# 1、对于自定义绘制的控件\组件

在其初始化调用时,必须设定控件的样式

//表明控件需要时重绘，减少闪烁

This.SetStyle（ControlStyles.AllPaintingInWmPaint,true）;

//表明控件双重绘制减少闪烁，绘制完成直接呈现

This.SetStyle(ControlStyles.DoubleBuffer,true);

//表明调账控件大小之后绘制

This.SetStyle(ControlStyles.ResizeRedraw,true);

//表明控件可以被选中

This.SetStyle(ControlStyles.Selectable,true);

//表明控件的透明度

This.SetStyle(ControlStyles.SupportsTransparetBackColor,true)

绘制控件的新方法

Protected override void OnPaint（PaintEventArgs e）

{

base.OnPaint(e);

//在这里可以将

//构建绘画对象，笔，颜色，绘制的图形（创建对应图形所需要的东西）

Graphics graphics = e.Graphics;

//表明绘制的直线，曲线，出现抗拒尺的形状，这意味着在绘制图形时，使用抗锯齿技术来平滑处理边缘，

//以减少锯齿状的边缘，使图形看起来更加平滑和清晰。抗锯齿技术是为了改善图形的视觉质量。

graphics.SmoothingMode=System.Drawing.Drawing2D.SmoothingMode.AntiAlias;

//这指定了在绘制文本时要使用 ClearType 技术以更好地渲染文本。

//ClearType 是一种亚像素抗锯齿技术，用于改善文本的清晰度和可读性。

graphics.TextRenderingHint=System.Drawing.Text.TextRenderingHint.ClearTypeGridFit;

//在绘制时，可以旋转和平移坐标系

//平移坐标系

graphics.TranslateTransform(this.Width/2, this.Width/2);

//旋转坐标系

graphics.RotateTransform(-90);

//绘制字符串

// 首先要得到需要绘制的高度和宽度，坐标

// 4、通过Grapics，算出文本大小

float x = Convert.ToSingle(this.Width\*this.TextScale\*0.5f\*Math.Cos(slace\*Math.PI/180.0f));

float y = Convert.ToSingle(this.Width\*this.TextScale\*0.5f\*Math.Sin(slace\*Math.PI/180.0f));

string text = ((rangeAvg\*i+RangeMin)).ToString();

//存储一对有序的浮点数对，一般为矩形的高度和宽度

SizeF size = graphics.MeasureString(text, this.Font);

//存储一组浮点数（4个）,表示坐标，长，宽

RectangleF rc = new RectangleF(x-size.Width\*0.5f, y, size.Width, size.Height);

graphics.DrawString(text, this.Font, new SolidBrush(this.ForeColor), rc, stringFormat);

}

# 2、对于窗体的切换

思路，首先是通过，定义一个枚举Enum,来将几个From窗体的Text

通过自定义的按钮NaviButton其自定义属性TitleText，对应着窗体的Text，写一个通用方法NaviButtonCommon\_Clicke

通过判断其除非的事件对象的类型

1. 判断除非事件对象是不是NaviButton控件
2. 判断其TitleName是不是所创建Enum中包含的窗体名Enum.IsDefined(typeof(FormNames), navi.TitleName)；
3. 通过(FormName)Enum.Parse（typerof（FormNames），navi.TitleName,true）,用Enum.Parse转换将字符串转化为Enum类型，ture表示不区分大小
4. 最后通过Switch来判断是何窗体Name，
5. 在进行窗体切换前，需要判断，承载窗体容器的Panel，中的其他控件，是不是窗体，并且判断是不是我们所选择的窗体，是则不变，不是则判断是否大于临时窗体，不是删除，是则缓存；如果不是所选择的窗体，则通过Switch来判断是何窗体显示

# 3、对于工控的PLC分析

1、对于PLC，我们将其分为三类，第一类Decive，对应的是工控控制器的IP地址，端口号，或者从站地址，在其创建一个Group的List集合

2、Group表示的就是工控软件的存储区，输入线圈0X01，输出线圈0X00，输入寄存器0X03，输出寄存器0X04,在其中创建对应的Variable的List集合，其中存储每个存储区种的所有变量

3、Variable，表明所有的变量名，变量类型，偏移量或长度，以及起始地址，以及所属的Grounp

对此需要创建Decive的INI文件，来存储IP地址和端口

这里我们使用了INIconfigHelper,在其中有两个方法，一个写入，提供一个标题名，一个键值对，一个键值对的值，以及文件路径；一个读取，分别需要提供文件类的标题Section，以及Key用来提取我们所需要的值，以及当键值对不存在的时候返回的值，以及文件路径

对于Group文件，需要通过MiniExcel.Query<Group>(groupPath).ToList()还有写入

MiniExcel.SaveAs(groupPath,TotalGroup,overwriteFile:true);将其保存为Excel文件，需要通过MiniExcel.dll实现，将一个对象保存到Excel种，也可也将Excel转化为一个对象，学到一个新的List方法

TotalGroup.FindAll(c => c.GroupName==groupName).ToList().Count>0

将一个List<T>,如果T是一个对象，就寻找其中满足某一个属性要求的对象全部寻找出来

对于Variable来说，就是通过MiniExcel.dll来保存变量以及读取变量

# 4、定义全局的对象CommonModel

1. 首先在对象中定义好PLC的配置参数的Decive属性，方便整个程序，各个不同的窗体，在进行对PLC数据的采集以及将所需要修改的变量值传递给PLC。
2. 创建全局的委托，这样以来就可以调用Montior窗体，添加日志的方法，让添加日志的方法变为一个通用的方法。在Montior中创建添加日志的方法，在其中跨线程调用时，判断this.lst\_Loginfo.InvokeReruired是否跨线程访问，通过系统的多线程this.lst\_LogInfo.invoke(new Action<int,string>(递归添加日志的方法),num,level)。这样一来就是一个通用的添加日志方法

# 多线程实现实时数据同时实现PLC通信重连

1. 首先是实时数据的读取，在Main主窗体中在窗体的Load绘制时，就通过Task.Run(new action(()=>{DecivCommunication(CommonModel.Decive)}),ConCellationtokenSource.token);在DeciveCommunication(Decive decive);方法中while（!ConCelLationtokenSource.IsCancellationRequested）通过判断是否请求，中断多线程，来保证一直读取PLC数据。
2. 在读取数据之前，需要判断是否其读取的存储区是输入线圈/输出线圈，还是输入寄存器/输出寄存器。
3. 输入/输出线圈，直接读取，需要知道寄存区的起始地址以及长度；读取数据为byte[]，判断输入/输出线圈的数据类型（可以不用判断因为只可能是DataType.Bool），将其变为（DataType）Enum.Parse（oftype（DataType）,byte[],true）最后variable.value=BitLib.GetBitfromByteArray(读取出来的数据，起始地址，偏移量)，线圈变量实际地址=变量地址-通信组地址（储存区地址）
4. 对于输入/输出寄存器，读取需要读取寄存区的起始地址以及长度，不过对于长度需要乘2；需要判断寄存器中的变量是何类型，对应不同的DataType需要采取不同的方法解析读取出来的byte[]数据，bool，short，ushort，int，uint，double，float，long，ulong，都是采取大端读取，对于string，Hexstring，bytearray，不用采取大小端，需要知道其长度
5. 对于变量的数据改变，因为采用的Modbus Slave，我们设定了比列系数，所以需要通过计算得到具体的值
6. 对于短线重连，其实要在公共对象CommonModel中创建一个Bool类型的标志位，这样来判断是否是第一次连接，再次重连多线程延时5秒
7. 在Decive中创建了对应的public object this[string key]索引，通过判断索引是否存在，存在读取其中对应变量的value值，不存在添加其变量，并且将其value值，添加进去

# 通用报警信息添加

1. 因为所调用的是新阁的Thing.DataConvert的数据转换的Bll，所以创建一个全局Observablection<string>,用来标记解除报警和报警，当其元素为0则解除报警，当每次添加一个之后，对应的也增加对应显示的报警。
2. 在Decive中创建AlarmTrigEvent事件，在Decive中也要创建判断变量触发报警事件，第二点在FrmMain主窗体中，读取变量数据的时候，其实已经进行了判断是否触发报警，触发报警就触发报警事件

# 对于实时数据二维曲线绘制

1. 首先我们利用了SeeSharpTools.JY.GUI.dll这个逻辑库，调用到工具箱中利用其中的StripChartX的控件
2. 对于控件的属性设计，第一点：首先将其X轴的数据显示设计为TimeStamp，this.StripChartX.XdataType=SeeSharpTools.JY.GUI.StripChartX.XAxisDataType.TimeStamp,这样X轴就是对应的获取当前本地时间作为数据点；this.StripChartX.XdataStyle= SeeSharpTools.JY.GUI.StripChartX.XAxisDataType.String,这样就需要同事设定X轴和Y轴，并且一一对应才会显示曲线,可以在程序中设定也可也直接在绘制UI面板层设定；第二点：设定TimeStampFormat的style显示格式：this.StripChartX.TimeStampFormat=”HH:MM:SS”;第三点：显示图列可见，也就是绘制的实时曲线对应的标注可见this.StripChartX.legendVisible=ture;默认也是ture;第四点：显示绘制多少点位，this.StripChartX.DispalyPoint=40000;第五点：限制Y轴的最大值和最小值this.StripChartX.Ydata.Maximum/Minimum=100/0;
3. 绘制曲线有两种方式，第一种是调用Plotsingle（list.Toarray（））这个方式是已经Xdata设定为TimeStamp,并且Y轴设定为this.chart\_ActualTrend.Series[i].YPlotAxis=SeeSharpTools.JY.GUI.StripChartXAxis.PlotAxis.Primary;第二种，就是设定XdataStyle=String,并且在给Ydata赋值是需要设定一个Double[,]数组来对应其Ydata数据值的位置,J表示为某个曲线的Y值，I表示某个曲线YData[j, i] = Convert.ToDouble(dt.Rows[i][j+1]);每次设置完再设置Xdata的数据

# 在自定义控件中自定义事件

1、在自定义的控件中创建事件，首先在自定义控件的命名控件和自定义控件类间，声明[DefaultEvne(“ControlDoubleClick”)]对此需要让在自定义控件中所调用的控件，全都绑定此事件，在窗体中调用此控件的时候，在点击此自定义控件中其他控件时，才会自动生产此事件，

public event EventHandler ControlDoubleClick;

[Browsable(true)]

[Category("自定义事件")]

[Description("自定义双击事件")]