ads练习

练习1

step1：

- 新建工程
- 添加引用

step2：

添加控件

创建Adsclient

```
public TcAdsClient _client = null;
....
_client = new TcAdsClient();
```

连接plc

```
_client.Connect(851);
```

创建句柄

```
public int _handBOOl = 0;
public int _handByte = 0;
public int _handInt = 0;
public int _handArray = 0;
public int _handStruct = 0;
```

```
_handBOOl = _client.CreateVariableHandle("MAIN.Data_Bool");
_handByte = _client.CreateVariableHandle("MAIN.Data_Byte");
_handInt = _client.CreateVariableHandle("MAIN.Data_Int");
_handArray = _client.CreateVariableHandle("MAIN.Data_Array");
_handStruct = _client.CreateVariableHandle("MAIN.Data_Struct");
```

> 如果读不到，会抛出异常，应该加上try...catch

释放句柄

```
_client.DeleteVariableHandle(_handBOOl);
_client.DeleteVariableHandle(_handByte);
_client.DeleteVariableHandle(_handInt);
_client.DeleteVariableHandle(_handArray);
_client.DeleteVariableHandle(_handStruct);
```

step3 开始读写

使用read/write

```
AdsStream stream = new AdsStream(100);
AdsBinaryReader reader = new AdsBinaryReader(stream);
_client.Read(_handBOOl, stream);
textBox1.Text = reader.ReadBoolean().ToString();
_client.Read(_handByte, stream);
stream.Seek(0,System.IO.SeekOrigin.Begin);
textBox2.Text = reader.ReadByte().ToString();
_client.Read(_handInt, stream);
stream.Seek(0, System.IO.SeekOrigin.Begin);
textBox3.Text = reader.ReadInt16().ToString();
_client.Read(_handArray, stream);
stream.Seek(0, System.IO.SeekOrigin.Begin);
textBox4.Text = reader.ReadInt16().ToString();
textBox5.Text = reader.ReadInt16().ToString();
textBox6.Text = reader.ReadInt16().ToString();
textBox7.Text = reader.ReadInt16().ToString();
_client.Read(_handStruct, stream);
stream.Seek(0, System.IO.SeekOrigin.Begin);
textBox8.Text = reader.ReadInt16().ToString();
textBox9.Text = reader.ReadByte().ToString();
reader.ReadByte();
textBox10.Text = reader.ReadSingle().ToString();
```

写入

```
AdsStream stream = new AdsStream(100);
AdsBinaryWriter writer = new AdsBinaryWriter(stream);
writer.Write((short)int.Parse(textBox14.Text));
_client.Write(_handArray, stream,0,8);
```

```
            stream.Seek(0, System.IO.SeekOrigin.Begin);
            writer.Write(Int16.Parse(textBox18.Text));
            writer.Write(Byte.Parse(textBox19.Text));
            writer.Write((Byte)0);
            writer.Write(float.Parse(textBox20.Text));
            _client.Write(_handStruct, stream,0,8);
```

使用readany/writeany

读取

```
textBox1.Text = _client.ReadAny(_handBOOl, typeof(Boolean)).ToString();
short[] arry = (short[])_client.ReadAny(_handArray, typeof(short[]),new int[]{4});
textBox4.Text = arry[0].ToString();
```

读取结构体 先声明结构体

```
[StructLayout(LayoutKind.Sequential, Pack = 8)]
public struct TestStruct
{
    [MarshalAs(UnmanagedType.I2)]
    public short boolVal;
    [MarshalAs(UnmanagedType.I1)]
    public byte byVal;
    [MarshalAs(UnmanagedType.R4)]
    public float fVal;
}
```

然后读取

```
TestStruct tstruct = (TestStruct)_client.ReadAny(_handStruct, typeof(TestStruct));
```

使用tryread/trywrite

```
            int readBytes = 0;
            AdsStream stream = new AdsStream(100);
            AdsBinaryReader reader = new AdsBinaryReader(stream);
            AdsErrorCode code = _client.TryRead((uint)AdsReservedIndexGroups.SymbolValueByHandle,
(uint)0, stream,out readBytes);
            textBox1.Text = reader.ReadBoolean().ToString();
            stream.Seek(0, System.IO.SeekOrigin.Begin);
            _client.TryRead((uint)AdsReservedIndexGroups.SymbolValueByHandle, (uint)_handByte,
stream,out readBytes);
            textBox2.Text = reader.ReadByte().ToString();
```

```
            stream.Seek(0, System.IO.SeekOrigin.Begin);
            _client.TryRead((uint)AdsReservedIndexGroups.SymbolValueByHandle, (uint)_handStruct,
stream,out readBytes);
            textBox8.Text = reader.ReadInt16().ToString();
            textBox9.Text = reader.ReadByte().ToString();
            reader.ReadByte();
            textBox10.Text = reader.ReadSingle().ToString();
            stream.Seek(0, System.IO.SeekOrigin.Begin);
            _client.TryRead((uint)AdsReservedIndexGroups.SymbolValueByHandle, (uint)_handInt,
stream,out readBytes);
            textBox3.Text = reader.ReadInt16().ToString();
            _client.TryRead((uint)AdsReservedIndexGroups.SymbolValueByHandle, (uint)_handArray,
stream,out readBytes);
            stream.Seek(0, System.IO.SeekOrigin.Begin);
            textBox4.Text = reader.ReadInt16().ToString();
            textBox5.Text = reader.ReadInt16().ToString();
            textBox6.Text = reader.ReadInt16().ToString();
            textBox7.Text = reader.ReadInt16().ToString();
```

tryread和trywrite和之前函数最大的不同是，如果读不出来，会返回errorcode，而不是抛出异常。

异步读取

首先，初始化的时候，加入回调函数， 根据回调函数类型的不同，有两种

```
_client.AdsNotification += new AdsNotificationEventHandler(adsClient_AdsNotification);
```

```
_client.AdsNotificationEx += new AdsNotificationExEventHandler(adsClient_AdsNotificationEx);
```

> 经过我测试，如果有第二种回调函数，第一种不生效

启动回调

```
 _notificationHandleInt = _client.AddDeviceNotificationEx("MAIN.Data_Int", AdsTransMode.Cyclic,
1000, 0, this, typeof(short));
```

停止回调

```
if (_notificationHandleInt != 0)
    _client.DeleteDeviceNotification(_notificationHandleInt);
```

回调函数的处理

```
private void adsClient_AdsNotificationEx(object sender, AdsNotificationExEventArgs e)
{
    Debug.WriteLine(e.Value.ToString());
}
```

第二种

```
void adsClient_AdsNotification(object sender, AdsNotificationEventArgs e)
{
    AdsBinaryReader reader = new AdsBinaryReader(e.DataStream);
    Debug.WriteLine("adsClient_AdsNotification " + reader.ReadInt16().ToString());
}
```

考虑到不可以跨线程操作UI，可以用以下的方式操作

```
            this.Invoke(new Action(() =>
            {
                textBox1.Text = reader.ReadInt16().ToString();
            }));
```

练习2

测试adsserver 目标，通过高级语言实现adsserver

原理 在本地twincat上注册一个端口，用于接收ads通讯的数

好处 不需要轮训去获取变量的，可以使用plc主动触发的方式

step1

重新建立一个控制台程序,并添加ads引用

step2

新建一个类，继承自TcAdsServer

```
public class AdsService: TcAdsServer
{
    public AdsService(ushort port, string portName)
    : base(port, portName)
    {

    }
}
```

> 注意：里面的构造函数是必须，原因是adsserver没有无参数的构造函数

然后重写两个函数 分别是读取adsread命令的处理和adswrite命令的处理

```
//当客户端执行写入函数时执行。
public override void AdsWriteInd(AmsAddress rAddr,
    uint invokeId,
    uint indexGroup,
    uint indexOffset,
```

```
    uint cbLength,
    byte[] data)
{
    Debug.WriteLine("AdsWriteInd");
}
//当客户端执行读取函数时执行。
public override void AdsReadInd(AmsAddress rAddr,
uint invokeId,
uint indexGroup,
uint indexOffset,
uint cbLength)
{
    Debug.WriteLine("AdsReadInd");
}
```

然后我们在控制台把这个类声明一下，然后传入一个端口

```
static void Main(string[] args)
{
    AdsService _ads = new AdsService(12345, "test");
    _ads.Connect();
    Console.ReadLine();
}
```

启动，然后测试一下

> readline的函数的作用演示

通过ads自带的测试工具，发现有超时，原因很简单，我们在接收到消息的时候，并没有写返回，加入返回。

```
AdsWriteRes(rAddr, invokeId, 0);


AdsReadRes(rAddr, invokeId, 0, 5, data2);
```

这就完成了一个简易的adsserver，可以在函数里面对ads命令进行相应的处理

练习3

测试ads调用函数 首先在plc里加入一个功能块FB_Math 加入一个函数M_Add

```
{attribute 'TcRpcEnable'}
METHOD M_Add
VAR_INPUT
    enable:BOOL;
    a:DINT;
    b:DINT;
END_VAR
VAR_OUTPUT
    result:DINT;
    e2:BOOL;
```

```
    count:int;
END_VAR



IF enable THEN
    result:=a+b;
    gMathCallCount:=gMathCallCount+1;
ELSE
    result :=0;
END_IF
e2:=TRUE;

count:= gMathCallCount;
```

这么写的主要目的是为了看下input和output的对齐方式。

在原来的程序基础上加个按钮 首先，我们假定是按照8字节对齐，

```
            AdsStream readStream = new AdsStream(12);
            AdsStream writeStream = new AdsStream(9);
            AdsBinaryReader binReader = new AdsBinaryReader(readStream);
            AdsBinaryWriter binWriter = new AdsBinaryWriter(writeStream);
            binWriter.Write(true);
            binWriter.Seek(4, System.IO.SeekOrigin.Begin);
            binWriter.Write((int)110);
            binWriter.Write((int)1);
            _client.ReadWrite((int)AdsReservedIndexGroups.SymbolValueByHandle, _hMathAdd,
readStream, writeStream);
```

经过测试，发现是不对的，然后我们按照不对齐的方式来试下

```
            AdsStream readStream = new AdsStream(7);
            AdsStream writeStream = new AdsStream(9);
            AdsBinaryReader binReader = new AdsBinaryReader(readStream);
            AdsBinaryWriter binWriter = new AdsBinaryWriter(writeStream);
            binWriter.Write(true);
           // binWriter.Seek(4, System.IO.SeekOrigin.Begin);
            binWriter.Write((int)110);
            binWriter.Write((int)1);
            _client.ReadWrite((int)AdsReservedIndexGroups.SymbolValueByHandle, _hMathAdd,
readStream, writeStream);
```

发现这样是可以的，因此大概可以总结出来规律，函数调用时的input和output是不存在对齐的。

练习4

测试ads调用复杂结构体

对于一些超复杂的结构体，使用普通的datastream的方式是很难读取的，比较方便的方式是申请一个同等大小的结构体，然后用readany去读取

```
TYPE stAxis :
STRUCT
    DriverIO                    AT%I*:UINT; //与驱动器Digital inputs关联
    DriverErrorID               AT%I*:UINT; //与驱动器Error code关联
//  ModelOperation              AT%Q*:SINT; //操作模式与驱动器Model of operation关联
    Q_Brake                     AT%Q*:BOOL; //刹车
    AxisCamDog1                 AT%Q*:BOOL; //凸轮轴原点接近1
    AxisCamDog2                 AT%Q*:BOOL; //凸轮轴原点接近2
    LimitP                          :BOOL; //正极限
    LimitN                          :BOOL; //负极限
    HOME                            :BOOL; //原点
    HomingDirection                 :BOOL; //回原点方向
    EXT1                            :BOOL; //外部锁存
    ToDriverIO                      :DWORD;//驱动器IO数据位

    ServoOn_HMI                     :BOOL; //伺服ON按钮
    ServoOnInt                      :BOOL; //初始化伺服ON
    Homing_HMI                      :BOOL; //回原点启动
    HomingInt                       :BOOL; //初始化回原点
    BrakeBtn_HMI                    :BOOL; //刹车
    Inch_HMI                        :BOOL; //寸动
    JogF_HMI                        :BOOL; //正向微动
    JogB_HMI                        :BOOL; //反向微动
    StationIni                      :BOOL; //轴所在工位初始化中
    AutoPStart_HMI                  :BOOL; //轴自动停位启动
    StopCond                        :BOOL; //轴停止条件

    ServoOn_L                       :BOOL; //伺服ON_L
    OriginOk_L                      :BOOL; //原点OK标志
    SingleCycle                     :BOOL; //轴所在工位单循环
    Busy                            :BOOL; //轴忙
    ActPos                          :LREAL;//轴当前实际位置
    AutoPosition                    :LREAL;//轴自动停位
    Alarm                           :DWORD;  //报警:00伺服报警 01负极限异常 02正极限异常 03原点接近
异常 04轴未找到原点 05轴不在自停位 06伺服驱动器异常 08轴未伺服On
    ErrorID                         :UDINT;


    ActionEnable                    :BOOL; //轴可动作
    ActionEnableM                   :BOOL; //轴可动作M
    PosEnable                       :BOOL; //轴手动基本条件
    PosEnableM                      :BOOL; //轴手动基本条件M
    BackAutoPosEN                   :BOOL; //轴可自动回自停位
    Homing                          :BOOL; //轴回原点
    MotionMethod                    :BOOL; //运动方式 0相对1绝对
    ServoOn_M                       :BOOL; //伺服ON
    Inch_M                          :BOOL; //寸动
    JogF_M                          :BOOL; //正向微动
    JogB_M                          :BOOL; //反向微动
    AutoPStart_M                    :BOOL; //轴自动停位启动
    BrakeBtn_M                      :BOOL; //刹车

    GrabSafetyPos                   :BOOL;  //抓取安全位
    PlacedSafetyPos                 :BOOL;  //放料安全位

    PosHandStart                    :ARRAY[1..10] OF BOOL; //轴手动启动
    PosHandStartM                   :ARRAY[1..10] OF BOOL; //轴手动启动
```

```
        PosAutoStart                    :ARRAY[1..10] OF BOOL; //轴自动启动
        PosStart_M                      :ARRAY[1..10] OF BOOL; //轴启动
        Position_L                      :ARRAY[1..10] OF BOOL; //轴位指示

        PositionDone                    :BOOL; //轴动作完成

        PosHandEnable                   :ARRAY[1..10] OF BOOL; //轴手动条件

        VelocityTotal                   :LREAL;//速度汇总
    //
        originCompensation              :LREAL;//原点补偿

        InchLength                       :LREAL;//寸动长度
        Override                         :LREAL;//速度倍率
        JogVelocity                      :LREAL;//微动速度
        VelocityManual                   :LREAL;//轴手动速度
        VelocityAuto                     :LREAL;//轴自动速度
        VelocityAutoM                    :LREAL;//轴自动速度M
        VelocityDebug                    :LREAL;//调试速度
        HomeVelocity                     :LREAL;//原点补偿与寸动速度
        ErrorRangeSet                    :LREAL;//轴误差范围设定
        AlarmCount                       :DINT; //报警计数

        PositionSet                      :ARRAY[1..10] OF LREAL;//轴位置设置

        DTerminalVelocity                :LREAL;//对端子速度
        TerminalvelocityManual           :LREAL;//对端子手动速度
        TerminalvelocityAuto             :LREAL;//对端子自动速度
    END_STRUCT
    END_TYPE
```

在GVL里面，我们把这个结构体声明成数组

```
        stAxis              :ARRAY[1..10] OF stAxis;          //伺服轴
```

首先，高级语言定义结构体

```
        [StructLayout(LayoutKind.Sequential, Pack = 8)]
        public struct Plc_StAxis
        {
            [MarshalAs(UnmanagedType.U2)]
            public UInt16 DriverIO;
            [MarshalAs(UnmanagedType.U2)]
            public UInt16 DriverErrorID;

            [MarshalAs(UnmanagedType.I1)]
            public bool Q_Brake;
            [MarshalAs(UnmanagedType.I1)]
            public bool AxisCamDog1;
            [MarshalAs(UnmanagedType.I1)]
            public bool AxisCamDog2;
            [MarshalAs(UnmanagedType.I1)]
            public bool LimitP;
            [MarshalAs(UnmanagedType.I1)]
            public bool LimitN;
            [MarshalAs(UnmanagedType.I1)]
```

```
        public bool HOME;
        [MarshalAs(UnmanagedType.I1)]
        public bool HomingDirection;
        [MarshalAs(UnmanagedType.I1)]
        public bool EXT1;
        [MarshalAs(UnmanagedType.I1)]
        public bool ToDriverIO;

        [MarshalAs(UnmanagedType.I1)]
        public bool ServoOn_HMI;
        [MarshalAs(UnmanagedType.I1)]
        public bool ServoOnInt;
        [MarshalAs(UnmanagedType.I1)]
        public bool Homing_HMI;
        [MarshalAs(UnmanagedType.I1)]
        public bool HomingInt;
        [MarshalAs(UnmanagedType.I1)]
        public bool BrakeBtn_HMI;
        [MarshalAs(UnmanagedType.I1)]
        public bool Inch_HMI;
        [MarshalAs(UnmanagedType.I1)]
        public bool JogF_HMI;
        [MarshalAs(UnmanagedType.I1)]
        public bool JogB_HMI;
        [MarshalAs(UnmanagedType.I1)]
        public bool StationIni;
        [MarshalAs(UnmanagedType.I1)]
        public bool AutoPStart_HMI;
        [MarshalAs(UnmanagedType.I1)]
        public bool StopCond;

        [MarshalAs(UnmanagedType.I1)]
        public bool ServoOn_L;
        [MarshalAs(UnmanagedType.I1)]
        public bool OriginOk_L;
        [MarshalAs(UnmanagedType.I1)]
        public bool SingleCycle;
        [MarshalAs(UnmanagedType.I1)]
        public bool Busy;
        [MarshalAs(UnmanagedType.R8)]
        public double ActPos;
        [MarshalAs(UnmanagedType.R8)]
        public double AutoPosition;

        [MarshalAs(UnmanagedType.U4)]
        public Int32 Alarm1;
        [MarshalAs(UnmanagedType.U4)]
        public Int32 ErrorID;

        [MarshalAs(UnmanagedType.I1)]
        public bool ActionEnable;   //轴可动作
        [MarshalAs(UnmanagedType.I1)]
        public bool ActionM;   //轴手动基本条件
        [MarshalAs(UnmanagedType.I1)]
        public bool PosEnable;
        [MarshalAs(UnmanagedType.I1)]
        public bool PosEnableM;
        [MarshalAs(UnmanagedType.I1)]
        public bool BackAutoPosEN;
        [MarshalAs(UnmanagedType.I1)]
        public bool Homing;   //轴回原点
```

```
[MarshalAs(UnmanagedType.I1)]
public bool MotionMethod;   //轴初始化回原点
[MarshalAs(UnmanagedType.I1)]
public bool ServoOn_M;   //伺服ON
[MarshalAs(UnmanagedType.I1)]
public bool Inch_M;   //寸动
[MarshalAs(UnmanagedType.I1)]
public bool JogF_M;   //正向微动
[MarshalAs(UnmanagedType.I1)]
public bool JogB_M;   //反向微动
[MarshalAs(UnmanagedType.I1)]
public bool AutoPStart_M;   //轴自动停位启动
[MarshalAs(UnmanagedType.I1)]
public bool BrakeBtn_M;   //刹车

[MarshalAs(UnmanagedType.I1)]
public bool GrabSafetyPos;
[MarshalAs(UnmanagedType.I1)]
public bool PlacedSafetyPos;

[MarshalAs(UnmanagedType.ByValArray, SizeConst = 10)]
public byte[] PosHandStart;   //轴手动启动
[MarshalAs(UnmanagedType.ByValArray, SizeConst = 10)]
public byte[] PosHandStartM;   //轴手动启动
[MarshalAs(UnmanagedType.ByValArray, SizeConst = 10)]
public byte[] PosAutoStart;   //轴自动启动
[MarshalAs(UnmanagedType.ByValArray, SizeConst = 10)]
public byte[] PosStart_M;   //轴启动
[MarshalAs(UnmanagedType.ByValArray, SizeConst = 10)]
public byte[] Position_L;   //轴位指示

[MarshalAs(UnmanagedType.I1)]
public bool PositionDone;   //轴动作完成

[MarshalAs(UnmanagedType.ByValArray, SizeConst = 10)]
public byte[] PosHandEnable;   //轴手动条件

[MarshalAs(UnmanagedType.R8)]
public double VelocityTotal;  //速度汇总
[MarshalAs(UnmanagedType.R8)]
public double OriginCompensation;  //原点补偿

[MarshalAs(UnmanagedType.R8)]
public double InchingLength;  //寸动长度
[MarshalAs(UnmanagedType.R8)]
public double Override;  //速度倍率
[MarshalAs(UnmanagedType.R8)]
public double JogVelocity;  //微动速度
[MarshalAs(UnmanagedType.R8)]
public double VelocityManual;  //轴手动速度
[MarshalAs(UnmanagedType.R8)]
public double VelocityAuto;  //轴自动速度
[MarshalAs(UnmanagedType.R8)]
public double VelocityAutoM;  //轴自动速度M
[MarshalAs(UnmanagedType.R8)]
public double VelocityDebug;  //调试速度

[MarshalAs(UnmanagedType.R8)]
public double HomingVelocity;  //原点补偿与寸动速度
[MarshalAs(UnmanagedType.R8)]
```

```
        public double ErrorRangeSet;  //轴误差范围设定
        [MarshalAs(UnmanagedType.U4)]
        public Int32 AlarmCount;   //报警计数

        [MarshalAs(UnmanagedType.ByValArray, SizeConst = 10)]
        public double[] PositionSet;

        [MarshalAs(UnmanagedType.R8)]
        public double DTerminalVelocity;  //对端子速度
        [MarshalAs(UnmanagedType.R8)]
        public double TerminalvelocityManual;  //对端子手动速度
        [MarshalAs(UnmanagedType.R8)]
        public double TerminalvelocityAuto;    //对端子自动速度
    }

    [StructLayout(LayoutKind.Sequential, Pack = 8)]
    public struct Plc_StAxisInfo
    {
        [MarshalAs(UnmanagedType.ByValArray, SizeConst = 9)]
        public Plc_StAxis[] Axis;
    }
```

以readany的方式读出来