# PIC 系统框架说明

## 概述：

PIC V2 在上一个版本的基础上增加了多数据和多应用程序支持，并加入PortalServiceProvider用于支持PortalService的灵活实现。VirtualPathProvider用于提供MasterPage的灵活实现。

## 架构：

1. **技术架构**

数据库（MS SQL Server）

操作系统（Windows Server 2003/2008）

Castle、Newtosoft.Json、Postsharp、Aspose

WCF、WF

PIC.Core、PIC.Portal、PIC.Component

业务系统（EPC、考核、OA等）

ExtJs、Silverlight

相比V1版除使用了Castle中的ActiveRecord外，还用了NVelocity用于编码生成。数据导入使用了Aspose组件（操作MS Office的组件，很好用并且用在服务器端，可以考虑在操作MS Office时使用）。

1. **命名空间**

**PIC 的核心（PIC.Core）**

PIC.Portal、PIC.Portal.ServicesProvider、PIC.Portal.Web.VirtualPathProvider

PIC.Core、PIC.Component

PIC.Portal.Services、PIC.Portal.Web

业务系统（EPC、考核、OA等）

整合了V1版中的PIC, PIC.Common, PIC.Data三个项目，使在编写相关模块时只要引用一个PIC.Core项目就可以了；

**PIC 组件（PIC.Component）**

主要用于第三方组件的包装与整合已使实际使用更加简单，目前主要整合的有MS Office Excel和MS Project的数据操作；

**PIC资源（PIC.Resources）**

主要用于维护在项目中使用的程序集（第三方或开源程序集）或工具（代码生成工具等）；

**PIC 门户（PIC.Portal）**

**Model文件夹**

维护的是PIC.Portal使用的实体（V1版本中的PIC.Portal.Entity和PIC.Portal.Rule），与原版本兼容，但命名空间同为PIC.Portal.Model，Portal的实体的基类（EntityBase<T>）也修改为ModelBase<T>。

**Services文件夹**

维护的是PIC.Portal以及在其上建立的应用，使用的服务。

1. Code下维护的是编码生成服务
2. Data 下维护的数据导入导出服务
3. FileSystem下为文件系统服务
4. Log下为日志服务
5. PortalService为为整个Portal提供上下文信息与系统认证等功能的类，其中有很多静态方法可以在应用实体类或页面中直接调用。

**Web文件夹**

维护的是Portal Web开发中需要使用的类，相当与V1版本中的PIC.Web

**PIC 门户服务服务Provider（PIC.Portal.ServicesProvider）**

目前提供了门户默认使用的WebPortalServiceProvider（以PIC.Portal.Services为基础）和以金慧登录服务为基础的GwPortalServiceProvider。

**PIC 虚拟目录Provider（PIC.Portal.Web.VirtualPathProvider.Ext）**

虚拟目录的提供者，目前提供了以ExtJs为框架的提供者。后面可根据需要扩展为其他框架（MVC或WAP页面，当然前台Web页面也要修改）

**PIC 门户服务（PIC.Portal.Services）**

基于WCF用于为门户系统提供用户登录、文件上传等服务

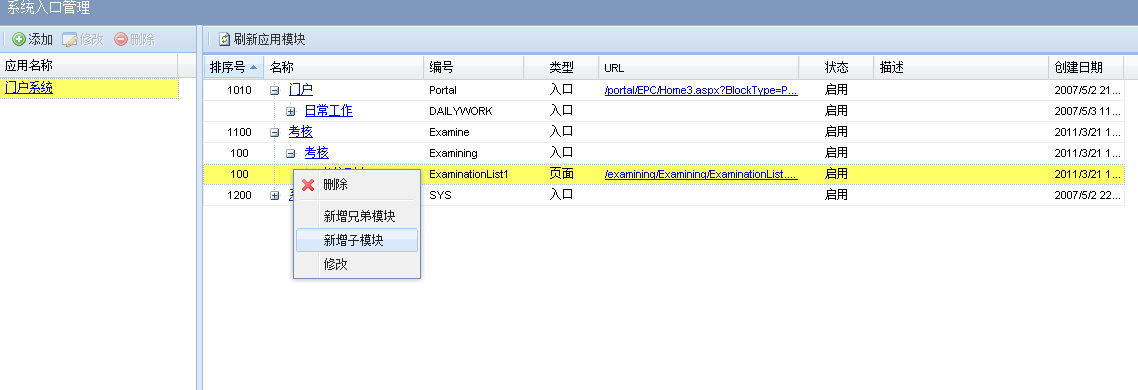
**PIC 门户服务（PIC.Portal.Web）**

门户的ASP.NET工程。

1. **门户系统**

相比于V1重新组织了系统管理的UI展现，组织人员管理和模块权限维护放在了一起方便操作，之前的数据模板维护改为模板维护，加入编码模板模块

1. 系统入口管理，如下图：



（系统入口管理）

分为两部分，左边为应用管理（为以后扩展使用，目前没有意义），右边可通过右键完成模板的增、删、改。

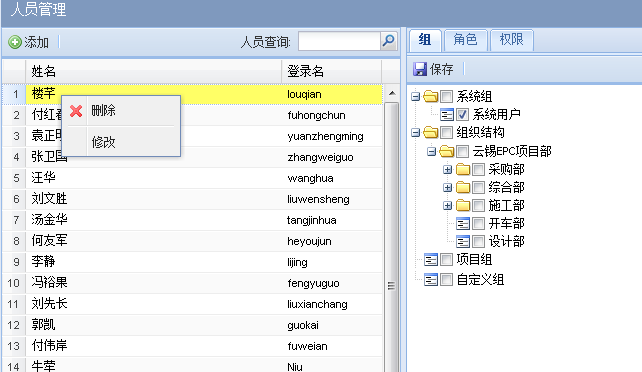
1. 系统图表维护

意义不大，用于系统中获取相关上传图片（可能改为系统文件管理更有意义）

1. 人员、组、角色管理

三个模块组织基本相似，左边为相应管理区域，右边为相应属性赋值区域

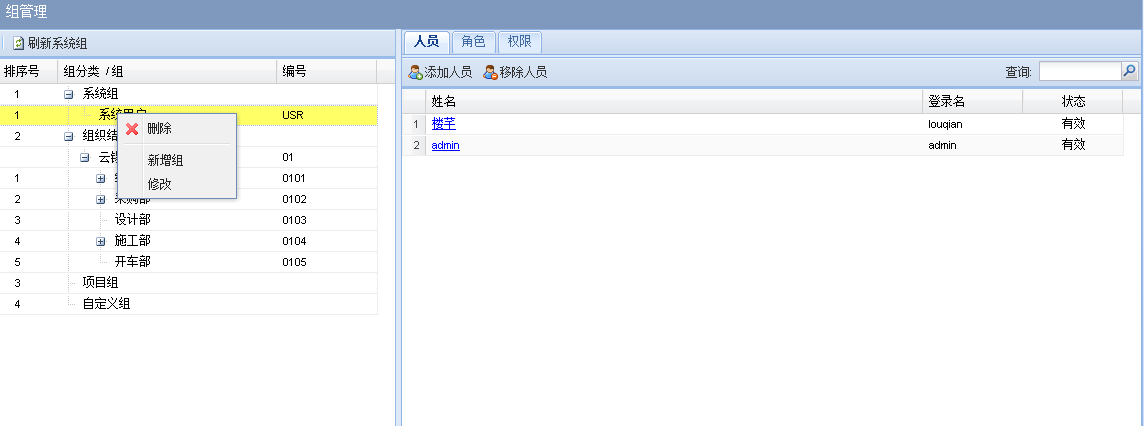
人员管理，如下图：



（人员管理）

左侧可以添加人员，并可右键修改删除人员，右边可以为相应人员设置所属组，角色及所拥有的权限。

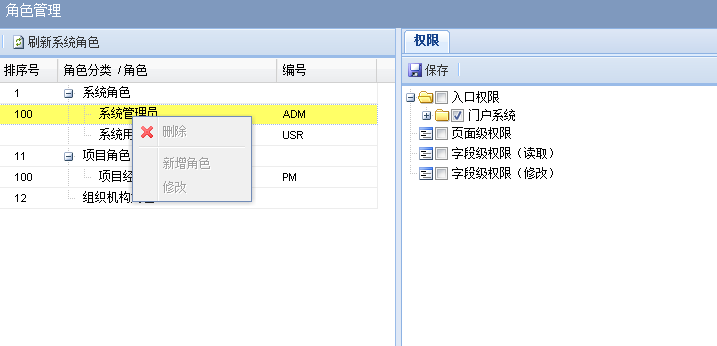
组管理，如下图：



（组管理）

左侧右键可实现组的删除、修改及子组的增加，右边可设置对应组下的人员，组所拥有的角色及相关组所拥有的权限。

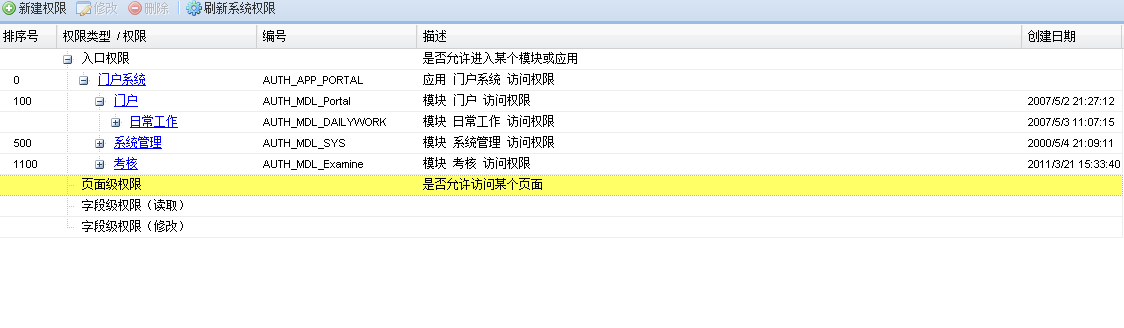
角色管理，如下图：



（角色管理）

左侧维护角色的增删改，目前角色分为系统角色，项目角色和组织机构角色，其中系统角色为系统定义无法编辑。右侧为相应角色所拥有的权限，若为人员或组分配了相应角色，则此人员或组将拥有此角色所拥有的权限。

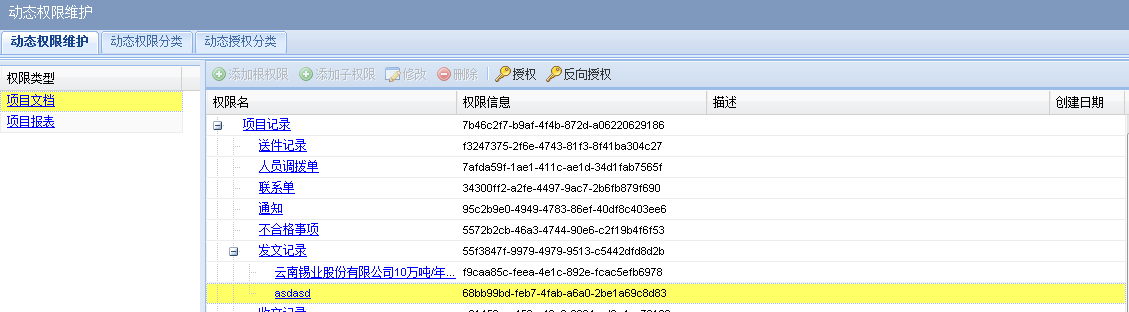
1. 权限管理，如下图：



（权限管理）

目前此模块实际价值只是实现入口权限的查看

1. 动态权限管理



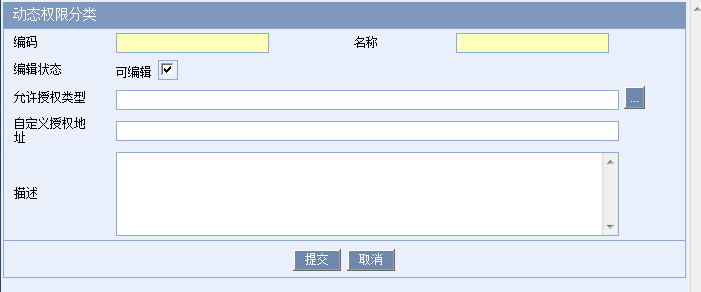
（动态权限管理）

PIC中所谓动态权限是系统在运行过程中由程序动态产生的权限。动态权限管理模块允许手工对动态生成的权限进行授权操作。下面叙述一个项目文档的动态权限在开发与维护中的应用：



（动态权限分类管理）

1. 在动态权限分类中建立项目文档类型



（动态权限分类编辑）

“允许授权类型”为此类型的动态权限允许授权给哪些对象，一般有（组，人员和角色）；

“自定义授权地址”为此类型的自定义的授权地址（否则位系统默认授权地址DPermissionGrant.aspx）

1. 当程序运行时，比如用户上传了一个项目文档，当操作完成时，调用动态权限API创建对应相应文档的动态权限，示例代码：

|  |
| --- |
| public void DoCreate()  {  this.DocState = StateEnum.New;  this.CreateTime = DateTime.Now;  this.DoValidate();  // 事务开始  this.CreateAndFlush();  DynamicAuth dauth = new DynamicAuth();  this.SetDAuthData(dauth);  dauth.DoCreate();  }  /// <summary>  /// 根据文档节点设置动态权限数据  /// </summary>  private void SetDAuthData(DynamicAuth dauth)  {  dauth.Name = this.FileName;  dauth.Data = this.Id;  dauth.Grantable = true;  dauth.CreaterID = this.CreateId;  dauth.CreaterName = this.CreateName;  dauth.Description = this.Remark;  // dauth.SortIndex = this.SortIndex;  if (!String.IsNullOrEmpty(this.NodeId))  {  DynamicAuth pdauth = DynamicAuth.FindFirst(Expression.Eq("Data", this.NodeId));  dauth.ParentID = pdauth.DynamicAuthID;  dauth.Path = String.Format("{0}.{1}", (pdauth.Path == null ? String.Empty : pdauth.Path), pdauth.DynamicAuthID);  dauth.PathLevel = pdauth.PathLevel + 1;  }  dauth.CatalogCode = PRJ\_DocStructure.DAuthCatalogCode; // 动态权限编码  dauth.Tag = DAuthTag; // 动态权限附加信息  } |

1. 此时用户可以在动态权限维护中对动态生成的权限进行授权操作
2. 在用户访问此文档时，程序根据设置的权限进行访问控制（需要手工写代码进行控制）
3. 数据导入、导出模板（Excel）

数据导入、导出模板根据Excel文件的批注生成模板配置信息并保存在数据库中。

数据导入模板命令符：

#： 模板属性符号，如：#{Target:” PRJ\_PostDuty”} 表示数据将被导入表”PRJ\_PostDuty”，其他配置字段，请参考类PIC.Portal.ImportTemplatePropertyNode中的相关属性

@: 模板列符号，如：@{Post}表示此标识对应与数据库中表的Post字段，其他配置字段，请参考类PIC.Portal.ImportTemplateColumnNode中的相关属性

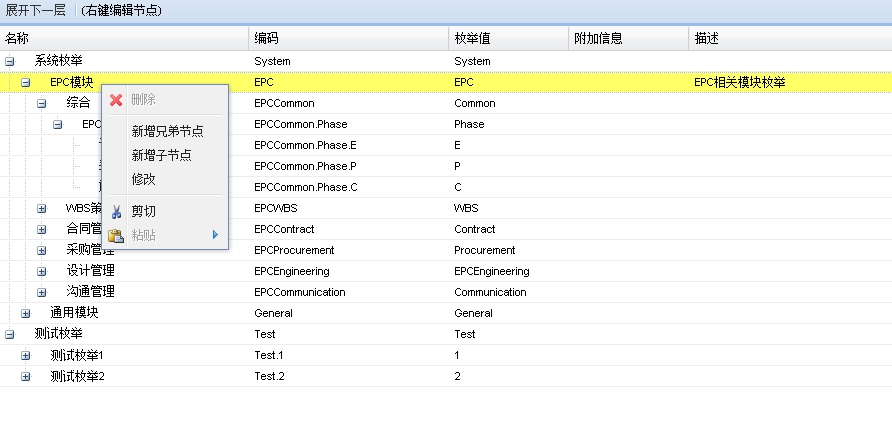
$: 模板命令符号，如：${Begin}表示导入数据有此开始，其他配置字段，请参考类PIC.Portal.ImportTemplateCommandNode中的相关属性

数据导出模板命令符基本上同导入模板，具体请参考相关示例。

1. 模板编码

模板编码采用Castle的NVelocity模块生成编码。编码的语法参考NVelocity的标准语法（VTL, <http://www.velocity.apache.org/engine/devel/vtl-reference-guide.html>），其中编码的预定义上下文请参考类CodeGeneratorContext

1. 系统枚举、系统参数



（系统枚举维护页面）

可以设置系统枚举的附加信息，从而可以把系统枚举作为系统参数使用。

1. **系统开发**

若开发新的模块（EPC、OA等），需要注意以下几点

1. 一般新模块需要创建两个项目， Model和Web（比如：PIC.Examing.Model和PIC.Examing.Web）。
2. 为了实现不同应用共享FormAuthentication实现单点登录，需要在web.config下的system.web配置项中加入配置块：

|  |
| --- |
| <machineKey validationKey="3049C51F1722673C113BB077CDCCDE167185152D9D0A49F254D478C75A461053405D8088F5AB1D87F2C9CFD34EE37F492B51B20883226CFCAB52FA9AE0D3570C"  decryptionKey="82CF4219DCDB289EC37B5FBBC551BC9ED01E284D2F6137B2"  validation="SHA1" /> |

1. 若存在多数据库，一般需要为每个数据库建一个基类（比如：ExamModelBase）,并在web.config下的activerecord配置项增加相关配置类，在程序初始化中(Global.asax中InitApplication代码段)加入相应的实体类应用程序集。

初始化新应用实体类程序集：

|  |
| --- |
| PortalService.Initialize(new string[] { "PIC.Examining.Model" }) |

相关ActiveRecord配置段:

|  |
| --- |
| <activerecord>  <config>  <add key="connection.driver\_class" value="NHibernate.Driver.SqlClientDriver" />  <add key="dialect" value="NHibernate.Dialect.MsSql2005Dialect" />  <add key="connection.provider" value="NHibernate.Connection.DriverConnectionProvider" />  <add key="connection.connection\_string" value="Server=.;DataBase=NCRL\_PICPortal;User ID=sa;Password=sasa;Persist Security Info = True" />  <add key="proxyfactory.factory\_class" value="NHibernate.ByteCode.Castle.ProxyFactoryFactory, NHibernate.ByteCode.Castle" />  </config>  <config type="PIC.Examining.Model.ExamModelBase`1, PIC.Examining.Model">  <add key="connection.driver\_class" value="NHibernate.Driver.SqlClientDriver" />  <add key="dialect" value="NHibernate.Dialect.MsSql2005Dialect" />  <add key="connection.provider" value="NHibernate.Connection.DriverConnectionProvider" />  <add key="connection.connection\_string" value="Server=.;DataBase=NCRL\_PICExamine;User ID=sa;Password=sasa;Persist Security Info = True" />  <add key="proxyfactory.factory\_class" value="NHibernate.ByteCode.Castle.ProxyFactoryFactory, NHibernate.ByteCode.Castle" />  </config>  </activerecord> |