

VC 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 **VC** 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, **VC++6.0**。

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)的单元格式。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)。即是说返回的字节为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分别交换强转为响应的数据类型。程序中ChangeVar函数可以改变这种字节顺序。这些都是由Modbus协议规定的。

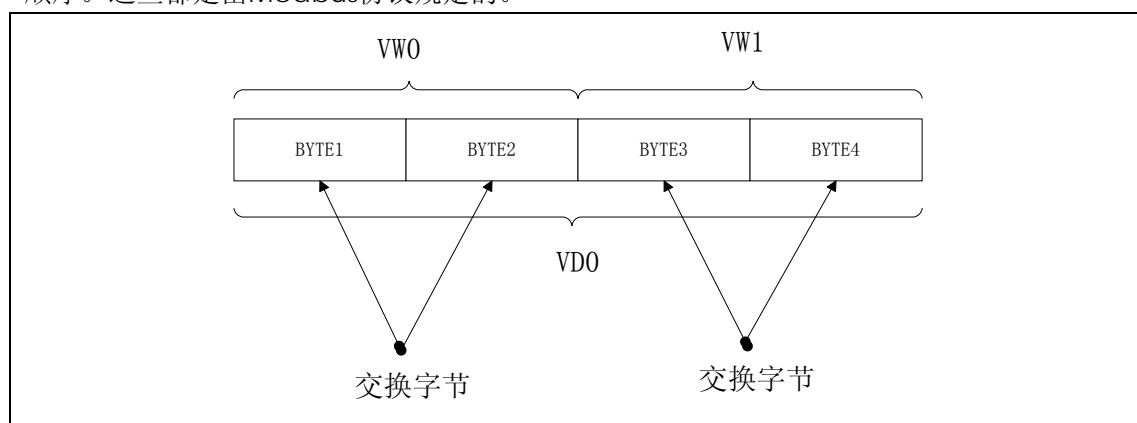


图 1 字节顺序

在 VC 中使用 Mscomm

本例程使用的 VC 编辑器是 VS6.0。VC2003 和 VS2005 的操作过程基本一致。

添加 Activex(mscomm32.ocx)组件

首先,新建一 MFC 对话框程序,在资源视图中,打开对话框编辑界面。在对话框界面上点右键,选择 "插入 ActiveX 控件",弹出 "插入 ActiveX 控件" 对话框(见图 1)。选中 "Microsoft Communications Control,version 6.0", 其路径为"C:\WINDOWS\SYSTEM32\MSCOMM32.OCX",点确定,此时在对话框界面中增加了

一个电话一样的控件 (见图 2)。在其上点右键选 “建立类向导”, 选中“Member variables” 页,如图 3 所示。双击“IDC_MSCOMM1”(可能不同,此值可以查看的控件的 ID 值),第一次添加该控件会弹出确认对话框,点确定。弹出“Confirms Classes”对话框(见图 4)。 可以修改类名称和相关文件,点确定。在弹出的“Add Member variable”对话框(见图 5)中输入变量名称 “m_comm”,点确定。此时,类视图中增加了一个 CMScomm 类(刚才输入类名),它包装了所有与 Mscomm 组件通信的接口(全部为方法)。我只需要在程序中调用这方法即可。注:这些方法一般都是以“Get/Set + 属性名”, 或者 “put/get + 下划线 + 属性名”的形式命名。调用相关方法时,请根据 VC6.0 生成的类中包含的方法为准。如我的 VC 生成类的方法为 put/get 方式。

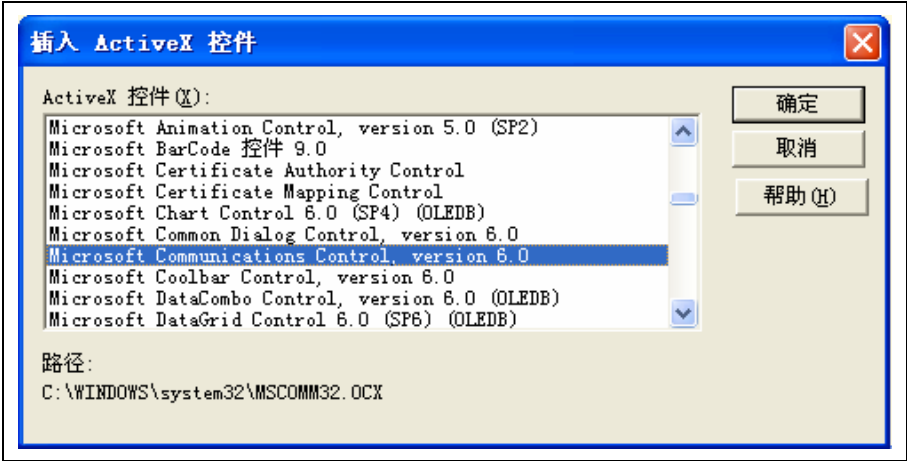


图 1 插入 Activex 控件对话框



图 2 Mscomm 组件

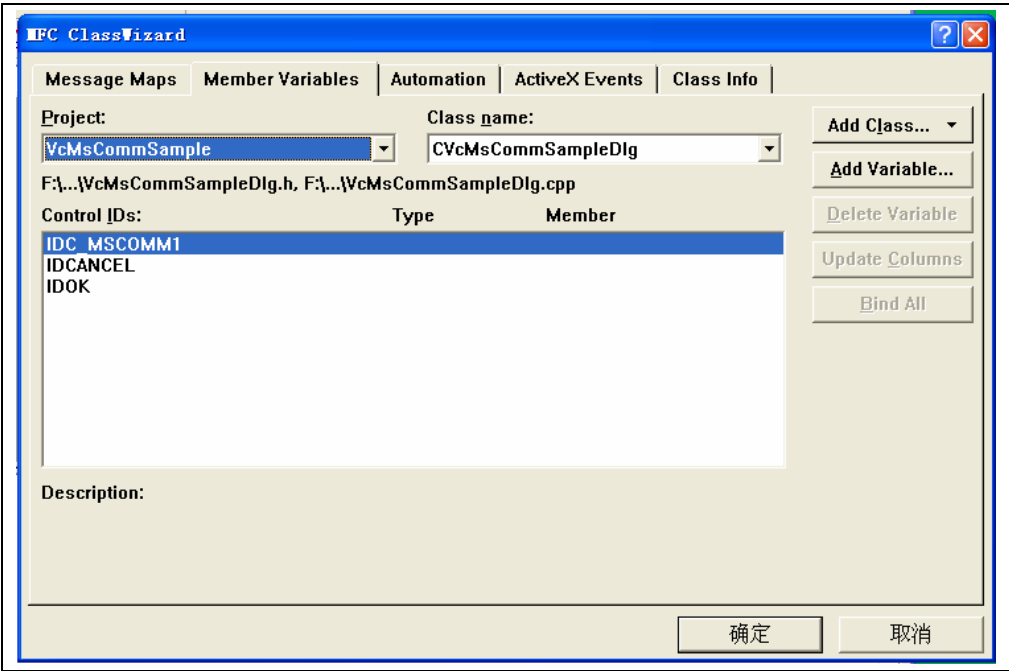


图 3 添加控件变量对话框

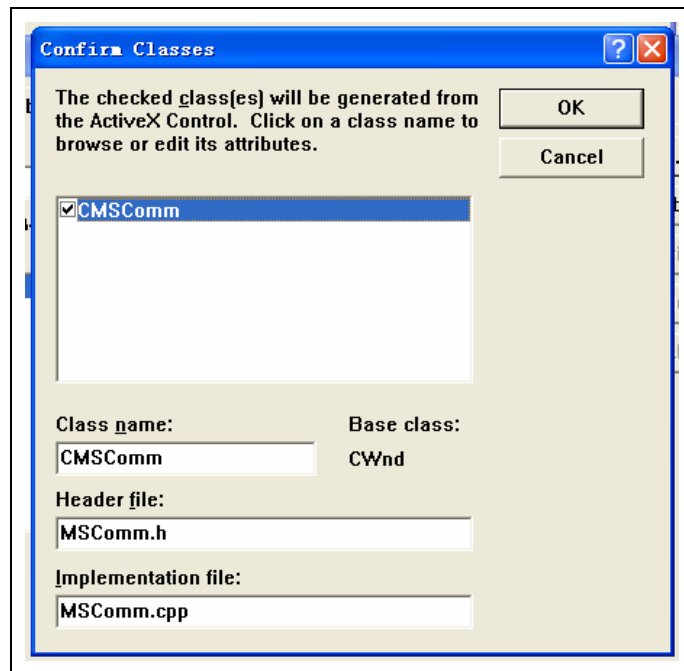


图 4 添加类对话框

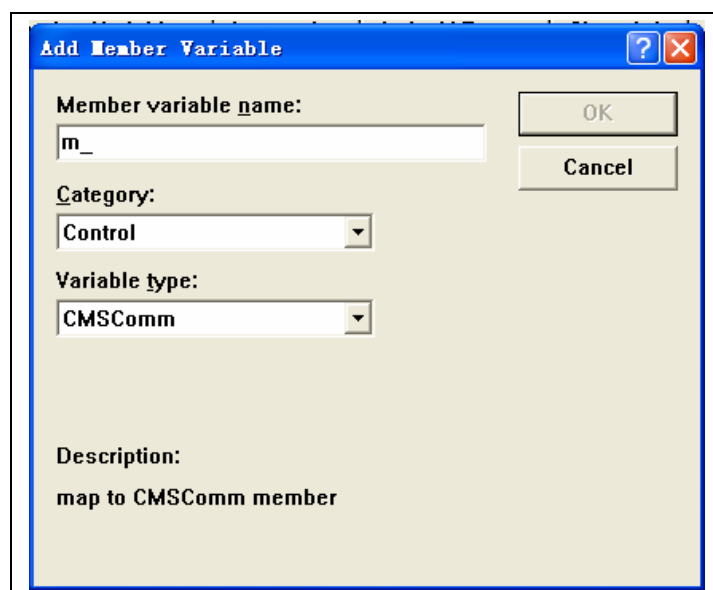


图 5 添加成员变量对话框

添加初始化代码

接下来,我们需要初始化这个串口,在类视图中,展开 CmscommSampleDlg,双击 OnInitDialog 函数(见图 6)。在“Return true;”加入以下代码:

```
m_comm.put_CommPort(1);           // 设置通信端口号
if ( !m_comm.get_PortOpen() )      // 打开串口
    m_comm.put_PortOpen(TRUE);
```

```
m_comm.put_Settings("9600,n,8,1");           // 设置串口配置
```

添加一些表要的界面元素。如图 7 所示。开关操作主要演示变量读写;寄存器读写主要演示读写单个寄存器;浮点数\整形读写操作主要演示多个寄存器读写,并转为不同的数据类型。优化通信是将所有的读操作合并处理,减轻串口通信压力。如不需要可不用关注,使用未优化通信即可。至此,我的准备工作基本完成,剩下的只需要编写用来发送和接收处理的代码。

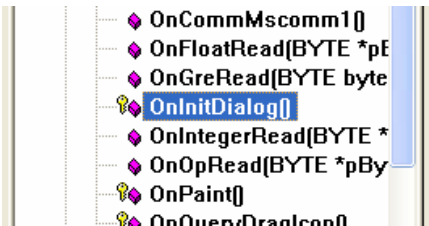


图 6 类视图



图 7 MscommSample 对话框

读写及返回处理

要通信,首先要知道通信的内容是什么?然后才是怎么通信的问题。通信的数据就是 Mod- bus 协议内容。更直白点就是对串口设备的读写请求。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

到这里,我们的需要发送的命令和接收数据都清楚了。如果你对 Modbus 协议还不是很清楚,可以参考 Modbus 协议介绍。

现在需要解决的问题是怎么通信 (即读写)。在 Mscomm 概述中我们讲过发送请求数据一般步骤是:

```
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)。
```

对于我们编写人机界面软件而言,需要不断读取 PLC 设备中的数值,来获取最新的数据。我们可以采用定时器。当程序启动或在用户点击开启按钮后,我们打开读取定时器,程序每过一段时间便调用我们编写好的读取函数。在用户点下“写入”,我们将用户的最新输入值发送到 PLC 设备中。

然而,还有一些问题需要引起注意。由于串口通信速度较慢,当我们发送一个请求后,需要等到返回或超时后才能发送下一个请求。而定时器并不关心这些,事件一到,便调用读取函数。这样就极有可能造成通信错误。为解决此问题,可以有俩种方法。一是设置一个全局变量(通信锁),需要通信时,首先检查变量锁是否有其它请求正在通信,若无,则锁定该变量并通信。等待接收数据或接收超时后解锁;否则放弃通信。这样可能造成有时不能写入的问题。这种方法在 VB 和 Delphi 中使用。另一种方法是维护一个请求队列,当需要读写请求时,将其添加到请求队列中。VC 将使用这一方法。

请求队列的数据单元定义为一个结构体,并在其中增加几个常用 Modbus 协议打包方法。ID 表示操作标志符号,当接收数据后使用;nStation 为设备站地址;nAddr 为操作起始地址;nLen 表示操作的数据长度;pSend 表示发送的数据。每次请求时调用这些方法完成打包,并添加到队列末尾。当超时定时器发生、队列为空定时器发生或接收到数据时,再发送下一个请求。

```
typedef struct _tagSendTask  
{  
    UINT    nID;           // 任务 ID 号  
    int     nSendLen;      // 发送数据的长度  
    int     nRevLen;       // 接收数据的长度  
    char    szSend[270];   // 发送的数据  
    char    szRev[270];    // 接收的数据  
}  
  
void InitReadBit( UINT ID, BYTE nStation, UINT nAddr, UINT nLen )  
void InitReadRegister( UINT ID, BYTE nStation, UINT nAddr, UINT nLen )  
void InitWriteBit( UINT ID, BYTE nStation, UINT nAddr, BYTE* pSend,UINT nLen )  
void InitWriteRegister(UINT ID, BYTE nStation, UINT nAddr, BYTE *pSend,UINT  
nLen )
```


1. 打包。在OnTime()函数中的 if(nIDEvent == 2) 中添加以下代码。

```
pSend = new SendTask;                //构造新的请求
pSend->InitReadRegister( REGISTERREAD,STATION,2337,1); //初始化读请求
m_SendList.push_back(pSend);          //添加到请求队列末尾
```

2. 发送。在SendPackage函数中。

```
M_comm.put_InputMode(1);                //设置获取数据的方式
m_comm.put_InputLen(0);                  //设置获取全部缓冲区数据
m_comm.put_InBufferCount(0);             //清空缓冲区
m_comm.put_RThreshold(m_pSend->nRevLen); //设置触发事件的接收字符数
CByteArray cmd_arr;
COleVariant _var;
cmd_arr.SetSize(m_pSend->nSendLen);
memcpy(&cmd_arr[0], m_pSend->szSend,m_pSend->nSendLen);
_var = cmd_arr;
m_comm.put_Output(_var);                 //发送数据
SetTimer(4,500,NULL);                   //设置超时时间
```

3. 接收。在界面中,双击mscomm组件图标,将事件处理方法添加到程序中。并添加以下代码。

```
KillTimer( 4 );                        //关闭超时定时器
int nBufCnt = m_comm.get_InBufferCount();//获取返回数据长度
COleVariant var = m_comm.get_Input(); //获取数据
if ( m_comm.get_CommEvent() == 2 && m_pSend->nRevLen ==
m_comm.get_RThreshold() )
{
    // 做返回处理
}
SendPackage();                         //发送下一个请求
```

4. 返回处理。先将ColeVariant类型的变量转成byte型变量。然后根据我发送时设定的ID来区分各个请求。也可以根据返回数据的命令号,起始地址和长度来区分。然后再调用OnRegisterRead函数,刷新界面。

```
GetSafeArray((BYTE*)m_pSend->szRev,var); //转换数据类型
switch( m_pSend->nID )
```

```

{
case ON_OFF_GRE_READ:
    break;
case ON_OFF_RED_READ:
    break;
case REGISTERREAD:
    OnRegisterRead( (BYTE*)&m_pSend->szRev[3]);    //刷新界面
    break;
case INTEGERREAD:
    break;
case FLOATREAD:
    break;
case OPREAD:
    break;
case ON_OFF_RED_WRITE:
case ON_OFF_GRE_WRITE:
case REGISTERWRITE:
case INTEGERWRITE:
case FLOATWRITE:
    m_ListCtl.InsertString(0,"写入数据成功");
    break;
default:
    ASSERT(FALSE);
    break;
}

```

这样程序中就可以显示PLC设备中的寄存器值了。另外在ButtonClick事件处理函数添加以下代码,便可以把写请求加入到请求队列中。

```

UpdateData();                                //交换界面和后台数据
CString strInfo;
WORD  wTemp;
wTemp = atoi(m_strRegEdit);                  //将输入值转为WORD数据
ChangeVar((BYTE*)&wTemp,2);                  //交换字节顺序
PSendTask pT = new SendTask;                 //构造请求
pT->InitWriteRegister(REGISTERWRITE,1,2337,(BYTE*)&wTemp,1);    //初始化写
请求
m_SendList.push_back(pT);                    //添加到请求队列末尾
UpdateData(FALSE);                           //将计算后的数据显示到界面

```

这样我们的寄存器读写就可以正常使用,使用同样的方法可以。采用相同的步骤便可以位、浮点数、整形等读写。