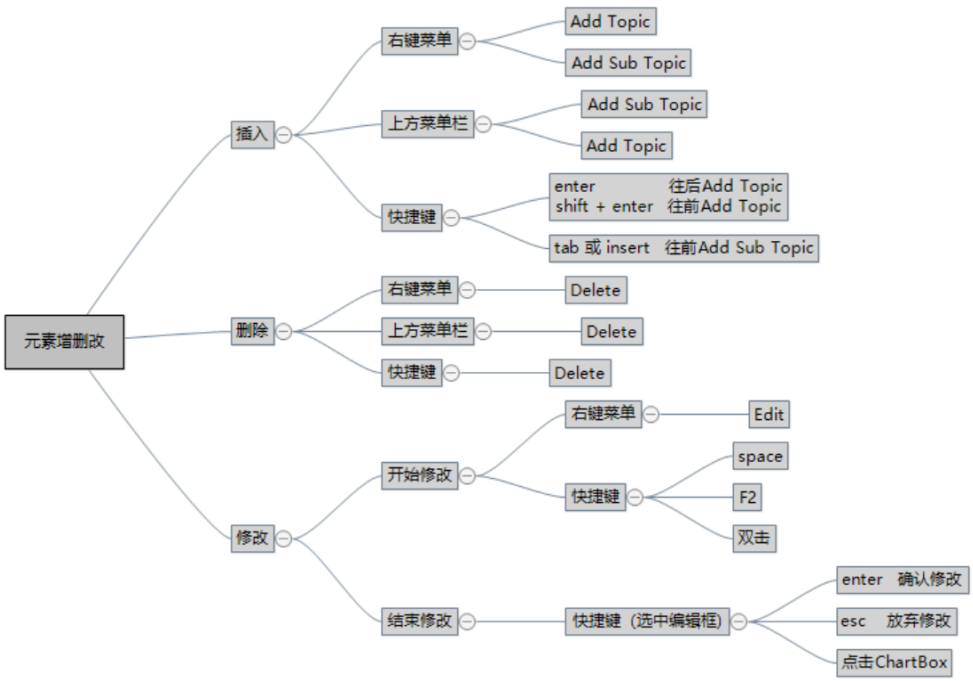
元素插入、删除、修改

1、代码入口

有三种方式会引起元素增删改：右键菜单、快捷键、菜单栏。

具体调用情况如下：



整体流程可分为：发起动作 -> 生成命令 -> 执行命令 -> 界面变化

2、插入

2.1 调用逻辑

入口操作	调用情况
上方菜单栏 add sub topic	TsbAddSubTopic_Click -> AddSubTopic
上方菜单栏 add topic	TsbAddTopic_Click -> AddTopic -> AddSubTopic
右键菜单 add sub topic	MenuAddSubTopic_Click -> AddSubTopic
右键菜单 add topic	MenuAddTopic_Click -> AddTopic -> AddSubTopic
快捷键 enter	ChartBox_KeyDown -> OnChartKeyDown -> ShortcutKeys.Haldle -> ShortcutKeys.Register -> AddTopic -> AddSubTopic
快捷键 shift + enter	ChartBox_KeyDown -> OnChartKeyDown -> ShortcutKeys.Haldle -> ShortcutKeys.Register -> AddTopicFront -> AddTopic -> AddSubTopic
快捷键 tab	ChartBox_KeyDown -> OnChartKeyDown -> ShortcutKeys.Haldle -> ShortcutKeys.Register -> AddSubTopic
快捷键 insert	ChartBox_KeyDown -> OnChartKeyDown -> ShortcutKeys.Haldle -> ShortcutKeys.Register -> AddSubTopic

（1）添加topic，即新增它的兄弟节点，等同于**新增一个它的父节点的孩子节点。**

有一个例外，是根节点不存在兄弟节点，所以对于根节点，等同于**新增根节点的孩子节点。**

【这块逻辑后续可能会改，因为目前blumind只允许有一个根】

（2）添加subtopic，**新增它的孩子节点。**

所以根据不同情况，最后归结点都是调用AddSubTopic。

2.2 AddSubTopic

```
Topic AddSubTopic(Topic parentTopic, Topic reference, bool atFront)
{
    //根据atFront计算新增topic索引
    var index = -1;
    if (reference != null)
        index = parentTopic.Children.IndexOf(reference);
    if (!atFront)
        index++;

    Topic subTopic = new Topic(GetNewSubTopicText(parentTopic));//根据索引命名新topic
    AddTopic(parentTopic, subTopic, index);//执行操作，见2.3
    Select(subTopic);
    return subTopic;

}
```

parentTopic为父topic，atFront表示新增topic是否在reference的前面

根据reference计算新增topic在父topic的index值，atFront决定是在reference前还是后

new Topic，根据父topic的孩子个数+1来初始化新增topic的名字

====理解====

就是为了找到新增topic放在哪个topic下的哪个位置

选中节点（非根）

添加sub topic，新增topic放在自己下面的首位；（parentTopic为自己,reference为null,atFront为true）

添加topic，新增topic放在自己前后；（parentTopic为自己父节点,reference为自己）

选中节点（根）

添加sub topic，新增topic放在自己下面的首位；（parentTopic为自己,reference为null,atFront为true）

添加topic，新增topic放在自己下面的首位；（parentTopic为自己,reference为null,atFront为true）

========

2.3 AddTopic

```
//MindMapView.Command.cs
public void AddTopic(Topic parentTopic, Topic subTopic, int index)
{
    var command = new AddTopicCommand(parentTopic, subTopic, index);   //生成命令
    ExecuteCommand(command);                                           //执行命令
}
```

【跳到第5章补充内容】

2.3.1 生成命令

```
//MindMap/Commands/AddTopicCommand.cs
public AddTopicCommand(Topic parentTopic, Topic subTopic, int index)
{
    ...
    ParentTopic = parentTopic;
    SubTopics = new Topic[] { subTopic };
    Index = index;
}
```

2.3.2 执行命令

对于插入来说，具体执行命令如下：

```
//MindMap/Commands/AddTopicCommand.cs
public override bool Execute()
{
    if (Index >= 0 && Index < ParentTopic.Children.Count)
```

```
    {
        var index = Index;
        foreach (var st in SubTopics)
        {
            ParentTopic.Children.Insert(index, st);
            index++;
        }
    }
    else
    {
        foreach (var st in SubTopics)
        {
            ParentTopic.Children.Add(st);
        }
    }
    return true;
}
```

遍历插入到指定索引位。

2.3.3回滚命令

```
//MindMap/Commands/AddTopicCommand.cs
public override bool Rollback()
{
    foreach (var st in SubTopics)
    {
       if (ParentTopic.Children.Contains(st))
       {
            ParentTopic.Children.Remove(st);
       }
    }
}
```

撤回时，将节点从孩子节点集合中移除。

2.4引起视图变化

ItemAdded -> Children_ItemAdded -> OnTopicAdded -> UpdateView(ChangeTypes.All);

3、删除

3.1 调用逻辑

入口操作	调用情况
右键菜单 delete	MenuDelete_Click -> DeleteObject -> Delete
上方菜单栏 delete	TsbAddSubTopic_Click -> DeleteObject -> Delete
快捷键 delete	ChartBox_KeyDown -> OnChartKeyDown -> ShortcutKeys.Haldle -> ShortcutKeys.Register -> DeleteObject -> Delete

3.2 Delete

参数mapObjects是什么？可以是link、topic、widget

```
//MindMapView.Command.cs
private void Delete(ChartObject[] mapObjects)
{
    if (mapObjects != null && mapObjects.Length > 0)
    {
        //根节点不能删
        foreach (ChartObject mapObject in mapObjects)
        {
            if (mapObject is Topic && ((Topic)mapObject).IsRoot)
            {
                return;
            }
        }

        DeleteCommand command = new DeleteCommand(mapObjects);//生成命令
        ExecuteCommand(command);//执行命令
    }
}
```

3.2.1生成命令

```
public DeleteCommand(ChartObject[] mapObjects)
{
    MapObjects = mapObjects;
    if (MapObjects == null || MapObjects.Length == 0)
    {
        throw new ArgumentNullException();
    }
}
```

3.2.2执行命令

```
public override bool Execute()
{
    Parents = new Dictionary<ChartObject, object>();
    Indices = new Dictionary<ChartObject, int>();
    return DeleteObjects(MapObjects, Parents, Indices);
}
```

```
public static bool DeleteObjects(ChartObject[] mapObjects,
        Dictionary<ChartObject, object> parents, Dictionary<ChartObject, int> indices)
{
    for (int i = 0; i < mapObjects.Length; i++)
    {
        ChartObject mo = mapObjects[i];
        if (mo is Link)
        {
            Link line = (Link)mo;
            parents[mo] = line.From;
            line.From.Links.Remove(line);
        }
        else if (mo is Topic)
        {
            Topic topic = (Topic)mapObjects[i];
            parents[mo] = topic.ParentTopic;
            indices[mo] = topic.Index;
            topic.ParentTopic.Children.Remove(topic);
        }
        else if (mo is Widget)
        {
            Widget widget = (Widget)mapObjects[i];
            parents[mo] = widget.WidgetContainer;
            indices[mo] = widget.WidgetContainer.IndexOf(widget);
            widget.WidgetContainer.Remove(widget);
        }
    }
}
```

mapObjects的类型可以是link\topic\widget，因此删除也分为这三种情况。

如果删除的是link，先找到link的起始topic，把它加入parent字典，同时把它的该条link移除。

如果删除的是topic，先找到topic的父topic，把它加入parent，把索引加入indices，同时把它的该孩子移除。

如果删除的是widget，先找到widget对应的IWidgetContainer，同样的方式移除。

存放这些删除object的parent\indices字典是为了撤回。

【删除topic，会删除它（作为起始点）包含的link、widget、lines、children】

【这里应该是有bug，删除link终点topic，link不消失】

3.2.3回滚命令

```
public override bool Rollback()
{
    return UndeleteObjects(MapObjects, Parents, Indices);
}
```

```
public static bool UndeleteObjects(ChartObject[] MapObjects, Dictionary<ChartObject, object>
Parents, Dictionary<ChartObject, int> Indices)
{
    for (int i = 0; i < MapObjects.Length; i++)
    {
        ChartObject mo = MapObjects[i];
        object parent = null;
        int index = -1;
        Parents.TryGetValue(mo, out parent);
        Indices.TryGetValue(mo, out index);

        if (mo is Topic)
        {
            Topic topic = (Topic)mo;
            Topic parentTopic = (Topic)parent;
            if (index > -1 && index < parentTopic.Children.Count)
                parentTopic.Children.Insert(index, topic);
            else
                parentTopic.Children.Add(topic);
        }
        else if (mo is Link)
        {
            Topic parentTopic = (Topic)parent;
            parentTopic.Links.Add((Link)mo);
        }
        else if (mo is Widget)
        {
            var container = (IWidgetContainer)parent;
            if (index > -1 && index < container.WidgetsCount)
                container.Insert(index, (Widget)mo);
            else
                container.Add((Widget)mo);
        }
    }
}
```

3.3引起视图变化

删除topic:

RemoveItem -> Children_ItemRemoved -> OnTopicRemoved -> Map_TopicRemoved -> OnTopicRemoved -> UpdateView(ChangeTypes.Layout);

删除widget:

RemoveItem -> Widgets_ItemRemoved -> OnWidgetRemoved ->

-> Map_WidgetRemoved -> Unselect -> EndUpdateView -> UpdateView(ChangeTypes.Visual);(+1-1更新样式)

-> UpdateView(ChangeTypes.All);（更新所有）

删除link:

RemoveItem -> Links_ItemRemoved -> Link.From.set -> OnFromChanged -> OnLinkVisibleChanged -> Map_LinkVisibleChanged -> UpdateView(ChangeTypes.All);

4、修改

4.1调用逻辑

修改分为两步：开始修改、结束修改，调用逻辑如下。

入口操作	场景	调用情况
右键菜单 edit	开始修改	MenuEdit_Click -> EditObject
快捷键	开始修改	ChartBox_KeyDown -> OnChartKeyDown -> ShortcutKeys.Haldle -> ShortcutKeys.Register -> EditObject
快捷键 F2	开始修改	ChartBox_KeyDown -> OnChartKeyDown -> ShortcutKeys.Haldle -> ShortcutKeys.Register -> EditObject
快捷键 双击	开始修改	ChartBox_DoubleClick -> OnChartDoubleClick -> EditObject

入口操作	场景	调用情况
快捷键 enter	结束修改	InternalTextBox_KeyDown -> ApplyEdit -> EditControl_Apply -> EndEdit(true)
快捷键 esc	结束修改	InternalTextBox_KeyDown -> CancelEdit -> EditControl_Cancel -> EndEdit(false)
快捷键 ChartBox点击	结束修改	ChartBox_MouseDown -> OnChartMouseDown -> EndEdit(true)

4.2开始修改

必须选中ITextObject，才能修改（目前已知能修改的是topic文字，link文字）

新增了一个EditControl控件（文本框+外层可拖拽框的组合），是MindMapView里的控件，置于顶层

```
public virtual void EditObject(ITextObject tobj)
{
    if (EditControl == null)
        EditControl = CreateEditControl();  //MindMapView里的控件EditControl
    EditControl.BringToFront();
```

```
    EditingObject = tobj;
    EditControl.Text = tobj.Text;  //EditControl 的文本

    //重置 EditControl 位置及大小
    //chartbox字体大小  和  选中文本字体大小   比较后决定了编辑框的大小
    ResetEditControlBounds(tobj);

    Font font;
    if(tobj.Font != null)
        font = tobj.Font;
    else
        font = ChartBox.Font;
    if (Zoom != 1.0f)
        font = new Font(font.FontFamily, font.Size * Zoom, font.Style);
    EditControl.Font = font;  //EditControl的字体

    EditMode = true;//EditMode的值改变，触发OnEditModeChanged
}
```

```
public bool EditMode
{
    get { return _EditMode; }
    private set
    {
        if (_EditMode != value)
        {
            _EditMode = value;
            OnEditModeChanged();
        }
    }
}
```

```
void OnEditModeChanged()
{
    //EditMode为true，EditControl控件显示并设为焦点
    if (EditMode)
    {
        if (EditControl != null)
        {
            EditControl.Show();
            EditControl.Focus();
        }
    }
    //EditMode为false，EditControl控件隐藏，MindMapView设为焦点
    else
    {
        if (EditControl != null && EditControl.Visible)
        {
            EditControl.Hide();
        }


        EditingObject = null;
```

```
        if (this.CanFocus)
            this.Focus();
    }
}
```

4.3结束修改

从4.1调用逻辑可以看到，不同入口EndEdit传参不同。true:表示接收修改，即要修改文本内容；false:放弃修改。

```
protected virtual void EndEdit(bool acceptChange)
{
    //...
    if (acceptChange)
    {
        if (EditingObject.Text != EditControl.Text)
        {
            ChangeObjectText(EditingObject, EditControl.Text);
        }
    }

    EditMode = false;//EditMode的值改变，触发OnEditModeChanged
}
```

4.4修改文本命令

```
//MindMapView.Command.cs
public void ChangeObjectText(ITextObject tobj, string newText)
{
    ChangeTextCommand command = new ChangeTextCommand(tobj, newText);
    ExecuteCommand(command);
}
```

4.4.1生成命令

```
//ChangeTextCommand.cs
public ChangeTextCommand(ITextObject tobj, string newText)
{
    TheObject = tobj;
    NewText = newText;
}
```

4.4.2执行命令

```
public override bool Execute()
{
    OldText = TheObject.Text;//需要记录旧文本内容，便于撤回
    TheObject.Text = NewText; //set Text的值改变，触发OnTextChanged
    return true;

}
```

4.4.3回滚命令

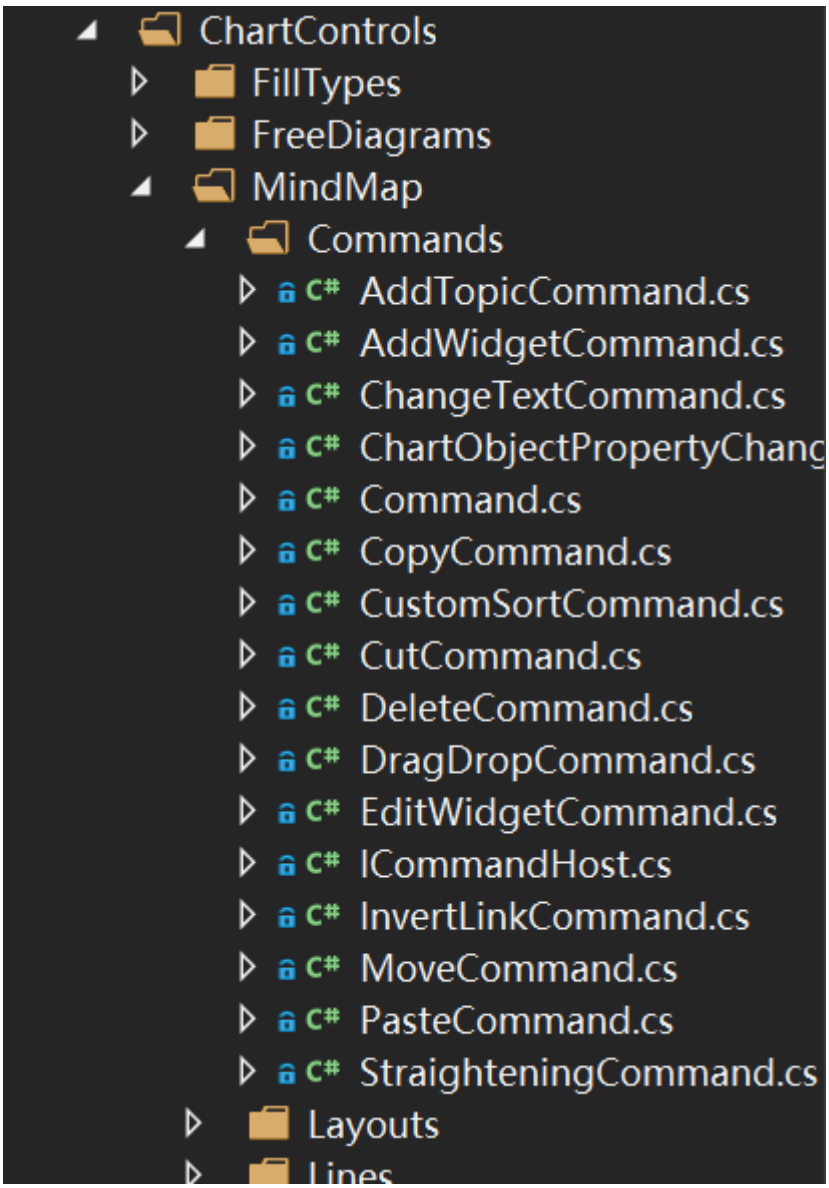
```
public override bool Rollback()
{
    TheObject.Text = OldText;
    return true;
}
```

4.5引起视图变化

Text.set -> OnTextChanged -> OnPropertyChanged -> ChartObjectPropertyChanged -> Map_ChartObjectPropertyChanged -> UpdateView(ChangeTypes.All)

5、补充

5.1命令位置



方法名	调用底层命令类	用途
CustomSort	CustomSortCommand	改变选中节点的孩子节点顺序
ChangeObjectText	ChangeTextCommand	改变文本
AddTopic/AddTopics	AddTopicCommand	添加节点（一个/多个）
DargDropTo	DragDropCommand	拖拽
Paste	PasteCommand	粘贴
Cut	CutCommand	剪切（topic、widgets）
Copy	CopyCommand	复制（topic、widgets）
Delete	DeleteCommand	删除
AddWidget	AddWidgetCommand	添加挂件
。。。	。。。	。。。

5.2命令内容

这些命令都是为undo、redo服务的，一条命令主要包括：

（1）操作需要的成员属性，如topic、link、index等；

（2）构造函数（生成命令，不同操作对实例命令的赋值不同）；

（3）执行命令方法 Execute

（4）回滚命令方法 Rollback

5.3如何执行

调用ChartControl.cs中的ExecuteCommand(command)执行该命令。

```
//ChartControls/ChartControl.cs
public bool ExecuteCommand(Command command)
{
    ...
    if (command.Execute())
    {
        if (command.NoteHistory) //copycommand重写为false，其他命令都为true，即复制命令不需要入栈
        {
            CommandHistory.Push(command);
            UndoCommandHistory.Clear();
            OnCommandHistoryChanged();
        }
        ...
    }
    ...
```

```
}
```

执行命令，除了copy操作，其他命令压入栈CommandHistory，并清空栈UndoCommandHistory（这个栈可以理解成redo内容，即有新命令后，redo清空，无法执行恢复功能）。