Mars 数据库接口开发指南

V0.1

2022年2月9日

Mars 数据库提供了2种类型的接口用于对外数据的交换，“驱动接口、消费接口”或者叫做“下接口、上接口”。驱动接口主要面向数据的录入，消费接口面向数据的查询和检索。驱动接口提供了数据高速、高效的写入功能，消费接口提供了查询的高效以及接口类型的多样性等。驱动接口面向数据从硬件设备或软件的采集、消费接口面向前端UI界面以及数据分析软件等。

## 驱动接口

Mars 通过支持不同类型的数据采集驱动，实现和各种硬件设备或软件的系统交换，从而实现数据的录入。缺省Mars提供3种驱动，Sim、Spider、DirectAccess 。遵循驱动开发规范，用户可以开发出自己的驱动，在Mars 默认提供的其他驱动满足不了要求的时候。

### Sim 驱动。

Sim　驱动是一个数据模拟驱动，内置了Sin、Cos、Step(步进)等数据模拟函数。要使用Sim驱动模拟数据，则只需在添加数据变量时，在变量的驱动的属性种指定Sim，同时寄存器种填写Sin等模拟函数即可。

### **Spider 驱动。**

Spider 是一个独立的数据采集平台，目前支持常用的各种通信协议。Spider 除了支持各种硬件、软件设备的采集之外还支持多个Spider设备间的级联、数据的断点缓冲续传、以及自定义C#脚本嵌入等功能。

Spider驱动用于和Spider之间进行通信。Spider 和Mars 数据库间是多进行的，之间通过网络进行通信。Mars数据库和Spider间是一对多的关系，解决超大规模的数据采集问题。

用户可以有两种方式通过Spider扩展自己的功能，1：遵循Spider内部的不同协议设备开发规范，开发出适应自己的设备协议(具体参考Spider协议驱动开发规范)。2：通过Spider 驱动 客户端 SDK API，开发自己的采集程序。

* Spider 驱动客户端API开发规范。

1. SDK DLL名称：SpiderDriver.ClientApi.dll 采用.net standard 2.1 开发。首先在你的工程中引用上述DLL。
2. 通过构建类SpiderDriver.ClientApi.DriverProxy 对象的实例，实现和Mars 驱动进行通信。
3. SpiderDriver.ClientApi.DriverProxy 接口介绍。
4. Open(string serverip, int port) 使用API的第一步是使用该方法连接远程服务。Open 之后，在网络断线之后，内部自动提供了断线重连机制，外部不需要再次处理。可以通过该对象的IsConnected 属性判断网络是否建立连接。在系统退出时，调用Close主动通知服务，断开连接。
5. Login(string username, string password, int timeount = 5000) 在网络建立连接后，紧接着需要登录。这里用户名、密码时数据库开发时指定的账户信息。在网络通信断了，再次连接成功之后，需要再次登录。退出登录：Logout();
6. 初始化相关函数。

QueryAllTagIdAndNames(int timeount = 5000)：查询所有的驱动配置成Spider 的变量的Id和名称以及类型。

QueryTagId(IEnumerable<string> tags,int timeout = 5000)：查询一组变量的ID值。

GetDriverRecordTypeTagIds(int timeount = 5000)：查询所有记录类型为驱动的变量的ID集合。历史记录类型为驱动的变量，支持历史数据断线缓冲续传的功能，在网络通信正常时，需要将断线期间的历史数据上传进去。具体参考历史数据上传接口。

CheckRecordTypeByTagId(IEnumerable<int> ids, int timeout = 5000)：检查变量的历史记录类型是否为驱动记录类型。

1. 设置变量的实时值。

SetTagValueAndQuality(List<RealTagValue> values, int timeout = 5000) 设置一组变量的实时值和质量戳。

SetTagValueAndQuality2(List<RealTagValue2> ids)设置一组变量的实时值和质量戳。所有XXX2格式的函数在功能上和XXX函数是一样的，区别在于XXX2函数多适用于通过网闸的单向数据传输，这种情况在使用时不需要先登录，而是每次数据请求时把账户信息带上。

SetTagValueAndQualityAsync(List<RealTagValue> values, int timeout = 5000) 设置一组变量的实时值和质量戳。XXXAsync 是一种异步调用的方式，功能上和XXX一样，区别在于XXXAsync不会等到服务器反馈，就会返回。

SetTagValueAndQuality(RealDataBuffer data, int timeout = 5000)设置一组变量的实时值和质量戳。RealDataBuffer 类型的data的填充，可以通过AppendValue 函数填充数据。

SetTagRealAndHisValue(List<RealTagValue> ids, int timeout = 5000)设置一组变量的实时值和对于历史记录类型为值改变的变量同时写入一个历史值进入缓存。其他函数重载形式，参考SetTagValueAndQuality。

AppendRegistorDataChangedCallBack(IEnumerable<int> ids,int timeout=5000) 追加注册哪些变量的值改变时，通知到客户端。这里的值改变是指通过消费接口设置变量的值。（多用于用户通过界面进行参数设置，需要将这些参数写入到设备或者第三方软件等场景。）

UnRegistorDataChangedCallBack(IEnumerable<int> ids, int timeout = 5000) 取消注册变量。

ResetRegistorDataChangedCallBack(int timeout = 5000) 取消所有注册。

1. 设置变量的历史值。

只有变量的历史记录类型为驱动记录的时候，才支持通过API写入历史数据。定时、值改变等形式，历史数据由数据库自身定时自动生成，生成的历史记录不可修改。

SetTagHisValue(int id, TagType type, IEnumerable<TagValue> values, int timeout = 5000) 设置一个变量的一组历史值。

SetTagHisValue(int id, TagType type, HisDataBuffer data, int timeout = 5000)设置一个变量的一组历史值。Data 通过AppendValue(datetime,value,quality) 函数设置一组历史值。

所有XXX2、XXXSync格式的重载函数，同设置实时数据部分一致。

SetMutiTagHisValue(Dictionary<int,TagValueAndType> idvalues,int timeout=5000) 一次设置一组变量的历史值。 idvalues 是变量ID和值得集合。

bool SetMutiTagHisValue(HisDataBuffer data, int timeout = 5000) 一次设置一组变量的历史值。Data 通过AppendValue(id,datetime,value,quality) 函数设置一组历史值。

1. 数据库回写回调。

ProcessDataPushDelegate ValueChanged { get; set; } 当驱动收到数据库往驱动回写数据时，调用此回调。使用示例：

driverProxy.ValueChanged = new SpiderDriver.ClientApi.DriverProxy.ProcessDataPushDelegate((values) =>

{

foreach (var vv in values)

{

Debug.Print("收到数据 ID:"+ vv.Key + ",值：" + vv.Value.ToString());

}

});

DatabaseChangedDelegate DatabaseChanged { get; set; } 数据库发生变动后，调用此回调。通常响应此回调，用于重新初始化工作。使用示例：

driverProxy.DatabaseChanged = new DatabaseChangedDelegate((real, his) => {

if(real) Debug.Print("变量实时配置发生变化");

if(his) Debug.Print("变量历史配置发生变化");

});

### **DirectAccess 驱动。**

DirectAccess 驱动是一种用于直接访问数据库的驱动，不像Spider那样独立（建立自己的变量，然后通过关联建立和数据库变量间的联系），他直接使用数据里寄存处的配置和外部直接进行交互。

支持两种API：.net DirectAccess Client API 和Grpc Client API。

* .net DirectAccess Client API 是一种底层的、高效的访问接口。
* Grpc Client API 是通过GRPC 协议进行通信的接口，Java 程序可以通过该接口和Mars数据库进行通信。GRPC Client API 通过一个独立的 exe程序(DirectAccessGrpc) 程序实现GRPC 协议和DirectAccess Client API 的转接，从而实现通过GRPC 协议和Mars数据库的通讯。

#### .net DirectAccess Client API

* 客户端工程引用DirectAccessDriver.ClientApi.dll
* 通过DirectAccessDriver.ClientApi.Proxy 和DirectAccess 驱动服务进行通信。
* API 接口介绍。DirectAccess Proxy类的功能包含了Spider Proxy 接口的所有功能，具体参考Spider API 章节。这里列出新增部分。
* string Login2(string username, string password, int timeount = 5000) 用于客户端对外再次当做服务器对外提供服务时，验证客户端发过来的登录信息。
* void Hart() 心跳函数。当长时间网络没有数据通信时，这个时候网络中断的时候，会出现程序无法通知网络断开的情况。这里通过定时调用该函数，解决这个问题。
* Dictionary<int, Tuple<string, byte>> QueryTagIdAndNamesbyFilter(string filter, int timeout = 5000) 查询变量的寄存器中包含指定关键字的变量。当自己扩展DirectAccess 驱动的时候（例如实现一个OPC UA Client采集器时，可以在寄存器中扩展包含OPCUA关键字，OPC UA Client采集器可以查询包含OPCUAC关键字的寄存器对应的变量，从而实现针对性的数据采集）。

1. **Grpc Client API**

* 使能GrpcClientAPI需要启动协议转换程序DirectAccessGrpc.exe
* .net 客户端可以直接引用DirectAccessGrpc.Client.dll;非.net 客户端可以通过Data.proto 配置文件生成相应的代理对象。
* .net 客户端代理对象DirectAccess.Client 介绍。
* bool Login(string username,string password) 用户登录。每次登录有效期10分钟，在此期间如果没有任何数据通信则会超时，超时以后需要重新登录，超时前可以通过心跳维持。
* Logout() 用户登出。
* Hart() 心跳，维持登录。
* UpdateTagValue(Dictionary<string,Tuple<object,byte>> values) 更新变量的实时值。
* UpdateTagHisValue(Dictionary<string,List<Tuple<DateTime,object,byte>>> values) 更新变量的历史值。

### **自定义驱动开发。**

驱动开发首先引用Cdy.Tag.Driver.dll ,需要实现IProducterDriver 接口，然后在驱动配置文件(Config/Driver.cfg)中添加配置。

1. **IProducterDriver 接口介绍。**

* 属性

/// <summary>

/// 名称,唯一不可冲突

/// </summary>

string Name { get; }

驱动名称，在变量开发环境中显示的名称，要保证驱动名称的唯一性。

/// <summary>

/// 支持的寄存器集合

/// </summary>

string[] Registors { get; }

支持的寄存器名称的集合。

* 方法

开发时接口方法

/// <summary>

/// 获取某个数据库参数配置

/// </summary>

/// <param name="database">数据库名称</param>

/// <returns></returns>

Dictionary<string,string> GetConfig(string database);

/// <summary>

/// 更新针对某个数据库的配置

/// </summary>

/// <param name="database">数据库名称</param>

/// <param name="config">配置</param>

void UpdateConfig(string database, Dictionary<string, string> config);

上述方法主要用于针对该驱动的配置，例如对于支持网络服务的驱动， 这里进行服务端口等参数。

运行时接口方法

/// <summary>

/// 初始化

/// </summary>

/// <returns></returns>

bool Init();

数据库在加载时，会调用所有驱动的初始化函数。可以在这里进行初始化功能。

/// <summary>

/// 开始

/// </summary>

/// <returns></returns>

bool Start(IRealTagProduct tagQuery,ITagHisValueProduct tagHisValueService);

数据库启动时，会调用此函数启动驱动。通过IRealTagProduct 接口，可以对数据库变量进行读取写入、查询符合此驱动的变量、订购数据库变量反写事件等。ITagHisValueProduct 用于对数据库变量的历史数据进行写入，通常用于历史记录类型为驱动的情况下，对数据库变量的历史记录进行更新。

/// <summary>

/// 停止

/// </summary>

/// <returns></returns>

bool Stop();

/// <summary>

/// 暂停

/// </summary>

/// <returns></returns>

bool Pause();

暂停和恢复主要在数据库热重启时，会先让驱动暂停，等数据更新完成后再调用恢复。

/// <summary>

/// 恢复

/// </summary>

/// <returns></returns>

bool Resume();

1. **IRealTagProduct接口。**

* 通过变量的Id设置变量的实时值 SetTagValue
* 获取变量的值 GetTagValueForProductor
* 通过寄存器地址获取变量的Id GetTagIdsByLinkAddress
* 通过寄存器地址获取变量的实例 GetTagByLinkAddress
* 订购数据库反写事件 SubscribeValueChangedForProducter
* 取消订购数据库反写 UnSubscribeValueChangedForProducter
* 提交数据变动SubmiteNotifyChanged ，在批量写入一组数据后，最后调用此方法，通知数据写入完成。

1. **ITagHisValueProduct 接口**

* 获取变量历史记录类型为驱动的变量的集合，GetManualRecordTagId
* 根据变量的Id获取记录类型，GetTagRecordType
* 设置变量的历史值，SetTagHisValue

## 消费接口

消费接口面向前端UI、数据分析等软件的一类接口。这类接口的特定是：高效的数据实时、历史数据查询、大量客户端并发访问、安全验证、接口协议多样性等。目前提供了Web API、GRPC API、HighSpeed　API 三种类型的接口，用户也可以遵循接口开发规范，开发自己的接口。

消费接口具体协议软件实现（XXXProxy）和数据库核心功能（InDBRun）间通过多进程方式实现，两者之间通过网络进行通信。这么设计的优点有：

1. 减少接口和核心间的耦合，核心做到很精简，达到较高的性能
2. 接口实现和核心可以分别部署不同的机器上，或在同一个机器上。
3. 接口实现和核心间是多对一的关系，即使同一协议的接口也可以有多个实例，从而应对大量客户端高频次访问带来的瓶颈问题。

### Web API

通过WebAPI 访问数据库数据，可以实现实时数据的查询和修改以及历史数据的查询。实时数据的修改，主要用于上层应用对参数的修改，他的特性的是实时性比较高，能够及时的将数据下发的硬件设备，批量数据效率低。

启用WebAPI接口，只需要运行DbWebApi.exe即可。该Exe和数据库可以分散在不同的机器上。该Exe通过读取配置文件(Config/DbWebApi.cfg)，获取数据的网络配置信息以及登录到数据库使用的账户信息。配置文件如下：

<Config ServerPort="14331">

<ProxyClient Ip="127.0.0.1" Port="14330" LoginUser="Admin" LoginPassword="Admin" WorkMode="0" PollCircle="1000" IsUseStandardHisDataServer="false" UseHttps="false"/>

</Config>

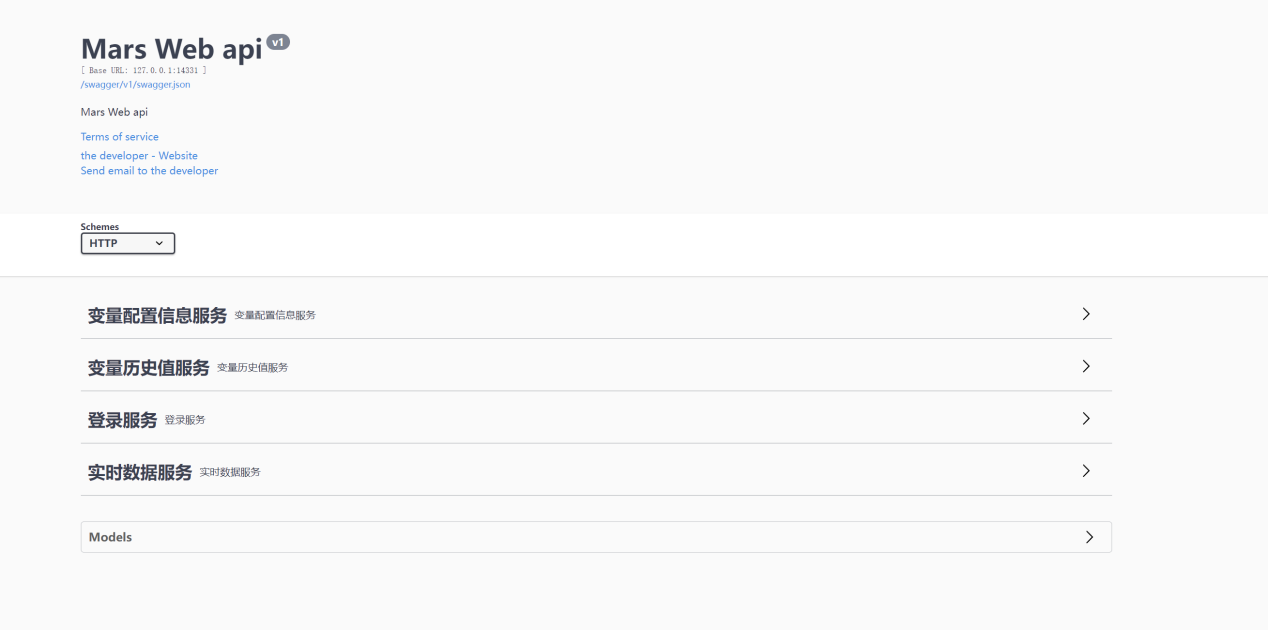
ServerPort：WebAPI对外服务的端口

ProxyClient中WorkMode指定接口和数据库间通信的方式；0：定时查询，PollCircle:指定查询周期，单位毫秒。1:数据库改变通知。

UseHttps：WebAPI对外Web服务是否使用Https 协议。

IsUseStandardHisDataServer：查询历史数据是否从一个独立的历史数据服务进行查询，这里暂时无用，始终设置为:False 即可。

在接口运行起来之后，通过服务的网址http://127.0.0.1:14331/ 直接查看接口帮助信息。如下图:



整个WebAPI提供了4大类服务。

1. 登录服务

/Login/TryLogin ：Post 用户登录，登录成功以后返回一个Token，有效期10分钟。

/Login/Logout ：Post 注销账户信息。

/Login/Hart ：Post 心跳，维持Token不超时。

1. 变量配置服务

/TagInfo : Get 根据变量名称，获取变量的配置信息

/TagInfo/GetByGroup: Get根据变量组获取变量的配置信息

/TagInfo/ListTagGroup:Get 枚举变量组

1. 实时数据服务

/RealData: Get 获取变量的实时值(值、时间、质量戳)

/RealData: Post 修改变量的值

/RealData/Value: Get 获取变量的实时值（只包括值）

/RealData/ValueAndQuality: Get 获取变量的实时值（只包括值、质量戳）

1. 历史数据服务



### **GRPC API**

通过GRPC 协议访问数据库数据，实现实时数据的查询和修改以及历史数据的查询。实时数据的修改，主要用于上层应用对参数的修改，他的特性的是实时性比较高，能够及时的将数据下发的硬件设备，批量数据效率低。

相较于WebAPI 接口，GRPC具有较高的效率以及较小的数据传输量。要想使用GRPC接口，需要运行DBGrpcApi.exe进程。该Exe和数据库可以分散在不同的机器上。该Exe通过读取配置文件(Config/DBGrpcApi.cfg)，获取数据的网络配置信息以及登录到数据库使用的账户信息。具体配置文件内容，同WebAPI。

c#客户端需要引用DBGrpcApi.Client.dll ,通过DBGrpcAPI.Client 类实现对数据库的访问。接口使用说明如下：

1. 构建Client 对象，new Client(string ip,int port);

登录相关

1. 登录 Login(string username,string password)，每次登录有效期10分钟，在此期间如果没有任何数据通信则会超时，超时以后需要重新登录，超时前可以通过心跳维持。
2. 心跳 Hart();
3. 退出登录 Logout();

变量信息相关

1. List<Tagbase> GetTagByNameRequest(IEnumerable<string> tagnames) 根据变量的名称获取变量的配置。
2. List<Tagbase> GetTagByGroup(string group) 获取某个变量组下的所有变量
3. List<string> ListTagGroups() 枚举变量组

实时数据相关

1. Dictionary<string,object> ReadRealValue(List<string> tags) 通过变量的名称读取变量的实时值。
2. Dictionary<int, object> ReadRealValueById(List<int> tagIds,string group) 通过变量的Id获取变量的实时值。
3. Dictionary<string, Tuple<int, object>> ReadRealValueAndQuality(List<string> tags) 通过变量的名称获取变量实时值和质量戳。
4. Dictionary<int, Tuple<int, object>> ReadRealValueAndQualityById(List<int> tagIds, string group)通过变量的Id获取变量实时值和质量戳。
5. Dictionary<string, Tuple<int,DateTime, object>> ReadRealValueAndQualityTime(List<string> tags) 通过变量的名称获取变量实时值、质量戳和变量更新的时间。
6. Dictionary<int, Tuple<int, DateTime, object>> ReadRealValueAndQualityTimeById(List<int> tagIds, string group) 通过变量的Id获取变量实时值、质量戳和变量更新的时间。
7. bool SetTagValue(string tag,object value) 设置变量的值。
8. bool SetTagValue(Dictionary<string,string> values) 一次设置一组变量的值。

历史数据相关

1. Dictionary<string, HisDataValueCollection> ReadAllHisValue(List<string> tags, DateTime starttime, DateTime endTime) 读取变量某个时间段内的所有值。所有值是指某个时间内，数据库真实记录的真实的值。与之相反的是，查询历史数据是会做数据拟合出一个值：例如8:00:00记录了一个值，8:00:02记录了一个值，如果查询8:00:01时的值，则会通过数据拟合出该时刻的值。而查询所有值的时候，则会返回8:00:00和8:00:02两个值。
2. Dictionary<string, HisDataValueCollection> ReadHisValue(List<string> tags,DateTime starttime,DateTime endTime,TimeSpan duration,Cdy.Tag.QueryValueMatchType type) 通过数据拟合的方式查询一组时刻的历史值。Cdy.Tag.QueryValueMatchType 数据拟合类型，取值有： Previous(取前一个值),After(取后一个值),Closed(取较近的值),Linear(线性插值)

历史数据查询统计相关

1. Dictionary<string,StatisticsValueCollection > ReadStatisticsValue(List<string> tags,DateTime starttime,DateTime endtime) 查询某个时间内，变量的统计值。在数据库运行过程中，会以小时为单位对数值型变量，进行最大值、最小值、平均值统计。
2. Tuple<DateTime, object> FindTagValue(string tag, DateTime startTime, DateTime endTime, NumberStatisticsType type, object value, double interval=0) 在某个时间段内，查找数值型变量的值满足查询条件的第一个值的时间和值。Interval 是偏差，当查询条件等于的时候，由于数值精度的问题可能当时的值无法精确等于，这里就引入一个偏差。NumberStatisticsType 比较类型，支持： GreatValue(大于某个值的值),LowValue(小于某个值的值), EqualsValue(是否有等于某个值的值)
3. Tuple<DateTime, object> FindTagValues(string tag, DateTime startTime, DateTime endTime, NumberStatisticsType type, object value, double interval=0) 在某个时间段内，查找数值型变量的值满足查询条件的所有值和对应时间的集合。
4. double? CalTagValueKeepTime(string tag, DateTime startTime, DateTime endTime, NumberStatisticsType type, object value, object interval) 统计某个时间段内某个变量的值满足指定条件的时间。
5. Tuple<double,List<DateTime>> FindNumberTagMaxValue(string tag,DateTime startTime,DateTime endTime) 查找某个时间段的某个数值型变量的值得最大值。
6. Tuple<double, List<DateTime>> FindNumberTagMinValue(string tag, DateTime startTime, DateTime endTime) 查询某个时间段内某个数值型变量的值的最小值。
7. double FindNumberTagAvgValue(string tag, DateTime startTime, DateTime endTime) 计算某个时间段内某个数值型变量的值的平均值。

### **HighSpeed API**

对于数据通信要求效率比较高的情况，可以使用该接口。他直接使用底层Socket进行通信，提高了数据的吞吐量。

要想使用该接口，需要运行DBHighApi.exe进程。该Exe和数据库可以分散在不同的机器上。该Exe通过读取配置文件(Config/DBHighApi.cfg)，获取数据的网络配置信息以及登录到数据库使用的账户信息。具体配置文件内容，同WebAPI。

c#客户端需要引用DBHighApi.Client.dll ,通过DBHighApi.ApiClient 类实现对数据库的访问。接口使用说明如下：

登录相关

1. bool Open(string serverip, int port) 和数据库建立连接。
2. bool Login(string username,string password,int timeount= 30000) 登录。
3. void Hart() 心跳。当长时间网络没有数据通信时，这个时候网络中断 的时候，会出现程序无法通知网络断开的情况。这里通过定时调用 该函数，解决这个问题。

配置信息相关

1. string GetRunnerDatabase(int timeout = 5000) 获取正在运行的数据库名称
2. Dictionary<string, int> ListAllTag(int timeout = 30000) 获取所有的变量和ID 的集合。
3. List<string> ListALlTagGroup(int timeout = 30000) 获取所有变量组。
4. Dictionary<string,int> GetTagIds( List<string> tagNames, int timeout = 5000) 根据变量名称获取变量的ID。
5. Dictionary<string, int> ListTagByGroup(string group,int timeout = 30000) 根据变量组获取变量。

实时值相关

1. bool RegistorTagValueCallBack(int minid, int maxid, int timeout = 5000) 追加注册需要订购值改变主动通知变量。当数据库的值发生变量时，主动通知客户端。
2. bool RegistorTagValueCallBack(List<int> ids, int timeout = 5000) 追加注册需要订购值改变主动通知变量。
3. bool ClearRegistorTagValueCallBack(int timeout = 5000) 清空所有订购。
4. Dictionary<int, Tuple<object, DateTime, byte>> GetRealData(List<int> ids,bool nocache=false, int timeout = 5000) 获取变量的实时值、质量、最后更新时间。
5. Dictionary<int, object> GetRealDataValueOnly(List<int> ids, bool nocache = false, int timeout = 5000) 获取变量的实时值。
6. Dictionary<int, Tuple<object,byte>> GetRealDataValueAndQualityOnly(List<int> ids,bool nocache=false, int timeout = 5000) 获取变量的实时值、质量戳。
7. bool SetTagValue(int id,byte valueType,object value,int timeout=5000) 设置单个变量的值。
8. bool SetTagValue(List<int> id, List<byte> valueType, List<object> value, int timeout = 5000) 一次设置一组变量的值。

历史值相关

1. HisQueryResult<T> QueryAllHisValue(int id,DateTime startTime,DateTime endTime,int timeout=5000) 查询某个变量的某个时间段内的记录的所有值。根据变量的类型T的取值有：bool,byte,short,ushort,int,uint,long,ulong,string,datetimg,float,double,IntPointData,UIntPointData,IntPoint3Data,UIntPoint3Data,LongPointData,ULongPointData,LongPoint3Data,ULongPoint3Data 。根据HisQueryResult的Count属性，获取值得个数，然后通过函数：T GetValue(int index,out DateTime time,out byte qulity) 读取具体的值。
2. HisQueryResult<T> QueryHisValueAtTimes<T>(int id, List<DateTime> times, Cdy.Tag.QueryValueMatchType matchType, int timeout = 5000) 查询在指定时刻上变量的值。
3. HisQueryResult<T> QueryHisValueForTimeSpan<T>(int id,DateTime startTime,DateTime endTime,TimeSpan span,QueryValueMatchType matchType, int timeout = 5000) 查询在指定时刻上变量的值，通过指定时间间隔确定时间点。
4. NumberStatisticsQueryResult QueryStatisticsValue(int id, DateTime startTime, DateTime endTime, int timeout = 5000) 查询统计值。在数据库运行过程中，会以小时为单位对数值型变量，进行最大值、最小值、平均值统计。NumberStatisticsQueryResult QueryStatisticsAtTimes(int id, List<DateTime> times, int timeout = 5000) 查询指定时刻上的统计值。
5. Tuple<DateTime, object> FindTagValue(string tag, DateTime startTime, DateTime endTime, NumberStatisticsType type, object value, double interval=0) 在某个时间段内，查找数值型变量的值满足查询条件的第一个值的时间和值。Interval 是偏差，当查询条件等于的时候，由于数值精度的问题可能当时的值无法精确等于，这里就引入一个偏差。NumberStatisticsType 比较类型，支持： GreatValue(大于某个值的值),LowValue(小于某个值的值), EqualsValue(是否有等于某个值的值)
6. Tuple<DateTime, object> FindTagValues(string tag, DateTime startTime, DateTime endTime, NumberStatisticsType type, object value, double interval=0) 在某个时间段内，查找数值型变量的值满足查询条件的所有值和对应时间的集合。
7. double? CalTagValueKeepTime(string tag, DateTime startTime, DateTime endTime, NumberStatisticsType type, object value, object interval) 统计某个时间段内某个变量的值满足指定条件的时间。
8. Tuple<double,List<DateTime>> FindNumberTagMaxValue(string tag,DateTime startTime,DateTime endTime) 查找某个时间段的某个数值型变量的值得最大值。
9. Tuple<double, List<DateTime>> FindNumberTagMinValue(string tag, DateTime startTime, DateTime endTime) 查询某个时间段内某个数值型变量的值的最小值。
10. double FindNumberTagAvgValue(string tag, DateTime startTime, DateTime endTime) 计算某个时间段内某个数值型变量的值的平均值