

Qt 串行通讯

Hanford

2016 年 11 月 09 日

目 录

第 1 章 Qt 串行通讯	1
1.1 配置.pro 文件	1
1.2 查询串口信息	1
1.3 配置、打开串口	3
1.4 setRequestToSend在Windows上的BUG	5
1.5 读取串口数据	6
1.6 发送串口数据	7
1.7 同步读取	7
1.8 本文示例代码	8
1.9 Qt 示例代码	10

第 1 章 Qt 串行通讯

最近要在 Android 手机上开发串行通讯程序，为此学习了一下 Qt 的串行通讯。本文中，Qt 的版本为 5.7.0。

1.1 配置.pro 文件

使用 Qt 5.7.0 创建 “Qt Widgets Application” 类型的项目，然后修改.pro 文件，如下图所示：

```
1 #-----
2 #
3 # Project created by QtCreator 2016-11-09T14:56:
4 #
5 #-----
6
7 QT      += core gui serialport
8
9 greaterThan(QT_MAJOR_VERSION, 4): QT += widgets
10
11 TARGET = QtSerialPort
12 TEMPLATE = app
```

图 1.1

给变量 QT 增加 serialport，说明程序里将使用串行通讯相关的类。

1.2 查询串口信息

本节将通过代码查找系统里的串口，然后填入下图所示的下拉列表框中。



图 1.2

函数 `QSerialPortInfo::availablePorts` 会返回系统所有的串口，它的使用请参考如下代码：

```
#include <QSerialPortInfo>
#include <map>

std::map<int,QString>    mapPort;
QString sPort;
int    nPort;
foreach (const QSerialPortInfo &info,QSerialPortInfo::availablePorts())
{//foreach 遍历 QSerialPortInfo::availablePorts() 的返回值
    sPort  =  info.portName();        //串口名称，如： COM5
    nPort  =  GetIntInStr(sPort);      //根据串口名称获取串口号，如： 5
    if(nPort >= 0)
    {
        mapPort[nPort]  =  sPort;    //根据串口号排序，加入 map
    }
}
```

函数 `GetIntInStr` 根据串口名称（如 COM5）获取串口号（如： 5），其代码如下：

```
/******\
从字符串里提取整数
s    [in]    字符串
返回：提取出来的整数，-1 表示错误
/******/
int GetIntInStr(const QString&s)
{
    bool    bOK    =    false;    //是否发现了数字
```

```

int    n    = 0;
int    nLenS = s.length(); //字符串长度
ushort c    = 0;
for(int i = 0; i < nLenS; ++i)
{
    c = s[i].unicode();
    if(c >= '0' && c <= '9')
    {
        bOK = true;
        n    = n * 10 + (c - '0');
    }
}
if(!bOK)
{
    n = -1; //没有数字, 返回 -1
}
return n;
}

```

根据 `std::map<int,QString> mapPort` 填充下拉列表框 `QComboBox cboPort` 的代码如下:

```

ui->cboPort->clear();
for(std::map<int,QString>::iterator it = mapPort.begin();
    it != mapPort.end(); ++it)
{
    ui->cboPort->addItem(it->second);
}

```

1.3 配置、打开串口

配置、打开串口的代码如下:

```

#include <QSerialPort>
m_port = new QSerialPort();
m_port->setPortName("COM1");           //打开 COM1
m_port->setBaudRate(9600);              //波特率: 9600
m_port->setParity(QSerialPort::NoParity); //校验法: 无
m_port->setDataBits(QSerialPort::Data8); //数据位: 8
m_port->setStopBits(QSerialPort::OneStop); //停止位: 1
m_port->setFlowControl(QSerialPort::NoFlowControl); //流控制: 无
if(m_port->open(QIODevice::ReadWrite))
{ //成功打开串口
    m_port->setRequestToSend(true);      //设置 RTS 为高电平
}

```

```

m_port->setDataTerminalReady(true);    //设置 DTR 为高电平
}

```

首先 new 一个 QSerialPort 对象，然后设置该对象的串行通讯参数，最后调用 QSerialPort::open 函数打开串口。

这里需要说明一下流控制。通讯的双方 A 和 B，假如 A 给 B 发送数据时，B 反应过慢，A 不管不顾的不停发送数据，结果会导致数据丢失。为了防止这种情况发生，可使用流控制（也叫握手）。

软件流控制（XON/XOFF）：通讯的一方（B）如果不能及时处理串口数据，会给对方（A）发送 XOFF 字符，对方接收到这个字符后，会停止发送数据；B 不再忙的时候，会给 A 发送 XON 字符，A 接收到这个字符后，会接着发送数据。软件流控制最大的问题就是不能传输 XON 和 XOFF。

硬件流控制（RTS/CTS）：硬件流控制需要按下图连接两个串口设备的 RTS 和 CTS。

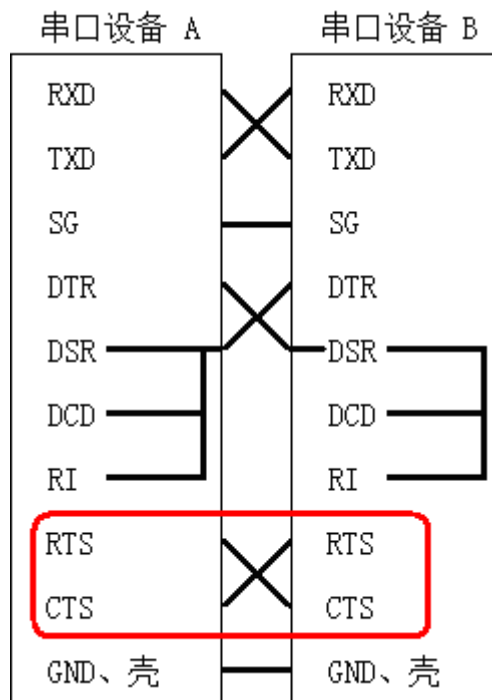


图 1.3

通讯的一方（B）如果不能及时处理串口数据，会设置自己的 RTS 为低电平，B 的 RTS 连着对方（A）的 CTS，A 发现自己的 CTS 为低电平，将停止发送数据；B 不再忙的时候，会设置自己的 RTS 为高电平，A 发现自己的 CTS

为高电平，将接着发送数据。

上面的代码中，设置流控制为无，其含义为：不管对方是否能够反应过来，这边只管发送数据。

当流控制为硬件时，系统会自动管理 RTS 和 DTR 的状态。否则，应该设置 RTS 和 DTR 为高电平，通知对方可以发送串口数据了。

1.4 setRequestToSend 在 Windows 上的 BUG

经测试，流控制为无时，调用 `m_port->setRequestToSend(true)` 是没有任何效果的。

下面的 Qt 源代码节选自文件 `C:\Qt\Qt5.7.0\5.7\Src\qtserialport\src\serialport\qserialport_win.cpp`（`C:\Qt\Qt5.7.0` 是 Qt 的安装目录）

```
bool QSerialPortPrivate::setRequestToSend(bool set)
{
    if (!::EscapeCommFunction(handle, set ? SETRTS : CLRRTS)) {
        setError(getSystemError());
        return false;
    }
    return true;
}
```

上面的代码调用 `EscapeCommFunction(handle, SETRTS)` 似乎没什么问题，但是请看下面的代码：

```
bool QSerialPortPrivate::setFlowControl(QSerialPort::FlowControl flowControl)
{
    ... ..
    dcb.fRtsControl = RTS_CONTROL_DISABLE;
    switch (flowControl) {
    case QSerialPort::NoFlowControl:
        break;
    case QSerialPort::SoftwareControl:
        break;
    case QSerialPort::HardwareControl:
        dcb.fRtsControl = RTS_CONTROL_HANDSHAKE;
        break;
    ... ..
}
```

硬件流控制时 `dcb.fRtsControl` 为 `RTS_CONTROL_HANDSHAKE`，这个没

问题。问题出在 `dcb.fRtsControl = RTS_CONTROL_DISABLE` 上，它直接禁用了 RTS，所以 `EscapeCommFunction(handle, SETRTS)` 并不能设置 RTS 为高电平。

那么 `m_port->setDataTerminalReady(true)` 为什么又是正常的呢？看代码：

```
bool QSerialPortPrivate::setDataTerminalReady(bool set)
{
    ...    ...    ...
    dcb.fDtrControl = set ? DTR_CONTROL_ENABLE : DTR_CONTROL_DISABLE;
    ...    ...    ...
}
```

set 为 true 时，`dcb.fDtrControl` 为 `DTR_CONTROL_ENABLE`，所以可以设置 DTR 为高电平。

1.5 读取串口数据

`m_port->readAll` (`QIODevice::readAll`) 用来读取串口数据。不过，它是异步执行的。什么是异步呢？那就是即使对方还没有发送串口数据，`m_port->readAll` 也会立即返回，而不是傻傻的等着对方发送数据过来后再返回。

既然是异步的，那么何时读取串口数据就成为了关键。Qt 提供的方案就是使用信号、槽。

```
connect(m_port, SIGNAL(readyRead()), this, SLOT(slotReadData()));
```

当对方发送串口数据后，将触发 `m_port` 的信号 `QIODevice::readyRead`。上面的代码将信号 `readyRead` 与槽函数 `slotReadData` 连接了起来，因此槽函数 `slotReadData` 将被调用，其代码如下：

```
void Widget::slotReadData()
{
    QByteArray data;
    const int nMax = 64 * 1024;
    for(;;)
    {
        data = m_port->readAll(); //读取串口数据
        if(data.isEmpty())
        { //没有读取到串口数据就退出循环
            break;
        }
        //读取到的串口数据，加入到 QByteArray m_dataCom
        m_dataCom.append(data);
    }
}
```



```
        if(m_dataCom.size() > nMax)
        { //防止 m_dataCom 过长
            m_dataCom = m_dataCom.right(nMax);
        }
    }
    ui->txtRecv->setText(m_dataCom); //将 m_dataCom 显示到文本框
    ui->txtRecv->moveCursor(QTextCursor::End); //移动文本框内的插入符
}
```

1.6 发送串口数据

`m_port->write (QIODevice::write)` 用来发送串口数据，不过它也是异步的。也就是说：代码 `m_port->write("123");` 会立即返回，至于数据 "123" 何时会发送给对方，那是操作系统的事情。操作系统不忙的时候，才会做此项工作。

参考如下代码：

```
char szData[32];
strcpy(szData,"123");
m_port->write(szData);
```

`m_port->write(szData);` 这行代码会立即返回。操作系统在空闲时才会将 `szData` 里的数据发送给对方。问题是：`m_port->write` 返回后，变量 `szData` 有可能已经被析构掉了，此时操作系统再访问 `szData` 所指向的内存，结果不可预料。

为此，需要改进上述代码，将异步通讯更改为同步通讯：

```
char szData[32];
strcpy(szData,"123");
m_port->write(szData);
m_port->waitForBytesWritten(5000);
```

就增加了一行代码 `m_port->waitForBytesWritten(5000);` 其含义为：操作系统把串口数据 `szData` 发送出去后，`m_port->waitForBytesWritten` 才会返回。不过，总不能无限制等下去吧？5000 就是等待时间的最大值，其单位为毫秒，5000 毫秒就是 5 秒。

1.7 同步读取

异步通讯的效率比较高，但是代码结构比较复杂。有时，需要同步读取。如：给对方发送字符串 `Volt`，对方回应电压值 5。代码会这么写：

```
m_port->write("Volt");
m_port->waitForBytesWritten(5000);
QByteArray data;
for(;;)
{
    data = m_port->readAll(); //读取串口数据
    if(!data.isEmpty())
    { //读到数据了，退出循环
        break;
    }
}
```

通过一个无限循环，将异步读取变成了同步读取。不过，上述代码运行时，CPU 占用率将会达到 100%（单核 CPU）。为此，需要改进代码：

```
m_port->write("Volt");
m_port->waitForBytesWritten(5000);
QByteArray data;
while(m_port->waitForReadyRead(3000))
{
    data = m_port->readAll(); //读取串口数据
    if(!data.isEmpty())
    { //读到数据了，退出循环
        break;
    }
}
```

修改了一行代码 `m_port->waitForReadyRead(3000)`，其含义为等待对方发送串口数据过来。如果对方发送串口数据过来了，它返回 `true`，然后使用 `m_port->readAll` 读取串口数据；如果对方在 3 秒内都没有发送串口数据过来，它返回 `false`，退出循环。

1.8 本文示例代码

本文示例代码已上传至 git 服务器，具体如下：

<https://github.com/hanford77/SerialPort>

<https://git.oschina.net/hanford/SerialPort>

为了便于测试，可使用 Virtual Serial Port Driver 7.1 创建一个串口对 COM100、COM200，如下图所示：

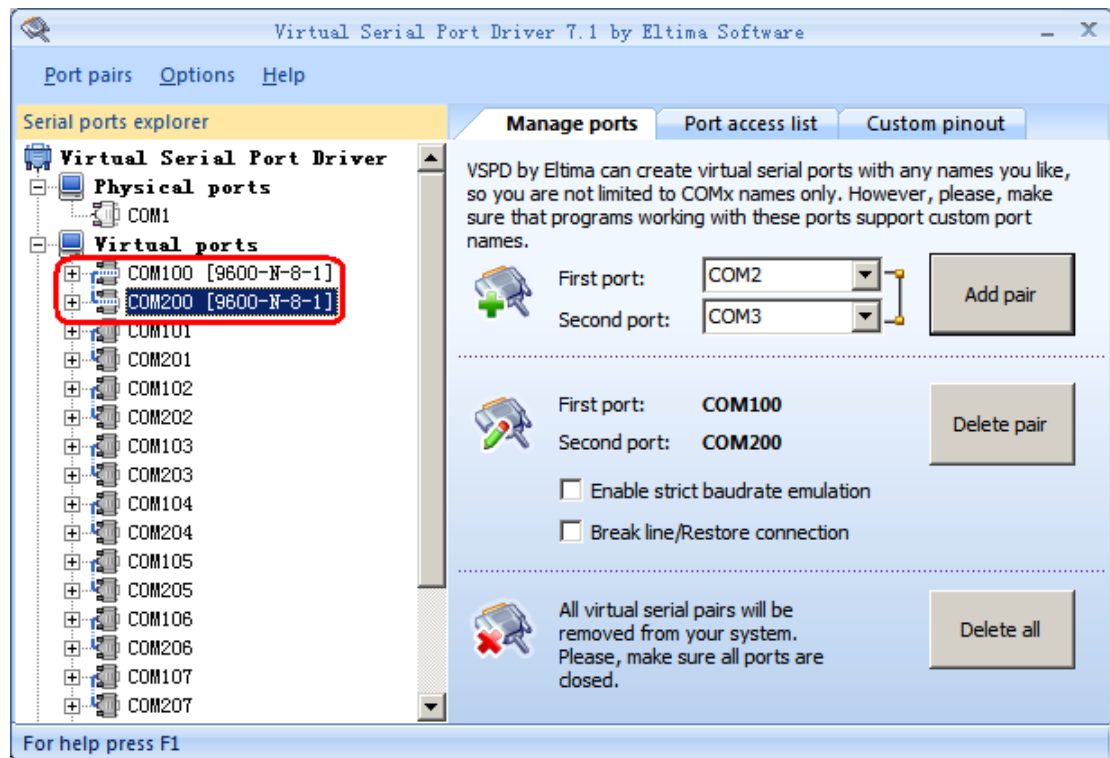


图 1.4

上图的 COM100、COM200 是虚拟出来的。COM100 发送的数据将会被 COM200 接收到，反之亦然。下图是测试界面：



图 1.5

本文示例代码打开了 COM100，另一个串行通讯程序打开了 COM200。两者可以相互发送数据。

1.9 Qt 示例代码

安装完 Qt 5.7.0 后，Qt 串行通讯的示例代码也被安装了，如下图所示：

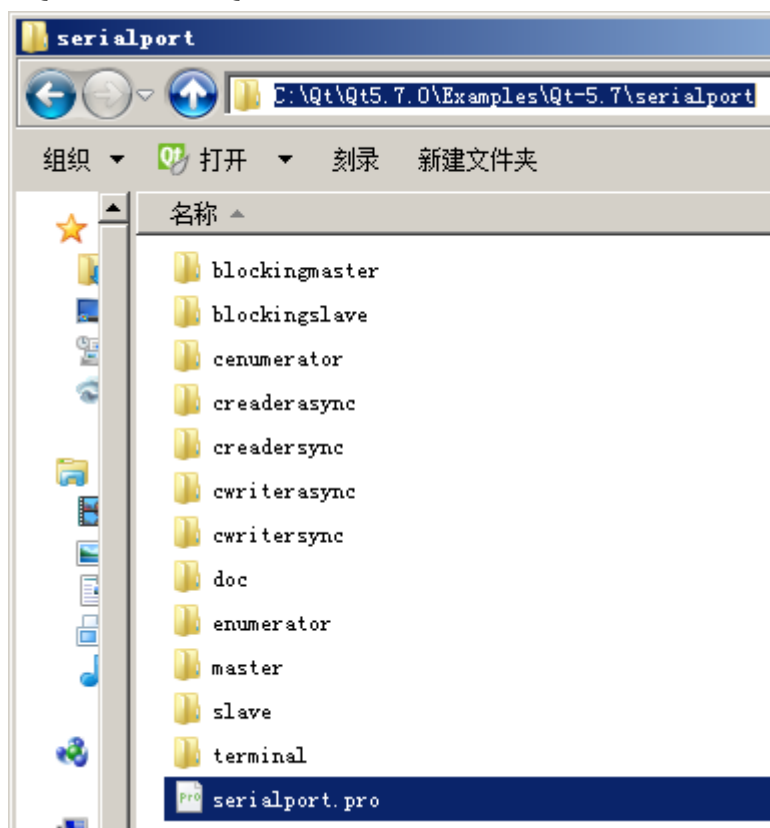


图 1.6

C:\Qt\Qt5.7.0 是笔者安装 Qt 5.7.0 时的安装目录。

需要进一步了解 Qt 串行通讯的可查看这些示例。如：creaderasync 表示异步读取；creadersync 表示同步读取……更多信息查看 Qt 帮助。