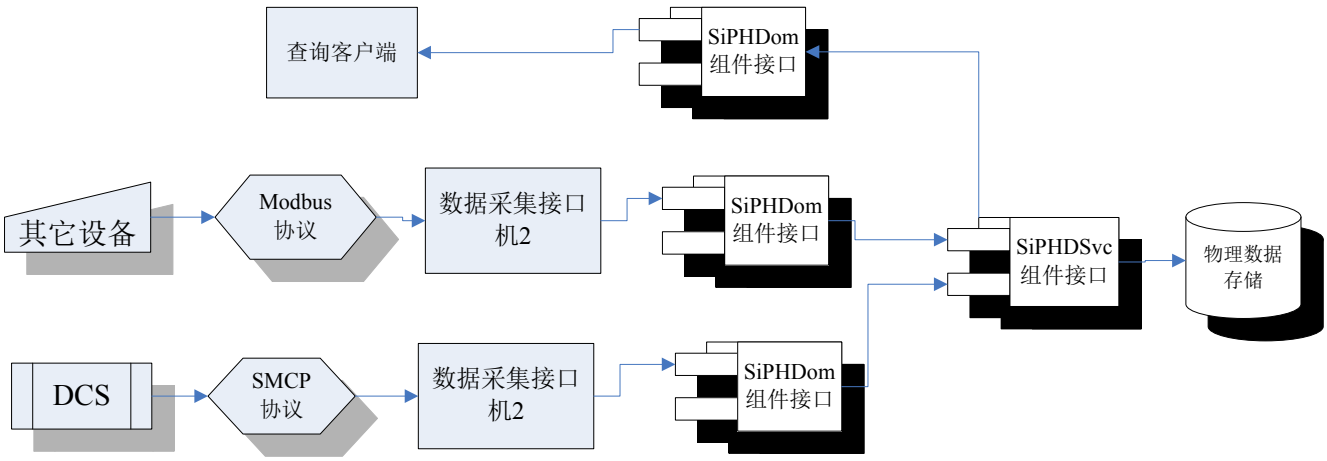
版本： 2.0

SIPHD 客户端开发流程(Java 版)

1、概述

SIPHD 实时历史数据库的服务器与客户端的通信机制如下图所示：



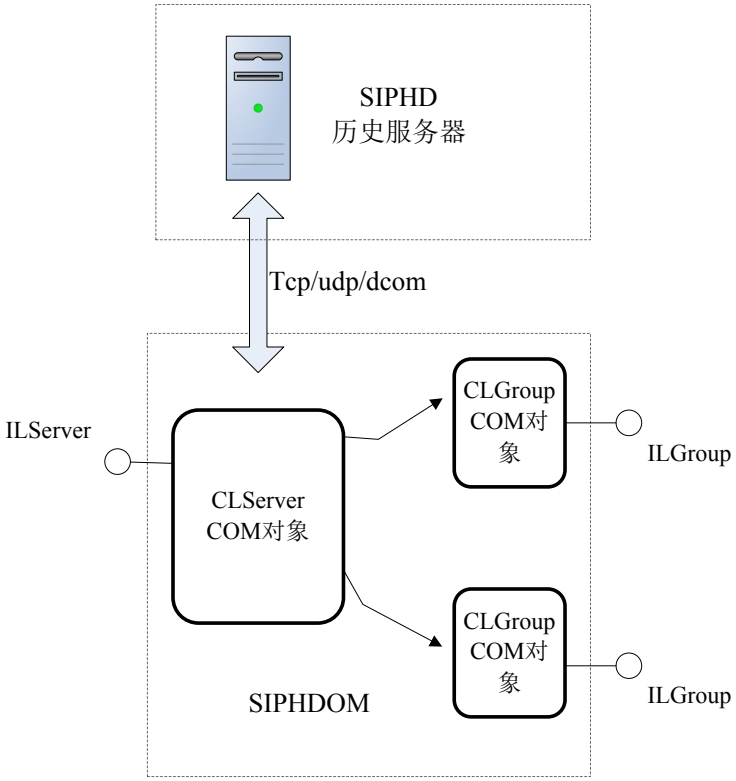
SIPHDOM 组件是 SIPHD 实时历史数据库提供的用于客户端通信的进程内 COM 组件，所有的客户端程序（如接口机，趋势，报表等）都通过该组件来间接访问历史服务器，该组件提供的主要功能包括：

- ✧ 标签点信息的查询
- ✧ 实时数据的存储、查询
- ✧ 历史数据的查询

SIPHDOM 组件内部结构示意图如下图所示，对外提供两种类型接口：

- ✧ ILServer 接口，历史服务器访问基本接口，提供与历史服务器的基本访问接口及服务器的连接、断开、标签点信息的查询等功能；
- ✧ ILGroup 接口，标签点组访问接口，提供对特定标签点集合的实时数据存储和查询等功能。

通过 ILServer 接口的标签点组管理功能，可以对标签点进行编组访问(例如对同一设备的标签点加入到同一个标签点组中)。



2.基本数据对象

2.1 标签信息数据格式

SIPHD历史数据库接口通讯上传递的标签信息对象为ILTagInfo，其结构为：

```
ILTagInfo
{
    bstr   Name;          ///名称
    bstr   Description;   ///描述
    long   ID;            ///ID
    long   Type;          ///标签类型,0模拟量;1开关量;2数字量;
    long   DataSource;    ///数据源ID
    long   Period;        ///采集周期;
    long   TimeOut;       ///超时时间
    long   Snap;          ///强制归档时间
    float  High;          ///量程高限
    float  Low;           ///量程低限
    float  err;           ///压缩算法静态死区
    float  diverr;        ///压缩算法斜率死区
    bstr   Unit;          ///单位
};
```

ILTagInfo为标签点信息查询的基本数据对象，对模拟量、开关量和数字量都用该结构表示，但有些属性只对模拟量有效，如量程高限、量程低量、静态死区、斜率死区、单位等。

标签类型枚举值：

值	标签类型
0	模拟量
1	开关量
2	数字量

2.2.统计点信息数据格式

SIPHD历史数据库接口通讯上传递的统计点对象为ILTotalInfo，其结构为：

```
ILTotalInfo
{
    bstr   Name;          ///名称
    bstr   Description;   ///描述
    bstr   STag;          ///源点名称
    long   Mode;          ///统计类型
    long   CType;          ///统计函数
    long   Period;        ///采样周期;
    long   TPeriod;       ///统计周期
    long   TimeUnit;      ///统计周期时间单位
    float  High;          ///量程高限
    float  Low;           ///量程低限
    long   SnapTime;      ///强制归档时间
    long   TimeOut;       ///超时时间
    float  Err;           ///压缩算法静态死区
    float  DivErr;        ///压缩算法斜率死区
    bstr   Unit;          ///单位
};
```

ILTotalInfo为统计点信息查询的基本数据对象。

其中，统计点模式枚举值有：

值	统计类型
0	周期计算
1	移动计算
2	累计计算

统计函数枚举值有：

值	统计函数
0	求和
1	平均
2	最小

3	最大
4	范围
5	标准差
6	中间
7	中间差

2.3.中间点信息数据格式

SIPHD历史数据库接口通讯上传递的统计点对象为ILCalcInfo，其结构为：

```
ILCalcInfo
{
    bstr   Name;          ///名称
    bstr   Description;   ///描述
    bstr   Express;       ///计算表达式
    long   Period;        ///采样周期;
    float  High;          ///量程高限
    float  Low;           ///量程低限
    long   SnapTime;      ///强制归档时间
    long   TimeOut;       ///超时时间
    float  Err;           ///压缩算法静态死区
    float  DivErr;        ///压缩算法斜率死区
    bstr   Unit;          ///单位
};
```

ILCalcInfo为统计点信息查询的基本数据对象。

2.4 标签点类型及实时值数据格式

SIPHD历史数据库目前支持的标签点类型有：

- ✧ 模拟量
- ✧ 开关量
- ✧ 数字量

SIPHD历史数据库在网络上传输的实时数据结构为ILValue，其结构为：

```
ILValue
{
    __int64 m_tTime;         ///数据时间戳
    short m_nMillisecond;    ///毫秒值
    short m_nQAStatus;       ///质量位
    short m_nTVStatus;       ///状态位
    VARIANT m_Value;         ///数值
```

};

其中，质量位占2位物理空间，状态位有14位物理空间，状态位的数值含义定义如下表(见TVStatus枚举定义)：

值	数据状态含义
0	TVStatus::tvsNoData，本时间点无数据
1	TVStatus::tvsArchive，归档数据
2	TVStatus::tvsInner，插值数据
3	TVStatus::tvsOuter，外插数据
4	TVStatus::tvsSnap，超时强存数据
5	TVStatus::tvsStop，采集停止点数据
6	TVStatus::tvsStart，采集起始点数据
7	TVStatus::tvsCalc，计算点数据
8	TVStatus::tvsDisCarded，丢弃数据

质量位枚举值：

值	数据状态含义
0	qasGood，正常数据
1	qasGood，坏点数据
2	qasUnCertain，质量状态不确定

实时值存入 m_Value 时应视标签点的类型进行分别赋值：

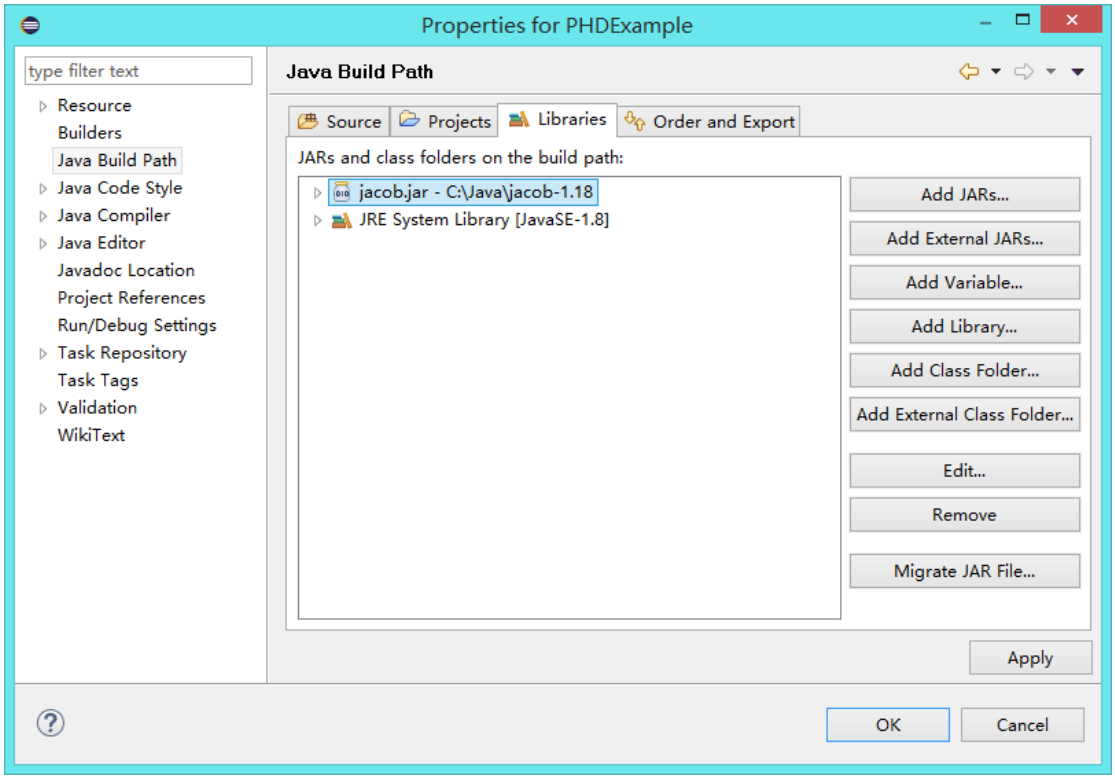
标签点类型	数值类型
模拟量点	m_Value 数据类型 VT_R4;
开关量点	m_Value 数据类型应为 VT_BOOL;
数字量点	m_Value 数据类型 VT_I4;

4、数据采集客户端开发流程

数据采集客户端是指定时将实时数据存储到数据库的客户端程序。
其开发的基本流程为：

4.1 向 Java 项目中添加 Jacob 库

向新建的的Java项目中添加Jacob库，如下图所示；



4.2 连接服务器

数据库通过接口 ILServer::Connect()(详细说明见 3.1.1)接口与指定服务器建立连接。

例如：

```
ILServer pServer = new ActiveXComponent("SiPHDOM.LServer");
Dispatch.call(pServer,"Connect",0, “192.162.1.10”, “test”, “adm”, “123”, 1666);
```

4.3 获取标签点信息

查询服务器数据库中的已创建标签点信息主要通过以下接口：

- ✧ CLServer::QueryTags()，SQL查询指定条件的标签点名称，例如，ID > 2 && ID < 1000等。

采集接口机的标签点信息可以通过上述两接口从服务器在线获取，亦可以从本地点表文件中获取，此时应保证文件中的标签点信息与数据库一致。

接口机标签点信息获取完成以后，通过ILServer接口创建若干个标签点组，

并把标签点加入到相应的标签点组中。

代码示例：

```
String szCmd = "ID>0";
String[] tags = new String[0];
tags = Dispatch.call(pServer,    "QueryTags",szCmd);
```

4.4 创建标签点组

通过ILServer接口创建若干个标签点组，并把标签点加入到相应的标签点组中；

代码示例：

```
Dispatch pGroup= Dispatch.call(pServer,    "NewGroup",
“Read”).toDispatch();

iID1=Dispatch.call(pGroup, "NewItem","tag001").toInt();
iID2=Dispatch.call(pGroup, "NewItem","tag002").toInt();
```

4.5 实时采集设备数据，并实时递交至 SIPHDOM

对现场设备数据进行预处理成TripleItem数据格式，通过ILGroup:: SetTriple()接口对各标签点进行赋值，并定时通过IGroup::Update()接口递交至历史服务器。

代码示例：

```
long rt = Dispatch.call(pGroup, "GetIndex",0).toInt();
//java代码
Date dt=new Date();
long tt=Date2TimeT(dt);
double fValue=88.8f;
Dispatch.call(pGroup, "SetTripleAt",0, tt, 0, 0, 1, fValue);
fValue=77.7f;
Dispatch.call(pGroup, "SetTripleAt",1, tt, 0, 0, 1, fValue);
Dispatch.call(pGroup, "Update",new Variant(true));
```

4.6 连接属性设置

通过对CLServer的RetryNum和TimeOut属性的修改，可以设置SIPHDOM与历史服务器的故障重次数及连接超时时间。

4.7 通讯故障恢复功能

当接口机与服务器通讯发生故障后(调用 ILServer 和 ILGroup 接口时返回异常)，SIPHDOM 将周期进行网络状态的轮询与自动恢复，故障恢复后 SIPHDOM 自动恢复与服务器的连接。

4.8 与服务器断开连接

完成数据采集后，通过 ILServer::DisConnect()接口与服务器断开连接。
代码示例：

```
pServer.DisConnect();
```

5. 数据查询客户端开发流程

查询客户端二次开发流程及接口说明：

5.1 向 Java 项目中添加 SiPHDOM 引用

同4.1；

5.2 连接服务器

同4.2；

5.3 获取标签点信息

同4.3；

5.4 创建标签点组

同 4.4；

5.5 实时数据查询

5.5.1 通过访问组LGroup查询

按照一定的刷新周期调用ILGroup::Update(false)接口来实现标签组本地实时值缓存与服务器的同步，然后通过ILGroup:: GetValue ()接口来读取组内所有标签点的当前值。

代码示例：

```
Dispatch.call(pGroup, "Update",new Variant(false));

  //依次读实时数据
```

```
foreach(int i=0; i < nCount;i++)
{
Dispatch pValue=Dispatch.call(pGroup, "GetValue",iIndex).toDispatch();

    if(pValue!=null)
    {
        long tt=(long)Dispatch.get(pValue, "Time").toInt();
        short uQaulity = Dispatch.get(pValue, "Quality").toShort();
        short uStat = Dispatch.get(pValue, "Stat").toShort();
        double fValue = Dispatch.get(pValue, "Value").toDouble();
        fValue = fValue;
    }
}
```

5.5.2 通过LServer直接查询

通过ILServer的GetRTValues接口，可以同时查询多个标签的实时值。
代码示例：

```
int[] intArray = { 3, 4};
Dispatch pList = Dispatch.call(m_pServer, "GetRTValues",
intArray).toDispatch();

 //依次读实时数据
int iIndex = 0;
int iCount=Dispatch.get(pList, "Count").toInt();
for(int i=0;i<iCount;i++)
{
    Dispatch pValue=Dispatch.call(pList,
"GetValue",i).toDispatch();
    long tt=(long)Dispatch.get(pValue, "Time").toInt();
    short uQaulity = Dispatch.get(pValue, "Quality").toShort();
    short uStat = Dispatch.get(pValue, "Stat").toShort();
    double fValue = Dispatch.get(pValue, "Value").toDouble();
    fValue = fValue;
}
```

5.6 历史时刻值查询

5.6.1 通过访问组Group查询

通过设置Group组对象的Time时间来设置历史时刻，通过调用ILGroup::Update(false)接口来实现历史时刻值的服务器查询，然后通过ILGroup::GetValue ()接口来读取组内所有标签点的查询结果值；
示例代码：

```
Date dt=new Date(115,9,28,11,38,30);
String szTime = dt.toString();
Dispatch.put(pGroup, "Time", dt);
Dispatch.call(pGroup, "Update",new Variant(false));

    //依次读实时数据
  for(int i=0; i < nCount;i++)
  {
      ILValue pValue = pGroup.GetValue(i);
      if(pValue!=null)
      {
          long tt=(long)Dispatch.get(pValue, "Time").toInt();
          short uQaulity = Dispatch.get(pValue, "Quality").toShort();
          short uStat = Dispatch.get(pValue, "Stat").toShort();
          double fValue = Dispatch.get(pValue, "Value").toDouble();
          fValue = fValue;
      }
   }
```

5.6.2 通过LServer直接查询

通过ILServer的GetHTValues接口，可以同时查询多个标签的历史时刻值。代码示例：

```
int[] intArray = {3, 4};
Date dTime=new Date(116,9,30,13,10,0);
Dispatch pList = Dispatch.call(m_pServer, "GetHTValues", intArray, dTime).toDispatch();

int iIndex = 0;
//依次读历史数据
int iCount=Dispatch.get(pList, "Count").toInt();
for(int i=0;i<iCount;i++)
{
Dispatch pValue=Dispatch.call(pList, "GetValue",i).toDispatch();
    long tt=(long)Dispatch.get(pValue, "Time").toInt();
    short uQaulity = Dispatch.get(pValue, "Quality").toShort();
    short uStat = Dispatch.get(pValue, "Stat").toShort();
    double fValue = Dispatch.get(pValue, "Value").toDouble();
    fValue = fValue;
}

return true;
```

5.7 历史时段值查询

5.7.1 超过1天自动取1024点方式查询

通过ILServer:: GetHistValue ()接口可以同时查询标签点的某段时间的历史数据，当起始时间与结束时间的间隔超过1天时，自动获取1024个插值点数据。
示例代码：

```
Date dtStart=new Date(115,8,28,11,39,0);
Date dtEnd = new Date(115,8,28,11,44,0);

Dispatch pList=Dispatch.call(pServer, "GetHistValue",m_iID1, dtStart,
dtEnd).toDispatch();

//依次读历史数据
int iCount=Dispatch.get(pList, "Count").toInt();
for(int i=0;i<iCount;i++)
{
    Dispatch pValue=Dispatch.call(m_pGroup, "GetValue",i).toDispatch();
    long tt=(long)Dispatch.get(pValue, "Time").toInt();
    short uQaulity = Dispatch.get(pValue, "Quality").toShort();
      short uStat = Dispatch.get(pValue, "Stat").toShort();
      double fValue = Dispatch.get(pValue, "Value").toDouble();
      fValue = fValue;

}
```

5.7.2 可指定取点个数方式

通过ILServer:: SelectHistValue ()接口可以同时查询标签点的某段时间的历史数据，可以指定插值取点的个数iSelNum，当iSelNum为0时表示不进行插值，获取所有历史存储数据，当iSelNum大于0时，按照指定个数进行历史数据插值取样，
示例代码：

```
Date dtStart=new Date(116,10,21,10,9,0);
Date dtEnd = new Date(116,10,21,10,30,0);
int iID = 3;

///iSelNum=0为取所有原始值，不进行插值
int iSelNum = 120;

Dispatch pList=Dispatch.call(m_pServer, "SelectHistValue",iID, dtStart, dtEnd,
```

iSelNum).toDispatch();

```
//依次读历史数据
int iCount=Dispatch.get(pList, "Count").toInt();
for(int i=0;i<iCount;i++)
{
    Dispatch pValue=Dispatch.call(pList, "GetValue",i).toDispatch();
    long tt=(long)Dispatch.get(pValue, "Time").toInt();
    short uQaulity = Dispatch.get(pValue, "Quality").toShort();
      short uStat = Dispatch.get(pValue, "Stat").toShort();
      double fValue = Dispatch.get(pValue, "Value").toDouble();
      fValue = fValue;
}
```

5.8 插入历史数据

通过ILServer::NewValueList()可以创建历史数据数组对象，在IValueList中通过ILValueList::NewValue()接口增加历史数据对象，最后通过ILValueList::InsertToServer()接口向服务器插入历史数据。

示例代码：

```
Dispatch pList = Dispatch.call(m_pServer, "NewValueList").toDispatch();

int iID = 3;
//数值准备，依次创建历史数值
int iCount=Dispatch.get(pList, "Count").toInt();
Date dTime=new Date(116,9,30,13,10,0);
long tt=Date2TimeT(dTime);
for(int i=0;i<iCount;i++)
{
Dispatch pValue=Dispatch.call(pList, "NewValue").toDispatch();
    Dispatch.put(pValue, "ID", iID);
    Dispatch.put(pValue, "Time", tt+i);
    Dispatch.put(pValue, "MS", 0);
    Dispatch.put(pValue, "Quality", 0);
    Dispatch.put(pValue, "Stat", 1);
    double fVal = i;
    Dispatch.put(pValue, "Value", fVal);
}
/// 递交历史数据
Dispatch.call(m_pServer, "InsertToServer");
```

5.9 与服务器断开连接

完成查询后，通过 ILServer::DisConnect()接口与服务器断开连接。

6 标签管理功能开发

查询客户端二次开发流程及接口说明：

6.1 向 C#项目中添加 SiPHDOM 引用

同4.1；

6.2 连接服务器

同4.2；

6.3 获取标签点信息

同4.3；

6.4 创建标签点组

同 4.4；

6.5 新建一个标签点

新建一个标签点通过UpdateTag实现，首参数ID为-1表示是新建标签点，否则为更新已有标签属性，新建标签点时数据库中不能已存在同名标签，否则新建失败。

代码示例：

```
long   iID = -1;
String szTagName = "Tag009";       //标签名称
String szDescr = "test tag 001";  //描述
long   iType = 0;                  //数据类型，0模拟量;1开关量;2数字量;
long   iDataSource = 0;
```

```
        long   iPeriod = 1;                 //采集周期
        long   iTimeOut = 10;
        long   iSnapTime = 600;             //超时时间
        float  fHigh = 100;
        float  fLow = 0;
        float  fErr = 0.5f;
        float  fDivErr = 0.5f;
        String szUnit = "kpa";              //单位

        long newID=Dispatch.call(m_pServer,
"UpdateTag",iID,szTagName,szDescr,iType,iDataSource,iPeriod,iTimeOut,
iSnapTime, fHigh, fLow, fErr, fDivErr, szUnit).toInt();
        if (newID != -1)
            return true;
        else
            return false;
```

6.6 同时新建多个标签点

通过调用ILServer::NewTagList接口来创建标签组合列表，然后通过ILTagList::NewTagInfo ()接口来创建新的标签对象ILTagInfo，修改各标签对象的属性后，调用ILTagList::UpdateToServer()接口实现向服务器的最终递交。
示例代码：

```
        Dispatch pTagList = null;
        Dispatch pTag = null;
pTagList = Dispatch.call(m_pServer,  "NewTagList").toDispatch();//

            if (pTagList != null)
            {
                for (int i=0; i < 5; i++)
                {
                    pTag = Dispatch.call(pTagList,
"NewTagInfo").toDispatch();
                    szTagName = String.format("Tag01%d", i);
                    Dispatch.put(pTag, "Name", szTagName);
                    szDescr = String.format("test tag 00%d", i);
                    Dispatch.put(pTag, "Description", szDescr);
                    Dispatch.put(pTag, "Type", iType);
                    Dispatch.put(pTag, "DataSource", iDataSource);
                    Dispatch.put(pTag, "Period", iPeriod);
                    Dispatch.put(pTag, "TimeOut", iTimeOut);
                    Dispatch.put(pTag, "Snap", iSnapTime);
```

```
            Dispatch.put(pTag, "High", fHigh);
            Dispatch.put(pTag, "Low", fLow);
            Dispatch.put(pTag, "err", fErr);
            Dispatch.put(pTag, "diverr", fDivErr);
            Dispatch.put(pTag, "Unit", szUnit);
        }
    }

     Dispatch.call(pTagList,  "UpdateToServer").toDispatch();
```

6.7 通过 ID 查询标签信息

通过ILServer:: GetTagInfoByID()接口可以查询标签点的标签信息。
示例代码：

```
        long   iID = 3;
        Dispatch pTag = null;
pTag = Dispatch.call(m_pServer,  "GetTagInfoByID", iID).toDispatch();//

        if (pTag != null)
        {
            szTagName = Dispatch.get(pTag, "Name").toString();

            szDescr = Dispatch.get(pTag, "Description").toString();
            iType = Dispatch.get(pTag, "Type").toInt();
            iDataSource = Dispatch.get(pTag, "DataSource").toInt();
            iPeriod = Dispatch.get(pTag, "Period").toInt();
            iTimeOut = Dispatch.get(pTag, "TimeOut").toInt();
            iSnapTime = Dispatch.get(pTag, "Snap").toInt();
            fHigh = Dispatch.get(pTag, "High").toFloat();
            fLow = Dispatch.get(pTag, "Low").toFloat();
            fErr = Dispatch.get(pTag, "err").toFloat();
            fDivErr = Dispatch.get(pTag, "diverr").toFloat();
            szUnit = Dispatch.get(pTag, "Unit").toString();
        }
```

6.8 通过名称查询标签信息

通过 ILServer:: GetTagInfoByName()接口可以查询标签点的标签信息。
示例代码：

```
        String szTagName = "Tag001";       //标签名称
        Dispatch pTag = null;
```

```
pTag = Dispatch.call(m_pServer,  "GetTagInfoByName",
szTagName).toDispatch();//

            if (pTag != null)
            {
                iID = Dispatch.get(pTag, "ID").toInt();

                szDescr = Dispatch.get(pTag, "Description").toString();
                iType = Dispatch.get(pTag, "Type").toInt();
                iDataSource = Dispatch.get(pTag, "DataSource").toInt();
                iPeriod = Dispatch.get(pTag, "Period").toInt();
                iTimeOut = Dispatch.get(pTag, "TimeOut").toInt();
                iSnapTime = Dispatch.get(pTag, "Snap").toInt();
                fHigh = Dispatch.get(pTag, "High").toFloat();
                fLow = Dispatch.get(pTag, "Low").toFloat();
                fErr = Dispatch.get(pTag, "err").toFloat();
                fDivErr = Dispatch.get(pTag, "diverr").toFloat();
                szUnit = Dispatch.get(pTag, "Unit").toString();
            }
```

6.9 通过 SQL 语句查询标签信息

通过 ILServer:: SQLQueryTag()接口可以实现 SQL 语句过滤查询标签点的标签信息。

示例代码：

```
        Dispatch pTagList = null;
        Dispatch pTag = null;
pTagList = Dispatch.call(m_pServer,  "SQLQueryTag",
szSQL).toDispatch();//

            if (pTagList != null)
            {
                long nNum = Dispatch.get(pTagList, "Count").toInt();
                for (int i=0; i < nNum; i++)
                {
                    pTag = Dispatch.call(pTagList, "GetTagInfo",
i).toDispatch();
                    iID = Dispatch.get(pTag, "ID").toInt();

                    szDescr = Dispatch.get(pTag,
"Description").toString();
                    iType = Dispatch.get(pTag, "Type").toInt();
                    iDataSource = Dispatch.get(pTag,
"DataSource").toInt();
```

```
                iPeriod = Dispatch.get(pTag, "Period").toInt();
                iTimeOut = Dispatch.get(pTag, "TimeOut").toInt();
                iSnapTime = Dispatch.get(pTag, "Snap").toInt();
                fHigh = Dispatch.get(pTag, "High").toFloat();
                fLow = Dispatch.get(pTag, "Low").toFloat();
                fErr = Dispatch.get(pTag, "err").toFloat();
                fDivErr = Dispatch.get(pTag, "diverr").toFloat();
                szUnit = Dispatch.get(pTag, "Unit").toString();
            }
```

6.10 获取所有标签信息

通过 ILServer:: GetAllTag()接口可以查询所有标签点的标签信息，包括模拟量、开关量和数字量。

示例代码：

```
        Dispatch pTagList = null;
        Dispatch pTag = null;
pTagList = Dispatch.call(m_pServer,  "GetAllTag").toDispatch();//

            if (pTagList != null)
            {
                long nNum = Dispatch.get(pTagList, "Count").toInt();
                for (int i=0; i < nNum; i++)
                {
                    pTag = Dispatch.call(pTagList, "GetTagInfo",
i).toDispatch();
                    iID = Dispatch.get(pTag, "ID").toInt();

                    szDescr = Dispatch.get(pTag,
"Description").toString();
                    iType = Dispatch.get(pTag, "Type").toInt();
                    iDataSource = Dispatch.get(pTag,
"DataSource").toInt();
                    iPeriod = Dispatch.get(pTag, "Period").toInt();
                    iTimeOut = Dispatch.get(pTag, "TimeOut").toInt();
                    iSnapTime = Dispatch.get(pTag, "Snap").toInt();
                    fHigh = Dispatch.get(pTag, "High").toFloat();
                    fLow = Dispatch.get(pTag, "Low").toFloat();
                    fErr = Dispatch.get(pTag, "err").toFloat();
                    fDivErr = Dispatch.get(pTag, "diverr").toFloat();
                    szUnit = Dispatch.get(pTag, "Unit").toString();
                }
            }
```

6.11 通过 ID 删除标签点

通过 ILServer:: DeleteTagByID()接口可以删除指定的标签点。
示例代码：

```
long   iID = 3;
String szTagName = "Tag001";
Dispatch.call(m_pServer,  "DeleteTagByName", szTagName);//
```

6.12 通过名称删除标签点

通过 ILServer:: DeleteTagByName()接口可以删除指定的标签点。
示例代码：

```
String szTagName = "Tag001";
Dispatch.call(m_pServer,  "DeleteTagByName", szTagName);//
```

6.13 新建一个统计点

通过 ILServer:: UpdateTotal()接口可以新建一个统计点。
示例代码：

```
String szName = "Total001";       //标签名称
String szDescr = "test Total 001";   //描述
String szSTag = "Tag001";           //源点名称
long   iMode = 0;                  //统计类型;
long   iCType = 0;                 //统计函数
long   iPeriod = 1;                //采样周期
long   iTPeriod = 60;              //统计周期
long   iTimeUnit = 0;              //统计周期时间单位
float  fHigh = 100;
float  fLow = 0;
long   iSnaptime = 600;               //强制归档时间
long   iTimeOut = 10;                 //超时时间
float  fErr = 100;
float  fDivErr = 0;

String szUnit = "kpa";             //单位

Dispatch.call(m_pServer,
"UpdateTotal",szName,szDescr,szSTag,iMode,iCType,iPeriod,iTPeriod,iTi
meUnit, fHigh, fLow, iSnapTime, iTimeOut, fErr, fDivErr, szUnit);
```

6.14 同时新建多个统计点

通过调用ILServer::NewTotalList接口来创建统计点组合列表，然后通过ILTagList:: NewTotalInfo ()接口来创建新的标签对象ILTotalInfo，修改各标签对象的属性后，调用ILTotalList::UpdateToServer()接口实现向服务器的最终递交。
示例代码：

```
        Dispatch pTotalList = null;
        Dispatch pTotal = null;
pTotalList = Dispatch.call(m_pServer,  "NewTotalList").toDispatch();//

            if (pTotalList != null)
            {
                for (int i=0; i < 5; i++)
                {
                    pTotal = Dispatch.call(pTotalList,
"NewTotalInfo").toDispatch();
                    szTagName = String.format("Total00%d", i+1);
                    Dispatch.put(pTotal, "Name", szTagName);
                    szDescr = String.format("test tag 00%d", i);
                    Dispatch.put(pTotal, "Description", szDescr);
                    szSTag = String.format("Tag00%d", i+1);
                    Dispatch.put(pTotal, "STag", szSTag);
                    Dispatch.put(pTotal, "Mode", iMode);
                    Dispatch.put(pTotal, "CType", iCType);
                    Dispatch.put(pTotal, "Period", iPeriod);
                    Dispatch.put(pTotal, "TPeriod", iTPeriod);
                    Dispatch.put(pTotal, "TimeUnit", iTimeUnit);
                    Dispatch.put(pTotal, "High", fHigh);
                    Dispatch.put(pTotal, "Low", fLow);
                    Dispatch.put(pTotal, "SnapTime", iSnapTime);
                    Dispatch.put(pTotal, "TimeOut", iTimeOut);
                    Dispatch.put(pTotal, "Err", fErr);
                    Dispatch.put(pTotal, "DivErr", fDivErr);
                    Dispatch.put(pTotal, "Unit", szUnit);
                }
            }

            Dispatch.call(pTotalList,  "UpdateToServer").toDispatch();
```

6.15 通过名称查询统计点信息

通过 ILServer:: GetTotalInfo()接口可以查询统计点的属性信息。
示例代码：

```
        Dispatch pTotal = null;
pTotal = Dispatch.call(m_pServer,  "GetTotalInfo",
szName).toDispatch();//

            if (pTotal != null)
            {
                szDescr = Dispatch.get(pTotal,
"Description").toString();
                szSTag = Dispatch.get(pTotal, "STag").toString();
                iMode = Dispatch.get(pTotal, "Mode").toInt();
                iCType = Dispatch.get(pTotal, "CType").toInt();
                iPeriod = Dispatch.get(pTotal, "Period").toInt();
                iTPeriod = Dispatch.get(pTotal, "TPeriod").toInt();
                iTimeUnit = Dispatch.get(pTotal, "TimeUnit").toInt();
                fHigh = Dispatch.get(pTotal, "High").toFloat();
                fLow = Dispatch.get(pTotal, "Low").toFloat();
                iSnapTime = Dispatch.get(pTotal, "SnapTime").toInt ();
                iTimeOut = Dispatch.get(pTotal, "TimeOut").toInt ();
                fErr = Dispatch.get(pTotal, "Err").toFloat();
                fDivErr =  Dispatch.get(pTotal, "DivErr").toFloat();
                szUnit = Dispatch.get(pTotal, "Unit").toString();
            }
```

6.16 获取所有统计点信息

通过 ILServer:: GetAllTotal()接口可以查询所有统计点的信息集合。
示例代码：

```
        Dispatch pTotalList = null;
        Dispatch pTotal = null;
pTotalList = Dispatch.call(m_pServer,  "GetAllTotal").toDispatch();//

            if (pTotalList != null)
            {
                long nNum = Dispatch.get(pTotalList, "Count").toInt();

                for (int i=0; i < nNum; i++)
                {
```

```
                szName = Dispatch.get(pTotal, "Name").toString();
                szDescr = Dispatch.get(pTotal,
"Description").toString();
                szSTag = Dispatch.get(pTotal, "STag").toString();
                iMode = Dispatch.get(pTotal, "Mode").toInt();
                iCType = Dispatch.get(pTotal, "CType").toInt();
                iPeriod = Dispatch.get(pTotal, "Period").toInt();
                iTPeriod = Dispatch.get(pTotal, "TPeriod").toInt();
                iTimeUnit = Dispatch.get(pTotal, "TimeUnit").toInt();
                fHigh = Dispatch.get(pTotal, "High").toFloat();
                fLow = Dispatch.get(pTotal, "Low").toFloat();

                szUnit = Dispatch.get(pTotal, "Unit").toString();
            }
        }
```

6.17 删除统计点

通过 ILServer:: DeleteTotal()接口可以删除指定统计点。
示例代码：

```
        String szName = "Total001";
        Dispatch.call(m_pServer,  "DeleteTotal", szName);//
```

7.18 新建一个中间点

通过 ILServer:: UpdateCalc()接口可以新建一个中间点。
示例代码：

```
        String szName = "Calc001";          //标签名称
        String szDescr = "test Calc 001";   //描述
        String szExpress = "Tag001+2";           //源点名称
        long   iMode = 0;                  //统计类型;
        long   iCType = 0;                 //统计函数
        long   iPeriod = 1;                //采样周期
        float  fHigh = 100;
        float  fLow = 0;
        String szUnit = "kpa";              //单位
        long   iSnaptime = 600;                //强制归档时间
        long   iTimeOut = 10;                  //超时时间
        float  fErr = 100;
        float  fDivErr = 0;
```

```
        Dispatch.call(m_pServer,
"UpdateCalc",szName,szDescr,iPeriod,szExpress,fHigh, fLow, iSnapTime,
iTimeOut, fErr, fDivErr, szUnit);
```

6.19 同时新建多个中间点

通过调用ILServer::NewCalcList接口来创建中间点组合列表，然后通过ILTagList:: NewCalcInfo ()接口来创建新的标签对象ILCalcInfo，修改各标签对象的属性后，调用ILCalcList::UpdateToServer()接口实现向服务器的最终递交。
示例代码：

```
        Dispatch pCalcList = null;
        Dispatch pCalc = null;
pCalcList = Dispatch.call(m_pServer,  "NewCalcList").toDispatch();//

            if (pCalcList != null)
            {
                for (int i=0; i < 5; i++)
                {
                    pCalc = Dispatch.call(pCalcList,
"NewCalcInfo").toDispatch();
                    szName = String.format("Calc00%d", i+2);
                    Dispatch.put(pCalc, "Name", szName);
                    szDescr = String.format("test tag 00%d", i);
                    Dispatch.put(pCalc, "Description", szDescr);
                    szExpress = String.format("Tag00%d+2", i+2);
                    Dispatch.put(pCalc, "Express", szExpress);
                    Dispatch.put(pCalc, "Period", iPeriod);
                    Dispatch.put(pCalc, "High", fHigh);
                    Dispatch.put(pCalc, "Low", fLow);
                    Dispatch.put(pCalc, "Unit", szUnit);
                    Dispatch.put(pCalc, "SnapTime", iSnapTime);
                    Dispatch.put(pCalc, "TimeOut", iTimeOut);
                    Dispatch.put(pCalc, "Err", fErr);
                    Dispatch.put(pCalc, "DivErr", fDivErr);

                }
            }

            Dispatch.call(pCalcList,  "UpdateToServer").toDispatch();
```

6.20 通过名称查询中间点信息

通过 ILServer:: GetCalcInfo()接口可以查询中间点的属性信息。
示例代码：

```
        Dispatch pCalc = null;
pCalc = Dispatch.call(m_pServer,  "GetCalcInfo",
szName).toDispatch();//

            if (pCalc != null)
            {
                szDescr = Dispatch.get(pCalc, "Description").toString();
                szSTag = Dispatch.get(pCalc, "STag").toString();
                iMode = Dispatch.get(pCalc, "Mode").toInt();
                iCType = Dispatch.get(pCalc, "CType").toInt();
                iPeriod = Dispatch.get(pCalc, "Period").toInt();
                iTPeriod = Dispatch.get(pCalc, "TPeriod").toInt();
                iTimeUnit = Dispatch.get(pCalc, "TimeUnit").toInt();
                fHigh = Dispatch.get(pCalc, "High").toFloat();
                fLow = Dispatch.get(pCalc, "Low").toFloat();
                iSnapTime = Dispatch.get(pTotal, "SnapTime").toInt ();
                iTimeOut = Dispatch.get(pTotal, "TimeOut").toInt ();
                fErr = Dispatch.get(pTotal, "Err").toFloat();
                fDivErr =  Dispatch.get(pTotal, "DivErr").toFloat();
                szUnit = Dispatch.get(pCalc, "Unit").toString();
            }
```

6.21 获取所有中间点信息

通过 ILServer:: GetAllCalc()接口可以查询所有中间点的信息集合。
示例代码：

```
        Dispatch pCalcList = null;
        Dispatch pCalc = null;
pCalcList = Dispatch.call(m_pServer,  "GetAllCalc").toDispatch();//

            if (pCalcList != null)
            {
                long nNum = Dispatch.get(pCalcList, "Count").toInt();

                for (int i=0; i < nNum; i++)
                {
                    szName = Dispatch.get(pCalc, "Name").toString();
```

```
                szDescr = Dispatch.get(pCalc,
"Description").toString();
                szSTag = Dispatch.get(pCalc, "STag").toString();
                iMode = Dispatch.get(pCalc, "Mode").toInt();
                iCType = Dispatch.get(pCalc, "CType").toInt();
                iPeriod = Dispatch.get(pCalc, "Period").toInt();
                iTPeriod = Dispatch.get(pCalc, "TPeriod").toInt();
                iTimeUnit = Dispatch.get(pCalc, "TimeUnit").toInt();
                fHigh = Dispatch.get(pCalc, "High").toFloat();
                fLow = Dispatch.get(pCalc, "Low").toFloat();

                szUnit = Dispatch.get(pCalc, "Unit").toString();
            }
        }
```

6.22 删除中间点

通过 ILServer:: DeleteCalc()接口可以删除指定中间点。
示例代码：

```
        String szName = "Calc001";
        Dispatch.call(m_pServer,  "DeleteCalc", szName);//
```

7 ILServer 接口说明

7.1 ILServer::Connect

```
HRESULT Connect(
        [in] LConnectionType eType,
        [in] BSTR bstrHost,
        [in] BSTR bstrDBName,
        [in] BSTR bstrUserName,
        [in] BSTR bstrPassword,
        [in] VARIANT vParam);
```

描述： 与指定服务器建立连接；

参数	说明
eType	指定 siphdDom 与远方服务器的通信协议，

	✧ CT_TCP：TCP 协议 ✧ CT_DCOM：DCOM 方式 ✧ CT_UDP：UDP 协议
bstrDBName	PHD 服务器名字或 IP
bstrDBName	SIS 数据库名称
bstrUserName	该数据库的登录用户名
bstrPassword	数据库登录密码
vParam	连接参数，在 TCP 或 UDP 协议下指与服务器通信的端口号，默认为 1666。

返回值：

返回值	说明
S_OK	连接成功
其它	连接失败

7.2 ILServer::DisConnect

```
HRESULT DisConnect();
```

描述：

与服务器断开连接；

返回值：

返回值	说明
S_OK	断开成功
其它	断开失败

7.3 ILServer:: NewGroup

```
HRESULT NewGroup(
[in] BSTR bstrGrpName,
[out,retval] ILGroup ** ppVal);
```

描述： 新建标签点组.

参数	说明
bstrGrpName	标签组名称
ILGroup	新生成标签组对象的访问指针

返回值：

返回值	说明
S_OK	创建成功
其它	创建失败

7.4 ILServer:: RemoveGroup

```
HRESULT RemoveGroup (
[in] BSTR bstrGrpName);
```

描述：

删除指定标签点组.

参数	说明
bstrGrpName	标签组名称

返回值：

返回值	说明
S_OK	删除成功
其它	删除失败

7.5 ILServer:: GetHistValue

```
HRESULT GetHistValue([in] VARIANT ID,
[in] DATE Start,
[in] DATE End,
[out,retval] ILValuList* pts);
```

描述：

查询指定ID的一段历史时间内的所有归档历史数据.

参数	说明
ID	标签点 ID
Start	起始时间
End	结束时间

pts	查询结果返回值，为 IValueList 对象

返回值：

返回值	说明
S_OK	递交成功
其它	递交失败

7.6 ILServer:: SelectHistValue

```
HRESULT SelectHistValue([in] VARIANT ID,
[in] DATE Start,
[in] DATE End,
[in] ULONG iSelNum,
[out,retval] ILValuList* pts);
```

描述：

查询指定ID的一段历史时间内的所有归档历史数据.

参数	说明
ID	标签点 ID
Start	起始时间
End	结束时间
iSelNum	插值取样个数，等于 0 时表示不插值，返回所有存储原始数据。
pts	查询结果返回值，为 IValueList 对象

返回值：

返回值	说明
S_OK	递交成功
其它	递交失败

7.7 ILServer:: GetGroup

```
HRESULT GetGroup(
[in] BSTR bstrGrpName,
[out,retval] ILGroup ** ppVal);
```

描述：

查询指令标签组的访问指针.

参数	说明

bstrGrpName	待查询标签组名
ppVal	标签组对象访问指针

返回值：

返回值	说明
S_OK	查询成功
其它	查询成功

7.8 ILServer:: QueryTags

```
HRESULT QueryTag(
[in] BSTR bstrCmd,
[out,retval] VARIANT *pNames);
```

描述：

查询标签点信息.

参数	说明
bstrCmd	查询 SQL 语句
pNames	查询结果数据,字符串数组

返回值：

返回值	说明
S_OK	查询成功
其它	查询失败

7.9 ILServer::UpdateTag

```
HRESULT UpdateTag(
[in] LONG iID,
[in] BSTR bstrName,
[in] BSTR bstrDescr,
[in] LONG iType,
[in] LONG iDataSource,
[in] LONG iPeriod,
[in] LONG iTimeOut,
[in] LONG iSnapTime,
[in] float fHigh,
[in] float fLow,
[in] float fErr,
[in] float fDivErr,
```

[in] BSTR bstrUnit,
[out,retval] long *NewID);

描述：

新建或更新标签点信息，包括模拟量、开关量和数字量点都可以用此接口.当iID为-1时，表示新建标签点，新建成功NewID将返回新建点的ID；当iID不为-1时表示是更新指定ID标签的标签属性。

参数	说明
iID	指定标签 ID，-1 表示新建标签点
bstrName	标签点名称
bstrDescr	标签点描述
iType	标签类型,0 模拟量;1 开关量;2 数字量
iDataSource	数据源 ID
iPeriod	采集周期，单位为秒
iTimeOut	通讯超时时间，单位为秒
iSnapTime	强制归档最大间隔时间，单位为秒
fHigh	量程高限，模拟量有效
fLow	量程低限，模拟量有效
fErr	压缩算法静态死区，一般取满量程千分之五，设为 0 表示不进行压缩
fDivErr	压缩算法斜率死区，一般取满量程千分之五，设为 0 表示不进行压缩
bstrUnit	单位，模拟量有效
NewID	新建标签时，返回新建点 ID

返回值：

返回值	说明
S_OK	递交成功
其它	递交失败

7.10 ILServer:: NewTagList

HRESULT NewTagList(
[out,retval] ILTagList** pTagList);

描述：

创建标签列表对象，创建成功后可以向标签列表中添加标签对象，用于批量处理标签点信息.

参数	说明

pTagList	创建标签列表

返回值：

返回值	说明
S_OK	创建成功
其它	创建失败

7.11 ILServer::GetTagInfoByID

```
HRESULT GetTagInfoByID(
[in] LONG iID,
[out,retval] ILTagInfo** pTagInfo);
```

描述：

通过ID查询指定标签点的标签信息.

参数	说明
iID	标签点 ID
pTagInfo	返回指定 ID 的标签信息对象，标签点不存在返回 NULL

返回值：

返回值	说明
S_OK	查询成功
其它	查询失败

7.12 ILServer::GetTagInfoByName

```
HRESULT GetTagInfoByName(
[in] BSTR bstrName,
[out,retval] ILTagInfo** pTagInfo);
```

描述：

通过名称查询指定标签点的标签信息.

参数	说明
bstrName	指定标签的名称
pTagInfo	返回指定名称的标签信息对象，标签点不存在返回 NULL

返回值：

返回值	说明

S_OK	查询成功
其它	查询失败

7.13 ILServer::GetIDofNames

```
HRESULT GetIDofNames(
[in] Variant Names,
[out,retval] Variant* IDs);
```

描述：

查询多个标签点名称的ID值.

参数	说明
Names	标签名称数组
IDs	返回 ID 数组

返回值：

返回值	说明
S_OK	查询成功
其它	查询失败

7.14 ILServer::SQLQueryTag

```
HRESULT SQLQueryTag(
[in] BSTR bstrCmd,
[out,retval] ILTagList** pTagList);
```

描述：

通过SQL语句查询标签点信息.

参数	说明
bstrCmd	SQL 查询语句，如 ID > 0
pTagList	查询结果，标签信息列表对象

返回值：

返回值	说明
S_OK	查询成功
其它	查询失败

7.15 ILServer::GetAllTag

```
HRESULT GetAllTag(
[out,retval] ILTagList** pTagList);
```

描述：

获取所有标签点的标签信息.

参数	说明
pTagList	查询结果，标签信息列表对象

返回值：

返回值	说明
S_OK	查询成功
其它	查询失败

7.16 ILServer::DeleteTagByID

```
HRESULT DeleteTagByID(
[in] LONG iID);
```

描述：

删除指定ID的标签点.

参数	说明
iID	标签点 ID

返回值：

返回值	说明
S_OK	删除成功
其它	删除失败

7.17 ILServer::DeleteTagByName

```
HRESULT DeleteTagByName(
[in] BSTR bstrName);
```

描述：

删除指定名称的标签点.

参数	说明

bstrName	指定标签的名称

返回值：

返回值	说明
S_OK	删除成功
其它	删除失败

7.18 ILServer::UpdateTotal

```
HRESULT UpdateTotal(
[in] BSTR bstrName,
[in] BSTR bstrDescr,
[in] BSTR bstrSTag,
[in] LONG iMode,
[in] LONG iCType,
[in] LONG iPeriod,
[in] LONG iTPeriod,
[in] LONG iTimeUnit,
[in] float fHigh,
[in] float fLow,
[in] LONG  iSnapTime,
[in] LONG  iTimeOut,
[in] float fErr,
[in] float fDivErr,
[in] BSTR bstrUnit);
```

描述：

新建或更新统计点信息。

参数	说明
bstrName	统计点名称
bstrDescr	统计点描述
bstrSTag	源点名称
iMode	统计类型
iCType	统计函数
iPeriod	采集周期，单位为秒
iTPeriod	统计周期

iTimeUnit	统计周期的时间单位
fHigh	量程高限
fLow	量程低限
iSnapTime	强制归档最大间隔时间，单位为秒
iTimeOut	通讯超时时间，单位为秒
fErr	压缩算法静态死区，一般取满量程千分之五，设为 0 表示不进行压缩
fDivErr	压缩算法斜率死区，一般取满量程千分之五，设为 0 表示不进行压缩
bstrUnit	单位

返回值：

返回值	说明
S_OK	递交成功
其它	递交失败

其中，统计类型iMode的枚举值表为：

值	统计类型
0	周期计算
1	移动计算
2	累计计算

统计函数枚举值表为：

值	统计函数
0	求和
1	平均
2	最小
3	最大
4	范围
5	标准差
6	中间
7	中间差

统计周期的时间单位枚举值为：

值	时间单位
0	秒

1	分钟
2	小时
3	天
4	周
5	月
6	季
7	年
8	无限长

7.19 ILServer:: NewTotalList

```
HRESULT NewTotalList(
[out,retval] ILTotalList** pTotalList);
```

描述：

创建统计点列表对象，创建成功后可以向统计点列表中添加统计点对象，用于批量处理统计点信息.

参数	说明
pTotalList	创建统计点列表

返回值：

返回值	说明
S_OK	创建成功
其它	创建失败

7.20 ILServer::GetTotalInfo

```
HRESULT GetTotalInfo (
[in] BSTR bstrName,
[out,retval] ILTotalInfo** pTotalInfo);
```

描述：

通过名称查询指定统计点的属性信息.

参数	说明
bstrName	指定统计点的名称
pTotalInfo	返回指定名称的统计点信息对象，统计点不存在返回 NULL

返回值：

返回值	说明
S_OK	查询成功
其它	查询失败

7.21 ILServer::GetAllTotal

```
HRESULT GetAllTotal(
[out,retval] ILTotalList** pTotalList);
```

描述：

获取所有统计点的属性信息.

参数	说明
pTotalList	查询结果，统计点信息列表对象

返回值：

返回值	说明
S_OK	查询成功
其它	查询失败

7.22 ILServer::DeleteTotal

```
HRESULT DeleteTotal(
[in] BSTR bstrName);
```

描述：

删除指定名称的统计点.

参数	说明
bstrName	指定统计点的名称

返回值：

返回值	说明
S_OK	删除成功
其它	删除失败

7.23 ILServer::UpdateCalc

```
HRESULT UpdateCalc(
[in] BSTR bstrName,
[in] BSTR bstrDescr,
[in] BSTR bstrExpress,
[in] LONG iPeriod,
[in] float fHigh,
[in] float fLow,
[in] LONG  iSnapTime,
[in] LONG  iTimeOut,
[in] float fErr,
[in] float fDivErr,
[in] BSTR bstrUnit);
```

描述：

新建或更新计算点信息。

参数	说明
bstrName	计算点名称
bstrDescr	计算点描述
bstrExpress	计算表达式
iPeriod	采集周期，单位为秒
fHigh	量程高限
fLow	量程低限
iSnapTime	强制归档最大间隔时间，单位为秒
iTimeOut	通讯超时时间，单位为秒
fErr	压缩算法静态死区，一般取满量程千分之五，设为 0 表示不进行压缩
fDivErr	压缩算法斜率死区，一般取满量程千分之五，设为 0 表示不进行压缩
bstrUnit	单位

返回值：

返回值	说明
S_OK	递交成功
其它	递交失败

7.24 ILServer:: NewCalcList

```
HRESULT NewCalcList(
[out,retval] ILCalcList** pCalcList);
```

描述：

创建计算点列表对象，创建成功后可以向计算点列表中添加计算点对象，用于批量处理计算点信息.

参数	说明
pCalcList	创建计算点列表

返回值：

返回值	说明
S_OK	创建成功
其它	创建失败

7.25 ILServer::GetCalcInfo

```
HRESULT GetCalcInfo (
[in] BSTR bstrName,
[out,retval] ILCalcInfo** pCalcInfo);
```

描述：

通过名称查询指定计算点的属性信息.

参数	说明
bstrName	指定计算点的名称
pCalcInfo	返回指定名称的计算点信息对象，计算点不存在返回 NULL

返回值：

返回值	说明
S_OK	查询成功
其它	查询失败

7.26 ILServer::GetAllCalc

```
HRESULT GetAllCalc(
[out,retval] ILCalcList** pCalcList);
```

描述：

获取所有计算点的属性信息.

参数	说明
pCalcList	查询结果，计算点信息列表对象

返回值：

返回值	说明
S_OK	查询成功
其它	查询失败

7.27 ILServer::DeleteCalc

```
HRESULT DeleteCalc(
[in] BSTR bstrName);
```

描述：

删除指定名称的计算点.

参数	说明
bstrName	指定计算点的名称

返回值：

返回值	说明
S_OK	删除成功
其它	删除失败

7.28 ILServer:: GetRTValues

```
HRESULT GetRTValues([in] VARIANT IDs,
[out,retval]ILValueList* pValueList);
```

描述：

查询指定多个ID的实时值数据.

参数	说明
IDs	标签点 ID 数组
pValueList	查询实时值结果列表

返回值：

返回值	说明
S_OK	递交成功
其它	递交失败

7.29 ILServer:: GetHTValues

```
HRESULT GetHTValues([in] VARIANT IDs,
[in] DATE dTime,
[out,retval]ILValueList* pValueList);
```

描述：

查询指定多个ID的历史时刻值数据.

参数	说明
IDs	标签点 ID 数组
dTime	历史时刻时间
pValueList	查询实时值结果列表

返回值：

返回值	说明
S_OK	递交成功
其它	递交失败

7.30 ILServer::NewValueList

```
HRESULT NewValueList(
[out,retval]ILValueList* pValueList);
```

描述：

创建历史数据数组对象.

参数	说明
pValueList	查询实时值结果列表

返回值：

返回值	说明
S_OK	创建成功
其它	创建失败

8 ILGroup 接口说明

8.1 ILGroup::NewItem

```
HRESULT NewItem(
[in] BSTR bstrTag,
[out, retval] long *pnID);
```

描述： 向组内添加新标签点，若添加成功(数据库内存在该点)，则返回该标签点的ID。

参数	说明
bstrTag	标签点名称
pnID	查询成功时带回该标签点的 ID 值

返回值：

返回值	说明
S_OK	添加成功，数据中存在指定标签名
其它	添加成功，该标签点名不存在或通讯故障

8.2 ILGroup::NewItems

```
HRESULT NewItems(
[in] VARIANT Tags,
[out, retval] VARIANT* IDs);
```

描述：向标签点组同时添加多个标签点

参数	说明
Tags	标签点名称数组(VT_BSTR 类型的 SAFEArray 结构)
IDs	VT_UINT 类型的 SafeArray 结构； 若点存在则返回其 ID,若点不存在返回 ID=0xFFFFFFFF

返回值：

返回值	说明
S_OK	添加成功
其它	添加失败

8.3 ILGroup::Clear

HRESULT Clear();
描述： 清空组内所有标签点
返回值：

返回值	说明
S_OK	清除成功
其它	清除失败

8.4 ILGroup::Delete

HRESULT Delete(void);
描述： 删除当前组对象
返回值：

返回值	说明
S_OK	删除成功
其它	删除失败

8.5 ILGroup::Update

HRESULT Update(
[in] VARIANT_BOOL bUpdateToServer);
描述： 与服务器进行一次实时数据同步，bUpdate为true，表示存储组内实时数据至服务器，false表示从服务器读取实时数据.

参数	说明
bUpdateToServer	true，递交组内实时值至服务器， false,从服务器读取实时值刷新组内实时值

返回值：

返回值	说明
S_OK	同步成功
其它	同步失败

8.6 ILGroup::SetTriple

HRESULT SetTriple(
[in] ULONG nID,
[in] LONGLONG lTime,

[in] SHORT nMS,
[in] SHORT nQAStatus,
[in] SHORT nTVStatus,
[in] VARIANT vValue);

描述： 刷新标签组内指定ID标签点的实时值，nID为标签点在数据库的全局唯一ID，对组内本地实时值缓存进行刷新；

参数	说明
nID	标签点 ID，数据库内全局唯一
lTime	数值时间戳
nMS	毫秒值
nQSStatus	数值质量位，2 位地址空间
nTVStatus	数值状态值，14 位地址空间
vValue	实时值

返回值：

返回值	说明
S_OK	刷新成功
其它	刷新失败

8.7 ILGroup::SetTripleAt

HRESULT SetTripleAt(
[in] ULONG nIndex,
[in] LONGLONG lTime,
[in] SHORT nMS,
[in] SHORT nQAStatus,
[in] SHORT nTVStatus,
[in] VARIANT vValue);

描述：刷新组内指定次序标签点的实时值，其中nIndex为标签点在组内缓存中的次序，对组内本地实时值缓存进行刷新。

参数	说明
nIndex	为标签点在标签组内缓存中的次序
lTime	数值时间戳
nMS	毫秒值
nQSStatus	数值质量位
nTVStatus	数值状态值
vValue	实时值

返回值：

返回值	说明
S_OK	刷新成功
其它	刷新失败

8.8 ILGroup::GetValue

```
HRESULT GetValue(
[in] ULONG nIndex,
[out,retval] ILValue* ppVal);
```

描述：取组内指定次序点的实时值，该当前值指组内本地缓存内的实时值。

参数	说明
nIndex	为标签点在标签组内缓存中的次序
ppVal	返回数值，为 ILValue 类型

返回值：

返回值	说明
S_OK	刷新成功
其它	刷新失败

9 ILTagList 接口说明

9.1 ILTagList::get_Count

```
HRESULT get_Count(
[out, retval] long *pnCount);
```

描述： 查询标签列表中标签个数.

参数	说明
pnCount	返回标签数目

返回值：

返回值	说明
S_OK	查询成功
其它	查询失败

9.2 ILTagList::GetTagInfo

```
HRESULT GetTagInfo(
[in] LONG iIndex,
[out,retval] ILTagInfo** pTagInfo);
```

描述：

通过查询指定次序的标签信息.

参数	说明
iIndex	标签点序号
pTagInfo	返回指定的标签信息对象，标签点不存在返回 NULL

返回值：

返回值	说明
S_OK	查询成功
其它	查询失败

9.3 ILTagList::NewTagInfo

```
HRESULT NewTagInfo(
[out,retval] ILTagInfo** pTagInfo);
```

描述：

新建新的标签信息.

参数	说明
pTagInfo	返回创建成功的标签信息对象，创建成败则返回 NULL

返回值：

返回值	说明
S_OK	创建成功
其它	创建失败

9.4 ILTagList::UpdateToServer

```
HRESULT UpdateToServer();
```

描述：

把标签列表中的所有标签信息向服务器递交，用此方法可以同时新建多个标

签点.

参数	说明

返回值：

返回值	说明
S_OK	同步成功
其它	同步失败

10 ILTotalList 接口说明

10.1 ILTotalList::get_Count

```
HRESULT get_Count(
[out, retval] long *pnCount);
```

描述： 查询统计点列表中统计点个数.

参数	说明
pnCount	返回统计点数目

返回值：

返回值	说明
S_OK	查询成功
其它	查询失败

10.2 ILTotalList::GetTotalInfo

```
HRESULT GetTotalInfo(
[in] LONG iIndex,
[out,retval] ILTotalInfo** pTotalInfo);
```

描述：

通过查询指定次序的统计点信息.

参数	说明
iIndex	统计点序号
pTotalInfo	返回指定的统计信息对象，统计点不存在返回 NULL

返回值：

返回值	说明

S_OK	查询成功
其它	查询失败

10.3 ILTotalList::NewTotalInfo

```
HRESULT NewTotalInfo(
[out,retval] ILTotalInfo** pTotalInfo);
```

描述：

新建新的统计点信息.

参数	说明
pTotalInfo	返回创建成功的统计点信息对象，创建成败则返回 NULL

返回值：

返回值	说明
S_OK	创建成功
其它	创建失败

10.4 ILTotalList::UpdateToServer

```
HRESULT UpdateToServer();
```

描述：

把统计列表中的所有统计信息向服务器递交，用此方法可以同时新建多个统计点.

参数	说明

返回值：

返回值	说明
S_OK	同步成功
其它	同步失败

11 ILCalcList 接口说明

11.1 ILCalcList::get_Count

```
HRESULT get_Count(
[out, retval] long *pnCount);
```

描述： 查询统计点列表中统计点个数.

参数	说明
pnCount	返回统计点数目

返回值：

返回值	说明
S_OK	查询成功
其它	查询失败

11.2 ILCalcList::GetCalcInfo

```
HRESULT GetCalcInfo(
[in] LONG iIndex,
[out,retval] ILCalcInfo** pCalcInfo);
```

描述：

通过查询指定次序的统计点信息.

参数	说明
iIndex	统计点序号
pCalcInfo	返回指定的统计信息对象，统计点不存在返回 NULL

返回值：

返回值	说明
S_OK	查询成功
其它	查询失败

11.3 ILCalcList::NewCalcInfo

```
HRESULT NewCalcInfo(
```

[out,retval] ILCalcInfo** pCalcInfo);

描述：

新建新的统计点信息.

参数	说明
pCalcInfo	返回创建成功的统计点信息对象，创建成败则返回 NULL

返回值：

返回值	说明
S_OK	创建成功
其它	创建失败

11.4 ILCalcList::UpdateToServer

HRESULT UpdateToServer();

描述：

把统计列表中的所有统计信息向服务器递交，用此方法可以同时新建多个统计点.

参数	说明

返回值：

返回值	说明
S_OK	同步成功
其它	同步失败

12 ILValueList 接口说明

12.1 ILValueList::get_Count

HRESULT get_Count(
[out, retval] long *pnCount);

描述： 查询数值点个数.

参数	说明
pnCount	返回统计点数目

返回值：

返回值	说明

S_OK	查询成功
其它	查询失败

12.2 ILValueList::GetValue

```
HRESULT GetValue(
[in] LONG iIndex,
[out,retval] ILValue** pValue);
```

描述：

查询指定次序的数值点信息.

参数	说明
iIndex	数值点序号
pValue	返回指定的数值点对象，数值点不存在返回 NULL

返回值：

返回值	说明
S_OK	查询成功
其它	查询失败

12.3 ILValueList::NewValue

```
HRESULT NewValue(
[out,retval] ILValue** pValue);
```

描述：

新建新的数值点对象.

参数	说明
pValue	返回创建成功的数值点信息对象，创建成败则返回 NULL

返回值：

返回值	说明
S_OK	创建成功
其它	创建失败

12.4 ILValueList::UpdateToServer

```
HRESULT UpdateToServer();
```

描述：

把数值列表中的所有数值向服务器递交.

参数	说明

返回值：

返回值	说明
S_OK	递交成功
其它	递交失败

12.5 ILValueList::UpdateToServer

```
HRESULT InsertToServer();
```

描述：

把数值列表中的所有历史数值插入到服务器.

参数	说明

返回值：

返回值	说明
S_OK	插入成功
其它	插入失败