作者： 陈颂文

日期： 2017/11/23

版本： V1.0

E_mail: s.chen@beckhoff.com.cn

BECKHOFF New Automation Technology

上海市江场三路市北工业园区
163 号 5 楼（200436）
TEL: 020-38010300
FAX: 020-38010303

基于 TC3 的 PLC HMI 报警功能

概 述

技术说明文档模板，为了使测试信息更加可靠，需要详细描述所用的硬件和软件版本，包含但不仅限于以下内容。

文档中包含的文件

文件名称	文件说明
《TC3_PLC_HMI_Alarm》	测试所用的 plc 程序

备 注

关键字：TC3
HMI
报警

免责声明

我们已对本文档描述的内容做测试。但是差错在所难免，无法保证绝对正确并完全满足您的使用需求。本文档的内容可能随时更新，也欢迎您提出改进建议。

文档内容可能随时更新
如有改动，恕不事先通知

基于 TC3 的 PLC HMI 报警功能
V1.0

一、测试设备

1.硬件设备：

工程师自用笔记本电脑。32 位 Win7 操作系统测试 OK。

2.软件版本：

TwinCAT 版本号为 3.1.4022.4

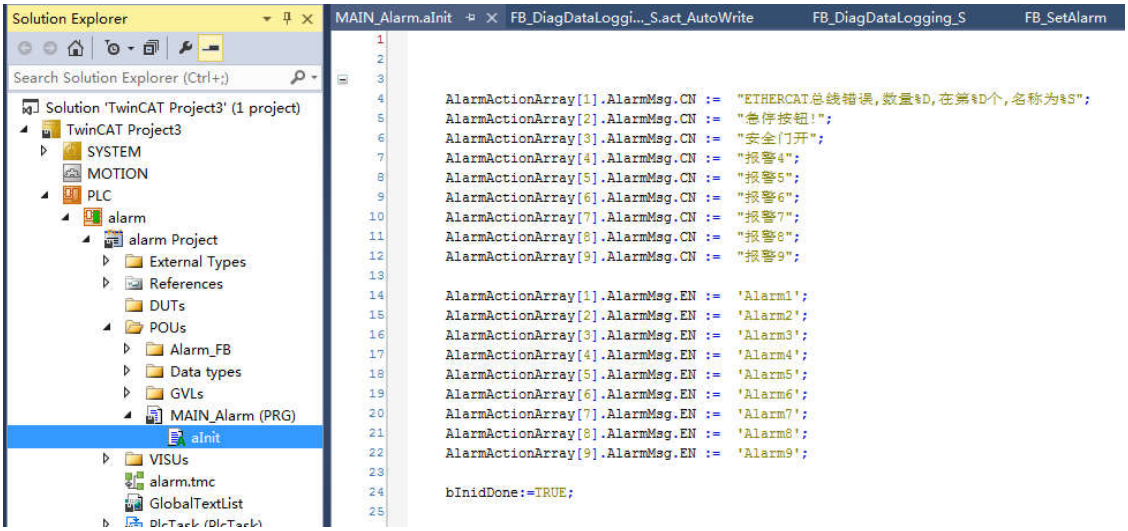
重要提示：TC3 版本必须等于或高于 4022！否则写记录文件时会乱码！编译也不通过。

二、正文：

一）报警功能块

1：定义报警 I D 号及其中英文显示的信息。

提示，善于使用 EXCEL 的功能，可以快速编写报警信息。



例如报警信息定义如下：

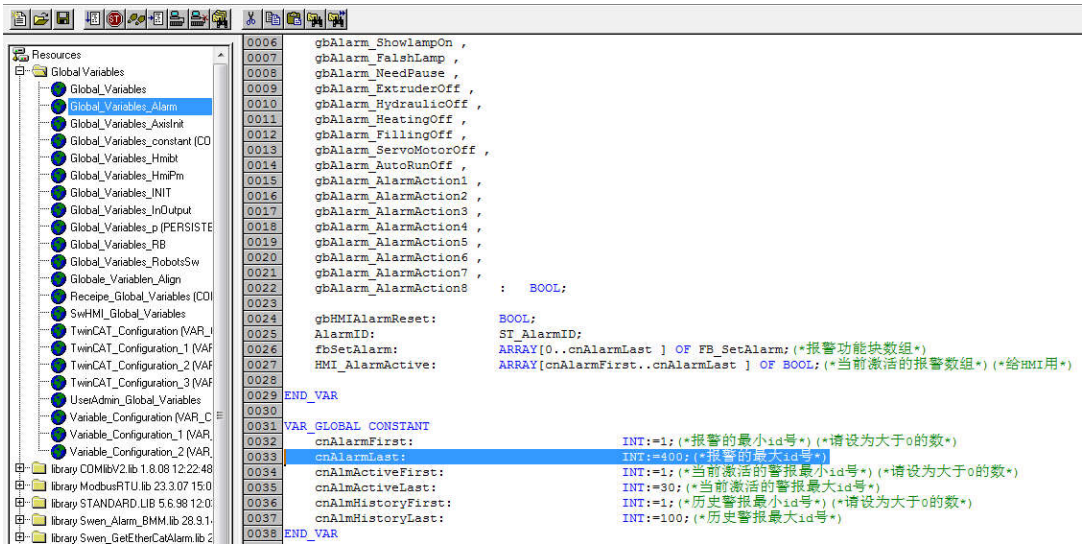
```
AlarmActionArray[1].AlarmMsg.CN :=  "ETHERCAT总线错误,数量%D,在第%D个,名称为%S";
AlarmActionArray[2].AlarmMsg.CN :=  "急停按钮!";
AlarmActionArray[3].AlarmMsg.CN :=  "安全门开";
AlarmActionArray[4].AlarmMsg.CN :=  "报警4";
AlarmActionArray[5].AlarmMsg.CN :=  "报警5";
AlarmActionArray[6].AlarmMsg.CN :=  "报警6";
AlarmActionArray[7].AlarmMsg.CN :=  "报警7";
AlarmActionArray[8].AlarmMsg.CN :=  "报警8";
AlarmActionArray[9].AlarmMsg.CN :=  "报警9";

AlarmActionArray[1].AlarmMsg.EN :=  'Alarm1';
AlarmActionArray[2].AlarmMsg.EN :=  'Alarm2';
AlarmActionArray[3].AlarmMsg.EN :=  'Alarm3';
AlarmActionArray[4].AlarmMsg.EN :=  'Alarm4';
AlarmActionArray[5].AlarmMsg.EN :=  'Alarm5';
AlarmActionArray[6].AlarmMsg.EN :=  'Alarm6';
AlarmActionArray[7].AlarmMsg.EN :=  'Alarm7';
AlarmActionArray[8].AlarmMsg.EN :=  'Alarm8';
AlarmActionArray[9].AlarmMsg.EN :=  'Alarm9';
```

４：ＨＭＩ画面已经把报警列表显示出来了，直接使用就可以了。

当前报警 HMI 画面：

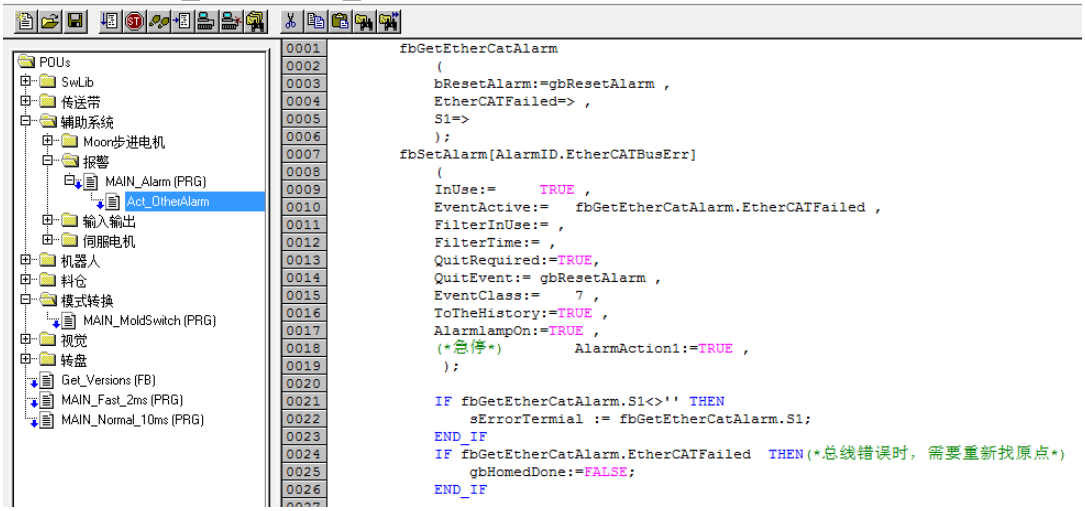
历史报警 HMI 画面：

5：定义报警的最大 i d 号。一般不需要改变。除非你的报警很大超过 50 个。




```
0006    gbAlarm_ShowlampOn ,
0007    gbAlarm_FalshLamp ,
0008    gbAlarm_NeedPause ,
0009    gbAlarm_ExtruderOff ,
0010    gbAlarm_HydraulicOff ,
0011    gbAlarm_HeatingOff ,
0012    gbAlarm_FillingOff ,
0013    gbAlarm_ServoMotorOff ,
0014    gbAlarm_AutoRunOff ,
0015    gbAlarm_AlarmAction1 ,
0016    gbAlarm_AlarmAction2 ,
0017    gbAlarm_AlarmAction3 ,
0018    gbAlarm_AlarmAction4 ,
0019    gbAlarm_AlarmAction5 ,
0020    gbAlarm_AlarmAction6 ,
0021    gbAlarm_AlarmAction7 ,
0022    gbAlarm_AlarmAction8    :   BOOL;
0023
0024    gbHMIAlarmReset:        BOOL;
0025    AlarmID:                ST_AlarmID;
0026    fbSetAlarm:             ARRAY[0..cnAlarmLast ] OF FB_SetAlarm;(*报警功能块数组*)
0027    HMI_AlarmActive:        ARRAY[cnAlarmFirst..cnAlarmLast ] OF BOOL;(*当前激活的报警数组*)(*给HMI用*)
0028
0029 END_VAR
0030
0031 VAR_GLOBAL CONSTANT
0032    cnAlarmFirst:                       INT:=1;(*报警的最小id号*)(*请设为大于0的数*)
0033    cnAlarmLast:                        INT:=400;(*报警的最大id号*)
0034    cnAlmActiveFirst:                   INT:=1;(*当前激活的警报最小id号*)(*请设为大于0的数*)
0035    cnAlmActiveLast:                    INT:=30;(*当前激活的警报最大id号*)
0036    cnAlmHistoryFirst:                  INT:=1;(*历史警报最小id号*)(*请设为大于0的数*)
0037    cnAlmHistoryLast:                   INT:=100;(*历史警报最大id号*)
0038 END_VAR
```

6：在 MAIN_Alarm. Act_OtherAlarm 里面编写报警的触发条件。




```
0001        fbGetEtherCatAlarm
0002            (
0003            bResetAlarm:=gbResetAlarm ,
0004            EtherCATFailed=> ,
0005            S1=>
0006            );
0007        fbSetAlarm[AlarmID.EtherCATBusErr]
0008            (
0009            InUse:=     TRUE ,
0010            EventActive:=   fbGetEtherCatAlarm.EtherCATFailed ,
0011            FilterInUse:= ,
0012            FilterTime:= ,
0013            QuitRequired:=TRUE,
0014            QuitEvent:= gbResetAlarm ,
0015            EventClass:=    7 ,
0016            ToTheHistory:=TRUE ,
0017            AlarmlampOn:=TRUE ,
0018            (*急停*)        AlarmAction1:=TRUE ,
0019             );
0020
0021            IF fbGetEtherCatAlarm.S1<>'' THEN
0022                sErrorTermial := fbGetEtherCatAlarm.S1;
0023            END_IF
0024            IF fbGetEtherCatAlarm.EtherCATFailed  THEN(*总线错误时，需要重新找原点*)
0025                gbHomedDone:=FALSE;
0026            END_IF
```

其中报警功能块在全局变量里面已经定义了，不需要重复定义。

```
fbSetAlarm:             ARRAY[0..cnAlarmLast ] OF FB_SetAlarm;(*报警功能块数组*)
```

7：注意报警触发功能块的引脚定义：

```
FUNCTION_BLOCK FB_SetAlarm(*设置报警功能块,每个报警调用一次*)
(*=========================================
by Swen 陈颂文
GuangZhou
2009-08-26
=========================================*)
VAR_INPUT
    InUse:                  BOOL;(*是否使用*)
    EventId:                INT;(*报警号*)
    EventMessage:           STRING;(*报警信息*)(*只是程序检查用，可以不填*)
    EventActive:            BOOL;(*触发信号*)
    FilterInUse:            BOOL:=FALSE;(* 是否使用滤波器*)(*对触发信号进行滤波，就是触发信号超过多少时间后才报警*)
    FilterTime:             TIME;(*滤波时间*)
    QuitRequired:           BOOL;(*是否自动复位：true=复位信号为1时复位报警信号;false=触发信号为0时复位报警信号*)
    QuitEvent:              BOOL;(*复位信号*)
    EventClass:             Alm_EventClass;(*报警等级*)(*一般急停的 7 等级，其它 3 等级就好了*)
    ToTheHistory:           BOOL;(*是否记录到报警历史记录*)
    (* event actions    *)  (*－－－－－－下面的引脚为选用－－－－－－－－－－*)
    AlarmlampOn:            BOOL;(*是否需要报警灯亮*)
    WtcOff:                 BOOL;(*关壁厚*)
    ExtruderOff:            BOOL;(*关螺杆*)
    HydraulicOff:           BOOL;(*关液压*)
    HeatingOff:             BOOL;(*关加热*)
    FillingOff:             BOOL;(*关灌装*)
    ServoMotorOff:          BOOL;(*关伺服电机*)
    AutoRunOff:             BOOL;(*退出自动循环*)
    AlarmAction1:           BOOL;(*用户定义报警动作*)(*例如定义为黄灯亮,则该报警号有触发时,主报警程序的该输出信号为1*)
    AlarmAction2:           BOOL;(*用户定义报警动作*)
    AlarmAction3:           BOOL;(*用户定义报警动作*)
    AlarmAction4:           BOOL;(*用户定义报警动作*)
    AlarmAction5:           BOOL;(*用户定义报警动作*)
    AlarmAction6:           BOOL;(*用户定义报警动作*)
    AlarmAction7:           BOOL;(*用户定义报警动作*)
    AlarmAction8:           BOOL;(*用户定义报警动作*)
END_VAR
```

其中：

QuitRequired:　True 时，为自锁类型，必须按了复位按钮才能消除。

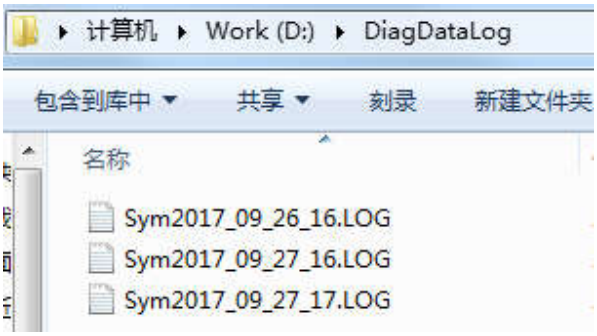
False 时，报警触发(EventActive)有就报警，没有了就自动消除。

EventActive:　报警触发输入

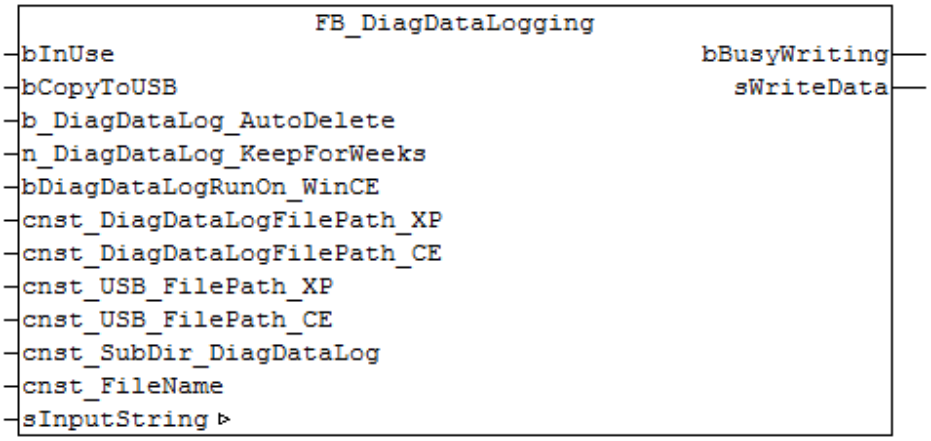
二）报警的历史记录文件

例如，在控制器里面保存历史报警的 LOG 文档。工程师开发方便的把文档拷贝出来做分析。如果工程师不在车间现场，客户也可以把该文件夹发送给工程师，帮助寻找并解决问题。

如下图，历史记录文件，以“年月日时”为名称。也就是说，每小时产生一个新的文件，以防止文件过大。



1：后台记录功能块的使用



提供了功能块 FB_DiagDataLogging 作为数据处理的后台程序。
例如在例子程序里面的“MAIN_Alarm”程序里面就定义并调用了该功能块。
同时在全局变量里面，定义了 string 类型的” gsWriteToLog”变量。并作为输入输出类型，链接到了 FB_DiagDataLogging 的 sInputString 引脚。

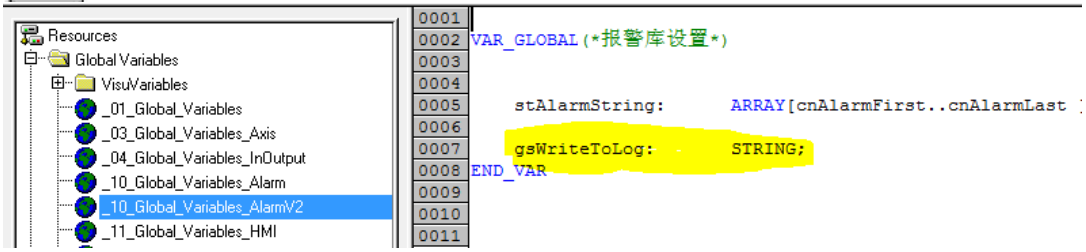
工作流程原理：
1：检测到有报警，则把报警需要记录的字符串赋值给” gsWriteToLog”变量
2：FB_DiagDataLogging 检测到” gsWriteToLog”变量不为零，则记录下需要写的字符串。记录值放到 [1..10] OF STRING 的中间变量内。做中间存储变量的目的，是当文件没有写完的时候，又有报警记录进来，则可以缓存一下。
3：FB_DiagDataLogging 把” gsWriteToLog”变量置零，等待其它报警的触发。
4：FB_DiagDataLogging 检测到[1..10] OF STRING 的中间变量不为零，则开始写文件。
5：写文件结束。

缺点：
1：TC3 能用中文字符，但是**请先转换成 UTF8 的格式**。可以使用“string_to_utf8”功能。
2：当同一个 PLC 周期有几个报警同时触发时，只能记录到一个最后刷新数据。
解决方法：把几个报警的字符串合并成一个字符串。例如：
gsWriteToLog:= CONCAT(gsWriteToLog, sNewLog);

不同 PLC 周期则没有问题。因为有中间缓存数据，有记忆功能。

```
0001 PROGRAM MAIN_Alarm
0002 VAR
0003
0004     bInitState:              BOOL:=FALSE;
0005     i:                       INT;
0006
0007     gAlarmMain:              FB_AlarmMain;
0008
0009     TON_Delay:               TON;
0010     TON_TimeOutCount:        TON;
0011     fbDiagDataLogging:       FB_DiagDataLogging;
0012
0013 END_VAR
0014
```

```
0001
0002 IF NOT bInitState THEN
0003     FOR i:=cnAlarmFirst TO cnAlarmLast DO
0004         fbSetAlarm[i].EventId:=i;
0005     END_FOR
0006     bInitState:=TRUE;
0007 ELSE
0008     TON_Delay(IN:=TRUE , PT:=T#2s );
0009     IF TON_Delay.Q THEN
0010         Act_OtherAlarm();
0011         gAlarmMain();
0012         gAlarmMain.FalshLamp;(*闪灯*)
0013         gbAlarm_StopAlarm := gAlarmMain.AlarmAction1;(*停止级别警报*)
0014         gbAlarm_NeedPause := gAlarmMain.AlarmAction2;(*需要暂停*)
0015     END_IF
0016 END_IF
0017
0018
0019 fbDiagDataLogging(sInputString:=gsWriteToLog);(*历史记录文件*)
0020
```



使用诀窍：

1：写”gsWriteToLog”变量只用一个 PLC 周期就可以了。如果持续多个 PLC 周期则会写很多次！建议使用上升沿触发写一次就行了。

2：FB_DiagDataLogging 可以实例化多个。

把不同实例里面的下面输入引脚改变一下，就可以写多个不同文件啦！

```
bDiagDataLogRunOn_WinCE:        BOOL:=FALSE;
cnst_DiagDataLogFilePath_XP:    STRING(255):='D:\';
cnst_DiagDataLogFilePath_CE:    STRING(255):='Hard Disk\';

cnst_USB_FilePath_XP:           STRING(255):='E:\';
cnst_USB_FilePath_CE:           STRING(255):='Hard Disk2\';

cnst_SubDir_DiagDataLog:        STRING(255):='DiagDataLog\';
cnst_FileName:                  STRING(255):='Sym';(*文件的前缀名*)
```

比如：
报警记录写到文件夹《DiagDataLog》，文件前缀为“Event”。
系统信息，如 CPU 启动、CPU 温度、电压等等信息，写到文件夹《DiagSymLog》，文件前缀为“Sym”。
其它需要记录的信息等等。

2：报警功能块的使用

功能块 FB_SetAlarm 的输入引脚 ToTheLogFile 为 TRUE 的时候，则记录到全局变量 gsWriteToLog，则写入到文件。当引脚 ToTheLogFile 为空的时候，则不记录到文件。

代码如下：

```
0017 IF F_TRIG_GetTime.Q THEN
0018         AlarmActionArray[EventId].EventTime:=SYSTEMTIME_TO_DT(fb_NT_GetTime.TIMESTR);
0019         IF EventMessage<>'' THEN
0020             sNewLog:=CONCAT( CONCAT(CONCAT('ID',INT_TO_STRING(EventId)),','),EventMessage);(*写入报警记录文件*)
0021             gsWriteToLog:= CONCAT(gsWriteToLog, sNewLog);
0022         END_IF
0023 END_IF
```

3：报警功能块的使用 2

当需要定制化修改报警显示的内容时，可以使用新的引脚：D1,D2,S1(需要注意大小写)

例如：
我们定义了报警的中文内容如下：
AlarmActionArray[1].AlarmMsg.CN := "ETHERCAT 总线错误,数量%D,在第%D 个,名称为%S";

当我们产生了总线报警的时候，我们不单单需要给出报警，还有提示用户是哪个模块发生了错误导致的总线错误。这时的报警就是定制化的可修改报警了。
在“ETHERCAT 总线错误,数量%D,在第%D 个,名称为%S”z 这个报警内容中，包含了字符两个“%D”和一个“%S”。通过报警功能块的新引脚 D1,D2,S1,把报警内容的字符替换成引脚内容，实现定制化报警的功能。

例如:
报警定义：ETHERCAT 总线错误,数量%D,在第%D 个,名称为%S　(其中%D ,%S 需要注意为大写)

D1:=2
D2:=8
S1:='EL3314'
那么,报警输出显示为：THERCAT 总线错误,数量 2,在第 8 个,名称为 EL3314

1	2017/11/23,11:36:38,ID1,ETHERCAT总线错误,数量2,在第8个,名称为EL3314

历史报警，可以区分出来是哪个模块：

1	2017/11/23,13:48:57,ID1,ETHERCAT总线错误,数量5,在第3个,名称为EI1008
2	2017/11/23,13:47:55,ID1,ETHERCAT总线错误,数量5,在第3个,名称为EI2008
3	2017/11/23,13:47:38,ID1,ETHERCAT总线错误,数量2,在第8个,名称为EL3314
4	2017/11/23,12:37:40,ID1,ETHERCAT总线错误,数量2,在第8个,名称为EL3314
5	2017/11/23,12:37:39,ID1,ETHERCAT总线错误,数量2,在第8个,名称为EL3314
6	2017/11/23,12:37:39,ID1,ETHERCAT总线错误,数量2,在第8个,名称为EL3314

比如：NC 报警的时候，可以在报警信息里面显示报警代码。
超时报警的时候，可以显示超过了多少时间。
显示产品重量不合格的时候，显示产品的实际重量。
等等其他运用.....

4：报警功能块的使用 3
上面实现了功能块增加定制字符的功能。
其实，如果需要更彻底的修改显示内容，只需要每次触发报警之前，改变一下 AlarmActionArray[AlarmID].AlarmMsg.CN 的内容就可以了。