第 3 章　串口通信程序设计

学习内容和目标

学习内容：

- 了解 RS-232 串口通信原理与应用方法。
- 学习 PC 上串口通信的仿真过程。
- 学习串口通信程序设计。

学习目标：

(1) 掌握在单机上通过仿真工具的通信编程技术和实现能力。

(2) 在点对点串口通信程序设计全过程的系统实现能力。

3.1　串口通信基本原理和应用方法

3.1.1　串口通信原理

串口通信协议包括 RS-232、RS-422 和 RS-485 三种标准。

RS-232 在 1962 年发布，命名为 EIA-232-E，作为工业标准，以保证不同厂家产品之间的兼容。RS-232-C 是美国电子工业协会(Electronic Industry Association，EIA)制定的一种串行物理接口标准。RS 是英文“推荐标准”的缩写，232 为标识号，C 表示修改次数。RS-232-C 总线标准设有 25 条信号线，包括一个主通道和一个辅助通道。在多数情况下主要使用主通道，对于一般双工通信，仅需几条信号线就可实现，如一条发送线、一条接收线及一条地线。RS-232-C 标准规定的数据传输速率为每秒 50、75、100、150、300、600、1200、2400、4800、9600、19 200 波特。RS-232-C 标准规定，驱动器允许有 2500pF 的电容负载，通信距离将受此电容限制，例如，采用 150pF/m 的通信电缆时，最大通信距离为 15m；若每米电缆的电容量减小，通信距离可以增加。传输距离短的另一原因是 RS-232 属单端信号传送，存在共地噪声和不能抑制共模干扰等问题，因此一般用于 20m 以内的通信。

目前 RS-232 是 PC 与通信工业中应用最广泛的一种串行接口。RS-232 被定义为一种在低速率串行通信中增加通信距离的单端标准。RS-232 采取不平衡传输方式，即所谓单端通信。有 9 针和 25 针两种引脚，如图 3-1 和图 3-2 所示。

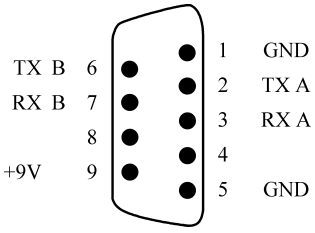

图 3-1　RS-232 的 DB9 连接器引脚

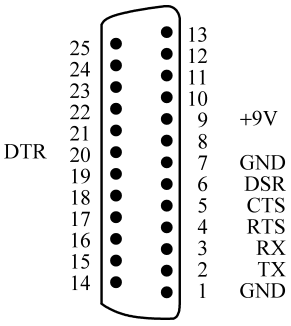

图 3-2　RS-232 的 DB25 连接器引脚

具体引脚定义如表 3-1 所示。

表 3-1　RS-232 串口引脚定义

9 针串口(DB9)			25 针串口(DB25)		
针号	功能说明	缩写	针号	功能说明	缩写
1	数据载波检测	DCD	8	数据载波检测	DCD
2	接收数据	RXD	3	接收数据	RXD
3	发送数据	TXD	2	发送数据	TXD
4	数据终端准备	DTR	20	数据终端准备	DTR
5	信号地	GND	7	信号地	GND
6	数据设备准备好	DSR	6	数据准备好	DSR
7	请求发送	RTS	4	请求发送	RTS
8	清除发送	CTS	5	清除发送	CTS
9	振铃指示	DELL	22	振铃指示	DELL

收、发端的数据信号是相对于信号地，例如，从 DTE 设备发出的数据在使用 DB25 连接器时是 2 脚相对 7 脚(信号地)的电平。典型的 RS-232 信号在正负电平之间摆动，在发送数据时，发送端驱动器输出正电平在＋5～＋15V，负电平在－5～－15V。当无数据传输时，线上为 TTL，从开始传送数据到结束，线上电平从 TTL 电平到 RS-232 电平再返回 TTL 电平。接收器典型的工作电平在＋3～＋12V 与－3～－12V。由于发送电平与接收电平的差仅为 2V 至 3V，所以其共模抑制能力差，再加上双绞线上的分布电容，其传送距离最大约为 15m，最高速率为 20kb/s。RS-232 是为点对点(即只用一对收、发设备)通信而设计的，其驱动器负载为 3～7kΩ。所以 RS-232 适合本地设备之间的通信。

RS-232C 串口通信接线方法(三线制)：

首先，串口传输数据只要有接收数据针脚和发送针脚就能实现：同一个串口的接收脚和发送脚直接用线相连，两个串口相连或一个串口和多个串口相连，即

- 同一个串口的接收脚和发送脚直接用线相连，对 9 针串口和 25 针串口，均是 2 与

3 直接相连。

• 两个不同串口(不论是同一台计算机的两个串口或分别是不同计算机的串口)。

表 3-2 是对微机标准串行口而言的,还有许多非标准设备,如接收 GPS 数据或电子罗盘数据,只要记住一个原则：接收数据针脚(或线)与发送数据针脚(或线)相连,彼此交叉,信号地对应相接。图 3-3 是 RS-232 的 9 针串口线。

表 3-2 RS-232C 串口通信接线方法

9 针—9 针		25 针—25 针		9 针—25 针	
2	3	3	2	2	2
3	2	2	3	3	3
5	5	7	7	5	7

RS-422 由 RS-232 发展而来,它是为了弥补 RS-232 之不足而提出的。为改进 RS-232 通信距离短、速率低的缺点,RS-422 定义了一种平衡通信接口,将传输速率提高到 10Mb/s,传输距离延长到 4000ft(1ft＝0.3048m)(速率低于 100kb/s 时),并允许在一条平衡总线上连接最多 10 个接收器。RS-422 是一种单机发送、多机接收的单向、平衡传输规范,被命名为 TIA/EIA-422-A 标准。

图 3-3 基于 RS-232 的 DB9 的实际串口线

为扩展应用范围,EIA 又于 1983 年在 RS-422 基础上制定了 RS-485 标准,增加了多点、双向通信能力,即允许多个发送器连接到同一条总线上,同时增加了发送器的驱动能力和冲突保护特性,扩展了总线共模范围,后命名为 TIA/EIA-485-A 标准。由于 EIA 提出的建议标准都是以“RS”作为前缀,所以在通信工业领域,仍然习惯将上述标准以 RS 作前缀称谓。

三种串口通信标准的特点如表 3-3 所示。注意,RS-232-C、RS-422 与 RS-485 标准只对接口的电气特性做出规定,而不涉及接插件、电缆或协议。

在 RS-232 的规范中,电压值在＋3～＋15V(一般使用＋6V)称为“0”或“ON”。电压在－3～－15V(一般使用－6V)称为“1”或“OFF”;计算机上的 RS-232“高电位”约为 9V,而“低电位”则约为－9V。而 RS-485 采用正负两根信号线作为传输线路。两线间的电压差为＋2～＋6V 表示逻辑“1”,两线间的电压差为－6～－2V 表示逻辑“0”。

表 3-3　三种串口通信标准的特点比较

规　　定		RS-232	RS-422	RS-485
工作方式		单端	差分	差分
节点数		1 收 1 发	1 发 10 收	1 发 32 收
最大传输电缆长度		50ft	400ft	400ft
最大传输速率		20kb/s	10Mb/s	10Mb/s
最大驱动输出电压		+/−25V	−0.25～+6V	−7～+12V
驱动器输出信号电平(负载最小值)	负载	+/−5～+/−15V	+/−2.0V	+/−1.5V
驱动器输出信号电平(空载最大值)	空载	+/−25V	+/−6V	+/−6V
驱动器负载阻抗		3～7kΩ	100Ω	54Ω
摆率(最大值)		30V/μs	N/A	N/A
接收器输入电压范围		+/−15V	−10～+10V	−7～+12V
接收器输入门限		+/−3V	+/−200mV	+/−200mV
接收器输入电阻		3～7kΩ	4kΩ(最小)	≥12kΩ
驱动器共模电压		—	−3～+3V	−1～+3V
接收器共模电压		—	−7～+7V	−7～+12V

强烈建议不要带电插拔串口,否则串口易损坏。不同编码机制不能混接,如 RS-232C 不能直接与 RS-422 接口相连,市面上有专门的各种转换器卖,必须通过转换器才能连接。

串行通信可分为同步通信和异步通信两种类型。较为广泛采用的是异步通信,如 RS-232,异步通信的标准数据格式如图 3-4 所示。

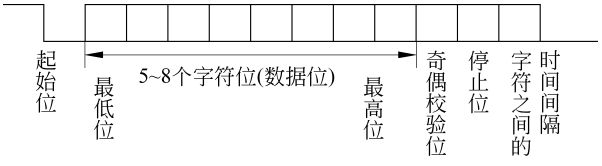

图 3-4　异步通信数据格式

从如图 3-4 所示的格式可以看出,异步通信的特点是逐个字符地传输,并且每个字符的传送总是以起始位开始、以停止位结束,字符之间没有固定的时间间隔要求。每一次有一个起始位,紧接着是 5～8 个数据位,再后为校验位,可以是奇检验,也可以是偶校验,也可不设置,最后是 1 比特、1.5 比特或 2 比特的停止位,停止位后面是不定长度的空闲位。停止位和空闲位都规定为高电平,这样就保证起始位开始处一定有一个下降沿,以此标识开始传送数据。

在串行通信中,数据通常是在两个站之间传送,按照数据在通信线路上的传送方向可

分为3种基本传送方式：单工、半双工和全双工。RS-232为全双工工作模式，其信号的电压是参考地线而得到的，可以同时进行数据的传送和接收。在实际应用中采用RS-232接口，信号的传输距离可以达到15m。

RS-485为半双工工作模式，其信号由正负两条线路信号准位相减而得，是差分输入方式，抗共模干扰能力强，即抗噪声干扰性好；实际应用中其传输距离可达1200m。RS-485具有多站能力，即一对多的主从通信。

3.1.2 串口通信仿真设计方法

这里需要用到两个仿真工具：一个是虚拟串口仿真工具，它能够仿真多对串口；另一个串口调试工具，如串口调试助手、串口精灵等，有事半功倍之效果。这对于串口通信程序的调试是非常必要的。用户能够在软件正式使用之前，先在单机上完成程序调试任务。

典型的虚拟串口仿真工具是VSPD(Virtual Serial Port Driver)，该软件由著名的软件公司Eltima制作。该软件可从其官方网站下载，压缩包里含有注册的DLL，可以无限制地使用。该软件运行稳定，允许模仿多个串口。可以虚拟两个串口然后连接起来实现自发自收调试，让程序从一个串口读数据，另外一个串口就用来连接串口调试工具。同时，还可以使用C/C++、C#、Delphi、Visual Basic等所有支持DLL的语言去编程模拟和控制串口。

这里使用的软件是VSPD 7.2英文版，其主界面如图3-5所示，已经配置好一对串口COM1和COM2。

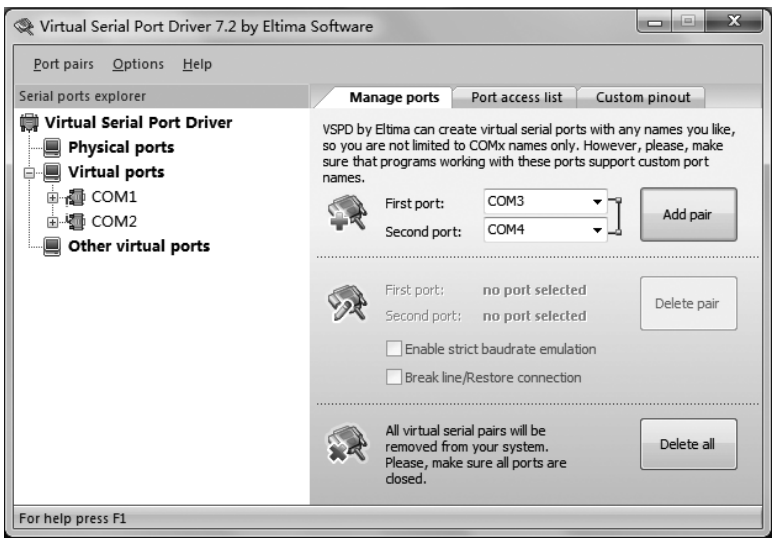

图3-5 VSPD主界面

然后查看设备管理，如图3-6所示。可见，已创建好了两个串口，并且两串口已连接。接着，打开串口调试工具，如图3-7所示。如果其串口选择了COM1，则开发的程序选择COM2，反之亦然。注意，两者的通信参数，如波特率、校验位、数据位和停止位都必须保

持一致。之后，双方就可以相互发送字符串和文件。

图 3-6　查看设备管理器上的串口配置信息

图 3-7　串口调试工具示例

3.2　串口通信编程类介绍

在 Visual Studio .NET 环境中编写串口通信程序，有以下三种简便方法。

- 采用 Visual Studio 原来的 MSComm 控件，这是最简单、最方便的方法，但需要注册。
- 微软在.NET 推出了一个串口控件，基于.NET 的 P/Invoke 调用方法实现。
- .NET Framework 2.0 类库开始包含了 SerialPort 类，方便地实现了所需要串口

通信的多种功能。以下仅介绍这种方法。

3.2.1 SerialPort 类介绍

1. 命名空间

System.IO.Ports 命名空间包含了控制串口重要的 SerialPort 类，该类提供了同步 I/O 和事件驱动的 I/O、对引脚和中断状态的访问以及对串行驱动程序属性的访问，所以在程序代码起始位置需加入 using System.IO.Ports。

2. 串口的通信参数

串口通信最常用的参数就是通信端口号及通信格式（波特率、数据位、停止位和校验位），在 MSComm 中相关的属性是 CommPort 和 Settings。SerialPort 类与 MSComm 有一些区别。

1）通信端口号

[PortName]属性获取或设置通信端口，包括但不限于所有可用的 COM 端口，请注意该属性返回类型为 String，不是 Mscomm.CommPort 的 short 类型。通常情况下，PortName 正常返回的值为 COM1、COM2、…、SerialPort 类支持的最大端口数突破了 CommPort 控件中 CommPort 属性不能超过 16 的限制，大大方便了用户串口设备的配置。

2）通信格式

SerialPort 类对分别用[BaudRate]、[Parity]、[DataBits]、[StopBits]属性设置通信格式中的波特率、数据位、停止位和校验位，其中[Parity]和[StopBits]分别是枚举类型 Parity、StopBits，Parity 类型中枚举了 Odd（奇）、Even（偶）、Mark、None、Space，Parity 枚举了 None、One、OnePointFive、Two。

SerialPort 类提供了七个重载的构造函数，既可以对已经实例化的 SerialPort 对象设置上述相关属性的值，也可以使用指定的端口名称、波特率和奇偶校验位数据位和停止位直接初始化 SerialPort 类的新实例。

3. 串口的打开和关闭

SerialPort 类没有采用 MSComm.PortOpen＝True/False 设置属性值打开关闭串口，相应的是调用类的 Open()和 Close()方法。

4. 数据的发送和读取

Serial 类调用重载的 Write 和 WriteLine 方法发送数据，其中 WriteLine 可发送字符串并在字符串末尾加入换行符，读取串口缓冲区的方法有许多，其中除了 ReadExisting 和 ReadTo，其余的方法都是同步调用，线程被阻塞直到缓冲区有相应的数据或大于 ReadTimeOut 属性设定的时间值后，引发 ReadExisting 异常。

5. DataReceived 事件

该事件类似于 MSComm 控件中的 OnComm 事件，DataReceived 事件在接收到[ReceivedBytesThreshold]设置的字符个数或接收到文件结束字符并将其放入输入缓冲

区时被触发。其中[ReceivedBytesThreshold]相当于 MSComm 控件的[Rthreshold]属性,该事件的用法与 MsComm 控件的 OnComm 事件在 CommEvent 为 comEvSend 和 comEvEof 时是一致的。

3.2.2　SerialPort 的使用

SerialPort 类的使用很方便。在进行串口通信时,一般的流程是设置通信端口号及波特率、数据位、停止位和校验位,再打开端口连接,发送数据,接收数据,最后关闭端口连接这样几个步骤。

数据接收的设计方法比较重要,采用轮询的方法比较浪费时间,在 Visual Basic 中的延时方法中一般会调用 API 并用 DOEvents 方法来处理,但程序不易控制,建议采用 DataReceived 事件触发的方法,合理地设置 ReceivedBytesThreshold 的值,若接收的是定长数据,则将 ReceivedBytesThreshold 设为接收数据的长度,若接收数据的结尾是固定字符或字符串,则可采用 ReadTo 的方法或在 DataReceived 事件中判断接收的字符是否满足条件。

SerialPort 类读取数据的许多方法是同步阻塞调用,尽量避免在主线程中调用,可以使用异步处理或线程间处理调用这些读取数据的方法。

由于 DataReceived 事件在辅线程被引发,当收到完整的一条数据,返回主线程处理或在窗体上显示时,请注意跨线程的处理,C＃可采用控件异步委托的方法 Control.BeginInvoke 及同步委托的方法 Invoke。

3.2.3　C＃ SerialPort 运行方式

SerialPort 中串口数据的读取与写入有较大的不同。由于串口不知道数据何时到达,因此有两种方法可以实现串口数据的读取:一是线程实时读串口;二是事件触发方式实现。由于线程实时读串口的效率不是十分高效,因此比较好的方法是事件触发的方式。在 SerialPort 类中有 DataReceived 事件,当串口的读缓存有数据到达时则触发 DataReceived 事件,其中 SerialPort.ReceivedBytesThreshold 属性决定了当串口读缓存中数据多少个时才触发 DataReceived 事件,默认为 1。

另外,SerialPort.DataReceived 事件的运行比较特殊,其运行在辅线程中,不能与主线程中的显示数据控件直接进行数据传输,必须用间接方式实现。如下:

```
SerialPort spSend;          //spSend,spReceive 用虚拟串口连接,它们之间可以相互
                            //传输数据。spSend 发送数据
SerialPort spReceive;       //spReceive 接收数据
TextBox txtSend;            //发送区
TextBox txtReceive;         //接收区
Button btnSend;             //数据发送按钮
delegate void HandleInterfaceUpdateDelegate(string text);   //委托,此为重点
HandleInterfaceUpdateDelegate interfaceUpdateHandle;
public void InitClient()    //窗体控件已在初始化
```

```
{
    interfaceUpdateHandle=new HandleInterfaceUpdateDelegate(UpdateTextBox);
                            //实例化委托对象
    spSend.Open();          //SerialPort 对象在程序结束前必须关闭,在此说明
    spReceive.DataReceived+=Ports.SerialDataReceivedEventHandler(spReceive_
    DataReceived);
    spReceive.ReceivedBytesThreshold=1;
    spReceive.Open();
}

public void btnSend_Click(object sender,EventArgs e)
{
    spSend.WriteLine(txtSend.Text);
}

public void spReceive_DataReceived(object sender,Ports.SerialDataReceived-
EventArgs e)
{
    byte[] readBuffer=new byte[spReceive.ReadBufferSize];
    spReceive.Read(readBuffer,0,readBuffer.Length);
    this.Invoke(interfaceUpdateHandle,new string[]{Encoding.Unicode.
    GetString(readBuffer)});
}

private void UpdateTextBox(string text)
{
    txtReceive.Text=text;
}
```

3.3 串口通信编程实例

下面给出一个实例，描述如何设计和实现 RS-232 串口通信程序。

使用 SerialPort 类，必须增加以下命名空间：

```
using System.IO.Ports;
```

设计界面如图 3-8 所示，包括了串口参数设置、发送字符信息、发送文件信息和接收字符信息的功能。其中，定义了 2 个定时器，time1 作为文件信息发送用，time2 作为数据接收用。状态条 statusStrip1 可显示通信参数和通信状态。

3.3.1 串口通信参数设置

窗体名称为 SPSet.cs，设计界面如图 3-9 所示。

图 3-8　串口通信程序设计界面

窗体初始化信息：

```
private void ComSet_Load(object sender, EventArgs e)
{
    //串口
    string[] ports=SerialPort.GetPortNames();
    foreach (string port in ports)
    {
        cmbPort.Items.Add(port);
    }
    cmbPort.SelectedIndex=0;

    //波特率
    cmbBaudRate.Items.Add("110");
    cmbBaudRate.Items.Add("300");
    cmbBaudRate.Items.Add("1200");
    cmbBaudRate.Items.Add("2400");
    cmbBaudRate.Items.Add("4800");
    cmbBaudRate.Items.Add("9600");
    cmbBaudRate.Items.Add("19200");
    cmbBaudRate.Items.Add("38400");
    cmbBaudRate.Items.Add("57600");
    cmbBaudRate.Items.Add("115200");
    cmbBaudRate.Items.Add("230400");
    cmbBaudRate.Items.Add("460800");
    cmbBaudRate.Items.Add("921600");
```

图 3-9　通信参数设置界面

```
    cmbBaudRate.SelectedIndex=5;
    //数据位
    cmbDataBits.Items.Add("8");
    cmbDataBits.Items.Add("7");
    cmbDataBits.Items.Add("6");
    cmbDataBits.SelectedIndex=0;
    //停止位
    cmbStopBit.Items.Add("1");
    cmbStopBit.SelectedIndex=0;
    //校验位
    cmbParity.Items.Add("无");
    cmbParity.SelectedIndex=0;
}

//"确定"按钮的单击处理事件
private void button1_Click(object sender, EventArgs e)
{
    //以下 4 个参数都是从窗体 MainForm 传入的
    MainForm.strPortName=cmbPort.Text;
    MainForm.strBaudRate=cmbBaudRate.Text;
    MainForm.strDataBits=cmbDataBits.Text;
    MainForm.strStopBits=cmbStopBit.Text;
    DialogResult=DialogResult.OK;
}
```

3.3.2 主程序设计

1. 设置全局变量

```
protected Boolean stop=false;
protected Boolean conState=false;
private StreamReader sRead;
string strRecieve;
bool bAccept=false;
SerialPort sp=new SerialPort();          //实例化串口通信类
//以下定义 4 个公有变量,用于参数传递
public static string strPortName="";
public static string strBaudRate="";
public static string strDataBits="";
public static string strStopBits="";
```

2. 通信参数设置

按钮“设置串口”的单击处理事件：

```
private void btnSetSP_Click(object sender, EventArgs e)     //串口设置
```

```
{
    timer1.Enabled=false;
      sp.Close();
      ComSet dlg=new ComSet();          //调用通信参数设置窗体
      if (dlg.ShowDialog()==DialogResult.OK)
      {
          sp.PortName=strPortName;
          sp.BaudRate=int.Parse(strBaudRate);
          sp.DataBits=int.Parse(strDataBits);
          sp.StopBits=(StopBits)int.Parse(strStopBits);
          sp.ReadTimeout=500;
      }
}
```

3. 按钮“打开串口”的单击处理事件

该按钮起到翻转作用，单击“打开串口”成功后，串口打开，其名称变为“关闭串口”；再单击又恢复到“关闭”状态，名称变为“打开串口”。如此，起到了两个按钮的作用。

```
private void btnSwitchSP_Click(object sender, EventArgs e)//打开串口
{
    if (btnSwitchSP.Text=="打开串口")
    {
        if(strPortName !="" && strBaudRate !="" && strDataBits !="" &&
        strStopBits !="")
        {
            try
            {
                if (sp.IsOpen)
                {
                    sp.Close();
                    sp.Open();        //打开串口
                }
                else
                {
                    sp.Open();        //打开串口
                }
                btnSwitchSP.Text="关闭串口";
                groupBox1.Enabled=true;
                groupBox2.Enabled=true;
                this.toolStripStatusLabel1.Text="端口号: "+sp.PortName+" | ";
                this.toolStripStatusLabel2.Text="波特率: "+sp.BaudRate+" | ";
                this.toolStripStatusLabel3.Text="数据位: "+sp.DataBits+" | ";
                this.toolStripStatusLabel4.Text="停止位: "+sp.StopBits+" | ";
                this.toolStripStatusLabel5.Text="";
```

```
            }
                catch (Exception ex)
                {
                    MessageBox.Show("错误: "+ex.Message,"C#串口通信");
                }
            }
            else
            {
                MessageBox.Show("请先设置串口!","RS-232串口通信");
            }
        }
        else
        {
            timer1.Enabled=false;
            btnSwitchSP.Text="打开串口";
            sp.Close();
            groupBox1.Enabled=false;
            groupBox2.Enabled=false;
            this.toolStripStatusLabel1.Text="端口号: 端口未打开 | ";
            this.toolStripStatusLabel2.Text="波特率: 端口未打开 | ";
            this.toolStripStatusLabel3.Text="数据位: 端口未打开 | ";
            this.toolStripStatusLabel4.Text="停止位: 端口未打开 | ";
            this.toolStripStatusLabel5.Text="";
        }
    }
```

4. 发送字符信息

```
private void btnSendData_Click(object sender, EventArgs e)    //数据发送
{
    if (sp.IsOpen)
    {
        try
        {
            sp.Encoding=System.Text.Encoding.GetEncoding("GB2312");
               //GB2312即信息交换用汉字编码字符集
            sp.Write(txtSend.Text);
        }
        catch (Exception ex)          //若应用程序出现调试问题
        {
              MessageBox.Show("错误: "+ex.Message);
        }
    }
    else
    {
```

```
        MessageBox.Show("请先打开串口!");
    }
}
```

要注意,发送编码应选择 GB2312,这样才能够正确发送汉字信息。

5. 文件信息发送

需要用到定时器 time1,用于循环发送文件内容。这里只能处理文本文件。

```
private void btnSendFile_Click(object sender, EventArgs e)//文件发送
{
    string fileName=textBox1.Text;
    if (fileName=="")
    {
        MessageBox.Show("请选择要发送的文件!", "Error");
        return;
    }
    else
    {
        sRead=new StreamReader(fileName);
    }
    timer1.Start();
}

//定时器处理事件
private void timer1_Tick(object sender, EventArgs e)
{
    string str1;
    str1=sRead.ReadLine();
      if (str1==null)
    {
        timer1.Stop();
        sRead.Close();
        MessageBox.Show("文件发送成功!","C#串口通信");
        this.toolStripStatusLabel5.Text="";
        return;
    }
    byte[] data=Encoding.Default.GetBytes(str1);
    sp.Write(data, 0, data.Length);
    this.toolStripStatusLabel5.Text="      文件发送中...";
}
```

6. 字符信息接收

单击“接收数据”按钮后,系统能自动接收对方的信息。

```
private void btnReceiveData_Click(object sender, EventArgs e)     //接收信息
```

```
{
    sp.Encoding=System.Text.Encoding.GetEncoding("GB2312");  //汉字编码字符集
    if (sp.IsOpen)
    {
        timer2.Enabled=true;
    }
    else
    {
        MessageBox.Show("请先打开串口!");
    }
}

//定时器 2 处理事件
private void timer2_Tick(object sender, EventArgs e)
{
    string str=sp.ReadExisting();
    string str4=str.Replace("\r", "\r\n");
    textBox2.AppendText(str4);
    textBox2.ScrollToCaret();
}
```

下面用到了接收信息的代理功能，此为设计要点之一。

```
delegate void DelegateAcceptData();
void fun()
{
    while (bAccept )
    { AcceptData(); }
}

delegate void reaction();
void AcceptData()
{
    if (textBox2.InvokeRequired)
    {
        try
        {
            DelegateAcceptData ddd=new DelegateAcceptData(AcceptData);
            this.Invoke(ddd, new object[] { });
        }
        catch { }
    }
    else
    {
        try
```

```
            {
                strRecieve=sp.ReadExisting();
                textBox2.Text+=strRecieve;
            }
            catch (Exception e) { }
        }
    }
```

3.3.3　串口通信程序测试

以下是程序运行情况，如图 3-10 所示。左侧为程序主界面，使用串口 COM1；右侧为使用的串口调试工具 SComAssistant，使用串口 COM2。

图 3-10 展示了字符收发和文件信息发送的运行状态。

图 3-10　串口通信程序的运行情况

为了调试方便，还设计了“导出信息”功能，能够将接收的信息全部保存到文本文件中。下面介绍的是实现代码。

```
private void btnOutputInfo_Click(object sender, EventArgs e)//导出信息
{
    try
    {
        string path=@ "c: \output.txt";
        string content=this.textBox2.Text;
        FileStream fs=new FileStream(path, FileMode.OpenOrCreate,
        FileAccess.Write);
        StreamWriter write=new StreamWriter(fs);
        write.Write(content);
        write.Flush();
        write.Close();
        fs.Close();
        MessageBox.Show("接收信息导出在"+path);
```

```
        }
        catch (Exception exp)
        {
            MessageBox.Show(exp.ToString());
        }
    }
```

小　　结

串口通信功能位于计算机网络的物理层，主要使用串口通信协议 RS-232C、RS-422 和 RS-485。本章重点分析了 RS-232 的通信原理和技术要求，通过 VS.NET 内置的串口通信类 SerialPort，设计和实现了一个简易的串口通信程序，该程序具备中英文字符的收发功能和文本文件的发送功能。

在设计中，要充分运用虚拟串口仿真工具 VSPD 和串口调试工具。这是串口通信程序的设计和调试中必不可少的条件。

实 验 项 目

1. 在 3.3 节内容的基础上，增加校验位设置功能。

2. 在 3.3 节内容的基础上，为串口通信程序增加“接收文件并保存”的功能。

3. 利用 RS-232C 串口线，在两台 PC 上调试串口通信程序，实现字符信息和文件传输。

第 4 章　基于 TCP 协议的程序设计

学习内容和目标

学习内容：

- 了解 TCP 协议的特点与数据包格式。
- 理解阻塞模式和非阻塞模式的特点及其应用。
- 掌握同步套接字编程和异步套接字编程方法。
- 掌握 TcpListener 和 TcpClient 的综合应用方法。

学习目标：

(1) 掌握基于 TCP 协议的同步/异步套接字编程方法。

(2) 学会基于 C/S 结构的网络聊天程序的设计与实现。

TCP(Transfer Control Protocol,传输控制协议),是 TCP/IP 协议簇中能够实现可靠数据传送的传输层协议,提供面向连接的可靠传输服务。与 IP 协议相结合,TCP 代表了网络协议的核心。

在第 1 章已经了解到,面向连接套接字调用时采用 TCP 协议,属于经典的客户机/服务器模式。基于 TCP 协议的编程,是学习网络通信应用程序设计的重要开端,对后续 UDP、FTP、SMTP/POP3 等协议编程也都具有参考性和比较性。

4.1　TCP 协议介绍

TCP 的工作主要是建立连接,然后从应用层程序中接收数据并进行传输。TCP 采用虚电路连接方式进行工作,在发送数据前需要在发送方和接收方之间建立一个连接,数据发送出去后,发送方会等待接收方给出一个确认性应答,否则发送方将认为此数据丢失,并重新发送此数据。

TCP 所提供服务的主要特点如下：

(1) 面向连接的传输；

(2) 端到端的通信；

(3) 高可靠性,确保传输数据的正确性,不出现丢失或乱序；

（4）全双工方式传输；

（5）采用字节流方式，即以字节为单位传输字节序列；

（6）紧急数据传送功能。

4.1.1 TCP 数据包格式

TCP 协议的数据包格式如图 4-1 所示，其中源端口号和目的端口号可用于套接字编程。另外 6 个标志位，即 URG、ACK、PSH、RST、SYN 和 FIN 的高级应用已经在第 2 章介绍过。

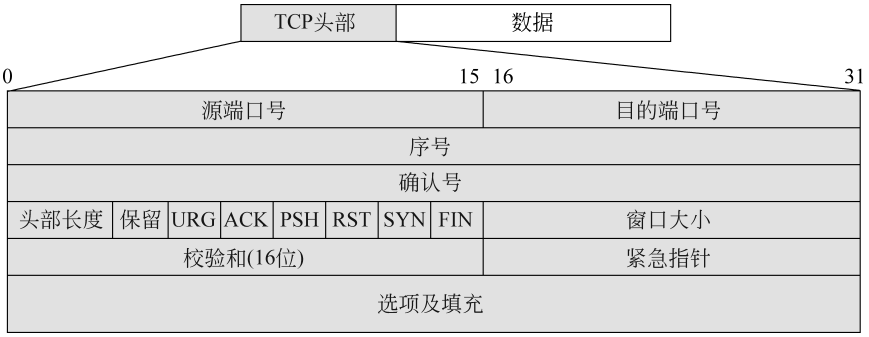

图 4-1 TCP 数据包格式

4.1.2 TCP 协议的通信特点

TCP 协议的握手通信遵从三次握手和四次挥手规则。

建立连接协议（三次握手）的过程如下：

（1）客户端发送一个带 SYN 标志的 TCP 报文到服务器。这是三次握手过程中的第一个报文。

（2）服务器端回应客户端，这是三次握手中的第二个报文，这个报文同时带 ACK 标志和 SYN 标志。因此，它既表示对刚才客户端 SYN 报文的回应；同时又标志 SYN 给客户端，询问客户端是否准备好进行数据通信。

（3）客户必须再次回应服务段一个 ACK 报文，这是第三个报文。

由于 TCP 连接是全双工的，因此每个方向都必须单独进行关闭。这原则是当一方完成它的数据发送任务后就能发送一个 FIN 来终止这个方向的连接。收到一个 FIN 只意味着这一方向上没有数据流动，一个 TCP 连接在收到一个 FIN 后仍能发送数据。首先进行关闭的一方将执行主动关闭，而另一方执行被动关闭。因此，TCP 连接的终止协议（四次挥手）如下。

（1）TCP 客户端发送一个 FIN，用来关闭客户到服务器的数据传送（报文段 4）。

（2）服务器收到这个 FIN，它发回一个 ACK，确认序号为收到的序号加 1（报文段 5）。和 SYN 一样，一个 FIN 将占用一个序号。

（3）服务器关闭客户端的连接，发送一个 FIN 给客户端（报文段 6）。

(4) 客户端发回 ACK 报文确认,并将确认序号设置为收到序号加 1(报文段 7)。

4.1.3　TCP 的常见端口

在第 1 章的 TCP/IP 协议簇中,应用层具有 FTP 等多个协议,它们位于 TCP 和 UDP 协议基础之上。其中,基于 TCP 协议的常见服务和端口如表 4-1 所示。

表 4-1　TCP 协议的常见端口及应用

端口号	服务进程	描　　述
20	FTP	文件传输协议(数据连接)
21	FTP	文件传输协议(控制连接)
23	Telnet	虚拟终端网络
25	SMTP	简单邮件传输协议
53	DNS	域名服务器
80	HTTP	超文本传输协议
111	RPC	远程过程调用

4.2　阻塞/非阻塞模式及其应用

首先,要了解以下几个容易混淆的概念:同步(synchronous)、异步(asynchronous)、阻塞(blocking)和非阻塞(unblocking)。

- 同步方式:指客户机在发送请求后,必须获得服务器的回应后才能发送下一个请求。此时,所有请求将会在服务器得到同步。
- 异步方式:指客户机在发送请求后,不必等待服务器的回应就能够发送下一个请求。
- 阻塞方式:指执行套接字的调用函数只有在得到结果之后才会返回;在调用结果返回之前,当前线程会被挂起,即此套接字一直阻塞在线程调用上,不会执行下一条语句。
- 非阻塞方式:指执行套接字的调用函数时,即使不能立即得到结果,该函数也不会阻塞当前线程,而是立即返回。

可见,同步和异步属于通信模式,而阻塞和非阻塞是属于套接字模式。一般而言,在实现效果方面,同步和阻塞方式一致,异步和非阻塞方式一致。

4.2.1　典型的阻塞模式

在默认情形下,所有的套接字都是阻塞模式,它的阻塞函数(主要是 accept()、connect()、send()、recv())直到操作完成才会返回控制权。在套接字上产生阻塞模式有以下四种情形。