通用平台开发指南

目录

1 简介

平台采用 mvc 开发方式

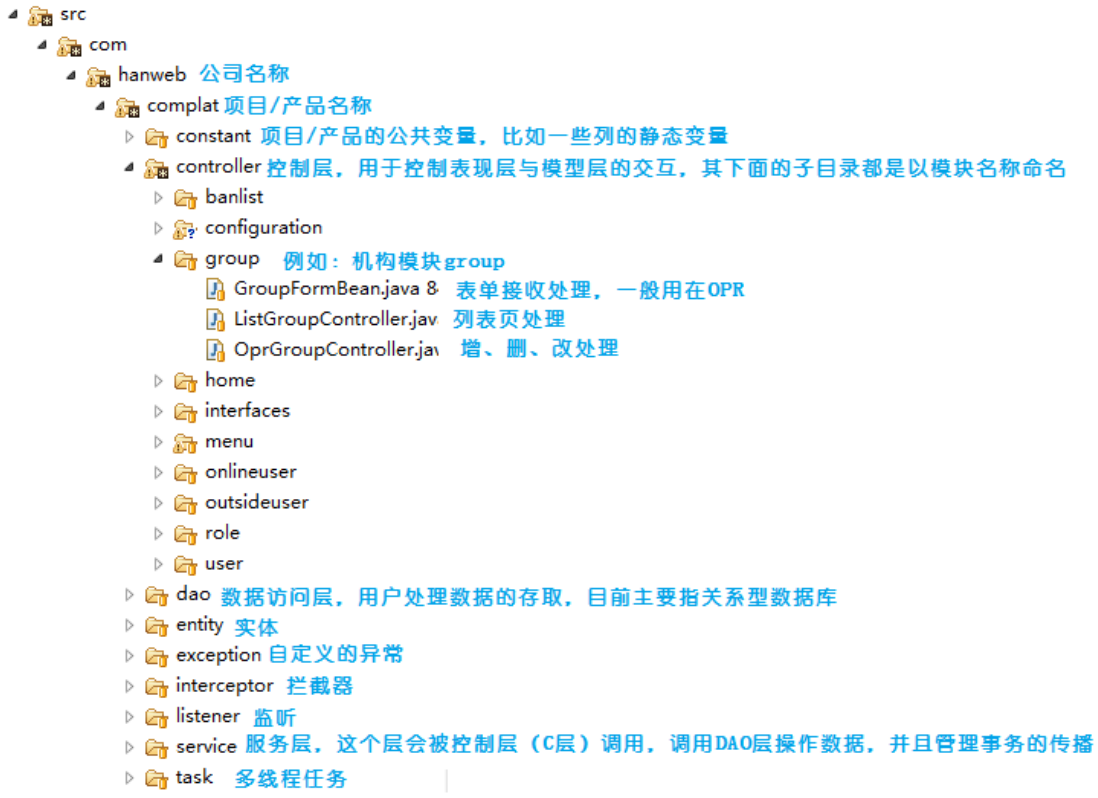
m：模型层，包含 service、dao、entity

v：表现层，泛指 jsp，也可以是 html 或者别的表现形式

c：控制层，用于控制 v 与 m 的交互

2 目录结构（规范）

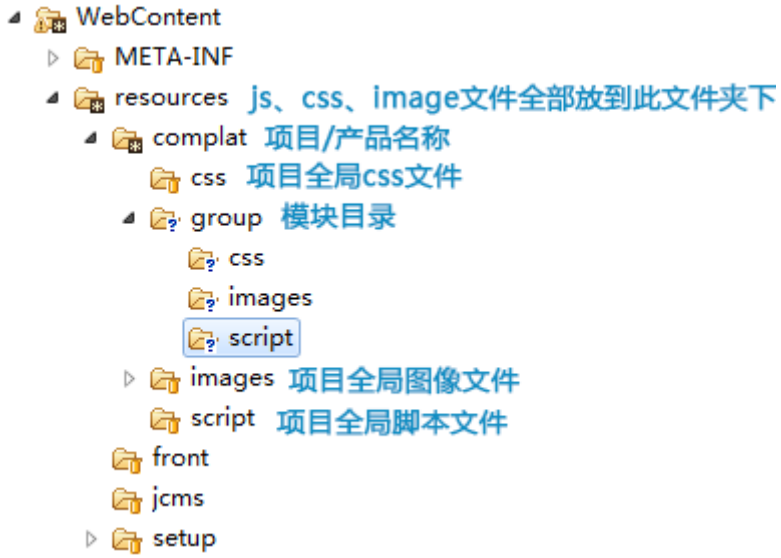
2.1 JAVA 目录



2.2 JSP 目录


WebContent
META-INF
resources
setup
ui
WEB-INF 目前jsp文件全部放在WEB-INF下
config
lib
pages 表现层容器
complat 项目/产品名称，这个下面存放所有当前项目或者产品的表现层文件，例如jsp
banlist
configuration
group 例如：机构管理
group_import.jsp 85125 1: 机构导入
group_list.jsp 85052 13-6-: 机构列表
group_menu.jsp 84069 13-
group_opr.jsp 85067 13-6- 机构操作
group.xls 84838 13-6-4 下
include
onlineuser
outsideuser
role
user
Copy of index.jsp 80542 13-5
index.jsp 84387 13-6-3 上午1C
menu.jsp 80542 13-5-17 上午1
develop
front
setup
support

2.3 JS、CSS、IMAGES 目录



2.4 SPRING 配置文件目录

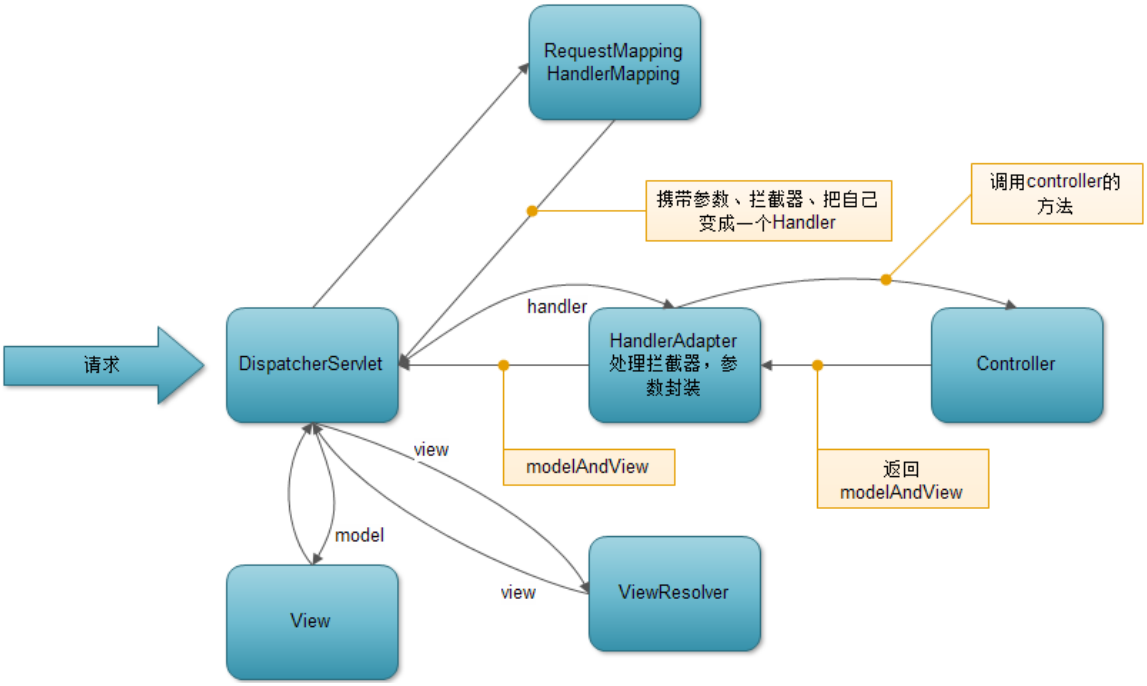
2.5 SQL 文件目录

2.6 国际化配置

3 编码

3.1 SpringMVC

3.1.1 请求处理流程图



3.1.2 Controller 的编写

Controller 其实就是一个普通的 bean，加上@Controller 就可以了，下面我们来看段代码：

```java
@Controller 声明控制器
@RequestMapping("manager/demo/form") 请求地址
public class FormController {
    此方法的访问地址为：manager/demo/form/form_show.do
    @RequestMapping("form_show")请求地址
    public ModelAndView showForm() {返回到页面的都为这个
        ModelAndView modelAndView = new ModelAndView("/demo/form/page");需要转到的JSP地址
        return modelAndView;
    }
    此方法的访问地址为：manager/demo/form/form_submit.do
    @RequestMapping("form_submit")
    @ResponseBody 直接返回到body，也就是直接返回字符串而不是到 jsp，通常用于ajax
    public String submitForm(@RequestParam("name") String name, 表示request的参数name的值传入方法的name
            @RequestParam("password") String password) {
        StringBuffer sb = new StringBuffer();
        sb.append("name:").append(name).append("\tpassword:").append(password);
        return sb.toString();
    }
}
```

3.2 Control 控制层编写

3.2.1 LIST 列表

3.2.1.1 概述

编写列表 Controller 时我们一般将列表分成几个部分，看下图：

```java
@Controller
@RequestMapping("manager/user")
public class ListUserController implements GridViewDelegate {1

    @Autowired
    private GridViewService gridViewService; 2

    @RequestMapping("list")
    public GridView list(HttpServletRequest request, GridView gridView, String groupId,  3
            String name, String loginName) {
        gridView.setDelegate(this);  4
        gridView.setViewName("complat/user/user_list");  5
        createButton(gridView);  6
        createHead(gridView);  7
        Integer userGroupId;
        if (NumberUtil.getInt(groupId) == 0) {
            User currentUser = UserSessionInfo.getCurrentUserInfo(request);
            userGroupId = currentUser.getRangeId();
        } else {
            userGroupId = NumberUtil.getInt(groupId);
        }
        createBody(gridView, userGroupId, name, loginName);  8
        gridView.addQueryParam("groupId", userGroupId + "");  9
        gridView.setPosition("用户管理");  10
        gridView.setSearchPlaceholder("请输入用户登录名");  11

        gridView.addObject("groupId", userGroupId);
        gridView.addObject("name", name);
        gridView.addObject("loginName", loginName);
        return gridView;
    }
```

1、列表页的 Controller 一定要实现一个 GridViewDelegate 接口，并且重写 createRow 方法，createRow 方法是用来处理数据库中每行数据在列表页的每行上的显示。

2、注入一个 GridVIewService，这个 service 是用于处理列表页的业务也就是执行 sql 语句

3、在列表页主体方法中一定要有一个 GridView 类型参数

4、将 GridVIew 参数的 delegate 设置为 this，实际上这里需要 set 一个实现了 GridViewDelegate 的类

5、设置列表页的展现页面

PS：前面 5 步是准备工作，也是必要条件，下面开始创建列表页展现，图中的方法参数根据业务需要定制，不要按照图例照搬，方法名可以照搬

6、创建列表页的头部功能按钮

7、创建列表的头部名称

8、创建列表数据

9、设置需要保持的参数，例如从树的节点打开了的列表页带了一个机构 id，而这个机构 id 需要在翻页、检索、等等地方用到，这是就需要保持

10、设置当前位置

11、设置普通检索输入框中的提示

3.2.1.2　列表的头部功能按钮


```
/**
 * 创建按钮
 *
 * @param gridView
 */
private void createButton(GridView gridView) {
    gridView.addButton(Button.ADD);
    gridView.addButton(Button.REMOVE);
    gridView.addButton(Button.IMPORT);
    gridView.addButton(Button.EXPORT);
}
```

GridView 的 addButton 方法用来添加头部功能按钮

Button 类可以通过 getInstance 静态方法来创建。当然也已经提供了多个默认按钮，有：新增（ADD）、修改（EDIT）、保存（SAVE）、删除（REMOVE）等等

3.2.1.3　列表的头部名称


```
/**
 * 创建表头
 *
 * @param gridView
 */
private void createHead(GridView gridView) {
    gridView.addHead(Head.getInstance().setCheckbox(true).setField("iid"));
    gridView.addHead(Head.getInstance().setField("name").setTitle("姓名").setAlign("left")
            .setWidth(200).setResizable(true));
    gridView.addHead(Head.getInstance().setField("loginname").setTitle("登录名")
            .setAlign("center").setWidth(100));
    gridView.addHead(Head.getInstance().setField("groupid").setTitle("机构id").setHidden(true));
    gridView.addHead(Head.getInstance().setField("groupname").setTitle("所属机构")
            .setAlign("center").setWidth(80));
    gridView.addHead(Head.getInstance().setField("headship").setTitle("职务").setAlign("center")
            .setWidth(80));
    gridView.addHead(Head.getInstance().setField("enable").setTitle("账号开启").setAlign("center")
            .setWidth(80));
}
```

GridView 的 addHead 方法用来添加列表的头部名称

Head 类可以通过 getInstance 静态方法来创建，创建之后可以调用里面的 set 方法来定制 head 的表现形式，并且可以连续 set，这些方法都通俗易懂就不在那里说明意思了。可以去看 api。

field 为列的唯一名称，在页面上 js 操作列的时候也是使用这个列名称。

title 为显示的列名称。

3.2.1.4 创建列表页具体的数据


```
/**
 * 创建列表
 *
 * @param gridView
 */
private void createBody(GridView gridView, Integer groupId, String name, String loginName) {
   1 GridViewSql gridViewSql = GridViewSql.getInstance(gridView);
    gridViewSql
           2 .addSelectField("iid")
             .addSelectField("name")
             .addSelectField("loginname")
             .addSelectField(
                     "(SELECT name FROM " + Tables.GROUP + " b WHERE b.iid = groupid) groupname")
             .addSelectField("headship").addSelectField("enable").setTable(Tables.USER);
    String where = ""; //仅过滤系统管理员
    if (NumberUtil.getInt(groupId) > 0) {
        where = "groupid = :groupId";
        gridViewSql.addParam("groupId", groupId);
    } else {
         where = "loginname <> 'admin'";
    }
    String userName = gridView.getSearchText();
    if(StringUtil.isNotEmpty(userName)){
        where += " AND (name LIKE :userName OR loginname LIKE :userName)";
        gridViewSql.addParam("userName", "%" + userName + "%");
    } else {
        if (StringUtil.isNotEmpty(name)) {
          4 where += " AND name LIKE :name";
          5 gridViewSql.addParam("name", "%" + name + "%");
        }
        if(StringUtil.isNotEmpty(loginName)){
            where += " AND loginname LIKE :loginName";
            gridViewSql.addParam("loginName", "%" + loginName + "%");
        }
    }

   6 gridViewSql.setWhere(where);
   7 gridViewService.find(gridViewSql);
}
```

创建列表大部分情况是需要和数据库交互，我们看看怎么使用 GridViewService 来取得数据库数据。

1、创建一个 GridViewSql 对象。

2、添加字段，可以是普通字段也可以是子查询，支持连续添加

3、设定需要查询的表，这里不限定一张表，可以是多表加别名，也就是 sql 的 from 怎么写这里就怎么写

4、组织 where 语句，where 语句使用 preparedstatement 语法方式，例如 WHERE userid=:id AND username=:name

5、为 where 语句中的 preparedstatement 填充值，使用这种方式可以不考虑 sql 注入，如果直接将值拼到 where 语句中，那就有 sql 注入的风险。

6、将 where 语句放到 GridViewSql 中，排序的语句也可以写在 where 中

7、使用 GridViewService 的 find 方法来获取 GridViewSql 中调用的数据

3.2.1.5 设置需要保持的参数

```
@RequestMapping("list")
public GridView list(HttpServletRequest request, GridView gridView, String groupId,
        String name, String loginName) {
    gridView.setDelegate(this);
    gridView.setViewName("complat/user/user_list");
    createButton(gridView);
    createHead(gridView);
    Integer userGroupId;
    if (NumberUtil.getInt(groupId) == 0) {
        User currentUser = UserSessionInfo.getCurrentUserInfo(request);
        userGroupId = currentUser.getRangeId();
    } else {
        userGroupId = NumberUtil.getInt(groupId);
    }
    createBody(gridView, userGroupId, name, loginName);
    gridView.addQueryParam("groupId", userGroupId + "");  带到Gird控件和页面
    gridView.setPosition("用户管理");
    gridView.setSearchPlaceholder("请输入用户登录名");

    gridView.addObject("groupId", userGroupId);  带到页面
    gridView.addObject("name", name);
    gridView.addObject("loginName", loginName);
    return gridView;
}
```

GridView 可以携带需要保持的参数到 Grid 控件，在切换 pagesize、翻页、检索都会带着这个参数去 controller；高级检索的参数不需要这样带，高级检索的值需要通过 addObject 方法带到页面相应的表单控件当中

其中带到 grid 控件的值在页面上可以用 jstl 表达式获取，同 addObject

3.2.1.6 createRow 方法

```
@Override                                    1                    2
public void createRow(GridRow gridRow, Map<String, Object> rowData) {
    String iid = StringUtil.getString(rowData.get("iid"));3
    String name = StringUtil.getString(rowData.get("name"));
    String loginName = StringUtil.getString(rowData.get("loginname"));
    String groupId = StringUtil.getString(rowData.get("groupid"));
    String groupName = StringUtil.getString(rowData.get("groupname"));
    String headship = StringUtil.getString(rowData.get("headship"));
    String enable = StringUtil.getString(rowData.get("enable"));

  4 gridRow.addCell("iid", iid);
    gridRow.addCell("name", name, gridRow.createOnclick("edit", iid, name));5
    gridRow.addCell("loginname", loginName);
    gridRow.addCell("groupid", groupId);
    gridRow.addCell("groupname", groupName);
    gridRow.addCell("headship", headship);
    gridRow.addCell("enable", "1".equals(enable) ? "开启" : "关闭",
            gridRow.createOnclick("modifyEnable", iid, enable));
}
```

createRow 方法用来创建列表页的每一行数据，该方法有两个参数，一个是 GridRow，第二个是 Map，这两个参数都是实例化好了第二个参数 Map 中是有值的也就是数据库的行数据。

1、GridRow，Grid 的行控件行对象

2、rowData，数据库的行数据，其中 key 为 GridViewSql 的 selectField 的值，value 为对应的数据库的值

3、根据 selectField 的值取出对应的数据库的值

4、给 GridRow 增加列的值，第一个参数为列名称，第二个参数为值

5、可以通过 GridRow 来创建一个 script 的调用，第一个参数为方法名称，后面可以有任意个参数。例如页面有个 js 方法为 function edit(username,userid) 这里就可以写成 gridRow.createOnclick("edit ",username,userid)

通过以上步骤一个列表页的 java 业务基本就完成了

3.2.2 OPR 操作

3.2.2.1 概述

OPR 操作的 Controller 概括起来看一般只有 2 种操作

1、打开页面：通过 ModelAndView 转到一个 jsp

2、接受提交：接受 post 或者 get 等方式的数据提交

所以我们的 requestmapping 的命名方式就为 xxx_show 打开页面、xxxx_submit 提交页面，而相应的方法名称就为 showXxxx、submitXxxx。

3.2.2.2 新增

```
@RequestMapping(value = "add_show") 1
public ModelAndView showAdd(HttpServletRequest request, String pid) {
    ModelAndView modelAndView = new ModelAndView("complat/group/group_opr");
    Group group = new Group();      2
    Group tempGroup = groupService.findByIid(NumberUtil.getInt(pid));
    if (tempGroup != null) {
        group.setPid(tempGroup.getIid());
        group.setPname(tempGroup.getName());
    }

    modelAndView.addObject("url", "add_submit.do"); 3
    modelAndView.addObject("group", group);
    this.addOtherObject(request, modelAndView);
    return modelAndView;4
}
```

1、首先按照规范我们的 requestmapping 的值为 add_show，表示打开新增页面

2、创建一个 ModelAndView 并且指向相应的 jsp

3、通过 ModelAndView 将表单提交的地址带到页面上（也就是说，当打开新增页面的时候，同时将下一步操作的 url 带入新增页面，一般都是放入新增页面的 form 的 action 中）

4、返回 ModelAndView

3.2.2.3 保存

```
1 @RequestMapping(value = "add_submit")
2 @ResponseBody
3 public String submitAdd(GroupFormBean group)4{
      boolean isSuccess = false;
      String message = "";
      try {
          isSuccess = groupService.add(group);
      } catch (OperationException e) {
          message = e.getMessage();
      }
      Script script = Script.getInstance(); 5
      if (isSuccess) {
          script.refreshNode(NumberUtil.getInt(group.getPid()) + "").closeDialogAndReload(); 6
      } else {
          script.addAlert("新增失败！" + message);
      }
      return script.getScript(); 7
  }
```

1、首先按照规范我们的 requestmapping 的值为 add_submit，表示进行新增的保存操作。

2、由于我们的提交之后并不需要转到页面而是需要输出 js 代码来执行，比如：关闭弹窗、刷新列表页、操作树节点等，所以这时候我们用 responsebody

3、返回 js 代码就是一串字符串，所以这里返回 String

4、这个是一个普通的 javaBean，只是对提交来的参数进行了封装，当然如果参数少（少于 6 个）的话就不用建一个 bean 了，这个可以参考之前学习的 springmvc 内容

5、做完业务之后我们需要输出相应的 js 了，这时候我们创建一个 Script。

6、Script 类支持连续的 set，里面有封装好的 js 操作，关闭窗口，刷新父页面、alert 提示、操作树、如果提供的 js 操纵不满足需要可以使用它的 addScript 方法添加需要的 js 操作代码。

7、通过 getScript 方法返回 js 代码

3.2.2.4 编辑

```
@RequestMapping(value = "modify_show") 1
public ModelAndView showModify(HttpServletRequest request, String iid) {
    ModelAndView modelAndView = new ModelAndView("complat/group/group_opr"); 2

    Group group = groupService.findByIid(Integer.parseInt(iid));
    modelAndView.addObject("url", "modify_submit.do"); 3
    modelAndView.addObject("group", group);
    this.addOtherObject(request, modelAndView);
    return modelAndView; 4
}
```

1、首先按照规范我们的 requestmapping 的值为 modify_show，表示打开新增页面

2、创建一个 ModelAndView 并且指向相应的 jsp

3、通过 ModelAndView 将表单提交的地址带到页面上（也就是说，当打开编辑页面的时候，同时将下一步操作的 url 带入编辑页面，一般都是放入编辑页面的 form 的 action 中）

4、返回 ModelAndView

3.2.2.5 修改

```
1 @RequestMapping(value = "modify_submit")
2 @ResponseBody
3 public String submitModify(GroupFormBean group)4{
      boolean isSuccess = false;
      String message = "";
      try {
          isSuccess = groupService.modify(group);
      } catch (OperationException e) {
          message = e.getMessage();
      }
      Script script = Script.getInstance(); 5
      if (isSuccess) {
          if (group.getPrevPid() != group.getPid()) {
              script.removeNodes(NumberUtil.getInt(group.getIid()) + "");
          }
        6 script.refreshNode(NumberUtil.getInt(group.getPid()) + "").closeDialogAndReload();
      } else {
          script.addAlert("更新失败！" + message);
      }

      return script.getScript(); 7
  }
```

1、首先按照规范我们的 requestmapping 的值为 modify_submit，表示进行编辑的保存操作。

2、由于我们的提交之后并不需要转到页面而是需要输出 js 代码来执行，比如：关闭弹窗、刷新列表页、操作树节点等，所以这时候我们用 responsebody

3、返回 js 代码就是一串字符串，所以这里返回 String

4、这个是一个普通的 javaBean，只是对提交来的参数进行了封装，当然如果参数少（少于 6 个）的话就不用建一个 bean 了，这个可以参考之前学习的 springmvc 内容

5、做完业务之后我们需要输出相应的 js 了，这时候我们创建一个 Script。

6、Script 类支持连续的 set，里面有封装好的 js 操作，关闭窗口，刷新父页面、alert 提示、操作树、如果提供的 js 操纵不满足需要可以使用它的 addScript 方法添加需要的 js 操作代码。

7、通过 getScript 方法返回 js 代码

3.2.2.6 删除

```
@RequestMapping(value = "remove") 1
@ResponseBody 2
public String remove(String ids, String pid) {
    Script script = Script.getInstance(); 3
    boolean isSuccess = false;
    String message = "";
    try {
        isSuccess = groupService.removeByIds(ids);
    } catch (OperationException e) {
        message = e.getMessage();
    }
    String[] idArray = StringUtil.split(ids, ",");
    if (isSuccess) {
        script.removeNodes(idArray).reload();4
    } else {
        script.addAlert("删除失败！" + message);
    }
    return script.getScript(); 5
}
```

1、删除的 requestMapping 为 remove

2、删除通常是列表页上的操作，处理完之后需要刷新页面，也就是输出 js 所以用 responsebody

3、做完业务之后我们需要输出相应的 js 了，这时候我们创建一个 Script。

4、定制需要的 js

5、输出 js

3.2.2.7 单文件上传

单文件上传指的是使用 html 的<input type="file"/>来上传

```
@RequestMapping(value = "import_submit")
@ResponseBody
public String submitImport(MultipartFile file) {1
    Script script = Script.getInstanceWithJsLib();
    String message = "";
    if (file.isEmpty()) {
        message = SpringUtil.getMessage("import.nofile");
    } else {
        File filePath = new File("/tempfile");2
        try {
            ControllerUtil.writeMultipartFileToFile(filePath, file);3
            message = groupService.importGroup(filePath);
        } catch (OperationException e) {
            message = e.getMessage();
        }
    }
    script.addScript("parent.showMessage('" + message + "')");

    return script.getScript();
}
```

1、接受文件上传的方法参数必须要有一个 MultipartFile 对象，此对象对应上传的文件，对象的引用名称对应 html 的 file 控件的 name

2、创建一个 File，这个 File 是用来保存文件的，MultipartFile 对象里面保存的是上传文件的数据流，我们需要将流写入 file 文件中。

3、使用 ControllerUitl 中方法将流写入文件，写好之后就可以操作目标文件了。

单文件上传就这样完成了

3.2.2.8 多文件上传

1、头部需要使用 h:head 来加载必要的 css 和 js

```
<h:head fileupload="true"></h:head>
```

2、增加多文件选择按钮和文件列表

```
<a type="button" class="btn btn-small btn-success upload_btn">
    <i class="icon-cloud-upload"></i>添加文件
</a>
<ul class="upload_filelist">
```

3、使用多文件上传插件

```
$(function() {
    $('.upload_btn').multifileupload( {
      a dialogUrl : '${uploadUrl}',
      b filters : [ {
          c title : '图片文件（jpg,gif,png）',
          d extensions : 'jpg,gif,png'
        }, {
            title : '压缩包（zip）',
            extensions : 'zip'
        } ]
    });
});
```

a、指定一个打开上传对话框的地址，这个地址通过 controller 传来，并且是绝对地址：/应用名称/manager/component/multifileupload.do，打开的对话框如下

b、filters 为允许上传的文件扩展名设置

c、title 为选择文件对话框的提示

d、extensions 为扩展名

4、点击“添加文件”按钮，可选择做多 10 个文件。点击“开始上传”

5、上传成功，窗口自动关闭，并将上传成功的文件返回表单页面，可点击图标进行删除

6、在表单提交的时候，增加脚本获取上传文件的 json 字符串，键为 uuid，值为原文件名

```
var files = $('.upload_btn').multifileupload('getFiles');
var filesJson = JSON.stringify(files);
```

7、在表单数据的处理时，可以解析上传文件的 JSON，通过查询表 complat_tempfile，找到临时目录及文件名，从临时目录，将临时文件复制到最终存储目录下。临时目录默认为应用下的 tempfile

3.2.2.9 下载

下载有两种方式，第一种是直接连接到文件下载；第二种是读取文件流下载，这里我们说的是读取文件流下载，第一种方式只要做跳转就可以了。

```
@RequestMapping(value = "downloadfile")
@ResponseBody 1
public FileResource 2downloadFile() {
    File file = new File(BaseInfo.getRealPath()
            + "/WEB-INF/pages/complat/group/group.xls");
    FileResource fileResource = ControllerUtil.getFileResource(file,
            "group.xls"); 3

    return fileResource;
}
```

1、用流的方式下载文件返回的一定不是页面所以这里 responsebody

2、流的方式下载要求返回一个 FileResource 对象

3、使用 ControllerUtil 的 getFileResource 方法创建一个 FileResource 对象，第一个参数为要下载的文件对象，第二个参数为下载显示的名称

3.3 Service 业务层编写

3.3.1 概述

1、接受 controller 传来的参数，做一些必要验证

2、调用 Dao 层来获取数据，可以在一个方法里面调用不同的 Dao 完成不同表的数据获取，也可以调用 Service

3、事务贯穿 Service 的方法被调用开始到结束，过程中不管调用 dao 还是 service 都具有事务

4、捕获异常将友好的错误信息返回给 controller，数据清理使本层返回的数据是 controller 可以直接使用的

5、service 需要配置在 spring 的 xml 中

3.3.2 方法名与事务

service 层的事务控制是根据方法名称来的，使用不同的方法名称前缀可以获得不同的事务方式。

方法名前缀（可以直接使用）	事务特性
add	可写

remove	可写
modify	可写
import	可写
check	可写
find	只读
is	只读
export	只读
get	只读

1、如果方法名称没有使用上述任意一种前缀，此方法的生命周期将不会伴随事务，但是事务是可以传播的，如果是从有事务的方法来调用无事务的方法，那无事务的方法也会具有事务。

2、可写事务可以进行查询和修改表数据操作；只读事务不可以对数据库的数据进行修改，否则会报错。

3.3.3 示例

```
public class OutsideUserService {

    @Autowired
1   private OutsideUserDao outsideUserDao;

    /**
     * 新增外网用户
     *
     * @param outsideUser
     *            外网用户实体
     * @return true - 成功<br/>
     *         false - 失败
     * @throws OperationException
     *             界面异常
     */              2
    public boolean add(OutsideUserFormBean outsideUser) throws OperationException {
        if (outsideUser == null) {
            return false;
        }

      3 int num = outsideUserDao.findIidByLoginName(outsideUser.getLoginName());
        if (num > 0) {
            throw new OperationException("存在相同登录名的用户！");4
        }

        outsideUser.setPwd(Md5Util.md5encode(outsideUser.getPwd()));
        outsideUser.setRegtime(new Date());
        outsideUser.setPinYin(PinyinUtil.hanziToPinyin(outsideUser.getName()));
        int iid = outsideUserDao.insert(outsideUser); 5

        return iid > 0 ? true : false;
    }
```

1、注入对应的 Dao

2、定义方法名，例子为新增操作方法名为 add

3、第一次调用 dao 的方法

4、由于我们方法定义的返回值为 boolean，只是知道新增成功还是失败，别的信息不知道，这时我们需要定义一个异常，在出现不满足的情况下，带着异常信息一起抛出，由 controller 层捕获异常并获取异常信息再考虑是否要将异常信息反映到页面上。

5、新增保存

3.3.4 spring 配置

controller 调用 service 时，我们一般用自动注入方式 autowired，使用自动注入方式的先决条件是 spring 知道有这个 service，所以我们需要将这个 service 配置到 spring 的 xml 中。

示例中的配置文件路径为 WebContent/WEB-INF/springxml/complat/service.xml

而我们做具体产品或项目使用配置文件为自己项目下路径为

WebContent/WEB-INF/springxml/project/service.xml

只需要将 service 配置成如下方式就可以了,并且每个 service 都要写在里面

```
<bean class="com.hanweb.complat.service.GroupService"></bean>
<bean class="com.hanweb.complat.service.UserService"></bean>
<bean id="ldapService" class="com.hanweb.complat.service.LdapService"></bean>
<bean class="com.hanweb.complat.service.TempFileService"></bean>
<bean class="com.hanweb.complat.service.RoleService"></bean>
<bean class="com.hanweb.complat.service.RoleRelationService"></bean>
<bean class="com.hanweb.complat.service.RightService"></bean>
<bean class="com.hanweb.complat.service.GroupManagerService"></bean>
<bean class="com.hanweb.complat.service.OutsideUserService"></bean>
<bean class="com.hanweb.complat.service.BanListService"></bean>
```

3.4 Dao 数据访问层编写

3.4.1 概述

dao 是负责与持久化的数据操作，可以是数据库、索引库、文件、其他网络资源。

在平台中提供了 baseJdbcDao 用于操作数据库，所有操作数据库的 dao 必须继承与它，否则不能操作数据库。

dao 需要配置在 spring 的 xml 中

3.4.2 方法

这里提到的方法规则是一种规范与习惯和 service 的方法规则没有联系

方法名前缀（可以直接使用）	意义
insert	新增
delete	删除
update	更新
find	查找

3.4.3 示例

```
public class RoleDAO extends BaseJdbcDAO<Integer, Role> { 1

    /**
     * 通用角色id获得角色信息
     *
     * @param iid
     *            角色id
     * @return
     */
    public Role findByIid(int iid) {
2       String sql = "SELECT iid, name, isdefault, spec, type FROM " + Tables.ROLE
                + " WHERE iid=:iid";
3       Query query = createQuery(sql);
4       query.addParameter("iid", iid);
5       Role role = this.queryForEntity(query);
        return role;
    }
```

1、首先需要将创建的 dao 继承 BaseJdbcDao，要求传入 2 个类型，第一个类型为对应实体的主键类型，第二个类型为实体类型，这样之后就可以使用 BaseJdbcDao 的实体操作方法直接让 sql 与实体进行交互。

2、组织 sql 语句，此语句是 preparedstatement 形式

3、创建一个 query

4、填充 preparedstatement 的参数

5、调用实体方法获得实体

3.4.4 spring 配置

service 调用 dao 时，我们一般用自动注入方式 autowired，使用自动注入方式的先决条件是 spring 知道有这个 dao，所以我们需要将这个 dao 配置到 spring 的 xml 中。

示例中的配置文件路径为 WebContent/WEB-INF/springxml/complat/dao.xml

而我们做具体产品或项目使用配置文件为自己项目下路径为

WebContent/WEB-INF/springxml/project/dao.xml

```
<bean class="com.hanweb.complat.dao.GroupDao"></bean>
<bean class="com.hanweb.complat.dao.UserDao"></bean>
<bean id="ldapDao" class="com.hanweb.complat.dao.LdapDao"></bean>
<bean class="com.hanweb.complat.dao.TempFileDao"></bean>
```

3.4.5 实体操作方法说明

要使用 BaseJdbcDao 的实体操作，首先实体必须配置了 annotation（详见 Entity 编写），一定有主键，并且相应的 Dao 在继承 BaseJdbcDao 时给定了主键类型和实体类型，不支持对象之间的级联，下面以 GroupDao 为例

3.4.5.1 新增

GroupDao.insert(group)，返回新增后的主键

3.4.5.2 删除

GroupDao.deleteById(id)

GroupDao .deleteByIds(ids)

3.4.5.3 修改

主键必须有值

GroupDao.update(group)

3.4.5.4 查找

GroupDao. queryForEntityById(id)：根据主键值查找一个实体

GroupDao. queryForEntity(query)：根据 sql 查找一个实体

GroupDao. queryForEntities(query)：根据 sql 查找多个实体，返回 list

3.4.6 普通操作方法说明

普通操作方式没有新增方法，并且普通操作方式取出的数据是 Map 类型，多行数据是 List<Map>类型。修改需要用到 UpdateSql 类（参见 3.4.7）下面以 GroupDao 为例

3.4.6.1 删除

GroupDao.delete(query)

3.4.6.2 修改

GroupDao. update(UpdateSql)

3.4.6.3 查找

GroupDao. query(Query)

更多方法参见 api

3.4.7 UpdateSql

UpdateSql 是一个辅助做 update 操作的工具，用于帮助开发者组织 update 语句。

下面的例子需要用的 update 语句为：

```
UPDATE user SET accesstime=xxxxx,ip=xxxxx WHERE iid=:iid
```

```
/**
 * 登录成功后更新登录时间
 *
 * @param iid
 *            用户ID
 * @param ip
 *            用户IP地址
 * @param accesstime
 *            登录时间
 * @return true - 成功<br/>
 *         false - 失败
 */
public boolean updateLoginInfo(int iid, String ip, Date accesstime) {
  1 UpdateSql sql = new UpdateSql(Tables.USER);
  2 sql.addDate("accesstime", accesstime);
  3 sql.addString("ip", ip);
  4 sql.setWhere("iid = :iid");
  5 sql.addWhereParamInt("iid", iid);

  6 return super.update(sql);
}
```

1、首先 new 一个 UpdateSql，构造方法要求传入一个表名

2、根据字段的类型设置需要 update 的字段

3、同 2

4、设置 where 语句，注意：where 语句使用 preparedstatement 语法方式

5、根据 where 的字段类型设置相应的字段值

6、调用 dao 的 update 方法完成修改操作

Update 可以 add 什么类型的字段参看 api。

3.5 Entity 编写

3.5.1 概述

平台支持 entity 使用 SQL 注释，使用了 SQL 注释可以通过 dao 直接保存、修改、删除以及获取实体而不需要写 sql 语句。

目前注释有 Table、Id、Colum、OnlyQuery 这 4 种，同时系统会根据带有注释的实体来自动建表。

3.5.2 示例

```
@Table(name = Tables.GROUP)
public class Group implements Serializable {

    /**
     * 主键
     */
    @Id
    @Column(type = ColumnType.INT)
    private Integer iid;

    /**
     * 机构名称
     */
    @Column(type = ColumnType.VARCHAR)
    private String name;

    /**
     * 机构描述
     */
    @Column(type = ColumnType.VARCHAR)
    private String spec;

    /**
     * 父id
     */
    @Column(type = ColumnType.INT)
    private Integer pid;
```

3.5.3 Table

```
@Table(name = Tables.GROUP)
public class Group implements Serializable
```

用于描述表与实体的对应关系

Table 必须加在实体上，name 为表名称

3.5.4 Id

```
@Id
@Column(type = ColumnType.INT)
private Integer iid;
```

用于描述主键字段，并且可以设定主键的生成策略，默认为自增长。一个实体只能有一个主键~!!

Id 必须加在含有 Column 描述的实体属性上，Id 可以设置 GeneratorType

GeneratorType 类型	意义
NATIVE	自增长
UUID	32 为 uuid

3.5.5 Column

```
@Column(type = ColumnType.VARCHAR)
private String name;
```

用于描述数据库字段与实体字段的对应关系，类型为必填项

Column 属性	意义
name	数据库字段名称，不写默认为所标注的实体属性名称的小写
type	数据库字段类型，使用 ColumnType 来指定
length	数据库字段长度
update	是否在使用 update 方法时更新

ColumnType 支持所有数据库，不用担心类型有没有，系统会自动判断

ColumnType 类型	数据库类型	默认长度
DATE	date 类型 yyyy-MM-dd	20
DATETIME	datetime 类型 yyyy-MM-dd hh:mm:ss	20
INT	int 类型	11
DOUBLE	double 类型	0
FLOAT	float 类型	20
VARCHAR	varchar 类型	255
CHAR	char 类型	50
TEXT	大字段类型	0

3.5.6 OnlyQuery

```
/**
 * 父机构名称
 */
@OnlyQuery
private String pname;
```

OnlyQuery 表示实体的字段只参与 select 查询，比如实体有个属性并不是要存数据库的，但这个值又是从数据库获取的（别的表的字段），这时我们还是希望 dao 可以返回实体，就可以给对应的属性加上这个注释，这个注释可以指定字段名称，如果不写默认为属性名称的小写。

3.5.7 实体普通属性处理

对于不写 annotation 的实体属性只参与 select 查询，只要 select 的字段与实体属性字段的小写形式对应，那 select 出来的值将会被 set 到实体的属性中。

或许大家会疑惑，对普通属性已经处理了还要 OnlyQuery 有什么用，首先 OnlyQuery 可以指定字段名称，其次指定了 OnlyQuery 映射的速度会比不写来的快。

3.6 JAVA 验证框架

3.6.1 概述

JAVA 验证框架是在 Bean 中利用对于一些属性加入注释信息，然后利用验证工具类去验证 bean 的合法性，不合法将会抛出异常，验证的注释可以放在任何 bean 上面不一定放在 entity 上。

3.6.2 示例

1、对需要验证的 bean 加上注释

```java
public class TestFormBean {

    @NotNull(message="姓名不能为空")
    @Length(min=1,max=6,message="姓名必须大于1个字小于6个字")
    private String name = null;

    @NotNull(message="年龄不能为空")
    @Range(min=12,max=50,message="老人与小孩不可以")
    private Integer age = null;

    @NotNull(message="体重不能为空")
    @Max(value=100,message="体重不能超过100")
    @Min(value=10,message="体重不能小于10")
    private Double weight = null;

    @NotNull(message="报名日期不能为空")
    @DateRange(min="2013-1-9",max="2013-3-20",message="报名日期在13年1月9日-13年3月20")
    private Date createDate = null;

    @NotNull(message="email不能为空")
    @Email(message="email格式不对")
    private String email = null;

    @NotNull(message="证书不能为空")
    @MinSize(value=1,message="证书至少需要1个")
    private List<String> list = null;
```

2、验证，并且获取验证失败的 message

```java
try {
    // 开始验证testFormBean
    ValidationUtil.validation(testFormBean);
} catch (ValidationException e) {
    // 获得验证失败的message
    String message = e.getMessage();
}
```

3.6.3 验证类型

注意：必须先非 null 验证之后再做别的验证

注释	意义	备注
@NotNull	不为 null	只针对 null
	字符串验证	
@NotBlank	不为空字符串	自动 trim
@Length	字符串长度范围	
@MaxLength	字符串最大长度	
@MinLength	字符串最小长度	
@NotEqual	字符串比较	
@HasSubstring	字符串是否包含另一个字符串	

@Email	邮件验证
	数字验证
@Range	数字范围
@Max	数字不超过
@Min	数字不小于
	日期验证
@DateRange	日期范围
	集合和数组验证
@Size	集合和数组范围
@MinSize	最小 size
@MaxSize	最大 size
	正则表达式验证
@MatchPattern	正则匹配
@NotMatchPattern	正则不匹配

3.7 树

组织一棵树的节点我们需要用到两个类，第一个 Tree，这个是树的主体，主要负责承载节点，第二个是 TreeNode，这个主要用来创建节点，然后放到 Tree 里面。

```
// 组织树
Tree tree = Tree.getInstance(); 1

tree.addNode(TreeNode.getInstance(rangeId + "", "", nodeName, "/manager/group/list.do"));
2            3
List<Group> groupList = groupService.findChildGroupByIid(NumberUtil.getInt(rangeId));

for (Group group : groupList) {
    tree.addNode(TreeNode.getInstance(group.getIid() + "", rangeId + "", group.getName(),
            "/manager/group/list.do", group.getIsParent(), false));
}

                                                    4
modelAndView.addObject("tree", tree.parse());
```

1、 创建一个树的实例
2、 使用 tree 的 AddNode 方法装载节点。
3、 使用 TreeNode 的 getInstance 来创建节点；创建多个节点需要多次调用 AddNode 方法。
4、 TreeNode 组织完毕后使用 tree 的 parse 方法来输出 tree 所需的 json 数据。

具体的方法参见 API

3.8　计划任务（线程池）

计划任务是由任务管理器统一调度和执行的，编写代码时我们首先创建一个任务，并且设置此任务的 id、name 以及执行的时间计划，并实现 doWork 方法，doWork 方法是此任务具体执行的内容，设置好之后将此任务放入任务管理器，到时间即可执行，通过任务管理器可以控制任务的暂停、重启、停止、删除。doWork 方法在执行的过程中任务管理对任务的操作会在下一轮生效。

3.8.1 创建任务

```
public class TempFileTask extends BaseTask
```

建立一个类并继承 BaseTask 类，并重写相应的方法。

重写的方法	意义	备注
setTaskId	设置任务的唯一 id	必须
setTaskName	设置任务的名称	必须
setTaskGroup	设置任务所在的组	可以为 null
setTaskSchedule	设置计划	使用 TaskScheduleBuilder 来定制计划
addParam	增加额外参数给 dowork 用	
doWork(JobDataMap dataMap)	任务具体的业务	dataMap 为 addParam 的值
config	设置任务属性	设置 setTaskId、setTaskName、setTaskGroup、setTaskSchedule、addParam

例子

```java
public class TempFileTask extends BaseTask {
    @Override
    protected void config() {
        setTaskId("clean_temp_file");
        setTaskName("临时文件清除");
        TaskScheduleBuilder taskScheduleBuilder = TaskScheduleBuilder.getInstance();
        taskScheduleBuilder.setHour("1");
        setTaskSchedule(taskScheduleBuilder.getSchedule());
    }

    @Override
    protected void doWork(JobDataMap dataMap) {
        logger.debug("清除暂存附件开始...");
        try {
            if (BaseInfo.isPrepared()) {
                SpringUtil.getBean("complat_TempFileService", TempFileService.class)
                        .removeYesterday();
            }
        } catch (Exception e) {
            logger.warn("清除暂存附件异常", e);
        }
    }
}
```

3.8.2 定制计划

TaskScheduleBuilder 用来定制执行计划

方法	意义	静态方法	备注
getInstance	获得一个 TaskScheduleBuilder 实例	是	获得一个任务周期，不可以直接使用，需要设置时间：秒、分钟、小时、天、月、星期、年等最后调用 getSchedule 来获取相应的表达式来使用
setSecond	设置分钟	否	
setHour	设置小时	否	
setDay	设置日期	否	
setMonth	设置月份	否	
setWeek	设置星期	否	

setYear	设置年	否	
getSchedule	获得任务周期表达式	否	
getEveryHourSchedule	获取按小时的任务周期	是	可以直接使用
getEveryMinuteSchedule	获取按分钟的任务周期	是	可以直接使用
getEverySecondSchedule	获取按秒的任务周期	是	可以直接使用

例子

使用 getInstance 创建的任务计划（每天 0 点执行）

```
@Override
protected Object setTaskSchedule() {
    TaskScheduleBuilder taskScheduleBuilder = TaskScheduleBuilder.getInstance();
    taskScheduleBuilder.setHour("0");
    return taskScheduleBuilder.getSchedule();
}
```

使用 getEveryMinuteSchedule 创建的任务计划（每 5 分钟执行一次）

```
@Override
protected Object setTaskSchedule() {
    return TaskScheduleBuilder.getEveryMinuteSchedule(5);
}
```

3.8.3 任务管理

使用 TaskManager 来管理任务，TaskManager 提供了如下方法

方法	意义
stopAllTask	停止所有任务
addTask	将任务加入管理
pauseTask	暂停任务
resumeTask	重启暂停的任务
removeTask	删除任务

例子

```
TaskManager.addTask(new TempFileTask());
```

只有将 task 交给 TaskManager 来管理，task 才会启动

3.9　拦截器

3.9.1 概览

拦截器是根据 url 地址匹配来起作用，拦截器可以在：接到请求、执行 controller 方法完成、请求完成这 3 个点来执行相关的程序。

3.9.2 创建拦截器

```
public class OperationLogInterceptor extends BaseInterceptor
```

建立一个类并继承 BaseInterceptor 类，并重写相应的方法。

重写的方法	意义	备注
before	接到请求、还未执行 controller 方法	返回 true 继续执行、返回 false 中断请求
after	Controller 方法执行完之后	
complete	请求完成之后	

3.9.3 配置拦截器

拦截器需要配置到 spring 的 xml 中
平台的配置到了 WebContent/WEB-INF/springMVC-servlet.xml。

其他产品或者项目需要配置到
WebContent/WEB-INF/springxml/springMVC-servlet-project.xml。

配置方式为如下

```
<mvc:interceptors> 1
    <mvc:interceptor> 2
        <mvc:mapping path="/manager/**" /> 3
        <bean class="com.hanweb.common.interceptor.OperationLogInterceptor" /> 4
    </mvc:interceptor>
</mvc:interceptors>
```

1、 拦截器容器，在这个拦截器容器下可以配置多个拦截器
2、 配置一个拦截器
3、 配置拦截器作用的 url
4、 配置拦截器的 bean

3.10 h 标签

3.10.1 概述

h 标签用在 jsp 上，可以加入页面组建、可以导入外部 js、css 文件、可以创建 grid、可以创建提示控件、可以创建验证码控件。

3.10.2 h 标签的声明

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@ taglib prefix="h" uri="/WEB-INF/tag/hanweb-tags.tld"%>
```

<%@ taglib prefix="h" uri="/WEB-INF/tag/hanweb-tags.tld"%>

3.10.3 h:head

```
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>通用开发平台3.0</title>
<h:head validity="true" message="true" placeholder="true"></h:head>
```

h:head 使用在 html 的 head 标签中，设置相应的页面组件为 true 可以自动导入相应必须的 css 和 js。

head 的页面组件有以下几种

标签	意义	备注
calendar	日历	
cookie	Cookie 读写	
dialog	对话框	
fonticon	字体图标	
grid	列表	列表页必须
hmenu	下拉菜单	
hmultiselect	多选菜单	
message	消息	
pagetype	页面类型	值为page或者dialog，并且每个页面都需要设置

tabs	选项卡	
tree	树	有树的页面都需要
upload	多文件上传	
validity	Js 验证框架	
tip	提示	
combox	下拉选单	
loadmask	遮罩层	
fileupload	多文件上传	
json	json 解析	

3.10.4 h:import

```
<head>
<meta http-equiv="Content-Type" content="text/html; charset=
<title>Insert title here</title>
<h:head validity="true"></h:head>
<h:import type="css" path="/setup/css/main.css"></h:import>
```

import 标签用来引入外部的 js 和 css，并且都是用绝对路径。
如上图所示，type 一定要给 css 或者 js，path 是引用文件的绝对路径

3.10.5 h:tip

tip 是一个提示工具，用来创建一个图标，默认是一个问号图标也可以自定，鼠标移动上去之后出现提示信息。

tip 标签一共有 3 个属性

属性	意义	备注
title	提示信息	必填
clazz	Css 样式名称	选填
src	图标的绝对地址	选填，默认为问号

3.10.6 h:verifycode

verifycode 是一个验证码生成标签，会在标签位置生成一个图片验证码，并且验证码记录在对应的 session 中。

Verifycode 一共有 3 个属性

属性	意义	备注
width	图片高度	
height	图片宽度	
var	验证码在 session 中的 key	默认为 rand

3.10.7 h:grid

grid 是一个列表页组件，每个列表页必须使用此标签，列表页数据表格将会显示在此标签的位置。
grid 只有一个属性为 id,这个 id 是 dom 的 id 如果页面上只有一个 grid 这个属性可以忽略不写

3.11 国际化

国际化文件名称为 messages_en_US.properties(英语)、messages_zh_CN.properties(中文)
其中配置文件为键值对格式
如:
right.used=权限名称已经被占用

```
#成功提示
add.success=新增成功
modify.success=编辑成功
remove.success=删除成功
add.fail=新增失败
modify.fail=编辑失败
remove.fail=删除失败
right.wrong=您没有此操作的权限
```

调用这些提示使用 SpringUtil 的 getMessage 方法，传入 key 即可

3.12 工具类

这里只列出工具类和相应的作用，具体方法参见 api

类名称	作用
CacheUtil	Ehcache 工具，可以创建和操作缓存
DateUtil	日期操作类，可以转换日期格式
ExcelUtil	操作 excel 文件

FileUtil	操作各种文件
FreemarkerUtil	解析 freemarker 模板
HttpBLF	模拟 http 请求
ImageUtil	图片工具类，目前只支持图片缩放
JsonUtil	Object 转换为 json 字符串
MailSend	邮件发送工具
MathUtil	数学运算工具
Md5Util	Md5 加解密工具
NumberUtil	数字工具，object 转各种数字类型
PinyinUtil	汉字转全音、首字母
Properties	Properties 文件读写工具
RarFile	Rar 文件解压工具，不支持压缩
ReflectionUtils	反射工具
SpringUtil	Spring 工具、可以获得 spring 的 bean
StringUtil	字符串操作工具，各种都有
ZipUtil	Zip 压缩解压缩工具
ControllerUtil	Controller 层工具、可以操作 cookie，ip
MultipartFileInfo	获取上传文件的信息工具
Script	Controller 输出 js 工具
XmlDocument	Xml 解析工具
XmlObject	Xml 与 object 转换工具
IpUtils	Ip 工具，可以获得 ip 的地域信息

3.13 系统变量

JAVA 变量

类	方法	意义
BaseInfo	isPrepared	系统是否注册并初始化

BaseInfo	getRealPath	获得系统的物理路径
BaseInfo	getContextPath	获得系统应用路径
BaseInfo	getDbType	获得系统数据库类型
BaseInfo	isReg	是否注册
BaseInfo	getAppName	获得系统名称
BaseInfo	isKick	是否踢人
BaseInfo	isSso	是否开启 sso
BaseInfo	getPagesPath	jsp 的路径
Settings	getSettings	获得 Setting 实例
Settings	isEnableVerifyCode	是否开启验证码
Settings	getFileTmp	获得文件上传暂存目录
Settings	getImageDir	获得高级编辑器图片上传目录
Settings	getAttachmentDir	获得高级编辑器附件上传目录

以上路径都无 “/” 结尾

JSP(JSTL)变量，必须使用 h:head 标签才生效

key	意义	用法
contextPath	获得系统应用路径	${contextPath}

以上路径都无 “/” 结尾

3.14 JS 函数

3.14.1 列表页中的功能按钮函数

```
function toolbarAction(action) { 1
    switch (action) {
    case 'remove': 2
        var ids = getCheckedIds();
        if (ids == "") {
            alert("未选中任何记录");
            return;
        }
        confirm("您确定要删除这" + ids.split(",").length + "条记录吗", function() {
            ajaxSumbit("remove.do?pid=${pid}&ids=" + ids);
        });
        break;
    case 'add': 3
        openDialog('group/add_show.do?pid=${pid}', 550, 560, {
            title : '机构新增'
        });
        break;
    case 'import':
        openDialog('group/import_show.do', 550, 470, {
            title : '机构导入'
        });
        break;
    case 'export':
        var ids = getCheckedIds();
        window.open("export.do?ids=" + ids + "&pid=${pid}");
        break;
    }
}
```

1、 列表页上的功能按钮的函数名称为 toolbarAction, 参数 action
2、 执行按钮参数为 remove 的方法
3、 执行按钮参数为 add 的方法

3.14.2 列表页相关函数

方法名称	作用
getCheckedIds	获得勾选了的 checkbox 的值
getCheckedAttr(fieldname)	获得勾选了的列的值
updateCell(index, cellValues)	更新列，第一个参数为列的 index，第二个值为列的值

3.14.3 对话框

方法名称	作用
openDialog(url, width, height, options, id)	打开对话框,第一个参数为 url,第二个宽度,第三个高度,第四个配置项,第五个为 id,通常使用前 3 个参数即可
getParentWindow	获得触发对话框的页面的 window 对象
refreshParentWindow	刷新触发对话框的页面
getDialog	获得当前对话框
closeDialog(refresh)	关闭对话框,参数为是否刷新父页面

3.14.4 提示

系统对 alert 和 confirm 做了封装,使用时需要注意和以往不同

1、alert

alert 和以前一样直接 alert('xxxx')就可以了,如果需要别的样式的 alert 可以给第二个参数 alert('xxxx','info'),如果想点了确定之后做些回调就可以给第三个参数,alert('xxxx','error',function(){xxxxxx})

类型有 error、info、question、warning

2、confirm

confirm 和以前的区别最大

confirm('提示信息',functio(){},function(){}),第一个参数是提示信息,第二个参数是点击了确定后调用的函数,第三个参数为点击取消后调用的函数。

3.14.5 AJAX 提交

1、使用了验证框架的 form,将会自动使用 ajax 提交。

如果遇到上传文件就不可以使用 ajax 了,一般我们选用 iframe 方式提交,在使用验证框架时我们这样写:注意红字部分

```
$('#mainForm').validity(
    function(){
        $('#appname').require('应用名称必须填写');
        $('#password').require('密码必须填写');
    }
,{type:'iframe'});
```

如果只用验证框架的验证,不用提交的话上面的红字部分换成:{auto:false}

2、没有使用验证框架的 form 可以给 form 加一个属性为 data-posttype，值为 ajax，将会自动使用 ajax 提交。

3、直接访问地址方式需要使用 ajaxSubmit(url,setting)这个函数来触发 ajax 提交，比如列表页的删除功能
setting 是 ajax 提交的相关属性一共有 4 个：type:'post'，error:function(){}，success:function(msg){}，data:data

3.14.6 树

```
<script type="text/javascript">
    $(function() {
        var zNodes = ${tree}; 1
        var setting = { 2
            async : {
                enable : true,
                url : '${ctx}/manager/menu/menuwithurlforgroup_search.do',
                autoParam : [ "id=groupId", "isDisabled" ],
                type : 'get',
            }
        };
        $("#tree").tree(setting, zNodes); 3
    });
</script>
<style type="text/css">
body {
    margin: 5px;
}
</style>
</head>
<body>
    <ul id="tree" class="ztree"></ul> 4
</body>
```

1、 获得树的节点数据，普通 json 格式
2、 配置树，具体配置项参看 ztree 的 api
3、 创建树，传入配置和节点，创建树使用的 id 要与对应 html 元素的 id 一样
4、 树的 html 元素，必须是 ul

树的 js 操作，方法调用的格式为$('xxxx').tree().方法()

方法名称	作用
expandAll(expand)	展开树，参数为是否展开
addNodes(nodes)	新增节点，参数为节点 json 数据
removeNode(id)	按照 id 删除节点
updateNode(node)	更新节点，参数为节点 json 数据
refreshNode(id,type, isSilent)	刷新 ajax 节点，第一个参数为节点 id

3.14.7 SELECT 自动选中

```
<select id="dbType" name="dbType" style="width: 182px;" data-value="${dbtype}">
    <option value="1">ORACLE</option>
    <option value="2">MSSQL</option>
    <option value="5">MYSQL</option>
</select>
```

当给 select 加上 data-value 属性，值给相应的 option 的值，对应的 option 将会被选中；例如这里 dat-type="5"，mysql 将会被选中。

3.14.8 CHECKBOX 自动选中

```
<input type="checkbox" name="sync" value="1" data-value="${sync }"/>
```

当 data-value 的值与 value 的值相等或者包含时选中

3.14.9 RADIO 自动选中

```
<label>
    <input type="radio" name="enableVerifyCode"
        value="true" data-value="${setting.enableVerifyCode }"/>开
</label>
<label>
    <input type="radio" name="enableVerifyCode"
        value="false" data-value="${setting.enableVerifyCode }"/>关
</label>
```

当 data-value 的值与 value 的值相等时选中

3.14.10 JS 验证框架

3.14.10.1 引入 JS 库

```
<h:head validity="true"></h:head>
```

3.14.10.2 使用方法

在 form 表单中，id 为“myform”。在表单提交时，对<input>标签进行输入验证，方法如下：

```
$(function(){
    $("#myform").validity(function() {
            $("#name").require("名称不能为空")
                    .maxLength(33, "名称不能超过33个字");
    });
});
```

```
<form method="post"id="myform">
    <inputtype="text"id="name"name="name"value=""/>
</form>
```

3.14.10.3 参数说明

```
$(function(){
    $("#myform").validity(function() {
        .........验证过程
    },{ type:'',
        auto:,
        beforeSubmit:function(result){},
        success:function(){},
        error:function(){}
    });
});
```

type（String 型）：表示提交方式，默认为 ajax，可以改成 iframe 表单就会自动的使用隐藏的 iframe 提交，不需要在表单上指定也不需要自己写隐藏的 iframe

auto（boolean 型）：表示是否自动提交，默认为 true，设为 false 之后只会验证不会提交表单

beforeSubmit（函数型）：在提交之前做什么，result 参数为验证是否通过，在 beforeSubmit 方法的最后返回 false 可以阻止表单的提交，即使验证通过也不会提交。

success（函数型）：请求成功回调函数
error（函数型）：请求失败回调函数

3.14.10.4 函数 API

函数名称	参数	说明
require(msg)	msg：提示信息	必填验证。如果元素的值为空，即弹出提示，信息为 msg。
match(rule, msg)	rule:匹配的正则 msg:提示信息	正则匹配。如果元素的值不能匹配 rule 规则，即弹出提示，信息为 msg。
range(min,max,msg)	min:最小数字 max:最大数字 msg：提示信息	验证值的范围。如果元素的值大于 max、或者小于 min，即弹出提示，信息为 msg。
maxLength(max,msg)	max:最大长度 msg：提示信息	最大长度验证。元素的值的长度超过 max，即弹出提示，信息为 msg
minLength(min,msg)	min:最小长度 msg:提示信息	最小长度验证。元素的值的长度小于 min，即弹出提示，信息为 msg
assert(function,msg)	function:自定义	自定义验证。如果 function 返回 true，

	验证的函数 msg:提示信息	即通过验证,否则验证失败,弹出提示,信息为 msg
radioChecked(val, msg)	val:radio 的某个元素的值 msg: 提示信息	判断 radio 的值为 val 的某个元素是否被选中。如果未选中,即弹出提示,信息为 msg
radioNotChecked(val, msg)	val:radio 的某个元素的值 msg: 提示信息	判断 radio 的值为 val 的某个元素是否未被选中。如果选中,即弹出提示,信息为 msg
checkboxChecked(msg)	msg: 提示信息	判断 checkbox 是否被选中。如果未选中,即弹出提示,信息为 msg
nonHtml(msg)	msg: 提示信息	验证是否不包含特殊字符,如果包含,即弹出提示,信息为 msg

3.14.10.5 正则表达式的属性

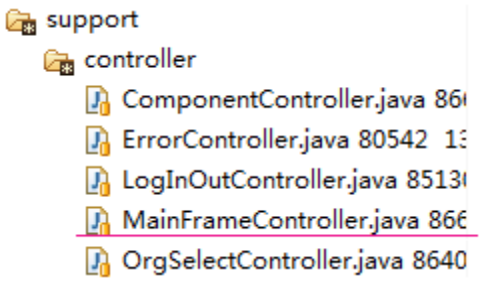
Datatype 属性	说明
intege1	正整数
intege2	负整数
num	数字
Num1	正数 (正整数 + 0)
Num2	负数 (负整数 + 0)
decmal	浮点数
decmal1	正浮点数
decmal2	负浮点数
decmal3	浮点数
decmal4	非负浮点数 (正浮点数 + 0)
decmal5	非正浮点数 (负浮点数 + 0)
email	邮件
color	颜色
chinese	仅中文
ascii	仅 ACSII 字符
zipcode	邮编
mobile	手机
ip4	ip 地址
picture	图片
rar	rar 压缩文件
zip	zip 压缩文件
qq	QQ 号码
tel	国内电话

username	用来用户注册。匹配由数字、26 个英文字母或者下划线组成的字符串
username1	用来用户注册。匹配字母、数字、下划线、中文组成，不能以下划线开头和结尾
letter	字母
letter_u	大写字母
letter_l	小写字母
codeid	机构标识。大小写字母及数字
idcard	身份证
url	URL
nonHtml	不包括 HTML 标签

4 SETUP

项目或者产品中的 setup 都写入 AppSetupController,对应的 jsp 为/setup/appsetup.jsp,可以随便改里面的内容。

5 顶层菜单



顶层菜单项的 controller 在 MainFrameController 中设置，jsp 在/pages/support/mainframe.jsp

6 规范

具体规范参照之前的编码规范，这里只做补充

6.1 Controller

1、Controller 命名：

所有的 Controller 都以：操作+模块+Controller 来命名。

例如：

操作用户模块 OprUserController，用户列表 ListUserController

2、Controller 的 RequestMapping 命名：

Controller 类上的 RequestMapping 路径一般只到模块名称也就是全路径的最后一层，并且前后不用斜杠"/"。

例如：

127.0.0.1:9091/springmvc/manager/user/list.do

RequestMapping 路径为 manager/user

3、方法的 RequestMapping 命名：

Controller 方法上的 RequestMapping 一般定义为操作名称，并且不用斜杠"/"开头，不用'.do'结尾。

例如：

127.0.0.1:9091/springmvc/manager/user/list.do，RequestMapping 路径为 list

对于需要打开页面的 opr 操作我们使用路径为：功能_show，比如打开新增页面 add_show

接受 form 提交的路径为：功能_submit，比如提交新增 add_submit

打开修改页面：modify_show，提交修改：modify_submit，删除：remove

4、方法命名：

Controller 的方法命名源自于 RequestMapping 的命名

例如：

RequestMapping 为 list，方法为 list

RequestMapping 为 add_show，方法为 showAdd

RequestMapping 为 add_submit，方法为 submitAdd

RequestMapping 为 modify_show，方法为 showModify

RequestMapping 为 modify_submit，方法为 submitModify

RequestMapping 为 remove，方法为 remove

6.2 JSP 命名

jsp 的命名采用：模块名_功能 来命名。

例如

列表页：user_list.jsp
操作页：user_opr.jsp