# 总体设计

系统如何分层，是否使用接口，源文件的目录结构及命名规范；主页面与子页面的链接方式（对话框or跳转），只读字段的显示，动态字段的处理，验证，提示框与确认框的实现，页面冻结，什么情况下使用ajax，什么情况下使用传统的请求/响应；如何实现字典基本操作；

# 数据库

存储过程/函数命名规范

# javascript

所有事件处理函数的绑定都在js中完成，

所有的请求发出后，都应该使用$.blockUI冻结界面

程序变量

html元素jquery变量：

$<组件id>

首先，使用jquery.validate框架对字段进行基本的验证，如果验证通过，再对多个字段间的逻辑进行验证，必要的情况下，可以使用ajax方式到后台进行验证，主要用在需要从数据库读取数据的情况下。如果使用ajax方式，需要设置为同步请求方式；如果涉及到多个ajax请求，应该使用串行的方式，而不要使用嵌套的方式。

# jsp

css文件引用

字典加载

页面内容

js文件引用

div:

id=divOperation

form:

formOperation

action: <modelPath>/operation

a:

id=aOperation

input:

[id=<fieldname>]

name=<fieldname>

td:

[id=<operation>\_<fieldname>]

# java

## Controller

类

命名规范：使用设计文档中指定的名称，应当符合以下格式，否则，应向设计人员提出质疑：

<ModelPath>Controller

方法

命名规范：使用设计文档中指定的名称，应当符合以下格式，否则，应向设计人员提出质疑：

与url一致，ajax方式的请求url以ajax开头

含有业务逻辑校验的ajax方法必须返回AjaxResult类型的值，此处建议框架提供不带参数的AjaxResult.success()方法，表示请求成功；否则使用AjaxResult.fail(message)来表示请求失败，参数message用于描述失败的具体原因。

提示：Controller内部能进行更新操作，设置持久化对象的状态，因为无法使用事务进行控制。

@RequestHistory与history:解析

redirect:解析

RequestHistoryInterceptor

会将请求url与参数保存到session中

RequestHistoryTilesViewResolver

会将请求重定向到history:后面的url上，如果用户在本次登录期间已经访问过该url，则使用上次访问时的参数

## Service

类：

命名规范

<Entity>Service

建议统一使用@Transactional对service类进行注解

建议不使用接口

方法

命名规范

处理查询返回值中的字典类型的字段：

//getDao().getEntityManager().detach();

decodeBean, decodePage

## Dao

建议BaseDao的update操作能够自动实现以下逻辑：

Entity e2 = findByPk(e.getRid());

BeanUtils.mergeProperties(e2, e);

save(e2);

## Domain

基类的使用：id，createUser, createTime, updateUser, updateTime, isNew

@Transient的使用

@ManyToOne的使用

## 字典常量

程序中经常会遇到使用字典代码的地方，如：使用set方法设置字段值，使用sql/jpql等设置字段值，为了应对可能的字典代码的变化，建议在这些地方使用常量，避免使用子面值，尤其是sql语句中，要把使用字典代码的地方使用参数化的方式来处理。

字典常量有以下几种常见的实现方式：

* 常量

优点：简单；缺点：只能定义key，不能定义value（有些时候，jsp可能会用到，比如根据转译后的字典值来判断适用的操作，显示默认值等）

Public class Gender {

/\*\* 男 \*/

Public static final String NAN=”1”;

/\*\* 女 \*/

Public static final String NV=”2”;

}

* 枚举

优点：可以定义key和value；缺点：不能使用继承简化

Public enum Gender{

NAN(“1”,”男”), NV(“2”, “女”)

Private String key;

Private String value;

Private Gender(key,value) {

This.key = key;

This.value = value;

}

Public String getKey() {

Return key;

}

Public String getValue() {

Return value;

}

}

* 继承

优点：可以定义key和value，可以使用继承来简化一些代码；缺点：稍微有点复杂

Public class Dictionary() {

Public String key;

Public String value;

protected Dictionary(key, value) {

This.key = key;

This.value = value;

}

Public String getKey() {

Return key;

}

Public String getValue() {

Return value;

}

}

Public final class Gender extends Dictionary {

Public static final Dictionary NAN = new Dictionary(“1”, “男”);

Public static final Dictionary NV = new Dictionary(“2”, “女”);

}