Click Here

百度知道>提示信息 知道宝贝找不到问题了>_提示信息 知道宝贝找不到问题了>_ byte //word => ushort //DWord => uint //Int => short //Dint => int //Real => float //LReal => double //String => string //DateTimeLong=>dateTime //s7wstring=>string 数据类涉及到最后的转换。 需要留意 以上一篇文章博途数据块为例： 点击转至在线》点击第二个点。 监视数据块中的值的签化 图中偏移量为所在数据块的隔的偏移量为占用的字节 以第一个bool值举例： var a = plc.Read("DB1.DBX0.0");//会重复发送读cp请求，效率较低，非特殊情况不建议便用 plc.Write("DB1.DBX0.0", true);//与方法的参数为，数据库类型默认是数据库：DataBlock。 第几个数据，偏移量，数据类型，读取多少位 可以看到a值读取到设置的初始值false 在第2步中修改了bool值之后，博途软件中监视值换成了true 以上代码读取一次会重新建立一个tcp连接，非常消耗资源。 在实际开发中不建议使用 可以用以下方案： int db = 1; var boolDemo = plc.Read(DataTypeDataBlock, db, 0, VarType.Bit, 1); Console.WriteLine("bool值[打印：" + boolDemo); plc.Write(DataTypeDataBlock, db, 0, !boolDemoRead); 在读中第一个参数，标识数据块类型。 第二个标识数据块。 我们是db1，所以写：1 偏移量为0 类型为bit 获取长度（除了字符类型和数组组类其他值类型的示例：以下代码是其他值类型的示例：PlcplcPlc.CpuType.S71500, "192.168.43.14", 0, 1); plc.Open(); if (plc.IsConnected) { Console.WriteLine("PLC连接成功"); } var a = plc.Read("DB1.DBX0.0", true); var db = 1; var boolDemo = plc.Read(DataTypeDataBlock, db, 0, VarType.Bit, 1); Console.WriteLine("bool值[打印：" + boolDemo); plc.Write(DataTypeDataBlock, db, 0, false); var boolDemoRead = plc.Read(DataTypeDataBlock, db, 0, VarType.Bit, 1); Console.WriteLine("修改后bool值[打印：" + boolDemoRead); var wordDemo = plc.Read(DataTypeDataBlock, db, 2, VarType.Int, 1); Console.WriteLine("word值[打印：" + wordDemo); plc.Write(DataTypeDataBlock, db, 2, (ushort)9); var wordDemoRead = plc.Read(DataTypeDataBlock, db, 2, VarType.Int, 1); Console.WriteLine("修改后word值[打印：" + wordDemoRead); var DwordDemo = plc.Read(DataTypeDataBlock, db, 4, VarType.Int, 1); Console.WriteLine("Dword值[打印：" + DwordDemo); plc.Write(DataTypeDataBlock, db, 4, 8); var DwordDemoRead = plc.Read(DataTypeDataBlock, db, 4, VarType.Int, 1); Console.WriteLine("修改后Dword值[打印：" + DwordDemoRead); var intDemo = plc.Read(DataTypeDataBlock, db, 8, VarType.Int, 1); Console.WriteLine("int值[打印：" + intDemo); short readInt = 9; plc.Write(DataTypeDataBlock, db, 8, readInt); var intDemoRead = plc.Read(DataTypeDataBlock, db, startByteAdr: 8, VarType.Int, 1); Console.WriteLine("修改后int值[打印：" + intDemoRead); var DintDemo = plc.Read(DataTypeDataBlock, db, 10, VarType.Int, 1); Console.WriteLine("Dint值[打印：" + DintDemo); plc.Write(DataTypeDataBlock, db, 10, 100); var DintDemoRead = plc.Read(DataTypeDataBlock, db, 10, VarType.Int, 1); Console.WriteLine("修改后Dint值[打印：" + DintDemoRead); var RealDemo = plc.Read(DataTypeDataBlock, db, 14, VarType.Int, 1); Console.WriteLine("Real值[打印：" + RealDemo); plc.Write(DataTypeDataBlock, db, 14, (float)99.9); var DrealDemoRead = plc.Read(DataTypeDataBlock, db, 14, VarType.Int, 1); Console.WriteLine("修改后Real值[打印：" + DrealDemoRead); var LRealDemo = plc.Read(DataTypeDataBlock, db, 18, VarType.Int, 1); Console.WriteLine("LReal值[打印：" + LrealDemo); plc.Write(DataTypeDataBlock, db, 18, 88.88); var LrealDemoRead = plc.Read(DataTypeDataBlock, db, 18, VarType.Int, 1); Console.WriteLine("修改后Lreal值[打印：" + LrealDemoRead); var bytDemo = plc.Read(DataTypeDataBlock, db, 290, VarType.Byte, 1); Console.WriteLine("byte值[打印：" + bytDemo); plc.Write(DataTypeDataBlock, db, 290, (byte)2); var bytDemoRead = plc.Read(DataTypeDataBlock, db, 290, VarType.Byte, 1); Console.WriteLine("修改后byte值[打印：" + bytDemoRead); var dateDemo = plc.Read(DataTypeDataBlock, db, 282, VarType.DateTime, 1); Console.WriteLine("date值[打印：" + dateDemo); plc.Write(DataTypeDataBlock, 2, 282, SystemDateTimeNow); var dateDemoRead = plc.Read(DataTypeDataBlock, db, 282, VarType.String, 1); Console.WriteLine("修改后data值[打印：" + dateDemoRead); var charDemo = plc.Read(DataTypeDataBlock, db, 4, VarType.String, 1); Console.WriteLine("char值[打印：" + charDemo); plc.Write(DataTypeDataBlock, db, 4, "a"); var charDemoRead = plc.Read(DataTypeDataBlock, db, 4, VarType.Int, 1); Console.WriteLine("修改后char[打印：" + charDemoRead); 比较特殊的： string： 需要先获取string值的所占长度， 再拿到具体byte值。 转换为utf8格式的ascii码 具体代码中有体现 +1表示获取到长度 +2表示获取到跳过偏移长度的字符 string类型只能存储ascii码 ，需要注意 ，不能存储中文 #pragma warning disable var count = (byte)plc.Read(DataTypeDataBlock, db, 26 + 1, VarType.Byte, 1); byte[] stringDemo = plc.ReadBytes(DataTypeDataBlock, db, 26 + 2, count); string stringValue = Encoding.Default.GetString(stringDemo); Console.WriteLine("string值[打印：" + stringValue); plc.Write(DataTypeDataBlock, db, 26 + 1, (byte)"nihao".Length); plc.Write(DataTypeDataBlock, db, 26 + 2, "nihao"); string stringDemoRead = Encoding.Default.GetString(stringDemo); Console.WriteLine("修改后string值[打印：" + stringDemoRead); wstring： wstring的读取比较difficult ，需要注意的是获取的字符需要为254个。 因为符号占用了4个字节 var wstringDemo = plc.Read(DataTypeDataBlock, db, startByteAdr: 292, VarType.S7WString, varCount: 254); array： array区别于其他方法的原因是：在读取时需要进行转换，例如便用float[] 将博途的real类型转换为float数组类型。 在读取时count值也需要和plc中设置的长度一致 在写入时，可以在读取的值上做修改如： arraydemo。 也可以自定义一个新数组，注意新数组的长度可以少于plc中设定的长度。 那么长度只能的值会写入生效。 例如plc中设置长度为2，我写入的长度只有1，那么数组第一个值会写入，plc中第一个值会改变，第二个值则不变。 如果新数组的长度大于plc中设置的长度，则抛出异常 具体可以看代码实现 //读取array float[] arrayDemo = (float[])plc.Read(DataTypeDataBlock, db, 804, VarType.Real, 10);//array转为特殊，他设定的范围值可能有多种类型 , real , int , word , bool等 , 需要根据不同类型做处理 Console.WriteLine("array值[打印：" + System.String.Join(", ", arrayDemo)); //写入array并打印 for (int i = 0; i < arrayDemo.Length; i++) { arrayDemo[i] += 0.1F; } plc.Write(DataTypeDataBlock, db, 804, arrayDemo);//在原有的基础上修改 float[] arrayDemo2 = { 0.11F, 0.22F }.//如果只改变其中一个帮助方法修改传值的array plc.Write(DataTypeDataBlock, db, 804, arrayDemo2);//可以修改少数几个。 如果只修改其中几个则修改后灵整修改 指针不能超过长度 float[] arrayDemoRead = (float[])plc.Read(DataTypeDataBlock, db, 804, VarType.Real, 10); Console.WriteLine("修改后array值打印：" + System.String.Join(",", arrayDemoRead)); 实现了一个帮助类 , 导入即用：资源链接： 文章来源： ◆◆权声明： 本文为博主原创文章 , 遵循CC 4.0 BY-SA 知识共享协议 , 转载请附上原文出处链接和本声明。