，等待超时后策略：忽略超时提示程序继续执行
9.
10. --等待控制箱多路数字量输入
11. WaitMultiDI(0,3,1,1000,0)--多路与，IO 端口编号:DI0 和 DI1，DI0 开，DI1 关，最大等待时间：1000ms，超时后策略：程序停止并提示超时。
12. WaitMultiDI(1,3,3,1,0)--多路或，IO 端口编号:DI0 和 DI1，DI0 开，DI1 开，最大等待时间：1000ms，超时后策略：程序停止并提示超时。
13.
14. --等待控制箱模拟量输入
15. WaitAI(0,0,20,1000,0)--IO 端口：控制箱 AI0，条件：<，数值：20，最大等待时间 ：1000ms,超时后策略：程序停止并提示超时。
16.
17. --等待工具模拟量输入
18. WaitToolAI(0,0,20,1000,0)--IO 端口：控制箱 End-AI0，条件：<，数值：20，最大等待时间：1000ms, 超时后策略：程序停止并提示超时。
```

3.1.3 暂停

Pause：暂停

表 3-7 Pause 详细参数

属性	说明
原型	Pause (num)
描述	进行子程序的调用
参数	• num:自定义数值
返回值	无

FRLua 已定义了以下暂停方式

代码 3-2 Pause 示例

```
1.  Pause(0)  --无功能
2.  Pause(2)  --气缸未到位
3.  Pause(3)  --螺钉未到位
4.  Pause(4)  --浮锁处理
5.  Pause(5)  --滑牙处理
```

3.1.4 子程序

NewDofile：子程序调用

表 3-8 NewDofile 详细参数

属性	说明
原型	NewDofile (name_path, layer, id)
描述	进行子程序的调用
参数	• name_path：包含文件子程序的文件路径，"/fruser/###.lua"； • layer：调用子程序的层号； • id：id 编号。
返回值	无

DofileEnd：子程序调用结束

表 3-9 DofileEnd 详细参数

属性	说明
原型	DofileEnd ()
描述	子程序调用结束
参数	无
返回值	无

代码 3-3 子程序调用与关闭示例

```
1.  --调用 dofile1.lua 子程序
2.  NewDofile("/fruser/dofile1.lua",1,1);
3.
4.  DofileEnd();--子程序调用结束
```

3.1.5 变量

变量的基础内容详见 2.1.3 节。在 FRLua 还中定义了查询变量类型以及系统变量查询和赋值。

RegisterVar：变量类型查询

表 3-10 RegisterVar 详细参数

属性	说明
原型	RegisterVar (type, name)
描述	变量类型查询
参数	• type: 变量类型 • name: 变量名
返回值	无

GetSysVarValue：获取系统变量

表 3-11 GetSysVarValue 详细参数

属性	说明
原型	GetSysVarValue (s_var)
描述	获取系统变量
参数	• s_var：系统变量名称。
返回值	var_value：系统变量值

SetSysVarValue，设置系统变量

表 3-12 SetSysVarValue 详细参数

属性	说明
原型	SetSysVarValue (s_var, value)
描述	设置系统变量
参数	• s_var：系变量名称； • value：输入的变量值。
返回值	无

代码 3-4 FRLua 变量和系统变量值相关操作示例

```
1.  local frValue1 = 0.0
2.  RegisterVar("number","frValue1")--数字型变量查询
3.  local frString = “X:3.4, Y:0.0”
4.  RegisterVar("string"," frString")--字符型变量查询
5.
6.  TEST_1 = GetSysVarValue(s_var_3)--获取系统变量值,并赋值给 TEST_1
7.  SetSysVarValue(s_var_3,1) --设置系统变量值
```

3.2 运动指令

3.2.1 点到点

PTP：点到点

表 3-13 PTP 详细参数

属性	说明
原型	PTP (point_name, ovl, blendT, offset_flag, offset_x, offset_y, offset_z, offset_rx, offset_ry, offset_rz)
描述	点到点运动
参数	• point_name:目标示教点位名称； • ovl:调试速度，范围[0~100%]； • blendT:[-1]-不平滑，[0~500]-平滑时间，单位[ms]； • offset_flag:[0]-不偏移，[1]-工件/基坐标系下偏移，[2]-工具坐标系下偏移 默认 0； • offset_x~offset_rz:偏移量，单位[mm][°] 。
返回值	无

MoveJ：关节空间运动

表 3-14 MoveJ 详细参数

属性	说明
原型	MoveJ (j1, j2, j3, j4, j5, j6, x, y, z, rx, ry, rz, tool, user, speed, acc, ovl, ep1, ep2, ep3, ep4, blendT, offset, offset_x, offset_y, offset_z, offset_rx, offset_ry, offset_rz)
描述	关节空间运动
参数	• j1~j6:目标关节位置，单位[°]； • x,y,z,rx,ry,rz:目标笛卡尔位姿，单位[mm][°]； • tool:工具号； • user:工件号；

表 3-14（续表）

属性	说明
参数	• speed:速度，范围[0~100%]； • acc:加速度，范围[0~100%]，暂不开放； • ovl:调试速度，范围[0~100%]； • ep1~ep4:外部轴 1 位置~外部轴 4 位置； • blendT:[-1]-不平滑，[0~500]-平滑时间，单位[ms]； • offset:[0]-不偏移，[1]-工件/基坐标系下偏移，[2]-工具坐标系下偏移； • offset_x~offset_rz:偏移量，单位[mm][°]。
返回值	无

代码 3- 5 利用点到点指令进行运动示例

```
1.  --利用 MoveJ 进行运动
2.  x,y,z,rx,ry,rz=GetForwardKin(149.135,-79.058,-78.558,-145.409,-94.182,88.654)
3.  MoveJ(149.135,-79.058,-78.558,-145.409,-
    94.182,88.654,x,y,z,rx,ry,rz,1,0,100,180,100,0.000,0.000,0.000,0.000,0,0,0,0,0,0,0,0)
4.  --利用 PTP 进行运动
5.  PTP(DW01,100,-1,0)--目标点位名称：DW01，速度百分比：100，是否阻塞：是（-1-停止），是否偏移：0-否
6.  PTP(DW01,100,10,0)
7.  -- 目标点位名称：DW01，速度百分比：100，是否阻塞：否（10-在此平滑过渡时间 10ms），是否偏移：0-否
8.  PTP(DW01,100,10,1,0,0,0,0,0,0)
9.  -- 目标点位名称：DW01，速度百分比：100，是否阻塞：否（10ms），是否偏移：是（1-工件/基坐标系下偏移），位姿偏移量：[0.0,0.0,0.0,0.0,0.0,0.0]
10. PTP(DW01,100,10,2,0,0,0,0,0,0)
11. --目标点位名称：DW01，速度百分比：100，是否阻塞：否（10ms），是否偏移：是（2-工具坐标系下偏移），位姿偏移量：[0.0,0.0,0.0,0.0,0.0,0.0]
```

3.2.2 直线

Lin：直线运动

表 3-15 Lin 详细参数

属性	说明
原型	Lin (point_name, ovl, blendR, search, offset_flag, offset_x, offset_y, offset_z, offset_rx, offset_ry, offset_rz)
描述	直线 Lin 运动
参数	• point_name:目标点位名称； • ovl:调试速度，[0~100] 默认 100.0;

表 3-15（续表）

属性	说明
参数	• blendR: [-1.0]-运动到位（阻塞），[0~1000]-平滑半径（非阻塞），单位[mm]; • search:[0]-不（焊丝）寻位，[1]-（焊丝）寻位； • offset:[0]-不偏移，[1]-工件/基坐标系下偏移，[2]-工具坐标系下偏移； • offset_x~offset_rz:偏移量，单位[mm][°]。
返回值	无

MoveL：笛卡尔空间直线运动

表 3-16 MoveL 详细参数

属性	说明
原型	MoveL (j1, j2, j3, j4, j5, j6, x, y, z, rx, ry, rz, tool, user, speed, acc, ovl, ep1,ep2, ep3, ep4, blendR, search, offset, offset_x, offset_y, offset_z, offset_rx, offset_ry, offset_rz)
描述	笛卡尔空间直线运动
参数	• j1~j6:目标关节位置，单位[°]； • x,y,z,rx,ry,rz:目标笛卡尔位姿，单位[mm][°]； • tool:工具号； • user:工件号； • speed:速度，范围[0~100%]； • acc:加速度，范围[0~100%]，暂不开放； • ovl:调试速度，范围[0~100%]； • ep1~ep4:外部轴 1 位置~外部轴 4 位置； • blendR:[-1]-不平滑，[0~1000]-平滑半径，单位[mm]； • search:[0]-不焊丝寻位，[1]-焊丝寻位； • offset:[0]-不偏移，[1]-工件/基坐标系下偏移，[2]-工具坐标系下偏移； • offset_x~offset_rz:偏移量，单位[mm][°]。
返回值	无

代码 3-6 利用直线指令进行运动示例

```
1.  --利用 MoveL 进行直线运动
2.  j1,j2,j3,j4,j5,j6=GetInverseKin(0,-315.039,327.526,786.334,0.052,-32.916,-32.464,-1)
3.  MoveL(j1,j2,j3,j4,j5,j6,-315.039,327.526,786.334,0.052,-32.916,-32.464, 1,0, 100, 180,
    100, -1, 0.000,0.000,0.000,0.000,0,0,0,0,0,0,0,0)
4.  --基础 Lin 直线运动
5.  Lin(DW01,100,-1,0,0)--目标点位名称：DW01，速度百分比：100，是否阻塞：是（-
```

代码 3-6（续）

```
    1），是否寻位：否，是否偏移：否
6.  Lin(DW01,100,10,0,0)--目标点位信息：DW01，速度百分比：100，是否阻塞：否
    （10mm），是否寻位：否，是否偏移：否
```

3.2.3 圆弧

ARC：圆弧运动

表 3-17 ARC 详细参数

属性	说明
原型	ARC (point_p_name, poffset, offset_px, offset_py, offset_pz, offset_prx, offset_pry, offset_prz, point_t_name, toffset, offset_tx, offset_ty, offset_tz, offset_trx, offset_try, offset_trz, ovl, blend)
描述	ARC 圆弧运动
参数	• point_p_name:圆弧中间点名称； • poffset:[0]-不偏移，[1]-工件/基坐标系下偏移，[2]-工具坐标系下偏移； • offset_px~offset_prz:偏移量，单位[mm][°]； • point_t_name:圆弧终点名称; • toffset:[0]-不偏移，[1]-工件/基坐标系下偏移，[2]-工具坐标系下偏移； • offset_tx~offset_trz:偏移量，单位[mm][°]。 • ovl:调试速度，范围[0~100%]； • blendR:[-1]-不平滑，[0~1000]-平滑半径，单位[mm]。
返回值	无

MoveC：笛卡尔空间圆弧运动

表 3-18 MoveC 详细参数

属性	说明
原型	MoveC (pj1, pj2, pj3, pj4, pj5, pj6, px, py, pz, prx, pry, prz, ptool, puser, pspeed, pacc, pep1, pep2, pep3, pep4, poffset, offset_px, offset_py, offset_pz, offset_prx, offset_pry, offset_prz, tj1, tj2, tj3, tj4, tj5, tj6, tx, ty, tz, trx, try, trz, ttool, tuser, tspeed, tacc, tep1, tep2, tep3, tep4, toffset, offset_tx, offset_ty, offset_tz, offset_trx, offset_try, offset_trz, ovl, blendR)
描述	笛卡尔空间圆弧运动
参数	• pj1~pj6:路径点关节位置，单位[°]； • px,py,pz,prx,pry,prz:路径点笛卡尔位姿，单位[mm][°]； • ptool:工具号； • puser:工件号； • pspeed:速度，范围[0~100%]；

表 3-18（续表）

属性	说明
参数	• pacc:加速度，范围[0~100%]，暂不开放； • pep1~pep4:外部轴 1 位置~外部轴 4 位置； • poffset:[0]-不偏移，[1]-工件/基坐标系下偏移，[2]-工具坐标系下偏移； • offset_px~offset_prz:偏移量，单位[mm][°]； • tj1~tj6:目标点关节位置，单位[°]； • tx,ty,tz,trx,try,trz:目标点笛卡尔位姿，单位[mm][°]； • ttool:工具号； • tuser:工件号； • tspeed:速度，范围[0~100%]； • tacc:加速度，范围[0~100%]，暂不开放； • tep1~tep4:外部轴 1 位置~外部轴 4 位置； • toffset:[0]-不偏移，[1]-工件/基坐标系下偏移，[2]-工具坐标系下偏移； • offset_tx~offset_trz:偏移量，单位[mm][°]。 • ovl:调试速度，范围[0~100%]； • blendR:[-1]-不平滑，[0~1000]-平滑半径，单位[mm]。
返回值	无

代码 3- 7 利用圆弧指令进行运动示例

```
1.  --利用 MoveC 进行圆弧运动
2.  pj1,pj2,pj3,pj4,pj5,pj6=GetInverseKin(0,388.104,-462.265,-5.226,177.576,-
    1.292,143.417,-1)
3.  tj1,tj2,tj3,tj4,tj5,tj6=GetInverseKin(0, 271.474,-476.328,3.739,179.502,-2.433,134.753,-1)
4.  MoveC(pj1,pj2,pj3,pj4,pj5,pj6,388.104,-462.265,-5.226,177.576,-
    1.292,143.417,1,0,100,180,0.000,0.000,0.000,0.000,0,0,0,0,0,0,0,
    tj1,tj2,tj3,tj4,tj5,tj6,271.474,-476.328,3.739,179.502,                    -
    2.433,134.753,1,0,100,180,0.000,0.000,0.000,0.000,0,0,0,0,0,0,0,0,100,-1)
5.  --利用 ARC 进行基础圆弧运动
6.  PTP(DW01,100,-1,0)--点到点方式，运动到起始点位
7.  --Lin(DW01,100,-1,0,0)--直线方式，运动到起始点位
8.  ARC(DW02,0,0,0,0,0,0,0,DW03,0,0,0,0,0,0,0,100,-1)
9.  --DW02 圆弧运动的中间点位，0-不偏移；DW03：圆弧终点坐标，0-不偏移，100-运
    动速度百分比，-1-在终点停止
10.
11. --利用 ARC 基于基坐标偏移的圆弧运动
12. PTP(DW01, 100, -1, 0)--点到点方式，运动到起始点位
13. ARC(DW02, 1, 1, 2, 3, 4, 5, 6, DW03, 1, 11, 12, 13, 14, 15, 16, 100, -1)
14. --DW02 圆弧运动的中间点位，1-基坐标偏移；1，2，3，4，5，6-偏移的笛卡尔坐标，
    DW03：圆弧终点坐标，1-基坐标偏移，11，12，13，14，15，16-偏转的笛卡尔坐标，
```

代码 3-7（续）

```
    100-运动速度百分比，-1-在终点偏移
15.
16. -- 利用 ARC 基于工具坐标偏移的圆弧运动
17. PTP(DW01,100,-1,0)--点到点方式，运动到起始点位
18. ARC(DW02,2,1,2,3,4,5,6,DW03,2,11,12,13,14,15,16,100,-1)
19. --DW02 圆弧运动的中间点位，2-工具标偏移；1，2，3，4，5，6-偏移的笛卡尔坐标，
    DW03：圆弧终点坐标，2-工具坐标偏移，11，12，13，14，15，16-偏转的笛卡尔坐
    标，100-运动速度百分比，-1-在终点偏移
20.
21. --利用 ARC 开启平滑的圆弧运动
22. PTP(DW01,100,-1,0)--点到点方式，运动到起始点位
23. ARC(DW02,0,0,0,0,0,0,0,0,DW03,0,0,0,0,0,0,0,0,100,30)
24. --DW02 圆弧运动的中间点位，0-不偏移；DW03：圆弧终点坐标，0-不偏移，100-运
    动速度百分比，30-平滑 30mm
```

3.2.4 整圆

Circle：整圆运动（笛卡尔空间）

表 3-19 Circle 详细参数

属性	说明
原型	Circle (pj1, pj2, pj3, pj4, pj5, pj6, px, py, pz, prx, pry, prz, ptool, puser, pspeed, pacc, pep1, pep2, pep3, pep4, tj1, tj2, tj3, tj4, tj5, tj6, tx, ty, tz, trx, try, trz, ttool, tuser, tspeed, tacc, tep1, tep2, tep3, tep4, ovl, offset, offset_x, offset_y, offset_z, offset_rx, offset_ry, offset_rz)
描述	整圆运动（笛卡尔空间）
参数	• pj1~pj6:路径点关节位置，单位[°]； • px,py,pz,prx,pry,prz:路径点笛卡尔位姿，单位[mm][°]； • ptool:工具号； • puser:工件号； • pspeed:速度，范围[0~100%]； • pacc:加速度，范围[0~100%]，暂不开放； • pep1~pep4:外部轴 1 位置~外部轴 4 位置； • tj1~tj6:目标点关节位置，单位[°]； • tx,ty,tz,trx,try,trz:目标点笛卡尔位姿，单位[mm][°]； • ttool:工具号； • tuser:工件号； • tspeed:速度，范围[0~100%]； • tacc:加速度，范围[0~100%]，暂不开放； • tep1~tep4:外部轴 1 位置~外部轴 4 位置；

表 3-19（续表）

属性	说明
参数	• ovl:调试速度，范围[0~100%]； • offset:[0]-不偏移，[1]-工件/基坐标系下偏移，[2]-工具坐标系下偏移； • offset_x~offset_rz:偏移量，单位[mm][°]。
返回值	无

Circle：整圆运动

表 3-20 新 Circle 详细参数

属性	说明
原型	Circle (pos_p_name, pos_t_name, ovl, offset_flag, offset, offset_x, offset_y, offset_z, offset_rx, offset_ry, offset_rz)
描述	整圆运动
参数	• pos_p_name:整圆中间点 1 名称； • pos_t_name:整圆中间点 2 名称; • ovl:调试速度，范围[0~100%]； • offset:[0]-不偏移，[1]-工件/基坐标系下偏移，[2]-工具坐标系下偏移； • offset_x~offset_rz:偏移量，单位[mm][°]。
返回值	无

代码 3-8 利用整圆指令进行运动

```
1.  --整圆运动（笛卡尔空间）
2.  pj1,pj2,pj3,pj4,pj5,pj6=GetInverseKin(0,388.104,-462.265,-5.226,177.576,-
    1.292,143.417,-1)
3.  tj1,tj2,tj3,tj4,tj5,tj6=GetInverseKin(0, 271.474,-476.328,3.739,179.502,-2.433,134.753,-1)
4.  Circle(pj1,pj2,pj3,pj4,pj5,pj6,388.104,-462.265,-5.226,177.576,-1.292, 143.417, 1, 0, 100,
    180, 0.000, 0.000, 0.000, 0.000, tj1, tj2, tj3, tj4, tj5, tj6, 271.474, -476.328, 3.739, 179.502,-
    2.433,134.753,1,0,100,180,0.000,0.000,0.000,0.000,100,0,0,0,0,0,0,0)
5.
6.  --Circle 整圆运动
7.  PTP(DW01,100,-1,0)--点到点运动到起始点位
8.  --Lin(DW01,100,-1,0,0)--直线运动到起始点位
9.
10. Circle(DW02,DW03,100,0)
11. --DW02 整圆运动的中间点位（路径点 1）；DW03：圆弧终点坐标（路径点 2），100-
    运动速度百分比，0-不偏移
12.
13. Circle(DW02,DW03,100,1,0,0,10,0,0,0)
14. -- DW02 整圆运动的中间点位；DW03：圆弧终点坐标，100-运动速度百分比，1-基
```

代码 3-8（续）

```
    于基坐标偏移，0,0,10,0,0,0 关节偏移角度
15.
16. Circle(DW02,DW03,100,2, 0,0,10,0,0,0)
17. -- DW02 整圆运动的中间点位；DW03：圆弧终点坐标，100-运动速度百分比，2-基
    于工具坐标偏移，0,0,10,0,0,0 关节偏移角度
```

3.2.5 螺旋

Spiral：螺旋运动

表 3-21 Spiral 详细参数

属性	说明
原型	Spiral (pos_1_name, pos_2_name, pos_3_name, ovl, offset_flag, offset_x, offset_y, offset_z, offset_rx, offset_ry, offset_rz, circle_num, circle_angle_Co_rx, circle_angle_Co_ry, circle_angle_Co_rz, rad_add, rotaxis_add)
描述	Spiral 螺旋运动
参数	• pos_1_name，螺旋线中间点 1 名称； • pos_2_name，螺旋线中间点 2 名称； • pos_3_name，螺旋线中间点 3 名称； • ovl:调试速度，范围[0~100%]，默认 100.0; • offset_flag:[0]-不偏移，[1]-工件/基坐标系下偏移，[2]-工具坐标系下偏移默认 0; • offset_x~offset_rz:偏移量，单位[mm][°]； • circle_num，螺旋圈数； • circle_angle_Co_rx~ circle_angle_Co_rx:姿态角修正，单位[°] • rad_add: 半径增量，单位 [mm]; • rotaxis_add: 转轴方向增量，单位 [mm]。
返回值	无

代码 3-9 螺旋指令运动示例

```
1.  --基础 Spiral 螺旋运动
2.  Spiral (DW01, DW02, DW03, 100,0,0,0,0,0,0,0,5,0,0,0,10,10)
3.  --DW01-螺旋线中间点 1 名称，DW02-螺旋线中间点 2 名称，DW03-螺旋线中间点 3
    名称，100-调试速度，0-不偏移，5-螺旋圈数，（0，0，0）姿态角修正，10-半径增
    量，10-转轴方向增量；
4.  --基坐标偏移的 Spiral 螺旋运动
5.  Spiral (DW01, DW02, DW03,100,1,1,0,0,10, 0,0,0,0,0,0,10,10)
```

代码 3-9（续）

```
6.  --DW01-螺旋线中间点 1 名称，DW02-螺旋线中间点 2 名称，DW03-螺旋线中间点 3 名称，100-调试速度，1-基坐标偏移，（0,0,10, 0,0,0）-偏移参数，5-螺旋圈数，（0，0，0）姿态角修正，10-半径增量，10-转轴方向增量；
7.  --工具坐标偏移的 Spiral 螺旋运动
8.  Spiral(DW01,DW02,DW03,100,2,1, 0,0,10, 0,0,0,0,0,0,10,10)
9.  --DW01-螺旋线中间点 1 名称，DW02-螺旋线中间点 2 名称，DW03-螺旋线中间点 3 名称，100-调试速度，0-工具坐标偏移，（0,0,10, 0,0,0）-偏移参数，5-螺旋圈数，（0，0，0）姿态角修正，10-半径增量，10-转轴方向增量。
```

3.2.6 新螺旋

NewSpiral：新螺旋运动

表 3-22 NewSpiral 详细参数

属性	说明
原型	NewSpiral (desc_pos_name, ovl, offset_flag=2, offset_x, offset_y, offset_z, offset_rx, offset_ry, offset_rz, circle_num, circle_angle, rad_init, rad_add，rot_direction)
描述	NewSpiral 新螺旋运动
参数	• desc_pos_name:新螺旋运动起始点位名称； • ovl:调试速度，范围[0~100%]，默认 100.0; • offset_flag:[0]-不偏移，[1]-工件/基坐标系下偏移，[2]-工具坐标系下偏移默认 2; • offset_x~offset_rz:偏移量，单位[mm][°]； • circle_num: 螺旋圈数; • circle_angle: 螺旋倾角，单位[°]; • rad_init: 螺旋初始半径，单位[mm]; • rad_add: 半径增量，单位[mm]; • rotaxis_add: 转轴方向增量，单位[mm]; • rot_direction: 旋转方向，0-顺时针，1-逆时针。
返回值	无

代码 3-10 新螺旋指令运动示例

```
1.  --顺时针 N-Spiral 螺旋运动
2.  PTP(DW01,100,0,2,50,0,0,-30,0,0)--利用 PTP 运动到螺旋线起点（固定的运动方式）
3.  --DW01-螺旋线起点 1 名称，100-调试速度，2-工具坐标偏移，（50，0，0，-30，0，0）-偏移参数（x,y,z,rx,ry,rz）
4.  NewSpiral(DW01,100,2,50,0,0,- 30,0,0,5,30,50,10,15,0)
5.  --DW01-螺旋线起点 1 名称,100-调试速度,2-工具坐标偏移，（50，0，0，-30，0，0）
```

代码 3-10（续）

```
    -偏移参数（x,y,z,rx,ry,rz）,5-螺旋圈数，30-螺旋倾角，50-初始半径，10-半径增量，
    15 转轴方向增量，0-顺时针

6.  --逆时针 N-Spiral 螺旋运动
7.  PTP(DW01,100,0,2,50,0,0,-30,0,0)--利用 PTP 运动到螺旋线起点
8.  NewSpiral(DW01,100,2,50,0,0,- 30,0,0,5,30,50,10,15,1)
9.  --DW01-螺旋线起点 1 名称,100-调试速度,2-工具坐标偏移，（50，0，0，-30，0，0）
    -偏移参数（x,y,z,rx,ry,rz）,5-螺旋圈数，30-螺旋倾角，50-初始半径，10-半径增量，
    15 转轴方向增量，1-逆时针
```

3.2.7 水平螺旋

H-Spiral 水平螺旋运动由 HorizonSpiralMotionStart 和 HorizonSpiralMotionEnd 组合完成。

HorizonSpiralMotionStart：水平螺旋运动开始

表 3-23 HorizonSpiralMotionStart 详细参数

属性	说明
原型	HorizonSpiralMotionStart (rad，vel, rot_direction, circle_angle)
描述	水平螺旋运动开始
参数	• rad: 螺旋半径，单位[mm]; • vel:旋转速度，单位[rev/s]; • rot_direction: 旋转方向，0-顺时针，1-逆时针; • circle_angle: 螺旋倾角，单位[°]
返回值	无

HorizonSpiralMotionEnd：结束水平螺旋运动

表 3-24 HorizonSpiralMotionEnd 详细参数

属性	说明
原型	HorizonSpiralMotionEnd ()
描述	水平螺旋运动结束
参数	无
返回值	无

代码 3-11 H-Spiral 水平螺旋运动示例

```
1.  --顺时针的 H-Spiral 水平螺旋
2.  HorizonSpiralMotionStart(30,2,0,20)
3.  --水平螺旋，30-旋转半径，2-选择速度，0-顺时针旋转，20-旋转倾斜角度
4.  Lin(DW01,100,-1,0,0)
5.  HorizonSpiralMotionEnd()--水平螺旋结束
6.
7.  --逆时针的 H-Spiral 水平螺旋
8.  HorizonSpiralMotionStart(30,2,1,20)
9.  --水平螺旋，30-旋转半径，2-选择速度，1-逆时针旋转，20-旋转倾斜角度
10. Lin(DW01,100,-1,0,0)
11. HorizonSpiralMotionEnd()--水平螺旋结束
```

3.2.8 样条

样条指令分为样条组起始，样条段和样条组结束三部分，样条组开始是样条运动的起始标志，样条段目前节点图包含 SPL、SLIN、SCIRC，样条组结束是样条运动的结束标志。

SplineStart：样条运动开始

表 3-25 SplineStart 详细参数

属性	说明
原型	SplineStart ()
描述	样条组开始
参数	无
返回值	无

SPL 方式一：SPTP 类型的样条段

表 3-26 SPTP 详细参数

属性	说明
原型	SPTP(point_name, ovl)
描述	SPTP 样条段
参数	• point_name:目标点位名称； • ovl:调试速度，范围[0~100%]。
返回值	无

SPL 方式二：SplinePTP 类型的样条段

表 3-27 SplinePTP 详细参数

属性	说明
原型	SplinePTP (j1, j2, j3, j4, j5, j6, x, y, z, rx, ry, rz, tool, user, speed, acc, ovl)
描述	SplinePTP 样条运动
参数	• j1~j6:目标关节位置，单位[°]； • x,y,z,rx,ry,rz:目标笛卡尔位姿，单位[mm][°]； • tool:工具号； • user:工件号； • speed:速度，范围[0~100%]； • acc:加速度，范围[0~100%]，暂不开放； • ovl:调试速度，范围[0~100%]。
返回值	无

SLIN 方式一：SLIN 类型的样条段

表 3-28 SLIN 详细参数

属性	说明
原型	SLIN (point_name, ovl)
描述	SLIN 样条段
参数	• point_name:目标点位名称； • ovl:调试速度，范围[0~100%]。
返回值	无

SLIN 方式二：SplineLINE 类型的样条段

表 3-29 SplineLINE 详细参数

属性	说明
原型	SplineLINE (j1, j2, j3, j4, j5, j6, x, y, z, rx, ry, rz, tool, user, speed, acc, ovl)
描述	SplineLINE 样条段
参数	• j1~j6:目标关节位置，单位[°]； • x,y,z,rx,ry,rz:目标笛卡尔位姿，单位[mm][°]； • tool:工具号； • user:工件号； • speed:速度，范围[0~100%]； • acc:加速度，范围[0~100%]，暂不开放； • ovl:调试速度，范围[0~100%]。
返回值	无

SCIRC 方式一：SCIRC 类型的样条段

表 3-30 SCIRC 详细参数

属性	说明
原型	SCIRC(pos_p_name, pos_t_name, ovl)
描述	SCIRC 样条段
参数	• pos_p_name:圆弧中间点名称； • pos_t_name:圆弧终点名称； • ovl:调试速度，范围[0~100%]。
返回值	无

SCIRC 方式二：SplineCIRC 类型的样条段

表 3-31 SplineCIRC 详细参数

属性	说明
原型	SplineCIRC (pj1, pj2, pj3, pj4, pj5, pj6, px, py, pz, prx, pry, prz, ptool, puser, pspeed, pacc, tj1, tj2, tj3, tj4, tj5, tj6, tx, ty, tz, trx, try, trz, ttool, tuser, tspeed, tacc,ovl)
描述	SplineCIRC 样条段
参数	• pj1~pj6:圆弧中间点关节位置，单位[°]； • px,py,pz,prx,pry,prz: 圆弧中间点笛卡尔位姿，单位[mm][°]； • ptool: 圆弧中间点工具号； • puser: 圆弧中间点工件号； • pspeed: 圆弧中间点速度，范围[0~100%]； • pacc: 圆弧中间点加速度，范围[0~100%]，暂不开放； • tj1~tj6:圆弧终点关节位置，单位[°]； • tx,ty,tz,trx,try,trz: 圆弧终点笛卡尔位姿，单位[mm][°]； • ttool: 圆弧终点工具号； • tuser: 圆弧终点工件号； • tspeed: 圆弧终点速度，范围[0~100%]； • tacc: 圆弧终点加速度，范围[0~100%]，暂不开放； • ovl:调试速度，范围[0~100%]。
返回值	无

SplineEnd：样条组结束

表 3-32 SplineEnd 详细参数

属性	说明
原型	SplineEnd ()

表 3-32（续表）

属性	说明
描述	SplineEnd 样条组结束
参数	无
返回值	无

代码 3-12 方式一样条运动示例

```
1.  -- SPTP 样条段的样条运动
2.  SplineStart()--样条运动开始
3.  SPTP(DW01,100)--DW01-点位名称，100-调试速度
4.  SPTP(DW02,100)--DW02-点位名称，100-调试速度
5.  SPTP(DW03,100)--DW03-点位名称，100-调试速度
6.  SPTP(DW04,100)--DW04-点位名称，100-调试速度
7.  SplineEnd()--样条运动结束
8.
9.  --SLIN 样条段的样条运动
10. SplineStart()--样条运动开始
11. SLIN(DW01,100)--DW01-点位名称，100-调试速度
12. SLIN(DW02,100)--DW02-点位名称，100-调试速度
13. SLIN(DW03,100)--DW03-点位名称，100-调试速度
14. SLIN(DW04,100)--DW04-点位名称，100-调试速度
15. SplineEnd()--样条运动结束
16.
17. -- SCIRC 样条段的样条运动
18. SplineStart()--样条运动开始
19. SCIRC(DW01,DW02,100)-- DW01-圆弧中间点位名称，DW02 圆弧终点位名称，100-
    调试速度 100%
20. SCIRC(DW03,DW04,100) -- DW03-圆弧中间点位名称，DW04 圆弧终点位名称，100-
    调试速度 100%
21. SCIRC(DW05,DW06,100) -- DW05-圆弧中间点位名称，DW06 圆弧终点位名称，100-
    调试速度 100%
22. SCIRC(DW07,DW08,100) -- DW07-圆弧中间点位名称，DW08 圆弧终点位名称，100-
    调试速度 100%
23. SplineEnd()--样条运动结束
```

代码 3-13 方式二样条运动示例

```
1.  --SplinePTP 样条段的样条运动
2.  SplineStart()--样条运动开始
3.  SplinePTP(-88.938,-67.089,-119.074,-57.750,78.739,-53.107,-154.495,-456.371,271.098,-
```

代码 3-13（续）

```
    172.005,-27.192,-130.384,1,0,100,180,100)
4.  SplinePTP(-50.137,-67.089,-119.074,-57.750,78.739,-53.108,165.568,-452.472,271.098, -172.005, -27.192,-91.582,1,0,100,180,100)
5.  SplinePTP(-116.604,-103.398,-106.020,-60.282,89.088,-26.541,-340.231,-440.449, 59.996,179.861,-0.950,179.936,1,0,100,180,100)
6.  SplinePTP(-117.355,-89.202,-120.591,-59.927,89.057,-27.266,-297.517,-341.920,69.240,179.817,-0.966,179.910,1,0,100,180,100)
7.  SplineEnd()--样条运动结束

8.  --SplineLINE 样条段的样条运动
9.  SplineStart()--样条运动开始
10. SplineLINE(-88.938,-67.089,-119.074,-57.750,78.739,-53.107,-154.495,-456.371, 271.098, -172.005,-27.192,-130.384,1,0,100,180,100)
11. SplineLINE(-50.137,-67.089,-119.074,-57.750,78.739,-53.108,165.568,-452.472, 271.098,-172.005,-27.192,-91.582,1,0,100,180,100)
12. SplineLINE(-116.604,-103.398,-106.020,-60.282,89.088,-26.541,-340.231,-440.449, 59.996,179.861,-0.950,179.936,1,0,100,180,100)
13. SplineLINE(-117.355,-89.202,-120.591,-59.927,89.057,-27.266,-297.517,-341.920, 69.240,179.817,-0.966,179.910,1,0,100,180,100)
14. SplineEnd()--样条运动结束
15.
16. --SplineCIRC 样条段的样条运动
17. SplineStart()--样条运动开始
18. SplineCIRC(-88.938, -67.089, -119.074, -57.750, 78.739, -53.107, -154.495, -456.371, 271.098, -172.005, -27.192, -130.384, 1, 0, 100, 180, -50.137, -67.089, -119.074, -57.750, 78.739, -53.108, 165.568, -452.472, 271.098, -172.005, -27.192, -91.582, 1,0,100,180,100)
19.
20. SplineCIRC(-116.604, -103.398, -106.020, -60.282, 89.088, -26.541, -340.231, -440.449, 59.996, 179.861, -0.950, 179.936, 1, 0, 100, 180, -117.355, -89.202, -120.591, -59.927, 89.057, -27.266,-297.517,-341.920,69.240,179.817,-0.966,179.910,1,0,100,180,100)
21.
22. SplineCIRC(-110.420, -104.178,-103.638,-59.952,89.153,-26.480,-297.923,-494.668, 71.977,-178.379,-1.753,-173.981,1,0,100,180,-115.854,-85.308,-123.979,-59.371,89.058,-27.513,-279.135,-330.367,72.864,-179.245,-1.455,-178.362,1,0,100,180,100)
23.
24. SplineCIRC(-108.752,-104.491,-103.204,-59.601,89.035,-26.465,-285.668,-507.818, 73.101, -178.008,-2.068,-172.346,1,0,100,180,-123.459,-95.123,-113.554,-62.039,87.801, -26.914,-361.158,-339.783,72.733,178.366,-1.636,173.492,1,0,100,180,100)
25. SplineEnd()--样条运动结束
```

3.2.9 新样条

NewSplineStart：新样条多点轨迹起始

表 3-33 NewSplineStart 详细参数

属性	说明
原型	NewSplineStart (Con_mode, Gac_time)
描述	新样条多点轨迹起始
参数	• Con_mode：控制模式，0-圆弧过渡点，1-给定过渡点； • Gac_time：全局平均衔接时间，大于 10。
返回值	无

NewSP：方式一新样条多点轨迹段

表 3-34 NewSP 详细参数

属性	说明
原型	NewSP (point_name, ovl, blendR, islast_point)
描述	新样条多点轨迹段
参数	• point_name：点位名称; • ovl:调试速度，范围[0~100%]; • blendR：平滑半径[0~1000]，单位 [mm]; • islast_point：是否最后一个点，0-否，1-是。
返回值	无

NewSplinePoint：方式二新样条多点轨迹段

表 3-35 NewSplinePoint 详细参数

属性	说明
原型	NewSplinePoint(j1, j2, j3, j4, j5, j6, x, y, z, rx, ry, rz, tool, user, speed, acc, ovl , blendR)
描述	新样条多点轨迹段
参数	• j1~j6:目标关节位置，单位[°]; • x,y,z,rx,ry,rz:目标笛卡尔位姿，单位[mm][°]; • tool:工具号; • user:工件号; • speed:速度，范围[0~100%]; • acc:加速度，范围[0~100%]，暂不开放; • ovl:调试速度，范围[0~100%]; • blendR:[-1]-不平滑，[0~1000]-平滑半径，单位[mm]。
返回值	无

NewSplineEnd：样条组结束

表 3-36 NewSplineEnd 详细参数

属性	说明
原型	NewSplineEnd ()
描述	新样条组结束
参数	无
返回值	无

代码 3-14 方式一新样条指令运动示例

```
1.  --N-Spline 圆弧过渡点控制模式的新样条运动
2.  NewSplineStart(0,10)--样条运动开始,0-圆弧过渡点控制模式，10-全局平均衔接时间
    10ms
3.  NewSP(DW01,100,10,0)--DW01-点位名称，100-调试速度，10-平滑过渡半径 10mm，
    0-不是最后一个点
4.  NewSP(DW02,100,10,0)--DW02-点位名称，100-调试速度，10-平滑过渡半径 10mm，
    0-不是最后一个点
5.  NewSP(DW03100,10,0)--DW03-点位名称，100-调试速度，10-平滑过渡半径 10mm，
    0-不是最后一个点
6.  NewSP(DW04,100,10,1)--DW04-点位名称，100-调试速度，10-平滑过渡半径 10mm，
    1-是最后一个点
7.  NewSplineEnd()—新样条运动结束

8.  --N-Spline 给定路径点控制模式的新样条运动
9.  NewSplineStart(1,10)--样条运动开始,1-给定路径点控制模式，10-全局平均衔接时间
    10ms
10. NewSP(DW01,100,10,0)--DW01-点位名称，100-调试速度，10-平滑过渡半径 10mm，
    0-不是最后一个点
11. NewSP(DW02,100,10,0)--DW02-点位名称，100-调试速度，10-平滑过渡半径 10mm，
    0-不是最后一个点
12. NewSP(DW03,100,10,0)--DW03-点位名称，100-调试速度，10-平滑过渡半径 10mm，
    0-不是最后一个点
13. NewSP(DW04,100,10,1)--DW04-点位名称，100-调试速度，10-平滑过渡半径 10mm，
    1-是最后一个点
14. NewSplineEnd()--新样条运动结束
```

代码 3-15 方式二新样条指令运动示例

```
1.  --N-Spline 圆弧过渡点控制模式的新样条运动
2.  NewSplineStart(0,10)--样条运动开始,0-圆弧过渡点控制模式，10-全局平均衔接时间
```

代码 3-15（续）

```
    10ms
3.
4.  NewSplinePoint(-88.938,-67.089,-119.074,-57.750,78.739,-53.107,-154.495,-456.371,
    271.098,-172.005,-27.192,-130.384,1,0,100,180,100,10,0)
5.  NewSplinePoint(-50.137,-67.089,-119.074,-57.750,78.739,-53.108,165.568,-452.472,
    271.098,-172.005,-27.192,-91.582,1,0,100,180,100,10,0)
6.  NewSplinePoint(-116.604,-103.398,-106.020,-60.282,89.088,-26.541,-340.231,-440.449,
    59.996,179.861,-0.950,179.936,1,0,100,180,100,10,0)
7.  NewSplinePoint(-117.355,-89.202,-120.591,-59.927,89.057,-27.266,-297.517,-341.920,
    69.240,179.817,-0.966,179.910,1,0,100,180,100,10,1)
8.  NewSplineEnd()—新样条运动结束
9.
10. --N-Spline 给定路径点控制模式的新样条运动
11. NewSplineStart(1,10)
12. --样条运动开始,1-给定路径点控制模式，10-全局平均衔接时间 10ms
13.
14. NewSplinePoint(-88.938,-67.089,-119.074,-57.750,78.739,-53.107,-154.495,-
    456.371,271.098,-172.005,-27.192,-130.384,1,0,100,180,100,0,0)
15. NewSplinePoint(-50.137,-67.089,-119.074,-57.750,78.739,-53.108,165.568,-
    452.472,271.098,-172.005,-27.192,-91.582,1,0,100,180,100,0,0)
16. NewSplinePoint(-116.604,-103.398,-106.020,-60.282,89.088,-26.541,-340.231,-
    440.449,59.996,179.861,-0.950,179.936,1,0,100,180,100,0,0)
17. NewSplinePoint(-117.355,-89.202,-120.591,-59.927,89.057,-27.266,-297.517,-
    341.920,69.240,179.817,-0.966,179.910,1,0,100,180,100,0,1)
18. NewSplineEnd()--新样条运动结束
```

3.2.10 摆动

WeaveStart：摆动开始

表 3-37 WeaveStart 详细参数

属性	说明
原型	WeaveStart(weaveNum)
描述	摆动开始
参数	• weaveNum： 摆焊参数配置编号。
返回值	无

WeaveEnd：摆动结束

表 3-38 WeaveEnd 详细参数

属性	说明
原型	WeaveEnd(weaveNum)
描述	摆动结束
参数	• weaveNum： 摆焊参数配置编号。
返回值	无

WeaveStartSim：仿真摆动开始

表 3-39 WeaveStartSim 详细参数

属性	说明
原型	WeaveStartSim(weaveNum)
描述	仿真摆动开始
参数	• weaveNum： 摆焊参数配置编号。
返回值	无

WeaveEndSim：仿真摆动结束

表 3-40 WeaveEndSim 详细参数

属性	说明
原型	WeaveEndSim (weaveNum)
描述	仿真摆动结束
参数	• weaveNum： 摆焊参数配置编号。
返回值	无

WeaveInspectStart：开始轨迹预警

表 3-41 WeaveInspectStart 详细参数

属性	说明
原型	WeaveInspectStart (weaveNum)
描述	开始轨迹预警
参数	• weaveNum： 摆焊参数配置编号。
返回值	无

WeaveInspectEnd：停止轨迹预警

表 3-42 WeaveInspectEnd 详细参数

属性	说明
原型	WeaveInspectEnd (weaveNum)
描述	停止轨迹预警
参数	• weaveNum： 摆焊参数配置编号。
返回值	无

代码 3-16 摆动指令运动示例

```
1.  WeaveInspectStart(0);--开始轨迹预警
1.  Lin(DW01,100,0,0,0)
2.  WeaveInspectEnd(0);--停止轨迹预警
3.
4.  WeaveStartSim(weaveNum) -- 仿真摆动开始
5.  Lin(DW01,100,0,0,0)
6.  WeaveEndSim(weaveNum) -- 仿真摆动结束
7.
8.  WeaveStart(weaveNum) -- 开始摆动
9.  Lin(DW01,100,0,0,0)
10. WeaveEnd(weaveNum) -- 结束摆动
```

3.2.11 轨迹复现

LoadTPD：轨迹预加载

表 3-43 LoadTPD 详细参数

属性	说明
原型	LoadTPD(name)
描述	轨迹预加载
参数	• name:轨迹名。
返回值	无

MoveTPD：轨迹复现

表 3-44 MoveTPD 详细参数

属性	说明
原型	MoveTPD(name, blend, ovl)
描述	轨迹复现
参数	• name:轨迹名，/fruser/traj/trajHelix_aima_2.txt;; • blend:是否平滑，0-不平滑，1-平滑; • ovl:调试速度，范围[0~100]。
返回值	无

代码 3-17 轨迹复现示例

```
1.  LoadTPD("20lin")
2.  MoveTPD("20lin",0,25)
```

3.2.12 点偏移

点偏移指令为整体偏移指令，输入各个偏移量，将开启指令和关闭指令添加到程序中，在开始和关闭中间的运动指令会基于基坐标（或工件坐标）进行偏移。

PointsOffsetEnable：点位整体偏移开始

表 3-45 PointsOffsetEnable 详细参数

属性	说明
原型	PointsOffsetEnable(flag, x,y,z,rx,ry,rz)
描述	点位整体偏移开始
参数	• flag:0-基坐标或工件坐标系下偏移， 2-工具坐标系下偏移; • x,y,z,rx,ry,rz: 位姿偏移量，单位[mm][°]。
返回值	无

PointsOffsetDisable：点位整体偏移结束

表 3-46 PointsOffsetDisable 详细参数

属性	说明
原型	PointsOffsetDisable()
描述	点位整体偏移结束
参数	无
返回值	无

代码 3- 18 点偏移示例

```
1. PointsOffsetEnable (0,0,0,10,0,0,0)
2. --点位整体偏移开始，0-基坐标或工件坐标系下偏移，（0,0,10,0,0,0）-偏移量
3. PTP(DW01,100,-1,0)--点到点运动
4. PointsOffsetDisable()--点位整体偏移结束
```

3.2.13 伺服

伺服控制（笛卡尔空间运动）指令，该指令可以通过绝对位姿控制或基于当前位姿偏移来控制机器人运动。

ServoMoveStart：伺服运动开始

表 3-47 ServoMoveStart 详细参数

属性	说明
原型	ServoMoveStart()
描述	伺服运动开始
参数	无
返回值	无

ServoMoveEnd：伺服运动结束

表 3-48 ServoMoveEnd 详细参数

属性	说明
原型	ServoMoveEnd()
描述	伺服运动结束
参数	无
返回值	无

ServoCart：笛卡尔空间伺服模式运动

表 3-49 ServoCart 详细参数

属性	说明
原型	ServoCart (mode, x, y, z, Rx, Ry, Rz, pos_gainx, pos_gainy, pos_gainz, pos_gainrx, pos_gainry, pos_gainrz, acc, vel, cmdT, filterT, gain)
描述	笛卡尔空间伺服模式运动
参数	• mode:[0]-绝对运动(基坐标系)，[1]-增量运动(基坐标系)，[2]-增量运动(工具坐标系)；

表 3-49（续表）

属性	说明
参数	• x,y,z,Rx,Ry,Rz: 标笛卡尔位姿或位姿增量，单位[mm]； • pos_gainx, pos_gainy, pos_gainz, pos_gainrx, pos_gainry, pos_gainrz:位姿增量比例系数，仅在增量运动下生效，范围 [0~1]； • acc:加速度，范围 [0~100]； • vel:速度，范围 [0~100]； • cmdT: 指令下发周期，单位 s，建议范围[0.001~0.0016]； • filterT:滤波时间，滤波时间，单位 s； • gain:目标位置的比例放大器。
返回值	无

代码 3-19 伺服示例

```
1.  --伺服控制
2.  mode = 2
3.  --[0]-绝对运动(基坐标系)，[1]-增量运动(基坐标系)，[2]-增量运动(工具坐标系)
4.  count = 0
5.  ServoMoveStart()  --伺服运动开始
6.  While (count<100) do
7.      ServoCart(mode, 0.0,0.0,0.5,0.0,0.0,0.0, 40)  --笛卡尔空间伺服模式运动
8.      count = count + 1
9.      WaitMs(10)
10. end
11. ServoMoveEnd()  --伺服运动结束
```

3.2.14 轨迹

轨迹（Trajctory）指令适用于相机直接给定轨迹的通用接口，满足在已有固定格式的离散的轨迹点文件时，可导入系统使得机器人按照导入文件的轨迹进行运动。

1.轨迹文件导入功能：选择本地计算机文件导入机器人控制系统；

2.轨迹预加载：选择已导入的轨迹文件通过指令加载；

3.轨迹运动：通过预加载的轨迹文件和选择的调试速度组合指令下发机器人运动；

4.打印轨迹点编号：在机器人运行轨迹的过程中打印轨迹点编号，以便查看当前运动的进度。

LoadTrajectory：轨迹预加载

表 3-50 LoadTrajectory 详细参数

属性	说明
原型	LoadTrajectory (name)
描述	轨迹预加载
参数	• name:轨迹名,如：/fruser/traj/trajHelix_aima_1.txt。
返回值	无

通过 GetTrajectoryStartPose、GetActualTCPNum 和 GetActualWObjNum 分别获取轨迹起始点位参数、工具坐标号和工件坐标号。

GetTrajectoryStartPose：获取轨迹起始位姿

表 3-51 GetTrajectoryStartPose 详细参数

属性	说明
原型	GetTrajectoryStartPose (name)
描述	获取轨迹起始位姿
参数	• name:轨迹名,如：/fruser/traj/trajHelix_aima_1.txt。
返回值	desc_pose {x,y,z,rx,ry,rz}

GetActualTCPNum：获取当前工具坐标系编号

表 3-52 GetActualTCPNum 详细参数

属性	说明
原型	GetActualTCPNum (flag)
描述	获取当前工具坐标系编号
参数	• flag：0-阻塞，1-非阻塞 默认 1。
返回值	tool_id:工具坐标系编号

GetActualWObjNum：获取当前工件坐标系编号

表 3-53 GetActualWObjNum 详细参数

属性	说明
原型	GetActualWObjNum (flag)
描述	获取当前工具坐标系编号
参数	• flag：0-阻塞，1-非阻塞 默认 1。
返回值	wobj_id:工件坐标系编号

MoveCart：笛卡尔空间点到点运动

表 3-54 MoveCart 详细参数

属性	说明
原型	MoveCart (desc_pos, ool, user, vel, acc, ovl, blendT, config)
描述	笛卡尔空间点到点运动
参数	• desc_pos:目标笛卡尔位置； • tool:工具号，[0~14]； • user:工件号，[0~14]； • vel:速度，范围 [0~100]，默认为 100; • acc:加速度，范围 [0~100]，暂不开放,默认为 100; • ovl:调试速度，范围[0~100%]; •blendT:[-1.0]-运动到位 (阻塞)，[0~500]-平滑时间 (非阻塞)，单位 [ms] 默认为 -1.0; • config:关节配置，[-1]-参考当前关节位置求解，[0~7]-依据关节配置求解，默认为 -1。
返回值	无

MoveTrajectory：轨迹复现

表 3-55 MoveTrajectory 详细参数

属性	说明
原型	MoveTrajectory (name, ovl)
描述	轨迹复现
参数	• name:轨迹名,如：/fruser/traj/trajHelix_aima_1.txt； • ovl:调试速度，范围[0~100%]。
返回值	无

GetTrajectoryPointNum：获取轨迹点编号

表 3-56 GetTrajectoryPointNum 详细参数

属性	说明
原型	GetTrajectoryPointNum ()
描述	获取轨迹点编号
参数	无
返回值	Num: 轨迹点编号

代码 3-20 轨迹示例

```
1.  --轨迹
2.  LoadTrajectory("/fruser/traj/trajHelix_aima_1.txt")--预加载轨迹文件的绝对路径
3.  startPose = GetTrajectoryStartPose("/fruser/traj/trajHelix_aima_1.txt")
4.  --获取轨迹起始位姿
5.  tool_num = GetActualTCPNum()--获取当前工具坐标系编号
6.  wobj_num = GetActualWObjNum()--获取当前工件坐标系编号
7.  MoveCart(startPose,tool_num,wobj_num,100,100,25,-1,-1)
8.  --笛卡尔空间点到点运动运动到轨迹起始点位，startPose-目标笛卡尔位置，100-速度，
    100-加速度，-1-到位停止，-1-关节按配置求解
9.  MoveTrajectory("/fruser/traj/trajHelix_aima_1.txt",25)
10. --轨迹复现，/fruser/traj/trajHelix_aima_1.txt-轨迹文件名称，25-调试速度（调试速度）
11. num = GetTrajectoryPointNum()--获取轨迹点编号
12. RegisterVar("number","num")--将编号信息打印出来
```

3.2.15 轨迹 J

TrajctoryJ 指令与 Trajctory 一样，适用于相机直接给定轨迹的通用接口，满足在已有固定格式的离散的轨迹点文件时，可导入系统使得机器人按照导入文件的轨迹进行运动。

LoadTrajectoryJ：轨迹预处理

表 3-57 LoadTrajectoryJ 详细参数

属性	说明
原型	LoadTrajectoryJ(name, ovl, opt)
描述	轨迹预处理
参数	• name:轨迹名,如：/fruser/traj/trajHelix_aima_2.txt; • ovl：调试速度，范围[0~100]; • opt：0-路径点，1-控制点。
返回值	无

MoveTrajectoryJ：轨迹复现

表 3-58 MoveTrajectoryJ 详细参数

属性	说明
原型	MoveTrajectoryJ ( )
描述	轨迹复现
参数	无
返回值	无

代码 3-21 轨迹 J 示例

```
1.  LoadTrajectoryJ("/fruser/traj/trajHelix_aima_2.txt",30,0)
2.  --轨迹 J 预处理，/fruser/traj/trajHelix_aima_2.txt-轨迹文件名称，30-调试速度，1-控制点
3.  startPose = GetTrajectoryStartPose("/fruser/traj/trajHelix_aima_2.txt")
4.  --获取轨迹起始位姿
5.  tool_num = GetActualTCPNum()--获取当前工具坐标系编号
6.  wobj_num = GetActualWObjNum()--获取当前工件坐标系编号
7.  MoveCart(startPose,tool_num,wobj_num,100,100,25,-1,-1)
8.  --笛卡尔空间点到点运动运动到轨迹起始点位，startPose-目标笛卡尔位置，100-速度，100-加速度，-1-到位停止，-1-关节按配置求解
9.  MoveTrajectoryJ()
10. num = GetTrajectoryPointNum()--获取轨迹点编号
11. RegisterVar("number","num")--将编号信息打印出来
```

3.2.16 DMP

DMP/dmpMotion 是一种轨迹模仿学习的方法，需要事先规划参考轨迹。DMP 具体路径为以新的起点模仿参考轨迹的新轨迹。

DMP：轨迹模仿

表 3-59 DMP 详细参数

属性	说明
原型	DMP (point_name, ovl)
描述	轨迹模仿
参数	• point_name:目标点位名称 • ovl:调试速度，范围[0~100%]。
返回值	无

dmpMotion：轨迹模仿

表 3-60 dmpMotion 详细参数

属性	说明
原型	dmpMotion (joint_pos, desc_pos，tool, user, vel, acc, ovl, exaxis_pos)
描述	轨迹模仿
参数	• joint_pos:目标关节位置，单位[°]； • desc_pos:目标笛卡尔位姿，单位 [mm][°] 默认初值为[0.0,0.0,0.0,0.0,0.0,0.0]，默认值调用正运动学求解返回值; • tool:工具号，[0~14]；

表 3-60（续表）

属性	说明
参数	• user:工件号，[0~14]； • vel:速度百分比，[0~100] 默认 100.0; • acc：加速度百分比，[0~100]，暂不开放； • ovl:调试速度，范围[0~100%]; • exaxis_pos: 外部轴 1 位置 ~ 外部轴 4 位置 默认[0.0,0.0,0.0,0.0]。
返回值	无

代码 3-22 轨迹模仿示例

```
1.  --DMP 轨迹模仿
2.  DMP(DW01,100)--轨迹模仿，DW01-起始点位名称，100-调试速度
3.  --dmpMotion 轨迹模仿
4.  dmpMotion({-88.938,-67.089,-119.074,-57.750,78.739,-53.107},{-154.495,-456.371,
    271.098,-172.005,-27.192,-130.384},1,0,100,180,100, {0.000,0.000,0.000,0.000})
```

3.2.17 工件转换

WPTrsf，工件坐标系转换，该指令实现时在其执行内部的 PTP、LIN 指令，工件坐标系下点位自动转换。

WorkPieceTrsfStart：工件坐标转换开始

表 3-61 WorkPieceTrsfStart 详细参数

属性	说明
原型	WorkPieceTrsfStart (id)
描述	工件坐标转换开始
参数	• id:目标工件坐标系编号，如 0-wobjcoord0, 1-wobjcoord1。
返回值	无

WorkPieceTrsfEnd：工件坐标转换结束

表 3-62 WorkPieceTrsfEnd 详细参数

属性	说明
原型	WorkPieceTrsfEnd ( )
描述	工件坐标转换开始

表 3-62（续表）

属性	说明
参数	无
返回值	无

代码 3- 23 工件坐标转换示例

```
1.  --进行工件坐标转换
2.  WorkPieceTrsfStart(1)--工件坐标转换开始，1 工件坐标系编号
3.  PTP(DW01,100,0,0)
4.  --DW01-需要转换的点位名称，100-调试速度，0-阻塞（停止），0-不偏移
5.  PTP(DW02,100,0,0)
6.  --DW02-需要转换的点位名称，100-调试速度，0-阻塞（停止），0-不偏移
7.  PTP(DW03,100,0,0)
8.  --DW03-需要转换的点位名称，100-调试速度，0-阻塞（停止），0-不偏移
9.  --工件转换结束
10. WorkPieceTrsfEnd()
```

3.2.18 工具转换

工具坐标系转换（ToolTrsf），该指令实现时在其执行内部的 PTP、LIN 指令，工具坐标系下点位自动转换。

SetToolList：设置工具坐标系

表 3-63 SetToolList 详细参数

属性	说明
原型	SetToolList (name)
描述	设置工具坐标系
参数	• name:目标工具坐标系名称，如 toolcoord0, toolcoord1。
返回值	无

ToolTrsfStart：工具坐标系转换开始

表 3-64 ToolTrsfStart 详细参数

属性	说明
原型	ToolTrsfStart (id)

表 3-64（续表）

属性	说明
描述	工具坐标系转换开始
参数	• id:目标工具坐标系编号，如 0-toolcoord0, 1-toolcoord1。
返回值	无

ToolTrsfEnd：工具坐标系转换结束

表 3-65 ToolTrsfEnd 详细参数

属性	说明
原型	ToolTrsfEnd ()
描述	工具坐标系转换结束
参数	无
返回值	无

代码 3-24 工具坐标转换示例

```
1. --工具坐标转换
2. SetToolList(toolcoord0)--设置工具坐标转换，toolcoord0-目的工具坐标名称
3. ToolTrsfStart(0)--工具坐标转换开始，0-工具坐标系编号
4. PTP(DW01,100,0,0)--DW01-需要转换的点位名称，100-调试速度，0-阻塞（停止），
   0-不偏移
5. PTP(DW02,100,0,0)--DW02-需要转换的点位名称，100-调试速度，0-阻塞（停止），
   0-不偏移
6. PTP(DW03,100,0,0)--DW03-需要转换的点位名称，100-调试速度，0-阻塞（停止），
   0-不偏移
7. ToolTrsfEnd()--工具坐标转换结束
```

3.3 控制指令

3.3.1 数字 IO

数字“IO”指令分为设置 IO（SetDO/SPLCSetDO）和获取 IO（GetDI/SPLCGetDI）两部分。

SetDO：设置控制箱数字量阻塞输出

表 3-66 SetDO 详细参数

属性	说明
原型	SetDO (id, status, smooth, thread)
描述	设置控制箱数字量阻塞输出
参数	• id:io 编号，0~7:控制箱 DO0~DO7，8~15:控制箱 CO0~CO7； • status:0-Flase，1-True； • smooth:0-Break，1-Serious； • thread:是否应用线程，0-否，1-是。
返回值	无

SPLCSetDO：设置控制箱数字量非阻塞输出

表 3-67 SPLCSetDO 详细参数

属性	说明
原型	SPLCSetDO (id, status)
描述	设置控制箱数字量非阻塞输出
参数	• id:io 编号, 0~7:控制箱 DO0~DO7，8~15:控制箱 CO0~CO7； • status:0-False，1-True。
返回值	无

SetToolDO：设置工具数字量阻塞输出

表 3-68 SetToolDO 详细参数

属性	说明
原型	SetToolDO (id, status, smooth, thread)
描述	设置工具数字量阻塞输出
参数	• id: io 编号，0 - End-DO0, 1 - End-DO1； • status:0-False，1-True； • smooth:0-break，1-Serious； • thread:是否应用线程，0-否，1-是。
返回值	无

SPLSetToolDO：设置工具数字量非阻塞输出

表 3-69 SPLSetToolDO 详细参数

属性	说明
原型	SPLSetToolDO (id, status, smooth, thread)
描述	设置工具数字量非阻塞输出
参数	• id: io 编号，0-End-DO0, 1-End-DO1； • status:0-False，1-True。
返回值	无

GetDI：阻塞获取控制箱数字量输入

表 3-70 GetDI 详细参数

属性	说明
原型	ret = GetDI(id，thread)
描述	阻塞获取控制箱数字量输入
参数	• id:io 编号, 0~7:控制箱 DI0~DI7，8~15:控制箱 CI0~CI7； • thread:是否应用线程，0-否，1-是。
返回值	• ret: 0-无效，1-有效

SPLCGetDI：非阻塞获取 IO

表 3-71 SPLCGetDI 详细参数

属性	说明
原型	SPLCGetDI (id, status, stime)
描述	非阻塞获取控制箱数字量输入
参数	• id:io 编号, 0~7:控制箱 DI0~DI7，8~15:控制箱 CI0~CI7； • status:0-False，1-True； • stime:等待时间单位[ms]。
返回值	• ret: 0-无效，1-有效

GetToolDI：阻塞获取工具数字量输入

表 3-72 GetToolDI 详细参数

属性	说明
原型	GetToolDI (id, thread)
描述	阻塞获取工具数字量输入
参数	• id:io 编号, 0-End-DI0，1-End-DI1；

表 3-72（续表）

属性	说明
参数	• thread:是否应用线程，0-否，1-是。
返回值	• ret: 0-无效，1-有效。

SPLCGetDI：非阻塞获取 IO

表 3-73 SPLCGetDI 详细参数

属性	说明
原型	SPLCGetToolDI (id, status, stime)
描述	非阻塞获取控制箱数字量输入
参数	• id:io 编号, 0-End-DI0，1-End-DI1； • status:0-False，1-True； • stime:等待时间单位[ms]。
返回值	• ret: 0-无效，1-有效

代码 3-25 数字 IO 示例

```
1.  --设置数字 IO
2.  SetDO(0,1,0,1)--设置控制箱数字量阻塞输出
3.  SPLCSetDO(1,1)--设置控制箱数字量非阻塞输出
4.  SetToolDO(1,0,1,1) --设置工具数字量阻塞输出
5.  SPLCSetToolDO(1,0) --设置工具数字量非阻塞输出
6.  --PTP(DW01,100,0,0)--点到点运动方式
7.
8.  --获取数字 IO
9.  Ret1 = GetDI(0,1)--阻塞获取控制箱数字量输入
10. Ret2 =SPLCGetDI(1,0,1000)--非阻塞获取控制箱数字量输入
11. Ret3=GetToolDI(1,0) --阻塞获取工具数字量输入
12. Ret4 =SPLCGetToolDI(1,0,100) --非阻塞获取工具数字量输入
```

3.3.2 模拟 IO

在该指令中，分为设置模拟输出（SetAO/SPLCSetAO）和获取模拟输入（GetAI/SPLCGetAI）两部分功能。

SetAO：设置控制箱模拟量阻塞输出

表 3-74 SetAO 详细参数

属性	说明
原型	SetAO (id, value, thread)
描述	设置控制箱模拟量阻塞输出
参数	• id:io 编号，0 - AI0, 1 - AI1； •value:电流或电压值百分比，范围[0~100%]对应电流值[0~20mA]或电压[0~10V]； • thread:是否应用线程，0-否，1-是。
返回值	无

SPLCSetAO：设置控制箱模拟量非阻塞输出

表 3-75 SPLCSetAO 详细参数

属性	说明
原型	SPLCSetAO (id, value)
描述	设置控制箱模拟量非阻塞输出
参数	• id:io 编号，0 - AI0, 1 - AI1； •value:电流或电压值百分比，范围[0~100%]对应电流值[0~20mA]或电压[0~10V]。
返回值	无

SetToolAO：设置工具模拟量输出

表 3-76 SetToolAO 详细参数

属性	说明
原型	SetToolAO (id, value, thread)
描述	设置工具拟量阻塞输出
参数	• id:io 编号，0 - End-AO0； •value:电流或电压值百分比，范围[0~100%]对应电流值[0~20mA]或电压[0~10V]； • thread:是否应用线程，0-否，1-是。
返回值	无

SPLCSetToolAO：设置工具模拟量非阻塞输出

表 3-77 SPLCSetToolAO 详细参数

属性	说明
原型	SPLCSetToolAO (id, value)
描述	设置工具模拟量非阻塞输出
参数	• id:io 编号,0 - End-AO0； •value:电流或电压值百分比，范围[0~100%]对应电流值[0~20mA]或电压[0~10V]。
返回值	无

GetAI：获取控制箱模拟量输入

表 3-78 GetAI 详细参数

属性	说明
原型	GetAI(id, thread)
描述	阻塞获取控制箱模拟量输入
参数	• id:io 编号，0 - AI0, 1 - AI1； • thread:是否应用线程，0-否，1-是。
返回值	Value: 输入电流或电压值百分比，范围 [0~100] 对应电流值 [0~20mA] 或电压 [0~10V]

SPLCGetAI：非阻塞获取控制箱模拟量输入

表 3-79 SPLCGetAI 详细参数

属性	说明
原型	SPLCGetAI (id, condition, value, stime)
描述	非阻塞获取控制箱模拟量输入
参数	• id:io 编号，0 - AI0, 1 - AI1； • value:数值，1%~100%； • condition:0 - >，1 - <; • stime:最大时间，单位[ms]；
返回值	Status:返回状态，1 - 成功，0 - 失败。

GetToolAI：阻塞获取工具模拟量输入

表 3-80 GetToolAI 详细参数

属性	说明
原型	GetToolAI (id, thread)
描述	阻塞获取工具模拟量输入
参数	• id:io 编号，0 - AI0, 1 - AI1； • thread:是否应用线程，0-否，1-是。
返回值	Value: 输入电流或电压值百分比，范围 [0~100] 对应电流值 [0~20mA] 或电压 [0~10V]

SPLCGetToolAI：非阻塞获取工具模拟量输入

表 3-81 SPLCGetToolAI 详细参数

属性	说明
原型	SPLCGetToolAI (id, condition, value, stime)
描述	非阻塞获取工具箱模拟量输入
参数	• id:io 编号，0 - AI0, 1 - AI1； • value:数值，1%~100%； • condition:0 - >，1 - <; • stime:最大时间，单位[ms]；
返回值	Status:返回状态，1 - 成功，0 - 失败。

代码 3-26 模拟 IO 示例

```
1.  --设置模拟量
2.  SetAO(0,10,0)--设置控制箱模拟量阻塞输出
3.  SPLCSetAO(1,10)--设置控制箱模拟量非阻塞输出
4.  SetToolAO(0,10,0) --设置工具模拟量阻塞输出
5.  SPLCSetToolAO(0,10) --设置工具模拟量非阻塞输出
6.  --获取模拟量
7.  Value1 = GetAI(0,0)--阻塞获取控制箱模拟量输入
8.  Value2 =SPLCGetAI(1,0,30,1000)--非阻塞获取控制箱模拟量输入
9.  Value3 =GetToolAI(0,1) --阻塞获取工具模拟量输入
10. Value3 =SPLCGetToolAI(0,0,10,50) --非阻塞获取工具模拟量输入
```

3.3.3 虚拟 IO

Virtual-IO 为虚拟的 IO 控制指令，实现设置或获取模拟外部 DI 和 AI 状态。

SetVirtualDI：设置模拟外部 DI

表 3-82 SetVirtualDI 详细参数

属性	说明
原型	SetVirtualDI (id, status)
描述	设置模拟外部 DI
参数	• id:io 编号, 0~15:控制箱 Vir-Ctrl-DI0~DI5； • status:0-Flase，1-True。
返回值	无

SetVirtualToolDI：设置模拟外部工具 DI

表 3-83 SetVirtualToolDI 详细参数

属性	说明
原型	SetVirtualToolDI (id, status)
描述	设置模拟外部工具 DI
参数	• id: io 编号，0 - Vir-End-DI0, 1 - Vir-End-DI1； • status:0-Flase，1-True。
返回值	无

GetVirtualDI：获取模拟外部 DI

表 3-84 GetVirtualDI 详细参数

属性	说明
原型	GetVirtualDI (id)
描述	获取模拟外部 DI
参数	• id:io 编号, 0~15:控制箱 Vir-Ctrl-DI0~DI5。
返回值	ret: 0-无效，1-有效。

GetVirtualToolDI：获取模拟外部工具 DI

表 3-85 GetVirtualToolDI 详细参数

属性	说明
原型	GetVirtualToolDI (id)
描述	获取模拟外部工具 DI

表 3-85（续表）

属性	说明
参数	• id: io 编号，0 - Vir-End-DI0, 1 - Vir-End-DI1。
返回值	ret: 0-无效，1-有效。

SetVirtualAI：设置模拟外部 AI

表 3-86 GetVirtualAI 详细参数

属性	说明
原型	SetVirtualAI (id, value)
描述	设置模拟外部 AI
参数	• id:io 编号，0 - Vir-Ctrl-AI0, 1 - Vir-Ctrl-AI1； • value:对应电流值[0~20mA]或电压[0~10V]。
返回值	无

SetVirtualToolAI：设置模拟外部工具 AI

表 3-87 GetVirtualToolAI 详细参数

属性	说明
原型	SetVirtualToolAI (id, value)
描述	设置模拟外部工具 AI
参数	• id:io 编号，0 - Vir-End-AI0； • value:对应电流值[0~20mA]或电压[0~10V]。
返回值	无

GetVirtualAI，获取模拟外部 AI

表 3-88 GetVirtualAI 详细参数

属性	说明
原型	GetVirtualAI (id)
描述	获取模拟外部 AI
参数	• id:io 编号，0 - Vir-Ctrl-AI0, 1 - Vir-Ctrl-AI1。
返回值	Value: 输入电流或电压值百分比，范围 [0~100] 对应电流值 [0~20mA] 或电压 [0~10V]

GetVirtualToolAI，获取模拟外部工具 AI

表 3-89 GetVirtualToolAI 详细参数

属性	说明
原型	Value = GetVirtualToolAI (id)
描述	获取模拟外部工具 AI
参数	• id:io 编号，0 - Vir-End-AI0。
返回值	Value: 输入电流或电压值百分比，范围 [0~100] 对应电流值 [0~20mA] 或电压 [0~10V]

代码 3-27 虚拟 IO 示例

```
1.  --模拟外部 DI 设置与获取
2.  SetVirtualDI(0,1)--设置模拟外部 DI，0-端口号 DI0，1-True
3.  SetVirtualAI(0,5)--设置模拟外部 AI，0-端口号 AI0，5-数值 5ma
4.  Ret1 = GetVirtualDI(1)--获取模拟外部 DI，1-端口号 DI1
5.  Value1 = GetVirtualAI(1)--获取模拟外部 AI，1-端口号 AI1
6.
7.  --模拟外部工具 DI 设置与获取
8.  SetVirtualToolDI(1,0) --设置模拟外部工具 DI
9.  SetVirtualToolAI(0,12) --设置模拟外部工具 AI
10. Ret2 = GetVirtualToolDI(1) --获取模拟外部工具 DI
11. Value2 = GetVirtualToolAI(0) --获取模拟外部工具 AI
```

3.3.4 运动 DO

运动 DO 的相关指令分为连续输出模式和单次输出模式实现直线运动过程中，根据设定的间隔，连续输出 DO 信号功能。

MoveDOStart：运动中并行设置控制箱 DO 状态开始

表 3-90 MoveDOStart 详细参数

属性	说明
原型	MoveDOStart (doNum，distance，dutyCycle)
描述	运动中并行设置控制箱 DO 状态开始
参数	• doNum:控制箱 DO 编号，0~7: 控制箱 DO0~DO7，8~15:控制箱 CO0~CO7; • distance:间隔距离，范围:0~500, 单位[mm，默认 10]; • dutyCycle: 输出脉冲占空比单位[%]，0~99, 默认 50%。
返回值	无

MoveDOStop：运动中并行设置控制箱 DO 状态停止

表 3-91 MoveDOStop 详细参数

属性	说明
原型	MoveDOStop ()
描述	运动中并行设置控制箱 DO 状态停止
参数	无
返回值	无

MoveToolDOStart：运动中并行设置工具 DO 状态开始

表 3-92 MoveToolDOStart 详细参数

属性	说明
原型	MoveToolDOStart (doNum，distance，dutyCycle)
描述	运动中并行设置工具 DO 状态开始
参数	• doNum:工具 DO 编号，0 - End-DO0，1 - End-DO1； • distance:间隔距离，范围:0~500，单位[mm，默认 10]; • dutyCycle：输出脉冲占空比单位[%]，0~99，默认 50%。
返回值	无

MoveToolDOStop：运动中并行设置工具 DO 状态停止

表 3-93 MoveToolDOStop 详细参数

属性	说明
原型	MoveToolDOStop ()
描述	运动中并行设置工具 DO 状态停止
参数	无
返回值	无

代码 3-28 运动 DO 示例

```
1.  --控制箱
2.  MoveDOStart (1,10,50)
3.  --设置运动 DO 连续输出，1-端口号 DO1，10-时间间隔 10mm，50-输出脉冲占空比
    50%
4.  Lin(DW01,100,-1,0,0)--直线运动
5.  MoveDOStop()--停止运动 DO 输
```

代码 3-28（续）

```
6.
7.  --工具
8.  MoveToolDOStart(0,10,50)
9.  Lin(DW01,100,-1,0,0) --直线运动
10. MoveToolDOStop()--停止运动 DO 输
```

3.3.5 运动 AO

MoveAO，配合运动指令使用时，可实现在运动过程中，根据实时 TCP 速度按比例输出 AO 信号

MoveAOStart：控制箱运动 AO 开始

表 3-94 MoveAOStart 详细参数

属性	说明
原型	MoveAOStart (AONum, maxTCPSpeed, maxAOPercent, zeroZoneCmp)
描述	控制箱运动 AO 开始
参数	• AONum:控制箱 AO 编号, 0 - AO0,1 - AO1; • maxTCPSpeed：最大 TCP 速度值[1-5000mm/s]，默认 1000； • maxAOPercent：最大 TCP 速度值对应的 AO 百分比，默认 100%； • zeroZoneCmp：死区补偿值 AO 百分比，整形，默认为 20%，范围[0-100]。
返回值	无

MoveAOStop：控制箱运动 AO 结束

表 3-95 MoveAOStop 详细参数

属性	说明
原型	MoveAOStop()
描述	控制箱运动 AO 结束
参数	无
返回值	无

MoveToolAOStart：工具运动 AO 开始

表 3-96 MoveAOStart 详细参数

属性	说明
原型	MoveToolAOStart (AONum, maxTCPSpeed, maxAOPercent, zeroZoneCmp)
描述	工具运动 AO 开始
参数	• AONum:控制箱 AO 编号, 0 - AO0,1 - AO1; • maxTCPSpeed：最大 TCP 速度值[1-5000mm/s]，默认 1000； • maxAOPercent：最大 TCP 速度值对应的 AO 百分比，默认 100%； • zeroZoneCmp：死区补偿值 AO 百分比，整形，默认为 20%，范围[0-100]。
返回值	无

MoveToolAOStop：工具运动 AO 结束

表 3-97 MoveAOStop 详细参数

属性	说明
原型	MoveToolAOStop ()
描述	工具运动 AO 结束
参数	无
返回值	无

代码 3-29 运动 AO 示例

```
1.  --控制箱
2.  MoveAOStart(1,1000,100,20)--设置运动 AO 输出，1-端口号 AO1，1000-最大 TCP 速度，100-最大 TCP 速度百分比，20-死区补偿值 AO 百分比
3.  Lin(DW01,100,0,0,0) --直线运动
4.  MoveAOStop()--停止运动 AO 输出
5.
6.  --工具
7.  MoveToolAOStart (0,1000,100,20)
8.  Lin(DW01,100,0,0,0)
9.  MoveToolAOStop ()
```

3.3.6 扩展 IO

扩展 IO（Aux-IO）是机器人与 PLC 通讯控制外部扩展 IO 的指令功能，需

要机器人与 PLC 建立 UDP 通讯，在原有的 16 路输入输出基础上，可以扩展 128 路输入输出。

ExtDevSetUDPComParam：配置 UDP 通讯数据

表 3-98 ExtDevSetUDPComParam 详细参数

属性	说明
原型	ExtDevSetUDPComParam (ip, port, period)
描述	UDP 扩展轴通讯参数配置
参数	• ip：PLC IP 地址； • port：端口号； • period：通讯周期(ms)。
返回值	无

ExtDevLoadUDPDriver 加载 UDP 通信

表 3-99 ExtDevLoadUDPDriver 详细参数

属性	说明
原型	ExtDevLoadUDPDriver()
描述	加载 UDP 通信
参数	无
返回值	无

代码 3-30 运动 AO 示例

```
1.  -- UDP 扩展轴通讯参数配置与加载
2.  ExtDevSetUDPComParam("192.168.58.88",2021,2)
3.  --UDP 通讯配置，"192.168.58.88"-IP 地址，2021-端口号，2-通讯周期
4.  ExtDevLoadUDPDriver()--加载 UDP 驱动程序以启用通信。
5.  WaitMs(500)--等待 500 毫秒，确保 UDP 驱动程序已正确加载。
```

SetAuxDO：设置扩展 DO

表 3-100 SetAuxDO 详细参数

属性	说明
原型	SetAuxDO (DONum, status, smooth, thread)
描述	设置扩展 DO
参数	• DONum: DO 编号，范围[0~127]；

表 3-102（续表）

属性	说明
参数	• status:0-Flase，1-True； • smooth:0-Break，1-Serious； • thread:是否应用线程，0-否，1-是。
返回值	无

GetAuxDI：获取扩展 DI

表 3-101 GetAuxDI 详细参数

属性	说明
原型	GetAuxDI (DINum)
描述	获取扩展 DI 值
参数	• DINum: DI 编号，范围[0~127]。
返回值	isOpen： 0-关；1-开

SetAuxAO：设置扩展 AO

表 3-102 SetAuxAO 详细参数

属性	说明
原型	SetAuxAO (AONum, value, thread)
描述	设置扩展 AO
参数	• id: AO 编号，范围[0~3]； •value:电流或电压值百分比，范围[0~100%]对应电流值[0~20mA]或电压[0~10V]； • thread:是否应用线程，0-否，1-是。
返回值	无

GetAuxAI：获取扩展 AI 值

表 3-103 GetAuxAI 详细参数

属性	说明
原型	GetAuxAI (AINum, thread)
描述	获取扩展 AI 值
参数	• AINum: AuxAI 编号，范围[0~3]； • thread:是否应用线程，0-否，1-是。
返回值	Value: 输入电流或电压值百分比，范围 [0~100] 对应电流值 [0~20mA] 或电压 [0~10V]

WaitAuxDI：等待扩展 DI 输入

表 3-104 WaitAuxDI 详细参数

属性	说明
原型	WaitAuxDI(DINum,bOpen,time, timeout)
描述	等待扩展 DI 输入
参数	• DINum： DI 编号； • bOpen：开关 True-开,False-关； • time：最大等待时间(ms)； • timeout：等待超时处理 0-停止报错，1-继续等待，2-一直等待。
返回值	无

WaitAuxAI：等待扩展 AI 输入

表 3-105 WaitAuxAI 详细参数

属性	说明
原型	WaitAuxAI (AINum, sign, value, time, timeout)
描述	等待扩展 AI 输入
参数	• AINum： AI 编号； • sign：0-大于；1-小于； • value：AI 值； • time：最大等待时间(ms)； • timeout：等待超时处理 0-停止报错，1-继续等待，2-一直等待。
返回值	无

代码 3-31 扩展 IO 指令集示例

```
1.  --设置扩展 DO
2.  SetAuxDO(0,1,1,0)
3.  --设置扩展 AO
4.  SetAuxAO(0,10,0)
5.  --等待扩展 DI 输入
6.  WaitAuxDI(0,0,1000,0)
7.  --等待扩展 AI 输入
8.  WaitAuxAI(0,0,50,1000,0)
9.  --获取扩展 DI 值
10. Ret = GetAuxDI(0,0)
11. --获取扩展 AI 值
12. Value = GetAuxAI(0,0)
```

3.3.7 坐标系

坐标系指令分为“设置工具坐标系”和“设置工件坐标系”两部分功能。

SetToolList:设置工具坐标系列表

表 3-106 SetToolList 详细参数

属性	说明
原型	SetToolList(name)
描述	设置工具坐标系列表
参数	• name:工具坐标系名称，如 toolcoord0。
返回值	无

SetWObjList：设置工件坐标系列表

表 3-107 SetWObjList 详细参数

属性	说明
原型	SetWObjList (name)
描述	设置工具坐标系列表
参数	• name:工件坐标系名称，如 wobjcoord0。
返回值	无

代码 3-32 坐标系示例

```
1.  SetWObjList(wobjcoord0) --设置工件坐标
2.  SetToolList(toolcoord0) --设置工具坐标
```

3.3.8 模式切换

通过 Mode 可以实现机器人模式的切换，该指令可切换机器人到手动模式，通常在一个程序结尾处添加，以便用户在程序运行结束后，使机器人自动切换到手动模式，拖动机器人。

Mode：机器人模式切换到手动模式

表 3-108 Mode 详细参数

属性	说明
原型	Mode(state)
描述	控制机器人切换到手动模式
参数	• state： 0-机器人模式，默认 1-手动模式。
返回值	无

代码 3-33 坐标系示例

```
1. Lin(DW01,100,-1,0,0)
2. Lin(DW02,100,-1,0,0)
3. Lin(DW03,100,-1,0,0)
4. Mode(1)
```

3.3.9 碰撞等级

设置碰撞等级，可以在程序运行中实时调节各轴碰撞等级，更灵活的部署应用场景。在自定义百分比模式下，1~100%，对应 0~100N。

SetAnticollision：碰撞等级设置

表 3-109 SetAnticollision 详细参数

属性	说明
原型	SetAnticollision (mode, level, config)
描述	设置碰撞等级
参数	• mode:0-标准等级，1-自定义百分比； • level={j1,j2,j3,j4,j5,j6}:碰撞阈值，共 11 个等级，1 为等级 1，2 为等级 2，100 为关闭碰撞等级； • config:0-不更新配置文件，1-更新配置文件，默认为 0。
返回值	无

代码 3-34 坐标系示例

```
1. level={4,4,4,4,4,5}
2. SetAnticollision(0, level,0)--设置碰撞等级，0-标准模式，level -各个关节碰撞等级，0-不更新配置文件
1. level1={40,40,40,40,40,50}
2. SetAnticollision(1, level1,0)--设置碰撞等级，1-自定义百分比模式，level -各个关节碰撞阈值，0-不更新配置文件
```

3.3.10 加速度

Acc 指令是实现机器人加速度可单独设置功能，通过调节运动指令加调试速度，可以增加或减小加减速时间，实现机器人动作节拍时间可调。

SetOaccScale，设置机器人加速度

表 3-110 SetOaccScale 详细参数

属性	说明
原型	SetOaccScale(acc)
描述	设置机器人加速度
参数	• acc:机器人加速度百分比。
返回值	无

代码 3-35 加速度示例

```
1.  SetOaccScale (20)--20-设置加速度百分比
```

3.4 外设指令

3.4.1 夹爪

ActGripper：夹爪激活/复位

表 3- 111 ActGripper 详细参数

属性	说明
原型	ActGripper(index,action)
描述	激活夹爪
参数	• index:夹爪编号； • action:0-复位，1-激活。
返回值	无

MoveGripper：夹爪运动控制参数

表 3-112 MoveGripper 详细参数

属性	说明
原型	MoveGripper (index, pos, vel, force, max_time, block)
描述	设置夹爪运动控制参数
参数	• index：夹爪编号，范围[1~8]； • pos:位置百分比，范围[0~100]； • vel:速度百分比，范围[0~100]； • force:力矩百分比，范围[0~100]； • max_time：最大等待时间，范围[0~30000]，单位 ms； • block:是否阻塞，0-阻塞，1-非阻塞。
返回值	无

代码 3-36 夹爪示例

```
1.  ActGripper(1,0)--夹爪复位，1-夹爪编号，0-复位
2.  WaitMs(1000)--等待 1000ms,确保夹爪复位成功
3.  ActGripper(1,1)--夹爪激活，1-夹爪编号，1-夹爪激活
4.  WaitMs(10)--等待 1000ms,确保夹爪复位成功
5.  MoveGripper(1,62,27,51,3000,1)
6.  --控制夹爪运动，1-夹爪编号，62-夹爪位置，27-夹爪开闭速度，51-夹爪开闭力矩，
    3000-夹爪最大等待时间，0-阻塞
```

3.4.2 喷枪

喷枪指令可以控制喷枪“开始喷涂”、“停止喷涂”、“开始清枪”和“停止轻枪”等动作。

SprayStart：喷涂开始

表 3-113 SprayStart 详细参数

属性	说明
原型	SprayStart ()
描述	喷涂开始
参数	无
返回值	无

SprayStop：停止喷涂

表 3-114 SprayStop 详细参数

属性	说明
原型	SprayStop ()
描述	喷涂停止
参数	无
返回值	无

PowerCleanStart：开始清枪

表 3-115 PowerCleanStart 详细参数

属性	说明
原型	PowerCleanStart ()
描述	开始清枪
参数	无
返回值	无

PowerCleanStop：清枪停止

表 3-116 PowerCleanStop 详细参数

属性	说明
原型	PowerCleanStop ()
描述	清枪停止
参数	无
返回值	无

代码 3-37 喷枪示例

```
1.  Lin(Spraystart,100,-1,0,0)--开始喷涂点，运动到喷涂起点
2.
3.  SprayStart()--开始喷涂
4.  Lin(Sprayline,100,-1,0,0)--喷涂路径
5.  Lin(template3,100,-1,0,0)--停止喷涂点
6.  SprayStop()--停止喷涂
7.
8.  Lin(template4,100,-1,0,0)--清枪点，运动到清枪点，等待清枪处理
9.
10. PowerCleanStart()--开始清枪
11. WaitMs(5000) --清枪时间  5000ms
12. PowerCleanStop()--停止清枪
```

3.4.3 扩展轴

扩展轴分为控制器-PLC（UDP）和控制器-伺服驱动器（485）两种模式。

EXT_AXIS_PTP：UDP 模式扩展轴运动

表 3-117 EXT_AXIS_PTP 详细参数

属性	说明
原型	EXT_AXIS_PTP (mode, name, Vel)
描述	UDP 模式扩展轴运动
参数	• mode：运动方式，0-异步，1-同步； • Name：点名称； • Vel：调试速度。
返回值	无

ExtAxisMoveJ：UDP 模式扩展轴运动

表 3-118 ExtAxisMoveJ 详细参数

属性	说明
原型	ExtAxisMoveJ (mode, E1，E2，E3，E4, Vel)
描述	UDP 模式扩展轴运动
参数	• mode：运动方式，0-异步，1-同步；； • E1，E2，E3，E4：外部轴位置 • Vel：调试速度。
返回值	无

ExtAxisSetHoming：UDP 扩展轴回零

表 3-119 ExtAxisSetHoming 详细参数

属性	说明
原型	ExtAxisSetHoming(axisID，mode，searchVel ，latchVel)
描述	UDP 扩展轴回零
参数	• axisID：轴号[1-4]； • mode：回零方式 0 当前位置回零，1 负限位回零，2-正限位回零； • searchVel 寻零速度(mm/s)； • latchVel：寻零箍位速度(mm/s)。
返回值	无

ExtAxisServoOn：UDP 扩展轴使能

表 3-120 ExtAxisServoOn 详细参数

属性	说明
原型	ExtAxisServoOn(axisID，status)
描述	UDP 扩展轴使能
参数	• axisID：轴号[1-4]； • status：0-去使能；1-使能。
返回值	无

代码 3- 38 UDP 扩展轴示例

```
1.  --UDP 扩展轴示例
2.  ExtDevSetUDPComParam("192.168.58.88",2021,10)--配置 UDP 通讯参数
```

代码 3-38（续）

```
3.  ExtDevLoadUDPDriver()--加载 UDP 驱动程序以启用通信
4.  WaitMs(500)--等待 500 毫秒，确保 UDP 驱动程序已正确加载
5.
6.  ExtAxisServoOn(1,0)--去使能，将 1 号轴去使能
7.  ExtAxisServoOn(1,1)--使能，将 1 号轴使能
8.  ExtAxisSetHoming(1,0,40,45)--回零，1-扩展轴编号，0-当前位置回零，40-寻零速度，
    45-寻零箍位速度
9.  WaitMs(1000)--等待 1000 毫秒
10.
11. EXT_AXIS_PTP(0,DW01,100)--运动指令，0-异步方式运动，DW01-点位名称，100-调
    试速度
12. WaitMs(1000)--等待 1000 毫秒
```

控制器-伺服驱动器（485）模式对扩展轴参数进行配置。

AuxServosetStatusID：设置状态反馈中 485 扩展轴数据轴号

表 3-121 AuxServosetStatusID 详细参数

属性	说明
原型	AuxServosetStatusID(servoId)
描述	设置状态反馈中 485 扩展轴数据轴号
参数	• servoId：伺服驱动器 ID，范围[1-15],对应从站 ID。
返回值	无

AuxServoEnable：设置 485 扩展轴是否使能

表 3-122 AuxServoEnable 详细参数

属性	说明
原型	AuxServoEnable(servoId, status)
描述	设置 485 扩展轴使能/去使能
参数	• servoId：伺服驱动器 ID，范围[1-15],对应从站 ID； • status：使能状态，0-去使能， 1-使能。
返回值	无

AuxServoSetControlMode，设置 485 扩展轴控制的模式

表 3-123 AuxServoSetControlMode 详细参数

属性	说明
原型	AuxServoSetControlMode(servoId, mode)
描述	设置 485 扩展轴控制模式
参数	• servoId：伺服驱动器 ID，范围[1-15],对应从站 ID； • mode：控制模式，0-位置模式，1-速度模式。
返回值	无

AuxServoHoming：设置 485 扩展轴回零方式

表 3-124 AuxServoHoming 详细参数

属性	说明
原型	AuxServoHoming(servoId, mode, searchVel, latchVel,)
描述	设置 485 扩展轴回零
参数	• servoId：伺服驱动器 ID，范围[1-15],对应从站 ID； • mode：回零模式，1-当前位置回零；2-负限位回零；3-正限位回零; • searchVel： 回零速度，mm/s 或°/s; • latchVel：箍位速度，mm/s 或°/s。
返回值	无

AuxServoSetTargetSpeed：速度模式下设置 485 扩展轴目标速度

表 3-125 AuxServoSetTargetSpeed 详细参数

属性	说明
原型	AuxServoSetTargetSpeed(servoId, speed, acc)
描述	设置 485 扩展轴目标速度(速度模式)
参数	• servoId：伺服驱动器 ID，范围[1-15],对应从站 ID； • speed：目标速度，mm/s 或°/s。
返回值	无

AuxServoSetTargetPos：位置模式下设置 485 扩展轴目标位置

表 3-126 AuxServoSetTargetPos 详细参数

属性	说明
原型	AuxServoSetTargetPos (servoId, pos, speed)
描述	设置 485 扩展轴目标位置(位置模式)

表 3-126（续表）

属性	说明
参数	• servoId：伺服驱动器 ID，范围[1-15],对应从站 ID； • pos:目标位置; • speed：目标速度，mm/s 或°/s。
返回值	无

代码 3-39 控制器+伺服驱动器轴示例

```
--控制器+伺服驱动器（位置模式）
AuxServoSetStatusID(1)--设置状态反馈中 485 扩展轴数据轴号
AuxServoEnable(1,0)--设置 485 扩展轴使能，1-伺服驱动器 ID，0-去使能
WaitMs(500)--等待 500 毫秒
AuxServoEnable(1,1)--设置 485 扩展轴使能，1-伺服驱动器 ID，1-使能
WaitMs(500)--等待 500 毫秒
AuxServoHoming(1,1,10,10)--设置 485 扩展轴回零方式，1-伺服驱动器 ID，1-当前位置回零，10-回零速度，10-箍位速度
WaitMs(500)--等待 500 毫秒
AuxServoSetTargetPos(1,300,30)--设置 485 扩展轴目标位置(位置模式)，1-伺服驱动器 ID，300-目标位置，30-目标速度
WaitMs(500)--等待 500 毫秒

--控制器+伺服驱动器（速度模式）
AuxServoSetStatusID(1)--设置状态反馈中 485 扩展轴数据轴号
AuxServoEnable(1,0)--设置 485 扩展轴使能，1-伺服驱动器 ID，0-去使能
WaitMs(500)--等待 500 毫秒
AuxServoSetControlMode(1,1)--设置 485 扩展轴控制模式，1-伺服驱动器 ID，1-速度模式
WaitMs(500)--等待 500 毫秒
AuxServoEnable(1,1)--设置 485 扩展轴使能，1-伺服驱动器 ID，1-使能
WaitMs(500)--等待 500 毫秒
AuxServoHoming(1,1,10,10)--设置 485 扩展轴回零方式，1-伺服驱动器 ID，1-当前位置回零，10-回零速度，10-箍位速度
WaitMs(500)--等待 500 毫秒
AuxServoSetTargetSpeed(1, 30)--设置 485 扩展轴目标位置(速度模式)，1-伺服驱动器 ID，30-目标速度
WaitMs(500)--等待 500 毫秒
```

3.4.4 传送带

ConveyorIODetect：IO 实时检测

表 3-127 ConveyorIODetect 详细参数

属性	说明
原型	ConveyorIODetect(max_t)
描述	传送带工件 IO 实时检测
参数	• max_t： 最大检测时间，单位 ms。
返回值	无

ConveyorGetTrackData：位置实时检测

表 3-128 ConveyorGetTrackData 详细参数

属性	说明
原型	ConveyorGetTrackData(mode)
描述	位置实时检测，获取位置当前状态
参数	• mode： 1-跟踪抓取 2-跟踪运动 3-TPD 跟踪。
返回值	无

ConveyorTrackStart：开启传动带跟踪

表 3-129 ConveyorTrackStart 详细参数

属性	说明
原型	ConveyorTrackStart(status)
描述	传动带跟踪开始
参数	• status： 状态，1-启动，0-停止。
返回值	无

ConveyorTrackEnd：停止传动带跟踪

表 3-130 ConveyorTrackEnd 详细参数

属性	说明
原型	ConveyorTrackEnd()
描述	传动带跟踪停止
参数	无
返回值	无

代码 3-40 传送带示例

```
1.  PTP(conveyorstart,30,-1,0)--机器人抓取起点
2.  While(1) do--循环抓取
3.      ConveyorIODetect(10000)--IO 实时检测物体
4.      ConveyorGetTrackData(1)--物体位置获取
5.      ConveyorTrackStart(1)--传送带跟踪开始
6.      Lin(cvrCatchPoint,10,-1,0,0)--机器人到达抓取点
7.      MoveGripper(1,255,255,0,10000)--夹爪抓取物体
8.      Lin(cvrRaisePoint,10,-1,0,0)--机器人提起
9.      ConveyorTrackEnd()--传送带跟踪结束
10.     PTP(conveyorraise,30,-1,0)--机器人到达等待点
11.     PTP(conveyorend,30,-1,0)--机器人到达放置点
12.     MoveGripper(1,0,255,0,10000)--夹爪松开
13.     PTP(conveyorstart,50,-1,0)--机器人再次回到抓取起点等待下次抓取
14. end--结束
```

3.4.5 打磨设备

PolishingUnloadComDriver：卸载打磨头通讯驱动

表 3-131 PolishingUnloadComDriver 详细参数

属性	说明
原型	PolishingUnloadComDriver ()
描述	打磨头通讯驱动卸载
参数	无
返回值	无

PolishingLoadComDriver：加载打磨头通讯驱动

表 3-132 PolishingLoadComDriver 详细参数

属性	说明
原型	PolishingLoadComDriver ()
描述	打磨头通讯驱动加载
参数	无
返回值	无

PolishingDeviceEnable：设备使能设置

表 3-133 PolishingDeviceEnable 详细参数

属性	说明
原型	PolishingDeviceEnable (status)
描述	打磨头设备使能
参数	• status：0-下使能，1-上使能。
返回值	无

PolishingClearError：错误清除

表 3-134 PolishingClearError 详细参数

属性	说明
原型	PolishingClearError ()
描述	清除打磨头设备错误信息
参数	无
返回值	无

PolishingTorqueSensorReset：打磨头力传感器清零

表 3-135 PolishingTorqueSensorReset 详细参数

属性	说明
原型	PolishingTorqueSensorReset ()
描述	打磨头力传感器清零。
参数	无
返回值	无

PolishingSetTargetVelocity：打磨头转速设置

表 3-136 PolishingSetTargetVelocity 详细参数

属性	说明
原型	PolishingSetTargetVelocity (rot)
描述	打磨头转速设置
参数	• rot：转速，单位[r/min]。
返回值	无

PolishingSetTargetTorque：设定力

表 3-137 PolishingSetTargetTorque 详细参数

属性	说明
原型	PolishingSetTargetTorque (setN)
描述	设置打磨头有设定力
参数	• setN：设定力，单位[N]。
返回值	无

PolishingSetTargetPosition：设置打磨头伸出距离

表 3-138 PolishingSetTargetPosition 详细参数

属性	说明
原型	PolishingSetTargetPosition (distance)
描述	设置打磨头伸出距离
参数	• distance：伸出距离，单位[mm]。
返回值	无

PolishingSetOperationMode：设置打磨头控制模式

表 3-139 PolishingSetOperationMode 详细参数

属性	说明
原型	PolishingSetTargetPosition (mode)
描述	打磨头模式设置
参数	• mode ：1-回零模式，2-位置模式，3-力矩模式。
返回值	无

PolishingSetTargetTouchForce：接触力设置

表 3-140 PolishingSetTargetTouchForce 详细参数

属性	说明
原型	PolishingSetTargetTouchForce (conN)
描述	接触力设置
参数	• conN：接触力，单位[N]。
返回值	无

PolishingSetTargetTouchTime：设定力过渡时间设置

表 3-141 PolishingSetTargetTouchTime 详细参数

属性	说明
原型	PolishingSetTargetTouchForceTime (settime)
描述	设定力过渡时间设置
参数	• settime：时间，单位[ms]。
返回值	无

PolishingSetWorkPieceWeight：工件重量设置

表 3-142 PolishingSetWorkPieceWeight 详细参数

属性	说明
原型	PolishingSetWorkPieceWeight (weight)
描述	工件重量设置
参数	• weight：重量，单位[N]。
返回值	无

代码 3-41 打磨设备示例

```
1.  PolishingLoadComDriver()-- 加载打磨头通讯驱动
2.  PolishingDeviceEnable(1) -- 设备上使能
3.  PolishingClearError(1)-- 清除打磨头设备错误信息
4.  PolishingTorqueSensorReset()-- 力传感器清零
5.  PolishingSetTargetVelocity(500) -- 设置打磨头转速
6.  PolishingSetTargetTorque(10) -- 设置打磨头设定力
7.  PolishingSetTargetPosition(100)-- 设置打磨头伸出距离
8.  PolishingSetOperationMode(3) -- 设置打磨头控制模式
9.  PolishingSetTargetTouchForce(5) -- 设置打磨头接触力
10. PolishingSetTargetTouchTime(500) -- 设置打磨头接触力过渡时间
11. PolishingSetWorkPieceWeight(20) -- 设置工件重量
12. PolishingUnloadComDriver()-- 卸载打磨头通讯驱动
```

3.5 焊接指令

3.5.1 焊接

WeldingSetCurrent：设置焊接电流

表 3-143 WeldingSetCurrent 详细参数

属性	说明
原型	WeldingSetCurrent(ioType,current,blend,AOIndex)
描述	设置焊接电流
参数	• ioType： 类型 0-控制器 IO； 1-数字通信协议； • current： 焊接电流值(A)； • blend:平滑，0-不平滑，1-平滑； • AOIndex： 焊接电流控制箱模拟量输出端口(0-1)，当模式为数字通讯协议时，blend 为 0，AOIndex 为 0。
返回值	无

WeldingSetVoltage：设置焊接电压

表 3-144 WeldingSetVoltage 详细参数

属性	说明
原型	WeldingSetVoltage(ioType,voltage, blend ,AOIndex)
描述	设置焊接电压
参数	• ioType： 类型 0-控制器 IO； 1-数字通讯协议； • voltage： 焊接电压值(V)； • blend:平滑，0-不平滑，1-平滑； • AOIndex：焊接电流控制 AO 端口(0-1)，当模式为数字通讯协议时，blend 为 0，AOIndex 为 0 协议时，blend 为 0，AOIndex 为 0。
返回值	无

ARCStart：起弧

表 3-145 ARCStart 详细参数

属性	说明
原型	ARCStart (ioType, arcNum, timeout)
描述	起弧
参数	• ioType： 类型 0-控制器 IO； 1-数字通讯协议； • arcNum： 焊接工艺编号； • timeout： 最大等待时间。
返回值	无

ARCEnd：收弧

表 3-146 ARCEnd 详细参数

属性	说明
原型	ARCEnd(ioType, arcNum, timeout)
描述	收弧
参数	• ioType： 类型 0-控制器 IO； 1-数字通讯协议； • arcNum： 焊接工艺编号； • timeout： 最大等待时间。
返回值	无

SetAspirated：送气

表 3-147 SetAspirated 详细参数

属性	说明
原型	SetAspirated(ioType, airControl)
描述	送气
参数	• ioType： 0-控制器 IO；1-数字通讯协议； • airControl： 送气控制 0-停止送气；1-送气。
返回值	无

SetReverseWireFeed：反向送丝

表 3-148 SetReverseWireFeed 详细参数

属性	说明
原型	SetReverseWireFeed(ioType, wireFeed)
描述	反向送丝
参数	• ioType： 0-控制器 IO；1-数字通讯协议； • wireFeed： 送丝控制 0-停止送丝；1-送丝。
返回值	无

SetForwardWireFeed：正向送丝

表 3-149 SetForwardWireFeed 详细参数

属性	说明
原型	SetForwardWireFeed(ioType, wireFeed)
描述	正向送丝

表 3-149（续表）

属性	说明
参数	• ioType： 0-控制器 IO；1-数字通讯协议 • wireFeed： 送丝控制 0-停止送丝；1-送丝
返回值	无

代码 3-42 焊接示例

```
1.  --控制器 IO 焊接
2.  weldIOType = 0--设置模式控制器 IO
3.  --设置电流电压
4.  WeldingSetCurrent(weldIOType,2,1,0)--电流设置，2-焊接电压 2A，1-焊接电流控制 AO 端口 1，0-不平滑
5.  WeldingSetVoltage(weldIOType,2,1,0)--电压设置，2-焊接电压 2A，1-焊接电流控制 AO 端口 1，0-不平滑
6.
7.  --运动到焊接起点
8.  PTP(mulitilinesafe,10,-1,0)
9.  PTP(mulitilineorigin1,10,-1,0)
10. --起弧
11. ARCStart(weldIOType,0,1000)--起弧，weldIOType-控制器 IO 模式，0-焊接工艺编号 0，1000-最大等待时间 1000ms
12. Lin(DW01,100,-1,0,0)
13. ARCEnd(weldIOType,0,1000)--收弧，weldIOType-控制器 IO 模式，0-焊接工艺编号 0，1000-最大等待时间 1000ms
14. --送气
15. SetAspirated(weldIOType,1)--送气，weldIOType-控制器 IO 模式，1-开启
16. Lin(DW01,100,-1,0,0)
17. SetAspirated(weldIOType,0)--停气，weldIOType-控制器 IO 模式，0-停止
18. WaitMs(1000)--等待 1000 毫秒
19. --正向送丝
20. SetForwardWireFeed(weldIOType,1)--正向送丝，weldIOType-控制器 IO 模式，1-开启
21. Lin(DW01,100,-1,0,0)
22. SetForwardWireFeed(weldIOType,0)--正向送丝，weldIOType-控制器 IO 模式，0-停止
23. WaitMs(1000)--等待 1000 毫秒
24. --反向送丝
25. SetReverseWireFeed(weldIOType,1)--反向送丝，weldIOType-控制器 IO 模式，1-开启
26. Lin(DW01,100,-1,0,0)
27. SetReverseWireFeed(weldIOType,0)--反向送丝，weldIOType-控制器 IO 模式，0-停止
28. WaitMs(1000)--等待 1000 毫秒
```

3.5.2 电弧跟踪

ArcWeldTraceControl：电弧跟踪控制

表 3-150 ArcWeldTraceControl 详细参数

属性	说明
原型	ArcWeldTraceControl(flag,delaytime, isLeftRight, klr, tStartLr, stepMaxLr, sumMaxLr, isUpLow, kud, tStartUd, stepMaxUd, sumMaxUd, axisSelect, referenceType, referSampleStartUd, referSampleCountUd, referenceCurrent)
描述	电弧跟踪控制
参数	• flag： 开关，0-关；1-开; • delayTime：滞后时间，单位 ms; • isLeftRight：左右偏差补偿 0-关闭，1-开启; • klr：左右调节系数(灵敏度); • tStartLr：左右开始补偿时间 cyc; • stepMaxLr：左右每次最大补偿量 mm; • sumMaxLr：左右总计最大补偿量 mm; • isUpLow：上下偏差补偿 0-关闭，1-开启; • kud：上下调节系数(灵敏度); • tStartUd：上下开始补偿时间 cyc; • stepMaxUd：上下每次最大补偿量 mm; • sumMaxUd：上下总计最大补偿量; • axisSelect：上下坐标系选择，0-摆动；1-工具；2-基座; • referenceType：上下基准电流设定方式，0-反馈；1-常数; • referSampleStartUd：上下基准电流采样开始计数(反馈)，cyc; • referSampleCountUd：上下基准电流采样循环计数(反馈)，cyc; • referenceCurrent：上下基准电流 mA。
返回值	无

ArcWeldTraceReplayStart：电弧跟踪加多层多道补偿开启

表 3-151 ArcWeldTraceReplayStart 详细参数

属性	说明
原型	ArcWeldTraceReplayStart ( )
描述	电弧跟踪加多层多道补偿开启
参数	无
返回值	无

ArcWeldTraceReplayEnd：

表 3-152 ArcWeldTraceReplayEnd 详细参数

属性	说明
原型	ArcWeldTraceReplayEnd ( )
描述	电弧跟踪加多层多道补偿关闭
参数	无
返回值	无

MultilayerOffsetTrsfToBase：偏移量坐标变化-多层多道焊接

表 3-153 MultilayerOffsetTrsfToBase 详细参数

属性	说明
原型	MultilayerOffsetTrsfToBase (pointO.x, pointO.y, pointO.z, pointX.x, pointX.y, pointX.z, pointZ.x, pointZ.y, pointZ.z, dx, dy, dry)
描述	偏移量坐标变化-多层多道焊接
参数	• pointO.x, pointO.y, pointO.z:基准点 O 笛卡尔位姿; • pointX.x, pointX.y, pointX.z:基准点 X 向偏移方向点笛卡尔位姿; • pointZ.x, pointZ.y, pointZ.z:基准点 Z 向偏移方向点笛卡尔位姿; • dx: x 方向偏移量，单位[mm]; • dy: x 方向偏移量，单位[mm]; • dry: 绕 y 轴偏移量，单位[°]。
返回值	offset_x,offset_y,offset_z,offset_rx,offset_ry,offset_rz:偏移量

代码 3-43 电弧跟踪示例

```
1.  --运动到焊接起始点位
2.  PTP(mulitilinesafe,10,-1,0)
3.  PTP(mulitilineorigin1,10,-1,0)
4.
5.  --焊接（第一道位置）
6.  ARCStart(1,0,3000)
7.  WeaveStart(0)
8.  ArcWeldTraceControl(1,0,1,0.06,5,5,50,1,0.06,5,5,55,0,0,4,1,10)
9.  Lin(mulitilineorigin2,1,-1,0,0)
10. ArcWeldTraceControl(0,0,1,0.06,5,5,50,1,0.06,5,5,55,0,0,4,1,10)
11. WeaveEnd(0)
12. ARCEnd(1,0,3000)
13. PTP(mulitilinesafe,10,-1,0)
```

代码 3-43（续）

```
14. Pause(0) --无功能
15.
16. --焊接（第二道位置）
17. offset_x,offset_y,offset_z,offset_rx,offset_ry,offset_rz = MultilayerOffsetTrsfToBase
    (mulitilineorigin1, mulitilineX1,mulitilineZ1,10,0,0)--偏移量坐标变化-多层多道焊接
18. PTP(mulitilineorigin1,10,-1,1,offset_x,offset_y,offset_z,offset_rx,offset_ry,offset_rz)
19. ARCStart(1,0,3000)
20. offset_x,offset_y,offset_z,offset_rx,offset_ry,offset_rz = MultilayerOffsetTrsfToBase
    (mulitilineorigin2,mulitilineX2,mulitilineZ2,10,0,0) --偏移量坐标变化-多层多道焊接
21. ArcWeldTraceReplayStart()--电弧跟踪加多层多道补偿开启
22.
23. Lin(mulitilineorigin2,2,-1,0,1,offset_x,offset_y,offset_z,offset_rx,offset_ry,offset_rz)
24. ArcWeldTraceReplayEnd()--电弧跟踪加多层多道补偿关闭
25. ARCEnd(1,0,3000)
26. PTP(mulitilinesafe,10,-1,0)
27. Pause(0) --无功能
28.
29. --焊接（第三道位置）
30. offset_x,offset_y,offset_z,offset_rx,offset_ry,offset_rz = MultilayerOffsetTrsfToBase
    (mulitilineorigin1,mulitilineX1,mulitilineZ1,0,10,0) --偏移量坐标变化-多层多道焊接
31. PTP(mulitilineorigin1,10,-1,1,offset_x,offset_y,offset_z,offset_rx,offset_ry,offset_rz)
32. ARCStart(1,0,3000)
33. offset_x,offset_y,offset_z,offset_rx,offset_ry,offset_rz = MultilayerOffsetTrsfToBase
    (mulitilineorigin2,mulitilineX2,mulitilineZ2,0,10,0) --偏移量坐标变化-多层多道焊接
34. ArcWeldTraceReplayStart()--电弧跟踪加多层多道补偿开启
35. Lin(mulitilineorigin2,2,-1,0,1,offset_x,offset_y,offset_z,offset_rx,offset_ry,offset_rz)
36. ArcWeldTraceReplayEnd()--电弧跟踪加多层多道补偿关闭
37. ARCEnd(1,0,3000)
38. PTP(mulitilinesafe,10,-1,0)
```

3.5.3 激光跟踪

激光器跟踪需要进行传感器的加载，传感器开启，进行激光跟踪，数据记录，传感器取点运动，寻位命令共同完成

LoadPosSensorDriver：传感器加载

表 3-154 LoadPosSensorDriver 详细参数

属性	说明
原型	LoadPosSensorDriver (choiceid)
描述	传感器功能选择加载
参数	• choiceid：功能编号，101-睿牛 RRT-SV2-BP, 102-创想 CXZK-RBTA4L,103-全视 FV-160G4-WD-PP-RL,104-同舟激光传感器,105-奥太激光传感器。
返回值	无

UnloadPosSensorDriver：传感器卸载

表 3-155 UnloadPosSensorDriver 详细参数

属性	说明
原型	UnloadPosSensorDriver (choiceid)
描述	传感器功能选择卸载
参数	• choiceid： 功能编号, 101-睿牛 RRT-SV2-BP, 102-创想 CXZK-RBTA4L, 103-全视 FV-160G4-WD-PP-RL, 104-同舟激光传感器, 105-奥太激光传感器。
返回值	无

LTLaserOn：打开传感器

表 3-156 LTLaserOn 详细参数

属性	说明
原型	LTLaserOn (Taskid)
描述	打开传感器
参数	• Taskid：选择焊缝类型（睿牛 RRT-SV2-BP，创想 CXZK-RBTA4L），选择任务号（全视 FV-160G4-WD-PP-RL，奥太激光传感器），选择解决方案（同舟激光传感器）。
返回值	无

LTLaserOff：关闭传感器

表 3-157 LTLaserOff 详细参数

属性	说明
原型	LTLaserOff ()

表 3-157（续表）

属性	说明
描述	关闭传感器
参数	无
返回值	无

LTTrackOn：开始跟踪

表 3-158 LTTrackOn 详细参数

属性	说明
原型	LTLaserOn (toolid)
描述	开始跟踪
参数	• toolid：坐标系名称。
返回值	无

LTTrackOff：关闭跟踪

表 3-159 LTTrackOff 详细参数

属性	说明
原型	LTLaserOff ( )
描述	关闭跟踪
参数	无
返回值	无

LaserSensorRecord：数据记录

表 3-160 LaserSensorRecord 详细参数

属性	说明
原型	LaserSensorRecord (features，time, speed)
描述	数据记录
参数	• features：功能选择，0-停止记录，1-实时跟踪，2-开始记录，3-轨迹复现（选择轨迹复现时，可以选择激光跟踪复现）； • time：等待时间； • speed：运行速度。
返回值	无

MoveLTR：激光跟踪复现

表 3-161 MoveLTR 详细参数

属性	说明
原型	MoveLTR ( )
描述	激光跟踪复现（只有数据记录选择轨迹复现后，才能使用此命令）
参数	无
返回值	无

LTSearchStart：开始寻位

表 3-162 LTSearchStart 详细参数

属性	说明
原型	LTSearchStart (refdirection, refdpion, ovl, length, max_time, toolid)
描述	开始寻位
参数	• refdirection：方向，0-+x，1- -x，2-+y，3- -y,4- +z,5- -z,6-指定方向（自定义参考点方向）； • refdpiont：方向点，当方向为 6 时，需要指定方向点，其他默认为{0，0，0，0，0，0}； • ovl:速度百分比，单位[%]； • length：长度，单位[mm]； • max_time：最大寻位时间，单位[ms]； • toolid：坐标系名称。
返回值	无

LTSearchStop：停止寻位

表 3-163 LTSearchStop 详细参数

属性	说明
原型	LTSearchStop ( )
描述	停止寻位
参数	无
返回值	无

代码 3-44 激光跟踪示例

```
1.  LoadPosSensorDriver(101) -- 加载传感器驱动
2.  LTLaserOn(1) -- 打开传感器
```

代码 3-44（续）

```
3.  LTTrackOn(1) -- 开始跟踪
4.  LaserSensorRecord(2, 5, 30) -- 记录数据
5.  LTSearchStart(0, 0, 50, 100, 5000, 1) -- 寻位
6.  MoveLTR()-- 激光跟踪复现
7.  LTSearchStop()-- 停止寻位
8.  LTTrackOff()-- 关闭跟踪
9.  LTLaserOff()-- 关闭传感器
10.
11. -- 卸载传感器驱动
12. UnloadPosSensorDriver(101)
```

3.5.4 激光记录

激光记录指令实现激光跟踪记录起点、终点取出功能，使机器人可以自动运动到起点位置，适用于从工件外部开始运动并进行激光跟踪记录的场合，同时上位机可获取记录数据中起点、终点的信息，用于后续运动。

MoveToLaserRecordStart：运动到焊缝开始点位

表 3-164 MoveToLaserRecordStart 详细参数

属性	说明
原型	MoveToLaserRecordStart ( )
描述	运动到焊缝开始点位
参数	无
返回值	无

MoveToLaserRecordEnd：运动到焊缝结束点位

表 3-165 MoveToLaserRecordEnd 详细参数

属性	说明
原型	MoveToLaserRecordEnd ( )
描述	运动到焊缝开始点位
参数	无
返回值	无

代码 3-45 激光记录示例

```
1.  Lin(recordStartPt,100,-1,0,0)--运动到焊缝开始位置
2.  LaserSensorRecord(2,10,30)--记录焊缝开始点位
3.  Lin(recordEndPt,100,-1,0,0)--运动到焊缝结束位置
4.  LaserSensorRecord(0,10,30)--记录焊缝结束点位
5.  MoveToLaserRecordStart(1,30)--运动到焊接开始点位
6.  ARCStart(0,0,1000)--起弧
7.  LaserSensorRecord(3,10,30)--焊缝轨迹复现
8.  MoveLTR()--焊缝线性移动
9.  ARCEnd(0,0,1000)--关弧
10. MoveToLaserRecordEnd(1,30)--运动到焊接结束点位
```

3.5.5 焊丝寻位

焊丝寻位指令一般应用于焊接场景中，需要焊机与机器人 IO 和运动指令相结合使用。

WireSearchStart：焊丝寻位开始

表 3-166 WireSearchStart 详细参数

属性	说明
原型	WireSearchStart (refPos, searchVel, searchDis, autoBackFlag, autoBackVel, autoBackDis, offectFlag)
描述	焊丝寻位开始
参数	• pedlocation：基准位置是否更新，0-不更新，1-更新； • searchVel： 寻位速度 %； • searchDis： 寻位距离 mm； • autoBackFlag： 自动返回标志，0-不自动；-自动； • autoBackVel： 自动返回速度 %； • autoBackDis： 自动返回距离 mm； • offectFlag： 1-带偏移量寻位；2-示教点寻位。
返回值	无

WireSearchEnd：焊丝寻位结束

表 3-167 WireSearchEnd 详细参数

属性	说明
原型	WireSearchEnd (refPos, searchVel, searchDis, autoBackFlag, autoBackVel, autoBackDis, offectFlag)
描述	焊丝寻位结束

表 3-167（续表）

属性	说明
参数	• pedlocation：基准位置是否更新，0-不更新，1-更新； • searchVel： 寻位速度 %； • searchDis： 寻位距离 mm； • autoBackFlag： 自动返回标志，0-不自动；-自动； • autoBackVel： 自动返回速度 %； • autoBackDis： 自动返回距离 mm； • offectFlag： 1-带偏移量寻位；2-示教点寻位。
返回值	无

GetWireSearchOffset：计算焊丝寻位偏移量

表 3-168 GetWireSearchOffset 详细参数

属性	说明
原型	GetWireSearchOffset (seamType, method, varNameRef, varNameRes)
描述	计算焊丝寻位偏移量
参数	• seamType： 焊缝类型； • method： 计算方法； • varNameRef： 基准点 1-6，“#”表示无点变量； • varNameRes： 接触点 1-6，“#”表示无点变量。
返回值	无

WireSearchWait：等待焊丝寻位完成

表 3-169 SetPointToDatabase 详细参数

属性	说明
原型	WireSearchWait(varname)
描述	等待焊丝寻位完成
参数	• varName： 接触点名称 “RES0” ~ “RES99”。
返回值	无

SetPointToDatabase：焊丝寻位接触点写入数据库

表 3-170 SetPointToDatabase 详细参数

属性	说明
原型	SetPointToDatabase (varName, pos)
描述	焊丝寻位接触点写入数据库

表 3-170（续表）

属性	说明
参数	• varName： 接触点名称 “RES0” ~ “RES99”； • pos：接触点数据 x, y, x, a, b, c。
返回值	无

代码 3- 46 焊丝寻位示例

```
1.  WireSearchStart(1,10,300,1,10,10,0)--焊丝寻位开始
2.  Lin(2dx1,10,0,0,0,0)--寻位参考点起点
3.  Lin(2dx2,10,0,1,0)--寻位参考点方向点
4.  WireSearchWait("REF0")--等待焊丝寻位完成
5.  Lin(2dy1,10,0,0,0,0)--寻位参考点起点
6.  Lin(2dy2,10,0,1,0)--寻位参考点方向点
7.  WireSearchWait("REF1")--等待焊丝寻位完成
8.  WireSearchEnd(1,10,300,1,10,10,0)--焊丝寻位结束
9.  WireSearchStart(0,10,300,1,10,10,0)
10. Lin(2dx1,10,0,0,0,0)--寻位起点
11. Lin(2dx2,10,0,1,0)--寻位方向点
12. WireSearchWait("RES0")
13. Lin(2dy1,10,0,0,0,0)--寻位起点
14. Lin(2dy2,10,0,1,0)--寻位方向点
15. WireSearchWait("RES1")
16. WireSearchEnd(0,10,300,1,10,10,0)
17. f1,x1,y1,z1,a1,b1,c1=GetWireSearchOffset(0,1,"REF0","REF1","#","#","#","#","RES0","
    RES1","#","#","#","#")--计算寻位偏移量
18. RegisterVar("number","f1")
19. RegisterVar("number","x1")
20. RegisterVar("number","y1")
21. RegisterVar("number","z1")
22. RegisterVar("number","a1")
23. RegisterVar("number","b1")
24. RegisterVar("number","c1")
25. PointsOffsetEnable(f1,x1,y1,z1,a1,b1,c1)--运动偏移
26. Lin(test1,10,0,0,0)
27. Lin(test2,10,0,0,0)
28. PointsOffsetDisable()
```

3.5.6 姿态调整

PostureAdjustOn：开启姿态调整

表 3-171 PostureAdjustOn 详细参数

属性	说明
原型	PostureAdjustOn (plate_type, direction_type={PosA, PosB, PosC}, time, paDisatance_1, inflection_type, paDisatance_2, paDisatance_3 , paDisatance_4, paDisatance_5)
描述	开启姿态调整
参数	• plate_type：板材类型，0-波纹板，1-瓦楞板，2-围栏板，4-波纹甲壳钢 • direction_type：运动方向，从左到右（direction_type 为 PosA, PosB, PosC），从右到左（direction_type 为 PosA, PosC, PosB） • time：姿态调整时间，单位[ms]； • paDisatance_1：第一段长度，单位[mm]； • inflection_type：拐点类型，0-由上往下，1-由下往上； • paDisatance_2：第二段长度，单位[mm]； • paDisatance_3：第三段长度，单位[mm]； • paDisatance_4：第四段长度，单位[mm]； • paDisatance_5：第五段长度，单位[mm]。
返回值	无

PostureAdjustOff：关闭姿态调整

表 3-172 PostureAdjustOff 详细参数

属性	说明
原型	PostureAdjustOff ( )
描述	关闭姿态调整
参数	无
返回值	无

代码 3-47 姿态调整示例

```
1.  --开启姿态调整
2.  PostureAdjustOn(0,PosA,PosB,PosC,1000,100,0,100,100,100,100)
3.
4.  PTP(DW01,100,10,0)
5.  --关闭姿态调整
6.  PostureAdjustOff()
```

3.6 力控指令

3.6.1 力控集

FT_Guard：碰撞检测

表 3-173 FT_Guard 详细参数

属性	说明
原型	FT_Guard (flag, tool_id, select_Fx, select_Fy, select_Fz, select_Tx, select_TY, select_Tz, Value_Fx，Value_Fy，Value_Fz，Value_Tx，Value_TY，Value_Tz，max_threshold_Fx,max_threshold_Fy,max_threshold_Fz, max_threshold_Tx, max_threshold_Ty, max_threshold_Tz, min_threshold_Fx, min_threshold_Fy, min_threshold_Fz, min_threshold_Tx, min_threshold_Ty, min_threshold_Tz)
描述	碰撞检测
参数	• flag：力矩开启标志，0-关闭碰撞守护，1-开启碰撞守护； • tool_id：坐标系名称； •select_Fx~ select_Tz: 选择六个自由度是否检测碰撞，0-不检测，1-检测, select_Tx 设定为不选择； • Value_Fx~ Value_Tz：六个自由度的当前值, Value_Tx 设定为 0； • max_threshold_Fx~ max_threshold_Tz: 六个自由度最大阈值，max_threshold_Tx 设为 0； • min_threshold_Fx~ min_threshold_Tz：六个自由度最小阈值，min_threshold_Tx 设为 0。
返回值	无

代码 3-48 力控集碰撞检测模式示例

```
1.  -- FT_Guard 碰撞检测
2.  FT_Guard(1,2,1,0,0,0,0,0,1,0.752,-3.173,0.001,0.001,0.004,5,0,0,0,0,0,2,0,0,0,0,0) --力/矩碰撞检测开启
3.  Lin(fguard1,100,-1,0,0)
4.  Lin(ftguard2,100,-1,0,0)--运动指令
5.  FT_Guard(0,2,1,0,0,0,0,0,1,0.752,-3.173,0.001,0.001,0.004,5,0,0,0,0,0,2,0,0,0,0,0) --力/矩碰撞检测关闭
```

FT_Control：恒力控制

表 3-174 FT_Control 详细参数

属性	说明
原型	FT_Control (flag, sensor_num, select, force_torque, gain, adj_sign, ILC_sign, max_dis, max_ang)
描述	恒力控制

表 3-174（续表）

属性	说明
参数	• flag：恒力控制开启标志，0-关，1-开； • sensor_num：力传感器编号； • select：六个自由度是否检测 fx,fy,fz,mx,my,mz，0-不生效，1-生效； • force_torque：检测力/力矩，单位 N 或 Nm； • gain：f_p,f_i,f_d,m_p,m_i,m_d,力 PID 参数，力矩 PID 参数； • adj_sign：自适应启停状态，0-关闭，1-开启； • ILC_sign: ILC 控制启停状态，0-停止，1-训练，2-实操； • max_dis：最大调整距离； • max_ang：最大调整角度。
返回值	无

代码 3-49 力控集恒力控制模式示例

```
1.  --FT_Control, 恒力控制
2.  FT_Control(1,1,0,0,1,0,0,0,0,0,-15,0,0,0,0.0001,0,0,0,0,0,1,0,50,0) --力/矩运动控制开启
3.  while(1) do
4.      Lin(ftcontrol1,30,-1,0,0)
5.      Lin(ftcontrol2,30,-1,0,0)
6.      Lin(ftcontrol1,30,-1,0,0)
7.  end --运动指令
8.  FT_Control(0,1,0,0,1,0,0,0,0,0,-15,0,0,0,0.0001,0,0,0,0,0,1,0,50,0)--力/矩运动控制关闭
```

FT_SpiralSearch：螺旋线插入

表 3-175 FT_SpiralSearch 详细参数

属性	说明
原型	FT_SpiralSearch(rcs, dr,ft ,max_t_ms, max_vel)
描述	螺旋线插入
参数	• rcs：参考坐标系，0-工具坐标系，1-基坐标系 • dr：每圈半径进给量，单位 mm 默认 0.7; • ft：力/扭矩阈值，fx,fy,fz,tx,ty,tz，范围[0~100]； • max_t_ms：最大探索时间，单位 ms; • max_vel：线速度最大值，单位 mm/s。
返回值	无

代码 3-50 力控集螺旋插入模式示例

```
1.  --FT_Spiral, 螺旋插入
2.  FT_Control(1,1,0,0,1,0,0,0,0,0,-10,0,0,0,0.0001,0,0,0,0,0,0,0,1000,0) --力/矩运动控制开启
3.  FT_SpiralSearch(0,2,1,60000,2) --螺旋插入
```

代码 3-50（续）

```
4.  FT_Control(0,1,0,0,1,0,0,0,0,0,-10,0,0,0,0.0005,0,0,0,0,0,0,0,1000,0) --力/矩运动控制关闭
```

FT_ComplianceStart：柔顺控制开启

表 3-176 FT_ComplianceStart 详细参数

属性	说明
原型	FT_ComplianceStart(p,force)
描述	柔顺控制开启
参数	• p: 位置调节系数或柔顺系数; • force：柔顺开启力阈值，单位 N。
返回值	无

FT_ComplianceStop：柔顺控制关闭

表 3-177 FT_ComplianceStop 详细参数

属性	说明
原型	FT_ComplianceStop ()
描述	柔顺控制关闭
参数	无
返回值	无

代码 3-51 力控集柔顺控制模式示例

```
1.  --柔顺控制
2.  while(1) do
3.      FT_ComplianceStart(0.001,10)--柔顺控制开启
4.      FT_Control(1,2,0,0,1,0,0,0,0,0,-30,0,0,0,0.001,0,0,0,0,0,0,0,1000,0) --力/矩运动控制开
    启
5.
6.      Lin(com1,30,-1,0,0)
7.      Lin(com2,30,-1,0,0)
8.      Lin(com1,30,-1,0,0)
9.      Lin(com2,30,-1,0,0)--运动指令
10.
11.     FT_Control(0,2,0,0,1,0,0,0,0,0,-10,0,0,0,0.005,0,0,0,0,0,1,0,1000,0) --力/矩运动控制关
    闭
12.
13.     FT_ComplianceStop()--柔顺控制关闭
14. end
```

FT_RotInsertion：旋转插入

表 3-178 FT_RotInsertion 详细参数

属性	说明
原型	FT_RotInsertion(rcs, angVelRot, ft, max_angle, orn, max_angAcc, rotorn)
描述	旋转插入
参数	• rcs：参考坐标系，0-工具坐标系，1-基坐标系； • angVelRot 旋转角速度，单位 deg/s； • ft：力或力矩阈值 (0~100)，单位 N 或 Nm; • max_angle 最大旋转角度，单位 deg； • orn：力/扭矩方向，1-沿 z 轴方向，2-绕 z 轴方向; • max_angAcc 最大旋转加速度，单位 deg/s^2，暂不使用，默认为 0； • rotorn：旋转方向，1-顺时针，2-逆时针。
返回值	无

代码 3-52 力控集旋转插入模式示例

```
1.  --FT_Rot, 旋转插入
2.  FT_Control(1,1,0,0,1,0,0,0,0,0,-30,0,0,0,0.0001,0,0,0,0,0,0,0,1000,0) --力/矩运动控制开启
3.  FT_RotInsertion(0,1,5,300,1,0,1) --旋转插入
4.  FT_Control(0,1,0,0,1,0,0,0,0,0,-30,0,0,0,0.0001,0,0,0,0,0,0,0,1000,0) --力/矩运动控制关闭
```

FT_LinInsertion：直线插入

表 3-179 FT_LinInsertion 详细参数

属性	说明
原型	FT_LinInsertion(rcs, ft, lin_v , lin_a , disMax, linorn)
描述	直线插入
参数	• rcs：参考坐标系，0-工具坐标系，1-基坐标系； • ft：力或力矩阈值 (0~100)，单位 N 或 Nm; • lin_v：直线速度，单位 mm/s 默认 1; • lin_a：直线加速度，单位 mm/s^2，暂不使用 默认 0； • disMax：最大插入距离，单位 mm; • linorn：插入方向:0-负方向，1-正方向。
返回值	无

代码 3-53 力控集直线插入模式示例

```
1.  --FT_Lin, 直线插入
2.  FT_Control(1,1,0,0,1,0,0,0,0,0,-5,0,0,0,0.0001,0,0,0,0,0,0,0,1000,0) --力/矩运动控制开启
```

代码 3-53（续）

```
3.  FT_LinInsertion(0,12,3,0,100,1) --直线插入
4.  FT_Control(0,1,0,0,1,0,0,0,0,0,-10,0,0,0,0.0005,0,0,0,0,0,0,0,1000,0) --力/矩运动控制关闭
```

FT_FindSurface：表面定位

表 3-180 FT_FindSurface 详细参数

属性	说明
原型	FT_FindSurface (rcs, dir, axis, lin_v, lin_a , disMax, ft)
描述	表面定位
参数	• rcs： 参考坐标系，0-工具坐标系，1-基坐标系； • dir：移动方向，1-正方向，2-负方向； • axis：移动轴，1-x，2-y，3-z； • lin_v：探索直线速度，单位 mm/s 默认 3; • lin_a：探索直线加速度，单位 mm/s^2 默认 0; • disMax：大探索距离，单位 mm; • ft：动作终止力阈值，单位 N。
返回值	无

FT_CalCenterStart：计算中间平面位置开始

表 3-181 FT_CalCenterStart 详细参数

属性	说明
原型	FT_CalCenterStart ()
描述	计算中间平面位置开始
参数	无
返回值	无

FT_CalCenterEnd：计算中间平面位置结束

表 3-182 FT_CalCenterEnd 详细参数

属性	说明
原型	FT_CalCenterEnd ()
描述	计算中间平面位置结束
参数	无
返回值	pos={x,y,z,rx,ry,rz}

代码 3-54 力控集各种模式下表面定位示例

```
1.  --FT_FindSurface,表面定位
2.  PTP(1,30,-1,0)--初始位置
3.  FT_FindSurface(0,1,3,1,0,100,5)--表面定位
4.  --FT_CalCenter,中心定位
5.  PTP(1,30,-1,0)--初始位置
6.  FT_CalCenterStart()--表面定位开始
7.  FT_Control(1,10,0,0,1,0,0,0,0,0,-10,0,0,0,0.00001,0,0,0,0,0,0,100,0)--力/矩运动控制开启
8.  FT_FindSurface(1,2,2,10,0,200,5)--定位平面 A
9.  FT_Control(0,10,0,0,1,0,0,0,0,0,-10,0,0,0,0.00001,0,0,0,0,0,0,100,0)--力/矩运动控制关闭
10. PTP(1,30,-1,0)--初始位置
11. FT_Control(1,10,0,0,1,0,0,0,0,0,-10,0,0,0,0.00001,0,0,0,0,0,0,100,0)--力/矩运动控制开启
12. FT_FindSurface(1,1,2,20,0,200,5)--定位平面 B
13. FT_Control(0,10,0,0,1,0,0,0,0,0,10,0,0,0,0.00001,0,0,0,0,0,0,100,0)-力/矩运动控制关闭
14. pos={}--定义数组 pos
15. pos = FT_CalCenterEnd()--获取定位中心笛卡尔位姿
16. MoveCart(pos,GetActualTCPNum(),GetActualWObjNum(),30,10,100,-1,0)--运动至定位的中
    心位置
```

FT_Click：点按力探测

表 3-183 FT_Click 详细参数

属性	说明
原型	FT_Click (ft, lin_v, lin_a, disMax)
描述	点按力探测
参数	• ft：力或力矩阈值 (0~100)，单位 N 或 Nm; • lin_v：直线速度，单位 mm/s 默认 1; • lin_a：直线加速度，单位 mm/s^2，暂不使用 默认 0; • disMax：最大插入距离，单位 mm。
返回值	无

代码 3-55 力控集各种模式下点按力探测示例

```
1.  --FT_Click,点按力探测
2.  PTP(1,30,-1,0)--初始位置
3.  FT_Click(0, 5, 5, 0, 100, 0)--点按力探测
```

3.6.2 扭矩记录

扭矩记录指令，实现扭矩实时记录碰撞检测功能。

TorqueRecordStart：扭矩记录开始

表 3-184 TorqueRecordStart 详细参数

属性	说明
原型	TorqueRecordStart (flag, negativeValues, positiveValues，collisionTime)
描述	扭矩记录开始/停止
参数	• flag：平滑选择，0-不平滑，1-平滑； • negativeValues：各个关节的负阈值{j1,j2,j3,j4,j5,j6}； • positiveValues: 各个关节的正阈值{j1,j2,j3,j4,j5,j6}； • collisionTime：各个关节的碰撞检测持续时间{j1,j2,j3,j4,j5,j6}。
返回值	无

TorqueRecordEnd：扭矩记录停止

表 3-185 TorqueRecordEnd 详细参数

属性	说明
原型	TorqueRecordEnd ( )
描述	扭矩记录停止
参数	无
返回值	无

TorqueRecordReset：扭矩记录复位

表 3-186 TorqueRecordReset 详细参数

属性	说明
原型	TorqueRecordReset ( )
描述	扭矩记录复位
参数	无
返回值	无

代码 3-56 扭矩记录示例

```
1.  negativeValues = {-0.1, -0.1, -0.1, -0.1, -0.1, -0.1}
2.  positiveValues = {0.1, 0.1, 0.1, 0.1, 0.1, 0.1}
3.  collisionTime = {500, 500, 500, 500, 500, 500}
4.  TorqueRecordStart(1,negativeValues,positiveValues,collisionTime)
5.  TorqueRecordEnd()
6.  WaitMs(1000)--等待 1000 毫秒
7.  TorqueRecordReset()
```

3.7 通讯指令

3.7.1 Modbus

1） Modbus-TCP

Modbus 指令功能为基于 Modbus-TCP 协议的总线功能，用户可以通过相关指令控制机器人与 ModbusTCP client 或 server 通讯（主站与从站通讯），对线圈，离散量，寄存器进行读写操作。

ModbusMasterWriteDO：写数字输出（写线圈）

表 3-187 ModbusMasterWriteDO 详细参数

属性	说明
原型	ModbusMasterWriteDO (Modbus_name, Register_name, Register_num, {Register_Value})
描述	Modbus-TCP 写数字输出
参数	• Modbus_name：Modbus 的主站名称； • Register_name：DO 名称； • Modbus_name：Modbus 的主站名称； • Register_name：DO 名称； • Register_num: 寄存器数量； • {Register_Value}：寄存器值，寄存器值的数量与寄存器数量一致 {value_1, value_2, … }。
返回值	无

ModbusMasterReadDO：读数字输出（读线圈）

表 3-188 ModbusMasterReadDO 详细参数

属性	说明
原型	ModbusMasterReadDO (Modbus_name, Register_name, Register_num)
描述	Modbus-TCP 写数字输出
参数	• Modbus_name：Modbus 的主站名称； • Register_name：DO 名称； • Register_num: 寄存器数量。
返回值	Reg_value1, Reg_value2, …：int 值，根据 Register_num 的值返回相对应数量的值

ModbusMasterReadDI：读数字输入（读离散输入）

表 3-189 ModbusMasterReadDI 详细参数

属性	说明
原型	ModbusMasterReadDI (Modbus_name, Register_name, Register_num)
描述	Modbus-TCP 读数字输入
参数	• Modbus_name：Modbus 的主站名称； • Register_name：DI 名称； • Register_num: 寄存器数量；
返回值	Reg_value1, Reg_value2, …：int 值，根据 Register_num 的值返回相对应数量的值

代码 3-57 数字输入输出示例

```
1.  ModbusMasterWriteDO(Modbus_0,Register_1,1,{1})
2.  --写数字输出，Modbus_0-主站名称，Register_1-DO 名称，1-寄存器数量，{1}-寄存器值
3.  DO_value = ModbusMasterReadDO(Modbus_0,Register_1,1)
4.  --读数字输出，Modbus_0-主站名称，Register_1-DO 名称,1-寄存器数量
5.  DI_value = ModbusMasterReadDI(Modbus_0,Register_2,1)
6.  --读数字输出，Modbus_0-主站名称，Register_2-DI 名称,1-寄存器数量
```

ModbusMasterWriteAO：写模拟输出（保持寄存器）

表 3-190 ModbusMasterWriteAO 详细参数

属性	说明
原型	ModbusMasterWriteAO (Modbus_name, Register_name, Register_num, {Register_Value})
描述	Modbus-TCP 写模拟输出
参数	• Modbus_name：Modbus 的主站名称 • Register_name：AO 名称； • Register_num: 寄存器数量； • {Register_Value}：寄存器值，寄存器值的数量与寄存器数量一致 {value_1, value_2, … }。
返回值	无

ModbusMasterReadAO：读模拟输出（读保持寄存器）

表 3-191 ModbusMasterReadAO 详细参数

属性	说明
原型	ModbusMasterReadAO (Modbus_name, Register_name, Register_num)
描述	Modbus-TCP 读模拟输出
参数	• Modbus_name：Modbus 的主站名称； • Register_name：AO 名称； • Register_num: 寄存器数量。
返回值	reg_value：寄存器值

ModbusMasterReadAI：读模拟输入（读输入寄存器）

表 3-192 ModbusMasterReadAI 详细参数

属性	说明
原型	ModbusMasterReadAI (Modbus_name, Register_name, Register_num)
描述	Modbus-TCP 读输入寄存器
参数	• Modbus_name：Modbus 的主站名称； • Register_name：AO 名称； • Register_num: 寄存器数量。
返回值	Reg_value1, Reg_value2, …：int 值，根据 Register_num 的值返回相对应数量的值

代码 3-58 模拟输入输出示例

```
1.  --模拟输出设置
2.  ModbusMasterWriteAO(Modbus_0,Register_3,1,{2})
3.  --写模拟输出，Modbus_0-主站名称，Register_3-AO 名称，1-寄存器数量，{2}-寄存器值
4.  --模拟输出设置
5.  ModbusMasterWriteAO(Modbus_0,Register_3,1,{2})
6.  --写模拟输出，Modbus_0-主站名称，Register_3-AO 名称，1-寄存器数量，{2}-寄存器值
7.  AO_value = ModbusMasterReadAO(Modbus_0,Register_3,1)
8.  --读模拟输出，Modbus_0-主站名称，Register_3-AO 名称，1-寄存器数量
9.  AI_value = ModbusMasterReadAO(Modbus_0,Register_2,1)
10. --读模拟输入，Modbus_0-主站名称，Register_2-AI 名称，1-寄存器数量
```

ModbusMasterWaitDI，等待数字输入设置（等待离散输入值）

表 3-193 ModbusMasterWaitDI 详细参数

属性	说明
原型	ModbusMasterWaitDI (Modbus_name, Register _name, Waiting_state, Waiting_time)
描述	Modbus-TCP 等待 s 数字输入设置
参数	• Modbus_name：Modbus 的主站名称； • Register_name：DI 名称； • Waiting_state: 等待状态，1-Ture,0-Flase； • Waiting_time：超时时间单位[ms]。
返回值	无

ModbusMasterWaitAI：等待模拟输入设置（等待输入寄存器值）

表 3-194 ModbusMasterWaitAI 详细参数

属性	说明
原型	ModbusMasterWaitAI (Modbus_name, Register_name, Waiting_state, Register_Value， Waiting_time)
描述	Modbus-TCP 等待模拟输入设置
参数	• Modbus_name：Modbus 的主站名称； • Register_name：AI 名称； • Waiting_state: 等待状态，1 - <, 0 - >； • Register_Value:寄存器值； • Waiting_time：超时[ms]。
返回值	无

代码 3-59 等待数字/模拟输入输出示例

```
1. ModbusMasterWaitDI(Modbus_0, Register_0, 1, 1000)
2. --Modbus_0-主站名称，Register_0-DI 名称，1-True,1000-超时时间 ms
3. ModbusMasterWaitAI(Modbus_0,Register_2,0,13,1000)
4. --Modbus_0-主站名称，Register_2-AI 名称,0->，13-寄存器值，1000-超时时
   间 ms
```

ModbusSlaveWriteDO：从站数字输出设置（写离散输入）

表 3-195 ModbusSlaveWriteDO 详细参数

属性	说明
原型	ModbusSlaveWriteDO (Register_name, Register_num, {Register_Value})
描述	Modbus-TCP 从站写数字输出设置
参数	• Register_name：DO 名称； • Register_num: 寄存器数量； • {Register_Value}：寄存器值，寄存器值的数量与寄存器数量一致{value_1, value_2, … }。
返回值	无

ModbusSlaveReadDO：读数字输出（读离散输入）

表 3-196 ModbusSlaveReadDO 详细参数

属性	说明
原型	ModbusSlaveReadDO (Register_name, Register_num)
描述	Modbus-TCP 从读写数字输出
参数	• Register_name：DO 名称； • Register_num: 寄存器数量；
返回值	{Register_Value}：寄存器值，寄存器值的数量与寄存器数量一致{value_1, value_2, … }

ModbusSlaveReadDI：读数字输入（读线圈）

表 3-197 ModbusSlaveReadDI 详细参数

属性	说明
原型	ModbusMasterReadDI (Register_name, Register_num)
描述	Modbus-TCP 从站读数字输入
参数	• Register_name：DI 名称； • Register_num: 寄存器数量。
返回值	Reg_value1, Reg_value2, …：根据 Register_num 的值返回相对应数量的值

代码 3-60 从站数字输入/输出设置

```
1.  --从站数字输出设置
2.  ModbusSlaveWriteDO(DO0,1,{2})
```

代码 3-60（续）

```
3.  --写数字输出，DO0-DO 号，1-寄存器数量，{2}-寄存器值
4.  DO_value = ModbusSlaveReadDO(DO0,1)
5.  --读数字输出，DO0-DO 号，1-寄存器数量
6.
7.  --数字输入设置
8.  ModbusSlaveReadDI(DI1,3)
9.  --读数字输入，DI1-DI 名称，3-寄存器数量
```

ModbusSlaveWaitDI：等待数字输入设置（等待线圈值）

表 3-198 ModbusSlaveWaitDI 详细参数

属性	说明
原型	ModbusSlaveWaitDI (Register_name, Waiting_state, Waiting_time)
描述	Modbus-TCP 等待数字输入设置
参数	• Register_name：DI 名称； • Waiting_state: 等待状态，1-Ture,0-Flase； • Waiting_time：等待时间单位[ms]。
返回值	无

ModbusSlaveWaitAI：等待模拟输入设置（等保持寄存器值）

表 3-199 ModbusSlaveWaitAI 详细参数

属性	说明
原型	ModbusSlaveWaitAI (Register_name, Waiting_state,Register_Value , Waiting_time)
描述	Modbus-TCP 从站等待模拟输入设置
参数	• Register_name：AI 名称； • Waiting_state: 等待状态，1-<, 0->； • Register_Value:寄存器值； • Waiting_time：超时[ms]。
返回值	无

代码 3-61 从站等待数字/模拟输入设置

```
1.  ModbusSlaveWaitDI(DI2,0,100)
2.  --等待数字输入设置:DI2-DI 名称，0-false，100-等待时间 ms
3.  ModbusSlaveWaitAI(AI1,0,12,133))
4.  --AI1-AI 名称，0->,12-寄存器值，133-超时时间 ms
```

ModbusRegRead：读寄存器指令

表 3-200 ModbusRegRead 详细参数

属性	说明
原型	ModbusRegRead（fun_code，reg_add，reg_num，add,isthread）
描述	读取寄存器指令
参数	•fun_code： 功能码，1-0x01-线圈，2-0x02-离散量，3-0x03-保持寄存器，4-0x04-输入寄存器； • reg_add： 寄存器地址； • reg_num： 寄存器数量； • add：地址； • isthread：是否应用线程，0-否，1-是。
返回值	无

ModbusRegGetData：读寄存器数据

表 3-201 ModbusRegGetData 详细参数

属性	说明
原型	ModbusRegGetData（reg_num,isthread）
描述	读取寄存器数据
参数	• reg_num： 寄存器数量； • isthread：是否应用线程，0-否，1-是。
返回值	reg_value: 数组变量

ModbusRegWrite：写寄存器

表 3-202 ModbusRegWrite 详细参数

属性	说明
原型	ModbusRegWrite（fun_code，reg_add，reg_num，reg_value，add, isthread）
描述	写寄存器
参数	•fun_code 功能码，5-0x05-单个线圈，6-0x06-单个寄存器，15-0x0f-多个线圈，16-0x10-多个寄存器； • reg_add：单个线圈、单个寄存器、多个线圈、多个寄存器寄存器地址； • reg_num： 寄存器数量； • reg_value： 字节数组； • add：地址； • isthread：是否应用线程，0-否，1-是。
返回值	无

代码 3-62 Modbus-RTU 指令示例

```
1.  addr = 0x1000
2.  va1 = {400, 600, 900, 700}
3.  ret = {}
4.  ModbusRegWrite(10, addr, 4,va1, 1, 0)
5.  --1-0x10-多个寄存器，addr-寄存器地址，4-寄存器数量，va1-字节数组，1-地址，0-不
    应用线程
6.  WaitMs(10)
7.  ModbusRegRead(4, addr, 4, 1, 0)
8.  --1-0x04-输入寄存器，addr-寄存器地址，4-寄存器数量，1-地址，0-不应用线程
9.  WaitMs(10)
10. ret = ModbusRegGetData(4, 0)
11. --读寄存器数据，4-寄存器数量，0-不应用线程
12. WaitMs(10)
```

3.7.2 Xmlrpc

Xmlrpc 是一种通过 sockets 使用 xml 在程序之间传输数据的远程过程调用方法。通过这种方法，机器人控制器可以在远端的程序/服务调用功能函数（可带参数）并获取返回的结构性数据。

XmlrpcClientCall：数据远程调用

表 3-203 XmlrpcClientCall 详细参数

属性	说明
原型	XmlrpcClientCall (url, func, type， func_ Para)
描述	数据远程调用
参数	• url：服务端 url； • func: 调用函数； • type:传入参数的类型，1-double 类型数组，2-string 类型数组； • func_ Para：调用函数参数。
返回值	无

代码 3-63 Xmplrpc 指令示例

```
1.  --double 型数组示例：
2.  xmlrpcclientcall("http://192.168.58.20:50000/rpc2","example.array",1,{1.0,2.0,3.0})
3.  --http://192.168.58.20:50000/rpc2-服务端 url，example.array-调用函数名，1-类型，
    {1.0,2.0,3.0}-调用函数参数
4.  --string 型数组示例：
5.  xmlrpcclientcall("http://192.168.58.20:50000/rpc2","example.array",2,{"hello","world"})
```

代码 3-63（续）

```
6.  --http://192.168.58.20:50000/rpc2-服务端 url，example.array-调用函数名，0-类型，
    {1.0,2.0,3.0}-调用函数参数
```

3.8 辅助指令

3.8.1 辅助线程

FRLua 提供辅助线程功能，用户可以定义一个辅助线程与主线程同时运行，辅助线程主要与外部设备进行数据交互，

NewAuxThread：创建辅助线程

表 3-204 NewAuxThread 详细参数

属性	说明
原型	NewAuxThread (func_name, func_ Para)
描述	创建辅助线程
参数	• func_name: 调用函数； • func_ Para：调用函数参数。
返回值	无

代码 3-64 辅助线程示例

```
1.  --辅助线程函数定义
2.  function auxThread_TCPCom(ip, port)
3.      local flag = 0
4.      SetSysNumber(1, 0)--系统变量 1 赋值 0
5.      while 1 do
6.          if flag == 0 then
7.              flag = SocketOpen(ip,port, "socket_0")--与服务端建立连接
8.          elseif flag == 1 then
9.              SocketSendString("hello world","socket_0",1)
10.             n,svar = SocketReadAsciiFloat(1,"socket_0",0)--与服务端交互数据
11.             if n == 1 then
12.                 SetSysNumber(1, svar)--系统变量 1 赋值 svar
13.             end
14.         end
15.     end
16. end
17. --创建辅助线程
```

代码 3-64（续）

```
18.  NewAuxThread(auxThread_TCPCom, {"127.0.0.1",8010})
19.  WaitMs(100)
20.  while 1 do
21.      v   = GetSysNumber(1)--获取系统变量 1 值
22.      if v == 100 then
23.          PTP(P1,10,0,0)
24.      elseif v == 200 then
25.          PTP(P2,10,0,0)
26.      end
27.  end
```

3.8.2 调用函数

FRLua 将机器人接口函数提供给客户选择，并提示客户该函数所需要的参数，方便客户编写脚本指令

例如提供的 GetInverseKinRef 和 GetInverseKinHasSolution 函数。

GetInverseKinRef：逆运动学求解-指定位置参考

表 3-205 GetInverseKinRef 详细参数

属性	说明
原型	GetInverseKinRef (type, desc_pos, joint_pos_ref)
描述	逆运动学，工具位姿求解关节位置，参考指定关节位置求解
参数	•type:0-绝对位姿(基坐标系)，1-相对位姿（基坐标系），2-相对位姿（工具坐标系）； • desc_pos：{x,y,z,rx,ry,rz}工具位姿，单位[mm][°]； • joint_pos_ref：{j1,j2,j3,j4,j5,j6}，关节参考位置，单位[°]。
返回值	j1,j2,j3,j4,j5,j6:关节位置，单位[°]

GetInverseKinHasSolution，逆运动学求解-是否有解

表 3-206 GetInverseKinHasSolution 详细参数

属性	说明
原型	GetInverseKinHasSolution (type, desc_pos, joint_pos_ref)
描述	逆运动学，工具位姿求解关节位置 是否有解
参数	•type:0-绝对位姿(基坐标系)，1-相对位姿（基坐标系），2-相对位姿（工具坐标系）

表 3-206（续表）

属性	说明
参数	• desc_pos：{x,y,z,rx,ry,rz}工具位姿，单位[mm][°]； • joint_pos_ref：{j1,j2,j3,j4,j5,j6]，关节参考位置，单位[°]。
返回值	result:“True”-有解，“False”-无解

代码 3-65 调用函数示例

```
1.  J1={95.442,-101.149,-98.699,-68.347,90.580,-47.174}
2.  P1={75.414,568.526,338.135,-178.348,-0.930,52.611]
3.  ret_1 = GetInverseKinRef(0,P1,J1)
4.  --逆运动学求解-指定参考位置,0-绝对位姿(基坐标系)，P1-工具位姿，J1-关节参考位置
5.  ret_2 = GetInverseKinHasSolution(0,P1,J1)
6.  --逆运动学求解-是否有解,0-绝对位姿(基坐标系)，P1-工具位姿，J1-关节参考位置
```

3.8.3 点位表

PointTableSwitch：点位切换

表 3-207 PointTableSwitch 详细参数

属性	说明
原型	PointTableSwitch(point_table_name)
描述	点位表切换
参数	• point_table_name：要切换的点位表名称 pointTable1.db,当点位表为空，即“ ”时，表示将 lua 程序更新为未应用点位表的初始程序，系统模式。
返回值	无

代码 3-66 点位表示例

```
1.  PointTableSwitch("point_table_a.db")
```