# Extjs Tree Panel

1. Tree设置自连接的Entity对象。（自己对自己的一对多和多对一）

@Entity

@Table(name="t\_company")

**public** **class** Company

{

**private** Long id;

**private** String companyName;

//其他属性

**private** Company parentNode;//多个子节点 对 一个父节点

**private** List<Company> childNodes = **new** ArrayList<Company>(); //一个节点对多个子节点

@ManyToOne(cascade=CascadeType.***ALL***)

**public** Company getParentNode() {

**return** parentNode;

}

@OneToMany(mappedBy="parentNode" ,cascade=CascadeType.***ALL***)

**public** List<Company> getChildNodes() {

**return** childNodes;

}

//getter & setter

｝

1. 编写Dao层实现：查询根节点和子节点的方法。
   1. 根据父节点的id是否为null判断（根节点和子节点）。
   2. 注意HQL传参数为null被过滤掉！必须自己编写查询根节点的方法。

@Repository

**public** **interface** ICompanyDao **extends** CrudRepository<Company, Long>{

/\*\*

\* 查询根节点

\*/

@Query("from Company c where c.parentNode.id = null")

**public** List<Company> findParentNodes();

/\*\*

\* 根据父节点ID查询出子节点, 父节点为null的时候被data jpa过滤掉。所以无法传null

\*/

@Query("from Company c where c.parentNode.id = ?1")

**public** List<Company> findChildNodes(Long parentId);//null

}

1. 编写Service，把Entity封装为Extjs的TreeStore（TreeModel固定）的数据格式。
   1. TreeNode属性：<http://blog.csdn.net/cl05300629/article/details/20466111>

**public** **class** TreeNode

{

**private** Long id;

**private** String text;//显示的节点文本

**private** **boolean** expanded = **false**;//节点是否展开

**private** **boolean** leaf = **false**; //是否为子节点，根据当前节点是否有子节点判断

//private List<TreeNode> children = ArrayList<TreeNode>(); //子节点集合

//private String iconCls; //节点图标样式

//getter & setter

}

* 1. Sevice接口方法：返回封装好的TreeNode

**public** List<TreeNode> findNodes(Long parentId);

@Override

**public** List<TreeNode> findNodes(Long parentId)

{

List<TreeNode> nodeList = **new** ArrayList<TreeNode>();

List<Company> lists;

**if**(parentId==**null**) {

lists = companyDao.findParentNodes();

}**else** {

lists = companyDao.findChildNodes(parentId);

}

**for**(Company c : lists) {

TreeNode node = **new** TreeNode();

node.setId(c.getId());

node.setText(c.getCompanyName());

**if**(c.getChildNodes()!=**null**) {

**if**(c.getChildNodes().size()>0) {

node.setLeaf(**false**);//设置为父节点

}**else** {

node.setLeaf(**true**);//设置为子节点

}

}

nodeList.add(node);

}

**return** nodeList;

}

1. 编写测试用例
   1. 初始化测试数据（3层以上层次关系Tree）
   2. 测试查询根节点和子节点
   3. 测试删除一个节点
   4. 测试更新和保存一个新的节点
2. 编写Web层与前端交互（Web的方法根据浏览器的请求参数来设置参数列表）, node为TreePanel的默认属性。根节点:root,子节点：id值

@RequestMapping("/findNodes")

**public** @ResponseBody List<TreeNode> findNodesByParentId(@RequestParam("node") String node)

{

**if**(node.equals("root")) {

**return** companyService.findNodes(**null**);

}**else** {

**return** companyService.findNodes(Long.*parseLong*(node));

}

}

1. 编写前端部分
   1. ModelPanel模块面板

admin-dashboard\classic\src\view\order\Company.js

* 1. TreePanel嵌入到ModelPanel模块面板

admin-dashboard\classic\src\view\order\CompanyTree.js

Ext.define('Admin.view.order.CompanyTree', { //1.修改文件路径

extend: 'Ext.tree.Panel', //2.继承的组件类型

id:'companyTree',

xtype: 'companyTree',

title:'<b>组织架构</b>',

bind:'{companyLists}'//viewModel

});

* 1. TreeStore嵌入到ViewModel

1.新建admin-dashboard\app\store\order\CompanyStore.js

Ext.define('Admin.store.order.CompanyStore', {

extend: 'Ext.data.TreeStore',//TreePanel使用TreeStore

alias: 'store.companyStore',

proxy: {

type: 'ajax',

url: '**company/findNodes.json**', //后台Controller中的接口url地址

reader: {

type:'json'

}

},

root : {

text : '组织架构',

expanded : true //发送node=root

}

});

2.新建admin-dashboard\classic\src\view\order\CompanyViewModel.js

Ext.define('Admin.view.order.CompanyViewModel', {

extend: 'Ext.app.ViewModel',

alias: 'viewmodel.companyViewModel',

stores: {

companyLists: {

type: 'companyStore',

autoLoad: true

}

}

});

把CompanyViewModel绑定到模块主视图admin-dashboard\classic\src\view\order\Company.js

把CompanyViewModel的companyLists绑定到CompanyTree视图admin-dashboard\classic\src\view\order\CompanyTree.js