

Qt 编写串口通信程序全程图文讲解（一）（原创）

首先说明我们的编程环境是 windows xp 下，在 Qt Creator 中进行，如果在 Linux 下或直接用源码编写，程序稍有不同，请自己改动。

在 Qt 中并没有特定的串口控制类，现在大部分人使用的是第三方写的 `qextserialport` 类，我们这里也是使用的该类。我们可以去

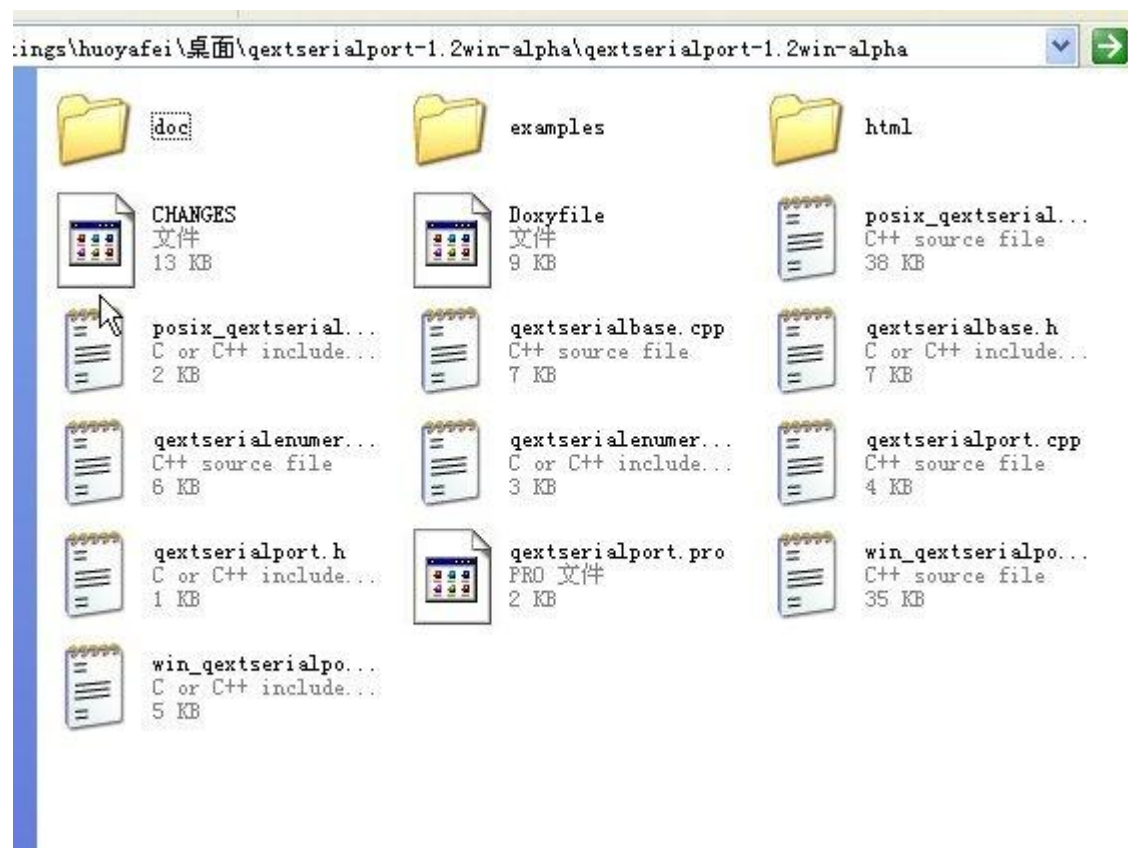
<http://sourceforge.net/projects/qextserialport/files/>

进行下载，也可以去下载论坛上的

<http://www.qtcn.org/bbs/read.php?tid=22847>

下载到的文件为：[qextserialport-1.2win-alpha.zip](#)

其内容如下图：



我们在 windows 下只需要使用其中的 6 个文件：

`qextserialbase.cpp` 和 `qextserialbase.h`，`qextserialport.cpp` 和 `qextserialport.h`，`win_qextserialport.cpp` 和 `win_qextserialport.h`

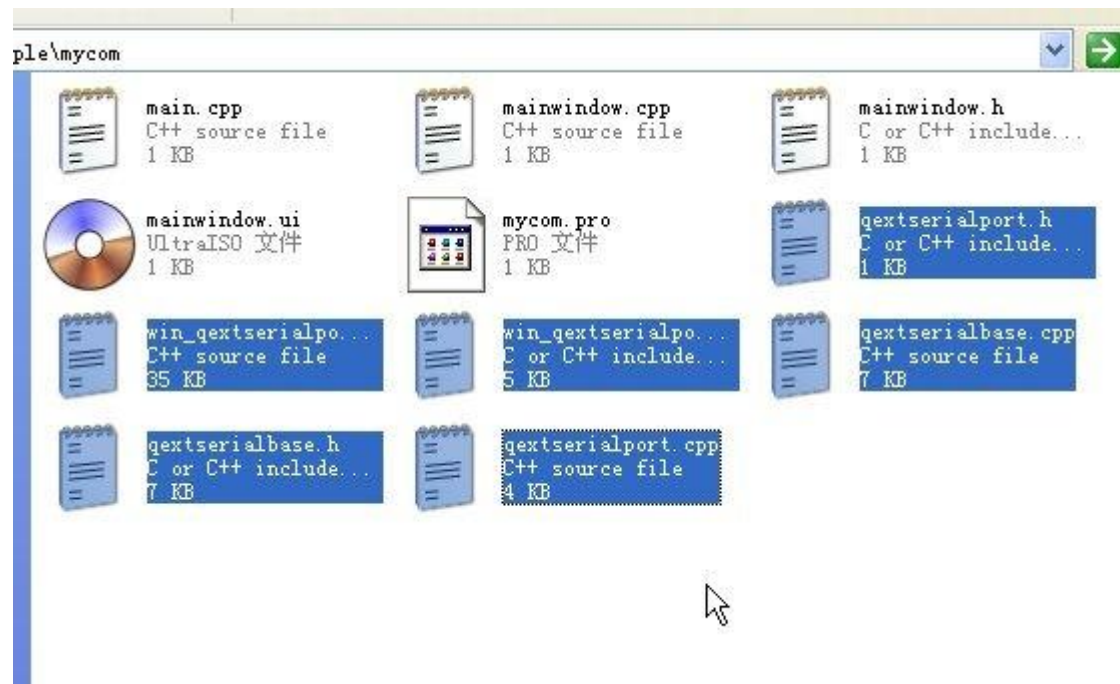
如果在 Linux 下只需将 `win_qextserialport.cpp` 和 `win_qextserialport.h` 换为 `posix_qextserialport.cpp` 和 `posix_qextserialport.h` 即可。

下面我们将讲述详细编程过程，这里我们先给出完整的程序，然后再进行逐句分析。

1.打开 Qt Creator，新建 Qt4 Gui Application，工程名设置为 **mycom**，其他使用默认选项。

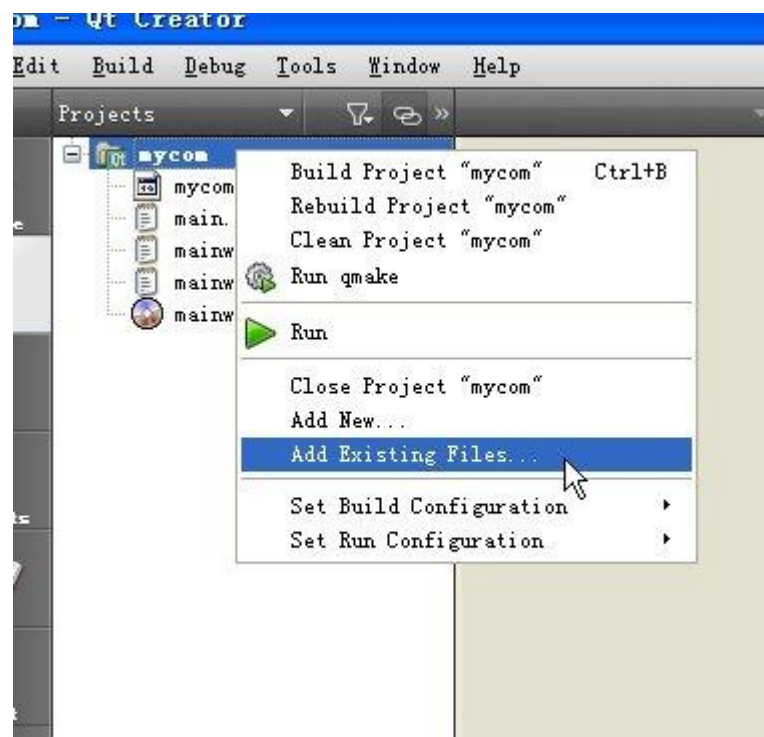
（注意：建立的工程路径不能有中文。）

2.将上面所说的 6 个文件复制到工程文件夹下，如下图。

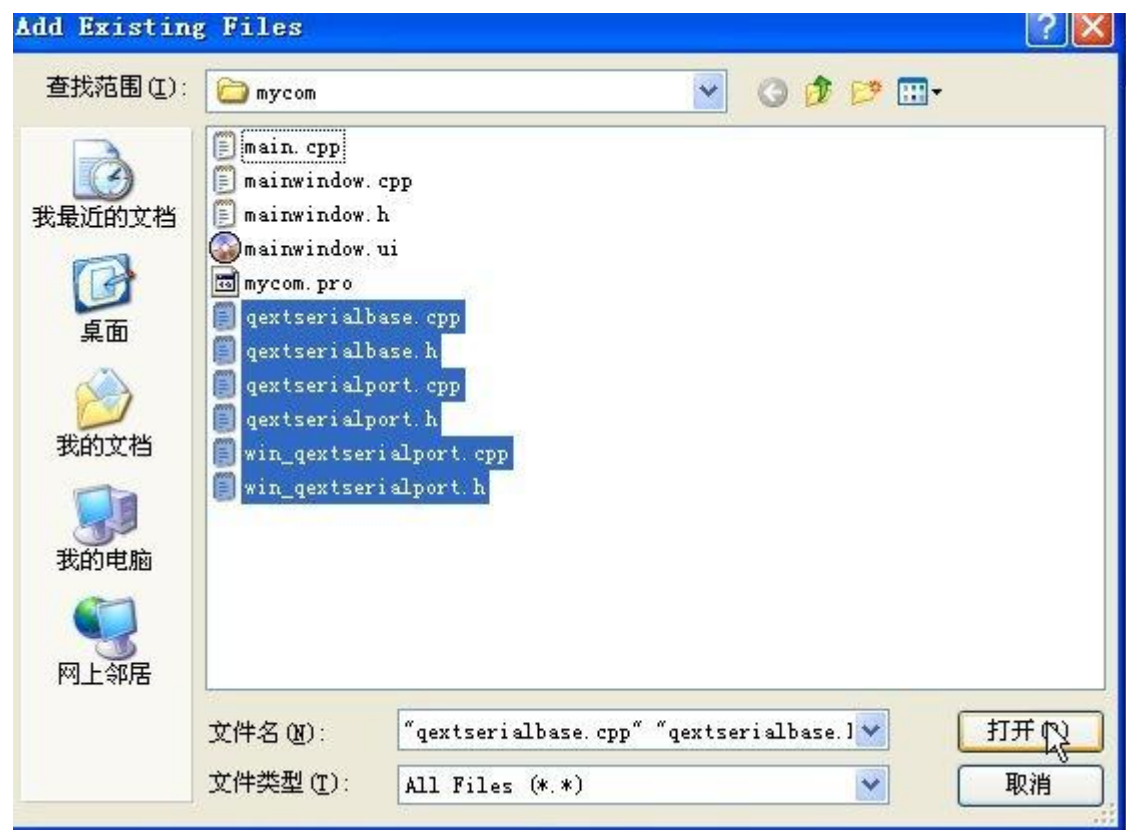


3.在工程中添加这 6 个文件。

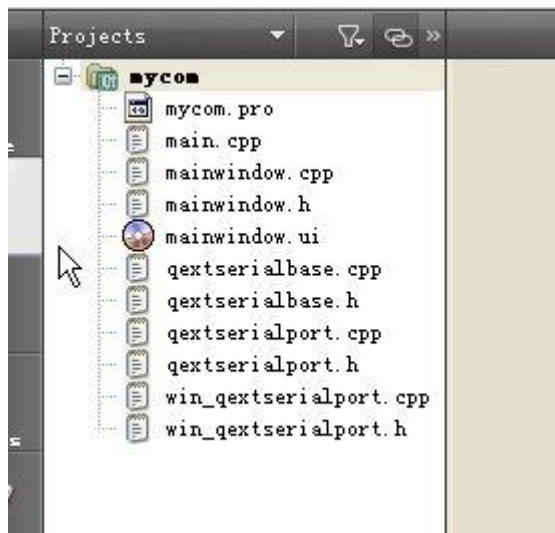
在 Qt Creator 中左侧的文件列表上，鼠标右击工程文件夹，在弹出的菜单中选择 **Add Existing Files**，添加已存在的文件。如下图：



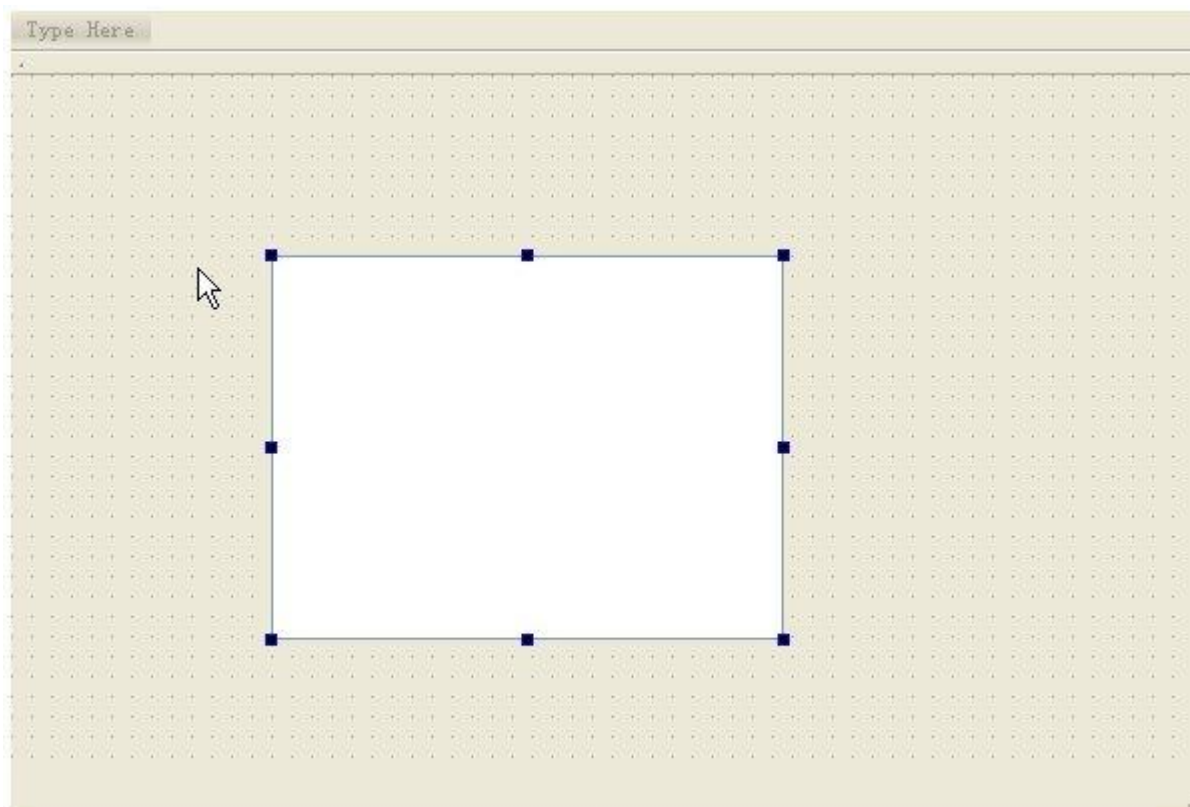
选择工程文件夹里的那 6 个文件，进行添加。如下图。



添加好后文件列表如下图所示：



4. 点击 **mainwindow.ui**，在窗口上加入一个 **Text Browser**，用来显示信息。如下图。



5. 在 **mainwindow.h** 的相应位置添加头文件 **#include "win_qextserialport.h"**，添加对象声明 **Win_QextSerialPort *myCom;**，添加槽函数声明 **void readMyCom();**，添加完后，如下图。

```

mainwindow.h
1  #ifndef MAINWINDOW_H
2  #define MAINWINDOW_H
3
4  #include <QtGui/QMainWindow>
5  #include "win_qextserialport.h"//加入头文件
6  namespace Ui
7  {
8      class MainWindow;
9  }
10
11  class MainWindow : public QMainWindow
12  {
13      Q_OBJECT
14
15  public:
16      MainWindow(QWidget *parent = 0);
17      ~MainWindow();
18
19  private:
20      Ui::MainWindow *ui;
21      Win_QextSerialPort *myCom; //声明对象
22
23  private slots:
24      void readMyCom(); //声明读串口槽函数
25
26  };
27
28  #endif // MAINWINDOW_H
29

```

6.在 mainwindow.cpp 的类的构造函数中添加如下语句。

```
MainWindow::MainWindow(QWidget *parent)
```

```
: QMainWindow(parent), ui(new Ui::MainWindow)
```

```
{
```

```
ui->setupUi(this);
```

```
struct PortSettings myComSetting =
```

```
{BAUD9600,DATA_8,PAR_NONE,STOP_1,FLOW_OFF,500};
```

```
//定义一个结构体，用来存放串口各个参数
```

```
myCom = new
```

```
Win_QextSerialPort("com1",myComSetting,QextSerialBase::EventDriven);
```

```
//定义串口对象，并传递参数，在构造函数里对其进行初始化
```

```
myCom ->open(QIODevice::ReadWrite);
```

```
//以可读写方式打开串口
```

```
connect(myCom,SIGNAL(readyRead()),this,SLOT(readMyCom()));
```

```
//信号和槽函数关联，当串口缓冲区有数据时，进行读串口操作
```

```
}
```

在下面添加 readMyCom()函数的定义，添加如下代码。

```
void MainWindow::readMyCom() //读串口函数
```

```

{
QByteArray temp = myCom->readAll();
//读取串口缓冲区的所有数据给临时变量 temp
ui->textBrowser->insertPlainText(temp);
//将串口的数据显示在窗口的文本浏览器中
}

```

添加完代码后如下图。



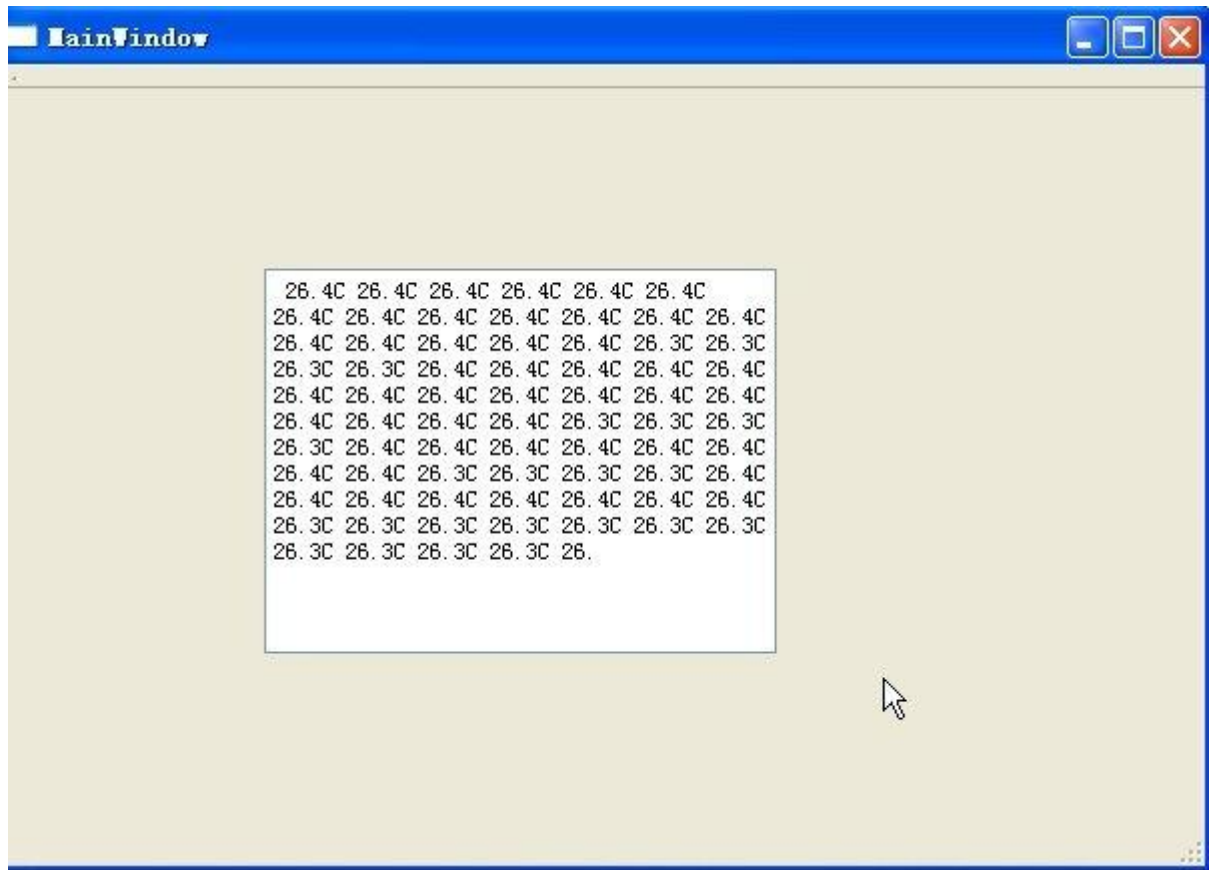
```

mainwindow.cpp  MainWindow::readMyCom()

1  #include "mainwindow.h"
2  #include "ui_mainwindow.h"
3
4  MainWindow::MainWindow(QWidget *parent)
5      : QMainWindow(parent), ui(new Ui::MainWindow)
6  {
7      ui->setupUi(this);
8      struct PortSettings myComSetting = {BAUD9600, DATA_8, PAR_NONE, STOP_1, FLOW_OFF, 500};
9      //定义一个结构体，用来存放串口各个参数
10     myCom = new Win_QextSerialPort("com1", myComSetting, QextSerialBase::EventDriven);
11     //定义串口对象，并传递参数，在构造函数里对其进行初始化
12     myCom ->open(QIODevice::ReadWrite);
13     //以可读写方式打开串口
14     connect(myCom, SIGNAL(readyRead()), this, SLOT(readMyCom()));
15     //信号和槽函数关联，当串口缓冲区有数据时，进行读串口操作
16 }
17
18 MainWindow::~MainWindow()
19 {
20     delete ui;
21 }
22 void MainWindow::readMyCom() //读串口函数
23 {
24     QByteArray temp = myCom->readAll();
25     //读取串口缓冲区的所有数据给临时变量temp
26     ui->textBrowser->insertPlainText(temp);
27     //将串口的数据显示在窗口的文本浏览器中
28 }
29

```

此时如果运行程序，已经能实现读取串口数据的功能了。我们将单片机采集的温度信息由串口传给计算机，效果如下图。



这样最简单的串口通信程序就完成了。可以看到它只需要加入几行代码即可，非常简单。

在下一篇中我们将详细分析添加的每一条语句。

Qt 编写串口通信程序全程图文讲解（二）（原创）

上一篇文章中已经介绍了实现最简单的串口接收程序的编写，这篇将对程序内容进行分析。

1. 首先应说明操作串口的流程。

步骤一：设置串口参数，如：波特率，数据位，奇偶校验，停止位，数据流控制等。

步骤二：选择串口，如 windows 下的串口 1 为“com1”，Linux 下为“ttyS0”等。

步骤三：读或写串口。

步骤四：关闭串口。

（我们上一个程序没有写串口和关闭串口的功能，打开串口也是在构造函数里完成的，因为那只是为了用最简单的方法完成串口程序的编写。在以后的文章里我们将会对它进行修改和完善。）

2. 下面我们将按照上面的操作串口的流程，讲解第一个程序的编写。

第一，我们在写程序之前，应该浏览一下那 6 个文件，大概看一下它们里面都是什么内容，各个文件各个类之间有什么联系。在 `win_qextserialport.cpp` 文件中，我们看它的最后一个构造函数，会发现，串口可以在这里进行初始化。



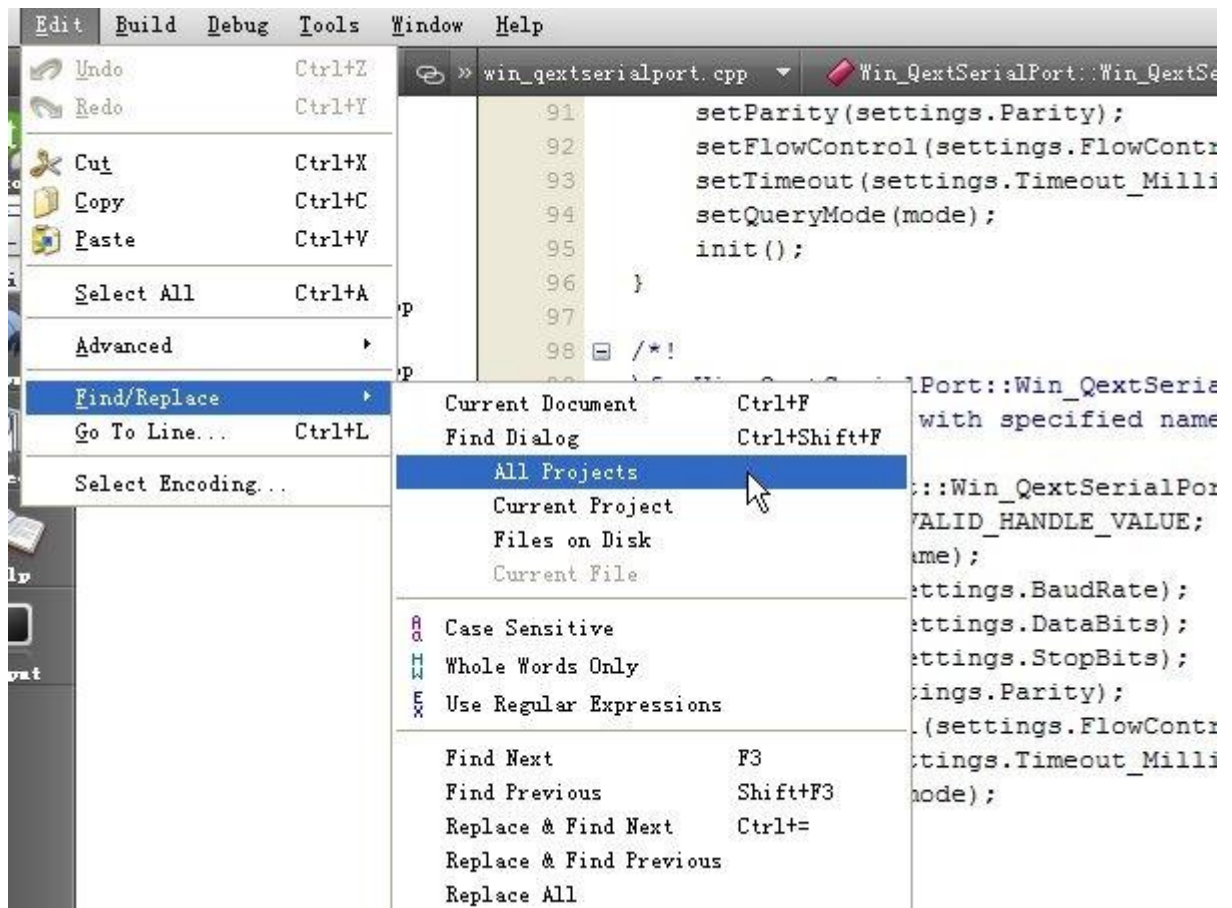
```
Win_QextSerialPort::Win_QextSerialPort(const QString & name, const PortSettings&
settings, QextSerialBase::QueryMode mode) {
```

```
Win_Handle=INVALID_HANDLE_VALUE;
setPortName(name);
setBaudRate(settings.BaudRate);
setDataBits(settings.DataBits);
setStopBits(settings.StopBits);
setParity(settings.Parity);
setFlowControl(settings.FlowControl);
setTimeout(settings.Timeout_Millisecc);
setQueryMode(mode);
init();
}
```

它共有三个参数，其中第一个参数 `const QString & name`，应该是串口的名字，是 `QString` 类型，我们可以用串口 1 即“com1”，不用过多说明。下面我们主要研究第二个和第三个参数。

第二，我们查看第二个参数的位置。

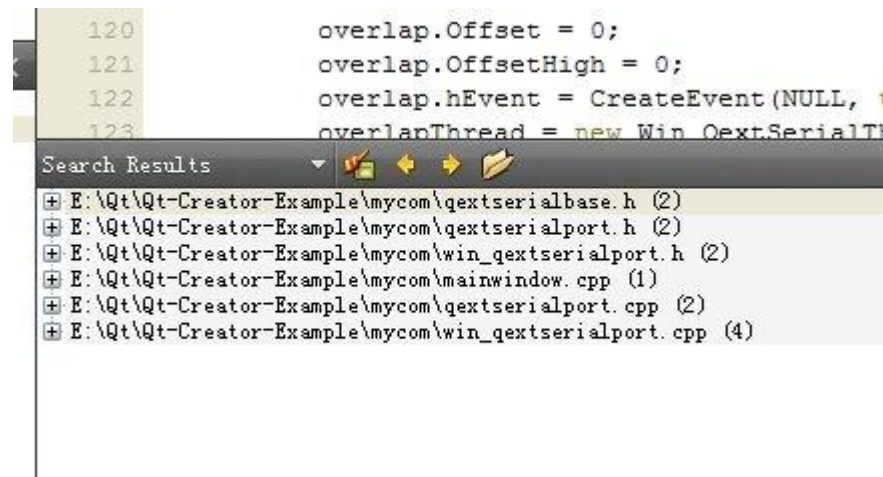
在 Qt Creator 的菜单中选择 `Edit->Find/Replace->All projects`，如下图。



在弹出的对话框中输入要查找的内容 **PortSettings**，如下图。



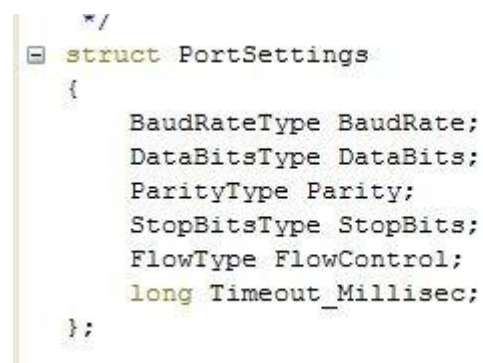
点击 **Search** 后，便能在下面显示出整个工程中所有 **PortSettings** 的位置。如下图。



我们点击第一条，可以看到在 qextserialbase.h 文件中有一个 struct PortSettings，如下图。



我们双击这一条，进入相应的文件。如下图。



struct PortSettings

```

{
    BaudRateType BaudRate;
    DataBitsType DataBits;
    ParityType Parity;

```

```
StopBitsType StopBits;  
FlowType FlowControl;  
long Timeout_Millisec;  
};
```

可以看到在这个结构体里定义了串口初始化的各个参数，而对于 **BaudRateType** 等类型的定义，我们在这个结构体的上面可以看到，它们是多个枚举变量。如下图。

```
qextserialbase.h  PortSettings  
72  enum BaudRateType  
73  {  
74      BAUD50,           //POSIX ONLY  
75      BAUD75,           //POSIX ONLY  
76      BAUD110,          //POSIX ONLY  
77      BAUD134,          //POSIX ONLY  
78      BAUD150,          //POSIX ONLY  
79      BAUD200,          //POSIX ONLY  
80      BAUD300,  
81      BAUD600,  
82      BAUD1200,  
83      BAUD1800,         //POSIX ONLY  
84      BAUD2400,  
85      BAUD4800,  
86      BAUD9600,  
87      BAUD14400,        //WINDOWS ONLY  
88      BAUD19200,  
89      BAUD38400,  
90      BAUD56000,        //WINDOWS ONLY  
91      BAUD57600,  
92      BAUD76800,        //POSIX ONLY  
93      BAUD115200,  
94      BAUD128000,       //WINDOWS ONLY  
95      BAUD256000,       //WINDOWS ONLY  
96  };  
97  
98  enum DataBitsType  
99  {  
100      DATA_5,  
101      DATA_6,  
102      DATA_7,  
103      DATA_8
```

这时我们便应该明白了，这个结构体便是实现串口参数设置的。



第三，定义串口参数。

```
BaudRateType BaudRate;
```

波特率设置，我们设置为 9600，即程序中用 BAUD9600;

```
DataBitsType DataBits;
```

数据位设置，我们设置为 8 位数据位，即 DATA_8;

```
ParityType Parity;
```

奇偶校验设置，我们设置为无校验，即 PAR_NONE;

```
StopBitsType StopBits;
```

停止位设置，我们设置为 1 位停止位，即 STOP_1;

FlowType FlowControl;

数据流控制设置，我们设置为无数据流控制，即 **FLOW_OFF**;

long Timeout_Millisec;

延时设置，我们设置为延时 500ms，即 500;

这样便写出了程序中的那句：

struct PortSettings myComSetting =

{BAUD9600,DATA_8,PAR_NONE,STOP_1,FLOW_OFF,500};

我们定义了一个结构体变量 **myComSetting**，并对其进行了初始化。



第四，设置第三个参数。

我们先按上面的方法找到它的定义位置，在 **qextserialbase.h** 中，如下图。

```
class QextSerialBase : public QIODevice
{
    Q_OBJECT

public:
    enum QueryMode {
        Polling,
        EventDriven
    };

protected:
    QMutex* mutex;
    QString port;
    PortSettings Settings;
    ulong lastErr;
    QextSerialBase::QueryMode _queryMode;

    virtual qint64 readData(char * data, qint64 maxSize)=0;
    virtual qint64 writeData(const char * data, qint64 maxSize)=0;

public:
```

可以看到查询模式也是枚举变量，有两个选项，我们选择第二个 **EventDriven**，事件驱动。

到这里，我们就可以定义 **Win_QextSerialPort** 类的变量了，就是那句：

myCom = new

Win_QextSerialPort("com1",myComSetting,QextSerialBase::EventDriven);

它完成了串口的选择和串口的初始化。



第五，写打开串口函数和读串口函数。

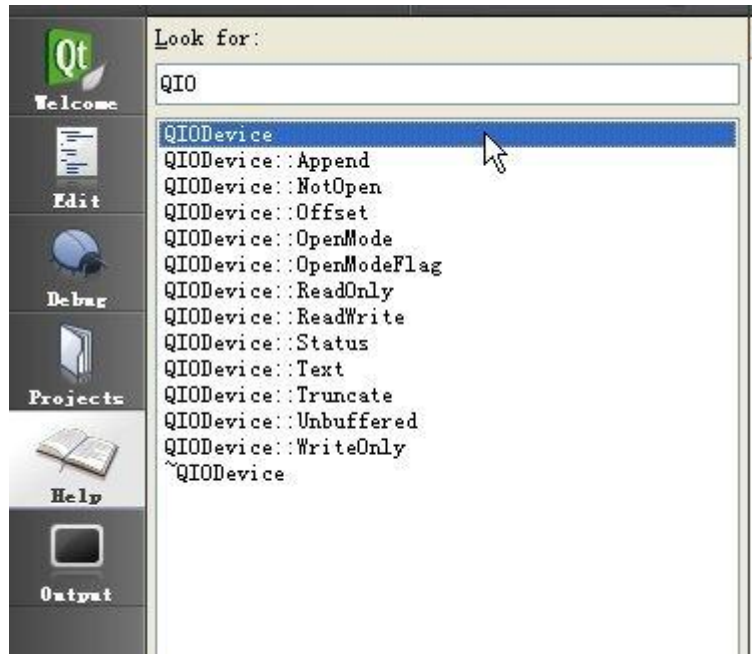
查看 **win_qextserialport.h** 文件，我们会发现 **Win_QextSerialPort** 类继承自 **QextSerialBase** 类。

```
*/
class Win_QextSerialPort: public QextSerialBase
{
```

查看 `qextserialbase.h` 文件，我们会发现 `QextSerialBase` 类继承自 `QIODevice` 类。

```
*/  
class QextSerialBase : public QIODevice  
{
```

我们在 Qt 的帮助中查看 `QIODevice` 类，如下图。



其部分内容如下图。可以看到其中有 `enum OpenModeFlag { NotOpen, ReadOnly, WriteOnly, ReadWrite, ..., Unbuffered }`, `virtual bool open (OpenMode mode)`, `QByteArray readAll ()` 等内容。

- `enum OpenModeFlag { NotOpen, ReadOnly, WriteOnly, ReadWrite, ..., Unbuffered }`

Public Functions

- `QIODevice ()`
- `QIODevice (QObject * parent)`
- `virtual ~QIODevice ()`
- `virtual bool atEnd () const`
- `virtual qint64 bytesAvailable () const`
- `virtual qint64 bytesToWrite () const`
- `virtual bool canReadLine () const`
- `virtual void close ()`
- `QString errorString () const`
- `bool getChar (char * c)`
- `bool isOpen () const`
- `bool isReadable () const`
- `virtual bool isSequential () const`
- `bool isTextModeEnabled () const`
- `bool isWritable () const`
- `virtual bool open (OpenMode mode)`
- `OpenMode openMode () const`
- `qint64 peek (char * data, qint64 maxSize)`
- `QByteArray peek (qint64 maxSize)`
- `virtual qint64 pos () const`
- `bool putChar (char c)`
- `qint64 read (char * data, qint64 maxSize)`
- `QByteArray read (qint64 maxSize)`
- `QByteArray readAll ()`

而下面的信号函数中有 `void readyRead ()`；它可以查看串口是否有新的数据传来。

- `QByteArray readAll ()`
- `qint64 readLine (char * data, qint64 maxSize)`
- `QByteArray readLine (qint64 maxSize = 0)`
- `virtual bool reset ()`
- `virtual bool seek (qint64 pos)`
- `void setTextModeEnabled (bool enabled)`
- `virtual qint64 size () const`
- `void ungetChar (char c)`
- `virtual bool waitForBytesWritten (int msecs)`
- `virtual bool waitForReadyRead (int msecs)`
- `qint64 write (const char * data, qint64 maxSize)`
- `qint64 write (const char * data)`
- `qint64 write (const QByteArray & byteArray)`
- 29 public functions inherited from `QObject`

Signals

- `void aboutToClose ()`
- `void bytesWritten (qint64 bytes)`
- `void readChannelFinished ()`
- `void readyRead ()`

所以，我们可以用这个类里的这些函数操作串口。



如程序中的语句：

```
myCom ->open(QIODevice::ReadWrite);
//我们调用了其中的 open 函数，用 ReadWrite 可读写的方式进行打开串口；
connect(myCom,SIGNAL(readyRead()),this,SLOT(readMyCom()));
//我们关联信号 readyRead(), 和自己写的槽函数 readMyCom(), 当串口有数据传来时进行
读串口操作。
void MainWindow::readMyCom() //自己写的读串口函数
{
    QByteArray temp = myCom->readAll();
    //我们调用 readAll()函数，读取串口中所有数据，在上面可以看到其返回值是 QByteArray
    类型。
    ui->textBrowser->insertPlainText(temp);
    //调用 insertPlainText()函数，是窗口上的文本浏览器中连续输出数据，而不是每次写数据
    前都清除以前的
    //数据，可以在 Qt 的帮助里查看这个函数的说明
}
```



这样我们便写完了所有的语句，最后只需要在 `mainwindow.h` 文件中加入相应的头文件，对象声明，函数声明即可。

这里需要说明的是我们一定要学会查看文件和使用帮助文档，将我们不懂得知识一点一点搞明白。

在下一篇中我们将会加入“打开串口”，“关闭串口”，“发送数据”三个按钮，将整个程序进行完善。

Qt 编写串口通信程序全程图文讲解（三）（原创）

下面的程序在第一篇文章中所写的程序上进行了一些改进。加入打开和关闭串口，发送数据等功能。

首先，加入了“打开串口”，“关闭串口”“传送数据”三个按钮，加入了一个行编辑框 **Line Edit**。它们的命名如下：

“打开串口”按钮命名为：openMyComBtn

“关闭串口”按钮命名为：closeMyComBtn

“传送数据”按钮命名为：sendMsgBtn

要传送数据的行编辑框命名为：sendMsgLineEdit

界面如下图。



第二步，在“打开串口”按钮上右击，选择 **Go to slot** 选项，然后选择 **clicked()** 选项，进入它的单击事件槽函数中，将上个程序中在构造函数里写的语句全部剪切到这里。然后加入几句按钮的状态设置语句。如下：

```
void MainWindow::on_openMyComBtn_clicked()

{
    struct PortSettings myComSetting =
    {BAUD9600,DATA_8,PAR_NONE,STOP_1,FLOW_OFF,500};
    //定义一个结构体，用来存放串口各个参数
    myCom = new
    Win_QextSerialPort("com1",myComSetting,QextSerialBase::EventDriven);
    //定义串口对象，并传递参数，在构造函数里对其进行初始化
    myCom ->open(QIODevice::ReadWrite);
    //以可读写方式打开串口
    connect(myCom,SIGNAL(readyRead()),this,SLOT(readMyCom()));
    //信号和槽函数关联，当串口缓冲区有数据时，进行读串口操作
    //
    ui->openMyComBtn->setEnabled(false); //打开串口后“打开串口”按钮不可用
    ui->closeMyComBtn->setEnabled(true); //打开串口后“关闭串口”按钮可用
    ui->sendMsgBtn->setEnabled(true); //打开串口后“发送数据”按钮可用
}
```



然后在构造函数里也添加几句按钮初始状态设置语句，如下：

```
MainWindow::MainWindow(QWidget *parent)
: QMainWindow(parent), ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    ui->closeMyComBtn->setEnabled(false); //开始“关闭串口”按钮不可用
    ui->sendMsgBtn->setEnabled(false); //开始“发送数据”按钮不可用
}
```

更改后程序如下图所示：

```
MainWindow::MainWindow(QWidget *parent)
: QMainWindow(parent), ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    ui->closeMyComBtn->setEnabled(false); //开始“关闭串口”按钮不可用
    ui->sendMsgBtn->setEnabled(false); //开始“发送数据”按钮不可用
}

MainWindow::~MainWindow()
{
    delete ui;
}

void MainWindow::readMyCom() //读串口函数
{
    QByteArray temp = myCom->readAll();
    //读取串口缓冲区的所有数据给临时变量temp
    ui->textBrowser->insertPlainText(temp);
    //将串口的数据显示在窗口的文本浏览器中
}

void MainWindow::on_openMyComBtn_clicked()
{
    struct PortSettings myComSetting = {BAUD9600, DATA_8, PAR_NONE, STOP_1, FLOW_OFF, 500};
    //定义一个结构体，用来存放串口各个参数
    myCom = new Win_QextSerialPort("com1", myComSetting, QextSerialBase::EventDriven);
    //定义串口对象，并传递参数，在构造函数里对其进行初始化
    myCom->open(QIODevice::ReadWrite);
    //以可读写方式打开串口
    connect(myCom, SIGNAL(readyRead()), this, SLOT(readMyCom()));
    //信号和槽函数关联，当串口缓冲区有数据时，进行读串口操作
    ui->openMyComBtn->setEnabled(false); //打开串口后“打开串口”按钮不可用
    ui->closeMyComBtn->setEnabled(true); //打开串口后“关闭串口”按钮可用
    ui->sendMsgBtn->setEnabled(true); //打开串口后“发送数据”按钮可用
}
```

这时运行程序，效果如下：



第三步，按上面的方法进入“关闭串口”按钮和“发送数据”按钮的单击事件的槽函数，更改如下。

```
void MainWindow::on_closeMyComBtn_clicked()    //关闭串口槽函数

{
    myCom->close();    //关闭串口，该函数在 win_qextserialport.cpp 文件中定义
    ui->openMyComBtn->setEnabled(true); //关闭串口后“打开串口”按钮可用
    ui->closeMyComBtn->setEnabled(false); //关闭串口后“关闭串口”按钮不可用
    ui->sendMsgBtn->setEnabled(false); //关闭串口后“发送数据”按钮不可用
}
```



```
void MainWindow::on_sendMsgBtn_clicked()    //发送数据槽函数
{
    myCom->write(ui->sendMsgLineEdit->text().toAscii());
    //以 ASCII 码形式将行编辑框中的数据写入串口
}
```

}

```
39 void MainWindow::on_closeMyComBtn_clicked()
40 {
41     myCom->close();
42     ui->openMyComBtn->setEnabled(true);           //关闭串口后“打开串口”按钮可用
43     ui->closeMyComBtn->setEnabled(false);          //关闭串口后“关闭串口”按钮不可用
44     ui->sendMsgBtn->setEnabled(false);             //关闭串口后“发送数据”按钮不可用
45 }
46
47 void MainWindow::on_sendMsgBtn_clicked()
48 {
49     myCom->write(ui->sendMsgLineEdit->text().toAscii()); //以ASCII码形式将数据写入串口
50 }
```

最终效果如下：

(将数据 x 发送给单片机，单片机返回 you send message is : x)



在下一篇文章中将对程序更进一步的完善。

Qt 编写串口通信程序全程图文讲解（四）终（原创）

本文一开始先讲解对程序的改进，在文章最后将要讲解一些重要问题。

第一，在窗口中加入一些组合框 **Combo Box**，它们的名称及条目如下：

串口：portNameComboBox，条目为：COM1，COM2

波特率：baudRateComboBox，条目为：9600，115200

数据位：dataBitsComboBox，条目为：8，7

校验位：parityComboBox，条目为：无，奇，偶

停止位：stopBitsComboBox，条目为：1，2

（注：在窗口上的 Combo Box 上双击，在弹出的对话框上按“+”号，可添加条目。）

改好的窗口如下所示：



第二，更改“打开串口”按钮的单击事件槽函数。

```
void MainWindow::on_openMyComBtn_clicked()
```

```
{  
    QString portName = ui->portNameComboBox->currentText(); //获取串口名  
    myCom = new Win_QextSerialPort(portName, QextSerialBase::EventDriven);  
    //定义串口对象，并传递参数，在构造函数里对其进行初始化  
    myCom ->open(QIODevice::ReadWrite); //打开串口
```



```
    if(ui->baudRateComboBox->currentText() == tr("9600")) //根据组合框内容对串口进行设置  
        myCom->setBaudRate(BAUD9600);
```



```
else if(ui->baudRateComboBox->currentText() == tr("115200"))  
myCom->setBaudRate(BAUD115200);
```



```
if(ui->dataBitsComboBox->currentText() == tr("8"))  
myCom->setDataBits(DATA_8);  
else if(ui->dataBitsComboBox->currentText() == tr("7"))  
myCom->setDataBits(DATA_7);
```



```
if(ui->parityComboBox->currentText() == tr("无"))  
myCom->setParity(PAR_NONE);  
else if(ui->parityComboBox->currentText() == tr("奇"))  
myCom->setParity(PAR_ODD);  
else if(ui->parityComboBox->currentText() == tr("偶"))  
myCom->setParity(PAR_EVEN);
```



```
if(ui->stopBitsComboBox->currentText() == tr("1"))  
myCom->setStopBits(STOP_1);  
else if(ui->stopBitsComboBox->currentText() == tr("2"))  
myCom->setStopBits(STOP_2);
```



```
myCom->setFlowControl(FLOW_OFF);  
myCom->setTimeout(500);
```



```
connect(myCom,SIGNAL(readyRead()),this,SLOT(readMyCom()));  
//信号和槽函数关联，当串口缓冲区有数据时，进行读串口操作
```

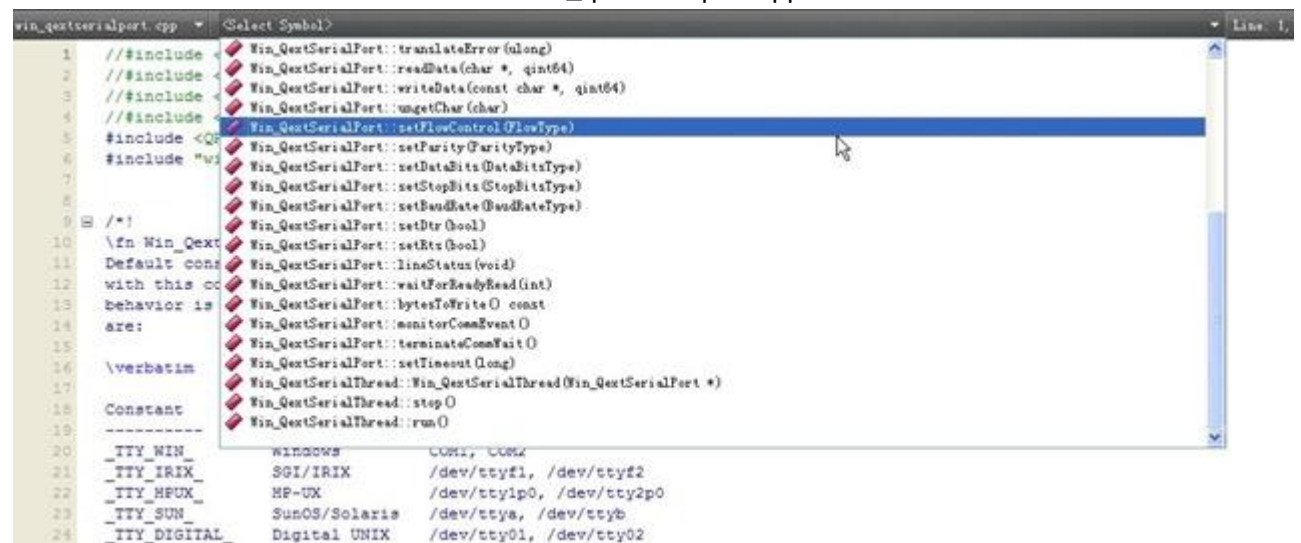


```
ui->openMyComBtn->setEnabled(false); //打开串口后“打开串口”按钮不可用  
ui->closeMyComBtn->setEnabled(true); //打开串口后“关闭串口”按钮可用  
ui->sendMsgBtn->setEnabled(true); //打开串口后“发送数据”按钮可用
```



```
ui->baudRateComboBox->setEnabled(false); //设置各个组合框不可用  
ui->dataBitsComboBox->setEnabled(false);  
ui->parityComboBox->setEnabled(false);  
ui->stopBitsComboBox->setEnabled(false);  
ui->portNameComboBox->setEnabled(false);  
}
```

这里我们先获取串口的名称，然后调用另一个构造函数对 `myCom` 进行定义，这个构造函数里没有串口的设置参数。然后打开串口。然后获取串口的设置数据，用 `setBaudRate()` 等一系列函数进行串口的设置，这些函数都在 `win_qextserialport.cpp` 文件中定义，如下图。



对于这几个函数应该很好理解，这里不再解释。在最后我们对添加的那几个组合框进行了不可用设置，使其在串口打开的情况下不能选择。

程序如下：

```
24 void MainWindow::on_openMyComBtn_clicked()
25 {
26     QString portName = ui->portNameComboBox->currentText(); //获取串口名
27     myCom = new Win_QextSerialPort(portName, QextSerialBase::EventDriven);
28     //定义串口对象，并传递参数，在构造函数里对其进行初始化
29     myCom->open(QIODevice::ReadWrite); //打开串口
30
31     if(ui->baudRateComboBox->currentText() == tr("9600")) //根据组合框内容对串口进行设置
32         myCom->setBaudRate(BAUD9600);
33     else if(ui->baudRateComboBox->currentText() == tr("115200"))
34         myCom->setBaudRate(BAUD115200);
35
36     if(ui->dataBitsComboBox->currentText() == tr("8"))
37         myCom->setDataBits(DATA_8);
38     else if(ui->dataBitsComboBox->currentText() == tr("7"))
39         myCom->setDataBits(DATA_7);
40
41     if(ui->parityComboBox->currentText() == tr("无"))
42         myCom->setParity(PAR_NONE);
43     else if(ui->parityComboBox->currentText() == tr("奇"))
44         myCom->setParity(PAR_ODD);
45     else if(ui->parityComboBox->currentText() == tr("偶"))
46         myCom->setParity(PAR_EVEN);
47
48     if(ui->stopBitsComboBox->currentText() == tr("1"))
49         myCom->setStopBits(STOP_1);
50     else if(ui->stopBitsComboBox->currentText() == tr("2"))
51         myCom->setStopBits(STOP_2);
52
53     myCom->setFlowControl(FLOW_OFF);
54     myCom->setTimeout(500);
55
56     connect(myCom, SIGNAL(readyRead()), this, SLOT(readMyCom()));
57     //信号和槽函数关联，当串口缓冲区有数据时，进行读串口操作
58
59     ui->openMyComBtn->setEnabled(false); //打开串口后“打开串口”按钮不可用
60     ui->closeMyComBtn->setEnabled(true); //打开串口后“关闭串口”按钮可用
61     ui->sendMsgBtn->setEnabled(true); //打开串口后“发送数据”按钮可用
62     ui->baudRateComboBox->setEnabled(false); //设置各个组合框不可用
63     ui->dataBitsComboBox->setEnabled(false);
64     ui->parityComboBox->setEnabled(false);
65     ui->stopBitsComboBox->setEnabled(false);
66     ui->portNameComboBox->setEnabled(false);
67 }
```

第三，更改“关闭串口”按钮单击事件的槽函数。

```
void MainWindow::on_closeMyComBtn_clicked()

{
    myCom->close();
    ui->openMyComBtn->setEnabled(true); //关闭串口后“打开串口”按钮可用
    ui->closeMyComBtn->setEnabled(false); //关闭串口后“关闭串口”按钮不可用
    ui->sendMsgBtn->setEnabled(false); //关闭串口后“发送数据”按钮不可用
    ui->baudRateComboBox->setEnabled(true); //设置各个组合框可用
    ui->dataBitsComboBox->setEnabled(true);
    ui->parityComboBox->setEnabled(true);
    ui->stopBitsComboBox->setEnabled(true);
    ui->portNameComboBox->setEnabled(true);
}
```

这里只是加入了一些使组合框在“关闭串口”按钮按下后变为可用的语句。

程序如下：

```
69 void MainWindow::on_closeMyComBtn_clicked()
70 {
71     myCom->close();
72     ui->openMyComBtn->setEnabled(true); //关闭串口后“打开串口”按钮可用
73     ui->closeMyComBtn->setEnabled(false); //关闭串口后“关闭串口”按钮不可用
74     ui->sendMsgBtn->setEnabled(false); //关闭串口后“发送数据”按钮不可用
75
76     ui->baudRateComboBox->setEnabled(true); //设置各个组合框可用
77     ui->dataBitsComboBox->setEnabled(true);
78     ui->parityComboBox->setEnabled(true);
79     ui->stopBitsComboBox->setEnabled(true);
80     ui->portNameComboBox->setEnabled(true);
81 }
```

第四，更改 **main.cpp** 文件。

```
#include <QtGui/QApplication>

#include <QTextCodec> //加入头文件
#include "mainwindow.h"
int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    QTextCodec::setCodecForTr(QTextCodec::codecForLocale());
    //使程序可处理中文
    MainWindow w;
    w.show();
    return a.exec();
}
```

因为上面的语句中用到了中文，为了能使程序识别中文，我们需要在主函数中加入这些语句。

程序如下：

```
main.cpp* main(int, char *[])
1  #include <QtGui/QApplication>
2  #include <QTextCodec>    //加入头文件
3  #include "mainwindow.h"
4
5  int main(int argc, char *argv[])
6  {
7      QApplication a(argc, argv);
8      QTextCodec::setCodecForTr(QTextCodec::codecForLocale());
9      //使程序可处理中文
10     MainWindow w;
11     w.show();
12     return a.exec();
13 }
```

第五，运行程序。

打开后程序界面如下。



正常发送 1 后如下。



设置为“奇校验”后，发送完 1 的效果如下图。



到这里，整个程序就完全写完了。

重要问题说明：

（下面所说的第一个程序是指第一篇文章中写的那个程序，第二个程序是指第三篇文章中那个程序，第三个程序是指本篇文章中所写的程序。）

问题一：第一个程序中

```
struct PortSettings myComSetting =  
{BAUD9600,DATA_8,PAR_NONE,STOP_1,FLOW_OFF,500};
```

```
myCom = new  
Win_QextSerialPort("com1",myComSetting,QextSerialBase::EventDriven);  
这两句代码如果换为下面一行：
```

```
myCom = new Win_QextSerialPort("com1",QextSerialBase::EventDriven);
```


你再运行一下程序，是不是还能用？那我们的串口设置的结构体 `myComSetting` 没有用吗？你可以把上面的结构体里的波特率由 `9600` 改为 `115200`，如果这个结构体有用，那么程序不可能再接收到数据，不过，你再运行一下程序，是这样吗？

如此看来，我们对串口进行的设置果真没用，那默认的串口设置是什么呢？我们先看下一个问题。

问题二：我们同时打开第三个程序和第二个程序。（注意：两个程序的串口不能同时打开，所以打开一

个程序的串口时要将另一个程序的串口关闭。）

我们先在第三个程序上按默认设置打开串口，发送数据 `1`。然后关闭串口，在第二个程序上打开串口，发送数据 `1`。可以看到两个程序上接受到的信息都正确。如下图。



我们关闭第二个程序上的串口，再将第三个程序上设置为奇校验，然后打开串口，发送数据 `1`，可以看到其收到的数据显示乱码。这时我们关闭第三个程序上的串口，打开第二个程序上的串口，发送数据 `1`，你会惊奇地发现，它收到的信息也是乱码。如下图。



这到底是怎么回事呢？我们也可以去网上下载其他的串口助手进行实验，也可以改变波特率进行实验。由所有的结果得出的结论只能是：我们用那个结构体作为参数传过去后，并没有

对串口进行设置，而程序运行使用的串口设置是系统以前保留的设置。那么，为什么会这样呢？我们看下面的一个问题。

问题三：我们将第三个程序中的那行代码

```
myCom ->open(QIODevice::ReadWrite);
```

放到设置串口的语句之后，

```
connect(myCom,SIGNAL(readyRead()),this,SLOT(readMyCom()));
```

这句之前，然后再运行程序。你会发现程序的串口设置功能已经不起作用了。

其实，上面的三个问题是一个问题，它的结论就是，[写串口程序时，要先打开串口再对它进行设置](#)，不然设置就不会起作用。所以，这里应该说明，第一个和第二个程序都是不太正确的，正确的方法应该是像第三个程序一样，先定义 `Win_QextSerialPort` 类对象，然后打开串口，再用那几个函数对串口进行设置。

到这里整篇文章就结束了。对于其中的一些问题也只是我个人的观点，由于水平有限，所以理解上可能会有偏差，或者错误，请广大网友批评指正。我写这篇文章的目的只是想让 **Qt** 初学者能更轻松的用 **Qt** 写出串口通信程序，及掌握 **Qt** 写程序时的一些技巧。如果你从我的文章中学到了一点知识，那么我的这篇文章就算是没有白写。

最后，如果你喜欢我的写作风格，并且初学 **Qt**，可以在我的空间查看 [Qt Creator 系列教程](#)，希望能对你的入门有所帮助。