



# Dobot TCP/IP 远程控制 接口文档



# 目录

前言

1 概述

2 Dashboard指令

2.1 控制相关指令

2.2 设置相关指令

2.3 计算和获取相关指令

2.4 IO相关指令

2.5 Modbus相关指令

3 运动指令

3.1 通用说明

3.2 指令列表

4 实时反馈信息

5 通用错误码

# 前言

## 目的

本手册介绍了Dobot四轴工业机械臂控制器TCP/IP二次开发接口及其使用方式，帮助用户了解和开发基于TCP/IP的机械臂控制软件。

## 读者对象

本手册适用于：

- 客户
- 销售工程师
- 安装调测工程师
- 技术支持工程师

## 修订记录

时间	修订记录
2024/01/12	拆分四轴控制器的TCP接口说明为独立的文档，对应控制器版本1.5.9.0

# 1 概述

由于基于TCP/IP的通讯具有成本低、可靠性高、实用性强、性能高等特点，许多工业自动化项目对基于TCP/IP协议控制机器人的需求广泛，因此Dobot机器人在TCP/IP协议的基础上，提供了丰富的接口用于与外部设备的交互。

关于TCP/IP协议的支持，控制器版本需V1.5.5.0及以上。

## 端口说明

根据设计，Dobot工业机器人会开启29999、30003、30004、30005以及30006服务器端口；

- 29999服务器端口：上位机可以通过29999端口直接发送一些**设置相关指令**给机器人，或者**主动获取**机器人的某些状态，这些功能被称为Dashboard。
- 30003服务器端口：上位机可以通过30003端口直接发送一些机器人**运动相关指令**给机器人，控制机器人进行运动。
- 30004、30005以及30006服务器端口：30004端口即实时反馈端口，客户端**每8ms**能收到一次机器人实时状态信息。30005端口**每200ms**反馈机器人的信息。30006端口为**可配置**的反馈机器人信息端口(默认为**每50ms**反馈，如需修改，请联系技术支持)。通过实时反馈端口每次收到的数据包有1440个字节，这些字节以标准的格式排列。

### 说明：

控制器1.5.9及以上版本可配置30005端口的反馈周期，如需修改，请联系技术支持。

## 消息格式

消息命令与消息应答都是 ASCII 码格式(字符串形式)。

上位机**下发消息**格式如下：

消息名称 (Param1,Param2,Param3.....ParamN)

由消息名称和参数组成，参数放在括号内，每一个参数之间以英文逗号“,”相隔，一个完整的消息以右括号结束。

TCP/IP远程控制指令不区分大小写格式，如以下三种写法都会被识别为使能机器人的指令：

- ENABLEROBOT()
- enablerobot()
- eNabLErobOt()

机器人收到命令后，会返回**应答消息**，格式如下：

```
ErrorID,{value,...,valueN},消息名称(Param1,Param2,Param3.....ParamN);
```

- ErrorID 为0时表示命令接收成功，返回非0则代表命令有错误，详见[通用错误码](#)；
- {value,...,valueN} 表示返回值，没有返回值则返回{}；
- 消息名称(Param1,Param2,Param3.....ParamN) 为下发的命令消息。

例如：

下发：

```
MovL(-500,100,200,150)
```

返回：

```
0,{},MovL(-500,100,200,150);
```

0表示接收成功，{}表示没有返回值。

下发：

```
Mov(-500,100,200,150)
```

返回：

```
-10000,{},Mov(-500,100,200,150);
```

-10000表示命令不存在，{}表示没有返回值。

## 立即指令与队列指令

TCP/IP远程控制指令分为立即指令和队列指令：

- 立即指令下发后会被立刻执行，并返回执行结果。
- 队列指令下发后会立刻返回，但不会被立刻执行，而是进入后台算法队列排队，等待被执行。

Dashboard指令（29999端口下发）大部分为立即指令，部分与运动和IO相关的指令为队列指令，本文档会在指令标题中特别标注。

运动相关指令（30003端口下发）均为队列指令。

如果在队列指令后面调用立即指令，立即指令可能会在队列指令完成之前执行，如下面的例子：

```
MovJ(-100,100,200,150) // 队列指令  
RobotMode() // 立即指令
```

在这个例子中，RobotMode() 会在机器人完成运动前执行，返回的状态为7（运动中）。

如果希望确保立即指令执行时前序指令都已执行完毕，可在调用立即指令前先调用Sync()指令，该指令会阻塞程序执行直到前序的指令全部执行完毕，如下面的例子：

```
MovJ(-500,100,200,150) // 队列指令  
Sync()  
RobotMode() // 立即指令
```

在这个例子中，RobotMode() 会在机器人完成运动后执行，返回的状态为5（空闲）。

## 获取DEMO

越疆提供了各种编程语言的二次开发DEMO，托管于[Github](#)上，请自行获取所需的DEMO，并参照DEMO进行二次开发。

## 2 Dashboard指令

Dashboard指令需要通过**29999**端口下发。

## 2.1 控制相关指令

### EnableRobot (立即指令)

#### 原型

```
EnableRobot(load,centerX,centerY,centerZ)
```

#### 描述

使能机械臂。执行队列指令（机械臂运动、队列IO等）前必须先使能机械臂。

#### 参数

参数名	类型	说明
load	double	设置负载重量，取值范围不能超过各个型号机器人的负载范围。单位：kg
centerX	double	X方向偏心距离。取值范围：-500 ~ 500，单位：mm
centerY	double	Y方向偏心距离。取值范围：-500 ~ 500，单位：mm
centerZ	double	Z方向偏心距离。取值范围：-500 ~ 500，单位：mm

均为可选参数，可携带的参数数量如下：

- 0：不携带参数，表示使能时不设置负载重量和偏心参数。
- 1：携带一个参数，该参数表示负载重量。
- 4：携带四个参数，分别表示负载重量和偏心参数。

#### 返回

```
ErrorID,{},EnableRobot(load,centerX,centerY,centerZ);
```

#### 示例

```
EnableRobot()
```

使能机器人，不设置负载重量和偏心参数。

```
EnableRobot(0.5)
```

使能机器人并设置负载重量0.5kg。



```
EnableRobot(0.5,0,0,5.5)
```

使能机器人并设置负载重量1.5kg，Z方向偏心5.5mm。

## DisableRobot（立即指令）

### 原型

```
DisableRobot()
```

### 描述

下使能机器人。

### 返回

```
ErrorID, {}, DisableRobot();
```

### 示例

```
DisableRobot()
```

下使能机器人。

## ClearError（立即指令）

### 原型

```
ClearError()
```

### 描述

清除机器人报警。清除报警后，用户可以根据RobotMode来判断机器人是否还处于报警状态。部分报警需要解决报警原因或者重启控制柜后才能清除。



#### 说明：

清除报警后，需要通过Continue指令重新开启运动队列。

### 返回

```
ErrorID, {}, ClearError();
```

## 示例

```
ClearError()
```

清除机器人报警。

## ResetRobot（立即指令）

### 原型

```
ResetRobot()
```

### 描述

停止机器人，清空已规划的指令队列。

### 返回

```
ErrorID, {}, ResetRobot();
```

## 示例

```
ResetRobot()
```

停止机器人并清空已规划的指令队列。

## RunScript（立即指令）

### 原型

```
RunScript(projectName)
```

### 描述

运行指定工程。

### 参数

参数名	类型	说明
projectName	string	工程文件的名称

### 返回

```
ErrorID, {}, RunScript(projectName);
```

## 示例

```
RunScript("demo")
```

运行名称为demo的工程。

## StopScript（立即指令）

### 原型

```
StopScript()
```

### 描述

停止正在运行的工程。

### 返回

```
ErrorID, {}, StopScript();
```

## 示例

```
StopScript()
```

停止正在运行的工程。

## PauseScript（立即指令）

### 原型

```
PauseScript()
```

### 描述

暂停正在运行的工程。

### 返回

```
ErrorID, {}, PauseScript();
```

## 示例

```
PauseScript()
```

暂停正在运行的工程。

## ContinueScript（立即指令）

### 原型

```
ContinueScript()
```

### 描述

继续已暂停的工程。

### 返回

```
ErrorID, {}, ContinueScript();
```

### 示例

```
ContinueScript()
```

继续已暂停的工程。

## Pause（立即指令）

### 原型

```
Pause()
```

### 描述

暂停非工程下发的运动指令（一般情况下即TCP下发的运动指令）,不清空运动队列。

### 返回

```
ErrorID, {}, Pause();
```

### 示例

```
Pause()
```

暂停非工程下发的运动指令。

## Continue（立即指令）

### 原型

```
Continue()
```

### 描述

与Pause指令对应，继续运行Pause暂停的运动指令。或者用于发生碰撞、报警后导致的机器人停止后，重新恢复接收运动指令并运行。

### 返回

```
ErrorID, {}, Continue();
```

### 示例

```
Continue()
```

继续运行Pause暂停的运动指令。

## StartDrag（立即指令）

### 原型

```
StartDrag()
```

### 描述

机械臂进入拖拽模式。机械臂处于报错状态下时，无法通过该指令进入拖拽模式。

### 返回

```
ErrorID, {}, StartDrag();
```

### 示例

```
StartDrag()
```

无报警时，机械臂进入拖拽模式。

## StopDrag（立即指令）

## 原型

```
StopDrag()
```

## 描述

机械臂退出拖拽模式。机械臂处于报错状态下时，无法通过该指令退出拖拽模式。

## 返回

```
ErrorID, {}, StopDrag();
```

## 示例

```
StopDrag()
```

无报警时，机械臂退出拖拽模式。

# EmergencyStop（立即指令）

## 原型

```
EmergencyStop()
```

## 描述

紧急停止机械臂。急停后机械臂会下电并报警，需要清除报警后才能重新上电和使能。

## 返回

```
ErrorID, {}, EmergencyStop();
```

## 示例

```
EmergencyStop()
```

紧急停止机器人并下电。

# Wait（队列指令）

## 原型

```
wait(time)
```

## 描述

指令队列延时一段时间。

控制器1.5.9及以上版本支持该指令。

## 参数

参数名	类型	说明
time	int	延时的时间，单位是ms。范围是(0,3600*1000)

## 返回

```
ErrorID,{},wait(time);
```

## 示例

```
wait(1000)
```

指令队列延时1000ms。

## 2.2 设置相关指令

### SpeedFactor（立即指令）

#### 原型

```
SpeedFactor(ratio)
```

#### 描述

设置全局速度比例。

- 机械臂点动时实际运动加速度/速度比例 = 控制软件点动设置中的值 x 全局速度比例。

例：控制软件设置的关节速度为12°/s，全局速率为50%，则实际点动速度为12°/s x 50% = 6°/s。

- 机械臂再现时实际运动加速度/速度比例 = 运动指令可选参数设置的的比例 x 控制软件再现设置中的值 x 全局速度比例。

例：控制软件设置的坐标系速度为2000mm/s，全局速率为50%，运动指令设置的速率为80%，则实际运动速度为2000mm/s x 50% x 80%= 800mm/s。

该命令设置的全局速率仅在本次TCP/IP控制模式中生效，未设置时沿用进入TCP/IP控制模式前控制软件设置的值。

#### 参数

参数名	类型	说明
ratio	int	全局运动速度比例，取值范围：1~100

#### 返回

```
ErrorID,{},SpeedFactor(ratio);
```

#### 示例

```
SpeedFactor(80)
```

设置全局运动速度比例为80%。

### User（队列指令）



## 原型

```
User(index)
```

## 描述

设置全局用户坐标系。用户下发运动指令时可选择用户坐标系，如未指定，则会使用全局用户坐标系。

该命令设置的全局用户坐标系仅在本次TCP/IP控制模式中生效，未设置时默认的全局用户坐标系为进入TCP/IP控制模式前控制软件设置的坐标系。

## 参数

参数名	类型	说明
index	int	已标定的用户坐标系索引。需要通过控制软件等方式标定后才可在此处通过索引选择。

## 返回

```
ErrorID,{},User(index);
```

若ErrorID返回-1，表示设置的用户坐标索引索引不存在；

## 示例

```
User(1)
```

设置用户坐标系1为全局用户坐标系。

# Tool（队列指令）

## 原型

```
Tool(index)
```

## 描述

设置全局工具坐标系。用户下发运动指令时可选择工具坐标系，如未指定，则会使用全局工具坐标系。

该命令设置的全局工具坐标系仅在本次TCP/IP控制模式中生效，未设置时默认的全局工具坐标系为进入TCP/IP控制模式前控制软件设置的坐标系。

## 参数

参数名	类型	说明
Tool	int	已标定的工具坐标系索引。需要通过控制软件等方式标定后才可在此处通过索引选择。

## 返回

```
ErrorID, {}, Tool(index);
```

若ErrorID返回-1，表示设置的工具坐标索引不存在；

## 示例

```
Tool(1)
```

设置工具坐标系1为全局工具坐标系。

# SetPayload（队列指令）

## 原型

```
SetPayload(weight, inertia)
```

## 描述

设置机械臂末端负载。

## 参数

参数名	类型	说明
weight	float	设置负载重量，取值范围不能超过各个型号机器人的负载范围。单位：kg
inertia	float	<b>可选参数。</b> 负载惯量，单位：kgm <sup>2</sup>

## 返回

```
ErrorID, {}, SetPayload(weight, inertia);
```

## 示例

```
SetPayload(0.3)
```

设置末端负载重量为0.3kg。

## AccJ（队列指令）

### 原型

```
AccJ(R)
```

### 描述

设置关节运动方式的加速度比例。

该命令设置的加速度比例仅在本次TCP/IP控制模式中生效，未设置时默认值为100。

### 参数

参数名	类型	说明
R	int	加速度比例。取值范围：[1,100]

### 返回

```
ErrorID,{},AccJ(R);
```

### 示例

```
AccJ(50)
```

设置关节运动方式的加速度比例为50%。

## AccL（队列指令）

### 原型

```
AccL(R)
```

### 描述

设置直线和弧线运动方式的加速度比例。

该命令设置的加速度比例仅在本次TCP/IP控制模式中生效，未设置时默认值为100。

### 参数

参数名	类型	说明
R	int	加速度比例。取值范围：[1,100]

## 返回

```
ErrorID, {}, AccL(R);
```

## 示例

```
AccL(50)
```

设置直线和弧线运动方式的加速度比例为50%。

# SpeedJ（队列指令）

## 原型

```
SpeedJ(R)
```

## 描述

设置关节运动方式的速度比例。

该命令设置的速度比例仅在本次TCP/IP控制模式中生效，未设置时默认值为100。

## 参数

参数名	类型	说明
R	int	速度比例。取值范围：[1,100]

## 返回

```
ErrorID, {}, SpeedJ(R);
```

## 示例

```
SpeedJ(50)
```

设置关节运动方式的速度比例为50%。

# SpeedL（队列指令）

## 原型

```
SpeedL(R)
```

## 描述

设置直线和弧线运动方式的速度比例。

该命令设置的速度比例仅在本次TCP/IP控制模式中生效，未设置时默认值为100。

## 参数

参数名	类型	说明
R	int	速度比例。取值范围：[1,100]

## 返回

```
ErrorID,{},SpeedL(R);
```

## 示例

```
SpeedL(50)
```

设置直线和弧线运动方式的速度比例为50%。

# Arch（队列指令）

## 原型

```
Arch(Index)
```

## 描述

设置Jump运动的全局门型参数索引。用户调用Jump运动指令时可以指定使用的门型参数，若未指定，则使用全局门型参数索引。

该命令设置的门型参数索引仅在本次TCP/IP控制模式中生效，未设置时默认值为0。

## 参数

参数名	类型	说明
Index	int	门型参数索引。需要通过控制软件等方式设置后才可在此处通过索引选择。

## 返回

```
ErrorID,{},Arch(Index);
```

## 示例

```
Arch(1)
```

设置全局门型参数索引为1。

## CP（队列指令）

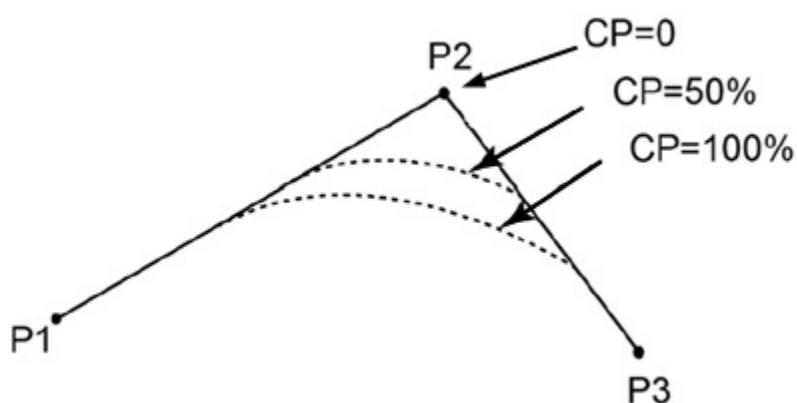
### 原型

```
CP(R)
```

### 描述

设置平滑过渡比例，即机械臂连续运动经过多个点时，经过中间点是以直角方式过渡还是以曲线方式过渡。该指令对Jump运动无效。

该命令设置的平滑过渡比例仅在本次TCP/IP控制模式中生效，未设置时默认值为0，表示不平滑过渡。



### 参数

参数名	类型	说明
R	unsigned int	平滑过渡比例。取值范围：[0, 100]

### 返回

```
ErrorID, {}, CP(R);
```

## 示例

```
CP(50)
```

设置平滑过渡比例为50。

## SetArmOrientation（队列指令）

### 原型

```
SetArmOrientation(LorR)
```

### 描述

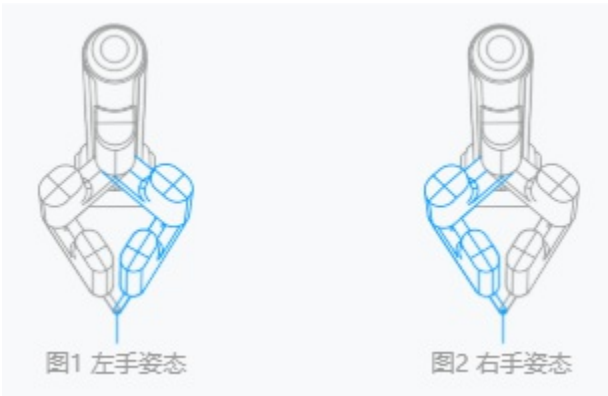
设置运动目标点的手系。运动目标点为笛卡尔坐标点时，可通过手系确定机械臂唯一姿态。设置手系后，后续目标点为笛卡尔坐标点的运动命令会根据手系规划运动轨迹。

该命令设置的手系仅在本次TCP/IP控制模式中生效，未设置时表示不指定手系。

此条指令为M1 Pro特有。

### 参数

参数名	类型	说明
LorR	int	0表示左手系，1表示右手系



### 返回

```
ErrorID, {}, SetArmOrientation(LorR);
```

### 示例

```
SetArmOrientation(1)
```

设置M1 Pro为左手系。

## SetCollisionLevel（队列指令）

## 原型

```
SetCollisionLevel(level)
```

## 描述

设置碰撞检测等级。

该命令设置的碰撞检测等级仅在本次TCP/IP控制模式中生效，未设置时沿用进入TCP/IP控制模式前控制软件设置的值。

## 参数

参数名	类型	说明
level	int	碰撞检测等级，0表示关闭碰撞检测，1~5数字越大灵敏度越高

## 返回

```
ErrorID,{},SetCollisionLevel(level);
```

## 示例

```
SetCollisionLevel(1)
```

设置碰撞检测等级为1。

# SetUser

## 原型：

```
SetUser(index,table)
```

## 描述：

修改指定的用户坐标系。

## 必选参数：

参数名	类型	说明
index	int	用户坐标系索引，取值范围：[0,9]，坐标系0初始值为基坐标系。
table	string	修改后的用户坐标系，格式为{x, y, z, r}，大括号可不带，建议使用CalcUser指令获取。



## 返回

```
ErrorID, {}, SetUser(index, table);
```

## 示例:

```
SetUser(1, {10, 10, 10, 0})  
// SetUser(1, 10, 10, 10, 0)
```

修改用户坐标系1为X=10, Y=10, Z=10, R=0。

## CalcUser

### 原型:

```
CalcUser(index, matrix_direction, table)
```

### 描述:

计算用户坐标系。

### 必选参数:

参数名	类型	说明
index	int	用户坐标系索引，取值范围：[0,9]，坐标系0初始值为基坐标系。
matrix_direction	int	计算的方向。 1表示左乘，即index指定的坐标系沿基坐标系偏转table指定的值； 0表示右乘，即index指定的坐标系沿自己偏转table指定的值。
table	string	用户坐标系偏移值，格式为{x, y, z, r}，大括号可不带。

### 返回:

```
ErrorID, {x, y, z, r}, CalcUser(index, matrix_direction, table);
```

其中{x, y, z, r}为计算得出的用户坐标系。

### 示例1:

```
newUser = CalcUser(1, 1, {10, 10, 10, 10})  
// newUser = CalcUser(1, 1, 10, 10, 10, 10)
```

计算用户坐标系1左乘{10,10,10,10}后的值。计算过程可等价：一个初始位姿与用户坐标系1相同的坐标系，沿基坐标系平移{x=10, y=10, z=10}并旋转r=10后，得到的新坐标系为newUser。

### 示例2：

```
newUser = CalcUser(1,0,{10,10,10,10})  
// newUser = CalcUser(1,0,10,10,10,10)
```

计算用户坐标系1右乘{10,10,10,10}后的值。计算过程可等价：一个初始位姿与用户坐标系1相同的坐标系，沿用户坐标系1平移{x=10, y=10, z=10}并旋转r=10后，得到的新坐标系为newUser。

## SetTool

### 原型：

```
SetTool(index,table)
```

### 描述：

修改指定的工具坐标系。

### 必选参数：

参数名	类型	说明
index	int	工具坐标系索引，取值范围：[0,9]，坐标系0初始值为法兰坐标系。
table	string	修改后的工具坐标系，格式为{x, y, z, r}，大括号可不带，建议使用CalcTool指令获取。

### 返回

```
ErrorID,{},SetTool(index,table);
```

### 示例：

```
SetTool(1,{10,10,10,0})  
// SetTool(1,10,10,10,0)
```

修改工具坐标系1为X=10, Y=10, Z=10, R=0。

## CalcTool

### 原型：

```
CalcTool(index,matrix_direction,table)
```

### 描述:

计算工具坐标系。

### 必选参数:

参数名	类型	说明
index	int	工具坐标系索引，取值范围：[0,9]，坐标系0初始值为法兰坐标系。
matrix_direction	int	计算的方向。 1表示左乘，即index指定的坐标系沿法兰坐标系偏转table指定的值； 0表示右乘，即index指定的坐标系沿自己偏转table指定的值。
table	string	工具坐标系偏移值，格式为{x, y, z, r}，大括号可不带。

### 返回:

```
ErrorID,{x,y,z,r},CalcTool(index,matrix_direction,table);
```

其中{x, y, z, r}为计算得出的工具坐标系。

### 示例1:

```
CalcTool(1,1,{10,10,10,10})  
// CalcTool(1,1,10,10,10,10)
```

计算工具坐标系1左乘{10,10,10,10}后的值。计算过程可等价于：一个初始位姿与工具坐标系1相同的坐标系，沿法兰坐标系平移{x=10, y=10, z=10}并旋转r=10后，得到的新坐标系为newTool。

### 示例2:

```
CalcTool(1,0,{10,10,10,10})  
// CalcTool(1,0,10,10,10,10)
```

计算工具坐标系1右乘{10,10,10,10}后的值。计算过程可等价于：一个初始位姿与工具坐标系1相同的坐标系，沿工具坐标系1平移{x=10, y=10, z=10}并旋转r=10后，得到的新坐标系为newTool。

## 2.3 计算和获取相关指令

### RobotMode（立即指令）

#### 原型

```
RobotMode()
```

#### 描述

获取机器人当前状态。

#### 返回

```
ErrorID,{Value},RobotMode();
```

Value取值范围如下：

取值	定义	说明
1	ROBOT_MODE_INIT	初始化
2	ROBOT_MODE_BRAKE_OPEN	有任意关节的抱闸松开
3	ROBOT_MODE_POWER_STATUS	本体未上电
4	ROBOT_MODE_DISABLED	未使能（无抱闸松开）
5	ROBOT_MODE_ENABLE	使能且空闲（未运行工程且无报警）
6	ROBOT_MODE_BACKDRIVE	拖拽模式
7	ROBOT_MODE_RUNNING	运行状态，包括轨迹复现/拟合中，机器人执行运动命令中，工程运行中。
8	ROBOT_MODE_RECORDING	轨迹录制模式
9	ROBOT_MODE_ERROR	有未清除的报警。此状态优先级最高，无论机械臂处于什么状态，有报警时都返回9
10	ROBOT_MODE_PAUSE	暂停状态
11	ROBOT_MODE_JOG	点动中

#### 示例

```
RobotMode()
```

获取机器人当前状态。

## GetAngle（立即指令）

### 原型

```
GetAngle()
```

### 描述

获取机械臂当前位姿的关节坐标。

### 返回

```
ErrorID,{J1,J2,J3,J4},GetAngle();
```

{J1,J2,J3,J4,J5,J6}表示机械臂当前位姿的关节坐标。

### 示例

```
GetAngle()
```

获取机械臂当前位姿的关节坐标。

## GetPose（立即指令）

### 原型

```
GetPose(User=0,Tool=0)
```

### 描述

获取机械臂当前位姿的笛卡尔坐标。

### 参数

参数名	类型	说明
User	int	已标定的用户坐标系索引。
Tool	int	已标定的工具坐标系索引。

均为**可选参数**。不传时默认为全局用户和工具坐标系。

### 返回

```
ErrorID,{X,Y,Z,R},GetPose());
```

{X,Y,Z,R}表示机械臂当前位姿的笛卡尔坐标。

### 示例1

```
GetPose()
```

获取机械臂当前位姿在全局用户和工具坐标系下的笛卡尔坐标。

### 示例2

```
GetPose(User=1,Tool=0)
```

获取机械臂当前位姿在用户坐标系1和工具坐标系0下的笛卡尔坐标。

## GetErrorID（立即指令）

### 原型

```
GetErrorID()
```

### 描述

获取机器人当前报错的错误码。

### 返回

```
ErrorID,{{{id,...,id}, [id], [id], [id], [id], [id], [id]}},GetErrorID());
```

- [id,...,id]为控制器以及算法报警信息，无报警时返回[]，有多个报警时以英文逗号“,”相隔。其中碰撞检测值为-2，其余报警含义请参考控制器错误描述文件alarm\_controller.json
- 后面四个[id]分别表示机械臂四个伺服的报警信息，无报警时返回[]。报警含义请参考伺服错误描述alarm\_servo.json

### 示例

```
GetErrorID()
```

获取机器人当前报错的错误码。

## PositiveSolution（立即指令）

## 原型

```
PositiveSolution(J1,J2,J3,J4,User,Tool)
```

## 描述

进行正解运算：给定机械臂各关节角度，计算机械臂末端在给定的笛卡尔坐标系中的坐标值。

## 参数

参数名	类型	说明
J1	double	J1轴位置，单位：度
J2	double	J2轴位置，单位：度
J3	double	J3轴位置，单位：度
J4	double	J4轴位置，单位：度
User	int	已标定的用户坐标系索引
Tool	int	已标定的工具坐标系索引

## 返回

```
ErrorID,{x,y,z,r},PositiveSolution(J1,J2,J3,J4,User,Tool);
```

{x,y,z,r}为点位的笛卡尔坐标值。

## 示例

```
PositiveSolution(0,0,-90,0,1,1)
```

关节坐标为{0,0,-90,0}，计算机械臂末端在用户坐标系1和关节坐标系1下的笛卡尔坐标。

# InverseSolution（立即指令）

## 原型

```
InverseSolution(X,Y,Z,R,User,Tool,isJointNear,JointNear)
```

## 描述

进行逆解运算：给定机械臂末端在给定的笛卡尔坐标系中的坐标值，计算机械臂各关节角度。

由于笛卡尔坐标仅定义了TCP的空间坐标与旋转角，所以机械臂可以通过多种不同的姿态到达同一个位姿，意味着一个位姿变量可以对应多个关节变量。为得出唯一的解，系统需要一个指定的关节坐标，选择最接近该关节坐标的解作为逆解结果。关于该关节坐标的设置，详见isJointNear和JointNear参数。

## 参数

参数名	类型	说明
X	double	X轴位置，单位：mm
Y	double	Y轴位置，单位：mm
Z	double	Z轴位置，单位：mm
R	double	R轴位置，单位：度
User	int	已标定的用户坐标系索引
Tool	int	已标定的工具坐标系索引
isJointNear	int	<b>可选参数。</b> 用于设置JointNear参数是否有效。 为0或不携带时表示JointNear无效，系统根据机械臂当前关节角度就近选解。 为1时表示根据JointNear就近选解。
JointNear	string	<b>可选参数。</b> 用于就近选解的关节坐标。

## 返回

```
ErrorID,{J1,J2,J3,J4},InverseSolution(X,Y,Z,R>User,Tool,isJointNear,JointNear);
```

{J1,J2,J3,J4}为点位的关节坐标值。

## 示例

```
InverseSolution(473.000000,-141.000000,469.000000,-180.000000,0,0)
```

机械臂末端在用户坐标系0和关节坐标系0下的笛卡尔坐标为{473,-141,469,-180}，计算关节坐标，选择机械臂当前关节角度的最近解。

```
InverseSolution(473.000000,-141.000000,469.000000,-180.000000,0,0,1,{0,0,-90,0})
```

机械臂末端在用户坐标系0和关节坐标系0下的笛卡尔坐标为{473,-141,469,-180}，计算关节坐标，选择{0,0,-90,0}的最近解。



## 2.4 IO相关指令

### DO（队列指令）

#### 原型

```
DO(index,status)
```

#### 描述

设置数字输出端口状态（队列指令）。

#### 参数

参数名	类型	说明
index	int	DO端子的编号
status	int	DO端子的状态, 1: 有信号; 0: 无信号

#### 返回

```
ErrorID,{},DO(index,status);
```

#### 示例

```
DO(1,1)
```

设置DO1为有信号。

### DOExecute（立即指令）

#### 原型

```
DOExecute(index,status)
```

#### 描述

设置数字输出端口状态（立即指令）。

#### 参数

参数名	类型	说明

index	int	DO端子的编号
status	int	DO端子的状态, 1: 有信号; 0: 无信号

## 返回

```
ErrorID, {}, DOExecute(index, status);
```

## 示例

```
DOExecute(1, 1)
```

无视指令队列, 立即设置DO1为有信号。

# DOGroup (立即指令)

## 原型

```
DOGroup(index1, value1, index2, value2, ..., indexN, valueN)
```

## 描述

设置多个数字输出端口状态 (立即指令) 。

## 参数

参数名	类型	说明
index1	int	第一个DO端子的编号
value1	int	第一个DO端子的状态, 1: 有信号; 0: 无信号
...	...	...
indexN	int	第N个DO端子的编号
valueN	int	第N个DO端子的状态, 1: 有信号; 0: 无信号

## 返回

```
ErrorID, {}, DOGroup(index1, value1, index2, value2, ..., indexn, valuen);
```

## 示例

```
DOGroup(4, 1, 6, 0, 2, 1, 7, 0)
```

设置DO4为有信号, DO6为无信号, DO2为有信号, DO7为无信号。

## ToolDO（队列指令）

### 原型

```
ToolDO(index,status)
```

### 描述

设置末端数字输出端口状态（队列指令）。

### 参数

参数名	类型	说明
index	int	末端DO端子的编号
status	int	末端DO端子的状态，1：有信号；0：无信号

### 返回

```
ErrorID,{},ToolDO(index,status);
```

### 示例

```
ToolDO(1,1)
```

设置末端DO1为有信号。

## ToolDOExecute（立即指令）

### 原型

```
ToolDOExecute(index,status)
```

### 描述

设置末端数字输出端口状态（立即指令）。

### 参数

参数名	类型	说明
index	int	末端DO端子的编号
status	int	末端DO端子的状态，1：有信号；0：无信号

### 返回

```
ErrorID,{}), ToolDOExecute(index,status);
```

## 示例

```
ToolDOExecute(1,1)
```

无视指令队列，立即设置末端DO1为有信号。

## DI（立即指令）

### 原型

```
DI(index)
```

### 描述

获取DI端口的状态。

### 参数

参数名	类型	说明
index	int	DI端子的编号

### 返回

```
ErrorID,{value},DI(index);
```

value表示DI端子的状态，0为无信号，1为有信号

## 示例

```
DI(1)
```

获取DI1的状态。

## ToolDI（立即指令）

### 原型

```
ToolDI(index)
```

### 描述

获取末端DI端口的状态。

### 参数

参数名	类型	说明
index	int	末端DI端子的编号

### 返回

```
ErrorID,{value},ToolDI(index);
```

value表示末端DI端子的状态，0为无信号，1为有信号

### 示例

```
ToolDI(1)
```

获取末端DI1的状态。

## 2.5 Modbus相关指令

### ModbusCreate（立即指令）

#### 原型

```
ModbusCreate(ip,port,slave_id,isRTU)
```

#### 描述

创建Modbus主站，并和从站建立连接。最多支持同时连接5个设备。

#### 参数

参数名	类型	说明
ip	string	从站IP地址。
port	int	从站端口
slave_id	int	从站ID
isRTU	int	<b>可选参数。</b> 如果不携带或为0，建立modbusTCP通信； 如果为1，建立modbusRTU通信

#### 返回

```
ErrorID,{index},ModbusCreate(ip,port,slave_id,isRTU);
```

- ErrorID为0表示创建成功，-1表示创建失败，其余错误码请参考通用错误码
- index为返回的主站索引，取值范围0~4，后续调用其他Modbus指令时使用

#### 示例

```
ModbusCreate(127.0.0.1,60000,1,1)
```

建立RTU通信主站，连接本机的Modbus从站，端口为60000，从站ID为1。

### ModbusClose（立即指令）

#### 原型

```
ModbusClose(index)
```

## 描述

和Modbus从站断开连接，释放主站。

## 参数

参数名	类型	说明
index	int	创建主站时返回的主站索引

## 返回

```
ErrorID,{},ModbusClose(index);
```

## 示例

```
ModbusClose(0)
```

释放索引为0的Modbus主站。

# GetInBits（立即指令）

## 原型

```
GetInBits(index,addr,count)
```

## 描述

读取Modbus从站触点寄存器（离散输入）地址的值。

## 参数

参数名	类型	说明
index	int	创建主站时返回的主站索引
addr	int	触点寄存器起始地址
count	int	连续读取触点寄存器的值的数量。取值范围1~16

## 返回

```
ErrorID,{value1,value2,...,valuen},GetInBits(index,addr,count);
```

{value1,value2,...,valuen}为读取的值，数量与count相同。

## 示例

```
GetInBits(0,3000,5)
```

从地址为3000的触点寄存器开始读取5个值。

## GetInRegs（立即指令）

### 原型

```
GetInRegs(index,addr,count,valType)
```

### 描述

按照指定的数据类型，读取Modbus从站输入寄存器地址的值。

### 参数

参数名	类型	说明
index	int	创建主站时返回的主站索引
addr	int	输入寄存器起始地址
count	int	连续读取输入寄存器的值的数量。取值范围：[1, 4]
valType	string	<b>可选参数。</b> 读取的数据类型： 为空或U16：16位无符号整数（2个字节，占用1个寄存器）； U32：32位无符号整数（4个字节，占用2个寄存器） F32：32位单精度浮点数（4个字节，占用2个寄存器） F64：64位双精度浮点数（8个字节，占用4个寄存器）

### 返回

```
ErrorID,{value1,value2,...,valuen},GetInBits(index,addr,count);
```

{value1,value2,...,valuen}为读取的值，数量与count相同。

### 示例

```
GetInRegs(0,4000,3)
```

从地址为4000的输入寄存器开始读取3个值，值类型为U16。

## GetCoils（立即指令）

### 原型

```
GetCoils(index,addr,count)
```



## 描述

读取Modbus从站线圈寄存器地址的值。

## 参数

参数名	类型	说明
index	int	创建主站时返回的主站索引
addr	int	线圈寄存器起始地址。
count	int	连续读取线圈寄存器的值的数量。取值范围：[1, 16]

## 返回

```
ErrorID,{value1,value2,...,valuen},GetCoils(index,addr,count);
```

{value1,value2,...,valuen}为读取的值，数量与count相同。

## 示例

```
GetCoils(0,1000,3)
```

从地址为1000的线圈寄存器开始读取3个值。

# SetCoils（立即指令）

## 原型

```
SetCoils(index,addr,count,valTab)
```

## 描述

将指定的值写入线圈寄存器指定的地址。

## 参数

参数名	类型	说明
index	int	创建主站时返回的主站索引
addr	int	线圈寄存器起始地址。
count	int	连续写入线圈寄存器的值的数量。取值范围：[1, 16]
valTab	string	要写入的值，数量与count相同。

## 返回

```
ErrorID, {}, SetCoils(index, addr, count, valTab);
```

## 示例

```
SetCoils(0, 1000, 3, {1, 0, 1})
```

从地址为1000的线圈寄存器开始连续写入3个值，分别为1，0，1。

# GetHoldRegs（立即指令）

## 原型

```
GetHoldRegs(index, addr, count, valType)
```

## 描述

按照指定的数据类型，读取Modbus从站保持寄存器地址的值。

## 参数

参数名	类型	说明
index	int	创建主站时返回的主站索引
addr	int	保持寄存器起始地址。
count	int	连续读取保持寄存器的值的数量。取值范围：[1, 4]
valType	string	<b>可选参数。</b> 读取的数据类型： 为空或U16：16位无符号整数（2个字节，占用1个寄存器）； U32：32位无符号整数（4个字节，占用2个寄存器） F32：32位单精度浮点数（4个字节，占用2个寄存器） F64：64位双精度浮点数（8个字节，占用4个寄存器）

## 返回

```
ErrorID, {value1, value2, ..., valuen}, GetHoldRegs(index, addr, count, valType);
```

{value1, value2, ..., valuen}为读取的值，数量与count相同。

## 示例

```
GetHoldRegs(0, 3095, 1)
```

从地址为3095的保持寄存器开始读取1个值，值类型为U16。

## SetHoldRegs（立即指令）

### 原型

```
SetHoldRegs(index,addr, count,valTab,valType)
```

### 描述

按照指定的数据类型，将指定的值写入Modbus从站保持寄存器指定的地址。

### 参数

参数名	类型	说明
index	int	创建主站时返回的主站索引
addr	int	保持寄存器起始地址。
count	int	连续写入保持寄存器的值的数量。取值范围：[1, 4]
valTab	string	要写入的值，数量与count相同。
valType	string	<b>可选参数。</b> 读取的数据类型： 为空或U16：16位无符号整数（2个字节，占用1个寄存器）； U32：32位无符号整数（4个字节，占用2个寄存器） F32：32位单精度浮点数（4个字节，占用2个寄存器） F64：64位双精度浮点数（8个字节，占用4个寄存器）

### 返回

```
ErrorID,{},SetHoldRegs(index,addr, count,valTab,valType);
```

### 示例

```
SetHoldRegs(0,3095,2,{6000,300}, U16)
```

从地址为3095的保持寄存器开始写入两个U16类型的值，分别为6000和300。

## 3 运动指令

运动相关指令需要通过**30003**端口下发。

## 3.1 通用说明

### 坐标系参数

笛卡尔坐标系相关的运动指令，可选参数的User和Tool用于指定目标点的用户和工具坐标系：

- 如果携带了User和Tool参数，则User和Tool分别用于指定已标定的用户坐标系和工具坐标系的索引。
- 如果不携带User和Tool参数，则使用全局用户和工具坐标系，详见[设置相关指令](#)中的User和Tool指令说明。

### 速度参数

可选参数中的SpeedJ/SpeedL/AccJ/AccL用于指定机械臂执行该运动指令时的加速度和速度比例。

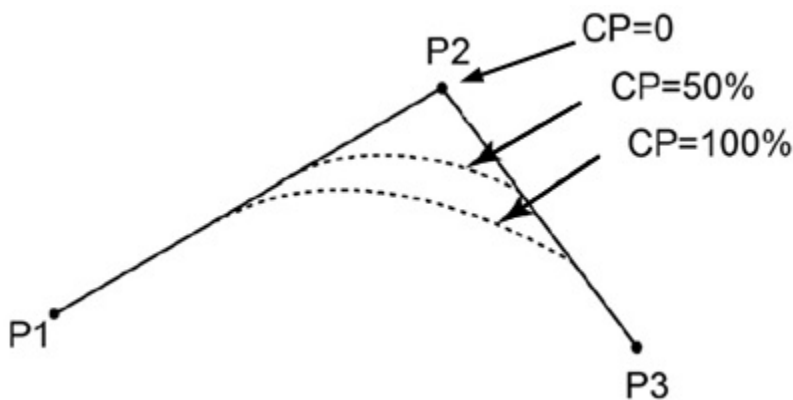
机械臂实际运动加速度/速度比例 = 运动指令可选参数设置的比例 × 控制软件再现设置中的值 × 全局速率

例：控制软件设置的坐标系速度为2000mm/s，全局速率为50%，运动指令设置的速率为80%，则实际运动速度为2000mm/s × 50% × 80% = 800mm/s。

未通过可选参数指定运动加速度/速度比例时，默认使用全局设置，详见[设置相关指令](#)中的SpeedJ/SpeedL/AccJ/AccL指令。

### 平滑过渡参数

可选参数中的CP用于指定当前运动指令到下一条运动指令之间的平滑过渡比例，即机械臂连续运动经过多个点时，经过中间点是以直角方式过渡还是以曲线方式过渡。



### 使用限制

TCP运动指令不支持在可选参数中携带“SYNC”参数实现同步功能，请使用Syncn()或者SyncAll()指令。

**上述参数的含义在后文中不再赘述。**

# 3.2 指令列表

**i 说明:**  
运动相关指令均为队列指令。

## MovJ

### 原型

```
MovJ(X,Y,Z,R,User=index,Tool=index,SpeedJ=R,AccJ=R,CP=R)
```

### 描述

从当前位置以关节运动方式运动至笛卡尔坐标目标点。关节运动的轨迹非直线，所有关节会同时完成运动。

### 参数

参数名	类型	说明
X	double	目标点X轴位置，单位：mm
Y	double	目标点Y轴位置，单位：mm
Z	double	目标点Z轴位置，单位：mm
R	double	目标点R轴位置，单位：度

### 返回

```
ErrorID,{},MovJ(X,Y,Z,R);
```

### 示例

```
MovJ(-100,100,200,150,AccJ=50)
```

机械臂从当前位置以50%的加速度通过关节运动方式运动至笛卡尔坐标点{-100,100,200,150}。

## MovL

### 原型

```
MovL(X,Y,Z,R,User=index,Tool=index,SpeedL=R,AccL=R,CP=R)
```

## 描述

从当前位置以直线运动方式运动至笛卡尔坐标目标点。

## 参数

参数名	类型	说明
X	double	目标点X轴位置，单位：mm
Y	double	目标点Y轴位置，单位：mm
Z	double	目标点Z轴位置，单位：mm
R	double	目标点R轴位置，单位：度

## 返回

```
ErrorID,{},MovL(X,Y,Z,R);
```

## 示例

```
MovL(-100,100,200,150,SpeedL=60)
```

机械臂从当前位置以60%的速度通过直线运动方式运动至笛卡尔坐标点{-100,100,200,150}。

# JointMovJ

## 原型

```
JointMovJ(J1,J2,J3,J4,SpeedJ=R,AccJ=R,CP=R)
```

## 描述

从当前位置以关节运动方式运动至关节坐标目标点。

## 参数

参数名	类型	说明
J1	double	目标点 J1轴位置，单位：度
J2	double	目标点 J2轴位置，单位：度
J3	double	目标点 J3轴位置，单位：度
J4	double	目标点 J4轴位置，单位：度



## 返回

```
ErrorID, {}, JointMovJ(J1, J2, J3, J4);
```

## 示例

```
JointMovJ(0, 0, -90, 0, SpeedJ=60, AccJ=50)
```

机械臂从当前位置以60%的速度和50%的加速度通过关节运动方式运动至关节坐标点{0,0,-90,0}。

# MovLIO

## 原型

```
MovLIO(X, Y, Z, R, {Mode, Distance, Index, Status}, ..., {Mode, Distance, Index, Status}, User=index, Tool=index, SpeedL=R, AccL=R, CP=R)
```

## 描述

从当前位置以直线运动方式运动至笛卡尔坐标目标点，运动时并行设置数字输出端口状态。

## 参数

参数名	类型	说明
X	double	目标点X轴位置，单位：mm
Y	double	目标点Y轴位置，单位：mm
Z	double	目标点Z轴位置，单位：mm
R	double	目标点R轴位置，单位：度

{Mode,Distance,Index,Status}为并行数字输出参数，用于设置当机械臂运动到指定距离或百分比时，触发指定DO。可设置多组，参数具体含义如下：

参数名	类型	说明
Mode	int	触发模式。0表示距离百分比，1表示距离数值
Distance	int	指定距离。 Distance为正数时，表示离起点的距离； Distance为负数时，表示离目标点的距离； Mode为0时，Distance表示和总距离的百分比；取值范围：(0,100]； Mode为1时，Distance表示距离的值。单位：mm
Index	int	DO端子的编号

Status	int	要设置的DO状态，0表示无信号，1表示有信号
--------	-----	------------------------

## 返回

```
ErrorID, {}, MovLIO(X,Y,Z,R,{Mode,Distance,Index,Status},...,{Mode,Distance,Index,Status},SpeedL=R,AccL=R);
```

## 示例

```
MovLIO(-100,100,200,150,{0,50,1,0})
```

机械臂从当前位置通过直线运动方式运动至笛卡尔坐标点{-100,100,200,150}，并在运动到50%的距离时将DO1设为无信号。

# MovJIO

## 原型

```
MovJIO(X,Y,Z,R,{Mode,Distance,Index,Status},...,{Mode,Distance,Index,Status},User=index,Tool=index,SpeedJ=R,AccJ=R,CP=R)
```

## 描述

从当前位置以关节运动方式运动至笛卡尔坐标目标点，运动时并行设置数字输出端口状态。

## 参数

参数名	类型	说明
X	double	目标点X轴位置，单位：mm
Y	double	目标点Y轴位置，单位：mm
Z	double	目标点Z轴位置，单位：mm
R	double	目标点R轴位置，单位：度

{Mode,Distance,Index,Status}为并行数字输出参数，用于设置当机械臂运动到指定距离或百分比时，触发指定DO。可设置多组，参数具体含义如下：

参数名	类型	说明
Mode	int	触发模式。0表示距离百分比，1表示距离数值
Distance	int	指定距离。 Distance为正数时，表示离起点的距离； Distance为负数时，表示离目标点的距离； Mode为0时，Distance表示和总距离的百分比；取值范围：(0,100]；

		Mode为1时，Distance表示距离的值。单位：mm
Index	int	DO端子的编号
Status	int	要设置的DO状态，0表示无信号，1表示有信号

## 返回

```
ErrorID, {}, MovJIO(X,Y,Z,R,{Mode,Distance,Index,Status},...,{Mode,Distance,Index,Status});
```

## 示例

```
MovJIO(-100,100,200,150,{0,50,1,0})
```

机械臂从当前位置通过关节运动方式运动至笛卡尔坐标点{-100,100,200,150}，并在运动到50%的距离时将DO1设为无信号。

# Arc

## 原型

```
Arc(X1,Y1,Z1,R1,X2,Y2,Z2,R2,User=index,Tool=index,SpeedL=R,AccL=R,CP=R)
```

## 描述

从当前位置以圆弧插补方式运动至目标点。

需要通过当前位置，圆弧中间点，运动目标点三个点确定一个圆弧，因此当前位置不能在P1和P2确定的直线上。

## 参数

参数名	类型	说明
X1	double	圆弧中间点X轴位置，单位：mm
Y1	double	圆弧中间点Y轴位置，单位：mm
Z1	double	圆弧中间点Z轴位置，单位：mm
R1	double	圆弧中间点R轴位置，单位：度
X2	double	目标点X轴位置，单位：mm
Y2	double	目标点Y轴位置，单位：mm
Z2	double	目标点Z轴位置，单位：mm
R2	double	目标点R轴位置，单位：度

## 返回

```
ErrorID, {}, Arc(X1, Y1, Z1, R1, X2, Y2, Z2, R2);
```

## 示例

```
Arc(-350, -200, 200, 150, -300, -250, 200, 150)
```

机械臂从当前位置通过圆弧运动方式经由笛卡尔坐标点{-350,-200,200,150}运动至笛卡尔坐标点{-300,-250,200,150}。

## Circle

```
Circle3({X1,Y1,Z1,R1},{X2,Y2,Z2,R2},count,User=index,Tool=index,SpeedL=R,AccL=R)
```

## 描述

从当前位置进行整圆插补运动，运动指定圈数后重新回到当前位置。

需要通过当前位置，P1，P2三个点确定一个整圆，因此当前位置不能在P1和P2确定的直线上，且三个点确定的整圆不能超出机械臂的运动范围。

## 参数

参数名	类型	说明
X1	double	P1点X轴位置，单位：mm
Y1	double	P1点Y轴位置，单位：mm
Z1	double	P1点Z轴位置，单位：mm
R1	double	P1点R轴位置，单位：度
X2	double	P2点X轴位置，单位：mm
Y2	double	P2点Y轴位置，单位：mm
Z2	double	P2点Z轴位置，单位：mm
R2	double	P2点R轴位置，单位：度
count	int	进行整圆运动的圈数，取值范围1~999。

## 返回

```
ErrorID, {}, Circle3({X1,Y1,Z1,R1},{X2,Y2,Z2,R2},count);
```

## 示例

```
Circle3({-350,-200,200,150},{-300,-250,200,150},1)
```

机械臂沿当前点和两个指定点确定的圆运动一周回到当前点。

## MoveJog

### 原型

```
MoveJog(axisID,CoordType=typeValue,User=index,Tool=index)
```

### 描述

点动机械臂。命令下发后机械臂会沿指定轴持续点动，需要再下发MoveJog()停止机械臂运动。另外，机械臂点动时下发携带任意非指定string的MoveJog(string)也会使机械臂停止运动。

控制器1.5.6及以上版本支持该命令。

### 参数

参数名	类型	说明
axisID	string	点动运动轴 J1+ 表示关节1正方向运动， J1- 表示关节1负方向运动 J2+ 表示关节2正方向运动， J2- 表示关节2负方向运动 J3+ 表示关节3正方向运动， J3- 表示关节3负方向运动 J4+ 表示关节4正方向运动， J4- 表示关节4负方向运动 X+ 表示X轴正方向运动， X- 表示X轴负方向运动 Y+ 表示Y轴正方向运动， Y- 表示Y轴负方向运动 Z+ 表示Z轴正方向运动， Z- 表示Z轴负方向运动 R+ 表示R轴正方向运动， R- 表示R轴负方向运动
CoordType	int	<b>可选参数。</b> 仅当axisID指定笛卡尔坐标系的轴时生效，指定运动轴所属的坐标系。0表示用户坐标系， 1表示工具坐标系

### 返回

```
ErrorID, {}, MoveJog(axisID,CoordType=typeValue,User=index,Tool=index);
```

## 示例

```
MoveJog(j2-)  
// 停止点动  
MoveJog()
```

沿J2轴负方向点动，然后停止点动。

# Sync

## 原型

```
Sync()
```

## 描述

阻塞程序执行队列指令，待队列最后的指令执行完后才返回。

## 返回

```
ErrorID, {}, Sync();
```

## 示例

```
MovJ(x,y,z,r)  
Sync()  
RobotMode()
```

待机器人运动到{x,y,z,rx,ry,rz}后，再获取机器人当前状态。

# RelMovJUser

## 原型

```
RelMovJUser(OffsetX,OffsetY,OffsetZ,OffsetR, User,SpeedJ=R, AccJ=R,Tool=Index,CP=R)
```

## 描述

沿用户坐标系进行相对运动，末端运动方式为关节运动。

控制器1.5.6及以上版本支持该命令。

## 参数

参数名	类型	说明
offsetX	double	X轴方向偏移量，单位：mm
offsetY	double	Y轴方向偏移量，单位：mm
offsetZ	double	Z轴方向偏移量，单位：mm
offsetR	double	R轴偏移量，单位：度
User	int	选择已标定的用户坐标系索引

返回

```
ErrorID,{},RelMovJUser(OffsetX,OffsetY,OffsetZ,OffsetR,User,SpeedJ=R,AccJ=R,Tool=Index);
```

示例

```
RelMovJUser(10,10,10,0,0)
```

机械臂沿用户坐标系0进行相对关节运动，在X、Y、Z轴上各偏移10mm。

RelMovLUser

原型

```
RelMovLUser(OffsetX,OffsetY,OffsetZ,OffsetR,User,SpeedL=R,AccL=R,Tool=Index,CP=R)
```

描述

沿用户坐标系进行相对运动，末端运动方式为直线运动。

控制器1.5.6及以上版本支持该命令。

参数

参数名	类型	说明
offsetX	double	X轴方向偏移量，单位：mm
offsetY	double	Y轴方向偏移量，单位：mm
offsetZ	double	Z轴方向偏移量，单位：mm
offsetR	double	R轴偏移量，单位：度
User	int	选择已标定的用户坐标系索引

返回

```
ErrorID,{},RelMovJUser(OffsetX,OffsetY,OffsetZ,OffsetR,User);
```

示例

```
RelMovLUser(10,10,10,0,0)
```

机械臂沿用户坐标系0进行相对直线运动，在X、Y、Z轴上各偏移10mm。

# RelJointMovJ

## 原型

```
RelJointMovJ(Offset1,Offset2,Offset3,Offset4,SpeedJ=R,AccJ=R,CP=R)
```

## 描述

沿关节坐标系进行相对运动，末端运动方式为关节运动。

控制器1.5.6及以上版本支持该命令。

## 参数

参数名	类型	说明
offset1	double	J1轴偏移量，单位：度
offset2	double	J2轴偏移量，单位：度
offset3	double	J3轴偏移量，单位：度
offset4	double	J4轴偏移量，单位：度

## 返回

```
ErrorID,{},RelJointMovJ(Offset1,Offset2,Offset3,Offset4);
```

## 示例

```
RelJointMovJ(10,10,10,0)
```

机械臂J1，J2，J3轴分别偏移10度。

# MovJExt

## 原型

```
MovJExt(Angle|Distance,SpeedE=50,AccE=50,Sync=1)
```

## 描述

控制滑轨（扩展轴）运动到目标角度或位置。

## 参数

参数名	类型	说明



Angle or Distance	float	运动的目标角度或者距离。该参数的含义取决于扩展轴工艺中高级设置里设置的运动类型（关节/直线）。单位：类型为关节时为度，类型为直线时为毫米
SpeedE	int	<b>可选参数。</b> 运动速度比例，取值范围：1~100。默认值为100
AccE	int	<b>可选参数。</b> 运动加速度比例，取值范围：1~100。默认值为100
Sync	int	<b>可选参数。</b> 同步标识，取值范围：0或1。默认值为0。 SYNC = 0表示异步执行，调用后立即返回，不关注指令执行情况； SYNC = 1表示同步执行，调用后，待指令执行完才返回。

## 返回

```
ErrorID,{},MovJExt(Angle|Distance,SpeedE=50,AccE=50,Sync=1);
```

## 示例

```
MovJExt(300)
```

假设扩展轴运动类型为毫米，表示扩展轴运动至300mm处。

# SyncAll

## 原型

```
SyncAll()
```

## 描述

阻塞程序执行队列指令，待队列中所有指令执行完后才返回。

此指令主要用于有扩展轴的场景：扩展轴和机械臂是各自独立运动的，Sync指令在队列中最后一条指令（假设为机械臂运动指令）执行完成后就会返回，此时队列前面的扩展轴运动指令可能还未执行完毕（反之亦然）；若想要确保队列中所有指令都执行完成，可使用SyncAll命令。

## 返回

```
ErrorID,{},SyncAll();
```

## 示例

```
MovJ(x1,y1,z1,r1)
MovJExt(distance)
MovJ(x2,y2,z2,r2)
SyncAll()
RobotMode()
```

---

待机械臂和拓展轴都完成运动后，再获取机器人当前状态。

## 4 实时反馈信息

控制器通过**30004、30005以及30006端口**实时反馈机器人状态信息。通过实时反馈端口每次收到的数据包有1440个字节，这些字节以标准的格式排列，如下表所示。

实时反馈的数据以小端（低位优先）的方式存储，即一个值用多个字节存储时，数据的低位存储在靠前的字节中。

例如，某个数据值为1234，转换为二进制为0000 0100 1101 0010，通过两个字节传递，第一个字节为1101 0010（二进制值的低8位），第二个字节为0000 0100（二进制值的高8位）。

含义	数据类型	值的数目	字节大小	字节位置值	描述
MessageSize	unsigned short	1	2	0000 ~ 0001	消息字节总长度
N/A	unsigned short	3	6	0002 ~ 0007	保留位
DigitalInputs	uint64	1	8	0008 ~ 0015	当前数字输入端子状态，详见 <a href="#">DI/DO说明</a>
DigitalOutputs	uint64	1	8	0016 ~ 0023	当前数字输出端子状态，详见 <a href="#">DI/DO说明</a>
RobotMode	uint64	1	8	0024 ~ 0031	机器人模式，含义详见 <a href="#">RobotMode指令说明</a>
TimeStamp	uint64	1	8	0032 ~ 0039	Unix时间戳（单位ms）
N/A	uint64	1	8	0040 ~ 0047	保留位
TestValue	uint64	1	8	0048 ~ 0055	内存结构测试标准值 0x0123 4567 89AB CDEF
N/A	double	1	8	0056 ~	保留位

				0063	
SpeedScaling	double	1	8	0064 ~ 0071	速度比例
N/A	double	1	8	0072 ~ 0079	保留位
VMain	double	1	8	0080 ~ 0087	控制板电压
VRobot	double	1	8	0088 ~ 0095	机器人电压
IRobot	double	1	8	0096 ~ 0103	机器人电流
N/A	double	1	8	0104 ~ 0111	保留位
N/A	double	1	8	0112 ~ 0119	保留位
N/A	double	3	24	0120 ~ 0143	保留位
N/A	double	3	24	0144 ~ 0167	保留位
N/A	double	3	24	0168 ~ 0191	保留位
QTarget	double	6	48	0192 ~ 0239	目标关节位置
QDTarget	double	6	48	0240 ~ 0287	目标关节速度
QDDTarget	double	6	48	0288 ~ 0335	目标关节加速度
				0336	

ITarget	double	6	48	~ 0383	目标关节电流
MTarget	double	6	48	0384 ~ 0431	目标关节扭矩
QActual	double	6	48	0432 ~ 0479	实际关节位置
QDActual	double	6	48	0480 ~ 0527	实际关节速度
IActual	double	6	48	0528 ~ 0575	实际关节电流
ActualTCPForce	double	6	48	0576 ~ 0623	保留位
ToolVectorActual	double	6	48	0624 ~ 0671	TCP笛卡尔实际坐标值
TCPSpeedActual	double	6	48	0672 ~ 0719	TCP笛卡尔实际速度值
TCPForce	double	6	48	0720 ~ 0767	TCP力值（通过关节电 流计算）
ToolVectorTarget	double	6	48	0768 ~ 0815	TCP笛卡尔目标坐标值
TCPSpeedTarget	double	6	48	0816 ~ 0863	TCP笛卡尔目标速度值
MotorTemperatures	double	6	48	0864 ~ 0911	关节温度
JointModes	double	6	48	0912 ~ 0959	关节控制模式，8表示 位置模式，10表示力 矩模式
VActual	double	6	48	960 ~ 1007	关节电压

HandType	char	4	4	1008 ~ 1011	手系，含义详见 <a href="#">SetArmOrientation</a> 指令的参数说明
User	char	1	1	1012	用户坐标系
Tool	char	1	1	1013	工具坐标系
RunQueuedCmd	char	1	1	1014	算法队列运行标志
PauseCmdFlag	char	1	1	1015	算法队列暂停标志
VelocityRatio	char	1	1	1016	关节速度比例(0~100)
AccelerationRatio	char	1	1	1017	关节加速度比例 (0~100)
JerkRatio	char	1	1	1018	关节加加速度比例 (0~100)
XYZVelocityRatio	char	1	1	1019	笛卡尔位置速度比例 (0~100)
RVelocityRatio	char	1	1	1020	笛卡尔姿态速度比例 (0~100)
XYZAccelerationRatio	char	1	1	1021	笛卡尔位置加速度比例 (0~100)
RAccelerationRatio	char	1	1	1022	笛卡尔姿态加速度比例 (0~100)
XYZJerkRatio	char	1	1	1023	笛卡尔位置加加速度比例 (0~100)
RJerkRatio	char	1	1	1024	笛卡尔姿态加加速度比例 (0~100)
BrakeStatus	char	1	1	1025	机器人抱闸状态，详见 <a href="#">BrakeStatus</a> 说明
EnableStatus	char	1	1	1026	机器人使能状态
DragStatus	char	1	1	1027	机器人拖拽状态
RunningStatus	char	1	1	1028	机器人运行状态
ErrorStatus	char	1	1	1029	机器人报警状态
JogStatusCR	char	1	1	1030	机器人点动状态
RobotType	char	1	1	1031	机器人型号，详见 <a href="#">RobotType</a> 说明
DragButtonSignal	char	1	1	1032	保留位
EnableButtonSignal	char	1	1	1033	保留位

RecordButtonSignal	char	1	1	1034	保留位
ReappearButtonSignal	char	1	1	1035	保留位
JawButtonSignal	char	1	1	1036	保留位
SixForceOnline	char	1	1	1037	保留位
N/A	char	1	82	1038-1119	保留位
MActual[6]	double	6	48	1120~1167	四个关节的实际扭矩
Load	double	1	8	1168-1175	末端负载重量 (单位kg)
CenterX	double	1	8	1176-1183	末端负载X方向偏心距离 (单位mm)
CenterY	double	1	8	1184-1191	末端负载Y方向偏心距离 (单位mm)
CenterZ	double	1	8	1192-1199	末端负载Z方向偏心距离 (单位mm)
User[6]	double	6	48	1200-1247	用户坐标系坐标值
Tool[6]	double	6	48	1248-1295	工具坐标系坐标值
TraceIndex	double	1	8	1296-1303	轨迹复现运行索引
SixForceValue[6]	double	6	48	1304-1351	保留位
TargetQuaternion[4]	double	4	32	1352-1383	[qw,qx,qy,qz] 目标四元数
ActualQuaternion[4]	double	4	32	1384-1415	[qw,qx,qy,qz] 实际四元数
N/A	char	1	24	1416~1440	保留位
TOTAL			1440		1440byte package

## DI/DO说明

DI/DO各占8个字节，每个字节有8位（二进制），最大可表示DI/DO各64个端口的状态。每个字节从低到高每一位表示一个端子的状态，1表示对应端子为有信号状态，0表示对应端子无信号或者无对应端子。

例如第一个字节为0x01（00000001），第二个字节为0x00（00000000），其余字节都为全0，则表示DI1为1，其余端子为0。

### BrakeStatus说明

该字节按位表达各个关节的抱闸状态，对应位为1是表示该关节的抱闸已松开。位数与关节的对应关系如下表：

7	6	5	4	3	2	1	0
保留位	保留位	关节1	关节2	关节3	关节4	保留位	保留位

示例：

0x03（00000100）：关节4抱闸松开

### RobotType说明

取值	代表机型
1	MG400
2	M1 Pro



## 5 通用错误码

错误码	描述	备注
0	无错误	下发成功
-1	没有获取成功	命令接收失败/执行失败
...	...	...
-10000	命令错误	下发的命令不存在
-20000	参数数量错误	下发命令中的参数数量错误
-30001	第一个参数的参数类型错误	-30000表示参数类型错误 最后一位1表示下发第1个参数的参数类型错误
-30002	第二个参数的参数类型错误	-30000表示参数类型错误 最后一位2表示下发第2个参数的参数类型错误
...	...	...
-40001	第一个参数的参数范围错误	-40000表示参数范围错误 最后一位1表示下发第1个参数的参数范围错误
-40002	第二个参数的参数范围错误	-40000表示参数范围错误 最后一位2表示下发第2个参数的参数范围错误
...	...	...