C# uses DA and UA to access PLC through KepServer

1. Introduction to the overall structure of PLC access via KepServer

One: Kepserver is divided into client and server.

1. The server is responsible for interacting with the PLC (data reading and writing) through the driver of each manufacturer's equipment integrated by itself.

2. The client completes the collection of PLC data by interacting with the server for our use, and writes the data into the PLC through the server for control.

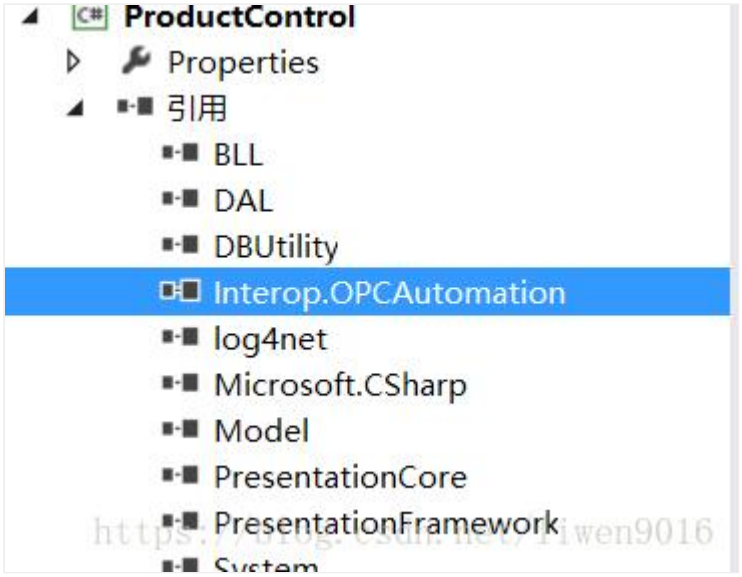
Two: Access PLC through DA mode, use C# development

1. If Client and Server are not on the same machine, then both machines need to be configured with DCOM authorization mode.

2. If you can't find the COM factory object during OPC connection, you can try regsvr32 OPCDAAuto.dll, generally in the system32 or sysWOW64 directory.

3. When we use C# to access the PLC through Kepserver, the actual principle is to call the dynamic library Interop.OPCAutomation.dll for us. This dynamic library can be understood as a client of KepServer. This client provides us with an interface to read and write PLC. , We call through the interface and interact with the KepServer server to implement PLC operation. The following steps begin to briefly introduce the specific operations.

4. After creating a new C# project, add a reference to the dynamic library at the reference place,



5. First instantiate an OPCServer object

```
OPCServer server = new OPCServer();
```

Then call the Connect() method of the server object. This method needs to provide two parameters. The source code of this method is declared as

```
1 [DispId(1610743826)]
2     [MethodImpl(MethodImplOptions.InternalCall, MethodCodeType = MethodCodeType.Runti
3     void Connect([MarshalAs(UnmanagedType.BStr), In] string ProgID, [MarshalAs(Unmana
```

The format of these two parameters is very fixed. For example, to connect to the V6 version, ProgID writes "KEPware.KEPServerEx.V6", Node is actually the IP address of the computer where the KepServer server is located. If the Client and Server are on the same machine, you can write 127.0.0.1

Speak far away, we start to connect

```
server.Connect(ProgID, Node);
```

Next we declare 4 variables

TOP

```
OPCGroups groups; 2 OPCGroup group;
OPCItems items;
OPCItem item;
```

OPCGroups is the biggest concept under "Server". It is a collection of OPCGroup. Below you can see the relationship between these four.

"Server" has the property OPCGroups directly, we can get it directly.

```
groups = server.OPCGroups; //Get group jih
                        groups.DefaultGroupIsActive = true; //Set the group collecti
                        groups.DefaultGroupDeadband = 0; //Set dead zone
                        groups.DefaultGroupUpdateRate = 200;//Set the update frequen
```

It is mentioned here that OPCGroups is a collection of groups, and you can get a group, OPCGroup, by calling its Add(string) method. You will find later that many of our operations are operated in units of OPCGroup.

```
group = server.OPCGroups.Add(tmpGroupName);
                        group.IsSubscribed = true; //Whether it is a subscription
                        group.UpdateRate = 200; //Refresh frequency
                        group.DataChange += mygroup.onDataChange; //Callback functic
                        group.AsyncReadComplete += mygroup.onAsyncReadComplete; //As
                        group.AsyncWriteComplete += mygroup.onAsyncWriteComplete; //
                        group.AsyncCancelComplete += mygroup.onAsyncCancelComplete;/
```

Pay attention to the parameters of the Add() method here, the declaration of this method is

```
 [DispId(1610743822)]

[MethodImpl(MethodImplOptions.InternalCall, MethodCodeType = MethodCodeType.Runtime
    [return: MarshalAs(UnmanagedType.Interface)] 4
OPCGroup Add([MarshalAs(UnmanagedType.Struct), In, Optional] object Name);
```

Doesn't it look scary, it's actually very simple, just the name of this group. Just like our name. Haha. The most important thing here is actually in the subscription mode, the DataChange event, any d

TOP

changes in this group will trigger this event, call our callback function to process the data.

```
//Group object event processing virtual function: data change
        public virtual void onDataChange(int TransactionID, int NumItems, ref Array C
        {
            if (DataChange != null)
                DataChange(this, TransactionID, NumItems, ref ClientHandles, ref Item
        }
```

In other words, when the PLC value changes, this event is triggered, and then we can get the value of the changed variable here for the processing of our program logic.

Next is items (closer and closer to us), it is a collection of items. What is an item? The item corresponds to the specific variable in our kepserver and corresponds to the address of our PLC.

标记名称	地址	数据类型	扫描速率	缩放	说明
Fault_RB01_Fault_ProgNo_Match	K0058	Boolean	100	无	1#机器人程序匹配错误
Fault_RB01_Fault_Electrode_GameOver	K0055	Boolean	100	无	1#机器人电极寿终故障
Fault_RB01_Fault_in_Process	K0056	Boolean	100	无	1#机器人过程故障
Fault_RB01_Fault_WeldMachine_No_Ready	K0063	Boolean	100	无	1#机器人焊机未准备好故障
Fault_RB01_Fault_Weld	K0060	Boolean	100	无	1#机器人焊接故障
Fault_RB01_Fault_No_Weld_with_Current	K0057	Boolean	100	无	1#机器人焊接未投入报警
Fault_RB01_Fault_WeldGun_Temp	K0061	Boolean	100	无	1#机器人焊枪温度报警故障
Fault_RB01_Fault_WeldGun_TipCh_PreAlert	K0062	Boolean	100	无	1#机器人换帽预警
Fault_RB01_Fault_Air	K0054	Boolean	100	无	1#机器人气压故障
Fault_RB01_Fault_Water	K0059	Boolean	100	无	1#机器人循环水故障
Fault_RB02_Fault_ProgNo_Match	K0068	Boolean	100	无	2#机器人程序匹配错误
Fault_RB02_Fault_Electrode_GameOver	K0065	Boolean	100	无	2#机器人电极寿终故障
Fault_RB02_Fault_in_Process	K0066	Boolean	100	无	2#机器人过程故障
Fault_RB02_Fault_WeldMachine_No_Ready	K0073	Boolean	100	无	2#机器人焊机未准备好故障
Fault_RB02_Fault_Weld	K0070	Boolean	100	无	2#机器人焊接故障
Fault_RB02_Fault_No_Weld_with_Current	K0067	Boolean	100	无	2#机器人焊接未投入报警
Fault_RB02_Fault_WeldGun_Temp	K0071	Boolean	100	无	2#机器人焊枪温度报警故障

that's it. Get items method is very simple, Groups directly contains this attribute

```
items = group.OPCItems;
```

After getting the items, call the items.Add(para1, para2) method to add the items to the items and return the item object. The source code of this method is

```
[DispId(1610743819)]
   [MethodImpl(MethodImplOptions.InternalCall, MethodCodeType = MethodCodeType.Runti
   [return: MarshalAs(UnmanagedType.Interface)]
   OPCItem AddItem([MarshalAs(UnmanagedType.BStr), In] string ItemID, [In] int Clien
```

Two parameters of Add() need special attention. Especially the first parameter: ItemID, this must be consistent with the item in kepserver, if it is inconsistent, this method will report an exception a

TOP

add failure. ItemID is a string, and its composition is the Channel name in Kepserver. + Device name. + Group name. + Tag name. such as:

The ItemID of the PLC address K0058 is CX62KX63.ER.Alarm.Fault_RB01_Fault_ProgNo_Match. It must be noted here that it is best to copy it, and do not write it wrong. The second parameter is the item handle in the group. This parameter is unique within the group and is used to represent this item.

You're done here. After the PLC data changes, the DataChannge callback function can be triggered, and we can get the changed number.

But sometimes, we need to actively read and write PLC. Synchronous reading and synchronous writing. There are two ways:

Method 1: Through item. Directly call iem.read(), item.write(), fill in the response parameters, you can achieve read and write.

Method two: through the item group.

```
1 //Synchronous reading
2  group.SyncRead((short)OPCDataSource.OPCDevice, NumItems, ServerHandles, out Values,
3  //Synchronous write
4  group.SyncWrite(NumItems, ServerHandles, Values, out Errors);
```

If you use asynchronous, then you can only pass the item group.

```
1 //Asynchronous reading
2
  group.AsyncRead(NumItems, ServerHandles, out Errors, TransactionID, out CancelID);
3  //Asynchronous write 4
group.AsyncWrite(NumItems, ServerHandles, Values, out Errors, TransactionID, out Cance
```

TOP

Our operation of PLC is nothing more than that. Automatic feedback of data changes, synchronous read, asynchronous read, synchronous write, asynchronous write. The operation on the data is written here.

Next we have to use up resources to remember to release. Freeing up resources is easy.

Remove group

```
OPCGroups groups;
            groups = server.OPCGroups;
            groups.Remove(group.GroupName);
```

Disconnect

```
      //Disconnect KEPServer
        public static void DisconnectServer()
        {
            server.Disconnect();
        }
```

Well, the steps for DA to operate KepServer are written here. I won't post the source code here, because my source code is in the project, and it can't be used directly when taken alone. It depends on many auxiliary classes and database operations. If you have any questions, you can add QQ group: 633204942 to discuss and consult together: the next chapter writes UA.

Intelligent Recommendation

OPC UA access

Explanation With the rapid development of WEB and mobile Internet, OPC UA is becoming more and more widely used. I need it recently, I have organized it myself, and I will share it with you. Start I u...