# Yaheen Java主框架说明文档

**文档版本：V2.1.4**

**代码版本：V2.2.2**

## 框架目录

### 简述

整个主框架的项目结构都是基本的Web项目形式，src是存放Java源代码，sql是存放更新或者初始化的sql语句，WebContent是存放网页文件，work是运行时文件，build是构建文件，主要是前三个文件比较重要。

### src文件夹（Java文件结构）

所有的Java类都存放在这里，所有的Java文件都必须放在net.yasion下面，common是主框架的Java代码，demo是样例Java代码，而且每一个新项目都要在net.yasion下面重新新建一个属于自己项目的文件夹（注意：不要使用demo作为自己的文件夹）。

Java类存放文件夹结构（这里是以主框架[common]为例）：

annotation：注解类

constant：常量类

core：框架核心类

dao：DAO层接口以及实现类

dao.impl：DAO接口实现类

dto：数据转换层类

model：实体类

support：相关支持的组件

service：业务逻辑层接口以及实现类

service.impl：业务逻辑层接口实现类

utils：工具类

web：Web展现层控制类

web.action：Action类

web.interceptor：拦截器类

web.listener：监听器类

web.resolver：解析器类

web.tag：标签类

web.view：视图类

web.processor：处理器类

web.binder：捆绑器类

web.validator：校验期类

一般来说，所有项目的Java代码结构都需要上面的基本文件夹（core除外），其他项目必须参照这些结构来创建相应的文件夹。

### sql文件夹

所有的sql更新或者初始化sql文件都必须存在在这里，方便其他开发人员接手时候，直接可以在本地运行起来。每一个新项目都要在sql下面重新新建一个属于自己项目的文件夹。

### WebContent文件夹

#### 简述

前台页面都存放在这个文件夹下面，包括页面文件，比较特殊的还包括配置文件、属性文件。WebContent文件夹结构，static是存放js、css、img的静态文件，upload是上传文件时候存放文件的文件夹，WEB-INF是存放Web项目私有数据的文件夹，所以所有的jsp文件以及配置文件都必须放在这个文件夹内！

#### static文件夹

基本的文件结构：

static/common：主框架的静态文件目录（禁止存放其他项目的文件）

static/custom：

自定义项目的静态文件目录，比如demo的静态文件就放在里面，所以每个新项目都需要在下面新建一个文件夹，来存放自己的静态文件，具体还要区分开js、css、img、html的文件夹，可以参照主框架文件夹的结构。

static/framework：主体框架的UI目录。

static/frontend：前台框架的静态文件目录

static/plugins：

前台插件目录，就是一些JQuery插件。如果有新增插件需要放在这个目录内。

#### upload文件夹

上传文件的存放目录，每个新项目都需要在下面新建一个自己的文件夹来存放自己的上传文件。

#### WEB-INF文件夹

基本的文件结构：

WEB-INF/cache：hibernate缓存存放目录

WEB-INF/config：配置文件的存放目录

WEB-INF/config/hibernate/mapping：hibernate映射文件目录

WEB-INF/config/hibernate/cache：hibernate缓存配置文件目录

WEB-INF/config/properties：属性文件目录

WEB-INF/config/properties/constant：所有可配置常量的目录

WEB-INF/config/properties/jdbc：JDBC可配常量目录

WEB-INF/config/properties/logger：日志可配常量目录

WEB-INF/config/spring：spring配置文件目录

WEB-INF/config/tld：标签配置文件目录

WEB-INF/lib：jar包目录

WEB-INF/logs：运行日志存放目录

WEB-INF/pages：

jsp存放目录，每个新项目都需要在下面新建一个自己的文件夹来存放自己的jsp文件，并且jsp文件需要按模块来划分存放。

### 注意事项

每一个新项目，都需要新建自己的文件夹，来存放自己的文件，下面列出一下：

|  |  |  |
| --- | --- | --- |
| 路径 | 作用 | 备注/例子 |
| src/net.yasion | 存放新项目的Java文件 | net.yasion.demo |
| sql | 存放新项目的sql文件 | sql/demo |
| WebContent/static/custom | 存放新项目的静态文件 | WebContent/static/custom/demo，js、css、img、html文件还需要按模块划分一下文件夹，可以参考WebContent/static/common |
| WebContent/WEB-INF/config/hibernate/mapping | 存放新项目的Hinerate映射文件 | WebContent/WEB-INF/config/hibernate/mapping/demo |
| WebContent/WEB-INF/pages | 存放新项目的JSP网页文件 | WebContent/WEB-INF/pages/demo，jsp页面还需要按模块划分一下文件夹，可以参考WebContent/WEB-INF/pages/common |
| WebContent/WEB-INF/config/tld | 存放新项目的tld配置文件 | WebContent/WEB-INF/config/tld/demo |
| WebContent/upload | 存放新项目的上传文件 | WebContent/upload/demo |

具体的其实都可以参考一下common的文件目录结构。

## 项目类型

### 简述

由于历史原因，主框架是分开两种的项目类型的——无配置文件项目和配置文件项目。svn上也有两个不同的框架目录区分这两种项目。

### 无配置文件项目

#### 简述

所谓的无配置文件就是Spring 3.1版本后支持的完全不需要任何配置文件就可以启动的项目，甚至连web.xml也不需要。不过由于这种方式未对标签作支持，所以暂时还是保留了web.xml，主要就是用来注册标签。其实这个方式是把配置文件写成累的形式，并不是真的不需要任何配置，就是把配置改成了类的实现。net.yasion.common.core.initializer.WebInitializer、net.yasion.common.core.initializer.config. ApplicationContextConfig、net.yasion.common.core.initializer.config.SpringMVCConfig就是web.xml、applicationContext.xml、spring-servlet-config.xml的对应实现，其目的就是提供封装性，提供了灵活性。凡是有net.yasion.common.core.initializer文件夹的项目都是无配置文件项目。具体无配置文件项目实现方式的请参考网上spring资料。

#### 项目目录

svn://svn.yaheen.com/projects/trunk/yaheen

凡是这个目录的分支，都是无配置文件项目。

### 配置文件项目

#### 简述

一般的web项目的形式。

#### 项目目录

svn://svn.yaheen.com/projects/branches/yasion

凡是这个目录的分支，都是配置文件项目。

## 框架基础文件

### Java类

#### IBaseDAO

##### 作用

IBaseDAO接口属于数据库层（DAO）的基础接口，声明了许多基础方法，所以所有操作DAO层的接口必须先继承该接口，已提供相应的基本支持。

#### BaseDAOImpl

##### 作用

BaseDAOImpl为IBaseDAO接口提供基础实现的抽象类，所有操作DAO层的实现类一般都需要继承该类，已提供基础的实现。并且BaseDAOImpl必须实现getEntityClass方法，目的是为了得到对应操作实体的类类型。

#### BaseDTO

##### 作用

BaseDTO为数据转换层（DTO）的基础类，它提供了一般的基础字段以被展现层标志或者控制，并且这些字段跟BaseModel的部分字段类似，其目的为了方便展现层和DAO层数据的转换。所以一般数据转换层的类都需要继承该类。

#### BaseModel

##### 作用

BaseModel为实体层（model）的基础类，它提供了一般的基础字段，基础字段跟数据库以及映射文件的字段一一对应。任何实体类都必须继承该类。

#### IBaseService

##### 作用

IBaseService为事务逻辑层（service）的基础接口，声明了许多基础的方法，所以所有操作事务逻辑层的接口必须先继承该接口，已提供相应的基本支持。

#### BaseServiceImpl

##### 作用

BaseServiceImpl为IBaseService接口提供基础实现的抽象类，所有操作业务逻辑层的实现类一般都需要继承该类，已提供基础的实现。并且BaseServiceImpl必须实现getDefaultDAO方法，目的是为了得到操作对应默认实体的DAO对象。

比如：public class LogServiceImpl extends BaseServiceImpl<TbLog, LogDTO, String>的默认DAO对象就是IBaseDAO<TbLog, String>

#### BaseAction

##### 作用

BaseAction为展现层（action）的基础类，一般所有展现层的类都需要继承该类。

#### 泛型绑定

上面提到的Java类，都通过泛型来绑定了操作对象的，其目的是为了提供内聚，降低耦合。具体说明一下；

POJO：该泛型出现在DAO和业务逻辑层，代表可以操作的实体类型。其绑定为POJO extends BaseModel<ID>，意思就是其操作实体必须是继承于BaseModel的实体，并且其ID类型必须与当前指定ID类型一致。

ID：该泛型出现在DAO、业务逻辑层、实体层、数据转换层，代表实体的ID类型，其绑定为ID extends Serializable，意思就是这些ID必须是可以序列化的(Hibernate本身也是使用Serializable类型的ID)。

DTO：该泛型出只现在业务逻辑层，代表可以操作的DTO类型。其绑定为DTO extends BaseDTO<ID>，意思就是其操作数据转换类必须是继承于BaseDTO的数据转换类，并且其ID类型必须与当前指定ID类型一致。

#### 注意事项

开发时候一般来说，所有的DAO层接口都需要继承IBaseService接口；DAO层接口的实现类必须继承BaseServiceImpl抽象类，并且实现对应的接口。

比如：

接口

public interface ILogService extends IBaseService<TbLog, LogDTO, String>

实现类

public class LogServiceImpl extends BaseServiceImpl<TbLog, LogDTO, String> implements ILogService

业务逻辑层也是跟DAO层类似。

比如：

接口

public interface ILogService extends IBaseService<TbLog, LogDTO, String>

实现类

public class LogServiceImpl extends BaseServiceImpl<TbLog, LogDTO, String> implements ILogService

### Hibernate映射文件

#### BaseModel.hbm.xml

BaseModel.hbm.xml是所有映射文件的基础文件，其定义基础字段，所以所有的映射文件必须继承该文件（实体类）。

#### 注意事项

因为主框架使用了Hibernate的继承映射的方式来制定，所以除了BaseModel.hbm.xml文件以外，所有文件的类定义方式需要使用union-subclass方式。由于使用了union-subclass方式，所以id的生成器必须使用uuid.hex，否则两个表的id相同时会造成程序出错。也因此数据库的id字段需要32个字节。

具体的映射文件配置如下：

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

<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"

"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<!-- Mapping file autogenerated by MyEclipse Persistence Tools -->

<hibernate-mapping>

<union-subclass name="net.yasion.common.model.TbLog" extends="net.yasion.common.model.BaseModel" table="tb\_log">

<property name="title" type="java.lang.String">

<column name="title" length="128" />

</property>

<property name="loggerType" type="java.lang.String">

<column name="logger\_type" length="32" />

</property>

<property name="userId" type="java.lang.String">

<column name="user\_id" />

</property>

<property name="userName" type="java.lang.String">

<column name="user\_name" />

</property>

<property name="ipAddress" type="java.lang.String">

<column name="ip\_address" length="128" />

</property>

<property name="url" type="java.lang.String">

<column name="url" />

</property>

<property name="logTime" type="java.lang.String">

<column name="log\_time" length="64" />

</property>

<property name="content" type="java.lang.String">

<column name="content" length="512" />

</property>

</union-subclass>

</hibernate-mapping>

### Javascript文件

#### base.js

框架级别的JS写在这里，一般的页面都需要引入该文件，作为基础的方法支持。不过一般通过<c:import url="/WEB-INF/pages/common/default/header.jsp" />引入。

### CSS文件

#### base.css

框架级别的CSS写在这里，一般的页面都需要引入该文件，作为基础的方法支持。不过一般通过<c:import url="/WEB-INF/pages/common/default/header.jsp" />引入。

### JSP文件

#### cssHeader.jsp

基础css的引入页面。一般用于JSP的引入。包括一些前台插件的css文件和base.css文件。

#### globalHeader.jsp

Js全局变量的引入页面。一般用于JSP的引入。

包括：

baseUrl：项目路径

jsessionId：当前的cookies形式sessionId

sessionId：当前的sessionId

#### jsHeader.jsp

基础Js的引入页面。一般用于JSP的引入。包括一些前台插件的Js文件和base. js文件。

#### modalDialog.jsp

基础窗口的引入页面。一般用于JSP的引入。这些窗口已经被插件控制可以直接使用。

包括：

alertDialog：警告窗口

successDialog：成功提示窗口

confirmDialog：确认提示窗口

#### ueEditorHeader.jsp

UE编辑器（百度编辑器）的引入页面。一般用于JSP的引入。

#### header.jsp

前面所有jsp文件的引入文件（ueEditorHeader.jsp、modalDialog.jsp需要单独引入，并且modalDialog.jsp一般是在页面尾部引入）。因此其实一般来说只要引入这个页面就可以的了。不过分开来写是为了避免特殊情况或者一些冲突情况发生。

引入方式：

<c:import url="/WEB-INF/pages/common/default/header.jsp" />

## 框架注解

### BeanUtilsNoCopyValue

#### 作用

该注解作用在AfxBeanUtils.copySamePropertyValue方法，当该方法的isUseNoCopyValue参数为true的时候，被BeanUtilsNoCopyValue注解修饰的Field的值将不会被复制。

#### 作用域

所有POJO类型的Field

#### 处理方法

AfxBeanUtils::public static void copySamePropertyValue(Object src, Object des, boolean isUseNoCopyValue, String[] ignoreProperties)

### ConverterArrayParam

#### 作用

在页面提交，准备将页面的数据绑定到展现层的映射方法的对应参数的时候，被ConverterArrayParam注解修饰的映射方法参数，将会找出与其参数名相同的页面数据，把其转换为数组的形式，其数组类型就是参数类型。

#### 数据格式

由于是处理页面传过来的数据，所以对传过来的格式用要求，

比如：

参数类型：String[] ids

页面数据格式：ids=19,20,23 或者 ids=19;20;23

#### 作用域

展现层的映射方法中的参数

#### 处理方法

AnnotationArgumentResolver::protected Object resolveConverterArrayParam(MethodParameter parameter, ModelAndViewContainer mavContainer, NativeWebRequest webRequest, WebDataBinderFactory binderFactory) throws Exception

### ModelAliasLink && Alias

#### 定义

ModelAliasLink

thisAlias: 指定当前Field的别名(用于多层连接)

alias: 通过Alias指定多层连接情况

Alias

item: 一个别名的组合，可以多级别名

alias: 对应item的别名

#### 作用

在BaseDAOImpl创建Criteria时候，通过ModelAliasLink指定当前实体创建的Criteria的别名，其目的就是实现实体查询时候关系映射的自动连接（默认情况下Hibernate是不会自动连接当前实体的关系映射实体的，这种情况下，多级映射的查询会报错，不过通过id的还是会生效的），并且通过Alias可以实现多级的自动连接。

#### 作用域

ModelAliasLink：所用POJO的关系映射Field

Alias：ModelAliasLink内

#### 处理方法

BaseDAOImpl:: protected final Criteria createCriteria(Criterion[] criterions, String alias)

#### 使用说明

鉴于改注解使用比较复杂所以需要一个特别的说明。

例子：

一级连接情况：

net.yasion.tsdp.model.TbAbilityCourse

@ModelAliasLink

private TbCourse tbCourse;

直接将tbCourse连接起来，可以通过tbCourse.name=’XXX’查询。

多级连接情况：

net.yasion.tsdp.model.TbAbilityCourse

@ModelAliasLink(thisAlias = "tbWorkAbility", alias = { @Alias(item = "tbWorkAbility.tbWorkTask", alias = "tbWorkAbilitytbWorkTask") }) //注解

private TbWorkAbility tbWorkAbility; //字段

首先必须要为当前字段定义好别名，一般都是于字段名一样，上面使用了tbWorkAbility，然后进行多接连接，将tbWorkAbility下面的tbWorkTask字段(另外一个实体的关系映射)声明为一个item，指定别名为tbWorkAbilitytbWorkTask，这样在当前查询时候可以直接通过tbWorkAbilitytbWorkTask.name=’XXX’来表明查询当前实体对象的tbWorkAbility属性的tbWorkTask属性的name字段为XXX，默认情况下是会报错的，因为它没有连接起来。

假如还有多几级，就可以写成：

net.yasion.tsdp.model.TbAbilityCourse

@ModelAliasLink(thisAlias = "tbWorkAbility", alias = { @Alias(item = "tbWorkAbility.tbWorkTask", alias = "tbWorkAbilitytbWorkTask"),@Alias(item = "tbWorkAbilitytbWorkTask.tbWorkProject", alias = "tbWorkAbilitytbWorkTasktbWorkProject"),@Alias(item = "tbWorkAbilitytbWorkTasktbWorkProject.tbProject", alias = "tbWorkAbilitytbWorkTasktbWorkProjecttbProject") }) //注解

private TbWorkAbility tbWorkAbility; //字段

### ModelDTOCriterion

#### 作用

该注解是为了方便页面查询的编码，特别在复杂的页面查询中。在页面提交，准备将页面的数据绑定到展现层的映射方法的对应参数的时候，被ModelDTOCriterion注解修饰的映射方法参数（必须是BaseDTO类型或其子类），将会调用参数的generateDefCriterion方法，这个方法的两个参数是requestURL和paramsMap，就是请求的Url和请求包含的参数。如果是复杂的查询，只用在DTO中覆盖generateDefCriterion方法，就可以实现查询对象Criterion的控制，并且在generateDefCriterion最后需要显式的对defCriterion设置值，否则是不生效的！当后台查询时候，就会使用defCriterion作为查询对象来查询。

#### 作用域

局限于展现层的映射方法中DTO类型(? extends BaseDTO<ID>)参数，并且一般只用于查询和列表方法的参数上，用在保存和更新的方法参数可能会出问题。

#### 处理方法

AnnotationArgumentResolver::protected Object resolveModelDTOCriterion(MethodParameter parameter, ModelAndViewContainer mavContainer, NativeWebRequest webRequest, WebDataBinderFactory binderFactory) throws Exception

### ModelDTOSearch

#### 作用

该注解是为了方便页面查询的编码，特别在复杂的页面查询中。在页面提交，准备将页面的数据绑定到展现层的映射方法的对应参数的时候，被ModelDTOCriterion注解修饰的映射方法参数（必须是BaseDTO类型或其子类），将会调用参数的generateDefOperateRelation方法，这个方法的两个参数是requestURL和paramsMap，就是请求的Url和请求包含的参数。这个方法是定义页面传过来的参数的查询时候的等价关系的，就是比如id需要eq的关于，name需要like的关系，createTime需要between…and的关系。只要覆盖generateDefOperateRelation方法，页面的参数将会按指定的关系来查询，并且在generateDefOperateRelation最后需要显式的对operateRelation/operateRelationMap设置值，否则是不生效的！

#### 作用域

局限于展现层的映射方法中DTO类型(? extends BaseDTO<ID>)参数，并且一般只用于查询和列表方法的参数上，用在保存和更新的方法参数可能会出问题。

#### 处理方法

AnnotationArgumentResolver::protected Object resolveModelDTOSearch(MethodParameter parameter, ModelAndViewContainer mavContainer, NativeWebRequest webRequest, WebDataBinderFactory binderFactory) throws Exception

### ModelJson

#### 作用

在页面提交，准备将页面的数据绑定到展现层的映射方法的对应参数的时候，被ModelJson注解修饰的映射方法参数，将会把页面接收到的JSON数据（标准的JSON字符串，并不是通过jQuery.parseJSON转换的数据），根据参数类型转换为对应的对象。如果传过来的不是JSON数据，将会出错！

#### 作用域

展现层的映射方法中的POJO类型参数

#### 处理方法

AnnotationArgumentResolver::protected Object resolveModelJson(MethodParameter parameter, ModelAndViewContainer mavContainer, NativeWebRequest webRequest, WebDataBinderFactory binderFactory) throws Exception

#### 注意事项

值得注意的是，如果提交的数据过多，这个注解可能出错。虽然可以生成对象，但是在Action层不能再通过request.getParameter等方式获取到参数，为了避免获取不到参数，这里已经处理了，当出错的时候，可以通过 request.getAttribute(CommonConstants.MODEL\_JSON\_BACKUP\_ATTRIBUTE\_NAME);获取那些json的数据，以便后面使用！

### 注意事项

ModelJson、ModelDTOSearch、ModelDTOCriterion、ConverterArrayParam三个注解是互斥的注解，也就是说，同时在一个参数上使用这个这些注解，只会有一个注解生效。

## 框架常量

### ConfigurableConstants

#### 作用

一般常量类和可用Properties文件配置的常量类的父类。本类既保持了Constants的static和final(静态与不可修改)特性,又拥有了可用Properties文件配置的特征, 主要是应用了Java语言中静态初始化代码的特性。

#### 使用说明

一般不能直接使用，只作为基类继承。读取配置文件的常量子类需要在静态块中调用load方法，以指定配置文件的位置。

### CommonConstants

#### 作用

定义主框架级别所使用的全部常量。一般不要需改。

### ContentTypeConstants

#### 作用

定义所有http请求的contentType，一般用在上传和下载时候检查文件的类型。

#### 使用说明

通过getProperty(String key)方法获取对应的contentType，key为文件扩展名

比如

apk：getProperty("apk") 获取到 application/vnd.android.package-archive

### 注意事项

一般新开一个项目的时候都需要独立建立一个常量类，并继承ConfigurableConstants，常量类一般的命名方式是：项目名+Constants。注意就是不要把项目的常量写到CommonConstants、ConfigurableConstants、ContentTypeConstants中，因为都是框架级别的。

## 框架拦截器

### LoginInterceptor（登录拦截器）

#### 作用

判断当前操作/请求是否已经在登录情况下，没有登录提示登录并且跳转回指定页面。

#### 可设置的成员变量

redirectUrl：无登录时候重定向的地址，默认是/index.do

excludeUrls：排除的请求url，指定那些请求不进行登录检测

### PermissionControllInterceptor（权限控制拦截器）

#### 作用

判断当前用户是否有权限访问这个url，没有权限或者没有登录则提示并且跳转回指定页面。

#### 可设置的成员变量

redirectUrl：无登录时候重定向的地址，默认是/index.do

excludeUrls：排除的请求url，指定那些请求不进行登录检测

defWhiteUrls：默认白名单

defBlackUrls：默认黑名单

### SharedSessionInterceptor（共享Session拦截器）

#### 作用

获取请求参数中指定的参数——jsessionid（jsessionid为sessionId），将指定sessionId的session的属性复制到当前session，实现session共享。

## 框架解析器

### AnnotationArgumentResolver（注解参数解析器）

#### 作用

用于在页面提交数据后，控制页面的数据绑定到展现层的映射方法的对应参数的解析器。SpringMVC 3.1增加的新特性。注解ModelJson、ModelDTOSearch、ModelDTOCriterion、ConverterArrayParam就是通过这个解析器进行处理。具体资料请参考SpringMVC 3.1。

### ExtInternalResourceViewResolver（内部视图解析器）

#### 作用

实现返回页面时候，当返回字符串以“/”开头，则忽略SpringMVC指定视图前缀。就是可以直接返回到WebContent下面的页面。默认情况下只返回到WebContent/pages目录下的页面。

## 框架标签

### yahnc标签库

#### 简述

本框架的使用的核心标签都定义在此。

通过<%@ taglib uri="/yahncore" prefix="yahnc"%>引入

#### operatedUnitSelectInfo

##### 作用

生成“信息所属单位”的选择框，用于用户在属于多个单位时候，保存记录时候需要选择这条记录所属的单位时候使用。

##### 标签属性

isSurroundTable：是否被table包围，一般用于自动生成td、tr包围选择框，默认true

selectStyle：选择框的style属性，默认width: 206px;

unitPropertyName：选择框的name和id的值，默认operatedUnitId

value：单位的值，默认null

##### 标签类

net.yasion.common.web.tag.OperatedUnitSelectInfoTag

#### constants

##### 作用

把指定的常量类内的全部/部分变量写入到页面，使得通过常量的变量名直接在页面访问常量值。

##### 标签属性

clazz：

指定的常量类的详细名称，默认是net.yasion.common.constant. CommonConstants

var：

指定需要的变量名，多个间用逗号隔开，如果没有指定则是全部的变量，默认null

##### 标签类

net.yasion.common.web.tag.ConstantsTag

##### 使用说明

<yahnc:constants />

<yahnc:constants clazz="net.yasion.rigs.constant.RIGSConstants" />

引入后，可以直接通过页面访问变量值

例如：

${COMMON\_DATA\_FORMAT} 直接使用CommonConstants的COMMON\_DATA\_FORMAT，会在页面输出：yyyy-MM-dd HH:mm:ss

#### pagelet

##### 作用

生成分页的选择框

##### 标签属性

url：查询数据的请求url，必填

showOnOne：当小于一页时候是否显示选择框，默认true

script：是否生成操作js脚本，默认true

resultSet：数据集合对象，用于生成分页，必填

##### 标签类

net.yasion.common.web.tag.PageletTag

### yahnfn方法库

#### 简述

本框架的使用的方法标签都定义在此。

通过<%@ taglib uri="/yahnfunction" prefix="yahnfn"%>引入

#### isPermissions

##### 作用

判断指定用户是否有指定的权限，指定权限通过权限code来指定

##### 方法参数

net.yasion.common.model.TbUser：指定的用户

java.lang.String[]：指定权限的code，数组类型

##### 方法返回值

是否有权限，true/false

##### 方法定义

PermissionELFunction:: public static Boolean isPermissions(TbUser user, String[] permissionCodes)

##### 使用说明

<c:if test="${yahnfn:isPermissions(sessionScope.login\_user,fn:split(AWARDAUDIT,','))}">

<button class="btn btn-danger" data-toggle="modal" onclick="doAudit('${status}');">审核</button>

</c:if>

只有当前用户拥有权限编码为AWARDAUDIT的权限才显示审核按钮

#### isRoles

##### 作用

判断指定用户是否有指定的角色，指定角色通过角色code来指定

##### 方法参数

net.yasion.common.model.TbUser：指定的用户

java.lang.String[]：指定角色的code，数组类型

##### 方法返回值

是否有权限，true/false

##### 方法定义

PermissionELFunction:: public static Boolean isRoles(TbUser user, String[] roleCodes)

##### 使用说明

<c:if test="${yahnfn:isRoles(sessionScope.login\_user,fn:split(ADMIN\_ROLE\_CODE,','))}">

<button class="btn btn-danger" data-toggle="modal" onclick="doDelete();">删除</button>

</c:if>

只有当前用户拥有角色编码为ADMIN\_ROLE\_CODE的角色才显示删除按钮

#### 附加

由于部分功能的需要，已经把net.yasion.common.utils.HtmlStringUtils的方法加入到yahnfn方法库中，此处不再作详细说明。

## 框架视图

### ExtInternalResourceView

#### 作用

在request中增加部署路径,共享session字符串,sessionId等属性,方便处理部署路径问题

## 框架监听器

无

## 框架工具类

### 简述

在net.yasion.common.utils下面定义了许多的工具类，开发时候先不要想自己实现，看看这里有没有相应的工具类先，减小开发量、避免代码重复。工具类都是比较安全的一般情况下都可以直接使用。

### AfxBeanUtils（Bean工具类）

#### 作用

该类提供相关Bean操作的帮助，通过反射方式实现。

#### 常用方法

##### copyProperties

###### 作用

按照属性名称，复制源对象属性值到目标对象属性，排除不复制ignoreProperties中指定的字段名称。

###### 参数

Object src：源对象

Object des：目标对象

String[] ignoreProperties：排除的字段名

##### copySamePropertyValue

###### 作用

按照属性名称，复制源对象属性值到目标对象属性，通过属性上BeanUtilsNoCopyValue来控制属性是否复制。当被BeanUtilsNoCopyValue注解的属性将不进行复制。

###### 参数

Object src：源对象

Object des：目标对象

boolean isUseNoCopyValue：是否启用BeanUtilsNoCopyValue注解来控制值的复制

String[] ignoreProperties：指定不进行复制的的字段的名称

##### getFieldValue

###### 作用

获取指定对象指定属性的值。

###### 参数

Object target：指定对象

Class<?> clazz：指定类型，在获取静态属性时候使用

String fieldName：指定的属性名

###### 返回值

指定属性的值

##### setFieldValue

###### 作用

设置指定对象指定属性的值

###### 参数

Object target：指定对象

Class<?> clazz：指定类型，在获取静态属性时候使用

String fieldName：指定的属性名

Object fieldValue：要设置的属性值

##### setFieldValues

###### 作用

设置指定对象多个属性的值

###### 参数

Object target：指定对象

Class<?> clazz：指定类型，在获取静态属性时候使用

String[] fieldNames：指定的多个属性名

Object[] fieldValues：要设置的多个属性值

##### getAllFieldsValue

###### 作用

获取到指定对象所有的属性值，可以指定是否用子类属性值覆盖父类同名属性值

###### 参数

Object target：指定对象

Class<?> clazz：指定类型，在获取静态属性时候使用

boolean isSubOverrideParent：指定是否用子类属性值覆盖父类同名属性值，避免冲突

###### 返回值

以键值对的形式返回指定属性值，形式：属性名-属性值。

##### invokeMethod

###### 作用

执行指定对象的指定方法

###### 参数

Object target：指定对象

String methodName：指定方法名

Object[] argArr：方法对应的参数值

Class<?>[] parameterTypes：参数类型，用于确定唯一的方法

###### 返回值

方法执行后的返回值

##### getField

###### 作用

获取指定类型的指定属性Field对象

###### 参数

Class<?> clazz：指定类型

String fieldName：指定属性名称

###### 返回值

指定属性的Field对象

##### getWrapperObject

###### 作用

把基本类型转换成对应包装类型，可以直接传入基本类型的数组，如果传入的不是基本类型，则返回本身的这个对象

###### 参数

Object baseType：基本类型

###### 返回值

对应的包装类

##### getComplexFieldNames

###### 作用

获取指定类型的复杂数据类型属性名

###### 参数

Class<?> classType：指定类型

List<String> excludesList：排除的属性名

boolean isCheckBaseClass：是否查找父类

###### 返回值

返回对应的属性名

### ExcelUtil

#### 作用

提供Excel操作的帮助类，此类比较特殊，需要实例化，所以就不说明具体方法了。可以参考具体的例子。pms里面有一个例子。

### HibernateUtils

#### 作用

提供Hibernate的基本操作帮助

#### 常用方法

##### getSessionfactory

###### 作用

获取当前的Sessionfactory

###### 返回值

当前的Sessionfactory

##### getCurrentSession

###### 作用

获取当前的Hibernate Session，每次数据库请求都会有一个Hibernate Session。

###### 返回值

当前的Hibernate Session

##### clearCache

###### 作用

强制清除当前Hibernate Session的缓存

##### flushCache

###### 作用

强制进行从内存到数据库的同步

##### evictCache

###### 作用

把指定的缓冲对象进行清除

###### 参数

Object entity：指定的对象

### HtmlStringUtils

#### 作用

处理Html字符串的帮助类，具体方法类似Apache的StringUtils，不做详细说明。

### HttpUtils

#### 作用

Http对象帮助类，包括request、response、cookies等操作功能

#### 常用方法

##### download

###### 作用

触发文件下载，浏览器下载文件

###### 参数

response：请求响应对象

filePath：要下载文件的路径(绝对路径)

showName：下载时候显示的文件，要带上扩展名

###### 返回值

是否成功触发下载

###### 重载

response：请求响应对象

bytes：对应文件的二进制数据

showName：下载时候显示的文件,要带上扩展名

##### upload

###### 作用

上传文件

###### 参数

request：请求对象，必须为MultipartHttpServletRequest实例，否则上传失败

fileFieldName：对应表单的File元素的名称

fileSavePath：文件保存的路径 ，必须为绝对路径，不需要带上文件名

###### 返回值

返回UploadFile对象，对象包含基本上传文件信息

##### getQueryParams

###### 作用

获取所有参数的键值队，并使用URLDecoder以UTF-8格式转码。如果请求是以post方法提交的， 那么将通过HttpServletRequest::getParameterMap获取

###### 参数

request：请求对象

###### 返回值

一个map，[参数名/值数组]

##### getIpAddr

###### 作用

获取访问者IP，在一般情况下使用Request.getRemoteAddr()即可，但是经过nginx等反向代理软件后，这个方法会失效。本方法先从Header中获取X-Real-IP，如果不存在再从X-Forwarded-For获得第一个IP(用,分割)， 如果还不存在则调用Request .getRemoteAddr()。

###### 参数

request：请求对象

###### 返回值

访问者IP

##### copyHttpInternalObjectAttribute

###### 作用

制http内置对象的Attribute(暂时只支持HttpSession/HttpServletRequest/HttpServletResponse类型内置对象的复制)

###### 参数

srcHttpInternalObject：源http内置对象

desHttpInternalObject：目标http内置对象

##### getProxy

###### 作用

获取代理服务配置

###### 参数

hostname：代理服务器的主机名

port：代理服务器的端口

scheme：代理服务器使用的协议

###### 返回值

对应的代理配置

##### createSSLClient

###### 作用

生成SSL的客户端可关闭的请求对象

###### 参数

keyStoreFilePath：密码保护文件路径

keyStorePassword：密码保护文件解密密码,默认为null,即是没有密码

keyStoreType：密码保护文件的类型,默认为Java keystore

keyStoreProvider：密码保护文件的提供者,默认null

trustStrategy：信任策略,默认只相信自己的CA和所有自签名的证书

supportedProtocols：SSL要支持的协议

###### 返回值

返回一个SSL的客户端可关闭的请求对象

##### get

###### 作用

使用代理方式调用get访问指定地址并且获取数据,然后将数据返回

###### 参数

httpClient：指定的httpClient,可以是SSL的Client,提供灵活性,不指定使用默认的

url：访问的地址

proxyConfig：代理配置,可以通过getProxy方法获取

###### 返回值

返回响应的数据

##### post

###### 作用

代理方式调用post访问指定地址并且获取数据,然后将数据返回

###### 参数

httpClient：指定的httpClient,可以是SSL的Client,提供灵活性,不指定使用默认的

postValMap：post过去的参数

url：访问的地址

proxyConfig：代理配置,可以通过getProxy方法获取

###### 返回值

返回响应的数据

##### toDownload

###### 作用

根据指定的url对其执行下载操作

###### 参数

httpClient：指定的httpClient,可以是SSL的Client,提供灵活性

url：指定的地址

postValMap：post过去的参数

proxyConfig：代理配置,可以通过getProxy方法获取

downloadFileSavePath：

下载保存的文件路径,可以是整个文件的绝对路径,可以是目录;如果是目录的时候,baseFileName必须指定

baseFileName：文件名,只包含文件名的,不包含扩展名的文件名

getMethod：是否使用get方式请求

###### 返回值

是否下载执行成功

### MD5Util

#### 作用

MD5帮助类。

### Num62Utils

#### 作用

62进制数字帮助类

### UserUtils

#### 作用

登录用户帮助类，方便操作当前Session中登录的User对象

#### 常用方法

##### getCurrentUser

###### 作用

获取当前Session中已登录的User对象

###### 返回值

当前Session中已登录的User对象

##### setCurrentUser

###### 作用

把指定User对象设置到当前Session中

###### 参数

TbUser user：指定User对象

##### removeCurrentUser

###### 作用

移除当前Session中的User对象

##### refreshCurrentUser

###### 作用

刷新当前Session中的User对象，就是重新加载一次

##### getCurrentUnit

###### 作用

获取当前Session中已登录的User对象的单位对象

###### 返回值

返回当前Session中已登录的User对象

##### isCurrentRealSuperUser

###### 作用

判断当前用户是不是真正的超级管理员，只能是TbSuperUser那种，角色指定的也不行。

###### 返回值

是否真正的超级管理员

### WebContextUtils

#### 作用

提供Web容器的相关操作

#### 常用方法

##### getWebContextRealPath

###### 作用

获取静态获取项目的绝对路径,并且拼接上propertyFileName指定的路径(也就是说获取项目路径下的propertyFileName目录的绝对路径)

###### 参数

propertyFileName：文件路径

###### 返回值

绝对路径

##### getWebContextWebInfRealPath

###### 作用

静态获取项目WEB-INF的绝对路径

###### 参数

propertyFileName：文件路径

###### 返回值

WEB-INF的绝对路径

##### isWebContextPath

###### 作用

判断指定的path是不是基于web context的路径

###### 参数

path：指定路径

###### 返回值

是否web context路径

##### isWebInfPath

###### 作用

判断指定的path是不是基于web-inf的路径

###### 参数

path：指定路径

###### 返回值

是否web-inf路径

## 框架管理类以及适配器

### SpringBeanManager

#### 作用

Spring Bean 管理类，可以在spring容器中获取指定bean。并且提供当前项目所有Spring容器的访问。

#### 常用方法

##### getAppContextSet

###### 作用

获取当前项目所有的Spring容器，以集合的形式返回

###### 返回值

当前项目所有的Spring容器

##### getBean

###### 作用

从容器获取指定的bean

###### 参数

String beanName：bean名称

###### 返回值

返回对应bean，没有找到则返回null

###### 重载

Class<T> clazz：指定类型

Class<? extends T> implClazz：指定实现类

##### getBeansWithAnnotation

###### 作用

按类型上的注解，获取bean，可能是多个bean

###### 参数

Class<? extends Annotation> clazz：类注解类类型

###### 返回值

返回bean键值队(bean名,值)，没有找到则返回null

##### getBeansOfType

###### 作用

按指定类类型，获取bean，可能是多个bean

###### 参数

Class<T> clazz：指定的类类型

###### 返回值

返回bean键值队(bean名,值)，没有找到则返回null

### HttpInternalObjectManager

#### 作用

管理http内置对象以及线程唯一变量的管理类，提供直接获取Http内置对象的方法，可以在任何地方，不需要任何参数，降低耦合。该管理类可以获取servletContext、HttpSession、HttpServletRequest request、HttpServletResponse response等对象。

#### 常用方法

##### getServletContext

###### 作用

获取当前项目的ServletContext对象

###### 返回值

当前项目的ServletContext对象

##### getCurrentSession

###### 作用

获取当前的session

###### 返回值

当前的session

##### getCurrentRequest

###### 作用

获取当前的request，严格来说是获取当前线程中的请求对象

###### 返回值

当前的request

##### getCurrentResponse

###### 作用

获取当前的response，严格来说是获取当前线程中的响应对象

###### 返回值

当前的response

##### putThreadVariable

###### 作用

把某些要求线程内唯一的变量给HttpInternalObjectManager管理

###### 参数

String variableName：指定的变量名

Object variableValue：变量值

##### getThreadVariable

###### 作用

根据变量名获取线程内指定的变量值

###### 参数

String variableName：变量名

###### 返回值

变量值

##### removeThreadVariable

###### 作用

移除线程内的变量，就是说HttpInternalObjectManager不再管理它

###### 参数

String variableName：变量名

###### 返回值

被移除的变量值

### UurpAdaptor

#### 作用

这个适配器叫做用户单位角色权限适配器，负责对用户/单位/角色/权限等对象的相关关联操作。

#### 常用方法

##### findUnitTree

###### 作用

通过迭代器的原理，递归找出指定单位Id的所有下属单位(包含下属单位的下属单位……)

###### 参数

String parentUnitId：指定的单位Id

###### 返回值

所有下属单位对象的列表

##### findUnitChildren

###### 作用

找出指定单位Id的下属单位

###### 参数

String parentUnitId：指定的单位Id

###### 返回值

下属单位对象的列表

##### findUnitUsers

###### 作用

找出指定单位Id下属单位的用户

###### 参数

String parentUnitId：指定的单位Id

###### 返回值

下属单位的用户对象的列表

##### findUnitAllUsers

###### 作用

通过迭代器的原理，递归找出指定单位Id的所有的用户(包含下属单位的下属单位的用户……)

###### 参数

String parentUnitId：指定的单位Id

###### 返回值

下属单位的所有用户对象的列表

##### findTopUnit

###### 作用

通过迭代器的原理，递归找出指定单位Id的最顶级单位(就是父单位为null)

###### 参数

String unitId：指定的单位Id

###### 返回值

最顶级单位对象

##### queryUnitTree

###### 作用

跟findUnitTree一样的效果，只是实现方式是通过SQL查询

##### queryUnitChildren

###### 作用

跟findUnitChildren一样的效果，只是实现方式是通过SQL查询

##### queryUnitUsers

###### 作用

跟findUnitUsers一样的效果，只是实现方式是通过SQL查询

##### queryUnitAllUsers

###### 作用

跟findUnitAllUsers一样的效果，只是实现方式是通过SQL查询

## 框架特殊属性以及方法介绍

### 简述

有些比较特殊的方法或者属性，在这里说明一下

### BaseDTO::criteriaLogicExpression

#### 声明

private String criteriaLogicExpression

#### 作用

And/Or表达式。

例如：And(DTO字段名1,Or(And(DTO字段名2,DTO字段名3),DTO字段名4))，service层调用BaseServiceImpl::getPropertyCriterionByDTO时候，根据DTO有无表达式自动处理，BaseServiceImpl::getPropertyCriterionByDTO生成的Criterion查询对象会按criteriaLogicExpression指定好的关系生成。

例子：

or(And(descr[1],Or(code[1],And(whiteUrls,or(code[0],code[2],code[1],AND(descr[2],code[3]))))),descr[0],name[2],name[1],name[3],descr,descr[3])，现在已经支持非常复杂的关系查询。

### BaseDTO::operateRelationMap

#### 声明

private Map<String, OperatorType> operateRelationMap = new HashMap<String, List<OperatorType>>()

#### 作用

操作关系。

例如：字段名1要一个LIKE操作,直接put入("字段名1",Arrays.asList(new OperatorType[]{OperatorType.LIKE}))，在service层调用BaseServiceImpl::getPropertyCriterionByDTO生成Criterion查询对象时候会按LIKE去生成”字段名1”和”字段值1”的操作关系。

### BaseDTO::operateValue

#### 声明

private Map<String, Object> operateValue = new HashMap<String, Object>()

#### 作用

操作值。

例如:指定字段名1要一个值,直接put入("字段名1",val),在service层调用BaseServiceImpl::getPropertyCriterionByDTO生成Criterion查询对象时候会用这个”val”去代替原本属性中的值来生成。

### BaseDTO::defCriterion

#### 声明

private Criterion defCriterion = null

#### 作用

默认的Criterion查询条件。如果defCriterion不为null，那么defCriterion将会取代BaseServiceImpl::getPropertyCriterionByDTO生成Criterion查询对象，作为BaseServiceImpl::getPropertyCriterionByDTO的返回值。

### BaseDTO::excludeCopyValueFieldNames和BaseModel:: excludeCopyValueFieldNames

#### 声明

private List<String> excludeCopyValueFieldNames = this.getDefExcludeCopyValueFieldNames()

#### 作用

复制值时候排除的字段的名称，在调用copyValueTo的时候，在excludeCopyValueFieldNames中的字段名的字段将不会被复制。

### BaseDTO::onlyCopyValueFieldNames和BaseModel:: onlyCopyValueFieldNames

#### 声明

private List<String> onlyCopyValueFieldNames = new ArrayList<String>();

#### 作用

复制值时候只复制的字段的名称，在调用copyValueTo的时候，如果onlyCopyValueFieldNames不为空，只复制onlyCopyValueFieldNames所指定名称的字段。

### BaseDAOImpl::getOrders

#### 声明

protected Order[] getOrders()

#### 作用

获得Dao对应的默认排序对象，可以覆盖这个方法实现自己的排序方式。

#### 返回值

排序对象数组

### BaseServiceImpl::getPropertyCriterionByDTO

#### 声明

protected final List<Criterion> getPropertyCriterionByDTO(BaseDTO<? extends Serializable> dto)

#### 作用

根据指定DTO，得到以DTO 为基础的Criterion查询条件对象。

#### 参数

BaseDTO<? extends Serializable> dto：指定的dto对象

#### 返回值

返回Criterion查询对象列表

### BaseServiceImpl::processCriteriaExpression

#### 声明

private Criterion processCriteriaExpression(Map<String, Criterion> criterionMap, String expression)

#### 作用

用来处理查询对象复杂的And/Or关系的函数，通过DTO指定的And/Or表达式来计算，具体的And/Or表达式是取自DTO的criteriaLogicExpression属性。processCriteriaExpression在BaseServiceImpl::getPropertyCriterionByDTO内部被调用，处理复杂的And/Or关系。

#### 参数

Map<String, Criterion> criterionMap：

查询条件对象map，作为每个字段的基本查询条件对象。其实就是已经生成的Criterion查询对象

String expression：

And/Or表达式(例如:And(DTO字段名1,Or(And(DTO字段名2,DTO字段名3),DTO字段名4)))，如果表达式不正确将不会得到正确的结果。其实expression是取自DTO的criteriaLogicExpression属性

#### 返回值

返回处理后的查询条件对象，如果处理过处出错直接返回null

### BaseServiceImpl::setEntityCreateDefaultValue

#### 声明

protected void setEntityCreateDefaultValue(POJO entity, DTO dto)

#### 作用

在创建的时候,设置实体的部分公共属性的默认值,比如:创建单位Id、创建日期、创建人Id。

#### 参数

POJO entity：

指定实体

DTO dto：

对应的DTO对象

### BaseServiceImpl::setEntityModifiedDefaultValue

#### 声明

protected void setEntityModifiedDefaultValue(POJO entity, DTO dto)

#### 作用

在修改的时候,设置实体的部分公共属性的默认值,比如:修改单位Id、修改日期、修改人Id。

#### 参数

POJO entity：

指定实体

DTO dto：

对应的DTO对象

### BaseServiceImpl::setEntityLogicalDeleteValue

#### 声明

protected void setEntityLogicalDeleteValue(POJO entity)

#### 作用

在逻辑删除的时候,设置实体的逻辑删除标志和删除日期。

#### 参数

POJO entity：

指定实体

### BaseServiceImpl::setEntityLogicalDefaultValue

#### 声明

protected void setEntityLogicalDefaultValue(DTO dto)

#### 作用

在保存或者修改的时候,设置排除逻辑删除的标志位,避免恶意注入。

#### 参数

DTO dto：

对应的DTO对象

### BaseServiceImpl::getLogicalDeleteCriterion

#### 声明

protected Criterion getLogicalDeleteCriterion(DTO dto)

#### 作用

获取逻辑删除的条件对象。

#### 参数

DTO dto：

DTO对象

#### 返回值

返回条件对象

### BaseServiceImpl::getPermissionCriterion

#### 声明

protected Criterion getPermissionCriterion(DTO dto)

#### 作用

获取权限控制的条件对象。

#### 参数

DTO dto：

DTO对象

#### 返回值

返回条件对象

### BaseDTO::modelDTOSearchOnField

#### 声明

private boolean modelDTOSearchOnField = false;

#### 作用

默认情况下，使用ModelDTOSearch注解时候，获取搜索值的时候只在operateValue中查找，当这个标志为true，则在operateValue中找不到值时候会在Field中也进行查找。

### BaseDTO::isExcludedProperty

#### 声明

public boolean isExcludedProperty(String propertyName, Object propertyValue)

#### 作用

在BaseServiceImpl::getPropertyCriterionByDTO调用的时候，凡是使isExcludedProperty返回true的属性和值，将不会被生成Criterion查询条件对象，也就是说isExcludedProperty是用来定义那些属性会被排除的方法。

#### 参数

String propertyName

属性名，在BaseServiceImpl::getPropertyCriterionByDTO调用时候传入，就是DTO对象当前的属性名。

Object propertyValue：

属性值，在BaseServiceImpl::getPropertyCriterionByDTO调用时候传入，就是DTO对象当前的属性值。

#### 返回值

是否排除

### BaseDTO::isDefExcludedProperty

#### 声明

public final boolean isDefExcludedProperty(String propertyName, Object propertyValue)

#### 作用

BaseDTO::isExcludedProperty的作用一样，只不过是对应了默认的排除方式，避免开发人员错误覆盖了，因此这个方法不能被覆盖。其他都是一样的。

### BaseDTO::generateDefOperateRelation

#### 声明

public void generateDefOperateRelation(String requestURL, Map<String, String[]> paramsMap)

#### 作用

生成默认的operateRelationMap的值，此方面之前已经说过在AnnotationArgumentResolver::resolveModelDTOSearch会被调用，以生成默认的操作关系，这些操作关系会影响到后面BaseServiceImpl::getPropertyCriterionByDTO生成的Criterion查询对象。

#### 参数

String requestURL：

请求URL，AnnotationArgumentResolver传入，其目的是可以提供一些判断。

Map<String, String[]> paramsMap：

请求的参数，AnnotationArgumentResolver传入，其目的是可以提供一些判断。

#### 使用说明

LogDTO:: generateDefOperateRelation

@Override

public void generateDefOperateRelation(String requestURL, Map<String, String[]> paramsMap) {

HashMap<String, OperatorType> operateRelationMap = new HashMap<String, OperatorType>();

operateRelationMap.put("title", OperatorType.LIKE);

operateRelationMap.put("loggerType", OperatorType.EQ);

operateRelationMap.put("userId", OperatorType.EQ);

operateRelationMap.put("userName", OperatorType.LIKE);

operateRelationMap.put("ipAddress", OperatorType.LIKE);

operateRelationMap.put("url", OperatorType.LIKE);

operateRelationMap.put("logTime", OperatorType.BETWEEN);

operateRelationMap.put("content", OperatorType.LIKE);

this.setOperateRelation(operateRelationMap);

}

### BaseDTO::generateDefCriterion

#### 声明

public void generateDefCriterion(String requestURL, Map<String, String[]> paramsMap)

#### 作用

生成默认的defCriterion的值，此方面之前已经说过在AnnotationArgumentResolver:: resolveModelDTOCriterion会被调用，以生成默认的Criterion查询对象，默认的Criterion查询对象会代替BaseServiceImpl::getPropertyCriterionByDTO生成的Criterion查询对象，作为BaseServiceImpl::getPropertyCriterionByDTO返回值。

#### 参数

String requestURL：

请求URL，AnnotationArgumentResolver传入，其目的是可以提供一些判断。

Map<String, String[]> paramsMap：

请求的参数，AnnotationArgumentResolver传入，其目的是可以提供一些判断。

### BaseDTO::getDefExcludeCopyValueFieldNames和BaseModel::getDefExcludeCopyValueFieldNames

#### 声明

public List<String> getDefExcludeCopyValueFieldNames()

#### 作用

获取默认的copyValuesTo排除字段的名称，可以覆盖这个方法来来定义自己要排除的字段。

#### 返回值

排除字段的名称

### BaseDTO::copyValuesTo和BaseModel:: copyValuesTo

#### 声明

public void copyValuesTo(Object obj)

#### 作用

根据字段名，将当前对象的字段值复制到obj对象对应的字段,在excludeCopyValueFieldNames指定了的字段将不进行复制；如果指定了onlyCopyValueFieldNames，那么只复制onlyCopyValueFieldNames指定的字段。

#### 参数

Object obj：指定对象

### BaseAction::setToPageContext

#### 声明

protected void setToPageContext(Object... objs)

#### 作用

将指定的objs多个对象的全部变量值写入到页面内，通过request::setAttribute实现。

#### 参数

Object... objs：指定的多个对象

## 特殊功能开发

### 日志功能

#### 简述

由于很多系统都需要记录用户的操作，所以本框架也不例外，提供了日志功能，日志功能针对具体的方法，通过Spring AOP实现。现在日志功能暂时提供了两个类型的日志，一种是登录日志，记录登录用户，登录IP，登录备注等信息；另外一种是普通日志，记录方法的方法名、参数、返回值、调用对象等信息。如果需要另外的类型，需要自己开发，现在来说已经够用。

#### 开发过程

由于是Spring AOP开发的，所以增加某个方法或者类型的日志功能很简单，只需要在指定的方法或者指定的类型增加注解@Logger就可以了，当然可以指定日志类型@Logger(type = LoggerType.LOGIN)，默认情况下是LoggerType.NORMAL普通日志类型。

如果需要开发新的日志类型，就先要增加LoggerType的类型，LoggerType是枚举类，然后在LoggerAspect中增加saveXXXXXLog方法，实现相关类型的日志功能，XXXXX一般跟LoggerType指定的类型名相同，然后再把saveXXXXXLog加入到LoggerAspect::atAnnotationAroundAdvice中如下的位置：

try {// 捕获保存日志出错,避免整个线程崩溃

LoggerType loggerType = logger.type();

if (LoggerType.LOGIN.equals(loggerType)) {

this.saveLoginLog(pjp, throwsEx, user);

} else if (LoggerType.NORMAL.equals(loggerType)) {

this.saveNormalLog(pjp, retVal, throwsEx, user);

} else if (LoggerType.XXXXX.equals(loggerType)) {

this.saveXXXXXLog(pjp, retVal, throwsEx, user);

} else {

// 其他

}

} catch (Exception e) {

e.printStackTrace();

}

开发完相应类型的日志功能后，只需要在需要指定类型日志功能的方法或者类上加上注解@Logger(type = LoggerType.XXXXX)既可生效。日志功能的开发都是这样，相对比较方便。

#### 注意事项

值得一提，以为日志功能是一个比较特殊的功能，无论是方法执行是否成功都必须记录起来，所以日志功能的数据库事务不能跟当前执行的模块功能共用，不然的话当模块功能执行出错，数据库回滚，记录的日志也会跟着回滚。虽然框架已经提供了相关控制的实现，但是不排除后续的开发有新的需求，所以大家开发的时候也要注意这个问题。

### 搜索功能

#### 描述

每个系统，搜索功能都必不可小的，为了方便每个模块的搜索功能的开发，已经比较系统整理了代码，可以比较高效的开发每个模块的搜索功能。其实搜索功能跟之前的BaseDTO::generateDefOperateRelation、BaseDTO::generateDefCriterion、BaseDTO::isExcludedProperty方法都有一定的联系，因为这些方法会影响到BaseServiceImpl::getPropertyCriterionByDTO生成的Criterion查询对象，而查询也是主要通过Criterion查询对象来体现的。

其实搜索功能对不同的复杂度的搜索有三种不同的实现：

第一种是最普通的，也是默认已经实现好的，就是直接依赖SpringMVC的数据绑定，直接获取页面数据，生成DTO对象，不过由这个DTO对象生成的Criterion查询对象，默认所有操作属性都是“相等”关系。这种情况下的搜索功能其实作用不大，一般都排不上用场。

第二种实现方式是最常用的实现方式，它使用@ModelDTOSearch注解，需要覆盖指定的方法BaseDTO::generateDefOperateRelation，来指定属性和值的操作关系，同时页面也需要处理一下。不过这种情况下的查询功能在遇到非常复杂的查询时候也是排不上用场的，比如多级的级联查询、子查询、复合查询等，这种情况下只能用第三种实现方式了。

第三种实现方式最灵活，但是工作量会大一点，因为完全你需要自己定义好整个BaseDTO::defCriterion查询对象，然后这个查询对象会代替BaseServiceImpl::getPropertyCriterionByDTO生成的Criterion查询对象，以完成查询。第三种实现方式需要使用@ModelDTOCriterion注解，需要覆盖BaseDTO::generateDefCriterion方法，实现BaseDTO::defCriterion的生成，当然具体怎么控制看开发人员了，因为这个方法有了页面提交的全部数据。

第一、第二种方式可以通过BaseDTO::criteriaLogicExpression属性来定义好表达式，控制属性之间的关系。例如：criteriaLogicExpression=”And(DTO字段名1,Or(And(DTO字段名2,DTO字段名3),DTO字段名4))”。由于第一种方式已经默认实现了，下面只说明第二第三种方式的开发过程。

#### 开发过程

##### 第二种实现方式

###### 过程描述

在已经写好基本功能的情况下，搜索功能的开发需要在DTO层、展现层Action、JSP页面三处做修改。下面以日志管理模块做例子。

###### 展示层

展示层需要在搜索的请求方法的DTO参数在加上@ModelDTOSearch注解。以实现调用BaseDTO::generateDefOperateRelation方法，生成好DTO属性与值之间的操作关系，并且值会重新绑定到BaseDTO::operateValue的键值队中。

代码如下：

@RequestMapping("/logger/list.do")

public String list(HttpServletRequest request, @ModelDTOSearch LogDTO dto, ModelMap model, Integer pageNumber)

###### 数据转换层

DTO层需要覆盖BaseDTO::generateDefOperateRelation方法，定义好属性和值之间的操作关系。

代码如下：

@Override

public void generateDefOperateRelation(String requestURL, Map<String, String[]> paramsMap) {

HashMap<String, OperatorType> operateRelationMap = new HashMap<String, OperatorType>();

operateRelationMap.put("title", OperatorType.LIKE);

operateRelationMap.put("loggerType", OperatorType.EQ);

operateRelationMap.put("userId", OperatorType.EQ);

operateRelationMap.put("userName", OperatorType.LIKE);

operateRelationMap.put("ipAddress", OperatorType.LIKE);

operateRelationMap.put("url", OperatorType.LIKE);

operateRelationMap.put("logTime", OperatorType.BETWEEN);

operateRelationMap.put("content", OperatorType.LIKE);

this.setOperateRelation(operateRelationMap);

}

###### 前台页面

页面需要指定格式的属性名，因为前台页面的提交过程和翻页过程已经统一收到框架Javascript的控制。一般需要一个搜索表单来提交这些搜索数据，搜索后的值需要重新写回页面，以方便搜索翻页的控制，所以需要一个隐藏的表单来保存写回页面的搜索的值，这个隐藏的表单规定命名为id=”backupForm”；搜索表单的表单规定命名为id=”searchForm”。

前台页面使用了搜索框的形式，用post表单的方式提交搜索，以及带翻页的搜索控制。还有需要在表单html属性action指定好查询所请求的url。

代码如下：

<!--搜索框 START-->

<div class="modal hide fade" id="searchDialog">

<form action="${base}/logger/list.do?1=1" method="post" id="searchForm" class="form-horizontal">

<div class="modal-header">

<a class="close" data-dismiss="modal">×</a>

<h3 id="tableTitle">搜索信息</h3>

</div>

<div class="modal-body">

<table>

<tbody>

<tr>

<td>

<div class="form-group">

<label class="sr-only" for="title" style="width: 280px;">主题</label> <input type="text" class="form-control" name="title" placeholder="*主题*" value="${fn:escapeXml(title)}" />

</div>

</td>

<td>

<div class="form-group">

<label class="sr-only" for="loggerType">日志类型</label> <select class="form-control" name="loggerType" style="width: 206px;" value="${fn:escapeXml(loggerType)}" placeholder="*日志类型*">

<option value=""></option>

<option value="LOGIN" <c:if test="${fn:escapeXml(loggerType) == 'LOGIN'}">selected="selected"</c:if>>登录日志</option>

<option value="NORMAL" <c:if test="${fn:escapeXml(loggerType) == 'NORMAL'}">selected="selected"</c:if>>普通日志</option>

</select>

</div>

</td>

</tr>

<tr>

<td>

<div class="form-group">

<label class="sr-only" for="userName">用户名称</label> <input type="text" class="form-control" name="userName" placeholder="*用户名称*" value="${fn:escapeXml(userName)}" />

</div>

</td>

<td>

<div class="form-group">

<label class="sr-only" for="ipAddress">IP地址</label> <input type="text" class="form-control" name="ipAddress" placeholder="IP*地址*" value="${fn:escapeXml(ipAddress)}" />

</div>

</td>

</tr>

<tr>

<td>

<div class="form-group">

<label class="sr-only" for="url">访问地址</label> <input type="text" class="form-control" name="url" placeholder="*访问地址*" value="${fn:escapeXml(url)}" />

</div>

</td>

<td></td>

</tr>

<tr>

<td colspan="2">

<div class="form-group">

<label class="sr-only" for="logTime">记录时间</label>&nbsp;&nbsp;&nbsp;从&nbsp;&nbsp;&nbsp;<input type="text" name="logTime" value="${fn:escapeXml(operateValue['logTime'][0])}" placeholder="*记录时间*" onClick="WdatePicker({dateFmt:'yyyy-MM-dd HH:mm:ss'});" />&nbsp;&nbsp;&nbsp;到&nbsp;&nbsp;&nbsp;<input

type="text" name="logTime" value="${fn:escapeXml(operateValue['logTime'][1])}" placeholder="*记录时间*" onClick="WdatePicker({dateFmt:'yyyy-MM-dd HH:mm:ss'});" />

</div>

</td>

</tr>

</tbody>

</table>

</div>

</form>

<form action="${base}/logger/list.do?1=1" method="post" id="backupForm" style="display: none;">

<input type="hidden" name="title" value="${fn:escapeXml(title)}" /> <input type="hidden" name="loggerType" value="${fn:escapeXml(loggerType)}" /> <input type="hidden" name="userName" value="${fn:escapeXml(userName)}" /> <input type="hidden" name="ipAddress"

value="${fn:escapeXml(ipAddress)}" /> <input type="hidden" name="url" value="${fn:escapeXml(url)}" /> <input type="hidden" name="logTime" value="${fn:escapeXml(operateValue['logTime'][0])}" /> <input type="hidden" name="logTime"

value="${fn:escapeXml(operateValue['logTime'][1])}" />

</form>

<div class="modal-footer">

<button class="btn btn-primary" id="searchBtn" data-toggle="modal">搜索</button>

<button class="btn btn-warning" id="clearSearchBtn" data-toggle="modal">清空</button>

<button type="button" class="btn" data-dismiss="modal">关闭</button>

</div>

</div>

<!--搜索框 END-->

##### 第三种实现

###### 过程描述

跟第二种方式的修改地方一样，同样需要需要在DTO层、展现层Action、JSP页面三处做修改。

###### 展示层

展示层需要在搜索的请求方法的DTO参数在加上@ModelDTOCriterion注解。以实现调用BaseDTO::generateDefCriterion方法，生成BaseDTO::defCriterion查询对象。

###### 数据转换层

DTO层需要覆盖BaseDTO::generateDefCriterion方法，具体实现BaseDTO::defCriterion的生成，也是这个方式的核心部分。

###### 前台页面

前台的页面跟第二种方式完全一样。

### 分页功能

#### 简述

分页功能也是框架必不可小的部分，本框架作了整理，实现分页功能只需要一个标签即可。具体的Javascript控制都已经统一编写好，并且能够支持搜索的翻页操作。

#### 开发过程

非常简单，只需要一个标签——yahnc:pagelet，在标签指定好分页请求的url和查询结果返回的对象即可。

代码如下：

<!-- 分页 START -->

<tr>

<td colspan="9"><yahnc:pagelet url="${base}/logger/list.do?1=1" resultSet="${resultSet}" /></td>

</tr>

<!-- 分页 END -->

### 指定内容所属单位功能

#### 简述

由于本框架设计时候指定了一个用户可以属于多个部门，所以在保存信息时候的operatedUnitId(createUnitId和modifiedUnitId)字段，需要指定一个部门，所以就有了这个功能了。本框架作了整理，实现该功能也只需要一个标签即可。具体的Javascript控制都已经统一编写好。

#### 开发过程

也是非常简单，只需要一个标签——yahnc:operatedUnitSelectInfo。一般不用填写任何参数，默认即可。

代码如下：

<yahnc:operatedUnitSelectInfo />

## 框架core包

### 简述

net.yasion.common下面，有一个最神秘的包——core，这个包含这个框架最核心的技术，也是包含了Spring的最核心的技术，包括Spring IOC、Spring AOP、Spring无配置启动方式(在配置文件项目下没有使用这个技术)等等。

### net.yasion.common.core.bean

#### 简述

这个包下面实现都是spring bean的载入、初始化、管理、包装等功能。主要是Spring IOC技术。

#### ApplicationContextListener

##### 作用

这个监听器实现Spring容器的获取，在spring容器启动完成后会触发ContextRefreshedEvent事件，这个事件带有一个事件参数，参数里面包含了启动完的容器，因为可以在此处获取到初始化成功的Spring容器。获取到容器就可以把这些容器交给SpringBeanManager统一管理，所以SpringBeanManager也是在这个ContextRefreshedEvent事件之后才真正有用。换一句话说，ApplicationContextListener初始化了SpringBeanManager。值得一提的是，展现层的所有映射方法的url也是在这里加载的，已提供给后面的权限控制使用。

#### SpringBeanManager

##### 作用

它是Spring Bean 管理类，可以在spring容器中获取指定bean。并且提供当前项目所有Spring容器的访问。

### net.yasion.common.core.cache

#### 简述

提供了框架对缓存控制管理的功能

#### WebEhCacheRegionFactory

##### 作用

用于Web的WebEhCacheRegionFactory。可以基于Web根目录指定diskStore路径

### net.yasion.common.core.http

#### 简述

这个包下面提供了Http内置对象的载入、初始化、管理。主要是通过SpringMVC的拦截器、ServletContextAttributeListener监听器、HttpSessionListener监听器来实现相关功能的。

#### HttpInternalObjectInterceptor

##### 作用

SpringMVC的拦截器，该拦截器为Http内置对象管理器载入HttpServletRequest、HttpServletResponse、HttpSession对象。SpringMVC在接受请求时候，会调用这个拦截器，传入请求对象和响应对象，所以这个是最好的时机，通过HttpInternalObjectInterceptor::preHandle时候为管理器载入这些对象。

#### ServletListener

##### 作用

Servlet的监听器，该监听器为Http内置对象管理器载入ServletContext。该监听器在ServletContext初始化的时候或者ServletContext操作属性的时候，都会被调用。传入事件对象，可以从事件对象内获取ServletContext。管理器就是ServletListener::contextInitialized时候载入ServletContext对象，ServletListener::contextDestroyed时候移除ServletContext对象。

#### SessionListener

##### 作用

Session监听器，该监听器为Http内置对象管理器载入HttpSession。其实在HttpInternalObjectInterceptor时候已经载入了HttpSession，但是由于某些原因，HttpInternalObjectInterceptor可能获取到的HttpSession不正确，所以为了确保HttpSession的正确性，必须设置Session监听器。该监听器在HttpSession创建或者销毁的时候都会被调用，所以也是载入HttpSession的好地方。管理器就是在SessionListener::sessionCreated时候载入HttpSession对象，SessionListener::sessionDestroyed时候移除HttpSession对象。

#### HttpInternalObjectManager

##### 作用

管理http内置对象以及线程唯一变量的管理类，提供直接获取Http内置对象的方法，可以在任何地方，不需要任何参数，降低耦合。该管理类可以获取servletContext、HttpSession、HttpServletRequest request、HttpServletResponse response等对象。

### net.yasion.common.core.initializer（配置文件项目没有这个包）

#### 简述

这个包是无配置文件项目的核心包，控制整个项目的启动过程，实现相关配置的内容。其中，WebInitializer是项目启动的控制中心，类似web.xml，ApplicationContextConfig和SpringMVCConfig是相应的配置文件类，实现类似applicationContext.xml、spring-servlet-config.xml的功能。从而控制整个框架的的启动。具体的无配置文件的原理请参考网上spring资料。

#### WebInitializer

##### 作用

框架启动的地方，WebInitializer::onStartup就是主要的方法，控制整个框架的启动过程，创建相应的监听器和过滤器、载入相应的配置类、创建相关的Servlet，然后整个项目开始服务。实现web.xml的功能。

#### ApplicationContextConfig

##### 作用

配置文件类，实现applicationContext.xml的功能，执行创建数据源、Hibernate session工厂、事务管理类等工作。

#### SpringMVCConfig

##### 作用

配置文件类，实现spring-servlet-config.xml的功能，执行创建请求适配器、请求匹配器、视图解析器等工作。

##### 注意事项

SpringMVCConfig的包扫描注解@ComponentScan使用时候要注意，useDefaultFilters非常重要,不加的话后果严重,不加的时候这个会扫描所有的类,会造成部分功能失效,因为这个是子容器,如果不配置useDefaultFilters,连sessionFactory也有可能会被他再次注入导致延迟加载失效

### net.yasion.common.core.logger

#### 简述

这个包下面是日志的相关功能，通过是Spring AOP技术实现的。net.yasion.common.core.logger包下面包括了日志注解类、日志枚举类、日志切面类。通过枚举类来区分日志的类型，暂时就是登录日志还是普通日志两种；通过注解来标志那些方法/类来区分是否需要日志记录；通过切面类来实现运行时的扫描，将相关日志的操作动态加入指定的对象里面。

#### Logger和NoLogger

##### 作用

这里个类都是注解类，Logger用于标识哪些方法或者类需要日志功能，被Logger标识的类内所有的方法都会有日志功能。NoLogger跟Logger相反，用于标识哪些方法或者类不需要日志功能。如果一个方法或者类同时被Logger和NoLogger标识，那么以NoLogger为主，就是说也是不会进行日志记录的。

#### LoggerAspect

##### 作用

日志的切面类，定义许多关于日志功能的切点和通知，实现运行时日志功能加载到指定切入点。当收到指定的切点收到通知时候执行指定日志的记录，将记录写入到数据库。日志切面类已经实现了普通日志和登录日志的记录，如有新需求，请自行增加相关实现。

#### LoggerType

##### 作用

日志枚举类，指定日志的类型，现在有两种类型。默认是普通类型。

### net.yasion.common.core.uurp

#### 简述

这个包下面是用户/单位/角色/权限的相关功能的关联实现功能，主要负责他们之间的关系控制、查找、设置等。

#### UurpAdaptor和UurpAdaptorImpl

##### 作用

接口和实现类负责对用户/单位/角色/权限等对象的相关关联操作。

## 前台插件

### 简述

框架提供了一些比较常用的前台插件，存放在WebContent/static/plugins目录下，页面默认提供的插件是：bootstrap、select2、jquery、jquery-validation、JSON、uploadify、My97DatePicker。这些默认的引入，定义在WebContent/WEB-INF/pages/common/jsHeader.jsp文件内。剩下的插件还包括chart、ckeditor、excanvas、FlatUI、ueditor，下面简单介绍一下插件。

### AwesomeChartJS

#### 作用

Awesome Chart JS 是一个Javascript生成图表的类库，它利用了 HTML5 的 canvas 标签来创建统计图表。此类库就是为了减轻开发者的工作量，使用它只需书写几行代码便能生成漂亮的图表。

#### 版本

v2.0

#### 官网

<http://cyberpython.github.com/AwesomeChartJS/>

### base64

#### 作用

Awesome Chart JS 是一个Javascript生成图表的类库，它利用了 HTML5 的 canvas 标签来创建统计图表。此类库就是为了减轻开发者的工作量，使用它只需书写几行代码便能生成漂亮的图表。

#### 版本

v2.15

#### 官网

<http://travis-ci.org/dankogai/js-base64>

### jquery

#### 作用

这个非常常用，不多说。

#### 版本

v1.11.0

#### 官网

<http://jquery.com/>

#### 注意事项

JQuery 1.9.1之后版本部分方法变动,比如:live()方法被删除

As of jQuery 1.7, the .live() method is deprecated. Use .on() to attach event handlers.

Users of older versions of jQuery should use .delegate() in preference to .live().

This method provides a means to attach delegated event handlers to the document element of a page,

which simplifies the use of event handlers when content is dynamically added to a page.

See the discussion of direct versus delegated events in the.on() method for more information.

改进后的使用建议：

1. $(selector).live(events, data, handler); // jQuery 1.3+

2. $(document).delegate(selector, events, data, handler); // jQuery 1.4.3+

3. $(document).on(events, selector, data, handler); // jQuery 1.7+

### bootstrap

#### 作用

一套基于Jquery的前台框架，简洁、直观、强悍的前端开发框架，让web开发更迅速、简单。

#### 版本

bootstrap(主版本)：v2.3.2

bootstrap-modalmanager：v2.2.4

bootstrap-modal：v2.2.4

#### 官网

<http://www.bootcss.com/>

<http://v2.bootcss.com/>

#### 注意事项

bootstrap现在区分开两个版本bootstrap2和bootstrap3，请不要搞混乱了，他们是不兼容的。

#### 样例

<http://v2.bootcss.com/getting-started.html>

### chart

#### 作用

一套以bootstrap为基础的框架。Chart是一个简单、面向对象、为设计者和开发者准备的图表绘制工具库。

#### 版本

v0.1.0

#### 官网

<http://www.bootcss.com/p/chart.js/>

#### 样例

<http://www.bootcss.com/p/chart.js/>

#### 文档

<http://www.bootcss.com/p/chart.js/docs/>

### ckeditor

#### 作用

CKEditor一般指FCKEDITOR。FCKeditor是目前最优秀的可见即可得网页编辑器之一，它采用JavaScript编写。具备功能强大、配置容易、跨浏览器、支持多种编程语言、开源等特点。它非常流行，互联网上很容易找到相关技术文档，国内许多WEB项目和大型网站均采用了FCKeditor。

#### 版本

v4.3.3

#### 官网

<http://ckeditor.com/>

#### 样例

<http://ckeditor.com/demo>

### excanvas

#### 作用

一个使不支持canvas标签的IE浏览器支持canvas的插件，由Google编写，相当强大。

#### 版本

v2.0

#### 项目svn地址

由于官方网站被封，网上版本相对比较旧，只能svn上有更新，其svn的地址

<http://explorercanvas.googlecode.com/svn/trunk>（需要翻墙情况下同步）

#### 注意事项

Include the ExplorerCanvas tag in the same directory as your HTML files, and

add the following code to your page, preferably in the <head> tag.

<!--[if IE]><script type="text/javascript" src="excanvas.js"></script><![endif]-->

If you run into trouble, please look at the included example code to see how

to best implement this

### FlatUI

#### 作用

一套以bootstrap为基础的框架。Flat UI是基于Bootstrap做的Metro化改造，由Designmodo提供。Flat UI包含了很多Bootstrap提供的组件，但是外观更加漂亮。在此强烈推荐！

#### 版本

v0.0.3

#### 官网

<http://www.bootcss.com/p/flat-ui/>

#### 样例

<http://www.bootcss.com/p/flat-ui/>

### Flotr2

#### 作用

Flotr2 是一个用于绘制 HTML5 图形和图表的开源 JS 库，是 flotr 的分支，但移除了 Prototype 的依赖以及其他方面很多改进。

#### 版本

v0.1.0

#### 官网

<http://www.humblesoftware.com/flotr2/documentation>

#### 样例

<http://www.humblesoftware.com/flotr2/index>

### jquery-ui

#### 作用

jQuery UI 是以 jQuery 为基础的开源 JavaScript 网页用户界面代码库。包含底层用户交互、动画、特效和可更换主题的可视控件。我们可以直接用它来构建具有很好交互性的web应用程序。所有插件测试能兼容IE 6.0+, Firefox 3+, Safari 3.1+, Opera 9.6+, 和GoogleChrome。

#### 版本

v1.11.3

#### 官网

<http://jqueryui.com/>

### jquery-validation

#### 作用

jQuery Validate 插件为表单提供了强大的验证功能，让客户端表单验证变得更简单，同时提供了大量的定制选项，满足应用程序各种需求。该插件捆绑了一套有用的验证方法，包括 URL 和电子邮件验证，同时提供了一个用来编写用户自定义方法的 API。所有的捆绑方法默认使用英语作为错误信息，且已翻译成其他 37 种语言。

#### 版本

v1.11.1

#### 官网

<http://jqueryvalidation.org/>

#### 样例

<http://www.w3cschool.cc/jquery/jquery-plugin-validate.html>

### JSCharts3

#### 作用

JSCharts 是一款免费的开源JavaScript图表脚本库，支持XML数据格式，可以帮助用户快速创建各类图表，无需你具备相应的技术知识，甚至也不用你操作那些繁琐的专业工具，省时省力。

#### 版本

v3.06

#### 官网

<http://www.jscharts.com/>

### JSON

#### 作用

JSON提供了json的序列化和反序列化方法，可以将一个json对象转换成json字符串，也可以将一个json字符串转换成一个json对象。

#### 版本

未知

#### 官网

<http://www.JSON.org/>

### My97DatePicker

#### 作用

支持静态限制，动态限制，脚本自定义限制，以及无效天和无效日期功能，利用这样功能可以任意定制不能选择的日期，这些日期即使毫无规律，毫无连续性，也可以通过这些功能的组合使用轻松搞定。

#### 版本

v4.72 Release

#### 官网

<http://www.my97.net/>

#### 文档

<http://www.my97.net/dp/demo/index.htm>

### select2

#### 作用

美化下拉框jQuery插件扩展自chosen，它能让丑陋的、很长的select选择框变的更好看、更方便，支持搜索，远程数据集，以及无限滚动的结果。

#### 版本

v3.5.1

#### 官网

<http://ivaynberg.github.io/select2/>

#### 样例

<http://ivaynberg.github.io/select2/>

#### 文档

<http://ivaynberg.github.io/select2/#documentation>

### ueditor

#### 作用

UEditor是由百度web前端研发部开发所见即所得富文本web编辑器，具有轻量，可定制，注重用户体验等特点，开源基于MIT协议，允许自由使用和修改代码。

#### 版本

v1.3.6

#### 官网

<http://ueditor.baidu.com/website/>

#### 文档

<http://fex-team.github.io/ueditor/>

### uploadify

#### 作用

来自国外的一款优秀jQuery插件，主要功能是批量上传文件，此插件在项目中已被广泛之用，但是也发现不少问题，于是决定花点时间搞清这个插件的来龙去脉，其中很多东西是翻译自官网。

#### 版本

Uploadify（主版本）：v3.2.1

SWFObject：v2.2

#### 官网

<http://www.uploadify.com/>

#### 文档

<http://www.uploadify.com/documentation/>

#### 样例

<http://www.uploadify.com/demos/>

### winpop

#### 作用

一个js弹出

### zTree

#### 作用

zTree 是一个依靠 jQuery 实现的多功能 “树插件”。优异的性能、灵活的配置、多种功能的组合是 zTree 最大优点。zTree 是开源免费的软件（MIT 许可证）。如果您对 zTree 感兴趣或者愿意资助 zTree 继续发展下去，可以进行捐助。

#### 版本

v3.5.17

#### 官网

<http://www.ztree.me/v3/main.php#_zTreeInfo>

## 前台Javascript公共方法

### 简述

根据前台的需要已经在众多的Javascript方法中，抽取了最常用的方法作为公共方法，扩展其公用性，可重写特性。具体的方法都已经提取到base.js中，其中凡是main、开头的方法都是公共的初始化方法，作为组件的初始化以及初始化事件绑定，UnBind方法是解绑事件控制，因此只有重要的方法才说明，其他类似方法不做说明。

### resetEditDialog

#### 作用

重置编辑窗口

### mainClearForm

#### 作用

清除编辑窗口的数据

#### 参数

formJQObj：指定的编辑表单的JQ对象

### setAjaxForm

#### 作用

异步获取数据，设置编辑表单值

#### 参数

respData：异步请求的响应数据

textStatus：异步请求的返回状态

jqXHR：异步请求XHR对象

id：当前指定获取数据的id

formJQObj：编辑表单的JQ对象

### setChildrenEntityVal

#### 作用

指定parentNode=”true”的元素，在设置值的时候，需要重写setChildrenEntityVal方法，实现相关子节点设置值，一般作为必须重写的方法使用。

#### 参数

respData：异步请求的响应数据

formJQObj：编辑表单的JQ对象

parentJQObj：父元素的JQ对象

id：当前指定获取数据的id

childrenEntitys：

DTO对象子属性的json对象，通常是JSON数组，一对多的多的一方的数组对象

### escapeCharacter

#### 作用

转义指定的字符

#### 参数

val：指定字符

#### 返回值

转义之后的字符

### getJSON

#### 作用

根据指定的表单JQ对象，生成JSON字符串

#### 参数

formJQObj：指定的表单JQ对象

#### 返回值

生成的JSON字符串

### getData

#### 作用

异步获取数据

#### 参数

url：获取数据的url

formJQObj：编辑表单的JQ对象

id：当前指定获取数据的id

successCallback：成功之后的回调函数

### sendData

#### 作用

异步发送数据

#### 参数

dataObj：数据对象，可以是JSON对象，可以是JQ编辑表单对象

successCallback：成功之后的回调函数

url：接收发送数据的url

dialogId：编辑窗口的Id

errorCallback：失败之后的回调函数

### doBeforeJSON

#### 作用

在getJSON将数据转换成JSON字符串之前调用的方法，一般是事件函数

#### 参数

dataObj：数据对象，可以是JSON对象，可以是JQ编辑表单对象

### setAjaxFormCallback

#### 作用

setAjaxForm方法调用完成之后调用的回调函数

#### 参数

respData：异步请求的响应数据

textStatus：异步请求的返回状态

jqXHR：异步请求XHR对象

id：当前指定获取数据的id

formJQObj：编辑表单的JQ对象

## 前台自定义属性

### 简述

由于前台页面控制需要，框架针对html元素定义了几个自定义属性，结合Javascript(base.js主要实现)实现相关的控制。

### keepReadOnly

#### 作用

在Javascript调用方法重置表单的时候，这个属性为true的元素将被保持readOnly状态。

#### 作用域

任何元素

#### Javascript处理方法

base.js::resetEditDialog

#### 可选值

true/false

#### 默认值

false

### parentNode

#### 作用

指明当前元素是不是父元素，如果为父元素，则js处理时候会针对性的对其子元素进行相应处理。

比如：

Javascript调用方法清空表单时候把子元素直接删除，而不是清除值。

Javascript调用方法设置值的时候会自动调用setChildrenEntityVal方法(真实使用时候一定要重写这个方法，实现子元素的生成，因为框架无法确定子元素怎么生成)，设置/生成子元素。

Javascript调用方法获取值生成JSON的时候，根据是否父元素，生成时候数组格式的子元素JSON。

#### 作用域

任何元素

#### Javascript处理方法

base.js::mainClearForm、base.js::setAjaxForm、base.js::getJSON

#### 可选值

true/false

#### 默认值

false

#### 必须附加指定属性

childrenEntityName：指定了parentNode的元素必须指定这个属性，这个属性的值必须与后台指定的DTO类的子属性成员的名称一致，否则DTO将获取不了这些值。通常是一对多时候才使用parentNode属性的。问卷模块就是使用了这个属性，可以参考。

### children

#### 作用

当指定了parentNode为true时候，children元素将会自动配合生成，所以无需手动添加。

#### 作用域

任何元素

#### Javascript处理方法

base.js::mainClearForm、base.js::setAjaxForm、base.js::getJSON

#### 可选值

true/false

#### 默认值

false

### clear

#### 作用

由于框架的Javascript清除数据的方法，默认只清楚非隐藏的元素的值，所以如果有需要清除隐藏元素的值，可以在隐藏元素指定clear=”true”，则这个隐藏元素的值也会被清除。

#### 作用域

任何有value属性的元素

#### Javascript处理方法

base.js::mainClearForm

#### 可选值

true/false

#### 默认值

非隐藏元素：true，隐藏元素：false

### default

#### 作用

select非动态的默认值处理，在Javascript调用方法清除数据的时候，当select元素default=”true”，那么会使用该select元素被标志了default属性的options作为默认值(option[default])。

#### 作用域

select元素(只能使用true/false值)、option元素(值通过不填)

#### Javascript处理方法

base.js::mainClearForm

#### 可选值

true/false/不填

#### 默认值

select元素：false，option元素：undefined

### select2

#### 作用

标志select元素是否启用了select2功能

#### 作用域

select元素

#### Javascript处理方法

base.js::mainClearForm、base.js::setAjaxForm、base.js::getJSON

#### 可选值

true/false

#### 默认值

false

### def-value

#### 作用

select动态默认值处理，当select/input元素使用了select2功能，并且使用了异步加载数据时候，通过def-value指定其默认值(因为是异步加载不能通过option[default]指定默认值了)，然后通过select2(“val”,值)方法设置。

#### 作用域

select元素、input元素

#### Javascript处理方法

base.js::mainClearForm

#### 可选值

任何值

#### 默认值

undefined

### jsonArray

#### 作用

select动态默认值处理，当select/input元素使用了select2功能，并且使用了异步加载数据时候，指定了select2是不是多选。在Javascript调用方法清除值、生成JSON时候常用，如果是多选的select2不指定这个属性为true，select2设置值可能失败，因为多选时候select2需要设置一个数组，而不是一个字符串；获取值的时候也是一样。

#### 作用域

select元素、input元素

#### Javascript处理方法

base.js::mainClearForm、base.js::setAjaxForm、base.js::getJSON

#### 可选值

true/false

#### 默认值

false

### ueditor

#### 作用

指定该元素是否使用了UEditor，因为使用了UEditor的元素获取值或者设置值的时候需要使用指定的API。

#### 作用域

一般都是textarea元素

#### Javascript处理方法

base.js::mainClearForm、base.js::setAjaxForm

#### 可选值

true/false

#### 默认值

false

### set

#### 作用

Javascript调用方法设置值的时候，如果set指定为false时候则不设值。

#### 作用域

任何元素

#### Javascript处理方法

base.js::setAjaxForm

#### 可选值

true/false

#### 默认值

true

### base64

#### 作用

Javascript调用方法设置值的时候，如果base64指定为true时候，则对这个值进行Base64编码。

#### 作用域

任何元素

#### Javascript处理方法

base.js::setAjaxForm、base.js::getJSON

#### 可选值

true/false

#### 默认值

false

### notSelectDefault

#### 作用

Javascript调用方法设置值的时候，如果notSelectDefault指定为true时候，则在select没有找到对应的设置值的时候，使用option[default]指定的值作为默认值。

#### 作用域

select元素

#### Javascript处理方法

base.js::setAjaxForm

#### 可选值

true/false

#### 默认值

false

### escape

#### 作用

Javascript调用方法获取值的时候，如果escape指定为false时候，则不对值进行转义，escape默认值是true，也就是说默认情况都是对值进行转义，避免数据出错。

#### 作用域

任何元素

#### Javascript处理方法

base.js:: getJSON

#### 可选值

true/false

#### 默认值

true

## 前台动态风格

### 简述

在这个版本已经支持网站的动态风格控制，通过common.properties配置文件的frameworkStyle、commonStyle、customStyle、frontendStyle几个配置控制，默认情况下是使用“defualt”风格。这些配置指定在展现层跳转页面的时候，自动加入相关的风格路径，就是说

比如：跳转到common/logger/list.jsp

实际上跳转到 common/default/logger/list.jsp

因此，在安排前台jsp页面位置的时候，需要加入多一层的风格目录；在common的页面目录已经加入default风格目录，以后开发的新系统也要自己相应的加入相关的风格目录。

例子：/pages/common 更改为

/pages/common/default

也因此，Action层控制返回时候也不能直接返回String或者ModelAndView，需要通过BaseReturnPageProcessor来控制。BaseReturnPageProcessor也分成四种，CommonReturnPageProcessor、CustomReturnPageProcessor、FrameworkReturnPageProcessor、FrontendReturnPageProcessor。

### BaseReturnPageProcessor

#### 作用

作为返回页面处理器的父类，提供返回页面的基本处理，提供returnViewName、returnModelAndView两种返回方式。

returnViewName：

返回页面路径字符串，就是相当于之前的直接返回字符串。

returnModelAndView：

返回ModelAndView对象，里面包括Model(带到前台的数据)和View(指定页面)的两个方面的数据。

### CommonReturnPageProcessor

#### 作用

common使用的返回页面处理器，自动返回到common目录下指定风格目录的页面。

#### 例子

// 返回到common/default/logger/list.jsp

return new CommonReturnPageProcessor("logger/list").returnViewName();

### CustomReturnPageProcessor

#### 作用

自定义项目使用的返回页面处理器，自动返回到自定义项目目录下指定风格目录的页面。一般情况下，在自己的新项目下面都是使用这个处理器的，第一个参数指定具体模块的页面路径，第二个参数指定新项目的路径。

#### 例子

// 返回到tlep/default/attachment/list.jsp

return new CustomReturnPageProcessor("attachment/list", "tlep").returnViewName();

### FrameworkReturnPageProcessor

#### 作用

主体框架使用的返回页面处理器，自动返回到主体框架目录下指定风格目录的页面。

#### 例子

// 返回到framework/default/main.jsp

return new FrameworkReturnPageProcessor("main").returnViewName();

### FrontendReturnPageProcessor

#### 作用

前台框架使用的返回页面处理器，自动返回到前台框架目录下指定风格目录的页面。

#### 例子

// 返回到frontend/default/index/questionnaireList.jsp

return new FrontendReturnPageProcessor("index/questionnaireList").returnViewName();

### 注意事项

因为使用了动态的前台风格，所以在指定项目的页面目录时候，需要多加一层风格目录。

例子：

旧版

pages/demo/

pages/tlep/logger

新版

pages/demo/default/

pages/tlep/default/logger

## 框架基本配置文件

### 简述

本框架使用很多配置文件，包括Spring的配置文件和属性文件，这里详细说明一下。

### Spring配置文件（无配置文件项目没有这些文件）

#### 简述

主要是一些spring的启动配置文件，还有一些就是springMVC的配置文件。无配置文件项目不需要这些文件。他们分别就是两个文件WebContent/WEB-INF/config/spring/applicationContext.xml、WebContent/WEB-INF/config/spring/spring-servlet-config.xml。

#### applicationContext.xml

##### 作用

Spring的主要配置文件、主容器的配置文件，影响整个框架。重要部分包括了数据源的配置、Hibernate Session工厂的配置、事务管理器的配置、主容器Spring AOP的配置，次要一点的配置还包括了JDBC属性文件位置的指定，类扫描路径的配置。

##### 注意事项

凡是“${ }”包围着的配置，都是动态的配置，读取自JDBC属性文件的值。通过org.springframework.beans.factory.config.PropertyPlaceholderConfigurer类实现相关读取。通过PropertyPlaceholderConfigurer:: location指定JDBC属性文件的位置。

类扫描路径时候请注意，对web包中的所有类进行扫描，以完成Bean创建和自动依赖注入的功能;不扫描带有@Controller注解的类。因为这些类已经随容器启动时，在spring-servlet-config.xml中扫描过一遍了

#### spring-servlet-config.xml

##### 作用

SpringMVC的配置文件，Spring子容器的配置文件，子容器会自动继承父容器，就是applicationContext.xml配置的容器。主要就是SpringMVC的类扫描路径的配置、请求适配器的配置、请求映射器的配置、视图解析器的配置、子容器Spring AOP的配置。

##### 注意事项

类扫描路径时候请注意，对web包中的所有类进行扫描，以完成Bean创建和自动依赖注入的功能;扫描业务组件，让spring不扫描带有@Service @Repository等注解的类(留在applicationContext.xml中扫描@Service @Repository等注解的类)，防止事务失效。

### 属性文件

#### 简述

主要用于可配置性的属性，优点是不需要更改代码，JDBC属性和控制性的属性可以定义在属性文件中。主框架里面四个属性文件WebContent/WEB-INF/config/properties/constant/common.properties、WebContent/WEB-INF/config/properties/constant/config.properties、WebContent/WEB-INF/config/properties/constant/contentType.properties、WebContent/WEB-INF/config/properties/jdbc/mysql.properties的作用。

#### common.properties

##### 作用

跟net.yasion.common.constant.CommonConstants类绑定，定义了框架内使用到的属性。

##### 属性

实体包路径：modelPaeckageName = net.yasion.tsdp.model

分页大小：pageSize = 15

登录拦截器的排除请求url：loginExcludeUrls = /login.do, /logout.do, /index.do

权限拦截器重定向url：permissionRedirectUrl =

权限拦截器的排除请求url：

permissionExcludeUrls = /login.do, /logout.do, /index.do

权限拦截器默认白名单：

permissionDefWhiteUrls = /user/info.do, /user/infoUpdate.do, /unit/ajaxLoadUnit.do, /unit/searchUnitById.do

权限拦截器默认黑名单：

permissionDefBlackUrls = /permission/[A-Za-z0-9]\*.do, /role/[A-Za-z0-9]\*.do, /user/[A-Za-z0-9]\*.do, /unit/[A-Za-z0-9]\*.do, /rolePermission/[A-Za-z0-9]\*.do, /userRole/[A-Za-z0-9]\*.do, /logger/[A-Za-z0-9]\*.do

管理员角色编码：adminRoleCode = ADMIN , SUPERADMIN

#### config.properties

##### 作用

用于测试

#### contentType.properties

##### 作用

跟net.yasion.common.constant.ContentTypeConstants类绑定，用于定义所有http请求的contentType。

##### 属性

contentType

#### mysql.properties

##### 作用

JDBC的基本属性，包括数据源的属性、c3p0连接池的属性、Hibernate的属性，跟applicationContext.xml配置文件或者net.yasion.common.core.initializer.config. ApplicationContextConfig类配合使用。

## 框架高级支持功能

### 简述

框架高级支持功能，主要就是最新支持的缓存功能、DTO复杂查询功能，日志功能，一般情况下，这些功能只需要按默认使用就可以，但是有些情况可以通过配置开启这些高级功能。

### Hibernate缓存功能

#### 作用

Hibernate缓存功能，可以通过配置开启二级缓存、查询缓存，提高性能和效率。默认情况是开启二级缓存和查询缓存的。

#### 简述

##### 一级缓存

###### 简述

Session级别的缓存（又称作事务缓存），Hibernate内置的，不能卸除。缓存只能被当前Session对象访问。缓存的生命周期依赖于Session的生命周期，当Session被关闭后，缓存也就结束生命周期。

###### 数据放入缓存时间点

1. save()。当session对象调用save()方法保存一个对象后，该对象会被放入到session的缓存中。
2. get()和load()。当session对象调用get()或load()方法从数据库取出一个对象后，该对象也会被放入到session的缓存中。
3. 使用HQL和QBC等从数据库中查询数据。

###### 原理

在同一个Session里面，第一次调用get()方法， Hibernate先检索缓存中是否有该查找对象，发现没有，Hibernate发送SELECT语句到数据库中取出相应的对象，然后将该对象放入缓存中，以便下次使用，第二次调用get()方法，Hibernate先检索缓存中是否有该查找对象，发现正好有该查找对象，就从缓存中取出来，不再去数据库中检索，没有再次发送select语句。

###### 清除缓存

1. evit()将指定的持久化对象从缓存中清除，释放对象所占用的内存资源，指定对象从持久化状态变为脱管状态，从而成为游离对象。
2. clear()将缓存中的所有持久化对象清除，释放其占用的内存资源。
3. 其他缓存操作
4. contains()判断指定的对象是否存在于缓存中。
5. flush()刷新缓存区的内容，使之与数据库数据保持同步。

##### 二级缓存

###### 简述

当我们重启一个Session，第二次调用load或者get方法检索同一个对象的时候会重新查找数据库，会发select语句信息。

1. 原因

一个session不能取另一个session中的缓存。

1. 性能上的问题

假如是多线程同时去取Category这个对象，load一个对象，这个对像本来可以放到内存中的，可是由于是多线程，是分布在不同的session当中的，所以每次都要从数据库中取，这样会带来查询性能较低的问题。

1. 解决方案

使用二级缓存。

1. 什么是二级缓存

SessionFactory级别的缓存，可以跨越Session存在，可以被多个Session所共享。

###### 适合情况

1. 经常被访问
2. 改动不大
3. 数量有限
4. 不是很重要的数据，允许出现偶尔并发的数据

比如：这样的数据非常适合放到二级缓存中的。

用户的权限

用户的数量不大，权限不多，不会经常被改动，经常被访问

组织机构

不会经常被改动，经常被访问

又比如：

改动频繁，类里面对象特别多，BBS好多帖子，这些帖子20000多条，哪些放到缓存中，不能确定。除非你确定有一些经常被访问的，数据量并不大，改动非常少，这样的数据非常适合放到二级缓存中的。

###### 原理

Hibernate如何将数据库中的数据放入到二级缓存中？注意，你可以把缓存看做是一个Map对象，它的Key用于存储对象ID，Value用于存储POJO。首先，当我们使用Hibernate从数据库中查询出数据，获取检索的数据后，Hibernate将检索出来的对象的ID放入缓存中key 中，然后将具体的POJO放入value中，等待下一次再次向数据查询数据时，Hibernate根据你提供的ID先检索一级缓存，若有且配置了二级缓存，则检索二级缓存，如果还没有则才向数据库发送SQL语句，然后将查询出来的对象放入缓存中。

###### 使用二级缓存

所在配置文件：/WEB-INF/config/properties/jdbc/mysql.properties

1. 打开二级缓存

<!-- 使用二级缓存 -->

hibernate.cache.use\_second\_level\_cache=true

1. 配置二级缓存

<!--设置缓存的类型，设置缓存的提供商-->

hibernate.cache.region.factory\_class=net.yasion.common.core.cache.ehcache.facotry.WebEhCacheRegionFactory

<!--提供商的配置文件路径-->

net.sf.ehcache.configurationResourceName=/WEB-INF/config/hibernate/cache/ehcache/ehcache.xml

1. 配置ehcache.xml

<ehcache xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="http://ehcache.org/ehcache.xsd" maxBytesLocalDisk="50G">

<!--缓存到硬盘的路径-->

<diskStore path="/WEB-INF/cache" />

<!--

默认设置

maxElementsInMemory : 在內存中最大緩存的对象数量。

eternal : 缓存的对象是否永远不变。

timeToIdleSeconds ：可以操作对象的时间。

timeToLiveSeconds ：缓存中对象的生命周期，时间到后查询数据会从数据库中读取。

overflowToDisk ：内存满了，是否要缓存到硬盘。

-->

<defaultCache maxElementsInMemory="10000" eternal="false" timeToIdleSeconds="120" timeToLiveSeconds="120" overflowToDisk="true" />

<!--

指定缓存的对象。下面出现的的属性覆盖上面出现的，没出现的继承上面的。

-->

<!--

java文件注解查找cache方法名的策略：如果不指定hbm.xml/java注解中的region="ehcache.xml中的cache name的属性值",则使用name名为com.yasion.common.model.TbUser的cache(即类的全路径名称)。如果不存在与类名匹配的cache名称, 则用 defaultCache。

如果User包含set集合, 则需要另行指定其cache,

例如User包含citySet集合, 则也需要添加配置到ehcache.xml中

-->

<cache name="net.yasion.common.model.TbUser" maxElementsInMemory="10000" eternal="false" timeToLiveSeconds="120" overflowToDisk="true" />

<cache name="javaClassName" maxElementsInMemory="2000" eternal="false" timeToIdleSeconds="120" timeToLiveSeconds="120" overflowToDisk="true" />

<!—

<cache name="net.yasion.common.model.TbUnit" maxElementsInMemory="200"

eternal="false" timeToIdleSeconds="50" timeToLiveSeconds="60"

overflowToDisk="true"></cache>

-->

</ehcache>

1. 被缓存对象中hbm文件配置

在需要被缓存的对象中hbm文件中的<class>标签下添加一个<cache>子标签

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

<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"

"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<!-- Mapping file autogenerated by MyEclipse Persistence Tools -->

<hibernate-mapping>

<class name="net.yasion.common.model.BaseModel" abstract="true">

<cache usage="read-write" />

<id name="id" type="java.lang.String" length="32">

<column name="id" />

<generator class="uuid.hex" />

</id>

<!— 省略部分代码…… -->

</class>

</hibernate-mapping>

存在一对多的关系，想要在在获取一方的时候将关联的多方缓存起来，需要再集合属性下添加<cache>子标签，这里需要将关联的对象的 hbm文件中必须在存在<class>标签下也添加<cache>标签，不然Hibernate只会缓存ID。

<set name="orders" inverse="true" cascade="all" lazy="false" fetch="join">

<cache usage="read-only" />

<key column="customer\_id"></key>

<one-to-many class="com.suxiaolei.hibernate.pojos.Order" />

</set>

##### 查询缓存

###### 简述

在hibernate的使用中，大家多数时间都在讨论一级缓存和二级缓存，而往往忽略了查询缓存。其实hibernate的查询缓存在使用过程中也起着同样重要的作用。hibernate的查询缓存是主要是针对普通属性结果集的缓存， 而对于实体对象的结果集只缓存id。

###### 数据存放特点

1. 针对普通属性结果集的缓存
2. 对是实体对象的结果集,只缓存id
3. 使用查询缓存,需要打开查询缓存,并且在调用list方法之前需要显示的调用query.setCacheable(true);
4. 查询缓存与session无关,即在一个session上做了查询缓存,那么在另一个session中可以取到查询缓存的内容,不再发出SQL
5. 查询缓存只对query.list()操作有效,query.iterate()操作不会使用查询缓存
6. 要注意区别:一级缓存与二级缓存是缓存实体,而查询缓存是缓存普通属性.

###### 原理

在一级缓存,二级缓存和查询缓存都打开的情况下作查询操作时这样的：查询普通属性，会先到查询缓存中取，如果没有，则查询数据库；查询实体，会先到查询缓存中取id，如果有，则根据id到缓存(一级/二级)中取实体，如果缓存中取不到实体，再查询数据库。

和一级/二级缓存不同，查询缓存的生命周期 ，是不确定的，当前关联的表发生改变时，查询缓存的生命周期结束。

###### 使用查询缓存

设置 hibernate.cache.use\_query\_cache=true,即配置查询缓存。

如果需要"查询缓存"，还需要在使用Query和Criteria()时设置其setCacheable(true)属性。

#### 框架二级缓存——Ehcache

##### 简述

框架默认使用的第三方缓存是Ehcache，不过可以通过修改hibernate.cache.region.factory\_class属性指定其他的第三方缓存。

EhCache 是一个纯Java的进程内缓存框架，具有快速、精干等特点，是Hibernate中默认的CacheProvider。

Ehcache是一种广泛使用的开源Java分布式缓存。主要面向通用缓存,Java EE和轻量级容器。它具有内存和磁盘存储，缓存加载器,缓存扩展,缓存异常处理程序,一个gzip缓存servlet过滤器,支持REST和SOAP api等特点。

框架已经对默认regionFactory做了修改，使其可以基于Web根目录指定diskStore目录。

##### 配置使用

默认情况下，Ehcache的配置文件是存放在/WEB-INF/config/hibernate/cache/ehcache/ehcache.xml，当然也是可以通过修改net.sf.ehcache.configurationResourceName属性指定新的路径。

diskStore配置，缓存文件的存放位置，现在增加支持/WEF-INF目录下面的路径，框架默认配置就是/WEB-INF/cache文件夹内。

框架只是使用了Ehcache的默认配置，只对defaultCache做了配置。如果其他项目有需要加入其他的缓存配置。

具体详细的Ehcache配置可以参考: <http://haohaoxuexi.iteye.com/blog/2112170>

值得一提：

1. 现在的Ehcache的diskStore属性支持基于Web根目录指定路径，不过一般只能指定WEB-INF目录下面。
2. hbm.xml文件因为是继承于BaseModel.hbm.xml的，BaseModel已经配置了cache属性，所以默认情况下，所有的类都支持了二级缓存。

#### 框架查询缓存

##### 简述

框架默认开启查询缓存

##### 配置使用

1. 通过指定hibernate.cache.use\_query\_cache属性决定是否开启查询缓存。
2. 通过setCacheable(true/false)指定当前查询是否使用查询缓存，如果使用框架提供的BaseDAOImpl查询方法，默认是使用查询缓存的。不过如果有其他特殊需求，可以自行关掉缓存。

##### Bug

在使用查询缓存的时候，如果是使用原生的SQL，是不能使用查询缓存的，由于Hibernate的bug导致，暂时无法修复，框架也已经停止了原生SQL的查询缓存。

bug记录：

Hibernate Bug :NamedNativeQuerys are not cacheable ; https://hibernate.atlassian.net/browse/HHH-9111

### 基于DTO复杂查询功能

#### 作用

普通情况下，我们的搜索功能基本可以应付大部分的一般的DTO的查询，

例如

查询提交这种情况：code like ‘%Test%’ AND name like ‘%m%’ AND descr == ‘Test’

这种情况下直接通过重写generateDefOperateRelation方法指定操作关系就可以完成了。

即使再复杂一些的情况

例如

(code like ‘%Test%’ OR name like ‘%m%’) AND descr == ‘Test’

也可以通过重写generateDefOperateRelation方法指定操作关系和通过设置criteriaLogicExpression来指定逻辑关系表达式，完美控制这个查询。

这些方法上面都已经介绍过，所以不再详细说明。而现在所说的DTO复杂查询是为了解决上面的方法的不足。

假设情况

(code like ‘%Test%’ OR code IS NULL OR code ==’’) AND name like ‘%m%’ AND descr == ‘Test’ AND (flag IS NULL OR flag == ‘D’ ) AND (flag == ‘D’ OR code like ‘%Test%’)

通过上面的方法是无法解决这种复杂的查询的，因为他不支持多个操作关系的指定，并且取值时候也无法控制，所以要使用这种复杂的DTO查询功能。

#### 具体使用

##### 指定多个操作关系

为了补充之前的查询的不方便，现在经过修改可以支持多个操作关系的指定，具体可以通过重写BaseDTO::generateDefOperateRelation、设置BaseDTO::setOperateRelationMap指定，也保留了原来的BaseDTO::setOperateRelation方式，支持只有一个操作关系的指定。

代码

@Override

public void generateDefOperateRelation(String requestURL, Map<String, String[]> paramsMap) {

HashMap<String, List<OperatorType>> operateRelationMap = new HashMap<String, List<OperatorType>>();

operateRelationMap.put("name", new ArrayList<OperatorType>(Arrays.asList(new OperatorType[]{OperatorType.LIKE})));

operateRelationMap.put("code", new ArrayList<OperatorType>(Arrays.asList(new OperatorType[]{OperatorType.LIKE,OperatorType.ISNULL, OperatorType.EQ})));

operateRelationMap.put("descr", new ArrayList<OperatorType>(Arrays.asList(new OperatorType[]{OperatorType.EQ})));

operateRelationMap.put("flag", new ArrayList<OperatorType>(Arrays.asList(new OperatorType[]{OperatorType.EQ,OperatorType.ISNULL})));

this.setOperateRelationMap(operateRelationMap);

}

通过上面的代码就可以指定多个操作关系。只是注意的是，在页面传值的时候，有多少个操作关系，就要指定对应字段的多少个值(上面代码在传值时候需要指定code三个值，flag两个值，descr、name一个值)，这些操作关系会一对一那样自动的去配对对应的值，不需要开发人员操心其配对值的细节。

传值的过程，比如

code=Test&code=&code=#ISNULL#&name=m&descr=Test&flag=#ISNULL#&flag=D

上面的值可以自动被code的LIKE、EQ、ISNULL；name的LIKE；descr的EQ；flag的EQ、ISNULL分别配对到，当然ISNULL并不是通过“#ISNULL#”配对到的，只是这样写用来表明可以通过传值来标志ISNULL，然后后台手动设置进去而已。

当指定的值的个数跟操作关系的个数对应不上的时候，系统会把值为主导。就是说，如果操作关系比值多了，会直接忽略这个操作关系；如果是值多于操作关系，则多出来的值会用EQ来作为其操作关系。

##### 指定逻辑关系表达式

虽然上面指定了多个操作符，但是因为上面逻辑关系比较复杂，需要指定逻辑关系表达式才能让他正常执行。倘若不指定的话，框架只会将有多个操作关系的字段通过OR来连接，然后再通过AND连接其他字段。

因此，这个表达式才是这个复杂查询的核心，看代码吧。

代码

this.setCriteriaLogicExpression("AND(OR(code[0],code[1],code[2]),name,descr,OR(flag[1],flag[0]),AND(flag[0],code[0]))");

对的，就需要一行代码，主要复杂就是为什么这么写。其实里面的OA/AND是指定的逻辑关系，只是用方法的形式标示，OR/AND方法都可以带多个参数，支持多层逻辑嵌套。其次一方面“code[0]”这些是什么意思，这种写法包含了下标([X])，其实这个意思就是使用code的一个操作关系(下标为0)代替这个位置，相当于code like %XXX%；又由于操作关系会自动去配对对用的值(就是配对第一个传入的值)，所以“code[0]”就是等于code like ‘%Test%’(这里假设code传入的第一个值是‘Test’)；而如果直接写字段名的，比如“name”，就是取name的第一个操作关系LIKE(刚好name只有一个操作关系)；如此类推，我们就可以得出整句sql：(code like ‘%Test%’ OR code IS NULL OR code ==’’) AND name like ‘%m%’ AND descr == ‘Test’ AND (flag IS NULL OR flag == ‘D’ ) AND (flag == ‘D’ OR code like ‘%Test%’)了。

通过上面我们已经知道整个操作的过程了，基本上可以完成操作使用了！

##### 自动拼接剩下操作关系

由于有时候表达式没有全部使用完全部的操作关系，再默认的情况下，系统会把剩下的操作关系以OR的关系把相同字段的操作关系连起来，在用AND把不同字段的操作关系连起来，最后用AND将剩余部分的操作关系连接到按表达式生成的操作关系表达式中。

不过如果你有需要可以通过

BaseDTO::setDealRemaind(false);

来关闭这个功能，这样的话，剩余的操作关系会被忽略掉。

#### 实现原理

通过手动解析表达式字符串，结合操作关系列表，动态控制Criterion的创建，具体请看源代码。

#### 注意事项

1. 因为这个DTO复杂查询是基于之前的一般DTO查询的，所以要先理解一般的DTO查询才能看懂这里！！
2. 在操作关系配对前台传过来的值的时候，因为一般都是一个操作关系对应一个值的，所以一般不用担心；但是比如BETWEEN、ISNULL、ISEMPTY、IN等操作关系时候要注意，BETWEEN会用两个值；ISNULL、ISEMPTY不会使用任何值；IN会使用剩下的所有的值，所以在使用IN的时候，必须把IN放到操作关系列表的最后，否则改组操作关系只有IN以及IN之前的操作关系生效。
3. 关于下标问题，如果想“code”字段那样，有三个操作关系，如果表达式错误写了“code[3]”，处理时候系统会忽略掉不处理。
4. 表达式的正确问题，如果表达式编写错误，系统会直接忽略表达式，用没有表达式的情况来处理，上面已经说了这种默认处理方式了。
5. 由于这个复杂DTO查询功能是基于一般DTO查询功能升级的，所以请在理解了一般DTO查询之后才使用这个功能。

### 公共日志功能

#### 作用

为了方便开发和生产环境调试，加入第三方日志系统，将日志输出到控制台，并且打印到指定的文件。与之前的登陆日志不同，登陆日志那些是框架自身开发的日志系统，通过Spring AOP实现；而这个公共日志是完全依靠第三方的配置文件控制的。框架默认使用Log4j作为默认实现。

#### 框架公共日志实现——Log4j

##### 简述

Log4j是Apache的一个开放源代码项目，通过使用Log4j，我们可以控制日志信息输送的目的地是控制台、文件、GUI组件，甚至是套接口服务器、NT的事件记录器、UNIX Syslog守护进程等；我们也可以控制每一条日志的输出格式；通过定义每一条日志信息的级别，我们能够更加细致地控制日志的生成过程。最令人感兴趣的就是，这些可以通过一个配置文件来灵活地进行配置，而不需要修改应用的代码。

##### 配置使用

Log4j其实已经配置好，可以直接使用。不过一般配置的是Log4j的日志打印的控制，可以通过配置文件控制日志的级别和日志输出的目标。框架的Log4j配置文件位置：/WEB-INF/config/properties/logger/log4j.properties。具体配置介绍如下，

###### 配置文件

Log4J配置文件的基本格式如下：

#配置根Logger

log4j.rootLogger = [ level ] , appenderName1 , appenderName2 ,…

#配置日志信息输出目的地Appender

log4j.appender.appenderName = fully.qualified.name.of.appender.class

log4j.appender.appenderName.option1 = value1

…

log4j.appender.appenderName.optionN = valueN

#配置日志信息的格式（布局）

log4j.appender.appenderName.layout = fully.qualified.name.of.layout.class

log4j.appender.appenderName.layout.option1 = value1

…

log4j.appender.appenderName.layout.optionN = valueN

其中 [level] 是日志输出级别，共有5级：

FATAL 0

ERROR 3

WARN 4

INFO 6

DEBUG 7

Appender 为日志输出目的地，Log4j提供的appender有以下几种：

org.apache.log4j.ConsoleAppender（控制台），

org.apache.log4j.FileAppender（文件），

org.apache.log4j.DailyRollingFileAppender（每天产生一个日志文件），

org.apache.log4j.RollingFileAppender（文件大小到达指定尺寸的时候产生一个新的文件），

org.apache.log4j.WriterAppender（将日志信息以流格式发送到任意指定的地方）

Layout：日志输出格式，Log4j提供的layout有以下几种：

org.apache.log4j.HTMLLayout（以HTML表格形式布局），

org.apache.log4j.PatternLayout（可以灵活地指定布局模式），

org.apache.log4j.SimpleLayout（包含日志信息的级别和信息字符串），

org.apache.log4j.TTCCLayout（包含日志产生的时间、线程、类别等等信息）

打印参数: Log4J采用类似C语言中的printf函数的打印格式格式化日志信息，如下:

-X号: X信息输出时左对齐；

%p: 输出日志信息优先级，即DEBUG，INFO，WARN，ERROR，FATAL,

%d: 输出日志时间点的日期或时间，默认格式为ISO8601，也可以在其后指定格式，比如：%d{yyy MMM dd HH:mm:ss,SSS}，输出类似：2002年10月18日 22：10：28，921

%r: 输出自应用启动到输出该log信息耗费的毫秒数

%c: 输出日志信息所属的类目，通常就是所在类的全名

%t: 输出产生该日志事件的线程名

%l: 输出日志事件的发生位置，相当于%C.%M(%F:%L)的组合,包括类目名、发生的线程，以及在代码中的行数。举例：Testlog4.main (TestLog4.java:10)

%x: 输出和当前线程相关联的NDC(嵌套诊断环境),尤其用到像java servlets这样的多客户多线程的应用中。

%%: 输出一个"%"字符

%F: 输出日志消息产生时所在的文件名称

%L: 输出代码中的行号

%m: 输出代码中指定的消息,产生的日志具体信息

%n: 输出一个回车换行符，Windows平台为"/r/n"，Unix平台为"/n"输出日志信息换行

可以在%与模式字符之间加上修饰符来控制其最小宽度、最大宽度、和文本的对齐方式。如：

1. %20c：指定输出category的名称，最小的宽度是20，如果category的名称小于20的话，默认的情况下右对齐。
2. %-20c:指定输出category的名称，最小的宽度是20，如果category的名称小于20的话，"-"号指定左对齐。
3. %.30c:指定输出category的名称，最大的宽度是30，如果category的名称大于30的话，就会将左边多出的字符截掉，但小于30的话也不会有空格。
4. %20.30c:如果category的名称小于20就补空格，并且右对齐，如果其名称长于30字符，就从左边较远输出的字符截掉。

###### 代码中初始化Logger

1. 在程序中调用BasicConfigurator.configure()方法：给根记录器增加一个ConsoleAppender，输出格式通过PatternLayout设为"%-4r [%t] %-5p %c %x - %m%n"，还有根记录器的默认级别是Level.DEBUG。
2. 配置放在文件里，通过命令行参数传递文件名字，通过PropertyConfigurator.configure(args[x])解析并配置。
3. 配置放在文件里，通过环境变量传递文件名等信息，利用log4j默认的初始化过程解析并配置。
4. 配置放在文件里，通过应用服务器配置传递文件名等信息，利用一个特殊的servlet来完成配置。

###### 为不同的 Appender 设置日志输出级别

当调试系统时，我们往往注意的只是异常级别的日志输出，但是通常所有级别的输出都是放在一个文件里的，如果日志输出的级别是BUG！？那就慢慢去找吧。

这时我们也许会想要是能把异常信息单独输出到一个文件里该多好啊。当然可以，Log4j已经提供了这样的功能，我们只需要在配置中修改Appender的Threshold 就能实现，比如下面的例子：

[配置文件]

### set log levels ###

log4j.rootLogger = debug , stdout , D , E

### 输出到控制台 ###

log4j.appender.stdout = org.apache.log4j.ConsoleAppender

log4j.appender.stdout.Target = System.out

log4j.appender.stdout.layout = org.apache.log4j.PatternLayout

log4j.appender.stdout.layout.ConversionPattern = %d{ABSOLUTE} %5p %c{ 1 }:%L - %m%n

### 输出到日志文件 ###

log4j.appender.D = org.apache.log4j.DailyRollingFileAppender

log4j.appender.D.File = ${webapp.root}/WEB-INF/logs/error.log

log4j.appender.D.Append = true

log4j.appender.D.Threshold = DEBUG ## 输出DEBUG级别以上的日志

log4j.appender.D.layout = org.apache.log4j.PatternLayout

log4j.appender.D.layout.ConversionPattern = %-d{yyyy-MM-dd HH:mm:ss} [ %t:%r ] - [ %p ] %m%n

### 保存异常信息到单独文件 ###

log4j.appender.D = org.apache.log4j.DailyRollingFileAppender

log4j.appender.D.File = ${webapp.root}/WEB-INF/logs/error.log ## 异常日志文件名

log4j.appender.D.Append = true

log4j.appender.D.Threshold = ERROR ## 只输出ERROR级别以上的日志!!!

log4j.appender.D.layout = org.apache.log4j.PatternLayout

log4j.appender.D.layout.ConversionPattern = %-d{yyyy-MM-dd HH:mm:ss} [ %t:%r ] - [ %p ] %m%n

[代码中使用]

public class TestLog4j {

public static void main(String[] args) {

PropertyConfigurator.configure( " /WEB-INF/config/properties/logger/log4j.properties" );

Logger logger = Logger.getLogger(TestLog4j. class );

logger.debug( " debug " );

logger.error( " error " );

}

}

运行一下，看看异常信息是不是保存在了一个单独的文件error.log中

#### 框架实现Appender——ExternallyDailyRollingFileAppender

##### 作用

由于原来的Appender的局限性，重新开发一个方便易用的Appender。原生的Appender缺点有如下，

DailyRollingFileAppender:只能按日期来滚动，不能控制超过指定数量进行删除。造成日志过量。

RollingFileAppender:只能滚动，不能按日期控制。

基于上面的缺点，框架重新开发一个同时支持两者的Appender——ExternallyDailyRollingFileAppender，除了集成上面两者的有点外，另外加入了可以准确控制打印级别的功能，就是一般的Appender如果指定了打印某个日志级别，就会连比他严重的级别的日志也会打印，现在是只打印这个日志级别的日志。个新Appender可以通过属性设置，启用这个准确打印功能。

##### 新功能

###### 准确打印日至级别

1. 作用

控制日志准确打印日志级别。

1. 指定属性

log4j.appender.XXXX.Exactly = true

true:启用功能

false:关闭功能

默认:启用功能

###### 指定日志文件存在最大个数

1. 作用

控制日志文件个数，避免日志文件过多

1. 指定属性

log4j.appender.XXXX.MaxBackupIndex = 30

指定数量即可，如果为0，表示不限制。

默认:0

###### 文件最大容量

1. 作用

控制单个日志文件的大小，避免文件过大造成浏览困难。

1. 指定属性

log4j.appender.XXXX.MaxFileSize = 20MB

指定大小即可，如果为0，表示不限制。

默认:0

###### 日期滚动格式

1. 作用

指定日期滚动的格式，使文件名按指定格式插入日期

1. 指定属性

log4j.appender.debugfile.DatePattern = yyyy-MM-dd

指定Java的日期格式

默认为null

##### 注意事项

当同时指定控制文件大小和按日期滚动时候，生成的日志文件会先按大小划分，然后在按日期划分，生成的文件就会变成：

debug\_2015-03-15\_1.log

debug\_2015-03-15\_2.log

debug\_2015-03-15\_3.log

debug\_2015-03-16\_1.log

debug\_2015-03-16\_2.log

debug\_2015-03-16\_3.log

类似的文件格式。

当只指定控制文件大小，生成的日志文件按大小划分，生成的文件就会变成：

debug\_1.log

debug\_2.log

debug\_3.log

当只指定按日期滚动时候，生成的日志文件会按日期划分，生成的文件就会变成：

debug\_2015-03-15.log

debug\_2015-03-16.log

debug\_2015-03-17.log

另外，DatePattern属性与其他的Appender的用法不同，只需要指定格式即可，不用指定怎么连接的细节。File属性支持直接/的形式指定Web Context根目录。

#### 框架日志配置文件

#All Startup

log4j.rootLogger = DEBUG, stdout, debugfile, errorfile, warnfile, infofile, fatalfile

#Defualt

#log4j.rootLogger = INFO, stdout, errorfile, warnfile, infofile

#All Not Startup

#log4j.rootLogger =

log4j.appender.stdout = org.apache.log4j.ConsoleAppender

log4j.appender.stdout.Threshold = INFO

log4j.appender.stdout.encoding = UTF-8

log4j.appender.stdout.Target = System.out

log4j.appender.stdout.layout = org.apache.log4j.PatternLayout

log4j.appender.stdout.layout.ConversionPattern = [%d{yyyy-MM-dd HH:mm:ss SSS}] %-5p [%c(line:%L):(%t) %x] %m (consume:%-4r) %n

log4j.appender.debugfile = net.yasion.common.core.logger.log4j.appender.ExternallyDailyRollingFileAppender

log4j.appender.debugfile.DatePattern = yyyy-MM-dd

log4j.appender.debugfile.Exactly = true

log4j.appender.debugfile.MaxBackupIndex = 30

log4j.appender.debugfile.MaxFileSize = 20MB

log4j.appender.debugfile.Threshold = DEBUG

log4j.appender.debugfile.encoding = UTF-8

log4j.appender.debugfile.File = /WEB-INF/logs/debug.log

log4j.appender.debugfile.Append = true

log4j.appender.debugfile.ImmediateFlush = true

log4j.appender.debugfile.layout = org.apache.log4j.PatternLayout

log4j.appender.debugfile.layout.ConversionPattern = [%d{yyyy-MM-dd HH:mm:ss SSS}] %-5p [%c(line:%L):(%t) %x] %m (consume:%-4r) %n

log4j.appender.errorfile = net.yasion.common.core.logger.log4j.appender.ExternallyDailyRollingFileAppender

log4j.appender.errorfile.DatePattern = yyyy-MM-dd

log4j.appender.errorfile.Exactly = true

log4j.appender.errorfile.MaxBackupIndex = 30

log4j.appender.errorfile.MaxFileSize = 20MB

log4j.appender.errorfile.Threshold = ERROR

log4j.appender.errorfile.encoding = UTF-8

log4j.appender.errorfile.File = /WEB-INF/logs/error.log

log4j.appender.errorfile.Append = true

log4j.appender.errorfile.ImmediateFlush = true

log4j.appender.errorfile.layout = org.apache.log4j.PatternLayout

log4j.appender.errorfile.layout.ConversionPattern = [%d{yyyy-MM-dd HH:mm:ss SSS}] %-5p [%c(line:%L):(%t) %x] %m (consume:%-4r) %n

log4j.appender.warnfile = net.yasion.common.core.logger.log4j.appender.ExternallyDailyRollingFileAppender

log4j.appender.warnfile.DatePattern = yyyy-MM-dd

log4j.appender.warnfile.Exactly = true

log4j.appender.warnfile.MaxBackupIndex = 30

log4j.appender.warnfile.MaxFileSize = 20MB

log4j.appender.warnfile.Threshold = WARN

log4j.appender.warnfile.encoding = UTF-8

log4j.appender.warnfile.File = /WEB-INF/logs/warn.log

log4j.appender.warnfile.Append = true

log4j.appender.warnfile.ImmediateFlush = true

log4j.appender.warnfile.layout = org.apache.log4j.PatternLayout

log4j.appender.warnfile.layout.ConversionPattern = [%d{yyyy-MM-dd HH:mm:ss SSS}] %-5p [%c(line:%L):(%t) %x] %m (consume:%-4r) %n

log4j.appender.infofile = net.yasion.common.core.logger.log4j.appender.ExternallyDailyRollingFileAppender

log4j.appender.infofile.DatePattern = yyyy-MM-dd

log4j.appender.infofile.Exactly = true

log4j.appender.infofile.MaxBackupIndex = 30

log4j.appender.infofile.MaxFileSize = 20MB

log4j.appender.infofile.Threshold = INFO

log4j.appender.infofile.encoding = UTF-8

log4j.appender.infofile.File = /WEB-INF/logs/info.log

log4j.appender.infofile.Append = true

log4j.appender.infofile.ImmediateFlush = true

log4j.appender.infofile.layout = org.apache.log4j.PatternLayout

log4j.appender.infofile.layout.ConversionPattern = [%d{yyyy-MM-dd HH:mm:ss SSS}] %-5p [%c(line:%L):(%t) %x] %m (consume:%-4r) %n

log4j.appender.fatalfile = net.yasion.common.core.logger.log4j.appender.ExternallyDailyRollingFileAppender

log4j.appender.fatalfile.DatePattern = yyyy-MM-dd

log4j.appender.fatalfile.Exactly = true

log4j.appender.fatalfile.MaxBackupIndex = 30

log4j.appender.fatalfile.MaxFileSize = 20MB

log4j.appender.fatalfile.Threshold = FATAL

log4j.appender.fatalfile.encoding = UTF-8

log4j.appender.fatalfile.File = /WEB-INF/logs/fatal.log

log4j.appender.fatalfile.Append = true

log4j.appender.fatalfile.ImmediateFlush = true

log4j.appender.fatalfile.layout = org.apache.log4j.PatternLayout

log4j.appender.fatalfile.layout.ConversionPattern = [%d{yyyy-MM-dd HH:mm:ss SSS}] %-5p [%c(line:%L):(%t) %x] %m (consume:%-4r) %n

从配置文件可以知道，默认情况下控制台打印出INFO级别的日志，而其他级别的日志会打印到/WEB-INF/logs目录下面对应的日志文件中，使用分块的Appender，保持每个文件10MB的大小，保存10个文件。这些配置都可以按实际需求修改。

## 框架公共模块

### 系统设置

#### 单位管理

##### 功能

实现单位的增删改查

#### 用户管理

##### 功能

实现用户的增删改查

#### 角色管理

##### 功能

实现角色的增删改查

#### 用户角色绑定

##### 功能

实现用户和角色之间的关联

#### 权限管理

##### 功能

实现权限的增删改查

#### 角色权限绑定

##### 功能

实现角色和权限之间的关联

#### 日志管理

##### 功能

实现日志的删除查看

### 系统文件

#### 统一文件管理

##### 功能

实现所有上传的文件的删除查看

#### UEditor文件管理

##### 功能

实现UEditor上传的文件的删除查看

### 调查问卷

#### 问卷管理

##### 功能

实现问卷的增删改查

#### 题目管理

##### 功能

实现题目的增删改查

#### 问卷统计

##### 功能

实现相关问卷的数据统计

### 个人空间

#### 个人资料

##### 功能

实现个人资料编辑

#### 个人经历

##### 功能

实现个人经历编辑

## 开发流程

### 说明

虽然已经把整个框架做了很详细的介绍，但是还是需要一个例子才能说明清楚，下面以日志管理模块为例子，详细说一下怎么开发一个模块。

一般来说开发的顺序，首先自己做一下基本的需求分析，接着创建数据库表，然后编写好Java代码，再配置映射文件（可以跟实体创建时候一同整理），最后编写前台页面的代码的。

### 基本需求分析

由于日志功能只提供用户的查看，所以不需要提供能修改功能。日志是由系统记录，并且日志保存的数据库事务不能跟任何功能模块的数据库事务共用，原因前面已经说过了，由于框架默认的保存方式的实现是共用数据库事务的，所以需要提供一个特殊的保存方法。日志可以被删除，所以需要一个删除方法。日志需要被查看或者查询，所有需要查询方法。

前台页面需要一个基本的显示，所以我们使用列表页面，加上一个展示信息的窗口来查看详细信息。提供删除、刷新、搜索按钮，通过双击具体一条记录来查看详细信息。于是，我们得出前台页面只是一个基本的要求，特殊的只有搜索功能。其实就是一个很普通的功能。

### 创建数据表

创建指定的数据库表，由于是框架的限制，id（主键）的字段必须要使用varchar(32)的类型，原因就是框架需要使用union-subclass方式，id生成器必须使用uuid.hex，Hibernate的id生成器生成的uuid都是32个字符长度的字符串。

### Java代码

#### 说明

一般来说，Java代码的创建顺序，先建好实体和DTO，再去建DAO、Service，最后才是Action。

#### 实体层（Model）

实体一般都是通过Hibernate反向工程生成的，有一些地方需要修改，以符合框架的要求。生成的实体必须修改成继承于BaseModel<String>，泛型绑定是String的原因是因为id是字符串；删除implements Serializable，因为BaseModel已经实现了Serializable；接着生成一个serialVersionUID；再把实体的id、createUserId、modifiedUserId、createDate、modifiedDate、createUnitId、modifiedUnitId、status、flag字段删除，因为BaseModel已经有相关的字段了。

代码如下：

package net.yasion.common.model;

/\*\*

\* TbLog entity. @author MyEclipse Persistence Tools

\*/

public class TbLog extends BaseModel<String> {

private static final long serialVersionUID = 1695609152782529947L;

// Fields

private String title;

private String loggerType;

private String userId;

private String userName;

private String ipAddress;

private String url;

private String logTime;

private String content;

//。。。。。省略余下getter和setter

#### 数据转换层（DTO）

DTO层的类一般就只需要把model层的类的字段复制过来，生成对应getter和setter，修改继承为BaseDTO<String>类，并且删除serialVersionUID字段既可。不过如果有一些不同字段，就要自行在DTO类添加，比如保存密码的字段，DTO类需要一个旧密码、新密码、新密码确认密码字段，而model层只需要密码字段，并且 这些转换需要自行处理好。

代码如下：

package net.yasion.common.dto;

import java.util.HashMap;

import java.util.Map;

import net.yasion.common.utils.enumeration.OperatorType;

public class LogDTO extends BaseDTO<String> {

// Fields

private String title;

private String loggerType;

private String userId;

private String userName;

private String ipAddress;

private String url;

private String logTime;

private String content;

//。。。。。省略余下getter和setter

#### 数据库层（DAO）

DAO层需要先创建一个接口，该接口必须继承于IBaseDAO<TbLog, String>，IBaseDAO需要绑定实体和id的泛型，就是对应的TbLog和String，并且前面说过我们需要自己实现一个特殊的保存方法，所以接口声明一个saveOnNewSession方法，提供独立事务的保存方法。

代码如下：

package net.yasion.common.dao;

import java.io.Serializable;

import net.yasion.common.model.TbLog;

public interface ILogDAO extends IBaseDAO<TbLog, String> {

public Serializable saveOnNewSession(TbLog entity);

}

接着创建该接口的实现类，该实现类必须继承于BaseDAOImpl<TbLog, String>并且实现implements ILogDAO。

代码如下：

package net.yasion.common.dao.impl;

import java.io.Serializable;

import net.yasion.common.dao.ILogDAO;

import net.yasion.common.model.TbLog;

import org.hibernate.Session;

import org.hibernate.Transaction;

import org.hibernate.criterion.Order;

import org.springframework.stereotype.Repository;

@Repository("logDAO")

public class LogDAOImpl extends BaseDAOImpl<TbLog, String> implements ILogDAO {

@Override

protected Class<TbLog> getEntityClass() {

return TbLog.class;

}

@Override

public Serializable saveOnNewSession(TbLog entity) {

Serializable id = null;

Session session = sessionFactory.openSession();

Transaction tx = session.beginTransaction();

try {

id = session.save(entity);

tx.commit();

} catch (Exception e) {

tx.rollback();

e.printStackTrace();

} finally {

session.close();

}

return id;

}

@Override

protected Order[] getOrders() {

return new Order[] { Order.desc("logTime") };

}

}

DAO层的实现类必须加上@Repository("XXXXX")注解的修饰，意思是把实现类交给Spring管理，方便Spring自动注入，Repository是用来表明这是DAO层的自动注入。"XXXXX"就是spring bean的bean名字。当自动注入的时候，spring会把bean注入到名字为XXXXX的变量中。

DAO层的类必须实现抽象方法BaseDAOImpl::getEntityClass，以表明其操作的实体。

上面的实现类已经实现了saveOnNewSession方法，并且该实现类覆盖了BaseDAOImpl::getOrders方法，因为日志需要按日期递减的形式显示。BaseDAOImpl::getOrders是为数据查询时候提供默认的排序方式的。

#### 事务逻辑层（Service）

Service层也需要先创建一个接口，该接口需要继承于IBaseService<TbLog, LogDTO, String>接口，IBaseService需要绑定实体、DTO、id的泛型，就是对应的TbLog、LogDTO、String。由于DAO层提供了saveOnNewSession的接口，但是saveOnNewSession不是统一的接口，所以Serivce层也需要声明一个saveOnNewSession方法，以提供展现层的调用。

代码如下：

package net.yasion.common.service;

import net.yasion.common.dto.LogDTO;

import net.yasion.common.model.TbLog;

public interface ILogService extends IBaseService<TbLog, LogDTO, String> {

public TbLog saveOnNewSession(TbLog log);

}

接着创建该接口的实现类，该实现类必须继承于BaseServiceImpl<TbLog, LogDTO, String>并且实现implements ILogService。

代码如下：

package net.yasion.common.service.impl;

import net.yasion.common.dao.IBaseDAO;

import net.yasion.common.dao.ILogDAO;

import net.yasion.common.dto.LogDTO;

import net.yasion.common.model.TbLog;

import net.yasion.common.service.ILogService;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import org.springframework.transaction.annotation.Transactional;

@Service("logService")

@Transactional

public class LogServiceImpl extends BaseServiceImpl<TbLog, LogDTO, String> implements ILogService {

private ILogDAO logDAO;

public ILogDAO getLogDAO() {

return logDAO;

}

@Autowired

public void setLogDAO(ILogDAO logDAO) {

this.logDAO = logDAO;

}

@Override

public TbLog saveOnNewSession(TbLog log) {

this.logDAO.saveOnNewSession(log);

return log;

}

@Override

protected IBaseDAO<TbLog, String> getDefaultDAO() {

return this.logDAO;

}

}

跟DAO层一样，Service层的类也要加入相关自动注入的注解。Service的实现类必须加上@Service ("XXXXX")注解的修饰。由于Service层涉及到数据库的事务，所以还需要一个注解——@Transactional注解，这个表示这个类对数据库的操作会受到事务的控制，而且@Transactional注解还可以针对具体的方法，对只读方法（只查询数据的方法）可以加入注解@Transactional(readOnly = true)表明只读事务。在类型上添加了@Transactional注解的类的所有方法，默认的方法注解就是@Transactional(readOnly = false)。

对于@Autowired注解，就是自动注入注解，凡是被这个注解修饰的变量/setter方法，就会把指定跟这个变量名/setter方法名对应一样的spring bean注入到该变量/set到该方法中，以实现spring的自动注入。因此，之前DAO层声明了@Repository("logDAO")的实现类，会自动注入到LogServiceImpl::logDAO变量中。

Service层的类必须实现抽象方法BaseServiceImpl::getDefaultDAO，以表明其操作的默认DAO对象。

上面的实现类已经实现了saveOnNewSession方法，展现层可以调用它以实现相关保存操作。

#### 展现层（Action）

展现层比较自由，只需要继承于BaseAction，然后在类型上面加上@Controller注解即可。至于具体的请求对应什么的处理方法，只需要在指定方法上@RequestMapping("/XXXX/XXXX.do")就可以关联对应请求，这里就是相当于凡是"/XXXX/XXXX.do"请求过来的，都由这个方法处理。值得一提，凡是要返回页面的请求，与其关联的方法都必须有返回值，以表示返回的页面。一般都是String类型的返回值，直接表示放回页面的路径，也可以是ModelAndView类型，效果跟String一样。

代码如下：

package net.yasion.common.web.action;

import java.io.PrintWriter;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import net.sf.json.JSONObject;

import net.sf.json.JsonConfig;

import net.yasion.common.annotation.ModelDTOSearch;

import net.yasion.common.constant.CommonConstants;

import net.yasion.common.core.logger.enumeration.LoggerType;

import net.yasion.common.dto.LogDTO;

import net.yasion.common.model.TbLog;

import net.yasion.common.service.ILogService;

import net.yasion.common.utils.AfxBeanUtils;

import net.yasion.common.utils.interfaces.IResultSet;

import org.apache.commons.lang.StringUtils;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Controller;

import org.springframework.ui.ModelMap;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RequestMethod;

import org.springframework.web.bind.annotation.RequestParam;

import org.springframework.web.bind.annotation.ResponseBody;

@Controller

public class LogAct extends BaseAction {

private ILogService logService;

public ILogService getLogService() {

return logService;

}

@Autowired

public void setLogService(ILogService logService) {

this.logService = logService;

}

@RequestMapping("/logger/list.do")

public String list(HttpServletRequest request, LogDTO dto, ModelMap model, Integer pageNumber) {

pageNumber = (pageNumber != null && pageNumber != 0 ? pageNumber : 1);

IResultSet<TbLog> resultSet = logService.findByDTO(dto, pageNumber, CommonConstants.PAGESIZE);

model.put("resultSet", resultSet);

return "common/logger/list";

}

@RequestMapping(method = RequestMethod.POST, value = "/logger/delete.do")

@ResponseBody

public void delete(@RequestParam(value = "ids[]") String[] ids, ModelMap model, HttpServletResponse response) {

PrintWriter out = null;

try {

response.setContentType("text/json;charset=UTF-8");

response.setCharacterEncoding("UTF-8");

response.setHeader("Cache-Control", "no-cache");

out = response.getWriter();

int count = logService.removeByIds(ids);

if (0 < count) {

out.write("{\"result\":true,\"msg\":\"delete success\",\"count\":\"" + count + "\"}");

} else {

out.write("{\"result\":false,\"msg\":\"delete result : 0\"}");

}

} catch (Exception e) {

e.printStackTrace();

out.write("{\"result\":false,\"msg\":\"Message:" + e.toString() + "\\r\\n\\r\\nCauseBy:" + e.getCause() + "\"}");

}

}

@RequestMapping(method = RequestMethod.POST, value = "/logger/find.do")

@ResponseBody

public void find(String id, ModelMap model, HttpServletResponse response) {

PrintWriter out = null;

try {

response.setContentType("text/json;charset=UTF-8");

response.setCharacterEncoding("UTF-8");

response.setHeader("Cache-Control", "no-cache");

out = response.getWriter();

TbLog entity = logService.findById(id);

LogDTO dto = new LogDTO();

AfxBeanUtils.copySamePropertyValue(entity, dto);

dto.setLoggerType(LoggerType.LOGIN.toString().equals(dto.getLoggerType()) ? "登陆日志" : "普通日志");

dto.setOperatedUnitId((StringUtils.isBlank(entity.getModifiedUnitId()) ? entity.getCreateUnitId() : entity.getModifiedUnitId()));

JsonConfig jsonConfig = new JsonConfig();

jsonConfig.setExcludes(AfxBeanUtils.getComplexFieldNames(LogDTO.class));

JSONObject jsonModel = JSONObject.fromObject(dto, jsonConfig);

JSONObject jsonResult = new JSONObject();

jsonResult.element("result", true);

jsonResult.element("entity", jsonModel);

out.write(jsonResult.toString());

} catch (Exception e) {

e.printStackTrace();

out.write("{\"result\":false,\"msg\":\"Message:" + e.toString() + "\\r\\n\\r\\nCauseBy:" + e.getCause() + "\"}");

}

}

}

对于@Autowired注解，跟Service层那里一样。因此，之前Service层声明了@Service("logService")的实现类，会自动注入到LogAct::logService变量中。其他的就是请求处理的相关方法。

list方法：展现日志页面，实现日志基本的查询（现在查询还没有加上去）

delete方法：实现删除指定日志

find方法：实现指定日志的获取

当action层开发完成，就可以开始前台页面的编写工作了。

### Hbm.xml映射文件

#### 操作

映射文件也是通过Hibernate反向工程跟实体一同生成的，所以也有一些地方需要修改。映射文件必须修改成继承于net.yasion.common.model.BaseModel，class类型修改为union-subclass；删除id、createUserId、modifiedUserId、createDate、modifiedDate、createUnitId、modifiedUnitId、status、flag的映射，因为BaseModel的映射文件已经有相关的映射了。

代码如下：

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

<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"

"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<!-- Mapping file autogenerated by MyEclipse Persistence Tools -->

<hibernate-mapping>

<union-subclass name="net.yasion.common.model.TbLog" extends="net.yasion.common.model.BaseModel" table="tb\_log">

<property name="title" type="java.lang.String">

<column name="title" length="128" />

</property>

<property name="loggerType" type="java.lang.String">

<column name="logger\_type" length="32" />

</property>

<property name="userId" type="java.lang.String">

<column name="user\_id" />

</property>

<property name="userName" type="java.lang.String">

<column name="user\_name" />

</property>

<property name="ipAddress" type="java.lang.String">

<column name="ip\_address" length="128" />

</property>

<property name="url" type="java.lang.String">

<column name="url" />

</property>

<property name="logTime" type="java.lang.String">

<column name="log\_time" length="64" />

</property>

<property name="content" type="java.lang.String">

<column name="content" length="512" />

</property>

</union-subclass>

</hibernate-mapping>

#### 注意事项

值得注意的地方，如果映射文件有catalog属性，必须把它删除，否则会绑死了数据库，导致一些莫名其妙的问题。

### 前台页面

#### 说明

前台页面的开发，先创建好对应模块的javascript文件、css文件、jsp文件，把这些文件放到指定的目录下。

分别是：

WebContent/static/common/js/logger/list.js

WebContent/static/common/css/logger/list.css

WebContent/WEB-INF/pages/common/logger/list.jsp

当然如果是新项目就不能放到common文件夹中了，需要放到自己项目的文件夹中。

#### JSP页面

直接按要求实现相关代码就是，不过记得引入框架提供的统一的公共头文件。以控制统一的样式和风格。当然如果需求指明了不同的风格，就按照需要引入相应的头文件吧。

统一的公共头文件引入：

<c:import url="/WEB-INF/pages/common/header.jsp" />

CSS的头文件引入：

<c:import url="/WEB-INF/pages/common/cssHeader.jsp" />

JS的头文件引入：

<c:import url="/WEB-INF/pages/common/jsHeader.jsp" />

全局JS变量的头文件引入：

<c:import url="/WEB-INF/pages/common/globalHeader.jsp" />

当然你还必须把当前模块的js文件和css文件引入。

<script src="${base}/static/common/js/logger/list.js"></script>

<link href="${base}/static/common/css/logger/list.css" rel="stylesheet" type="text/css" />

代码如下：

<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>

<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>

<%@ taglib uri="http://java.sun.com/jsp/jstl/functions" prefix="fn"%>

<%@ taglib uri="/yahnfunction" prefix="yahnfn"%>

<%@ taglib uri="/yahncore" prefix="yahnc"%>

<yahnc:constants />

<!DOCTYPE>

<html>

<head>

<title>日志管理</title>

<meta http-equiv="content-type" content="text/html; charset=UTF-8" />

<c:import url="/WEB-INF/pages/common/header.jsp" />

<script src="${base}/static/common/js/logger/list.js"></script>

<link href="${base}/static/common/css/logger/list.css" rel="stylesheet" type="text/css" />

</head>

<body style="padding: 10px;">

<table class="table table-striped table-hover table-bordered" id="mainTable">

<caption>

<h1>日志管理</h1>

</caption>

<thead>

<tr>

<td id="id" class="span1"><label class="checkbox"><input type="checkbox" class="checkbox checkAll" />序号</label></td>

<td id="title" class="span1">主题</td>

<td id="loggerType" class="span1">日志类型</td>

<td id="userName" class="span1">用户名称</td>

<td id="ipAddress" class="span1">IP地址</td>

<td id="url" class="span2">访问地址</td>

<td id="logTime" class="span2">记录时间</td>

</tr>

</thead>

<tbody>

<c:if test="${not empty resultSet}">

<c:forEach var="log" items="${resultSet.resultList}" varStatus="status">

<tr onselectstart="return false;" style="-moz-user-select: none">

<td class="span1"><label class="checkbox"><input type="checkbox" class="checkbox" />${status.count}</label><input type="hidden" name="loggerId" identity="true" value="${fn:escapeXml(log.id)}" /></td>

<td class="span1">${fn:escapeXml(log.title)}</td>

<c:choose>

<c:when test="${fn:escapeXml(log.loggerType) eq 'LOGIN'}">

<td class="span1">登陆日志</td>

</c:when>

<c:otherwise>

<td class="span1">普通日志</td>

</c:otherwise>

</c:choose>

<td class="span1">${fn:escapeXml(log.userName)}</td>

<td class="span1">${fn:escapeXml(log.ipAddress)}</td>

<td class="span2">${fn:escapeXml(log.url)}</td>

<td class="span2">${fn:escapeXml(log.logTime)}</td>

</tr>

</c:forEach>

</c:if>

</tbody>

<tfoot>

<!-- 分页 START -->

<tr>

<td colspan="9"><yahnc:pagelet url="${base}/logger/list.do?1=1" resultSet="${resultSet}" /></td>

</tr>

<!-- 分页 END -->

<tr>

<td colspan="9">

<button class="btn btn-primary" onclick="location.reload();">刷新</button> <c:if test="${yahnfn:isRoles(sessionScope.login\_user,fn:split(ADMIN\_ROLE\_CODE,','))}">

<button class="btn btn-danger" data-toggle="modal" onclick="doDelete();">删除</button>

</c:if>

<button class="btn btn-info" data-toggle="modal" data-target="#searchDialog">搜索</button>

</td>

</tr>

</tfoot>

</table>

<!--增加编辑框 START-->

<div class="modal hide fade" id="editDialog">

<form action="save.do" method="post" id="saveForm" class="form-horizontal">

<input type="hidden" id="operType" /> <input type="hidden" id="url" /> <input type="hidden" name="id" clear="true" />

<div class="modal-header">

<a class="close" data-dismiss="modal">×</a>

<h3 id="tableTitle">日志信息</h3>

</div>

<div class="modal-body">

<table>

<tbody>

<tr>

<td><label class="sr-only" for="title" style="width: 280px;">主题&nbsp;&nbsp;&nbsp;</label> <input type="text" class="form-control" name="title" placeholder="*主题*" readonly="readonly" /></td>

<td><label class="sr-only" for="loggerType">日志类型</label> <input type="text" class="form-control" name="loggerType" placeholder="*日志类型*" readonly="readonly" /></td>

</tr>

<tr>

<td>

<div class="form-group">

<label class="sr-only" for="userId">用户ID</label> <input type="text" class="form-control" name="userId" placeholder="*用户*ID" readonly="readonly" />

</div>

</td>

<td><label class="sr-only" for="userName">用户名称</label> <input type="text" class="form-control" name="userName" placeholder="*用户名称*" readonly="readonly" /></td>

</tr>

<tr>

<td>

<div class="form-group">

<label class="sr-only" for="ipAddress">IP地址</label> <input type="text" class="form-control" name="ipAddress" placeholder="IP*地址*" readonly="readonly" />

</div>

</td>

<td><label class="sr-only" for="url">访问地址</label> <input type="text" class="form-control" name="url" placeholder="*访问地址*" readonly="readonly" /></td>

</tr>

<tr>

<td>

<div class="form-group">

<label class="sr-only" for="logTime">记录时间</label> <input type="text" class="form-control" name="logTime" placeholder="*记录时间*" readonly="readonly" />

</div>

</td>

<td></td>

</tr>

<tr>

<td colspan="2">

<div class="form-group">

<label class="sr-only" for="content">备注</label>

<textarea style="width: 500px;" name="content" readonly="readonly"></textarea>

</div>

</td>

</tr>

</tbody>

</table>

</div>

</form>

<div class="modal-footer">

<button type="button" class="btn" data-dismiss="modal">关闭</button>

</div>

</div>

<!--增加编辑框 END-->

<c:import url="/WEB-INF/pages/common/modalDialog.jsp" />

</body>

</html>

上面的页面是还没有实现搜索功能。因为分页功能简单，已经直接通过标签实现。

#### Javascript脚本

Js脚本只需要实现相关的查看和删除的控制就可以了，因为不会涉及到新增和修改的功能。搜索功能已经在框架公共的Js处理了。

代码如下：

/\*\*

\* 列表页匹配框JS

\*/

$(function() {

$("select").select2();// 启用Select2插件

$("#searchDialog select[name=loggerType]").select2({

placeholder : "日志类型",

allowClear : true

});

});

function doDelete() {

var idObjs = $("#mainTable tbody input[type=checkbox]:checked:not(:hidden)").parent().next("[type=hidden][name=loggerId]");

if (0 < idObjs.length) {

$('#confirmDialog #confirmDialogH4').text('操作确认提示');

$('#confirmDialog #confirmContext').text('是否确认删除这' + idObjs.length + '条记录');

$('#confirmDialog #confirmBtn').text('确认');

$('#confirmDialog #confirmBtn').click(function() {

var ids = "{\"ids\":[";

var vals = "";

idObjs.each(function(index, element) {

if (idObjs.length - 1 > index) {

vals += ("\"" + element.value + "\",");

} else {

vals += ("\"" + element.value + "\"");

}

});

ids = (ids + vals + "]}");

ids = jQuery.parseJSON(ids);

sendData(ids, deleteSuccess, '/logger/delete.do');

$('#confirmDialog').modal('hide');

});

$('#confirmDialog #cancelBtn').text('取消');

$('#confirmDialog #cancelBtn').click(function() {

$('#confirmDialog').modal('hide');

});

$("#confirmDialog").modal('show');

}

}

function doSaveOrUpdata(oper, caller, type) {

if ("save" === oper) {

$("#editDialog #operType").val("save");

$("#editDialog #tableTitle").text("日志信息");

$("#editDialog #url").val("/logger/save.do");

$("#editDialog input:eq(3)").removeAttr("readonly");

$('#editDialog').modal('show');

} else {

$("#editDialog #operType").val("update");

$("#editDialog #tableTitle").text("日志信息");

$("#editDialog #url").val("/logger/update.do");

var idObj = $(caller).find("input[type=hidden][name=loggerId]");

getData("/logger/find.do", $("#editDialog"), idObj.val(), setAjaxForm);

}

}

#### CSS样式

代码如下：

@CHARSET "UTF-8";

form input[type="text"] {

height: auto;

}

### 加入搜索功能

#### 说明

上面已经完成了基本的一个功能的开发，下面是加入搜索功能。该功能需要对DTO层和action层做一下修改，前台JSP页面加入一个搜索窗口。

#### DTO层修改

覆盖BaseDTO::generateDefOperateRelation即可。

代码如下：

public class LogDTO extends BaseDTO<String> {

// Fields

private String title;

private String loggerType;

private String userId;

private String userName;

private String ipAddress;

private String url;

private String logTime;

private String content;

//**省略**getter**和**setter...

@Override

public void generateDefOperateRelation(String requestURL, Map<String, String[]> paramsMap) {

HashMap<String, OperatorType> operateRelationMap = new HashMap<String, OperatorType>();

operateRelationMap.put("title", OperatorType.LIKE);

operateRelationMap.put("loggerType", OperatorType.EQ);

operateRelationMap.put("userId", OperatorType.EQ);

operateRelationMap.put("userName", OperatorType.LIKE);

operateRelationMap.put("ipAddress", OperatorType.LIKE);

operateRelationMap.put("url", OperatorType.LIKE);

operateRelationMap.put("logTime", OperatorType.BETWEEN);

operateRelationMap.put("content", OperatorType.LIKE);

this.setOperateRelation(operateRelationMap);

}

}

#### Action层修改

在查询的请求处理方法上的DTO对象参数加上@ModelDTOSearch注解即可。

代码如下：

@RequestMapping("/logger/list.do")

public String list(HttpServletRequest request, @ModelDTOSearch LogDTO dto, ModelMap model, Integer pageNumber)

//**省略代码**...

#### JSP页面修改

加入查询窗口，并且指定好查询请求的url和对应的字段即可。

代码如下：

<!--搜索框 START-->

<div class="modal hide fade" id="searchDialog">

<form action="${base}/logger/list.do?1=1" method="post" id="searchForm" class="form-horizontal">

<div class="modal-header">

<a class="close" data-dismiss="modal">×</a>

<h3 id="tableTitle">搜索信息</h3>

</div>

<div class="modal-body">

<table>

<tbody>

<tr>

<td>

<div class="form-group">

<label class="sr-only" for="title" style="width: 280px;">主题</label> <input type="text" class="form-control" name="title" placeholder="*主题*" value="${fn:escapeXml(title)}" />

</div>

</td>

<td>

<div class="form-group">

<label class="sr-only" for="loggerType">日志类型</label> <select class="form-control" name="loggerType" style="width: 206px;" value="${fn:escapeXml(loggerType)}" placeholder="*日志类型*">

<option value=""></option>

<option value="LOGIN" <c:if test="${fn:escapeXml(loggerType) == 'LOGIN'}">selected="selected"</c:if>>登录日志</option>

<option value="NORMAL" <c:if test="${fn:escapeXml(loggerType) == 'NORMAL'}">selected="selected"</c:if>>普通日志</option>

</select>

</div>

</td>

</tr>

<tr>

<td>

<div class="form-group">

<label class="sr-only" for="userName">用户名称</label> <input type="text" class="form-control" name="userName" placeholder="*用户名称*" value="${fn:escapeXml(userName)}" />

</div>

</td>

<td>

<div class="form-group">

<label class="sr-only" for="ipAddress">IP地址</label> <input type="text" class="form-control" name="ipAddress" placeholder="IP*地址*" value="${fn:escapeXml(ipAddress)}" />

</div>

</td>

</tr>

<tr>

<td>

<div class="form-group">

<label class="sr-only" for="url">访问地址</label> <input type="text" class="form-control" name="url" placeholder="*访问地址*" value="${fn:escapeXml(url)}" />

</div>

</td>

<td></td>

</tr>

<tr>

<td colspan="2">

<div class="form-group">

<label class="sr-only" for="logTime">记录时间</label>&nbsp;&nbsp;&nbsp;从&nbsp;&nbsp;&nbsp;<input type="text" name="logTime" value="${fn:escapeXml(operateValue['logTime'][0])}" placeholder="*记录时间*" onClick="WdatePicker({dateFmt:'yyyy-MM-dd HH:mm:ss'});" />&nbsp;&nbsp;&nbsp;到&nbsp;&nbsp;&nbsp;<input

type="text" name="logTime" value="${fn:escapeXml(operateValue['logTime'][1])}" placeholder="*记录时间*" onClick="WdatePicker({dateFmt:'yyyy-MM-dd HH:mm:ss'});" />

</div>

</td>

</tr>

</tbody>

</table>

</div>

</form>

<form action="${base}/logger/list.do?1=1" method="post" id="backupForm" style="display: none;">

<input type="hidden" name="title" value="${fn:escapeXml(title)}" /> <input type="hidden" name="loggerType" value="${fn:escapeXml(loggerType)}" /> <input type="hidden" name="userName" value="${fn:escapeXml(userName)}" /> <input type="hidden" name="ipAddress"

value="${fn:escapeXml(ipAddress)}" /> <input type="hidden" name="url" value="${fn:escapeXml(url)}" /> <input type="hidden" name="logTime" value="${fn:escapeXml(operateValue['logTime'][0])}" /> <input type="hidden" name="logTime"

value="${fn:escapeXml(operateValue['logTime'][1])}" />

</form>

<div class="modal-footer">

<button class="btn btn-primary" id="searchBtn" data-toggle="modal">搜索</button>

<button class="btn btn-warning" id="clearSearchBtn" data-toggle="modal">清空</button>

<button type="button" class="btn" data-dismiss="modal">关闭</button>

</div>

</div>

<!--搜索框 END-->

### 注意事项

##### 命名规范

模块的命名规则都是根据模块功能的英文翻译来指定的，比如现在的权限，就是permission。

Java类的命名上的一些规定：

接口必须以“I”开头的

实现类必须以“Impl”结尾

DAO层的接口必须以“DAO”结尾

DAO层的接口实现类必须以“DAOImpl”结尾

Service层的接口必须以“Service”结尾

Service层的接口实现类必须以“ServiceImpl”结尾

DTO层的类必须以“DTO”结尾

Model层的类必须以“Tb”开头

Action层的类必须以“Act”结尾

然后模块功能的翻译加上这些控制，

那么权限模块的Java类的名称就变成这样：

DAO：IPermissionDAO、PermissionDAOImpl

Service：IPermissionService、PermissionServiceImpl

DTO：PermissionDTO

Model：TbPermission

Action：PermissionAct

Hbm.xml映射文件命名的规定：

等于实体类型

前台页面的命名的规定：

展示页面（列表页面）命名为“list.jsp”

编辑页面（如果有）命名为“edit.jsp”

导出页面（如果有）命名为“table.jsp”

对应js脚本和css样式也是按照这样的命名只不过把扩展名改了而已。

那么权限模块的前台页面名称就变成这样：

list.jsp、list.js、list.css

##### 文件存放位置

假如有新项目，请先按照框架目录的要求，创建好相应的文件夹。然后前台页面再按照模块划分好文件夹。

例子：

rigs项目

Java类：

src/net/yasion/rigs/web/action

src/net/yasion/rigs/constant

src/net/yasion/rigs/dao

src/net/yasion/rigs/dao/impl

src/net/yasion/rigs/dto

src/net/yasion/rigs/model

src/net/yasion/rigs/service

src/net/yasion/rigs/service/impl

sql文件：

sql/rigs

hbm.xml映射文件：

WebContent/WEB-INF/config/hibernate/mapping/rigs

tld标签配置文件：

WebContent/WEB-INF/config/tld/rigs

Js脚本：

WebContent/static/custom/rigs/js

Css样式：

WebContent/static/custom/rigs/css

Img图像：

WebContent/static/custom/rigs/img

Jsp页面：

WebContent/WEB-INF/pages/rigs

上传文件：

WebContent/upload/rigs

按模块划分前台文件：

Jsp页面

WebContent/WEB-INF/pages/rigs/awards

WebContent/WEB-INF/pages/rigs/extend

WebContent/WEB-INF/pages/rigs/issueSituaction

WebContent/WEB-INF/pages/rigs/paper

……

Js脚本

WebContent/static/custom/rigs/js/award

WebContent/static/custom/rigs/js/extend

WebContent/static/custom/rigs/js/issueSituaction

WebContent/static/custom/rigs/js/paper

……

Css样式

WebContent/static/custom/rigs/css/award

WebContent/static/custom/rigs/css/extend

WebContent/static/custom/rigs/css/issueSituaction

WebContent/static/custom/rigs/css/paper

……

Img样式

WebContent/static/custom/rigs/img/award

WebContent/static/custom/rigs/img/extend

WebContent/static/custom/rigs/img/issueSituaction

WebContent/static/custom/rigs/img/paper

……

## 注意事项

### 修改规范

不要随便修改任何common文件夹的文件，避免整个框架受影响；有特殊公共功能先自己加入到指定模块，然后再上报，最后再由我整理到主框架。

### 指定编码

上传文件时候必须使用HttpUtils的upload方法进行上传，因为HttpUtils会进行md5比较是否重复上传和记录上传文件数据到数据库等操作。