**ProjectManager新增通讯设备**

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 新建ModbusRTU插件工程 3](#_Toc13632)

[1.1 新建插件工程 3](#_Toc27340)

[1.2 ModbusRTU\_dependencies.pri文件说明 3](#_Toc8254)

[1.3 ModbusRTU.pro文件说明 3](#_Toc25299)

[1.4 插件接口说明 4](#_Toc713)

[1.5 实现ModbusRTU插件 6](#_Toc31355)

[1.6 编译插件工程 25](#_Toc8344)

[2 ProjectManager相关配置 26](#_Toc2776)

[2.1 配置文件 26](#_Toc27077)

[2.2 ModbusRTU配置 28](#_Toc6566)

[3 ProjectManager插件相关源码和UI 29](#_Toc29437)

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

# 1 新建ModbusRTU插件工程

## 新建插件工程

在{文件HmiFuncDesigner.pro所在目录} \Devices新建ModbusRTU目录, 在ModbusRTU目录新建如下文件:

ModbusRTU.pro

ModbusRTU.cpp,

ModbusRTU.h

ModbusRTU.json

ModbusRTU\_dependencies.pri

文件内容可能有改动，详细内容请阅读源码文件！！

## ModbusRTU\_dependencies.pri文件说明

文件内容：

QTC\_PLUGIN\_NAME = ModbusRTU

QTC\_LIB\_DEPENDS +=

QTC\_PLUGIN\_DEPENDS +=

其中变量“QTC\_PLUGIN\_NAME”的值为生成的插件名称。最后生成的插件位于目录：HmiFuncDesignerBin\deviceplugins

## ModbusRTU.pro文件说明

文件内容：

include(../../HmiFuncDesignerDevicePlugin.pri)

SOURCES += ModbusRTU.cpp

HEADERS += ModbusRTU.h

DISTFILES += ModbusRTU.json

其中文件“HmiFuncDesignerDevicePlugin.pri”指定了工程模板，配置，依赖等。

## 插件接口说明

插件接口{文件HmiFuncDesigner.pro所在目录} \Devices\IDevicePlugin\IDevicePlugin.h

文件内容：

#ifndef IDEVICEPLUGIN\_H

#define IDEVICEPLUGIN\_H

#include <QStringList>

#include <QPair>

#include <QVector>

#include <QMap>

#include <QXmlStreamWriter>

#include <QXmlStreamReader>

#include <QXmlStreamAttribute>

/\*

\* 注意本类不要派生自QObject

\*/

class IDevicePlugin

{

public:

virtual ~IDevicePlugin() {}

// 获取设备默认属性

virtual void getDefaultDeviceProperty(QVector<QPair<QString, QString>>& properties) = 0;

// 获取设备默认属性数据类型

virtual void getDefaultDevicePropertyDataType(QVector<QPair<QString, QString>>& properties\_type) = 0;

// 保存属性

virtual void writeProperties(QString &szProperties, QVector<QPair<QString, QString>>& properties) = 0;

// 加载属性

virtual void readProperties(QString &szProperties, QVector<QPair<QString, QString>>& properties) = 0;

// 设置设备属性

virtual void setDeviceProperty(QVector<QPair<QString, QString>>& properties) = 0;

// 生成块读变量

virtual bool buildBlockReadTags(const QString &xmlDevTags, const QString &properties, QString &xmlDevBlockReadTags, QVector<QPair<QString, QString>>& idToBlockId) = 0;

// 获取设备描述信息

virtual QString getDeviceDescInfo() = 0;

QString getValue2ByValue1(const QString &szVal1, QVector<QPair<QString, QString>>& properties) {

for (int i = 0; i < properties.size(); ++i) {

QPair<QString, QString> pair = properties[i];

if (pair.first == szVal1 && pair.second != "")

return pair.second;

}

// 获取默认属性值

QVector<QPair<QString, QString>> defaultProperties;

this->getDefaultDeviceProperty(defaultProperties);

for (int i = 0; i < defaultProperties.size(); ++i) {

QPair<QString, QString> pair = defaultProperties[i];

if (pair.first == szVal1)

return pair.second;

}

return "";

}

int tagLength(const QString &type) {

if(type == "bool" || type == "int8" || type == "uint8" || type == "bcd8"){

return 1;

} else if(type == "int16" || type == "uint16" || type == "bcd16") {

return 2;

} else if(type == "int32" || type == "uint32" || type == "float32" || type == "bcd32") {

return 4;

} else if(type == "double" || type == "int64" || type == "uint64") {

return 8;

}

return 0;

}

};

QT\_BEGIN\_NAMESPACE

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

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

QT\_END\_NAMESPACE

#endif // IDEVICEPLUGIN\_H

## 实现ModbusRTU插件

新建ModbusRTU实现类，继承类QObject ，类IDevicePlugin实现插件接口。

文件ModbusRTU.h内容如下：

#ifndef MODBUSRTU\_H

#define MODBUSRTU\_H

#include <QObject>

#include "../IDevicePlugin/IDevicePlugin.h"

typedef struct tagTagInfo {

int id;

quint32 offset;

quint32 length;

bool use;

} TTagInfo;

class ModbusRTU : public QObject, IDevicePlugin

{

Q\_OBJECT

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

Q\_INTERFACES(IDevicePlugin)

public:

ModbusRTU();

// 获取设备默认属性

void getDefaultDeviceProperty(QVector<QPair<QString, QString>>& properties) Q\_DECL\_OVERRIDE;

// 获取设备默认属性数据类型

void getDefaultDevicePropertyDataType(QVector<QPair<QString, QString>>& properties\_type) Q\_DECL\_OVERRIDE;

// 保存属性

void writeProperties(QString &szProperties, QVector<QPair<QString, QString>>& properties) Q\_DECL\_OVERRIDE;

// 加载属性

void readProperties(QString &szProperties, QVector<QPair<QString, QString>>& properties) Q\_DECL\_OVERRIDE;

// 设置设备属性

void setDeviceProperty(QVector<QPair<QString, QString>>& properties) Q\_DECL\_OVERRIDE;

// 生成块读变量

bool buildBlockReadTags(const QString &xmlDevTags, const QString &properties, QString &xmlDevBlockReadTags, QVector<QPair<QString, QString>>& idToBlockId) Q\_DECL\_OVERRIDE;

// 获取设备描述信息

QString getDeviceDescInfo() Q\_DECL\_OVERRIDE;

private:

void loadInfos(QXmlStreamReader \*r, QMap<QString, QVector<TTagInfo>> &infos, QString &dev);

private:

QVector<QPair<QString, QString>> m\_properties; // 插件私有属性

bool m\_bStartAddrBit0 = true; // 内存地址起始位为0

};

#endif // MODBUSRTU\_H

文件ModbusRTU.cpp内容如下：

#include "ModbusRTU.h"

#include <QDebug>

#define COIL\_STATUS tr("线圈状态")

#define DISCRETE\_STATUS tr("离散量状态")

#define INPUT\_REGISTER tr("输入寄存器")

#define HOLDING\_REGISTER tr("保持寄存器")

ModbusRTU::ModbusRTU() {

}

///

/// \brief ModbusRTU::getDefaultDeviceProperty

/// \details 获取设备默认属性

/// \param properties

///

void ModbusRTU::getDefaultDeviceProperty(QVector<QPair<QString, QString>>& properties) {

properties.clear();

properties.append(qMakePair(tr("设备ID"), QString("1")));

properties.append(qMakePair(tr("位组包最大寄存器个数"), QString("100")));

properties.append(qMakePair(tr("字组包最大寄存器个数"), QString("20")));

properties.append(qMakePair(tr("失败重试次数"), QString("2")));

properties.append(qMakePair(tr("通讯超时时间(s)"), QString("1")));

properties.append(qMakePair(tr("通讯间隔时间(ms)"), QString("200")));

properties.append(qMakePair(tr("尝试恢复间隔(s)"), QString("2")));

properties.append(qMakePair(tr("内存地址起始位为0"), QString("true")));

properties.append(qMakePair(tr("写线圈功能码为15"), QString("false")));

properties.append(qMakePair(tr("写寄存器功能码为16"), QString("false")));

properties.append(qMakePair(tr("8位逆序"), QString("false")));

properties.append(qMakePair(tr("16位低字节在前高字节在后"), QString("false")));

properties.append(qMakePair(tr("32位低字节在前高字节在后"), QString("false")));

properties.append(qMakePair(tr("64位低字节在前高字节在后"), QString("false")));

}

///

/// \brief ModbusRTU::getDefaultDevicePropertyDataType

/// \details 获取设备默认属性数据类型

/// \param properties\_type

///

void ModbusRTU::getDefaultDevicePropertyDataType(QVector<QPair<QString, QString>>& properties\_type) {

properties\_type.clear();

properties\_type.append(qMakePair(tr("设备ID"), QString("int")));

properties\_type.append(qMakePair(tr("位组包最大寄存器个数"), QString("int")));

properties\_type.append(qMakePair(tr("字组包最大寄存器个数"), QString("int")));

properties\_type.append(qMakePair(tr("失败重试次数"), QString("int")));

properties\_type.append(qMakePair(tr("通讯超时时间(s)"), QString("int")));

properties\_type.append(qMakePair(tr("通讯间隔时间(ms)"), QString("int")));

properties\_type.append(qMakePair(tr("尝试恢复间隔(s)"), QString("int")));

properties\_type.append(qMakePair(tr("内存地址起始位为0"), QString("bool")));

properties\_type.append(qMakePair(tr("写线圈功能码为15"), QString("bool")));

properties\_type.append(qMakePair(tr("写寄存器功能码为16"), QString("bool")));

properties\_type.append(qMakePair(tr("8位逆序"), QString("bool")));

properties\_type.append(qMakePair(tr("16位低字节在前高字节在后"), QString("bool")));

properties\_type.append(qMakePair(tr("32位低字节在前高字节在后"), QString("bool")));

properties\_type.append(qMakePair(tr("64位低字节在前高字节在后"), QString("bool")));

}

///

/// \brief ModbusRTU::writeProperties

/// \details 保存属性

/// \param szProperties 属性字符串

/// \param properties 属性

///

void ModbusRTU::writeProperties(QString &szProperties, QVector<QPair<QString, QString>>& properties) {

QStringList szListProperties;

szListProperties << QString("%1=%2").arg("id").arg(getValue2ByValue1(tr("设备ID"), properties));

szListProperties << QString("%1=%2").arg("bitMaxRegPacket").arg(getValue2ByValue1(tr("位组包最大寄存器个数"), properties));

szListProperties << QString("%1=%2").arg("wordMaxRegPacket").arg(getValue2ByValue1(tr("字组包最大寄存器个数"), properties));

szListProperties << QString("%1=%2").arg("commFailRetryTimes").arg(getValue2ByValue1(tr("失败重试次数"), properties));

szListProperties << QString("%1=%2").arg("commTimeout").arg(getValue2ByValue1(tr("通讯超时时间(s)"), properties));

szListProperties << QString("%1=%2").arg("commIntervalTime").arg(getValue2ByValue1(tr("通讯间隔时间(ms)"), properties));

szListProperties << QString("%1=%2").arg("commResumeTime").arg(getValue2ByValue1(tr("尝试恢复间隔(s)"), properties));

szListProperties << QString("%1=%2").arg("startAddrBit").arg(getValue2ByValue1(tr("内存地址起始位为0"), properties));

szListProperties << QString("%1=%2").arg("writeCoilFn").arg(getValue2ByValue1(tr("写线圈功能码为15"), properties));

szListProperties << QString("%1=%2").arg("writeRegFn").arg(getValue2ByValue1(tr("写寄存器功能码为16"), properties));

szListProperties << QString("%1=%2").arg("addr8").arg(getValue2ByValue1(tr("8位逆序"), properties));

szListProperties << QString("%1=%2").arg("addr16").arg(getValue2ByValue1(tr("16位低字节在前高字节在后"), properties));

szListProperties << QString("%1=%2").arg("addr32").arg(getValue2ByValue1(tr("32位低字节在前高字节在后"), properties));

szListProperties << QString("%1=%2").arg("addr64").arg(getValue2ByValue1(tr("64位低字节在前高字节在后"), properties));

szProperties = szListProperties.join("|");

}

///

/// \brief ModbusRTU::readProperties

/// \details 加载属性

/// \param szProperties 属性字符串

/// \param properties 属性

///

void ModbusRTU::readProperties(QString &szProperties, QVector<QPair<QString, QString>>& properties) {

properties.clear();

QStringList szListProperties = szProperties.split("|");

foreach(QString szKeyValue, szListProperties) {

if (szKeyValue.startsWith("id=")) {

QString val = szKeyValue.replace("id=", "");

if(val == "") val = "1";

properties.append(qMakePair(tr("设备ID"), val));

}

if (szKeyValue.startsWith("bitMaxRegPacket=")) {

QString val = szKeyValue.replace("bitMaxRegPacket=", "");

if(val == "") val = "100";

properties.append(qMakePair(tr("位组包最大寄存器个数"), val));

}

if (szKeyValue.startsWith("wordMaxRegPacket=")) {

QString val = szKeyValue.replace("wordMaxRegPacket=", "");

if(val == "") val = "20";

properties.append(qMakePair(tr("字组包最大寄存器个数"), val));

}

if (szKeyValue.startsWith("commFailRetryTimes=")) {

QString val = szKeyValue.replace("commFailRetryTimes=", "");

if(val == "") val = "2";

properties.append(qMakePair(tr("失败重试次数"), val));

}

if (szKeyValue.startsWith("commTimeout=")) {

QString val = szKeyValue.replace("commTimeout=", "");

if(val == "") val = "1";

properties.append(qMakePair(tr("通讯超时时间(s)"), val));

}

if (szKeyValue.startsWith("commIntervalTime=")) {

QString val = szKeyValue.replace("commIntervalTime=", "");

if(val == "") val = "200";

properties.append(qMakePair(tr("通讯间隔时间(ms)"), val));

}

if (szKeyValue.startsWith("commResumeTime=")) {

QString val = szKeyValue.replace("commResumeTime=", "");

if(val == "") val = "2";

properties.append(qMakePair(tr("尝试恢复间隔(s)"), val));

}

if (szKeyValue.startsWith("startAddrBit=")) {

QString val = szKeyValue.replace("startAddrBit=", "");

if(val == "") val = "true";

properties.append(qMakePair(tr("内存地址起始位为0"), val));

}

if (szKeyValue.startsWith("writeCoilFn=")) {

QString val = szKeyValue.replace("writeCoilFn=", "");

if(val == "") val = "false";

properties.append(qMakePair(tr("写线圈功能码为15"), val));

}

if (szKeyValue.startsWith("writeRegFn=")) {

QString val = szKeyValue.replace("writeRegFn=", "");

if(val == "") val = "false";

properties.append(qMakePair(tr("写寄存器功能码为16"), val));

}

if (szKeyValue.startsWith("addr8=")) {

QString val = szKeyValue.replace("addr8=", "");

if(val == "") val = "false";

properties.append(qMakePair(tr("8位逆序"), val));

}

if (szKeyValue.startsWith("addr16=")) {

QString val = szKeyValue.replace("addr16=", "");

if(val == "") val = "false";

properties.append(qMakePair(tr("16位低字节在前高字节在后"), val));

}

if (szKeyValue.startsWith("addr32=")) {

QString val = szKeyValue.replace("addr32=", "");

if(val == "") val = "false";

properties.append(qMakePair(tr("32位低字节在前高字节在后"), val));

}

if (szKeyValue.startsWith("addr64=")) {

QString val = szKeyValue.replace("addr64=", "");

if(val == "") val = "false";

properties.append(qMakePair(tr("64位低字节在前高字节在后"), val));

}

}

}

///

/// \brief ModbusRTU::setDeviceProperty

/// \details 设置设备属性

/// \param properties

///

void ModbusRTU::setDeviceProperty(QVector<QPair<QString, QString>>& properties) {

m\_properties.clear();

m\_properties.append(properties);

QString szVal = getValue2ByValue1(tr("内存地址起始位为0"), m\_properties);

m\_bStartAddrBit0 = (szVal.toLower() == "true") ? true : false;

}

///

/// \brief ModbusRTU::getDeviceDescInfo

/// @details 获取设备描述信息

/// \return 设备描述信息

///

QString ModbusRTU::getDeviceDescInfo() {

#if 0

<?xml version="1.0"/>

<Device Name="ModbusRTU" AllDataType="bool|int16|uint16|int32|uint32|float32|double|bcd16|bcd32" SupportProtocol="ModbusRTU">

<RegAreas>

<RegArea Name="线圈状态" Alias="0x" Min="0x0000" Max="0xFFFF" DataType="bool" SubArea=""/>

<RegArea Name="离散量状态" Alias="1x" Min="0x0000" Max="0xFFFF" DataType="bool" SubArea=""/>

<RegArea Name="输入寄存器" Alias="3x" Min="0x0000" Max="0xFFFF" DataType="int16|uint16|int32|uint32|float32|double|bcd16|bcd32" SubArea=""/>

<RegArea Name="保持寄存器" Alias="4x" Min="0x0000" Max="0xFFFF" DataType="int16|uint16|int32|uint32|float32|double|bcd16|bcd32" SubArea=""/>

</RegAreas>

</Device>

#endif

QString szDeviceDescInfo;

QStringList szListDataType;

QStringList szListSubArea;

QXmlStreamWriter writer(&szDeviceDescInfo);

writer.setAutoFormatting(true);

writer.writeStartDocument();

writer.writeStartElement("Device"); // <Device>

writer.writeAttribute("Name", "ModbusRTU");

szListDataType.clear();

szListDataType << tr("bool")

<< tr("int16")

<< tr("uint16")

<< tr("int32")

<< tr("uint32")

<< tr("float32")

<< tr("double")

<< tr("bcd16")

<< tr("bcd32");

writer.writeAttribute("AllDataType", szListDataType.join("|"));

// 设备支持的所有协议

QStringList szListSupportProtocol;

szListSupportProtocol << "ModbusRTU";

writer.writeAttribute("SupportProtocol", szListSupportProtocol.join("|"));

// 设备支持的所有寄存器区

writer.writeStartElement("RegAreas"); // <RegAreas>

writer.writeStartElement("RegArea"); // <RegArea>

writer.writeAttribute("Name", COIL\_STATUS);

writer.writeAttribute("Alias", "0x");

writer.writeAttribute("Min", "0x0000");

writer.writeAttribute("Max", "0xFFFF");

szListDataType.clear();

szListDataType << tr("bool");

writer.writeAttribute("DataType", szListDataType.join("|"));

szListSubArea.clear();

writer.writeAttribute("SubArea", szListSubArea.join("|"));

writer.writeEndElement(); // <RegArea/>

writer.writeStartElement("RegArea"); // <RegArea>

writer.writeAttribute("Name", DISCRETE\_STATUS);

writer.writeAttribute("Alias", "1x");

writer.writeAttribute("Min", "0x0000");

writer.writeAttribute("Max", "0xFFFF");

szListDataType.clear();

szListDataType << tr("bool");

writer.writeAttribute("DataType", szListDataType.join("|"));

szListSubArea.clear();

writer.writeAttribute("SubArea", szListSubArea.join("|"));

writer.writeEndElement(); // <RegArea/>

writer.writeStartElement("RegArea"); // <RegArea>

writer.writeAttribute("Name", INPUT\_REGISTER);

writer.writeAttribute("Alias", "3x");

writer.writeAttribute("Min", "0x0000");

writer.writeAttribute("Max", "0xFFFF");

szListDataType.clear();

szListDataType << tr("int16")

<< tr("uint16")

<< tr("int32")

<< tr("uint32")

<< tr("float32")

<< tr("double")

<< tr("bcd16")

<< tr("bcd32");

writer.writeAttribute("DataType", szListDataType.join("|"));

szListSubArea.clear();

writer.writeAttribute("SubArea", szListSubArea.join("|"));

writer.writeEndElement(); // <RegArea/>

writer.writeStartElement("RegArea"); // <RegArea>

writer.writeAttribute("Name", HOLDING\_REGISTER);

writer.writeAttribute("Alias", "4x");

writer.writeAttribute("Min", "0x0000");

writer.writeAttribute("Max", "0xFFFF");

szListDataType.clear();

szListDataType << tr("int16")

<< tr("uint16")

<< tr("int32")

<< tr("uint32")

<< tr("float32")

<< tr("double")

<< tr("bcd16")

<< tr("bcd32");

writer.writeAttribute("DataType", szListDataType.join("|"));

szListSubArea.clear();

writer.writeAttribute("SubArea", szListSubArea.join("|"));

writer.writeEndElement(); // <RegArea/>

writer.writeEndElement(); // <RegAreas/>

writer.writeEndElement(); // <Device/>

writer.writeEndDocument();

return szDeviceDescInfo;

}

void ModbusRTU::loadInfos(QXmlStreamReader \*r, QMap<QString, QVector<TTagInfo>> &infos, QString &dev)

{

QString addr;

TTagInfo info;

if(r->name().toString() == "tag") {

foreach(QXmlStreamAttribute attr, r->attributes()) {

QString attrName = attr.name().toString();

if(attrName == "addr"){

addr = attr.value().toString();

} else if(attrName == "id"){

info.id = attr.value().toInt();

} else if(attrName == "dev"){

dev = attr.value().toString();

} else if(attrName == "offset"){

info.offset = attr.value().toInt();

} else if(attrName == "type"){

info.length = tagLength(attr.value().toString());

}

}

if(infos.count(addr) > 0) {

QVector<TTagInfo> &vecInfo = infos[addr];

vecInfo.append(info);

} else {

QVector<TTagInfo> vecInfo;

vecInfo.append(info);

infos[addr] = vecInfo;

}

}

}

/\*\*

\* @brief ModbusRTU::buildBlockReadTags 生成块读变量

\* @param xmlDevTags 同一设备的所有寄存器变量

\* @param xmlDevBlockReadTags 同一设备的所有块读变量

\* @param idToBlockId 变量ID对应的块读变量ID

\* @return true-成功, false-失败

\*/

bool ModbusRTU::buildBlockReadTags(const QString &xmlDevTags, const QString &properties, QString &xmlDevBlockReadTags, QVector<QPair<QString, QString>>& idToBlockId)

{

#if 0

xmlDevTags -->

<tags>

<tag addr="4x" addr2="" dev="ModbusRTU" group="" id="7" name="4x1" offset="0" offset2="0" remark="" type="uint16" unit="" writeable="1" blockReadId="60001"/>

<tag addr="4x" addr2="" dev="ModbusRTU" group="" id="8" name="4x2" offset="1" offset2="0" remark="" type="uint16" unit="" writeable="1" blockReadId="60001"/>

<tag addr="0x" addr2="" dev="ModbusRTU" group="" id="2" name="0x00" offset="0" offset2="0" remark="" type="bool" unit="" writeable="1" blockReadId="60002"/>

</tags>

xmlDevBlockReadTags -->

<block\_tags>

<tag addr="4x" addr2="" dev="ModbusRTU" group="" id="60001" name="4x1" offset="0" offset2="0" remark="" type="4:reg" unit="" writeable="0" blockReadId=""/>

<tag addr="0x" addr2="" dev="ModbusRTU" group="" id="60002" name="0x00" offset="0" offset2="0" remark="" type="1:reg" unit="" writeable="0" blockReadId=""/>

</block\_tags>

#endif

int bitMaxRegPacket = 0;

int wordMaxRegPacket = 0;

QStringList szListProperties = properties.split("|");

foreach(QString szKeyValue, szListProperties) {

if (szKeyValue.startsWith("bitMaxRegPacket=")) {

QString val = szKeyValue.replace("bitMaxRegPacket=", "").trimmed();

if(!val.isEmpty()) {

bitMaxRegPacket = val.toInt();

}

}

if (szKeyValue.startsWith("wordMaxRegPacket=")) {

QString val = szKeyValue.replace("wordMaxRegPacket=", "").trimmed();

if(!val.isEmpty()) {

wordMaxRegPacket = val.toInt();

}

}

}

// 不需要块读

if(bitMaxRegPacket <= 0 && wordMaxRegPacket <= 0) {

return false;

}

QString dev = "";

QMap<QString, QVector<TTagInfo>> mapInfos;

// 解析设备变量节点

QXmlStreamReader r(xmlDevTags);

while(!r.atEnd() && !r.hasError()) {

if(r.readNext() == QXmlStreamReader::StartElement) {

if(r.name() == "tag") {

loadInfos(&r, mapInfos, dev);

}

}

}

#if 0

{

qDebug() << "dev: " << dev;

QList<QString> memInfo = mapInfos.keys();

qDebug() << "memInfo: " << memInfo;

foreach (QString info, memInfo) {

QVector<TTagInfo> &vecInfo = mapInfos[info];

foreach(TTagInfo tagInfo, vecInfo) {

qDebug() << "tagInfo: " << tagInfo.id << tagInfo.offset << tagInfo.length;

}

}

}

#endif

///////////////////////////////////////////////////////////////////////////

/// 生成打包变量

///

quint16 min0xAddr = 0xffff;

quint16 max0xAddr = 0;

quint16 var0xCnt = 0;

quint16 min1xAddr = 0xffff;

quint16 max1xAddr = 0;

quint16 var1xCnt = 0;

quint16 min3xAddr = 0xffff;

quint16 max3xAddr = 0;

quint16 var3xCnt = 0;

quint16 min4xAddr = 0xffff;

quint16 max4xAddr = 0;

quint16 var4xCnt = 0;

QList<QString> memInfo = mapInfos.keys();

foreach (QString info, memInfo) {

QVector<TTagInfo> &vecInfo = mapInfos[info];

foreach(TTagInfo tagInfo, vecInfo) {

if(info == "0x") {

if(min0xAddr > tagInfo.offset) {

min0xAddr = tagInfo.offset;

}

if(max0xAddr <= tagInfo.offset + tagInfo.length) {

max0xAddr = tagInfo.offset + tagInfo.length;

}

var0xCnt++;

} else if(info == "1x") {

if(min1xAddr > tagInfo.offset) {

min1xAddr = tagInfo.offset;

}

if(max1xAddr <= tagInfo.offset + tagInfo.length) {

max1xAddr = tagInfo.offset + tagInfo.length;

}

var1xCnt++;

} else if(info == "3x") {

if(min3xAddr > tagInfo.offset) {

min3xAddr = tagInfo.offset;

}

if(max3xAddr <= tagInfo.offset + tagInfo.length / 2 - 1) {

max3xAddr = tagInfo.offset + tagInfo.length / 2 - 1;

}

var3xCnt++;

} else if(info == "4x") {

if(min4xAddr > tagInfo.offset) {

min4xAddr = tagInfo.offset;

}

if(max4xAddr <= tagInfo.offset + tagInfo.length / 2 - 1) {

max4xAddr = tagInfo.offset + tagInfo.length / 2 - 1;

}

var4xCnt++;

}

}

}

#if 0

qDebug() << "0x number: " << var0xCnt << ", min addr: " << min0xAddr << ", max addr: " << max0xAddr;

qDebug() << "1x number: " << var1xCnt << ", min addr: " << min1xAddr << ", max addr: " << max1xAddr;

qDebug() << "3x number: " << var3xCnt << ", min addr: " << min3xAddr << ", max addr: " << max3xAddr;

qDebug() << "4x number: " << var4xCnt << ", min addr: " << min4xAddr << ", max addr: " << max4xAddr;

#endif

QMap<QString, QVector<TTagInfo \*>> mapBlockInfos;

QVector<TTagInfo \*> vecInfo0x;

mapBlockInfos["0x"] = vecInfo0x;

QVector<TTagInfo \*> vecInfo1x;

mapBlockInfos["1x"] = vecInfo1x;

QVector<TTagInfo \*> vecInfo3x;

mapBlockInfos["3x"] = vecInfo3x;

QVector<TTagInfo \*> vecInfo4x;

mapBlockInfos["4x"] = vecInfo4x;

int iNextPackageVarID = 60001;

// 创建0x组包变量

if(var0xCnt > 1) {

int num = (max0xAddr - min0xAddr + 1) / bitMaxRegPacket;

if(((max0xAddr - min0xAddr + 1) % bitMaxRegPacket) > 0) {

num++;

}

//qDebug() << "0x pack variable number: " << num;

for(int j=0; j<num; j++) {

TTagInfo \*pInfoObj = new TTagInfo;

pInfoObj->id = iNextPackageVarID;

pInfoObj->offset = min0xAddr + j \* bitMaxRegPacket;

pInfoObj->length = bitMaxRegPacket;

pInfoObj->use = false;

iNextPackageVarID++;

mapBlockInfos["0x"].append(pInfoObj);

}

}

// 创建1x组包变量

if(var1xCnt > 1) {

int num = (max1xAddr - min1xAddr + 1) / bitMaxRegPacket;

if(((max1xAddr - min1xAddr + 1) % bitMaxRegPacket) > 0) {

num++;

}

//qDebug() << "1x pack variable number: " << num;

for(int j=0; j<num; j++) {

TTagInfo \*pInfoObj = new TTagInfo;

pInfoObj->id = iNextPackageVarID;

pInfoObj->offset = min1xAddr + j \* bitMaxRegPacket;

pInfoObj->length = bitMaxRegPacket;

pInfoObj->use = false;

iNextPackageVarID++;

mapBlockInfos["1x"].append(pInfoObj);

}

}

// 创建3x组包变量

if(var3xCnt > 1) {

int num = (max3xAddr - min3xAddr + 1) / wordMaxRegPacket;

if(((max3xAddr - min3xAddr + 1) % wordMaxRegPacket) > 0) {

num++;

}

//qDebug() << "3x pack variable number: " << num;

for(int j=0; j<num; j++) {

TTagInfo \*pInfoObj = new TTagInfo;

pInfoObj->id = iNextPackageVarID;

pInfoObj->offset = min3xAddr + j \* wordMaxRegPacket;

pInfoObj->length = wordMaxRegPacket;

pInfoObj->use = false;

iNextPackageVarID++;

mapBlockInfos["3x"].append(pInfoObj);

}

}

// 创建4x组包变量

if(var4xCnt > 1) {

int num = (max4xAddr - min4xAddr + 1) / wordMaxRegPacket;

if(((max4xAddr - min4xAddr + 1) % wordMaxRegPacket) > 0) {

num++;

}

//qDebug() << "4x pack variable number: " << num;

for(int j=0; j<num; j++) {

TTagInfo \*pInfoObj = new TTagInfo;

pInfoObj->id = iNextPackageVarID;

pInfoObj->offset = min4xAddr + j \* wordMaxRegPacket;

pInfoObj->length = wordMaxRegPacket;

pInfoObj->use = false;

iNextPackageVarID++;

mapBlockInfos["4x"].append(pInfoObj);

}

}

///////////////////////////////////////////////////////////////////////////

/// 变量关联组包变量

///

QXmlStreamWriter writer(&xmlDevBlockReadTags);

writer.setAutoFormatting(true);

writer.writeStartDocument();

writer.writeStartElement("block\_tags"); // <block\_tags>

QVector<TTagInfo \*> &vecTagInfo0x = mapBlockInfos["0x"];

quint32 dwFindMax0xAddr = 0; // 包内的最大地址

quint32 dwFindMin0xAddr = 0xffff; // 包内的最小地址

foreach(TTagInfo \*pObj, vecTagInfo0x) {

quint32 iMinAddrOffset = pObj->offset;

quint32 iMaxAddrOffset = pObj->length;

dwFindMax0xAddr = iMinAddrOffset;

dwFindMin0xAddr = iMaxAddrOffset;

QList<QString> memInfo = mapInfos.keys();

foreach (QString info, memInfo) {

if(info != "0x") {

continue;

}

QVector<TTagInfo> &vecInfo = mapInfos[info];

foreach(TTagInfo tagInfo, vecInfo) {

if(tagInfo.offset >= iMinAddrOffset && (tagInfo.offset + tagInfo.length) <= iMaxAddrOffset) {

pObj->use = true;

QPair<QString, QString> idPair;

idPair.first = QString::number(tagInfo.id);

idPair.second = QString::number(pObj->id);

idToBlockId.append(idPair);

if((tagInfo.offset + tagInfo.length) > dwFindMax0xAddr) {

dwFindMax0xAddr = tagInfo.offset + tagInfo.length;

}

if(dwFindMin0xAddr > tagInfo.offset) {

dwFindMin0xAddr = tagInfo.offset;

}

}

}

}

writer.writeStartElement("tag"); // <tag>

writer.writeAttribute("addr", "0x");

writer.writeAttribute("addr2", "");

writer.writeAttribute("dev", dev);

writer.writeAttribute("group", "");

writer.writeAttribute("id", QString::number(pObj->id));

writer.writeAttribute("name", QString("0x\_%1").arg(QString::number(pObj->id)));

writer.writeAttribute("offset", QString::number(pObj->offset));

writer.writeAttribute("offset2", "");

writer.writeAttribute("remark", "");

writer.writeAttribute("type", QString("%1:reg").arg(QString::number(dwFindMax0xAddr - dwFindMin0xAddr)));

writer.writeAttribute("unit", "");

writer.writeAttribute("writeable", "0");

writer.writeAttribute("blockReadId", "");

writer.writeEndElement(); // <tag/>

}

QVector<TTagInfo \*> &vecTagInfo1x = mapBlockInfos["1x"];

quint32 dwFindMax1xAddr = 0; // 包内的最大地址

quint32 dwFindMin1xAddr = 0xffff; // 包内的最小地址

foreach(TTagInfo \*pObj, vecTagInfo1x) {

quint32 iMinAddrOffset = pObj->offset;

quint32 iMaxAddrOffset = pObj->length;

dwFindMax1xAddr = iMinAddrOffset;

dwFindMin1xAddr = iMaxAddrOffset;

QList<QString> memInfo = mapInfos.keys();

foreach (QString info, memInfo) {

if(info != "1x") {

continue;

}

QVector<TTagInfo> &vecInfo = mapInfos[info];

foreach(TTagInfo tagInfo, vecInfo) {

if(tagInfo.offset >= iMinAddrOffset && (tagInfo.offset + tagInfo.length) <= iMaxAddrOffset) {

pObj->use = true;

QPair<QString, QString> idPair;

idPair.first = QString::number(tagInfo.id);

idPair.second = QString::number(pObj->id);

idToBlockId.append(idPair);

if((tagInfo.offset + tagInfo.length) > dwFindMax1xAddr) {

dwFindMax1xAddr = tagInfo.offset + tagInfo.length;

}

if(dwFindMin1xAddr > tagInfo.offset) {

dwFindMin1xAddr = tagInfo.offset;

}

}

}

}

writer.writeStartElement("tag"); // <tag>

writer.writeAttribute("addr", "1x");

writer.writeAttribute("addr2", "");

writer.writeAttribute("dev", dev);

writer.writeAttribute("group", "");

writer.writeAttribute("id", QString::number(pObj->id));

writer.writeAttribute("name", QString("1x\_%1").arg(QString::number(pObj->id)));

writer.writeAttribute("offset", QString::number(pObj->offset));

writer.writeAttribute("offset2", "");

writer.writeAttribute("remark", "");

writer.writeAttribute("type", QString("%1:reg").arg(QString::number(dwFindMax1xAddr - dwFindMin1xAddr)));

writer.writeAttribute("unit", "");

writer.writeAttribute("writeable", "0");

writer.writeAttribute("blockReadId", "");

writer.writeEndElement(); // <tag/>

}

QVector<TTagInfo \*> &vecTagInfo3x = mapBlockInfos["3x"];

quint32 dwFindMax3xAddr = 0; // 包内的最大地址

quint32 dwFindMin3xAddr = 0xffff; // 包内的最小地址

foreach(TTagInfo \*pObj, vecTagInfo3x) {

quint32 iMinAddrOffset = pObj->offset;

quint32 iMaxAddrOffset = pObj->length;

dwFindMax3xAddr = iMinAddrOffset;

dwFindMin3xAddr = iMaxAddrOffset;

QList<QString> memInfo = mapInfos.keys();

foreach (QString info, memInfo) {

if(info != "3x") {

continue;

}

QVector<TTagInfo> &vecInfo = mapInfos[info];

foreach(TTagInfo tagInfo, vecInfo) {

if(tagInfo.offset >= iMinAddrOffset && (tagInfo.offset + tagInfo.length) <= iMaxAddrOffset) {

pObj->use = true;

QPair<QString, QString> idPair;

idPair.first = QString::number(tagInfo.id);

idPair.second = QString::number(pObj->id);

idToBlockId.append(idPair);

if((tagInfo.offset + tagInfo.length) > dwFindMax3xAddr) {

dwFindMax3xAddr = tagInfo.offset + tagInfo.length / 2;

}

if(dwFindMin3xAddr > tagInfo.offset) {

dwFindMin3xAddr = tagInfo.offset;

}

}

}

}

writer.writeStartElement("tag"); // <tag>

writer.writeAttribute("addr", "3x");

writer.writeAttribute("addr2", "");

writer.writeAttribute("dev", dev);

writer.writeAttribute("group", "");

writer.writeAttribute("id", QString::number(pObj->id));

writer.writeAttribute("name", QString("3x\_%1").arg(QString::number(pObj->id)));

writer.writeAttribute("offset", QString::number(pObj->offset));

writer.writeAttribute("offset2", "");

writer.writeAttribute("remark", "");

writer.writeAttribute("type", QString("%1:reg").arg(QString::number(dwFindMax3xAddr - dwFindMin3xAddr)));

writer.writeAttribute("unit", "");

writer.writeAttribute("writeable", "0");

writer.writeAttribute("blockReadId", "");

writer.writeEndElement(); // <tag/>

}

QVector<TTagInfo \*> &vecTagInfo4x = mapBlockInfos["4x"];

quint32 dwFindMax4xAddr = 0; // 包内的最大地址

quint32 dwFindMin4xAddr = 0xffff; // 包内的最小地址

foreach(TTagInfo \*pObj, vecTagInfo4x) {

quint32 iMinAddrOffset = pObj->offset;

quint32 iMaxAddrOffset = pObj->length;

dwFindMax4xAddr = iMinAddrOffset;

dwFindMin4xAddr = iMaxAddrOffset;

QList<QString> memInfo = mapInfos.keys();

foreach (QString info, memInfo) {

if(info != "4x") {

continue;

}

QVector<TTagInfo> &vecInfo = mapInfos[info];

foreach(TTagInfo tagInfo, vecInfo) {

if(tagInfo.offset >= iMinAddrOffset && (tagInfo.offset + tagInfo.length) <= iMaxAddrOffset) {

pObj->use = true;

QPair<QString, QString> idPair;

idPair.first = QString::number(tagInfo.id);

idPair.second = QString::number(pObj->id);

idToBlockId.append(idPair);

if((tagInfo.offset + tagInfo.length) > dwFindMax4xAddr) {

dwFindMax4xAddr = tagInfo.offset + tagInfo.length / 2;

}

if(dwFindMin4xAddr > tagInfo.offset) {

dwFindMin4xAddr = tagInfo.offset;

}

}

}

}

writer.writeStartElement("tag"); // <tag>

writer.writeAttribute("addr", "4x");

writer.writeAttribute("addr2", "");

writer.writeAttribute("dev", dev);

writer.writeAttribute("group", "");

writer.writeAttribute("id", QString::number(pObj->id));

writer.writeAttribute("name", QString("4x\_%1").arg(QString::number(pObj->id)));

writer.writeAttribute("offset", QString::number(pObj->offset));

writer.writeAttribute("offset2", "");

writer.writeAttribute("remark", "");

writer.writeAttribute("type", QString("%1:reg").arg(QString::number(dwFindMax4xAddr - dwFindMin4xAddr)));

writer.writeAttribute("unit", "");

writer.writeAttribute("writeable", "0");

writer.writeAttribute("blockReadId", "");

writer.writeEndElement(); // <tag/>

}

writer.writeEndElement(); // <block\_tags/>

writer.writeEndDocument();

qDeleteAll(vecTagInfo0x);

qDeleteAll(vecTagInfo1x);

qDeleteAll(vecTagInfo3x);

qDeleteAll(vecTagInfo4x);

mapBlockInfos.clear();

return true;

}

## 编译插件工程

编译ModbusRTU插件工程，生成的ModbusRTU.dll文件位于目录HmiFuncDesignerBin\deviceplugins\ModbusRTU.dll

# ProjectManager相关配置

## 2.1 配置文件

修改配置文件HmiFuncDesignerBin\bin\Config\communication\_device.ini

配置文件内容如下：

####################################

## 设备列表

[DeviceSupportList]

list-count=6 #设备个数

list-1=GenDataProgram-模拟数据程序

list-2=IOModule-IO模块

list-3=PlcDevice-PLC

**list-4=ModbusDevice-Modbus设备**

list-5=GaugeDevice-仪表设备

list-6=CustomDevice-用户自定义设备

####################################

# 模拟数据程序

[GenDataProgram]

list-count=0

list-1=

######################################

# IO模块

[IOModule]

list-count=0

list-1=

######################################

# PLC设备

[PlcDevice]

list-count=0

list-1=Mitsubishi

list-2=SIEMENS

# SIEMENS-PLC设备

[SIEMENS]

list-count=0

list-1=S7\_200-COM

# Mitsubishi-PLC设备

[Mitsubishi]

list-count=0

list-1=FX2N-COM

######################################

# Modbus设备

**[ModbusDevice]**

**list-count=2**

**list-1=SerialModbus**

**list-2=TCPIPModbus**

# Serial\_Modbus设备

# COM, NET, BUS, OPC

[SerialModbus]

**list-count=1**

**list-1=ModbusRTU-COM #COM-串口设备**

list-2=ModbusASCII-COM

#list-count=4

#list-1=ModbusRTU\_slave-COM

#list-2=ModbusRTU-COM

#list-3=ModbusASCII\_slave-COM

#list-4=ModbusASCII-COM

# TcpIP\_Modbus设备

[TCPIPModbus]

**list-count=1**

**list-1=TCPIPModbus-NET #NET-网络设备**

#list-count=2

#list-1=TCPIPModbus\_slave-NET

#list-2=TCPIPModbus-NET

######################################

# 仪表设备

[GaugeDevice]

list-count=0

list-1=

######################################

# 用户自定义设备

[CustomDevice]

list-count=0

list-1=

## 2.2 ModbusRTU配置

ModbusRTU属于“ModbusDevice-Modbus设备”设备类中的“ModbusDevice”类别中的“SerialModbus”类，在“SerialModbus”类中ModbusRTU属于串口通信类别的，顾取值为“ModbusRTU-COM”。

####################################

## 设备列表

[DeviceSupportList]

list-count=配置项数量

list-第几项=ModbusDevice-Modbus设备

######################################

# Modbus设备

[ModbusDevice]

list-count=配置项数量

list-第几项=SerialModbus

######################################

# Serial\_Modbus设备

# COM, NET, BUS, OPC

[SerialModbus]

list-count=配置项数量

list-第几项=ModbusRTU-COM

# ProjectManager插件相关源码和UI

文件NewComDeviceDialog.cpp

文件VariableEditDialog.cpp

