第 1 章　轻松体验串口通信编程与调试

[内容提要]

首先，我们用串口调试助手来体验串口通信，熟悉这个调试工具是必要的，以后所有的实例都可以通过它来测试。

然后，编写一个串口通信小程序，没有用 VC 编写过串口通信程序的读者可以来感受一下用 VC 编写串口程序是多么简单的事情。您不需要对 VC 很熟练，只要能启动 VC 就可以了；以前从未接触过串口数据通信也没关系，只要我们一步一步编写并运行完这个程序，您就能有“不过如此”的感觉了，至于其他的概念，我们慢慢在编程中掌握。

本章最后，我们还要编写一个 DOS 环境下的串口程序，体验一下 Windows 环境与 DOS 环境下串口编程的区别。了解这些，对在各种应用场合编写串口通信程序是有帮助的。

当然，还应该具备以下条件：

■　**硬件环境**

能运行 Windows 9X/2000/XP 的 PC 机，配有两个串行口（可以是同一台 PC 机或分属于两台 PC 机的两个串口，串口连接线（若身边没有，可以参考第 11.2.2 节 3 线制连线方法自己制作一根连接线）。

■　**软件环境**

Visual C++ 6.0, Windows 9X/2000/XP

对于初次接触串口通信的读者，特别要注意：

注意　串口线插拔时，请关闭计算机电源，以免烧毁计算机的串行电路。自己制作的串口连线，使用前，用万用表测试一下焊接是否有错。

1.1　使用串口调试助手来体验串口通信

首先来看串口调试助手的功能。一旦熟悉了这个工具，你以后就会省很多事，所以值得花点时间。

串口调试助手是一个专门用来调试串口程序的工具软件，由本书作者编写，目前有大量工程人员和软件编程人员在使用，在作者技术主页和各大软件下载网站均可下载。其界面如图 1.1.1 所示，使用平台为 Windows 9X/NT/2000/XP，2.2 版本的程序仅供三线制（NONMODEM）串口调试之用，所有功能均置于界面上，一目了然，其义自明。界面分为串口设置区（左上角）、接收显示区和发送输入区，下面分别说明。

图 1.1.1 串口调试助手 V2.2

■ 串口设置区

在串口设置区可以设置串口、波特率、检验位、数据位及停止位等串口通信参数。参数设置好后，可以“打开串口”，若正常，则指示灯亮，显示为 ；否则会提示“串口不存在或被其他设备占用”，指示灯灭，显示为 。

■ 接收显示区

接收区可设置为自动清空，也可手动清空。设置为自动清空时，每接收一定行数后，清空接收区。为了看清楚接收的数据，可以让显示暂时停止（但数据仍在接收），之后可以继续显示。默认显示格式为 ASCII 码显示，也可以设置为十六进制显示，十六进制显示时，每个十六进制之间会有一个空隔，如 01 23 00 34 A5。接收的数据可以保存成文本文件，单击“保存显示数据”即可实现这一功能。默认保存文件夹为 C:\COMDATA，可以通过“更改”按钮改变保存的文件夹，以文件名 REC01.TXT，REC02.TXT，REC03.TXT 等（后面序号自动递增）保存。

■ 发送输入区

发送输入区可以填写需要发送的数据。输入要发送的字符数据后，单击“手动发送”按钮，则普通文本可以发送一次；若选中了“自动发送”，则会每隔设定的发送周期内发送一次，直到去掉“自动发送”为止。值得注意的一点是，选中“十六进制发送”后，发送框中所填字符每两个字符之间应有一个空隔，如：01 23 00 34 45。还有一些特殊的字符，如回车换行，则直接敲入回车即可。

另外，为了方便调试，程序设置了一些小功能。如窗口悬浮功能：单击程序左下角的图钉按钮，可以使程序置于最上层，保持可见，这时图钉按钮形状由 变为 。

程序还可放大至全屏，当需要扩大接收窗口以方便观看数据时，可以单击右上角最大化按钮；在程序的最下端，还有目前串口状态及接收与发送的字符数，计数可以清零。

现在，我们就要利用串口调试助手来体验一下串口通信了。普通计算机一般使用 9 针串口，如图 1.1.2 所示，如果有两台计算机，则可以用串口线连接起来，若是一台计算机上有两个串口，也可以把它们直接连接起来就可以了。

注意：再次提醒插拔串口线时，别忘记关掉计算机电源。

图 1.1.2　PC 机上 9 针串口的形状

接好串口线后，打开计算机，启动串口调试助手；若在同一计算机上，则将先启动的串口号设置为 COM2，再次启动串口调试助手（等于在同一计算机上启动两个程序），并将串口号设置为 COM1，实际设置时，串口号要根据自己的计算机配置设定相应的串口号。通信双方的串口参数要一致。下面，在一个串口调试助手的输入框输入 12345678ABCDEFG（加上回车换行），并选中“自动发送”，一切顺利的话，程序运行情况应如图 1.1.3 所示。读者还可以自己测试一下其他的功能：十六进制发送与接收、数据保存等，这些在今后实际调试中经常会用到。

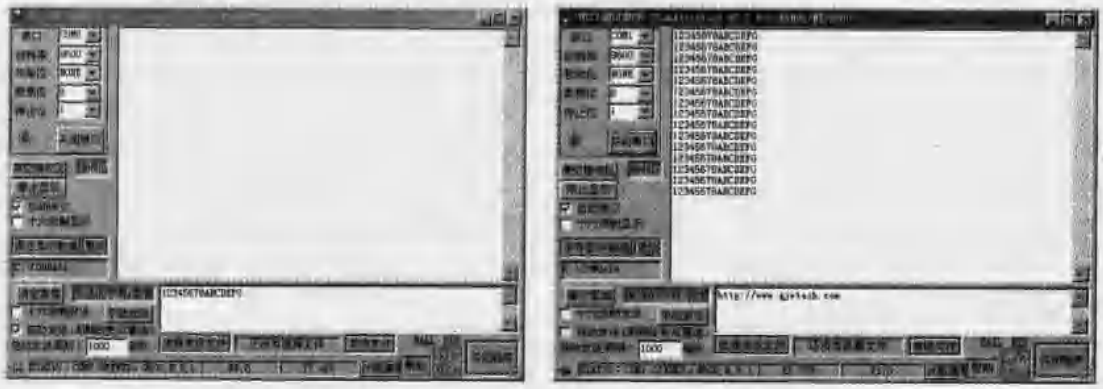

图 1.1.3　串口调试助手发送接收测试

到这里，我们可以对自己说：“我能用串口调试助手接收发送数据了。”这在以后调试许多设备时都是有用的，当然，更便于我们调试自己编写的程序。接下来，我们就要花半小时来做自己的串口程序了。我希望读者读完这本书后，会在几分钟之内写出自己的串口程序，就像我一样，有了这个基础，我们以后编写 TCP/IP 的程序，会一样很快的。是啊，有什么能

比自己编写的程序顺利运行更能让我们高兴的呢！

1.2　体验 Windows 环境下的 Visual C++串口通信编程

好的，让我们行动起来。如果你是初次接触串口通信编程，就跟着我一步一步来编写这个程序，为了方便学习，尽量与我的源代码保持一致。在这个程序中，应用了 VC 自带的 MSComm 控件，先不必去管 MSComm 控件的具体特性，第 3 章中会对其做详细介绍。

1．建立应用程序工程 SCommTest

打开 Visual C++ 6.0，建立一个基于对话框的 MFC 应用程序：SCommTest。然后在主对话框中添加控件，最后效果如图 1.2.1 所示。其中的电话状图标是 MSComm 控件，参照第 2 步的方法添加到对话框中。

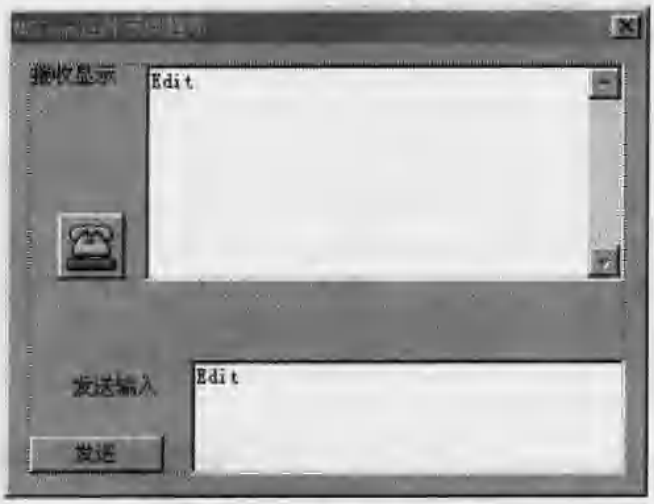

图 1.2.1　对话框添加控件后的状态

然后用 ClassWizard 为相应控件添加变量，控件的属性设置情况如表 1-2-1 所列。

表 1-2-1 控件及其属性设置情况

控件	控件 ID	Caption	需要添加的变量及变量类型
静态文本	IDC_STATIC	接收显示	
静态文本	IDC_STATIC	发送输入	
编辑框	IDC_EDIT_RXDATA		m_strEditRXData Value CString
编辑框	IDC_EDIT_TXDATA		m_strEditTXData Value CString
按钮	IDC_BUTTON_MANUALSEND	发送	
MSComm 控件	IDC_MSCOMM1		m_ctrlComm　control

2．在当前工程中添加 MSComm 控件

单击菜单 Add To Project-> Components and Controls...，就打开了如图 1.2.2 所示的添加组（控）件对话框。

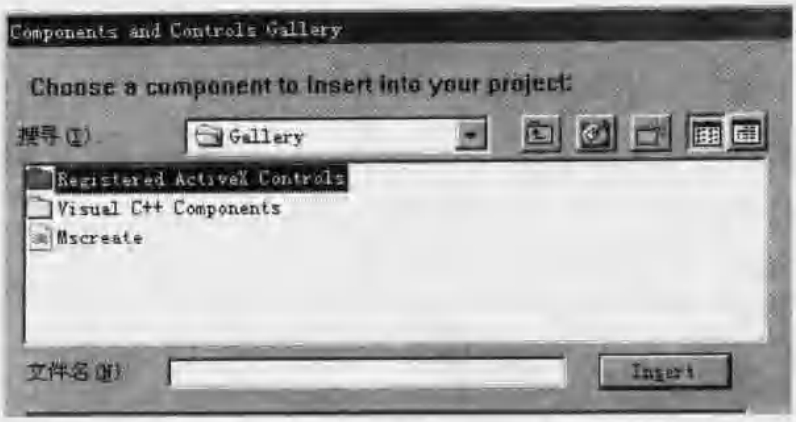

图 1.2.2　添加组件、控件对话框

再在其中双击“Registered ActiveX Controls”项（这时要稍等一会，这个过程较慢），就出现了如图 1.2.3 所示的控件选择（Component and Controls Gallery）对话框。在该对话框中选择“Microsoft Communications Control, version 6.0”控件。

ⓘ注意：如果在控件列表中看不到 Microsoft Communications Control, version 6.0，那可能是你在安装 VC6 时没有把 ActiveX 一项选上，这时需要重新安装 VC6，选上 ActiveX 就可以了。

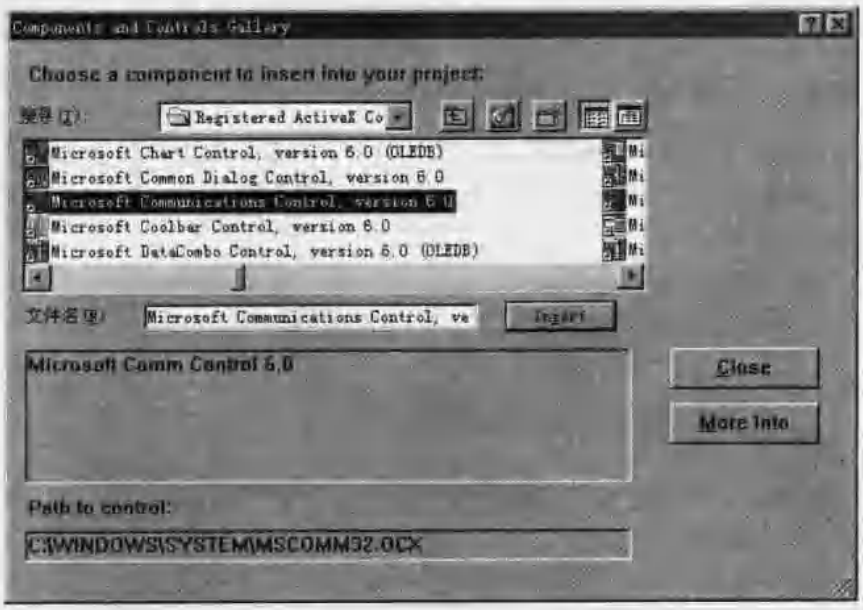

图 1.2.3　控件选择对话框

再单击“Insert”按钮，提示“Insert this component?”，确认后，可以看到如图 1.2.4 所示的加入 CMSComm 类的确认（Confirm Class）对话框，提示加入到当前工程中的 CMSComm 类头文件为 MSComm.h，实现文件为 MSComm.cpp。单击“OK”按钮，（Confirm Class）对话框关闭。再单击“Close”关闭（Component and Controls Gallery）对话框。在 VC 集成环境

中，当前工程的 ClassView 中就出现了 CMSComm 类。同时，在对话框资源控件中出现了一个电话机形状的控件（如图 1.2.5 所示），这就是 MSComm 控件。要在对话框中应用该控件，还需要将这个控件图标用鼠标拖入对话框中，这个对话框就成了 MSComm 控件的“宿主”。

提示：利用这种添加控件的方法，对之后的串口消息事件处理会提供很大的方便，ClassWizard 会自动在当前程序工程中进行消息类的映射，这一点我们在以下的编程中会体会到。

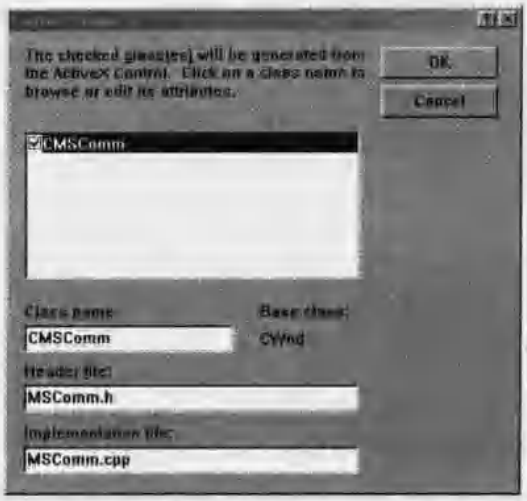

图 1.2.4　添加 CMSComm 类的确认对话框

图 1.2.5　MSComm 控件出现在工程资源中

3. 初始化串口：设置 MSComm 控件的属性

打开 ClassWizard->Member Variables 页，如图 1.2.6 所示，选中控件 IDC_MSCOMM1，再单击“Add Varialbe...”按钮，在 CSCommTestDlg 类中为控件 IDC_MSCOMM1 添加 CMSComm 控制变量 m_ctrlComm。

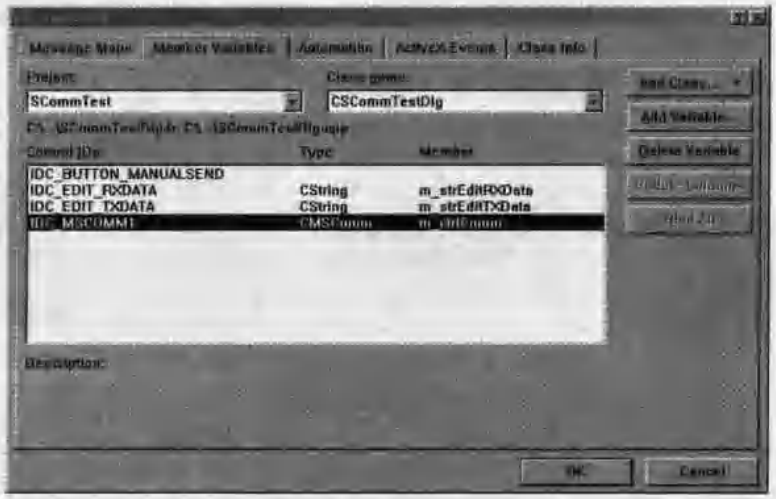

图 1.2.6　为控件 IDC_MSCOMM1 添加控制变量

通过以上操作，ClassWizard 自动在 SCommTestDlg.h 中加入了#include "mscomm.h"语句。

```
//{{AFX_INCLUDES()
#include "mscomm.h"
//}}AFX_INCLUDES
```

下面，在 CSCommTestDlg::OnInitDialog()函数中写入对串口的初始化语句，串口初始化语句由 IDC_MSCOMM1 的 CMSComm 控制变量 m_ctrlComm 来设置串口控件属性。代码如下：

```
BOOL CSCommTestDlg::OnInitDialog()
{
    CDialog::OnInitDialog();
     … …
    // TODO: Add extra initialization here
    m_ctrlComm.SetCommPort(1);   //选择 COM1
    //波特率 9600，无校验，8 个数据位，1 个停止位
    m_ctrlComm.SetInputMode(1); //输入方式为二进制方式
    m_ctrlComm.SetInBufferSize(1024); //设置输入缓冲区大小
    m_ctrlComm.SetOutBufferSize(512); //设置输出缓冲区大小
    //波特率 9600，无校验，8 个数据位，1 个停止位
    m_ctrlComm.SetSettings("9600,n,8,1");
    //参数 1 表示每当串口接收缓冲区中有多于
    //或等于 1 个字符时将引发一个接收数据的 OnComm 事件
    if(!m_ctrlComm.GetPortOpen())
        m_ctrlComm.SetPortOpen(TRUE);//打开串口
    m_ctrlComm.SetRThreshold(1);
    m_ctrlComm.SetInputLen(0);  //设置当前接收区数据长度为 0
    m_ctrlComm.GetInput();    //先预读缓冲区以清除残留数据

    return TRUE;  // return TRUE  unless you set the focus to a control
}
```

4．添加串口事件消息处理函数 OnComm()

MSComm 控件一般用事件驱动方式从串口接收数据，也就是消息处理，当串口有事件发生时，程序调用消息函数来处理数据。打开 ClassWizard->Member Variables 页，如图 1.2.7 所示，打开 ClassWizard—>Message Maps，在 Class Name 中选择类 CSCommTestDlg，再在 Object IDs 中选择 IDC_MSCOMM1，然后在 Message 中双击消息 OnComm（或单击“Add Function”按钮），在弹出的对话框中将函数名改为 OnComm（好记而已），单击“OK”，就加入了串口事件的消息处理函数。

我们注意到，这时自动添加了以下代码：

头文件 SCommTestDlg.h 中：

```
// Generated message map functions
//{{AFX_MSG(CSCommTestDlg)
… …
afx_msg void OnComm();
   … …
DECLARE_EVENTSINK_MAP()
//}}AFX_MSG
```

实现文件 SCommTestDlg.cpp 中：

```
BEGIN_EVENTSINK_MAP(CSCommTestDlg, CDialog)
```

```
    //{{AFX_EVENTSINK_MAP(CSCommTestDlg)
     ON_EVENT(CSCommTestDlg, IDC_MSCOMM1, 1 /* OnComm */, OnComm, VTS_NONE)
     //}}AFX_EVENTSINK_MAP
END_EVENTSINK_MAP()

void CSCommTestDlg::OnComm()
{
    // TODO: Add your control notification handler code here
}
```

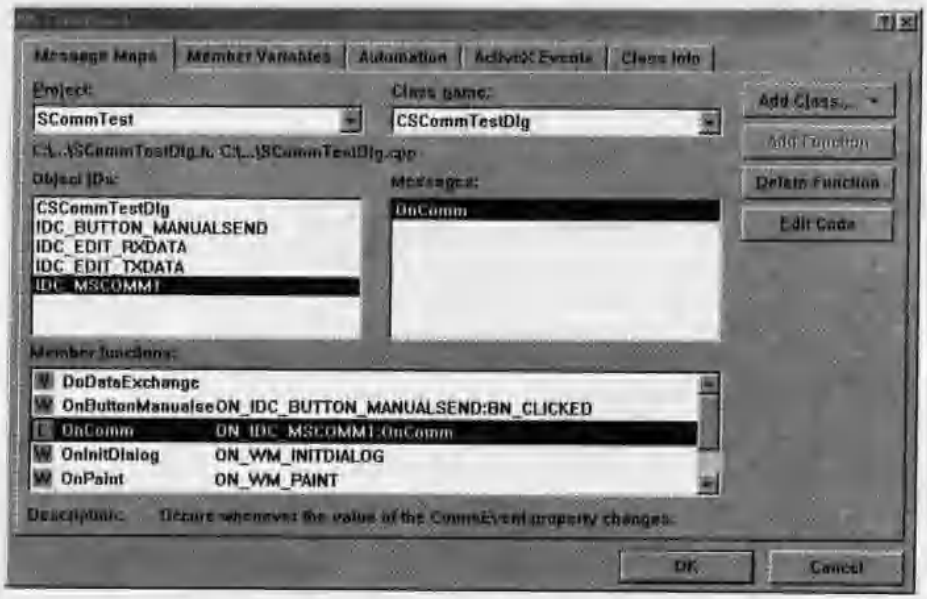

图 1.2.7　为控件 IDC_MSCOMM1 添加消息事件处理函数 OnComm()

下面编写函数 OnComm()中的代码，主要任务是从串口接收数据并显示在接收编辑框中。

```
void CSCommTestDlg::OnComm()
{
    // TODO: Add your control notification handler code here
    VARIANT variant_inp;
    COleSafeArray safearray_inp;
    LONG len,k;
    BYTE rxdata[2048]; //设置 BYTE 数组
    CString strtemp;
    if(m_ctrlComm.GetCommEvent()==2) //事件值为 2 表示接收缓冲区内有字符
    {
        variant_inp=m_ctrlComm.GetInput(); //读缓冲区
        safearray_inp=variant_inp; //VARIANT 型变量转换为 ColeSafeArray 型变量
        len=safearray_inp.GetOneDimSize(); //得到有效数据长度
        for(k=0;k<len;k++)
            safearray_inp.GetElement(&k,rxdata+k);//转换为 BYTE 型数组
        for(k=0;k<len;k++)                    //将数组转换为 Cstring 型变量
        {
            BYTE bt=*(char*)(rxdata+k);    //字符型
            strtemp.Format("%c",bt);    //将字符送入临时变量 strtemp 存放
            m_strEditRXData+=strtemp;  //加入接收编辑框对应字符串
        }
    }
```

```
    UpdateData(FALSE);  //更新编辑框内容
}
```

5. 发送数据

先为发送按钮添加一个单击消息，即 BN_CLICKED 处理函数，打开 ClassWizard→Message Maps，选择类 CSCommTestDlg，选中 IDC_BUTTON_MANUALSEND，双击 BN_CLICKED 添加 OnButtonManualsend()函数，如图 1.2.8 所示。

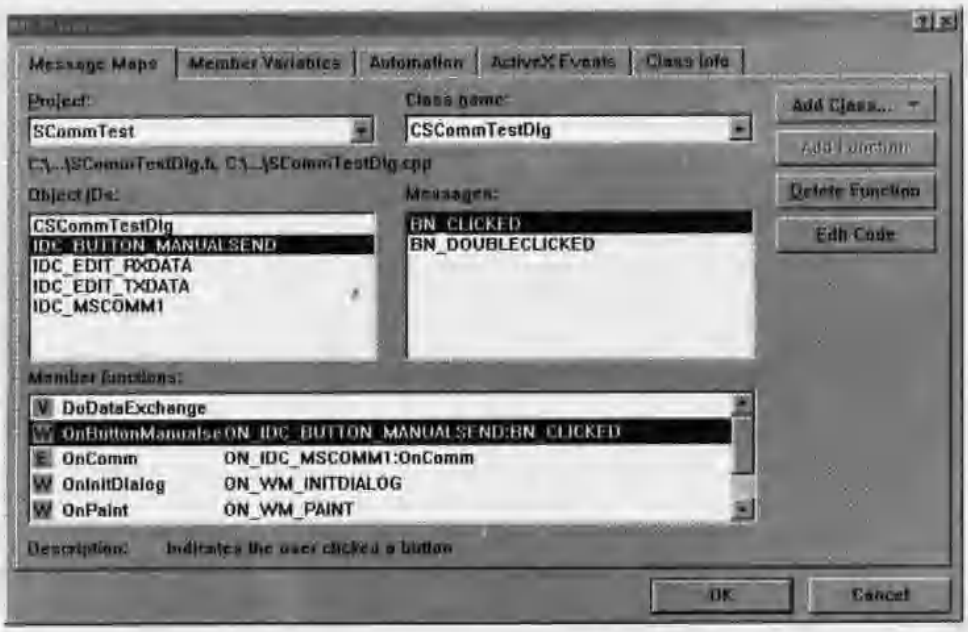

图 1.2.8　添加 OnButtonManualsend()函数

然后，在函数中添加如下代码：

```
void CSCommTestDlg::OnButtonManualsend()
{
  // TODO: Add your control notification handler code here
  UpdateData(TRUE); //读取编辑框内容
  m_ctrlComm.SetOutput(COleVariant(m_strTXData));//发送数据
}
```

全部程序编写完成后，SCommTestDlg.h 文件代码如下：

```
// SCommTestDlg.h : header file
//
//{{AFX_INCLUDES()
#include "mscomm.h"
//}}AFX_INCLUDES

#if !defined(AFX_SCOMMTESTDLG_H__22F2B146_69C2_11D5_870E_00E04C3F78CA__INCLUDED_)
#define
AFX_SCOMMTESTDLG_H__22F2B146_69C2_11D5_870E_00E04C3F78CA__INCLUDED_
```

```
#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

/////////////////////////////////////////////////////////////////////////////
// CSCommTestDlg dialog

class CSCommTestDlg : public CDialog
{
// Construction
public:
    CSCommTestDlg(CWnd* pParent = NULL);  // standard constructor

// Dialog Data
    //{{AFX_DATA(CSCommTestDlg)
    enum { IDD = IDD_SCOMMTEST_DIALOG };
    CMSComm   m_ctrlComm;
    CString   m_strEditRXData;
    CString   m_strEditTXData;
    //}}AFX_DATA

    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CSCommTestDlg)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    //}}AFX_VIRTUAL

// Implementation
protected:
    HICON m_hIcon;

    // Generated message map functions
    //{{AFX_MSG(CSCommTestDlg)
    virtual BOOL OnInitDialog();
    afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
    afx_msg void OnPaint();
    afx_msg HCURSOR OnQueryDragIcon();
    afx_msg void OnComm();
    afx_msg void OnButtonManualsend();
    DECLARE_EVENTSINK_MAP()
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous
line.

#endif
// !defined(AFX_SCOMMTESTDLG_H__22F2B146_69C2_11D5_870E_00E04C3F78CA__INCLUDED_)
```

需要两个串口来测试程序。这两个串口可以在一台计算机上，也可以分别在两台计算机上，用串口线将其连接。图 1.2.9 所示的情况为运行本程序控制 COM1，在发送编辑框中输入 1234567890ABCDEFG，再打开串口调试助手控制 COM2，单击“发送”按钮，再单击串口调试助手的“手动发送”按钮，程序运行正常。

图 1.2.9　利用串口调试助手测试串口接收与发送功能

图 1.2.10 所示的测试情况为先将串口号设置为 COM1，运行程序；再将语句：

```
m_ctrlComm.SetCommPort(1); //COM1
```

修改为：

```
m_ctrlComm.SetCommPort(2); //COM2
```

再运行程序，并在两个程序的“发送输入”框中分别填上“串口 1：MSComm 控件测试程序”和“串口 2：MSComm 控件测试程序”，单击“发送”按钮。当然，如果在界面上有串口选择功能，就不必那么麻烦了。

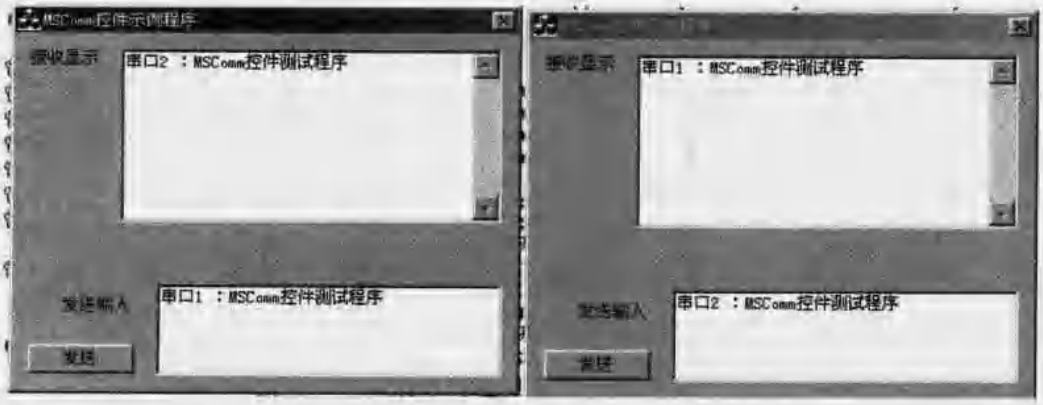

图 1.2.10　利用程序本身进行测试

好的，我们的第一个串口程序就成功运行了，在高兴的同时，还要注意，这只不过是初步的东西，我们真正要做的是如何把数据取出来，把一定格式的命令字符或数据发送出去。编程也是初步的，以后还有一些深层的知识需要了解。

1.3 体验 DOS 环境下 Turbo C 串口通信编程

目前，单片机和嵌入式系统的应用越来越广，且编程语言绝大部分为 C 语言，其编程方式大多与 DOS 环境下 C 语言编程大同小异，所以，了解 DOS 下串口通信编程方式也是十分必要的，这样，我们才可以不惧任何编程环境的挑战了。

在本书中，使用 Turbo C++ 3.0 编译器来调试程序，目前使用较多的是 Turbo C 2.0 和 Turbo C++ 3.0，推荐使用后者。Turbo C++ 3.0 不仅编程界面比 Turbo C 2.0 友好不少，可以使用鼠标，而且 C 语句与 C++语句兼容，如可以在编程中使用类，这就能让我们应用大量的程序源代码资源，这同时也是 C 语言的发展趋势，初学者更是可以通过它来学习 C++语言编程。不熟悉 Turbo C++3.0 编译环境的读者可以先看一下附录 A，了解 Turbo C++ 3.0 和 Turbo C 2.0 的使用方法及安装注意事项。

DOS 下串口通信属于底层编程，一般有中断和查询两种通信方式，作为初始的体验，在这个程序中用了中断方式。这个程序涉及许多硬件的设置问题，读者先不必做详细了解，第 6 章中会有详细介绍，这里只要体验一下 DOS 操作系统下的串口通信。

程序如下（源代码可在本书所附光盘的第 1 章找到：COMRX.CPP）：

```
//////////////////////////////////////////////////////////////
//COMRX.CPP  for asyn serial communication (only RX)
//edited by Xiong Guangming and Gong Jianwei
//Turbo C++3.0
//////////////////////////////////////////////////////////////
#include <stdio.h>
#include <dos.h>
#include <conio.h>

#define BUFFLEN 1024

void InitCOM();  //初始化串口
void OpenPort();  //打开串口
void ClosePort(); //关闭串口，释放串口资源
//新的中断函数，注意在TC2.0下，下面函数的...要去掉
void interrupt far  asyncint(...);
//中断向量：用于保存中断现场
void interrupt(*asyncoldvect){...};

unsigned char Buffer[BUFFLEN];
int buffin=0;
int buffout=0;
//unsigned char ch;

//COM1 产生的硬件中断号为 IQR4，对应的中断向量为 0CH
//打开 COM1
void OpenPort()
{
    unsigned char ucTemp;
    InitCOM();  //初始化串口
```

```
    //读入由参数给定的中断向量值，并将它作为中断函数的远地址
    asyncoldvect=getvect(0x0c);
    disable();      //关中断
    inportb(0x3f8);
    inportb(0x3fe);
    inportb(0x3fb);
    inportb(0x3fa);
    outportb(0x3fc,0x08|0x0b);
    outportb(0x3f9,0x01);
    ucTemp=inportb(0x21)&0xef;
    outportb(0x21,ucTemp);
    setvect(0x0c,asyncint);
    enable();       //开中断
}

//中断服务程序，从 COM1 接收数据
//注意在 TC2.0 下，下面函数的...要去掉
void interrupt far asyncint(...)
{
    //unsigned char ch;
    Buffer[buffin++] = inportb(0x3f8);// 读字符到缓冲区
    if (buffin >= BUFFLEN)  // 缓冲区满
        buffin=0;           // 指针复位
    outportb(0x20,0x20);
}

void ClosePort(void) //关闭中断
{
    disable();
    outportb(0x3f9,0x00);
    outportb(0x3fc,0x00);
    outportb(0x21,inportb(0x21)&0x10);
    enable();
    setvect(0x0c,asyncoldvect);
}

void InitCOM()// 对 COM1 串口初始化，设置串口参数
{

    outportb(0x3fb,0x80);  //将设置波特率
    /* 设置波特率，低位在前、高位在后；(部分波特率器参数如下)
   波特率    分频器 H   分频器 L
     ...
     4800      00       18H
     7200      00       10H
     9600      00       0CH
     ....
 */
    outportb(0x3f8,0x0C);   //波特率为 9600bps
    outportb(0x3f9,0x00);

    /*设置停止位、奇偶校验位、等
    D7：为 1 表示设置波特率；为 0，其他；
    D6：为 1，是，强迫在数据线上输出逻辑 0；若为 0，则否；
    D5：1，校验位可变；0，不变；
    D4D3：×0，无校验位；01，奇校验位；11，偶校验位；
    D2：0,1 个停止位；1，1.5 个停止位；
    D1D0：00，5 位数据位；01，6 位数据位；10，7 位数据位；11，8 位数据位；
```

```
    */
    outportb(0x3fb,0x03);   //8 个数据位，1 个停止位，无奇偶校验

    outportb(0x3fc,0x08|0x0b);
    outportb(0x3f9,0x01);
}


unsigned char read_char(void)
{
    unsigned unch;
    if(buffout != buffin)
    {
        unch = Buffer[buffout];
        buffout++;
        if(buffout >= BUFFLEN)
            buffout=0;
        return(unch);
    }
    else
        return(0xff);
}

//以下为主函数
void main()
{
    unsigned char unChar;
    short bExit_Flag=0;

    OpenPort(); //打开串口

    fprintf(stdout, "\n\nReady to Receive DATA\n"
            "press [ESC] to quit...\n\n");

    do {
        if (kbhit())
        {
            unChar=getch();
            /* Look for an ESC key */
            switch (unChar)
            {
            case 0x1B:  //ESC的ASCII值为27
                bExit_Flag = 1; /* Exit program */
                break;
            //You may want to handle other keys here
            }
        }
        unChar = read_char(); //从缓冲区中读数
        if (unChar != 0xff)
        {
            fprintf(stdout,"%c",unChar);
        }
    } while (!bExit_Flag);

    ClosePort(); //关闭串口
}
```

下面对程序进行测试。在 Turbo C++ 3.0 中编译运行程序后，得到了 comrx.exe 可执行文件。连接好串口线后（同一台计算机的两个串口或不同计算机的串口），首先在串口调试助

手中，将串口号设置为 COM2（串口 2）、波特率 9600bps、8 个数据位、1 个停止位、无奇偶校验，并在发送输入框中填入“12345678ABCDEFG”，后加换行（直接按回车键即可），选上“自动发送”，自动发送周期为 1000ms。

如图 1.3.1 所示，打开 MS-DOS 窗口（Windows 98 下，从开始→程序→MS-DOS 方式；Windows 2000 下，从开始→程序→附件→命令提示符），运行 comrx 就可以看到 COM1（串口 1）接收的数据了，也可以看到从串口调试助手发来的数据了。

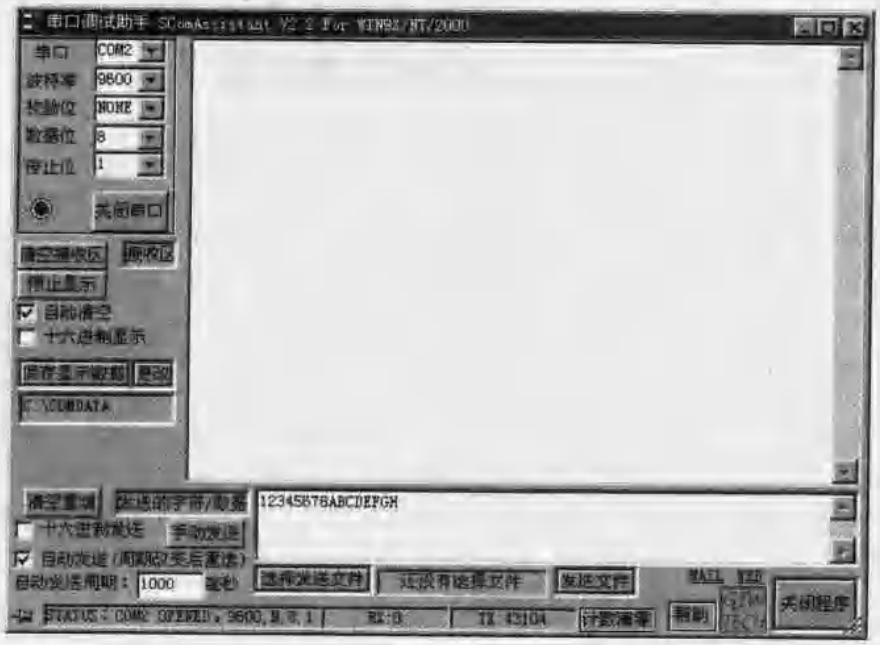

图 1.3.1 comrx 程序的测试情况

第2章　多线程串口编程工具CSerialPort类

[内容提要]

本章介绍一个非常好用的多线程串口编程工具CSerialPort类，用它可以很轻松地完成一般串口编程任务，几分钟就可搭好串口通信框架。编程者可以从烦心的框架编写中解脱出来，只需将精力放在通信协议的编制及数据处理上。初次接触VC串口编程的读者在用这个类做过程序后，一定会喜欢上它的，等熟悉底层Windows API编程后，我们还可以对这个类进行必要的改造，使之更加强健，帮助我们完成特定的任务。

和MSComm控件相比，这个类打包时，不需要加入其他的文件，而且函数都是开放透明的，允许我们进行改造，还有，它不需要我们去理解有些对于初学者较难掌握的数据类型，所以对初学者更适用。在本书里的例程里，也大量应用了这个类。

本章第2.1节里，对CSerialPort类的功能及成员函数进行了介绍，如果读者刚开始只想完成编程任务，则可以不看这一节，等到想对这个类进行改造或者以后学习用API函数来编写串口程序时，再来仔细看看。

在第2.2节和第2.3节中，分别用CSerialPort类做了基于对话框的例程和基于单文档的例程，一步一步地详细介绍了编程过程，相信即使对VC编程环境不太熟悉的读者，也会很快掌握这些内容。这里仍然可以通过串口调试助手来测试本章程序。

第2.4节是对CSerialPort类的改进。在实际应用中，发现了一些CSerialPort类中一些不完善的地方，笔者根据自己以及其他应用者的一些补充，对类的代码进行了改进。

2.1　CSerialPort类的功能及成员函数介绍

CSerialPort类是由Remon Spekreijse提供的免费串口类，作者还为这个类制作了一个例程，原类的地址为http://www.codeguru.com/network/serialport.shtml，Codeguru是一个非常不错的源代码网站，编程的朋友们应该常去看看。

CSerialPort类支持线连接（非MODEM）的串口编程操作，编写的程序在Windows 98/NT/2000/XP操作系统下可很好地运行，但在Windows Me操作系统下会出现死机的现象，作者没有仔细探究过这个问题，这也许是Windows Me对CSerialPort类中用到的API函数不兼容所致(想当然了)。

CSerialPort类是基于多线程的，其工作流程如下：首先设置好串口参数，再开启串口监测工作线程，串口监测工作线程监测到串口接收到的数据、流控制事件或其他串口事件后，就以消息方式通知主程序，激发消息处理函数来进行数据处理，这是对接收数据而言的；发送数据可直接向串口发送。

CSerialPort类定义的消息如表2-1-1所示。

表 2-1-1　类消息说明

消息名称	消息号	功能说明
WM_COMM_BREAK_DETECTED	WM_USER+1	检测到输入中断
WM_COMM_CTS_DETECTED	WM_USER+2	检测到 CTS (清除发送)信号状态改变
WM_COMM_DSR_DETECTED	WM_USER+3	检测到 DSR (数据设备准备就绪)信号状态改变
WM_COMM_ERR_DETECTED	WM_USER+4	发生线状态错误（包括 CE_FRAME, CE_OVERRUN, 和 CE_RXPARITY）
WM_COMM_RING_DETECTED	WM_USER+5	检测到响铃指示信号
WM_COMM_RLSD_DETECTED	WM_USER+6	检测到 RLSD (接收线信号)状态改变
WM_COMM_RXCHAR	WM_USER+7	接收到一个字符并已放入接收缓冲区
WM_COMM_RXFLAG_DETECTED	WM_USER+8	检测到接收到字符（该字符已放入接收缓冲区）事件
WM_COMM_TXEMPTY_DETECTED	WM_USER+9	检测到发送缓冲区最后一个字符已经被发送

这里先重点介绍几个经常要用到的函数：

1．串口初始化函数 InitPort

这个函数是用来初始化串口的，即设置串口的通信参数：需要打开的串口号、波特率、奇偶校验方式、数据位、停止位，这里还可以用来进行事件的设定。

```
BOOL CSerialPort::InitPort(CWnd* pPortOwner,    // 所属主窗口 the owner (CWnd) of the port
                                                //(receives message)
          UINT  portnr,                         // 串口号 portnumber
          UINT  baud,                           // 波特率 baudrate
          char  parity,                         // 校验方式 parity
          UINT  databits,                       // 数据位 databits
          UINT  stopbits,                       // 停止位 stopbits
          DWORD dwCommEvents,                   // EV_RXCHAR、EV_CTS 事件设置
    UINT  writebuffersize)                  // 设置发送缓冲区大小 size to the writebuffer
```

如果串口初始化成功，就返回.TRUE，若串口被其他设备占用、不存在或存在其他故障，就返回 FALSE，编程者可以在这儿提示串口操作是否成功。

如果在当前主窗口中调用这个函数，那么 pPortOwner 可用 this 指针表示。串口号在函数中做了限制，只能用 1，2，3 和 4 四个串口号，而事实上在编程时可能用到更多串口号，可以通过注释掉本函数中的“assert(portnr > 0 && portnr < 5);”语句取消对串口号的限制。

2．启动串口通信监测线程函数 StartMonitoring()

串口初始化成功后，就可以调用 BOOL StartMonitoring()来启动串口监测线程。线程启动成功，返回 TRUE。

```
// start comm watching
BOOL CSerialPort::StartMonitoring()
{
    if (!(m_Thread = AfxBeginThread(CommThread, this)))
        return FALSE;
    TRACE("Thread started\n");
    return TRUE;
}
```

调用 InitPort()和 StartMonitoring()后，串口就被打开，各种串口状态和事件就可以被监测到。