第5章 串口操作和第三方控件的使用

CHAPTER 5

UART 接口是嵌入式、物联网开发最重要的接口之一，有着十分广泛的应用。Qt 自从 5.1 版开始提供了串口操作库，大幅降低了串口开发的难度。对于更早版本的 Qt，虽然官方没有给出串口支持库，但是有热心的开发者开发了第三方的串口库，如 QextSerialPort。虽然是第三方库，但是在功能、性能上均有良好的表现。

事实上，Qt 作为一个历史悠久的、开源的平台，有着许多优秀的第三方库，QextSerialPort 只是其中之一。Qt 的第三方库涉及图形图表、复杂控件、功能增强等许多方面。

在本章的基础知识部分，首先详细介绍了 Qt 的串口操作类 QSerialPort 的使用，然后以 QUC SDK 为例介绍了 Qt 第三方控件库的使用，最后介绍了窗口菜单的使用。

在实践案例部分，使用本章介绍的知识为 V0.1 版简易气象站程序增加了下列功能：

(1) 使用第三方控件库 QUC SDK 重新改进程序界面。

(2) 使用串口操作类通过串口读取硬件模块的数据，并对数据进行处理和显示。

(3) 增加历史数据曲线功能，方便查看历史数据。

(4) 增加窗口菜单和快捷键。

5.1 基础知识

视频讲解

5.1.1 Qt 串口通信类的使用

Qt 5 提供了串口操作相关的类 QSerialPortInfo 和 QSerialPort。使用 QSerialPortInfo 类可以检测系统的串口信息，如 COM 号、设备位置、厂商信息等。使用 QSerialPort 类可以完成串口的具体操作，如打开或关闭串口、读写数据等。在实际应用中，常常先用 QSerialPortInfo 类检测可用的串口，然后创建 QSerialPort 类对象来操作串口。要使用这两个类，应在 pro 文件中添加模块：

```
QT += serialport
```

同时在程序中包含头文件：

```
#include <QSerialPort>
#include <QSerialPortInfo>
```

下面通过一个例程介绍串口的基本操作，如图 5.1 所示。该程序具有获取串口信息、打开/关闭串口、读写串口数据等功能(参见示例代码\ch5\ch5-1QSerialPortDemo\)。

图 5.1　串口操作演示程序界面

1. 获取所有串口设备信息

要获取计算机中所有串口设备的信息，需要调用 QSerialPortInfo 类的静态成员函数 availablePorts()。该函数的原型为：

```
static QList<QSerialPortInfo> availablePorts();
```

该函数会将计算机中的所有串口设备信息保存到 QList 链表中。链表的每一个元素都是一个 QSerialPortInfo 类对象，含有串口设备的 COM 号、描述信息、序列号等信息。使用时可以通过 foreach 语句将链表中的元素取出并处理。例如，可以将“获取串口”按钮的槽函数修改如下，从而将串口信息显示在组合框控件 comboBoxPortList 中：

```
void MainWindow::on_pushButtonGetPortList_clicked()
{
    foreach (QSerialPortInfo info, QSerialPortInfo::availablePorts())
    {
        ui->comboBoxPortList->addItem(info.portName());
    }
}
```

运行程序并单击“获取串口”按钮，组合框会显示出计算机的所有串口，如图 5.2 所示。

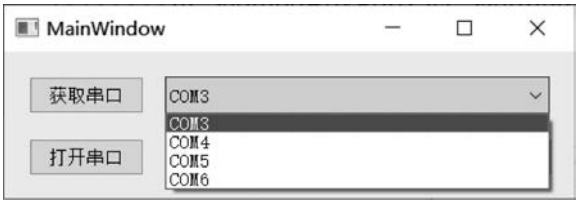


图 5.2　获取串口号并显示

如果要显示串口的描述文字，可以将代码修改为：

```
foreach (QSerialPortInfo info, QSerialPortInfo::availablePorts())
{
    ui->comboBoxPortList->addItem(QString(info.portName() + " " + info.description()));
}
```

此时的运行结果如图 5.3 所示。

图 5.3　获取串口号和描述文字并显示

2. 设置串口参数并打开串口

使用串口前需要对串口的 COM 号、波特率、奇偶校验等参数进行设置，具体步骤为：

(1) 定义 QSerialPort 类对象。

(2) 设置串口的工作参数，如 COM 号、波特率、奇偶校验等。

(3) 使用 open()函数打开串口。

(4) 确认串口已打开。

在本例中，首先为主窗口类增加成员变量：

```
private:
    QSerialPort *m_port = new QSerialPort();        //(1)定义类对象
```

然后在“打开串口”按钮的槽函数中设置串口的参数并打开串口：

```
void MainWindow::on_pushButtonOpenPort_clicked()
{
    //(2)设置串口的工作参数
    m_port->setPortName(ui->comboBoxPortList->currentText());     //选取串口
    m_port->setBaudRate(QSerialPort::Baud9600);                   //设置波特率
    m_port->setDataBits(QSerialPort::Data8);                      //设置数据位数
    m_port->setParity(QSerialPort::NoParity);                     //设置校验类型
    m_port->setStopBits(QSerialPort::OneStop);                    //设置停止位长度
    m_port->setFlowControl(QSerialPort::NoFlowControl);           //设置流控制

    //(3)打开串口
    m_port->open(QIODevice::ReadWrite);

    //(4)判断串口是否已打开
    if (m_port->isOpen())
    {
        qDebug() << "打开串口成功";
    }
```

```
    else
    {
        qDebug() << "打开串口失败";
    }
}
```

代码中的波特率(QSerialPort::Baud9600)、数据位数(QSerialPort::Data8)、校验类型(QSerialPort::NoParity)、停止位长度(QSerialPort::OneStop)、流控制(QSerialPort::NoFlowControl)都是 QSerialPort 类的枚举类型成员。它们的取值在帮助文档中有详细的介绍。isOpen()函数用于判断串口是否已打开。成功打开串口是读写数据、关闭串口、清空缓冲区等操作的前提,因此进行判断非常必要。

3. 串口的读写

QSerialPort 类的父类 QIODevice 提供了公有的 write()、read()、readAll()等函数。QSerialPort 类继承了这些函数,从而实现串口数据的读写操作。在本例中,在"写数据"按钮的槽函数中调用 write()函数向串口写数据(下面均假设串口 m_port 已打开):

```
void MainWindow::on_pushButtonWriteData_clicked()
{
    QByteArray data = "This is a test.";
    qDebug() << m_port->write(data);
}
```

调用 write()函数后,数据会写入串口缓冲区,等待硬件完成发送操作。write()函数的返回值为实际写出的字节数。在这个例子中,因为 data 变量的实际长度为 15,所以 write()函数的返回值为 15。

类似地,也可以通过 read()(读取指定长度的数据)、readLine()(读一行数据)、readAll()(读取所有数据)等函数读取串口接收缓冲区的内容。例如,在"读数据"按钮的槽函数中,可以调用 readAll()函数读取数据:

```
void MainWindow::on_pushButtonReadData_clicked()
{
    qDebug() << m_port->readAll();
}
```

4. 关闭串口

串口使用完成后,需要调用 close()函数关闭串口,从而释放资源。在本例中,在"关闭串口"的槽函数中完成这一操作:

```
void MainWindow::on_pushButtonClosePort_clicked()
{
    m_port->close();
}
```

如果尝试关闭一个没有打开的串口，则会出现错误 QSerialPort::NotOpenError。

5. 清空缓冲区

clear()函数可以根据需要清空输入/输出缓冲区中的内容。默认情况下，该函数会同时清空输入和输出缓冲区。如果使用 QSerialPort 类的枚举型变量 Direction 作为参数，则可以有选择地清空输入和输出缓冲区。枚举型变量 Direction 的定义如下：

```
enum Direction
{
    Input = 1,
    Output = 2,
    AllDirections = Input | Output
};
```

例如，下列第一行代码清空了输入缓冲区，第二行代码同时清空了输入和输出缓冲区：

```
m_port->clear(Input);
m_port->clear(AllDirections);
```

如果清空成功，则 clear()函数返回 true，否则返回 false。


视频讲解

5.1.2 Qt 的第三方控件库——QUC SDK

1. Qt 的第三方库

Qt 提供了许多功能强大的库，并保证这些库在不同操作系统下具有一致的行为。但是如果需要实现一些特殊的效果，就需要自行对库进行改进。很多热心的开发者将自己设计的库共享到了网上。下面简单介绍几个常见的 Qt 第三方库。

(1) QWT 全称是 Qt Widgets for Technical Applications。这是一个基于 LGPL 版权协议的开源项目，包含 GUI 组件和实用类。QWT 不但提供了刻度、滑块、刻度盘、指南针、温度计、旋钮等控件，还可生成各种专业图表。图 5.4 是使用 QWT 生成的对数坐标系下的图形。

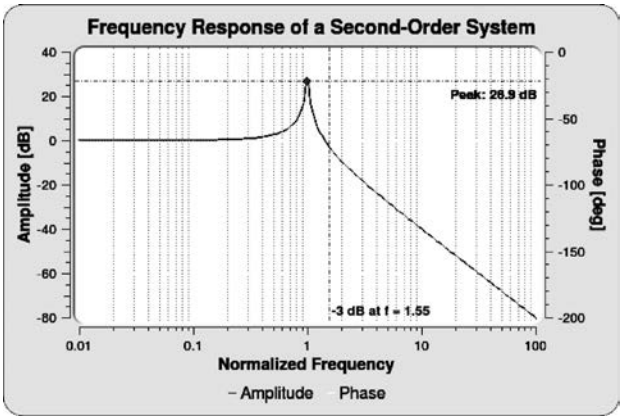


图 5.4 QWT 生成的图表

（2）QCustomPlot 是一个用于绘图和数据可视化的控件包，可以生成美观、高品质的图形，并能将图形导出为多种常见格式，如矢量 PDF 文件或 PNG、JPG、BMP 等位图。开发者对代码进行了细致的优化，能支持实时可视化应用。图 5.5 是使用 QCustomPlot 生成的曲线。

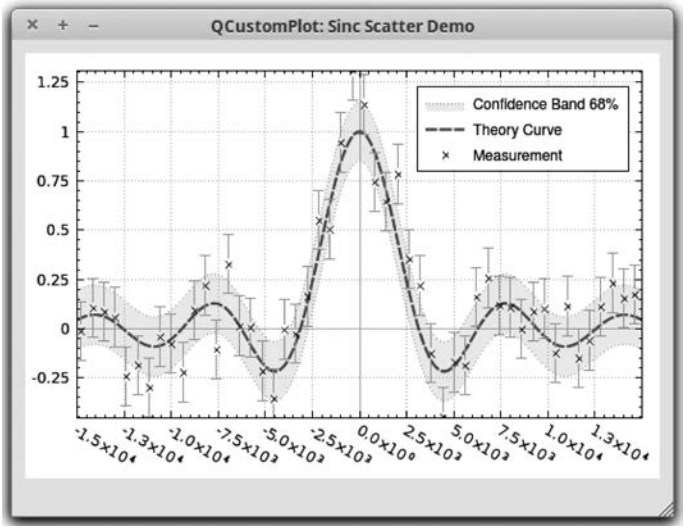


图 5.5　QCustomPlot 生成的曲线

（3）QextSerialPort 是一个广泛应用的第三方串口操作库。支持 Qt 2～Qt 5，可以运行在 Windows、Linux、macOS X、FreeBSD 等系统下。因为 Qt 4 不支持串口操作，所以 QextSerialPort 是 Qt 4 用户为数不多的选择之一。

（4）QUC SDK。SDK（Software Development Kit，软件开发工具包）是编程领域常用的名词，通常指为特定的软件框架、硬件平台、操作系统等开发应用程序时使用的开发工具集合。QUC SDK 是一套由国内开发者设计、维护的界面库，有超过 60 个精美的控件，涵盖了各种仪表盘、进度条、指南针、曲线图、标尺、温度计、导航条、导航栏、高亮按钮、滑动选择器等内容。控件的绝大多数效果只要设置几个属性即可实现。QUC SDK 支持 MinGW、MSVC、gcc 等编译器，可直接集成到 Qt Creator 中。图 5.6 展示了 QUC SDK 提供的部分控件。

由于 QUC SDK 控件的色彩简洁明快、使用方便，因此本书以 QUC SDK 为例介绍 Qt 第三方库的使用方法。

2. QUC SDK 的安装

因为 QUC SDK 的作者没有公开源代码，所以只能访问该项目的 GitHub 主页下载编译好的库文件。为了便于后续的使用，建议将整个项目下载到本地。

下面是 QUC SDK 项目中部分文件夹的结构。include 文件夹存放了 QUC SDK 库的头文件，后面会使用到这个文件夹的内容。sdkdemo 文件夹存放着示例文件。sdk_V 开头的文件夹存放着不同开发环境需要使用的库文件。snap 文件夹存放着各种控件的运行截图。

图 5.6　QUC SDK 控件示例

```
QUC SDK
├──include
├──sdkdemo
├──sdk_V20211010_mingw                    //MinGW 版库文件
|   ├──qt_5_9_9_mingw53_32
|   ├──qt_5_12_3_mingw73_32
|   ├──qt_5_12_3_mingw73_64               //最终选择的库
|   └── …
├──sdk_V20211010_msvc                     //MSVC 版库文件
|   ├──qt_5_12_3_msvc2017_32
|   ├──qt_5_14_0_msvc2017_64
|   └── …
├──sdk_V20220222_static                   //静态版库文件
|   ├──qt5_linux_gcc_32
|   ├──qt5_win_mingw_32
|   └── …
└──snap
```

QUC SDK 提供了适用于不同开发环境的库文件，使用时需要根据自己的开发环境进行选择。具体选择标准是：

（1）库文件的版本应等于或略低于 Qt 的版本。

（2）库文件的编译器版本应符合当前安装的编译器版本。如果使用了错误版本的库文件，会导致程序无法编译。

因为本书安装的 Qt 版本为 5.14.2、编译器为 MinGW 7.3.0 64bit，所以选择的库文件为 qt_5_12_3_mingw73_64。解压该库文件可以得到 quc.dll、libquc.a、qucd.dll、libqucd.a 这 4 个文件。将它们复制到 Qt 的 designer 文件夹下即可完成库文件的安装。在第 1 章介绍 Qt 的安装时，将 Qt 安装在 D:\Qt\下，这时 designer 文件夹的路径为：

```
D:\Qt\Qt5.14.2\Tools\QtCreator\bin\plugins\designer
```

完成库文件的安装后，重启 Qt Creator 就可以在控件列表看到新安装的控件了。QUC SDK 提供的所有控件都位于 Quc Widgets 栏目内，如图 5.7 所示。

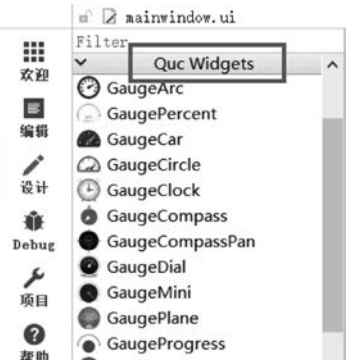


图 5.7　QUC SDK 提供的部分控件

3. QUC SDK 的使用

要在项目中使用 QUC SDK 需要进行简单的配置，具体操作步骤如下（参见示例代码\ch5\ch5-2QUCDemo\）：

（1）复制 SDK 头文件。在项目文件夹下新建子文件夹（如 SDK），然后将 QUC SDK 的 include 文件夹中的所有文件（均为头文件）复制到 SDK 文件夹中。

（2）复制库文件。将库文件解压得到的 quc.dll 和 qucd.dll 复制到 SDK 文件夹中。

（3）引用库文件。在项目的 pro 文件末尾增加如下代码：

```
INCLUDEPATH += $$PWD/SDK

CONFIG(release, debug|release){
LIBS       += -L$$PWD/SDK/ -lquc
} else {
unix {LIBS += -L$$PWD/SDK/ -lquc}
else {LIBS += -L$$PWD/SDK/ -lqucd}
}
```

代码中的 PWD 是 Present Working Directory（当前工作目录）的缩写，指项目所在的文件夹。SDK 指第（1）步中新建的 SDK 文件夹。如果使用了其他文件夹名称，则应将代码中的 SDK 替换为所使用的名称。

经过这样的配置，就可以像使用 Qt 自带控件一样使用 QUC SDK 的控件了。在示例代码中使用了 gaugeArc 和 gaugeCompassPan 两个控件，运行效果如图 5.8 所示。

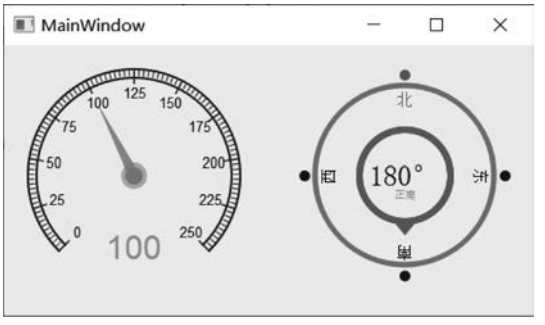


图 5.8　gaugeArc 和 gaugeCompassPan 控件的外观

QUC SDK 控件的外观几乎都可以通过属性进行修改。如果要调整 gaugeArc 控件指针的位置，只要修改控件的 value 属性即可，如图 5.9 所示。

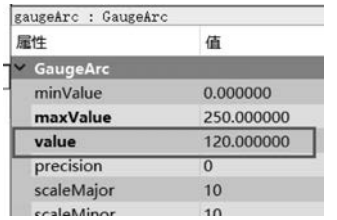


图 5.9　gaugeArc 控件的部分属性

在程序运行过程中，也可以在代码中使用控件的 setValue() 函数修改控件的指针位置，如：

```
ui->gaugeArc->setValue(110);
ui->gaugeCompassPan->setValue(190);
```

QUC SDK 并没有给出控件的文档。要全面了解控件的功能，可以查阅 SDK 文件夹中控件的头文件来了解控件支持的操作。

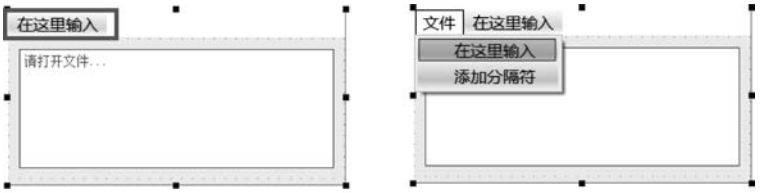

视频讲解

5.1.3　窗口菜单的使用

在程序和用户之间的交互过程中，菜单是极为常用的工具。大多数软件都会在主界面设计一套菜单。像 Microsoft Office 更是原创性地将常规菜单升级为了 Ribbon 选项卡，进一步提高了用户操作的效率。Qt 作为一款功能完善的开发平台，也能够方便地为窗口添加菜单。下面通过一个例子来学习 Qt 中菜单的使用方法（见示例代码\ch5\ch5-3MenuDemo\）。

1. 为窗口添加菜单

基于 QMainWindow 类的窗口默认带有菜单。如图 5.10(a)所示，只要双击窗口左上角的“在这里输入”，便可以进入菜单编辑状态。输入“文件”两个字后按 Enter 键，Qt Creator 会自动创建名为“文件”的菜单并展开，如图 5.10(b)所示。双击“文件”菜单中的“在这里输入”可以新增菜单项，双击“添加分隔符”可以添加一个分隔符。图 5.11 是本例最终完成的菜单。



(a) 双击“在这里输入”进入编辑状态　　(b) 新建的“文件”菜单

图 5.10　添加文件菜单的过程

2. 为菜单添加快捷键

使用快捷键可以大幅提高操作效率。例如，在 Windows 的记事本中，按 Alt+F 键会打开“文件”菜单，如图 5.12 所示。在“文件”菜单中按 N 键会新建文档，按 X 键会退出记事本。不过无论是否打开“文件”菜单，都可以通过快捷键 Ctrl+N 在记事本中新建一个文件。同样都是快捷键，为什么有的要打开菜单才能用，有的不需要打开菜单就能用呢？

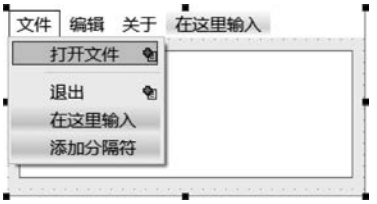


图 5.11　最终完成的菜单

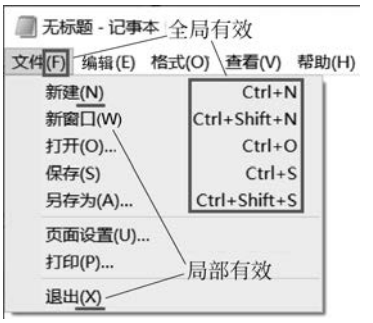


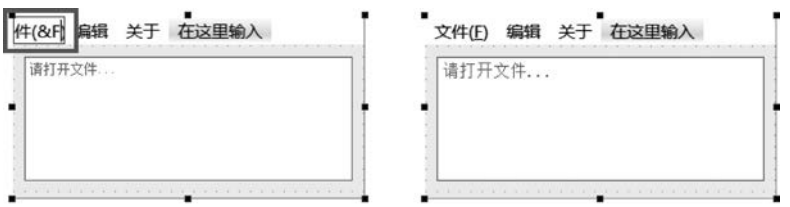
图 5.12　Windows 记事本的部分快捷键

这是因为快捷键有自己的作用范围。对记事本而言，Ctrl＋N 是全局有效的，哪怕不打开“文件”菜单也能够使用。但是“退出”菜单项的快捷键 X 是局部有效的，只有在菜单打开以后才能起作用。全局快捷键和局部快捷键的显示位置是不一样的，如图 5.12 所示。局部快捷键紧跟菜单名称或菜单项名称，常用括号包围起来。而全局快捷键位于菜单项的右侧，通常需要同时使用 2 个或 3 个按键。

在 Qt 中，可以根据需要为菜单和菜单项增加不同作用范围的快捷键。

(1) 为“文件”菜单增加全局快捷键 Alt＋F。

要为“文件”菜单添加快捷键 Alt＋F，只要在“文件”二字后面添加“(&F)”(不含引号)。因为菜单的快捷键一般都放在括号中，所以在“文件”二字后面添加“(&F)”更符合习惯，如图 5.13(a)所示。在这种情况下，菜单名会变成“文件(F)”，如图 5.13(b)所示。



(a) 处于编辑状态的菜单　　(b) 完成编辑的菜单

图 5.13　为“文件”菜单增加快捷键

(2) 为“退出”菜单项增加局部快捷键 X。

要为“退出”菜单项增加局部快捷键 X，只要在“退出”两个字后面增加“(&X)”即可，如图 5.14 所示。

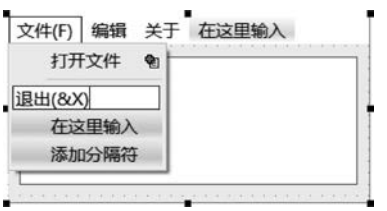


图 5.14　为“退出”菜单项增加局部有效的快捷键 X

(3) 为“退出”菜单项增加全局快捷键 Ctrl＋X。

要为菜单项增加全局快捷键，需要使用 Action Editor。Action Editor 是 Qt 的菜单编辑器，位于 Qt 设计界面下方，如图 5.15 所示。在 Action

Editor 中，详细列出了每个菜单项的名称、显示文字、快捷键、提示文字的信息，可以直接用鼠标和键盘修改。

图 5.15　Action Editor 的位置

要为“退出”菜单项增加全局快捷键 Ctrl＋X，只需要在 Action Editor 中双击“退出”按钮对应的行，打开“编辑动作”对话框，如图 5.16 所示。单击 Shortcut 文本框，然后同时按键盘上的 Ctrl 键和 X 键，从而完成快捷键录入。图 5.17 是添加了快捷键的菜单截图。

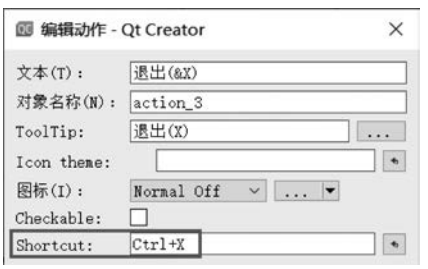


图 5.16　使用 Action Editor 为“退出”菜单项增加全局快捷键 Ctrl＋X

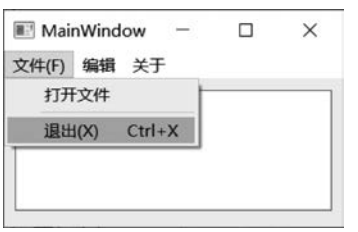


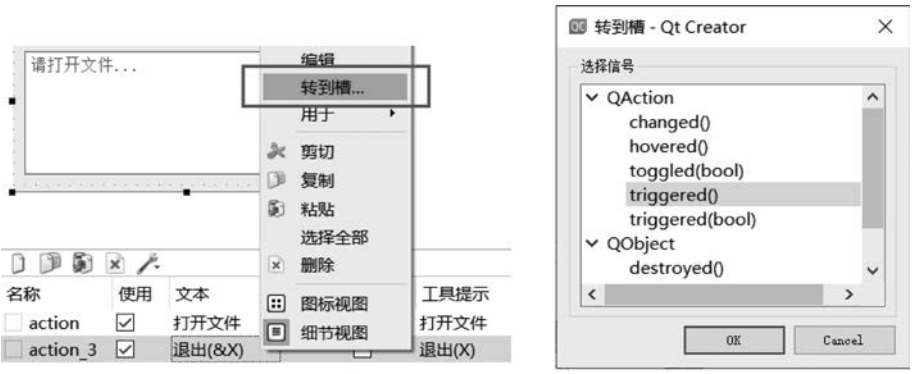
图 5.17　添加了快捷键的菜单截图

3. 为菜单增加动作

至此，菜单项已经有了快捷键。但是按这些快捷键并没有任何反应。这是因为还没有为菜单项增加对应的动作，也就是没有完成菜单项的槽函数。

要为菜单增加动作，可以在 Action Editor 中右击菜单项对应的行，在弹出的菜单中选择“转到槽”，如图 5.18 所示。在“转到槽”对话框中选择 triggered()（即菜单项被触发），并单击 OK 按钮。系统会自动定位到菜单项的槽函数 on_action_3_triggered()。因为退出按钮的功能是关闭窗口，所以只需要调用 close()函数即可：

```
void MainWindow::on_action_3_triggered()
{
    this->close();
}
```



(a) Action Editor的右键菜单　　(b) “转到槽”对话框

图 5.18　“转到槽”菜单项及对话框

再次编译、运行程序，就可以通过快捷键关闭程序了。

5.2　实践案例：简易气象站程序 V0.2 的实现


视频讲解

下面使用本章所学知识对 V0.1 版简易气象站程序进行改进，包括使用 QUC SDK 更新程序界面，增加串口操作和数据读取功能，增加菜单和快捷键。这一版的程序代码见“示例代码\ch5\ch5-4SimpleWeatherStationV0.2\”。

5.2.1　使用 QUC SDK 升级程序界面

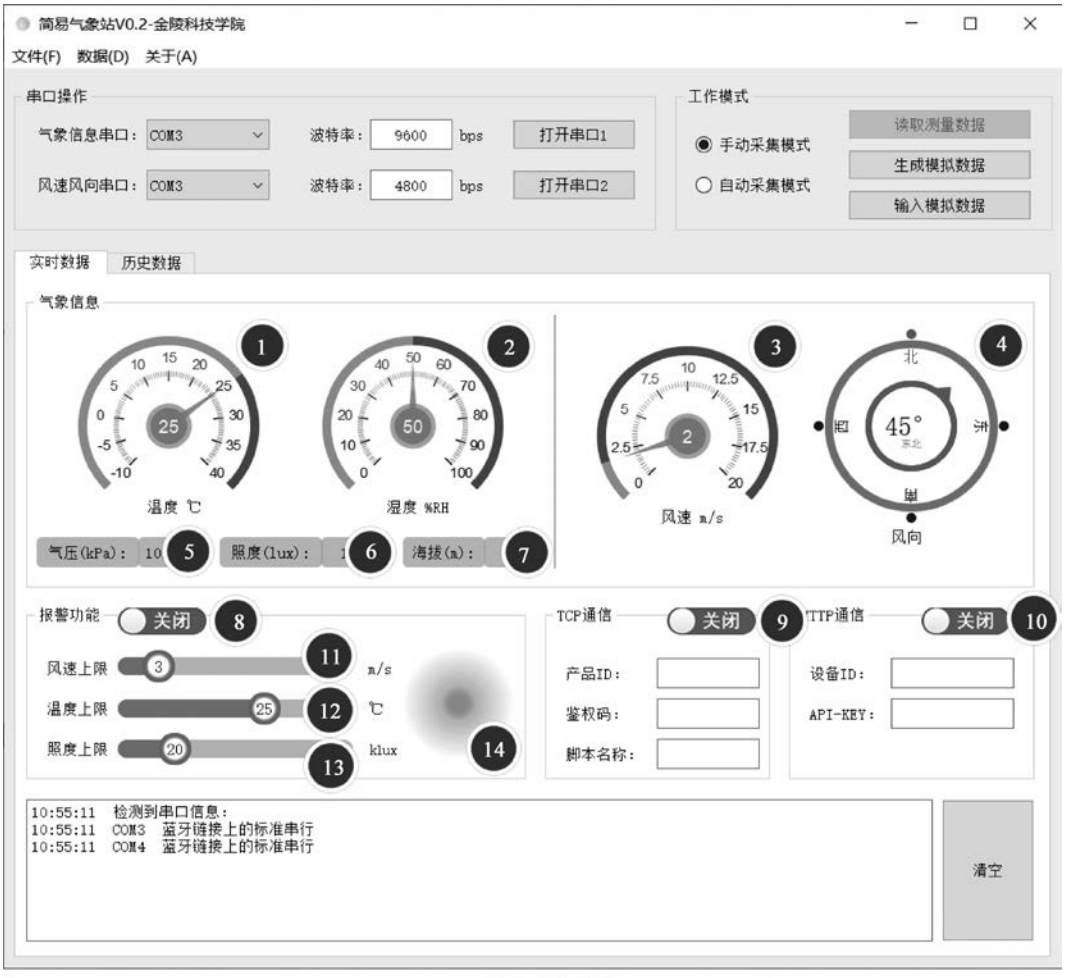
使用 QUC SDK 升级程序界面的步骤相对简单，只要用新的控件替换掉默认控件即可。图 5.19 是升级完成的界面。相对于 V0.1 版程序，主要使用了 QUC SDK 的 GaugeSimple、GaugeCompassPan、NavLabel、ImageSwitch、XSlider、LightPoint、WaveChart 这几种控件，并用 Tab Widget 控件增加了多页面显示功能。表 5.1 给出了 V0.2 版程序中使用的 QUC SDK 控件的信息。界面中使用的 Qt 自带控件则不再赘述。

表 5.1　界面中使用的 QUC SDK 控件

序号	控件类型	控件名称	序号	控件类型	控件名称
1	GaugeSimple	gaugeSimpleTemperature	5	NavLabel	navLabelPressure
2	GaugeSimple	gaugeSimpleHumidity	6	NavLabel	navLabelIllumination
3	GaugeSimple	gaugeSimpleWindSpeed	7	NavLabel	navLabelAltitude
4	GaugeCompassPan	gaugeCompassPanWindDirection	8	ImageSwitch	imageSwitchAlarm

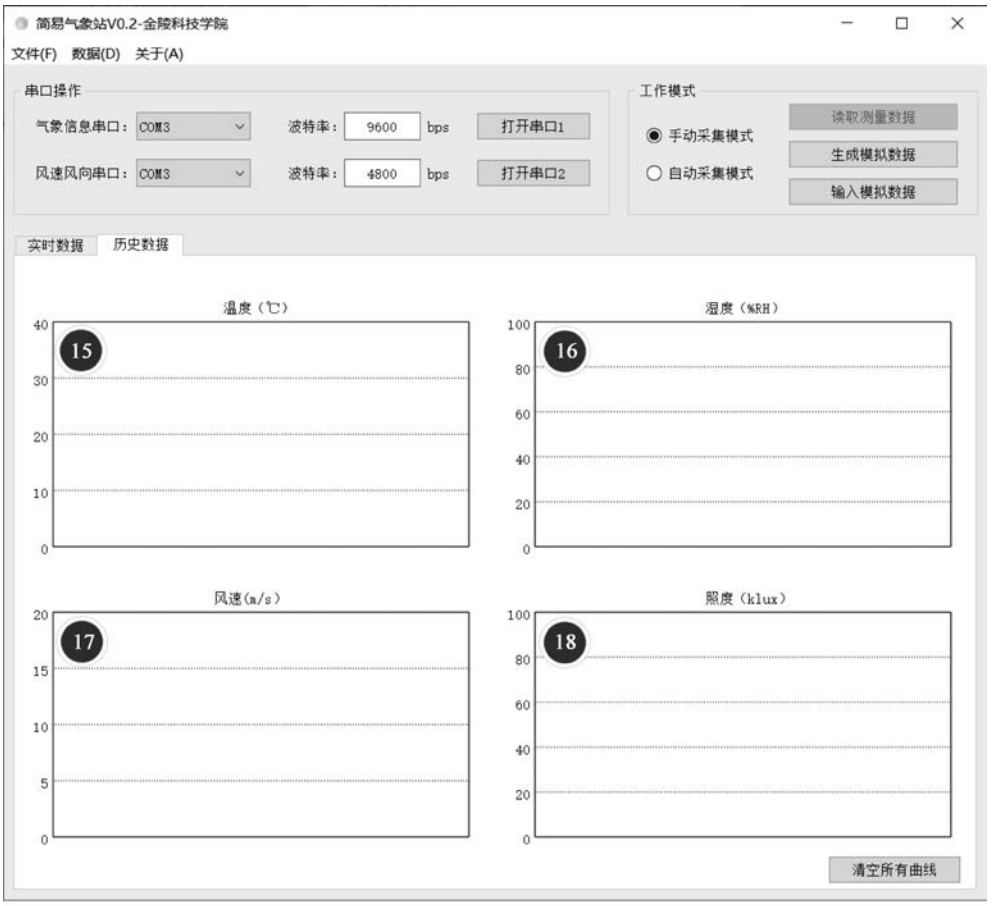
续表

序号	控件类型	控件名称	序号	控件类型	控件名称
9	ImageSwitch	imageSwitchHTTP	14	LightPoint	lightPoint
10	ImageSwitch	imageSwitchTCP	15	WaveChart	waveChartTemperature
11	XSlider	xsliderWindSpeedLimit	16	WaveChart	waveChartHumidity
12	XSlider	xsliderTemperatureLimit	17	WaveChart	waveChartWindSpeed
13	XSlider	xsliderIlluminationLimit	18	WaveChart	waveChartIllumination



(a) 气象信息界面

图 5.19 使用 QUC SDK 更新后的界面



(b) 历史数据界面

图 5.19　（续）

5.2.2　串口操作功能的实现

在 V0.1 版程序中已经设计好了串口操作区的界面。下面利用本章所学的知识完成这部分的功能。

1. 串口信息的读取和显示

在这一版程序中为主窗口类增加了读取串口信息的函数 updateSerialInfo()。在主窗口的构造函数中调用该函数，从而在启动后立即读取串口信息。在后续章节中，还会对该函数做进一步改进，从而实现实时更新串口信息的功能。updateSerialInfo()函数的代码如下：

```
void MainWindow::updateSerialInfo()
{
    ui->comboBoxUart1->clear();
    ui->comboBoxUart2->clear();
```

```
    printLog("检测到串口信息:");

    foreach (const QSerialPortInfo &info, QSerialPortInfo::availablePorts())
    {
        ui->comboBoxUart1->addItem(info.portName());
        ui->comboBoxUart2->addItem(info.portName());
        printLog(info.portName(), info.description());
    }

    ui->comboBoxUart1->model()->sort(0);   //对串口列表进行排序
    ui->comboBoxUart2->model()->sort(0);
}
```

图 5.20 是程序运行后自动检测并显示的串口信息。

图 5.20　程序运行后自动检测并显示的串口信息

2. 定义串口类对象

在进行串口操作前，首先为主窗口类增加两个 QSerialPort 类对象，分别对应气象信息串口和风速风向串口：

```
private:
    QSerialPort *m_serialWeather;
    QSerialPort *m_serialWind;
```

同时在构造函数中为两个对象申请内存：

```
    m_serialWeather = new QSerialPort();
    m_serialWind = new QSerialPort();
```

3. 打开和关闭串口

程序使用两个按钮分别控制两个串口的打开和关闭，并用主窗口类的成员变量 m_nSerialWeatherOpenedFlag和 m_nSerialWindOpenedFlag 表示串口的打开状态。“打开串口 1”按钮（控件名为 pushButtonOpenUart1）的槽函数代码如下：

```
1   void MainWindow::on_pushButtonOpenUart1_clicked()
2   {
```

```
3       if (m_nSerialWeatherOpenedFlag == 0)
4       {
5            m_serialWeather->setPortName(ui->comboBoxUart1->currentText());
6            m_serialWeather->setBaudRate(ui->lineEditBaudRate1->text().toInt());
7            m_serialWeather->setDataBits(QSerialPort::Data8);
8            m_serialWeather->setParity(QSerialPort::NoParity);
9            m_serialWeather->setStopBits(QSerialPort::OneStop);
10           m_serialWeather->setFlowControl(QSerialPort::NoFlowControl);
11           m_serialWeather->open(QIODevice::ReadWrite);
12
13           if (m_serialWeather->isOpen())
14           {
15              printLog("串口 1 已打开", ui->comboBoxUart1->currentText());
16              ui->pushButtonOpenUart1->setText("关闭串口 1");
17              m_nSerialWeatherOpenedFlag = 1;
18           }
19           else
20           {
21              printLog("串口 1 打开失败");
22           }
23       }
24       else
25       {
26           printLog("串口 1 已关闭");
27           ui->pushButtonOpenUart1->setText("打开串口 1");
28           m_nSerialWeatherOpenedFlag = 0;
29           m_serialWeather->close();
30       }
31     }
32  }
```

在上述代码中，首先通过标志位 m_nSerialWeatherOpenedFlag 判断串口的打开状态（第 3 行）。如果未打开，则进行打开操作（第 5～11 行），并判断打开是否成功（第 13～22 行）。如果已打开，则关闭串口（第 26～29 行）。

“打开串口 2”按钮（控件名为 pushButtonOpenUart2）的槽函数与上述代码几乎完全相同，此处不再赘述。

5.2.3　GY-39 模块的数据读取和处理

要想正确获取 GY-39 模块测量的气象信息，需要经过串口数据读取、数据完整性校验、数据解析 3 个步骤。在程序中，为 ClassGY39 类增添了 3 个成员函数 readSerialData()、verifySerialData()、parseSerialData()分别实现这些功能。

1. 串口数据读取

readSerialData()函数用于读取串口数据，并对读取的数据进行初步处理。为了让它能配合不同的串口进行工作，将串口类对象的指针作为形参传入。该函数的代码如下：

```
1   int ClassGY39::readSerialData(QSerialPort * serialPort)
2   {
3       QByteArray qbaWeatherData = serialPort->readAll();
4       if (qbaWeatherData.length() % 24 != 0 || qbaWeatherData.length() == 0)
                                                                    //检查数据长度
5       {
6           return -1;                                              //数据长度不正确
7       }
8       qbaWeatherData = qbaWeatherData.right(24);                  //取最后一组数据
9
10      if (verifySerialData(qbaWeatherData) != 0)                  //校验数据
11      {
12          return -2;                                              //数据校验错误
13      }
14
15      parseSerialData(qbaWeatherData);                            //解析数据
16      return 0;
17  }
```

GY-39 模块每秒报告一次气象信息，每组气象信息的长度为 24 字节。由于读取气象信息的频率不确定，因此串口缓冲区内可能会同时包含多组气象信息。为了获取其中最新的一组信息，首先要判断接收到的数据是否是 24 字节的整数倍（第 4 行），然后取出最后 24 字节进行后续处理（第 8 行）。第 10 行调用函数 verifySerialData()对读取的数据进行校验。如果校验结果正确，则调用函数 parseSerialData()进行数据解析（第 15 行）。

2. 数据完整性校验

GY-39 模块的数据校验的步骤是将所有的数据相加，取结果的低 8 位作为校验位。在下列代码中使用 foreach 语句分别对光照强度数据以及温度和湿度数据进行了校验。

```
int ClassGY39::verifySerialData(QByteArray qbaSerialData)
{
    unsigned int nSum = 0;                                          //保存求和结果
    foreach (char cTmp, qbaSerialData.left(8))                      //开始校验光照强度数据
    {
        nSum += cTmp;                                               //进行累加操作
    }

    if ((nSum % 256) != (unsigned char)qbaSerialData.at(8))         //求和结果是否等于校验位
    {
        return -1;                                                  //不等于则返回-1,等于则继续
    }

    nSum = 0;                                                       //求和结果清零
    foreach (char cTmp, qbaSerialData.right(15).left(14))           //开始校验气象数据
    {
        nSum += cTmp;                                               //进行累加操作
```

```
    }

    if ((nSum % 256) != (unsigned char)qbaSerialData.at(23)) //求和结果是否等于校验位
    {
        return -2;                                          //不等于则返回-2,等于则继续
    }
    return 0;
}
```

3. 数据解析

parseSerialData()函数负责将原始数据解析成具体的测量结果(计算公式见 2.2.4 节)。需要特别注意的是,读取的串口数据需要存储在 QByteArray 变量中,但是 QByteArray 会将其中的每个元素都作为有符号数处理。在某些特定的情况下,计算结果会出现问题。以下列原始测量数据为例(下画线标出的是照度数据)。

```
5A 5A 15 04 00 2E 12 EC F9 5A 5A 45 0A 0B 58 00 9A 9E 4E 14 13 00 00 13
```

由于照度的测量结果总是非负的(气压、湿度等亦同),因此数据的每一字节都代表了一个正数。正常情况下,实际的照度值应该为:

$$
\begin{aligned}
&(0x00 \ll 24) + (0x2E \ll 16) + (0x12 \ll 8) + 0xEC \\
&= (0000\ 0000B \ll 24) + (0010\ 1110B \ll 16) + (0001\ 0010B \ll 8) + 1110\ 1100B \\
&= 0 + 3\ 014\ 656 + 4\ 608 + 236 \\
&= 3\ 019\ 500
\end{aligned}
$$

但是 QByteArray 会将每一字节都作为有符号数处理。这就会导致计算结果变为(带下画线的是 QByteArray 误认的符号位):

$$
\begin{aligned}
&(0x00 \ll 24) + (0x2E \ll 16) + (0x12 \ll 8) + 0xEC \\
&= (\underline{0}000\ 0000B \ll 24) + (\underline{0}010\ 1110B \ll 16) + (\underline{0}001\ 0010B \ll 8) + \underline{1}110\ 1100B \\
&= 0 + 3\ 014\ 656 + 4\ 608 - 20 \\
&= 3\ 019\ 244 \neq 3\ 019\ 500
\end{aligned}
$$

要解决这一问题,可以将 QByteArray 转换为 C 语言中常用的 unsigned char 类型进行计算。

但是对于温度、海拔高度数据,情况又有所不同。这两个数据的测量结果既可以是正数,也可以是负数。所以这两个数据的原始测量结果的最高字节是有符号的,其余字节是无符号的。在计算数据的过程中,不同字节应当按不同的方式来计算。

按照上面的思路,可以得到 parseSerialData()的代码:

```
int ClassGY39::parseSerialData(QByteArray qbaSerialData)
{
    unsigned char *cData = (unsigned char *)qbaSerialData.data();  //转换为无符号数组
```

```
    int nIllumi = (cData[4] << 24) + (cData[5] << 16) + (cData[6] << 8) + cData[7];
                  //光照数据恒正,采用无符号的数据进行计算
    float fTemp = ((qbaSerialData[13] << 8) + cData[14]) / 100.0;
                  //温度数据高字节采用有符号的数据计算,低字节采用无符号的数据计算
    float fPre = ((cData[15] << 24) + (cData[16] << 16) + (cData[17] << 8) + cData[18]) /
100.0 / 1000.0;
    int nHum = ((cData[19] << 8) + cData[20]) / 100.0;
    int nAlti = (qbaSerialData[21] << 8) + cData[22];

    setIllumination(nIllumi);
    setTemperature(fTemp);
    setPressure(fPre);
    setHumidity(nHum);
    setAltitude(nAlti);

    return 0;
}
```

5.2.4 PR-3000 模块的数据读取和处理

PR-3000 模块的情况与 GY-39 模块的情况稍有不同。由于 PR-3000 模块的数据读取流程较为复杂，涉及 Modbus 协议问询帧和应答帧的交替处理，因此只为 ClassPR3000 类增加了两个成员函数，即负责数据读取和解析的 readSerialData() 和负责 CRC16 校验的 crc16Verify()。

1. 串口数据读取和解析

首先为 ClassPR3000 类增加两个 QByteArray 类型的成员变量：m_qbaRequestWS 和 m_qbaRequestWD，分别用于存储风速和风向问询帧：

```
private:
    QByteArray m_qbaRequestWS = QByteArray::fromHex("010300000001840A");
    QByteArray m_qbaRequestWD = QByteArray::fromHex("020300000002C438");
```

然后完成函数 readSerialData()：

```
1   int ClassPR3000::readSerialData(QSerialPort * serialPort)
2   {
3       QEventLoop eventLoop;                          //定义事件循环
4
5       serialPort->write(m_qbaRequestWS);             //风速部分发送问询帧
6       QTimer::singleShot(200, &eventLoop, SLOT(quit()));
7       eventLoop.exec();
8       QByteArray qbaWSData = serialPort->readAll();
9
10      if (0 != crc16Verify(qbaWSData.left(5), qbaWSData.right(2)))
11      {
```

```
12          return -1;
13      }
14
15      float fWindSpeed = (qbaWSData.at(3) * 256 + qbaWSData.at(4)) / 10.0;
16      setWindSpeed(fWindSpeed);
17
18      //风向部分与上述风速部分类似,此处略去
19      return 0;
20  }
```

因为该函数涉及 Modbus 协议的通信,所以流程较为复杂。代码第 5 行发送了风速模块的问询帧。由于模块在执行指令时需要一定的时间,因此在第 6 行和第 7 行添加了一个 EventLoop(事件循环)。通过事件循环可以临时阻塞当前程序,并在满足一定条件后继续运行程序。关于事件循环的细节将在第 6 章讲解。当风速模块返回了风速信息后,事件循环自动退出,并运行 readAll()函数读取缓冲区内容(第 8 行)。代码的第 10～13 行是对风速信息进行 CRC16 校验。第 15 行和第 16 行对校验后的数据进行解析。风向部分的代码与风速部分的代码十分类似,可以参阅示例代码。

2. CRC 校验的实现

CRC 算法的思路在第 1 章中已经做了介绍。在实际应用中,CRC16 的实现有查表法和实时计算法两种。查表法是提前制作好 CRC16 的参考表格,校验时根据数据和参考表格经过简单计算迅速得到结果。实时计算法则是严格按照 CRC16 的算法,根据实际数据进行计算。以下是 CRC16 实时计算法的代码:

```
int ClassPR3000::crc16Verify(QByteArray qbaData, QByteArray qbaCheckSum)
{
   quint16 data8, crc16 = 0xFFFF;

   for (int i = 0; i < qbaData.size(); i++)
   {
      data8 = qbaData.at(i) & 0x00FF;
      crc16 ^= data8;
      for (int j = 0; j < 8; j++)
      {
         if (crc16 & 0x0001)
         {
            crc16 >>= 1;
            crc16 ^= 0xA001;
         }
         else
         {
            crc16 >>= 1;
         }
      }
   }
```

```
    crc16 = (crc16 >> 8) + (crc16 << 8);

    if ((crc16 / 256) == (unsigned char)qbaCheckSum.at(0))
    {
        if ((crc16 % 256) == (unsigned char)qbaCheckSum.at(1))
        {
            return 0;
        }
    }
    return 1;
}
```

5.2.5 界面更新函数的进一步修改

在第 4 章中已经实现了界面更新函数 updateUI()。本章因为更换了控件，所以需要对该函数进行更新。更新后的代码如下：

```
1   void MainWindow::updateUI()
2   {
3       ui->gaugeSimpleHumidity->setValue(m_GY39Device->getHumidity());
4       ui->gaugeSimpleTemperature->setValue(m_GY39Device->getTemperature());
5       ui->gaugeSimpleWindSpeed->setValue(m_PR3000Device->getWindSpeed());
6       ui->gaugeCompassPanWindDirection->setValue(m_PR3000Device->getWindDirection());
7
8       ui->navLabelPressure->setText(QString::number(m_GY39Device->getPressure(), 'f', 3));
9       ui->navLabelIllumination->setText(QString::number(m_GY39Device->getIllumination()));
10      ui->navLabelAltitude->setText(QString::number(m_GY39Device->getAltitude()));
11
12      ui->waveChartTemperature->addData(m_GY39Device->getTemperature());
13      ui->waveChartHumidity->addData(m_GY39Device->getHumidity());
14      ui->waveChartIllumination->addData(m_GY39Device->getIllumination() / 1000);
15      ui->waveChartWindSpeed->addData(m_PR3000Device->getWindSpeed());
16  }
```

在第 8 行中，由于气压以 Pa 为单位存放在变量中，但是在显示过程中希望以 kPa 为单位进行显示，因此按照浮点数的形式进行转换，并保留 3 位小数。历史记录页面中的控件类型为 WaveChart。调用该类控件的 addData()函数可以将数据添加到控件中（第 12～15 行）。

5.2.6 手动读取数据的实现

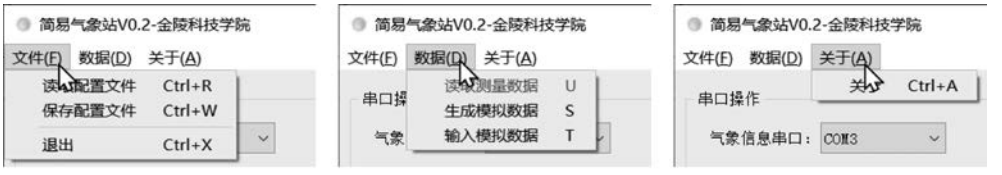
单击“读取测量数据”按钮后，程序会读取硬件模块的数据并显示在界面上，还会根据报警开关的状态确定是否开启报警功能。要实现这些功能，需要在按钮的槽函数中加入如下内容：

```
1   void MainWindow::on_pushButtonGetHardwareData_clicked()
2   {
3       int nGY39DataValidFlag = -1, nPR3000DataValidFlag = -1;
4       if (m_nSerialWeatherOpenedFlag == 1)
5       {
6           nGY39DataValidFlag = m_GY39Device->readSerialData(m_serialWeather);
7       }
8
9       if (m_nSerialWindOpenedFlag == 1)
10      {
11          nPR3000DataValidFlag = m_PR3000Device->readSerialData(m_serialWind);
12      }
13
14      if ((nGY39DataValidFlag == 0) || (nPR3000DataValidFlag == 0))
15      {
16        updateUI();
17        if (ui->imageSwitchAlarm->getChecked())
18        {
19            alarm();
20        }
21  }
```

在该代码中，第 6 行和第 11 行依次读取 GY-39 模块和 PR-3000 模块的数据，并使用变量 nGY39DataValidFlag 和 nPR3000DataValidFlag 记录数据是否有效。如果有效，则变量的取值为 0。只要有一个模块的数据读取成功，程序就可以更新界面（第 16 行）。紧接着程序会判断报警功能是否启用（第 17 行）。如果启用，则调用 alarm()函数进行判断和报警（第 19 行）。

5.2.7　菜单功能的实现

虽然 V0.2 版的程序的功能相对简单，但是仍设计了菜单和快捷键，如图 5.21 所示。



(a) “文件”菜单　　(b) “数据”菜单　　(c) “关于”菜单

图 5.21　程序菜单的内容

在所有的菜单项中，本章可以实现的只有“文件”菜单的“退出”、“数据”菜单的“读取测量数据”、“关于”菜单的“关于”。因为退出功能在前面已经介绍过了，所以此处主要讲解“读取测量数据”菜单项和“关于”菜单项的实现。

在 Action Editor 中将“读取测量数据”菜单项命名为 menuGetHardwareData。由于

“读取测量数据”菜单项的功能与“读取测量数据”按钮的功能相同，因此可以在菜单项的槽函数中直接调用“读取测量数据”按钮的槽函数，即

```
void MainWindow::on_menuGetHardwareData_triggered()
{
    on_pushButtonGetHardwareData_clicked();
}
```

按照习惯，单击“关于”菜单项后会弹出一个介绍程序相关信息的对话框。本例使用 Qt 消息对话框类 QMessageBox 的静态函数 information()显示一个简易的信息(information)对话框。要实现这一功能，首先要引用头文件：

```
#include <QMessageBox>
```

然后将“关于”菜单项的槽函数修改为：

```
void MainWindow::on_menuAbout_triggered()
{
    QMessageBox::information(this, "关于","简易气象站 V0.2\r\n——金陵科技学院电子信息工程学院");
}
```

information()函数的第一个参数是父窗体的指针，第二个参数是对话框的标题，第三个参数是对话框的内容。图 5.22 是“关于”对话框的运行结果。

图 5.22　气象站程序的“关于”对话框

Qt 提供了 5 种不同类型的对话框。除了上面使用的信息对话框外，还有 about(关于)、question(询问)、warning(警告)、critical(关键错误)对话框。这几种对话框的用法大致相同，读者可以顺次尝试。

5.3　程序运行结果

本章主要完成了界面的更新和串口数据的读取、处理。下面是这部分功能的测试结果。

（1）图 5.23 是程序启动后的界面。程序启动后首先检测计算机的串口，并将串口信息输出在日志区域和串口列表中。在本例中，程序共检测到了 4 个串口信息，其中有两个来自于 USB 转接板。此外，程序启动后界面中的各个控件显示默认值，警告图标为绿色。

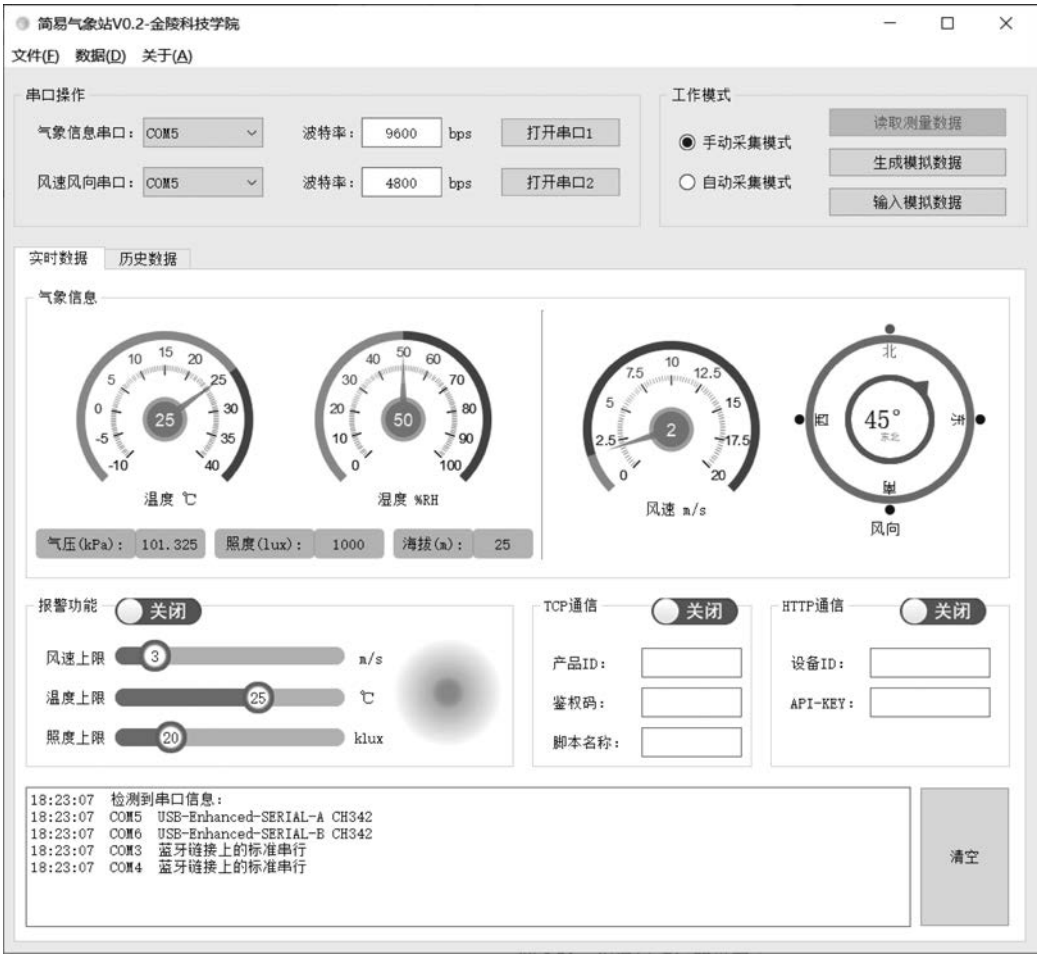


图 5.23　程序启动后的界面

（2）图 5.24 是打开串口并读取一组数据后，程序的运行结果。选择串口号并打开串口后，程序首先会在日志区域输出打开串口的结果。结果中的“已打开”代表打开成功。单击“读取测量数据”按钮后，程序会通过硬件读取数据并显示在日志区域和控件中。由于报警功能处于关闭状态，虽然风速、温度、照度等数据均超过了限值，但是程序不会报警。

（3）打开报警功能并重新读取一组测量数据，如图 5.25 所示。因为风速和照度值均超过了限值，所以程序开始报警。报警的现象与第 4 章相同，包括红色闪烁的图标和日志文字提示。

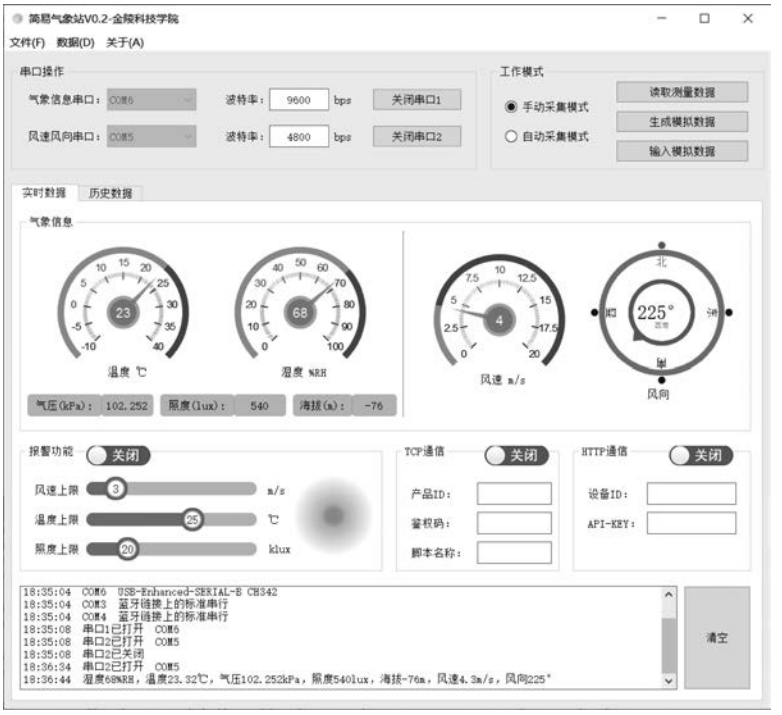


图 5.24　读取并显示一组测量数据

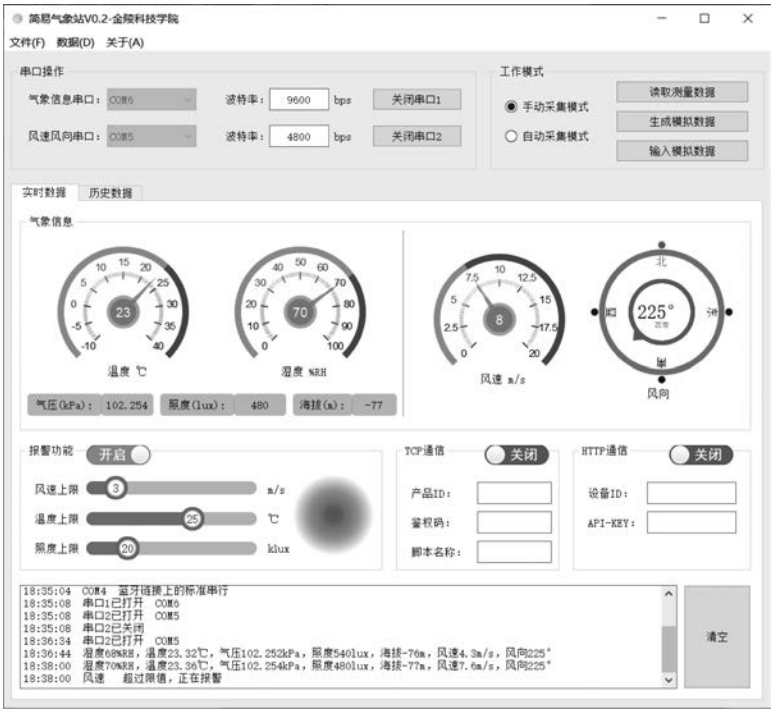


图 5.25　测量结果超过限值时，程序报警

（4）切换到历史数据界面，重复进行几次测量，程序自动显示历史记录曲线，如图 5.26 所示。需要注意的是，QUC SDK 无法自动调整纵坐标的显示范围，只能按照事先设定好的范围显示。对于温度这种变化缓慢的数据，其图形经常呈一条直线。要解决这一问题，可以使用变量记录下数据的最大值和最小值，然后调用控件的函数调整纵坐标范围。

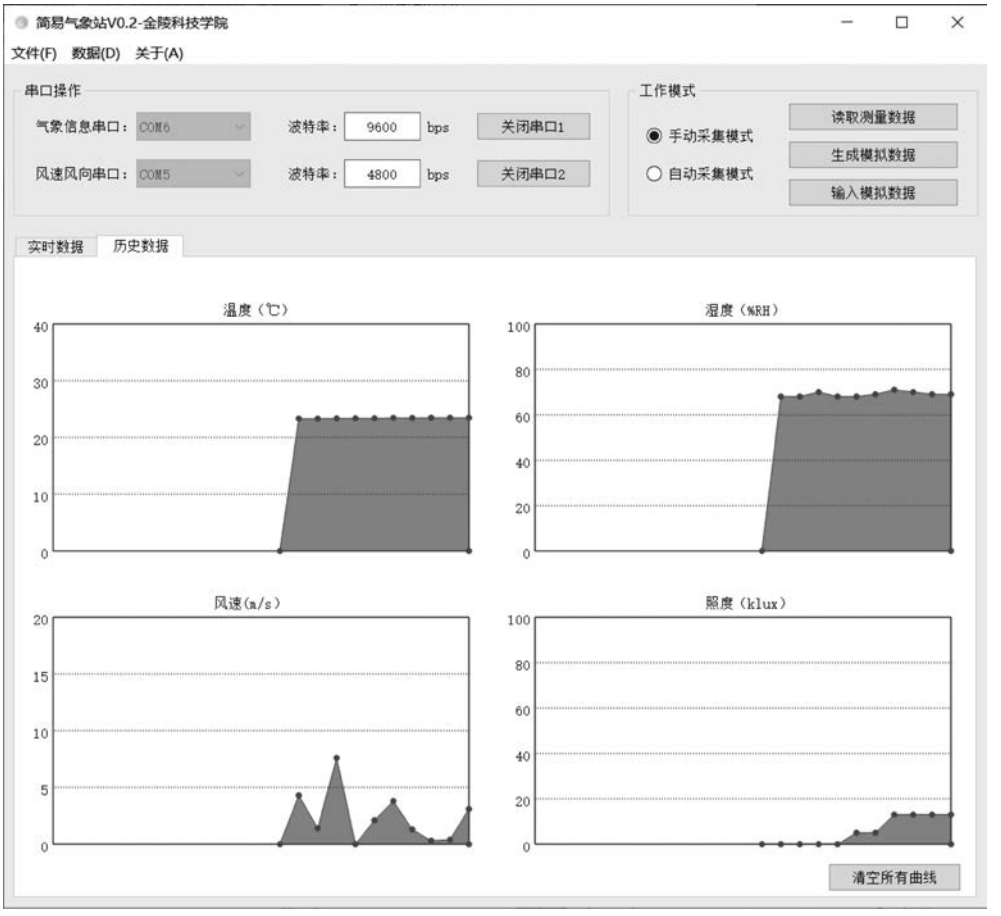


图 5.26　历史数据界面

5.4　本章小结

本章介绍了 Qt 中串口操作的方法、QUC SDK 的安装和使用方法、菜单的使用方法 3 部分内容，同时利用这些知识完成了简易气象站 V0.2 版的程序，实现了读取硬件数据的功能。在学习了本章的内容后，还可以学习串口蓝牙、串口 Wi-Fi 等模块的使用方法，并通过 Qt 编写程序控制这些模块；也可以试着动手编写一个简单的串口调试助手软件；如果有绘制图形、曲线的需要，也可以学习 QCustomPlot、QWT 等库的使用。

扩展阅读：阿里巴巴——中国重要的开源参与者

近几年，开源在国内异常火热，其全称为开放源代码，最大的特点是开放。任何人都可以得到软件的源代码，并可以修改、学习甚至重新发布代码。百度、阿里巴巴、华为、腾讯、浪潮等公司均是我国重要的开源软件贡献者。

早在 2010 年时，阿里巴巴的工程师们便在杭州开源了第一个项目——Dubbo。这是一款高性能、轻量级的开源服务框架，可提供面向接口代理的高性能 RPC 调用、智能容错和负载均衡、服务自动注册和发现。之后几年，阿里巴巴又相继开源了 Fastjson、Druid、Sea.js、Arale 等项目。

2017 年 9 月，阿里巴巴发起了 OpenMessaging 项目。这一项目也正式入驻 Linux 基金会，成为国内首个在全球范围发起的分布式计算领域的国际标准。在随后的一年里，OpenMessaging 开源标准社区又吸引了十余家企业的参与，获得了 RocketMQ、RabbitMQ 和 Pulsar 等 3 个消息开源厂商的支持。

迄今为止，阿里巴巴开源项目数已超过 2700 个，覆盖大数据、云、AI、数据库、中间件、硬件等多个领域。这些开源的项目收获了超过一百万颗星(Star)，参与贡献的开发人员达到几万人。阿里巴巴已成为十多个国内外开源基金会重要成员，包括 CNCF、MariaDB 基金会白金会员。

阿里巴巴开源技术委员会负责人曾说过，“各种成就的背后，离不开每一个开发者的耕耘和创造。我们经常发现，当各种喧嚣归于平静，当各种繁华归于平淡，我们的工程师们依然不变初心，追求着自己的梦想：通过代码这一种最直接的语言，通过开源这一种最简单的方式，寻找着技术路上的下一个突破点，寻找着技术对于社会创造的更多价值。开源是开发者最大的同心圆，未来，我们希望与更多开源人一起，用技术普惠世界。”