**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 | 2017.12.13 |

# 目录

目录 2

1. 程序总体说明 4

1.1. 程序技术架构 4

1.2. 关于WPF 4

1.3. 动态绘制树形结构 4

1.4. 代码行数统计 7

2. 程序代码结构说明 16

2.1. 代码项目模块说明 16

2.2. 程序主要代码功能模块 16

2.3. 文件后缀名说明 16

3. 服务端数据管理模块 18

3.1. 主程序界面布局代码 18

3.2. 数据管理界面代码 18

3.3. 数据集概要信息表示代码 22

3.4. 数据集列表代码 25

4. 服务端用户管理模块 28

4.1. 用户管理主界面代码 28

4.2. 管理用户信息代码 28

4.3. 管理用户权限代码 33

5. 服务端直连数据库模块 37

5.1. 连接mysql/sqlser界面代码 37

5.2. 连接sqlser业务代码 39

5.3. 连接mysql业务代码 43

6. 服务端导入计算树结构模块 49

6.1. 服务端树节点定义代码 49

6.2. 服务端计算树结构代码 50

6.3. 树结构计算完写入Sqlite代码 58

6.4. 树结构计算完写入Mysql代码 62

6.5. 检查csv数据错误代码 67

7. 客户端读取树结构模块 73

7.1. 客户端树节点定义代码 73

7.2. 客户端管理树结构代码 73

7.3. 客户端从Sqlite读取树结构代码 79

7.4. 客户端从Mysql读取树结构代码 81

8. 客户端动态绘制树形结构模块 84

8.1. 树容器结构代码 84

8.2. 树节点结构代码 92

8.3. 树控件绘制算法代码 97

8.4. 树控件操作处理逻辑代码 110

8.5. 线状树结构风格代码 113

8.6. 组织结构树视图风格代码 116

8.7. 自定义树控件界面布局代码 121

8.8. 自定义树控件后台处理逻辑代码 123

9. 客户端数据导出模块 131

9.1. 导出保存为Csv文件 131

9.2. 导出保存为Png图片 135

9.3. 导出保存为PDF文件 136

10. 客户端查询分页模块 141

10.1. 模糊查询条件处理代码 141

10.2. 分页处理代码 144

11. 加密解密模块 148

11.1. DES加密解密代码 148

11.2. RSA加密解密代码 150

12. 授权注册模块 156

12.1. 注册管理程序界面代码 156

12.2. 注册管理程序生成密钥代码 157

12.3. 注册界面代码 159

12.4. 注册帮助类代码 161

# 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；由于本地数据库存储采用的是sqite，所以对于大规模的海量数据，最好采用SSD硬盘，能显著地加快数据读写速度。

## 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。

## 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；

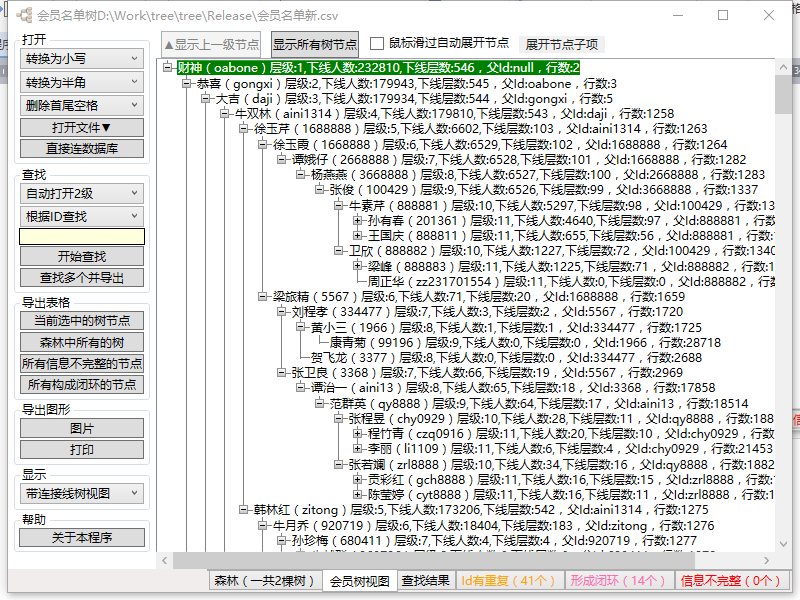
所以也没必要全部同时显示出来所有数据，我们只需要根据屏幕大小，动态加载当前呈现在屏幕中的节点即可，用户点开下一级节点，就临时再从数据中去找到相关的节点数据，临时呈现在屏幕上，用户关闭该节点，就从屏幕中销毁该节点，这样既达到了想要的效果，又不需要同时加载所有数据，对计算机内存等要求不必那么高。

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：

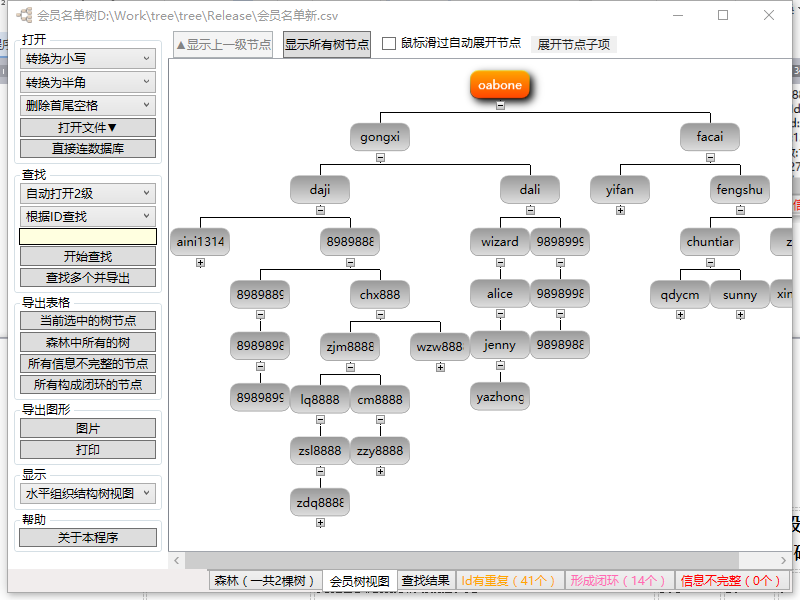
* **最简约风格：**

****

* **带连接线的风格**

****

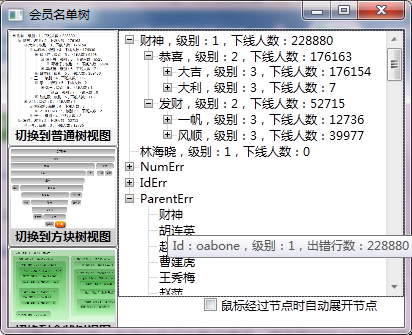
* **水平组织结构树状视图**



* **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** | **文件名称** | **代码行数** | **空行数** | **设计器代码行数** | **注释行数** |
| AutoUpdate.csproj | App.xaml.cs | 15 | 1 | 0 | 3 |
| AutoUpdate.xaml.cs | 155 | 11 | 0 | 22 |
| AutoUpdateVer.cs | 53 | 4 | 0 | 12 |
| AssemblyInfo.cs | 31 | 5 | 0 | 11 |
| **合计** | **254** | **21** | **0** | **48** |
| MemberTreeAdmin.csproj | App.xaml.cs | 15 | 1 | 0 | 3 |
| ConnMysql.cs | 203 | 16 | 0 | 17 |
| ConnSqlserver.cs | 199 | 14 | 0 | 18 |
| IConnDB.cs | 26 | 1 | 0 | 11 |
| WindowConnDB.xaml.cs | 300 | 23 | 0 | 16 |
| DBUtil.cs | 129 | 7 | 0 | 17 |
| IMyTreeDBA.cs | 36 | 7 | 0 | 11 |
| MyTreeDBASqlite.cs | 177 | 23 | 0 | 23 |
| MyTreeDBAMysql.cs | 225 | 19 | 0 | 27 |
| WindowCsvErrCheck.xaml.cs | 242 | 22 | 0 | 26 |
| WindowColsCheck.xaml.cs | 64 | 4 | 0 | 11 |
| TextUtil.cs | 125 | 9 | 0 | 19 |
| MyTrees.cs | 347 | 44 | 0 | 29 |
| MyTreeNode.cs | 85 | 8 | 0 | 8 |
| MainWindow.xaml.cs | 79 | 10 | 0 | 3 |
| AdminDataSet.xaml.cs | 111 | 8 | 0 | 18 |
| AdminUser.xaml.cs | 87 | 6 | 0 | 13 |
| BtnDataset.xaml.cs | 57 | 3 | 0 | 11 |
| BtnUserInfo.xaml.cs | 63 | 3 | 0 | 11 |
| ConnDBView.xaml.cs | 158 | 12 | 0 | 16 |
| UserInfoSet.xaml.cs | 218 | 15 | 0 | 20 |
| UserPrivilege1.xaml.cs | 164 | 14 | 0 | 16 |
| UserPrivilege2.xaml.cs | 164 | 15 | 0 | 16 |
| WindowAdmin.xaml.cs | 41 | 6 | 0 | 4 |
| AssemblyInfo.cs | 53 | 8 | 0 | 26 |
| Resources.Designer.cs | 63 | 0 | 63 | 0 |
| Settings.Designer.cs | 26 | 0 | 26 | 0 |
| **合计** | **3457** | **298** | **89** | **390** |
| MemberTreeCommon.csproj | DatasetInfo.cs | 35 | 2 | 0 | 11 |
| DBSession.cs | 87 | 8 | 0 | 11 |
| IMyTreeDB.cs | 30 | 1 | 0 | 11 |
| MyTreeDBMysql.cs | 225 | 19 | 0 | 19 |
| MyTreeDBSqlite.cs | 155 | 15 | 0 | 13 |
| UserAdmin.cs | 264 | 17 | 0 | 26 |
| UserInfo.cs | 87 | 9 | 0 | 11 |
| DatasetBtn.xaml.cs | 71 | 7 | 0 | 11 |
| DatasetInfoView.xaml.cs | 133 | 9 | 0 | 11 |
| DatasetListView.xaml.cs | 129 | 7 | 0 | 11 |
| MyStatusBar.xaml.cs | 141 | 9 | 0 | 19 |
| ProgressView.xaml.cs | 116 | 6 | 0 | 13 |
| WelcomeView.xaml.cs | 49 | 3 | 0 | 3 |
| WindowAbout.xaml.cs | 50 | 3 | 0 | 11 |
| LoginView.xaml.cs | 51 | 5 | 0 | 12 |
| WindowVerLog.xaml.cs | 80 | 5 | 0 | 12 |
| AssemblyInfo.cs | 31 | 5 | 0 | 11 |
| RegConfig.cs | 46 | 3 | 0 | 11 |
| SoftReg.cs | 233 | 12 | 0 | 16 |
| SoftRegView.xaml.cs | 94 | 4 | 0 | 11 |
| INotify.cs | 19 | 1 | 0 | 8 |
| InvokeDelegate.cs | 17 | 1 | 0 | 8 |
| MemData.cs | 40 | 2 | 0 | 11 |
| SysInfo.cs | 40 | 2 | 0 | 11 |
| TimingUtil.cs | 56 | 3 | 0 | 11 |
| **合计** | **2279** | **158** | **0** | **303** |
| MemberTreeTrial.csproj | App.xaml.cs | 15 | 1 | 0 | 3 |
| ExportCSV.cs | 166 | 19 | 0 | 5 |
| ExportIMG.cs | 56 | 2 | 0 | 11 |
| ExportPDF.cs | 190 | 15 | 0 | 30 |
| MainWindow.xaml.cs | 48 | 6 | 0 | 3 |
| MyTrees.cs | 284 | 30 | 0 | 18 |
| SampleData1.cs | 219 | 1 | 0 | 11 |
| SampleData2.cs | 1,020 | 1 | 0 | 11 |
| DatasetInfoView.xaml.cs | 131 | 8 | 0 | 11 |
| MyNodeInfo.xaml.cs | 105 | 11 | 0 | 13 |
| MyNodeList.xaml.cs | 70 | 4 | 0 | 3 |
| MyTreeNode.cs | 45 | 6 | 0 | 2 |
| BoxTreeStyle.xaml.cs | 54 | 5 | 0 | 12 |
| LineTreeStyle.xaml.cs | 25 | 2 | 0 | 1 |
| GraphTreeStyle.xaml.cs | 65 | 6 | 0 | 0 |
| MyStatusBar.xaml.cs | 139 | 9 | 0 | 32 |
| SearchFilter.xaml.cs | 85 | 7 | 0 | 12 |
| SearchResult.xaml.cs | 32 | 2 | 0 | 3 |
| WelcomeView.xaml.cs | 23 | 1 | 0 | 3 |
| MyTreeView.xaml.cs | 416 | 48 | 0 | 27 |
| WindowAbout.xaml.cs | 49 | 3 | 0 | 11 |
| ItemsPanelOrientationConverter.cs | 54 | 5 | 0 | 12 |
| MyBoxTreeView.xaml.cs | 145 | 10 | 0 | 36 |
| DPoint.cs | 22 | 2 | 0 | 3 |
| ITreeNode.cs | 20 | 2 | 0 | 3 |
| LayeredTreeDraw.cs | 550 | 67 | 0 | 30 |
| MyGraphView.xaml.cs | 157 | 11 | 0 | 34 |
| TreeConnection.cs | 21 | 2 | 0 | 0 |
| TreeContainer.cs | 374 | 39 | 0 | 4 |
| TreeNode.cs | 209 | 20 | 0 | 0 |
| TreeNodeGroup.cs | 61 | 13 | 0 | 0 |
| MyLineTreeView.xaml.cs | 149 | 12 | 0 | 26 |
| TreeViewLineConverter.cs | 25 | 2 | 0 | 1 |
| MySquareView.xaml.cs | 149 | 10 | 0 | 47 |
| INotify.cs | 19 | 1 | 0 | 8 |
| InvokeDelegate.cs | 17 | 1 | 0 | 8 |
| TimingUtil.cs | 56 | 3 | 0 | 11 |
| WindowView.xaml.cs | 112 | 8 | 0 | 6 |
| AssemblyInfo.cs | 53 | 8 | 0 | 26 |
| Resources.Designer.cs | 63 | 0 | 63 | 0 |
| Settings.Designer.cs | 26 | 0 | 26 | 0 |
| **合计** | **5519** | **403** | **89** | **477** |
| MemberTreeView.csproj | App.xaml.cs | 15 | 1 | 0 | 3 |
| IMyTreeDBV.cs | 31 | 4 | 0 | 11 |
| MyTreeDBVSqlite.cs | 88 | 4 | 0 | 13 |
| MyTreeDBVMysql.cs | 86 | 4 | 0 | 11 |
| ExportCSV.cs | 174 | 20 | 0 | 8 |
| ExportIMG.cs | 56 | 2 | 0 | 11 |
| ExportPDF.cs | 192 | 15 | 0 | 31 |
| MainWindow.xaml.cs | 95 | 11 | 0 | 3 |
| MyTrees.cs | 275 | 23 | 0 | 4 |
| ConnDBView.xaml.cs | 162 | 10 | 0 | 15 |
| MyNodeInfo.xaml.cs | 100 | 12 | 0 | 13 |
| MyNodeList.xaml.cs | 95 | 6 | 0 | 9 |
| MyTreeNode.cs | 32 | 4 | 0 | 1 |
| BoxTreeStyle.xaml.cs | 54 | 5 | 0 | 12 |
| LineTreeStyle.xaml.cs | 25 | 2 | 0 | 1 |
| GraphTreeStyle.xaml.cs | 65 | 6 | 0 | 0 |
| PageBar.xaml.cs | 148 | 14 | 0 | 23 |
| SearchFilter.xaml.cs | 162 | 9 | 0 | 12 |
| SearchResult.xaml.cs | 32 | 2 | 0 | 3 |
| UserInfoWindow.xaml.cs | 77 | 5 | 0 | 11 |
| MyTreeView.xaml.cs | 523 | 53 | 0 | 92 |
| ItemsPanelOrientationConverter.cs | 54 | 5 | 0 | 12 |
| MyBoxTreeView.xaml.cs | 145 | 10 | 0 | 36 |
| DPoint.cs | 22 | 2 | 0 | 3 |
| ITreeNode.cs | 20 | 2 | 0 | 3 |
| LayeredTreeDraw.cs | 550 | 67 | 0 | 30 |
| MyGraphView.xaml.cs | 157 | 11 | 0 | 34 |
| TreeConnection.cs | 21 | 2 | 0 | 0 |
| TreeContainer.cs | 374 | 39 | 0 | 4 |
| TreeNode.cs | 209 | 20 | 0 | 0 |
| TreeNodeGroup.cs | 61 | 13 | 0 | 0 |
| MyLineTreeView.xaml.cs | 149 | 12 | 0 | 26 |
| TreeViewLineConverter.cs | 25 | 2 | 0 | 1 |
| MySquareView.xaml.cs | 149 | 10 | 0 | 47 |
| WindowView.xaml.cs | 154 | 12 | 0 | 10 |
| AssemblyInfo.cs | 53 | 8 | 0 | 26 |
| Resources.Designer.cs | 63 | 0 | 63 | 0 |
| Settings.Designer.cs | 26 | 0 | 26 | 0 |
| **合计** | **4719** | **427** | **89** | **519** |
| RSACommon.csproj | BigInteger.cs | 3,129 | 705 | 0 | 555 |
| EncryptHelper.cs | 105 | 6 | 0 | 35 |
| Program.cs | 22 | 1 | 0 | 0 |
| AssemblyInfo.cs | 36 | 4 | 0 | 17 |
| RSAHelper.cs | 227 | 12 | 0 | 67 |
| **合计** | **3519** | **728** | **0** | **674** |
| SetupAdmin | Files.wxs | 32 | 1 | 0 | 2 |
| License.rtf | 12 | 1 | 0 | 0 |
| Setup.wxs | 79 | 5 | 0 | 18 |
| WixUI\_zh-cn.wxl | 527 | 26 | 0 | 0 |
| **合计** | **650** | **33** | **0** | **20** |
| SetupReg | Files.wxs | 15 | 1 | 0 | 2 |
| License.rtf | 12 | 1 | 0 | 0 |
| Setup.wxs | 51 | 5 | 0 | 18 |
| WixUI\_zh-cn.wxl | 527 | 26 | 0 | 0 |
| **合计** | **605** | **33** | **0** | **20** |
| SetupTrial | Files.wxs | 16 | 0 | 0 | 0 |
| License.rtf | 12 | 1 | 0 | 0 |
| Setup.wxs | 50 | 5 | 0 | 18 |
| WixUI\_zh-cn.wxl | 527 | 26 | 0 | 0 |
| **合计** | **605** | **32** | **0** | **18** |
| SetupView | Files.wxs | 29 | 1 | 0 | 2 |
| License.rtf | 12 | 1 | 0 | 0 |
| Setup.wxs | 70 | 5 | 0 | 18 |
| WixUI\_zh-cn.wxl | 527 | 26 | 0 | 0 |
| **合计** | **638** | **33** | **0** | **20** |
| SoftRegister.csproj | App.xaml.cs | 15 | 1 | 0 | 3 |
| AssemblyInfo.cs | 31 | 5 | 0 | 11 |
| WPFAssemblyInfo.cs | 27 | 8 | 0 | 11 |
| RegConfig.cs | 33 | 3 | 0 | 13 |
| WindowMain.xaml.cs | 122 | 5 | 0 | 15 |
| WindowHelp.xaml.cs | 31 | 1 | 0 | 11 |
| **合计** | **259** | **23** | **0** | **64** |
| SoftUnRegister.csproj | App.xaml.cs | 15 | 1 | 0 | 3 |
| AssemblyInfo.cs | 31 | 5 | 0 | 11 |
| WPFAssemblyInfo.cs | 27 | 8 | 0 | 11 |
| Window1.xaml.cs | 54 | 2 | 0 | 12 |
| **合计** | **127** | **16** | **0** | **37** |
| **全部项目数量**  **合计：12** | **166** | **2,2631** | **2206** | **267** | **2590** |

# 程序代码结构说明

## 代码项目模块说明

该项目包含12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。

|  |  |
| --- | --- |
| **项目名称** | **说明** |
| AutoUpdate | 自动更新 |
| MemberTreeCommon | 公共调用模块 |
| MemberTreeTrial | 试用版 |
| MemberTreeView | 客户端 |
| MemberTreeAdmin | 管理端 |
| RSACommon | 加密解密公共模块 |
| SoftRegister | 授权注册工具 |
| SoftUnRegister | 反注册工具 |
| SetupAdmin | 管理端安装包 |
| SetupTrial | 试用版安装包 |
| SetupView | 客户端安装包 |
| SetupSoftReg | 授权注册工具安装包 |

## 程序主要代码功能模块

* **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tab和csv数据源模块**
* **读取sqlserver和mysql数据源模块**
* **打印、导出数据（到csv文件、图片、pdf）模块**
* **查找、数据处理等模块**
* **数据读写sqlite模块**
* **数据读写mysql模块**
* **csv数据误检查模块**
* **数据动态列处理模块**
* **大数据量下分页模块**
* **用户权限管理模块**
* **数据加密解密（RSA/DES）模块**
* **注册授权模块**

## 文件后缀名说明

|  |  |  |
| --- | --- | --- |
| **文件类型** | **文件后缀名** | **文件说明** |
| 项目配置文件 | \*.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 |
| 数据文件 | \*.csv | 逗号分割文件，存储数据 |
| \*.tab | tab键分割文件，存储数据 |
| \*.key | 数据库会话信息文件，存储保存mysq数据库会话连接信息 |
| \*reginfo | 注册信息文件，保存待注册机器的机器码和注册用户信息 |
| \*.regkey | 注册密钥文件，保存注册密钥，用以用户进行授权注册 |
| **源代码文件** | **\*.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** |
| **安装包配置文件** | **\*.wxs** | **安装包制作源代码文件** |
| **\*.wxl** | **安装包制作资源文件** |

上述文件类型中，只有最后几个加粗的文件类型，可能是我们需要手工编写的。

# 服务端数据管理模块

## 主程序界面布局代码

<?xml version="1.0" encoding="utf-8"?>  
<UserControl   
    x:Class="MemberTree.WindowAdmin"  
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"   
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"  
    xmlns:my="clr-namespace:MemberTree"  
    xmlns:my1="clr-namespace:MemberTree;assembly=MemberTreeCommon">  
    <Grid>  
        <Grid.RowDefinitions>  
            <RowDefinition Height="auto" />  
            <RowDefinition />  
            <RowDefinition Height="auto" />  
        </Grid.RowDefinitions>  
        <TabControl x:Name="mainTab" Grid.RowSpan="2">  
            <TabItem>  
                <TabItem.Header>  
                    <TextBlock Margin="10,5" FontWeight="Bold" FontSize="15" Text="会员关系树数据集管理" MouseDown="TextBlock\_MouseDown"/>  
                </TabItem.Header>  
                <my:AdminDataSet x:Name="adminDataset"/>  
            </TabItem>  
            <TabItem x:Name="tabAdminUser" Visibility="Hidden">  
                <TabItem.Header>  
                    <TextBlock Margin="10,5" FontWeight="Bold" FontSize="15" Text="用户及权限管理"/>  
                </TabItem.Header>  
                <my:AdminUser x:Name="adminUser" />  
            </TabItem>  
        </TabControl>  
        <my1:ProgressView Grid.RowSpan="2" x:Name="progressView" Visibility="Hidden"/>  
        <my1:MyStatusBar Grid.Row="2" x:Name="myStatusBar"/>  
    </Grid>  
</UserControl>

## 数据管理界面代码

<UserControl x:Class="MemberTree.AdminDataSet"  
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"  
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"  
    xmlns:my="clr-namespace:MemberTree"  
    xmlns:my1="clr-namespace:MemberTree;assembly=MemberTreeCommon">  
    <UserControl.Resources>  
        <Style TargetType="Button">  
            <Setter Property="Margin" Value="2" />  
            <Setter Property="Padding" Value="2" />  
        </Style>  
        <Style TargetType="ComboBox">  
            <Setter Property="Margin" Value="2" />  
        </Style>  
    </UserControl.Resources>  
    <Grid>  
        <Grid.RowDefinitions>  
            <RowDefinition />  
            <RowDefinition Height="auto" />  
        </Grid.RowDefinitions>  
        <Grid.ColumnDefinitions>  
            <ColumnDefinition Width="Auto"/>  
            <ColumnDefinition/>  
        </Grid.ColumnDefinitions>  
        <StackPanel Grid.RowSpan="2" Background="Azure">  
            <GroupBox BorderThickness="1" BorderBrush="DarkBlue">  
                <GroupBox.Header>  
                    <StackPanel Orientation="Horizontal">  
                        <Image Source="/MemberTreeAdmin;component/Image/option.jpg" Width="36" Height="36"/>  
                        <TextBlock Text="数据导入选项" FontSize="16" VerticalAlignment="Center"/>  
                    </StackPanel>  
                </GroupBox.Header>  
                <StackPanel>  
                    <ComboBox Name="comboToLower">  
                        <ComboBoxItem IsSelected="True">不转换大小写</ComboBoxItem>  
                        <ComboBoxItem>转换为小写</ComboBoxItem>  
                        <ComboBoxItem>转换为大写</ComboBoxItem>  
                    </ComboBox>  
                    <ComboBox Name="comboToHalf">  
                        <ComboBoxItem IsSelected="True">不转换全半角</ComboBoxItem>  
                        <ComboBoxItem>转换为半角</ComboBoxItem>  
                        <ComboBoxItem>转换为全角</ComboBoxItem>  
                    </ComboBox>  
                    <ComboBox Name="comboTrim">  
                        <ComboBoxItem IsSelected="True">不删除空格</ComboBoxItem>  
                        <ComboBoxItem>删除首尾空格</ComboBoxItem>  
                        <ComboBoxItem>删除首部空格</ComboBoxItem>  
                        <ComboBoxItem>删除尾部空格</ComboBoxItem>  
                    </ComboBox>  
                </StackPanel>  
            </GroupBox>  
            <GroupBox BorderThickness="1" BorderBrush="DarkBlue">  
                <GroupBox.Header>  
                    <StackPanel Orientation="Horizontal">  
                        <Image Source="/MemberTreeAdmin;component/Image/import.jpg" Width="36" Height="36"/>  
                        <TextBlock Text="导入数据并计算" FontSize="16" VerticalAlignment="Center"/>  
                    </StackPanel>  
                </GroupBox.Header>  
                <StackPanel>  
                    <GroupBox BorderBrush="LightBlue">  
                        <GroupBox.Header>  
                            <StackPanel Orientation="Horizontal">  
                                <Image Source="/MemberTreeAdmin;component/Image/db.jpg" Width="30" Height="30"/>  
                                <TextBlock Text="从数据库导入" FontSize="14" VerticalAlignment="Center"/>  
                            </StackPanel>  
                        </GroupBox.Header>  
                        <StackPanel>  
                            <Button Click="ButtonConnSqlserver\_Click">  
                                <StackPanel Orientation="Horizontal">  
                                    <Image Source="/MemberTreeAdmin;component/Image/sqlserver.jpg" Width="25" Height="25"/>  
                                    <TextBlock Text="从sqlserver导入" FontSize="12" VerticalAlignment="Center"/>  
                                </StackPanel>  
                            </Button>  
                            <Button Click="ButtonConnectMysql\_Click">  
                                <StackPanel Orientation="Horizontal">  
                                    <Image Source="/MemberTreeAdmin;component/Image/mysql.jpg" Width="25" Height="25"/>  
                                    <TextBlock Text="从mysql导入" FontSize="12" VerticalAlignment="Center"/>  
                                </StackPanel>  
                            </Button>  
                        </StackPanel>  
                    </GroupBox>  
                    <GroupBox BorderBrush="LightBlue">  
                        <GroupBox.Header>  
                            <StackPanel Orientation="Horizontal">  
                                <Image Source="/MemberTreeAdmin;component/Image/file.jpg" Width="30" Height="30"/>  
                                <TextBlock Text="从文件导入" FontSize="14" VerticalAlignment="Center"/>  
                            </StackPanel>  
                        </GroupBox.Header>  
                        <StackPanel>  
                            <Button Click="ButtonCheck\_Click">  
                                <StackPanel Orientation="Horizontal">  
                                    <Image Source="/MemberTreeAdmin;component/Image/check.jpg" Width="25" Height="25"/>  
                                    <TextBlock Text="检查csv文件" FontSize="12" VerticalAlignment="Center"/>  
                                </StackPanel>  
                            </Button>  
                            <Button Click="ButtonOpen\_Click" x:Name="btnImportCsv">  
                                <StackPanel Orientation="Horizontal">  
                                    <Image Source="/MemberTreeAdmin;component/Image/csv.jpg" Width="25" Height="25"/>  
                                    <TextBlock Text="导入csv文件" FontSize="12" VerticalAlignment="Center"/>  
                                </StackPanel>  
                            </Button>  
                            <Button Click="ButtonOpen\_Click" x:Name="btnImportTab">  
                                <StackPanel Orientation="Horizontal">  
                                    <Image Source="/MemberTreeAdmin;component/Image/tab.jpg" Width="25" Height="25"/>  
                                    <TextBlock Text="导入tab文件" FontSize="12" VerticalAlignment="Center"/>  
                                </StackPanel>  
                            </Button>  
                        </StackPanel>  
                    </GroupBox>  
                </StackPanel>  
            </GroupBox>  
            <GroupBox BorderThickness="1" BorderBrush="DarkBlue">  
                <GroupBox.Header>  
                    <StackPanel Orientation="Horizontal">  
                        <Image Source="/MemberTreeAdmin;component/Image/manage.jpg" Width="36" Height="36"/>  
                        <TextBlock Text="数据集管理" FontSize="16" VerticalAlignment="Center"/>  
                    </StackPanel>  
                </GroupBox.Header>  
                <Button Click="ButtonDelete\_Click" >  
                    <StackPanel Orientation="Horizontal">  
                        <Image Source="/MemberTreeAdmin;component/Image/delete.jpg" Width="25" Height="25"/>  
                        <TextBlock Text="删除选中的数据集" FontSize="12" VerticalAlignment="Center"/>  
                    </StackPanel>  
                </Button>  
            </GroupBox>  
            <GroupBox BorderThickness="1" BorderBrush="DarkBlue">  
                <GroupBox.Header>  
                    <StackPanel Orientation="Horizontal">  
                        <Image Source="/MemberTreeAdmin;component/Image/about.jpg" Width="36" Height="36"/>  
                        <TextBlock Text="关于" FontSize="16" VerticalAlignment="Center"/>  
                    </StackPanel>  
                </GroupBox.Header>  
                <StackPanel>  
                    <Button Margin="10,5" Padding="10,5" Content="软件功能特点" Click="BtnAbout\_Click"/>  
                    <Button Margin="10,5" Padding="10,5" Content="历史版本记录" Click="BtnVerLog\_Click"/>  
                </StackPanel>  
            </GroupBox>  
        </StackPanel>  
        <my1:DatasetListView Grid.Column="1" x:Name="datasetListView" />  
        <my1:DatasetInfoView Grid.Row="1" Grid.Column="1" x:Name="datasetInfoView"/>  
    </Grid>  
</UserControl>

## 数据集概要信息表示代码

**namespace** MemberTree  
{  
    /// <**summary**>  
    /// 所有数据集的集合列表  
    /// <**/summary**>  
    **public** partial class DatasetListView : UserControl  
    {  
        **private** InvokeStringDelegate *startupDelegate*;  
        **private** DatasetBtn *selectDatasetBtn* = **null**;  
        **public** DatasetListView()  
        {  
            **InitializeComponent**();  
        }  
          
        **public** string **GetSelectDataset**()  
        {  
            **if**(*selectDatasetBtn* == **null**)  
            {  
                return **null**;  
            }  
            **else**  
            {  
                return *selectDatasetBtn*.DatasetName;  
            }  
        }  
          
        **public** void **RefreshDB**(IMyTreeDB treeDB, string userId)  
        {  
            *mainPanel*.Children.**Clear**();  
            List<DatasetInfo> dbList = treeDB.**GetDatasets**();  
            **if**(userId != "") //如果用户权限启用，则进行用户数据集权限筛选  
            {  
                List<DatasetInfo> allowDbList = **new** List<DatasetInfo>();  
                List<string> allowDbName = UserAdmin.**GetAllowDataByUser**(userId);  
                **foreach** (DatasetInfo db **in** dbList)   
                {  
                    **if**(allowDbName.**Contains**(db.*Name*))  
                    {  
                        allowDbList.**Add**(db);  
                    }  
                }  
                dbList = allowDbList;  
            }  
            **if**(dbList.Count > 0)  
            {  
                **foreach** (DatasetInfo db **in** dbList)  
                {  
                    DatasetBtn btn = **new** DatasetBtn(db);  
                    btn.MouseDown += **Btn\_Click**;  
                    btn.Background = Brushes.Azure;  
                    *mainPanel*.Children.**Add**(btn);  
                }  
            }  
            **else**  
            {  
                Button btn = **new** Button();  
                btn.Content = "没有发现可用的数据！";  
                btn.Height = 50;  
                btn.Width = 200;  
                btn.Background = Brushes.Red;  
                *mainPanel*.Children.**Add**(btn);  
            }  
        }  
          
        **public** void **SetCallBack**(InvokeStringDelegate startupDelegate)  
        {  
            **this**.*startupDelegate* = startupDelegate;  
        }  
          
        **private** void **Btn\_Click**(object sender, RoutedEventArgs e)  
        {  
            DatasetBtn btn = sender **as** DatasetBtn;  
            **if**(*selectDatasetBtn* != **null**)  
            {  
                **if**(*selectDatasetBtn* == btn)  
                {  
                    return;  
                }  
                *selectDatasetBtn*.**UnSelect**();  
            }  
            btn.**Select**();  
            *selectDatasetBtn* = btn;  
              
            **if**(*startupDelegate* != **null**)  
            {  
                *startupDelegate*.**Invoke**(btn.DatasetName);  
            }  
        }  
          
        **public** void **DeleteBtn**(string btnTxt)  
        {  
            **if**(*selectDatasetBtn*.DatasetName == btnTxt)  
            {  
                *mainPanel*.Children.**Remove**(*selectDatasetBtn*);  
                *selectDatasetBtn* = **null**;  
            }  
            **else**  
            {  
                **foreach** (UIElement ele **in** *mainPanel*.Children)  
                {  
                    DatasetBtn btn = ele **as** DatasetBtn;  
                    **if**(btn.DatasetName == btnTxt)  
                    {  
                        *mainPanel*.Children.**Remove**(btn);  
                        break;  
                    }  
                }  
            }  
        }  
    }  
}

## 数据集列表代码

**namespace** MemberTree  
{  
    /// <**summary**>  
    /// 所有数据集的集合列表  
    /// <**/summary**>  
    **public** partial class DatasetListView : UserControl  
    {  
        **private** InvokeStringDelegate *startupDelegate*;  
        **private** DatasetBtn *selectDatasetBtn* = **null**;  
        **public** DatasetListView()  
        {  
            **InitializeComponent**();  
        }  
          
        **public** string **GetSelectDataset**()  
        {  
            **if**(*selectDatasetBtn* == **null**)  
            {  
                return **null**;  
            }  
            **else**  
            {  
                return *selectDatasetBtn*.DatasetName;  
            }  
        }  
          
        **public** void **RefreshDB**(IMyTreeDB treeDB, string userId)  
        {  
            *mainPanel*.Children.**Clear**();  
            List<DatasetInfo> dbList = treeDB.**GetDatasets**();  
            **if**(userId != "") //如果用户权限启用，则进行用户数据集权限筛选  
            {  
                List<DatasetInfo> allowDbList = **new** List<DatasetInfo>();  
                List<string> allowDbName = UserAdmin.**GetAllowDataByUser**(userId);  
                **foreach** (DatasetInfo db **in** dbList)   
                {  
                    **if**(allowDbName.**Contains**(db.*Name*))  
                    {  
                        allowDbList.**Add**(db);  
                    }  
                }  
                dbList = allowDbList;  
            }  
            **if**(dbList.Count > 0)  
            {  
                **foreach** (DatasetInfo db **in** dbList)  
                {  
                    DatasetBtn btn = **new** DatasetBtn(db);  
                    btn.MouseDown += **Btn\_Click**;  
                    btn.Background = Brushes.Azure;  
                    *mainPanel*.Children.**Add**(btn);  
                }  
            }  
            **else**  
            {  
                Button btn = **new** Button();  
                btn.Content = "没有发现可用的数据！";  
                btn.Height = 50;  
                btn.Width = 200;  
                btn.Background = Brushes.Red;  
                *mainPanel*.Children.**Add**(btn);  
            }  
        }  
          
        **public** void **SetCallBack**(InvokeStringDelegate startupDelegate)  
        {  
            **this**.*startupDelegate* = startupDelegate;  
        }  
          
        **private** void **Btn\_Click**(object sender, RoutedEventArgs e)  
        {  
            DatasetBtn btn = sender **as** DatasetBtn;  
            **if**(*selectDatasetBtn* != **null**)  
            {  
                **if**(*selectDatasetBtn* == btn)  
                {  
                    return;  
                }  
                *selectDatasetBtn*.**UnSelect**();  
            }  
            btn.**Select**();  
            *selectDatasetBtn* = btn;  
              
            **if**(*startupDelegate* != **null**)  
            {  
                *startupDelegate*.**Invoke**(btn.DatasetName);  
            }  
        }  
          
        **public** void **DeleteBtn**(string btnTxt)  
        {  
            **if**(*selectDatasetBtn*.DatasetName == btnTxt)  
            {  
                *mainPanel*.Children.**Remove**(*selectDatasetBtn*);  
                *selectDatasetBtn* = **null**;  
            }  
            **else**  
            {  
                **foreach** (UIElement ele **in** *mainPanel*.Children)  
                {  
                    DatasetBtn btn = ele **as** DatasetBtn;  
                    **if**(btn.DatasetName == btnTxt)  
                    {  
                        *mainPanel*.Children.**Remove**(btn);  
                        break;  
                    }  
                }  
            }  
        }  
    }  
}

# 服务端用户管理模块

## 用户管理主界面代码

<UserControl x:Class="MemberTree.AdminUser"  
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"  
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"  
    xmlns:my="clr-namespace:MemberTree">  
    <Grid>  
        <Grid.RowDefinitions>  
            <RowDefinition Height="Auto"/>  
            <RowDefinition/>  
        </Grid.RowDefinitions>  
  
        <StackPanel Background="Azure" Orientation="Horizontal">  
            <CheckBox Content="是否启用用户及权限管理功能" x:Name="ckEnable" Margin="20,10,10,10" Click="CkEnable\_Click"/>  
              
            <ToolBar Name="toolBar">  
                <RadioButton Content="用户基本信息管理" Name="btnUserAdmin" Margin="10,2" Padding="5,2"   
                             BorderBrush="Black" BorderThickness="1" Checked="switchTabPage\_Checked"/>  
                <RadioButton Content="权限管理(用户——>数据集)" Name="btnUsrDst" Margin="10,2" Padding="5,2"   
                             BorderBrush="Black" BorderThickness="1" Checked="switchTabPage\_Checked"/>  
                <RadioButton Content="权限管理(数据集——>用户)" Name="btnDstUsr" Margin="10,2" Padding="5,2"   
                             BorderBrush="Black" BorderThickness="1" Checked="switchTabPage\_Checked"/>  
            </ToolBar>  
        </StackPanel>  
          
        <Grid Name="gridContent" Grid.Row="1"/>  
    </Grid>  
</UserControl>

## 管理用户信息代码

**namespace** MemberTree  
{  
    /// <**summary**>  
    /// Interaction logic for UserInfoSet.xaml  
    /// <**/summary**>  
    **public** partial class UserInfoSet : UserControl  
    {  
        **public** UserInfoSet()  
        {  
            **InitializeComponent**();  
        }  
           
        **public** void **RefreshUserList**()  
        {  
            *userList*.ItemsSource = UserAdmin.**GetUserInfoList**();  
        }  
          
        **private** void **SetEnabled**(**bool** enabled, **params** Control[] controls)  
        {  
            **foreach** (Control control **in** controls)   
            {  
                control.IsEnabled = enabled;  
            }  
        }  
          
        **private** **bool** **CheckTxtIsNotNull**(TextBox txtBox, string txtTip)  
        {  
            **if**(txtBox.Text == "")  
            {  
                txtBox.BorderBrush = Brushes.Red;  
                MessageBox.**Show**(txtTip + "不能为空！");  
                WindowAdmin.*notify*.**SetStatusMessage**(txtTip + "不能为空！");  
                return **false**;  
            }  
            **else**  
            {  
                txtBox.BorderBrush = Brushes.LightBlue;  
                return **true**;  
            }  
        }  
          
        //选择用户变化  
        void **UserList\_SelectionChanged**(object sender, SelectionChangedEventArgs e)  
        {  
            UserInfo userInfo = *userList*.SelectedItem **as** UserInfo;  
            **if**(userInfo != **null**)  
            {  
                **SetEnabled**(**true**, *btnModify*, *btnResetPwd*, *btnDelete*);  
  
                *txtID*.Text = userInfo.ID;  
                *txtName*.Text = userInfo.Name;  
                *txtRemark*.Text = userInfo.Remark;  
                WindowAdmin.*notify*.**SetStatusMessage**("当前选中的"+userInfo.**ToLongString**());  
            }  
            **else**  
            {  
                **SetEnabled**(**false**, *btnModify*, *btnResetPwd*, *btnDelete*);  
            }  
        }  
          
        //新增用户  
        void **BtnNew\_Click**(object sender, RoutedEventArgs e)  
        {  
            *gridUserInfo*.Visibility = **Visibility**.*Visible*;  
            *userList*.IsEnabled = **false**;  
            *userList*.SelectedIndex = -1;  
            *btnNew*.IsEnabled = **false**;  
            *txtID*.IsEnabled = **true**;  
            *txtID*.**Clear**();  
            *txtName*.**Clear**();  
            *txtRemark*.**Clear**();  
            *txtPwd1*.**Clear**();  
            *txtPwd2*.**Clear**();  
            WindowAdmin.*notify*.**SetStatusMessage**("开始新增用户。。。");  
        }  
          
        //修改用户信息  
        void **BtnModify\_Click**(object sender, RoutedEventArgs e)  
        {  
            *gridUserInfo*.Visibility = **Visibility**.*Visible*;  
            **SetEnabled**(**false**, *btnNew*, *btnModify*, *btnResetPwd*, *btnDelete*, *userList*);  
            WindowAdmin.*notify*.**SetStatusMessage**("正在修改用户信息。。。");  
        }  
          
        //重置用户密码  
        void **BtnResetPwd\_Click**(object sender, RoutedEventArgs e)  
        {  
            *gridUserPwd*.Visibility = **Visibility**.*Visible*;  
            **SetEnabled**(**false**, *btnNew*, *btnModify*, *btnResetPwd*, *btnDelete*, *userList*);  
            WindowAdmin.*notify*.**SetStatusMessage**("正在重置用户密码。。。");  
        }  
          
        //保存用户信息  
        void **BtnSaveUser\_Click**(object sender, RoutedEventArgs e)  
        {  
            **if**(**CheckTxtIsNotNull**(*txtID*, "用户ID") && **CheckTxtIsNotNull**(*txtName*, "用户姓名"))  
            {  
                **if**(!*txtID*.IsEnabled) //修改现有的用户信息  
                {  
                    UserAdmin.**UpdateUserInfo**(*txtID*.Text, *txtName*.Text, *txtRemark*.Text);  
                    **RefreshUserList**();  
                    *gridUserInfo*.Visibility = **Visibility**.*Collapsed*;  
                    *userList*.IsEnabled = **true**;  
                    *btnNew*.IsEnabled = **true**;  
                    WindowAdmin.*notify*.**SetStatusMessage**("修改用户信息成功!");  
                }  
                **else** **if**(UserAdmin.**GetUserInfo**(*txtID*.Text) != **null**) //首先判断ID是否已存在  
                {  
                    *txtID*.BorderBrush = Brushes.Red;  
                    *txtID*.**SelectAll**();  
                    MessageBox.**Show**("当前用户ID在数据库中已存在，请使用其他ID！");  
                    WindowAdmin.*notify*.**SetStatusMessage**("当前用户ID在数据库中已存在，请使用其他ID！");  
                }  
                **else** //增加新的用户  
                {  
                    *gridUserInfo*.Visibility = **Visibility**.*Collapsed*;  
                    *gridUserPwd*.Visibility = **Visibility**.*Visible*;  
                }  
            }  
        }  
          
        //修改用户密码  
        void **BtnSavePwd\_Click**(object sender, RoutedEventArgs e)  
        {  
            **if**(*txtPwd1*.Password == *txtPwd2*.Password && *txtPwd1*.Password != "")  
            {  
                **if**(*txtID*.IsEnabled)  
                {  
                    UserInfo userInfo = **new** UserInfo(*txtID*.Text, *txtName*.Text, EncryptHelper.**Encrypt**(*txtPwd1*.Password), *txtRemark*.Text);  
                    UserAdmin.**AddUserInfo**(userInfo);  
                    **RefreshUserList**();  
                    *txtID*.IsEnabled = **false**;  
                    WindowAdmin.*notify*.**SetStatusMessage**("增加新的用户成功!");  
                }  
                **else**  
                {  
                    UserAdmin.**UpdateUserPwd**(*txtID*.Text, EncryptHelper.**Encrypt**(*txtPwd1*.Password));  
                    **SetEnabled**(**true**, *btnModify*, *btnResetPwd*, *btnDelete*);  
                    WindowAdmin.*notify*.**SetStatusMessage**("重置用户密码成功!");  
                }  
                *gridUserPwd*.Visibility = **Visibility**.*Collapsed*;  
                *userList*.IsEnabled = **true**;  
                *btnNew*.IsEnabled = **true**;  
            }  
            **else**  
            {  
                MessageBox.**Show**("两次输入的密码不一致！");  
                WindowAdmin.*notify*.**SetStatusMessage**("两次输入的密码不一致!");  
            }  
        }  
          
        //取消保存用户信息或修改用户密码  
        void **BtnCancel\_Click**(object sender, RoutedEventArgs e)  
        {  
            **if**(*txtID*.IsEnabled) //增加新的用户  
            {  
                *txtID*.IsEnabled = **false**;  
                WindowAdmin.*notify*.**SetStatusMessage**("已经取消新增用户！");  
            }  
            **else**  //修改现有的用户信息  
            {  
                **SetEnabled**(**true**, *btnModify*, *btnResetPwd*, *btnDelete*);  
                WindowAdmin.*notify*.**SetStatusMessage**("已经取消修改用户信息！");  
            }  
            Button btn = sender **as** Button;  
            Grid grid = btn.Parent **as** Grid;  
            grid.Visibility = **Visibility**.*Collapsed*;  
            *userList*.IsEnabled = **true**;  
            *btnNew*.IsEnabled = **true**;  
        }  
          
        //修改是否启用  
        void **Enable\_Check**(object sender, RoutedEventArgs e)  
        {  
            CheckBox checkBox = e.Source **as** CheckBox;  
            **if**(checkBox != **null**)  
            {  
                string userId = checkBox.Tag.**ToString**();  
                **bool** isEnable = (**bool**)checkBox.IsChecked;  
                UserAdmin.**UpdateUserEnabled**(userId, isEnable);  
                  
                string enableStr = isEnable ? "启用" : "停用";  
                WindowAdmin.*notify*.**SetStatusMessage**("已经将用户"+userId+enableStr);  
            }  
        }  
          
        //删除  
        void **BtnDelete\_Click**(object sender, RoutedEventArgs e)  
        {  
            string userStr = string.**Format**("用户ID：{0}，用户姓名：{1}", *txtID*.Text, *txtName*.Text);  
            **if**(MessageBox.**Show**("确定要删除该用户?\n"+userStr, "删除确认", **MessageBoxButton**.*YesNo*) == **MessageBoxResult**.*Yes*)  
            {  
                UserAdmin.**DeleteUserInfo**(*txtID*.Text);  
                **RefreshUserList**();  
                WindowAdmin.*notify*.**SetStatusMessage**("删除用户成功！被删除的" + userStr);  
            }  
        }  
    }  
}

## 管理用户权限代码

**namespace** MemberTree  
{  
    /// <**summary**>  
    /// Interaction logic for UserPrivilege.xaml  
    /// <**/summary**>  
    **public** partial class UserPrivilege1 : UserControl  
    {  
        **private** BtnUserInfo *selectedUserBtn*;  
        **public** UserPrivilege1()  
        {  
            **InitializeComponent**();  
              
            **RefreshUserList**();  
            **RefreshDatasetList**();  
        }  
          
        **public** void **RefreshUserList**()  
        {  
            *panelUser*.Children.**Clear**();  
            List<UserInfo> userList = UserAdmin.**GetUserInfoList**();  
            **foreach** (UserInfo user **in** userList) {  
                BtnUserInfo btn = **new** BtnUserInfo(user);  
                btn.MouseDown += **btnUser\_Click**;  
                *panelUser*.Children.**Add**(btn);  
            }  
        }  
          
        **public** void **RefreshDatasetList**()  
        {  
            *panelDataset*.Children.**Clear**();  
            List<DatasetInfo> dbList = MyTrees.*treeDB*.**GetDatasets**();  
            **foreach** (DatasetInfo db **in** dbList) {  
                BtnDataset cbx = **new** BtnDataset(db);  
                cbx.MouseDown += **cbxDataset\_Click**;  
                *panelDataset*.Children.**Add**(cbx);  
            }  
        }  
          
        //选中某个用户时，可以修改该用户对应的数据集权限  
        **private** void **btnUser\_Click**(object sender, RoutedEventArgs e)  
        {  
            BtnUserInfo btnUser = sender **as** BtnUserInfo;  
            **if**(*selectedUserBtn* != **null**)  
            {  
                *selectedUserBtn*.isSelected = **false**;  
            }  
            *selectedUserBtn* = btnUser;  
            btnUser.isSelected = **true**;  
            *btnModify*.IsEnabled = **true**;  
              
            //根据用户ID获取对应的数据权限  
            List<string> allowData = UserAdmin.**GetAllowDataByUser**(btnUser.UserId);  
            **foreach** (BtnDataset cbx **in** *panelDataset*.Children)  
            {  
                cbx.isSelected = (allowData.**Contains**(cbx.DatasetName));  
            }  
        }  
          
        //点击某个数据集时，自动改变其选中状态  
        **private** void **cbxDataset\_Click**(object sender, RoutedEventArgs e)  
        {  
            BtnDataset cbx = sender **as** BtnDataset;  
            cbx.isSelected = !cbx.isSelected;  
        }  
          
        void **BtnModify\_Click**(object sender, RoutedEventArgs e)  
        {  
            *btnSelectAll*.IsEnabled = **true**;  
            *btnSelectNone*.IsEnabled = **true**;  
            *btnSave*.IsEnabled = **true**;  
            *btnCancel*.IsEnabled = **true**;  
            *panelDataset*.IsEnabled = **true**;  
            *panelUser*.IsEnabled = **false**;  
            *btnModify*.IsEnabled = **false**;  
        }  
          
        void **BtnSelectAll\_Click**(object sender, RoutedEventArgs e)  
        {  
            **foreach** (BtnDataset cbx **in** *panelDataset*.Children)  
            {  
                cbx.isSelected = **true**;  
            }  
        }  
      
        void **BtnSelectNone\_Click**(object sender, RoutedEventArgs e)  
        {  
            **foreach** (BtnDataset cbx **in** *panelDataset*.Children)  
            {  
                cbx.isSelected = **false**;  
            }  
        }  
          
        void **BtnSave\_Click**(object sender, RoutedEventArgs e)  
        {  
            //根据用户ID获取对应的数据权限  
            string usrId = *selectedUserBtn*.UserId;  
            List<string> allowData = UserAdmin.**GetAllowDataByUser**(usrId);  
            **foreach** (BtnDataset cbx **in** *panelDataset*.Children)  
            {  
                string dsName = cbx.DatasetName;  
                **if**((**bool**)cbx.isSelected)  
                {  
                    **if**(!allowData.**Contains**(dsName))  
                    {  
                        UserAdmin.**AddUserPrivilege**(usrId, dsName);  
                    }  
                }  
                **else**  
                {  
                    **if**(allowData.**Contains**(dsName))  
                    {  
                        UserAdmin.**DeleteUserPrivilege**(usrId, dsName);  
                    }  
                }  
            }  
              
            *btnSelectAll*.IsEnabled = **false**;  
            *btnSelectNone*.IsEnabled = **false**;  
            *btnSave*.IsEnabled = **false**;  
            *btnCancel*.IsEnabled = **false**;  
            *panelDataset*.IsEnabled = **false**;  
            *panelUser*.IsEnabled = **true**;  
            *btnModify*.IsEnabled = **true**;  
            WindowAdmin.*notify*.**SetStatusMessage**(string.**Format**("成功修改了用户{0}的所对应的数据集权限！", usrId));  
        }  
          
        void **btnCancel\_Click**(object sender, RoutedEventArgs e)  
        {  
            *btnSelectAll*.IsEnabled = **false**;  
            *btnSelectNone*.IsEnabled = **false**;  
            *btnSave*.IsEnabled = **false**;  
            *btnCancel*.IsEnabled = **false**;  
            *panelDataset*.IsEnabled = **false**;  
            *panelUser*.IsEnabled = **true**;  
            *btnModify*.IsEnabled = **true**;  
              
            //根据用户ID获取对应的数据权限  
            List<string> allowData = UserAdmin.**GetAllowDataByUser**(*selectedUserBtn*.UserId);  
            **foreach** (BtnDataset cbx **in** *panelDataset*.Children)  
            {  
                cbx.isSelected = (allowData.**Contains**(cbx.DatasetName));  
            }  
        }  
    }  
}

# 服务端直连数据库模块

## 连接mysql/sqlser界面代码

<?xml version="1.0" encoding="utf-8"?>  
<UserControl x:Class="MemberTree.ConnDBView"  
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"   
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">  
    <UserControl.Resources>  
        <Style TargetType="TextBlock">  
            <Setter Property="Padding" Value="5" />  
            <Setter Property="VerticalAlignment" Value="Center" />  
            <Setter Property="HorizontalAlignment" Value="Right" />  
        </Style>  
        <Style TargetType="TextBox">  
            <Setter Property="Width" Value="120" />  
            <Setter Property="Height" Value="20" />  
            <Setter Property="HorizontalAlignment" Value="Left" />  
            <Setter Property="VerticalAlignment" Value="Center" />  
        </Style>  
        <Style TargetType="Button">  
            <Setter Property="Padding" Value="5,3" />  
            <Setter Property="HorizontalAlignment" Value="Center" />  
            <Setter Property="VerticalAlignment" Value="Center" />  
            <Setter Property="IsEnabled" Value="False" />  
        </Style>  
    </UserControl.Resources>  
    <Grid Margin="5">  
        <Grid.ColumnDefinitions>  
            <ColumnDefinition/>  
            <ColumnDefinition/>  
            <ColumnDefinition/>  
            <ColumnDefinition/>  
            <ColumnDefinition/>  
            <ColumnDefinition Width="1.5\*"/>  
        </Grid.ColumnDefinitions>  
        <Grid.RowDefinitions>  
            <RowDefinition/>  
            <RowDefinition Height="Auto"/>  
        </Grid.RowDefinitions>  
        <GroupBox Header="会话列表" Grid.ColumnSpan="2">  
            <ListBox x:Name="sessionList" SelectionChanged="SessionList\_SelectionChanged"/>  
        </GroupBox>  
        <GroupBox Header="会话信息" Grid.Column="2" Grid.ColumnSpan="4">  
            <Grid x:Name="gridSessionInfo" IsEnabled="False">  
                <Grid.ColumnDefinitions>  
                    <ColumnDefinition />  
                    <ColumnDefinition Width="2\*" />  
                </Grid.ColumnDefinitions>  
                <Grid.RowDefinitions>  
                    <RowDefinition/>  
                    <RowDefinition Height="2\*"/>  
                    <RowDefinition/>  
                    <RowDefinition/>  
                    <RowDefinition/>  
                    <RowDefinition/>  
                </Grid.RowDefinitions>  
                  
                <TextBlock Text="会话名称："/>  
                <TextBox Grid.Column="1" x:Name="txtSessionName" Text="{Binding SessionName}" TextChanged="SessionChanged"/>  
                  
                <TextBlock Grid.Row="1" Text="备注信息："/>  
                <TextBox Grid.Row="1" Grid.Column="1" x:Name="txtSessionRemark" TextChanged="SessionChanged"  
                         Text="{Binding SessionRemark}" Width="150" Height="50" TextWrapping="WrapWithOverflow"/>  
                  
                <TextBlock Grid.Row="2" Text="服务器IP："/>  
                <TextBox Grid.Row="2" Grid.Column="1" x:Name="txtDBServer" Text="{Binding ServerIP}" TextChanged="TextChanged"/>  
                  
                <TextBlock Grid.Row="3" Text="端口号："/>  
                <TextBox Grid.Row="3" Grid.Column="1" x:Name="txtDBPort" Text="{Binding Port}" TextChanged="TextChanged"/>  
          
                <TextBlock Grid.Row="4" Text="用户名："/>  
                <TextBox Grid.Row="4" Grid.Column="1" x:Name="txtDBUserID" Text="{Binding UserID}" TextChanged="TextChanged"/>  
                  
                <TextBlock Grid.Row="5" Text="密码："/>  
                <PasswordBox Grid.Row="5" Grid.Column="1" x:Name="txtDBPwd" Width="120" Height="20"  
                             HorizontalAlignment="Left" PasswordChanged="TxtPwd\_PasswordChanged"/>  
            </Grid>  
        </GroupBox>  
        <Button Grid.Row="1" IsEnabled="True" Content="新建会话" x:Name="btnNew" Click="btnNew\_Click" />  
        <Button Grid.Row="1" Grid.Column="1" Content="删除会话" x:Name="btnDelete" Click="btnDelete\_Click"/>  
        <Button Grid.Row="1" Grid.Column="3" Content="保存修改" x:Name="btnSave" Click="btnSave\_Click"/>  
        <Button Grid.Row="1" Grid.Column="4" Content="测试连接" x:Name="btnTest" Click="btnTest\_Click"/>  
        <Button Grid.Row="1" Grid.Column="5" Content="连接到数据库" x:Name="btnConnect" Click="btnConnect\_Click"/>  
    </Grid>  
</UserControl>

## 连接sqlser业务代码

**namespace** MemberTree  
{  
    /// <**summary**>  
    /// Description of ConnMysql.  
    /// <**/summary**>  
    **public** class ConnMysql : IConnDB  
    {  
        **private** MySqlConnection *conn*;  
        **private** MySqlCommand *cmd*;  
          
        **public** ConnMysql()  
        {  
        }  
  
        #**region** IConnDB implementation  
  
        **public** **bool** **Connect**(string IP, string usr, string pwd, string portstr)  
        {  
            **uint** port  = 3306;  
            **if**(!**uint**.**TryParse**(portstr, **out** port))  
            {  
                MessageBox.**Show**("端口号必须为大于0的数字！");  
            }      
              
            MySqlConnectionStringBuilder sqlStrBuilder = **new** MySqlConnectionStringBuilder();  
            sqlStrBuilder.Server = IP; //server ip  
            sqlStrBuilder.Port = port; //端口号  
            sqlStrBuilder.UserID = usr;  //用户名  
            sqlStrBuilder.Password = pwd;  //密码  
  
            *conn* = **new** MySqlConnection(sqlStrBuilder.ConnectionString);  
            **try** {  
                *conn*.**Open**();  
                **if**(*conn*.**Ping**())  
                {  
                    *cmd* = **new** MySqlCommand();  
                    *cmd*.Connection = *conn*;  
                    *conn*.**Close**();  
                    return **true**;  
                }  
                *conn*.**Close**();  
                return **false**;  
            } **catch** (Exception ex) {  
                MessageBox.**Show**("数据库连接错误:"+ex.Message);  
                return **false**;  
            }  
        }  
  
        **public** List<string> **GetAllDB**()  
        {  
            List<string> result = **new** List<string>();  
             **try**  
            {  
                //查出所有的数据库名  
                *conn*.**Open**();  
                *cmd*.CommandText = "SELECT SCHEMA\_NAME FROM information\_schema.SCHEMATA";  
                MySqlDataReader reader = *cmd*.**ExecuteReader**();  
                **while** (reader.**Read**())  
                {  
                    result.**Add**(reader.**GetString**(0));  
                }  
                reader.**Close**();  
            }  
            **catch** (Exception ex)  
            {  
                MessageBox.**Show**(ex.Message);  
            }  
            **finally**  
            {  
                 *conn*.**Close**();  
            }  
            return result;  
        }  
  
        **public** List<string> **GetAllTab**(string db)  
        {  
            List<string> result = **new** List<string>();  
            **try**  
            {  
                //查出所有的表名  
                *conn*.**Open**();  
                *conn*.**ChangeDatabase**(db);  
                *cmd*.CommandText = string.**Format**("SELECT TABLE\_NAME FROM INFORMATION\_SCHEMA.TABLES WHERE TABLE\_SCHEMA = '{0}'", db);  
                MySqlDataReader reader = *cmd*.**ExecuteReader**();  
                **while** (reader.**Read**())  
                {  
                    result.**Add**(reader.**GetString**(0));  
                }  
                reader.**Close**();  
            }  
            **catch** (Exception ex)  
            {  
                MessageBox.**Show**(ex.Message);  
            }  
            **finally**  
            {  
                 *conn*.**Close**();  
            }  
            return result;  
        }  
  
        **public** List<string> **GetAllCol**(string tb)  
        {  
            List<string> result = **new** List<string>();  
            **try**  
            {  
                //查出所有的列名  
                *conn*.**Open**();  
                *cmd*.CommandText = string.**Format**("select column\_name from information\_schema.columns where table\_name = '{0}'", tb);  
                MySqlDataReader reader = *cmd*.**ExecuteReader**();  
                **while** (reader.**Read**())  
                {  
                    result.**Add**(reader.**GetString**(0));  
                }  
                reader.**Close**();  
            }  
            **catch** (Exception ex)  
            {  
                MessageBox.**Show**(ex.Message);  
            }  
            **finally**  
            {  
                *conn*.**Close**();  
            }  
            return result;  
        }  
  
        **public** **bool** **ExportData**(StreamWriter mysw, List<string> headCols, string tab, WindowConnDB win)  
        {  
            **bool** result = **false**;  
               //保存到tab文件中，以tab键分割  
            **try**  
            {  
                //先查出总数量  
                *conn*.**Open**();  
                *cmd*.CommandText = "select count(\*) from " + tab;  
                MySqlDataReader sdr = *cmd*.**ExecuteReader**();  
                sdr.**Read**();  
                **int** allRow = sdr.**GetInt32**(0);  
                sdr.**Close**();  
  
                //再查询所有的数据  
                *cmd*.CommandText = "select " + string.**Join**(",", headCols) + " from " + tab;  
                sdr = *cmd*.**ExecuteReader**();  
  
                **int** row = 0;  
                **int** step = allRow > 100 ? allRow / 100 : 1;  
                object[] objs = **new** object[sdr.FieldCount];  
                  
                **while** (sdr.**Read**())  
                {  
                    sdr.**GetValues**(objs);  
                    **for** (**int** i = 0; i < objs.Length; i++) {  
                        string obj = objs[i].**ToString**();  
                        **if**(obj.**Contains**("\t"))  
                        {  
                            objs[i] = obj.**Replace**("\t", " ");  
                        }  
                    }  
                    string line = string.**Join**("\t", objs);  
                    mysw.**WriteLine**(line);  
                      
                    row++;  
                    **if** (row % step == 0)  
                    {  
                        win.*prograss*.Value = (**int**)(100.0 \* row / allRow);  
                        win.*labelMessage*.Text = "正在导出第" + row + "（总共" + allRow + "）";  
                        win.**DoEvents**();  
                    }  
                }  
   
                sdr.**Close**();  
                result = **true**;  
            }  
            **catch** (Exception ex)  
            {  
                MessageBox.**Show**(ex.Message);  
            }  
            **finally**  
            {  
                 *conn*.**Close**();  
                 mysw.**Close**();  
            }  
            return result;  
        }  
  
        #**endregion**  
    }  
}

## 连接mysql业务代码

**namespace** MemberTree  
{  
    /// <**summary**>  
    /// Description of ConnSqlserver.  
    /// <**/summary**>  
    **public** class ConnSqlserver : IConnDB  
    {  
        **private** SqlConnection *conn*;  
        **private** SqlCommand *cmd*;  
        **public** ConnSqlserver()  
        {  
        }  
          
        #**region** IConnDB implementation  
      
        **public** **bool** **Connect**(string IP, string usr, string pwd, string port)  
        {  
             SqlConnectionStringBuilder sqlStrBuilder = **new** SqlConnectionStringBuilder();  
            sqlStrBuilder.DataSource = IP; //server ip  
//            sqlStrBuilder.InitialCatalog = txtDBName.Text; //数据库名  
            sqlStrBuilder.UserID = usr;  //用户名  
            sqlStrBuilder.Password = pwd;  //密码  
            sqlStrBuilder.IntegratedSecurity = **false**; //false:用户名密码验证；true：windows身份验证  
            *conn* = **new** SqlConnection(sqlStrBuilder.ConnectionString);  
            **try**  
            {  
                *cmd* = *conn*.**CreateCommand**();  
                *conn*.**Open**();  
                *conn*.**Close**();  
                return **true**;  
            }  
            **catch** (Exception ex)  
            {  
                MessageBox.**Show**("数据库连接错误:" + ex.Message);  
                return **false**;  
            }  
        }  
      
        **public** List<string> **GetAllDB**()  
        {  
            List<string> result = **new** List<string>();  
            **try**  
            {  
                //查出所有的数据库名  
                *conn*.**Open**();  
                *cmd*.CommandText = "SELECT Name FROM Master..SysDatabases ORDER BY Name";  
                SqlDataReader reader = *cmd*.**ExecuteReader**();  
                **while** (reader.**Read**())  
                {  
                    result.**Add**(reader.**GetString**(0));  
                }  
                reader.**Close**();  
            }  
            **catch** (Exception ex)  
            {  
                MessageBox.**Show**(ex.Message);  
            }  
            **finally**  
            {  
                 *conn*.**Close**();  
            }  
            return result;  
        }  
      
        **public** List<string> **GetAllTab**(string db)  
        {  
            List<string> result = **new** List<string>();  
            **try**  
            {  
                //查出所有的表名  
                *conn*.**Open**();  
                *conn*.**ChangeDatabase**(db);  
                *cmd*.CommandText = string.**Format**("SELECT Name FROM SysObjects Where XType='U' ORDER BY Name");  
                SqlDataReader reader = *cmd*.**ExecuteReader**();  
                **while** (reader.**Read**())  
                {  
                    result.**Add**(reader.**GetString**(0));  
                }  
                reader.**Close**();  
            }  
            **catch** (Exception ex)  
            {  
                MessageBox.**Show**(ex.Message);  
            }  
            **finally**  
            {  
                 *conn*.**Close**();  
            }  
            return result;  
            return result;  
        }  
      
        **public** List<string> **GetAllCol**(string tb)  
        {  
            List<string> result = **new** List<string>();  
            **try**  
            {  
                //查出所有的列名  
                *conn*.**Open**();  
                *cmd*.CommandText = string.**Format**("select name from syscolumns where id=(select max(id) from sysobjects where xtype='u' and name='{0}')", tb);  
                SqlDataReader reader = *cmd*.**ExecuteReader**();  
                **while** (reader.**Read**())  
                {  
                    result.**Add**(reader.**GetString**(0));  
                }  
                reader.**Close**();  
            }  
            **catch** (Exception ex)  
            {  
                MessageBox.**Show**(ex.Message);  
            }  
            **finally**  
            {  
                 *conn*.**Close**();  
            }  
            return result;  
        }  
      
        **public** **bool** **ExportData**(StreamWriter mysw, List<string> headCols, string tab, WindowConnDB win)  
        {  
            **bool** result = **false**;  
            **int** allRow = 0;  
            **int** row = 0;  
               //保存到tab文件中，以tab键分割  
            **try**  
            {  
                //先查出总数量  
                *conn*.**Open**();  
                *cmd*.CommandText = "select count(\*) from " + tab;  
                SqlDataReader sdr = *cmd*.**ExecuteReader**();  
                sdr.**Read**();  
                allRow = sdr.**GetInt32**(0);  
                sdr.**Close**();  
  
                //再查询所有的数据  
                *cmd*.CommandText = "select " + string.**Join**(",", headCols) + " from " + tab;  
                sdr = *cmd*.**ExecuteReader**();  
  
                 
                **int** step = allRow > 100 ? allRow / 100 : 1;  
                object[] objs = **new** object[sdr.FieldCount];  
                  
                **while** (sdr.**Read**())  
                {  
                    sdr.**GetValues**(objs);  
                    **for** (**int** i = 0; i < objs.Length; i++) {  
                        string obj = objs[i].**ToString**();  
                        **if**(obj.**Contains**("\t"))  
                        {  
                            objs[i] = obj.**Replace**("\t", " ");  
                        }  
                    }  
                    string line = string.**Join**("\t", objs);  
                    mysw.**WriteLine**(line);  
                      
                    row++;  
                    **if** (row % step == 0)  
                    {  
                        win.*prograss*.Value = (**int**)(100.0 \* row / allRow);  
                        win.*labelMessage*.Text = "正在导出第" + row + "（总共" + allRow + "）";  
                        win.**DoEvents**();  
                    }  
                }  
   
                sdr.**Close**();  
                result = **true**;  
            }  
            **catch** (Exception ex)  
            {  
                MessageBox.**Show**(ex.Message);  
            }  
            **finally**  
            {  
                 win.*prograss*.Visibility = **Visibility**.*Hidden*;  
                 win.*labelMessage*.Text = "导出完成！一共导出" + row + "（总共" + allRow + "）";  
                 *conn*.**Close**();  
                 mysw.**Close**();  
            }  
            return result;  
        }  
      
        #**endregion**  
    }  
}

# 服务端导入计算树结构模块

## 服务端树节点定义代码

**public** class MyTreeNode  
    {          
        **public** string *SysId*;  
//        public string Name;  
        **public** string *TopId*;  
        **public** **int** *Level*;  
          
        //直接下属孩子节点数量  
        **public** **int** *SubCount*;  
        //所有的后代子孙节点数量  
        **public** **int** *ChildrenCount*;  
        //所有的后代子孙最深级别数  
        **public** **int** *ChildrenLevels*;  
          
        //子节点集合  
//        public List<MyTreeNode> ChildrenNodes = new List<MyTreeNode>();  
          
        **public** MyTreeNode()  
        {  
        }  
          
        **public** MyTreeNode(string sysId, string topId)  
        {  
            **this**.*SysId* = sysId;  
//            this.Name = name;  
            **this**.*TopId* = topId;  
            **this**.*Level* = 0;  
   
            **switch** (TextUtil.*enUpperLower*)   
            {  
                **case** **EnumUpperLower**.*Lower*:  
                    *SysId* = *SysId*.**ToLower**();  
                       *TopId* = *TopId*.**ToLower**();  
                    break;  
                **case** **EnumUpperLower**.*Upper*:  
                    *SysId* = *SysId*.**ToUpper**();  
                    *TopId* = *TopId*.**ToUpper**();  
                    break;  
                **default**:  
                    break;  
            }  
             
            **switch** (TextUtil.*enTrim*)  
            {  
                **case** **EnumTrim**.*All*:  
                    *SysId* = *SysId*.**Trim**();  
                    *TopId* = *TopId*.**Trim**();  
                    break;  
                **case** **EnumTrim**.*Start*:  
                    *SysId* = *SysId*.**TrimStart**();  
                    *TopId* = *TopId*.**TrimStart**();  
                    break;  
                **case** **EnumTrim**.*End*:  
                    *SysId* = *SysId*.**TrimEnd**();  
                    *TopId* = *TopId*.**TrimEnd**();  
                    break;  
                **default**:  
                    break;  
            }  
             
            **switch** (TextUtil.*enDBCSBC*)  
            {  
                **case** **EnumDBCSBC**.*DBC*:  
                    *SysId* = TextUtil.**SBCToDBC**(*SysId*);  
                    *TopId* = TextUtil.**SBCToDBC**(*TopId*);  
                    break;  
                **case** **EnumDBCSBC**.*SBC*:  
                    *SysId* = TextUtil.**DBCToSBC**(*SysId*);  
                    *TopId* = TextUtil.**DBCToSBC**(*TopId*);  
                    break;  
                **default**:  
                    break;  
            }  
        }  
    }

## 服务端计算树结构代码

**namespace** MemberTree  
{  
    /// <**summary**>  
    /// 保存树节点数据，计算树结构算法  
    /// <**/summary**>  
    **public** static class MyTrees  
    {  
        **internal** static IMyTreeDBA *treeDB*;  
        **public** static void **InitTreeDB**(**bool** isSqlite)  
        {  
            **if**(isSqlite)  
            {  
                *treeDB* = **new** MyTreeDBASqlite();  
            }  
            **else**  
            {  
                *treeDB* = **new** MyTreeDBAMysql();  
            }  
        }  
          
        **public** static void **SetDBName**(string dbName)  
        {  
             *treeDB*.**ConnectDB**(dbName);  
        }  
          
        #**region** 查找  
          
        **public** static MyTreeNode **FindParentNode**(string parentId)  
        {  
            **if**(parentId != "")  
            {  
                **if** (*allNodes*.**ContainsKey**(parentId))  
                {  
                    return *allNodes*[parentId];  
                }  
            }  
  
            return **null**;  
        }  
          
        #**endregion**  
          
         
        #**region** 计算构造树结构的具体数据结构和算法  
  
        **internal** static **int** *allNodeCount* = 0;  
          
        **internal** static **int** TreeNodeCount  
        {  
            get  
            {  
                **int** count = *TreeRootNodes*.Count;  
                **foreach** (MyTreeNode node **in** *TreeRootNodes*)   
                {  
                    count += node.*ChildrenCount*;  
                }  
                return count;  
            }  
        }  
           
        **private** static Dictionary<string, MyTreeNode> *allNodes* = **new** Dictionary<string, MyTreeNode>();  
          
        **internal** static List<MyTreeNode> *TreeRootNodes* = **new** List<MyTreeNode>();  
  
        **internal** static List<MyTreeNode> *NoParentNodes* = **new** List<MyTreeNode>();  
          
        **internal** static List<MyTreeNode> *IdConflictNodes* = **new** List<MyTreeNode>();  
          
        **internal** static Dictionary<string, MyTreeNode> *RingNodes* = **new** Dictionary<string, MyTreeNode>();  
          
        #**endregion**  
          
        #**region** 读取数据(csv或tab)，计算树结构，写入数据库  
  
        **public** static void **OpenDBFile**(string filepath, string separator, **bool** checkHead)  
        {  
            List<string> dbs = *treeDB*.**GetDatasetNames**();  
            FileInfo fileInfo = **new** FileInfo(filepath);  
            string dbName = fileInfo.Name.**Replace**(fileInfo.Extension, "").**ToLower**();  
            **if**(dbs.**Contains**(dbName))  
            {  
                **MessageBoxResult** msgResult = MessageBox.**Show**("该数据文件已经存在，是否重新计算并覆盖旧文件？","提示",**MessageBoxButton**.*OKCancel*);  
                **if**(msgResult == **MessageBoxResult**.*OK*)  
                {  
                    *treeDB*.**DeleteDB**(dbName);  
                }  
                **else**  
                {  
                    return;  
                }  
            }  
              
            TimingUtil.**StartTiming**();  
             
            //读取文件数据并构造树节点-------------------------------------  
            **bool** readSuccess = **ReadLine2Node**(filepath, separator, checkHead);  
            **if**(readSuccess)  
            {  
                //将树节点计算并构造树结构-------------------------------------  
                **ConstructTree**();  
                //写入数据库-------------------------------------------------  
                **Write2DB**(filepath, dbName, separator);  
                  
                WindowAdmin.*notify*.**SetStatusMessage**(TimingUtil.**EndTiming**());  
            }  
            **else**  
            {  
                WindowAdmin.*notify*.**SetProcessBarVisible**(**false**);  
            }  
            *allNodes*.**Clear**();  
            *NoParentNodes*.**Clear**();  
            *IdConflictNodes*.**Clear**();  
            *TreeRootNodes*.**Clear**();  
            *RingNodes*.**Clear**();  
        }  
          
        //读取文件数据并构造树节点  
        **private** static **bool** **ReadLine2Node**(string filepath, string separator, **bool** confirm)  
        {  
            Encoding encoding = TextUtil.**GetFileEncodeType**(filepath);  
            StreamReader mysr = **new** StreamReader(filepath, encoding);  
            **int** row = 0;  
            **try**  
            {  
                WindowAdmin.*notify*.**SetProcessBarVisible**(**true**);  
                WindowAdmin.*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** (!DBUtil.**CheckHead**(firstLine, separator, confirm)) //检查表头  
                {  
                    string errMsg = "文件格式不正确，最少必须包含三列，前三列为“会员id,上级会员id,会员姓名”且顺序固定，请重新选择正确的文件！";  
                    MessageBox.**Show**(errMsg);  
                    WindowAdmin.*notify*.**SetStatusMessage**(errMsg);  
                    mysr.**Close**();  
                    return **false**;  
                }  
                **while**(!mysr.EndOfStream)  
                {  
                    string line = mysr.**ReadLine**();  
                    string[] aryline = line.**Split**(**new** String[] { separator }, **StringSplitOptions**.*None*);  
                    **if**(!DBUtil.**CheckLen**(aryline))  
                    {  
                        //发现出错的数据  
                        string errMsg = "该csv数据文件中包含错误数据，请先对csv文件进行检查校准！";  
                        MessageBox.**Show**(errMsg);  
                        WindowAdmin.*notify*.**SetStatusMessage**(errMsg);  
                        mysr.**Close**();  
                        return **false**;  
                    }  
  
                    MyTreeNode myNode = **new** MyTreeNode(aryline[0], aryline[1]);  
                      
                    **if**(*allNodes*.**ContainsKey**(myNode.*SysId*)) //ID有重复的节点  
                    {  
                        *IdConflictNodes*.**Add**(myNode);  
                    }  
                    **else**  
                    {  
                        *allNodes*.**Add**(myNode.*SysId*, myNode);  
                    }  
  
                    row++;  
                    **if** (row % 1000 == 0)  
                    {  
                        WindowAdmin.*notify*.**SetStatusMessage**("【第一步：正在读取第" + row + "个节点】——>【第二步：构造树结构】——>【第三步：写入数据库】");  
                    }  
                }  
                  
                *allNodeCount* = row;  
            }  
            **catch** (Exception ex)  
            {  
                WindowAdmin.*notify*.**SetProcessBarVisible**(**false**);  
                WindowAdmin.*notify*.**SetStatusMessage**("文件读取出错！+\n" + ex.Message);  
                mysr.**Close**();  
                return **false**;  
            }  
              
            mysr.**Close**();  
            return **true**;  
        }  
  
        //将树节点计算并构造树结构  
        **private** static void **ConstructTree**()  
        {  
            **int** row = 0;  
            **try**  
            {  
                **foreach** (MyTreeNode node **in** *allNodes*.Values)  
                {  
                    //将节点加入树中合适的位置去  
                    **ConstructTree**(node);  
  
                    row++;  
                    **if** (row % 1000 == 0)  
                    {  
                        WindowAdmin.*notify*.**SetProcessBarValue**(row \* 100.0 / *allNodeCount*);  
                        WindowAdmin.*notify*.**SetStatusMessage**("【第一步：读取数据完成】——>【第二步：正在构造树结构" + row + "/" + *allNodeCount* +"】——>【第三步：写入数据库】");  
                    }  
                }  
                **foreach** (MyTreeNode node **in** *IdConflictNodes*)  
                {  
                    //将节点加入树中合适的位置去  
                    **ConstructTree**(node);  
                }  
                  
                #**region** 找出所有构成树的节点  
                **foreach** (MyTreeNode node **in** *NoParentNodes*)   
                {  
                    **if**(node.*ChildrenCount* > 0)  
                    {  
                        *TreeRootNodes*.**Add**(node);  
                    }  
                }  
                **foreach** (MyTreeNode node **in** *TreeRootNodes*)   
                {  
                    *NoParentNodes*.**Remove**(node);  
                }  
                #**endregion**  
            }  
            **catch** (Exception ex)  
            {  
                WindowAdmin.*notify*.**SetStatusMessage**("发生异常：" + ex.Message + "在第" + row + "行!");  
            }  
        }  
  
        //构建树（将节点加进树结构中合适的位置）  
        **private** static void **ConstructTree**(MyTreeNode myNode)  
        {  
            //是否包含父节点  
            MyTreeNode parentNode = **FindParentNode**(myNode.*TopId*);  
            **if** (parentNode != **null**)  
            {  
                **ChildrenCountInc**(myNode);//所有父节点的子孙节点加1  
//                parentNode.ChildrenNodes.Add(myNode);  
                parentNode.*SubCount*++;  
            }  
            **else**  
            {  
                //父节点不存在  
                *NoParentNodes*.**Add**(myNode);  
            }  
        }  
  
        //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）  
        **private** static void **ChildrenCountInc**(MyTreeNode node)  
        {  
            //帮助判断是否存在闭环  
            Dictionary<string, MyTreeNode> parentList = **new** Dictionary<string, MyTreeNode>();  
            //parentList.Add(node.SysId, node);  
  
            **ushort** deepLevel = 0; //深度（父节点到子节点之间的层级数之差）  
            MyTreeNode parent = **FindParentNode**(node.*TopId*);  
            **while** (parent != **null**)  
            {  
                //判断是否构成闭环  
                **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.*ChildrenCount*++;  
                deepLevel++;  
                **if**(parent.*ChildrenLevels* < deepLevel)  
                {  
                    parent.*ChildrenLevels* = deepLevel;  
                }  
                  
                //继续循环遍历查找父节点的父节点，直到根节点  
                parent = **FindParentNode**(parent.*TopId*);  
            }  
              
            **if**(node.*Level* == 0)  
            {  
                node.*Level* = (**ushort**)(deepLevel + 1);  
            }  
        }  
              
        **private** static void **Write2DB**(string filepath, string dbName, string separator)  
        {  
            **if**(*treeDB*.**CreateDB**(dbName))  
            {  
                **int** row = 0;  
                Encoding encoding = TextUtil.**GetFileEncodeType**(filepath);  
                StreamReader mysr = **new** StreamReader(filepath, encoding);  
                *treeDB*.**BeginInsert**();  
//                try  
//                {  
                    string line = mysr.**ReadLine**();  
                    **while**(!mysr.EndOfStream)  
                    {  
                        line = mysr.**ReadLine**();  
                        string[] aryline = line.**Split**(**new** String[] { separator }, **StringSplitOptions**.*None*);  
                        *treeDB*.**InsertNodes**(MyTrees.*allNodes*[aryline[0]] ,aryline);  
  
                        row++;  
                        **if** (row % 100 == 0)  
                        {  
                            *treeDB*.**TransCommit**(**false**);  
                            WindowAdmin.*notify*.**SetProcessBarValue**(row \* 100.0 / *allNodeCount*);  
                            WindowAdmin.*notify*.**SetStatusMessage**("【第一步：读取数据完成】——>【第二步：构造树结构完成】——>【第三步：正在写入数据库" + row + "/" + *allNodeCount* +"】");  
                        }  
                    }  
                    *treeDB*.**TransCommit**(**true**);  
                      
                    WindowAdmin.*notify*.**SetStatusMessage**("【第一步：读取数据完成】——>【第二步：构造树结构完成】——>【第三步：正在创建数据库索引。。。】");   
                    *treeDB*.**CreateIndex**();   
  
                    WindowAdmin.*notify*.**SetStatusMessage**("【第一步：读取数据完成】——>【第二步：构造树结构完成】——>【第三步：正在写入数据概要信息。。。】");   
                    *treeDB*.**CreateProfile**();   
                      
                    WindowAdmin.*notify*.**SetProcessBarVisible**(**false**);  
                    WindowAdmin.*notify*.**SetStatusMessage**("计算完成！");  
//                }  
//                catch (Exception ex)  
//                {  
//                    WindowAdmin.notify.SetProcessBarVisible(false);  
//                    WindowAdmin.notify.SetStatusMessage("文件读取出错！+\n" + ex.Message);  
//                }  
                mysr.**Close**();  
            }  
        }  
          
        #**endregion**  
    }  
}

## 树结构计算完写入Sqlite代码

**namespace** MemberTree  
{  
    /// <**summary**>  
    /// Description of DBHelper.  
    /// <**/summary**>  
    **public** class MyTreeDBASqlite : MyTreeDBSqlite, IMyTreeDBA  
    {  
        **public** **bool** **CreateDB**(string dbName)  
        {  
            string dbFile = MemData.*MemDataSqlite* + "/" + dbName + ".db";  
  
            SQLiteConnection.**CreateFile**(dbFile);  
            **ConnectDB**(dbName);  
            **OpenDB**();  
              
            //创建profile表  
            *cmd*.CommandText = "create table tree\_profile("   
                + "k varchar(16),"  
                + "v varchar(32))";  
            *cmd*.**ExecuteNonQuery**();  
              
            //创建calc表  
            *cmd*.CommandText = "create table tree\_calc"   
                + DBUtil.**GetCreateTableSql**();  
            *cmd*.**ExecuteNonQuery**();  
              
            **CloseDB**();  
              
            return **true**;  
        }  
          
        **public** void **DeleteDB**(string dbName)  
        {  
            **if**(*cmd* != **null**)  
            {  
                *cmd*.**Dispose**();  
            }  
            **if**(*conn* != **null**)  
            {  
                *conn*.**Dispose**();    
            }  
            GC.**Collect**();    
            GC.**WaitForPendingFinalizers**();    
            string dbFile = MemData.*MemDataSqlite* + "/" + dbName + ".db";  
            File.**Delete**(dbFile);  
        }  
  
        //插入数据  
        **public** void **BeginInsert**()  
        {  
            **OpenDB**();  
            *trans* = *conn*.**BeginTransaction**();  
              
            //创建插入sql语句  
            StringBuilder sb = **new** StringBuilder();  
            sb.**Append**("insert into tree\_calc values(");  
            //字段总个数=可选字段+必选字段(7个)  
            **int** colCount = DBUtil.*OptCols*.Count + 7;  
            sb.**Append**("@0");  
            **for** (**int** j = 1; j < colCount; j++)   
            {  
                sb.**Append**(",@");  
                sb.**Append**(j);  
            }  
            sb.**Append**(")");  
            *cmd*.CommandText = sb.**ToString**();  
        }  
  
        **public** void **InsertNodes**(MyTreeNode node, string[] optCols)  
        {  
            *cmd*.Parameters.**AddRange**(**new**[] {  
            **new** SQLiteParameter("@0", node.*SysId*),  
            **new** SQLiteParameter("@1", node.*TopId*),  
            **new** SQLiteParameter("@2", optCols[2]),  
            **new** SQLiteParameter("@3", node.*Level*),  
            **new** SQLiteParameter("@4", node.*ChildrenLevels*),  
            **new** SQLiteParameter("@5", node.*SubCount*),  
            **new** SQLiteParameter("@6", node.*ChildrenCount*)});  
              
            **for** (**int** i = 3; i < optCols.Length; i++)   
            {  
                *cmd*.Parameters.**Add**(**new** SQLiteParameter("@"+(i+4), optCols[i]));  
            }  
              
            *cmd*.**ExecuteNonQuery**();  
        }  
          
        **public** void **TransCommit**(**bool** end)  
        {  
            **if**(end)  
            {  
                *trans*.**Commit**();  
                **CloseDB**();  
            }  
        }  
          
        **public** void **CreateIndex**()   
        {   
            **OpenDB**();   
            *cmd*.CommandText = "create index idxsysid on tree\_calc(sysid)";  
            *cmd*.**ExecuteNonQuery**();   
            *cmd*.CommandText = "create index idxtopid on tree\_calc(topid)";   
            *cmd*.**ExecuteNonQuery**();   
            **CloseDB**();   
        }            
  
        **public** void **CreateProfile**()   
        {   
            **OpenDB**();   
            *trans* = *conn*.**BeginTransaction**();   
            *cmd*.CommandText = "insert into tree\_profile values(@k,@v)";   
              
            //全部节点总数量   
            *cmd*.Parameters.**AddWithValue**("@k","AllNodeCount");   
            *cmd*.Parameters.**AddWithValue**("@v", MyTrees.*allNodeCount*);   
            *cmd*.**ExecuteNonQuery**();   
              
            //构成树的节点数量   
            *cmd*.Parameters.**AddWithValue**("@k","TreeNodeCount");   
            *cmd*.Parameters.**AddWithValue**("@v", MyTrees.TreeNodeCount);   
            *cmd*.**ExecuteNonQuery**();   
  
            //表格可选列标题  
            **foreach** (string head **in** DBUtil.*OptCols*) {   
                *cmd*.Parameters.**AddWithValue**("@k","TableOptCol");   
                *cmd*.Parameters.**AddWithValue**("@v", head);   
                *cmd*.**ExecuteNonQuery**();   
            }   
  
            //形成树的根节点   
            **foreach** (MyTreeNode node **in** MyTrees.*TreeRootNodes*) {   
                *cmd*.Parameters.**AddWithValue**("@k","Tree");   
                *cmd*.Parameters.**AddWithValue**("@v", node.*SysId*);   
                *cmd*.**ExecuteNonQuery**();   
            }  
   
            //单个叶子节点   
            **foreach** (MyTreeNode node **in** MyTrees.*NoParentNodes*) {   
                *cmd*.Parameters.**AddWithValue**("@k","Leaf");   
                *cmd*.Parameters.**AddWithValue**("@v", node.*SysId*);   
                *cmd*.**ExecuteNonQuery**();   
            }   
  
            //ID有冲突的节点   
            **foreach** (MyTreeNode node **in** MyTrees.*IdConflictNodes*) {   
                *cmd*.Parameters.**AddWithValue**("@k","Conflict");   
                *cmd*.Parameters.**AddWithValue**("@v", node.*SysId*);   
                *cmd*.**ExecuteNonQuery**();   
            }   
  
            //形成闭环的节点   
            **foreach** (string node **in** MyTrees.*RingNodes*.Keys) {   
                *cmd*.Parameters.**AddWithValue**("@k","Ring");   
                *cmd*.Parameters.**AddWithValue**("@v", node);   
                *cmd*.**ExecuteNonQuery**();   
            }   
  
            *trans*.**Commit**();   
            **CloseDB**();  
        }   
    }  
}

## 树结构计算完写入Mysql代码

**namespace** MemberTree  
{  
    /// <**summary**>  
    /// Description of DBHelper.  
    /// <**/summary**>  
    **public** class MyTreeDBAMysql : MyTreeDBMysql, IMyTreeDBA  
    {  
        **public** **bool** **CreateDB**(string dbName)  
        {  
            *dbNameProfile* = dbName + "\_profile";  
            *dbNameCalc* = dbName + "\_calc";  
            **OpenDB**();  
              
            //创建profile表  
            *cmd*.CommandText = "create table "   
                + *dbNameProfile*   
                + "(k varchar(16),"   
                + "v varchar(32))engine=MyISAM";  
            *cmd*.**ExecuteNonQuery**();  
              
            //创建calc表  
            *cmd*.CommandText = "create table "   
                + *dbNameCalc*   
                + DBUtil.**GetCreateTableSql**()  
                + "engine=MyISAM";  
            *cmd*.**ExecuteNonQuery**();  
              
            **CloseDB**();  
      
            return **true**;  
        }  
          
        **public** void **DeleteDB**(string dbName)  
        {  
            **OpenDB**();  
            *cmd*.CommandText = "drop table if exists "+dbName+"\_calc";  
            *cmd*.**ExecuteNonQuery**();  
            *cmd*.CommandText = "drop table if exists "+dbName+"\_profile";  
            *cmd*.**ExecuteNonQuery**();  
            *cmd*.CommandText = "delete from tree\_userprivilege where data\_name = '"+ dbName +"'";  
            *cmd*.**ExecuteNonQuery**();  
            **CloseDB**();  
        }  
          
        //插入数据  
        **public** void **BeginInsert**()  
        {  
            **OpenDB**();  
              
            **ConstructInsertSql**(100);  
        }  
          
        //创建插入sql语句  
        **private** void **ConstructInsertSql**(**int** count)  
        {  
            StringBuilder sb = **new** StringBuilder();  
            sb.**Append**("insert into "+*dbNameCalc*+" values");  
              
            //每次提交100个数据  
            **for** (**int** i = 0; i < count; i++)   
            {  
                //字段总个数=可选字段+必选字段(7个)  
                **int** colCount = DBUtil.*OptCols*.Count + 7;  
                sb.**Append**("(@0");  
                sb.**Append**("\_");  
                sb.**Append**(i);  
                **for** (**int** j = 1; j < colCount; j++) {  
                    sb.**Append**(",@");  
                    sb.**Append**(j);  
                    sb.**Append**("\_");  
                    sb.**Append**(i);  
                }  
                sb.**Append**("),");  
            }  
            sb.**Remove**(sb.Length-1, 1);  
            *cmd*.CommandText = sb.**ToString**();  
        }  
  
        **int** *n* = 0;  
        **public** void **InsertNodes**(MyTreeNode node, string[] optCols)  
        {  
            *cmd*.Parameters.**AddRange**(**new**[] {  
            **new** MySqlParameter("@0\_"+*n*, node.*SysId*),  
            **new** MySqlParameter("@1\_"+*n*, node.*TopId*),  
            **new** MySqlParameter("@2\_"+*n*, optCols[2]),  
            **new** MySqlParameter("@3\_"+*n*, node.*Level*),  
            **new** MySqlParameter("@4\_"+*n*, node.*ChildrenLevels*),  
            **new** MySqlParameter("@5\_"+*n*, node.*SubCount*),  
            **new** MySqlParameter("@6\_"+*n*, node.*ChildrenCount*)});  
              
            **for** (**int** i = 3; i < optCols.Length; i++)   
            {  
                *cmd*.Parameters.**Add**(**new** MySqlParameter("@"+(i+4)+"\_"+*n*, optCols[i]));  
            }  
            *n*++;  
        }  
          
        **public** void **TransCommit**(**bool** end)  
        {  
            **if**(end)  
            {  
                **ConstructInsertSql**(*n*);  
            }  
            *cmd*.**ExecuteNonQuery**();  
            *cmd*.Parameters.**Clear**();  
            *n* = 0;  
            **if**(end)  
            {  
                **CloseDB**();  
            }  
        }  
          
        **public** void **CreateIndex**()  
        {  
            **OpenDB**();  
            *cmd*.CommandText = "create index idxsysid1 on "+*dbNameCalc*+"(sysid)";  
            *cmd*.**ExecuteNonQuery**();  
            *cmd*.CommandText = "create index idxtopid1 on "+*dbNameCalc*+"(topid)";  
            *cmd*.**ExecuteNonQuery**();  
            **CloseDB**();  
        }  
          
        **public** void **CreateProfile**()  
        {  
            StringBuilder sb = **new** StringBuilder();  
            **OpenDB**();  
            //全部节点总数量  
            *cmd*.CommandText = "insert into "+*dbNameProfile*+" values(@k,@v)";  
            *cmd*.Parameters.**AddWithValue**("@k","AllNodeCount");  
            *cmd*.Parameters.**AddWithValue**("@v", MyTrees.*allNodeCount*);  
            *cmd*.**ExecuteNonQuery**();  
            //构成树的节点数量  
            *cmd*.Parameters.**Clear**();  
            *cmd*.Parameters.**AddWithValue**("@k","TreeNodeCount");  
            *cmd*.Parameters.**AddWithValue**("@v", MyTrees.TreeNodeCount);  
            *cmd*.**ExecuteNonQuery**();  
            //表格可选列标题  
            **InsertProfile**(sb, "TableOptCol", DBUtil.*OptCols*);  
            //形成树的根节点  
            **InsertProfile**(sb, "Tree", MyTrees.*TreeRootNodes*);  
            //单个叶子节点  
            **InsertProfile**(sb, "Leaf", MyTrees.*NoParentNodes*);  
            //ID有冲突的节点  
            **InsertProfile**(sb, "Conflict", MyTrees.*IdConflictNodes*);  
            //形成闭环的节点  
            List<MyTreeNode> nodeList = **new** List<MyTreeNode>(MyTrees.*RingNodes*.Values);  
            **InsertProfile**(sb, "Ring", nodeList);  
            **CloseDB**();  
        }  
          
        **private** void **InsertProfile**(StringBuilder sb, string nodeType, List<MyTreeNode> nodes)  
        {  
            List<string> nodestr = **new** List<string>();  
            **foreach** (MyTreeNode node **in** nodes) {  
                nodestr.**Add**(node.*SysId*);  
            }  
            **InsertProfile**(sb, nodeType, nodestr);  
        }  
          
        **private** void **InsertProfile**(StringBuilder sb, string nodeType, List<string> nodes)  
        {  
            **if**(nodes.Count > 0)  
            {  
                //如果数量太大，则分阶段插入  
                **int** stepCount = nodes.Count;  
                **int** step = nodes.Count / 1000;  
                **int** mod = nodes.Count % 1000;  
                  
                //大于1000的部分，每组1000进行分组插入  
                **for** (**int** n = 0; n < step; n++)   
                {  
                    sb.**Clear**();  
                    *cmd*.Parameters.**Clear**();  
                    sb.**Append**("insert into ");  
                    sb.**Append**(*dbNameProfile*);  
                    sb.**Append**(" values(@k0,@v0)");  
                    *cmd*.Parameters.**AddWithValue**("@k0",nodeType);  
                    *cmd*.Parameters.**AddWithValue**("@v0", nodes[0]);  
                    **for** (**int** i = 1; i < 1000; i++)   
                    {  
                        sb.**Append**(",(@k" + i);  
                        sb.**Append**(",@v" + i + ")");  
                        *cmd*.Parameters.**AddWithValue**("@k" + i, nodeType);  
                        *cmd*.Parameters.**AddWithValue**("@v" + i, nodes[i+n\*1000]);  
                    }  
                    *cmd*.CommandText = sb.**ToString**();  
                    *cmd*.**ExecuteNonQuery**();  
                }  
                  
                //不足1000的部分  
                sb.**Clear**();  
                *cmd*.Parameters.**Clear**();  
                sb.**Append**("insert into ");  
                sb.**Append**(*dbNameProfile*);  
                sb.**Append**(" values(@k0,@v0)");  
                *cmd*.Parameters.**AddWithValue**("@k0",nodeType);  
                *cmd*.Parameters.**AddWithValue**("@v0", nodes[0]);  
                **for** (**int** i = 1; i < mod; i++)   
                {  
                    sb.**Append**(",(@k" + i);  
                    sb.**Append**(",@v" + i + ")");  
                    *cmd*.Parameters.**AddWithValue**("@k" + i, nodeType);  
                    *cmd*.Parameters.**AddWithValue**("@v" + i, nodes[i+step\*1000]);  
                }  
                *cmd*.CommandText = sb.**ToString**();  
                *cmd*.**ExecuteNonQuery**();  
            }  
        }  
    }  
}

## 检查csv数据错误代码

**namespace** MemberTree  
{  
    /// <**summary**>  
    /// Interaction logic for ErrLinesConfirm.xaml  
    /// <**/summary**>  
    **public** partial class CsvErrCheck : Window  
    {  
        **public** CsvErrCheck()  
        {  
            **InitializeComponent**();  
        }  
          
        **int** *allCount*, *errCount*;  
        string *filepath*;  
        **int** *allColsCount*;  
          
        **private** **bool** **CheckHead**(string headline)  
        {  
            string[] heads = headline.**Split**(**new** **char**[] { ',' });  
            **if**(heads.Length > 2)  
            {  
                *btnName*.Content = "会员姓名：" + heads[2];  
                **if**(heads.Length > 1)  
                {  
                    *btnTopid*.Content = "上级会员ID：" + heads[1];  
                    **if**(heads.Length > 0)  
                    {  
                        *btnSysid*.Content = "会员ID：" + heads[0];  
                    }  
                }  
            }  
              
            **if**(heads.Length<3)  
            {  
                **SetStatusMessage**("文件格式不正确，最少必须包含三列，前三列为“会员id,上级会员id,会员姓名”且顺序固定，请重新选择正确的文件！");  
                return **false**;  
            }  
            **else**  
            {  
                *allColsCount* = heads.Length;  
                *grpOptCols*.Header = "可选列（共" + (*allColsCount*-3) + "列）";  
                **for** (**int** i = 3; i < heads.Length; i++) {  
                    Button btn = **new** Button();  
                    btn.Content = heads[i];  
                    *mainPanel*.Children.**Add**(btn);  
                }  
                return **true**;  
            }  
        }  
          
        //检测  
        **private** void **btnCheck\_Click**(object sender, RoutedEventArgs e)  
        {  
            OpenFileDialog openfileDlg = **new** OpenFileDialog();  
            openfileDlg.Title = "打开要作为会员树数据源的文件";  
            openfileDlg.Filter = "CSV文件|\*.csv";  
            **if** (openfileDlg.**ShowDialog**() == **true**)  
            {  
                *allCount* = 0;  
                *errCount* = 0;  
                *txtErrLines*.Text = "";  
                *filepath* = openfileDlg.FileName;  
                Encoding encoding = TextUtil.**GetFileEncodeType**(*filepath*);  
                StreamReader mysr = **new** StreamReader(*filepath*, encoding);  
                *filepath* = *filepath*.**Replace**(".csv","\_1.csv");  
                StreamWriter mysw = **new** StreamWriter(*filepath*, **false**, Encoding.UTF8);  
  
                **try**  
                {  
                    **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** (**CheckHead**(firstLine))  
                    {  
                        mysw.**WriteLine**(firstLine);  
                          
                        **while**(!mysr.EndOfStream)  
                        {  
                            string line = mysr.**ReadLine**();  
                            **Line2TreeNode**(line, mysw);  
                        }  
                          
                        **if**(*txtErrLines*.Text == "")  
                        {  
                            *btnSave*.IsEnabled = **false**;  
                            **SetStatusMessage**("检查完成，正确数据行数：" + *allCount* + "，出错数据行数："+ *errCount*);  
                            MessageBox.**Show**("恭喜，该文件没有错误，可以直接导入计算！");  
                        }  
                          **else**  
                          {  
                              *btnSave*.IsEnabled = **true**;  
                              **SetStatusMessage**("检查完成，正确数据行数：" + *allCount* + "，出错数据行数："+ *errCount* +"，错误数据都存储在下面的文本框中。");  
                              MessageBox.**Show**("该文件出现了多处错误，错误数据都存储在下面的文本框中，请将这些错误人工校准之后重新填回文本框中，点击“修改保存到文件”按钮");  
                          }  
                    }  
                }  
                **catch** (Exception ex)  
                {  
                    **SetStatusMessage**("文件读取出错！+\n" + ex.Message);  
                    return;  
                }  
                **finally**  
                {  
                    mysr.**Close**();  
                    mysw.**Close**();  
                }  
            }  
        }  
          
        **private** void **Line2TreeNode**(string line, StreamWriter mysw)  
        {  
            string[] aryline = line.**Split**(**new** **char**[] { ',' });  
            **if**(aryline.Length == *allColsCount*)  
            {  
                mysw.**WriteLine**(line);  
            }  
            **else**  
            {  
                **if**(line != "")  
                {  
                    **AddErrLine**(line);  
                }  
                return;  
            }  
          
              
              
//            if(myNode.SysId == "") //信息不完整（ID为空）的节点  
//            {  
//                IdNullNodes.Add(myNode);  
//            }  
//               else if(allNodes.ContainsKey(myNode.SysId)) //ID有重复的节点  
//            {  
//                IdConflictNodes.Add(myNode);  
//            }  
//            else  
//            {  
//                allNodes.Add(myNode.SysId, myNode);  
//            }  
  
            *allCount*++;  
            **if** (*allCount* % 1000 == 0)  
            {  
                **SetStatusMessage**("正在检查数据，正确数据行数：" + *allCount* + "，出错数据行数："+ *errCount*);  
            }  
        }  
          
        //保存  
        **private** void **btnSave\_Click**(object sender, RoutedEventArgs e)  
        {  
            StreamWriter mysw = **new** StreamWriter(*filepath*, **true**, Encoding.UTF8);  
            string[] lines = *txtErrLines*.Text.**Split**(**new** String[]{Environment.NewLine}, **StringSplitOptions**.*RemoveEmptyEntries*);  
            *txtErrLines*.Text = "";  
            *errCount* = 0;  
            **foreach** (string line **in** lines)   
            {  
                **Line2TreeNode**(line, mysw);  
            }  
            mysw.**Close**();  
              
            **if**(*txtErrLines*.Text == "")  
            {  
                *btnSave*.IsEnabled = **false**;  
                **SetStatusMessage**("保存完成，最终合适数据行数：" + *allCount*);  
                MessageBox.**Show**("恭喜，保存完成！");  
            }  
              **else**  
              {  
                  **SetStatusMessage**("合并保存完成，正确数据行数：" + *allCount* + "，出错数据行数："+ *errCount* +"，错误数据都存储在下面的文本框中。");  
                  MessageBox.**Show**("您处理过的数据仍然有多处错误，错误数据都存储在下面的文本框中，请将这些错误人工校准之后重新填回文本框中，点击“修改保存到文件”按钮");  
              }  
        }  
          
        #**region**  
          
        //设置状态栏提示文本  
        **private** delegate void ShowTextDelegate(string message);  
        **private** ShowTextDelegate *setStatusDelegate* = **null**;  
        **private** ShowTextDelegate *addErrDelegate* = **null**;  
        **public** void **SetStatusMessage**(string message)  
        {  
            **if** (*setStatusDelegate* == **null**)  
            {  
                *setStatusDelegate* = **new** ShowTextDelegate(**SetStatusMessageImp**);  
            }  
            **this**.Dispatcher.**Invoke**(*setStatusDelegate*, message);  
            **DoEvents**();  
        }  
        **private** void **SetStatusMessageImp**(string message)  
        {  
            **this**.*txtHeader*.Text = message;  
        }  
          
        **public** void **AddErrLine**(string line)  
        {  
            **if** (*addErrDelegate* == **null**)  
            {  
                *addErrDelegate* = **new** ShowTextDelegate(**AddErrLineImp**);  
            }  
            **this**.Dispatcher.**Invoke**(*addErrDelegate*, line);  
            **DoEvents**();  
        }  
        **private** void **AddErrLineImp**(string line)  
        {  
            **this**.*txtErrLines*.**AppendText**(line);  
            **this**.*txtErrLines*.**AppendText**(Environment.NewLine);  
            *errCount*++;  
        }  
  
        **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**  
    }  
}

# 客户端读取树结构模块

## 客户端树节点定义代码

**namespace** MemberTree  
{  
    **public** class MyTreeNode  
    {  
        **public** string SysId { get; set; }  
        **public** string TopId { get; set; }  
        **public** string Name { get; set; }  
        **public** **int** Level { get; set; }  
        **public** **int** ChildrenLevels { get; set; }  
        **public** **int** ChildrenCount { get; set; }  
        **public** **int** ChildrenCountAll { get; set; }  
          
        **public** List<string> *OtherProps* = **new** List<string>();  
          
        **public** MyTreeNode()  
        {  
        }  
  
        **public** override string **ToString**()  
        {  
            return string.**Format**("{0}({1}),父ID{2},所在层级{3},下级层数{4},直接下级{5},下级会员总数{6}",  
                                 Name, SysId, TopId, Level, ChildrenLevels, ChildrenCount, ChildrenCountAll);  
        }  
    }  
}

## 客户端管理树结构代码

**namespace** MemberTree  
{  
    /// <**summary**>  
    /// 保存树节点数据，计算树结构算法  
    /// <**/summary**>  
    **public** static class MyTrees  
    {  
        **internal** static IMyTreeDBV *treeDB*;  
        **public** static void **InitTreeDB**(**bool** isSqlite)  
        {  
            **if**(isSqlite)  
            {  
                *treeDB* = **new** MyTreeDBVSqlite();  
            }  
            **else**  
            {  
                *treeDB* = **new** MyTreeDBVMysql();  
            }  
        }  
          
        **public** static void **SetDBName**(string dbName)  
        {  
             *treeDB*.**ConnectDB**(dbName);  
        }  
          
        #**region** 打开关闭连接  
        **private** static **int** *openTimes* = 0;  
        **public** static void **OpenDB**()  
        {  
            *treeDB*.**OpenDB**();  
            *openTimes*++;  
        }  
        **public** static void **CloseDB**()  
        {  
            *openTimes*--;  
            **if**(*openTimes* == 0)  
            {  
                *treeDB*.**CloseDB**();  
            }  
        }  
        #**endregion**  
          
        #**region** 查询特定不同类型的节点  
          
        **internal** static **int** **GetTreeRootNodesCount**()  
        {  
            MyTrees.**OpenDB**();  
            string sql = "select count(\*) from "  
                + *treeDB*.TableName + "\_calc where sysid in (select v from "   
                + *treeDB*.TableName + "\_profile where k='Tree')";  
            **int** count = *treeDB*.**SearchCount**(sql);  
            MyTrees.**CloseDB**();  
            return count;  
        }  
        **internal** static List<MyTreeNode> **GetTreeRootNodes**(**int** pageNo, **int** pageSize)  
        {  
            MyTrees.**OpenDB**();  
            string sql = "select sysid,topid,name,level,sublevel,subcount,subcountall from "  
                + *treeDB*.TableName + "\_calc where sysid in (select v from "   
                + *treeDB*.TableName + "\_profile where k='Tree') order by subcountall desc limit "  
                + (pageNo \* pageSize) + ", " + pageSize;  
            List<MyTreeNode> treeRootNodes = *treeDB*.**SearchNode**(sql);  
            MyTrees.**CloseDB**();  
            return treeRootNodes;  
        }  
  
        **internal** static **int** **GetIdConflictNodesCount**()  
        {  
            MyTrees.**OpenDB**();  
            string sql = "select count(\*) from "  
                + *treeDB*.TableName + "\_calc where sysid in (select v from "   
                + *treeDB*.TableName + "\_profile where k='Conflict')";  
            **int** count = *treeDB*.**SearchCount**(sql);  
            MyTrees.**CloseDB**();  
            return count;  
        }  
        **internal** static List<string> **GetIdConflictNodes**(**int** pageNo, **int** pageSize)  
        {  
            MyTrees.**OpenDB**();  
            string sql = "select \* from "  
                + *treeDB*.TableName + "\_calc where sysid in (select v from "   
                + *treeDB*.TableName + "\_profile where k='Conflict') limit "  
                + (pageNo \* pageSize) + ", " + pageSize;  
            List<string> conflictNodes = *treeDB*.**SearchString**(sql);  
            MyTrees.**CloseDB**();  
            return conflictNodes;  
        }  
          
        **internal** static **int** **GetLeafAloneNodesCount**()  
        {  
            MyTrees.**OpenDB**();  
            string sql = "select count(\*) from "  
                + *treeDB*.TableName + "\_calc where sysid in (select v from "   
                + *treeDB*.TableName + "\_profile where k='Leaf')";  
            **int** count = *treeDB*.**SearchCount**(sql);  
            MyTrees.**CloseDB**();  
            return count;  
        }  
        **internal** static Dictionary<string, string> **GetLeafAloneNodes**(**int** pageNo, **int** pageSize)  
        {  
            MyTrees.**OpenDB**();  
            string sql = "select \* from "  
                + *treeDB*.TableName + "\_calc where sysid in (select v from "   
                + *treeDB*.TableName + "\_profile where k='Leaf') limit "  
                + (pageNo \* pageSize) + ", " + pageSize;  
            List<string> nodes = *treeDB*.**SearchString**(sql);  
            MyTrees.**CloseDB**();  
              
            Dictionary<string, string> leafAloneNodes = **new** Dictionary<string, string>();  
            **foreach** (string node **in** nodes) {  
                string sysid = node.**Substring**(0,node.**IndexOf**(","));  
                **if**(!leafAloneNodes.**ContainsKey**(sysid))  
                {  
                    leafAloneNodes.**Add**(sysid, node);  
                }  
            }  
            return leafAloneNodes;  
        }  
        **private** static List<string> *leafAloneNodeIds*;  
        **internal** static List<string> **GetLeafAloneNodeIds**()  
        {  
            **if**(*leafAloneNodeIds* == **null**)  
            {  
                *leafAloneNodeIds* = **new** List<string>();  
                MyTrees.**OpenDB**();  
                string sql = "select v from " + *treeDB*.TableName + "\_profile where k='Leaf'";  
                *leafAloneNodeIds* = *treeDB*.**SearchString**(sql);  
                MyTrees.**CloseDB**();  
            }  
            return *leafAloneNodeIds*;  
        }  
          
           **internal** static **int** **GetRingNodesCount**()  
           {  
               MyTrees.**OpenDB**();  
            string sql = "select count(\*) from "  
                + *treeDB*.TableName + "\_calc where sysid in (select v from "   
                + *treeDB*.TableName + "\_profile where k='Ring')";  
            **int** count = *treeDB*.**SearchCount**(sql);  
            MyTrees.**CloseDB**();  
            return count;  
           }  
        **internal** static Dictionary<string, string> **GetRingNodes**(**int** pageNo, **int** pageSize)  
        {  
            MyTrees.**OpenDB**();  
            string sql = "select \* from "  
                + *treeDB*.TableName + "\_calc where sysid in (select v from "   
                + *treeDB*.TableName + "\_profile where k='Ring') limit "  
                + (pageNo \* pageSize) + ", " + pageSize;  
            List<string> nodes = *treeDB*.**SearchString**(sql);  
            MyTrees.**CloseDB**();  
              
            Dictionary<string, string> ringNodes = **new** Dictionary<string, string>();  
            **foreach** (string node **in** nodes) {  
                string sysid = node.**Substring**(0,node.**IndexOf**(","));  
                **if**(!ringNodes.**ContainsKey**(sysid))  
                {  
                    ringNodes.**Add**(sysid, node);  
                }  
            }  
            return ringNodes;  
        }  
           **private** static List<string> *ringNodes*;  
        **internal** static List<string> **GetRingNodeIds**()  
        {  
            **if**(*ringNodes* == **null**)  
            {  
                *ringNodes* = **new** List<string>();  
                MyTrees.**OpenDB**();  
                string sql = "select v from " + *treeDB*.TableName + "\_profile where k='Ring'";  
                *ringNodes* = *treeDB*.**SearchString**(sql);  
                MyTrees.**CloseDB**();  
            }  
            return *leafAloneNodeIds*;  
        }  
          
        **private** static List<string> *tableOptCols*;  
        **internal** static List<string> **GetTableOptCols**()  
        {  
            **if**(*tableOptCols* == **null**)  
            {  
                MyTrees.**OpenDB**();  
                string sql = "select v from "  
                    + *treeDB*.TableName   
                    + "\_profile where k='TableOptCol'";  
                *tableOptCols* = *treeDB*.**SearchString**(sql);  
                MyTrees.**CloseDB**();  
            }  
            return *tableOptCols*;  
        }  
              
        #**endregion**  
  
        #**region** 自定义查找  
          
        **public** static List<MyTreeNode> **FindToRootNodeList**(string parentId)  
        {  
            List<MyTreeNode> nodes = **new** List<MyTreeNode>();  
            MyTreeNode parentNode = **GetNodeBySysId**(parentId);  
            **while**(parentNode != **null**)  
            {  
                nodes.**Add**(parentNode);  
                parentNode = **GetNodeBySysId**(parentNode.TopId);  
            }  
              
            return nodes;  
        }  
          
        **public** static List<string> **FindToRootAllList**(string parentId)  
        {  
            List<string> nodes = **new** List<string>();  
            string parentNode = **GetStringBySysId**(parentId);  
            **while**(parentNode != **null**)  
            {  
                nodes.**Add**(parentNode);  
                string[] parentNodes = parentNode.**Split**(**new** **char**[] { ',' });  
                parentNode = **GetStringBySysId**(parentNodes[1]);  
            }  
              
            return nodes;  
        }  
          
        **public** static List<MyTreeNode> **FindBySql**(string sql, List<string> param)  
        {  
            *treeDB*.**OpenDB**();  
            List<MyTreeNode> result = *treeDB*.**SearchNode**(sql, param);  
            *treeDB*.**CloseDB**();  
            return result;  
        }  
          
        **public** static MyTreeNode **GetNodeBySysId**(string sysId)  
        {  
            string sql = "select sysid,topid,name,level,sublevel,subcount,subcountall from "  
                + *treeDB*.TableName + "\_calc where sysid='" + sysId + "' order by subcountall desc";  
            List<MyTreeNode> result = *treeDB*.**SearchNode**(sql);  
            **if**(result.Count>0)  
                return result[0];  
            **else**  
                return **null**;  
        }  
  
        **public** static string **GetStringBySysId**(string sysId)  
        {  
            string sql = "select \* from " + *treeDB*.TableName + "\_calc where sysid='" + sysId + "' order by subcountall desc";  
            List<string> result = *treeDB*.**SearchString**(sql);  
            **if**(result.Count>0)  
                return result[0];  
            **else**  
                return **null**;  
        }  
          
        **public** static List<MyTreeNode> **GetNodesByTopId**(string topId)  
        {  
            string sql = "select sysid,topid,name,level,sublevel,subcount,subcountall from "  
                + *treeDB*.TableName + "\_calc where topid='" + topId + "' order by subcountall desc";  
            return *treeDB*.**SearchNode**(sql);  
        }  
          
        **public** static List<string> **GetAllByTopIds**(string topIds)  
        {  
            string sql = "select \* from "+*treeDB*.TableName+"\_calc where topid in ("+topIds+") order by subcountall desc";  
            return *treeDB*.**SearchString**(sql);  
        }  
          
        #**endregion**  
    }  
}

## 客户端从Sqlite读取树结构代码

/\*  
 \* 由SharpDevelop创建。  
 \* 用户： TomChen  
 \* 日期: 2016/12/15  
 \* 时间: 23:08  
 \*   
 \* 要改变这种模板请点击 工具|选项|代码编写|编辑标准头文件  
 \*/  
**using** System;  
**using** System.Collections.Generic;  
**using** System.Data;  
**using** System.IO;  
**using** System.Text;  
**using** MySql.Data.MySqlClient;  
  
**namespace** MemberTree  
{  
    /// <**summary**>  
    /// Description of DBHelper.  
    /// <**/summary**>  
    **public** class MyTreeDBVMysql : MyTreeDBMysql, IMyTreeDBV  
    {  
        **public** List<MyTreeNode> **SearchNode**(string sql)  
        {  
            *cmd*.CommandText = sql;  
            MySqlDataReader reader = *cmd*.**ExecuteReader**();  
            List<MyTreeNode> result = **new** List<MyTreeNode>();  
            **while** (reader.**Read**())  
            {  
                MyTreeNode node = **new** MyTreeNode()  
                {  
                    SysId = reader.**GetString**(0),   
                    TopId = reader.**GetString**(1),  
                    Name = reader.**GetString**(2),  
                    Level = reader.**GetInt32**(3),  
                    ChildrenLevels = reader.**GetInt32**(4),  
                    ChildrenCount = reader.**GetInt32**(5),  
                    ChildrenCountAll = reader.**GetInt32**(6)  
                };  
                result.**Add**(node);  
            }  
            reader.**Close**();  
            return result;  
        }  
          
        **public** List<MyTreeNode> **SearchNode**(string sql, List<string> param)  
        {  
            **for** (**int** i = 0; i < param.Count; i++)   
            {  
                *cmd*.Parameters.**AddWithValue**(i.**ToString**(), param[i]);  
            }  
            List<MyTreeNode> result = **SearchNode**(sql);  
            *cmd*.Parameters.**Clear**();  
            return result;  
        }  
          
        **public** List<string> **SearchString**(string sql)  
        {  
            *cmd*.CommandText = sql;  
            MySqlDataReader reader = *cmd*.**ExecuteReader**();  
            List<string> result = **new** List<string>();  
            object[] oj = **new** object[reader.FieldCount];  
            **while** (reader.**Read**())  
            {  
                reader.**GetValues**(oj);  
                string node = string.**Join**(",", oj);  
                result.**Add**(node);  
            }  
            reader.**Close**();  
            return result;  
        }  
          
        **public** **int** **SearchCount**(string sql)  
        {  
            *cmd*.CommandText = sql;  
            MySqlDataReader reader = *cmd*.**ExecuteReader**();  
            **int** result = 0;  
            **while** (reader.**Read**())  
            {  
                result = reader.**GetInt32**(0);  
            }  
            reader.**Close**();  
            return result;  
        }  
    }    
}

## 客户端从Mysql读取树结构代码

**namespace** MemberTree  
{  
    /// <**summary**>  
    /// Description of DBHelper.  
    /// <**/summary**>  
    **public** class MyTreeDBVSqlite : MyTreeDBSqlite, IMyTreeDBV  
    {  
        **public** List<MyTreeNode> **SearchNode**(string sql)  
        {  
            *cmd*.CommandText = sql;  
            SQLiteDataReader reader = *cmd*.**ExecuteReader**();  
            List<MyTreeNode> result = **new** List<MyTreeNode>();  
            **while** (reader.**Read**())  
            {  
                MyTreeNode node = **new** MyTreeNode()  
                {  
                    SysId = reader.**GetString**(0),   
                    TopId = reader.**GetString**(1),  
                    Name = reader.**GetString**(2),  
                    Level = reader.**GetInt32**(3),  
                    ChildrenLevels = reader.**GetInt32**(4),  
                    ChildrenCount = reader.**GetInt32**(5),  
                    ChildrenCountAll = reader.**GetInt32**(6),  
                };  
                result.**Add**(node);  
            }  
            reader.**Close**();  
            return result;  
        }  
          
        **public** List<MyTreeNode> **SearchNode**(string sql, List<string> param)  
        {  
            **for** (**int** i = 0; i < param.Count; i++)   
            {  
                *cmd*.Parameters.**AddWithValue**(i.**ToString**(), param[i]);  
            }  
            List<MyTreeNode> result = **SearchNode**(sql);  
            *cmd*.Parameters.**Clear**();  
            return result;  
        }  
          
        **public** List<string> **SearchString**(string sql)  
        {  
            *cmd*.CommandText = sql;  
            SQLiteDataReader reader = *cmd*.**ExecuteReader**();  
//            WindowView.notify.SetStatusMessage("正在查询子节点。。。");  
            List<string> result = **new** List<string>();  
            object[] oj = **new** object[reader.FieldCount];  
            **while** (reader.**Read**())  
            {  
                reader.**GetValues**(oj);  
                string node = string.**Join**(",", oj);  
                result.**Add**(node);  
            }  
//            WindowView.notify.SetStatusMessage("查询子节点完成，子节点数量："+result.Count);  
            reader.**Close**();  
            return result;  
        }  
          
        **public** **int** **SearchCount**(string sql)  
        {  
            *cmd*.CommandText = sql;  
            SQLiteDataReader reader = *cmd*.**ExecuteReader**();  
            **int** result = 0;  
            **while** (reader.**Read**())  
            {  
                result = reader.**GetInt32**(0);  
            }  
            reader.**Close**();  
            return result;  
        }  
    }    
}

# 客户端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);  
            }  
        }  
    }  
}

# 客户端数据导出模块

## 导出保存为Csv文件

**namespace** MemberTree  
{  
    **public** class ExportCSV  
    {  
        **private** static **int** *row*;  
        **private** static **int** *allRow*;        
          
        **public** static void **ExportNodes**(MyTreeNode node)  
        {  
            **if** (node == **null**)  
            {  
                MessageBox.**Show**("必须选中一个节点！");  
                return;  
            }  
            SaveFileDialog saveFileDlg = **new** SaveFileDialog();  
            saveFileDlg.Title = "选择将会员树导出为文件的位置";  
//            saveFileDlg.Filter = "CSV文件|\*.csv|Excel2007文件|\*.xlsx";  
            saveFileDlg.Filter = "CSV文件|\*.csv";  
            saveFileDlg.FileName = node.**ToString**();  
            **if** (saveFileDlg.**ShowDialog**() == **true**)  
            {  
                **if**(File.**Exists**(saveFileDlg.FileName))  
                {  
                    File.**Delete**(saveFileDlg.FileName);  
                }  
                ExportCSV.**Export2CSV**(node, saveFileDlg.FileName);  
            }  
        }  
  
        **private** static List<string> *ringNodeIds* = **new** List<string>();  
        //判断闭环是否关闭  
        **private** static **bool** **isRingClose**(string id)  
        {  
            **if** (MyTrees.**GetRingNodeIds**().**Contains**(id))  
            {  
                **if** (*ringNodeIds*.**Contains**(id))  
                {  
                    return **true**;  
                }  
                **else**  
                {  
                    *ringNodeIds*.**Add**(id);  
                }  
            }  
            return **false**;  
        }  
  
        **public** static void **Export2CSV**(MyTreeNode node, string outputfile)  
        {  
            *ringNodeIds*.**Clear**();  
            WindowView.*notify*.**SetProcessBarVisible**(**true**);  
            WindowView.*notify*.**SetStatusMessage**("开始导出文件......");  
            TimingUtil.**StartTiming**();  
    
            StreamWriter mysw = **new** StreamWriter(outputfile, **true**, Encoding.Default);  
             
            List<string> optCols = MyTrees.**GetTableOptCols**();  
            string header = string.**Join**(",", optCols.**ToArray**());  
            header = "会员ID,父级ID,会员姓名,所在层级,下级层数,直接下级会员数,下级会员总数," + header;  
            mysw.**WriteLine**(header);  
  
            StringBuilder allLines = **new** StringBuilder();  
            MyTrees.**OpenDB**();  
              
            *row* = 2;  
            *allRow* = node.ChildrenCountAll + 1;  
    
            //导出所有父节点  
            **ExportAllParents2CSV**(mysw, allLines, node);  
              
            //导出该节点  
            string nodestr = MyTrees.**GetStringBySysId**(node.SysId);  
            mysw.**WriteLine**(nodestr);  
              
            //导出该节点所有子节点  
            **ExportAllChildren2CSV**(mysw, allLines, node);  
  
            MyTrees.**CloseDB**();  
            mysw.**Close**();  
  
            WindowView.*notify*.**SetStatusMessage**(TimingUtil.**EndTiming**());  
            WindowView.*notify*.**SetProcessBarVisible**(**false**);  
        }  
          
        //导出所有父节点  
        **private** static void **ExportAllParents2CSV**(StreamWriter mysw, StringBuilder allLines, MyTreeNode node)  
        {  
            WindowView.*notify*.**SetStatusMessage**("正在导出该节点的所有父节点。。。");  
            List<string> parentNodes = MyTrees.**FindToRootAllList**(node.TopId);  
            parentNodes.**Reverse**();  
            *allRow* += parentNodes.Count;  
              
            **for** (**int** i = 0; i < parentNodes.Count; i++)   
            {  
                mysw.**WriteLine**(parentNodes[i]);  
            }  
        }  
          
        //导出所有子孙节点  
        **private** static void **ExportAllChildren2CSV**(StreamWriter mysw, StringBuilder allLines, MyTreeNode node)  
        {  
            List<string> topIds = **new** List<string>();  
            List<string> subNodes = MyTrees.**GetAllByTopIds**("'"+node.SysId+"'");  
            **int** levelNum = 1;  
            **while** (subNodes.Count > 0)   
            {  
                topIds.**Clear**();  
                **for** (**int** i = 0; i < subNodes.Count; i++)   
                {  
                    mysw.**WriteLine**(subNodes[i]);  
                    topIds.**Add**("'"+subNodes[i].**Substring**(0,subNodes[i].**IndexOf**(","))+"'");  
                }  
                WindowView.*notify*.**SetStatusMessage**("正在导出该节点的第"+levelNum+"层（共"+subNodes.Count+"个）子节点，共有"+node.ChildrenLevels+"层子节点");  
                WindowView.*notify*.**SetProcessBarValue**((**int**)(100.0 \* levelNum / node.ChildrenLevels));  
                levelNum++;  
                subNodes = MyTrees.**GetAllByTopIds**(string.**Join**(",", topIds));  
            }  
        }  
  
        **private** static void **Export2CSVImp**(StreamWriter mysw, StringBuilder allLines, MyTreeNode node, **bool** recu)  
        {  
            **if** (**isRingClose**(node.SysId))  
            {  
                return;  
            }  
  
            allLines.**Clear**();  
            allLines.**Append**(node.SysId);  
            allLines.**Append**(",");  
            allLines.**Append**(node.TopId);  
            allLines.**Append**(",");  
            allLines.**Append**(node.Name);  
            allLines.**Append**(",");  
            allLines.**Append**(node.Level);  
            allLines.**Append**(",");  
            allLines.**Append**(node.ChildrenLevels);  
            allLines.**Append**(",");  
            allLines.**Append**(node.ChildrenCount);  
            allLines.**Append**(",");  
            allLines.**Append**(node.ChildrenCountAll);  
            allLines.**Append**(",");  
            **foreach** (string otherProp **in** node.*OtherProps*)   
            {  
                allLines.**Append**(",");  
                allLines.**Append**(otherProp);  
            }  
              
            mysw.**WriteLine**(allLines.**ToString**());  
            *row*++;  
            **if** (*row* % 100 == 0)  
            {  
                WindowView.*notify*.**SetProcessBarValue**((**int**)(100.0 \* *row* / *allRow*));  
                WindowView.*notify*.**SetStatusMessage**("正在导出第" + *row* + "个节点（总共" + *allRow* + "个节点）");  
            }  
  
            **if**(recu)  
            {  
                List<MyTreeNode> childrenNodes = MyTrees.**GetNodesByTopId**(node.SysId);  
                **foreach** (MyTreeNode subNode **in** childrenNodes)  
                {  
                    **Export2CSVImp**(mysw ,allLines, subNode, recu);  
                }  
            }  
        }  
    }  
}

## 导出保存为Png图片

**namespace** MemberTree  
{  
    /// <**summary**>  
    /// Description of ExportIMG.  
    /// <**/summary**>  
    **public** class ExportIMG  
    {          
        **internal** static void **SaveNode2Image**(MyTreeView mytreeview, MyTreeNode node)  
        {  
            **if**(node == **null**)  
            {  
                  MessageBox.**Show**("必须选中一个节点！");  
            }  
            **else**  
            {  
                SaveFileDialog saveFileDlg = **new** SaveFileDialog();  
                saveFileDlg.Title = "选择将会员树导出为文件的位置";  
                saveFileDlg.Filter = "png格式|\*.png";  
                saveFileDlg.FileName = node.**ToString**();  
                **if** (saveFileDlg.**ShowDialog**() == **true**)  
                {  
                    **SaveImage**(mytreeview.*memberTreeView*, saveFileDlg.FileName);  
                }  
            }  
        }  
          
        **internal** static void **SaveImage**(FrameworkElement frmEle, string imgFile)  
        {  
            FileStream fs = **new** FileStream(imgFile, **FileMode**.*Create*);  
            **int** width = (**int**)frmEle.ActualWidth;  
            **int** height = (**int**)frmEle.ActualHeight;  
            RenderTargetBitmap bmp = **new** RenderTargetBitmap(width, height, 96, 96, PixelFormats.Default);  
            bmp.**Render**(frmEle);  
            BitmapEncoder encoder = **new** PngBitmapEncoder();  
            encoder.Frames.**Add**(BitmapFrame.**Create**(bmp));  
            encoder.**Save**(fs);  
            fs.**Close**();  
            fs.**Dispose**();  
        }  
    }  
}

## 导出保存为PDF文件

**namespace** MemberTree  
{  
    /// <**summary**>  
    /// Description of ExportPDF.  
    /// <**/summary**>  
    **public** class ExportPDF  
    {  
//        private const string TMP\_DIR = "temp";  
          
        **public** static void **Export2PDF**(MyTreeView mytreeview, MyTreeNode node)  
        {  
            **if**(node == **null**)  
            {  
                  MessageBox.**Show**("必须选中一个节点！");  
                  return;  
            }  
            // disable once SuggestUseVarKeywordEvident  
            SaveFileDialog openfileDlg = **new** SaveFileDialog();  
            openfileDlg.Title = "选择将会员树导出为pdf文件的位置";  
            openfileDlg.Filter = "pdf文件|\*.pdf";  
            openfileDlg.FileName = node.**ToString**();  
            **if** (openfileDlg.**ShowDialog**() == **true**)  
            {  
                TimingUtil.**StartTiming**();  
                Directory.**CreateDirectory**(MemData.*MemDataTemp*);  
                **ExportAllImgs**(mytreeview, node);  
                  
                //定义一个Document，并设置页面大小为A4，竖向   
        //            Document doc = new Document(PageSize.A4);  
                Document doc = **new** Document();  
                PdfWriter.**GetInstance**(doc, **new** FileStream(openfileDlg.FileName, **FileMode**.*Create*));  
                  
                //设置PDF的头信息，一些属性设置，在Document.Open 之前完成  
                doc.**AddAuthor**("TomChen");  
                doc.**AddCreationDate**();  
                doc.**AddCreator**("湖南警察学院");  
                doc.**AddSubject**("将选中的会员树导出为PDF格式，如果数据量大，则导出为多张PDF页面");  
                doc.**AddTitle**("将选中的会员树导出为PDF");  
                doc.**AddKeywords**("会员树,会员层级,PDF");  
                //自定义头   
                //doc.AddHeader("Expires", "0");  
                  
                doc.**Open**();  
                  
                string[] imgfiles = Directory.**GetFiles**(MemData.*MemDataTemp*, "\*.png");  
                  
                //首页  
                //写入文字  
                Paragraph paragraph = **new** Paragraph("ID:" + node.SysId, **new** Font(Font.**FontFamily**.*TIMES\_ROMAN*, 30, 0, BaseColor.*BLUE*));  
                doc.**Add**(paragraph);  
                paragraph = **new** Paragraph("TopID:" + node.TopId, **new** Font(Font.**FontFamily**.*TIMES\_ROMAN*, 30, 0, BaseColor.*BLUE*));  
                doc.**Add**(paragraph);  
                paragraph = **new** Paragraph("Level:" + node.Level, **new** Font(Font.**FontFamily**.*TIMES\_ROMAN*, 30, 0, BaseColor.*BLUE*));  
                doc.**Add**(paragraph);  
                paragraph = **new** Paragraph("Sub1:" + node.ChildrenCount, **new** Font(Font.**FontFamily**.*TIMES\_ROMAN*, 30, 0, BaseColor.*BLUE*));  
                doc.**Add**(paragraph);  
                paragraph = **new** Paragraph("SubLevel:" + node.ChildrenLevels, **new** Font(Font.**FontFamily**.*TIMES\_ROMAN*, 30, 0, BaseColor.*BLUE*));  
                doc.**Add**(paragraph);  
                paragraph = **new** Paragraph("SubAll:" + node.ChildrenCount, **new** Font(Font.**FontFamily**.*TIMES\_ROMAN*, 30, 0, BaseColor.*BLUE*));  
                doc.**Add**(paragraph);  
                paragraph = **new** Paragraph("----------------------------------------", **new** Font(Font.**FontFamily**.*COURIER*, 20, 0, BaseColor.*GREEN*));  
                doc.**Add**(paragraph);  
                paragraph = **new** Paragraph("Export pictures count:" + imgfiles.Length, **new** Font(Font.**FontFamily**.*COURIER*, 20, 0, BaseColor.*GREEN*));  
                doc.**Add**(paragraph);  
                  
                WindowView.*notify*.**SetProcessBarVisible**(**true**);  
                  
                **for** (**int** i = 0; i < imgfiles.Length; i++)   
                {  
                    //写入图片  
                    iTextSharp.text.Image img = iTextSharp.text.Image.**GetInstance**(imgfiles[i]);  
                    doc.**SetPageSize**(**new** Rectangle(img.Width, img.Height));  
                    //新建一页  
                    doc.**NewPage**();  
                    //img.ScaleAbsoluteWidth(PageSize.A4.Width);  
                    //img.ScaleAbsoluteHeight(PageSize.A4.Height - 100);  
                    //img.SetAbsolutePosition((PageSize.POSTCARD.Width - img.ScaledWidth) / 2, (PageSize.POSTCARD.Height - img.ScaledHeight) / 2);  
                    doc.**Add**(img);  
                    WindowView.*notify*.**SetStatusMessage**("正在生成PDF文件"+ i + "/" + imgfiles.Length);  
                    WindowView.*notify*.**SetProcessBarValue**(i \* 100.0 /imgfiles.Length);  
                }  
          
                doc.**Close**();  
                Directory.**Delete**(MemData.*MemDataTemp*, **true**);  
                WindowView.*notify*.**SetProcessBarVisible**(**false**);  
                WindowView.*notify*.**SetStatusMessage**(TimingUtil.**EndTiming**());  
                MessageBox.**Show**("导出PDF完成！");  
            }  
        }  
          
        //导出过程中未打开的子节点的集合  
        **private** static List<MyTreeNode> *exportNodes* = **new** List<MyTreeNode>();  
        **private** static void **ExportAllImgs**(MyTreeView mytreeview, MyTreeNode node)  
        {  
            mytreeview.**BeginExportImg**();  
            *exportNodes*.**Clear**();  
            *exportNodes*.**Add**(node);  
            MyTrees.**OpenDB**();  
            **while**(*exportNodes*.Count>0)  
            {  
                **ExportImg**(mytreeview, *exportNodes*[0]);  
            }  
            mytreeview.**EndExportImg**();  
            MyTrees.**CloseDB**();  
        }  
          
        **private** static void **ExportImg**(MyTreeView mytreeview, MyTreeNode node)  
        {  
            mytreeview.**SetRootNode**(node);  
            //如果全部子节点数量小于50，则直接全部打开  
            **if**(node.ChildrenCountAll<50)  
            {  
                mytreeview.**ExpandRootNode**(node.ChildrenLevels);  
            }  
            **else**  
            {  
                //已打开的节点数量  
                **int** expNodeCount = node.ChildrenCount;  
                  
                //这个循环用于防止出现大量的上下级只有一个子节点的情况导致打开节点数量过少  
                TreeViewItem rootItem = mytreeview.*memberTreeView*.Items[0] **as** TreeViewItem;  
                MyTreeNode rootNode = rootItem.Tag **as** MyTreeNode;  
                **while**(rootNode.ChildrenCount == 1)  
                {  
                    mytreeview.**ExpandNode**(rootItem, 1);  
                    rootItem = rootItem.Items[0] **as** TreeViewItem;  
                    rootNode = rootItem.Tag **as** MyTreeNode;  
                    expNodeCount++;  
                }  
                  
                //打开一级子节点，再逐个判断数量  
                mytreeview.**ExpandNode**(rootItem, 1);  
  
                **foreach** (TreeViewItem subItem **in** rootItem.Items)  
                {  
                    MyTreeNode subNode = subItem.Tag **as** MyTreeNode;  
                    **if**(subNode.ChildrenCountAll >0)  
                    {  
                        **if**(subNode.ChildrenCountAll < 10 || subNode.ChildrenCountAll + expNodeCount < 50)  
                        {//如果全部子节点数量小于10，或者全部子节点数量加上已打开的节点数量小于50，则该子节点全部打开  
                            mytreeview.**ExpandNode**(subItem, subNode.ChildrenLevels);  
                            expNodeCount += subNode.ChildrenCountAll;  
                        }  
                        **else** **if**(subNode.ChildrenCount + expNodeCount < 50)  
                        {//如果则该节点一级子节点数量加上已打开的节点数量小于50，则该子节点打开子一级节点  
                            mytreeview.**ExpandNode**(subItem, 1);  
                            expNodeCount += subNode.ChildrenCount;  
                            //遍历孙子节点  
                            **foreach** (TreeViewItem grandItem **in** subItem.Items)  
                            {  
                                MyTreeNode grandNode = grandItem.Tag **as** MyTreeNode;  
                                **if**(grandNode.ChildrenCountAll < 10 || grandNode.ChildrenCountAll + expNodeCount < 50)  
                                {  
                                    mytreeview.**ExpandNode**(grandItem, grandNode.ChildrenLevels);  
                                    expNodeCount += grandNode.ChildrenCountAll;  
                                }  
                                **else**  
                                {  
                                    *exportNodes*.**Add**(grandNode);  
                                }  
                            }  
                        }  
                        **else**  
                        {//如果全部子节点数量加上已打开的节点数量大于50，则该子节点不打开  
                            *exportNodes*.**Add**(subNode);  
                        }  
                    }  
                }  
            }  
            *exportNodes*.**Remove**(node);  
            WindowView.*notify*.**SetStatusMessage**("正在导出图片"+ node.SysId);  
            ExportIMG.**SaveImage**(mytreeview.*memberTreeView*, MemData.*MemDataTemp* + "/" + node.SysId + ".png");  
        }  
    }  
}

# 客户端查询分页模块

## 模糊查询条件处理代码

**namespace** MemberTree  
{  
    /// <**summary**>  
    /// Interaction logic for SearchFilter.xaml  
    /// <**/summary**>  
    **public** partial class SearchFilter : UserControl  
    {  
        **private** List<ComboBox> *comboCols* = **new** List<ComboBox>();  
        **private** List<TextBox> *txtCols* = **new** List<TextBox>();  
        **private** List<string> *searchParams*;  
        **public** SearchFilter()  
        {  
            **InitializeComponent**();  
        }  
          
        **internal** void **InitCols**()  
        {          
            List<string> optCols = MyTrees.**GetTableOptCols**();  
            **for** (**int** i = 0; i < optCols.Count; i++)   
            {  
                *mainGrid*.RowDefinitions.**Add**(**new** RowDefinition());  
              
                TextBlock header = **new** TextBlock();  
                header.Text = optCols[i];  
                *mainGrid*.Children.**Add**(header);  
                Grid.**SetRow**(header, i+8);  
                  
                ComboBox comb = **new** ComboBox();  
                ComboBoxItem comItem1 = **new** ComboBoxItem();  
                comItem1.Content = "等于";  
                comItem1.IsSelected = **true**;  
                comb.Items.**Add**(comItem1);  
                ComboBoxItem comItem2 = **new** ComboBoxItem();  
                comItem2.Content = "开头";  
                comb.Items.**Add**(comItem2);  
                ComboBoxItem comItem3 = **new** ComboBoxItem();  
                comItem3.Content = "结尾";  
                comb.Items.**Add**(comItem3);  
                ComboBoxItem comItem4 = **new** ComboBoxItem();  
                comItem4.Content = "包含";  
                comb.Items.**Add**(comItem4);  
                *mainGrid*.Children.**Add**(comb);  
                Grid.**SetRow**(comb, i+8);  
                Grid.**SetColumn**(comb, 1);  
                *comboCols*.**Add**(comb);  
                  
                TextBox val = **new** TextBox();  
                *mainGrid*.Children.**Add**(val);  
                Grid.**SetRow**(val, i+8);  
                Grid.**SetColumn**(val, 2);  
                *txtCols*.**Add**(val);  
            }  
        }  
          
        **public** string **GetSearchSql**()  
        {  
            *searchParams* = **new** List<string>();  
            StringBuilder sb = **new** StringBuilder();  
            **AddFilter**(sb, "sysid", *txtSysid*, *comboSysid*);  
            **AddFilter**(sb, "topid", *txtTopid*, *comboTopid*);  
            **AddFilter**(sb, "name", *txtName*, *comboName*);  
            **AddFilter**(sb, "level", *txtLevel*, *comboLevel*);  
            **AddFilter**(sb, "sublevel", *txtSubLevels*, *comboSubLevels*);  
            **AddFilter**(sb, "subcount", *txtSubCount*, *comboSubCount*);  
            **AddFilter**(sb, "subcountall", *txtAllSubCount*, *comboAllSubCount*);  
            List<string> optCols = MyTrees.**GetTableOptCols**();  
            **for** (**int** i = 0; i < optCols.Count; i++)   
            {  
                **AddFilter**(sb, optCols[i], *txtCols*[i], *comboCols*[i]);  
            }  
  
            **if**(sb.Length>0)  
            {  
                return "select sysid,topid,name,level,sublevel,subcount,subcountall from "   
                    + MyTrees.*treeDB*.TableName + "\_calc where 1=1" + sb + " order by subcountall desc limit 0, 100";  
            }  
            **else**  
            {  
                return **null**;  
            }  
        }  
        **public** List<string> **GetSearchParams**()  
        {  
            return *searchParams*;  
        }  
          
        **private** void **AddFilter**(StringBuilder sb, string col, TextBox txtBox, ComboBox comboOpr)  
        {  
            string txt = txtBox.Text.**Trim**();  
            **if**(txt != "")  
            {  
                string opr = comboOpr.SelectionBoxItem.**ToString**();  
                sb.**Append**(" and ");  
                sb.**Append**(col);  
                **if**(opr == "等于")  
                {  
                    sb.**Append**(" = @" + *searchParams*.Count);  
                    *searchParams*.**Add**(txt);  
                }  
                **else** **if**(opr == "大于")  
                {  
                    sb.**Append**(" > @" + *searchParams*.Count);  
                    *searchParams*.**Add**(txt);  
                }  
                **else** **if**(opr == "小于")  
                {  
                    sb.**Append**(" < @" + *searchParams*.Count);  
                    *searchParams*.**Add**(txt);  
                }  
                **else** **if**(opr == "开头")  
                {  
                    sb.**Append**(" like @" + *searchParams*.Count);  
                    *searchParams*.**Add**(txt+"%");  
                }  
                **else** **if**(opr == "结尾")  
                {  
                    sb.**Append**(" like @" + *searchParams*.Count);  
                    *searchParams*.**Add**("%"+txt);  
                }  
                **else** **if**(opr == "包含")  
                {  
                    sb.**Append**(" like @" + *searchParams*.Count);  
                    *searchParams*.**Add**("%"+txt+"%");  
                }  
            }  
        }  
          
        //清除重置查找条件  
        **private** void **BtnClearFilter\_Click**(object sender, RoutedEventArgs e)  
        {  
            *txtSysid*.**Clear**();  
            *txtTopid*.**Clear**();  
            *txtName*.**Clear**();  
            *txtLevel*.**Clear**();  
            *txtSubLevels*.**Clear**();  
            *txtSubCount*.**Clear**();  
            *txtAllSubCount*.**Clear**();  
            **for** (**int** i = 0; i < *txtCols*.Count; i++)   
            {  
                *txtCols*[i].**Clear**();  
            }  
        }  
    }  
}

## 分页处理代码

**namespace** MemberTree  
{  
    /// <**summary**>  
    /// Interaction logic for PageBar.xaml  
    /// <**/summary**>  
    **public** partial class PageBar : UserControl  
    {  
        **private** **int** *pageNo* = 1;  // 当前页    
        **private** **int** *pageSize* = 50;  // 每页记录数    
        **private** **int** *totalCount* = 0;  // 总记录数  
        **private** **int** *pageCount* = 1;  // 总页数  
    
        **public** PageBar()   
        {  
            **InitializeComponent**();  
            **this**.*prePageBtn*.IsEnabled = **false**;    
        }  
          
        **public** **int** PageSize {  
            get {    
                return *pageSize*;    
            }    
            set {    
                **if** (**value** > 0) {    
                    *pageSize* = **value**;    
                }    
            }    
        }   
        /// <**summary**>  
        /// 获取分页数据委托  
        /// <**/summary**>  
        /// <**param** **name="pageNo"**>页码数<**/param**>  
        /// <**param** **name="pageSize"**>每页记录数<**/param**>  
        **public** delegate void GetDataDelegate(**int** pageNo, **int** pageSize);    
        **private** GetDataDelegate *getDataDelegateHandler*;    
        **public** PageBar.GetDataDelegate GetDataDelegateHandler {    
            set {    
                *getDataDelegateHandler* = **value**;    
            }    
        }  
          
        **public** void **Init**(**int** totalCount)  
        {  
            **this**.*totalCount* = totalCount;  
            **if**(totalCount % *pageSize* == 0)  
            {  
                *pageCount* = totalCount / *pageSize*;  
            }  
            **else**  
            {  
                *pageCount* = totalCount / *pageSize* + 1;    
            }  
            **if**(*pageCount* == 1)  
            {  
                **this**.Visibility = **Visibility**.*Collapsed*;  
            }  
            **else**  
            {  
                **this**.Visibility = **Visibility**.*Visible*;  
            }  
              
            **GotoPage**(1);  
        }  
          
        **public** void **GotoPage**(**int** pageNo) {  
            **if** (pageNo <= 0) {    
                pageNo = 1;    
            }    
    
            **this**.*pageNo* = pageNo;    
    
            **try** {    
                *getDataDelegateHandler*(pageNo - 1, *pageSize*);  
                // 页码显示    
                **this**.*totalCountTbk*.Text = *totalCount*.**ToString**();  
                **this**.*pageNoTbk*.Text = pageNo.**ToString**();  
                **this**.*pageCountTbk*.Text = *pageCount*.**ToString**();  
    
                // 按钮状态    
                **this**.*prePageBtn*.IsEnabled = pageNo > 1;    
                **this**.*firstPageBtn*.IsEnabled = pageNo > 1;    
                **this**.*nextPageBtn*.IsEnabled = pageNo < *pageCount*;    
                **this**.*lastPageBtn*.IsEnabled = pageNo < *pageCount*;    
            } **catch** (Exception ex) {  
                **this**.*pageNoTbk*.Text = "";    
                **this**.*pageCountTbk*.Text = "";    
            }    
        }  
    
        // 首页事件       
        **private** void **firstPageBtn\_Click**(object sender, RoutedEventArgs e) {    
            **GotoPage**(1);   
        }    
    
        // 上一页   
        **private** void **prePageBtn\_Click**(object sender, RoutedEventArgs e) {    
            **if** (*pageNo* > 1) {    
                *pageNo* -= 1;    
                **GotoPage**(*pageNo*);    
            }    
        }    
    
        // 下一页     
        **private** void **nextPageBtn\_Click**(object sender, RoutedEventArgs e) {    
            **if** (*pageNo* == 1 || *pageNo* < *pageCount*) {    
                *pageNo* += 1;    
                **GotoPage**(*pageNo*);    
            }    
        }    
    
        // 末页  
        **private** void **lastPageBtn\_Click**(object sender, RoutedEventArgs e) {    
            **GotoPage**(*pageCount*);    
        }    
    
        // 跳转到指定页   
        **private** void **gotoBtn\_Click**(object sender, RoutedEventArgs e) {    
            **try** {    
                **int** pageNo = Convert.**ToInt32**(**this**.*gotoPageNoTb*.Text);    
                **if** (pageNo >= 1 && pageNo <= *pageCount*) {    
                    **GotoPage**(pageNo);    
                } **else** {    
                    MessageBox.**Show**("请输入正确的页码范围：1 ~ " + *pageCount*);    
                }    
            } **catch** (Exception ex) {  
                MessageBox.**Show**("输入的页码不正确！"+ex.Message);  
            }    
        }  
    }    
}

# 加密解密模块

## DES加密解密代码

**namespace** MemberTree  
{  
  
    /// <**summary**>  
    /// 加密解密  
    /// <**/summary**>  
    **public** static class EncryptHelper  
    {  
        /// <**summary**>  
        /// 加密字符  
        /// <**/summary**>  
        **public** const string *SECRETKEY* = "HNPOLICE";  
  
         /// <**summary**>  
        /// 加密文件  
        /// <**/summary**>  
        /// <**param** **name="filename"**>文件存放路径<**/param**>  
        /// <**param** **name="msg"**>加密内容<**/param**>  
        **public** static void **FileEncrypt**(string filename, string msg, string seckey=*SECRETKEY*)  
        {  
            FileStream fsEncrypted = **new** FileStream(filename,**FileMode**.*Create*, **FileAccess**.*Write*);  
            DESCryptoServiceProvider DES = **new** DESCryptoServiceProvider();  
            DES.Key = ASCIIEncoding.ASCII.**GetBytes**(seckey);  
            DES.IV = ASCIIEncoding.ASCII.**GetBytes**(seckey);  
            ICryptoTransform desencrypt = DES.**CreateEncryptor**();  
            CryptoStream cryptostream = **new** CryptoStream(fsEncrypted,  desencrypt, **CryptoStreamMode**.*Write*);  
            **byte**[] fsInput = Encoding.UTF8.**GetBytes**(msg);  
            cryptostream.**Write**(fsInput, 0, fsInput.Length);  
            cryptostream.**Close**();  
            fsEncrypted.**Close**();  
        }  
          
        /// <**summary**>  
        /// 解密文件  
        /// <**/summary**>  
        /// <**param** **name="filename"**>打开文件路径<**/param**>  
        /// <**returns**>返回加密文件的内容<**/returns**>  
        **public** static string **FileDecrypt**(string filename, string seckey=*SECRETKEY*)  
        {  
            DESCryptoServiceProvider DES = **new** DESCryptoServiceProvider();  
            DES.Key = ASCIIEncoding.ASCII.**GetBytes**(seckey);  
            DES.IV = ASCIIEncoding.ASCII.**GetBytes**(seckey);  
            FileStream fsread = **new** FileStream(filename,  **FileMode**.*Open*,  **FileAccess**.*Read*);  
            ICryptoTransform desdecrypt = DES.**CreateDecryptor**();  
            CryptoStream cryptostreamDecr = **new** CryptoStream(fsread,  desdecrypt,  **CryptoStreamMode**.*Read*);  
            StreamReader read = **new** StreamReader(cryptostreamDecr, Encoding.UTF8);  
            string reft = read.**ReadToEnd**();  
            fsread.**Flush**();  
            fsread.**Close**();  
            return reft;  
        }  
          
         #**region** 加密字符串    
        /// <**summary**> /// 加密字符串     
        /// <**/summary**>    
        /// <**param** **name="str"**>要加密的字符串<**/param**>    
        /// <**returns**>加密后的字符串<**/returns**>    
        **public** static string **Encrypt**(string str, string seckey=*SECRETKEY*)  
        {      
            DESCryptoServiceProvider descsp = **new** DESCryptoServiceProvider();   //实例化加/解密类对象  
            **byte**[] key = Encoding.ASCII.**GetBytes**(seckey); //定义字节数组，用来存储密钥  
            **byte**[] data = Encoding.ASCII.**GetBytes**(str);//定义字节数组，用来存储要加密的字符串    
            MemoryStream MStream = **new** MemoryStream(); //实例化内存流对象        
            //使用内存流实例化加密流对象     
            CryptoStream CStream = **new** CryptoStream(MStream, descsp.**CreateEncryptor**(key, key), **CryptoStreamMode**.*Write*);  
            CStream.**Write**(data, 0, data.Length);  //向加密流中写入数据        
            CStream.**FlushFinalBlock**();              //释放加密流        
            return Convert.**ToBase64String**(MStream.**ToArray**());//返回加密后的字符串    
        }    
        #**endregion**   
    
        #**region** 解密字符串     
        /// <**summary**>    
        /// 解密字符串     
        /// <**/summary**>    
        /// <**param** **name="str"**>要解密的字符串<**/param**>    
        /// <**returns**>解密后的字符串<**/returns**>    
        **public** static string **Decrypt**(string str, string seckey=*SECRETKEY*)    
        {        
            DESCryptoServiceProvider descsp = **new** DESCryptoServiceProvider();   //实例化加/解密类对象  
            **byte**[] key = Encoding.ASCII.**GetBytes**(seckey); //定义字节数组，用来存储密钥      
            **byte**[] data = Convert.**FromBase64String**(str);//定义字节数组，用来存储要解密的字符串    
            MemoryStream MStream = **new** MemoryStream(); //实例化内存流对象        
            //使用内存流实例化解密流对象         
            CryptoStream CStream = **new** CryptoStream(MStream, descsp.**CreateDecryptor**(key, key), **CryptoStreamMode**.*Write*);     
            CStream.**Write**(data, 0, data.Length);      //向解密流中写入数据       
            CStream.**FlushFinalBlock**();               //释放解密流        
            return Encoding.ASCII.**GetString**(MStream.**ToArray**());       //返回解密后的字符串    
        }    
        #**endregion**   
    }  
}

## RSA加密解密代码

**namespace** RSACommon  
{  
    /// <**summary**>  
    /// 非对称RSA加密类 可以参考  
    /// 若是私匙加密 则需公钥解密  
    /// 反正公钥加密 私匙来解密  
    /// 需要BigInteger类来辅助  
    /// <**/summary**>  
    **public** static class RSAHelper  
    {  
        /// <**summary**>  
        /// RSA的容器 可以解密的源字符串长度为 DWKEYSIZE/8-11   
        /// <**/summary**>  
        **public** const **int** *DWKEYSIZE* = 1024;  
          
        **public** const string *publicKey* ="AwEAAdRuUg26vNnnSLp1JRPsDWQWjr/S+fX67Blv2Er5XiIuksPWbBq9L7WpcPN2yiiQdlOlqhLgMigKaDaHwRp2ob8y2aCCja1Vi3nZymFK23h9wWwdfPuV0vfnuQ74EcF7K6vOTw6iOcaOUTvDe3tZuS9raCgdfaLrPKzwotc0Jn31";  
  
        /// <**summary**>  
        /// RSA加密的密匙结构  公钥和私匙  
        /// <**/summary**>  
        **public** class RSAKey  
        {  
            **public** string PublicKey{get;set;}  
            **public** string PrivateKey{get;set;}  
        }  
  
        #**region** 得到RSA的解谜的密匙对  
        /// <**summary**>  
        /// 得到RSA的解谜的密匙对  
        /// <**/summary**>  
        /// <**returns**><**/returns**>  
        **public** static RSAKey **GetRASKey**()  
        {  
            RSACryptoServiceProvider.UseMachineKeyStore = **true**;  
            //声明一个指定大小的RSA容器  
            RSACryptoServiceProvider rsaProvider = **new** RSACryptoServiceProvider(*DWKEYSIZE*);  
            //取得RSA容易里的各种参数  
            **RSAParameters** p = rsaProvider.**ExportParameters**(**true**);  
  
            return **new** RSAKey()  
            {  
                PublicKey = **ComponentKey**(p.*Exponent*,p.*Modulus*),  
                PrivateKey = **ComponentKey**(p.*D*,p.*Modulus*)  
            };  
        }  
        #**endregion**  
  
        #**region** 检查明文的有效性 DWKEYSIZE/8-11 长度之内为有效 中英文都算一个字符  
        /// <**summary**>  
        /// 检查明文的有效性 DWKEYSIZE/8-11 长度之内为有效 中英文都算一个字符  
        /// <**/summary**>  
        /// <**param** **name="source"**><**/param**>  
        /// <**returns**><**/returns**>  
        **public** static **bool** **CheckSourceValidate**(string source)  
        {  
            return (*DWKEYSIZE* / 8 - 11) >= source.Length;  
        }  
        #**endregion**  
  
        #**region** 组合解析密匙  
        /// <**summary**>  
        /// 组合成密匙字符串  
        /// <**/summary**>  
        /// <**param** **name="b1"**><**/param**>  
        /// <**param** **name="b2"**><**/param**>  
        /// <**returns**><**/returns**>  
        **private** static string **ComponentKey**(**byte**[] b1, **byte**[] b2)  
        {  
            List<**byte**> list = **new** List<**byte**>();  
            //在前端加上第一个数组的长度值 这样今后可以根据这个值分别取出来两个数组  
            list.**Add**((**byte**)b1.Length);  
            list.**AddRange**(b1);  
            list.**AddRange**(b2);  
            **byte**[] b = list.**ToArray**<**byte**>();  
            return Convert.**ToBase64String**(b);  
        }  
  
        /// <**summary**>  
        /// 解析密匙  
        /// <**/summary**>  
        /// <**param** **name="key"**>密匙<**/param**>  
        /// <**param** **name="b1"**>RSA的相应参数1<**/param**>  
        /// <**param** **name="b2"**>RSA的相应参数2<**/param**>  
        **private** static void **ResolveKey**(string key, **out** **byte**[] b1, **out** **byte**[] b2)  
        {  
            //从base64字符串 解析成原来的字节数组  
            **byte**[] b = Convert.**FromBase64String**(key);  
            //初始化参数的数组长度  
            b1=**new** **byte**[b[0]];  
            b2=**new** **byte**[b.Length-b[0]-1];  
            //将相应位置是值放进相应的数组  
            **for** (**int** n = 1, i = 0, j = 0; n < b.Length; n++)  
            {  
                **if** (n <= b[0])  
                {  
                    b1[i++] = b[n];  
                }  
                **else** {  
                    b2[j++] = b[n];  
                }  
            }  
        }  
        #**endregion**  
  
        #**region** 字符串加密解密 公开方法  
        /// <**summary**>  
        /// 字符串加密  
        /// <**/summary**>  
        /// <**param** **name="source"**>源字符串 明文<**/param**>  
        /// <**param** **name="key"**>密匙<**/param**>  
        /// <**returns**>加密遇到错误将会返回原字符串<**/returns**>  
        **public** static string **EncryptString**(string source,string key=*publicKey*)  
        {  
            string encryptString = string.*Empty*;  
            **byte**[] d;  
            **byte**[] n;  
            **try**  
            {  
                **if** (!**CheckSourceValidate**(source))  
                {  
                    **throw** **new** Exception("source string too long");  
                }  
                //解析这个密钥  
                **ResolveKey**(key, **out** d, **out** n);  
                BigInteger biN = **new** BigInteger(n);  
                BigInteger biD = **new** BigInteger(d);  
                encryptString= **EncryptString**(source, biD, biN);  
            }  
            **catch**  
            {  
                encryptString = source;  
            }  
            return encryptString;  
        }  
  
        /// <**summary**>  
        /// 字符串解密  
        /// <**/summary**>  
        /// <**param** **name="encryptString"**>密文<**/param**>  
        /// <**param** **name="key"**>密钥<**/param**>  
        /// <**returns**>遇到解密失败将会返回原字符串<**/returns**>  
        **public** static string **DecryptString**(string encryptString, string key=*publicKey*)  
        {  
            string source = string.*Empty*;  
            **byte**[] e;  
            **byte**[] n;  
            **try**  
            {  
                //解析这个密钥  
                **ResolveKey**(key, **out** e, **out** n);  
                BigInteger biE = **new** BigInteger(e);  
                BigInteger biN = **new** BigInteger(n);  
                source = **DecryptString**(encryptString, biE, biN);  
            }  
            **catch** {  
                source = encryptString;  
            }  
            return source;  
        }  
        #**endregion**  
  
        #**region** 字符串加密解密 私有  实现加解密的实现方法  
        /// <**summary**>  
        /// 用指定的密匙加密   
        /// <**/summary**>  
        /// <**param** **name="source"**>明文<**/param**>  
        /// <**param** **name="d"**>可以是RSACryptoServiceProvider生成的D<**/param**>  
        /// <**param** **name="n"**>可以是RSACryptoServiceProvider生成的Modulus<**/param**>  
        /// <**returns**>返回密文<**/returns**>  
        **private** static string **EncryptString**(string source, BigInteger d, BigInteger n)  
        {  
            **int** len = source.Length;  
            **int** len1 = 0;  
            **int** blockLen = 0;  
            **if** ((len % 128) == 0)  
                len1 = len / 128;  
            **else**  
                len1 = len / 128 + 1;  
            string block = "";  
            StringBuilder result = **new** StringBuilder();  
            **for** (**int** i = 0; i < len1; i++)  
            {  
                **if** (len >= 128)  
                    blockLen = 128;  
                **else**  
                    blockLen = len;  
                block = source.**Substring**(i \* 128, blockLen);  
                **byte**[] oText = System.Text.Encoding.Default.**GetBytes**(block);  
                BigInteger biText = **new** BigInteger(oText);  
                BigInteger biEnText = biText.**modPow**(d, n);  
                string temp = biEnText.**ToHexString**();  
                result.**Append**(temp).**Append**("@");  
                len -= blockLen;  
            }  
            return result.**ToString**().**TrimEnd**('@');  
        }  
  
        /// <**summary**>  
        /// 用指定的密匙加密   
        /// <**/summary**>  
        /// <**param** **name="source"**>密文<**/param**>  
        /// <**param** **name="e"**>可以是RSACryptoServiceProvider生成的Exponent<**/param**>  
        /// <**param** **name="n"**>可以是RSACryptoServiceProvider生成的Modulus<**/param**>  
        /// <**returns**>返回明文<**/returns**>  
        **private** static string **DecryptString**(string encryptString, BigInteger e, BigInteger n)  
        {  
            StringBuilder result = **new** StringBuilder();  
            string[] strarr1 = encryptString.**Split**(**new** **char**[] { '@' }, **StringSplitOptions**.*RemoveEmptyEntries*);  
            **for** (**int** i = 0; i < strarr1.Length; i++)  
            {  
                string block = strarr1[i];  
                BigInteger biText = **new** BigInteger(block, 16);  
                BigInteger biEnText = biText.**modPow**(e, n);  
                string temp = System.Text.Encoding.Default.**GetString**(biEnText.**getBytes**());  
                result.**Append**(temp);  
            }  
            return result.**ToString**();  
        }  
        #**endregion**  
    }  
}

# 授权注册模块

## 注册管理程序界面代码

<Window x:Class="SoftRegister.WindowMain"  
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"  
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"  
    Title="网络传销会员层级分析系统——软件授权注册工具"   
    WindowStartupLocation="CenterScreen" ResizeMode="NoResize"  
    Height="300" Width="500">  
    <Window.Background>  
        <LinearGradientBrush EndPoint="0.5,1" StartPoint="0.5,0">  
            <GradientStop Color="LightGreen" Offset="0" />  
            <GradientStop Color="White" Offset="0.8" />  
            <GradientStop Color="LightCoral" Offset="1" />  
        </LinearGradientBrush>  
    </Window.Background>  
    <Grid Margin="10">  
        <Grid.RowDefinitions>  
            <RowDefinition/>  
            <RowDefinition/>  
            <RowDefinition Height="Auto"/>  
            <RowDefinition/>  
            <RowDefinition Height="Auto"/>  
        </Grid.RowDefinitions>  
        <Grid.ColumnDefinitions>  
            <ColumnDefinition/>  
            <ColumnDefinition Width="Auto"/>  
        </Grid.ColumnDefinitions>  
        <TextBlock Grid.ColumnSpan="2" Foreground="LightCyan" FontWeight="Bold" FontSize="22" Margin="10">  
            <TextBlock.Effect>  
                <DropShadowEffect Opacity="0.6" />  
            </TextBlock.Effect>  
            <Run Text="网络传销会员层级分析系统" />  
        </TextBlock>  
        <TextBlock Grid.Row="1" FontSize="18" Margin="15, 5, 0, 5" Foreground="RoyalBlue">  
            <Run Text="软件授权注册工具 v5.13" />  
        </TextBlock>  
          
        <Button Grid.Row="1" Grid.Column="1" Margin="10" Padding="3" Content="查看注册流程帮助" Click="ButtonHelp\_Click"/>  
          
        <GroupBox Grid.Row="2" Header="待注册文件（\*.reginfo）存放路径">  
            <TextBox x:Name="txtRegInfo" IsReadOnly="True"/>  
        </GroupBox>  
        <Button Grid.Row="2" Grid.Column="1" Margin="10" Padding="5" x:Name="btnBrowser"   
                Content="导入待注册文件" Click="BtnBrowser\_Click"/>  
          
        <TextBlock x:Name="txtRegMsg" Grid.Row="3" Grid.ColumnSpan="2" VerticalAlignment="Center"   
                   TextWrapping="WrapWithOverflow" FontSize="15" Foreground="Red" Margin="5"  
                   Text="请先导入客户发过来的注册信息文件(\*.reginfo)，然后再生成注册密钥文件(\*.regkey)，发送回给客户供用户注册。"/>  
      
        <GroupBox Grid.Row="4" Header="注册密钥文件（\*.regkey）生成路径">  
            <TextBox x:Name="txtRegKey" IsReadOnly="True" Background="LightGray"/>  
        </GroupBox>  
        <Button Grid.Row="4" Grid.Column="1" Margin="10" Padding="5" IsEnabled="False"   
                x:Name="btnOK" Content="生成注册密钥文件" Click="BtnOK\_Click"/>  
    </Grid>  
</Window>

## 注册管理程序生成密钥代码

**namespace** SoftRegister  
{  
    /// <**summary**>  
    /// Interaction logic for Window1.xaml  
    /// <**/summary**>  
    **public** partial class WindowMain : Window  
    {  
        **private** string *cpu*, *disk*, *com*, *usr*;  
        **private** string *privateKey* = "gBwg95CF15fq/kBiXqSCr0s/imffU=";  
        **public** WindowMain()  
        {  
            **InitializeComponent**();  
            RegConfig.**Init**();  
        }  
          
        **private** void **BtnBrowser\_Click**(object sender, RoutedEventArgs e)  
        {  
            OpenFileDialog openfileDlg = **new** OpenFileDialog();  
            openfileDlg.Title = "选择需要进行注册的用户信息文件(\*.reginfo)";  
            openfileDlg.Filter = "注册信息文件|\*.reginfo";  
            **if** (openfileDlg.**ShowDialog**() == **true**)  
            {  
                *txtRegInfo*.Text = openfileDlg.FileName;  
                **if**(**DecryptRegInfo**(openfileDlg.FileName))  
                {  
                    *txtRegMsg*.Text = string.**Format**("授权给（公司/单位：{0}，用户：{1}）", *com*, *usr*);  
                    *txtRegKey*.Text = openfileDlg.FileName.**Replace**(".reginfo", ".regkey");  
                    *btnOK*.IsEnabled = **true**;  
                }  
                **else**  
                {  
                    *txtRegMsg*.Text = "注册信息文件不正确！";  
                    *txtRegKey*.Text = "";  
                    *btnOK*.IsEnabled = **false**;  
                }  
            }  
        }  
          
        //对注册信息文件进行解密  
        **private** **bool** **DecryptRegInfo**(string file)  
        {  
            string regMsg = EncryptHelper.**FileDecrypt**(file);  
            **int** subIndex = 1;  
            *disk* = regMsg.**Substring**(0, subIndex);  
            **while**(**ContainsTimes**(regMsg, *disk*) >= 4)  
            {  
                subIndex++;  
                *disk* = regMsg.**Substring**(0, subIndex);  
            }  
            *disk* = regMsg.**Substring**(0, subIndex-1);  
            **if**(**ContainsTimes**(regMsg, *disk*) == 4)  
            {  
                string[] regList = regMsg.**Split**(**new** String[]{*disk*}, **StringSplitOptions**.*RemoveEmptyEntries*);  
                *cpu* = RSAHelper.**DecryptString**(regList[0], *privateKey*);  
                *com* = RSAHelper.**DecryptString**(regList[1], *privateKey*);  
                *usr* = RSAHelper.**DecryptString**(regList[2], *privateKey*);  
                return **true**;  
            }  
            return **false**;  
        }  
  
        //判断字符串包含次数  
        **private** **int** **ContainsTimes**(string MainStr, string subStr)  
        {  
            Regex ex = **new** Regex(subStr);  
            return ex.**Matches**(MainStr).Count;  
        }  
  
        //生成注册密钥文件  
        **private** void **BtnOK\_Click**(object sender, RoutedEventArgs e)  
        {  
            **try**   
            {  
                //加密处理生成密钥文件\*.regkey  
                List<string> regList = **new** List<string>();  
                regList.**Add**(RSAHelper.**EncryptString**(*cpu*+*disk*, *privateKey*));  
                regList.**Add**(RSAHelper.**EncryptString**(*com*, *privateKey*));  
                regList.**Add**(RSAHelper.**EncryptString**(*usr*, *privateKey*));  
                **foreach** (string confKey **in** RegConfig.*config*.Keys)   
                {  
                    string key = RSAHelper.**EncryptString**(confKey, *privateKey*);  
                    string val = RSAHelper.**EncryptString**(RegConfig.*config*[confKey], *privateKey*);  
                    regList.**Add**(key + *disk* + val);  
                }  
                string regMsg = string.**Join**(*cpu*, regList);  
                EncryptHelper.**FileEncrypt**(*txtRegKey*.Text, regMsg);  
                MessageBox.**Show**("生成注册密钥文件成功！\n");  
            }   
            **catch** (Exception ex)  
            {  
                MessageBox.**Show**("生成注册密钥文件失败！\n"+ex.Message);  
            }  
        }  
        **private** void **ButtonHelp\_Click**(object sender, RoutedEventArgs e)  
        {  
            WindowHelp winHelp = **new** WindowHelp();  
            winHelp.**ShowDialog**();  
        }  
    }  
}

## 注册界面代码

<UserControl x:Class="MemberTree.SoftRegView"  
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"  
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"   
    Height="330" Width="600">  
    <UserControl.Background>  
        <LinearGradientBrush EndPoint="0.5,1" StartPoint="0.5,0">  
            <GradientStop Color="LightGreen" Offset="0" />  
            <GradientStop Color="White" Offset="0.8" />  
            <GradientStop Color="LightCoral" Offset="1" />  
        </LinearGradientBrush>  
    </UserControl.Background>  
    <Grid Margin="10">  
        <Grid.RowDefinitions>  
            <RowDefinition/>  
            <RowDefinition/>  
            <RowDefinition Height="2\*"/>  
        </Grid.RowDefinitions>  
        <TextBlock Foreground="LightCyan" FontWeight="Bold" FontSize="26" Margin="15, 25,15,5">  
            <TextBlock.Effect>  
                <DropShadowEffect Opacity="0.6" />  
            </TextBlock.Effect>  
            <Run Text="网络传销会员层级分析系统" />  
        </TextBlock>  
        <TextBlock Grid.Row="1" Margin="15, 10" FontSize="18" Foreground="Red">  
            <Run Text="软件尚未注册授权，请联系版权方获取授权密钥文件!" />  
        </TextBlock>  
          
        <TabControl Grid.Row="2" FontSize="14">  
            <TabItem Header="步骤一：我没有注册密钥文件">  
                <Grid Margin="10">  
                    <Grid.RowDefinitions>  
                        <RowDefinition/>  
                        <RowDefinition/>  
                        <RowDefinition/>  
                    </Grid.RowDefinitions>  
                    <Grid.ColumnDefinitions>  
                        <ColumnDefinition Width="Auto"/>  
                        <ColumnDefinition/>  
                        <ColumnDefinition Width="Auto"/>  
                    </Grid.ColumnDefinitions>  
                    <TextBlock Text="公司/单位：" VerticalAlignment="Center"/>  
                    <TextBox Grid.Column="1" Margin="5" x:Name="txtxCom"/>  
                    <TextBlock Grid.Row="1" Text="用户姓名：" VerticalAlignment="Center"/>  
                    <TextBox Grid.Row="1" Grid.Column="1" Margin="5" x:Name="txtxUsr"/>  
                    <Button Grid.RowSpan="2" Grid.Column="2" Margin="5,15" Padding="5" x:Name="btnRegInfo"   
                            Content="生成注册信息文件" Click="BtnRegInfo\_Click"/>  
                      
                    <TextBlock Grid.Row="3" Grid.ColumnSpan="3" Foreground="RoyalBlue" TextWrapping="WrapWithOverflow"  
                               Text="请填写上面的注册信息，然后生成待注册信息文件(\*.reginfo); 将待注册信息文件发送给软件版权方，获取授权密钥文件（\*.regkey），进行注册。"/>      
                </Grid>  
                  
            </TabItem>  
            <TabItem Header="步骤二：我已有授权密钥文件">  
                <Grid>  
                    <Grid.RowDefinitions>  
                        <RowDefinition/>  
                        <RowDefinition/>  
                    </Grid.RowDefinitions>  
                    <Grid.ColumnDefinitions>  
                        <ColumnDefinition Width="Auto"/>  
                        <ColumnDefinition />  
                        <ColumnDefinition Width="Auto"/>  
                    </Grid.ColumnDefinitions>  
                    <TextBlock Text="密钥文件：" VerticalAlignment="Center"/>  
                    <TextBox Grid.Column="1" Margin="10,16" x:Name="txtRegKey" IsReadOnly="True"/>  
                    <Button Grid.Column="2" Margin="15" Padding="5" x:Name="btnBrowser"   
                            Content="导入密钥文件" Click="BtnBrowser\_Click"/>  
                      
                    <TextBlock x:Name="txtRegMsg" Grid.Row="3" Grid.ColumnSpan="2" VerticalAlignment="Center"   
                           TextWrapping="WrapWithOverflow" FontSize="15" Foreground="Red" Margin="5"  
                           Text="请导入版权方发过来的注册密钥文件(\*.regkey)，然后再点击“注册”按钮进行注册。" />  
                      
                    <Button Grid.Row="1" Grid.Column="2" Margin="15" Padding="5" x:Name="btnRegKey"   
                            IsEnabled="False" Content="注  册" Click="BtnRegKey\_Click"/>  
                </Grid>  
            </TabItem>  
        </TabControl>  
    </Grid>  
</UserControl>

## 注册帮助类代码

**namespace** MemberTree  
{  
    **public** class SoftReg  
    {  
        **public** static string *Com* = "";  
        **public** static string *Usr* = "";  
          
        **private** static string *diskStr* = "";  
        **private** static string GetDisk  
        {  
            get{  
                **if**(*diskStr* == "")  
                {  
                    //获取当前系统磁盘符方法1，返回：C:  
                    string sysVolume = Environment.**GetEnvironmentVariable**("systemdrive");  
                    ManagementClass mc = **new** ManagementClass("Win32\_NetworkAdapterConfiguration");  
                    ManagementObject disk = **new** ManagementObject("win32\_logicaldisk.deviceid=\""+ sysVolume +"\"");  
                    disk.**Get**();  
                    *diskStr* = disk.**GetPropertyValue**("VolumeSerialNumber").**ToString**();  
                }  
                return *diskStr*;  
            }  
        }  
  
        **private** static string *cpuStr* = "";  
        **private** static string getCpu  
        {  
            get{  
                **if**(*cpuStr* == "")  
                {  
                    ManagementClass myCpu = **new** ManagementClass("win32\_Processor");  
                    ManagementObjectCollection myCpuConnection = myCpu.**GetInstances**();  
                    **foreach** (ManagementObject myObject **in** myCpuConnection)  
                    {  
                        *cpuStr* = myObject.Properties["Processorid"].Value.**ToString**();  
                        break;  
                    }  
                }  
                return *cpuStr*;  
            }  
        }  
  
        //生成机器码  
        **public** static string **getMNum**()  
        {  
            string strNum = getCpu + GetDisk;//机器码=Cpu序列号+硬盘序列号  
            return strNum;  
        }  
          
        //生成注册信息文件  
        **public** static void **getRegInfo**(string filePath, string com, string usr)  
        {  
            **try**   
            {  
                string cpu = RSAHelper.**EncryptString**(getCpu);  
                com = RSAHelper.**EncryptString**(com);  
                usr = RSAHelper.**EncryptString**(usr);  
                string regMsg = GetDisk + cpu + GetDisk + com + GetDisk + usr + GetDisk;  
                EncryptHelper.**FileEncrypt**(filePath, regMsg);  
                MessageBox.**Show**("生成注册信息文件成功！\n");  
            }   
            **catch** (Exception ex)  
            {  
                MessageBox.**Show**("生成注册信息文件失败！\n"+ex.Message);  
            }  
        }  
          
        //对注册密钥文件进行解密  
        **public** static **bool** **DecryptRegKey**(string file)  
        {  
            **try**   
            {  
                string regMsg = EncryptHelper.**FileDecrypt**(file);  
                string[] regList = regMsg.**Split**(**new** String[]{getCpu}, **StringSplitOptions**.*RemoveEmptyEntries*);  
                string mNum = RSAHelper.**DecryptString**(regList[0]);  
                **if**(mNum == **getMNum**())  
                {  
                    *Com* = RSAHelper.**DecryptString**(regList[1]);  
                    *Usr* = RSAHelper.**DecryptString**(regList[2]);  
                    return **true**;  
                }  
                **else**  
                {  
                    MessageBox.**Show**("注册密钥机器码不正确！");  
                    return **false**;  
                }  
            }   
            **catch** (Exception ex)  
            {  
                MessageBox.**Show**("解析注册密钥文件失败！\n"+ex.Message);  
                return **false**;  
            }  
        }  
          
        //安装注册密钥  
        **public** static **bool** **InstallRegKey**(string file)  
        {  
            **try**   
            {  
                string regMsg = EncryptHelper.**FileDecrypt**(file);  
                string[] regList = regMsg.**Split**(**new** String[]{getCpu}, **StringSplitOptions**.*RemoveEmptyEntries*);  
                RegistryKey retkey = Registry.*CurrentUser*.**OpenSubKey**("software", **true**).**CreateSubKey**("MemTree");  
                retkey.**SetValue**("MNum", regList[0]);  
                retkey.**SetValue**("Com", regList[1]);  
                retkey.**SetValue**("Usr", regList[2]);  
                **for** (**int** i = 3; i < regList.Length; i++)   
                {  
                    string[] retItems = regList[i].**Split**(**new** String[]{GetDisk}, **StringSplitOptions**.*RemoveEmptyEntries*);  
                    RegistryKey retkeySql = retkey.**CreateSubKey**("SQL");  
                    retkeySql.**SetValue**(retItems[0], retItems[1]);  
                }  
                return **true**;  
            }   
            **catch** (Exception ex)  
            {  
                MessageBox.**Show**("安装注册密钥失败！\n"+ex.Message);  
                return **false**;  
            }  
        }  
          
        //判断是否已经注册  
        **public** static **bool** **hasReged**()  
        {  
            **try**   
            {  
                RegistryKey retkey = Registry.*CurrentUser*.**OpenSubKey**("software").**OpenSubKey**("MemTree");  
                **if**(retkey != **null**)  
                {  
                    string mNum = retkey.**GetValue**("MNum").**ToString**();  
                    string com = retkey.**GetValue**("Com").**ToString**();  
                    string usr = retkey.**GetValue**("Usr").**ToString**();  
                    **if**(mNum!=**null** && com!=**null** && usr!=**null**)  
                    {  
                        *Com* = RSAHelper.**DecryptString**(com);  
                        *Usr* = RSAHelper.**DecryptString**(usr);  
                        mNum = RSAHelper.**DecryptString**(mNum);  
                        **if**(mNum == **getMNum**())  
                        {  
                            RegistryKey retkeySql = retkey.**OpenSubKey**("SQL");  
                            **if**(retkeySql!=**null**)  
                            {  
                                **if**(RegConfig.**InitConfig**(retkeySql))  
                                {  
                                    return **true**;  
                                }  
                            }  
                        }  
                        **else**  
                        {  
                            MessageBox.**Show**("注册码不正确！");  
                        }  
                    }  
                }  
            }   
            **catch** (Exception ex)  
            {  
                MessageBox.**Show**("读取注册信息失败！\n"+ex.Message);  
            }  
              
            return **false**;  
        }  
  
        //----------------------------------------------------------------------------------------------------------------  
        **public** **int**[] *intCode* = **new** **int**[127];//存储密钥  
        **public** **int**[] *intNumber* = **new** **int**[25];//存机器码的Ascii值  
        **public** **char**[] *Charcode* = **new** **char**[25];//存储机器码字  
  
        **public** void **setIntCode**()//给数组赋值小于10的数  
        {  
            **for** (**int** i = 1; i < *intCode*.Length; i++)  
            {  
                *intCode*[i] = i % 9;  
            }  
        }  
  
        //生成注册码  
        **public** string **getRNum**(string str)  
        {  
            **setIntCode**();//初始化127位数组  
            **for** (**int** i = 1; i < *Charcode*.Length; i++)//把机器码存入数组中  
            {  
                *Charcode*[i] = Convert.**ToChar**(str.**Substring**(i - 1, 1));  
            }  
            **for** (**int** j = 1; j < *intNumber*.Length; j++)//把字符的ASCII值存入一个整数组中。  
            {  
                *intNumber*[j] = *intCode*[Convert.**ToInt32**(*Charcode*[j])] + Convert.**ToInt32**(*Charcode*[j]);  
            }  
            string strAsciiName = "";//用于存储注册码  
            **for** (**int** j = 1; j < *intNumber*.Length; j++)  
            {  
                **if** (*intNumber*[j] >= 48 && *intNumber*[j] <= 57)//判断字符ASCII值是否0－9之间  
                {  
                    strAsciiName += Convert.**ToChar**(*intNumber*[j]).**ToString**();  
                }  
                **else** **if** (*intNumber*[j] >= 65 && *intNumber*[j] <= 90)//判断字符ASCII值是否A－Z之间  
                {  
                    strAsciiName += Convert.**ToChar**(*intNumber*[j]).**ToString**();  
                }  
                **else** **if** (*intNumber*[j] >= 97 && *intNumber*[j] <= 122)//判断字符ASCII值是否a－z之间  
                {  
                    strAsciiName += Convert.**ToChar**(*intNumber*[j]).**ToString**();  
                }  
                **else**//判断字符ASCII值不在以上范围内  
                {  
                    **if** (*intNumber*[j] > 122)//判断字符ASCII值是否大于z  
                    {  
                        strAsciiName += Convert.**ToChar**(*intNumber*[j] - 10).**ToString**();  
                    }  
                    **else**  
                    {  
                        strAsciiName += Convert.**ToChar**(*intNumber*[j] - 9).**ToString**();  
                    }  
                }  
            }  
            return strAsciiName;  
        }  
    }  
}