|  |
| --- |
| 16 |
| VS2010学习及理实一体化开发笔记 |
| V2.0学习手册经验 |
|  |
| V 2.0  2017-5 |

目录

[1 二次开发简介 8](#_Toc481583081)

[1.1 二次开发流程 8](#_Toc481583082)

[1.2 术语 8](#_Toc481583083)

[2 软件包配置及使用 8](#_Toc481583084)

[2.1 软件包的介绍 8](#_Toc481583085)

[2.2 Windows下软件包配置及使用 9](#_Toc481583086)

[2.2.1 C++二次开发库配置 9](#_Toc481583087)

[3 常见错误处理办法及技巧 13](#_Toc481583088)

[3.1 返回数据类型 13](#_Toc481583089)

[3.2 网络相关接口 13](#_Toc481583090)

[3.2.1 网络初始化 13](#_Toc481583091)

[3.2.2 扫描控制器设备 13](#_Toc481583092)

[3.2.3 网络连接 14](#_Toc481583093)

[3.2.4 检查网络状态 14](#_Toc481583094)

[3.2.5 网络断开 15](#_Toc481583095)

[3.3 系统 15](#_Toc481583096)

[3.3.1 控制器重启 15](#_Toc481583097)

[3.3.2 控制器启动状态 15](#_Toc481583098)

[3.4 获取信息 16](#_Toc481583099)

[3.4.1 获取状态数据 16](#_Toc481583100)

[3.4.2 获取坐标数据 16](#_Toc481583101)

[3.5 运行模式 17](#_Toc481583102)

[3.5.1 设置使能状态 17](#_Toc481583103)

[3.5.2 获取使能状态 17](#_Toc481583104)

[3.5.3 设置工作模式 18](#_Toc481583105)

[3.5.4 获取工作模式 18](#_Toc481583106)

[3.5.5 设置坐标系类型 19](#_Toc481583107)

[3.5.6 获取坐标系类型 19](#_Toc481583108)

[3.5.7 设置组 20](#_Toc481583109)

[3.5.8 设置当前窗口 20](#_Toc481583110)

[3.5.9 获取当前窗口 21](#_Toc481583111)

[3.5.10 设置当前倍率 21](#_Toc481583112)

[3.5.11 获取当前倍率 21](#_Toc481583113)

[3.6 寄存器 22](#_Toc481583114)

[3.6.1 设置IR寄存器 22](#_Toc481583115)

[3.6.2 获取IR寄存器 22](#_Toc481583116)

[3.6.3 设置DR寄存器 23](#_Toc481583117)

[3.6.4 获取DR寄存器 23](#_Toc481583118)

[3.6.5 设置JR寄存器 23](#_Toc481583119)

[3.6.6 获取JR寄存器 24](#_Toc481583120)

[3.6.7 设置LR寄存器 25](#_Toc481583121)

[3.6.8 获取LR寄存器 25](#_Toc481583122)

[3.6.9 设置工具坐标 26](#_Toc481583123)

[3.6.10 获取工具坐标 26](#_Toc481583124)

[3.6.11 设置工件坐标 26](#_Toc481583125)

[3.6.12 获取工件坐标 27](#_Toc481583126)

[3.6.13 设置工具坐标号 27](#_Toc481583127)

[3.6.14 获取工具坐标号 28](#_Toc481583128)

[3.6.15 设置工件坐标号 28](#_Toc481583129)

[3.6.16 获取工件坐标号 28](#_Toc481583130)

[3.7 获取轴数 29](#_Toc481583131)

[3.7.1 获取机器人内部轴轴数 29](#_Toc481583132)

[3.7.2 获取机器人外部轴轴数 29](#_Toc481583133)

[3.8 坐标 30](#_Toc481583134)

[3.8.1 获取关节坐标 30](#_Toc481583135)

[3.8.2 获取笛卡尔坐标 30](#_Toc481583136)

[3.8.3 获取外部轴坐标 31](#_Toc481583137)

[3.9 文件 31](#_Toc481583138)

[3.9.1 发送文件给控制器 31](#_Toc481583139)

[3.9.2 从控制器获取文件 32](#_Toc481583140)

[3.10 手动模式运行 32](#_Toc481583141)

[3.10.1 看门狗设置 32](#_Toc481583142)

[3.10.2 开始运动关节 32](#_Toc481583143)

[3.10.3 停止运动关节 33](#_Toc481583144)

[3.11 自动模式运行 33](#_Toc481583145)

[3.11.1 发送主程序 33](#_Toc481583146)

[3.11.2 获取主程序 34](#_Toc481583147)

[3.11.3 加载主程序 34](#_Toc481583148)

[3.11.4 清空子程序数组 35](#_Toc481583149)

[3.11.5 发送子程序 35](#_Toc481583150)

[3.11.6 加载子程序 35](#_Toc481583151)

[3.11.7 开始程序 36](#_Toc481583152)

[3.11.8 停止程序 36](#_Toc481583153)

[3.11.9 暂停程序 36](#_Toc481583154)

[3.11.10 获取程序状态 37](#_Toc481583155)

[3.12 单步 37](#_Toc481583156)

[3.12.1 单步进入 37](#_Toc481583157)

[3.12.2 单步跳出 38](#_Toc481583158)

[3.12.3 单步执行 38](#_Toc481583159)

[3.12.4 运行到任意行 38](#_Toc481583160)

[3.13 标定 39](#_Toc481583161)

[3.13.1 工具坐标标定 39](#_Toc481583162)

[3.13.2 基坐标标定 41](#_Toc481583163)

[3.14 运动设置 42](#_Toc481583164)

[3.14.1 运动到点 42](#_Toc481583165)

[3.14.2 设置零点 43](#_Toc481583166)

[3.14.3 保存零点 43](#_Toc481583167)

[3.14.4 设置轴限位信息 44](#_Toc481583168)

[3.14.5 获取轴限位信息 44](#_Toc481583169)

[3.14.6 保存内部轴软限位 45](#_Toc481583170)

[3.14.7 保存外部轴软限位 45](#_Toc481583171)

[3.15 IO设置 46](#_Toc481583172)

[3.15.1 数字输入输出端 46](#_Toc481583173)

[3.15.2 外部运行配置 53](#_Toc481583174)

[3.15.3 模拟输入输出端 55](#_Toc481583175)

[3.15.4 获取IO信息 61](#_Toc481583176)

[3.16 用户PLC设置 61](#_Toc481583177)

[3.16.1 设置用户PLC 61](#_Toc481583178)

[3.16.2 获取用户PLC使能状态 62](#_Toc481583179)

[3.16.3 判断用户PLC运行状态 62](#_Toc481583180)

[3.16.4 启动PLC 62](#_Toc481583181)

[3.16.5 停止PLC 63](#_Toc481583182)

[3.17 其他指令 63](#_Toc481583183)

[3.17.1 获取系统信息 63](#_Toc481583184)

[3.17.2 清除错误 64](#_Toc481583185)

[3.17.3 设置语言 65](#_Toc481583186)

[3.17.4 开启打印 65](#_Toc481583187)

[3.17.5 获取SO（库）版本 65](#_Toc481583188)

[3.17.6 获取HMCApi版本 66](#_Toc481583189)

[3.18 通用接口 66](#_Toc481583190)

[4 开发者实例 68](#_Toc481583191)

[4.1 寄存器操作 68](#_Toc481583192)

[4.1.1 操作接口 68](#_Toc481583193)

[4.1.2 例子 68](#_Toc481583194)

[4.2 程序运行 69](#_Toc481583195)

[4.2.1 程序运行状态图 69](#_Toc481583196)

[4.2.2 例子 71](#_Toc481583197)

[4.3 单步运行 73](#_Toc481583198)

[4.4 标定工具/基坐标系 73](#_Toc481583199)

[4.4.1 标定工具坐标 73](#_Toc481583200)

[4.4.2 标定基坐标 73](#_Toc481583201)

[4.5 手动运行 74](#_Toc481583202)

[附录 75](#_Toc481583203)

[附录1 寄存器索引值范围 75](#_Toc481583204)

[附录2 数据类型定义（DataDef.h） 75](#_Toc481583205)

# 二次开发简介

通过二次开发，用户可以仅关注需要的东西，通过接口定制软件功能，而不用了解底层功能到底如何实现，能够加快开发进度，提高开发效率。

## 二次开发流程

华数二型数控系统二次开发软件提供接口，能够采集并设置下位机数据，从而实现对下位机的控制。用户只需要自己开发自己关注的功能，然后通过调用封装好的二次开发接口，来调用底层数据，调用成功后将值返回给用户。用户可以完全脱离示教器来控制和监控机器人，实现总控等相关功能。

## 术语

IPC —— 机器人控制器。

HscApi —— 二次开发库的应用层接口。

HMCApi —— 二次开发库的中间层接口。

# 软件包配置及使用

目前针对的主流开发平台Linux和Windows，提供了不同的二次开发软件包。

## 软件包的介绍

对于Windows开发，提供C++开发库，具体包含如下几个文件。具体如下图所示：

表 1软件包列表

|  |  |  |
| --- | --- | --- |
| **开发环境** | | **SDK包文件** |
| Windows | C++ | 动态库、静态库文件（TCPComm.dll、HMCAPI.dll、HSCAPI.dll 、HMCAPI.lib 、HSCAPI.lib） |
| include头文件文件夹 |

## 理实一体化软件模块连接华数IPC

### 使用理实一体化连接华数IPC对应的局域网时3两种情况：

1. 使用电脑直接连接IPC将其网线插入IPC的HPC-102端口，与电脑进行直接连接。网络接线连接完毕后，需要将电脑的IP地址设置成和IPC同一网络段位，比如：90.0.0.XXX;
2. 使用电脑直接连接交换机，此交换机与IPC通过网线直连，将网线插入已经连接IPC的交换机，交换机同时连接IPC的HPC-102端口，网络接线连接完毕后，需要将电脑的IP地址设置成和IPC同一网络段位，比如：90.0.0.XXX;
3. 使用电脑直接连接路由器，将网线插入已经连接IPC的路由器的LAN口或者采用WIFI无线连接路由器，同时将电脑的的无线地址设置为90.0.0.XXX;，路由器与IPC的HPC-102端口直连。网络接线连接完毕后，需要将电脑的IP地址设置成和IPC同一网络段位，比如：90.0.0.XXX;

### 常见故障排除

### 无法正常连接IPC，需要检查IPC的网路是否畅通，在Cmd中输入对应的ip进行网络畅通性检测。Ping 90.0.0.1

* 检查电脑的IP设置是否和IPC的网络处于同一段位。90.0.0.xxx。
* 若无法Ping通，则需要进行网络连接的检查，检查网络连接的IPC端口是否选择正确。
* 路由器和交换机是否正常工作，若为正常工作，则需要重启路由器或交换机。必要时需要恢复路由器的出场设置。
* 关闭电脑上的其他VPN连接。
* 使用排除法进行故障检查

### 常见故障排除

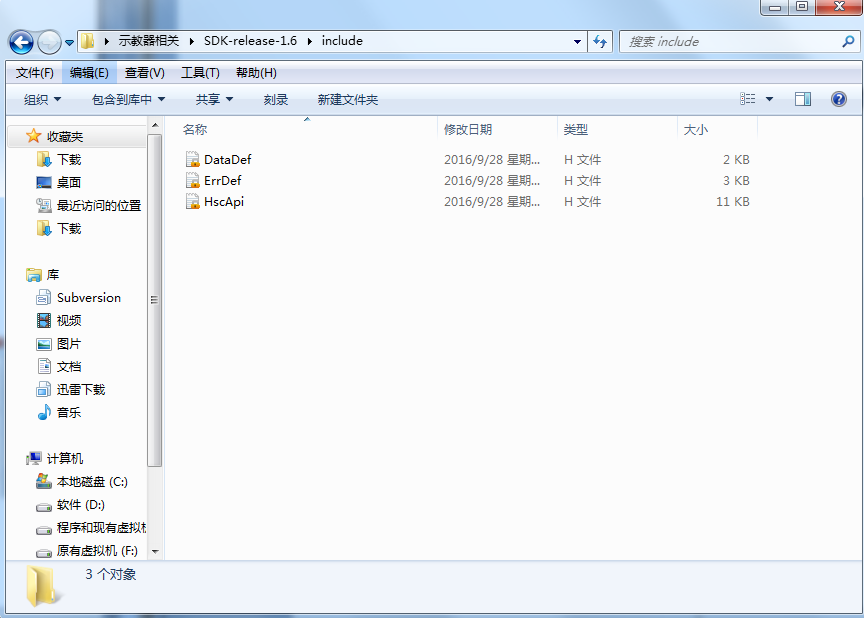
## Windows下软件包配置及使用

在Windows平台下进行基于华数二型的二次开发，需将HMCAPI相关的软件包文件加入自己的工程文件夹中。下面就分别介绍Microsoft Visual Studio 10.0中的使用环境下C++开发库。注意，此库只能在release版本下才能成功运行。

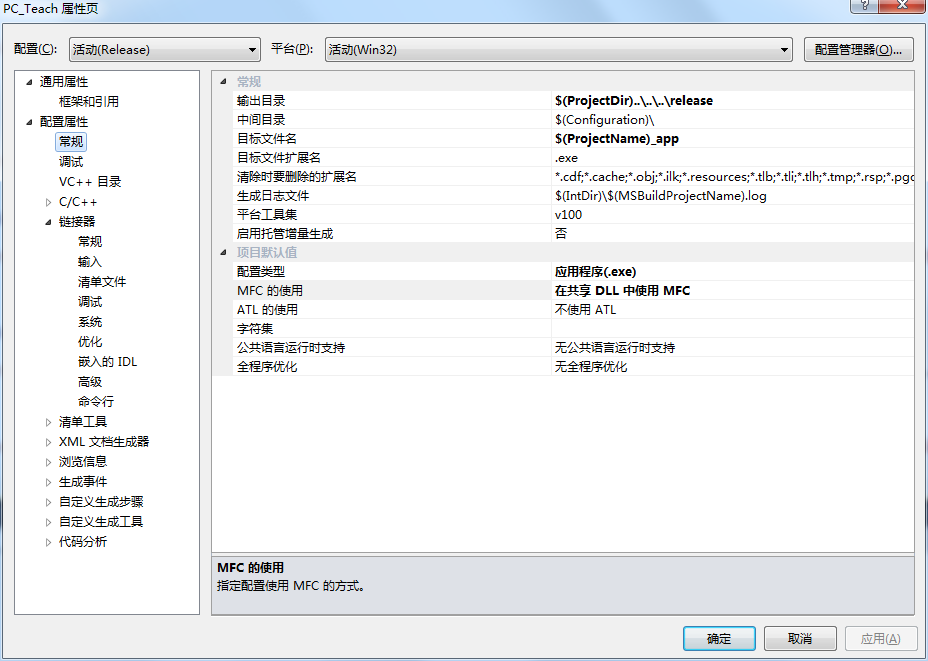
### C++二次开发库配置

Microsoft Visual Studio 10.0配置开发环境步骤：

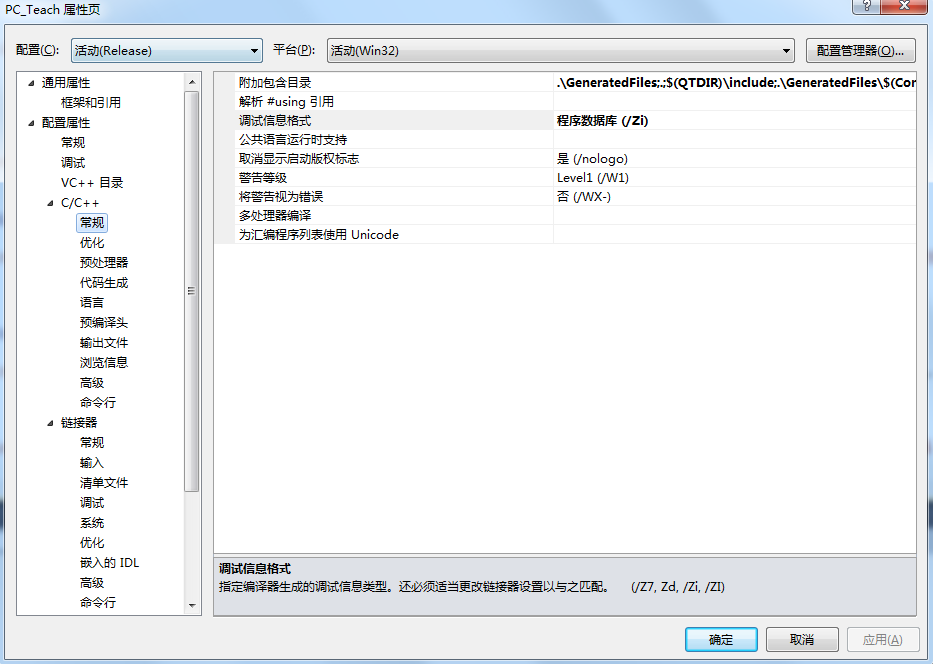
1. 启动Microsoft Visual Studio 10.0，新建一个应用程序工程demo\_C++；
2. 在新建工程中添加头文件：



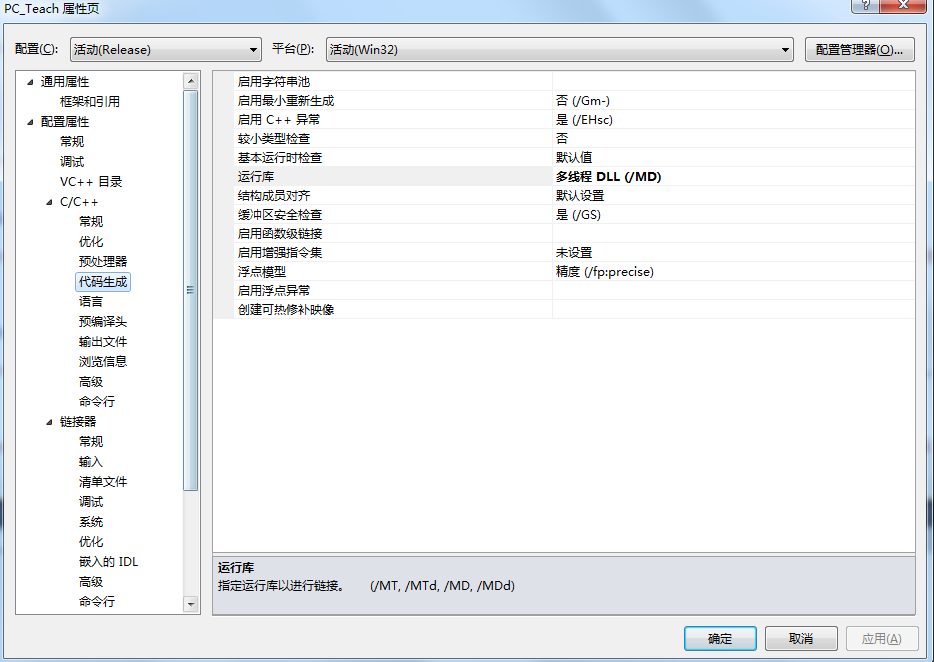
1. 配置常规属性：



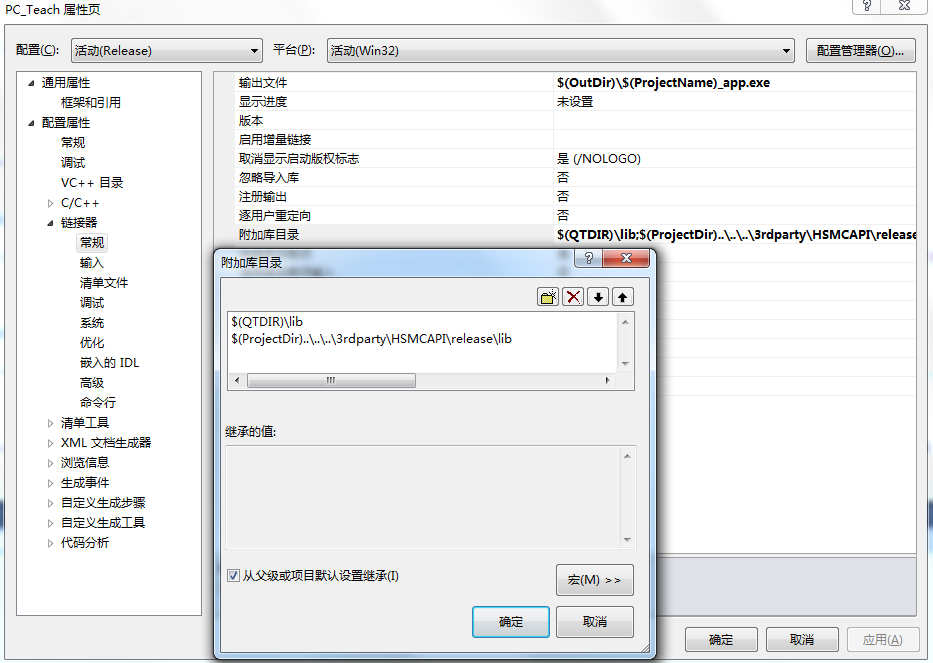
1. 配置C/C++（常规）：



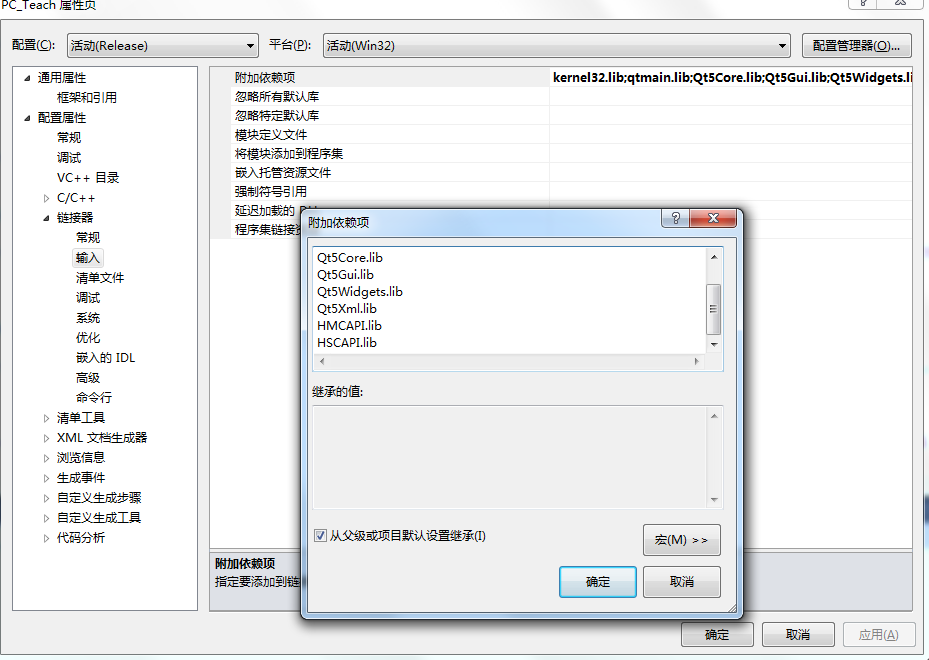
1. 配置C/C++（代码生成）：



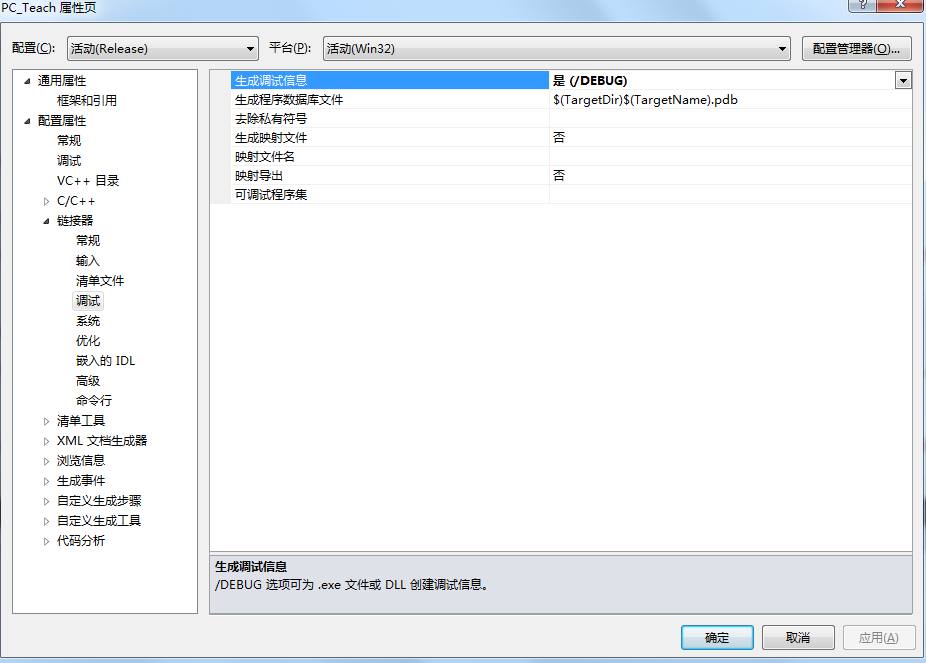
1. 配置链接器（常规）：



1. 配置链接器（输入）



1. 设置调试信息：



# 常见错误处理办法及技巧

## C++常见误

C++常见错误分为两类编译错误和调试错误

## 编译错误分类

### 编译警告错误

1. 

此原因为采用了int类型的数据与size\_t类型的数据进行对比，导致警告问题出现。因此需要把int类型的数据修改为unsigned int的类型数据。或者将变量定义为size\_t类型的数据。

### 扫描控制器设备

HMCErrCode NetSacnDevice(std::vector<HscDevice>& deviceVector);

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode NetSacnDevice(std::vector<HscDevice>& deviceVector); | |
| 参数 | deviceVector | 获取到的设备信息，包括各IPC的名字、IP地址、SN序列号 |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret1 = 0, ret2 = 0;  std::vector<HscDevice> deviceVector;  ret1 = hscapi->NetInit(); // 注意必须先初始化  ret2 = hscapi->NetSacnDevice (deviceVector); | |

### 网络连接

HMCErrCode NetConnect(const std::string& rIp, const unsigned short rPort);

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode NetConnect(const std::string& rIp, const unsigned short rPort); | |
| 参数 | rIp | IPC的IP地址 |
| rPort | 端口号（可用的是**5001、5003、5004、5005**这4个端口） |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  ret = hscapi->NetConnect("90.0.0.1", 5005); | |

### 检查网络状态

HMCErrCode NetIsConnect();

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode NetIsConnect (); | |
| 参数 | 无 | |
| 返回值 | HMCErrCode | 0：网络连通；其他：网络出现问题 |
| 示例 | HMCErrCode ret = 0;  ret = hscapi->NetIsConnect (); | |

### 网络断开

HMCErrCode NetExit();

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode NetExit (); | |
| 参数 | 无 | |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  ret = hscapi->NetExit (); | |
| 备注 | * 网络断开执行的时间较长 | |

## 系统

### 控制器重启

HMCErrCode SysReboot ();

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode SysReboot (); | |
| 参数 | 无 | |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  ret = hscapi->SysReboot (); | |

### 控制器启动状态

HMCErrCode SysIsBootComplete (int& bootState);

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode SysIsBootComplete (int& bootState); | |
| 参数 | bootState | 控制器启动状态（0：启动未完成；1：启动完成且正常；其它：其它情况） |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  int state;  ret = hscapi->SysIsBootComplete (state); | |
| 备注 | 建议使用“bootState != 0”来判断启动是否完成。 | |

## 获取信息

### 获取状态数据

HMCErrCode getStateData(StateData &state);

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode getStateData(StateData &state); | |
| 参数 | state | 状态数据，包括模式、运行程序名字、倍率等 |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  StateData state;  ret = hscapi->getStateData(state); | |

### 获取坐标数据

HMCErrCode getPositionData(PosData &datas);

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode getPositionData(PosData &datas); | |
| 参数 | datas | 当前坐标数据，包括关节坐标、笛卡尔坐标、外部轴坐标 |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  PosData datas;  ret = hscapi->getStateData(datas); | |

## 运行模式

### 设置使能状态

HMCErrCode SetEnableState(enable en) ;

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode SetEnableState(enable en); | |
| 参数 | en | 使能状态   * DISABLE 禁止 * ENABLE 使能 |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  ret = hscapi->SetEnableState(ENABLE); //打开使能 | |

### 获取使能状态

HMCErrCode GetEnableState(enable& en) ;

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode GetEnableState(enable& en); | |
| 参数 | en | 使能状态   * DISABLE 禁止 * ENABLE 使能 |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  enable en;  ret = hscapi->GetEnableState(en); //获取使能状态 | |

### 设置工作模式

HMCErrCode SetWorkMode(const WorkMode mode) ;

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode SetWorkMode(const WorkMode mode); | |
| 参数 | mode | 工作模式   * WORK\_MODE\_AUT 自动 * WORK\_MODE\_T1 手动（T1） * WORK\_MODE\_T2 手动（T2） * WORK\_MODE\_EXT 外部 |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  ret = hscapi->SetWorkMode(WORK\_MODE\_T1); //设置为T1手动运行模式 | |

### 获取工作模式

HMCErrCode GetWorkMode(WorkMode& mode) ;

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode GetWorkMode(WorkMode& mode); | |
| 参数 | mode | 工作模式   * WORK\_MODE\_AUT 自动 * WORK\_MODE\_T1 手动（T1） * WORK\_MODE\_T2 手动（T2） * WORK\_MODE\_EXT 外部 |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  WorkMode workmode;  ret = hscapi->GetWorkMode(workmode); //获取运行模式 | |

### 设置坐标系类型

HMCErrCode setFrame(const FrameType& frame);

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode setFrame(const FrameType& frame); | |
| 参数 | frame | 坐标系类型   * COORD\_TYPE\_JOINT 关节坐标系 * COORD\_TYPE\_WORLD 世界坐标系 * COORD\_TYPE\_BASE 基坐标系 * COORD\_TYPE\_TOOL 工具坐标系 |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  ret = hscapi->setFrame(COORD\_TYPE\_JOINT); //设置为轴坐标系 | |

### 获取坐标系类型

HMCErrCode GetFrameType(FrameType& frame);

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode GetFrameType(FrameType& frame); | |
| 参数 | frame | 坐标系类型   * COORD\_TYPE\_JOINT 关节坐标系 * COORD\_TYPE\_WORLD 世界坐标系 * COORD\_TYPE\_BASE 基坐标系 * COORD\_TYPE\_TOOL 工具坐标系 |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  FrameType frametype;  ret = hscapi->GetFrameType(frametype); //获取坐标系类型 | |

### 设置组

HMCErrCode setMotionGroup(const GROUP\_TYPE &group);

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode setMotionGroup(const GROUP\_TYPE &group); | |
| 参数 | group | 组名   * GROUP\_TYPE\_ROBOT 内部轴 * GROUP\_TYPE\_EXT\_AXES 外部轴 |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  ret = hscapi->setMotionGroup(GROUP\_TYPE\_ROBOT); | |

### 设置当前窗口

HMCErrCode setScreen(const TpScreen & screen);

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode setScreen(const TpScreen & screen); | |
| 参数 | screen | 窗口类型   * SCREEN\_JOG 点动窗口 * SCREEN\_PRG 程序窗口 |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  ret = hscapi->setScreen(SCREEN\_PRG); //设置为程序窗口 | |

### 获取当前窗口

HMCErrCode GetTpScreen(TpScreen& screen)；

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode GetTpScreen(TpScreen& screen); | |
| 参数 | screen | 窗口类型 |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  TpScreen tpscreen;  ret = hscapi->GetTpScreen(tpscreen); //获取窗口类型 | |

### 设置当前倍率

HMCErrCode SetOverride(const unsigned int nOverride)；

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode SetOverride(const unsigned int nOverride); | |
| 参数 | nOverride | 倍率，范围：[1, 100] |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  ret = hscapi->SetOverride(20); | |

### 获取当前倍率

HMCErrCode GetOverride(int& nOverride);

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode GetOverride(int& nOverride); | |
| 参数 | nOverride | 倍率 |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  int override = 0;  ret = hscapi->GetOverride(override); | |

## 寄存器

### 设置IR寄存器

HMCErrCode SetIR(int index, long value);

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode SetIR(int index, long value); | |
| 参数 | index | 索引 |
| value | 整数 |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  ret = hscapi->SetIR(1, 5); | |
| 备注 | 寄存器相关接口的索引值范围请参考附录1。 | |

### 获取IR寄存器

HMCErrCode GetIR(int index, long& value);

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode GetIR(int index, long& value); | |
| 参数 | index | 索引 |
| value | 整数 |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  long value = 0;  ret = hscapi->GetIR(1, value); | |

### 设置DR寄存器

HMCErrCode SetDR(int index, double value);

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode SetDR(int index, double value); | |
| 参数 | index | 索引 |
| value | 浮点数 |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  ret = hscapi->SetDR(1, 5.0); | |

### 获取DR寄存器

HMCErrCode GetDR(int index, double& value);

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode GetDR(int index, double& value); | |
| 参数 | index | 索引 |
| value | 浮点数 |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  double value = 0.0;  ret = hscapi->GetDR(1, value); | |

### 设置JR寄存器

HMCErrCode SetJR(int index, const JointPos& value);

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode SetJR(int index, const JointPos& value); | |
| 参数 | index | 索引 |
| value | 关节坐标 |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  JointPos value;  value.push\_back(1.0);  value.push\_back(2.0);  value.push\_back(3.0);  value.push\_back(4.0); //以4轴为例，坐标为{1, 2, 3, 4}  ret = hscapi->SetJR(1, value); | |

### 获取JR寄存器

HMCErrCode GetJR(int index, JointPos& value);

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode GetJR(int index, JointPos& value); | |
| 参数 | index | 索引 |
| value | 关节坐标 |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  JointPos value;  ret = hscapi->GetJR(1, value); //以4轴为例，若成功，则1、2、3、4轴数据在value[0]、value[1]、value[2]、value[3]中 | |

### 设置LR寄存器

HMCErrCode SetLR(int index, const DcartPos& value);

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode SetLR(int index, const DcartPos& value); | |
| 参数 | index | 索引 |
| value | 笛卡尔坐标 |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  DcartPos value;  value.push\_back(1.0);  value.push\_back(2.0);  value.push\_back(3.0);  value.push\_back(4.0); //以4维坐标为例，坐标为#{1, 2, 3, 4}  ret = hscapi->SetLR(1, value); | |

### 获取LR寄存器

HMCErrCode GetLR(int index, DcartPos& value);

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode GetLR(int index, DcartPos& value); | |
| 参数 | index | 索引 |
| value | 笛卡尔坐标 |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  DcartPos value;  ret = hscapi->GetLR(1, value); //以4维坐标为例，若成功，则1、2、3、4维数据在value[0]、value[1]、value[2]、value[3]中 | |

### 设置工具坐标

HMCErrCode setToolFrame(const int index, const std::string & data);

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode setToolFrame(const int index, const std::string & data); | |
| 参数 | index | 索引 |
| data | 笛卡尔坐标（字符串） |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  ret = hscapi->setToolFrame(1, “#{10, 10, 10, 10}”); | |

### 获取工具坐标

HMCErrCode getToolFrame (const int index, DcartPos& value);

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode GetLocation getToolFrame (const int index, DcartPos& value); | |
| 参数 | index | 索引 |
| value | 笛卡尔坐标 |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  DcartPos value;  ret = hscapi->getToolFrame(1, value); | |

### 设置工件坐标

HMCErrCode setBaseFrame(const int index,const std::string & data);

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode setBaseFrame(const int index,const std::string & data); | |
| 参数 | index | 索引 |
| data | 笛卡尔坐标（字符串） |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  ret = hscapi->setBaseFrame(1, “#{10, 10, 10, 10}”); | |

### 获取工件坐标

HMCErrCode getBaseFrame(const int index, DcartPos& value);

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode getBaseFrame(const int index, DcartPos& value); | |
| 参数 | index | 索引 |
| value | 笛卡尔坐标 |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  DcartPos value;  ret = hscapi->getBaseFrame(1, value); | |

### 设置工具坐标号

HMCErrCode SetToolFrameNum(const int nToolNum);

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode SetToolFrameNum(const int nToolNum); | |
| 参数 | nToolNum | 工具坐标系号，范围：[1, 16] |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  ret = hscapi->SetToolFrameNum(1); | |

### 获取工具坐标号

HMCErrCode GetToolFrameNum(int& nToolNum);

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode GetToolFrameNum(int& nToolNum); | |
| 参数 | nToolNum | 工具坐标系号，范围：[1, 16] |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  int toolnum = 0;  ret = hscapi->GetToolFrameNum(toolnum); | |

### 设置工件坐标号

HMCErrCode SetBaseFrameNum(const int nBaseNum);

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode SetBaseFrameNum(const int nBaseNum); | |
| 参数 | nToolNum | 基坐标系号，范围：[1, 16] |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  ret = hscapi->SetBaseFrameNum(1); | |

### 获取工件坐标号

DLL\_XPORT HMCErrCode GetBaseFrameNum(int& nBaseNum);

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode GetBaseFrameNum(int& nBaseNum); | |
| 参数 | nToolNum | 基坐标系号，范围：[1, 16] |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  Int basenum = 0;  ret = hscapi->GetBaseFrameNum(basenum); | |

## 获取轴数

### 获取机器人内部轴轴数

HMCErrCode GetRobotAxesNumber(int& RobotAxesNumber);

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode GetRobotAxesNumber(int& RobotAxesNumber); | |
| 参数 | RobotAxesNumber | 内部轴轴数（4轴或6轴） |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  int robotAxesNumber;  ret = hscapi->GetRobotAxesNumber (robotAxesNumber); | |

### 获取机器人外部轴轴数

HMCErrCode GetExtAxesNumber(int& ExtAxesNumber);

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode GetExtAxesNumber(int& ExtAxesNumber); | |
| 参数 | ExtAxesNumber | 外部轴轴数（0或2轴） |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  int extAxesNumber;  ret = hscapi->GetExtAxesNumber (extAxesNumber); | |

## 坐标

### 获取关节坐标

HMCErrCode GetJointPos(JointPos& jPos);

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode GetJointPos(JointPos& jPos); | |
| 参数 | jPos | 返回的关节坐标数据 |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  JointPos jointpos;  ret = hscapi->GetJointPos(jointpos); | |

### 获取笛卡尔坐标

HMCErrCode GetDcartPos(DcartPos& dPos);

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode GetDcartPos(DcartPos& dPos); | |
| 参数 | dPos | 返回的笛卡尔坐标数据 |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  DcartPos dcartpos;  ret = hscapi->GetDcartPos(dcartpos); | |

### 获取外部轴坐标

HMCErrCode GetExtAxesPos(JointPos& jPos);

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode GetExtAxesPos(JointPos& jPos); | |
| 参数 | jPos | 返回的关节坐标数据 |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  JointPos jointpos;  ret = hscapi->GetExtAxesPos(jointpos); | |

## 文件

### 发送文件给控制器

HMCErrCode NetSendFile(const std::string& srcFilename, const std::string& dstFilename);

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode NetSendFile(const std::string& srcFilename, const std::string& dstFilename); | |
| 参数 | srcFilename | 本地文件路径（绝对路径） |
| dstFilename | IPC路径，默认为**“/FFS0/SSMC/文件名”** |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  ret = hscapi->NetSendFile(“F:/A1.PRG”, “/FFS0/SSMC/A1.PRG”); | |

### 从控制器获取文件

HMCErrCode NetGetFile(const std::string& srcFilename, const std::string& dstFilename);

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode NetGetFile(const std::string& srcFilename, const std::string& dstFilename); | |
| 参数 | srcFilename | IPC路径，默认为**“/FFS0/SSMC/文件名”** |
| dstFilename | 本地文件路径（绝对路径） |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  ret = hscapi->NetGetFile(“/FFS0/SSMC/A1.PRG”, “F:\\A1.PRG”); | |

## 手动模式运行

### 看门狗设置

HMCErrCode tpJogFeedDog();

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode tpJogFeedDog (); | |
| 参数 | 无(200ms调用一次) | |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  ret = hscapi->tpJogFeedDog(); | |
| 备注 | 在手动模式、点动窗口（SCREEN\_JOG）下，必须每**200ms**调用一次。 | |

### 开始运动关节

HMCErrCode StartMoving(const int nAxis, const Direction nDirect);

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode StartMoving(const int nAxis, const Direction nDirect); | |
| 参数 | nAxis | 轴号 （1~6：内部轴；7~8：外部轴） |
| nDirect | 方向   * DIRECTION\_POSITIVE 正向 * DIRECTION\_NEGTIVE 负向 |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  ret = hscapi->StartMoving(1, DIRECTION\_POSITIVE); //1轴开始正向运动 | |

### 停止运动关节

HMCErrCode StopMoving();

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode StopMoving (); | |
| 参数 | 无 | |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  ret = hscapi->StopMoving (); | |

## 自动模式运行

### 发送主程序

HMCErrCode SendUserProgFile(const std::string& filePath, const std::string& fileName);

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode SendUserProgFile(const std::string& filePath, const std::string& fileName); | |
| 参数 | filePath | 本地文件路径（绝对路径） |
| fileName | 文件名 |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  ret = hscapi->SendUserProgFile(“F:\\LIB”, “MAIN.PRG”); | |
| 备注 | 调用该函数时，控制器会生成预加载程序\_MAIN.PRG。预加载程序的作用是递归的加载子程序及主程序。 | |

### 获取主程序

HMCErrCode GetUserProgName(std::string& name);

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode GetUserProgName(std::string& name); | |
| 参数 | name | 加载的程序名 |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  std::string progname;  ret = hscapi->GetUserProgName(progname); | |

### 加载主程序

HMCErrCode ProgLoad(const std::string& progName);

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode ProgLoad(const std::string& progName); | |
| 参数 | progName | 加载的文件名 |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  ret = hscapi->ProgLoad(“MAIN.PRG”); | |
| 备注 | 从IPC磁盘中加载已有的程序。 | |

### 清空子程序数组

HMCErrCode resetSubFuncArray();

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode resetSubFuncArray (); | |
| 参数 | 无 | |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  ret = hscapi->resetSubFuncArray(); | |

### 发送子程序

HMCErrCode SendSubFuncFile(const std::string& srcFilePath, const std::string& dstFileName);

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode SendSubFuncFile(const std::string& srcFilePath, const std::string& dstFileName); | |
| 参数 | srcFilePath | 本地文件路径（绝对路径） |
| dstFileName | 文件名 |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  ret = hscapi->SendSubFuncFile(“F:\\LIB”, “LIB1.LIB”); | |

### 加载子程序

HMCErrCode loadSubLib(const std::string &subName);

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode loadSubLib(const std::string &subName); | |
| 参数 | subName | 子程序名字 |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  ret = hscapi->loadSubLib(“LIB1.LIB”); | |

### 开始程序

HMCErrCode ProgStart();

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode ProgStart (); | |
| 参数 | 无 | |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  ret = hscapi->ProgStart(); | |

### 停止程序

HMCErrCode ProgStop();

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode ProgStop (); | |
| 参数 | 无 | |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  ret = hscapi->ProgStop(); | |

### 暂停程序

HMCErrCode ProgPause();

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode ProgPause (); | |
| 参数 | 无 | |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  ret = hscapi->ProgPause(); | |

### 获取程序状态

HMCErrCode GetUserProgStatus(TaskStatus& status);

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode GetUserProgStatus(TaskStatus& status); | |
| 参数 | status | 获取的程序状态，包括程序状态、错误代码、当前运行行号、程序名等信息 |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  TaskStatus taskstatus;  ret = hscapi->GetUserProgStatus(taskstatus); | |
| 备注 | TaskStatus中有一个ProgState枚举类型的成员，关于ProgState各枚举值所代表的意义，可参考开发者示例中的“程序运行状态图”一节的说明。 | |

## 单步

### 单步进入

HMCErrCode StepIn();

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode StepIn (); | |
| 参数 | 无 | |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  ret = hscapi->StepIn(); | |
| 备注 | 程序已加载，但还未连续运行时可调用该函数。 | |

### 单步跳出

HMCErrCode StepOut ();

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode StepOut (); | |
| 参数 | 无 | |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  ret = hscapi->StepOut(); | |
| 备注 | 程序已加载，但还未连续运行时可调用该函数。 | |

### 单步执行

HMCErrCode StepOver ();

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode StepOver (); | |
| 参数 | 无 | |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  ret = hscapi->StepOver(); | |
| 备注 | 程序已加载，但还未连续运行时可调用该函数。 | |

### 运行到任意行

HMCErrCode runArbitraryLine(const int &no);

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode runArbitraryLine(const int &no); | |
| 参数 | no | 行号 |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  ret = hscapi->runArbitraryLine(10); | |

## 标定

### 工具坐标标定

#### 设置类型

HMCErrCode tpToolCalibrationType(const CALIBRATIONTYPE &type);

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode tpToolCalibrationType(const CALIBRATIONTYPE &type); | |
| 参数 | type | 类型   * SINGLE\_POINT 单点 * MULTI\_POINT 多点 * ORIENTATION\_POINT 方向 |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  ret = hscapi->tpToolCalibrationType(MULTI\_POINT); | |

#### 标定点

HMCErrCode tpCalibratePoint(int pointNum);

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode tpCalibratePoint(int pointNum); | |
| 参数 | pointNum | 标定索引号（≥1） |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  ret = hscapi->tpCalibratePoint(1); | |

#### 删除点

HMCErrCode tpRemoveCalibrationPoint(int pointNum);

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode tpRemoveCalibrationPoint(int pointNum); | |
| 参数 | pointNum | 标定索引号（≥1） |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  ret = hscapi->tpRemoveCalibrationPoint(1); | |

#### 工具标定

HMCErrCode tpToolCalibration(int index);

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode tpToolCalibration(int index); | |
| 参数 | index | 工具坐标寄存器索引值，范围：[1, 16] |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  ret = hscapi->tpToolCalibration(1); | |

#### 保存工具坐标标定

HMCErrCode saveTpToolCalibration ();

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode saveTpToolCalibration (); | |
| 参数 | 无 | |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  ret = hscapi->saveTpToolCalibration(); | |

### 基坐标标定

#### 设置变量

HMCErrCode setCalibrateBaseValue(int index, const std::string& value);

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode setCalibrateBaseValue(int index, const std::string& value); | |
| 参数 | index | 标定索引号，范围：[1, 3] |
| value | 笛卡尔坐标 |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  ret = hscapi->setCalibrateBaseValue(1, “#{1, 0, 0, 0}”); | |

#### 基坐标标定

HMCErrCode calibrateBase(unsigned short index);

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode calibrateBase(unsigned short index); | |
| 参数 | index | 基坐标寄存器索引值，范围：[1, 16] |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  ret = hscapi->balibrateBase(1); | |

#### 保存基坐标标定

HMCErrCode saveCalibrateBase ();

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode saveCalibrateBase (); | |
| 参数 | 无 | |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  ret = hscapi->saveCalibrateBase(); | |

## 运动设置

### 运动到点

HMCErrCode moveToPoint(const std::string &point ,const WorkGroup& groupNo, const MoveType& type);

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode moveToPoint(const std::string &point ,const WorkGroup& groupNo, const MoveType& type); | |
| 参数 | point | 目标点坐标（关节坐标或笛卡尔坐标） |
| groupNo | 组名   * ROBOT 内部轴 * EXT\_AXES 外部轴 |
| type | 运动类型   * MOVE 关节运动 * MOVES 直线运动 |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  ret = hscapi->moveToPoint(“{10, 10, 10, 10}”, ROBOT, MOVE); | |

### 设置零点

HMCErrCode setHomePositionValue(const GROUP\_TYPE &group, const std::string &data);

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode setHomePositionValue(const GROUP\_TYPE &group, const std::string &data); | |
| 参数 | group | 组名   * GROUP\_TYPE\_ROBOT 内部轴 * GROUP\_TYPE\_EXT\_AXES 外部轴 |
| data | 零点坐标（关节坐标） |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  ret = hscapi->setHomePositionValue(GROUP\_TYPE\_ROBOT, “{1,2,3,4}”); | |

### 保存零点

HMCErrCode saveHomePosition(const GROUP\_TYPE &group);

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode saveHomePosition(const GROUP\_TYPE &group); | |
| 参数 | group | 组名   * GROUP\_TYPE\_ROBOT 内部轴 * GROUP\_TYPE\_EXT\_AXES 外部轴 |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  ret = hscapi->saveHomePosition(GROUP\_TYPE\_ROBOT); | |

### 设置轴限位信息

HMCErrCode setLimitInfo(const GROUP\_TYPE &group, int index, double negtive, double positive, bool en);

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode setLimitInfo(const GROUP\_TYPE &group, int index, double negtive, double positive, bool en); | |
| 参数 | group | 组名   * GROUP\_TYPE\_ROBOT 内部轴 * GROUP\_TYPE\_EXT\_AXES 外部轴 |
| index | 第几轴，范围：≥1 |
| negtive | 负方向软限位值 |
| positive | 正方向软限位值 |
| en | 软限位使能开关 |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  ret = hscapi->setLimitInfo(GROUP\_TYPE\_ROBOT, 1, -120, 120, true); | |
| 备注 | 软限位设置后立即生效。 | |

### 获取轴限位信息

HMCErrCode getLimitInfo(const GROUP\_TYPE &group, int index, double &negtive, double &positive, bool &en);

|  |  |  |
| --- | --- | --- |
|  | HMCErrCodegetLimitInfo(const GROUP\_TYPE &group, int index, double &negtive, double &positive, bool &en); | |
| 参数 | group | 组名   * GROUP\_TYPE\_ROBOT 内部轴 * GROUP\_TYPE\_EXT\_AXES 外部轴 |
| index | 第几轴，范围：≥1 |
| negtive | 负方向软限位值 |
| positive | 正方向软限位值 |
| en | 软限位使能开关 |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  double neg, pos;  bool en;  ret = hscapi->getLimitInfo(GROUP\_TYPE\_ROBOT, 1, neg, pos, en); | |

### 保存内部轴软限位

HMCErrCode saveRobPlimit();

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode saveRobPlimit(); | |
| 参数 | 无 | |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  ret = hscapi->saveRobPlimit(); | |
| 备注 | 没有保存的软限位更改会在IPC重启后丢失。 | |

### 保存外部轴软限位

HMCErrCode saveEaPlimit();

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode saveEaPlimit(); | |
| 参数 | 无 |  |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  ret = hscapi->saveEaPlimit(); | |
| 备注 | 没有保存的软限位更改会在IPC重启后丢失。 | |

## IO设置

### 数字输入输出端

#### 获取数字输入端总数

HMCErrCode getDinNum(int& value);

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode getDinNum(int& value); | |
| 参数 | value | 值 |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  int value;  ret = hscapi->getDinNum(value); | |

#### 获取数字输出端总数

HMCErrCode getDoutNum(int& value);

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode getDoutNum(int& value); | |
| 参数 | value | 值 |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  int value;  ret = hscapi->getDoutNum(value); | |

#### 获取数字输入端起始id

HMCErrCode getInputStartId(int &value);

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode getInputStartId(int &value); | |
| 参数 | value | 值 |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  int value;  ret = hscapi->getInputStartId(value); | |
| 备注 | 起始id仅对IPC内部有意义。  对于本库所提供的接口，默认IO索引从1开始。 | |

#### 获取数字输出端起始id

HMCErrCode getOutputStartId(int &value);

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode getOutputStartId(int &value); | |
| 参数 | value | 值 |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  int value;  ret = hscapi->getOutputStartId(value); | |
| 备注 | 起始id仅对IPC内部有意义。  对于本库所提供的接口，默认IO索引从1开始。 | |

#### 获取每个通道的IO数量

HMCErrCode getPortCountPerSlice(int &value);

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode getPortCountPerSlice(int &value); | |
| 参数 | value | 值 |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  int value;  ret = hscapi->getPortCountPerSlice(value); | |

#### 获取数字输入端通道数量

HMCErrCode getInputSliceCount(int &value);

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode getInputSliceCount(int &value); | |
| 参数 | value | 值 |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  int value;  ret = hscapi->getInputSliceCount(value); | |
| 备注 | 数字输入端数量 = 数字输入端通道数量 × 每个通道的IO数量 | |

#### 获取数字输出端通道数量

HMCErrCode getOutputSliceCount(int &value);

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode getOutputSliceCount(int &value); | |
| 参数 | value | 值 |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  int value;  ret = hscapi->getOutputSliceCount(value); | |
| 备注 | 数字输出端数量 = 数字输出端通道数量 × 每个通道的IO数量 | |

#### 获取IO的虚拟状态

HMCErrCode getIOGroupVirtualFlag(const IOType & io, const int &group, int &value);

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode getIOGroupVirtualFlag(const IOType & io,const int &group,int &value); | |
| 参数 | Io | IO类型   * IO\_TYPE\_DIN 数字输入 * IO\_TYPE\_DOUT 数字输出 |
| Group | IO组号，范围：≥1 |
| Value | 一组IO的状态 |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  int state;  ret = hscapi->getIOGroupVirtualFlag(IO\_TYPE\_DIN, 1, state); | |
| 备注 | * 数字输出输出的状态分为**REAL**（真实）和**VIRTUAL**（虚拟）两种，在不同状态下操作并不相同：   VIRTUAL状态数字输入端，可以设置值，作为虚拟的输入；  REAL状态数字输入端，设置端口值不会起作用；  VIRTUAL状态数字输出端，可以设置值，但不会反映在模块端口；  REAL状态数字输出端，可以设置值，且会反映在模块端口上。   * IO信息可分组获取，1组为**32个**IO，例如64个输入端的分组为：   组1：1~32号输入端  组2：33~64号输入端   * 获取到的value为32位整数，**从低到高位**分别对应32个IO的状态，其中1代表VIRTUAL，0代表REAL | |

#### 获取所有数字I/O的状态

HMCErrCode getAllIOState(IOType io, std::vector<bool>& vecState);

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode getAllIOState(IOType io, std::vector<bool>& vecState); | |
| 参数 | Io | IO类型   * IO\_TYPE\_DIN 数字输入 * IO\_TYPE\_DOUT 数字输出 |
| vecState | 所有IO的状态 |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  std::vector<bool> state;  ret = hscapi->getAllIOState(IO\_TYPE\_DIN, state); | |

#### 设置虚拟输入标志

HMCErrCode SetVinFlag(unsigned short inputId, bool stat);

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode SetVinFlag(unsigned short inputId, bool stat); | |
| 参数 | inputId | 数字输入端索引，范围：≥1 |
| Stat | 状态   * true 虚拟（VIRTUAL） * false 真实（REAL） |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  ret = hscapi->SetVinFlag(1, true); | |
| 备注 | * 改变运行模式后，所有数字输入输出端的状态将恢复为REAL | |

#### 设置虚拟输出标志

HMCErrCode SetVoutFlag(unsigned short outputId, bool stat);

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode SetVoutFlag(unsigned short outputId, bool stat); | |
| 参数 | outputId | 数字输出端索引，范围：≥1 |
| Stat | 状态   * true 虚拟（VIRTUAL） * false 真实（REAL） |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  ret = hscapi->SetVoutFlag(1, true); | |
| 备注 | * 改变运行模式后，所有数字输入输出端的状态将恢复为REAL。 | |

#### 获取IO的值

HMCErrCode getIOGroup(const IOType & io, const int &group, int &value);

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode getIOGroup(const IOType & io,const int &group,int &value); | |
| 参数 | Io | IO类型   * IO\_TYPE\_DIN 数字输入 * IO\_TYPE\_DOUT 数字输出 |
| Group | IO组号，范围：≥1 |
| Value | 一组IO的值 |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  int value;  ret = hscapi->getIOGroup(IO\_TYPE\_DIN, 1, value); | |
| 备注 | * IO信息可分组获取，1组为**32个**IO，例如64个输入端的分组为：   组1：1~32号输入端  组2：33~64号输入端  获取到的value为32位整数，**从低到高位**分别对应32个IO的值 | |

#### 获取所有数字I/O的值

HMCErrCode getAllIOValue(IOType io, std::vector<bool>& vecValue);

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode getAllIOValue(IOType io, std::vector<bool>& vecValue); | |
| 参数 | Io | IO类型   * IO\_TYPE\_DIN 数字输入 * IO\_TYPE\_DOUT 数字输出 |
| vecValue | 所有IO的值 |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  std::vector<bool> value;  ret = hscapi->getAllIOValue(IO\_TYPE\_DIN, value); | |

#### 设置IO值

HMCErrCode setIOValue(IOType io, unsigned short ioId, bool value);

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode setIOValue(IOType io, unsigned short ioId, bool value); | |
| 参数 | Io | IO类型   * IO\_TYPE\_DIN 数字输入 * IO\_TYPE\_DOUT 数字输出 |
| IoId | 索引，范围：≥1 |
| value | 值   * true ON * false OFF |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  ret = hscapi->setIOValue(IO\_TYPE\_DIN, 1, true); | |
| 备注 | * 改变运行模式后，所有数字输入输出端的值将恢复为模块的真实值。 | |

### 外部运行配置

#### 设置外部控制输入标志

HMCErrCode setInputSigMap(int lSysSigIndex , int lInputIndex);

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode setInputSigMap(int lSysSigIndex , int lInputIndex); | |
| 参数 | lSysSigIndex | 外部控制输入标志索引，范围：≥1 |
| lInputIndex | 数字输入索引，范围：≥1 |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  ret = hscapi->setInputSigMap(1, 1); | |
| 备注 | 不可以在自动运行模式下设置。  设置后立即生效。 | |

#### 设置外部控制输出标志

HMCErrCode setOutputSigMap(int lSysSigIndex , int lOutputIndex);

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode setOutputSigMap(int lSysSigIndex , int lOutputIndex); | |
| 参数 | lSysSigIndex | 外部控制输出标志索引，范围：≥1 |
| lOutputIndex | 数字输出索引，范围：≥1 |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  ret = hscapi->setOutputSigMap(1, 1); | |
| 备注 | 不可以在自动运行模式下设置。  设置后立即生效。 | |

#### 保存外部控制标志设置

HMCErrCode saveSignalReg();

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode saveSignalReg (); | |
| 参数 | 无 | |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  ret = hscapi->saveSignalReg (); | |
| 备注 | 没有保存的更改会在IPC重启后丢失。 | |

### 模拟输入输出端

#### 获取模拟输入端总数

HMCErrCode getIOGroup getAinNum(int& value);

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode getAinNum(int& value); | |
| 参数 | value | 值 |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  int value;  ret = hscapi->getAinNum(value); | |

#### 获取模拟输出端总数

HMCErrCode getAoutNum(int& value);

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode getAoutNum(int& value); | |
| 参数 | value | 值 |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  int value;  ret = hscapi->getAoutNum(value); | |

#### 获取模拟输入端起始id

HMCErrCode getAInputStartId(int &value);

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode getAInputStartId(int &value); | |
| 参数 | value | 值 |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  int value;  ret = hscapi->getAInputStartId(value); | |
| 备注 | 起始id仅对IPC内部有意义。  对于本库所提供的接口，默认IO索引从1开始。 | |

#### 获取模拟输出端起始id

HMCErrCode getAOutputStartId(int &value);

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode getAOutputStartId(int &value); | |
| 参数 | value | 值 |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  int value;  ret = hscapi->getAOutputStartId(value); | |
| 备注 | 起始id仅对IPC内部有意义。  对于本库所提供的接口，默认IO索引从1开始。 | |

#### 获取模拟输入端通道数量

HMCErrCode getAInputPortCount(int &value);

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode getAInputPortCount(int &value); | |
| 参数 | value | 值 |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  int value;  ret = hscapi->getAInputPortCount(value); | |
| 备注 | 模拟输入端数量 = 模拟输入端通道数量 × 每个通道的IO数量 | |

#### 获取模拟输出端通道数量

HMCErrCode getAOutputPortCount(int &value);

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode getAOutputPortCount(int &value); | |
| 参数 | value | 值 |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  int value;  ret = hscapi->getAOutputPortCount(value); | |
| 备注 | 模拟输出端数量 = 模拟输出端通道数量 × 每个通道的IO数量 | |

#### 获取输入电压最大值

HMCErrCode getAInputMaxValue(double &value);

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode getAInputMaxValue(double &value); | |
| 参数 | value | 值（单位：V） |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  double value;  ret = hscapi->getAInputMaxValue(value); | |

#### 获取输出电压最大值

HMCErrCode getAOutputMaxValue(double &value);

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode getAOutputMaxValue(double &value); | |
| 参数 | value | 值（单位：V） |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  double value;  ret = hscapi->getAOutputMaxValue(value); | |

#### 获取输入电压最小值

HMCErrCode getAInputMinValue(double &value);

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode getAInputMinValue(double &value); | |
| 参数 | value | 值（单位：V） |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  double value;  ret = hscapi->getAInputMinValue(value); | |

#### 获取输出电压最小值

HMCErrCode getAOutputMinValue(double &value);

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode getAOutputMinValue(double &value); | |
| 参数 | value | 值（单位：V） |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  double value;  ret = hscapi->getAOutputMinValue(value); | |

#### 获取输入端模拟量精度

HMCErrCode getAInputAccValue(double &value);

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode getAInputAccValue(double &value); | |
| 参数 | value | 值（单位：V） |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  double value;  ret = hscapi->getAInputAccValue(value); | |

#### 获取输出端模拟量精度

HMCErrCode getAOutputAccValue(double &value);

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode getAOutputAccValue(double &value); | |
| 参数 | value | 值（单位：V） |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  double value;  ret = hscapi->getAOutputAccValue(value); | |

#### 获取模拟输入端的值

HMCErrCode GetAInputValue(const int id, double &value);

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode GetAInputValue(const int id, double &value); | |
| 参数 | Id | 输入端id，范围：≥1 |
| value | 值 |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  double value;  ret = hscapi->GetAInputValue(1, value); | |

#### 获取模拟输出端的值

HMCErrCode GetAOutputValue(const int id, double &value);

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode GetAOutputValue(const int id, double &value); | |
| 参数 | Id | 输出端id，范围：≥1 |
| value | 值 |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  double value;  ret = hscapi->GetAOutputValue(1, value); | |

#### 设置模拟输出端的值

HMCErrCode setAOutputValue(int id, double value);

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode setAOutputValue(int id, double value); | |
| 参数 | Id | 输出端id，范围：≥1 |
| value | 值 |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  ret = hscapi->setAOutputValue(1, 1.0); | |

### 获取IO信息

HMCErrCode getIOGroupInfo(IOGroupInfo &datas);

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode getIOGroupInfo(IOGroupInfo &datas); | |
| 参数 | datas | 组信息，包括所有数字输入输出端的状态和值、模拟输入输出端的值 |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  IOGroupInfo datas;  ret = hscapi->getIOGroup(datas); | |

## 用户PLC设置

### 设置用户PLC

HMCErrCode setUserPlcState (int en);

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode setUserPlcState (int en); | |
| 参数 | en | 0：关闭用户PLC功能；1：打开用户PLC功能 |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = hscapi->setUserPlcState(1); //打开用户PLC功能 | |

### 获取用户PLC使能状态

HMCErrCode getUserPlcState (int &en);

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode getUserPlcState (int &en); | |
| 参数 | en | 0：关闭用户PLC功能；1：打开用户PLC功能 |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  int en;  ret = hscapi->getUserPlcState(en); //获取用户PLC功能 | |

### 判断用户PLC运行状态

HMCErrCode userPlcIsRunning(int &run);

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode userPlcIsRunning(int &run); | |
| 参数 | Run | 0：停止；非0：运行 |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  int en;  ret = hscapi->userPlcIsRunning(en); //获取用户PLC功运行状态 | |

### 启动PLC

HMCErrCode startUserPlc(int &state);

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode startUserPlc(int &state); | |
| 参数 | State | 0：成功；非0：失败 |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  int en;  ret = hscapi->startUserPlc(en); //启动PLC | |

### 停止PLC

HMCErrCode stopUserPlc(int &state);

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode stopUserPlc(int &state); | |
| 参数 | State | 0：成功；非0：失败 |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  int en;  ret = hscapi->stopUserPlc(en); //停止PLC | |

## 其他指令

### 获取系统信息

HMCErrCode GetSysMessage(int& type, int& num, std::string& strMessage, unsigned long ulWaitTime);

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode GetSysMessage(int& type, int& num, std::string& strMessage, unsigned long ulWaitTime); | |
| 参数 | type | 报警类型（0：Info；1：Note；2：Error；-1：其它） |
| num | 获取到的错误号 |
| strMessage | 获取到的报警类容 |
| ulWaitTime | 等待超时时间（毫秒），建议大于200 |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  int type;  int num;  std::string msg;  ret = hscapi->GetSysMessage(type, num, msg, 500); | |
| 备注 | * 由于调用该接口的线程会被阻塞，因此**应在一个专用线程里调用**。 * 接口在调用时阻塞，在下面三种情况下退出：   等待超时，返回0x800002，num=0，strMessage=””；  限定时间内获取到报警，返回0  出现其它错误，返回非0，num=0，strMessage=””。 | |

### 清除错误

HMCErrCode clearFault();

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode clearFault (); | |
| 参数 | 无 | |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  ret = hscapi->clearFault (); | |
| 备注 | 接口需求：驱动器错误会导致机器人无法继续使用，调用本接口可以清除驱动器错误。  使用建议：出现驱动器错误时，必定会上报类型为Error（即type=2）的报警（反过来不成立），开发者应该实现一种方式（例如点击按钮）让用户触发调用本接口，以确保可以清除驱动器错误。 | |

### 设置语言

HMCErrCode SetMessageLang(const int &no);

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode SetMessageLang(const int &no); | |
| 参数 | no | 0：英语；1：中文 |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  ret = hscapi->SetMessageLang(0);  ret = hscapi->SetMessageLang(1); | |
| 备注 | * 使用中文必须添加一个翻译文件，否则无效。 | |

### 开启打印

HMCErrCode setPrintDebug(bool en);

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode setPrintDebug(bool en); | |
| 参数 | en | 打印状态 |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  ret = hscapi->setPrintDebug(1); //打开 | |
| 备注 | 本接口无实际功能，不必使用。 | |

### 获取SO（库）版本

HMCErrCode GetSOVersion(std::string& value);

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode GetSOVersion(std::string& value); | |
| 参数 | value | 值 |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  std::string ver;  ret = hscapi->GetSOVersion(ver); | |
| 备注 | HscApi版本号。 | |

### 获取HMCApi版本

std::string getHMCVersion();

|  |  |  |
| --- | --- | --- |
|  | std::string getHMCVersion(); | |
| 参数 | 无 | |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | std::string ver = hscapi->getHMCVersion(); | |
| 备注 | HMCApi版本号。 | |

## 通用接口

HMCErrCode executeCmdResponse(const std::string& cmdStr, std::string& response, int priority);

|  |  |  |
| --- | --- | --- |
|  | HMCErrCode executeCmdResponse(const std::string& cmdStr, std::string& response, int priority); | |
| 参数 | cmdStr | 要执行的机器人语言命令 |
| response | 执行命令的返回信息 |
| priority | 命令处理的优先级，范围：[0, 4] |
| 返回值 | HMCErrCode | 0：成功；其他：失败 |
| 示例 | HMCErrCode ret = 0;  ret = hscapi->executeCmdResponse(“do something”, outstr, 3); | |
| 备注 | 关于命令优先级priority，越小的值优先级越高。  一般来说，设置类的命令优先级较高，获取类的命令优先级较低（例如使能信号优先级为2，设置寄存器优先级为3，读取寄存器优先级为4）。 | |

# 常见语法使用技巧

## 寄存器操作

### 操作接口

寄存器操作包括设置、读取和保存。

设置和读取是立刻生效的，保存的作用是为了在控制器重启后仍保留所设置的值。

下面是各类寄存器对应不同操作的接口整理：

表格 1 设置寄存器

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| IR | DR | JR | LR | REF | BASE | TOOL |
| SetIR | SetDR | SetJR | SetLR |  | setBaseFrame | setToolFrame |
| setVar | setVar | setVar | setVar | setVar | setVar | setVar |

表格 2 读取寄存器

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| IR | DR | JR | LR | REF | BASE | TOOL |
| GetIR | GetDR | GetJR | GetLR | GetString | GetString | GetString |
| GetLong | GetDouble | GetJoint | GetLocation | GetJoint | GetLocation | GetLocation |
|  |  |  |  |  | getBaseFrame | getToolFrame |

表格 3 保存寄存器

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| IR | DR | JR | LR | REF | BASE | TOOL |
| saveVar | saveVar | saveVar | saveVar | saveVar | saveVar | saveVar |

### 例子

|  |  |
| --- | --- |
| **操作** | **代码** |
| 获取IR[1] | executeCmdResponse("?IR[1]", outStr, 3) |
| 设置IR[1] | executeCmdResponse("IR[1]=5", outStr, 3) |
| 设置IR[1] | setVar("IR[1]", "5") |
| 获取REF[1] | executeCmdResponse("?REF [1]", outStr, 3) |
| 设置REF[1] | executeCmdResponse("REF [1]={10, 10, 10, 10}", outStr, 3) |
| 设置REF[1] | setVar("REF[1]", "{10, 10, 10, 10}") |
| 获取BASE\_FRAME[1] | executeCmdResponse("?BASE\_FRAME [1]", outStr, 3) |
| 设置BASE\_FRAME[1] | executeCmdResponse("BASE\_FRAME [1]=#{10, 10, 10, 10}", outStr, 3) |
| 设置BASE\_FRAME[1] | setVar("BASE\_FRAME[1]", "#{10, 10, 10, 10}") |

## Boost学习

### 智能指针

智能指针是为了管理c++中的内存而诞生：

C++引入异常机制后，智能指针由一种一种技巧升级为一种非常重要的技术，因为如果没有智能指针，程序员必须保证new对象能在正确的时机delete，四处编写异常捕获代码以释放资源，而智能指针可以在退出作用域时-不管是正常流程离开或是因异常离开-总调用delete来析构在堆上动态分配的对象。

C++98的推出的“自动指针” std::auto\_ptr

int main()

{

auto\_ptr<class> p1(new class);

…

}

### Scoped\_ptr

### Shared\_ptr

### Shared\_ptr是一个最像指针的智能指针，是boos.smart\_ptr库中最有用价值，最重要的组成部分，也是最有用的，被C++11标准收录。

1. 类操作

Shared\_ptr比scope\_ptr复杂的多

1. 操作函数

Shared\_ptr与scope\_ptr都是用于管理new动态分配的智能指针：他们都重载了\*和-操作符以模仿原始指针的行为。并提供显示bool类型判断指针的有效性，get（）可以获取得到原始指针，并且不提供指针算术操作，也不管理new[]产生的动态数组指针。

1. Shared\_ptr<int> spi(new int);

Assert(spi) ;

\*spi =222;

Shared\_ptr<string> sps(new string(“smart”));

Assert(sps->size() == 5);

Shared\_ptr<int> spi2(new int[10]);//危险不能正确的释放内存

## 单步运行

单步运行包括StepIn、StepOut、StepOver三个接口的使用。

|  |
| --- |
| <1>、SendUserProgFile (“F:”, “MAIN.PRG”)  <2>、ProgLoad ("MAIN.PRG")  <3>、 StepIn() //加载程序后调用，单步进入  <4>、StepOut() //单步跳出  <5>、StepOver() //单步执行 |

## 标定工具/基坐标系

### 标定工具坐标

标定工具坐标：

|  |
| --- |
| <1>、tpToolCalibrationType(MULTI\_POINT)  <2>、tpRemoveCalibrationPoint(1)  <3>、tpCalibratePoint(1)  <4>、tpRemoveCalibrationPoint(2)  <5>、tpCalibratePoint(2)  <6>、tpToolCalibration(1)  <7>、saveTpToolCalibration() |

### 标定基坐标

标定基坐标：

|  |
| --- |
| <1>、setCalibrateBaseValue(1, value笛卡尔坐标值) //#{10, 10, 10, 10}  <2>、setCalibrateBaseValue(2, value笛卡尔坐标值) //#{20, 20, 20, 20}  <3>、setCalibrateBaseValue(3, value笛卡尔坐标值) //#{30, 30, 30, 30}  <4>、calibrateBase(1)  <5>、saveCalibrateBase() |

## 手动运行

手动运行（调用StartMoving）之前要设置相关模式：

|  |
| --- |
| <1>、SetEnableState(ENABLE) //打开使能  <2>、SetWorkMode(WORK\_MODE\_T1) //设置为手动运行模式  <3>、setScreen(SCREEN\_JOG) //设置为点动窗口  <4>、setMotionGroup(GROUP\_TYPE\_ROBOT) //设置为内部轴运动  <5>、setFrame(COORD\_TYPE\_JOINT) //设置为轴坐标系  <6>、StartMoving(1, DIRECTION\_POSITIVE) //1轴正向运动开始 |

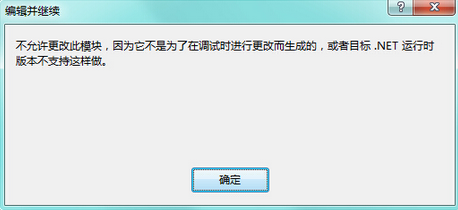
# 关于VS项目平台的x86,x64,Any CPU以及Debug和Release的区别

## x86、x64、Any CPU的区别

1. x86平台编译出来的exe(可执行文件)或dll(动态链接库)都是32位的。以此类推，x64对应的则是64位的。而Any CPU则是取决于当前的操作系统，若操作系统是32位的，则编译出来的程序就是32位的，反之编译出来的就是64位程序。
2. 若你的启动项目，即主程序（编译出来是exe文件的）是x86平台下编译的，而它所依赖的一个项目（或动态链接库）是由x64位平台编译出来的，则会提示“未能加载文件或程序集…或它的某一个依赖性。试图加载格式不正确的程序。”之类的错误。这是因为32位程序不能加载64位的dll，更不能调用其中的类、方法和对象等。反之，若主程序是由x64平台编译出来的，而dll是x86的呢？答案。。也是不行的！**说到这里，基本上可以确定一点：dll和主程序的生成平台只要一致就可以**。那么问题来了！若主程序是Any CPU编译的，而dll是由x86或者x64平台编译的，又或者，主程序是x86或者x64平台编译的，而dll是由Any CPU编译的呢？正确答案就是两者在“特殊情况”下都是可行的。为什么说是“特殊情况”下呢？因为第一点（红色字体处）也提到了，Any CPU取决于操作系统，而结合第二点上半部分（红色字体处）提到的，只要dll和主程序的生成平台一致，那么也是可行的。
3. **由上可知：若主程序是在Any CPU平台下编译的，那么编译dll的平台必须要和编译主程序的操作系统一致才行。若主程序是在x86或者x64平台下编译的，那么dll就必须要和主程序一致。不过有个特例：若dll由Any CPU编译的，那么这个dll就可以被32位和64位的主程序调用。**这是为什么呢？请看第三点。
4. 虽说Any CPU编译出来的程序取决于操作系统，**但是由Any CPU编译出来dll则取决于调用它的主程序**，即若主程序是32位的，那么dll也是32位的，若主程序是64位的，那么dll就是64位的。所以dll一般采用Any CPU平台编译，而主程序一般采用x86平台编译。
5. **使用x86平台调试的时候，是可以一边调试一边修改代码的，而x64平台下则不行（启用了编辑并继续的）。**
6. **总结：**

总结：了解了它们的区别之后，我们打包程序的时候就知道如何取舍了。一般来说，在不清楚客户的电脑是什么类型的操作系统的情况下，或者两者都有的时候，则主程序采用x86平台编译，dll采用Any CPU平台编译。若清楚的知道是64位操作系统，那么就可以都按x64平台编译了。

## Debug和Release的区别

Debug通常称为调试版本，它包含调试信息，并且不作任何优化，便于程序员调试程序。Release称为发布版本，它往往是进行了各种优化，使得程序在代码大小和运行速度上都是最优的，以便用户很好地使用。所以一般都是用Release来打包发布程序，另外，Release打包出来的程序也相对来说会小一些。   
在Release模式下是不能一边调试一边修改代码的（启用了编辑并继续的）。调试时修改代码提示图： 

# C++中常见的头文件包含方法及分类

## C++中第三方库的3中文件

1. Include是存放工程所需要的头文件的文件夹，比如宏、结构定义、函数定义、类型定义等等。
2. Bin包含可执行文件，存放的是编译器本身的可执行二进制文件。DLL(Dynamic Linkable Library)文件，动态链接库，它提供给你一些可以直接拿来用的变量、函数或类。

只要遵循约定的DLL接口规范和调用方式，用各种语言编写的DLL都可以相互调用。譬如Windows提供的系统DLL（其中包括了Windows的API），在任何开发环境中都能被调用，不在乎其是Visual Basic、Visual C++还是Delphi。

1. Lib文件是库文件（与DLL文件相类似），存放了静态链接库文件，供其它程序调用的，直接打不开。一个lib文件是obj文件的集合查，看LIB和DLL文件都可以通过OLLYDBG中LOADDLL插件来反汇编查看各个函数的过程。
2. Dll与lib的区别

* lib是编译时需要的，dll是运行时需要的。

如果要完成[源代码](http://baike.baidu.com/item/%E6%BA%90%E4%BB%A3%E7%A0%81)的编译，有lib就够了。

如果要使动态连接的程序运行起来，有dll就够了。

在开发和调试阶段，当然最好都有。

* 一般的动态库程序有lib文件和dll文件。lib文件是必须在编译期就连接到应用程序中的，而dll文件是运行期才会被调用的。如果有dll文件，那么对应的lib文件一般是一些索引信息，具体的实现在dll文件中。如果只有lib文件，那么这个lib文件是[静态编译](http://baike.baidu.com/item/%E9%9D%99%E6%80%81%E7%BC%96%E8%AF%91)出来的，索引和实现都在其中。静态编译的lib文件有好处：给用户安装时就不需要再挂动态库了。但也有缺点，就是导致应用程序比较大，而且失去了动态库的灵活性，在[版本升级](http://baike.baidu.com/item/%E7%89%88%E6%9C%AC%E5%8D%87%E7%BA%A7)时，同时要发布新的应用程序才行。
* 在动态库的情况下，有两个文件，一个是引入库（.LIB）文件，一个是DLL文件，引入库文件包含被DLL导出的函数的名称和位置，DLL包含实际的函数和数据，应用程序使用LIB文件链接到所需要使用的DLL文件，库中的函数和数据并不复制到[可执行文件](http://baike.baidu.com/item/%E5%8F%AF%E6%89%A7%E8%A1%8C%E6%96%87%E4%BB%B6)中，因此在应用程序的可执行文件中，存放的不是被调用的函数代码，而是DLL中所要调用的函数的[内存地址](http://baike.baidu.com/item/%E5%86%85%E5%AD%98%E5%9C%B0%E5%9D%80)，这样当一个或多个应用程序运行时再把程序代码和被调用的函数代码链接起来，从而节省了内存资源。从上面的说明可以看出，DLL文件必须随应用程序一起发行，否则应用程序将会产生错误。

# 第三方库的添加和应用

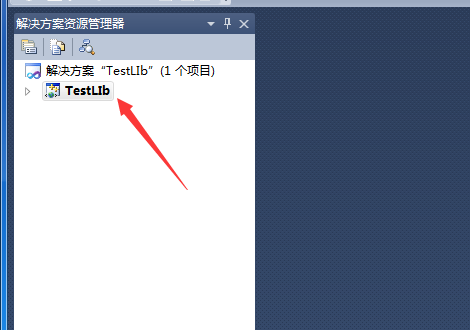
## 第三方库的意义

## 第三方库的的添加流程

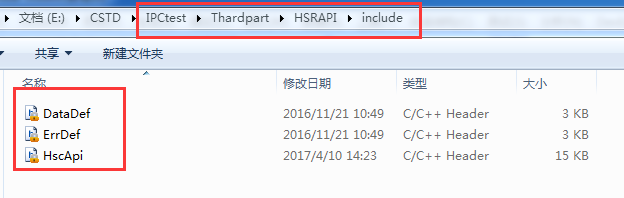
1. 添加华数机器人控制器API接口函数

**C++添加第三方库文件的基本流程**

1. 首先新建工程

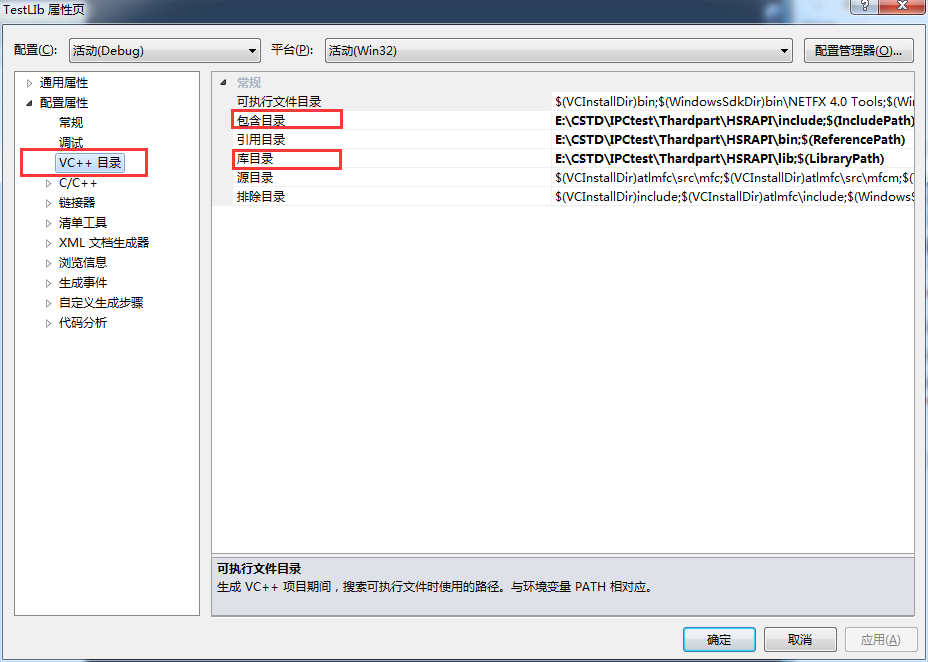


1. 将第三库对应的.h文件添加到工程中

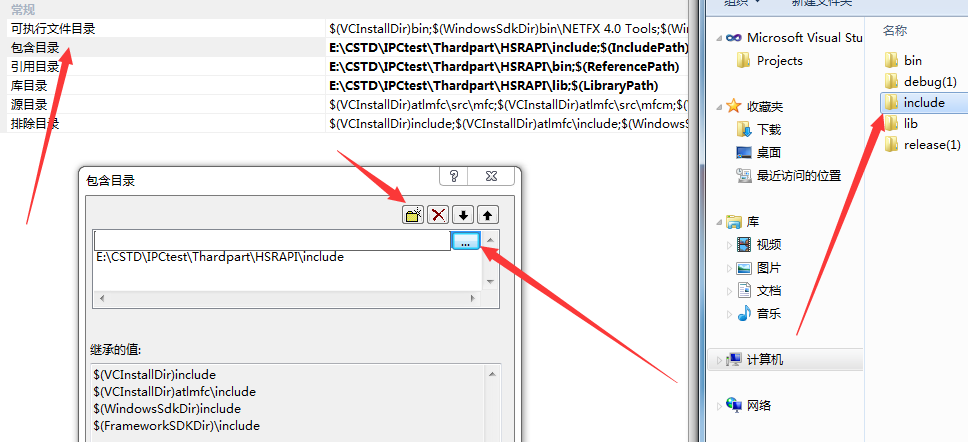


头文件不需要移动，就放置在第三方库中，直接通过添加，添加到工程中。

1. 添加对应的VC++目录



1. 添加包含目录（include）



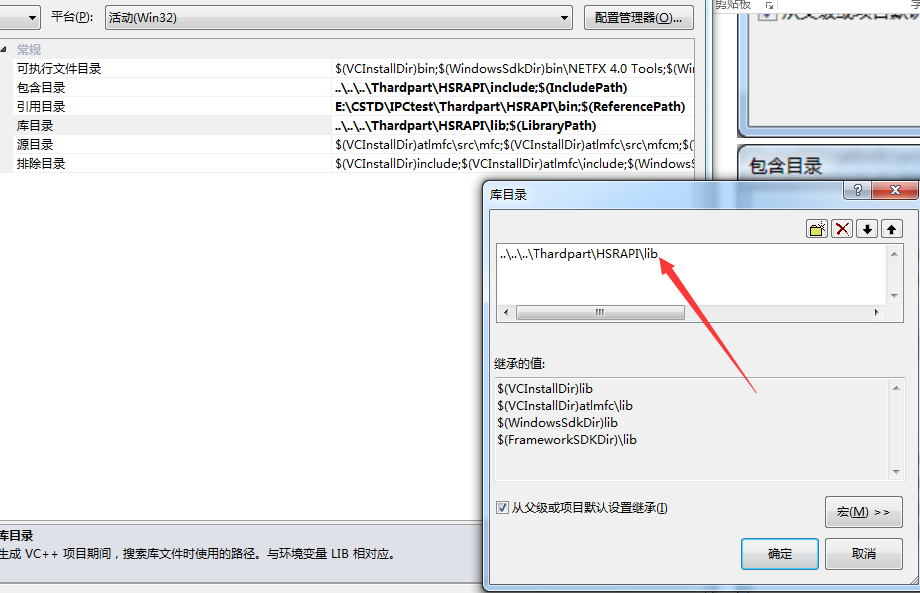
找到第三方库下的include文件夹选择确认。此时include目录已经添加完毕

为了便于工程的移植可以将路径设置为相对路径：

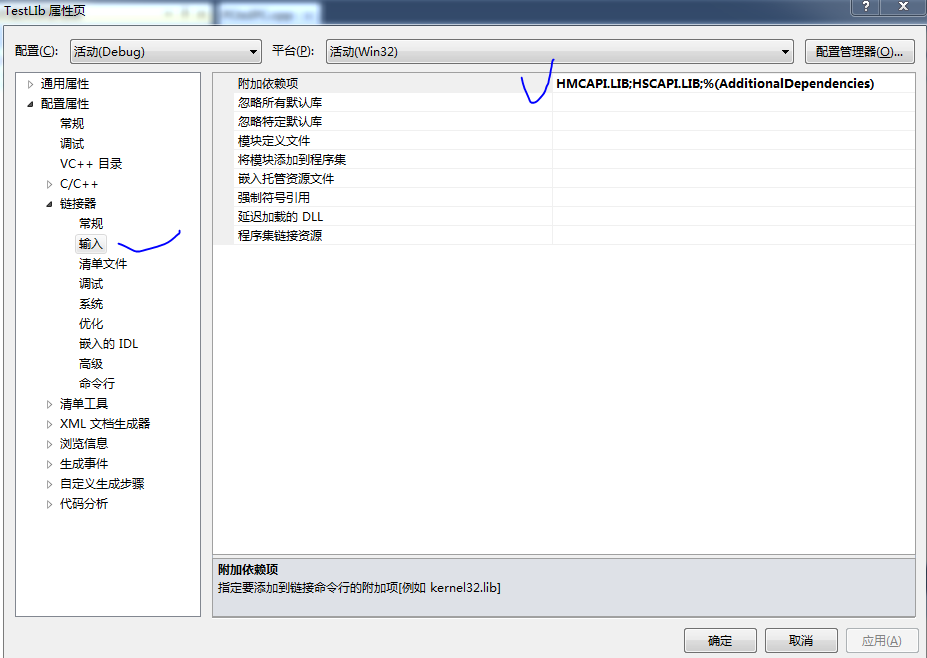
..\..\..\Thardpart\HSRAPI\include

1. 添加库目录文件

使用相对搜索路径，将库文件添加到工程目录下

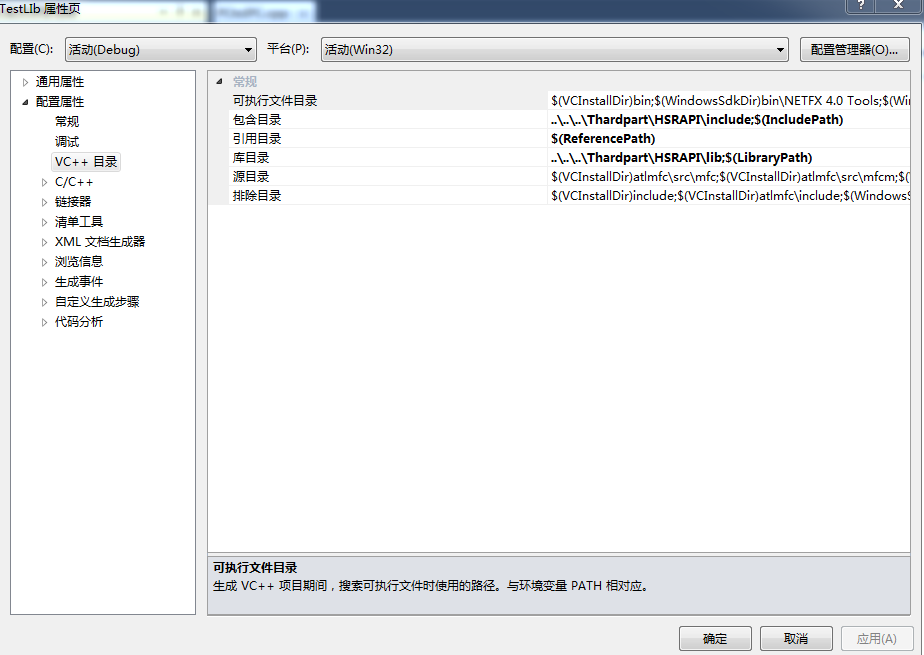


1. 添加链接库中的输入（附加依赖项）



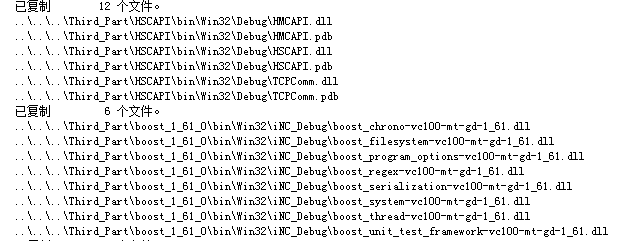
链接库中的附加依赖项为Lib中的文件夹下的lib文件，分别在附加依赖项中输入对应的名称。

1. 添加完成的最终目录文件



## 第三方库的分类

## 第三方库的执行过程



## 第三方库的意义

# 附录

## 附录1 寄存器索引值范围

各寄存器索引值范围：

|  |  |
| --- | --- |
| **寄存器类型** | **索引值范围** |
| IR | [1, 100] |
| DR | [1, 100] |
| JR | [1, 100] |
| LR | [1, 100] |
| BASEFRAME | [1, 16] |
| TOOLFRAME | [1, 16] |
| ER | [1, 100] |

## 附录2 数据类型定义（DataDef.h）

|  |
| --- |
| #pragma once  #include <string>  #include <vector>  typedef unsigned long HMCErrCode;  //笛卡尔坐标  typedef std::vector<double> DcartPos;  //关节坐标  typedef std::vector<double> JointPos;  //关节属性  typedef std::vector<int> JointConfig;  struct HscDevice  {  std::string strName; //控制器名  std::string strIP; //IP地址  std::string strSN; //SN序列号  };  enum enable  {  DISABLE,  ENABLE,  };  enum WorkMode  {  WORK\_MODE\_AUT,  WORK\_MODE\_T1,  WORK\_MODE\_T2,  WORK\_MODE\_EXT,  WORK\_MODE\_NONE,  };  enum FrameType  {  COORD\_TYPE\_JOINT,  COORD\_TYPE\_BASE,  COORD\_TYPE\_TOOL,  COORD\_TYPE\_WORLD,  COORD\_TYPE\_NONE,  };  enum TpScreen  {  // SCREEN\_DIAG,  // SCREEN\_STP,  SCREEN\_JOG,  SCREEN\_PRG,  // SCREEN\_DATA,  // SCREEN\_IO,  // SCREEN\_TRM,  SCREEN\_NONE,  };  enum Direction  {  DIRECTION\_POSITIVE = 0,  DIRECTION\_NEGTIVE = 1,  };  enum VarName  {  EXT\_PRG,  REF,  TOOL\_FRAME,  BASE\_FRAME,  IR,  DR,  JR,  LR,  ER,  };  enum WorkGroup  {  ROBOT,  EXT\_AXES,  };  enum MoveType  {  MOVE,  MOVES,  };  //组类型  enum GROUP\_TYPE  {  GROUP\_TYPE\_ROBOT,  GROUP\_TYPE\_EXT\_AXES,  };  enum IOType  {  IO\_TYPE\_DIN,  IO\_TYPE\_DOUT,  // IO\_TYPE\_AIN,  // IO\_TYPE\_AOUT,  };  enum IOState  {  IO\_OFF,  IO\_ON,  };  enum WorkState  {  WORK\_STATE\_START,  WORK\_STATE\_PAUSE,  WORK\_STATE\_STOP,  WORK\_STATE\_ALARM,  };  enum ProgMode  {  PROG\_MODE\_ONECE, //单周，程序只运行一遍  PROG\_MODE\_STEP = 6, //单步  PROG\_MODE\_REPEAT = 4, //循环，程序循环运行  };  enum ProgState  {  STATE\_RUNNING = 1,  STATE\_STOPPED,  STATE\_ERROR = 4,  STATE\_TERMINATED,  STATE\_READY = 7,  STATE\_KILLSTART = 9,  STATE\_KILLED,  };  class TaskStatus  {  public:  ProgState state; //取模256(258%256 = 2(STATE\_STOPPED))，即2和258都是stopped状态  int error; //错误代码  int source; //当前运行行号  std::string progName; //主程序名  std::string currFileName; //当前运行程序名  };  enum CALIBRATIONTYPE  {  SINGLE\_POINT, //单点  MULTI\_POINT, //多点  ORIENTATION\_POINT, //  OTHER\_POINT, //其他  };  struct StateData  {  int en; //0-disable 1-enable  TaskStatus prgState;  WorkMode workMode;  FrameType coord\_type;  int vord;  int tool\_frame\_num;  int base\_frame\_num;  int tp\_increment; //增量模式,目前没有用到  };  struct PosData  {  JointPos robotJointPos;  DcartPos robotDcartPos;  JointPos extJointPos;  };  struct IOGroupInfoget  {  //输入IO组值，如果有4个组，则返回4个整形数据的数组，如果对应组的某一位为1，则该IO位输出1  std::vector<int> DI\_GRP;  //输入IO组虚拟状态值，如果对应组的某一位为1，则该IO位为虚拟状态  std::vector<int> VI\_MASK;  //输出IO组值  std::vector<int> DO\_GRP;  //输出IO组虚拟状态标识值  std::vector<int> VO\_MASK;  //模拟量输入输出  std::vector<double> AO\_VALUE;  std::vector<double> AI\_VALUE;  }; |