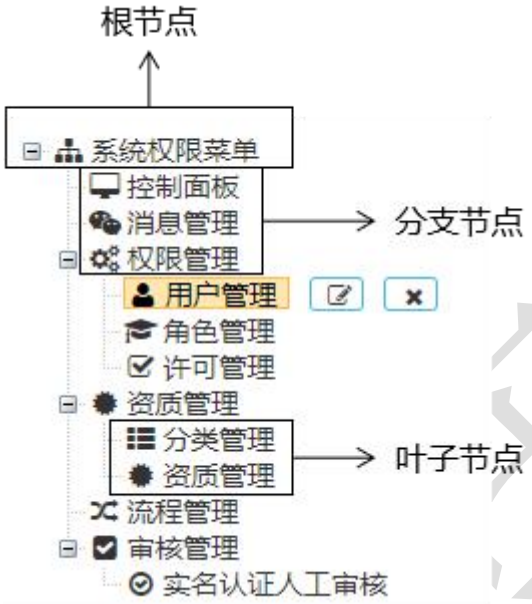
尚筹网

[08-后台管理系统-菜单维护]

1 树形结构基础知识介绍

1.1 节点类型



约定：整个树形结构节点的层次最多只能有 3 级。

1.2 在数据库表中表示树形结构

1.2.1 创建菜单的数据库表

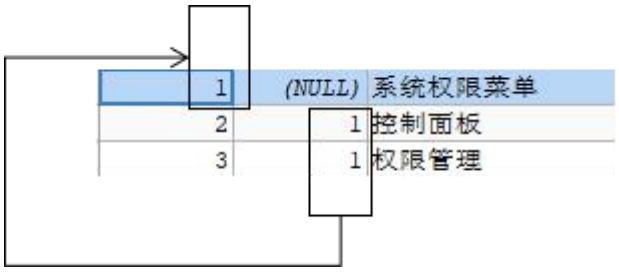
```
create table t_menu
(
    id                  int(11) not null auto_increment,
    pid                 int(11),
    name                  varchar(200),
    url                 varchar(200),
    icon                varchar(200),
    primary key (id)
);
```

1.2.2 插入数据

```
insert into `t_menu` (`id`, `pid`, `name`, `icon`, `url`) values('1',NULL,'系统权限菜单','glyphicon glyphicon-th-list',NULL);
insert into `t_menu` (`id`, `pid`, `name`, `icon`, `url`) values('2','1','控制面板','glyphicon glyphicon-dashboard','main.htm');
insert into `t_menu` (`id`, `pid`, `name`, `icon`, `url`) values('3','1','权限管理','glyphicon glyphicon glyphicon-tasks',NULL);
insert into `t_menu` (`id`, `pid`, `name`, `icon`, `url`) values('4','3','用户维护','glyphicon glyphicon-user','user/index.htm');
insert into `t_menu` (`id`, `pid`, `name`, `icon`, `url`) values('5','3','角色维护','glyphicon glyphicon-king','role/index.htm');
insert into `t_menu` (`id`, `pid`, `name`, `icon`, `url`) values('6','3','菜单维护','glyphicon glyphicon-lock','permission/index.htm');
insert into `t_menu` (`id`, `pid`, `name`, `icon`, `url`) values('7','1','业务审核','glyphicon glyphicon-ok',NULL);
insert into `t_menu` (`id`, `pid`, `name`, `icon`, `url`) values('8','7','实名认证审核','glyphicon glyphicon-check','auth_cert/index.htm');
insert into `t_menu` (`id`, `pid`, `name`, `icon`, `url`) values('9','7','广告审核','glyphicon glyphicon-check','auth_adv/index.htm');
insert into `t_menu` (`id`, `pid`, `name`, `icon`, `url`) values('10','7','项目审核','glyphicon glyphicon-check','auth_project/index.htm');
insert into `t_menu` (`id`, `pid`, `name`, `icon`, `url`) values('11','1','业务管理','glyphicon glyphicon-th-large',NULL);
insert into `t_menu` (`id`, `pid`, `name`, `icon`, `url`) values('12','11','资质维护','glyphicon glyphicon-picture','cert/index.htm');
insert into `t_menu` (`id`, `pid`, `name`, `icon`, `url`) values('13','11','分类管理','glyphicon glyphicon-equalizer','certtype/index.htm');
insert into `t_menu` (`id`, `pid`, `name`, `icon`, `url`) values('14','11','流程管理','glyphicon glyphicon-random','process/index.htm');
insert into `t_menu` (`id`, `pid`, `name`, `icon`, `url`) values('15','11','广告管理','glyphicon glyphicon-hdd','advert/index.htm');
insert into `t_menu` (`id`, `pid`, `name`, `icon`, `url`) values('16','11','消息模板','glyphicon glyphicon-comment','message/index.htm');
insert into `t_menu` (`id`, `pid`, `name`, `icon`, `url`) values('17','11','项目分类','glyphicon glyphicon-list','projectType/index.htm');
insert into `t_menu` (`id`, `pid`, `name`, `icon`, `url`) values('18','11','项目标签','glyphicon glyphicon-tags','tag/index.htm');
insert into `t_menu` (`id`, `pid`, `name`, `icon`, `url`) values('19','1','参数管理','glyphicon glyphicon-list-alt','param/index.htm');
```

1.2.3 关联方式

子节点通过 pid 字段关联到父节点的 id 字段，建立父子关系。



1	(NULL)	系统权限菜单
2	1	控制面板
3	1	权限管理

根节点的 pid 为 null

1.3 在 Java 类中表示树形结构

1.3.1 基本方式

在 Menu 类中使用 List<Menu> **children** 属性存储当前节点的子节点。

1.3.2 为了配合 zTree 所需要添加的属性

pid 属性：找到父节点

name 属性：作为节点名称

icon 属性：当前节点使用的图标

open 属性：控制节点是否默认打开

url 属性：点击节点时跳转的位置

1.4 按钮增删改查的规则

level 0：根节点

添加子节点

level 1：分支节点

修改

添加子节点

没有子节点：可以删除

有子节点：不能删除

level 2：叶子节点

修改

删除

2 菜单维护：页面显示树形结构

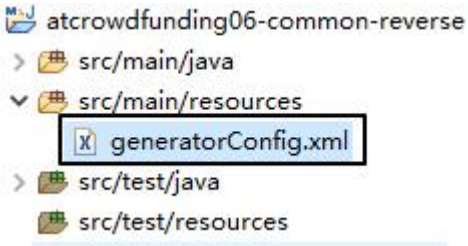
2.1 目标

将数据库中查询得到的数据到页面上显示出来。

2.2 思路

数据库查询全部→Java 对象组装→页面上使用 zTree 显示

2.3 代码：逆向工程



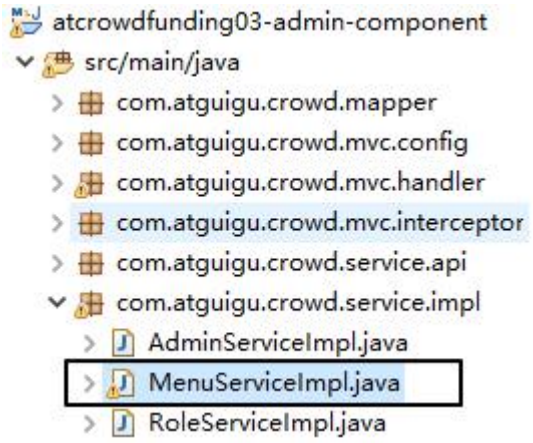

```
<table tableName="t_menu" domainObjectName="Menu" />
```

逆向生成的 Menu 实体类需要做一些调整：

```
public class Menu {

    // 主键
    private Integer id;

    // 父节点的 id
    private Integer pid;

    // 节点名称
    private String name;

    // 节点附带的 URL 地址，是将来点击菜单项时要跳转的地址
    private String url;

    // 节点图标的样式
    private String icon;

    // 存储子节点的集合，初始化是为了避免空指针异常
    private List<Menu> children = new ArrayList<>();

    // 控制节点是否默认为打开装，设置为 true 表示默认打开
    private Boolean open = true;
```

2.4 代码：将数据在 Java 代码中组装成树形结构

2.4.1 service 方法



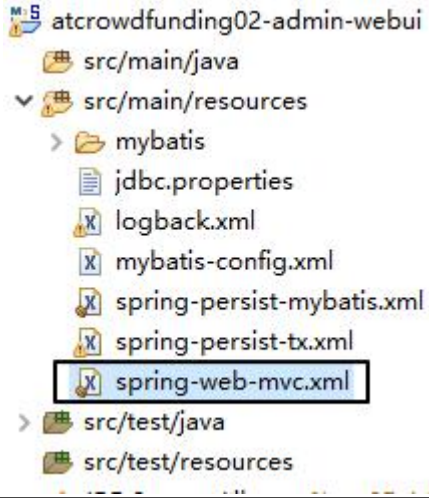

```
@Override
public List<Menu> getAll() {
    return menuMapper.selectByExample(new MenuExample());
}
```

2.4.2 handler 方法


```
@ResponseBody
@RequestMapping("/menu/get/whole/tree.json")
public ResultEntity<Menu> getWholeTreeNew() {

    // 1.查询全部的 Menu 对象
    List<Menu> menuList = menuService.getAll();

    // 2.声明一个变量用来存储找到的根节点
    Menu root = null;

    // 3.创建 Map 对象用来存储 id 和 Menu 对象的对应关系便于查找父节点
    Map<Integer, Menu> menuMap = new HashMap<>();
```

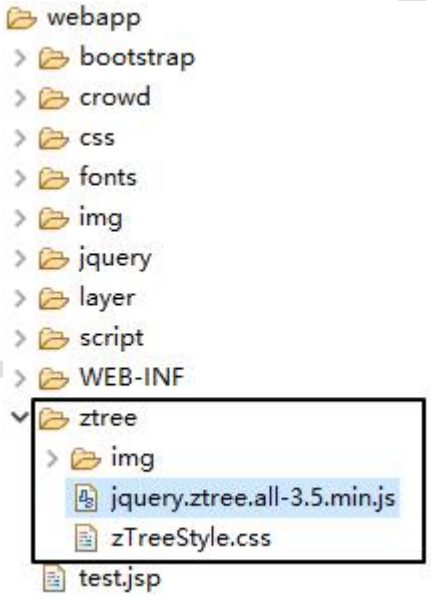
```
    // 4.遍历 menuList 填充 menuMap
    for (Menu menu : menuList) {

        Integer id = menu.getId();

        menuMap.put(id, menu);
    }

    // 5.再次遍历 menuList 查找根节点、组装父子节点
    for (Menu menu : menuList) {

        // 6.获取当前 menu 对象的 pid 属性值
        Integer pid = menu.getPid();

        // 7.如果 pid 为 null，判定为根节点
        if(pid == null) {
            root = menu;

            // 8.如果当前节点是根节点，那么肯定没有父节点，不必继续执行
            continue ;
        }

        // 9.如果 pid 不为 null，说明当前节点有父节点，那么可以根据 pid 到 menuMap 中查找对应的 Menu 对象
        Menu father = menuMap.get(pid);

        // 10.将当前节点存入父节点的 children 集合
        father.getChildren().add(menu);
    }

    // 11.经过上面的运算，根节点包含了整个树形结构，返回根节点就是返回整个树
    return ResultEntity.successWithData(root);
}
```

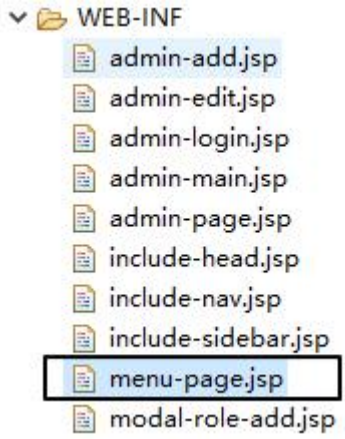
2.5 代码：跳转页面




```
<mvc:view-controller path="/menu/to/page.html" view-name="menu-page"/>
```

2.6 代码：引入 zTree 环境






```
<link rel="stylesheet" href="ztree/zTreeStyle.css"/>
<script type="text/javascript" src="ztree/jquery.ztree.all-3.5.min.js"></script>
```

2.7 代码：页面上使用 zTree 初步显示树形结构（假数据）

```
// 1.创建 JSON 对象用于存储对 zTree 所做的设置
var setting = {};

// 2.准备生成树形结构的 JSON 数据
var zNodes =[
    { name:"父节点 1 - 展开", open:true,
            ......
                    { name:"叶子节点 234"}
                ]}
        ]},
    { name:"父节点 3 - 没有子节点", isParent:true}

];

// 3.初始化树形结构
$.fn.zTree.init($("#treeDemo"), setting, zNodes);
```

2.8 代码：在页面上使用真实数据显示树形结构

```
// 1.准备生成树形结构的 JSON 数据，数据的来源是发送 Ajax 请求得到
$.ajax({
    "url": "menu/get/whole/tree.json",
    "type":"post",
    "dataType":"json",
    "success":function(response){
        var result = response.result;
```

```
            if(result == "SUCCESS") {

                // 2.创建 JSON 对象用于存储对 zTree 所做的设置
                var setting = {};

                // 3.从响应体中获取用来生成树形结构的 JSON 数据
                var zNodes = response.data;

                // 4.初始化树形结构
                $.fn.zTree.init($("#treeDemo"), setting, zNodes);
            }

            if(result == "FAILED") {
                layer.msg(response.message);
            }
        }
});
```

2.9 代码：修改默认图标为真实图标

```
// 修改默认的图标
function myAddDiyDom(treeId, treeNode) {

    // treeId 是整个树形结构附着的 ul 标签的 id
    console.log("treeId="+treeId);

    // 当前树形节点的全部的数据，包括从后端查询得到的 Menu 对象的全部属性
    console.log(treeNode);

    // zTree 生成 id 的规则
    // 例子：treeDemo_7_ico
    // 解析：ul 标签的 id_当前节点的序号_功能
    // 提示：“ul 标签的 id_当前节点的序号”部分可以通过访问 treeNode 的 tId 属性得到
    // 根据 id 的生成规则拼接出来 span 标签的 id
    var spanId = treeNode.tId + "_ico";

    // 根据控制图标的 span 标签的 id 找到这个 span 标签
    // 删除旧的 class
    // 添加新的 class
    $("#"+spanId)
        .removeClass()
        .addClass(treeNode.icon);
```

```
}
```

2.10 代码：实现“点了不跑”

```
var setting = {
    "view": {
        "addDiyDom": myAddDiyDom
    },
    "data": {
        "key": {
            "url": "maomi"
        }
    }
};
```

"maomi" ⟶ 属性名 ⟶ treeNode ⟶ 找不到 ⟶ 不跳转

```
var setting = {
    "view": {
        "addDiyDom": myAddDiyDom
    },
    "data": {
        "key": {
            "url": "maomi"
        }
    }
};
```

2.11 代码：显示按钮组

2.11.1 思路和步骤

第一步：控制A是否显示
第二步：明确具体按钮的添加规则
第三步：准备好按钮的 HTML 标签
第四步：根据按钮规则把按钮填充到 span 中

2.11.2 myRemoveHoverDom(treeId, treeNode)函数


```
// 在鼠标离开节点范围时删除按钮组
```

```
function myRemoveHoverDom(treeId, treeNode) {

    // 拼接按钮组的 id
    var btnGroupId = treeNode.tId + "_btnGrp";

    // 移除对应的元素
    $("#"+btnGroupId).remove();

}
```

2.11.3 myAddHoverDom(treeId, treeNode)函数


```
// 在鼠标移入节点范围时添加按钮组
function myAddHoverDom(treeId, treeNode) {

    // 按钮组的标签结构：<span><a><i></i></a><a><i></i></a></span>
    // 按钮组出现的位置：节点中 treeDemo_n_a 超链接的后面

    // 为了在需要移除按钮组的时候能够精确定位到按钮组所在 span，需要给 span 设置有规律的 id
    var btnGroupId = treeNode.tId + "_btnGrp";

    // 判断一下以前是否已经添加了按钮组
    if($("#"+btnGroupId).length > 0) {
        return ;
    }

    // 准备各个按钮的 HTML 标签
    var addBtn = "<a id='"+treeNode.id+"' class='btn btn-info dropdown-toggle btn-xs' style='margin-left:10px;padding-top:0px;' href='#' title='添加子节点'>  <i class='fa fa-fw fa-plus rbg '></i></a>";
    var removeBtn = "<a id='"+treeNode.id+"' class='btn btn-info dropdown-toggle btn-xs' style='margin-left:10px;padding-top:0px;' href='#' title='删除节点'>  <i class='fa fa-fw fa-times rbg '></i></a>";
    var editBtn = "<a id='"+treeNode.id+"' class='btn btn-info dropdown-toggle btn-xs' style='margin-left:10px;padding-top:0px;' href='#' title='修改节点'>  <i class='fa
```

```
fa-fw fa-edit rbg '></i></a>";

    // 获取当前节点的级别数据
    var level = treeNode.level;

    // 声明变量存储拼装好的按钮代码
    var btnHTML = "";

    // 判断当前节点的级别
    if(level == 0) {
        // 级别为 0 时是根节点，只能添加子节点
        btnHTML = addBtn;
    }

    if(level == 1) {
        // 级别为 1 时是分支节点，可以添加子节点、修改
        btnHTML = addBtn + " " + editBtn;

        // 获取当前节点的子节点数量
        var length = treeNode.children.length;

        // 如果没有子节点，可以删除
        if(length == 0) {
            btnHTML = btnHTML + " " + removeBtn;
        }

    }

    if(level == 2) {
        // 级别为 2 时是叶子节点，可以修改、删除
        btnHTML = editBtn + " " + removeBtn;
    }

    // 找到附着按钮组的超链接
    var anchorId = treeNode.tId + "_a";

    // 执行在超链接后面附加 span 元素的操作
    $("#"+anchorId).after("<span id='"+btnGroupId+"'>"+btnHTML+"</span>");

}
```

2.12 代码：把生成树形结构的代码封装到函数


```
// 生成树形结构的函数
function generateTree() {
    // 1.准备生成树形结构的 JSON 数据，数据的来源是发送 Ajax 请求得到
    $.ajax({
        "url": "menu/get/whole/tree.json",
        "type":"post",
        "dataType":"json",
        "success":function(response){
            var result = response.result;
            if(result == "SUCCESS") {

                // 2.创建 JSON 对象用于存储对 zTree 所做的设置
                var setting = {
                    "view": {
                        "addDiyDom": myAddDiyDom,
                        "addHoverDom": myAddHoverDom,
                        "removeHoverDom": myRemoveHoverDom
                    },
                    "data": {
                        "key": {
                            "url": "maomi"
                        }
                    }
                };

                // 3.从响应体中获取用来生成树形结构的 JSON 数据
                var zNodes = response.data;

                // 4.初始化树形结构
                $.fn.zTree.init($("#treeDemo"), setting, zNodes);
            }

            if(result == "FAILED") {
                layer.msg(response.message);
            }
```

```
            }
        });
}
```

menu-page.jsp 页面上调用函数即可

```
$(function(){

    // 调用专门封装好的函数初始化树形结构
    generateTree();

});
```

3 菜单维护：添加子节点

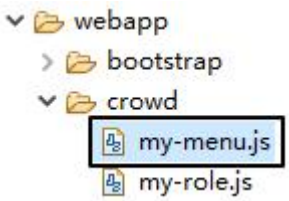
3.1 目标

给当前节点添加子节点，保存到数据库并刷新树形结构的显示。

3.2 思路

3.3 前端代码

3.3.1 给“+”按钮添加 class 值

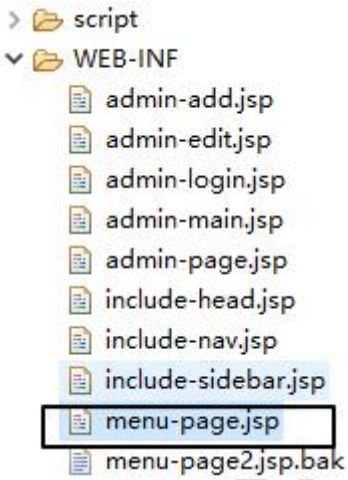



```
var addBtn = "<a id='"+treeNode.id+"' class='addBtn btn btn-info……
```

顺便把另外两个按钮的 class 也加上！

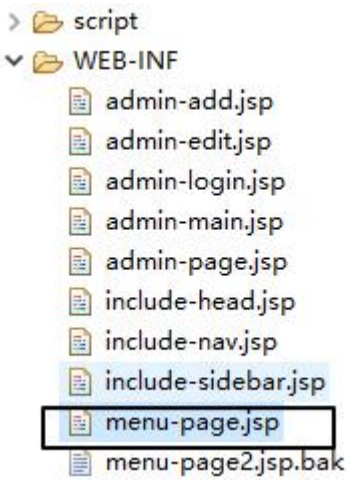
```
var removeBtn = "<a id='"+treeNode.id+"' class='removeBtn btn btn-info……
var editBtn = "<a id='"+treeNode.id+"' class='editBtn btn btn-info……
```

3.3.2 给“+”按钮绑定单击响应函数




```
// 给添加子节点按钮绑定单击响应函数
$("#treeDemo").on("click",".addBtn",function(){

    // 将当前节点的 id，作为新节点的 pid 保存到全局变量
    window.pid = this.id;

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

    return false;
});
```

3.3.3 给添加子节点的模态框中的保存按钮绑定单击响应函数




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

    // 收集表单项中用户输入的数据
    var name = $.trim($("#menuAddModal [name=name]").val());
    var url = $.trim($("#menuAddModal [name=url]").val());

    // 单选按钮要定位到“被选中”的那一个
    var icon = $("#menuAddModal [name=icon]:checked").val();

    // 发送 Ajax 请求
    $.ajax({
        "url":"menu/save.json",
        "type":"post",
        "data":{
            "pid": window.pid,
            "name":name,
            "url":url,
            "icon":icon
        },
        "dataType":"json",
        "success":function(response){
            var result = response.result;

            if(result == "SUCCESS") {
                layer.msg("操作成功！");

                // 重新加载树形结构，注意：要在确认服务器端完成保存操作后再刷新
                // 否则有可能刷新不到最新的数据，因为这里是异步的
                generateTree();
```

```
                }

                if(result == "FAILED") {
                    layer.msg("操作失败！"+response.message);
                }
            },
            "error":function(response){
                layer.msg(response.status+" "+response.statusText);
            }
        });

        // 关闭模态框
        $("#menuAddModal").modal("hide");

        // 清空表单
        // jQuery 对象调用 click()函数，里面不传任何参数，相当于用户点击了一下
        $("#menuResetBtn").click();
});
```

3.4 后端代码

```
// handler 代码
@ResponseBody
@RequestMapping("/menu/save.json")
public ResultEntity<String> saveMenu(Menu menu) {

    // Thread.sleep(2000);

    menuService.saveMenu(menu);

    return ResultEntity.successWithoutData();
}
```

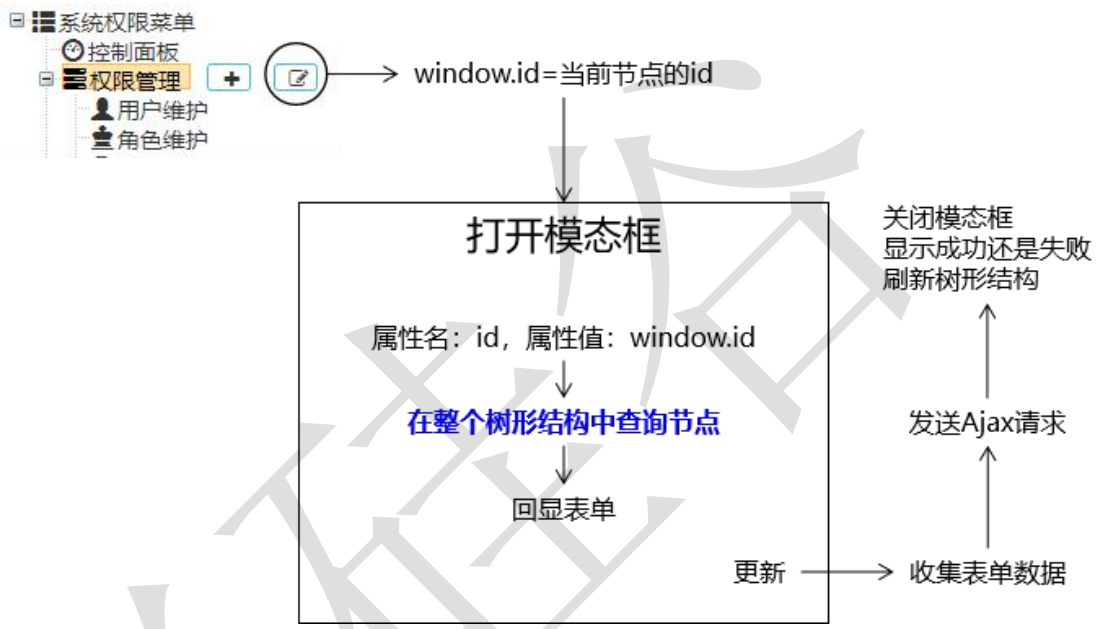
```
// service 代码
    @Override
    public void saveMenu(Menu menu) {
        menuMapper.insert(menu);
    }
```

4 菜单维护：更新节点

4.1 目标

修改当前节点的基本属性。不更换父节点。

4.2 思路



4.3 前端代码

4.3.1 给“☑”按钮绑定单击响应函数

```
// 给编辑按钮绑定单击响应函数
$("#treeDemo").on("click",".editBtn",function(){

    // 将当前节点的 id 保存到全局变量
    window.id = this.id;

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

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

    // 根据 id 属性查询节点对象
    // 用来搜索节点的属性名
```

```
        var key = "id";

        // 用来搜索节点的属性值
        var value = window.id;

        var currentNode = zTreeObj.getNodeByParam(key, value);

        // 回显表单数据
        $("#menuEditModal [name=name]").val(currentNode.name);
        $("#menuEditModal [name=url]").val(currentNode.url);

        // 回显 radio 可以这样理解：被选中的 radio 的 value 属性可以组成一个数组，
        // 然后再用这个数组设置回 radio，就能够把对应的值选中
        $("#menuEditModal [name=icon]").val([currentNode.icon]);

        return false;
});
```

4.3.2 给更新节点的模态框中的更新按钮绑定单击响应函数

```
// 给更新模态框中的更新按钮绑定单击响应函数
$("#menuEditBtn").click(function(){

        // 收集表单数据
        var name = $("#menuEditModal [name=name]").val();
        var url = $("#menuEditModal [name=url]").val();
        var icon = $("#menuEditModal [name=icon]:checked").val();

        // 发送 Ajax 请求
        $.ajax({
                "url":"menu/update.json",
                "type":"post",
                "data":{
                        "id": window.id,
                        "name":name,
                        "url":url,
                        "icon":icon
                },
                "dataType":"json",
                "success":function(response){
                        var result = response.result;

                        if(result == "SUCCESS") {
```

```
                    layer.msg("操作成功！");

                    // 重新加载树形结构，注意：要在确认服务器端完成保存操作后再刷新
                    // 否则有可能刷新不到最新的数据，因为这里是异步的
                    generateTree();
                }

                if(result == "FAILED") {
                    layer.msg("操作失败！"+response.message);
                }
            },
            "error":function(response){
                layer.msg(response.status+" "+response.statusText);
            }
        });

        // 关闭模态框
        $("#menuEditModal").modal("hide");

});
```

4.4 后端代码

```
// handler 代码
@ResponseBody
@RequestMapping("/menu/update.json")
public ResultEntity<String> updateMenu(Menu menu) {

    menuService.updateMenu(menu);

    return ResultEntity.successWithoutData();
}
```

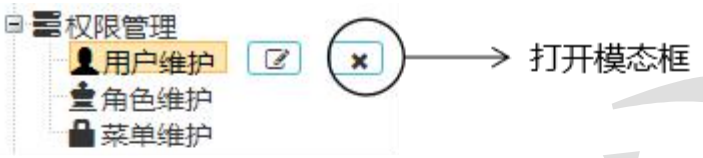
```
// service 代码
    @Override
    public void updateMenu(Menu menu) {

        // 由于 pid 没有传入，一定要使用有选择的更新，保证“pid”字段不会被置空
        menuMapper.updateByPrimaryKeySelective(menu);
    }
```

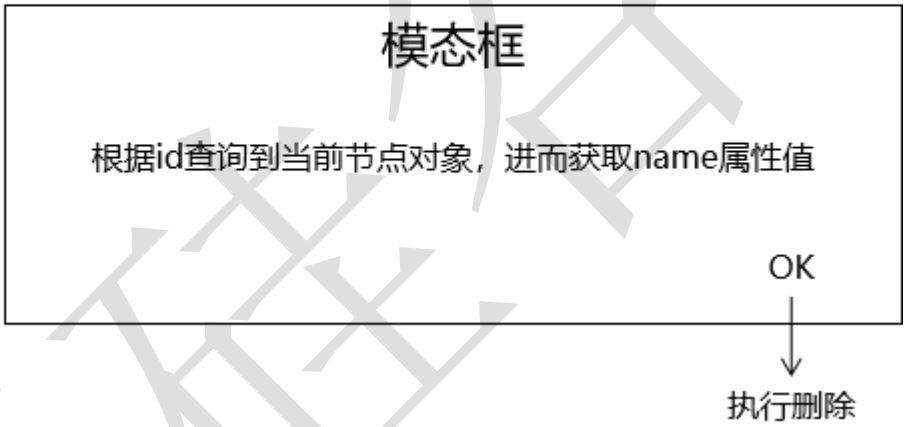
5 菜单维护：删除节点

5.1 目标

删除当前节点

5.2 思路





5.3 前端代码

5.3.1 给“×”按钮绑定单击响应函数

```
// 给“×”按钮绑定单击响应函数
$("#treeDemo").on("click",".removeBtn",function(){

    // 将当前节点的 id 保存到全局变量
    window.id = this.id;

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

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

    // 根据 id 属性查询节点对象
    // 用来搜索节点的属性名
```

```
        var key = "id";

        // 用来搜索节点的属性值
        var value = window.id;

        var currentNode = zTreeObj.getNodeByParam(key, value);

        $("#removeNodeSpan").html("【 <i class='"+currentNode.icon+"'></i>"+currentNode.name+"】");

        return false;
});
```

5.3.2 给确认模态框中的 OK 按钮绑定单击响应函数

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

        $.ajax({
                "url":"menu/remove.json",
                "type":"post",
                "data":{
                        "id":window.id
                },
                "dataType":"json",
                "success":function(response){
                        var result = response.result;

                        if(result == "SUCCESS") {
                                layer.msg("操作成功！");

                                // 重新加载树形结构，注意：要在确认服务器端完成保存操作后再刷新
                                // 否则有可能刷新不到最新的数据，因为这里是异步的
                                generateTree();
                        }

                        if(result == "FAILED") {
                                layer.msg("操作失败！"+response.message);
                        }
                },
                "error":function(response){
                        layer.msg(response.status+" "+response.statusText);
                }
```

```
        });

        // 关闭模态框
        $("#menuConfirmModal").modal("hide");
});
```

5.3.3 后端代码

```
// handler 代码
@ResponseBody
@RequestMapping("/menu/remove.json")
public ResultEntity<String> removeMenu(@RequestParam("id") Integer id) {

        menuService.removeMenu(id);

        return ResultEntity.successWithoutData();
}
```

```
// service 代码
        @Override
        public void removeMenu(Integer id) {
                menuMapper.deleteByPrimaryKey(id);
        }
```

5.3.4