**LODAS软件设计文档**

# 概述

LODAS软件系统现有功能梳理，新功能开发设计，用于指导LODAS后续软件开发与优化

# 功能设计

## 原有Aging功能改造

### 具有AutoAF选项

作用：不选择用户需要三步走进行检查

Step 1 start position

Step 2 AF

Step 3 Inspection

如果选择了AutoAF，用户无论是按start position 还是按Inspection，都会自动执行这三步。实际就是autoInspecting

### 具有aging选项

与AutoAF无关，如果没有AutoAF，则Step 3 Inspection循环多次

如果有AutoAF，则整个autoInspecting循环多次

Aging的控制是在ScanEnded时走了一个ScanAgingEnded分支，会重新启动btnStartInspection\_Click

## Stage重构

### 坐标系

#### 原点的概念

左上角为原点：X轴往右是增大，Y往下是增大

左下角为原点：X轴往右是增大，Y往上是增大，与数学中的几何坐标系一致

圆心为原点：X轴往右是增大，Y往上是增大，与数学中的几何坐标系一致

#### 当前小机的原点分类

机械坐标系：由cpd530决定，8号机与BI12不同

用户坐标系：在map的左下角，stage上的天是Y 最大值 ， stage上的地是0点，stage天在上，地在下时，左侧是X 0点，右侧X 最大值。

用户坐标系，是软件上看到的实际坐标系，是左下角为原点

Mask（glass）坐标系 与用户坐标系方向性一致：用户选择recipe之后，

如果是四边形，原点是白色选择区域的左下角

如果是圆形，mask的原点是圆形的圆心

另外，从软件角度讲，在map绘制时，vs程序中的坐标比如**Graphics，Rectangle，mouse移动时的坐标**，使用的是左上角为原点

8号机的机械坐标系与用户坐标系的关系

1 机械原点位置，镜头在左上角（这里的左上角，要以天在上 地在下的方向判断）

2 镜头X要增加，实际机械X增加

3 镜头Y要增大，实际机械Y要减小

BI12的机械坐标系与用户坐标系的关系

1 机械原点位置，镜头左下角

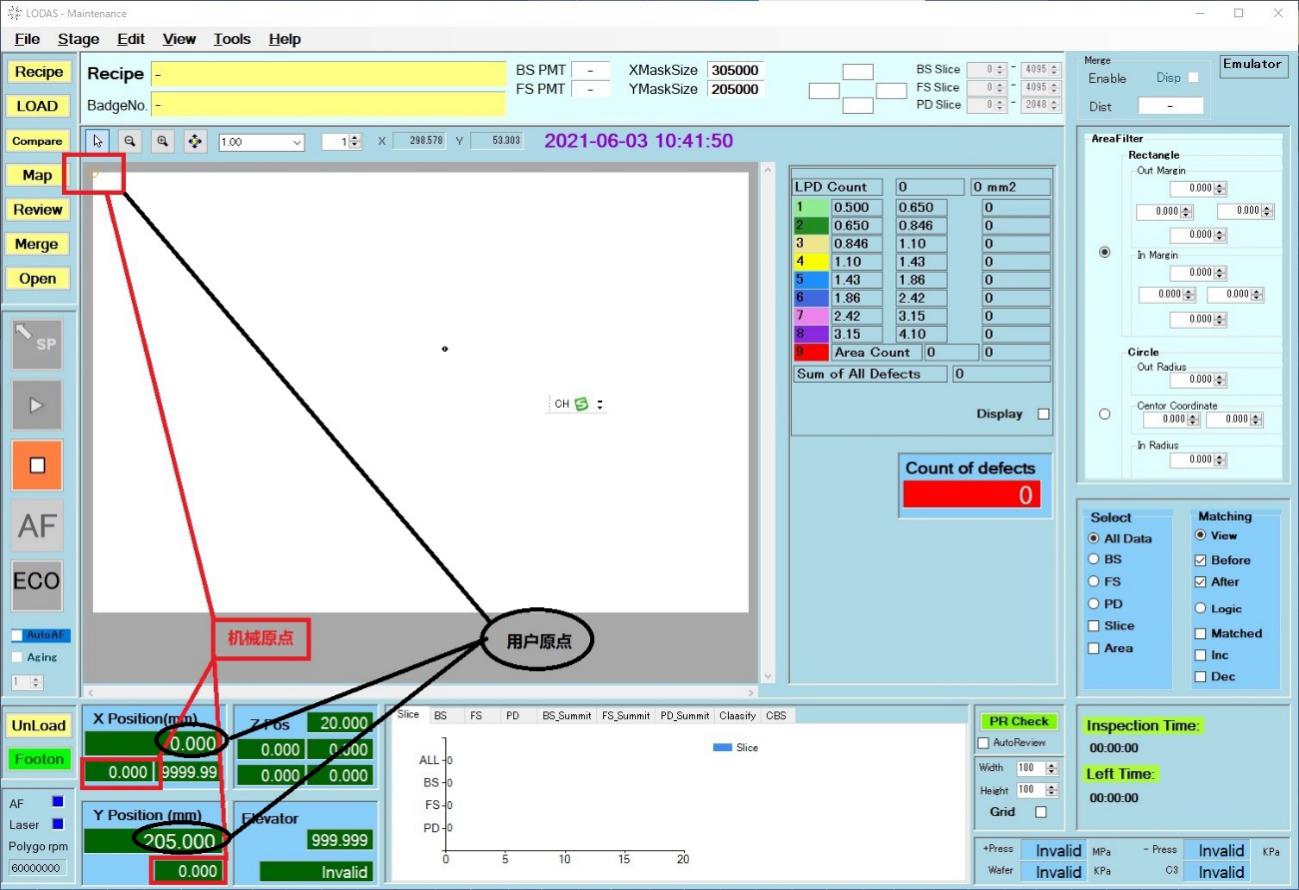
2 镜头X要增大，实际机械坐标要增大 （X轴与镜头一起运动）

3 镜头Y要增大，实际机械Y要增大

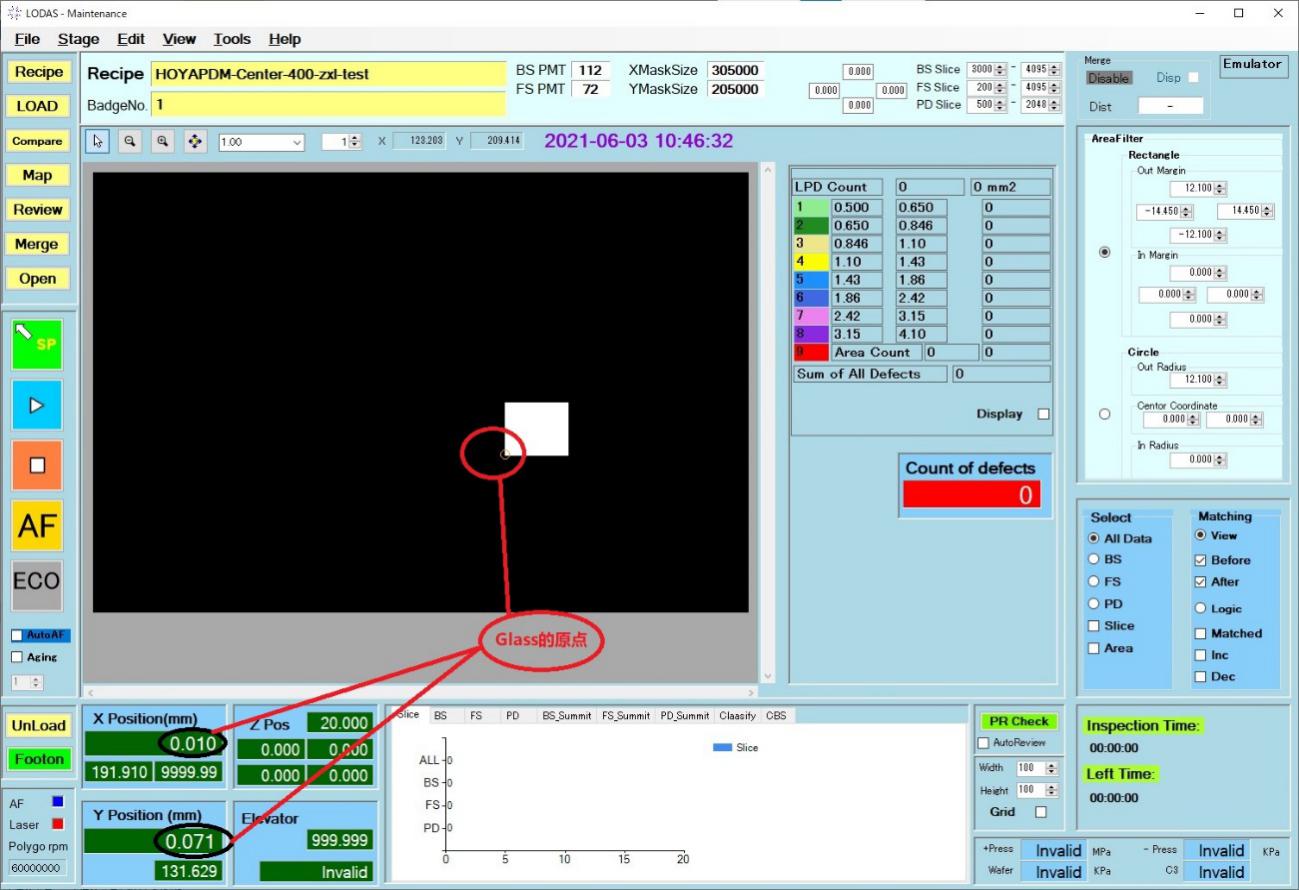
8号机与BI12异同点：

1 X轴的移动相同

2 Y轴的移动不同



图表 0‑1：机械原点和用户原点示意



图表 0‑2：Glass的原点示意

需要注意的是，scan的时候不是从glass的原点开始，而是从左上角开始扫描，并且产生的缺陷坐标原始坐标是和机械坐标一致的。保存数据和显示glass坐标时，需要做转化

当前程序中配置也两个Size，StageSize和MaskSize，当前配置的是一样大小，才能正常工作。

所以代码中的MaskSize含义已经不准确，真正的MaskSize是就通过recipe当中的offset来完成设置的

Stage中重要的成员stg\_ofs：因为StageSize和MaskSize相同，所以在激光头在位的情况下是0；只有选择ccd时，才会增加一个ccd的位置偏移。

stg\_ofs 在外部使用时不用关心，因为在移动的底层函数GoAbsCore，已经处理掉

## 仿真模块设计

用于实现离线仿真，提高开发效率。离线仿真需要仿真的模块主要针对硬件进行模拟。重点的实现对象是DPOS GDET STAGE ROBOT4个子模块

### DPOS GDET

是LODAS开发的硬件板卡，真实的DLL模块需要硬件PCI和PCIE板卡以及驱动支持。

离线模块需要模拟寄存器读写，以及缺陷数据读取

DPOS所有功能都是依赖对寄存器的读写操作，主要需要实现FootOn与FootOff功能

GDET 除了有寄存器的读写，还有缺陷数据的读取，Image数据读取，波形数据读取

### STAGE

需要模拟平台移动，坐标上报，Origin

#### 设计思路

STAGE使用工厂模式，直接从基类扩展一个子类，然后根据具体的动作要求进行详细设计

基类：StgHandler.

子类：StgEmulatorHandler

#### 设计规格

1 需要实现多轴运动，X1 X2 Y目前至少需要实现3个轴，并且需要单独控制移动

2 需要实现Origin功能

3 需要实现速度调节

4 需要实现X1 X2 Y轴单独移动

5 移动的精度是微米级别

#### 需求实现

多轴单独移动运行

需要多线程控制，每个线程控制一个轴

线程内使用While + sleep方式，一次循环移动的距离 ， sleep的时间两个参数可调节

两个参数需要根据速度进行调节

Origin功能

Origin功能与普通的移动没有区别，在速度下发时设置运行模式为ORG模式即可

### Robot

需要模拟玻璃的移动

# 新需求开发

## Io集成

### 集成条件

1. 拿到厂商提供的demo（DIO864\_W\_7\_30J）

里面有ethernet和usb，目前使用的是ethernet

Ethernet 里面

示例代码位于DIO864\_W\_7\_30J\Ethernet\sample目录下，vcs为C#工程

另外还有一个驱动程序驱动位于DIO864\_W\_7\_30J\Ethernet\dll目录下

### 函数接口

C#接口

使用的头文件hedio864.cs

使用的DLL是D:\code\tools\DIO864\_W\_7\_30J1\DIO864\_W\_7\_30J\Ethernet\dll\x64\System64\ hedio864.dll

#### 设备获取过程

ret = Ed864l1a.hed864\_ProgLinkSet(0, "", 0); //常驻程序链接设置（自动检测）

ret = Ed864l1a.hed864\_GetProgVersion(ref pver, ref lver);//常駐プログラムバージョン取得と表示

ret = Hedio864.ed864\_GetDeviceCount(ref cnt); // // ボード枚数取得

ret = Hedio864.ed864\_GetDeviceInfo(ref cnt, hInf); // 获取设备

public const uint CNTMAX = 16; // デバイス最大個数(HETN/HWIF合わせて16枚まで)

public Hedio864.HETNDEVINF[] hInf = new Hedio864.HETNDEVINF[CNTMAX];

public int gDevNo = 0; // デバイスインデックス番号

public short gDevTyp = 0; // デバイスタイプ

public short gBrdId = 0; // ボードID

#### 打开一个设备

gDevNo = CmbInfo.SelectedIndex;

gInf = hInf[gDevNo];

gDevTyp = gInf.DevTyp;

gBrdId = gInf.BrdID;

ret = Hedio864.ed864\_GetOpenDevCount(gDevTyp, gBrdId, ref nNum); //检查设备是否已经打开

ret = Hedio864.ed864\_OpenDevice(gDevTyp, gBrdId); // デバイスオープン

ret = Hedio864.ed864\_rVersion(gDevTyp, gBrdId, ver); // デバイスバージョン情報取得

//フィルター値を全ﾎﾟｰﾄ/全ﾋﾞｯﾄを1msに設定(U/T/W専用関数を使用)

gFilter = 10; //filter no=10(1ms)

ret = Hedio864.ed864\_SetFilter(gDevTyp, gBrdId, 0xff, (ushort)gFilter);

#### 关闭设备

// すべての出力ポートに０を出力

ret = Hedio864.ed864\_wOutDW(gDevTyp, gBrdId, 0);

// デバイスクローズ

ret = Hedio864.ed864\_CloseDevice(gDevTyp, gBrdId);

#### IO操作

ret = Hedio864.ed864\_rOutDW(gDevTyp, gBrdId, ref rdt); //读取out 32bit 返回0 代表成功

ret = Hedio864.ed864\_wOutDW(gDevTyp, gBrdId, wdt); // 写入32bit 返回0 代表成功

ret = Hedio864.ed864\_rInDW(gDevTyp, gBrdId, ref indt); // 读取 in

### LODAS库

#### HdioLib

#### 接口

1. 打开设备

默认只有一台设备，打开设备时先获取设备

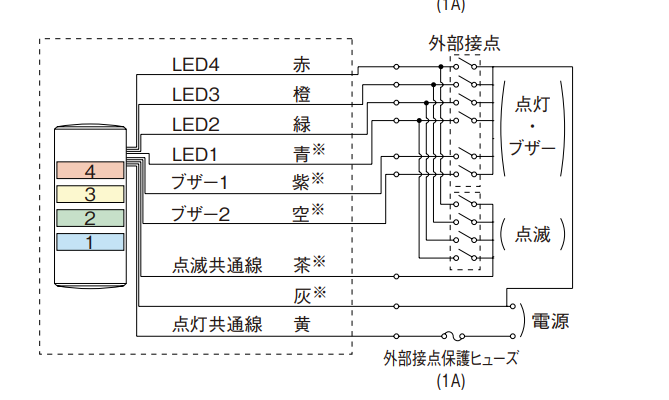
1. 关闭设备
2. In口读取
3. Out读取
4. Out 写入

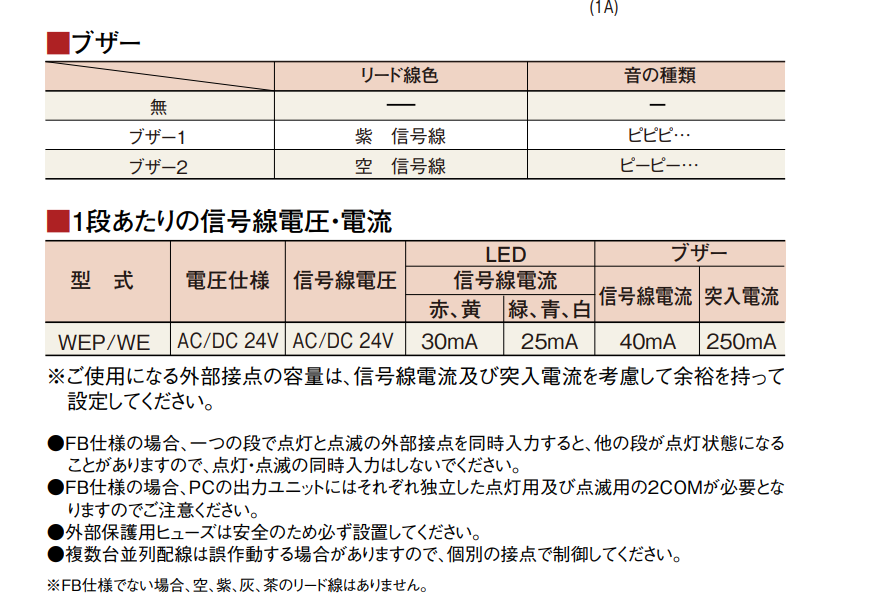
#### 要点

使用静态函数接口

## 告警灯

选型：WEP/WE



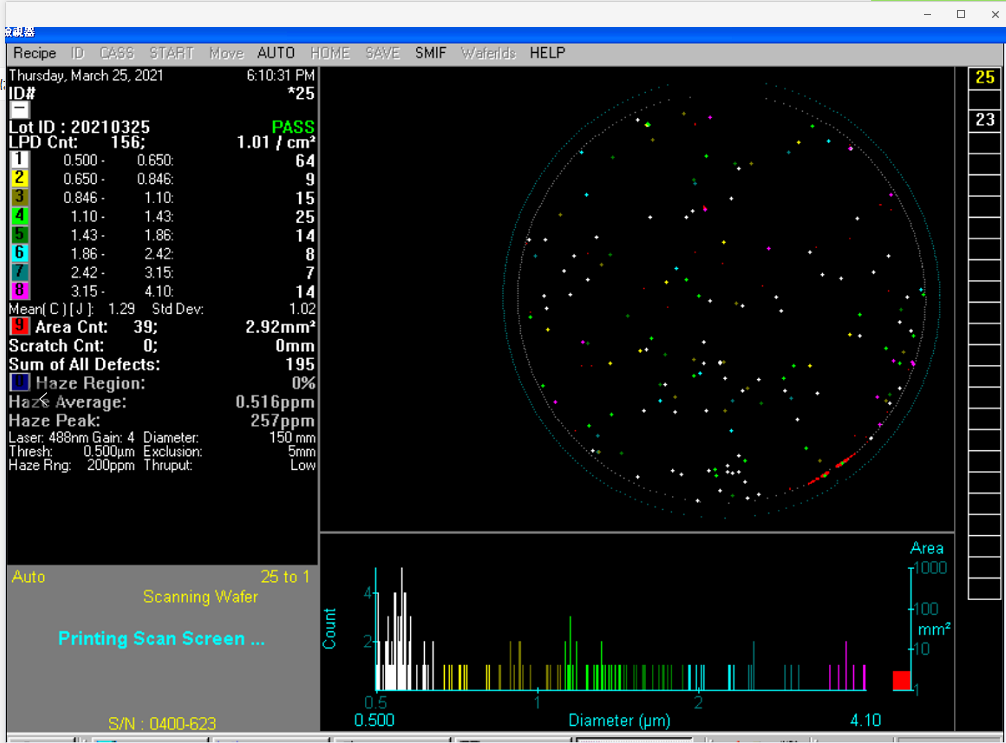


## MES开发

## 基于缺陷大小的结果显示

Classify Based on Size (CBS)

### 需求分析



如图所示，客户的结果基于缺陷的大小进行分类，而当前我们的结果基于光高进行分类。

本次实现的重点是将现有数据从光高转换为大小，找到缺陷大小与光高的对应关系

1 简单的对应，缺陷的大小与FS BS中的最大值有一定的比例关系

2 复杂的对应，通过缺陷的merge功能，classify功能计算出像素面积，即缺陷的大小

### 模块设计

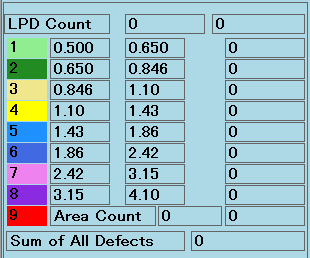
#### 概述

1. 对应的缺陷大小有9个档，共有9个上下限值，需要通过ini进行配置
2. 缺陷大小对应的FS BS光高大小要进行对应，需要进行ini配置
3. 在检查过程中计算出缺陷的大小，并保存到缺陷信息当中。
4. 设计界面并显示输出结果。

#### 界面显示开发

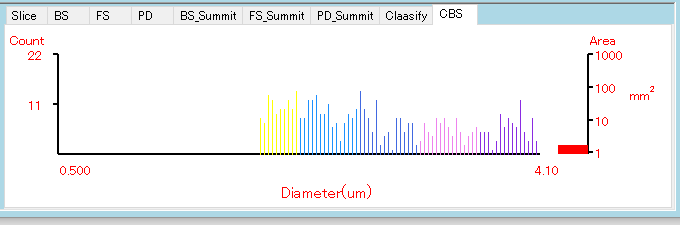
由两个界面组成

界面1：统计结果



界面1使用的数据是9个区间

界面2：柱状图



界面2使用的数据颜色部分使用了9个区间，柱状图使用了205个区间，按照光高20为跨度，将4096分成了205个区间。

#### 数据结构

1. Res中增加数据结构

value struct CBSValue {

Double cbsLeftThreshold; // config value

Double cbsRightThreshold; // config value

Int32 leftThreshold; // config value

Int32 rightThreshold; // config value

Int32 defectCount; //calculation value

};

array<CBSValue>^ cbsArray;

array<CBSValue>^ cbsArray\_M;

当前保存了merge和非merge两个结果 array的大小为9个（用于界面1的数据显示）

array<Int32>^ cbsCount;

array<Int32>^ cbsCount\_M;

用于界面2的结果显示，array的大小为4096个；即跨度是1，在界面2显示中目前使用的跨度是20，如果后面要调整，该数据结构不需要变动

res初始化时分配空间并从ini读取并保存配置值

各区间的颜色分配

1 System::Drawing::Color::LightGreen

2 System::Drawing::Color::ForestGreen

3 System::Drawing::Color::Khaki

4 System::Drawing::Color::Yellow

5 System::Drawing::Color::DodgerBlue

6 System::Drawing::Color::RoyalBlue

7 System::Drawing::Color::Violet

8 System::Drawing::Color::BlueViolet

9 System::Drawing::Color::Red

#### 对外接口

public: Void CBSProcessAll();

在动态检查时调用，生成CBS数据

public: Int32 CBSGetIndex(Int32 val);

返回一个值属于的区间，用于颜色分配，颜色与区间一一对应

#### 流程

单个数据的cbsLevel在缺陷生成时计算得到

整体的数组的cbs的统计区别如下情况

1 离线时读取文件时生成

2 在线时在检查结束时生成

3 如果是用户使用了Filter更新，则在更新时生成

更新时涉及CBS数据的更新，和CBS界面的更新

CBS界面的更新使用 UpdateCBSStatus

CBS数据的更新

如果是在线和离线的使用CBSProcessAll

如果是Filter更新使用CBSProcessFilter

## 多国语言支持

由于目前业务发展壮大，客户所在地区不同，客户对于软件的本地化需求也越来越多。

### 功能描述

在IniForm中，独立设置一个语言页，客户可自行选择需要的语言，重启软件即刻生效。

目标设计一个独立的多国语言模块，可用于所有UI界面，仅需在每个Form初始化的时候调用一次，即可实现语言的本地化切换，且需要独立UI的文本翻译文件，方便今后翻译的外包、翻译的更新、以及软件的开发。

### 设计思路

既然要独立翻译文件，那就选择以某种格式来存储翻译文件。现在最常见的5种配置文件格式有Text、INI、XML、JSON、YAML，目前暂定使用JSON格式，原因是JSON格式相比其他格式更简洁且易修改。

JSON文件目前打算以”Form名.语言缩写”为文件名存储，方便排序和查找。例如：MainForm.chs、MainForm.jpn、ReviewForm.chs、ReviewForm.jpn等。

多国语言模块方面，目前的设计思路是传入两个参数，一个是需要读取的JSON文件，二是传入当前窗体的控件合集this->Controls。模块解析JSON文件，根据JSON文件把翻译类型分为4类：窗体Text值、项目Items内容、提示面板Tips文本、特殊变量值。然后遍历当前窗体的控件合集this->Controls，将控件和JSON本文进行匹配，找到对应的翻译值进行赋值。

### 功能实现

待完善。