C 函数使用手册

Version 1.0.1

遨博（北京）智能科技有限公司

使用手册会定期进行检查和修正，更新后的内容将出现在新版本中。本手册中的内容或信息如有变更，恕不另行通知。
对本手册中可能出现的任何错误或遗漏，或因使用本手册及其中所述产品而引起的意外或间接伤害，遨博（北京）智能科技有限公司概不负责。
安装、使用产品前，请阅读本手册。
请保管好本手册，以便可以随时阅读和参考。

目录

简介

AUBO API 是基于网络实现的，提供了大量用于操作机械臂的接口，包括登录与退出。本文件定义了使用机械臂的接口类，是基于 C 开发的，其中类中的方法是操作机械臂的接口。

这些接口可被应用来实现 2D 智能相机的使用、3D 视觉集成、码垛示例的实现、软件控制力控、工具端控制力控等这些功能。

本手册包含了五个部分。第 1 章介绍数据类型，第 2 章介绍了接口函数的定义，第 3 章介绍了错误码，第 4 章介绍了接口函数的使用案例，第 5 章介绍了配置环境。

注意：接口中关于长度的单位都为米，关于角度的单位都为弧度。

1 数据类型

1.1 机械臂运动相关的类型

```
/**
 * @brief 位置信息
 **/
struct Pos
{
    double x;
    double y;
    double z;
};
```

```
/**
 * @brief 姿态的四元素表示方法
 **/
struct Ori
{
    double w;
    double x;
    double y;
    double z;
};
```

```
/**
 * @brief 姿态的欧拉角表示方法
 **/
struct Rpy
{
    double rx;
    double ry;
    double rz;
};
```

```
typedef struct
{
    double jointPos[ARM_DOF];
}JointParam;
```

```
/**
  * @brief 描述关节的速度和加速度
```

```
  */
typedef  struct
{
    double  jointPara[ARM_DOF];
}JointVelcAccParam;
```

```
/**
 * @brief 机械臂的路点信息
 **/
typedef  struct
{
    cartesianPos_U    cartPos;              // 机械臂的位置信息(x,y,z)
    Ori               orientation;          // 机械臂姿态信息,四元素(w,x,y,z)
    double            jointpos[ARM_DOF];    // 机械臂关节角信息
}wayPoint_S;
```

```
/**
 *  @brief 描述运动属性中的偏移属性
 */
typedef  struct
{
    bool  ena;                          // 是否使能偏移
    float  relativePosition[3];         // 偏移量 x,y,z
    Ori   relativeOri;                  // 姿态偏移
}MoveRelative;
```

```
/**
 * @brief 示教模式枚举
 **/
enum  teach_mode
{
    NO_TEACH = 0,
    JOINT1,
    JOINT2,
    JOINT3,
    JOINT4,
    JOINT5,
    JOINT6,
    MOV_X,
    MOV_Y,
    MOV_Z,
    ROT_X,
    ROT_Y,
    ROT_Z
```

```
};
```

```
/**
 * @brief 运动轨迹枚举
 **/
enum move_track
{
    NO_TRACK = 0,

    //for moveJ and moveL
    TRACKING,

    //cartesian motion for moveP
    ARC_CIR,
    CARTESIAN_MOVEP,
    CARTESIAN_CUBICSPLINE,
    CARTESIAN_UBSPLINEINTP,
    CARTESIAN_GNUBSPLINEINTP,
    CARTESIAN_LOOKAHEAD,

    //joint motion for moveP
    JIONT_CUBICSPLINE,
    JOINT_UBSPLINEINTP,
    JOINT_GNUBSPLINEINTP,

    ARC,
    CIRCLE,
    ARC_ORI_ROTATED,
    CIRCLE_ORI_ROTATED,

    ORI_POSITION_ROTATE_CIRCUMFERENCE=101,
};
```

```
typedef struct
{
    double jointMaxAcc[ARM_DOF];      //关节型运动的最大加速度
    double jointMaxVelc[ARM_DOF];     //关节型运动的最大速度
    double endMaxLineAcc;             //末端型运动的最大加速度
    double endMaxLineVelc;            //末端型运动的最大速度
    MoveRelative relative;            //偏移参数
    CoordCalibrateByJointAngleAndTool relativeOnCoord;//偏移量基于那个坐标系
    double blendRadius;               //交融半径
    ToolInEndDesc toolInEndDesc;      //工具属性
}MoveProfile_t;
```

```
/**
 * @brief 旋转运动的转轴
 **/
typedef struct{
    double rotateAxis[3];
}Move_Rotate_Axis;
```

```
/**
 * @brief 逆解
 **/
typedef struct{
    wayPoint_S waypoint[8];
    int solution_count;
}ik_solutions;
```

1.2 机械臂关节状态

```
/**
 * 描述机械臂的关节状态
 */
typedef struct PACKED
{
    int     jointCurrentI;          // 关节电流 Current of driver
    int     jointSpeedMoto;         // 关节速度 Speed of driver
    float   jointPosJ;      // 关节角 Current position in radian
    float   jointCurVol;    // 关节电压 Rated voltage of motor. Unit: mV
    float   jointCurTemp;       // 当前温度 Current temprature of joint
    int     jointTagCurrentI;   // 电机目标电流 Target current of motor
    float   jointTagSpeedMoto;// 电机目标速度 Target speed of motor
    float   jointTagPosJ;       // 目标关节角 Target position of joint in radian
    uint16  jointErrorNum;      // 关节错误码 Joint error of joint num
}JointStatus;
```

1.3 诊断信息

```
/**
 * 机械臂诊断信息
 **/
typedef struct PACKED
{
    uint8   armCanbusStatus; // CAN 通信状态:0x01~0x80：关节 CAN 通信错误
                             // （每个关节占用 1bit） 0x00：无错误
```

```
    float     armPowerCurrent;              // 机械臂 48V 电源当前电流
    float     armPowerVoltage;              // 机械臂 48V 电源当前电压
    bool      armPowerStatus;               // 机械臂 48V 电源状态（开、关）
    char      contorllerTemp;               // 控制箱温度
    uint8     contorllerHumidity;           // 控制箱湿度
    bool      remoteHalt;                   // 远程关机信号
    bool      softEmergency;                // 机械臂软急停
    bool      remoteEmergency;              // 远程急停信号
    bool      robotCollision;               // 碰撞检测位
    bool      forceControlMode;             // 机械臂进入力控模式标志位
    bool      brakeStuats;                  // 刹车状态
    float     robotEndSpeed;                // 末端速度
    int       robotMaxAcc;                  // 最大加速度
    bool      orpeStatus;                   // 上位机软件状态位
    bool      enableReadPose;               // 位姿读取使能位
    bool      robotMountingPoseChanged;     // 安装位置状态
    bool      encoderErrorStatus;           // 磁编码器错误状态
    bool      staticCollisionDetect;        // 静止碰撞检测开关
    uint8     jointCollisionDetect;         // 关节碰撞检测 每个关节占用 1bit
                                            // 0-无碰撞 1-存在碰撞
    bool      encoderLinesError;  // 光电编码器不一致错误：0-无错误，1-有错误
    bool      jointErrorStatus;             // joint error status
    bool      singularityOverSpeedAlarm;    // 机械臂奇异点过速警告
    bool      robotCurrentAlarm;            // 机械臂电流错误警告
    uint8     toolIoError;                  // tool error
    bool      robotMountingPoseWarning;     // 机械臂安装位置错位
                                            //（只在力控模式下起作用）
    uint16    macTargetPosBufferSize;       // mac 缓冲器长度，预留
    uint16    macTargetPosDataSize;         // mac 缓冲器有效数据长度，预留
    uint8     macDataInterruptWarning;      // mac 数据中断，预留
    uint8     controlBoardAbnormalStateFlag;    //主控板(接口板)异常状态标志
}RobotDiagnosis;
```

1.4 回调函数类型

```
/**
 * @brief                获取实时关节状态回调函数类型
 * @param jointStatus    当前的关节状态;
 * @param size           上一个参数（jointStatus）的长度;
 * @param arg            使用者在注册回调函数中传递的第二个参数;
 */
typedef void (*RealTimeJointStatusCallback) (const aubo_robot_namespace::JointSt
atus *jointStatus, int size, void *arg);
```

```
/**
 * @brief                    获取实时路点信息的回调函数类型
 * @param Waypoint           当前的路点信息；
 * @param arg                使用者在注册回调函数中传递的第二个参数；
 */
typedef void (*RealTimeRoadPointCallback) (const aubo_robot_namespace::Waypoint_S *Waypoint, void *arg);
```

```
/**
 *@brief                     获取实时末端速度的回调函数类型
 *@param speed               当前的末端速度;
 *@param arg                 使用者在注册回调函数中传递的第二个参数;
 */
typedef void (*RealTimeEndSpeedCallback) (double speed, void *arg);
```

```
/**
 *@brief                     获取实时 Movep 执行进度的回调函数类型
 *@param num                 当前的 Movep 执行进度;
 *@param arg                 使用者在注册回调函数中传递的第二个参数;
 */
typedef void (*RealTimeMovepStepNumNotifyCallback) (int num, void *arg);
```

```
/**
 * @brief                    获取机械臂事件信息的回调函数类型
 * @param arg                使用者在注册回调函数中传递的第二个参数;
 */
typedef void (*RobotEventCallback) (const aubo_robot_namespace::RobotEventInfo *eventInfo, void *arg);
```

```
/**
 * @brief                    日志输出对应的回调函数类型
 * @param logLevel           日志级别;
 * @param str                日志信息;
 */
typedef void (*RobotLogPrintCallback) (aubo_robot_namespace::LOG_LEVEL logLevel, const char *str, void *arg);
```

1.5 工具参数类型

```
/** 工具描述，工具的运动学参数
 *
 *   该工具用于描述一个工具或者工具的运动学参数
 */
```

```
typedef  struct
{
    Pos          toolInEndPosition;      // 工具相对法兰盘的位置
    Ori          toolInEndOrientation;   // 工具相对法兰盘的姿态
}ToolInEndDesc;
```

```
typedef  ToolInEndDesc  ToolKinematicsParam;  //运动学参数
```

```
/**
 * 工具动力学参数描述
 *
 * 注意：
 *          机械臂上电之前，安装在机器人末端的工具发生改变时都需要重新
 * 设置工具的动力学参数。
 *          一般情况下，工具的动力学参数和运动学参数是需要一起设置的；
 * 切记：
 *          该参数如果不能正确设置会影响机械臂的安全等级和运动轨迹。
 **/
typedef  struct
{
    double        positionX;     // 工具重心的 X 坐标
    double        positionY;     // 工具重心的 Y 坐标
    double        positionZ;     // 工具重心的 Z 坐标
    double        payload;       // 工具重量
    ToolInertia  toolInertia;    // 工具惯量，预留，使用是全部设置为 0
}ToolDynamicsParam;
```

1.6 工具标定

```
typedef  struct
{
    int  posCalibrateNum ;                          //用于位置标定点的数量
    wayPoint_S  posCalibrateWaypoint[4];        //位置标定点
    int  oriCalibrateNum;    //用于姿态标定点的数量
    wayPoint_S  oriCalibrateWaypoint[3];                    //姿态标定点
    ToolKinematicsOriCalibrateMathod    CalibrateMathod;      //姿态标定方法

}ToolCalibrate;
```

1.7 坐标系标定

```
/**
```

```
* 坐标系描述
*
* 该结构体描述一个坐标系。系统通过该结构体描述一个坐标系(基座坐标系,
* 用户坐标系，末端坐标系或工具坐标系)。
*
* 坐标系分 3 种类型:      基座坐标系(BaseCoordinate);
*                         用户坐标系(WorldCoordinate);
*                         末端坐标系或工具坐标系(EndCoordinate);
*
* 定义:
*       基座坐标系是根据机械臂基座建立的坐标系;
*       用户坐标系是用户坐标系定义在工件上，在机器人动作允许范围内的任
* 意位置，设定任意角度的 X、Y、Z 轴，原点位于机器人抓取的工件上，坐标
* 系的方向根据客户需要任意定义。
*       末端坐标系是安装在机器人末端的工具坐标系，原点及方向都是随着末
* 置与角度不断变化的，该坐标系实际是将基础坐标系通过旋转及位移变化而
* 来的；法兰盘是一个特殊的末端坐标系。
*
* 结构体参数描述：
*     coordType：坐标系类型,描述坐标系属于那种类型
*     methods：用户坐标系的标定方法，仅在 coordType 为用户坐标系(WorldC
* oordinate)时有效；
*     wayPointArray[3]：标定用户坐标系的 3 个路点信息，仅在 coordType 为
* 用户坐标系(WorldCoordinate)时有效；
*     toolDesc：末端工具描述，当 coordType＝WorldCoordinate，表示标定用
* 户坐标系时，安装在机器人末端的工具;当 coordType＝EndCoordinate，描述是
* 哪个工具的坐标系
*
* 使用说明:
*     基座坐标系：coordType=BaseCoordinate，其他参数默认
*     用户坐标系：  coordType=WorldCoordinate
*                   methods：为标定方法
*                   wayPointArray[3]：为标定坐标系的 3 个路点
*                   toolDesc：标定用户坐标系时，安装在机器人末端的工具
*     末端坐标系或工具坐标系：  coordType=EndCoordinate
*                               methods：缺省，不需要设置
*                               wayPointArray[3]：缺省，不需要设置
*                               toolDesc：机器人末端的工具
*
* 备注：
*     法兰盘为特殊的工具,工具描述中的位置设置为(0,0,0)，
* 姿态信息设置为(1,0,0,0)
*     其结构体定义为
*       {
```

```
 *              pos{0,0,0},
 *              Ori{1,0,0,0}
 *           }
 *
 *      该结构同时用于用户坐标系的标定．一般通过示教 3 个示教点实现，第一
 * 个示教点是用户坐标系的原点；第二个和第三个示教点的选择根据标定方法
 * 来确定,遵循右手手法．
 */
typedef  struct
{
    coordinate_refer        coordType;          // 坐标系类型
    CoordCalibrateMathod    methods;            // 用户坐标系的标定方法
    JointParam              wayPointArray[3];   // 用于标定用户坐标系的 3 个点
                                                // (关节角)
    ToolInEndDesc           toolDesc;           // 工具描述
}CoordCalibrateByJointAngleAndTool;
```

```
typedef  CoordCalibrateByJointAngleAndTool  CoordCalibrate;  //坐标系描述
```

1.8 IO 相关数据类型

```
/**
 * @brief  IO 的类型枚举
 *
 **/
typedef  enum
{
    RobotBoardControllerDI,     // 接口板控制器 DI(数字量输入)，
                                // 只读(一般系统内部使用)
    RobotBoardControllerDO,     // 接口板控制器 DO(数字量输出)，
                                // 只读(一般系统内部使用)
    RobotBoardControllerAI,     // 接口板控制器 AI(模拟量输入)，
                                // 只读(一般系统内部使用)
    RobotBoardControllerAO,     // 接口板控制器 AO(模拟量输出)，
                                // 只读(一般系统内部使用)

    RobotBoardUserDI,           // 接口板用户 DI(数字量输入)，可读可写
    RobotBoardUserDO,           // 接口板用户 DO(数字量输出)，可读可写
    RobotBoardUserAI,           // 接口板用户 AI(模拟量输入)，可读可写
    RobotBoardUserAO,           // 接口板用户 AO(模拟量输出)，可读可写

    RobotToolDI,                // 工具端 DI
    RobotToolDO,                // 工具端 DO
    RobotToolAI,                // 工具端 AI
```

```
    RobotToolAO,                // 工具端 AO
}RobotIoType;
```

```
/**
 * IO 类型
 **/
typedef enum
{
    IO_IN = 0,              //输入
    IO_OUT                  //输出
}ToolIOType;
```

```
/**
 * 工具的电源类型
 **/
typedef enum
{
    OUT_0V  = 0,
    OUT_12V = 1,
    OUT_24V = 2
}ToolPowerType;
```

```
typedef enum                        //ＩＯ状态
{
    IO_STATUS_INVALID = 0,      //有效
    IO_STATUS_VALID             //无效
}IO_STATUS;
```

```
typedef enum
{
    TOOL_DIGITAL_IO_0 = 0,
    TOOL_DIGITAL_IO_1 = 1,
    TOOL_DIGITAL_IO_2 = 2,
    TOOL_DIGITAL_IO_3 = 3
}ToolDigitalIOAddr;
```

```
/**
  * 综合描述一个 IO
  **/
typedef struct PACKED
{
    char            ioId[32];           //IO-ID 目前未使用
    RobotIoType     ioType;             //IO 类型
```

```
    char            ioName[32];     //IO 名称
    int             ioAddr;         //IO 地址
    double          ioValue;        //IO 状态
}RobotIoDesc;
```

2 接口函数

2.1 机械臂系统接口

2.1.1 登录

int rs_login(RSHD rshd, const char * addr, int port);	
功能描述:	登录，与机械臂服务器建立网络连接。该接口的成功是调用其他接口的前提，只有在该接口正确返回的情况下，才能使用其他接口。
参数说明:	1. rshd：机械臂控制上下文句柄。 2. addr：机械臂服务器的 IP 地址。 3. port：机械臂服务器的端口号，默认为 8899。
返回值:	成功: 返回 RS_SUCC。
	失败: 返回错误号。

2.1.2 退出登录

int rs_logout(RSHD rshd);	
功能描述:	退出登录，断开与机械臂服务器的连接。
参数说明:	rshd：机械臂控制上下文句柄。
返回值:	成功: 返回 RS_SUCC。
	失败: 返回错误号。

2.1.3 启动机械臂

int rs_robot_startup(RSHD rshd, const ToolDynamicsParam *tool_dynamics, uint8 colli_class, bool read_pos, bool static_colli_detect, int board_maxacc, ROBOT_SERVICE_STATE *state);	
功能描述:	启动机械臂，即初始化-----该操作会完成机械臂的上电，松刹车，设置碰撞等级，设置动力学参数等功能。
参数说明:	1. rshd：机械臂控制上下文句柄。 2. tool_dynamics：动力学参数，如果末端夹持工具，此参数应该根据具体的来设定；如果末端没有夹持工具，将此参数的各项设置为 0。 3. colli_class：碰撞等级。 4. read_pos：是否允许读取位置，默认是 true。

	5. static_colli_detect：是否允许侦测静态碰撞，默认为 true。 6. board_maxacc：接口板允许的最大加速度，默认为 1000。 7. state：传出参数，机械臂启动状态，具体参考 ROBOT_SERVICE_STATE 类型。机械臂启动结果只有 result == ROBOT_SERVICE_WORKING 表示机械臂启动成功，否则表示启动失败。
返回值:	成功: 返回 RS_SUCC。
	失败: 返回错误号。

2.1.4 关机

int rs_robot_shutdown(RSHD rshd);	
功能描述:	机械臂断电。
参数说明:	rshd：机械臂控制上下文句柄。
返回值:	成功: 返回 RS_SUCC。
	失败: 返回错误号。

2.1.5 初始化机械臂控制库

int rs_initialize(void);	
功能描述:	初始化机械臂控制库。
参数说明:	无。
返回值:	成功: 返回 RS_SUCC。
	失败: 返回错误号。

2.1.6 反初始化机械臂控制

int rs_uninitialize(void);	
功能描述:	反初始化机械臂控制库。
参数说明:	无。
返回值:	成功: 返回 RS_SUCC。
	失败: 返回错误号。

2.1.7 创建机械臂控制上下文句柄

int rs_create_context(RSHD *rshd);	
功能描述:	创建机械臂控制上下文句柄。
参数说明:	rshd：机械臂控制上下文句柄。
返回值:	成功: 返回 RS_SUCC。

	失败: 返回错误号。

2.1.8 注销机械臂控制上下文句柄

int rs_destory_context(RSHD rshd);	
功能描述:	注销机械臂控制上下文句柄。
参数说明:	rshd：机械臂控制上下文句柄。
返回值:	成功: 返回 RS_SUCC。
	失败: 返回错误号。

2.2 状态推送

2.2.1 设置是否允许实时关节状态推送

int rs_enable_push_realtime_joint_status(RSHD rshd, bool enable);	
功能描述:	设置是否允许实时关节状态推送。
参数说明:	1. rshd：机械臂控制上下文句柄。 2. enable：true 表示允许，false 表示不允许。
返回值:	成功: 返回 RS_SUCC。
	失败: 返回错误号。

2.2.2 注册用于获取关节状态的回调函数

int rs_setcallback_realtime_joint_status(RSHD rshd, const RealTimeJointStatusCallback ptr, void *arg);	
功能描述:	注册用于获取关节状态的回调函数。 注册回调函数后，服务器实时推送当前的关节状态信息。 频率 30ms。
参数说明:	1. rshd：机械臂控制上下文句柄。 2. ptr：获取实时关节状态信息的回调函数指针。 3. arg：这个参数系统不做任何处理，只是进行了缓存，当系统调用已注册回调函数时，该参数会通过回调函数的参数传回。
返回值:	成功: 返回 RS_SUCC。
	失败: 返回错误号。

2.2.3 设置是否允许实时路点信息推送

int rs_enable_push_realtime_roadpoint(RSHD rshd, bool enable);

功能描述:	设置是否允许实时路点信息推送。
参数说明:	1. rshd：机械臂控制上下文句柄。 2. enable：true 表示允许，false 表示不允许。
返回值:	成功: 返回 RS_SUCC。
	失败: 返回错误号。

2.2.4 注册用于获取实时路点的回调函数

int rs_setcallback_realtime_roadpoint(RSHD rshd, const RealTimeRoadPointCallback ptr, void *arg);	
功能描述:	注册用于获取实时路点信息的回调函数。 注册回调函数后，服务器实时推送当前的路点信息。 频率 30ms。
参数说明:	1. rshd：机械臂控制上下文句柄。 2. ptr：获取实时路点信息的回调函数指针。 3. arg：这个参数系统不做任何处理，只是进行了缓存，当系统调用已注册回调函数时，该参数会通过回调函数的参数传回。
返回值:	成功: 返回 RS_SUCC。
	失败: 返回错误号。

2.2.5 设置是否允许实时末端速度推送

int rs_enable_push_realtime_end_speed(RSHD rshd, bool enable);	
功能描述:	设置是否允许实时末端速度推送。
参数说明:	1. rshd：机械臂控制上下文句柄。 2. enable：true 表示允许，false 表示不允许。
返回值:	成功: 返回 RS_SUCC。
	失败: 返回错误号。

2.2.6 注册用于获取实时末端速度的回调函数

int rs_setcallback_realtime_end_speed(RSHD rshd, const RealTimeEndSpeedCallback ptr, void *arg);	
功能描述:	注册用于获取实时末端速度的回调函数。 注册回调函数后，服务器会实时推送当前的末端速度。 频率 30ms。
参数说明:	1. rshd：机械臂控制上下文句柄。 2. ptr：获取实时末端速度信息的回调函数指针。 3. arg：这个参数系统不做任何处理，只是进行了缓存，当系统

	调用已注册回调函数时，该参数会通过回调函数的参数传回。
返回值:	成功: 返回 RS_SUCC。
	失败: 返回错误号。

2.2.7 注册用于获取机械臂事件信息的回调函数

int rs_setcallback_robot_event(RSHD rshd, const RobotEventCallback ptr, void *arg);	
功能描述:	注册用于获取机械臂事件信息的回调函数。 注册回调函数后，服务器会实时推送机械臂事件信息。 频率 30ms。 注意：关于事件信息的推送没有提供是否允许推送的接口，因为机械臂的很多重要通知都是通过事件推送实现的，所以事件信息是系统默认推送的，不允许取消的。
参数说明:	1. rshd：机械臂控制上下文句柄。 2. ptr：获取机械臂事件信息的函数指针。当 ptr==NULL 时，相等于取消回调函数的注册。 3. arg：这个参数系统不做任何处理，只是进行了缓存，当系统调用已注册回调函数时，该参数会通过回调函数的参数传回。
返回值:	成功: 返回 RS_SUCC。
	失败: 返回错误号。

2.2.8 注册 movep 进度通知的回调函数

int rs_setcallback_movep_step_num(RSHD rshd, const RealTimeMovepStepNumNotifyCallback ptr, void *arg);	
功能描述:	注册 movep 进度通知的回调函数。
参数说明:	1. rshd：机械臂控制上下文句柄。 2. ptr：获取机械臂 movep 进度通知的函数指针，当 ptr==NULL 时，相等于取消回调函数的注册。 3. arg：这个参数系统不做任何处理，只是进行了缓存，当回调函数触发时，该参数会通过回调函数的参数传回。
返回值:	成功: 返回 RS_SUCC。
	失败: 返回错误号。

2.2.9 注册日志输出回调函数

int rs_setcallback_robot_loginfo(RSHD rshd, const RobotLogPrintCallback ptr, void *arg);	
功能描述:	注册日志输出回调函数。

参数说明:	1. rshd：机械臂控制上下文句柄。 2. ptr：获取机械臂日志输出的函数指针，当 ptr==NULL 时，相等于取消回调函数的注册。 3. arg：这个参数系统不做任何处理，只是进行了缓存，当回调函数触发时，该参数会通过回调函数的参数传回。
返回值:	成功: 返回 RS_SUCC。
	失败: 返回错误号。

2.3 机械臂运动相关的接口

本部分接口是关于运动相关的接口，包括运动属性的设置和多种形式的运动。

注意：在运动之前应该设置好对应的运动属性。

本机械臂接口支持下面几种运动:

1. 关节运动：对应接口函数为 rs_move_joint();
2. 直线运动：对应接口函数为 rs_move_line();
3. 轨迹运动：对应接口函数为 rs_move_track();
4. 旋转运动：对应接口函数为 rs_move_rotate();

根据上面运动的扩展运动:

1. 关节运动到目标位置：对应接口函数为 rs_move_joint_toByRelative()，rs_move_joint_to();
2. 直线运动到目标位置：对应接口函数为 rs_move_line_toByRelative()，rs_move_line_to();

注意：如果用到偏移量的话需要设置偏移量属性。如果用到了工具需要在运动之前设置工具的运动学参数（参见接口 robotServiceSetToolKinematicsParam）和动力学参数(参见接口 robotServiceSetToolDynamicsParam)。

2.3.1 初始化全局的运动属性

int rs_init_global_move_profile(RSHD rshd);	
功能描述:	初始化全局的运动属性。对运动属性进行初始化，将各个属性设置为初始值。调用此函数时机械臂不运动，该函数初始化各个运动属性为默认值。 初始化后各属性的默认值为: 0:关节型运动的最大速度和最大加速度属性：关节最大速度默认为 25 度每秒；关节最大加速度默认为 25 度每秒方。当运动类型为关节型运动时有效。 1:末端型运动的最大线速度和最大线加速度属性：末端最大速度默认为 0.03 米每秒;末端最大加速度默认为 0.03 米每秒方； 末端型运动的最大角速度和最大角加速度属性：末端最大速度默认为 100 度每秒;末端最大加速度默认为 100 度每秒方； 当运动类型为末端型运动时有效。 2:路点信息缓存属性：默认路点缓存为空，轨迹运动时使用。 3:交融半径属性：默认为 0.02 米。轨迹运动子类型为 MOVEP 时使用。 4:轨迹运动中圆轨迹的圈数属性：默认为 0，当轨迹运动类型等于 ARC_CIR 时该属性生效；且圆轨迹的圈数等于零时，ARC_CIR 代表圆弧，圆轨迹的圈数大于零时，ARC_CIR 代表圆。

	5:运动属性之偏移量属性: 默认没有偏移，除示教运动外的所有运动有效。 6:工具参数属性属性: 无工具即工具的位置为 0 0 0 。 7:示教坐标系属性: 示教坐标系为基座标系，仅适用于示教运动。
参数说明:	rshd：机械臂控制上下文句柄。
返回值:	成功: 返回 RS_SUCC。
	失败: 返回错误号。

2.3.2 设置关节型运动的最大加速度

int rs_set_global_joint_maxacc(RSHD rshd, const JointVelcAccParam *max_acc);	
功能描述:	设置关节型运动的最大加速度。 **注意：用户在设置速度和加速度时，需要根据运动的类型设置。**
参数说明:	1. rshd：机械臂控制上下文句柄。 2. max_acc：关节运动的最大加速度，单位rad/s^2。
返回值:	成功: 返回 RS_SUCC。
	失败: 返回错误号。

2.3.3 设置关节型运动的最大速度

int rs_set_global_joint_maxvelc(RSHD rshd, const JointVelcAccParam *max_velc);	
功能描述:	设置关节型运动的最大速度。 **注意：用户在设置速度和加速度时，需要根据运动的类型设置。**
参数说明:	1. rshd：机械臂控制上下文句柄。 2. max_velc：关节运动的最大速度，单位rad/s。
返回值:	成功: 返回 RS_SUCC。
	失败: 返回错误号。

2.3.4 获取关节型运动的最大加速度

int rs_get_global_joint_maxacc(RSHD rshd, JointVelcAccParam *max_acc);	
功能描述:	获取关节型运动的最大加速度。
参数说明:	1. rshd：机械臂控制上下文句柄。 2. max_acc：传出参数，表示关节运动最大加速度，单位rad/s^2
返回值：	成功: 返回 RS_SUCC。
	失败: 返回错误号。

2.3.5 获取关节型运动的最大速度

int rs_get_global_joint_maxvelc(RSHD rshd, JointVelcAccParam *max_velc);	
功能描述:	获取关节型运动的最大速度。
参数说明:	1. rshd：机械臂控制上下文句柄。 2. max_velc：传出参数，表示关节运动最大速度，单位rad/s。
返回值：	成功: 返回 RS_SUCC。
	失败: 返回错误号。

2.3.6 设置末端型运动的最大线加速度

int rs_set_global_end_max_line_acc(RSHD rshd, double max_acc);	
功能描述:	设置末端型运动的最大加速度。 **注意：用户在设置速度和加速度时，需要根据运动的类型设置。**
参数说明:	1. rshd：机械臂控制上下文句柄。 2. max_acc：末端型运动的最大加速度，单位m/s^2。
返回值:	成功: 返回 RS_SUCC。
	失败: 返回错误号。

2.3.7 设置末端型运动的最大线速度

int rs_set_global_end_max_line_acc(RSHD rshd, double max_acc);	
功能描述:	设置末端型运动的最大线速度。 **注意：用户在设置速度和加速度时，需要根据运动的类型设置。**
参数说明:	1. rshd：机械臂控制上下文句柄。 2. max_acc：末端型运动的最大线速度，单位m/s。
返回值:	成功: 返回 RS_SUCC。
	失败: 返回错误号。

2.3.8 获取末端型运动的最大线加速度

int rs_get_global_end_max_line_acc(RSHD rshd, double *max_acc);	
功能描述:	获取末端型运动最大线加速度。
参数说明:	1. rshd：机械臂控制上下文句柄。 2. max_acc：传出参数，末端型运动的最大线加速度，单位m/s^2
返回值:	成功: 返回 RS_SUCC。
	失败: 返回错误号。

2.3.9 获取末端型运动的最大线速度

int rs_get_global_end_max_line_velc(RSHD rshd, double *max_velc);	
功能描述:	获取末端型运动的最大线速度。
参数说明:	1. rshd：机械臂控制上下文句柄。 2. max_velc：传出参数，末端型运动最大速度，单位m/s。
返回值:	成功: 返回 RS_SUCC。
	失败: 返回错误号。

2.3.10 设置末端型运动的最大角加速度

int rs_set_global_end_max_angle_acc(RSHD rshd, double max_acc);	
功能描述:	设置末端型运动的最大角加速度。 **注意：用户在设置速度和加速度时，需要根据运动的类型设置。**
参数说明:	1. rshd：机械臂控制上下文句柄。 2. max_acc：末端型运动的最大角加速度，单位rad/s^2。
返回值:	成功: 返回 RS_SUCC。
	失败: 返回错误号。

2.3.11 设置末端型运动的最大角速度

int rs_set_global_end_max_angle_velc(RSHD rshd, double max_velc);	
功能描述:	设置末端型运动的最大角速度。 **注意：用户在设置速度和加速度时，需要根据运动的类型设置。**
参数说明:	1. rshd：机械臂控制上下文句柄。 2. max_velc：末端型运动的最大角速度，单位rad/s。
返回值:	成功: 返回 RS_SUCC。
	失败: 返回错误号。

2.3.12 获取末端型运动的最大角加速度

int rs_get_global_end_max_angle_acc(RSHD rshd, double *max_acc);	
功能描述:	获取末端型运动的最大角加速度。
参数说明:	1. rshd：机械臂控制上下文句柄。 2. max_acc：传出参数，末端型运动的最大角加速度，单位rad/s^2。
返回值:	成功: 返回 RS_SUCC。
	失败: 返回错误号。

2.3.13 获取末端型运动的最大角速度

int rs_get_global_end_max_angle_velc(RSHD rshd, double *max_velc);	
功能描述:	获取末端型运动的最大角速度。
参数说明:	1. rshd：机械臂控制上下文句柄。 2. max_velc：传出参数，末端型运动的最大角速度，单位rad/s
返回值:	成功: 返回 RS_SUCC。
	失败: 返回错误号。

2.3.14 清除路点容器

int rs_remove_all_waypoint(RSHD rshd);	
功能描述:	清除路点容器，通常用于新的轨迹运动之前。
参数说明:	rshd：机械臂控制上下文句柄。
返回值:	成功: 返回 RS_SUCC。
	失败: 返回错误号。

2.3.15 添加路点

int rs_add_waypoint(RSHD rshd, double joint_radian[ARM_DOF]);	
功能描述:	添加路点，用于轨迹运动中。
参数说明:	1. rshd：机械臂控制上下文句柄。 2. joint_radian：关节角。
返回值:	成功: 返回 RS_SUCC。
	失败: 返回错误号。

2.3.16 设置交融半径

int rs_set_blend_radius(RSHD rshd, double radius);	
功能描述:	设置交融半径。 交融半径的范围：0.0m~0.05m。 **注意：交融半径必须大于 0.0。**
参数说明:	1. rshd：机械臂控制上下文句柄。 2. radius：交融半径，单位 m。
返回值:	成功: 返回 ErrnoSucc。
	失败: 返回错误号。

2.3.17 设置圆轨迹时圆的圈数

int rs_set_circular_loop_times(RSHD rshd, int times);

功能描述:	设置圆运动圈数。 **注意：当轨迹类型为 ARC_CIR 时有效。**
参数说明:	1. rshd：机械臂控制上下文句柄。 2. times：圆的圈数。 times = 0 时，表示圆弧运动； times > 0 时，表示圆运动，且表示圆的圈数
返回值:	成功: 返回 ErrnoSucc。
	失败: 返回错误号。

2.3.18 设置基于基座标系运动偏移量

int rs_set_relative_offset_on_base(RSHD rshd, const MoveRelative *relative);	
功能描述:	设置基于基坐标系下的相对偏移属性。
参数说明:	1. rshd：机械臂控制上下文句柄。 2. relative：基于基坐标系的偏移。
返回值:	成功: 返回 ErrnoSucc。
	失败: 返回错误号。

2.3.19 设置基于用户标系运动偏移量

int rs_set_relative_offset_on_user(RSHD rshd, const MoveRelative *relative, const CoordCalibrate *user_coord);	
功能描述:	设置基于用户标系运动偏移量。
参数说明:	1. rshd：机械臂控制上下文句柄。 2. relative：基于用户坐标系的偏移(x, y, z)，单位：米。 3. user_coord：用户坐标系。
返回值:	成功: 返回 ErrnoSucc。
	失败: 返回错误号。

2.3.20 取消提前到位设置

int rs_set_no_arrival_ahead(RSHD rshd);	
功能描述:	取消提前到位设置。
参数说明:	rshd：机械臂控制上下文句柄。
返回值:	成功: 返回 ErrnoSucc。
	失败: 返回错误号。

2.3.21 设置提前到位距离模式

int rs_set_arrival_ahead_distance(RSHD rshd, double distance);

功能描述:	设置提前到位距离模式。 **注意：跟随模式之提前到位，当前仅适用于关节运动。**
参数说明:	1. rshd：机械臂控制上下文句柄。 2. distance：距离，单位 m。
返回值:	成功: 返回 ErrnoSucc。
	失败: 返回错误号。

2.3.22 设置提前到位时间模式

int rs_set_arrival_ahead_time(RSHD rshd, double sec);	
功能描述:	设置提前到位时间模式。 **注意：跟随模式之提前到位，当前仅适用于关节运动。**
参数说明:	1. rshd：机械臂控制上下文句柄。 2. second：提前到位时间，单位 s。
返回值:	成功: 返回 ErrnoSucc。
	失败: 返回错误号。

2.3.23 设置提前到位交融半径模式

int rs_set_arrival_ahead_blend(RSHD rshd, double radius);	
功能描述:	设置提前到位交融半径模式。 **注意：跟随模式之提前到位，当前仅适用于关节运动。**
参数说明:	1. rshd：机械臂控制上下文句柄。 2. radius：交融半径，单位 m。
返回值:	成功: 返回 ErrnoSucc。
	失败: 返回错误号。

2.3.24 关节运动

int rs_move_joint(RSHD rshd, double joint_radian[ARM_DOF], bool isblock = true);	
功能描述:	运动接口之关节运动。
参数说明:	1. rshd：机械臂控制上下文句柄。 2. joint_radian：关节角。 3. IsBolck：是否阻塞。 IsBolck==true 代表阻塞，机械臂运动直到到达目标位置或者出现故障后返回。 IsBolck==false 代表非阻塞，立即返回，运动指令发送成功就返回，函数返回后机械臂开始运动。
返回值:	成功: 返回 ErrnoSucc。
	失败: 返回错误号。

2.3.25 机械臂轴动

int rs_move_joint_ex(RSHD rshd, MoveProfile_t *move_profile, double joint_radian[ARM_DOF], bool isblock = true);	
功能描述:	运动接口之关节运动。
参数说明:	1. rshd：机械臂控制上下文句柄。 2. move_profile：运动属性 3. joint_radian：关节角。 4. IsBolck：是否阻塞。 IsBolck==true 代表阻塞，机械臂运动直到到达目标位置或者出现故障后返回。 IsBolck==false 代表非阻塞，立即返回，运动指令发送成功就返回，函数返回后机械臂开始运动。
返回值:	成功: 返回 ErrnoSucc。
	失败: 返回错误号。

2.3.26 保持当前位姿通过关节运动的方式运动到目标位置

int rs_move_joint_to(RSHD rshd, const Pos *target, const ToolInEndDesc *tool, bool isblock=true);	
功能描述:	保持当前姿态通过轴动运动的方式运动到目标位置。
参数说明:	1. rshd：机械臂控制上下文句柄。 2. target：基于用户平面表示的目标位置。 3. tool：工具参数，当没有使用工具时，将此参数设置为 0 。 4. IsBolck：是否阻塞。 IsBolck==true 代表阻塞，机械臂运动直到到达目标位置或者出现故障后返回。 IsBolck==false 代表非阻塞，立即返回，运动指令发送成功就返回，函数返回后机械臂开始运动。
返回值:	成功: 返回 ErrnoSucc。
	失败: 返回错误号。

2.3.27 基于跟随模式的轴动

int rs_follow_mode_move_joint(RSHD rshd, double joint_radian[ARM_DOF]);	
功能描述:	基于跟随模式的轴动接口。
参数说明:	1. rshd：机械臂控制上下文句柄。 2. joint_radian：关节角。

返回值:	成功: 返回 ErrnoSucc。
	失败: 返回错误号。

2.3.28 直线运动

int rs_move_line(RSHD rshd, double joint_radian[ARM_DOF], bool isblock = true);	
功能描述:	运动接口之直线运动，属于末端型运动。
参数说明:	1. rshd：机械臂控制上下文句柄。 2. joint_radian：关节角。 3. IsBlock：是否阻塞。 IsBolck==true 代表阻塞，机械臂运动直到到达目标位置或者出现故障后返回。 IsBolck==false 代表非阻塞，立即返回，运动指令发送成功就返回，函数返回后机械臂开始运动。
返回值:	成功: 返回 ErrnoSucc。
	失败: 返回错误号。

2.3.29 机械臂保持当前姿态直线运动

int rs_move_line_ex(RSHD rshd, MoveProfile_t *move_profile, double joint_radian[ARM_DOF], bool isblock = true);	
功能描述:	运动接口之直线运动。
参数说明:	1. rshd：机械臂控制上下文句柄。 2. move_profile：运动属性 3. joint_radian：关节角。 4. IsBolck：是否阻塞。 IsBolck==true 代表阻塞，机械臂运动直到到达目标位置或者出现故障后返回。 IsBolck==false 代表非阻塞，立即返回，运动指令发送成功就返回，函数返回后机械臂开始运动。
返回值:	成功: 返回 ErrnoSucc。
	失败: 返回错误号。

2.3.30 保持当前位姿通过直线运动的方式运动到目标位置

int rs_move_line_to(RSHD rshd, const Pos *target, const ToolInEndDesc *tool,

bool isblock=true);	
功能描述:	保持当前位姿通过直线运动的方式运动到目标位置。
参数说明:	1. rshd：机械臂控制上下文句柄。 2. target：基于用户平面表示的目标位置。 3. tool：工具参数，当没有使用工具时，将此参数设置为 0 。 4. IsBolck：是否阻塞。 IsBolck==true 代表阻塞，机械臂运动直到到达目标位置或者出现故障后返回。 IsBolck==false 代表非阻塞，立即返回，运动指令发送成功就返回，函数返回后机械臂开始运动。
返回值:	成功: 返回 ErrnoSucc。
	失败: 返回错误号。

2.3.31 保持当前位置变换姿态做旋转运动

int rs_move_rotate(RSHD rshd, const CoordCalibrate *user_coord, const Move_Rotate_Axis *rotate_axis, double rotate_angle, bool isblock = true);	
功能描述:	运动接口之保持当前位置变换姿态做旋转运动，即机械臂的末端点绕着指定转轴做旋转运动（位置保持不变）。
参数说明:	1. rshd：机械臂控制上下文句柄。 2. user_coord：用户坐标系。 3. rotate_axis：转轴[x,y,z]。比如，(1,0,0)表示沿 Y 轴转动。 4. rotate_angle：绕转轴转动的转角，单位rad。 5. IsBlock：是否阻塞。 IsBolck==true 代表阻塞，机械臂运动直到到达目标位置或者出现故障后返回。 IsBolck==false 代表非阻塞，立即返回，运动指令发送成功就返回，函数返回后机械臂开始运动。
返回值:	成功: 返回 ErrnoSucc。
	失败: 返回错误号。

2.3.32 根据起始点姿态及基坐标系下表示的旋转轴、旋转角，获取目标位姿

int rs_get_rotate_target_waypiont(RSHD rshd, const aubo_robot_namespace::wayPoint_S *originateWayPointOnBaseCoord, const double rotateAxisOnBaseCoord[], double rotateAngle,

aubo_robot_namespace::wayPoint_S *targetWayPointOnBaseCoord);	
功能描述:	根据当前位姿及基坐标系下表示的旋转轴、旋转角，获取目标位姿
参数说明:	1. rshd：机械臂控制上下文句柄。 2. originateWayPointOnBaseCoord：起始路点信息。 3. rotateAxisOnBaseCoord：基坐标系下表示的旋转轴。 4. rotateAngle：旋转角。 5. targetWayPointOnBaseCoord：传出参数，目标路点信息。
返回值:	成功: 返回 ErrnoSucc。
	失败: 返回错误号。

2.3.33 将用户坐标系描述的旋转轴变换到基坐标系下描述

int rs_get_rotateaxis_user_to_Base(RSHD rshd, const aubo_robot_namespace::Ori *oriOnUserCoord, const double rotateAxisOnUserCoord[], double rotateAxisOnBaseCoord[]);	
功能描述:	将用户坐标系描述的旋转轴变换到基坐标系下描述的旋转轴。
参数说明:	1. rshd：机械臂控制上下文句柄。 2. oriOnUserCoord：用户坐标系姿态。 3. rotateAxisOnUserCoord：用户坐标系下描述的旋转轴。 4. rotateAxisOnBaseCoord：传出参数，基坐标系下描述的旋转轴
返回值:	成功: 返回 ErrnoSucc。
	失败: 返回错误号。

2.3.34 保持当前位置变换姿态旋转运动至目标路点

int rs_move_rotate_to_waypoint(RSHD rshd, aubo_robot_namespace::wayPoint_S *target_waypiont, bool isblock = true);	
功能描述:	旋转运动到目标路点。
参数说明:	1. rshd：机械臂控制上下文句柄。 2. target_waypoint：目标路点。 3. IsBlock：是否阻塞。 IsBolck==true 代表阻塞，机械臂运动直到到达目标位置或者出现故障后返回。 IsBolck==false 代表非阻塞，立即返回，运动指令发送成功就返回，函数返回后机械臂开始运动。
返回值:	成功: 返回 ErrnoSucc。
	失败: 返回错误号。

2.3.35 轨迹运动

int rs_move_track(RSHD rshd, move_track sub_move_mode, bool isblock = true);	
功能描述:	轨迹运动。
参数说明:	1. rshd：机械臂控制上下文句柄。 2. sub_move_mode：轨迹运动类型：2：圆弧；3：轨迹。 3. IsBolck：是否阻塞。 IsBolck==true 代表阻塞，机械臂运动直到到达目标位置或者出现故障后返回。 IsBolck==false 代表非阻塞，立即返回，运动指令发送成功就返回，函数返回后机械臂开始运动。
返回值:	成功: 返回 ErrnoSucc。
	失败: 返回错误号。

2.3.36 开始示教

int rs_teach_move_start(RSHD rshd, teach_mode mode, bool dir);	
功能描述:	开始示教。
参数说明:	1. rshd：机械臂控制上下文句柄。 2. mode：示教模式。 示教关节：JOINT1, JOINT2, JOINT3, JOINT4, JOINT5, JOINT6。 位置示教：MOV_X, MOV_Y, MOV_Z。 姿态示教：ROT_X, ROT_Y, ROT_Z。 3. direction：运动方向，正方向 true，反方向 false。
返回值:	成功: 返回 ErrnoSucc。
	失败: 返回错误号。

2.3.37 设置示教运动的坐标系

int rs_set_teach_coord(RSHD rshd, const CoordCalibrate *teach_coord);	
功能描述:	设置示教运动的坐标系。
参数说明:	1. rshd：机械臂控制上下文句柄。 2. teach_coord：示教坐标系。
返回值:	成功: 返回 ErrnoSucc。
	失败: 返回错误号。

2.3.38 结束示教

int rs_teach_move_stop(RSHD rshd);	
功能描述:	结束示教。

参数说明:	rshd：机械臂控制上下文句柄。
返回值:	成功: 返回 ErrnoSucc。
	失败: 返回错误号。

2.3.39 清除服务器上的非在线轨迹运动数据

int rs_clear_offline_track(RSHD rshd);	
功能描述:	清除服务器离线轨迹的路点。
参数说明:	rshd：机械臂控制上下文句柄。
返回值:	成功: 返回 ErrnoSucc。
	失败: 返回错误号。

2.3.40 添加离线轨迹路点

int rs_append_offline_track_waypoint(RSHD rshd, const JointParam waypoints[], int waypoint_count);	
功能描述:	通过路点容器添加离线轨迹路点到服务器。
参数说明:	1. rshd：机械臂控制上下文句柄。 2. waypoints：路点数组 (路点个数小于等于 3000)。 3. waypoint_count：路点数组大小。
返回值:	成功: 返回 ErrnoSucc。
	失败: 返回错误号。

2.3.41 向服务器添加非在线轨迹运动路点文件

int rs_append_offline_track_file(RSHD rshd, const char* filename);	
功能描述:	通过路点文件的形式添加离线轨迹路点到服务器。
参数说明:	1. rshd：机械臂控制上下文句柄。 2. filename：路点文件全路径,路点文件的每一行包含六个关节的关节角(弧度),用逗号隔开。
返回值:	成功: 返回 ErrnoSucc。
	失败: 返回错误号。

2.3.42 通知服务器启动非在线轨迹运动

int rs_startup_offline_track(RSHD rshd);	
功能描述:	启动离线轨迹的运行。
参数说明:	rshd：机械臂控制上下文句柄。
返回值:	成功: 返回 ErrnoSucc。
	失败: 返回错误号。

2.3.43 通知服务器停止非在线轨迹运动

int rs_stop_offline_track(RSHD rshd);	
功能描述:	结束离线轨迹运动。
参数说明:	rshd：机械臂控制上下文句柄。
返回值:	成功: 返回 ErrnoSucc。
	失败: 返回错误号。

2.3.44 通知服务器进入 tcp 转 can 透传模式

int rs_enter_tcp2canbus_mode(RSHD rshd);	
功能描述:	进入 tcp 转 can 透传模式。
参数说明:	rshd：机械臂控制上下文句柄。
返回值:	成功: 返回 ErrnoSucc。
	失败: 返回错误号。

2.3.45 通知服务器退出 tcp 转 can 透传模式

int rs_leave_tcp2canbus_mode(RSHD rshd);	
功能描述:	退出 tcp 转 can 透传模式。
参数说明:	rshd：机械臂控制上下文句柄。
返回值:	成功: 返回 ErrnoSucc。
	失败: 返回错误号。

2.3.46 透传运动路点到 CANBUS

int rs_set_waypoint_to_canbus(RSHD rshd, double joint_radian[ARM_DOF]);	
功能描述:	退出 tcp 转 can 透传模式。
参数说明:	1. rshd：机械臂控制上下文句柄。 2. joint_radian：关节角，单位：rad。
返回值:	成功: 返回 ErrnoSucc。
	失败: 返回错误号。

2.4 工具接口

2.4.1 正解

int rs_forward_kin(RSHD rshd,

const double joint_radian[ARM_DOF], wayPoint_S *waypoint);	
功能描述:	正解，此函数为正解函数，已知关节角求对应位置的位置和姿态。
参数说明:	1. rshd：机械臂控制上下文句柄。 2. joint_radian：六个关节的关节角，单位rad。 3. waypoint：传出参数，正解得到得路点。
返回值:	成功: 返回 ErrnoSucc。
	失败: 返回错误号。

2.4.2 逆解

int rs_inverse_kin(RSHD rshd, double joint_radian[ARM_DOF], const Pos *pos, const Ori *ori, wayPoint_S *waypoint);	
功能描述:	逆解。根据位置信息(x,y,z)和对应位置的参考姿态(w,x,y,z)得到对应位置的关节角信息。 逆运动学问题：对某个机器人，当给出机器人手部（法兰盘中心）在基座标系中所处的位置和姿态时，求出其对应的关节角信息。
参数说明:	1. rshd：机械臂控制上下文句柄。 2. joint_radian:参考点六个关节的关节角，单位rad。 3. pos：目标路点的位置。 4. ori：目标路点的参考姿态。 例如：可以获取当前位置位姿作为此参数，这样相当于保持当前姿态。 5. waypoint：传出参数，目标路点信息。
返回值:	成功: 返回 ErrnoSucc。
	失败: 返回错误号。

int rs_inverse_kin_closed_form(RSHD rshd, const Pos *pos, const Ori *ori, ik_solutions *solutions);	
功能描述:	逆解，此函数为机械臂逆解函数，根据位置信息(x,y,z)和对应位置的参考姿态(w,x,y,z)得到对应位置的关节角信息。
参数说明:	1. rshd：机械臂控制上下文句柄。 2. pos：目标路点的位置。 3. ori：目标路点的参考姿态。 4. solutions：传出参数，目标路点信息（最多八组解）。
返回值:	成功: 返回 ErrnoSucc。
	失败: 返回错误号。

2.4.3 设置基坐标系

int rs_set_base_coord(RSHD rshd);	
功能描述:	设置基坐标系。
参数说明:	rshd：机械臂控制上下文句柄。
返回值:	成功: 返回 ErrnoSucc。
	失败: 返回错误号。

2.4.4 设置用户坐标系

int rs_set_user_coord(RSHD rshd, const CoordCalibrate *user_coord);	
功能描述:	设置用户坐标系。
参数说明:	1. rshd：机械臂控制上下文句柄。 2. user_coord：用户坐标系。
返回值:	成功: 返回 ErrnoSucc。
	失败: 返回错误号。

2.4.5 基坐标系转用户坐标系

int rs_base_to_user(RSHD rshd, const Pos *pos_onbase, const Ori *ori_onbase, const CoordCalibrate *user_coord, const ToolInEndDesc *tool_pos, Pos *pos_onuser, Ori *ori_onuser);	
功能描述:	将法兰盘中心基于基坐标系下的位置和姿态转成工具末端基于用户坐标系下的位置和姿态。 扩展 1：法兰盘中心可以看成是一个特殊的工具，即工具的位置为(0,0,0)姿态为（1,0,0,0）。 因此当工具为(0,0,0)时，相当于将法兰盘中心基于基坐标系下的位置和姿态转成法兰盘中心基于用户坐标系下的位置和姿态。 扩展 2：用户坐标系也可以选择成基坐标系，即：user_coord.coordType = BaseCoordinate。 因此当用户平面为基坐标系时，相当于将法兰盘中心基于基坐标系下的位置和姿态转成工具末端基于基坐标系下的位置和姿态，即在基坐标系加工具。
参数说明:	1. rshd：机械臂控制上下文句柄。 2. pos_onbase：法兰盘中心基于基坐标系下的位置信息（x,y,z）。 3. ori_onbase：法兰盘中心基于基坐标系下的姿态信息(w, x, y,

	z)。 4. user_coord：用户坐标系。 5. tool_pos：工具参数。 6. pos_onuser：传出参数，工具末端基于用户坐标系下的位置信息。 7. ori_onuser：传出参数，工具末端基于用户坐标系下的姿态信息。
返回值:	成功: 返回 ErrnoSucc。
	失败: 返回错误号。

2.4.6 基坐标系转基坐标得到工具末端点的位置和姿态

int rs_base_to_base_additional_tool(RSHD rshd, const Pos *flange_center_pos_onbase, const Ori *flange_center_ori_onbase, const ToolInEndDesc *tool, Pos *tool_end_pos_onbase, Ori *tool_end_ori_onbase);	
功能描述:	法兰盘中心在基坐标系下的位置和姿态转工具末端在基坐标系下的位置和姿态。
参数说明:	1. rshd：机械臂控制上下文句柄。 2. flange_center_pos_onbase：法兰盘中心基于基坐标系下的位置信息。 3. flange_center_ori_onbase：法兰盘中心基于基坐标系下的姿态信息。 4. tool：工具参数。 5. tool_end_pos_onbase：传出参数，工具末端基于基坐标系下的位置信息。 6. tool_end_ori_onbase：传出参数，工具末端基于基坐标系下的姿态信息。
返回值:	成功: 返回 ErrnoSucc。
	失败: 返回错误号。

2.4.7 用户坐标系转基坐标系

int rs_user_to_base(RSHD rshd, const Pos *pos_onuser, const Ori *ori_onuser, const CoordCalibrate *user_coord, const ToolInEndDesc *tool_pos, Pos *pos_onbase, Ori *ori_onbase);

功能描述:	将工具末端基于用户坐标系下的位置和姿态转成法兰盘中心基于基坐标系下的位置和姿态。 扩展 1：法兰盘中心可以看成是一个特殊的工具，即工具的位置为(0,0,0)姿态为（1,0,0,0） 因此当工具的位置为(0,0,0)姿态为（1,0,0,0）时，表示 pos_onuser 和 ori_onuser 是无工具的。 扩展 2：用户坐标系也可以选择成基坐标系，即：user_coord.coordType = BaseCoordinate。 因此当用户平面为基坐标系时，相当于将工具末端基于基坐标系下的位置和姿态转成法兰盘中心基于基坐标系下的位置和姿态。 扩展 3：利用该函数和逆解组合实现。当用户提供自定义坐标系（特殊为基坐标系）下工具末端的位置和姿态得到基坐标系下法兰盘中心的位置和姿态，然后在逆解，得到目标路点。
参数说明:	1. rshd：机械臂控制上下文句柄。 2. pos_onuser：工具末端基于用户坐标系下的位置信息。 3. ori_onuser：工具末端基于用户坐标系下的姿态信息。 4. user_coord：参考坐标系，通过该参数确定一个坐标系。 5. tool_pos：工具参数。 6. pos_onbase：传出参数，法兰盘中心基于基坐标系下的位置信息。 7. ori_onbase：传出参数，法兰盘中心基于基坐标系下的姿态信息。 **注意：这个的坐标系转换不支持末端系，即不支持 userCoord==EndCoordinate,如果 userCoord==EndCoordinate 会报参数错误(ErrCode_ParamError)。**
返回值:	成功: 返回 ErrnoSucc。
	失败: 返回错误号。

2.4.8 根据位置获取目标路点信息

int rs_get_target_waypoint_by_position(RSHD rshd, const aubo_robot_namespace::wayPoint_S &sourceWayPointOnBaseCoord, const aubo_robot_namespace::CoordCalibrateByJointAngleAndTool &userCoordSystem, const aubo_robot_namespace::Pos &toolEndPosition, const aubo_robot_namespace::ToolInEndDesc &toolInEndDesc, aubo_robot_namespace::wayPoint_S &targetWayPointOnBaseCoord);	
功能描述:	根据位置参数求得目标路点信息（姿态同源路点）
参数说明:	1. rshd：机械臂控制上下文句柄。 2. sourceWayPointOnBaseCoord：起始路点（基于基坐标系）。 3. userCoordSystem：用户坐标系。 4. toolEndPosition：末端工具位置。

	5. toolInEndDesc：末端工具参数。 6. targetWayPointOnBaseCoord：传出参数，目标路点（基于基坐标系）。
返回值:	成功: 返回 ErrnoSucc。
	失败: 返回错误号。

2.4.9 四元数转欧拉角

int rs_quaternion_to_rpy(RSHD rshd, const Ori *ori , Rpy *rpy);	
功能描述:	四元数转欧拉角。
参数说明:	1. rshd：机械臂控制上下文句柄。 2. ori：姿态的四元数表示方法。 3. rpy：传出参数，姿态的欧拉角表示方法。
返回值:	成功: 返回 ErrnoSucc。
	失败: 返回错误号。

2.4.10 欧拉角转四元数

int rs_rpy_to_quaternion(RSHD rshd, const Rpy *rpy, Ori *ori);	
功能描述:	欧拉角转四元数。
参数说明:	1. rshd：机械臂控制上下文句柄。 2. rpy：姿态的欧拉角表示方法。 3. ori：传出参数，姿态的四元素表示方法。
返回值:	成功: 返回 ErrnoSucc。
	失败: 返回错误号。

2.5 机械臂控制接口

2.5.1 机械臂控制

int rs_robot_control(RSHD rshd, RobotControlCommand cmd);	
功能描述:	机械臂控制。
参数说明:	1. rshd：机械臂控制上下文句柄。 2. cmd：控制命令。
返回值:	成功: 返回 ErrnoSucc。
	失败: 返回错误号。

2.5.2 机械臂快速停止

int rs_move_fast_stop(RSHD rshd);	
功能描述:	机械臂快速停止。 **注意：rs_move_fast_stop 需要在与 move 不同的线程中调用。且 rs_move_fast_stop 调用后需要停止 move 线程，否则会接着执行下一个 move 指令。**
参数说明:	rshd：机械臂控制上下文句柄。
返回值:	成功: 返回 ErrnoSucc。
	失败: 返回错误号。

2.5.3 机械臂运动停止

int rs_move_stop(RSHD rshd);	
功能描述:	机械臂运动停止。 **注意：rs_move_stop 需要在与 move 不同的线程中调用。**
参数说明:	rshd：机械臂控制上下文句柄。
返回值:	成功: 返回 ErrnoSucc。
	失败: 返回错误号。

2.5.4 暂停机械臂运动

int rs_move_continue(RSHD rshd);	
功能描述:	暂停机械臂运动。
参数说明:	rshd：机械臂控制上下文句柄。
返回值:	成功: 返回 ErrnoSucc。
	失败: 返回错误号。

2.5.5 暂停后恢复机械臂运动

int rs_move_continue(RSHD rshd);	
功能描述:	暂停后恢复机械臂运动。
参数说明:	rshd：机械臂控制上下文句柄。
返回值:	成功: 返回 ErrnoSucc。
	失败: 返回错误号。

2.6 末端工具接口

2.6.1 设置无工具的动力学参数

int rs_set_none_tool_dynamics_param(RSHD rshd);	
功能描述:	设置无工具的动力学参数。
参数说明:	rshd：机械臂控制上下文句柄。
返回值:	成功: 返回 ErrnoSucc。
	失败: 返回错误号。

2.6.2 设置工具的动力学参数

int rs_set_tool_dynamics_param(RSHD rshd, const ToolDynamicsParam *tool);	
功能描述:	设置工具的动力学参数。
参数说明:	1. rshd：机械臂控制上下文句柄。 2. tool：工具动力学参数。
返回值:	成功: 返回 ErrnoSucc。
	失败: 返回错误号。

2.6.3 获取工具的动力学参数

__attribute__ ((deprecated)) int rs_get_tool_dynamics_param(RSHD rshd, ToolDynamicsParam *tool);	
功能描述:	获取工具的动力学参数。
参数说明:	1. rshd：机械臂控制上下文句柄。 2. tool：传出参数，工具动力学参数。
返回值:	成功: 返回 ErrnoSucc。
	失败: 返回错误号。

2.6.4 获取工具的动力学参数

```
inline int rs_get_tool_dynamics_param2(RSHD rshd, ToolDynamicsParam *tool)
{
    int retval = rs_get_tool_dynamics_param(rshd, tool);
    tool->positionX /= 1000.;
    tool->positionY /= 1000.;
    tool->positionZ /= 1000.;

    return retval;
}
```

功能描述:	获取工具的动力学参数。
参数说明:	1. rshd：机械臂控制上下文句柄。 2. tool：传出参数，工具动力学参数。 **注意：由于 rs_get_tool_dynamics_param 获取的位置值单位是 mm，增加此函数将单位与 rs_set_tool_dynamics_param 统一。**
返回值:	成功: 返回 ErrnoSucc。
	失败: 返回错误号。

2.6.5 设置无工具的运动学参数

int rs_set_none_tool_kinematics_param(RSHD rshd);	
功能描述:	设置无工具运动学参数。
参数说明:	rshd：机械臂控制上下文句柄。
返回值:	成功: 返回 ErrnoSucc。
	失败: 返回错误号。

2.6.6 设置工具的运动学参数

int rs_set_tool_kinematics_param(RSHD rshd, const ToolKinematicsParam *tool);	
功能描述:	设置工具的运动学参数。
参数说明:	1. rshd：机械臂控制上下文句柄。 2. tool：工具运动学参数。
返回值:	成功: 返回 ErrnoSucc。
	失败: 返回错误号。

2.6.7 设置工具的运动学参数

int rs_set_tool_end_param(RSHD rshd,const ToolInEndDesc *tool);	
功能描述:	设置工具的运动学参数。
参数说明:	1. rshd：机械臂控制上下文句柄。 2. tool：工具运动学参数。
返回值:	成功: 返回 ErrnoSucc。
	失败: 返回错误号。

2.6.8 获取工具的运动学参数

int rs_get_tool_kinematics_param(RSHD rshd, ToolKinematicsParam *tool);	
功能描述:	获取工具的运动学参数
参数说明:	1. rshd：机械臂控制上下文句柄。 2. tool：传出参数，工具运动学参数。

返回值:	成功: 返回 ErrnoSucc。
	失败: 返回错误号。

2.7 设置和获取机械臂相关参数接口

2.7.1 获取当前的连接状态

int rs_get_login_status(RSHD rshd, bool *status);	
功能描述:	获取当前的连接状态。 该函数用于查看与机械臂服务器的连接状态。
参数说明:	1. rshd：机械臂控制上下文句柄。 2. status：传出参数，连接状态。true：在线；false：离线。
返回值:	成功: 返回 RS_SUCC。
	失败: 返回错误号。

2.7.2 设置当前机械臂模式：仿真或真实

int rs_set_work_mode(RSHD rshd, RobotWorkMode mode);	
功能描述:	设置当前机械臂模式：仿真或真实。
参数说明:	1. rshd：机械臂控制上下文句柄。 2. mode：仿真或真实的枚举类型。 机械臂仿真模式:RobotRunningMode.RobotModeSimulator， 机械臂真实模式:RobotRunningMode.RobotModeReal。
返回值:	成功: 返回 ErrnoSucc。
	失败: 返回错误号。

2.7.3 获取当前机械臂模式

int rs_get_work_mode(RSHD rshd, RobotWorkMode *mode);	
功能描述:	获取机械臂当前工作模式。
参数说明:	1. rshd：机械臂控制上下文句柄。 2. mode：传出参数，仿真或真实的枚举类型，表示机械臂当前模式。
返回值:	成功: 返回 ErrnoSucc。
	失败: 返回错误号。

2.7.4 获取重力分量

int rs_get_gravity_component(RSHD rshd, RobotGravityComponent *gravity);

功能描述:	获取重力分量。
参数说明:	1. rshd：机械臂控制上下文句柄。 2. gravity：传出参数，重力分量，需连接真实机器人。
返回值:	成功: 返回 ErrnoSucc。
	失败: 返回错误号。

2.7.5 获取当前碰撞等级

int rs_get_collision_class(RSHD rshd, int grade);	
功能描述:	获取碰撞等级。
参数说明:	1. rshd：机械臂控制上下文句柄。 2. grade：传出参数，碰撞等级，需连接真实机器人。
返回值:	成功: 返回 ErrnoSucc。
	失败: 返回错误号。

2.7.6 设置碰撞等级

int rs_set_collision_class(RSHD rshd, int grade);	
功能描述:	设置碰撞等级。
参数说明:	1. rshd：机械臂控制上下文句柄。 2. grade：碰撞等级：0~10。
返回值:	成功: 返回 ErrnoSucc。
	失败: 返回错误号。

```
int rs_set_collision_class2(RSHD rshd, int grade, CollisionMode mode)
{
    return rs_set_collision_class(rshd, ((int)mode << 5) | grade);
}
```

功能描述:	设置碰撞等级。
参数说明:	1. rshd：机械臂控制上下文句柄。 2. grade：碰撞等级：0~10。 3. mode：碰撞模式。
返回值:	成功: 返回 ErrnoSucc。
	失败: 返回错误号。

2.7.7 获取设备信息

int rs_get_device_info(RSHD rshd, RobotDevInfo *dev);	
功能描述:	获取机器人设备信息，需连接真实机器人。
参数说明:	1. rshd：机械臂控制上下文句柄。 2. dev：设备信息。

返回值:	成功: 返回 ErrnoSucc。
	失败: 返回错误号。

2.7.8 获取是否存在真实机械臂

int rs_is_have_real_robot(RSHD rshd , bool *exist);	
功能描述:	获取是否存在真实机械臂
参数说明:	1. rshd：机械臂控制上下文句柄。 2. exist：传出参数，true：存在真实机械臂；false：不存在真实机械臂。
返回值:	成功: 返回 ErrnoSucc。
	失败: 返回错误号。

2.7.9 获取机械臂关节状态

int rs_get_joint_status(RSHD rshd, JointStatus jointStatus[ARM_DOF]);	
功能描述:	获取机械臂关节状态
参数说明:	1. rshd：机械臂控制上下文句柄。 2. jointStatus：传出参数，关节状态。
返回值:	成功: 返回 ErrnoSucc。
	失败: 返回错误号。

2.7.10 获取机械臂诊断信息

int rs_get_diagnosis_info(RSHD rshd, RobotDiagnosis *robotDiagnosisInfo);	
功能描述:	获取机械臂诊断信息。
参数说明:	1. rshd：机械臂控制上下文句柄。 2. robotDiagnosisInfo：传出参数，机械臂诊断信息。
返回值:	成功: 返回 ErrnoSucc。
	失败: 返回错误号。

2.7.11 根据错误号获取错误信息

const char* rs_get_error_information_by_errcode(RSHD rshd, RobotErrorCode err_code);	
功能描述:	根据错误号获取错误信息。
参数说明:	1. rshd：机械臂控制上下文句柄。 2. err_code：错误号。
返回值:	成功: 返回错误信息。
	失败: 返回错误号。

2.7.12 获取机械臂当前路点信息

int rs_get_current_waypoint(RSHD rshd, wayPoint_S *waypoint);	
功能描述:	获取机械臂当前路点信息。
参数说明:	1. rshd：机械臂控制上下文句柄。 2. waypoint：传出参数，当前路点信息。
返回值:	成功: 返回 ErrnoSucc。
	失败: 返回错误号。

2.7.13 当前机械臂是否运行在联机模式

int rs_is_online_mode(RSHD rshd, bool *isonline);	
功能描述:	返回当前机械臂是否运行在联机模式。
参数说明:	1. rshd：机械臂控制上下文句柄。 2. isonline：传出参数，true：联机 false：非联机。
返回值:	成功: 返回 ErrnoSucc。
	失败: 返回错误号。

2.7.14 当前机械臂是否运行在联机主模式

int rs_is_online_master_mode(RSHD rshd, bool *ismaster);	
功能描述:	返回当前机械臂是否运行在联机主模式。
参数说明:	1. rshd：机械臂控制上下文句柄。 2. ismaster：传出参数，true：联机主模式/手动模式 false：联机从模式。
返回值:	成功: 返回 ErrnoSucc。
	失败: 返回错误号。

2.7.15 获取机械臂当前运行状态

int rs_get_robot_state(RSHD rshd, RobotState *state);	
功能描述:	获取机械臂当前运行状态。 **注意：需要与 move 在不同线程里。**
参数说明:	1. rshd：机械臂控制上下文句柄。 2. state：传出参数，运行状态。 机械臂当前停止:RobotStatus.Stopped 机械臂当前运行:RobotStatus.Running 机械臂当前暂停:RobotStatus.Paused 机械臂当前恢复:RobotStatus.Resumed

返回值:	成功: 返回 ErrnoSucc。
	失败: 返回错误号。

2.7.16 获取 socket 连接状态

int rs_get_socket_status(RSHD rshd, bool *connected);	
功能描述:	获取 socket 连接状态。
参数说明:	1. rshd：机械臂控制上下文句柄。 2. connected：传出参数，socket 连接状态。true:已连接 false：未连接。
返回值:	成功: 返回 ErrnoSucc。
	失败: 返回错误号。

2.7.17 获取机械臂末端速度

int rs_get_robot_end_speed(RSHD rshd, float *endspeed);	
功能描述:	获取机械臂末端速度。
参数说明:	1. rshd：机械臂控制上下文句柄。 2. endspeed：传出参数，末端速度。
返回值:	成功: 返回 ErrnoSucc。
	失败: 返回错误号。

2.8 接口板 IO 相关的接口

接口板 IO 主要分为两大块，分别是控制器 IO 和用户 IO，每个 IO 都有对应的名称和地址，在使用是可以通过名称或地址两种形式获取和设置 IO 的状态。

2.8.1 获取接口板指定 IO 集合的配置信息

int rs_get_board_io_config(RSHD rshd, RobotIoType type, std::vector<RobotIoDesc> *config);	
功能描述:	获取接口板指定 IO 集合的配置信息。
参数说明:	1. rshd：机械臂控制上下文句柄。 2. type：IO 类型。 3. config：传出参数，IO 配置信息的集合。
返回值:	成功: 返回 ErrnoSucc。
	失败: 返回错误号。

2.8.2 根据接口板 IO 类型和名称设置 IO 状态

int rs_set_board_io_status_by_name(RSHD rshd, RobotIoType type, const char *name, double val);	
功能描述:	根据接口板 IO 类型和名称设置 IO 状态。
参数说明:	1. rshd：机械臂控制上下文句柄。 2. type：IO 类型。 3. name：IO 名称。 4. val：IO 状态。
返回值:	成功: 返回 ErrnoSucc。
	失败: 返回错误号。

2.8.3 根据接口板 IO 类型和地址设置 IO 状态

int rs_set_board_io_status_by_addr(RSHD rshd, RobotIoType type, int addr, double val);	
功能描述:	根据接口板 IO 类型和地址设置 IO 状态。
参数说明:	1. rshd：机械臂控制上下文句柄。 2. type：IO 类型。 3. addr：IO 地址。 4. val：IO 状态。
返回值:	成功: 返回 ErrnoSucc。
	失败: 返回错误号。

2.8.4 根据接口板 IO 类型和名称获取 IO 状态

int rs_get_board_io_status_by_name(RSHD rshd, RobotIoType type, const char *name, double *val);	
功能描述:	根据接口板 IO 类型和名称获取 IO 状态。
参数说明:	1. rshd：机械臂控制上下文句柄。 2. type：IO 类型。 3. name：IO 名称。 4. val：传出参数，IO 状态。
返回值:	成功: 返回 ErrnoSucc。
	失败: 返回错误号。

2.8.5 根据接口板 IO 类型和地址获取 IO 状态

int rs_get_board_io_status_by_addr(RSHD rshd, RobotIoType type, int addr, double *val);	
功能描述:	根据接口板 IO 类型和地址获取 IO 状态。
参数说明:	1. rshd：机械臂控制上下文句柄。

	2. type：IO 类型。 3. addr：IO 地址。 4. val：传出参数，IO 状态。
返回值:	成功: 返回 ErrnoSucc。
	失败: 返回错误号。

2.9 工具 IO 相关的接口

2.9.1 设置工具端电源电压类型

int rs_set_tool_power_type(RSHD rshd, ToolPowerType type);	
功能描述:	设置工具端电源电压类型。
参数说明:	1. rshd：机械臂控制上下文句柄。 2. type：工具电源电压类型。 0：OUT_0V 1：OUT_12V 2：OUT_24V
返回值:	成功: 返回 ErrnoSucc。
	失败: 返回错误号。

2.9.2 获取工具端电源电压类型

int rs_get_tool_power_type(RSHD rshd, ToolPowerType *type);	
功能描述:	获取工具端电源电压类型。
参数说明:	1. rshd：机械臂控制上下文句柄。 2. type：传出参数，工具电源电压类型。 0：OUT_0V 1：OUT_12V 2：OUT_24V
返回值:	成功: 返回 ErrnoSucc。
	失败: 返回错误号。

2.9.3 获取工具端的电源电压数值

int rs_get_tool_power_voltage(RSHD rshd, double *voltage);	
功能描述:	获取工具端的电源电压数值。
参数说明:	1. rshd：机械臂控制上下文句柄。 2. voltage：传出参数，工具端的电源电压值，单位：V。
返回值:	成功: 返回 ErrnoSucc。
	失败: 返回错误号。

2.9.4 设置工具端数字量 IO 的类型：输入或者输出

int rs_set_tool_io_type(RSHD rshd, ToolDigitalIOAddr addr, ToolIOType type);	
功能描述:	设置工具端数字量 IO 的类型：输入或者输出。
参数说明:	1. rshd：机械臂控制上下文句柄。 2. addr：IO 地址。 3. type：IO 类型：输入或者输出。
返回值:	成功: 返回 ErrnoSucc。
	失败: 返回错误号。

2.9.5 根据名称获取工具端 IO 的状态

int rs_get_tool_io_status(RSHD rshd, const char *name, double *val);	
功能描述:	根据名称获取工具端 IO 的状态。
参数说明:	1. rshd：机械臂控制上下文句柄。 2. name：IO 名称。 3. value：传出参数，IO 状态。
返回值:	成功: 返回 ErrnoSucc。
	失败: 返回错误号。

2.9.6 根据名称设置工具端 IO 的状态

int rs_set_tool_do_status(RSHD rshd, const char *name, IO_STATUS status);	
功能描述:	根据名称获取工具端 IO 的状态。
参数说明:	1. rshd：机械臂控制上下文句柄。 2. name：IO 名称。 3. status：传出参数，IO 状态。
返回值:	成功: 返回 ErrnoSucc。
	失败: 返回错误号。

3 错误码

3.1 接口函数错误码定义

错误号	错误代码	错误信息
0	InterfaceCallSuccCode	成功
10000	ErrCode_Base	
10001	ErrCode_Failed	通用失败
10002	ErrCode_ParamError	参数错误
10003	ErrCode_ConnectSocketFailed	Socket 连接失败
10004	ErrCode_SocketDisconnect	Socket 断开连接
10005	ErrCode_CreateRequestFailed	创建请求失败
10006	ErrCode_RequestRelatedVariableError	请求相关的内部变量出错
10007	ErrCode_RequestTimeout	请求超时
10008	ErrCode_SendRequestFailed	发送请求信息失败
10009	ErrCode_ResponseInfoIsNULL	响应信息为空
10010	ErrCode_ResolveResponseFailed	解析响应失败
10011	ErrCode_FkFailed	正解出错
10012	ErrCode_IkFailed	逆解出错
10013	ErrCode_ToolCalibrateError	工具标定参数有错
10014	ErrCode_ToolCalibrateParamError	工具标定参数有错
10015	ErrCode_CoordinateSystemCalibrateError	坐标系标定失败
10016	ErrCode_BaseToUserConvertFailed	基坐标系转用户坐标系失败
10017	ErrCode_UserToBaseConvertFailed	用户坐标系转基坐标系失败
10018	ErrCode_MotionRelatedVariableError	运动相关的内部变量出错
10019	ErrCode_MotionRequestFailed	运动请求失败
10020	ErrCode_CreateMotionRequestFailed	生成运动请求失败
10021	ErrCode_MotionInterruptedByEvent	运动被事件中断
10022	ErrCode_MotionWaypointVetorSizeError	运动相关的路点容器的长度不符合规定
10023	ErrCode_ResponseReturnError	服务器响应返回错误

10024	ErrCode_RealRobotNoExist	真实机械臂不存在，因为有些接口只有在真是机械臂存在的情况下才可以被调用
10025	ErrCode_moveControlSlowStopFailed	调用缓停接口失败
10026	ErrCode_moveControlFastStopFailed	调用急停接口失败
10027	ErrCode_moveControlPauseFailed	调用暂停接口失败
10028	ErrCode_moveControlContinueFailed	调用继续接口失败

3.2 由于控制器异常事件导致的错误码

错误号	错误代码	错误信息
21001	ErrCodeMoveJConfigError	关节运动属性配置错误
21002	ErrCodeMoveLConfigError	直线运动属性配置错误
21003	ErrCodeMovePConfigError	轨迹运动属性配置错误
21004	ErrCodeInvailConfigError	无效的运动属性配置
21005	ErrCodeWaitRobotStopped	等待机器人停止
21006	ErrCodeJointOutRange	超出关节运动范围
21007	ErrCodeFirstWaypointSetError	请正确设置 MOVEP 第一个路点
21008	ErrCodeConveyorTrackConfigError	传送带跟踪配置错误
21009	ErrCodeConveyorTrackTrajectoryTypeError	传送带轨迹类型错误
21010	ErrCodeRelativeTransformIKFailed	相对坐标变换逆解失败
21011	ErrCodeTeachModeCollision	示教模式发生碰撞
21012	ErrCodeextErnalToolConfigError	运动属性配置错误,外部工具或手持工件配置错误

21101	ErrCodeTrajectoryAbnormal	轨迹异常
21102	ErrCodeOnlineTrajectoryPlanError	轨迹规划错误
21103	ErrCodeOnlineTrajectoryTypeIIError	二型在线轨迹规划失败
21104	ErrCodeIKFailed	逆解失败
21105	ErrCodeAbnormalLimitProtect	动力学限制保护
21106	ErrCodeConveyorTrackingFailed	传送带跟踪失败
21107	ErrCodeConveyorOutWorkingRange	超出传送带工作范围
21108	ErrCodeTrajectoryJointOutOfRange	关节超出范围
21109	ErrCodeTrajectoryJointOverspeed	关节超速
21110	ErrCodeOfflineTrajectoryPlanFailed	离线轨迹规划失败
21111	ErrCodeTrajectoryJointAccOutOfRange	轨迹异常,关节加速度超限
21120	ErrCodeForceModeException	力控模式异常
21121	ErrCodeForceModeIKFailed	轨迹异常，力控模式下失败
21122	ErrCodeForceModeTrackJointverspeed	关节超速
21200	ErrCodeControllerIKFailed	控制器异常，逆解失败
21201	ErrCodeControllerStatusException	控制器异常，状态异常
21202	ErrCodeControllerTrackingLost	关节跟踪误差过大
21203	ErrCodeMonitorErrTrackingLost	关节跟踪误差过大
21204	ErrCodeMonitorErrNoArrivalInTime	预留
21205	ErrCodeMonitorErrCurrentOverload	预留
21206	ErrCodeMonitorErrJointOutOfRange	机械臂关节超出限制范围
21207	ErrCodeFifoDataTimeNotRead	缓存区超时未更新
21300	ErrCodeMoveEnterStopState	运动进入到 stop 阶段
21301	ErrCodeMoveInterruptedByEvent	运动被未知事件中断

3.3 由于硬件层异常事件导致的错误码

错误号	错误代码	错误信息
22001	ErrCodeHardwareErrorNotify	机械臂硬件错误

		不能区分是哪种硬件异常才会返回该错误
22101	ErrCodeJointError	机械臂关节错误
22102	ErrCodeJointOverCurrent	机械臂关节过流
22103	ErrCodeJointOverVoltage	机械臂关节过压
22104	ErrCodeJointLowVoltage	机械臂关节欠压
22105	ErrCodeJointOverTemperature	机械臂关节过温
22106	ErrCodeJointHallError	机械臂关节霍尔错误
22107	ErrCodeJointEncoderError	机械臂关节编码器错误
22108	ErrCodeJointAbsoluteEncoderError	机械臂关节绝对编码器错误
22109	ErrCodeJointCurrentDetectError	机械臂关节当前位置错误
22110	ErrCodeJointEncoderPollustion	机械臂关节编码器污染。建议采取措施:警告性通知
22111	ErrCodeJointEncoderZSignalError	机械臂关节编码器 Z 信号错误
22112	ErrCodeJointEncoderCalibrateInvalid	机械臂关节编码器校准失效
22113	ErrCodeJoint_IMU_SensorInvalid	机械臂关节 IMU 传感器失效
22114	ErrCodeJointTemperatureSensorError	机械臂关节温度传感器出错
22115	ErrCodeJointCanBusError	机械臂关节 CAN 总线出错
22116	ErrCodeJointCurrentError	机械臂关节当前电流错误
22117	ErrCodeJointCurrentPositionError	机械臂关节当前位置错误
22118	ErrCodeJointOverSpeed	机械臂关节超速
22119	ErrCodeJointOverAccelerate	机械臂关节加速度过大错误
22120	ErrCodeJointTraceAccuracy	机械臂关节跟踪精度错误
22121	ErrCodeJointTargetPositionOutOfRange	机械臂关节目标位置超范围
22122	ErrCodeJointTargetSpeedOutOfRange	机械臂关节目标速度超范围

22123	ErrCodeJointCollision	建议采取措施:暂停当前运动
22200	ErrCodeDataAbnormal	机械臂信息异常
22201	ErrCodeRobotTypeError	机械臂类型错误
22202	ErrCodeAccelerationSensorError	机械臂加速度计芯片错误
22203	ErrCodeEncoderLineError	机械臂编码器线数错误
22204	ErrCodeEnterDragAndTeachModeError	机械臂进入拖动示教模式错误
22205	ErrCodeExitDragAndTeachModeError	机械臂退出拖动示教模式错误
22206	ErrCodeMACDataInterruptionError	机械臂 MAC 数据中断错误
22207	ErrCodeDriveVersionError	驱动器版本错误(关节固件版本不一致)
22300	ErrCodeInitAbnormal	机械臂初始化异常
22301	ErrCodeDriverEnableFailed	机械臂驱动器使能失败
22302	ErrCodeDriverEnableAutoBackFailed	机械臂驱动器使能自动回应失败
22303	ErrCodeDriverEnableCurrentLoopFailed	机械臂驱动器使能电流环失败
22304	ErrCodeDriverSetTargetCurrentFailed	机械臂驱动器设置目标电流失败
22305	ErrCodeDriverReleaseBrakeFailed	机械臂释放刹车失败
22306	ErrCodeDriverEnablePostionLoopFailed	机械臂使能位置环失败
22307	ErrCodeSetMaxAccelerateFailed	设置最大加速度失败
22400	ErrCodeSafetyError	机械臂安全出错
22401	ErrCodeExternEmergencyStop	机械臂外部紧急停止
22402	ErrCodeSystemEmergencyStop	机械臂系统紧急停止
22403	ErrCodeTeachpendantEmergencyStop	机械臂示教器紧急停止
22404	ErrCodeControlCabinetEmergencyStop	机械臂控制柜紧急停止
22405	ErrCodeProtectionStopTimeout	机械臂保护停止

		超时
22406	ErrCodeEeducedModeTimeout	机械臂缩减模式超时
22500	ErrCodeSystemAbnormal	机械臂系统异常
22501	ErrCode_MCU_CommunicationAbnormal	机械臂 mcu 通信异常
22502	ErrCode485CommunicationAbnormal	机械臂 485 通信异常
22550	ErrCodeSoftEmergency	软急停
22600	ErrCodeArmPowerOff	控制柜接触器断开导致机械臂 48V 断电

4 使用案例

4.1 使用 SDK 构建一个最简单的机械臂的控制工程

本案例是使用 SDK 来构建一个最简单的机械臂的控制工程。

程序的主要流程是：（1）机械臂登录（2）上电初始化（3）模拟业务（4）机械臂关机（5）退出登录。

example.h 代码如下：

```
#pragma once
#include <string>
#include "rsdef.h"

//机械臂登录
bool example_login(RSHD& rshd, const char* addr, int port);

//启动机械臂（必须连接真实机械臂）
bool example_robotStartup(RSHD rshd);

//关节机械臂（必须连接真实机械臂）
bool example_robotShutdown(RSHD rshd);

//机械臂退出登录
bool example_logout(RSHD rshd);
```

example.cpp 代码如下：

```
#include "example.h"
#include "rsdef.h"

#include <Windows.h>
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <algorithm>

//机械臂登录
bool example_login(RSHD& rshd, const char* addr, int port)
{
    bool result = false;
    rshd = RS_FAILED;
    //初始化接口库
```

```
    if(rs_initialize() == RS_SUCC)
    {
        //创建上下文
        if(rs_create_context(&rshd) == RS_SUCC)
        {
            //机械臂登录
            if(rs_login(rshd, addr, port) == RS_SUCC)
            {
                result = true;
                std::cout << "机械臂登录成功" << std::endl;
            }
            else
            {
                std::cerr << "机械臂登录失败" << std::endl;
            }
        }
        else
        {
            std::cerr << "创建上下文失败" << std::endl;
        }
    }
    else
    {
        //初始化接口库失败
        std::cerr << "初始化接口库失败" << std::endl;
    }

    return result;
}

//启动机械臂（必须连接真实机械臂）
bool example_robotStartup(RSHD rshd)
{
    bool result = false;
    //工具的动力学参数
    ToolDynamicsParam tool_dynamics = { 0 };
    //机械臂碰撞等级
    uint8 colli_class = 6;
    //机械臂启动是否读取姿态（默然开启）
    bool read_pos = true;
    //机械臂静态碰撞检测（默认开启）
    bool static_colli_detect = true;
    //机械臂最大加速度（系统自动控制，默认为 30000）
    int board_maxacc = 30000;
```

```
    //机械臂服务启动状态
    ROBOT_SERVICE_STATE state = ROBOT_SERVICE_READY;
    if(rs_robot_startup(rshd, &tool_dynamics, colli_class, read_pos, static_colli_detec
t, board_maxacc, &state) == RS_SUCC)
    {
        result = true;
        std::cout << "机械臂上电成功，机械臂状态：" << state << std::endl;
    }
    else
    {
        std::cerr << "机械臂上电失败。" << std::endl;
    }
    return result;
}

//关闭机械臂（必须连接真实机械臂）
bool example_robotShutdown(RSHD rshd)
{
    bool result = false;
    if(rs_robot_shutdown(rshd) == RS_SUCC)
    {
        result = true;
        std::cout << "机械臂断电成功" << std::endl;
    }
    else
    {
        std::cerr << "机械臂断电失败" << std::endl;
    }
    return result;
}

//退出登录
bool example_logout(RSHD rshd)
{
    return rs_logout(rshd) == RS_SUCC ? true : false;
}
```

main.cpp 代码如下:

注意：要将下面代码中的 IP 地址（ROBOT_ADDR）修改为对应服务器的 IP 地址。

```
#include "example.h"
#include "windows.h"

#define ROBOT_ADDR "192.168.1.100"
```

```
#define ROBOT_PORT 8899

//机械臂控制上下文句柄
RSHD g_rshd = -1;

int main(int argc, char* argv[])
{
    //登录服务器
    if (example_login(g_rshd, ROBOT_ADDR, ROBOT_PORT))
    {
        //机械臂上电(必须连接真实机械臂)
        example_robotStartup(g_rshd);
        //模拟业务

        //关闭机械臂（必须连接真实机械臂）
        example_robotShutdown(g_rshd);

        //退出登录
        example_logout(g_rshd);
    }
    else
    {
        std::cout << "机械臂登录失败。" << std::endl;
    }

    //反初始化接口库
    rs_uninitialize();

    std::cout << "Please enter to exit" << std::endl;
    getchar();

    return 0;
}
```

4.2 用回调函数的方式来获取实时信息

4.2.1 获取实时路点信息

本案例是用回调函数的方式来获取实时路点信息。

example.h 相关代码如下：

```
#pragma once
#include <string>
#include "rsdef.h"

//获取实时路点信息的回调函数类型
void callback_RealTimeRoadPoint(const aubo_robot_namespace::wayPoint_S* wayP
oint, void* arg);

//实时路点信息回调函数
bool example_callbackRobotRoadPoint(RSHD rshd);
```

example.cpp 相关代码如下：

```
#include "example.h"
#include "rsdef.h"

#include <Windows.h>
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <algorithm>

#define M_PI 3.14159265358979323846

//获取实时路点信息的回调函数类型
void callback_RealTimeRoadPoint(const aubo_robot_namespace::wayPoint_S* wayP
oint, void* arg)
{
    //打印路点信息
    std::cout << "----------------------------------------------------打印路点信息----------
----------------------------------------" << std::endl;
    std::cout << "pos.x = " << wayPoint->cartPos.position.x << "    ";
    std::cout << "pos.y = " << wayPoint->cartPos.position.y << "    ";
    std::cout << "pos.z = " << wayPoint->cartPos.position.z << std::endl;

    std::cout << "ori.w = " << wayPoint->orientation.w << "    ";
```

```
    std::cout << "ori.x = " << wayPoint->orientation.x   << "     ";
    std::cout << "ori.y = " << wayPoint->orientation.y   << "     ";
    std::cout << "ori.z = " << wayPoint->orientation.z   << std::endl;

    std::cout << "joint_1 = " << wayPoint->jointpos[0] * 180.0 / M_PI << "     ";
    std::cout << "joint_2 = " << wayPoint->jointpos[1] * 180.0 / M_PI << "     ";
    std::cout << "joint_3 = " << wayPoint->jointpos[2] * 180.0 / M_PI << "     ";
    std::cout << "joint_4 = " << wayPoint->jointpos[3] * 180.0 / M_PI << "     ";
    std::cout << "joint_5 = " << wayPoint->jointpos[4] * 180.0 / M_PI << "     ";
    std::cout << "joint_6 = " << wayPoint->jointpos[5] * 180.0 / M_PI << std::en
dl;

    std::cout << "----------------------------------------------------------------------------
-------------------------------------" << std::endl;

}


//实时路点信息回调函数
bool example_callbackRobotRoadPoint(RSHD rshd)
{
    bool result = false;
    //允许实时路点信息推送
    if(rs_enable_push_realtime_roadpoint(rshd, true) == RS_SUCC)
    {
        if (rs_setcallback_realtime_roadpoint(rshd, callback_RealTimeRoadPoint, N
ULL))
        {
            result = true;
        }
        else
        {
            std::cerr << "调用 rs_setcallback_realtime_roadpoint 函数失败" << st
d::endl;
        }
    }
    else
    {
        std::cerr << "调用 rs_enable_push_realtime_roadpoint 函数失败" << std::en
dl;
    }
    return result;
}
```

main.cpp 代码如下：

注意：要将下面代码中的 IP 地址（ROBOT_ADDR）修改为对应服务器的 IP 地址。

```
#include "example.h"
#include "windows.h"

#define ROBOT_ADDR "192.168.1.100"
#define ROBOT_PORT 8899

//机械臂控制上下文句柄
RSHD g_rshd = -1;

int main(int argc, char* argv[])
{
    //登录服务器
    if (example_login(g_rshd, ROBOT_ADDR, ROBOT_PORT))
    {
        //机械臂上电(必须连接真实机械臂)
        example_robotStartup(g_rshd);

        //用回调函数获取实时路点信息
        example_callbackRobotRoadPoint(g_rshd);
        //延时 3 秒，观察回调函数
        Sleep(3000);
    }
    else
    {
        std::cout << "机械臂登录失败。" << std::endl;
    }

    //反初始化接口库
    rs_uninitialize();

    std::cout << "Please enter to exit" << std::endl;
    getchar();

    return 0;
}
```

4.2.2 获取实时关节状态信息

本案例是用回调函数的方式来获取实时关节状态信息。

example.h 相关代码如下：

```
#pragma once
#include <string>
#include "rsdef.h"

//获取实时关节状态信息的回调函数类型
void callback_RealTimeJointStatus(const aubo_robot_namespace::JointStatus* jointSt
atus, int size, void* arg);

//实时关节状态信息回调函数
bool example_callbackJointStatus(RSHD rshd);
```

example.cpp 相关代码如下：

```
#include "example.h"
#include "rsdef.h"

#include <Windows.h>
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <algorithm>

#define M_PI 3.14159265358979323846

//获取实时关节状态信息回调函数类型
void callback_RealTimeJointStatus(const aubo_robot_namespace::JointStatus* jointSt
atus, int size, void* arg)
{
    std::cout << "----------关节状态信息----------" << std::endl;

    for (int i = 0; i < size; i++)
    {
        std::cout << "关节 ID：" << i + 1 << "  ";
        std::cout << "关节电流：" << jointStatus[i].jointCurrentI << "  ";
        std::cout << "关节速度：" << jointStatus[i].jointSpeedMoto << "  ";
        std::cout << "关节角：" << jointStatus[i].jointPosJ * 180 / M_PI << "
";
        std::cout << "关节电压：" << jointStatus[i].jointCurVol << "  ";
        std::cout << "当前温度：" << jointStatus[i].jointCurTemp << "  ";
```

```
        std::cout << "电机目标电流：" << jointStatus[i].jointTagCurrentI << "   ";
        std::cout << "电机目标速度：" << jointStatus[i].jointTagSpeedMoto << "
 ";
        std::cout << "目标关节角：" << jointStatus[i].jointTagPosJ << "   ";
        std::cout << "关节错误码：" << jointStatus[i].jointErrorNum << std::endl;
    }

    std::cout << std::endl;
}


//实时关节状态信息回调函数
bool example_callbackJointStatus(RSHD rshd)
{
    bool result = false;

    //允许实时关节信息推送
    if (rs_enable_push_realtime_joint_status(rshd, true) == RS_SUCC)
    {
        if (rs_setcallback_realtime_joint_status(rshd, callback_RealTimeJointStatus,
NULL))
        {
            result = true;
        }
        else
        {
            std::cerr << "调用 rs_setcallback_realtime_joint_status 函数失败" <<
std::endl;
        }
    }
    else
    {
        std::cerr << "调用 rs_enable_push_realtime_joint_status 函数失败" << std::
endl;
    }

    return result;
}
```

main.cpp 代码如下:

注意：要将下面代码中的 IP 地址（ROBOT_ADDR）修改为对应服务器的 IP 地址。

```
#include "example.h"
#include "windows.h"
```

```
#define ROBOT_ADDR "192.168.1.100"
#define ROBOT_PORT 8899

//机械臂控制上下文句柄
RSHD g_rshd = -1;

int main(int argc, char* argv[])
{
    //登录服务器
    if (example_login(g_rshd, ROBOT_ADDR, ROBOT_PORT))
    {
        //机械臂上电(必须连接真实机械臂)
        example_robotStartup(g_rshd);

        //用回调函数获取实时关节状态信息
        example_callbackJointStatus(g_rshd);
        //延时 3 秒，观察回调函数
        Sleep(3000);
    }
    else
    {
        std::cout << "机械臂登录失败。" << std::endl;
    }

    //反初始化接口库
    rs_uninitialize();

    std::cout << "Please enter to exit" << std::endl;
    getchar();

    return 0;
}
```

4.2.3 获取实时末端速度信息

本案例是用回调函数的方式来获取实时末端速度信息。

example.h 相关代码如下:

```
#pragma once
#include <string>
#include "rsdef.h"
//获取实时末端速度的回调函数类型
void callback_RealTimeEndSpeed(double speed, void* arg);

//实时末端速度回调函数
bool example_callbackEndSpeed(RSHD rshd);
```

example.cpp 相关代码如下:

```
#include "example.h"
#include "rsdef.h"

#include <Windows.h>
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <algorithm>

//获取实时末端速度信息的回调函数类型
void callback_RealTimeEndSpeed(double speed, void* arg)
{
    std::cout << "实时末端速度为：" << speed << std::endl;
}


//实时末端速度信息回调函数
bool example_callbackEndSpeed(RSHD rshd)
{
    bool result = false;

    //允许实时末端速度信息推送
    if (rs_enable_push_realtime_end_speed(rshd, true) == RS_SUCC)
    {
        if (rs_setcallback_realtime_end_speed(rshd, callback_RealTimeEndSpeed, N
ULL))
        {
            result = true;
```

```
        }
        else
        {
            std::cerr << "调用 rs_setcallback_realtime_end_speed 函数失败" << st
d::endl;
        }
    }
    else
    {
        std::cerr << "调用 rs_enable_push_realtime_end_speed 函数失败" << std::e
ndl;
    }

    return result;
}
```

main.cpp 代码如下:

注意：要将下面代码中的 IP 地址（ROBOT_ADDR）修改为对应服务器的 IP 地址。

```
#include "example.h"
#include "windows.h"

#define ROBOT_ADDR "192.168.1.100"
#define ROBOT_PORT 8899

//机械臂控制上下文句柄
RSHD g_rshd = -1;

int main(int argc, char* argv[])
{
    //登录服务器
    if (example_login(g_rshd, ROBOT_ADDR, ROBOT_PORT))
    {
        //机械臂上电(必须连接真实机械臂)
        example_robotStartup(g_rshd);

        //用回调函数获取实时末端速度
        example_callbackEndSpeed(g_rshd);
        //延时 3 秒，观察回调函数
        Sleep(3000);
    }
    else
    {
        std::cout << "机械臂登录失败。" << std::endl;
```

```
    }

    //反初始化接口库
    rs_uninitialize();

    std::cout << "Please enter to exit" << std::endl;
    getchar();

    return 0;
}
```

4.2.4 获取机械臂的事件信息

本案例是用回调函数的方式来获取机械臂的事件信息。

example.h 代码如下：

```
#pragma once
#include <string>
#include "rsdef.h"

//机械臂事件的回调函数类型
void callback_RealTimeEventInfo(const aubo_robot_namespace::RobotEventInfo* ev
entInfo, void* arg);

//获取机械臂事件信息回调函数
bool example_callbackEventInfo(RSHD rshd);
```

example.cpp 代码如下：

```
#include "example.h"
#include "rsdef.h"

#include <Windows.h>
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <algorithm>

//获取机械臂事件信息的回调函数类型
void callback_RealTimeEventInfo(const aubo_robot_namespace::RobotEventInfo* ev
entInfo, void* arg)
{
```

```
    std::cout << "----------获取机械臂事件信息----------" << std::endl;
    std::cout << "事件类型：" << eventInfo->eventType << std::endl;
    std::cout << "事件码：" << eventInfo->eventCode << std::endl;
}

//获取机械臂事件信息回调函数
bool example_callbackEventInfo(RSHD rshd)
{
    bool result = false;

    if (rs_setcallback_robot_event(rshd, callback_RealTimeEventInfo, NULL) == RS
_SUCC)
    {
        result = true;
    }
    else
    {
        std::cerr << "调用 rs_setcallback_robot_event 函数失败" << std::endl;
    }

    return result;
}
```

main.cpp 代码如下:

注意：要将下面代码中的 IP 地址（ROBOT_ADDR）修改为对应服务器的 IP 地址。

```
#include "example.h"
#include "windows.h"

#define ROBOT_ADDR "192.168.1.100"
#define ROBOT_PORT 8899

//机械臂控制上下文句柄
RSHD g_rshd = -1;

int main(int argc, char* argv[])
{
    //登录服务器
    if (example_login(g_rshd, ROBOT_ADDR, ROBOT_PORT))
    {
        //机械臂上电(必须连接真实机械臂)
        example_robotStartup(g_rshd);
```

```
        //用回调函数获取机械臂的事件信息
        example_callbackEventInfo(g_rshd);
        //延时 3 秒，观察回调函数
        Sleep(3000);
    }
    else
    {
        std::cout << "机械臂登录失败。" << std::endl;
    }

    //反初始化接口库
    rs_uninitialize();

    std::cout << "Please enter to exit" << std::endl;
    getchar();

    return 0;
}
```

4.2.5 获取日志信息

本案例是用回调函数的方式来获取日志信息。

example.h 代码如下:

```
#pragma once
#include <string>
#include "rsdef.h"

//获取日志信息的回调函数类型
void callback_RealTimeLogInfo(aubo_robot_namespace::LOG_LEVEL logLevel, co
nst char* str, void* arg);

//获取函数的日志信息回调函数
bool example_callbackLogInfo(RSHD rshd);
```

example.cpp 代码如下:

```
#include "example.h"
#include "rsdef.h"

#include <Windows.h>
#include <iostream>
#include <fstream>
```

```
#include <sstream>
#include <string>
#include <algorithm>

//获取日志信息的回调函数类型
void callback_RealTimeLogInfo(aubo_robot_namespace::LOG_LEVEL logLevel, co
nst char* str, void* arg)
{
    std::cout << "日志级别：" << logLevel << std::endl;
    std::cout << "日志内容：" << str << std::endl;
}

//获取函数的日志信息回调函数
bool example_callbackLogInfo(RSHD rshd)
{
    bool result = false;

    if (rs_setcallback_robot_loginfo(rshd, callback_RealTimeLogInfo, NULL) == RS
_SUCC)
    {
        result = true;
    }
    else
    {
        std::cerr << "调用 rs_setcallback_robot_loginfo 函数失败" << std::endl;
    }

    return result;
}
```

main.cpp 代码如下：

注意：要将下面代码中的 IP 地址（ROBOT_ADDR）修改为对应服务器的 IP 地址。

```
#include "example.h"
#include "windows.h"

#define ROBOT_ADDR "192.168.1.100"
#define ROBOT_PORT 8899

//机械臂控制上下文句柄
RSHD g_rshd = -1;

int main(int argc, char* argv[])
{
```

```
    //登录服务器
    if (example_login(g_rshd, ROBOT_ADDR, ROBOT_PORT))
    {
        //机械臂上电(必须连接真实机械臂)
        example_robotStartup(g_rshd);

        //用回调函数获取日志信息
        example_callbackLogInfo(g_rshd);
        //延时 3 秒，观察回调函数
        Sleep(3000);
    }
    else
    {
        std::cout << "机械臂登录失败。" << std::endl;
    }

    //反初始化接口库
    rs_uninitialize();

    std::cout << "Please enter to exit" << std::endl;
    getchar();

    return 0;
}
```

4.3 正逆解

本案例是用 SDK 来实现机器人运动学正解与逆解的功能。

example.h 相关代码如下:

```
#pragma once
#include <string>
#include "rsdef.h"

//正逆解
void example_ik_fk(RSHD rshd);
```

example.cpp 相关代码如下:

```
#include "example.h"
#include "rsdef.h"

#include <Windows.h>
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <algorithm>

#define M_PI 3.14159265358979323846

//正逆解
void example_ik_fk(RSHD rshd)
{
    aubo_robot_namespace::wayPoint_S wayPoint;
    ik_solutions solutions;
    double jointAngle[aubo_robot_namespace::ARM_DOF] = { -0.000003, -0.12726
7, -1.321122, 0.376934, -1.570796, -0.000008 };

    //获得正解
    if (RS_SUCC == rs_forward_kin(rshd, jointAngle, &wayPoint))
    {
        std::cout << std::endl;
        std::cout << "正解成功。" << std::endl;

        std::cout << "位置:  " << "x = " << wayPoint.cartPos.position.x << " "
                               << "y = " << wayPoint.cartPos.position.y << " "
                               << "z = " << wayPoint.cartPos.position.z << " "
                               << std::endl;
```

```
        std::cout << "姿态（四元数）：  "
                  << "w = " << wayPoint.orientation.w << " "
                  << "x = " << wayPoint.orientation.x << " "
                  << "y = " << wayPoint.orientation.y << " "
                  << "z = " << wayPoint.orientation.z << " "
                  << std::endl;

        aubo_robot_namespace::Rpy rpy_fk;
        rs_quaternion_to_rpy(rshd, &wayPoint.orientation, &rpy_fk);
        std::cout << "姿态（欧拉角）：  "
                  << "x = " << rpy_fk.rx * 180 / M_PI << " "
                  << "y = " << rpy_fk.ry * 180 / M_PI << " "
                  << "z = " << rpy_fk.rz * 180 / M_PI << " "
                  << std::endl;
    }
    else
    {
        std::cerr << "正解失败。" << std::endl;
    }

    //获得最优逆解
    double startPointJointAngle[aubo_robot_namespace::ARM_DOF] = { 0.0 / 180.
0 * M_PI,  0.0 / 180.0 * M_PI,  0.0 / 180.0 * M_PI, 0.0 / 180.0 * M_PI, 0.0
/ 180.0 * M_PI,0.0 / 180.0 * M_PI };

    aubo_robot_namespace::Pos targetPosition;
    targetPosition.x = -0.400;
    targetPosition.y = -0.1215;
    targetPosition.z = 0.5476;

    aubo_robot_namespace::Rpy rpy;
    aubo_robot_namespace::Ori targetOri;
    rpy.rx = 180.0 / 180.0 * M_PI;
    rpy.ry = 0.0 / 180.0 * M_PI;
    rpy.rz = -90.0 / 180.0 * M_PI;
    rs_rpy_to_quaternion(rshd, &rpy, &targetOri);

    if (RS_SUCC == rs_inverse_kin(rshd, startPointJointAngle, &targetPosition, &t
argetOri, &wayPoint))
    {
        std::cout << std::endl;
        std::cout << "逆解成功。" << std::endl;
        std::cout << "最优逆解为："
            << "关节 1: " << wayPoint.jointpos[0] * 180 / M_PI<< " "
```

```
            << "关节 2: " << wayPoint.jointpos[1] * 180 / M_PI << " "
            << "关节 3: " << wayPoint.jointpos[2] * 180 / M_PI << " "
            << "关节 4: " << wayPoint.jointpos[3] * 180 / M_PI << " "
            << "关节 5: " << wayPoint.jointpos[4] * 180 / M_PI << " "
            << "关节 6: " << wayPoint.jointpos[5] * 180 / M_PI << std::endl;
    }
    else
    {
        std::cerr << "逆解失败。" << std::endl;
    }

    //获得逆解集
    if (RS_SUCC == rs_inverse_kin_closed_form(rshd, &targetPosition, &targetOri,
&solutions))
    {
        std::cout << std::endl;
        std::cout << "逆解成功。"
                  << "一共有" << solutions.solution_count  << "组解。"
                  << std::endl;

        for (int i = 0; i < solutions.solution_count; i++)
        {
            std::cout << "第" << i + 1 << "组解：" << std::endl;
            for (int j = 0; j < 6; j++)
            {
                std::cout << "关节" << j + 1 << ": " << solutions.waypoint[i].j
ointpos[j] * 180 / M_PI << std::endl;

            }
        }
    }
    else
    {
        std::cerr << "逆解失败。" << std::endl;
    }
}
```

main.cpp 代码如下:

注意：要将下面代码中的 IP 地址（ROBOT_ADDR）修改为对应服务器的 IP 地址。

```
#include "example.h"
#include "windows.h"
```

```
#define ROBOT_ADDR "192.168.1.100"
#define ROBOT_PORT 8899

//机械臂控制上下文句柄
RSHD g_rshd = -1;

int main(int argc, char* argv[])
{
    //登录服务器
    if (example_login(g_rshd, ROBOT_ADDR, ROBOT_PORT))
    {
        //机械臂上电(必须连接真实机械臂)
        example_robotStartup(g_rshd);

        //正逆解
        example_ik_fk(g_rshd);
    }
    else
    {
        std::cout << "机械臂登录失败。" << std::endl;
    }

    //反初始化接口库
    rs_uninitialize();

    std::cout << "Please enter to exit" << std::endl;
    getchar();

    return 0;
}
```

4.4 坐标系转换

4.4.1 基坐标系转用户坐标系 rs_base_to_user()

rs_base_to_user 函数示例 1

本示例是将法兰盘中心在基坐标系下的位置和姿态转成工具在用户坐标系下的位置和姿态。

example.h 代码如下:

```
#pragma once
#include <string>
#include "rsdef.h"

//基坐标系转用户坐标系 rs_base_to_user
//示例 1：将法兰盘中心在基坐标系下的位置和姿态转成工具在用户坐标系下的位置和姿态
void example_base_to_user1(RSHD rshd);
```

example.cpp 代码如下:

```
#include "example.h"
#include "rsdef.h"

#include <Windows.h>
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <algorithm>

#define M_PI 3.14159265358979323846

//基坐标系转用户坐标系 rs_base_to_user
//示例 1：将法兰盘中心在基坐标系下的位置和姿态转成工具在用户坐标系下的位置和姿态
void example_base_to_user1(RSHD rshd)
{
    aubo_robot_namespace::Pos flangeCenterPosOnBase;//法兰盘中心在基坐标系下的位置
    flangeCenterPosOnBase.x = 0.420750;
    flangeCenterPosOnBase.y = 0.072954;
```

```
    flangeCenterPosOnBase.z = 0.595753;

    aubo_robot_namespace::Rpy flangeCenterRpyOnBase;//法兰盘中心在基坐标系下的姿态（欧拉角）
    flangeCenterRpyOnBase.rx = 148.947708 * M_PI / 180;
    flangeCenterRpyOnBase.ry = 14.759964 * M_PI / 180;
    flangeCenterRpyOnBase.rz = 107.579117 * M_PI / 180;

    aubo_robot_namespace::Ori flangeCenterOriOnBase;//法兰盘中心在基坐标系下的姿态（四元数）
    rs_rpy_to_quaternion(rshd, &flangeCenterRpyOnBase, &flangeCenterOriOnBase);

    aubo_robot_namespace::CoordCalibrateByJointAngleAndTool userCoord;//用户坐标系
    userCoord.coordType = aubo_robot_namespace::WorldCoordinate;
    userCoord.methods = aubo_robot_namespace::Origin_AnyPointOnPositiveXAxis_AnyPointOnFirstQuadrantOfXOYPlane;

    userCoord.wayPointArray[0].jointPos[0] = -75.093279 * M_PI / 180;
    userCoord.wayPointArray[0].jointPos[1] = 28.544643 * M_PI / 180;
    userCoord.wayPointArray[0].jointPos[2] = -114.313905 * M_PI / 180;
    userCoord.wayPointArray[0].jointPos[3] = -62.769247 * M_PI / 180;
    userCoord.wayPointArray[0].jointPos[4] = -87.343517 * M_PI / 180;
    userCoord.wayPointArray[0].jointPos[5] = -27.888262 * M_PI / 180;

    userCoord.wayPointArray[1].jointPos[0] = -89.239837 * M_PI / 180;
    userCoord.wayPointArray[1].jointPos[1] = 23.936171 * M_PI / 180;
    userCoord.wayPointArray[1].jointPos[2] = -122.299277 * M_PI / 180;
    userCoord.wayPointArray[1].jointPos[3] = -65.208902 * M_PI / 180;
    userCoord.wayPointArray[1].jointPos[4] = -85.011123 * M_PI / 180;
    userCoord.wayPointArray[1].jointPos[5] = -41.87417 * M_PI / 180;

    userCoord.wayPointArray[2].jointPos[0] = -77.059212 * M_PI / 180;
    userCoord.wayPointArray[2].jointPos[1] = 35.509518 * M_PI / 180;
    userCoord.wayPointArray[2].jointPos[2] = -101.108547 * M_PI / 180;
    userCoord.wayPointArray[2].jointPos[3] = -56.433133 * M_PI / 180;
    userCoord.wayPointArray[2].jointPos[4] = -87.006734 * M_PI / 180;
    userCoord.wayPointArray[2].jointPos[5] = -29.827440 * M_PI / 180;

    aubo_robot_namespace::ToolInEndDesc toolUserCoord;
    toolUserCoord.toolInEndPosition.x = 0;
    toolUserCoord.toolInEndPosition.y = 0;
    toolUserCoord.toolInEndPosition.z = 0.45;
    toolUserCoord.toolInEndOrientation.w = 1;
```

```
    toolUserCoord.toolInEndOrientation.x = 0;
    toolUserCoord.toolInEndOrientation.y = 0;
    toolUserCoord.toolInEndOrientation.z = 0;
    userCoord.toolDesc = toolUserCoord;

    aubo_robot_namespace::ToolInEndDesc toolInEndDesc;
    toolInEndDesc.toolInEndPosition.x = 0;
    toolInEndDesc.toolInEndPosition.y = 0;
    toolInEndDesc.toolInEndPosition.z = 0.45;
    toolInEndDesc.toolInEndOrientation.w = 1;
    toolInEndDesc.toolInEndOrientation.x = 0;
    toolInEndDesc.toolInEndOrientation.y = 0;
    toolInEndDesc.toolInEndOrientation.z = 0;

    aubo_robot_namespace::Pos toolEndPosOnUser;//工具末端在用户坐标系下的位置
    aubo_robot_namespace::Ori toolEndOriOnUser;//工具末端在用户坐标系下的姿态

    //将法兰盘中心在基坐标系下的位置姿态转化为工具末端在用户坐标系下的位置和姿态
    rs_base_to_user(rshd, &flangeCenterPosOnBase, &flangeCenterOriOnBase, &userCoord, &toolInEndDesc, &toolEndPosOnUser, &toolEndOriOnUser);
    std::cout << "工具末端在用户坐标系下的位置: ";
    std::cout << "(" << toolEndPosOnUser.x << ", " << toolEndPosOnUser.y << ", " << toolEndPosOnUser.z << ")";
    std::cout << std::endl;

    std::cout << "工具末端在用户坐标系下的姿态（四元数）: ";
    std::cout << "(" << toolEndOriOnUser.w << ", " << toolEndOriOnUser.x << ", " << toolEndOriOnUser.y << ", " << toolEndOriOnUser.z << ")";
    std::cout << std::endl;

    aubo_robot_namespace::Rpy toolEndRpyOnUser;
    rs_quaternion_to_rpy(rshd, &toolEndOriOnUser, &toolEndRpyOnUser);
    std::cout << "工具末端在用户坐标系下的姿态（欧拉角）: ";
    std::cout << "(" << toolEndRpyOnUser.rx * 180 / M_PI << ", " << toolEndRpyOnUser.ry * 180 / M_PI << ", " << toolEndRpyOnUser.rz * 180 / M_PI << ")";
    std::cout << std::endl;
}
```

main.cpp 代码如下:

注意：要将下面代码中的 IP 地址（ROBOT_ADDR）修改为对应服务器的 IP 地址。

```
#include "example.h"
#include "windows.h"

#define ROBOT_ADDR "192.168.1.100"
#define ROBOT_PORT 8899

//机械臂控制上下文句柄
RSHD g_rshd = -1;

int main(int argc, char* argv[])
{
    //登录服务器
    if (example_login(g_rshd, ROBOT_ADDR, ROBOT_PORT))
    {
        //机械臂上电(必须连接真实机械臂)
        example_robotStartup(g_rshd);

        //基坐标系转用户坐标系 rs_base_to_user
        //示例 1：将法兰盘中心在基坐标系下的位置和姿态转成工具在用户坐标系下的位置和姿态
        example_base_to_user1(g_rshd);
    }
    else
    {
        std::cout << "机械臂登录失败。" << std::endl;
    }

    //反初始化接口库
    rs_uninitialize();

    std::cout << "Please enter to exit" << std::endl;
    getchar();

    return 0;
}
```

rs_base_to_user 函数示例 2

本示例是将法兰盘中心在基坐标系下的位置和姿态转成法兰盘中心在用户坐标系下的位置和姿态。法兰盘中心可看成是一个位置(0,0,0)姿态(1,0,0,0)的特殊的工具。

example.h 代码如下:

```
#pragma once
#include <string>
#include "rsdef.h"

//示例 2：将法兰盘中心在基坐标系下的位置和姿态转成法兰盘中心在用户坐标系下的位置和姿态
void example_base_to_user2(RSHD rshd);
```

example.cpp 代码如下:

```
#include "example.h"
#include "rsdef.h"

#include <Windows.h>
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <algorithm>



#define M_PI 3.14159265358979323846

//示例 2：将法兰盘中心在基坐标系下的位置和姿态转成法兰盘中心在用户坐标系下的位置和姿态
void example_base_to_user2(RSHD rshd)
{
    aubo_robot_namespace::Pos flangeCenterPosOnBase;//法兰盘中心在基坐标系下的位置
    flangeCenterPosOnBase.x = 0.420750;
    flangeCenterPosOnBase.y = 0.072954;
    flangeCenterPosOnBase.z = 0.595753;

    aubo_robot_namespace::Rpy flangeCenterRpyOnBase;//法兰盘中心在基坐标系下的姿态（欧拉角）
    flangeCenterRpyOnBase.rx = 148.947708 * M_PI / 180;
    flangeCenterRpyOnBase.ry = 14.759964 * M_PI / 180;
```

```
    flangeCenterRpyOnBase.rz = 107.579117 * M_PI / 180;

    aubo_robot_namespace::Ori flangeCenterOriOnBase;//法兰盘中心在基坐标系下的姿态（四元数）
    rs_rpy_to_quaternion(rshd, &flangeCenterRpyOnBase, &flangeCenterOriOnBase);

    aubo_robot_namespace::CoordCalibrateByJointAngleAndTool userCoord;//用户坐标系
    userCoord.coordType = aubo_robot_namespace::WorldCoordinate;
    userCoord.methods = aubo_robot_namespace::Origin_AnyPointOnPositiveXAxis_AnyPointOnFirstQuadrantOfXOYPlane;

    userCoord.wayPointArray[0].jointPos[0] = -75.093279 * M_PI / 180;
    userCoord.wayPointArray[0].jointPos[1] = 28.544643 * M_PI / 180;
    userCoord.wayPointArray[0].jointPos[2] = -114.313905 * M_PI / 180;
    userCoord.wayPointArray[0].jointPos[3] = -62.769247 * M_PI / 180;
    userCoord.wayPointArray[0].jointPos[4] = -87.343517 * M_PI / 180;
    userCoord.wayPointArray[0].jointPos[5] = -27.888262 * M_PI / 180;

    userCoord.wayPointArray[1].jointPos[0] = -89.239837 * M_PI / 180;
    userCoord.wayPointArray[1].jointPos[1] = 23.936171 * M_PI / 180;
    userCoord.wayPointArray[1].jointPos[2] = -122.299277 * M_PI / 180;
    userCoord.wayPointArray[1].jointPos[3] = -65.208902 * M_PI / 180;
    userCoord.wayPointArray[1].jointPos[4] = -85.011123 * M_PI / 180;
    userCoord.wayPointArray[1].jointPos[5] = -41.87417 * M_PI / 180;

    userCoord.wayPointArray[2].jointPos[0] = -77.059212 * M_PI / 180;
    userCoord.wayPointArray[2].jointPos[1] = 35.509518 * M_PI / 180;
    userCoord.wayPointArray[2].jointPos[2] = -101.108547 * M_PI / 180;
    userCoord.wayPointArray[2].jointPos[3] = -56.433133 * M_PI / 180;
    userCoord.wayPointArray[2].jointPos[4] = -87.006734 * M_PI / 180;
    userCoord.wayPointArray[2].jointPos[5] = -29.827440 * M_PI / 180;

    aubo_robot_namespace::ToolInEndDesc toolUserCoord;
    toolUserCoord.toolInEndPosition.x = 0;
    toolUserCoord.toolInEndPosition.y = 0;
    toolUserCoord.toolInEndPosition.z = 0.45;
    toolUserCoord.toolInEndOrientation.w = 1;
    toolUserCoord.toolInEndOrientation.x = 0;
    toolUserCoord.toolInEndOrientation.y = 0;
    toolUserCoord.toolInEndOrientation.z = 0;
    userCoord.toolDesc = toolUserCoord;

    aubo_robot_namespace::ToolInEndDesc toolInEndDesc;
```

```
    toolInEndDesc.toolInEndPosition.x = 0;
    toolInEndDesc.toolInEndPosition.y = 0;
    toolInEndDesc.toolInEndPosition.z = 0;
    toolInEndDesc.toolInEndOrientation.w = 1;
    toolInEndDesc.toolInEndOrientation.x = 0;
    toolInEndDesc.toolInEndOrientation.y = 0;
    toolInEndDesc.toolInEndOrientation.z = 0;

    aubo_robot_namespace::Pos flangeCenterPosOnUser;
    aubo_robot_namespace::Ori flangeCenterOriOnUser;

    //将法兰盘中心在基坐标系下的位置和姿态转成法兰盘中心在用户坐标系下的位置和姿态
    rs_base_to_user(rshd, &flangeCenterPosOnBase, &flangeCenterOriOnBase, &userCoord, &toolInEndDesc, &flangeCenterPosOnUser, &flangeCenterOriOnUser);

    std::cout << "法兰盘中心在用户坐标系下的位置: ";
    std::cout << "(" << flangeCenterPosOnUser.x << ", " << flangeCenterPosOnUser.y << ", " << flangeCenterPosOnUser.z << ")";
    std::cout << std::endl;

    std::cout << "法兰盘中心在用户坐标系下的姿态（四元数）: ";
    std::cout << "(" << flangeCenterOriOnUser.w << ", " << flangeCenterOriOnUser.x << ", " << flangeCenterOriOnUser.y << ", " << flangeCenterOriOnUser.z << ")";
    std::cout << std::endl;

    aubo_robot_namespace::Rpy flangeCenterRpyOnUser;
    rs_quaternion_to_rpy(rshd, &flangeCenterOriOnUser, &flangeCenterRpyOnUser);
    std::cout << "法兰盘中心在用户坐标系下的姿态（欧拉角）: ";
    std::cout << "(" << flangeCenterRpyOnUser.rx * 180 / M_PI << ", " << flangeCenterRpyOnUser.ry * 180 / M_PI << ", " << flangeCenterRpyOnUser.rz * 180 / M_PI << ")";
    std::cout << std::endl;
}
```

main.cpp 代码如下:

注意：要将下面代码中的 IP 地址（ROBOT_ADDR）修改为对应服务器的 IP 地址。

```
#include "example.h"
#include "windows.h"
```

```
#define ROBOT_ADDR "192.168.1.100"
#define ROBOT_PORT 8899

//机械臂控制上下文句柄
RSHD g_rshd = -1;

int main(int argc, char* argv[])
{
    //登录服务器
    if (example_login(g_rshd, ROBOT_ADDR, ROBOT_PORT))
    {
        //机械臂上电(必须连接真实机械臂)
        example_robotStartup(g_rshd);

        //示例 2：将法兰盘中心在基坐标系下的位置和姿态转成法兰盘中心在用户坐标系下的位置和姿态
        example_base_to_user2(g_rshd);

    }
    else
    {
        std::cout << "机械臂登录失败。" << std::endl;
    }

    //反初始化接口库
    rs_uninitialize();

    std::cout << "Please enter to exit" << std::endl;
    getchar();

    return 0;
}
```

rs_base_to_user 函数示例 3

本示例是将法兰盘中心在基坐标系下的位置和姿态转成工具在基坐标系下的位置和姿态。

example.h 代码如下：

```
#pragma once
#include <string>
#include "rsdef.h"

//示例 3：将法兰盘中心在基坐标系下的位置和姿态转成工具在基坐标系下的位置和姿态
void example_base_to_user3(RSHD rshd);
```

example.cpp 代码如下：

```
#include "example.h"
#include "rsdef.h"

#include <Windows.h>
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <algorithm>



#define M_PI 3.14159265358979323846

//示例 3：将法兰盘中心在基坐标系下的位置和姿态转成工具在基坐标系下的位置和姿态
void example_base_to_user3(RSHD rshd)
{
    aubo_robot_namespace::Pos flangeCenterPosOnBase;//法兰盘中心在基坐标系下的位置
    flangeCenterPosOnBase.x = 0.420750;
    flangeCenterPosOnBase.y = 0.072954;
    flangeCenterPosOnBase.z = 0.595753;

    aubo_robot_namespace::Rpy flangeCenterRpyOnBase;//法兰盘中心在基坐标系下的姿态（欧拉角）
    flangeCenterRpyOnBase.rx = 148.947708 * M_PI / 180;
    flangeCenterRpyOnBase.ry = 14.759964 * M_PI / 180;
```

```
    flangeCenterRpyOnBase.rz = 107.579117 * M_PI / 180;

    aubo_robot_namespace::Ori flangeCenterOriOnBase;//法兰盘中心在基坐标系下的姿态（四元数）
    rs_rpy_to_quaternion(rshd, &flangeCenterRpyOnBase, &flangeCenterOriOnBase); //欧拉角转四元数

    aubo_robot_namespace::CoordCalibrateByJointAngleAndTool baseCoord;//基坐标系
    baseCoord.coordType = aubo_robot_namespace::BaseCoordinate;

    aubo_robot_namespace::ToolInEndDesc toolInEndDesc;//工具参数
    toolInEndDesc.toolInEndPosition.x = 0;
    toolInEndDesc.toolInEndPosition.y = 0;
    toolInEndDesc.toolInEndPosition.z = 0.45;
    toolInEndDesc.toolInEndOrientation.w = 1;
    toolInEndDesc.toolInEndOrientation.x = 0;
    toolInEndDesc.toolInEndOrientation.y = 0;
    toolInEndDesc.toolInEndOrientation.z = 0;

    aubo_robot_namespace::Pos toolEndPosOnBase;//工具末端在基坐标系下的位置
    aubo_robot_namespace::Ori toolEndOriOnBase;//工具末端在基坐标系下的姿态

    //将法兰盘中心在基坐标系下的位置和姿态转成工具在基坐标系下的位置和姿态
    rs_base_to_user(rshd, &flangeCenterPosOnBase, &flangeCenterOriOnBase, &baseCoord, &toolInEndDesc, &toolEndPosOnBase, &toolEndOriOnBase);

    std::cout << "工具末端在基坐标系下的位置: ";
    std::cout << "(" << toolEndPosOnBase.x << ", " << toolEndPosOnBase.y << ", " << toolEndPosOnBase.z << ")";
    std::cout << std::endl;

    std::cout << "工具末端在基坐标系下的姿态（四元数）: ";
    std::cout << "(" << toolEndOriOnBase.w << ", " << toolEndOriOnBase.x << ", " << toolEndOriOnBase.y << ", " << toolEndOriOnBase.z << ")";
    std::cout << std::endl;

    aubo_robot_namespace::Rpy toolEndRpyOnBase;
    rs_quaternion_to_rpy(rshd, &toolEndOriOnBase, &toolEndRpyOnBase);
    std::cout << "工具末端在基坐标系下的姿态（欧拉角）: ";
    std::cout << "(" << toolEndRpyOnBase.rx * 180 / M_PI << ", " << toolEndRpyOnBase.ry * 180 / M_PI << ", " << toolEndRpyOnBase.rz * 180 / M_PI << ")";
```

```
    std::cout << std::endl;

}
```

main.cpp 代码如下:

注意：要将下面代码中的 IP 地址（ROBOT_ADDR）修改为对应服务器的 IP 地址。

```
#include "example.h"
#include "windows.h"

#define ROBOT_ADDR "192.168.1.100"
#define ROBOT_PORT 8899

//机械臂控制上下文句柄
RSHD g_rshd = -1;

int main(int argc, char* argv[])
{
    //登录服务器
    if (example_login(g_rshd, ROBOT_ADDR, ROBOT_PORT))
    {
        //机械臂上电(必须连接真实机械臂)
        example_robotStartup(g_rshd);

        //示例 3：将法兰盘中心在基坐标系下的位置和姿态转成工具在基坐标系下的位置和姿态
        example_base_to_user3(g_rshd);
    }
    else
    {
        std::cout << "机械臂登录失败。" << std::endl;
    }

    //反初始化接口库
    rs_uninitialize();

    std::cout << "Please enter to exit" << std::endl;
    getchar();

    return 0;
}
```

4.4.2 基坐标系转基坐标得到工具末端点的位置姿态 rs_base_to_base_additional_tool()

rs_base_to_base_additional_tool 函数示例

本示例是将法兰盘中心在基坐标系下的位置和姿态转成工具在基坐标系下的位置和姿态。

example.h 代码如下:

```
#pragma once
#include <string>
#include "rsdef.h"

//基坐标系转基坐标得到工具末端点的位置和姿态 rs_base_to_base_additional_tool
//示例：将法兰盘中心在基坐标系下的位置和姿态转成工具在基坐标系下的位置和姿态
void example_base_to_base_additional_tool(RSHD rshd);
```

example.cpp 代码如下:

```
#include "example.h"
#include "rsdef.h"

#include <Windows.h>
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <algorithm>



#define M_PI 3.14159265358979323846

//基坐标系转基坐标得到工具末端点的位置和姿态 rs_base_to_base_additional_tool
//示例：将法兰盘中心在基坐标系下的位置和姿态转成工具在基坐标系下的位置和姿态
void example_base_to_base_additional_tool(RSHD rshd)
{
    aubo_robot_namespace::Pos flangeCenterPosOnBase;//法兰盘中心在基坐标系下的位置
    flangeCenterPosOnBase.x = 0.420750;
    flangeCenterPosOnBase.y = 0.072954;
```

```
    flangeCenterPosOnBase.z = 0.595753;

    aubo_robot_namespace::Rpy flangeCenterRpyOnBase;//法兰盘中心在基坐标系下的姿态（欧拉角）
    flangeCenterRpyOnBase.rx = 148.947708 * M_PI / 180;
    flangeCenterRpyOnBase.ry = 14.759964 * M_PI / 180;
    flangeCenterRpyOnBase.rz = 107.579117 * M_PI / 180;

    aubo_robot_namespace::Ori flangeCenterOriOnBase;//法兰盘中心在基坐标系下的姿态（四元数）
    rs_rpy_to_quaternion(rshd, &flangeCenterRpyOnBase, &flangeCenterOriOnBase);

    aubo_robot_namespace::ToolInEndDesc toolInEndDesc;//工具参数
    toolInEndDesc.toolInEndPosition.x = 0;
    toolInEndDesc.toolInEndPosition.y = 0;
    toolInEndDesc.toolInEndPosition.z = 0.45;
    toolInEndDesc.toolInEndOrientation.w = 1;
    toolInEndDesc.toolInEndOrientation.x = 0;
    toolInEndDesc.toolInEndOrientation.y = 0;
    toolInEndDesc.toolInEndOrientation.z = 0;

    aubo_robot_namespace::Pos toolEndPosOnBase;//工具末端在基坐标系下的位置
    aubo_robot_namespace::Ori toolEndOriOnBase;//工具末端在基坐标系下的姿态

    //将法兰盘中心在基坐标系下的位置和姿态转成工具在基坐标系下的位置和姿态
    rs_base_to_base_additional_tool(rshd, &flangeCenterPosOnBase, &flangeCenterOriOnBase, &toolInEndDesc, &toolEndPosOnBase, &toolEndOriOnBase);

    std::cout << "工具末端在基坐标系下的位置: ";
    std::cout << "(" << toolEndPosOnBase.x << ", " << toolEndPosOnBase.y << ", " << toolEndPosOnBase.z << ")";
    std::cout << std::endl;

    std::cout << "工具末端在基坐标系下的姿态（四元数）: ";
    std::cout << "(" << toolEndOriOnBase.w << ", " << toolEndOriOnBase.x << ", " << toolEndOriOnBase.y << ", " << toolEndOriOnBase.z << ")";
    std::cout << std::endl;

    aubo_robot_namespace::Rpy toolEndRpyOnBase;
    rs_quaternion_to_rpy(rshd, &toolEndOriOnBase, &toolEndRpyOnBase);
    std::cout << "工具末端在基坐标系下的姿态（欧拉角）: ";
    std::cout << "(" << toolEndRpyOnBase.rx * 180 / M_PI << ", " << toolEndRpyOnBase.ry * 180 / M_PI << ", " << toolEndRpyOnBase.rz * 180 / M_PI <
```

```
< ")";
    std::cout << std::endl;
}
```

main.cpp 代码如下:

注意：要将下面代码中的 IP 地址（ROBOT_ADDR）修改为对应服务器的 IP 地址。

```
#include "example.h"
#include "windows.h"

#define ROBOT_ADDR "192.168.1.100"
#define ROBOT_PORT 8899

//机械臂控制上下文句柄
RSHD g_rshd = -1;

int main(int argc, char* argv[])
{
    //登录服务器
    if (example_login(g_rshd, ROBOT_ADDR, ROBOT_PORT))
    {
        //机械臂上电(必须连接真实机械臂)
        example_robotStartup(g_rshd);


        //基坐标系转基坐标得到工具末端点的位置和姿态 rs_base_to_base_additio
nal_tool
        //示例：将法兰盘中心在基坐标系下的位置和姿态转成工具在基坐标系下
的位置和姿态
        example_base_to_base_additional_tool(g_rshd);
    }
    else
    {
        std::cout << "机械臂登录失败。" << std::endl;
    }

    //反初始化接口库
    rs_uninitialize();

    std::cout << "Please enter to exit" << std::endl;
    getchar();
```

```
    return 0;
}
```

4.4.3 用户坐标系转基坐标系 rs_user_to_base()

rs_user_to_base 函数示例 1

本示例是将工具末端在用户坐标系下的位置和姿态转成法兰盘中心在基坐标系下的位置和姿态。

example.h 代码如下:

```
#pragma once
#include <string>
#include "rsdef.h"

//用户坐标系的位置和姿态信息转基坐标系下的位置和姿态信息 rs_user_to_base
//示例 1：将工具末端在用户坐标系下的位置和姿态转成法兰盘中心在基坐标系下的位置和姿态
void example_user_to_base1(RSHD rshd);
```

example.cpp 代码如下:

```
#include "example.h"
#include "rsdef.h"

#include <Windows.h>
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <algorithm>



#define M_PI 3.14159265358979323846

//用户坐标系的位置和姿态信息转基坐标系下的位置和姿态信息 rs_user_to_base
//示例 1：将工具末端在用户坐标系下的位置和姿态转成法兰盘中心在基坐标系下的位置和姿态
void example_user_to_base1(RSHD rshd)
{
```

```
    aubo_robot_namespace::Pos toolEndPosOnUser;//工具末端在用户坐标系下的位
置
    toolEndPosOnUser.x = 0.972055;
    toolEndPosOnUser.y = -0.508566;
    toolEndPosOnUser.z = 0.583447;

    aubo_robot_namespace::Ori toolEndOriOnUser;//工具末端在用户坐标系下的姿
态（四元数）
    aubo_robot_namespace::Rpy toolEndRpyOnUser;//工具末端在用户坐标系下的姿
态（欧拉角）
    toolEndRpyOnUser.rx = 148.947357 * M_PI / 180;
    toolEndRpyOnUser.ry = 14.759711 * M_PI / 180;
    toolEndRpyOnUser.rz = 107.579247 * M_PI / 180;
    rs_rpy_to_quaternion(rshd, &toolEndRpyOnUser, &toolEndOriOnUser);

    aubo_robot_namespace::CoordCalibrateByJointAngleAndTool userCoord;//用户坐
标系
    userCoord.coordType = aubo_robot_namespace::WorldCoordinate;
    userCoord.methods = aubo_robot_namespace::Origin_AnyPointOnPositiveXAxis_
AnyPointOnFirstQuadrantOfXOYPlane;

    userCoord.wayPointArray[0].jointPos[0] = -75.093279 * M_PI / 180;
    userCoord.wayPointArray[0].jointPos[1] = 28.544643 * M_PI / 180;
    userCoord.wayPointArray[0].jointPos[2] = -114.313905 * M_PI / 180;
    userCoord.wayPointArray[0].jointPos[3] = -62.769247 * M_PI / 180;
    userCoord.wayPointArray[0].jointPos[4] = -87.343517 * M_PI / 180;
    userCoord.wayPointArray[0].jointPos[5] = -27.888262 * M_PI / 180;

    userCoord.wayPointArray[1].jointPos[0] = -89.239837 * M_PI / 180;
    userCoord.wayPointArray[1].jointPos[1] = 23.936171 * M_PI / 180;
    userCoord.wayPointArray[1].jointPos[2] = -122.299277 * M_PI / 180;
    userCoord.wayPointArray[1].jointPos[3] = -65.208902 * M_PI / 180;
    userCoord.wayPointArray[1].jointPos[4] = -85.011123 * M_PI / 180;
    userCoord.wayPointArray[1].jointPos[5] = -41.87417 * M_PI / 180;

    userCoord.wayPointArray[2].jointPos[0] = -77.059212 * M_PI / 180;
    userCoord.wayPointArray[2].jointPos[1] = 35.509518 * M_PI / 180;
    userCoord.wayPointArray[2].jointPos[2] = -101.108547 * M_PI / 180;
    userCoord.wayPointArray[2].jointPos[3] = -56.433133 * M_PI / 180;
    userCoord.wayPointArray[2].jointPos[4] = -87.006734 * M_PI / 180;
    userCoord.wayPointArray[2].jointPos[5] = -29.827440 * M_PI / 180;

    aubo_robot_namespace::ToolInEndDesc toolUserCoord;
    toolUserCoord.toolInEndPosition.x = 0;
```

```
    toolUserCoord.toolInEndPosition.y = 0;
    toolUserCoord.toolInEndPosition.z = 0.45;
    toolUserCoord.toolInEndOrientation.w = 1;
    toolUserCoord.toolInEndOrientation.x = 0;
    toolUserCoord.toolInEndOrientation.y = 0;
    toolUserCoord.toolInEndOrientation.z = 0;
    userCoord.toolDesc = toolUserCoord;

    aubo_robot_namespace::ToolInEndDesc toolInEndDesc;
    toolInEndDesc.toolInEndPosition.x = 0;
    toolInEndDesc.toolInEndPosition.y = 0;
    toolInEndDesc.toolInEndPosition.z = 0.45;
    toolInEndDesc.toolInEndOrientation.w = 1;
    toolInEndDesc.toolInEndOrientation.x = 0;
    toolInEndDesc.toolInEndOrientation.y = 0;
    toolInEndDesc.toolInEndOrientation.z = 0;

    aubo_robot_namespace::Pos flangeCenterPosOnBase;//法兰盘中心在基坐标系下
的位置
    aubo_robot_namespace::Ori flangeCenterOriOnBase;//法兰盘中心在基坐标系下
的姿态（四元数）

    //将工具末端在用户坐标系下的位置和姿态转成法兰盘中心在基坐标系下的位
置和姿态
    rs_user_to_base(rshd, &toolEndPosOnUser, &toolEndOriOnUser, &userCoord,
&toolInEndDesc, &flangeCenterPosOnBase, &flangeCenterOriOnBase);

    std::cout << "法兰盘中心在基坐标系下的位置: ";
    std::cout << "(" << flangeCenterPosOnBase.x << ", " << flangeCenterPosOnB
ase.y << ", " << flangeCenterPosOnBase.z << ")";
    std::cout << std::endl;

    std::cout << "法兰盘中心在基坐标系下的姿态（四元数）: ";
    std::cout << "(" << flangeCenterOriOnBase.w << ", " << flangeCenterOriOnB
ase.x << ", " << flangeCenterOriOnBase.y << ", " << flangeCenterOriOnBase.z
<< ")";
    std::cout << std::endl;

    aubo_robot_namespace::Rpy flangeCenterRpyOnBase;//法兰盘中心在基坐标系下
的姿态（欧拉角）
    rs_quaternion_to_rpy(rshd, &flangeCenterOriOnBase, &flangeCenterRpyOnBase);
    std::cout << "法兰盘中心在基坐标系下的姿态（欧拉角）: ";
    std::cout << "(" << flangeCenterRpyOnBase.rx * 180 / M_PI << ", " << flan
geCenterRpyOnBase.ry * 180 / M_PI << ", " << flangeCenterRpyOnBase.rz * 18
```

```
0 / M_PI << ")";
    std::cout << std::endl;

}
```

main.cpp 代码如下:

注意：要将下面代码中的 IP 地址（ROBOT_ADDR）修改为对应服务器的 IP 地址。

```
#include "example.h"
#include "windows.h"

#define ROBOT_ADDR "192.168.1.100"
#define ROBOT_PORT 8899

//机械臂控制上下文句柄
RSHD g_rshd = -1;

int main(int argc, char* argv[])
{
    //登录服务器
    if (example_login(g_rshd, ROBOT_ADDR, ROBOT_PORT))
    {
        //机械臂上电(必须连接真实机械臂)
        example_robotStartup(g_rshd);

        //用户坐标系的位置和姿态信息转基坐标系下的位置和姿态信息 rs_user_to_base
        //示例 1：将工具末端在用户坐标系下的位置和姿态转成法兰盘中心在基坐标系下的位置和姿态
        example_user_to_base1(g_rshd);
    }
    else
    {
        std::cout << "机械臂登录失败。" << std::endl;
    }

    //反初始化接口库
    rs_uninitialize();

    std::cout << "Please enter to exit" << std::endl;
    getchar();

    return 0;
}
```

rs_user_to_base 函数示例 2

本示例是将法兰盘中心在用户坐标系下的位置和姿态转成法兰盘中心在基坐标系下的位置和姿态。

example.h 代码如下：

```
#pragma once
#include <string>
#include "rsdef.h"

//示例 2：将法兰盘中心在用户坐标系下的位置和姿态转成法兰盘中心在基坐标系下的位置和姿态
void example_user_to_base2(RSHD rshd);
```

example.cpp 代码如下：

```
#include "example.h"
#include "rsdef.h"

#include <Windows.h>
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <algorithm>



#define M_PI 3.14159265358979323846

//示例 2：将法兰盘中心在用户坐标系下的位置和姿态转成法兰盘中心在基坐标系下的位置和姿态
void example_user_to_base2(RSHD rshd)
{
    aubo_robot_namespace::Pos flangeCenterPosOnUser;//法兰盘中心在用户坐标系下的位置
    flangeCenterPosOnUser.x = 0.721111;
    flangeCenterPosOnUser.y = -0.485044;
    flangeCenterPosOnUser.z = 0.956239;

    aubo_robot_namespace::Ori flangeCenterOriOnUser;//法兰盘中心在用户坐标系下的姿态（四元数）
    aubo_robot_namespace::Rpy flangeCenterRpyOnUser;//法兰盘中心在用户坐标系下的姿态（欧拉角）
```

```
    flangeCenterRpyOnUser.rx = 148.947357 * M_PI / 180;
    flangeCenterRpyOnUser.ry = 14.759711 * M_PI / 180;
    flangeCenterRpyOnUser.rz = 107.579247 * M_PI / 180;
    rs_rpy_to_quaternion(rshd, &flangeCenterRpyOnUser, &flangeCenterOriOnUser);

    aubo_robot_namespace::CoordCalibrateByJointAngleAndTool userCoord;//用户坐
标系
    userCoord.coordType = aubo_robot_namespace::WorldCoordinate;
    userCoord.methods = aubo_robot_namespace::Origin_AnyPointOnPositiveXAxis_
AnyPointOnFirstQuadrantOfXOYPlane;

    userCoord.wayPointArray[0].jointPos[0] = -75.093279 * M_PI / 180;
    userCoord.wayPointArray[0].jointPos[1] = 28.544643 * M_PI / 180;
    userCoord.wayPointArray[0].jointPos[2] = -114.313905 * M_PI / 180;
    userCoord.wayPointArray[0].jointPos[3] = -62.769247 * M_PI / 180;
    userCoord.wayPointArray[0].jointPos[4] = -87.343517 * M_PI / 180;
    userCoord.wayPointArray[0].jointPos[5] = -27.888262 * M_PI / 180;

    userCoord.wayPointArray[1].jointPos[0] = -89.239837 * M_PI / 180;
    userCoord.wayPointArray[1].jointPos[1] = 23.936171 * M_PI / 180;
    userCoord.wayPointArray[1].jointPos[2] = -122.299277 * M_PI / 180;
    userCoord.wayPointArray[1].jointPos[3] = -65.208902 * M_PI / 180;
    userCoord.wayPointArray[1].jointPos[4] = -85.011123 * M_PI / 180;
    userCoord.wayPointArray[1].jointPos[5] = -41.87417 * M_PI / 180;

    userCoord.wayPointArray[2].jointPos[0] = -77.059212 * M_PI / 180;
    userCoord.wayPointArray[2].jointPos[1] = 35.509518 * M_PI / 180;
    userCoord.wayPointArray[2].jointPos[2] = -101.108547 * M_PI / 180;
    userCoord.wayPointArray[2].jointPos[3] = -56.433133 * M_PI / 180;
    userCoord.wayPointArray[2].jointPos[4] = -87.006734 * M_PI / 180;
    userCoord.wayPointArray[2].jointPos[5] = -29.827440 * M_PI / 180;

    aubo_robot_namespace::ToolInEndDesc toolUserCoord;
    toolUserCoord.toolInEndPosition.x = 0;
    toolUserCoord.toolInEndPosition.y = 0;
    toolUserCoord.toolInEndPosition.z = 0.45;
    toolUserCoord.toolInEndOrientation.w = 1;
    toolUserCoord.toolInEndOrientation.x = 0;
    toolUserCoord.toolInEndOrientation.y = 0;
    toolUserCoord.toolInEndOrientation.z = 0;
    userCoord.toolDesc = toolUserCoord;

    aubo_robot_namespace::ToolInEndDesc toolInEndDesc;
    toolInEndDesc.toolInEndPosition.x = 0;
```

```
    toolInEndDesc.toolInEndPosition.y = 0;
    toolInEndDesc.toolInEndPosition.z = 0;
    toolInEndDesc.toolInEndOrientation.w = 1;
    toolInEndDesc.toolInEndOrientation.x = 0;
    toolInEndDesc.toolInEndOrientation.y = 0;
    toolInEndDesc.toolInEndOrientation.z = 0;

    aubo_robot_namespace::Pos flangeCenterPosOnBase;//法兰盘中心在基坐标系下的位置
    aubo_robot_namespace::Ori flangeCenterOriOnBase;//法兰盘中心在基坐标系下的姿态（四元数）

    //将法兰盘中心在用户坐标系下的位置和姿态转成法兰盘中心在基坐标系下的位置和姿态
    rs_user_to_base(rshd, &flangeCenterPosOnUser, &flangeCenterOriOnUser, &userCoord, &toolInEndDesc, &flangeCenterPosOnBase, &flangeCenterOriOnBase);

    std::cout << "法兰盘中心在基坐标系下的位置: ";
    std::cout << "(" << flangeCenterPosOnBase.x << ", " << flangeCenterPosOnBase.y << ", " << flangeCenterPosOnBase.z << ")";
    std::cout << std::endl;

    std::cout << "法兰盘中心在基坐标系下的姿态（四元数）: ";
    std::cout << "(" << flangeCenterOriOnBase.w << ", " << flangeCenterOriOnBase.x << ", " << flangeCenterOriOnBase.y << ", " << flangeCenterOriOnBase.z << ")";
    std::cout << std::endl;

    aubo_robot_namespace::Rpy flangeCenterRpyOnBase;//法兰盘中心在基坐标系下的姿态（欧拉角）
    rs_quaternion_to_rpy(rshd, &flangeCenterOriOnBase, &flangeCenterRpyOnBase);
    std::cout << "法兰盘中心在基坐标系下的姿态（欧拉角）: ";
    std::cout << "(" << flangeCenterRpyOnBase.rx * 180 / M_PI << ", " << flangeCenterRpyOnBase.ry * 180 / M_PI << ", " << flangeCenterRpyOnBase.rz * 180 / M_PI << ")";
    std::cout << std::endl;

}
```

main.cpp 代码如下:

注意：要将下面代码中的 IP 地址（ROBOT_ADDR）修改为对应服务器的 IP 地址。


```
#include "example.h"
#include "windows.h"

#define ROBOT_ADDR "192.168.1.100"
#define ROBOT_PORT 8899

//机械臂控制上下文句柄
RSHD g_rshd = -1;

int main(int argc, char* argv[])
{
    //登录服务器
    if (example_login(g_rshd, ROBOT_ADDR, ROBOT_PORT))
    {
        //机械臂上电(必须连接真实机械臂)
        example_robotStartup(g_rshd);

        //示例 2：将法兰盘中心在用户坐标系下的位置和姿态转成法兰盘中心在基坐标系下的位置和姿态
        example_user_to_base2(g_rshd);
    }
    else
    {
        std::cout << "机械臂登录失败。" << std::endl;
    }

    //反初始化接口库
    rs_uninitialize();

    std::cout << "Please enter to exit" << std::endl;
    getchar();

    return 0;
}
```

rs_user_to_base 函数示例 3

本示例是将工具末端在基坐标系下的位置和姿态转成法兰盘中心在基坐标系下的位置和姿态。

example.h 代码如下:

```
#pragma once
#include <string>
#include "rsdef.h"

//示例 3：将工具末端在基坐标系下的位置和姿态转成法兰盘中心在基坐标系下的位置和姿态
void example_user_to_base3(RSHD rshd);
```

example.cpp 代码如下:

```
#include "example.h"
#include "rsdef.h"

#include <Windows.h>
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <algorithm>


#define M_PI 3.14159265358979323846

//示例 3：将工具末端在基坐标系下的位置和姿态转成法兰盘中心在基坐标系下的位置和姿态
void example_user_to_base3(RSHD rshd)
{
    aubo_robot_namespace::Pos toolEndPosOnBase;//工具末端在基坐标系下的位置
    toolEndPosOnBase.x = 0.671693;
    toolEndPosOnBase.y = 0.049429;
    toolEndPosOnBase.z = 0.222961;

    aubo_robot_namespace::Ori toolEndOriOnBase;//工具末端在基坐标系下的姿态（四元数）
    aubo_robot_namespace::Rpy toolEndRpyOnBase;//工具末端在基坐标系下的姿态（欧拉角）
```

```
    toolEndRpyOnBase.rx = 148.947708 * M_PI / 180;
    toolEndRpyOnBase.ry = 14.759964 * M_PI / 180;
    toolEndRpyOnBase.rz = 107.579117 * M_PI / 180;
    rs_rpy_to_quaternion(rshd, &toolEndRpyOnBase, &toolEndOriOnBase);

    aubo_robot_namespace::CoordCalibrateByJointAngleAndTool baseCoord;
    baseCoord.coordType = aubo_robot_namespace::BaseCoordinate;

    aubo_robot_namespace::ToolInEndDesc toolInEndDesc;
    toolInEndDesc.toolInEndPosition.x = 0;
    toolInEndDesc.toolInEndPosition.y = 0;
    toolInEndDesc.toolInEndPosition.z = 0.45;
    toolInEndDesc.toolInEndOrientation.w = 1;
    toolInEndDesc.toolInEndOrientation.x = 0;
    toolInEndDesc.toolInEndOrientation.y = 0;
    toolInEndDesc.toolInEndOrientation.z = 0;

    aubo_robot_namespace::Pos flangeCenterPosOnBase;//法兰盘中心在基坐标系下
的位置
    aubo_robot_namespace::Ori flangeCenterOriOnBase;//法兰盘中心在基坐标系下
的姿态（四元数）

    //将工具末端在基坐标系下的位置和姿态转成法兰盘中心在基坐标系下的位置
和姿态
    rs_user_to_base(rshd, &toolEndPosOnBase, &toolEndOriOnBase, &baseCoord,
&toolInEndDesc, &flangeCenterPosOnBase, &flangeCenterOriOnBase);

    std::cout << "法兰盘中心在基坐标系下的位置: ";
    std::cout << "(" << flangeCenterPosOnBase.x << ", " << flangeCenterPosOnB
ase.y << ", " << flangeCenterPosOnBase.z << ")";
    std::cout << std::endl;

    std::cout << "法兰盘中心在基坐标系下的姿态（四元数）: ";
    std::cout << "(" << flangeCenterOriOnBase.w << ", " << flangeCenterOriOnB
ase.x << ", " << flangeCenterOriOnBase.y << ", " << flangeCenterOriOnBase.z
<< ")";
    std::cout << std::endl;

    aubo_robot_namespace::Rpy flangeCenterRpyOnBase;//法兰盘中心在基坐标系下
的姿态（欧拉角）
    rs_quaternion_to_rpy(rshd, &flangeCenterOriOnBase, &flangeCenterRpyOnBase);
    std::cout << "法兰盘中心在基坐标系下的姿态（欧拉角）: ";
    std::cout << "(" << flangeCenterRpyOnBase.rx * 180 / M_PI << ", " << flan
geCenterRpyOnBase.ry * 180 / M_PI << ", " << flangeCenterRpyOnBase.rz * 18
```

```
0 / M_PI << ")";
    std::cout << std::endl;

}
```

main.cpp 代码如下：

注意：要将下面代码中的 IP 地址（ROBOT_ADDR）修改为对应服务器的 IP 地址。

```
#include "example.h"
#include "windows.h"

#define ROBOT_ADDR "192.168.1.100"
#define ROBOT_PORT 8899

//机械臂控制上下文句柄
RSHD g_rshd = -1;

int main(int argc, char* argv[])
{
    //登录服务器
    if (example_login(g_rshd, ROBOT_ADDR, ROBOT_PORT))
    {
        //机械臂上电(必须连接真实机械臂)
        example_robotStartup(g_rshd);

        //示例 3：将工具末端在基坐标系下的位置和姿态转成法兰盘中心在基坐标系下的位置和姿态
        example_user_to_base3(g_rshd);
    }
    else
    {
        std::cout << "机械臂登录失败。" << std::endl;
    }

    //反初始化接口库
    rs_uninitialize();

    std::cout << "Please enter to exit" << std::endl;
    getchar();

    return 0;
}
```

4.5 设置和获取机械臂相关参数

本示例是设置和获取机械臂相关的参数。

example.h 代码如下:

```
#pragma once
#include <string>
#include "rsdef.h"

//设置和获取机械臂相关参数
void example_robot_param(RSHD rshd);
```

example.cpp 代码如下:

```
#include "example.h"
#include "rsdef.h"

#include <Windows.h>
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <algorithm>



#define M_PI 3.14159265358979323846

//设置和获取机械臂相关参数
void example_robot_param(RSHD rshd)
{
    int ret;

    //1.获取机械臂关节状态
    aubo_robot_namespace::JointStatus jointStatus[6];
    ret = rs_get_joint_status(rshd, jointStatus);
    std::cout << std::endl;
    if (ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
        std::cout << "获取关节状态成功." << std::endl;

        if (ret == RS_SUCC)
        {
            std::cout << "获取关节状态成功。" << std::endl;
```

```
            std::cout << "----------关节状态信息----------" << std::endl;

            for (int i = 0; i < 6; i++)
            {
                std::cout << "关节 ID：" << i + 1 << "   ";
                std::cout << "关节电流：" << jointStatus[i].jointCurrentI << "
";
                std::cout << "关节速度：" << jointStatus[i].jointSpeedMoto <<
"   ";
                std::cout << "关节角：" << jointStatus[i].jointPosJ * 180 / M_
PI << "   ";
                std::cout << "关节电压：" << jointStatus[i].jointCurVol << "
";
                std::cout << "当前温度：" << jointStatus[i].jointCurTemp << "
 ";
                std::cout << "电机目标电流：" << jointStatus[i].jointTagCurrentI
 << "   ";
                std::cout << "电机目标速度：" << jointStatus[i].jointTagSpeedM
oto << "   ";
                std::cout << "目标关节角：" << jointStatus[i].jointTagPosJ * 18
0 / M_PI << "   ";
                std::cout << "关节错误码：" << jointStatus[i].jointErrorNum <<
 std::endl;
            }

            std::cout << std::endl;

        }
        else
        {
            std::cerr << "获取关节状态失败。错误号为" << ret << std::endl;
        }

    }
    else
    {
        std::cerr << "获取关节状态失败." << std::endl;
    }

    //2.获取真实臂是否存在
    bool IsRealRobotExist = false;
    ret = rs_is_have_real_robot(rshd, &IsRealRobotExist);

    std::cout << std::endl;
```

```
    if (ret == RS_SUCC)
    {
        std::cout << "真实机械臂是否存在：" << IsRealRobotExist << std::endl;
    }
    else
    {
        std::cerr << "ERROR:获取机械臂真实臂是否存在失败。错误号：" << re
t << std::endl;
    }

    //3.机械臂诊断信息
    aubo_robot_namespace::RobotDiagnosis robotDiagnosis;
    ret = rs_get_diagnosis_info(rshd, &robotDiagnosis);

    std::cout << std::endl;
    if (ret == RS_SUCC)
    {
        std::cout << "获取诊断信息成功" << std::endl;
        std::cout << std::endl << "----------机械臂诊断信息-------" << std::endl;

        std::cout << std::endl << "    " << "CAN 通信状态:" << (int)robotDiagno
sis.armCanbusStatus;
        std::cout << std::endl << "    " << "电源当前电流:" << robotDiagnosis.ar
mPowerCurrent;
        std::cout << std::endl << "    " << "电源当前电压:" << robotDiagnosis.ar
mPowerVoltage;

        (robotDiagnosis.armPowerStatus) ? std::cout << std::endl << "    " << "4
8V 电源状态:开" : std::cout << std::endl << "    " << "48V 电源状态:关";

        std::cout << std::endl << "    " << "控制箱温度:" << (int)robotDiagnosis.
contorllerTemp;
        std::cout << std::endl << "    " << "控制箱湿度:" << (int)robotDiagnosis.
contorllerHumidity;
        std::cout << std::endl << "    " << "远程关机信号:" << robotDiagnosis.re
moteHalt;
        std::cout << std::endl << "    " << "机械臂软急停:" << robotDiagnosis.s
oftEmergency;
        std::cout << std::endl << "    " << "远程急停信号:" << robotDiagnosis.re
moteEmergency;
        std::cout << std::endl << "    " << "碰撞检测位:" << robotDiagnosis.robo
tCollision;
        std::cout << std::endl << "    " << "进入力控模式标志位:" << robotDiag
nosis.forceControlMode;
```

```
        std::cout << std::endl << "    " << "刹车状态:" << robotDiagnosis.brakeS
tuats;
        std::cout << std::endl << "    " << "末端速度:" << robotDiagnosis.robotE
ndSpeed;
        std::cout << std::endl << "    " << "最大加速度:" << robotDiagnosis.robo
tMaxAcc;
        std::cout << std::endl << "    " << "上位机软件状态位:" << robotDiagno
sis.orpeStatus;
        std::cout << std::endl << "    " << "位姿读取使能位:" << robotDiagnosi
s.enableReadPose;
        std::cout << std::endl << "    " << "安装位置状态:" << robotDiagnosis.ro
botMountingPoseChanged;
        std::cout << std::endl << "    " << "磁编码器错误状态:" << robotDiagno
sis.encoderErrorStatus;
        std::cout << std::endl << "    " << "静止碰撞检测开关:" << robotDiagno
sis.staticCollisionDetect;
        std::cout << std::endl << "    " << "关节碰撞检测:" << robotDiagnosis.jo
intCollisionDetect;
        std::cout << std::endl << "    " << "光电编码器不一致错误:" << robotDi
agnosis.encoderLinesError;
        std::cout << std::endl << "    " << "关节错误状态:" << robotDiagnosis.jo
intErrorStatus;
        std::cout << std::endl << "    " << "奇异点过速警告:" << robotDiagnosi
s.singularityOverSpeedAlarm;
        std::cout << std::endl << "    " << "电流错误警告:" << robotDiagnosis.ro
botCurrentAlarm;
        std::cout << std::endl << "    " << "tool error:" << (int)robotDiagnosis.to
olIoError;
        std::cout << std::endl << "    " << "安装位置错位:" << robotDiagnosis.ro
botMountingPoseWarning;
        std::cout << std::endl << "    " << "mac 缓冲器长度:" << robotDiagnosi
s.macTargetPosBufferSize;
        std::cout << std::endl << "    " << "mac 缓冲器有效数据长度:" << robot
Diagnosis.macTargetPosDataSize;
        std::cout << std::endl << "    " << "mac 数据中断:" << robotDiagnosis.m
acDataInterruptWarning;

        std::cout << std::endl << "---------------------------------end." << std::endl;

    }
    else
    {
        std::cerr << "获得机械臂诊断信息失败。错误码：" << ret << std::endl;
    }
```

```
    //4.获取登录状态
    bool connectStatus;
    ret = rs_get_login_status(rshd, &connectStatus);
    std::cout << std::endl;
    if (ret == RS_SUCC)
    {
        std::cout << "获取机械臂登录状态成功。" << std::endl;
        std::cout << "机械臂的登录状态： " << connectStatus << std::endl;
    }
    else
    {
        std::cerr << "获取登录状态失败。错误码：" << ret << std::endl;
    }


    //5.设置机械臂当前工作模式
    aubo_robot_namespace::RobotWorkMode workmode = aubo_robot_namespace::R
obotModeReal;
    ret = rs_set_work_mode(rshd, workmode);
    std::cout << std::endl;
    if (ret == RS_SUCC)
    {
        std::cout << "设置机械臂当前工作模式成功" << std::endl;
        std::cout << "设置机械臂当前工作模式: " << workmode << std::endl;
    }
    else
    {
        std::cerr << "设置机械臂当前工作模式失败。错误码：" << ret << std::e
ndl;
    }


    //6.获取机械臂当前工作模式
    aubo_robot_namespace::RobotWorkMode workmode2;
    ret = rs_get_work_mode(rshd, &workmode2);
    std::cout << std::endl;
    if (ret == RS_SUCC)
    {
        std::cout << "获取机械臂当前工作模式成功" << std::endl;
        std::cout << "获取机械臂当前工作模式: " << workmode2 << std::endl;
    }
    else
```

```
    {
        std::cerr << "获取机械臂当前工作模式失败。错误码：" << ret << std::endl;
    }

    //7.获取重力分量
    aubo_robot_namespace::RobotGravityComponent gravity;
    ret = rs_get_gravity_component(rshd, &gravity);
    std::cout << std::endl;
    if (ret == RS_SUCC)
    {
        std::cout << "获取重力分量成功" << std::endl;
        std::cout << "**** 重力分量 ****" << std::endl;
        std::cout << "x = " << gravity.x << std::endl;
        std::cout << "y = " << gravity.y << std::endl;
        std::cout << "z = " << gravity.z << std::endl;
    }
    else
    {
        std::cerr << "获取重力分量失败。错误码：" << ret << std::endl;
    }


    //8.设置机械臂碰撞等级
    int collisionClass = 5;
    ret = rs_set_collision_class(rshd, collisionClass);
    std::cout << std::endl;
    if (ret == RS_SUCC)
    {
        std::cout << "设置机械臂碰撞等级成功" << std::endl;
        std::cout << "设置机械臂碰撞等级: " << collisionClass << std::endl;
    }
    else
    {
        std::cerr << "设置机械臂碰撞等级失败。错误码：" << ret << std::endl;
    }


    //9.获取碰撞等级
    int collisiongrade;
    ret = rs_get_collision_class(rshd, &collisiongrade);
    std::cout << std::endl;
    if (ret == RS_SUCC)
    {
```

```
        std::cout << "获取碰撞等级成功" << std::endl;
        std::cout << "获取碰撞等级: " << collisiongrade << std::endl;
    }
    else
    {
        std::cerr << "获取碰撞等级失败。错误码：" << ret << std::endl;
    }


    //10.获取设备信息
    aubo_robot_namespace::RobotDevInfo robotdevinfo;
    ret = rs_get_device_info(rshd, &robotdevinfo);
    std::cout << std::endl;
    if (ret == RS_SUCC)
    {
        std::cout << "获取设备信息成功" << std::endl;
        std::cout << "**** 机械臂设备信息 ****" << std::endl;
        std::cout << "设备版本号 revision : " << robotdevinfo.revision << std::en
dl;
        std::cout << "从设备版本号 slave version : " << robotdevinfo.slave_versi
on << std::endl;
        std::cout << "IO 扩展版本号 extern IO version : " << robotdevinfo.extio
_version << std::endl;
        std::cout << "厂家 ID manu_id : " << robotdevinfo.manu_id << std::endl;
        std::cout << "机械臂类型 joint_type : " << robotdevinfo.joint_type << st
d::endl;
        for (int i = 0; i < 8; i++)
        {
            if (i == 6)
            {
                std::cout << "Tool" << " 硬件版本 hardware version is : " <<
robotdevinfo.joint_ver[i].hw_version << std::endl;
                std::cout << "Tool" << " 软件版本 software version is : " <<
robotdevinfo.joint_ver[i].sw_version << std::endl;
            }
            else if (i == 7)
            {
                std::cout << "Base" << " 硬件版本 hardware version is : " <<
robotdevinfo.joint_ver[i].hw_version << std::endl;
                std::cout << "Base" << " 软件版本 software version is : " <<
robotdevinfo.joint_ver[i].sw_version << std::endl;
            }
            else
            {
```

```
                std::cout << "关节 joint[" << i + 1 << "] 硬件版本 hardware version is : " << robotdevinfo.joint_ver[i].hw_version << std::endl;
                std::cout << "关节 joint[" << i + 1 << "] 软件版本 software version is : " << robotdevinfo.joint_ver[i].sw_version << std::endl;
            }
        }
        for (int i = 0; i < 8; i++)
        {
            if (i == 0)
            {
                std::cout << "Interface Board ID is : " << robotdevinfo.jointProductID[i].productID << std::endl;
            }
            else if (i == 7)
            {
                std::cout << "Tool ID is : " << robotdevinfo.jointProductID[i].productID << std::endl;
            }
            else
            {
                std::cout << "关节 joint[" << i << "] ID is : " << robotdevinfo.jointProductID[i].productID << std::endl;
            }
        }
    }
    else
    {
        std::cerr << "获取设备信息失败。错误码：" << ret << std::endl;
    }



    //11.获取机械臂当前运行状态
    aubo_robot_namespace::RobotState robotstate;
    ret = rs_get_robot_state(rshd, &robotstate);
    std::cout << std::endl;
    if (ret == RS_SUCC)
    {
        std::cout << "获取机械臂当前运行状态成功" << std::endl;
        std::cout << "机械臂当前运行状态: " << robotstate << std::endl;
    }
    else
    {
        std::cerr << "获取机械臂当前运行状态失败。错误码：" << ret << std::e
```

```
ndl;
    }

    //12.获取当前路点信息
    aubo_robot_namespace::wayPoint_S wayPoint;
    ret = rs_get_current_waypoint(rshd, &wayPoint);
    std::cout << std::endl;
    if (ret == RS_SUCC)
    {
        std::cout << "获取当前路点信息成功" << std::endl;
        std::cout << std::endl << "----------当前路点信息----------" << std::e
ndl;
        //位置信息
        std::cout << "位置 Pos: ";
        std::cout << "x:" << wayPoint.cartPos.position.x << " ";
        std::cout << "y:" << wayPoint.cartPos.position.y << " ";
        std::cout << "z:" << wayPoint.cartPos.position.z << std::endl;
        //姿态信息
        std::cout << "姿态 Ori: ";
        std::cout << "w:" << wayPoint.orientation.w << " ";
        std::cout << "x:" << wayPoint.orientation.x << " ";
        std::cout << "y:" << wayPoint.orientation.y << " ";
        std::cout << "z:" << wayPoint.orientation.z << std::endl;
        aubo_robot_namespace::Rpy tempRpy;
        rs_quaternion_to_rpy(rshd, &wayPoint.orientation, &tempRpy);
        std::cout << "欧拉角 Rpy: ";
        std::cout << "RX:" << tempRpy.rx * 180.0 / M_PI << " RY:" << temp
Rpy.ry * 180.0 / M_PI
            << " RZ:" << tempRpy.rz * 180.0 / M_PI << std::endl;
        //关节信息
        std::cout << "关节位置: " << std::endl;
        for (int i = 0; i < aubo_robot_namespace::ARM_DOF; i++)
        {
            std::cout << "关节" << i + 1 << ": " << wayPoint.jointpos[i] << "
~ " << wayPoint.jointpos[i] * 180.0 / M_PI << std::endl;
        }
    }
    else
    {
        std::cerr << "获取当前路点信息失败。错误码：" << ret << std::endl;
    }


    //13.是否在联机模式
```

```
    bool isonlinemode;
    ret = rs_is_online_mode(rshd, &isonlinemode);
    std::cout << std::endl;
    if (ret == RS_SUCC)
    {
        std::cout << "获取是否在联机模式 成功" << std::endl;
        std::cout << "机械臂是否在联机模式 : " << isonlinemode << std::endl;
    }
    else
    {
        std::cerr << "获取是否在联机模式 失败。错误码：" << ret << std::endl;
    }

    //14.是否在联机主模式
    bool isonlinemastermode;
    ret = rs_is_online_master_mode(rshd, &isonlinemastermode);
    std::cout << std::endl;
    if (ret == RS_SUCC)
    {
        std::cout << "是否在联机主模式成功" << std::endl;
        std::cout << "机械臂是否在联机主模式: " << isonlinemastermode << st
d::endl;
    }
    else
    {
        std::cerr << "是否在联机主模式失败。错误码：" << ret << std::endl;
    }


    //15.获取机械臂安全配置
    aubo_robot_namespace::RobotSafetyConfig safeconfig;
    ret = rs_get_robot_safety_config(rshd, &safeconfig);
    std::cout << std::endl;
    if (ret == RS_SUCC)
    {
        std::cout << "获取机械臂安全配置成功" << std::endl;
        std::cout << "缩减模式的关节速度限制：" << std::endl;
        for (int i = 0; i < 6; i++)
        {
            std::cout << "关节[" << i + 1 << "] 速度 = " << safeconfig.robotR
educedConfigJointSpeed[i] << std::endl;
        }
        std::cout << "缩减模式的 TCP 速度限制 = " << safeconfig.robotReduced
ConfigTcpSpeed << std::endl;
```

```
        std::cout << "缩减模式的 TCP 力 = " << safeconfig.robotReducedConfigT
cpForce << std::endl;
        std::cout << "缩减模式的动量 = " << safeconfig.robotReducedConfigMo
mentum << std::endl;
        std::cout << "缩减模式的功率 = " << safeconfig.robotReducedConfigPow
er << std::endl;
    }
    else
    {
        std::cerr << "获取机械臂安全配置失败。错误码：" << ret << std::endl;
    }

    //16.设置机械臂安全配置
    aubo_robot_namespace::RobotSafetyConfig safeconfig2;
    safeconfig2.robotReducedConfigJointSpeed[0] = 15 / 180.0 * M_PI;
    safeconfig2.robotReducedConfigJointSpeed[1] = 15 / 180.0 * M_PI;
    safeconfig2.robotReducedConfigJointSpeed[2] = 15 / 180.0 * M_PI;
    safeconfig2.robotReducedConfigJointSpeed[3] = 15 / 180.0 * M_PI;
    safeconfig2.robotReducedConfigJointSpeed[4] = 15 / 180.0 * M_PI;
    safeconfig2.robotReducedConfigJointSpeed[5] = 15 / 180.0 * M_PI;
    safeconfig2.robotReducedConfigTcpSpeed = 10;

    ret = rs_get_robot_safety_config(rshd, &safeconfig2);
    std::cout << std::endl;
    if (ret == RS_SUCC)
    {
        std::cout << "设置机械臂安全配置成功" << std::endl;
    }
    else
    {
        std::cerr << "设置机械臂安全配置失败。错误码：" << ret << std::endl;
    }

    //17.获取 socket 连接状态
    bool socketStatus;
    ret = rs_get_socket_status(rshd, &socketStatus);
    std::cout << std::endl;
    if (ret == RS_SUCC)
    {
        std::cout << "获取 socket 连接状态成功" << std::endl;
        std::cout << "socket 连接状态：" << socketStatus << std::endl;
    }
    else
    {
```

```
        std::cerr << "获取 socket 连接状态失败。错误码：" << ret << std::endl;
    }

    //18.获取机械臂末端速度
    float endspeed;
    ret = rs_get_robot_end_speed(rshd, &endspeed);
    std::cout << std::endl;
    if (ret == RS_SUCC)
    {
        std::cout << "获取机械臂末端速度成功" << std::endl;
        std::cout << "机械臂末端速度：" << endspeed << std::endl;
    }
    else
    {
        std::cerr << "获取机械臂末端速度失败。错误号：" << ret << std::endl;
    }
}
```

main.cpp 代码如下:

注意：要将下面代码中的 IP 地址（ROBOT_ADDR）修改为对应服务器的 IP 地址。

```
#include "example.h"
#include "windows.h"

#define ROBOT_ADDR "192.168.1.100"
#define ROBOT_PORT 8899

//机械臂控制上下文句柄
RSHD g_rshd = -1;

int main(int argc, char* argv[])
{
    //登录服务器
    if (example_login(g_rshd, ROBOT_ADDR, ROBOT_PORT))
    {
        //机械臂上电(必须连接真实机械臂)
        example_robotStartup(g_rshd);

        //设置和获取机械臂相关参数
        example_robot_param(g_rshd);
    }
    else
```

```
    {
        std::cout << "机械臂登录失败。" << std::endl;
    }

    //反初始化接口库
    rs_uninitialize();

    std::cout << "Please enter to exit" << std::endl;
    getchar();

    return 0;
}
```

4.6 IO

4.6.1 工具 IO

本示例是关于工具 IO。

example.h 代码如下：

```
#pragma once
#include <string>
#include "rsdef.h"

//工具 IO
void example_ToolIO(RSHD rshd);
```

example.cpp 代码如下：

```
#include "example.h"
#include "rsdef.h"

#include <Windows.h>
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <algorithm>
```

```
#define M_PI 3.14159265358979323846

//工具 IO
void example_ToolIO(RSHD rshd)
{
    int ret;

    //1.设置工具端电源电压类型
    aubo_robot_namespace::ToolPowerType type = aubo_robot_namespace::OUT_24
V;
    ret = rs_set_tool_power_type(rshd, type);
    std::cout << std::endl;
    if (ret == RS_SUCC)
    {
        std::cout << "INFO:设置工具端电压类型成功. 当前设置的类型:" << type
 << std::endl;
    }
    else
    {
        std::cerr << "ERROR:设置工具端电压类型失败." << std::endl;
    }

    //2.获取工具端电源电压类型
    aubo_robot_namespace::ToolPowerType type2;
    ret = rs_get_tool_power_type(rshd, &type2);
    std::cout << std::endl;
    if (ret == RS_SUCC)
    {
        std::cout << "INFO:获取电源电压类型成功.    结果为:" << type2 << st
d::endl;
    }
    else
    {
        std::cerr << "ERROR:设置工具端电压类型失败." << std::endl;
    }


    //3.获取工具端电源电压状态
    double ToolPowerVoltageStatus;

    ret = rs_get_tool_power_voltage(rshd, &ToolPowerVoltageStatus);
    std::cout << std::endl;
    if (ret == RS_SUCC)
    {
```

```
        std::cout << "INFO:获取工具端电源电压成功  电源电压:" << ToolPower
VoltageStatus << std::endl;
    }
    else
    {
        std::cout << "ERROR:获取工具端电源电压失败." << std::endl;
    }


    //4.设置工具端数字量 IO 的类型：输入或者输出
    aubo_robot_namespace::ToolDigitalIOAddr addr;
    aubo_robot_namespace::ToolIOType value;

    addr = aubo_robot_namespace::TOOL_DIGITAL_IO_1;
    value = aubo_robot_namespace::IO_OUT;

    ret = rs_set_tool_io_type(rshd, addr, value);
    std::cout << std::endl;
    if (ret == RS_SUCC)
    {
        std::cout << "INFO:设置工具端数字量 IO 的类型成功. 当前设置 addr:" <
< addr << "  类型:" << value << std::endl;
    }
    else
    {
        std::cerr << "ERROR:设置工具端数字量 IO 的类型失败. addr:" << addr
<< std::endl;
    }

    //5.根据名称设置工具端数字量 IO 的状态
    aubo_robot_namespace::IO_STATUS value2 = aubo_robot_namespace::IO_STA
TUS_VALID;
    ret = rs_set_tool_do_status(rshd, TOOL_IO_0, value2);
    std::cout << std::endl;
    if (ret == aubo_robot_namespace::InterfaceCallSuccCode)
    {
        std::cout << "INFO:设置工具端数字量 IO 的状态成功. 当前设置 name:"
<< TOOL_IO_0 << "  状态:" << value2 << std::endl;
    }
    else
    {
        std::cerr << "ERROR:设置工具端数字量 IO 的状态失败. name:" << TOO
L_IO_0 << std::endl;
    }
```

```
    Sleep(1000);

    //6.获取工具端 IO 状态
    double status = 0;
    std::cout << std::endl;
    //首先设置 tool_io_0 为数字输出
    if (RS_SUCC == rs_set_tool_io_type(rshd, TOOL_DIGITAL_IO_0, IO_OUT))
    {
        //设置 tool_io_0 数字输出有效
        if (RS_SUCC == rs_set_tool_do_status(rshd, TOOL_IO_0, IO_STATUS_V
ALID))
        {
            std::cout << "设置 " << TOOL_IO_0 << " 成功" << std::endl;
        }
        else
        {
            std::cerr << "设置 " << TOOL_IO_0 << " 失败" << std::endl;
        }

        //获取 tool_io_0 数字输出的状态
        if (RS_SUCC == rs_get_tool_io_status(rshd, TOOL_IO_0, &status))
        {
            std::cout << "获取 " << TOOL_IO_0 << " = " << status << std::e
ndl;
        }
        else
        {
            std::cerr << "获取 " << TOOL_IO_0 << " 失败" << std::endl;
        }
    }
}
```

main.cpp 代码如下:

注意：要将下面代码中的 IP 地址（ROBOT_ADDR）修改为对应服务器的 IP 地址。

```
#include "example.h"
#include "windows.h"

#define ROBOT_ADDR "192.168.1.100"
#define ROBOT_PORT 8899
```

```
//机械臂控制上下文句柄
RSHD g_rshd = -1;

int main(int argc, char* argv[])
{
    //登录服务器
    if (example_login(g_rshd, ROBOT_ADDR, ROBOT_PORT))
    {
        //机械臂上电(必须连接真实机械臂)
        example_robotStartup(g_rshd);

        //工具 IO
        example_ToolIO(g_rshd);
    }
    else
    {
        std::cout << "机械臂登录失败。" << std::endl;
    }

    //反初始化接口库
    rs_uninitialize();

    std::cout << "Please enter to exit" << std::endl;
    getchar();

    return 0;
}
```

4.6.2 用户 IO

本示例是关于工具 IO。

example.h 代码如下:

```
#pragma once
#include <string>
#include "rsdef.h"

//用户 IO
void example_boardIO(RSHD rshd);
```

example.cpp 代码如下:

```
#include "example.h"
#include "rsdef.h"

#include <Windows.h>
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <algorithm>



#define M_PI 3.14159265358979323846

////用户 IO
void example_boardIO(RSHD rshd)
{
    int ret;

    //1.根据接口板 IO 类型和名称设置 IO 状态
    ret = rs_set_board_io_status_by_name(rshd, aubo_robot_namespace::RobotBoard
UserDO, "U_DO_17", 1);
    Sleep(1000);
    std::cout << std::endl;
    if (ret != aubo_robot_namespace::InterfaceCallSuccCode)
    {
        std::cerr << "根据接口板 IO 类型和名称设置 IO 状态失败!" << std::endl;
    }
    else
    {
```

```
        std::cout << "根据接口版 IO 类型和名称设置 IO 状态成功！" << std::endl;
    }

    //2.根据接口板 IO 类型和地址设置 IO 状态
    int ioaddr2 = 40;
    double iovalue2 = 1;
    std::cout << std::endl;
    ret = rs_set_board_io_status_by_addr(rshd, aubo_robot_namespace::RobotBoardUserDO, ioaddr2, iovalue2);
    if (ret != RS_SUCC)
    {
        std::cerr << "根据接口板 IO 类型和地址设置 IO 状态失败!" << std::endl;
    }
    else
    {
        std::cout << "根据接口版 IO 类型和地址设置 IO 状态成功！" << std::endl;
    }
    Sleep(1000);
    std::cout << std::endl;


    //3.根据接口板 IO 类型和名称获取 IO 状态
    double value;
    std::cout << std::endl;
    rs_get_board_io_status_by_name(rshd, aubo_robot_namespace::RobotBoardUserDO, "U_DO_02", &value);
    std::cerr << "DO_02 状态:" << value << std::endl;

    //4.根据接口板 IO 类型和地址获取 IO 状态
    //获取 U_DI_00 的状态
    aubo_robot_namespace::RobotIoType iotype4 = aubo_robot_namespace::RobotBoardUserDI;
    int ioaddr3 = 36;
    double iovalue3;
    std::cout << std::endl;
    rs_get_board_io_status_by_addr(rshd, iotype4, ioaddr3, &iovalue3);
    std::cout << "addr " << ioaddr3 << " = " << iovalue3 << std::endl;
    //获取 U_DO_00 的状态
    iotype4 = aubo_robot_namespace::RobotBoardUserDO;
    ioaddr3 = 40;
    std::cout << std::endl;
    rs_get_board_io_status_by_addr(rshd, iotype4, ioaddr3, &iovalue3);
```

```
    std::cout << "addr " << ioaddr3 << " = " << iovalue3 << std::endl;
}
```

main.cpp 代码如下:

注意：要将下面代码中的 IP 地址（ROBOT_ADDR）修改为对应服务器的 IP 地址。

```
#include "example.h"
#include "windows.h"

#define ROBOT_ADDR "192.168.1.100"
#define ROBOT_PORT 8899

//机械臂控制上下文句柄
RSHD g_rshd = -1;

int main(int argc, char* argv[])
{
    //登录服务器
    if (example_login(g_rshd, ROBOT_ADDR, ROBOT_PORT))
    {
        //机械臂上电(必须连接真实机械臂)
        example_robotStartup(g_rshd);

        //用户 IO
        example_boardIO(g_rshd);
    }
    else
    {
        std::cout << "机械臂登录失败。" << std::endl;
    }

    //反初始化接口库
    rs_uninitialize();

    std::cout << "Please enter to exit" << std::endl;
    getchar();

    return 0;
}
```

4.6.3 安全 IO

本示例是关于安全 IO 之进入退出缩减模式。

example.h 代码如下:

```
#pragma once
#include <string>
#include "rsdef.h"

//安全 IO 之缩减模式
void example_reduce_mode(RSHD rshd);
```

example.cpp 代码如下:

```
#include "example.h"
#include "rsdef.h"

#include <Windows.h>
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <algorithm>



#define M_PI 3.14159265358979323846

//安全 IO 之缩减模式
void example_reduce_mode(RSHD rshd)
{
    //退出缩减模式
    rs_exit_reduce_mode(rshd);

    //机械臂安全配置
    aubo_robot_namespace::RobotSafetyConfig speed_config;
    speed_config.robotReducedConfigJointSpeed[0] = 15 / 180.0 * M_PI;
    speed_config.robotReducedConfigJointSpeed[1] = 15 / 180.0 * M_PI;
    speed_config.robotReducedConfigJointSpeed[2] = 15 / 180.0 * M_PI;
    speed_config.robotReducedConfigJointSpeed[3] = 15 / 180.0 * M_PI;
    speed_config.robotReducedConfigJointSpeed[4] = 15 / 180.0 * M_PI;
    speed_config.robotReducedConfigJointSpeed[5] = 15 / 180.0 * M_PI;
    speed_config.robotReducedConfigTcpSpeed = 10;
    rs_set_robot_safety_config(rshd, &speed_config);

    //进入缩减模式
```

```
    rs_enter_reduce_mode(rshd);

    //初始化运动属性
    rs_init_global_move_profile(rshd);

    //设置关节运动最大加速度
    aubo_robot_namespace::JointVelcAccParam jointMaxAcc;
    jointMaxAcc.jointPara[0] = 30 * M_PI / 180;
    jointMaxAcc.jointPara[1] = 30 * M_PI / 180;
    jointMaxAcc.jointPara[2] = 30 * M_PI / 180;
    jointMaxAcc.jointPara[3] = 30 * M_PI / 180;
    jointMaxAcc.jointPara[4] = 30 * M_PI / 180;
    jointMaxAcc.jointPara[5] = 30 * M_PI / 180;
    rs_set_global_joint_maxacc(rshd, &jointMaxAcc);

    //设置关节运动最大速度
    aubo_robot_namespace::JointVelcAccParam jointMaxVelc;
    jointMaxVelc.jointPara[0] = 30 * M_PI / 180;
    jointMaxVelc.jointPara[1] = 30 * M_PI / 180;
    jointMaxVelc.jointPara[2] = 30 * M_PI / 180;
    jointMaxVelc.jointPara[3] = 30 * M_PI / 180;
    jointMaxVelc.jointPara[4] = 30 * M_PI / 180;
    jointMaxVelc.jointPara[5] = 30 * M_PI / 180;
    rs_set_global_joint_maxvelc(rshd, &jointMaxVelc);

    //关节角 jointAngle
    double jointAngle[6] = {};
    jointAngle[0] = 60.443151 * M_PI / 180;
    jointAngle[1] = 42.275463 * M_PI / 180;
    jointAngle[2] = -97.679737 * M_PI / 180;
    jointAngle[3] = -49.990510 * M_PI / 180;
    jointAngle[4] = -90.007372 * M_PI / 180;
    jointAngle[5] = 62.567046 * M_PI / 180;

    //关节运动
    rs_move_joint(rshd, jointAngle, true);

    //退出缩减模式
    rs_exit_reduce_mode(rshd);

}
```

main.cpp 代码如下:

注意：要将下面代码中的 IP 地址（ROBOT_ADDR）修改为对应服务器的 IP 地址。

```
#include "example.h"
#include "windows.h"

#define ROBOT_ADDR "192.168.1.100"
#define ROBOT_PORT 8899

//机械臂控制上下文句柄
RSHD g_rshd = -1;

int main(int argc, char* argv[])
{
    //登录服务器
    if (example_login(g_rshd, ROBOT_ADDR, ROBOT_PORT))
    {
        //机械臂上电(必须连接真实机械臂)
        example_robotStartup(g_rshd);

        //安全 IO 之缩减模式
        example_reduce_mode(g_rshd);
    }
    else
    {
        std::cout << "机械臂登录失败。" << std::endl;
    }

    //反初始化接口库
    rs_uninitialize();

    std::cout << "Please enter to exit" << std::endl;
    getchar();

    return 0;
}
```

4.7 TCP 转 CAN 透传模式

本示例是在 CAN 透传模式下获取关节状态。

example.h 代码如下:

```
#pragma once
#include <string>
#include "rsdef.h"

//TCP 转 CAN 透传模式
//示例：获取关节状态
void example_TCP2CANBUS(RSHD rshd);
```

example.cpp 代码如下:

```
#include "example.h"
#include "rsdef.h"

#include <Windows.h>
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <algorithm>



#define M_PI 3.14159265358979323846

//TCP 转 CAN 透传模式
//示例：获取关节状态
void example_TCP2CANBUS(RSHD rshd)
{
    //进入 TCP 转 CAN 透传模式
    int ret = rs_enter_tcp2canbus_mode(rshd);
    if (ret == RS_SUCC)
    {
        std::cout << "进入 TCP 转 CAN 透传模式成功。" << std::endl;
    }
    else
    {
        std::cerr << "进入 TCP 转 CAN 透传模式失败。错误号为" << ret << st
d::endl;
    }
```

```
    aubo_robot_namespace::JointStatus jointStatus[6];
    //获取机械臂关节状态
    ret = rs_get_joint_status(rshd, jointStatus);
    if (ret == RS_SUCC)
    {
        std::cout << "获取关节状态成功。" << std::endl;
        std::cout << "----------关节状态信息----------" << std::endl;

        for (int i = 0; i < 6; i++)
        {
            std::cout << "关节 ID：" << i + 1 << "  ";
            std::cout << "关节电流：" << jointStatus[i].jointCurrentI << "  ";
            std::cout << "关节速度：" << jointStatus[i].jointSpeedMoto << "  ";
            std::cout << "关节角：" << jointStatus[i].jointPosJ * 180 / M_PI <<
 "  ";
            std::cout << "关节电压：" << jointStatus[i].jointCurVol << "  ";
            std::cout << "当前温度：" << jointStatus[i].jointCurTemp << "  ";
            std::cout << "电机目标电流：" << jointStatus[i].jointTagCurrentI <<
"  ";
            std::cout << "电机目标速度：" << jointStatus[i].jointTagSpeedMoto
<< "  ";
            std::cout << "目标关节角：" << jointStatus[i].jointTagPosJ * 180 /
M_PI << "  ";
            std::cout << "关节错误码：" << jointStatus[i].jointErrorNum << std::
endl;
        }

        std::cout << std::endl;

    }
    else
    {
        std::cerr << "获取关节状态失败。错误号为" << ret << std::endl;
    }

    //退出 TCP 转 CAN 透传模式
    rs_leave_tcp2canbus_mode(rshd);
}
```

main.cpp 代码如下:

注意：要将下面代码中的 IP 地址（ROBOT_ADDR）修改为对应服务器的 IP 地址。

```
#include "example.h"
#include "windows.h"

#define ROBOT_ADDR "192.168.1.100"
#define ROBOT_PORT 8899

//机械臂控制上下文句柄
RSHD g_rshd = -1;

int main(int argc, char* argv[])
{
    //登录服务器
    if (example_login(g_rshd, ROBOT_ADDR, ROBOT_PORT))
    {
        //机械臂上电(必须连接真实机械臂)
        example_robotStartup(g_rshd);

        //TCP 转 CAN 透传模式
        //示例：获取关节状态
        example_TCP2CANBUS(g_rshd);
    }
    else
    {
        std::cout << "机械臂登录失败。" << std::endl;
    }

    //反初始化接口库
    rs_uninitialize();

    std::cout << "Please enter to exit" << std::endl;
    getchar();

    return 0;
}
```

4.8 关节运动

4.8.1 rs_move_joint 函数

本示例是关节运动到指定关节角。

example.h 代码如下:

```
#pragma once
#include <string>
#include "rsdef.h"

bool example_moveJ(RSHD rshd);
```

example.cpp 代码如下:

```
#include "example.h"
#include "rsdef.h"

#include <Windows.h>
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <algorithm>



#define M_PI 3.14159265358979323846

//关节运动
bool example_moveJ(RSHD rshd)
{
    bool result = false;

    //初始化运动属性
    rs_init_global_move_profile(rshd);

    //设置关节运动最大加速度
    aubo_robot_namespace::JointVelcAccParam jointMaxAcc;
    jointMaxAcc.jointPara[0] = 30 * M_PI / 180;
    jointMaxAcc.jointPara[1] = 30 * M_PI / 180;
    jointMaxAcc.jointPara[2] = 30 * M_PI / 180;
    jointMaxAcc.jointPara[3] = 30 * M_PI / 180;
    jointMaxAcc.jointPara[4] = 30 * M_PI / 180;
```

```
    jointMaxAcc.jointPara[5] = 30 * M_PI / 180;
    rs_set_global_joint_maxacc(rshd, &jointMaxAcc);

    //设置关节运动最大速度
    aubo_robot_namespace::JointVelcAccParam jointMaxVelc;
    jointMaxVelc.jointPara[0] = 30 * M_PI / 180;
    jointMaxVelc.jointPara[1] = 30 * M_PI / 180;
    jointMaxVelc.jointPara[2] = 30 * M_PI / 180;
    jointMaxVelc.jointPara[3] = 30 * M_PI / 180;
    jointMaxVelc.jointPara[4] = 30 * M_PI / 180;
    jointMaxVelc.jointPara[5] = 30 * M_PI / 180;
    rs_set_global_joint_maxvelc(rshd, &jointMaxVelc);

    //关节角
    double initPos[6] = {
        -0.000172 / 180 * M_PI,
        -7.291862 / 180 * M_PI,
        -75.604718 / 180 * M_PI,
        21.596727 / 180 * M_PI,
        -89.999982 / 180 * M_PI,
        -0.00458 / 180 * M_PI
    };

    //关节运动
    if (rs_move_joint(rshd, initPos) == RS_SUCC)
    {
        result = true;
        std::cout << "关节运动成功" << std::endl;
    }
    else
    {
        std::cerr << "关节运动失败" << std::endl;
    }

    return result;
}
```

main.cpp 代码如下:

注意：要将下面代码中的 IP 地址（ROBOT_ADDR）修改为对应服务器的 IP 地址。

```
#include "example.h"
#include "windows.h"
```

```
#define ROBOT_ADDR "192.168.1.100"
#define ROBOT_PORT 8899

//机械臂控制上下文句柄
RSHD g_rshd = -1;

int main(int argc, char* argv[])
{
    //登录服务器
    if (example_login(g_rshd, ROBOT_ADDR, ROBOT_PORT))
    {
        //机械臂上电(必须连接真实机械臂)
        example_robotStartup(g_rshd);

        //关节运动
        example_moveJ(g_rshd);
    }
    else
    {
        std::cout << "机械臂登录失败。" << std::endl;
    }

    //反初始化接口库
    rs_uninitialize();

    std::cout << "Please enter to exit" << std::endl;
    getchar();

    return 0;
}
```

4.8.2 rs_move_joint_to 函数

示例 1：法兰盘中心在基坐标系下的位置

本示例是关节运动到目标位置。

example.h 代码如下:

```
#pragma once
#include <string>
#include "rsdef.h"

//关节运动到目标位置 rs_move_joint_to
//示例 1：法兰盘中心在基坐标系下
void example_move_joint_to1(RSHD rshd);
```

example.cpp 代码如下:

```
#include "example.h"
#include "rsdef.h"

#include <Windows.h>
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <algorithm>



#define M_PI 3.14159265358979323846

//关节运动到目标位置 rs_move_joint_to
//示例 1：法兰盘中心在基坐标系下
void example_move_joint_to1(RSHD rshd)
{
    //初始化运动属性
    rs_init_global_move_profile(rshd);

    //设置关节运动最大加速度
    aubo_robot_namespace::JointVelcAccParam jointMaxAcc;
    jointMaxAcc.jointPara[0] = 30 * M_PI / 180;
    jointMaxAcc.jointPara[1] = 30 * M_PI / 180;
    jointMaxAcc.jointPara[2] = 30 * M_PI / 180;
    jointMaxAcc.jointPara[3] = 30 * M_PI / 180;
```

```
jointMaxAcc.jointPara[4] = 30 * M_PI / 180;
jointMaxAcc.jointPara[5] = 30 * M_PI / 180;
rs_set_global_joint_maxacc(rshd, &jointMaxAcc);

//设置关节运动最大速度
aubo_robot_namespace::JointVelcAccParam jointMaxVelc;
jointMaxVelc.jointPara[0] = 30 * M_PI / 180;
jointMaxVelc.jointPara[1] = 30 * M_PI / 180;
jointMaxVelc.jointPara[2] = 30 * M_PI / 180;
jointMaxVelc.jointPara[3] = 30 * M_PI / 180;
jointMaxVelc.jointPara[4] = 30 * M_PI / 180;
jointMaxVelc.jointPara[5] = 30 * M_PI / 180;
rs_set_global_joint_maxvelc(rshd, &jointMaxVelc);

//起始路点
double initPos[6] = {
    173.108713 / 180 * M_PI,
    -12.075005 / 180 * M_PI,
    -83.663342 / 180 * M_PI,
    -15.641249 / 180 * M_PI,
    -89.140000 / 180 * M_PI,
    -28.328713 / 180 * M_PI
};

//关节运动到起始路点
rs_move_joint(rshd, initPos, true);

//设置工具参数，为法兰盘中心
aubo_robot_namespace::ToolInEndDesc toolEnd;
toolEnd.toolInEndPosition.x = 0;
toolEnd.toolInEndPosition.y = 0;
toolEnd.toolInEndPosition.z = 0;
toolEnd.toolInEndOrientation.w = 1;
toolEnd.toolInEndOrientation.x = 0;
toolEnd.toolInEndOrientation.y = 0;
toolEnd.toolInEndOrientation.z = 0;

//设置基坐标系
rs_set_base_coord(rshd);

//设置目标位置，法兰盘中心在基坐标系下的位置
aubo_robot_namespace::Pos posOnBase;
posOnBase.x = -0.2;
posOnBase.y = -0.6;
```

```
    posOnBase.z = 0;

    //轴动运动至目标位置
    rs_move_joint_to(rshd, &posOnBase, &toolEnd, true);

}
```

main.cpp 代码如下:

注意：要将下面代码中的 IP 地址（ROBOT_ADDR）修改为对应服务器的 IP 地址。

```
#include "example.h"
#include "windows.h"

#define ROBOT_ADDR "192.168.1.100"
#define ROBOT_PORT 8899

//机械臂控制上下文句柄
RSHD g_rshd = -1;

int main(int argc, char* argv[])
{
    //登录服务器
    if (example_login(g_rshd, ROBOT_ADDR, ROBOT_PORT))
    {
        //机械臂上电(必须连接真实机械臂)
        example_robotStartup(g_rshd);

        //关节运动到目标位置 rs_move_joint_to
        //示例 1：法兰盘中心在 Base 坐标系下
        example_move_joint_to1(g_rshd);
    }
    else
    {
        std::cout << "机械臂登录失败。" << std::endl;
    }

    //反初始化接口库
    rs_uninitialize();

    std::cout << "Please enter to exit" << std::endl;
    getchar();
```

```
    return 0;
}
```

示例 2：工具末端在基坐标系下的位置

本示例是关节运动到目标位置。

example.h 代码如下:

```
#pragma once
#include <string>
#include "rsdef.h"

//关节运动到目标位置 rs_move_joint_to
//示例 3：工具末端在基坐标系下
void example_move_joint_to3(RSHD rshd);
```

example.cpp 代码如下:

```
#include "example.h"
#include "rsdef.h"

#include <Windows.h>
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <algorithm>



#define M_PI 3.14159265358979323846

//示例 3：工具末端在基坐标系下
void example_move_joint_to3(RSHD rshd)
{
    //初始化运动属性
    rs_init_global_move_profile(rshd);

    //设置关节运动最大加速度
    aubo_robot_namespace::JointVelcAccParam jointMaxAcc;
    jointMaxAcc.jointPara[0] = 30 * M_PI / 180;
    jointMaxAcc.jointPara[1] = 30 * M_PI / 180;
```

```
jointMaxAcc.jointPara[2] = 30 * M_PI / 180;
jointMaxAcc.jointPara[3] = 30 * M_PI / 180;
jointMaxAcc.jointPara[4] = 30 * M_PI / 180;
jointMaxAcc.jointPara[5] = 30 * M_PI / 180;
rs_set_global_joint_maxacc(rshd, &jointMaxAcc);

//设置关节运动最大速度
aubo_robot_namespace::JointVelcAccParam jointMaxVelc;
jointMaxVelc.jointPara[0] = 30 * M_PI / 180;
jointMaxVelc.jointPara[1] = 30 * M_PI / 180;
jointMaxVelc.jointPara[2] = 30 * M_PI / 180;
jointMaxVelc.jointPara[3] = 30 * M_PI / 180;
jointMaxVelc.jointPara[4] = 30 * M_PI / 180;
jointMaxVelc.jointPara[5] = 30 * M_PI / 180;
rs_set_global_joint_maxvelc(rshd, &jointMaxVelc);

//起始路点
double initPos[6] = {
    173.108713 / 180 * M_PI,
    -12.075005 / 180 * M_PI,
    -83.663342 / 180 * M_PI,
    -15.641249 / 180 * M_PI,
    -89.140000 / 180 * M_PI,
    -28.328713 / 180 * M_PI
};

//关节运动到起始路点
rs_move_joint(rshd, initPos, true);

//设置工具参数，为工具末端
aubo_robot_namespace::ToolInEndDesc toolEnd;
toolEnd.toolInEndPosition.x = -0.177341;
toolEnd.toolInEndPosition.y = 0.002327;
toolEnd.toolInEndPosition.z = 0.146822;
toolEnd.toolInEndOrientation.w = 1;
toolEnd.toolInEndOrientation.x = 0;
toolEnd.toolInEndOrientation.y = 0;
toolEnd.toolInEndOrientation.z = 0;

//设置基坐标系
rs_set_base_coord(rshd);

//设置目标位置，工具末端在基坐标系下的位置
aubo_robot_namespace::Pos posOnBase;
```

```
    posOnBase.x = 0.6;
    posOnBase.y = 0.0;
    posOnBase.z = 0.5;

    //轴动运动至目标位置
    rs_move_joint_to(rshd, &posOnBase, &toolEnd, true);
}
```

main.cpp 代码如下:

注意：要将下面代码中的 IP 地址（ROBOT_ADDR）修改为对应服务器的 IP 地址。

```
#include "example.h"
#include "windows.h"

#define ROBOT_ADDR "192.168.1.100"
#define ROBOT_PORT 8899

//机械臂控制上下文句柄
RSHD g_rshd = -1;

int main(int argc, char* argv[])
{
    //登录服务器
    if (example_login(g_rshd, ROBOT_ADDR, ROBOT_PORT))
    {
        //机械臂上电(必须连接真实机械臂)
        example_robotStartup(g_rshd);

        //关节运动到目标位置 rs_move_joint_to
        //示例 3：工具末端在 Base 坐标系下
        example_move_joint_to3(g_rshd);
    }
    else
    {
        std::cout << "机械臂登录失败。" << std::endl;
    }

    //反初始化接口库
    rs_uninitialize();

    std::cout << "Please enter to exit" << std::endl;
    getchar();
```

```
    return 0;
}
```

4.9 跟随模式

4.9.1 跟随模式的轴动 rs_follow_mode_move_joint

本示例是跟随模式的轴动。

example.h 代码如下:

```
#pragma once
#include <string>
#include "rsdef.h"

//基于跟随模式的轴动
bool example_follow_mode_moveJ(RSHD rshd);
```

example.cpp 代码如下:

```
#include "example.h"
#include "rsdef.h"

#include <Windows.h>
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <algorithm>



#define M_PI 3.14159265358979323846

//基于跟随模式的轴动
bool example_follow_mode_moveJ(RSHD rshd)
{
    bool result = false;
    //初始化运动属性
    rs_init_global_move_profile(rshd);

    //设置关节型运动的最大加速度
    aubo_robot_namespace::JointVelcAccParam jointMaxAcc;
```

```
    jointMaxAcc.jointPara[0] = 50.0 / 180.0 * M_PI;
    jointMaxAcc.jointPara[1] = 50.0 / 180.0 * M_PI;
    jointMaxAcc.jointPara[2] = 50.0 / 180.0 * M_PI;
    jointMaxAcc.jointPara[3] = 50.0 / 180.0 * M_PI;
    jointMaxAcc.jointPara[4] = 50.0 / 180.0 * M_PI;
    jointMaxAcc.jointPara[5] = 50.0 / 180.0 * M_PI;          //接口要求单位是弧
度
    rs_set_global_joint_maxacc(rshd, &jointMaxAcc);

    //设置关节型运动的最大速度
    aubo_robot_namespace::JointVelcAccParam jointMaxVelc;
    jointMaxVelc.jointPara[0] = 50.0 / 180.0 * M_PI;
    jointMaxVelc.jointPara[1] = 50.0 / 180.0 * M_PI;
    jointMaxVelc.jointPara[2] = 50.0 / 180.0 * M_PI;
    jointMaxVelc.jointPara[3] = 50.0 / 180.0 * M_PI;
    jointMaxVelc.jointPara[4] = 50.0 / 180.0 * M_PI;
    jointMaxVelc.jointPara[5] = 50.0 / 180.0 * M_PI;    //接口要求单位是弧度
    rs_set_global_joint_maxvelc(rshd, &jointMaxVelc);

    //该位置为机械臂的初始位置
    double initPos[6] = {
        -0.000172 / 180 * M_PI,
        -7.291862 / 180 * M_PI,
        -75.604718 / 180 * M_PI,
        21.596727 / 180 * M_PI,
        -89.999982 / 180 * M_PI,
        -0.00458 / 180 * M_PI
    };

    //首先运动到初始位置
    if (rs_move_joint(rshd, initPos) == RS_SUCC)
    {
        result = true;
        std::cout << "关节运动到初始位置" << std::endl;
    }
    else
    {
        std::cerr << "关节运动失败" << std::endl;
    }

    //基于跟随模式的关节运动
    for (int i = 0; i < 1000; i++)
    {
        initPos[0] = initPos[0] + 0.0001;
```

```
        std::cout << initPos[0] * 180 / M_PI << std::endl;

        if (rs_follow_mode_move_joint(rshd, initPos) == RS_SUCC)
        {
            std::cout << "基于跟随模式的轴动成功" << std::endl;
        }
        else
        {
            result = false;
            std::cerr << "基于跟随模式的轴动失败" << std::endl;
        }
    }

    return result;
}
```

main.cpp 代码如下：

注意：要将下面代码中的 IP 地址（ROBOT_ADDR）修改为对应服务器的 IP 地址。

```
#include "example.h"
#include "windows.h"

#define ROBOT_ADDR "192.168.1.100"
#define ROBOT_PORT 8899

//机械臂控制上下文句柄
RSHD g_rshd = -1;

int main(int argc, char* argv[])
{
    //登录服务器
    if (example_login(g_rshd, ROBOT_ADDR, ROBOT_PORT))
    {
        //机械臂上电(必须连接真实机械臂)
        example_robotStartup(g_rshd);

        //基于跟随模式的轴动
        example_follow_mode_moveJ(g_rshd);
    }
    else
    {
        std::cout << "机械臂登录失败。" << std::endl;
```

```
    }

    //反初始化接口库
    rs_uninitialize();

    std::cout << "Please enter to exit" << std::endl;
    getchar();

    return 0;
}
```

4.9.2 跟随模式之提前到位

本示例是跟随模式之提前到位。

example.h 代码如下：

```
#pragma once
#include <string>
#include "rsdef.h"

//跟随模式之提前到位
void example_arrival_ahead(RSHD rshd);
```

example.cpp 代码如下：

```
#include "example.h"
#include "rsdef.h"

#include <Windows.h>
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <algorithm>


#define M_PI 3.14159265358979323846

//跟随模式之提前到位
void example_arrival_ahead(RSHD rshd)
{
```

```
//初始化运动属性
rs_init_global_move_profile(rshd);

//设置关节运动最大加速度
aubo_robot_namespace::JointVelcAccParam jointMaxAcc;
jointMaxAcc.jointPara[0] = 30 * M_PI / 180;
jointMaxAcc.jointPara[1] = 30 * M_PI / 180;
jointMaxAcc.jointPara[2] = 30 * M_PI / 180;
jointMaxAcc.jointPara[3] = 30 * M_PI / 180;
jointMaxAcc.jointPara[4] = 30 * M_PI / 180;
jointMaxAcc.jointPara[5] = 30 * M_PI / 180;
rs_set_global_joint_maxacc(rshd, &jointMaxAcc);

//设置关节运动最大速度
aubo_robot_namespace::JointVelcAccParam jointMaxVelc;
jointMaxVelc.jointPara[0] = 30 * M_PI / 180;
jointMaxVelc.jointPara[1] = 30 * M_PI / 180;
jointMaxVelc.jointPara[2] = 30 * M_PI / 180;
jointMaxVelc.jointPara[3] = 30 * M_PI / 180;
jointMaxVelc.jointPara[4] = 30 * M_PI / 180;
jointMaxVelc.jointPara[5] = 30 * M_PI / 180;
rs_set_global_joint_maxvelc(rshd, &jointMaxVelc);

//起始路点
double initPos[6] = {
    173.108713 / 180 * M_PI,
    -12.075005 / 180 * M_PI,
    -83.663342 / 180 * M_PI,
    -15.641249 / 180 * M_PI,
    -89.140000 / 180 * M_PI,
    -28.328713 / 180 * M_PI
};

//关节运动到起始路点
rs_move_joint(rshd, initPos, true);

double jointAngle[aubo_robot_namespace::ARM_DOF] = { 0 };
for (int i = 0; i < 5; i++)
{
    if (i % 2 == 0)
    {
        //跟随模式之提前到位  当前仅适用于关节运动
        //设置提前到位的距离模式
        rs_set_arrival_ahead_distance(rshd, 0.2);
```

```
        }
        else
        {
            rs_set_no_arrival_ahead(rshd);
        }

        jointAngle[0] = 20.0 / 180.0 * M_PI;
        jointAngle[1] = 0.0 / 180.0 * M_PI;
        jointAngle[2] = 90.0 / 180.0 * M_PI;
        jointAngle[3] = 0.0 / 180.0 * M_PI;
        jointAngle[4] = 90.0 / 180.0 * M_PI;
        jointAngle[5] = 0.0 / 180.0 * M_PI;

        int ret = rs_move_joint(rshd, jointAngle, true);
        if (ret != RS_SUCC)
        {
            std::cerr << "运动 1 失败。错误号为：" << ret << std::endl;
            break;
        }
        else
        {
            std::cerr << "运动 1 成功。i = " << i << std::endl;
        }

        jointAngle[0] = 50.0 / 180.0 * M_PI;
        jointAngle[1] = 40.0 / 180.0 * M_PI;
        jointAngle[2] = 78.0 / 180.0 * M_PI;
        jointAngle[3] = 20.0 / 180.0 * M_PI;
        jointAngle[4] = 66.0 / 180.0 * M_PI;
        jointAngle[5] = 0.0 / 180.0 * M_PI;

        ret = rs_move_joint(rshd, jointAngle, true);
        if (ret != RS_SUCC)
        {
            std::cerr << "运动 2 失败。错误号为：" << ret << std::endl;
            break;
        }
        else
        {
            std::cerr << "运动 2 成功。 i = " << i << std::endl;
        }
    }

}
```

main.cpp 代码如下:

注意：要将下面代码中的 IP 地址（ROBOT_ADDR）修改为对应服务器的 IP 地址。

```
#include "example.h"
#include "windows.h"

#define ROBOT_ADDR "192.168.1.100"
#define ROBOT_PORT 8899

//机械臂控制上下文句柄
RSHD g_rshd = -1;

int main(int argc, char* argv[])
{
    //登录服务器
    if (example_login(g_rshd, ROBOT_ADDR, ROBOT_PORT))
    {
        //机械臂上电(必须连接真实机械臂)
        example_robotStartup(g_rshd);

        //跟随模式之提前到位
        example_arrival_ahead(g_rshd);
    }
    else
    {
        std::cout << "机械臂登录失败。" << std::endl;
    }

    //反初始化接口库
    rs_uninitialize();

    std::cout << "Please enter to exit" << std::endl;
    getchar();

    return 0;
}
```

4.10 直线运动

4.10.1 rs_move_line 函数

本示例是直线运动到指定关节角。

example.h 代码如下:

```
#pragma once
#include <string>
#include "rsdef.h"

//机械臂保持当前姿态直线运动测试
void example_moveL(RSHD rshd);
```

example.cpp 代码如下:

```
#include "example.h"
#include "rsdef.h"

#include <Windows.h>
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <algorithm>



#define M_PI 3.14159265358979323846

//机械臂保持当前姿态直线运动测试
void example_moveL(RSHD rshd)
{
    //初始化运动属性
    rs_init_global_move_profile(rshd);

    //设置关节运动最大加速度
    aubo_robot_namespace::JointVelcAccParam jointMaxAcc;
    jointMaxAcc.jointPara[0] = 30 * M_PI / 180;
    jointMaxAcc.jointPara[1] = 30 * M_PI / 180;
    jointMaxAcc.jointPara[2] = 30 * M_PI / 180;
    jointMaxAcc.jointPara[3] = 30 * M_PI / 180;
    jointMaxAcc.jointPara[4] = 30 * M_PI / 180;
    jointMaxAcc.jointPara[5] = 30 * M_PI / 180;
```

```
rs_set_global_joint_maxacc(rshd, &jointMaxAcc);

//设置关节运动最大速度
aubo_robot_namespace::JointVelcAccParam jointMaxVelc;
jointMaxVelc.jointPara[0] = 30 * M_PI / 180;
jointMaxVelc.jointPara[1] = 30 * M_PI / 180;
jointMaxVelc.jointPara[2] = 30 * M_PI / 180;
jointMaxVelc.jointPara[3] = 30 * M_PI / 180;
jointMaxVelc.jointPara[4] = 30 * M_PI / 180;
jointMaxVelc.jointPara[5] = 30 * M_PI / 180;
rs_set_global_joint_maxvelc(rshd, &jointMaxVelc);

//设置末端型运动最大加速度
double lineMaxAcc = 0.2;
rs_set_global_end_max_line_acc(rshd, lineMaxAcc);

//设置末端型运动最大速度
double lineMaxVelc = 0.2;
rs_set_global_end_max_line_velc(rshd, lineMaxVelc);


//初始位置
double initPos[6] = {
    173.108713 / 180 * M_PI,
    -12.075005 / 180 * M_PI,
    -83.663342 / 180 * M_PI,
    -15.641249 / 180 * M_PI,
    -89.140000 / 180 * M_PI,
    -28.328713 / 180 * M_PI
};

//关节运动到初始位置
rs_move_joint(rshd, initPos);

double jointAngle[6] = {};
jointAngle[0] = 173.106678 * M_PI / 180;
jointAngle[1] = -6.304596 * M_PI / 180;
jointAngle[2] = -63.515227 * M_PI / 180;
jointAngle[3] = -1.265599 * M_PI / 180;
jointAngle[4] = -89.161453 * M_PI / 180;
jointAngle[5] = -28.22 * M_PI / 180;

//直线运动
int ret = rs_move_line(rshd, jointAngle, true);
```

```
    if (ret == RS_SUCC)
    {
        std::cout << "直线运动成功" << std::endl;
    }
    else
    {
        std::cerr << "直线运动失败" << std::endl;
    }
}
```

main.cpp 代码如下:

注意：要将下面代码中的 IP 地址（ROBOT_ADDR）修改为对应服务器的 IP 地址。

```
#include "example.h"
#include "windows.h"

#define ROBOT_ADDR "192.168.1.100"
#define ROBOT_PORT 8899

//机械臂控制上下文句柄
RSHD g_rshd = -1;

int main(int argc, char* argv[])
{
    //登录服务器
    if (example_login(g_rshd, ROBOT_ADDR, ROBOT_PORT))
    {
        //机械臂上电(必须连接真实机械臂)
        example_robotStartup(g_rshd);

        //机械臂保持当前姿态直线运动测试
        example_moveL(g_rshd);
    }
    else
    {
        std::cout << "机械臂登录失败。" << std::endl;
    }

    //反初始化接口库
    rs_uninitialize();

    std::cout << "Please enter to exit" << std::endl;
```

```
    getchar();

    return 0;
}
```

4.10.2 rs_move_line_to 函数

示例 1：法兰盘中心在基坐标系下的位置

本示例是直线运动到目标位置。

example.h 代码如下:

```
#pragma once
#include <string>
#include "rsdef.h"

//直线运动到目标位置 rs_move_line_to
//示例 1：法兰盘中心在 Base 坐标系下
void example_move_line_to1(RSHD rshd);
```

example.cpp 代码如下:

```
#include "example.h"
#include "rsdef.h"

#include <Windows.h>
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <algorithm>



#define M_PI 3.14159265358979323846

//直线运动到目标位置 rs_move_line_to
//示例 1：法兰盘中心在基坐标系下
void example_move_line_to1(RSHD rshd)
{
    //初始化运动属性
    rs_init_global_move_profile(rshd);
```

```
//设置关节运动最大加速度
aubo_robot_namespace::JointVelcAccParam jointMaxAcc;
jointMaxAcc.jointPara[0] = 30 * M_PI / 180;
jointMaxAcc.jointPara[1] = 30 * M_PI / 180;
jointMaxAcc.jointPara[2] = 30 * M_PI / 180;
jointMaxAcc.jointPara[3] = 30 * M_PI / 180;
jointMaxAcc.jointPara[4] = 30 * M_PI / 180;
jointMaxAcc.jointPara[5] = 30 * M_PI / 180;
rs_set_global_joint_maxacc(rshd, &jointMaxAcc);

//设置关节运动最大速度
aubo_robot_namespace::JointVelcAccParam jointMaxVelc;
jointMaxVelc.jointPara[0] = 30 * M_PI / 180;
jointMaxVelc.jointPara[1] = 30 * M_PI / 180;
jointMaxVelc.jointPara[2] = 30 * M_PI / 180;
jointMaxVelc.jointPara[3] = 30 * M_PI / 180;
jointMaxVelc.jointPara[4] = 30 * M_PI / 180;
jointMaxVelc.jointPara[5] = 30 * M_PI / 180;
rs_set_global_joint_maxvelc(rshd, &jointMaxVelc);

//设置末端型运动最大加速度
double lineMaxAcc = 0.2;
rs_set_global_end_max_line_acc(rshd, lineMaxAcc);

//设置末端型运动最大速度
double lineMaxVelc = 0.2;
rs_set_global_end_max_line_velc(rshd, lineMaxVelc);


//初始位置
double initPos[6] = {
    173.108713 / 180 * M_PI,
    -12.075005 / 180 * M_PI,
    -83.663342 / 180 * M_PI,
    -15.641249 / 180 * M_PI,
    -89.140000 / 180 * M_PI,
    -28.328713 / 180 * M_PI
};

//关节运动到初始位置
rs_move_joint(rshd, initPos);

//设置工具参数，为法兰盘中心
```

```
    aubo_robot_namespace::ToolInEndDesc toolEnd;
    toolEnd.toolInEndPosition.x = 0;
    toolEnd.toolInEndPosition.y = 0;
    toolEnd.toolInEndPosition.z = 0;
    toolEnd.toolInEndOrientation.w = 1;
    toolEnd.toolInEndOrientation.x = 0;
    toolEnd.toolInEndOrientation.y = 0;
    toolEnd.toolInEndOrientation.z = 0;

    //设置基坐标系
    //rs_set_base_coord(rshd);

    //设置目标位置，法兰盘中心在基坐标系下的位置
    aubo_robot_namespace::Pos posOnBase;
    posOnBase.x = -0.2;
    posOnBase.y = -0.6;
    posOnBase.z = 0;

    //直线运动至目标位置
    rs_move_line_to(rshd, &posOnBase, &toolEnd, true);
}
```

main.cpp 代码如下:

注意：要将下面代码中的 IP 地址（ROBOT_ADDR）修改为对应服务器的 IP 地址。

```
#include "example.h"
#include "windows.h"

#define ROBOT_ADDR "192.168.1.100"
#define ROBOT_PORT 8899

//机械臂控制上下文句柄
RSHD g_rshd = -1;

int main(int argc, char* argv[])
{
    //登录服务器
    if (example_login(g_rshd, ROBOT_ADDR, ROBOT_PORT))
    {
        //机械臂上电(必须连接真实机械臂)
        example_robotStartup(g_rshd);
```

```
        //直线运动到目标位置 rs_move_line_to
        //示例 1：法兰盘中心在 Base 坐标系下
        example_move_line_to1(g_rshd);
    }
    else
    {
        std::cout << "机械臂登录失败。" << std::endl;
    }

    //反初始化接口库
    rs_uninitialize();

    std::cout << "Please enter to exit" << std::endl;
    getchar();

    return 0;
}
```

示例 2：工具末端在基坐标系下的位置

本示例是直线运动到目标位置。

example.h 代码如下:

```
#pragma once
#include <string>
#include "rsdef.h"

//示例 3：工具末端在 Base 坐标系下
void example_move_line_to3(RSHD rshd);
```

example.cpp 代码如下:

```
#include "example.h"
#include "rsdef.h"

#include <Windows.h>
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <algorithm>
```

```
#define M_PI 3.14159265358979323846

void example_move_line_to3(RSHD rshd)
{
    //初始化运动属性
    rs_init_global_move_profile(rshd);

    //设置关节运动最大加速度
    aubo_robot_namespace::JointVelcAccParam jointMaxAcc;
    jointMaxAcc.jointPara[0] = 30 * M_PI / 180;
    jointMaxAcc.jointPara[1] = 30 * M_PI / 180;
    jointMaxAcc.jointPara[2] = 30 * M_PI / 180;
    jointMaxAcc.jointPara[3] = 30 * M_PI / 180;
    jointMaxAcc.jointPara[4] = 30 * M_PI / 180;
    jointMaxAcc.jointPara[5] = 30 * M_PI / 180;
    rs_set_global_joint_maxacc(rshd, &jointMaxAcc);

    //设置关节运动最大速度
    aubo_robot_namespace::JointVelcAccParam jointMaxVelc;
    jointMaxVelc.jointPara[0] = 30 * M_PI / 180;
    jointMaxVelc.jointPara[1] = 30 * M_PI / 180;
    jointMaxVelc.jointPara[2] = 30 * M_PI / 180;
    jointMaxVelc.jointPara[3] = 30 * M_PI / 180;
    jointMaxVelc.jointPara[4] = 30 * M_PI / 180;
    jointMaxVelc.jointPara[5] = 30 * M_PI / 180;
    rs_set_global_joint_maxvelc(rshd, &jointMaxVelc);

    //设置末端型运动最大加速度
    double lineMaxAcc = 0.2;
    rs_set_global_end_max_line_acc(rshd, lineMaxAcc);

    //设置末端型运动最大速度
    double lineMaxVelc = 0.2;
    rs_set_global_end_max_line_velc(rshd, lineMaxVelc);


    //初始位置
    double initPos[6] = {
        173.108713 / 180 * M_PI,
        -12.075005 / 180 * M_PI,
        -83.663342 / 180 * M_PI,
        -15.641249 / 180 * M_PI,
```

```
        -89.140000 / 180 * M_PI,
        -28.328713 / 180 * M_PI
    };

    //关节运动到初始位置
    rs_move_joint(rshd, initPos);

    //设置工具参数，为工具末端
    aubo_robot_namespace::ToolInEndDesc toolEnd;
    toolEnd.toolInEndPosition.x = -0.177341;
    toolEnd.toolInEndPosition.y = 0.002327;
    toolEnd.toolInEndPosition.z = 0.146822;
    toolEnd.toolInEndOrientation.w = 1;
    toolEnd.toolInEndOrientation.x = 0;
    toolEnd.toolInEndOrientation.y = 0;
    toolEnd.toolInEndOrientation.z = 0;

    //设置基坐标系
    rs_set_base_coord(rshd);

    //设置目标位置，工具末端在基坐标系下的位置
    aubo_robot_namespace::Pos posOnBase;
    posOnBase.x = 0.56;
    posOnBase.y = -0.10;
    posOnBase.z = 0.33;

    //直线运动至目标位置
    rs_move_line_to(rshd, &posOnBase, &toolEnd, true);
}
```

main.cpp 代码如下:

注意：要将下面代码中的 IP 地址（ROBOT_ADDR）修改为对应服务器的 IP 地址。

```
#include "example.h"
#include "windows.h"

#define ROBOT_ADDR "192.168.1.100"
#define ROBOT_PORT 8899

//机械臂控制上下文句柄
RSHD g_rshd = -1;
```

```
int main(int argc, char* argv[])
{
    //登录服务器
    if (example_login(g_rshd, ROBOT_ADDR, ROBOT_PORT))
    {
        //机械臂上电(必须连接真实机械臂)
        example_robotStartup(g_rshd);

        //示例 3：工具末端在 Base 坐标系下
        example_move_line_to3(g_rshd);
    }
    else
    {
        std::cout << "机械臂登录失败。" << std::endl;
    }

    //反初始化接口库
    rs_uninitialize();

    std::cout << "Please enter to exit" << std::endl;
    getchar();

    return 0;
}
```

4.11 偏移运动

4.11.1 rs_set_relative_offset_on_base 函数

示例 1：法兰盘中心在基坐标系下

本示例是机械臂做姿态偏移运动，法兰盘中心在基坐标系下沿 X 轴姿态偏移。

example.h 代码如下：

```
#pragma once
#include <string>
#include "rsdef.h"

//示例 1：法兰盘中心在基坐标系下
void example_move_relarive1(RSHD rshd);
```

example.cpp 代码如下：

```
#include "example.h"
#include "rsdef.h"

#include <Windows.h>
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <algorithm>



#define M_PI 3.14159265358979323846

//示例 1：法兰盘中心在基坐标系下
void example_move_relarive1(RSHD rshd)
{
    //初始化运动属性
    rs_init_global_move_profile(rshd);

    //设置关节运动最大加速度
    aubo_robot_namespace::JointVelcAccParam jointMaxAcc;
    jointMaxAcc.jointPara[0] = 30 * M_PI / 180;
    jointMaxAcc.jointPara[1] = 30 * M_PI / 180;
    jointMaxAcc.jointPara[2] = 30 * M_PI / 180;
```

```
jointMaxAcc.jointPara[3] = 30 * M_PI / 180;
jointMaxAcc.jointPara[4] = 30 * M_PI / 180;
jointMaxAcc.jointPara[5] = 30 * M_PI / 180;
rs_set_global_joint_maxacc(rshd, &jointMaxAcc);

//设置关节运动最大速度
aubo_robot_namespace::JointVelcAccParam jointMaxVelc;
jointMaxVelc.jointPara[0] = 30 * M_PI / 180;
jointMaxVelc.jointPara[1] = 30 * M_PI / 180;
jointMaxVelc.jointPara[2] = 30 * M_PI / 180;
jointMaxVelc.jointPara[3] = 30 * M_PI / 180;
jointMaxVelc.jointPara[4] = 30 * M_PI / 180;
jointMaxVelc.jointPara[5] = 30 * M_PI / 180;
rs_set_global_joint_maxvelc(rshd, &jointMaxVelc);

//设置末端型运动最大加速度
double lineMaxAcc = 0.2;
rs_set_global_end_max_line_acc(rshd, lineMaxAcc);

//设置末端型运动最大速度
double lineMaxVelc = 0.2;
rs_set_global_end_max_line_velc(rshd, lineMaxVelc);

//设置偏移
aubo_robot_namespace::MoveRelative relativeOnBase;
relativeOnBase.ena = true;
relativeOnBase.relativePosition[0] = 0;
relativeOnBase.relativePosition[1] = 0;
relativeOnBase.relativePosition[2] = 0;
aubo_robot_namespace::Rpy rpyRelative;
aubo_robot_namespace::Ori oriRelative;
rpyRelative.rx = 10.0 / 180 * M_PI; //要加一位小数点
rpyRelative.ry = 0.0 / 180 * M_PI;
rpyRelative.rz = 0.0 / 180 * M_PI;
rs_rpy_to_quaternion(rshd, &rpyRelative, &oriRelative);
relativeOnBase.relativeOri = oriRelative;

//偏移：法兰盘中心在基坐标系下沿 X 轴姿态偏移 10 度
rs_set_relative_offset_on_base(rshd, &relativeOnBase);

//关节角
double jointAngle[6] = {};
jointAngle[0] = 173.108713 * M_PI / 180;
jointAngle[1] = -12.075005 * M_PI / 180;
```

```
    jointAngle[2] = -83.663342 * M_PI / 180;
    jointAngle[3] = -15.641249 * M_PI / 180;
    jointAngle[4] = -89.140000 * M_PI / 180;
    jointAngle[5] = -28.328713 * M_PI / 180;

    //关节运动
    rs_move_joint(rshd, jointAngle, true);
}
```

main.cpp 代码如下：

注意：要将下面代码中的 IP 地址（ROBOT_ADDR）修改为对应服务器的 IP 地址。

```
#include "example.h"
#include "windows.h"

#define ROBOT_ADDR "192.168.1.100"
#define ROBOT_PORT 8899

//机械臂控制上下文句柄
RSHD g_rshd = -1;

int main(int argc, char* argv[])
{
    //登录服务器
    if (example_login(g_rshd, ROBOT_ADDR, ROBOT_PORT))
    {
        //机械臂上电(必须连接真实机械臂)
        example_robotStartup(g_rshd);

        //示例 1：法兰盘中心在 Base 坐标系下
        example_move_relarive1(g_rshd);
    }
    else
    {
        std::cout << "机械臂登录失败。" << std::endl;
    }

    //反初始化接口库
    rs_uninitialize();

    std::cout << "Please enter to exit" << std::endl;
    getchar();
```

```
    return 0;
}
```

示例 2：工具末端在基坐标系下

本示例是机械臂做姿态偏移运动，工具在基坐标系下沿 X 轴姿态偏移。

example.h 代码如下：

```
#pragma once
#include <string>
#include "rsdef.h"

//示例 4：工具末端在基坐标系下
void example_move_relarive4(RSHD rshd);
```

example.cpp 代码如下：

```
#include "example.h"
#include "rsdef.h"

#include <Windows.h>
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <algorithm>



#define M_PI 3.14159265358979323846

//示例 4：工具末端在基坐标系下
void example_move_relarive4(RSHD rshd)
{
    //初始化运动属性
    rs_init_global_move_profile(rshd);

    //设置关节运动最大加速度
    aubo_robot_namespace::JointVelcAccParam jointMaxAcc;
    jointMaxAcc.jointPara[0] = 30 * M_PI / 180;
    jointMaxAcc.jointPara[1] = 30 * M_PI / 180;
    jointMaxAcc.jointPara[2] = 30 * M_PI / 180;
```

```
jointMaxAcc.jointPara[3] = 30 * M_PI / 180;
jointMaxAcc.jointPara[4] = 30 * M_PI / 180;
jointMaxAcc.jointPara[5] = 30 * M_PI / 180;
rs_set_global_joint_maxacc(rshd, &jointMaxAcc);

//设置关节运动最大速度
aubo_robot_namespace::JointVelcAccParam jointMaxVelc;
jointMaxVelc.jointPara[0] = 30 * M_PI / 180;
jointMaxVelc.jointPara[1] = 30 * M_PI / 180;
jointMaxVelc.jointPara[2] = 30 * M_PI / 180;
jointMaxVelc.jointPara[3] = 30 * M_PI / 180;
jointMaxVelc.jointPara[4] = 30 * M_PI / 180;
jointMaxVelc.jointPara[5] = 30 * M_PI / 180;
rs_set_global_joint_maxvelc(rshd, &jointMaxVelc);

//设置末端型运动最大加速度
double lineMaxAcc = 0.2;
rs_set_global_end_max_line_acc(rshd, lineMaxAcc);

//设置末端型运动最大速度
double lineMaxVelc = 0.2;
rs_set_global_end_max_line_velc(rshd, lineMaxVelc);

//设置工具
aubo_robot_namespace::ToolInEndDesc toolEnd;
toolEnd.toolInEndPosition.x = -0.177341;
toolEnd.toolInEndPosition.y = 0.002327;
toolEnd.toolInEndPosition.z = 0.146822;
toolEnd.toolInEndOrientation.w = 1;
toolEnd.toolInEndOrientation.x = 0;
toolEnd.toolInEndOrientation.y = 0;
toolEnd.toolInEndOrientation.z = 0;
rs_set_tool_kinematics_param(rshd, &toolEnd);

//设置偏移
aubo_robot_namespace::MoveRelative relativeOnBase;
relativeOnBase.ena = true;
relativeOnBase.relativePosition[0] = 0;
relativeOnBase.relativePosition[1] = 0;
relativeOnBase.relativePosition[2] = 0;
aubo_robot_namespace::Rpy rpyRelative;
aubo_robot_namespace::Ori oriRelative;
rpyRelative.rx = 10.0 / 180 * M_PI; //要加一位小数点
rpyRelative.ry = 0.0 / 180 * M_PI;
```

```
    rpyRelative.rz = 0.0 / 180 * M_PI;
    rs_rpy_to_quaternion(rshd, &rpyRelative, &oriRelative);
    relativeOnBase.relativeOri = oriRelative;

    //偏移：工具在基坐标系下沿 X 轴姿态偏移 10 度
    rs_set_relative_offset_on_base(rshd, &relativeOnBase);

    //关节角
    double jointAngle[6] = {};
    jointAngle[0] = 173.108713 * M_PI / 180;
    jointAngle[1] = -12.075005 * M_PI / 180;
    jointAngle[2] = -83.663342 * M_PI / 180;
    jointAngle[3] = -15.641249 * M_PI / 180;
    jointAngle[4] = -89.140000 * M_PI / 180;
    jointAngle[5] = -28.328713 * M_PI / 180;

    //关节运动
    rs_move_joint(rshd, jointAngle, true);
}
```

main.cpp 代码如下:

注意：要将下面代码中的 IP 地址（ROBOT_ADDR）修改为对应服务器的 IP 地址。

```
#include "example.h"
#include "windows.h"

#define ROBOT_ADDR "192.168.1.100"
#define ROBOT_PORT 8899

//机械臂控制上下文句柄
RSHD g_rshd = -1;

int main(int argc, char* argv[])
{
    //登录服务器
    if (example_login(g_rshd, ROBOT_ADDR, ROBOT_PORT))
    {
        //机械臂上电(必须连接真实机械臂)
        example_robotStartup(g_rshd);

        //示例 4：工具末端在 Base 坐标系下
        example_move_relarive4(g_rshd);
```

```
    }
    else
    {
        std::cout << "机械臂登录失败。" << std::endl;
    }

    //反初始化接口库
    rs_uninitialize();

    std::cout << "Please enter to exit" << std::endl;
    getchar();

    return 0;
}
```

4.11.2 rs_set_relative_offset_on_user 函数

示例 1：法兰盘中心在用户坐标系下

本示例是机械臂做姿态偏移运动，法兰盘中心在用户坐标系下沿 X 轴姿态偏移

example.h 代码如下：

```
#pragma once
#include <string>
#include "rsdef.h"

//示例 2：法兰盘中心在用户坐标系下
void example_move_relarive2(RSHD rshd);
```

example.cpp 代码如下：

```
#include "example.h"
#include "rsdef.h"

#include <Windows.h>
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <algorithm>
```

```
#define M_PI 3.14159265358979323846

void example_move_relarive2(RSHD rshd)
{
    //初始化运动属性
    rs_init_global_move_profile(rshd);

    //设置关节运动最大加速度
    aubo_robot_namespace::JointVelcAccParam jointMaxAcc;
    jointMaxAcc.jointPara[0] = 30 * M_PI / 180;
    jointMaxAcc.jointPara[1] = 30 * M_PI / 180;
    jointMaxAcc.jointPara[2] = 30 * M_PI / 180;
    jointMaxAcc.jointPara[3] = 30 * M_PI / 180;
    jointMaxAcc.jointPara[4] = 30 * M_PI / 180;
    jointMaxAcc.jointPara[5] = 30 * M_PI / 180;
    rs_set_global_joint_maxacc(rshd, &jointMaxAcc);

    //设置关节运动最大速度
    aubo_robot_namespace::JointVelcAccParam jointMaxVelc;
    jointMaxVelc.jointPara[0] = 30 * M_PI / 180;
    jointMaxVelc.jointPara[1] = 30 * M_PI / 180;
    jointMaxVelc.jointPara[2] = 30 * M_PI / 180;
    jointMaxVelc.jointPara[3] = 30 * M_PI / 180;
    jointMaxVelc.jointPara[4] = 30 * M_PI / 180;
    jointMaxVelc.jointPara[5] = 30 * M_PI / 180;
    rs_set_global_joint_maxvelc(rshd, &jointMaxVelc);

    //设置末端型运动最大加速度
    double lineMaxAcc = 0.2;
    rs_set_global_end_max_line_acc(rshd, lineMaxAcc);

    //设置末端型运动最大速度
    double lineMaxVelc = 0.2;
    rs_set_global_end_max_line_velc(rshd, lineMaxVelc);

    //设置偏移
    aubo_robot_namespace::MoveRelative relativeOnUser;
    relativeOnUser.ena = true;
    relativeOnUser.relativePosition[0] = 0;
    relativeOnUser.relativePosition[1] = 0;
    relativeOnUser.relativePosition[2] = 0;
    aubo_robot_namespace::Rpy rpyRelative;
    aubo_robot_namespace::Ori oriRelative;
```

```
    rpyRelative.rx = 10.0 / 180 * M_PI; //要加一位小数点
    rpyRelative.ry = 0.0 / 180 * M_PI;
    rpyRelative.rz = 0.0 / 180 * M_PI;
    rs_rpy_to_quaternion(rshd, &rpyRelative, &oriRelative);
    relativeOnUser.relativeOri = oriRelative;

    //设置用户坐标系
    aubo_robot_namespace::CoordCalibrateByJointAngleAndTool userCoord;
    userCoord.coordType = aubo_robot_namespace::WorldCoordinate;
    userCoord.methods = aubo_robot_namespace::Origin_AnyPointOnPositiveXAxis_
AnyPointOnFirstQuadrantOfXOYPlane;

    userCoord.wayPointArray[0].jointPos[0] = -75.093279 * M_PI / 180;
    userCoord.wayPointArray[0].jointPos[1] = 28.544643 * M_PI / 180;
    userCoord.wayPointArray[0].jointPos[2] = -114.313905 * M_PI / 180;
    userCoord.wayPointArray[0].jointPos[3] = -62.769247 * M_PI / 180;
    userCoord.wayPointArray[0].jointPos[4] = -87.343517 * M_PI / 180;
    userCoord.wayPointArray[0].jointPos[5] = -27.888262 * M_PI / 180;

    userCoord.wayPointArray[1].jointPos[0] = -89.239837 * M_PI / 180;
    userCoord.wayPointArray[1].jointPos[1] = 23.936171 * M_PI / 180;
    userCoord.wayPointArray[1].jointPos[2] = -122.299277 * M_PI / 180;
    userCoord.wayPointArray[1].jointPos[3] = -65.208902 * M_PI / 180;
    userCoord.wayPointArray[1].jointPos[4] = -85.011123 * M_PI / 180;
    userCoord.wayPointArray[1].jointPos[5] = -41.87417 * M_PI / 180;

    userCoord.wayPointArray[2].jointPos[0] = -77.059212 * M_PI / 180;
    userCoord.wayPointArray[2].jointPos[1] = 35.509518 * M_PI / 180;
    userCoord.wayPointArray[2].jointPos[2] = -101.108547 * M_PI / 180;
    userCoord.wayPointArray[2].jointPos[3] = -56.433133 * M_PI / 180;
    userCoord.wayPointArray[2].jointPos[4] = -87.006734 * M_PI / 180;
    userCoord.wayPointArray[2].jointPos[5] = -29.827440 * M_PI / 180;

    aubo_robot_namespace::ToolInEndDesc toolUserCoord;
    toolUserCoord.toolInEndPosition.x = -0.177341;
    toolUserCoord.toolInEndPosition.y = 0.002327;
    toolUserCoord.toolInEndPosition.z = 0.146822;
    toolUserCoord.toolInEndOrientation.w = 1;
    toolUserCoord.toolInEndOrientation.x = 0;
    toolUserCoord.toolInEndOrientation.y = 0;
    toolUserCoord.toolInEndOrientation.z = 0;
    userCoord.toolDesc = toolUserCoord;

    //偏移：法兰盘中心在用户坐标系下沿 X 轴姿态偏移 10 度
```

```
    rs_set_relative_offset_on_user(rshd, &relativeOnUser, &userCoord);

    //关节角
    double jointAngle[6] = {};
    jointAngle[0] = 173.108713 * M_PI / 180;
    jointAngle[1] = -12.075005 * M_PI / 180;
    jointAngle[2] = -83.663342 * M_PI / 180;
    jointAngle[3] = -15.641249 * M_PI / 180;
    jointAngle[4] = -89.140000 * M_PI / 180;
    jointAngle[5] = -28.328713 * M_PI / 180;

    //关节运动
    rs_move_joint(rshd, jointAngle, true);
}
```

main.cpp 代码如下:

注意：要将下面代码中的 IP 地址（ROBOT_ADDR）修改为对应服务器的 IP 地址。

```
#include "example.h"
#include "windows.h"

#define ROBOT_ADDR "192.168.1.100"
#define ROBOT_PORT 8899

//机械臂控制上下文句柄
RSHD g_rshd = -1;

int main(int argc, char* argv[])
{
    //登录服务器
    if (example_login(g_rshd, ROBOT_ADDR, ROBOT_PORT))
    {
        //机械臂上电(必须连接真实机械臂)
        example_robotStartup(g_rshd);

        //示例 2：法兰盘中心在 User 坐标系下
        example_move_relarive2(g_rshd);
    }
    else
    {
        std::cout << "机械臂登录失败。" << std::endl;
    }
```

```
    //反初始化接口库
    rs_uninitialize();

    std::cout << "Please enter to exit" << std::endl;
    getchar();

    return 0;
}
```

示例 2：工具末端在用户坐标系下

本示例是机械臂做偏移运动，工具在用户坐标系下沿 X 轴姿态偏移

example.h 代码如下：

```
#pragma once
#include <string>
#include "rsdef.h"

//示例 5：工具末端在用户坐标系下
void example_move_relarive5(RSHD rshd);
```

example.cpp 代码如下：

```
#include "example.h"
#include "rsdef.h"

#include <Windows.h>
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <algorithm>


#define M_PI 3.14159265358979323846

//示例 5：工具末端在用户坐标系下
void example_move_relarive5(RSHD rshd)
{
    //初始化运动属性
    rs_init_global_move_profile(rshd);
```

```
//设置关节运动最大加速度
aubo_robot_namespace::JointVelcAccParam jointMaxAcc;
jointMaxAcc.jointPara[0] = 30 * M_PI / 180;
jointMaxAcc.jointPara[1] = 30 * M_PI / 180;
jointMaxAcc.jointPara[2] = 30 * M_PI / 180;
jointMaxAcc.jointPara[3] = 30 * M_PI / 180;
jointMaxAcc.jointPara[4] = 30 * M_PI / 180;
jointMaxAcc.jointPara[5] = 30 * M_PI / 180;
rs_set_global_joint_maxacc(rshd, &jointMaxAcc);

//设置关节运动最大速度
aubo_robot_namespace::JointVelcAccParam jointMaxVelc;
jointMaxVelc.jointPara[0] = 30 * M_PI / 180;
jointMaxVelc.jointPara[1] = 30 * M_PI / 180;
jointMaxVelc.jointPara[2] = 30 * M_PI / 180;
jointMaxVelc.jointPara[3] = 30 * M_PI / 180;
jointMaxVelc.jointPara[4] = 30 * M_PI / 180;
jointMaxVelc.jointPara[5] = 30 * M_PI / 180;
rs_set_global_joint_maxvelc(rshd, &jointMaxVelc);

//设置末端型运动最大加速度
double lineMaxAcc = 0.2;
rs_set_global_end_max_line_acc(rshd, lineMaxAcc);

//设置末端型运动最大速度
double lineMaxVelc = 0.2;
rs_set_global_end_max_line_velc(rshd, lineMaxVelc);

//设置工具
aubo_robot_namespace::ToolInEndDesc toolEnd;
toolEnd.toolInEndPosition.x = -0.177341;
toolEnd.toolInEndPosition.y = 0.002327;
toolEnd.toolInEndPosition.z = 0.146822;
toolEnd.toolInEndOrientation.w = 1;
toolEnd.toolInEndOrientation.x = 0;
toolEnd.toolInEndOrientation.y = 0;
toolEnd.toolInEndOrientation.z = 0;
rs_set_tool_kinematics_param(rshd, &toolEnd);

//设置偏移
aubo_robot_namespace::MoveRelative relativeOnUser;
relativeOnUser.ena = true;
relativeOnUser.relativePosition[0] = 0;
```

```
    relativeOnUser.relativePosition[1] = 0;
    relativeOnUser.relativePosition[2] = 0;
    aubo_robot_namespace::Rpy rpyRelative;
    aubo_robot_namespace::Ori oriRelative;
    rpyRelative.rx = 10.0 / 180 * M_PI; //要加一位小数点
    rpyRelative.ry = 0.0 / 180 * M_PI;
    rpyRelative.rz = 0.0 / 180 * M_PI;
    rs_rpy_to_quaternion(rshd, &rpyRelative, &oriRelative);
    relativeOnUser.relativeOri = oriRelative;

    //设置用户坐标系
    aubo_robot_namespace::CoordCalibrateByJointAngleAndTool userCoord;
    userCoord.coordType = aubo_robot_namespace::WorldCoordinate;
    userCoord.methods = aubo_robot_namespace::Origin_AnyPointOnPositiveXAxis_
AnyPointOnFirstQuadrantOfXOYPlane;

    userCoord.wayPointArray[0].jointPos[0] = -75.093279 * M_PI / 180;
    userCoord.wayPointArray[0].jointPos[1] = 28.544643 * M_PI / 180;
    userCoord.wayPointArray[0].jointPos[2] = -114.313905 * M_PI / 180;
    userCoord.wayPointArray[0].jointPos[3] = -62.769247 * M_PI / 180;
    userCoord.wayPointArray[0].jointPos[4] = -87.343517 * M_PI / 180;
    userCoord.wayPointArray[0].jointPos[5] = -27.888262 * M_PI / 180;

    userCoord.wayPointArray[1].jointPos[0] = -89.239837 * M_PI / 180;
    userCoord.wayPointArray[1].jointPos[1] = 23.936171 * M_PI / 180;
    userCoord.wayPointArray[1].jointPos[2] = -122.299277 * M_PI / 180;
    userCoord.wayPointArray[1].jointPos[3] = -65.208902 * M_PI / 180;
    userCoord.wayPointArray[1].jointPos[4] = -85.011123 * M_PI / 180;
    userCoord.wayPointArray[1].jointPos[5] = -41.87417 * M_PI / 180;

    userCoord.wayPointArray[2].jointPos[0] = -77.059212 * M_PI / 180;
    userCoord.wayPointArray[2].jointPos[1] = 35.509518 * M_PI / 180;
    userCoord.wayPointArray[2].jointPos[2] = -101.108547 * M_PI / 180;
    userCoord.wayPointArray[2].jointPos[3] = -56.433133 * M_PI / 180;
    userCoord.wayPointArray[2].jointPos[4] = -87.006734 * M_PI / 180;
    userCoord.wayPointArray[2].jointPos[5] = -29.827440 * M_PI / 180;

    aubo_robot_namespace::ToolInEndDesc toolUserCoord;
    toolUserCoord.toolInEndPosition.x = -0.177341;
    toolUserCoord.toolInEndPosition.y = 0.002327;
    toolUserCoord.toolInEndPosition.z = 0.146822;
    toolUserCoord.toolInEndOrientation.w = 1;
    toolUserCoord.toolInEndOrientation.x = 0;
    toolUserCoord.toolInEndOrientation.y = 0;
```

```
    toolUserCoord.toolInEndOrientation.z = 0;
    userCoord.toolDesc = toolUserCoord;

    //偏移：工具末端在用户坐标系下沿 X 轴姿态偏移 10 度
    rs_set_relative_offset_on_user(rshd, &relativeOnUser, &userCoord);

    //关节角
    double jointAngle[6] = {};
    jointAngle[0] = 173.108713 * M_PI / 180;
    jointAngle[1] = -12.075005 * M_PI / 180;
    jointAngle[2] = -83.663342 * M_PI / 180;
    jointAngle[3] = -15.641249 * M_PI / 180;
    jointAngle[4] = -89.140000 * M_PI / 180;
    jointAngle[5] = -28.328713 * M_PI / 180;

    //关节运动
    rs_move_joint(rshd, jointAngle, true);

}
```

main.cpp 代码如下:

注意：要将下面代码中的 IP 地址（ROBOT_ADDR）修改为对应服务器的 IP 地址。

```
#include "example.h"
#include "windows.h"

#define ROBOT_ADDR "192.168.1.100"
#define ROBOT_PORT 8899

//机械臂控制上下文句柄
RSHD g_rshd = -1;

int main(int argc, char* argv[])
{
    //登录服务器
    if (example_login(g_rshd, ROBOT_ADDR, ROBOT_PORT))
    {
        //机械臂上电(必须连接真实机械臂)
        example_robotStartup(g_rshd);

        //示例 5：工具末端在 User 坐标系下
        example_move_relarive5(g_rshd);

    }
```

```
    else
    {
        std::cout << "机械臂登录失败。" << std::endl;
    }

    //反初始化接口库
    rs_uninitialize();

    std::cout << "Please enter to exit" << std::endl;
    getchar();

    return 0;
}
```

示例 3：工具末端在工具坐标系下

本示例是机械臂做姿态偏移运动，工具在工具坐标系下沿 X 轴姿态偏移

example.h 代码如下：

```
#pragma once
#include <string>
#include "rsdef.h"

//示例 3：工具末端在工具坐标系下
void example_move_relarive3(RSHD rshd);
```

example.cpp 代码如下：

```
#include "example.h"
#include "rsdef.h"

#include <Windows.h>
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <algorithm>



#define M_PI 3.14159265358979323846
```

```
//示例 3：工具末端在工具坐标系下
void example_move_relarive3(RSHD rshd)
{
    //初始化运动属性
    rs_init_global_move_profile(rshd);

    //设置关节运动最大加速度
    aubo_robot_namespace::JointVelcAccParam jointMaxAcc;
    jointMaxAcc.jointPara[0] = 30 * M_PI / 180;
    jointMaxAcc.jointPara[1] = 30 * M_PI / 180;
    jointMaxAcc.jointPara[2] = 30 * M_PI / 180;
    jointMaxAcc.jointPara[3] = 30 * M_PI / 180;
    jointMaxAcc.jointPara[4] = 30 * M_PI / 180;
    jointMaxAcc.jointPara[5] = 30 * M_PI / 180;
    rs_set_global_joint_maxacc(rshd, &jointMaxAcc);

    //设置关节运动最大速度
    aubo_robot_namespace::JointVelcAccParam jointMaxVelc;
    jointMaxVelc.jointPara[0] = 30 * M_PI / 180;
    jointMaxVelc.jointPara[1] = 30 * M_PI / 180;
    jointMaxVelc.jointPara[2] = 30 * M_PI / 180;
    jointMaxVelc.jointPara[3] = 30 * M_PI / 180;
    jointMaxVelc.jointPara[4] = 30 * M_PI / 180;
    jointMaxVelc.jointPara[5] = 30 * M_PI / 180;
    rs_set_global_joint_maxvelc(rshd, &jointMaxVelc);

    //设置末端型运动最大加速度
    double lineMaxAcc = 0.2;
    rs_set_global_end_max_line_acc(rshd, lineMaxAcc);

    //设置末端型运动最大速度
    double lineMaxVelc = 0.2;
    rs_set_global_end_max_line_velc(rshd, lineMaxVelc);

    //设置偏移
    aubo_robot_namespace::MoveRelative relativeOnEnd;
    relativeOnEnd.ena = true;
    relativeOnEnd.relativePosition[0] = 0;
    relativeOnEnd.relativePosition[1] = 0;
    relativeOnEnd.relativePosition[2] = 0;
    aubo_robot_namespace::Rpy rpyRelative;
    aubo_robot_namespace::Ori oriRelative;
    rpyRelative.rx = 10.0 / 180 * M_PI; //要加一位小数点
    rpyRelative.ry = 0.0 / 180 * M_PI;
```

```
    rpyRelative.rz = 0.0 / 180 * M_PI;
    rs_rpy_to_quaternion(rshd, &rpyRelative, &oriRelative);
    relativeOnEnd.relativeOri = oriRelative;

    //设置工具
    aubo_robot_namespace::ToolInEndDesc toolEnd;
    toolEnd.toolInEndPosition.x = -0.177341;
    toolEnd.toolInEndPosition.y = 0.002327;
    toolEnd.toolInEndPosition.z = 0.146822;
    toolEnd.toolInEndOrientation.w = 1;
    toolEnd.toolInEndOrientation.x = 0;
    toolEnd.toolInEndOrientation.y = 0;
    toolEnd.toolInEndOrientation.z = 0;
    rs_set_tool_kinematics_param(rshd, &toolEnd);

    //设置工具坐标系
    aubo_robot_namespace::CoordCalibrateByJointAngleAndTool endCoord;
    endCoord.coordType = aubo_robot_namespace::EndCoordinate;
    endCoord.toolDesc = toolEnd;
    //偏移：工具在工具坐标系下沿 X 轴姿态偏移 10 度
    rs_set_relative_offset_on_user(rshd, &relativeOnEnd, &endCoord);

    //关节角
    double jointAngle[6] = {};
    jointAngle[0] = 173.108713 * M_PI / 180;
    jointAngle[1] = -12.075005 * M_PI / 180;
    jointAngle[2] = -83.663342 * M_PI / 180;
    jointAngle[3] = -15.641249 * M_PI / 180;
    jointAngle[4] = -89.140000 * M_PI / 180;
    jointAngle[5] = -28.328713 * M_PI / 180;

    //关节运动
    rs_move_joint(rshd, jointAngle, true);
}
```

main.cpp 代码如下:

注意：要将下面代码中的 IP 地址（ROBOT_ADDR）修改为对应服务器的 IP 地址。

```
#include "example.h"
#include "windows.h"

#define ROBOT_ADDR "192.168.1.100"
```

```
#define ROBOT_PORT 8899

//机械臂控制上下文句柄
RSHD g_rshd = -1;

int main(int argc, char* argv[])
{
    //登录服务器
    if (example_login(g_rshd, ROBOT_ADDR, ROBOT_PORT))
    {
        //机械臂上电(必须连接真实机械臂)
        example_robotStartup(g_rshd);

        //示例 3：工具末端在 Tool 坐标系下
        example_move_relarive3(g_rshd);
    }
    else
    {
        std::cout << "机械臂登录失败。" << std::endl;
    }

    //反初始化接口库
    rs_uninitialize();

    std::cout << "Please enter to exit" << std::endl;
    getchar();

    return 0;
}
```

4.12 旋转运动

4.12.1 rs_move_rotate 函数

示例 1：法兰盘中心在基坐标系下

本示例是法兰盘中心在用户坐标系下沿 X+方向旋转，当前位置保持不变。

example.h 代码如下：

```
#pragma once
#include <string>
#include "rsdef.h"

//示例 1：法兰盘中心在基坐标系下
void example_MoveRotate1(RSHD rshd);
```

example.cpp 代码如下：

```
#include "example.h"
#include "rsdef.h"

#include <Windows.h>
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <algorithm>



#define M_PI 3.14159265358979323846

//示例 1：法兰盘中心在基坐标系下
void example_MoveRotate1(RSHD rshd)
{
    //初始化运动属性
    rs_init_global_move_profile(rshd);

    //设置关节运动最大加速度
    aubo_robot_namespace::JointVelcAccParam jointMaxAcc;
    jointMaxAcc.jointPara[0] = 30 * M_PI / 180;
    jointMaxAcc.jointPara[1] = 30 * M_PI / 180;
    jointMaxAcc.jointPara[2] = 30 * M_PI / 180;
```

```
    jointMaxAcc.jointPara[3] = 30 * M_PI / 180;
    jointMaxAcc.jointPara[4] = 30 * M_PI / 180;
    jointMaxAcc.jointPara[5] = 30 * M_PI / 180;
    rs_set_global_joint_maxacc(rshd, &jointMaxAcc);

    //设置关节运动最大速度
    aubo_robot_namespace::JointVelcAccParam jointMaxVelc;
    jointMaxVelc.jointPara[0] = 30 * M_PI / 180;
    jointMaxVelc.jointPara[1] = 30 * M_PI / 180;
    jointMaxVelc.jointPara[2] = 30 * M_PI / 180;
    jointMaxVelc.jointPara[3] = 30 * M_PI / 180;
    jointMaxVelc.jointPara[4] = 30 * M_PI / 180;
    jointMaxVelc.jointPara[5] = 30 * M_PI / 180;
    rs_set_global_joint_maxvelc(rshd, &jointMaxVelc);

    //起始路点
    double initPos[6] = {
        173.108713 / 180 * M_PI,
        -12.075005 / 180 * M_PI,
        -83.663342 / 180 * M_PI,
        -15.641249 / 180 * M_PI,
        -89.140000 / 180 * M_PI,
        -28.328713 / 180 * M_PI
    };

    //关节运动到起始路点
    rs_move_joint(rshd, initPos, true);

    //设置基坐标系
    aubo_robot_namespace::CoordCalibrateByJointAngleAndTool baseCoord;
    baseCoord.coordType = aubo_robot_namespace::BaseCoordinate;

    //旋转轴和旋转角度
    Move_Rotate_Axis axis;
    axis.rotateAxis[0] = 1;
    axis.rotateAxis[1] = 0;
    axis.rotateAxis[2] = 0;
    double rotateAngle = 5.0 * M_PI / 180;

    //旋转运动：法兰盘中心在基坐标系下沿 X+方向旋转 5 度，当前位置保持不变
    rs_move_rotate(rshd, &baseCoord, &axis, rotateAngle, true);

}
```

main.cpp 代码如下:

注意：要将下面代码中的 IP 地址（ROBOT_ADDR）修改为对应服务器的 IP 地址。

```
#include "example.h"
#include "windows.h"

#define ROBOT_ADDR "192.168.1.100"
#define ROBOT_PORT 8899

//机械臂控制上下文句柄
RSHD g_rshd = -1;

int main(int argc, char* argv[])
{
    //登录服务器
    if (example_login(g_rshd, ROBOT_ADDR, ROBOT_PORT))
    {
        //机械臂上电(必须连接真实机械臂)
        example_robotStartup(g_rshd);

        //示例 1：法兰盘中心在基坐标系下
        example_MoveRotate1(g_rshd);
    }
    else
    {
        std::cout << "机械臂登录失败。" << std::endl;
    }

    //反初始化接口库
    rs_uninitialize();

    std::cout << "Please enter to exit" << std::endl;
    getchar();

    return 0;
}
```

示例 2：法兰盘中心在用户坐标系下

本示例是法兰盘中心在用户坐标系下沿 X+方向旋转，当前位置保持不变。

example.h 代码如下：

```
#pragma once
#include <string>
#include "rsdef.h"

//示例 2：法兰盘中心在用户坐标系下
void example_MoveRotate2(RSHD rshd);
```

example.cpp 代码如下：

```
#include "example.h"
#include "rsdef.h"

#include <Windows.h>
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <algorithm>



#define M_PI 3.14159265358979323846

//示例 2：法兰盘中心在用户坐标系下
void example_MoveRotate2(RSHD rshd)
{
    //初始化运动属性
    rs_init_global_move_profile(rshd);

    //设置关节运动最大加速度
    aubo_robot_namespace::JointVelcAccParam jointMaxAcc;
    jointMaxAcc.jointPara[0] = 30 * M_PI / 180;
    jointMaxAcc.jointPara[1] = 30 * M_PI / 180;
    jointMaxAcc.jointPara[2] = 30 * M_PI / 180;
    jointMaxAcc.jointPara[3] = 30 * M_PI / 180;
    jointMaxAcc.jointPara[4] = 30 * M_PI / 180;
    jointMaxAcc.jointPara[5] = 30 * M_PI / 180;
    rs_set_global_joint_maxacc(rshd, &jointMaxAcc);
```

```
    //设置关节运动最大速度
    aubo_robot_namespace::JointVelcAccParam  jointMaxVelc;
    jointMaxVelc.jointPara[0] = 30 * M_PI / 180;
    jointMaxVelc.jointPara[1] = 30 * M_PI / 180;
    jointMaxVelc.jointPara[2] = 30 * M_PI / 180;
    jointMaxVelc.jointPara[3] = 30 * M_PI / 180;
    jointMaxVelc.jointPara[4] = 30 * M_PI / 180;
    jointMaxVelc.jointPara[5] = 30 * M_PI / 180;
    rs_set_global_joint_maxvelc(rshd, &jointMaxVelc);

    //起始路点
    double initPos[6] = {
        173.108713 / 180 * M_PI,
        -12.075005 / 180 * M_PI,
        -83.663342 / 180 * M_PI,
        -15.641249 / 180 * M_PI,
        -89.140000 / 180 * M_PI,
        -28.328713 / 180 * M_PI
    };

    //关节运动到起始路点
    rs_move_joint(rshd, initPos, true);

    //设置用户坐标系
    aubo_robot_namespace::CoordCalibrateByJointAngleAndTool userCoord;
    userCoord.coordType = aubo_robot_namespace::WorldCoordinate;
    userCoord.methods = aubo_robot_namespace::Origin_AnyPointOnPositiveXAxis_
AnyPointOnFirstQuadrantOfXOYPlane;

    userCoord.wayPointArray[0].jointPos[0] = -75.093279 * M_PI / 180;
    userCoord.wayPointArray[0].jointPos[1] = 28.544643 * M_PI / 180;
    userCoord.wayPointArray[0].jointPos[2] = -114.313905 * M_PI / 180;
    userCoord.wayPointArray[0].jointPos[3] = -62.769247 * M_PI / 180;
    userCoord.wayPointArray[0].jointPos[4] = -87.343517 * M_PI / 180;
    userCoord.wayPointArray[0].jointPos[5] = -27.888262 * M_PI / 180;

    userCoord.wayPointArray[1].jointPos[0] = -89.239837 * M_PI / 180;
    userCoord.wayPointArray[1].jointPos[1] = 23.936171 * M_PI / 180;
    userCoord.wayPointArray[1].jointPos[2] = -122.299277 * M_PI / 180;
    userCoord.wayPointArray[1].jointPos[3] = -65.208902 * M_PI / 180;
    userCoord.wayPointArray[1].jointPos[4] = -85.011123 * M_PI / 180;
    userCoord.wayPointArray[1].jointPos[5] = -41.87417 * M_PI / 180;

    userCoord.wayPointArray[2].jointPos[0] = -77.059212 * M_PI / 180;
```

```
    userCoord.wayPointArray[2].jointPos[1] = 35.509518 * M_PI / 180;
    userCoord.wayPointArray[2].jointPos[2] = -101.108547 * M_PI / 180;
    userCoord.wayPointArray[2].jointPos[3] = -56.433133 * M_PI / 180;
    userCoord.wayPointArray[2].jointPos[4] = -87.006734 * M_PI / 180;
    userCoord.wayPointArray[2].jointPos[5] = -29.827440 * M_PI / 180;

    aubo_robot_namespace::ToolInEndDesc toolUserCoord;
    toolUserCoord.toolInEndPosition.x = -0.177341;
    toolUserCoord.toolInEndPosition.y = 0.002327;
    toolUserCoord.toolInEndPosition.z = 0.146822;
    toolUserCoord.toolInEndOrientation.w = 1;
    toolUserCoord.toolInEndOrientation.x = 0;
    toolUserCoord.toolInEndOrientation.y = 0;
    toolUserCoord.toolInEndOrientation.z = 0;
    userCoord.toolDesc = toolUserCoord;

    //旋转轴和旋转角度
    Move_Rotate_Axis axis;
    axis.rotateAxis[0] = 1;
    axis.rotateAxis[1] = 0;
    axis.rotateAxis[2] = 0;
    double rotateAngle = 10.0 * M_PI / 180;

    //旋转运动：法兰盘中心在用户坐标系下沿 X+方向旋转 10 度，当前位置保持
不变
    rs_move_rotate(rshd,&userCoord, &axis, rotateAngle, true);
}
```

main.cpp 代码如下:

注意：要将下面代码中的 IP 地址（ROBOT_ADDR）修改为对应服务器的 IP 地址。

```
#include "example.h"
#include "windows.h"

#define ROBOT_ADDR "192.168.1.100"
#define ROBOT_PORT 8899

//机械臂控制上下文句柄
RSHD g_rshd = -1;

int main(int argc, char* argv[])
{
```

```
    //登录服务器
    if (example_login(g_rshd, ROBOT_ADDR, ROBOT_PORT))
    {
        //机械臂上电(必须连接真实机械臂)
        example_robotStartup(g_rshd);

        //示例 2：法兰盘中心在用户坐标系下
        example_MoveRotate2(g_rshd);
    }
    else
    {
        std::cout << "机械臂登录失败。" << std::endl;
    }

    //反初始化接口库
    rs_uninitialize();

    std::cout << "Please enter to exit" << std::endl;
    getchar();

    return 0;
}
```

示例 3：工具末端在工具坐标系下

本示例是工具末端在工具坐标系下沿 X+方向旋转，当前位置保持不变。

example.h 代码如下：

```
#pragma once
#include <string>
#include "rsdef.h"

//示例 3：工具末端在工具坐标系下
void example_MoveRotate3(RSHD rshd);
```

example.cpp 代码如下：

```
#include "example.h"
#include "rsdef.h"

#include <Windows.h>
```

```
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <algorithm>


#define M_PI 3.14159265358979323846

//示例 3：工具末端在工具坐标系下
void example_MoveRotate3(RSHD rshd)
{
    //初始化运动属性
    rs_init_global_move_profile(rshd);

    //设置关节运动最大加速度
    aubo_robot_namespace::JointVelcAccParam jointMaxAcc;
    jointMaxAcc.jointPara[0] = 30 * M_PI / 180;
    jointMaxAcc.jointPara[1] = 30 * M_PI / 180;
    jointMaxAcc.jointPara[2] = 30 * M_PI / 180;
    jointMaxAcc.jointPara[3] = 30 * M_PI / 180;
    jointMaxAcc.jointPara[4] = 30 * M_PI / 180;
    jointMaxAcc.jointPara[5] = 30 * M_PI / 180;
    rs_set_global_joint_maxacc(rshd, &jointMaxAcc);

    //设置关节运动最大速度
    aubo_robot_namespace::JointVelcAccParam jointMaxVelc;
    jointMaxVelc.jointPara[0] = 30 * M_PI / 180;
    jointMaxVelc.jointPara[1] = 30 * M_PI / 180;
    jointMaxVelc.jointPara[2] = 30 * M_PI / 180;
    jointMaxVelc.jointPara[3] = 30 * M_PI / 180;
    jointMaxVelc.jointPara[4] = 30 * M_PI / 180;
    jointMaxVelc.jointPara[5] = 30 * M_PI / 180;
    rs_set_global_joint_maxvelc(rshd, &jointMaxVelc);

    //起始路点
    double initPos[6] = {
        173.108713 / 180 * M_PI,
        -12.075005 / 180 * M_PI,
        -83.663342 / 180 * M_PI,
        -15.641249 / 180 * M_PI,
        -89.140000 / 180 * M_PI,
        -28.328713 / 180 * M_PI
    };
```

```
    //关节运动到起始路点
    rs_move_joint(rshd, initPos, true);

    //设置工具
    aubo_robot_namespace::ToolInEndDesc toolEnd;
    toolEnd.toolInEndPosition.x = -0.177341;
    toolEnd.toolInEndPosition.y = 0.002327;
    toolEnd.toolInEndPosition.z = 0.146822;
    toolEnd.toolInEndOrientation.w = 1;
    toolEnd.toolInEndOrientation.x = 0;
    toolEnd.toolInEndOrientation.y = 0;
    toolEnd.toolInEndOrientation.z = 0;
    rs_set_tool_kinematics_param(rshd, &toolEnd);

    //设置工具坐标系
    aubo_robot_namespace::CoordCalibrateByJointAngleAndTool endCoord;
    endCoord.coordType = aubo_robot_namespace::EndCoordinate;
    endCoord.toolDesc = toolEnd;

    //旋转轴和旋转角度
    Move_Rotate_Axis axis;
    axis.rotateAxis[0] = 0;
    axis.rotateAxis[1] = 0;
    axis.rotateAxis[2] = 1;
    double rotateAngle = 5.0 * M_PI / 180;

    //旋转运动：工具末端在工具坐标系下沿 Z+方向旋转 5 度，当前位置保持不变
    rs_move_rotate(rshd, &endCoord, &axis, rotateAngle, true);

}
```

main.cpp 代码如下:

注意：要将下面代码中的 IP 地址（ROBOT_ADDR）修改为对应服务器的 IP 地址。

```
#include "example.h"
#include "windows.h"

#define ROBOT_ADDR "192.168.1.100"
#define ROBOT_PORT 8899

//机械臂控制上下文句柄
```

```
RSHD g_rshd = -1;

int main(int argc, char* argv[])
{
    //登录服务器
    if (example_login(g_rshd, ROBOT_ADDR, ROBOT_PORT))
    {
        //机械臂上电(必须连接真实机械臂)
        example_robotStartup(g_rshd);

        //示例 3：工具末端在工具坐标系下
        example_MoveRotate3(g_rshd);
    }
    else
    {
        std::cout << "机械臂登录失败。" << std::endl;
    }

    //反初始化接口库
    rs_uninitialize();

    std::cout << "Please enter to exit" << std::endl;
    getchar();

    return 0;
}
```

示例 4：工具末端在基坐标系下

本示例是工具末端在基坐标系下沿 X+方向旋转，当前位置保持不变。

example.h 代码如下：

```
#pragma once
#include <string>
#include "rsdef.h"

//示例 4：工具末端在基坐标系下
void example_MoveRotate4(RSHD rshd);
```

example.cpp 代码如下：

```
#include "example.h"
#include "rsdef.h"

#include <Windows.h>
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <algorithm>


#define M_PI 3.14159265358979323846

//示例 4：工具末端在基坐标系下
void example_MoveRotate4(RSHD rshd)
{
    //初始化运动属性
    rs_init_global_move_profile(rshd);

    //设置关节运动最大加速度
    aubo_robot_namespace::JointVelcAccParam jointMaxAcc;
    jointMaxAcc.jointPara[0] = 30 * M_PI / 180;
    jointMaxAcc.jointPara[1] = 30 * M_PI / 180;
    jointMaxAcc.jointPara[2] = 30 * M_PI / 180;
    jointMaxAcc.jointPara[3] = 30 * M_PI / 180;
    jointMaxAcc.jointPara[4] = 30 * M_PI / 180;
    jointMaxAcc.jointPara[5] = 30 * M_PI / 180;
    rs_set_global_joint_maxacc(rshd, &jointMaxAcc);

    //设置关节运动最大速度
    aubo_robot_namespace::JointVelcAccParam jointMaxVelc;
    jointMaxVelc.jointPara[0] = 30 * M_PI / 180;
    jointMaxVelc.jointPara[1] = 30 * M_PI / 180;
    jointMaxVelc.jointPara[2] = 30 * M_PI / 180;
    jointMaxVelc.jointPara[3] = 30 * M_PI / 180;
    jointMaxVelc.jointPara[4] = 30 * M_PI / 180;
    jointMaxVelc.jointPara[5] = 30 * M_PI / 180;
    rs_set_global_joint_maxvelc(rshd, &jointMaxVelc);

    //起始路点
    double initPos[6] = {
        173.108713 / 180 * M_PI,
        -12.075005 / 180 * M_PI,
        -83.663342 / 180 * M_PI,
```

```
        -15.641249 / 180 * M_PI,
        -89.140000 / 180 * M_PI,
        -28.328713 / 180 * M_PI
    };

    //关节运动到起始路点
    rs_move_joint(rshd, initPos, true);

    //设置工具
    aubo_robot_namespace::ToolInEndDesc toolEnd;
    toolEnd.toolInEndPosition.x = -0.177341;
    toolEnd.toolInEndPosition.y = 0.002327;
    toolEnd.toolInEndPosition.z = 0.146822;
    toolEnd.toolInEndOrientation.w = 1;
    toolEnd.toolInEndOrientation.x = 0;
    toolEnd.toolInEndOrientation.y = 0;
    toolEnd.toolInEndOrientation.z = 0;
    rs_set_tool_kinematics_param(rshd, &toolEnd);

    //设置基坐标系
    aubo_robot_namespace::CoordCalibrateByJointAngleAndTool baseCoord;
    baseCoord.coordType = aubo_robot_namespace::BaseCoordinate;

    //旋转轴和旋转角度
    Move_Rotate_Axis axis;
    axis.rotateAxis[0] = 1;
    axis.rotateAxis[1] = 0;
    axis.rotateAxis[2] = 0;
    double rotateAngle = 10.0 * M_PI / 180;

    //旋转运动：工具末端在基坐标系下沿 X+方向旋转 10 度，当前位置保持不变
    rs_move_rotate(rshd, &baseCoord, &axis, rotateAngle, true);
}
```

main.cpp 代码如下:

注意：要将下面代码中的 IP 地址（ROBOT_ADDR）修改为对应服务器的 IP 地址。

```
#include "example.h"
#include "windows.h"

#define ROBOT_ADDR "192.168.1.100"
#define ROBOT_PORT 8899
```

```
//机械臂控制上下文句柄
RSHD g_rshd = -1;

int main(int argc, char* argv[])
{
    //登录服务器
    if (example_login(g_rshd, ROBOT_ADDR, ROBOT_PORT))
    {
        //机械臂上电(必须连接真实机械臂)
        example_robotStartup(g_rshd);

        //示例 4：工具末端在基坐标系下
        example_MoveRotate4(g_rshd);
    }
    else
    {
        std::cout << "机械臂登录失败。" << std::endl;
    }

    //反初始化接口库
    rs_uninitialize();

    std::cout << "Please enter to exit" << std::endl;
    getchar();

    return 0;
}
```

示例 5：工具末端在用户坐标系下

本示例是工具末端在用户坐标系下沿 X+方向旋转，当前位置保持不变。

example.h 代码如下：

```
#pragma once
#include <string>
#include "rsdef.h"

//示例 5：工具末端在用户坐标系下
void example_MoveRotate5(RSHD rshd);
```

example.cpp 代码如下：

```
#include "example.h"
#include "rsdef.h"

#include <Windows.h>
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <algorithm>



#define M_PI 3.14159265358979323846

//示例 5：工具末端在用户坐标系下
void example_MoveRotate5(RSHD rshd)
{
    //初始化运动属性
    rs_init_global_move_profile(rshd);

    //设置关节运动最大加速度
    aubo_robot_namespace::JointVelcAccParam jointMaxAcc;
    jointMaxAcc.jointPara[0] = 30 * M_PI / 180;
    jointMaxAcc.jointPara[1] = 30 * M_PI / 180;
    jointMaxAcc.jointPara[2] = 30 * M_PI / 180;
    jointMaxAcc.jointPara[3] = 30 * M_PI / 180;
    jointMaxAcc.jointPara[4] = 30 * M_PI / 180;
    jointMaxAcc.jointPara[5] = 30 * M_PI / 180;
    rs_set_global_joint_maxacc(rshd, &jointMaxAcc);
```

```
    //设置关节运动最大速度
    aubo_robot_namespace::JointVelcAccParam  jointMaxVelc;
    jointMaxVelc.jointPara[0] = 30 * M_PI / 180;
    jointMaxVelc.jointPara[1] = 30 * M_PI / 180;
    jointMaxVelc.jointPara[2] = 30 * M_PI / 180;
    jointMaxVelc.jointPara[3] = 30 * M_PI / 180;
    jointMaxVelc.jointPara[4] = 30 * M_PI / 180;
    jointMaxVelc.jointPara[5] = 30 * M_PI / 180;
    rs_set_global_joint_maxvelc(rshd, &jointMaxVelc);

    //起始路点
    double initPos[6] = {
        173.108713 / 180 * M_PI,
        -12.075005 / 180 * M_PI,
        -83.663342 / 180 * M_PI,
        -15.641249 / 180 * M_PI,
        -89.140000 / 180 * M_PI,
        -28.328713 / 180 * M_PI
    };

    //关节运动到起始路点
    rs_move_joint(rshd, initPos, true);

    //设置工具
    aubo_robot_namespace::ToolInEndDesc toolEnd;
    toolEnd.toolInEndPosition.x = -0.177341;
    toolEnd.toolInEndPosition.y = 0.002327;
    toolEnd.toolInEndPosition.z = 0.146822;
    toolEnd.toolInEndOrientation.w = 1;
    toolEnd.toolInEndOrientation.x = 0;
    toolEnd.toolInEndOrientation.y = 0;
    toolEnd.toolInEndOrientation.z = 0;
    rs_set_tool_kinematics_param(rshd, &toolEnd);

    //设置用户坐标系
    aubo_robot_namespace::CoordCalibrateByJointAngleAndTool userCoord;
    userCoord.coordType = aubo_robot_namespace::WorldCoordinate;
    userCoord.methods = aubo_robot_namespace::Origin_AnyPointOnPositiveXAxis_
AnyPointOnFirstQuadrantOfXOYPlane;

    userCoord.wayPointArray[0].jointPos[0] = -75.093279 * M_PI / 180;
    userCoord.wayPointArray[0].jointPos[1] = 28.544643 * M_PI / 180;
    userCoord.wayPointArray[0].jointPos[2] = -114.313905 * M_PI / 180;
    userCoord.wayPointArray[0].jointPos[3] = -62.769247 * M_PI / 180;
```

```
    userCoord.wayPointArray[0].jointPos[4] = -87.343517 * M_PI / 180;
    userCoord.wayPointArray[0].jointPos[5] = -27.888262 * M_PI / 180;

    userCoord.wayPointArray[1].jointPos[0] = -89.239837 * M_PI / 180;
    userCoord.wayPointArray[1].jointPos[1] = 23.936171 * M_PI / 180;
    userCoord.wayPointArray[1].jointPos[2] = -122.299277 * M_PI / 180;
    userCoord.wayPointArray[1].jointPos[3] = -65.208902 * M_PI / 180;
    userCoord.wayPointArray[1].jointPos[4] = -85.011123 * M_PI / 180;
    userCoord.wayPointArray[1].jointPos[5] = -41.87417 * M_PI / 180;

    userCoord.wayPointArray[2].jointPos[0] = -77.059212 * M_PI / 180;
    userCoord.wayPointArray[2].jointPos[1] = 35.509518 * M_PI / 180;
    userCoord.wayPointArray[2].jointPos[2] = -101.108547 * M_PI / 180;
    userCoord.wayPointArray[2].jointPos[3] = -56.433133 * M_PI / 180;
    userCoord.wayPointArray[2].jointPos[4] = -87.006734 * M_PI / 180;
    userCoord.wayPointArray[2].jointPos[5] = -29.827440 * M_PI / 180;

    userCoord.toolDesc = toolEnd;

    //旋转轴和旋转角度
    Move_Rotate_Axis axis;
    axis.rotateAxis[0] = 1;
    axis.rotateAxis[1] = 0;
    axis.rotateAxis[2] = 0;
    double rotateAngle = 10.0 * M_PI / 180;

    //旋转运动：工具末端在用户坐标系下沿 X+方向旋转 10 度，当前位置保持不
变
    rs_move_rotate(rshd, &userCoord, &axis, rotateAngle, true);
}
```

main.cpp 代码如下:

注意：要将下面代码中的 IP 地址（ROBOT_ADDR）修改为对应服务器的 IP 地址。

```
#include "example.h"
#include "windows.h"

#define ROBOT_ADDR "192.168.1.100"
#define ROBOT_PORT 8899

//机械臂控制上下文句柄
```

```
RSHD g_rshd = -1;

int main(int argc, char* argv[])
{
    //登录服务器
    if (example_login(g_rshd, ROBOT_ADDR, ROBOT_PORT))
    {
        //机械臂上电(必须连接真实机械臂)
        example_robotStartup(g_rshd);

        //示例 5：工具末端在用户坐标系下
        example_MoveRotate5(g_rshd);

    }
    else
    {
        std::cout << "机械臂登录失败。" << std::endl;
    }

    //反初始化接口库
    rs_uninitialize();

    std::cout << "Please enter to exit" << std::endl;
    getchar();

    return 0;
}
```

4.13 轨迹运动

4.13.1 rs_move_track 函数

示例 1：圆运动

本示例是机械臂做轨迹运动之圆运动。

example.h 代码如下:

```
#pragma once
#include <string>
#include "rsdef.h"

//轨迹运动 rs_move_track
//示例 1：圆运动
void example_MoveTrack1(RSHD rshd);
```

example.cpp 代码如下:

```
#include "example.h"
#include "rsdef.h"

#include <Windows.h>
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <algorithm>



#define M_PI 3.14159265358979323846

//轨迹运动 rs_move_track
//示例 1：圆运动
void example_MoveTrack1(RSHD rshd)
{
    //初始化运动属性
    rs_init_global_move_profile(rshd);

    //设置关节型运动的最大加速度
    aubo_robot_namespace::JointVelcAccParam jointMaxAcc;
    jointMaxAcc.jointPara[0] = 50.0 / 180.0 * M_PI;
```

```
    jointMaxAcc.jointPara[1] = 50.0 / 180.0 * M_PI;
    jointMaxAcc.jointPara[2] = 50.0 / 180.0 * M_PI;
    jointMaxAcc.jointPara[3] = 50.0 / 180.0 * M_PI;
    jointMaxAcc.jointPara[4] = 50.0 / 180.0 * M_PI;
    jointMaxAcc.jointPara[5] = 50.0 / 180.0 * M_PI;          //接口要求单位是弧
度
    rs_set_global_joint_maxacc(rshd, &jointMaxAcc);

    //设置关节型运动的最大速度
    aubo_robot_namespace::JointVelcAccParam jointMaxVelc;
    jointMaxVelc.jointPara[0] = 50.0 / 180.0 * M_PI;
    jointMaxVelc.jointPara[1] = 50.0 / 180.0 * M_PI;
    jointMaxVelc.jointPara[2] = 50.0 / 180.0 * M_PI;
    jointMaxVelc.jointPara[3] = 50.0 / 180.0 * M_PI;
    jointMaxVelc.jointPara[4] = 50.0 / 180.0 * M_PI;
    jointMaxVelc.jointPara[5] = 50.0 / 180.0 * M_PI;    //接口要求单位是弧度
    rs_set_global_joint_maxvelc(rshd, &jointMaxVelc);

    //设置末端型运动的最大加速度
    double endMoveMaxAcc;
    endMoveMaxAcc = 0.2; //单位米每秒
    rs_set_global_end_max_line_acc(rshd, endMoveMaxAcc);
    rs_set_global_end_max_angle_acc(rshd, endMoveMaxAcc);

    //设置末端型运动的最大速度
    double endMoveMaxVelc;
    endMoveMaxVelc = 0.2; //单位米每秒
    rs_set_global_end_max_line_velc(rshd, endMoveMaxVelc);
    rs_set_global_end_max_angle_velc(rshd, endMoveMaxVelc);

    //准备点
    double jointAngle[aubo_robot_namespace::ARM_DOF] = { 0 };
    jointAngle[0] = -0.000003;
    jointAngle[1] = -0.127267;
    jointAngle[2] = -1.321122;
    jointAngle[3] = 0.376934;
    jointAngle[4] = -1.570796;
    jointAngle[5] = -0.000008;

    //关节运动到准备点
    int ret = rs_move_joint(rshd, jointAngle);
    if (ret != RS_SUCC)
    {
        std::cerr << "关节运动到准备点失败。      错误号: " << ret << std::end
```

```
l;
    }

    //添加圆轨迹路点
    jointAngle[0] = -0.000003;
    jointAngle[1] = -0.127267;
    jointAngle[2] = -1.321122;
    jointAngle[3] = 0.376934;
    jointAngle[4] = -1.570796;
    jointAngle[5] = -0.000008;
    rs_add_waypoint(rshd, jointAngle);

    jointAngle[0] = -0.211675;
    jointAngle[1] = -0.325189;
    jointAngle[2] = -1.466753;
    jointAngle[3] = 0.429232;
    jointAngle[4] = -1.570794;
    jointAngle[5] = -0.211680;
    rs_add_waypoint(rshd, jointAngle);

    jointAngle[0] = -0.037186;
    jointAngle[1] = -0.224307;
    jointAngle[2] = -1.398285;
    jointAngle[3] = 0.396819;
    jointAngle[4] = -1.570796;
    jointAngle[5] = -0.037191;
    rs_add_waypoint(rshd, jointAngle);

    //设置圆的圈数
    rs_set_circular_loop_times(rshd, 3);

    //开始轨迹运动
    ret = rs_move_track(rshd, ARC_CIR);
    if (RS_SUCC != ret)
    {
        std::cerr << "圆轨迹运动失败。错误号:" << ret << std::endl;
    }
    else
    {
        std::cout << "圆轨迹运动成功。" << std::endl;
    }
}
```

main.cpp 代码如下:

注意：要将下面代码中的 IP 地址（ROBOT_ADDR）修改为对应服务器的 IP 地址。

```
#include "example.h"
#include "windows.h"

#define ROBOT_ADDR "192.168.1.100"
#define ROBOT_PORT 8899

//机械臂控制上下文句柄
RSHD g_rshd = -1;

int main(int argc, char* argv[])
{
    //登录服务器
    if (example_login(g_rshd, ROBOT_ADDR, ROBOT_PORT))
    {
        //机械臂上电(必须连接真实机械臂)
        example_robotStartup(g_rshd);

        //轨迹运动 rs_move_track
        //示例 1：圆运动
        example_MoveTrack1(g_rshd);
    }
    else
    {
        std::cout << "机械臂登录失败。" << std::endl;
    }

    //反初始化接口库
    rs_uninitialize();

    std::cout << "Please enter to exit" << std::endl;
    getchar();

    return 0;
}
```

示例 2：圆弧运动

本示例是机械臂做轨迹运动之圆弧运动。

example.h 代码如下:

```
#pragma once
#include <string>
#include "rsdef.h"

//示例 2：圆弧运动
void example_MoveTrack2(RSHD rshd);
```

example.cpp 代码如下:

```
#include "example.h"
#include "rsdef.h"

#include <Windows.h>
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <algorithm>



#define M_PI 3.14159265358979323846

//示例 2：圆弧运动
void example_MoveTrack2(RSHD rshd)
{
    //初始化运动属性
    rs_init_global_move_profile(rshd);

    //设置关节型运动的最大加速度
    aubo_robot_namespace::JointVelcAccParam jointMaxAcc;
    jointMaxAcc.jointPara[0] = 50.0 / 180.0 * M_PI;
    jointMaxAcc.jointPara[1] = 50.0 / 180.0 * M_PI;
    jointMaxAcc.jointPara[2] = 50.0 / 180.0 * M_PI;
    jointMaxAcc.jointPara[3] = 50.0 / 180.0 * M_PI;
    jointMaxAcc.jointPara[4] = 50.0 / 180.0 * M_PI;
    jointMaxAcc.jointPara[5] = 50.0 / 180.0 * M_PI;          //接口要求单位是弧
度
    rs_set_global_joint_maxacc(rshd, &jointMaxAcc);

    //设置关节型运动的最大速度
```

```
    aubo_robot_namespace::JointVelcAccParam jointMaxVelc;
    jointMaxVelc.jointPara[0] = 50.0 / 180.0 * M_PI;
    jointMaxVelc.jointPara[1] = 50.0 / 180.0 * M_PI;
    jointMaxVelc.jointPara[2] = 50.0 / 180.0 * M_PI;
    jointMaxVelc.jointPara[3] = 50.0 / 180.0 * M_PI;
    jointMaxVelc.jointPara[4] = 50.0 / 180.0 * M_PI;
    jointMaxVelc.jointPara[5] = 50.0 / 180.0 * M_PI;    //接口要求单位是弧度
    rs_set_global_joint_maxvelc(rshd, &jointMaxVelc);

    //设置末端型运动的最大加速度
    double endMoveMaxAcc;
    endMoveMaxAcc = 0.2; //单位米每秒
    rs_set_global_end_max_line_acc(rshd, endMoveMaxAcc);
    rs_set_global_end_max_angle_acc(rshd, endMoveMaxAcc);

    //设置末端型运动的最大速度
    double endMoveMaxVelc;
    endMoveMaxVelc = 0.2; //单位米每秒
    rs_set_global_end_max_line_velc(rshd, endMoveMaxVelc);
    rs_set_global_end_max_angle_velc(rshd, endMoveMaxVelc);

    //准备点
    double jointAngle[aubo_robot_namespace::ARM_DOF] = { 0 };
    jointAngle[0] = -0.000003;
    jointAngle[1] = -0.127267;
    jointAngle[2] = -1.321122;
    jointAngle[3] = 0.376934;
    jointAngle[4] = -1.570796;
    jointAngle[5] = -0.000008;

    //关节运动到准备点
    int ret = rs_move_joint(rshd, jointAngle);
    if (ret != RS_SUCC)
    {
        std::cerr << "关节运动到准备点失败。        错误号: " << ret << std::end
l;
    }

    //添加圆弧轨迹路点
    jointAngle[0] = -0.000003;
    jointAngle[1] = -0.127267;
    jointAngle[2] = -1.321122;
    jointAngle[3] = 0.376934;
    jointAngle[4] = -1.570796;
```

```
    jointAngle[5] = -0.000008;
    rs_add_waypoint(rshd, jointAngle);

    jointAngle[0] = 0.200000;
    jointAngle[1] = -0.127267;
    jointAngle[2] = -1.321122;
    jointAngle[3] = 0.376934;
    jointAngle[4] = -1.570794;
    jointAngle[5] = -0.000008;
    rs_add_waypoint(rshd, jointAngle);

    jointAngle[0] = 0.600000;
    jointAngle[1] = -0.127267;
    jointAngle[2] = -1.321122;
    jointAngle[3] = 0.376934;
    jointAngle[4] = -1.570796;
    jointAngle[5] = -0.000008;
    rs_add_waypoint(rshd, jointAngle);

    //设置圆弧
    rs_set_circular_loop_times(rshd, 0);

    //开始轨迹运动
    ret = rs_move_track(rshd, ARC_CIR);
    if (RS_SUCC != ret)
    {
        std::cerr << "圆弧轨迹运动失败。  错误号:" << ret << std::endl;
    }
    else
    {
        std::cout << "圆弧轨迹运动成功。" << std::endl;
    }
}
```

main.cpp 代码如下:

注意：要将下面代码中的 IP 地址（ROBOT_ADDR）修改为对应服务器的 IP 地址。

```
#include "example.h"
#include "windows.h"

#define ROBOT_ADDR "192.168.1.100"
#define ROBOT_PORT 8899
```

```
//机械臂控制上下文句柄
RSHD g_rshd = -1;

int main(int argc, char* argv[])
{
    //登录服务器
    if (example_login(g_rshd, ROBOT_ADDR, ROBOT_PORT))
    {
        //机械臂上电(必须连接真实机械臂)
        example_robotStartup(g_rshd);

        //示例 2：圆弧运动
        example_MoveTrack2(g_rshd);
    }
    else
    {
        std::cout << "机械臂登录失败。" << std::endl;
    }

    //反初始化接口库
    rs_uninitialize();

    std::cout << "Please enter to exit" << std::endl;
    getchar();

    return 0;
}
```

示例 3：MOVEP

本示例是机械臂做轨迹运动之 MOVEP 运动。

example.h 代码如下:

```
#pragma  once
#include  <string>
#include  "rsdef.h"

//示例 3：MOVEP 运动
void  example_MoveTrack3(RSHD  rshd);
```

example.cpp 代码如下:

```
#include  "example.h"
#include  "rsdef.h"

#include  <Windows.h>
#include  <iostream>
#include  <fstream>
#include  <sstream>
#include  <string>
#include  <algorithm>



#define  M_PI  3.14159265358979323846

//示例 3：MOVEP 运动
void  example_MoveTrack3(RSHD  rshd)
{
    //初始化运动属性
    rs_init_global_move_profile(rshd);

    //设置关节型运动的最大加速度
    aubo_robot_namespace::JointVelcAccParam  jointMaxAcc;
    jointMaxAcc.jointPara[0]  =  50.0  /  180.0  *  M_PI;
    jointMaxAcc.jointPara[1]  =  50.0  /  180.0  *  M_PI;
    jointMaxAcc.jointPara[2]  =  50.0  /  180.0  *  M_PI;
    jointMaxAcc.jointPara[3]  =  50.0  /  180.0  *  M_PI;
    jointMaxAcc.jointPara[4]  =  50.0  /  180.0  *  M_PI;
    jointMaxAcc.jointPara[5]  =  50.0  /  180.0  *  M_PI;          //接口要求单位是弧
度
    rs_set_global_joint_maxacc(rshd,  &jointMaxAcc);
```

```
    //设置关节型运动的最大速度
    aubo_robot_namespace::JointVelcAccParam jointMaxVelc;
    jointMaxVelc.jointPara[0] = 50.0 / 180.0 * M_PI;
    jointMaxVelc.jointPara[1] = 50.0 / 180.0 * M_PI;
    jointMaxVelc.jointPara[2] = 50.0 / 180.0 * M_PI;
    jointMaxVelc.jointPara[3] = 50.0 / 180.0 * M_PI;
    jointMaxVelc.jointPara[4] = 50.0 / 180.0 * M_PI;
    jointMaxVelc.jointPara[5] = 50.0 / 180.0 * M_PI;    //接口要求单位是弧度
    rs_set_global_joint_maxvelc(rshd, &jointMaxVelc);

    //设置末端型运动的最大加速度
    double endMoveMaxAcc;
    endMoveMaxAcc = 0.2; //单位米每秒
    rs_set_global_end_max_line_acc(rshd, endMoveMaxAcc);
    rs_set_global_end_max_angle_acc(rshd, endMoveMaxAcc);

    //设置末端型运动的最大速度
    double endMoveMaxVelc;
    endMoveMaxVelc = 0.2; //单位米每秒
    rs_set_global_end_max_line_velc(rshd, endMoveMaxVelc);
    rs_set_global_end_max_angle_velc(rshd, endMoveMaxVelc);

    //准备点
    double jointAngle[aubo_robot_namespace::ARM_DOF] = { 0 };
    jointAngle[0] = -0.000003;
    jointAngle[1] = -0.127267;
    jointAngle[2] = -1.321122;
    jointAngle[3] = 0.376934;
    jointAngle[4] = -1.570796;
    jointAngle[5] = -0.000008;

    //关节运动到准备点
    int ret = rs_move_joint(rshd, jointAngle);
    if (ret != RS_SUCC)
    {
        std::cerr << "关节运动到准备点失败。      错误号: " << ret << std::end
l;
    }

    //添加 MOVEP 轨迹路点
    jointAngle[0] = -0.000003;
    jointAngle[1] = -0.127267;
    jointAngle[2] = -1.321122;
```

```
    jointAngle[3] = 0.376934;
    jointAngle[4] = -1.570796;
    jointAngle[5] = -0.000008;
    rs_add_waypoint(rshd, jointAngle);

    jointAngle[0] = 0.200000;
    jointAngle[1] = -0.127267;
    jointAngle[2] = -1.321122;
    jointAngle[3] = 0.376934;
    jointAngle[4] = -1.570794;
    jointAngle[5] = -0.000008;
    rs_add_waypoint(rshd, jointAngle);

    jointAngle[0] = 0.600000;
    jointAngle[1] = -0.127267;
    jointAngle[2] = -1.321122;
    jointAngle[3] = 0.376934;
    jointAngle[4] = -1.570796;
    jointAngle[5] = -0.000008;
    rs_add_waypoint(rshd, jointAngle);

    //设置交融半径
    rs_set_blend_radius(rshd, 0.03);

    //开始轨迹运动
    ret = rs_move_track(rshd, CARTESIAN_MOVEP);
    if (RS_SUCC != ret)
    {
        std::cerr << "MOVEP 轨迹运动失败。错误号:" << ret << std::endl;
    }
    else
    {
        std::cout << "MOVEP 轨迹运动成功。" << std::endl;
    }
}
```

main.cpp 代码如下:

注意：要将下面代码中的 IP 地址（ROBOT_ADDR）修改为对应服务器的 IP 地址。

```
#include "example.h"
#include "windows.h"
```

```
#define ROBOT_ADDR "192.168.1.100"
#define ROBOT_PORT 8899

//机械臂控制上下文句柄
RSHD g_rshd = -1;

int main(int argc, char* argv[])
{
    //登录服务器
    if (example_login(g_rshd, ROBOT_ADDR, ROBOT_PORT))
    {
        //机械臂上电(必须连接真实机械臂)
        example_robotStartup(g_rshd);

        //示例 3：MOVEP 运动
        example_MoveTrack3(g_rshd);
    }
    else
    {
        std::cout << "机械臂登录失败。" << std::endl;
    }

    //反初始化接口库
    rs_uninitialize();

    std::cout << "Please enter to exit" << std::endl;
    getchar();

    return 0;
}
```

4.14 示教运动

本示例是法兰盘中心在基坐标系下做关节示教。

example.h 代码如下:

```
#pragma once
#include <string>
#include "rsdef.h"

//示教运动
//示例：法兰盘中心在基坐标系下
void example_TeachMove(RSHD rshd);
```

example.cpp 代码如下:

```
#include "example.h"
#include "rsdef.h"

#include <Windows.h>
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <algorithm>

//示教运动
//示例：法兰盘中心在基坐标系下
void example_TeachMove(RSHD rshd)
{
    //初始化运动属性
    rs_init_global_move_profile(rshd);

    //设置关节运动最大加速度
    aubo_robot_namespace::JointVelcAccParam jointMaxAcc;
    jointMaxAcc.jointPara[0] = 30 * M_PI / 180;
    jointMaxAcc.jointPara[1] = 30 * M_PI / 180;
    jointMaxAcc.jointPara[2] = 30 * M_PI / 180;
    jointMaxAcc.jointPara[3] = 30 * M_PI / 180;
    jointMaxAcc.jointPara[4] = 30 * M_PI / 180;
    jointMaxAcc.jointPara[5] = 30 * M_PI / 180;
    rs_set_global_joint_maxacc(rshd, &jointMaxAcc);

    //设置关节运动最大速度
```

```
    aubo_robot_namespace::JointVelcAccParam jointMaxVelc;
    jointMaxVelc.jointPara[0] = 30 * M_PI / 180;
    jointMaxVelc.jointPara[1] = 30 * M_PI / 180;
    jointMaxVelc.jointPara[2] = 30 * M_PI / 180;
    jointMaxVelc.jointPara[3] = 30 * M_PI / 180;
    jointMaxVelc.jointPara[4] = 30 * M_PI / 180;
    jointMaxVelc.jointPara[5] = 30 * M_PI / 180;
    rs_set_global_joint_maxvelc(rshd, &jointMaxVelc);

    //设置末端型运动最大加速度
    double lineMaxAcc = 0.2;
    rs_set_global_end_max_line_acc(rshd, lineMaxAcc);

    //设置末端型运动最大速度
    double lineMaxVelc = 0.2;
    rs_set_global_end_max_line_velc(rshd, lineMaxVelc);

    //设置示教坐标系为基坐标系
    aubo_robot_namespace::CoordCalibrateByJointAngleAndTool baseCoord;
    rs_set_teach_coord(rshd, &baseCoord);

    //设置示教模式——关节示教
    aubo_robot_namespace::teach_mode teachmode1;
    teachmode1 = aubo_robot_namespace::JOINT1;

    //开始示教
    rs_teach_move_start(rshd, teachmode1, true);
    Sleep(2000);

    //结束示教
    rs_teach_move_stop(rshd);
}
```

main.cpp 代码如下:

注意：要将下面代码中的 IP 地址（ROBOT_ADDR）修改为对应服务器的 IP 地址。

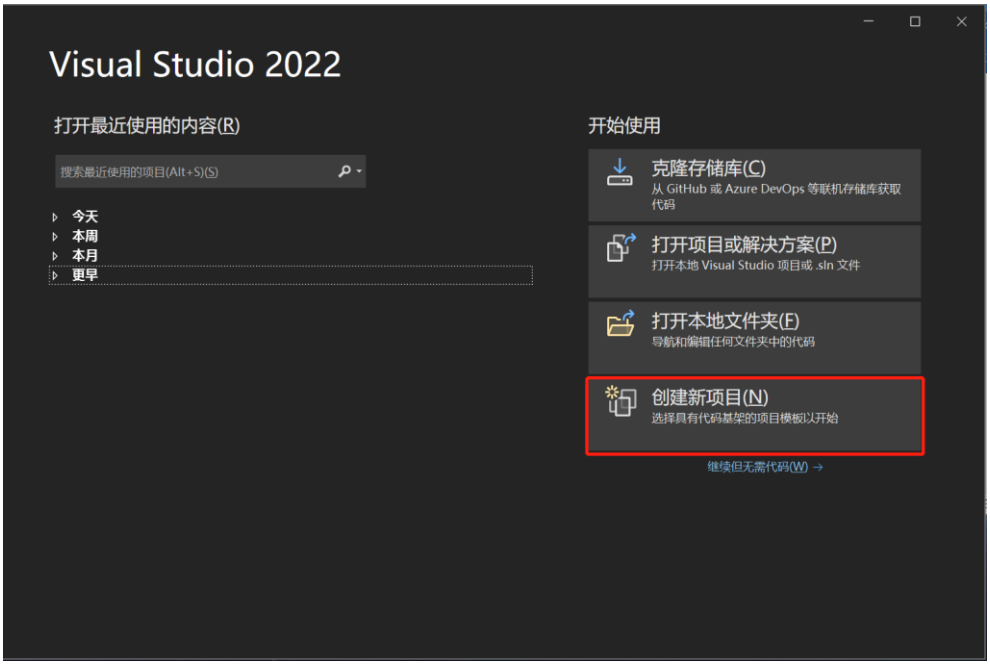
```
#include "example.h"
#include "windows.h"

#define ROBOT_ADDR "192.168.1.100"
#define ROBOT_PORT 8899
```

```
//机械臂控制上下文句柄
RSHD g_rshd = -1;

int main(int argc, char* argv[])
{
    //登录服务器
    if (example_login(g_rshd, ROBOT_ADDR, ROBOT_PORT))
    {
        //机械臂上电(必须连接真实机械臂)
        example_robotStartup(g_rshd);

        //示教运动
        //示例：法兰盘中心在基坐标系下
        example_TeachMove(g_rshd);
    }
    else
    {
        std::cout << "机械臂登录失败。" << std::endl;
    }

    //反初始化接口库
    rs_uninitialize();

    std::cout << "Please enter to exit" << std::endl;
    getchar();

    return 0;
}
```

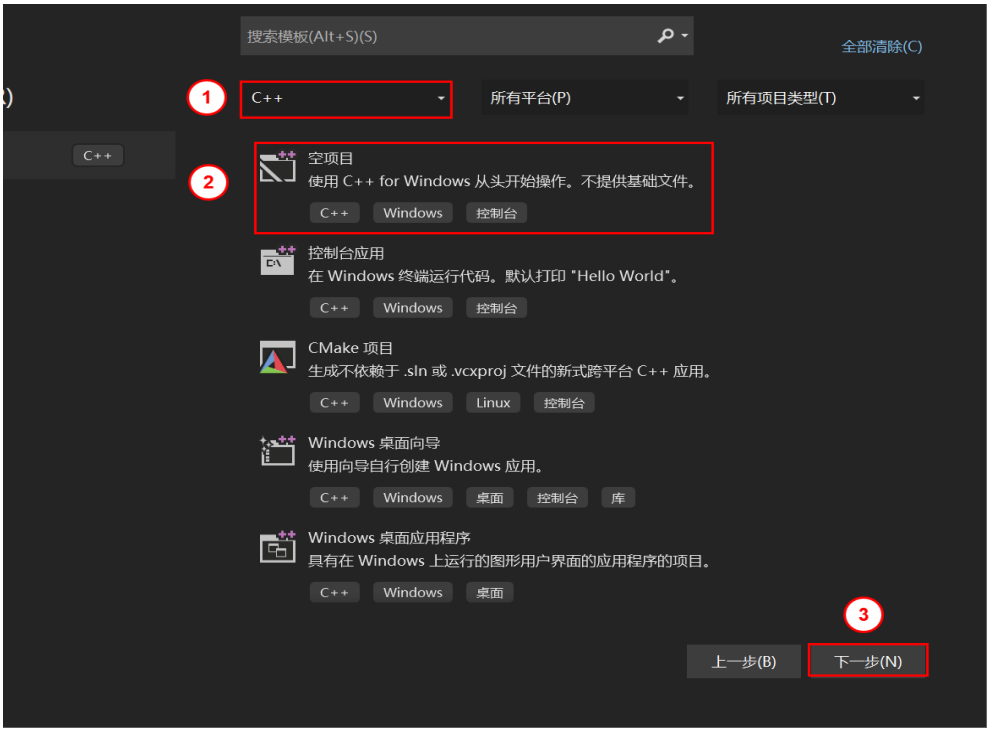
5 环境配置说明

5.1 创建新的程序

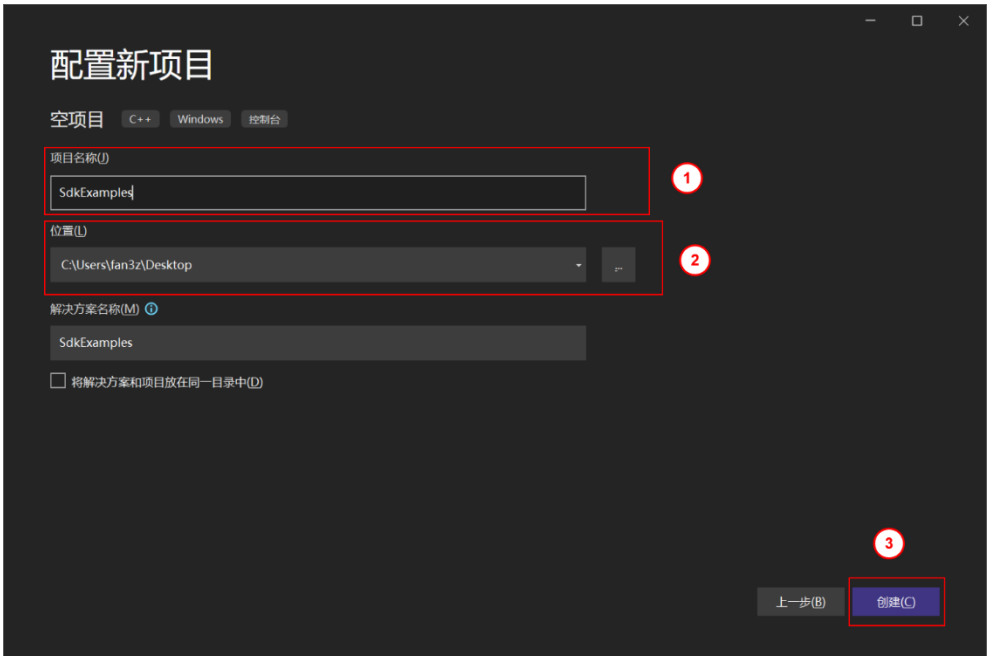
1. 打开 Visual Studio 2022，点击“创建新项目”。



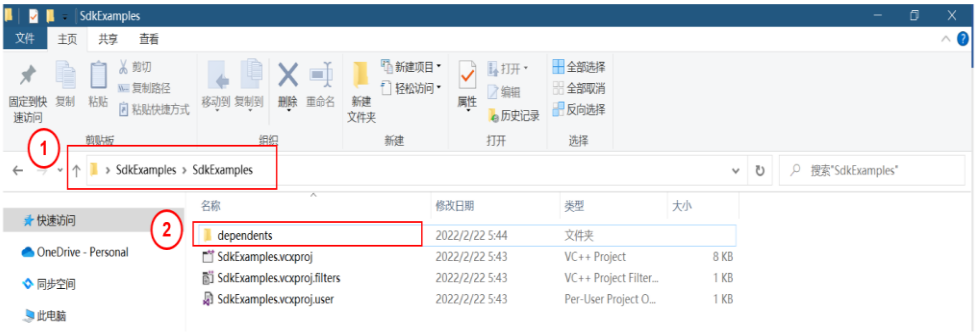
2. 选择“C++”→点击“空项目”→点击“下一步”。



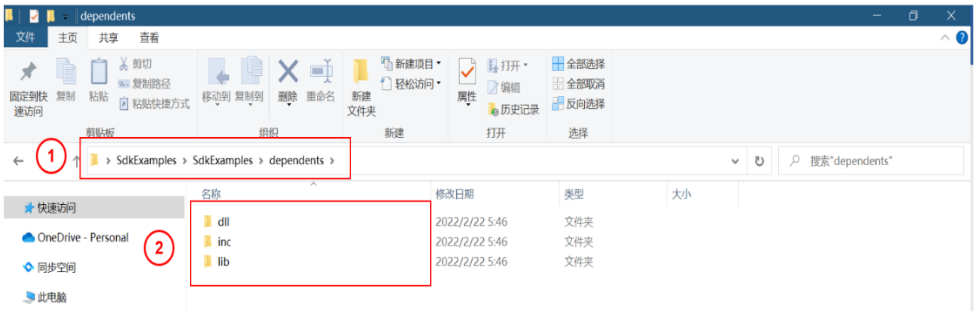
3. 输入项目名称，选择保存位置，点击“创建”。



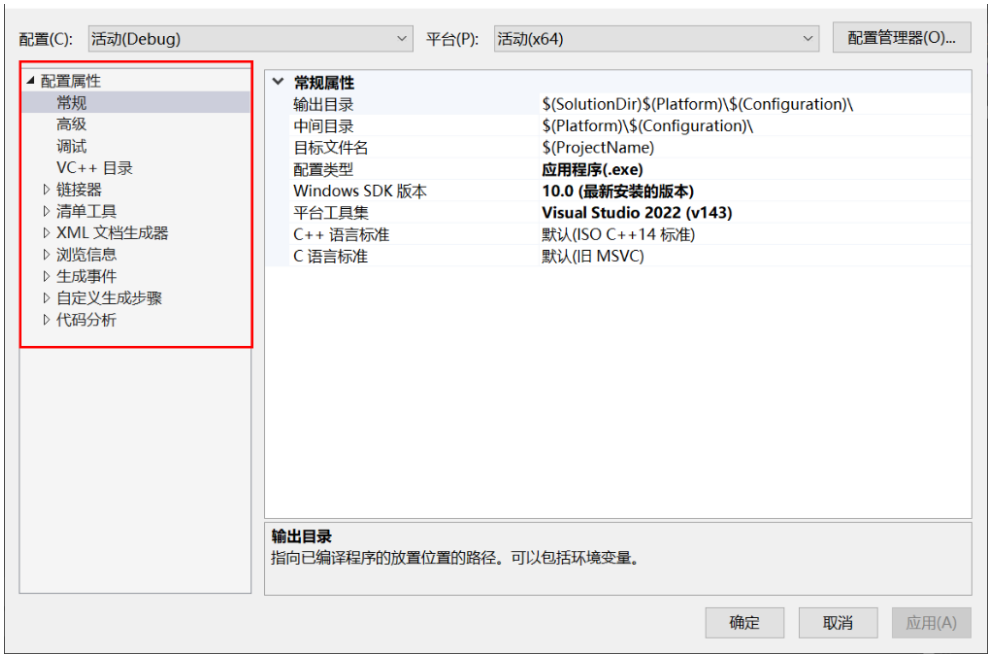
4. 打开项目所在的文件夹，新建文件夹，命名为 dependents，用来存放库文件等依赖。



5. 打开 dependents 文件夹，创建 3 个文件夹，分别命名为 dll、inc、lib，用来存放 dll 文件、头文件和 lib 库文件。

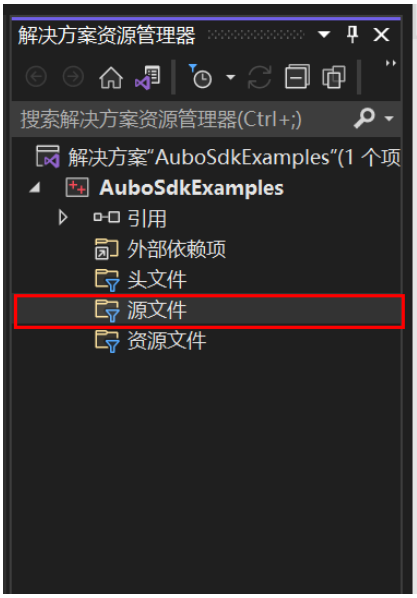


6. 在 dll、inc、lib 三个文件夹中，添加完相应的文件后，回到 Visual Studio 2020。在菜单栏“项目”，点击“属性”。

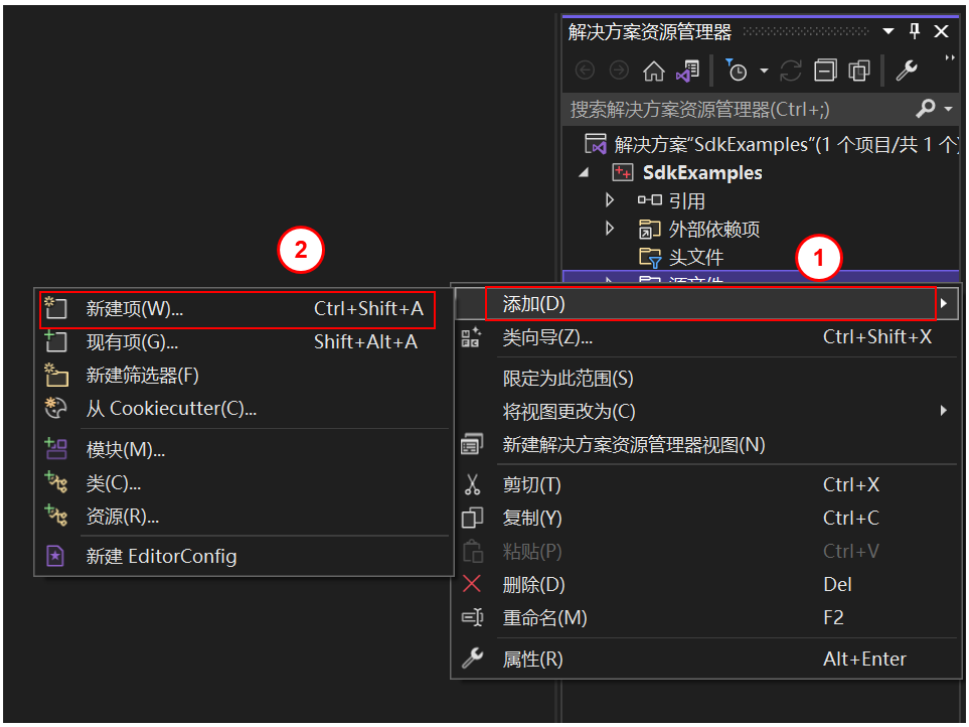


7. 配置属性里没有“C/C++”项，解决办法：（1）写一些代码之后，再编译，“C/C++”就会在配置属性里出现了。（2）新建一个.cpp 文件

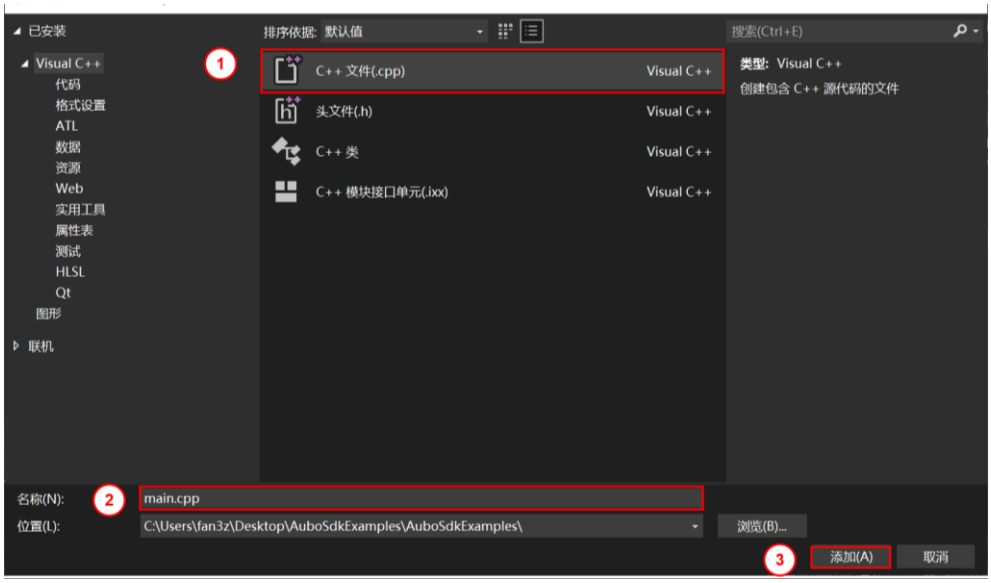
在“解决方案管理器”中，选中“源文件”，鼠标右键。



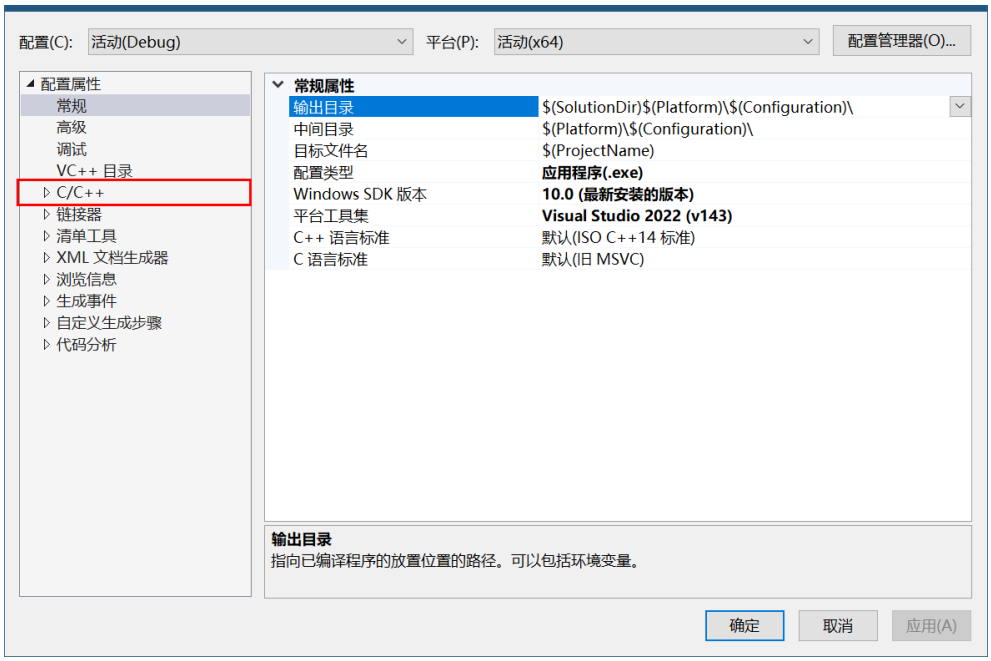
单击“添加”→“新建项”。



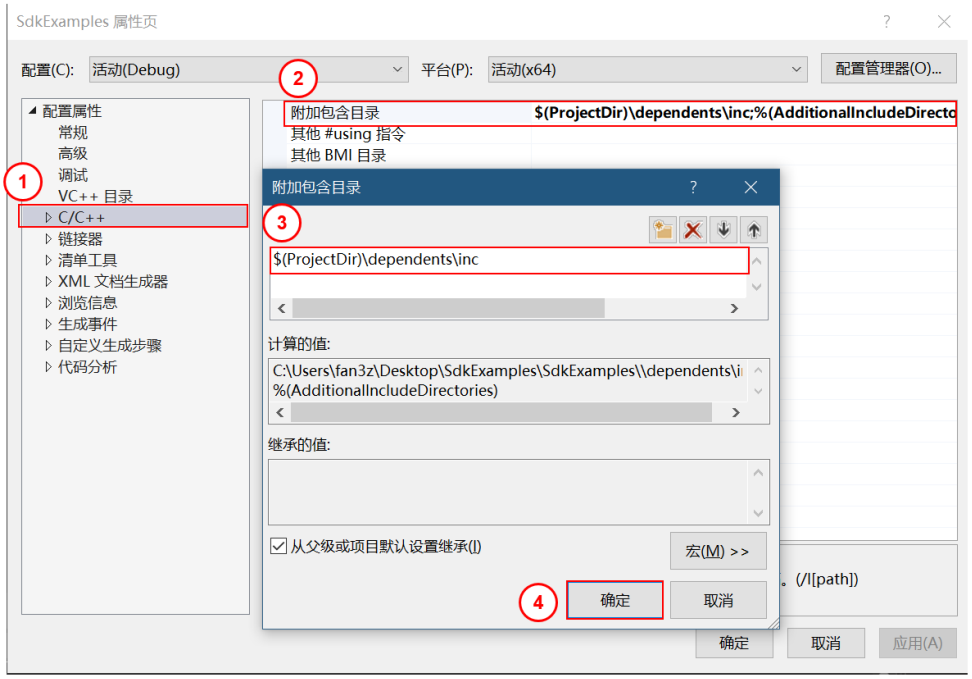
选中“C++文件”，给文件命名，单击“添加”。



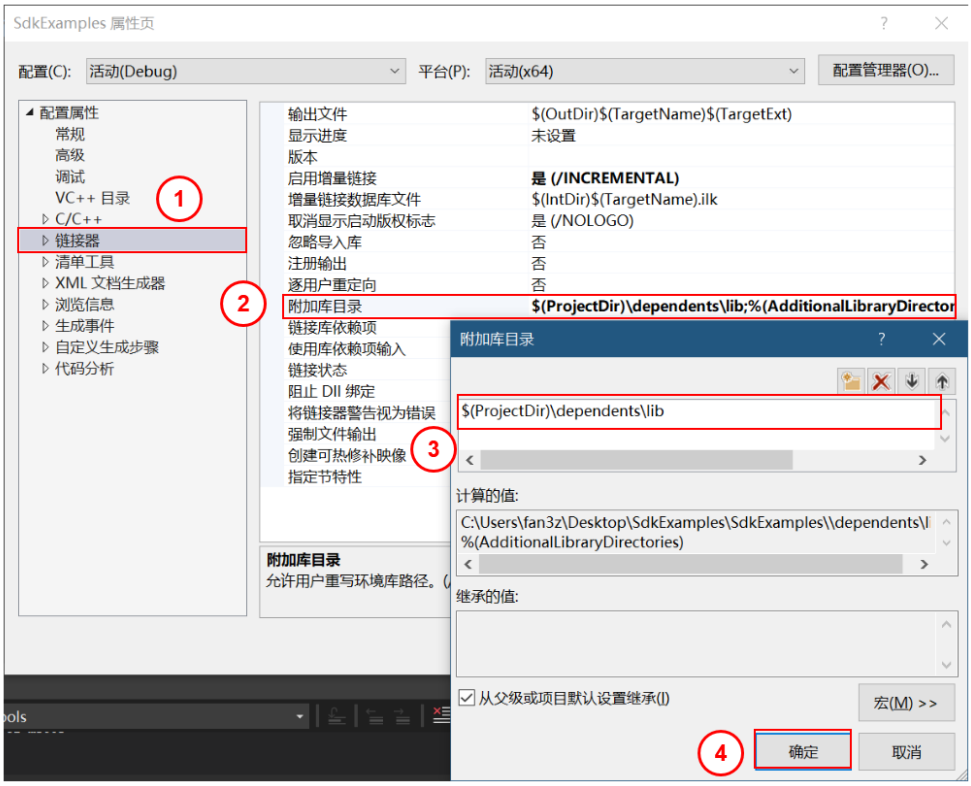
在菜单栏中，选择“项目”→“属性”。此时，“C/C++”出现在“配置属性”栏中。



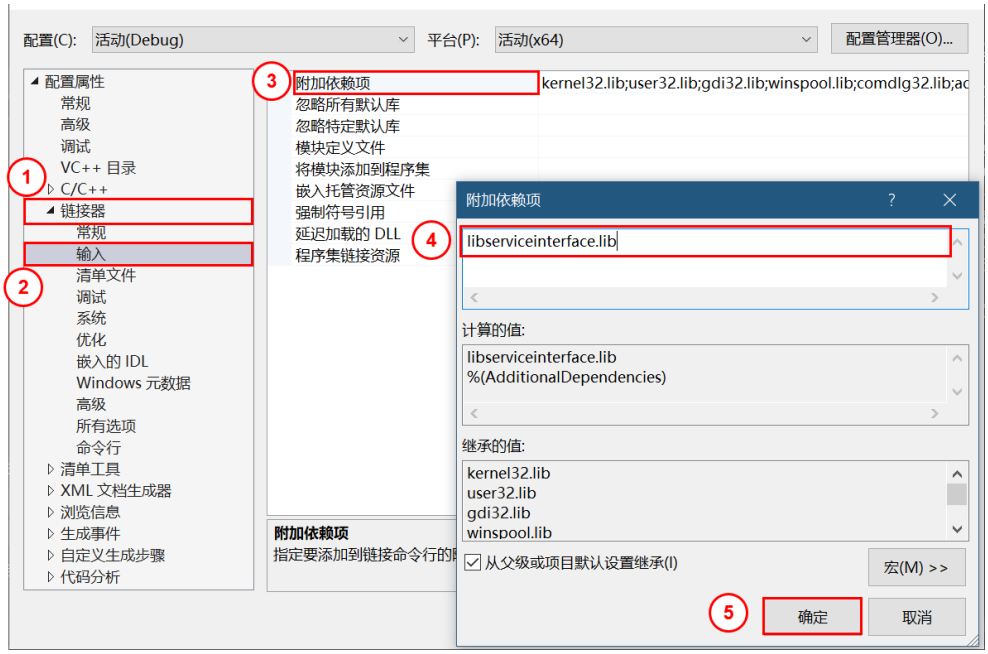
8. 添加工程的头文件目录：点击“C/C++”→“附加包含目录”→添加头文件所在的路径→点击“确定”。



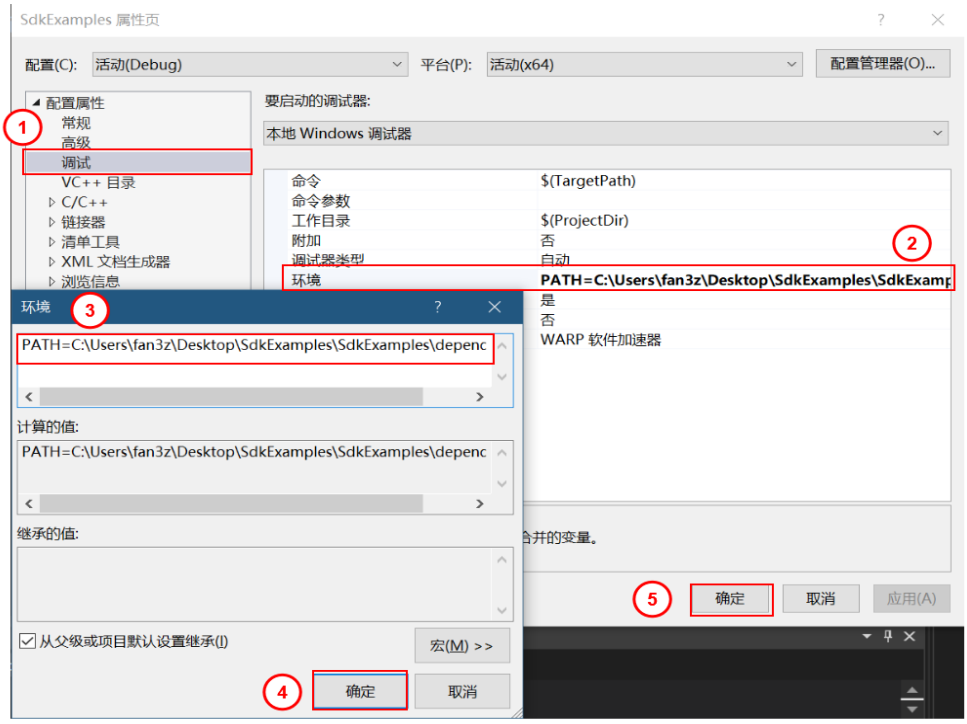
9. 添加 lib 库文件：“链接器”→“附加库目录”→添加路径→点击“确定”。



10. “链接器”→“输入”→“附加依赖项”→输入 lib 库文件名→点击“确定”。



11. 添加 dll 文件：“调试”→“环境”→添加路径→点击“确定”→点击“确定”。



12. 在 main.cpp 中写一个机械臂登录的例程。

```cpp
#include "rsdef.h"
#include <iostream>
#include <string>

#define M_PI 3.14159265358979323846
#define ROBOT_ADDR "192.168.221.13"
#define ROBOT_PORT 8899

RSHD rshd = -1;

int main()
{
    //初始化接口库
    if (rs_initialize() == RS_SUCC)
    {
        std::cout << "初始化成功。" << std::endl;
        //创建上下文
        if (rs_create_context(&rshd) == RS_SUCC)
        {
            //登录机械臂服务器
            if (rs_login(rshd, ROBOT_ADDR, ROBOT_PORT) == RS_SUCC)
            {
                //登录成功
                std::cout << "登录成功。" << std::endl;
```

连臂或 aubo 虚拟机，运行程序。

如下图打印出“登录成功”的语句，说明环境配置成功。

```
sdk log: disenable joint1 360.
sdk log: login completed, set joint1 retate 360 falg success. joint1Rot360Enable=0
Joint safety range from auboserver is invalid
登录成功。
sdk log: User blocking call motion interface.
```