设 计 文 件

名称　QTouch3 驱动设计说明书

编号

版本　V3.0

武汉舜通智能科技有限公司

编　制		工　艺	
校　核		标准化	
审　核		批　准	

版本号	更改人	更改日期	更改说明	变更编号
3.0	高伟锋	2020.6.16	完成 3.0 版本	

目录

1 目的和范围

1.1 目的

本文档详细介绍了 Qtouch3 驱动设计的基本格式和接口，使用这些接口函数可实现与设备的通讯、QTouch3 通讯配置的连接和实时数据库的交互。

开发的环境需求：VS2012+Qt5.4。

配置完开发环境后，提供的示例代码里面有自带的 pro 工程文件，用户可以双机运行 make.bat 生成针对 vs 的工程。

2 开发环境部署

2.1 安装 VS2012

在微软官网上下载即可，安装过程中可以选择默认安装，也可以自定义；选择自定义安装需要勾选C++的编程语言。

2.2 安装 Qt5.4

Qt 下载地址 http://download.qt.io/new_archive/qt/5.4/5.4.2/，选择的Qt版本为qt-opensource-windows-x86-msvc2012_opengl-5.4.2.exe。

下载完后，直接双机exe进行安装，安装过程直接默认即可。

2.3 Qt 和 Vs 集成环境安装(不是必须)

为了方便在VS上直接开发Qt的项目工程，需安装qt-vsaddin集成工具，可以下载qt-vs-addin-1.2.2-opensource进行安装，下载地址：http://download.qt.io/archive/vsaddin/1.2.2/。

2.4 环境变量设置

在系统中需添加如下环境变量：（添加环境变量的方法：右键“我的电脑”->属性->高级系统设置->环境变量；可以在用户变量中添加）

变量	值
QTDIR	QT的安装路径，比如：D:\Qt\Qt5.4.2\5.4\msvc2012_opengl
QMAKESPEC	win32-msvc2012
PATH（注意：PATH变量会存在多个值，win10可以直接新建增加一行，填入值即可，非win10的，添加的时候，在最后先写入分号再填值）	%QTDIR%\bin
include（注意事项同PATH）	%QTDIR%\include
lib（注意事项同PATH）	%QTDIR%\lib

环境变量设置完后，win10系统可以立即生效，非win10的可能需要重启才能生效；确认生效的方法，调出命令行cmd，然后输入qmake -v,有如下打印结果即可证明Qt的环境部署成功；

3 驱动开发包组成（以 modbusRTU 为例）

表 1

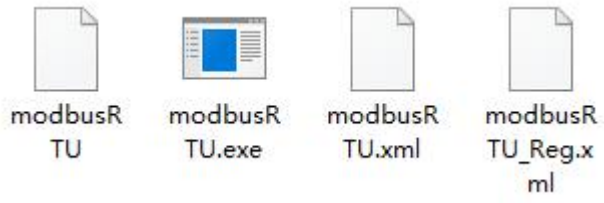
序号	一级文件	二级文件
1	common	ramrt 为 QTouch 数据接口，lib 为依赖的库
2	driverSDK	modbusRTU 源程序
3	modbusRTU	ioitem 协议参数配置内容

4 驱动执行依赖关系（以 modbusRTU 为例）

表 2

序号	文件	依赖关系
1	modbusRTU	增加一个 modusRTU 驱动之后,从 drive 包中拷贝出来的 linux 下执行程序
2	modbusRTU.exe	增加一个 modusRTU 驱动之后,从 drive 包中拷贝出来的 win 下执行程序
3	modbusRTU.xml	增加一个 modusRTU 驱动之后,由驱动设置窗口配置的初始化参数
4	modbusRTU_Reg.xml	由数据库管理配置的 IO 关联参数,根据这个文件进行发送帧配置

工程文件包组成如下（名称由用户在创建链路的时候自定义设置）：



5 源码主要接口文件

表 3

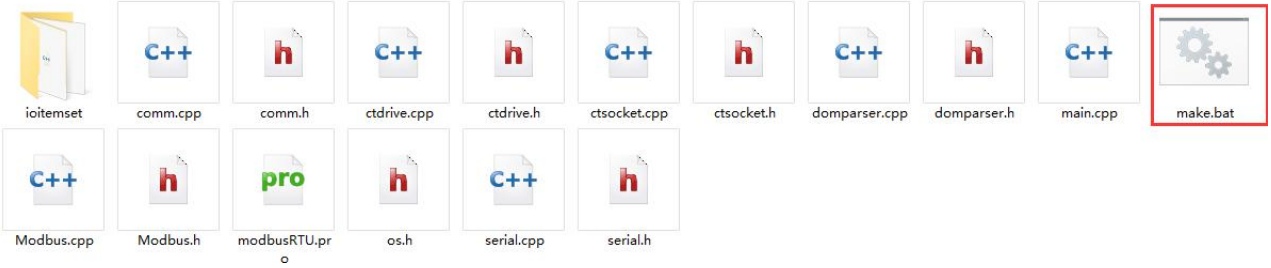
序号	主要接口文件	描述

序号	主要接口文件	描述
1	ramrt.h	实时数据接口
2	ramdrive.h	驱动特有的数据区，包含驱动的软喂狗、启动状态、链路通信状态等
3	domparser.h ，domparser.cpp	XML 文件读取，获取用户配置信息的接口
4	serial.h，serial.cpp	串口操作封装，对串口读写之调用（主要调用 QSerialPort）
5	ctsocket.h ，ctsocket.cpp	TCP 服务器端封装类（主要提供给驱动监视接口使用）
6	comm.h ，comm.cpp	对驱动程序处理的父类，共有特性初始化
7	Modbus.h ，Modbus.cpp	特定 Modbus 程序通讯处理类（协议处理类）
8	ctdrive.h，ctdrive.cpp	主框架处理类
9	main.cpp	程序 main 入口

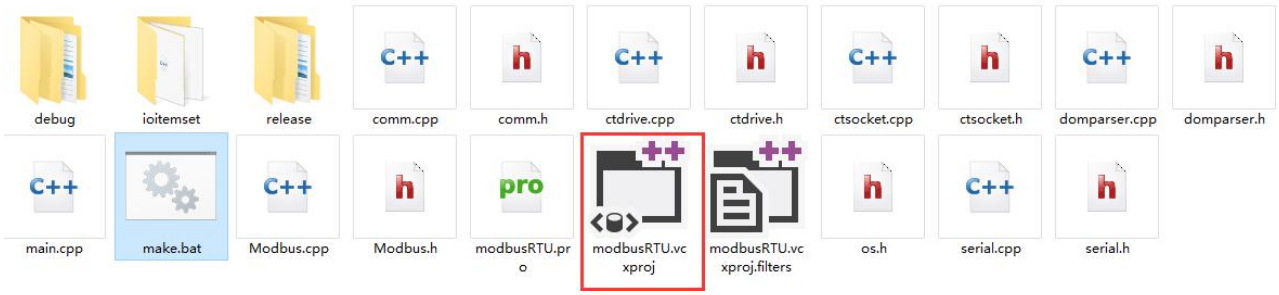
6 源码分析

6.1 VS 工程建立

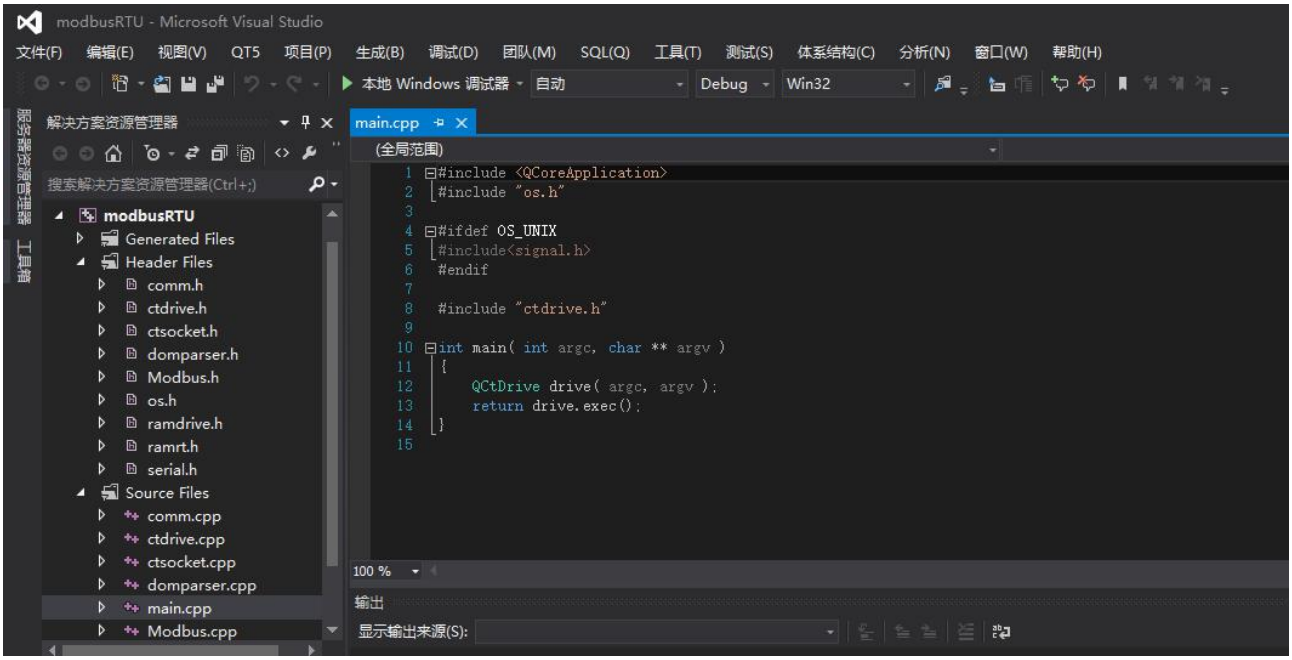
打开源码文件夹，可以双机make.bat生成VS的工程



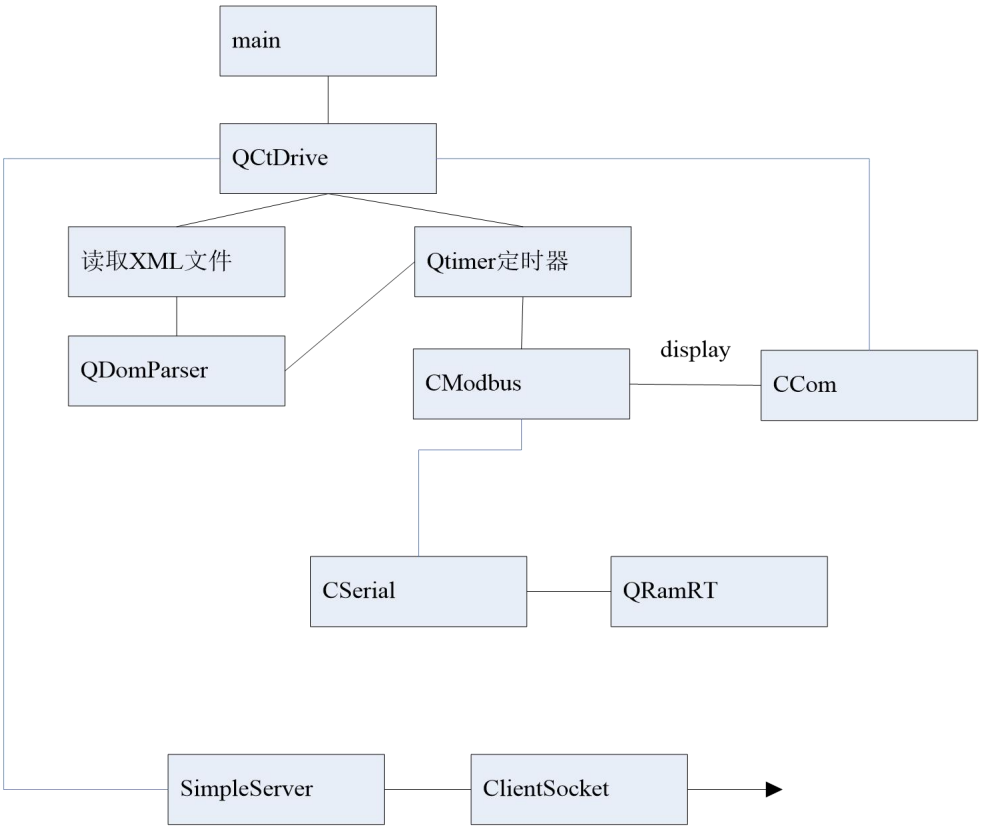
执行make.bat后，文件夹下会生成相应的VS工程文件，下面标注的文件即为vs的工程文件；



用vs2012打开该文件即可



6.2 执行流程



6.3 主要函数说明

QCtDrive管理整个驱动资源，负责调度驱动周期执行，驱动监视转发；（该类在实际开发过程中不需要变动）

1、QCtDrive构造函数：

```
QCtDrive::QCtDrive(int argc,char **argv )
: QCoreApplication( argc, argv )
{
    server = NULL ;
    client = NULL ;
    freshTimer = new QTimer(this);
    //驱动周期执行定时器
    //定时周期为在 QTouch 上配置的链路帧间隔
    connect( freshTimer, SIGNAL(timeout()), this, SLOT(freshRun()) );

#ifdef OS_WIN
    //处理驱动配置文件参数读取
    //组织驱动协议所需的结构信息
    setExistFile(argv[0]);
#endif

#ifdef OS_UNIX
    char *index ;
    index = rindex(argv[0],'/');
    QString filename(index);
    setExistFile(filename);
#endif
}
```

2、工程配置文件读取，初始化相关参数：该函数实行完后，驱动初始化全部完成，并进入循环执行模式

```
void QCtDrive::setExistFile(QString file)
{
#ifdef OS_WIN
    QTextCodec * codec = QTextCodec::codecForName( "GB2312" );
    QString name ,path;
    int index =0 ;
    file = file.replace(0x5c, "/");
    index = file.lastIndexOf("/",-1,Qt::CaseInsensitive);
    name = file.right(file.length() - index -1);
    name = name.left(name.length() - 4);
    filePath = file.left(index);
    fileName = name;
//读取资源配置
protocol.ReadXmlFile( codec->toUnicode(filePath.toLatin1()) ,codec->toUnicode(fileName.toLatin1()) );
#endif

#ifdef OS_UNIX
```

```
        file = file.right(file.length()-1);
        protocol.ReadXmlFile("/home/ctstor/ctfiles" ,file);
    #endif
        //启动驱动监视端口，驱动监视工具可以通过该端口获取驱动整个通信过程的收发报文；
        server = new SimpleServer( 5002 + protocol.rmtPort ,this );
    connect( server,SIGNAL(newConnect(ClientSocket*)),this ,SLOT(newConnect(ClientSocket*)
) );
        //创建打开数据区
        protocol.CreateDriveRam();
        //启动驱动循环处理定时器
        freshTimer->start(protocol.cycTime);
        //驱动常规初始化
        protocol.initStart();
    }
    3、驱动监视转发：
    void QCtDrive::freshDisplay(Buf buf)
    {
        if(client == NULL)
            return ;
        if(client->state() == QTcpSocket::UnconnectedState )
            return ;
        int len = sizeof(buf);
        client->write((char *)&buf,len);
    }
```

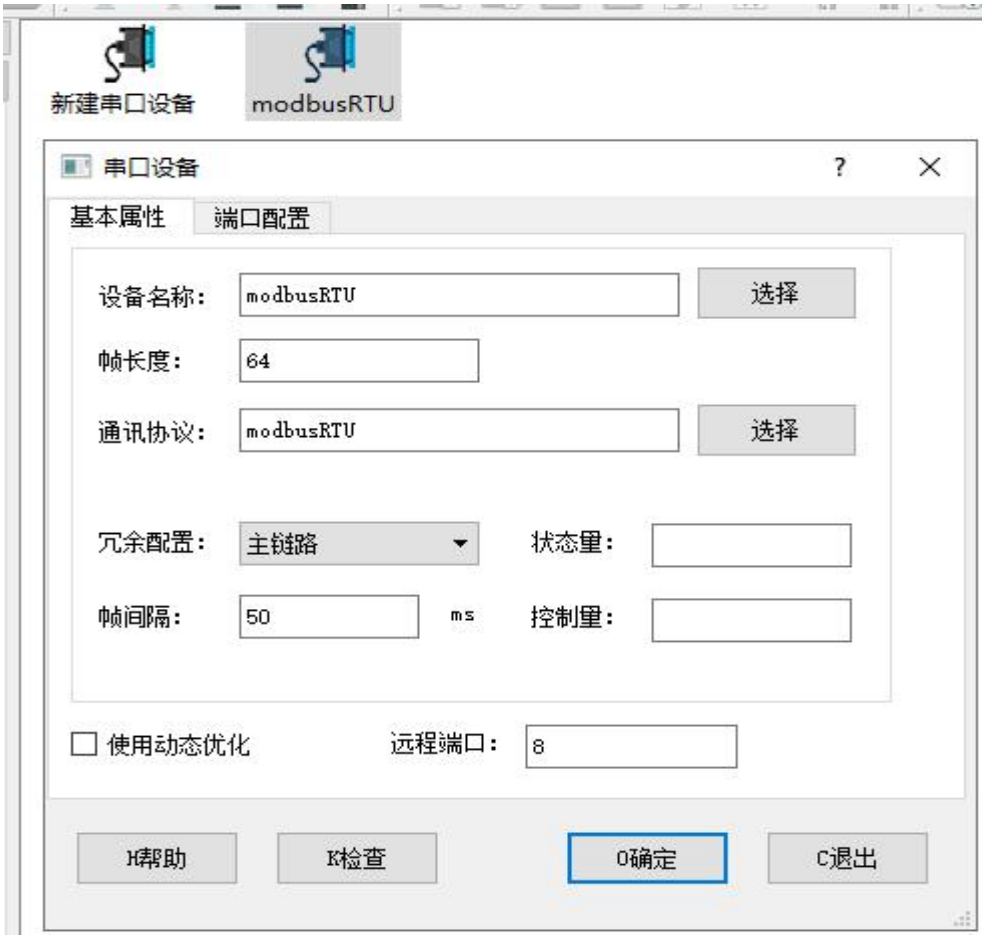
CCom类是协议处理的基类，主要处理函数ReadXmlFile；（该类在实际开发过程中不需要变动）

```
    void CCom::ReadXmlFile(QString path ,QString file)
    {
        QTextCodec * codec = QTextCodec::codecForName( "GB2312" );
        QString name;
        QString queryStr;
        int comNo = 0 ;
        int verifyBit = 0 ;
        int bResult = 0 ;

        name = file;

        //读取同名 xml 配置文件
        //读取驱动链路配置信息，针对 modbusRTU，主要是获取串口的：串口号、波特率、校验位、停
    止位、超时时间，帧间隔、帧长度等信息
        if(domparser.readCommLinkLayer(path,name))
        {
            //串口初始化
            comNo = domparser.comm.sComNum.right(domparser.comm.sComNum.length()-3).toInt();
```

```
            if(domparser.comm.sVerifyBit == codec->toUnicode("无校验"))
                verifyBit = 0 ;
            else if(domparser.comm.sVerifyBit == codec->toUnicode("偶校验"))
                verifyBit = 1 ;
            else if(domparser.comm.sVerifyBit == codec->toUnicode("奇校验"))
                verifyBit = 2 ;
            else if(domparser.comm.sVerifyBit == codec->toUnicode("固定 1 校验"))
                verifyBit = 3 ;
            else if(domparser.comm.sVerifyBit == codec->toUnicode("固定 0 校验"))
                verifyBit = 4 ;
        //初始化串口
        pSerial->SetValue(comNo,domparser.comm.iBaudRate,domparser.comm.iDataBit,domparser.
comm.iTimeOut,
                domparser.comm.iStopBit,verifyBit,50);

            //打开串口
            bResult = pSerial->OpenCom();
            //读取采集数据点 xml 配置文件
            domparser.readCommLinkLayer_Reg(path,name);
            //初始化数据读取包结构
            domparser.initRequestReg();
            querynum = domparser.RegList.count();

            //初始化 openinfo，该信息在启动驱动监视工具时，会将相关信息显示在工具上，方便用户
        查看串口的打开状态
            openinfo.comNo = comNo ;
            openinfo.bandRateLo = (domparser.comm.iBaudRate >> 8 ) & 0xFF;
            openinfo.bandRateHi = domparser.comm.iBaudRate & 0xFF ;
            openinfo.dataBit = domparser.comm.iDataBit ;
            openinfo.verifyBit = verifyBit ;
            openinfo.stopBit = domparser.comm.iStopBit ;
            openinfo.frameLen = domparser.comm.frameLen ;
            openinfo.cycTimeLo = (domparser.comm.cycTime >> 8) & 0xFF;
            openinfo.cycTimeHi = domparser.comm.cycTime  & 0xFF;
            openinfo.bResult = 1 ;

            //其他设置
            cycTime = domparser.comm.cycTime ;
            rdyType = domparser.comm.rdyType;
        //  dvStart = domparser.comm.dvStart ;
            rmtPort = domparser.comm.rmtPort ;
        }
    }
```

QDomParser类专门处理配置文件读取以及进行协议结构化信息处理，主要函数readCommLinkLayer和readCommLinkLayer_Reg；

1、读取串口链路配置信息(该函数在实际开发过程中不需要变动)

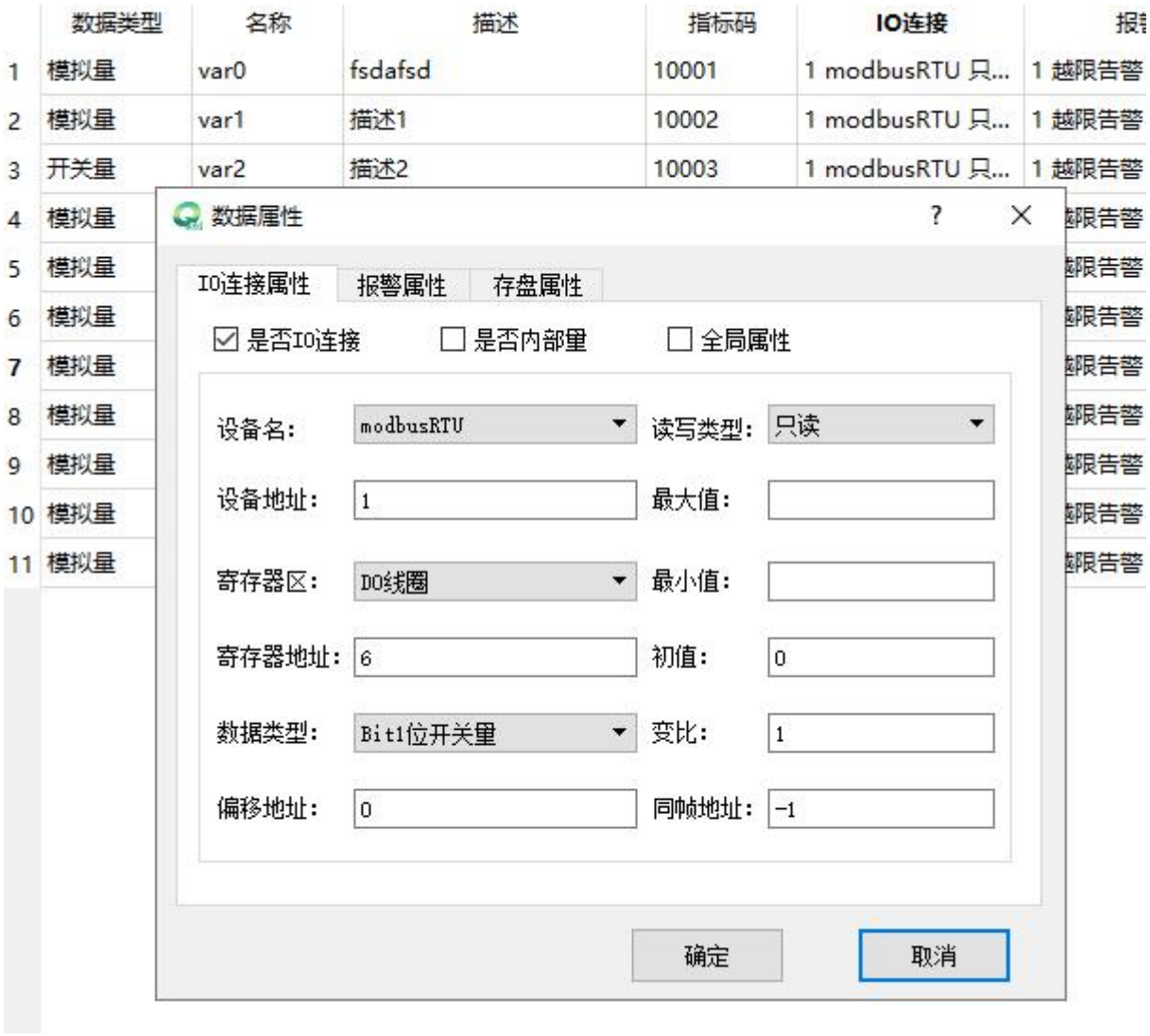


读取的内容即为上图的配置信息；

```
bool QDomParser::readCommLinkLayer(QString path ,QString sname)
{
    QDomDocument doc;
    QString name = QString(sname+".xml");
    name = path+ "/" + name ;
    QFile file(name);
    QString errorStr;
    int errorLine;
    int errorColumn;
    if (!doc.setContent(&file, true, &errorStr, &errorLine,
        &errorColumn)) {
        qWarning("Line %d, column %d: %s", errorLine, errorColumn,
            errorStr.toLatin1().data());
        return false;
    }
    QDomElement doc_root = doc.documentElement();
    if (doc_root.tagName() != sname) {
```

```
        qWarning("The file is not a runconfig file");
        return false ;
    }
    QDomElement element;
    QDomNode doc_node = doc_root.firstChild();
    while (!doc_node.isNull())
    {
        element = doc_node.toElement();
        if (element.tagName() == "sName")//驱动名称
            comm.sName = element.text();
        else if (element.tagName() == "frameLen")//帧长度
            comm.frameLen = element.text().toInt();
        else if (element.tagName() == "sProName")//协议名称
            comm.pName = element.text();
        else if (element.tagName() == "rdyType")//冗余配置
            comm.rdyType = element.text();
        else if (element.tagName() == "cycTime")//帧间隔
            comm.cycTime = element.text().toInt();
        else if (element.tagName() == "bStop")//是否使用动态优化
            comm.bStop = (bool)element.text().toInt();
        else if (element.tagName() == "sComNum")//串口号
            comm.sComNum = element.text();
        else if (element.tagName() == "iBaudRate")//波特率
            comm.iBaudRate = element.text().toInt();
        else if (element.tagName() == "iDataBit")//数据位
            comm.iDataBit = element.text().toInt();
        else if (element.tagName() == "iStopBit")//停止位
            comm.iStopBit = element.text().toInt();
        else if (element.tagName() == "sVerifyBit")//校验位
            comm.sVerifyBit = element.text();
        else if (element.tagName() == "iTimeOut")//串口超时
            comm.iTimeOut = element.text().toInt();
        else if (element.tagName() == "stateVar")//状态变量
            comm.stateVar = element.text();
        else if (element.tagName() == "ctlVar")//控制变量
            comm.ctlVar = element.text();
        else if (element.tagName() == "rmtPort")//远程端口
            comm.rmtPort = element.text().toInt();
        doc_node = doc_node.nextSibling();
    }
    return true;
}
```

2、读取配置点信息；（该方法在实际开发过程中，需根据实际的协议进行动态调整）



上图为单个点的配置信息；

```
bool QDomParser::readCommLinkLayer_Reg(QString path ,QString sname)
{
    QDomDocument doc;
    QString pname = QString(sname+"_Reg.xml");
    pname = path+ "/" + pname ;
    QFile file(pname);
    QString errorStr;
    int errorLine;
    int errorColumn;
    if (!doc.setContent(&file, true, &errorStr, &errorLine,
        &errorColumn)) {
        qWarning("Line %d, column %d: %s", errorLine, errorColumn,
            errorStr.toLatin1().data());
        return false;
    }
    QDomElement doc_root = doc.documentElement();
```

```
        if (doc_root.tagName() != QString(sname)) {
            qWarning("The file is not a runconfig file");
            return false ;
        }
        QTextCodec* m_textCodec = QTextCodec::codecForName("GB2312");
        QDomElement element;
        QDomNode doc_node = doc_root.firstChild();
        while (!doc_node.isNull())
        {
                element = doc_node.toElement();
                QString rtype = element.attribute("rtype"); //读写模式,（固定三种模式，只读、读写、只写）
                DataVar datavar  ;  //变量点对象
                QString stype = element.attribute("dtype");//点对象的数据类型，（该类型是由ioitem 里面定义的，建议按下下面进行固定处理，也可以自行定义类型说明）
                if ( stype == m_textCodec->toUnicode("Bit1 位开关量") )
                {
                     datavar.dtype = BIT_DATA;
                }
                else if ( stype==  m_textCodec->toUnicode("Byte8 位无符号数") )
                {
                     datavar.dtype = BYTE_DATA;
                }
                else if ( stype == m_textCodec->toUnicode("Char8 位有符号数") )
                {
                     datavar.dtype = CHAR_DATA;
                }
                else if ( stype == m_textCodec->toUnicode("Short16 位有符号数(AB)") )
                {
                     datavar.dtype = SHORT_DATA_1;
                }
                else if ( stype == m_textCodec->toUnicode("Short16 位有符号数(BA)") )
                {
                     datavar.dtype = SHORT_DATA_2;
                }
                else if ( stype == m_textCodec->toUnicode("Word16 位无符号数(AB)") )
                {
                    datavar.dtype = WORD_DATA_1;
                }
                else if ( stype == m_textCodec->toUnicode("Word16 位无符号数(BA)") )
                {
                     datavar.dtype = WORD_DATA_2;
                }
                else if ( stype == m_textCodec->toUnicode("Long32 位有符号数(ABCD)") )
```

```
{
     datavar.dtype = LONG_DATA_1;
}
else if ( stype == m_textCodec->toUnicode("Long32 位有符号数(BADC)") )
{
     datavar.dtype = LONG_DATA_2;
}
else if ( stype == m_textCodec->toUnicode("Long32 位有符号数(CDAB)") )
{
     datavar.dtype = LONG_DATA_3;
}
else if ( stype == m_textCodec->toUnicode("Long32 位有符号数(DCBA)") )
{
    datavar.dtype = LONG_DATA_4;
}
else if ( stype == m_textCodec->toUnicode("Dword32 位无符号数(ABCD)") )
{
    datavar.dtype = DWORD_DATA_1;
}
else if ( stype == m_textCodec->toUnicode("Dword32 位无符号数(BADC)") )
{
    datavar.dtype = DWORD_DATA_2;
}
else if ( stype == m_textCodec->toUnicode("Dword32 位无符号数(CDAB)") )
{
    datavar.dtype = DWORD_DATA_3;
}
else if ( stype == m_textCodec->toUnicode("Dword32 位无符号数(DCBA)") )
{
    datavar.dtype = DWORD_DATA_4;
}
else if ( stype == m_textCodec->toUnicode("Float 单精度浮点数(ABCD)") )
{
    datavar.dtype = FLOAT_DATA_1;
}
else if ( stype == m_textCodec->toUnicode("Float 单精度浮点数(BADC)") )
{
    datavar.dtype = FLOAT_DATA_2;
}
else if ( stype == m_textCodec->toUnicode("Float 单精度浮点数(CDAB)") )
{
    datavar.dtype = FLOAT_DATA_3;
}
else if ( stype == m_textCodec->toUnicode("Float 单精度浮点数(DCBA)") )
```

```
        {
            datavar.dtype = FLOAT_DATA_4;
        }
        else if ( stype == m_textCodec->toUnicode("Double 双精度浮点数") )
        {
            datavar.dtype = Double_DATA;
        }
        else if ( stype == m_textCodec->toUnicode("String 字符串") )
        {
            datavar.dtype = STRING_DATA;
        }
        else if ( stype == m_textCodec->toUnicode("BCD") )
        {
            datavar.dtype = BCD_32;
        }
        else
            datavar.dtype = SHORT_DATA_1;
        datavar.id    = element.attribute("id").toInt();//对应设备地址
        datavar.add   = element.attribute("add").toInt();//对应寄存器地址
        datavar.zero  = element.attribute("zero").toDouble();//对应初值
        datavar.k     = element.attribute("k").toDouble();//对应变比
        datavar.index = element.attribute("index").toInt();//对应点的内存编号，由
QTouch 分配，该变量在 QTouch 是唯一标识数据的点号
        QString sReg   = element.attribute("reg");//寄存器类型，（该部分是自定义，不
同协议定义的寄存器类型不同，具体定义在 ioitem 中处理）
        if (sReg == m_textCodec->toUnicode("DO 线圈"))  //数据区转换
            datavar.reg = DO_REG;
        if (sReg == m_textCodec->toUnicode("DI 离散输入寄存器"))
            datavar.reg = DI_REG;
        if (sReg == m_textCodec->toUnicode("AO 保持寄存器"))
            datavar.reg = AO_REG;
        if (sReg == m_textCodec->toUnicode("AI 输入寄存器"))
            datavar.reg = AI_REG;
        datavar.max   = element.attribute("max");//点的最大值
        datavar.min   = element.attribute("min");//点的最小值
        datavar.offsetadd = element.attribute("offsetadd");//偏移地址
        //2015.9.11 add 设备状态
        DataStatu datastatu;
        if (sReg == m_textCodec->toUnicode("设备状态"))
        {
            datastatu.id = element.attribute("id").toInt();
            datastatu.index = element.attribute("index").toInt();
            StatuList.append(datastatu);
        }
```

```
            else
            {
                 //读写分链表存储
                if(rtype == m_textCodec->toUnicode("只读"))
                {
                    VarList.append(datavar);
                }
                else if(rtype == m_textCodec->toUnicode("只写"))
                {
                    WriteList.append(datavar);
                }
                else
                {
                    DataVar datavar2 ;
                    datavar2 = datavar;
                    WriteList.append(datavar);
                    VarList.append(datavar2);
                }
            }
        doc_node = doc_node.nextSibling();
    }
    int tablenum = VarList.count();
    //规划查询帧
    //根据设备地址、寄存器类型、寄存器地址、帧长度对配置的点进行分组管理
    //分组后，每个组可以发一次请求即可获取整个组的数据；
    //此部分是根据具体协议进行处理
    if (tablenum > 0)
    {
        tidy( tablenum );   //整理排序表
        for (int i = 0 ; i < tablenum ; i++ )
        {
            DataVar DaVar = VarList[i];
            if ( judgeReg(DaVar.id, DaVar.add,DaVar.dtype,DaVar.reg) )
            {
                insertReg(DaVar, type, order); //粘贴
            }else
            {
                newReg(DaVar, type );  //新建
            }
        }
    }
    return true;
}
```

CModbus协议处理类由CCom类派生而来：主要处理协议通过链路的收发、数据解析、数据写入QTouch实时数据区、处理由上而下的控制指令

1、循环执行体Run函数(该函数在实际开发过程中不建议做过大的改动)

```
void CModbus::Run()
{
    //add link heart
    //添加链路存活软喂狗
    Driveramrt->SetLinkHeart(domparser.comm.rmtPort, -1);
    //扫描是否有协议指令操作
    ScanCmd();
    //检查是否有对驱动本体的控制操作
    CheckContrl();
    if(!pSerial->IsOpen()){
        //设置通信链路的状态
        Driveramrt->SetLinkState(domparser.comm.rmtPort, 1);
        return;
    }
    unsigned short shCRC ;

    if(nCurrentIndex >= querynum) nCurrentIndex = 0;

    Driveramrt->SetLinkState(domparser.comm.rmtPort, 0);

    //如果前面有指令下发操作，该周期内就不发请求，
    //防止下发和请求一起发送造成设备不响应
    if(bExecCmd){
        bExecCmd = false;
        return;
    }

    iOutputLen = 8;
    //从组好的数据请求包链表中取出当前序列报文发送
    chOutput = domparser.RegList[nCurrentIndex].chReq;
    int nRet =pSerial->WriteCom( chOutput, 8 );

    if ( nRet == iOutputLen )
        display( MSG_COMM_OUTPUT );

    //从串口中读取设备的响应
    iInputLen =
pSerial->ReadCom( chInput,domparser.RegList[nCurrentIndex].nRequestReadLen);
    chInput[iInputLen] = 0 ;

    int StatuIndex = 0;
```

```
int StatuCount = 0;
int isWriteStatu = 0;
    StatuCount = domparser.StatuList.count();
for (int n=0; n<(int)domparser.StatuList.count(); n++)
{
    if ( chOutput[0] == domparser.StatuList[n].id )
    {
        StatuIndex = domparser.StatuList[n].index;
        isWriteStatu = 0;
        break;
    }
    else
    {
        isWriteStatu = 1;
    }
}

if( iInputLen > 3 )
{
    iCommCount[nCurrentIndex] = 0 ;
    display( MSG_COMM_INPUT );
    //判断设备地址是否匹配
    if ( *( chOutput + 0) != *( chInput + 0 ) )
    {
        iCommCount[nCurrentIndex]++;

        nCurrentIndex++;
        return;
    }
    //判断功能码是否匹配
    if( *( chOutput + 1) != *( chInput + 1 ) )
    {
        iCommCount[nCurrentIndex]++;
        nCurrentIndex++;
        return;
    }
    //判断校验是否正确
    shCRC = CRC16( chInput, iInputLen - 2 );
    if ( StatuCount > 0 && isWriteStatu==0)
    {
        ramrt->SetItemValue(0,StatuIndex,1);//0 异常 1 正常
    }
    //数据解析
    Parse( domparser.RegList[nCurrentIndex], chInput, iInputLen,shCRC);
```

```
        }
        else
        {
            //收到的数据异常，异常包计数
            //累计计数到 3 次，认为该链路异常
            iCommCount[nCurrentIndex] ++ ;
            if( iCommCount[nCurrentIndex] > 3 )
            {
                if ( StatuCount > 0 && isWriteStatu==0)
                {
                    ramrt->SetItemValue(0,StatuIndex,0);//0 异常 1 正常
                }
                iCommCount[nCurrentIndex] = 0 ;
                Driveramrt->SetLinkState(domparser.comm.rmtPort, 1);
            }
        }

        nCurrentIndex++;

    }
```

2、数据解析Parse函数（该函数需要根据具体的协议进行调整）

```
    void CModbus::Parse(DataReg pdataReg,unsigned char *chInput, unsigned short
iInputLen,unsigned short shCRC)
    {
        unsigned int i, intTmp , iCount;
        unsigned char chTemp  ;
        unsigned short ushTmp ;
        short shTmp ;
        double iValue;
        float flTmp;
        int bytecount;  //字节序号
        int bitcount;   //位序号

        char chBufTemp[8];
        double doubleTemp=0.0;

        unsigned short chCRC[2];
        chCRC[0] = (shCRC >> 8) & 0xFF ;
        chCRC[1] = shCRC & 0xFF ;
        //判断校验是否正确
        if( *(chInput + iInputLen - 1 ) != chCRC[1] || *(chInput + iInputLen - 2 ) != chCRC[0] )
            return ;
```

```
iCount = *( chInput + 2 );
//根据不同的寄存器区
//不同的数据类型进行解析
if(  pdataReg.reg == DO_REG || pdataReg.reg == DI_REG  )
{//DO 和 DI 区的数据解析
    for( i = 0; i < (int)pdataReg.dataNUM; i++ ){

        bytecount = (pdataReg.dataIO[i].count>>3) ;
        bitcount = pdataReg.dataIO[i].count - (bytecount<<3);

        chTemp = *( chInput + 3 + bytecount );
        iValue = chTemp >> bitcount & 0x01 ;
        ramrt->SetItemValue(0,pdataReg.dataIO[i].index,iValue);
    }
}

else if ( pdataReg.reg == AO_REG || pdataReg.reg == AI_REG)
{//AO 和 AI 数据区的解析
    for( i = 0; i < (int)pdataReg.dataNUM; i++ )
    {
        if ( pdataReg.dataIO[i].dtype == BIT_DATA )
        {//bit 数据类型
            ushTmp = (*( chInput + 3 + (pdataReg.dataIO[i].count<<1) )<<8)
                + *(chInput + 3 + (pdataReg.dataIO[i].count<<1) + 1) ;
            iValue = (ushTmp >> pdataReg.dataIO[i].doffsetadd.toInt()) & 0x01;
        }
        else if ( pdataReg.dataIO[i].dtype==  BYTE_DATA )
        {//byte 数据类型
            if(pdataReg.dataIO[i].doffsetadd.toInt() == 0)
                ushTmp = *( chInput + 3 + (pdataReg.dataIO[i].count<<1) + 1) ;
            else
                ushTmp = *( chInput + 3 + (pdataReg.dataIO[i].count<<1));

            iValue = ushTmp ;

        }
        else if ( pdataReg.dataIO[i].dtype == SHORT_DATA_1 )
        {//short 数据类型(高前低后)
            ushTmp = (*( chInput + 3 + (pdataReg.dataIO[i].count<<1))<<8)
                + *(chInput + 3 + (pdataReg.dataIO[i].count<<1) + 1) ;
            memcpy( &shTmp, &ushTmp, 2 );
            iValue = shTmp ;
        }
        else if ( pdataReg.dataIO[i].dtype == SHORT_DATA_2 )
```

```
{//short 数据类型(低前高后)
    ushTmp = (*( chInput + 3 + (pdataReg.dataIO[i].count<<1)+1)<<8)
        + *(chInput + 3 + (pdataReg.dataIO[i].count<<1)) ;
    memcpy( &shTmp, &ushTmp, 2 );
    iValue = shTmp ;
}
else if ( pdataReg.dataIO[i].dtype == WORD_DATA_1)
{//word 数据类型(高前低后)
    ushTmp = (*( chInput + 3 + (pdataReg.dataIO[i].count<<1))<<8)
        + *(chInput + 3 + (pdataReg.dataIO[i].count<<1) + 1) ;
    iValue = ushTmp ;
}
else if ( pdataReg.dataIO[i].dtype == WORD_DATA_2)
{//word 数据类型(低前高后)
    ushTmp = (*( chInput + 3 + (pdataReg.dataIO[i].count<<1)+1)<<8)
        + *(chInput + 3 + (pdataReg.dataIO[i].count<<1)) ;
    iValue = ushTmp ;
}
else if ( pdataReg.dataIO[i].dtype == FLOAT_DATA_1 )
{//float 数据类型(ABCD)
    intTmp = (*( chInput + 3 + (pdataReg.dataIO[i].count<<1))<<24)
        + (*(chInput + 3 + (pdataReg.dataIO[i].count<<1) + 1)<<16)
        + (*(chInput + 3 + (pdataReg.dataIO[i].count<<1) + 2)<<8)
        + *(chInput + 3 + (pdataReg.dataIO[i].count<<1) + 3);
    memcpy(&flTmp,&intTmp,4);
    iValue = flTmp ;
}
else if ( pdataReg.dataIO[i].dtype == FLOAT_DATA_2 )
{//float 数据类型(BADC)
    intTmp = (*( chInput + 3 + (pdataReg.dataIO[i].count<<1))<<16)
        + (*(chInput + 3 + (pdataReg.dataIO[i].count<<1) + 1)<<24)
        + (*(chInput + 3 + (pdataReg.dataIO[i].count<<1) + 2))
        + (*(chInput + 3 + (pdataReg.dataIO[i].count<<1) + 3)<<8);
    memcpy(&flTmp,&intTmp,4);
    iValue = flTmp ;
}
else if ( pdataReg.dataIO[i].dtype == FLOAT_DATA_3 )
{//float 数据类型(CDAB)
    intTmp = (*( chInput + 3 + (pdataReg.dataIO[i].count<<1))<<8)
        + (*(chInput + 3 + (pdataReg.dataIO[i].count<<1) + 1))
        + (*(chInput + 3 + (pdataReg.dataIO[i].count<<1) + 2)<<24)
        + (*(chInput + 3 + (pdataReg.dataIO[i].count<<1) + 3)<<16);
    memcpy(&flTmp,&intTmp,4);
    iValue = flTmp ;
```

```
}
else if ( pdataReg.dataIO[i].dtype == FLOAT_DATA_4 )
{//float 数据类型(DCBA)
    intTmp = (*( chInput + 3 + (pdataReg.dataIO[i].count<<1)))
        + (*(chInput + 3 + (pdataReg.dataIO[i].count<<1) + 1)<<8)
        + (*(chInput + 3 + (pdataReg.dataIO[i].count<<1) + 2)<<16)
        + (*(chInput + 3 + (pdataReg.dataIO[i].count<<1) + 3)<<24);
    memcpy(&flTmp,&intTmp,4);
    iValue = flTmp ;
}
else if ( pdataReg.dataIO[i].dtype == Double_DATA )
{//double 数据类型
    chBufTemp[0] = *( chInput + 3 + (pdataReg.dataIO[i].count<<1)+1);
    chBufTemp[1] = *( chInput + 3 + (pdataReg.dataIO[i].count<<1)+0);
    chBufTemp[2] = *( chInput + 3 + (pdataReg.dataIO[i].count<<1)+3);
    chBufTemp[3] = *( chInput + 3 + (pdataReg.dataIO[i].count<<1)+2);
    chBufTemp[4] = *( chInput + 3 + (pdataReg.dataIO[i].count<<1)+5);
    chBufTemp[5] = *( chInput + 3 + (pdataReg.dataIO[i].count<<1)+4);
    chBufTemp[6] = *( chInput + 3 + (pdataReg.dataIO[i].count<<1)+7);
    chBufTemp[7] = *( chInput + 3 + (pdataReg.dataIO[i].count<<1)+6);

    memcpy(&doubleTemp,&chBufTemp,8);

    iValue = doubleTemp ;
}
else if ( pdataReg.dataIO[i].dtype == DWORD_DATA_1 )
{//dword 数据类型(ABCD)
 intTmp = (*( chInput + 3 + (pdataReg.dataIO[i].count<<1))<<24)
        + (*(chInput + 3 + (pdataReg.dataIO[i].count<<1) + 1)<<16)
        + (*(chInput + 3 + (pdataReg.dataIO[i].count<<1) + 2)<<8)
        + *(chInput + 3 + (pdataReg.dataIO[i].count<<1) + 3);

    iValue = intTmp ;
}
else if ( pdataReg.dataIO[i].dtype == DWORD_DATA_2 )
{//dword 数据类型(BADC)
    intTmp = (*( chInput + 3 + (pdataReg.dataIO[i].count<<1))<<16)
        + (*(chInput + 3 + (pdataReg.dataIO[i].count<<1) + 1)<<24)
        + (*(chInput + 3 + (pdataReg.dataIO[i].count<<1) + 2))
        + (*(chInput + 3 + (pdataReg.dataIO[i].count<<1) + 3)<<8);

    iValue = intTmp ;
}
else if ( pdataReg.dataIO[i].dtype == DWORD_DATA_3 )
```

```
{//dword 数据类型(CDAB)
    intTmp = (*( chInput + 3 + (pdataReg.dataIO[i].count<<1))<<8)
        + (*(chInput + 3 + (pdataReg.dataIO[i].count<<1) + 1))
        + (*(chInput + 3 + (pdataReg.dataIO[i].count<<1) + 2)<<24)
        + (*(chInput + 3 + (pdataReg.dataIO[i].count<<1) + 3)<<16);

    iValue = intTmp ;
}
else if ( pdataReg.dataIO[i].dtype == DWORD_DATA_4 )
{//dword 数据类型(DCBA)
    intTmp = (*( chInput + 3 + (pdataReg.dataIO[i].count<<1)))
        + (*(chInput + 3 + (pdataReg.dataIO[i].count<<1) + 1)<<8)
        + (*(chInput + 3 + (pdataReg.dataIO[i].count<<1) + 2)<<16)
        + (*(chInput + 3 + (pdataReg.dataIO[i].count<<1) + 3)<<24);

    iValue = intTmp ;
}
else if ( pdataReg.dataIO[i].dtype == LONG_DATA_1 )
{//long 数据类型(ABCD)
    unsigned int uvalue = (*( chInput + 3 + (pdataReg.dataIO[i].count<<1))<<24)
        + (*(chInput + 3 + (pdataReg.dataIO[i].count<<1) + 1)<<16)
        + (*(chInput + 3 + (pdataReg.dataIO[i].count<<1) + 2)<<8)
        + *(chInput + 3 + (pdataReg.dataIO[i].count<<1) + 3);

    int temp = 0;

    memcpy(&temp, &uvalue, 4);

    iValue = temp;
}
else if ( pdataReg.dataIO[i].dtype == LONG_DATA_2 )
{//long 数据类型(BADC)
    unsigned int uvalue = (*( chInput + 3 + (pdataReg.dataIO[i].count<<1))<<16)
        + (*(chInput + 3 + (pdataReg.dataIO[i].count<<1) + 1)<<24)
        + (*(chInput + 3 + (pdataReg.dataIO[i].count<<1) + 2))
        + (*(chInput + 3 + (pdataReg.dataIO[i].count<<1) + 3)<<8);

    int temp = 0;

    memcpy(&temp, &uvalue, 4);

    iValue = temp;
}
else if ( pdataReg.dataIO[i].dtype == LONG_DATA_3 )
```

```
                {//long 数据类型(CDAB)
                    unsigned int uvalue = (*( chInput + 3 + (pdataReg.dataIO[i].count<<1))<<8)
                        + (*(chInput + 3 + (pdataReg.dataIO[i].count<<1) + 1))
                        + (*(chInput + 3 + (pdataReg.dataIO[i].count<<1) + 2)<<24)
                        + (*(chInput + 3 + (pdataReg.dataIO[i].count<<1) + 3)<<16);

                    int temp = 0;

                    memcpy(&temp, &uvalue, 4);

                    iValue = temp;
                }
                else if ( pdataReg.dataIO[i].dtype == LONG_DATA_4 )
                {//long 数据类型(DCBA)
                    unsigned int uvalue = (*( chInput + 3 + (pdataReg.dataIO[i].count<<1)))
                        + (*(chInput + 3 + (pdataReg.dataIO[i].count<<1) + 1)<<8)
                        + (*(chInput + 3 + (pdataReg.dataIO[i].count<<1) + 2)<<16)
                        + (*(chInput + 3 + (pdataReg.dataIO[i].count<<1) + 3)<<24);

                    int temp = 0;

                    memcpy(&temp, &uvalue, 4);

                    iValue = temp;
                }

                //根据变比、零值、最大、最小等设置条件，进行最终数据的计算
                iValue = iValue * pdataReg.dataIO[i].dK + pdataReg.dataIO[i].dZero;

                if ( pdataReg.dataIO[i].dMax != "" )
                {
                    if (iValue > pdataReg.dataIO[i].dMax.toDouble())
                        iValue = pdataReg.dataIO[i].dMax.toDouble();
                }
                if ( pdataReg.dataIO[i].dMin != "" )
                {
                    if (iValue < pdataReg.dataIO[i].dMin.toDouble())
                        iValue = pdataReg.dataIO[i].dMin.toDouble();
                }
                //写到 QTouch 的实时数据区
                ramrt->SetItemValue(0,pdataReg.dataIO[i].index,iValue);
            }
        }
    }
```

3、执行扫描执行函数ScanCmd函数

```
// 执行收到的命令
void CModbus::ScanCmd()
{
    //判断是否是退出指令
    //当整个 QTouch 运行系统正常退出时，此消息会发出
    if(strcmp(ramrt->GetKcData(0)->dataKC,"EXIT") == 0){
        quitDrives();
        return ;
    }
    //判断是否有指令
    if(ramrt->GetKcFlag(0) == 1)  //控制标志
    {
        //获取指令内容
        int wrindex = ramrt->GetKcData(0)->kcNo ;  //获取写序号
        char chCmd[200];
        strcpy(chCmd,ramrt->GetKcData(0)->dataKC); //获取写 Value

        DataVar writevar ;
        if( findWiteKcData(wrindex,writevar) ) //找到写结构
        {
            //ExecCmd 函数需根据具体的协议进行调整
            if ( ExecCmd(writevar, chCmd))      //执行控制结构
                ramrt->SetKcFlag(0,0);          //清空标志
        }
    }
    //驱动专有指令区，在处理快速并行的下发操作时，会用到此部分；
    if (Driveramrt->GetExistCmd(domparser.comm.rmtPort))
    {
        char newcmd[32];
        Driveramrt->GetCmd(domparser.comm.rmtPort, newcmd);

        QStringList Qcmd =QString(newcmd).split(" ");
        if(Qcmd.count()>1){
            DataVar writevar ;
            int nIndex=Qcmd[0].toInt();
            if( findWiteKcData(nIndex,writevar) ) //找到写结构
            {
                bExecCmd = true;
                if ( ExecCmd(writevar, Qcmd))      //执行控制结构
                    Driveramrt->ClearCmdMark(domparser.comm.rmtPort);
            }
        }
    }
```

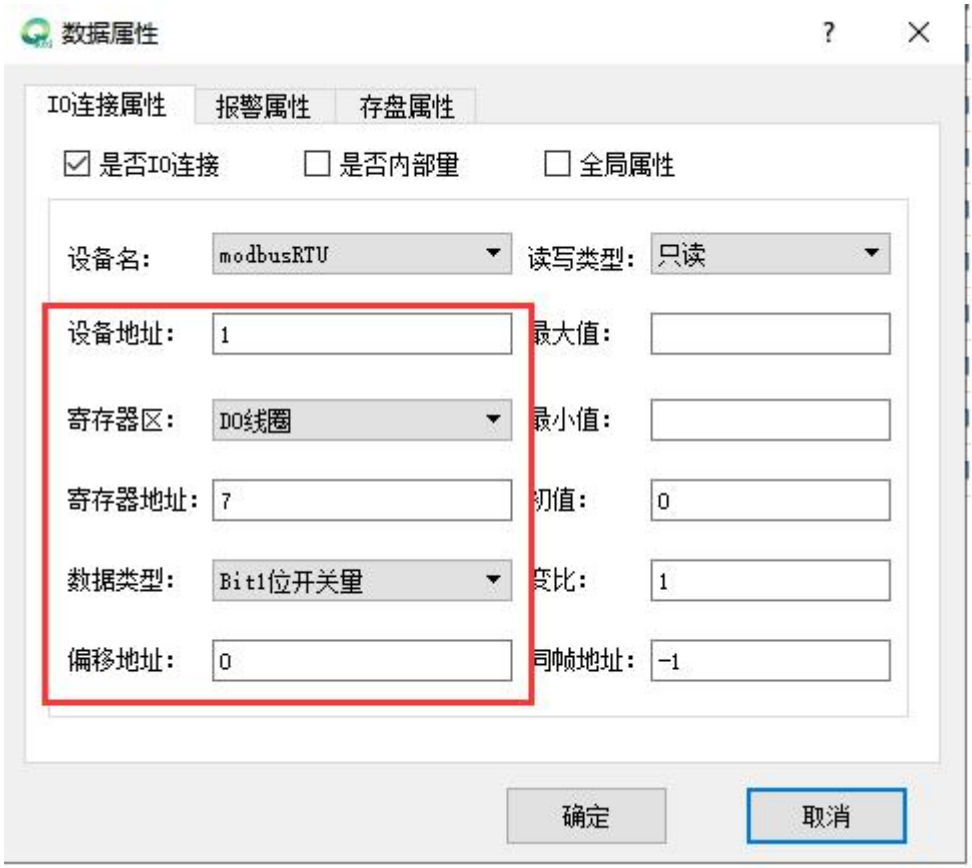
```
}
```

7 驱动自定义配置

每一个驱动在QTouch上的IO配置会存在一些差异，用户可以根据自己的需求进行重新设计；每个驱动的source下面有对应的一个ioitem文件夹，修改QDevSet::QDevSet()的初始化即可（如下图）；

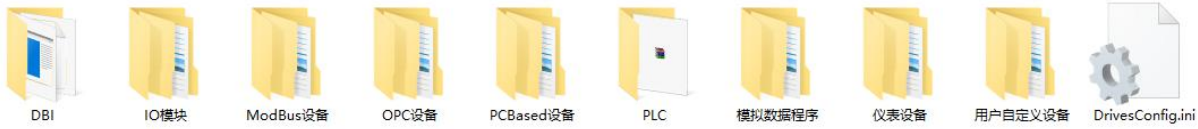
```
QDevSet::QDevSet()
{
    QTextCodec * codec = QTextCodec::codecForName( "GB2312" );

    //设备地址
    value.sidTaget = codec->toUnicode("设备地址") ;
    value.sidValue = codec->toUnicode("1") ;

    //寄存器区
    value.sregTaget = codec->toUnicode("寄存器区") ;
    value.sregValue << codec->toUnicode("DO线圈") ;
    value.sregValue << codec->toUnicode("DI离散输入寄存器");
    value.sregValue << codec->toUnicode("AO保持寄存器") ;
    value.sregValue << codec->toUnicode("AI输入寄存器") ;


    //寄存器地址
    value.saddTaget = codec->toUnicode("寄存器地址") ;
    value.saddValue = codec->toUnicode("0") ;

    //数据类型
    value.stypeTaget = codec->toUnicode("数据类型") ;
    value.stypeValue << codec->toUnicode("Bit1位开关量");
    value.stypeValue << codec->toUnicode("Char8位有符号数");
    value.stypeValue << codec->toUnicode("Byte8位无符号数");
    value.stypeValue << codec->toUnicode("Short16位有符号数(AB)");
    value.stypeValue << codec->toUnicode("Short16位有符号数(BA)");
    value.stypeValue << codec->toUnicode("Word16位无符号数(AB)");
    value.stypeValue << codec->toUnicode("Word16位无符号数(BA)");
    value.stypeValue << codec->toUnicode("Long32位有符号数(ABCD)");
    value.stypeValue << codec->toUnicode("Long32位有符号数(BADC)");
    value.stypeValue << codec->toUnicode("Long32位有符号数(CDAB)");
    value.stypeValue << codec->toUnicode("Long32位有符号数(DCBA)");
    value.stypeValue << codec->toUnicode("Dword32位无符号数(ABCD)");
    value.stypeValue << codec->toUnicode("Dword32位无符号数(BADC)");
    value.stypeValue << codec->toUnicode("Dword32位无符号数(CDAB)");
    value.stypeValue << codec->toUnicode("Dword32位无符号数(DCBA)");
    value.stypeValue << codec->toUnicode("Float单精度浮点数(ABCD)");
```

生成的ioitem.dll跟驱动exe放在同一目录下面；进过步骤8的集成方法，用户在配置自定义驱动的io配置，即可呈现代码设计的内容；



8 用户自定义驱动方法

用户可以将自己定义的驱动文件放到QTouch的drives目录下，按照一定的规格保存，即可以实现自定义驱动的添加，QTouch的参数系统就会找到这个文件。



规则文件为 DrivesConfig.ini，主要对 drives 下面的文件夹进行管理，用户可以增加一个专用的文件夹。文件夹分为 2 级管理，如上图所示为一级文件夹，主要是对设备的分类，一般用户可以不新建一级文件夹，在一级文件夹之下，可以新建自己的文件夹，用于将自定义的程序保存进来。

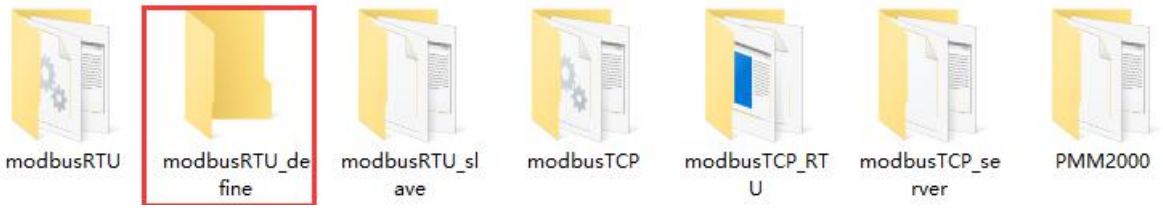
以 ModBus 设备为例：

```
[Drive4]
DriveName=ModBus设备
SonNum=6
SonDrive0=modbusRTU
SonDrive1=modbusTCP
SonDrive2=modbusRTU_slave
SonDrive3=modbusASCII
SonDrive4=modbusTCP_server
SonDrive5=modbusTCP_RTU
```

如上表明：在 ModBus 设备这个文件夹下面有 6 个子文件夹，如用户要增加一个专用文件夹则可以如下操作：

```
[Drive4]
DriveName=ModBus设备
SonNum=7
SonDrive0=modbusRTU
SonDrive1=modbusTCP
SonDrive2=modbusRTU_slave
SonDrive3=modbusASCII
SonDrive4=modbusTCP_server
SonDrive5=modbusTCP_RTU
SonDrive6=modbusRTU_define
```

完成这个设置之后，在 Modbus 设备下就可以建立一个自定义文件夹了：

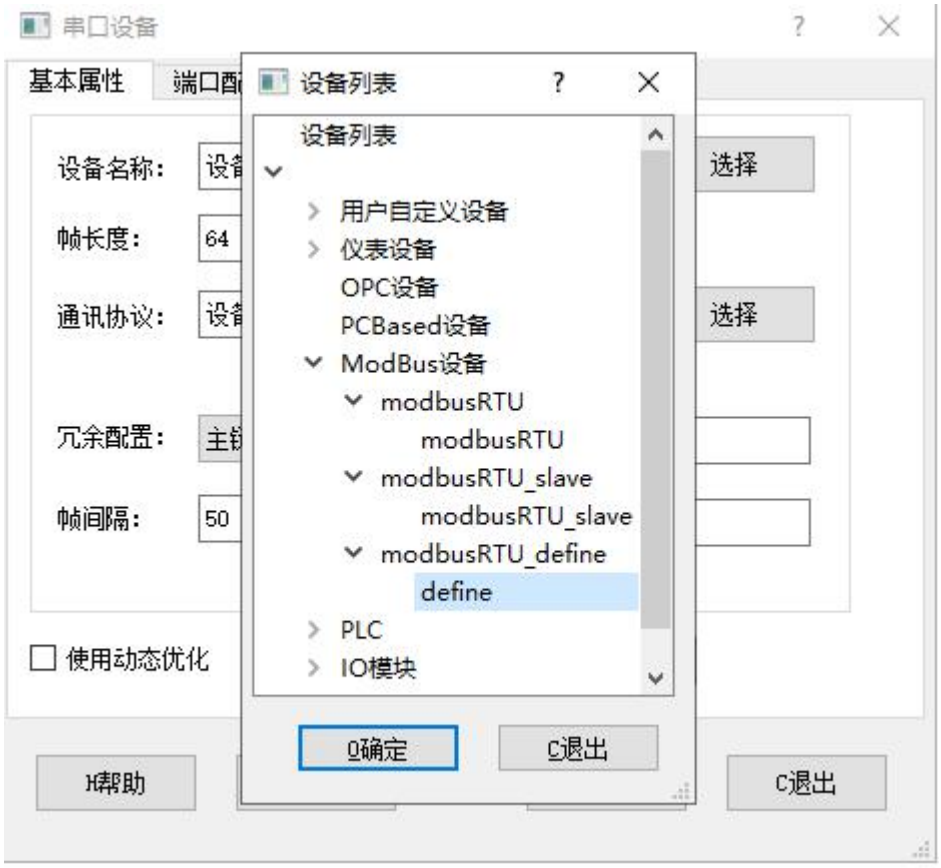


在自定义文件夹下，用户就可以添加自定义驱动，需要增加一个配置文件 CONFIG.ini：

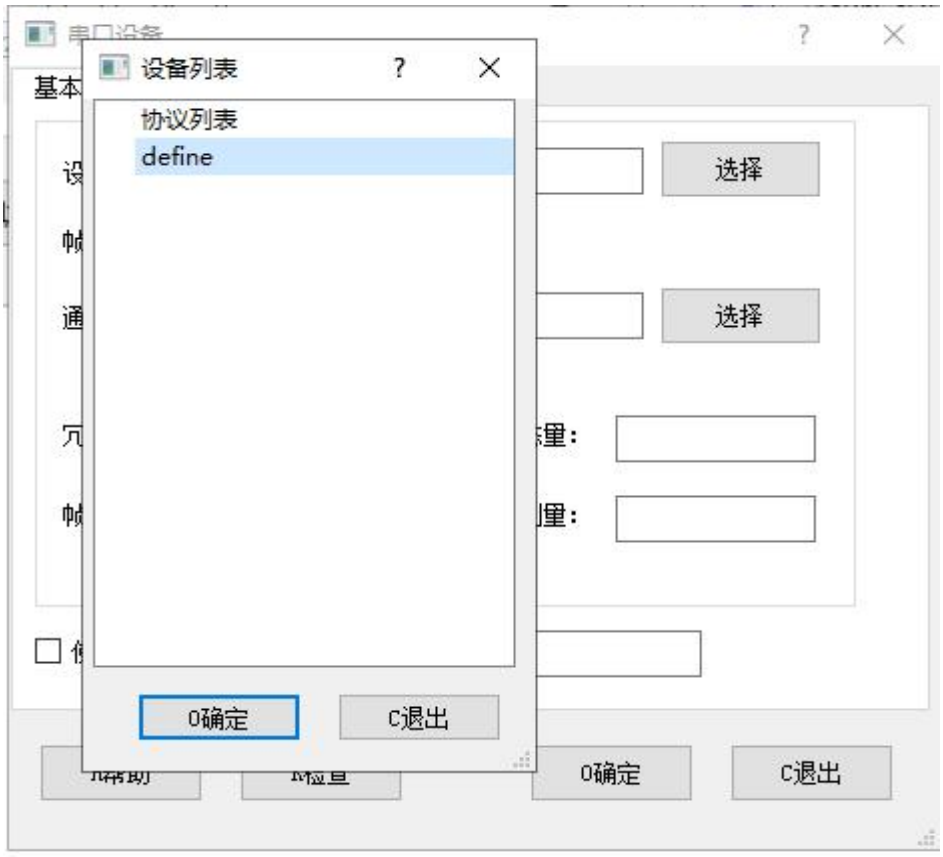
```
[Drive]
DrivesNum=1

[Drive0]
DriveName=define
DriveType=1
ProtocolNum=1
Protocol0=define
```

文件夹内容如下：

完成后在 QTouch 的设备选择中就会出现自定义的选项：



协议选择中就会出现自定义协议的选项：



通过以上的操作即可完成客户自己开发的驱动集成