JEECG 微云快速开发平台

开发指南

2015/12/03
www.jeecg.org
Jeecg 社区

目录

1. 前言 7
1.1. 技术背景 7
1.2. 平台介绍 7
1.3. 平台优势 8
1.4. 平台架构 8
1.5. 技术支持 10
2. JEECG 框架初探 11
2.1. 演示系统 11
2.2. 示例代码 13
3. JEECG 注意规则 14
4. 项目编码规范 15
4.1. 项目编码规范 15
4.2. 详细说明 16
4.3. 举例讲解命名规范 17
5. JEECG 目录结构 17
5.1. 配置文件目录结构 17
5.2. Java 源码目录结构 18
5.3. 单元测试代码结构 18
5.4. JSP 页面目录结构 19
6. JEECG 开发环境搭建 19
6.1. JAVA 环境配置 19
6.2. 开发环境搭建 21
6.2.1. 项目导入开发环境 21
6.2.2. 数据库初始化及数据源配置 25
6.2.3. 发布工程 26
6.2.4. 项目启动 27
6.3. Maven 开发环境搭建 28

7. 代码生成器.....32
7.1. 代码生成器配置.....32
7.2. 数据表创建.....33
7.3. 代码生成.....33
7.4. 代码生成扫描路径配置.....35
7.5. 功能测试.....36
7.5.1. 添加菜单并授权.....36
7.5.2. 功能测试.....37
7.6. 代码生成器使用规则.....38
7.6.1. 建表规范.....38
7.6.2. 页面生成规则.....39
7.7. 一对多的代码生成.....39
7.7.1. 一对多代码生成器使用.....39
7.7.2. 使用规范.....40
7.8. Online 表单开发.....40
7.8.1. 原理.....40
7.8.2. 使用.....41
7.9. Online 表单风格.....42
7.9.1. 介绍.....42
7.9.2. Java 包目录.....42
7.9.3. 实现原理.....44
7.9.4. 使用.....44
7.10. Online 报表配置.....46
7.10.1. 原理.....46
7.10.2. 使用.....47
7.11. Online 图表配置.....48
8. 查询 HQL 过滤器.....50
8.1. 数据过滤现状分析.....50
8.2. 查询条件 SQL 生成器.....50
8.2.1. 实现原理.....50

8.2.2. 查询规则.....51
8.2.3. 具体实现.....51
8.3. 查询过滤器高级特性.....52
8.3.1. 组合条件查询.....53
8.3.2. 字段范围查询.....53
8.3.3. 查询字段添加日期控件.....54
8.3.4. 日期字段的数据格式化.....54
8.3.5. 数据列表合计功能.....55
9. 标签中使用数据字典.....57
9.1. 标签参数.....57
9.2. 使用案例.....57
10. 表单校验组件 ValidForm.....59
10.1. Validform 使用入门.....59
10.2. 绑定附加属性.....59
10.3. 初始化参数说明.....64
10.4. Validform 对象[方法支持链式调用].....68
10.5. 调用外部插件.....74
10.6. Validform 的公用对象.....74
11. 基础用户权限.....75
11.1. 权限设计.....75
11.2. 权限设计目标.....76
11.3. 权限设计.....76
11.3.1. 数据表.....76
11.3.2. 页面菜单.....77
11.3.3. 页面控件权限.....77
11.3.4. 自定义页面控件权限.....81
11.3.5. 数据权限.....82
11.3.6. 数据权限系统上下文变量.....84
11.4. 菜单自动加载.....85
11.4.1. 背景.....85

11.4.2. 设计思路.....85
11.4.3. 具体实现.....85
11.4.4. 示例.....86
12. 多数据源.....87
12.1. 多数据源背景.....87
12.2. 多数据源设计原理.....87
12.3. 多数据源的使用与维护.....87
13. 国际化.....88
13.1. 国际化背景.....88
13.2. 国际化实现原理.....88
13.3. 国际化的使用场景.....89
13.4. 国际化语言维护.....92
13.5. lang_key 的命名规范.....92
13.6. 国际化后的问号处理.....93
13.7. 不想看到 lang_key.....93
14. 附录.....93
14.1. UI 库常用控件参考示例.....93
14.2. 开发技巧：采用 IFrame 打开页面.....94
14.3. 开发技巧：组合查询实现方法.....95
14.4. Formvalid 新增属性 tiptype 的使用.....96
14.5. 使用 toolbar 自定义 js 参数规则.....97
14.6. 表单字段重复校验方法.....98
15. MiniDao 介绍.....98
15.1. MiniDao 简介及特征.....98
15.2. 接口和 SQL 文件对应目录.....99
15.3. MiniDao 接口配置.....100
15.4. 测试代码.....101
15.5. 环境搭建.....101
15.5.1. MiniDao 与 Spring 集成.....101
15.5.2. MiniDao 与 Hibernate 集成.....103

16. Excel 导入导出.....103
16.1. 注解介绍.....103
16.2. Excel 导入.....105
16.3. Excel 导出.....106
16.4. Excel 模板导出.....108
16.4.1. 模板参数规则.....108
16.4.2. 模板导出.....108

1. 前言

1.1. 技术背景

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！

1.2. 平台介绍

JEECG［J2EE Code Generation］是一款基于代码生成器的微信快速开发平台，采用代码生成+手工 MERGE 半智能开发模式， 可以帮助解决 Java 项目60%的重复工作，让开发更多关注业务逻辑。既能快速提高开发效率，帮助公司节省人力成本，同时又不失扩展性和灵活性。

JEECG 宗旨：简单功能由代码生成器生成使用；复杂业务采用表单自定义，业务流程使用工作流来实现、扩展出任务接口，供开发编写业务逻辑。 实现了流程任务节点和任务接口的灵活配置，既保证了公司流程的保密行，又减少了开发人员的工作量。

JEECG 采用 SpringMVC+MiniDao 持久层+UI 标签组件作为基础架构，采用面向声明的开发模式,基于泛型方式编写极少代码即可实现复杂的数据展示、数据编辑、表单处理等功能，再配合代码生成器的使用将 JavaEE 的开发效率提高6倍以上，可以将代码减少60%以上。

JEECG V3.0 版本四大技术点： **1.代码生成器 2. UI 标签组件 3.在线流程设计 4. 系统日志记录**。

- **代码生成器**：支持多种数据模型,根据表生成对应的 Entity,Service,Dao,Controller,JSP 等,增删改查功能生成直接使用
- **UI 标签组件**：针对 WEB UI 进行标准封装，页面统一采用 UI 标签实现功能：数据 datagrid,表单校验,Popup,Tab 等， 实现 JSP 页面零 JS，开发维护非常高效
- **在线流程设计**：在线流程定义，采用开源 Activiti 流程引擎，实现在线画流程,自定义表单,表单挂接,业务流转， 流程监控， 流程跟踪， 流程委托等
- **系统日志记录**：系统操作日志详细记录，帮助运维人员进行系统分析和故障排查。

最新版本及特性

- JEECG V3.0,经过了专业压力测试,性能测试，保证后台数据的准确性和页面访问速度
- 支持多种浏览器：IE, 火狐, Google 等浏览器访问速度都很快
- 支持数据库：Mysql,Oracle10g 等
- 基础权限：用户，角色，菜单权限，按钮权限，数据权限
- 智能报表集成：简易的图像报表工具和 Excel 导入导出
- Web 容器测试通过的有 Jetty 和 Tomcat6
- 待推出功能：分布式部署，云计算，移动平台开发，规则引擎，代码生成器(eclipse 插件)
- 要求 JDK1.6+

1.3. 平台优势

✓ 采用主流开源技术框架，容易上手；代码生成器依赖性低,很方便的扩展能力，可完全实现二次开发；

✓ 开发效率很高,代码生成器支持多种数据模型：单表数据模型、单表自关联模型和一对多(父子表)数据模型，代码生成功能直接使用；

✓ 查询 SQL 过滤器，后台不需要写代码，页面追加查询字段，查询功能自动实现

✓ 页面校验自动生成(必须输入、数字校验、金额校验、时间控件等)；

✓ 基础的用户权限：菜单，按钮权限，角色

✓ 常用共通封装，各种工具类(定时任务,短信接口,邮件发送,Excel 导出等),基本满足 80%项目需求

✓ 集成简易报表工具,图像报表和数据导出非常方便,可极其方便的生成 pdf、excel、word 等报表；

✓ 集成工作流引擎 Activiti5，并实现了只需在页面配置流程转向，可极大的简化工作流的开发；用 Activiti5 的流程设计器画出流程走向，一个工作流基本就完成了，只需进行流程的配置或者写很少量的 java 代码

1.4. 平台架构

架构技术： Spring MVC+Hibernate4+UI 快速开发库+Spring JDBC+Highcharts 图形报表+Jquery+Ehcache。

设计思想：零配置（ 约定大于配置）

各技术点说明

[1]代码生成器

代码生成器用于生成规范的后台代码+统一风格的前台页面+表单校验。

单表模型，单表自关联模型和一对多(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\邮箱\手机号\QQ 号等等);

C.支持 Oracle/Mysql/Postgres 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].查询条件过滤器

页面加查询条件，只需配置页面对应的查询属性，后台不需要写任何逻辑判断，JEECG 查询过滤器机制会自动追加查询条件至 HQL

[3].UI 快速开发库

UI 快速开发库，针对 WEB UI 进行标准封装，页面统一采用 UI 标签实现功能：数据 datagrid、表单校验、Popup、Tab、选择器、自动补全功能等，实现 JSP 页面零 JS，开发维护非常高效

[4].智能工作流

在线流程定义，采用开源 Activiti 流程引擎，实现在线画流程,自定义表单,表单挂接,业务流转，流程监控，流程跟踪，流程委托等

[5].表单 Form 校验组件

前台页面字段校验采用 Validform 表单检验组件，支持手机号码、邮箱帐号、IP 地址等常用的数值验证及字长验证

[6].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

[7].基础用户权限

权限功能：用户、角色、权限（菜单权限+按钮权限）

[8].Ehcache 缓存机制

Ehcache 缓存自定义标签（永久缓存/临时缓存）

[9].报表封装

Excel 简易导出工具类+Highcharts 图形报表

[10].MiniDao 数据持久层

(1) O/R mapping 不用设置 xml，零配置便于维护

(2) 不需要了解 JDBC 的知识

(3) SQL 语句和 java 代码的分离

(4) 可以自动生成 SQL 语句

(5) 接口和实现分离，不用写持久层代码，用户只需写接口，以及某些接口方法对应的 sql 它会通过 AOP 自动生成实现类

(6) 支持自动事务处理和手动事务处理

(7) 支持与 hibernate 轻量级无缝集成

(8) MiniDao 吸收了 Hibernate+mybatis 的优势，支持实体维护和 SQL 分离

(9) SQL 支持脚本语言

[11].安全的事务回滚机制+安全的数据乐观锁机制

[12].系统日志记录，便于问题追踪

1.5. 技术支持

作者：张代浩

邮箱： jeecg@sina.com

技术论坛： www.jeecg.org

技术支持： JEECG 开源社区

说明:提供偿服务(公司培训,技术支持,解决使用 JEECG 平台过程中出现的疑难问题)

2. JEECG 框架初探

2.1. 演示系统

打开浏览器，输入 JEECG 演示环境地址：http://demo.jeecg.org:8090/，可以看到如图 2-1 所示的登录界面。

图 2-1 演示系统登录界面

点击【登陆】按钮，进入演示系统的主界面，如图 2-2 所示。

图 2-2 演示系统主界面

在 JEECG 演示系统中的功能模块包括自定义表单、ONLINE 开发、消息推送、系统监控、统计管理、系统管理、常用示例等。其中，用户管理、流程设计器的界面截图如图 2-3 和图 2-4 所示。

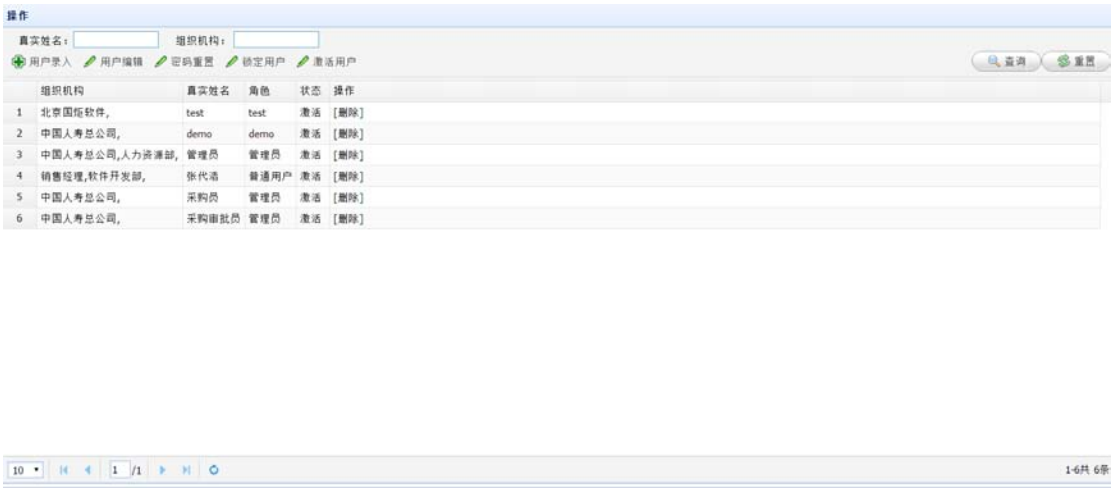


图 2-3 用户管理界面

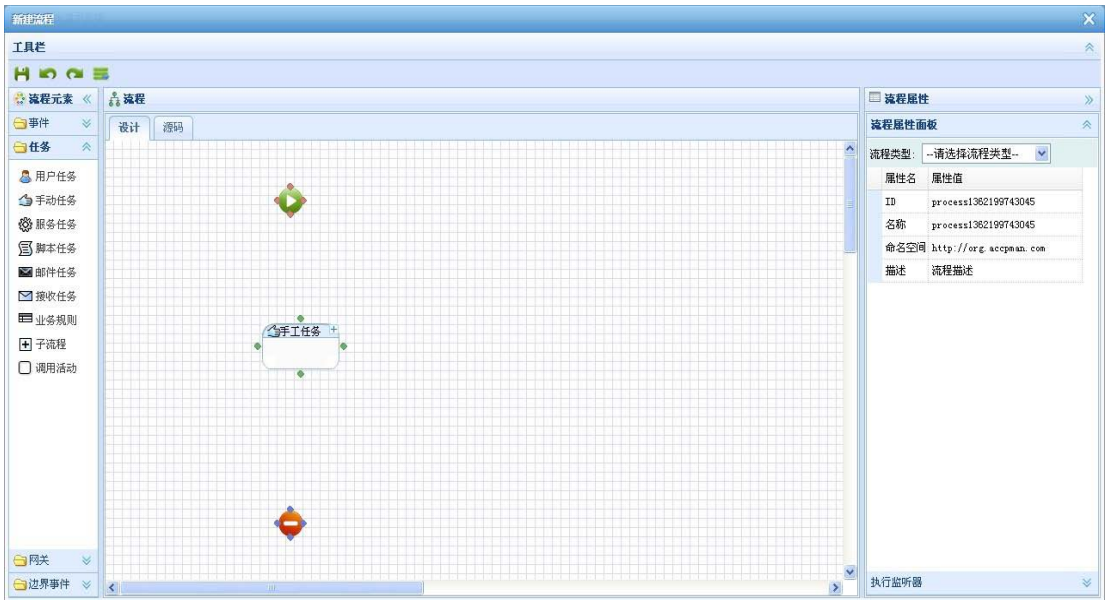


图 2-4 流程设计器

2.2. 示例代码

用户管理中的用户列表和用户维护所用的 jsp 页面代码分别如图 2-5和图 2-6所示。

```
<t:datagrid name="userList" title="common.operation" actionUrl="userController.do?datagrid"
    fit="true" fitColumns="true" idField="id" queryMode="group" sortName="createDate" sortOrder="desc">
    <t:dgCol title="common.id" field="id" hidden="true"></t:dgCol>
    <t:dgCol title="common.username" sortable="false" field="userName" query="true"></t:dgCol>
    <%--update-start--Author:zhangguoming  Date:20140827 for：通过用户对象的关联属性值获取组织机构名称（多对多关联）--%>
    <%--<t:dgCol title="common.department" field="TSDepart_id" query="true" replace="${departsReplace}"></t:dgCol>--%>
    <t:dgCol title="common.department" sortable="false" field="userOrgList.tsDepart.departname" query="false"></t:dgCol>
    <%--update-end--Author:zhangguoming  Date:20140827 for：通过用户对象的关联属性值获取组织机构名称（多对多关联）--%>
    <t:dgCol title="common.real.name" field="realName" query="true"></t:dgCol>
    <t:dgCol title="common.role" field="userKey" ></t:dgCol>
    <t:dgCol title="common.createby" field="createBy" hidden="true"></t:dgCol>
    <t:dgCol title="common.createtime" field="createDate" formatter="yyyy-MM-dd" hidden="true"></t:dgCol>
    <t:dgCol title="common.updateby" field="updateBy" hidden="true"></t:dgCol>
    <t:dgCol title="common.updatetime" field="updateDate" formatter="yyyy-MM-dd" hidden="true"></t:dgCol>
    <t:dgCol title="common.status" sortable="true" field="status" replace="common.active_1,common.inactive_0,super.admin_-1" ></t:dgCol>
    <t:dgCol title="common.operation" field="opt" width="100"></t:dgCol>
    <t:dgDelOpt title="common.delete" url="userController.do?del&id={id}&userName={userName}" />
    <t:dgToolBar title="common.add.param" langArg="common.user" icon="icon-add" url="userController.do?addorupdate" funname="add"></t:dgToolBar>
    <t:dgToolBar title="common.edit.param" langArg="common.user" icon="icon-edit" url="userController.do?addorupdate" funname="update"></t:dgToolBar>
    <t:dgToolBar title="common.password.reset" icon="icon-edit" url="userController.do?changepasswordforuser" funname="update"></t:dgToolBar>
    <t:dgToolBar title="common.lock.user" icon="icon-edit" url="userController.do?lock&lockvalue=0" funname="lockObj"></t:dgToolBar>
    <t:dgToolBar title="common.unlock.user" icon="icon-edit" url="userController.do?lock&lockvalue=1" funname="unlockObj"></t:dgToolBar>

</t:datagrid>
```

图 2-5 列表页面代码

图 2-6 用户管理页面代码

3. JEECG 注意规则

1. 列表页面，datagrid 的 name 属性不允许存在重复的，否则页面显示白板：

```
<t:dategrid name="jeecgDemoList" title="开发 DEMO 列表"
actionUrl="jeecgDemoController.do?datagrid" idField="id" fit="true">
```

2. 表单验证采用 Validform

3. 时间控件采用 my97，不要使用 easyui 的时间控件，因为加载效率慢

4. 上传文件使用规则

使用 SWFUpload 插件上传，可同时传多个文件，需要安装 Falsh 软件，

```
<div class="form">
 <t:upload name="fiels" buttonText="上传文件"
  uploader="systemController.do?saveFiles" extend="office"
  id="file_upload"  formData="documentTitle"></t:upload>
</div>
<div class="form" id="filediv" style="height:50px">
</div>
```

extend：office 表示可上传 offices 格式后缀的文件，pic 表示可上传图片格式后缀的文件

上传文件大小：未限制

5. 流程配置表单后，业务申请必须重新创建

6. jsp 代码注释规范，采用隐式注释不能用显式注释，不然标签还是能读到 ：

```
隐式注释：<%-- --%>
显式注释：<!-- -->
```

7. 表单布局两种风格：1.table 2.div

1.table 例如：jeecg/demo/jeecgDemo/jeecgDemo.jsp

2.div 例如：webpage/system/role/role.jsp

8. postgres 数据库建表规范

字段名字大小写有区别，请注意

9. 菜单采用 frame 方式打开方法

```
dataSourceController.do?goDruid&isIframe
```

10. 页面组件 ID 命名规范

[1].dategrid 组件 name

```
<t:dategrid name="userMe"
```

[2].组合查询 DIV

```
<div id="userMetb"
```

[3].查询按钮对应的 js 方法

```
<a href="#" class="easyui-linkbutton" iconCls="icon-search"
onclick="userMesearch()">查询</a>
```

4. 项目编码规范

4.1. 项目编码规范

1. 项目编码格式为 UTF-8(包括:java,jsp,css,js)
2. sevice 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：*Controller

   项目没有 DAO, SQL 写在 Service 层

   代码层次目录按照自动生成目录
3. 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:sql 跟接口目录 Service 是一个目录;

例如: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

4.数据库表设计规范

(1).主键字段为 id

(2).每个字段必须加备注

5.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

6.Entity 和数据库自定命名规范

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）

4.2. 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].Controller 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

4.3. 举例讲解命名规范

例如：表名：jeecg_sys_demo

第一部分：代码文件命名规则如下：

首先:表名采用驼峰写法转换为 Java 代码使用单词　jeecg_sys_demo => JeecgSysDemo

[1].control 命名 :JeecgSysDemoControl

[2].Servlice 命名:JeecgSysDemoServiceI/JeecgSysDemoServiceImpl

[3].JSP 命名：jeecg-sys-demo.jsp(表单页面)

jeecg-sys-demo-list.jsp（列表页面）

jeecg-sys-demo-*.jsp(新增表单页面例如：detail)

[4].control 中方法命名：

页面触发业务方法以 do*开头

页面跳转方法以 go*开头

（方法标签注释需和方法名保持一致）

[5].page/entity 字段定义必须是对象类型

int --> Integer

5. JEECG 目录结构

5.1. 配置文件目录结构

JEECG 中的配置文件目录结构如图 11-1 所示。

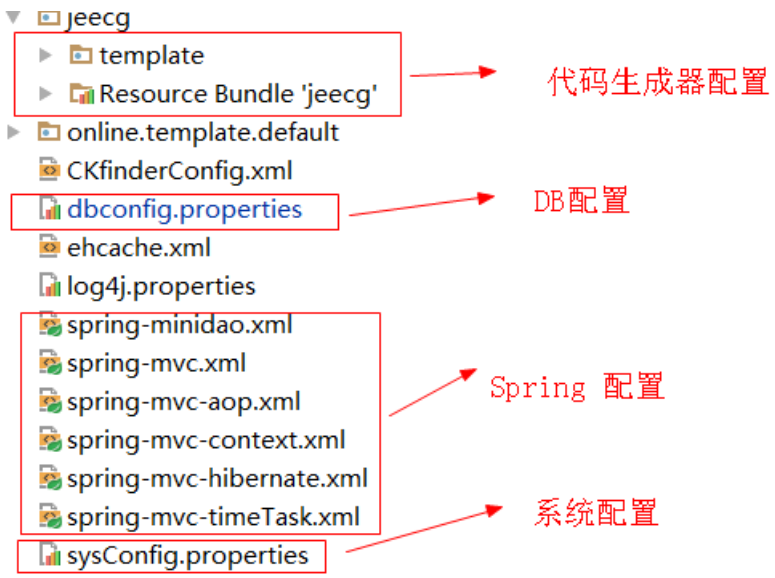


图 11-1 JEECG 配置文件目录结构

5.2. Java 源码目录结构

JEECG 中的 Java 源码目录结构如图 11-2 所示。

图 11-2 Java 源码目录结构

5.3. 单元测试代码结构

JEECG 中的单元测试代码存放的目录结构如图 11-3 所示。

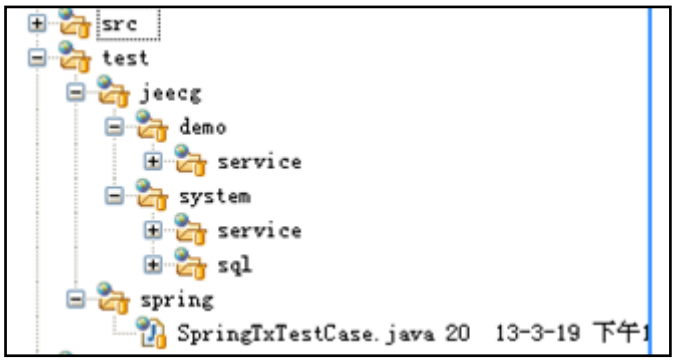


图 11-3 单元测试代码结构

5.4. JSP 页面目录结构

JEECG 中的 JSP 文件存放的目录结构如图 11-4 所示。

图 11-4 JSP 页面目录结构

6. JEECG 开发环境搭建

JEECG 推荐的开发环境为 Myeclipse8.5/Eclipse3.7+JDK1.6+Tomcat6.0

6.1. JAVA 环境配置

通过 Oracle 的官方地址下载 JDK 开发包：

http://www.oracle.com/technetwork/java/javase/downloads/index.html

推荐下载最新的 Java SE 6 版本,目前最新的 Java SE 6 SDK 版本为 Update 43，如图 3-1 所示。

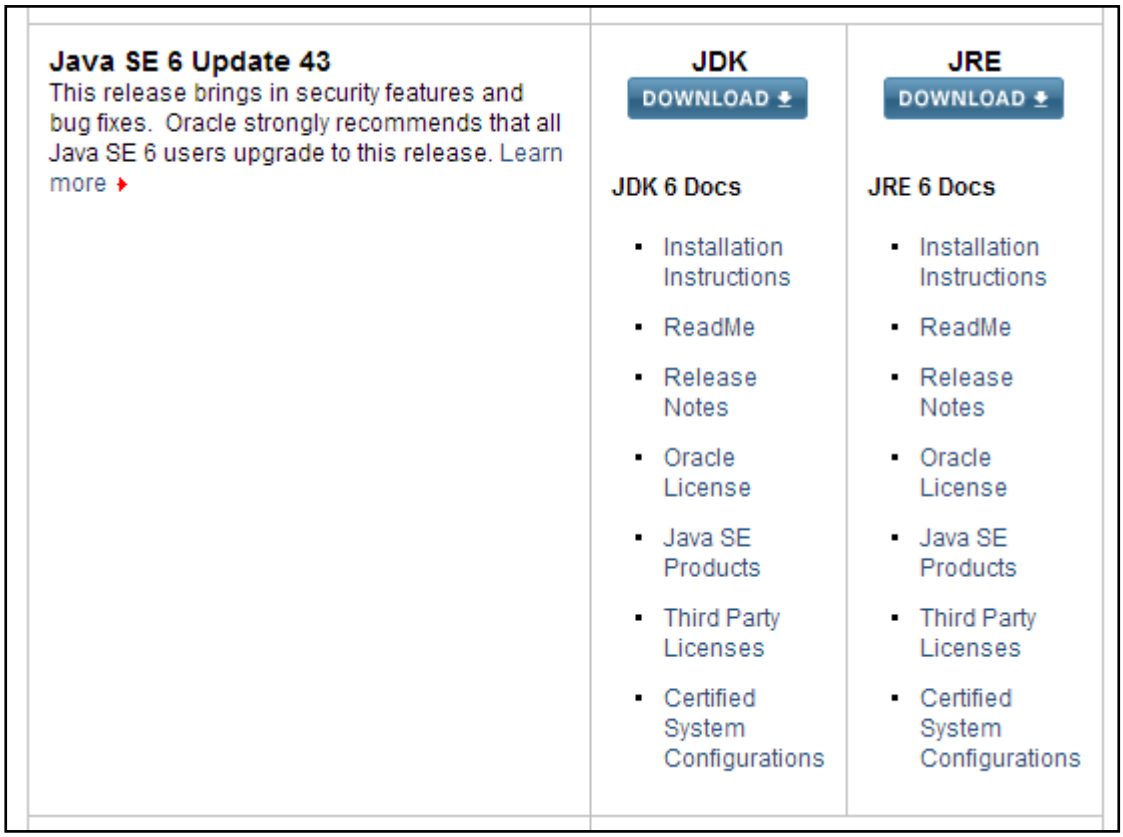
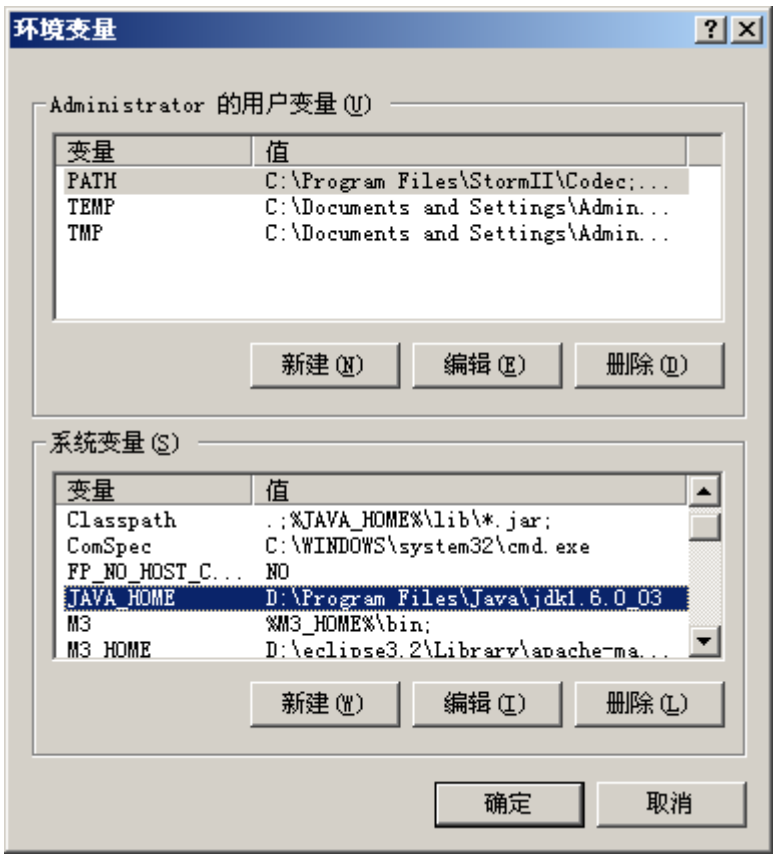


图 3-1 最新 SDK 下载链接

将下载的开发包安装到本机非中文路径的目录中，如本机的 D:\Program Files\Java\jdk1.6.0_43。

安装完 JDK 之后，需要配置本机的环境变量如下：

```
JAVA_HOME= D:\Program Files\Java\jdk1.6.0_43
PATH=%JAVA_HOME%/bin;
Classpath=.;%JAVA_HOME%\lib;
```



6.2. 开发环境搭建

6.2.1. 项目导入开发环境

JEECG 的目前最新版本为 V3.4.2，源代码地址：

https://code.csdn.net/zhangdaiscott/JEECG

将下载到的源代码解压到本地磁盘，通过 MyEclipse 的 Import->Existing Projects into Workspace 功能将源代码导入到 MyEclipse 开发环境中，在项目导入之后，需要对编译环境进行检查，如果编译环境中缺少 J2EE 支持的话，需要手动加上，如图 3-2 所示。

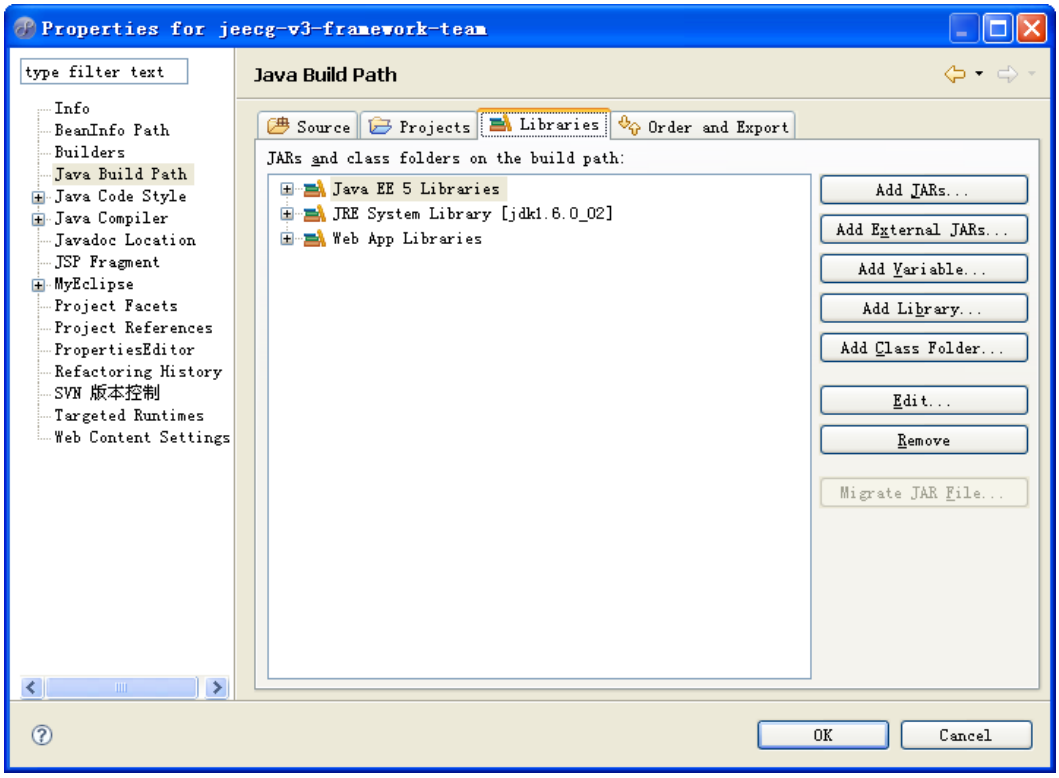


图 3-2 Myeclipse 编译环境

如果你使用的是 eclipse，而不是 MyEclipse 做为开发工具，将项目导入到 eclipse 之后，需要为项目添加 eclipse 的 WTP 项目支持。

在导入的工程上右键 Propertis->Project Facets,选择 Convert to faceted from…，如图 3-3 所示。

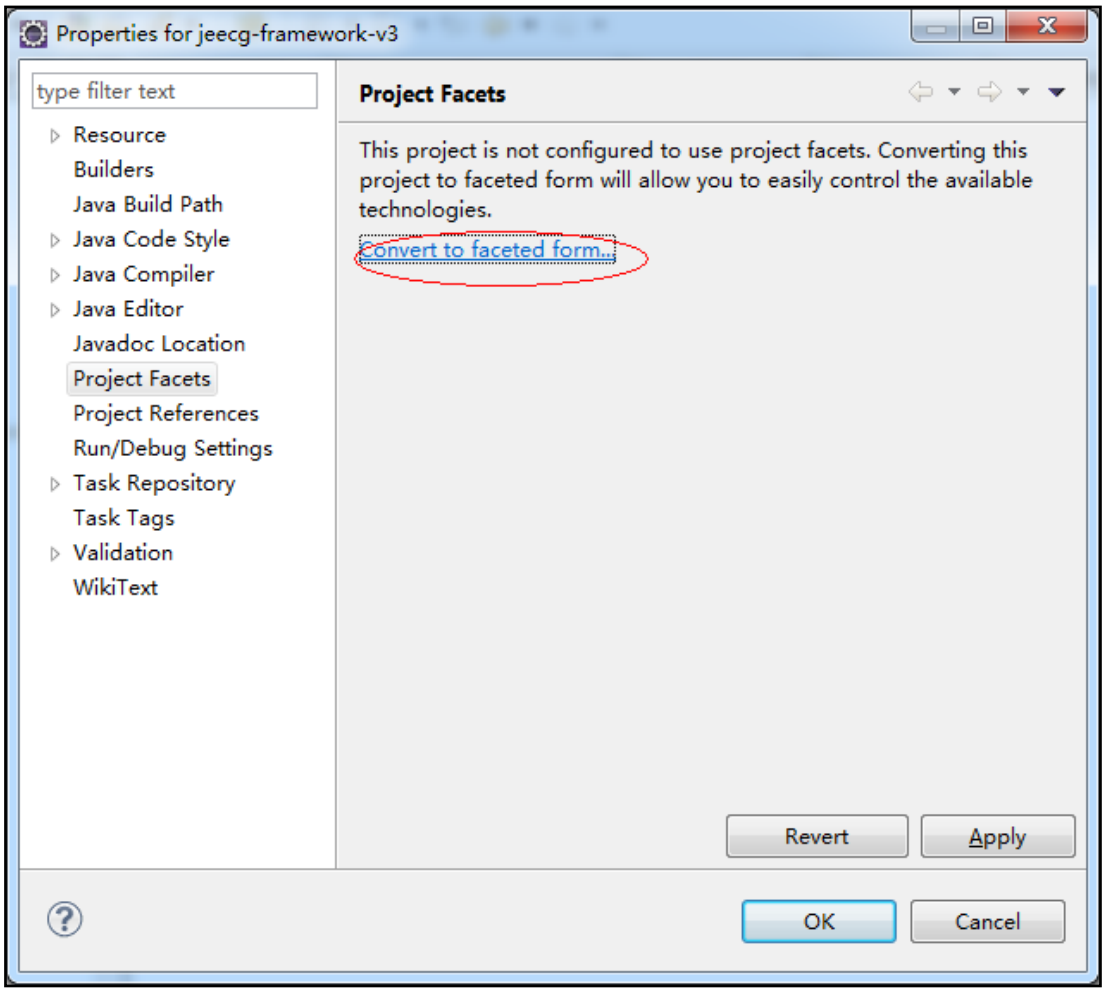


图 3-3 为工程添加 WTP 项目支持

在打开的界面中，勾选“Dynamic Web Module”和“Java”，分别选择其 Version 为 2.5 和 1.6，并为项目添加 Tomcat 的运行时支持，如图 3-4 所示。

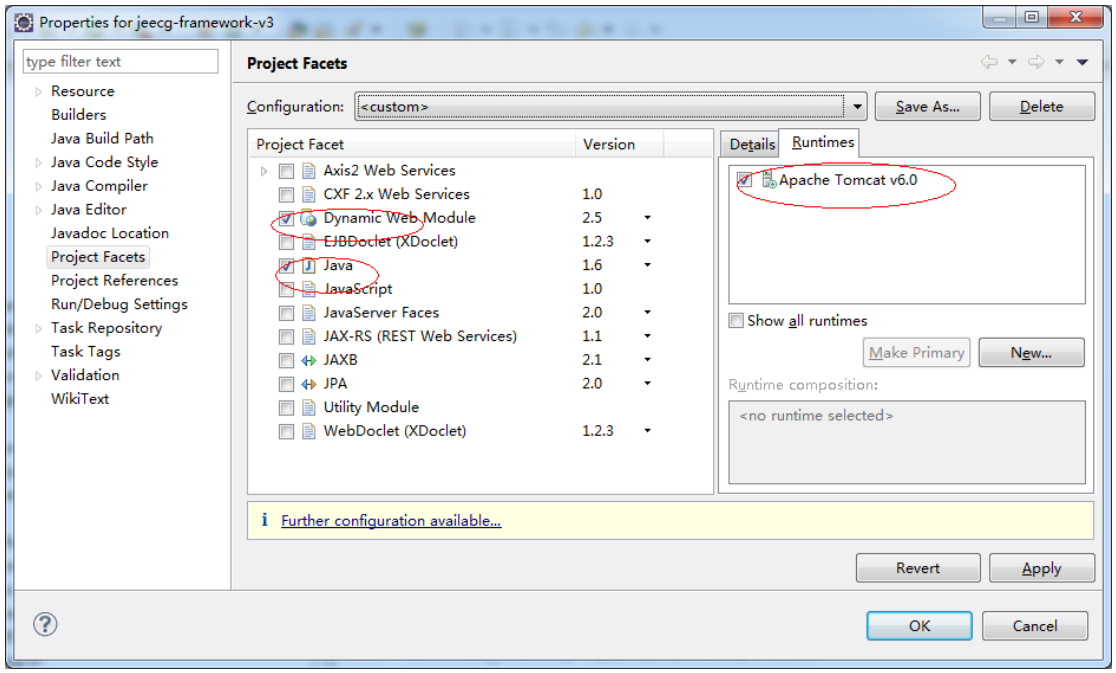


图 3-4 Project Facets 及运行时选择

选择完 Project Facets 之后，点击界面下方的“Futher configuration avaliable”链接，在弹出的新窗口中，填写 Content directory 的值为“WebRoot”，并将 Generate web.xml deployment descriptor 前面的复选框取消勾选，并为“Content root”命名为合适的值，如图 3-5 所示。

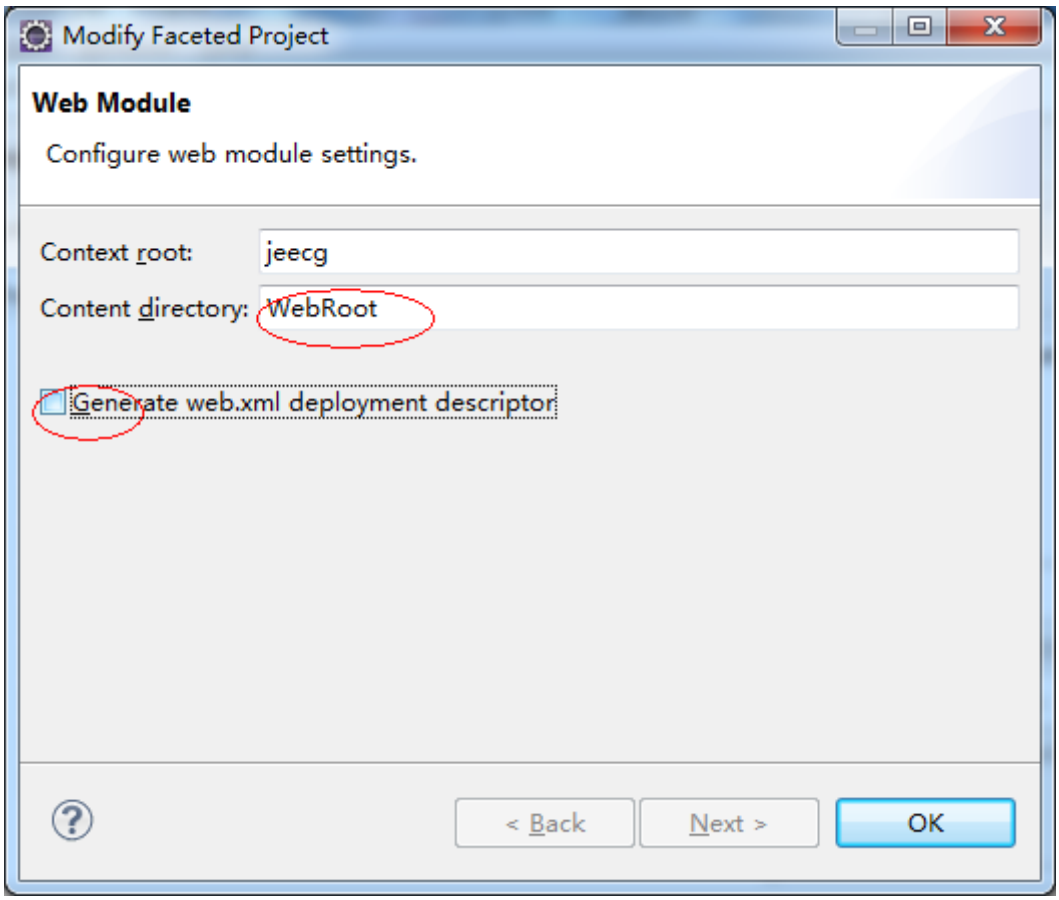


图 3-5 Web Module 设置

确定之后，完成对工程的 web 化支持。

6.2.2. 数据库初始化及数据源配置

路径：/jeecg-v3-simple/resources/dbconfig.properties 的文件是 JEECG 的数据库配置文件，Oracle/Mysql/Postgres 等数据库的连接配置在此文件中都有示例，以下仅以 mysql 为例做配置讲解。

在 mysql 数据库中新建一编码为 UTF8 的数据库 jeecg。

配置好数据库连接：

```
#MySQL
hibernate.dialect=org.hibernate.dialect.MySQLDialect
validationQuery.sqlserver=SELECT 1
jdbc.url.jeecg=jdbc:mysql://localhost:3306/jeecg?useUnicode=true&characterEncoding=UTF-8
jdbc.username.jeecg=root
jdbc.password.jeecg=root
jdbc.dbType=mysql
```

由于 JEECG 采用的是 hibernat 注解方式管理表，故不需再去手动创建表。当初次使用本框架时需要创建表，所以需要配置 create 来自动创建表，初次之后请选用其他配置属性，如 update，否则每次启动工程时都会重新建表，表中的数据也会丢失。

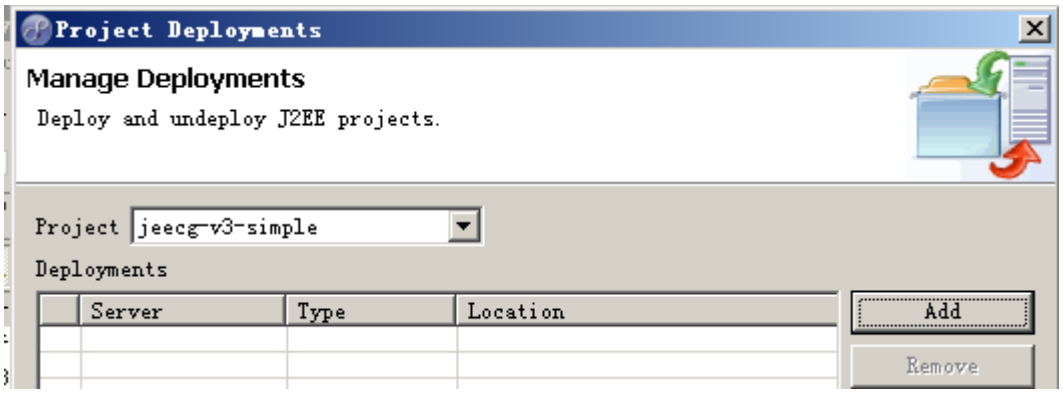
```
32
33#更新|创建|验证数据库表结构|不作改变  默认update(create,validate,none)
34hibernate.hbm2ddl.auto=create
```

6.2.3. 发布工程

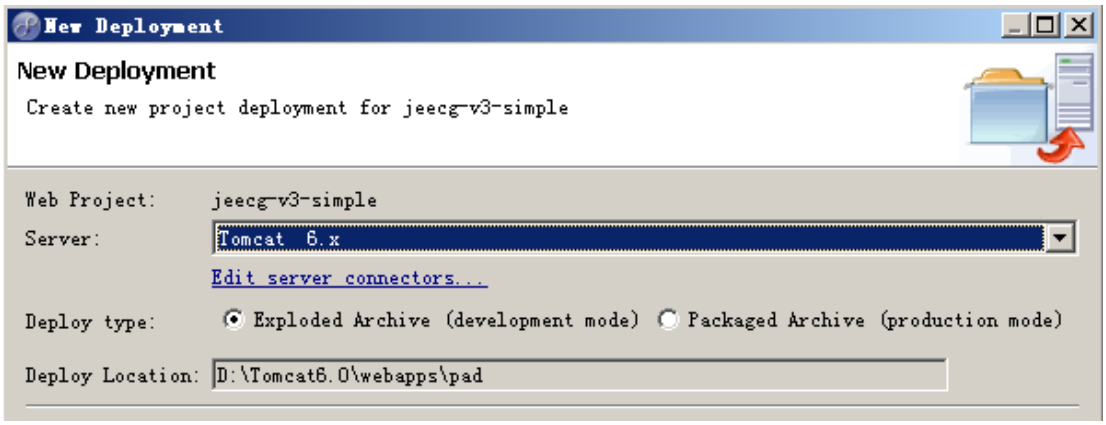
配置好 JDK 环境并创建数据库后，在 MyEclipse 中将工程发布到 tomcat 下，发布操作步骤如下，

1、点击菜单栏中的 Deploy MyEclipse J2ee Project to Server

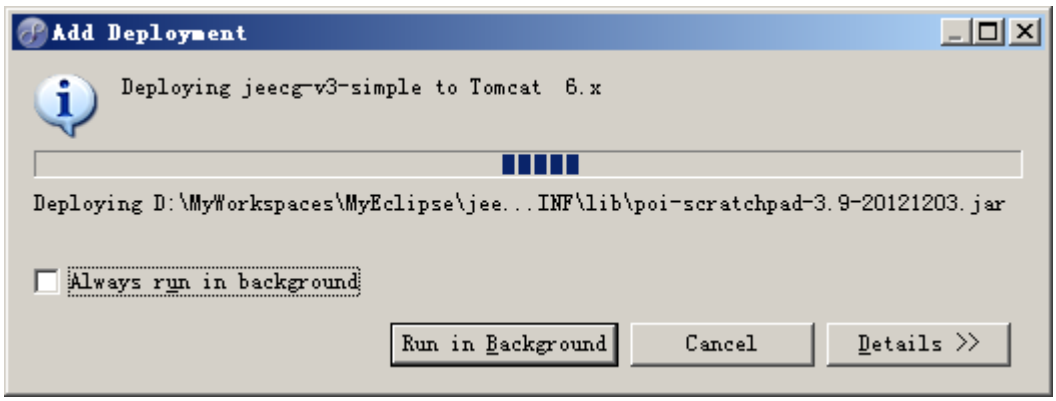
2、在弹出的窗口 project 处选择导入的 JEECG 工程，点击右侧的 Add 按钮



3、在弹出的窗口 server 处选择 tomcat，然后点击下方的 Finish 按钮，回到上一窗口，点 OK 按钮



4 耐心等待发布过程，成功布置后，启动工程



启动项目会自动建表，此时使用 show tables;命令查看数据库中的表，可以看到如图 3-6 的结果，已经有 45 张表入库。

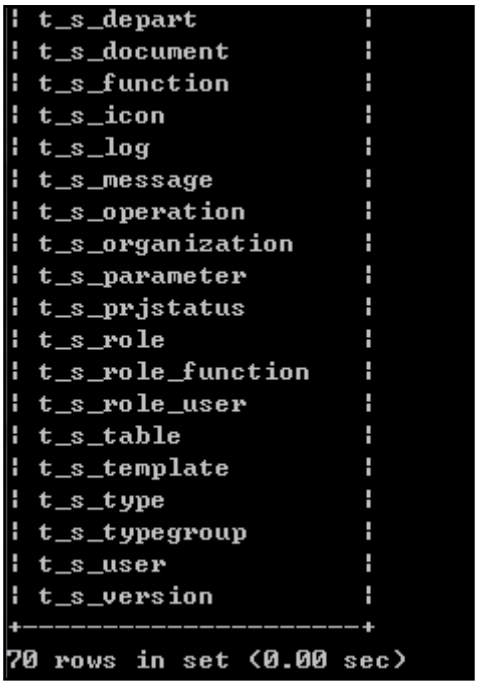


图 3-6 数据库初始化

6.2.4. 项目启动

Tomcat 启动成功以后，在浏览器地址栏中输入 http://localhost:8080/jeecg/，打开的界面如图 3-9 所示。

图 3-7 项目登录页面

初始化数据:点击是否初始化数据,进行数据初始化

登陆:输入用户名密码 admin/123456，登陆进入主界面，如图 3-10 所示。

图 3-8 项目主界面

至此，开发环境搭建完成。

6.3. Maven 开发环境搭建

在搭建 jeecg 的 maven 开发环境之前，需要先配置好本机的 maven 环境，并在 eclipse 中安装好 m2eclipse 插件。

1. maven 版本的工程目录，代码结构如图 3-11 所示。

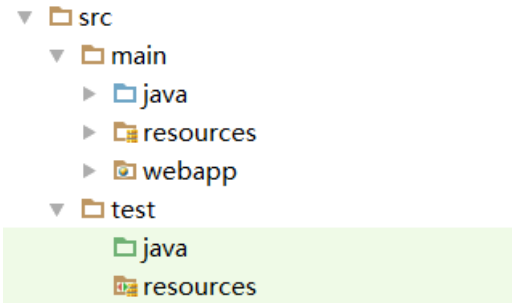


图 3-9 JEECG-MAVEN 工程目录结构

2. 针对本机开发环境（这里以 eclipse 为例），调整依赖包和项目属性

首先在工程上右键->properties，在 builders 选项卡中删除掉不存在或不需要的 builders，如图 3-12 所示。

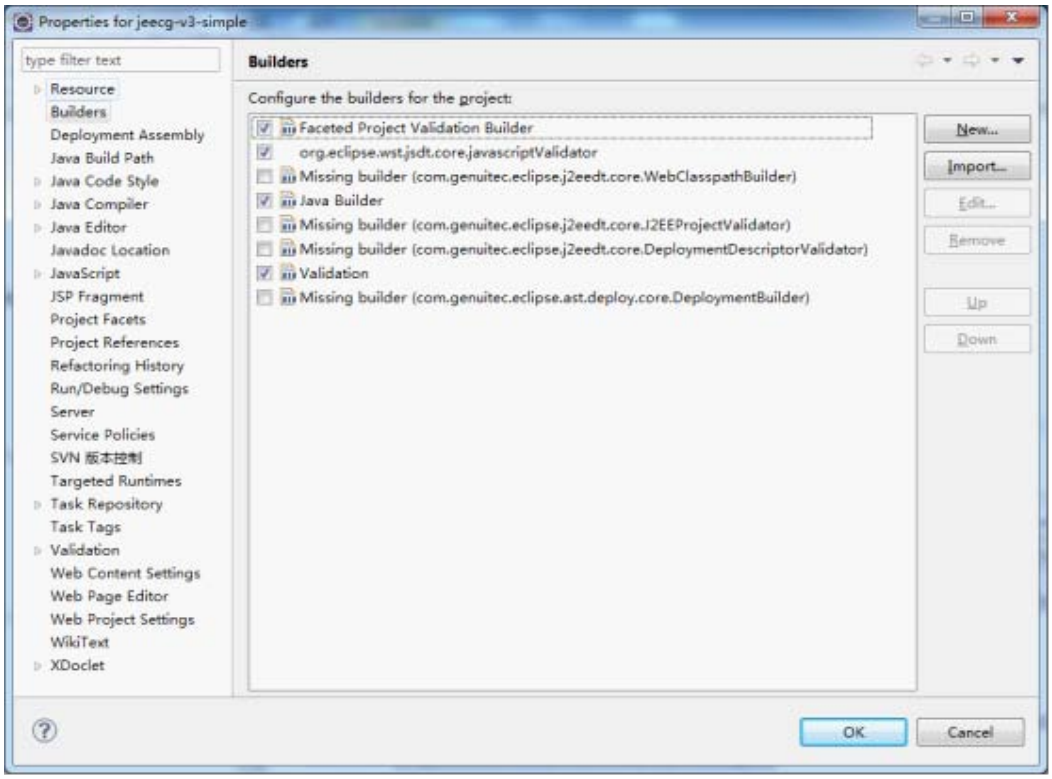


图 3-10 builders 设置

然后进入 Java Build Path 选项卡->Libraries，将除 jre 依赖之外的所有依赖包删除，如图 3-13 所示。

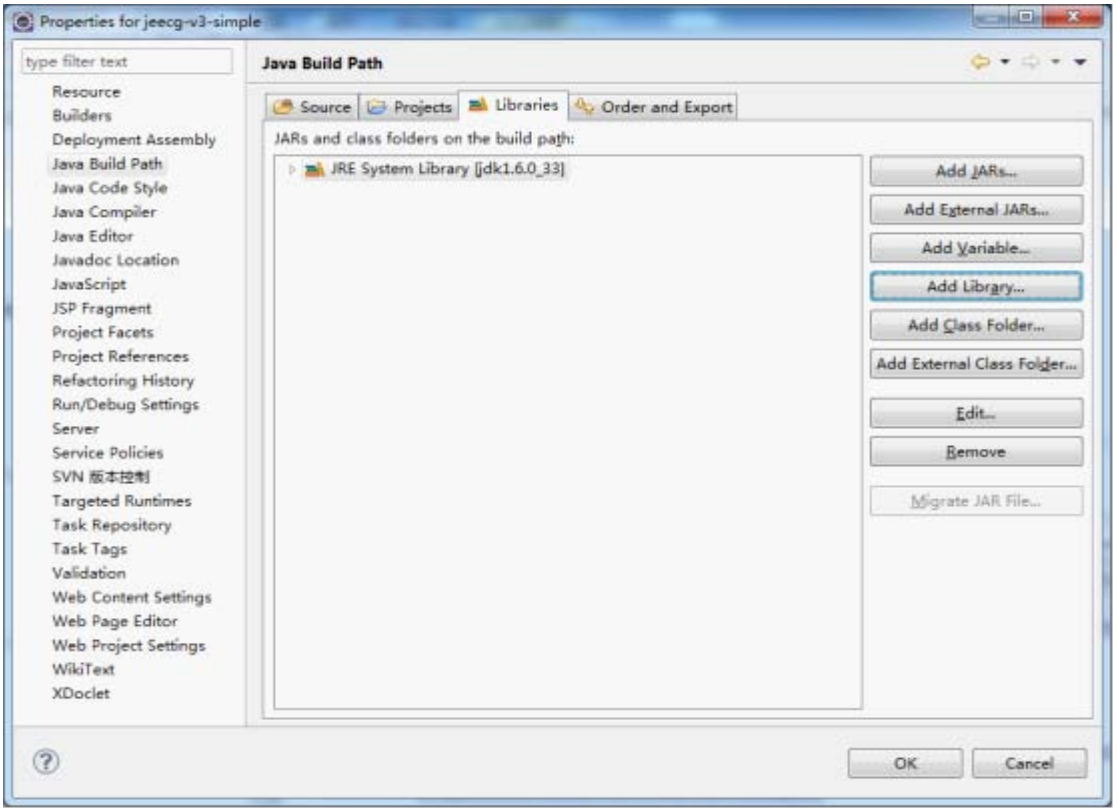


图 3-11 依赖库设置

3. 对工程增加 Maven 依赖

在工程目录上面右键->Maven->Enable Dependency Management。此时，maven 插件会把 maven 依赖包加入到工程中，目录结构如图 3-14 所示。

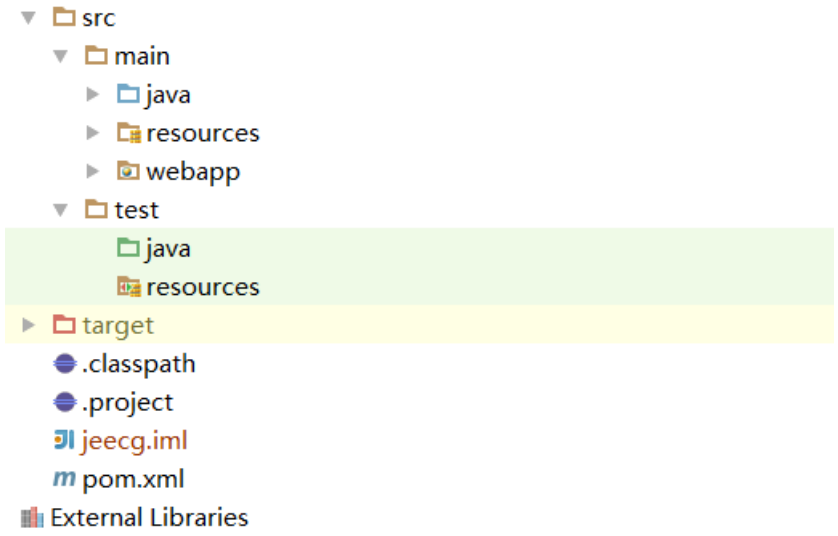


图 3-12 增加 Maven 功能的工程目录

4. 运行项目

在工程目录上面右键->Run As->7 Maven Build，在弹出的运行设置的 Goals 中填写”tomcat:run”，如果在运行时，不需要跑单元测试程序，可以把 Skip Test 给勾选上，如图 3-15 所示。

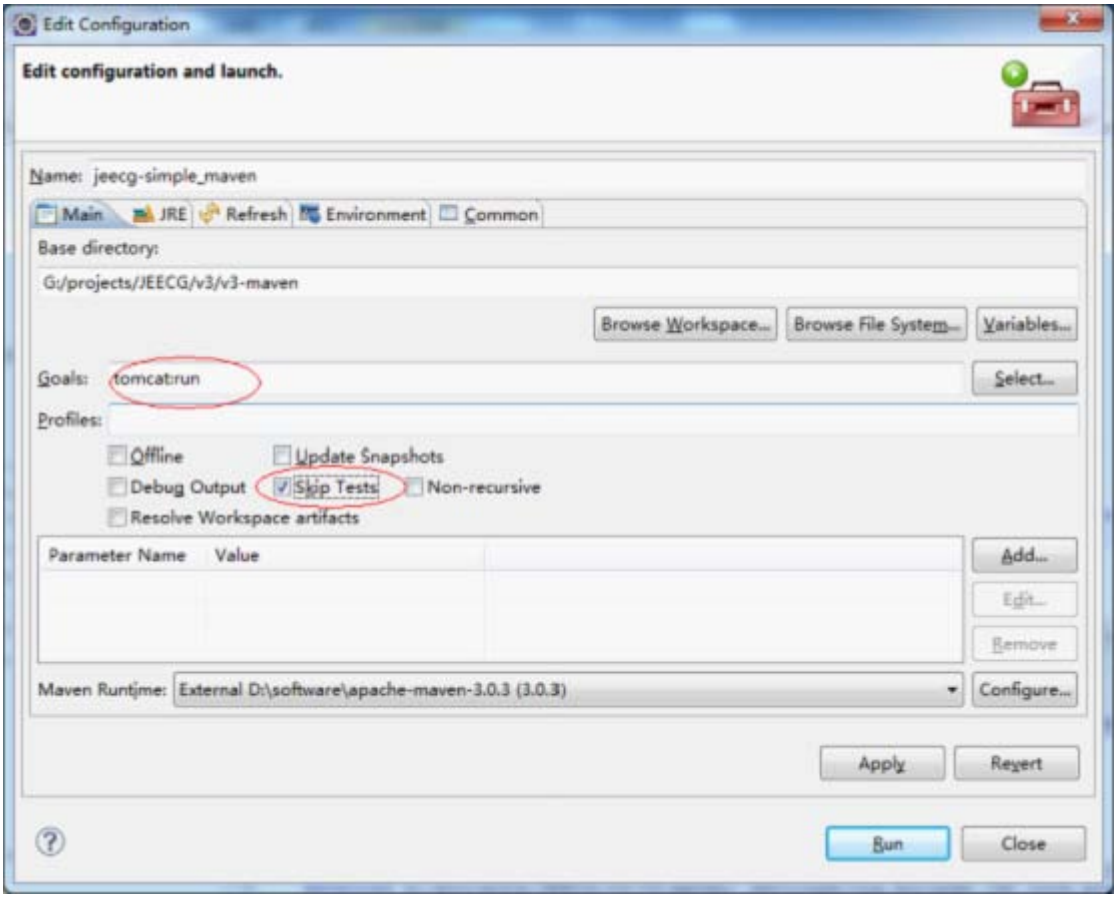


图 3-13 运行项目

项目运行之后的访问地址为：http://localhost:8080/jeecg。

5. 项目打包

在工程上面右键->Run As->Maven Package，打包完成之后的 war 包位于 target/jeecg.war，如图 3-16 所示。

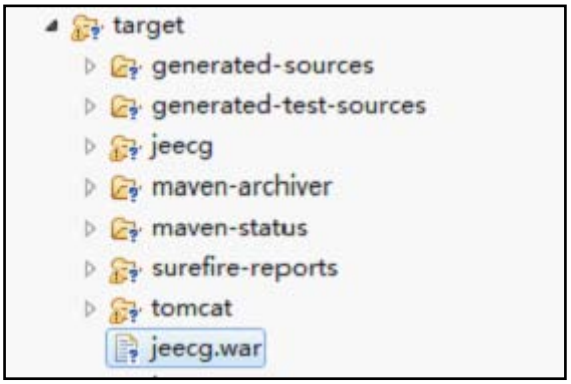


图 3-14 项目打包

7. 代码生成器

本章通过一个实际的示例来讲解 JEECG 代码生成器的使用

7.1. 代码生成器配置

代码生成器有两个配置文件：一个用于数据源的配置，一个用于代码生成器的参数配置。这两个配置文件分别是 resources/jeecg 目录的 jeecg_database.properties 和 jeecg_config.properties。

1. **jeecg_database.properties**：数据源配置文件，为保证能顺利生成代码，将文件中的数据源配置修改为/resources/dbconfig.properties 文件中同样的配置
2. **jeecg_config.properties**：生成器参数配置文件，各参数说明如表 4-1 所示：

表 4-1 代码生成器参数说明

参数	参数说明	默认值	取值范围
source_root_package	Source floders on build path (JAVA 文件的根目录)	src	
webroot_package	WEB 应用文件的根目录（例如：jsp）	WebRoot	
bussi_package	业务包（举例：比如 ERP 中的一个大的模块销售模块目录） 特点：支持多级目录例如 [com.sys]	Demo	
templatepath	代码生成器使用的模板文件目录	jeecg/template	
system_encoding	项目编码	utf-8	
jeecg_generate_table_id	自定义主键命名	id	目前表主键只能命名 ID
jeecg_ui_search_filed_num	配置代码生成器生成的 JSP 页面，默认前几个字段生成查询条件	1	
jeecg_filed_convert	数据库表字段转换为实体字段是采用原生态，还是采用驼峰写法转换	true	true/false
ui_filter_fields	根据过滤器自动在表中生成创建人、创建时间、修改人、修改时间等值（映射的字段参照“表 4 2	create_date,create_by,create_name,update_date,up	

	建表模板”）	date_by,update_name	
project_path	项目路径		

7.2. 数据表创建

现在有一张员工表 person，其建表 SQL 为：

```
CREATE TABLE `person` (
   `ID` varchar(32) 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  (`ID`)
) 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。

将建表 SQL 在数据库管理器里面执行，完成对 person 表的创建。

7.3. 代码生成

运行“/src/test/JeecgOneGUI.java”文件，打开代码生成器并输入相应的参数如图 4-1 所示。

图 4-1 员工信息维护的代码生成器

执行【生成】之后，可以在源代码目录 src 中（即 jeecg_config.properties 文件中的参数 jeecg_config.properties 指向的包）看到新生成的 java 代码文件，如图 4-2 所示。

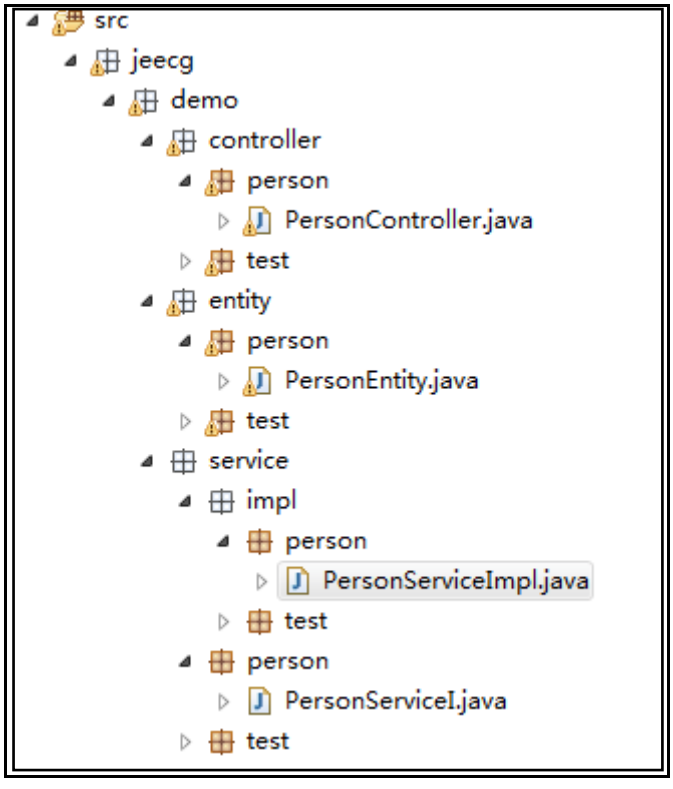


图 4-2 生成的 java 文件

同样地，可以在 WebRoot/webpage 中看到新生成的 jsp 页面，如图 4-3 所示。

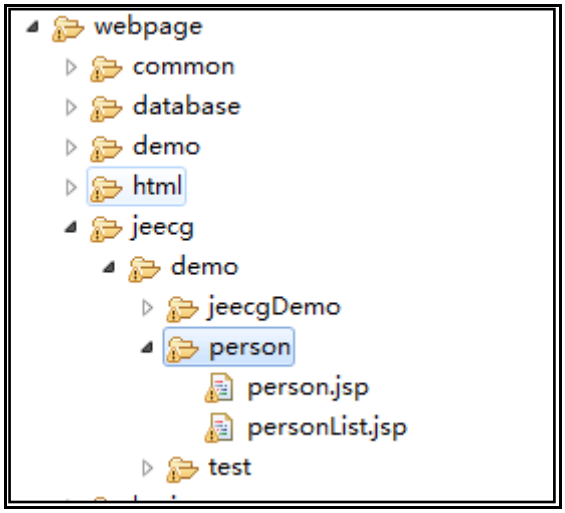


图 4-3 生成的 JSP 文件

生成代码结构说明

1. 添加和修改页面在一个 JSP 页面中
2. service 层接口和实现都继承父类

7.4. 代码生成扫描路径配置

用代码生成器生成代码后，需要进行相关配置配置，扫描注入 control、service、entity 等；

详细操作步骤如下：

1. control 扫描配置，在 spring-mvc.xml 文件里面

```
<!-- 加载controller的时候,不加载service,因为此时事务并未生效,若此时加载了service,那么事物无法对service进行拦截 -->
<context:component-scan base-package="jeecg.*,com.chenhb.*">
      <context:exclude-filter type="annotation" expression="org.springframework.stereotype.Service" />
</context:component-scan>
```

配置属于自己的controller

2. Service 扫描路径配置, spring-mvc-hibernate.xml

```
<!-- 加载service,此时类排除类controller,因为controller已经spring-mvc中加载过了 -->
 <context:component-scan base-package="jeecg.*">
       <context:exclude-filter type="annotation" expression="org.springframework.st
</context:component-scan>
```

加载自己的service

```
  <context:component-scan base-package="com.chenhb.*">
       <context:exclude-filter type="annotation" expression="org.springframework.st
 </context:component-scan>
```

3. 实体 Entity 扫描路径配置, spring-mvc-hibernate.xml

7.5. 功能测试

7.5.1. 添加菜单并授权

重新启动 Tomcat，进入系统主界面->系统管理->菜单管理，点击菜单录入，添加员工管理菜单，如图 4-4 所示。

菜单地址内容为:类映射名.do?方法名，如 personController.do?person

图 4-4 员工管理的菜单添加

菜单添加完成之后，需要将该菜单分配给管理员角色，重新登录系统后，可以在系统管理模块下看到子菜单【员工管理】，如图 4-5 所示。

图 4-5 新增的员工管理菜单项

7.5.2. 功能测试

点击菜单项【员工管理】，打开员工管理的主界面如图 4-6 所示。

图 4-6 员工管理主界面

点击【录入】按钮，在弹出的对话框中录入员工基本信息，如图 4-7 所示。

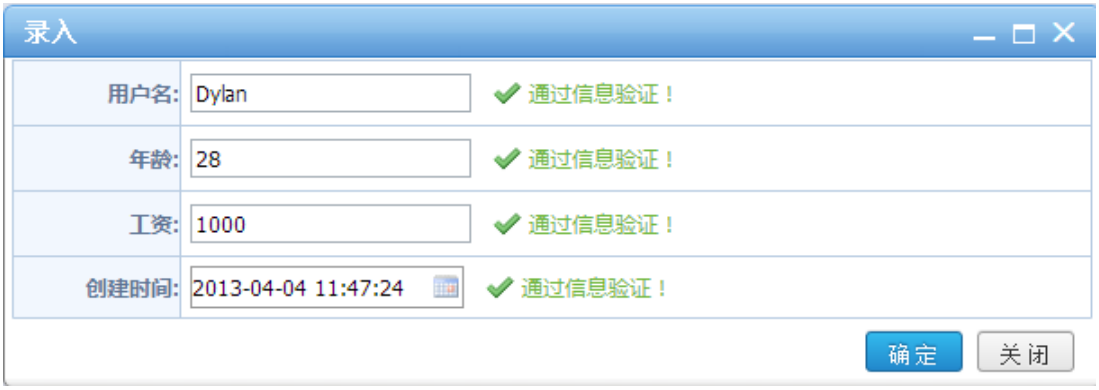


图 4-7 员工信息录入

点击确定按钮，对信息进行保存，此时可以在用户列表中看到新录入的信息，同时在数据库中也可以看到数据被保存入库，如图 4-8 所示。

	ID	NAME	AGE	SALARY	createDt
1	4028ba813dd31cc8013dd328f0400020	Dylan	28	1000.00	2013-04-04 11:47:24

图 4-8 信息被正确保存入库

7.6. 代码生成器使用规则

7.6.1. 建表规范

1. 表必须有唯一主键：ID（字符类型 32 位）

备注： 主键采用 UUID 方式生成

主键支持自定义，修改 jeecg_config.properties 的参数[jeecg_generate_table_id]即可;

2. 如需使用框架自动生成表创建人，创建时间等，必须字段参见“表 4 2 建表模板”

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

注：请按照建表模板表 4-2 来创建新表，模板表中原有的字段，生成器会过滤不在页面生成。

表 4-2 建表模板

字段名	类型	长度	备注	主键
ID	varchar	36	主键	TURE
CREATE_BY	varchar	36	创建人	
CREATE_NAME	varchar	32	创建人名字	
CREATE_DATE	datetime	0	创建时间	
UPDATE_BY	varchar	36	修改人	

UPDATE_NAME	varchar	32	修改人名字
UPDATE_DATE	datetime	0	修改时间
DELFLAG	int	2	删除标记
DEL_DATE	datetime	0	删除时间

7.6.2. 页面生成规则

说明：JSP 页面字段的文本内容，取表字段的注释前 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

7.7. 一对多的代码生成

7.7.1. 一对多代码生成器使用

单表的代码生成器入口类是 test.JeecgOneGUI；

一对多的代码生成器入口类是 test.JeecgOneToMainUtil;

一对多的代码生成器使用示例：

```
//第一步：设置主表
CodeParamEntity codeParamEntityIn = new CodeParamEntity();
codeParamEntityIn.setTableName("jeecg_order_main");//主表[表名]
codeParamEntityIn.setEntityName("Demo4ManyKey");  //主表[实体名]
codeParamEntityIn.setEntityPackage("jeecg");  //主表[包名]
codeParamEntityIn.setFtlDescription("订单主数据");  //主表[描述]

//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("jeecg_order_custom");//子表[表名]
po.setEntityName("DemoMany4CustomKey");//子表[实体名]
po.setEntityPackage("jeecg");            //子表[包]
po.setFtlDescription("订单客户明细");  //子表[描述]
po.setForeignKeys(new String[]{"GORDER_ID","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("jeecg_order_product");                    //子表[表名]
po2.setEntityName("DemoMany4ProductKey");                              //子表[实体名]
po2.setEntityPackage("jeecg");                                          //子表[包]
po2.setFtlDescription("订单产品明细");                                  //子表[描述]
po2.setForeignKeys(new String[]{"GORDER_ID","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);
```

7.7.2. 使用规范

1. 目前代码生成器默认的主键生成策略为 UUID
2. 主表和子表的目录最好保持一致
3. 子表和主表的外键规则如下：

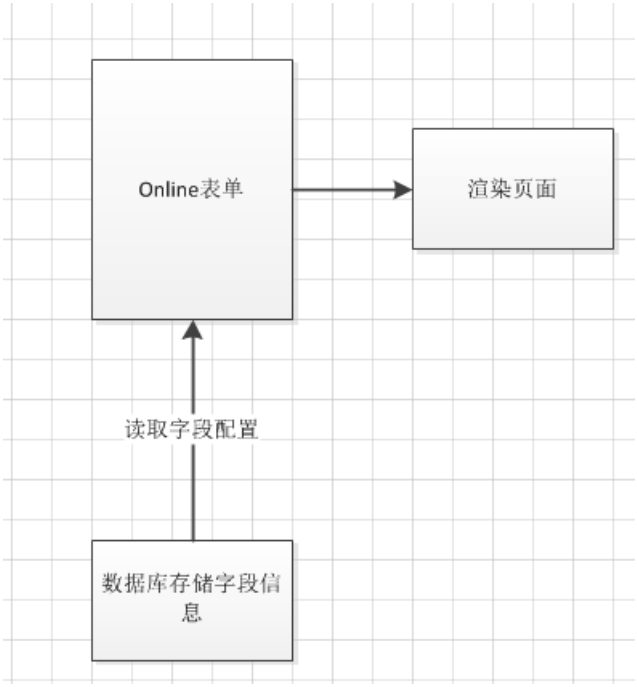
   a)主表和子表的外键字段名字，必须相同（除主键 ID 外）

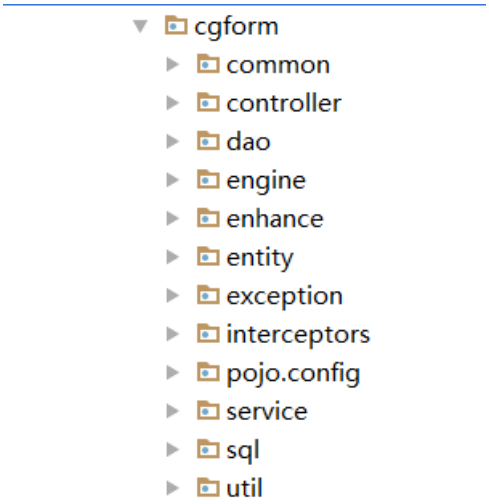
   b)子表引用主表主键 ID 作为外键，外键字段必须以_ID 结尾

7.8. Online 表单开发

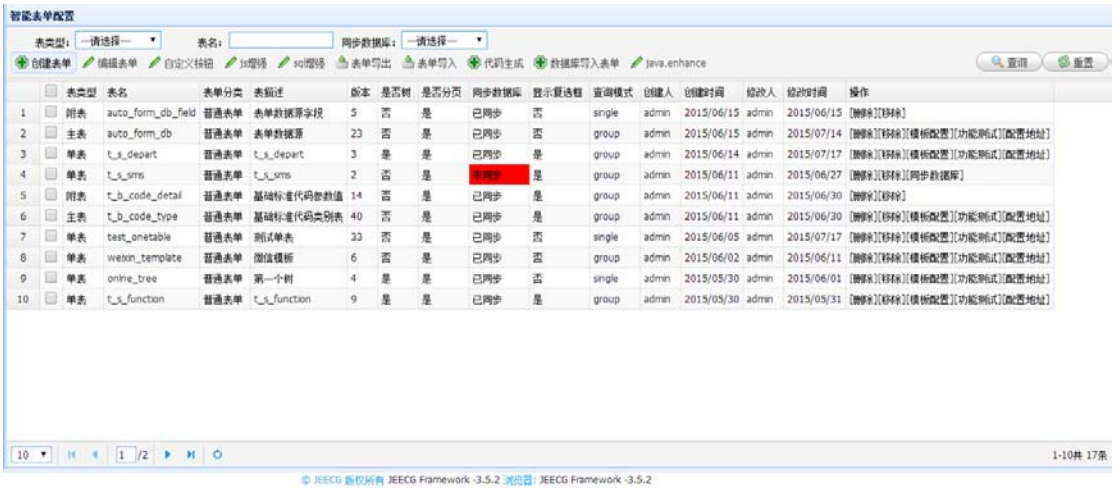
7.8.1. 原理

系统提供 online 表单开发，将表单配置信息存入数据库，即可通过 freemaker 引擎在线渲染，也可生成代码进行二次开发，方便灵活。


Online表单
渲染页面
读取字段配置
数据库存储字段信息

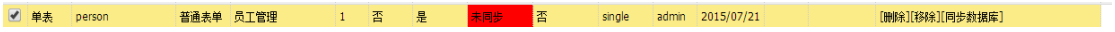

cgform
common
controller
dao
engine
enhance
entity
exception
interceptors
pojo.config
service
sql
util

7.8.2. 使用


智能表单配置

创建表单

online 表单开发中 各属性配置与代码生成器使用规则类似。其中，【表单风格】可使用自己的模板（online 表单风格功能）。



新创建的表单配置信息存在数据库里，但数据库中无 person 表。使用【同步数据库】功能，将执行建表 sql 语句，可在数据库中生成配置的 person 表，同步后

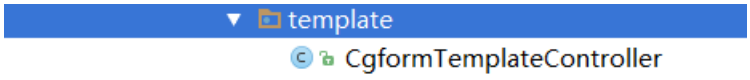
此时，数据库中已存在相应的表，点击【功能测试】可进行增删改查操作，此功能通过对 freemaker 模板进行渲染实现，如需添加其他复杂业务，可生成代码进行二次开发。

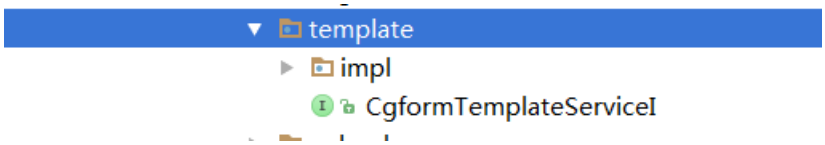
7.9. Online 表单风格

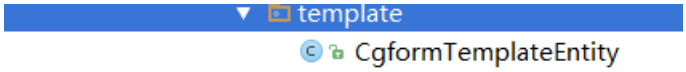
7.9.1. 介绍

在线表单风格可让用户上传自己的模板并立刻展示。

7.9.2. Java 包目录

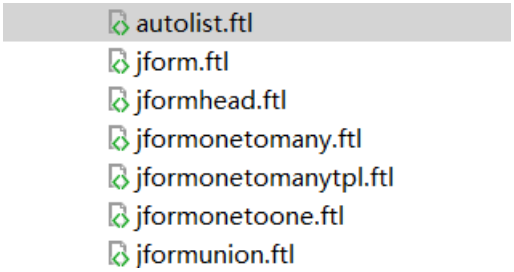



template
impl
CgformTemplateServiceI

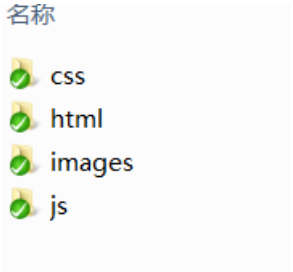


7.9.3. 实现原理

编写基于freemarker的展示模板；系统通过freemarker引擎渲染模板，产生html页面。



上传压缩包中各式如下：



页面展示模板请放入html文件夹

列表展示页面部分代码：

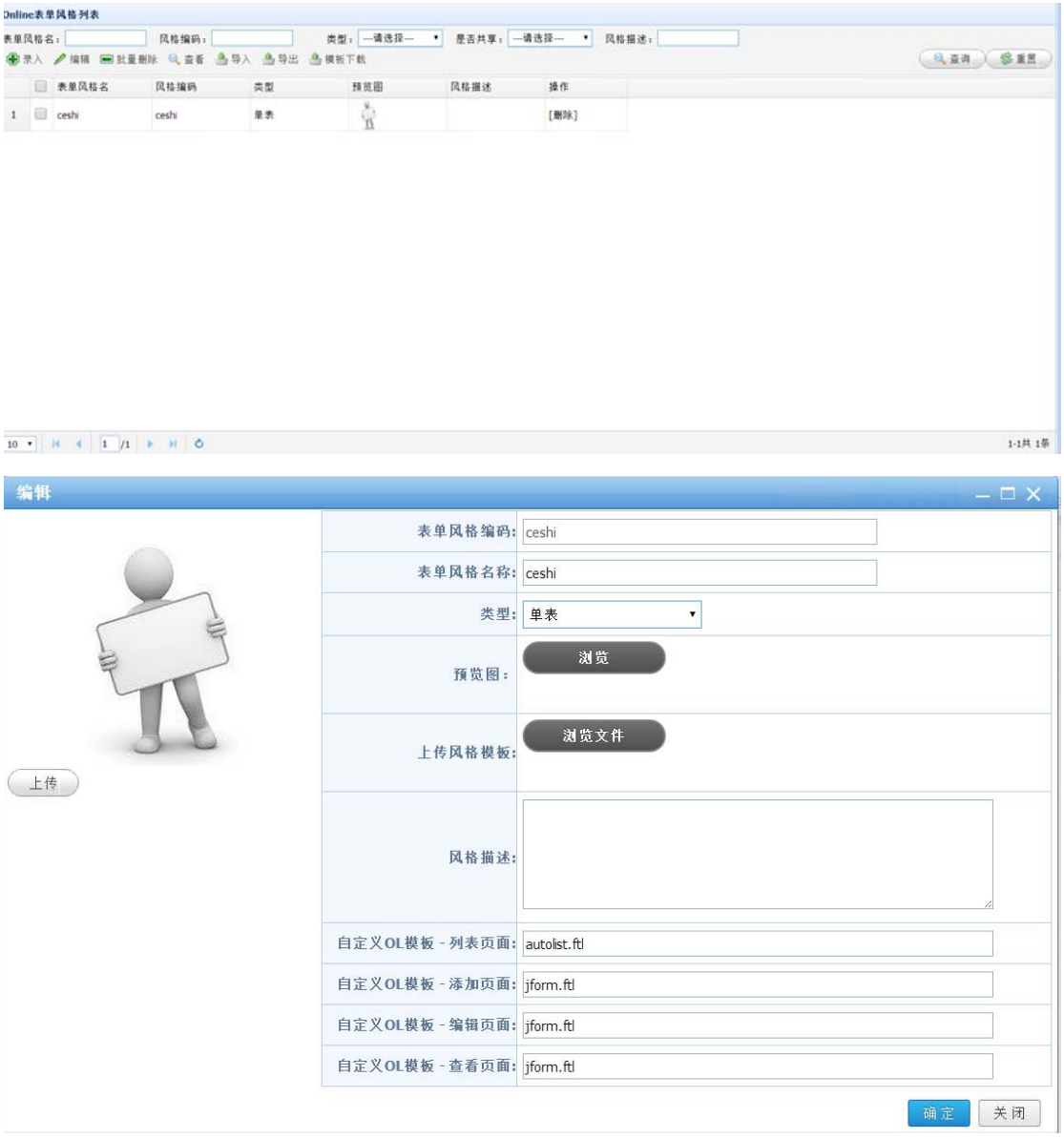
```
${config_iframe}

<!--update-start--Author:luobaoli  Date:20150703 for: 将本文档中所有href="#"修改为href="javascript:void(0)",避免re
<script type="text/javascript">
/**
*表单的高度,表单的宽度
**/
var ${config_id}Fw = 700,${config_id}Fh = 400;

$(function(){
    $.get("cgFormHeadController.do?checkIsExit&checkIsTableCreate&name=${config_id}",
    function(data){
        data = $.parseJSON(data);
        if(data.success){
            createDataGrid${config_id}();
        }else{
            alertTip('表:<span style="color:red;">${config_id}</span>还没有生成,请到表单配置生成表');
        }
    });
});

function createDataGrid${config_id}(){
    var initUrl = 'cgAutoListController.do?datagrid&configId=${config_id}&field=${fileds}${initquery}';
    initUrl = encodeURI(initUrl);
    $('#${config_id}List').<#if config_istree=="Y">treegrid<#else>datagrid</#if>(
```

7.9.4. 使用



模板上传后开在【Online表单开发】功能编辑页面选择

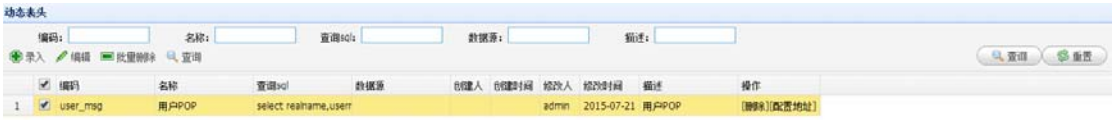
点击【Online表单开发】列表页面【功能测试】可查看效果

7.10. Online 报表配置

7.10.1. 原理

通过对 sql 进行解析配置需要显示的字段，通过 sql 查询直接得出数据

7.10.2. 使用



【sql 解析】功能可对输入的 sql 进行分析得出字段信息，【数据源】功能可查询不同的数据库，实现一个程序在多数据库的自由切换。【sql 解析】代码如下：

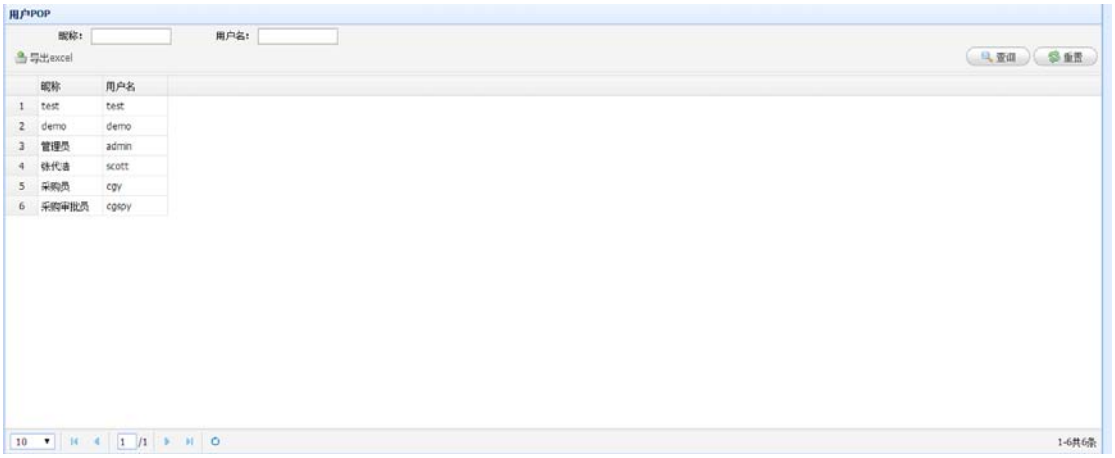
```
@SuppressWarnings("unchecked")
@RequestMapping(params = "getFields", method = RequestMethod.POST)
@ResponseBody
public Object getSqlFields(String sql,String dbKey){
    List<String> result = null;
    Map reJson = new HashMap<String, Object>();
    try{
        //update-begin--Author:张忠亮  Date:20150619 for：sql解析多数据源
        if(StringUtils.isNotBlank(dbKey)){
            List<Map<String,Object>> dataList=DynamicDBUtil.findList(dbKey,SqlUtil.jeecgCreatePageSql(sql,nul
            if(dataList.size()<1){
                throw new BusinessException("该报表sql没有数据");
            }
            Set fieldsSet= dataList.get(0).keySet();
            result = new ArrayList<String>(fieldsSet);
        }else{
            result = cgReportService.getSqlFields(sql);
        }
        //update-end--Author:张忠亮  Date:20150619 for：sql解析多数据源
    }catch (Exception e) {
        e.printStackTrace();
```

页面展示采用 freemaker 渲染：

```
</script>
<table width="100%"   id="${config_id}List" toolbar="#${config_id}Listtb"></table>
<div id="${config_id}Listtb" style="...">
<div name="searchColums">
    <#list config_queryList  as x>
        <span style="...">
        <span style="..." title="${x['field_txt']}">${x['field_txt']}: </span>
        <#if x['search_mode']=="group">
            <input type="text" name="${x['field_name']}_begin"  style="..."  <#if x['field_type']=="Date">c
            <span style="...">~</span>
            <input type="text" name="${x['field_name']}_end"  style="..." <#if x['field_type']=="Date">clas
        </#if>
        <#if x['search_mode']=="single">
                <#if  (x['field_dictlist']?size >0)>
                <select name = "${x['field_name']}" WIDTH="100" style="...">
                <option value = "">---请选择---</option>
                <#list x['field_dictlist']  as xd>
                    <option value = "${xd['typecode']}">${xd['typename']}</option>
                </#list>
                </select>
```

配置完成后可通过【配置地址】功能配置菜单查看。

注意：菜单需配置权限。



7.11. Online 图表配置

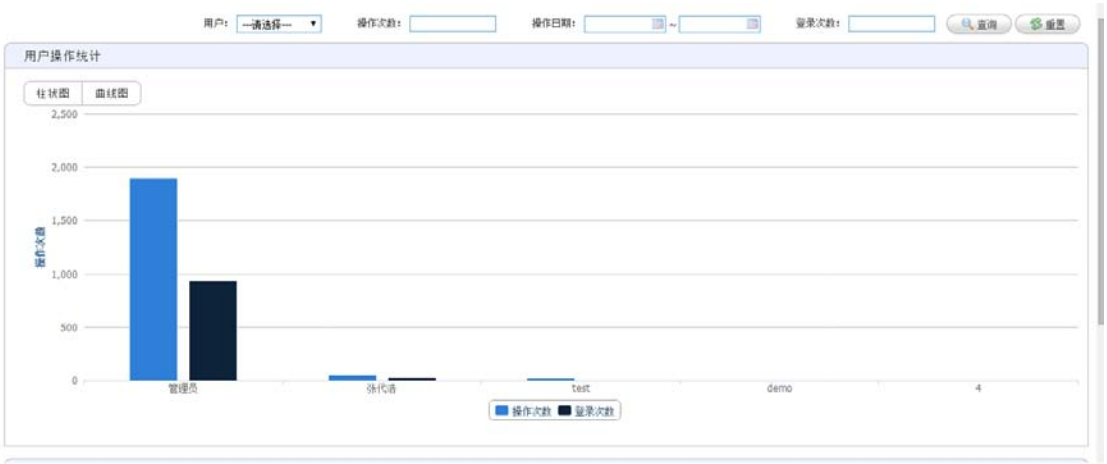
原理

实现原理与 online 报表类似，通过对 sql 进行分析得出相应字段，通过图表进行展示。

模板部分代码如下：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-tran
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<!-- context path -->
${config_iframe}
</head>
<body>
<div class="operations" style="...">
    <div class="bd3">
        <#list config_queryList as x>
            <span style="...">
            <span style="..." title="${x['field_txt']}">${x['field_txt']}: </span>
            <#if x['search_mode']=="group">
                <input type="text" name="${x['field_name']}_begin"  style="..." <#if x['field_type']=="Dat
                <span style="...">~</span>
                <input type="text" name="${x['field_name']}_end"  style="..." <#if x['field_type']=="Date">
            </#if>
            <#if x['search_mode']=="single">
                <#if  (x['field_dictlist']?size >0)>
                    <#if x['field_type']=="ComboGrid">
                        <input type="text" name="${x['field_name']}" id="${x['field_name']}"  style="   "/>
```

效果展示



8. 查询 HQL 过滤器

8.1. 数据过滤现状分析

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 Hibernate 和 Ibatis 实现方式分析：

[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

8.2. 查询条件 SQL 生成器

8.2.1. 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。

实现的功能：实现了"模糊查询"，"包含查询"，"不匹配查询"等 SQL 匹配功能。

特点：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。

8.2.2. 查询规则

要求：页面查询字段，需跟后台 Action(或 Controller)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).

[5].时间范围范围查询

jsp 页面中使用的 name：需要查询的日期类型字段名本身（什么都不加），表示查询时查询等于该字段时间的数据

begin：需要查询的日期类型字段名（首字母大写），表示查询开始时间 查询时查询大于等于开始时间的数据

end：需要查询的日期类型字段名（首字母大写），表示查询结束时间查询时查询小于等于结束时间的数据

使用举例：

字段名称 private Date birthday

查询开始时间 beginBirthday

查询结束时间 endBirthday

8.2.3. 具体实现

第一步：页面实现

说明:为 dategrid 字段,追加属性 query="true"，自动加载出查询框,如图 5-1 所示。

```
<t:datagrid name="jeecgNoteList" title="公告" actionUrl="jeecgNoteController.do?datagrid"
idField="id" fit="true">
   <t:dgCol title="编号" field="id" hidden="false"></t:dgCol>
   <t:dgCol title="年龄" field="age" query="true"></t:dgCol>
   <t:dgCol title="生日" field="birthday" formatter="yyyy-MM-dd"></t:dgCol>
   <t:dgCol title="出生日期" field="createdt" formatter="yyyy-MM-dd hh:mm:ss"></t:dgCol>
   <t:dgCol title="用户名" field="name"  query="true"></t:dgCol>
   <t:dgCol title="工资" field="salary" ></t:dgCol>
   <t:dgCol title="操作" field="opt" width="100"></t:dgCol>
   <t:dgDelOpt title="删除" url="jeecgNoteController.do?del&id={id}" />
   <t:dgToolBar title="录入" icon="icon-add" url="jeecgNoteController.do?addorupdate"
funname="add"></t:dgToolBar>
   <t:dgToolBar title="编辑" icon="icon-edit" url="jeecgNoteController.do?addorupdate"
funname="update"></t:dgToolBar>
  </t:datagrid>
```

图 5-1 JSP 代码实现

第二步：controller 层处理

Controller 中对应的处理逻辑如图 5-2 所示。

```
CriteriaQuery cq = new CriteriaQuery(TSUser.class, dataGrid);
//查询条件组装器
org.jeecgframework.core.extend.hqlsearch.HqlGenerateUtil.installHql(cq, user);
```

图 5-2 Controller 代码

8.3. 查询过滤器高级特性

datagrid 中的查询过滤器默认是单条件查询，即在设置多个 dgCol 的 query=”true”之后，查询条件中同时只能有一个条件被使用，生成的页面效果如图 5-3 所示。

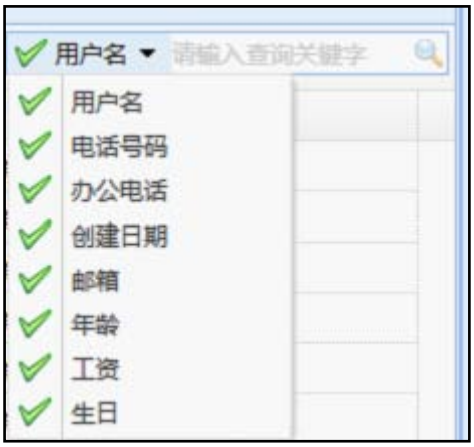


图 5-3 默认查询过滤器效果

当然，可以通过 datagrid 和 dgCol 的参数设置来达到更高级的查询过滤功能，如组合查询条件和值范围查询。

8.3.1. 组合条件查询

设置<t:dategrid>标签的 queryMode=”group”（该参数值默认为”single”，即单条件查询），在页面生成时，会生成一个组合查询条件输入面板。生成的页面效果如图 5-4 所示。

图 5-4 组合查询过滤器效果

8.3.2. 字段范围查询

设置<t:dgCol>标签的 queryMode=”group”，在页面生成时，会生成一个范围输入框。生成的页面效果如图 5-4 所示。

图 5-5 字段范围查询效果

字段范围查询会为该字段生成两个输入框，name 分别为“字段名_begin”和“字段名_end”，具体的查询功能需要在后台接收这两个输入框的内容，并把查询条件加入到 HQL 中。示例如下：

```
@RequestMapping(params = "datagrid")
public void datagrid(JeecgDemo jeecgDemo,HttpServletRequest request,
HttpServletResponse response, DataGrid dataGrid) {
     CriteriaQuery cq = new CriteriaQuery(JeecgDemo.class, dataGrid);
    //查询条件组装器
    org.jeecgframework.core.extend.hqlsearch.HqlGenerateUtil.installHql(cq, jeecgDemo);
    String ctBegin = request.getParameter("createTime_begin");
    String ctEnd = request.getParameter("createTime_end");
    if(ctBegin!=null && ctEnd!=null){
          try {
                cq.ge("createTime", new SimpleDateFormat("yyyy-MM-dd").parse(ctBegin));
                cq.le("createTime", new SimpleDateFormat("yyyy-MM-dd").parse(ctEnd));
           } catch (ParseException e) {
                 e.printStackTrace();
           }
           cq.add();
    }
     this.jeecgDemoService.getDataGridReturn(cq, true);
     TagUtil.datagrid(response, dataGrid);
```

```
}
```

在控制器中用 request 接收传递到后台的查询条件，或者直接在方法参数列表里填上，springmvc 会帮我们获得。

然后将得到的范围查询条件添加到 CriteriaQuery 对象中，最后调用 CriteriaQuery 的 add()方法加载生成 hql。

至此，范围查询就完成了。

8.3.3. 查询字段添加日期控件

例如，要给创建日期的范围查询条件框添加日期控件，首先为创建日期添加范围查询：

```
<t:dgCol title="创建日期" field="createTime" formatter="yyyy-MM-dd hh:mm:ss"
query="true" queryMode="group"></t:dgCol>
```

用 jquery 为生成的 createTime_start 和 createTime_end 两个输入框添加日期控件。

```
$(document).ready(function(){
$("input[name='createTime_begin']").attr("class","easyui-datebox");
$("input[name='createTime_end']").attr("class","easyui-datebox");
);
```

最终的效果如图 5-6 所示。

图 5-6 查询条件添加日期控件效果

8.3.4. 日期字段的数据格式化

在 dategrid 中，对于日期字段，可以通过设置<d:dgCol>的 formatter 属性配置格式化方式，实现对日期数据的格式化，如：

```
<t:dgCol title="创建日期" field="createTime" formatter="yyyy-MM-dd hh:mm:ss"
query="true" queryMode="group"></t:dgCol>
```

对于日期的格式化方式，可以参考 JDK 参考手册中 SimpleDateFormat 中对于日期和时间模式的说明，如图 5-7 所示。

字母	日期或时间元素	表示	示例
G	Era 标志符	Text	AD
y	年	Year	1996; 96
M	年中的月份	Month	July; Jul; 07
w	年中的周数	Number	27
W	月份中的周数	Number	2
D	年中的天数	Number	189
d	月份中的天数	Number	10
F	月份中的星期	Number	2
E	星期中的天数	Text	Tuesday; Tue
a	Am/pm 标记	Text	PM
H	一天中的小时数（0-23）	Number	0
k	一天中的小时数（1-24）	Number	24
K	am/pm 中的小时数（0-11）	Number	0
h	am/pm 中的小时数（1-12）	Number	12
m	小时中的分钟数	Number	30
s	分钟中的秒数	Number	55
S	毫秒数	Number	978
z	时区	General time zone	Pacific Standard Time; PST; GMT-08:00
Z	时区	RFC 822 time zone	-0800

图 5-7 日期和时间模式

8.3.5. 数据列表合计功能

进行数据的列表展示时，为数据显示合计数是一个很有用的功能，在 jeecg 的 datagrid 中实现该功能的效果如图 5-8 所示。

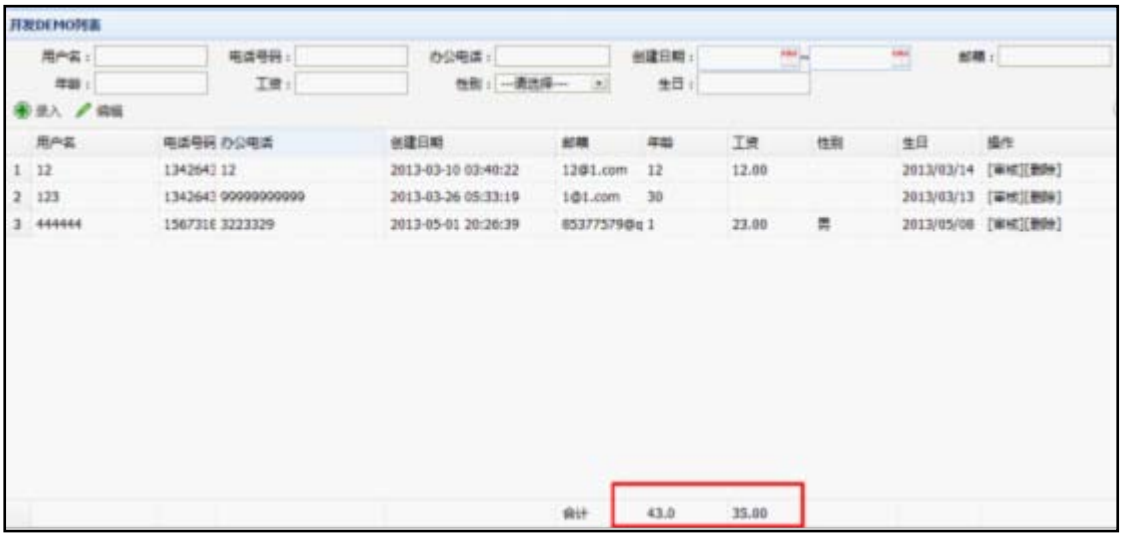


图 5-8 列表数据合计效果图

该功能的实现，主要是通过在加载 datagrid 的数据时，统计出所需的合计值，并放在 datagrid 对象的 footer 中。示例代码如下：

```
1 @RequestMapping(params = "datagrid")
2 publicvoid datagrid(JeecgDemo jeecgDemo,HttpServletRequest request,
HttpServletResponse response, DataGrid dataGrid) {
3       CriteriaQuery cq = new CriteriaQuery(JeecgDemo.class, dataGrid);
```

```
4 //查询条件组装器
5
org.jeecgframework.core.extend.hqlsearch.HqlGenerateUtil.installHql(cq,
jeecgDemo);
6       String ctBegin = request.getParameter("createTime_begin");
7       String ctEnd = request.getParameter("createTime_end");
8 if(StringUtil.isNotEmpty(ctBegin)&& StringUtil.isNotEmpty(ctEnd)){
9 try {
10              cq.ge("createTime", new
SimpleDateFormat("yyyy-MM-dd").parse(ctBegin));
11              cq.le("createTime", new
SimpleDateFormat("yyyy-MM-dd").parse(ctEnd));
12          } catch (ParseException e) {
13              e.printStackTrace();
14          }
15          cq.add();
16      }
17 this.jeecgDemoService.getDataGridReturn(cq, true);
18 //update-begin--Author:zhaojunfu  Date:20130520 for：TASK #109 datagrid
标签没有封装合计功能
19      String total_salary =
String.valueOf(jeecgDemoService.findOneForJdbc("select sum(salary) as ssum
from jeecg_demo").get("ssum"));
20 /*
21 * 说明：格式为 字段名:值(可选，不写该值时为分页数据的合计) 多个合计 以 , 分割
22       */
23  dataGrid.setFooter("salary:"+total_salary+",age,email:合计");
24 //update-end--Author:zhaojunfu  Date:20130520 for：TASK #109 datagrid 标
签没有封装合计功能
25      TagUtil.datagrid(response, dataGrid);
26   }
```

在该示例代码中，需要重点注意的是这里的第 23 行：

```
dataGrid.setFooter("salary:"+total_salary+",age,email:合计");
```

setFooter()方法接收一个字符串，其格式为 s：字段名[:值]，其中值为选填项，填了则使用给定的值，没填则自动统计分页合计，示例：

```
salary:35.00,age,email:合计
```

这里将 salary 的合计值通过查询数据库得出，而 age 则通过当前分页数据自动合计，email 给定一个值“合计”，其作用是在 datagrid 对应于 email 列的下方显示一个说明信息。

9. 标签中使用数据字典

数据字典为系统中可能用到的字典类型数据提供了使用的便利性和可维护性。以下拉框标签<t:dictSelect>为例进行讲解

9.1. 标签参数

属性名	类型	描述	是否必须	默认值
typeGroupCode	string	字典分组编码	是	null
field	string	对应表单	是	null
id	string	唯一标识	否	null
title	string	显示文本	否	null
defaultVal	string	默认值	否	null
divClass	string	DIV 框默认样式	否	form
labelClass	string	LABEL 默认样式	否	Validform_label
hasLabel	string	是否显示 Label	否	null
type	string	控件格式 select|radio|checkbox	否	Select
dictTable	string	自定义字典表	否	null
dictField	string	自定义字典表的匹配字段-字典的编码值	否	null
dictText	string	自定义字典表的显示文本-字典的显示值	否	null
dictCondition	string	自定义字典表的显示文本-字典查询条件	否	null
extendJson	string	扩展参数(json 格式)	否	null

应用示例：

```
<t:dictSelect field="name" typeGroupCode="process" title="流程类型"></t:dictSelect>
```

9.2. 使用案例

通过利用数据字典来做性别下拉框的案例来详细说明数据字典的使用步骤。

步骤一：数据字典维护

首先，在数据字典菜单下录入一个字典编码为“sex”、字典名称为“性别类型”的字典项，如图 6-1 所示。

图 6-1 字典分组添加

然后在该分组中分别添加“男性”、“女性”两个类型，并分别设置其类型编码为“1”和“0”，如图 6-2 所示。

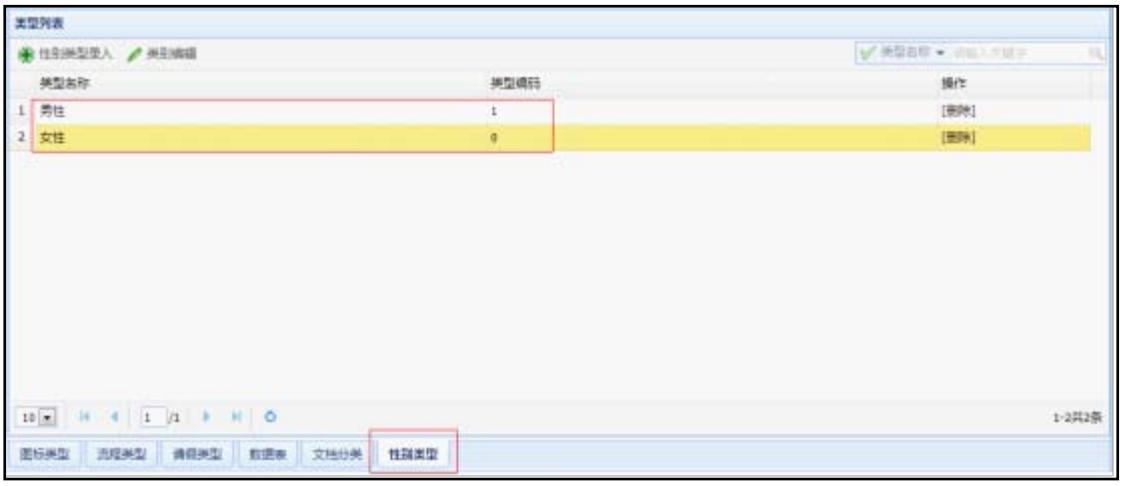


图 6-2 为分组添加字典类型

步骤二：在页面中使用字典

首先，在 JSP 页面中引入 JEECG_UI 标签库：

```
<%@include file="/context/mytags.jsp"%>
```

然后就可以使用 dictSelect 标签把性别按下拉框显示出来了。代码如下：

```
<t:dictSelect field="sexField" typeGroupCode="sex"></t:dictSelect>
```

其中，typeGroupCode 就是我们在步骤一中定义的类型分组编码“sex”。页面显示结果如图 6-3 所示。

图 6-3 字典展现效果

10. 表单校验组件 ValidForm

10.1. Validform 使用入门

1、引入 css

请查看下载文件中的 style.css，把里面 Validform 必须部分复制到你的 css 中（文件里这个注释 "/*==========以下部分是 Validform 必须的==========*/" 之后的部分是必须的）。(之前发现有部分网友把整个 style.css 都引用在了页面里，然后发现样式冲突了)

2、引入 js （jquery 1.4.2 以上版本都可以）

```
<scripttype="text/javascript" src="http://validform.rjboy.cn/wp-content/themes/validform/js/jquery-1.6.2.min.js"></script>
<scripttype="text/javascript" src="http://validform.rjboy.cn/Validform/v5.1/Validform_v5.1_min.js"></script>
```

3、给需要验证的表单元素绑定附加属性

```
<form class="demoform"/>
<inputtype="text" value="" name="name" datatype="s5-16" errormsg="昵称至少5个字符,最多16个字符！"/>
</form>
```

4、初始化，就这么简单

```
$(".demoform").Validform();
```

注：

1、Validform 有非压缩、压缩和 NCR 三个版本提供下载，NCR 是通用版，当你页面因编码问题，提示文字出现乱码时可以使用这个版本；

2、Validform 没有限定必须使用 table 结构，它可以适用于任何结构，需要在 tiptype 中定义好位置关系。

10.2. 绑定附加属性

凡要验证格式的元素均需绑定 datatype 属性，datatype 可选值内置有 10 类，用来指定不同的验证格式。

如果还不能满足您的验证需求，可以传入自定义 datatype，自定义 datatype 是一个非常强大的功能，通过它可以满足你的任何需求。

可以绑定的附加属性有：datatype、nullmsg、sucmsg、errormsg、ignore、recheck、tip、altercss、ajaxurl 和 plugin 等。

绑定方法如下所示：

```
<!--ajax实时验证用户名-->
<input type="text" value="" name="name" datatype="e" ajaxurl="valid.php?myparam1=value1&myparam2=value2"
sucmsg="用户名验证通过！" nullmsg="请输入用户名！" errormsg="请用邮箱或手机号码注册！" />

<!--密码-->
<input type="password" value="" name="userpassword" datatype="*6-15" errormsg="密码范围在6~15位之间！" />
<!--确认密码-->
<input type="password" value="" name="userpassword2" datatype="*" recheck="userpassword" errormsg="您两次输入的
账号密码不一致！" />

<!--默认提示文字-->
<textarea tip="请在这里输入您的意见。" errormsg="很感谢您花费宝贵时间给我们提供反馈，请填写有效内容！"
datatype="s" altercss="gray" class="gray" name="msg" value="">请在这里输入您的意见。</textarea>

<!--使用swfupload插件-->
<input type="text" plugin="swfupload" class="inputxt" disabled="disabled" value="">
<input type="hidden" value="" pluginhidden="swfupload">

<!--使用passwordStrength插件-->
<input type="password" errormsg="密码至少6个字符,最多18个字符！" datatype="*6-18" plugin="passwordStrength"
class="inputxt" name="password" value="">
<div class="passwordStrength" style="display:none;"><b>密码强度：</b> <span>弱</span><span>中</span><span
class="last">强</span></div>

<!--使用DatePicker插件-->
<input type="text" plugin="datepicker" class="inputxt" name="birthday" value="">
```

说明：

内置基本的 datatype 类型有： * | *6-16 | n | n6-16 | s | s6-18 | p | m | e | url

*：检测是否有输入，可以输入任何字符，不留空即可通过验证；

*6-16：检测是否为 6 到 16 位任意字符；

n：数字类型；

n6-16：6 到 16 位数字；

s：字符串类型；

s6-18：6 到 18 位字符串；

p：验证是否为邮政编码；

m：手机号码格式；

e：email 格式；

url：验证字符串是否为网址。

自定义 datatype 的名称，可以由字母、数字、下划线、中划线和*号组成。

形如"*6-16"的 datatype，Validform 会自动扩展，可以指定任意的数值范围。如内置基本类型有"*6-16"，那么你绑定 datatype="*4-12"就表示 4 到 12 位任意字符。如果你自定义了一个 datatype="zh2-4"，表示 2 到 4 位中文字符，那么 datatype="zh2-6"就表示 2 到 6 位中文字符。

5.2 版本之后，datatype 支持规则累加或单选。用","分隔表示规则累加；用"|"分隔表示规则多选一，即只要符合其中一个规则就可以通过验证，绑定的规则会依次验证，只要验证通过，后面的规则就会忽略不再比较。如绑定 datatype="m|e"，表示既可以填写手机号码,也能填写邮箱地址,如果知道填入的是手机号码,那么就不会再检测他是不是邮箱地址；datatype="zh,s2-4"，表示要符合自定义类型"zh"，也要符合规则"s2-4"。

注：

5.2.1 版本之后，datatype 支持：

直接绑定正则：如可用这样写 datatype="/\w{3,6}/i"，要求是 3 到 6 位的字母，不区分大小写；

支持简单的逻辑运算：如

datatype="m | e, *4-18 | /\w{3,6}/i | /^validform\.rjboy\.cn$/"，

这个表达式的意思是：可以是手机号码；或者是邮箱地址，但字符长度必须在 4 到 18 位；或者是 3 到 6 位的字母，不区分大小写；或者输入 validform.rjboy.cn，区分大小写。这里","分隔相当于逻辑运算里的"&&"； "|"分隔相当于逻辑运算里的"||"；不支持括号运算。

nullmsg

当表单元素值为空时的提示信息，不绑定，默认提示"请填入信息！"。

如：nullmsg="请填写用户名！"

5.3 版开始，对于没有绑定 nullmsg 的对象，会自动查找 class 为 Validform_label 下的文字作为提示文字:

如这样的 html 结构：

*用户名：<input type="text" val="" datatype="s" />

当这个文本框里没有输入时的出错信息就会是："请输入用户名！"

这里 Validform_label 跟 input 之间的位置关系，不一定是要同级关系，同级里没有找到的话，它还会在同级的子级、父级的同级、父级的同级的子级里查找。

sucmsg 5.3+

当表单元素通过验证时的提示信息，不绑定，默认提示"通过信息验证！"。

如：sucmsg="用户名还未被使用，可以注册！"

5.3 版开始，也可以在实时验证返回的 json 数据里返回成功的提示文字，请查看附加属性 ajaxurl 的介绍。

errormsg

输入内容不能通过验证时的提示信息，默认提示"请输入正确信息！"。

如：errormsg="用户名必须是 2 到 4 位中文字符！"

5.3 版开始，Validform 可以根据你绑定的 datatype 智能的输出相应出错信息，具体介绍请查看 tipmsg

ignore

绑定了 ignore="ignore"的表单元素，在有输入时，会验证所填数据是否符合 datatype 所指定数据类型，

没有填写内容时则会忽略对它的验证；

recheck

表单里面经常需要检查两次密码输入是否一致，recheck 就是用来指定需要比较的另外一个表单元素。

如：recheck="password1"，那么它就会拿当前元素的值跟该表单下，name 为"password1"的元素比较。

tip

表单里经常有些文本框需要默认就显示一个灰色的提示文字，当获得焦点时提示文字消失，失去焦点时提示文字显示。tip 属性就是用来实现这个效果。它通常和 altercss 搭配使用。

如<input type="text" value="默认提示文字" class="gray intxt" tip="默认提示文字" altercss="gray" />

altercss

它需要和 tip 属性配合使用，altercss 指定的样式名，会在文本框获得焦点时被删除，没有输入内容而失去焦点时重新加上。

ajaxurl

指定 ajax 实时验证的后台文件的地址。

后台页面如 valid.php 文件中可以用 $_POST["param"] 接收到值，Ajax 中会 POST 过来变量 param 和 name。param 是文本框的值，name 是文本框的 name 属性。

5.2 版本开始，可以在 ajaxurl 指定的 url 后绑定参数，如：

```
ajaxurl="valid.php?myparam1=value1&myparam2=value2";
```

5.3 之前的版本中，该文件输出的字符会作为错误信息显示在页面上，如果验证通过需输出小写字母"y"。

在 5.3 版中，实时验证的返回数据做了调整，须是含有 status 值的 json 数据！跟 callback 里的 ajax 返回数据格式统一，建议不再返回字符串"y"或"n"。目前这两种格式的数据都兼容。

注：

如果 ajax 校验通过，会在该元素上绑定 validform_valid 值为 true。可以通过设置该值来控制验证能不能通过，如验证码的验证，第一次验证通过后，不小心右点击了下验证码图片，验证码换了，但是仍然指示这个对象已经通过了验证，这时可以手动调整该值：$("#name")[0].validform_valid="false"。

怎样设置 ajax 的参数，具体可以查看 Validform 对象的 config 方法。

plugin

指定需要使用的插件。

5.3 版开始，对于日期、swfupload 和密码强度检测这三个插件，绑定了 plugin 属性即可以初始化对应的插件，可以不用在 validform 初始化时传入空的 usePlugin 了。

如，你要使用日期插件，5.3 之前版本需要这样初始化：

```
$(".demoform").Validform({
    usePlugin:{
        datepicker:{}
    }
});
```

5.3 版开始，不需要传入这些空对象了，只需在表单元素上绑定 plugin="datepicker" 就可以，初始化直接这样：

```
$(".demoform").Validform();
```

10.3. 初始化参数说明

所有可用的参数如下：

```
$(".demoform").Validform({
    btnSubmit:"#btn_sub",
    btnReset:".btn_reset",
    tiptype:1,
    ignoreHidden:false,
        dragonfly:false,
    tipSweep:true,
    showAllError:false,
    postonce:true,
    ajaxPost:true,
    datatype:{
        "*6-20":/^[^\s]{6,20}$/,
        "z2-4":/^[\u4E00-\u9FA5\uf900-\ufa2d]{2,4}$/,
        "username":function(gets,obj,curform,regxp){
            //参数gets是获取到的表单元素值，obj为当前表单元素，curform为当前验证的表单，regxp为内置的一些正则表达式的引用;
            var reg1=/^[\w\.]{4,16}$/,
                reg2=/^[\u4E00-\u9FA5\uf900-\ufa2d]{2,8}$/;

            if(reg1.test(gets)){returntrue;}
            if(reg2.test(gets)){returntrue;}
            returnfalse;

            //注意return可以返回true 或 false 或字符串文字，true表示验证通过，返回字符串表示验证失败，字符串作为错误提示显示，返回false则用errmsg或默认的错误提示;
        },
        "phone":function(){
            //5.0版本之后，要实现二选一的验证效果，datatype的名称不需要以"option_"开头;
        }
    },
    usePlugin:{
        swfupload:{},
        datepicker:{},
        passwordstrength:{},
        jqtransform:{
            selector:"select,input"
        }
    },
    beforeCheck:function(curform){
        //在表单提交执行验证之前执行的函数，curform参数是当前表单对象。
        //这里明确return false的话将不会继续执行验证操作;
```

```
    },
    beforeSubmit:function(curform){
        //在验证成功后，表单提交前执行的函数，curform 参数是当前表单对象。
        //这里明确 return false 的话表单将不会提交;
    },
    callback:function(data){
        //返回数据 data 是 json 格式，{"info":"demo info","status":"y"}
        //info: 输出提示信息;
        //status: 返回提交数据的状态,是否提交成功。如可以用"y"表示提交成功，"n"表示提交失败，在
ajax_post.php 文件返回数据里自定字符，主要用在 callback 函数里根据该值执行相应的回调操作;
        //你也可以在 ajax_post.php 文件返回更多信息在这里获取，进行相应操作;
        //ajax 遇到服务端错误时也会执行回调，这时的 data 是{ status:**, statusText:**, readyState:**,
responseText:** };

        //这里执行回调操作;
        //注意：如果不是 ajax 方式提交表单，传入 callback，这时 data 参数是当前表单对象，回调函数会
在表单验证全部通过后执行，然后判断是否提交表单，如果 callback 里明确 return false，则表单不会提交，
如果 return true 或没有 return，则会提交表单。
    }
});
```

参数说明：【所有参数均为可选项】

必须是表单对象执行 Validform 方法，示例中".demoform"就是绑定在 form 元素上的 class 名称;

btnSubmit

指定当前表单下的哪一个按钮触发表单提交事件，如果表单下有 submit 按钮时可以省略该参数。示例中".btn_sub"是该表单下要绑定点击提交表单事件的按钮;

btnReset

".btn_reset"是该表单下要绑定点击重置表单事件的按钮;

tiptype

可用的值有：1、2、3、4 和 function 函数，默认 tiptype 为 1。(3、4 是 5.2.1 版本新增)

1=>自定义弹出框提示;

2=>侧边提示(会在当前元素的父级的 next 对象的子级查找显示提示信息的对象，表单以 ajax 提交时会弹出自定义提示框显示表单提交状态);

3=>侧边提示(会在当前元素的 siblings 对象中查找显示提示信息的对象，表单以 ajax 提交时会弹出自定义提示框显示表单提交状态);

4=>侧边提示(会在当前元素的父级的 next 对象下查找显示提示信息的对象，表单以 ajax 提交时不显示表单的提交状态)；

如果上面的 4 种提示方式不是你需要的，你可以给 tiptype 传入自定义函数。通过自定义函数，可以实现你想要的任何提示效果：

```
tiptype:function(msg,o,cssctl){
    //msg：提示信息;
    //o:{obj:*,type:*,curform:*},
    //obj 指向的是当前验证的表单元素（或表单对象，验证全部验证通过，提交表单时 o.obj 为该表单对象），
    //type 指示提示的状态，值为 1、2、3、4， 1：正在检测/提交数据，2：通过验证，3：验证失败，4：提示 ignore 状态,
    //curform 为当前 form 对象;
    //cssctl:内置的提示信息样式控制函数，该函数需传入两个参数：显示提示信息的对象和当前提示的状态（既形参 o 中的 type）;
}
```

具体参见 demo 页。

tiptype 不为 1 时，Validform 会查找 class 为"Validform_checktip"的标签显示提示信息。tiptype=1 时，会自动创建弹出框显示提示信息。

Validform_checktip 和表单元素之间的位置关系，会根据 tiptype 的值有对应的结构，上面已经做了说明。

5.3 版本开始，如果页面里没有显示出错信息的标签，会根据 tiptype 值自动创建 Validform_checktip 对象。

ignoreHidden

可用值： true | false。

默认为 false，当为 true 时对:hidden 的表单元素将不做验证;

dragonfly

可用值： true | false。

默认 false，当为 true 时，值为空时不做验证;

tipSweep

可用值： true | false。

默认为 false， 5.3 版中做了修正，在各种 tiptype 下， 为 true 时提示信息将只会在表单提交时触发显示，各表单元素 blur 时不会触发信息提示；

showAllError

可用值： true | false。

默认为 false，true：提交表单时所有错误提示信息都会显示；false：一碰到验证不通过的对象就会停止检测后面的元素，只显示该元素的错误信息；

postonce

可用值： true | false。

默认为 false，指定是否开启二次提交防御，true 开启，不指定则默认关闭；

为 true 时，在数据成功提交后，表单将不能再继续提交。

ajaxPost

可用值： true | false。

默认为 false，使用 ajax 方式提交表单数据，将会把数据 POST 到 config 方法或表单 action 属性里设定的地址；

datatype

传入自定义 datatype 类型，可以是正则，也可以是函数。

```
datatyp:{
    "zh2-4":/^[\u4E00-\u9FA5\uf900-\ufa2d]{2,4}$/,
    "phone":function(gets,obj,curform,regxp){
        //参数 gets 是获取到的表单元素值，
        //obj 为当前表单元素，
        //curform 为当前验证的表单，
        //regxp 为内置的一些正则表达式的引用。
//return false 表示验证出错，没有 return 或者 return true 表示验证通过。
    }
}
```

具体示例请参考 demo 页；

usePlugin

目前已整合 swfupload、datepicker、passwordstrength 和 jqtransform 四个插件，在这里传入这些插件使用时需要传入的参数。datepicker 在 Validform 内调用时另外扩展了几个比较实用的参数，具体请参考 demo 页；

beforeCheck(curform)

在表单提交执行验证之前执行的函数，curform 参数获取到的是当前表单对象。函数 return false 的话将不会继续执行验证操作；

beforeSubmit(curform)

在表单验证通过，提交表单数据之前执行的函数，data 参数是当前表单对象。函数 return false 的话表单将不会提交；

callback

在使用 ajax 提交表单数据时，数据提交后的回调函数。返回数据 data 是 Json 格式：{"info":"demo info","status":"y"}

info：输出提示信息，

status：返回提交数据的状态,是否提交成功，"y"表示提交成功，"n"表示提交失败，在 ajax_post.php 文件返回数据里自定字符，主要用在 callback 函数里根据该值执行相应的回调操作。你也可以在 ajax_post.php 文件返回更多信息在这里获取，进行相应操作；

如果不是 ajax 方式提交表单，传入 callback，这时 data 参数是当前表单对象，回调函数会在表单验证全部通过后执行，然后判断是否提交表单，如果 callback 里 return false，则表单不会提交，如果 return true 或没有 return，则会提交表单。

5.3 版开始，ajax 遇到服务端错误时也会执行回调，这时的 data 是{ status:**, statusText:**, readyState:**, responseText:** }

10.4. Validform 对象[方法支持链式调用]

如示例 var demo=$(".formsub").Validform()，那么 demo 对象会有以下属性和方法可以调用：

tipmsg【object】

如：demo.tipmsg.s="error! no message inputed.";

通过该对象可以修改除 tit 以外的其他提示文字，这样可以实现同一个页面的不同表单使用不同的提示文字。

具体可修改的提示文字

```
$.Tipmsg={//默认提示文字;
tit:"提示信息",
w:{
      "*":"不能为空！",
      "*6-16":"请填写 6 到 16 位任意字符！",
      "n":"请填写数字！",
      "n6-16":"请填写 6 到 16 位数字！",
      "s":"不能输入特殊字符！",
      "s6-18":"请填写 6 到 18 位字符！",
      "p":"请填写邮政编码！",
      "m":"请填写手机号码！",
      "e":"邮箱地址格式不对！",
      "url":"请填写网址！"
},
def:"请填写正确信息！",
undef:"datatype 未定义！",
reck:"两次输入的内容不一致！",
r:"通过信息验证！",
c:"正在检测信息…",
s:"请填入信息！",
s_auto:"请{填写}{0}！",
v:"所填信息没有经过验证，请稍后…",
p:"正在提交数据…"
};
```

要修改 tit（弹出框的标题文字）的话，可以这样：$.Tipmsg.tit="Message Box"，则弹出框的标题文字会换成"Message Box"。

注：5.3+

$.Tipmsg.w 里，形如"*6-16"的提示文字，里面的数字是会被替换的。如绑定 datatype="*2-18"，那它默认的出错信息就会是"请填写 2 到 18 位任意字符！"，可以通过 $.Tipmsg.w 这个对象修改和扩展默认错误信息，如果你已经设置了"zh2-4"的提示信息是"2-4 位中文",那么"zh2-8"出错的信息就自动会是:"2-8 位中文"。对于自定义的 datatype，在扩展默认信息时，注意错误信息的名字要跟 datatype 名字一样，如上面示例是：$.Tipmsg.w["zh2-4"]="2-4 位中文"。对于多页面或一个页面多表单有相同 datatype 来说，在$.Tipmsg.w 中扩展默认提示信息是个很好的选择。目前只能通过$.Tipmsg.w 扩展，还不能使用 Validform 对象的 tipmsg 属性来扩展。

$.Tipmsg.s_auto 是用来指定在没有绑定 nullmsg，且指定了标签 Validform_label 时的默认提示信息。"{0}"是会被找到的 Validform_label 里的文字替换掉的，"{填写}"里的文字在绑定了"recheck"属性的表单元素上检测时是会不显示的。

具体示例请参见 demo 页。

dataType【object】

获取内置的一些正则：

```
{
    "match":/^(.+?)(\d+)-(\d+)$/,
    "*":/[\w\W]+/,
    "*6-16":/^[\w\W]{6,16}$/,
    "n":/^\d+$/,
    "n6-16":/^\d{6,16}$/,
    "s":/^[\u4E00-\u9FA5\uf900-\ufa2d\w\.\s]+$/,
    "s6-18":/^[\u4E00-\u9FA5\uf900-\ufa2d\w\.\s]{6,18}$/,
    "p":/^[0-9]{6}$/,
    "m":/^13[0-9]{9}$|14[0-9]{9}|15[0-9]{9}$|18[0-9]{9}$/,
    "e":/^\w+([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*$/,
    "url":/^(\w+:\/\/)?\w+(\.\w+)+.*$/
}
```

addRule(rule)【返回值：Validform】

可以通过 Validform 对象的这个方法来给表单元素绑定验证规则，绑定验证类型中列出的附加属性都可以通过这个方法绑定。

```
demo.addRule([
    {
        ele:"#name",
        datatype:"s6-18",
        ajaxurl:"valid.php",
        nullmsg:"请输入昵称！",
        errormsg:"昵称至少 6 个字符,最多 18 个字符！"
    },
    {
        ele:"#userpassword",
        datatype:"*6-16",
        nullmsg:"请设置密码！",
        errormsg:"密码范围在 6~16 位之间！"
    },
    {
        ele:"#userpassword2",
        datatype:"*",
        recheck:"userpassword",
```

```
            nullmsg:"请再输入一次密码！",
            errormsg:"您两次输入的账号密码不一致！"
        }
]);
```

其中 ele 是指定要绑定规则的对象，会在 Validform 对象下查找这些对象。

eq(n)【返回值：Validform】

获取 Validform 对象的第 n 个元素。

如你页面上有多个 form 的 class 都是 formsub，执行上面的验证绑定，得到的 demo 对象就可以操作所有这些表单，如果你要对其中某个表单执行某些操作，那么就可以使用这个方法。

如 demo.eq(0).resetForm()，重置第一个表单。

ajaxPost(flag,sync,url)【返回值：Validform】

以 ajax 方式提交表单。flag 为 true 时，跳过验证直接提交，sync 为 true 时将以同步的方式进行 ajax 提交。

参数 url 是 5.3 版新增，传入了 url 地址时，表单会提交到这个地址

如 demo.ajaxPost(true)，不做验证直接 ajax 提交表单。

abort()【返回值：Validform】

终止 ajax 的提交。

如执行上面的 ajaxPost()之后，发现某些项填写不对，想取消表单提交，那么就可以执行这个操作：demo.abort()

submitForm(flag,url)【返回值：Validform】

以初始化时传入参数的设置方式提交表单，flag 为 true 时，跳过验证直接提交。

参数 url 是 5.3 版新增，传入了 url 地址时，表单会提交到这个地址

如 demo.submitForm(true)，不做验证直接提交表单。

resetForm()【返回值：Validform】

重置表单。

如 demo.resetForm()，重置表单到初始状态。

resetStatus()【返回值：Validform】

重置表单的提交状态。传入了 postonce 参数的话，表单成功提交后状态会设置为"posted"，重置提交状态可以让表单继续可以提交。

如 demo.resetStatus()

getStatus()【返回值：String】

获取表单的提交状态，normal：未提交，posting：正在提交，posted：已成功提交过。

如 demo.getStatus()

setStatus(status)【返回值：Validform】

设置表单的提交状态，可以设置 normal，posting，posted 三种状态，不传参则设置状态为 posting，这个状态表单可以验证，但不能提交。

如 demo.setStatus("posted")

ignore(selector)【返回值：Validform】

忽略对所选择对象的验证，不传入 selector 则忽略所有表单元素。

如 demo.ignore("select,textarea,#name")，忽略 Validform 对象下所有 select，textarea 及一个 id 为"name"元素的验证。

unignore(selector)【返回值：Validform】

将 ignore 方法所忽略验证的对象重新获取验证效果，不传入 selector 则恢复验证所有表单元素。

如 demo.unignore("select,textarea,#name")，恢复 Validform 对象下所有 select，textarea 及一个 id 为"name"元素的验证。

check(bool,selector)【返回值：Boolean】

bool 为 true 时则只验证不显示提示信息

对指定对象进行验证(默认验证当前整个表单)，通过返回 true，否则返回 false（绑定实时验证的对象，格式符合要求时返回 true，而不会等 ajax 的返回结果）

如 demo.check()，验证当前整个表单，且只验证但不显示对错信息。

config(setup)　5.3+　【返回值：Validform】

setup 参数是一个对象。

如：

```
demo.config({
    url:"这里指定提交地址",
    ajaxpost:{
        //可以传入$.ajax()能使用的，除 dataType 外的所有参数;
    },
    ajaxurl:{
        //可以传入$.ajax()能使用的，除 dataType 外的所有参数;
```

```
    }
})
```

可用参数：

url：指定表单的提交路径，这里指定的路径会覆盖表单 action 属性所指定的路径

ajaxpost：表单以 ajax 提交时，可以在这里配置 ajax 的参数

ajaxurl：配置实时验证 ajax 的参数

(1) 执行 config 可以动态设置、添加参数，如：

```
demo.config({
    url:"http://validform.rjboy.cn"
});
$(".save").click(function(){
    demo.config({
        ajaxpost:{
            timeout:1000
        }
    });
});
```

那么在点击 save 按钮后，demo 所对应的表单的配置为:

```
config={
    url:"http://validform.rjboy.cn",
    ajaxpost:{
        timeout:1000
    }
}
```

(2) 参数 url 的优先级：form 表单的 action 所指定的提交地址会被 config.url 覆盖，config.url 会被 config.ajaxpost.url 覆盖，config.ajaxpost.url 会被 Validform 对象的方法 submitForm(flag,url)和 ajaxPost(flag,sync,url)里的 url 覆盖。

如果表单里没有指定 action 提交地址，那么就会提交到 config.url 设定的地址。

(3) 考虑到整个验证框架的逻辑，传入 dataType 参数不会起作用，不会被覆盖，ajax 必须返回含有 status 值的 json 数据。

另外注意的是：传入的 success 和 error 方法里，能多获取到一个参数，如：

```
demo.config={
    ajaxpost:{
        url:"",
        timeout:1000,
        ...,
```

```
            success:function(data,obj){
                //data 是返回的 json 数据;
                //obj 是当前表单的 jquery 对象;
            },
            error:function(data,obj){
                //data 是{ status:**, statusText:**, readyState:**, responseText:** };
                //obj 是当前表单的 jquery 对象;
            }
        },
        ajaxurl:{
            success:function(data,obj){
                //data 是返回的 json 数据;
                //obj 是当前正做实时验证表单元素的 jquery 对象;
                //注意：5.3 版中，实时验证的返回数据须是含有 status 值的 json 数据！
                //跟 callback 里的 ajax 返回数据格式统一，建议不再返回字符串"y"或"n"。
目前这两种格式的数据都兼容。
            }
        }
}
```

10.5. 调用外部插件

文件上传 - swfupload

选择文件后立即上传

表单验证通过后上传文件

密码强度检测 - passwordStrength

密码强度提示与验证提示同时显示

在符合验证要求时才有密码强度提示

日期控件 - datePicker

默认效果

选择日期后执行回调函数

表单美化 - jqtransform

默认对所有元素美化

美化指定的表单元素

10.6. Validform 的公用对象

$.Datatype

可以通过$.Datatype 对象来扩展 datatype 类型。

如$.Datatype.zh=/^[\u4E00-\u9FA5\uf900-\ufa2d]{1,}$/

$.Tipmsg

可以通过$.Tipmsg 对象来修改默认提示文字。具体可修改的提示文字请查看 Validform 对象的 tipmsg 属性。

如果 Validform 对象的 tipmsg 属性没有找到相关的提示信息，那么就会到$.Tipmsg 中查找对应提示文字。

如$.Tipmsg.tit="msg box"; //设置默认弹出框的标题文字。

$.Showmsg(msg)

调用 Validform 自定义的弹出框。

参数 msg 是要显示的提示文字。

如$.Showmsg("这是提示文字"); //如果不传入信息则不会有弹出框出现，像 $.Showmsg()这样是不会弹出提示框的。

$.Hidemsg()

关闭 Validform 自定义的弹出框。

如$.Hidemsg()

11. 基础用户权限

11.1. 权限设计

基本概念

权限管理模块涉及到的实体有：用户、角色和系统资源(包括系统菜单、页面控件、数据资源等)。用户可以拥有多个角色，角色可以被分配给多个用户。而权限的意思就是对某个资源的某个操作. 一般通用的权限管理模块规定：所谓资源即应用系统中提供的要进行鉴权才能访问的资源(比如各类数据,系统菜单)；所谓操作即增加、修改、删除、查询等操作。

权限模型

用户权限模型，指的是用来表达用户信息及用户权限信息的数据模型。即能证明“你是谁？”、“你能访问哪些受保护资源？”。

用户与角色之间构成多对多关系。表示同一个用户可以拥有多个角色，一个角色可以被多个用户所拥有。

角色与资源之间构成多对多关系。表示同一个资源可以被多个角色访问，一个角色可以访问多个资源。

权限设计模型如图 8-1 所示。

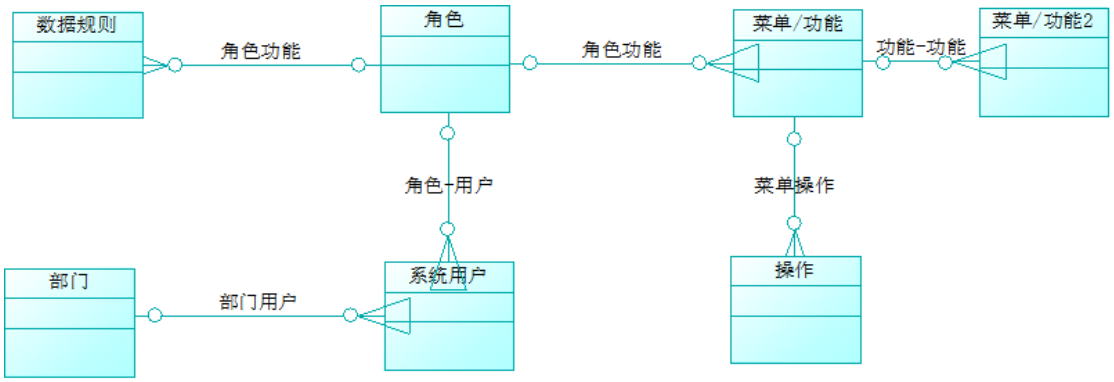


图 8-1 权限设计模型

11.2. 权限设计目标

权限设计及权限管理的目标包括：

1) 对用户授予相应的角色

2) 对角色授予不同的菜单

3) 对角色授予不同的操作页面控件操作权限

4) 进行数据级别的权限控制（行级别、列级别）

11.3. 权限设计

11.3.1. 数据表

数据表	实体类	说明
t_s_user	org.jeecgframework.web.system.pojo.base.TSUser	[用户权限]系统用户表
t_s_base_user	org.jeecgframework.web.system.pojo.base.TSBaseUser	[用户权限]系统用户父类表
t_s_role	org.jeecgframework.web.system.pojo.base.TSRole	[用户权限]角色
t_s_role_user	org.jeecgframework.web.system.pojo.base.TSRoleUser	[用户权限]用户角色

t_s_depart	org.jeecgframework.web.system.pojo.base.TSDepart	[用户权限]部门机构表
t_s_role_function	org.jeecgframework.web.system.pojo.base.TSRoleFunction	[用户权限]角色权限表
t_s_operation	org.jeecgframework.web.system.pojo.base.TSOperation	[用户权限]操作权限表
t_s_function	org.jeecgframework.web.system.pojo.base.TSFunction	[用户权限]菜单权限表
t_s_data_rule	org.jeecgframework.web.system.pojo.base.TSDataRule	[用户权限]数据权限表

11.3.2. 页面菜单

权限管理的相关菜单如图 8-2 所示。

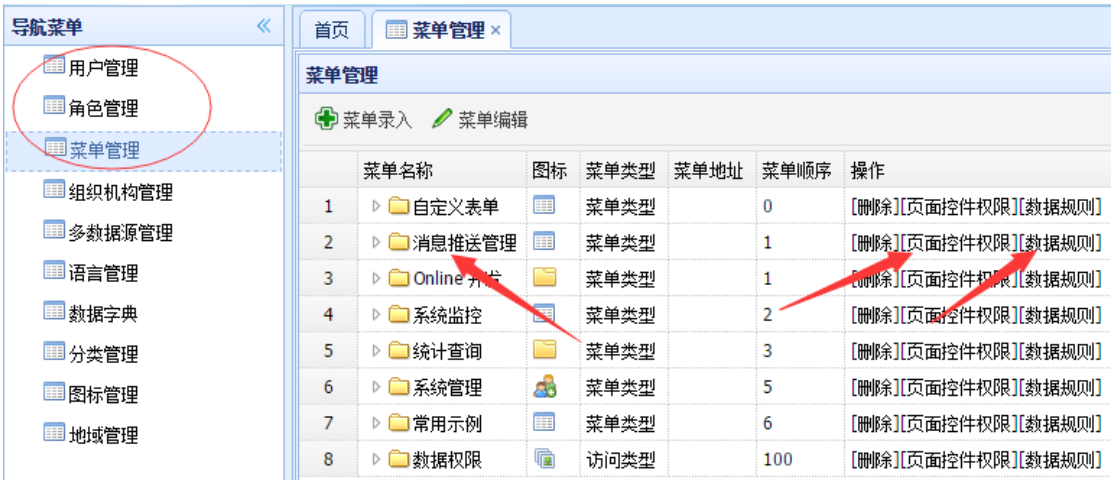


图 8-2 权限菜单

11.3.3. 页面控件权限

使用说明

页面控件级别的权限依赖于菜单权限，也就是说，需要先为角色分配菜单，在已分配的菜单中，可以选择可以操作的页面控件。

页面控件权限的添加在菜单管理页面，点击对应菜单的【页面控件权限】，设置该菜单页面相关控件的操作权限，如图 8-3 所示。

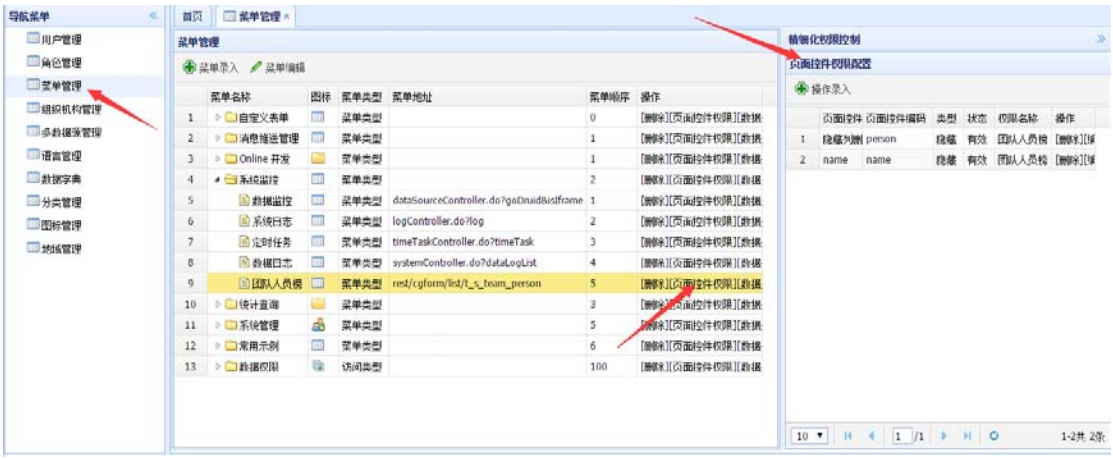


图 8-3 操作按钮设置

页面控件操作权限的分配在角色管理页面，在权限设置时，先为角色分配菜单，点击相应的菜单，在右侧的“操作按钮列表”面板中显示该菜单可分配的操作按钮，如图 8-4 所示。

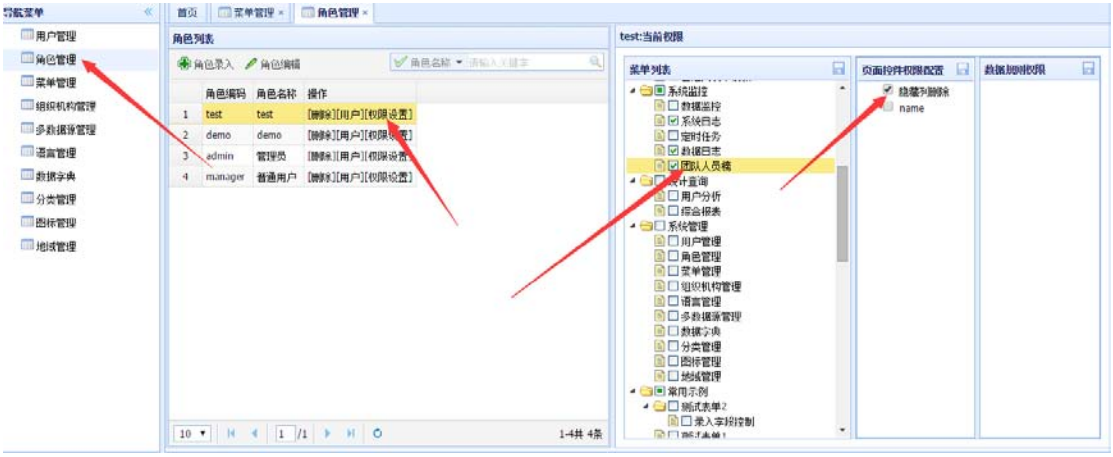


图 8-4 按钮权限分配

开发说明

在 JEECG 系统中，可以通过系统的全局变量配置来决定是否启用按钮权限。如下：

/resources/sysConfig.properties 中 button.authority.jeecg 参数值如下： true(开启按钮权限) false(关闭按钮权限) 当为 false 时默认拥有所有按钮权限，如（3）

DateGridTag 中根据系统的配置进行页面控件权限的控制：

（1）系统开启页面控件权限并且 DateGridTag 里面相关的页面控件操作有配置则根据配置做页面控件权限的控制；

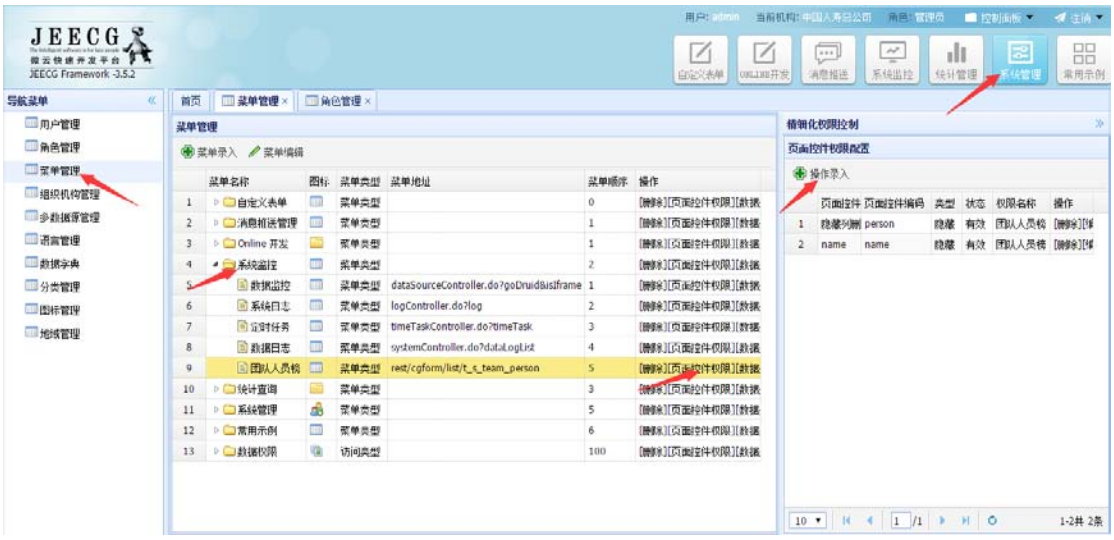
（2）系统开启页面控件权限但是 DateGridTag 里面相关的页面控件操作没有配置则不作页面控件操作的控制；

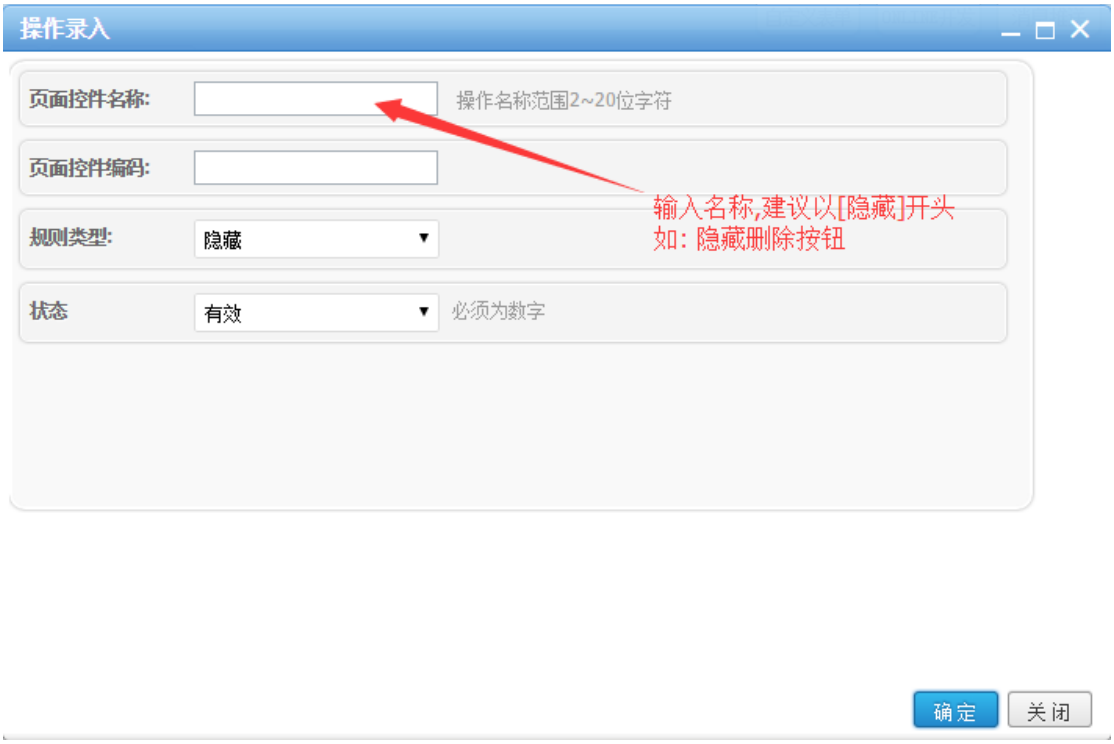
（3）系统关闭页面控件权限则所有的页面控件不做权限的控制；

（4）admin 用户页面控件权限不做限制。

操作步骤

① 页面控件权限设置：进入【系统管理】→【菜单管理】→【系统监控】，点击【团队人员榜】的【页面控件权限】



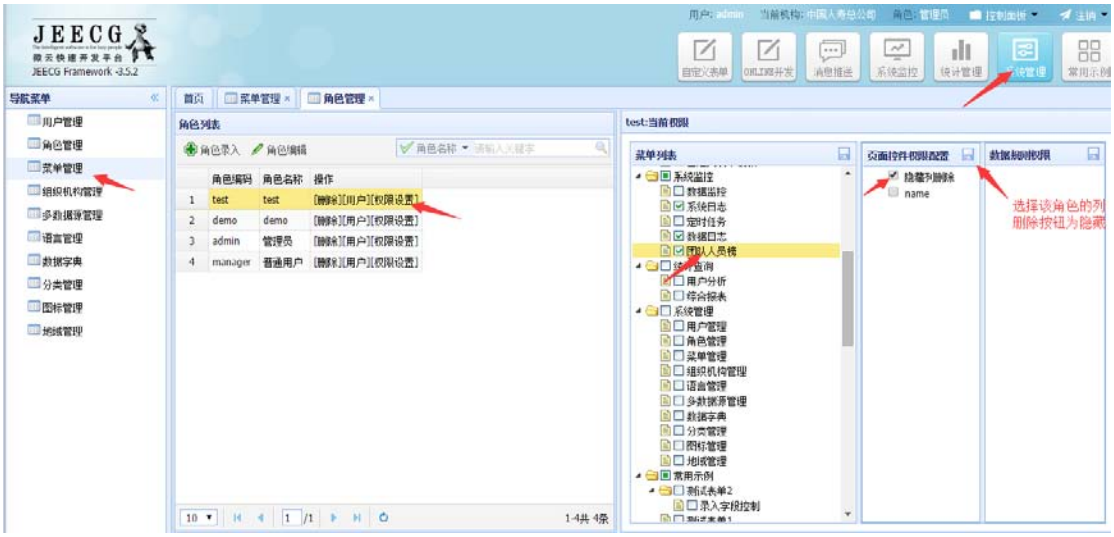


	页面控件名称	页面控件编码	类型	状态	权限名称	操作
1	隐藏列删除	person	隐藏	有效	团队人员榜	[删除][编辑]
2	name	name	隐藏	有效	团队人员榜	[删除][编辑]

② 代码中对按钮加入操作代码。

```
<t:datagrid name="tSTeamPersonList" checkbox="true" fitColumns="false" title="团队人员榜" actionUrl="tSTeamPersonController.do?datagrid" 
 <t:dgCol title="主键" field="id" hidden="true" queryMode="single" width="120"></t:dgCol>
 <t:dgCol title="创建人名称" field="createName" hidden="true" queryMode="single" width="120"></t:dgCol>
 <t:dgCol title="创建人登录名称" field="createBy" hidden="true" queryMode="single" width="120"></t:dgCol>
 <t:dgCol title="创建日期" field="createDate" formatter="yyyy-MM-dd" hidden="true" queryMode="single" width="120"></t:dgCol>
 <t:dgCol title="更新人名称" field="updateName" hidden="true" queryMode="single" width="120"></t:dgCol>
 <t:dgCol title="更新人登录名称" field="updateBy" hidden="true" queryMode="single" width="120"></t:dgCol>
 <t:dgCol title="更新日期" field="updateDate" formatter="yyyy-MM-dd" hidden="true" queryMode="single" width="120"></t:dgCol>
 <t:dgCol title="所属部门" field="sysOrgCode" hidden="true" queryMode="single" width="120"></t:dgCol>
 <t:dgCol title="所属公司" field="sysCompanyCode" hidden="true" queryMode="single" width="120"></t:dgCol>
 <t:dgCol title="名称" field="name" queryMode="single" width="120"></t:dgCol>
 <t:dgCol title="头像" field="imgSrc" image="true" queryMode="single" imageSize="imageSize(200,50)" width="120"></t:dgCol>
 <t:dgCol title="简介" field="introduction" queryMode="single" width="500"></t:dgCol>
 <t:dgCol title="加入时间" field="jionDate" formatter="yyyy-MM-dd" queryMode="single" width="120"></t:dgCol>
 <t:dgCol title="操作" field="opt" width="100"></t:dgCol>
 <t:dgDelOpt title="删除" url="tSTeamPersonController.do?doDel&id={id}" operationCode="person"/>
 <t:dgToolBar title="录入" icon="icon-add" url="tSTeamPersonController.do?goAdd" funname="add"></t:dgToolBar>
 <t:dgToolBar title="编辑" icon="icon-edit" url="tSTeamPersonController.do?goUpdate" funname="update"></t:dgToolBar>
 <t:dgToolBar title="批量删除" icon="icon-remove" url="tSTeamPersonController.do?doBatchDel" funname="deleteALLSelect"></t:dgToolBar>
 <t:dgToolBar title="查看" icon="icon-search" url="tSTeamPersonController.do?goUpdate" funname="detail"></t:dgToolBar>
 <t:dgToolBar title="导入" icon="icon-put" funname="ImportXls"></t:dgToolBar>
 <t:dgToolBar title="导出" icon="icon-putout" funname="ExportXls"></t:dgToolBar>
</t:datagrid>
```

③ 角色管理中对菜单设置按钮权限



④ 开启按钮权限

```
#DEV(生产模式)/PUB(开发模式)
sqlReadMode=PUB

#database type for spring jdbc
jdbc.url.jeecg=mysql

#按钮权限开关
button.authority.jeecg=true
```

sysConfig.properties文件的
button.authority.jeecg=true

⑤ 以角色为【test 用户】的账户登录系统。

11.3.4. 自定义页面控件权限

Jeecg 中，目前按页面控件权限设置，是通过对平台自己封装的标签（<t:dgFunOpt 等）进行设置。而在开发的过程中，有一些按钮标签是普通的<a href>或<button>形式的。对于这种普通开发者自定义页面控件的权限设置，目前 jeecg 也可以支持了。具体设置方法如下：

1. 给页面上的自定义页面控件增加 id 或 class 。

```
<a id='workerinputhref' href="#" class="easyui-linkbutton workerAddButton" p
                        onclick="add('基本信息录入','workerBaseController.do

 <a   href="#" class="easyui-linkbutton workerAddButton workerotherinput' pl
                        onclick="update('银行卡录入','workerBankController.

 <a    href="#" class="easyui-linkbutton workerAddButton workerotherinput'
                        onclick="update('教育经历录入','workerEduController.d
```

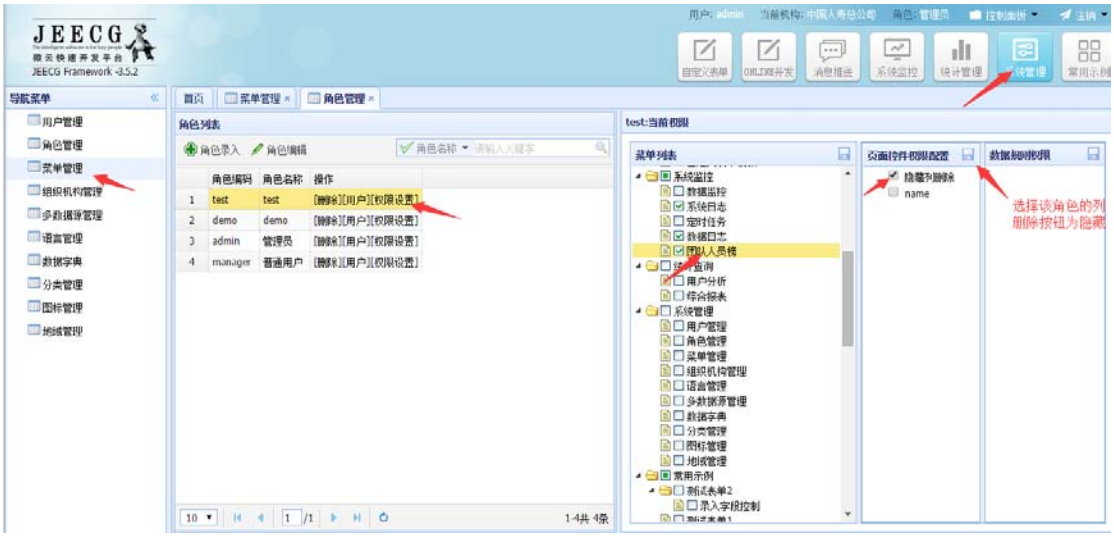
小提示：对于具有相同权限的多个按钮，可以设定一个共同的 class，将会更加便捷。

2. 将自定义页面控件的 id 或 class 设置到操作按钮中。

方式一： ID 设置

方式二： Class 设置

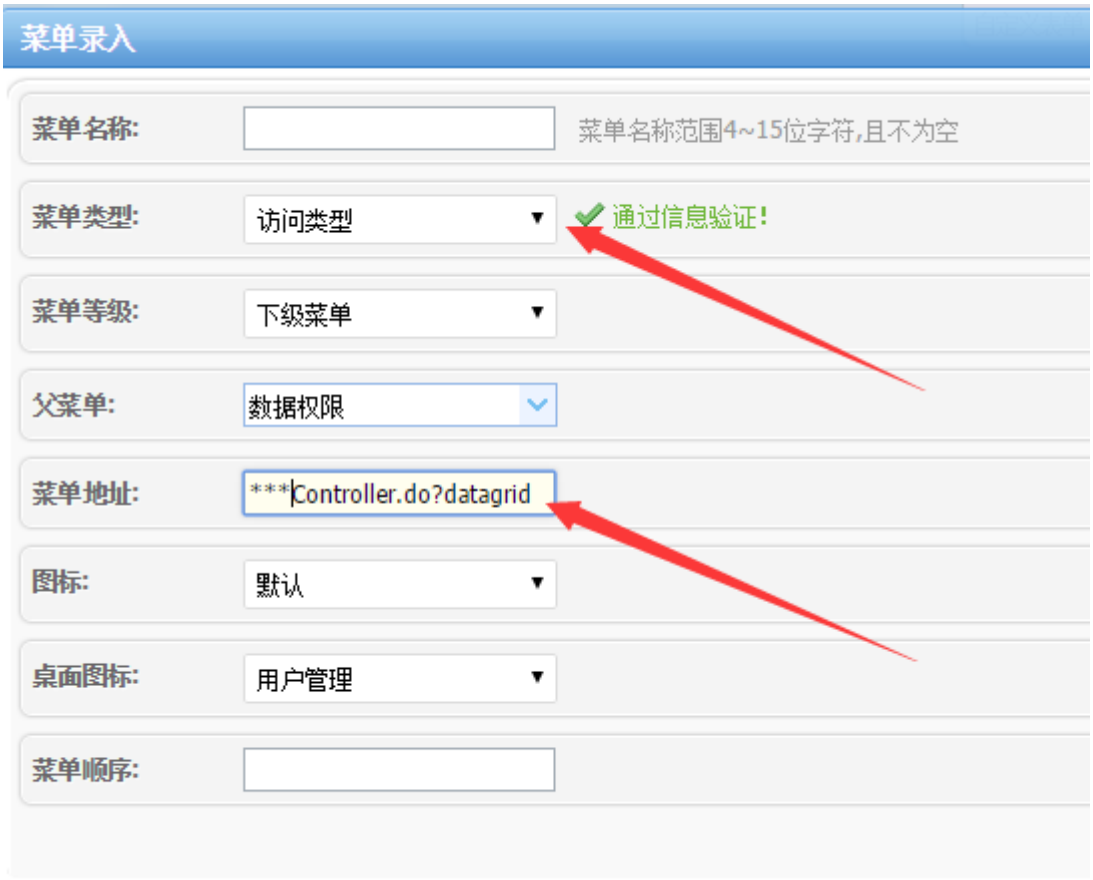
3. 在角色列表中，进行正常的权限设置就可以了

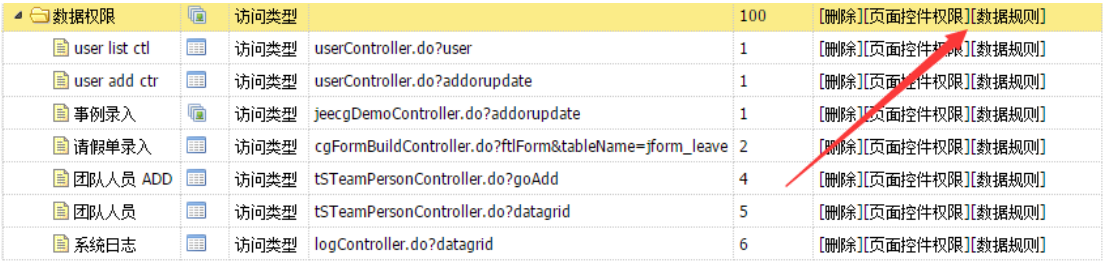


11.3.5. 数据权限

数据权限级别的权限对数据的请求访问进行控制，可设定数据权限规则，即定义数据查询过滤的规则，每个表中都有公司 ID、单位 ID 和用户 ID 等，可自由选择其一作为过滤条件。

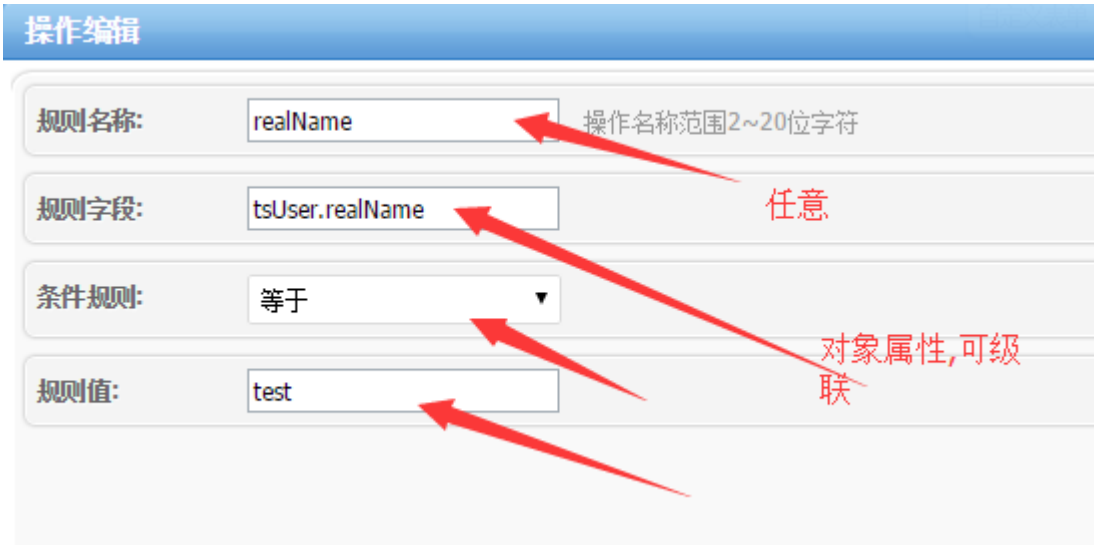
新建菜单项,选择菜单类型为访问类型,且菜单地址为具体获取数据的请求地址(一般为***Controller.do?datagrid),添加完成后,点击右侧的“数据规则”按钮进入页面规则列表页面。





数据权限		访问类型		100	[删除][页面控件权限][数据规则]
user list ctl		访问类型	userController.do?user	1	[删除][页面控件权限][数据规则]
user add ctr		访问类型	userController.do?addorupdate	1	[删除][页面控件权限][数据规则]
事例录入		访问类型	jeecgDemoController.do?addorupdate	1	[删除][页面控件权限][数据规则]
请假单录入		访问类型	cgFormBuildController.do?ftlForm&tableName=jform_leave	2	[删除][页面控件权限][数据规则]
团队人员 ADD		访问类型	tSTeamPersonController.do?goAdd	4	[删除][页面控件权限][数据规则]
团队人员		访问类型	tSTeamPersonController.do?datagrid	5	[删除][页面控件权限][数据规则]
系统日志		访问类型	logController.do?datagrid	6	[删除][页面控件权限][数据规则]

点击数据规则页面的“操作录入”按钮，弹出页面录入一条规则。



填写说明：

（1）规则名称：[字段名称]=[规则值]限制（可自定义）

（2）规则字段：[字段名称]

（3）条件规则：大于/大于等于/小于/小于等于/等于/包含/模糊/不等于

（4）规则值：指定值

例如：

规则名称：部门=00001

规则字段：department

条件规则：等于

规则值：00001

以上规则表示页面查询数据时将在 sql 中添加 department=‘00001’的条件，即按照部门=00001 进行了数据权限控制。

规则字段如需要使用对象级联的方式,需要在实体中配置对象映射关系。

11.3.6. 数据权限系统上下文变量

注意：规则字段处可以填写系统字段，包括：

sys_company_code 当前登录用户公司编号

sys_org_code 当前登录用户部门编号

sys_user_code 当前登录用户账号

sys_user_name 当前用户真实名称

sys_date 当前日期

sys_time 当前时间

写法如下：#{sysOrgCode}

针对列表配置数据权限过滤，不是列表加载页面的请求地址，而是数据加载的请求地址。

11.4. 菜单自动加载

11.4.1. 背景

配置**菜单**和**页面控件操作权限**是个很繁琐的工作，所以考虑采用系统自动加载方式

11.4.2. 设计思路

采用注释标签，在代码层标示菜单和菜单操作权限，系统启动的时候扫描整个工程看菜单是否已经配置到表里，如果没有，系统自动将配置菜单加载到系统表里

（默认配置的菜单都是一级的，需要用户手工调整菜单目录）

11.4.3. 具体实现

自动加载菜单开关

1. 配置文件

resources\sysConfig.properties

2. 参数

#auto scan menu flag true or false

auto.scan.menu.flag=true

说明：当不需要自动加载菜单的时候，改为 false，因为加载菜单会牺牲启动性能

3. 菜单标签

```
/**
 * 菜单注释标签
 * 系统启动自动加载菜单配置
 * Class级别
 */public@interfaceAutoMenu
```

参数说明：

参数名	说明	默认值
Name	菜单名称	

level	等级	0
url	菜单地址	
icon	图标	402880e740ec1fd70140ec2064ec0002
order	顺序	0

3. 菜单操作权限标签

```
/**
 * 菜单操作按钮注释标签
 * 系统启动自动加载菜单对应的操作权限
 * Method级别
 */public@interfaceAutoMenuOperation
```

参数	说明	默认值
name	操作名称	
code	操作码	
codeType	操作码类型 (Tag/Id/Css)	MenuCodeType.TAG
icon	图标	空
status	状态	0

11.4.4. 示例

1.菜单标签使用例子

```
@Controller
@RequestMapping("/jeecgDemoController")
@AutoMenu(name = "menu常用Demo", url =
"jeecgDemoController.do?jeecgDemo")
publicclass JeecgDemoController extends BaseController {
```

2.菜单操作标签使用例子

```
/**
 * 添加JeecgDemo例子
 *
 * @param ids
 * @return
 */
@RequestMapping(params = "save")
@ResponseBody
@AutoMenuOperation(name="添加",code = "add")
```

```
public AjaxJson save(JeecgDemo jeecgDemo, HttpServletRequest
request)
```

12. 多数据源

12.1. 多数据源背景

通常一个系统只需要连接一个数据库就可以了，Jeecg 数据源是配置在 spring-mvc-hibernate.xml 文件中，这种数据源我们叫做主数据源。但是在企业应用的开发中往往会和其他子系统交互，特别是对于一些数据实时性要求比较高的数据，我们就需要做实时连接查询，而不是做同步。这个时候就需要用到多数据源。

举个简单的例子某企业要做订单网上订单系统这里面就可以涉及到多个子系统的连接，比如：产品主数据的数据源，项目管理系统的数据源（项目可以产品订单）等多个不同数据库类似的数据源，他们可能是 ORACLE，SQL SERVER，MYSQL 等多种混合数据源。

12.2. 多数据源设计原理

多数据源的设计有多种方案，网上有很多是设计在 spring-mvc-hibernate.xml 文件中，通过在不同线程中 datasource 动态切换来实现，这样使得配置，程序太过复杂，所以 Jeecg 并没有使用这种方法。

Jeecg 多数据源设计很简单就是直接利用 springjdbc 连接，跟 Hibernate 或者 MyBatis 等持久化框架无关。就是我们最原始的类似 JDBC 的连接原理一样，只不过我们这里使用了 springjdbc 更加成熟的连接方法。

我们把多数据源的配置信息单独放在 t_s_data_source 表里维护管理。

12.3. 多数据源的使用与维护

在 TOMCAT 启动时添加到缓存(缓存机制查看类：InitListener)，以 t_s_data_source.key 字段为 key 值，以 t_s_data_source 的实体类为 value 放到 static Map 中。

系统管理-多数据源管理菜单可以实际对 t_s_data_source 表的维护在 Service 层即可以调用 org.jeecgframework.core.util.DynamicDBUtil 中的方法得到数据库连接并可以执行 SQL。

举例：

插入修改数据：DynamicDBUtil.update(‘SAP_DB’,’ delete from user ??’);

返回：int 。

查询单条数据： DynamicDBUtil .findOne(‘SAP_DB’,’ delete from user ??’)

返回：Map<String, Object> 。

查询数据列表：DynamicDBUtil.findList(‘SAP_DB’,’ delete from user ??’);

返回：List<Map<String, Object>> 。

13. 国际化

13.1. 国际化背景

没有国际化的框架是一个不完整的框架，特别在全球信息化的今天，国际化不再是鸡肋，而是在选择开发平台时必须首要的考试因素，特别在有些公司平台是否国际化具有一票否决要素，所以我们要搞国际化，而不是仅仅是简单的高大尚。

13.2. 国际化实现原理

国际听起来简单，但要框架中要做到与框架的无缝衔接，松耦合还是有一定难度的。有些系统有时不需要做国际化，在做国际化如果快速发现哪些没有被国际化的字段等，这些问题都开发者在开发过程中实际需要解决的问题，而这一切我们都已经帮你想到了。

用户在登录页面选择语言后点击登录后在 LoginControler.java 中 checkuser 方法中可以看到下面代码，当前语言会被放到 session 中。

```
if (req.getParameter("langCode")!=null) {
    req.getSession().setAttribute("lang", req.getParameter("langCode"));
}
```

MutiLangServiceImpl.java 中 有三个方法，分别介绍下作用：

initAllMutiLang() --- TOMCAT 启动时会被自动加载，把国际化的表内容加载到内存中，方便以后快速调用，即加载表 t_s_muti_lang 中所有记录并放在 Cache 中。

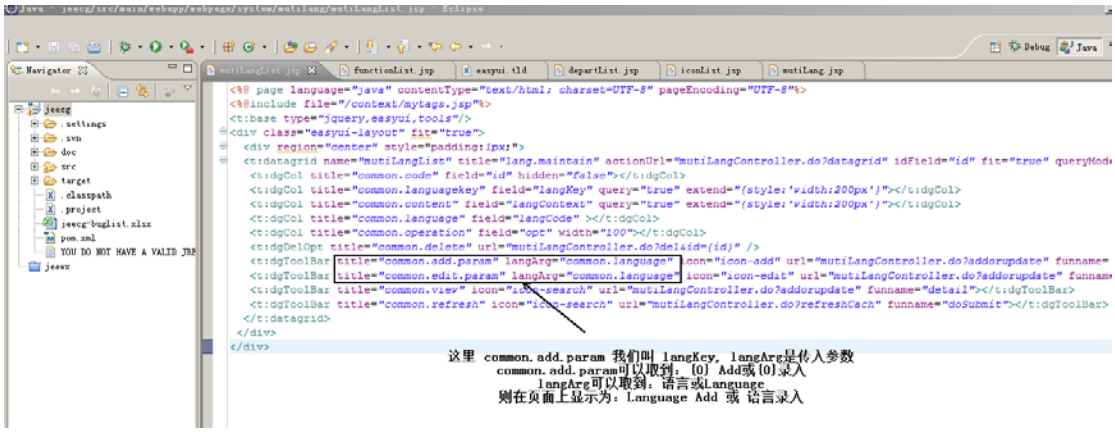
getLang(String lang_key) --- 传入需要国际化的 Lang Key 自动根据用户当前语言来得到国际化后的值，如在某处调用如： String value = getLang(“common.status”)，如果当前语言为中文则 value 为状态，英文为 Status

refleshMutiLangCach() --- 刷新 Cache，在对语言进行维护的时候，比如添加一个语言可以自动把新加的自动加入到 Cache 里，保证实时生效。

13.3. 国际化的使用场景

我们以国际化维护页面国际化为例

List 及 Tag 国际化



langKey 可以接受输入多个参数，举例如：

lang. congratulation.edit.success 对应中文是：恭喜你，{0}修改{1}。

则在页面上我们可以传入的 langArg 的参数样式为：

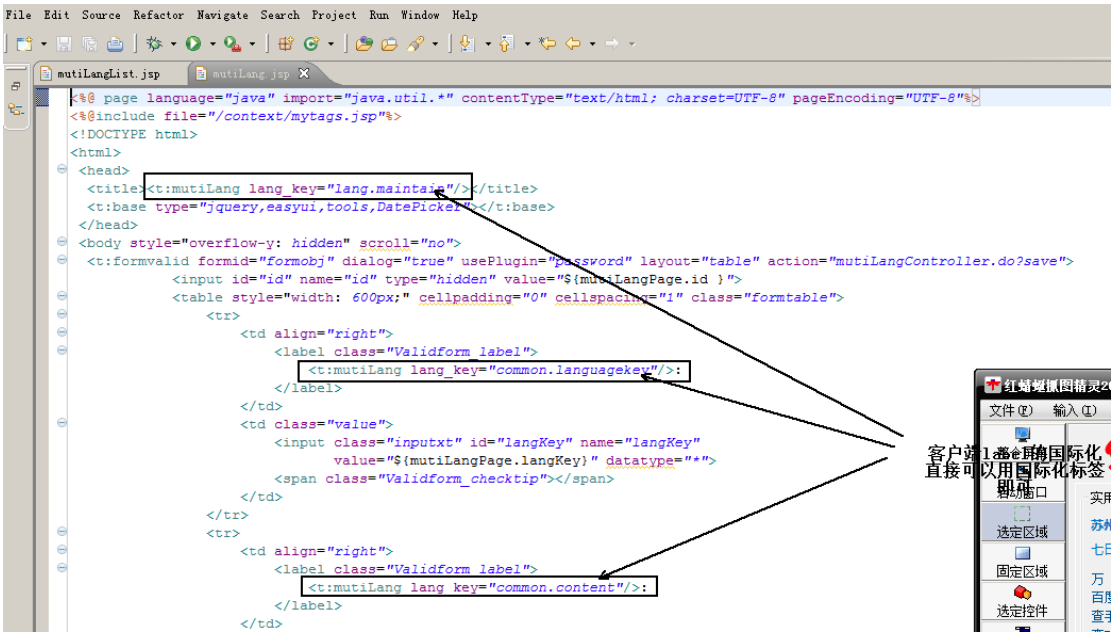
langArg=” lang.function.name, common.success”

lang.function.name 会替代{0}, common.success 会替代{1}

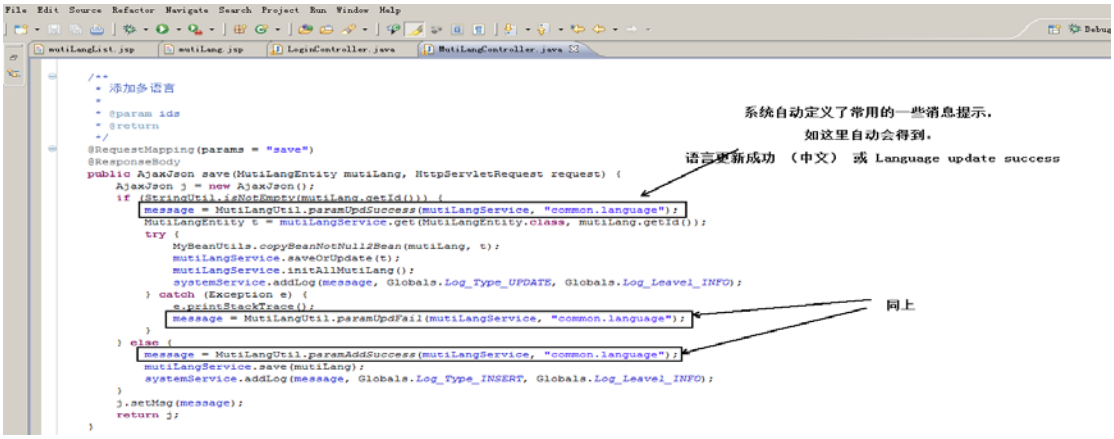
```
<div class="easyui-layout" fit="true">↓
<div region="center" style="padding: 1px;">↓
<t:datagrid sortName="createDate" sortOrder="desc" name="tablePropertyList" title="smart.form.config" fitColumns="false"↓
            actionUrl="cgFormHeadController.do?datagrid" idField="id" fit="true" queryMode="group" checkbox="true">↓
    <t:dgCol title="common.id" field="id" hidden="false"></t:dgCol>↓
    <t:dgCol title="table.type" field="jformType" replace="single.table_1,master.table_2,slave.table_3" query="true"></t:dgCol>↓
    <t:dgCol title="table.name" field="tableName" query="true" autocomplete="true" />↓
    <t:dgCol title="table.desc" field="content"></t:dgCol>↓
    <t:dgCol title="common.version" field="jformVersion"></t:dgCol>↓
    <t:dgCol title="is.tree" field="isTree" replace="common.yes_Y,common.no_N"></t:dgCol>↓
    <t:dgCol title="is.page" field="isPagination" replace="common.yes_Y,common.no_N"></t:dgCol>↓
    <t:dgCol title="sync.db" field="isDbSynch" replace="has.sync_Y,have.nosync_N" style="background:red;_N" query="true"></t:dgC
    <t:dgCol title="show.checkbox" field="isCheckbox" replace="common.yes_Y,common.no_N"></t:dgCol>↓
    <t:dgCol title="query.module" field="querymode"></t:dgCol>↓
    <t:dgCol title="common.createby" field="createBy" hidden="false"></t:dgCol>↓
    <t:dgCol title="common.createtime" field="createDate" formatter="yyyy/MM/dd"></t:dgCol>↓
    <t:dgCol title="common.updateby" field="updateBy" hidden="false"></t:dgCol>↓
    <t:dgCol title="common.updatetime" field="updateDate" formatter="yyyy/MM/dd"></t:dgCol>↓
```

Replace里面多语言

客户端 label 的国际化



Java 后台国际化



后台树的国际化

如果你需要对树做国际化，只要在表里把树的名称换成 lang_key 的值并且在树返回的最后一步加代码：MutiLangUtil.setMutiTree(treeGrids, mutiLangService);

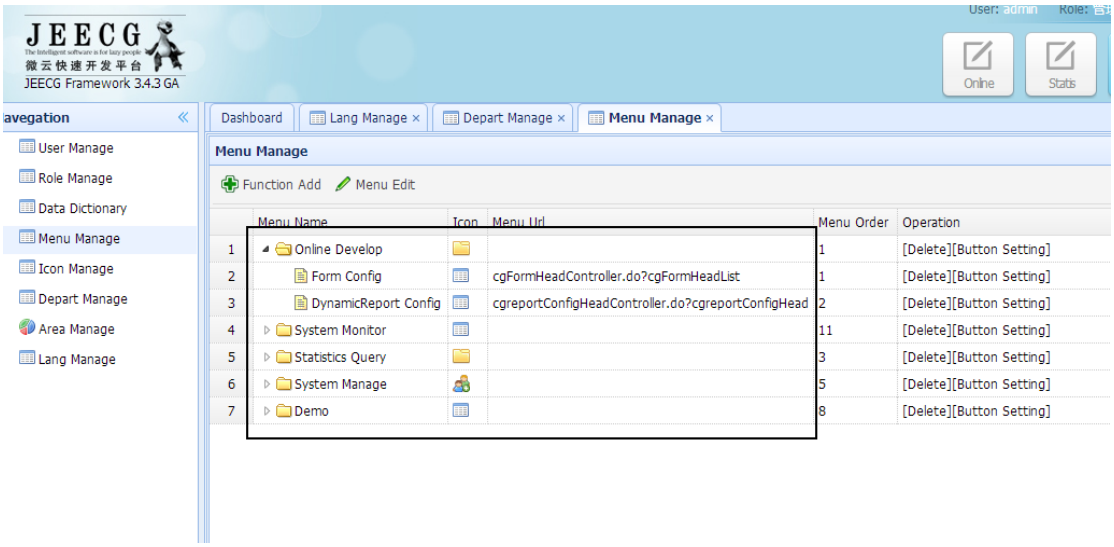
就可以实现，具体可以参考菜单管理：

```
        }
        cq.addOrder("functionOrder", SortDirection.asc);
        cq.add();
        List<TSFunction> functionList = systemService.getListByCriteriaQuery(
                cq, false);
        List<TreeGrid> treeGrids = new ArrayList<TreeGrid>();
        TreeGridModel treeGridModel = new TreeGridModel();
        treeGridModel.setIcon("TSIcon_iconPath");
        treeGridModel.setTextField("functionName");
        treeGridModel.setParentText("TSFunction_functionName");
        treeGridModel.setParentId("TSFunction_id");
        treeGridModel.setSrc("functionUrl");
        treeGridModel.setIdField("id");
        treeGridModel.setChildList("TSFunctions");
        // 添加排序字段
        treeGridModel.setOrder("functionOrder");
        treeGrids = systemService.treegrid(functionList, treeGridModel);

        MutiLangUtil.setMutiTree(treeGrids, mutiLangService);
        return treeGrids;
    }

    /**
     * 权限列表
     */
```

把树的结构放到这个方法里执行就OK了



Js 国际化

由于 Js 文件中无法用国际化标签，所以只能在 src\main\webapp\plug-in\mutiLang 目录中添加 us.js 及 zh-cn.js，这个文件已经在 BaseTag 中第一个默认引入了。

13.4. 国际化语言维护

系统管理菜单中语言管理就可以对语言进行维护管理，注意不要添加已经存在的语言内容。若是有重复的内容，系统在添加的时候会自动提示。

13.5. lang_key 的命名规范

- 系统已经有的国际化的就不用重复添加了，如: common.status，对应 状态，如果有状态直接使用即要，不要再添加相同名称不同 lang_key 的进去。
- 如果碰到国际化表中没有的，则需要自己添加。

  lang_key 在 t_s_mutilang 表中命名的格式建议是 xxx.xxx（或者有更多的点）
- 如果是不常用的，如某个模块中有个标签叫“学生”，则命名时可以叫：lang.student
- 如果是一些公共的，则命名时用 common 开头，如“操作”可能在系统各处都有可能用到，则命名可以叫：common.operation
- 不用要一个单词来命名作为 lang_key，如 student.
- 如果是一个句子 lang_key 就写下这个句子，单词间用点来分开，如：

  common.please.select.one.record.to.edit 对应 Please select one record to edit
- 如果是一个句子 lang_context 首字母需要大写，其他小写，按正常英文句子的来拼写，如：Please select one record to edit， 而不是 Please Select One Record To Edit.
- 如果不是句子，如表头上显示，则 lang_context 可以全部首字母大写，如: User Name, Role Name
- 前后命名要一致，比如部门有的人喜欢用简写如:depart，哪后面注意就全部用简写，不要一会出现 depart，一会出现 department 很不协调。也不要一会出现 desc，一会出现 description.
- 单词不要出现拼写错误。
- 翻译要准确，不要出现无关内容。比如有某个网页上的标题上中文是：学生添加，你可以用 lang.student.add，有的人发现这是个标题就可能会写成 lang.title.student.add，title 这是这个出现的位置，请不要加这样的位置信息。
- 所有 List 页面(特例除外)的添加，修改都是用以下风格，对应模块修改 langArg 的值

  <t:dgToolBar title="common.add.param" langArg="common.language"

  <t:dgToolBar title="common.edit.param" langArg="common.language"

翻译过来就是 模块名+录入(编辑)，比如：语言录入，用户录入等，不要只放“录入”以保证所有页面的页格一致。

13.6. 国际化后的问号处理

那是在提醒你该 lang_key 没有做国际化，你只要在语言维护中添加即可，问号的目的就是个提醒，可以让你一眼就看出哪些没有做国际化。如果你讨厌看到问号同样把 common.notfind.langkey 对应的 lang_context 会值由 ？改成空

13.7. 不想看到 lang_key

公司目前只做国内项目，不想看到 lang_key，此时还是可以按原来一样写代码，比如：

<t:datagrid name="departList" title="common.department.list" 改成

<t:datagrid name="departList" title="部门列表"

再把 common.notfind.langkey 对应的 lang_context 会值由 ？改成空就可以了。

14. 附录

14.1. UI 库常用控件参考示例

序号	控件	解决方案	参考示例
1	datagrid 数据列表，字段采用数据字典显示文本	<t:dgCol title="状态" sortable="true" field="status" replace="正常_1,禁用_0,超级管理员_-1"></t:dgCol>	WebRoot/webpage/system/user/userList.jsp
2	树列表展现		参考示例[菜单管理]：WebRoot/webpage/system/function/functionList.jsp
3	POPUP 实现	<t:choose hiddenName="roleid" hiddenid="id" url="userController.do?roles" name="roleList" icon="icon-choose" title="角色列表" textname="roleName" isclear="true"></t:choose>	/WebRoot/webpage/system/user/user.jsp
4	下拉菜单实现		WebRoot/webpage/system/user/user.jsp

5	radio 控件		WebRoot/webpage/system/user/user.jsp
6	数据列表展示		WebRoot/webpage/system/user/userList.jsp
7	常用组件 DEMO 地址	上传/表单验证/Excel 导入/Excel 导出/ 页面不同弹出方式/树界面展示/自动补全/一对多示例/tabs 切换	/WebRoot/webpage/demo/*
8	下拉菜单多级联动		
9	一对多明细行加下拉项		
10	datagrid 数据列表，时间字段格式化		
11	数据行全选		
12	重复校验		

14.2. 开发技巧：采用 IFrame 打开页面

目前在 JEECG 开发平台中，为了提高 easyui 的性能，tab 的打开采用 href 方式，但是 href 方式存在如下问题：

1.href 只加载目标 URL 的 html 片段

这个特性是由 jQuery 封装的 ajax 请求处理机制所决定的，所以目标 URL 页面里不需要有 html，

head,body 等标签，即使有这些元素，也会被忽略，所以放在 head 标签里面的任何脚本也不会被引入或者执行。

2.短暂的页面混乱：

href 链接的页面比较复杂的时候，easyui 对其渲染往往需要一个较长的过程

当加载的页面布局较为复杂，或者有较多的 js 脚本需要运行的时候，就不好处理了。

所以，综合考虑，如果页面样式、js 简单就采用系统默认的 href 方式打开 tab 页。

如果页面复杂，不好拆分，则采用 iframe 方式打开 tab。采用 ifrme 方式，需要在配置菜单的时候，加上&isIframe 标识，如下所示：

```
dataSourceController.do?goDruid&isIframe
```

需要注意：改为 iframe 方式的页面需要在 head 中追加：

```
<t:base type="ckeditor,jquery,easyui,tools"></t:base>
```

14.3. 开发技巧：组合查询实现方法

简述：代码生成器默认生成的查询方式为单字段查询，如果想实现字段组合查询，需要采用如下方式。

实现步骤：

第一步：设置 dategrid 字段查询属性 query="true"

第二步：对应 query="true"的 dategrid 字段设置查询字段组件

```
<input type="text" name="userName" id="userName" style="width: 80px"/>
```

第三步：设置查询按钮

```
<a href="#" class="easyui-linkbutton" iconCls="icon-search" onclick="userListsearch()">查询</a>
```

注意点：

1. 这种写法 t:dgToolBar 这个标签不能使用，不然会有冲突，查询 form 显示不出来;

2. 查询函数的名字规则"[dategrid 组件 name]search()"

[1]. dategrid 组件 name

```
<t:dategrid name="userMe"
```

[2]. 组合查询 DIV

```
<div id="userMetb"
```

[3]. 查询按钮对应的 js 方法

```
<a href="#" class="easyui-linkbutton" iconCls="icon-search" onclick="userMesearch()">查询</a>
```

参考示例：/WebRoot/webpage/system/user/userList.jsp

示例代码如图 12-1 所示：

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<%@include file="/context/mytags.jsp"%>
<t:datagrid name="userList" title="用户管理" actionUrl="userController.do?datagrid" idField="id">
      <t:dgCol title="编号" field="id" hidden="false"></t:dgCol>
      <t:dgCol title="用户名" sortable="false" field="userName" query="true" width="20"></t:dgCol>
      <t:dgCol title="部门" field="TSDepart_departname"></t:dgCol>
      <t:dgCol title="真实姓名" field="realName" query="true"></t:dgCol>
      <t:dgCol title="状态" sortable="true" field="status" replace="正常_1,禁用_0,超级管理员_-1"></t:dgCol>
      <t:dgCol title="操作" field="opt" width="100"></t:dgCol>
      <t:dgFunOpt funname="szqm(id)" title="设置签名" />
      <t:dgDelOpt title="删除" url="userController.do?del&id={id}&userName={userName}" />
</t:datagrid>
<div id="userListtb" style="padding: 3px; height: 25px">
 <div style="float:left;">
  <a href="#" class="easyui-linkbutton" plain="true" icon="icon-add" onclick="add('用户录入','userController.do?
addorupdate')">用户录入</a>
  <a href="#" class="easyui-linkbutton" plain="true" icon="icon-add" onclick="update('用户编辑','userController.do?
addorupdate','id')">用户编辑</a>
 </div>
 <div align="right">
    用户名:<input type="text" name="userName" id="userName" style="width: 80px"/>
    真实姓名:<input type="text" name="realName" id="realName" style="width: 80px"/>
   <a href="#" class="easyui-linkbutton" iconCls="icon-search" onclick="userListsearch()">查询</a>
 </div>
</div>a
```

图 12-1 组合查询示例代码

14.4. Formvalid 新增属性 tiptype 的使用

Formvalid 中的 tiptype 用来定义提示信息的显示方式，一共有 4 种取值，在其官方的说明中，不同取值的含义如下：

取值	含义
1	自定义弹出框提示；
2	侧边提示(会在当前元素的父级的 next 对象的子级查找显示提示信息的对象，表单以 ajax 提交时会弹出自定义提示框显示表单提交状态)；
3	侧边提示(会在当前元素的 siblings 对象中查找显示提示信息的对象，表单以 ajax 提交时会弹出自定义提示框显示表单提交状态)；
4	侧边提示(会在当前元素的父级的 next 对象下查找显示提示信息的对象，表单以 ajax 提交时不显示表单的提交状态)

在 jeecg 中，tiptype 的属性配置代码如下：

```
<t:formvalid formid="formobj" dialog="true" usePlugin="password" layout="table"
tiptype="1" action="jeecgOrderMainController.do?save">
```

与官方的用法不同的是，JEECG 中对取值为 1 时的样式以及校验方式进行了改造，官方版是在提交时才给出提示，而 JEECG 中是在 onblur 的时候就会提示，当输入正确后，1 秒中后会自动消失。

注：<t:formvalid>标签中不写 tiptype 时默认为 4.即侧边显示。

使用建议：单表可以不用给定 tiptype 属性，即使用默认的侧边校验，主从表的数据校验给定 tiptype="1"。

单表和主从表的数据校验提示效果分别如图 12-2 和图 12-3 所示。

图 12-2 单表使用侧边提示方式

图 12-3 主从表使用弹出提示方式

14.5. 使用 toolbar 自定义 js 参数规则

第一步：定义按钮

```
<t:dgToolBar title="JS增强" icon="icon-edit"
url="cgFormHeadController.do?jsPlugin"
funname="jsPlugin"></t:dgToolBar>
```

第二步：定义js方法

三个参数说明：
1.三个参数缺一不可
2.三个参数顺序不能变
3.有且只有三个参数
4.id为datagrid的name属性

```
function jsPlugin(title,url,id){
var rowData = $('#'+id).datagrid('getSelected');
if (!rowData) {
tip('请选择编辑项目');
return;
}
url += '&id='+rowData.id;
                $.dialog({
content: "url:"+url,
```

```
lock : true,
title:"JS增强编辑["+rowData.tableName+"-"+rowData.content+"]",
opacity : 0.3,
width:900,
height:500,
cache:false,
    ok: function(){
iframe = this.iframe.contentWindow;
          iframe.goForm();
return false;
},
    cancelVal: '关闭',
    cancel: true /*为true等价于function(){}*/
});
}
```

14.6. 表单字段重复校验方法

目的：实现通用表单字段重复校验，

例如:部门管理模块，部门名称重复校验

```
     <input name="departname" class="inputxt"
value="${depart.departname }" validType="t_s_depart,departname,id"
datatype="s3-10">
```

1）代码配置

给input标签，增加validType属性，格式如：t_s_depart,departname,id 即（数据表名称、对应的数据库字段、业务实体的隐藏域主键的Id属性）

2）消息提示方式，两种方式

[1].提示弹出层：如下所示：给t:formvalid 增加tiptype="1" 属性

```
    <t:formvalid formid="formobj" tiptype="1" layout="table"…
```

[2].提示信息在文本框后面提示

不需要给t:formvalid 增加任何属性。

15. MiniDao 介绍

15.1. MiniDao 简介及特征

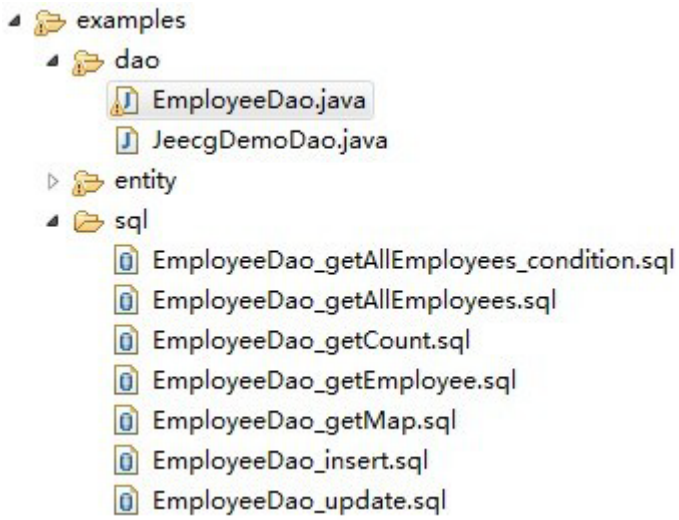
MiniDao 是 Jeecg 自己的持久化解决方案，具备了 Hibernate 实体维护和 Mybaits SQL 分离的两大优势。具有以下特征:

- 1.O/R mapping 不用设置 xml，零配置便于维护

- 2. 不需要了解 JDBC 的知识
- 3. SQL 语句和 java 代码的分离
- 4. 可以自动生成 SQL 语句
- 5. 接口和实现分离，不用写持久层代码，用户只需写接口，以及某些接口方法对应的 sql 它会通过 AOP 自动生成实现类
- 6. 支持自动事务处理和手动事务处理
- 7. 支持与 hibernate 轻量级无缝集成
- 8. MiniDao 整合了 Hibernate+mybatis 的两大优势，支持实体维护和 SQL 分离
- 9. SQL 支持脚本语言

※向下兼容 Hibernate 实体维护方式,实体的增删改查 SQL 自动生成

15.2. 接口和 SQL 文件对应目录



第一步：接口定义[EmployeeDao.java]

```
@MiniDao
public interface EmployeeDao {
 @Arguments("employee")
 public List<Map> getAllEmployees(Employee employee);

 @Arguments("empno")
 Employee getEmployee(String empno);
```

```
    @Arguments({"empno","name"})
    Map getMap(String empno,String name);

    @Sql("SELECT count(*) FROM employee")
    Integer getCount();

    @Arguments("employee")
    int update(Employee employee);

    @Arguments("employee")
    void insert(Employee employee);
}
```

第二步：SQL文件[EmployeeDao_getAllEmployees.sql]

```
SELECT * FROM employee where 1=1
<#if employee.age ?exists>
and age = :employee.age
</#if>
<#if employee.name ?exists>
and name = :employee.name
</#if>
<#if employee.empno ?exists>
and empno = :employee.empno
</#if>
```

15.3. MiniDao 接口配置

```
<!-- 注册 MiniDao 接口 -->
<bean class="org.jeecgframework.minidao.factory.MiniDaoBeanFactory">
<property name="packagesToScan">
<list>
<value>examples.dao.*</value>
```

```
</list>
</property>
</bean>
```

15.4. 测试代码

```
public class Client {
public static void main(String args[]) {
    BeanFactory factory = new ClassPathXmlApplicationContext(
            "applicationContext.xml");

    EmployeeDao employeeDao = (EmployeeDao) factory.getBean("employeeDao");
    Employee employee = new Employee();
    List<Map> list =  employeeDao.getAllEmployees(employee);
    for(Map mp:list){
        System.out.println(mp.get("id"));
        System.out.println(mp.get("name"));
        System.out.println(mp.get("empno"));
        System.out.println(mp.get("age"));
        System.out.println(mp.get("birthday"));
        System.out.println(mp.get("salary"));
    }
}
}
```

15.5. 环境搭建

环境：Spring 3.X,Hibernate 3 以上

MiniDao 依赖：org.jeecgframework.minidao-1.2.1.jar

15.5.1. MiniDao 与 Spring 集成

第一步：新建MiniDao的spring配置文件

文件名：spring-minidao.xml（可以自定义），只要让 spring 扫描到这个文件即可。

扫描方法一：

在 web.xml 中的 spring 监听器中扫描规则中包含 spring-minidao.xml。

```
<context-param>
<param-name>contextConfigLocation</param-name>
<param-value>classpath:spring-*.xml</param-value>
</context-param>
```

扫描方法二：

在 spring 的配置文件中引入 **spring-minidao.xml**。

```
<import resource="classpath*:spring-minidao.xml" />
```

MiniDao 配置详解

MiniDao 对 springJdbc 的支持（MiniDao 的核心基础配置）

```
<!-- MiniDao 动态代理类 -->
<bean id="miniDaoHandler"
class="org.jeecgframework.minidao.aop.MiniDaoHandler">
<!-- springjdbc，必须依赖 -->
<property name="jdbcTemplate" ref="jdbcTemplate"></property>
</bean>
```

```
<!-- 注册 MiniDao 接口 -->
<bean class="org.jeecgframework.minidao.factory.MiniDaoBeanFactory">
<property name="packagesToScan">
<list>
<!-- 可以以包名注册，自动扫描包下面的 Dao 接口 -->
<value>jeecg.cgreport.dao.*</value>
<value>jeecg.cgform.dao.*</value>
</list>
</property>
```

```
        </bean>
```

15.5.2. MiniDao 与 Hibernate 集成

前提是 Hibernate 已经由 Spring 容器管理，在 MiniDao 的配置文件中直接加入 Hibernate 支持即可。

```
<!-- Hibernate MiniDao -->
    <bean id="genericBaseCommonDao"
<!-- 对 hibernate 实体通用操作的实现类 -->
class="org.jeecgframework.minidao.hibernate.dao.impl.GenericBaseCommonDao">
<!-- hibernate 的 sessionFactory -->
        <property name="sessionFactory">
            <ref bean="sessionFactory" />
        </property>
    </bean>
```

16. Excel 导入导出

Excel 的导入导出抽取通用功能,简化大家对 POI 的操作,对实体对象进行简单的注解配置就可以完成导入导出,模板的使用更是可以让打造漂亮的 Excle 报表,从而使大家从重复的工作中解脱出来,更加关注与业务的处理.

16.1. 注解介绍

注解名	作用对象	描述	是否必须
Excel	字段	对 Excel 字段的 cell 属性设置	是
ExcelCollection	字段	对集合对象进行标记表示一对多导出	否
ExcelTarget	***实体***	唯一标识(建议设置)	否
ExcelIgnore	字段	导出是忽略这个对应(避免无限循环)	否
ExcelEntity	字段	导出的对象,即关联属性	否

应用实例:

```java
@ExcelTarget(id="courseEntity")
public class CourseEntity implements java.io.Serializable {
    /**主键*/
    private java.lang.String id;
    /**课程名称*/
    @Excel(exportName="课程名称",orderNum="1",needMerge=true)
    private java.lang.String name;
    /**课程名称*/
    @ExcelIgnore
    private CourseEntity cname;
    /**老师主键*/
    @ExcelEntity()
    private TeacherEntity teacher;

    @ExcelCollection(exportName="选课学生",orderNum="4")
    private List<StudentEntity> students;
```

主要属性介绍

ExcelTarget: id 设置导出 Excel 的 ID,字段可以根据 ID 判断是否需要导出;

Excel|ExcelEntity|ExcelCollection:exportName 设置导出的名称显示在 Excel 的表头

同时可以填写如课程_courseEnity 这样只有在导出 CourseEntity 这个对象的时候才导出,不标记 ID 视为都导出

Excel| ExcelCollection:orderNum 设置这个字段的顺序,既所在的列数,默认是按照字段的顺序排序

Eexcel|exportConvertSign**,Excel|** importConvertSign**,Excel|** imExConvert 这三个功能都是 Excel 注解的属性,他们的作用也是相似,就是你对直接重数据拿出的数据进行处理,即自定义处理函数,第一个在导出时使用,第二个导入时使用,如果导入导出都要使用可以直接使用第三个,默认是 0 不使用设置为 1 就使用例子

Sex 这个大家经常用,一般数据是 0,1 但是显示需要男女

```java
/**学生性别*/
@Excel(exportName="学生性别",imExConvert=1)
private java.lang.String sex;
```

这样我们在导入导出都使用我们自己定义的函数而不是 get,set

```java
public String convertGetSex(){
    return this.sex.equals("0")?"男生":"女生";
}
public void convertSetSex(String sex ){
    this.sex = sex.equals("男生")?"0":"1";
}
```

自定义函数的规则是普通的 get,set 函数前面加 convert

Excel|exportType 导出类型 1 是文本 2 是图片,3 是函数默认是文本

当导出图片时我们最好使用 exportFieldWidth 和 exportFieldHeight 设置列的高和宽

16.2. Excel 导入

导出工具类 **ExcelImportUtil**

提供两个函数,都是 List 返回集合

1.处理文件

```
public static Collection<?>importExcel(File file, Class<?> pojoClass,
                    ImportParams params) {}
```

2.处理流

```
public static    Collection<?>importExcelByIs(InputStream inputstream,
                    Class<?> pojoClass, ImportParams params){}
```

导入参数设置 **ImportParams**

根据我们导入的模板进行参数设置

现在我们以课程为例进行导入先看下模板

	课程名称	老师姓名	老师照片	选课学生	
1					
2				学生姓名	学生性别
3	海贼王	路飞		卓洛	男生
4				山治	男生

ImportParams|titleRows 表格标题行数,默认 0 我们没有标题所以也是 0

ImportParams|secondTitleRows 这个就是表头列数,我们这里有 2 列所以设置为 2

ImportParams|keyIndex 主键列,面向一对多,比如课程对学生,我们的课程名称就是主键,所以这个 keyIndex 就是 0

ImportParams|needSave 是否需要保存,默认不需要

ImportParams|saveUrl 如果保存的上传的 Excel 保存的路径,默认是如 TestEntity 这个类保存路径就是 upload/excelUpload/Test/yyyyMMddHHmss_*****保存名称上传时间_五位随机数

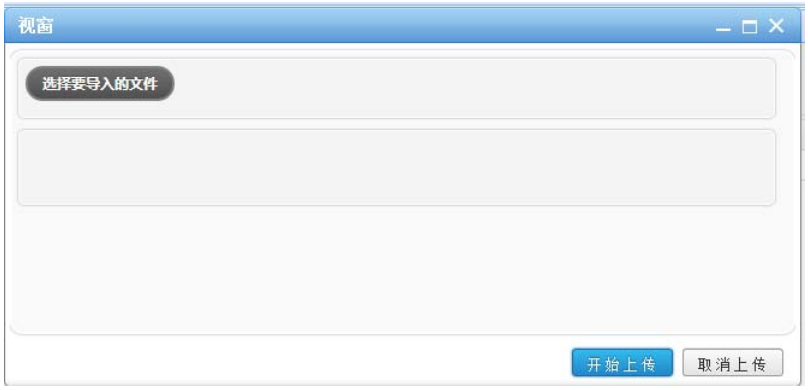
例子:课程的导入

步骤一对需要导入的对象进行注解

```java
@ExcelTarget(id="courseEntity")
public class CourseEntity implements java.io.Serializable {
    /**主键*/
    private java.lang.String id;
    /**课程名称*/
    @Excel(exportName="课程名称",orderNum="1",needMerge=true)
    private java.lang.String name;
    /**课程名称*/
    @ExcelIgnore
    private CourseEntity cname;
    /**老师主键*/
    @ExcelEntity()
    private TeacherEntity teacher;

    @ExcelCollection(exportName="选课学生",orderNum="4")
    private List<StudentEntity> students;
```

步骤二,添加上传页面



步骤三:后台进行接收处理函数

根据我们的模板我们创建相应的对象如:

```java
ImportParams params = new ImportParams();
params.setTitleRows(0);
params.setSecondTitleRows(2);
params.setNeedSave(true);
```

然后调用 Util 就可以返回我们需要的 list 的,之后进行处理就可以了

```java
List<CourseEntity> listCourses = (List<CourseEntity>)
        ExcelImportUtil.importExcelByIs(file.getInputStream(),
                CourseEntity.class,params);
```

导入就完成了

16.3. Excel 导出

导出工具类 **ExcelExportUtil**

提供两个函数

//创建多个 Sheet

public static HSSFWorkbook exportExcel(List<Map<String, Object>> list) {}

//创建一个 sheet

public static HSSFWorkbook exportExcel(ExcelTitle entity,

Class<?> pojoClass, Collection<?> dataSet){}

导出参数类 **ExcelTitle** 设置导出的标题

ExcelTitle|title 导出 Excel 的标题

ExcelTitle|secondTitle 导出 Excel 的第二标题

ExcelTitle|sheetName 导出的 Sheet 的名称

ExcelTitle|color 导出的 Excel 的标题和第二标题的背景色

ExcelTitle|headerColor 导出的 Excel 的属性行的背景色

下面我们利用上面导入的注解,进行导出

步骤一:在界面添加导出工具:

```
<t:dgToolBar title="导出Excel" icon="icon-search" onclick="courseListExportXls();" ></t:dgToolBar>
```

```
function courseListExportXls() {
    JeecgExcelExport("courseController.do?exportXls","courseList");
}
```

JeecgExcelExport 提供了代入查询条件的功能,第一个参数是导出 url,第二个是 datagrid 的名称.

步骤二后台建立函数,首先查询出来 list 然后

```
List<CourseEntity> courses = this.courseService.getListByCriteriaQuery(cq,false);
workbook = ExcelExportUtil.exportExcel(new ExcelTitle("课程列表", "导出人:Jeecg",
        "导出信息"), CourseEntity.class, courses);
fOut = response.getOutputStream();
workbook.write(fOut);
```

通过工具类转换成 workbook 输出到前台

效果

16.4. Excel 模板导出

16.4.1. 模板参数规则

参数设置规则 {{key}} key 可以支持 a.b.c 支持 map 或者 enity

如:{{jeecg}},{{obj.test}}

模板 1:

{{month}}份部门工资汇总					
类 别		人数	金额(元)	本年累计金额（元）	备注
后勤部门		{{i1.per}}	{{i1.mon}}	{{i1.summon}}	
技术部	企业部	{{i2.per}}	{{i2.mon}}	{{i2.summon}}	
	移动部	{{i3.per}}	{{i3.mon}}	{{i3.summon}}	
	GIS部	{{i4.per}}	{{i4.mon}}	{{i4.summon}}	
市场部	大客户部	{{i5.per}}	{{i5.mon}}	{{i5.summon}}	
	个人市场部	{{i6.per}}	{{i6.mon}}	{{i6.summon}}	
行政部		{{i7.per}}	{{i7.mon}}	{{i7.summon}}	
合 计		0	0	0	

模板 2

课程测试				
			导出日期{{year}}	
课程名称	老师姓名	老师照片	选课学生	
			学生姓名	学生性别
总课程: {{sunCourses}}	总老师:{{obj.name}}			

其中模板一是统计模板,模板二同时要输出详情

可以把多个模板放到一个 Excel 里面,倒是只要指定 Sheet 序号就可以了

16.4.2. 模板导出

工具类依然是 ExcelExportUtil 同样两个方法

//导出使用模板同时使用集合导出多列

public static Workbook exportExcel(TemplateExportParams params,

Class<?> pojoClass, Collection<?> dataSet, Map<String, Object> map){}

//只导出模板

public static Workbook exportExcel(TemplateExportParams params,

Map<String, Object> map){}

TemplateExportParams 模板参数

emplateUrl 模板路径

sheetName 导出 sheet 名称

sheetNum 导出 sheet 序号,默认是 0

使用案例(之前的步骤同 Excel 导出)

第一个是导出,我们使用模板一,值进行值替换这里我们模拟一个 map

```
Map<String,Object> temp;
for(int i = 1;i<8;i++){
    temp = new HashMap<String, Object>();
    temp.put("per", i*10);
    temp.put("mon", i*1000);
    temp.put("summon", i*10000);
    map.put("i"+i, temp);
}
workbook = ExcelExportUtil.exportExcel(
        new TemplateExportParams("export/template/exportTemp.xls",1),map);
fOut = response.getOutputStream();
workbook.write(fOut);
```

看一下输出结果:格式依然在

10份部门工资汇总					
类 别		人数	金额(元)	本年累计金额（元）	备注
后勤部门		10	1000	10000	
技术部	企业部	20	2000	20000	
	移动部	30	3000	30000	
	GIS部	40	4000	40000	
市场部	大客户部	50	5000	50000	
	个人市场部	60	6000	60000	
行政部		70	7000	70000	
合 计		0	0	0	

第二个导出,同时导出详情(使用第二个模板)

```
Map<String,Object> map = new HashMap<String, Object>();
map.put("year", "2013");
map.put("sunCourses", courses.size());
Map<String,Object> obj = new HashMap<String, Object>();
map.put("obj", obj);
obj.put("name", courses.size());
workbook = ExcelExportOfTemplateUtil.exportExcel(new TemplateExportParams("export/template/exportTemp.xls"),
        CourseEntity.class, courses,map);
fOut = response.getOutputStream();
workbook.write(fOut);
```

这里我们统计了课程数量教师人数,同时也输出了课程详情,

课程测试				
			导出日期2013	
课程名称	老师姓名	老师照片	选课学生	
			学生姓名	学生性别
总课程: 1	总老师:1			
			卓洛	男生
海贼王	路飞		山治	男生