串行通讯之
.NET SerialPort

Hanford

2016 年 11 月 15 日

目 录

第 1 章 串行通讯之.NET SerialPort....2
1 枚举串口....2
2 打开/关闭串口....2
3 写数据....3
3.1 写二进制数据....3
3.2 写文本数据....4
4 读数据....5
4.1 读二进制数据....6
4.2 读一个字节....7
4.3 读一个字符....7
4.4 读全部文本....7
4.5 读文本到某个字符串....8
4.6 读一行文本....8
4.7 DataReceived事件....8
5 流控制....9
5.1 软件流控制（XON/XOFF）....10
5.2 硬件流控制（RTS/CTS）....10
6 输入信号....11

第 1 章　串行通讯之.NET SerialPort

.NET 库中类 System.IO.Ports.SerialPort 用于串行通讯，本文对其使用进行简要说明。

1　枚举串口

函数 System.IO.Ports.SerialPort.GetPortNames 将获得系统所有的串口名称。C#代码如下：

```
string[] arrPort = System.IO.Ports.SerialPort.GetPortNames();
foreach (string s in arrPort)
{
}
```

2　打开/关闭串口

下面的 C# 代码将打开 COM100:1200,N,8,1

```
System.IO.Ports.SerialPort m_sp = new System.IO.Ports.SerialPort();
private void btnOpen_Click(object sender, EventArgs e)
{
    m_sp.PortName = "COM100";                       //串口
    m_sp.BaudRate = 1200;                           //波特率：1200
    m_sp.Parity = System.IO.Ports.Parity.None;      //校验法：无
    m_sp.DataBits = 8;                              //数据位：8
    m_sp.StopBits = System.IO.Ports.StopBits.One;   //停止位：1
    try
    {
        m_sp.Open();                //打开串口
        m_sp.DtrEnable = true;      //设置 DTR 为高电平（含义见下文）
        m_sp.RtsEnable = true;      //设置 RTS 为高电平（含义见下文）
    }
    catch (System.Exception ex)
    {//打开串口出错，显示错误信息
        MessageBox.Show(ex.Message);
    }
```

```
}
```

下面的 C# 代码将关闭打开的串口

```
if(m_sp.IsOpen)   //判断串口是否已经被打开
{
    m_sp.Close();   //关闭串口
}
```

3　写数据

System.IO.Ports.SerialPort 用于写串口数据的成员函数有四个，如下所示：

函　　数	说　明
void Write(byte[] buffer, int offset, int count); void Write(char[] buffer, int offset, int count);	写二进制数据
void Write(string text);	写文本数据
void WriteLine(string text);	写一行文本

3.1　写二进制数据

void Write(byte[] buffer, int offset, int count);和 void Write(char[] buffer, int offset, int count);用于写二进制数据。它们的区别仅仅在于第一个参数不同：byte[]是无符号的，char[]是有符号的。对于二进制数据而言，byte、char 没有实质的区别。

下面的 C#代码，将写 1024 个 00H：

```
try
{
    if(m_sp.IsOpen)
    {
        byte[] bt = new byte[1024];
        m_sp.Write(bt,0,bt.Length); //写 1024 个 00H
    }
}
catch (System.Exception ex)
{//显示错误信息
    MessageBox.Show(ex.Message);
}
```

注意：

1、Write 函数是同步的。以上面的代码为例，1024 个 00H 在发送完之前，

Write 函数是不会返回的。波特率 1200，发送 1024 个字节大概要耗时 9 秒。如果这段代码在主线程里，那么这 9 秒内整个程序将处于假死状态：无法响应用户的键盘、鼠标输入；

2、WriteTimeout 属性用于控制 Write 函数的最长耗时。它的默认值为 System.IO.Ports.SerialPort.InfiniteTimeout，也就是-1。其含义为：Write 函数不将所有数据写完绝不返回。可以修改此属性，如下面的代码：

```
try
{
    if(m_sp.IsOpen)
    {
        byte[] bt = new byte[1024];
        m_sp.WriteTimeout = 5000; //Write 函数最多耗时 5000 毫秒
        m_sp.Write(bt,0,bt.Length);  //写 1024 个 00H
    }
}
catch (System.Exception ex)
{//显示错误信息
    MessageBox.Show(ex.Message);
}
```

上面的代码中，设置 WriteTimeout 属性为 5 秒。所以 Write 写数据时最多耗时 5 秒，超过这个时间未发的数据将被舍弃，Write 函数抛出异常 TimeoutException 后立即返回。

3.2　写文本数据

下面是 void Write(string text)的示例代码

```
try
{
    if(m_sp.IsOpen)
    {
        m_sp.Encoding = System.Text.Encoding.GetEncoding(936);
        m_sp.Write("串行通讯");
    }
}
catch (System.Exception ex)
{//显示错误信息
    MessageBox.Show(ex.Message);
}
```

首先设置代码页为 936（即 GBK 码），Write(string text)函数根据代码页把

字符串“串行通讯”转换为二进制数据，如下所示：

字符串	串	行	通	讯
内码	B4 AE	D0 D0	CD A8	D1 B6

然后把二进制数据 B4 AE D0 D0 CD A8 D1 B6 发送出去。

函数 void WriteLine(string text);等价于 void Write(text + NewLine)。参考下面的代码：

```
try
{
    if(m_sp.IsOpen)
    {
        m_sp.Encoding = System.Text.Encoding.GetEncoding(936);
        m_sp.NewLine = "\r\n";
        m_sp.WriteLine("串行通讯");
    }
}
catch (System.Exception ex)
{//显示错误信息
    MessageBox.Show(ex.Message);
}
```

代码 m_sp.NewLine = "\r\n";设置行结束符为回车（0DH）换行（0AH）。m_sp.WriteLine("串行通讯");等价于 m_sp.Write("串行通讯"+m_sp.NewLine);也就是 m_sp.Write("串行通讯\r\n");

最终，发送出去的二进制数据为 B4 AE D0 D0 CD A8 D1 B6 0D 0A。

4　读数据

System.IO.Ports.SerialPort 用于读串口数据的成员函数有七个，如下所示：

函　　数	说　明
int ReadByte();	读取一个字节
int ReadChar();	读取一个字符
int Read(byte[] buffer, int offset, int count); int Read(char[] buffer, int offset, int count);	读取二进制数据
string ReadExisting();	读取全部文本
string ReadTo(string value);	读取文本到某个字符串
string ReadLine();	读取一行文本

4.1　读二进制数据

下面的 C#代码，将读取 3 个字节的串口数据

```
if(m_sp.IsOpen)
{
    try
    {
        byte[] b = new byte[3];
        int n = m_sp.Read(b,0,3);   //返回值是读取到的字节数
    }
    catch (System.Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}
```

注意：

1、Read 函数是同步的。以上面的代码为例，3 个字节的数据被读取之前，Read 函数是不会返回的。如果这段代码在主线程里，那么整个程序将处于假死状态；

2、ReadTimeout 属性用于控制 Read 函数的最长耗时。它的默认值为 System.IO.Ports.SerialPort.InfiniteTimeout，也就是-1。其含义为：Read 函数未读取到串口数据之前是不会返回的。可以修改此属性，如下面的代码：

```
if(m_sp.IsOpen)
{
    try
    {
        byte[] b = new byte[3];
        m_sp.ReadTimeout = 2000;
        int n = m_sp.Read(b,0,3);   //返回值是读取到的字节数
    }
    catch (System.Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}
```

上面的代码中，设置 ReadTimeout 属性为 2 秒。所以 Read 函数读数据时最多耗时 2 秒。超过这个时间未读取到数据，Read 函数将抛出异常 TimeoutException，然后返回。

4.2　读一个字节

int ReadByte();与 int Read(byte[] buffer, int offset, int count);类似，它的特点就是只读取一个字节的串口数据。

4.3　读一个字符

int ReadChar();是读取一个字符，这个稍微复杂些。它可能读取 1~3 个字节的数据，然后合为一个字符。

如：m_sp.Encoding = System.Text.Encoding.GetEncoding(936);即字符串编码为 GBK。给 m_sp 发送“串”的 GBK 编码 B4 AE。ReadChar 首先读取一个字节得到 B4H。这是一个汉字的区码，还得读取一个字节得到位码。最终 ReadChar 读取的是 B4 AE。ReadChar 的返回值是 Unicode 编码，即返回前会把 GBK 编码 B4 AE 转换为 Unicode 编码 0x4E32。

再如：m_sp.Encoding = System.Text.Encoding.UTF8;即字符串编码为 UTF8。给 m_sp 发送“串”的 UTF8 编码 E4 B8 B2。ReadChar 会读取三个字节的串口数据 E4 B8 B2，然后将其转换为 Unicode 编码 0x4E32，并返回这个数值。

4.4　读全部文本

函数 string ReadExisting();读取串口输入缓冲区中的所有二进制数据，然后将其转换为字符串，最后返回字符串。

注意：

1、ReadExisting 会立即返回。如果输入缓冲区内没有数据，直接返回长度为零的空字符串；

2、ReadExisting 读取输入缓冲区后，有时会留几个字节。参考下面的代码：

```
m_sp.Encoding = System.Text.Encoding.GetEncoding(936);
string s = m_sp.ReadExisting();
int n = m_sp.BytesToRead;     //输入缓冲区剩余的字节数
```

“串”、“行”的 GBK 编码分别为 B4 AE 和 D0 D0。

首先发送 B4 AE D0 给 m_sp，运行上述代码。ReadExisting 将获得 B4 AE D0，“B4 AE”会被解释为“串”，D0 是汉字的区码，所以 ReadExisting 会将 D0 保留在输入缓冲区内。上述代码的运行结果就是：s 为"串"，n 为 1；

然后发送 D0 给 m_sp，运行上述代码。ReadExisting 将获得 D0 D0，“D0 D0”会被解释为“行”。上述代码的运行结果就是：s 为"行"，n 为 0。

4.5 读文本到某个字符串

函数 string ReadTo(string value);将在串口输入缓冲区内查找字符串 value。找到了，就返回 value 之前的字符串，同时清除缓冲区内 value 及其之前的数据；未找到，就一直等待，直至超时。

4.6 读一行文本

函数 string ReadLine();等价于 ReadTo(NewLine)。使用前，请设置 NewLine 属性，指定行结束符。

4.7 DataReceived 事件

串口输入缓冲区获得新数据后，会以 DataReceived 事件通知 System.IO.Ports.SerialPort 对象，可以在此时读取串口数据。请参考下面两段代码：

```
m_sp.ReceivedBytesThreshold = 1;
m_sp.DataReceived+=new System.IO.Ports.SerialDataReceivedEventHandler(m_sp_DataReceived);
```

```
void m_sp_DataReceived(object sender, System.IO.Ports.SerialDataReceivedEventArgs e)
{
    int nRead = m_sp.BytesToRead;
    if (nRead > 0)
    {
        byte[] data = new byte[nRead];
        m_sp.Read(data, 0, nRead);
    }
}
```

m_sp.ReceivedBytesThreshold = 1;的含义：串口输入缓冲区获得新数据后，将检查缓冲区内已有的字节数，大于等于 ReceivedBytesThreshold 就会触发 DataReceived 事件。这里设置为 1，显然就是一旦获得新数据后，立即触发 DataReceived 事件。

m_sp.DataReceived+=new System.IO.Ports.SerialDataReceivedEventHandler

(m_sp_DataReceived);的含义：对于 DataReceived 事件，用函数 m_sp_DataReceived 进行处理。

回调函数 m_sp_DataReceived 用于响应 DataReceived 事件，通常在这个函数里读取串口数据。它的第一个参数 sender 就是事件的发起者。上面的代码中，sender 其实就是 m_sp。也就是说：多个串口对象可以共用一个回调函数，通过 sender 可以区分是哪个串口对象。

回调函数是被一个多线程调用的，它不在主线程内。所以，不要在这个回调函数里直接访问界面控件。如下面的代码将将读取到的串口数据转换为字符串，然后显示在按钮 Open 上。红色代码处将产生异常。

```
void m_sp_DataReceived(object sender, System.IO.Ports.SerialDataReceivedEventArgs e)
{
    int nRead = m_sp.BytesToRead;
    if (nRead > 0)
    {
        byte[] data = new byte[nRead];
        m_sp.Read(data, 0, nRead);
        btnOpen.Text = System.Text.Encoding.Default.GetString(data);
    }
}
```

可使用 Invoke 或 BeginInvoke 改进上面的红色代码：

```
BeginInvoke(new Action<string>((x)=>{btnOpen.Text = x;})
            ,new Object[] {System.Text.Encoding.Default.GetString(data)});
```

5　流控制

串行通讯的双方，如果有一方反应较慢，另一方不管不顾的不停发送数据，就可能造成数据丢失。为了防止这种情况发生，需要使用流控制。

流控制也叫握手，System.IO.Ports.SerialPort 的 Handshake 属性用于设置流控制。它有四种取值：

取　值	说　明
System.IO.Ports.Handshake.None	无
System.IO.Ports.Handshake.XOnXOff	软件
System.IO.Ports.Handshake.RequestToSend	硬件
System.IO.Ports.Handshake.RequestToSendXOnXOff	硬件和软件

5.1　软件流控制（XON/XOFF）

串口设备 A 给串口设备 B 发送数据。B 忙不过来时（B 的串口输入缓冲区快满了）会给 A 发送字符 XOFF（一般为 13H），A 将暂停发送数据；B 的串口输入缓冲区快空时，会给 A 发送字符 XON（一般为 11H），A 将继续发送数据。

软件流控制最大的问题在于：通讯双方不能传输字符 XON 和 XOFF。

5.2　硬件流控制（RTS/CTS）

RTS/CTS 流控制是硬件流控制的一种，需要按下图连线：

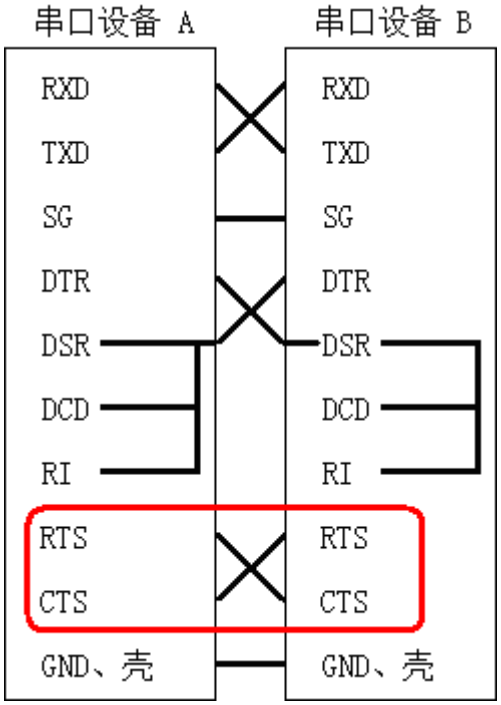

图 1

串口设备 A 给串口设备 B 发送数据。B 忙不过来时（B 的串口输入缓冲区快满了）会设置自己的 RTS 为低电平，这样 A 的 CTS 也变为低电平。A 发现自己的 CTS 为低电平后，会停止发送数据；B 的串口输入缓冲区快空时，会设置自己的 RTS 为高电平，这样 A 的 CTS 也变为高电平。A 发现自己的 CTS 为高电平后，会继续发送数据。

相同的道理，DTR/DSR 也可以做硬件流控制。

现在再来看看如下代码：

```
m_sp.Open();
m_sp.DtrEnable  =  true;
m_sp.RtsEnable  =  true;
```

为什么打开串口时需要设置 DTR、RTS 为高电平呢？原因就在于：如果对方设置了硬件流控制，而这边的 DTR、RTS 为低电平，那么对方就不会给这边发送数据。

需要注意的是：RTS/CTS 硬件流控制下，RTS 的电平由系统自行调整。调用 m_sp.RtsEnable = true;改变 RTS 的电平将会导致异常。

6　输入信号

上一节中，属性 DtrEnable、RtsEnable 可以控制输出信号 DTR、RTS。与之相应的，属性 CDHolding、CtsHolding、DsrHolding 可读取输入信号。

CtsHolding 为 true，说明对方的RTS为高电平（请按图 1所示连线）。

DsrHolding 为 true，说明对方的DTR为高电平（请按图 1所示连线）。