

前言

随着WEB UI 框架(EasyUI /Jquery UI /Ext /Dwz)等的逐渐成熟,系统界面逐渐实现统一化,代码生成器也可以生成统一规范的界面! 代码生成+手工MERGE半智能开发将是新的趋势,生成的代码可节省50%工作量,快速提高开发效率!!!

JEECG[J2EE Code Generation] 是一款基于代码生成器的智能开发框架.

史上第一款基于代码生成器的开源J2EE框架,单表和一对多表,增删改查功能自动生成;

JAVA编程有很多重复机械代码,生成器可以帮助解决50%的重复工作,让开发更多关注业务逻辑,从而实现代码生成+手工MERGE的半智能开发.**JEECG**敏捷框架可以有效解决信息孤岛问题,生成统一代码、统一规范、统一设计思路,使你在这个平台上,快速开发出高效高质量代码,缩短项目开发周期。

功能特点

- 架构技术: Struts2+Spring3+Hibernate4+EasyUI1.3
- 代码生成器: 自动生成美观大方的前台页面及后台代码
- 查询条件生成器: 动态拼SQL,追加查询条件
- 页面校验器: 采用EasyUI检验机制
- 完整Spring用户权限
- 报表整合: Excel简易导出工具+Highcharts图形报表
- workflow设计器让业务系统更灵活
- 常用共通封装(数据字典/邮件发送/定时任务/短信接口/Freemarker工具类等..)
- 兼容IE 6、IE 8+和Google等浏览器
- 支持SQL Server、Oracle和MySQL等主流数据库

支持的表关系模型包括

- 1.单表数据模型
- 2.一对多(父子表)数据模型

JEECG代码生成器在总结以往的项目基础上,抽象出2种基础数据模型,它可以根据不同的数据模型智能的生成多套不同的展示形式,让开发更快速,更简单

Google Code: <http://code.google.com/p/jeecg/>

QQ群: 106259349 , 106838471

联系邮箱: zhangdaiscott@163.com

博客: <http://blog.csdn.net/zhangdaiscott>

目录

[第一章 JEECG 架构介绍](#)

[第二章 项目部署说明](#)

[第三章 代码生成器-演示](#)

[第四章 代码生成器-使用规则](#)

[第五章 查询过滤器-查询条件HQL生成器](#)

[第六章 校验器-页面表单校验](#)

[第七章 轻量级权限设计-\(菜单+按钮权限\)](#)

[第八章 项目编码规范](#)

第一章 JEECG 架构介绍

架构技术: Struts2+Spring3+Hibernate4+EasyUI1.3+Spring JDBC

配置思想: 零配置 (约定大于配置)

实现技术点:

[1].代码生成器 (规范的后台代码+统一风格的前台页面)

单表模型和一对多(父子表)数据模型, 增删改查功能生成直接使用;

特点:

A.前台页面字段对应数据库字段生成;

B. 页面字段校验自动生成 (数字类型\必须项\金额类型\时间控件);

C.支持Oracle/Mysql数据库

注意: 代码生成包括JSP页面生成, 代码无需修改, 增删改查功能直接配置使用

[2].查询条件生成器

页面加查询条件, 后台不需要写任何逻辑判断, 动态拼SQL追加查询条件

[3].页面校验器(EasyUI 页面检验机制)

前台页面字段校验采用EasyUI

[4].常用共通封装

数据字典/ 邮件发送/ 定时任务/短信接口/Freemarker模板工具/Jquery

[5].完整Spring用户权限

权限功能: 权限, 角色, 用户(功能直接使用)

[6].Ehcache缓存机制

Ehcache缓存自定义标签 (永久缓存/临时缓存)

[7].报表封装

Excel简易导出工具类+Highcharts图形报表

[8].Hibernate+Spring jdbc 组合使用

Hibernate+Spring jdbc组合使用 (单表操作使用Hibernate; 复杂SQL采用SQL),

[1]SQL设计方案:DB SQL抽离出Java代码, 采用命名规范根据类名和方法名创建SQL文件, 存储SQL;

[2]. 程序自动读取SQL;

[3].SQL读取模式:开发模式和发布模式[SQL加载内存]。

[9].安全的事务回滚机制+安全的数据乐观锁机制

第二章 项目部署说明

提醒: A.项目为MyEclipse工程,如果你的eclipse需要自行调整;
B.目前已经测试通过的数据库有MySQL5、Oracle10g、SqlServer200
C.建议使用google浏览器, EasyUI使用其他浏览器加载慢

项目部署步骤:

第一步. 解压程序, 导入MyEclipse

第二步. 修改配置文件

[1]. 代码生成器配置文件: resources/configuration/database.properties (数据库连接)
resources/configuration/source_path.properties(代码生成路径配置)

[2]. 框架配置文件: resources/config.properties (数据库连接)

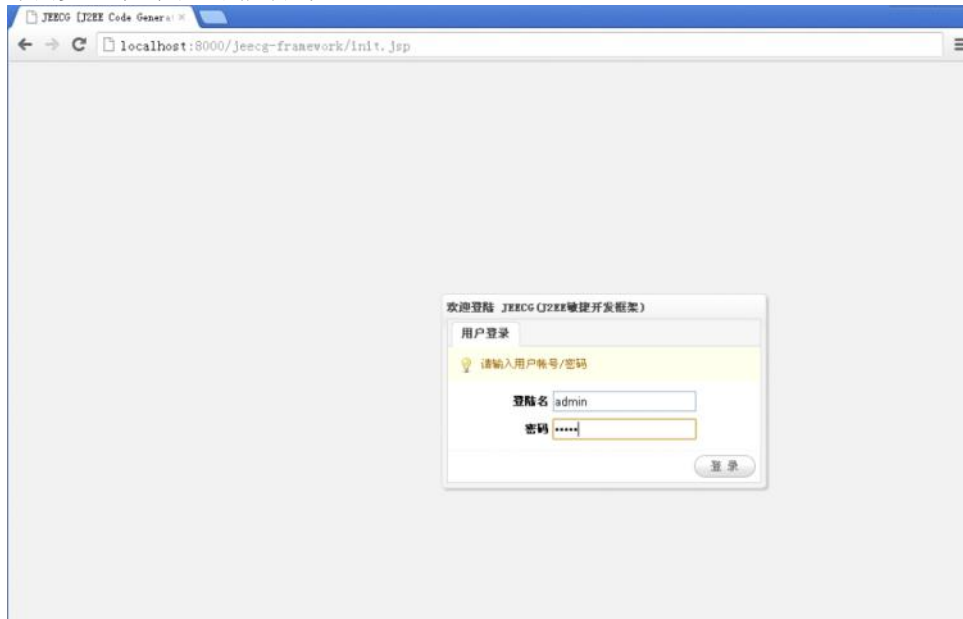
注意:hibernate.hbm2ddl.auto=create

此属性表示项目每次启动, 重新创建数据库表;如果不想重新创建数据库表可改为none

第三步. 发布程序启动项目, 执行init.jsp 初始化数据库数据

例如:<http://localhost:8000/jeecg-framework/init.jsp>

第四步: 登陆系统, 用户账号: admin/admin



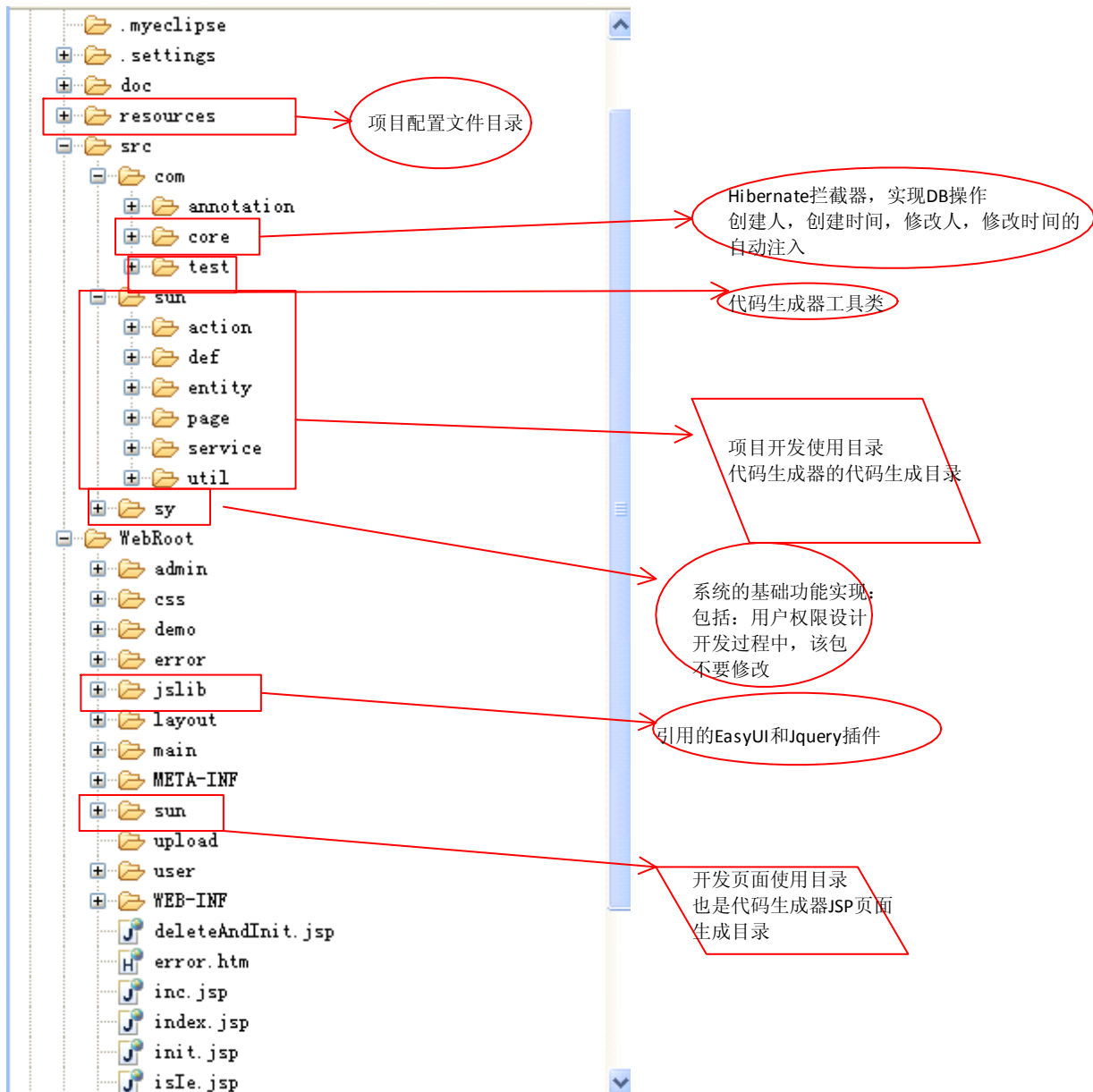
第五步. 代码生成器工具类 (生成器如何使用, 请参照《第四章:生成器使用规则》)

com.test.CodeUtil (单表模型)

com.test.CodeOneToMainUtil (一对多父子模型)

项目工程目录介绍:

JEECG框架: 是一个以代码生成器为基础, 封装了完善的基础功能模块: 包括用户权限, 常用共通封装(数据字典, 邮件接口, 定时任务, Highcharts图像报表等..)的成熟框架, 可直接用于项目开发;



常见部署问题:

说明: 如果你不是使用myeclipse, 切换其他工程的时候, 可能会出现一些问题;

问题整理:

- [1]. WEB目录不是: WebRoot
- [2]. 项目Class设置不是: WebRoot/WEB-INF/classes
- [3]. 创建的表没有字段[obid][create_dt]

第三章 代码生成器-演示

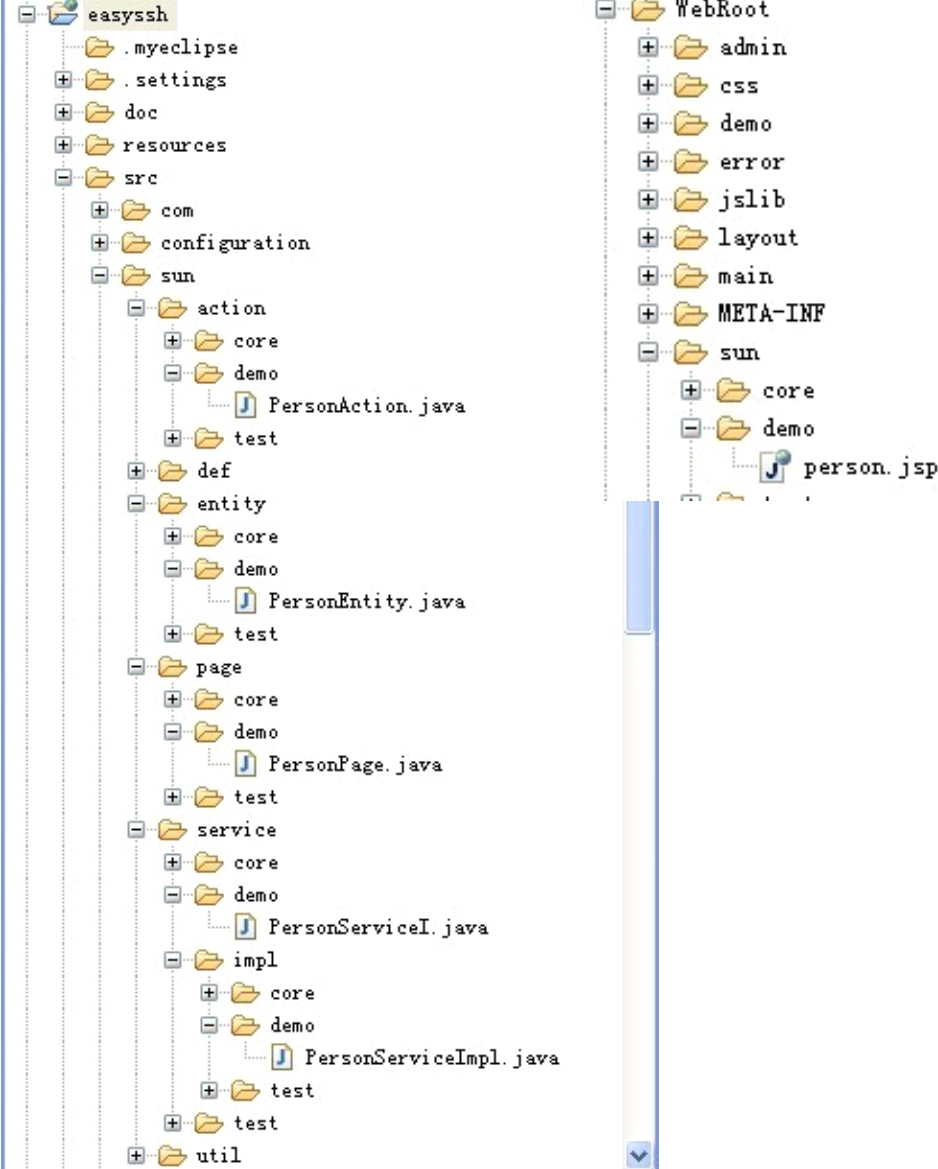
代码生成器界面:

- A.动态选择需要生成文件 (ServiceI\ServiceImpl\Jsp\Action\Entity\Page)
- B.动态选择JSP模板（两种页面风格：行编辑/详细页编辑）

一.代码生成器页面



二.生成代码截图



三.生成的JSP页面（增删改查功能直接使用;页面校验自动生成 {数字类型、金额类型、时间控件、必输等校验}）

搜索条件

查询字段

创建时间

至

最后修改时间

至

搜索

取消

分公司列表

增加

删除

修改

保存

取消编辑

取消选中

	<input type="checkbox"/>	分公司名称	备注
1	<input type="checkbox"/>	北京分公司	北京分公司
2	<input type="checkbox"/>	济南分公司	备注

演示代码生成器使用步骤

##步骤一：生成表SQL

```
CREATE TABLE `person` (  
  `OBID` varchar(36) NOT NULL default '' COMMENT '主键',  
  `NAME` varchar(32) default NULL COMMENT '用户名',  
  `AGE` int(11) default NULL COMMENT '年龄',  
  `SALARY` decimal(10,2) default NULL COMMENT '工资',  
  `createDt` datetime default NULL COMMENT '创建时间',  
  PRIMARY KEY (`OBID`)  
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

注意：建表时，必须给每个字段加上注释，代码生成器会根据注释去生成页面字段对应的显示文本

##步骤二 配置菜单

演示完毕

第四章 代码生成器-使用规则

[一].建表规范

- [1].表必须有唯一主键: OBID (字符类型 36位)
备注: 主键采用UUID方式生成
- [2].表必需字段 (创建人, 创建时间等..)
- [3].表字段必须有注释
备注: JSP页面字段文本, 是根据表字段注释来生成
- [4]. 主表和子表的外键字段名字, 必须相同 (除主键OBID外)
- [5]. 子表引用主表主键OBID作为外键, 外键字段必须以OBID结尾

注释: 请按照建表模板来创建新表, 模板中原有的字段, 生成器会过滤不在页面生成
建表模板:

字段名	类型	长度	备注	主键
OBID	varchar	36	主键	TURE
其他字段..				
CRTUSER	varchar	36	创建人	
CRTUSER_NAME	varchar	32	创建人名字	
CREATE_DT	datetime	0	创建时间	
MODIFIER	varchar	36	修改人	
MODIFIER_NAME	varchar	32	修改人名字	
MODIFY_DT	datetime	0	修改时间	
DELFLAG	int	2	删除标记	
DEL_DT	datetime	0	删除时间	

[二].页面生成规则

- 说明: JSP页面字段的文本内容, 取字段的注释前6位字符(如果建表字段注释为空, 则页面字段文本会为空)
- A.默认生成的JSP页面前五个字段为必须项, 其他字段为非必须输入 (需要自己手工加)
 - B.数据库字段类型为: datetime -->对应页面字段, 会自动追加[年月日-时分秒]时间控件
 - C.数据库字段类型为: date -->对应页面会字段, 自动追加[年月日]时间控件
 - D.数据库字段类型为: Int/Number-->对应页面字段, 会自动追加数字校验 (不允许输入小数)
 - E.数据库字段类型为: float/double/decimal-->对应页面页面字段, 会自动追加数字校验 (允许输入小数)
 - F.如果表字段为字符类型, 并且设置了长度, 页面输入框会自动设置maxlength对应表字段长度

[三].代码生成器工具类

A.单表模型工具类: com.test.CodeUtil

- 代码生成器界面:
- A.动态选择需要生成文件(Service\ServiceImpl\Jsp\Action\Entity\Page)
 - B.动态选择JSP模板 (两种页面风格: 行编辑/详细页编辑)



A-1.详细页编辑页面风格:

公告版 团期收费项目

团购服务项目列表

增加 删除 修改 取消选中

	<input type="checkbox"/>	服务类别	服务名称	备注
1	<input checked="" type="checkbox"/>	酒店	四星含早	
2	<input type="checkbox"/>	团期打包价	团期打包产品（吃住行玩）	
3	<input type="checkbox"/>	机票	斯航UL889/UL107往返	无
4	<input type="checkbox"/>			
5	<input type="checkbox"/>			

编辑团购服务项目

服务类别

地接

服务名称

四星含早

备注

编辑

A-2.行编辑页面风格

团购服务项目列表

增加 删除 修改 保存 取消编辑 取消选中

	<input type="checkbox"/>	服务项目类型	服务项目	供应商	成本价	销售价	日期段
1	<input type="checkbox"/>	酒店	五星级酒店双人间标准	12	12	12	12
2	<input type="checkbox"/>	团期打包价	团期打包产品（吃住行玩）	111	7000	8000	
3	<input checked="" type="checkbox"/>						

B.一对多模型(父子表)工具类：生成的页面明细可以动态添加行

com.test.CodeOneToMainUtil[一对多的增删改查]

第一步:设置一对多代码生成需要参数，执行方法生成代码

```
public static void main(String[] args) {
    //-----
    //第一步：设置主表
    CodeParamEntity codeParamEntityIn = new CodeParamEntity();
    codeParamEntityIn.setTableName("t60_gbuy_order");//主表[表名]
    codeParamEntityIn.setEntityName("GbuyOrder");//主表[实体名]
    codeParamEntityIn.setEntityPackage("order");//主表[包名]
    codeParamEntityIn.setFtlDescription("订单抬头");//主表[描述]
    codeParamEntityIn.setFtl_mode(CodeGenerateOneToMany.FTL_MODE_B);//主表[模板 A:明细单页布局显示 B:明细采用Tab布局展现]

    //第二步：设置子表集合
    List<SubTableEntity> subTabParamIn = new ArrayList<SubTableEntity>();
    //[1].子表一
    SubTableEntity po = new SubTableEntity();
    po.setTableName("t60_gbuy_order_custom");//子表[表名]
    po.setEntityName("GbuyOrderCustom");//子表[实体名]
    po.setEntityPackage("order");//子表[包]
    po.setFtlDescription("订单客户明细");//子表[描述]
    po.setForeignKeys(new String[]{"GORDER_OBID","GO_ORDER_CODE"});//子表[外键:与主表关联外键]
    subTabParamIn.add(po);
    //[2].子表二
    SubTableEntity po2 = new SubTableEntity();
    po2.setTableName("t60_gbuy_order_product");//子表[表名]
    po2.setEntityName("GbuyOrderProduct");//子表[实体名]
    po2.setEntityPackage("order");//子表[包]
    po2.setFtlDescription("订单产品明细");//子表[描述]
    po2.setForeignKeys(new String[]{"GORDER_OBID","GO_ORDER_CODE"});//子表[外键:与主表关联外键]
    subTabParamIn.add(po2);
    codeParamEntityIn.setSubTabParam(subTabParamIn);

    //第三步：一对多(父子表)数据模型,代码生成
    CodeGenerateOneToMany.oneToManyCreate(subTabParamIn, codeParamEntityIn);
}
```

第二步:配置菜单，查看生成功能

编辑订单抬头

订单抬头

订单号

A001

客户

张小烟

联系人

小麦

区号

12

电话

15011000065

手机

15011000065

传真

邮箱

订单人数

0

总价(不含返)

返款

备注

审核状态

0

审核人ID

审核人

订单客户明细

订单产品明细

订单产品明细

添加

删除

序号	服务项目类型	产品名称	个数	单位	单价	小计	备注	团购服务项目
<input type="checkbox"/>	4	产品名称1						
<input type="checkbox"/>	2	产品名称1						
<input type="checkbox"/>	1	产品名称1						
<input type="checkbox"/>	3	产品名称1						

<

>

关闭

提交

第五章 查询过滤器-查询条件HQL生成器

现状分析: 项目开发的查询页面都会有很多查询条件, 开发追加查询条件的工作繁琐又很浪费时间。

这块工作量主要在: 页面加查询字段和后台代码逻辑判断, 追加查询条件;

目前JAVA持久层主流框架分析:

[1]. Hibatente 技术实现:

A. 页面追加查询字段;

B. 后台代码需加逻辑判断, 判断字段是否为空, 手工拼SQL追加查询条件;

[2]. IBATIS 技术实现:

A. 页面追加查询字段;

B. 后台不需写代码, 但是需在XML文件中追加该字段非空判断和查询条件;

特点: 常规功能的页面查询方式只能是"全匹配"和"模糊查询", 对于特殊的"包含查询"和"不匹配查询", 只能写特殊逻辑代码

查询条件SQL生成器 [实现原理]

根据页面传递到后台的参数, 动态判断字段是否为空, 自动拼SQL追加查询条件

特点: 实现了"模糊查询", "包含查询", "不匹配查询"等SQL匹配功能;

实现方法: 页面仅仅追加一个查询字段, 后台不需要写任何代码, 查询功能自动实现;

查询条件SQL生成器 [查询规则]

要求: 页面查询字段, 需跟Action中Page的字段对应一致, 后台不需写代码自动生成HQL, 追加查询条件;

默认生成的查询条件是全匹配;

查询匹配方式分类:

[1]. 全匹配查询: 查询数据没有特殊格式, 默认为全匹配查询

[2]. 模糊查询: 查询数据格式需加星号[*] 例如: {MD*/MD*/M*D*}

[3]. 包含查询: 查询数据格式采用逗号分隔[,] 例如: {01, 03} (含义: in('01', '03'))

[4]. 不匹配查询: 查询数据格式需要加叹号前缀[!] 例如: {!123} (含义: 不等于123)

特殊说明: 查询不为Null的方法=!null (大小写没关系)

查询不为空字符串的方法=!(只有一个叹号)

第六章 校验器-页面表单校验

页面字段校验方法:

1. 必须输入

A. INPUT (文本输入)

```
<input name="kfName" type="text" class="easyui-validatebox" data-  
options="required:true" missingMessage="请填写控房名称"/>
```

B. SELECT (下拉选择) (注意: 去掉class="easyui-validatebox")

```
<input name="hotelid" type="text" data-options="required:true" style="width:  
155px;"/>
```

2. 输入数字无小数点

```
<input name="doubleprice" class="easyui-numberbox" missingMessage="请填写双人单价名称"/>
```

3. 金额数字

```
<input name="doubleprice" class="easyui-numberbox" data-  
options="required:true,precision:2,groupSeparator:','" missingMessage="请填写双人单价名称"/>
```

4. 只读样式设置

```
class="inputread" readonly="readonly"
```

5. 输入框文本输入长度限制属性

```
maxlength="8"
```

6. 字段重复校验 (加属性: validType)

[1]. 例子:

酒店英文名

```
<input name="nameeng" validType="duplicate_hotel_nameeng"/>
```

[2]. validType列表

查看js:

main/include/js/validatebox-duplicate.js

[3]. validType校验类型, 追加方法

main/include/js/validatebox-duplicate.js

```
//-----
```

```
//第一部分:
```

```
duplicate_trnspname: {  
    validator: function (value, param) {  
        return duplicate_trnspname(value);  
    },  
    message: '交通方式中文名字, 在系统中已经存在!'  
}
```

```
//第二部分
```

```
//[交通方式]中文全称
```

```
var duplicate_trnspname = function(value){  
    var returnflag = ajax_check('TransportwayEntity', 'trnspname', value);  
    return returnflag;  
}
```

说明: TransportwayEntity: 页面字段对应的实体

trnspname: 页面字段

```
//-----
```

注意:

```
<input name="nameeng" validType="duplicate_hotel_nameeng"/>
```

该写法只适合于代码生成器生成的代码;

对于特殊页面的校验如下: (主要是将主键的ID名, 作为参数传递)

```
<input type="hidden" name="obid" id="obid_update"/>
```

```
<input name="hotelname" class="easyui-validatebox"
```

```

        validType="duplicate_hotelname['obid_update']" />
//-----
1. 时间格式化JS函数
dateFormatYMD(date)

2. 页面校验样式
easyui-combobox
easyui-datebox
easyui-datetimebox
easyui-validatebox
easyui-numberbox

金额格式: data-options="precision:2,groupSeparator:','"
必须输入: data-options="required:true"

注意:时间字段:missingMessage="请填写基本餐名称"这个必须去掉 加上: editable="false"(不允许
输入)

3. 不为空判断:StringUtils.isNotBlank

4. 下拉菜单禁止输入属性:
editable: false,
//-----

```

第七章 轻量级权限设计(菜单+按钮权限)

当前分析: 目前权限模块的设计, 模型很多也很成熟, 各种精细控制也很完善, 但同时因为权限设计的太精细化, 也产生一个问题: 用户系统操作越来越复杂;

实际上用到这么精细设计的项目也并不多。

一般项目只需控制到菜单级别, 即使需要控制到按钮权限, 也是少数几个页面, 所以我就想设计一个轻量级权限:

以菜单权限为主, 按钮权限为副, 达到用户操作简单, 条例清晰, 又能满足菜单和按钮权限控制!!

简述我的权限设计:

菜单和按钮权限采用分离设计, 二者没有关联关系而是相互独立。

(采用一张表, 存储菜单和权限, 设计权限类型字段加以区分)

权限分类:

[1]. 菜单权限 (权限类型: 1)

[2]. 按钮权限 (权限类型: 2)

权限控制说明:

[1]. 菜单权限:

用户只有配置了菜单权限, 才能看到对应的菜单;

[2]. 按钮权限:

A. 没有配置的按钮权限, 所有人都可以访问;

B. 按钮权限进行了配置, 则只有分配权限的人才能访问;

按钮权限自定义标签使用:

```
<%@ taglib uri="/btn-tags" prefix="btn"%>
```

```
<btn:access privilege="personAction!add.action">add();</btn:access>
```

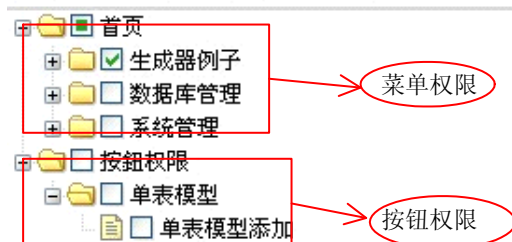
权限配置:

目前只有三层: 用户-角色-权限(菜单权限、按钮权限)

拥有权限

角色管理, 生成器例子, 数据库管理, druid监控, 首页, 字典管理, 单表模型, BUG管理, 用户管理, 按钮权限, 菜单管理, 系统管理, 按钮权限, 单表模型添加, 一对多模型

BUG管理, 生成器例子, 单表模型, 一对多模型



备注: 我的设计是以快速开发为宗旨, 简单清晰, 当然不能适合所有的系统, 有的系统功能需要非常精细化, 哪肯定得采用一些比较成熟的权限模型;

第八章 项目编码规范

【一】. 项目编码规范

- A. 项目编码格式为UTF-8(包括: java, jsp, css, js)
- B. service接口命名: *ServiceI
service实现命名: *ServiceImpl
entity命名: *Entity
page页面form命名: *Page
action命名: *Action
项目没有DAO SQL写在Service层
代码层次目录按照自动生成目录
- C. SQL文件目录和命名规范
 - (1). 所有SQL必须大写, 不允许用*, 全部替换为字段
 - (2). SQL文件根目录为:src\sun\sql, 子目录跟service必须保持一致
 - (3). SQL文件命名: [service名字]_[方法名字].sql
- D. 数据库表设计规范
 - (1). 主键字段为obid
 - (2). 每个字段必须加备注
- E. action中的方法
 - 配置菜单的方法: 以go开头 (其他方法不允许以go开头)
 - 触发业务逻辑的方法: 以do开头
 - 页面跳转的: 以to开头

F. Entity和数据库自定义命名规范

采用: 驼峰写法 (每个单词首字母小写、其他字母小写的写法) 转成中画线写法 (所有字母小写, 单词与单词之间以中画线隔开)

【二】. 详细说明:

[1]. SQL层讲解

- A. 项目没有DAO SQL写在Service层, 数据库取数和DB操作通过service层来实现
- B. 如果使用硬代码SQL, 一个方法对应一个SQL的话, 可以采用框架封装的方式来存储SQL文件 (表示采用命名规范来存储SQL)
 - 存储方式:
 - (1). 所有SQL必须小写, 不允许用*, 全部替换为字段
 - (2). SQL文件根目录为:src\sun\sql, 子目录跟service必须保持一致
 - (3). SQL文件命名: [service名字]_[方法名字].sql
 - 读取方式: `String sql = SqlUtil.getMethodSql(SqlUtil.getMethodUrl());`

SQL定位方法: `ctrl+shift+r` 参数: 方法名, 前面加*

[2]. action层页面数据封装

- 1. 页面列表数据方法: `datagrid`
- 2. 查询条件在ACTION 层`datagrid(pram)`方法执行前加
- 3. `getPagesFromEntitys` 封装查询数据; 举例: 将酒店ID改为NAME文本显示