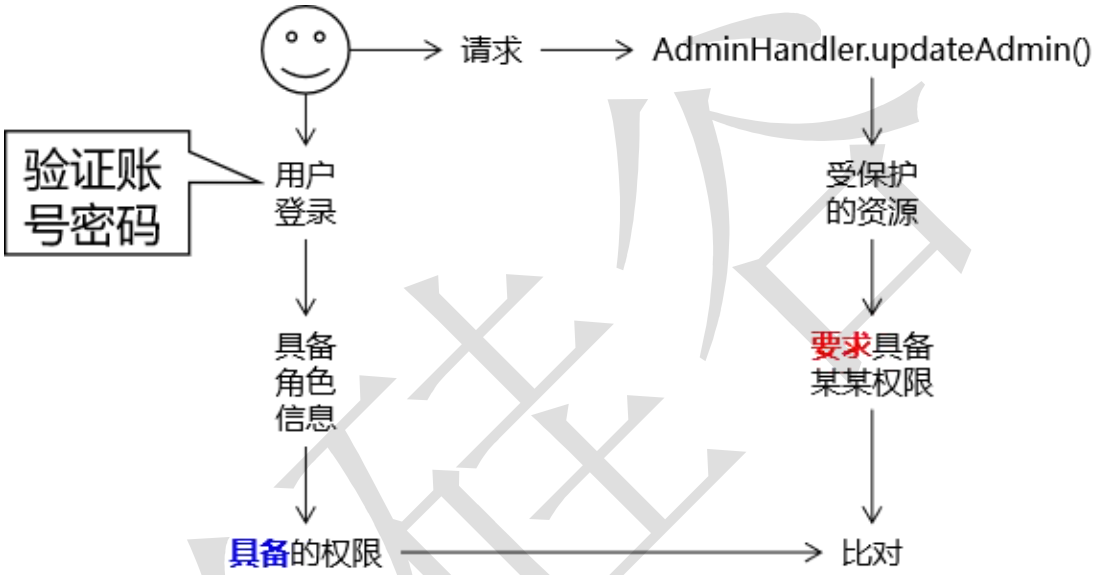
尚筹网

[09-后台管理系统-分配]

1 权限控制

权限控制机制的本质就是“用钥匙开锁”。

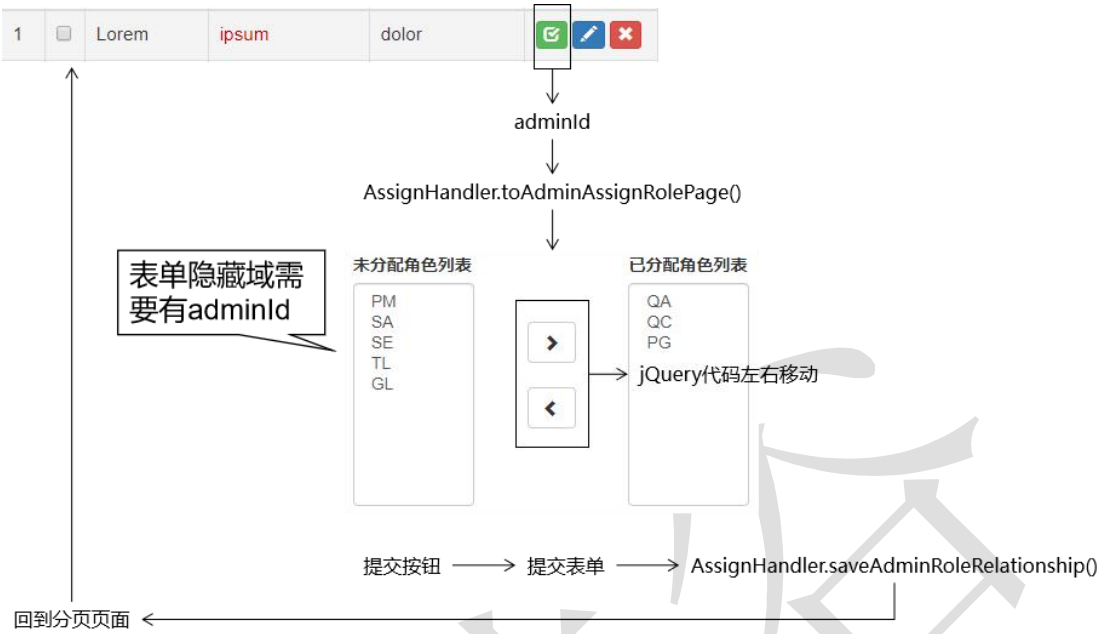


2 给 Admin 分配 Role

2.1 目标

通过页面操作把 Admin 和 Role 之间的关联关系保存到数据库。

2.2 思路



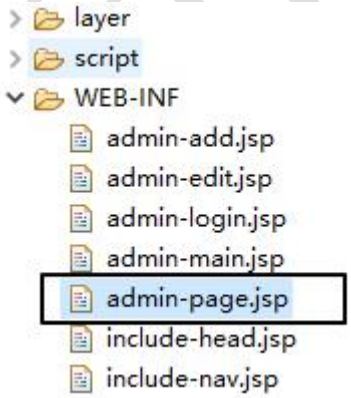
2.3 代码：前往分配页面

2.3.1 创建保存 Admin-Role 关联关系的数据库表

```
CREATE TABLE `project_crowd`.`inner_admin_role` ( `id` INT NOT NULL AUTO_INCREMENT, `admin_id` INT, `role_id` INT, PRIMARY KEY (`id`) );
```

这个表并不对应现实生活中或项目业务功能中的一个具体实体，所以没有对应的实体类，也不通过逆向工程做逆向生成。

2.3.2 修改“分配”按钮

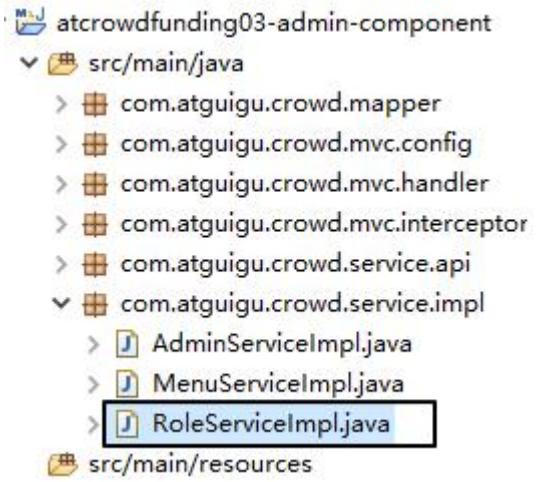



```
<a href="assign/to/assign/role/page.html?adminId=${admin.id }&pageNum=${requestScope.pageInfo.pageNum }&keyword=${param.keyword }" class="btn btn-success btn-xs"><i class=" glyphicon glyphicon-check"></i></a>
```

2.3.3 创建 AssignHandler

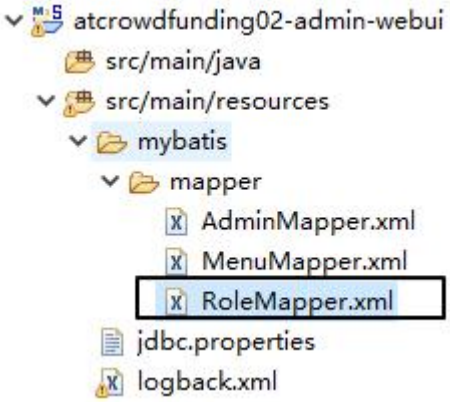

```
@RequestMapping("/assign/to/assign/role/page.html")
public String toAssignRolePage(

        @RequestParam("adminId") Integer adminId,

        ModelMap modelMap

        ) {

    // 1.查询已分配角色
    List<Role> assignedRoleList = roleService.getAssignedRole(adminId);

    // 2.查询未分配角色
    List<Role> unAssignedRoleList = roleService.getUnAssignedRole(adminId);

    // 3.存入模型（本质上其实是：request.setAttribute("attrName",attrValue);
    modelMap.addAttribute("assignedRoleList", assignedRoleList);
    modelMap.addAttribute("unAssignedRoleList", unAssignedRoleList);

    return "assign-role";
}
```

2.3.4 RoleService 中的方法




```
@Override
public List<Role> getAssignedRole(Integer adminId) {

    return roleMapper.selectAssignedRole(adminId);
}

@Override
public List<Role> getUnAssignedRole(Integer adminId) {
    return roleMapper.selectUnAssignedRole(adminId);
}
```

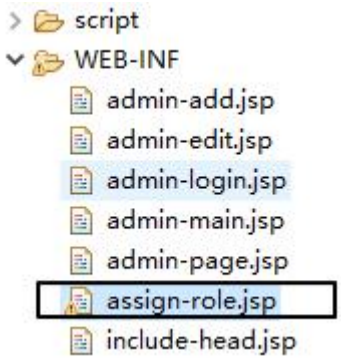
2.3.5 SQL 语句




```
<select id="selectAssignedRole" resultMap="BaseResultMap">
    select id,name from t_role where id in (select role_id from inner_admin_role where admin_id=#{adminId})
</select>
<select id="selectUnAssignedRole" resultMap="BaseResultMap">
    select id,name from t_role where id not in (select role_id from inner_admin_role where
```

```
admin_id=#{adminId})
</select>
```

2.3.6 在页面上显示角色数据



对 option 标签进行说明：

```
<option value="将来在提交表单时一起发送给 handler 的值">在浏览器上让用户看到的数据</option>
```

实际显示角色信息时：

```
<option value="角色的 id">角色的名称</option>
```

举例：

```
<option value="5">市场部经理</option>
```

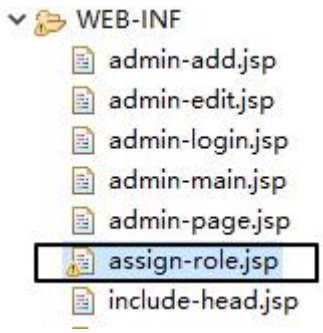
页面具体代码：

```
<c:forEach items="${requestScope.unAssignedRoleList }" var="role">
    <option value="${role.id }">${role.name }</option>
</c:forEach>
```

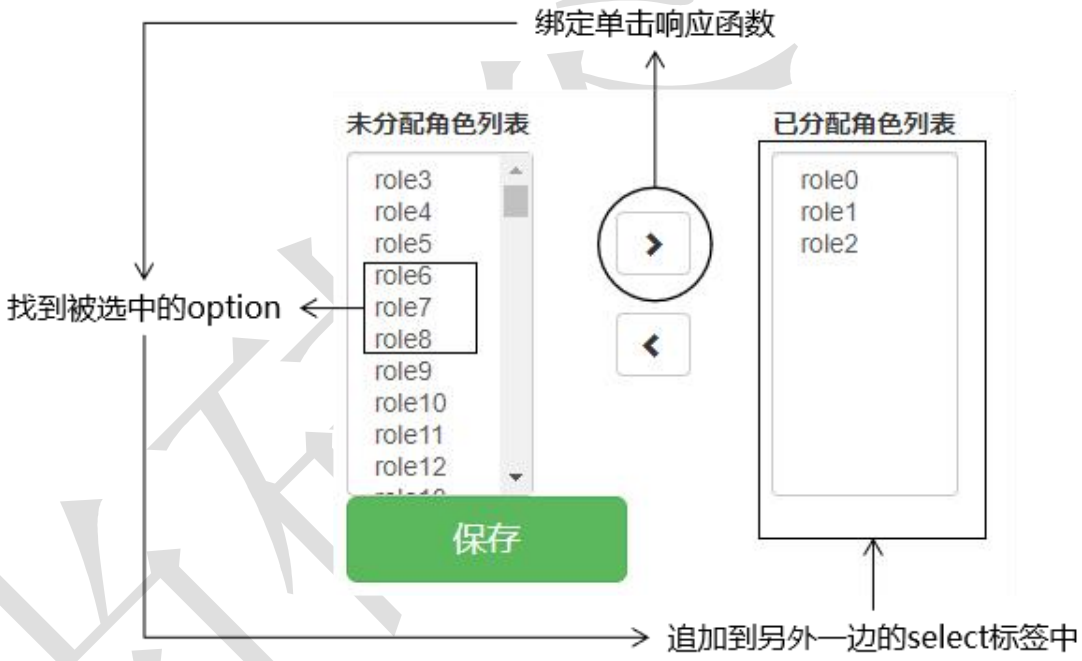
```
<c:forEach items="${requestScope.assignedRoleList }" var="role">
    <option value="${role.id }">${role.name }</option>
</c:forEach>
```

2.3.7 调整表单让表单能够提交数据

参考下面文件：



2.3.8 jQuery 代码




```
$("#toRightBtn").click(function(){

    // select 是标签选择器
    // :eq(0)表示选择页面上的第一个
    // :eq(1)表示选择页面上的第二个
    //  “>”表示选择子元素
    // :selected 表示选择“被选中的”option
    // appendTo()能够将 jQuery 对象追加到指定的位置
    $("select:eq(0)>option:selected").appendTo("select:eq(1)");

});
$("#toLeftBtn").click(function(){

    // select 是标签选择器
    // :eq(0)表示选择页面上的第一个
```

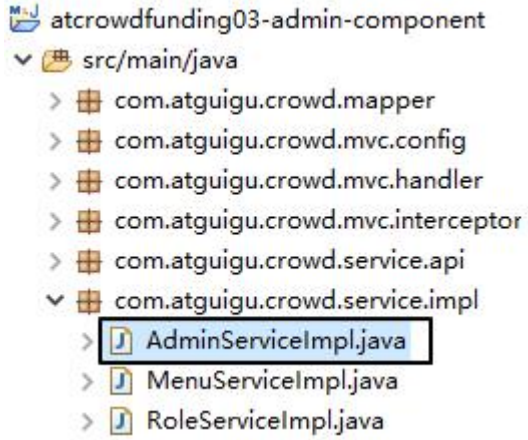
```
    // :eq(1)表示选择页面上的第二个
    // “>”表示选择子元素
    // :selected 表示选择“被选中的”option
    // appendTo()能够将 jQuery 对象追加到指定的位置
    $("select:eq(1)>option:selected").appendTo("select:eq(0)");

});
```

2.4 代码：执行分配

2.4.1 handler 方法

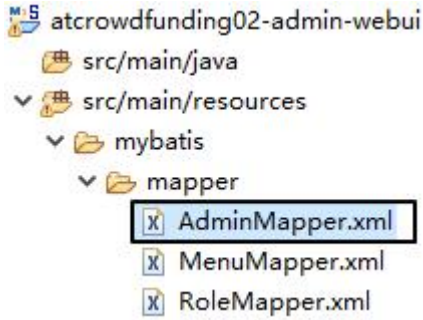

```
@RequestMapping("/assign/do/role/assign.html")
public String saveAdminRoleRelationship(
            @RequestParam("adminId") Integer adminId,
            @RequestParam("pageNum") Integer pageNum,
            @RequestParam("keyword") String keyword,

            // 我们允许用户在页面上取消所有已分配角色再提交表单，所以可以不提供 roleIdList 请求参数
            // 设置 required=false 表示这个请求参数不是必须的
            @RequestParam(value="roleIdList", required=false) List<Integer> roleIdList
        ) {

    adminService.saveAdminRoleRelationship(adminId, roleIdList);

    return "redirect:/admin/get/page.html?pageNum="+pageNum+"&keyword="+keyword;
}
```

2.4.2 Service 方法




```
@Override
public void saveAdminRoleRelationship(Integer adminId, List<Integer> roleIdList) {

    // 旧数据如下：
    // adminId    roleId
    // 1          1（要删除）
    // 1          2（要删除）
    // 1          3
    // 1          4
    // 1          5
    // 新数据如下：
    // adminId    roleId
    // 1          3（本来就有）
    // 1          4（本来就有）
    // 1          5（本来就有）
    // 1          6（新）
    // 1          7（新）
    // 为了简化操作：先根据 adminId 删除旧的数据，再根据 roleIdList 保存全部新的数据

    // 1.根据 adminId 删除旧的关联关系数据
    adminMapper.deleteOLdRelationship(adminId);

    // 2.根据 roleIdList 和 adminId 保存新的关联关系
    if(roleIdList != null && roleIdList.size() > 0) {
        adminMapper.insertNewRelationship(adminId, roleIdList);
    }
}
```

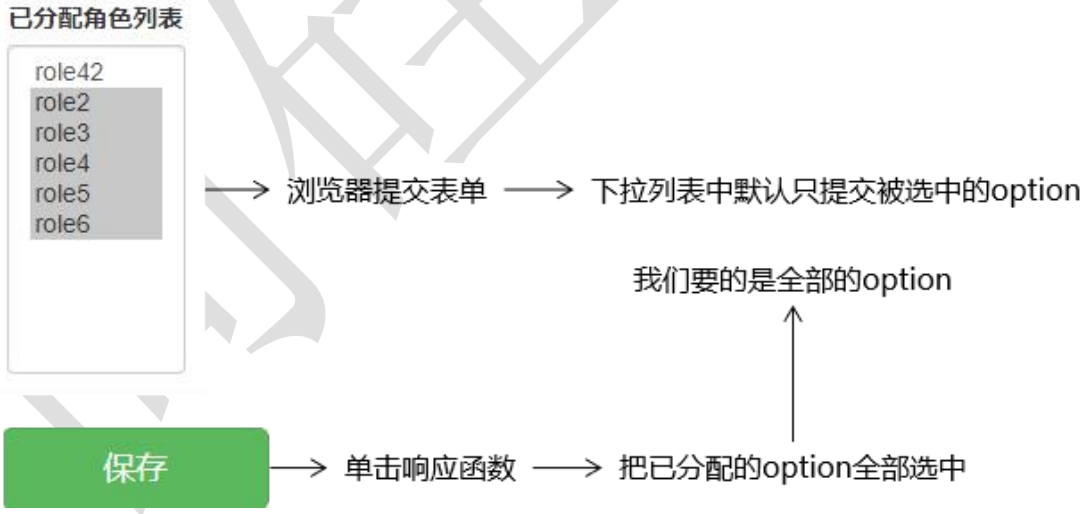
2.4.3 SQL 语句




```
    <delete id="deleteOLdRelationship">
        delete from inner_admin_role where admin_id=#{adminId}
    </delete>
```

```
<insert id="insertNewRelationship">
    insert into inner_admin_role(admin_id,role_id) values
    <foreach collection="roleIdList" item="roleId"
separator=",">(#{adminId},#{roleId})</foreach>
</insert>
```

2.4.4 修正 Bug




```
$("#submitBtn").click(function(){
    // 在提交表单前把“已分配”部分的 option 全部选中
    $("select:eq(1)>option").prop("selected","selected");

    // 为了看到上面代码的效果，暂时不让表单提交
    // return false;
});
```

3 给 Role 分配 Auth

3.1 目标

把角色和权限的关联关系保存到数据库。

3.2 思路

3.3 代码：t_auth 建表

```
CREATE TABLE `t_auth` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `name` varchar(200) DEFAULT NULL,
  `title` varchar(200) DEFAULT NULL,
```

```
  `category_id` int(11) DEFAULT NULL,
  PRIMARY KEY (`id`)
);
```

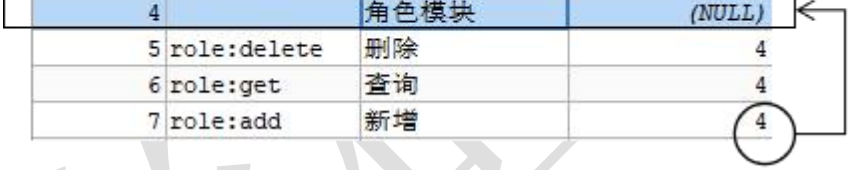
```
INSERT INTO t_auth(id,`name`,title,category_id) VALUES(1,'','用户模块',NULL);
INSERT INTO t_auth(id,`name`,title,category_id) VALUES(2,'user:delete','删除',1);
INSERT INTO t_auth(id,`name`,title,category_id) VALUES(3,'user:get','查询',1);
INSERT INTO t_auth(id,`name`,title,category_id) VALUES(4,'','角色模块',NULL);
INSERT INTO t_auth(id,`name`,title,category_id) VALUES(5,'role:delete','删除',4);
INSERT INTO t_auth(id,`name`,title,category_id) VALUES(6,'role:get','查询',4);
INSERT INTO t_auth(id,`name`,title,category_id) VALUES(7,'role:add','新增',4);
```

name 字段：给资源分配权限或给角色分配权限时使用的具体值，将来做权限验证也是使用 name 字段的值来进行比对。建议使用英文。

title 字段：在页面上显示，让用户便于查看的值。建议使用中文。

category_id 字段：关联到当前权限所属的分类。这个关联不是到其他表关联，而是就在当前表内部进行关联，关联其他记录。所以说，t_auth 表中是依靠 category_id 字段建立了“节点”之间的父子关系。

id	name	title	category_id
1		用户模块	*(NULL)*
2	user:delete	删除	1
3	user:get	查询	1
4		角色模块	*(NULL)*
5	role:delete	删除	4
6	role:get	查询	4
7	role:add	新增	4



name 字段中值的格式：中间的“:”没有任何特殊含义。不论是我们自己写的代码还是将来使用的框架都不会解析“:”。如果不用“:”，用“%、@、&、*、-”等等这样的符号也都是可以的。

模块:操作名

user:delete

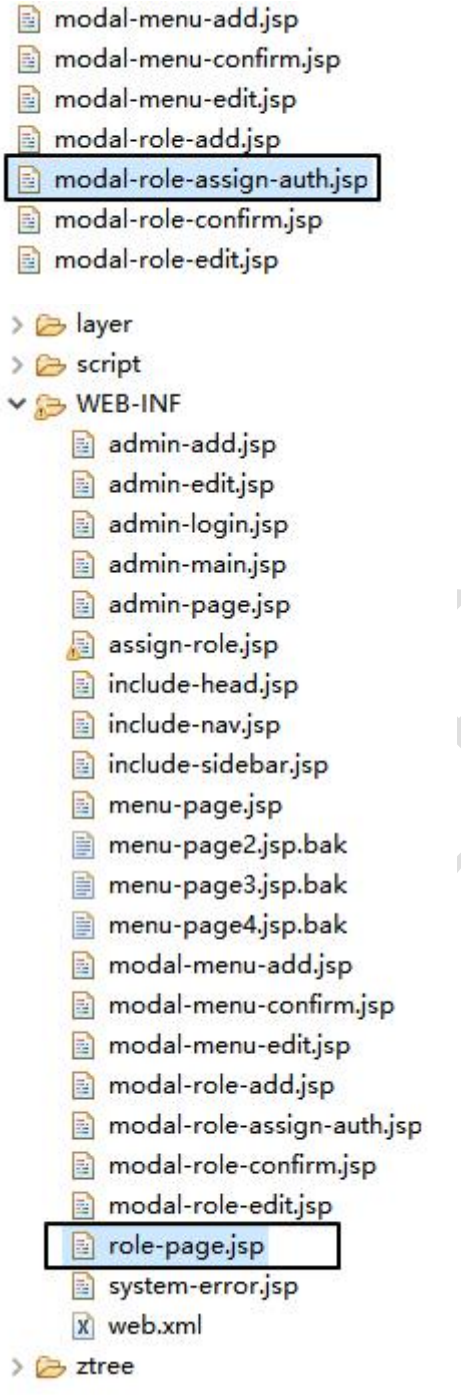
user:get

role:delete

……

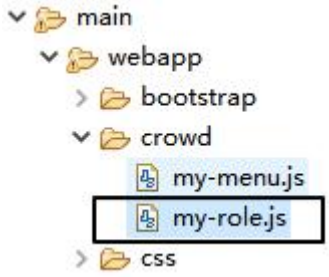
3.4 代码：打开模态框

3.4.1 准备模态框



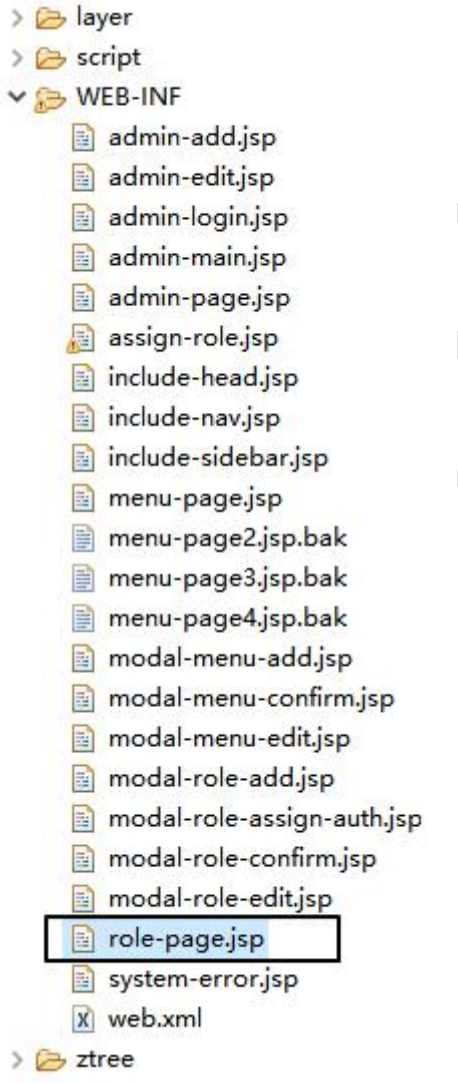

```
<%@include file="/WEB-INF/modal-role-assign-auth.jsp" %>
```

3.4.2 给“☑”按钮绑定单击响应函数



在 fillTableBody()函数中，修改 checkBtn

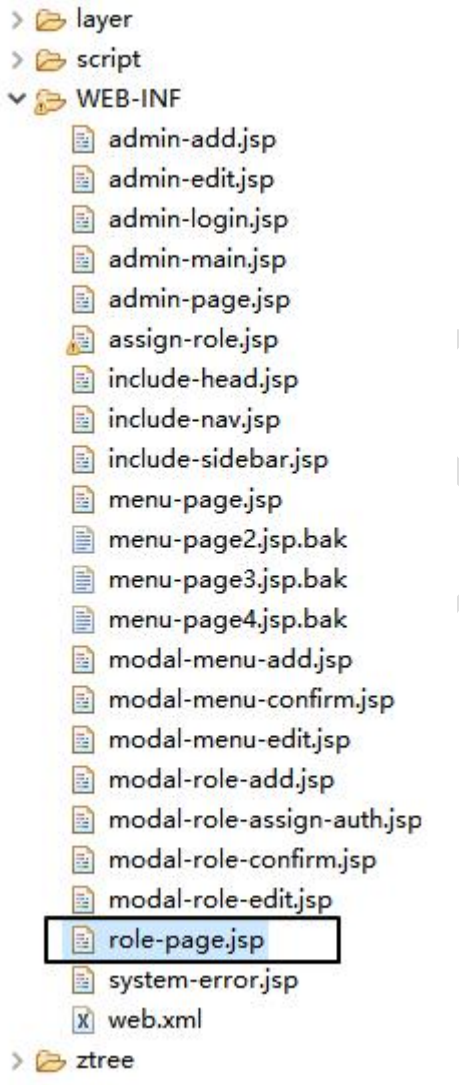
```
var checkBtn = "<button id='"+roleId+"' type='button' class='btn btn-success btn-xs checkBtn'><i class=' glyphicon glyphicon-check'></i></button>";
```




```
// 13.给分配权限按钮绑定单击响应函数
$("#rolePageBody").on("click",".checkBtn",function(){

```

```
    // 打开模态框
    $("#assignModal").modal("show");

    // 在模态框中装载树 Auth 的形结构数据
    fillAuthTree();
});
```

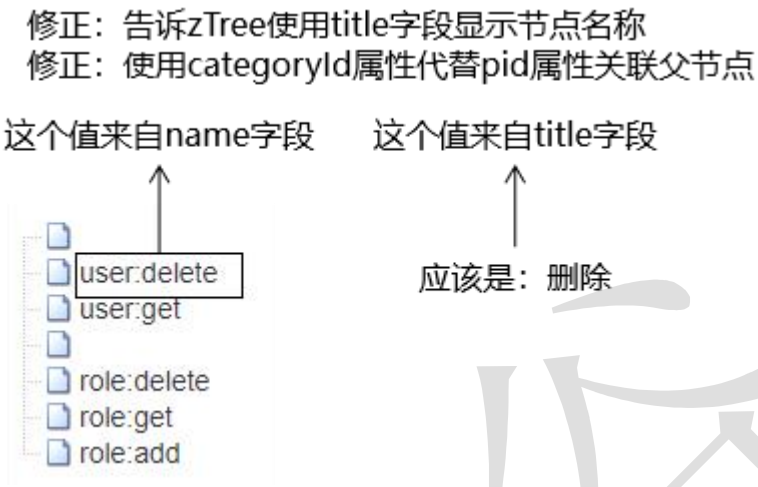
3.4.3 加入 zTree 的环境



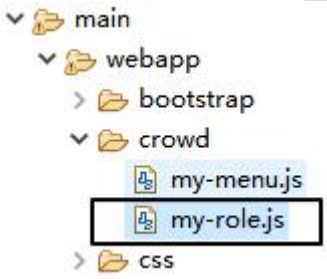

```
<%@include file="/WEB-INF/include-head.jsp"%>
<link rel="stylesheet" href="css/pagination.css" />
<script type="text/javascript" src="jquery/jquery.pagination.js"></script>
<link rel="stylesheet" href="ztree/zTreeStyle.css"/>
<script type="text/javascript" src="ztree/jquery.ztree.all-3.5.min.js"></script>
<script type="text/javascript" src="crowd/my-role.js"></script>
```

注意：引入的时候 zTree 的 JavaScript 库文件要放在 my-role.js 的前面

3.4.4 显示不正常分析



3.4.5 函数 fillAuthTree()




```
// 声明专门的函数用来在分配 Auth 的模态框中显示 Auth 的树形结构数据
function fillAuthTree() {

    // 1.发送 Ajax 请求查询 Auth 数据
    var ajaxReturn = $.ajax({
        "url":"assgin/get/all/auth.json",
        "type":"post",
        "dataType":"json",
        "async":false
    });

    if(ajaxReturn.status != 200) {
        layer.msg("请求处理出错！响应状态码是：" +ajaxReturn.status+ " 说明是：
"+ajaxReturn.statusText);
        return ;
    }

    // 2.从响应结果中获取 Auth 的 JSON 数据
```

```
// 从服务器端查询到的 list 不需要组装成树形结构，这里我们交给 zTree 去组装
var authList = ajaxReturn.responseJSON.data;

// 3.准备对 zTree 进行设置的 JSON 对象
var setting = {
    "data": {
        "simpleData": {

            // 开启简单 JSON 功能
            "enable": true,

            // 使用 categoryId 属性关联父节点，不用默认的 pId 了
            "pIdKey": "categoryId"
        },
        "key": {
            // 使用 title 属性显示节点名称，不用默认的 name 作为属性名了
            "name": "title"
        }
    },
    "check": {
        "enable": true
    }
};

// 4.生成树形结构
// <ul id="authTreeDemo" class="ztree"></ul>
$.fn.zTree.init($("#authTreeDemo"), setting, authList);

// 获取 zTreeObj 对象
var zTreeObj = $.fn.zTree.getZTreeObj("authTreeDemo");

// 调用 zTreeObj 对象的方法，把节点展开
zTreeObj.expandAll(true);

// 5.查询已分配的 Auth 的 id 组成的数组
ajaxReturn = $.ajax({
    "url":"assign/get/assigned/auth/id/by/role/id.json",
    "type":"post",
    "data":{
        "roleId":window.roleId
    },
    "dataType":"json",
```

```
            "async":false
        });

        if(ajaxReturn.status != 200) {
            layer.msg("请求处理出错！响应状态码是：  "+ajaxReturn.status+"  说明是：
"+ajaxReturn.statusText);
            return ;
        }

        // 从响应结果中获取 authIdArray
        var authIdArray = ajaxReturn.responseJSON.data;

        // 6.根据 authIdArray 把树形结构中对应的节点勾选上
        // ①遍历 authIdArray
        for(var i = 0; i < authIdArray.length; i++) {
            var authId = authIdArray[i];

            // ②根据 id 查询树形结构中对应的节点
            var treeNode = zTreeObj.getNodeByParam("id", authId);

            // ③将 treeNode 设置为被勾选

            // checked 设置为 true 表示节点勾选
            var checked = true;

            // checkTypeFlag 设置为 false，表示不“联动”，不联动是为了避免把不该勾选的勾
选上
            var checkTypeFlag = false;

            // 执行
            zTreeObj.checkNode(treeNode, checked, checkTypeFlag);
        }
}
```

3.4.6 创建角色到权限之间关联关系的中间表

```
CREATE TABLE `project_crowd`.`inner_role_auth` ( `id` INT NOT NULL AUTO_INCREMENT,
`role_id` INT, `auth_id` INT, PRIMARY KEY (`id`) );
```

3.4.7 根据 role_id 查询 auth_id

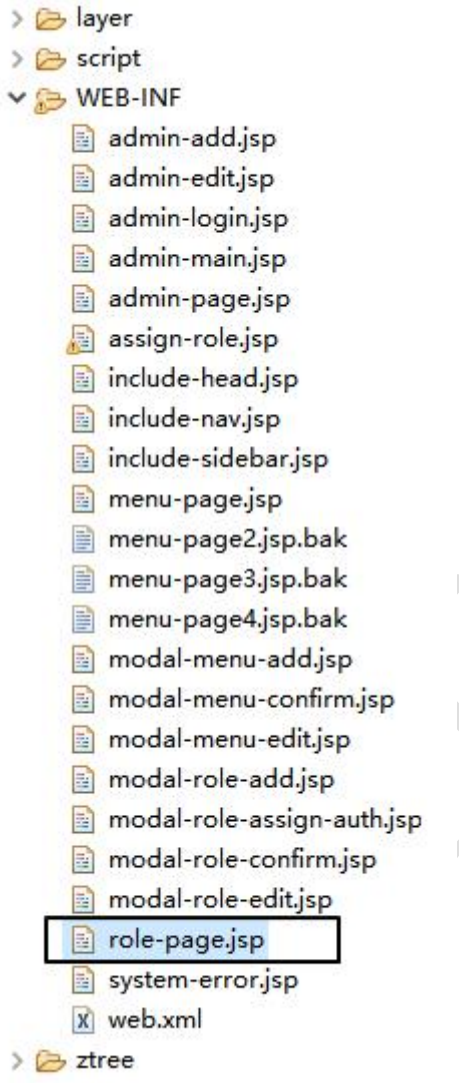
```
// handler 方法
@ResponseBody
@RequestMapping("/assign/get/assigned/auth/id/by/role/id.json")
public ResultEntity<List<Integer>> getAssignedAuthIdByRoleId(
        @RequestParam("roleId") Integer roleId) {

    List<Integer> authIdList = authService.getAssignedAuthIdByRoleId(roleId);

    return ResultEntity.successWithData(authIdList);
}
```

```
// service 方法
@Override
public List<Integer> getAssignedAuthIdByRoleId(Integer roleId) {
    return authMapper.selectAssignedAuthIdByRoleId(roleId);
}
```

```
// SQL
<select id="selectAssignedAuthIdByRoleId" resultType="int">
    select auth_id from inner_role_auth where role_id=#{roleId}
</select>
```

3.5 代码：执行分配

3.5.1 给“分配”按钮绑定单击响应函数



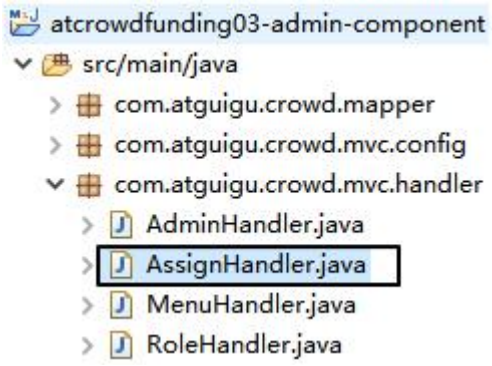

```
// 14.给分配权限模态框中的“分配”按钮绑定单击响应函数
$("#assignBtn").click(function(){

    // ①收集树形结构的各个节点中被勾选的节点
    // [1]声明一个专门的数组存放 id
    var authIdArray = [];

    // [2]获取 zTreeObj 对象
    var zTreeObj = $.fn.zTree.getZTreeObj("authTreeDemo");

    // [3]获取全部被勾选的节点
    var checkedNodes = zTreeObj.getCheckedNodes();
```

```
    // [4]遍历 checkedNodes
    for(var i = 0; i < checkedNodes.length; i++) {
        var checkedNode = checkedNodes[i];

        var authId = checkedNode.id;

        authIdArray.push(authId);
    }

    // ②发送请求执行分配
    var requestBody = {
        "authIdArray":authIdArray,

        // 为了服务器端 handler 方法能够统一使用 List<Integer>方式接收数据，roleId 也存入数组
        "roleId":[window.roleId]
    };

    requestBody = JSON.stringify(requestBody);

    $.ajax({
        "url":"assign/do/role/assign/auth.json",
        "type":"post",
        "data":requestBody,
        "contentType":"application/json;charset=UTF-8",
        "dataType":"json",
        "success":function(response){
            var result = response.result;
            if(result == "SUCCESS") {
                layer.msg("操作成功！");
            }
            if(result == "FAILED") {
                layer.msg("操作失败！"+response.message);
            }
        },
        "error":function(response) {
            layer.msg(response.status+" "+response.statusText);
        }
    });

    $("#assignModal").modal("hide");
```

```
});
```

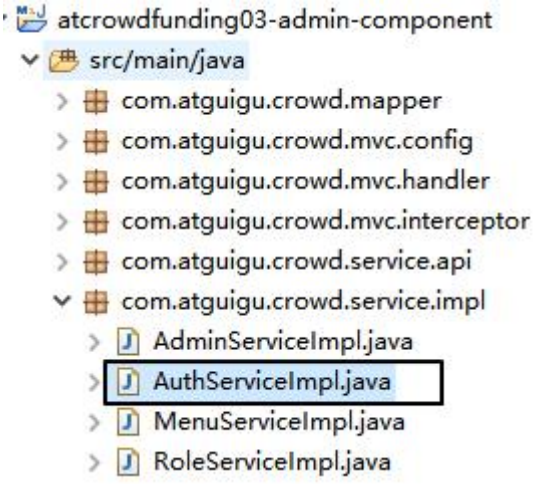
3.5.2 后端执行分配，保存关联关系

// handler 方法




```
@ResponseBody
@RequestMapping("/assign/do/role/assign/auth.json")
public ResultEntity<String> saveRoleAuthRelathinship(
        @RequestBody Map<String, List<Integer>> map) {

    authService.saveRoleAuthRelathinship(map);

    return ResultEntity.successWithoutData();
}
```

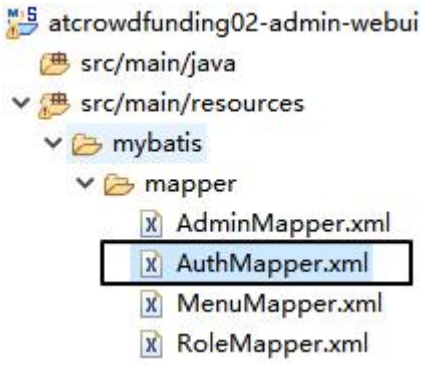
// service 方法




```
@Override
public void saveRoleAuthRelathinship(Map<String, List<Integer>> map) {

    // 1.获取 roleId 的值
    List<Integer> roleIdList = map.get("roleId");
    Integer roleId = roleIdList.get(0);
```

```
        // 2.删除旧关联关系数据
        authMapper.deleteOldRelationship(roleId);

        // 3.获取 authIdList
        List<Integer> authIdList = map.get("authIdArray");

        // 4.判断 authIdList 是否有效
        if(authIdList != null && authIdList.size() > 0) {
            authMapper.insertNewRelationship(roleId, authIdList);
        }
    }
```

<!-- SQL -->




```
<delete id="deleteOldRelationship">
    delete from inner_role_auth where role_id=#{roleId}
</delete>
<insert id="insertNewRelationship">
    insert into inner_role_auth(auth_id,role_id) values
    <foreach collection="authIdList" item="authId"
separator=",">(#{authId},#{roleId})</foreach>
</insert>
```

4 给 Menu 分配 Auth

略