widgets挂件相关

0、挂件类型

目前挂件分为3种，每种有它自己的类，都继承自Widget（Widget继承自ChartObject）。

进度条 —— ProgressBar —— ProgressBarWidget

图标 —— Icon —— PictureWidget

备注 —— Remark —— NoteWidget

1、调用入口

点击上方菜单栏

```
//MindMapView.UI.cs
void TsbAddProgressBar_Click(object sender, EventArgs e)
{
    AddProgressBar();
}
void TsbAddIcon_Click(object sender, EventArgs e)
{
    AddIcon();
}
void TsbAddRemark_Click(object sender, EventArgs e)
{
    AddRemark();
}
```

选中topic右键菜单

```
//MindMapChartPage.cs

void MenuAddIcon_Click(object sender, EventArgs e)
{
    mindMapView1.AddIcon();
}
void MenuAddProgressBar_Click(object sender, EventArgs e)
{
    mindMapView1.AddProgressBar();
}
void MenuAddRemark_Click(object sender, EventArgs e)
{
    mindMapView1.AddRemark();
}
```

2、添加挂件

2.1添加方法

```
//MindMapView.Edit.cs
public void AddProgressBar()
{
     AddWidget(ProgressBarWidget.TypeID, new ProgressBarWidget(), true);
}

public void AddIcon()
{
    var dialog = new AddIconDialog();
    if (dialog.ShowDialog(this) == DialogResult.OK)
    {
        var template = new PictureWidget();
        template.Image = dialog.CurrentObject;
        AddWidget(PictureWidget.TypeID, template, dialog.NeedMoreOptions);
    }
}

public void AddRemark()
{
    var dialog = new NoteWidgetDialog();
    if (Clipboard.ContainsText())
        dialog.Remark = ClipboardHelper.GetHtml();

    if (dialog.ShowDialog(this) == DialogResult.OK)
    {
        var template = new NoteWidget();
        template.Remark = dialog.Remark;
        AddWidget(NoteWidget.TypeID, template, false);
    }
}
```

2.2AddWidget

三种挂件添加时都会调用AddWidget

typeID表示挂件种类（string类型）(如“PICTURE"\"NOTES"\"PROGRESSBAR")

template表示添加的具体挂件类

showDialog表示是否需要配置属性的弹窗

```
void AddWidget(string typeID, Widget template, bool showDialog)
{
    if (showDialog)
    {
        var dialog = new PropertyDialog();
        dialog.SelectedObject = template;
        if (dialog.ShowDialog(this) == System.Windows.Forms.DialogResult.OK)
        {
            AddWidgetCommand command = new AddWidgetCommand(SelectedTopics, typeID, template);
            ExecuteCommand(command);
        }
    }
    else
    {
        AddWidgetCommand command = new AddWidgetCommand(SelectedTopics, typeID, template);
        ExecuteCommand(command);
    }
}
```

2.2.1生成命令

```
public AddWidgetCommand(Topic[] topics, string widgetType, Widget template)
{
    Topics = topics;  //当前选中的topic
    WidgetType = widgetType;//挂件类型
    Template = template;//挂件
}
```

2.2.2执行命令

```
//AddWidgetCommand.cs
public override bool Execute()
{
    Widgets = new Widget[Topics.Length];
    for (int i = 0; i < Topics.Length; i++)  //这里循环是因为可以一次选中多个topic批量添加挂件
    {
        Topic topic = Topics[i];
        Widget widget = Widget.Create(WidgetType);
        Widgets[i] = widget;
        if (widget != null)
        {
            if (Template != null)
                Template.CopyTo(widget); //把模板挂件的属性赋值给new widget
            topic.Widgets.Add(widget);
            widget.OnAddByCommand(topic); //空方法
        }
    }
    return true;
}
```

2.3更新视图

InsertItem -> Widgets_ItemAdded -> OnPropertyChanged -> OnChartObjectPropertyChanged -> UpdateView（具体如何更新见第4章）

3、挂件属性

3.1共有属性（父类属性）

属性	用途	所属类	复用类
Text	文件名	ChartObject	PictureWidget
Remark	备注	ChartObject	PictureWidget、NoteWidget、ProgressBarWidget
DisplayIndex	序号	Widgets	PictureWidget、NoteWidget、ProgressBarWidget
Padding	边距	Widgets	PictureWidget、NoteWidget、ProgressBarWidget
Alignment	位置	Widgets	PictureWidget、NoteWidget、ProgressBarWidget
CustomWidth	自定义宽度	Widgets	PictureWidget、NoteWidget、ProgressBarWidget
CustomHeight	自定义高度	Widgets	PictureWidget、NoteWidget、ProgressBarWidget
Hyperlink	超链接	Widgets	PictureWidget、NoteWidget、ProgressBarWidget

3.2特有属性

3.2.1NoteWidget

属性	用途	使用类	备注
BackColor	背景色	NoteWidget	默认为空，且属性栏无法设置
ForeColor	前景色	NoteWidget	默认为空，且属性栏无法设置
TypeID	挂件类型	NoteWidget	固定为”NOTES“

3.2.2ProgressBarWidget

属性	用途	使用类	备注
Maximum	最大值	ProgressBarWidget	默认值100，且属性栏无法设置
Minimum	最小值	ProgressBarWidget	默认值0，且属性栏无法设置
Value	进度	ProgressBarWidget	Progress(%)，浮点型
ShowText	显示进度	ProgressBarWidget	bool类型
AutoCalculation	自动计算	ProgressBarWidget	bool类型，计算孩子节点进度的平均值
Color	进度条颜色	ProgressBarWidget	
BackColor	进度条背景色	ProgressBarWidget	
ForeColor	前景色	ProgressBarWidget	进度文字的颜色
TypeID	挂件类型	ProgressBarWidget	固定为”PROGRESSBAR“

3.2.3PictureWidget

属性	用途	使用类	备注
ImageUrl	图片完整路径	PictureWidget	若从图片库里导入则不需要完整路径
SizeType	图片尺寸类型	PictureWidget	thumb\original（实际）\customize（默认16*16）
OriginalSize	图片原始尺寸	PictureWidget	不可修改属性
EmbedIn	嵌入	PictureWidget	bool类型
ThumbImage	缩略图	PictureWidget	
Data	图片	PictureWidget	Image类型
TypeID	挂件类型	PictureWidget	固定为”PICTURE“

3.3属性持久化

属性保存到xml文档 Serialize(XmlDocument dom, XmlElement node)

```
 node.SetAttribute("image_url", ImageUrl);
```

xml文档加载属性 Deserialize(Version documentVersion, XmlElement node)

```
 ImageUrl = node.GetAttribute("image_url");
```

3.4修改属性更新视图

```
[DefaultValue(WidgetAlignment.Right)]
public virtual WidgetAlignment Alignment
{
    get { return _Alignment; }
    set
    {
        if (_Alignment != value)
        {
            WidgetAlignment old = _Alignment;
            _Alignment = value;
            OnPropertyChanged("Alignment", old, _Alignment, ChangeTypes.All);
        }
    }
}
```

例如修改Alignment属性，触发OnPropertyChanged，最后触发UpdateView

4、挂件定位及视图更新

整个定位及更新逻辑：

（1）触发更新视图时，先会重新计算布局，然后Paint重绘。

（2）在布局时，计算挂件所占区域大小。

（3）计算挂件location（相对于node原点），同时给widget.Bounds赋值

在计算位置时，根据orderby DisplayIndex的结果遍历

（4）布局确定后开始Paint，遍历topic的每个widget，调用具体widget Paint方法画图。

4.1挂件大小及定位

在布局计算topic大小的时候，同时计算了挂件的大小和位置

XXXLayout.cs Layout -> CalculateSize -> CalculateNodeSize

↓

Layouter.cs CalculateWidgets(topic, WidgetAlignment.Left, e)

```
protected virtual Size CalculateNodeSize(Topic topic, MindMapLayoutArgs e)
{
    //...
    // Widgets Size
    var rectLeft = CalculateWidgets(topic, WidgetAlignment.Left, e);
    var rectTop = CalculateWidgets(topic, WidgetAlignment.Top, e);
    var rectRight = CalculateWidgets(topic, WidgetAlignment.Right, e);
    var rectBottom = CalculateWidgets(topic, WidgetAlignment.Bottom, e);
    int maxWidth = Helper.GetMax(rectText.Width, rectTop.Width, rectBottom.Width) +
rectLeft.Width + rectRight.Width;

    int totalHeight = Helper.GetMax(rectText.Height + rectTop.Height + rectBottom.Height,
rectLeft.Height, rectRight.Height);
    rectText.Width = Helper.GetMax(rectText.Width, rectTop.Width, rectBottom.Width);
    /*
    每块挂件区域大小如何计算，见4.1.1
    -------------------------
    |    |    Top      |    |
    |  L | ----------- | R  |
    |    |    Text     |    |
    |    | ----------- |    |
    |    |   Bottom    |    |
    -------------------------
    最大宽度 = max(top,text,bottom) + l + r
    最大高度 = max(l,r, top+text+bottom )
    */

    //...
    // Widgets Location
    int hh1 = rectText.Height + rectTop.Height + rectBottom.Height;
    if (hh1 < rect.Height)
    {
        int hh = (rect.Height - hh1) / 3;
        rectText.Height += hh;
        rectTop.Height += hh;
        rectBottom.Height += hh;
    }
    //上中下区域高度总和小于 总区域高度，那么上中下区域高度分别加一点

    if (!rectText.IsEmpty)
    {
        rectText = new Rectangle(
                   rect.X + rectLeft.Width,
                   rect.Y + rectTop.Height,
                   rect.Width - rectLeft.Width - rectRight.Width,
                   rect.Height - rectTop.Height - rectBottom.Height);
    }
    rectLeft.Height = rectRight.Height = Helper.GetMax(rectLeft.Height, rectRight.Height,
rectText.Height + rectTop.Height + rectBottom.Height);
    rectLeft.Y = rectRight.Y = rect.Y;
    rectLeft.X = rect.X;
    rectRight.X = rectLeft.Right + rectText.Width;
    rectTop.X = rectBottom.X = rectText.Left;
    rectTop.Width = rectBottom.Width = rectText.Width;
    rectTop.Y = rect.Y;
    rectBottom.Y = rectText.Bottom;
    topic.TextBounds = rectText;
    ResetWidgetLocation(e, topic, WidgetAlignment.Left, rectLeft);
    ResetWidgetLocation(e, topic, WidgetAlignment.Top, rectTop);
    ResetWidgetLocation(e, topic, WidgetAlignment.Right, rectRight);
    ResetWidgetLocation(e, topic, WidgetAlignment.Bottom, rectBottom);
}
```

4.1.1挂件区域大小

```
Rectangle CalculateWidgets(Topic topic, WidgetAlignment alignment, MindMapLayoutArgs e)
{
    var widgets = topic.FindWidgets(alignment);
    //挂件索引位置，见4.3

    if (widgets.Length == 0)
        return Rectangle.Empty;

    var rect = Rectangle.Empty;
    var fitSize = Size.Empty;
    foreach (var widget in widgets)
    {
        var rectW = new Rectangle(Point.Empty, widget.CalculateSize(e));
        //widget.CalculateSize对于noteWidget来说是Size(16, 16)
        rectW.Inflate(widget.Padding, widget.Padding);
        if (widget.CustomWidth.HasValue)
            rectW.Width = widget.CustomWidth.Value;
        if (widget.CustomHeight.HasValue)
            rectW.Height = widget.CustomHeight.Value;

        widget.Bounds = rectW;
        //这里对widget.Bounds赋值， (x,y) 为 (0, 0) , 大小为(初始大小+padding)或自定义大小

        rectW.Width += e.Chart.WidgetMargin;
        rectW.Height += e.Chart.WidgetMargin;

        switch (alignment)
        {
            case WidgetAlignment.Left:
            case WidgetAlignment.Right:
                    /*
                    在Widget.cs里，默认为false且不可修改
                    [Browsable(false)]
                     public virtual bool FitContainer
                     {
                         get { return false; }
                     }

                    为false表示 :
                    对于左侧挂件来说，挤在一竖条排布，而不是横向排布。挂件是上下位置分布。
                    左侧区域宽 = max(左侧挂件中最大宽度)
                    左侧区域高 = 左侧挂件高度和

                    为true表示:
                    对于左侧挂件来说，横向排布，挂件是左右关系。
                    左侧区域宽 = 左侧挂件宽度和
                    左侧区域高 = max(左侧挂件中最大高度)
                    */

                if (widget.FitContainer)
                {
                    fitSize.Width += rectW.Width;

                    fitSize.Height = Math.Max(fitSize.Height, rectW.Height);
                }
                else
                {
                    rect.Width = Math.Max(rect.Width, rectW.Width);
                    rect.Height += rectW.Height;
                }
                break;
            case WidgetAlignment.Top:
            case WidgetAlignment.Bottom:
                if (widget.FitContainer)
                {
                    fitSize.Height += rectW.Height;
                    fitSize.Width = Math.Max(fitSize.Width, rectW.Width);
                }
                else
                {
                    rect.Width += rectW.Width;
                    rect.Height = Math.Max(rect.Height, rectW.Height);
                }
                break;
        }
    }

    switch (alignment)
    {
        case WidgetAlignment.Left:
        case WidgetAlignment.Right:
            rect.Width += fitSize.Width;
            rect.Height = Math.Max(rect.Height, fitSize.Height);
            break;
        case WidgetAlignment.Top:
        case WidgetAlignment.Bottom:
            rect.Width = Math.Max(rect.Width, fitSize.Width);
            rect.Height += fitSize.Height;
            break;
    }

    rect.Width += e.Chart.WidgetMargin;
    rect.Height += e.Chart.WidgetMargin;

    return rect;
}
```

4.1.2挂件定位

```
void ResetWidgetLocation(MindMapLayoutArgs e, Topic topic, WidgetAlignment alignment, Rectangle
rect)
{
    var widgets = topic.FindWidgets(alignment);
    if (widgets.Length == 0)
        return;

    //按照c2的思路左侧挂件横向排布, w.FitContainer均为true,dynamicWidgets为空
    var dynamicWidgets = widgets.Where(w => !w.FitContainer).ToArray();
    if (alignment == WidgetAlignment.Left || alignment == WidgetAlignment.Right)
    {
        var totalHeight = dynamicWidgets.Sum(w => w.Bounds.Height);
        var maxWidth = dynamicWidgets.Length > 0 ? dynamicWidgets.Max(w => w.Bounds.Width) : 0;
        var widgetMargin = Math.Max(e.Chart.WidgetMargin, (rect.Height - totalHeight) /
(dynamicWidgets.Length + 1));

        int y = rect.Y + widgetMargin;
        int x = rect.X + e.Chart.WidgetMargin;
        int dx = -1;
        int dy = y;
        foreach (var w in widgets)
        {
            var rw = w.Bounds;

            if (w.FitContainer)
            {
                rw.X = x;
                rw.Y = rect.Y + e.Chart.WidgetMargin;
                rw.Height = rect.Height - e.Chart.WidgetMargin * 2;
                x += rw.Width + e.Chart.WidgetMargin;
            }
            else
            {
               //...
            }

            w.Bounds = rw;
            //挂件区域  (x,y)为基于node原点的偏移x,y, 宽度不变, 高度为总高度-2*margin
        }
    }
    else if (alignment == WidgetAlignment.Top || alignment == WidgetAlignment.Bottom)
    {
        //...
    }
}
```

4.2挂件视图更新

(目前只看remark，跟C2挂件样式相近)

//GeneralRender.cs

```
void _PaintNode(Topic topic, RenderArgs e)
{
    //...
    // widgets
    foreach(var widget in topic.Widgets)
    {
        widget.Paint(e);
        if (widget.Selectable && widget.Selected)
        {
            DrawSelectRectangle(e.Graphics, e.Chart.SelectColor, widget.Bounds);
        }
    }

    //...
}
```

```
//NoteWidget.cs
public override void Paint(RenderArgs e)
{
    if (this.Hover)//鼠标是否悬浮，悬浮会多画一个区域
    {
        PaintHelper.DrawHoverBackground(e.Graphics, Bounds,
           (Container is Topic) ? ((Topic)Container).RealBackColor : SystemColors.Highlight);
    }

    Rectangle rect = DisplayRectangle;
    /*
    【补充】
    protected Rectangle DisplayRectangle
    {
        get
        {
            var rect = Bounds;
            rect.Inflate(-Padding, -Padding);
            return rect;
        }
    }
    */

    Image iconRemark = Properties.Resources.note_small;
    rect.X += Math.Max(0, (rect.Width - iconRemark.Width) / 2);
    rect.Y += Math.Max(0, (rect.Height - iconRemark.Height) / 2);
    rect.Width = Math.Min(rect.Width, iconRemark.Width);
    rect.Height = Math.Min(rect.Height, iconRemark.Height);
    e.Graphics.DrawImage(iconRemark, rect, 0, 0, iconRemark.Width, iconRemark.Height);
}
```

4.3挂件索引位置

```
public Widget[] FindWidgets(WidgetAlignment alignment)
{
    return (from w in Widgets
            where w.Alignment == alignment
            orderby w.DisplayIndex
            select w).ToArray();
}
```

DisplayIndex初始值是0

5、选中挂件及右侧属性栏变化

Chart.cs ChartBox_MouseDown

MindMapView.Mouse.cs OnChartMouseDown

 PressObject = HitTest(e.X, e.Y); //找到选中的挂件

 Select(PressObject.Widget, !Helper.TestModifierKeys(Keys.Control));

...

DocumentForm.cs SelectedObjects.set

 OnSelectedObjectsChanged

 ShowProperty