**网络传销会员层级分析系统**

**程序核心源代码**

|  |  |
| --- | --- |
| 开发单位 | 湖南警察学院 |
| 需求设计 | 赵薇，张悦 |
| 程序开发 | 陈俊涛 |
| 开发时间 | 2017.7 - 2017.12 |
| 文档撰写 | 陈俊涛 |
| 撰写时间 | 2017.12.12 |

# 目录

目录 2

1. 程序总体说明 3

1.1. 程序技术架构 3

1.2. 关于WPF 3

1.3. 关于EPPlus 3

1.4. 动态绘制树形结构 4

1.5. 代码行数统计 6

2. 程序代码结构说明 8

2.1. 代码项目模块说明 8

2.2. 文件后缀名说明 8

2.3. 源代码目录树说明 9

3. 主程序界面部分源代码 9

3.1. 主程序界面布局代码 9

3.2. 主程序界面后台业务处理逻辑代码 13

4. 树形结构构造算法模块 23

5. 动态绘制树形结构模块 35

5.1. 树容器结构代码 35

5.2. 树节点结构代码 44

5.3. 树控件绘制算法代码 49

5.4. 树控件操作处理逻辑代码 62

5.5. 线状树结构风格代码 65

5.6. 组织结构树视图风格代码 69

5.7. 自定义树控件界面布局代码 73

5.8. 自定义树控件后台处理逻辑代码 75

6. 读取Excel和csv数据源模块 83

6.1. 读取Excel文件 83

6.2. 读取CSV文件 85

7. 读取sqlserver数据源模块 87

7.1. 读取数据库界面布局代码 87

7.2. 连接到数据库业务处理逻辑代码 95

8. 打印、导出数据导出数据模块 102

8.1. 打印和导出图片 102

8.2. 导出数据到Excel或csv文件 103

9. 查找、数据处理等模块 111

9.1. 查找多个记录界面布局代码 111

9.2. 查找多个记录业务逻辑处理代码 112

9.3. 数据处理辅助类代码 115

# 程序总体说明

## 程序技术架构

* 开发语言：C# 5.0
* 开发工具：Visual Studio2015及以上版本，或者SharpDevelop5.0及以上版本
* 依赖软件环境：.NET Framework 4.0及以上版本（XP默认安装2.0，Win7默认安装3.5，Win8默认安装4.0，win10默认安装4.5，所以如果操作系统是Win8和win10，就不用额外安装.net framework，如果操作系统是XP或Win7，需要额外安装.net framework4.0）
* 硬件环境：由于采用了WPF绘图，所以最好独立显卡，由于程序处理的数据量可能超过20万行以上，所以内存最好4G以上。

## 关于WPF

最初程序技术选型的时候，考虑过使用Python语言，结合Tkinter来绘制界面，但是很快发现行不通，原因在于我们的程序需要处理的数据量超过20万行甚至更多，Python作为解释型语言在性能上无法胜任，后来考虑Windows平台上开发桌面程序最方便的C#。但是在选择界面绘制框架时，没有考虑以前的WinForm，而采用了WPF。

*WPF 为Windows Presentation Foundation的首字母缩写 ，中文译为“*[*Windows*](http://baike.baidu.com/view/4821.htm)*呈现基础”，其原来代号为“Avalon”，WPF是微软新一代*[*图形系统*](http://baike.baidu.com/view/7751478.htm)*，运行在.NET Framework 3.0及以上版本下，为用户界面、2D/3D 图形、文档和媒体提供了统一的描述和操作方法。基于DirectX 9/10技术的WPF不仅带来了前所未有的3D界面，而且其图形向量渲染引擎也大大改进了传统的2D界面，比如Vista中的半透明效果的*[*窗体*](http://baike.baidu.com/view/230361.htm)*等都得益于WPF。WPF是Windows操作系统中一次重大变革，与早期的GDI+/GDI不同。WPF是基于DirectX引擎的，支持GPU硬件加速，在不支持硬件加速时也可以使用软件绘制。高级别的线程进行绘制，提高使用者的体验。自动识别显示器分辨率并进行缩放。*

上面一段介绍来自百度百科中对于WPF的介绍，翻译成人话就是：以前的绘图技术都是使用CPU进行绘图，但是CPU除了绘图之外，还要干很多其他事情，这样CPU负担加重，绘图效率不尽人意。而WPF采用DirectX技术直接使用GPU显卡进行绘图，CPU发出绘图的指令，GPU显卡进行绘图，这样就大大提高了绘图效率和现实效果。

## 关于EPPlus

EPPlus主要用于读写Excel和Word文档。

由于我们的程序需要直接读取Excel数据作为数据源，所以需要操纵Excel数据的库，微软自带的也有，但是有个缺点就是需要目标机器上安装微软的Office程序，而Office程序已经1个多G了，并且需要收费，安装的话既占空间，还要进行破解，麻烦的很。

本人机器上安装的是免费的WPS，只有一百多M，而且免费。最后就采用了不依赖于Office程序的Excel库，网上有很多，但是最后选择了EPPlus，读取速度很快，而且小巧，使用简单方便。

以下介绍来自维基百科：

*Office Open XML（缩写：Open XML、OpenXML或OOXML），是由Microsoft开发的一种以XML为基础并以ZIP格式压缩的电子文件，支持Word、Excel、Office Note、PPT等文件格式。OOXML在2006年12月成为了ECMA规范的一部分，编号为ECMA-376；并于2008年4月国际标准化组织(ISO)的表決，在两个月公布为ISO／IEC 29500国际标准。从Microsoft Office 2007开始，Office Open XML文件格式已经成为Microsoft Office默认的文件格式。Microsoft Office 2010支持对ECMA-376标准文档的读操作，ISO/IEC 29500 Transitional的读/写，ISO/IEC 29500 Strict的读取。Microsoft Office 2013同时支持ISO/IEC 29500 Strict的读写操作。*

*EPPlus就是一个通过Open XML方式来读写Office文件的开源.NET类库，所以使用它生成Office文件完全不需要Microsoft Office（当然如果你需需要查看生成的文件就需要Office了）。它的官方网址是：*[*http://epplus.codeplex.com/*](http://epplus.codeplex.com/)*。如果需要体验本文中提到的效果，需要从这个网址下载最新版本的类库，我现在使用的这个名为EPPlus.dll类库仅658K，非常方便部署。*

## 动态绘制树形结构

这里的动态绘制是关键点。

因为我们程序读取了超过几十万条数据，如果同时将几十万条数据都加载在界面上，那么显然是不现实的：

首先，数据量太大，加速速度太慢；

再者，同时显示大量数据到内存中，内存显然不够用；

最后，数据全部都显示出来，不但屏幕放不下，而且用户肉眼也无法全部看完所有数据；

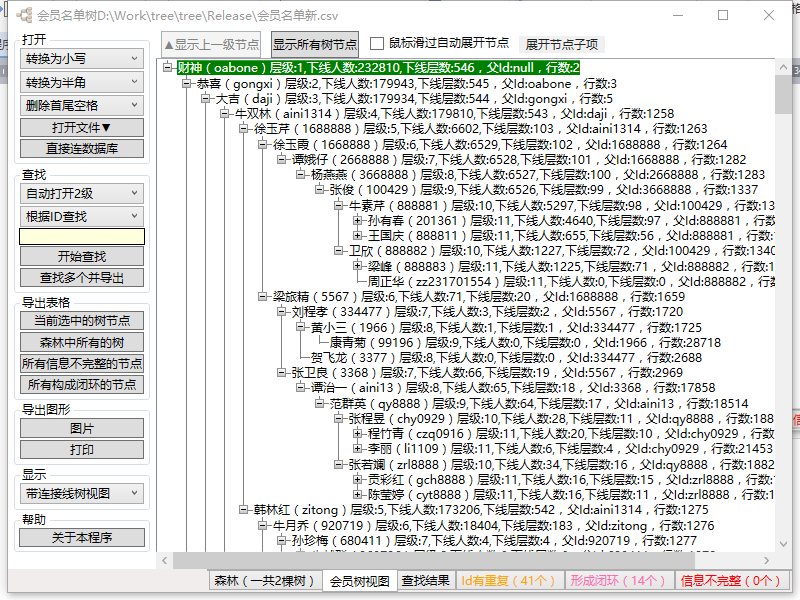
所以也没必要全部同时显示出来所有数据，我们只需要根据屏幕大小，动态加载当前呈现在屏幕中的节点即可，用户点开下一级节点，就临时再从数据中去找到相关的节点数据，临时呈现在屏幕上，用户关闭该节点，就从屏幕中销毁该节点，这样既达到了想要的效果，又不需要同时加载所有数据，对计算机内存等要求不必那么高。

最初找了很多风格的树形控件，但经过一系列排除（比如有点显示效果不好看或者不是用户想要的，有的不是动态加载的），最后保留了四种样式：

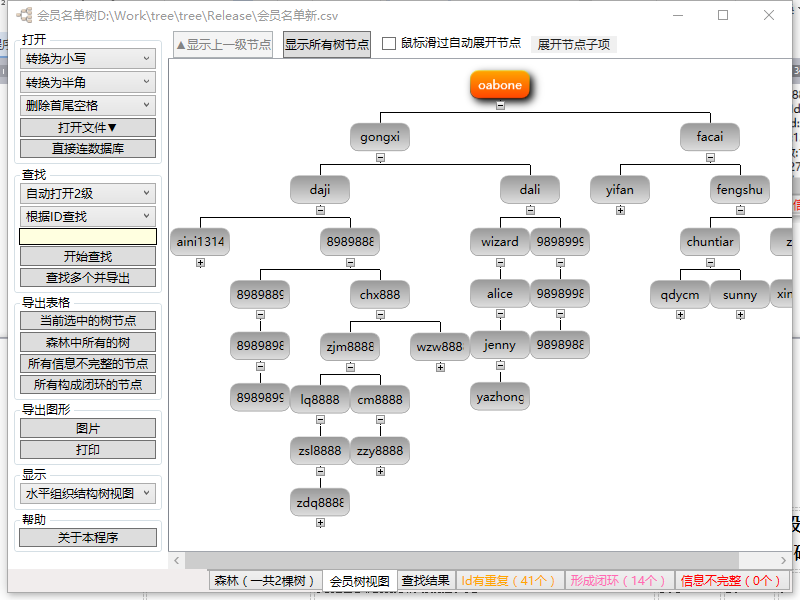
* **最简约风格：**

****

* **带连接线的风格**

****

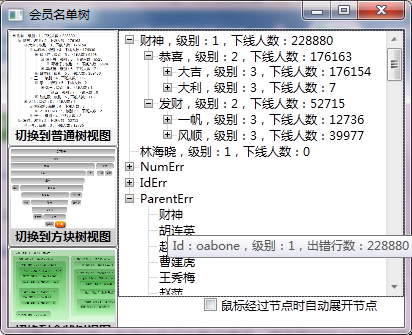
* **水平组织结构树状视图**



* **水平组织结构树状视图**

****

**下面这些是我们最初技术选型时的一些其他风格树：**

****

****

## 代码行数统计

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **项目名称** | **代码文件极其路径** | **代码行数** | **空行数** | **设计器代码行数** | **注释行数** |
| MemberTree\MemberTree.csproj | ConnDBWindow.xaml.cs | 308 | 19 | 0 | 19 |
| HelpWindow.xaml.cs | 27 | 1 | 0 | 3 |
| MyForest.xaml.cs | 42 | 3 | 0 | 11 |
| MyTreeView\BoxTreeStyle.xaml.cs | 54 | 5 | 0 | 12 |
| MyTreeView\LineTreeStyle.xaml.cs | 25 | 2 | 0 | 1 |
| SearchMultiInput.xaml.cs | 175 | 13 | 0 | 12 |
| TextUtilTool.cs | 100 | 5 | 0 | 20 |
| DraingTree.cs | 77 | 14 | 0 | 6 |
| Export.cs | 328 | 46 | 0 | 120 |
| MyTreeView\GraphTreeStyle.xaml.cs | 65 | 6 | 0 | 0 |
| TreeViewGraph\DPoint.cs | 22 | 2 | 0 | 3 |
| TreeViewGraph\ITreeNode.cs | 20 | 2 | 0 | 3 |
| TreeViewGraph\LayeredTreeDraw.cs | 550 | 67 | 0 | 30 |
| TreeViewGraph\TreeConnection.cs | 21 | 2 | 0 | 0 |
| TreeViewGraph\TreeContainer.cs | 374 | 39 | 0 | 4 |
| TreeViewGraph\TreeNode.cs | 209 | 20 | 0 | 0 |
| TreeViewGraph\TreeNodeGroup.cs | 61 | 13 | 0 | 0 |
| MyNodeList.xaml.cs | 32 | 2 | 0 | 3 |
| MyTreeNode.cs | 521 | 55 | 0 | 15 |
| MyTreeView\MyTreeView.xaml.cs | 334 | 33 | 0 | 20 |
| 副本 MyTreeNode.cs | 482 | 47 | 0 | 435 |
| App.xaml.cs | 15 | 1 | 0 | 3 |
| MainWindow.xaml.cs | 412 | 30 | 0 | 61 |
| TreeViewGraph\MyGraphView.xaml.cs | 145 | 12 | 0 | 10 |
| Properties\AssemblyInfo.cs | 55 | 8 | 0 | 27 |
| Properties\Resources.Designer.cs | 63 | 0 | 63 | 0 |
| Properties\Settings.Designer.cs | 26 | 0 | 26 | 0 |
| Visualizing\Visualizing.csproj | DotParser.cs | 104 | 10 | 0 | 0 |
| DotViewer.xaml.cs | 204 | 32 | 0 | 19 |
| GraphLoader.cs | 202 | 36 | 0 | 12 |
| GraphPaginator.cs | 110 | 14 | 0 | 3 |
| ToolTipController.cs | 79 | 10 | 0 | 6 |
| Properties\AssemblyInfo.cs | 64 | 12 | 0 | 31 |
| Properties\Resources.Designer.cs | 63 | 0 | 63 | 0 |
| Properties\Settings.Designer.cs | 26 | 0 | 26 | 0 |
| GraphElement.cs | 170 | 23 | 0 | 14 |
| ToolTipEvent.cs | 21 | 3 | 0 | 0 |
| **合计** | **代码文件数总数：37** | **5586** | **587** | **178** | **903** |

# 程序代码结构说明

## 代码项目模块说明

该项目包含两个工程，每个工程中包含若干个源代码文件、资源文件、界面设计文件等等。

其中第二个项目Visualizing\Visualizing.csproj是当时在网上找的一个绘制树形图形的控件，效果非常好看，但是实现逻辑太复杂，最后没有采用，但是作为一个参考代码保留在代码中，方便以后维护升级时参考。

只有第一个项目MemberTree\MemberTree.csproj才是实现我们程序功能的项目，该项目包含了以下几大模块：

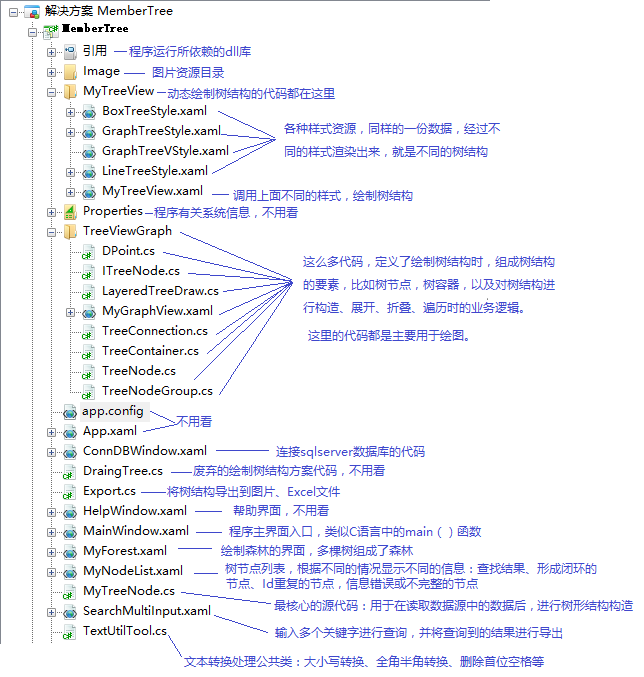
* **程序主界面模块**
* **树形结构构造算法模块**
* **动态绘制树形结构模块**
* **读取Excel和csv数据源模块**
* **读取sqlserver数据源模块**
* **打印、导出数据（到图片，到excel）模块**
* **查找、数据处理等模块**

## 文件后缀名说明

|  |  |
| --- | --- |
| 文件后缀名 | 文件说明 |
| \*.sln | 项目文件，双击这个文件可以使用VS打开真个项目进行编辑 |
| \*.suo | 自动生成，不管它 |
| \*.csproj | 工程文件，每个项目包含若干个工程，不管它 |
| \*.config | 自动生成，不管它 |
| **\*.cs** | **源代码文件，所有的自己手写的业务逻辑和算法代码都在这里** |
| **\*.xaml** | **界面文件，类似xml文件，但是主要描述程序图形界面的** |
| **\*.xaml.cs** | **界面逻辑处理代码文件，对与界面有关的业务逻辑进行处理** |

上述文件类型中，只有最后三个加粗的文件类型，可能是我们需要手工编写的。

## 源代码目录树说明



# 主程序界面部分源代码

## 主程序界面布局代码

<Window xmlns:my="clr-namespace:MemberTree"  x:Class="MemberTree.MainWindow"  
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"  
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"  
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"   
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"   
        Title="会员名单树" mc:Ignorable="d" Width="800" Height="600"   
        WindowStartupLocation="CenterScreen">  
    <Window.Resources>  
        <Style TargetType="Button">  
            <Setter Property="Margin" Value="1"></Setter>  
        </Style>  
        <Style TargetType="ComboBox">  
            <Setter Property="Margin" Value="1"></Setter>  
        </Style>  
    </Window.Resources>  
    <Grid Name="mainGrid">  
        <Grid.ColumnDefinitions>  
            <ColumnDefinition Width="Auto" />  
            <ColumnDefinition />  
        </Grid.ColumnDefinitions>  
        <Grid.RowDefinitions>  
            <RowDefinition />  
            <RowDefinition Height="auto"/>  
        </Grid.RowDefinitions>  
        <StackPanel Margin="5,0">  
            <GroupBox Header="打开">  
                <StackPanel>  
                    <ComboBox Name="comboToLower">  
                        <ComboBoxItem IsSelected="True">转换为小写</ComboBoxItem>  
                        <ComboBoxItem>转换为大写</ComboBoxItem>  
                        <ComboBoxItem>不转换大小写</ComboBoxItem>  
                    </ComboBox>  
                    <ComboBox Name="comboToHalf">  
                        <ComboBoxItem IsSelected="True">转换为半角</ComboBoxItem>  
                        <ComboBoxItem>转换为全角</ComboBoxItem>  
                        <ComboBoxItem>不转换全半角</ComboBoxItem>  
                    </ComboBox>  
                    <ComboBox Name="comboTrim">  
                        <ComboBoxItem IsSelected="True">删除首尾空格</ComboBoxItem>  
                        <ComboBoxItem>删除首部空格</ComboBoxItem>  
                        <ComboBoxItem>删除尾部空格</ComboBoxItem>  
                        <ComboBoxItem>不删除空格</ComboBoxItem>  
                    </ComboBox>  
                    <Button Content="打开文件▼" Click="ButtonOpen\_Click" />  
                    <Button Content="直接连数据库" Click="ButtonConnectDB\_Click" />  
                </StackPanel>  
            </GroupBox>  
            <GroupBox Header="查找">  
                <StackPanel>  
                    <ComboBox Name="comboOpenLevel">  
                        <ComboBoxItem Tag="0">自动打开0级</ComboBoxItem>  
                        <ComboBoxItem Tag="1">自动打开1级</ComboBoxItem>  
                        <ComboBoxItem Tag="2" IsSelected="True">自动打开2级</ComboBoxItem>  
                        <ComboBoxItem Tag="3">自动打开3级</ComboBoxItem>  
                        <ComboBoxItem Tag="5">自动打开5级</ComboBoxItem>  
                        <ComboBoxItem Tag="10">自动打开10级</ComboBoxItem>  
                    </ComboBox>  
                    <ComboBox Name="comboSearchType">  
                        <ComboBoxItem IsSelected="True">根据ID查找</ComboBoxItem>  
                        <ComboBoxItem>根据姓名查找</ComboBoxItem>  
                        <ComboBoxItem>根据级别查找</ComboBoxItem>  
                        <ComboBoxItem>根据下线人数查找</ComboBoxItem>  
                        <ComboBoxItem>根据下线层数查找</ComboBoxItem>  
                    </ComboBox>  
                    <TextBox Name="txtSearch" BorderBrush="Black" Background="LightYellow"/>  
                    <Button Content="开始查找" Click="ButtonSearch\_Click"/>  
                    <Button Content="查找多个并导出" Click="ButtonSearchMulti\_Click"/>  
                </StackPanel>  
            </GroupBox>  
              
            <GroupBox Header="导出表格">  
                <StackPanel>  
                    <Button Content="当前选中的树节点" Click="ButtonExportSelectedNode\_Click"/>  
                    <Button Content="森林中所有的树" Click="ButtonExportTrees\_Click"/>  
                    <Button Content="所有信息不完整的节点" Click="ButtonExportInfoErrNodes\_Click"/>  
                    <Button Content="所有构成闭环的节点" Click="ButtonExportRingNodes\_Click"/>  
                </StackPanel>  
            </GroupBox>  
            <GroupBox Header="导出图形">  
                <StackPanel>  
                    <Button Name="btnExportImg" Content="图片" Click="btnExportImg\_Click" />  
                    <Button Name="btnPrint" Content="打印" Click="btnPrint\_Click" />  
                </StackPanel>  
            </GroupBox>  
            <GroupBox Header="显示">  
                <StackPanel>  
                    <ComboBox x:Name="comboTreeStyle" SelectionChanged="TreeStyle\_SelectionChanged">  
                        <ComboBoxItem>最简风格树视图</ComboBoxItem>  
                        <ComboBoxItem IsSelected="True">带连接线树视图</ComboBoxItem>  
                        <ComboBoxItem>水平组织结构树视图</ComboBoxItem>  
                        <ComboBoxItem>垂直组织结构树视图</ComboBoxItem>  
                    </ComboBox>  
                </StackPanel>  
            </GroupBox>  
            <GroupBox Header="帮助">  
                <StackPanel>  
                    <ToggleButton Name="btnHelp" Content="关于本程序" />  
                    <Popup StaysOpen="False" Placement="Center"   
                           PlacementTarget="{Binding ElementName=myTreeView}"   
                           IsOpen="{Binding ElementName=btnHelp, Path=IsChecked}">  
                        <my:HelpWindow />  
                    </Popup>  
                </StackPanel>  
            </GroupBox>  
        </StackPanel>  
        <my:MyNodeList x:Name="listNodes" Grid.Column="1" MouseDoubleClick="NodeListView\_MouseDoubleClick" Visibility="Hidden" />  
        <my:MyTreeView x:Name="myTreeView" Grid.Column="1" Visibility="Hidden" />  
        <my:MyForest x:Name="forestView" Grid.Column="1" Visibility="Hidden"/>  
        <UniformGrid x:Name="commonView" Grid.Column="1"  Visibility="Hidden">  
            <Button>  
                <StackPanel>  
                    <Image Source="/MemberTree;component/Image/forest.jpg" Width="150" Height="150"/>  
                    <TextBlock>森林</TextBlock>  
                </StackPanel>  
            </Button>  
            <Button Grid.Column="1">  
                <StackPanel>  
                    <Image Source="/MemberTree;component/Image/ring.jpg" Width="150" Height="150"/>  
                    <TextBlock>构成闭环的节点</TextBlock>  
                </StackPanel>  
            </Button>  
            <Button Grid.Row="1">  
                <StackPanel>  
                    <Image Source="/MemberTree;component/Image/conflict.jpg" Width="150" Height="150"/>  
                    <TextBlock>ID有冲突的节点</TextBlock>  
                </StackPanel>  
            </Button>  
            <Button Grid.Row="1" Grid.Column="1">  
                <StackPanel>  
                    <Image Source="/MemberTree;component/Image/damage.jpg" Width="150" Height="150"/>  
                    <TextBlock>节点信息不完整的节点</TextBlock>  
                </StackPanel>  
            </Button>  
        </UniformGrid>  
        <StatusBar Grid.Row="1" Grid.ColumnSpan="2">  
            <StatusBarItem>  
                <TextBlock Name="statusMessage" Text="就绪" />  
            </StatusBarItem>  
            <StatusBarItem>  
                <ProgressBar Name="progressBar" Width="300" Height="20" Visibility="Collapsed" />  
            </StatusBarItem>  
            <StatusBarItem Name="statusView" HorizontalAlignment="Right" Padding="0" Visibility="Collapsed">  
                <TabControl  Name="tabView" Padding="0" TabStripPlacement="Bottom" SelectionChanged="TabView\_Changed">  
                    <TabItem Name="tabNoParent" Header="森林"/>  
                    <TabItem Name="tabTree" Header="会员树视图"/>  
                    <TabItem Name="tabFindResult" Header="查找结果"/>  
                    <TabItem Name="tabIdErr" Header="Id有重复的节点" Foreground="Orange" />  
                    <TabItem Name="tabRingErr" Header="形成闭环的节点" Foreground="HotPink"/>  
                    <TabItem Name="tabNodeInfoErr" Header="节点信息不完整的节点" Foreground="Red" />  
                </TabControl>  
            </StatusBarItem>  
        </StatusBar>  
    </Grid>  
</Window>

## 主程序界面后台业务处理逻辑代码

**namespace** MemberTree  
{  
    **public**interfaceINotify  
    {  
        void**SetProcessBarVisible**(**bool** visible);  
        void**SetProcessBarValue**(**int** step);  
        void**SetStatusMessage**(string message);  
    }  
    ///<**summary**>  
    /// MainWindow.xaml 的交互逻辑  
    ///<**/summary**>  
    **public**partialclassMainWindow : Window, INotify  
    {  
        **internal**staticINotify*notify* = **null**;  
  
        **private**List<MyTreeNode>*findResultNodes* = **new**List<MyTreeNode>();  
  
        **public**MainWindow()  
        {  
            **InitializeComponent**();  
            *notify* = **this**;  
        }  
  
        #**region** 左侧按钮事件  
        //打开文件  
        **private**void**ButtonOpen\_Click**(object sender, RoutedEventArgs e)  
        {  
            OpenFileDialog openfileDlg = **new**OpenFileDialog();  
            openfileDlg.InitialDirectory = System.Windows.Forms.Application.StartupPath;  
            openfileDlg.Title = "打开要作为会员树数据源的文件";  
            openfileDlg.Filter = "CSV文件|\*.csv|Excel2007文件|\*.xlsx";  
            **if** (openfileDlg.**ShowDialog**() == **true**)  
            {  
  
                **int** upperLower = *comboToLower*.SelectedIndex;  
                **int** DBCSBC = *comboToHalf*.SelectedIndex;  
                **int** trim = *comboTrim*.SelectedIndex;  
                **if** (openfileDlg.FileName.**EndsWith**(".csv"))  
                {  
                    MyTreeNode.**OpenCSVFile**(openfileDlg.FileName, upperLower, DBCSBC, trim);  
                }  
                **else**  
                {  
                    MyTreeNode.**OpenExcelFile**(openfileDlg.FileName, upperLower, DBCSBC, trim);  
                }  
                **this**.Title = "会员名单树" + openfileDlg.FileName;  
                *myTreeView*.**SetRootNode**(MyTreeNode.RootNode);  
                //myGraphView.InitMyTree();  
                //显示统计信息  
                *tabNoParent*.Header = "森林（一共" + MyTreeNode.NoParentNodes.Count + "棵树）";  
                *tabIdErr*.Header = "Id有重复（" + MyTreeNode.IdConflictNodes.Count + "个）";  
                *tabNodeInfoErr*.Header = "信息不完整（" + MyTreeNode.**GetNodeInfoErrCount**() + "个）";  
                *tabRingErr*.Header = "形成闭环（" + MyTreeNode.RingNodes.Count + "个）";  
            }  
        }  
         //直接连接数据库  
        **private**void**ButtonConnectDB\_Click**(object sender, RoutedEventArgs e)  
        {  
            ConnDBWindow connDB = **new**ConnDBWindow();  
            connDB.**ShowDialog**();  
        }  
  
        //查找  
        **private**void**ButtonSearch\_Click**(object sender, RoutedEventArgs e)  
        {  
            **if** (*txtSearch*.Text != "")  
            {  
                string searchType = (*comboSearchType*.SelectedItem **as**ComboBoxItem).Content.**ToString**();  
  
                **if** (searchType == "根据ID查找")  
                {  
                    *findResultNodes* = MyTreeNode.**FindNodeById**(*txtSearch*.Text);  
                }  
                **elseif** (searchType == "根据姓名查找")  
                {  
                    *findResultNodes* = MyTreeNode.**FindNodeByName**(*txtSearch*.Text);  
                }  
                **elseif** (searchType == "根据级别查找")  
                {  
                    *findResultNodes* = MyTreeNode.**FindNodeByLevel**(*txtSearch*.Text);  
                }  
                **elseif** (searchType == "根据下线人数查找")  
                {  
                    **int** txt2num = 0;  
                    **if** (**int**.**TryParse**(*txtSearch*.Text, **out** txt2num))  
                    {  
                        *findResultNodes* = MyTreeNode.**FindNodeByDescendantCount**(txt2num);  
                    }  
                }  
                **elseif** (searchType == "根据下线层数查找")  
                {  
                    **int** txt2num = 0;  
                    **if** (**int**.**TryParse**(*txtSearch*.Text, **out** txt2num))  
                    {  
                        *findResultNodes* = MyTreeNode.**FindNodeByDescendantLevels**(txt2num);  
                    }  
                }  
  
                *listNodes*.NodeListView.ItemsSource = *findResultNodes*;  
                *tabView*.SelectedItem = *tabFindResult*;  
            }  
            **else**  
            {  
                MessageBox.**Show**("输入不能为空！");  
            }  
        }  
          
         //查找多个并导出  
        **private**void**ButtonSearchMulti\_Click**(object sender, RoutedEventArgs e)  
        {  
            SearchMultiInput searchMultiWindow = **new**SearchMultiInput();  
            searchMultiWindow.**ShowDialog**();  
        }  
  
        #**region** 导出表格  
   
        //导出选中的树节点  
        **private**void**ButtonExportSelectedNode\_Click**(object sender, RoutedEventArgs e)  
        {  
            MyTreeNode node = *myTreeView*.**GetSelectedNode**();  
            **if** (node != **null**)  
            {  
                Export.**ExportNodes**(**new**List<MyTreeNode>{node});  
             }  
            **else**  
            {  
                MessageBox.**Show**("必须选中一个节点！");  
            }  
        }  
          
        //导出全部森林中的树  
        **private**void**ButtonExportTrees\_Click**(object sender, RoutedEventArgs e)  
        {  
            **if**(MyTreeNode.NoParentNodes.Count >0)  
            {  
                Export.**ExportNodes**(MyTreeNode.NoParentNodes);  
            }  
            **else**  
            {  
                MessageBox.**Show**("森林中没有树！");  
            }  
        }  
          
        //导出信息不完整的节点  
        **private**void**ButtonExportInfoErrNodes\_Click**(object sender, RoutedEventArgs e)  
        {  
            **if**(MyTreeNode.NodeInfoErrNodes.Count >0)  
            {  
                Export.**ExportNodes**(MyTreeNode.NodeInfoErrNodes);  
            }  
            **else**  
            {  
                MessageBox.**Show**("没有信息不完整的节点！");  
            }  
        }  
  
        //导出全部闭环中的节点  
        **private**void**ButtonExportRingNodes\_Click**(object sender, RoutedEventArgs e)  
        {  
            **if**(MyTreeNode.RingNodes.Count >0)  
            {  
                Export.**ExportNodes**(MyTreeNode.RingNodes.Values.**ToList**());  
            }  
            **else**  
            {  
                MessageBox.**Show**("没有形成闭环的节点！");  
            }  
        }          
                  
        #**endregion**  
  
        //导出图片  
        **private**void**btnExportImg\_Click**(object sender, RoutedEventArgs e)  
        {  
            SaveFileDialog openfileDlg = **new**SaveFileDialog();  
            openfileDlg.InitialDirectory = System.Windows.Forms.Application.StartupPath;  
            openfileDlg.Title = "选择将会员树导出为文件的位置";  
            openfileDlg.Filter = "png格式|\*.png";  
            **if** (openfileDlg.**ShowDialog**() == **true**)  
            {  
                FileStream fs = **new**FileStream(openfileDlg.FileName, **FileMode**.*Create*);  
                **int** width = (**int**)*myTreeView*.*memberTreeView*.ActualWidth;  
                **int** height = (**int**)*myTreeView*.*memberTreeView*.ActualHeight;  
                RenderTargetBitmap bmp = **new**RenderTargetBitmap(width, height, 96, 96, PixelFormats.Default);  
                bmp.**Render**(*myTreeView*.*memberTreeView*);  
                BitmapEncoder encoder = **new**PngBitmapEncoder();  
                encoder.Frames.**Add**(BitmapFrame.**Create**(bmp));  
                encoder.**Save**(fs);  
                fs.**Close**();  
                fs.**Dispose**();  
            }  
        }  
  
        //打印  
        **private**void**btnPrint\_Click**(object sender, RoutedEventArgs e)  
        {  
            PrintDialog printDlg = **new**PrintDialog();  
            printDlg.UserPageRangeEnabled = **true**;  
  
            **if** (printDlg.**ShowDialog**() == **true**)  
            {  
                 printDlg.**PrintVisual**(*myTreeView*.*memberTreeView*, "打印当前会员树视图");  
            }  
        }  
          
        //切换树状视图的显示风格  
        **private**void**TreeStyle\_SelectionChanged**(object sender, SelectionChangedEventArgs e)  
        {  
            **if** (*myTreeView* != **null**)  
            {  
                string selectedContent = ((ComboBoxItem)*comboTreeStyle*.SelectedItem).Content.**ToString**();  
                **switch** (selectedContent)  
                {  
                    **case**"最简风格树视图":  
                        *myTreeView*.*memberTreeView*.Resources.**Clear**();  
                        break;  
                    **case**"带连接线树视图":  
                        *myTreeView*.*memberTreeView*.Resources.Source = **new**Uri("LineTreeStyle.xaml", **UriKind**.*RelativeOrAbsolute*);  
                        break;  
                    **case**"水平组织结构树视图":  
                        *myTreeView*.*memberTreeView*.Resources.Source = **new**Uri("GraphTreeStyle.xaml", **UriKind**.*RelativeOrAbsolute*);  
                        break;  
                    **case**"垂直组织结构树视图":  
                        *myTreeView*.*memberTreeView*.Resources.Source = **new**Uri("GraphTreeVStyle.xaml", **UriKind**.*RelativeOrAbsolute*);  
                        break;  
                    **case**"盒状风格树视图":  
                        *myTreeView*.*memberTreeView*.Resources.Source = **new**Uri("BoxTreeStyle.xaml", **UriKind**.*RelativeOrAbsolute*);  
                        break;  
                    **default**:  
                        break;  
                }  
            }  
        }  
  
        //切换视图  
        **private**void**TabView\_Changed**(object sender, SelectionChangedEventArgs e)  
        {  
            **if** (*tabView*.SelectedItem == *tabTree*)  
            {  
                *myTreeView*.Visibility = **Visibility**.*Visible*;  
                *listNodes*.Visibility = **Visibility**.*Collapsed*;  
            }  
            **else**  
            {  
                *listNodes*.Visibility = **Visibility**.*Visible*;  
                *myTreeView*.Visibility = **Visibility**.*Collapsed*;  
                **if** (*tabView*.SelectedItem == *tabFindResult*)  
                {  
                    *listNodes*.NodeListView.ItemsSource = *findResultNodes*;  
                }  
                **elseif** (*tabView*.SelectedItem == *tabIdErr*)  
                {  
                    *listNodes*.NodeListView.ItemsSource = MyTreeNode.IdConflictNodes;  
                }  
                **elseif** (*tabView*.SelectedItem == *tabNoParent*)  
                {  
                    *listNodes*.NodeListView.ItemsSource = MyTreeNode.NoParentNodes;  
                }  
                **elseif** (*tabView*.SelectedItem == *tabNodeInfoErr*)  
                {  
                    *listNodes*.NodeListView.ItemsSource = MyTreeNode.NodeInfoErrNodes;  
                }  
                **elseif** (*tabView*.SelectedItem == *tabRingErr*)  
                {  
                    *listNodes*.NodeListView.ItemsSource = MyTreeNode.RingNodes.Values;  
                }  
            }  
        }  
          
//        //切换视图  
//        private void TabView\_Changed(object sender, SelectionChangedEventArgs e)  
//        {  
//            if (tabView.SelectedItem == tabTree)  
//            {  
//                myTreeView.Visibility = Visibility.Visible;  
//                listNodes.Visibility = Visibility.Collapsed;  
//                forestView.Visibility =Visibility.Collapsed;  
//            }  
//            else if(tabView.SelectedItem == tabNoParent)  
//            {   
//                forestView.Visibility = Visibility.Visible;  
//                listNodes.Visibility = Visibility.Collapsed;  
//                myTreeView.Visibility =Visibility.Collapsed;  
//                forestView.SetForest(MyTreeNode.NoParentNodes);  
//            }  
//            else  
//            {  
//                listNodes.Visibility = Visibility.Visible;  
//                myTreeView.Visibility = Visibility.Collapsed;  
//                forestView.Visibility =Visibility.Collapsed;  
//                if (tabView.SelectedItem == tabFindResult)  
//                {  
//                    listNodes.NodeListView.ItemsSource = findResultNodes;  
//                }  
//                else if (tabView.SelectedItem == tabIdErr)  
//                {  
//                    listNodes.NodeListView.ItemsSource = MyTreeNode.IdConflictNodes;  
//                }  
//                else if (tabView.SelectedItem == tabNodeInfoErr)  
//                {  
//                    listNodes.NodeListView.ItemsSource = MyTreeNode.NodeInfoErrNodes;  
//                }  
//                else if (tabView.SelectedItem == tabRingErr)  
//                {  
//                    listNodes.NodeListView.ItemsSource = MyTreeNode.RingNodes.Values;  
//                }  
//            }  
//        }  
          
        //用户双击选中进入查看某个查找结果  
        **private**void**NodeListView\_MouseDoubleClick**(object sender, MouseButtonEventArgs e)  
        {  
            ListView currentList = (sender **as**MyNodeList).NodeListView;  
            MyTreeNode selectedNode = currentList.SelectedItem **as**MyTreeNode;  
            **if** (selectedNode != **null**)  
            {  
                *myTreeView*.**SetRootNode**(selectedNode);  
                //myGraphView.AddFindedNode(selectedNode);  
                *tabView*.SelectedItem = *tabTree*;  
  
                **int** maxLevel = **int**.**Parse**((*comboOpenLevel*.SelectedItem **as**ComboBoxItem).Tag.**ToString**());  
                *myTreeView*.**ExpandAllNodes**(maxLevel);//打开所有节点  
            }  
        }  
        #**endregion**  
         
        #**region** INotify接口  
          
        //设置进度条是否显示  
        **private**delegatevoid ProcessBarVisibleDelegate(**bool** visible);  
        **private** ProcessBarVisibleDelegate *processBarVisibleDelegate* = **null**;  
        **public**void**SetProcessBarVisible**(**bool** visible)  
        {  
            **if** (*processBarVisibleDelegate* == **null**)  
            {  
                *processBarVisibleDelegate* = **new** ProcessBarVisibleDelegate(**SetProcessBarVisibleImp**);  
            }  
            **this**.Dispatcher.**Invoke**(*processBarVisibleDelegate*, visible);  
            **DoEvents**();  
        }  
        **private**void**SetProcessBarVisibleImp**(**bool** visible)  
        {  
            **this**.*progressBar*.Value = 0;  
            **this**.*progressBar*.Visibility = visible ? **Visibility**.*Visible* : **Visibility**.*Collapsed*;  
            *mainGrid*.IsEnabled = !visible;  
            Cursor = visible ? Cursors.Wait : Cursors.Arrow;  
            *statusView*.Visibility = visible ? **Visibility**.*Collapsed* : **Visibility**.*Visible*;  
        }  
  
        //设置进度条进度(0-100)  
        **private**delegatevoid ProcessBarValueDelegate(**int** step);  
        **private** ProcessBarValueDelegate *processBarValueDelegate* = **null**;  
        **public**void**SetProcessBarValue**(**int** step)  
        {  
            **if** (*processBarValueDelegate* == **null**)  
            {  
                *processBarValueDelegate* = **new** ProcessBarValueDelegate(**SetProcessBarValueImp**);  
            }  
            **this**.Dispatcher.**Invoke**(*processBarValueDelegate*, step);  
            **DoEvents**();  
        }  
        **private**void**SetProcessBarValueImp**(**int** step)  
        {  
            **this**.*progressBar*.Value = step;  
        }  
  
        //设置状态栏提示文本  
        **private**delegatevoid ShowTextDelegate(string message);  
        **private** ShowTextDelegate *showTextDelegate* = **null**;  
        **public**void**SetStatusMessage**(string message)  
        {  
            **if** (*showTextDelegate* == **null**)  
            {  
                *showTextDelegate* = **new** ShowTextDelegate(**SetStatusMessageImp**);  
            }  
            **this**.Dispatcher.**Invoke**(*showTextDelegate*, message);  
            **DoEvents**();  
        }  
        **private**void**SetStatusMessageImp**(string message)  
        {  
            *statusMessage*.Text = message;  
        }  
  
        **private**void**DoEvents**()  
        {  
            DispatcherFrame frame = **new**DispatcherFrame();  
            Dispatcher.CurrentDispatcher.**BeginInvoke**(**DispatcherPriority**.*Background*,  
                **new** DispatcherOperationCallback(delegate(object f)  
                {  
                    (f **as**DispatcherFrame).Continue = **false**;  
  
                    return**null**;  
                }  
            ), frame);  
            Dispatcher.**PushFrame**(frame);  
        }   
        #**endregion**   
    }  
  
}

# 树形结构构造算法模块

**public**classMyTreeNode  
    {  
        **public**string SysId { get; set; }  
        **public**string RealName { get; set; }  
        **public**string TopId { get; set; }  
        **public**string Level { get; set; }  
        **publicint** LineCount { get; set; }  
  
        **public**MyTreeNode()  
        { }  
  
        **public**MyTreeNode(string sysId, string realName, string topId, string level, **int** lineCount, **int** upperLower, **int** DBCSBC, **int** trim)  
        {  
            **this**.SysId = sysId;  
            **this**.RealName = realName;  
            **this**.TopId = topId;  
            **this**.Level = level;  
            **this**.LineCount = lineCount;  
              
            **if**(trim == 0)  
            {  
                SysId = SysId.**Trim**();  
                TopId = TopId.**Trim**();  
            }  
            **elseif**(trim == 1)  
            {  
                SysId = SysId.**TrimStart**();  
                TopId = TopId.**TrimStart**();  
            }  
            **elseif**(trim == 2)  
            {  
                SysId = SysId.**TrimEnd**();  
                TopId = TopId.**TrimEnd**();  
            }  
            **if** (upperLower == 0)  
            {  
                SysId = SysId.**ToLower**();  
                TopId = TopId.**ToLower**();  
            }  
            **elseif** (upperLower == 1)  
            {  
                SysId = SysId.**ToUpper**();  
                TopId = TopId.**ToUpper**();  
            }  
            **if** (DBCSBC == 0)  
            {  
                SysId = TextUtilTool.**SBCToDBC**(SysId);  
                TopId = TextUtilTool.**SBCToDBC**(TopId);  
            }  
            **elseif** (DBCSBC == 1)  
            {  
                SysId = TextUtilTool.**DBCToSBC**(SysId);  
                TopId = TextUtilTool.**DBCToSBC**(TopId);  
            }  
        }  
  
        //所有的后代子孙节点数量  
        **publicint** DescendantCount { get; set; }  
        //所有的后代子孙最深级别数  
        **publicint** DescendantLevels { get; set; }  
  
        //返回父节点  
        **public**MyTreeNode ParentNode { get; set; }  
  
        **private**MyTreeNode**FindParentNode**()  
        {  
            **if**(**this**.TopId != "")  
            {  
                **if** (*allNodes*.**ContainsKey**(**this**.TopId))  
                {  
                    return*allNodes*[**this**.TopId];  
                }  
            }  
  
            return**null**;  
        }  
  
        **private**List<MyTreeNode>**FindAllParentNode**()  
        {  
            List<MyTreeNode> result = **new**List<MyTreeNode>();  
            **if** (*allNodes*.**ContainsKey**(**this**.TopId))  
            {  
                result.**Add**(*allNodes*[**this**.TopId]);  
            }  
            **foreach** (MyTreeNode node **in***idConflictNodes*)  
            {  
                **if** (node.SysId == **this**.TopId)  
                {  
                    result.**Add**(node);  
                }  
            }  
  
            return result;  
        }  
  
        //子节点集合  
        **private**List<MyTreeNode>*childrenNodes* = **new**List<MyTreeNode>();  
        **public**List<MyTreeNode> ChildrenNodes  
        {  
            get { return*childrenNodes*; }  
        }  
        #**region** 查找  
        **public**staticList<MyTreeNode>**FindNodeById**(string id)  
        {  
            List<MyTreeNode> result = **new**List<MyTreeNode>();  
            **if** (*allNodes*.**ContainsKey**(id))  
            {  
                result.**Add**(*allNodes*[id]);  
            }  
            **foreach** (MyTreeNode node **in***idConflictNodes*)  
            {  
                **if** (node.SysId == id)  
                {  
                    result.**Add**(node);  
                }  
            }  
  
            return result;  
        }  
  
        **public**staticList<MyTreeNode>**FindNodeByLevel**(string level)  
        {  
            List<MyTreeNode> result = **new**List<MyTreeNode>();  
            **foreach** (MyTreeNode node **in**MyTreeNode.*allNodes*.Values)  
            {  
                  **if** (node.Level == level)  
                {  
                    result.**Add**(node);  
                }  
            }  
            **foreach** (MyTreeNode node **in***idConflictNodes*)  
            {  
                **if** (node.Level == level)  
                {  
                    result.**Add**(node);  
                }  
            }  
            return result;  
        }  
  
        **public**staticList<MyTreeNode>**FindNodeByName**(string name)  
        {  
            List<MyTreeNode> result = **new**List<MyTreeNode>();  
            **foreach** (MyTreeNode node **in**MyTreeNode.*allNodes*.Values)  
            {  
                **if** (node.RealName == name)  
                {  
                    result.**Add**(node);  
                }  
            }  
            **foreach** (MyTreeNode node **in***idConflictNodes*)  
            {  
                **if** (node.RealName == name)  
                {  
                    result.**Add**(node);  
                }  
            }  
            return result;  
        }  
  
        **public**staticList<MyTreeNode>**FindNodeByDescendantCount**(**int** descendantCount)  
        {  
            List<MyTreeNode> result = **new**List<MyTreeNode>();  
            **foreach** (MyTreeNode node **in**MyTreeNode.*allNodes*.Values)  
            {  
                **if** (node.DescendantCount == descendantCount)  
                {  
                    result.**Add**(node);  
                }  
            }  
            **foreach** (MyTreeNode node **in***idConflictNodes*)  
            {  
                **if** (node.DescendantCount == descendantCount)  
                {  
                    result.**Add**(node);  
                }  
            }  
            return result;  
        }  
          
        **public**staticList<MyTreeNode>**FindNodeByDescendantLevels**(**int** descendantLevels)  
        {  
            List<MyTreeNode> result = **new**List<MyTreeNode>();  
            **foreach** (MyTreeNode node **in**MyTreeNode.*allNodes*.Values)  
            {  
                **if** (node.DescendantLevels == descendantLevels)  
                {  
                    result.**Add**(node);  
                }  
            }  
            **foreach** (MyTreeNode node **in***idConflictNodes*)  
            {  
                **if** (node.DescendantLevels == descendantLevels)  
                {  
                    result.**Add**(node);  
                }  
            }  
            return result;  
        }  
  
        #**endregion**  
  
        **public**static**intGetNodeInfoErrCount**()  
        {  
            **int** count = *nodeInfoErrNodes*.Count;  
            **foreach** (MyTreeNode node **in***nodeInfoErrNodes*)  
            {  
                count += node.DescendantCount;  
            }  
            return count;  
        }  
  
        #**region** 静态数据和静态方法  
  
        **public**static**int***allNodesCount*;  
        **private**staticDictionary<string, MyTreeNode>*allNodes* = **new**Dictionary<string, MyTreeNode>();  
  
        **private**staticMyTreeNode*rootNode* = **null**;  
        **public**staticMyTreeNode RootNode  
        {  
            get { return*rootNode*; }  
        }  
  
        **private**staticList<MyTreeNode>*noParentNodes* = **new**List<MyTreeNode>();  
        **public**staticList<MyTreeNode> NoParentNodes   
        {   
            get { return*noParentNodes*; }  
        }  
          
        **private**staticList<MyTreeNode>*idConflictNodes* = **new**List<MyTreeNode>();  
        **public**staticList<MyTreeNode> IdConflictNodes  
        {  
            get { return*idConflictNodes*; }  
        }  
  
        **private**staticDictionary<string, MyTreeNode>*ringNodes* = **new**Dictionary<string, MyTreeNode>();  
        **public**staticDictionary<string, MyTreeNode> RingNodes  
        {  
            get { return*ringNodes*; }  
        }  
  
        **private**staticList<MyTreeNode>*nodeInfoErrNodes* = **new**List<MyTreeNode>();  
        **public**staticList<MyTreeNode> NodeInfoErrNodes  
        {  
            get { return*nodeInfoErrNodes*; }  
        }  
  
        **public**staticvoid**OpenExcelFile**(string filepath, **int** upperLower, **int** DBCSBC, **int** trim)  
        {  
            **ClearAllNodes**();  
  
            FileInfo excelFile = **new**FileInfo(filepath);  
            **if** (!excelFile.Exists)  
            {  
                MessageBox.**Show**("文件" + filepath + "不存在！");  
            }  
            **else**  
            {  
                **int** row = 2;  
                **try**  
                {  
                    MainWindow.*notify*.**SetProcessBarVisible**(**true**);  
                    MainWindow.*notify*.**SetStatusMessage**("正在读取Excel文件......");  
  
                    **using** (ExcelPackage package = **new**ExcelPackage(excelFile))  
                    {  
                        ExcelWorksheet ws = package.Workbook.Worksheets[1];  
                        **if** (ws.Cells[1,1].Text.**ToLower**() != "sysid"||  
                           ws.Cells[1,2].Text.**ToLower**() != "realname"||  
                           ws.Cells[1,3].Text.**ToLower**() != "topid"||  
                           ws.Cells[1,4].Text.**ToLower**() != "layer")  
                        {  
                                MessageBox.**Show**("文件格式不正确，第一行必须由sysid,realname,topid,layer四个标题头组成！");  
                            return;  
                        }  
  
                        MainWindow.*notify*.**SetStatusMessage**("开始初始化数据结构......");  
                        //先将总行数读出来  
                        *allNodesCount* = ws.Dimension.End.Row - 1;  
  
                        **int** step = *allNodesCount*>100 ? *allNodesCount* / 100 : 1;  
                        **for** (**int** i = 0; i <*allNodesCount*; i++)  
                        {  
                            string sysId = ws.Cells[row, 1].Text;  
                            string realName = ws.Cells[row, 2].Text;  
                            string topId = ws.Cells[row, 3].Text;  
                            string lyr = ws.Cells[row, 4].Text;  
  
                            MyTreeNode myNode = **new**MyTreeNode(sysId, realName, topId, lyr, row, upperLower, DBCSBC, trim);  
  
                            **if** (*allNodes*.**ContainsKey**(myNode.SysId))  
                            {  
                                *idConflictNodes*.**Add**(myNode);  
                            }  
                            **else**  
                            {  
                                *allNodes*.**Add**(myNode.SysId, myNode);  
                            }  
  
                            row++;  
                            **if** (row % step == 0)  
                            {  
                                MainWindow.*notify*.**SetProcessBarValue**((**int**)(100.0 \* row / *allNodesCount*));  
                                MainWindow.*notify*.**SetStatusMessage**("正在处理生成第" + row + "个节点（总共" + *allNodesCount* + "个节点）");  
                            }  
                        }  
  
                        row = 0;  
                        **foreach** (MyTreeNode node **in***allNodes*.Values)  
                        {  
                            //将节点加入树中合适的位置去  
                            **ConstructTree**(node);  
  
                            row++;  
                            **if** (row % step == 0)  
                            {  
                                MainWindow.*notify*.**SetProcessBarValue**((**int**)(100.0 \* row / *allNodesCount*));  
                                MainWindow.*notify*.**SetStatusMessage**("正在构造树结构，加载第" + row + "个节点（总共" + *allNodesCount* + "个节点）");  
                            }  
                        }  
                        **foreach** (MyTreeNode node **in***idConflictNodes*)  
                        {  
                            //将节点加入树中合适的位置去  
                            **ConstructTree**(node);  
                        }  
  
                        MainWindow.*notify*.**SetStatusMessage**("");  
                    }  
                }  
                **catch** (Exception ex)  
                {  
                    MainWindow.*notify*.**SetStatusMessage**("发生异常：" + ex.Message + "在第" + row + "行!");  
                }  
                MainWindow.*notify*.**SetProcessBarVisible**(**false**);  
            }  
        }  
  
        **public**staticvoid**OpenCSVFile**(string filepath, **int** upperLower, **int** DBCSBC, **int** trim)  
        {  
            **ClearAllNodes**();  
            Encoding encoding = TextUtilTool.**GetFileEncodeType**(filepath);  
            StreamReader mysr = **new**StreamReader(filepath, encoding);  
            string[] allLines = **null**;  
            **try**  
            {  
                MainWindow.*notify*.**SetProcessBarVisible**(**true**);  
                MainWindow.*notify*.**SetStatusMessage**("开始读取文件......");  
  
                string firstLine = mysr.**ReadLine**(); //第一行是表头，读取之后不处理，直接跳过  
                **if** (firstLine.**ToLower**() != "sysid,realname,topid,layer")  
                {  
                    MessageBox.**Show**("文件格式不正确，第一行必须由sysid,realname,topid,layer四个标题头组成！");  
                    return;  
                }  
  
                string allStr = mysr.**ReadToEnd**();  
                string[] newLineStr = **new**string[] { Environment.NewLine };  
                allLines = allStr.**Split**(newLineStr, **StringSplitOptions**.*RemoveEmptyEntries*);  
            }  
            **catch** (Exception ex)  
            {  
                MainWindow.*notify*.**SetStatusMessage**("文件读取出错！+\n" + ex.Message);  
            }  
            **finally**  
            {  
                mysr.**Close**();  
            }  
  
            **int** row = 1;  
            *allNodesCount* = allLines.Length;  
            **int** step = *allNodesCount*>100 ? *allNodesCount* / 100 : 1;  
            **try**  
            {  
                **foreach** (string line **in** allLines)  
                {  
                    string[] aryline = line.**Split**(**newchar**[] { ',' });  
                    MyTreeNode myNode = **new**MyTreeNode(aryline[0], aryline[1], aryline[2], aryline[3], row + 1, upperLower, DBCSBC, trim);  
  
                    **if** (*allNodes*.**ContainsKey**(myNode.SysId))  
                    {  
                        *idConflictNodes*.**Add**(myNode);  
                    }  
                    **else**  
                    {  
                        *allNodes*.**Add**(myNode.SysId, myNode);  
                    }  
  
                    row++;  
                    **if** (row % step == 0)  
                    {  
                        MainWindow.*notify*.**SetProcessBarValue**((**int**)(100.0 \* row / *allNodesCount*));  
                        MainWindow.*notify*.**SetStatusMessage**("正在处理生成第" + row + "个节点（总共" + *allNodesCount* + "个节点）");  
                    }  
                }  
  
                row = 0;  
                **foreach** (MyTreeNode node **in***allNodes*.Values)  
                {  
                    //将节点加入树中合适的位置去  
                    **ConstructTree**(node);  
  
                    row++;  
                    **if** (row % step == 0)  
                    {  
                        MainWindow.*notify*.**SetProcessBarValue**((**int**)(100.0 \* row / *allNodesCount*));  
                        MainWindow.*notify*.**SetStatusMessage**("正在构造树结构，加载第" + row + "个节点（总共" + *allNodesCount* + "个节点）");  
                    }  
                }  
                **foreach** (MyTreeNode node **in***idConflictNodes*)  
                {  
                    //将节点加入树中合适的位置去  
                    **ConstructTree**(node);  
                }  
  
                MainWindow.*notify*.**SetStatusMessage**("");  
            }  
            **catch** (Exception ex)  
            {  
                MainWindow.*notify*.**SetStatusMessage**("发生异常：" + ex.Message + "在第" + row + "行!");  
            }  
            MainWindow.*notify*.**SetProcessBarVisible**(**false**);  
        }  
  
        **private**staticvoid**ClearAllNodes**()  
        {  
            *allNodesCount* = 0;  
            *allNodes*.**Clear**();  
            *idConflictNodes*.**Clear**();  
            *idConflictNodes*.**Clear**();  
            *noParentNodes*.**Clear**();  
            *ringNodes*.**Clear**();  
        }  
  
        //构建树（将节点加进树结构中合适的位置）  
        **private**staticvoid**ConstructTree**(MyTreeNode myNode)  
        {  
            string lyr = myNode.Level;  
  
            //节点信息不完整的节点  
            **if** (myNode.SysId == "")  
            {  
                *nodeInfoErrNodes*.**Add**(myNode);  
            }  
            **else**  
            {  
                //是否包含父节点  
                MyTreeNode parentNode = myNode.**FindParentNode**();  
                **if** (parentNode != **null**)  
                {  
                    **DescendantCountInc**(myNode);//所有父节点的子孙节点加1  
                    parentNode.*childrenNodes*.**Add**(myNode);  
                    myNode.ParentNode = parentNode;  
                }  
                **else**  
                {  
                    //父节点不存在  
                    *noParentNodes*.**Add**(myNode);  
                    **if**(myNode.Level == "")  
                    {  
                        myNode.Level = "1";  
                    }  
                }  
            }  
        }  
  
        //所有父节点的子孙节点数自增，（如果需要的话，所有父节点的子孙节点最深层级数自增）  
        **public**staticvoid**DescendantCountInc**(MyTreeNode node)  
        {  
            Dictionary<string, MyTreeNode> parentList = **new**Dictionary<string, MyTreeNode>();  
            //parentList.Add(node.SysId, node);  
  
            **int** deepLevel = 0; //深度（父节点到子节点之间的层级数之差）  
            MyTreeNode parent = node.**FindParentNode**();  
            **while** (parent != **null**)  
            {  
                **if** (parentList.**ContainsKey**(parent.SysId))  
                {  
                    **if** (!*ringNodes*.**ContainsKey**(node.SysId))  
                    {  
                        *ringNodes*.**Add**(node.SysId, node);  
                    }  
  
                    **foreach** (string item **in** parentList.Keys)  
                    {  
                        **if** (!*ringNodes*.**ContainsKey**(item))  
                        {  
                            *ringNodes*.**Add**(item, parentList[item]);  
                        }  
                    }  
                    break;  
                }  
                parentList.**Add**(parent.SysId, parent);  
  
                parent.DescendantCount++;  
                deepLevel++;  
                **if**(parent.DescendantLevels < deepLevel)  
                {  
                    parent.DescendantLevels = deepLevel;  
                }  
                parent = parent.**FindParentNode**();  
            }  
              
            **if**(node.Level == "")  
            {  
                node.Level = (deepLevel + 1).**ToString**();  
            }  
        }  
  
        #**endregion**  
    }

# 动态绘制树形结构模块

## 树容器结构代码

**namespace** TreeContainer  
{  
    **public**classTreeContainer : Panel  
    {  
        LayeredTreeDraw*\_ltd*;  
        **int***\_iNextNameSuffix* = 0;  
  
        **public**TreeContainer()  
        {  
        }  
  
        **public**List<**TreeConnection**> Connections  
        {  
            get  
            {  
                **if** (*\_ltd* != **null**)  
                {  
                    return*\_ltd*.Connections;  
                }  
                **else**  
                {  
                    return**null**;  
                }  
            }  
        }  
  
        #**region** Dependency Properties=====================================  
        #**region** Root  
        **public**staticreadonlyDependencyProperty*RootProperty* =  
            DependencyProperty.**Register**(  
                "Root",  
                **typeof**(String),  
                **typeof**(TreeContainer),  
                **new**FrameworkPropertyMetadata(  
                    **null**,  
                    **FrameworkPropertyMetadataOptions**.*AffectsMeasure* |  
                    **FrameworkPropertyMetadataOptions**.*AffectsArrange* |  
                    **FrameworkPropertyMetadataOptions**.*AffectsParentMeasure* |  
                    **FrameworkPropertyMetadataOptions**.*AffectsParentArrange* |  
                    **FrameworkPropertyMetadataOptions**.*AffectsRender* |  
                    0,  
                    **null**,  
                    **null**,  
                    **true**  
                ),  
                **null**  
            );  
  
        **public**string Root  
        {  
            get  
            {  
                return (string)**GetValue**(*RootProperty*);  
            }  
            set  
            {  
                **SetValue**(*RootProperty*, **value**);  
            }  
        }  
        #**endregion**  
  
        #**region** VerticalJustification  
        **public**staticreadonlyDependencyProperty*VerticalJustifcationProperty* =  
            DependencyProperty.**Register**(  
                "VerticalJustification",  
                **typeof**(**VerticalJustification**),  
                **typeof**(TreeContainer),  
                **new**FrameworkPropertyMetadata(  
                    **VerticalJustification**.*top*,  
                    **FrameworkPropertyMetadataOptions**.*AffectsMeasure* |  
                    **FrameworkPropertyMetadataOptions**.*AffectsArrange* |  
                    **FrameworkPropertyMetadataOptions**.*AffectsParentMeasure* |  
                    **FrameworkPropertyMetadataOptions**.*AffectsParentArrange* |  
                    **FrameworkPropertyMetadataOptions**.*AffectsRender* |  
                    0,  
                    **null**,  
                    **null**,  
                    **true**  
                ),  
                **null**  
            );  
  
        **publicVerticalJustification** VerticalJustification  
        {  
            get  
            {  
                return (**VerticalJustification**)**GetValue**(*VerticalJustifcationProperty*);  
            }  
            set  
            {  
                **SetValue**(*VerticalJustifcationProperty*, **value**);  
            }  
        }  
  
        #**endregion**  
  
        #**region** VerticalBufferProperty  
        **public**staticreadonlyDependencyProperty*VerticalBufferProperty* =  
            DependencyProperty.**Register**(  
                "VerticalBuffer",  
                **typeof**(**double**),  
                **typeof**(TreeContainer),  
                **new**FrameworkPropertyMetadata(  
                    10.0,  
                    **FrameworkPropertyMetadataOptions**.*AffectsMeasure* |  
                    **FrameworkPropertyMetadataOptions**.*AffectsArrange* |  
                    **FrameworkPropertyMetadataOptions**.*AffectsParentMeasure* |  
                    **FrameworkPropertyMetadataOptions**.*AffectsParentArrange* |  
                    **FrameworkPropertyMetadataOptions**.*AffectsRender* |  
                    0,  
                    **null**,  
                    **null**,  
                    **false**  
                ),  
                **null**  
            );  
  
        **publicdouble** VerticalBuffer  
        {  
            get { return (**double**)**GetValue**(*VerticalBufferProperty*); }  
            set { **SetValue**(*VerticalBufferProperty*, **value**); }  
        }  
  
        #**endregion**  
  
        #**region** HorizontalBufferSubtreeProperty  
        **public**readonlystaticDependencyProperty*HorizontalBufferSubtreeProperty* =  
            DependencyProperty.**Register**(  
                "HorizontalBufferSubtree",  
                **typeof**(**double**),  
                **typeof**(TreeContainer),  
                **new**FrameworkPropertyMetadata(  
                    10.0,  
                    **FrameworkPropertyMetadataOptions**.*AffectsMeasure* |  
                    **FrameworkPropertyMetadataOptions**.*AffectsArrange* |  
                    **FrameworkPropertyMetadataOptions**.*AffectsParentMeasure* |  
                    **FrameworkPropertyMetadataOptions**.*AffectsParentArrange* |  
                    **FrameworkPropertyMetadataOptions**.*AffectsRender* |  
                    0,  
                    **null**,  
                    **null**,  
                    **false**  
                ),  
                **null**  
            );  
  
        **publicdouble** HorizontalBufferSubtree  
        {  
            get { return (**double**)**GetValue**(*HorizontalBufferSubtreeProperty*); }  
            set { **SetValue**(*HorizontalBufferSubtreeProperty*, **value**); }  
        }  
        #**endregion**  
  
        #**region** HorizontalBufferProperty  
        **public**readonlystaticDependencyProperty*HorizontalBufferProperty* =  
            DependencyProperty.**Register**(  
                "HorizontalBuffer",  
                **typeof**(**double**),  
                **typeof**(TreeContainer),  
                **new**  FrameworkPropertyMetadata(  
                    10.0,  
                    **FrameworkPropertyMetadataOptions**.*AffectsMeasure* |  
                    **FrameworkPropertyMetadataOptions**.*AffectsArrange* |  
                    **FrameworkPropertyMetadataOptions**.*AffectsParentMeasure* |  
                    **FrameworkPropertyMetadataOptions**.*AffectsParentArrange* |  
                    **FrameworkPropertyMetadataOptions**.*AffectsRender* |  
                    0,  
                    **null**,  
                    **null**,  
                    **false**  
                ),  
                **null**  
            );  
  
        **publicdouble** HorizontalBuffer  
        {  
            get { return (**double**)**GetValue**(*HorizontalBufferProperty*); }  
            set { **SetValue**(*HorizontalBufferProperty*, **value**); }  
        }  
        #**endregion**  
        #**endregion**  
  
        #**region** Parenting=================================================  
        **private**void**SetParents**(TreeNode tnRoot)  
        {  
            // First pass to clear all parents  
            **foreach** (UIElement uiel **in** InternalChildren)  
            {  
                TreeNode tn = uiel **as**TreeNode;  
                **if** (tn != **null**)  
                {  
                    tn.**ClearParent**();  
                }  
            }  
  
            // Second pass to properly set them from their children...  
            **foreach** (UIElement uiel **in** InternalChildren)  
            {  
                TreeNode tn = uiel **as**TreeNode;  
                **if** (tn != **null**&& tn != tnRoot)  
                {  
                    tn.**SetParent**();  
                }  
            }  
        }  
        #**endregion**  
  
        #**region** Public utilities==========================================  
        **public**void**Clear**()  
        {  
            **foreach** (TreeNode tnCur **in** Children)  
            {  
                **UnregisterName**(tnCur.Name);  
            }  
            Children.**Clear**();  
        }  
  
        **public**void**ClearNodeChildren**(TreeNode tnParent)  
        {  
            **foreach** (TreeNode node **in** tnParent.TreeChildren)  
            {  
                **UnregisterName**(node.Name);  
                Children.**Remove**(node);  
            }  
        }  
  
        **private**void**SetName**(TreeNode tn, string strName)  
        {  
            tn.Name = strName;  
            **RegisterName**(strName, tn);  
        }  
  
        **public**TreeNode**AddRoot**(Object objContent, string strName)  
        {  
            TreeNode tnNew = **new**TreeNode();  
            **SetName**(tnNew, strName);  
            tnNew.Content = objContent;  
            Children.**Add**(tnNew);  
            Root = strName;  
            return tnNew;  
        }  
  
        **public**TreeNode**AddRoot**(Object objContent)  
        {  
            return**AddRoot**(objContent, **StrNextName**());  
        }  
  
        **public**TreeNode**AddNode**(Object objContent, string strName, string strParent)  
        {  
            TreeNode tnNew = **new**TreeNode();  
            **SetName**(tnNew, strName);  
            tnNew.Content = objContent;  
            tnNew.TreeParent = strParent;  
            Children.**Add**(tnNew);  
            return tnNew;  
        }  
  
        **private**string**StrNextName**()  
        {  
            return"\_\_TreeNode" + *\_iNextNameSuffix*++;  
        }  
  
        **public**TreeNode**AddNode**(Object objContent, string strName, TreeNode tnParent)  
        {  
            return**AddNode**(objContent, strName, tnParent.Name);  
        }  
  
        **public**TreeNode**AddNode**(Object objContent, TreeNode tnParent)  
        {  
            return**AddNode**(objContent, **StrNextName**(), tnParent.Name);  
        }  
        #**endregion**  
  
        #**region** Panel overrides ==========================================  
        **protected**override**SizeMeasureOverride**(**Size** availableSize)  
        {  
            **if** (Children.Count == 0)  
            {  
                return**newSize**(100, 20);  
            }  
  
            **Size** szFinal = **newSize**(0, 0);  
            string strRoot = Root;  
            TreeNode tnRoot = **this**.**FindName**(strRoot) **as**TreeNode;  
  
            **foreach** (UIElement uiel **in** InternalChildren)  
            {  
                uiel.**Measure**(availableSize);  
                **Size** szThis = uiel.DesiredSize;  
  
                **if** (szThis.Width > szFinal.Width || szThis.Height > szFinal.Height)  
                {  
                    szFinal = **newSize**(  
                        Math.**Max**(szThis.Width, szFinal.Width),  
                        Math.**Max**(szThis.Height, szFinal.Height));  
                }  
            }  
  
            **if** (tnRoot != **null**)  
            {  
                **SetParents**(tnRoot);  
                *\_ltd* = **new**LayeredTreeDraw(tnRoot, HorizontalBuffer, HorizontalBufferSubtree, VerticalBuffer, **VerticalJustification**.*top*);  
                *\_ltd*.**LayoutTree**();  
                szFinal = **newSize**(*\_ltd*.PxOverallWidth, *\_ltd*.PxOverallHeight);  
            }  
  
            return szFinal;  
        }  
  
        **protected**override**SizeArrangeOverride**(**Size** finalSize)  
        {  
            **foreach** (UIElement uiel **in** InternalChildren)  
            {  
                TreeNode tn = uiel **as**TreeNode;  
                **Point** ptLocation = **newPoint**(0, 0);  
                **if** (tn != **null**)  
                {  
                    ptLocation = **newPoint**(*\_ltd*.**X**(tn), *\_ltd*.**Y**(tn));  
                }  
                uiel.**Arrange**(**newRect**(ptLocation, uiel.DesiredSize));  
            }  
  
            return finalSize;  
        }  
        #**endregion**  
  
        #**region** Connection Rendering======================================  
        static**PointPtFromDPoint**(**DPoint** dpt)  
        {  
            return**newPoint**(dpt.*X*, dpt.*Y*);  
        }  
  
        **protected**overridevoid**OnRender**(System.Windows.Media.DrawingContext dc)  
        {  
            **base**.**OnRender**(dc);  
            **if** (Connections != **null**)  
            {  
                //连接线是否反锯齿显示（斜线需要反锯齿显示，横竖线则不需要）  
                //RenderOptions.SetEdgeMode(this, EdgeMode.Aliased);  
  
                SolidColorBrush brsh = **new**SolidColorBrush(Colors.Black);  
                brsh.Opacity = 0.5;  
                Pen pen = **new**Pen(brsh, 1.0);  
                **Point** ptLast = **newPoint**(0, 0);  
                **bool** fHaveLastPoint = **false**;  
  
                **foreach** (**TreeConnection** tcn **in** Connections)  
                {  
                    fHaveLastPoint = **false**;  
                    **foreach** (**DPoint** dpt **in** tcn.LstPt)  
                    {  
                        **if** (!fHaveLastPoint)  
                        {  
                            ptLast = **PtFromDPoint**(tcn.LstPt[0]);  
                            fHaveLastPoint = **true**;  
                            continue;  
                        }  
                        dc.**DrawLine**(pen, **PtFromDPoint**(dpt), ptLast);  
                        ptLast = **PtFromDPoint**(dpt);  
                    }  
                }  
            }  
        }  
        #**endregion**  
  
    }  
}

## 树节点结构代码

**namespace** TreeContainer  
{  
    **public**classTreeNode : ContentControl, ITreeNode  
    {  
        #**region** Dependency Properties  
        #**region** Collapsed  
        **public**staticreadonlyDependencyProperty*CollapsedProperty* =  
            DependencyProperty.**Register**(  
                "Collapsed",  
                **typeof**(**bool**),  
                **typeof**(TreeNode),  
                **new**FrameworkPropertyMetadata(  
                    **false**,  
                    **FrameworkPropertyMetadataOptions**.*AffectsMeasure* |  
                    **FrameworkPropertyMetadataOptions**.*AffectsArrange* |  
                    **FrameworkPropertyMetadataOptions**.*AffectsParentMeasure* |  
                    **FrameworkPropertyMetadataOptions**.*AffectsParentArrange* |  
                    **FrameworkPropertyMetadataOptions**.*AffectsRender* |  
                    0,  
                    **CollapsePropertyChange**,  
                    **CollapsePropertyCoerce**,  
                    **true**  
                ),  
                **null**  
            );  
  
        **public**staticvoid**CollapsePropertyChange**(DependencyObject o, **DependencyPropertyChangedEventArgs** e)  
        {  
            TreeNode tn = o **as**TreeNode;  
            **if** (tn != **null**&& tn.Collapsible)  
            {  
                **bool** fCollapsed = ((**bool**)e.NewValue);  
                **foreach** (TreeNode tnCur **in**LayeredTreeDraw.**VisibleDescendants**<TreeNode>(tn))  
                {  
                    tnCur.Visibility = fCollapsed ? **Visibility**.*Hidden* : **Visibility**.*Visible*;  
                }  
            }  
        }  
  
        **private**staticobject**CollapsePropertyCoerce**(DependencyObject d, object value)  
        {  
            TreeNode tn = (TreeNode)d;  
            **bool** fCollapsed = (**bool**)value;  
            **if** (!tn.Collapsible)  
            {  
                fCollapsed = **false**;  
            }  
            return fCollapsed;  
        }  
  
        **publicbool** Collapsed  
        {  
            get { return (**bool**)**GetValue**(*CollapsedProperty*); }  
            set { **SetValue**(*CollapsedProperty*, **value**); }  
        }  
        #**endregion**  
  
        #**region** Collapsible  
        **public**staticreadonlyDependencyProperty*CollapsibleProperty* =  
            DependencyProperty.**Register**(  
                "Collapsible",  
                **typeof**(**bool**),  
                **typeof**(TreeNode),  
                **new**FrameworkPropertyMetadata(  
                    **true**,  
                    **FrameworkPropertyMetadataOptions**.*AffectsMeasure* |  
                    **FrameworkPropertyMetadataOptions**.*AffectsArrange* |  
                    **FrameworkPropertyMetadataOptions**.*AffectsParentMeasure* |  
                    **FrameworkPropertyMetadataOptions**.*AffectsParentArrange* |  
                    **FrameworkPropertyMetadataOptions**.*AffectsRender* |  
                    0,  
                    **CollapsiblePropertyChange**,  
                    **null**,  
                    **true**  
                ),  
                **null**  
            );  
  
        static**public**void**CollapsiblePropertyChange**(DependencyObject o, **DependencyPropertyChangedEventArgs** e)  
        {  
            TreeNode tn = o **as**TreeNode;  
            **if** (((**bool**)e.NewValue) == **false**&& tn != **null**)  
            {  
                tn.Collapsed = **false**;  
            }  
        }  
  
        **publicbool** Collapsible  
        {  
            get { return (**bool**)**GetValue**(*CollapsibleProperty*); }  
            set { **SetValue**(*CollapsibleProperty*, **value**); }  
        }  
        #**endregion**  
  
        #**region** TreeParent  
        **public**staticreadonlyDependencyProperty*TreeParentProperty* =  
            DependencyProperty.**Register**(  
                "TreeParent",  
                **typeof**(string),  
                **typeof**(TreeNode),  
                **new**FrameworkPropertyMetadata(  
                    **null**,  
                    **FrameworkPropertyMetadataOptions**.*AffectsMeasure* |  
                    **FrameworkPropertyMetadataOptions**.*AffectsArrange* |  
                    **FrameworkPropertyMetadataOptions**.*AffectsParentMeasure* |  
                    **FrameworkPropertyMetadataOptions**.*AffectsParentArrange* |  
                    **FrameworkPropertyMetadataOptions**.*AffectsRender* |  
                    0,  
                    **null**,  
                    **null**,  
                    **true**  
                ),  
                **null**  
            );  
  
        **public**staticTreeNode**GetParentElement**(TreeNode tn)  
        {  
            TreeContainer tc;  
            TreeNode tnParent;  
  
            **if** (tn == **null**)  
            {  
                return**null**;  
            }  
            tc = tn.Parent **as**TreeContainer;  
            **if** (tc == **null**)  
            {  
                return**null**;  
            }  
            string strParent = tn.TreeParent;  
            **if** (strParent == **null**)  
            {  
                return**null**;  
            }  
  
            tnParent = tc.**FindName**(strParent) **as**TreeNode;  
            **if** (tnParent == **null**)  
            {  
                return**null**;  
            }  
            return tnParent;  
        }  
  
        **public**string TreeParent  
        {  
            get { return (string)**GetValue**(*TreeParentProperty*); }  
            set { **SetValue**(*TreeParentProperty*, **value**); }  
        }  
        #**endregion**  
        #**endregion**  
  
        #**region** Constructors=========================================  
        **public**TreeNode()  
        {  
            TreeChildren = **new**TreeNodeGroup();  
            Background = Brushes.Transparent;  
        }  
  
        staticTreeNode()  
        {  
        }  
        #**endregion**  
  
        #**region** Parenting=========================================  
        **internal**void**ClearParent**()  
        {  
            TreeChildren = **new**TreeNodeGroup();  
        }  
  
        **internalboolSetParent**()  
        {  
            TreeNode tn = **GetParentElement**(**this**);  
            **if** (tn == **null**)  
            {  
                return**false**;  
            }  
            tn.TreeChildren.**Add**(**this**);  
            return**true**;  
        }  
        #**endregion**  
  
        #**region** ITreeNode Members=========================================  
        **public**object PrivateNodeInfo { get; set; }  
  
        **public**TreeNodeGroup TreeChildren { get; **private**set; }  
  
        **publicdouble** TreeHeight  
        {  
            get  
            {  
                return DesiredSize.Height;  
            }  
        }  
  
        **publicdouble** TreeWidth  
        {  
            get  
            {  
                return DesiredSize.Width;  
            }  
        }  
        #**endregion**  
    }  
}

## 树控件绘制算法代码

**namespace** TreeContainer  
{  
    #**region** Enums  
    **publicenumVerticalJustification**  
    {  
        top,  
        center,  
        bottom  
    }  
    #**endregion**  
  
    **public**classLayeredTreeDraw  
    {  
  
        #**region** Private variables  
        ITreeNode*\_tnRoot*;  
        **double***\_pxBufferHorizontal*;  
        **double***\_pxBufferHorizontalSubtree*;  
        **double***\_pxBufferVertical*;  
        List<**TreeConnection**>*\_lsttcn* = **new**List<**TreeConnection**>();  
        List<**double**>*\_lstLayerHeight* = **new**List<**double**>();  
        **VerticalJustification***\_vj*;  
        staticTreeNodeGroup*\_tngEmpty* = **new**TreeNodeGroup();  
        #**endregion**  
  
        #**region** Properties  
        **publicdouble** PxOverallHeight { get; **private**set;  }  
  
        **publicdouble** PxOverallWidth  
        {  
            get { return**Info**(*\_tnRoot*).SubTreeWidth; }  
        }  
  
        **public**List<**TreeConnection**> Connections  
        {  
            get { return*\_lsttcn*; }  
        }  
        #**endregion**  
  
        #**region** Constructor  
        **public**LayeredTreeDraw(  
            ITreeNode tnRoot,  
            **double** pxBufferHorizontal,  
            **double** pxBufferHorizontalSubtree,  
            **double** pxBufferVertical,  
            **VerticalJustification** vj)  
        {  
            *\_pxBufferHorizontal* = pxBufferHorizontal;  
            *\_pxBufferHorizontalSubtree* = pxBufferHorizontalSubtree;  
            *\_pxBufferVertical* = pxBufferVertical;  
            PxOverallHeight = 0.0;  
            *\_tnRoot* = tnRoot;  
            *\_vj* = vj;  
        }  
        #**endregion**  
  
        #**region** PrivateInfo Access  
        **private**staticLayeredTreeInfo**Info**(ITreeNode ign)  
        {  
            return (LayeredTreeInfo)ign.PrivateNodeInfo;  
        }  
  
        **publicdoubleX**(ITreeNode tn)  
        {  
            **if** (**Info**(tn) == **null**)  
            {  
                return0;  
            }  
            return**Info**(tn).pxFromLeft;  
        }  
  
        **publicdoubleY**(ITreeNode tn)  
        {  
            **if** (**Info**(tn) == **null**)  
            {  
                return0;  
            }  
            return**Info**(tn).pxFromTop;  
        }  
        #**endregion**  
  
        #**region** Enumerations over nodes  
        static**public**IEnumerable<T>**VisibleDescendants**<T>(ITreeNode tn)  
        {  
            **foreach** (ITreeNode tnCur **in** tn.TreeChildren)  
            {  
                **if** (!tnCur.Collapsed)  
                {  
                    **foreach** (T item **inVisibleDescendants**<T>(tnCur))  
                    {  
                        yieldreturn item;  
                    }  
                }  
                yieldreturn (T)tnCur;  
            }  
        }  
  
  
        static**public**IEnumerable<T>**Descendants**<T>(ITreeNode tn)  
        {  
            **foreach** (ITreeNode tnCur **in** tn.TreeChildren)  
            {  
                **foreach** (T item **inDescendants**<T>(tnCur))  
                {  
                    yieldreturn item;  
                }  
                yieldreturn (T)tnCur;  
            }  
        }  
        #**endregion**  
  
        #**region** Layout  
        #**region** Top Level Layout routines  
        **public**void**LayoutTree**()  
        {  
            **LayoutTree**(*\_tnRoot*, 0);  
            **DetermineFinalPositions**(*\_tnRoot*, 0, 0, **Info**(*\_tnRoot*).pxLeftPosRelativeToBoundingBox);  
        }  
  
        **private**void**LayoutTree**(ITreeNode tnRoot, **int** iLayer)  
        {  
            **if** (**GetChildren**(tnRoot).Count == 0)  
            {  
                **LayoutLeafNode**(tnRoot);  
            }  
            **else**  
            {  
                **LayoutInteriorNode**(tnRoot, iLayer);  
            }  
  
            **UpdateLayerHeight**(tnRoot, iLayer);  
        }  
  
        **private**staticvoid**LayoutLeafNode**(ITreeNode tnRoot)  
        {  
            **double** width = tnRoot.TreeWidth;  
            LayeredTreeInfo lti = **new**LayeredTreeInfo(width, tnRoot);  
            lti.*lstPosLeftBoundaryRelativeToRoot*.**Add**(0);  
            lti.*lstPosRightBoundaryRelativeToRoot*.**Add**(width);  
            tnRoot.PrivateNodeInfo = lti;  
        }  
  
        **private**void**LayoutInteriorNode**(ITreeNode tnRoot, **int** iLayer)  
        {  
            ITreeNode tnLast = **null**;  
            TreeNodeGroup tng = **GetChildren**(tnRoot);  
            ITreeNode itn = tng[0];  
            LayeredTreeInfo ltiThis;  
  
            **LayoutAllOurChildren**(iLayer, tnLast, tng);  
  
            // This width doesn't account for the parent node's width...  
            ltiThis = **new**LayeredTreeInfo(**CalculateWidthFromInterChildDistances**(tnRoot), tnRoot);  
            tnRoot.PrivateNodeInfo = ltiThis;  
  
            // ...so that this centering may place the parent node negatively while the "width" is the width of  
            // all the child nodes.  
            **CenterOverChildren**(tnRoot, ltiThis);  
            **DetermineParentRelativePositionsOfChildren**(tnRoot);  
            **CalculateBoundaryLists**(tnRoot);  
        }  
  
        **private**void**LayoutAllOurChildren**(**int** iLayer, ITreeNode tnLast, TreeNodeGroup tng)  
        {  
            List<**Double**> lstLeftToBB = **new**List<**double**>();  
            List<**int**> lstResponsible = **new**List<**int**>();  
            **for** (**int** i = 0; i < tng.Count; i++)  
            {  
                ITreeNode tn = tng[i];  
                **LayoutTree**(tn, iLayer + 1);  
                **RepositionSubtree**(i, tng, lstLeftToBB, lstResponsible);  
                tnLast = tn;  
            }  
        }  
        #**endregion**  
  
        #**region** Parent Relative Positioning  
        **private**staticvoid**CenterOverChildren**(ITreeNode tnRoot, LayeredTreeInfo ltiThis)  
        {  
            // We should be centered between  the connection points of our children...  
            ITreeNode tnLeftMost = tnRoot.TreeChildren.**LeftMost**();  
            **double** pxLeftChild = **Info**(tnLeftMost).pxLeftPosRelativeToBoundingBox + tnLeftMost.TreeWidth / 2;  
            ITreeNode tnRightMost = tnRoot.TreeChildren.**RightMost**();  
            **double** pxRightChild = **Info**(tnRightMost).pxLeftPosRelativeToBoundingBox + tnRightMost.TreeWidth / 2;  
            ltiThis.pxLeftPosRelativeToBoundingBox = (pxLeftChild + pxRightChild - tnRoot.TreeWidth) / 2;  
  
            // If the root node was wider than the subtree, then we'll have a negative position for it.  We need  
            // to readjust things so that the left of the root node represents the left of the bounding box and  
            // the child distances to the Bounding box need to be adjusted accordingly.  
            **if** (ltiThis.pxLeftPosRelativeToBoundingBox <0)  
            {  
                **foreach** (ITreeNode tnChildCur **in** tnRoot.TreeChildren)  
                {  
                    **Info**(tnChildCur).pxLeftPosRelativeToBoundingBox -= ltiThis.pxLeftPosRelativeToBoundingBox;  
                }  
                ltiThis.pxLeftPosRelativeToBoundingBox = 0;  
            }  
        }  
  
        **private**void**DetermineParentRelativePositionsOfChildren**(ITreeNode tnRoot)  
        {  
            LayeredTreeInfo ltiRoot = **Info**(tnRoot);  
            **foreach** (ITreeNode tn **inGetChildren**(tnRoot))  
            {  
                LayeredTreeInfo ltiCur = **Info**(tn);  
                ltiCur.pxLeftPosRelativeToParent = ltiCur.pxLeftPosRelativeToBoundingBox - ltiRoot.pxLeftPosRelativeToBoundingBox;  
            }  
        }  
        #**endregion**  
  
        #**region** Width Calculation  
        **privatedoubleCalculateWidthFromInterChildDistances**(ITreeNode tnRoot)  
        {  
            **double** pxWidthCur;  
            LayeredTreeInfo lti;  
            **double** pxWidth = 0.0;  
  
            lti = **Info**(tnRoot.TreeChildren.**LeftMost**());  
            pxWidthCur = lti.pxLeftPosRelativeToBoundingBox;  
  
            // If a subtree extends deeper than it's left neighbors then at that lower level it could potentially extend beyond those neighbors  
            // on the left.  We have to check for this and make adjustements after the loop if it occurred.  
            **double** pxUndercut = 0.0;  
  
            **foreach** (ITreeNode tn **in** tnRoot.TreeChildren)  
            {  
                lti = **Info**(tn);  
                pxWidthCur += lti.pxToLeftSibling;  
  
                **if** (lti.pxLeftPosRelativeToBoundingBox > pxWidthCur)  
                {  
                    pxUndercut = Math.**Max**(pxUndercut, lti.pxLeftPosRelativeToBoundingBox - pxWidthCur);  
                }  
                  
                // pxWidth might already be wider than the current node's subtree if earlier nodes "undercut" on the  
                // right hand side so we have to take the Max here...  
                pxWidth = Math.**Max**(pxWidth, pxWidthCur + lti.SubTreeWidth - lti.pxLeftPosRelativeToBoundingBox);  
  
                // After this next statement, the BoundingBox we're relative to is the one of our parent's subtree rather than  
                // our own subtree (with the exception of undercut considerations)  
                lti.pxLeftPosRelativeToBoundingBox = pxWidthCur;  
            }  
            **if** (pxUndercut >0.0)  
            {  
                **foreach** (ITreeNode tn **in** tnRoot.TreeChildren)  
                {  
                    **Info**(tn).pxLeftPosRelativeToBoundingBox += pxUndercut;  
                }  
                pxWidth += pxUndercut;  
            }  
  
            // We are never narrower than our root node's width which we haven't taken into account yet so  
            // we do that here.  
            returnMath.**Max**(tnRoot.TreeWidth, pxWidth);  
        }  
        #**endregion**  
  
        #**region** Boundary Lists  
        **private**void**CalculateBoundaryLists**(ITreeNode tnRoot)  
        {  
            LayeredTreeInfo lti = **Info**(tnRoot);  
            lti.*lstPosLeftBoundaryRelativeToRoot*.**Add**(0.0);  
            lti.*lstPosRightBoundaryRelativeToRoot*.**Add**(tnRoot.TreeWidth);  
            **DetermineBoundary**(tnRoot.TreeChildren, **true**/\* fLeft \*/, lti.*lstPosLeftBoundaryRelativeToRoot*);  
            **DetermineBoundary**(tnRoot.TreeChildren.**Reverse**(), **false**/\* fLeft \*/, lti.*lstPosRightBoundaryRelativeToRoot*);  
  
        }  
  
        **private**void**DetermineBoundary**(IEnumerable<ITreeNode> entn, **bool** fLeft, List<**double**> lstPos)  
        {  
            **int** cLayersDeep = 1;  
            List<**double**> lstPosCur;  
            **foreach** (ITreeNode tnChild **in** entn)  
            {  
                LayeredTreeInfo ltiChild = **Info**(tnChild);  
  
                **if** (fLeft)  
                {  
                    lstPosCur = ltiChild.*lstPosLeftBoundaryRelativeToRoot*;  
                }  
                **else**  
                {  
                    lstPosCur = ltiChild.*lstPosRightBoundaryRelativeToRoot*;  
                }  
  
                **if** (lstPosCur.Count >= lstPos.Count)  
                {  
                    **using** (IEnumerator<**double**> enPosCur = lstPosCur.**GetEnumerator**())  
                    {  
                        **for** (**int** i = 0; i < cLayersDeep - 1; i++)  
                        {  
                            enPosCur.**MoveNext**();  
                        }  
  
                        **while** (enPosCur.**MoveNext**())  
                        {  
                            lstPos.**Add**(enPosCur.Current + ltiChild.pxLeftPosRelativeToParent);  
                            cLayersDeep++;  
                        }  
                    }  
                }  
            }  
        }  
        #**endregion**  
  
        #**region** Repositioning Children  
        **private**void**ApportionSlop**(**int** itn, **int** itnResponsible, TreeNodeGroup tngSiblings)  
        {  
            LayeredTreeInfo lti = **Info**(tngSiblings[itn]);  
            ITreeNode tnLeft = tngSiblings[itn - 1];  
  
            **double** pxSlop = lti.pxToLeftSibling - tnLeft.TreeWidth - *\_pxBufferHorizontal*;  
            **if** (pxSlop >0)  
            {  
                **for** (**int** i = itnResponsible + 1; i < itn; i++)  
                {  
                    **Info**(tngSiblings[i]).pxToLeftSibling += pxSlop \* (i - itnResponsible) / (itn - itnResponsible);  
                }  
                lti.pxToLeftSibling -= (itn - itnResponsible - 1) \* pxSlop / (itn - itnResponsible);  
            }  
        }  
  
        **private**void**RepositionSubtree**(  
            **int** itn,  
            TreeNodeGroup tngSiblings,  
            List<**double**> lstLeftToBB,  
            List<**int**> lsttnResponsible)  
        {  
            **int** itnResponsible;  
            ITreeNode tn = tngSiblings[itn];  
            LayeredTreeInfo lti = **Info**(tn);  
  
            **if** (itn == 0)  
            {  
                // No shifting but we still have to prepare the initial version of the  
                // left hand skeleton list  
                **foreach** (**double** pxRelativeToRoot **in** lti.*lstPosRightBoundaryRelativeToRoot*)  
                {  
                    lstLeftToBB.**Add**(pxRelativeToRoot + lti.pxLeftPosRelativeToBoundingBox);  
                    lsttnResponsible.**Add**(0);  
                }  
                return;  
            }  
  
            ITreeNode tnLeft = tngSiblings[itn - 1];  
            LayeredTreeInfo ltiLeft = **Info**(tnLeft);  
            **int** iLayer;  
            **double** pxHorizontalBuffer = *\_pxBufferHorizontal*;  
  
            **double** pxNewPosFromBB = **PxCalculateNewPos**(lti, lstLeftToBB, lsttnResponsible, **out** itnResponsible, **out** iLayer);  
            **if** (iLayer != 0)  
            {  
                pxHorizontalBuffer = *\_pxBufferHorizontalSubtree*;  
            }  
  
            lti.pxToLeftSibling = pxNewPosFromBB - lstLeftToBB.**First**() + tnLeft.TreeWidth + pxHorizontalBuffer;  
  
            **int** cLevels = Math.**Min**(lti.*lstPosRightBoundaryRelativeToRoot*.Count, lstLeftToBB.Count);  
            **for** (**int** i = 0; i < cLevels; i++)  
            {  
                lstLeftToBB[i] = lti.*lstPosRightBoundaryRelativeToRoot*[i] + pxNewPosFromBB + pxHorizontalBuffer;  
                lsttnResponsible[i] = itn;  
            }  
            **for** (**int** i = lstLeftToBB.Count; i < lti.*lstPosRightBoundaryRelativeToRoot*.Count; i++)  
            {  
                lstLeftToBB.**Add**(lti.*lstPosRightBoundaryRelativeToRoot*[i] + pxNewPosFromBB + pxHorizontalBuffer);  
                lsttnResponsible.**Add**(itn);  
            }  
  
            **ApportionSlop**(itn, itnResponsible, tngSiblings);  
        }  
  
        **privatedoublePxCalculateNewPos**(  
            LayeredTreeInfo lti,  
            List<**double**> lstLeftToBB,  
            List<**int**> lstitnResponsible,  
            **outint** itnResponsible,  
            **outint** iLayerRet)  
        {  
            **double** pxOffsetToBB = lstLeftToBB[0];  
            **int** cLayers = Math.**Min**(lti.*lstPosLeftBoundaryRelativeToRoot*.Count, lstLeftToBB.Count);  
            **double** pxRootPosRightmost = 0.0;  
            iLayerRet = 0;  
  
            **using** (IEnumerator<**double**> enRight = lti.*lstPosLeftBoundaryRelativeToRoot*.**GetEnumerator**(),  
                enLeft = lstLeftToBB.**GetEnumerator**())  
            **using** (IEnumerator<**int**> enResponsible = lstitnResponsible.**GetEnumerator**())  
            {  
                itnResponsible = -1;  
  
                enRight.**MoveNext**();  
                enLeft.**MoveNext**();  
                enResponsible.**MoveNext**();  
                **for** (**int** iLayer = 0; iLayer < cLayers; iLayer++)  
                {  
                    **double** pxLeftBorderFromBB = enLeft.Current;  
                    **double** pxRightBorderFromRoot = enRight.Current;  
                    **double** pxRightRootBasedOnThisLevel;  
                    **int** itnResponsibleCur = enResponsible.Current;  
  
                    enLeft.**MoveNext**();  
                    enRight.**MoveNext**();  
                    enResponsible.**MoveNext**();  
  
                    pxRightRootBasedOnThisLevel = pxLeftBorderFromBB - pxRightBorderFromRoot;  
                    **if** (pxRightRootBasedOnThisLevel > pxRootPosRightmost)  
                    {  
                        iLayerRet = iLayer;  
                        pxRootPosRightmost = pxRightRootBasedOnThisLevel;  
                        itnResponsible = itnResponsibleCur;  
                    }  
                }  
            }  
  
            return pxRootPosRightmost;  
        }  
        #**endregion**  
  
        #**region** Height Calculations  
        **private**void**UpdateLayerHeight**(ITreeNode tnRoot, **int** iLayer)  
        {  
            **while** (*\_lstLayerHeight*.Count <= iLayer)  
            {  
                *\_lstLayerHeight*.**Add**(0.0);  
            }  
            *\_lstLayerHeight*[iLayer] = Math.**Max**(tnRoot.TreeHeight, *\_lstLayerHeight*[iLayer]);  
        }  
  
        **private** System.**DoubleCalcJustify**(**double** height, **double** pxRowHeight)  
        {  
            **double** dRet = 0.0;  
  
            **switch** (*\_vj*)  
            {  
                **caseVerticalJustification**.*top*:  
                    break;  
  
                **caseVerticalJustification**.*center*:  
                    dRet = (pxRowHeight - height) / 2;  
                    break;  
  
                **caseVerticalJustification**.*bottom*:  
                    dRet = pxRowHeight - height;  
                    break;  
            }  
  
            return dRet;  
        }  
        #**endregion**  
  
        #**region** Collapse handling  
        **private**TreeNodeGroup**GetChildren**(ITreeNode tn)  
        {  
            **if** (tn.Collapsed)  
            {  
                return*\_tngEmpty*;  
            }  
            return tn.TreeChildren;  
        }  
        #**endregion**  
  
        #**region** Second pass to convert parent relative positions to absolute positions  
        **private**void**DetermineFinalPositions**(ITreeNode tn, **int** iLayer, **double** pxFromTop, **double** pxParentFromLeft)  
        {  
            **double** pxRowHeight = *\_lstLayerHeight*[iLayer];  
            LayeredTreeInfo lti = **Info**(tn);  
            **double** pxBottom;  
            **DPoint** dptOrigin;  
  
            lti.pxFromTop = pxFromTop + **CalcJustify**(tn.TreeHeight, pxRowHeight);  
            pxBottom = lti.pxFromTop + tn.TreeHeight;  
            **if** (pxBottom > PxOverallHeight)  
            {  
                PxOverallHeight = pxBottom;  
            }  
            lti.pxFromLeft = lti.pxLeftPosRelativeToParent + pxParentFromLeft;  
            dptOrigin = **newDPoint**(lti.pxFromLeft + tn.TreeWidth / 2, lti.pxFromTop + tn.TreeHeight);  
            iLayer++;  
            TreeNodeGroup tng = **GetChildren**(tn);  
            **foreach** (ITreeNode tnCur **in** tng)  
            {  
                //斜线连接  
                //List<DPoint> lstcpt = new List<DPoint>();  
                //LayeredTreeInfo ltiCur = Info(tnCur);  
                //lstcpt.Add(dptOrigin);  
                //DetermineFinalPositions(tnCur, iLayer, pxFromTop + pxRowHeight + \_pxBufferVertical, lti.pxFromLeft);  
                //lstcpt.Add(new DPoint(ltiCur.pxFromLeft + tnCur.TreeWidth / 2, ltiCur.pxFromTop));  
                //\_lsttcn.Add(new TreeConnection(tn, tnCur, lstcpt));  
  
                //横竖线连接  
                List<**DPoint**> lstcpt = **new**List<**DPoint**>();  
                LayeredTreeInfo ltiCur = **Info**(tnCur);  
                lstcpt.**Add**(dptOrigin);  
                **DetermineFinalPositions**(tnCur, iLayer, pxFromTop + pxRowHeight + *\_pxBufferVertical*, lti.pxFromLeft);  
                //If parent node has only one child then no changes here, just a normal TreeConnection  
                **if** (tng.Count == 1)  
                {  
                    lstcpt.**Add**(**newDPoint**(ltiCur.pxFromLeft + tnCur.TreeWidth / 2, ltiCur.pxFromTop));  
                    *\_lsttcn*.**Add**(**newTreeConnection**(tn, tnCur, lstcpt));  
                }  
                **else**  
                {  
                    //If parent node has more than one child then add the extra connection points  
                    **double** halfHeight = (ltiCur.pxFromTop - dptOrigin.*Y*) / 2;  
                    **DPoint** p2 = **newDPoint**(dptOrigin.*X*, dptOrigin.*Y* + halfHeight);  
                    lstcpt.**Add**(p2);  
                    **DPoint** p3 = **newDPoint**(ltiCur.pxFromLeft + tnCur.TreeWidth / 2, dptOrigin.*Y* + halfHeight);  
                    lstcpt.**Add**(p3);  
                    **DPoint** p4 = **newDPoint**(ltiCur.pxFromLeft + tnCur.TreeWidth / 2, ltiCur.pxFromTop + 5);  
                    lstcpt.**Add**(p4);  
                    *\_lsttcn*.**Add**(**newTreeConnection**(tn, tnCur, lstcpt));  
                }  
            }  
        }  
        #**endregion**  
  
        #**endregion**  
  
        #**region** Internal classes  
        **private**classLayeredTreeInfo  
        {  
            **publicdouble** SubTreeWidth { get; set; }  
            **publicdouble** pxLeftPosRelativeToParent { get; set; }  
            **publicdouble** pxLeftPosRelativeToBoundingBox { get; set; }  
            **publicdouble** pxToLeftSibling { get; set; }  
            **publicdouble** pxFromTop { get; set; }  
            **publicdouble** pxFromLeft { get; set; }  
            **public**ITreeNode ign { get; **private**set; }  
            **public**List<**double**>*lstPosLeftBoundaryRelativeToRoot* = **new**List<**double**>();  
            **public**List<**double**>*lstPosRightBoundaryRelativeToRoot* = **new**List<**double**>();  
  
            ///<**summary**>  
            /// Initializes a new instance of the GraphLayoutInfo class.  
            ///<**/summary**>  
            **public**LayeredTreeInfo(**double** subTreeWidth, ITreeNode tn)  
            {  
                SubTreeWidth = subTreeWidth;  
                pxLeftPosRelativeToParent = 0;  
                pxFromTop = 0;  
                ign = tn;  
            }  
        }  
        #**endregion**  
    }  
}

## 树控件操作处理逻辑代码

**namespace** MemberTree  
{  
    ///<**summary**>  
    /// MyGraphView.xaml 的交互逻辑  
    ///<**/summary**>  
    **public**partialclassMyGraphView : UserControl  
    {  
        **public**MyGraphView()  
        {  
            **InitializeComponent**();  
        }  
  
        **public**void**InitMyTree**()  
        {  
            *memberTreeView*.**Clear**();  
  
            //加载所有根节点  
            Button rootBtn = **new**Button();  
            rootBtn.Content = "会员列表";  
            TreeNode memNode = *memberTreeView*.**AddRoot**(rootBtn);  
            **foreach** (MyTreeNode subNode **in**MyTreeNode.NoParentNodes)  
            {  
                Button subItem = **NewTreeViewItem**(subNode);  
                TreeNode rootNode = *memberTreeView*.**AddNode**(subItem, memNode);  
                rootNode.Tag = subNode;  
                rootNode.Collapsed = **true**;  
            }  
        }  
  
        **public**void**AddFindedNode**(MyTreeNode node)  
        {  
            *memberTreeView*.**Clear**();  
  
            Button rootBtn = **NewTreeViewItem**(node);  
            TreeNode memNode = *memberTreeView*.**AddRoot**(rootBtn);  
            **foreach** (MyTreeNode subNode **in** node.ChildrenNodes)  
            {  
                Button subItem = **NewTreeViewItem**(subNode);  
                TreeNode rootNode = *memberTreeView*.**AddNode**(subItem, memNode);  
                rootNode.Tag = subNode;  
                rootNode.Collapsed = **true**;  
            }  
        }  
  
        **private**Button**NewTreeViewItem**(MyTreeNode subNode)  
        {  
            Button btn = **new**Button();  
            btn.Content = subNode.RealName;  
            btn.ToolTip = "级别：" + subNode.Level + "，下线人数：" + subNode.DescendantCount;  
            btn.Click += **new** RoutedEventHandler(**btn\_Click**);  
            btn.MouseEnter += **item\_MouseEnter**;  
            btn.MouseMove += **item\_MouseMove**;  
            return btn;  
        }  
  
        void**item\_Expanded**(TreeNode tn)  
        {  
            MyTreeNode node = tn.Tag **as**MyTreeNode;  
            **if** (node != **null**)  
            {  
                List<MyTreeNode> childrenNodes = node.ChildrenNodes;  
                **foreach** (MyTreeNode subNode **in** childrenNodes)  
                {  
                    Button grandson = **NewTreeViewItem**(subNode);  
                    TreeNode newNode = *memberTreeView*.**AddNode**(grandson, tn);  
                    newNode.Tag = subNode;  
                    newNode.Collapsed = **true**;  
                }  
            }  
        }  
  
        void**item\_Collapsed**(TreeNode tn)  
        {  
            *memberTreeView*.**ClearNodeChildren**(tn);  
        }  
  
        void**item\_MouseEnter**(object sender, MouseEventArgs e)  
        {  
            **if** (*isAutoExpand*.IsChecked == **true**)  
            {  
                TreeViewItem item = sender **as**TreeViewItem;  
                //Console.WriteLine("===============鼠标进入了" + item.Header);  
                //item.IsExpanded = true;  
                e.Handled = **true**;  
            }  
        }  
  
        **bool***isCurrentLeave* = **false**;  
        void**item\_MouseMove**(object sender, MouseEventArgs e)  
        {  
            **if** (*isAutoExpand*.IsChecked == **true**)  
            {  
                Button item = sender **as**Button;  
                TreeNode tn = (TreeNode)(item.Parent);  
                MyTreeNode node = tn.Tag **as**MyTreeNode;  
                **if** (node != **null**)  
                {  
                   // statusText.Text = "Id：" + node.SysId + "，TopId：" + node.TopId;  
                }  
                *isCurrentLeave* = **true**;  
                e.Handled = **true**;  
            }  
        }  
  
        **private**void**btn\_Click**(object sender, RoutedEventArgs e)  
        {  
            Button btn = e.OriginalSource **as**Button;  
            **if** (btn != **null**)  
            {  
                TreeNode tn = (TreeNode)(btn.Parent);  
                 
                //if (tn.TreeChildren.Count > 0)  
                //{  
                    **if** (tn.Collapsed)  
                    {  
                        btn.Background = Brushes.White;  
                        **item\_Expanded**(tn);  
                        tn.Collapsed =**false**;  
                    }  
                    **else**  
                    {  
                        btn.Background = Brushes.Red;  
                        **item\_Collapsed**(tn);  
                        tn.Collapsed=**true**;  
                    }  
                //}  
            }  
        }  
    }  
}

## 线状树结构风格代码

<ResourceDictionary xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"  
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"  
    xmlns:local="clr-namespace:MemberTree">  
          
    <local:TreeViewLineConverter x:Key="LineConverter"/>  
  
    <!-- Toggle Button -->  
    <Style x:Key="ExpandCollapseToggleStyle"TargetType="ToggleButton">  
        <Setter Property="Focusable"Value="False"/>  
        <Setter Property="Template">  
            <Setter.Value>  
                <ControlTemplate TargetType="ToggleButton">  
                    <Grid Width="15"Height="13"SnapsToDevicePixels="True">  
                        <Rectangle Width="9"Height="9"Stroke="#919191"SnapsToDevicePixels="true">  
                            <Rectangle.Fill>  
                                <LinearGradientBrush EndPoint="0.5,2"StartPoint="0.5,0">  
                                    <GradientStop Color="White"Offset="0"/>  
                                    <GradientStop Color="Silver"Offset="0.5"/>  
                                    <GradientStop Color="LightGray"Offset="1"/>  
                                </LinearGradientBrush>  
                            </Rectangle.Fill>  
                        </Rectangle>  
                        <Rectangle x:Name="ExpandPath"Width="1"Height="5"Stroke="Black"SnapsToDevicePixels="true"/>  
                        <Rectangle Width="5"Height="1"Stroke="Black"SnapsToDevicePixels="true"/>  
                    </Grid>  
                    <ControlTemplate.Triggers>  
                        <Trigger Property="IsChecked"Value="True">  
                            <Setter Property="Visibility"  TargetName="ExpandPath"Value="Collapsed"/>  
                        </Trigger>  
                    </ControlTemplate.Triggers>  
                </ControlTemplate>  
            </Setter.Value>  
        </Setter>  
    </Style>  
  
    <!-- TreeViewItem -->  
    <Style TargetType="{x:Type TreeViewItem}">  
        <Setter Property="Background"Value="Transparent"/>  
        <Setter Property="Padding"Value="1,0,0,0"/>  
        <Setter Property="Template">  
            <Setter.Value>  
                <ControlTemplate TargetType="{x:Type TreeViewItem}">  
                    <Grid>  
                        <Grid.ColumnDefinitions>  
                            <ColumnDefinition MinWidth="19"Width="Auto"/>  
                            <ColumnDefinition Width="Auto"/>  
                            <ColumnDefinition Width="\*"/>  
                        </Grid.ColumnDefinitions>  
                        <Grid.RowDefinitions>  
                            <RowDefinition Height="Auto"/>  
                            <RowDefinition/>  
                        </Grid.RowDefinitions>  
  
                        <!-- Connecting Lines -->  
                        <Rectangle x:Name="HorLn"Margin="9,1,0,0"Height="1"Stroke="Gray"SnapsToDevicePixels="True"/>  
                        <Rectangle x:Name="VerLn"Margin="0,0,1,0"Width="1"Stroke="Gray"SnapsToDevicePixels="true"Grid.RowSpan="2"/>  
                        <ToggleButton x:Name="Expander"Margin="-1,0,0,0"Style="{StaticResource ExpandCollapseToggleStyle}"IsChecked="{Binding Path=IsExpanded, RelativeSource={RelativeSource TemplatedParent}}"/>  
                        <Border Name="Bd"Grid.Column="1"Background="{TemplateBinding Background}"BorderBrush="{TemplateBinding BorderBrush}"BorderThickness="{TemplateBinding BorderThickness}"Padding="{TemplateBinding Padding}"SnapsToDevicePixels="True">  
                            <ContentPresenter x:Name="PART\_Header"ContentSource="ToolTip"HorizontalAlignment="{TemplateBinding HorizontalContentAlignment}"MinWidth="20"/>  
                        </Border>  
                        <ItemsPresenter x:Name="ItemsHost"Grid.Row="1"Grid.Column="1"Grid.ColumnSpan="2"/>  
                    </Grid>  
                      
                    <ControlTemplate.Triggers>  
                        <!-- This trigger changes the connecting lines if the item is the last in the list -->  
                        <DataTrigger Binding="{Binding RelativeSource={RelativeSource Self}, Converter={StaticResource LineConverter}}"Value="true">  
                            <Setter TargetName="VerLn"Property="Height"Value="9"/>  
                            <Setter TargetName="VerLn"Property="VerticalAlignment"Value="Top"/>  
                        </DataTrigger>  
                        <Trigger Property="IsExpanded"Value="false">  
                            <Setter TargetName="ItemsHost"Property="Visibility"Value="Collapsed"/>  
                        </Trigger>  
                        <Trigger Property="HasItems"Value="false">  
                            <Setter TargetName="Expander"Property="Visibility"Value="Hidden"/>  
                        </Trigger>  
                        <MultiTrigger>  
                            <MultiTrigger.Conditions>  
                                <Condition Property="HasHeader"Value="false"/>  
                                <Condition Property="Width"Value="Auto"/>  
                            </MultiTrigger.Conditions>  
                            <Setter TargetName="PART\_Header"Property="MinWidth"Value="75"/>  
                        </MultiTrigger>  
                        <MultiTrigger>  
                            <MultiTrigger.Conditions>  
                                <Condition Property="HasHeader"Value="false"/>  
                                <Condition Property="Height"Value="Auto"/>  
                            </MultiTrigger.Conditions>  
                            <Setter TargetName="PART\_Header"Property="MinHeight"Value="19"/>  
                        </MultiTrigger>  
                        <Trigger Property="IsSelected"Value="true">  
                            <Setter TargetName="Bd"Property="Background"Value="{DynamicResource {x:Static SystemColors.HighlightBrushKey}}"/>  
                            <Setter Property="Foreground"Value="{DynamicResource {x:Static SystemColors.HighlightTextBrushKey}}"/>  
                        </Trigger>  
                        <MultiTrigger>  
                            <MultiTrigger.Conditions>  
                                <Condition Property="IsSelected"Value="true"/>  
                                <Condition Property="IsSelectionActive"Value="false"/>  
                            </MultiTrigger.Conditions>  
                            <Setter TargetName="Bd"Property="Background"Value="Green"/>  
                            <Setter Property="Foreground"Value="White"/>  
                        </MultiTrigger>  
                        <Trigger Property="IsEnabled"Value="false">  
                            <Setter Property="Foreground"Value="{DynamicResource {x:Static SystemColors.GrayTextBrushKey}}"/>  
                        </Trigger>  
                    </ControlTemplate.Triggers>  
                </ControlTemplate>  
            </Setter.Value>  
        </Setter>  
    </Style>  
          
</ResourceDictionary>

## 组织结构树视图风格代码

<ResourceDictionary   
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"  
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"  
  xmlns:local="clr-namespace:MemberTree">  
  
    <local:HorzLineConv x:Key="horzLineConv"/>  
    <local:VertLineConv x:Key="vertLineConv"/>  
  
    <!-- Toggle Button -->  
    <Style x:Key="ExpandCollapseToggleStyle"TargetType="ToggleButton">  
        <Setter Property="Focusable"Value="False"/>  
        <Setter Property="Template">  
            <Setter.Value>  
                <ControlTemplate TargetType="ToggleButton">  
                    <Grid Width="15"Height="13"SnapsToDevicePixels="True">  
                        <Rectangle Width="9"Height="9"Stroke="#919191"SnapsToDevicePixels="true">  
                            <Rectangle.Fill>  
                                <LinearGradientBrush EndPoint="0.5,2"StartPoint="0.5,0">  
                                    <GradientStop Color="White"Offset="0"/>  
                                    <GradientStop Color="Silver"Offset="0.5"/>  
                                    <GradientStop Color="LightGray"Offset="1"/>  
                                </LinearGradientBrush>  
                            </Rectangle.Fill>  
                        </Rectangle>  
                        <Rectangle x:Name="ExpandPath"Width="1"Height="5"Stroke="Black"SnapsToDevicePixels="true"/>  
                        <Rectangle Width="5"Height="1"Stroke="Black"SnapsToDevicePixels="true"/>  
                    </Grid>  
                    <ControlTemplate.Triggers>  
                        <Trigger Property="IsChecked"Value="True">  
                            <Setter Property="Visibility"  TargetName="ExpandPath"Value="Collapsed"/>  
                        </Trigger>  
                    </ControlTemplate.Triggers>  
                </ControlTemplate>  
            </Setter.Value>  
        </Setter>  
    </Style>  
  
    <Style TargetType="TreeViewItem">  
          
        <Style.Resources>  
            <LinearGradientBrush x:Key="ItemAreaBrush"StartPoint="0.5, 0"EndPoint="0.5, 1">  
                <GradientStop Color="#66000000"Offset="0" />  
                <GradientStop Color="#22000000"Offset="1" />  
            </LinearGradientBrush>  
            <LinearGradientBrush x:Key="SelectedItemAreaBrush"StartPoint="0.5, 0"EndPoint="0.5, 1">  
                <GradientStop Color="Orange"Offset="0" />  
                <GradientStop Color="OrangeRed"Offset="1" />  
            </LinearGradientBrush>  
            <LinearGradientBrush x:Key="ItemBorderBrush"StartPoint="0.5, 0"EndPoint="0.5, 1">  
                <GradientStop Color="LightGray"Offset="0" />  
                <GradientStop Color="Gray"Offset="1" />  
            </LinearGradientBrush>  
            <LinearGradientBrush x:Key="SelectedItemBorderBrush"StartPoint="0.5, 0"EndPoint="0.5, 1">  
                <GradientStop Color="Yellow"Offset="0" />  
                <GradientStop Color="Black"Offset="1" />  
            </LinearGradientBrush>  
            <DropShadowBitmapEffect x:Key="DropShadowEffect" />  
        </Style.Resources>  
  
        <Setter Property="Template">  
            <Setter.Value>  
                <ControlTemplate TargetType="TreeViewItem">  
                    <Grid>  
                        <!-- Main Grid-->  
                        <Grid.RowDefinitions>  
                            <RowDefinition Height="Auto"/>  
                            <!-- Horizontal line-->  
                            <RowDefinition Height="Auto"/>  
                            <!--The top row contains the item's content.-->  
                            <RowDefinition Height="\*" />  
                            <!-- Item presenter(children) -->  
                        </Grid.RowDefinitions>  
  
                        <Grid Grid.Row="0">  
                            <!-- Horizontal line grid -->  
                            <Grid.ColumnDefinitions>  
                                <ColumnDefinition Width="\*"/>  
                                <ColumnDefinition Width="\*"/>  
                            </Grid.ColumnDefinitions>  
  
                            <!-- Horizontal line to the left -->  
                            <Rectangle  Grid.Column="0"HorizontalAlignment="Stretch"Stroke="Black"SnapsToDevicePixels="true">  
                                <Rectangle.Height>  
                                    <Binding Mode="OneWay"Converter="{StaticResource horzLineConv}"ConverterParameter="left"   
                                              RelativeSource="{RelativeSource AncestorLevel=1, AncestorType={x:Type TreeViewItem}}"/>  
                                </Rectangle.Height>  
                            </Rectangle>  
  
                            <!-- Horizontal line to the right -->  
                            <Rectangle Grid.Column="1"HorizontalAlignment="Stretch"Stroke="Black"SnapsToDevicePixels="true">  
                                <Rectangle.Height>  
                                    <Binding Mode="OneWay"Converter="{StaticResource horzLineConv}"ConverterParameter="right"   
                                             RelativeSource="{RelativeSource AncestorLevel=1, AncestorType={x:Type TreeViewItem}}"/>  
                                </Rectangle.Height>  
                            </Rectangle>  
                        </Grid>  
                        <!-- End of Horizontal line grid -->  
  
                        <Grid Grid.Row="1">  
                            <!-- Header grid -->  
                            <Grid.RowDefinitions>  
                                <RowDefinition Height="Auto"/>  
                                <!-- Vert. line above node    -->  
                                <RowDefinition Height="\*"/>  
                                <!-- Header -->  
                                <RowDefinition Height="Auto"/>  
                                <!-- Vert line below node    -->  
                            </Grid.RowDefinitions>  
                            <!-- Vertical line above node -->  
                            <Rectangle Grid.Row="0"Height="10"Stroke="Black"SnapsToDevicePixels="true">  
                                <Rectangle.Width>  
                                    <Binding Mode="OneWay"Converter="{StaticResource vertLineConv}"ConverterParameter="top"   
                                             RelativeSource="{RelativeSource AncestorLevel=1, AncestorType={x:Type TreeViewItem}}"/>  
                                </Rectangle.Width>  
                            </Rectangle>  
  
                            <!-- Header -->  
                            <Border Grid.Row="1"Name="Bd"Background="{StaticResource ItemAreaBrush}"BorderBrush="{StaticResource ItemBorderBrush}"   
                                    BorderThickness="0.6"CornerRadius="8"Padding="6"Width="60">  
                                <ContentPresenter ContentSource="Header"HorizontalAlignment="Center"VerticalAlignment="Center" />  
                            </Border>  
  
                            <!-- Vertical line below node -->  
                            <Rectangle Grid.Row="2"  Height="10"Stroke="Black"SnapsToDevicePixels="true">  
                                <Rectangle.Width>  
                                    <Binding Mode="OneWay"Converter="{StaticResource vertLineConv}"ConverterParameter="bottom"   
                                             RelativeSource="{RelativeSource  AncestorLevel=1, AncestorType={x:Type TreeViewItem}}"/>  
                                </Rectangle.Width>  
                            </Rectangle>  
                            <ToggleButton x:Name="Expander"Grid.Row="2"  Style="{StaticResource ExpandCollapseToggleStyle}"   
                                          IsChecked="{Binding Path=IsExpanded, RelativeSource={RelativeSource TemplatedParent}}"/>  
                        </Grid>  
                        <!-- End of Header grid -->  
  
                        <ItemsPresenter x:Name="ItemsHost"  Grid.Row="2"/>  
                        <!-- Children -->  
  
                    </Grid>  
                    <!-- End of Main grid -->  
  
                    <ControlTemplate.Triggers>  
                        <Trigger Property="IsExpanded"Value="false">  
                            <Setter TargetName="ItemsHost"Property="Visibility"Value="Collapsed"/>  
                        </Trigger>  
                        <Trigger Property="HasItems"Value="false">  
                            <Setter TargetName="Expander"Property="Visibility"Value="Hidden"/>  
                        </Trigger>  
                        <!--When the item is selected in the TreeView, use the "selected" colors and give it a drop shadow. -->  
                        <Trigger Property="IsSelected"Value="True">  
                            <Setter TargetName="Bd"Property="Panel.Background"Value="{StaticResource SelectedItemAreaBrush}" />  
                            <Setter TargetName="Bd"Property="Border.BorderBrush"Value="{StaticResource SelectedItemBorderBrush}" />  
                            <Setter TargetName="Bd"Property="TextElement.Foreground"Value="{DynamicResource {x:Static SystemColors.HighlightTextBrushKey}}" />  
                            <Setter TargetName="Bd"Property="Border.BitmapEffect"Value="{StaticResource DropShadowEffect}" />  
                        </Trigger>  
                    </ControlTemplate.Triggers>  
                </ControlTemplate>  
            </Setter.Value>  
        </Setter>  
  
        <Setter Property="ItemsPanel">  
            <Setter.Value>  
                <ItemsPanelTemplate>  
                    <StackPanel HorizontalAlignment="Center"IsItemsHost="True"Orientation="Horizontal"/>  
                </ItemsPanelTemplate>  
            </Setter.Value>  
        </Setter>  
    </Style>  
</ResourceDictionary>

## 自定义树控件界面布局代码

<?xml version="1.0" encoding="utf-8"?>  
<UserControl  
    x:Class="MemberTree.MyTreeView"xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation" xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006" xmlns:d="http://schemas.microsoft.com/expression/blend/2008" xmlns:local="clr-namespace:MemberTree"  
    mc:Ignorable="d"  
    d:DesignHeight="500"  
    d:DesignWidth="700">  
    <Grid>  
        <Grid.RowDefinitions>  
            <RowDefinition  
                Height="Auto" />  
            <RowDefinition />  
        </Grid.RowDefinitions>  
        <StackPanel  
            Orientation="Horizontal">  
            <Button  
                x:Name="btnUpLevelNode"  
                Content="▲显示上一级节点"  
                Click="btnUpLevelNode\_Click"  
                Margin="5,0"  
                IsEnabled="False" />  
            <Button  
                x:Name="btnAllNode"  
                Content="显示所有树节点"  
                Click="btnAllNode\_Click"  
                Margin="5,0" />  
            <CheckBox  
                x:Name="isAutoExpand"  
                Content="鼠标滑过自动展开节点"  
                Margin="5" />  
            <Menu  
                x:Name="nodeContextMenu"Margin="5">  
                <MenuItem  
                    Header="展开节点子项">  
                    <MenuItem  
                        Header="展开5级子项"  
                        Tag="5"  
                        Click="item\_Expand" />  
                    <MenuItem  
                        Header="展开10级子项"  
                        Tag="10"  
                        Click="item\_Expand" />  
                    <MenuItem  
                        Header="展开20级子项"  
                        Tag="20"  
                        Click="item\_Expand" />  
                    <MenuItem  
                        Header="展开50级子项"  
                        Tag="50"  
                        Click="item\_Expand" />  
                    <MenuItem  
                        Header="展开100级子项"  
                        Tag="100"  
                        Click="item\_Expand" />  
                    <MenuItem  
                        Header="展开所有子项"  
                        Tag="100000"  
                        Click="item\_Expand" />  
                </MenuItem>  
            </Menu>  
        </StackPanel>  
        <ScrollViewer  
            Grid.Row="1"  
            HorizontalScrollBarVisibility="Auto"  
            VerticalScrollBarVisibility="Auto">  
            <TreeView  
                x:Name="memberTreeView">  
                <TreeView.Resources>  
                    <ResourceDictionary  
                        Source="LineTreeStyle.xaml" />  
                </TreeView.Resources>  
            </TreeView>  
        </ScrollViewer>  
    </Grid>  
</UserControl>

## 自定义树控件后台处理逻辑代码

**namespace** MemberTree  
{  
    ///<**summary**>  
    /// MyTreeView.xaml 的交互逻辑  
    ///<**/summary**>  
    **public**partialclassMyTreeView : UserControl  
    {  
        **public**MyTreeView()  
        {  
            **InitializeComponent**();  
              
            //让鼠标在TreeView滚动滚轮时ScrollViewer能够滚动  
            *memberTreeView*.PreviewMouseWheel +=    (sender, e) => {  
                var eventArg = **new**MouseWheelEventArgs(e.MouseDevice, e.Timestamp, e.Delta);  
                eventArg.RoutedEvent = UIElement.*MouseWheelEvent*;  
                eventArg.Source = sender;  
                **this**.*memberTreeView*.**RaiseEvent**(eventArg);  
            };  
        }  
  
        **private**staticList<string>*ringNodeIds* = **new**List<string>();  
        //判断闭环是否关闭  
        **private**static**boolisRingClose**(string id)  
        {  
            **if** (MyTreeNode.RingNodes.**ContainsKey**(id))  
            {  
                **if** (*ringNodeIds*.**Contains**(id))  
                {  
                    return**true**;  
                }  
                **else**  
                {  
                    *ringNodeIds*.**Add**(id);  
                }  
            }  
            return**false**;  
        }  
  
        **private**static**int***levels* = 0;  
        **public**void**ExpandAllNodes**(**int** maxLevel)  
        {  
            *levels* = 0;  
            **ExpandAllNodesIml**(*memberTreeView*.Items[0] **as**TreeViewItem, maxLevel);  
        }  
        **public**void**ExpandAllNodesIml**(TreeViewItem item, **int** maxLevel)  
        {  
            **if** (*levels*< maxLevel)  
            {  
                *levels*++;  
                **foreach** (TreeViewItem subItem **in** item.Items)  
                {  
                    item.IsExpanded = **true**;  
                    **ExpandAllNodesIml**(subItem, maxLevel);  
                }  
                *levels*--;  
            }  
        }  
  
        **private**TreeViewItem**NewTreeViewItem**(MyTreeNode node)  
        {  
            TreeViewItem item = **new**TreeViewItem();  
            item.Header = node.SysId;  
            item.ToolTip = node.RealName +"（"+ node.SysId + "）层级:" + node.Level + ",下线人数:" + node.DescendantCount + ",下线层数:" + node.DescendantLevels + "，父Id:" + node.TopId + "，行数:" + node.LineCount;  
            item.Tag = node;  
            item.MouseEnter += **item\_MouseEnter**;  
            //item.MouseLeave += item\_MouseLeave;  
            //item.MouseMove += item\_MouseMove;  
            return item;  
        }  
  
        void**item\_Expanded**(object sender, RoutedEventArgs e)  
        {  
            TreeViewItem item = e.Source **as**TreeViewItem;  
            **foreach** (TreeViewItem subItem **in** item.Items)  
            {  
                MyTreeNode node = subItem.Tag **as**MyTreeNode;  
                **if** (**isRingClose**(node.SysId))  
                {  
                    return;  
                }  
                **if** (!subItem.IsExpanded)  
                {  
                    **if** (node != **null**)  
                    {  
                        List<MyTreeNode> childrenNodes = node.ChildrenNodes;  
                        **foreach** (MyTreeNode subNode **in** childrenNodes)  
                        {  
                            TreeViewItem grandson = **NewTreeViewItem**(subNode);  
                            subItem.Items.**Add**(grandson);  
                        }  
                    }  
                }  
            }  
        }  
  
        void**item\_Collapsed**(object sender, RoutedEventArgs e)  
        {  
            TreeViewItem item = e.Source **as**TreeViewItem;  
            **foreach** (TreeViewItem subItem **in** item.Items)  
            {  
                **if** (!subItem.IsExpanded)  
                {  
                    subItem.Items.**Clear**();  
                }  
            }  
        }  
  
        void**item\_MouseEnter**(object sender, MouseEventArgs e)  
        {  
            **if** (*isAutoExpand*.IsChecked == **true**)  
            {  
                TreeViewItem item = sender **as**TreeViewItem;  
                //Console.WriteLine("===============鼠标进入了" + item.Header);  
                item.IsExpanded = **true**;  
                e.Handled = **true**;  
            }  
        }  
  
        **bool***isCurrentLeave* = **false**;  
        void**item\_MouseMove**(object sender, MouseEventArgs e)  
        {  
            **if** (*isAutoExpand*.IsChecked == **true**)  
            {  
                TreeViewItem item = sender **as**TreeViewItem;  
                //Console.WriteLine("$$$$$$$$$$鼠标移动了"+item.Header);  
                *isCurrentLeave* = **true**;  
                e.Handled = **true**;  
            }  
        }  
       
        void**item\_MouseLeave**(object sender, MouseEventArgs e)  
        {  
            **if** (*isAutoExpand*.IsChecked == **true**&&*isCurrentLeave*)  
            {  
                TreeViewItem item = sender **as**TreeViewItem;  
                //Console.WriteLine(".......................鼠标退出了" + item.Header);  
                item.IsExpanded = **false**;  
                *isCurrentLeave* = **false**;  
            }  
        }  
  
        //显示上一级节点  
        **private**void**btnUpLevelNode\_Click**(object sender, RoutedEventArgs e)  
        {  
            TreeViewItem oldRootItem = *memberTreeView*.Items[0] **as**TreeViewItem;  
            MyTreeNode oldRootNode = oldRootItem.Tag **as**MyTreeNode;  
            MyTreeNode newRooNode = oldRootNode.ParentNode;  
            **if** (**isRingClose**(newRooNode.SysId))  
            {  
                return;  
            }  
  
            //先移除旧的根节点  
            oldRootItem.Expanded -= **item\_Expanded**;  
            oldRootItem.Collapsed -= **item\_Collapsed**;  
            *memberTreeView*.Items.**Remove**(oldRootItem);  
  
            //添加新的根节点  
            TreeViewItem newRootItem = **NewTreeViewItem**(newRooNode);  
            newRootItem.IsExpanded = **true**;  
            newRootItem.Expanded += **item\_Expanded**;  
            newRootItem.Collapsed += **item\_Collapsed**;  
            *memberTreeView*.Items.**Add**(newRootItem);  
  
            //新的根节点添加子节点  
            List<MyTreeNode> childrenNodes = newRooNode.ChildrenNodes;  
            **bool** hasNotAdded = **true**;  
            **foreach** (MyTreeNode subNode **in** childrenNodes)  
            {  
                **if** (hasNotAdded)  
                {  
                    **if** (oldRootNode.SysId == subNode.SysId)  
                    {  
                        newRootItem.Items.**Add**(oldRootItem);  
                        hasNotAdded = **false**;  
                        continue;  
                    }  
                }  
                TreeViewItem subItem = **NewTreeViewItem**(subNode);  
                newRootItem.Items.**Add**(subItem);  
  
                List<MyTreeNode> grandNodes = subNode.ChildrenNodes;  
                **foreach** (MyTreeNode grandNode **in** grandNodes)  
                {  
                    TreeViewItem grandItem = **NewTreeViewItem**(grandNode);  
                    subItem.Items.**Add**(grandItem);  
                }  
            }  
  
            //判断当前根节点是否存在父节点  
            **if** (newRooNode.ParentNode != **null**)  
            {  
                *btnUpLevelNode*.IsEnabled = **true**;  
            }  
            **else**  
            {  
                *btnUpLevelNode*.IsEnabled = **false**;  
            }  
        }  
  
        **public**void**SetRootNode**(MyTreeNode rootNode)  
        {  
            *memberTreeView*.Items.**Clear**();  
            *ringNodeIds*.**Clear**();  
  
            **if** (rootNode != **null**)  
            {  
                TreeViewItem rootItem = **NewTreeViewItem**(rootNode);  
                rootItem.Expanded += **item\_Expanded**;  
                rootItem.Collapsed += **item\_Collapsed**;  
  
                *memberTreeView*.Items.**Add**(rootItem);  
                List<MyTreeNode> childrenNodes = rootNode.ChildrenNodes;  
                **foreach** (MyTreeNode subNode **in** childrenNodes)  
                {  
                    TreeViewItem subItem = **NewTreeViewItem**(subNode);  
                    rootItem.Items.**Add**(subItem);  
                }  
  
                //判断当前根节点是否存在父节点  
                **if** (rootNode.ParentNode != **null**)  
                {  
                    *btnUpLevelNode*.IsEnabled = **true**;  
                }  
                **else**  
                {  
                    *btnUpLevelNode*.IsEnabled = **false**;  
                }  
            }  
        }  
  
        //获取选中的节点  
        **public**MyTreeNode**GetSelectedNode**()  
        {  
            TreeViewItem selectedItem = *memberTreeView*.SelectedItem **as**TreeViewItem;  
            **if** (selectedItem != **null**)  
            {  
                MyTreeNode node = (*memberTreeView*.SelectedItem **as**TreeViewItem).Tag **as**MyTreeNode;  
                **if** (node != **null**)  
                {  
                    return node;  
                      
                }  
            }  
  
            return**null**;  
        }  
  
        **private**void**btnAllNode\_Click**(object sender, RoutedEventArgs e)  
        {  
            **SetRootNode**(MyTreeNode.RootNode);  
  
            **if** (MyTreeNode.NoParentNodes.Count >0)  
            {  
                TreeViewItem treeItem = **new**TreeViewItem();  
                treeItem.Header = "森林（共" + MyTreeNode.NoParentNodes.Count + "棵树)";  
                treeItem.ToolTip = "森林（共" + MyTreeNode.NoParentNodes.Count + "棵树)";  
                treeItem.Expanded += **item\_Expanded**;  
                treeItem.Collapsed += **item\_Collapsed**;  
                *memberTreeView*.Items.**Add**(treeItem);  
                **foreach** (MyTreeNode node **in**MyTreeNode.NoParentNodes)  
                {  
                    TreeViewItem subItem = **NewTreeViewItem**(node);  
                    treeItem.Items.**Add**(subItem);  
                }  
            }  
  
            **if** (MyTreeNode.NodeInfoErrNodes.Count >0)  
            {  
                TreeViewItem nodeInfoErrItem = **new**TreeViewItem();  
                nodeInfoErrItem.Header = "节点信息不完整的节点（共" + MyTreeNode.**GetNodeInfoErrCount**() + "个)";  
                nodeInfoErrItem.ToolTip = "节点信息不完整的节点（共" + MyTreeNode.**GetNodeInfoErrCount**() + "个)";  
                nodeInfoErrItem.Expanded += **item\_Expanded**;  
                nodeInfoErrItem.Collapsed += **item\_Collapsed**;  
                *memberTreeView*.Items.**Add**(nodeInfoErrItem);  
                **foreach** (MyTreeNode node **in**MyTreeNode.NodeInfoErrNodes)  
                {  
                    TreeViewItem subItem = **NewTreeViewItem**(node);  
                    nodeInfoErrItem.Items.**Add**(subItem);  
                }  
            }  
  
            **if** (MyTreeNode.RingNodes.Count >0)  
            {  
                TreeViewItem ringErrItem = **new**TreeViewItem();  
                ringErrItem.Header = "形成闭环的节点（共" + MyTreeNode.RingNodes.**Count**() + "个)";  
                ringErrItem.ToolTip = "形成闭环的节点（共" + MyTreeNode.RingNodes.**Count**() + "个)";  
                //nodeInfoErrItem.Expanded += item\_Expanded;  
                //nodeInfoErrItem.Collapsed += item\_Collapsed;  
                *memberTreeView*.Items.**Add**(ringErrItem);  
                **foreach** (MyTreeNode node **in**MyTreeNode.RingNodes.Values)  
                {  
                    TreeViewItem subItem = **NewTreeViewItem**(node);  
                    ringErrItem.Items.**Add**(subItem);  
                }  
            }  
  
            *btnUpLevelNode*.IsEnabled = **false**;  
        }  
          
  
        //展开选中项的子项  
        **private**void**item\_Expand**(object sender, RoutedEventArgs e)  
        {  
            TreeViewItem treeItem = *memberTreeView*.SelectedItem **as**TreeViewItem;  
            **if** (treeItem != **null**)  
            {  
                MenuItem menu = sender **as**MenuItem;  
                **int** expLevel = **int**.**Parse**(menu.Tag.**ToString**());  
                **if**(expLevel >10)  
                {  
                    string warnTxt = "你确定要一次性展开"+expLevel+"层子项吗？\n展开层级过大可能会由于数据量太大而造成程序卡死。";  
                    **MessageBoxResult** result = MessageBox.**Show**(warnTxt,"警告",**MessageBoxButton**.*YesNo*);  
                    **if**(result == **MessageBoxResult**.*No*)  
                    {  
                        return;  
                    }  
                }  
                  
                *levels* = 0;  
                **ExpandAllNodesIml**(treeItem, expLevel);  
            }  
        }  
    }  
}

# 读取Excel和csv数据源模块

## 读取Excel文件

**public**staticvoid**OpenExcelFile**(string filepath, **int** upperLower, **int** DBCSBC, **int** trim)  
{  
    **ClearAllNodes**();  
  
    FileInfo excelFile = **new**FileInfo(filepath);  
    **if** (!excelFile.Exists)  
    {  
        MessageBox.**Show**("文件" + filepath + "不存在！");  
    }  
    **else**  
    {  
        **int** row = 2;  
        **try**  
        {  
            MainWindow.*notify*.**SetProcessBarVisible**(**true**);  
            MainWindow.*notify*.**SetStatusMessage**("正在读取Excel文件......");  
  
            **using** (ExcelPackage package = **new**ExcelPackage(excelFile))  
            {  
                ExcelWorksheet ws = package.Workbook.Worksheets[1];  
                **if** (ws.Cells[1,1].Text.**ToLower**() != "sysid"||  
                   ws.Cells[1,2].Text.**ToLower**() != "realname"||  
                   ws.Cells[1,3].Text.**ToLower**() != "topid"||  
                   ws.Cells[1,4].Text.**ToLower**() != "layer")  
                {  
                        MessageBox.**Show**("文件格式不正确，第一行必须由sysid,realname,topid,layer四个标题头组成！");  
                    return;  
                }  
  
                MainWindow.*notify*.**SetStatusMessage**("开始初始化数据结构......");  
                //先将总行数读出来  
                *allNodesCount* = ws.Dimension.End.Row - 1;  
  
                **int** step = *allNodesCount*>100 ? *allNodesCount* / 100 : 1;  
                **for** (**int** i = 0; i <*allNodesCount*; i++)  
                {  
                    string sysId = ws.Cells[row, 1].Text;  
                    string realName = ws.Cells[row, 2].Text;  
                    string topId = ws.Cells[row, 3].Text;  
                    string lyr = ws.Cells[row, 4].Text;  
  
                    MyTreeNode myNode = **new**MyTreeNode(sysId, realName, topId, lyr, row, upperLower, DBCSBC, trim);  
  
                    **if** (*allNodes*.**ContainsKey**(myNode.SysId))  
                    {  
                        *idConflictNodes*.**Add**(myNode);  
                    }  
                    **else**  
                    {  
                        *allNodes*.**Add**(myNode.SysId, myNode);  
                    }  
  
                    row++;  
                    **if** (row % step == 0)  
                    {  
                        MainWindow.*notify*.**SetProcessBarValue**((**int**)(100.0 \* row / *allNodesCount*));  
                        MainWindow.*notify*.**SetStatusMessage**("正在处理生成第" + row + "个节点（总共" + *allNodesCount* + "个节点）");  
                    }  
                }  
  
                row = 0;  
                **foreach** (MyTreeNode node **in***allNodes*.Values)  
                {  
                    //将节点加入树中合适的位置去  
                    **ConstructTree**(node);  
  
                    row++;  
                    **if** (row % step == 0)  
                    {  
                        MainWindow.*notify*.**SetProcessBarValue**((**int**)(100.0 \* row / *allNodesCount*));  
                        MainWindow.*notify*.**SetStatusMessage**("正在构造树结构，加载第" + row + "个节点（总共" + *allNodesCount* + "个节点）");  
                    }  
                }  
                **foreach** (MyTreeNode node **in***idConflictNodes*)  
                {  
                    //将节点加入树中合适的位置去  
                    **ConstructTree**(node);  
                }  
  
                MainWindow.*notify*.**SetStatusMessage**("");  
            }  
        }  
        **catch** (Exception ex)  
        {  
            MainWindow.*notify*.**SetStatusMessage**("发生异常：" + ex.Message + "在第" + row + "行!");  
        }  
        MainWindow.*notify*.**SetProcessBarVisible**(**false**);  
    }  
}

## 读取CSV文件

**public**staticvoid**OpenCSVFile**(string filepath, **int** upperLower, **int** DBCSBC, **int** trim)  
{  
    **ClearAllNodes**();  
    Encoding encoding = TextUtilTool.**GetFileEncodeType**(filepath);  
    StreamReader mysr = **new**StreamReader(filepath, encoding);  
    string[] allLines = **null**;  
    **try**  
    {  
        MainWindow.*notify*.**SetProcessBarVisible**(**true**);  
        MainWindow.*notify*.**SetStatusMessage**("开始读取文件......");  
  
        string firstLine = mysr.**ReadLine**(); //第一行是表头，读取之后不处理，直接跳过  
        **if** (firstLine.**ToLower**() != "sysid,realname,topid,layer")  
        {  
            MessageBox.**Show**("文件格式不正确，第一行必须由sysid,realname,topid,layer四个标题头组成！");  
            return;  
        }  
  
        string allStr = mysr.**ReadToEnd**();  
        string[] newLineStr = **new**string[] { Environment.NewLine };  
        allLines = allStr.**Split**(newLineStr, **StringSplitOptions**.*RemoveEmptyEntries*);  
    }  
    **catch** (Exception ex)  
    {  
        MainWindow.*notify*.**SetStatusMessage**("文件读取出错！+\n" + ex.Message);  
    }  
    **finally**  
    {  
        mysr.**Close**();  
    }  
  
    **int** row = 1;  
    *allNodesCount* = allLines.Length;  
    **int** step = *allNodesCount*>100 ? *allNodesCount* / 100 : 1;  
    **try**  
    {  
        **foreach** (string line **in** allLines)  
        {  
            string[] aryline = line.**Split**(**newchar**[] { ',' });  
            MyTreeNode myNode = **new**MyTreeNode(aryline[0], aryline[1], aryline[2], aryline[3], row + 1, upperLower, DBCSBC, trim);  
  
            **if** (*allNodes*.**ContainsKey**(myNode.SysId))  
            {  
                *idConflictNodes*.**Add**(myNode);  
            }  
            **else**  
            {  
                *allNodes*.**Add**(myNode.SysId, myNode);  
            }  
  
            row++;  
            **if** (row % step == 0)  
            {  
                MainWindow.*notify*.**SetProcessBarValue**((**int**)(100.0 \* row / *allNodesCount*));  
                MainWindow.*notify*.**SetStatusMessage**("正在处理生成第" + row + "个节点（总共" + *allNodesCount* + "个节点）");  
            }  
        }  
  
        row = 0;  
        **foreach** (MyTreeNode node **in***allNodes*.Values)  
        {  
            //将节点加入树中合适的位置去  
            **ConstructTree**(node);  
  
            row++;  
            **if** (row % step == 0)  
            {  
                MainWindow.*notify*.**SetProcessBarValue**((**int**)(100.0 \* row / *allNodesCount*));  
                MainWindow.*notify*.**SetStatusMessage**("正在构造树结构，加载第" + row + "个节点（总共" + *allNodesCount* + "个节点）");  
            }  
        }  
        **foreach** (MyTreeNode node **in***idConflictNodes*)  
        {  
            //将节点加入树中合适的位置去  
            **ConstructTree**(node);  
        }  
  
        MainWindow.*notify*.**SetStatusMessage**("");  
    }  
    **catch** (Exception ex)  
    {  
        MainWindow.*notify*.**SetStatusMessage**("发生异常：" + ex.Message + "在第" + row + "行!");  
    }  
    MainWindow.*notify*.**SetProcessBarVisible**(**false**);  
}

# 读取sqlserver数据源模块

## 读取数据库界面布局代码

<?xml version="1.0" encoding="utf-8"?>  
<Window  
    x:Class="MemberTree.ConnDBWindow"xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation" xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"  
    WindowStartupLocation="CenterScreen"  
    WindowStyle="ToolWindow"  
    Title="连接到SQL Server 2005"  
    Height="300"  
    Width="500">  
    <Grid  
        RenderTransformOrigin="0.5,0.5">  
        <Grid.ColumnDefinitions>  
            <ColumnDefinition  
                Width="0.248979591836734\*" />  
            <ColumnDefinition  
                Width="0.255102040816328\*" />  
            <ColumnDefinition  
                Width="0.248979591836735\*" />  
            <ColumnDefinition  
                Width="0.246938775510203\*" />  
        </Grid.ColumnDefinitions>  
        <Grid.RowDefinitions>  
            <RowDefinition  
                Height="0.156716417910447\*" />  
            <RowDefinition  
                Height="0.156716417910449\*" />  
            <RowDefinition  
                Height="0.16044776119403\*" />  
            <RowDefinition  
                Height="0.152985074626865\*" />  
            <RowDefinition  
                Height="0.171641791044775\*" />  
            <RowDefinition  
                Height="0.111940298507462\*" />  
            <RowDefinition  
                Height="0.0895522388059698\*" />  
        </Grid.RowDefinitions>  
        <TextBlock  
            Text="服务器IP或机器名："  
            Height="17"  
            Width="108"  
            Margin="0,12,3,0"  
            VerticalAlignment="Top"  
            HorizontalAlignment="Right"  
            Grid.Row="0"  
            Grid.Column="0" />  
        <TextBlock  
            Text="用户名："  
            Height="16"  
            Width="56"  
            Margin="52,16.8366666666666,0,0"  
            VerticalAlignment="Top"  
            HorizontalAlignment="Left"  
            Grid.Row="1"  
            Grid.Column="0" />  
        <TextBlock  
            Text="数据库名："  
            Height="16"  
            Width="60"  
            Margin="50,0,0,11"  
            VerticalAlignment="Bottom"  
            HorizontalAlignment="Left"  
            Grid.Row="2"  
            Grid.Column="0" />  
        <TextBlock  
            Text="密码："  
            Height="16"  
            Width="55"  
            Margin="0,16.8366666666666,6,0"  
            VerticalAlignment="Top"  
            HorizontalAlignment="Right"  
            Grid.Row="1"  
            Grid.Column="2" />  
        <TextBlock  
            Text="表名："  
            Height="16"  
            Width="55"  
            Margin="0,0,6,11"  
            VerticalAlignment="Bottom"  
            HorizontalAlignment="Right"  
            Grid.Row="2"  
            Grid.Column="2" />  
        <TextBox  
            Text="."  
            x:Name="txtDBServer"  
            Grid.Column="1"  
            Grid.Row="0"  
            HorizontalAlignment="Right"  
            VerticalAlignment="Top"  
            Margin="0,10.1627083333333,6,0"  
            Width="116"  
            Height="22"  
            IsEnabled="False"  
            IsReadOnly="False" />  
        <RadioButton  
            Content="使用本机"  
            x:Name="rbLocalHost"  
            RenderTransformOrigin="0.4855,0.5"  
            IsChecked="True"  
            Checked="rbLocalHost\_Checked"  
            Grid.Column="2"  
            Grid.Row="0"  
            HorizontalAlignment="Left"  
            VerticalAlignment="Bottom"  
            Margin="23,0,0,13"  
            Width="70"  
            Height="16" />  
        <RadioButton  
            x:Name="rbOtherHost"  
            Checked="rbOtherHost\_Checked"  
            Content="手动输入"  
            Grid.Column="3"  
            Grid.Row="0"  
            HorizontalAlignment="Right"  
            VerticalAlignment="Top"  
            Margin="0,13,12,0"  
            Width="106"  
            Height="16" />  
        <TextBox  
            x:Name="txtUserName"  
            Height="22"  
            Width="116"  
            Margin="0,12,6,0"  
            VerticalAlignment="Top"  
            HorizontalAlignment="Right"  
            Grid.Row="1"  
            Grid.Column="1" />  
        <TextBlock  
            Text="\*sysid列"  
            Grid.Column="0"  
            Grid.Row="3"  
            HorizontalAlignment="Left"  
            VerticalAlignment="Bottom"  
            Margin="58,0,0,10"  
            Width="52"  
            Height="16" />  
        <TextBlock  
            Text="\*topid列"  
            Grid.Column="2"  
            Grid.Row="3"  
            HorizontalAlignment="Left"  
            VerticalAlignment="Bottom"  
            Margin="49,0,0,10"  
            Width="60"  
            Height="16" />  
        <TextBlock  
            Text="realname列"  
            Grid.Column="0"  
            Grid.Row="4"  
            HorizontalAlignment="Left"  
            VerticalAlignment="Bottom"  
            Margin="42,0,0,16"  
            Width="68"  
            Height="16" />  
        <TextBlock  
            Text="layer列"  
            Grid.Column="2"  
            Grid.Row="4"  
            HorizontalAlignment="Left"  
            VerticalAlignment="Bottom"  
            Margin="58.5,0,0,16"  
            Width="60"  
            Height="16" />  
        <PasswordBox  
            x:Name="txtPwd"  
            Height="22"  
            Width="112"  
            Margin="0,12,8,0"  
            VerticalAlignment="Top"  
            HorizontalAlignment="Right"  
            Grid.Row="1"  
            Grid.Column="3" />  
        <ProgressBar  
            Grid.Column="0"  
            Grid.ColumnSpan="4"  
            Grid.Row="6"  
            HorizontalAlignment="Stretch"  
            VerticalAlignment="Stretch"  
            Margin="3,3,3,3"  
            x:Name="prograss"  
            Visibility="Collapsed" />  
        <TextBlock  
            Text=""  
            HorizontalAlignment="Left"  
            VerticalAlignment="Top"  
            Margin="11,6.5,0,0"  
            Width="230"  
            Height="16"  
            Grid.ColumnSpan="2"  
            Grid.Column="0"  
            Grid.Row="5"  
            x:Name="labelMessage" />  
        <ComboBox  
            x:Name="txtSysid"  
            IsEnabled="False"  
            IsReadOnly="False"  
            Grid.Column="1"  
            Grid.Row="3"  
            HorizontalAlignment="Right"  
            VerticalAlignment="Top"  
            Margin="0,12,7,0"  
            Width="112"  
            Height="22">  
            <ComboBox  
                Grid.Column="0"  
                Grid.Row="4"  
                Grid.RowSpan="2"  
                HorizontalAlignment="Right"  
                VerticalAlignment="Top"  
                Margin="0,34,5,0"  
                Width="105"  
                Height="20" />  
        </ComboBox>  
        <ComboBox  
            x:Name="txtRealname"  
            Grid.Column="1"  
            Grid.Row="4"  
            HorizontalAlignment="Right"  
            VerticalAlignment="Top"  
            Margin="0,11,6,0"  
            Width="114"  
            Height="22"  
            IsEnabled="False"  
            IsReadOnly="False" />  
        <ComboBox  
            x:Name="txtTopid"  
            IsEnabled="False"  
            IsReadOnly="False"  
            Grid.Column="3"  
            Grid.Row="3"  
            HorizontalAlignment="Right"  
            VerticalAlignment="Top"  
            Margin="0,12,7,0"  
            Width="112"  
            Height="22" />  
        <ComboBox  
            x:Name="txtLayer"  
            Grid.Column="3"  
            Grid.Row="4"  
            HorizontalAlignment="Right"  
            VerticalAlignment="Top"  
            Margin="0,11,8,0"  
            Width="112"  
            Height="22"  
            IsReadOnly="False"  
            IsEnabled="False" />  
        <TextBox  
            x:Name="txtDBName"  
            Grid.Column="1"  
            Grid.Row="2"  
            HorizontalAlignment="Right"  
            VerticalAlignment="Top"  
            Margin="0,12,5,0"  
            Width="116"  
            Height="22" />  
        <ComboBox  
            x:Name="txtTableName"  
            Grid.Column="3"  
            Grid.Row="2"  
            HorizontalAlignment="Right"  
            VerticalAlignment="Top"  
            Margin="0,12,8,0"  
            Width="112"  
            Height="22"  
            IsReadOnly="False"  
            IsEnabled="False"  
            SelectionChanged="txtTableName\_SelectionChanged">  
            <ComboBox  
                Grid.Column="0"  
                Grid.Row="4"  
                Grid.RowSpan="2"  
                HorizontalAlignment="Right"  
                VerticalAlignment="Top"  
                Margin="0,34,5,0"  
                Width="105"  
                Height="20" />  
        </ComboBox>  
        <Button  
            x:Name="btnConnect"  
            Click="btnConnect\_Click"  
            Content="连接到数据库"  
            Grid.Column="1"  
            Grid.Row="5"  
            HorizontalAlignment="Left"  
            VerticalAlignment="Top"  
            Margin="13,3,0,0"  
            Width="91"  
            Height="23" />  
        <Button  
            x:Name="btnExport"  
            Click="btnExport\_Click"  
            Content="导出数据到csv"  
            Grid.Column="2"  
            Grid.Row="5"  
            HorizontalAlignment="Left"  
            VerticalAlignment="Top"  
            Margin="13,3,0,0"  
            Width="90"  
            Height="23" />  
        <Button  
            x:Name="btnClose"  
            Click="btnClose\_Click"  
            Content="取消"  
            Grid.Column="3"  
            Grid.Row="5"  
            HorizontalAlignment="Left"  
            VerticalAlignment="Top"  
            Margin="13,3,0,0"  
            Width="90"  
            Height="23" />  
    </Grid>  
</Window>

## 连接到数据库业务处理逻辑代码

**namespace** MemberTree  
{  
    ///<**summary**>  
    /// Interaction logic for ConnDBWindow.xaml  
    ///<**/summary**>  
    **public**partialclassConnDBWindow : Window  
    {  
        **public**ConnDBWindow()  
        {  
            **InitializeComponent**();  
        }  
        void**rbLocalHost\_Checked**(object sender, RoutedEventArgs e)  
        {  
            *txtDBServer*.Text= ".";  
            *txtDBServer*.IsEnabled = **false**;  
            *txtDBServer*.Background = Brushes.Gray;  
        }  
          
        void**rbOtherHost\_Checked**(object sender, RoutedEventArgs e)  
        {  
            *txtDBServer*.Text= "";  
            *txtDBServer*.IsEnabled = **true**;  
            *txtDBServer*.Background = Brushes.White;  
        }  
      
        **privateboolCheckTxtBox**(TextBox txtBox, string name)  
        {  
            **if**(txtBox.Text=="")  
            {  
                txtBox.BorderBrush = Brushes.Red;  
                MessageBox.**Show**(name +"不能为空！");  
                return**false**;  
            }  
            return**true**;  
        }  
          
        **privateboolCheckComboBox**(ComboBox txtBox, string name)  
        {  
            **if**(txtBox.SelectedIndex == -1)  
            {  
                txtBox.BorderBrush = Brushes.Red;  
                MessageBox.**Show**(name +"不能为空！");  
                return**false**;  
            }  
            return**true**;  
        }  
          
        **private**SqlConnection**Connect**()  
        {  
            **if**(!**CheckTxtBox**(*txtDBServer*,"服务器名称"))  
            {  
                   return**null**;  
            }  
            **if**(!**CheckTxtBox**(*txtDBName*,"数据库名称"))  
            {  
                   return**null**;  
            }  
            **if**(!**CheckTxtBox**(*txtUserName*,"用户名"))  
            {  
                   return**null**;  
            }  
              
            // 使用一个IntPtr类型值来存储加密字符串的起始点    
            **IntPtr** p = System.Runtime.InteropServices.Marshal.**SecureStringToBSTR**(*txtPwd*.SecurePassword);   
            // 使用.NET内部算法把IntPtr指向处的字符集合转换成字符串    
            string password = System.Runtime.InteropServices.Marshal.**PtrToStringBSTR**(p);  
            **if**(password=="")  
            {  
                *txtPwd*.BorderBrush = Brushes.Red;  
                MessageBox.**Show**("密码不能为空！");  
            }  
          
            string connStr = "server="+ *txtDBServer*.Text  
                + ";database=" + *txtDBName*.Text  
                + ";uid=" + *txtUserName*.Text  
                + ";pwd=" + password;  
            SqlConnection sConn = **new**SqlConnection(connStr);  
            **try**  
            {  
                sConn.**Open**();  
                return sConn;  
            }  
            **catch** (Exception ex)  
            {  
                MessageBox.**Show**("数据库连接错误:" + ex.Message);  
                *txtTableName*.IsEnabled = **false**;  
                *txtSysid*.IsEnabled = **false**;  
                *txtRealname*.IsEnabled = **false**;  
                *txtTopid*.IsEnabled = **false**;  
                *txtLayer*.IsEnabled = **false**;  
            }  
            return**null**;  
        }  
  
        void**btnConnect\_Click**(object sender, RoutedEventArgs e)  
        {  
            *txtTableName*.IsEnabled = **true**;  
            *txtTableName*.Items.**Clear**();  
            SqlConnection sConn = **Connect**();  
            **if**(sConn != **null**)  
            {  
                **try**  
                {  
                    //先查出所有的表名  
                    string sqlStr = "SELECT Name FROM SysObjects Where XType='U' ORDER BY Name";  
                    SqlCommand sCmd = **new**SqlCommand(sqlStr, sConn);  
                    SqlDataReader  sdr = sCmd.**ExecuteReader**();  
                    **while** (sdr.**Read**())  
                    {    
                        *txtTableName*.Items.**Add**(sdr.**GetString**(0));  
                    }  
                    sdr.**Close**();  
                }  
                **catch** (Exception ex)  
                {  
                    MessageBox.**Show**(ex.Message);  
                    *txtTableName*.IsEnabled = **false**;  
                }  
                **finally**  
                {  
                     sConn.**Close**();  
                }  
            }  
        }  
          
        void**btnClose\_Click**(object sender, RoutedEventArgs e)  
        {  
            **this**.**Close**();  
        }  
          
        //表名变化时，更新列名下拉框  
        void**txtTableName\_SelectionChanged**(object sender, SelectionChangedEventArgs e)  
        {  
            *txtSysid*.IsEnabled = **true**;  
            *txtRealname*.IsEnabled = **true**;  
            *txtTopid*.IsEnabled = **true**;  
            *txtLayer*.IsEnabled = **true**;  
            *txtSysid*.Items.**Clear**();  
            *txtRealname*.Items.**Clear**();  
            *txtTopid*.Items.**Clear**();  
            *txtLayer*.Items.**Clear**();  
            SqlConnection sConn = **Connect**();  
            **try**  
            {  
                //先查出所有的表名  
                string sqlStr = "select name from syscolumns where id=(object\_id('" + *txtTableName*.SelectedValue + "'))";  
                SqlCommand sCmd = **new**SqlCommand(sqlStr, sConn);  
                SqlDataReader  sdr = sCmd.**ExecuteReader**();  
                **while** (sdr.**Read**())  
                {  
                    *txtSysid*.Items.**Add**(sdr.**GetString**(0));  
                    *txtRealname*.Items.**Add**(sdr.**GetString**(0));  
                    *txtTopid*.Items.**Add**(sdr.**GetString**(0));  
                    *txtLayer*.Items.**Add**(sdr.**GetString**(0));  
                }  
                sdr.**Close**();  
            }  
            **catch** (Exception ex)  
            {  
                MessageBox.**Show**(ex.Message);  
                *txtSysid*.IsEnabled = **false**;  
                *txtRealname*.IsEnabled = **false**;  
                *txtTopid*.IsEnabled = **false**;  
                *txtLayer*.IsEnabled = **false**;  
            }  
            **finally**  
            {  
                 sConn.**Close**();  
            }  
        }   
          
        void**btnExport\_Click**(object sender, RoutedEventArgs e)  
        {  
            **if**(!**CheckComboBox**(*txtTableName*,"数据表名"))  
            {  
                return;  
            }  
            **if**(!**CheckComboBox**(*txtSysid*,"sysid列"))  
            {  
                return;  
            }  
            **if**(!**CheckComboBox**(*txtTopid*,"topid列"))  
            {  
                return;  
            }  
            SqlConnection sConn = **Connect**();  
            **try**  
            {   
                //先查出总数量  
                SqlCommand sCmd = **new**SqlCommand("select count(\*) from [" + *txtTableName*.SelectedValue + "]", sConn);  
                SqlDataReader  sdr = sCmd.**ExecuteReader**();  
                sdr.**Read**();  
                **int** allRow = sdr.**GetInt32**(0);  
                sdr.**Close**();  
                  
                //再查下所有的数据  
                StringBuilder strBld = **new**StringBuilder();  
                strBld.**Append**("select [" + *txtSysid*.SelectedValue + "], [" );  
                **if**(*txtRealname*.SelectedIndex != -1)  
                {  
                    strBld.**Append**(*txtRealname*.SelectedValue + "], [");  
                }  
                strBld .**Append**(*txtTopid*.SelectedValue);  
                **if**(*txtLayer*.SelectedIndex != -1)  
                {  
                    strBld.**Append**("], [" + *txtLayer*.SelectedValue);  
                }  
                strBld.**Append**("] from [" + *txtTableName*.SelectedValue + "]");  
                sCmd.CommandText = strBld.**ToString**();  
                sdr = sCmd.**ExecuteReader**();  
                  
                *prograss*.Visibility = **Visibility**.*Visible*;  
                *labelMessage*.Text = "开始导出数据到CSV文件......";  
                **DoEvents**();  
      
                StringBuilder allLines = **new**StringBuilder("sysid,realname,topid,layer");  
                **int** row = 0;  
                **int** step = allRow >100 ? allRow / 100 : 1;  
                **while** (sdr.**Read**())  
                {     
                    allLines.**Append**("\r\n");  
                    allLines.**Append**(sdr[0].**ToString**().**Replace**(',',' '));  
                    allLines.**Append**(",");  
                    **if**(*txtRealname*.SelectedIndex != -1)  
                    {  
                        allLines.**Append**(sdr[1].**ToString**().**Replace**(',',' '));  
                        allLines.**Append**(",");  
                        allLines.**Append**(sdr[2].**ToString**().**Replace**(',',' '));  
                    }  
                    **else**  
                    {  
                        allLines.**Append**(",");  
                        allLines.**Append**(sdr[1].**ToString**().**Replace**(',',' '));  
                    }  
                    allLines.**Append**(",");  
                    **if**(*txtLayer*.SelectedIndex != -1)  
                    {  
                         
                        allLines.**Append**(sdr[3].**ToString**().**Replace**(',',' '));  
                    }  
                    row++;  
                    **if** (row % step == 0)  
                    {  
                        *prograss*.Value = (**int**)(100.0 \* row / allRow);  
                        *labelMessage*.Text = "正在导出第" + row + "个节点（总共" + allRow + "个节点）";  
                        **DoEvents**();  
                    }  
                }  
                sdr.**Close**();  
                sConn.**Close**();  
                  
                //保存到csv文件中  
                string csvName = "user" + **DateTime**.Now.**ToString**("yyyyMMddhhmmss") + ".csv";  
                **using**(StreamWriter mysw = **new**StreamWriter(csvName, **false**, Encoding.UTF8))  
                {  
                    mysw.**Write**(allLines);  
                }  
                MessageBox.**Show**("数据已经保存到" + System.Windows.Forms.Application.StartupPath + "目录下！");  
                **this**.**Close**();  
            }  
            **catch** (Exception ex)  
            {  
                MessageBox.**Show**(ex.Message);  
            }  
            **finally**  
            {  
                 sConn.**Close**();  
            }  
        }  
          
        **private**void**DoEvents**()  
        {  
            DispatcherFrame frame = **new**DispatcherFrame();  
            Dispatcher.CurrentDispatcher.**BeginInvoke**(**DispatcherPriority**.*Background*,  
                **new** DispatcherOperationCallback(delegate(object f)  
                {  
                    (f **as**DispatcherFrame).Continue = **false**;  
  
                    return**null**;  
                }  
            ), frame);  
            Dispatcher.**PushFrame**(frame);  
        }  
          
    }  
}

# 打印、导出数据导出数据模块

## 打印和导出图片

//导出图片  
**private**void**btnExportImg\_Click**(object sender, RoutedEventArgs e)

{  
    SaveFileDialog openfileDlg = **new**SaveFileDialog();  
    openfileDlg.InitialDirectory = System.Windows.Forms.Application.StartupPath;  
    openfileDlg.Title = "选择将会员树导出为文件的位置";  
    openfileDlg.Filter = "png格式|\*.png";  
    **if** (openfileDlg.**ShowDialog**() == **true**)  
    {  
        FileStream fs = **new**FileStream(openfileDlg.FileName, **FileMode**.*Create*);  
        **int** width = (**int**)*myTreeView*.*memberTreeView*.ActualWidth;  
        **int** height = (**int**)*myTreeView*.*memberTreeView*.ActualHeight;  
     RenderTargetBitmap bmp = **new**RenderTargetBitmap(width, height, 96, 96, PixelFormats.Default);  
        bmp.**Render**(*myTreeView*.*memberTreeView*);  
        BitmapEncoder encoder = **new**PngBitmapEncoder();  
        encoder.Frames.**Add**(BitmapFrame.**Create**(bmp));  
        encoder.**Save**(fs);  
        fs.**Close**();  
        fs.**Dispose**();  
    }  
}  
  
//打印  
**private**void**btnPrint\_Click**(object sender, RoutedEventArgs e)  
{  
    PrintDialog printDlg = **new**PrintDialog();  
    printDlg.UserPageRangeEnabled = **true**;  
  
    **if** (printDlg.**ShowDialog**() == **true**)  
    {  
         printDlg.**PrintVisual**(*myTreeView*.*memberTreeView*, "打印当前会员树视图");  
    }  
}

## 导出数据到Excel或csv文件

**namespace** MemberTree  
{  
    **public**classExport  
    {  
        **private**static**int***row*;  
        **private**static**int***allRow*;  
        **private**static**int***step*;          
  
        //public static void ExportAll2CSV(string outputfile)  
        //{  
        //    MainWindow.notify.SetProcessBarVisible(true);  
        //    MainWindow.notify.SetStatusMessage("开始导出文件......");  
  
        //    StringBuilder allLines = new StringBuilder("会员ID,会员姓名,所在层级,下线人数");  
        //    row = 2;  
        //    foreach (var item in MyTreeNode.AllNodes.Values)  
        //    {  
        //        foreach (MyTreeNode node in item.Values)  
        //        {  
        //            allLines.Append("\n");  
        //            allLines.Append(node.SysId);  
        //            allLines.Append(",");  
        //            allLines.Append(node.RealName);  
        //            allLines.Append(",");  
        //            allLines.Append(node.Level);  
        //            allLines.Append(",");  
        //            allLines.Append(node.DescendantCount);  
        //            row++;  
        //            if (row % 1000 == 0)  
        //            {  
        //                MainWindow.notify.SetProcessBarValue((int)(100.0 \* row / MyTreeNode.allNodesCount));  
        //                MainWindow.notify.SetStatusMessage("正在导出第" + row + "个节点（总共" + MyTreeNode.allNodesCount + "个节点）");  
        //            }  
        //        }  
        //    }  
  
        //    MainWindow.notify.SetStatusMessage("正在将数据写入CSV文件。。。");  
  
        //    StreamWriter mysw = new StreamWriter(outputfile, false, Encoding.UTF8);  
        //    mysw.Write(allLines);  
        //    mysw.Close();  
  
        //    MainWindow.notify.SetStatusMessage("会员树导出到CSV成功！");  
        //    MainWindow.notify.SetProcessBarVisible(false);  
        //}  
  
        //public static void ExportAll2Excel(string outputfile)  
        //{  
        //    FileInfo newFile = new FileInfo(outputfile);  
        //    if (newFile.Exists)  
        //    {  
        //        newFile.Delete();  
        //        newFile = new FileInfo(outputfile);  
        //    }  
  
        //    MainWindow.notify.SetProcessBarVisible(true);  
        //    MainWindow.notify.SetStatusMessage("开始导出文件......");  
  
        //    ExcelPackage pck = new ExcelPackage(newFile);  
        //    var ws = pck.Workbook.Worksheets.Add("会员名单列表");  
  
        //    //Headers  
        //    ws.Cells["A1"].Value = "会员ID";  
        //    ws.Cells["B1"].Value = "会员姓名";  
        //    ws.Cells["C1"].Value = "所在层级";  
        //    ws.Cells["D1"].Value = "下线人数";  
        //    ws.Cells["A1:D1"].Style.Font.Bold = true;  
  
        //    ws.View.FreezePanes(2, 1);  
        //    ws.Select("A2");  
  
        //    row = 2;  
        //    foreach (var item in MyTreeNode.AllNodes.Values)  
        //    {  
        //        foreach (MyTreeNode node in item.Values)  
        //        {  
        //            ws.Cells[row, 1].Value = node.SysId;  
        //            ws.Cells[row, 2].Value = node.RealName;  
        //            ws.Cells[row, 3].Value = node.Level;  
        //            ws.Cells[row, 4].Value = node.DescendantCount;  
        //            row++;  
        //            if (row % 100 == 0)  
        //            {  
        //                MainWindow.notify.SetProcessBarValue((int)(100.0 \* row / MyTreeNode.allNodesCount));  
        //                MainWindow.notify.SetStatusMessage("正在导出第" + row + "个节点（总共" + MyTreeNode.allNodesCount + "个节点）");  
        //            }  
        //        }  
        //    }  
  
        //    //List<MyTreeNode> roots = new List<MyTreeNode>();  
        //    //if (MyTreeNode.AllNodes.ContainsKey(1))  
        //    //{  
        //    //    roots.AddRange(MyTreeNode.AllNodes[1].Values);  
        //    //}  
        //    //row = 1;  
        //    //for (int i = 0; i < roots.Count; i++)  
        //    //{  
        //    //    AddRow(ws, roots[i], 1,"", notify);  
        //    //}  
  
        //    MainWindow.notify.SetStatusMessage("正在将数据写入Excel文件。。。");  
  
        //    pck.Save();  
  
        //    MainWindow.notify.SetStatusMessage("会员树导出到Excel成功！");  
        //    MainWindow.notify.SetProcessBarVisible(false);  
        //}  
  
        //private static void AddRow(ExcelWorksheet ws, MyTreeNode node, int level, string parentAddrStr)  
        //{  
        //    ws.Cells[row, level].Value = node.RealName;  
        //    //ws.Cells[row, level + 1].Value = "Id:" + node.SysId;  
        //    ws.Cells[row, level + 1].Value = "级别:" + node.Level;  
        //    ws.Cells[row, level + 2].Value = "下线:" + node.DescendantCount;  
  
  
        //    //ws.Cells[row, level].Style.Font.Bold = true;  
        //    //ws.Cells[row, level, row, level + 3].Style.Fill.PatternType = ExcelFillStyle.Solid;  
        //    //ws.Cells[row, level, row, level + 3].Style.Fill.BackgroundColor.SetColor(Color.Red);  
  
        //    //if (parentAddrStr != "")  
        //    //{  
        //    //    ws.Cells[row, level].Hyperlink = new ExcelHyperLink(parentAddrStr, UriKind.Relative);  
        //    //}  
  
        //    row++;  
        //    if (row % 10 == 0)  
        //    {  
        //        MainWindow.notify.SetProcessBarValue((int)(100.0 \* row / MyTreeNode.allNodesCount));  
        //        MainWindow.notify.SetStatusMessage("正在导出第" + row + "个节点（总共" + MyTreeNode.allNodesCount + "个节点）");  
        //    }  
  
        //    for (int i = 0; i < node.ChildrenNodes.Count; i++)  
        //    {  
        //        AddRow(ws, node.ChildrenNodes[i], level + 1, ws.Cells[row, level].Address);  
  
        //        //ws.Cells[row, level, row + children[i].DescendantCount, level].Value = "┃";  
        //        //if (i == children.Count - 1)  
        //        //{  
        //        //    ws.Cells[children[i].DescendantCount + 1, level].Value = "┗━━━";  
        //        //}  
        //        //else  
        //        //{  
        //        //    ws.Cells[children[i].DescendantCount + 1, level].Value = "┣━━━";  
        //        //}  
        //    }  
        //}  
          
        **public**staticvoid**ExportNodes**(List<MyTreeNode> nodes)  
        {  
            SaveFileDialog openfileDlg = **new**SaveFileDialog();  
            openfileDlg.InitialDirectory = System.Windows.Forms.Application.StartupPath;  
            openfileDlg.Title = "选择将会员树导出为文件的位置";  
            openfileDlg.Filter = "CSV文件|\*.csv|Excel2007文件|\*.xlsx";  
            **if** (openfileDlg.**ShowDialog**() == **true**)  
            {  
                **if** (openfileDlg.FileName.**EndsWith**(".csv"))  
                {  
                    Export.**Export2CSV**(nodes, openfileDlg.FileName);  
                }  
                **else**  
                {  
                    Export.**Export2Excel**(nodes, openfileDlg.FileName);  
                }  
            }  
        }  
  
        **private**staticList<string>*ringNodeIds* = **new**List<string>();  
        //判断闭环是否关闭  
        **private**static**boolisRingClose**(string id)  
        {  
            **if** (MyTreeNode.RingNodes.**ContainsKey**(id))  
            {  
                **if** (*ringNodeIds*.**Contains**(id))  
                {  
                    return**true**;  
                }  
                **else**  
                {  
                    *ringNodeIds*.**Add**(id);  
                }  
            }  
            return**false**;  
        }  
  
        **public**staticvoid**Export2CSV**(List<MyTreeNode> nodes, string outputfile)  
        {  
            *ringNodeIds*.**Clear**();  
            MainWindow.*notify*.**SetProcessBarVisible**(**true**);  
            MainWindow.*notify*.**SetStatusMessage**("开始导出文件......");  
  
            StringBuilder allLines = **new**StringBuilder("会员ID,会员姓名,所在层级,下线人数,下线层数,顶点ID");  
            *row* = 2;  
            *allRow* = 0;  
            **for** (**int** i = 0; i < nodes.Count; i++)   
            {  
                *allRow* += nodes[i].DescendantCount;  
            }  
            *step* = *allRow*>100 ? *allRow* / 100 : 1;  
            **for** (**int** i = 0; i < nodes.Count; i++)   
            {  
                **Export2CSVImp**(allLines, nodes[i], nodes[i].SysId);  
            }  
  
            MainWindow.*notify*.**SetStatusMessage**("正在将数据写入CSV文件。。。");  
  
            StreamWriter mysw = **new**StreamWriter(outputfile, **false**, Encoding.Default);  
            mysw.**Write**(allLines);  
            mysw.**Close**();  
  
            MainWindow.*notify*.**SetStatusMessage**("");  
            MainWindow.*notify*.**SetProcessBarVisible**(**false**);  
        }  
  
        **private**staticvoid**Export2CSVImp**(StringBuilder allLines, MyTreeNode node, string topMostId)  
        {  
            **if** (**isRingClose**(node.SysId))  
            {  
                return;  
            }  
  
            allLines.**Append**("\r\n");  
            allLines.**Append**(node.SysId);  
            allLines.**Append**(",");  
            allLines.**Append**(node.RealName);  
            allLines.**Append**(",");  
            allLines.**Append**(node.Level);  
            allLines.**Append**(",");  
            allLines.**Append**(node.DescendantCount);  
            allLines.**Append**(",");  
            allLines.**Append**(node.DescendantLevels);  
            allLines.**Append**(",");  
            allLines.**Append**(topMostId);  
            *row*++;  
            **if** (*row* % *step* == 0)  
            {  
                MainWindow.*notify*.**SetProcessBarValue**((**int**)(100.0 \* *row* / *allRow*));  
                MainWindow.*notify*.**SetStatusMessage**("正在导出第" + *row* + "个节点（总共" + *allRow* + "个节点）");  
            }  
  
            **foreach** (MyTreeNode subNode **in** node.ChildrenNodes)  
            {  
                **Export2CSVImp**(allLines, subNode, topMostId);  
            }  
        }  
  
        **public**staticvoid**Export2Excel**(List<MyTreeNode> nodes, string outputfile)  
        {  
            *ringNodeIds*.**Clear**();  
            FileInfo newFile = **new**FileInfo(outputfile);  
            **if** (newFile.Exists)  
            {  
                newFile.**Delete**();  
                newFile = **new**FileInfo(outputfile);  
            }  
  
            MainWindow.*notify*.**SetProcessBarVisible**(**true**);  
            MainWindow.*notify*.**SetStatusMessage**("开始导出文件......");  
  
            ExcelPackage pck = **new**ExcelPackage(newFile);  
            ExcelWorksheet ws = pck.Workbook.Worksheets.**Add**("会员名单列表");  
  
            //Headers  
            ws.Cells["A1"].Value = "会员ID";  
            ws.Cells["B1"].Value = "会员姓名";  
            ws.Cells["C1"].Value = "所在层级";  
            ws.Cells["D1"].Value = "下线人数";  
            ws.Cells["E1"].Value = "下线层数";  
            ws.Cells["F1"].Value = "顶点ID";  
            ws.Cells["A1:F1"].Style.Font.Bold = **true**;  
  
            ws.View.**FreezePanes**(2, 1);  
            ws.**Select**("A2");  
  
            *row* = 2;  
            *allRow* = 0;  
            **for** (**int** i = 0; i < nodes.Count; i++)   
            {  
                *allRow* += nodes[i].DescendantCount;  
            }  
            *step* = *allRow*>100 ? *allRow* / 100 : 1;  
            **for** (**int** i = 0; i < nodes.Count; i++)   
            {  
                **Export2ExcelImp**(ws, nodes[i], nodes[i].SysId);  
            }  
  
            MainWindow.*notify*.**SetStatusMessage**("正在将数据写入Excel文件。。。");  
  
            pck.**Save**();  
  
            MainWindow.*notify*.**SetStatusMessage**("");  
            MainWindow.*notify*.**SetProcessBarVisible**(**false**);  
        }  
  
        **private**staticvoid**Export2ExcelImp**(ExcelWorksheet ws, MyTreeNode node, string topMostId)  
        {  
            **if** (**isRingClose**(node.SysId))  
            {  
                return;  
            }  
  
            ws.Cells[*row*, 1].Value = node.SysId;  
            ws.Cells[*row*, 2].Value = node.RealName;  
            ws.Cells[*row*, 3].Value = node.Level;  
            ws.Cells[*row*, 4].Value = node.DescendantCount;  
            ws.Cells[*row*, 5].Value = node.DescendantLevels;  
            ws.Cells[*row*, 6].Value = topMostId;  
  
            **foreach** (MyTreeNode subNode **in** node.ChildrenNodes)  
            {  
                *row*++;  
                **if** (*row* % *step* == 0)  
                {  
                    MainWindow.*notify*.**SetProcessBarValue**((**int**)(100.0 \* *row* / *allRow*));  
                    MainWindow.*notify*.**SetStatusMessage**("正在导出第" + *row* + "个节点（总共" + *allRow* + "个节点）");  
                }  
  
                **Export2ExcelImp**(ws, subNode, topMostId);  
            }  
        }  
    }  
}

# 查找、数据处理等模块

## 查找多个记录界面布局代码

<Window x:Class="MemberTree.SearchMultiInput"  
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"  
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"  
    Title="查找导出多个"Height="500"Width="250"WindowStartupLocation="CenterScreen">  
    <Grid>  
        <Grid.RowDefinitions>  
            <RowDefinition Height="Auto"/>  
            <RowDefinition Height="Auto"/>  
            <RowDefinition />  
            <RowDefinition Height="Auto"/>  
        </Grid.RowDefinitions>  
        <GroupBox Header="查找结果要保存的位置">  
            <Grid>  
                <Grid.ColumnDefinitions>  
                    <ColumnDefinition />  
                    <ColumnDefinition Width="30"/>  
                </Grid.ColumnDefinitions>  
                <TextBox Name="txtPath"IsReadOnly="True"TextWrapping="Wrap" />  
                <Button Content="浏览"Grid.Column="1"Click="Browse\_Click"/>  
            </Grid>  
        </GroupBox>  
        <ComboBox Name="comboSearchType"Grid.Row="1">  
            <ComboBoxItem IsSelected="True">根据ID查找</ComboBoxItem>  
            <ComboBoxItem>根据姓名查找</ComboBoxItem>  
            <ComboBoxItem>根据级别查找</ComboBoxItem>  
            <ComboBoxItem>根据下线人数查找</ComboBoxItem>  
            <ComboBoxItem>根据下线层数查找</ComboBoxItem>  
        </ComboBox>  
        <GroupBox Name="groupUsr"Header="输入多个查询关键字，每个关键字一行"Grid.Row="2">  
            <TextBox Name="txtUsr"TextWrapping="Wrap"AcceptsReturn="True"VerticalScrollBarVisibility="Visible"/>  
        </GroupBox>  
        <ScrollViewer Name="scrollUsr"Grid.Row="2"Visibility="Hidden">  
            <StackPanel Name="listUsr"/>  
        </ScrollViewer>  
        <Button Name="btnOK"Content="开始查找"Grid.Row="3"IsEnabled="False"Click="Button\_Click"/>          
    </Grid>  
</Window>

## 查找多个记录业务逻辑处理代码

**namespace** MemberTree  
{  
    ///<**summary**>  
    /// Interaction logic for SearchMultiInput.xaml  
    ///<**/summary**>  
    **public**partialclassSearchMultiInput : Window  
    {  
        **public**SearchMultiInput()  
        {  
            **InitializeComponent**();  
            *btnOK*.Content = "开始查找";  
        }  
          
        **private**void**Browse\_Click**(object sender, RoutedEventArgs e)  
        {  
            SaveFileDialog openfileDlg = **new**SaveFileDialog();  
            openfileDlg.InitialDirectory = System.Windows.Forms.Application.StartupPath;  
            openfileDlg.Title = "选择导出为文件的位置";  
            openfileDlg.Filter = "CSV文件|\*.csv";  
            **if** (openfileDlg.**ShowDialog**() == **true**)  
            {  
                *txtPath*.Text = openfileDlg.FileName;  
                *btnOK*.IsEnabled = **true**;  
            }  
        }  
          
        **private**void**Button\_Click**(object sender, RoutedEventArgs e)  
        {  
            **if**(*btnOK*.Content == "开始查找")  
            {  
                *btnOK*.IsEnabled = **false**;  
                *groupUsr*.Visibility = **Visibility**.*Hidden*;  
                *scrollUsr*.Visibility = **Visibility**.*Visible*;  
                  
                string[] usrIds =  *txtUsr*.Text.**Split**(**new**String[] {"\r\n"}, **StringSplitOptions**.*RemoveEmptyEntries*);  
                List<MyTreeNode> findResultNodes = **new**List<MyTreeNode>();  
                **for** (**int** i = 0; i < usrIds.Length; i++)   
                {  
                    CheckBox newCheck = **new**CheckBox();  
                    newCheck.Content = usrIds[i];  
                    newCheck.IsEnabled = **false**;  
                    *listUsr*.Children.**Add**(newCheck);  
                      
                    //查找  
                    List<MyTreeNode> findNodes = **SearchNode**(usrIds[i]);  
                    **if**(findNodes.Count >0)  
                    {  
                        findResultNodes.**AddRange**(findNodes);  
                        newCheck.IsChecked = **true**;  
                        *scrollUsr*.**ScrollToEnd**();  
                    }  
                      
                       **DoEvents**();  
                }  
                  
                //导出  
                **Export2CSV**(findResultNodes, *txtPath*.Text);  
                  
                **if**(findResultNodes.Count >0)  
                {  
                    MessageBox.**Show**("查找并导出结果完成！\n一共查找到了"+findResultNodes.Count+"个结果！");  
                }  
                **else**  
                {  
                    MessageBox.**Show**("没有查找到任何结果！");  
                }  
                *btnOK*.IsEnabled = **true**;  
                *btnOK*.Content = "返回编辑模式";  
            }  
            **else**  
            {  
                *groupUsr*.Visibility = **Visibility**.*Visible*;  
                *scrollUsr*.Visibility = **Visibility**.*Hidden*;  
                *listUsr*.Children.**Clear**();  
                *btnOK*.Content = "开始查找";  
            }  
        }  
          
        **private**void**DoEvents**()  
        {  
            DispatcherFrame frame = **new**DispatcherFrame();  
            Dispatcher.CurrentDispatcher.**BeginInvoke**(**DispatcherPriority**.*Background*,  
                **new** DispatcherOperationCallback(delegate(object f)  
                {  
                    (f **as**DispatcherFrame).Continue = **false**;  
  
                    return**null**;  
                }  
            ), frame);  
            Dispatcher.**PushFrame**(frame);  
        }  
          
        **private**List<MyTreeNode>**SearchNode**(string searchTxt)  
        {  
            List<MyTreeNode> findNodes = **new**List<MyTreeNode>();  
            string searchType = (*comboSearchType*.SelectedItem **as**ComboBoxItem).Content.**ToString**();  
            **if** (searchType == "根据ID查找")  
            {  
                findNodes = MyTreeNode.**FindNodeById**(searchTxt);  
            }  
            **elseif** (searchType == "根据姓名查找")  
            {  
                findNodes = MyTreeNode.**FindNodeByName**(searchTxt);  
            }  
            **elseif** (searchType == "根据级别查找")  
            {  
                findNodes = MyTreeNode.**FindNodeByLevel**(searchTxt);  
            }  
            **elseif** (searchType == "根据下线人数查找")  
            {  
                **int** txt2num = 0;  
                **if** (**int**.**TryParse**(searchTxt, **out** txt2num))  
                {  
                    findNodes = MyTreeNode.**FindNodeByDescendantCount**(txt2num);  
                }  
            }  
            **elseif** (searchType == "根据下线层数查找")  
            {  
                **int** txt2num = 0;  
                **if** (**int**.**TryParse**(searchTxt, **out** txt2num))  
                {  
                    findNodes = MyTreeNode.**FindNodeByDescendantLevels**(txt2num);  
                }  
            }  
            return findNodes;  
        }  
          
        **private**void**Export2CSV**(List<MyTreeNode> nodes ,string outputfile)  
        {  
            StringBuilder allLines = **new**StringBuilder("会员ID,会员姓名,所在层级,父节点ID,下线人数,下线层数,所在行数");  
            **for** (**int** i = 0; i < nodes.Count; i++)   
            {  
                MyTreeNode node = nodes[i];  
                allLines.**Append**("\r\n");  
                allLines.**Append**(node.SysId);  
                allLines.**Append**(",");  
                allLines.**Append**(node.RealName);  
                allLines.**Append**(",");  
                 allLines.**Append**(node.Level);  
                allLines.**Append**(",");  
                allLines.**Append**(node.TopId);  
                allLines.**Append**(",");  
                allLines.**Append**(node.DescendantCount);  
                allLines.**Append**(",");  
                allLines.**Append**(node.DescendantLevels);  
                allLines.**Append**(",");  
                allLines.**Append**(node.LineCount);  
            }  
  
            StreamWriter mysw = **new**StreamWriter(outputfile, **false**, Encoding.Default);  
            mysw.**Write**(allLines);  
            mysw.**Close**();  
        }  
    }  
}

## 数据处理辅助类代码

**namespace** MemberTree  
{  
    ///<**summary**>  
    ///  判断文件的编码，全角半角的相互转换    
    ///<**/summary**>    
    **public**classTextUtilTool  
    {  
        //判断文件编码  
        **public**staticEncoding**GetFileEncodeType**(string filename)  
        {  
            FileStream fs = **new**FileStream(filename, **FileMode**.*Open*, **FileAccess**.*Read*);  
            BinaryReader br = **new**BinaryReader(fs);  
            **Byte**[] buffer = br.**ReadBytes**(2);  
            fs.**Close**();  
            **if** (buffer[0] >= 0xEF)  
            {  
                **if** (buffer[0] == 0xEF&& buffer[1] == 0xBB)  
                {  
                    returnEncoding.UTF8;  
                }  
                **elseif** (buffer[0] == 0xFE&& buffer[1] == 0xFF)  
                {  
                    returnEncoding.BigEndianUnicode;  
                }  
                **elseif** (buffer[0] == 0xFF&& buffer[1] == 0xFE)  
                {  
                    returnEncoding.Unicode;  
                }  
                **else**  
                {  
                    returnEncoding.Default;  
                }  
            }  
            **else**  
            {  
                returnEncoding.Default;  
            }  
        }  
  
        ///<**summary**>  
        /// 半角转成全角    
        /// 半角空格32,全角空格12288    
        /// 其他字符半角33~126,其他字符全角65281~65374,相差65248    
        ///<**/summary**>    
        ///<**paramname="input"**><**/param**>    
        ///<**returns**><**/returns**>    
        **public**staticstring**DBCToSBC**(string input)  
        {  
            **char**[] cc = input.**ToCharArray**();  
            **for** (**int** i = 0; i < cc.Length; i++)  
            {  
                **if** (cc[i] == 32)  
                {  
                    // 表示空格    
                    cc[i] = (**char**)12288;  
                    continue;  
                }  
                **if** (cc[i] <127&& cc[i] >32)  
                {  
                    cc[i] = (**char**)(cc[i] + 65248);  
                }  
            }  
            return**new**string(cc);  
        }  
  
        ///<**summary**>  
        /// 全角转半角    
        /// 半角空格32,全角空格12288    
        /// 其他字符半角33~126,其他字符全角65281~65374,相差65248    
        ///<**/summary**>    
        ///<**paramname="input"**><**/param**>    
        ///<**returns**><**/returns**>    
        **public**staticstring**SBCToDBC**(string input)  
        {  
            **char**[] cc = input.**ToCharArray**();  
            **for** (**int** i = 0; i < cc.Length; i++)  
            {  
                **if** (cc[i] == 12288)  
                {  
                    // 表示空格    
                    cc[i] = (**char**)32;  
                    continue;  
                }  
                **if** (cc[i] >65280&& cc[i] <65375)  
                {  
                    cc[i] = (**char**)(cc[i] - 65248);  
                }  
  
            }  
            return**new**string(cc);  
        }  
    }  
}