## 测点变量

测点变量（全局变量）在整个工程中全局有效，其本质是功能块类型的变量。测点变量在AT测点变量视图中进行编辑，同时记录该操作，在AT保存时，测点变量的操作记录，利用共享内存，发送给总控，实现数据的一致。

AT中对测点变量的编辑，包括导入测点、增加测点（通过右键增加、通过模块增加、通过变量自动声明增加）、修改测点（修改点名，修改非点名项）、插入测点、删除测点、转移测点以及剪切、复制、粘贴、撤销、恢复等辅助功能。

1. 图 ‑1：测点功能图

### 1.1.X总控交互

测点变量为保证变量名称全局唯一，在增加（插入）测点、修改测点名称、删除测点、（AT关闭）和转移测点时要与总控通讯，得到总控正确的应答以后，才能执行相应的操作。而对测点变量的所有操作，会在保存时将所有操作一次发到总控，使总控中的数据与AT中的数据保持一致。框架是测点操作的向总控发送消息的中继。

对测点变量进行操作时，测点向框架发送的消息类型有：

#define VAR\_ADD\_PIN 1 增加测点

#define VAR\_MODIFY\_PIN 2 修改测点名

#define VAR\_DEL\_PIN 4 删除测点

#define VAR\_ATCLOSE\_PIN 7 AT关闭时与MACS通讯，通知总控清除没用点名

对测点变量进行操作时，框架向总控发送的消息类型有：

#define ID\_MODULECONFIG 61 保存工程时给工程总控发送硬件操作的消息

#define ID\_DATABASECONFIG 62 通知总控接收数据

#define ID\_NOTIFYNAME 63 通知总控名称变化

对测点变量进行操作时，总控通讯返回的应答值为：

#define PN\_RENAME 2 //点名已存在

#define PN\_VALID 1 //点名有效

#define MACS\_CLOSR -1 //总控未启动

#define SERVER\_MACS\_CLOSR 3 //服务器端总控未启动

下图是与总控交互的流程：



1. 图 ‑00：与总控交互流程
2. 在CAppGlobalFunc::GetStation()->m\_dbContainer.m\_pchData中写入测点操作的相关信息。
3. 向框架发送相应测点操作类型的消息

int iResult = ::SendMessage(CAppGlobalFunc::GetMainFrmHwnd(),

WM\_VAR\_ASK\_FRAME,VAR\_ADD\_PIN,0);

1. 框架收到消息，调用库函数 将m\_pchData所指的内容写入到共享内存中，g\_pMsgQueue->sendMsg(m\_pchData)。
2. 向总控发送消息

iResult = ::SendMessage((HWND)g\_lHMIhWnd,g\_uiRemoteMessage,

ID\_NOTIFYNAME,iSN);

其中：g\_uiRemoteMessage用于与总控通讯的系统注册消息，iSN为站号；

1. 总控收到消息，根据操作类型执行相应的操作，之后将操作得到的数据写入共享内存，并向框架返回应答值iResult。
2. 框架得到总控的应答信息iResult，若iResult为-1表示总控未启动。
3. 若iResult有效，则将读取共享内存的数据

g\_pMsgQueue->recieveMsg(pStation->m\_dbContainer.m\_pchData,100,300);

1. 测点得到总控通讯的返回值iResult，若iResult无效，显示相应的错误类型；若iResult有效，读取总控应答的信息。

### 1.1.Y 装载配置文件

配置文件决定测点变量的类型以相关的约束条件。通过下列方式打开工程，都会加载测点配置文件：新建工程、工程打开、打开最近工程列表中的工程、打开指定目录的文件。调用的总接口是：CAutoThinkApp::LoadFilesOfPinVar()。具体调用些列接口装载配置文件：



图 ‑00：实现过程

#### 1.1.Y.1加载db.dat文件

测点类型文件db.dat定义测点类型、每种类型各点项的默认值及点项之间的约束关系。其类型和点项对应的数据结构是CDBClassSet、DB\_ClassItem，约束关系对应的数据结构为DB\_Limited、DB\_ComLimited DB\_Limited3。

1）db.dat的文件格式



图1 db.dat文件格式

其中，测点类型的结构如下图所示





图2 类型n数据

其中，属性（点项）结构如下图所示



图3 点项n数据

2）用类CDBClassSet处理测点类型信息，用类DB\_ClassItem处理点项信息。解析中用到的数据结构具体定义如下：

* 类型中各属性（点项）类型为DB\_ClassItem，该类型的数据结构为：

typedef struct{

int ver;

int index;//对象创建时的序号，以后不可改变。此序号用来排序

string name;

string des;

ItemDataType dataType;//数据类型（编译前）

int offset;//偏移（编译前）

int size;//大小指个数总的size＝size\*sizeof(datatype)（编译前）

ItemEditType editType;//编辑类型

string fontname;//字体

int fontsize;//字体大小

COLORREF bkClr;//背景色

COLORREF txClr;//字颜色

char initvalue[64];//初始值

stringList comboxStr;//在组合框中的字符

BOOL bVisible;//是否可视

BOOL bDownLoad;//是否下装

BOOL bOnLineChanged;//是否在线改变

BOOL bComm;//是否上传；

//编译后的信息

long cmpdt;//编译后数据类型

long cmpdtl;//编译后数据长度

long cmpoffset;//编译后偏移

long cmpsuboffset;//编译后位偏移

long cmpsublength;//编译后位长度

BOOL bIO;//是否IO项目（ver>=2时有效）

long nHisFlag;//历史库标志，0：不进，1：模拟，2：开关（ver>=2时有效）

long bEnableEmpty;//为初值 表示不允许为空，0表示允许空，1表示不允许空（ver>=3时有效）

long nDBType;//匹配的点项类型 编译后数据类型（ver>=3时有效）

//编辑信息

UINT flag;//点被改变 （ver>=4时有效）

BOOL bFoot; //是否功能块引脚（ver>=4时有效）

}DB\_ClassItem;

* 类型中定义的上下限约束类型为DB\_Limited，该类型的数据结构为：

Typedef struct{

int ver;//结构版本

string colname;属性名

string minv;最小值

string maxv;最大值

string szErrMsg;

}DB\_Limited; //上下限约束

* 类型中定义的关系约束类型为DB\_RLimited，该类型的数据结构为：

typedef struct{

int ver;//结构版本

string rcolname;//右属性名

long lSign; //符号 0>,1=,2<;3>=;4<=;5!=;

string rvalue;右属性值

string colname;//属性名

string minv;//最小值

string maxv;//最大值

string szErrMsg;

}DB\_RLimited;//关系约束（当前db.dat中无此约束）

* 类型中定义的范围约束类型为DB\_RangeLimited，该类型的数据结构为：

typedef struct{

int ver;//结构版本

string rcolname;

long lSign;//符号 0>,1=,2<;3>=;4<=;5!=;

string rvalue;

string colname;

stringList rangeLimited;

string szErrMsg;

}DB\_RangeLimited;//范围约束（当前db.dat中无此约束）

* 类型中定义的组合唯一约束类型为DB\_ComLimited，该类型的数据结构为：

typedef struct{

int ver;//结构版本

string name;

stringList rangeLimited;

string szErrMsg;

}DB\_ComLimited;//组合唯一约束（SN+DN+CN唯一）

* 类型中定义的Limited3约束类型为DB\_Limited3，该类型的数据结构为：

typedef struct{

int ver;

string scolname;

string svalue;

string dcolname1;

int iSign;//符号0>,1=,2<;3>=;4<=;5!=;

string dcolname2;

string szErrMsg;

}DB\_Limited3;（MD<LL<AL<AH<HH<MU等）

3） 对配置文件中类型库的管理是由类CDBClassManage完成的，其加载配置文件的接口是：

* void CDBClassManage::Load()

功能说明： 装载配置文件中类型库记录

参 数： 无



图 ‑0：装载配置文件流程图

* void CDBClassSet::Load(ifstream &fs)

功能说明：装载配置文件中的CDBClassSet记录

参数说明：fs 与db.dat关联的输入流

函数流程图如下图所示。



图 ‑0：读取点项流程图

* void operator >>(ifstream &fs,DB\_ClassItem &value)

功能说明：运算符>>重载，用于读取点项的数据信息。

参数说明：fs 与db.dat关联的输入流；

value 结构体DB\_ClassItem的对象。

附加说明: 该函数是全局函数。

程序流程图（省略）。

4）增加测点全局变量组，将测点类型写入对应的数据容器。主要接口有:

* void CAutoThinkApp::CreateNewPinGlobalGroup()

功能说明：增加测点全局变量组

参数说明：空

函数流程图如下图所示：



图 ‑0：增加全局变量组流程图

#### 1.1.Y.2加载其他文件

配置文件wg.ini、Transfer.ini、ModuleChannel.in和UT.ini定义了点项其他相关信息。下面给出个配置文件的作用以对应的数据类型：wg.ini定义了各种硬件模块对应的WG（增益）、TP（信号类型）、MU、MD之间的关系，其数据类型是CDP\_WG\_Table和CDP\_WG；Transfer.ini定义可相互转移的测点类型，其数据类型是CTransferTable；ModuleChannel.ini定义各模块不同通道的默认信号类型，其数据类型是CModuleChannelTPTable和CModuleChannelTPSet；UT.ini定义量纲选项，其数据类型是CUTManage。

加载配置文件的总接口是： void CAppGlobalFunc::VarLoadIniFile()。通过下列接口完成加载工作。



下面wg.ini为例，说明加载配置文件格式和程序流程，其他配置文件加载过程类似，故不再赘述。



图 ‑0：wg.ini文件格式

* void CHWTP::Load()

功能说明：加载配置文件wg.ini

参数说明：空

函数说明：实际加载wg.ini文件是由接口CDP\_WG\_Table::Load()完成的。

* void CDP\_WG\_Table::Load()

功能说明：加载配置文件wg.ini

参数说明：空

函数流程图如下图所示：



图 ‑0：wg.ini读取流程图

* void CDP\_WG::Load()

功能说明：读取模块增益信息

参数说明：空

附加说明：结构体DP\_WG用以记录模块增益、信号类型、量程上和量程下等信息，其数据结构如下：

typedef struct{

int wg;//增益

CString emType;//信号类型

double mu,md; //量程上、量程下

}DP\_WG;

函数流程图如下图所示：



图 ‑0：读取模块增益信息流程图

### 1.1.1导入测点

导入变量是AT与总控之间数据交互的一部分，完成的功能是：按照约定的数据格式解析变量的各个属性（例如：变量名、变量说明等），并按照解析出来的属性生成新的全局变量。数据库总控将每个站中的测点变量保存到相应的文件中，文件格式如下：

1. 文件格式：

表 3‑1：文件格式

|  |  |  |
| --- | --- | --- |
| 名称 | 类型 | 大小（byte） |
| 类型个数 | char | 1 |
| 类型1信息 |  |  |
| 类型2信息 |  |  |
| …… |  |  |
| 类型n信息 |  |  |

表 3‑2：类型信息格式

|  |  |  |
| --- | --- | --- |
| 名称 | 类型 | 大小（byte） |
| 类型名称 |  | 32 |
| 变量个数 | int | 4 |
| 变量1信息 |  |  |
| 变量2信息 |  |  |
| …… |  |  |
| 变量n信息 |  |  |

表 3‑3：变量信息格式

|  |  |  |
| --- | --- | --- |
| 名称 | 类型 | 大小（byte） |
| 变量名长度 | char | 1 |
| 变量名 | char[] | 变量名长度 |
| 成员个数 | int | 4 |
| 成员1信息 |  |  |
| 成员2信息 |  |  |
| …… |  |  |
| 成员n信息 |  |  |

表 3‑4：成员信息格式

|  |  |  |
| --- | --- | --- |
| 名称 | 类型 | 大小（byte） |
| 成员名长度 | char | 1 |
| 成员名 | char[] | 成员名长度 |
| 成员值长度 | char | 1 |
| 成员值 | char[] | 成员值长度 |

1. 2）文件名：站号.var

3）其他的约定

* 每个类型的变量导入同一个变量组，组名即为类型名。
* PN（点名）即为变量名，因此，成员中无需再使用PN。
* 导入某一类型变量时，该类型一定存在，否则该组变量无法导入。
* 当导入过程中某一个变量或某一组变量出现异常时，导入继续进行，顺序解析下一个或下一组变量。
* 由于导入的文件为总控编译通过的文件，因此，除重名检查外，无需对标识符进行其他检查。

主要接口函数有：

* void CDBContainer::LoadControlVar(char \*pcPath)

参数说明： pcPath 文件路径+文件名

功能说明：导入测点变量总入口

返 回 值：无

函数流程图如下图所示。

图 ‑2：导入测点变量总入口

* void CDBContainer::LoadTypeInfo(CFile& VarFile, bool bControl)

参数说明：VarFile 文件指针

bControl 是否导入测点变量

true：导入测点变量

false：导入内部变量组

功能说明：导入类型信息

返 回 值：无

说 明：根据需求的变化，要求导入一组内部变量"ENG\_CAL"，此功能的实现融合到导入测点变量的函数中，以参数bControl区分，具体见函数流程图。

函数流程图如下图所示。

图 ‑3：导入类型信息

* void CDBContainer::LoadVarInfo(CFile& VarFile, CGlobalVarContainer\* &pGlobalVarContainer, CFunctionBlockType \*pFBType)

参数说明：VarFile 文件对象

pGlobalVarContainer 待添加的全局变量容器

pFBType 类型指针

功能说明：导入变量信息

返 回 值：无

函数流程图如下图所示。





图 ‑4：导入变量信息

* void CDBContainer::LoadMemberInfo(CFile& VarFile,CFunctionBlockDB\* pFB,bool bIstheATvar,CBaseDB\* pATBase,CGlobalVarContainer\* pContainer)

参数说明：VarFile 文件对象

pFB 新创建的变量指针

bIstheATvar 是否与AT中的测点变量重名，true：重名，false：不重名

pATBase 重名时AT中的变量指针

pContainer 待添加的变量容器指针

功能说明：导入变量信息

返 回 值：无

说 明：由于测点变量的结构和AT中的CBase结构不一致，因此在CfunctionBlockDB中建立结构m\_NonPinPropertyMap和m\_NonPinPropertyArray存储非功能块引脚。变量名、变量别名、变量说明属于非功能块引脚，但是其他模块读取以上信息又从CBase结构中读取，因此KKS、DS要在两处赋值，即变量本身和非功能块引脚表。

函数流程图如下图所示。



图 ‑5：导入成员信息

### 增加测点

新增测点变量“是否可强制”、“掉电保护”、“HMI”属性默认为“TRUE”，不可见， “是否上网”属性默认为“FALSE”，可见。

增加的测点记录于CGlobalVarContainer中。每次增加测点时都要记录相应的操作，保存时，通知总控增加相同的数据。增加测点的方式包括：在测点变量视图中增加测点、通过增加硬件模块增加测点、通过自动声明增加测点。

#### 在测点变量视图中增加测点

在测点变量视图中增加测点的方法是：点击鼠标右键，弹出右键菜单，选择“增加变量”。不能通过增加硬件模块增加的测点都可以通过此方法增加。

实现流程如下：



1. 图 ‑6：测点变量视图中增加测点实现流程

右键增加测点响应接口为：

* void CPinGlobalVarView::OnAddVar()

功能：在测点变量视图中增加测点变量

参数：无

返回值：无

说明：选择“增加变量”时，响应此接口，通过下图所示的调用操作完成测点增加

1. 图 ‑7：实现过程

被调用接口功能如下：

* CString CAppGlobalFunc::GetNewPNFromMACS(CString strType)

功能：向总控请求新的点名

参数：strType：新增点的类型

返回值：新的点名

说明：测点名必须保证在所有控制站间唯一，由于AT只能判断本控制站中的点名唯一性，所以新增测点必须由总控给定一个全局唯一的名字。点名唯一性处理见（）

* bool CGlobalVarContainer::AddPinRecord(CFunctionBlockType \*pFBType,CString strNewName)

功能：在测点变量容器中增加新的测点记录

参数：pFBType：新的测点类型，strNewName：新的测点名

返回值：true：增加成功，false：增加失败

说明：此接口内先根据类型和测点名构造一个新的测点变量，该变量各引脚默认值在db.dat配置文件中定义，然后将该变量增加到容器的末端，具体流程如下图所示：



1. 图 ‑8：在测点变量容器中增加新的测点记录流程

实现过程如下：

图 ‑9：实现过程

* CFunctionBlockDB::CFunctionBlockDB(CFunctionBlockType \*pType,CString strName)

功能：根据类型和点名构造新的测点变量

参数： pType：被构造变量的类型， strName：新的变量名

返回值：无

说明： 根据类型和点名构造新的测点，设置测点属性值，然后根据db.dat文件定义该类型中各点项的默认值为新的测点变量各引脚赋值（db.dat文件结构见附件）。具体流程如下图所示：



1. 图 ‑10：构造测点变量流程

* void CFunctionBlockDB::SetPinValueWithDB()

功能说明：用配置文件的数据为功能块类型引脚重新设值

参 数：空

附加说明：功能块类型引脚即为在库中可以看到的引脚。以DVI 类型为例，通过下列方式查看：库管理器－>IO.HLF－>DI－>DVI－>右击鼠标，弹出菜单，选择“查看“ －>功能块变量区说明。



图 ‑10：功能块类型引脚重新设值流程

* void CGlobalVarContainer::AddVar(CBaseDB \*pVar)

功能：在变量容器中追加新的变量

参数： pVar：指向新的变量

返回值：无

* bool CDBClassInstSet::AddInst(CGridItemSet \*pGridItemSet,CFunctionBlockDB \*pFBDB)

功能：根据当前的显示方式和新增加的测点变量构造新的测点实例，供显示时使用

参数：pGridItemSet：当前各点项的显示方式，pFBDB：新的测点变量

返回值：true：增加成功，false：增加失败

说明：由于测点变量显示的点项较多，用户为了突出希望看到的点，可能会用颜色、单元格大小等来对该点项进行标注，测点实例就是为了记录用户的标注而设计的。现在AT没有这方面的要求，测点实例就成了多余的、而在现在代码中又必须被维护的结构，为了不产生新的问题，测点实例暂时还不会被去掉，希望以后能去掉该结构。

* CAppGlobalFunc::RecordAddTestPinOper(CFunctionBlockDB \*pFBDB)

功能：记录增加操作

参数： pFBDB：新的测点变量

返回值：无

说明：根据已定义的数据格式，记录增加测点操作，保存时通知总控增加该测点。记录增加测点操作数据结构见表（）

* void CPinGlobalVarView::FillToLine(int iRow, CDBClassInst \*pInst,CHWTPSet \*pTPSet/\* = NULL\*/)

功能：在测点变量视图中增加一行新的测点

参数：iRow：新增加的行号，pInst：新的测点实例

返回值：无

说明：先在测点变量视图中的grid控件上新增一行，然后用pInst填充这一行

* CAppGlobalFunc::SetModifiedProjectFlag(true)：设置工程修改标识
* CAppGlobalFunc::SetCompileProjectFlag(AT\_PRJ\_UNCOMPILED)：设置工程编译标识
* ::SendMessage(frmwnd,WM\_VAR\_NOTIFY\_IEC,0,0)：通知程序区更新
* CAppGlobalFunc::SetPinVarChange(true)：设置测点变化标识

#### 通过增加硬件模块增加测点

工程中DVI，AVI，AVO，DVO，REALIN，PUI，SOE\_DH，BITIN，REALOUT，BITOUT，WORDOUT，AVI6，WORDIN，LONGIN，LONGOUT，BYTEIN，BYTEOUT等类型的测点只能通过增加硬件模块增加。DP增加硬件模块时，变量定义要做两部分工作：增加测点、增加模块对应的诊断变量。硬件配置记录通道信息，并由框架通知变量增加对应的测点。

通过增加硬件模块增加测点的方法是：硬件配置->在设备库选择机柜->在机柜面板“未配置”区域鼠标右击->模块添加->选择需要的模块类型。

通过增加硬件模块批量增加测点的方法是：硬件配置->在设备库选择机柜->在机柜面板“未配置”区域之外鼠标右击->在弹出菜单中选择批量增加->在机柜面板“未配置”区域鼠标右击->弹出批量增加对话框，填写批量增加的模块数。

1. 测点相关的枚举及结构体定义

硬件模块操作的动作类型如下：

typedef enum emActionType

{

HW\_ADD\_SLAVE = 1, //添加模块

HW\_DEL\_SLAVE, //删除模块

HW\_MODIFY\_STATIONNUM, //更改模块站号

HW\_CUT\_SLAVE, //剪切模块

HW\_PASTE\_SLAVE, //粘贴模块

HW\_UNDO\_SLAVE, //撤销

HW\_REDO\_SLAVE //恢复

};

下列结构体tagHWChannelInfoPerSlave和tagHWChannelPointInfo用于硬件配置与变量定义的交互，记录需生成变量的各模块的各通道信息。

typedef struct tagHWChannelInfoPerSlave

{

char strSlaveName[32]; //从站名

BYTE byOldSlaveAddr; //修改前的设备地址

BYTE byNewSlaveAddr; //修改后的设备地址

CList<CString,CString&>\* pChPointTypeList; //此模块内的测点类型名列表, CMap<CString,LPCTSTR,unsigned short,unsigned short>\* pChannelNumMap;

//各类型通道个数，关键字为通道对应的测点类型

CMap<CString,LPCTSTR,unsigned short,unsigned short>\* pStartChannelNoMap;

//各类型通道的起始通道号，关键字为通道对应的测点类型

}HWChannelInfoPerSlave;

typedef struct tagHWChannelPointInfo

{

unsigned short usSlaveNum; //从站个数

BYTE byActionType; //动作类型

HWChannelInfoPerSlave\* pChannelInfoPerSlave;//各从站中通道的信息

}HWChannelPointInfo;

1. 模块类型信息，每种模块类型包含多种模块。下表举例说明了每种类型的模块信。

表3-0 模块类型信息

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **模块类型** | **模块个数** | **名称（举例）** | **通道个数** | **类型个数** | **类型名称** |
| AI | 13 | SM410 | 8 | 1 | AVI( CT修改需设置交叉引用表，MT项 ) |
| AO | 5 | SM510.. | 8 | 1 | AVO |
| DI | 2 | SM610.. | 16 | 1 | DVI |
| PI | 3 | SM620.. | 8 | 1 | PUI( TP值为：累计\频率型) |
| DO | 2 | SM710.. | 8 | 1 | DVO |
| SOE | 2 | SM611.. | 16 | 1 | SOE\_DH |
| TURBIN SPEED IO | 4 | SM630.. | 8 | 5 | DVI; DVO; REALIN; REALOUT; WORDOUT |
| TURBIN VALUE CONTROL | 2 | SM460.. | 8 | 4 | BITIN; BITOUT; REALIN; REALOUT |
| 其他 | 2 | SM020-MASTER | 0 | 0 | 0 |

1. 通道信息如下：

通道号、通道名称、通道类型、通道说明、初始值、通道地址和是否上网等属性。其中：

名称：命名规则为DPIO+站号+设备地址+通道号。

通道地址：直接地址形式，例如：

SM410 ：%IB0 输入区地址0，一个字节。

SM610 ：%IX17.0 输入区地址17，第0位。

通道类型： 为WORD类型等。

1. 框架中定义的消息响应接口为：

char CMainFrame::OnMessageHWChannelInfo(WPARAM wParam, LPARAM lParam)

DP根据该接口的返回值判断测点是否增加成功，返回值0：测点增加成功，返回值i（表示第i个模块）：第i个模块测点增加失败，如果返回值为i，DP不增加第i个模块。

增加硬件模块增加测点流程如下图所示：



1. 图 ‑11：增加硬件模块增加测点流程

实现过程如下图所示：



1. 图 ‑12：实现过程

为实现增加测点操作，该接口首先解析增加硬件模块时记录的数据包，判断是否为增加操作，然后调用按调用顺序）：

* char CGlobalVarContainer::HWAddPinRecord(CString strModelName,CString strDNValue, CString strCNValue)

功能：在容器中增加一个测点变量记录

参数：strModelName：模块名，strDNValue：新增加模块的设备地址，strCNValue：新增测点对应的通道号

返回值：0：不能正常与总控通讯，3：不能与服务器端总控通讯，-1：增加失败

说明：硬件模块的每一个通道都对应一个测点变量，该接口增加一个通道对应的的测点变量

一个通道的测点变量增加流程如下图所示：



1. 图 ‑13：增加一个通道对应的测点变量流程



图 ‑13：增加一个通道对应的测点变量流程（新）

* void CGlobalVarContainer::AddDevVar(CString strDNValue)

功能：增加诊断变量

参数： strDNValue：新增加模块的设备地址

返回值：无

说明：每增加一个硬件模块都对应增加一个诊断变量，诊断变量的命名规则为“FIO”+站号+‘\_’+设备地址，变量类型为HSGetState，诊断变量属于内部变量，构造的诊断变量记录于CGlobalVarContainer 中m\_globalVarMapm\_globalVarList，并记录于

CDBContainer中m\_globalAllVarMap保持全局唯一。增加诊断变量流程如下图所示：



1. 图 ‑14：增加诊断变量流程

#### 通过变量自动声明增加测点

通过变量自动声明增加测点的方法是（以增加DMI类型测点变量为例）：在库管理器中视图中，选择库“I/O.HLF”，点击“DM”文件夹，选中测点类型“DMI”，将该模块拖至IEC视图区。

变量自动声明增加测点的接口是CGlobalFunc::AutoVarDeclare，测点增加的详细程序流程参照本文档3.6节“变量自动声明相关内容，在此不赘述。

### 删除测点

不同类型的测点删除方式有所不同，通过硬件模块增加的测点，都必须通过删除对应模块来删除，其它类型的测点可以直接在测点变量视图中来删除（选中被删除的测点，通过右键菜单中“删除变量”或“Delete”快捷键来删除）。每次删除测点时都要记录相应的操作，保存时，通知总控删除对应的数据。

#### 在测点变量视图中删除测点

在测点变量视图中删除测点的方法是：选中被删除的测点变量，点击鼠标右键，弹出右键菜单，选择“删除变量”，或“Delete”快捷键。

删除流程如下图所示：



1. 图 ‑15：在测点变量视图中删除测点流程

通过这种方式删除测点响应接口为：

* void CPinGlobalVarView::OnDelVar()

功能：在测点变量视图中删除测点变量

参数：无

返回值：无

说明：选择“删除变量”或按“Delete”快捷键时，响应此接口

实现过程如下：



1. 图 ‑16：实现过程

被调用接口功能如下：

* CAppGlobalFunc::GetOnlineProjectFlag()

功能：判断过程是否是在线状态，如果是在线状态则不允许删除变量

* bool CAppGlobalFunc::IsHWModuleChannel(CString strType)

功能：判断当前操作的测点是否是通过模块增加的，如果是则不允许删除

参数：strType：被删除测点的类型

返回值：true：通过模块增加的，false：不是通过模块增加的

说明：DVI,AVI,AVO,DVO,REALIN,PUI,SOE\_DH,BITIN,REALOUT,BITOUT,WORDOUT,AVI6,WORDIN,LONGIN,LONGOUT,BYTEIN,BYTEOUT等类型的点只能通过模块操作，其他类型的点可以直接在视图中编辑。

* bool CDBClassInstSet::DelInsts(CStringArray \*pstrInsts)

功能：在测点实例记录中删除一组测点实例

参数：pstrInsts：被删除测点的名字

返回值：true：删除成功，false：删除失败

说明：测点实例是当初设计留下的问题，以后维护时去掉该结构

* bool CGlobalVarContainer::DelPinInsts(CStringArray \*pstrInsts)

功能：在测点容器中删除一组测点变量

参数：pstrInsts：被删除测点的名字

返回值：true：删除成功，false：删除失败

说明：删除测点容器中的一组测点数据，记录删除操作

* Bool CGlobalVarContainer::DelNormalRecord(CString pinName,CObject \*pOperSub/\* = NULL\*/)

功能：删除普通全局变量

参数：pinName：被删除测点的名字；pOperSub：记录删除操作

返回值：true：删除成功，false：删除失败

说明：删除普通全局变量，需要在全局变量数据容器m\_globalVarList和m\_globalVarMap中删除记录，并要在m\_globalAllVarMap和m\_globalAllVarRefMap中删除对应的记录。



图 ‑00：删除普通全局变量流程图

* void DeleteRow(CGridCtrl \*pGrid,CArray<int,int> \*pRowArr)

功能：在测点变量视图的gridctrl中删除对应的测点记录

参数：pGrid：gridctrl空间指针，pRowArr：被删除的行号

返回值：void

说明：删除被选中的行，重新调整各行序号

* void CAppGlobalFunc::TellMACSDelPName(CString strPinName)
* 功能：通知总控删除点名记录

参数：strPinName：被删除的测点名

返回值：void

* bool CCrossTable::SetPOUFlag(CString strVarName)

功能：测点变量被删除后，则引用该变量的所有POU的编译和修改标志都要改变

* void CAppGlobalFunc::SetModifiedProjectFlag(true)

功能：测点被删除后，设置工程修改标识

* void CAppGlobalFunc::SetCompileProjectFlag(AT\_PRJ\_UNCOMPILED)

功能：测点被删除后，设置工程编译标识

* ::SendMessage(frmwnd,WM\_VAR\_NOTIFY\_IEC,0,0)

功能：测点被删除后，通知IEC视图刷新

* void CAppGlobalFunc::CloseDetailDialog(CStringArray &strArrWName)

功能：关闭strArrWName指定的详细对话框

参数：strArrWName：被删除测点的名字记录

返回值：void

说明：如果被删除的测点有详细对话框被打开，则关闭该对话框

* CAppGlobalFunc::SetPinVarChange(true)

功能：测点被删除后，设置测点修改标识

#### 通过删除硬件模块删除测点

工程中DVI，AVI，AVO，DVO，REALIN，PUI，SOE\_DH，BITIN，REALOUT，BITOUT，WORDOUT，AVI6，WORDIN，LONGIN，LONGOUT，BYTEIN，BYTEOUT等类型的测点只能通过删除硬件模块删除。DP删除硬件模块时，变量定义要做两部分工作：删除对应测点、删除模块对应的诊断变量。硬件配置记录操作信息，并由框架通知变量完成相应操作。

通过增加硬件模块删除测点的方法是：硬件配置->在设备库选择机柜->在机柜面板选中欲删除的模块，鼠标右击->在弹出对话框中选择“删除”。

通过增加硬件模块批量删除测点的方法是：硬件配置->在设备库选择机柜->在机柜面板，按shift建，选中欲删除的模块->按下delete键。

框架中定义的消息响应接口为：

char CMainFrame::OnMessageHWChannelInfo(WPARAM wParam, LPARAM lParam)

依据硬件模块动作类型HW\_DEL\_SLAVE，执行相应的删除操作。 DP根据该接口的返回值判断测点是否删除成功，返回值0：测点删除成功，返回值i（表示第i个模块）：第i个模块测点删除失败，如果返回值为i，DP不删除第i个模块。

删除硬件模块测点流程如下图所示：



1. 图 ‑17：增加删除模块删除测点流程

在得到删除模块个数usSlaveNum，第i个模块对应的测点类型个数usChannelTypeCount及设备地址（strSlaveAddr）后，再通过如下图所示的过程删除第j个类型（strChannelTypeName）的测点变量组中设备地址为strSlaveAddr的点：

图 ‑18：实现过程

被调用接口功能如下：

* bool CGlobalVarContainer::HWDelPinInst(CString strDNValue).

功能：删除第j个测点类型（strChannelTypeName）对应的测点变量组中设备地址为strDNValue的测点变量

参数：strDNValue：设备地址

返回值：true：删除成功，false：删除失败

说明：每个被删除的硬件模块都对应一个设备地址，通过设备地址删除该模块对应的一组测点变量；删除测点时，若发生类型转移，则需要根据Transfer.ini文件查找转移到的类型，删除转移类型中对应的测点变量，其对应的数据结构为：Transfer\_Type\_Pair转移的类型对(目前只有AVI和AVI6可以互相转化)。

根据设备地址删除容器中测点变量的流程如下图所示：



1. 图 ‑19：根据设备地址删除容器中测点变量流程

* void CGlobalVarContainer::DelDevVar(CString strDNValue)

功能：根据设备地址删除对应的诊断变量

参数：strDNValue：设备地址

返回值：无

说明：从诊断变量容器中删除对应的诊断变量

m\_globalVarMap m\_globalVarList m\_globalAllVarMap

通过删除硬件模块删除测点的流程如下图所示：



1. 图 ‑20：通过删除硬件模块删除测点流程

### 插入测点

工程中非DVI，AVI，AVO，DVO，REALIN，PUI，SOE\_DH，BITIN，REALOUT，BITOUT，WORDOUT，AVI6，WORDIN，LONGIN，LONGOUT，BYTEIN，BYTEOUT等类型的测点都可以进行插入测点操作。

插入测点变量可以看作是特殊的增加测点操作，只不过是在特殊的位置添加。

只能在测点变量视图中插入测点，方法：选中测点变量，点击鼠标右键弹出右键菜单，选择“插入变量”，即可在被选中的测点前面插入一行新的测点变量。

插入测点变量流程如下图所示：



1. 图 ‑21：插入测点变量流程

插入测点响应接口为：

* void CPinGlobalVarView::OnInsertVar()

功能：在测点变量视图中插入测点变量

参数：无

返回值：无

说明：选择“插入变量”时，响应此接口，通过下图所示的调用操作完成测点增加



1. 图 ‑22：实现过程

各接口功能如下：

* void CPinGlobalVarView::GetVarNameByRow(int iRow, CGridCtrl \*pGrid, CString &strKey)

功能：根据选中的行号得到被选中测点的名字

* CString CAppGlobalFunc::GetNewPNFromMACS(CString strType)

功能：向总控请求新的点名，方法见3.1.8

* bool CGlobalVarContainer::InsertPinRecord(CString strName,CFunctionBlockType \*pFBType,CString strBeforePN,int &iIndex)

功能：在测点变量组中插入新的测点变量

参数： strName:新的点名，pFBType：测点类型，strBeforePN：在strBeforePN指定的测点前面插入新测点，iIndex：新测点在变量组中的索引值（返回使用）

返回值：true：插入成功，false：插入失败

说明：实现流程如下图所示：



1. 图 ‑23：在测点变量组中插入新的测点变量流程

* bool CDBClassInstSet::InsertInst(CGridItemSet \*pGridItemSet,CFunctionBlockDB \*pFBDB,int iRow,CDBClassInst \*&pInst)

功能：在测点实例组中构造并插入新的测点实例

参数： pGridItemSet:当前显示项记录，pFBDB：新的测点变量，iRow：插入位置，pInst：新插入的测点实例（返回使用）

返回值：true：插入成功，false：插入失败

说明：测点实例是遗留问题，以后去掉

* void CAppGlobalFunc::RecordInsertTestPinOper(CFunctionBlockDB \*pFBDB, CString strNextPN)

功能：记录插入测点操作

参数： pFBDB：新插入的测点变量，strNextPN：在strNextPN指定的测点之前插入新的测点

返回值：void

说明：记录数据格式见3.1.7

### A 剪切、复制和粘贴

工程中非DVI，AVI，AVO，DVO，REALIN，PUI，SOE\_DH，BITIN，REALOUT，BITOUT，WORDOUT，AVI6，WORDIN，LONGIN，LONGOUT，BYTEIN，BYTEOUT等类型的测点都可以进行剪切、复制和粘贴测点操作，可以站内和站间进行粘贴。

#### 1.1.A.1 剪切测点变量

可在测点变量视图中剪切测点，方法：选中测点变量，点击鼠标右键弹出右键菜单，选择“剪切”，即可在被选中的测点前面插入一行新的测点变量。剪切的本质是删除操作，将选中的变量复制到剪切板上。

删除流程如下图所示：



图 ‑00：在测点变量视图中剪切测点流程

剪切测点实现过程为：

 图 ‑00：实现过程

被调用的主要接口如下：

* void CGlobalVarContainer::CutRecord(CObList \*pCutList,CStringArray \*pstrNextPNArr)

功能：在测点变量视图中插入测点变量

参数：pCutList被剪切的变量记录，pstrNextPNArr 被剪切的变量下一变量的记录

返回值：无

说明： 剪切变量的本质操作是删除操作，本接口的操作流先按照图 ‑00：删除普通全局变量流程图。

* void CPinGlobalVarDoc::SetPinInsts()

功能：更新测点实例

参数：无

返回值：无

#### 1.1.A.2 复制测点变量

可在测点变量视图中复制测点，方法：选中测点变量，点击鼠标右键弹出右键菜单，选择“复制”，即可在被选中的测点变量复制到剪切板上。

粘贴测点实现过程为：



图 ‑00：实现过程

复制测点变量的操作较为简单，可以参照剪切测点变量程序流程，故在此不赘述。

#### 1.1.A.3 粘贴测点变量

可在测点变量视图中粘贴测点，方法：将选中测点变量放置于剪切板中，点击鼠标右键弹出右键菜单，选择“粘贴”，即可在被选中的测点最后加入一行新的测点变量。剪切的本质是增加操作，将剪切板中的变量加入到数据容器中，并更新相应的试图和标志。

删除流程如下图所示：



图 ‑00：在测点变量视图中剪切测点流程

粘贴测点实现过程为：



图 ‑00：实现过程

被调用的主要接口如下：

* bool CGlobalVarContainer::PasteRecord(CObList \*pPasteList

,CList<CbaseDB,CBaseDB\*> \*pDBList)

功能：粘贴变量。

参数：pPasteList 被粘贴的变量指针，

pDBList 输出被粘贴的变量记录（点名有可能被改变）

返回值：true 粘贴成功，false 粘贴失败。

说明：实现流程如下图所示：



图 ‑00：在测点变量视图中粘贴测点流程

### 修改测点

测点变量各点项的有效性判断是在配置文件（db.dat）中定义的，该文件中定义的点项之间的约束关系如下：

* SN+DN+CN 唯一
* AREANO（区号）取值范围[0,31]
* 报警限制：MD<AL3<LL<AL<AH<HH<AH3<MU
* TP（信号类型）限制：不同模块对应的TP值范围不同，由ModuleChannel.ini配置文件决定
* OT（输出上限）> OB（输出下限）
* PU（PV的量程上限）> PD（PV的量程下限）
* FN（故障信息）取值范围[0,255]
* CE（指令信息）取值范围[0,255]
* SE（状态信息）取值范围[0,255]
* ZT（运行状态）取值范围[0,255]
* SU（给定上限幅） > SD（给定下限幅）

测点变量视图中并不是所有的项都是可以编辑的，如SN（站号）、DN（设备地址）、CN（通道号）、MT（模块类型）等项不可编辑，DN项只能通过修改硬件模块的设备地址来修改对应测点变量的DN项值，其它可编辑项可以在测点变量视图中或在该变量的详细对话框中修改。测点变量的修改可分为在测点变量视图中修改、在详细对话框中修改、通过修改硬件模块设备地址（DN）修改。

#### 在测点变量视图中修改

测点变量可编辑的项都可以在测点变量视图中修改，修改流程如下图所示：



1. 图 ‑24：在测点变量视图中修改流程

修改过程中主要调用的函数如下（按调用顺序）：

* void CPinGlobalVarView::OnGridStartEdit(NMHDR \*pNotifyStruct, LRESULT\* pResult)

功能：修改前保存旧值

参数： 参见MSDN

返回值：无

说明：在修改点项之前保存修改前的值，用于修改失败后恢复旧值

* void CPinGlobalVarView::OnGridEndEdit(NMHDR \*pNotifyStruct, LRESULT\* pResult)

功能：修改项值，检查新值是否有效

参数：参见MSDN

返回值：无

说明：修改项值后响应此接口，该接口首先根据被修改变量的行号得到被修改变量的名字，然后调用下面的接口完成修改的有效性判断 。其实现流程如下图所示：



1. 图 ‑25：修改流程

实现过程如下图所示：



1. 图 ‑26：实现过程

各接口功能如下：

* Bool CDBClassInstSet::EditItem(CfunctionBlockDB \*pFBDB,CString strType ,CString strOldPN, CString strItem, CString &strNewValue)

功能：修改测点实例属性值

参数：pFBDB：被修改的测点变量；

strType：类型名

strOldPN：被修改测点变量的PN

strItem：被修改的属性名

strNewValue：修改值

返回值：true：修改成功 false：修改失败

说明：首先检查新值是否有效，如果有效则修改测点实例中该点项的值。检查新值有效性的有下列接口具体完成：

* bool CDBClassInst::CheckNewValueValidity(CString strItemName, CString &strNewValue)

功能：根据配置文件中定义的约束判断项值是否有效

参数： strItemName被修改的点项名，strNewValue：修改值

返回值：true：修改成功，false：修改失败

说明：检查新值是否有效，进行值与类型是否匹配、限制、约束等检查。

具体流程如下图所示：

图 ‑27：有效值检查流程

* bool CGlobalVarContainer::EditPinItem( CString strPN, CString strItem,CString strValue)

功能：修改测点变量点项的值

参数： strPN被修改测点的名字，strItem：被修改的点项名，strValue：修改值

返回值：true：修改成功，false：修改失败

说明：修改测点变量的点项值。



图3-00 修改测点变量属性值流程

* void CFunctionBlockDB::ModifyTestPinItemOper(CString strPN, CString strItem, CString strValue)

功能：记录修改操作

参数： strPN：被修改测点的名字，strItem：被修改的点项名，strValue：修改值

返回值：无

说明：记录修改操作，保存时用于与总控通讯

#### 在详细对话框中修改

双击测点变量，打开其详细对话框，在此对话框中可以修改点项（可被编辑的项）的值。在详细对话框中修改点项的操作同在测点变量视图中修改。通过详细对话框可以修改变量的别名和初始值，其中点项DN、SN、CN、MT和WG属性不可修改。

在详细对话框中修改测点流程如下图所示：



1. 图 ‑28：在详细对话框中修改测点流程

修改过程主要调用的函数如下（按调用顺序）：

* void CDlgVarDetail::OnGridStartEdit(NMHDR \*pNotifyStruct, LRESULT\* pResult)

功能：修改前保存旧值

参数： 参见MSDN

返回值：无

说明：在修改点项之前保存修改前的值，用于修改失败后恢复旧值

* void CDlgVarDetail::OnGridEndEdit(NMHDR \*pNotifyStruct, LRESULT\* pResult)

功能：修改注释/初始值后判断是否正确， 并改变相应引脚的值。

参数：参见MSDN

返回值：无

说明：修改项值后响应此接口，调用下面的接口完成修改的有效性判断。

* bool CBaseDB::EditComment(CString strNewValue)

功能：修改变量说明，判断新值的有效性。

参数： strNewValue新的注释

返回值：true：修改初始值成功 false：修改失败

* bool CBaseDB::EditInitValue(CString strNewValue)

功能：修改变量初始值，判断新值的有效性。

参数： strNewValue新的初始值

返回值：true：修改初始值成功 false：修改失败

* bool CFunctionBlockDB::CheckTestPinValue(CString strPInName, CString &strNewValue)

功能：检查新值的有效性

参数： strPInName：被修改的点项名，strNewValue：修改值

返回值：true：修改成功，false：修改失败

说明：修改项未在配置文件中定义约束，则判断新值是否和类型匹配；修改项在配置文件中定义了约束，则根据约束检查新值的有效性。其实现流程同图 3-27。

* void CPinGlobalVarView::FillToLine(int iRow, CDBClassInst \*pInst,CHWTPSet \*pTPSet/\* = NULL\*/)

功能：重新填写测点视图中对应的变量各点项值

参数： iRow：行号 CDBClassInst：测点实例 pTPSet：信号类型集合（与模块对应）返回值：无

说明：在详细对话框中对点项修改后，需要通知测点变量视图同步更新显示的值。

初始值：WM\_VAR\_UPDATE\_TESTPINVIEW

SetModifiedProjectFlag SetCompileProjectFlag

#### 通过修改硬件模块设备地址修改

DVI，AVI，AVO，DVO，REALIN，PUI，SOE\_DH，BITIN，REALOUT，BITOUT，WORDOUT，AVI6，WORDIN，LONGIN，LONGOUT，BYTEIN，BYTEOUT等类型的测点DN项是在视图中时不可编辑的，在详细对话框中也不允许修改，但是修改模块设备地址时会修改该测点DN项的值。DP修改硬件模块设备地址时，变量定义要做两部分工作：修改对应测点DN项的值、修改该设备地址对应的诊断变量。修改设备地址时，硬件配置记录相应的数据，并由框架通知变量进行修改操作。

在修改硬件模块设备地址的方法是：在主机柜非“未配置”区，鼠标右击，在弹出菜单中选中“地址更新”，在“地址更新”对话框中，按照要求选择“背板序号”和“拔码号”（其中拔码号是4的整数倍，并且是未在面板中显示的0到255之间的整数，拔码号是修改地址的起始地址）。

框架中定义的消息响应接口为：

char CMainFrame::OnMessageHWChannelInfo(WPARAM wParam, LPARAM lParam)

修改流程如下图所示：



1. 图 ‑29：修改流程

在得到删除模块个数usSlaveNum，第i个模块对应的测点类型个数usChannelTypeCount及旧设备地址（strOldSlaveAddr）、新设备地址（strNewSlaveAddr）后，再通过如下图所示的过程修改第j个类型（strChannelTypeName）对应的测点变量组中设备地址为strOldSlaveAddr的点，实现过程如下：



1. 图 ‑30：实现过程

各接口功能如下：

* bool CGlobalVarContainer::HWEditDN(CString strOldDNValue, CString strNewDNValue)

功能：修改第j个测点类型（strChannelTypeName）对应的测点变量组中设备地址为strDNValue的测点变量

参数：strOldDNValue：被修改的设备地址，strNewDNValue：新的设备地址

返回值：true：修改成功，false：修改失败

说明：在第j个测点类型对应的测点变量组中中找设备地址为strOldDNValue的测点，修改其DN项值为strNewDNValue。在修改设备地址时要注意两个问题：1.是否发生类转移。2. 诊断变量的维护。

CtransferTable

修改变量组中测点变量的设备地址流程如下图所示：



1. 图 ‑31：修改容器中测点变量设备地址流程

* bool CGlobalVarContainer::DNValueExist(CString strDNValue)

功能：判断该容器中是否包含指定设备地址对应的侧点变量

参数：strDNValue：待查找的设备地址

返回值：true：存在 false：不存在

* void CGlobalVarContainer::DelDevVar(CString strDNValue)

功能：删除旧设备地址对应的诊断变量

参数：strDNValue：旧的设备地址

返回值：无

说明：修改设备地址时，先删除旧设备地址对应的诊断变量，再利用新的设备地址生成新的诊断变量

* void CGlobalVarContainer::AddDevVar(CString strDNValue)

功能：根据新的设备地址增加新的诊断变量

参数：strDNValue：新的设备地址

返回值：无

说明：修改设备地址时，先删除旧设备地址对应的诊断变量，再利用新的设备地址生成新的诊断变量

### 转移测点

工程中一些不同类型的测点是可以相互转移的，如AVI中的测点可以全部或部分转移到AVI6中，反之亦可以，可以发生转移的测点类型定义在配置文件Transfer.ini中，文件结构见附录

实现测点类型转移的方法：在测点视图中，选中待转移的测点，点击鼠标右键，弹出右键菜单，选择“转移”，即可将选中的测点转移到对应的类型中。

实现流程如下图所示：



1. 图 ‑32：转移测点实现流程

转移响应接口函数为：

* void CPinGlobalVarView::OnTransfer()

功能：转移测点变量

参数：无

返回值：无

说明：选择“转移”时，响应此接口，实现过程如下图所示：



1. 图 ‑33：实现过程

其接口功能如下：

* bool CPinGlobalVarDoc::TransferInsts(CStringArray \*pstrInsts)

功能：AVI和AVI6之间的相互转移

参数：pstrInsts：记录被转移测点变量的名字记录

返回值：true：转移成功，false：转移失败

* bool CDBClassInstSet::DelInsts(CStringArray \*pstrInsts)

功能：删除 pstrInsts指定的测点实例

参数：pstrInsts：记录被转移测点变量的名字记录

返回值：true：转移成功，false：转移失败

说明：为了维护方便，测点实例有机会要去掉

* bool CGlobalVarContainer::TransferPinInsts(CStringArray \*pstrInsts,CList<CBaseDB\*,CBaseDB\*> \*pToFBList =NULL)

功能：在测点变量容器中实现测点的转移

参数：pstrInsts：记录被转移测点变量的名字，pToFBList：转移后新类型的测点变量记录（返回使用）

返回值：true：转移成功，false：转移失败

说明：首先根据当前的测点类型找到转移到的类型，然后根据测点名收集被转移的测点变量，修改该变量的类型、offset等属性，将其添加到转移到的类型容器中，并在当前容器中删除记录。具体流程如下：





图 ‑34：在变量容器中实现测点的转移流程

* void CFunctionBlockDB::TestPinAssignValue(CFunctionBlockDB \*pOldFBDB)

功能：测点变量赋值

参数： pOldFBDB：被赋值的测点变量

返回值：无

* void CGlobalVarContainer::RemoveHWUndoRedoRecord(CString strDN, CString strCN

功能：通过模块删除测点变量时，删除撤销、恢复记录中对应的修改操作

参数： strDN：被删除模块的设备地址 strCN：被删除的通道地址

返回值：无

### 保存测点

为了保证AT与总控中的测点数据一致，AT中对测点增加、修改、删除、转移等操作时都会记录相应的操作数据于操作链表中，保存时，将这些操作数据放到一块共享内存，通知总控取数据、解析完成相应的操作。

#### 保存过程

保存测点时调用的接口为：

* void CAutoThinkApp::SendVarMessage()

功能：将AT中对测点的操作数据放到共享内存，通知总控读数据

参数： 无

返回值：无

说明：保存时调用此接口完成与总控的通讯。

实现流程如下图所示：



1. 图 ‑35：保存测点流程

#### 保存数据格式

由AT端对某个站的测点进行编辑，保存时将所有操作一次发到总控，使总控中的数据与AT中的数据保持一致。消息总结构如下：

4个字节的操作个数+每一个操作的数据

AT保存时向总控发送数据，数据格式如下：

表 3‑1：向总控发送数据格式

|  |  |  |
| --- | --- | --- |
| 增加测点数据格式 | | |
| 标记 | 1Byte | 值：1 |
| 点名 | 32Byte | 值：点名 |
| 点类型 | 32Byte | 值：类型名 |
| 站号 | 1Byte | 值：站号 |
| 修改点名数据格式 | | |
| 标记 | 1Byte | 值：2 |
| 原点名 | 32Byte | 值：原点名 |
| 新点名 | 32Byte | 值：新点名 |
| 修改点项（非点名项）数据格式 | | |
| 标记 | 1Byte | 值：3 |
| 点名 | 32Byte | 值：点名 |
| 项名 | 16Byte | 值：项名 |
| 项长度 | 1Byte | 值：项值的长度 |
| 项的值 | 项长度个Byte | 值：项的值（以字符串形式） |
| 删除测点数据格式 | | |
| 标记 | 1Byte | 值：4 |
| 点名 | 32Byte | 值：点名 |
| 插入测点数据格式 | | |
| 标记 | 1Byte | 值：6 |
| 点名 | 32Byte | 值：点名 |
| 点类型 | 32Byte | 值：类型名 |
| 站号 | 1Byte | 值：站号 |
| 该位置原来点名 | 32Byte | 值：点名 |
| 增加通道点（通过增加模块增加测点）数据格式 | | |
| 标记 | 1Byte | 值：5 |
| 点名 | 32Byte | 值：点名 |
| 点类型 | 32Byte | 值：类型名 |
| 站号 | 1Byte | 值：站号 |
| 设备号 | 1Byte | 值：模块地址 |
| 通道号 | 1Byte | 值：通道号 |

* AT保存时，将操作数据写入共享内存，然后向总控发送窗口消息ID\_ DATABASECONFIG（#define ID\_ DATABASECONFIG 62），通知总控接收数据
* 总控收到数据正确处理后，返回1作为正确接收标识
* AT根据返回值判断总控是否已正确接收数据，如果总控正确接收数据，AT清空操作记录，否则，AT保留原来的记录，下次保存时，将未保存的数据再次发给总控。

#### 测点操作的类图

* CPinBase：派生于Cobject，记录与总控通讯使用的测点操作的基类，也负责删 除操作
* CAddPinImpl：派生于CPinBase，记录增加测点操作
* CModifyPinItemImpl：派生于CPinBase，记录修改测点属性（除测点名）操作
* CModifyPNImpl：派生于CPinBase，记录修改测点名
* CInsertPinImpl：派生于CAddPinImpl，记录插入测点操作
* CHWAddPinImpl：派生于CAddPinImpl，记录通过硬件模块增加测点的操作



图 ‑35：测点操作类图

#### 测点操作的序列化

测点的所有操作记录在CList<CPinBase\*,CPinBase\*> m\_pinOperateList中，由CStation负责对该链表的维护。AT工程关闭时，将操作记录序列化。需要序列化的信息有：操作所占字节数m\_uiLen; 操作标识m\_cFlag; 点名m\_sPN。

* void CStation::Serialize(CArchive &ar)

功能：序列化工程信息

参数：ar ：CArchive对象的引用。

返回值：无

具体调用下列接口完成序列化任务：

* void CPinBase::Serialize(CArchive &ar)

功能：测点操作序列化

参数：ar ：CArchive对象的引用。

返回值：无

### 测点名唯一性处理

在AT上对测点进行操作时，因无法保证点名在总控工程的唯一性，所以对点名有关的操作都要向总控发送判断消息进行处理来保证点名工程内唯一。

AT向总控发送的数据格式：

表 3‑2：向总控发送数据格式

|  |  |  |
| --- | --- | --- |
| 向总控请求一个点名 | | |
| 标记 | 1Byte | 值：1 |
| 站号 | 1Byte | 值：站号 |
| 类型名 | 32Byte | 值：类型名 |
| 修改点名请求总控判断新的点名是否合法 | | |
| 标记 | 1Byte | 值：2 |
| 原点名 | 32Byte | 值：原点名 |
| 新点名 | 32Byte | 值：新点名 |

* AT增加新点时，向总控发送消息VAR\_ADD\_PIN（#define VAR\_ADD\_PIN 1）,向总控请求一个新的点名
* AT修改点名时，向总控发送消息VAR\_MODIFY\_PIN（#define VAR\_MODIFY\_PIN 2），请求总控判断新点名的合法性
* AT根据总控的返回值判断是否成功，返回值MACS\_CLOSR（#define MACS\_CLOSR -1）标识“不能正常与总控通讯”；返回值SERVER\_MACS\_CLOSR（#define SERVER\_MACS\_CLOSR 3）标识“不能与服务器端总控正常通讯”；返回值PN\_RENAME（#define PN\_RENAME 2）标识“点名已被定义”；返回值PN\_VALID（#define PN\_VALID 1）标识点名有效

### 测点变量的撤销恢复

#### 测点撤销恢复的类图

* CGlobalVarOpBase：派生于Cobject，测点变量的撤销、恢复基类操作
* CGlobalVarModifyImpl：派生于CGlobalVarOpBase，修改测点的撤销、恢复操作
* CGlobalVarSubImpl：派生于CGlobalVarOpBase，删除测点的撤销、恢复操作
* CGlobalVarAddImpl：派生于CGlobalVarOpBase，增加、插入测点的撤销、恢复操作



图 ‑35：测点操作类图

#### 测点撤销恢复

测点变撤销恢复操作基本是按“命令模式”设计的，其主要思想是将操作按动作划分为：增加、删除和修改三个类型，根据不同的操作执行相应的行为。涉及的数据结构如下：

全局变量撤销栈m\_UndoList和恢复栈：CObList m\_RedoList。

* void CPinGlobalVarView::OnUndo()

功能：撤销操作

参数：无。

返回值：无

说明：该函数是虚函数，执行具体的撤销工作是在其子类的OnUndo()中完成，其详细的继承关系可参见类图：图300

 图 ‑00：撤销操作流程图

* void CPinGlobalVarView::OnRedo()

功能：恢复操作

参数：无。

返回值：无

说明：恢复操作和撤销操作类似，故在此不赘述。