**HmiRunTime新增通讯设备插件**

修订历史记录

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **日期** | **版本** | **作者** | **审核者** | **说明** |
| 2022-11-13 | V1.0 | 王茂房 |  | 初稿 |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

**目 录**

目录

[1.1 新建实现文件 3](#_Toc13158)

[1.2 实现通讯设备接口 3](#_Toc14239)

[1.3 添加文件至ModbusRTU.pro 24](#_Toc32364)

[1.4 编译ModbusRTU工程 24](#_Toc27404)

本文以设备ModbusRTU为例说明如何在HmiRunTime新增通讯设备。本文仅供参考，具体请看实现源码。

## 新建实现文件

在{文件HmiRunTime.pro所在目录} \HmiRunTime\Vendor\新建**ModbusRTU**目录, 在**ModbusRTU**目录新建如下文件:

|  |  |
| --- | --- |
| **ModbusRTU.cpp** | **通讯协议实现类源文件** |
| **ModbusRTU.h** | 通讯协议实现类头文件 |
| **ModbusRTUImpl.cpp** | 通讯设备实现类源文件 |
| **ModbusRTUImpl.h** | 通讯设备实现类头文件 |
| **ModbusRTU.json** | 插件描述 |
| **ModbusRTU.pro** | 工程文件管理文件 |
| **ModbusRTU\_dependencies.pri** | 项目依赖文件 |

文件内容可能有改动，详细内容请阅读源码文件！！

## 实现通讯设备接口

在类ModbusRTU实现通讯设备接口，文件位于目录{文件HmiRunTime.pro所在目录} \HmiRunTime\Vendors\IVendorPlugin，文件名：IVendorPlugin.h。

具体接口如下：

#ifndef IDEVICEPLUGIN\_H

#define IDEVICEPLUGIN\_H

#include "../../HmiRunTime/Port/IPort.h"

#include "../../HmiRunTime/shared/realtimedb.h"

#include <QStringList>

/\*

\* 注意本类不要派生自QObject

\*/

class **IVendorPlugin**

{

public:

virtual ~***IVendorPlugin***() {}

// 初始化设备

virtual bool ***initailizeDevice***(void\* pObj) = 0;

// 连接设备

virtual bool ***connectDevice***(void\* pObj) = 0;

// 断开设备连接

virtual bool ***disconnectDevice***(void\* pObj) = 0;

// 反初始化设备

virtual bool ***unInitailizeDevice***(void\* pObj) = 0;

// 写变量前处理

virtual bool ***beforeWriteIOTag***(void\* pObj, RunTimeTag\* pTag) = 0;

// 写变量

virtual int ***writeIOTag***(void\* pObj, IPort \*pPort, RunTimeTag\* pTag) = 0;

// 写变量后处理

virtual bool ***afterWriteIOTag***(void\* pObj, RunTimeTag\* pTag) = 0;

// 读变量前处理

virtual bool ***beforeReadIOTag***(void\* pObj, RunTimeTag\* pTag) = 0;

// 读变量

virtual int ***readIOTag***(void\* pObj, IPort \*pPort, RunTimeTag\* pTag) = 0;

// 读变量后处理

virtual bool ***afterReadIOTag***(void\* pObj, RunTimeTag\* pTag) = 0;

// 从块读变量拷贝寄存器数据至普通变量

virtual bool ***copyTagDataFromBlockReadTag***(RunTimeTag\* pBlockReadTag, RunTimeTag\* pTag) = 0;

// 设置块读变量缓冲区长度

virtual void ***setBlockReadTagBufferLength***(RunTimeTag\* pBlockReadTag) = 0;

// 变量字节序转换为当前主机字节序

virtual bool ***convertIOTagBytesToNativeBytes***(void\* pObj, RunTimeTag\* pTag) = 0;

};

QT\_BEGIN\_NAMESPACE

#define DevicePluginInterface\_iid "HmiRunTime.Device.PluginInterface"

Q\_DECLARE\_INTERFACE(IVendorPlugin, DevicePluginInterface\_iid)

QT\_END\_NAMESPACE

#endif // IDEVICEPLUGIN\_H

通讯设备接口实现请阅读文件ModbusRTU.cpp，ModbusRTU.h，ModbusRTUImpl.cpp，ModbusRTUImpl.h。

ModbusRTU.h具体实现源码如下：

#ifndef MODBUSRTU\_H

#define MODBUSRTU\_H

#include <QObject>

#include "../IVendorPlugin/IVendorPlugin.h"

#include "ModbusRTUImpl.h"

class **ModbusRTU** : public QObject, IVendorPlugin

{

Q\_OBJECT

Q\_PLUGIN\_METADATA(**IID** DevicePluginInterface\_iid **FILE** "ModbusRTU.json")

Q\_INTERFACES(**IVendorPlugin**)

public:

**ModbusRTU**();

~***ModbusRTU***() Q\_DECL\_OVERRIDE;

// 初始化设备

bool ***initailizeDevice***(void\* pObj) Q\_DECL\_OVERRIDE;

// 连接设备

bool ***connectDevice***(void\* pObj) Q\_DECL\_OVERRIDE;

// 断开设备连接

bool ***disconnectDevice***(void\* pObj) Q\_DECL\_OVERRIDE;

// 反初始化设备

bool ***unInitailizeDevice***(void\* pObj) Q\_DECL\_OVERRIDE;

// 写变量前处理

bool ***beforeWriteIOTag***(void\* pObj, RunTimeTag\* pTag) Q\_DECL\_OVERRIDE;

// 写变量

int ***writeIOTag***(void\* pObj, IPort \*pPort, RunTimeTag\* pTag) Q\_DECL\_OVERRIDE;

// 写变量后处理

bool ***afterWriteIOTag***(void\* pObj, RunTimeTag\* pTag) Q\_DECL\_OVERRIDE;

// 读变量前处理

bool ***beforeReadIOTag***(void\* pObj, RunTimeTag\* pTag) Q\_DECL\_OVERRIDE;

// 读变量

int ***readIOTag***(void\* pObj, IPort \*pPort, RunTimeTag\* pTag) Q\_DECL\_OVERRIDE;

// 读变量后处理

bool ***afterReadIOTag***(void\* pObj, RunTimeTag\* pTag) Q\_DECL\_OVERRIDE;

// 从块读变量拷贝寄存器数据至普通变量

bool ***copyTagDataFromBlockReadTag***(RunTimeTag\* pBlockReadTag, RunTimeTag\* pTag) Q\_DECL\_OVERRIDE;

// 设置块读变量缓冲区长度

void ***setBlockReadTagBufferLength***(RunTimeTag\* pBlockReadTag)Q\_DECL\_OVERRIDE;

// 变量字节序转换为当前主机字节序

bool ***convertIOTagBytesToNativeBytes***(void\* pObj, RunTimeTag\* pTag) Q\_DECL\_OVERRIDE;

private:

ModbusRTUImpl m\_modbusRTUImplObj;

};

#endif // MODBUSRTU\_H

ModbusRTU.cpp具体实现源码如下：

#include "ModbusRTU.h"

#include "../../HmiRunTime/Vendor.h"

#include "../../HmiRunTime/shared/publicfunction.h"

ModbusRTU::**ModbusRTU**()

{

}

ModbusRTU::~***ModbusRTU***()

{

}

/\*\*

\* @brief ModbusRTU::initailizeDevice

\* @details 初始化设备

\* @param pObj 设备描述对象

\* @return false-失败, true-成功

\*/

bool ModbusRTU::***initailizeDevice***(void\* pObj)

{

Vendor\* pVendorObj = (Vendor\*)(pObj);

if(pVendorObj) {

}

return true;

}

/\*\*

\* @brief ModbusRTU::connectDevice

\* @details 连接设备

\* @param pObj 设备描述对象

\* @return false-失败, true-成功

\*/

bool ModbusRTU::***connectDevice***(void\* pObj)

{

(void)pObj;

return true;

}

/\*\*

\* @brief ModbusRTU::disconnectDevice

\* @details 断开设备连接

\* @param pObj 设备描述对象

\* @return false-失败, true-成功

\*/

bool ModbusRTU::***disconnectDevice***(void\* pObj)

{

(void)pObj;

return true;

}

/\*\*

\* @brief ModbusRTU::unInitailizeDevice

\* @details 反初始化设备

\* @param pObj 设备描述对象

\* @return false-失败, true-成功

\*/

bool ModbusRTU::***unInitailizeDevice***(void\* pObj)

{

(void)pObj;

return true;

}

/\*\*

\* @brief ModbusRTU::beforeWriteIOTag

\* @details 写变量前处理

\* @param pObj 设备描述对象

\* @param pTag 变量描述对象

\* @return false-失败, true-成功

\*/

bool ModbusRTU::***beforeWriteIOTag***(void\* pObj, RunTimeTag\* pTag)

{

(void)pObj;

(void)pTag;

return true;

}

/\*\*

\* @brief ModbusRTU::writeIOTag

\* @details 写变量

\* @param pObj 设备描述对象

\* @param pPort 端口操作接口

\* @param pTag 变量描述对象

\* @return 0-失败, 1-成功

\*/

int ModbusRTU::***writeIOTag***(void\* pObj, IPort \*pPort, RunTimeTag\* pTag)

{

m\_modbusRTUImplObj.setPort(pPort);

if(!m\_modbusRTUImplObj.*isCanWrite*(pObj, pTag)) {

return 1;

}

return m\_modbusRTUImplObj.*writeData*(pObj, pTag);

}

/\*\*

\* @brief ModbusRTU::afterWriteIOTag

\* @details 写变量后处理

\* @param pObj 设备描述对象

\* @param pTag 变量描述对象

\* @return false-失败, true-成功

\*/

bool ModbusRTU::***afterWriteIOTag***(void\* pObj, RunTimeTag\* pTag)

{

(void)pObj;

(void)pTag;

return true;

}

/\*\*

\* @brief ModbusRTU::beforeReadIOTag

\* @details 读变量前处理

\* @param pObj 设备描述对象

\* @param pTag 变量描述对象

\* @return false-失败, true-成功

\*/

bool ModbusRTU::***beforeReadIOTag***(void\* pObj, RunTimeTag\* pTag)

{

(void)pObj;

(void)pTag;

return true;

}

/\*\*

\* @brief ModbusRTU::readIOTag

\* @details 读变量

\* @param pObj 设备描述对象

\* @param pPort 端口操作接口

\* @param pTag 变量描述对象

\* @return 0-失败, 1-成功

\*/

int ModbusRTU::***readIOTag***(void\* pObj, IPort \*pPort, RunTimeTag\* pTag)

{

m\_modbusRTUImplObj.setPort(pPort);

if(!m\_modbusRTUImplObj.*isCanRead*(pObj, pTag)) {

return 1;

}

return m\_modbusRTUImplObj.*readData*(pObj, pTag);

}

// 从块读变量拷贝寄存器数据至普通变量

bool ModbusRTU::***copyTagDataFromBlockReadTag***(RunTimeTag\* pBlockReadTag, RunTimeTag\* pTag)

{

int iOffset = pTag->addrOffset - pBlockReadTag->addrOffset;

quint8 \*pDes = pTag->dataFromVendor;

quint8 \*pSrc = pBlockReadTag->dataFromVendor;

if(pTag->dataType == TYPE\_BOOL) {

quint32 iByte = iOffset / 8;

quint32 iBit = iOffset % 8;

pDes[0] = (pSrc[iByte] >> iBit) & 0x01;

} else {

for(int x=0; x<pTag->bufLength; x++) {

pDes[x] = pSrc[iOffset \* 2 + x];

}

}

#if 0

qDebug() << "BlockReadTag: addr " << pBlockReadTag->addrOffset << hexToString((char \*)pBlockReadTag->dataFromVendor, pBlockReadTag->bufLength);

qDebug() << " Tag: addr " << pTag->addrOffset << hexToString((char \*)pTag->dataFromVendor, pTag->bufLength);

#endif

return true;

}

// 设置块读变量缓冲区长度

void ModbusRTU::***setBlockReadTagBufferLength***(RunTimeTag\* pBlockReadTag)

{

if(pBlockReadTag && pBlockReadTag->isBlockRead) {

if(pBlockReadTag->addrType.toLower() == "3x" || pBlockReadTag->addrType.toLower() == "4x") {

pBlockReadTag->bufLength \*= 2;

}

else if(pBlockReadTag->addrType.toLower() == "0x" || pBlockReadTag->addrType.toLower() == "1x") {

quint8 bytes = pBlockReadTag->bufLength / 8;

quint8 bits = pBlockReadTag->bufLength % 8;

pBlockReadTag->bufLength = bytes;

if(bits) {

pBlockReadTag->bufLength += 1;

}

}

}

}

/\*\*

\* @brief ModbusRTU::afterReadIOTag

\* @details 读变量后处理

\* @param pObj 设备描述对象

\* @param pTag 变量描述对象

\* @return false-失败, true-成功

\*/

bool ModbusRTU::***afterReadIOTag***(void\* pObj, RunTimeTag\* pTag)

{

(void)pObj;

(void)pTag;

return true;

}

///

/// \brief ModbusRTU::convertIOTagBytesToNativeBytes

/// \details 变量字节序转换为当前主机字节序

/// \param pObj 设备描述对象

/// \param pTag 变量描述对象

/// \return true-成功, false-失败

///

bool ModbusRTU::***convertIOTagBytesToNativeBytes***(void\* pObj, RunTimeTag\* pTag)

{

return m\_modbusRTUImplObj.convertIOTagBytesToNativeBytes(pObj, pTag);

}

ModbusRTUImpl.h具体实现源码如下：

#ifndef MODBUSRTUIMPL\_H

#define MODBUSRTUIMPL\_H

#include <QString>

#include <QObject>

#include "../../HmiRunTime/shared/realtimedb.h"

#include "../../HmiRunTime/shared/public.h"

#include "../Public/Modbus.h"

#include "../../HmiRunTime/Port/IPort.h"

class **ModbusRTUImpl** : public Modbus

{

public:

**ModbusRTUImpl**();

~***ModbusRTUImpl***();

public:

quint16 **crc16**(quint8 \*pbuf, qint32 len);

bool **messageCheck**(quint8 \*inBuf, qint16 bufLen);

bool ***isCanWrite***(void\* pObj, RunTimeTag\* pTag) override;

int ***writeData***(void\* pObj, RunTimeTag\* pTag) override;

bool ***isCanRead***(void\* pObj, RunTimeTag\* pTag) override;

int ***readData***(void\* pObj, RunTimeTag\* pTag) override;

private:

// 生成modbus报文

quint16 ***makeMessagePackage***(quint8 \*pSendData,

void\* pObj,

RunTimeTag\* pTag,

TModbus\_ReadWrite RW\_flag,

quint16 \*retVarLen) override;

};

#endif // MODBUSRTUIMPL\_H

ModbusRTUImpl.cpp具体实现源码如下：

#include "ModbusRTUImpl.h"

#include "../Public/DataPack.h"

#include "../../HmiRunTime/shared/publicfunction.h"

#include "../../HmiRunTime/Vendor.h"

#include <QDebug>

const quint8 auchCRCHi[] = {

0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1,

0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40,

0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1,

0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,

0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1,

0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,

0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1,

0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,

0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,

0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40,

0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0,

0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,

0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1,

0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,

0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,

0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,

0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,

0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41,

0x00, 0xC1, 0x81, 0x40

} ;

const quint8 auchCRCLo[] = {

0x00, 0xC0, 0xC1, 0x01, 0xC3, 0x03, 0x02, 0xC2, 0xC6, 0x06, 0x07, 0xC7, 0x05, 0xC5,

0xC4, 0x04, 0xCC, 0x0C, 0x0D, 0xCD, 0x0F, 0xCF, 0xCE, 0x0E, 0x0A, 0xCA, 0xCB, 0x0B,

0xC9, 0x09, 0x08, 0xC8, 0xD8, 0x18, 0x19, 0xD9, 0x1B, 0xDB, 0xDA, 0x1A, 0x1E, 0xDE,

0xDF, 0x1F, 0xDD, 0x1D, 0x1C, 0xDC, 0x14, 0xD4, 0xD5, 0x15, 0xD7, 0x17, 0x16, 0xD6,

0xD2, 0x12, 0x13, 0xD3, 0x11, 0xD1, 0xD0, 0x10, 0xF0, 0x30, 0x31, 0xF1, 0x33, 0xF3,

0xF2, 0x32, 0x36, 0xF6, 0xF7, 0x37, 0xF5, 0x35, 0x34, 0xF4, 0x3C, 0xFC, 0xFD, 0x3D,

0xFF, 0x3F, 0x3E, 0xFE, 0xFA, 0x3A, 0x3B, 0xFB, 0x39, 0xF9, 0xF8, 0x38, 0x28, 0xE8,

0xE9, 0x29, 0xEB, 0x2B, 0x2A, 0xEA, 0xEE, 0x2E, 0x2F, 0xEF, 0x2D, 0xED, 0xEC, 0x2C,

0xE4, 0x24, 0x25, 0xE5, 0x27, 0xE7, 0xE6, 0x26, 0x22, 0xE2, 0xE3, 0x23, 0xE1, 0x21,

0x20, 0xE0, 0xA0, 0x60, 0x61, 0xA1, 0x63, 0xA3, 0xA2, 0x62, 0x66, 0xA6, 0xA7, 0x67,

0xA5, 0x65, 0x64, 0xA4, 0x6C, 0xAC, 0xAD, 0x6D, 0xAF, 0x6F, 0x6E, 0xAE, 0xAA, 0x6A,

0x6B, 0xAB, 0x69, 0xA9, 0xA8, 0x68, 0x78, 0xB8, 0xB9, 0x79, 0xBB, 0x7B, 0x7A, 0xBA,

0xBE, 0x7E, 0x7F, 0xBF, 0x7D, 0xBD, 0xBC, 0x7C, 0xB4, 0x74, 0x75, 0xB5, 0x77, 0xB7,

0xB6, 0x76, 0x72, 0xB2, 0xB3, 0x73, 0xB1, 0x71, 0x70, 0xB0, 0x50, 0x90, 0x91, 0x51,

0x93, 0x53, 0x52, 0x92, 0x96, 0x56, 0x57, 0x97, 0x55, 0x95, 0x94, 0x54, 0x9C, 0x5C,

0x5D, 0x9D, 0x5F, 0x9F, 0x9E, 0x5E, 0x5A, 0x9A, 0x9B, 0x5B, 0x99, 0x59, 0x58, 0x98,

0x88, 0x48, 0x49, 0x89, 0x4B, 0x8B, 0x8A, 0x4A, 0x4E, 0x8E, 0x8F, 0x4F, 0x8D, 0x4D,

0x4C, 0x8C, 0x44, 0x84, 0x85, 0x45, 0x87, 0x47, 0x46, 0x86, 0x82, 0x42, 0x43, 0x83,

0x41, 0x81, 0x80, 0x40

} ;

ModbusRTUImpl::**ModbusRTUImpl**()

{

}

ModbusRTUImpl::~***ModbusRTUImpl***()

{

}

quint16 ModbusRTUImpl::***makeMessagePackage***(quint8 \*pSendData,

void\* pObj,

RunTimeTag\* pTag,

TModbus\_ReadWrite RW\_flag,

quint16 \*retVarLen)

{

quint16 mesPi = 0;

quint32 tmpDataPos = 0;

quint32 tmpUnit = 0;

quint8 byteCount = 0, tmpLen = 0;

Vendor\* pVendorObj = (Vendor\*)(pObj);

TModbus\_CPUMEM cm = getCpuMem(pTag->addrType);

memset(tempBuffer, 0, sizeof(tempBuffer) / sizeof(quint8 ));

mesPi = 0;

//设备地址

tempBuffer[mesPi++] = this->devAddr(pVendorObj);

//功能代码

tempBuffer[mesPi++] = getFuncode(pObj, pTag, RW\_flag);

tmpDataPos = pTag->addrOffset;

if(pVendorObj != NULL && !this->isStartAddrBit(pVendorObj)) {

if(tmpDataPos > 0) {

tmpDataPos -= 1;

}

}

//开始地址

tempBuffer[mesPi++] = tmpDataPos >> 8;

tempBuffer[mesPi++] = tmpDataPos;

//读取个数

tmpUnit = getRegNum(pTag);

//根据读/写方式构造报文

if(RW\_flag == FLAG\_READ) {

//计算返回报文长度

tmpLen = pTag->bufLength + 3; // 3 = 一个设备地址 + 一个功能码 + 一个计数

tempBuffer[mesPi++] = tmpUnit >> 8;

tempBuffer[mesPi++] = tmpUnit;

} else if(RW\_flag == FLAG\_WRITE) {

tmpLen = 6; // 6 = 返回从机地址1， 功能代码1， 起始地址2以及强制线圈数2

byteCount = pTag->bufLength;

if(tempBuffer[1] != 0x06 && tempBuffer[1] != 0x05) { //功能码为10

tempBuffer[mesPi++] = tmpUnit >> 8;

tempBuffer[mesPi++] = tmpUnit;

tempBuffer[mesPi++] = byteCount;

}

if(cm == CM\_3x || cm == CM\_4x) {

switch(pTag->dataType) {

case TYPE\_INT16:

case TYPE\_UINT16: {

modbusChangeData(isAddr8(pObj), !isAddr16(pObj), isAddr32(pObj), isAddr64(pObj), pTag->dataToVendor, 2);

memcpy(&tempBuffer[mesPi], pTag->dataToVendor, 2);

mesPi += 2;

}

break;

case TYPE\_INT32:

case TYPE\_UINT32:

case TYPE\_FLOAT32: {

modbusChangeData(isAddr8(pObj), !isAddr16(pObj), isAddr32(pObj), isAddr64(pObj), pTag->dataToVendor, 4);

memcpy(&tempBuffer[mesPi], pTag->dataToVendor, 4);

mesPi += 4;

}

break;

case TYPE\_FLOAT64: {

modbusChangeData(isAddr8(pObj), !isAddr16(pObj), isAddr32(pObj), isAddr64(pObj), pTag->dataToVendor, 8);

memcpy(&tempBuffer[mesPi], pTag->dataToVendor, 8);

mesPi += 8;

}

break;

case TYPE\_ASCII2CHAR: {

modbusChangeData(isAddr8(pObj), !isAddr16(pObj), isAddr32(pObj), isAddr64(pObj), pTag->dataToVendor, 2);

memcpy(&tempBuffer[mesPi], pTag->dataToVendor, 2);

mesPi += 2;

}

break;

default: {

} break;

}

} else {

switch(pTag->dataType) {

case TYPE\_BOOL:

case TYPE\_INT8:

case TYPE\_UINT8: {

memcpy(&tempBuffer[mesPi], pTag->dataToVendor, 1);

mesPi += 1;

}

break;

case TYPE\_INT16:

case TYPE\_UINT16: {

RecoverSelfData(pTag->dataToVendor, 2);

memcpy(&tempBuffer[mesPi], pTag->dataToVendor, 2);

mesPi += 2;

}

break;

case TYPE\_INT32:

case TYPE\_UINT32:

case TYPE\_FLOAT32: {

RecoverSelfData(pTag->dataToVendor, 4);

memcpy(&tempBuffer[mesPi], pTag->dataToVendor, 4);

mesPi += 4;

}

break;

default: {

} break;

}

if(pTag->bufLength <= 1) {

// 增加功能码为05写BOOL的操作

if(tempBuffer[1] == 0x05) {

if(tempBuffer[4] == 0x01) {

tempBuffer[4] = 0xFF;

} else {

tempBuffer[4] = 0x00;

}

tempBuffer[5] = 0x00;

mesPi++;

}

}

}

}

quint16 tmpCRC16 = crc16(tempBuffer, mesPi);

tempBuffer[mesPi++] = tmpCRC16 >> 8;

tempBuffer[mesPi++] = tmpCRC16;

memcpy(pSendData, tempBuffer, mesPi);

\*retVarLen = tmpLen + 2;

return mesPi;

}

quint16 ModbusRTUImpl::**crc16**(quint8 \*pbuf, qint32 len)

{

quint8 uchCRCHi = 0xFF;

quint8 uchCRCLo = 0xFF;

qint32 uIndex = 0;

while (len--) {

uIndex = uchCRCHi ^ \*pbuf++;

uchCRCHi = uchCRCLo ^ auchCRCHi[uIndex];

uchCRCLo = auchCRCLo[uIndex];

}

return (uchCRCHi << 8 | uchCRCLo);

}

bool ModbusRTUImpl::**messageCheck**(quint8 \*inBuf, qint16 bufLen)

{

quint16 revCRC16 = 0, calCRC16 = 0;

calCRC16 = crc16(inBuf, bufLen - 2);

memcpy(&revCRC16, inBuf + bufLen - 2, 2);

RecoverSelfData((quint8 \*)&revCRC16, 2);

if(calCRC16 == revCRC16) {

return true;

}

return false;

}

/\*\*

\* @brief ModbusRTUImpl::isCanWrite

\* @details 判断该变量定义属性是否可以写

\* @param pObj 设备描述对象

\* @param pTag 设备变量

\* @return false-不可写，true-可写

\*/

bool ModbusRTUImpl::***isCanWrite***(void\* pObj, RunTimeTag\* pTag)

{

(void)pObj;

if(getCpuMem(pTag->addrType) == CM\_1x) {

return false;

} else if(getCpuMem(pTag->addrType) == CM\_3x) {

return false;

}

return true;

}

/\*\*

\* @brief ModbusRTUImpl::writeData

\* @details 写一个变量值到plc设备

\* @param pObj 设备描述对象

\* @param pTag 设备变量

\* @return 0-失败,1-成功

\*/

int ModbusRTUImpl::***writeData***(void\* pObj, RunTimeTag\* pTag)

{

(void)pObj;

quint16 msgLen = 0, revLen = 0;

Vendor\* pVendorObj = (Vendor\*)(pObj);

memset(pVendorObj->writeBuf, 0, sizeof(pVendorObj->writeBuf) / sizeof(quint8));

memset(pVendorObj->readBuf, 0, sizeof(pVendorObj->readBuf) / sizeof(quint8));

msgLen = *makeMessagePackage*(pVendorObj->writeBuf, pObj, pTag, FLAG\_WRITE, &revLen);

qDebug() << pVendorObj->m\_pVendorPrivateObj->m\_sDeviceName << ", Modbus Tx: " << hexToString((char \*)pVendorObj->writeBuf, msgLen);

if(getPort() != NULL) {

getPort()->*write*(pVendorObj->writeBuf, msgLen, pVendorObj->m\_pVendorPrivateObj->m\_iCommTimeout);

}

int resultlen = 0;

if(getPort() != NULL) {

resultlen = getPort()->*read*(pVendorObj->readBuf, revLen, pVendorObj->m\_pVendorPrivateObj->m\_iCommTimeout);

qDebug() << pVendorObj->m\_pVendorPrivateObj->m\_sDeviceName << ", Modbus Rx: " << hexToString((char \*)pVendorObj->readBuf, revLen);

}

if(resultlen == revLen && messageCheck(pVendorObj->readBuf, resultlen)) {

return 1;

}

return 0;

}

/\*\*

\* @brief ModbusRTUImpl::isCanRead

\* @details 判断该变量定义属性是否可以读

\* @param pObj 设备描述对象

\* @param pTag 设备变量

\* @return false-不可读，true-可读

\*/

bool ModbusRTUImpl::***isCanRead***(void\* pObj, RunTimeTag\* pTag)

{

(void)pObj;

Q\_UNUSED(pTag)

return true;

}

/\*\*

\* @brief ModbusRTUImpl::readData

\* @details 从plc设备读一个变量

\* @param pObj 设备描述对象

\* @param pTag 设备变量

\* @return 0-失败,1-成功

\*/

int ModbusRTUImpl::***readData***(void\* pObj, RunTimeTag\* pTag)

{

(void)pObj;

quint16 retSize = 0, msgLen = 0, revLen = 0;

qint16 i = 0, j = 0;

TModbus\_CPUMEM cm = getCpuMem(pTag->addrType);

Vendor\* pVendorObj = (Vendor\*)(pObj);

memset(pVendorObj->writeBuf, 0, sizeof(pVendorObj->writeBuf) / sizeof(quint8));

memset(pVendorObj->readBuf, 0, sizeof(pVendorObj->readBuf) / sizeof(quint8));

msgLen = *makeMessagePackage*(pVendorObj->writeBuf, pObj, pTag, FLAG\_READ, &revLen);

qDebug() << pVendorObj->m\_pVendorPrivateObj->m\_sDeviceName << ", Modbus Tx: " << hexToString((char \*)pVendorObj->writeBuf, msgLen);

if(getPort() != NULL) {

getPort()->*write*(pVendorObj->writeBuf, msgLen, pVendorObj->m\_pVendorPrivateObj->m\_iCommTimeout);

}

int resultlen = 0;

if(cm == CM\_0x || cm == CM\_1x) {

if(getPort() != NULL) {

resultlen = getPort()->*read*(pVendorObj->readBuf, 3, pVendorObj->m\_pVendorPrivateObj->m\_iCommTimeout);

}

if(resultlen != 3) {

return -2;

}

if(getPort() != NULL) {

resultlen = getPort()->*read*(&pVendorObj->readBuf[3], pVendorObj->readBuf[2] + 2, pVendorObj->m\_pVendorPrivateObj->m\_iCommTimeout);

}

if(resultlen != pVendorObj->readBuf[2] + 2) {

return -2;

}

if(pTag->dataType == TYPE\_BOOL) {

if(pVendorObj->readBuf[2] > 1) {

revLen = revLen + pVendorObj->readBuf[2] - 1;

}

}

} else {

if(getPort() != NULL) {

resultlen = getPort()->*read*(pVendorObj->readBuf, revLen, pVendorObj->m\_pVendorPrivateObj->m\_iCommTimeout);

}

qDebug() << pVendorObj->m\_pVendorPrivateObj->m\_sDeviceName << ", Modbus Rx: " << hexToString((char \*)pVendorObj->readBuf, revLen);

if(resultlen != revLen) {

return -2;

}

}

if(!messageCheck(pVendorObj->readBuf, revLen)) {

return 0;

}

memset(tempBuffer, 0, sizeof(tempBuffer) / sizeof(quint8 ));

// 返回数据的处理

if(pTag->dataType == TYPE\_BOOL) {

retSize = 1;

uint8ToBytes(pVendorObj->readBuf[3]&0x01, pTag->dataFromVendor);

} else if(pTag->dataType == TYPE\_INT16 || pTag->dataType == TYPE\_UINT16) {

pTag->updateVendorData(&pVendorObj->readBuf[3], 2);

if(cm == CM\_3x || cm == CM\_4x) {

modbusChangeData(isAddr8(pObj), !isAddr16(pObj), isAddr32(pObj), isAddr64(pObj), pTag->dataFromVendor, 2);

}

} else if(pTag->dataType == TYPE\_UINT32 || pTag->dataType == TYPE\_INT32 ||

pTag->dataType == TYPE\_FLOAT32) {

pTag->updateVendorData(&pVendorObj->readBuf[3], 4);

if(cm == CM\_3x || cm == CM\_4x) {

modbusChangeData(isAddr8(pObj), !isAddr16(pObj), isAddr32(pObj), isAddr64(pObj), pTag->dataFromVendor, 4);

}

} else if(pTag->dataType == TYPE\_FLOAT64) {

pTag->updateVendorData(&pVendorObj->readBuf[3], 8);

if(cm == CM\_3x || cm == CM\_4x) {

modbusChangeData(isAddr8(pObj), !isAddr16(pObj), isAddr32(pObj), isAddr64(pObj), pTag->dataFromVendor, 8);

}

} else if(pTag->dataType == TYPE\_UINT8 || pTag->dataType == TYPE\_INT8) {

retSize = pVendorObj->readBuf[2];

if(getFuncode(pObj, pTag, FLAG\_READ) == 0x01 || getFuncode(pObj, pTag, FLAG\_READ) == 0x02) {

j = retSize - 1;

for(i = 0; i < retSize; i++) {

\*(tempBuffer + (j--)) = pVendorObj->readBuf[3 + i];

}

pTag->updateVendorData(tempBuffer, retSize);

} else {

j = retSize / 2 - 1;

for(i = 0; i < retSize; i++, j--) {

// 8位逆序

if(isAddr8(pObj)) {

\*(tempBuffer + 2 \* j) = byteSwitchHigh4bitLow4bit(pVendorObj->readBuf[3 + i]);

i++;

\*(tempBuffer + 2 \* j + 1) = byteSwitchHigh4bitLow4bit(pVendorObj->readBuf[3 + i]);

} else {

\*(tempBuffer + 2 \* j) = pVendorObj->readBuf[3 + i];

i++;

\*(tempBuffer + 2 \* j + 1) = pVendorObj->readBuf[3 + i];

}

}

pTag->updateVendorData(tempBuffer, retSize);

}

} else if(pTag->dataType == TYPE\_BYTES) {

retSize = pVendorObj->readBuf[2];

if(getFuncode(pObj, pTag, FLAG\_READ) == 0x01 || getFuncode(pObj, pTag, FLAG\_READ) == 0x02) {

j = 0;

for(i = 0; i < retSize; i++) {

\*(tempBuffer + (j++)) = pVendorObj->readBuf[3 + i];

}

} else {

j = 0;

for(i = 0; i < retSize; i++, j++) {

\*(tempBuffer + 2 \* j + 1) = pVendorObj->readBuf[3 + i];

i++;

\*(tempBuffer + 2 \* j) = pVendorObj->readBuf[3 + i];

}

}

pTag->updateVendorData(tempBuffer, retSize);

}

return 1;

}

## 添加文件至ModbusRTU.pro

打开{文件HmiRunTime.pro所在目录} \HmiRunTime\Vendors\ModbusRTU\ModbusRTU.pro文件，把ModbusRTU实现文件分别加入至ModbusRTU.pro文件变量SOURCES，HEADERS，具体请阅读ModbusRTU.pro文件。

## 编译ModbusRTU工程