C#函数使用手册

版本：V2.1.4

时间：2022.06.14

上海节卡机器人科技有限公司　Shanghai JAKA Robotics Ltd
电话 Tel ：+400 006 2665 | 网站 Web:www.jaka.com
上海：上海市闵行区剑川路 610 号 33-35 懂 | Building 33-35,No.610 Jianchuan Rd, Minhang District, Shanghai
常州：江苏省常州市武进国家高新区武宜南路 377 号 10 号楼 | Building 10, No.377 South Wuyi Rd, Changzhou, Jiangsu

产权说明

版本记录

版本编号	版本日期	支持版本	说明
V1.0.0	2020.05.27	V1.4.10/V2.0.10 及以上	创建
V1.0.1	2020.07.17	V1.4.10/V2.0.10 及以上	增加目录 3.20、3.21、4.63--4.66 部分
V1.0.2	2020.09.27	V1.4.10/V2.0.10 及以上	增加目录 3.19、3.20、4.67-4.74 部分
V1.0.3	2020.10.22	V1.4.10/V2.0.10 及以上	增加目录 3.21、4.75--4.77 部分
V1.0.4	2020.11.25	V1.4.12/V2.0.12 及以上	修复阻塞运动卡顿问题，增加目录 4.78-4.82 部分
V1.0.5	2020.12.08	V1.4.12/V2.0.12 及以上	增加目录 3.17-3.19、4.14、4.16、4.84
V1.0.6	2020.01.18	V1.4.24/V2.0.24 及以上	修改目录 3.20、增加目录 4.35、4.39、4.84-4.98
V1.0.7	2021.04.28	V1.4.31/V1.5.12.17/V2.0.31 及以上	增加目录 3.25-3.29、4.86-4.87、4.101-4.108
V1.0.8	2021.08.30	V1.4.31/V1.5.12.17/V2.0.31 及以上	增加 API 使用说明
V2.0.1	2021.12.10	V1.4.24/V1.5.12.17/V2.0.24 及以上	增加 ftp 接口
V2.1.2	2022.07.01	V1.4.24/V1.5.12.17/V2.0.24 及以上	增加部分接口， 文档结构修改
V2.1.2	2022.11.29	V1.4.24/V1.5.12.17/V2.0.24 及以上	增加圆弧运动圈数指定
V2.1.4	2023.06.14	V1.5.13.08/V2.0.24 及以上	扩展力矩传感器监测数据类型 添加 FTP 目录查询示例 修正设置柔顺控制参数接口说明

目录

1.简介

jakaAPI 是基于网络通信协议 TCP/IP 实现的，提供了用于操作机械臂的接口，提供 python，C/C++，C#四种语言的支持。

本文是基于 C#开发的，其中类的方法是操作机械臂的接口。

2.文档须知

- 接口中的长度单位统一为毫米（mm），角度单位统一为弧度（rad）。
- 版本号查询方法：windows 中右击 dll 文件，选择属性，在“详细信息”选项卡中可以看到版本信息。linux 中输入命令 strings libjakaAPI.so | grep jakaAPI_version 查询版本信息。
- 节卡 SDK 使用的编码方式为 UTF-8 编码。
- Servo 模式下不能使用 joint_move、linear_move 等指令。

3.数据结构

3.1 回调函数类型

用户注册机械臂发生异常时的回调函数类型

```
/**
* @brief 机械臂回调函数指针
*/
delegate void CallBackFuncType(int error_code);
```

3.2 接口调用返回值列表

```
#define ERR_SUCC                    0               //调用成功
#define ERR_FUCTION_CALL_ERROR      2               //异常调用，调用接口异常，控制器不支持
#define ERR_INVALID_HANDLER         -1              //无效的控制句柄
```

```
#define ERR_INVALID_PARAMETER     -2              //无效的参数
#define ERR_COMMUNICATION_ERR     -3              //通信连接错误
#define ERR_KINE_INVERSE_ERR      -4              //逆解失败
#define ERR_EMERGENCY_PRESSED     -5              //急停开关被按下
#define ERR_NOT_POWERED           -6              //机械臂未上电
#define ERR_NOT_ENABLED           -7              //机械臂未使能
#define ERR_DISABLE_SERVOMODE     -8              //机械臂没有进入 servo 模式
#define ERR_NOT_OFF_ENABLE        -9              //机械臂没有关闭使能
#define ERR_PROGRAM_IS_RUNNING    -10             //程序正在运行，不允许操作
#define ERR_CANNOT_OPEN_FILE      -11             //无法打开文件，文件不存在
#define ERR_FTP_PREFROM             -14             //ftp 异常
```

3.3 笛卡尔空间位置数据类型

```
/**
* @brief 笛卡尔空间位置数据类型
*/
public struct CartesianTran
{

    public double x;        ///< x 轴坐标，单位 mm
    public double y;        ///< y 轴坐标，单位 mm
    public double z;        ///< z 轴坐标，单位 mm
};
```

3.4 欧拉角姿态数据类型

```
/**
* @brief 欧拉角姿态数据类型
*/
public struct Rpy
{
    public double rx;  ///< 绕固定轴 X 旋转角度，单位：rad
    public double ry;  ///< 绕固定轴 Y 旋转角度，单位：rad
    public double rz;  ///< 绕固定轴 Z 旋转角度，单位：rad
};
```

3.5 四元数姿态数据类型

```
/**
* @brief 四元数姿态数据类型
*/
```

```
public struct Quaternion
{
    public double s;
    public double x;
    public double y;
    public double z;
};
```

3.6 笛卡尔空间位姿类型

```
/**
*@brief 笛卡尔空间位姿类型
*/
public struct CartesianPose
{
    public CartesianTran tran;      ///< 笛卡尔空间位置
    public Rpy rpy;              ///< 笛卡尔空间姿态
};
```

3.7 旋转矩阵数据类型

```
/**
* @brief 旋转矩阵数据类型
*/
public struct RotMatrix
{
    public CartesianTran x;  ///< x 轴列分量
    public CartesianTran y;  ///< y 轴列分量
    public CartesianTran z; ///< z 轴列分量
};
```

3.8 程序运行状态枚举类型

```
/**
* @brief 程序运行状态枚举类型
*/
public enum ProgramState
{
    PROGRAM_IDLE,       ///< 机械臂停止运行
    PROGRAM_RUNNING,    ///< 机械臂正在运行
    PROGRAM_PAUSED      ///< 机械臂暂停
```

```
};
```

3.9 坐标系选择枚举类型

```
/**
* @brief 坐标系选择枚举类型
*/
public enum CoordType
{
    COORD_BASE,     ///< 基坐标系
    COORD_JOINT,    ///< 关节空间
    COORD_TOOL      ///< 工具坐标系
};
```

3.10 运动模式枚举类型

```
/**
* @brief jog 运动模式枚举
*/
public enum MoveMode
{
    ABS = 0,                    ///< 绝对运动
    INCR,                       ///< 增量运动
    CONTINUE                    ///< 连续运动或延工具坐标运动
};
```

3.11 系统检测数据类型

```
/**
* @brief 系统监测数据类型
*/
public struct SystemMonitorData
{
    public int scbMajorVersion;             ///<scb 主版本号
    public int scbMinorVersion;             ///<scb 次版本号
    public int cabTemperature;              ///<控制柜温度
    public double robotAveragePower;        ///<控制柜总线平均功率
    public double robotAverageCurrent;      ///<控制柜总线平均电流
    [MarshalAs(UnmanagedType.ByValArray, SizeConst = 6)]
    public double[] instCurrent;            ///<机械臂 6 个轴的瞬时电流
    [MarshalAs(UnmanagedType.ByValArray, SizeConst = 6)]
    public double[] instVoltage;            ///<机械臂 6 个轴的瞬时电压
```

```
    [MarshalAs(UnmanagedType.ByValArray, SizeConst = 6)]
    public double[] instTemperature;        ///<机械臂 6 个轴的瞬时温度
};
```

3.12 负载数据类型

```
/**
* @brief 负载数据类型
*/
public struct PayLoad
{
    public double mass;                ///<负载质量，单位：kg
    public CartesianTran centroid;      ///<负载质心，单位：mm
};
```

3.13 关节位置数据类型

```
/**
* @brief 关节位置数据类型
*/
public struct JointValue
{
    [MarshalAs(UnmanagedType.ByValArray, SizeConst = 6)]
    public double[] jVal ;        ///< 6 关节位置值，单位：rad
};
```

3.14 IO 类型

```
/**
* @brief IO 类型枚举
*/
public enum IOType
{
    IO_CABINET,     ///< 控制柜面板 IO
    IO_TOOL,        ///< 工具 IO
    IO_EXTEND       ///< 扩展 IO
};
```

3.15 机械臂状态数据类型

```
/**
```

```
 * @brief 机械臂状态数据
 */
public struct RobotState
{
    public int estoped;       ///< 是否急停
    public int poweredOn;     ///< 是否打开电源
    public int servoEnabled;    ///< 是否处于伺服模式
};
```

3.16 机械臂力矩数据类型

```
/**
 * @brief 机械臂力矩数据
 */
public struct TorqueValue
{
    [MarshalAs(UnmanagedType.ByValArray, SizeConst = 6)]
    public double[] jTorque;    ///< 是否使能
}
```

3.17 机械臂关节监测数据类型

```
/**
 * @brief 机器人关节监测数据类型
 */
[StructLayout(LayoutKind.Sequential)]
public struct JointMonitorData
{
    public double instCurrent;          ///< 瞬时电流
    public double instVoltage;          ///< 瞬时电压
    public double instTemperature;      ///< 瞬时温度
    public double instVel;              ///< 瞬时速度
    public double instTorq;             ///< 瞬时力矩
}
```

3.18 机械臂监测数据类型

```
/**
 * @brief 机械臂监测数据类型
 */
```

```
4.  public struct RobotMonitorData
5.  {
6.      double scbMajorVersion;                         ///< scb 主版本号
7.      double scbMinorVersion;                         ///< scb 小版本号
8.      double cabTemperature;                          ///< 控制器温度
9.      double robotAveragePower;                       ///< 机械臂平均电压
10.     double robotAverageCurrent;                     ///< 机械臂平均电流
11.     [MarshalAs(UnmanagedType.ByValArray, SizeConst = 6)]
12.     public JointMonitorData[] jointMonitorData;     ///< 机械臂 6 个关节的监测数据
13. }
```

3.19 力矩传感器监测数据类型

```
1.  /**
2.  * @brief 力矩传感器监测数据类型
3.  */
4.  [StructLayout(LayoutKind.Sequential)]
5.  public struct TorqSensorMonitorData
6.  {
7.      [MarshalAs(UnmanagedType.ByValArray, SizeConst = 20)]
8.      public char[] ip;                               ///< 力矩传感器 ip 地址
9.      int port;                                       ///< 力矩传感器端口号
10.     PayLoad payLoad;                                ///< 工具负载
11.     int status;                                     ///< 力矩传感器状态
12.     int errcode;                                    ///< 力矩传感器异常错误码
13.     [MarshalAs(UnmanagedType.ByValArray, SizeConst = 6)]
14.     public double[] actTorque;                      ///< 力矩传感器实际接触力值
15.     [MarshalAs(UnmanagedType.ByValArray, SizeConst = 6)]
16.     public double[] torque;                         ///< 力矩传感器原始读数值
17.     [MarshalAs(UnmanagedType.ByValArray, SizeConst = 6)]
18.     public double[] realTorque;                     ///< 力矩传感器实际接触力值（不随初始
    化选项变化）
19. }
```

3.20 机械臂状态监测数据类型

```
1.  /**
2.  * @brief 机器人状态监测数据,使用 get_robot_status 函数更新机器人状态数据
3.  */
4.
5.  [StructLayout(LayoutKind.Sequential)]
```

```
6.  public struct RobotStatus
7.  {
8.      public int errcode;                                    ///< 机器人运行出错时错误编号,0 为运行正常,其它为运行异常
9.      public int inpos;                                      ///< 机器人运动是否到位标志,0 为没有到位,1 为运动到位
10.     public int powered_on;                                 ///< 机器人是否上电标志,0 为没有上电,1 为上电
11.     public int enabled;                                    ///< 机器人是否使能标志,0 为没有使能,1 为使能
12.     public double rapidrate;                               ///< 机器人运动倍率
13.     public int protective_stop;                            ///< 机器人是否检测到碰撞,0 为没有检测到碰撞,1 为检测到碰撞
14.     public int emergency_stop;                             ///< 机器人是否急停,0 为没有急停,1 为急停
15.     [MarshalAs(UnmanagedType.ByValArray, SizeConst = 256)]
16.     public int []dout;                                     ///< 机器人控制柜数字输出信号,dout[0]为信号的个数
17.     [MarshalAs(UnmanagedType.ByValArray, SizeConst = 256)]
18.     public int []din;                                      ///< 机器人控制柜数字输入信号,din[0]为信号的个数
19.     [MarshalAs(UnmanagedType.ByValArray, SizeConst = 256)]
20.     public double []ain;                                   ///< 机器人控制柜模拟输入信号,ain[0]为信号的个数
21.     [MarshalAs(UnmanagedType.ByValArray, SizeConst = 256)]
22.     public double []aout;                                  ///< 机器人控制柜模拟输出信号,aout[0]为信号的个数
23.     [MarshalAs(UnmanagedType.ByValArray, SizeConst = 16)]
24.     public int[] tio_dout;                                 ///< 机器人末端工具数字输出信号,tio_dout[0]为信号的个数
25.     [MarshalAs(UnmanagedType.ByValArray, SizeConst = 16)]
26.     public int[] tio_din;                                  ///< 机器人末端工具数字输入信号,tio_din[0]为信号的个数
27.     [MarshalAs(UnmanagedType.ByValArray, SizeConst = 16)]
28.     public double[] tio_ain;                               ///< 机器人末端工具模拟输入信号,tio_ain[0]为信号的个数
29.     [MarshalAs(UnmanagedType.ByValArray, SizeConst = 3)]
30.     public int[] tio_key;                                  ///< 机器人末端工具按钮 [0]free;[1]point;[2]pause_resume;
31.     public Io_group extio;                                 ///< 机器人外部应用 IO
32.     public Io_group modbus_slave;                          ///< 机器人 Modbus 从站
33.     public Io_group profinet_slave;                        ///< 机器人 Profinet 从站
```

```
    public Io_group eip_slave;                                  ///< 机器人 Ethernet/IP 从站
    public int current_tool_id;                                 ///< 机器人目前使用的工具坐标系 id
    [MarshalAs(UnmanagedType.ByValArray, SizeConst = 6)]
    public double []cartesiantran_position;                     ///< 机器人末端所在的笛卡尔空间位置
    [MarshalAs(UnmanagedType.ByValArray, SizeConst = 6)]
    public double []joint_position;                             ///< 机器人关节空间位置
    public int on_soft_limit;                                   ///< 机器人是否处于限位,0 为没有触发限位保护,1 为触发限位保护
    public int current_user_id;                                 ///< 机器人目前使用的用户坐标系 id
    public int drag_status;                                     ///< 机器人是否处于拖拽状态,0 为没有处于拖拽状态,1 为处于拖拽状态
    public RobotMonitorData robot_monitor_data;                 ///< 机器人状态监测数据
    public TorqSensorMonitorData torq_sensor_monitor_data;      ///< 机器人力矩传感器状态监测数据
    public int is_socket_connect;                               ///< sdk 与控制器连接通道是否正常,0 为连接通道异常,1 为连接通道正常
}
```

3.21 机械臂错误码数据类型

```
/**
* @brief 机械臂错误码数据类型
*/
public struct ErrorCode
{
    public long code;                                   ///< 错误码编号
    [MarshalAs(UnmanagedType.ByValArray, SizeConst = 120)]
    public char[] message;                              ///< 错误码对应提示信息
}
```

3.22 轨迹复现配置参数存储数据类型

```
/**
* @brief 轨迹复现配置参数数据类型
*/
public struct TrajTrackPara
{
```

```
    public double xyz_interval;                         ///< 空间位置采集精度
    public double rpy_interval;                         ///< 姿态采集精度
    public double vel;                                  ///< 执行脚本运行速度
    public double acc;                                  ///< 执行脚本运行加速度
}
```

3.23 多个字符串存储数据类型

```
/**
* @brief 多个字符串存储数据类型
*/
public struct MultStrStorType
{
    public int len;                                     ///< 字符串个数
    [MarshalAs(UnmanagedType.ByValArray, SizeConst = 128*128)]
    public char[] message;                              ///< 错误码对应提示信息
}
```

3.24 运动参数可选项

```
/**
* @brief 可选参数项
*/
typedef struct
{
    int executingLineId;                                ///< 控制命令 id 编号
}OptionalCond;
```

3.25 网络异常机械臂自动终止类型枚举

```
/**
* @brief 网络异常机械臂运动自动终止类型枚举
*/
public enum ProcessType
{
    MOT_KEEP,  ///< 网络异常时机械臂继续保持原来的运动
    MOT_PAUSE, ///< 网络异常时机械臂暂停运动
```

```
    MOT_ABORT  ///< 网络异常时机械臂终止运动
}
```

3.26 柔顺控制参数类型

```
/**
* @brief 柔顺控制参数类型
*/
[StructLayout(LayoutKind.Sequential)]
public struct AdmitCtrlType
{
    int opt;              ///< 柔顺方向，可选值为 1 2 3 4 5 6 分别对应 fx fy fz mx my mz,0 代表没有勾选
    double ft_user;       ///< 用户用多大的力才能让机械臂的沿着某个方向以最大速度进行运动
    double ft_rebound;    ///< 回弹力:机械臂回到初始状态的能力
    double ft_constant;   ///< 恒力
    int ft_normal_track; ///< 法向跟踪是否开启，0 为没有开启，1 为开启
}
```

3.27 机械臂柔顺控制参数类型

```
/**
* @brief 机械臂柔顺控制参数类型
*/
[StructLayout(LayoutKind.Sequential)]
public struct RobotAdmitCtrl
{
    [MarshalAs(UnmanagedType.ByValArray, SizeConst = 6)]
    public AdmitCtrlType[] admit_ctrl;
}
```

3.28 速度柔顺控制等级和比率等级设置

```
/**
* @brief 速度柔顺控制等级和比率等级设置
* 速度柔顺控制分三个等级，并且  1>rate1>rate2>rate3>rate4>0
```

```
    * 等级为 1 时，只能设置 rate1,rate2 两个等级。rate3,rate4 的值为 0
    * 等级为 2 时，只能设置 rate1,rate2，rate3 三个等级。rate4 的值为 0
    * 等级为 3 时，能设置 rate1,rate2，rate3,rate4 4 个等级
    */
    [StructLayout(LayoutKind.Sequential)]
    public struct VelCom
    {
     int vc_level;                                    //速度柔顺控制等级
     double rate1;                                    //比率 1 等级
     double rate2;                                    //比率 2 等级
     double rate3;                                    //比率 3 等级
     double rate4;                                    //比率 4 等级
    }
```

3.29 力矩传感器的受力分量和力矩分量

```
    /**
    * @brief 力传感器的受力分量和力矩分量
    */
    [StructLayout(LayoutKind.Sequential)]
    public struct FTxyz
    {
     double fx;                                     // 沿 x 轴受力分量
     double fy;                                     // 沿 y 轴受力分量
     double fz;                                     // 沿 z 轴受力分量
     double tx;                                     // 绕 x 轴力矩分量
     double ty;                                     // 绕 y 轴力矩分量
     double tz;                                     // 绕 z 轴力矩分量
    }
```

3.30 DH 参数

```
/**
*  @brief DH 参数
*/
[StructLayout(LayoutKind.Sequential)]
public struct DHParam
{
    [MarshalAs(UnmanagedType.ByValArray, SizeConst = 6)]
    double[] alpha;
    [MarshalAs(UnmanagedType.ByValArray, SizeConst = 6)]
    double[] a;
    [MarshalAs(UnmanagedType.ByValArray, SizeConst = 6)]
```

```
    double[] d;
    [MarshalAs(UnmanagedType.ByValArray, SizeConst = 6)]
    double[] joint_homeoff;
}
```

3.31 rs485 信号量参数

```
[StructLayout(LayoutKind.Sequential)]
public struct SignInfo{
    [MarshalAs(UnmanagedType.ByValArray, SizeConst = 20)]
    char[] sig_name;//标识名
    int chn_id;     //RS485 通道 ID
    int sig_type;   //信号量类型
    int sig_addr;   //寄存器地址
    int value;      //值   设置时无效
    int frequency;  //信号量在控制器内部刷新频率不大于 10
}
```

3.32 rs485RTU 配置参数

```
[StructLayout(LayoutKind.Sequential)]
public struct ModRtuComm
{
    int chn_id;     //RS485 通道 ID   查询时 chn_id 作为输入参数
    int slaveId;    //当通道模式设置为 Modbus RTU 时，需额外指定 Modbus 从站节点 ID，其余模式可忽略
    int baudrate;   //波特率 4800,9600,14400,19200,38400,57600,115200,230400
    int databit;    //数据位 7，8
    int stopbit;    //停止位 1，2
    int parity;     //校验位 78-> 无校验 79->奇校验 69->偶校验
}
```

4.API

4.1 机械臂基础

4.1.1 创建机械臂控制句柄

```
/**
* @brief 创建机械臂控制句柄
* @param ip  控制器 ip 地址
* @param handle  机械臂控制句柄
* @return ERR_SUCC 成功 其他失败
*/
int create_handler(char[] ip,ref int handle);
```

4.1.2 销毁机械臂控制句柄

```
/**
* @brief 销毁机械臂控制句柄
* @param handle  机械臂控制句柄
* @return ERR_SUCC 成功 其他失败
*/
int destory_handler(ref int handle);
```

4.1.3 机械臂上电

```
/**
* @brief 打开机械臂电源
* @param handle  机械臂控制句柄
* @return ERR_SUCC 成功 其他失败
*/
int power_on(ref int handle);
```

4.1.4 机械臂下电

```
/**
* @brief 关闭机械臂电源
* @param handle  机械臂控制句柄
* @return ERR_SUCC 成功 其他失败
*/
int power_off(ref int handle);
```

4.1.5 机械臂关机

```
/**
* @brief 机械臂控制柜关机
* @param handle  机械臂控制句柄
* @return ERR_SUCC 成功 其他失败
*/
int  shut_down(ref int handle);
```

4.1.6 机械臂上使能

```
/**
* @brief 控制机械臂上使能
* @param handle  机械臂控制句柄
* @return ERR_SUCC 成功 其他失败
*/
int enable_robot(ref int handle);
```

4.1.7 机械臂下使能

```
/**
* @brief 控制机械臂下使能
* @param handle  机械臂控制句柄
* @return ERR_SUCC 成功 其他失败
*/
int disable_robot(ref int handle);
```

4.1.8 设置 SDK 是否开启调试模式

```
/**
* @brief 设置是否开启调试模式,选择 TRUE 时,开始调试模式,此时会在标准输出流中输出调试信息,选择 FALSE 时, 不输出调试信息
* @return ERR_SUCC 成功 其他失败
*/
int  set_debug_mode(ref int handle, bool mode);
```

代码示例：

```
/// <summary>
/// 开启 sdkdebug 模式
/// </summary>
static void example_set_debug_mode()
{
    int handle = 0;
    int ret = 0;
```

```
    //实例化机械臂，将 ip 切换为自己的 ip
    int result = jakaAPI.create_handler("192.168.2.160".ToCharArray(), ref handle);
    //开启调试模式
    jakaAPI.set_debug_mode(ref handle, true);
    Console.WriteLine("debug :true");
    //机械臂上电
    jakaAPI.power_on(ref handle);
    //机械臂上使能
    jakaAPI.enable_robot(ref handle);
    jakaAPI.destory_handler(ref handle);
}
```

4.1.9 设置 SDK 日志路径

```
/**
* @brief 设置 SDK 日志路径
* @param filepath SDK 日志路径
* @return ERR_SUCC 成功 其他失败
*/
int  set_SDK_filepath(ref int handle, ref char[] filepath);
```

4.1.10 获取 SDK 版本号

```
/**
* @brief 获取机械臂控制器版本号
* @param handle  机械臂控制句柄
* @param version SDK 版本号
* @return ERR_SUCC 成功 其他失败
*/
int get_sdk_version(ref int handle, StringBuilder version);
```

4.1.11 获取控制器 IP

```
/**
* @brief 获取控制器 IP
* @param controller_name 控制器名字
* @param ip_list 控制器 ip 列表，控制器名字为具体值时返回该名字所对应的控制器 IP 地址，控制器名字为空时，返回网段类内的所有控制器 IP 地址
* @return ERR_SUCC 成功 其他失败
*/
int  get_controller_ip(char[] controller_name, StringBuilder ip_list);
```

代码示例：

```
/// <summary>
```

```
/// 获取局域网内所有机械臂的 ip 及名称
/// </summary>
static void example_get_controller_ip()
{
    int handle = 0;
    int ret;
    //实例化机械臂，将 ip 切换为自己的 ip
    // int result = jakaAPI.create_handler("192.168.2.160".ToCharArray(), ref handle);
    StringBuilder ip_list = new StringBuilder("",3000);
    char[] controller_name1 = new char[10];
    controller_name1[0] = '\0';
    Console.WriteLine("start to get ip");
    //获取控制器 ip
    ret = jakaAPI.get_controller_ip( controller_name1, ip_list);
    Console.WriteLine("ip list get :{0}", ret);
    Console.WriteLine(" ip_list is :\n {0}", ip_list);
    jakaAPI.destory_handler(ref handle);
}
```

4.1.12 控制机械臂进入或退出拖拽模式

```
/**
* @brief 控制机械臂进入或退出拖拽模式
* @param handle  机械臂控制句柄
* @param enable  TRUE 为进入拖拽模式，FALSE 为退出拖拽模式
* @return ERR_SUCC 成功 其他失败
*/
int drag_mode_enable(ref int handle, bool enable);
```

代码示例：

```
/// <summary>
/// 进入拖拽模式，直到获得输入
/// </summary>
static void example_drag()
{
    bool in_drag = true;
    int handle = 0;
    //实例化机械臂，将 ip 切换为自己的 ip
    int result = jakaAPI.create_handler("192.168.2.160".ToCharArray(), ref handle);
    //机械臂上电
    jakaAPI.power_on( ref handle);
    //机械臂上使能
    jakaAPI.enable_robot( ref handle);
    //确认机械臂是否在拖拽模式下
```

```
    jakaAPI.is_in_drag_mode( ref handle, ref in_drag);
    Console.WriteLine("in_drag is : {0}", in_drag);
    //使能拖拽模式
    jakaAPI.drag_mode_enable( ref handle, true);
    System.Threading.Thread.Sleep(1000);
    jakaAPI.is_in_drag_mode( ref handle, ref in_drag);
    Console.WriteLine("in_drag is : {0}", in_drag);
    Console.WriteLine("input any word to disable dragmode");
    Console.Read();
    //去使能拖拽模式
    jakaAPI.drag_mode_enable(ref handle, false);
    System.Threading.Thread.Sleep(1000);
    jakaAPI.is_in_drag_mode(ref handle, ref in_drag);
    Console.WriteLine("in_drag is : {0}", in_drag);
    jakaAPI.destory_handler(ref handle);
}
```

4.1.13 查询机械臂是否处于拖拽模式

```
/**
* @brief 查询机械臂是否处于拖拽模式
* @param handle  机械臂控制句柄
* @param in_drag 查询结果 0 不在拖拽模式，1 为在拖拽模式下
* @return ERR_SUCC 成功 其他失败
*/
int is_in_drag_mode(ref int handle, ref bool in_drag);
```

4.2 机械臂运动

由控制器参与进行规划的运动

4.2.1 机械臂手动模式下运动

```
/**
* @brief 控制机械臂手动模式下运动
* @param handle  机械臂控制句柄
* @param aj_num 关节空间下代表关节号[0-5]，笛卡尔下依次为轴 x，y，z，rx，ry，rz
* @param move_mode 机械臂运动模式，增量运动，绝对运动，连续运动(即持续 jog 运动),参考 2.13 选择合适类型
* @param coord_type 机械臂运动坐标系，工具坐标系，基坐标系（当前的世界/用户坐标系）或关节空间，参考 3.12 选择合适类型
* @param vel_cmd 指令速度，旋转轴或关节运动单位为 rad/s，移动轴单位为 mm/s
* @param pos_cmd 指令位置，旋转轴或关节运动单位为 rad，移动轴单位为 mm
* @return ERR_SUCC 成功 其他失败
```

```
*/
int jog(ref int handle, int aj_num, JKTYPE.MoveMode move_mode, JKTYPE.CoordType coord_type, double vel_cmd, double pos_cmd);
```

代码示例：

```
/// <summary>
/// jog 运动 vel_cmd 超过 pi/4 会被限幅
/// </summary>
static void example_jog()
{
    int handle = 0;
    //实例化机械臂，将 ip 切换为自己的 ip
    int result = jakaAPI.create_handler("192.168.2.160".ToCharArray(), ref handle);
    jakaAPI.power_on(ref handle);//机械臂上电
    jakaAPI.enable_robot(ref handle);//机械臂上使能
    int aj_num = 0;
    //选择关节 1，运动模式：增量运动；关节坐标系；以 3.14/4 的速度沿关节 1 反方向运动 1rad。
    jakaAPI.jog(ref handle, aj_num, JKTYPE.MoveMode.INCR, JKTYPE.CoordType.COORD_JOINT, 3.14 / 4, -1);
    System.Threading.Thread.Sleep(1000);
    jakaAPI.jog_stop(ref handle, -1);//-1 代表停止所有轴的运动
    jakaAPI.destory_handler(ref handle);
}
```

4.2.2 机械臂手动模式下运动停止

```
/**
* @brief 控制机械臂手动模式下运动停止
* @param handle  机械臂控制句柄
* @param num  机械臂轴号 0-5，num 为-1 时，停止所有轴
* @return ERR_SUCC 成功 其他失败
*/
int jog_stop(ref int handle，ref int num);
```

4.2.3 机械臂关节运动

```
/**
* @brief 机械臂关节运动
* @param handle  机械臂控制句柄
* @param joint_pos 机械臂关节运动目标位置
* @param move_mode 指定运动模式：增量运动或绝对运动
* @param is_block 设置接口是否为阻塞接口，TRUE 为阻塞接口 FALSE 为非阻塞接口 阻塞时默认 30s 超时
* @param speed 机械臂关节运动速度，单位：rad/s
* @return ERR_SUCC 成功 其他失败
```

```
*/
int  joint_move(const JKHD* handle, const JointValue* joint_pos, MoveMode move_mode,BOOL is_block, double speed);
```

代码示例：

```
/// <summary>
/// joint move 关节限速 pi/2 rad/s
/// 支持阻塞和非阻塞，速度单位 rad/s
/// </summary>
static void example_joint_move()
{
    int handle = 0;
    Stopwatch sw = new Stopwatch();//计时器
    //实例化机械臂，将 ip 切换为自己的 ip
    int result = jakaAPI.create_handler("192.168.2.160".ToCharArray(), ref handle);
    jakaAPI.power_on(ref handle);
    jakaAPI.enable_robot(ref handle);
    //定义并初始化关 JointValue 变量
    JKTYPE.JointValue joint_pos = new JKTYPE.JointValue();
    joint_pos.jVal = new double[] { 1, 1, 1, 1, 1, 1 };
    Console.WriteLine("joint_move 指令执行开始");
    sw.Start();
    //阻塞运动到目标点位
    jakaAPI.joint_move(ref handle, ref joint_pos, JKTYPE.MoveMode.ABS, true, 1);
    sw.Stop();
    Console.WriteLine("joint_move 指令执行结束
{0}ms", sw.ElapsedTicks / (decimal)Stopwatch.Frequency);
    jakaAPI.destory_handler(ref handle);
}
```

4.2.4 机械臂末端直线运动

```
/**
* @brief 机械臂末端直线运动
* @param handle  机械臂控制句柄
* @param end_pos 机械臂末端运动目标位置
* @param move_mode 指定运动模式：增量运动或绝对运动
* @param is_block 设置接口是否为阻塞接口，TRUE 为阻塞接口 FALSE 为非阻塞接口 阻塞时默认 30s 超时
* @param speed 机械臂直线运动速度，单位：mm/s
* @return ERR_SUCC 成功 其他失败
*/
int linear_move(ref int handle, ref JKTYPE.CartesianPose end_pos, JKTYPE.MoveMode move_mode, bool is_block, double speed);
```

代码示例：

```
/// <summary>
/// linear move
/// 支持阻塞和非阻塞，速度单位 mm/s
/// 注意避免奇异点，常见奇异点：
/// * 工具末端位置在轴线 Z1 和 Z2 构成的平面上；
/// * 轴线 Z2,Z3,Z4 共面；
/// * 关节 5 角度为 0 或 180°，即 Z4,Z6 平行；
/// </summary>
static void example_linear_move()
{
    int handle = 0;
    Stopwatch sw = new Stopwatch();//计时器
    Console.WriteLine(sw.ElapsedTicks / (decimal)Stopwatch.Frequency);
    //实例化机械臂，将 ip 切换为自己的 ip
    int result = jakaAPI.create_handler("192.168.2.160".ToCharArray(), ref handle);
    jakaAPI.power_on(ref handle);
    jakaAPI.enable_robot(ref handle);
    Console.WriteLine("linear_move 指令执行开始");
    JKTYPE.CartesianPose tcp_pos = new JKTYPE.CartesianPose();
    tcp_pos.tran.z = -100;
    sw.Start();
    jakaAPI.linear_move(ref handle, ref tcp_pos, JKTYPE.MoveMode.INCR, true, 30);
    sw.Stop();
    Console.WriteLine("linear_move 指令执行结束
{0}s", sw.ElapsedTicks / (decimal)Stopwatch.Frequency);
    jakaAPI.destory_handler(ref handle);
}
```

4.2.5 机械臂扩展关节运动

```
/**
* @brief 机械臂关节运动
* @param joint_pos 机械臂关节运动目标位置
* @move_mode 指定运动模式：增量运动(相对运动)或绝对运动
* @param is_block 设置接口是否为阻塞接口，TRUE 为阻塞接口 FALSE 为非阻塞接口 阻塞时默认 30s 超时
* @param speed 机械臂关节运动速度，单位：rad/s
* @param acc 机械臂关节运动角加速度
* @param tol 机械臂关节运动终点误差
* @param option_cond 机械臂关节可选参数，如果不需要，该值可不赋值,填入空指针就可
* @return ERR_SUCC 成功 其他失败
*/
int  joint_move_extend(ref int handle, const JointValue* joint_pos, MoveMode move_mode, BOOL is_block, double speed, double acc, double tol, const OptionalCond* option_cond);
```

4.2.6 机械臂扩展末端直线运动

```
/**
* @brief 机械臂末端直线运动
* @param end_pos 机械臂末端运动目标位置
* @move_mode 指定运动模式：增量运动(相对运动)或绝对运动
* @param is_block 设置接口是否为阻塞接口，TRUE 为阻塞接口 FALSE 为非阻塞接口 阻塞时默认 30s 超时
* @param speed 机械臂直线运动速度，单位：mm/s
* @param acc 机械臂直线运动角加速度
* @param tol 机械臂直线运动终点误差
* @param option_cond 机械臂关节可选参数，如果不需要，该值可不赋值,填入空指针就可
* @return ERR_SUCC 成功 其他失败
*/
int  linear_move_extend(ref int handle, const CartesianPose* end_pos, MoveMode move_mode, BOOL is_block, double speed, double accel, double tol, const  OptionalCond* option_cond);
```

4.2.7 机械臂圆弧运动

```
/**
* @brief 机械臂末端圆弧运动
* @param end_pos 机械臂末端运动目标位置
* @param mid_pos 机械臂末端运中间点
* @move_mode 指定运动模式：增量运动(相对运动)或绝对运动
* @param is_block 设置接口是否为阻塞接口，TRUE 为阻塞接口 FALSE 为非阻塞接口阻塞时默认 30s 超时
* @param speed 机械臂直线运动速度，单位：mm/s
* @param acc 机械臂笛卡尔空间加速度
* @param tol 机械臂笛卡尔空间终点误差
* @param option_cond 机械臂关节可选参数，如果不需要，该值可不赋值,填入空指针就可
* @return ERR_SUCC 成功 其他失败
*/
int  circular_move(ref CartesianPose end_pos, ref CartesianPose mid_pos, int move_mode, BOOL is_block, double speed, double accel, double tol, ref  OptionalCond option_cond);
```

代码示例：

```
 /// <summary>
/// 圆弧运动
/// </summary>
static void example_circle_move()
{
    JKTYPE.OptionalCond opt = new JKTYPE.OptionalCond();
    JKTYPE.CartesianPose end_p = new JKTYPE.CartesianPose(), mid_p = new JKTYPE.CartesianPose();
    end_p.tran.x = -200; end_p.tran.y = 400; end_p.tran.z = 400;
```

```
    end_p.rpy.rx = -90 * PI / 180; end_p.rpy.ry = 0 * PI / 180; end_p.rpy.rz = 0 * PI / 180;
    mid_p.tran.x = -300; mid_p.tran.y = 400; mid_p.tran.z = 500;
    mid_p.rpy.rx = -90 * PI / 180; mid_p.rpy.ry = 0 * PI / 180; mid_p.rpy.rz = 0 * PI / 180;
    int handle = 0;
    int ret = 0;
    //实例化机械臂，将 ip 切换为自己的 ip
    int result = jakaAPI.create_handler("192.168.2.160".ToCharArray(), ref handle);
    //机械臂上电
    jakaAPI.power_on(ref handle);
    //机械臂上使能
    jakaAPI.enable_robot(ref handle);
    //定义并初始化关 JointValue 变量
    JKTYPE.JointValue joint_pos = new JKTYPE.JointValue();
    joint_pos.jVal = new double[] { 85.76 * PI / 180, -6.207 * PI / 180, 111.269 * PI / 180, 74.938 * PI / 180, 94.24 * PI / 180, 0 * PI / 180 };
    //关节空间运动，其中 ABS 代表绝对运动，TRUE 代表指令是阻塞的，1 代表速度为 1rad/s
    jakaAPI.joint_move(ref handle, ref joint_pos, JKTYPE.MoveMode.ABS, true, 1);
    //圆弧运动，其中 ABS 代表绝对运动，TRUE 代表指令是阻塞的，20 代表直线速度为 20mm/s，1 代表加速度，0.1 代表机械臂终点误差，opt 为可选参数。
    jakaAPI.circular_move(ref handle, ref end_p, ref mid_p, JKTYPE.MoveMode.ABS, true, 20, 1, 0.1, ref opt);
}
```

```
/**
* @brief 机器人末端圆弧运动
* @param end_pos 机器人末端运动目标位置
* @param mid_pos 机器人末端运中间点
* @param move_mode 指定运动模式：增量运动(相对运动)或绝对运动
* @param is_block 设置接口是否为阻塞接口，TRUE 为阻塞接口 FALSE 为非阻塞接口
* @param speed 机器人圆弧速度，单位：rad/s
* @param acc 机器人圆弧运动加速度，单位：rad/s^2
* @param tol 机器人关节运动终点误差，单位 mm
* @param option_cond 机器人关节可选参数，如果不需要，该值可不赋值,填入空指针就可
* @param circle_cnt 指定机器人圆弧运动圈数。为 0 时等价于 circular_move
* @return ERR_SUCC 成功 其他失败
*/
public static extern int circular_move_extend(ref int i, ref JKTYPE.CartesianPose end_pos, ref JKTYPE.CartesianPose mid_pos, JKTYPE.MoveMode move_mode, bool is_block, double speed, double acc, double tol, ref JKTYPE.OptionalCond option_cond, int circle_cnt);
```

代码示例：

```
using static jakaApi.jakaAPI;
using static jkType.JKTYPE;
```

```
3.
4.  namespace example
5.  {
6.      internal class Program
7.      {
8.          static void Main(string[] args)
9.          {
10.             int handle = -1;
11.             create_handler("192.168.20.139".ToCharArray(), ref handle);
12.             power_on(ref handle);
13.             enable_robot(ref handle);
14.
15.             double d2r = Math.PI / 180;
16.             JointValue start_jpos = new JointValue
17.             {
18.                 jVal = new double[] { 85.76 * d2r, -6.207 * d2r, 111.269 * d2r, 74.938 * d
    2r, 94.24 * d2r, 0 * d2r }
19.             };
20.             joint_move(ref handle, ref start_jpos, MoveMode.ABS, true, 100);
21.
22.             CartesianPose end_pos = new CartesianPose {
23.                 tran = new CartesianTran{x = -200, y = 400, z = 400},
24.                 rpy = new Rpy {rx = Math.PI * 0.5, ry = 0, rz = 0}
25.             };
26.             CartesianPose mid_pos = new CartesianPose {
27.                 tran = new CartesianTran {x = -300, y = 400, z = 500},
28.                 rpy = new Rpy {rx = Math.PI * 0.5, ry = 0, rz = 0}
29.             };
30.             OptionalCond opt = new OptionalCond{};
31.             circular_move_extend(ref handle, ref end_pos, ref mid_pos, MoveMode.ABS, true,
     100, 20, 0.1, ref opt, 2);
32.
33.             disable_robot(ref handle);
34.             power_off(ref handle);
35.         }
36.     }
37. }
```

4.2.8 机械臂设置阻塞运动超时时间

```
1.  /**
2.  * @brief 设置机器人阻塞等待超时时间
3.  * @param seconds 时间参数，单位秒 大于0.5
4.  * @return ERR_SUCC 成功 其他失败
```

```
*/
errno_t set_block_wait_timeout(ref int handle,float seconds);
```

4.2.9 机械臂运动终止

```
/**
* @brief 终止当前机械臂运动
* @return ERR_SUCC 成功 其他失败
*/
int  motion_abort(ref int handle);
```

代码示例：

```
/// <summary>
/// 运动终止
/// </summary>
static void example_motion_abort()
{
    int handle = 0;
    int ret = 0;
    //实例化机械臂，将 ip 切换为自己的 ip
    int result = jakaAPI.create_handler("192.168.2.160".ToCharArray(), ref handle);
    JKTYPE.JointValue joint_pos = new JKTYPE.JointValue();
    //机械臂上电
    jakaAPI.power_on(ref handle);
    //机械臂上使能
    jakaAPI.enable_robot(ref handle);
    //定义并初始化关 JointValue 变量
    Console.WriteLine("start_move");
    joint_pos.jVal = new double[] { 0 * PI / 180, 0 * PI / 180, 50 * PI / 180, 0 * PI / 180, 0 * PI / 180, 0 * PI / 180 };
    //关节空间运动，其中 ABS 代表绝对运动，TRUE 代表指令是阻塞的，1 代表速度为 1rad/s
    jakaAPI.joint_move(ref handle, ref joint_pos, JKTYPE.MoveMode.ABS, false, 1);
    System.Threading.Thread.Sleep(500);
    //运动 0.5s 后终止
    jakaAPI.motion_abort(ref handle);
    Console.WriteLine("stop_move");
    jakaAPI.destory_handler(ref handle);
}
```

4.2.10 查询机械臂运动是否停止

```
/**
* @brief 查询机械臂运动是否停止
* @param handle  机械臂控制句柄
```

```
* @param in_pos 查询结果 0 运动进行中，1 运动完成
* @return ERR_SUCC 成功 其他失败
*/
int is_in_pos(ref int handle, ref bool in_pos);
```

代码示例：

```
/// <summary>
/// 查询运动是否到位
/// </summary>
static void example_is_in_pos()
{
    bool in_pos = false;
    int handle = 0;
    //实例化机械臂，将 ip 切换为自己的 ip
    int result = jakaAPI.create_handler("192.168.2.160".ToCharArray(), ref handle);
    //机械臂上电
    jakaAPI.power_on(ref handle);
    //机械臂上使能
    jakaAPI.enable_robot(ref handle);
    //定义并初始化关 JointValue 变量
    JKTYPE.JointValue joint_pos = new JKTYPE.JointValue();
    joint_pos.jVal = new double[] { 1, 1, 1, 1, 1, 1 };
    Console.WriteLine("joint_move 指令执行开始");
    //非阻塞运动到目标点位
    jakaAPI.joint_move(ref handle, ref joint_pos, JKTYPE.MoveMode.ABS, false, 1);
    while (!in_pos)
    {
        //查询是否运动停止
        jakaAPI.is_in_pos(ref handle, ref in_pos);
        Console.WriteLine(" in_pos is :{0}", in_pos);
        System.Threading.Thread.Sleep(200);
    }
    jakaAPI.destory_handler(ref handle);
}
```

4.3 机械臂操作信息设置与查询

4.3.1 获取机械臂状态监测数据

```
/**
* @brief 获取机械臂状态数据
* @param status 机械臂状态
* @return ERR_SUCC 成功 其他失败
```

```
*/
int  get_robot_status(ref int handle, ref JKTYPE.RobotStatus status);
```

代码示例：

```
/// <summary>
/// 获取机械臂状态信息 24 种
/// </summary>
static void example_get_robot_status()
{
    int handle = 0;
    int ret = 0 ;
    JKTYPE.RobotStatus robstatus = new JKTYPE.RobotStatus();
    //实例化机械臂，将 ip 切换为自己的 ip
    int result = jakaAPI.create_handler("192.168.2.160".ToCharArray(), ref handle);
    //机械臂上电
    jakaAPI.power_on(ref handle);
    //机械臂上使能
    jakaAPI.enable_robot(ref handle);
    while (true)
    {
        ret++;
        System.Threading.Thread.Sleep(1000);
        //获取机械臂状态监测数据
        jakaAPI.get_robot_status(ref handle, ref robstatus);
        Console.WriteLine("{1}rapidrate:{0}\n", robstatus.rapidrate, ret);
    }
    jakaAPI.disable_robot(ref handle);
    jakaAPI.power_off(ref handle);
    jakaAPI.destory_handler(ref handle);
}
```

4.3.2 设置机械臂状态数据自动更新时间间隔

```
/**
 * @brief 设置机械臂状态数据自动更新时间间隔
 * @param handle  机械臂控制句柄
 * @param millisecond 时间参数，单位毫秒
 * @return ERR_SUCC 成功 其他失败
 */
int set_status_data_update_time_interval(ref int i, float millisecond);
```

代码示例：

```
/// <summary>
/// 设置机械臂状态 robotstatus 更新频率，默认尽可能快
/// </summary>
```

```
static void example_set_status_data_updata_interval()
{
    float milisec = 100;
    int ret;
    int handle = 0;
    //实例化机械臂，将 ip 切换为自己的 ip
    int result = jakaAPI.create_handler("192.168.2.160".ToCharArray(), ref handle);
    //机械臂上电
    jakaAPI.power_on(ref handle);
    //机械臂上使能
    jakaAPI.enable_robot(ref handle);
    //设置状态更新频率
    ret = jakaAPI.set_status_data_update_time_interval(ref handle, milisec);
}
```

4.3.3 设置数字输出变量

```
/**
* @brief 设置数字输出变量(DO)的值
* @param handle 机械臂控制句柄
* @param type DO 类型
* @param index DO 索引（从 0 开始）
* @param value DO 设置值
* @return ERR_SUCC 成功 其他失败
*/
int set_digital_output(ref int handle, JKTYPE.IOType type, int index, bool value);
```

代码示例：

```
/// <summary>
/// 设置和查询 IO
/// </summary>
static void example_set_digital_optput()
{
    bool DO2 = false;
    int handle = 0;
    //实例化机械臂，将 ip 切换为自己的 ip
    int result = jakaAPI.create_handler("192.168.2.160".ToCharArray(), ref handle);
    jakaAPI.power_on(ref handle);
    jakaAPI.enable_robot(ref handle);
    //查询 do2 的状态
    jakaAPI.get_digital_output(ref handle, JKTYPE.IOType.IO_CABINET, 2, ref DO2);
    Console.WriteLine("D03 = {0}\n", DO2);
    //io_cabinet 是控制柜面板 IO，2 代表 DO2，1 对应要设置的 DO 值。
    jakaAPI.set_digital_output(ref handle, JKTYPE.IOType.IO_CABINET, 2, true);
    System.Threading.Thread.Sleep(1000);
```

```
    //查询 do2 的状态
    jakaAPI.get_digital_output(ref handle, JKTYPE.IOType.IO_CABINET, 2, ref DO2);
    Console.WriteLine("D03 = {0}\n", DO2);
    jakaAPI.destory_handler(ref handle);
}
```

4.3.4 设置模拟输出变量

```
/**
* @brief 设置模拟输出变量的值(AO)的值
* @param handle 机械臂控制句柄
* @param type AO 类型
* @param index AO 索引 （从 0 开始）
* @param value AO 设置值
* @return ERR_SUCC 成功 其他失败
*/
int set_analog_output(ref int handle, JKTYPE.IOType type, int index, float value);
```

4.3.5 查询数字输入状态

```
/**
* @brief 查询数字输入(DI)状态
* @param handle 机械臂控制句柄
* @param type DI 类型
* @param index DI 索引 （从 0 开始）
* @param result DI 状态查询结果
* @return ERR_SUCC 成功 其他失败
*/
int get_digital_input(ref int handle, JKTYPE.IOType type, int index, ref bool result);
```

4.3.6 查询数字输出状态

```
/**
* @brief 查询数字输出(DO)状态
* @param handle 机械臂控制句柄
* @param type DO 类型
* @param index DO 索引 （从 0 开始）
* @param result DO 状态查询结果
* @return ERR_SUCC 成功 其他失败
*/
int get_digital_output(ref int handle, JKTYPE.IOType type, int index, ref bool result);
```

4.3.7 获取模拟量输入变量的值

```
/**
* @brief 获取模拟量输入变量(AI)的值
* @param handle 机械臂控制句柄
* @param type AI 的类型
* @param index AI 索引 （从 0 开始）
* @param result 指定 AI 状态查询结果
* @return ERR_SUCC 成功 其他失败
*/
int get_analog_input(ref int handle, JKTYPE.IOType type, int index, ref float result);
```

4.3.8 获取模拟量输出变量的值

```
/**
* @brief 获取模拟量输出变量(AO)的值
* @param handle 机械臂控制句柄
* @param type AO 的类型
* @param index AO 索引 （从 0 开始）
* @param result 指定 AO 状态查询结果
* @return ERR_SUCC 成功 其他失败
*/
int get_analog_output(ref int handle, JKTYPE.IOType type, int index, ref float result);
```

4.3.9 查询扩展 IO 是否运行

```
/**
* @brief 查询扩展 IO 模块是否运行
* @param handle 机械臂控制句柄
* @param is_running 扩展 IO 模块运行状态查询结果
* @return ERR_SUCC 成功 其他失败
*/
int is_extio_running(ref int handle, ref bool is_running);
```

4.3.10 设置工具信息

```
/**
* @brief 设置指定编号的工具信息
* @param handle 机械臂控制句柄
* @param id 工具编号,取值范围为[1,10]
* @param tcp 工具坐标系相对法兰坐标系偏置
* @param name 指定工具的别名
* @return ERR_SUCC 成功 其他失败
*/
```

```
int set_tool_data(ref int handle, int id, ref JKTYPE.CartesianPose tcp, char[] name);
```

代码示例：

```
/// <summary>
/// 切换工具坐标系及获取与设置工具坐标系信息
/// </summary>
static void example_tool()
{
    int id_ret = 0;
    int id_set = 0;
    int handle = 0;
    id_set = 2;
    JKTYPE.CartesianPose tcp_ret = new JKTYPE.CartesianPose();
    JKTYPE.CartesianPose tcp_set = new JKTYPE.CartesianPose();
    char[] name = new char[50];
    name = "test".ToCharArray();
    //实例化机械臂，将 ip 切换为自己的 ip
    int result = jakaAPI.create_handler("192.168.2.160".ToCharArray(), ref handle);
    //机械臂上电
    jakaAPI.power_on( ref handle);
    //查询当前使用的工具 ID
    jakaAPI.get_tool_id( ref handle,  ref id_ret);
    //获取当前使用的工具信息
    jakaAPI.get_tool_data( ref handle, id_ret,  ref tcp_ret);
    Console.WriteLine("id_using={0} \nx={1}, y={2}, z={3}\n", id_ret, tcp_ret.tran.x, tcp_ret.tran.y, tcp_ret.tran.y);
    Console.WriteLine("rx={0}, ry={1}, rz={2}\n", tcp_ret.rpy.rx, tcp_ret.rpy.ry, tcp_ret.rpy.rz);
    //初始化工具坐标
    tcp_set.tran.x = 0; tcp_set.tran.y = 0; tcp_set.tran.z = 10;
    tcp_set.rpy.rx = 120 * PI / 180; tcp_set.rpy.ry = 90 * PI / 180; tcp_set.rpy.rz = -90 * PI / 180;
    //设置工具信息
    jakaAPI.set_tool_data( ref handle, id_set,  ref tcp_set, name);
    //切换当前使用的工具坐标
    jakaAPI.set_tool_id( ref handle, id_set);
    System.Threading.Thread.Sleep(1000);
    //查询当前使用的工具 ID
    jakaAPI.get_tool_id( ref handle,  ref id_ret);
    //获取设置的工具信息
    jakaAPI.get_tool_data(ref handle, id_ret, ref tcp_ret);
    Console.WriteLine("id_using={0} \nx={1}, y={2}, z={3}\n", id_ret, tcp_ret.tran.x, tcp_ret.tran.y, tcp_ret.tran.y);
    Console.WriteLine("rx={0}, ry={1}, rz={2}\n", tcp_ret.rpy.rx, tcp_ret.rpy.ry, tcp_ret.rpy.rz);
```

```
38.     jakaAPI.destory_handler(ref handle);
39. }
```

4.3.11 获取工具信息

```
1.  /**
2.  * @brief 查询当前使用的工具信息
3.  * @param id 工具 ID 查询结果
4.  * @param tcp 工具坐标系相对法兰坐标系偏置
5.  * @return ERR_SUCC 成功 其他失败
6.  */
7.  int errno_t  get_tool_data(ref int handle, ref int id, ref CartesianPose tcp);
```

4.3.12 设置当前使用的工具 ID

```
1.  /**
2.  * @brief 设置当前使用的工具 ID
3.  * @param handle  机械臂控制句柄
4.  * @param id 工具坐标系 ID ,取值范围为[0,10]，0 为不使用工具即法兰中心
5.  * @return ERR_SUCC 成功 其他失败
6.  */
7.  int set_tool_id(ref int handle, int id);
```

4.3.13 查询当前使用的工具 ID

```
1.  /**
2.  * @brief 查询当前使用的工具 ID
3.  * @param handle  机械臂控制句柄
4.  * @param id 工具 ID 查询结果
5.  * @return ERR_SUCC 成功 其他失败
6.  */
7.  int get_tool_id(ref int handle, ref int id);
```

4.3.14 设定用户坐标系信息

```
1.  /**
2.  * @brief 设置指定编号的用户坐标系信息
3.  * @param handle 机械臂控制句柄
4.  * @param id 用户坐标系编号 ，取值范围为[1,10]
5.  * @param user_frame 用户坐标系偏置值
6.  * @param name 用户坐标系别名
7.  * @return ERR_SUCC 成功 其他失败
8.  */
```

```
int set_user_frame_data(ref int handle, int id, ref JKTYPE.CartesianPose user_frame, char[] name);
```

代码示例：

```
/// <summary>
/// 切换用户坐标系及获取与设置用户坐标系信息
/// </summary>
static void example_user_frame()
{
    int id_ret = 0;
    int id_set = 2;
    int handle = 0;
    JKTYPE.CartesianPose tcp_ret = new JKTYPE.CartesianPose();
    JKTYPE.CartesianPose tcp_set = new JKTYPE.CartesianPose();
    char[] name = new char[50];
    name = "test".ToCharArray();
    //实例化机械臂，将 ip 切换为自己的 ip
    int result = jakaAPI.create_handler("192.168.2.160".ToCharArray(), ref handle);
    //机械臂上电
    jakaAPI.power_on(ref handle);
    //查询当前使用的用户坐标系 ID
    jakaAPI.get_user_frame_id(ref handle, ref id_ret);
    //获取当前使用的用户坐标系信息
    jakaAPI.get_user_frame_data(ref handle, id_ret, ref tcp_ret);
    Console.WriteLine("id_using={0} \nx={1}, y={2}, z={3}\n", id_ret, tcp_ret.tran.x, tcp_ret.tran.y, tcp_ret.tran.y);
    Console.WriteLine("rx={0}, ry={1}, rz={2}\n", tcp_ret.rpy.rx, tcp_ret.rpy.ry, tcp_ret.rpy.rz);
    //初始化用户坐标系坐标
    tcp_set.tran.x = 0; tcp_set.tran.y = 0; tcp_set.tran.z = 10;
    tcp_set.rpy.rx = 120 * PI / 180; tcp_set.rpy.ry = 90 * PI / 180; tcp_set.rpy.rz = -90 * PI / 180;
    //设置用户坐标系信息
    jakaAPI.set_user_frame_data(ref handle, id_set, ref tcp_set, name);
    //切换当前使用的用户坐标系坐标
    jakaAPI.set_user_frame_id(ref handle, id_set);
    System.Threading.Thread.Sleep(1000);
    //查询当前使用的用户坐标系 ID
    jakaAPI.get_user_frame_id(ref handle, ref id_ret);
    //获取当前使用的用户坐标系信息
    jakaAPI.get_user_frame_data(ref handle, id_ret, ref tcp_ret);
    Console.WriteLine("id_using={0} \nx={1}, y={2}, z={3}\n", id_ret, tcp_ret.tran.x, tcp_ret.tran.y, tcp_ret.tran.y);
    Console.WriteLine("rx={0}, ry={1}, rz={2}\n", tcp_ret.rpy.rx, tcp_ret.rpy.ry, tcp_ret.rpy.rz);
```

```
    jakaAPI.destory_handler(ref handle);
}
```

4.3.15 获取用户坐标系信息

```
/**
* @brief 查询当前使用的用户坐标系信息
* @param id 用户坐标系 ID 查询结果
* @param tcp 用户坐标系偏置值
* @return ERR_SUCC 成功 其他失败
*/
int  get_user_frame_data(ref int handle handle, ref int id, ref CartesianPose tcp);
```

4.3.16 设置当前使用的用户坐标系 ID

```
/**
* @brief 设置当前使用的用户坐标系 ID
* @param handle  机械臂控制句柄
* @param id 用户坐标系 ID ，取值范围为[0,10]，其中 0 为世界坐标系
* @return ERR_SUCC 成功 其他失败
*/
int set_user_frame_id(ref int handle, int id);
```

4.3.17 查询当前使用的用户坐标系 ID

```
/**
* @brief 查询当前使用的用户坐标系 ID
* @param handle  机械臂控制句柄
* @param id 获取的结果
* @return ERR_SUCC 成功 其他失败
*/
int get_user_frame_id(ref int handle, ref int id);
```

4.3.18 获取机械臂状态

```
/**
* @brief 获取机械臂状态
* @param handle  机械臂控制句柄
* @param state 机械臂状态查询结果
* @return ERR_SUCC 成功 其他失败
*/
int get_robot_state(ref int handle, ref JKTYPE.RobotState state);
```

代码示例：

```
/// <summary>
/// 获取机械臂状态，(急停 上电 伺服使能)
/// </summary>
static void example_get_robot_state()
{
    int handle = 0;
    //实例化机械臂，将 ip 切换为自己的 ip
    int result = jakaAPI.create_handler("192.168.2.160".ToCharArray(), ref handle);
    JKTYPE.RobotState state = new JKTYPE.RobotState();
    //机械臂上电
    jakaAPI.power_on(ref handle);
    //机械臂上使能
    jakaAPI.enable_robot(ref handle);
    //获取机械臂状态(急停 上电 伺服模式)
    jakaAPI.get_robot_state(ref handle, ref state);
    Console.WriteLine("is e_stoped :{0} ", state.estoped);
    jakaAPI.destory_handler(ref handle);
}
```

4.3.19 获取当前设置下工具末端的位姿

```
/**
* @brief 获取当前设置下工具末端的位姿
* @param handle  机械臂控制句柄
* @param tcp_position 工具末端位置查询结果
* @return ERR_SUCC 成功 其他失败
*/
int get_tcp_position(ref int handle, ref JKTYPE.CartesianPose tcp_position);
```

代码示例：

```
/// <summary>
/// 获取机械臂 tcp 空间位姿
/// </summary>
static void example_get_tcp_position()
{
    int handle = 0;
    JKTYPE.CartesianPose tcp_pos = new JKTYPE.CartesianPose();
    //实例化机械臂，将 ip 切换为自己的 ip
    int result = jakaAPI.create_handler("192.168.2.160".ToCharArray(), ref handle);
    //机械臂上电
    jakaAPI.power_on(ref handle);
    //机械臂上使能
    jakaAPI.enable_robot(ref handle);
    //获取工具末端位姿
```

```
    jakaAPI.get_tcp_position(ref handle, ref tcp_pos);
    Console.WriteLine("tcp_position is :{0} {1} {2} {3} {4} {5}", tcp_pos.tran.x, tcp_pos.tran.y, tcp_pos.tran.z, tcp_pos.rpy.rx, tcp_pos.rpy.ry, tcp_pos.rpy.rz);
    jakaAPI.destory_handler(ref handle);
}
```

4.3.20 获取当前机械臂关节角度

```
/**
* @brief 获取当前机械臂关节角度
* @param handle  机械臂控制句柄
* @param joint_position 关节角度查询结果
* @return ERR_SUCC 成功 其他失败
*/
int get_joint_position(ref int handle, ref JKTYPE.JointValue joint_position);
```

代码示例：

```
/// <summary>
/// 获取机械臂关节空间关节角矩阵
/// </summary>
static void example_get_joint_position()
{
    int handle = 0;
    JKTYPE.JointValue jot_pos = new JKTYPE.JointValue();
    //实例化机械臂，将 ip 切换为自己的 ip
    int result = jakaAPI.create_handler("192.168.2.160".ToCharArray(), ref handle);
    //机械臂上电
    jakaAPI.power_on(ref handle);
    //机械臂上使能
    jakaAPI.enable_robot(ref handle);
    //获取工具末端位姿
    jakaAPI.get_joint_position(ref handle, ref jot_pos);
    Console.WriteLine("tcp_position is :{0} {1} {2} {3} {4} {5}", jot_pos.jVal[0], jot_pos.jVal[1], jot_pos.jVal[2], jot_pos.jVal[3], jot_pos.jVal[4], jot_pos.jVal[5]);
    jakaAPI.destory_handler(ref handle);
}
```

4.3.21 机械臂负载设置

```
/**
* @brief 机械臂负载设置
* @param handle  机械臂控制句柄
* @param payload 负载质心、质量数据
```

```
* @return ERR_SUCC 成功 其他失败
*/
int set_payload(ref int handle, ref JKTYPE.PayLoad payload);
```

代码示例：

```
/// <summary>
/// 获取与设置负载
/// </summary>
static void example_payload()
{
    int handle = 0;
    int ret;
    //实例化机械臂，将 ip 切换为自己的 ip
    int result = jakaAPI.create_handler("192.168.2.160".ToCharArray(), ref handle);
    JKTYPE.PayLoad payloadret = new JKTYPE.PayLoad();
    payloadret.mass = 0;
    payloadret.centroid.x = 0; payloadret.centroid.y = 0; payloadret.centroid.z = 0;
    JKTYPE.PayLoad payload_set = new JKTYPE.PayLoad();
    //查询当前负载数据
    jakaAPI.get_payload(ref handle, ref payloadret);
    Console.WriteLine(" payload mass is : {0} kg", payloadret.mass);
    Console.WriteLine(" payload center of mass is \nx: {0}  y: {1} z: {2} ", payloadret.centroid.x, payloadret.centroid.y, payloadret.centroid.z);
    payload_set.mass = 1.0;
    //单位 mm
    payload_set.centroid.x = 0; payload_set.centroid.y = 0; payload_set.centroid.z = 10;
    //设置当前负载数据
    jakaAPI.set_payload(ref handle, ref payload_set);
    //查询当前负载数据
    jakaAPI.get_payload(ref handle, ref payloadret);
    Console.WriteLine(" payload mass is : {0} kg", payloadret.mass);
    Console.WriteLine(" payload center of mass is \nx: {0}  y: {1} z: {2} ", payloadret.centroid.x, payloadret.centroid.y, payloadret.centroid.z);
    jakaAPI.destory_handler(ref handle);
}
```

4.3.22 获取机械臂负载数据

```
/**
* @brief 获取机械臂负载数据
* @param handle  机械臂控制句柄
* @param payload 负载查询结果
* @return ERR_SUCC 成功 其他失败
*/
```

```
int get_payload(ref int handle, ref JKTYPE.PayLoad payload);
```

4.3.23 设置 tioV3 电压参数

```
/**
* @brief 设置 tioV3 电压参数
* @param vout_enable 电压使能，0:关，1 开
* @param vout_vol 电压大小 0:24v 1:12v
* @return ERR_SUCC 成功 其他失败
*/
errno_t set_tio_vout_param(ref int handle, int vout_enable, int vout_vol);
```

4.3.24 获取 tioV3 电压参数

```
/**
* @brief 获取 tioV3 电压参数
* @param vout_enable 电压使能，0:关，1 开
* @param vout_vol 电压大小 0:24v 1:12v
* @return ERR_SUCC 成功 其他失败
*/
errno_t get_tio_vout_param(ref int handle, ref int vout_enable, ref int vout_vol);
```

4.3.25 获取机械臂状态

```
/**
* @brief 获取机械臂状态
* @param state 机械臂状态查询结果
* @return ERR_SUCC 成功 其他失败
*/
int get_tcp_position(ref int handle, ref JKTYPE.CartesianPose tcp_position);
```

4.3.26 TIO 添加或修改信号量

```
/**
* @brief 添加或修改信号量
* @param sign_info 信号量参数
* @return ERR_SUCC 成功 其他失败
*/
int add_tio_rs_signal(ref int i, JKTYPE.SignInfo sign_info);
```

4.3.27 TIO 删除信号量

```
/**
* @brief 删除信号量
* @param sig_name 信号量名称
* @return ERR_SUCC 成功 其他失败
*/
int del_tio_rs_signal(ref int i, char[] sig_name);
```

4.3.28 TIO RS485 发送指令

```
/**
* @brief RS485 发送命令
* @param chn_id 通道号
* @param data 数据字段
* @return ERR_SUCC 成功 其他失败
*/
int send_tio_rs_command(ref int i, int chn_id, byte[] data, int buffsize);
```

4.3.29 TIO 获取信号量信息

```
/**
* @brief 获取信号量信息
* @param SignInfo* 信号量信息数组
* @return ERR_SUCC 成功 其他失败
*/
int get_rs485_signal_info(ref int i, ref JKTYPE.SignInfo sign_info);
```

4.3.30 TIO 设置 TIO 模式

```
/**
* @brief 设置 tio 模式
* @param pin_type tio 类型 0 for DI Pins, 1 for DO Pins, 2 for AI Pins
* @param pin_type tio 模式 DI Pins: 0:0x00 DI2 为 NPN,DI1 为 NPN,1:0x01 DI2 为 NPN,DI1 为 PNP, 2:0x10 DI2 为 PNP,DI1 为 NPN,3:0x11 DI2 为 PNP,DI1 为 PNP
                                DO Pins: 低 8 位数据高 4 位为 DO2 配置,低四位为 DO1 配置,0x0 DO 为 NPN 输出, 0x1 DO 为 PNP 输出, 0x2 DO 为推挽输出, 0xF RS485H 接口
                                AI Pins: 0:模拟输入功能使能, RS485L 禁止, 1:RS485L 接口使能, 模拟输入功能禁止
* @return ERR_SUCC 成功 其他失败
```

```
*/
int set_tio_pin_mode(ref int i, int pin_type, int pin_mode);
```

4.3.31 TIO 获取 TIO 模式

```
/**
* @brief 获取 tio 模式
* @param pin_type tio 类型 0 for DI Pins, 1 for DO Pins, 2 for AI Pins
* @param pin_type tio 模式 DI Pins: 0:0x00 DI2 为 NPN,DI1 为 NPN,1:0x01 DI2 为 NPN,DI1 为 PNP, 2:0x10 DI2 为 PNP,DI1 为 NPN,3:0x11 DI2 为 PNP,DI1 为 PNP
                                DO Pins: 低 8 位数据高 4 位为 DO2 配置,低四位为 DO1 配置,0x0 DO 为 NPN 输出, 0x1 DO 为 PNP 输出, 0x2 DO 为推挽输出, 0xF RS485H 接口
                                AI Pins: 0:模拟输入功能使能, RS485L 禁止, 1:RS485L 接口使能, 模拟输入功能禁止
* @return ERR_SUCC 成功 其他失败
*/
int get_tio_pin_mode(ref int i, int pin_type, ref int pin_mode);
```

4.3.32 TIO RS485 通讯参数配置

```
/**
* @brief RS485 通讯参数配置
* @param ModRtuComm 当通道模式设置为 Modbus RTU 时，需额外指定 Modbus 从站节点 ID
* @return ERR_SUCC 成功 其他失败
*/
int set_rs485_chn_comm(ref int i, JKTYPE.ModRtuComm mod_rtu_com);
```

4.3.33 TIO RS485 通讯参数查询

```
/**
* @brief RS485 通讯参数查询
* @param ModRtuComm 查询时 chn_id 作为输入参数
* @return ERR_SUCC 成功 其他失败
*/
int get_rs485_chn_comm(ref int i, ref JKTYPE.ModRtuComm mod_rtu_com);
```

4.3.34 TIO RS485 通讯模式配置

```
/**
* @brief RS485 通讯模式配置
```

```
3.  * @param chn_id 0: RS485H, channel 1; 1: RS485L, channel 2
4.  * @param chn_mode 0: Modbus RTU, 1: Raw RS485, 2, torque sensor
5.  * @return ERR_SUCC 成功 其他失败
6.  */
7.  int set_rs485_chn_mode(ref int i, int chn_id, int chn_mode);
```

4.3.35 TIO RS485 通讯模式查询

```
1.  /**
2.  * @brief RS485 通讯模式查询
3.  * @param chn_id 输入参数 0: RS485H, channel 1; 1: RS485L, channel 2
4.  * @param chn_mode 输出参数  0: Modbus RTU, 1: Raw RS485, 2, torque sensor
5.  * @return ERR_SUCC 成功 其他失败
6.  */
7.  int get_rs485_chn_mode(ref int handle, int chn_id, ref int chn_mode);
```

4.4 机械臂安全状态设置

4.4.1 查询机械臂是否处于碰撞保护模式

```
1.  /**
2.  * @brief 查询机械臂是否处于碰撞保护模式
3.  * @param handle  机械臂控制句柄
4.  * @param in_collision 查询结果 0 为不在碰撞，1 为出现碰撞
5.  * @return ERR_SUCC 成功 其他失败
6.  */
7.  int is_in_collision(ref int handle, ref bool in_collision);
1.  /**
```

4.4.2 查询机械臂是否超出限位

```
2.  * @brief 查询机械臂是否超出限位
3.  * @param handle  机械臂控制句柄
4.  * @param on_limit 查询结果 0 未超限，1 超限
5.  * @return ERR_SUCC 成功 其他失败
6.  */
7.  int is_on_limit(ref int handle, ref bool on_limit);
```

4.4.3 碰撞之后从碰撞保护模式恢复

```
1.  /**
2.  * @brief 碰撞之后从碰撞保护模式恢复
3.  * @param handle  机械臂控制句柄
```

```
4.  * @return ERR_SUCC 成功 其他失败
5.  */
6.  int collision_recover(ref int handle);
```

代码示例：

```
1.  /// <summary>
2.  /// 碰撞保护状态查询及恢复
3.  /// </summary>
4.  static void example_collision_recover()
5.  {
6.      bool in_collision = false;
7.      int handle = 0;
8.      //实例化机械臂，将 ip 切换为自己的 ip
9.      int result = jakaAPI.create_handler("192.168.2.160".ToCharArray(), ref handle);
10.     //机械臂上电
11.     jakaAPI.power_on(ref handle);
12.     //机械臂上使能
13.     jakaAPI.enable_robot(ref handle);
14.     //查询是否处于碰撞保护状态
15.     jakaAPI.is_in_collision(ref handle, ref in_collision);
16.     if (in_collision)
17.     //如果处于碰撞保护模式，则从碰撞保护中恢复
18.     {
19.         jakaAPI.collision_recover(ref handle);
20.         Console.WriteLine("robot collision recover");
21.     }
22.     else
23.     {
24.         Console.WriteLine("robot is not collision");
25.     }
26.     jakaAPI.destory_handler(ref handle);
27. }
```

4.4.4 设置机械臂碰撞等级

```
1.  /**
2.  * @brief 设置机械臂碰撞等级
3.  * @param handle  机械臂控制句柄
4.  * @param level  碰撞等级，取值范围[0,5] ，其中 0 为关闭碰撞，1 为碰撞阈值 25N，2 为碰撞阈值 50N，3 为碰撞阈值 75N，4 为碰撞阈值 100N，5 为碰撞阈值 125N
5.  * @return ERR_SUCC 成功 其他失败
6.  */
7.  int set_collision_level(ref int handle,int level);
```

代码示例：

```
1.  /// <summary>
```

```
/// 碰撞等级的设置和查询
/// </summary>
static void  example_collision_level()
{
    int level = 0 ;
    int handle = 0;
    //实例化机械臂，将 ip 切换为自己的 ip
    int result = jakaAPI.create_handler("192.168.2.160".ToCharArray(), ref handle);
    //机械臂上电
    jakaAPI.power_on(ref handle);
    //机械臂上使能
    jakaAPI.enable_robot(ref handle);
    //查询当前碰撞等级
    jakaAPI.get_collision_level(ref handle, ref level);
    Console.WriteLine(" collision level is :{0}", level);
    //设置碰撞等级，[0,5]，0 为关闭碰撞，1 为碰撞阈值 25N，2 为碰撞阈值 50N，3 为碰撞阈值 75N，4 为碰撞阈值 100N，5 为碰撞阈值 125N，
    jakaAPI.set_collision_level(ref handle, 2);
    //查询当前碰撞等级
    jakaAPI.get_collision_level(ref handle, ref level);
    Console.WriteLine(" collision level is :{0}", level);
    jakaAPI.destory_handler(ref handle);
}
```

4.4.5 获取机器设置的碰撞等级

```
/**
* @brief 获取机械臂设置的碰撞等级
* @param handle  机械臂控制句柄
* @return ERR_SUCC 成功 其他失败
*/
int get_collision_level(ref int handle, ref int level);
```

4.4.6 注册机械臂出错时的回调函数

```
/**
* @brief 注册机械臂出现错误时的回调函数
* @param handle  机械臂控制句柄
* @param func 指向用户定义的函数的函数指针
* @param error_code 机械臂的错误码
*/
int set_error_handler(ref int i, CallBackFuncType func);
```

代码示例：

```
//错误处理，多线程方式进行
static void user_error_handle(int error_code)
{
    Console.WriteLine("{0}", error_code);
    int a = 0;
    a = Console.Read();
    Console.WriteLine("{0}\n", a);
}
//注册
static void example_set_err_handle()
{
    int handle = 0;
    //实例化机械臂，将 ip 切换为自己的 ip
    int result = jakaAPI.create_handler("192.168.2.160".ToCharArray(), ref handle);
    //机械臂上电
    jakaAPI.power_on(ref handle);
    //机械臂上使能
    jakaAPI.enable_robot(ref handle);
    //设置用户异常回调函数
    jakaAPI.set_error_handler(ref handle ,user_error_handle);
    while (true)
    {
        System.Threading.Thread.Sleep(1000);
        Console.WriteLine("nothing happen");
    }
    jakaAPI.destory_handler(ref handle);
}
```

4.4.7 设置机械臂错误码文件存放路径

```
/**
* @brief 设置错误码文件路径，需要使用 get_last_error 接口时需要设置错误码文件路径，如果不使用 get_last_error 接口，则不需要设置该接口
* @return ERR_SUCC 成功 其他失败
*/
int set_errorcode_file_path(ref int handle, char[] path);
```

4.4.8 获取机械臂目前发生的最后一个错误码

```
/**
* @brief 获取机械臂运行过程中最后一个错误码,当调用 clear_error 时，最后一个错误码会清零
* @return ERR_SUCC 成功 其他失败
*/
```

```
int  get_last_error(ref int handle, ref JKTYPE.ErrorCode code);
```

4.4.9 设置网络异常，机械臂终止运动等待时间

```
/**
* @brief 设置网络异常，SDK 与机械臂控制器失去连接后多长时间机械臂控制器终止机械臂当前运动
* @param handle  机械臂控制句柄
* @param millisecond 时间参数，单位毫秒
* @param mnt 网络异常时机械臂需要进行的动作类型
* @return ERR_SUCC 成功 其他失败
*/
int set_network_exception_handle(ref int i, float millisecond, JKTYPE.ProcessType mnt);
```

4.5 使用 APP 脚本程序

4.5.1 运行当前加载的作业程序

```
/**
* @brief 运行当前加载的作业程序
* @param handle  机械臂控制句柄
* @return ERR_SUCC 成功 其他失败
*/
int program_run(ref int handle);
```

代码示例：

```
/// <summary>
/// 加载在 app 上编写好的程序，并进行过程控制
/// </summary>
static void example_program()
{
    System.Text.StringBuilder name = new System.Text.StringBuilder();
    int handle = 0;
    int cur_line = 0;
    //实例化机械臂，将 ip 切换为自己的 ip
    int result = jakaAPI.create_handler("192.168.2.160".ToCharArray(), ref handle);
    JKTYPE.ProgramState pstatus = new JKTYPE.ProgramState();;
    //机械臂上电
    jakaAPI.power_on(ref handle);
    //机械臂上使能
    jakaAPI.enable_robot(ref handle);
    //加载预先通过 app 编辑的名字叫 simple 的脚本
    jakaAPI.program_load(ref handle, "simple".ToCharArray());
    //获取已加载的程序名
    jakaAPI.get_loaded_program(ref handle, name);
```

```
    Console.WriteLine("Pro_name is:  {0}", name);
    //运行当前加载的程序
    jakaAPI.program_run(ref handle);
    System.Threading.Thread.Sleep(10000);
    //暂停当前运行的程序
    jakaAPI.program_pause(ref handle);
    //获取当前执行程序的行号
    jakaAPI.get_current_line(ref handle, ref cur_line);
    Console.WriteLine("cur_line is: {0}", cur_line);
    //获取当前程序状态
    jakaAPI.get_program_state(ref handle, ref pstatus);
    Console.WriteLine("pro_status is: {0}", pstatus);
    System.Threading.Thread.Sleep(2000);
    //继续运行当前程序
    jakaAPI.program_resume(ref handle);
    System.Threading.Thread.Sleep(10000);
    //终止当前程序
    jakaAPI.program_abort(ref handle);
    jakaAPI.destory_handler(ref handle);
}
```

4.5.2 暂停当前运行的作业程序

```
/**
* @brief 暂停当前运行的作业程序
* @param handle  机械臂控制句柄
* @return ERR_SUCC 成功 其他失败
*/
int program_pause(ref int handle);
```

4.5.3 继续运行当前暂停的作业程序

```
/**
* @brief 继续运行当前暂停的作业程序
* @param handle  机械臂控制句柄
* @return ERR_SUCC 成功 其他失败
*/
int program_resume(ref int handle);
```

4.5.4 终止当前执行的作业程序

```
/**
* @brief 终止当前执行的作业程序
* @param handle  机械臂控制句柄
```

```
4.  * @return ERR_SUCC 成功 其他失败
5.  */
6.  int program_abort(ref int handle);
```

4.5.5 加载指定的作业程序

```
1.  /**
2.  * @brief 加载指定的作业程序 （加载轨迹复现数据，轨迹复现数据的加载需要在文件夹名字前加上 track/）
3.  * @param handle 机械臂控制句柄
4.  * @param file 程序文件路径
5.  * @return ERR_SUCC 成功 其他失败
6.  */
7.  int program_load(ref int handle, char[] file);
```

4.5.6 获取已加载的作业程序的名字

```
1.  /**
2.  * @brief 获取已加载的作业程序名字
3.  * @param handle 机械臂控制句柄
4.  * @param file 程序文件路径
5.  * @return ERR_SUCC 成功 其他失败
6.  */
7.  int get_loaded_program(ref int handle, StringBuilder file);
```

4.5.7 获取当前机械臂作业程序的执行行号

```
1.  /**
2.  * @brief 获取当前机械臂作业程序的执行行号
3.  * @param handle  机械臂控制句柄
4.  * @param curr_line 当前行号查询结果
5.  * @return ERR_SUCC 成功 其他失败
6.  */
7.  int get_current_line(ref int handle, ref int curr_line);
```

4.5.8 获取机械臂作业程序的执行状态

```
1.  /**
2.  * @brief 获取机械臂作业程序执行状态
3.  * @param handle  机械臂控制句柄
4.  * @param status 作业程序执行状态查询结果
5.  * @return ERR_SUCC 成功 其他失败
6.  */
7.  int get_program_state(ref int handle, ref JKTYPE.ProgramState status);
```

4.5.9 设置机械臂的运行倍率

```
/**
* @brief 设置机械臂运行倍率
* @param handle  机械臂控制句柄
* @param rapid_rate 是程序运行倍率，设置范围为[0,1]
* @return ERR_SUCC 成功 其他失败
*/
int set_rapidrate(ref int handle, double rapid_rate);
```

代码示例：

```
/// <summary>
/// 获取与设置机械臂的运行速度
/// </summary>
static void example_rapidrate()
{
    int handle = 0;
    double rapid_rate = 0;
    //实例化机械臂，将 ip 切换为自己的 ip
    int result = jakaAPI.create_handler("192.168.2.160".ToCharArray(), ref handle);
    //机械臂上电
    jakaAPI.power_on(ref handle);
    //机械臂上使能
    jakaAPI.enable_robot(ref handle);
    jakaAPI.get_rapidrate(ref handle, ref rapid_rate);
    Console.WriteLine("rapid_rate is : {0}", rapid_rate);
    //修改程序运行时机械臂运动速度
    jakaAPI.set_rapidrate(ref handle, 0.4);
    System.Threading.Thread.Sleep(100);
    jakaAPI.get_rapidrate(ref handle, ref rapid_rate);
    Console.WriteLine("rapid_rate is : {0}", rapid_rate);
    jakaAPI.destory_handler(ref handle);
}
```

4.5.10 获取机械臂的运行倍率

```
/**
* @brief 获取机械臂运行倍率
* @param handle 机械臂控制句柄
* @param rapid_rate 当前控制系统倍率
* @return ERR_SUCC 成功 其他失败
*/
int get_rapidrate(ref int handle, ref double rapid_rate);
```

4.6 机械臂轨迹复现

4.6.1 设置轨迹复现配置参数

```
/**
* @brief 设置轨迹复现配置参数
* @param para 轨迹复现配置参数
* @return ERR_SUCC 成功 其他失败
*/
int  set_traj_config(ref int handle, ref JKTYPE.TrajTrackPara para);
```

4.6.2 获取轨迹复现配置参数

```
/**
* @brief 获取轨迹复现配置参数
* @param para 轨迹复现配置参数
* @return ERR_SUCC 成功 其他失败
*/
int  get_traj_config(ref int handle, ref JKTYPE.TrajTrackPara para);
```

4.6.3 采集轨迹复现数据控制开关

```
/**
* @brief 采集轨迹复现数据控制开关
* @param mode 选择 TRUE 时，开始数据采集，选择 FALSE 时，结束数据采集
* @param filename 采集数据的存储文件名，当 filename 为空指针时，存储文件以当前日期命名
* @return ERR_SUCC 成功 其他失败
*/
int  set_traj_sample_mode(ref int handle, ref bool mode,char[] filename);
```

代码示例：

```
  /// <summary>
/// 轨迹复现
/// </summary>
static void example_traj_sample()
{
    bool samp_stu = false;
    int handle = 0;
    int ret = 0;
    //实例化机械臂，将 ip 切换为自己的 ip
    int result = jakaAPI.create_handler("192.168.2.160".ToCharArray(), ref handle);
    jakaAPI.set_debug_mode(ref handle, true);
    JKTYPE.JointValue joint_pos = new JKTYPE.JointValue();
    joint_pos.jVal = new double[] { 1, 1, 1, 1, 1, 1 };
```

```
    JKTYPE.JointValue joint_pos2 = new JKTYPE.JointValue();
    joint_pos2.jVal = new double[] { -1, 1, 1, 1, 1, 1 };
    //机械臂上电
    jakaAPI.power_on(ref handle);
    //机械臂上使能
    jakaAPI.enable_robot(ref handle);
    char[] name = new char[40];
    char[] trajname = new char[40];
    name = "testxx".ToCharArray() ;
    trajname = "track/testxx".ToCharArray();
    jakaAPI.joint_move(ref handle, ref joint_pos2, JKTYPE.MoveMode.ABS, true, 1);
    //开启轨迹复现数据采集开关
    jakaAPI.set_traj_sample_mode(ref handle, true, name);
    //查询轨迹复现采集状态
    jakaAPI.get_traj_sample_status(ref handle, ref samp_stu);

    //关节空间运动，其中 ABS 代表绝对运动，TRUE 代表指令是阻塞的，1 代表速度为 1rad/s
    jakaAPI.joint_move(ref handle, ref joint_pos, JKTYPE.MoveMode.ABS, true, 1);
    jakaAPI.joint_move(ref handle, ref joint_pos2, JKTYPE.MoveMode.ABS, true, 1);
    jakaAPI.set_traj_sample_mode(ref handle, false, name);
    System.Threading.Thread.Sleep(500);
    jakaAPI.generate_traj_exe_file(ref handle, name);
    System.Threading.Thread.Sleep(500);
    jakaAPI.program_load(ref handle, trajname);
    System.Threading.Thread.Sleep(100);
    jakaAPI.program_run(ref handle);
    jakaAPI.destory_handler(ref handle);
}
```

4.6.4 采集轨迹复现数据状态查询

```
/**
* @brief 采集轨迹复现数据状态查询
* @param mode 为 TRUE 时，数据正在采集，为 FALSE 时，数据采集结束，在数据采集状态时不允许再次开启数据采集开关
* @return ERR_SUCC 成功 其他失败
*/
int  get_traj_sample_status(ref int handle, ref bool sample_status);
```

4.6.5 查询控制器中已经存在的轨迹复现数据的文件名

```
/**
* @brief 查询控制器中已经存在的轨迹复现数据的文件名
```

```
3.  * @param filename 控制器中已经存在的轨迹复现数据的文件名
4.  * @return ERR_SUCC 成功 其他失败
5.  */
6.  int  get_exist_traj_file_name(ref int handle, ref JKTYPE.MultStrStorType filename);
```

4.6.6 重命名轨迹复现数据的文件名

```
1.  /**
2.  * @brief 重命名轨迹复现数据的文件名
3.  * @param src 原文件名
4.  * @param dest 目标文件名，文件名长度不能超过 100 个字符，文件名不能为空，目标文件名不支持中文
5.  * @return ERR_SUCC 成功 其他失败
6.  */
7.  int  rename_traj_file_name(ref int handle, ref char[] src,ref char[] dest);
```

4.6.7 删除控制器中轨迹复现数据文件

```
1.  /**
2.  * @brief 删除控制器中轨迹复现数据文件
3.  * @param filename 要删除的文件的文件名，文件名为数据文件名字
4.  * @return ERR_SUCC 成功 其他失败
5.  */
6.  int  remove_traj_file(ref int handle, ref char[] filename);
```

4.6.8 控制器中轨迹复现数据文件生成控制器执行脚本

```
1.  /**
2.  * @brief 控制器中轨迹复现数据文件生成控制器执行脚本
3.  * @param filename 数据文件的文件名，文件名为数据文件名字，不带后缀
4.  * @return ERR_SUCC 成功 其他失败
5.  */
6.  int  generate_traj_exe_file(ref int handle, ref char[] filename);
```

4.7 机械臂运动学

4.7.1 机械臂求解逆解

```
1.  /**
2.  * @brief 计算指定位姿在当前工具、当前安装角度以及当前用户坐标系设置下的逆解
3.  * @param handle  机械臂控制句柄
4.  * @param ref_pos 逆解计算用的参考关节空间位置
5.  * @param cartesian_pose 笛卡尔空间位姿值 ，欧拉角注意为弧度制
```

```
 * @param joint_pos 计算成功时关节空间位置计算结果
 * @return ERR_SUCC 成功 其他失败
 */
int kine_inverse(ref int handle, ref JKTYPE.JointValue ref_pos, ref JKTYPE.CartesianPose cartesian_pose, ref JKTYPE.JointValue joint_pos);
```

代码示例：

```
/// <summary>
/// 运动学 逆解 已知 tcp_pos，求 joint_pos
/// </summary>
static void example_kine_inverse()
{
    int handle = 0;
    int ret;
    //实例化机械臂，将 ip 切换为自己的 ip
    int result = jakaAPI.create_handler("192.168.2.160".ToCharArray(), ref handle);
    //初始化参考点
    JKTYPE.JointValue ref_jpos = new JKTYPE.JointValue();
    ref_jpos.jVal = new double[] { 0.558, 0.872, 0.872, 0.349, 0.191, 0.191 };
    //初始化笛卡尔空间点坐标
    JKTYPE.CartesianPose tcp_pos = new JKTYPE.CartesianPose();
    tcp_pos.tran.x = 243.568; tcp_pos.tran.y = 164.064; tcp_pos.tran.z = 300.002;
    tcp_pos.rpy.rx = -1.81826; tcp_pos.rpy.ry = -0.834253; tcp_pos.rpy.rz = -2.30243;
    //初始化返回值
    JKTYPE.JointValue joint_pos = new JKTYPE.JointValue();
    joint_pos.jVal = new double[] { 0, 0, 0, 0, 0, 0 };
    //机械臂逆解 已知 tcp_pos，求 joint_pos
    ret = jakaAPI.kine_inverse(ref handle, ref ref_jpos, ref tcp_pos, ref joint_pos);
    Console.WriteLine(" {0}   kine_inverse:", ret);//若返回值为-4 则逆解失败，更改参考点或数学上无解
    Console.WriteLine("{0} {1} {2} ", joint_pos.jVal[0], joint_pos.jVal[1], joint_pos.jVal[2]);
    Console.WriteLine("{0} {1} {2} ", joint_pos.jVal[3], joint_pos.jVal[4], joint_pos.jVal[5]);
    jakaAPI.destory_handler(ref handle);
}
```

4.7.2 机械臂求解正解

```
/**
 * @brief 计算指定关节位置在当前工具、当前安装角度以及当前用户坐标系设置下的位姿值
 * @param handle  机械臂控制句柄
 * @param joint_pos 关节空间位置
 * @param cartesian_pose 笛卡尔空间位姿计算结果
```

```
 * @return ERR_SUCC 成功 其他失败
 */
int kine_forward(ref int handle, ref JKTYPE.JointValue joint_pos, ref JKTYPE.CartesianPose cartesian_pose);
```

代码示例：

```
/// <summary>
/// 机械臂正解 已知 joint_pos,求 tcp_pos
/// </summary>
static void example_kine_forward()
{
    int handle = 0;
    int ret;
    //实例化机械臂，将 ip 切换为自己的 ip
    int result = jakaAPI.create_handler("192.168.2.160".ToCharArray(), ref handle);
    //初始化笛卡尔空间点坐标
    JKTYPE.CartesianPose tcp_pos = new JKTYPE.CartesianPose();
    //初始化返回值
    JKTYPE.JointValue joint_pos = new JKTYPE.JointValue();
    joint_pos.jVal = new double[] { 0.558, 0.872, 0.872, 0.349, 0.191, 0.191 };
    ret = jakaAPI.kine_forward(ref handle, ref joint_pos, ref tcp_pos);
    Console.WriteLine("tcp_pos is :\n x:{0}  y:{1}  z:{2} rx:{3}  ry:{4}  rz:{5}", tcp_pos.tran.x, tcp_pos.tran.y, tcp_pos.tran.z, tcp_pos.rpy.rx, tcp_pos.rpy.ry, tcp_pos.rpy.rz);
    jakaAPI.destory_handler(ref handle);
}
```

4.7.3 欧拉角到旋转矩阵的转换

```
/**
 * @brief 欧拉角到旋转矩阵的转换
 * @param handle  机械臂控制句柄
 * @param rpy 待转换的欧拉角数据
 * @param rot_matrix 转换后的旋转矩阵
 * @return ERR_SUCC 成功 其他失败
 */
int rpy_to_rot_matrix(ref int handle, ref JKTYPE.Rpy rpy, ref JKTYPE.RotMatrix rot_matrix);
```

代码示例：

```
  /// <summary>
/// 欧拉角->旋转矩阵->四元数
/// </summary>
static void example_rpy2rot_matr2quat()
{
    int handle = 0;
    int ret;
```

```
    //实例化机械臂，将 ip 切换为自己的 ip
    int result = jakaAPI.create_handler("192.168.2.160".ToCharArray(), ref handle);
    //初始化欧拉角
    JKTYPE.Rpy rpy = new JKTYPE.Rpy();
    rpy.rx = -1.81826; rpy.ry = -0.834253; rpy.rz = -2.30243;
    //初始化旋转矩阵
    JKTYPE.RotMatrix rot_matrix = new JKTYPE.RotMatrix();
    rot_matrix.x.x = 0; rot_matrix.y.x = 0; rot_matrix.z.x = 0;
    rot_matrix.x.y = 0; rot_matrix.y.y = 0; rot_matrix.z.y = 0;
    rot_matrix.x.z = 0; rot_matrix.y.z = 0; rot_matrix.z.z = 0;
    //初始化四元数
    JKTYPE.Quaternion quat = new JKTYPE.Quaternion();
    quat.s = 0; quat.x = 0; quat.y = 0; quat.z = 0;
    //欧拉角到旋转矩阵
    ret = jakaAPI.rpy_to_rot_matrix(ref handle, ref rpy, ref rot_matrix);
    Console.WriteLine("     eul2rotm");
    Console.WriteLine("{0} {1} {2}", rot_matrix.x.x, rot_matrix.y.x, rot_matrix.z.x);
    Console.WriteLine("{0} {1} {2}", rot_matrix.x.y, rot_matrix.y.y, rot_matrix.z.y);
    Console.WriteLine("{0} {1} {2}", rot_matrix.x.z, rot_matrix.y.z, rot_matrix.z.z);
    //旋转矩阵--->四元数
    ret = jakaAPI.rot_matrix_to_quaternion(ref handle, ref rot_matrix, ref quat);
    Console.WriteLine(" {0}   rotm2quat:", ret);
    Console.WriteLine("{0} {1} {2} {3} ", quat.s, quat.x, quat.y, quat.z);
    jakaAPI.destory_handler(ref handle);
}
```

4.7.4 旋转矩阵到欧拉角的转换

```
/**
* @brief 旋转矩阵到欧拉角的转换
* @param handle  机械臂控制句柄
* @param rot_matrix 待转换的旋转矩阵数据
* @param rpy 转换后的 RPY 欧拉角结果
* @return ERR_SUCC 成功 其他失败
*/
int rot_matrix_to_rpy(ref int handle, ref JKTYPE.RotMatrix rot_matrix, ref JKTYPE.Rpy rpy);
```

代码示例：

```
  /// <summary>
/// 四元数->旋转矩阵->欧拉角
/// </summary>
static void example_quat2rot_matr2rpy()
{
    int handle = 0;
```

```
    int ret;
    //实例化机械臂，将 ip 切换为自己的 ip
    int result = jakaAPI.create_handler("192.168.2.160".ToCharArray(), ref handle);
    //初始化欧拉角
    JKTYPE.Rpy rpy = new JKTYPE.Rpy();
    rpy.rx = 0; rpy.ry = 0; rpy.rz = 0;
    //初始化旋转矩阵
    JKTYPE.RotMatrix rot_matrix = new JKTYPE.RotMatrix();
    rot_matrix.x.x = 0; rot_matrix.y.x = 0; rot_matrix.z.x = 0;
    rot_matrix.x.y = 0; rot_matrix.y.y = 0; rot_matrix.z.y = 0;
    rot_matrix.x.z = 0; rot_matrix.y.z = 0; rot_matrix.z.z = 0;
    //初始化四元数
    JKTYPE.Quaternion quat = new JKTYPE.Quaternion();
    quat.s = 0.0629; quat.x = 0.522886; quat.y = -0.5592; quat.z = 0.6453;
    //四元数-->旋转矩阵
    ret = jakaAPI.quaternion_to_rot_matrix(ref handle, ref quat, ref rot_matrix);
    Console.WriteLine(" {0}    quatl2rotm:", ret);
    Console.WriteLine("{0} {1} {2}", rot_matrix.x.x, rot_matrix.y.x, rot_matrix.z.x);
    Console.WriteLine("{0} {1} {2}", rot_matrix.x.y, rot_matrix.y.y, rot_matrix.z.y);
    Console.WriteLine("{0} {1} {2}", rot_matrix.x.z, rot_matrix.y.z, rot_matrix.z.z);
    //旋转矩阵--->欧拉角
    ret = jakaAPI.rot_matrix_to_rpy(ref handle, ref rot_matrix, ref rpy);
    Console.WriteLine(" {0}    rotm2eul:", ret);
    Console.WriteLine("{0} {1} {2} ", rpy.rx, rpy.ry, rpy.rz);
    jakaAPI.destory_handler(ref handle);
}
```

4.7.5 四元数到旋转矩阵的转换

```
/**
* @brief 四元数到旋转矩阵的转换
* @param handle  机械臂控制句柄
* @param quaternion 待转换的四元数数据
* @param rot_matrix 转换后的旋转矩阵结果
* @return ERR_SUCC 成功 其他失败
*/
int quaternion_to_rot_matrix(ref int handle, ref JKTYPE.Quaternion quaternion, ref
JKTYPE.RotMatrix rot_matrix);
```

4.7.6 旋转矩阵到四元数的转换

```
/**
* @brief 旋转矩阵到四元数的转换
```

```
3.  * @param handle  机械臂控制句柄
4.  * @param rot_matrix 待转换的旋转矩阵
5.  * @param quaternion 转换后的四元数结果
6.  * @return ERR_SUCC 成功 其他失败
7.  */
8.  int rot_matrix_to_quaternion(ref int handle, ref JKTYPE.RotMatrix rot_matrix, ref
    JKTYPE.Quaternion quaternion);
```

4.8 机械臂伺服运动

4.8.1 机械臂伺服位置控制模式使能

```
1.  /**
2.  * @brief 机械臂伺服位置控制模式使能
3.  * @param handle  机械臂控制句柄
4.  * @param enable  TRUE 为进入伺服位置控制模式，FALSE 表示退出该模式
5.  * @return ERR_SUCC 成功 其他失败
6.  */
7.  int servo_move_enable(ref int handle, bool enable);
```

4.8.2 机械臂关节空间伺服模式运动

```
1.  /**
2.  * @brief 机械臂关节空间位置控制模式
3.  * @param handle  机械臂控制句柄
4.  * @param joint_pos 机械臂关节运动目标位置
5.  * @param move_mode 指定运动模式：绝对运动或相对运动
6.  * @return ERR_SUCC 成功 其他失败
7.  */
8.  int servo_j(ref int handle, ref JKTYPE.JointValue joint_pos, JKTYPE.MoveMode move_mode);
```

4.8.3 机械臂关节空间伺服模式运动扩展

```
1.  /**
2.  * @brief 机械臂关节空间位置控制模式
3.  * @param handle  机械臂控制句柄
4.  * @param joint_pos 机械臂关节运动目标位置
5.  * @move_mode 指定运动模式：增量运动或绝对运动
6.  * @step_num  倍分周期，servo_j 运动周期为 step_num*8ms，其中 step_num>=1
7.  * @return ERR_SUCC 成功 其他失败
8.  */
9.  int  servo_j_extend(ref int handle, ref JKTYPE.JointValue joint_pos, JKTYPE.MoveMode move_
    mode, int step_num);
```

4.8.4 机械臂笛卡尔空间伺服模式运动

```
/**
* @brief 机械臂笛卡尔空间位置控制模式
* @param handle  机械臂控制句柄
* @param cartesian_pose 机械臂笛卡尔空间运动目标位置
* @param move_mode 指定运动模式：ABS 代表绝对运动，INCR 代表相对运动
* @return ERR_SUCC 成功 其他失败
*/
int servo_p(ref int handle, ref JKTYPE.CartesianPose cartesian_pose, JKTYPE.MoveMode move_mode);
```

4.8.5 机械臂笛卡尔空间伺服模式运动扩展

```
/**
* @brief 机械臂笛卡尔空间位置控制模式
* @param handle  机械臂控制句柄
* @param cartesian_pose 机械臂笛卡尔空间运动目标位置
* @move_mode 指定运动模式：增量运动或绝对运动
* @step_num  倍分周期，servo_p 运动周期为 step_num*8ms，其中 step_num>=1
* @return ERR_SUCC 成功 其他失败
*/
int  servo_p_extend(ref int handle, ref JKTYPE.CartesianPose cartesian_pose, JKTYPE.MoveMode move_mode, int step_num);
```

4.8.6 机械臂 SERVO 模式下禁用滤波器

```
/**
* @brief SERVO 模式下不使用滤波器,该指令在 SERVO 模式下不可设置，退出 SERVO 后可设置
* @return ERR_SUCC 成功 其他失败
*/
int  servo_move_use_none_filter(ref int handle);
```

代码示例：

```
  /// <summary>
/// 禁用滤波器
/// </summary>
static void example_servo_use_none_filter()
{
    int ret;
    int handle = 0;
```

```
    //实例化机械臂，将 ip 切换为自己的 ip
    int result = jakaAPI.create_handler("192.168.2.160".ToCharArray(), ref handle);
    //机械臂上电
    jakaAPI.power_on(ref handle);
    //机械臂上使能
    jakaAPI.enable_robot(ref handle);
    ret = jakaAPI.servo_move_use_none_filter(ref handle);
}
```

4.8.7 机械臂 SERVO 模式下关节空间一阶低通滤波

```
/**
* @brief SERVO 模式下关节空间一阶低通滤波,该指令在 SERVO 模式下不可设置，退出 SERVO 后可设置
* @param cutoffFreq 一阶低通滤波器截止频率，默认 0.5hz
* @return ERR_SUCC 成功 其他失败
*/
int  servo_move_use_joint_LPF(ref int handle, double cutoffFreq);
```

代码示例：

```
/// <summary>
/// 关节一阶低通滤波
/// </summary>
static void example_servo_use_joint_LPF()
{
    int ret;
    int handle = 0;
    //实例化机械臂，将 ip 切换为自己的 ip
    int result = jakaAPI.create_handler("192.168.2.160".ToCharArray(), ref handle);
    //机械臂上电
    jakaAPI.power_on(ref handle);
    //机械臂上使能
    jakaAPI.enable_robot(ref handle);
    //servo 模式下关节空间一阶低通滤波,截止频率 0.5Hz
    ret = jakaAPI.servo_move_use_joint_LPF(ref handle, 0.5);
}
```

4.8.8 机械臂 SERVO 模式下关节空间非线性滤波

```
/**
* @brief SERVO 模式下关节空间非线性滤波,该指令在 SERVO 模式下不可设置，退出 SERVO 后可设置
* @param max_vr 笛卡尔空间姿态变化速度的速度上限值（绝对值）°/s
* @param max_ar 笛卡尔空间姿态变化速度的加速度上限值（绝对值）°/s^2
* @param max_jr 笛卡尔空间姿态变化速度的加加速度上限值（绝对值）°/s^3
```

```
* @return ERR_SUCC 成功 其他失败
*/
int  servo_move_use_joint_NLF(ref int handle, double max_vr, double max_ar, double max_jr);
```

代码示例：

```
/// <summary>
/// 关节非线性
/// </summary>
static void example_servo_use_joint_NLF()
{
    int ret;
    int handle = 0;
    //实例化机械臂，将 ip 切换为自己的 ip
    int result = jakaAPI.create_handler("192.168.2.160".ToCharArray(), ref handle);
    //机械臂上电
    jakaAPI.power_on(ref handle);
    //机械臂上使能
    jakaAPI.enable_robot(ref handle);
    //servo 模式下关节空间非线性滤波
    ret = jakaAPI.servo_move_use_joint_NLF(ref handle, 2, 2, 4);
}
```

4.8.9 机械臂 SERVO 模式下笛卡尔空间非线性滤波

```
/**
* @brief SERVO 模式下笛卡尔空间非线性滤波,该指令在 SERVO 模式下不可设置，退出 SERVO 后可设置
* @param max_vp 笛卡尔空间下移动指令速度的上限值（绝对值）。单位：mm/s
* @param max_ap 笛卡尔空间下移动指令加速度的上限值（绝对值）。单位：mm/s^2
* @param max_jp 笛卡尔空间下移动指令加加速度的上限值（绝对值）单位：mm/s^3
* @param max_vr 笛卡尔空间姿态变化速度的速度上限值（绝对值）°/s
* @param max_ar 笛卡尔空间姿态变化速度的加速度上限值（绝对值）°/s^2
* @param max_jr 笛卡尔空间姿态变化速度的加加速度上限值（绝对值）°/s^3
* @return ERR_SUCC 成功 其他失败
*/
int  servo_move_use_carte_NLF(ref int handle, double max_vp, double max_ap, double max_jp,
    double max_vr, double max_ar, double max_jr);
```

代码示例：

```
/// <summary>
/// 笛卡尔空间非线性滤波
/// </summary>
static void example_servo_use_carte_NLF()
{
    int handle = 0;
```

```
    int ret = 0;
    //实例化机械臂，将 ip 切换为自己的 ip
    int result = jakaAPI.create_handler("192.168.2.160".ToCharArray(), ref handle);
    //机械臂上电
    jakaAPI.power_on(ref handle);
    //机械臂上使能
    jakaAPI.enable_robot(ref handle);
    //servo 模式下笛卡尔空间非线性滤波
    ret = jakaAPI.servo_move_use_carte_NLF(ref handle, 2, 2, 4, 2, 2, 4);
}
```

4.8.10 机械臂 SERVO 模式下关节空间多阶均值滤波

```
/**
* @brief SERVO 模式下关节空间多阶均值滤波器,该指令在 SERVO 模式下不可设置，退出 SERVO 后可设置
* @param handle  机械臂控制句柄
* @param max_buf 均值滤波器缓冲区的大小
* @param kp 加速度滤波系数
* @param kv 速度滤波系数
* @param ka 位置滤波系数
* @return ERR_SUCC 成功 其他失败
*/
int servo_speed_foresight(ref int i, int max_buf, double kp);
```

代码示例：

```
/// <summary>
/// 关节空间多阶均值滤波
/// </summary>
static void example_servo_use_joint_MMF()
{
    int handle = 0;
    int ret = 0;
    //实例化机械臂，将 ip 切换为自己的 ip
    int result = jakaAPI.create_handler("192.168.2.160".ToCharArray(), ref handle);
    //机械臂上电
    jakaAPI.power_on(ref handle);
    //机械臂上使能
    jakaAPI.enable_robot(ref handle);
    //servo 模式下关节空间多阶均值滤波
    ret = jakaAPI.servo_move_use_joint_MMF(ref handle, 200, 2, 4, 2);
}
```

4.8.11 机械臂 SERVO 模式速度前瞻参数设置

```
/**
* @brief SERVO 模式下速度前瞻参数设置
* @param handle  机械臂控制句柄
* @param max_buf 缓冲区的大小
* @param kp 加速度滤波系数
* @return ERR_SUCC 成功 其他失败
*/
int servo_speed_foresight(ref int i, int max_buf, double kp);
```

代码示例：

```
/// <summary>
/// 速度前瞻
/// </summary>
static void example_speed_foresight()
{
    int handle = 0;
    int ret = 0;
    //实例化机械臂，将 ip 切换为自己的 ip
    int result = jakaAPI.create_handler("192.168.2.160".ToCharArray(), ref handle);
    //机械臂上电
    jakaAPI.power_on(ref handle);
    //机械臂上使能
    jakaAPI.enable_robot(ref handle);
    //servo 模式下速度前瞻
    ret = jakaAPI.servo_speed_foresight(ref handle, 200, 2);
}
```

4.9 力控机器人扩展

需要额外在工具末端安装力控传感器

4.9.1 设置传感器品牌

```
/**
* @brief 设置传感器品牌
* @param sensor_brand 传感器品牌，可选值为 1,2,3 分别代表不同品牌力矩传感器
* @return ERR_SUCC 成功 其他失败
*/
int set_torsenosr_brand(ref int handle, int sensor_brand);
```

4.9.2 获取传感器品牌

```
/**
* @brief 获取当前设置的传感器品牌
* @param sensor_brand 传感器品牌，可选值为1,2,3 分别代表不同品牌力矩传感器
* @return ERR_SUCC 成功 其他失败
*/
int get_torsenosr_brand(ref int handle, ref int sensor_brand);
```

4.9.3 开启或关闭力矩传感器

```
/**
* @brief 开启或关闭力矩传感器
* @param sensor_mode 0代表关闭传感器，1代表开启力矩传感器
* @return ERR_SUCC 成功 其他失败
*/
int set_torque_sensor_mode(ref int handle, int sensor_mode);
```

4.9.4 设置柔顺控制参数

```
/**
* @brief 设置柔顺控制参数
* @param axis 代表配置哪一轴，可选值为0~5
* @param opt 柔顺方向，可选值为 1 2 3 4 5 6分别对应 fx fy fz mx my mz 0代表没有勾选
* @param ftUser 阻尼力，表示用户用多大的力才能让机械臂的沿着某个方向以最大速度进行运动
* @param ftConstant 代表恒力，手动操作时全部设置为0
* @param ftNnormalTrack 法向跟踪，手动操作时全部设置为0,
* @param ftReboundFK 回弹力，表示机械臂回到初始状态的能力
* @return ERR_SUCC 成功 其他失败
*/
public static extern int set_admit_ctrl_config(ref int i, int axis, int opt, double ftUser,
double ftConstant, int ftNnormalTrack, double ftReboundFK);
```

4.9.5 开始辨识工具末端负载

```
/**
* @brief 开始辨识工具末端负载
* @param joint_pos 使用力矩传感器进行自动负载辨识时的结束位置
* @return ERR_SUCC 成功 其他失败
*/
int start_torq_sensor_payload_identify(ref int handle, ref JointValue joint_pos);
```

代码示例：

```
/// <summary>
/// 力控传感器（额外配件）辨识负载
/// </summary>
static void example_sensor_payload()
{
    Console.WriteLine("sensor in\n");
    JKTYPE.JointValue joint_pos= new JKTYPE.JointValue();
    JKTYPE.PayLoad pl = new JKTYPE.PayLoad(), pl_ret= new JKTYPE.PayLoad();
    int handle = 0;
    int ret = 0;
    //实例化机械臂，将 ip 切换为自己的 ip
    Console.WriteLine("login be\n");
    int result = jakaAPI.create_handler("10.5.5.100".ToCharArray(), ref handle);
    Console.WriteLine("login finish\n");
    //机械臂上电
    jakaAPI.power_on(ref handle);
    //机械臂上使能
    jakaAPI.enable_robot(ref handle);
    Console.WriteLine("enable finish\n");
    ret = jakaAPI.set_torsenosr_brand(ref handle, 2);//设置传感器品牌
    ret = jakaAPI.set_torque_sensor_mode(ref handle, 1);//开启力控传感器
    jakaAPI.get_joint_position(ref handle, ref joint_pos);
    joint_pos.jVal[3] += PI / 4;
    if (joint_pos.jVal[3] > 265 * PI / 180)
        joint_pos.jVal[3] -= PI / 2;
    joint_pos.jVal[4] += PI / 4;
    if (joint_pos.jVal[4] > 320 * PI / 180)
        joint_pos.jVal[4] -= PI / 2;
    joint_pos.jVal[5] += PI / 4;
    if (joint_pos.jVal[5] > 265 * PI / 180)
        joint_pos.jVal[5] -= PI / 2;
    //开始辨识传感器负载
    ret = jakaAPI.start_torq_sensor_payload_identify(ref handle, ref joint_pos);
    do
    {
        //查询传感器负载状态
        jakaAPI.get_torq_sensor_identify_staus(ref handle, ref ret);
        Console.WriteLine("{0}", ret);
        System.Threading.Thread.Sleep(1000);
    } while (1 == ret);
    //获取辨识结果
    ret = jakaAPI.get_torq_sensor_payload_identify_result(ref handle, ref pl);
    //设置传感器末端负载
    Console.WriteLine("payload: {0} {1} {2} {3}", pl.mass, pl.centroid.x, pl.centroid.y, p
l.centroid.z);
```

```
    ret = jakaAPI.set_torq_sensor_tool_payload(ref handle, ref pl);
    System.Threading.Thread.Sleep(10000);
    //获取当前设置的传感器末端负载
    ret = jakaAPI.get_torq_sensor_tool_payload(ref handle, ref pl_ret);
    System.Threading.Thread.Sleep(10000);
    jakaAPI.destory_handler(ref handle);
}
```

```
/**
```

4.9.6 获取末端负载辨识状态

```
 * @brief 获取末端负载辨识状态
 * @param identify_status 0 代表辨识完成，1 代表未完成，2 代表辨识失败
 * @return ERR_SUCC 成功 其他失败
 */
int get_torq_sensor_identify_staus(ref int handle, ref int identify_status);
```

```
/**
```

4.9.7 获取末端负载辨识结果

```
 * @brief 获取末端负载辨识结果
 * @param payload 末端负载
 * @return ERR_SUCC 成功 其他失败
 */
int get_torq_sensor_payload_identify_result(ref int handle, ref PayLoad payload);
```

4.9.8 设置传感器末端负载

```
/**
 * @brief 设置传感器末端负载
 * @param payload 末端负载
 * @return ERR_SUCC 成功 其他失败
 */
int set_torq_sensor_tool_payload(ref int handle, ref PayLoad payload);
```

4.9.9 获取传感器末端负载

```
/**
 * @brief 获取传感器末端负载
 * @param payload 末端负载
```

```
4.  * @return ERR_SUCC 成功 其他失败
5.  */
6.  int get_torq_sensor_tool_payload(ref int handle, ref PayLoad payload);
```

4.9.10 力控拖拽使能（导纳控制）

```
1.  /**
2.  * @brief 力控拖拽使能
3.  * @param enable_flag 0 为关闭力控拖拽使能，1 为开启
4.  * @return ERR_SUCC 成功 其他失败
5.  */
6.  int enable_admittance_ctrl(ref int handle, int enable_flag);
```

代码示例：

```
1.  /// <summary>
2.  /// 力控传感器（额外配件）导纳控制，z 轴
3.  /// </summary>
4.  static void example_enable_admittance_ctrl()
5.  {
6.      int handle = 0;
7.      int ret;
8.      //实例化机械臂，将 ip 切换为自己的 ip
9.      int result = jakaAPI.create_handler("10.5.5.100".ToCharArray(), ref handle);
10.     //机械臂上电
11.     jakaAPI.power_on(ref handle);
12.     //机械臂上使能
13.     jakaAPI.enable_robot(ref handle);
14.     //品牌,力控传感器品牌询问服务人员
15.     jakaAPI.set_torsenosr_brand(ref handle, 2);
16.     //开启力控传感器
17.     jakaAPI.set_torque_sensor_mode(ref handle, 1);
18.     //初始化传感器
19.     jakaAPI.set_compliant_type(ref handle, 1, 1);
20.     Console.WriteLine("inint sensor comple\n");
21.     //设置柔顺控制参数，z 轴向下 5N 恒力
22.     ret = jakaAPI.set_admit_ctrl_config(ref handle, 0, 0, 20, 5, 0, 0);//x 轴
23.     ret = jakaAPI.set_admit_ctrl_config(ref handle, 1, 0, 20, 5, 0, 0);//y 轴
24.     ret = jakaAPI.set_admit_ctrl_config(ref handle, 2, 3, 20, 5, 0, 0);//z 轴
25.     ret = jakaAPI.set_admit_ctrl_config(ref handle, 3, 0, 20, 5, 0, 0);//rx
26.     ret = jakaAPI.set_admit_ctrl_config(ref handle, 4, 0, 20, 5, 0, 0);//ry
27.     ret = jakaAPI.set_admit_ctrl_config(ref handle, 5, 0, 20, 5, 0, 0);//rz
28.     //设置力控拖拽使能，1 打开，0 关闭
29.     ret = jakaAPI.enable_admittance_ctrl(ref handle, 1);
30.     Console.WriteLine("enable_admittance_ctrl open! \n");
31.     Console.WriteLine("input any word to quit:\n");
```

```
    ret = Console.Read();
    Console.WriteLine("quit!!");
    ret = jakaAPI.enable_admittance_ctrl(ref handle, 0);
    ret = jakaAPI.set_admit_ctrl_config(ref handle, 2, 0, 20, 5, 0, 0);
    jakaAPI.set_torque_sensor_mode(ref handle, 0);
    Console.WriteLine("close\n");
    jakaAPI.destory_handler(ref handle);
}
```

4.9.11 设置力控类型和传感器初始化状态

```
/**
* @brief 设置力控类型和传感器初始化状态
* @param sensor_compensation 是否开启传感器补偿,1 代表开启即初始化,0 代表不初始化
* @param compliance_type 0 代表不使用任何一种柔顺控制方法 1 代表恒力柔顺控制,2 代表速度柔顺控制
* @return ERR_SUCC 成功 其他失败
*/
int set_compliant_type(ref int handle, int sensor_compensation, int compliance_type);
```

4.9.12 获取力控类型和传感器初始化状态

```
/**
* @brief 获取力控类型和传感器初始化状态
* @param sensor_compensation 是否开启传感器补偿,1 代表开启即初始化,0 代表不初始化
* @param compliance_type 0 代表不使用任何一种柔顺控制方法 1 代表恒力柔顺控制,2 代表速度柔顺控制
* @return ERR_SUCC 成功 其他失败
*/
int get_compliant_type(ref int handle, ref int sensor_compensation, ref int compliance_type);
```

4.9.13 获取力控柔顺控制参数

```
/**
* @brief 获取力控柔顺控制参数
* @param handle  机械臂控制句柄
* @param admit_ctrl_cfg 机械臂力控柔顺控制参数存储地址
* @return ERR_SUCC 成功 其他失败
*/
int get_admit_ctrl_config(ref int i, ref JKTYPE.RobotAdmitCtrl admit_ctrl_cfg);
```

4.9.14 设置力控柔顺控制参数

```
/**
* @brief 设置柔顺控制参数
```

```
3.  * @param axis 代表配置哪一轴，可选值为 0~5 柔顺方向，分别对应 fx fy fz mx my mz
4.  * @param opt  0 代表没有勾选，非 0 值代表勾选
5.  * @param ftUser 阻尼力，表示用户用多大的力才能让机械臂的沿着某个方向以最大速度进行运动
6.  * @param ftConstant 代表恒力，手动操作时全部设置为 0
7.  * @param ftNnormalTrack 法向跟踪，手动操作时全部设置为 0,
8.  * @param ftReboundFK 回弹力，表示机械臂回到初始状态的能力
9.  * @return ERR_SUCC 成功 其他失败
10. */
11. int set_admit_ctrl_config(ref int i, int axis, int opt, double ftUser, double ftConstant,
    int ftNnormalTrack, double ftReboundFK);
```

4.9.15 设置力控传感器通信参数

```
1.  /**
2.  * @brief 设置力控传感器通信参数
3.  * @param handle  机械臂控制句柄
4.  * @param type 0 为使用 tcp/ip 协议，1 为使用 RS485 协议
5.  * @param ip_addr 为力控传感器地址
6.  * @param port 为使用 tcp/ip 协议时力控传感器端口号
7.  * @return ERR_SUCC 成功 其他失败
8.  */
9.  int set_torque_sensor_comm(ref int i, int type, ref char[] ip_addr, int port);
```

4.9.16 获取力控传感器通信参数

```
1.  /**
2.  * @brief 获取力控传感器通信参数
3.  * @param handle  机械臂控制句柄
4.  * @param ip_addr 为力控传感器地址
5.  * @return ERR_SUCC 成功 其他失败
6.  */
7.  int get_torque_sensor_comm(ref int i, ref int type, ref byte ip_addr, ref int port);
```

在 c#中使用时，需要将 byte 转为 char：

```
string strGet = System.Text.Encoding.Default.GetString(ip_addr, 0, ip_addr.length())
;
```

4.9.17 关闭力控

```
1.  /**
2.  * @brief 关闭力控
3.  * @param handle  机械臂控制句柄
4.  * @return ERR_SUCC 成功 其他失败
5.  */
6.  int disable_force_control(ref int i);
```

4.9.18 设置速度柔顺控制参数

```
/**
* @brief 设置速度柔顺控制参数
* @param handle  机械臂控制句柄
* @param vel_cfg 为速度柔顺控制参数
* @return ERR_SUCC 成功 其他失败
*/
int set_vel_compliant_ctrl(ref int i, ref JKTYPE.VelCom vel);
```

4.9.19 设置柔顺控制力矩条件

```
/**
* @brief 设置柔顺控制力矩条件
* @param handle  机械臂控制句柄
* @param ft 为柔顺控制力矩条件
* @return ERR_SUCC 成功 其他失败
*/
int set_compliance_condition(ref int i, ref JKTYPE.FTxyz ft);
```

4.9.20 设置力控的低通滤波器参数

```
/**
* @brief 设置力控的低通滤波器的值
* @param torque_sensor_filter 低通滤波器的值,单位：Hz
*/
errno_t set_torque_sensor_filter(ref int handle, float torque_sensor_filter);
```

4.9.21 获取力控的低通滤波器参数

```
/**
* @brief 获取力控的低通滤波器的值
* @param torque_sensor_filter 低通滤波器的值,单位：Hz
*/
errno_t get_torque_sensor_filter(ref int handle, float *torque_sensor_filter);
```

4.9.22 设置力传感器的传感器限位参数配置

```
/**
* @brief 设置力传感器的传感器限位参数配置
* @param torque_sensor_soft_limit 力传感器的传感器限位参数
*        力限位 fx, fy, fz 单位：N
```

```
5.  *         力矩限位 tx, ty, tz 单位：N*m
6.  */
7.  errno_t set_torque_sensor_soft_limit(ref int
    handle, jkTYPE.FTxyz torque_sensor_soft_limit);
```

4.9.23 获取力传感器的传感器限位参数配置

```
1.  /**
2.  * @brief 获取力传感器的传感器限位参数配置
3.  * @param torque_sensor_soft_limit 力传感器的传感器限位参数
4.  *         力限位 fx, fy, fz 单位：N
5.  *         力矩限位 tx, ty, tz 单位：N*m
6.  */
7.  errno_t get_torque_sensor_soft_limit(ref int
    handle, jkTYPE.FTxyz *torque_sensor_soft_limit);
```

4.10 FTP 服务

4.10.1 初始化 FTP 客户端

```
1.          /**
2.          * @brief 初始化 ftp 客户端，与控制柜建立连接，可导出 program、track
3.          * @return ERR_SUCC 成功 其他失败
4.          */
5.          errno_t init_ftp_client(ref int i);
```

4.10.2 FTP 上传

```
1.          /**
2.          * @brief 从本地上传指定类型和名称的文件到控制器
3.          * @param remote 上传到控制器内部文件名绝对路径，若为文件夹需要以“\”或“/”结尾
4.          * @param local 本地文件名绝对路径
5.          * @param opt 1 单个文件 2 文件夹
6.          * @return ERR_SUCC 成功 其他失败
7.          */
8.          errno_t upload_file(ref int i,char[] local, char[] remote, int opt);
```

4.10.3 FTP 下载

```
1.          /**
2.          * @brief 从控制器下载指定类型和名称的文件到本地
3.          * @param remote 控制器内部文件名绝对路径，若为文件夹需要以“\”或“/”结尾
```

```
4.          * @param local 下载到本地文件名绝对路径
5.          * @param opt 1 单个文件 2 文件夹
6.          * @return ERR_SUCC 成功 其他失败
7.          */
8.          errno_t download_file(ref int i,char[] local, char[] remote, int opt);
```

4.10.4 FTP 目录查询

```
1.          /**
2.          * @brief 查询控制器指定目录下的结构
3.          * @param remote 控制器内部文件名原名称，查询轨迹“/track/”,查询脚本程序“/program/”
4.          * @param type 0 文件名和子目录名，1 文件名，2 子目录名
5.          * @param ret 查询结果
6.          * @return ERR_SUCC 成功 其他失败
7.          */
8.          errno_t get_ftp_dir(ref int i,const char[] remotedir, int type, StringBuilder ret);
```

```
1.  int handle = 0;
2.  int result = jakaAPI.create_handler("192.168.137.173".ToCharArray(), ref handle);
3.
4.  string remote_get_dir = "log";
5.  StringBuilder remote_get_res = new StringBuilder(4096);
6.  errno = jakaAPI.get_ftp_dir(ref handle, remote_get_dir, 0, remote_get_res);
7.  assert_0_or_exit(errno, "get ftp dir");
8.  Console.WriteLine("get_ftp_dir res: {0}", remote_get_res);
9.
10. jakaAPI.destory_handler(ref handle);
```

4.10.5 FTP 删除

```
1.          /**
2.          * @brief 从控制器删除指定类型和名称的文件
3.          * @param remote 控制器内部文件名
4.          * @param opt 1 单个文件 2 文件夹
5.          * @return ERR_SUCC 成功 其他失败
6.          */
7.          errno_t del_ftp_file(ref int i,char[] remote, int opt);
```

4.10.6 FTP 重命名

```
/**
* @brief 重命名控制器指定类型和名称的文件
* @param remote 控制器内部文件名原名称
* @param des 重命名的目标名
* @param opt 1 单个文件 2 文件夹
* @return ERR_SUCC 成功 其他失败
*/
errno_t rename_ftp_file(ref int i,char[] remote, char[] des, int opt);
```

4.10.7 关闭 FTP 客户端

```
/**
* @brief 断开与控制器 ftp 链接
* @return ERR_SUCC 成功 其他失败
*/
public static extern int close_ftp_client(ref int i);
```

5.反馈与勘误

文档中若出现不准确的描述或者错误，恳请读者指正批评。如果您在阅读过程中发现任何问题或者有想提出的意见，可以发送邮件到 support@jaka.com ，我们的同事会尽量一一回复。