# Dobot Magician API

# 接口说明

文档版本：V1.2.2

发布日期：2018-11-06

深圳市越疆科技有限公司

**免责申明**

在法律允许的最大范围内，本手册所描述的产品（含其硬件、软件、固件等）均“按照现状”提供，可能存在瑕疵、错误或故障，越疆不提供任何形式的明示或默示保证，包括但不限于适销性、质量满意度、适合特定目的、不侵犯第三方权利等保证；亦不对使用本手册或使用本公司产品导致的任何特殊、附带、偶然或间接的损害进行赔偿。

在使用本产品前详细阅读本使用手册及网上发布的相关技术文档并了解相关信息，确保在充分了解机器人及其相关知识的前提下使用机械臂。越疆建议您在专业人员的指导下使用本手册。该手册所包含的所有安全方面的信息都不得视为Dobot的保证，即便遵循本手册及相关说明，使用过程中造成的危害或损失依然有可能发生。

本产品的使用者有责任确保遵循相关国家的切实可行的法律法规，确保在越疆机械臂的使用中不存在任何重大危险。

## 越疆科技有限公司

地址：深圳市南山区同富裕工业城三栋三楼

网址：http://cn.dobot.cc/

# 前 言

## 目的

本文档旨在对 Dobot API 接口进行详细说明，并给出基于 Dobot API 接口开发应用程序的一般流程。

### 读者对象

本手册适用于：

- 客户工程师
- 安装调测工程师
- 技术支持工程师

### 修订记录

| 时间 | 修订记录 |
| --- | --- |
| 2018/11/06 | 修正了一些API的错误 |
| 2018/03/21 | 优化所有API |

### 符号约定

在本手册中可能出现下列标志，它们所代表的含义如下。

| 符号 | 说明 |
| --- | --- |
| ⚠危险 | 表示有高度潜在危险，如果不能避免，会导致人员死亡或严重伤害 |
| ⚠警告 | 表示有中度或低度潜在危害，如果不能避免，可能导致人员轻微伤害、机械臂毁坏等情况 |
| ⚠注意 | 表示有潜在风险，如果忽视这些文本，可能导致机械臂损坏、数据丢失或不可预知的结果 |
| 🕮说明 | 表示是正文的附加信息，是对正文的强调和补充 |

# 目 录

# 1. API 接口说明

## 1.1 Dobot 指令简介

控制器支持两种类型的指令：立即指令与队列指令。

- 立即指令：Dobot控制器在收到指令后立即处理该指令，而不管当前控制器是否在还在处理其他指令。
- 队列指令：Dobot控制器在收到指令后会将该指令放入控制器内部的指令队列中，Dobot控制器将顺序执行指令。

关于 Dobot 指令更具体的内容，可查询 Dobot 通信协议手册。

## 1.2 指令超时

### 1.2.1 设置指令超时时间

如*1.1 Dobot指令简介*中介绍，发送给Dobot控制器的所有指令都带有返回。当通信链路干扰等问题造成指令错误时，控制器将无法识别该条指令且无法返回。因此，每条下发给控制器的指令都可设置一个超时时间。该指令超时时间可以通过以下的API进行设置。

表 1.1 设置指令超时时间

| | |
|---|---|
| 原型 | int SetCmdTimeout( unsigned int cmdTimeout) |
| 功能 | 设置指令超时时间。当下发一条指令后如果需在规定时间内返回，则可调用此接口设置超时时间，判断该指令是否超时 |
| 功能 | cmdTimeout：指令超时时间。单位：ms |
| 返回 | DobotCommunicate_NoError：无错误 |

## 1.3 连接/断开

### 1.3.1 搜索 Dobot

表 1.2 搜索 Dobot

| | |
|---|---|
| 原型 | int SearchDobot(char *doboNametList, uint32_t maxLen) |
| 功能 | 搜索Dobot，动态库将搜索到的Dobot信息存储，并使用ConnectDobot连接Dobot |
| 参数 | dobotNameList：字符串数组，Dobot动态库会将搜索到的串口或UDP信息写入到dobotNameList。一个典型的dobotNameList的格式为"COM1 COM3 COM6 192.168.0.5"，多个串口或IP地址以空格分开<br>maxLen：字符串最大长度，以避免内存溢出 |
| 返回 | Dobot 数量 |

### 1.3.2 连接 Dobot

表 1.3 连接 Dobot 控制器接口说明

| | |
|---|---|
| 原型 | int ConnectDobot(const char *portName, uint32_t baudrate, char *fwType, char *version) |
| 功能 | 连接 Dobot。其中，portName 可以从 int SearchDobot(char *dobotList, uint32_t maxLen)的 char *dobotList 中得到<br>若 portName 为空，并直接调用 ConnectDobot，则动态库将自动连接随机搜索到的 Dobot |
| 参数 | portName：Dobot 端口名。对于串口，其值为“COM3”。对于 UDP，可能是“192.168.0.5”<br>baudrate：波特率<br>fwType：固件类型。包括：Dobot 和 Marlin<br>version：版本号 |
| 返回 | DobotConnect_NoError：连接 Dobot 成功<br>DobotConnect_NotFound：未找到 Dobot 端口<br>DobotConnect_Occupied：Dobot 端口被占用 |

⚠注意

请提前安装所需的驱动以便使API接口能识别Dobot控制器接口，详情请查询Dobot用户手册。

### 1.3.3 断开 Dobot

表 1.4 断开 Dobot 接口说明

| | |
|---|---|
| 原型 | int DisconnectDobot(void) |
| 功能 | 断开 Dobot |
| 参数 | 无 |
| 返回 | DobotConnect_NoError：无错误 |

### 1.3.4 示例：连接示例

程序 1.1 连接示例

```
#include "DobotDll.h"


int split(char **dst, char* str, const char* spl)
{
    int n = 0;
```

```
    char *result = NULL;
    result = strtok(str, spl);
    while( result != NULL )
    {
        strcpy(dst[n++], result);
        result = strtok(NULL, spl);
    }
    return n;
}

int main(void)
{
    int maxDevCount = 100;
    int maxDevLen = 20;

    char *devsChr = new char[maxDevCount * maxDevLen]();
    char **devsList = new char*[maxDevCount]();
    for(int i=0; i<maxDevCount; i++)
        devsList[i] = new char[maxDevLen]();

    SearchDobot(devsChr, 1024);
    split(devsList, devsChr, " ");
    ConnectDobot(devsList[0], 115200, NULL, NULL, NULL);

//控制 Dobot

    DisconnectDobot();

    delete[] devsChr;
    for(int i=0; i<maxDevCount; i++)
        delete[] devsList[i];
    delete[] devsList;
}
```

## 1.4 指令队列控制

Dobot控制器中有一个存放指令的队列，以达到顺序存储和执行指令的目的。同时，通过启动和停止指令，还可实现丰富的异步操作。

⚠注意

只有将“isQueued”参数设置为“1”的指令才能加入指令队列。

### 1.4.1 执行队列中的指令

表 1.5 执行指令

| | |
|---|---|
| 原型 | int SetQueuedCmdStartExec(void) |
| 功能 | Dobot控制器开始循环查询指令队列，如果队列中有指令，则顺序取出并执行，执行完一条指令后才会取出下一条继续执行 |
| 参数 | 无 |
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

### 1.4.2 停止执行队列中的指令

表 1.6 停止执行指令

| | |
|---|---|
| 原型 | int SetQueuedCmdStopExec(void) |
| 功能 | Dobot 控制器停止循环查询队列并停止执行指令。在停止过程中若 Dobot 控制器正在执行一条指令，则待该指令执行完成后再停止。 |
| 参数 | 无 |
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

### 1.4.3 强制停止执行队列中的指令

表 1.7 强制停止执行指令

| | |
|---|---|
| 原型 | int SetQueuedCmdForceStopExec(void) |
| 功能 | Dobot 控制器停止循环查询队列并停止执行指令。在停止过程中若 Dobot 控制器正在执行一条指令，则该指令将会停止执行。 |
| 参数 | 无 |
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

### 1.4.4 示例：同步处理 PTP 指令下发和队列控制

PTP详细说明请参见*0*

PTP功能。

程序 1.2 同步处理 PTP 指令下发和队列控制

```
#include "DobotDll.h"


int main(void)
{
    uint64_t queuedCmdIndex = 0;
    PTPCmd      cmd;

    cmd.ptpMode = 0;
    cmd.x           = 200;
    cmd.y           = 0;
    cmd.z           = 0;
    cmd.r           = 0;

    ConnectDobot(NULL, 115200, NULL, NULL, NULL);

    SetQueuedCmdStartExec();
    SetPTPCmd(&cmd, true, &queuedCmdIndex);
    SetQueuedCmdStopExec();

    DisconnectDobot();
}
```

### 1.4.5 示例：异步处理 PTP 指令下发和队列控制

程序 1.3 异步处理 PTP 指令下发和队列控制

```
#include "DobotDll.h"


// 主线程
int main(void)
{
    ConnectDobot(NULL, 115200, NULL, NULL, NULL);
}


int onButtonClick()
{
```

```
    static bool flag = True;
    if (flag)
        SetQueuedCmdStartExec();
    else
        SetQueuedCmdStopExec();
}

// 子线程
int thread(void)
{
    uint64_t queuedCmdIndex = 0;
    PTPCmd    cmd;

    cmd.ptpMode = 0;
    cmd.x          = 200;
    cmd.y          = 0;
    cmd.z          = 0;
    cmd.r          = 0;

    while(true)
        SetPTPCmd(&cmd, true, &queuedCmdIndex);
}
```

### 1.4.6 下载指令

Dobot 控制器支持将指令下载到控制器外部 Flash 中，然后通过控制器上的按键触发执行，即脱机运行。

表 1.8 下载指令

| | |
|---|---|
| 原型 | int SetQueuedCmdStartDownload(uint32_t totalLoop, uint32_t linePerLoop) |
| 功能 | 下载指令，当需要脱机运行时可调用此接口 |
| 参数 | totalLoop：脱机运行总次数<br>linePerLoop：指令单次循环次数。循环次数必须和下发的指令数相同，且下发指令必须设置为队列模式 |
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

### 1.4.7 停止下载指令

表 1.9 停止指令队列下载接口说明

| | |
|---|---|
| 原型 | int SetQueuedCmdStopDownload(void) |
| 功能 | 停止下载指令，当需要脱机运行时可调用此接口 |
| 参数 | 无 |
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

### 1.4.8 示例：下载 PTP 指令

程序 1.4 下载 PTP 指令

```
#include "DobotDll.h"


int main(void)
{
    uint64_t queuedCmdIndex = 0;
    PTPCmd    cmd;


    cmd.ptpMode = 0;
    cmd.x          = 200;
    cmd.y          = 0;
    cmd.z          = 0;
    cmd.r          = 0;


    ConnectDobot(NULL, 115200, NULL, NULL, NULL);


//只下发一条 PTP 指令，linePerLoop 参数设为 1
//总共循环两次，实际脱机运行两条 PTP 指令
    SetQueuedCmdStartDownload(2, 1);
    SetPTPCmd(&cmd, true, &queuedCmdIndex);
    SetQueuedCmdStopDownload();
    DisconnectDobot();
}
```

指令下载的一般流程是：

1） 调用下载指令 API。

2） 发送指令并将指令设置为队列模式。

3） 调用停止下载指令 API。

### 1.4.9 清空指令队列

该接口可以清空Dobot控制器中的指令队列。

表 1.10 清除指令队列

| | |
|---|---|
| 原型 | int SetQueuedCmdClear(void) |
| 功能 | 清空指令队列 |
| 参数 | 无 |
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

### 1.4.10 获取指令索引

在 Dobot 控制器指令队列机制中，有一个 64 位内部计数器。当控制器每执行完一条指令时，该计数器将自动加一。通过该指令，可以查询当前执行完成的指令的索引。

表 1.11 获取指令队列当前索引接口说明

| | |
|---|---|
| 原型 | int GetQueuedCmdCurrentIndex(uint64_t *queuedCmdCurrentIndex) |
| 功能 | 获取当前执行完成的指令的索引 |
| 参数 | queuedCmdCurrentIndex：指令索引 |
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

### 1.4.11 示例：获取指令索引实现运动同步

程序 1.5 获取指令索引实现运动同步

```
#include "DobotDll.h"


int main(void)
{
    uint64_t queuedCmdIndex = 0;
    uint64_t executedCmdIndex = 0;
    PTPCmd    cmd;

    cmd.ptpMode = 0;
    cmd.x       = 200;
    cmd.y       = 0;
    cmd.z       = 0;
```

```
    cmd.r           = 0;

    ConnectDobot(NULL, 115200, NULL, NULL, NULL);

    SetQueuedCmdStartExec();
    SetPTPCmd(&cmd, true, &queuedCmdIndex);

// 通过比较队列索引实现运动同步
While(executedCmdIndex < queuedCmdIndex)
    GetQueuedCmdCurrentIndex(&executedCmdIndex);
    SetQueuedCmdStopExec();
    DisconnectDobot();
}
```

## 1.5 设备信息

### 1.5.1 设置设备序列号

表 1.12 设置设备序列号

| | |
|---|---|
| 原型 | int SetDeviceSN(const char *deviceSN) |
| 功能 | 设置设备序列号。该接口仅在出厂时有效（需要特殊密码） |
| 参数 | deviceSN：字符串指针 |
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

### 1.5.2 获取设备序列号

表 1.13 获取设备序列号

| | |
|---|---|
| 原型 | int GetDeviceSN(char *deviceSN, uint32_t maxLen) |
| 功能 | 获取设备序列号 |
| 参数 | deviceSN：字符串指针<br>maxLen：字符串最大长度，以避免溢出 |
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

### 1.5.3 设置设备名称

表 1.14 设置设备名称

| | |
|---|---|
| 原型 | int SetDeviceName(const char *deviceName) |
| 功能 | 设置设备名称。当有多台机器时，可调用该接口设置设备名以作区分 |
| 参数 | deviceName：字符串指针 |
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

### 1.5.4 获取设备名称

表 1.15 获取设备名称

| | |
|---|---|
| 原型 | int GetDeviceName(char *deviceName, uint32_t maxLen) |
| 功能 | 获取设备名称。当有多台机器时，可调用该接口设置设备名以作区分 |
| 参数 | deviceName：字符串指针<br>maxLen：字符串最大长度，以避免溢出 |
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

### 1.5.5 获取设备版本号

表 1.16 获取设备版本号

| | |
|---|---|
| 原型 | int GetDeviceVersion(uint8_t *majorVersion, uint8_t *minorVersion, uint8_t *revision) |
| 功能 | 获取设备版本信息 |
| 参数 | majorVersion：主版本<br>minorVersion：次版本<br>revision：修订版本 |
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

### 1.5.6 设置滑轨状态

表 1.17 设置滑轨状态

| | |
|---|---|
| 原型 | int SetDeviceWithL(bool isEnable, bool isQueued, uint64_t *queuedCmdIndex ) |
| 功能 | 设置滑轨状态。若使用滑轨套件，则需调用该接口<br>在下发滑轨相关的指令前，必须先设置滑轨状态 |

 

| | |
|---|---|
| 参数 | isEnable：1，开启滑轨。0，关闭滑轨<br>isQueued：是否将该指令加入指令队列<br>queuedCmdIndex：若选择将指令加入队列，则表示指令在队列的索引号。否则，该参数无意义 |
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_BufferFull：指令队列已满<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

### 1.5.7 获取滑轨状态

表 1.18 获取滑轨状态

| | |
|---|---|
| 原型 | int GetDeviceWithL(bool * isWithL) |
| 功能 | 获取滑轨状态 |
| 参数 | isEnable：1，开启。0，关闭 |
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

### 1.5.8 获取设备时钟

表 1.19 获取设备时钟

| | |
|---|---|
| 原型 | int GetDeviceTime(uint32_t *deviceTime) |
| 功能 | 获取设备时钟 |
| 参数 | deviceTime：设备时钟 |
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

## 1.6 实时位姿

在 DobotV2.0 中，Dobot 控制器根据以下信息计算出实时位姿的基准值。

- 底座码盘读数（可通过回零得到）。
- 大臂角度传感器读数（上电或者按小臂UNLOCK按键时）。
- 小臂角度传感器读数（上电或者按小臂UNLOCK按键时）。

在控制 Dobot 时，Dobot 控制器将基于实时位姿的基准值，以及实时运动状态，更新实时位姿。

### 1.6.1 获取机械臂实时位姿

表 1.20 获取实时位姿

| | |
|---|---|
| 原型 | int GetPose(Pose *pose) |
| 功能 | 获取机械臂实时位姿 |
| 参数 | Pose 定义：<br>typedef struct tagPose {<br>float x; //机械臂坐标系 x<br>float y; //机械臂坐标系 y<br>float z; //机械臂坐标系 z<br>float r; //机械臂坐标系 r<br>float jointAngle[4]; //机械臂关节轴(底座、大臂、小臂、末端)角度<br>}Pose;<br>Pose：Pose 指针 |
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

### 1.6.2 获取滑轨实时位置

表 1.21 获取滑轨实时位置

| | |
|---|---|
| 原型 | int GetPoseL(float * l) |
| 功能 | 获取导轨实时位姿 |
| 参数 | l：滑轨当前位置。单位 mm |
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

### 1.6.3 重设机器人实时位姿

在以下情况，需重新设置实时位姿的基准值：

- 角度传感器损坏。
- 角度传感器精度太差。

表 1.22 设置实时位姿的基准值

| | |
|---|---|
| 原型 | int ResetPose(bool manual, float rearArmAngle, float frontArmAngle) |
| 功能 | 重新设置机械臂实时位姿 |
| 参数 | manual：表示是否自动重设姿态。0：自动重设姿态，无需设rearArmAngle及frontArmAngle。1：需设置rearArmAngle和frontArmAngle<br>rearArmAngle：大臂角度 |

| | |
|---|---|
| | frontArmAngle：小臂角度 |
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

## 1.7 报警功能

### 1.7.1 获取系统报警状态

表 1.23 获取系统报警状态

| | |
|---|---|
| 原型 | int GetAlarmsState(uint8_t *alarmsState, uint32_t *len, unsigned int maxLen) |
| 功能 | 获取系统报警状态 |
| 参数 | alarmsState：数组首地址。每一个字节可以标识8个报警项的报警状态，且MSB（Most Significant Bit）在高位，LSB（Least Significant Bit）在低位<br>len：报警所占字节<br>maxLen：数组最大长度，以避免溢出 |
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

### 1.7.2 清除系统所有报警

表 1.24 清除系统所有报警

| | |
|---|---|
| 原型 | int ClearAllAlarmsState(void) |
| 功能 | 清除系统所有报警 |
| 参数 | 无 |
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

## 1.8 回零功能

如果机械臂运行速度过快或者负载过大可能会导致精度降低，此时可执行回零操作，提高精度。

### 1.8.1 设置回零位置

表 1.25 设置回零位置

| | |
|---|---|
| 原型 | int SetHOMEParams(HOMEParams *homeParams, bool isQueued, uint64_t *queuedCmdIndex) |
| 功能 | 设置回零位置 |

| | |
|---|---|
| 参数 | HOMEParams 定义：<br>typedef struct tagHOMEParams {<br>float x; //机械臂坐标系 x<br>float y; //机械臂坐标系 y<br>float z; //机械臂坐标系 z<br>float r; //机械臂坐标系 r<br>}HOMEParams;<br>homeParams：HOMEParams 指针<br>isQueued：是否将该指令加入指令队列<br>queuedCmdIndex：若选择将指令加入队列，则表示指令在队列的索引号。否则，该参数无意义 |
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_BufferFull：指令队列已满<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

### 1.8.2 获取回零位置

表 1.26 获取回零位置

| | |
|---|---|
| 原型 | int GetHOMEParams(HOMEParams *homeParams) |
| 功能 | 获取回零位置 |
| 参数 | HOMEParams 定义：<br>typedef struct tagHOMEParams {<br>float x; //机械臂坐标系 x<br>float y; //机械臂坐标系 y<br>float z; //机械臂坐标系 z<br>float r; //机械臂坐标系 r<br>}HOMEParams;<br>homeParams：HOMEParams 指针 |
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

### 1.8.3 执行回零功能

表 1.27 执行回零功能接口说明

| | |
|---|---|
| 原型 | int SetHOMECmd(HOMECmd *homeCmd, bool isQueued, uint64_t *queuedCmdIndex) |

| | |
|---|---|
| 功能 | 执行回零功能。调用该接口前如果未调用 SetHOMEParams 接口，则表示直接回零至系统设置的位置。如果调用了 SetHOMEParams 接口，则回零至用户自定义的位置 |
| 参数 | HOMECmd 定义：<br>typedef struct tagHOMECmd {<br>uint32_t reserved; //保留<br>}HOMECmd;<br>homeCmd：HOMECmd 指针<br>isQueued：是否将该指令加入指令队列<br>queuedCmdIndex：若选择将指令加入队列，则表示指令在队列的索引号。否则，该参数无意义 |
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_BufferFull：指令队列已满<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

### 1.8.4 执行自动调平功能

如果机械臂大小臂角度传感器出现偏差，导致机械臂定位精度降低，可执行调平功能。如果对定位精度要求较高，需手动调平，详情请参见《Dobot Magicain 用户手册》。

表 1.28 执行自动调平功能接口说明

| | |
|---|---|
| 原型 | int SetAutoLevelingCmd(AutoLevelingCmd *autoLevelingCmd, bool isQueued, uint64_t *queuedCmdIndex) |
| 功能 | 执行自动调平功能 |
| 参数 | AutoLevelingCmd 定义：<br>typedef struct tagAutoLevelingCmd {<br>uint8_t controlFlag; //使能标志<br>float precision; //调平精度，最小值为 0.02<br>}AutoLevelingCmd;<br>autoLevelingCmd：AutoLevelingCmd 指针<br>isQueued：是否将该指令加入指令队列<br>queuedCmdIndex：若选择将指令加入队列，则表示指令在队列的索引号。否则，该参数无意义 |
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_BufferFull：指令队列已满<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

### 1.8.5 获取自动调平结果

表 1.29 获取自动调平结果

| | |
|---|---|
| 原型 | int GetAutoLevelingResult(float *precision) |
| 功能 | 获取自动调平结果 |
| 参数 | Precision：精度 |
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

## 1.9 HHT 功能

HHT（Hand-hold Teaching）即手持示教。默认情况下，用户按住小臂的圆形解锁按钮，可拖动机械臂到任意位置，松开按钮就可以自动保存一个存点。

### 1.9.1 设置触发模式

表 1.30 设置手持示教触发模式

| | |
|---|---|
| 原型 | int SetHHTTrigMode (HHTTrigMode hhtTrigMode) |
| 功能 | 设置手持示教信号触发模式。如果不调用该函数，则默认按键释放时触发 |
| 参数 | HHTTrigMode：<br>typedef enum tagHHTTrigMode{<br>TriggeredOnKeyReleased, //按键释放时触发<br>TriggeredOnPeriodicInterval //按键被按下的过程中触发<br>}HHTTrigMode;<br>hhtTrigMode：HHTTrigMode 枚举 |
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

### 1.9.2 获取触发模式

表 1.31 获取手持示教触发模式

| | |
|---|---|
| 原型 | int GetHHTTrigMode (HHTTrigMode *hhtTrigMode) |
| 功能 | 获取手持示教信号触发模式 |
| 参数 | typedef enum tagHHTTrigMode{<br>TriggeredOnKeyReleased, //按键释放时触发<br>TriggeredOnPeriodicInterval //按键被按下的过程中定时触发<br>}HHTTrigMode; |

| | |
|---|---|
| | hhtTrigMode：HHTTrigMode 枚举 |
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

### 1.9.3 设置手持示教使能状态

表 1.32 设置手持示教使能状态

| | |
|---|---|
| 原型 | int SetHHTTrigOutputEnabled (bool isEnabled) |
| 功能 | 设置手持示教状态 |
| 参数 | isEnabled：0，去使能。1，使能 |
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

### 1.9.4 获取手持示教功能使能状态

表 1.33 获取手持示教使能状态

| | |
|---|---|
| 原型 | int GetHHTTrigOutputEnabled (bool *isEnabled) |
| 功能 | 获取手持示教使能状态 |
| 参数 | isEnabled：0，去使能。1，使能 |
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

### 1.9.5 获取手持示教触发信号

表 1.34 获取手持示教信号

| | |
|---|---|
| 原型 | int GetHHTTrigOutput(bool *isTriggered) |
| 功能 | 获取手持示教信号。需调用 SetHHTTrigOutputEnabled 接口后才能使用 |
| 参数 | isTriggered：0，未触发。1，已触发 |
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

### 1.9.6 示例：手持示教

程序 1.6 手持示教

```
#include "DobotDll.h"
#include <queue>
#include <windows.h>
```

```
int main(void)
{
    ConnectDobot(NULL, 115200, NULL, NULL, NULL);

    SetHHTTrigMode(TriggeredOnPeriodicInterval);
    SetHHTTrigOutputEnabled(true);

    bool isTriggered = false;
    queue<Pose> poseQueue;
    Pose pose;
    while(true) {
        GetHHTTrigOutput(&isTriggered);
        if(isTriggered) {
            GetPose(&pose);
            poseQueue.push(pose);
        }
    }

    DisconnectDobot();
}
```

## 1.10 末端执行器

### 1.10.1 设置末端坐标偏移量

表 1.35 设置末端坐标偏移量

| | |
|---|---|
| 原型 | int SetEndEffectorParams(EndEffectorParams *endEffectorParams, bool isQueued, uint64_t *queuedCmdIndex) |
| 功能 | 设置末端坐标偏移参数，一般在末端安装了执行器才需设置<br>当使用标准的末端执行器（机械臂配套的末端执行器）时，请查询 Dobot 用户手册，得到其 X 轴与 Y 轴偏置，并调用本接口设置。其他情况下的末端执行器参数，需自行确认结构参数。 |
| 参数 | EndEffectorParams 定义：<br>typedef struct tagEndEffectorParams {<br>float xBias; //末端 X 方向偏移量<br>float yBias; //末端 Y 方向偏移量<br>float zBias; //末端 Z 方向偏移量<br>}EndEffectorParams; |

| | |
|---|---|
| | endEffectorParams：EndEffectorParams 指针<br>isQueued：是否将该指令加入指令队列<br>queuedCmdIndex：若选择将指令加入队列，则表示指令在队列的索引号。否则，该参数无意义 |
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_BufferFull：指令队列已满<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

### 1.10.2 获取末端坐标偏移量

表 1.36 获取末端坐标偏移量

| | |
|---|---|
| 原型 | int GetEndEffectorParams(EndEffectorParams *endEffectorParams) |
| 功能 | 获取末端坐标偏移量 |
| 参数 | EndEffectorParams 定义：<br>typedef struct tagEndEffectorParams {<br>float xBias; //末端 X 方向偏移量<br>float yBias; //末端 Y 方向偏移量<br>float zBias; //末端 Z 方向偏移量<br>}EndEffectorParams;<br>endEffectorParams：EndEffectorParams 指针 |
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

### 1.10.3 设置激光状态

表 1.37 设置激光状态

| | |
|---|---|
| 原型 | int SetEndEffectorLaser(bool enableCtrl, bool on, bool isQueued, uint64_t *queuedCmdIndex) |
| 功能 | 设置激光状态 |
| 参数 | enableCtrl：末端使能。0，未使能。1，使能<br>on：开启或停止激光。0，停止。1，开启<br>isQueued：是否将该指令加入指令队列<br>queuedCmdIndex：若选择将指令加入队列，则表示指令在队列的索引号。否则，该参数无意义 |
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_BufferFull：指令队列已满 |

| | |
|---|---|
| | DobotCommunicate_Timeout：指令无返回，导致超时 |

### 1.10.4 获取激光状态

表 1.38 获取激光状态

| | |
|---|---|
| 原型 | int GetEndEffectorLaser(bool *isCtrlEnabled, bool *isOn) |
| 功能 | 获取激光状态 |
| 参数 | isCtrlEnabled：末端是否使能。0，未使能.1，使能<br>isOn：激光是否开启。0，停止.1，开启 |
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

### 1.10.5 设置气泵状态

表 1.39 设置气泵状态

| | |
|---|---|
| 原型 | int SetEndEffectorSuctionCup(bool enableCtrl, bool suck, bool isQueued, uint64_t *queuedCmdIndex) |
| 功能 | 设置气泵状态 |
| 参数 | enableCtrl：末端使能。0，未使能.1，使能<br>suck：控制气泵吸气或吹气。0：吹气。1，吸气<br>isQueued：是否将该指令加入指令队列<br>queuedCmdIndex：若选择将指令加入队列，则表示指令在队列的索引号。否则，该参数无意义 |
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_BufferFull：指令队列已满<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

### 1.10.6 获取气泵状态

表 1.40 获取气泵状态

| | |
|---|---|
| 原型 | int GetEndEffectorSuctionCup(bool *isCtrlEnabled, bool *isSucked) |
| 功能 | 获取气泵状态 |
| 参数 | isCtrlEnabled：末端是否使能。0，未使能。1，使能<br>isSucked:气泵是否吸气。0：吹气。1，吸气 |
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

### 1.10.7 设置夹爪状态

表 1.41 设置夹爪状态

| | |
|---|---|
| 原型 | int SetEndEffectorGripper(bool enableCtrl, bool grip, bool isQueued, uint64_t *queuedCmdIndex) |
| 功能 | 设置夹爪状态 |
| 参数 | enableCtrl：末端使能。0，未使能。1，使能<br>grip：控制夹爪抓取或释放。0，释放。1，抓取<br>isQueued：是否将该指令加入指令队列<br>queuedCmdIndex：若选择将指令加入队列，则表示指令在队列的索引号。否则，该参数无意义 |
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_BufferFull：指令队列已满<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

### 1.10.8 获取夹爪状态

表 1.42 获取状态接口说明

| | |
|---|---|
| 原型 | int GetEndEffectorGripper(bool *isCtrlEnabled, bool *isGripped) |
| 功能 | 获取夹爪状态 |
| 参数 | isCtrlEnabled：末端是否使能。0，未使能。1，使能<br>isGripped：夹爪是否抓取。0，释放。1，抓取 |
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

## 1.11 JOG 功能

### 1.11.1 设置点动时各关节坐标轴的动速度和加速度

表 1.43 设置点动时各关节坐标轴的速度和加速度

| | |
|---|---|
| 原型 | int SetJOGJointParams(JOGJointParams *jogJointParams, bool isQueued, uint64_t *queuedCmdIndex) |
| 功能 | 设置点动时各关节坐标轴的速度和加速度 |
| 参数 | JOGJointParams 定义：<br>typedef struct tagJOGJointParams {<br>float velocity[4]; //4 轴关节速度 |

| | |
|---|---|
| | float acceleration[4]; //4 轴关节加速度<br>}JOGJointParams;<br>jogJointParam：JOGJointParams 指针<br>isQueued：是否将该指令加入指令队列<br>queuedCmdIndex：若选择将指令加入队列，则表示指令在队列的索引号。否则，该参数无意义 |
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_BufferFull：指令队列已满<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

### 1.11.2 获取点动时各关节坐标轴的动速度和加速度

表 1.44 获取点动时各关节坐标轴的速度和加速度

| | |
|---|---|
| 原型 | int GetJOGJointParams(JOGJointParams *jogJointParams) |
| 功能 | 获取点动时各关节坐标轴的速度和加速度 |
| 参数 | JOGJointParams 定义：<br>typedef struct tagJOGJointParams {<br>float velocity[4]; //4 轴关节速度<br>float acceleration[4]; //4 轴关节加速度<br>}JOGJointParams<br>jogJointParams：JOGJointParams 指针 |
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

### 1.11.3 设置点动时笛卡尔坐标轴的速度和加速度

表 1.45 设置点动时笛卡尔坐标轴的速度和加速度

| | |
|---|---|
| 原型 | int SetJOGCoordinateParams(JOGCoordinateParams *jogCoordinateParams,<br>bool isQueued, uint64_t *queuedCmdIndex) |
| 功能 | 设置点动时各坐标轴（笛卡尔）的速度和加速度 |
| 参数 | JOGCoordinateParams 定义：<br>typedef struct tagJOGCoordinateParams {<br>float velocity[4]; //4 轴坐标轴 X,Y,Z,R 速度<br>float acceleration[4]; //4 轴坐标轴 X,Y,Z,R 加速度<br>}JOGCoordinateParams;<br>jogCoordinateParams：JOGCoordinateParams 指针 |

| | |
|---|---|
| | isQueued：是否将该指令加入指令队列<br>queuedCmdIndex：若选择将指令加入队列，则表示指令在队列的索引号。否则，该参数无意义 |
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_BufferFull：指令队列已满<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

### 1.11.4 获取点动时笛卡尔坐标轴的速度和加速度

表 1.46 获取点动时笛卡尔坐标轴的速度和加速度

| | |
|---|---|
| 原型 | int GetJOGCoordinateParams(JOGCoordinateParams *jogCoordinateParams) |
| 功能 | 获取点动时各坐标轴（笛卡尔）的速度和加速度 |
| 参数 | JOGCoordinateParams 定义：<br>typedef struct tagJOGCoordinateParams {<br>float velocity[4]; //4 轴坐标轴 X,Y,Z,R 速度<br>float acceleration[4]; //4 轴坐标轴 X,Y,Z,R 加速度<br>}JOGCoordinateParams;<br>jogCoordinateParams： JOGCoordinateParams 指针 |
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

### 1.11.5 设置点动时滑轨速度和加速度

表 1.47 设置点动时滑轨速度和加速度

| | |
|---|---|
| 原型 | int SetJOGLParams(JOGLParams *jogLParams, bool isQueued, uint64_t *queuedCmdIndex) |
| 功能 | 设置点动时滑轨的速度和加速度 |
| 参数 | jogLParams 定义：<br>typedef struct tagJogLParams {<br>float velocity; //滑轨速度<br>float acceleration; //滑轨加速度<br>}JogLParams;<br>jogLParams：jogLParams 指针<br>isQueued：是否将该指令加入指令队列<br>queuedCmdIndex：若选择将指令加入队列，则表示指令在队列的索引号。否则，该参数无意义 |

| | |
|---|---|
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_BufferFull：指令队列已满<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

### 1.11.6 获取点动时滑轨速度和加速度

表 1.48 获取点动时滑轨速度和加速度

| | |
|---|---|
| 原型 | int GetJOGLParams(JOGLParams * jogLParams ) |
| 功能 | 获取点动时滑轨的速度和加速度 |
| 参数 | jogLParams 定义：<br>typedef struct tagJogLParams {<br>float velocity; //滑轨速度<br>float acceleration; //滑轨加速度<br>}JogLParams;<br>jogLParams：jogLParams 指针 |
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

### 1.11.7 设置点动速度百分比和加速度百分比

表 1.49 设置点动速度百分比和加速度百分比

| | |
|---|---|
| 原型 | int SetJOGCommonParams(JOGCommonParams *jogCommonParams, bool isQueued, uint64_t *queuedCmdIndex) |
| 功能 | 设置点动（关节坐标系下和笛卡尔坐标系下）速度百分比和加速度百分比 |
| 参数 | JOGCommonParams 定义：<br>typedef struct tagJOGCommonParams {<br>float velocityRatio; //速度比例，关节坐标轴点动和笛卡尔坐标轴点动共用<br>float accelerationRatio; //加速度比例，关节坐标轴点动和笛卡尔坐标轴点动共用<br>}JOGCommonParams;<br>jogCommonParams：JOGCommonParams 指针<br>isQueued：是否将该指令加入指令队列<br>queuedCmdIndex：若选择将指令加入队列，则表示指令在队列的索引号。否则，该参数无意义 |
| 返回 | DobotCommunicate_NoError：指令正常返回 |

| | |
|---|---|
| | DobotCommunicate_BufferFull：指令队列已满<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

### 1.11.8 获取点动速度百分比和加速度百分比

表 1.50 获取点动速度百分比和加速度百分比

| | |
|---|---|
| 原型 | int GetJOGCommonParams(JOGCommonParams *jogCommonParams) |
| 功能 | 获取点动（关节坐标系下和笛卡尔坐标系下）速度百分比和加速度百分比 |
| 参数 | JOGCommonParams 定义：<br>typedef struct tagJOGCommonParams {<br>float velocityRatio; //速度比例，关节点动和坐标轴点动共用<br>float accelerationRatio; //加速度比例，关节点动和坐标轴点动共用<br>}JOGCommonParams;<br>jogCommonParams：JOGCommonParams 指针 |
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

### 1.11.9 执行点动指令

表 1.51 执行点动指令接口说明

| | |
|---|---|
| 原型 | int SetJOGCmd(JOGCmd *jogCmd, bool isQueued, uint64_t *queuedCmdIndex) |
| 功能 | 执行点动指令。设置点动相关参数后可调用该接口 |
| 参数 | JOGCmd 定义：<br>typedef struct tagJOGCmd {<br>uint8_t isJoint; //点动方式：0，笛卡尔坐标轴点动；1，关节点动<br>uint8_t cmd; //点动命令（取值范围 0~10）<br>}JOGCmd;<br>//点动命令详细说明<br>enum {<br>IDLE, //空闲状态<br>AP_DOWN, //X+/Joint1+<br>AN_DOWN, //X-/Joint1-<br>BP_DOWN, //Y+/Joint2+<br>BN_DOWN, //Y-/Joint2-<br>CP_DOWN, //Z+/Joint3+<br>CN_DOWN, //Z-/Joint3- |

| | |
|---|---|
| | DP_DOWN, //R+/Joint4+<br>DN_DOWN, //R-/Joint4-<br>LP_DOWN, //L+<br>LN_DOWN //L-<br>};<br>jogCmd：JOGCmd 指针<br>isQueued：是否将该指令加入指令队列<br>queuedCmdIndex：若选择将指令加入队列，则表示指令在队列的索引号。否则，该参数无意义 |
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_BufferFull：指令队列已满<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

## 1.12 PTP 功能

PTP点位模式点位模式即实现点到点运动，Dobot M1的点位模式包括MOVJ、MOVL以及JUMP三种运动模式。不同的运动模式，示教后存点回放的运动轨迹不同。

- MOVJ：关节运动，由A点运动到B点，各个关节从A点对应的关节角运行至B点对应的关节角。关节运动过程中，各个关节轴的运行时间需一致，且同时到达终点，如图 1.1所示。

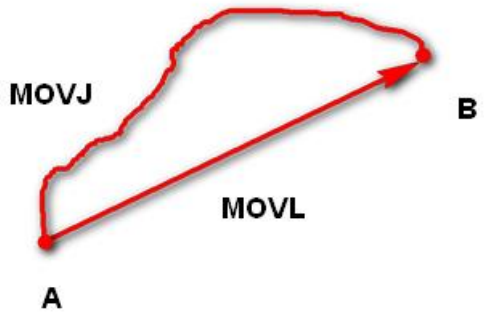


图 1.1 MOVL 和 MOVJ 运动模式

- MOVL：直线运动，A点到B点的路径为直线，如图 1.1所示。
- JUMP：门型轨迹，如图 1.2所示，由A点到B点的JUMP运动，先抬升高度Height，再平移到B点上方Height处，然后下降Height。


图 1.2 JUMP 运动模式

### 1.12.1 设置 PTP 模式下各关节坐标轴的速度和加速度

表 1.52 设置 PTP 模式下各关节坐标轴的速度和加速度

| | |
|---|---|
| 原型 | int SetPTPJointParams(PTPJointParams *ptpJointParams, bool isQueued, uint64_t *queuedCmdIndex) |
| 功能 | 设置 PTP 运动时各关节坐标轴的速度和加速度 |
| 参数 | PTPJointParams 定义：<br>typedef struct tagPTPJointParams{<br>float velocity[4]; //PTP 模式下 4 轴关节速度<br>float acceleration[4]; //PTP 模式下 4 轴关节加速度<br>}PTPJointParams;<br>ptpJointParams：PTPJointParams 指针<br>isQueued：是否将该指令指定为队列命令<br>queuedCmdIndex：若选择将指令加入队列，则表示指令在队列的索引号。否则，该参数无意义 |
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_BufferFull：指令队列已满<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

### 1.12.2 获取 PTP 模式下各关节坐标轴的速度和加速度

表 1.53 获取关节点位参数接口说明

| | |
|---|---|
| 原型 | int GetPTPJointParams(PTPJointParams *ptpJointParams) |
| 功能 | 获取 PTP 运动时各关节坐标轴的速度和加速度 |
| 参数 | PTPJointParams 定义：<br>typedef struct tagPTPJointParams {<br>float velocity[4]; //PTP 模式下 4 轴关节速度 |

| | |
|---|---|
| | float acceleration[4]; //PTP 模式下 4 轴关节加速度<br>}PTPJointParams;<br>ptpJointParams：PTPJointParams 指针 |
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

### 1.12.3 设置 PTP 模式下各笛卡尔坐标轴的速度和加速度

表 1.54 设置 PTP 运动时各笛卡尔关节坐标轴的速度和加速度

| | |
|---|---|
| 原型 | int SetPTPCoordinateParams(PTPCoordinateParams *ptpCoordinateParams, bool isQueued, uint64_t *queuedCmdIndex) |
| 功能 | 设置 PTP 运动时各笛卡尔坐标轴的速度和加速度 |
| 参数 | PTPCoordinateParams 定义：<br>typedef struct tagPTPCoordinateParams {<br>float xyzVelocity; //PTP 模式下 X,Y,Z 3 轴坐标轴速度<br>float rVelocity; //PTP 模式下末端 R 轴速度<br>float xyzAcceleration; //PTP 模式下 X,Y,Z 3 轴坐标轴加速度<br>float rAccleration; //PTP 模式下末端 R 轴加速度<br>} PTPCoordinateParams;<br>ptpCoordinateParams：PTPCoordinateParams 指针<br>isQueued:是否将该指令指定为队列命令<br>queuedCmdIndex：若选择将指令加入队列，则表示指令在队列的索引号。否则，该参数无意义 |
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_BufferFull：指令队列已满<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

### 1.12.4 获取 PTP 模式下各笛卡尔坐标轴的速度和加速度

表 1.55 获取 PTP 运动时各笛卡尔关节坐标轴的速度 s 和加速度

| | |
|---|---|
| 原型 | int GetPTPCoordinateParams(PTPCoordinateParams *ptpCoordinateParams) |
| 功能 | PTP 运动时各笛卡尔坐标轴的速度和加速度 |
| 参数 | PTPCoordinateParams 定义：<br>typedef struct tagPTPCoordinateParams {<br>float xyzVelocity; //PTP 模式下 X,Y,Z 3 轴坐标轴速度<br>float rVelocity; //PTP 模式下末端 R 轴速度 |

| | |
|---|---|
| | float xyzAcceleration; //PTP 模式下 X,Y,Z 3 轴坐标轴加速度<br>float rAccleration; //PTP 模式下末端 R 轴加速度<br>} PTPCoordinateParams;<br>ptpCoordinateParams：PTPCoordinateParams 指针 |
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

### 1.12.5 设置 JUMP 模式下抬升高度和最大抬升高度

表 1.56 设置 JUMP 模式下抬升高度和最大抬升高度

| | |
|---|---|
| 原型 | int SetPTPJumpParams(PTPJumpParams *ptpJumpParams, bool isQueued, uint64_t *queuedCmdIndex) |
| 功能 | 设置 JUMP 模式下抬升高度和最大抬升高度 |
| 参数 | PTPJumpParams 定义：<br>typedef struct tagPTPJumpParams {<br>float jumpHeight; //抬升高度<br>float zLimit; //最大抬升高度<br>}PTPJumpParams;<br>ptpJumpParams：PTPJumpParams 指针<br>isQueued：是否将该指令指定为队列命令<br>queuedCmdIndex：若选择将指令加入队列，则表示指令在队列的索引号。否则，该参数无意义 |
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_BufferFull：指令队列已满<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

### 1.12.6 获取 JUMP 模式下抬升高度和最大抬升高度

表 1.57 JUMP 模式下抬升高度和最大抬升高度

| | |
|---|---|
| 原型 | int GetPTPJumpParams(PTPJumpParams *ptpJumpParams) |
| 功能 | 获取 JUMP 模式下抬升高度和最大抬升高度 |
| 参数 | PTPJumpParams 定义：<br>typedef struct tagPTPJumpParams {<br>float jumpHeight; //抬升高度 |

| | |
|---|---|
| | float zLimit; //最大抬升高度<br>}PTPJumpParams;<br>ptpJumpParams：PTPJumpParams 指针 |
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

### 1.12.7 设置 JUMP 模式下扩展参数

表 1.58 设置 JUMP 模式下扩展参数

| | |
|---|---|
| 原型 | int SetPTPJump2Params( PTPJump2Params *ptpJump2Params, bool isQueued, uint64_t *queuedCmdIndex) |
| 功能 | 设置 JUMP 模式下扩展参数 |
| 参数 | PTPJump2Params 定义：<br>typedef struct tagPTPJump2Params {<br>float startJumpHeight; //起始点抬升高度<br>float endJumpHeight; //结束点抬升高度<br>float zLimit; //最大抬升高度<br>}PTPJump2Params;<br>ptpJump2Params：PTPJump2Params 指针<br>isQueued：是否将该指令指定为队列命令<br>queuedCmdIndex：若选择将指令加入队列，则表示指令在队列的索引号。否则，该参数无意义 |
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_BufferFull：指令队列已满<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

### 1.12.8 获取置 JUMP 模式下扩展参数

表 1.59 获取 JUMP 模式下扩展参数

| | |
|---|---|
| 原型 | int GetPTPJump2Params( PTPJump2Params *ptpJump2Params) |
| 功能 | 获取 JUMP 模式下扩展参数 |
| 参数 | PTPJump2Params 定义：<br>typedef struct tagPTPJump2Params {<br>float startJumpHeight; //起始点抬升高度<br>float endJumpHeight; //结束点抬升高度<br>float zLimit; //最大抬升高度 |

| | |
|---|---|
| | }PTPJump2Params;<br>ptpJump2Params：PTPJump2Params 指针 |
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

### 1.12.9 设置 PTP 模式下滑轨速度和加速度

表 1.60 设置 PTP 模式下滑轨速度和加速度

| | |
|---|---|
| 原型 | int SetPTPLParams(PTPLParams * ptpLParams,bool isQueued, uint64_t *queuedCmdIndex) |
| 功能 | 设置 PTP 模式下滑轨速度和加速度 |
| 参数 | PTPLParams 定义：<br>typedef struct tagPTPLParams {<br>float velocity; //滑轨速度<br>float acceleration; //滑轨加速度<br>}PTPLParams;<br>ptpLParams：PTPLParams 指针<br>isQueued：是否将该指令指定为队列命令<br>queuedCmdIndex：若选择将指令加入队列，则表示指令在队列的索引号。否则，该参数无意义 |
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_BufferFull：指令队列已满<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

### 1.12.10 获取 PTP 模式下滑轨速度和加速度

表 1.61 获取 PTP 模式下滑轨速度和加速度

| | |
|---|---|
| 原型 | int GetPTPLParams(PTPLParams *ptpLParams) |
| 功能 | 获取 PTP 模式下滑轨速度和加速度 |
| 参数 | PTPLParams 定义：<br>typedef struct tagPTPLParams {<br>float velocity; //滑轨速度<br>float acceleration; //滑轨加速度<br>}PTPLParams;<br>ptpLParams：PTPLParams 指针 |
| 返回 | DobotCommunicate_NoError：指令正常返回 |

| | |
|---|---|
| | DobotCommunicate_Timeout：指令无返回，导致超时 |

### 1.12.11 设置 PTP 运动的速度百分比和加速度百分比

表 1.62 设置 PTP 运动的速度百分比和加速度百分比

| | |
|---|---|
| 原型 | int SetPTPCommonParams(PTPCommonParams *ptpCommonParams, bool isQueued, uint64_t *queuedCmdIndex) |
| 功能 | 设置 PTP 运动的速度百分比和加速度百分比 |
| 参数 | PTPCommonParams 定义：<br>typedef struct tagPTPCommonParams {<br>float velocityRatio; //PTP 模式速度百分比，关节坐标轴和笛卡尔坐标轴共用<br>float accelerationRatio; //PTP 模式加速度百分比，关节坐标轴和笛卡尔坐标轴共用<br>}PTPCommonParams;<br>ptpCommonParams：PTPCommonParams 指针<br>isQueued：是否将该指令指定为队列命令<br>queuedCmdIndex：若选择将指令加入队列，则表示指令在队列的索引号。否则，该参数无意义 |
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_BufferFull：指令队列已满<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

### 1.12.12 获取 PTP 运动的速度百分比和加速度百分比

表 1.63 PTP 运动的速度百分比和加速度百分比

| | |
|---|---|
| 原型 | int GetPTPCommonParams(PTPCommonParams *ptpCommonParams) |
| 功能 | PTP 运动的速度百分比和加速度百分比 |
| 参数 | PTPCommonParams 定义：<br>typedef struct tagPTPCommonParams {<br>float velocityRatio; //PTP 模式速度百分比，关节坐标轴和笛卡尔坐标轴共用<br>float accelerationRatio; //PTP 模式加速度百分比，关节坐标轴和笛卡尔坐标轴共用<br>}PTPCommonParams;<br>ptpCommonParams：PTPCommonParams 指针 |
| 返回 | DobotCommunicate_NoError：指令正常返回 |

| | |
|---|---|
| | DobotCommunicate_Timeout：指令无返回，导致超时 |

### 1.12.13 执行 PTP 指令

表 1.64 执行 PTP 指令

| | |
|---|---|
| 原型 | int SetPTPCmd(PTPCmd *ptpCmd, bool isQueued, uint64_t *queuedCmdIndex) |
| 功能 | 执行 PTP 指令。设置 PTP 相关参数后，调用此函数可使机械臂运动至设置的目标点。 |
| 参数 | PTPCmd 定义：<br>typedef struct tagPTPCmd {<br>uint8_t ptpMode; //PTP 模式，取值范围：0~9<br>float x; //（x,y,z,r）为坐标参数，可为笛卡尔坐标、关节坐标、笛卡尔坐标增量或关节坐标增量<br>float y;<br>float z;<br>float r;<br>}PTPCmd;<br>//其中，ptpMode 取值如下：<br>enum {<br>JUMP_XYZ, //JUMP 模式，（x,y,z,r）为笛卡尔坐标系下的目标点坐标<br>MOVJ_XYZ, //MOVJ 模式，（x,y,z,r）为笛卡尔坐标系下的目标点坐标<br>MOVL_XYZ, //MOVL 模式，（x,y,z,r）为笛卡尔坐标系下的目标点坐标<br>JUMP_ANGLE, //JUMP 模式，（x,y,z,r）为关节坐标系下的目标点坐标<br>MOVJ_ANGLE, //MOVJ 模式，（x,y,z,r）为关节坐标系下的目标点坐标<br>MOVL_ANGLE, //MOVL 模式，（x,y,z,r）为关节坐标系下的目标点坐标<br>MOVJ_INC, //MOVJ 模式，（x,y,z,r）为关节坐标系下的坐标增量<br>MOVL_INC, //MOVL 模式，（x,y,z,r）为笛卡尔坐标系下的坐标增量<br>MOVJ_XYZ_INC, //MOVJ 模式，（x,y,z,r）为笛卡尔坐标系下的坐标增量<br>JUMP_MOVL_XYZ, //JUMP 模式，平移时运动模式为 MOVL。（x,y,z,r）为笛卡尔坐标系下的坐标增量 |

| | |
|---|---|
| | };<br>ptpCmd：PTPCmd 指针<br>isQueued：是否将该指令指定为队列命令<br>queuedCmdIndex：若选择将指令加入队列，则表示指令在队列的索引号。否则，该参数无意义 |
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_BufferFull：指令队列已满<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

### 1.12.14 执行带 I/O 控制的 PTP 指令

表 1.65 执行带 I/O 控制的 PTP 指令

| | |
|---|---|
| 原型 | int SetPTPPOCmd(PTPCmd *ptpCmd, ParallelOutputCmd *parallelCmd, int parallelCmdCount, bool isQueued, uint64_t *queuedCmdIndex) |
| 功能 | 执行带 I/O 控制的 PTP 指令。I/O 说明请参见《Dobot Magician 用户手册》 |
| 参数 | PTPCmd 定义：<br>typedef struct tagPTPCmd {<br>uint8_t ptpMode; //PTP 模式，取值范围：0~9<br>float x; //（x,y,z,r）为坐标参数，可为笛卡尔坐标、关节坐标、笛卡尔坐标增量或关节坐标增量<br>float y;<br>float z;<br>float r;<br>}PTPCmd;<br>//其中，ptpMode 取值如下：<br>enum {<br>JUMP_XYZ, //JUMP 模式，（x,y,z,r）为笛卡尔坐标系下的目标点坐标<br>MOVJ_XYZ, //MOVJ 模式，（x,y,z,r）为笛卡尔坐标系下的目标点坐标<br>MOVL_XYZ, //MOVL 模式，（x,y,z,r）为笛卡尔坐标系下的目标点坐标<br>JUMP_ANGLE, //JUMP 模式，（x,y,z,r）为关节坐标系下的目标点坐标<br>MOVJ_ANGLE, //MOVJ 模式，（x,y,z,r）为关节坐标系下的目标点坐标<br>MOVL_ANGLE, //MOVL 模式，（x,y,z,r）为关节坐标系下的目标点坐标 |

| | |
|---|---|
| | MOVJ_INC, //MOVJ 模式，（x,y,z,r）为关节坐标系下的坐标增量<br>MOVL_INC, //MOVL 模式，（x,y,z,r）为笛卡尔坐标系下的坐标增量<br>MOVJ_XYZ_INC, //MOVJ 模式，（x,y,z,r）为笛卡尔坐标系下的坐标增量<br>JUMP_MOVL_XYZ, //JUMP 模式，平移时运动模式为 MOVL。（x,y,z,r）为笛卡尔坐标系下的坐标增量<br>};<br>ParallelOutputCmd 定义：<br>typedef struct tagParallelOutputCmd {<br>uint8_t ratio; //设置运动时两点之间距离的百分比，即在该位置触发 I/O<br>uint16_t address; //I/O 地址。取值范围：1~20<br>uint8_t level; //输出值<br>}ParallelOutputCmd;<br>ptpCmd：PTPCmd 指针<br>parallelCmd：ParallelOutputCmd 指针<br>parallelCmdCount：I/O 个数<br>isQueued：是否将该指令指定为队列命令<br>queuedCmdIndex：若选择将指令加入队列，则表示指令在队列的索引号。否则，该参数无意义 |
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_BufferFull：指令队列已满<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

### 1.12.15 执行带滑轨的 PTP 指令

表 1.66 执行带滑轨的 PTP 指令

| | |
|---|---|
| 原型 | int SetPTPWithLCmd(PTPWithLCmd *ptpWithLCmd, bool isQueued, uint64_t *queuedCmdIndex) |
| 功能 | 执行带滑轨的 PTP 指令 |
| 参数 | PTPWithLCmd 定义：<br>typedef struct tagPTPWithLCmd {<br>uint8_t ptpMode; //PTP 模式，取值范围：0~9<br>float x; //（x,y,z,r）为坐标参数，可为笛卡尔坐标、关节坐标、笛卡尔坐标增量或关节坐标增量 |

| | |
|---|---|
| | float y;<br>float z;<br>float r;<br>float l; //滑轨运行距离<br>}PTPWithLCmd;<br>//其中，ptpMode 取值如下：<br>enum {<br>JUMP_XYZ, //JUMP 模式，（x,y,z,r）为笛卡尔坐标系下的目标点坐标<br>MOVJ_XYZ, //MOVJ 模式，（x,y,z,r）为笛卡尔坐标系下的目标点坐标<br>MOVL_XYZ, //MOVL 模式，（x,y,z,r）为笛卡尔坐标系下的目标点坐标<br>JUMP_ANGLE, //JUMP 模式，（x,y,z,r）为关节坐标系下的目标点坐标<br>MOVJ_ANGLE, //MOVJ 模式，（x,y,z,r）为关节坐标系下的目标点坐标<br>MOVL_ANGLE, //MOVL 模式，（x,y,z,r）为关节坐标系下的目标点坐标<br>MOVJ_INC, //MOVJ 模式，（x,y,z,r）为关节坐标系下的坐标增量<br>MOVL_INC, //MOVL 模式，（x,y,z,r）为笛卡尔坐标系下的坐标增量<br>MOVJ_XYZ_INC, //MOVJ 模式，（x,y,z,r）为笛卡尔坐标系下的坐标增量<br>JUMP_MOVL_XYZ, //JUMP 模式，平移时运动模式为 MOVL。（x,y,z,r）为笛卡尔坐标系下的坐标增量<br>};<br>ptpWithLCmd：PTPWithLCmd 指针<br>isQueued：是否将该指令指定为队列命令<br>queuedCmdIndex：若选择将指令加入队列，则表示指令在队列的索引号。否则，该参数无意义 |
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_BufferFull：指令队列已满<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

### 1.12.16 执行带 I/O 控制和滑轨的 PTP 指令

表 1.67 执行带 I/O 控制和滑轨的 PTP 指令

| | |
|---|---|
| 原型 | int SetPTPPOWithLCmd (PTPWithLCmd *ptpWithLCmd, ParallelOutputCmd *parallelCmd,int parallelCmdCount, bool isQueued, uint64_t *queuedCmdIndex) |
| 功能 | 执行带I/O控制和滑轨的PTP指令 |
| 参数 | PTPWithLCmd 定义：<br>typedef struct tagPTPWithLCmd {<br>uint8_t ptpMode; //PTP 模式，取值范围：0~9<br>float x; //（x,y,z,r）为坐标参数，可为笛卡尔坐标、关节坐标、笛卡尔坐标增量或关节坐标增量<br>float y;<br>float z;<br>float r;<br>float l; //滑轨运行距离<br>}PTPWithLCmd;<br>//其中，ptpMode 取值如下：<br>enum {<br>JUMP_XYZ, //JUMP 模式，（x,y,z,r）为笛卡尔坐标系下的目标点坐标<br>MOVJ_XYZ, //MOVJ 模式，（x,y,z,r）为笛卡尔坐标系下的目标点坐标<br>MOVL_XYZ, //MOVL 模式，（x,y,z,r）为笛卡尔坐标系下的目标点坐标<br>JUMP_ANGLE, //JUMP 模式，（x,y,z,r）为关节坐标系下的目标点坐标<br>MOVJ_ANGLE, //MOVJ 模式，（x,y,z,r）为关节坐标系下的目标点坐标<br>MOVL_ANGLE, //MOVL 模式，（x,y,z,r）为关节坐标系下的目标点坐标<br>MOVJ_INC, //MOVJ 模式，（x,y,z,r）为关节坐标系下的坐标增量<br>MOVL_INC, //MOVL 模式，（x,y,z,r）为笛卡尔坐标系下的坐标增量<br>MOVJ_XYZ_INC, //MOVJ 模式，（x,y,z,r）为笛卡尔坐标系下的坐标增量<br>JUMP_MOVL_XYZ, //JUMP 模式，平移时运动模式为 MOVL。（x,y,z,r）为笛卡尔坐标系下的坐标增量<br>};<br>ParallelOutputCmd 定义： |

| | |
|---|---|
| | typedef struct tagParallelOutputCmd {<br>uint8_t ratio; //设置运动时两点之间距离的百分比，即在该位置触发 I/O<br>uint16_t address; //I/O 地址。取值范围：1~20<br>uint8_t level; //输出值<br>}ParallelOutputCmd;<br>ptpCmd：PTPCmd 指针<br>parallelCmd：ParallelOutputCmd 指针<br>parallelCmdCount：I/O 个数<br>isQueued:是否将该指令指定为队列命令<br>queuedCmdIndex：若选择将指令加入队列，则表示指令在队列的索引号。否则，该参数无意义 |
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_BufferFull：指令队列已满<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

## 1.13 CP 功能

CP即连续运动轨迹。

### 1.13.1 设置 CP 运动的速度和加速度

表 1.68 设置连续轨迹功能参数接口说明

| | |
|---|---|
| 原型 | int SetCPParams(CPParams *cpParams, bool isQueued, uint64_t *queuedCmdIndex) |
| 功能 | 设置CP运动的速度和加速度 |
| 参数 | CPParams 定义：<br>typedef struct tagCPParams {<br>float planAcc; //规划加速度最大值<br>float junctionVel; //拐角速度最大值<br>union {<br>float acc; //实际加速度最大值，非实时模式时有效<br>float period; //插补周期，实时模式时有效<br>};<br>uint8_t realTimeTrack; //0：非实时模式：所有指令下发后再运行<br>1：实时模式：边下发指令边运行<br>}CPParams;<br>cpParams：CPParams 指针 |

| | |
|---|---|
| | isQueued：是否将该指令指定为队列命令<br>queuedCmdIndex：若选择将指令加入队列，则表示指令在队列的索引号。否则，该参数无意义 |
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_BufferFull：指令队列已满<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

### 1.13.2 获取 CP 运动的速度和加速度

表 1.69 获取连续轨迹功能参数接口说明

| | |
|---|---|
| 原型 | int GetCPParams(CPParams *cpParams) |
| 功能 | 获取连续轨迹模型下相关参数 |
| 参数 | CPParams 定义：<br>typedef struct tagCPParams {<br>float planAcc; //规划加速度最大值<br>float junctionVel; //拐角速度最大值<br>union {<br>float acc; //实际加速度最大值，非实时模式时有效<br>float period; //插补周期，实时模式时有效<br>};<br>uint8_t realTimeTrack; //0：非实时模式：所有指令下发后再运行<br>1：实时模式：边下发指令边运行<br>}CPParams;<br>cpParams：CPParams 指针 |
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

### 1.13.3 执行 CP 指令

表 1.70 执行 CP 指令

| | |
|---|---|
| 原型 | int SetCPCmd(CPCmd *cpCmd, bool isQueued, uint64_t *queuedCmdIndex) |
| 功能 | 执行 CP 指令 |
| 参数 | CPCmd 定义：<br>typedef struct tagCPCmd {<br>uint8_t cpMode; //CP 模式。0：相对模式，表示相对距离，即笛卡尔坐标增量。1：绝对模式，表示绝对距离， |

| | |
|---|---|
| | 即笛卡尔坐标系下目标点坐标<br>float x; //x,y,z 可以设置为坐标增量，也可设置为目的坐标点<br>float y;<br>float z;<br>union {<br>float velocity; //保留<br>float power; //保留<br>};<br>}CPCmd;<br>cpCmd：CPCmd 指针<br>isQueued：是否将该指令指定为队列命令<br>queuedCmdIndex：若选择将指令加入队列，则表示指令在队列的索引号。否则，该参数无意义 |
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_BufferFull：指令队列已满<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

⚠注意

当指令队列中有多条连续的CP指令时，Dobot控制器将自动前瞻。前瞻的条件是，队列中这些CP指令之间没有JOG、PTP、ARC、WAIT、TRIG等指令。

### 1.13.4 执行带激光雕刻的 CP 指令

表 1.71 执行带激光雕刻的 CP 指令

| | |
|---|---|
| 原型 | int SetCPLECmd (CPCmd *cpCmd, bool isQueued, uint64_t *queuedCmdIndex) |
| 功能 | 执行带激光雕刻的 CP 指令。此时激光功率将在运动命令下发时起效 |
| 参数 | CPCmd 定义：<br>typedef struct tagCPCmd {<br>uint8_t cpMode; //CP 模式。0：相对模式，表示相对距离，即笛卡尔坐标增量。1：绝对模式，表示绝对距离，即笛卡尔坐标系下目标点坐标<br>float x; //x,y,z 可以设置为坐标增量，也可设置为目的坐标点<br>float y;<br>float z; |

| | |
|---|---|
| | union {<br>float velocity; //保留<br>float power; //激光功率：0~100<br>}<br>}CPCmd;<br>cpCmd：CPCmd 指针<br>isQueued：是否将该指令指定为队列命令<br>queuedCmdIndex：若选择将指令加入队列，则表示指令在队列的索引号。否则，该参数无意义 |
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_BufferFull：指令队列已满<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

## 1.14 ARC 功能

圆弧模式即示教后存点回放的运动轨迹为圆弧。圆弧轨迹是空间的圆弧，由当前点、圆弧上任一点和圆弧结束点三点共同确定。圆弧总是从起点经过圆弧上任一点再到结束点，如图 1.3所示。

⚠注意

使用圆弧运动模式时，需结合其他运动模式确认圆弧上的三点，且三点不能在同一条直线上。

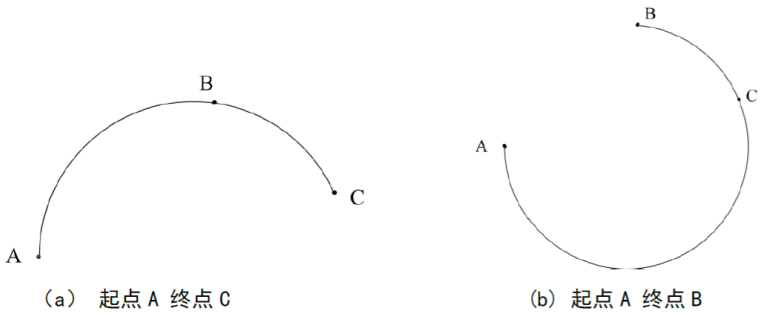


图 1.3 圆弧运动模式

### 1.14.1 设置 ARC 运动的速度和加速度

表 1.72 设置 ARC 运动的速度和加速度

| | |
|---|---|
| 原型 | int SetARCParams(ARCParams *arcParams, bool isQueued, uint64_t *queuedCmdIndex) |
| 功能 | 设置 ARC 运动的速度和加速度 |

| | |
|---|---|
| 参数 | ARCParams 定义：<br>typedef struct tagARCParams {<br>float xyzVelocity; //PTP 模式下 X,Y,Z 3 轴坐标轴速度<br>float rVelocity; //PTP 模式下末端 R 轴速度<br>float xyzAcceleration; //PTP 模式下 X,Y,Z 3 轴坐标轴加速度<br>float rAccleration; //PTP 模式下末端 R 轴加速度<br>} ARCParams;<br>arcParams：圆弧插补功能参数<br>isQueued：是否将该指令指定为队列命令<br>queuedCmdIndex：若选择将指令加入队列，则表示指令在队列的索引号。否则，该参数无意义 |
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_BufferFull：指令队列已满<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

### 1.14.2 获取 ARC 运动的速度和加速度

表 1.73 获取 ARC 运动的速度和加速度

| | |
|---|---|
| 原型 | int GetARCParams(ARCParams *arcParams) |
| 功能 | 获取 ARC 运动的速度和加速度 |
| 参数 | ARCParams 定义：<br>typedef struct tagARCParams {<br>float xyzVelocity; //PTP 模式下 X,Y,Z 3 轴坐标轴速度<br>float rVelocity; //PTP 模式下末端 R 轴速度<br>float xyzAcceleration; //PTP 模式下 X,Y,Z 3 轴坐标轴加速度<br>float rAccleration; //PTP 模式下末端 R 轴加速度<br>} ARCParams;<br>arcParams：ARCParams 指针 |
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

### 1.14.3 执行 ARC 指令

表 1.74 执行圆弧插补功能接口说明

| | |
|---|---|
| 原型 | int SetARCCmd(ARCCmd *arcCmd, bool isQueued, uint64_t *queuedCmdIndex) |

| | |
|---|---|
| 功能 | 执行 ARC 指令。设置 ARC 运动的速度和加速度后，调用该函数可使机械臂按 ARC 模式运动至目标点<br>该运动模式需结合其他模式一起使用，构成圆弧上三点 |
| 参数 | ARCCmd 定义：<br>typedef struct tagARCCmd {<br>struct {<br>float x;<br>float y;<br>float z;<br>float r;<br>}cirPoint; //圆弧中间点，需设置为笛卡尔坐标<br>struct {<br>float x;<br>float y;<br>float z;<br>float r;<br>}toPoint; //圆弧目标点，需设置为笛卡尔坐标<br>}ARCCmd;<br>arcCmd：ARCCmd 指针<br>isQueued：是否将该指令指定为队列命令<br>queuedCmdIndex：若选择将指令加入队列，则表示指令在队列的索引号。否则，该参数无意义 |
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_BufferFull：指令队列已满<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

### 1.14.4 执行 CIRCLE 指令

圆形模式与圆弧模式相似，示教后存点回放的运动轨迹为整圆。使用圆形模式时，也需结合其他运动模式确认圆形上的三点。

表 1.75 执行 CIRCLE 指令

| | |
|---|---|
| 原型 | int SetCircleCmd(CircleCmd *circleCmd, bool isQueued, uint64_t *queuedCmdIndex) |
| 功能 | 执行 CIRCLE 指令。设置整圆运动的速度和加速度后，调用该函数可使机械臂按 CIRCLE 模式运动至目标点<br>该运动模式需结合其他模式一起使用，构成圆上三点 |

| | |
|---|---|
| 参数 | CircleCmd 定义：<br>typedef struct tagCircleCmd{<br>struct {<br>float x;<br>float y;<br>float z;<br>float r;<br>}cirPoint; //圆弧中间点，需设置为笛卡尔坐标<br>struct {<br>float x;<br>float y;<br>float z;<br>float r;<br>}toPoint; //圆弧目标点，需设置为笛卡尔坐标<br>uint32_t count; //整圆个数<br>}CircleCmd;<br>circleCmd：CircleCmd 指针<br>isQueued：是否将该指令指定为队列命令<br>queuedCmdIndex：若选择将指令加入队列，则表示指令在队列的索引号。否则，该参数无意义 |
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_BufferFull：指令队列已满<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

## 1.15 丢步检测功能

### 1.15.1 设置丢步检测阈值

表 1.76 设置丢步检测阈值接口说明

| | |
|---|---|
| 原型 | int SetLostStepParams(float threshold) |
| 功能 | 设置丢步检测阈值，用于检测定位误差是否超过该阈值。如果超过该阈值，则说明电机丢步<br>如果用户不调用该接口，则丢步检测阈值默认为 5 |
| 参数 | Threshold：阈值 |
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_BufferFull：指令队列已满<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

### 1.15.2 执行丢步检测

表 1.77 设置丢步检测阈值接口说明

| | |
|---|---|
| 原型 | int SetLostStepCmd(bool isQueued, uint64_t *queuedCmdIndex) |
| 功能 | 执行丢步检测，当检测到丢步时会停止命令队列。该指令只能作为队列指令，即“isQueued”必须设置为“1” |
| 参数 | isQueued：是否将该指令指定为队列命令<br>queuedCmdIndex：若选择将指令加入队列，则表示指令在队列的索引号。否则，该参数无意义 |
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_BufferFull：指令队列已满<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

### 1.15.3 示例：丢步检测

程序 1.7 丢步检测

```
#include "DobotDll.h"


int main(void)
{
    uint64_t queuedCmdIndex = 0;
    PTPCmd    cmd;

    cmd.ptpMode = 0;
    cmd.x          = 200;
    cmd.y          = 0;
    cmd.z          = 0;
    cmd.r          = 0;

    ConnectDobot(NULL, 115200, NULL, NULL, NULL);

    SetQueuedCmdStartExec();
    SetPTPCmd(&cmd, true, &queuedCmdIndex);
    SetLostStepCmd(true, &queuedCmdIndex)
```

```
    SetQueuedCmdStopExec();


    DisconnectDobot();
}
```

## 1.16 WAIT 功能

### 1.16.1 执行时间等待指令

表 1.78 执行时间等待功能接口说明

| | |
|---|---|
| 原型 | int SetWAITCmd(WAITCmd *waitCmd, bool isQueued, uint64_t *queuedCmdIndex) |
| 功能 | 执行时间等待指令。如果需设置前个指令运行后的暂停时间，可调用此函数<br>该指令只能作为队列指令，“isQueued”必须设置为“1”。将该命令设置为立即指令可能会导致正在执行的 WAIT 队列指令的暂停时间变化 |
| 参数 | WAITCmd 定义：<br>typedef struct tagWAITCmd {<br>uint32_t timeout; //单位：ms<br>}WAITCmd;<br>waitCmd：时间等待功能变量<br>isQueued：是否将该指令指定为队列命令<br>queuedCmdIndex：若选择将指令加入队列，则表示指令在队列的索引号。否则，该参数无意义 |
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_BufferFull：指令队列已满<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

### 1.16.2 执行触发指令

表 1.79 执行触发指令

| | |
|---|---|
| 原型 | int SetTRIGCmd(TRIGCmd *trigCmd, bool isQueued, uint64_t *queuedCmdIndex) |
| 功能 | 执行触发指令<br>该指令只能作为队列指令，“isQueued”必须设置为“1”。将该命令设置为立即指令可能会导致正在执行的 TRIG 队列指令的触发条件变化。 |
| 参数 | TRIGCmd 定义：<br>typedef struct tagTRIGCmd {<br>uint8_t address; //I/O 地址。取值范围：1~20<br>uint8_t mode; //触发模式。0：电平触发。1：A/D 触发 |

| | |
|---|---|
| | uint8_t condition; //触发条件。<br>电平：0，等于。1，不等于<br>A/D：0，小于。1，小于等于<br>2，大于等于。3，大于<br>uint16_t threshold; //触发阈值：电平，0 或 1。A/D：0~4095<br>}TRIGCmd;<br>trigCmd：TRIGCmd 指针<br>isQueued：是否将该指令指定为队列命令<br>queuedCmdIndex：若选择将指令加入队列，则表示指令在队列的索引号。否则，该参数无意义 |
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_BufferFull：指令队列已满<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

## 1.17 EIO 功能

在 Dobot 控制器中，所有的扩展 I/O 都是统一编址的。根据现有情况，I/O 的功能包括以下内容：

- 高低电平输出功能。
- PWM输出功能。
- 读取输入高低电平功能。
- 读取输入模数转换值功能。

部分 I/O 可能同时具有以上的功能。在使用不同的功能时，需要先配置 I/O 的复用。I/O 详细说明，请参见《Dobot Magician 用户手册》。

### 1.17.1 设置 I/O 复用

表 1.80 设置 I/O 复用

| | |
|---|---|
| 原型 | int SetIOMultiplexing(IOMultiplexing *ioMultiplexing, bool isQueued, uint64_t *queuedCmdIndex) |
| 功能 | 设置 I/O 复用。在使用 I/O 前需要设置 I/O 复用功能。 |
| 参数 | IOMultiplexing 定义：<br>typedef struct tagIOMultiplexing {<br>uint8_t address; //I/O 地址。取值范围：1~20<br>uint8_t multiplex; //IO 功能。取值范围：0~6<br>}IOMultiplexing;<br>其中 mutiplex 支持的取值如下所示：<br>typedef enum tagIOFunction { |

| | |
|---|---|
| | IOFunctionDummy, //不配置功能<br>IOFunctionDO, //I/O 输出<br>IOFunctionPWM, //PWM 输出<br>IOFunctionDI, //I/O 输入<br>IOFunctionADC //A/D 输入<br>IOFunctionDIPU //上拉输入<br>IOFunctionDIPD //下拉输入<br>} IOFunction;<br>ioMultiplexing：IOMultiplexing 指针<br>isQueued：是否将该指令指定为队列命令<br>queuedCmdIndex：若选择将指令加入队列，则表示指令在队列的索引号。否则，该参数无意义 |
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_BufferFull：指令队列已满<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

### 1.17.2 读取 I/O 复用

表 1.81 读取 I/O 复用

| | |
|---|---|
| 原型 | int GetIOMultiplexing (IOMultiplexing *ioMultiplexing) |
| 功能 | 读取 I/O 复用 |
| 参数 | IOMultiplexing 定义：<br>typedef struct tagIOMultiplexing {<br>uint8_t address; //I/O 地址<br>uint8_t multiplex; //IO 功能。取值范围：0~6<br>}IOMultiplexing;<br>其中 mutiplex 支持的取值如下所示：<br>typedef enum tagIOFunction {<br>IOFunctionDummy, //不配置功能<br>IOFunctionDO, //I/O 输出<br>IOFunctionPWM, //PWM 输出<br>IOFunctionDI, //I/O 输入<br>IOFunctionADC //A/D 输入<br>IOFunctionDIPU //上拉输入<br>IOFunctionDIPD //下拉输入<br>} IOFunction;<br>ioMultiplexing：IOMultiplexing 指针 |

| | |
|---|---|
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

### 1.17.3 设置 I/O 输出电平

表 1.82 设置 I/O 输出电平

| | |
|---|---|
| 原型 | int SetIODO(IODO *ioDO, bool isQueued, uint64_t *queuedCmdIndex) |
| 功能 | 设置 I/O 输出电平 |
| 参数 | IODO 定义：<br>typedef struct tagIODO {<br>uint8_t address; //I/O 地址<br>uint8_t level; //输出电平。0：低电平。1：高电平<br>}IODO;<br>ioDO：IODO 指针<br>isQueued：是否将该指令指定为队列命令<br>queuedCmdIndex：若选择将指令加入队列，则表示指令在队列的索引号。否则，该参数无意义 |
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_BufferFull：指令队列已满<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

### 1.17.4 读取 I/O 输出电平

表 1.83 读取 I/O 输出电平

| | |
|---|---|
| 原型 | int GetIODO(IODO *ioDO) |
| 功能 | 读取 I/O 输出电平 |
| 参数 | IODO 定义：<br>typedef struct tagIODO {<br>uint8_t address; //I/O 地址<br>uint8_t level; //输出电平。0：低电平。1：高电平<br>}IODO;<br>ioDO：IODO 指针 |
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

### 1.17.5 设置 PWM 输出

表 1.84 设置 PWM 输出

| | |
|---|---|
| 原型 | int SetIOPWM(IOPWM *ioPWM, bool isQueued, uint64_t *queuedCmdIndex) |
| 功能 | 设置 PWM 输出 |
| 参数 | IOPWM 定义：<br>typedef struct tagIOPWM {<br>uint8_t address; //I/O 地址<br>float frequency; //PWM 频率。取值范围：10HZ~1MHz<br>float dutyCycle; //PWM 占空比。取值范围：0~100<br>}IOPWM;<br>ioPWM：IOPWM 指针<br>isQueued：是否将该指令指定为队列命令<br>queuedCmdIndex：若选择将指令加入队列，则表示指令在队列的索引号。否则，该参数无意义 |
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_BufferFull：指令队列已满<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

### 1.17.6 读取 PWM 输出

表 1.85 读取 PWM 输出

| | |
|---|---|
| 原型 | int GetIOPWM(IOPWM *ioPWM) |
| 功能 | 读取 PWM 输出 |
| 参数 | IOPWM 定义：<br>typedef struct tagIOPWM {<br>uint8_t address; //I/O 地址<br>float frequency; //PWM 频率。取值范围：10HZ~1MHz<br>float dutyCycle; //PWM 占空比。取值范围：0~100<br>}IOPWM;<br>ioPWM：IOPWM 指针 |
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

### 1.17.7 读取 I/O 输入电平

表 1.86 读取 I/O 输入电平

| | |
|---|---|
| 原型 | int GetIODI(IODI *ioDI) |

 

| | |
|---|---|
| 功能 | 读取 I/O 输入电平 |
| 参数 | IODI 定义：<br>typedef struct tagIODI {<br>uint8_t address; //I/O 地址<br>uint8_t level; //输入电平。0：低电平。1：高电平<br>}IODI;<br>ioDI：IODI 指针 |
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

### 1.17.8 读取 A/D 输入

表 1.87 读取 A/D 输入

| | |
|---|---|
| 原型 | int GetIOADC(IOADC *ioADC) |
| 功能 | 读取 A/D 输入 |
| 参数 | IOADC 定义：<br>typedef struct tagIOADC {<br>uint8_t address; //I/O 地址<br>uint16_t value; //A/D 输入值。取值范围：0~4095<br>}IOADC;<br>ioADC：IOADC 指针 |
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

### 1.17.9 设置扩展电机速度

表 1.88 设置扩展电机速度

| | |
|---|---|
| 原型 | int SetEMotor(EMotor *eMotor, bool isQueued, uint64_t *queuedCmdIndex) |
| 功能 | 设置扩展电机速度。调用此函数后电机会以一定的速度不停的运行 |
| 参数 | EMotor 定义：<br>typedef struct tagEMotor {<br>uint8_t index; //电机编号。0：Stepper1。 1：Stepper2<br>uint8_t isEnabled; //电机控制使能。0：未使能。1：使能<br>uint32_t speed; //电机控制速度（脉冲个数每秒）<br>}EMotor; |

| | |
|---|---|
| | eMotor：EMoto 指针<br>isQueued：是否将该指令指定为队列命令<br>queuedCmdIndex：若选择将指令加入队列，则表示指令在队列的索引号。否则，该参数无意义 |
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_BufferFull：指令队列已满<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

### 1.17.10 设置扩展电机速度和移动距离

表 1.89 设置扩展电机速度和移动距离接口说明

| | |
|---|---|
| 原型 | int SetEMotorS(EMotorS *eMotorS, bool isQueued, uint64_t *queuedCmdIndex) |
| 功能 | 设置扩展电机速度和移动距离。当需要以一定速度运行一段距离时可调用此函数 |
| 参数 | EMotorS 定义：<br>typedef struct tagEMotorS {<br>uint8_t index; //电机编号。0：Stepper1。 1：Stepper2<br>uint8_t isEnabled; //电机控制使能。0：未使能。1：使能<br>uint32_t speed; //电机控制速度（脉冲个数每秒）<br>uint32_t distance; //电机移动距离(脉冲个数)<br>}EMotorS;<br>eMotorS：EMotorS 指针<br>isQueued：是否将该指令指定为队列命令<br>queuedCmdIndex：若选择将指令加入队列，则表示指令在队列的索引号。否则，该参数无意义 |
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_BufferFull：指令队列已满<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

### 1.17.11 使能光电传感器

表 1.90 使能光电传感器

| | |
|---|---|
| 原型 | int SetInfaredSensor(bool enable,InfraredPort infraredPort) |
| 功能 | 设置光电传感器。 |
| 参数 | InfraredPort 定义：<br>enum InfraredPort{ |

| | |
|---|---|
| | IF_PORT_GP1;<br>IF_PORT_GP2;<br>IF_PORT_GP4;<br>IF_PORT_GP5;<br>};<br>enable：使能标志。0：未使能。1：使能<br>infraredPort：光电传感器连接至机械臂的接口，请选择对应的接口 |
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

### 1.17.12 获取光电传感器读数

表 1.91 获取光电传感器读数

| | |
|---|---|
| 原型 | int GetInfraredSensor (InfraredPort infraredPort, uint8_t *value) |
| 功能 | 获取光电传感器读数 |
| 参数 | InfraredPort 定义：<br>enum InfraredPort {<br>IF_PORT_GP1;<br>IF_PORT_GP2;<br>IF_PORT_GP4;<br>IF_PORT_GP5;<br>};<br>infraredPort：光电传感器连接至机械臂的接口，请选择对应的接口<br>value：传感器数值 |
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

### 1.17.13 使能颜色传感器

表 1.92 使能颜色传感器

| | |
|---|---|
| 原型 | int SetColorSensor(bool enable,ColorPort colorPort) |
| 功能 | 使能颜色传感器 |
| 参数 | ColorPort 定义：<br>enum ColorPort {<br>IF_PORT_GP1;<br>IF_PORT_GP2; |

| | |
|---|---|
| | IF_PORT_GP4;<br>IF_PORT_GP5;<br>};<br>enable：使能标志。0：未使能。1：使能<br>colorPort：颜色传感器连接至机械臂的接口，请选择对应的接口 |
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

### 1.17.14 获取颜色传感器读数

表 1.93 获取颜色传感器读数

| | |
|---|---|
| 原型 | int GetColorSensor( uint8_t *r, uint8_t *g, uint8_t *b ) |
| 功能 | 获取颜色传感器读数 |
| 参数 | r：红色，取值范围：0-255<br>g：绿色，取值范围：0-255<br>b：蓝色，取值范围：0-255 |
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

## 1.18 CAL 功能

### 1.18.1 设置角度传感器静态偏差

由于角度传感器焊接、机器状态等原因，大小臂上的角度传感器可能存在一个静态偏差。我们可以通过各种手段（如调平、与标准源比较），得到此静态偏差，并通过此 API 写入到设备中。

大/小臂角度=大/小臂角度传感器静态偏差值+大/小臂角度传感器读数*传感器线性化参数

基座角度=基座编码器静态偏差值+基座编码器读数

表 1.94 设置角度传感器静态偏差

| | |
|---|---|
| 原型 | int SetAngleSensorStaticError(float rearArmAngleError, float frontArmAngleError) |
| 功能 | 设置大小臂角度传感器静态偏差 |
| 参数 | rearArmAngleError：大臂角度传感器静态偏差<br>frontArmAngleError：小臂角度传感器静态偏差 |
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

### 1.18.2 读取角度传感器静态偏差

表 1.95 读取角度传感器静态偏差

| | |
|---|---|
| 原型 | int GetAngleSensorStaticError(float *rearArmAngleError, float *frontArmAngleError) |
| 功能 | 读取大/小臂角度传感器静态偏差 |
| 参数 | rearArmAngleError：大臂角度传感器静态偏差<br>frontArmAngleError：小臂角度传感器静态偏差 |
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

### 1.18.3 设置角度传感器线性化参数

表 1.96 设置角度传感器线性化参数

| | |
|---|---|
| 原型 | int SetAngleSensorCoef(float rearArmAngleCoef, float frontArmAngleCoef) |
| 功能 | 设置大/小臂角度传感器线性化参数 |
| 参数 | rearArmAngleCoef：大臂角度传感器线性化参数<br>frontArmAngleCoef：小臂角度传感器线性化参数 |
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

### 1.18.4 读取角度传感器线性化参数

表 1.97 读取角度传感器线性化参数

| | |
|---|---|
| 原型 | int GetAngleSensorCoef(float *rearArmAngleCoef, float *frontArmAngleCoef) |
| 功能 | 读取大/小臂角度传感器线性化参数 |
| 参数 | rearArmAngleCoef：大臂角度传感器线性化参数<br>frontArmAngleCoef：小臂角度传感器线性化参数 |
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

### 1.18.5 设置基座编码器静态偏差

表 1.98 设置基座编码器静态偏差

| | |
|---|---|
| 原型 | int SetBaseDecoderStaticError (float baseDecoderError) |

| | |
|---|---|
| 功能 | 设置基座编码器静态偏差 |
| 参数 | baseDecoderError：基座编码器静态偏差 |
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

### 1.18.6 读取基座编码器静态偏差

表 1.99 读取基座编码器静态偏差

| | |
|---|---|
| 原型 | int GetBaseDecoderStaticError(float *baseDecoderError) |
| 功能 | 读取基座编码器静态偏差 |
| 参数 | baseDecoderError：基座编码器静态偏差 |
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

## 1.19 WIFI 功能

DobotStudio可通过WIFI控制Dobot Maigican。Dobot Maigican连接WIFI模块后，需设置无线网络相关参数（IP地址、子网掩码、默认网关，使能WIFI等），使Dobot Magician接入无线局域网。接入成功后Dobot Magician无需通过USB即可连接DobotStudio。

### 1.19.1 使能 WIFI

表 1.100 设置 WIFI 配置模式接口说明

| | |
|---|---|
| 原型 | int SetWIFIConfigMode(bool enable) |
| 功能 | 使能 WIFI |
| 参数 | Enable：0，未使能。1，使能 |
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

### 1.19.2 获取 WIFI 状态

表 1.101 获取 WIFI 状态

| | |
|---|---|
| 原型 | int GetWIFIConfigMode(bool *isEnabled) |
| 功能 | 获取当前 WIFI 状态 |
| 参数 | isEnabled：0，未使能。1，使能 |
| 返回 | DobotCommunicate_NoError：指令正常返回 |

| | |
|---|---|
| | DobotCommunicate_Timeout：指令无返回，导致超时 |

### 1.19.3 设置 SSID

SSID（Service Set Identifier）：无线网络名称。

表 1.102 设置网络 SSID 接口说明

| | |
|---|---|
| 原型 | int SetWIFISSID(const char *ssid) |
| 功能 | 设置网络 SSID |
| 参数 | ssid：字符串指针 |
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

### 1.19.4 获取设置的 SSID

表 1.103 获取当前设置 SSID

| | |
|---|---|
| 原型 | int GetWIFISSID(char *ssid, uint32_t maxLen) |
| 功能 | 获取当前设置的 SSID |
| 参数 | ssid：字符串指针<br>maxLen：字符串最大长度，以避免溢出 |
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

### 1.19.5 设置 WIFI 密码

表 1.104 设置网络密码接口说明

| | |
|---|---|
| 原型 | int SetWIFIPassword(const char *password) |
| 功能 | 设置 WIFI 密码 |
| 参数 | password：字符串指针 |
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

### 1.19.6 获取 WIFI 密码

表 1.105 获取当前设置网络密码

| | |
|---|---|
| 原型 | int GetWIFIPassword(char *password, uint32_t maxLen) |
| 功能 | 获取 WIFI 密码 |

| | |
|---|---|
| 参数 | password：字符串指针<br>maxLen：字符串最大长度，以避免溢出 |
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

### 1.19.7 设置 IP 地址

表 1.106 设置 IP 地址

| | |
|---|---|
| 原型 | int SetWIFIIPAddress(WIFIIPAddress *wifiIPAddress) |
| 功能 | 设置 IP 地址 |
| 参数 | typedef struct tagWIFIIPAddress {<br>uint8_t dhcp; //是否开启 DHCP。0：关闭。1：开启<br>uint8_t addr[4]; //IP 地址分成四段，每段取值范围：0~255<br>}WIFIIPAddress;<br>wifiIPAddr：WIFIIPAddress 指针 |
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

### 1.19.8 获取设置的 IP 地址

表 1.107 获取当前设置 IP 地址接口说明

| | |
|---|---|
| 原型 | int GetWIFIIPAddress(WIFIIPAddress *wifiIPAddress) |
| 功能 | 获取设置的 IP 地址 |
| 参数 | typedef struct tagWIFIIPAddress {<br>uint8_t dhcp; //是否开启 DHCP。0：关闭。1：开启<br>uint8_t addr[4]; //IP 地址分成四段，每段取值范围：0~255<br>}WIFIIPAddress; |
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

### 1.19.9 设置子网掩码

表 1.108 设置子网掩码

| | |
|---|---|
| 原型 | int SetWIFINetmask(WIFINetmask *wifiNetmask) |
| 功能 | 设置子网掩码 |

| | |
|---|---|
| 参数 | typedef struct tagWIFINetmask {<br>uint8_t addr[4]; //IP 地址分成四段，每段取值范围：0~255<br>}WIFINetmask;<br>wifiNetmask：WIFINetmask 指针 |
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

### 1.19.10 获取设置的子网掩码

表 1.109 获取当前设置子网掩码接口说明

| | |
|---|---|
| 原型 | int GetWIFINetmask(WIFINetmask *wifiNetmask) |
| 功能 | 获取设置的子网掩码 |
| 参数 | typedef struct tagWIFINetmask {<br>uint8_t addr[4]; //IP 地址分成四段，每段取值范围：0~255<br>}WIFINetmask;<br>wifiNetmask：WIFINetmask 指针 |
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

### 1.19.11 设置网关

表 1.110 设置网关

| | |
|---|---|
| 原型 | int SetWIFIGateway(WIFIGateway *wifiGateway) |
| 功能 | 设置网关 |
| 参数 | typedef struct tagWIFIGateway {<br>uint8_t addr[4]; //IP地址分成四段，每段取值范：0~255<br>}WIFIGateway;<br>wifiGateway：WIFIGateway指针 |
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

### 1.19.12 获取设置的网关

表 1.111 获取当前设置子网掩码接口说明

| | |
|---|---|
| 原型 | int GetWIFIGateway(WIFIGateway *wifiGateway) |
| 功能 | 获取设置的网关 |

| | |
|---|---|
| 参数 | typedef struct tagWIFIGateway {<br>uint8_t addr[4]; //IP地址分成四段，每段取值范：0~255<br>}WIFIGateway;<br>wifiGateway：WIFIGateway 指针 |
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

### 1.19.13 设置 DNS

表 1.112 设置 DNS

| | |
|---|---|
| 原型 | int SetWIFIDNS(WIFIDNS *wifiDNS) |
| 功能 | 设置 DNS |
| 参数 | typedef struct tagWIFIDNS {<br>uint8_t addr[4]; //IP地址分成四段，每段取值范：0~255<br>}WIFIDNS;<br>wifiDNS：WIFIDNS 指针 |
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

### 1.19.14 获取设置的 DNS

表 1.113 获取设置的 DNS

| | |
|---|---|
| 原型 | int GetWIFIDNS(WIFIDNS *wifiDNS) |
| 功能 | 获取设置的 DNS |
| 参数 | typedef struct tagWIFIDNS {<br>uint8_t addr[4]; //IP 分成四段，每段取值范围 0~255<br>}WIFIDNS;<br>wifiDNS：DNS 结构体指针 |
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

### 1.19.15 获取当前 WIFI 模块的连接状态

表 1.114 获取当前 WIFI 模块连接状态

| | |
|---|---|
| 原型 | int GetWIFIConnectStatus(bool *isConnected) |
| 功能 | 获取当前 WIFI 模块连接状态 |

| 参数 | isConnected：0，未连接。1，连接 |
| --- | --- |
| 返回 | DobotCommunicate_NoError：指令正常返回<br>DobotCommunicate_Timeout：指令无返回，导致超时 |

## 1.20 其他功能

### 1.20.1 事件循环功能

在某些语言中，当调用 API 接口后，如果没有事件循环，应用程序将直接退出，导致指令没有下发至 Dobot 控制器。为避免这种情况发生，我们提供了事件循环接口，在应用程序退出前调用（目前已知需要做此处理的语言有 Python）。

表 1.115 事件循环功能接口说明

| 原型 | void DobotExec(void) |
| --- | --- |
| 功能 | 事件循环功能 |
| 参数 | 无 |
| 返回 | 无 |