|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | | |  | |
| **SMT开发框架使用手册** | | | | |
| **项目名称：** | | SMT开发框架 | | |
| **项目编号：** | | SMTO\_SFV1.0 | | |
| **文档编号：** | | SMTO\_PL\_ODV1.0 | | |
| **产品版本号：** | | V1.0 | | |
| **编制单位：** | | 神州通在线科技有限公司 | | |
| **编制日期：** | | 2012年2月20日 | | |
|  | | | | |
| 编制 | 审核 | | 批准 | 生效日期 |
| 宋涛 |  | |  |  |

**版本修订批准记录**

【 历次版本修订记录，版本修订描述重要说明】

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **提交日期** | **版本号** | **版本修订描述** | **作者** | **批准人** |
| 2012/2/20 | 1.0 | 初始版本 | 宋涛 |  |
|  |  |  |  |  |
|  |  |  |  |  |

目 录

[1 概述 7](#_Toc317861689)

[1.1 目的 7](#_Toc317861690)

[1.2 预期读者 7](#_Toc317861691)

[1.3 术语和缩写 7](#_Toc317861692)

[1.4 参考资料 7](#_Toc317861693)

[2 总体设计概览 8](#_Toc317861694)

[2.1 总体结构 8](#_Toc317861695)

[2.1.1 总体示意图 8](#_Toc317861696)

[2.1.2 层次结构 8](#_Toc317861697)

[2.2 开发框架组件结构 11](#_Toc317861698)

[2.2.1 工程结构 11](#_Toc317861699)

[3 代码结构组织 12](#_Toc317861700)

[3.1 前台代码结构 12](#_Toc317861701)

[3.1.1 示意图 12](#_Toc317861702)

[3.1.2 工程命名 12](#_Toc317861703)

[3.2 后台代码结构 13](#_Toc317861704)

[3.2.1 示意图 13](#_Toc317861705)

[3.2.2 工程命名 13](#_Toc317861706)

[4 前台开发指南 15](#_Toc317861707)

[4.1 前台主工程 15](#_Toc317861708)

[4.1.1 工程结构 15](#_Toc317861709)

[4.1.2 业务划分 16](#_Toc317861710)

[4.2 前台业务工程 18](#_Toc317861711)

[4.2.1 工程结构 18](#_Toc317861712)

[4.2.2 业务划分 18](#_Toc317861713)

[4.3 UI模式 19](#_Toc317861714)

[4.3.1 AJAX异步通讯 19](#_Toc317861715)

[4.3.2 Grid-Detail模式 19](#_Toc317861716)

[4.4 异步通讯编码规范 22](#_Toc317861717)

[4.4.1 JSON数据格式 22](#_Toc317861718)

[4.4.2 标准通讯模式 23](#_Toc317861719)

[4.4.3 传递简单对象到服务端，并返回简单对象 25](#_Toc317861720)

[4.4.4 传递两个简单对象到服务端，并返回简单对象 26](#_Toc317861721)

[4.4.5 传递对象列表到服务端，并返回对象列表 27](#_Toc317861722)

[4.4.6 传递一个对象及一个对象列表到服务端，并返回对象列表 29](#_Toc317861723)

[4.5 公共控件 32](#_Toc317861724)

[4.5.1 公司选择对话框 32](#_Toc317861725)

[4.5.2 部门选择对话框 34](#_Toc317861726)

[4.5.3 员工选择对话框 36](#_Toc317861727)

[4.5.4 日期控件 38](#_Toc317861728)

[4.6 模式/非模式对话框编码规范 39](#_Toc317861729)

[4.6.1 函数定义 39](#_Toc317861730)

[4.6.2 典型UI 40](#_Toc317861731)

[4.6.3 HTML代码 41](#_Toc317861732)

[4.6.4 JS代码 41](#_Toc317861733)

[4.6.5 C#代码 43](#_Toc317861734)

[4.7 Grid编码规范 44](#_Toc317861735)

[4.7.1 函数定义 44](#_Toc317861736)

[4.7.2 典型UI 46](#_Toc317861737)

[4.7.3 HTML代码 46](#_Toc317861738)

[4.7.4 JS代码 46](#_Toc317861739)

[4.7.5 C#代码 48](#_Toc317861740)

[4.8 审核控件编码规范 50](#_Toc317861741)

[5 后台开发指南 51](#_Toc317861742)

[6 前后台通讯开发指南 52](#_Toc317861743)

[7 数据访问开发指南 52](#_Toc317861744)

[8 公共服务使用指南 52](#_Toc317861745)

[9 部署指南 52](#_Toc317861746)

# 概述

## 目的

为解决神州通在线协同办公系统原有开发框架分层不明确，编码复杂，数据传输量大的弊端；改善代码调试困难，开发效率低下的问题；改善客户端下载数据量过大，界面反映缓慢，客户体验低下的问题，现启动神州通在线开发框架项目。

采用更加清晰的代码分层策略，使用成熟主流的技术架构，引入微软Enterprise Library，nHibernate，ASP.NET MVC。以提高开发效率。提升用户使用体验。

## 预期读者

开发（设计）人员，评审人员，其他授权人员。

## 术语和缩写

|  |  |
| --- | --- |
| **术语、缩略语** | **解释** |
| SMT | 深圳市神州通投资(集团)有限公司 |
| **WP** | **工作计划项目（本文以此项目作为示范）** |
|  |  |
|  |  |

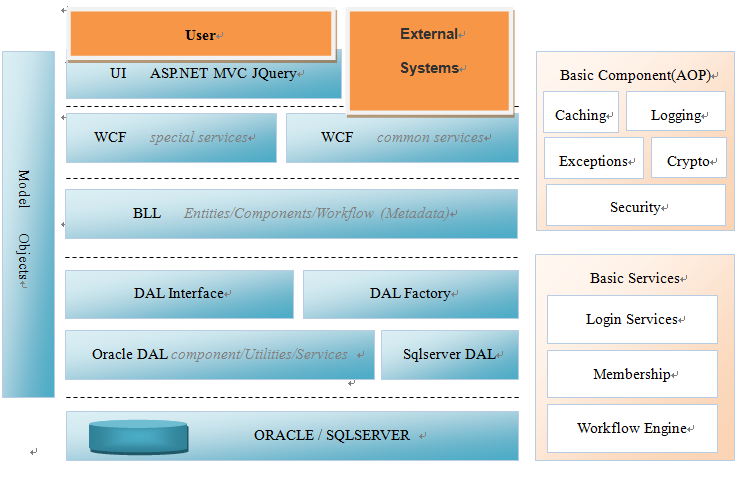
## 参考资料

|  |  |  |
| --- | --- | --- |
| **文档名称** | **版本号** | **日期** |
|  | 1 |  |
|  |  |  |
|  |  |  |
|  |  |  |

# 总体设计概览

## 总体结构

### 总体示意图



### 层次结构

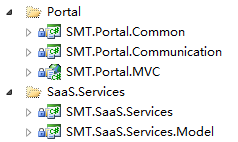
下面是针对每一层的基本介绍：

|  |  |  |
| --- | --- | --- |
| **序号** | **名称** | **概述** |
| 1 | UI 用户界面层 | 默认采用ASP.NET MVC+JQuery作为基本UI技术方案。由于UI层基于服务层调用，因此可以在UI层提出其他替代方案，比如Silverlight / WPF。  在UI层，负责使用者与整个系统的交互。在这一层中，理想的状态是不应包括系统的业务逻辑，逻辑代码，仅与界面元素有关。 |
| 2 | WCF 服务层 | 服务层，使用WCF作为服务方案，在服务层需要提供给系统本事提供的专业服务，以及需要定制开放给第三方的公共服务。  主要负责为UI提供服务接口，以及系统的分布式部署提供支持。 |
| 3 | BLL 业务逻辑层 | 是整个系统的核心，它与这个系统的业务紧密相关，是，业务层需要提供对业务粒度的封装，业务接口源于需求，以满足用户需求为设计根本。  主要负责实现整个系统的核心业务、以及调用数据访问层。  其具体也可以分为基础单元的业务实现。以及基于基础业务实现的一套完整的业务流程。 |
| 4 | DAL 接口层 | 为DAL层定义接口规范。主要是实现面向接口编程。脱离了与具体数据库的依赖。 |
| 5 | DAL 工厂 | 根据配置文件创建具体的DAL接口实现，比如 分别基于Oracle 、SqlServer、MySql等主流数据库 |
| 6 | Oracle DAL | DAL接口层的具体实现，基于Oracle数据库(默认支持)。  在DAL层的具体实现上可以选择可以使用ORM比如 NHibernate、Entity Framework、或者T-SQL（可以选择微软企业库Data Access Application Block ）。  主要是实现对数据表的Select，Insert，Update，Delete的操作。  备注：T-SQL语句使用XML进行可配置管理。并提供出XML的读取接口。 |
| 7 | SQL DAL | DAL接口层的具体实现，基于SqlServer数据库(扩展支持)。  在DAL层的具体实现上可以选择可以使用ORM比如 NHibernate、Entity Framework、或者T-SQL（可以选择微软企业库Data Access Application Block ）。  主要是实现对数据表的Select，Insert，Update，Delete的操作。 |
| 11 | Model Object | 业务实体，映射到数据库中的表 |
| 12 | 基础组件：Caching 缓存 | 为应用程序提供高速缓存来提供性能。缓存可支持永久性缓存（基于内存）和非永久性缓存（基于数据库或单独的缓存文件）。  缓存需要提供检测、添加、删除缓存数据。 |
| 13 | 基础组件：Crypto 加密 | 使用对称加密算法对敏感信息进行加密保存、或对其进行解密。对单台机器上使用的信息进行加密(without using keys)。  创建密码信息的散列值进行保存，使用时需要 对用户提供的密码散列值与保存值进行对比。  简化了开发人员为敏感信息进行加解密工作。  使用微软企业库Cryptography Application Block |
| 14 | 基础组件：Security 安全 | **暂定.**  通过一个或多个安全系统或机制，帮助开发人员在应用程 序中实现通用的安全相关任务。  提供认证、授权、角色管理、缓存用户登录时的安全凭据（认证、授权信息）。 |
| 15 | 基础组件：Exception 异常 | 一致的异常处理策略。对于异常的处理，不但要能够提供有效的信息，并且不会因为系统异常而泄漏系统的机密信息，而是隐藏敏感信息 |
| 16 | 基础组件：Logging 日志 | 提供简单标准一致的Logging机制。需要对业务及操作数据、系统运行异常做日志，希望能通过外部的配置能将日志输出到不同的存储区 。  能灵活指定哪类信息以何种格式输出、输出  到何处 ，能将日志定向到多个日志存储区。 |
| 17 | 基础服务：Login Services | 提供单点登录的支持，统一管理在线用户以及用户授权、认证信息。  可以结合Security组件使用。 |
| 18 | 基础服务：Membership Services权限/组织架构 服务 | 提供统一的用户授权服务、以及用户组织架构信息。 |
| 19 | 基础服务：Workflow Services 工作流引擎服务 | 提供统一的工作流引擎服务接口。 |

## 开发框架组件结构

### 工程结构

#### 示意图



#### 程序集

|  |  |  |
| --- | --- | --- |
| 序号 | 程序集 | 备注 |
| 1 | SMT.Portal.Common.dll | 开发框架Common组件包，封装框架的功能方法等等 |
| 2 | SMT.Portal.Communication.dll | 开发框架的通讯组件包，对WCF通讯层封装、解析进行统一封装 |
| 3 | SMT.SaaS.Services.dll | 开发框架的平台服务包，对协同办公平台的各种服务进行统一封装 |
| 4 | SMT.SaaS.Services.Model.dll | 开发框架平台服务包使用的实体程序集 |
| 5 | SMT.Portal.MVC.dll | 开发框架的MVC主包，由系统集成/部署管理员进行统一集成生成部署。 |

项目开发人员需要引用1、2、3、4这4个框架组件程序集。

系统集成/部署管理员统一对所有在此框架下开发的项目进行统一集成部署，生成一个SMT.Portal.MVC.dll的最终程序集及其他项目工程集，参考“9.部署指南”。

# 代码结构组织

## 前台代码结构

### 示意图



前台至少包括以上2个工程，其他工程可以根据项目自行添加。

### 工程命名

前台需要引用/创建两个工程：

#### 前台主工程

此工程无须项目人员创建，由开发框架统一提供外壳工程(Shell Project)，然后项目管理人员再添加到本项目解决方案中，可以重新命名为：

**SMT.XXXX.Client.UI**

其中XXXX为项目英文简称，如WP(工作计划)：SMT.WP.Client.UI。

此工程为ASP.Net MVC工程，需要引用前台业务层，具体结构请参考“4.前台开发指南”。

#### 前台业务工程

此工程由项目管理人员创建，命名规则为：

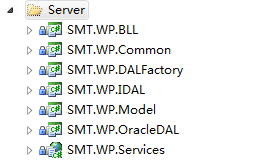
**SMT.XXXX.Client.BLL**

其中XXXX为项目英文简称，如WP(工作计划)：SMT.WP.Client.BLL。

具体结构请参考“4.前台开发指南”。

## 后台代码结构

### 示意图



后台必须包括至少以上7个工程，其他工程可以根据项目自行添加。

### 工程命名

#### 后台通用工程

此工程由项目管理人员创建，命名规则为：

**SMT.XXXX.Common**

其中XXXX为项目英文简称，如WP(工作计划)：SMT.WP.Common。

具体结构请参考“5.后台开发指南”。

#### 后台服务工程

此工程由项目管理人员创建，命名规则为：

**SMT.XXXX.Services**

其中XXXX为项目英文简称，如WP(工作计划)：SMT.WP.Services。

具体结构请参考“5.后台开发指南”。

#### 后台业务工程

此工程由项目管理人员创建，命名规则为：

**SMT.XXXX.BLL**

其中XXXX为项目英文简称，如WP(工作计划)：SMT.WP.BLL。

具体结构请参考“5.后台开发指南”。

#### 后台实体工程

此工程由项目管理人员创建，命名规则为：

**SMT.XXXX.Model**

其中XXXX为项目英文简称，如WP(工作计划)：SMT.WP.Model。

具体结构请参考“5.后台开发指南”。

#### 后台数据工厂工程

此工程由项目管理人员创建，命名规则为：

**SMT.XXXX.DALFactory**

其中XXXX为项目英文简称，如WP(工作计划)：SMT.WP.DALFactory。

具体结构请参考“5.后台开发指南”。

#### 后台数据接口工程

此工程由项目管理人员创建，命名规则为：

**SMT.XXXX.IDAL**

其中XXXX为项目英文简称，如WP(工作计划)：SMT.WP.IDAL。

具体结构请参考“5.后台开发指南”。

#### 后台数据实现工程

此类工程由项目管理人员创建，命名规则为：

**SMT.XXXX.YYYYDAL**

其中XXXX为项目英文简称，如WP(工作计划)；YYYY为数据库名称，如Oracle，综合命名为：

SMT.WP.OracleDAL。

本类工程可以根据项目的实际需要创建多个，如：  
SMT.WP.OracleDAL、SMT.WP.SQLDAL

具体结构请参考“5.后台开发指南”。

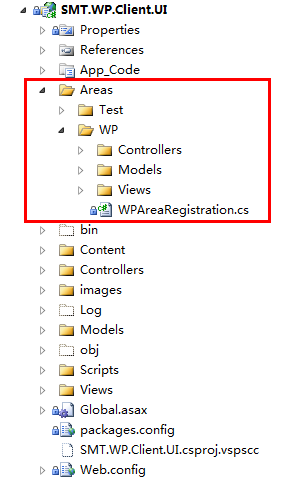
# 前台开发指南

## 前台主工程

### 工程结构

前台主工程是一个ASP.Net MVC的工程，目前由框架统一提供一个外壳工程给项目组使用，项目组直接将此外壳工程加入到自己的解决方案中，并可以任意命名。

工程详细目录结构如下：

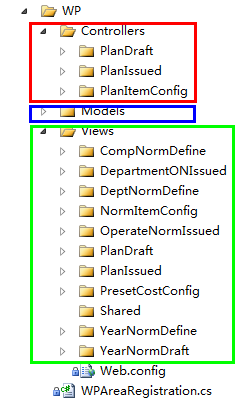


项目组在引入外壳工程后，新增一个Area，名称为本项目英文简称，如“WP”。

WP即为项目组进行编码的工作区，MVC中的模型、控制器及视图均包含在其中。

**Areas外部为开发框架提供的外壳编码，请勿做任何修改与变更，项目组开发人员只能在本项目Area内进行编码**。

### 业务划分



#### 视图

如图”Views”目录，包含所有的视图文件。

视图代码仅应该包括所有的UI代码，如HTML，JS，RAZOR代码等等。

#### 模型

如图”Models”目录，包含MVC中的模型代码。

**在我们的开发框架中，所有的模型/实体/数据访问等等，均独立为不同的工程，并不需要在此目录中放置模型代码，所以此目录保持为空即可。**

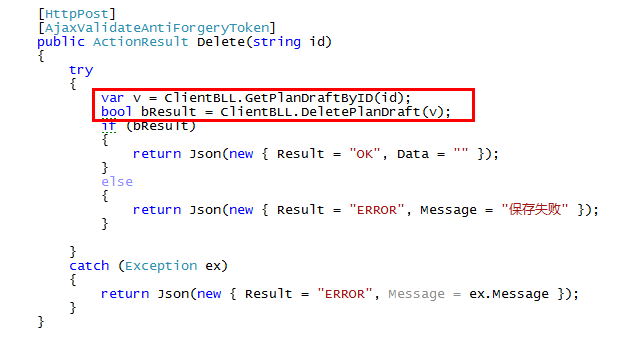
#### 控制器

如图”Controllers”目录，包含所有的控制器文件。

控制器代码将包含核心的MVC代码，所有的后台响应及调用均在控制器中。

**控制器内的代码应该保持简洁明了，与后台进行交互的代码及前台的一些业务逻辑代码，不应该放入控制器中，应该放入前台业务工程之中。**

如下所示：



Delete为控制器中的一个Action方法，需要进行一个删除数据的业务操作，如代码所示，删除的业务操作（其中还包含了与后台通讯的代码）由业务工程负责，控制器中仅进行调用，然后将结果包装后传递给前端浏览器中。

**类似“瘦客户端“的概念，我们可以称之为”瘦控制器“，即MVC结构中尽量保持精简，业务单独抽离出来**。

#### 前台业务

前台主工程应该只包括与UI相关的代码，与后台进行交互的代码及前台的一些业务逻辑代码，不应该放入前台主工程中，请在前台业务工程中进行编写。

参见4.2前台业务工程。

## 前台业务工程

前台业务工程由项目管理人员进行创建，命名为：SMT.XXXX.Client.BLL，其中XXXX为本项目英文简称。

前台业务工程为标准的类库工程，凡是与后台进行交互的代码及一些前台业务逻辑，均应该包括在本工程中，前台主工程应该只包括UI相关的代码。

### 工程结构

前台业务工程中的结构划分，可以遵照一般业务类库的结构划分，并无强制性的规范。

建议以业务模块来进行结构划分，创建不同的业务目录。

### 业务划分

#### 前台业务

有一些业务，适合放置于前台，那么应该放入前台业务工程中。

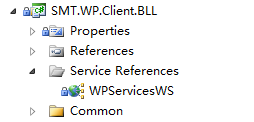
常见的一般为前台的一些校验代码，对象包装代码等等。

#### 前台公共组件

可以将前台需要的公共组件放入前台业务工程中。

#### 后台通讯

与后台服务端进行通讯的代码，应该统一放入前台业务工程中，如WCF服务端引用：



**后台通讯代码还应该包含实现框架通讯层的相关接口，具体请参考“6.前后台通讯开发指南”。**

## UI模式

### AJAX异步通讯

在前端浏览器中的所有与后台的交互操作，不使用传统B/S的全页面刷新的模式，均以AJAX异步操作、局部刷新的方式实现，开发框架使用jQuery作为前端框架库，所有的异步操作均通过jQuery异步操作库来实现，请参考。

### Grid-Detail模式

前台界面统一使用Grid-Detail模式，即主界面为一个Grid列表，新增/修改/查看操作均表现为弹出一个Detail窗体来进行操作，如下所示：

#### 主界面



主界面以一个数据表格为核心，上方一般有一个条件查询区，表格上方放置一个增/删/改/查的标准工具栏，右侧一般为一个审核状态下拉栏。

#### 新增

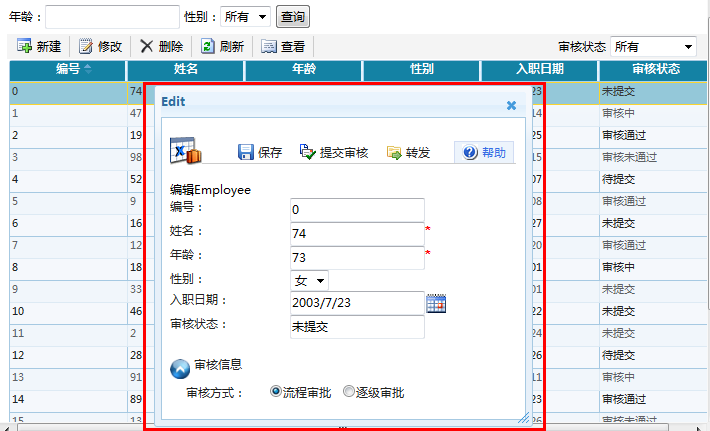
点击“新建”按钮，弹出一个新建对话框：



当点击“保存”按钮时，使用异步操作与后台进行通讯，将数据保存到后台数据库中。

#### 修改

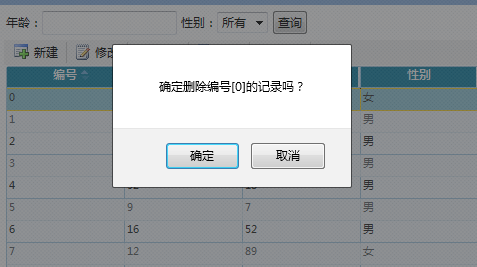
点击“修改”按钮，弹出一个修改对话框：



当点击“保存”按钮时，使用异步操作与后台进行通讯，将数据修改到后台数据库中。

#### 删除

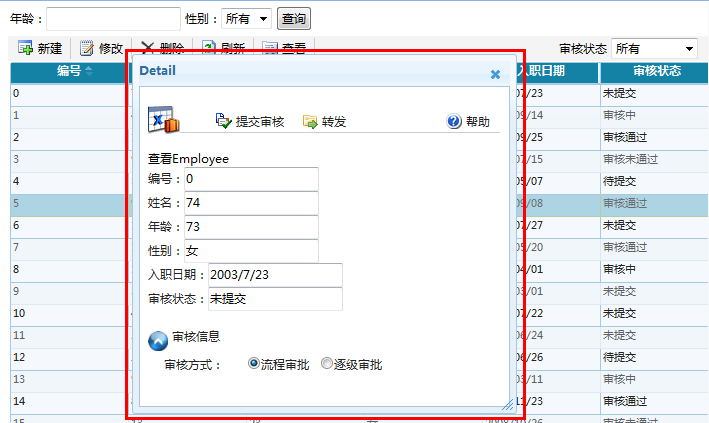
点击“修改”按钮，弹出一个删除确认对话框：



当点击“确认“时，使用异步操作与后台进行通讯，将数据从数据库中删除。

#### 查看

点击“查看”按钮，弹出一个查看对话框：



## 异步通讯编码规范

### JSON数据格式

开发框架页面与后端统一使用JSON数据格式进行通讯，关于JSON，可以参考：

<http://www.json.org/js.html>

页面与后端通讯时，传递的实际数据格式为字符串，json对象需要转换成json字符串才能传递到后台，因为http通讯都是传递字符，不可能传递一个js对象到后台去，所以我们需要在页面中将JSON对象和JSON字符串互相进行转换。

请注意json字符串和json对象的区别，json字符串是一个js中的标准字符串，只不过字符串的内容

符合json的格式；而json对象则是一个标准的js对象，只不过内容符合json的格式，必须转换为

字符串后才能传递给后台

#### JSON对象转字符串

//使用json对象，转换为json字符串

var jsObject = new Object();

jsObject.FirstName = "323";

jsObject.LastName = "2323";

var jsonString = JSON.stringify(jsObject);

//结果：jsonString = '{ "FirstName": "323", "LastName": "2323" }';

#### JSON字符串转对象

//使用json格式的字符串，转换为json对象

var jsonString = '{ "FirstName": "323", "LastName": "2323" }';

var jsonObject = JSON.parse(jsonString);

//可以调用：jsonObject.FirstName/LastName

### 标准通讯模式

发送对象到后台，后台返回对象到前台。

这个过程中可能发生两种错误：

1. 前后台通讯错误(ajax通讯错误）

这种情况代表AJAX通讯发生了错误，无法与后台正常通讯。

1. 前后台通讯正常，但是后台逻辑发生错误，不能返回正确的数据

这种情况代表AJAX通讯正常，但是后台业务逻辑可能发生了错误，比如无法获取数据，数据库可能无法连接，数据非法等等，此时后台应该统一封装返回对象给前端，通知前端后台发生错误，并可以对错误信息进行包装和展现。

示例：

点击按钮，与后台通讯，将一个UserName对象传递给后台，并从后台接收一个新的UserName对象。

UserName在后台的C#代码定义：

public class UserName

{

public string FirstName { get; set; }

public string LastName { get; set; }

}

前台js中不需要定义。

#### JS代码

页面View视图中的JS代码如下：

$("#btn12").click(function () {

//数据需要与后台的UserName对象的字段一一对应

var jsObject = new Object();

jsObject.FirstName = "323";

jsObject.LastName = "2323";

var vData = JSON.stringify(jsObject);

$.ajax({

type: "post", //异步通讯的类型，统一使用POST

url: "Ajax1/GetData",//URL，指向Ajax1控制器中的GetData方法（Action）

data: vData, //传递给后台的数据，统一使用JSON格式

dataType: "json", //指定服务器返回的数据类型

contentType: "application/json; charset=utf-8", //发送的内容编码类型

beforeSend: function (XMLHttpRequest) {

//此处可以用来显示加载动画

},

success: function (context, textStatus) {

if (context.Result == "OK") {

var returnObj = context.Data;

//returnObj为后台返回的UserName对象

} else {

//2.前后台通讯正常，但是后台逻辑发生错误，不能返回正确的数据

alert(context.Message);

}

},

complete: function (XMLHttpRequest, textStatus) {

//此处用来隐藏加载动画

},

error: function () {

//出错处理

//1.前后台通讯错误(ajax通讯错误）

}

});

});

#### C#代码

控制器中的Action代码如下：

//当页面传递某个JSON格式的数据到控制器中时，可以直接使用格式匹配的C#对象来作为

//Action参数，MVC框架会自动将页面传递过来的数据转换为匹配的C#对象

//注意：数据格式必须一一匹配

[HttpPost]

public JsonResult GetData(UserName userName)

{

//返回对象均包含后台处理是否成功的标志

try

{

return Json(new { Result = "OK", Data = userName });

}

catch (Exception ex)

{

//写日志

//做异常处理

//返回错误信息到前台

return Json(new { Result = "ERROR", Message = ex.Message });

}

}

### 传递简单对象到服务端，并返回简单对象

传递一个简单对象到服务端，并返回简单对象。

简单对象指的是对象仅包含基本类型的字段，不包含数组和子对象。

**注意：本例演示了3种json字符串的获取方法，建议使用第3种js对象构造法，这种方法更简单，更宜操作，特别是对复杂json对象进行包装时。**

#### JS代码

//传递简单对象到服务端，并返回简单对象

$("#btn8").click(function () {

//数据需要与后台的UserName对象的字段一一对应

//1.使用json字符串

var jsonString = '{ "FirstName": "323", "LastName": "2323" }';

//2.使用json对象，转换为json字符串

var jsonObject = { "FirstName": "323", "LastName": "2323" };

var jsonString2 = JSON.stringify(jsonObject);

//3.直接使用js对象构造json格式对象，再转换为json字符串(推荐使用）

var jsObject = new Object();

jsObject.FirstName = "323";

jsObject.LastName = "2323";

var jsonString3 = JSON.stringify(jsObject);

$.ajax({

type: "post",

url: "Ajax1/GetUserName",

data: jsonString3,

dataType: "json",

contentType: "application/json; charset=utf-8",

beforeSend: function (XMLHttpRequest) {

//此处可以用来显示加载动画

},

success: function (context, textStatus) {

var objstring = context.FirstName;

$("#txt8").attr("value", objstring);

},

complete: function (XMLHttpRequest, textStatus) {

//此处用来隐藏加载动画

},

error: function () {

//出错处理

}

});

});

#### C#代码

[HttpPost]

public JsonResult GetUserName(UserName userName)

{

//直接返回原对象，未进行异常处理封装，请参考标准通讯模式

return Json(userName);

}

### 传递两个简单对象到服务端，并返回简单对象

将两个简单对象包装为一个对象传递给服务端，并返回一个简单对象

#### JS代码

$("#btn11").click(function () {

//数据需要与后台的UserName对象的字段一一对应

//1.使用json字符串

var vData1 = '"user1":{ "FirstName": "323", "LastName": "2323" }';

var vData2 = '"user2":{ "FirstName": "666", "LastName": "666" }';

var vData = "{" + vData1 + "," + vData2 + "}";

//2.使用js对象

var userName1 = new Object();

userName1.FirstName = "323";

userName1.LastName = "2323";

var userName2 = new Object();

userName2.FirstName = "444";

userName2.LastName = "4444";

var jsObject = new Object();

jsObject.user1 = userName1;

jsObject.user2 = userName2;

var jsonString = JSON.stringify(jsObject);

$.ajax({

type: "post",

url: "Ajax1/SendTwoUser",

data: jsonString,

dataType: "json",

contentType: "application/json; charset=utf-8",

beforeSend: function (XMLHttpRequest) {

//此处可以用来显示加载动画

},

success: function (context, textStatus) {

var objstring = context.FirstName;

$("#txt11").attr("value", objstring);

},

complete: function (XMLHttpRequest, textStatus) {

//此处用来隐藏加载动画

},

error: function () {

//出错处理

}

});

});

#### C#代码

[HttpPost]

public JsonResult SendTwoUser(UserName user1,UserName user2)

{

//直接返回原对象，未进行异常处理封装，请参考标准通讯模式

return Json(user2);

}

### 传递对象列表到服务端，并返回对象列表

传递一个对象集合到服务端，并返回一个对象集合

#### JS代码

//传递对象列表到服务端，并返回对象列表

$("#btn9").click(function () {

//数据需要与后台的UserName对象的字段一一对应，且参数名称userNameList需要与后台GetUserList2中的一致

//1.json字符串

var jsonString = '{"userNameList":[{ "FirstName": "323", "LastName": "2323" },{ FirstName: "444", LastName: "4444" }]}';

//2.json对象

var jsonObject = { "userNameList": [{ "FirstName": "323", "LastName": "2323" }, { FirstName: "444", LastName: "4444"}] };

var jsonString2 = JSON.stringify(jsonObject);

//3.js对象(推荐使用）

var userName1 = new Object();

userName1.FirstName = "323";

userName1.LastName = "2323";

var userName2 = new Object();

userName2.FirstName = "444";

userName2.LastName = "4444";

var userNameArray = new Array();

userNameArray[0] = userName1;

userNameArray[1] = userName2;

var jsObject = new Object();

jsObject.userNameList = userNameArray;

var jsonString3 = JSON.stringify(jsonObject);

$.ajax({

type: "post",

url: "Ajax1/GetUserList2",

data: jsonString3,

dataType: "json",

contentType: "application/json; charset=utf-8",

beforeSend: function (XMLHttpRequest) {

//此处可以用来显示加载动画

},

success: function (context, textStatus) {

$("#txt9").attr("value", context[1].FirstName);

},

complete: function (XMLHttpRequest, textStatus) {

//此处用来隐藏加载动画

},

error: function () {

//出错处理

}

});

});

#### C#代码

[HttpPost]

public JsonResult GetUserList2(List<UserName> userNameList )

{

return Json(userNameList);

}

### 传递一个对象及一个对象列表到服务端，并返回对象列表

典型场景：主从表

在我们对一个主从表进行操作时，常常需要在前后台传递一个主表对象及其从表对象集合，此时需要对主从表对象分别进行封装，下面代码中展示了3种封装方法，推荐使用第3种js对象封装方法。

#### JS代码

//6.传递一个对象及一个对象列表到服务端，并返回对象列表(典型场景：主从表)

$("#btn10").click(function () {

//数据需要与后台的UserName对象的字段一一对应，且参数名称userNameList需要与

//后台GetUserList3中的一致

//对应后台Controller中的方法：

//public JsonResult GetUserList3(UserName user,List<UserName> userNameList)

//只要对象类型、名称匹配，系统会自动帮我们把json字符串转换为.Net对象

//如本例，系统会将"user"转换为.Net中的UserName对象

//将"userNameList"转换为.Net中的List<UserName>

//1.json字符串，可以直接传递给后台

var jsonString = '{"user":{"FirstName": "aaa", "LastName": "bbb"},"userNameList":[{ "FirstName": "323", "LastName": "2323" },{ "FirstName": "444", "LastName": "4444" }]}';

//2.json对象，需要转换成json字符串才能传递到后台，因为http通讯都是传递字符，不可能传递一个js对象到后台去

var jsonObject = { "user": { "FirstName": "aaa", "LastName": "bbb" }, "userNameList": [{ "FirstName": "323", "LastName": "2323" }, { "FirstName": "444", "LastName": "4444"}] };

var jsonString2 = JSON.stringify(jsonObject); //使用JSON.stringfy帮助方法将一个json对象转换为字符串

//3.js对象(推荐使用）

var userName1 = new Object();

userName1.FirstName = "323";

userName1.LastName = "2323";

var userName2 = new Object();

userName2.FirstName = "444";

userName2.LastName = "4444";

var userNameArray = new Array();

userNameArray[0] = userName1;

userNameArray[1] = userName2;

var jsObject = new Object();

var userName = new Object();

userName.FirstName = "aaa";

userName.LastName = "bbb";

jsonObject.user = userName;

jsObject.userNameList = userNameArray;

var jsonString3 = JSON.stringify(jsonObject);

$.ajax({

type: "post",

url: "Ajax1/GetUserList3",

data: jsonString3,

dataType: "json",

contentType: "application/json; charset=utf-8",

beforeSend: function (XMLHttpRequest) {

//此处可以用来显示加载动画

},

success: function (context, textStatus) {

$("#txt10").attr("value", context[1].FirstName);

},

complete: function (XMLHttpRequest, textStatus) {

//此处用来隐藏加载动画

},

error: function () {

//出错处理

}

});

});

#### C#代码

//请注意参数定义，与页面中的js代码需要保持匹配

[HttpPost]

public JsonResult GetUserList3(UserName user,List<UserName> userNameList)

{

return Json(userNameList);

}

## 公共控件

### 公司选择对话框

#### 函数定义

//显示公司选择对话框

$smt.showCompanyDialog(multiSelect, callback)

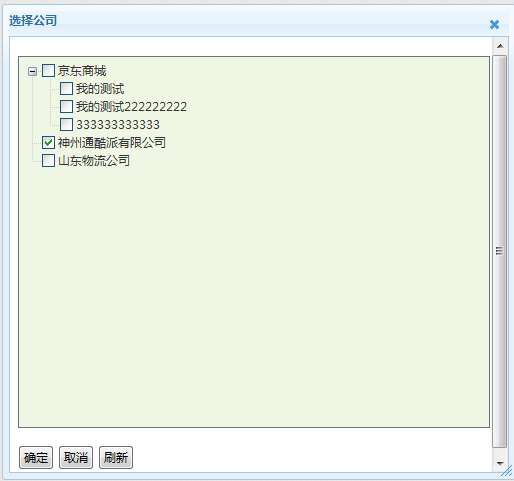
* multiSelect：是否可以多选
* callback：对话框关闭后的回调，用于返回选择的数据。

#### 典型UI

典型的UI如下：



一个文本框配合一个查看按钮，当点击查看按钮时，弹出选择公司的对话框：



当选择后，点击确定，返回具体的公司信息。

#### HTML代码

<p>选择公司:

<input id="txt\_showcompany" name="txt\_showcompany" type="text"/>

<input id="btn4\_showcompany" class="searchbutton" type="button"/>

</p><br/>

定义一个文本框和一个点击按钮。

#### JS代码

定义按钮点击事件：

//弹出公司选择对话框

$("#btn4\_showcompany").click(function () {

//true为多选，false为单选，showCompanyCallback为回调

$smt.showCompanyDialog(true, showCompanyCallback);

});

定义回调方法：

//公司对话框的回调

function showCompanyCallback(data) {

if (data == null)

return;

//单选时返回的data为单个对象

//属性为id,name,pId（父id）,type(1company,2department,3post,4employee)

//var str = data.id + " " + data.name + " " + data.type + " " + data.pId;

//多选时返回的data为对象数组

var str = "";

for (var i = 0; i < data.length; i++) {

var cnode = new Object();

cnode.id = data[i].id;

cnode.name = data[i].name;

cnode.type = data[i].type;

cnode.pId = data[i].pId;

str += cnode.name + " ";

}

$("#txt\_showcompany").val(str);

}

### 部门选择对话框

#### 函数定义

//显示部门选择对话框

$smt.showDepartmentDialog(multiSelect, callback)

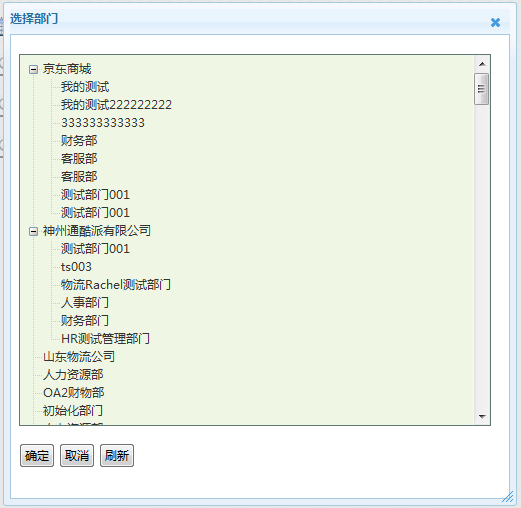
* multiSelect：是否可以多选
* callback：对话框关闭后的回调，用于返回选择的数据。

#### 典型UI

典型的UI如下：



一个文本框配合一个查看按钮，当点击查看按钮时，弹出选择部门的对话框：



当选择后，点击确定，返回具体的部门信息。

#### HTML代码

<p>选择部门:

<input id="txt\_showdept" name="txt\_showdept" type="text"/>

<input id="btn4\_showdept" class="searchbutton" type="button"/>

</p><br/>

定义一个文本框和一个点击按钮。

#### JS代码

定义按钮点击事件：

//弹出部门选择对话框

$("#btn4\_showdept").click(function () {

//true为多选，false为单选，showDepartmentCallback为回调

$smt.showDepartmentDialog(false, showDepartmentCallback);

});

定义回调方法：

//部门对话框的回调

function showDepartmentCallback(data) {

if (data == null)

return;

//单选时返回的data为单个对象

//属性为id,name,pId（父id）,type(1company,2department,3post,4employee)

var str = data.id + " " + data.name + " " + data.type + " " + data.pId;

//多选时返回的data为对象数组

// var str = "";

// for (var i = 0; i < data.length; i++) {

// var cnode = new Object();

// cnode.id = data[i].id;

// cnode.name = data[i].name;

// cnode.type = data[i].type;

// cnode.pId = data[i].pId;

// str += cnode.name + " ";

// }

$("#txt\_showdept").val(str);

}

### 员工选择对话框

#### 函数定义

//显示员工选择对话框

$smt.showEmployeeDialog(multiSelect, callback)

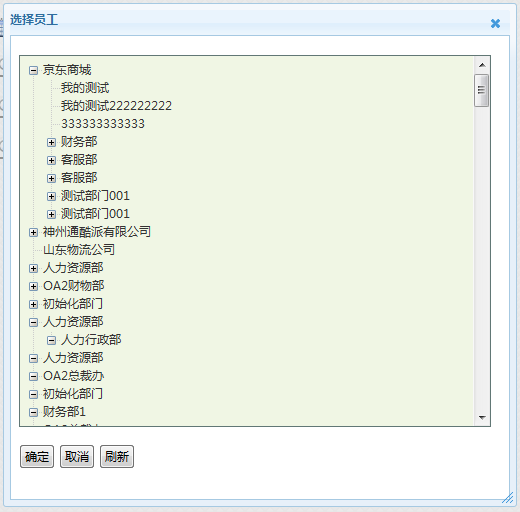
* multiSelect：是否可以多选
* callback：对话框关闭后的回调，用于返回选择的数据。

#### 典型UI

典型的UI如下：



一个文本框配合一个查看按钮，当点击查看按钮时，弹出选择公司的对话框：



当选择后，点击确定，返回具体的员工信息。

#### HTML代码

<p>选择员工:

<input id="txt\_showemp" name="txt\_showemp" type="text"/>

<input id="btn4\_showemp" class="searchbutton" type="button"/>

</p><br/>

定义一个文本框和一个点击按钮。

#### JS代码

定义按钮点击事件：

//弹出员工选择对话框

$("#btn4\_showemp").click(function () {

//true为多选，false为单选，showEmployeeCallback为回调

$smt.showEmployeeDialog(false, showEmployeeCallback);

});

定义回调方法：

//员工对话框的回调

function showEmployeeCallback(data) {

if (data == null)

return;

//单选时返回的data为单个对象

//属性为id,name,pId（父id）,type(1company,2department,3post,4employee)

//var str = data.id + " " + data.name + " " + data.type + " " + data.pId;

//多选时返回的data为对象数组

var str = "";

for (var i = 0; i < data.length; i++) {

var cnode = new Object();

cnode.id = data[i].id;

cnode.name = data[i].name;

cnode.type = data[i].type;

cnode.pId = data[i].pId;

str += cnode.name + " ";

}

$("#txt\_showemp").val(str);

}

### 日期控件

#### 函数定义

$("#datepicker ").datepicker();

在文本框上调用datepicker()方法。

#### 典型UI



点击右侧图标，弹出日期对话框：



选择后，日期返回到对话框中。

#### HTML代码

<p>选择日期: <input type="text" id="datepicker " /></p><br/>

#### JS代码

无

## 模式/非模式对话框编码规范

### 函数定义

#### 打开对话框

//根据页面上预定义的对话框id弹出此对话框，如果找不到此id对应的对话框，那么新增一个

$smt.openDialog(dId, dTitle, dHeight, dWidth, dUrl, dData, callback, isModel)

* dId：定义对话框的id，必须全局唯一
* dTitle：定义对话框的标题
* dHeight：定义对话框的高度
* dWidth：定义对话框的宽度
* dUrl：定义对话框加载的页面Url
* dData：要传入的数据，格式为：key1=value1&key2=value2&key3=value3
* callback：对话框关闭时调用的回调
* isModel：是否为模式对话框

**请勿大量新增对话框，而应该将常用的都封装起来，如：统一定义“subjectdialog”作为**

**科目对话框的id，所有需要科目对话框的地方，都应该如此调用：**

**$smt.openDialog("subjectDialog", "科目选择", "350", "500", "WP/Ajax2/ShowSubjectDialog", "");**

#### 关闭对话框

//关闭通过$smt.openDialog方法打开的指定id的窗口

$smt.closeDialog(dId, data)

* dId：要关闭的对话框的id
* data：要返回的数据，打开对话框时定义的回调会接收此数据

**注意：返回的数据可以是任何格式，对应在回调中要按传递的格式进行解析。**

#### 准备返回数据

//在关闭对话框前，准备好要返回的数据

$smt.prepareDialogReturnData(dId, data)

* dId：要关闭的对话框的id
* data：要返回的数据，打开对话框时定义的回调会接收此数据

当用户点击对话框的右上角“X”图标，或者按键盘“ESC”，都会关闭对话框。

一个典型的场景是：

对话框中有一个关闭按钮，关闭按钮调用closeDialog方法来关闭对话框，此时可以通过closeDialog把想要传递的数据返回回去；但是当用户直接关闭或者按“ESC”时，对话框会直接关闭，并未通过我们定义的closeDialog方法来关闭，此时就无法将数据传递回去。

为了解决这个问题，可以在获取到返回的数据后，马上调用prepareDialogReturnData方法，提前准备好返回的数据，之后无论如何关闭，打开对话框时定义的回调都会接收到我们prepare的数据。

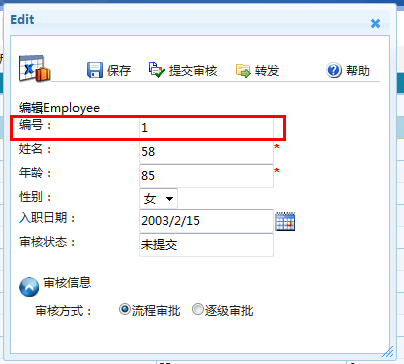
### 典型UI

Grid-Detail模式中的新增/编辑/查看对话框

以编辑为例：



选择编号1的数据进行编辑，点击修改按钮，弹出修改对话框，加载修改页面到对话框中，并把编号=1传递到对话框中。



当保存后，关闭对话框时，传递一个已修改的信息给Grid页面，提示需要进行数据刷新。

### HTML代码

无

### JS代码

#### Grid页面定义点击事件

//定义工具栏-编辑按钮事件

$(btnEditId).click(function () {

callEditAction();

return false;

});

function callEditAction() {

var $rowid = $(gridId).jqGrid('getGridParam', 'selrow');

if ($rowid == null) {

alert("请先选择一条记录");

return;

}

**var paramString = "id=" + $rowid;//传递给Edit页面的数据**

var actionUrl = "Test/Ajax3/Edit";

var title = "Edit";

var height = "600";

var width = "800";

$smt.showDetailDialog("TestAjax3\_editdialog", title, height, width, actionUrl, paramString, editCallback);

}

#### Grid页面定义回调函数

function editCallback(data) {

if (data[0] == "1") {

refreshEmpGrid();

}

}

#### Edit页面返回数据

//当更新数据成功时才告诉Grid页面需要刷新页面

function saveData() {

var jsonEmployee = JSON.stringify(getEmployee());

$.ajax({

type: "post",

url: "Ajax3/UpdateEmployee",

data: jsonEmployee,

dataType: "json",

contentType: "application/json; charset=utf-8",

beforeSend: function(XMLHttpRequest) {

//此处可以用来显示加载动画

},

success: function(context, textStatus) {

if (context.Result == "OK") {

alert("update success!");

var data = new Array();

data[0] = "1";

//var data = new Array{"1"};

$smt.prepareDialogReturnData("TestAjax3\_editdialog", data);

} else {

//2.前后台通讯正常，但是后台逻辑发生错误，不能返回正确的数据

alert(context.Message);

}

},

complete: function(XMLHttpRequest, textStatus) {

//此处用来隐藏加载动画

},

error: function() {

//出错处理

alert("error comm");

}

});

}

### C#代码

后台控制器代码：

[HttpPost]

[AjaxValidateAntiForgeryToken]

public ActionResult Edit(int id)

{

//传递数据到视图中

ViewBag.CustomMessage = "这是一个编辑窗口";

var emp = GetEmpById(id);

if (emp == null)

{

ViewBag.ModelIsNull = "true";

ViewBag.ErrorMessage = "无法找到ID为" + id + "的记录!";

emp = new Employee();

emp.StartDate = DateTime.Now;

}

return PartialView("Edit", emp);

}

## Grid编码规范

### 函数定义

#### 定义HTML

<div id="gridcontainerid" >

<table id="gridid" border="0" cellspacing="0" cellpadding="0"></table>

<div id="pageid" ></div>

</div>

定义一个Table及一个PageDIV即可。

一般外面再添加一个容器DIV。

#### 初始化

$("gridid").jqGrid({参数});

常用参数如下：

|  |  |
| --- | --- |
| 参数 | 描述 |
| url | tells us where to get the data. Typically this is a server-side function with a connection to a database which returns the appropriate information to be filled into the Body layer in the grid  从后台获取要显示数据的地址 |
| datatype | this tells jqGrid the type of information being returned so it can construct the grid. In this case we tell the grid that we expect xml data to be returned from the server, but other formats are possible. For a list of all available datatypes refer to API Methods  从服务器后台要返回的数据的类型，可以为json、xml等格式，本例中使用json |
| mtype | tells us how to make the ajax call: either ‘GET’ or ‘POST’. In this case we will use the GET method to retrieve data from the server  获取数据使用的方法，使用get或者post，mmorpg为get，本例使用get |
| colNames | an array in which we place the names of the columns. This is the text that appears in the head of the grid (Header layer). The names are separated with commas  为一个数组，用来显示的头部，在本例就是流水号、姓名等。 |
| colModel | an array that describes the model of the columns. This is the most important part of the grid. Here I explain only the options used above. an array that describes the model of the columns. This is the most important part of the grid. Here I explain only the options used above. For the complete list of options see colModelAPI **name** the name of the column. This name does not have to be the name from database table, but later we will see how we can use this when we have different data formats **index** the name passed to the server on which to sort the data (note that we could pass column numbers instead). Typically this is the name (or names) from database – this is server-side sorting, so what you pass depends on what your server expects to receive 索引，即该列使用从服务器返回的数据 **width** the width of the column, in pixels 使用像素来表示宽度，width **align** the alignment of the column 对齐方式 **sortable** pecifies if the data in the grid can be sorted on this column; if false, clicking on the header has no effect是否可以排列  该元素为核心元素，各种配置均在这边 |
| pager | defines that we want to use a pager bar to navigate through the records. This must be a valid html element; in our example we gave the div the id of “pager”, but any name is acceptable. Note that the Navigation layer (the “pager” div) can be positioned anywhere you want, determined by your html; in our example we specified that the pager will appear after the Body layer.  指定显示页数的层id |
| rowNum | sets how many records we want to view in the grid. This parameter is passed to the url for use by the server routine retrieving the data  默认显示的每页行数 |
| rowList | an array to construct a select box element in the pager in which we can change the number of the visible rows. When changed during the execution, this parameter replaces the rowNum parameter that is passed to the url  可以选择的行数，例如10、20、30等 |
| sortname | sets the initial sorting column. Can be a name or number. This parameter is added to the url for use by the server routine  默认排序的列 |
| viewrecords | defines whether we want to display the number of total records from the query in the pager bar  是否显示总记录数 |
| caption | sets the caption for the grid. If this parameter is not set the Caption layer will be not visible  标题 |

### 典型UI



### HTML代码

<div id="gridcontainerid" >

<table id="gridid" border="0" cellspacing="0" cellpadding="0"></table>

<div id="pageid" ></div>

</div>

### JS代码

#### 初始化

//通过Ajax调用来获取后台数据填充到Grid的Url

var loadDataUrl = "**Ajax3/LoadData**";

$("gridid").jqGrid({

url: loadDataUrl,

//指定界面上显示的列名

colNames: ["编号", "姓名", "年龄", "性别", "入职日期", "审核状态"],

//配置列映射，指定映射到实体的字段名称，索引等等

colModel: [

{ name: "Id", index: "Id", align: "left" },

{ name: "Name", index: "Name", align: "left" },

{ name: "Age", index: "Age", align: "left" },

{ name: "Sex", index: "Sex", align: "left" },

{ name: "StartDate", index: "StartDate", align: "left" },

{ name: "AuditState", index: "AuditState", align: "left" }

],

ondblClickRow: function () {

callDetailsAction(); //双击行时触发查看动作

},

//rownumbers:true, //是否显示行号

pager: $("pageid"), //分页器设置

sortname: "Id", //默认的排序字段和方向

sortorder: "asc"

}).navGrid(pagerId,

{ refresh: false, add: false, edit: false, del: false, search: false },

{}, //edit设置

{}, //add设置

{}, //delete设置

{} //search设置

);

#### 条件查询

//根据条件进行查询，查询结果填充到Grid

//指定3个查询参数，对应后台的LoadData方法

function doQuery() {

var paramsString = "?";

paramsString += "age=" + $("#querybar\_age").attr("value");

paramsString += "&sex=" + $("#querybar\_sex").attr("value");

paramsString += "&audit=" + $(selAuditId).attr("value");

var queryUrl = loadDataUrl + paramsString;

$("gridid").jqGrid('setGridParam',

{ url: queryUrl, page: 1 }).trigger("reloadGrid");

}

#### 刷新

//刷新Grid

function refreshEmpGrid() {

$("gridid").trigger("reloadGrid");

}

### C#代码

后台控制器中，提供一个数据查询的方法，以下为例：

/// <summary>

/// jqGrid调用的方法入口，根据传入的参数返回数据，

/// 【sidx,sord,page,rows】这4个参数名称不能修改

/// **本例中，增加age\sex\audit3个自定义参数**

/// </summary>

/// <param name = "sidx">指定排序字段</param>

/// <param name = "sord">指定排序方向</param>

/// <param name = "page">指定页（返回第几页的数据）</param>

/// <param name = "rows">指定每页几行数据</param>

/// <returns>返回根据参数获取的符合jqGrid格式的数据</returns>

[HttpPost]

public ActionResult LoadData(string sidx, string sord, int page, int rows,

**string age, string sex, string audit**)

{

//对Grid的4个标准查询参数进行了再封装

QueryParams queryParams = new QueryParams()

{

OrderField = sidx,

OrderDirection = sord,

Page = page,

PageRows = rows

};

//调用后台的真实数据库查询方法，获取分页后的数据

//此处请使用业务层的相关方法，返回指定页的数据

QueryResult<Employee> queryResult = GetEmployees(queryParams, **age, sex, audit**);

//转换为jqGrid需要的数据格式，结构为“{id,cell{key1:value1,key2:value2......}}”的数组，//其中“id/cell"为固定名称

var jqData = (from s in queryResult.Data

select new

{

id = s.Id,//定义当前行数据的一个传输识别名称，一般以主键为名称，可以省略

cell = new object[]

{

s.Id,

s.Name,

s.Age,

s.Sex,

s.StartDate.ToString("yyyy/MM/dd"),

s.AuditState

}

}).ToArray();

//以json格式传输，依次以”total/page/records/rows"封装，名称固定

var jsonData = new

{

total = queryResult.TotalPages,

page,

records = queryResult.TotalRecords,

rows = jqData

};

return Json(jsonData);

}

## 审核控件编码规范

### 实体说明

#### AuditInfo

public class AuditInfo

{

/// <summary>

///系统代号(必须字段)

/// </summary>

public string SystemCode

{

get;

set;

}

/// <summary>

/// 模块代号(必须字段)

/// </summary>

public string ModelCode

{

get;

set;

}

/// <summary>

///实体主键名（用于更改业务系统审核状态标识）

/// </summary>

public string EntityKey

{

get;

set;

}

/// <summary>

/// 表单ID

/// </summary>

public string FormID

{

get;

set;

}

/// <summary>

/// 流程类型 0:固定流程 1：自选流程

/// </summary>

public int FlowType

{

get;

set;

}

/// <summary>

/// 当前审核人

/// </summary>

public UserInfo CurrentApprovalUser

{

get;

set;

}

/// <summary>

/// 下一审核人（自选流程）

/// </summary>

public UserInfo NextApprovalUser

{

get;

set;

}

/// <summary>

/// 审核意见

/// </summary>

public string ApprovalContent

{

get;

set;

}

/// <summary>

///审核动作 通过/不通过

/// 0:审核通过

/// 1:审核不通过

/// </summary>

public int ApprovalResult

{

get;

set;

}

public string NextStateCode

{

get;

set;

}

/// <summary>

/// 提交标识

/// 首次提交 0

/// 审核中：1

/// 撤销：2

/// </summary>

public int SubmitFlag

{

get;

set;

}

### 函数定义

#### 公共审核JS

|  |
| --- |
| //cname ：控制器名称 如PublicControlController.cs cname是PublicControl  //aId:AuditControl ID 现规范使用实体主键ID值  // aState:0流程未提交状态，1流程属于已提交状态  //formid: 表单ID  //modelcode:模块代号与流程代号一致 |

showAudit: function (cname,aId, aState, formid, modelcode) {

｝

|  |
| --- |
| // aurl：Url地址（如PlanDraft/SubmitFlow），可以是Controller名称  // aId: AuditControl ID 现规范使用实体主键ID值  // aState: 0流程未提交状态，1流程属于已提交状态  // auditInfo:审核对象  // callback:审核成功回调函数  // selectCallBack:一个角色多人情况下，进行选人函数 |

submitAudit: function (/\*string\*/aurl, aId, aState, auditInfo, callback, selectCallBack) {

}

#### 定义HTML



|  |
| --- |
| @SmtHelper.AuditControl(“主键ID值”) |

定义一个AuditControl在form后面

#### 初始化JS代码

##### Create.cshtml

1.初始化按钮时，隐藏提交按钮，编写提交审核函数

function configButton() {

$("#topbar-submit").hide();

$("#topbar-zf").hide();

$smt.hideAudit("主键ID值");

});

function submitData() {

var obj = new Object();

obj.FormID = PlanDrafFormID;

obj.ModelCode = PlanDraftModelCode;

$smt.submitAudit("PlanDraft/SubmitFlow", "@Model.FICTIONID", "0", obj, submitCallback, SelectCallback);

}

submitCallback：提交完成后回调函数

SelectCallback： 一角色多成员情况下选人回调函数

2.ajax提交保存成功后，显示审核控件

success: function (context, textStatus) {

if (context.Result == "OK") {

PlanDrafFormID = context.Data[0];

PlanDraftModelCode = context.Data[1];

var data = new Array();

data[0] = "1";

$smt.prepareDialogReturnData("@ControllerName" + "\_createdialog", data);

$smt.showAudit("@ControllerName", "@Model.FICTIONID", "0", PlanDrafFormID, PlanDraftModelCode);

$("#topbar-submit").show();

}

}

##### Edit.cshtml

1. 修改状态下，初始化AuditControl与新增一致
2. configButton() 时无需隐藏按钮，增加显示审核控件JS

$smt.showAudit("@ControllerName", “表单ID”, "0", “表单ID”,”模块代号”);

##### Audit.cshtml

1. 审核界面初始化AuditControl

@SmtHelper.AuditControl(Model.FICTIONID)

1. configButton() 时无需隐藏按钮，增加显示审核控件JS

### 后台Controller代码

后台Controller继承IAudit,并进行实现

|  |
| --- |
| public JsonResult SubmitFlow(AuditInfo auditInfo)  {  auditInfo.SystemCode = "OA";//系统代号  auditInfo.CurrentApprovalUser = new UserInfo  {  UserID = this.UserID,  CompanyID = this.CompanyID,  PostID = this.PostID,  DepartmentID = this.DeptID,  UserName = this.UserName  };    T\_WP\_PLANDRAFT planDraft= new T\_WP\_PLANDRAFT();  try  {  planDraft = ClientBLL.GetPlanDraftByID(auditInfo.FormID);  }  catch(Exception e)  {  return Json(new { State = "2", UserInfo = e.Message });  }  auditInfo.EntityKey = "FICTIONID";  AuditResult result = AuditFlow<T\_WP\_PLANDRAFT>.SubmitFlow(planDraft, auditInfo);  //判断审核结果，变更审核状态(0:未提交;1:审核中;2:审核通过;3:审核未通过;4:待审核")  if (result.State != 0 || result.UserInfo == null)// 要选择一个角色下人员时，无需更新审核状态 {  planDraft.CHECKSTATE = result.CheckState;  var vResult = ClientBLL.UpdatePlanDraft(planDraft);  }  return Json(new { State = result.State, UserInfo = result.UserInfo });  } |

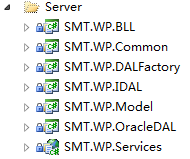
### 典型UI





# 后台开发指南

后台服务主要包括数据模型实体层、数据访问层、数据访问接口层、数据工厂、公共服务层、业务逻辑层、对外服务层。以下为实例参考图：



## 业务逻辑层

### 模块职能

负责处理项目业务逻辑功能，主要与数据访问接口层、数据工厂、数据模型实体层交互，通过数据接口获取数据。

### 规范说明

1. 层命名规范为SMT+业务系统名称简称+BLL；
2. 类命名为业务表名+BLL,如：”T\_WP\_COMPNORMDEFINEBLL.cs”,” T\_WP\_COMPNORMDEFINE”为业务表名；
3. 通过实例化抽象工厂接口方法来实现业务层数据库访问接口调用；
4. 数据分页显示时，需采用系统提供的DataPage实体泛型方法。

### 范例参看

1. 实例化数据库访问接口

private static readonly IT\_WP\_COMPNORMDEFINEDAL compNormDefineDAL = SMT.WP.DALFactory.DataAccess.CreateCompNormDefineDAL();

1. 与接口方法调用

public IList<T\_WP\_COMPNORMDEFINE> GetCompNormDefine(string strNormCode, string strNormName)

{

return compNormDefineDAL.GetCompNormDefine(strNormCode, strNormName);

}

1. 分页泛型DataPage调用

public DataPage<T\_WP\_COMPNORMDEFINE> ListCompNormDefine(DataPage<T\_WP\_COMPNORMDEFINE> dataPage, string strNormCode, string strNormName)

{

return compNormDefineDAL.ListCompNormDefine(dataPage,strNormCode,strNormName);

}

## 公共服务层

### 模块职能

公共服务层主要包括日志处理、异常机制、缓存等服务层公共服务信息。

### 规范说明

1. 层命名规范为SMT+业务系统名称简称+Common
2. 公共服务类编码取名为“Server”+公共服务功能模块，如记录后台日志信息，取名为：“ServerLogger.cs”。
3. 公共服务类中类及方法中需加相应的备注信息

### 范例参看

日志处理ServerLogger类，如下图所示：

public class ServerLogger

{

/// <summary>

/// 日志

/// </summary>

private static ILog serverLog;

/// <summary>

/// 初始化

/// </summary>

static ServerLogger()

{

XmlConfigurator.Configure();

if (serverLog == null)

{

serverLog = LogManager.GetLogger("WPServerLog");

}

}

/// <summary>

/// 写普通信息

/// </summary>

/// <param name="msg">消息</param>

public static void WriteInfo(string msg)

{

serverLog.Info(msg);

}

/// <summary>

/// 写错误信息

/// </summary>

/// <param name="msg">信息</param>

public static void WriteDebug(string msg)

{

serverLog.Debug(msg);

}

}

## 数据工厂

### 模块职能

通过配置文件及反身机制，把本项目的数据库的连接做成一个抽象的工厂。

### 规范说明

1. 层命名规范为SMT+业务系统名称简称+ DALFactory；
2. 类取名为“DataAccess.cs”，一般为”sealed”类；
3. DAL访问类型参数取名为SMT+业务系统名称简称+ DAL，如：



1. 数据库访问层path值通过配置文件获取

private static readonly string path = ConfigurationManager.AppSettings["SMT.WP.DAL"];

1. 接口通过反射机制实现

### 范例参看

创建公司经营指标定义接口

public static SMT.WP.IDAL.IT\_WP\_COMPNORMDEFINEDAL CreateCompNormDefineDAL()

{

string className = path + ".T\_WP\_COMPNORMDEFINEDAL";

return (SMT.WP.IDAL.IT\_WP\_COMPNORMDEFINEDAL)Assembly.Load(path).CreateInstance(className);

}

## 接口层

### 模块职能

定义项目的业务方法, 由逻辑层实现；

### 规范说明

1. 层命名规范为SMT+业务系统名称简称+ IDAL；
2. 类取名为I+业务表名+BLL,如：“IT\_WP\_COMPNORMDEFINEDAL”，“T\_WP\_COMPNORMDEFINE”为业务表名；

### 范例参看

1. 定义接口

/// <summary>

/// 公司经营指标定义接口

/// </summary>

public interface IT\_WP\_COMPNORMDEFINEDAL

1. 接口方法定义

/// <summary>

///根据ID获取公司经营指标定义信息

/// </summary>

/// <param name=" strID ">定义ID</param>

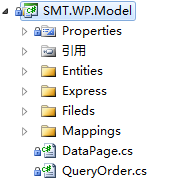
/// <returns></returns>

T\_WP\_COMPNORMDEFINE GetCompNormDefineByID(string strID);

## 实体层

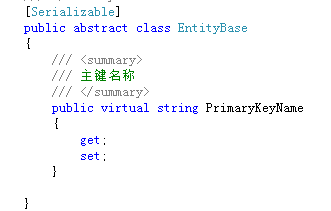
### 模块职能

提供统一的内部访问接口,主要由实体类、实体层映射、字段信息三部分组成，另外包含DataPage及QueryOrder，基类可根据各项目具体情况增减。实体层中，由干净的实体组成，不含任何业务信息，层参看图如下：

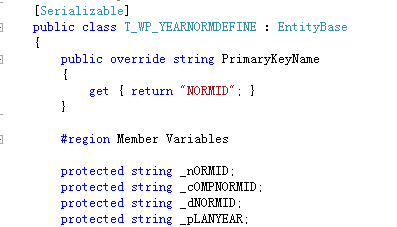


### 规范说明

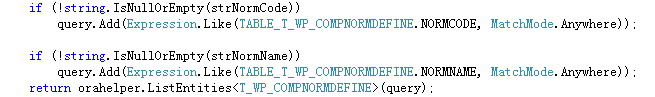
1. 层命名规范为SMT+业务系统名称简称+Model；
2. EntityBase基类，所有实体类的基类，目前基类包含主键名称，各项目可根据相关业务逻辑扩展基类，基类前需加[Serializable]关键字，参看示例如下图：



1. 实体类一般放在Entities文件夹下，实体类取名与业务表名一致，如” T\_WP\_YEARNORMDEFINE”,“T\_WP\_YEARNORMDEFINE”为业务表名，实体类前需加[Serializable]关键字，实体类继承EntityBase基类，并重写主键属性，参看示例如下图：



1. 字段信息，数据库表字段信息一般放在Fileds文件夹下，类取名为T+业务系统名称简称+ TableFields,以结构形式存储表字段信息，表结构名一般取名为”TABLE”+业务表名,项目中需用到字段时，需用字段变量处理,如：



1. 实体层映射(hibernate-mapping)，实体层映射放在Mappings文件夹下，实体类及实体层映射不包括子父表关联信息，需手动在对应的hbm文件中增加；
2. DataPage基类，内部封装基类，提供UI层分页基类；
3. QueryOrder基本，内部封装基类，提供UI层查询排序基类。

### 范例参考

1. Entities：

namespace SMT.WP.Model

{

#region T\_WP\_YEARNORMDRAFT

/// <summary>

/// T\_WP\_YEARNORMDRAFT object for NHibernate mapped table 'T\_WP\_YEARNORMDRAFT'.

/// </summary>

[Serializable]

public class T\_WP\_YEARNORMDRAFT : EntityBase

{

public override string PrimaryKeyName

{

get { return "DNORMID"; }

}

#region Member Variables

protected string \_dNORMID;

protected string \_pLANYEAR;

protected string \_cOMPANYID;

protected string \_cOMPANYNAME;

#endregion

#region Constructors

public T\_WP\_YEARNORMDRAFT() { }

public T\_WP\_YEARNORMDRAFT(string pLANYEAR, string cOMPANYID, string cOMPANYNAME

{

this.\_pLANYEAR = pLANYEAR;

this.\_cOMPANYID = cOMPANYID;

this.\_cOMPANYNAME = cOMPANYNAME;

}

#endregion

#region Public Properties

public virtual string DNORMID // virtual必填

{

get { return \_dNORMID; }

set

{

if (value != null && value.Length > 50)

throw new ArgumentOutOfRangeException("无效值：", value, value.ToString());

\_dNORMID = value;

}

}

public virtual string PLANYEAR // virtual必填

{

get { return \_pLANYEAR; }

set

{

if (value != null && value.Length > 4)

throw new ArgumentOutOfRangeException("无效值：", value, value.ToString());

\_pLANYEAR = value;

}

}

public virtual string COMPANYID // virtual必填

{

get { return \_cOMPANYID; }

set

{

if (value != null && value.Length > 50)

throw new ArgumentOutOfRangeException("无效值：", value, value.ToString());

\_cOMPANYID = value;

}

}

public virtual string COMPANYNAME // virtual必填

{

get { return \_cOMPANYNAME; }

set

{

if (value != null && value.Length > 400)

throw new ArgumentOutOfRangeException("无效值：", value, value.ToString());

\_cOMPANYNAME = value;

}

}

}

2. Mappings：

<?xml version="1.0" encoding="utf-8" ?>

<hibernate-mapping xmlns="urn:nhibernate-mapping-2.2">

<class name="SMT.WP.Model.T\_WP\_YEARNORMDRAFT, SMT.WP.Model" table="T\_WP\_YEARNORMDRAFT">

<id name="DNORMID" type="String" unsaved-value="null">

<column name="DNORMID" length="50" sql-type="VARCHAR2" not-null="true" unique="true" index="PK\_T\_WP\_YEARNORMDRAFT"/>

<generator class="assigned" />

</id>

<property name="PLANYEAR" type="String">

<column name="PLANYEAR" length="4" sql-type="VARCHAR2" not-null="true" index="IDX\_1T\_WP\_YEARNORMDRAFT"/>

</property>

<property name="COMPANYID" type="String">

<column name="COMPANYID" length="50" sql-type="VARCHAR2" not-null="true" index="IDX\_1T\_WP\_YEARNORMDRAFT"/>

</property>

<property name="COMPANYNAME" type="String">

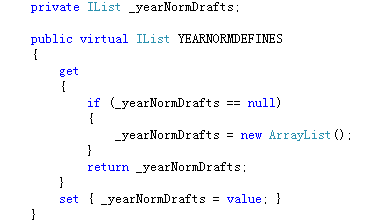
<column name="COMPANYNAME" length="400" sql-type="NVARCHAR2" not-null="false"/>

</property>

</class>

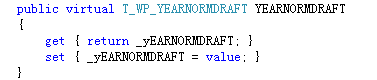
</hibernate-mapping>

1. 子父表操作，父表需实体需增加



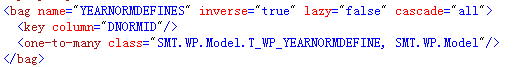
1. 子父表操作，子表实体需增加

// T\_WP\_YEARNORMDRAFT 为父表实体



1. 子父表操作，实体层映射层hbm 文件父表文件增加

//T\_WP\_YEARNORMDEFINE 为子表实体



1. 子父表操作，实体层映射层hbm 文件子表文件增加

// T\_WP\_YEARNORMDRAFT为父表实体



**注：框架组为实体类及hbm实体层映射开发基于CodeSmith生成模板，模板放在框架项目文件下，可通过模板自动生成实体类、映射体及表字段信息。**

## 数据访问层

### 模块职能

提供访问后台数据库,分为数据库访问基类及业务访问类两部分。

### 规范说明

1. 层命名规范为SMT+业务系统名称简称+ 当前使用数据库简称+DAL，如: SMT.WP.OracleDAL；
2. 基类取名为当前使用数据库简称+Helper,如：OracleHelper，基类使用说明章节请参考“数据库访问开发指南”；
3. 业务访问类以数据库表名+DAL组成,访问类继承相应的接口,如：T\_WP\_COMPNORMDEFINEDAL : IT\_WP\_COMPNORMDEFINEDAL。

### 范例参看

1. OracleHelper实例化

private static readonly OracleHelper orahelper = new OracleHelper();

1. 查询公司经营指标定义信息

/// <summary>

///查询公司经营指标定义信息

/// </summary>

/// <param name="compNormDefine">经营指标实体</param>

/// <param name="strNormCode">科目代号</param>

/// <param name="strNormName">科目名称</param>

/// <returns></returns>

public IList<T\_WP\_COMPNORMDEFINE> GetCompNormDefine(string strNormCode, string strNormName)

{

List<ICriterion> query = new List<ICriterion>(); //Nhibnate查询条件定义

if (!string.IsNullOrEmpty(strNormCode))

query.Add(Expression.Like(TABLE\_T\_WP\_COMPNORMDEFINE.NORMCODE, MatchMode.Anywhere));

if (!string.IsNullOrEmpty(strNormName))

query.Add(Expression.Like(TABLE\_T\_WP\_COMPNORMDEFINE.NORMNAME, MatchMode.Anywhere));

return orahelper.ListEntities<T\_WP\_COMPNORMDEFINE>(query);

}

1. 获取分页公司经营指标定义信息

/// <summary>

///获取分页公司经营指标定义信息

/// </summary>

/// <param name="dataPage">泛型分页实体</param>

/// <param name="strNormCode">科目代号</param>

/// <param name="strNormName">科目名称</param>

/// <returns></returns>

public DataPage<T\_WP\_COMPNORMDEFINE> ListCompNormDefine(DataPage<T\_WP\_COMPNORMDEFINE> dataPage, string strNormCode, string strNormName)

{

List<ICriterion> query = new List<ICriterion>(); //Nhibnate查询条件定义

if (!string.IsNullOrEmpty(strNormCode))

query.Add(Expression.Like(TABLE\_T\_WP\_COMPNORMDEFINE.NORMCODE, strNormCode,MatchMode.Anywhere));

if (!string.IsNullOrEmpty(strNormName))

query.Add(Expression.Like(TABLE\_T\_WP\_COMPNORMDEFINE.NORMNAME, strNormName,MatchMode.Anywhere));

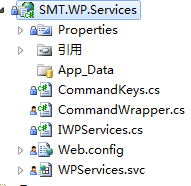
return orahelper.ListEntities<T\_WP\_COMPNORMDEFINE>(dataPage, query);

}

## WCF服务层

### 模块职能

将服务层的业务逻辑层封装，与通信层关联，以Object形式对外提供服务，层结构参看图如下：



### 规范说明

1. 层命名规范为SMT+业务系统名称简称+ Services；
2. 类CommandKeys.cs, 枚举，命令KEY列表，用于为客户端和服务端接口调用提供统一标识, 返回数据类型；
3. 类CommandWrapper，命令包装接口的实现，用于映射服务端接口的调用, InvokeCommand方法: 根据给定的参数请求匹配的命令, SearchCommandByKey：根据命令Key，寻找封装后的业务层逻辑方法；
4. 业务层中，每个类需通信时，需在CommandWrapper中增加一个CheckCommand方法，CheckCommand方法传值标准为object CheckCommand(CommandKeys commandkey, object Content, params string[] commandParams)。

### 范例参看

1. 计划申请CommandKeys枚举定义

#region 计划申请

PLANDRAFT\_E = 401, //获取单个实体

PLANDRAFT\_A = 402,// 新增

PLANDRAFT\_U = 403,//更新

PLANDRAFT\_D = 404,//删除

PLANDRAFT\_S = 405, //查询

PLANDRAFT\_SE = 406, //查询

PLANDRAFT\_AU = 407,//新增或更新

#endregion

1. CommandWrapper中SearchCommandByKey方法
2. private object SearchCommandByKey(CommandKeys commandkey, object Content, params string[] commandParams)

{

switch (commandkey)

{

case CommandKeys.COMPNORMDEFINE\_C: //CommandKeys标示Key判断

case CommandKeys.COMPNORMDEFINE\_R: //CommandKeys标示Key判断

case CommandKeys.COMPNORMDEFINE\_U: //CommandKeys标示Key判断

case CommandKeys.COMPNORMDEFINE\_F: //CommandKeys标示Key判断

case CommandKeys.COMPNORMDEFINE\_D: //CommandKeys标示Key判断

case CommandKeys.COMPNORMDEFINE\_H: //CommandKeys标示Key判断

return CheckCompNormDefineCommand(commandkey, Content, commandParams);

default:

break;

}

return null;

}

1. CheckCommand示例

/// <summary>

/// 检查集团公司经营指标定义

/// </summary>

/// <param name="commandkey">命名标示</param>

/// <param name="Content">实体对象</param>

/// <param name="commandParams">命令参数</param>

/// <returns></returns>

private object CheckCompNormDefineCommand(CommandKeys commandkey, object Content, params string[] commandParams)

{

T\_WP\_COMPNORMDEFINEBLL \_compNormDefineBLL = new T\_WP\_COMPNORMDEFINEBLL();

//实例化业务层类对象

switch (commandkey)

{

case CommandKeys.COMPNORMDEFINE\_R:

return \_compNormDefineBLL.GetCompNormDefineByID((string)Content); //单参数、参数非实体

case CommandKeys.COMPNORMDEFINE\_U:

return \_compNormDefineBLL.UpdateCompNormDefine((T\_WP\_COMPNORMDEFINE)Content); //单参数，参数为实体对象

case CommandKeys.COMPNORMDEFINE\_H:

{

if (commandParams.Length == 2)

return \_compNormDefineBLL.ListCompNormDefine((DataPage<T\_WP\_COMPNORMDEFINE>)Content, (string)commandParams[0], (string)commandParams[1]); //多参数，首参数为Content，其它参数commandParams自动处理

else

return false;

}

default:

return false;

}

}

# 前后台通讯开发指南

## 组件概述

### 说明

通讯组件主要的目的是同一托管通讯层的工作，实现客户端与服务端的统一访问与管理，通过对数据的通讯、加密、压缩等操作，在客户端与服务端通过对通讯组件的使用从而达到统一处理的目的。

用于提高通讯过程中的可靠性、安全性。

### 通讯原理

1. 客户端请求均由通讯组件包装为BaseObject对象，其中包含本次请求的命令标识，数据流等等；
2. 服务端返回由通讯组件包装的ResultObject对象，其中包含执行标志，结果数据流等等；
3. 服务端、客户端只需要关心业务，公布业务接口即可；
4. 开发人员实现通讯组件的接口，将不同的通讯对象映射到不同的业务接口即可。

### 程序集

程序集为：SMT.Portal.Communication.DLL

## 组件成员

|  |  |
| --- | --- |
| **成员名称** | **描述** |
| CommuncationManager | 通讯管理.统一处理通讯数据，客户端、服务端接口调用与数据处理 |
| ICommandWrapper | 接口，命令调用容器，用于托管对服务端命令的调用逻辑。 |
| IServicesProvider | 接口，用于在客户端提供服务接口的具体实现与调用 |
| BaseObject | 通信组件，数据传输的基对象，用于封装传输过程中的数据。 |
| ResultObject | 结果对象，继承自BaseObject，表示请求后返回的结果，其中包含了请求状态、和错误信息。 |

## 通讯对象

### BaseObject

namespace SMT.Portal.Communication

{

/// <summary>

/// 数据传输的基对象，用于封装传输过程中的数据。

/// 包含了命令标识、数据、请求参数

/// </summary>

[DataContract]

public class BaseObject

{

/// <summary>

/// 用户请求的Token，用于验证请求的合法性

/// </summary>

[DataMember]

public string Token { get; set; }

/// <summary>

/// 命令标识，用于在客户端和服务端映射对应的操作代码

/// </summary>

[DataMember]

public string CommandKey { get; set; }

/// <summary>

/// 数据，已经处理过的数据流。请求和返回的结果数据均存于此

/// </summary>

[DataMember]

public byte[] DataContent { get; set; }

/// <summary>

/// 请求参数列表

/// </summary>

[DataMember]

public string[] Params { get; set; }

}

}

### ResultObject

namespace SMT.Portal.Communication

{

/// <summary>

/// 结果对象，基础自BaseObject，表示请求后返回的结果，其中包含了请求状态、和错误信息。

/// </summary>

[DataContract]

public class ResultObject : BaseObject

{

/// <summary>

/// 请求状态，用于检测命令是否正确执行

/// </summary>

[DataMember]

public bool State { get; set; }

/// <summary>

/// 错误信息，用于包含自定义或者系统错误信息

/// </summary>

[DataMember]

public string ErrorMessage { get; set; }

}

}

## 服务端使用规范

### 服务端通讯服务实现

用于提供服务的通讯类应该实现如下签名的接口(以WCF为通讯技术，名称不限)：

/// <summary>

/// 服务端提供接口，用于在客户端提供服务接口的具体实现与调用

/// </summary>

[ServiceContract]

public interface IWPServices

{

/// <summary>

/// 根据请求对象执行一个请求，并返回结果对象

/// </summary>

/// <param name="dataContext">请求对象</param>

/// <returns>结果对象</returns>

[OperationContract]

ResultObject Execute(BaseObject baseObject);

}

以下以工作计划的服务为例：

namespace SMT.WP.Services

{

public class WPServices : IWPServices

{

**private CommuncationManager \_CommuncationManager = null;**

public WPServices()

{

if (\_CommuncationManager == null)

**\_CommuncationManager = new CommuncationManager(new CommandWrapper());**

}

public ResultObject Execute(BaseObject baseObject)

{

ResultObject resultObject;

try

{

**resultObject = \_CommuncationManager.InvokeCommand(baseObject);**

}

catch (Exception ex)

{

//捕获后台抛出的任何异常，封装后给前台，此处为后台最后一步

resultObject = new ResultObject();

resultObject.State = false;

resultObject.ErrorMessage = ex.Message;

ServerLogger.WriteError("WPServices.Execute", ex);

}

return resultObject;

}

}

}

通讯服务需要调用通讯组件的CommunicationManager来进行通讯管理，并使用一个ICommandWrapper的实例，请参考通讯服务代理实现。

### 服务端通讯代理实现

服务端必须实现通讯代理接口ICommandWrapper，根据预定义的命令值映射到具体的业务接口：

namespace SMT.Portal.Communication

{

/// <summary>

/// 接口，命令调用容器，用于托管对服务端命令的调用逻辑。

/// </summary>

public interface ICommandWrapper

{

/// <summary>

/// 调用命令

/// </summary>

/// <param name="key">命令标识</param>

/// <param name="Content">数据对象</param>

/// <param name="commandParams">请求参数列表</param>

/// <returns>命令执行后返回的结果</returns>

object InvokeCommand(string key, object Content, params string[] commandParams);

}

}

以工作计划为例：

namespace SMT.WP.Services

{

/// <summary>

/// 命令包装接口<see cref="ICommandWrapper"/>实现，用于映射服务端接口的调用

/// </summary>

public class CommandWrapper : ICommandWrapper

{

/// <summary>

/// 根据给定的参数，请求匹配的命令

/// 用于将接收到客户端命令转换为具体的业务接口调用

/// </summary>

/// <param name="key">命令标识</param>

/// <param name="Content">数据对象</param>

/// <param name="commandParams">请求参数列表</param>

/// <returns>命令执行后返回的结果</returns>

public object InvokeCommand(string key, object Content, params string[] commandParams)

{

**CommandKeys commandkey = (CommandKeys)int.Parse(key);**

return SearchCommandByKey(commandkey, Content, commandParams);

return false;

}

private object SearchCommandByKey(CommandKeys commandkey, object Content, params string[] commandParams)

{

switch (commandkey)

{

case CommandKeys.Method1:

//具体的转换方法

return bll1.Method1(Content, commandParams);

case CommandKeys.Method2:

//具体的转换方法

return bll1.Method2(Content, commandParams);

……

可以预定一个CommandKeys的枚举，用于定义命令值：

namespace SMT.WP.Services

{

/// <summary>

/// 枚举，命令KEY列表，用于为客户端和服务端接口调用提供统一标识

/// </summary>

public enum CommandKeys : int

{

#region company norm define

COMPNORMDEFINE\_C = 101, //添加

COMPNORMDEFINE\_R = 102, //获取单个实体数据

COMPNORMDEFINE\_U = 103, //更新

COMPNORMDEFINE\_D = 104, //删除

COMPNORMDEFINE\_F = 105, //获取实体，不分页

COMPNORMDEFINE\_H = 106, //获取实体分页

#endregion

此枚举服务端、客户端需要同时引用。

## 客户端使用规范

### 客户端通讯代理实现

用于提供服务的通讯代理应该实现IServicesProvider接口：

namespace SMT.WP.Client.BLL

{

/// <summary>

/// 客户端服务提供者，用于映射对WPServices的调用。

/// </summary>

public class WPServicesProvider : IServicesProvider

{

private WPServicesWS.WPServicesClient \_client = null;

public WPServicesProvider()

{

\_client = new WPServicesWS.WPServicesClient();

}

/// <summary>

/// 根据请求对象执行一个请求，并返回结果对象

/// </summary>

/// <param name="dataContext">请求对象</param>

/// <returns>结果对象</returns>

public ResultObject Execute(BaseObject dataContext)

{

if (\_client == null)

\_client = new WPServicesWS.WPServicesClient();

return \_client.Execute(dataContext);

}

}

}

代理服务内部需要创建一个服务端通讯服务的Client端代理实例。

### 客户端业务接口调用

以具体的业务类来说明，内部需要调用通讯组件的CommunicationManager来进行通讯管理，并传入一个客户端通讯代理实例：

public class T\_WP\_PLANDRAFTCLIENT

{

private CommuncationManager \_commManager = null;

public T\_WP\_PLANDRAFTCLIENT()

{

if (\_commManager == null)

\_commManager = new CommuncationManager(new **WPServicesProvider**());

}

// <summary>

/// 根据ID获取公司计划申请记录

/// </summary>

/// <param name="ID"></param>

/// <returns></returns>

public T\_WP\_PLANDRAFT GetPlanDraftByID(string strID)

{

**return \_commManager.Execute<T\_WP\_PLANDRAFT>(strID, ((int)CommandKeys.PLANDRAFT\_E).ToString());**

}

CommuncationManager接口定义如下：

/// <summary>

/// 根据给定的参数执行请求，并返回处理结果

/// </summary>

/// <typeparam name="TResult">结果类型，可以为实体、列表、布尔值等任意对象</typeparam>

/// <param name="obj">请求的实体，比如，增、删、改、查等对象</param>

/// <param name="commandKey">请求的命令标识，KEY</param>

/// <param name="commandParams">请求参数列表</param>

/// <returns>请求结果</returns>

public TResult Execute<TResult>(object obj, string commandKey, params string[] commandParams)

# 数据访问开发指南

此部分主要针对NHibernate架构的OracleHelper数据库访问基类说明

## 保存数据

### SaveEntity说明

方法定义：bool SaveEntity<T>(T t)

返回结果：返回True或则False，保存成功返回True，失败返回False

参数：

T：业务实体参数，实体中可包含多级子父关系

### SaveEntity范例

1.单表保存

T\_WP\_DEPTNORMDEFINE deptNormDefine=new T\_WP\_DEPTNORMDEFINE(); //实例化实体

deptNormDefine.DEPTNORMID = Guid.NewGuid().ToString();

deptNormDefine.CREATEDATE = DateTime.Now;

deptNormDefine.CREATEUSERID =“Test”;

deptNormDefine.EDITSTATE = "2";

bool bResult = orahelper.SaveEntity (deptNormDefine); // 保存部门指标定义

2.多表保存

T\_WP\_YEARNORMDRAFT yearNormDraft=new T\_WP\_YEARNORMDRAFT (); //实例化父表实体

yearNormDefine.DNORMID = Guid.NewGuid().ToString();

yearNormDraft.EDITSTATE = "1";

yearNormDraft.CHECKSTATE = "1";

yearNormDraft.APPLYDATE = DateTime.Now;

yearNormDraft.APPLYMAN = “Test”;

T\_WP\_YEARNORMDEFINE yearNormDefine= new T\_WP\_YEARNORMDEFINE (); //实例化子表实体

yearNormDefine.NORMID = Guid.NewGuid().ToString();

yearNormDefine.CREATEUSERID = “Test”;

yearNormDefine.CREATEDATE = DateTime.Now;

yearNormDefine.EDITSTATE = "2";

yearNormDefine.DNORMID = yearNormDraft.DNORMID; //子父表关联

yearNormDefine.YEARNORMDRAFT = yearNormDraft;

yearNormDraft.YEARNORMDEFINES.Add(yearNormDefine);

}

bool bResult = orahelper.SaveEntity (deptNormDefine);

## 新增或更新数据

### SaveOrUpdate方法1

#### 说明

方法定义：bool SaveOrUpdateEntity<T>(T t)

返回结果：返回True或False，新增后保存成功返回True，失败返回False

参数：

T：业务实体参数，实体中可包含多级子父关系

#### 范例

T\_WP\_DEPTNORMDEFINE deptNormDefine=new T\_WP\_DEPTNORMDEFINE(); //实例化实体

deptNormDefine.DEPTNORMID = Guid.NewGuid().ToString();

deptNormDefine.CREATEDATE = DateTime.Now;

deptNormDefine.CREATEUSERID =“Test”;

deptNormDefine.EDITSTATE = "2";

bool bResult = orahelper. SaveOrUpdateEntity (deptNormDefine); // 新增或更新部门指标定义

### SaveOrUpdate方法2

#### 说明

方法定义：bool SaveOrUpdateEntity<T>(List<T> t)

返回结果：返回True或False，新增后保存成功返回True，失败返回False

参数：

List<T>：泛型业务实体

#### 范例

// normItemConfigList T\_WP\_NORMITEMCONFIG的List实例对象

public bool SaveOrUpdateNormItemConfig(List<T\_WP\_NORMITEMCONFIG> normItemConfigList)

{

return orahelper.SaveOrUpdateEntity(normItemConfigList);

}

## 更新数据

### UpdateEntity说明

方法定义：bool UpdateEntity<T>(T t)

返回结果：返回True或False，更新成功返回True，失败返回False

参数：

T：泛型业务实体

### UpdateEntity范例

T\_WP\_DEPTNORMDEFINE deptNormDefine=new T\_WP\_DEPTNORMDEFINE(); //实例化实体

deptNormDefine.DEPTNORMID = Guid.NewGuid().ToString();

deptNormDefine.CREATEDATE = DateTime.Now;

deptNormDefine.CREATEUSERID =“Test”;

deptNormDefine.EDITSTATE = "2";

bool bResult = orahelper. UpdateEntity (deptNormDefine); // 新增或更新部门指标定义

## 删除数据

### DeleteEntity说明

方法定义：bool DeleteEntity<T>(T t)

返回结果：返回True或False，更新成功返回True，失败返回False

参数：

T：泛型业务实体

### DeleteEntity范例

var e = YearNormDraftBLL.GetYearNormDraftByID(DNormID); //获取具体实体数据

bool bResult = oraclehelper.DeleteEntity (e); //删除数据

## 查询数据

此部分主要根据NHibernate提供的一种直观的、可扩展的Criteria API,在ICriteria实例对象上设置一个或多个表达式；要求ICriteria接口返回需要的列表，根据表达式从数据库中返回对象。

### GetEntity方法

#### 说明

方法定义：T GetEntity<T>(string strID)

返回结果：根据主键ID，返回主键ID实体数据

参数：

strID：主键ID

#### 范例

//strID为实体主键ID

T\_WP\_YEARNORMDRAFT yearNormDraft= orahelper.GetEntity<T\_WP\_YEARNORMDRAFT>(strID);

### ListEntities方法1

#### 说明

方法定义：IList<T> ListEntities<T>(List<ICriterion> expression)

返回结果：根据Nhibernate 查询表达式返回IList实体数据

参数：

Expression：Nhibernate 查询表达式

#### 范例

List<ICriterion> query = new List<ICriterion>(); //实例化条件表达式

if (!string.IsNullOrEmpty(strPlanYear))

query.Add(Expression.Eq(TABLE\_T\_WP\_YEARNORMDEFINE.PLANYEAR, strPlanYear)); // Expression的等号条件表达式

if (!string.IsNullOrEmpty(strCompanyName))

query.Add(Expression.Like(TABLE\_T\_WP\_YEARNORMDEFINE.COMPANYNAME, strCompanyName, MatchMode.Anywhere)); // Expression的Like条件表达式

IList <T\_WP\_YEARNORMDEFINE> ListYearNormDefine=

orahelper.ListEntities<T\_WP\_YEARNORMDEFINE>(query); //返回年度经营定义表达式

### ListEntities方法2

#### 说明

方法定义：IList<T> ListEntities<T>(List<ICriterion> expression, Order order, int iPageIndex, int iPageSize, ref int iPageCount, ref int iCount) where T : new()

返回结果：返回IList实体当前分页数据

参数：



#### 范例

int iPageCount;

int iCount;

List<ICriterion> query = new List<ICriterion>(); //实例化条件表达式

if (!string.IsNullOrEmpty(strPlanYear))

query.Add(Expression.Eq(TABLE\_T\_WP\_YEARNORMDEFINE.PLANYEAR, strPlanYear)); // Expression的等号条件表达式

if (!string.IsNullOrEmpty(strCompanyName))

query.Add(Expression.Like(TABLE\_T\_WP\_YEARNORMDEFINE.COMPANYNAME, strCompanyName, MatchMode.Anywhere)); // Expression的Like条件表达式

IList <T\_WP\_YEARNORMDEFINE> ListYearNormDefine=

orahelper.ListEntities<T\_WP\_YEARNORMDEFINE>( query,”asc”,1,15, iPageCount, iCount); //实体年度经营定义当前分页数据

**注：此部分已被方法3取代，如非特使情况，建议不要使用。**

### ListEntities方法3

#### 说明

方法定义：DataPage<T> ListEntities<T>(DataPage<T> dataPage, IList<ICriterion> Expression) where T : class,new()

返回结果：返回实体DataPage列表分页

参数：



#### 范例

DataPage<T\_WP\_YEARNORMDRAFT> dataPage = new DataPage<T\_WP\_YEARNORMDRAFT>(); //实例化dataPage对象

dataPage.AddOrder(sidx, false); //sidx前台UI传来值

dataPage.PageIndex = page; //page前台UI传来值

dataPage.PageSize = rows; // rows前台UI传来值

List<ICriterion> query = new List<ICriterion>(); //实例化条件表达式

if (!string.IsNullOrEmpty(strPlanYear))

query.Add(Expression.Eq(TABLE\_T\_WP\_YEARNORMDEFINE.PLANYEAR, strPlanYear)); // Expression的等号条件表达式

if (!string.IsNullOrEmpty(strCompanyName))

query.Add(Expression.Like(TABLE\_T\_WP\_YEARNORMDEFINE.COMPANYNAME, strCompanyName, MatchMode.Anywhere)); // Expression的Like条件表达式

DataPage<T\_WP\_YEARNORMDEFINE> ListYearNormDefine=

orahelper.ListEntities<T\_WP\_YEARNORMDEFINE>( dataPage ,query); //返回年度经营定义表达式DataPage列表分页

### ListEntities方法4

#### 说明

方法定义6：DataPage<T> ListEntities<T>(DataPage<T> dataPage) where T : class,new()

返回结果：分页显示(DataPage中包含表达式，转换成NH表达式)

参数：



#### 范例

DataPage<T\_WP\_YEARNORMDRAFT> dataPage = new DataPage<T\_WP\_YEARNORMDRAFT>(); //实例化dataPage对象

dataPage.AddOrder(sidx, false); //sidx前台UI传来值

dataPage.PageIndex = page; //page前台UI传来值

dataPage.PageSize = rows; // rows前台UI传来值

DataPage<T\_WP\_YEARNORMDEFINE> ListYearNormDefine=

orahelper.ListEntities<T\_WP\_YEARNORMDEFINE>( dataPage); //返回年度经营定义表达式DataPage列表分页

### 其它说明

#### ICriteria

1. 结果集限制

使用ICriteria接口提供的Add方法添加Restrictions类中约束表达式可以限制一些结果集的作用。

//using NHibernate.Criterion;

public IList<Customer> Narrowing()

{

IList<Customer> customers = \_session.CreateCriteria(typeof(Customer))

.Add(Restrictions.Like("Firstname", "YJing%"))

.Add(Restrictions.Between("Lastname", "A%", "Y%"))

.List<Customer>();

return customers;

}

1. 结果集排序

使用ICriteria.Order对结果集排序，第二个参数true代表asc，false代表desc。例如下面例子查询Customer对象按FirstName降序、Lastname升序。

public IList<Customer> Order()

{

return \_session.CreateCriteria(typeof(Customer))

.Add(Restrictions.Like("Firstname","Y%"))

.AddOrder(new NHibernate.Criterion.Order("Firstname", false))

.AddOrder(new NHibernate.Criterion.Order("Lastname", true))

.List<Customer>();

}

1. 直接使用SQL

IList cats = sess.CreateCriteria(typeof(Cat))

.Add(Expression.Sql("lower({alias}.Name) like lower(?)", "Fritz%", NHibernateUtil.String )

.List();

## 事务处理

说明：事务处理机制主要是利用NHibernate 提供的ITransaction API，对ITransaction执行一次Commit()会把所有状态同步，把底层的数据库连接提交。

范例参考：

using (ITransaction tx = \_session.BeginTransaction())  
    {  
        try  
        {  
            int newId = (int)\_session.Save(customer);  
            \_session.Flush();  
            tx.Commit();  
            return newId;  
        }  
        catch (HibernateException)  
        {  
            tx.Rollback();  
            throw;  
        }

**注：架构组已将事务封装在相应的方法中，非特殊情况，不建议单独使用事务。**

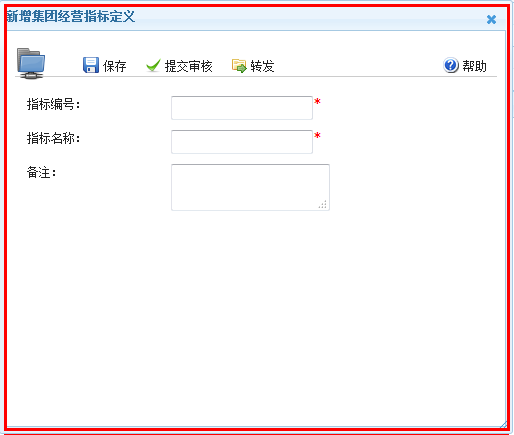
## 示例参看

### 单表操作示例

以下以工作计划(WP)中模块经营指标定义演示数据库操作，实例列表参看界面：



实例新增参看界面：



#### 查询数据

##### Client.UI的Control层

/// <summary>

/// jqGrid调用的方法入口，根据传入的参数返回数据，【sidx,sord,page,rows】这4个参数名称不能修改

/// </summary>

/// <param name = "sidx">指定排序字段</param>

/// <param name = "sord">指定排序方向</param>

/// <param name = "page">指定页（返回第几页的数据）</param>

/// <param name = "rows">指定每页几行数据</param>

/// <param name = " NormCode ">公司计划编号</param>

/// <param name = " NormName ">指标名称</param>

/// <returns>返回根据参数获取的符合jqGrid格式的数据</returns>



##### Client.BLL

/// <summary>

///查询经营指标定义信息

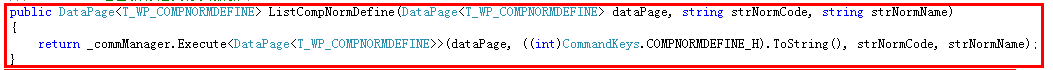
/// </summary>

/// <param name="dataPage">分页实体</param>

/// <param name="strNormCode">公司计划编号</param>

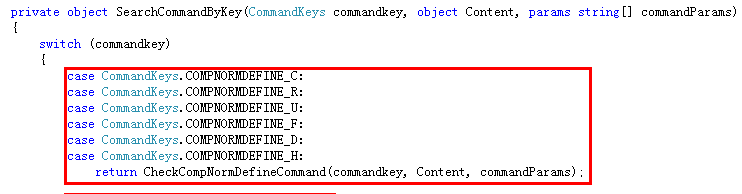
/// <param name="strNormName">指标名称</param>

/// <returns>经营指标定义分页数据</returns>



##### WP.Services

1. SearchCommandByKey写法



2. CheckCommand写法



##### WP.BLL

/// <summary>

/// 获取经营指标定义信息，分页

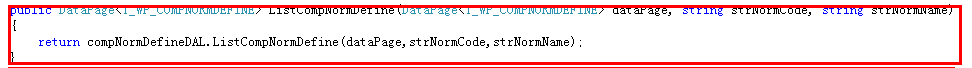
/// </summary>

/// <param name="dataPage">分页实体</param>

/// <param name="strNormCode">公司计划编号</param>

/// <param name="strNormName">指标名称</param>

/// <returns>经营指标定义分页数据</returns>



##### WP.IDAL

/// <summary>

/// 获取经营指标定义信息，分页

/// </summary>

/// <param name="dataPage">分页实体</param>

/// <param name="strNormCode">公司计划编号</param>

/// <param name="strNormName">指标名称</param>

/// <returns>经营指标定义分页数据</returns>

DataPage<T\_WP\_COMPNORMDEFINE> ListCompNormDefine(DataPage<T\_WP\_COMPNORMDEFINE> dataPage, string strNormCode, string strNormName);

##### WP.OracleDAL参看

/// <summary>

/// 获取经营指标定义信息，分页

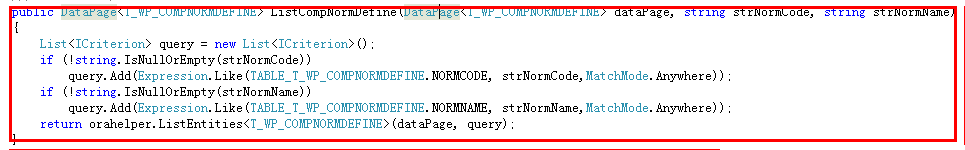
/// </summary>

/// <param name="dataPage">分页实体</param>

/// <param name="strNormCode">公司计划编号</param>

/// <param name="strNormName">指标名称</param>

/// <returns>经营指标定义分页数据</returns>



oraHelper为Nhibnate访问Oracle数据库的基类库，一般以以下方式定义：

private static readonly OracleHelper orahelper = new OracleHelper();

#### 新增数据

##### Client.UI的Control层

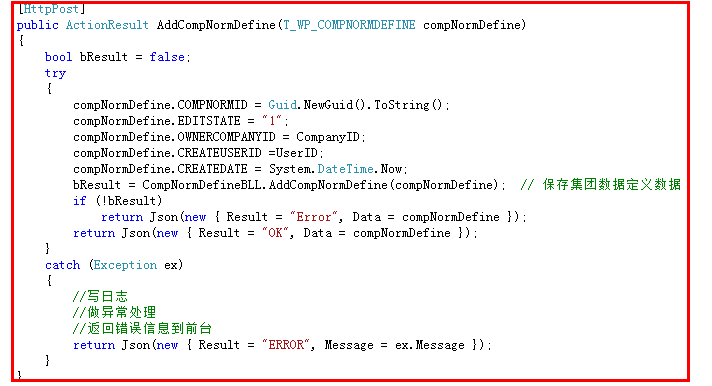
/// <summary>

/// 新增经营指标定义

/// </summary>

/// <param name="compNormDefine">经营指标定义实体，从views传递</param>

/// <returns> Json格式数据</returns>



##### Client.BLL

参看查询数据Client.BLL层处理方式

##### WP.Services

参看查询数据WP.Services层处理方式

##### WP.BLL

参看查询数据WP.BLL层处理方式

##### WP.IDAL

参看查询数据WP.IDAL层处理方式

##### WP.OracleDAL

/// <summary>

/// 新增公司经营指标定义

/// </summary>

/// <param name="compNormDefine">经营定义实体</param>

/// <returns>新增成功与失败，成功True、失败False </returns>

public bool AddCompNormDefine(T\_WP\_COMPNORMDEFINE compNormDefine)

{

return orahelper.SaveEntity<T\_WP\_COMPNORMDEFINE>(compNormDefine);

}

#### 更新数据

##### Client.UI的Control层

/// <summary>

/// 更新经营指标定义

/// </summary>

/// <param name="compNormDefine">经营指标定义实体，从views传递</param>

/// <returns> Json格式数据</returns>



**注：实体变量中，如果views传递过来的实体字段属性全部都有数据，可不用再从数据库中获取实体数据e,直接将views传递过来的实体更新到数据库中即可。**

##### Client.BLL

参看查询数据Client.BLL层处理方式

##### WP.Services

参看查询数据WP.Services层处理方式

##### WP.BLL

参看查询数据WP.BLL层处理方式

##### WP.IDAL

参看查询数据WP.IDAL层处理方式

##### WP.OracleDAL

/// <summary>

/// 更新公司经营指标定义

/// </summary>

/// <param name="compNormDefine">经营定义实体</param>

/// <returns>更新成功与失败，成功True、失败False </returns>

public bool UpdateCompNormDefine(T\_WP\_COMPNORMDEFINE compNormDefine)

{

return orahelper.UpdateEntity<T\_WP\_COMPNORMDEFINE>(compNormDefine);

}

#### 删除数据

##### Client.UI的Control层

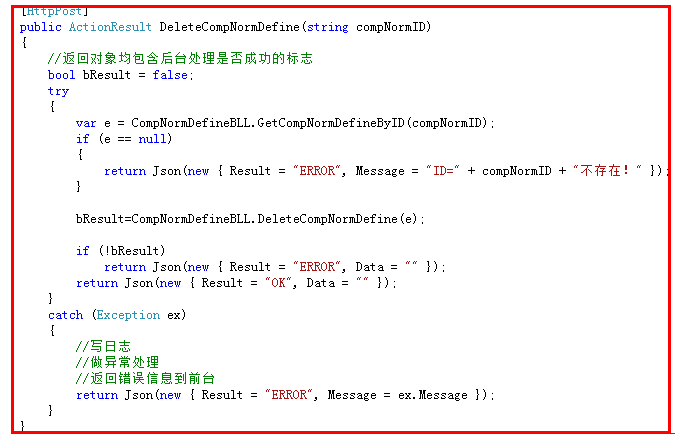
/// <summary>

/// 删除经营指标定义

/// </summary>

/// <param name="CompNormID">指标定义ID</param>

/// <returns>Json格式数据</returns>



##### Client.BLL

参看查询数据Client.BLL层处理方式

##### WP.Services

参看查询数据WP.Services层处理方式

##### WP.BLL

参看查询数据WP.BLL层处理方式

##### WP.IDAL

参看查询数据WP.IDAL层处理方式

##### WP.OracleDAL

/// <summary>

/// 删除公司经营指标定义

/// </summary>

/// <param name="compNormDefine">经营定义实体</param>

/// <returns>删除成功与失败，成功True、失败False</returns>

public bool DeleteCompNormDefine(T\_WP\_COMPNORMDEFINE compNormDefine)

{

return orahelper.DeleteEntity<T\_WP\_COMPNORMDEFINE>(compNormDefine);

}

### 子父表操作示例

以下以工作计划(WP)中模块部门预留费用设置及公司年度申请演示数据库操作，实例列表参看界面：



新增参看界面：



#### 查询数据

##### Client.UI的Control层

以子表为主，查询数据类似单表查询数据处理方式

/// <summary>

/// jqGrid调用的方法入口，根据传入的参数返回数据，【sidx,sord,page,rows】这4个参数名称不能修改

/// </summary>

/// <param name = "sidx">指定排序字段</param>

/// <param name = "sord">指定排序方向</param>

/// <param name = "page">指定页（返回第几页的数据）</param>

/// <param name = "rows">指定每页几行数据</param>

/// <param name = "PlanYear">计划年度</param>

/// <param name = "CompanyID">公司</param>

/// <param name = "CompanyName">公司名称</param>

/// <param name = "DepartmentID">部门</param>

/// <returns>返回根据参数获取的符合jqGrid格式的数据</returns>



##### Client.BLL

参看单表操作查询数据Client.BLL层处理方式

##### WP.Services

参看单表操作查询数据WP.Servies层处理方式

##### Client.BLL

参看单表操作查询数据WP.BLL层处理方式

##### WP.IDAL

/// <summary>

/// 获取年度预留机动费用设置方案，分页

/// </summary>

/// <param name="dataPage">预留配置实体分页</param>

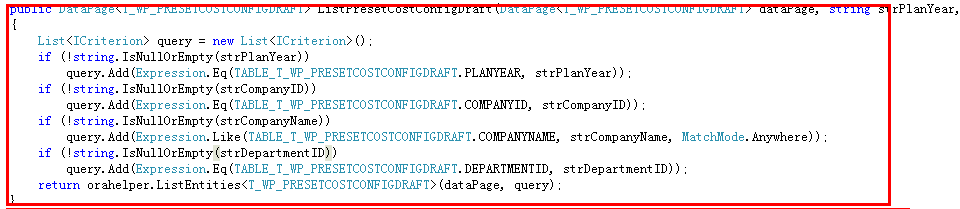
/// <param name = "PlanYear">计划年度</param>

/// <param name = "CompanyID">公司</param>

/// <param name = "CompanyName">公司名称</param>

/// <param name = "DepartmentID">部门</param>

/// <returns>当前数量分页实体数据</returns>



##### WP.OracleDAL

列表显示一般以主表为主，处理方式参考单表操作查询数据WP.OracleDAL层。

#### 新增数据

##### Client.UI的Control层

/// <summary>

/// 新增部门预留费用设置

/// </summary>

/// <param name="presetCostConfigDraft">预留配置方案,从views传递</param>

/// <param name="presetCostConfigList">预留配置信息,从views传递</param>

/// <returns>Json数据格式</returns>

[HttpPost]

public ActionResult AddConfig(T\_WP\_PRESETCOSTCONFIGDRAFT presetCostConfigDraft, IList<T\_WP\_PRESETCOSTCONFIG> presetCostConfigList)

{

bool bResult;

string strFormID = presetCostConfigDraft.DCOSTID;

//返回对象均包含后台处理是否成功的标志

try

{

presetCostConfigDraft.DEPARTMENTID = DeptID;

presetCostConfigDraft.DEPARTMENTNAME = DeptName;

presetCostConfigDraft.OWNERCOMPANYID = CompanyID;

presetCostConfigDraft.OWNERDEPARTMENTID = DeptID;

presetCostConfigDraft.OWNERID = UserID;

presetCostConfigDraft.OWNERPOSTID = PostID;

presetCostConfigDraft.EDITSTATE = "2";

presetCostConfigDraft.CREATECOMPANYID = CompanyID;

presetCostConfigDraft.CREATEDEPARTMENTID = DeptName;

foreach (var presetCostConfig in presetCostConfigList) //添加年度预留机动费用设置 {

presetCostConfig.COSTID = Guid.NewGuid().ToString();

presetCostConfig.CREATEUSERID = UserID;

presetCostConfig.CREATEDATE = DateTime.Now;

presetCostConfig.EDITSTATE = "2";

presetCostConfig.COSTTYPE = "2";

presetCostConfig.COMPANYID = CompanyID;

presetCostConfig.COMPANYNAME = CompanyName;

**presetCostConfig.DCOSTID = presetCostConfigDraft.DCOSTID;**

**presetCostConfig.PRESETCOSTCONFIGDRAFT = presetCostConfigDraft;**

**presetCostConfigDraft.PRESETCOSTCONFIGS.Add(presetCostConfig);** //子父关系设置

}

bResult = PresetCostConfigDraftBLL.AddPresetCostConfigDraft(presetCostConfigDraft); //保存预留设置申请

if(bResult)

{

return Json(new { Result = "OK", Data = vData });

}

return Json(new { Result = "Error", Data = presetCostConfigDraft });

}

catch (Exception ex)

{

//写日志

//做异常处理

//返回错误信息到前台

return Json(new { Result = "ERROR", Message = ex.Message });

}

}

##### Client.BLL

参看单表操作查询数据Client.BLL层处理方式

##### WP.Services

参看单表操作查询数据WP.Servies层处理方式

##### WP.BLL

参看单表操作查询数据WP.BLL层处理方式

##### WP.IDAL

参看单表操作查询数据WP.IDAL层处理方式

##### WP.OracleDAL

/// <summary>

/// 添加年度预留机动费用设置方案

/// </summary>

/// <param name="presetCostConfigDraft">配置方案实体</param>

/// <returns>新增成功与失败，成功True、失败False</returns>

public bool AddPresetCostConfigDraft(T\_WP\_PRESETCOSTCONFIGDRAFT presetCostConfigDraft)

{

return orahelper.SaveEntity<T\_WP\_PRESETCOSTCONFIGDRAFT>(presetCostConfigDraft);

}

#### 更新数据

##### Client.UI的Control层

1.子父表同时更新

/// <summary>

/// 更新预留配置方案信息

/// </summary>

/// <param name="presetCostConfigDraft">预留配置方案,从views传递</param>

/// <param name="presetCostConfigList">预留配置信息,从views传递</param>

/// <returns>Json数据格式</returns>

[HttpPost]

public ActionResult UpdateConfig(T\_WP\_PRESETCOSTCONFIGDRAFT presetCostConfigDraft, IList<T\_WP\_PRESETCOSTCONFIG> presetCostConfigList)

{

bool bResult = false;

try

{

var presetCostConfigDraftData= PresetCostConfigDraftBLL.GetPresetCostConfigDraftID(presetCostConfigDraft.DCOSTID);

if (presetCostConfigDraftData == null)

{

return Json(new { Result = "ERROR", Message = "ID=" + presetCostConfigDraft.DCOSTID + "不存在！" });

}

presetCostConfigDraft.OWNERCOMPANYID = presetCostConfigDraftData.OWNERCOMPANYID;

presetCostConfigDraft.OWNERDEPARTMENTID = presetCostConfigDraftData.OWNERDEPARTMENTID;

presetCostConfigDraft.OWNERID = presetCostConfigDraftData.OWNERID;

presetCostConfigDraft.OWNERPOSTID = presetCostConfigDraftData.OWNERPOSTID;

presetCostConfigDraft.EDITSTATE = presetCostConfigDraftData.EDITSTATE;

presetCostConfigDraft.CREATECOMPANYID = presetCostConfigDraftData.CREATECOMPANYID;

presetCostConfigDraft.CREATEDEPARTMENTID = presetCostConfigDraftData.CREATEDEPARTMENTID;

IList<T\_WP\_PRESETCOSTCONFIG> presetCostConfigListData = PresetCostConfigBLL.GetPresetCostConfig(presetCostConfigDraft.DCOSTID);

foreach (var presetCostConfig in presetCostConfigList) //更新公司年度经营指标定义明细表 {

var item = presetCostConfigListData.Where(c => c.COSTID == presetCostConfig.COSTID).FirstOrDefault();

if (item != null) //更新年度指标定义

{

presetCostConfig.CREATEUSERID = item.CREATEUSERID;

presetCostConfig.CREATEDATE = item.CREATEDATE;

presetCostConfig.EDITSTATE = item.EDITSTATE;

presetCostConfig.COSTTYPE = item.COSTTYPE;

presetCostConfig.COMPANYID = item.COMPANYID;

presetCostConfig.COMPANYNAME = item.COMPANYNAME;

presetCostConfig.UPDATEDATE = DateTime.Now;

presetCostConfig.UPDATEUSERID = UserID;

}

**presetCostConfig.DCOSTID = presetCostConfigDraft.DCOSTID;**

**presetCostConfig.PRESETCOSTCONFIGDRAFT = presetCostConfigDraft;**

**presetCostConfigDraft.PRESETCOSTCONFIGS.Add(presetCostConfig);** //子父关系设置

}

bResult = PresetCostConfigDraftBLL.UpdatePresetCostConfigDraft(presetCostConfigDraft); // 保存预留配置方案信息

if (bResult)

return Json(new { Result = "OK", Data = "" });

return Json(new { Result = "Error", Data = presetCostConfigDraft });

}

catch (Exception ex)

{

//写日志

//做异常处理

//返回错误信息到前台

return Json(new { Result = "ERROR", Message = ex.Message });

}

}

2.父表更新，子表新增后者修改状态

List<T\_WP\_PLANDRAFT> ListPlan = new List<T\_WP\_PLANDRAFT>();

T\_WP\_PLANDRAFT entity = new T\_WP\_PLANDRAFT();

entity.CHECKSTATE = "0";

entity.COMPANYID = this.CompanyID;

entity.FICTIONMAN = this.UserName;

entity.EDITSTATE = "1";

entity.OWNERCOMPANYID = this.CompanyID;

entity.OWNERDEPARTMENTID = this.DeptID;

if (planDraft.TacticNormDraft != null && planDraft.TacticNormDraft.Count > 0)

{

foreach (var v in planDraft.TacticNormDraft)

{

**if (string.IsNullOrEmpty(v.TACNORMID))**

**{**

**v.TACNORMID = Guid.NewGuid().ToString();**

**}**

v.FICTIONID = entity.FICTIONID;

v.COMPANYID = this.CompanyID;

v.CREATEUSERID = this.UserID;

v.CREATEDATE = DateTime.Now;

v.T\_WP\_PLANDRAFT = entity;

entity.TacticNormDraft.Add(v);

}

}

bool bResult = ClientBLL.**SaveOrUpdate**PlanDraft(ListPlan);

##### Client.BLL

参看单表操作查询数据Client.BLL层处理方式

##### WP.Services

参看单表操作查询数据WP.Servies层处理方式

##### WP.BLL

参看单表操作查询数据WP.BLL层处理方式

##### WP.IDAL

参看单表操作查询数据WP.IDAL层处理方式

##### WP.OracleDAL

1.子父表同时更新

/// <summary>

/// 更新年度预留机动费用设置方案

/// </summary>

/// <param name="presetCostConfigDraft">配置方案实体</param>

/// <returns>更新成功与失败，成功True、失败False</returns>

public bool UpdatePresetCostConfigDraft(T\_WP\_PRESETCOSTCONFIGDRAFT presetCostConfigDraft)

{

return orahelper.UpdateEntity<T\_WP\_PRESETCOSTCONFIGDRAFT>(presetCostConfigDraft);

}

2.父表更新，子表新增后者修改状态

/// <summary>

/// 批量新增与更新公司计划申请

/// </summary>

/// <param name="planDraft">批量计划申请实体数据</param>

/// <returns>批量更新更新成功与失败，成功True、失败False </returns>

public bool SaveOrUpdatePlanDraft(List<T\_WP\_PLANDRAFT> planDraft)

{

return oraclehelper.SaveOrUpdateEntity(planDraft);

}

#### 删除数据

##### Client.UI的Control层

删除子父表关系时，以主表为主，NHibernate自动关联的子表删除

/// <summary>

/// 删除预留机制

/// </summary>

/// <param name="DCostID">预留配置主键</param>

/// <returns>Json数据格式</returns>

[HttpPost]

public ActionResult DeleteConfig(string DCostID)

{

//返回对象均包含后台处理是否成功的标志

bool bResult = false;

try

{

var e = PresetCostConfigDraftBLL.GetPresetCostConfigDraftID(DCostID);

if (e == null)

{

return Json(new { Result = "ERROR", Message = "ID=" + DCostID + "不存在！" });

}

bResult = PresetCostConfigDraftBLL.DeletePresetCostConfigDraft(e);

if (!bResult)

return Json(new { Result = "ERROR", Data = "" });

return Json(new { Result = "OK", Data = "" });

}

catch (Exception ex)

{

//写日志

//做异常处理

//返回错误信息到前台

return Json(new { Result = "ERROR", Message = ex.Message });

}

}

##### Client.BLL

参看单表操作查询数据Client.BLL层处理方式

##### WP.Services

参看单表操作查询数据WP.Servies层处理方式

##### WP.BLL

参看单表操作查询数据WP.BLL层处理方式

##### WP.IDAL

参看单表操作查询数据WP.IDAL层处理方式

##### WP.OracleDAL

/// <summary>

/// 删除年度预留机动费用设置方案

/// </summary>

/// <param name="presetCostConfigDraft">配置方案实体</param>

/// <returns>删除成功与失败，成功True、失败False</returns>

public bool DeletePresetCostConfigDraft(T\_WP\_PRESETCOSTCONFIGDRAFT presetCostConfigDraft)

{

return orahelper.DeleteEntity<T\_WP\_PRESETCOSTCONFIGDRAFT>(presetCostConfigDraft);

}

# 公共服务使用指南

## 数据访问权限

### 数据访问权限定义

当用户登录后，访问数据时，需要对数据进行权限筛选，如下：

* 个人权限★

仅能访问自己的数据

* 岗位权限★★

可以访问当前岗位的所有数据

* 部门权限★★★

可以访问当前部门的所有数据

* 公司权限★★★★

可以访问当前公司的所有数据

* 自定义权限

可以访问自定义范围内的所有数据

### 权限过滤信息对象

/// <summary>

/// 用于权限数据过滤的对象

/// </summary>

public class FliterObject

{

/// <summary>

/// 用户ID

/// </summary>

public string UseID { get; set; }

/// <summary>

/// 模块ID

/// </summary>

public string ModelCode { get; set; }

}

### 代码示例

#### 客户端

不需要数据访问权限的客户端方法一般如下：

/// <summary>

/// 查询公司经营指标定义信息

/// </summary>

/// <param name="dataPage"></param>

/// <param name="strNormCode"></param>

/// <param name="strNormName"></param>

/// <returns></returns>

public DataPage<T\_WP\_COMPNORMDEFINE> ListCompNormDefine(DataPage<T\_WP\_COMPNORMDEFINE> dataPage, string strNormCode, string strNormName)

{

return \_commManager.Execute<DataPage<T\_WP\_COMPNORMDEFINE>>(dataPage, ((int)CommandKeys.COMPNORMDEFINE\_H).ToString(), strNormCode, strNormName);

}

其中dataPage为用户需要传递给后台的自定义对象。

**当我们需要做数据访问过滤时，需要构造一个权限过滤信息对象，并与用户自定义对象组合起来一起传递到后台**，如下示例：

public DataPage<T\_WP\_COMPNORMDEFINE> ListCompNormDefine(DataPage<T\_WP\_COMPNORMDEFINE> dataPage, **FliterObject fo**, string strNormCode, string strNormName)

{

var data = new object[] {dataPage,fo};

return \_commManager.Execute<DataPage<T\_WP\_COMPNORMDEFINE>>(**data**, ((int)CommandKeys.COMPNORMDEFINE\_H).ToString(), strNormCode, strNormName);

}

#### 服务端

不需要数据访问权限的服务端方法一般如下：

……

DataPage<T\_WP\_COMPNORMDEFINE> dataPage = (DataPage<T\_WP\_COMPNORMDEFINE>) Content;

return \_compNormDefineBLL.ListCompNormDefine(dataPage, (string) commandParams[0],

(string) commandParams[1]);

其中dataPage为前台传递过来的自定义对象。

**当我们需要做数据访问过滤时，需要接收从前台传递过来的权限过滤信息对象**，如下示例：

var dataPage = (DataPage<T\_WP\_COMPNORMDEFINE>) Content[0];

FilterObject fo = (FilterObject)Content[1];

return \_compNormDefineBLL.ListCompNormDefine(dataPage, **fo**, (string) commandParams[0],

(string) commandParams[1]);

在ListCompNormDefin方法内部，直接交由数据访问功能类处理即可，如下：

public DataPage<T\_WP\_COMPNORMDEFINE> ListCompNormDefine(DataPage<T\_WP\_COMPNORMDEFINE> dataPage, **FilterObject fo**,

string strNormCode, string strNormName)

{

List<ICriterion> query = new List<ICriterion>();

if (!string.IsNullOrEmpty(strNormCode))

query.Add(Expression.Like(TABLE\_T\_WP\_COMPNORMDEFINE.NORMCODE, strNormCode,MatchMode.Anywhere));

if (!string.IsNullOrEmpty(strNormName))

query.Add(Expression.Like(TABLE\_T\_WP\_COMPNORMDEFINE.NORMNAME, strNormName,MatchMode.Anywhere));

return orahelper.ListEntities<T\_WP\_COMPNORMDEFINE>(dataPage, **fo**, query);

}

## 用户及组织机构

创建用户及组织机构服务:



调用用户及组织机构相应的方法：



### 方法GetCompanyByEntityPerm

根据给定的权限编号、模块名称、用户ID获取对应的公司列表

#### 原型

List<Company> GetCompanyByEntityPerm(string userId, string prmCode, string entityName)

#### 参数

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名 | 类型 | 描述 | 操作 |
| userId | string | 用户ID | 必填 |
| prmCode | string | 权限编号 | 必填 |
| entityName | string | 实体名称 | 必填 |

#### 返回值

|  |  |
| --- | --- |
| 类型 | 描述 |
| Company | 公司列表 |

### 方法GetDepartmentByEntityPerm

根据给定的权限编号、模块名称、用户ID获取对应的部门列表

#### 原型

List<Department> GetDepartmentByEntityPerm(string userId, string prmCode, string entityName)

#### 参数

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名 | 类型 | 描述 | 操作 |
| userId | string | 用户ID | 必填 |
| prmCode | string | 权限编号 | 必填 |
| entityName | string | 实体名称 | 必填 |

#### 返回值

|  |  |
| --- | --- |
| 类型 | 描述 |
| Department | 部门列表 |

### 方法GetPostByEntityPerm

根据给定的权限编号、模块名称、用户ID获取对应的岗位列表

#### 原型

List<Post> GetPostByEntityPerm(string userId, string prmCode, string entityName)

#### 参数

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名 | 类型 | 描述 | 操作 |
| userId | string | 用户ID | 必填 |
| prmCode | string | 权限编号 | 必填 |
| entityName | string | 实体名称 | 必填 |

#### 返回值

|  |  |
| --- | --- |
| 类型 | 描述 |
| Post | 岗位列表 |

### 方法GetALLCompanyView

根据给定的用户ID获取所有公司列表

#### 原型

List<Company> GetALLCompanyView(string userID)

#### 参数

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名 | 类型 | 描述 | 操作 |
| userID | string | 用户ID | 必填 |

#### 返回值

|  |  |
| --- | --- |
| 类型 | 描述 |
| Company | 公司列表 |

### 方法GetAllDepartmentView

根据给定的用户ID获取所有部门列表

#### 原型

List<Department> GetAllDepartmentView(string userID)

#### 参数

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名 | 类型 | 描述 | 操作 |
| userID | string | 用户ID | 必填 |

#### 返回值

|  |  |
| --- | --- |
| 类型 | 描述 |
| Department | 部门列表 |

### 方法GetEmployeePostByPostID

根据岗位ID，获取当前岗位下的所有雇员信息

#### 原型

List<Employee> GetEmployeePostByPostID(string postID)

#### 参数

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名 | 类型 | 描述 | 操作 |
| postID | string | 岗位编号 | 必填 |

#### 返回值

|  |  |
| --- | --- |
| 类型 | 描述 |
| Employee | 雇员列表 |

## 工作流程

### 方法GetFlowInfo

检索流程数据

#### 原型

List< FLOW\_FLOWRECORDDETAIL\_T> GetFlowInfo(string FormID, string FlowGUID, string Flag, string ModelCode, string CompanyID, string EditUserID)

#### 参数

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名 | 类型 | 描述 | 操作 |
| FormID | string | 业务表单ID | 可选 |
| FlowGUID | string | 待审批流程GUID | 可选 |
| Flag | string | 审批标志(0-未审批,1-已审批) | 可选 |
| ModelCode | string | 模块代码 | 可选 |
| CompanyID | string | 公司代码 | 可选 |
| OfficeID | string | 岗位ID | 可选 |
| EditUserID | string | 审批人ID | 可选 |

#### 返回值

|  |  |
| --- | --- |
| 类型 | 描述 |
| FLOW\_FLOWRECORDDETAIL\_T | 返回FLOW\_FLOWRECORDDETAIL\_T对象数组 |

### 方法 SubmitFlow

提交表单到流程系统

#### 原型

DataResult SubimtFlow(SubmitData ApprovalData);

#### 参数

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名 | 类型 | 描述 | 操作 |
| FormID | string | 业务表单ID | 必填 |
| FlowGUID | string | 待审批流程GUID | Status为Add时不填  Status为Update时必填 |
| ModelCode | string | 模块代码 | 必填 |
| CompanyID | string | 公司代码 | 必填 |
| PostID | string | 岗位ID | 必填 |
| CreateUserID | string | 创建用户ID | 必填 |
| CreateUserName | string | 创建用户名称 | 必填 |
| NextStateCode | string | 自定义流程代码 | 可选 |
| AppUserId | string | 下一步骤人ID | 必填 |
| AppUserName | string | 下一步骤人名称 | 必填 |
| Content | string | 审批意见内容 | Status为Add时不填  Status为Update时必填 |
| AppOpt | string | 审批意见(0-不同意,1-同意) | Status为Add时不填  Status为Update时必填 |
| Status | string | 操作标志:Add-增加，Update-审批] | 必填 |

#### 返回值

|  |  |
| --- | --- |
| 类型 | 描述 |
| DataResult | 操作结束后返回DataResult类型对象,操作失败时flowResult=FAIL, userInfo.count=1;操作成功时flowResult= SUCCESS, userInfo.count=1；没有审批用户时flowResult=FAIL, userInfo=null；有多个审批用户时flowResult=FAIL, userInfo.count>1；审批结束时flowResult= END, userInfo.count=1 |

### 方法 GetAppUser

根据公司ID，模块ID，状态ID查询角色用户

#### 原型

List<UserInfo> GetAppUser(string CompanyID, string ModelCode, string StateName)

#### 参数

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名 | 类型 | 描述 | 操作 |
| CompanyID | string | 公司代码 | 必填 |
| ModelCode | string | 模块代码 | 必填 |
| FlowGUID | string | 审批流程GUID | 必填[新增时用StartFlow或者为空] |

#### 返回值

|  |  |
| --- | --- |
| 类型 | 描述 |
| List<UserInfo> | 返回UserInfo类型数组 |

### 方法AddConsultation

新增咨询

#### 原型

void AddConsultation(FLOW\_CONSULTATION\_T flowConsultation, SubmitData submitData)

#### 参数

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名 | 类型 | 描述 | 操作 |
| flowConsultation | FLOW\_CONSULTATION\_T | 咨询信息对象 | 必填 |
| submitData | SubmitData | 审核信息对象 | 必填 |

#### 返回值

无

### 方法ReplyConsultation

回复咨询

#### 原型

void ReplyConsultation(FLOW\_CONSULTATION\_T flowConsultation, SubmitData submitData)

#### 参数

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名 | 类型 | 描述 | 操作 |
| flowConsultation | FLOW\_CONSULTATION\_T | 咨询信息对象 | 必填 |
| submitData | SubmitData | 审核信息对象 | 必填 |

#### 返回值

无

### 对象SubmitData

流程审批提交数据对象

#### 对象说明

|  |  |  |  |
| --- | --- | --- | --- |
| **编号** | **属性** | **说明** | **数据类型** |
| 1 | FlowSelectType | 选择提交的类型（0：固定流程，1：自选流程） | string |
| 2 | FlomID | 业务表单ID | string |
| 3 | ModelCode | 模块代码 | string |
| 4 | CompanyID | 提交人公司ID | string |
| 5 | PostID | 提交人岗位ID | string |
| 6 | Approval  UserID | 提交人用户ID | string |
| 7 | Approval  UserName | 提交人用户名称 | string |
| 8 | NextStateCode | 下一审批节点代码 | string |
| 9 | NextApproval  UserId | 下一审批人用户ID | string |
| 10 | NextApproval  UserName | 下一审批人用户名称 | string |
| 11 | Approval  Content | 审批意见 | string |
| 12 | ApprovalResult | 审批结果(0:不同意，1：同意) | ApprovalResult |
| 13 | FlowType | 流程类型（0：审批流程，1：任务流程） | FlowType |
| 14 | SubmitFlag | 提交标志(0：提交新流程，1：审批流程) | SubmitFlag |
| 15 | XML | 业务数据XML | string |

### 类型ApprovalResult

审批意见结果

#### 类型说明

|  |  |  |  |
| --- | --- | --- | --- |
| **编号** | **名称** | **代码** | **值** |
| 0 | 不同意 | NoPass | 0 |
| 1 | 同意 | Pass | 1 |

### 类型FlowType

提交的流程类型

#### 类型说明

|  |  |  |  |
| --- | --- | --- | --- |
| **编号** | **名称** | **代码** | **值** |
| 0 | 审批流程 | Approval | 0 |
| 1 | 任务流程 | Task | 1 |
| 2 | 自动发起流程 | Pending | 2 |

### 类型SubmitFlag

提交处理标志

#### 类型说明

|  |  |  |  |
| --- | --- | --- | --- |
| **编号** | **名称** | **代码** | **值** |
| 0 | 新建新流程 | New | 0 |
| 1 | 审批流程 | Approval | 1 |

### 类型FlowResult

审批流程操作结果,枚举类型

#### 类型说明

|  |  |  |  |
| --- | --- | --- | --- |
| **编号** | **名称** | **代码** | **值** |
| 0 | 操作失败 | FAIL | 0 |
| 1 | 操作成功 | SUCCESS | 1 |
| 2 | 流程结束 | END | 2 |
| 3 | 多用户 | MULTIUSER | 3 |

### 对象UserInfo

用户信息

#### 对象说明

|  |  |  |  |
| --- | --- | --- | --- |
| 编号 | **名称** | **代码** | **数据类型** |
| 1 | 用户ID | UserID | String |
| 2 | 用户名称 | UserName | String |

### 对象DataResult

流程审批结束返回对象

#### 对象说明

|  |  |  |  |
| --- | --- | --- | --- |
| **编号** | **属性** | **说明** | **数据类型** |
| 1 | flowResult | 操作结果 | FlowResult |
| 2 | userInfo | 下一用户表 | UserInfo[] |
| 3 | Err | 错误信息 | string |
| 4 | AppState | 下一审核状态代码 | string |
| 5 | CheckState | 审批结果 | string |
| 6 | ModelFlowRelationID | 模块关联流程ID | string |
| 7 | SubModelCode | 子模块代码 | string |
| 8 | RunTime | 运行时间记录 | string |

### 对象FLOW\_CONSULTATION\_T

咨询实体对象

#### 对象说明

|  |  |  |  |
| --- | --- | --- | --- |
| **编号** | **属性** | **说明** | **数据类型** |
| 1 | CONSULTATIONCONTENT | 咨询内容 | string |
| 2 | CONSULTATIONDATE | 发送咨询的时间 | DateTime |
| 3 | CONSULTATIONID | Guid主键 | string |
| 4 | CONSULTATIONUSERID | 发送咨询用户ID | string |
| 5 | CONSULTATIONUSERNAME | 发送咨询用户名称 | string |
| 6 | FLAG | 标志位0未回复，1已回复 | string |
| 7 | FLOW\_FLOWRECORDDETAIL\_T | 对应的审核明细 | Flow\_FlowRecorddetail\_T |
| 8 | REPLYCONTENT | 回复内容 | string |
| 9 | REPLYDATE | 回复时间 | DateTime |
| 10 | REPLYUSERID | 回复用户ID | string |
| 11 | REPLYUSERNAME | 回复用户名称 | string |

## 消息引擎服务

### 方法SaveFlowTriggerData

流程驱动引擎发送消息

#### 原型

string SaveFlowTriggerData(string strFlowMessage, string strBOObject);

#### 参数

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名 | 类型 | 描述 | 操作 |
| strFlowMessage | string | 流程XML | 必填 |
| strBOObject | string | 业务系统XML | 必填 |

#### 返回值

|  |  |
| --- | --- |
| 类型 | 描述 |
| string | SaveXmlData success! |

### 方法 PendingTasks

查找待办任务

#### 原型

List<T\_FLOW\_ENGINEMSGLIST> PendingTasks(string strUserID, string strStatus, int iTop);

#### 参数

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名 | 类型 | 描述 | 操作 |
| strUserID | string | 用户ID | 必填 |
| strStatus | string | 状态 | open为待办任务  close为已处理任务 |
| iTop | string | Top | 选取多少条数据 |

#### 返回值

|  |  |
| --- | --- |
| 类型 | 描述 |
| T\_FLOW\_ENGINEMSGLIST | 待办任务实体 |

### 方法 PendingTasksParms

查找待办任务

#### 原型

List<T\_FLOW\_ENGINEMSGLIST> PendingTasksParms(MsgParms msgParams);

#### 参数

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名 | 类型 | 描述 | 操作 |
| MsgParms |  | public string UserID  public string Status  public int Top  public int LastDay | LastDay: 系统当前时间往前推 最近多少天 |
|  |  |  |  |
|  |  |  |  |

#### 返回值

|  |  |
| --- | --- |
| 类型 | 描述 |
| List<T\_FLOW\_ENGINEMSGLIST> | 返回T\_FLOW\_ENGINEMSGLIST类型数组 |

### 方法PendingMainTasksParms

查找待办任务，但T\_FLOW\_ENGINEMSGLIST实体中只有门户需要的字段数据：时间、内容、系统代号

#### 原型

List<T\_FLOW\_ENGINEMSGLIST> PendingMainTasksParms(MsgParms msgParams);

### 方法PendingDetailTasks

查找待办任务，但T\_FLOW\_ENGINEMSGLIST实体中有详细的待办任务信息

#### 原型

T\_FLOW\_ENGINEMSGLIST PendingDetailTasks(string strMsgID);

#### 参数

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名 | 类型 | 描述 | 操作 |
| strMsgID | string | T\_FLOW\_ENGINEMSGLIST实体中的MessageID |  |
|  |  |  |  |
|  |  |  |  |

### 方法EngineNotes

获取引擎消息（门户无弹出）

#### 原型

List<T\_FLOW\_ENGINENOTES> EngineNotes(string strUserID, string strStatus, int iTop);

#### 参数

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名 | 类型 | 描述 | 操作 |
| strUserID | String | 用户ID | 必填 |
| strStatus | String | 状态 | open为待办任务  close为已处理任务 |
| iTop | String | Top | 选取多少条数据 |

### 方法SaveEventData

保存定时触发

#### 原型

void SaveEventData(string strEventXml);

#### 参数

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名 | 类型 | 描述 | 操作 |
| strEventXml | string | <?xml version="1.0" encoding="utf-8" ?>  <System>  <CompanyCode>公司代号</CompanyCode>  <SystemCode>系统代号</SystemCode>  <ModelCode>模块代号</ModelCode>  <ApplicationOrderCode>00001-abcde52-968523-abc222</ApplicationOrderCode>  <TaskStartDate>2010/04/07</TaskStartDate>  <TaskStartTime>08:10</TaskStartTime>  <ProcessCycle>周期(Year,Day,Month,Hour,Minute)</ProcessCycle>  <ReceiveUser>User0001</ReceiveUser>  <ReceiveRole>Role0002</ReceiveRole>  <MessageBody>消息主体{ReceiveUser}</MessageBody>  <MsgLinkUrl>链接Url</MsgLinkUrl>  <ProcessWcfUrl>abc.svc</ProcessWcfUrl>  <WcfFuncName>CreateTimer</WcfFuncName>  <WcfFuncParamter>    </WcfFuncParamter>  <TriggerType>System/User</TriggerType>  <WcfParamSplitChar>Г</WcfParamSplitChar>  <WcfBinding>CustomBinding</WcfBinding>  </System> | ApplicationOrderCode：业务数据Guid  TaskStartDate:必须是：2010/05/04  TaskStartTime:必须是小时与分钟 如08:10  ProcessCycle：任务处理周期  ProcessWcfUrl：需要调用的Wcf地址  WcfFuncName:需要调用的Wcf方法  WcfFuncParamter: 需要调用的Wcf方法的参数  WcfParamSplitChar: 需要调用的Wcf方法的参数之  间分格符,必须为 “Г”  TriggerType：触发类型，是系统级还是用户级（System,User）  WcfBinding:是CustomBinding |

### 方法CancelEventTriggerData

取消定时触发

#### 原型

bool CancelEventTriggerData(string strApplicationOrderCode);

参数

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名 | 类型 | 描述 | 操作 |
| strApplicationOrderCode | string | 保存定时时使用的ApplicationOrderCode | 必填 |
|  |  |  |  |
|  |  |  |  |

### 方法TaskMsgClose

待办任务关闭方法

#### 原型

[OperationContract(IsOneWay = true)]

void TaskMsgClose(string strSystemCode, string strFormID, string strReceiveUser);

#### 参数

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名 | 类型 | 描述 | 操作 |
| strSystemCode | string | 系统代号 | 必填  OA、HR、LM等 |
| strFormID | string | 表单ID | 必填 |
| strReceiveUser | string | 接收用户 | 必填 |

### 方法MsgClose

待办任务与定时触发关闭

#### 原型

[OperationContract(IsOneWay = true)]

void MsgClose(string strMessageID, string strEventID);

#### 参数

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名 | 类型 | 描述 | 操作 |
| strMessageID | string | 待办任务ID | 可选 |
| strEventID | string | 定时触发ID | 可选 |
|  |  |  |  |

### 方法ModelMsgClose

待办任务关闭方法（此方法慎用，将关闭该用户在该模块下的所有待办任务）

#### 原型

void ModelMsgClose(string strModelCode, string strReceiveUser);

#### 参数

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名 | 类型 | 描述 | 操作 |
| strModelCode | string | 模块代号 | 必填 |
| strReceiveUser | string | 接收用户 | 必填 |
|  |  |  |  |

### 方法ApplicationMsgTrigger

不使用流程驱动情况下，发送消息或待办任务

#### 原型

void ApplicationMsgTrigger(List<CustomUserMsg> UserAndForm, string SystemCode, string ModelCode, string strXml, MsgType msgType);

#### 参数

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名 | 类型 | 描述 | 操作 |
| UserAndForm | List<CustomUserMsg> | CustomUserMsg:  UserID:用户ID  FormID:表单ID | 必填 |
| SystemCode | string | 系统代号 | 必填 |
| ModelCode | string | 模块代号 | 必填 |
| strXml | string | 业务系统实体转换成引擎规定格式的XML | 选填 |
| msgType | MsgType | 类型 | Task:待办任务  Msg:消息 |

### 方法ApplicationNotesTrigger

不使用流程驱动情况下，发送消息此消息内容可以自己定义

#### 原型

[OperationContract(IsOneWay = true)]

void ApplicationNotesTrigger(List<CustomUserMsg> UserAndForm, string SystemCode, string Content);

#### 参数

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名 | 类型 | 描述 | 操作 |
| UserAndForm | List<CustomUserMsg> | CustomUserMsg:  UserID:用户ID  FormID:表单ID | 必填 |
| SystemCode | string | 系统代号 | 必填 |
| Content | string | 消息内容 | 必填 |
|  |  |  |  |
|  |  |  |  |

### 方法发送邮件

不使用流程驱动情况下，发送消息此消息内容可以自己定义

#### 原型

[OperationContract(IsOneWay = true)]

void SendMail(List<MailParams> mailParams);

#### 参数

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名 | 类型 | 描述 | 操作 |
| mailParams | List<MailParams> | [DataContract]  public class MailParams  {  [DataMember]  public string ReceiveUserMail  { get; set; }  [DataMember]  public string MailTitle  { get; set; }  [DataMember]  public string MailContent  { get; set; }  } | 必填  ReceiveUserMail：接收邮箱  MailTitle:  邮件标题  MailContent:  邮件内容 |

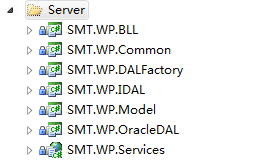
# 部署指南

部署以工作计划(WP)为例。

## 服务端(后台)部署

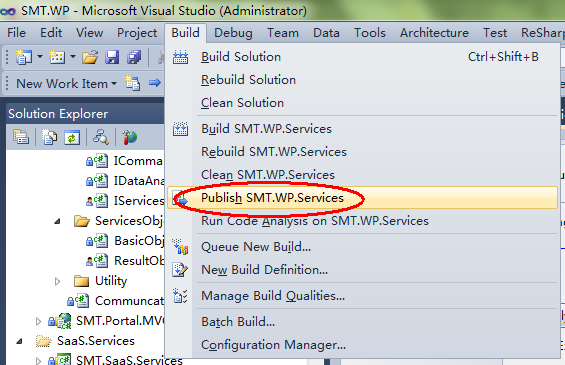
### 打包编译

工作计划服务端工程结构图如下：

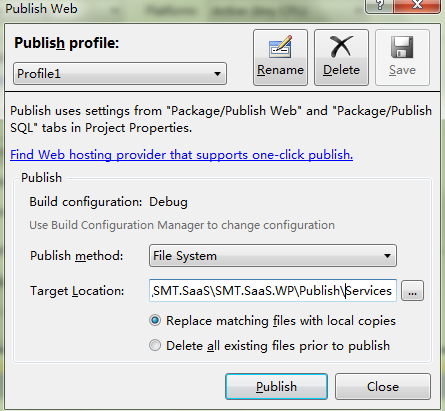


其中，SMT.WP.Services为服务入口工程。

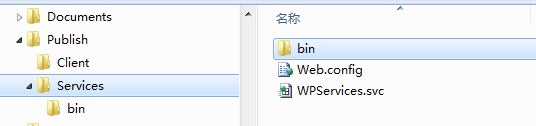
在VS中选中SMT.WP.Services，点击菜单”Build”，选择”Publish”：

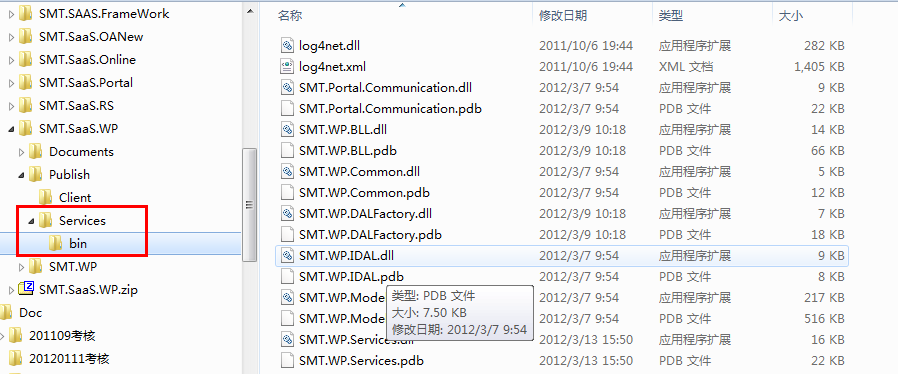


选择发布类型和目录：



点击”Publish”，将会在指定目录生成发布的文件(本例为Services目录)：

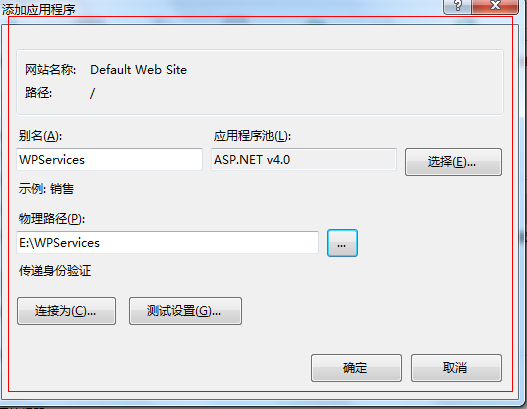




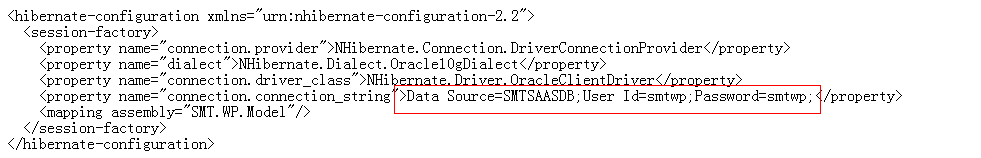
整个Services目录下的文件即为我们需要发布的全部文件。

### 部署到Web服务器

1. 将整个Services文件夹拷贝到Web服务器相应的位置，在程序🡪控制面板🡪管理工具🡪Internet信息服务器(IIS)管理器下，打开IIS配置管理界面；
2. 在网站🡪默认站点🡪添加应用程序，右键点”添加应用程序”，输入相应的别名，选择.net framework 4.0应用池，物理路径选项中选择刚拷贝到服务器上的文件夹后, 参看界面如下图所示：



1. 按”确定”，后台WCF服务配置完成;
2. 修改Web.config配置文件，在拷贝到服务器上的Services文件夹下，找到web.config配置文件，修改hibernate-configuration连接数据库配置信息。



1. 测试WCF配置是否成功，在刚配置完成的应用程序中，点击”WPServices.svc”文件，按“浏览”后，如出现以下界面，表示出现以下界面，表示WCF配置成功。

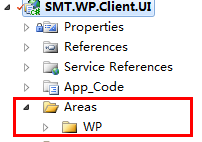


## 客户端(前台)部署

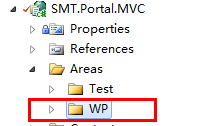
### 打包编译

工作计划客户端工程结构如下：





根据客户端开发规范，所有的工作计划客户端UI部分均实现在WP的工作区中，在编译打包时，由平台打包管理员统一进行处理，将Areas/WP整个目录集成(直接文件拷贝或者由源代码管理器获取)到门户工程的Areas中，如下：



门户工程需要引用SMT.WP.Client.UI工程所引用的所有程序集。

然后进行Publish，具体过程请参考服务端打包编译。

### 部署到Web服务器

1. 将整个工作计划WP文件夹拷贝到Web服务器平台文件夹Portal/Areas服务器相应的位置；
2. 将发布工作计划的三个dll文件拷贝到Web服务器平台文件夹Portal/bin文件夹下,dll文件如下图所示



1. 修改Web服务器平台文件夹Portal下的Web.config文件工作计划服务的地址,服务地址为9.1.2节部署的WCF地址，如下图所示：



# 元数据规范

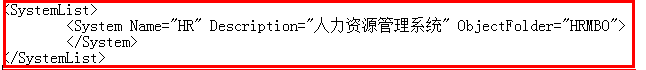
## BOSystemList.xml

### 概述

BOSystemList.xml：系统业务对象存放目录

### 节点属性说明

元数据文件：



节点属性说明：

SystemList：系统列表集；

System：具体系统节点；

Name：系统标识简称；

Description：系统描述；

ObjectFolder：表示人力系统业务对象存放在目录HRMBO中。

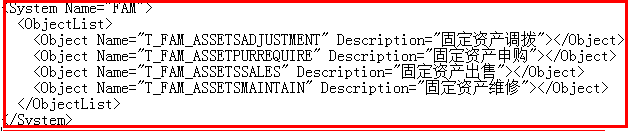
## BOList.xml

### 概述

BOList.xml：业务对象清单，用于保存所属系统所有对象清单。

### 节点属性说明

元数据文件：



节点属性说明：

1.System节点

System:具体系统节点；

Name：业务系统简称；

2. ObjectList节点

ObjectList：系统内业务对象列表；

Object：系统内业务对象节点；

Name：系统内业务对象名，一般与数据库物理表主表名一致；

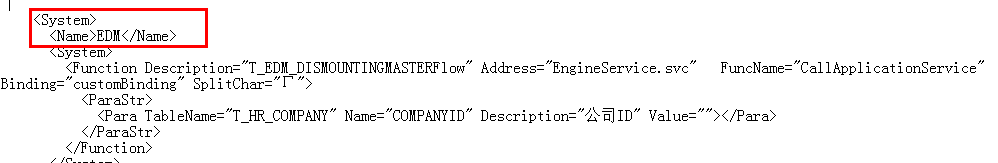
Description：系统内业务对象名描述。

## 分系统内其他XML文件

### 概述

分系统内其他XML文件：为具体的业务对象XML

### 节点属性说明



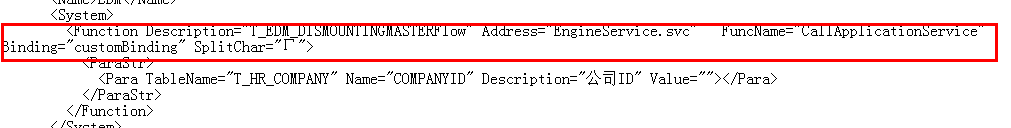
以System为标记。这个节点是向前兼容原有流程和引擎的配置，具体的含义由现有的流程和引擎系统定义。以下为属性节点具体说明：

#### Name节点

Name : 所属系统

#### 子System节点

##### Function节点



Description：流程描述；

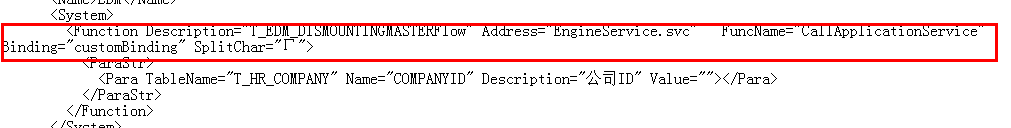
Address：WCF远程引用地址；

FuncName：函数名称；

Binding：绑定方法；

SplitChar：分解字符；

##### ParaStr节点



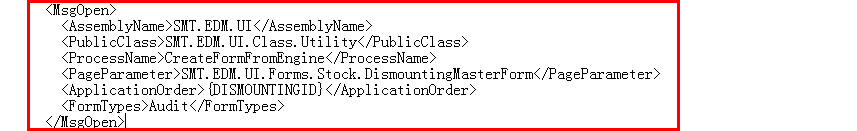
TableName：表名；

Name：名称；

Description：描述

Value：值

#### MsgOpen节点



AssemblyName：所属程序集名称（把它理解为表单的名称空间）；

PublicClass：公共类（表单对应的类名）；

ProcessName：公共方法（调用表单的方法名）；

PageParameter：参数-调用的模块名称；

ApplicationOrder：表单所需要的参数，如果需要多个参数，用分隔符{}分隔；

FormTypes：表单按钮状态；

KPI：判断 "IsKpi" （包含在 APPFIELDVALUE中）确定表单是否需要走KPI，值为0、1；

##### Object节点：



Name : 名称；

LableResourceID ： 资源ID；

Description ：备注；

DataType:数据类型(attachmentlist:附件类型，RTF：富文本)；

DataValue:属性值；

DataText:属性文本；

Key:主键；

IsAttach:是否带附件；

IsRTF:是否是富文件数据；

IsVisible：是否显示；

IsEncryption:是否加密；

ObjectList:子表嵌套；

如果DataType属性为attachmentlist，需要调用附件列表，参数为Value属性值

如果DataType属性为RTF，需要调用富文本接口，参数传递Value属性值获取富文本内容

##### Attribute节点：



Name：名称；

Description：描述；

IsVisible（0：不显示，1：显示）；

DataType：数据类型(attachmentlist:附件类型，RTF：富文本，string:字符类型，decimal:数字)；

DataValue：属性值；

#### BusinessObject节点

##### ObjectName节点



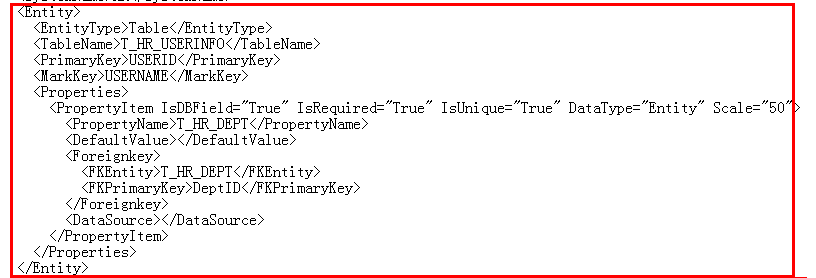
ObjectName：业务对象名称，非空节点；

##### SystemName节点



SystemName：所属系统，非空节点；

##### Entity节点



此部分主要是定义实体部分

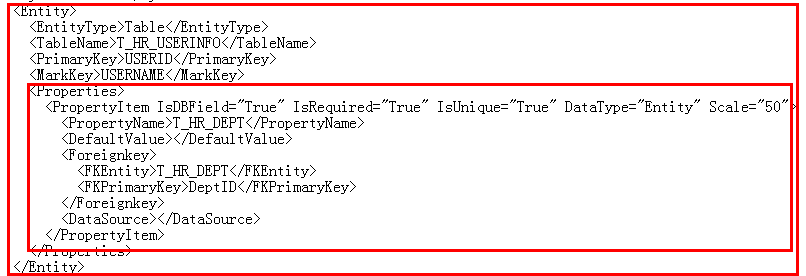
EntityType：实体类型，暂定两种：Table对应数据表，View对应视图，可选节点，默认为数据表，待扩展；

TableName：数据表或者视图名称；

PrimaryKey：主键，非空节点；

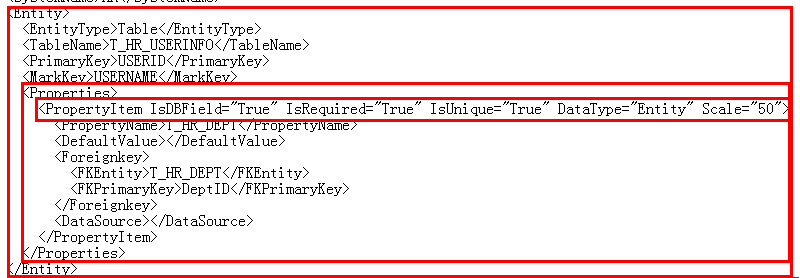
MarkKey：除主键外可以用来标示该实体的字段，一般用于显示交互，可选节点；

##### Properties节点



所有字段，对应实体中的公开字段

##### PropertyItem节点



IsDBField：是否为数据库的字段，不输入的情况下默认是True；

IsRequired：是否为非空字段，不输入默认否；

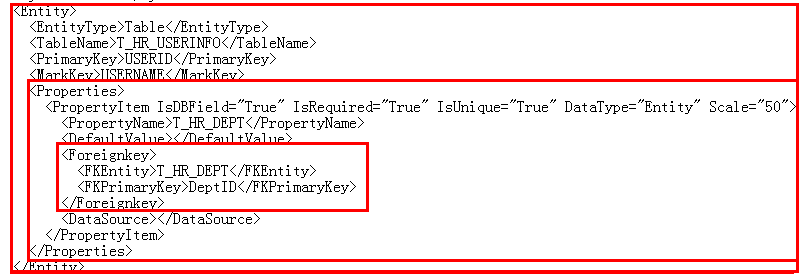
IsUnique：是否为唯一字段，不输入默认否；

DataType：数据类型，暂定String，Decimal，DateTime，Enum(定义为字符串形式），Bool，Entity Scale：尺度，如String类型，标示长度，如数值型标示精度，待扩展。

PropertyName：字段名称，非空节点；

DefaultValue：默认值，可选节点；

##### Foreignkey节点

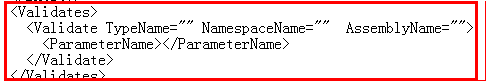


FKEntity：外键对象；

FKPrimaryKey：外键对象的主键；

DataSource：来源,待扩展如枚举类型的选项；

##### Validates节点



验证定义、使用IOC模式可以动态嵌入验证

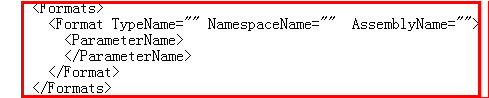
TypeName：验证类的类型；

NamespaceName：验证类所在的名称空间；

AssemblyName：验证类所在的程序集；

ParameterName：ParameterName指的是该验证需要的参数名称,可设置多个；

##### Formats节点



格式转换，如某个字段的大小写转换，如布尔值在界面上显示为“是”或者“否”，待扩展。

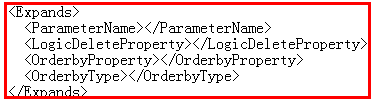
TypeName：验证类的类型；

NamespaceName：验证类所在的名称空间；

AssemblyName：验证类所在的程序集；

ParameterName：指的是该格式转换需要的参数名称，可设置多个；

##### BusinessDefine节点



BusinessDefine主要是业务属性定义，此部分有待扩展；

BusinessName：业务属性名；

##### Expands节点

扩展属性，所有扩展属性存储在字典中,字典的Key为ParameterName，字典值为ParameterName节点的值，统一存储为字符串；这个节点可用于不修改已有代码的情况下，加上附属的配置信息，常用于业务对象需要扩展业务，而这种业务又需要特定的配置信息；

ParameterName：参数名称；

LogicDeleteProperty：设置逻辑删除标志字段，该字段为字符串类型。该节点不为空，则删除时将该字段置为1，表示该实体已删除，如果该节点没有定义则物理删除，可选节点；

OrderbyProperty：列表查询默认排序字段，可选节点；

OrderbyType：列表查询默认排序方式，asc或者desc，默认为desc，可选节点。

### 参看示例

1. BusinessObject.xml示例文件

<?xml version="1.0" encoding="utf-8"?>

<System>

<Name>HRM</Name>

<Object Name="VacationRequest" LableResourceID="VacationRequest" IsAttach="False" IsRTF="True" Description="休假申请单">

<Attribute Name="EmployeeID" LableResourceID="EmployeeID" Description="员工号" DataType="String" DataValue="" DataText="" IsVisible="1"></Attribute>

<Attribute Name="EmployeeName" LableResourceID="EmployeeID" Description="员工姓名" DataType="decimal" DataValue="" DataText="" IsVisible="0"></Attribute>

<Attribute Name="StartDate" LableResourceID="EmployeeID" Description="请假开始时间" DataType="datetime" DataValue="" DataText="" IsVisible="1"></Attribute>

<Attribute Name="VacationDate" LableResourceID="EmployeeID" Description="请假天数" DataType="decimal" DataValue="" DataText="" IsVisible="1"></Attribute>

<ObjectList>

<ReferenceObject ObjectSystem="OA" Name="ApprovalDetail" LableResourceID="ApprovalDetail" ReferenceID="ApprovalID" Description="测试单明细" />

<ReferenceObject ObjectSystem="HR" Name="SYS\_Dictionary" LableResourceID="Dictionary" ReferenceID="DictionaryID" Description="公共字典" />

</ObjectList>

</Object>

</System>

<Name>HRM</Name>表示此业务对象属于HRM系统；

<Object Name="VacationRequest" Description="休假申请单">表示对象名为VacationRequest，描述为休假申请单；

<Attribute Name="EmployeeID" LableResourceID="EmployeeID" Description="员工号" DataType="String" DataValue="" DataText="" IsVisible="1"></Attribute>表示有一属性EmployeeID，LableResourceID描述为员工号，资源ID为EmployeeID，字符串类型，值为空,可以显示。

<ObjectList >

<ReferenceObject ObjectSystem="OA" Name="ApprovalDetail" LableResourceID="ApprovalDetail" ReferenceID="ApprovalID" Description="测试单明细" />

表示业务对象有关联对象，对象为OA系统中的ApprovalDetail，关联键为 ApprovalID

2.Attribute属性节点示例

<Attribute Name="CREATEUSERID" Description="单据创建人ID" DataType="string" DataValue="" IsVisible="0"></Attribute>

<Attribute Name="CREATEUSERNAME" Description="单据创建人姓名" DataType="string" DataValue="" IsVisible="0"></Attribute>

<Attribute Name="CREATECOMPANYID" Description="单据创建人公司ID" DataType="string" DataValue="" IsVisible="0"></Attribute>

**属性值说明**

**IsVisible（0：不显示，1：显示）**

**DataValue(string:字符类型，decimal:数字)**