

Delphi 例程说明

Mscomm 是微软提供给用户使用的,与串口设备进行通信的组件。本说明讲述如何使用 **Mscomm** 组件进行编程,实现与大连理工计算机控制工程有限公司 (简称: **DCCE**) 的 **PLC** 设备通信。主要包含 **Mscomm** 组件概述、**modbus** 协议介绍和 **Delphi** 下使用 **Mscomm** 等方面的内容。

环境准备

硬件需求: 大连理工计算机控制工程有限公司的 **PLC** 设备、有串口通信的计算机、串口数据线, **RS232 RS485** 转换接头。

软件需求: **Windows98/2000/XP** 操作系统, 大连理工计算机控制工程有限公司 **PLC** 调试软件, **Delphi2005**。

准备流程:

1. 将设备通过 **RS232-RS485** 转换器以调试状态 (首先短接调试端与地,然后连接电源) 连接在计算机上, 运行与设备型号相对应的 **PLC** 调试软件设置串口参数, 由于本例程设置 **Mscomm** 校验位, 数据位, 停止位, 波特率参数默认为 **N**、**8**、**1**、**9600**, **ModBus** 协议模块地址默认为 **1**, 所以将设备的校验位, 数据位, 停止位, 波特率分别设置为: 无校验, **8**, **1**, **9600**. 模块地址设置为 **1**, 同时要把将要与例程通信的端口设置为从口。最后要保存参数。
2. 设备断电, 将短接线移去, 重新通电, 直接运行本例程进行测试。

Mscomm 概述

Microsoft Communications Control (以下简称 **MSComm**) 是 **Microsoft** 公司提供的简化 **Windows** 下串行通信编程的 **ActiveX** 控件, 它为应用程序提供了通过串行接口收发数据的简便方法。**MSComm** 控件在串口编程时非常方便, 程序员不必去花时间去了解较为复杂的 **API** 函数, 而且在 **VC**、**VB**、**Delphi** 等语言中均可使用。具体的来说, 它提供了两种处理通信问题的方法: 事件驱动(**Event-driven**)方法和查询法。

事件驱动通讯是处理串行端口交互作用的一种非常有效的方法。在许多情况下, 在事件发生时需要得到通知, 例如, 在串口接收缓冲区中有字符, 或者 **Carrier Detect (CD)** 或 **Request To Send (RTS)** 线上一个字符到达或一个变化发生时。在这些情况下, 可以利用 **MSComm** 控件的 **OnComm** 事件捕获并处理这些通讯事件。**OnComm** 事件还可以检查和处理通讯错误。所有通讯事件和通讯错误的列表, 参阅 **CommEvent** 属性。在编程过程中, 就可以在 **OnComm** 事件处理函数中加入自己的处理代码。这种方法的优点是程序响应及时, 可靠性高。每个 **MSComm** 控件对应着一个串行端口。如果应用程序需要访问多个串行端口, 必须使用多个 **MSComm** 控件。

常用属性

MSComm 编程序必须了解的属性:

属性	功能
CommPort	设置并返回通讯端口号，缺省为 COM1
Settings	以字符串的形式设置并返回波特率、奇偶校验、数据位、停止位
PortOpen	设置并返回通讯端口的状态。也可以打开和关闭端口
Input	从接收缓冲区返回和删除字符
Output	向传输缓冲区写一个字符串
InputLen	设置每次 Input 读入的字符个数，缺省值为 0，表明读取接收缓冲区中的全部内容
InBufferCount	返回接收缓冲区中已接收到的字符数，将其置 0 可以清除接收缓冲区
InputMode	定义 Input 属性获取数据的方式（为 0：文本方式；为 1：二进制方式）
RThreshold	和 SThreshold 属性，表示在 OnComm 事件发生之前，接收缓冲区或发送缓冲区中可以接收的字符数

CommPort 属性设置或返回通讯端口号。在设计时，value 可以设置成从 1 到 16 的任何数（缺省值为 1）。但是如果用 PortOpen 属性打开一个并不存在的端口时，MSComm 控件会产生错误 68（设备无效）。必须在打开端口之前设置 CommPort 属性。

Settings 属性设置或返回串口波特率、奇偶校验、数据位、停止位参数。参数格式为 "BBBB,P,D,S" ;其中,BBBB 为波特率, P 为奇偶校验, D 为数据位数, S 为停止位数。value 的缺省值是: "9600,N,8,1"。其它效验字符为: 奇效验----‘O’;偶效验----‘E’;

InputMode 属性设置或返回通信方式。comInputModeText(0, 缺省值) 通过 Input 属性以文本方式取回数据。comInputModeBinary(1) 通过 Input 属性以二进制方式检取回数据。

RThreshold 属性在 MSComm 控件发送数据前设置为要接收的字符数。当接收字符后，若 Rthreshold 属性设置为 0（缺省值）则不产生 OnComm 事件。若设置 Rthreshold 为 n，接收缓冲区收到 n 个字符都会使 MSComm 控件都将产生 OnComm 事件。

InputLen 属性设置或返回 Input 属性从接收缓冲区读取的字符数。Input 属性从接收缓冲区中读取的字符数。InputLen 属性的缺省值是 0。设置 InputLen 为 0 时,使用 Input 将使 MSComm 控件读取接收缓冲区中全部的内容。若接收缓冲区中 InputLen 字符无效，Input 属性返回一个零长度字符串 ("")。

InBufferCount 属性为缓冲区中已接收的字符数。该属性在从输出格式为定长数据的机器读取数据时非常有用。

使用方法

使用 MSComm 控件的一般使用方法为:

初始化串口: 通讯端口号 (CommPort)、串口配置(Settings)。打开串口 (PortOpen)。设置通信方式 (InputMode = 1)。

发送请求数据: 清空缓冲区(InBufferCount = 0)、设置读取的长度 (InputLen = 0)、设置触发事件的接收长度 (Rthreshold = n)、设置发送内容 (Output)。

接收事件处理： 取回串口中的字符长度(InBufferCount)、 取回返回的数据 (Input)。

Modbus 协议介绍

通讯命令

Modbus 协议是一种通信标准,包含 RTU 和 ASCII。我们只需要了解 RTU 的读写命令及其打包过程。与我们编程相关的 Modbus 协议大致格式是: 站地址 + 命令号 + 起始地址 + 操作数量 + 数据段 + 效验码。其回复格式为: 站地址 + 命令号 +其它。

站地址一个 0~255 的设备标号,占用一个字节。命令号决定设备如何操作,如读位,写字,读寄存器,写寄存器等。起始地址是指寄存器在设备的地址编码,如 VW0 的绝对地址是 2336。操数量,是指操作(读或写)的单元个数。数据段只有写操作有,包含数据的字节长度和数据内容。 效验码用于检验传输数据的正确性。他们各自的命令格式及其回复见表 1 和表 2。

表 1 Modbus 读写命令格式

操作(命令号)	命令格式	字节数	举例（十六进制表示）
位 读 取 (01/02)	站地址(1) 功能号(1) 起始地址(2) 数量 (2) CRC(2)	8	01 01 00 01 00 02 CRC
字 读 取 (03/04)	站地址(1) 功能号(1) 起始地址(2) 数量 (2) CRC(2)	8	01 03 00 01 00 03 CRC
位写入(15)	站地址(1) 功能号(1) 起始地址(2) 操作位 数(2) 数据字节数(1) 数据(n) CRC(2)	9+ n	01 0F 00 13 00 0A 02 CD 01 CRC
字写入(16)	站地址(1) 功能号(1) 起始地址(2) 操作字 节数(2) 数据字节数(1) 数据(n) CRC(2)	9+ n	01 10 00 01 00 02 04 00 0A 01 02 CRC

表 2 Modbus 协议读写返回格式

操作(命令号)	返回数据格式	字节数	举例（十六进制表示）
位 读 取 (01/02)	站地址(1)-功能号(1)-字节数(1)-数 据(n)-CRC(2)	5+n	01 01 03 01 00 02 CRC
字 读 取 (03/04)	站地址(1)-功能号(1)-字节数(1)-数 据(n)-CRC(2)	5+n	01 03 04 01 00 03 01 CRC

位写入(15)	站地址(1)-功能号(1)-起始地址(2)- 8 操作位数(2)-CRC(2)	01 0F 00 13 00 0A CRC
字写入(16)	站地址(1)-功能号(1)-起始地址(2)- 8 操作字节数(2)-CRC(2)	01 10 00 01 00 02 CRC

数据格式

在读写过程中并不能直接传输数据。位数据需要将各个位从低到高的顺序排列。如读取v0.0-v0.4,返回的数据顺序是: (字节高端)-> v0.4 v0.3 ... v0.1 ->(字节低端)。如果返回的字节为5,二进制为"0000 0101",那么V0.0和V0.2为1。V0.1、V0.3和V0.4为0。写入数据顺序与此相同。

字数据需要将数据按照字节进行交换。如图1所示。读取VW0返回的数据为: byte1 byte2,那么VW0的实际数据为 byte2 byte1。如果VW0表示的为一个word类型数据,那么它的真实值为: byte2+byte1*256,即交换为"byte2 byte1"后,强转为word型的值。VD0类似,需要将VW0和VW1分别交换后强转为响应的数据类型。程序中部分函数将改变这种字节顺序。这些都是由Modbus协议规定的。

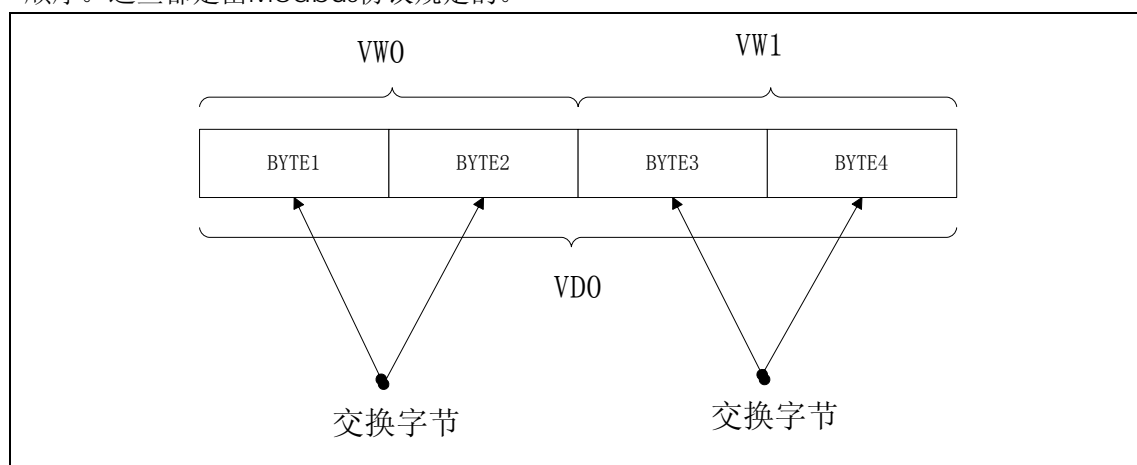


图 1 字节顺序

在 Delphi 中使用 Mscomm

我使用的编辑器是 Delphi 7.0。打开 Delphi7.0 程序,自动生成了一个 Form 程序框架。

添加 Activex(Mscomm32.ocx)组件

点工具栏上的 Component->"Import Activex Controls"(如图 2 所示)。在弹出对话框中点"Add"按钮,在 C:\windows\system32 中找到并选中 Mscomm32.ocx。点"Install"

->"确定"->"Yes"。这时在工具条的"Activex"页就增加了像电话一样的图标(如图 3 所示)。我可以将其拖拽到对话框上,进行相关编程。

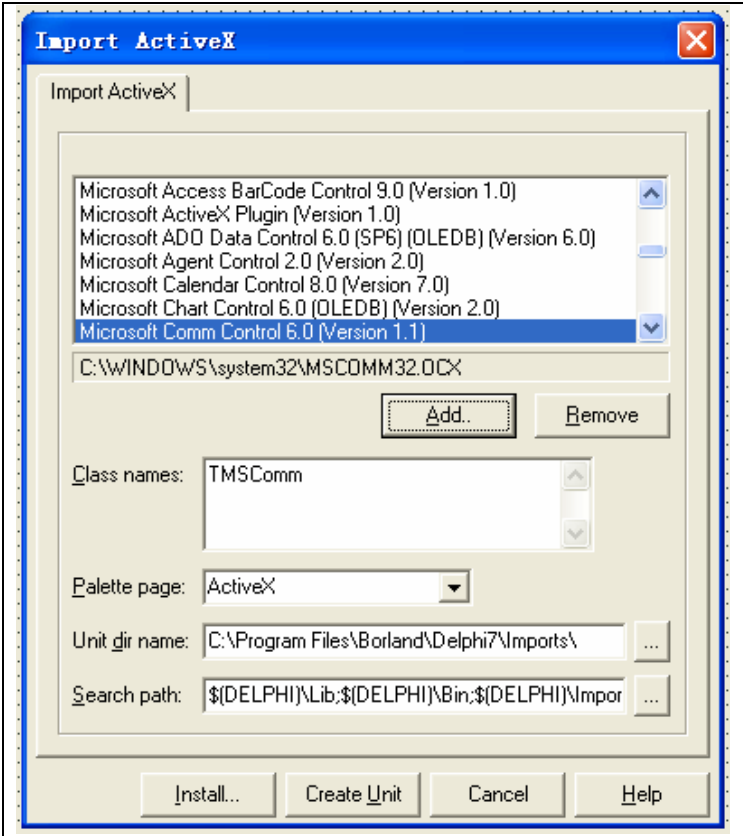


图 2 导入 Mscomm 组件

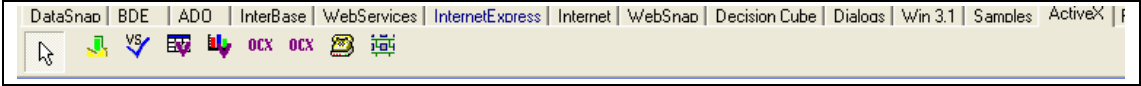


图 3 Activex 页面控件

添加初始化代码

然后向界面添加必要的元素。如图所示。



图 4 程序界面

在 Form 类定义中添加以下成员变量和方法。

```
{ Private declarations }
m_bComm : bool;           //是否正在通信中(通信锁)
m_bRedOpen : bool;        //红灯是否已经打开
m_bGreenOpen : bool;      //绿灯是否已经打开
m_nAddr : integer;        //此次通信操作的起始地址
function GetMyCrc( szSend : OleVariant; nLen : integer ) : integer; // 计 算
Crc 效验码
function GetLongInt( data : OleVariant; nIndex : integer ) : LongInt; //获取双字
并转为整形
function GetInteger( data : OleVariant; nIndex : integer ) : integer; //获取字并
转为整形
function GetSingle( data : OleVariant; nIndex : integer ) : Single; //获取双字并
转为浮点数
```

在 Form 空白处双击,进入 FormCreate 函数,在此函数中进行一些初始化。添加如下代码:

```
m_bRedOpen := false;      //初始化红灯为关闭状态
m_bGreenOpen := false;    //初始化绿灯为关闭状态
m_bComm := false;         //初始化未通信状态
m_nAddr := 0;             //初始化当前通信的操作起始地为 0.

RichEdit_IntegerEdit.Text := ""; //初始化 RichEdit 控件显示为空
RichEdit_IntegerShow.Text := "";
RichEdit_RegisterEdit.Text := "";
RichEdit_RegisterShow.Text := "";
RichEdit_FloatEdit.Text := "";
RichEdit_FloatShow.Text := "";
```

```

RichEdit_Ret.Text := "";

MsComm1.CommPort := 1;           //初始化通信端口号为 1
if (MsComm1.PortOpen <> true) then //
begin
    MsComm1.PortOpen := true;      //打开端口
    MsComm1.Settings := '9600,n,8,1'; //设置端口配置
    MsComm1.InputMode := 1;        //设置返回数据格式为二进制
end;

timer_ReadRed.Interval := 2000;    //启动读红灯位定时器
timer_ReadRed.Enabled := true;
Timer_ReadGreen.Interval := 2030;  //启动读绿灯位定时器
Timer_ReadGreen.Enabled := true;
Timer_ReadRegister.Interval := 2010; //启动读寄存器(2337)定时器
Timer_ReadRegister.Enabled := true;
timer_ReadInteger.Interval := 2040; //启动读整形 (2338)定时器
timer_ReadInteger.Enabled := true;
timer_ReadFloat.Interval := 1900;   //启动读浮点数 (2340)定时器
timer_readFloat.Enabled := true;

```

在其下方添加方法实现代码如下。GetMyCrc 计算 Crc 效验码。计算 szSend 中前 nLen 个字节的 CRC 值。

```

function TForm1.GetMyCrc( szSend : OleVariant; nLen : integer ):integer;
var i: integer;
    j: integer;
    CrcValue :integer;
BEGIN
    CrcValue := 65535;
    For i := 0 To nLen - 1 do
    begin
        CrcValue := szSend[i] Xor CrcValue;
        For j := 0 To 7 do
        begin
            if ( (CrcValue And 1) <> 0 ) then
            begin
                CrcValue := CrcValue div 2;
                CrcValue := CrcValue Xor 40961;
            end
            Else
            begin
                CrcValue := CrcValue div 2;
            end
        end
    end;
end;

```

```
end ;  
Result := CrcValue;  
END;
```

GetLongInt 函数是将 data 中第 nIndex(从 0 开始计算)字节开始的连续 4 个字节交换顺序后强制转为 LongInt 类型返回。GetInteger 函数是将 data 中第 nIndex 字节开始连续 2 个字节交换顺序后强制转为 Integer 类型返回。GetSingle 函数是将 data 中的第 nIndex 字节开始连续 4 个字节交换字节顺序后强制转为 Single 类型返回。如需要读取 VD1,且 VD1 的逻辑意义是一个浮点数,那么可以发送读 VD1 的请求。而 VD1 的数据保存在返回数据的第 3~6 个字节中。可以通过 GetSingle(data,3)取得该浮点数。其它类同。

```
// 获取整数(长整型)  
function TForm1.GetLongInt( data : OleVariant; nIndex : integer ) : LongInt;  
var nL : LongInt;  
    bTemp : array[0..3] of byte;  
begin  
    bTemp[0] := data[nIndex+1];  
    bTemp[1] := data[nIndex];  
    bTemp[2] := data[nIndex+3];  
    bTemp[3] := data[nIndex+2];  
    CopyMemory( @nL ,@bTemp[0], 4);  
    Result := nL;  
end;
```

```
// 获取整数(单个寄存器)  
function TForm1.GetInteger( data : OleVariant; nIndex : integer ) : integer;  
var  
    nR : integer;  
    bTemp : array[0..3] of byte;  
begin  
    bTemp[0] := data[nIndex+1];  
    bTemp[1] := data[nIndex];  
    CopyMemory( @nR, @bTemp[0], 2 );  
    Result := nR;  
end;
```

```
// 获取浮点数  
function TForm1.GetSingle( data : OleVariant; nIndex : integer ) : Single;  
var f :Single;  
    bTemp : array[0..3] of byte;  
begin  
    bTemp[0] := data[nIndex+1];  
    bTemp[1] := data[nIndex];  
    bTemp[2] := data[nIndex+3];
```



```
bTemp[3] := data[nIndex+2];
CopyMemory( @f,@bTemp[0],4);
Result := f;
end;
```

发送读取请求

到此,我们的准备工作已经完成。需要的是将通信添加到程序。通信之前,必须知道通信的内容是什么?然后是怎么通信的问题。通信的数据就是 Modbus 协议内容。更直白点就是对串口设备的读写请求及其返回。以读寄存器为例,我们需要操作的地址是 VW1 (绝对地址为 2336)。那么要读取这个地址的命令就是“?? 03 09 26 00 01 CRC”,写入的命令是“?? 16 09 26 00 01 02 ** ** CRC”。其中第一个字节“??”表示站地址,第二个字节“03”和“16”表示读写寄存器命令号,第三个和第四个字节“09 26”为寄存器地址的十六进制表示,十进制即为 2336。接着“00 01”表示读取的长度。写命令紧跟着的是写入数据的字节长度和写入的数据。最后读写命令都需要加入 CRC 效验码。

读: 01 03 09 26 00 01 66 5D

写: 01 16 09 26 00 01 02 (00 10) F9 A5 括号中的表示写入的实际数据。

他们的返回为:

读: 01 03 02 ** ** CRC

写: 01 16 09 26 00 00 CRC

如对 Modbus 协议还不是很清楚,请参阅 Modbus 协议介绍。其它寄存器的读写请求和返回类似。

寄存器的值存放在 PLC 设备中,它不会主动告诉程序,所以程序需要不断的读取来保证数据同步,这里采用 Windows 定时器每隔一段时间读一次。当用户点写入时,发送写入请求。开关变量、寄存器读写过程相同,下面以寄存器读写举例。

双击寄存器读写定时器,进入其处理函数。在其中添加以下代码:

```
var
    sendByte : OleVariant;           //发送的数据变量
    nCrc : integer;                  //保存 Crc 的变量
begin
    if m_bComm then                  //判断是否正在通信
    begin
        exit;                        //存在其它通信,退出
    end;
    m_bComm := true;                 //锁定通信
    sendByte := VarArrayCreate([0,7],varByte); //开辟发送数据空间

    sendByte[0] := 1;                //站地址
    sendByte[1] := 3;                //命令号
    sendByte[2] := 2337 div 256;      //起始地址高位
```

```

sendByte[3] := 2337 mod 256;      //起始地址低位
sendByte[4] := 0;                //读取个数高字节
sendByte[5] := 1;                //读取个数低字节

nCRC := GetMyCRC( sendByte, 6);   //计算 CRC
sendByte[6] := nCRC mod 256;      //CRC 低字节
sendByte[7] := nCRC div 256;      //CRC 高字节

MSComm1.InputMode := comInputModeBinary; //设置返回数据为二进制
MSComm1.InputLen := 0;             //设置读取全部缓冲区
MSComm1.InBufferCount := 0;        //清空缓冲区
MSComm1.RThreshold := 7;           //设置触发事件接收字节
长度
MSComm1.Output := sendByte;        //发送数据
m_nAddr := 2337;                   //保存操作地址(读命令返回无此信息)

Timer_OverTime.Interval := 200;    //启动超时定时器
Timer_OverTime.Enabled := true;

```

写入过程类似,双击“写入”按钮,进入其处理函数,添加以下代码。

```

var
  sendByte : OleVariant;           //发送的数据变量
  nCRC : integer;                  //保存 CRC 的变量
  nValue : integer;                //写入值
begin
  if ( m_bComm or (RichEdit_RegisterEdit.Text = "") ) then //检查是否存在通信或写入值为空
  begin
    exit;                          //不满足写入条件,退出。
  end;
  m_bComm := true;                 //锁定通信
  sendByte := VarArrayCreate([0,10],varByte); //开辟写入数据空间

  sendByte[0] := 1;                //站地址
  sendByte[1] := 16;               //命令号
  sendByte[2] := 2337 div 256;      //操作地址高字节
  sendByte[3] := 2337 mod 256;     //操作地址低字节
  sendByte[4] := 0;                //操作个数高字节
  sendByte[5] := 1;                //操作个数低字节
  sendByte[6] := 2;                //写入数据的字节长度

  nValue := strToInt(RichEdit_RegisterEdit.Text); //取得写入的数

  sendByte[7] := nValue div 256;    //写入数的高字节

```

```

sendByte[8] := nValue mod 256; //写入数的低字节

nCrc := GetMyCrc( sendByte, 9);    //取得 Crc 效验码
sendByte[9] := nCrc mod 256;    //Crc 低字节
sendByte[10] := nCrc div 256;    //Crc 高字节

MSComm1.InputMode := comInputModeBinary; //设置返回数据为二进制
MSComm1.InputLen := 0;                    //设置读取全部缓冲区数据
MSComm1.InBufferCount := 0;                //清空缓冲区
MSComm1.RThreshold := 8;                  //设置触发事件接收字节长度

MSComm1.Output := sendByte;                //发送数据
m_nAddr := 2337;                          //保存操作起始地址

Timer_OverTime.Interval := 200;           //启动超时定时器
Timer_OverTime.Enabled := true;

```

接收返回处理

读取和写入都有了,还缺少的是接收处理。接收处理函数是由 Mscomm 触发。在界面中选中 Mscomm 组件图标,在属性中选中事件页,有一个 OnComm 属性,在其右输入你的处理方法,如这里输入“OnCommEven”,并算计它。程序自动跳转到处理函数处。添加以下代码。

```

procedure TForm1.OnCommEven(Sender: TObject);
var  i:integer;                //临时变量
nLen : integer;                //接收数据的长度
reData:array of Variant;       //接收的数据
restr : string;                //接收数据的字符串表示
begin
  Timer_OverTime.Enabled := false; //结束超时定时器
  m_bComm := false;              //通信结束,解通信锁
  if ( MSComm1.CommEvent <> 2 ) then //判断 Mscomm 组件事件是否是接收事件。
  begin
    exit;                        //不是接收事件退出
  end;
  nLen := MSComm1.InBufferCount; //获取接收数据长度
  redata := MSComm1.Input;        // 接收数据
  restr:="";
  // 此处必须是 inputLen 和 RThreshold 及 vararrayhighbound(redata,1) 相等
  if ( MSComm1.RThreshold = nLen ) then
  begin
    for i:=0 to vararrayhighbound(redata,1) do // 打印数据
      restr:=restr + inttohex(redata[i],2)+' ';

```

```

    RichEdit_Ret.Text := restr;           //显示到 RichEdit 中
    if ( (redata[1] = 1) or (redata[1] = 2)) then    // 读位请求返回
    begin
        BitReadRet(redata);           //读位返回处理
    end
    else if ( (redata[1] = 3) or (redata[1] = 4) ) then    // 读字返回
    begin
        WordReadRet(redata);           //读字返回处理
    end
    else if (redata[1] = 15 ) then           // 写多位返回
    begin
    end
    else if (redata[1] = 16 ) then           // 写多字返回
    begin
    end
    else           // 其它情况 ,可能包含 5,6 命令,不建议使用 5 和 6 号命令
    begin
    end;
end;
m_nAddr := 0;
End;

```

WordReadRet 函数代码如下:

```

// 字读取返回处理
procedure TForm1.WordReadRet( data : OleVariant ) ;
var  nValue : integer;           //保存整数数
     nLongValue : LongInt;       //保存长整数数
     fValue : Single;           //保存浮点数
     strShow :string;           //显示的字符串
begin
    if ( m_nAddr = 2337 ) then           //寄存器读返回处理
    begin
        nValue := GetInteger(data,3);    //获取整数数
        strShow := IntToStr(nValue);    //转为字符串
        RichEdit_RegisterShow.Text := strShow;    //显示
    end
    else if (m_nAddr = 2338 ) then           //整形读返回处理
    begin
        nLongValue := GetLongInt(data,3);
        strShow := intToStr(nLongValue);
        RichEdit_IntegerShow.Text := strShow;
    end
    else if ( m_nAddr = 2340) then           //浮点数读返回处理
    begin
        fValue := GetSingle(data,3);
    end
end

```

```
    strShow := FloatToStr(fValue);  
    RichEdit_FloatShow.Text := strShow;  
end  
else  
begin  
    end;  
end;
```

到这里,程序便可以读取寄存器的值了,其它位读取,整数,浮点数读取类似。