JAKA

节卡机器人

SDK 手册

【Python】

文档版本：V2.1.14

SDK 版本：V2.1.14

产权说明

【版本记录】

版本编号	版本日期	控制器支持版本	说明
V1.0.0	2020.3.24	V1.4.10/V2.0.10 及以上	创建文档
V1.0.0	2020.6.24	V1.4.10/V2.0.10 及以上	1.增加对 joint_move、linear_move、servo_j 等不同运动指令的补充说明
V1.0.14	2020.7.24	V1.4.10/V2.0.10 及以上	1.新增接口 4.61 到 4.64 2.对 jog_stop 定义进行修改
V1.0.15	2021.04.27	V1.4.24/V1.5.12.17/V2.0.24 及以上	新增接口 4.69 到 4.110
V1.0.18	2021.08.30	V1.4.24/V1.5.12.17/V2.0.24 及以上	增加 API 使用说明
V2.1.1	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
V2.1.3	2022.11.29	V1.4.24/V1.5.13.17/V2.0.24 及以上	圆弧运动圈数指定
V2.1.4	2023.06.14	V1.4.24/V1.5.13.17/V2.0.24 及以上	力矩传感器状态监测数据扩充
V2.1.7	2024.01.28	V1.4.24（部分)/V1.5.13.17/V2.0.24 及以上	1）ADD: 增加设置和获取安装接口 2）FIX: 修改圆弧运动加速度单位 3）FIX:set_status_data_update_time_interval()函数名称错误 4）FIX: upload_file、upload_file 描述错误
V2.1.8	2024.3.21	V1.5.13.08 及以上	1）ADD：增加带姿态速度和姿态加速度的直线运动接口 2）ADD：增加带圆弧中间点运动模式的圆弧运动接口
V2.1.11	2024.4.30	V1.5.13.08 及以上	1)ADD:增加获取日志路径接口(get_SDK_filepath, static_Get_SDK_filepath) 2）FIX: 完善 10004 通信端口 3）FIX: 修复 set_ft_ctrl_frame 接口无法使用的问题 4）FIX: 修复 10001 端口指令错误重发的情况
V2.1.14	2024.09.30	V1.7.1.40 及以上	Fix： 1、重新定义 10004 端口机制,解决之前 10004 导致的 SDK 崩溃问题 2、 重新定义 motion block 机制，解决之前运动 block 不住的问题 3、 解决 inpos 到位不准问题

版本编号	版本日期	控制器支持版本	说明
			Add: 1、 增加设置 和 获取系统变量接口 set_user_var()、 get_user_var() 2、 增加 get_motion_status 接口：获取运动相关状态信息 3、增加部分内嵌 S 相关接口 头文件： 1、 头文件注释全部更新为英文 2、 头文件前增加版权信息及版本信息

目录

1. 简介

本文档为 python 版本的二次开发接口文档。

2. 文档须知

- 运行环境：linux python3.5 32 位 、windows Python3.7 64 位。
- 使用到参数的单位：毫米(mm)，角度(rad)。
- **非特别说明的代码示例中都默认机器人已经开机，并且上电上使能。**
- 文档中的所有代码示例都默认在机器人的工作空间内没有任何干涉。
- 文档中的示例都已经默认用户的 Python 解释器能够找到 jkrc 模块。

2.1 Linux 下的使用

Linux 需要将 libjakaAPI.so 和 jkrc.so 放在同一个文件夹下，并添加当前文件夹路径到环境变量，
export LD_LIBRARY_PATH=/xx/xx/

2.2 Windows 下的使用

Windows 需要将 jkrc 以及 jakaAPI.dll 放在同一个文件夹下，常见问题可以查询 FAQ。

2.3 动态库版本号查询方法

jkrc 的正常使用需要包含动态库文件，以下是动态库版本号的查询方法：

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。

3. 数据类型

3.1 IO 类型

JAKA 机器人有三种 IO ，分别是控制柜 IO，工具 IO，扩展 IO。

以下定义在后面的示例代码中会使用到：

```
IO_CABINET, # 控制柜面板 IO
IO_TOOL,     #工具 IO
IO_EXTEND,   # 扩展 IO
IO_REALY,    # 继电器 IO，目前仅 CAB V3 支持 DO
IO_MODBUS_SLAVE,    # Modbus 从站 IO,从 0 索引
IO_PROFINET_SLAVE, #Profinet 从站 IO,从 0 索引
IO_EIP_SLAVE         # ETHRENET/IP 从站 IO,从 0 索引
```

3.2 坐标系类型

JAKA 机器人有三种坐标系，分别是基坐标系/当前的用户坐标系，关节空间，工具坐标系
以下定义在后面的示例代码中会使用到：

```
COORD_BASE = 0     # 世界或者当前的用户坐标系
COORD_JOINT = 1    # 关节空间
COORD_TOOL = 2     #工具坐标系
```

3.3 运动类型

JAKA 机器人运动类型有两种，定义如下：

```
ABS = 0      # 绝对运动
INCR = 1     # 增量运动
CONTINUE = 2  #连续运动
```

3.4 函数返回值

```
1.  #define ERR_SUCC                  0              //success
2.  #define ERR_FUCTION_CALL_ERROR   2              //interface error or controller not support
3.  #define ERR_INVALID_HANDLER        -1          //invalid handler
4.  #define  ERR_INVALID_PARAMETER    -2           //invalid parameter
5.  #define ERR_COMMUNICATION_ERR  -3              //fail to connect
6.  #define ERR_KINE_INVERSE_ERR     -4             //kine_inverse error
7.  #define ERR_EMERGENCY_PRESSED    -5             //e-stop
8.  #define ERR_NOT_POWERED          -6             //not power on
```

```
#define ERR_NOT_ENABLED          -7             //not enable
#define ERR_DISABLE_SERVOMODE    -8             //not in servo mode
#define ERR_NOT_OFF_ENABLE       -9             //must turn off enable before power off
#define ERR_PROGRAM_IS_RUNNING   -10            //cannot operate, program is running
#define ERR_CANNOT_OPEN_FILE     -11            //cannot open file, or file doesn't exist
#define ERR_MOTION_ABNORMAL      -12            //motion abnormal
#define ERR_FTP_PERFORM            -14          //ftp error
#define ERR_VALUE_OVERSIZE       -15            //socket msg or value oversize
#define ERR_KINE_FORWARD         -16            //kine_forward error
#define ERR_EMPTY_FOLDER         -17            //not support empty folder
#define ERR_PROTECTIVE_STOP      -20            // protective stop
#define ERR_EMERGENCY_STOP      -21             // protective stop
#define ERR_SOFT_LIMIT          -22             // on soft limit
#define ERR_CMD_ENCODE          -30             // fail to encode cmd string
#define ERR_CMD_DECODE          -31             // fail to decode cmd string
#define ERR_UNCOMPRESS          -32             // fail to uncompress port 10004 string
#define ERR_MOVEL               -40             // move linear error
#define ERR_MOVEJ               -41             // move joint error
#define ERR_MOVEC               -42             // move circular error
#define ERR_MOTION_TIMEOUT      -50             // block_wait timeout
#define ERR_POWERON_TIMEOUT      -51            // power on timeout
#define ERR_POWEROFF_TIMEOUT      -52           // power off timeout
#define ERR_ENABLE_TIMEOUT      -53             // enable timeoff
#define ERR_DISABLE_TIMEOUT      -54            // disable timeout
#define ERR_USERFRAME_SET_TIMEOUT      -55      // set userframe timeout
#define ERR_TOOL_SET_TIMEOUT      -56           // set tool timeout
#define ERR_IO_SET_TIMEOUT      -60             // set io timeout
```

除了实例化机器人对象之外的其它接口的返回值都是一个元组。Get 类的接口的返回值形式为 **(errcode,data)** ，第一个元素是 errcode 错误码，成功调用 errcode 为 0，如果发生错误可能得到的错误码值可以查询上表。第二个元素是 data，如关节角度值等等。

例如使用 **get_joint_position()** 获取关节角度时：
成功获取关节角度: (0,[1,2,3,4,5,6])
因为通信错误，无法获得关节角度时：(-3,)

4. 接口

4.1 机械臂基础

4.1.1 实例化机器人

函数	RC(ip)
描述	实例化一个机器人对象
参数	ip: 机器人的 IP 地址，需要填入一个字符串只有正确的 IP 地址实例化出的对象才能控制机器人。
返回值	成功：返回一个机器人对象 失败：创建的对象会被销毁

代码示例：

```
import jkrc
robot = jkrc.RC("192.168.2.64") #返回一个机器人对象
```

4.1.2 机器人登录

函数	login()
描述	连接机器人控制器
参数	
返回值	成功：(0,) 失败：其它

4.1.3 机器人注销

函数	logout()
描述	断开控制器连接
参数	
返回值	成功：(0,) 失败：其它

代码示例：

```
import jkrc
robot = jkrc.RC("192.168.2.64")#返回一个机器人对象
robot.login()  #登录
pass
robot.logout() #登出
```

4.1.4 打开机器人电源

函数	power_on()
描述	打开机器人电源，给真实的机器人上电大概会有 8 秒钟左右的延迟
参数	
返回值	成功：(0,) 失败：其它

代码示例：

```
import jkrc
robot = jkrc.RC("192.168.2.64")    #返回一个机器人对象
robot.login()      #登录
robot.power_on() #上电
robot.logout()     #登出
```

4.1.5 关闭机器人电源

函数	power_off()
描述	关闭机器人电源
参数	
返回值	成功：(0,) 失败：其它

4.1.6 机器人控制柜关机

函数	shut_down()
描述	机器人控制柜关机
参数	
返回值	成功：(0,) 失败：其它

4.1.7 控制机器人上使能

函数	enable_robot()
描述	控制机器人上使能
参数	
返回值	成功：(0,) 失败：其它

代码示例：

```
import jkrc
robot = jkrc.RC("192.168.2.64")#返回一个机器人对象
robot.login()  #登录
robot.enable_robot()#
robot.logout()  #登出
```

4.1.8 控制机器人下使能

函数	disable_robot()
描述	控制机器人下使能
参数	
返回值	成功：(0,) 失败：其它

4.1.9 查询 SDK 版本号

函数	get_sdk_version()
描述	获取 SDK 版本号
参数	
返回值	成功：(0,version) 失败：其它

代码示例：

```
'''查询 sdk 版本号'''
import jkrc
robot = jkrc.RC("192.168.2.64")#
robot.login()  #
ret = robot.get_sdk_version()
print("SDK version  is:",ret[1])
robot.logout()  #登出
```

4.1.10 获取控制器 IP

函数	get_controller_ip ()
描述	获取控制器 IP
参数	
返回值	成功：(0, ip_list), 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 失败：其它

4.1.11 控制器机器人进入或退出拖拽模式

函数	drag_mode_enable(enable)
描述	控制器机器人进入或退出拖拽模式
参数	
返回值	成功：(0,) 失败：其它

代码示例：

```
import jkrc
import time
#坐标系
COORD_BASE  = 0
COORD_JOINT = 1
COORD_TOOL  = 2
#运动模式
ABS = 0
INCR= 1
robot = jkrc.RC("192.168.2.160")
robot.login()
robot.power_on()
robot.enable_robot()
robot.drag_mode_enable(True)
ret = robot.is_in_drag_mode()
print(ret)
a = input()
robot.drag_mode_enable(False)
ret = robot.is_in_drag_mode()
print(ret)
robot.logout()
```

4.1.12 查询机器人是否处于拖拽模式

函数	is_in_drag_mode()
描述	查询机器人是否处于拖拽模式
参数	
返回值	成功：(0,state)　state 为 1 代表机器人处于拖拽模式，0 则相反 失败：其它

4.1.13 设置 SDK 是否开启调试模式

函数	**set_debug_mode(mode)**
描述	设置 SDK 是否开启调试模式。
参数	mode: 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
返回值	成功：(0,) 失败：其它

4.1.14 设置 SDK 日志路径

函数	**set_SDK_filepath(filepath)**
描述	设置 SDK 日志路径
参数	filepath：文件路径
返回值	成功：(0,) 失败：其它

4.2 机械臂运动

由控制器参与进行规划的运动

4.2.1 控制机器人手动模式下运动

函数	**jog(aj_num , move_mode, coord_type, jog_vel, pos_cmd)**
描述	控制机器人手动模式下运动
参数	aj_num：axis_joint_based 标识值，在关节空间下代表轴号，1 轴到六轴的轴号分别对应数字 0 到 5，笛卡尔空间下依次为 x，y，z，rx，ry，rz 分别对应数字 0 到 5 move_mode：0 代表绝对运动，1 代表增量运动，2 代表连续运动 coord_type：机器人运动坐标系，工具坐标系，基坐标系（当前的世界/用户坐标系）或关节空间 jog_vel：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，速度的正负决定运动方向的正负。 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，当 move_mdoe 是绝对运动时参数可忽略
返回值	成功：(0,) 失败：其它

代码示例：

'''jog 运动'''

'''1.关节空间 jog'''

```python
# -*- coding: utf-8 -*-
import sys
sys.path.append('D:\\vs2019ws\PythonCtt\PythonCtt')
import time
import jkrc

#坐标系
COORD_BASE  = 0
COORD_JOINT = 1
COORD_TOOL  = 2
#运动模式
ABS = 0
INCR= 1
#关节 1~6 依次对应 0~5,

robot = jkrc.RC("192.168.2.194")#返回机器人对象
robot.login()#登录
robot.power_on() #上电
robot.enable_robot()
print("move1")
robot.jog(0,INCR,COORD_JOINT,30, 90)
time.sleep(5)#jog 为非阻塞指令，运动状态下接收 jog 指令会被丢弃
print("move2")
robot.jog(0,INCR,COORD_JOINT,5,-90)
time.sleep(3)
robot.jog_stop()
robot.logout()
```

'''2.笛卡尔空间 jog'''

```python
import jkrc
import time
COORD_BASE = 0    # 基坐标系
COORD_JOINT = 1   # 关节空间
COORD_TOOL = 2    #工具坐标系
ABS = 0             # 绝对运动
INCR = 1            # 增量运动
cart_x = 0          #x 方向
cart_y = 1          #y 方向
cart_z = 2          #z 方向
cart_rx = 3         #rx 方向
cart_ry = 4         #ry 方向
```

```
cart_rz = 5          #rz 方向
robot = jkrc.RC("192.168.2.64")#返回一个机器人对象
robot.login()      #登录
robot.jog(aj_num = cart_z ,move_mode = INCR,coord_type = COORD_BASE ,jog_vel = 5,pos_cmd = 10)   # z 正方向运动 10mm
robot.jog_stop()
robot.logout()      #登出
```

说明：

若机器人已经接近奇异姿态或达到关节限位，使用上述示例代码，将无法 jog 机器人

4.2.2 控制机器人手动模式下运动停止

函数	jog_stop(joint_num)
描述	控制机器人手动模式下运动停止，用于停止 jog
参数	joint_num: num 代表要停止运动的关节轴号，轴号 0 到 5，分别代表关节 1 到关节 6。值的注意的是**-1** 代表停止所有轴的运动。
返回值	成功：(0,) 失败：其它

代码示例：

```
# -*- coding: utf-8 -*-
import sys
sys.path.append('D:\\vs2019ws\PythonCtt\PythonCtt')
import time
import jkrc

#坐标系
COORD_BASE  = 0
COORD_JOINT = 1
COORD_TOOL  = 2
#运动模式
ABS = 0
INCR= 1
#关节 1~6 依次对应 0~5,

robot = jkrc.RC("192.168.2.160")#返回机器人对象
robot.login()#登录
robot.power_on() #上电
robot.enable_robot()
print("move1")
robot.jog(0,INCR,COORD_JOINT,30,PI)
time.sleep(5)#jog 为非阻塞指令，运动状态下接收 jog 指令会被丢弃
```

```
print("move2")
robot.jog(0,INCR,COORD_JOINT,5,-PI)
time.sleep(0.5)
robot.jog_stop(0)  #运动 0.5 秒后停止，与前一个示例代码对比
robot.logout()
```

4.2.3 机器人关节运动

函数	joint_move(joint_pos, move_mode, is_block, speed)
描述	机器人关节运动到目标点位
参数	joint_pos: 机器人关节运动目标位置。 move_mode: 0 代表绝对运动，1 代表相对运动 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，阻塞表示机器人运动完成才会有返回值，非阻塞表示接口调用完成立刻就有返回值。 speed: 机器人关节运动速度，单位：rad/s acc: 关节加速度默认 90rad/s^2
返回值	成功：(0,) 失败：其它

代码示例：

'''关节运动'''

```
# -*- coding: utf-8 -*-
# import sys
# sys.path.append('D:\\vs2019ws\PythonCtt\PythonCtt')
import time
import jkrc

#运动模式
ABS = 0
INCR= 1
joint_pos=[PI,PI/2,0,PI//4,0,0]
robot = jkrc.RC("192.168.2.160")#返回机器人对象
robot.login()#登录
robot.power_on() #上电
robot.enable_robot()
print("move1")
robot.joint_move(joint_pos,ABS,True,1)
time.sleep(3)
robot.logout()
```

4.2.4 机器人扩展关节运动

函数	joint_move_extend(joint_pos, move_mode, is_block, speed, acc, tol)
描述	机器人扩展关节运动。增加关节角加速度和关节运动终点误差。
参数	joint_pos: 机器人关节运动各目标关节角度。 move_mode: 指定运动模式, 0 为绝对运动，1 为增量运动，2 代表连续运动。 is_block: 是否为阻塞接口，True 为阻塞接口，False 为非阻塞接口。 speed: 机器人关节运动速度，单位：rad/s acc：机器人关节运动角加速度。 tol：机器人运动终点误差。
返回值	成功：(0,) 失败：其它

代码示例：

```
import jkrc                                    #导入模块
robot = jkrc.RC("192.168.2.226")  #返回一个机器人对象
robot.login()
robot.joint_move_extend(joint_pos=[1, 1, 1, 1, 1, 1],move_mode=0, is_block=True, speed=20, acc=5, tol=0.1)
robot.joint_move_extend(joint_pos=[-1, 1, 1, 1, 1, 0],move_mode=0, is_block=True, speed=20, acc=5, tol=0.1)
robot.logout() #登出
```

4.2.5 机器人末端直线运动

函数	linear_move(end_pos, move_mode, is_block, speed)
描述	机器人末端直线运动到目标点位
参数	end_pos: 机器人末端运动目标位置 move_mode: 0 代表绝对运动，1 代表相对运动 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，阻塞表示机器人运动完成才会有返回值，非阻塞表示接口调用完成立刻就有返回值。 speed: 机器人直线运动速度，单位：mm/s
返回值	成功：(0,) 失败：其它

代码示例：

```
'''直线运动'''
import time
```

```
import jkrc

#运动模式
ABS = 0
INCR= 1
tcp_pos=[0,0,-30,0,0,0]
robot = jkrc.RC("192.168.2.160")#返回机器人对象

robot.login()#登录

robot.power_on() #上电
robot.enable_robot()
print("move1")
#阻塞 沿 z 轴负方向 以 10mm/s 运动 30mm
ret=robot.linear_move(tcp_pos,INCR,True,10)
print(ret[0])
time.sleep(3)
robot.logout()
```

说明:由于机器人型号的不同，上述示例填写的位姿可能会逆解失败，返回 -4.

4.2.6 机器人扩展末端直线运动

函数	linear_move_extend(end_pos, move_mode, is_block, speed, acc, tol)
描述	机器人扩展末端直线运动。增加空间加速度和空间运动终点误差。
参数	end_pos: 机器人末端运动目标位置。 move_mode: 指定运动模式, 0 为绝对运动，1 为增量运动 is_block: 是否为阻塞接口，True 为阻塞接口，False 为非阻塞接口。 speed: 机器人笛卡尔空间运动速度，单位：mm/s acc：机器人笛卡尔空间加速度，单位：mm/s^2。 tol：机器人运动终点误差。
返回值	成功：(0,) 失败：其它

4.2.7 机器人末端圆弧运动

函数	circular_move (end_pos, mid_pos, move_mode, is_block, speed, acc, tol)
描述	机器人末端圆弧运动
参数	end_pos: 机器人末端运动目标位置。

	mid_pos: 机器人末端运动中间点。 move_mode: 指定运动模式, 0 为绝对运动，1 为增量运动，2 代表连续运动。 is_block: 是否为阻塞接口，True 为阻塞接口，False 为非阻塞接口。 speed: 机器人直线运动速度，单位：mm/s acc：机器人直线运动角加速度，单位：mm/s^2 tol：机器人运动终点误差。
返回值	成功：(0,) 失败：其它

代码示例：

```
1. import jkrc                                  #导入模块
2. robot = jkrc.RC("192.168.2.226")  #返回一个机器人对象
3. robot.login()
4. robot.circular_move(end_pos = [100, 100, 0, 0, 1, 1],mid_pos = [100, 0, 0, 0, 1, 0],move_mode= 0, is_block = True, speed = 20, acc = 5, tol = 0.1)
5. robot.logout() #登出
```

函数	**circular_move_extend (end_pos, mid_pos, move_mode, is_block, speed, acc, tol, cricle_cnt, opt_cond)**
描述	机器人末端圆弧运动
参数	end_pos: 机器人末端运动目标位置。 mid_pos: 机器人末端运动中间点。 move_mode: 指定运动模式, 0 为绝对运动，1 为增量运动，2 代表连续运动。 is_block: 是否为阻塞接口，True 为阻塞接口，False 为非阻塞接口。 speed: 机器人直线运动速度，单位：mm/s acc：机器人直线运动加速度，单位：mm/s^2 tol：机器人运动终点误差。 circle_cnt: 圆弧运动圈数。 opt_cond: 占位符，传 none。
返回值	成功：(0,) 失败：其它

代码示例：

```
1. import jkrc
2. import math
3. import traceback
4.
5.
6. _ABS = 0
7. _BLOCK = 1
8.
9.
10. try:
11.     rc = jkrc.RC("192.168.20.139")
12.     rc.login()
```

```
    rc.power_on()
    rc.enable_robot()

    start_pos = [-406.250, 116.250, 374.000, PI, 0,PI/2]
    mid_pos = [-468.650, 211.450, 374.000, PI, 0,PI/2]
    end_pos = [-295.050, 267.450, 374.000, PI, 0, PI/2]

    rc.joint_move([0] + [PI * 0.5] * 3 + [PI * -0.5, 0], 0, 1, 200)
    rc.linear_move(start_pos, _ABS, _BLOCK, 50)
    rc.circular_move_extend(end_pos, mid_pos, _ABS, _BLOCK, 250, 1200, 5, None, 5)
except Exception:
    traceback.print_exc()
```

4.2.8 机械臂设置阻塞运动超时时间

函数	set_block_wait_timeout (seconds)
描述	机械臂设置阻塞运动超时时间
参数	Seconds 大于 0.5
返回值	成功：(0,) 失败：其它

4.2.9 终止当前机械臂运动

函数	motion_abort ()
描述	可以在任何情况下终止机器人的运动
参数	
返回值	成功：(0,) 失败：其它

代码示例：

```
# -*- coding: utf-8 -*-
import sys
#sys.path.append('D:\\vs2019ws\PythonCtt\lib')
import time
import jkrc

robot = jkrc.RC("192.168.2.160")#返回一个机器人对象

robot.login()#登录

robot.power_on() #上电

robot.enable_robot()
print("move1")
```

```
robot.joint_move(joint_pos=[1,0,0,0,0,0],move_mode=1,is_block=False,speed=0.05)#增量运动
print("wait")
time.sleep(2)
print("stop")
robot.motion_abort()
robot.logout()#登出
```

4.2.10 查询机器人运动状态

函数	**get_motion_status()**
描述	查询机器人运动状态
参数	

返回值	成功：(0, data)，
	int motion_line; ///< the executing motion cmd id
	int motion_line_sdk; ///< reserved
	BOOL inpos; ///< previous motion cmd is done, should alway check queue info together
	BOOL err_add_line; ///< previous motion cmd is dropped by controller, like target is already reached
	int queue; ///< motion cmd number in buffer
	int active_queue; ///< motion cmd number in blending buffer
	BOOL queue_full; ///< motion buffer is full and motion cmds reveived at this moment will be dropped
	BOOL paused; ///< motion cmd is paused and able to resume
	失败：其它

4.2.11 查询机器人运动是否停止

函数	**is_in_pos()**
描述	查询机器人运动是否停止
参数	
返回值	成功：(0, state)，state 为 1 代表机器人停止，0 则相反 失败：其它

4.3 机械臂操作信息设置与查询

4.3.1 获取机械臂状态（simple）

函数	get_robot_status_simple()
描述	获取机械臂状态
参数	无
返回值	成功：(0,data) errcode:错误码 errmsg: 错误信息 powered_on: 是否上电 enabled: 是否上使能 失败：其它

4.3.2 获取机器人状态监测数据

函数	get_robot_status ()
描述	获取机器人状态数据监测数据,支持多线程安全
参数	
返回值	成功：(0, robotstatus), robotstatus 的长度为 24，robotstatus 返回数据顺序如下所示: 1. errcode 机器人运行出错时错误编号，0 为运行正常，其它为异常 2. inpos 机器人运动是否到位标志，0 为没有到位，1 为运动到位 3. powered_on 机器人是否上电标志，0 为没有上电，1 为上电 4. enabled 机器人是否使能标志，0 为没有使能，1 为使能 5. rapidrate 机器人运行倍率 6. protective_stop 机器人是否检测到碰撞，0 为没有检测到碰撞，1 则相反 7. drag_status 机器人是否处于拖拽状态，0 为没有处于拖拽状态，1 则相反 8. 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 9. current_user_id 机器人目前使用的用户坐标系 id 10. current_tool_id 机器人目前使用的工具坐标系 id 11. dout 机器人控制柜数字输出信号 12. din 机器人控制柜数字输入信号 13. aout 机器人控制柜模拟输出信号 14. ain 机器人控制柜模拟输入信号 15. tio_dout 机器人末端工具数字输出信号 16. tio_din 机器人末端工具数字输入信号 17. tio_ain 机器人末端工具模拟输入信号 18. extio 机器人外部扩展模块 IO 信号 19. cart_position 机器人末端的笛卡尔空间位置值 20. joint_position 机器人关节空间位置 21. robot_monitor_data 机器人状态监测数据（scb 主版本号、scb 小版本号、控

	制器温度、机器人平均电压、机器人平均电流、机器人 6 个关节的监测数据（瞬时电流、瞬时电压、瞬时温度）） 22. torq_sensor_monitor_data 机器人力矩传感器状态监测数据（力矩传感器 ip 地址、力矩传感器端口号、工具负载（负载质量、质心 x 轴坐标、质心 y 轴坐标、质心 z 轴坐标）、力矩传感器状态、力矩传感器异常错误码、6 个力矩传感器实际接触力值、6 个力矩传感器原始读数值、6 个力矩传感器实际接触力值（不随初始化选项变化）） 23. 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 24. emergency_stop 机器人是否急停，0 为没有按下急停，1 则相反 25. tio_key 机器人末端工具按钮 [0]free;[1]point;[2]末端灯光按钮; 失败：其它 **说明:** **若机器人无对应 IO，会返回如“no extio”字符串。** 错误码文件可以在 **JAKA 二次开发资料包\参考资料**中找到

代码示例：

```
1.  import sys
2.  import jkrc
3.  import time
4.
5.  robot = jkrc.RC("192.168.2.160")
6.  robot.login()
7.  ret = robot.get_robot_status()
8.  if ret[0] == 0:
9.      print("the joint position is :",ret[1])
10.     print(len(ret[1]))
11.     for i in range(len(ret[1])):
12.         print(ret[1][i])
13. else:
14.     print("some things happend,the errcode is: ",ret[0])
15. robot.logout()
```

4.3.3 设置机器人状态数据自动更新时间间隔

函数	**set_status_data_update_time_interval (millisecond)**
描述	设置机器人状态数据自动更新时间间隔，可以改变系统占用率，默认尽可能快地发送以保证实时性。（默认间隔：4ms)
参数	millisecond: 时间参数，单位：ms。
返回值	成功：(0,) 失败：其它

代码示例：

```
1.  import jkrc                                  #导入模块
```

```
robot = jkrc.RC("192.168.2.165")  #返回一个机器人对象
robot.login()
robot.set_status_data_update_time_interval(100)
robot.logout()
```

4.3.4 获取当前机器人的六个关节角度值

函数	get_joint_position()
描述	获取当前机器人的六个关节角度值
参数	
返回值	成功：(0, joint_pos), joint_pos 是一个包含 6 位元素的元组 (j1, j2, j3, j4, j5, j6)，j1, j2,j3, j4, j5, j6 分别代表关节 1 到关节 6 的角度值。 失败：其它

代码示例：

```
'''获得关节角度'''
# -*- coding: utf-8 -*-
import sys
#sys.path.append('D:\\vs2019ws\PythonCtt\lib')
import time
import jkrc

robot = jkrc.RC("192.168.2.160")#返回一个机器人对象
ret = robot.login()#登录
ret = robot.get_joint_position()
if ret[0] == 0:
    print("the joint position is :",ret[1])
else:
    print("some things happend,the errcode is: ",ret[0])
robot.logout()  #登出
```

4.3.5 获取当前设置下工具末端的位姿

函数	get_tcp_position()
描述	获取当前设置下工具末端的位姿
参数	
返回值	成功：(0, cartesian_pose), cartesian_pose 是一个包含 6 位元素的元组(x, y, z, rx ,ry, rz)，x, y, z, rx, ry, rz 代表机器人工具末端的位姿值 失败：其它

代码示例：

'''获得 tcp 位姿'''

```python
# -*- coding: utf-8 -*-
import sys
#sys.path.append('D:\\vs2019ws\PythonCtt\lib')
import time
import jkrc

robot = jkrc.RC("192.168.2.160")#返回一个机器人对象

ret = robot.login()#登录
ret = robot.get_tcp_position()
if ret[0] == 0:
    print("the tcp position is :",ret[1])
else:
    print("some things happend,the errcode is: ",ret[0])
robot.logout()  #登出
```

4.3.6 设置用户坐标系信息

函数	set_user_frame_data(id, user_frame, name)
描述	设置用户坐标系信息。
参数	id : 用户坐标系 ID，可选 ID 为 1 到 10,0 代表机器人基坐标系 user_frame: 用户坐标系参数[x,y,z,rx,ry,rz] name: 用户坐标系别名
返回值	成功：(0,) 失败：其它

4.3.7 获取用户坐标系信息

函数	get_user_frame_data(id)
描述	获取用户坐标系信息
参数	id 用户坐标系 ID 查询结果
返回值	成功：(0, id, tcp), id : 用户坐标系 ID，可选 ID 为 1 到 10, 0 代表机器人基坐标系 tcp: 用户坐标系参数[x,y,z,rx,ry,rz] 失败：其它

4.3.8 查询当前使用的用户坐标系 ID

函数	get_user_frame_id()
描述	查询当前使用的用户坐标系 ID
参数	
返回值	成功：(0, id)，id 值范围为 0 到 10, 0 代表机器人基坐标系 失败：其它

4.3.9 设置当前使用的用户坐标系 ID

函数	set_user_frame_id(id)
描述	设置当前使用的用户坐标系 ID
参数	id: 用户坐标系 ID
返回值	成功：(0,) 失败：其它

4.3.10 查询机器人当前使用的工具 ID

函数	get_tool_id()
描述	查询机器人当前使用的工具 ID
参数	
返回值	成功：(0, id), id 值范围为 0 到 10，0 代表末端法兰盘，已被控制器使用。 失败：其它

代码示例：

```
# -*- coding: utf-8 -*-
import sys
sys.path.append('D:\\vs2019ws\PythonCtt\PythonCtt')
import time
import jkrc


robot = jkrc.RC("192.168.2.160") #返回一个机器人对象
robot.login()                          #登录
ret = robot.get_tool_data(1)     #查询工具坐标系数据
if ret[0] == 0:
    print("the tool data is :",ret)
else:
    print("some things happend,the errcode is: ",ret[0])
robot.set_tool_data(1,[0,0,1,0,0,0],'testlx') #设置工具坐标系数据
```

```
time.sleep(0.5)
ret = robot.get_tool_data(1)    #查询工具坐标系数据
if ret[0] == 0:
    print("the tool data is :",ret)
else:
    print("some things happend,the errcode is: ",ret[0])
ret = robot.get_tool_id()     #查询工具坐标系 id
print("tool_id",ret)
robot.set_tool_id(1)          #设置工具坐标系数据
time.sleep(0.5)
ret = robot.get_tool_id()     #查询工具坐标系 id
print("tool_id",ret)
robot.logout()
```

4.3.11 设置指定编号的工具信息

函数	**set_tool_data(id, tcp, name)**
描述	设置指定编号的工具信息
参数	id：设置工具 ID，可选 ID 为 1 到 10，0 代表末端法兰盘，已被控制器使用。 tcp：设置工具坐标系参数[x,y,z,rx,ry,rz] name：用户坐标系别名
返回值	成功：(0,) 失败：其它

4.3.12 设置当前使用的工具 ID

函数	**set_tool_id(id)**
描述	设置当前使用的工具 ID
参数	id：工具坐标系 ID
返回值	成功：(0,) 失败：其它

4.3.13 查询目标工具坐标系的信息

函数	**get_tool_data(id)**
描述	查询机器人使用的工具信息
参数	
返回值	成功：(0, id, tcp)，id 值范围为 0 到 10，0 代表末端法兰盘，已被控制器使用。 tcp：工具坐标系参数[x,y,z,rx,ry,rz] 失败：其它

4.3.14 设置数字输出变量(DO)的值

函数	set_digital_output(iotype = a_type, index = a_number, value = a_value)
描述	设置数字输出变量(DO)的值
参数	iotype : DO 类型 index : DO 索引 value : DO 设置值
返回值	成功：(0,) 失败：其它

代码示例：

'''设置 DO3 的值为 1'''

```
# -*- coding: utf-8 -*-
import sys
sys.path.append('D:\\vs2019ws\PythonCtt\PythonCtt')
import time
import jkrc


IO_CABINET  = 0  #控制柜面板 IO
IO_TOOL = 1     #工具 IO
IO_EXTEND = 2   #扩展 IO

robot = jkrc.RC("192.168.2.160")
robot.login()
ret = robot.get_digital_output(0,2)
if ret[0] == 0:
    print("1the DO2 is :",ret[1])
else:
    print("some things happend,the errcode is: ",ret[0])
robot.set_digital_output(IO_CABINET, 2, 1)#设置 DO2 的引脚输出值为 1
time.sleep(0.1)
ret = robot.get_digital_output(0, 2)
if ret[0] == 0:
    print("2the DO2 is :",ret[1])
else:
    print("some things happend,the errcode is: ",ret[0])
robot.logout()  #登出
```

4.3.15 设置模拟输出变量的值(AO)的值

函数	set_analog_output(iotype = a_type, index = a_number, value = a_value)
描述	设置模拟输出变量的值(AO)的值
参数	iotype : AO 类型 index : AO 索引 value : AO 设置值
返回值	成功：(0,) 失败：其它

代码示例：

```
'''设置 AO4 的值为 1.55'''
import jkrc
IO_CABINET  =0  #控制柜面板 IO
IO_TOOL = 1     #工具 IO
IO_EXTEND = 2   #扩展 IO
robot = jkrc.RC("192.168.2.64")
robot.login()
robot.set_analog_output(iotype = IO_CABINET,index = 3,value = 1.55)#
robot.logout()  #登出
```

4.3.16 查询数字输入(DI)状态

函数	get_digital_input(iotype, index)
描述	查询数字输入(DI)状态
参数	iotype: DI 类型 index: DI 索引
返回值	成功：(0, value)，value: DI 状态查询结果 失败：其它

4.3.17 查询数字输出(DO)状态

函数	get_digital_output(iotype, index)
描述	查询数字输出(DO)状态
参数	iotype: DO 类型 index: DO 索引
返回值	成功：(0, value)，value: DO 状态查询结果 失败：其它

4.3.18 查询模拟量输入变量(AI)的值

函数	**get_analog_input(iotype, index)**
描述	获取模拟量输入变量(AI)的值
参数	iotype : AI 类型 index : AI 索引
返回值	成功：(0, value), value 是 AI 状态查询结果，为浮点数 失败：其它

4.3.19 查询模拟量输出变量(AO)的值

函数	**get_analog_output(type, index)**
描述	获取模拟量输出变量(AO)的值
参数	type : AO 类型 index : AO 索引
返回值	成功：(0, value), value 是 AO 状态查询结果，为浮点数 失败：其它

代码示例：

```
'''查询 AO4 的值为'''
import jkrc
IO_CABINET  =0  #控制柜面板 IO
IO_TOOL = 1     #工具 IO
IO_EXTEND = 2   #扩展 IO
robot = jkrc.RC("192.168.2.64")
robot.login()
robot.set_analog_output(iotype = IO_CABINET,index = 3,value = 1.55)
ret = robot.get_analog_output(iotype = IO_CABINET,index = 3)
print("AO value is:",ret[1])
robot.logout()  #登出
```

4.3.20 查询扩展 IO 模块是否运行

函数	**is_extio_running()**
描述	查询扩展 IO 模块是否运行
参数	
返回值	成功：(0, status)，status 为 1 时代表扩展 IO 模块，为 0 时则相反 失败：其它

4.3.21 机器人负载设置

函数	set_payload(mass = m, centroid = [x,y,z])
描述	机器人负载设置
参数	mass: 负载质量，单位: kg centroid: 负载质心坐标[x, y ,z]，单位: mm
返回值	成功：(0,) 失败：其它

代码示例：

```
import jkrc                                  #导入模块
robot = jkrc.RC("192.168.2.226")             #返回一个机器人对象
robot.login()
robot.set_payload(mass= 1, centroid =[0.01,0.02,0.03])
robot.logout()                                 #登出
```

4.3.22 获取机器人负载数据

函数	get_payload()
描述	获取机器人负载数据
参数	
返回值	成功：(0, payload), payload 是长度为 2 的元组 (mass, (x,y,z))，第一个元素是负载质量，第二个元素是质心坐标。 失败：其它

代码示例：

```
'''''  设置负载'''
import jkrc
robot = jkrc.RC("192.168.2.64") #返回一个机器人对象
robot.login()                          #登录
robot.set_payload(mass= 1,centroid =[0.01,0.02,0.03])
ret = robot.get_payload()
if ret[0] == 0:
    print("the payload is :",ret[1])
else:
    print("some things happend,the errcode is: ",ret[0])
robot.logout()
```

4.3.23 设置 tioV3 电压参数

函数	set_tio_vout_param (vout_enable ,vout_vol)
描述	4.3.23 设置 tioV3 电压参数
参数	vout_enable 电压使能，0:关，1 开 vout_vol 电压大小 0:24v 1:12v
返回值	成功：(0) 失败：其它

4.3.24 获取 tioV3 电压参数

函数	get_tio_vout_param (vout_enable ,vout_vol)
描述	获取 tioV3 电压参数
参数	vout_enable 电压使能，0:关，1 开 vout_vol 电压大小 0:24v 1:12v
返回值	成功：(0,(vout_enable ,vout_vol)) 失败：其它

4.3.25 获取机械臂状态

函数	get_robot_state ()
描述	获取机械臂状态
参数	
返回值	成功：(0,(estoped, power_on, servo_enabled)) estoped: 急停 0: 关， 1: 开 power_on: 上电 0: 关，1: 开 servo_enabled: 伺服使能 0: 关，1: 开 失败：其它

代码示例：

```
def example_get_robot_state():
    _RC_ADDRESS = "192.168.2.64"
    rc = jkrc.RC(_RC_ADDRESS)
    rc.login()
    ret, (estoped, power_on, servo_enabled) = rc.get_robot_state()
    print('ret is {}, estop: {}, power_on {}, servo_enabled: {}'.format(ret, estoped, power_on, servo_enabled))
```

4.3.26 TIO 添加或修改信号量

函数	add_tio_rs_signal (sign_info)

描述	添加或修改信号量
参数	sign_info: dict 信号量属性
返回值	成功：(0,) 失败：其它

代码示例：

```
def example_add_tio_rs_signal():
    rc = jkrc.RC(_RC_ADDRESS)
    rc.login()
    ret = rc.add_tio_rs_signal({
        'sig_name': 'signal_tmp',  //标识名
        'chn_id': 0,  //RS485 通道 ID
        'sig_type': 0,  //信号量类型
        'sig_addr': 0x1,  //寄存器地址
        'value': 5,  //值  设置时无效
        'frequency': 5  //信号量在控制器内部刷新频率不大于 10
    })
```

4.3.27 TIO 删除信号量

函数	del_tio_rs_signal (sign_name)
描述	添加或修改信号量
参数	sign_info: str 信号量标识名
返回值	成功：(0,) 失败：其它

代码示例：

```
def example_del_tio_rs_signal():
    rc = jkrc.RC(_RC_ADDRESS)
    rc.login()
    ret = rc.del_tio_rs_signal('signal_tmp')
    print('ret is {}'.format(ret))
```

4.3.28 TIO RS485 发送指令

函数	send_tio_rs_command(chn_id, cmd)
描述	RS485 发送指令
参数	chn_id: int 通道号 data: str 数据字段；十六进制字符串转换为字节数组
返回值	成功：(0,) 失败：其它

代码示例：

```
def example_send_tio_rs_command():
```

```
    rc = jkrc.RC(_RC_ADDRESS)
    rc.login()
    ret = rc.send_tio_rs_command(2, bytearray.fromhex("01 06 01 00 00 01"))
    print('ret is {}'.format(ret))
```

4.3.29 TIO 获取信号量信息

函数	get_rs485_signal_info()
描述	RS485 信号量信息
参数	
返回值	成功：(0,sign_info_list) sign_info_list: list 信号量信息数组 失败：其它

代码示例：

```
def example_get_rs485_signal_info():
    rc = jkrc.RC(_RC_ADDRESS)
    rc.login()
    ret, sign_info_list = rc.get_rs485_signal_info()
    print('ret is: {}, sign_info_list: {}'.format(ret, sign_info_list))
    # [{'value': 0, 'chn_id': 0, 'sig_addr': 0, 'sig_name': '', 'sig_type': 0, 'frequency': 0}, ...]  -> 4.3.26TIO 添加或修改信号量
```

4.3.30 TIO 设置 TIO 模式

函数	set_tio_pin_mode(pin_type, pin_mode)
描述	设置 tio 模式
参数	pin_type: tio 类型 0 for DI Pins, 1 for DO Pins, 2 for AI Pins pin_mode: 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
返回值	成功：(0,) 失败：其它

4.3.31 TIO 获取 TIO 模式

函数	get_tio_pin_mode(pin_type)
描述	获取 tio 模式
参数	pin_type: tio 类型 0 for DI Pins, 1 for DO Pins, 2 for AI Pins

返回值	成功：(0,pin_mode) 失败：其它

4.3.32 TIO 通讯参数配置

函数	**set_rs485_chn_comm(dict)**
描述	RS485 通讯参数配置

参数	当通道模式设置为 Modbus RTU 时，需额外指定 Modbus 从站节点 ID
	传参类型为字典：
	dict = {
	'chn_id':1,
	'slave_id':1,
	'baudrate': 115200,
	'databit': 8,
	'stopbit': 1,
	'parity': 78
	}

返回值	成功：(0,) 失败：其它

4.3.33 TIO RS485 通讯参数查询

函数	**get_rs485_chn_comm(chn_id)**
描述	RS485 通讯参数查询
参数	
返回值	成功：**(0, (chn_id, slave_id,baudrate, databit, stopbit, parity))** 失败：其它

4.3.34 TIO RS485 通讯模式配置

函数	**set_rs485_chn_mode(chn_id, chn_mode)**
描述	RS485 通讯模式配置
参数	**chn_id 0: RS485H, channel 1; 1:RS485L, channel 2** **chn_mode: 0: Modbus RTU, 1: Raw RS485, 2, torque sensor**
返回值	成功：**(0,)** 失败：其它

4.3.35 TIO RS485 通讯模式查询

函数	**get_rs485_chn_mode(chn_id)**
描述	RS485 通讯模式查询
参数	**chn_id 0: RS485H, channel 1; 1:RS485L, channel 2**
返回值	成功：**(0, chn_mode)** **chn_mode: 0: Modbus RTU, 1: Raw RS485, 2, torque sensor** 失败：其它

4.3.36 设置机器人安装角度

函数	**set_installation_angle(anglex, angley)**
描述	设置机器人安装角度
参数	anglex: x 方向安装角度，范围[0, PI]度 anglez: z 方向安装角度，范围[0, 360]度
返回值	成功：(0) 失败:其它

代码示例：

```
import jkrc
rc = jkrc.RC("192.168.137.152")
res = rc.login()
if res[0] != 0:
    raise "rc login failed."

anglex = 3.14
angley = 0
res = rc.set_installation_angle(anglex, angley)
if res[0] != 0:
    raise "set installation angle failed."

res = rc.get_installation_angle()
if res[0] != 0:
    raise "get installation angle failed."

print("installation angle:")
print("quat: [{x}, {y}, {z}, {s}]".format(s=res[1][0], x=res[1][1], y=res[1][2], z=res[1][3]))
print("rpy: [{rx}, {ry}, {rz}]".format(rx=res[1][4], ry=res[1][5], rz=res[1][6]))

rc.logout()
```

4.3.37 获取机器人安装角度

函数	get_installation_angle()
描述	设置机器人安装角度
参数	anglex: x 方向安装角度 anglez: z 方向安装角度
返回值	成功：(0, [qs, qx, qy, qz, rx, ry, rz])。安装角度四元数表示[qx, qy, qz, qs]，安装角度欧拉角表示[rx, ry, rz]，其中 ry 固定为 0。 失败:其它

代码示例：

```
import jkrc
rc = jkrc.RC("192.168.137.152")
res = rc.login()
if res[0] != 0:
    raise "rc login failed."

anglex = 3.14
angley = 0
res = rc.set_installation_angle(anglex, angley)
if res[0] != 0:
    raise "set installation angle failed."

res = rc.get_installation_angle()
if res[0] != 0:
    raise "get installation angle failed."

print("installation angle:")
print("quat: [{x}, {y}, {z}, {s}]".format(s=res[1][0], x=res[1][1], y=res[1][2], z=res[1][3]))
print("rpy: [{rx}, {ry}, {rz}]".format(rx=res[1][4], ry=res[1][5], rz=res[1][6]))

rc.logout()
```

4.3.38 设置系统变量

函数	set_user_var(id, value, name)
描述	设置已有系统变量信息
参数	id: 系统变量 id value: 系统变量的值 name: 系统变量的别名
返回值	成功：**(0,)** 失败：其它

4.3.39 获取系统变量

函数	get_user_var()
描述	用于获取系统变量信息
参数	无
返回值	成功：(0, data) 失败：其它

4.4 机械臂安全状态设置

4.4.1 查询机器人是否超出限位

函数	is_on_limit()
描述	查询机器人是否超出限位
参数	
返回值	成功：(0, state)，state 为 1 代表机器人超出限位，0 则相反 失败：其它

4.4.2 查询机器人是否处于碰撞保护模式

函数	is_in_collision()
描述	查询机器人是否处于碰撞保护模式
参数	
返回值	成功：(0, state)，state 为 1 代表机器人处于碰撞保护模式，0 则相反 失败：其它

4.4.3 碰撞之后从碰撞保护模式恢复

函数	collision_recover()
描述	碰撞之后从碰撞保护模式恢复
参数	
返回值	成功：(0,) 失败：其它

代码示例：

```
from typing import Counter
import jkrc
import time
```

```
4.
5.
6.  robot = jkrc.RC("192.168.2.160")
7.  robot.login()
8.  robot.power_on()
9.  robot.enable_robot()
10. ret = robot.get_collision_level()#获取当前碰撞等级
11. print(ret)
12. robot.set_collision_level(1)#设置碰撞等级
13. ret = robot.get_collision_level()
14. print(ret)
15. num = 0
16. while(1):
17.     ret = robot.is_in_collision()  #查询是否处于碰撞保护模式
18.     collision_status = ret[1]
19.     if collision_status == 1:
20.         time.sleep(5)
21.         robot.collision_recover()   #如果发生了碰撞，从碰撞保护模式恢复
22.         print(" in collision "+ str(num))
23.     else:
24.         print("the robot is not in collision "+ str(num))
25.     time.sleep(1)
26.     num=num+1
27.
28. robot.logout()
```

4.4.4 设置机器人碰撞等级

函数	set_collision_level(level)
描述	设置机器人碰撞等级
参数	level: 碰撞等级，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
返回值	成功：(0,) 失败：其它

4.4.5 获取机器人碰撞等级

函数	get_collision_level()
描述	获取机器人碰撞等级

参数	
返回值	成功：(0, level)，level 是返回的碰撞等级，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 失败：其它

4.4.6 获取最新的错误码

函数	get_last_error ()
描述	获取机器人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。如果要获得具体的错误信息，需要使用 set_errorcode_file_path 设置错误码文件路径，如果只是需要获取错误码，不需要调用 set_errorcode_file_path。
参数	
返回值	成功：(0, error) 失败：其它

代码示例：

```
import jkrc
robot = jkrc.RC("192.168.2.194")#返回一个机器人对象
robot.login()   #登录
robot.program_load("not_exist") #故意加载一个不存在的程序,引起报错
ret = robot.get_last_error ()#没有设置错误码文件路径，只能得到错误码，得不到具体错误信息
print(ret[1])
robot.set_errorcode_file_path("D:\\JAKA_ERROR_CODE.csv") #路径不能包含中文
ret = robot.get_last_error ()   #设置错误码文件路径后，能得到错误码和具体错误信息
print(ret[1])
robot.logout()  #登出
```

4.4.7 设置错误码文件的路径

函数	set_errorcode_file_path (errcode_file_path)
描述	设置错误码文件路径，使用 get_last_error 接口时，如果要获得具体的错误信息，需要

	使用此接口设置错误码文件路径，如果只是需要获取错误码，不需要调用此接口。 **注:路径不能包含中文否则无法使用**
参数	errcode_file_path：错误码文件存放路径，错误码文件可以在 JAKA 二次开发资料包\参考资料中找到
返回值	成功：(0,) 失败：其它

4.4.8 错误状态清除

函数	**clear_error()**
描述	错误状态清除。
参数	
返回值	成功：(0,) 失败：其它

代码示例：

参考碰撞保护模式恢复

4.4.9 设置网络异常时机器人自动终止运动类型

函数	**set_network_exception_handle (millisecond, mnt)**
描述	设置网络异常控制句柄，当网络出现异常情况时，控制机器人运动状态。
参数	millisecond: 时间参数，单位：ms。 mnt: 网络异常时机器人需要进行的动作类型, 0 代表机器人保持原来的运动，1 代表暂停运动，2 代表终止运动。
返回值	成功：(0,) 失败：其它

代码示例：

```
# -*- coding: utf-8 -*-
# import sys
# sys.path.append('D:\\vs2019ws\PythonCtt\PythonCtt')
import time
import jkrc

#运动模式

ABS = 0
INCR= 1

robot = jkrc.RC("192.168.2.160")#返回机器人对象

robot.login()#登录
```

```
robot.power_on() #上电
robot.enable_robot()
robot.set_network_exception_handle(100,2)#设置 100ms，暂停运动。
print("move1")
num=0
while(1):
    robot.joint_move([1,1,1,1,1,1],ABS,False,0.5)
    robot.joint_move([-1,1,1,1,1,1],ABS,False,0.5)
    num = num +1
    print(num)
    time.sleep(6)
robot.logout()
```

4.5 使用 APP 脚本程序

4.5.1 加载指定的作业程序

函数	program_load(file_name)
描述	加载指定的作业程序
参数	file_name：程序名称，传入的名称是一个字符串，比如 “file_name”
返回值	成功：(0,) 失败：其它

4.5.2 获取已加载的作业程序名称

函数	get_loaded_program()
描述	获取已加载的作业程序名称
参数	
返回值	成功：(0, file_name) 失败：其它

代码示例：

```
import jkrc
robot = jkrc.RC("192.168.2.64")#返回一个机器人对象
robot.login()  #登录
ret = robot.program_load("program_test")#加载通过 APP 编写的脚本 program_test 需要自己编写
ret = robot.get_loaded_program()
print("the loaded program is:",ret[1])
robot.logout() #登出
```

4.5.3 获取当前机器人作业程序的执行行号

函数	**get_current_line()**
描述	获取当前机器人作业程序的执行行号
参数	
返回值	成功：(0, curr_line), curr_line: 当前行号查询结果。 失败：其它

4.5.4 运行当前加载的作业程序

函数	**program_run()**
描述	运行当前加载的作业程序
参数	
返回值	成功：(0,) 失败：其它

4.5.5 停当前运行的作业程序

函数	**program_pause()**
描述	暂停当前运行的作业程序
参数	
返回值	成功：(0,) 失败：其它

4.5.6 继续运行当前暂停的作业程序

函数	**program_resume()**
描述	继续运行当前暂停的作业程序
参数	
返回值	成功：(0,) 失败：其它

4.5.7 终止当前执行的作业程序

函数	**program_abort()**
描述	终止当前执行的作业程序
参数	
返回值	成功：(0,)

	失败：其它

4.5.8 获取机器人作业程序执行状态

函数	get_program_state()
描述	获取机器人作业程序执行状态
参数	
返回值	成功：(0, state)，state 的值有三种可能分别是 0、1、2 0 代表程序停止运行 1 代表程序正在运行 2 代表程序暂停 失败：其它

代码示例：

```
# -*- coding: utf-8 -*-
import sys
sys.path.append('D:\\vs2019ws\PythonCtt\PythonCtt')
import time
import jkrc
import _thread


def print_state(name,robot):
    while(1):
        ret = robot.get_program_state() #查询程序运行状态，0 程序终止或无程序运行，1 程序运行中，2 暂停
        print("the robot program state is:",ret[1])
        time.sleep(1)

robot = jkrc.RC("192.168.2.160")#返回一个机器人对象

robot.login()  #登录

ret = robot.program_load("simple")#加载通过 APP 编写的脚本 program_test 需要自己编写

ret = robot.get_loaded_program()
print("the loaded program is:",ret[1])
robot.program_run()
_thread.start_new_thread( print_state,("p1_state", robot))#开启一个”p1”线程查询程序状态
time.sleep(10)

robot.program_pause() #暂停
```

```
time.sleep(10)
robot.program_resume() #恢复
time.sleep(10)
robot.program_abort()   #停止
time.sleep(3)
robot.logout()          #登出
```

4.5.9 设置机器人运行速度倍率

函数	**set_rapidrate(rapid_rate)**
描述	设置机器人运行速度倍率
参数	rapid_rate: 机器人运行速度倍率
返回值	成功：(0,) 失败：其它

4.5.10 获取机器人运行速度倍率

函数	**get_rapidrate()**
描述	获取机器人运行速度倍率
参数	
返回值	成功：(0, rapidrate)，rapidrate 是速度倍率，返回值是 0 到 1 之间的闭区间 失败：其它

4.6 轨迹复现

4.6.1 设置轨迹复现配置参数

函数	**set_traj_config(xyz_interval, rpy_interval, vel, acc)**
描述	设置轨迹复现配置参数，可设置空间位置采集精度，姿态采集精度，执行脚本运行速度，执行脚本运行加速度。
参数	xyz_interval：空间位置采集速度 rpy_interval：姿态采集精度 vel：执行脚本运行速度 acc：执行脚本运行加速度
返回值	成功：(0,) 失败：其它

代码示例：

```python
import jkrc
import time

#坐标系
COORD_BASE  = 0
COORD_JOINT = 1
COORD_TOOL  = 2
#运动模式
ABS = 0
INCR= 1

robot = jkrc.RC("192.168.2.160")
robot.login()
robot.power_on()
robot.enable_robot()
robot.joint_move(joint_pos =[1,1,1,1,1,1] ,move_mode = 0 ,is_block = True ,speed = 10 )
print("joint")
robot.set_traj_config([0.1, 0.1, 25, 100]) #设置轨迹复现参数，仅录制过程有效
time.sleep(0.1)
ret = robot.get_traj_config()#获取轨迹复现参数
print("traj_config:")
print(ret)
robot.set_traj_sample_mode(True, 'pythonTrack')#开启轨迹复现采集
time.sleep(0.1)
robot.joint_move(joint_pos =[-1,1,1,1,1,1] ,move_mode = 0 ,is_block = True ,speed = 30*3.14/180 )#阻塞运动
robot.joint_move(joint_pos =[1,1,1,1,1,1] ,move_mode = 0 ,is_block = True ,speed = 30*3.14/180 )
# robot.jog(2,INCR,COORD_BASE,10,-2)
# robot.jog(2,INCR,COORD_BASE,10,2)
robot.set_traj_sample_mode(False, 'pythonTrack')#结束轨迹复现采集
time.sleep(1)
res = robot.generate_traj_exe_file('pythonTrack')#将采集到的轨迹复现文件转化为可执行脚本
print(res)
robot.program_load("track/pythonTrack")#加载轨迹程序
time.sleep(0.1)
robot.program_run()
```

4.6.2 获取轨迹复现配置参数

函数	get_traj_config()
描述	获取轨迹复现配置参数，可获得空间位置采集精度，姿态采集精度， 执行脚本运行速度，执行脚本运行加速度。

参数	
返回值	成功：(0 , (xyz_interval, rpy_interval, vel, acc)) xyz_interval：空间位置采集速度 rpy_interval：姿态采集精度 vel：执行脚本运行速度 acc：执行脚本运行加速度 失败：其它

4.6.3 采集轨迹复现数据控制开关

函数	**set_traj_sample_mode(mode, filename)**
描述	采集轨迹复现数据控制开关。
参数	mode：控制模式，True 代表开始采集数据，False 代表结束数据采集。 filename：数据的存储文件名。
返回值	成功：(0,) 失败：其它

4.6.4 采集轨迹复现数据状态查询

函数	**get_traj_sample_status()**
描述	采集轨迹复现数据状态查询。注：在数据采集状态时，不允许再次开启数据采集开关。
参数	
返回值	成功：(0, sample_status), sample_status：数据状态，True 代表正在采集数据，False 代表数据采集结束。 失败：其它

4.6.5 查询控制器中已经存在的轨迹复现数据的文件名

函数	**get_exist_traj_file_name ()**
描述	查询控制器中已经存在的轨迹复现数据的文件名
参数	
返回值	成功：(0,) 失败：其它

4.6.6 重命名轨迹复现数据的文件名

函数	rename_traj_file_name (src, dest)
描述	重命名轨迹复现数据的文件名
参数	src：源文件名 dest：目标文件名，文件名长度不能超过 100 个字符，不能为空，目标文件名不支持中文。
返回值	成功：(0,) 失败：其它

代码示例：

```
import jkrc                                    #导入模块
robot = jkrc.RC("192.168.2.226")       #返回一个机器人对象
robot.login()
robot.rename_traj_file_name('/home/src', '/home/dst')
robot.logout() #登出
```

4.6.7 删除控制器中轨迹复现数据的文件

函数	remove_traj_file(filename)
描述	删除控制器中轨迹复现数据文件
参数	filename：所要删除的文件名
返回值	成功：(0,) 失败：其它

代码示例：

```
import jkrc                                    #导入模块
robot = jkrc.RC("192.168.2.226")  #返回一个机器人对象
robot.login()
robot.remove_traj_file('test')
robot.logout() #登出
```

4.6.8 控制器中轨迹复现数据文件生成控制器执行脚本

函数	generate_traj_exe_file(filename)
描述	控制器中轨迹复现数据文件生成控制器执行脚本
参数	filename：数据的文件名
返回值	成功：(0,) 失败：其它

4.7 机器人运动学

4.7.1 欧拉角到旋转矩阵的转换

函数	**rpy_to_rot_matrix(rpy = [rx,ry,rz])**
描述	欧拉角到旋转矩阵的转换
参数	rpy:待转换的欧拉角数据[rx,ry,rz]
返回值	成功：(0, rot_matrix), rot_matrix 是一个 3x3 的旋转矩阵 失败：其它

4.7.2 旋转矩阵到欧拉角的转换

函数	**rot_matrix_to_rpy(rot_matrix)**
描述	旋转矩阵到欧拉角的转换
参数	rot_matrix: 转换的旋转矩阵数据
返回值	成功：(0, rpy), rpy 是长度为 3 的欧拉角元组(rx,ry,rz) 失败：其它

4.7.3 四元数到旋转矩阵的转换

函数	**quaternion_to_rot_matrix (quaternion = [w,x,y,z])**
描述	四元数到旋转矩阵的转换
参数	quaternion: 待转换的四元数数据
返回值	成功：(0, rot_matrix), rot_matrix 是一个 3x3 的旋转矩阵 失败：其它

4.7.4 求机器人逆解

函数	**kine_inverse(ref_pos, cartesian_pose)**
描述	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。
参数	ref_pos: 关节空间参考位置，建议选用机器人当前关节位置。 cartesian_pose: 笛卡尔空间位姿计算结果.
返回值	成功：(0 , joint_pos) joint_pos 是一个包含 6 位元素的元组 (j1, j2, j3, j4, j5, j6)， j1, j2, j3, j4, j5, j6 分别代表关节 1 到关节 6 的角度值 失败：其它

4.7.5 求机器人正解

函数	kine_forward(joint_pos)
描述	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
参数	joint_pos: 关节空间位置
返回值	成功:(0, cartesian_pose), cartesian_pose 是一个包含 6 位元素的元组(x, y, z, rx, ry, rz)，x, y, z, rx, ry, rz 代表机器人工具末端的位姿值 失败：其它

代码示例：

```
'''机器人正解'''
import jkrc
robot = jkrc.RC("192.168.2.64") #返回一个机器人对象
robot.login()                   #登录
ret = robot.get_joint_position()
if ret[0] == 0:
    print("the joint position is :",ret[1])
else:
    print("some things happend,the errcode is: ",ret[0])
joint_pos = ret[1]
robot.kine_forward(joint_pos)   #求机器人正解
robot.logout()  #登出
```

4.7.6 旋转矩阵到四元数的转换

函数	rot_matrix_to_quaternion(rot_matrix)
描述	旋转矩阵到四元数的转换
参数	rot_matrix: 3x3 的旋转矩阵
返回值	成功：(0, quaternion), quaternion 是长度为 4 的四元数元组(w,x,y,z) 失败：其它

代码示例：

```
import jkrc
robot = jkrc.RC("192.168.2.160")
robot.login()
ret = robot.get_tcp_position()
print(ret)
rpy = [ret[1][3], ret[1][4], ret[1][5]]#获取 rpy
```

```
ret = robot.rpy_to_rot_matrix(rpy)#rpy 转换成旋转矩阵
print(ret)
rot_matrix = ret[1]#获取旋转矩阵
ret = robot.rot_matrix_to_rpy(rot_matrix)#旋转矩阵转换成 rpy
print(ret)
ret = robot.rot_matrix_to_quaternion(rot_matrix)#旋转矩阵转换成四元数
print(ret)
quaternion = ret[1]
ret = robot.quaternion_to_rot_matrix(quaternion)#旋转矩阵转换成四元数
print(ret)
robot.logout()
```

4.8 机械臂伺服运动

4.8.1 机器人 SERVO MOVE 模式使能

函数	servo_move_enable(enable)
描述	机器人 SERVO MOVE 模式使能
参数	enable : TRUE 为进入 SERVO MOVE 模式，FALSE 表示退出该模式
返回值	成功：(0,) 失败：其它

4.8.2 机器人关节空间伺服模式运动

函数	servo_j(joint_pos, move_mode)
描述	机器人关节空间位置控制模式,需要注意的事项如下： (a) 当用户使用这个接口的时候需要先调用 servo_move_enable(True)，进入位置控制模式 (b) 这条指令一般是高校科研当中做轨迹规划时使用。 (c) 当用户使用此模式控制机器人的运动的时候，控制器的规划器不参与运动的插补，位置指令会直接发给伺服，所以用户需要自己进行轨迹规划，否则机器人的运动效果会比较差，如抖动剧烈等现象，无法达到用户的预期。 (d) 由于控制器的控制周期为 8ms，建议用户的发送周期也为 8ms，并且需要连续发送，只发一次没有效果。如果网络状况较差，发送周期可以小于 8ms。 (e) Jaka 机器人的关节速度上限是 180 度/秒，如果发送的关节角度使关节速度超过了此上限。这条指令就会失效。比如发送的关节角度是[1.5,0.5,0.5,0.5,0.5,0.5](此

	处单位是度)，发送周期是 8ms，那么 1.5/0.008= 187.5 度/秒 ，超过了关节速度上限。那么指令将会失效。 (f) 使用完这条指令，需要使用 servo_move_enable(False),退出位置控制模式。 (g) 这条指令和前文提到的 joint_move()区别较大，joint_move 的插补是由控制器进行的，用户无需关心。用户使用 servo_j 指令时需要预先进行轨迹规划，否则使用效果会很差，无法达到预期。**无特殊需要，机器人在关节空间的运动建议使用 joint_move，而不是 servo_j。**
参数	joint_pos: 机器人关节运动目标位置。 move_mode 指定运动模式：0 为绝对运动，1 为增量运动
返回值	成功：(0,) 失败：其它

代码示例：

'''servo_j 的使用'''

```
# -*- coding: utf-8 -*-
import sys
sys.path.append('D:\\vs2019ws\PythonCtt\PythonCtt')
import time
import jkrc
ABS = 0           # 绝对运动
INCR = 1          # 增量运动
Enable = True
Disable = False
robot = jkrc.RC("192.168.2.160")#返回一个机器人对象
robot.login()#登录
robot.power_on() #上电
robot.enable_robot()
robot.servo_move_enable(Enable)  #进入位置控制模式
print("enable")
for i in range(200):
    robot.servo_j(joint_pos =[0.001,0,0,0,0,0.001],move_mode = INCR)#
for i in range(200):
    robot.servo_j(joint_pos =[-0.001,0,0,0,0,-0.001],move_mode = INCR)
robot.servo_move_enable(Disable)#退出位置控制模式
print("disable")
robot.logout()  #登出
```

4.8.3 机器人关节空间伺服模式运动扩展

函数	servo_j_extend(joint_pos, move_mode, setp_num)
描述	(a) 机器人关节空间位置控制模式,需要注意的事项如下： (b) 当用户使用这个接口的时候需要先调用 servo_move_enable(True)，进入位置控制模式 (c) 这条指令一般是高校科研当中做轨迹规划时使用。 (d) 当用户使用此模式控制机器人的运动的时候，控制器的规划器不参与运动的插补，位置指令会直接发给伺服，所以用户需要自己进行轨迹规划，否则机器人的运动效果会比较差，如抖动剧烈等现象，无法达到用户的预期。 (e) 由于控制器的控制周期为 8ms，建议用户的发送周期也为 8ms，并且需要连续发送，只发一次没有效果。如果网络状况较差，发送周期可以小于 8ms。 (f) Jaka 机器人的关节速度上限是 180 度/秒，如果发送的关节角度使关节速度超过了此上限。这条指令就会失效。比如发送的关节角度是[1.5,0.5,0.5,0.5,0.5,0.5](此处单位是度)，发送周期是 8ms，那么 1.5/0.008= 187.5 度/秒 ，超过了关节速度上限。那么指令将会失效。 (g) 使用完这条指令，需要使用 servo_move_enable(False),退出位置控制模式。 (h) 这条指令和前文提到的 joint_move()区别较大，joint_move 的插补是由控制器进行的，用户无需关心。用户使用 servo_j 指令时需要预先进行轨迹规划，否则使用效果会很差，无法达到预期。无特殊需要，机器人在关节空间的运动建议使用 joint_move，而不是 servo_j。
参数	joint_pos: 机器人关节运动目标位置。 move_mode 指定运动模式：0 为增量运动，1 为绝对运动 step_num: 倍分周期，servo_j 运动周期为 step_num*8ms，其中 step_num>=1。
返回值	成功：(0,) 失败：其它

4.8.4 机器人笛卡尔空间伺服模式运动

函数	servo_p(cartesian_pose, move_mode)
描述	机器人笛卡尔空间位置控制模式,需要注意的事项如下： (a) 当用户使用这个接口的时候需要先调用 servo_move_enable(True)，进入位置控制模式 (b) 这条指令一般是高校科研当中做轨迹规划时使用。 (c) 当用户使用此模式控制机器人的运动的时候，控制器的规划器不参与运动的插补，控制器进行逆解后，会将位置指令会直接发给伺服，所以用户需要自己进行轨迹规划，否则机器人的运动效果会比较差，如抖动剧烈等现象，无法达到用户的预期。 (d) 由于控制器的控制周期为 8ms，建议用户的发送周期也为 8ms，并且需要连续发送，只发一次没有效果。如果网络状况较差，发送周期可以小于 8ms。 (e) Jaka 机器人的关节速度上限是 180 度/秒，如果发送的位置使得机器人的关节速

	度超过了此上限。这条指令就会失效。 (f) 使用完这条指令，需要使用 servo_move_enable**(False)**,退出位置控制模式。 (g) 这条指令和前文提到的 linear_move()区别较大，linear_move 的插补是由控制器进行的，用户无需关心。用户使用 **servo_p** 指令时需要预先进行轨迹的规划，否则使用效果会很差，无法达到预期。**无特殊需要，机器人在笛卡尔空间的运动建议使用 linear_move，而不是 servo_p。**
参数	cartesian_pose: 机器人笛卡尔空间运动目标位置。 move_mode: 指定运动模式, 0 为绝对运动，1 为增量运动
返回值	成功：(0,) 失败：其它

代码示例：

'''servo_p 的使用'''

```
# -*- coding: utf-8 -*-
import sys
sys.path.append('D:\\vs2019ws\PythonCtt\PythonCtt')
import time
import jkrc

PI = 3.14

ABS = 0           # 绝对运动

INCR = 1          # 增量运动

Enable = True
Disable = False

robot = jkrc.RC("192.168.2.160")#返回一个机器人对象

robot.login()#登录

robot.power_on() #上电

robot.enable_robot()
joint_pos=[PI/3,PI/3,PI/3,PI/4,PI/4,0]
robot.joint_move(joint_pos,ABS,True,1)

robot.servo_move_enable(Enable)  #进入位置控制模式

print("enable")
for i in range(200):
    robot.servo_p(cartesian_pose = [1, 0, 0, 0, 0, 0],move_mode = INCR)
for i in range(200):
    robot.servo_p(cartesian_pose = [-1,0, 0, 0, 0, 0],move_mode = INCR)
```

```
robot.servo_move_enable(Disable)#退出位置控制模式
print("disable")
robot.logout()  #登出
```

4.8.5 机器人笛卡尔空间伺服模式运动扩展

函数	**servo_p_extend(cartesian_pose, move_mode, step_num)**
描述	(a) 机器人笛卡尔空间位置控制模式,需要注意的事项如下： (b) 当用户使用这个接口的时候需要先调用 servo_move_enable(True)，进入位置控制模式 (c) 这条指令一般是高校科研当中做轨迹规划时使用。 (d) 当用户使用此模式控制机器人的运动的时候，控制器的规划器不参与运动的插补，控制器进行逆解后，会将位置指令会直接发给伺服，所以用户需要自己进行轨迹规划，否则机器人的运动效果会比较差，如抖动剧烈等现象，无法达到用户的预期。 (e) 由于控制器的控制周期为 8ms，建议用户的发送周期也为 8ms，并且需要连续发送，只发一次没有效果。如果网络状况较差，发送周期可以小于 8ms。 (f) Jaka 机器人的关节速度上限是 180 度/秒，如果发送的位置使得机器人的关节速度超过了此上限。这条指令就会失效。 (g) 使用完这条指令，需要使用 servo_move_enable(False),退出位置控制模式。 (h) 这条指令和前文提到的 linear_move()区别较大，linear_move 的插补是由控制器进行的，用户无需关心。用户使用 servo_p 指令时需要预先进行轨迹的规划，否则使用效果会很差，无法达到预期。无特殊需要，机器人在笛卡尔空间的运动建议使用 linear_move，而不是 servo_p。
参数	cartesian_pose: 机器人笛卡尔空间运动目标位置 move_mode: 指定运动模式, 0 为绝对运动，1 为增量运动 step_num: 倍分周期，servo_p 运动周期为 step_num*8ms，其中 step_num>=1
返回值	成功：(0,) 失败：其它

4.8.6 机器人 SERVO 模式下禁用滤波器

函数	**servo_move_use_none_filter()**
描述	机器人 SERVO 模式下禁用滤波器，该指令在 SERVO 模式下不可设置，退出 SERVO 后可设置。
参数	
返回值	成功：(0,) 失败：其它

代码示例：

```
import jkrc                                  #导入模块
robot = jkrc.RC("192.168.2.226")  #返回一个机器人对象
robot.login()
robot.servo_move_use_none_filter()
robot.logout() #登出
```

4.8.7 机器人 SERVO 模式下关节空间一阶低通滤波

函数	servo_move_use_joint_LPF(cutoffFreq)
描述	机器人 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。
参数	cutoffFreq：一阶低通滤波器截止频率。
返回值	成功：(0,) 失败：其它

代码示例：

```
import jkrc                                  #导入模块
robot = jkrc.RC("192.168.2.226")  #返回一个机器人对象
robot.login()
robot.servo_move_use_joint_LPF(0.5)
robot.logout() #登出
```

4.8.8 机器人 SERVO 模式下关节空间非线性滤波

函数	servo_move_use_joint_NLF(max_vr, max_ar, max_jr)
描述	机器人 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。
参数	max_vr：笛卡尔空间姿态变化速度的速度上限值（绝对值）°/s 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 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
返回值	成功：(0,) 失败：其它

代码示例：

```
import jkrc                                  #导入模块
robot = jkrc.RC("192.168.2.226")  #返回一个机器人对象
robot.login()
robot.servo_move_use_joint_NLF(max_vr=2, max_ar=2, max_jr=4)
robot.logout() #登出
```

4.8.9 机器人 SERVO 模式下笛卡尔空间非线性滤波

函数	servo_move_use_carte_NLF(max_vp, max_ap, max_jp, max_vr, max_ar, max_jr)
描述	机器人 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。
参数	max_vp：笛卡尔空间下移动指令速度的上限值（绝对值）mm/s max_ap：笛卡尔空间下移动指令加速度的上限值（绝对值）mm/s^2 max_jp：笛卡尔空间下移动指令加加速度的上限值（绝对值）mm/s^3 max_vr：笛卡尔空间姿态变化速度的速度上限值（绝对值）°/s 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 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
返回值	成功：(0,) 失败：其它

代码示例：

```
import jkrc                                    #导入模块
robot = jkrc.RC("192.168.2.226")  #返回一个机器人对象
robot.login()
robot.servo_move_use_carte_NLF(max_vp=2, max_ap=2, max_jp=4,max_vr=2, max_ar=2, max_jr=4)

robot.logout() #登出
```

4.8.10 机器人 SERVO 模式下关节空间多阶均值滤波

函数	servo_move_use_joint_MMF(max_buf, kp, kv, ka)
描述	机器人 SERVO 模式下关节空间多阶均值滤波，该指令在 SERVO 模式下不可设置，退出 SERVO 后可设置。
参数	max_buf: 均值滤波器缓冲区的大小。 kp：加速度滤波系数。 kv：速度滤波系数 ka：位置滤波系数
返回值	成功：(0,) 失败：其它

代码示例：

```
import jkrc                                    #导入模块
robot = jkrc.RC("192.168.2.226")  #返回一个机器人对象
robot.login()
robot.servo_move_use_joint_MMF(max_buf=20 , kp=0.2 , kv=0.4 ,ka=0.2)
robot.logout() #登出
```

4.8.11 SERVO 模式下速度前瞻参数设置

函数	servo_speed_foresight (max_buf, kp)
描述	SERVO 模式下速度前瞻参数设置
参数	max_buf: 均值滤波器缓冲区的大小。 kp：加速度滤波系数。
返回值	成功：(0,) 失败：其它

4.9 力控机器人扩展

需要在工具末端装载额外的力控传感器，下列接口才能生效

4.9.1 设置传感器品牌

函数	set_torsensor_brand(sensor_brand)
描述	设置传感器品牌，输入数字代表对应的传感器品牌，可选值为 1，2，3 分别代表不同的力矩传感器。
参数	sensor_brand: 传感器品牌所对应的数字
返回值	成功：(0,) 失败：其它

代码示例：

```
import jkrc                                  #导入模块
robot = jkrc.RC("192.168.2.165")  #返回一个机器人对象
robot.login()
robot.set_torsenosr_brand(1)     #1 为 SONY 索尼半导体;2 为 BoschSensortec 博世;3 为 ST 意法半导体
robot.logout() #登出
```

4.9.2 获取传感器品牌

函数	get_torsensor_brand()
描述	获取当前所使用的传感器品牌，输出的数字代表不同的传感器品牌。
参数	
返回值	成功：(0, sensor_brand), sensor_brand: 传感器品牌，可选值为 1，2，3 分别代表不同的力矩传感器。 失败：其它

代码示例：

```
import jkrc                                  #导入模块
robot = jkrc.RC("192.168.2.165")  #返回一个机器人对象
robot.login()
ret=robot.get_torsenosr_brand()
if ret[0] == 1:
    print("the sensor_band is SONY 索尼半导体")
elif ret[0] == 2:
     print("the sensor_band is BoschSensortec 博世")
else:
    print("the sensor_band is ST 意法半导体")
robot.logout() #登出
```

4.9.3 开启或关闭力矩传感器

函数	**set_torque_sensor_mode(sensor_mode)**
描述	开启或关闭当前所使用的力矩传感器。
参数	sensor_mode: 传感器工作模式，0 代表关闭力矩传感器，1 代表开启力矩传感器。
返回值	成功：(0,) 失败：其它

4.9.4 设置柔顺控制参数

函数	**set_admit_ctrl_config(axis, opt, ftUser, ftConstant, ftNormalTrack, ftReboundFK)**
描述	设置机器人柔顺控制时，关节坐标轴编号，柔顺方向，阻尼力，回弹力，恒力，法向跟踪等参数。
参数	axis: 代表笛卡尔空间坐标轴编号，取值为：0~5,分别对应 fx fy fz mx my mz 方向 opt: 柔顺方向, 0 代表关闭，1 代表启动。 ftUser：阻尼力，表示用户需使用多大的力让机器人沿着某一方向以最大速度运动。 ftConstant: 恒力，手动操作时全部设置为 0。 ftNormalTrack：法向跟踪，手动操作时全部设置为 0。 ftReboundFK: 回弹力， 表示机器人回到初始状态的能力。
返回值	成功：(0,) 失败：其它

4.9.5 开始辨识工具末端负载

函数	**start_torq_sensor_payload_identify(joint_pos)**
描述	开始辨识工具末端负载，输入为一个元组含 6 个元素，分别对应结束位置 6 个关节角。
参数	joint_pos: 使用力矩传感器进行自动负载辨识时的结束位置，对应 6 个关节角

返回值	成功：(0,) 失败：其它

代码示例：

```python
# -*- coding: utf-8 -*-
import sys
#sys.path.append('D:\\vs2019ws\PythonCtt\lib')
import time
import jkrc
PI = 3.1415926

robot = jkrc.RC("10.5.5.100")#返回一个机器人对象
ret = robot.login()#登录
ret = robot.power_on()
ret = robot.enable_robot()
robot.set_torsenosr_brand(2)
robot.set_torque_sensor_mode(1)
robot.set_compliant_type(1, 1)
print("inint sensor comple")
print("ready to run")
ret = robot.get_joint_position()
joint_pos_origin = ret[1]
joint_pos = ret[1]
print(joint_pos)
joint_pos[3] += PI / 4
if (joint_pos[3] > 265 * PI / 180):
    joint_pos[3] -= 90
joint_pos[4] += PI / 4
if (joint_pos[4] > 320 * PI / 180):
    joint_pos[4] -= 90
joint_pos[5] += PI / 4
if (joint_pos[5] > 265 * PI / 180):
    joint_pos[5] -= PI
print(joint_pos)
ret = robot.start_torq_sensor_payload_identify(joint_pos)
time.sleep(1)
flag = 1
while (1 == flag):
    ret = robot.get_torq_sensor_identify_staus()
    print(ret)
    time.sleep(1)
    flag = ret[1]
print("identy_finish")
ret = robot.get_torq_sensor_payload_identify_result()
```

```
print(ret)
ret = robot.set_torq_sensor_payload()
print(ret)
ret = robot.get_torq_sensor_payload_identify_result()
print(ret)
robot.joint_move(joint_pos_origin,0,1,10)
print("back")
robot.logout()  #登出
```

4.9.6 获取末端负载辨识状态

函数	**get_torq_sensor_identify_status()**
描述	获取末端负载辨识状态
参数	identify_status: 辨识状态，0 代表辨识完成，1 代表未完成，2 代表辨识失败。
返回值	成功：(0,**identify_status**) 失败：其它

4.9.7 获取末端负载辨识结果

函数	**get_torq_sensor_payload_identify_result()**
描述	获取末端负载辨识结果，输入为负载质量，负载质心位置坐标
参数	mass: 负载质量，单位: kg centroid: 负载质心位置[x, y, z]，单位: mm
返回值	成功：(0,) 失败：其它

4.9.8 设置传感器末端负载

函数	**set_torq_sensor_tool_payload (mass, centroid)**
描述	设置传感器末端负载， 输入为负载质量，负载质心位置坐标。
参数	mass: 负载质量，单位: kg centroid: 负载质心坐标位置[x, y, z]，单位：mm
返回值	成功：(0,) 失败：其它

代码示例：

```
import jkrc                                  #导入模块
robot = jkrc.RC("192.168.2.226")        #返回一个机器人对象
```

```
robot.login()
robot.set_torq_sensor_tool_payload(mass= 1, centroid =[10,10,0])
robot.logout() #登出
```

4.9.9 获取传感器末端负载

函数	get_torq_sensor_tool_payload ()
描述	获取传感器末端负载质量，负载质心位置坐标。
参数	
返回值	成功：(0,(mass, centroid)), mass: 负载质量，单位: kg, centroid: 负载质心位置[x, y, z]，单位: mm 失败：其它

4.9.10 力控导纳使能控制

函数	enable_admittance_ctrl (enable_flag)
描述	力控导纳使能控制
参数	enable_flag: 标志，0 为关闭力控拖拽使能，1 为开启力控导纳使能。
返回值	成功：(0,) 失败：其它

代码示例：

```
# -*- coding: utf-8 -*-
import sys
#sys.path.append('D:\\vs2019ws\PythonCtt\lib')
import time
import jkrc


robot = jkrc.RC("10.5.5.100")#返回一个机器人对象
ret = robot.login()#登录
ret = robot.power_on()
ret = robot.enable_robot()
robot.set_torsenosr_brand(2)
robot.set_torque_sensor_mode(1)
robot.set_compliant_type(1, 1)
print("inint sensor comple")
print("ready to run")
#设置柔顺控制参数
ret = robot.set_admit_ctrl_config(0, 0, 20, 5, 0, 0)
```

```
ret = robot.set_admit_ctrl_config(1, 0, 20, 5, 0, 0)
ret = robot.set_admit_ctrl_config(2, 1, 20, 10, 0, 0)
ret = robot.set_admit_ctrl_config(3, 0, 20, 5, 0, 0)
ret = robot.set_admit_ctrl_config(4, 0, 20, 5, 0, 0)
ret = robot.set_admit_ctrl_config(5, 0, 20, 5, 0, 0)
#设置力控拖拽使能，1 打开，0 关闭
ret = robot.enable_admittance_ctrl(1)
print("enable_admittance_ctrl open! ")
print("input any word to quit:")
a = input()
ret = robot.enable_admittance_ctrl(0)
ret = robot.set_admit_ctrl_config(2, 0, 20, 5, 0, 0)
robot.set_torque_sensor_mode(0)
robot.logout()  #登出
```

4.9.11 设置力控类型和传感器初始化状态

函数	set_compliant_type(sensor_compensation, compliance_type)
描述	设置力控类型和传感器初始化状态
参数	sensor_compensation: 开启传感器补偿标志，1 代表开启初始化，0 代表不初始化 compliance_type: 力控类型，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
返回值	成功：(0,) 失败：其它

代码示例：

```
import jkrc                                  #导入模块
robot = jkrc.RC("192.168.2.226")             #返回一个机器人对象
robot.login()
robot.set_compliant_type(1,1)
robot.logout() #登出
```

4.9.12 获取力控类型和传感器初始化状态

函数	get_compliant_type()
描述	获取力控类型和传感器初始化状态
参数	
返回值	成功：(0, sensor_compensation, compliance_type) sensor_compensation: 开启传感器补偿标志，1 代表开启初始化，0 代表不初始化 compliance_type: 力控类型，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 失败：其它

4.9.13 获取力控柔顺控制参数

函数	**get_admit_ctrl_config()**
描述	获取力控柔顺控制参数，获取 6 个关节所对应的柔顺方向，阻尼力，回弹力，恒力，法向跟踪。
参数	
返回值	成功：(0, [[opt, ftUser, ftReboundFK, ftConstant, ftNormanlTrack], ……]) opt: 柔顺方向，可选值 1 2 3 4 5 6 分别对应 fx fy fz mx my mz 方向，0 代表没有勾选。 ftUser：阻尼力，表示用户需使用多大的力让机器人沿着某一方向以最大速度运动。 ftReboundFK: 回弹力， 表示机器人回到初始状态的能力。 ftConstant: 恒力，手动操作时全部设置为 0。 ftNormalTrack：法向跟踪，手动操作时全部设置为 0。 失败：其它

4.9.14 设置力矩传感器 IP 地址

函数	**set_torque_sensor_comm(type, ip_addr, port)**
描述	设置力矩传感器 IP 地址。
参数	type: 传感器类型 ip_addr: 传感器 ip 地址 port: 端口号
返回值	成功：(0,) 失败：其它

4.9.15 获取力矩传感器 IP 地址

函数	**get_torque_sensor_comm()**
描述	获取力矩传感器 IP 地址。
参数	
返回值	成功：(0, ip_addr), ip_addr: 传感器 ip 地址。 失败：其它

4.9.16 关闭力矩控制

函数	**disable_force_control ()**

描述	关闭力矩控制
参数	
返回值	成功：(0,) 失败：其它

4.9.17 设置速度柔顺控制参数

函数	**set_vel_compliant_ctrl (level, rate1, rate2, rate3, rate4)**
描述	设置速度柔顺控制参数，速度柔顺控制有 3 个等级，有 4 个比率等级。
参数	level：柔顺控制等级，等级分为：1, 2, 3 rate1：比率等级 1 rate2：比率等级 2 rate3：比率等级 3 rate4：比率等级 4 注： 比率等级之间的关系： 0<rate4<rate3<rate2<rate1<1; (a) 当 level=1 时，只能设置 rate1, rate2，而 rate3, rate4 的值全为 0 (b) 当 level=2 时，只能设置 rate1, rate2, rate3，而 rate4 的值为 0 (c) 当 level=3 时，能设置 rate1, rate2, rate3, rate4 的值
返回值	成功：(0,) 失败：其它

4.9.18 设置柔顺控制力矩条件

函数	**set_compliance_condition (fx, fy, fz, tx, ty, tz)**
描述	设置柔顺控制力矩条件。
参数	fx：沿着 x 轴受力，单位：N fy：沿着 y 轴受力，单位：N fz：沿着 z 轴受力，单位：N tx：绕着 x 轴的扭矩，单位：Nm ty：绕着 y 轴的扭矩，单位：Nm tz：绕着 z 轴的扭矩，单位：Nm
返回值	成功：(0,) 失败：其它

4.9.19 设置力控的低通滤波器参数

函数	**`set_torque_sensor_filter(HZ);`**
描述	设置柔顺控制力矩条件。

参数	HZ: 低通滤波器参数，浮点数 单位：HZ
返回值	成功：(0,) 失败：其它

4.9.20 获取力控的低通滤波器参数

函数	**get_torque_sensor_filter();**
描述	设置柔顺控制力矩条件。
参数	HZ: 低通滤波器参数，单位：HZ
返回值	成功：(0,HZ) 失败：其它

4.9.21 设置力传感器的传感器限位参数配置

函数	**set_torque_sensor_soft_limit(fx, fy, fz, tx, ty, tz);**
描述	设置柔顺控制力矩条件。
参数	fx: 沿着 x 轴受力，单位：N fy: 沿着 y 轴受力，单位：N fz: 沿着 z 轴受力，单位：N tx: 绕着 x 轴的扭矩，单位：Nm ty: 绕着 y 轴的扭矩，单位：Nm tz: 绕着 z 轴的扭矩，单位：Nm
返回值	成功：(0,) 失败：其它

4.9.22 获取力传感器的传感器限位参数配置

函数	**get_torque_sensor_soft_limit();**
描述	设置柔顺控制力矩条件。
参数	
返回值	成功：(0,(fx,fy,fz,tx,ty,tz)) fx: 沿着 x 轴受力，单位：N fy: 沿着 y 轴受力，单位：N fz: 沿着 z 轴受力，单位：N tx: 绕着 x 轴的扭矩，单位：Nm ty: 绕着 y 轴的扭矩，单位：Nm tz: 绕着 z 轴的扭矩，单位：Nm 失败：其它

4.9.23 传感器校零

函数	zero_end_sensor();
描述	传感器校零。
参数	
返回值	成功：(0,) 失败：其它

4.9.24 获取力控工具拖拽开启状态

函数	get_tool_drive_state();
描述	获取工具拖拽坐标系
参数	无
返回值	成功：(0,enable_flag, drive_state) enable_flag: 0 为关闭力控拖拽使能，1 为开启，drive_stat 是拖拽的当前状态是否触发奇异点、速度、关节限位预警 失败：其它

4.9.25 获取工具拖拽坐标系

函数	get_tool_drive_frame();
描述	获取工具拖拽坐标系
参数	无
返回值	成功：(0,ftFrame) ftFrame: 0 工具； 1 世界 失败：其它

4.9.26 设置工具拖拽坐标系

函数	set_tool_drive_frame(ftFrame);
描述	设置工具拖拽坐标系
参数	ftFrame: 0 工具； 1 世界
返回值	成功：(0,) 失败：其它

4.9.27 获取融合拖拽灵敏度

函数	get_fusion_drive_sensitivity_level();
描述	获取融合拖拽灵敏度

参数	无
返回值	成功：(0,level) level: 灵敏度等级，值 0-5， 0 为关 失败：其它

4.9.28 设置融合拖拽灵敏度

函数	`set_fusion_drive_sensitivity_level(level);`
描述	设置融合拖拽灵敏度
参数	level: 灵敏度等级，值 0-5， 0 为关
返回值	成功：(0,) 失败：其它

4.9.29 获取运动限制（奇异点和关节限位）预警范围

函数	`get_motion_limit_warning_range(warningRange);`
描述	获取运动到限制(奇异点和关节限位)预警范围。
参数	无
返回值	成功：(0,warningRange) **`warningRange: 预警范围`** 失败：其它

4.9.30 设置运动限制（奇异点和关节限位）预警范围

函数	`set_motion_limit_warning_range(warningRange);`
描述	设置运动到限制(奇异点和关节限位)预警范围。
参数	**`warningRange: 预警范围`**
返回值	成功：(0,) 失败：其它

4.9.31 获取力控限速

函数	`get_compliant_speed_limit(vel, angularvel);`
描述	获取力控限速。
参数	无
返回值	成功：(0,vel,angularvel) vel: 线速度限制，mm/s: **`angularvel: 角速度限制， rad/s`** 失败：其它

4.9.32 设置力控限速

函数	set_compliant_speed_limit(vel, angularvel);
描述	设置力控限速。
参数	vel: 线速度限制，mm/s: angularvel: 角速度限制， rad/s
返回值	成功：(0,) 失败：其它

4.9.33 获取力矩参考中心

函数	get_torque_ref_point();
描述	获取力矩参考中心。
参数	无
返回值	成功：(0,refpoint) refpoint: 0: 传感器中心； 1: TCP 中心 失败：其它

4.9.34 设置力矩参考中心

函数	set_torque_ref_point(refpoint);
描述	设置力矩参考中心 。
参数	refpoint: 0: 传感器中心； 1: TCP 中心
返回值	成功：(0,) 失败：其它

4.9.35 获取传感器灵敏度

函数	get_end_sensor_sensitivity_threshold ();
描述	获取传感器灵敏度。
参数	无
返回值	成功：(0,data) fx: 沿着 x 轴受力，单位：N fy: 沿着 y 轴受力，单位：N fz: 沿着 z 轴受力，单位：N tx: 绕着 x 轴的扭矩，单位：Nm ty: 绕着 y 轴的扭矩，单位：Nm tz: 绕着 z 轴的扭矩，单位：Nm 失败：其它

4.9.36 设置传感器灵敏度

函数	`set_end_sensor_sensitivity_threshold` **(fx, fy, fz, tx, ty, tz);**
描述	设置传感器灵敏度。
参数	fx: 沿着 x 轴受力，单位：N fy: 沿着 y 轴受力，单位：N fz: 沿着 z 轴受力，单位：N tx: 绕着 x 轴的扭矩，单位：Nm ty: 绕着 y 轴的扭矩，单位：Nm tz: 绕着 z 轴的扭矩，单位：Nm 6 维数组，0~1，越大传感器越不灵敏
返回值	成功：(0,data) 失败：其它

4.10 FTP 服务

4.10.1 FTP 初始化

函数	**init_ftp_client()**
描述	初始化 ftp 客户端，与控制柜建立连接，可导入导出 program、track
参数	none
返回值	成功：(0,) 失败：其它

4.10.2 FTP 关闭

函数	**close_ftp_client()**
描述	关闭 ftpclient
参数	none
返回值	成功：(0,) 失败：其它

4.10.3 查询控制器 FTP 的目录

函数	**get_ftp_dir(remotedir, type);**
描述	查询控制器目录

参数	remotedir： 控制器内部文件夹名称 type： 0 文件和文件夹 1 文件 2 文件夹
返回值	成功：(0,ret)ret 为字符串 失败：其它

代码示例：

```
import jkrc
robot = jkrc.RC("192.168.2.26")
robot.login()
dir= "/program/"
#登陆控制器，需要将 192.168.2.26 替换为自己控制器的 IP
robot.init_ftp_client()
result = robot.get_ftp_dir("/program/", 0)
print(result)
robot.close_ftp_client()
robot.logout()
```

4.10.4 下载 FTP 文件

函数	**download_file(local, remote, opt)**
描述	从机器人控制柜的 ftp 下载文件或文件夹到本地，查询轨迹“/track/”,查询脚本程序“/program/”
参数	remote 控制器内部文件名绝对路径，文件夹需要以"\"或“/”结尾取决于系统 如单个文件“/program/test/test.jks”或者文件夹“/program/test/” local 下载到本地文件名绝对路径 opt 1 单个文件 2 文件夹
返回值	成功：(0,) 失败：其它

代码示例：将 ftp 端的 program 文件夹下载到桌面 program 文件夹内

```
import jkrc
robot = jkrc.RC("192.168.2.26")#VMmodel
robot.login()
remote = "/program/"
local= "C:\\Users\\Administrator\\Desktop\\program\\track\\"
robot.init_ftp_client()
result = robot.download_file(local, remote, 2)
print(result)
robot.close_ftp_client()
robot.logout()
```

4.10.5 上传文件到 FTP

函数	upload_file(local, remote, opt)
描述	从本地上传指定类型和名称的文件到控制器
参数	remote 上传到控制器内部文件名绝对路径，文件夹需要以"\"或“/”结尾取决于系统 local 本地文件名绝对路径 opt 1 单个文件 2 文件夹
返回值	成功：(0,) 失败：其它

代码示例：将桌面的 lxxpro 文件夹内的所有文件和文件夹上传到 ftp 的 program/文件夹下

```
1.  import jkrc
2.  robot = jkrc.RC("192.168.2.26")#VMmodel
3.  robot.login()
4.  remote = "/program/"
5.  local= "C:\\Users\\Administrator\\Desktop\\lxxpro\\"
6.  robot.init_ftp_client()
7.  result = robot.upload_file(local, remote, 2)
8.  print(result)
9.  robot.close_ftp_client()
10. robot.logout()
```

4.10.6 重命名 FTP 上的文件

函数	upload_file(local, remote, opt)
描述	重命名控制器指定类型和名称的文件
参数	remote 控制器内部文件名绝对路径，文件夹需要以"\"或“/”结尾取决于系统 des 重命名的目标名 opt 1 单个文件 2 文件夹，重命名文件时会重命名文件夹内所有文件，便于/track/使用
返回值	成功：(0,) 失败：其它

代码示例：将 ftp 的 lxxpro 文件夹内的所有文件和文件夹重命名为 lxx

```
1.  import jkrc
2.
3.  robot = jkrc.RC("192.168.2.26")#VMmodel
4.  robot.login()
5.  remote = "/lxxpro/"
6.  des = "lxx"
7.  robot.init_ftp_client()
8.  result = robot.rename_ftp_file(remote, des, 2)
9.  print(result)
```

```
robot.close_ftp_client()
robot.logout()
```

4.10.7 删除文件到 FTP

函数	del_ftp_file(remote, opt)
描述	从控制器删除指定类型和名称的文件
参数	remote 控制器内部文件名绝对路径，文件夹需要以"\"或“/”结尾取决于系统 opt 1 单个文件 2 文件夹
返回值	成功：(0,) 失败：其它

代码示例：谨慎操作，该 demo 会删除所有程序！！！！

```
import jkrc

robot = jkrc.RC("192.168.2.26")#VMmodel
robot.login()
dir= "/program/"
robot.init_ftp_client()
result = robot.del_ftp_file("/program/", 2)
print(result)
robot.close_ftp_client()
robot.logout()
```

5. 反馈和勘误

文档中如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。