昆明光标科技有限公司

光标应用开发平台的技术特点

一、系统结构

系统结构严格按照三层结构进行设计和开发，整体的层次结构可参照**使用XPO开发时可以参考使用的架构**，只不过使用的技术已经大不相同。

实体层使用 ORM 组件来时行数据映射，使用的技术是 Castle ActiveRecord，其运行效率和性能较好，接近 Ado.net.

实体控制层和实体层合二为一，原因是实体层使用的是 CastleActiveRecord，根据其技术特点进行的调整。

界面表现层采用Visual WebGui,它的特点其官方网站已说得很清楚，也可查看**Visual WebGui 技术文档中文版下载**进行具体的了解，但不管它有什么特点，但对我公司来说，当时我们主要考滤以下几点：

1、 由于我公司原先是做 CS 应用的，我们有丰富的 WinForm 开发经验，这些经验可以丰常直接地过渡到 Web 开发上。

2、 我们做信息管理系统的主要关注的是我们提交到客户的软件是否完成了客户的需求，以及操作是否简单易用，

而我们为了速度，往往达不到简单易用的要求，但 Visual WebGui 满足了我们的需要，让程序员不必纠缠在 Html 代码的海洋里，我们需要的是可以完成业务的模块或软件，不希望得到的是技术玄乎的 JS 脚本特效等的实现，而没有正确满足客户的业务需求。

3、 如果我们够细致，我们可以发现，同样是相同的代码，只需改变名称空间，我们可以得到 Winform 应用，也可以得到 Web 应用，也就是说，我们只要在设计上细心点，就可以同时生成两种应用程序。

另外，系统以接口组织形式来组织，有严格的边界和依赖划分，各组件的装配使用了 CastleIoc 容器来进行装配，像日志追踪及一些业务逻辑过程控制我们使用 AOP 方面的技术，同样是基于 CastleAspectSharp。但你若愿意深究，我们对 Castle 的依赖也是可替换的。

另外，程序员和架构师的水平可能是有差别的，一般我们希望软件开发能像工业生产一样的流水线化，更加高效，降低对程序员的要求，让一般经过基本培训的人就可以开发，而不需要深究比如说：日志跟踪、性能分析、用户认证与授权等问题，我们只希望程序员只开发业务逻辑相关的代码，不涉及更多的内容，这些功能不希望普通程序员也要了解，延长学习曲线，增加企业成本。只要他们将业务逻辑的代码提交就自动装配到系统中，将极

大地降低企业的开发成本，另一方面，由于程序员需要关注的方面少了，也就更容易提交高质量的软件代码。

昆明光标科技的软件开发平台框架正是解决了以上困扰我们多年的问题。当然，不同的软件开发企业的需求可能是有差异的，以上所述也许并不是你所需的，当这些问题确实是我们要解决的问题。

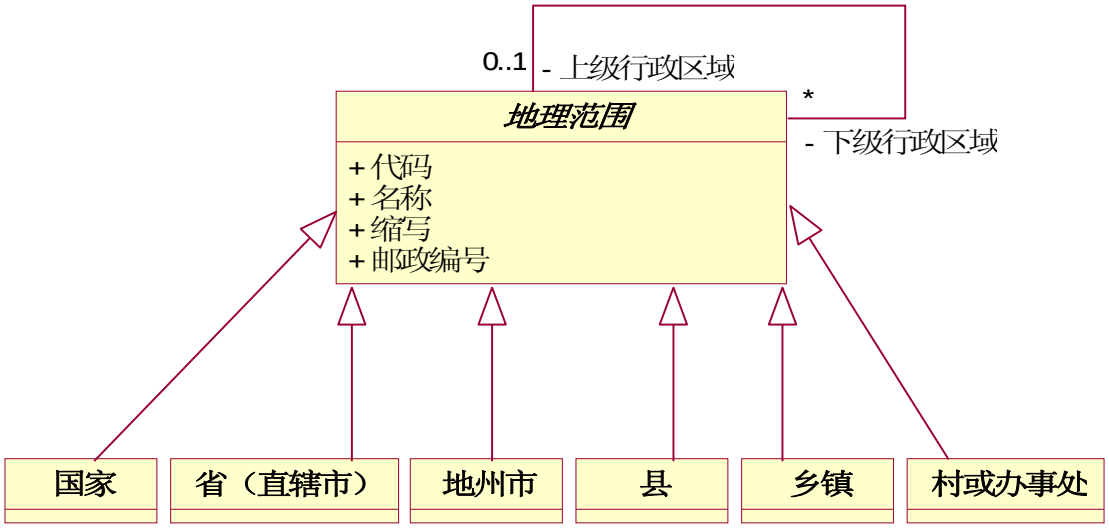
总而言之，我们希望达到的目标是：**一次开发，到处重复使用，按需定制**。

二、编码规范

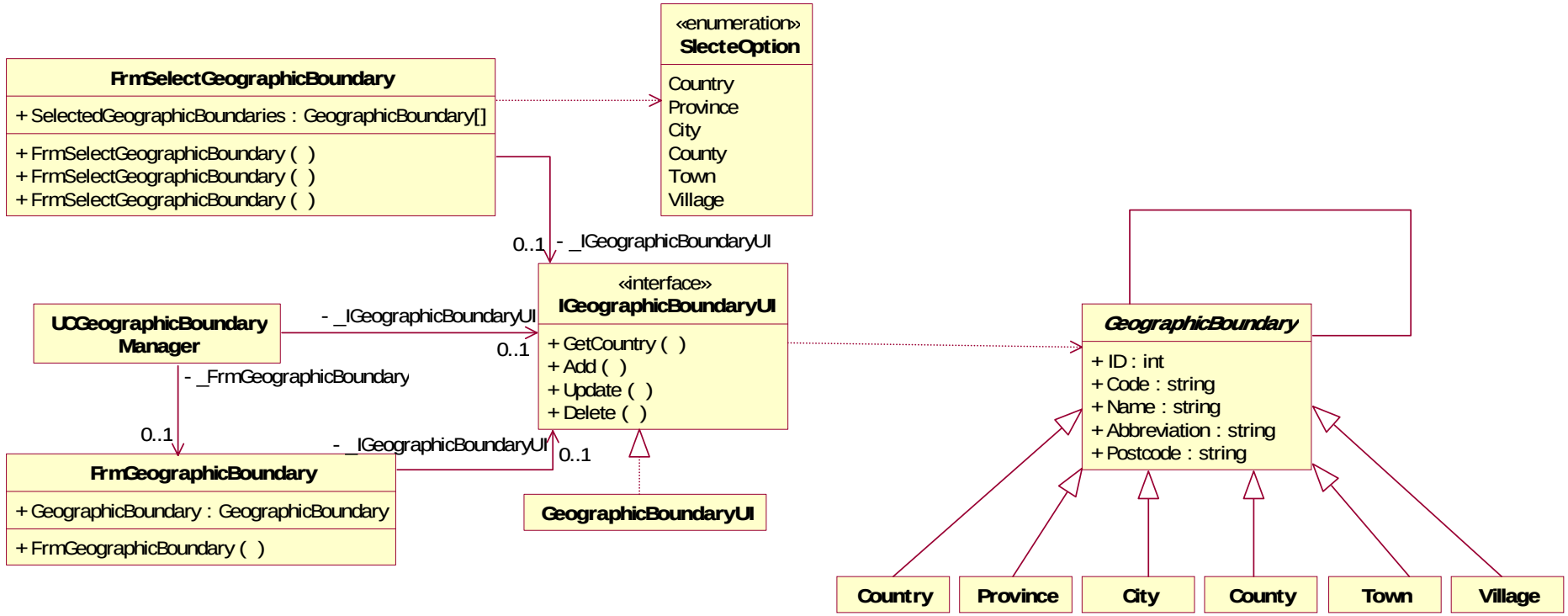
略，如有需要，我传一份文档。

三、结构示例

1、需求分析示例：



2、设计示例：


«enumeration»
SlecteOption
Country
Province
City
County
Town
Village
FrmSelectGeographicBoundary
+ SelectedGeographicBoundaries : GeographicBoundary[]
+ FrmSelectGeographicBoundary ( )
+ FrmSelectGeographicBoundary ( )
+ FrmSelectGeographicBoundary ( )
0..1
- _IGeographicBoundaryUI
«interface»
IGeographicBoundaryUI
+ GetCountry ( )
+ Add ( )
+ Update ( )
+ Delete ( )
UCGeographicBoundary Manager
- _IGeographicBoundaryUI
0..1
- _FrmGeographicBoundary
0..1
- _IGeographicBoundaryUI
0..1
FrmGeographicBoundary
+ GeographicBoundary : GeographicBoundary
+ FrmGeographicBoundary ( )
GeographicBoundaryUI
GeographicBoundary
+ ID : int
+ Code : string
+ Name : string
+ Abbreviation : string
+ Postcode : string
Country
Province
City
County
Town
Village

3、实体层实现：

```
/// <summary>
/// 地理范围
/// </summary>
[Serializable()]
[ActiveRecord("GeographicBoundary"),JoinedBase]
public abstract class GeographicBoundary : ActiveRecordBase<GeographicBoundary>
{
        ========== 构造函数 ==========
    public GeographicBoundary()
        : base()...

        ========== 公有属性 ==========
        ========== 编号 ==========
    private int _ID;
    [PrimaryKey(PrimaryKeyType.Identity, "ID")]
    public int ID...

    地理编号（或代码）
        ========== 名称 ==========
    private string _Name;
    /// <summary> ...
    [Property]
    public string Name...

    缩写
    邮政编号

    全称

    对象属性
```

```
/// <summary>
/// 国家
/// </summary>
[Serializable()]
[ActiveRecord("Country")]
public class Country : GeographicBoundary
{
        ======== 构造函数 ========
    public Country()
        : base()...

        ======== 属性 ========
        ======== CountryID ========
    private int _CountryID;

    [JoinedKey("Country_Id")]
    public int CountryID...

        ======== 公有静态方法 ========
    /// <summary>
    /// 取得所有国家
    /// </summary>
    /// <returns>国家列表</returns>
    public static Country[] FindAllCountry()...

}

/// <summary> ...
```

4、业务逻辑层及其实现：

```
public interface IGeographicBoundaryUI
{
    查询
    /// <summary>
    /// 查询国家
    /// </summary>
    /// <returns>国家列表</returns>
    Country[] GetCountry();

    业务逻辑
    /// <summary>
    /// 添加地理范围
    /// </summary>
    /// <param name="parent">上级地理范围</param>
    /// <param name="code">地理编号（或代码）</param>
    /// <param name="name">名称</param>
    /// <param name="abbreviation">缩写</param>
    /// <param name="postcode">邮政编号</param>
    /// <returns>地理范围</returns>
    GeographicBoundary Add(GeographicBoundary parent, string code, string name, string abbreviation, string postcode);
    /// <summary>
    /// 更新地理范围
    /// </summary>
    /// <param name="geographicBoundary">地理范围</param>
    /// <param name="code">地理编号（或代码）</param>
    /// <param name="name">名称</param>
    /// <param name="abbreviation">缩写</param>
    /// <param name="postcode">邮政编号</param>
    /// <returns>true 成功, false 失败</returns>
    bool Update(GeographicBoundary geographicBoundary, string code, string name, string abbreviation, string postcode);
    /// <summary>
```

```
public class GeographicBoundaryUI : IGeographicBoundaryUI
{
    私有变量

    构造函数

    查询

    ———— 业务逻辑 ————————————————————————————————————————————
    /// <summary>
    /// 添加地理范围
    /// </summary>
    /// <param name="parent">上级地理范围</param>
    /// <param name="code">地理编号（或代码）</param>
    /// <param name="name">名称</param>
    /// <param name="abbreviation">缩写</param>
    /// <param name="postcode">邮政编号</param>
    /// <returns>地理范围</returns>
    [IBeamSecurityAttribute(SecurityAction.Demand, FunctionID = ACPLID.FunUpdateGeographicBoundary,
        Name = "编辑地理范围", Catalog = "系统管理.地理范围", Description = "赋于用户编辑地理范围的权限")]
    public GeographicBoundary Add(GeographicBoundary parent, string code,
        string name, string abbreviation, string postcode)
    {
        数据校验

        业务逻辑
    }
    /// <summary>
    /// 更新地理范围
    /// </summary>
    /// <param name="geographicBoundary">地理范围</param>
    /// <param name="code">地理编号（或代码）</param>
```

请注意，此属性标签注册了功能的定义，并在系统执行到此方法前时进行权限验证，并且和.NetFramework紧密集成。

```
public bool Update(GeographicBoundary geographicBoundary, string code,
    string name, string abbreviation, string postcode)
{
    ———— 数据校验 ————————————————————————————————————————
    string mes = "更新地理范围时";
    if (geographicBoundary == null)
    {
        throw new ArgumentNullException(mes + "地理范围对象为空！");
    }
    if (!GeographicBoundary.Exists(geographicBoundary))
    {
        throw new ArgumentNullException(mes + "地理范围对象并不存在！");
    }
    if (name == null || name == string.Empty)
    {
        throw new ArgumentException(mes + "名称为空！");
    }
    ————————————————————————————————————————————————————

    业务逻辑
}
```

```
public bool Update(GeographicBoundary geographicBoundary, string code,
    string name, string abbreviation, string postcode)
{
    数据校验

    ———— 业务逻辑 ————————————————————————————————————————————
    using (TransactionScope trans = new TransactionScope())
    {
        try
        {
            geographicBoundary.Code = code;
            geographicBoundary.Name = name;
            geographicBoundary.Abbreviation = abbreviation;
            geographicBoundary.Postcode = postcode;
            geographicBoundary.Save();

            trans.VoteCommit();

            return true; ;
        }
        catch (Exception ee)
        {
            trans.VoteRollBack();
            this.logger.Error(string.Format("{0}出错：{1}", mes, ee.ToString()));
            return false;
        }
    }
    ————————————————————————————————————————————————————————
}
```

5、界面层实现：

```
[IBeamUI(Guid = UIConstent.GeographicBoundaryGuid, Catalog = UIConstent.GeographicBoundaryCatalog,
    Name = UIConstent.GeographicBoundaryName, DisplayMode = DisplayMode.DependOnPurview, UIType = UIType.Control,
    Icon = UIConstent.GeographicBoundaryIcon, IsDefault = false, FunctionIDs = UIConstent.GeographicBoundaryFunctionIDs)]
public partial class UCGeographicBoundaryManager : UserControl
{
    ———— 私有变量 ————
    UCGeographicBoundary ucGeographicBoundary;
    GeographicBoundary gb;
    TreeNode selectedTreeNode = null;

    ———— 构造函数 ————
    public UCGeographicBoundaryManager()...

    ———— 私有方法 ————
    void InitControl()...
    void InitControlData()...
    void SetControlEnable()...
    void LoadItemToTreeView(TreeNode parent)...
    void AddTreeNode(TreeNode parent, GeographicBoundary item)...

    ———— 事件处理 ————
    private void trvGeographicBoundary_BeforeExpand(object sender, TreeViewCancelEventArgs e)...
    private void trvGeographicBoundary_AfterSelect(object sender, TreeViewEventArgs e)...
    private void btnAddCountry_Click(object sender, EventArgs e)...
    void frmAddCountry_Closed(object sender, EventArgs e)...
    private void btnAdd_Click(object sender, EventArgs e)...
    void frmAddGeographicBoundary_Closed(object sender, EventArgs e)...
    private void btnUpdate_Click(object sender, EventArgs e)...
    void frmUpdateGeographicBoundary_Closed(object sender, EventArgs e)...
    private void btnDelete_Click(object sender, EventArgs e)...
    void frmDeleteGeographicBoundary_Closed(object sender, EventArgs e)...
```

请注意：此属性标签注册了在系统中出现的界面，和前面功能标签一样，也是系统自动装配的，程序无需关心，只需专注与业务。

清晰严谨的代码结构，是否也是您所需要的？


```
33      void frmDeleteGeographicBoundary_Closed(object sender, EventArgs e)
        {
            MessageBoxy frmDeleteGeographicBoundary = (MessageBoxy)sender;
            if (frmDeleteGeographicBoundary.DialogResult == DialogResult.Yes)
            {
                try
                {
                    IGeographicBoundaryUI iGeographicBoundaryUI = ComponentCreator.Creator[typeof(IGeographicBoundaryUI)] as IGeographicBoundaryUI;
                    if (iGeographicBoundaryUI.Delete(gb))
                    {
                        TreeNode parent = selectedTreeNode.Parent;
                        parent.Nodes.Remove(selectedTreeNode);
                        gb = null;
                        selectedTreeNode = null;
                        ucGeographicBoundary.GeographicBoundary = null;
                        SetControlEnable();
                    }
                    else
                    {
                        MessageBox.Show("地理范围删除失败！");
                    }
                }
                catch (Exception ee)
                {
#if (DEBUG)
                    System.Diagnostics.Debug.WriteLine(ee.ToString());
                    MessageBox.Show(ee.ToString());
#else
                    MessageBox.Show(ee.Message);
#endif
                }
            }
        }
```

6、如何注册功能（当运行到此处时对功能进行权限检查）：

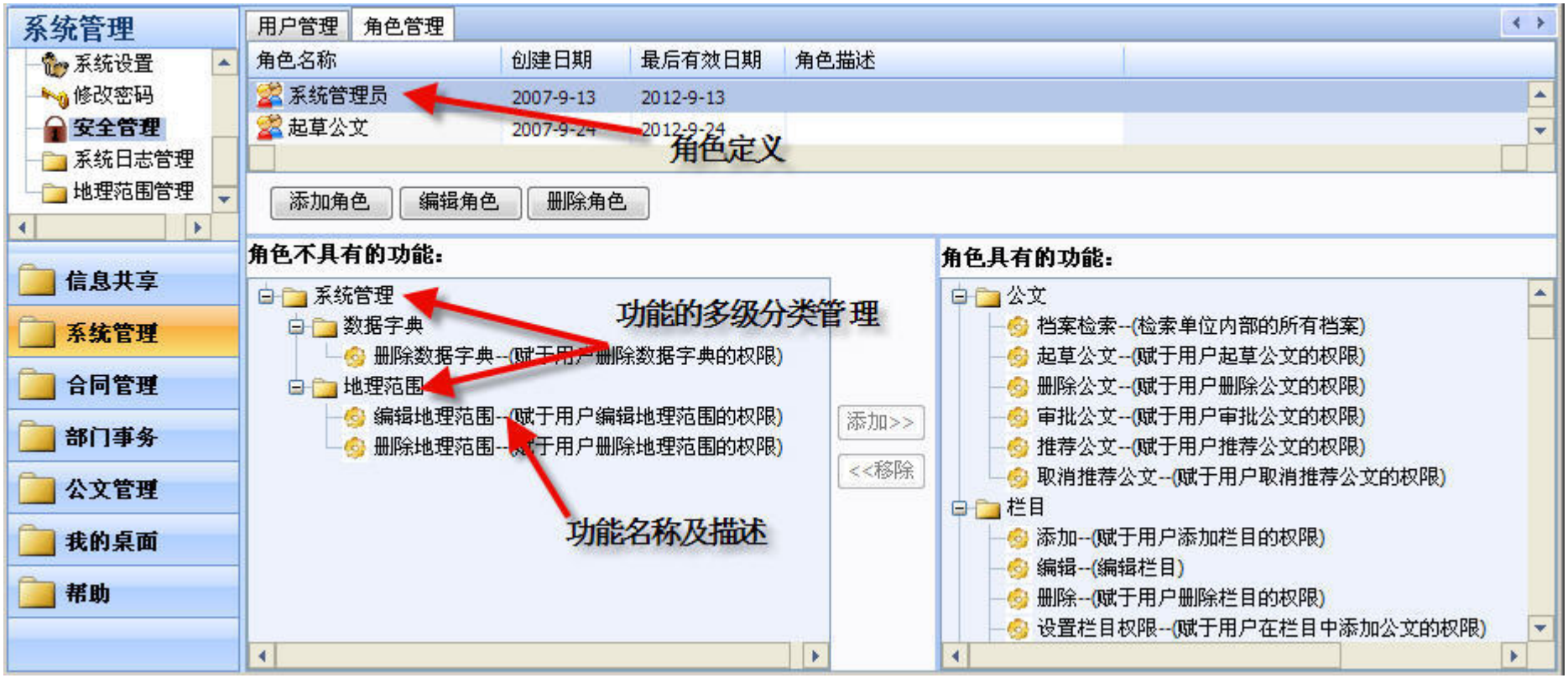
如果方法上没有加注权限检查标签，将不会进行权限检查，不像很多 AOP 实现方法，不管什么方法，都要

扫描一次，消耗资源。这里的权限检查和.NetFreamWork 的安全检查兼容集成。

另外，程序员不必关心系统是如何进行功能注册、用户注册、用户权限审查的，他只需开发业务方面的代码即可，框架处理了所有的这些内容。

程序员需要做的工作就只是在需要进行权限验证的方法加个标签，功能就会自动出现在权限管理中，和系统构成一个统一体。

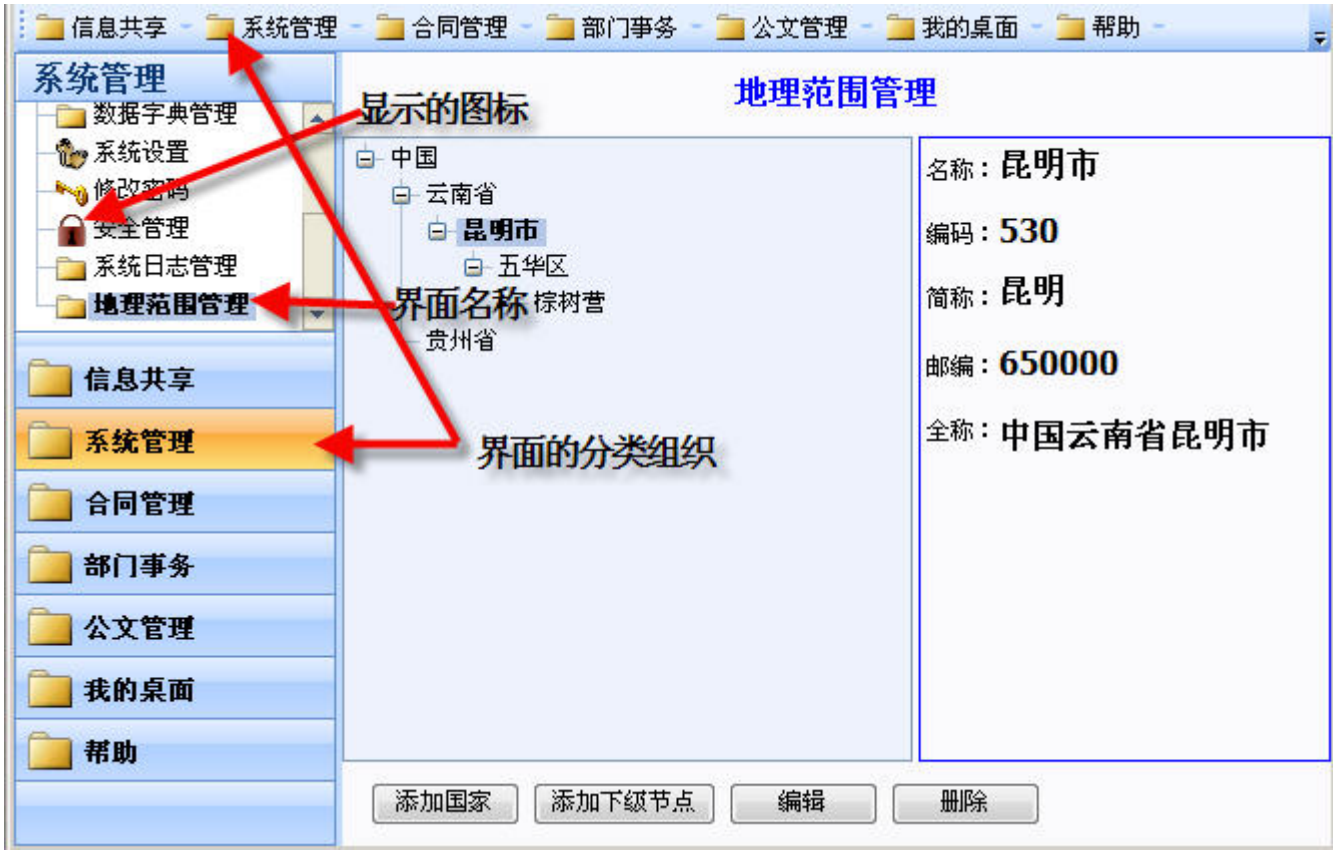
7、对应到权限管理中：


系统管理
系统设置
修改密码
安全管理
系统日志管理
地理范围管理
信息共享
系统管理
合同管理
部门事务
公文管理
我的桌面
帮助
用户管理
角色管理
角色名称
创建日期
最后有效日期
角色描述
系统管理员
2007-9-13
2012-9-13
起草公文
2007-9-24
2012-9-24
角色定义
添加角色
编辑角色
删除角色
角色不具有的功能：
系统管理
数据字典
删除数据字典--(赋于用户删除数据字典的权限)
地理范围
编辑地理范围--(赋于用户编辑地理范围的权限)
删除地理范围--(赋于用户删除地理范围的权限)
功能的多级分类管理
功能名称及描述
添加>>
<<移除
角色具有的功能：
公文
档案检索--(检索单位内部的所有档案)
起草公文--(赋于用户起草公文的权限)
删除公文--(赋于用户删除公文的权限)
审批公文--(赋于用户审批公文的权限)
推荐公文--(赋于用户推荐公文的权限)
取消推荐公文--(赋于用户取消推荐公文的权限)
栏目
添加--(赋于用户添加栏目的权限)
编辑--(编辑栏目)
删除--(赋于用户删除栏目的权限)
设置栏目权限--(赋于用户在栏目中添加公文的权限)

8、如何注册界面（让界面自动组织与装配）：

9、常量定义：

10、对应到注册后的显示：



与功能注册类似，注册界面所需做的工作就只需简单的在界面类中加个标签，系统自动会根据当前用户的权

限组织界面的显示及权限认证。实际上，所有这些，我们没有关注更多的方面，只需把业务实现好了，就一切 OK 了。

四、系统远景目标

我们的平台框架可以即是 Web 平台框架，也可以是 CS 应用平台框架，我们希望发展成应用服务器的容器，达到自动装配应用，而不只限于自动装配组件。

五、源码概览