智流形机器人

从 制 造 到 智 造

INSTRUCTION

智流形机器人控制系统

二次开发接口使用文档

版本变更表

版本	变更日期	编写人	变更说明
V1.0	2023/12/13	符传晶	
V1.1	2025/6/11	符传晶	

版权声明

声明

深圳市智流形机器人技术有限公司保留在没有预先通知的情况下修改产品或其特性的权利。

深圳市智流形机器人技术有限公司并不承担由于使用产品不当而产生的直接或是间接的伤害或损坏的责任。

联系我们

深圳市智流形机器人技术有限公司

Shenzhen iManifold Robot Technology Ltd.

地址：深圳市宝安区华丰国际机器人产业园二期 B 栋 410

智云流形科技（江阴）有限公司

iManifold Technology (Jiangyin) Ltd.

地址：江苏省无锡市江阴市长山大道 55 号天安数码城 60 幢 102 室

目　录

1 网络 API 开发平台介绍

本功能为工业机器人的可开发接口，设计成以网络形式的机器人控制器为目标，提供工业机器人的系统配置参数，变量资源，运动控制指令，机器人工作状态等接口。

通过本功能可对机器人系统进行二次开发，从而实现机器人实时运行状态的监控和收集，多设备互联互通和协同配合运行，远程操作控制和运维等需求。

部分特性：

1. 所有控制指令采用异步模式，用戶可通过监控机器人的状态进行逻辑控制；

2. 所有获取状态指令采用同步模式，调用线程会被阻塞；

3. 监控状态可选择调用接口（返回结构体包含所有机器人状态数据），或者选择通过 UDP 读取（UDP广播频率为 50ms，端口号为 23334，发送格式为二进制的结构体）；

4. 所有接口基于 C++11 开发；

5. 提供基于心跳的断线检查服务；

6. 实时任务等级为 50-100ms；

7. 所有单位采用 mm/deg；

8. 所有编号采用自然编号，即从 1 开始；

9. 除非特别提及，如果项目不存在则自动创建，如果已经存在则覆盖；

10. 提供基于 Win(MSVC2019_32/64bit)和 Linux(Ubuntu18.04)接口；

11. 功能仅在远程模式下生效.

1.1 开发平台配置

1.1.1 Win11 系统下使用

系统：win11

IDE：QT5.9.0

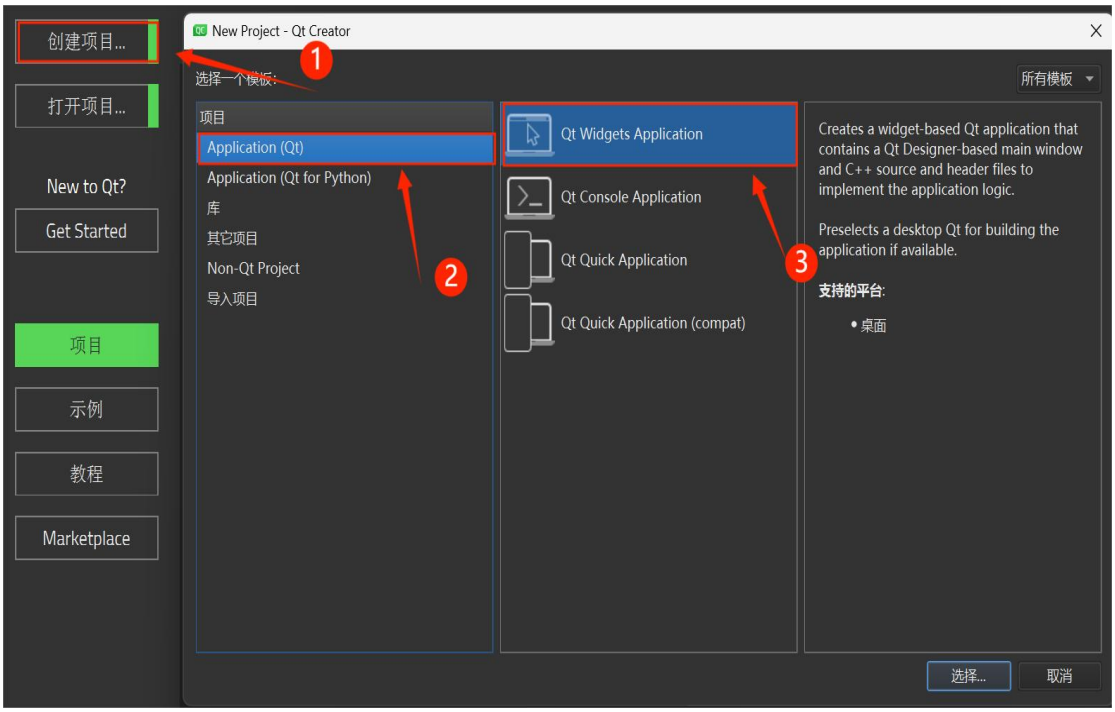
编译器：msvc2015 及以上版本

项目创建&加载 C++SDK

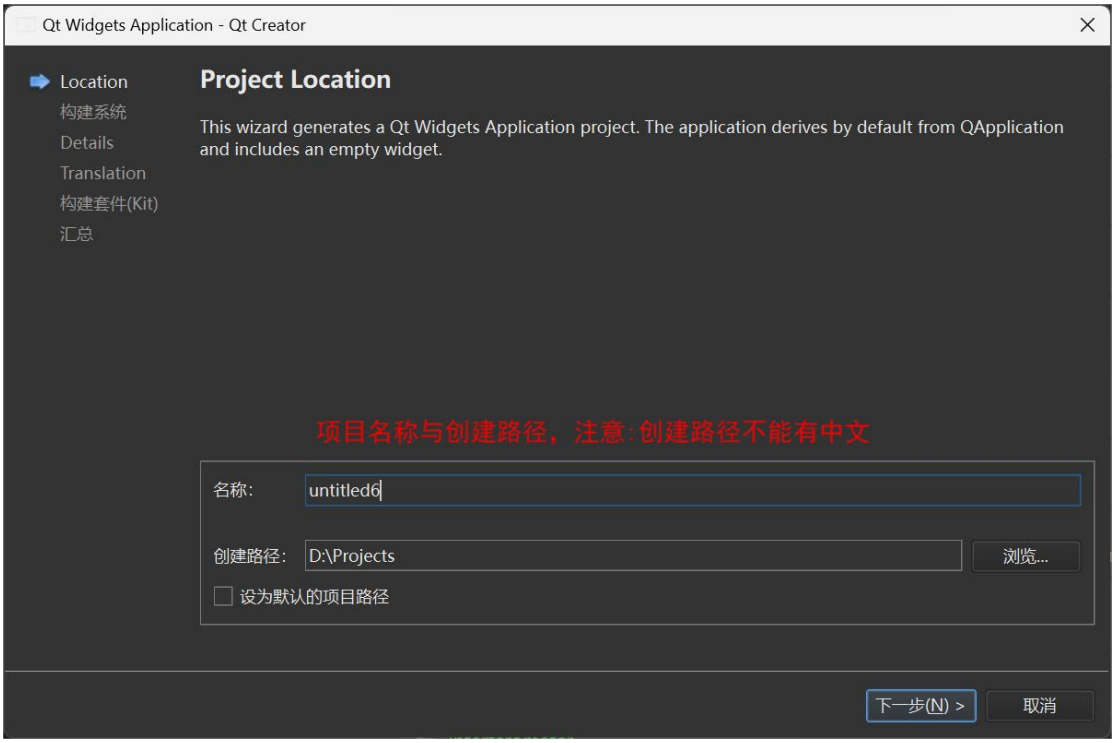
在 QT5.9.0 下创建 C++项目，此处创建的项目为带有 UI 界面的项目，方便后期调试。

创建 C++项目

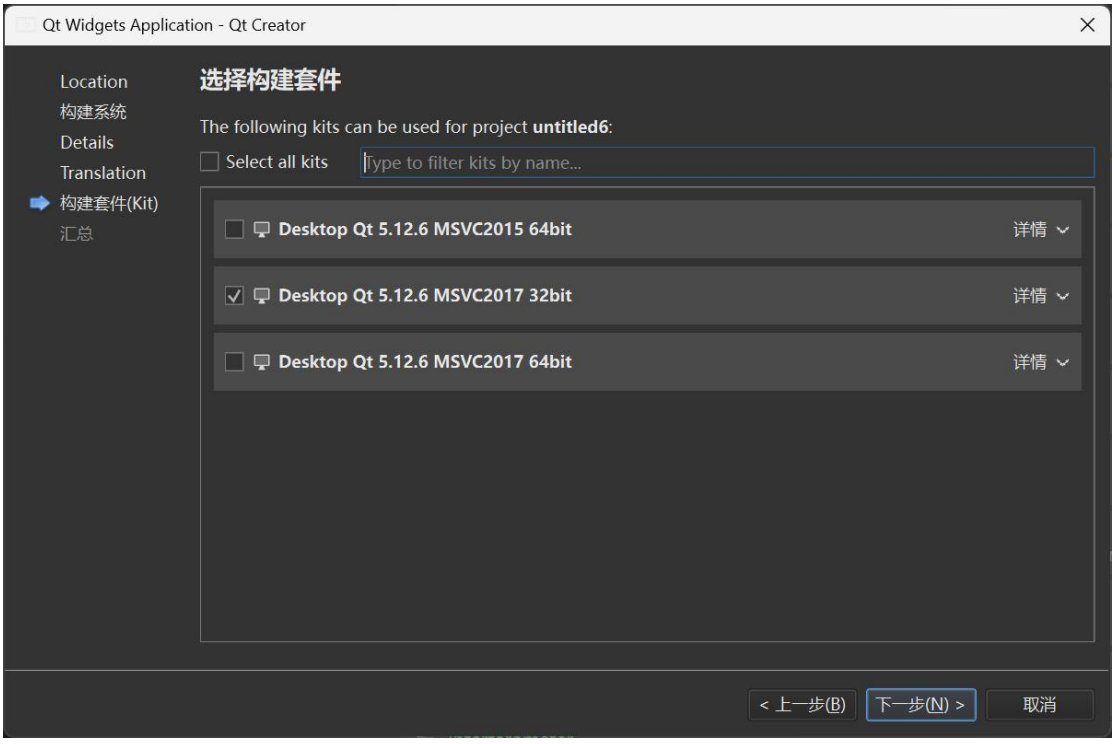
第一步，点击文件打开“新建项目或文件”,在项目选择界面根据下图提示选择项目模板，并且点击“choose”按钮，进入下一步。



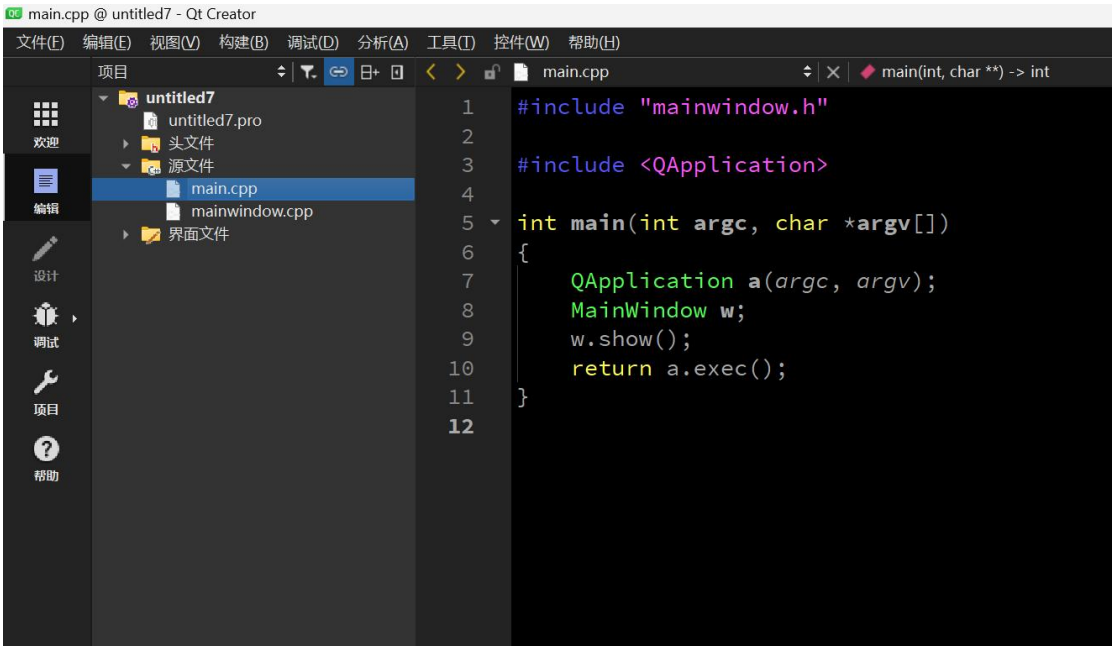
第二步，填写项目名称与确认创建项目的路径。此处需注意：创建路径及文件夹不能有中文。



第三步，选择 KITS、Details、汇总信息。如下图，默认即可。

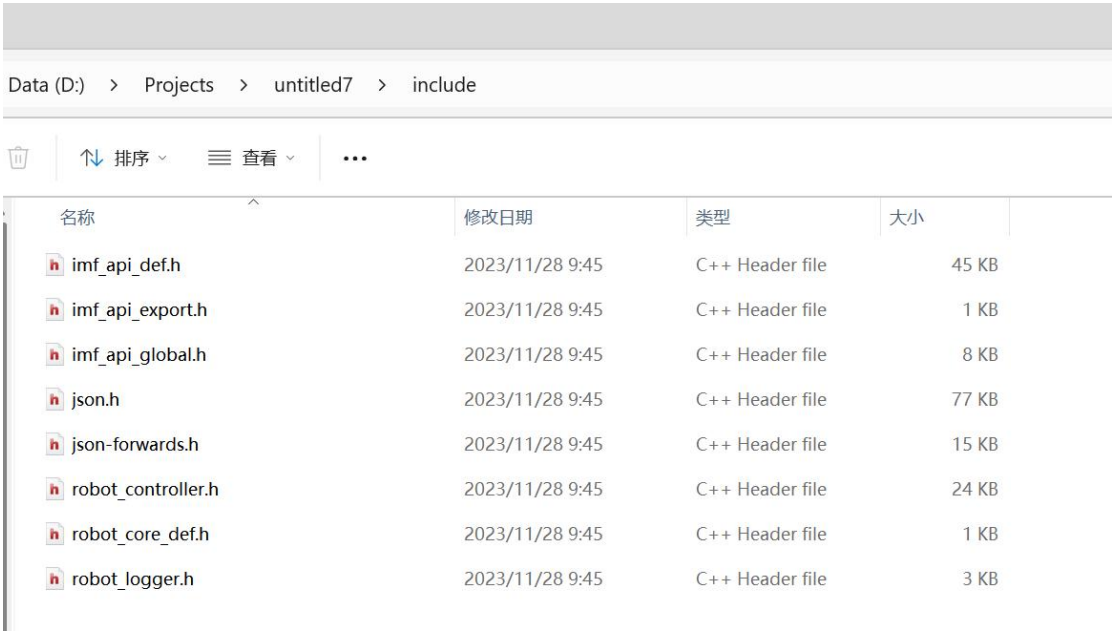


第四步，项目创建完成，以下界面为项目创建成功显示。

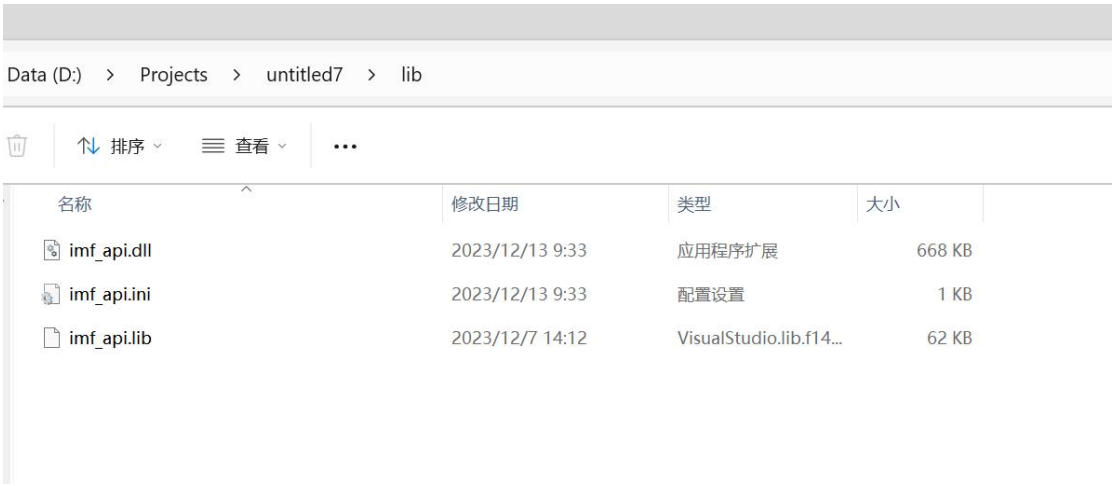


加载 SDK 文件

第一步，找到设备资料中的 API 文件夹，找到目录“D:\SVN\api\release v4.0.x\winx86\include”下的“include”文件夹。复制此文件夹至创建完成的 QT 项目文件夹中。



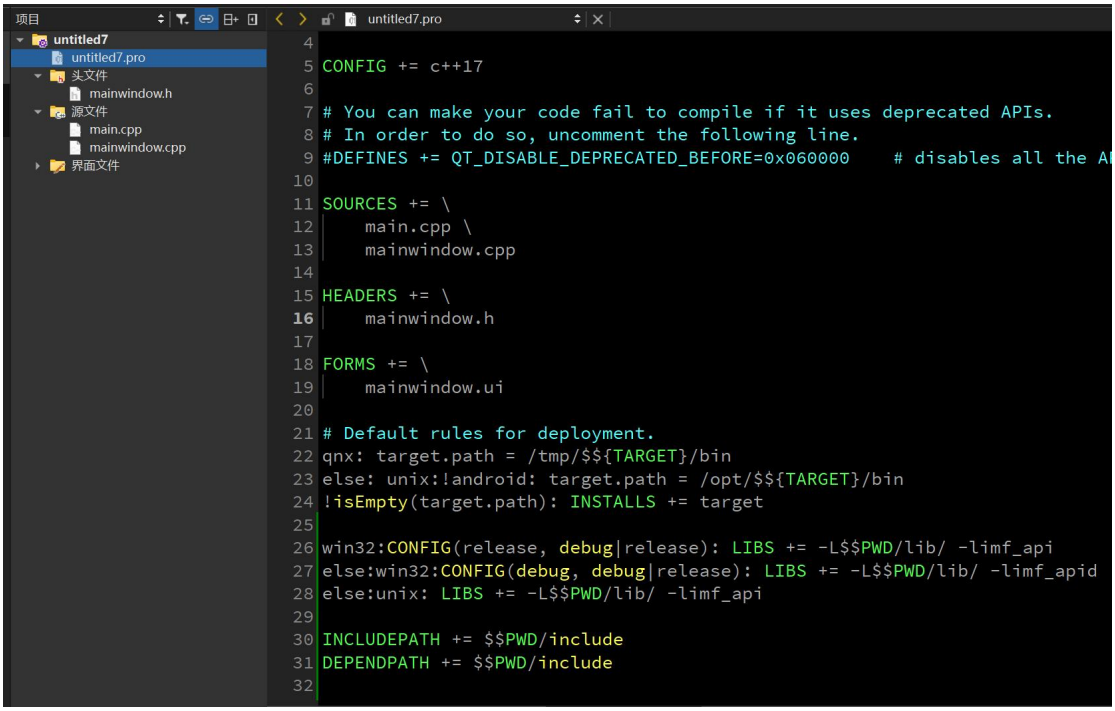
第二步，找到目录“D:\SVN\api\release v4.0.x\winx86\MSVC_2017_64bit”文件夹，复制此文件夹至创建完成的 QT 项目文件夹中,并更改文件夹名称为“lib”



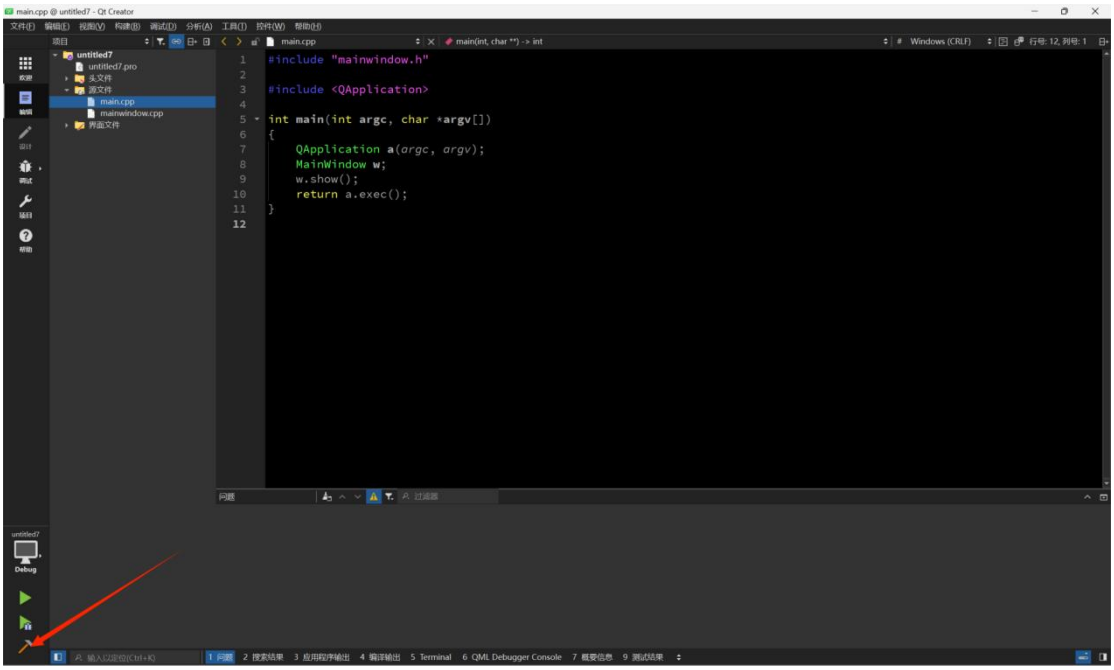
第三步，打开‘intitled7.pro’文件，在文件末尾处添加以下代码：

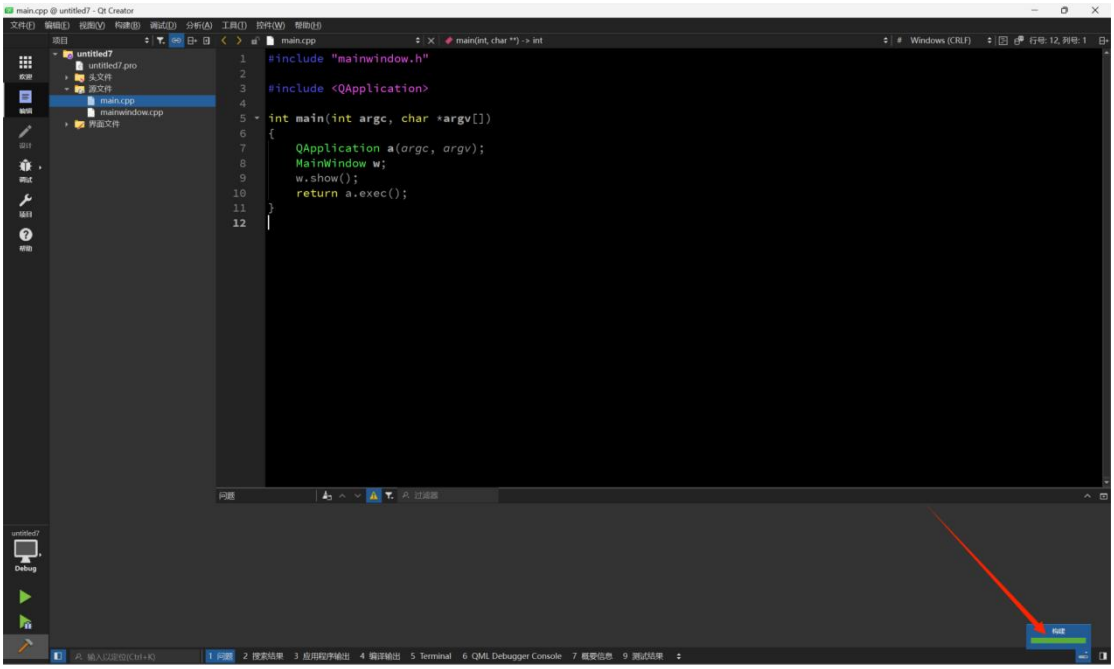
win32:CONFIG(release, debug|release): LIBS += -L$$PWD/lib/ -limf_api
else:win32:CONFIG(debug, debug|release): LIBS += -L$$PWD/lib/ -limf_apid
else:unix: LIBS += -L$$PWD/lib/ -limf_api

INCLUDEPATH += $$PWD/include
DEPENDPATH += $$PWD/include



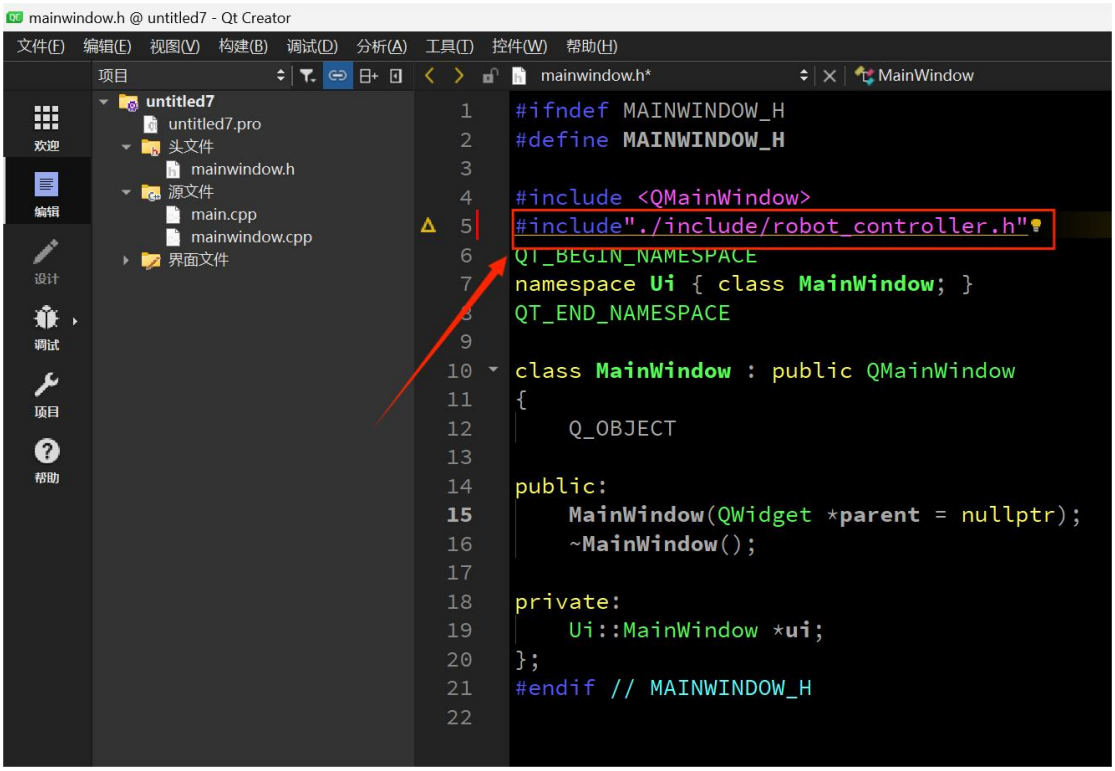
第四步，编译项目，导入 include 文件夹与 dll 文件。点击 QT 软件中的构建按钮，并等待构建完成，如未报错说明构建成功。





第五步，添加 rm_service.h 头文件。打开 QT 项目中的‘mainwindow.h’文件，在文件头部添加以下头文件：

#include"./include/robot_controller.h"



至此，项目创建&加载 C++ SDK 到此结束，接下来可以进行代码的编写。

1.1.2 Linux 系统下使用

开发环境说明

系统：Ubuntu 18.04

IDE：QT Creator(4.9.2)

编译器：默认

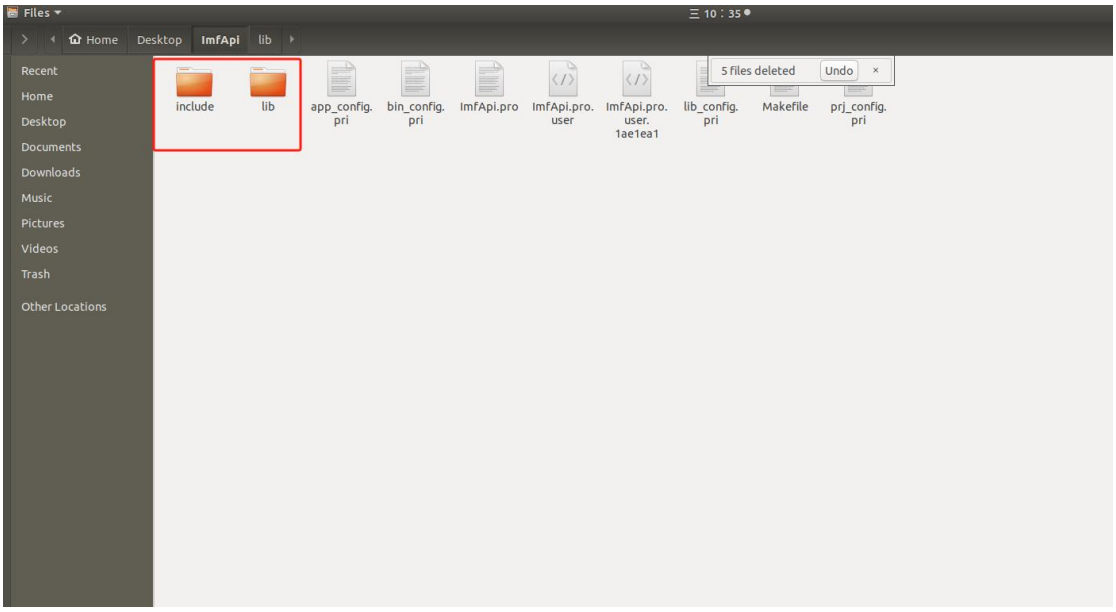
JixSDK 版本号：4.1.3

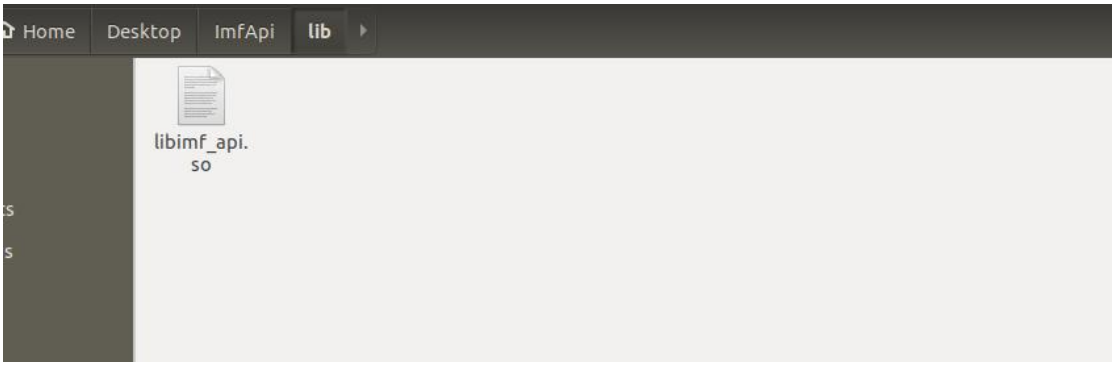
项目创建&加载 C++SDK

Ubuntu 下项目的创建及加载 SDK，与 win10 下项目的创建方法大致相同。

适用于 Ubuntu 系统的库文件位于‘D:\SVN\api\release v4.0.x\linux’文件夹下，复制此文件夹至 Linux 创建完成的 QT 项目文件夹中,并更改文件夹名称为“lib”,” include“文件夹无需替换，直接复制即可。

根据自身系统的不同，选择合适的库文件即可。此处，我使用的是虚拟机运行。





1.2 基于 Qt 的 demo 工程实例(随文档提供)

参考文件夹例程

2 结构体及枚举变量说明

2.1 机器人位置型变量结构体

```
class TRobotVarPoint: public TIdData
{
public:
    rcf::RobotCoord coord_type                    = rcf::RobotCoord::ACS;
    short pcs_idx                                 = 0;
    short wcs_idx                                 = 0;
    short tool_idx                                = 0;
    double joint_pos[rcf::k_MAX_ROBOT_JOINT]      = {0};
    double pos[3]                                 = {0};
    double ori[3]                                 = {0};
    rcf::TRobotPointConfig config;
    double ext_joint[rcf::k_MAX_EXT_JOINT]        = {0};
    rcf::ExtGroup ext_group = rcf::ExtGroup::Ungrouped;
    std::string comment                           = " " ;
    TRobotVarPoint(bool flag = false): is_inc_point(flag) {}
    inline bool isIncPoint() const
    {
        return is_inc_point;
    }
protected:
    bool is_inc_point = false;
};
```

结构体说明:

coord_type ： 点对应的坐标系类型

pcs_idx ： 点对应的 pcs 序号，仅对笛卡尔位置有效

wcs_id ： 点对应的 wcs 序号，仅对笛卡尔位置有效

short tool_idx ： 点对应的 tool 序号，仅对笛卡尔位置有效

joint_pos[rcf::k_MAX_ROBOT_JOINT] ： 关节位置，单位: deg / mm

pos[3] ： 笛卡尔位置，单位 ： mm

ori[3] ： 姿态 RPY 欧拉角，单位 ： deg

config ： 对应机器人构型，仅对笛卡尔位置有效

ext_joint[rcf::k_MAX_EXT_JOINT] ： 外部轴的关节位置

ext_group ： 点位绑定的外部轴，运动到点的时候外部轴会同步运动

comment ： 注释说明

is_inc_point ： 数据库内没有对应的数据，不要修改这个字段

2.2 机器人增量位置型变量结构体

```
class TRobotVarIncPoint: public TRobotVarPoint
{
public:
    TRobotVarIncPoint(): TRobotVarPoint(true) {}
};
```

结构体说明：

TRobotVarIncPoint() TRobotVarPoint(true){} ： 与位置型变量共用一个结构体，flag 为 true 时为增量位置型变量

2.3 机器人数值型变量结构体

```
typedef struct TRobotVarNumber: public TIdData
{
    double value              = 0;
    std::string comment       = "";
} TRobotVarNumber;
```

结构体说明 ：

Value ： 变量值

Comment ： 注释说明

2.4 机器人字符型变量结构体

```
typedef struct TRobotVarString: public TIdData
{
    std::string content     = "";
} TRobotVarString;
```

结构体说明:

Content ： 变量内容

2.5 机器人工程结构体

```
typedef struct
{
    std::string name    = "null";
    std::vector <TRobotProgram> program_list = {} ;
    std::vector <TRobotVarIncPoint> inc_point_list = {};
    std::vector <TRobotVarPoint> point_list = {};
    std::vector <TRobotVarString> string_list = {};
    std::vector <TRobotVarNumber> number_list = {};
} TRobotProject;
```

结构体说明:

Name ： 工程名

program_list ： 工程程序列表

inc_point_list ： 工程增量位置点列表

point_list ： 工程位置点列表

string_list ： 工程字符变量列表

number_list ： 工程数值型变量列表

2.6 机器人示教程序结构体

```
typedef struct TRobotProgram: public TNameData
{
```

```
    std::string last_modify_date ="";
    std::string content = "";
short type = 0;
}TRobotProgram;
```

结构体说明 ：

last_modify_date ：最近修改日期

content ： 内容，utf-8 编码

type ：程序的类型(program/procedure/function)，当前无效

2.7 机器人笛卡尔参数结构体

```
typedef struct TRobotCartesianParam: public TIdData
{
    double positive_limit[6] = {0};
    double negative_limit[6] = {0};
    double max_jog_vel[6]      = {0};
    double max_jog_acc[6]      = {0};
    double max_jog_dec[6]      = {0};
    double max_linear_vel      = 0;
    double max_linear_acc      = 0;
    double max_linear_dec      = 0;
    double max_angular_vel     = 0;
    double max_angular_acc     = 0;
    double max_angular_dec     = 0;
} TRobotCartesianParam;
```

结构体说明 ：

positive_limit[6] ：XYZABC 各个方向的正限位，单位： mm 或 deg

negative_limit[6] ：XYZABC 各个方向的负限位，单位： mm 或 deg

max_jog_vel[6] ：XYZABC 各个方向的 Jog 最大速度，单位： mm/s 或 deg/s

max_jog_acc[6] : XYZABC 各个方向的 Jog 最大加速度, 单位 : mm/s 或 deg/s

max_jog_dec[6] : XYZABC 各个方向的 Jog 最大减速度, 单位 : mm/s 或 deg/s

max_linear_vel : 末端合成最大线速度，单位 : mm/s

max_linear_acc : 末端合成最大线加速度，单位 : mm/s^2

max_linear_dec : 末端合成最大线减速度，单位 : mm/s^2

max_angular_vel : 末端合成最大角速度，单位 : deg/s

max_angular_acc : 末端合成最大角加速度，单位 : deg/s^2

max_angular_dec : 末端合成最大角减速度，单位 : deg/s^2

2.8 机器人规划器参数结构体

```
typedef struct TRobotProfileParam :  public TIdData
{
    rcf::ProfileType profile_type          = rcf::ProfileType::Cos;
    int smooth_size                        = 0;
    rcf::BlendType blend_type              = rcf::BlendType::Fusion;
    int max_follow_error                   = 0;
    int jog_jerk_time                      = 0;
    int replay_jerk_time                   = 0;
    int replay_vel                         = 0;
    int replay_acc                         = 0;
    int replay_dec                         = 0;
    int teach_max_joint_speed              = 0;
    int teach_max_cartesian_speed          = 0;
    rcf::SingularityPolicy singularity_policy =
rcf::SingularityPolicy::WarnAndStop;
    int server_on_delay                    = 0;
} TRobotProfileParam;
```

结构体说明 :

profile_type：规划器类型

smooth_size：平滑滤波尺寸

blend_type：平滑类型

max_follow_error：最大跟随误差，单位 ： pulse

jog_jerk_time： Jog 模式的 Jerk Time，单位 ： ms

replay_jerk_time：回放模式下默认 Jerk Time，单位 ： ms

replay_vel：回放速度默认百分比，范围 ： 0-100

replay_acc：回放加速度默认百分比，范围 ： 0-100

replay_dec：回放减速度默认百分比，范围 ： 0-100

teach_max_joint_speed：试教模式下最大关节运动百分比 0-100

teach_max_cartesian_speed：试教模式下最大笛卡尔运动速度，单位 mm/s

singularity_policy：奇异点策略

server_on_delay：伺服延迟，单位 ms

2.9 机器人模型参数结构体

```
typedef struct TRobotKinParam: public TIdData
{
    double param[k_KIN_PARAM_SIZE] = {0};
} TRobotKinParam;
```

结构体说明 ：

param[k_KIN_PARAM_SIZE]：模型结构参数

2.10 机器人系统硬件参数结构体

```
typedef struct TRobotHardwareParam: public TIdData
{
    long enc_resolution_plan                = 0;
    long enc_resolution_fb                  = 0;
    // int enc_port                         = 0;
    // int enc_type                         = 0;
    // short enc_auto_correct_interval      = 0;
```

```
    int motor_max_speed                         = 0;
    double motor_max_current                    = 0;
    double motor_trq_current_ratio              = 0;
    double reduction_ratio_joint                = 0;
    double reduction_ratio_motor                = 0;
    // double coupling_ratio[rcf::k_MAX_ROBOT_JOINT]      = {0};
} TRobotHardwareParam;
```

结构体说明:

enc_resolution_plan:编码器规划分辨率，单位: pulse/round

enc_resolution_fb:编码器反馈分辨率，单位: pulse/round

enc_port:编码器通讯端口，未使用

enc_type:编码器类型，未使用

enc_auto_correct_interval:编码器矫正周期, 单位: ms，未使用

motor_max_speed:电机最大转速，单位: rpm

motor_max_current:电机最大电流，单位: A

motor_trq_current_ratio:电机力矩常数，单位: N.m / A

reduction_ratio_joint:关节侧减速比

reduction_ratio_motor:电机侧减速比

coupling_ratio[rcf::k_MAX_ROBOT_JOINT]:与其他轴的耦合比，未使用

2.11机器人关节参数结构体

```
typedef struct TRobotJointParam: public TIdData
{
    double positive_limit = 0;
    double negative_limit = 0;
    double bias    = 0;
    bool reverse = 0;
    double max_jog_vel = 0;
    double max_jog_acc = 0;
```

```
    double max_jog_dec = 0;
    double max_vel = 0;
    double max_acc = 0;
    double max_dec = 0;
    unsigned char driver_id = 0;
    rcf::AxisUnit unit = AxisUnit::NOT_USED ;
} TRobotJointParam;
```

结构体说明:

positive_limit:轴正限位，单位: deg 或 mm

negative_limit:轴负限位，单位: deg 或 mm

bias:轴偏移大小, Joint = (1-2*reverse) * (Axis + bias), 单位: deg 或 mm

reverse:轴是否反向, Joint = (1-2*reverse) * (Axis + bias), 1 表示有效

max_jog_vel:Jog 最大速度，单位: deg/s 或 mm/s

max_jog_acc:Jog 最大加速度，单位: deg/s^2 或 mm/s^2

max_jog_dec:Jog 最大减速度，单位: deg/s^2 或 mm/s^2

max_vel:最大速度，单位: deg/s 或 mm/s

max_acc:最大加速度，单位: deg/s^2 或 mm/s^2

max_dec:最大减速度，单位: deg/s^2 或 mm/s^2

driver_id:轴对应的硬件驱动器地址，从 1 开始，一一对应

unit:轴类型，平移 / 旋转 / 未使用

2.12机器人外部轴参数结构体

```
typedef struct TRobotExtJointParam: public TRobotJointParam
{
    double positive_limit = 0;
    double negative_limit = 0;
    double bias    = 0;
    bool reverse = 0;
    double max_jog_vel = 0;
```

```
    double max_jog_acc = 0;
    double max_jog_dec = 0;
    double max_vel = 0;
    double max_acc = 0;
    double max_dec = 0;
    unsigned char driver_id = 0;
    rcf::AxisUnit unit = AxisUnit::NOT_USED ;
} TRobotExtJointParam;
```

结构体说明:

positive_limit:轴正限位，单位: deg 或 mm

negative_limit:轴负限位，单位: deg 或 mm

bias:轴偏移大小, Joint = (1-2*reverse) * (Axis + bias), 单位: deg 或 mm

reverse:轴是否反向, Joint = (1-2*reverse) * (Axis + bias), 1 表示有效

max_jog_vel:Jog 最大速度，单位: deg/s 或 mm/s

max_jog_acc:Jog 最大加速度，单位: deg/s^2 或 mm/s^2

max_jog_dec:Jog 最大减速度，单位: deg/s^2 或 mm/s^2

max_vel:最大速度，单位: deg/s 或 mm/s

max_acc:最大加速度，单位: deg/s^2 或 mm/s^2

max_dec:最大减速度，单位: deg/s^2 或 mm/s^2

driver_id:轴对应的硬件驱动器地址，从 1 开始，一一对应

unit:轴类型，平移 / 旋转 / 未使用

2.13机器人编码器零点参数结构体

```
typedef struct TRobotReferencePoint: public TIdData
{
    bool abs_enc_ref_valid[rcf::k_MAX_AXIS];
    long abs_enc_ref_count[rcf::k_MAX_AXIS];
} TRobotReferencePoint;
```

结构体说明:

abs_enc_ref_valid[rcf::k_MAX_AXIS]:编码器零点参数是否有效

abs_enc_ref_count[rcf::k_MAX_AXIS]:编码器零点参数

2.14机器人耦合比参数结构体

```
typedef struct TRobotCoupleParam: public TIdData
{
    int driving_joint;
    int driven_joint;
    double couple_ratio;
} TRobotCoupleParam;
typedef std::vector<TRobotCoupleParam> TRobotCoupleParamList;
```

结构体说明:

driving_joint:驱动轴

driven_joint:从动轴

couple_ratio:耦合比系数

2.15机器人 IO 配置结构体

```
typedef struct TRobotIOConfig: public TIdData
{
    int module;
    int function_id;
    std::string function_info;
    int group;
    int index;
    bool is_revert;
    int filter_time;
    bool is_enable;
    int io_type;
} TRobotIOConfig;
typedef std::vector<TRobotIOConfig> TRobotIOConfigList;
```

结构体说明:

Module:从属模块

function_id:功能代码

function_info:功能描述

group:数字量组

index:数字量点位

is_revert:有效值是否反向

filter_time:滤波时间，单位 ms

is_enable:是否启用

io_type:IO 类型，0 表示输入，1 表示输出

2.16背景任务配置结构体

```
typedef struct TRobotPlcConfig: public TIdData
{
    std::string project;
    std::string program;
    LineMode line_mode;
    TriggerMode trigger_mode;
    bool is_enabled;
} TRobotPlcConfig;
typedef std::vector<TRobotPlcConfig> TRobotPlcConfigList;
```

结构体说明:

Project:工程名

Program:程序名

line_mode:启动行

trigger_mode:启动模式

is_enabled:是否启用

2.17机器人回零结构体

目前回零只对应机器人轴，不涉及到外部轴

```
typedef struct TRobotHomePos: public TIdData
{
    double joint_pos[rcf::k_MAX_ROBOT_JOINT];
} TRobotHomePos;
```

结构体说明:

joint_pos[rcf::k_MAX_ROBOT_JOINT]:零点对应的机器人关节位置

2.18机器人系统其他参数结构体

```
typedef struct TRobotMiscSettings: public TIdData
{
    RemoteControlMode remote_control_mode = RemoteControlMode::CustomTcp;
    int init_coord = static_cast<int>(RobotCoord::ACS);
    int start_line = 0;
    bool check_connection = true;
    Movc2TeachCheckBehavior movc_behavior = Movc2TeachCheckBehavior::Linear;
    bool auto_brake_off = false;
} TRobotMiscSettings;
```

结构体说明:

remote_control_mode:远程控制模式

init_coord:初始坐标系

start_line:初始启动行

check_connection:是否断线检查

movc_behavior:Movc2 Jog 模式

auto_brake_off:是否打开自动松抱闸

2.19全局工作空间监控参数结构体

```
typedef struct TRobotGlobalWorkSpaceParam: public TIdData
{
```

```
    bool global_enable = false;
    bool workspace_do_enable = false;
    int workspace_do_group = 0;
    int workspace_do_index = 0 ;
    int workspace_do_value = 0;
    int workspace_bind_var = 0;
    bool obstacle_do_enable = false;
    int obstacle_do_group     = 0;
    int obstacle_do_index     = 0;
    int obstacle_do_value     = 0;
    int obstacle_bind_var     = 0;
    bool monitor_do_enable    = false;
    int monitor_do_group      = 0;
    int monitor_do_index      = 0;
    int monitor_do_value      = 0;
    int monitor_bind_var      = 0;
} TRobotGlobalWorkSpaceParam;
```

结构体说明:

global_enable:是否启用

workspace_do_enable:是否输出到 DO

workspace_do_group:输出 DO 的 group

workspace_do_index:输出 DO 的 index

workspace_do_value:输出 DO 的 值

workspace_bind_var:同时输出到全局变量， 0 表示不输出

obstacle_do_enable:是否输出到 DO

obstacle_do_group:输出 DO 的 group

obstacle_do_index:输出 DO 的 index

obstacle_do_value:输出 DO 的 值

obstacle_bind_var:同时输出到全局变量， 0 表示不输出

monitor_do_enable:是否输出到 DO

monitor_do_group:输出 DO 的 group

monitor_do_index:输出 DO 的 index

monitor_do_value:输出 DO 的 值

monitor_bind_var:同时输出到全局变量， 0 表示不输出

2.20工作空间监控参数结构体

```
typedef struct TRobotWorkSpaceParam: public TIdData
{
    WorkSpaceMode mode       = WorkSpaceMode::Disabled;
    WorkSpaceType type       = WorkSpaceType::WorkArea;
    uint16_t pcs_idx         = 0;
    EndPointType endpoint_type     = EndPointType::ToolTip;
    WorkSpaceShape shape             = WorkSpaceShape::Sphere;
    double shape_param[k_SHAPE_PARAM_NUM]   = {0};
    double offset[6]   = {0};
    double margin    = 0;
    bool do_enable = false;
    int do_group = 0;
    int do_index = 0;
    int do_value    = 0;
    bool di_enable = false;
    int di_group = 0;
    int di_index = 0;
    int di_value = 0;
    int bind_var = 0;
    bool ignore_lock_output = false;
} TRobotWorkSpaceParam;
```

结构体说明:

Mode:监控区域模式

Type:监控区域类型

pcs_idx:监控区域对应的工件坐标系

endpoint_type:监控点类型

shape:监控区域形状

shape_param[k_SHAPE_PARAM_NUM]:监控区域形状参数

offset[6]:监控区域偏移，偏移用于设置安全区域

margin:监控区域裕量

do_enable:是否输出 Do

do_group:输出 Do 的 group

do_index:输出 Do 的 index

do_value:输出 Do 的 值

di_enable:干涉区是否对应 Di

di_group:干涉区 Di 的 group

di_index:干涉区 Di 的 index

di_value:干涉区 Di 的 值

bind_var:绑定的全局变量编号，0 表示不绑定

ignore_lock_outpu:是否忽略锁定输出

2.21机器人驱动器动态参数结构体

```
typedef struct TRobotDynamicConfig: public TIdData
{
    bool is_valid = false;
    int joint = 0;
    ValueType type = ValueType::Payload;
    double value = 0;
    double joint_acc_coeff = 0;
    double joint_dec_coeff = 0;
    bool has_driver_pos_loop_gain = false;
```

```
    double driver_pos_loop_gain_coeff = 0;
    bool has_driver_estop_dec = false;
    double driver_estop_dec_coeff = 0;
} TRobotDynamicConfig;
typedef std::vector<TRobotDynamicConfig> TRobotDynamicConfigList;
```

结构体说明:

is_valid:本行配置是否生效

joint:本行记录对应的关节，从 1 开始

type:本行记录对应的变量类型 负载(kg)/偏心距(mm)/惯量（kg*m^2）

value:表示记录对应的变量值

joint_acc_coeff:表示加速度 mm/s/s or deg/s/s

joint_dec_coeff:表示减速度 mm/s/s or deg/s/s

has_driver_pos_loop_gain:对应驱动器参数是否有效

driver_pos_loop_gain_coeff:对应驱动器参数系数

has_driver_estop_dec:对应驱动器参数是否有效

driver_estop_dec_coeff:对应驱动器参数系数

2.22机器人驱动器参数结构体

```
typedef struct TRobotDriverParam: public TIdData
{
    double pos_loop_gain;
    uint32_t estop_dec;

} TRobotDriverParam;
```

结构体说明:

pos_loop_gain:对应 PDO，0x2040 位置环比例增益 1

estop_dec:对应 PDO，0x6085 紧急停止减速度 单位: inc/s^2

2.23机器人动态驱动参数配置结构体

```
typedef struct TRobotDynamicSetting: public TIdData
```

```
{
    bool is_enable = false;
    double payload = 0;
    double bias = 0;
    double inertia = 0;
} TRobotDynamicSetting;


enum class CoordSysAttribute : rcf::enum_type
{
    Normal = 0,
    ExtTCP
};
```

结构体说明:

is_enable:是否使能，默认 false

payload:对应负载，单位 kg

bias:对应偏心距，单位 mm

inertia:对应负载惯量，单位 kg*m^2

Normal:正常使用

ExtTCP:作为外部 TCP

2.24坐标系结构体结构体

```
typedef struct TCoordSys
{
    double pos[3]                                   = {0};
    double ori[3]                                   = {0};
    CoordSysAttribute attribute = CoordSysAttribute::Normal;
    std::string comment =                   "";
} TCoordSys;
```

结构体说明:

pos[3]:位置

ori[3]:姿态

attribute:坐标系的附加属性

comment:说明

2.25坐标系具体配置结构体

```
typedef struct TRobotCoordSys: public TIdData
{
    TCoordSys wcs;
    TCoordSys tcs;
    TCoordSys pcs1;
    TCoordSys pcs2;
} TRobotCoordSys;
```

结构体说明:

wcs:wcs 坐标系

tcs:tcs 坐标系

pcs1:pcs1 坐标系

pcs2:pcs2 坐标系

2.26上次保存的 jog 坐标系信息结构体

```
///     1. 该信息在系统初始化的时候配置给 motion
///     2. 该信息在用户修改当前坐标系的时候记录到数据库
typedef struct TRobotJogCoord: public TIdData
{
    int tcs_id = 1;
    int wcs_id = 1;
    int pcs1_id = 1;
    int pcs2_id = 1;
    rcf::RobotCoord coord = RobotCoord::ACS;
} TRobotJogCoord;
```

结构体说明:

tcs_id:当前 tool

wcs_id:当前 wcs id

pcs1_id:当前 pcs1 id

pcs2_id:当前 pcs2 id

coord:当前坐标系类型

2.27用户自定义信息结构体

```
typedef struct TUserData: public TNameData
{
    std::string table = "null";
    std::string version = "null";
    std::string data = "";
} TUserData;
```

结构体说明:

Table:表格名称

Version:对应版本

Data:二进制数据

2.28机器人系统日志信息结构体

```
typedef struct TLogInfo
{
    uint32_t code = 0;
    int level = 0;
    std::string message = "undefined";
    std::string source = "unknown";
    std::string timestamp = "invalid";
} TLogInfo;
```

结构体说明:

Code:错误代码

Level:报警等级

Message:报警信息

Source:报警单元

Timestamp:报警日期，格式为 "yyyy-MM-dd hh:mm:ss"

2.29实时机器人信息结构体

```
typedef struct RtPointInfo
        {
            uint64_t Rt_times;
            rcf::TRobotInforobot_info;
        } RtPointInfo;
```

结构体说明:

Rt_times:实时时间

TRobotInfo:机器人信息

2.30机器人系统配置参数类型

```
enum class RobotParam : rcf::enum_type
{
    // 机器人系统配置参数
    Carteisian = 0,
    Profile,
    Kin,
    Hardware,
    Joint,
    ExtJoint,
    ReferencePoint,
    CoordSys,
    JogCoord,
    CouplingRatio,
```

```
    // 以下参数为软件配置，和 motion 关系不大
    IOConfig = 0x0100,
    PlcConfig,
    HomePos,
    GlobalWorkSpace,
    WorkSpace,
    DriverParam,
    DynamicConfig,
    DynamicSetting,
    // 其他参数
    MiscSettings = 0xFFFF
};
```

结构体说明:

Carteisian:笛卡尔参数

Profile:规划器参数

Kin:模型参数

Hardware:硬件参数

Joint:关节参数

ExtJoint:外部轴参数

ReferencePoint:零点参数

CoordSys:坐标系参数

JogCoord:Jog 坐标系参数

CouplingRatio:耦合比参数

IOConfig: IO 配置

PlcConfig:背景任务配置

HomePos:零点参数

GlobalWorkSpace:全局工作空间

WorkSpace:工作空间

DriverParam:驱动器参数

DynamicConfig:动态配置

DynamicSetting:动态参数

MiscSettings:其他参数

2.31机器人系统 modbus 数据的定义

```
namespace rcf
{
static const uint32_t k_MAJOR = (1);
static const uint32_t k_MINOR = (8);
static const uint32_t k_PATCH = (0);
static const uint32_t k_MAGIC_NUMBER = (10000 * k_MAJOR + 100 * k_MINOR + k_PATCH);
//static const uint32_t k_MAGIC_NUMBER((k_MAJOR << 16) | (k_MINOR << 8) | k_PATCH);
//static const uint32_t k_MAGIC_NUMBER(31415926);
static const uint16_t k_USER_DATA_SIZE_IN_WORDS = (500);
static const uint16_t k_STRING_SIZE_LONG = (100);
static const uint16_t k_STRING_SIZE_MID = (50);
static const uint16_t k_STRING_SIZE_SHORT = (20);
static const uint16_t k_MAX_LOCAL_IO_GROUP = (10);
static const uint16_t k_MAX_ECAT_IO_GROUP = (50);
static const uint16_t k_MAX_AIO_COUNT = (50);
static const uint16_t k_MAX_ALLOWED_MODULES = (50);
static const uint16_t k_MAX_PLC_PROGRAMS = (10);
static const uint16_t k_MAX_WORKSPACE_NUM = (32);
static const uint32_t k_MAX_MODBUS_CHAR_SIZE = (k_STRING_SIZE_LONG);
static const uint16_t k_USER_CODE_LEN = (k_STRING_SIZE_SHORT);
static const uint16_t k_REG_CODE_LEN = (k_STRING_SIZE_MID);
```

```
    static const uint16_t k_MAX_VERSION_INFO_LEN =
(k_STRING_SIZE_MID);


    typedef struct
    {
        uint16_t bit0: 1;
        uint16_t bit1: 1;
        uint16_t bit2: 1;
        uint16_t bit3: 1;
        uint16_t bit4: 1;
        uint16_t bit5: 1;
        uint16_t bit6: 1;
        uint16_t bit7: 1;
        uint16_t bit8: 1;
        uint16_t bit9: 1;
        uint16_t bitA: 1;
        uint16_t bitB: 1;
        uint16_t bitC: 1;
        uint16_t bitD: 1;
        uint16_t bitE: 1;
        uint16_t bitF: 1;
    } TTBit16;


    typedef union
    {
        int16_t data;
        uint16_t uData;
        TTBit16 cmd;
    } TCMDDATA16;
```

结构体说明:

k_MAJOR:modbus 主版本号

k_MINOR:modbus 次版本号

k_PATCH:modbus 补丁号

k_MAGIC_NUMBER:modbus 校验数

k_USER_DATA_SIZE_IN_WORDS:用户 modbus 大小，单位：word

k_STRING_SIZE_LONG:长字符串大小

k_STRING_SIZE_MID:中字符串大小

k_STRING_SIZE_SHORT:短字符串大小

k_MAX_LOCAL_IO_GROUP:最大允许的本地 IO 组数

k_MAX_ECAT_IO_GROUP:最大允许的 ECATIO 组数

k_MAX_AIO_COUNT:最大允许的模拟量 IO 数

k_MAX_ALLOWED_MODULES:最多允许的生效工艺

k_MAX_PLC_PROGRAMS:最多允许同时执行的 plc 程序

k_MAX_WORKSPACE_NUM:最多同时允许工作空间监控

k_MAX_MODBUS_CHAR_SIZE:modbus 最大的字符串长度 @deprecated 请使用 ::k_STRING_SIZE_LONG

k_USER_CODE_LEN:用户码的大小 @deprecated 请使用 ::k_STRING_SIZE_SHORT

k_REG_CODE_LEN:注册码的大小 @deprecated 请使用 ::k_STRING_SIZE_MID

k_MAX_VERSION_INFO_LEN:版本字符串长度大小 @deprecated 请使用 ::k_STRING_SIZE_MID

```
/// task 响应的系统命令字结构体
typedef struct
{
    /// 以下结构体定义的高电平命令，只要为 1 会一直生效
    struct _X000
    {
```

```
        uint16_t three_stage_switch;
        uint16_t start_button;
        uint16_t pause_button;
        uint16_t estop_button;
        uint16_t mode_switch;
        uint16_t servo_prepare;
        uint16_t clear_alarm;
        uint16_t forward_check_button;
        uint16_t backward_check_button;
        uint16_t move_to_point;
        TCMDDATA16 jog_positive;
        TCMDDATA16 jog_negative;
        uint16_t forward_check;
        uint16_t robot_homing;
        uint16_t auto_align;
        TCMDDATA16 reserve[9];
    } hold_cmd;
```

结构体说明:

three_stage_switch:三段开关

start_button:开始按钮

pause_button:暂停按钮

estop_button:急停按钮

mode_switch:模式开关，取值参见全局变量

servo_prepare:伺服准备命令

clear_alarm:清除系统错误

forward_check_button:前进检查硬件按钮

backward_check_button:后退检查硬件按钮

move_to_point:运动到点功能

TCMDDATA16 jog_positive:16 轴正向 jog 命令

TCMDDATA16 jog_negative:16 轴反向 jog 命令

forward_check:前进检查标志，只有在部分界面下才允许前进检查

robot_homing:回零

auto_align:自动平齐

TCMDDATA16 reserve[9]:保留

```
    /// 以下结构体定义的上升沿命令，生效一次后会被自动复位
    struct _X001
    {
  //********************** modbus **********************//
        uint16_t speed_override;
        uint16_t force_do          ;
        uint16_t force_ao          ;
        uint16_t switch_coord    ;
        uint16_t simulate_axis ;
        uint16_t set_kin_limit ;
        uint16_t switch_prog_exec_mode;
        uint16_t regist_system ;
        uint16_t regist_feature_trial;
        uint16_t reset_abs_encoder;
        uint16_t cancel_collision_check;
        uint16_t switch_language;
        uint16_t update_driver_dyn_param;
        TCMDDATA16 reserve1[7];

  //********************** mysql update **********************//
        uint16_t project_var_number;
        uint16_t project_var_string;
        uint16_t project_var_point;
```

```
        uint16_t project_var_inc_point;
        uint16_t project_program;
        uint16_t project_name;

        uint16_t global_var_number;
        uint16_t global_var_string;
        uint16_t global_var_point;
        uint16_t global_var_inc_point;
        uint16_t robot_cartesian_param;
        uint16_t robot_hardware_param;
        uint16_t robot_reference_point_param;
        uint16_t robot_joint_param;
        uint16_t robot_ext_joint_param;
        uint16_t robot_profile_param;
        uint16_t robot_kin_param;
        uint16_t robot_coord_sys;
        uint16_t robot_io_config;
        uint16_t robot_coupling_ratio;
        uint16_t robot_bg_task;
        uint16_t robot_home_pos;
        uint16_t robot_global_workspace;
        uint16_t robot_workspace;
        uint16_t robot_settings;
        uint16_t robot_dynamic_param;
        TCMDDATA16 reserve2[8];
    } trigger_cmd;
} TCommand;
```

结构体说明:

speed_override:调速命令，示教模式下调节示教速度，回放模式下调节回放速度

force_do:强制输出 do

force_ao:强制输出 ao

switch_coord:切换坐标系类型或者 id

simulate_axis:轴仿真命令

set_kin_limit:运动限制命令

switch_prog_exec_mode:切换程序执行模式

regist_system:系统注册命令

regist_feature_trial:注册试用期功能

reset_abs_encoder:绝对值编码器清零

cancel_collision_check:取消防碰撞检查

switch_language:设置返回信息的语言

update_driver_dyn_param:更新驱动器参数

reserve1[7]:保留

project_var_number:当前工程对应的数据库工程数值型变量表发生变更，包括新建，修改，删除

project_var_strin;:当前工程对应的数据库工程字符型变量表发生变更，包括新建，修改，删除

project_var_point:当前工程对应的数据库工程位置型变量表发生变更，包括新建，修改，删除

project_var_inc_point:当前工程对应的数据库工程增量位置型变量表发生变更，包括新建，修改，删除

project_program:当前工程对应的数据库工程数值型变量表发生变更，包括新建，修改，删除

project_name:当前工程的名称发生变更，界面打开了新的工程

global_var_number:数据库中全局数值型变量表发生变更，包括新建，修改，删除

global_var_string:数据库中全局字符型变量表发生变更，包括新建，修改，删除

global_var_point:数据库中全局位置型变量表发生变更，包括新建，修改，删除

global_var_inc_point:数据库中全局增量型变量表发生变更，包括新建，修改，删除

robot_cartesian_param:数据库中机器人参数发生变更，包括新建，修改，删除

robot_hardware_param:数据库中表发生变更，包括新建，修改，删除

robot_reference_point_param:数据库中表发生变更，包括新建，修改，删除

robot_joint_param 数据库中表发生变更，包括新建，修改，删除

robot_ext_joint_param:数据库中全局增量型变量表发生变更，包括新建，修改，删除

robot_profile_param:数据库中全局增量型变量表发生变更，包括新建，修改，删除

robot_kin_param:数据库中全局增量型变量表发生变更，包括新建，修改，删除

robot_coord_sys:数据库中全局增量型变量表发生变更，包括新建，修改，删除

robot_io_config:数据库中全局增量型变量表发生变更，包括新建，修改，删除

robot_coupling_ratio:数据库中耦合比参数表发生变更，包括新建，修改，删除

robot_bg_task:数据库中后台任务参数表发生变更，包括新建，修改，删除

robot_home_pos:数据库中零点参数表发生变更，包括新建，修改，删除

robot_global_workspace:数据库中工作全局空间监控参数表发生变更，包括新建，修改，删除

robot_workspace:数据库中工作空间监控参数表发生变更，包括新建，修改，删除

robot_settings:数据库中系统设置参数表发生变更，包括新建，修改，删除

robot_dynamic_param:数据库中动态负载参数表发生变更，包括新建，修改，删除

TCMDDATA16 reserve2[8]:保留

```
/// task 传给 hmi 的系统状态响应字
typedef struct
{
    /// Y0.0 - Y9.15 corresponds to teach pendent do
    TTBit16 teach_pendent_reserve[10];

    /// Y10.0 - Y189.15 are reserved
    /// reserve[0]
    /// reserve[1]
    uint16_t reserve[8];
} TResponse;
```

结构体说明:

teach_pendent_reserve[10]:当前未使用，保留

reserve[0]:焊接状态，被占用

reserve[1]:当前轴组号，被占用

```
/// 驱动器状态
typedef struct
{
    /// 驱动器状态
    /// 0    :
    /// 1    :
    /// 2    :
    /// 3    :
    /// 4    :
    /// 5    :
    /// 10   :
    /// 200 :
    int16_t state;
```

```
    uint16_t error_code;
} TDriverStatus;
```

结构体说明:

0:初始化状态,驱动器尚未准备好,不允许上伺服操作

1:驱动器尚已准备好,允许用户上伺服操作

2:驱动器处于伺服使能状态

3:驱动器处于停止模式 0

4:驱动器处于停止模式 1

5:驱动器处于停止模式 2

10:驱动器处于故障报警状态

200:驱动器状态未知

error_code:驱动器报警代码，具体参考驱动器

```
/// 轴状态
typedef struct
{
    uint16_t positive_limit: 1;
    uint16_t negative_limit: 1;
    uint16_t is_moving: 1;
    uint16_t over_speeding: 1;
    uint16_t: 12;
} TAxisStatus;
```

结构体说明:

positive_limit:正限位标志位

negative_limit:负限位标志位

is_moving:轴运动标志位

over_speeding:轴超速标志位

2.32系统状态监控参数

```
/// 机器人系统状态结构体，机器人写入，上位机读取
typedef struct InputRegisterData
{
    TResponse    response;
    TAxisInfo    axis_info;
    TRobotInfo    robot_info;
    TExecInfo    exec_info;
    TPlcExecInfo    plc_exec_info[k_MAX_PLC_PROGRAMS];
    TSystemInfo    system_info;
    TIOInfo    io_info;
    TRegistrationInfo    reg_info;
    TFeatureTrialInfo    feature_trial_info;
    TCurrentCoordInfo    coord_info;
    TVersionInfo    version_info;
    TIOConfigInfo    io_config_info;
    TLanguageInfo    language_info;
    TGlobalWorkspaceInfo global_ workspace_info;
    TWorkspaceInfo    workspace_info[k_MAX_WORKSPACE_NUM];
    TDynamicParamInfo    dynamic_param_info;
    uint32_t    magic_number = k_MAGIC_NUMBER;
} InputRegisterData;
```

结构体说明:

Response:system response

axis_info:轴状态信息

robot_info:机器人位置信息

exec_info:主程序执行的信息

plc_exec_info[k_MAX_PLC_PROGRAMS]:plc 程序执行信息

system_info:系统信息

io_info:系统 IO 的信息

reg_info:系统注册信息

feature_trial_info:试用期功能信息

coord_info:当前坐标系信息

version_info:各个模块版本信息

io_config_info:IO 配置状态

language_info:统当前生效语言

global_workspace_info:全局工作空间状态

workspace_info[k_MAX_WORKSPACE_NUM]:工作空间监控状态

dynamic_param_info:各个轴的动态加减速配置

magic_number:用于检查结构体对齐

```
///轴信息
typedef struct
{
    double enc_pos[rcf::k_MAX_AXIS];
    TCMDDATA16 enc_ref_valid;
    double motor_speed[rcf::k_MAX_AXIS];
    double joint_vel[rcf::k_MAX_ROBOT_JOINT];
    double joint_acc[rcf::k_MAX_ROBOT_JOINT];
    double ext_vel[rcf::k_MAX_EXT_JOINT];
    double ext_acc[rcf::k_MAX_EXT_JOINT];
    double tool_lin_vel;
    double tool_lin_acc;
    double tool_ori_vel;
    double tool_ori_acc;
    TAxisStatus axis_status[rcf::k_MAX_AXIS];
    TDriverStatus driver_status[rcf::k_MAX_AXIS];
    TCMDDATA16 axis_sim_state;
```

```
    uint16_t kin_limit_enable;
} TAxisInfo;
```

结构体说明:

enc_pos[rcf::k_MAX_AXIS]:1-16 轴编码器读数

enc_ref_valid:1-16 轴编码器零点是否配置标志位

motor_speed[rcf::k_MAX_AXIS]:单位: rmp/min

joint_vel[rcf::k_MAX_ROBOT_JOINT]:单位: deg/s^2

joint_acc[rcf::k_MAX_ROBOT_JOINT]:单位: deg/s^3

ext_vel[rcf::k_MAX_EXT_JOINT]:单位: deg/s^2

ext_acc[rcf::k_MAX_EXT_JOINT]:单位: deg/s^3

tool_lin_vel:单位: mm/s

tool_lin_acc:单位: mm/s^2

tool_ori_vel:单位: deg/s

tool_ori_acc:单位: deg/s^2

axis_status[rcf::k_MAX_AXIS]:轴状 speed info for teach and replay 态结构体

driver_status[rcf::k_MAX_AXIS]:驱动器状态结构体

axis_sim_state:轴是否处于仿真的状态标志位

kin_limit_enable:限位功能是否开启

```
/// 机器人信息
typedef struct
{
    double acs_pos[rcf::k_MAX_ROBOT_JOINT];
    double ext_pos[rcf::k_MAX_EXT_JOINT];
    double kcs_pos[6];
    double pcs_pos[6];
    double pcs2_pos[6];
    double wcs_pos[6];
    double fcs_pos[6];
```

```
    uint16_t config_num;
    uint16_t ext_group;
} TRobotInfo;
```

结构体说明:

acs_pos[rcf::k_MAX_ROBOT_JOINT]:当前机器人轴位置

ext_pos[rcf::k_MAX_EXT_JOINT]:当前外部轴位置

kcs_pos[6]:当前工具在 KCS 下位置

pcs_pos[6]:当前工具在 PCS 下位置

pcs2_pos[6]:当前工具在 PCS2 下位置

wcs_pos[6]:当前工具在 WCS 下位置

fcs_pos[6]:当前法兰在 KCS 下位置

config_num:当前机器人位置的姿态号

ext_group:当前生效的外部轴号

```
/// 执行信息
typedef struct TExecInfo
{
    char program_name[k_STRING_SIZE_LONG] = {0};
    char project_name[k_STRING_SIZE_LONG] = {0};
    uint32_t exec_line = 0;
    uint16_t reserve = 0;
    uint16_t recover_session;       input_data: 0:
    uint32_t exec_uuid;
} TExecInfo;
```

结构体说明:

program_name[k_STRING_SIZE_LONG]:程序名称

project_name[k_STRING_SIZE_LONG]:工程名称

exec_line:当前正在执行的行号

reserve:保留字段 无意义

recover_session:作业完成,无需恢复，hold_data: 是否需要恢复

exec_uuid:执行行号唯一标识符，只要任务层更新行号该字段一定会发生变更

```
/// 背景任务执行信息
typedef struct TPlcExecInfo
{
    char program_name[k_STRING_SIZE_LONG] = {0};
    uint32_t exec_line = 0;
    uint16_t exec_status = 0;
    uint16_t placeholder = 0;
} TPlcExecInfo;
```

结构体说明:

program_name[k_STRING_SIZE_LONG]:程序名称

exec_line:当前正在执行的行号

exec_status:0:未启动 1：启动

placeholder:字节对齐占位符

```
/// 系统信息
typedef struct
{
    uint64_t loop_count;
    uint64_t cycle_time;
    uint64_t boot_time;
    uint64_t work_time;
    rcf::SystemStatus system_status;
    rcf::ServoStatus servo_status;
    rcf::WorkMode work_mode;
    char warning_message[k_STRING_SIZE_LONG];
```

```
    uint16_t replay_speed;
    uint16_t teach_speed;
    rcf::ProgramExecMode exec_mode;
} TSystemInfo;
```

结构体说明:

loop_count:循环模式下循环次数

cycle_time:循环模式下上一次完整循环的循环时间，unit: ms

boot_time:系统开机时间，unit: ms

work_time:当前程序执行的时间，unit: ms

system_status:机器人系统

servo_status:伺服状态

work_mode:系统工作模式

warning_message[k_STRING_SIZE_LONG]:报警信息

replay_speed:全局回放百分比速度。单位: %

teach_speed:全局示教百分比速度。单位: %

exec_mode:程序执行的模式

```
/// IO 状态，
/// 数字量 IO，区分板载和 ecat，模块 0 表示访问板载，之后表示访问 ecat
/// 模拟量 IO，板载和 ecat 放在一起排序，从板载开始
typedef struct
{
    uint16_t di_logic_local[k_MAX_LOCAL_IO_GROUP];
    uint16_t do_logic_local[k_MAX_LOCAL_IO_GROUP];
    uint16_t di_logic[k_MAX_ECAT_IO_GROUP];
    uint16_t do_logic[k_MAX_ECAT_IO_GROUP];
    double ai_value[k_MAX_AIO_COUNT];
    double ao_value[k_MAX_AIO_COUNT];
} TIOInfo;
```

结构体说明:

di_logic_local[k_MAX_LOCAL_IO_GROUP]: 本地数字量输入

do_logic_local[k_MAX_LOCAL_IO_GROUP]:本地数字量输出

di_logic[k_MAX_ECAT_IO_GROUP]:ecat 数字量输入

do_logic[k_MAX_ECAT_IO_GROUP]:ecat 数字量输出

ai_value[k_MAX_AIO_COUNT]:模拟量输入

ao_value[k_MAX_AIO_COUNT]:模拟量输出

```
/// 系统注册信息
typedef struct
{
    uint32_t registered_apps[rcf::k_MAX_ALLOWED_APPS];
    uint16_t robot_model;
    char user_code[k_STRING_SIZE_SHORT];
    char reg_code[k_STRING_SIZE_MID];
} TRegistrationInfo;
```

结构体说明:

registered_apps[rcf::k_MAX_ALLOWED_APPS]:注册生效的工艺 ID 列表

robot_model:注册生效的机器人模型

user_code[k_STRING_SIZE_SHORT]:机器人系统用户码

reg_code[k_STRING_SIZE_MID]:机器人系统注册码

```
/// 系统试用信息
typedef struct
{
    char is_enabled;
    char is_expired;
    uint16_t days_left;
    char expire_date[k_STRING_SIZE_MID];
```

```
    char trial_code[k_STRING_SIZE_MID];
} TFeatureTrialInfo;
```

结构体说明:

is_enabled:功能是否开启

is_expired:试用是否逾期

days_left:试用剩余天数

expire_date[k_STRING_SIZE_MID]:试用过期日期

trial_code[k_STRING_SIZE_MID]:机器人系统试用码

```
/// 当前坐标系信息
typedef struct
{
    int16_t tcs_id;
    int16_t wcs_id;
    int16_t pcs1_id;
    int16_t pcs2_id;
    rcf::RobotCoord coord;
} TCurrentCoordInfo;
```

结构体说明:

tcs_id: 当前 Tool 序号

wcs_id: 当前 wcs 序号

pcs1_id: 当前 pcs1 序号

pcs2_id: 当前 pcs2 序号

coord: 当前坐标系类型

```
/// 各个模块版本信息
typedef struct
{
```

```
        char
version[k_MAX_ALLOWED_MODULES][k_STRING_SIZE_MID];
        int16_t count;
    } TVersionInfo;
```

结构体说明:

version[k_MAX_ALLOWED_MODULES][k_STRING_SIZE_MID]: 版本信息，每个数组表示一个模块的版本

count:实际注册信息的模块数目，最多 ::k_MAX_ALLOWED_MODULES 个模块

```
    /// IO 配置信息
    typedef struct
    {
        uint16_t collision_check_enable;
        uint16_t reserve[9];
    } TIOConfigInfo;

    typedef struct
    {
        rcf::Language language;
    } TLanguageInfo;
```

结构体说明:

collision_check_enable:碰撞检查是否生效

reserve[9]:保留

```
    /// 全局工作空间信息
    typedef struct RobotGlobalWorkspaceData
    {
    //      uint16_t is_enabled                 : 1;
```

```
    uint16_t is_workspace_triggered : 1;
    uint16_t is_obstacle_triggered  : 1;
    uint16_t is_monitor_triggered   : 1;

    uint16_t : 13;
} TGlobalWorkspaceInfo;
```

结构体说明:

is_enabled:全局工作空间监控是否生效

is_workspace_triggered:工作空间监控是否触发

is_obstacle_triggered:障碍区监控是否触发

is_monitor_triggered:干涉区监控是否触发

```
/// 工作空间信息
typedef struct
{
    uint16_t type            : 2;
    uint16_t is_triggered    : 1;
    uint16_t is_occupied     : 1;
    uint16_t is_motion_paused : 1;
    uint16_t : 11;
} TWorkspaceInfo;
```

结构体说明:

Type:当前类型： 0:不生效 1:工作区 2:障碍区 3: 干涉区

is_triggered:是否触发，true: out of workspace / collide obstacle / in interference area;

is_occupied 如果是干涉区，表示是否进入，否则无效

is_motion_paused:如果是干涉区，表示 motion 是否暂停，否则无效

```
/// 动态参数配置信息
```

```
typedef struct
{
    uint16_t is_enable;
    double joint_acc[rcf::k_MAX_ROBOT_JOINT];
    double joint_dec[rcf::k_MAX_ROBOT_JOINT];
    uint32_t driver_pos_loop_gain[rcf::k_MAX_ROBOT_JOINT];
    uint32_t driver_rapid_dec[rcf::k_MAX_ROBOT_JOINT];
} TDynamicParamInfo;
```

结构体说明:

is_enable:功能是否启用

joint_acc[rcf:k_MAX_ROBOT_JOINT]:轴对应的实际加速度 unit: deg/s/s，0 表示该轴未配置或者该参数无效

joint_dec[rcf::k_MAX_ROBOT_JOINT]:轴对应的实际减速度 unit: deg/s/s，0 表示该轴未配置或者该参数无效

driver_pos_loop_gain[rcf::k_MAX_ROBOT_JOINT]:驱动器位置环增益，0 表示该轴未配置或者该参数无效

driver_rapid_dec[rcf::k_MAX_ROBOT_JOINT]:驱动器急停减速值，0 表示该轴未配置或者该参数无效

```
/// 强制 Do 参数
typedef struct
{
    uint16_t do_module;
    uint16_t do_port;
    uint16_t do_value;
} TForceDoInfo;
```

结构体说明:

do_module:0 表示本地模块，本地模块不设上限，1 以后表示 ethercat 模块，每个模块包含 16 路 io,

do_port:需要强制的端口号，0xFFFF 表示操作整个 module，对于本地范围为 1-16, 对于 ethercat，范围为 1-16

do_value:需要设置的 do 值，0 或者 1

```
/// 强制 Ao 参数
typedef struct
{
    uint16_t ao_id;
    double ao_value;
} TForceAoInfo;
```

结构体说明:

ao_id:ao 的 id 从 1 编号，先板载后 ethercat，需要确定板载规格

ao_value:需求的输出 Ao 值

```
/// 目标位置参数
typedef struct
{
    rcf::RobotCoord coord_type;
    short pcs_idx;
    short wcs_idx;
    short tool_idx;
    double joint_pos[rcf::k_MAX_ROBOT_JOINT];
    double pos[3];
    double ori[3];
    rcf::TRobotPointConfig config;
    double ext_joint[rcf::k_MAX_EXT_JOINT];
    rcf::ExtGroup ext_group;
} TTargetPoint;
```

结构体说明:

coord_type:点坐标的参考系类型类型

pcs_idx:参考工件坐标系的 ID，仅在 coord_type 为工件坐标系生效

wcs_idx:参考世界坐标系的 ID，仅在 coord_type 为工件坐标系生效

tool_idx:点坐标对应的工具坐标系号

joint_pos[rcf::k_MAX_ROBOT_JOINT]:关节位置: deg / mm

pos[3]:笛卡尔位置: unit: mm

ori[3]:位置点姿态，以 RPY(roll-pitch-yaw)表示

config:位置点的姿态号，适用于多解的情况

ext_joint[rcf::k_MAX_EXT_JOINT]:外部轴的位置，是否运动外部轴由 ext_group 指定

ext_group:生效的外部轴组，当运动到点的时候外部轴会随着一起运动

```
/// 运动到点参数
typedef struct
{
    TTargetPoint target;
    int motion_type;
} TMoveToPointInfo;
```

结构体说明:

Target:目标点信息

motion_type:运动类型, 0:joint, 1:cartesian

```
/// 机器人构型
typedef union TRobotPointConfig
{
    typedef struct Asbits
    {
        bool low_wrist: 1;
        bool low_elbow: 1;
```

```
        bool right_shoulder: 1;
        uint8_t: 5;
    } Asbits;
    Asbits    asbits;
    uint8_t asuint8 = 0;
} TRobotPointConfig;
```

结构体说明:

low_wrist:机器人构型：下腕

low_elbow:机器人构型：下肘

right_shoulder:机器人构型：右肩

```
/// 速度命令参数
typedef struct
{
    uint16_t teach_speed;
    uint16_t replay_speed;
} TSpeedCmdInfo;
```

结构体说明:

teach_speed:示教速度百分比

replay_speed:回放速度百分比

```
/// 寸动命令参数
typedef struct
{
    uint16_t is_inche_move;
} TJogInfo;
```

结构体说明:

is_inche_move:是否是寸动

/// 仿真轴参数

typedef struct

{

TCMDDATA16 axis_sim_flag;

} TSimulateAxisInfo;

结构体说明:

axis_sim_flag:1-16 轴是否仿真的标志位,系统后续支持更多轴可以继续增加

/// DH 设置参数

typedef struct

{

uint16_t is_enable;

} TSetKinLimitInfo;

结构体说明:

is_enable:是否开启该功能，采用结构体方便以后增加更多的参数

/// 自动平齐参数

typedef struct

{

uint16_t align_a;

uint16_t align_b;

uint16_t align_c;

} TAutoAlignInfo;

结构体说明:

align_a:是否平齐 a 方向的分量

align_b:是否平齐 b 方向的分量

align_c:是否平齐 c 方向的分量

/// 注册系统参数

typedef struct

```
{
    char reg_code[k_STRING_SIZE_MID];
} TRegistSystemInfo;
```

结构体说明:

reg_code[k_STRING_SIZE_MID]:注册码

```
/// 试用码参数
typedef struct
{
    char reg_code[k_STRING_SIZE_MID];
} TRegistFeatureTrialInfo;
```

结构体说明:

reg_code[k_STRING_SIZE_MID]:试用码

```
/// 重置绝对值编码器参数
typedef struct
{
    TCMDDATA16 enc_reset_flag;
} TResetAbsEncoderInfo;
```

结构体说明:

enc_reset_flag:1-16 轴是否清零标志位

```
/// 连接信息参数
typedef struct
{
    uint32_t heartbeat;
} TConnectionInfo;
```

结构体说明:

Heartbeat:心跳变量，由当前设备自动增加，超过 200ms 未变化表示物理断线

```
/// 上位机命令结构体，上位机写入，控制器读取
typedef struct HoldingRegisterData
{
    TCommand command;
    rcf::ProgramExecMode exec_mode;
    TExecInfo prog_info;
    TSpeedCmdInfo speed_info;
    TJogInfo jog_info;
    TForceDoInfo force_do_info;
    TForceAoInfo force_ao_info;
    TCurrentCoordInfo switch_coord_info;
    TMoveToPointInfo move_to_point_info;
    TSimulateAxisInfo simulate_axis_info;
    TSetKinLimitInfo set_kin_limit_info;
    TAutoAlignInfo auto_align_info;
    TRegistSystemInfo regist_system_info;
    TRegistFeatureTrialInfo regist_feature_trial_info;
    TResetAbsEncoderInfo reset_abs_enc_info;
    TConnectionInfo device_info;
    TLanguageInfo langage_info;
    uint32_t magic_number = k_MAGIC_NUMBER;
} HoldingRegisterData;
```

结构体说明:

Command:外部命令

exec_mode:程序执行模式

prog_info:执行的程序信息

speed_info:调速信息

jog_info:Jog 信息

force_do_info:数字量强制信息

force_ao_info:模拟量强制信息

switch_coord_info:切换坐标系信息

move_to_point_info:运动到点信息

simulate_axis_info:仿真轴信息

set_kin_limit_info:超限检查

auto_align_info:自动平齐参数

regist_system_info:注册信息

regist_feature_trial_info:试用码信息

reset_abs_enc_info:绝对值编码器清零信息

device_info:心跳信息

langage_info:语言信息

magic_number:用于检查结构体对齐

```
///监控变量
    static constexpr int WATCH_VAR_COUNT = 10;
    typedef struct UdpStruct
    {
        imf::rcf::InputRegisterData status;
        struct
        {
            double number[WATCH_VAR_COUNT];
        } watch;
    } UdpStruct;
```

2.33报警信息配置

```
namespace rcf
{
typedef struct
{
    uint32_t code = 0;
    //short level;
    //std::string source;
    std::string message = "";
    std::string timestamp = "";
} CRobotError;
```

结构体说明:

Code: error code of log

Level: level of the log

Source: source of the log

Message: message of the log

Timestamp: date time of event, format "yyyy-MM-dd hh:mm:ss.zzz"}

```
enum ErrorCode
{
    CONNECTION_ERROR = 0X1000,
    RESPONSE_TIMEOUT_ERROR = 0X2000,
    RUNTIME_ERROR = 0X3000,
    EXECUTION_ERROR = 0X4000,
};
```

2.34枚举类型

```
enum class WorkMode:enum_type{Teach = 0,Replay,Remote};
```

机器人工作模式枚举类说明：

Teach：示教模式

Replay：回放模式

Remote：远程模式

enum class RobotCoord: enum_type {*ACS* = 0, *KCS*, *PCS*, *WCS*, *TCS*};
坐标系枚举类说明:

ACS：关节坐标系，只有一个

KCS：机器人坐标系，只有一个

PCS：工件坐标系，预留 64 个

WCS：世界坐标系，预留 64 个

TCS：工具坐标系，预留 64 个

enum class RobotVar : enum_type {*Number* = 0, *Point*, *IncPoint*, *String*};
机器人变量类型枚举类说明：

Number ：数值型变量

Point：位置型变量

IncPoint：增量位置型变量

String：字符串变量

enum class SingularityPolicy: enum_type{*WarnAndStop* = 0,*SlowDown*};
奇异点策略枚举类说明:

WarnAndStop：运动过程中碰到奇异点时选择立即停止程序并报警

SlowDown：运动过程中碰到奇异点时选择减速缓慢通过直至无法通过报警

enum class AxisUnit : enum_type {*NOT_USED* = 0, *DEG, MM*};
轴类型枚举类说明:

NOT_USED： 关节轴未使用

DEG： 关节轴为旋转类型

MM： 关节轴为平移类型

enum class ProfileType : enum_type{*Cos* = 0, *Quintic*, *Door*, *SawTooth*, *Exp*, *Bell*};

规划器类型枚举类说明:

Cos： Jerk 是 Cos 函数的轨迹规划

Quintic： Jerk 是 5 次曲线的轨迹规划

Door： Jerk 是方波的轨迹规划

SawTooth： Jerk 是锯齿形的轨迹规划

Exp： Jerk 是指数曲线的轨迹规划

Bell： Jerk 是二次曲线的轨迹规划

enum class BlendType : enum_type{*Fusion* = 0, *Spline, VelBlend*};

过渡类型枚举类说明:

Fusion： 路径混合过渡，速度和加速度不突变，轨迹可控，但速度无法恒定，默认选项

Spline： 样条过渡，保证过渡段速度恒定，轨迹可控，但会导致加速度突变，突变小于 Arc 过渡。不适合关节轨迹和姿态规划

VelBlend： 速度过渡，即两条运动指令同时规划，进行速度叠加，得到新的位置。轨迹形状和进入 Blend 段时速度相关

enum class CoordSysAttribute: rcf::enum_type{*Normal* = 0,*ExtTCP* };

坐标系的附加属性枚举类说明：

Normal： 正常使用

ExtTCP： 作为外部 TCP

encm class WorkSpaceMode{*Disabled*=0,*ReplayOnly*,*TeachOnly*,
ReplayAndTeach};
工作空间监控模式枚举类说明：

Disabled：不生效

ReplayOnly：仅在回放模式生效

TeachOnly：仅在示教模式生效

ReplayAndTeach：示教/回放模式生效

enum class WorkSpaceType{*WorkArea* = 1, *ObstacleArea*, *MonitorArea* };
工作空间监控区域类型枚举类说明：

WorkArea：工作区

ObstacleArea：障碍区

MonitorArea：监控区

enum class EndPointType{*ToolTip* = 1,*FlangeCenter*};
工作空间监控点类型枚举类说明：

ToolTip：工具尖点

FlangeCenter：法兰中心

enum class WorkSpaceShape{*Sphere* = 1,*Cylinder*,*Cube*};
工作空间监控区形状类型枚举类说明：

Sphere：工具尖点

Cylinder：法兰中心

Cube：立方体

enum class RemoteControlMode: rcf::enum_type{*ModbusTcp* =0,
IO,*CustomTcp*,*ModbusSerial*};

机器人远程控制模式枚举类说明：

ModbusTcp：远程 modbus TCP 控制

IO：远程 IO 控制

CustomTcp：远程上位机 TCP 控制

ModbusSerial：远程 modbus 串口控制

enum class Movc2TeachCheckBehavior: rcf::enum_type {*Linear*,*Circular*};
机器人 Movc2 Jog 模式运动模式枚举类说明：

Linear：直线运动

Circular：圆弧运动

enum class LineMode: rcf::enum_type{*FirstLine*, *LastStopped*};
机器人启动模式枚举类说明：

FirstLine：首行启动

LastStopped：继续上次停止行启动

enum class TriggerMode:rcf::enum_type{*PowerOn*,*FollowMain*};
机器人背景任务触发模式枚举类说明：

PowerOn：开机启动

FollowMain：和主程序保持一致

enum class ExtGroup: enum_type{*Ungrouped* = 0,*ExtGroupA* = 0x01 << 0,
ExtGroupB = 0x01 << 1,*ExtGroupC* = 0x01 << 2,*ExtGroupD* = 0x01 << 3,
ExtGroupAB = ExtGroupA | ExtGroupB,*ExtGroupAC* = ExtGroupA | ExtGroupC,
ExtGroupAD = ExtGroupA | ExtGroupD,*ExtGroupBC* = ExtGroupB | ExtGroupC,

ExtGroupBD = ExtGroupB | ExtGroupD,*ExtGroupCD* = ExtGroupC | ExtGroupD,

ExtGroupABC = ExtGroupA | ExtGroupB | ExtGroupC,*ExtGroupABD* = ExtGroupA | ExtGroupB | ExtGroupD,

ExtGroupACD = ExtGroupA | ExtGroupC | ExtGroupD,*ExtGroupBCD* = ExtGroupB | ExtGroupC | ExtGroupD,

ExgGroupABCD= ExtGroupA | ExtGroupB | ExtGroupC | ExtGroupD,};

机器人外部轴组枚举类说明：

支持最多 4 个轴组，支持多个轴组同时运动

enum class ProgramExecMode : enum_type{*Once* = 0, *Step*,*Loop*};

机器人程序执行模式枚举类说明：

Once：单次模式

Step：单步模式

Loop：循环模式

enum DataType {*ReadOnly*, *ReadWrite*};

读写工艺模式枚举类说明：

ReadOnly：只读

ReadWrite：可读可写

enum class ValueType : rcf::enum_type{*Payload*,*Bias*,*Inertia*};

机器人驱动器记录对应的变量类型枚举类说明：

Payload：负载

Bias：工具偏移

Inertia：惯量

3 指令列表

3.1 指令列表一览

机器人接口函数列表	
connect	建立通讯
disConnect	断开通讯
isConnected	通讯状态返回
cmdServoOn	根据系统的轴配置，上伺服
cmdServoOff	根据系统的轴配置，下伺服
cmdStop	暂停
cmdClear	清除警告/报警
cmdSetWorkMode	设置当前机器人系统工作模式
cmdJogOn	控制机器人在指定坐标系下开始 jog 运动
cmdJogOff	控制机器人在指定坐标系下停止 jog 运动
cmdSetJogCoordIdx	切换系统生效的工件/工具坐标系编号
cmdSetJogCoordType	切换系统生效的坐标系类型
cmdMoveToPoint	控制机器人移动到目标点
cmdSetJogSpeedPercent	设置 Jog 速度百分比
cmdSetReplaySpeedPercent	设置回放速度百分比
cmdStart	执行示教程序
cmdPause	暂停执行程序
cmdResume	(中断后)继续执行程序
cmdRecove	重新恢复运行程序
cmdMoveRobotHome	机器人回零运动
cmdSimulateAxis	机器人轴仿真
cmdEnableKinLimit	启用关节限位
getRegisterData	获取系统状态及参数
getApiVersion	获取 api 版本
collectRobotPoint	开启采集机器人点位
stopCollectRobotPoint	停止采集机器人点位
getCollectRobotPoint	获取机器人点位信息
getRobotParam	按照 id 获取机器人配置参数

setRobotParam	按照 id 设置机器人配置参数
getProgram	读取指定程序
setProgram	保存指定程序
loadProgram	直接从数据库中读取指定工程下的所有程序到缓存，耗时操作~100ms 左右
getProjectVar	读取工程变量
setProjectVar	写入工程变量
loadProjectVar	从数据库中读取工程变量到缓存，耗时操作~100ms 左右
getPlcProgram	获取指定后台 plc 程序
setPlcProgram	设置指定 plc 程序
getPlcVar	获取 plc 程序变量
setPlcVar	设置 plc 程序变量
getGlobalVar	获取指定类型的全局变量
setGlobalVar	设置指定类型的全局变量
loadGlobalVar	从数据库中加载全部全局变量，耗时操作~100ms 左右
createRobotProject	如果工程已经存在，那么改接口直接返回 true 否则创建一个默认工程
getremoteworkmode	获取机器人远程模式（远程-示教；远程-回放）
getRobotParam	按照 id 获取机器人配置参数
setRobotParam	设置机器人配置参数
getDiBit	按位获取输入数字量
getDoBit	按位获取输出数字量
getDiMask	按掩码获取输入数字量
getDoMask	按掩码获取输出数字量
getAo	获取模拟输出量
getAi	获取模拟输入量
setAo	设置模拟量
setDoBit	按位设置输出数字量
setDoMask	按掩码设置输出数字量
getAppData	以十六进制形式获取工艺信息
setAppData	以十六进制形式设置工艺信息
getAppTData	获取工艺信息

setAppTData	设置工艺信息
getModbusData	获取控制器 modbus 数据
setModbusData	设置控制器 modbus 数据
getMysqlData	获取数据库信息
setMysqlData	设置数据库信息

3.2 连接管理接口

本节提供对机器人 TCP 连接的接口，具体接口如下：

3.2.1 connect

指令原型	**bool connect**(std::string ip, uint16_t tcp_port = 23333, uint16_t udp_port = 23334);
指令说明	建立通讯
指令参数	ip：固定端口为 23333
	tcp_port：默认 TCP 端口号 23333
	udp_port：默认 UDP 端口号 23334
指令返回值	Ture　false
注意事项	无
使用示例	Mainwindow.h imf::RobotController pTcpApi; Mainwindow.cpp pTcpApi.connect("192.168.101.3"，23333，23334);

3.2.2 disConnect

指令原型	**bool disConnect**();
指令说明	断开通讯
指令参数	无
指令返回值	Ture　false
注意事项	无
使用示例	Mainwindow.h imf::RobotController pTcpApi;

	Mainwindow.cpp pTcpApi.disConnect();

3.2.3 isConnected

指令原型	**bool isConnected**();
指令说明	通讯状态返回
指令参数	无
指令返回值	Ture false
注意事项	无
使用示例	Mainwindow.h imf::RobotController pTcpApi; Mainwindow.cpp pTcpApi.isConnected();

3.3 运动控制接口

本节提供对机器人运动控制的接口，具体接口如下

3.3.1 cmdServoOn

指令原型	**bool cmdServoOn**();
指令说明	上伺服
指令参数	无
指令返回值	执行是否成功
注意事项	仅在远程模式下调用
使用示例	Mainwindow.h imf::RobotController pTcpApi; Mainwindow.cpp pTcpApi.cmdServoOn();

3.3.2 cmdServoOff

指令原型	**bool cmdServoOff**();
指令说明	下伺服
指令参数	无

指令返回值	执行是否成功
注意事项	仅在远程模式下调用
使用示例	Mainwindow.h imf::RobotController pTcpApi; Mainwindow.cpp pTcpApi.cmdServoOff();

3.3.3 cmdStop

指令原型	**bool cmdStop**();
指令说明	暂停
指令参数	无
指令返回值	执行是否成功
注意事项	仅在远程-回放模式下调用
使用示例	Mainwindow.h imf::RobotController pTcpApi; Mainwindow.cpp pTcpApi.cmdStop();

3.3.4 cmdClear

指令原型	**bool cmdClear**();
指令说明	清除警告/报警
指令参数	无
指令返回值	执行是否成功
注意事项	无
使用示例	Mainwindow.h imf::RobotController pTcpApi; Mainwindow.cpp pTcpApi.cmdClear();

3.3.5 cmdSetWorkMode

指令原型	**bool cmdSetWorkMode**(rcf::WorkMode mode);

指令说明	设置当前机器人系统工作模式
指令参数	mode:工作模式
指令返回值	执行是否成功
注意事项	仅在远程-示教模式下调用
使用示例	Mainwindow.h imf::RobotController pTcpApi; Mainwindow.cpp pTcpApi.cmdSetWorkMode(imf::rcf::WorkMode::Teach);

3.3.6 cmdJogOn

指令原型	**bool cmdJogOn**(unsigned char axis, char dir,int auto_off=1000);
指令说明	控制机器人在指定坐标系下开始 jog 运动
指令参数	axis: jog 的轴号
	dir: jog 的方向（1 正方向，-1 是负方向）
	auto_off:jog 运动的时间，单位 mm/s
指令返回值	执行是否成功
注意事项	仅在远程-示教模式下调用
使用示例	Mainwindow.h imf::RobotController pTcpApi; Mainwindow.cpp pTcpApi.cmdJogOn(1,-1,1000);

3.3.7 cmdJogOff

指令原型	**bool cmdJogOff**(unsigned char axis);
指令说明	控制机器人在指定坐标系下停止 jog 运动
指令参数	Axis:jog 的轴号
指令返回值	执行是否成功
注意事项	仅在远程-示教模式下调用
使用示例	Mainwindow.h imf::RobotController pTcpApi; Mainwindow.cpp pTcpApi.cmdJogOff(1);

3.3.8 cmdSetJogCoordIdx

指令原型	**bool cmdSetJogCoordIdx**(rcf::RobotCoord type, uint16_t index = 1);
指令说明	切换系统生效的工件/工具坐标系编号
指令参数	type:目标坐标系
	index:目标坐标系 ID
指令返回值	执行是否成功
注意事项	仅在远程-示教模式下调用
使用示例	Mainwindow.h imf::RobotController pTcpApi; Mainwindow.cpp pTcpApi.cmdSetJogCoordIdx(imf::rcf::RobotCoord::PCS,1); ;

3.3.9 cmdSetJogCoordType

指令原型	**bool cmdSetJogCoordType**(rcf::RobotCoord type);
指令说明	切换系统生效的坐标系类型
指令参数	type:目标坐标系
指令返回值	执行是否成功
注意事项	仅在远程-示教模式下调用
使用示例	Mainwindow.h imf::RobotController pTcpApi; Mainwindow.cpp pTcpApi.cmdSetJogCoordType(imf::rcf::RobotCoord::ACS); ;

3.3.10 cmdMoveToPoint

指令原型	**bool cmdMoveToPoint**(const rcf::TMoveToPointInfo &info);
指令说明	控制机器人移动到目标点
指令参数	info:目标点信息结构体

指令返回值	执行是否成功
注意事项	仅在远程-回放模式下调用
使用示例	Mainwindow.h imf::RobotController pTcpApi; Mainwindow.cpp imf::rcf::TMoveToPointInfo TMoveToPoint; TMoveToPoint.motion_type=0; TMoveToPoint.target.config.asbits.low_elbow = 1; TMoveToPoint.target.config.asbits.low_wrist = 1; TMoveToPoint.target.config.asbits.right_shoulder = 1; TMoveToPoint.target.coord_type=imf::rcf::RobotCoord ::ACS; TMoveToPoint.target.ext_group=imf::rcf::ExtGroup ::Ungrouped; TMoveToPoint.target.ext_joint[1]=100; TMoveToPoint.target.joint_pos[1]=100; TMoveToPoint.target.ori[3]=100; TMoveToPoint.target.pos[3]=28; TMoveToPoint.target.pcs_idx=28; TMoveToPoint.target.tool_idx=28; TMoveToPoint.target.wcs_idx=28; pTcpApi.cmdMoveToPoint(TMoveToPoint);

3.3.11 cmdSetJogSpeedPercent

指令原型	**bool cmdSetJogSpeedPercent**(double percent);
指令说明	设置 Jog 速度百分比
指令参数	percent:Jog 百分比，范围[1-100]
指令返回值	执行是否成功
注意事项	仅在远程-示教模式下调用
使用示例	Mainwindow.h imf::RobotController pTcpApi; Mainwindow.cpp pTcpApi.cmdSetJogSpeedPercent(100);

3.3.12 cmdSetReplaySpeedPercent

指令原型	**bool cmdSetReplaySpeedPercent**(double percent);
指令说明	设置回放速度百分比
指令参数	percent:Jog 百分比，范围[1-100]
指令返回值	执行是否成功
注意事项	仅在远程-示教模式下调用
使用示例	Mainwindow.h imf::RobotController pTcpApi; Mainwindow.cpp pTcpApi.cmdSetReplaySpeedPercent(100);

3.3.13 cmdStart

指令原型	**bool cmdStart**(rcf::TExecInfo info, rcf::ProgramExecMode mode);
指令说明	执行示教程序（已经执行的程序状态将会被覆盖）
指令参数	info:需要执行的目标程序信息
	mode:需要执行的模式（单步，单次，或者循环）
指令返回值	执行是否成功
注意事项	仅在远程-回放模式下调用

使用示例	Mainwindow.h
	imf::RobotController pTcpApi;
	Mainwindow.cpp
	imf::rcf::ProgramExecMode enum_type;
	enum_type=imf::rcf::ProgramExecMode::Once;
	imf::rcf::TExecInfo TExe;
	QString project_name ("工程");
	QByteArray ject= project_name.toUtf8();
	memcpy(TExe.project_name,ject,100);
	QString program_name ("程序");
	QByteArray gram= program_name.toUtf8();
	memcpy(TExe.program_name,gram,100);
	pTcpApi.cmdSetJogSpeedPercent(50);
	pTcpApi.cmdSetJogSpeedPercent(50);

	pTcpApi.cmdStart(TExe,enum_type);

3.3.14 cmdPause

指令原型	**bool cmdPause**();
指令说明	(暂停后)继续执行程序,调用该命令会控制 motion 继续执行程序，而不会清空缓冲区
指令参数	无
指令返回值	执行是否成功
注意事项	仅在远程-回放模式下调用
使用示例	Mainwindow.h imf::RobotController pTcpApi; Mainwindow.cpp pTcpApi.cmdPause();

3.3.15 cmdResume

指令原型	**bool cmdResume**();
指令说明	(中断后)继续执行程序,调用该命令会控制 motion 继续执行程序，而不会清空缓冲区
指令参数	无
指令返回值	执行是否成功
注意事项	仅在远程-回放模式下调用
使用示例	Mainwindow.h imf::RobotController pTcpApi; Mainwindow.cpp pTcpApi.cmdResume();

3.3.16 cmdRecover

指令原型	**bool cmdRecover**();
指令说明	重新恢复运行程序
指令参数	无
指令返回值	执行是否成功

注意事项	仅在远程-回放模式下调用
使用示例	Mainwindow.h imf::RobotController pTcpApi; Mainwindow.cpp pTcpApi.cmdRecover();

3.3.17 cmdMoveRobotHome

指令原型	**bool cmdMoveRobotHome**();
指令说明	机器人回零运动
指令参数	无
指令返回值	执行是否成功
注意事项	仅在远程-回放模式下调用
使用示例	Mainwindow.h imf::RobotController pTcpApi; Mainwindow.cpp pTcpApi.cmdMoveRobotHome();

3.3.18 cmdSimulateAxis

指令原型	**bool cmdSimulateAxis**(uint16_t axis = 0x00);
指令说明	机器人轴仿真
指令参数	axis:按位仿真掩码，1 表示仿真，0 表示不处理，0x00FF 表示 1-8 轴进入仿真，0x0F 表示只把 1-4 轴仿真
指令返回值	执行是否成功
注意事项	仅在远程-回放模式下调用
使用示例	Mainwindow.h imf::RobotController pTcpApi; typedef struct { uint16_t bit0: 1; uint16_t bit1: 1; uint16_t bit2: 1; uint16_t bit3: 1; uint16_t bit4: 1;

	uint16_t bit5: 1;
	uint16_t bit6: 1;
	uint16_t bit7: 1;
	uint16_t bit8: 1;
	uint16_t bit9: 1;
	uint16_t bitA: 1;
	uint16_t bitB: 1;
	uint16_t bitC: 1;
	uint16_t bitD: 1;
	uint16_t bitE: 1;
	uint16_t bitF: 1;
	} TTBit16;
	typedef union
	{
	int16_t data;
	uint16_t uData;
	TTBit16 cmd;
	} TCMDDATA16;
	Mainwindow.cpp
	pTcpApi.cmdSimulate(simulate_info.axis_sim_flag.uData);

3.3.19 cmdEnableKinLimit

指令原型	**bool cmdEnableKinLimit**(bool enable = true);
指令说明	启用关节限位
指令参数	enable:是否启用标志位
指令返回值	执行是否成功
注意事项	仅在远程-回放模式下调用
使用示例	Mainwindow.h imf::RobotController pTcpApi; Mainwindow.cpp pTcpApi.cmdSimulate(true);

3.4 系统状态监控接口

本节提供对机器人实时点位信息的读取，具体接口如下

3.4.1 getRegisterData

指令原型	**bool getRegisterData**(imf::rcf::UdpStruct&status);
指令说明	获取系统状态及参数
指令参数	status:对应系统结构体（详细可看第三章节）
指令返回值	执行是否成功
注意事项	无

使用示例	Mainwindow.h
	imf::RobotController pTcpApi;
	Mainwindow.cpp
	imf::rcf::UdpStruct Struct;
	pTcpApi.getRegisterData(Struct);
	qDebug()<<Struct.status.coord_info.pcs1_id;
	qDebug()<<Struct.status.feature_trial_info.trial_code;

3.4.2 getApiVersion

指令原型	**std::string getApiVersion**();
指令说明	获取 api 版本
指令参数	无
指令返回值	执行是否成功
注意事项	无

使用示例	Mainwindow.h
	imf::RobotController pTcpApi;
	Mainwindow.cpp
	std::string versionStd = getApiVersion();
	QString versionQt = QString::fromStdString(versionStd);
	qDebug() << "API Version (Qt):" << versionQt;

3.4.3 collectRobotPoint

指令原型	**bool collectRobotPoint**(int period = 4);
指令说明	开启采集机器人点位
指令参数	Period:相对于总线通讯周期的倍率（默认 EtherCat 周期 1ms）
指令返回值	执行是否成功
注意事项	系统采集最小周期 4ms；周期设置应大于 4ms

使用示例	Mainwindow.h imf::RobotController pTcpApi; Mainwindow.cpp int period = 4； pTcpApi.collectRobotPoint(period);

3.4.4 stopCollectRobotPoint

指令原型	**bool stopCollectRobotPoint**();
指令说明	停止采集机器人点位
指令参数	无
指令返回值	执行是否成功
注意事项	无

使用示例	Mainwindow.h
	imf::RobotController pTcpApi;
	Mainwindow.cpp
	int period = 4；
	pTcpApi.collectRobotPoint(period);
	pTcpApi.stopCollectRobotPoint();

3.4.5 getCollectRobotPoint

指令原型	**bool getCollectRobotPoint**(std::vector < rcf::RtPointInfo > &point, ulong &count);
指令说明	获取机器人点位信息
指令参数	point：机器人点位
	count：点位数量
指令返回值	执行是否成功
注意事项	正在扫描时，下发该指令会失败；count 点位数量最多十万个位置点

使用示例	Mainwindow.h
	imf::RobotController pTcpApi;
	Mainwindow.cpp
	int period = 4；
	pTcpApi.collectRobotPoint(period);
	pTcpApi.stopCollectRobotPoint();

	std::vector<imf::rcf::RtPointInfo>point;
	ulong pointCount = 1;
	pTcpApi. getCollectRobotPoint(point, pointCount);

3.5 资源变量读写接口

资源类型包括：程序，工程变量，全局变量，后台 PLC 任务。
本节提供对这些参数读写的接口，具体接口如下：

3.5.1 getProgram

指令原型	**bool getProgram**(rcf::TRobotProgram &program, std::string project, std::string name);
指令说明	读取指定程序
指令参数	program:程序数据
	project:需要读取的工程名
	name:需要读取的程序名
指令返回值	执行是否成功
注意事项	该接口优先从缓存中查找，如果不存在则会从数据库中加载，此时等同于调用 bool loadProgram(TRobotProgram &program, QString project, QString name); 如果用户确定不会修改数据库，而只是需要在不同工程之间访问，采用缓存机制可以极大的优化访问速度
使用示例	Mainwindow.h imf::RobotController pTcpApi; Mainwindow.cpp imf::rcf::TRobotProgram Program; QString str1 = "工程"; QString str2 = "程序"; pTcpApi.getProgram(Program,str1.toStdString(), str2.toStdString());

3.5.2 setProgram

指令原型	**bool setProgram**(const rcf::TRobotProgram &program,

	std::string project);
指令说明	保存指定程序
指令参数	program:需要保存的工程
	project:需要保存到的工程名
指令返回值	执行是否成功
注意事项	该接口优先从缓存中查找，如果不存在则会从数据库中加载，此时会先调用 bool loadProgram(QString project);
使用示例	Mainwindow.h imf::RobotController pTcpApi; Mainwindow.cpp imf::rcf::TRobotProgram program; std::string project = "工程"; program.name = "程序"; program.content = "PROCEDURE\n Movj P=#GP1 V=50% Bl=0%;\nENDPROCEDURE" pTcpApi.setProgram(program, project);

3.5.3 loadProgram

指令原型	**bool loadProgram**(std::string project);
指令说明	直接从数据库中读取指定工程下的所有程序到缓存，耗时操作~100ms 左右
指令参数	project:需要读取的工程名字
指令返回值	执行是否成功
注意事项	如果指定工程没有加载，那么该接口将对工程进行缓存。如果已经缓存则仅仅会更新工程中的程序部分
使用示例	Mainwindow.h imf::RobotController pTcpApi; Mainwindow.cpp QString strProject = "工程"; pTcpApi.loadProgram(strProject.toStdString());

3.5.4 getProjectVar

指令原型	**bool getProjectVar**(rcf::TRobotVarPoint &item, std::string project, int id);
指令说明	读取工程变量，需要确保该工程正常存在
指令参数	item:工程变量
	project:工程名字
	id:自然编号，从 1 开始
指令返回值	执行是否成功
注意事项	该接口优先从缓存中查找，如果不存在则会从数据库中加载，此时等同于调用 bool loadProjectVar(RobotVar type, QString project);

使用示例	Mainwindow.h
	imf::RobotController pTcpApi;
	Mainwindow.cpp
	imf::rcf::TRobotVarIncPoint VarPoint;
	QString strProject = "工程";
	int id=11;
	pTcpApi.getProjectVar(VarPoint,strProject.toStdString(),id);

3.5.5 setProjectVar

指令原型	**bool setProjectVar**(const rcf::TRobotVarPoint &item, std::string project);
指令说明	写入工程变量，需要确保该工程正常存在
指令参数	item:工程变量
	project:工程名字
指令返回值	执行是否成功
注意事项	该接口优先从缓存中写入，如果不存在则会从数据库中加载，此时会先调用 bool loadProjectVar(RobotVar type, QString project);
使用示例	Mainwindow.h imf::RobotController pTcpApi; Mainwindow.cpp imf::rcf::TRobotVarPoint item;

	std::string project = "工程";
	item.config.asuint8 = 1;
	item.coord_type = imf::rcf::RobotCoord::ACS;
	item.tool_idx = 1;
	item.pcs_idx = 1;
	item.wcs_idx = 0;
	item.joint_pos[0] = -58;
	item.joint_pos[1] = 3.4;
	item.joint_pos[2] = 0.4;
	item.joint_pos[3] = 3.5;
	item.joint_pos[4] = 90;
	item.joint_pos[5] = -27;
	item.id = 1;
	pTcpApi.setProjectVar(item, project);

3.5.6 loadProjectVar

指令原型	**bool loadProjectVar**(rcf::RobotVar type, std::string project);
指令说明	从数据库中读取工程变量到缓存，耗时操作~100ms 左右
指令参数	type:变量类型
	project:需要读取的工程名
指令返回值	执行是否成功
注意事项	如果指定工程没有加载，那么该接口将对工程进行缓存。如果已经缓存则仅仅会更新工程中的相应部分
使用示例	Mainwindow.h imf::RobotController pTcpApi; Mainwindow.cpp QString strProject = "工程"; pTcpApi.loadProjectVar(imf::rcf::RobotVar::Point,strProject.toStdString());

3.5.7 getPlcProgram

指令原型	**bool getPlcProgram**(rcf::TRobotProgram &program, std::string name);

指令说明	获取指定后台 plc 程序
指令参数	program:后台 plc 程序结构体
	name：后台 plc 程序名称
指令返回值	执行是否成功
注意事项	无
使用示例	Mainwindow.h imf::RobotController pTcpApi; Mainwindow.cpp imf::rcf::TRobotProgram Program; std::string name="程序"; pTcpApi.getPlcProgram(Program, name);

3.5.8 setPlcProgram

指令原型	**bool setPlcProgram**(const rcf::TRobotProgram &new_prog);
指令说明	设置指定 plc 程序
指令参数	new_prog:程序名称
指令返回值	执行是否成功
注意事项	无
使用示例	Mainwindow.h imf::RobotController pTcpApi; Mainwindow.cpp imf::rcf::TRobotProgram Program; Program.last_modify_date="2023.11.11"; Program.content"程序内容"; pTcpApi.setPlcProgram(Program);

3.5.9 getPlcVar

指令原型	**bool getPlcVar**(rcf::TRobotVarPoint &item, int id);
指令说明	获取 plc 程序变量
指令参数	item:变量结构体
	id:需要读取的工程变量 id，从 1 开始
指令返回值	执行是否成功

注意事项	无
使用示例	Mainwindow.h imf::RobotController pTcpApi; Mainwindow.cpp imf::rcf::TRobotVarPoint VarPoint; pTcpApi.getPlcProgram(VarPoint);

3.5.10 setPlcVar

指令原型	**bool setPlcVar**(const rcf::TRobotVarPoint &item);
指令说明	设置 plc 程序变量
指令参数	item:变量结构体
指令返回值	执行是否成功
注意事项	无
使用示例	Mainwindow.h imf::RobotController pTcpApi; Mainwindow.cpp imf::rcf::TRobotVarPoint VarPoint; pTcpApi.setPlcProgram(VarPoint);

3.5.11 getGlobalVar

指令原型	**bool getGlobalVar**(rcf::TRobotVarPoint &item, int id);
指令说明	获取指定类型的全局变量
指令参数	item:变量结构体
	id:全局变量，从 1 开始
指令返回值	执行是否成功
注意事项	无
使用示例	Mainwindow.h imf::RobotController pTcpApi; Mainwindow.cpp imf::rcf::TRobotVarPoint VarPoint; pTcpApi.getGlobalVar(VarPoint);

3.5.12 setGlobalVar

指令原型	**bool setGlobalVar**(const rcf::TRobotVarPoint &item);
指令说明	设置指定类型的全局变量
指令参数	item 变量结构体
指令返回值	执行是否成功
注意事项	无

使用示例	Mainwindow.h
	imf::RobotController pTcpApi;
	Mainwindow.cpp
	imf::rcf::TRobotVarPoint item;
	std::string project = "工程";
	item.config.asuint8 = 1;
	item.coord_type = imf::rcf::RobotCoord::ACS;
	item.tool_idx = 1;
	item.pcs_idx = 1;
	item.wcs_idx = 0;
	item.joint_pos[0] = -58;
	item.joint_pos[1] = 3.4;
	item.joint_pos[2] = 0.4;
	item.joint_pos[3] = 3.5;
	item.joint_pos[4] = 90;
	item.joint_pos[5] = -27;
	item.id = 1;
	pTcpApi.setGlobalVar(item, project);

3.5.13 loadGlobalVar

指令原型	**bool loadGlobalVar**(rcf::RobotVar type);
指令说明	从数据库中加载全部全局变量，耗时操作~100ms 左右
指令参数	type:数据类型
指令返回值	执行是否成功
注意事项	常用于外部修改数据库。通常如果数据库被外部应用程序修改，需要调用此接口保证数据一致
使用示例	Mainwindow.h

	imf::RobotController pTcpApi; Mainwindow.cpp pTcpApi.loadProjectVar(imf::rcf::RobotVar::Point);

3.5.14 createRobotProject

指令原型	**bool createRobotProject**(std::string project_name);
指令说明	如果工程已经存在，那么改接口直接返回 true 否则创建一个默认工程，包括: 1.空程序 main 2.1000 个位置和增量位置 3.300 个变量，字符
指令参数	project_name:需要创建的工程名
指令返回值	执行是否成功
注意事项	无
使用示例	Mainwindow.h imf::RobotController pTcpApi; Mainwindow.cpp std::string project = "工程"; pTcpApi.cmdCreatRobotProject(project);

3.5.15 getRemoteWorkMode

指令原型	**bool getRemoteWorkMode**(rcf::WorkMode &mode);
指令说明	获取机器人远程模式（远程-示教；远程-回放）
指令参数	mode:工作模式
指令返回值	执行是否成功
注意事项	仅在远程模式下调用
使用示例	Mainwindow.h imf::RobotController pTcpApi; Mainwindow.cpp rcf::WorkMode mode; pTcpApi.getremoteworkmode(mode);

3.6 系统参数读写接口

系统参数包括：关节参数，笛卡尔参数，外部轴参数，系统硬件参数，编码器零点参数，规划器参数，模型参数，坐标系具体配置，回零关节位置，工作空间参数，系统其他参数，耦合比参数，IO 配置参数，背景任务配置参数。

本节提供对这些参数读写的接口，具体接口如下：

3.6.1 getRobotParam

指令原型	**bool getRobotParam**(rcf::TRobotCartesianParam &param, int id = 1);
指令说明	按照 id 获取机器人配置参数
指令参数	param:需要获取的机器人参数
	id:对应的 id,从 1 开始
指令返回值	执行是否成功
注意事项	无
使用示例	Mainwindow.h imf::RobotController pTcpApi; Mainwindow.cpp imf::rcf::TRobotCartesianParam Cartesian; pTcpApi.getRobotParam(Cartesian);

3.6.2 setRobotParam

指令原型	**bool setRobotParam**(const rcf::TRobotCartesianParam &param);
指令说明	设置机器人配置参数
指令参数	param:需要设置的机器人参数，同步到 Motion 生效
指令返回值	执行是否成功
注意事项	设置机器人参数可能导致机器人工作异常，请务必确认
使用示例	Mainwindow.h imf::RobotController pTcpApi; Mainwindow.cpp imf::rcf::TRobotCartesianParam Cartesian; Cartesian.positive_limit[6] = 10; Cartesian.negative_limit[6] = 10;

	Cartesian.max_jog_vel[6] = 55; Cartesian.max_jog_acc[6] = 80; Cartesian.max_jog_dec[6] = 40; Cartesian.max_linear_vel= 1000; Cartesian.max_linear_acc=4000; Cartesian.max_linear_dec=4000; Cartesian.max_angular_vel=300; Cartesian.max_angular_acc=800; Cartesian.max_angular_dec=800; pTcpApi.setRobotParam(Cartesian);

3.7 IO 控制接口

本节提供对对 IO 模块的读写，具体接口如下：

3.7.1 getDiBit

指令原型	**bool getDiBit**(uint16_t &value, uint16_t group, uint16_t idx);
指令说明	按位获取输入数字量
指令参数	value:数值
	group:模块号
	idx:IO 编号
指令返回值	执行是否成功
注意事项	无
使用示例	Mainwindow.h imf::RobotController pTcpApi; Mainwindow.cpp uint16_t value; pTcpApi.getDiBit(value,1,1);

3.7.2 getDoBit

指令原型	**bool getDoBit**(uint16_t &value, uint16_t group, uint16_t idx);
指令说明	按位获取输出数字量
指令参数	value:数值

	group:模块号
	idx:IO 编号
指令返回值	执行是否成功
注意事项	无
使用示例	Mainwindow.h imf::RobotController pTcpApi; Mainwindow.cpp uint16_t value; pTcpApi.getDoBit(value,1,1);

3.7.3 getDiMask

指令原型	**bool getDiMask**(uint16_t &value, uint16_t group,uint16_t mask);
指令说明	按位获取输入数字量
指令参数	value:数值
	group:模块号
	mask:位掩码
指令返回值	执行是否成功
注意事项	无
使用示例	Mainwindow.h imf::RobotController pTcpApi; Mainwindow.cpp uint16_t value; pTcpApi.getDiMask(value,1,1);

3.7.4 getDoMask

指令原型	**bool getDoMask**(uint16_t &value, uint16_t group,uint16_t mask);
指令说明	按位获取输入数字量
指令参数	value:数值
	group:模块号
	mask:位掩码

指令返回值	执行是否成功
注意事项	无
使用示例	Mainwindow.h imf::RobotController pTcpApi; Mainwindow.cpp uint16_t value; pTcpApi.getDoMask(value,1,1);

3.7.5 getAo

指令原型	**bool getAo**(double &value, uint16_t idx);
指令说明	按位获取输入数字量
指令参数	value:数值
	idx:IO 编号
指令返回值	执行是否成功
注意事项	无
使用示例	Mainwindow.h imf::RobotController pTcpApi; Mainwindow.cpp double value; pTcpApi.getAo(value,1);

3.7.6 getAi

指令原型	**bool getAi**(double &value, uint16_t idx);
指令说明	按位获取输入数字量
指令参数	value:数值
	idx:IO 编号
指令返回值	执行是否成功
注意事项	无
使用示例	Mainwindow.h imf::RobotController pTcpApi; Mainwindow.cpp double value;

	pTcpApi.getAi(value,1);

3.7.7 setAo

指令原型	**bool setAo**(uint16_t idx,double value);
指令说明	按位获取输入数字量
指令参数	idx:IO 编号
	value:数值
指令返回值	执行是否成功
注意事项	无
使用示例	Mainwindow.h imf::RobotController pTcpApi; Mainwindow.cpp pTcpApi.setAo(10,1);

3.7.8 setDoBit

指令原型	**bool setDoBit**(uint16_t group, uint16_t index, uint16_t value);
指令说明	按位获取输入数字量
指令参数	group：模块号
	index:IO 编号
	value:数值
指令返回值	执行是否成功
注意事项	无
使用示例	Mainwindow.h imf::RobotController pTcpApi; Mainwindow.cpp pTcpApi.setDoBit(1,7,1);

3.7.9 setDoMask

指令原型	**bool setDoMask**(uint16_t group, uint16_t mask, uint16_t value);
指令说明	按位获取输入数字量
指令参数	group:模块号

	mask:操作的位掩码
	value:数值
指令返回值	执行是否成功
注意事项	无
使用示例	Mainwindow.h imf::RobotController pTcpApi; Mainwindow.cpp pTcpApi.setDoBit(1,10,1);

3.8 工艺信息接口

本节提供对机器人工艺参数，Modbus 数据和数据库的读取，具体接口如下

3.8.1 getAppData

指令原型	**bool getAppData**(std::vector<uint16_t> &data, imf::rcf::DataType type, int offset_in_words, int app_id);
指令说明	以十六进制形式获取工艺信息
指令参数	data:十六进制形式的工艺信息
	type:寄存器类型
	offset_in_words:工艺数据所占的尺寸
	app_id:对应工艺 ID
指令返回值	执行是否成功
注意事项	无

使用示例	Mainwindow.h
	imf::RobotController pTcpApi;
	Mainwindow.cpp
	imf::rcf::DataType type=imf::rcf::DataType::ReadWrite;
	int offset_in_words=1;
	int app_id=21;
	std::vector<uint16_t> Vec(50);
	pTcpApi.getAppData(Vec,type,offset_in_words,app_id);

3.8.2 setAppData

指令原型	**bool setAppData**(const std::vector<uint16_t> &data, int offset_in_words, int app_id);
指令说明	以十六进制形式设置工艺信息
指令参数	data:十六进制形式的工艺信息
	type:寄存器类型
	offset_in_words:工艺数据所占的尺寸
	app_id:对应工艺 ID
指令返回值	执行是否成功
注意事项	无

使用示例	Mainwindow.h
	imf::RobotController pTcpApi;
	Mainwindow.cpp
	imf::rcf::DataType type=imf::rcf::DataType::ReadWrite;
	int offset_in_words=1;
	int app_id=21;
	std::vector<uint16_t> Vec(50);
	pTcpApi.setAppData(Vec,type,offset_in_words,app_id);

3.8.3 getAppTData

指令原型	**bool getAppTData**(TData &data, imf::rcf::DataType type, int offset_in_words, short app_id)
指令说明	获取工艺信息
指令参数	data:需要获取的工艺信息
	type:需要获取的数据类型
	offset_in_words:工艺数据所占的尺寸
	app_id:对应工艺 ID
指令返回值	执行是否成功
注意事项	无
使用示例	Mainwindow.h imf::RobotController pTcpApi; Mainwindow.cpp imf::rcf::DataType type=imf::rcf::DataType::ReadWrite;

	int offset_in_words=1;
	int app_id=21;
	TData data(50);
	pTcpApi.getAppTData(data,type,offset_in_words,app_id);

3.8.4 setAppTData

指令原型	**bool setAppTData**(const TData &data, int offset_in_words, short app_id)
指令说明	设置工艺信息
指令参数	data：需要写入工艺的信息
	offset_in_words：工艺数据所占的尺寸
	app_id：对应工艺 ID
指令返回值	执行是否成功
注意事项	无

使用示例	Mainwindow.h
	imf::RobotController pTcpApi;
	Mainwindow.cpp
	imf::rcf::DataType type=imf::rcf::DataType::ReadWrite;
	int offset_in_words=1;
	int app_id=21;
	TData data(50);
	pTcpApi.setAppTData(data,type,offset_in_words,app_id);

3.8.5 getModbusData

指令原型	**bool getModbusData**(std::vector < uint16_t > &data, imf::rcf::DataType type, int offset_in_words);
指令说明	以十六进制形式获取控制器 Modbus 信息
指令参数	data:需要获取的 Modbus 信息
	type:需要获取的寄存器类型
	offset_in_words:偏移地址，具体 ID 偏置咨询技术人员
指令返回值	执行是否成功
注意事项	无
使用示例	Mainwindow.h

	imf::RobotController pTcpApi;
	Mainwindow.cpp
	imf::rcf::DataType::ReadWrite;
	int offset_in_words=1;
	std::vector<uint16_t> Vec(50);
	ModbusData data(50);
	pTcpApi.getModbusData(Vec,type,offset_in_words,app_id);

3.8.6 setModbusData

指令原型	**bool setModbusData**(std::vector < uint16_t > &data, imf::rcf::DataType type, int offset_in_words);
指令说明	以十六进制形式设置控制器 Modbus 信息
指令参数	data:需要设置的 Modbus 信息
	type:需要设置的寄存器类型
	offset_in_words:偏移地址，具体 ID 偏置咨询技术人员
指令返回值	执行是否成功
注意事项	远程模式下，系统 modbus 很多功能不生效！！！不建议使用！！！

使用示例	Mainwindow.h
	imf::RobotController pTcpApi;
	Mainwindow.cpp
	imf::rcf::DataType::ReadWrite;
	int offset_in_words=1;
	std::vector<uint16_t> Vec(50);
	ModbusData data(50);
	pTcpApi.setModbusData(Vec,type,offset_in_words,app_id);

3.8.7 getMysqlData

指令原型	**bool getMysqlData**(std::string &mysqlData, std::string table, std::string name, std::string version, std::string name_id);
指令说明	获取数据库信息
指令参数	mysqlData:数据库信息，格式为字符串、Json
	table:数据库的表头名字
	name:数据库的名字
	version:数据库的版本

	name_id:数据库 name 对应序号
指令返回值	执行是否成功
注意事项	无
使用示例	Mainwindow.h imf::RobotController pTcpApi; Mainwindow.cpp imf::rcf::DataType::ReadWrite; std::string table = "MYSQL"; std::string name = "name"; std::string version= "v1.0"; std::string name_id=1; std::string mysqlData; pTcpApi.getMysqlData(mysqlData, table, name, version,name_id);

3.8.8 setMysqlData

指令原型	**bool setMysqlData**(std::string &mysqlData, std::string table, std::string name, std::string version, std::string name_id);
指令说明	设置数据库信息
指令参数	mysqlData:设置数据库信息，格式为字符串、Json
	table:设置数据库的表头名字
	name:设置数据库的名字
	version:设置数据库的版本
	name_id:设置数据库 name 对应序号
指令返回值	执行是否成功
注意事项	无
使用示例	Mainwindow.h imf::RobotController pTcpApi; Mainwindow.cpp imf::rcf::DataType::ReadWrite; std::string table = "MYSQL"; std::string name = "name"; std::string version= "v1.0"; std::string name_id=1;

	std::string mysqlData; pTcpApi.setMysqlData(mysqlData, table, name, version,name_id);