## 规格以模板管理

### 前端分层开发

#### 需求分析

前面的我们使用的html中写js代码，这样虽然写的时候可能感觉比较简单但是却不利于后期的维护，尤其是我们向要复用代码的时候，及其不便。于是我们***借鉴mvc模式***，结构，将js的***app***层 ***controller***层 ***service***层分离出来，这样提高了代码的可维护性。

#### 自定义服务

在 AngularJS 中，服务是一个函数或对象，可在你的 AngularJS 应用中使用。我们在上

次课中使用了**内置服务$http** .其实我们也可以自己来定义服务，而服务会封装一些操作。我

们在不同的控制器中可以调用同一个服务，这样服务的代码将会被重用。

我们现在就修改一下我们的品牌管理代码，使用自定义服务。

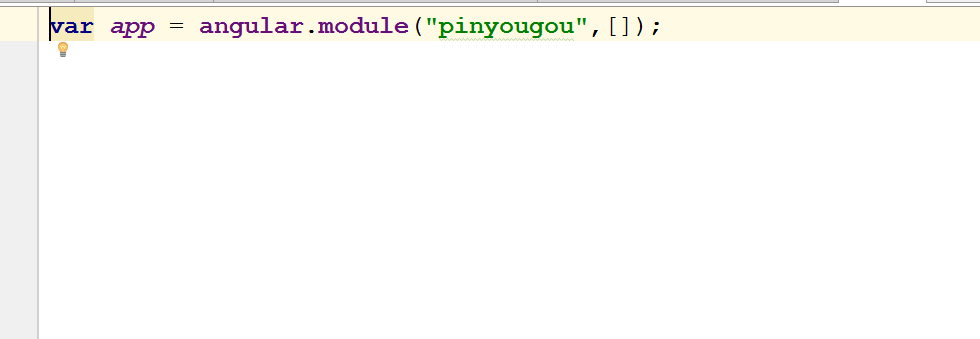
#### 1.3 代码分离：

应用层

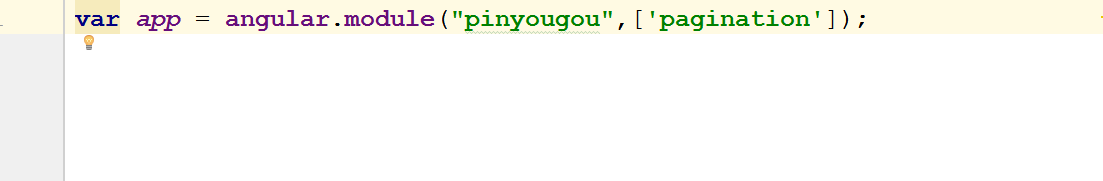


代码如下：

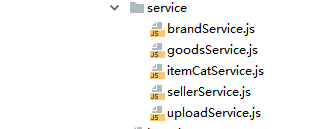
没有分页功能的应用：



带有分页功能的应用：



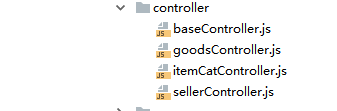
服务层



代码如下：（以品牌管理为例）

***app***.**service**(**"brandService"**, **function** ($http) {  
 *//查询所有* **this**.findAll=()=>{  
 **return** $http.get(**"../brand/list.do"**);  
 }  
  
 *//定义一个带条件的分页查询  
 //传入的参数表示 当前处于第几页 每页的数据条数 以及查询条件组成的对象* **this**.search = (page, pageSize, entity) => {  
 **return** $http.**post**(**"../brand/search.do?page="** + page + **"&pageSize="** + pageSize, entity);  
 }  
  
 *//执行保存功能* **this**.save = (url, entity) => {  
 **return** $http.**post**(url, entity);  
 }  
  
 *//执行删除功能* **this**.dele = (ids) => {  
 **return** $http.get(**"../brand/delete.do?ids="** + ids);  
 }  
  
 *//查询品牌列表 List<Map>* **this**.selectBrandList=()=>{  
 **return** $http.get(**"../brand/selectBrandList.do"**);  
 }  
  
})

控制层



代码如下：（以品牌控制层为例）

***app***.controller(**"brandController"**, **function** ($scope, $controller, brandService) {  
  
  
 *//1)让品牌控制器继承baseController* $controller(**"baseController"**, {**$scope**: $scope})  
 *//查找所有* $scope.findAll=()=>{  
 brandService.findAll().success(response=>{  
 $scope.**entity**=response;  
 })  
 }  
  
  
 *//2)进行分页查询（带条件） 继承得函数进行定义* $scope.search = () => {  
 *//获取当前的页数* **let** page = $scope.**paginationConf**.**currentPage**;  
 *//获取当前页面的数据条数* **let** pageSize = $scope.**paginationConf**.**itemsPerPage**;  
 *//调用 在服务里面定义好的服务进行查询* brandService.search(page, pageSize, $scope.**searchEntity**).success(response => {  
 *//查询成功得到PageResult对象 根据这个对象查找他的rows 属性就是分好页的数据 AngularJS绑定到页面* $scope.**brandList** = response.rows;  
 *//查询完成更新总的记录数 让原先的10 更新为 查询到的总记录数* $scope.**paginationConf**.**totalItems** = response.**total**;  
 })  
 }  
  
 *//实现新增功能或者修改功能* $scope.save = () => {  
 *//获得传来的entity的id* **let** id = $scope.**entity**.**id**;  
 *//初始化要跳转的url* **let** url = **"../brand/add.do"**;  
 **if** (id) {  
 url = **"../brand/update.do"**;  
 }  
 *//调用服务来执行这个函数* brandService.save(url, $scope.**entity**).success(response => {  
 **if** (response.**success**) {  
 *alert*(response.**message**);  
 } **else** {  
 *alert*(response.**message**);  
 }  
 })  
 }  
  
 *//点击修改刷新模态框中的值* $scope.updateUI = (brand) => {  
 $scope.**entity** = brand;  
 }  
  
 *//实现删除功能* $scope.dele = () => {  
 *//交由服务去执行删除功能* brandService.dele($scope.**selectdeIds**).success(response => {  
 **if** (response.**success**) { *//如果响应的结果为 不成功  
 alert*(response.**message**); *//提示删除失败* $scope.search(); *//刷新页面* } **else** {  
 *alert*(response.**message**); *//提示删除成功* }  
 })  
 }  
})

页面

引入我们写的js代码：

*<!--3.加载基本的模块-->*<**script src="../js/base\_pagination.js"**></**script**>  
*<!--4.引入baseController-->*<**script src="../js/controller/baseController.js"**></**script**>  
*<!--5.引入其它的service及控制器-->*<**script src="../js/service/brandService.js"**></**script**>  
<**script src="../js/controller/brandController.js"**></**script**>

### 控制器的继承

#### 2.1 需求分析 分页功能的复用

有些功能是每个页面都有可能用到的，比如分页，复选等等，如果我们再开发另一个功能，

还需要重复编写。怎么能让这些通用的功能只写一次呢？我们通过继承的方式来实现。

代码如下：

父类控制器

*//1.定义一个公共的控制器（父控制器）****app***.controller(**"baseController"**, **function** ($scope) {  
  
 *//实现分页控制* $scope.**paginationConf** = {  
 **currentPage**: 1, *//当前第几页 默认第一页* **totalItems**: 10, *//总数据条数* **itemsPerPage**: 10, *//每页数据条数* **perPageOptions**: [10,20,30], *//可以选择的将数据分成每页多少条* onChange: **function** () { *//每次窗口加载完了 调用的方法* $scope.search();  
 }  
 };  
  
 *//实现批量选择  
 //定义一个数组用来存放 复选框中被选择的 品牌id* $scope.**selectdeIds**=[];  
 *//用户点击复选框的时候 调用这个函数* $scope.updateSelection=(event,id)=>{  
 **if**(event.**target**.**checked**){ *//表示事件的结果是选中* $scope.**selectdeIds**.push(id);  
 }**else** {  
 *//得到取消选中的商品对应的id* **let** notIn=$scope.**selectdeIds**.indexOf(id);  
 *//将取消选商品的id 从数组中删除* $scope.**selectdeIds**.splice(notIn,1);  
 }  
 };  
});

子类控制器

让另外一个需要用他的控制器去继承它：

***app***.controller(**"brandController"**, **function** ($scope, $controller, brandService) {  
  
  
 *//1)让品牌控制器继承baseController* $controller(**"baseController"**, {**$scope**: $scope})  
 *//查找所有* $scope.findAll=()=>{  
 brandService.findAll().success(response=>{  
 $scope.**entity**=response;  
 })  
 }

$controller 也是 angular 提供的一个服务，可以实现伪继承，实际上就是与 BaseController 共

享$scope

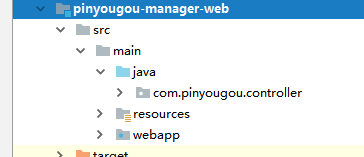
错误提示：



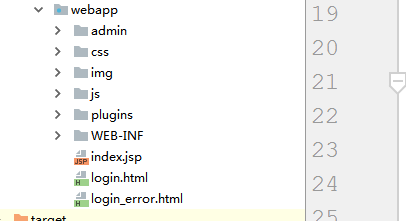
### 规格管理

#### 3.1 准备工作：

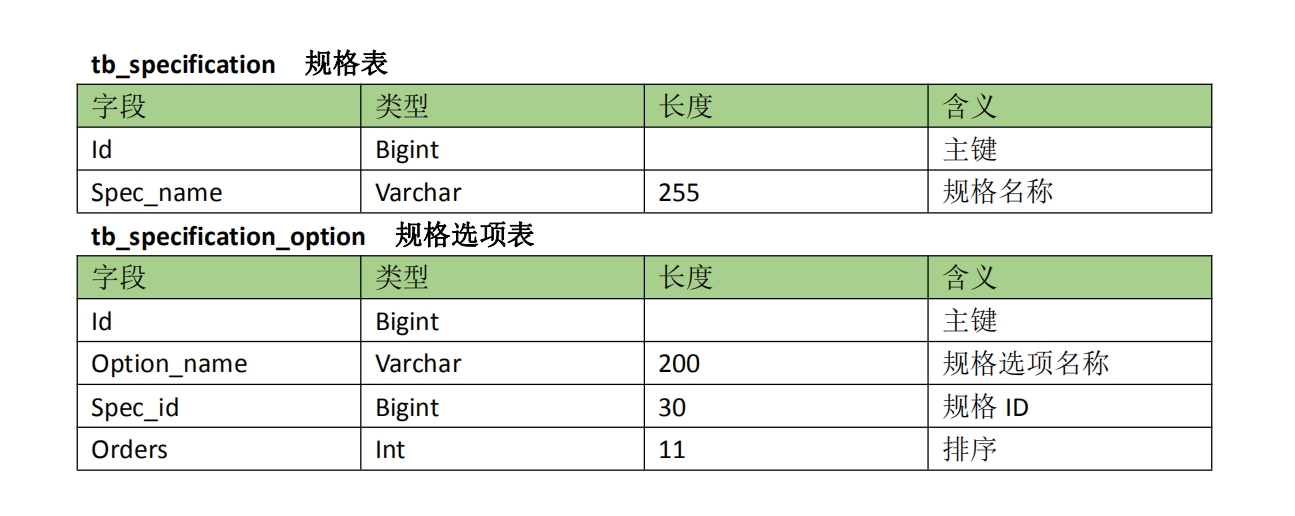
* 搭建好项目需要的包



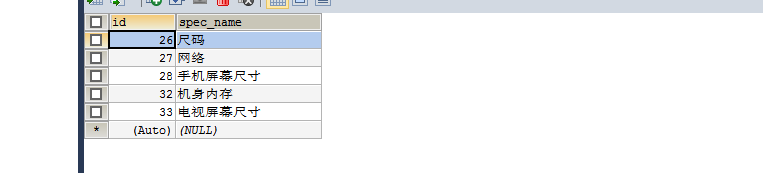
* 导入相关的静态资源前端页面（没有插入数据的）



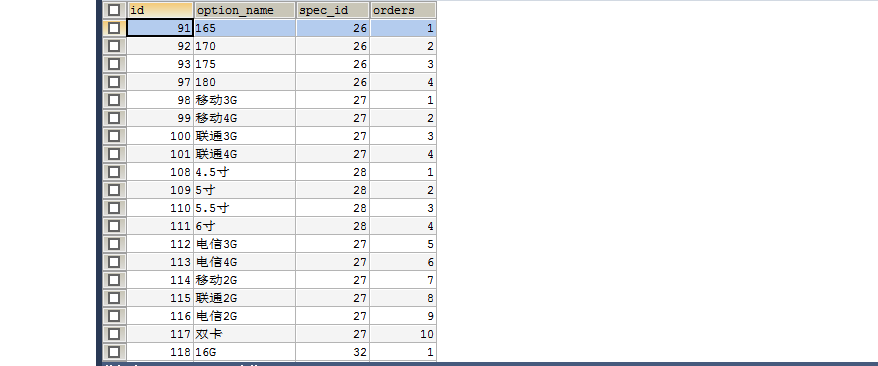
* 数据库的表字段的设计



规格表：



规格选项表：



#### 3.2 引入对应js代码

*<!--3.加载基本的模块-->*<**script src="../js/base.js"**></**script**>  
*<!--4.引入baseController-->*<**script src="../js/controller/baseController.js"**></**script**>  
*<!--5.引入其它的service及控制器-->*<**script src="../js/service/specificationService.js"**></**script**>  
<**script src="../js/controller/specificationController.js"**></**script**>

#### 3.3 放置分页组件



#### 3.4 指令与表达式

在body标签中定义app与controller：

<**body class="hold-transition skin-red sidebar-mini" ng-init="findAll()"  
 ng-app="pinyougou" ng-controller="specificationController"**>

列表的循环：

<**tr ng-repeat="spec in list"**>

这里的list 是后台返回的值（findAll方法传来的）

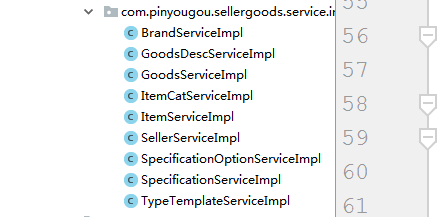
<**tr ng-repeat="spec in list"**>  
 <**td**><**input type="checkbox" ng-click="updateSelection($event,spec.id)"**></**td**>  
 <**td**>{{**spec**.**id**}}</**td**>  
 <**td**>{{**spec**.specName}}</**td**>  
 <**td class="text-center"**>  
 <**button type="button" ng-click="findOne(spec.id)" class="btn bg-olive btn-xs" data-toggle="modal"  
 data-target="#editModal"**>修改</**button**>  
 </**td**>  
</**tr**>

#### 3.5 对规格的CRUD

**Js代码：**

*//控制层****app***.controller(**'specificationController'** ,**function**($scope,$controller,specificationService){  
   
 $controller(**'baseController'**,{**$scope**:$scope});*//继承  
   
 //读取列表数据绑定到表单中* $scope.findAll=**function**(){  
 specificationService.findAll().success(  
 **function**(response){  
 $scope.**list**=response;  
 }   
 );  
 }   
  
 *//新增或者修改组合类 规格选项类(点击保存)* $scope.save=()=> {  
 *//预先设置修改的url* **let** url=**"../specification/add.do"**;  
 *//获得页面上修改按钮对应的id* **let** id=$scope.**entity**.**spec**.**id**;  
 *//判断如果id存在就是修改不存在就是添加* **if** (id){  
 url=**"../specification/update.do"**;  
 }  
 *//调用服务执行保存操作* specificationService.save(url,$scope.**entity**).success(response=>{  
 **if** (response.**success**){ *//如果修改成功  
 //弹出提示  
 alert*(response.**message**);  
 *//刷新页面* $scope.findAll();  
 }**else** {  
 *//弹出提示  
 alert*(response.**message**);  
 }  
 })  
 }  
  
 *//点击新增规格选项  
 //初始化规格组合类对象* $scope.**entity**={**spec**:{},**specificationOptionList**:[]}  
 *//新增规格选项* $scope.addSpecOptions=()=>{  
 *//向页面上的模态框中添加一条空的值 使其有输入框 供用户输入* $scope.**entity**.**specificationOptionList**.push({});  
 }  
  
 *//点击修改按钮触发* $scope.findOne=(id)=>{  
 specificationService.findOne(id).success(response=>{  
 $scope.**entity**=response;  
 })  
 }  
  
 *//批量删除* $scope.dele=()=>{  
 *//让服务区执行删除的方法* specificationService.dele($scope.**selectdeIds**).success((response)=>{  
 **if** (response.**success**){  
 *//删除成功就刷新* $scope.findAll();  
 *//清空批量的数组* $scope.**selectdeIds**=[];  
 }  
 })  
 }  
});

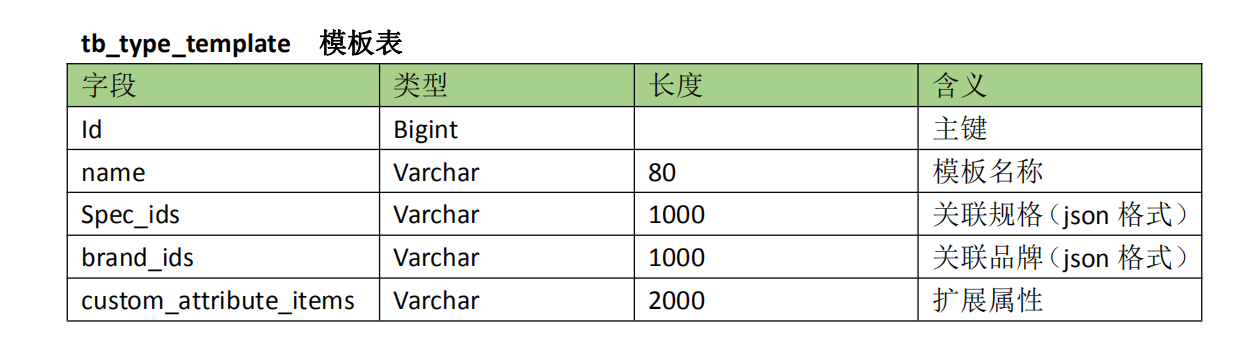
**实现类：**



*/\*\*  
 \* 规格选项 服务实现层  
 \*  
 \** ***@author*** *Administrator  
 \*/*@Service  
**public class** SpecificationServiceImpl **implements** SpecificationService {  
  
 @Autowired  
 **private** TbSpecificationMapper **specificationMapper**;  
 @Autowired  
 **private** TbSpecificationOptionMapper **specificationOptionMapper**;  
  
 */\*\*  
 \* 查询全部  
 \*/* @Override  
 **public** List<TbSpecification> findAll() {  
 **return specificationMapper**.selectByExample(**null**);  
 }  
  
 */\*\*  
 \* 按分页查询  
 \*/* @Override  
 **public** PageResult findPage(**int** pageNum, **int** pageSize) {  
 PageHelper.*startPage*(pageNum, pageSize);  
 Page<TbSpecification> page = (Page<TbSpecification>) **specificationMapper**.selectByExample(**null**);  
 **return new** PageResult(page.getTotal(), page.getResult());  
 }  
  
 */\*\*  
 \* 增加  
 \*/* @Override  
 **public void** add(Specification specification) {  
 *//先在规格表中插入 规格* **specificationMapper**.insert(specification.getSpec());  
 System.***out***.println(**"321321321"**);  
 System.***out***.println(specification.getSpec().getId());  
 System.***out***.println(specification.getSpecificationOptionList());  
 System.***out***.println(**"24323423423423434234"**);  
 *// 执行完上面的操作后，下面的规格id才会有值  
 //前提是在TbSpecificationMapper.xml的insert方法中要有如下配置：  
// <selectKey resultType="java.lang.Long" order="AFTER" keyProperty="id">  
// SELECT LAST\_INSERT\_ID() AS id  
// </selectKey>  
 //插入规格选项 首先获得规格的id 用来给规格选项的外键* Long id = specification.getSpec().getId();  
 *//遍历规格选项 给其外键赋值* List<TbSpecificationOption> optionList = specification.getSpecificationOptionList();  
 **for** (TbSpecificationOption tbSpecificationOption : optionList) {  
 *//设置规格选项的外键* tbSpecificationOption.setSpecId(id);  
 System.***out***.println(tbSpecificationOption.getOptionName());  
 *//添加到表中* **specificationOptionMapper**.insert(tbSpecificationOption);  
 }  
 }  
  
  
 */\*\*  
 \* 修改  
 \*/* @Override  
 **public void** update(TbSpecification specification) {  
 **specificationMapper**.updateByPrimaryKey(specification);  
 }  
  
 */\*\*  
 \* 根据ID获取实体  
 \*  
 \** ***@param id*** *\** ***@return*** *\*/* @Override  
 **public** Specification findOne(Long id) {  
 *//查询规格对象* TbSpecification specification = **specificationMapper**.selectByPrimaryKey(id);  
 *//根据规格对象查找对应的规格选项* TbSpecificationOptionExample exanmple=**new** TbSpecificationOptionExample();  
 TbSpecificationOptionExample.Criteria criteria = exanmple.createCriteria();  
 criteria.andSpecIdEqualTo(id);  
 List<TbSpecificationOption> tbSpecificationOptions = **specificationOptionMapper**.selectByExample(exanmple);  
 *//返回的带有规格选项的组合对象* Specification specification1 = **new** Specification();  
 specification1.setSpec(specification);  
 specification1.setSpecificationOptionList(tbSpecificationOptions);  
 **return** specification1;  
 }  
  
 */\*\*  
 \* 批量删除  
 \*/* @Override  
 **public void** delete(Long[] ids) {  
 **for** (Long id : ids) {  
 TbSpecificationOptionExample example = **new** TbSpecificationOptionExample();  
 TbSpecificationOptionExample.Criteria criteria = example.createCriteria();  
 criteria.andSpecIdEqualTo(id);  
 **specificationOptionMapper**.deleteByExample(example);  
 **specificationMapper**.deleteByPrimaryKey(id);  
 }  
 }  
  
  
 @Override  
 **public** PageResult findPage(TbSpecification specification, **int** pageNum, **int** pageSize) {  
 PageHelper.*startPage*(pageNum, pageSize);  
  
 TbSpecificationExample example = **new** TbSpecificationExample();  
 Criteria criteria = example.createCriteria();  
  
 **if** (specification != **null**) {  
 **if** (specification.getSpecName() != **null** && specification.getSpecName().length() > 0) {  
 criteria.andSpecNameLike(**"%"** + specification.getSpecName() + **"%"**);  
 }  
  
 }  
  
 Page<TbSpecification> page = (Page<TbSpecification>) **specificationMapper**.selectByExample(example);  
 **return new** PageResult(page.getTotal(), page.getResult());  
 }  
  
 *//查询规格列表 List<Map>* @Override  
 **public** List<Map> findSpecList() {  
 **return specificationMapper**.findSpecList();  
 }  
}

### 模板管理

#### 4.1 模板的表结构



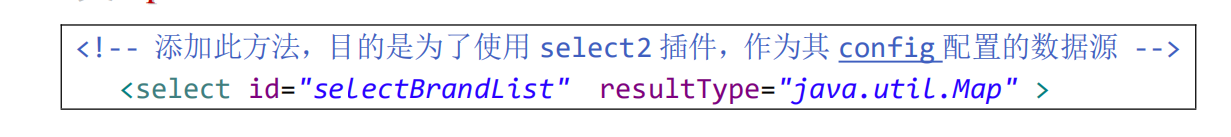
#### 4.2 模板列表

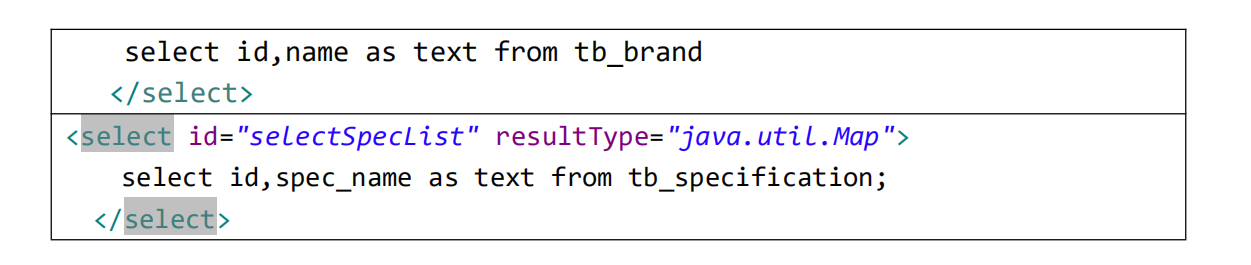
* **引入js**
* **指令表达式**

**与前面的一样**

#### 4.3 select2插件的使用

在mapper.xml中定义sql语句



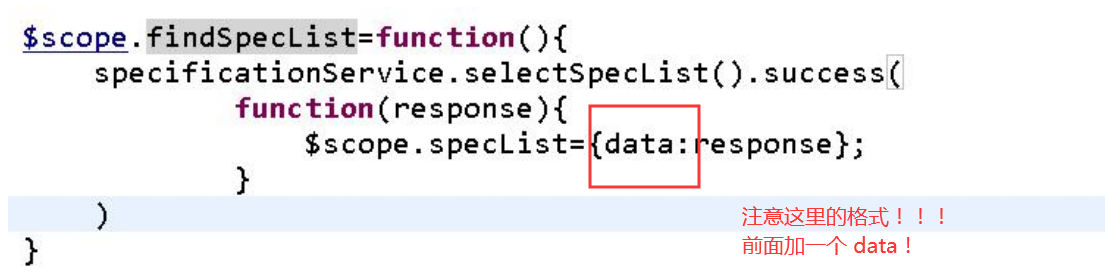




**第一步：查询品牌列表：**

**第二步：在模板控制器typeTemplateController.js中引入brandService服务**

**第三步：在模板控制器typeTemplateController.js定义方法：**



引入select2的相关插件以及js css库

*<!-- 下面引入select2插件相关的js及css库 -->*<**script src="../plugins/select2/select2.min.js" type="text/javascript"**></**script**>  
<**script type="text/javascript" src="../plugins/select2/angular-select2.js"**> </**script**>  
<**script src="../plugins/angularjs/pagination.js"**></**script**>

*<!-- 引入相关的类库 -->*<**script type="text/javascript" src="../plugins/angularjs/angular.min.js"**> </**script**>  
<**script type="text/javascript" src="../js/base\_pagination.js"**> </**script**>

#### 4.4模板的CRUD

Controller.js

*//控制层****app***.controller(**'typeTemplateController'** ,**function**($scope,$controller,brandService,specificationService,typeTemplateService){  
   
 $controller(**'baseController'**,{**$scope**:$scope});*//继承  
   
 //读取列表数据绑定到表单中* $scope.findAll=**function**(){  
 typeTemplateService.findAll().success(  
 **function**(response){  
 $scope.**list**=response;  
 }   
 );  
 }   
   
 *//分页* $scope.findPage=**function**(page,rows){   
 typeTemplateService.findPage(page,rows).success(  
 **function**(response){  
 $scope.**list**=response.rows;   
 $scope.**paginationConf**.**totalItems**=response.**total**;*//更新总记录数* }   
 );  
 }  
   
 *//查询实体* $scope.findOne=**function**(id){   
 typeTemplateService.findOne(id).success(  
 **function**(response){  
 $scope.**entity**= response;   
 }  
 );   
 }  
   
 *//保存* $scope.save=**function**(){   
 *//1.得到当前的模板id* **let** id = $scope.**entity**.**id**;  
 *//2.定义url地址* **let** url = **"../typeTemplate/add.do"**;  
 *//3.根据id是否有值，决定是添加还是修改操作* **if**(id){  
 url = **"../typeTemplate/update.do"**;  
 }  
 *//4.调用服务层* typeTemplateService.save(url,$scope.**entity**).success(response=>{  
 **if**(response.**success**){  
 $scope.findAll();  
 }**else**{  
 *alert*(response.**message**);  
 }  
 })  
 }  
   
   
 *//批量删除* $scope.dele=**function**(){   
 *//获取选中的复选框* typeTemplateService.dele( $scope.**selectIds** ).success(  
 **function**(response){  
 **if**(response.**success**){  
 $scope.findAll();*//刷新列表* $scope.**selectIds**=[];  
 }   
 }   
 );   
 }  
   
 $scope.**searchEntity**={};*//定义搜索对象   
   
 //搜索* $scope.search=**function**(page,rows){   
 typeTemplateService.search(page,rows,$scope.**searchEntity**).success(  
 **function**(response){  
 $scope.**list**=response.rows;   
 $scope.**paginationConf**.**totalItems**=response.**total**;*//更新总记录数* }   
 );  
 }  
 *//查询品牌列表* $scope.findBrandList=()=>{  
 brandService.selectBrandList().success(response=>{  
 $scope.**brandList**={**data**:response};  
 })  
 }  
 *//对模板表的相关数据进行初始化* $scope.**entity**={**customAttributeItems**:[]};  
 *//查询规格列表* $scope.findSpecList=()=>{  
 specificationService.findSpecList().success(response=>{  
 $scope.**specList**={**data**:response};  
 })  
 }  
 *//新增扩展属性* $scope.addAttributeAttr=()=>{  
 $scope.**entity**.**customAttributeItems**.push({});  
 }  
 *//删除扩展属性* $scope.removeCustomAttr=(index)=>{  
 $scope.**entity**.**customAttributeItems**.splice(index,1);  
 }  
 *//修改界面操作* $scope.updateUI=(template)=>{  
 $scope.**entity** = template;  
 *//将json字符串转换为json对象* $scope.**entity**.**specIds** = ***JSON***.parse($scope.**entity**.**specIds**);  
 $scope.**entity**.**brandIds** = ***JSON***.parse($scope.**entity**.**brandIds**);  
 $scope.**entity**.**customAttributeItems** = ***JSON***.parse($scope.**entity**.**customAttributeItems**);  
 }  
 *//处理json字符串为单个数据* $scope.jsonToString=(jsonString,key)=>{  
 *//1.将jsonString字符串转换为json对象* **let** arr = ***JSON***.parse(jsonString);  
 *//2.遍历这个json对象* **let** info = **""**;  
 **for**(**let** i = 0,len = arr.**length**; i < len;i++){  
 *//3.得到某个对象的指定的属性值* info += arr[i][key] + **","**;  
 }  
 *//4.返回最后的结果  
 //4.1)处理逗号* info = info.substr(0,info.length-1);  
 *//4.2)返回* **return** info;  
 }  
});

## 安全框架与商家入驻审核

### springSecurity框架

#### 简介

Spring Security 是一个能够为基于 Spring 的企业应用系统提供声明式的安全访问控制解

决方案的安全框架。它提供了一组可以在 Spring 应用上下文中配置的 Bean，充分利用了

Spring IoC，DI（控制反转 Inversion of Control ,DI:Dependency Injection 依赖注入）和 AOP（面

向切面编程）功能，为应用系统提供声明式的安全访问控制功能，减少了为企业系统安全控

制编写大量重复代码的工作。

### 运营商系统登录与安全控制

#### 2.1 在 pinyougou-manager-web 工程的 web.xml 文件中添加如下配置：

*<?***xml version="1.0" encoding="UTF-8"***?>*<**web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xmlns="http://java.sun.com/xml/ns/javaee"  
 xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"  
 version="2.5"**>  
  
 *<!--1.加载spring-security的配置文件-->* <**context-param**>  
 <**param-name**>contextConfigLocation</**param-name**>  
 <**param-value**>classpath\*:spring/spring-\*.xml</**param-value**>  
 </**context-param**>  
 *<!--2.加载spring容器-->* <**listener**>  
 <**listener-class**>org.springframework.web.context.ContextLoaderListener</**listener-class**>  
 </**listener**>  
 *<!--3.加载springsecurity的过滤器-->* <**filter**>  
 *<!--注意：这里的过滤器的名称必须为springSecurityFilterChain-->* <**filter-name**>springSecurityFilterChain</**filter-name**>  
 <**filter-class**>org.springframework.web.filter.DelegatingFilterProxy</**filter-class**>  
 </**filter**>  
 <**filter-mapping**>  
 <**filter-name**>springSecurityFilterChain</**filter-name**>  
 <**url-pattern**>/\*</**url-pattern**>  
 </**filter-mapping**>  
 *<!--1.加载springmvc-->* <**servlet**>  
 <**servlet-name**>springmvc</**servlet-name**>  
 <**servlet-class**>org.springframework.web.servlet.DispatcherServlet</**servlet-class**>  
 <**init-param**>  
 <**param-name**>contextConfigLocation</**param-name**>  
 <**param-value**>classpath\*:spring/springmvc.xml</**param-value**>  
 </**init-param**>  
 </**servlet**>  
 <**servlet-mapping**>  
 <**servlet-name**>springmvc</**servlet-name**>  
 <**url-pattern**>\*.do</**url-pattern**>  
 </**servlet-mapping**>  
  
 *<!--2.配置中文乱码的过滤器-->* <**filter**>  
 <**filter-name**>myFilter</**filter-name**>  
 <**filter-class**>org.springframework.web.filter.CharacterEncodingFilter</**filter-class**>  
 <**init-param**>  
 <**param-name**>encoding</**param-name**>  
 <**param-value**>UTF-8</**param-value**>  
 </**init-param**>  
 <**init-param**>  
 <**param-name**>forceEncoding</**param-name**>  
 <**param-value**>true</**param-value**>  
 </**init-param**>  
 </**filter**>  
 <**filter-mapping**>  
 <**filter-name**>myFilter</**filter-name**>  
 <**url-pattern**>/\*</**url-pattern**>  
 </**filter-mapping**>  
</**web-app**>

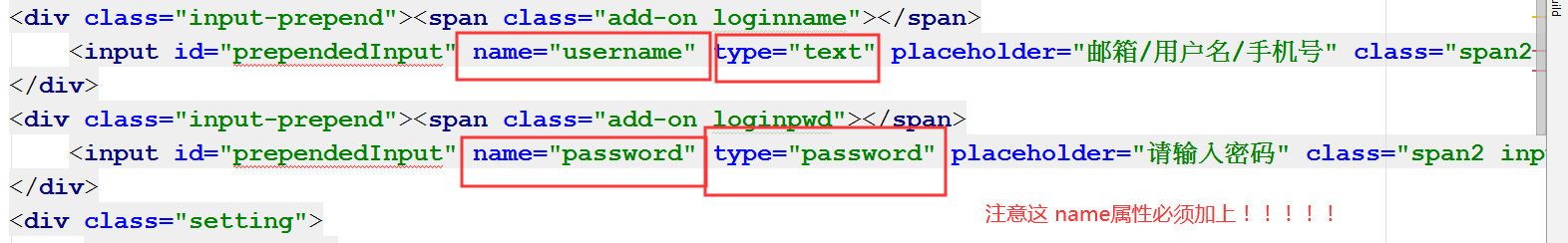
#### 2.2 在 classpath 下的 spring-security.xml 文件中添加如下配置：

*<?***xml version="1.0" encoding="UTF-8"***?>*<**beans:beans xmlns="http://www.springframework.org/schema/security"  
 xmlns:beans="http://www.springframework.org/schema/beans" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd  
 http://www.springframework.org/schema/security http://www.springframework.org/schema/security/spring-security.xsd"**>  
 *<!--1.放行指定的静态资源-->* <**http security="none" pattern="/login.html"**/>  
 <**http security="none" pattern="/css/\*\*"**/>  
 <**http security="none" pattern="/img/\*\*"**/>  
 <**http security="none" pattern="/js/\*\*"**/>  
 <**http security="none" pattern="/plugins/\*\*"**/>  
 *<!--2.定义http请求相关-->* <**http use-expressions="false"**>  
 <**intercept-url pattern="/\*\*" access="ROLE\_ADMIN"**/>  
 *<!--2.1）属性解析：-->  
 <!--login-page:代表登录页-->  
 <!--default-target-url:代表登录成功后自动跳转的页面-->  
 <!--authentication-failure-url:代表登录失败后跳转的页面-->  
 <!--always-use-default-target:不管登录前访问哪个页面，只能登录成功就跳转到目标页面-->* <**form-login login-page="/login.html" default-target-url="/admin/index.html" authentication-failure-url="/login.html"  
 always-use-default-target="true"**/>  
 *<!--2.2）如果在项目的页面中有框架，必须要设置如下内容，否则，框架不能使用-->* <**headers**>  
 <**frame-options policy="SAMEORIGIN"**/>  
 </**headers**>  
 *<!--2.3)禁用跨站请求伪造-->* <**csrf disabled="true"**/>  
 *<!--2.4)使用注销功能-->* <**logout**/>  
 </**http**>  
 *<!--3.定义认证管理器-->* <**authentication-manager**>  
 <**authentication-provider**>  
 <**user-service**>  
 <**user name="admin" authorities="ROLE\_ADMIN" password="123"**/>  
 <**user name="zhangsan" authorities="ROLE\_ADMIN" password="456"**/>  
 </**user-service**>  
 </**authentication-provider**>  
 </**authentication-manager**>  
</**beans:beans**>

#### 2.3 在mvc的控制器中定义一个登录的控制器Login.controller

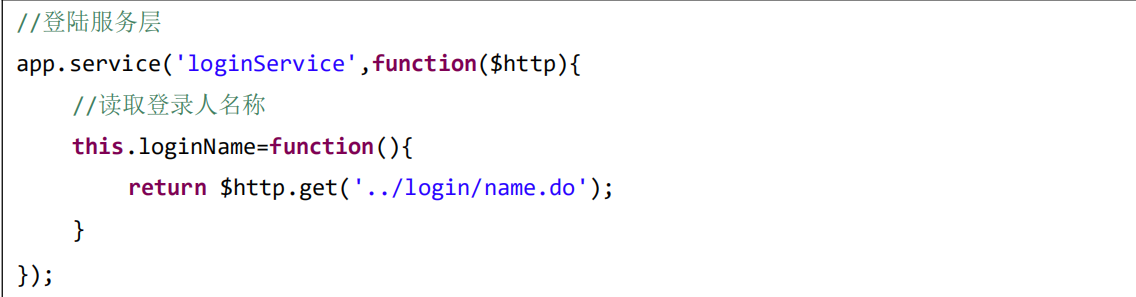
@RestController  
**public class** LoginController {  
  
 *//1.用户登录* @RequestMapping(**"login"**)  
 **public** Map login(){  
 *//1.1)得到当前登录的用户名* String name = SecurityContextHolder.*getContext*().getAuthentication().getName();  
 *//1.2)将此用户名放到一个map中* Map map = **new** HashMap();  
 map.put(**"name"**,name);  
 *//1.3)返回* **return** map;  
 }  
}

注意这里！！！！！！！！！！！！！！！！



#### 2.4 前端显示用户名

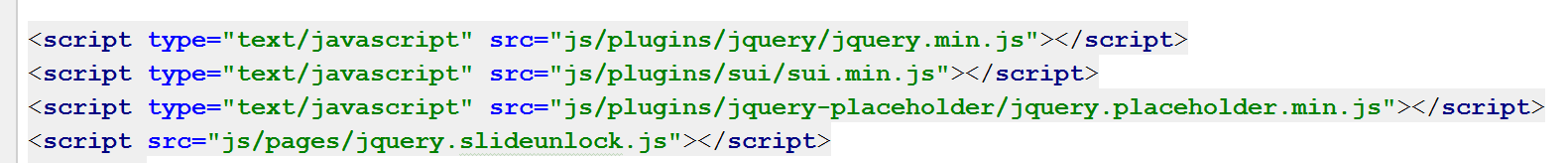
服务层



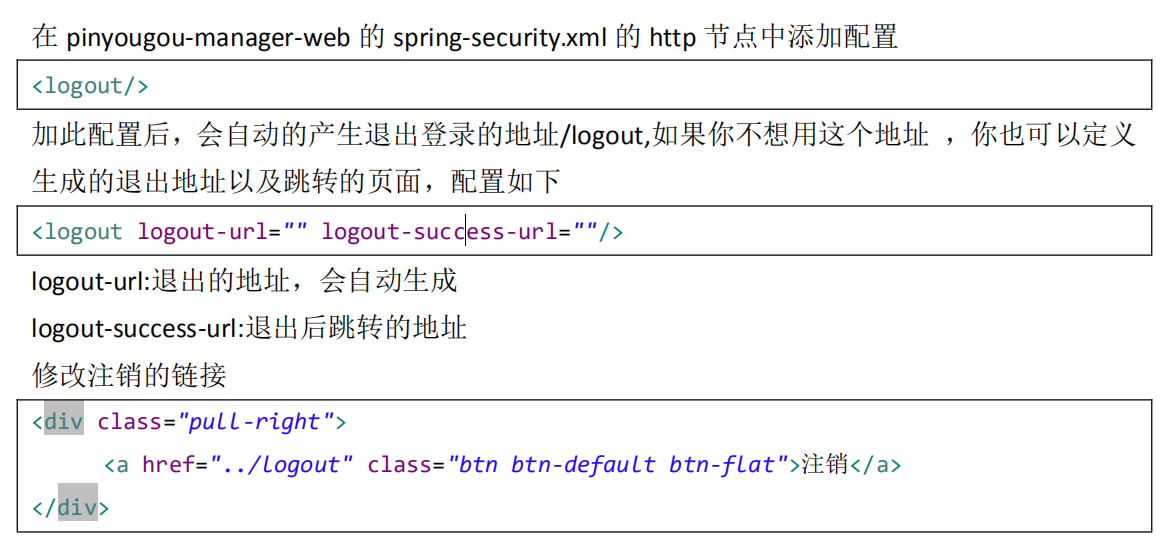
控制层



html引入



#### 2.5 退出登录



### 商家入驻

商家的数据表结构：



基本过程：

* 商家首先在注册商家页面注册然后manager-web审核 审核不通过则不能进入下一步
* 审核的过程发生在Dubbo注册的自定义的一个服务中
* 审核通过将这个商家的信息加入数据库
* 商家方可登录shop-web

### 4、Manager审核：





这个界面就是一个根据主键查询商家信息然后遍历此处省略。。。





审核就是改变 这个status值 根据状态值确定是否通过审核！！！

*//商家审核*$scope.updateStatus=(sellerId,status)=>{  
 sellerService.updateStatus(sellerId,status).success(response=>{  
 **if**(response.**success**){*//审核通过就刷新页面* $scope.findAll();  
 }**else**{  
 *alert*(response.**message**);  
 }  
 })  
}

### 5、商家的登录有安全控制

在shop-web中的xml中定义 与 manager相似的配置（略微不同）：

*<?***xml version="1.0" encoding="UTF-8"***?>*<**beans:beans xmlns="http://www.springframework.org/schema/security"  
 xmlns:beans="http://www.springframework.org/schema/beans"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xmlns:dubbo="http://code.alibabatech.com/schema/dubbo"  
 xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd  
 http://www.springframework.org/schema/security http://www.springframework.org/schema/security/spring-security.xsd http://code.alibabatech.com/schema/dubbo http://code.alibabatech.com/schema/dubbo/dubbo.xsd"**>  
 *<!--1.放行指定的静态资源-->* <**http security="none" pattern="/css/\*\*"**/>  
 <**http security="none" pattern="/img/\*\*"**/>  
 <**http security="none" pattern="/js/\*\*"**/>  
 <**http security="none" pattern="/plugins/\*\*"**/>  
 <**http security="none" pattern="/register.html"**/>  
 <**http security="none" pattern="/shoplogin.html"**/>  
 <**http security="none" pattern="/seller/add.do"** />  
 *<!--2.定义http请求相关-->* <**http use-expressions="false"**>  
 <**intercept-url pattern="/\*\*" access="ROLE\_ADMIN"**/>  
 *<!--2.1）属性解析：-->  
 <!--login-page:代表登录页-->  
 <!--default-target-url:代表登录成功后自动跳转的页面-->  
 <!--authentication-failure-url:代表登录失败后跳转的页面-->  
 <!--always-use-default-target:不管登录前访问哪个页面，只能登录成功就跳转到目标页面-->* <**form-login login-page="/shoplogin.html" default-target-url="/admin/index.html" authentication-failure-url="/shoplogin.html"  
 always-use-default-target="true"**/>  
 *<!--2.2）如果在项目的页面中有框架，必须要设置如下内容，否则，框架不能使用-->* <**headers**>  
 <**frame-options policy="SAMEORIGIN"**/>  
 </**headers**>  
 *<!--2.3)禁用跨站请求伪造-->* <**csrf disabled="true"**/>  
 *<!--2.4)使用注销功能-->* <**logout**/>  
 </**http**>  
 *<!--3.定义认证管理器-->* <**authentication-manager**>  
 <**authentication-provider user-service-ref="userService"**>  
 *<!--3.1）指定加密器，将商家的密码进行加密后再与数据库进行比对-->* <**password-encoder ref="encoder"**/>  
 </**authentication-provider**>  
 </**authentication-manager**>  
 *<!--3.2）定义密码加密器对象-->* <**beans:bean id="encoder" class="org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder"**/>  
 *<!--4.定义userService对象，用于当前的商家的角色的验证-->* <**beans:bean id="userService" class="com.pinyougou.shop.service.UserDetailServiceImpl"**>  
 <**beans:property name="sellerService" ref="sellerService"**/>  
 </**beans:bean**>  
 *<!--5.从zookeeper注册中心中取得serllerService这个服务-->* <**dubbo:application name="pinyougou-shop-web"**/>  
 <**dubbo:registry address="zookeeper://192.168.25.128:2181"**/>  
 <**dubbo:reference id="sellerService" interface="com.pinyougou.sellergoods.service.SellerService"**/>  
</**beans:beans**>

在shop-web 的web.xml中配置springSecurity的过滤器：

*<?***xml version="1.0" encoding="UTF-8"***?>*<**beans xmlns="http://www.springframework.org/schema/beans"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xmlns:context="http://www.springframework.org/schema/context"  
 xmlns:mvc="http://www.springframework.org/schema/mvc" xmlns:dubbo="http://code.alibabatech.com/schema/dubbo"  
 xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context.xsd http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc.xsd http://code.alibabatech.com/schema/dubbo http://code.alibabatech.com/schema/dubbo/dubbo.xsd"**>  
 *<!--0.加载属性文件-->* <**context:property-placeholder location="classpath\*:properties/\*.properties"**/>  
 *<!--1.加载控制器的扫描-->* <**context:component-scan base-package="com.pinyougou.shop.controller"**/>  
 *<!--2.添加注解驱动-->  
 <!--2.1)使用fastjson进行java对象与json字符串的互相转换-->* <**mvc:annotation-driven**>  
 <**mvc:message-converters register-defaults="true"**>  
 <**bean class="com.alibaba.fastjson.support.spring.FastJsonHttpMessageConverter"**>  
 <**property name="supportedMediaTypes" value="application/json"**/>  
 <**property name="features"**>  
 <**array**>  
 <**value**>WriteMapNullValue</**value**>  
 <**value**>WriteDateUseDateFormat</**value**>  
 </**array**>  
 </**property**>  
 </**bean**>  
 </**mvc:message-converters**>  
 </**mvc:annotation-driven**>  
 *<!--2.2)配置文件上传-->* <**bean id="multipartResolver" class="org.springframework.web.multipart.commons.CommonsMultipartResolver"**>  
 *<!--2.2.1)定义要上传的文件的字符集编码-->* <**property name="defaultEncoding" value="UTF-8"**/>  
 *<!--2.2.2）要上传的最大的文件大小-->* <**property name="maxUploadSize" value="5242880"**/>  
 </**bean**>  
 *<!--3.添加dubbo相关的配置-->* <**dubbo:application name="pinyougou-shop-web"**/>  
 <**dubbo:registry address="zookeeper://192.168.25.128:2181"**/>  
 <**dubbo:annotation package="com.pinyougou.shop.controller"**/>  
</**beans**>

#### 5.1 在pom中加入springSecurity相关的依赖

<dependency>

<groupId>org.springframework.security</groupId>

<artifactId>spring-security-web</artifactId>

<version>4.1.0.RELEASE</version>

</dependency>

<dependency>

<groupId>org.springframework.security</groupId>

<artifactId>spring-security-config</artifactId>

<version>4.1.0.RELEASE</version>

</dependency>

#### 5.2 自定义一个登录验证用的服务

**public class** UserDetailServiceImpl **implements** UserDetailsService {  
 **private** SellerService **sellerService**;  
  
 **public void** setSellerService(SellerService sellerService) {  
 **this**.**sellerService** = sellerService;  
 }  
  
 @Override  
 **public** UserDetails loadUserByUsername(String sellerId) **throws** UsernameNotFoundException {  
 *//1.根据商家id查询商家对象* TbSeller seller = **sellerService**.findSellerById(sellerId);  
 *//2.判断是否存在* **if**(seller != **null**){  
 *//2.1)判断是否己审核* **if**(seller.getStatus().equals(**"1"**)){  
 *//2.2)定义角色，并与返回的用户绑定* List<GrantedAuthority> authorities = **new** ArrayList<>();  
 authorities.add(**new** SimpleGrantedAuthority(**"ROLE\_ADMIN"**));  
 **return new** User(sellerId,seller.getPassword(),authorities);  
 }  
 }  
 **return null**;  
 }  
}

#### 5.3 在添加商家时对密码进行加密

*/\*\*  
 \* 增加  
 \** ***@param seller*** *\** ***@return*** *\*/*@RequestMapping(**"/add"**)  
**public** Result add(@RequestBody TbSeller seller){  
 **try** {  
 *//1.将商家的密码进行加密* BCryptPasswordEncoder encoder = **new** BCryptPasswordEncoder();  
 seller.setPassword(encoder.encode(seller.getPassword()));  
 *//2.将经过加密后的商家对象保存到数据库中* **sellerService**.add(seller);  
 **return new** Result(**true**, **"增加成功"**);  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 **return new** Result(**false**, **"增加失败"**);  
 }  
}

## 商品管理

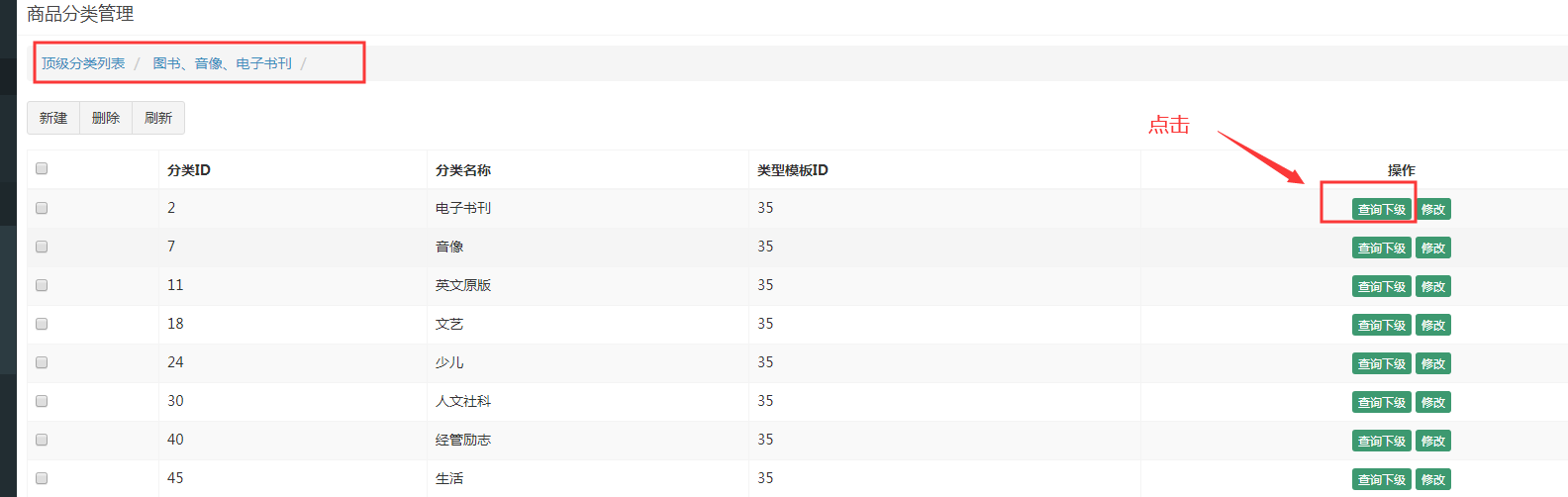
### 商品的分类

#### 需求分析

实现三级商品分类列表查询功能

进入页面首先显示所以一级分类，效果如下：







#### 表结构



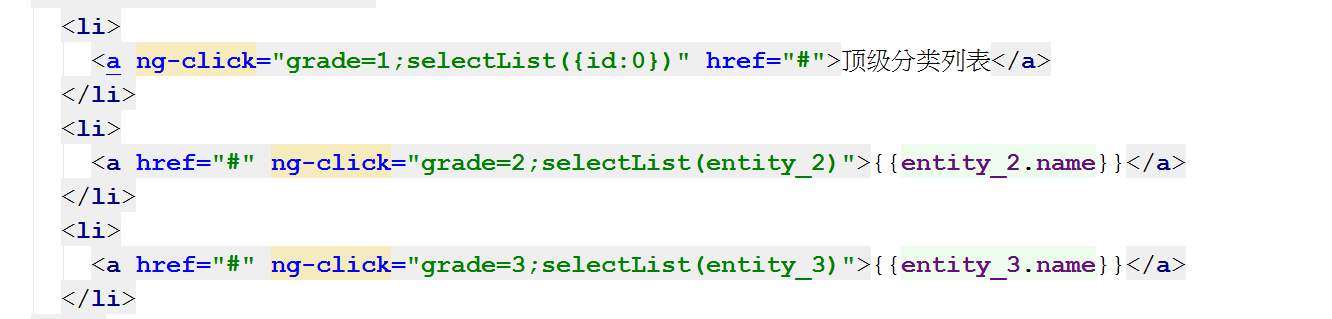
注意这个表是自关联的！！！！！！！！！！！！！！

#### 1.3 实现过程

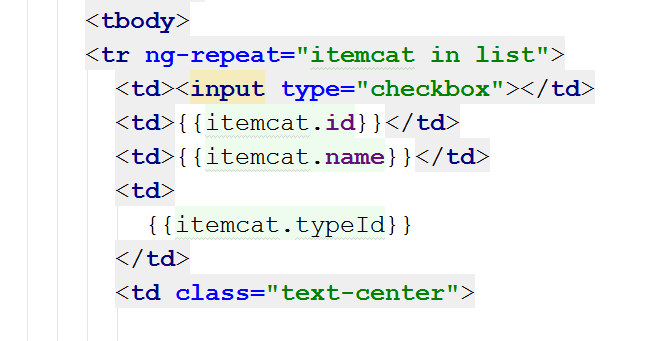
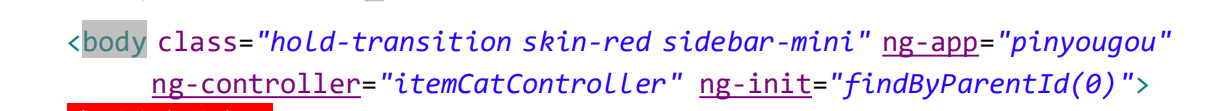
Js代码：

面包屑导航：：

*//根据当前点击的分类，确定对应分类的实体（一级：entity\_1,二级：entity\_2,三级：entity\_3）,然后，再根据这个id作为父id查询其子分类列表*$scope.selectList=(entity)=>{  
 **if**($scope.**grade** == 1){ *//一级分类* $scope.**entity\_1** = entity; *//一级分类* $scope.**entity\_2** = **null**; *//二级分类* $scope.**entity\_3** = **null**; *//三级分类* }**else if**($scope.**grade** == 2){  
 $scope.**entity\_2** = entity; *//二级分类* $scope.**entity\_3** = **null**; *//三级分类* }**else if**($scope.**grade** == 3){  
 $scope.**entity\_3** = entity; *//三级分类* }  
 *//根据分类的id作为父id查询子分类列表* $scope.findByParentId(entity.**id**);  
}



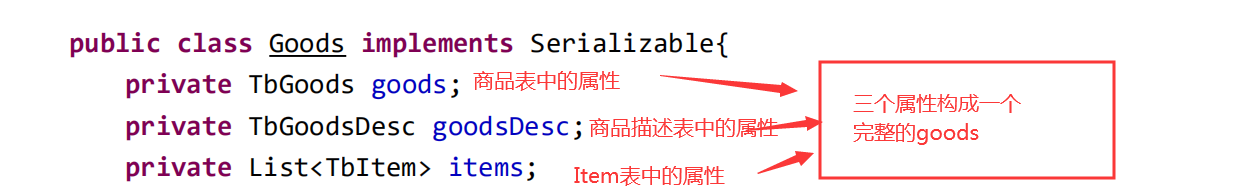
点击商品分类就 定义一个初始化方法出巡所有的顶级目录



后台就根据传来的id去差数据（外键查询就行了）

### 商品录入

#### 2.1 新建一个组合类Goods（pojo下）



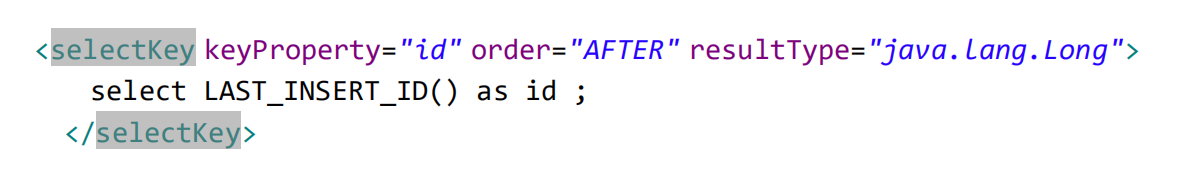
#### 2.2 在 pinyougou-sellgoods-interface 下修改 add():

#### 2.3 在 pinyougou-sellgoods-service 下修改 add():



#### 2.4 在 pinyougou-dao 下的 TbGoodsMapper.xml 文件的 insert 中添加如下：

让主键添加完了之后能被立即感知



### 富文本编辑器

#### 3.1 用途

富文本编辑器，Rich Text Editor, 简称 RTE, 它提供类似于 Microsoft Word 的编辑功能。常

用的富文本编辑器：

KindEditor http://kindeditor.net/

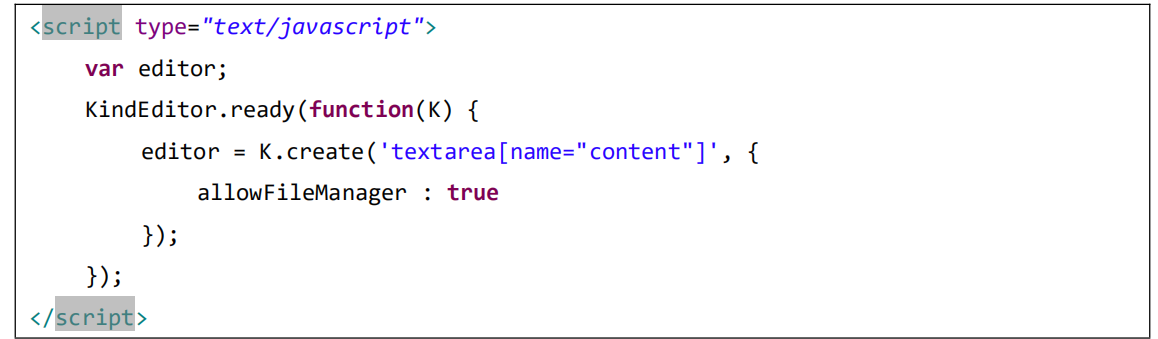
UEditor http://ueditor.baidu.com/website/

CKEditor <http://ckeditor.com/>

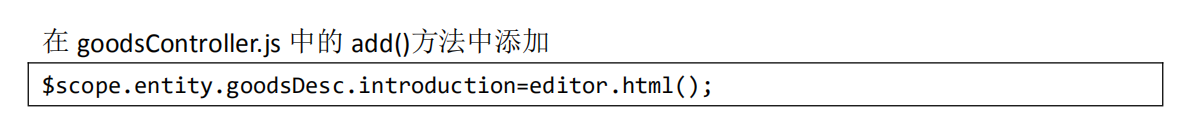
使用富文本编辑器完成商品介绍的录入：

初始化：

向html中加上如下js代码



如何提取编辑器的内容：



如何清空编辑器的内容：（用于新的修改的时候）



数据绑定：

举几个例子其他相同：



### 文件上传：分布式文件服务器FastDFS

#### 4.1、什么是FastDFS

FastDFS 是用 c 语言编写的一款开源的分布式文件系统。FastDFS 为互联网量身定制，

充分考虑了冗余备份、负载均衡、线性扩容等机制，并注重高可用、高性能等指标，使用

FastDFS 很容易搭建一套高性能的文件服务器集群提供文件上传、下载等服务。

FastDFS 架构包括 Tracker server 和 Storage server。客户端请求 Tracker server 进行文

件上传、下载，通过 Tracker server 调度最终由 Storage server 完成文件上传和下载。

Tracker server 作用是负载均衡和调度，通过 Tracker server 在文件上传时可以根据一些

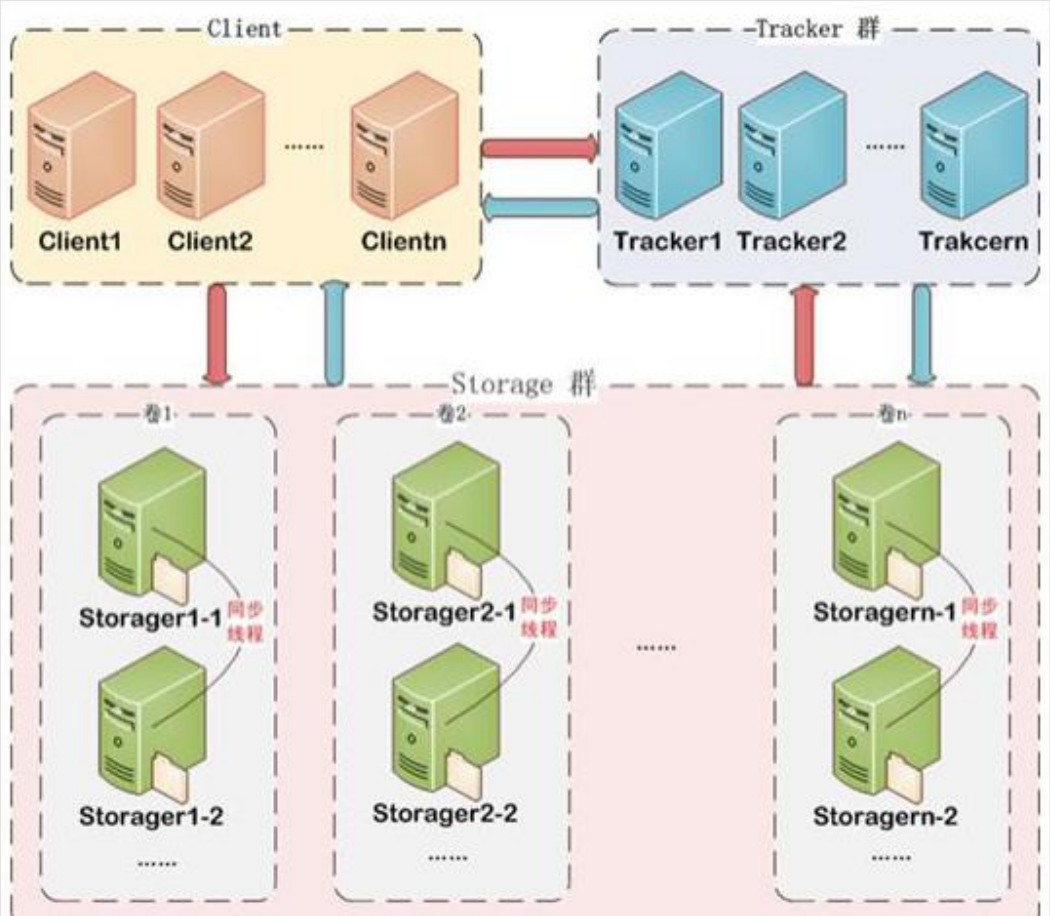
策略找到 Storage server 提供文件上传服务。可以将 tracker 称为追踪服务器或调度服务

器。

Storage server 作用是文件存储，客户端上传的文件最终存储在 Storage 服务器上，

Storageserver 没有实现自己的文件系统而是利用操作系统 的文件系统来管理文件。可以将

storage 称为存储服务器。



服务端两个角色：

Tracker：管理集群，tracker 也可以实现集群。每个 tracker 节点地位平等。收集 Storage

集群的状态。深圳市泽林信息-JAVAEE 电商项目 主讲：王峰 QQ：2568783935

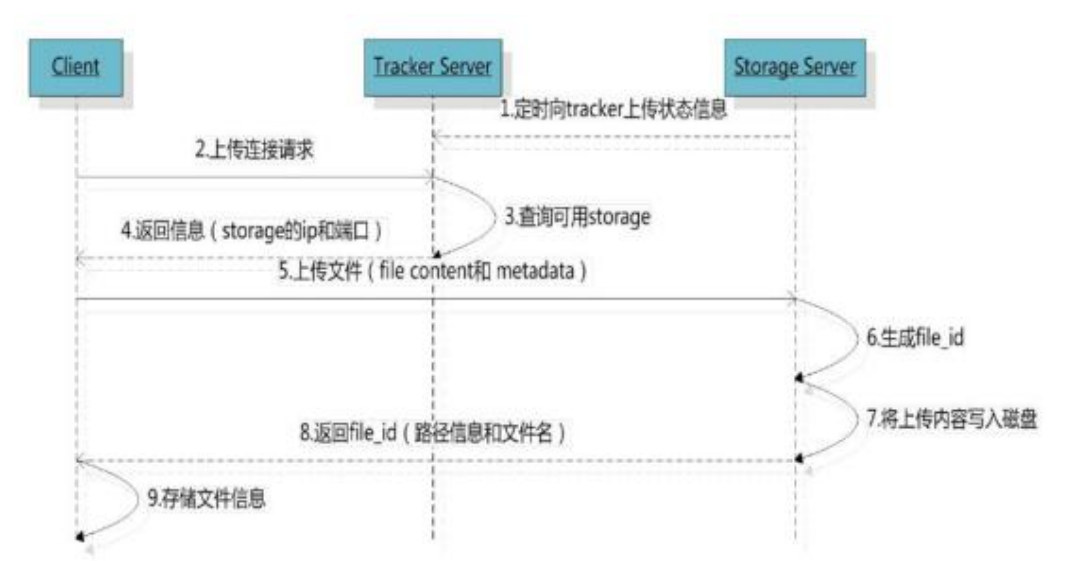
Storage：实际保存文件 Storage 分为多个组，每个组之间保存的文件是不同的。每

个组内部可以有多个成员，组成员内部保存的内容是一样的，组成员的地位是一致的，没有

主从的概念。

#### 4.2 文件上传与下载的流程

4.2.1 文件上传的流程



客户端上传文件后存储服务器将文件 ID 返回给客户端，此文件 ID 用于以后访问该文

件的索引信息。文件索引信息包括：组名，虚拟磁盘路径，数据两级目录，文件名。

例如：group1/M00/02/44/fjsdkfjhg24h5kjfhsdkjf345.gif

**组名**：文件上传后所在的 storage 组名称，在文件上传成功后有 storage 服务器返回，

需要客户端自行保存。

**虚拟磁盘路径**：storage 配置的虚拟路径，与磁盘选项 store\_path\*对应。如果配置了

store\_path0 则是 M00，如果配置了 store\_path1 则是 M01，以此类推。

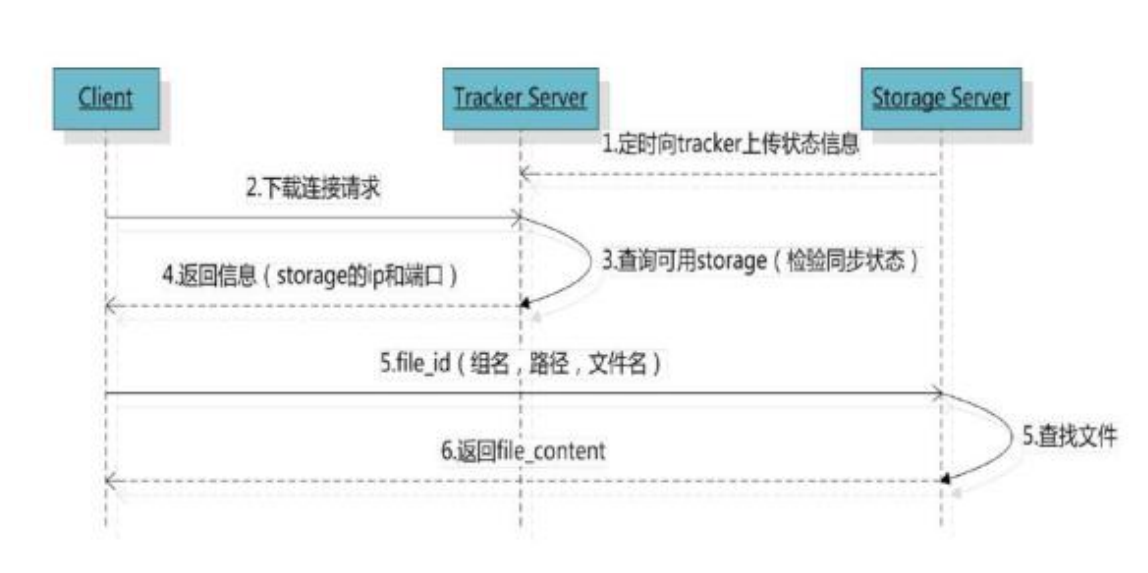
**数据两级目录**：storage 服务器在每个虚拟磁盘路径下创建的两级目录，用于存储数据

文件。

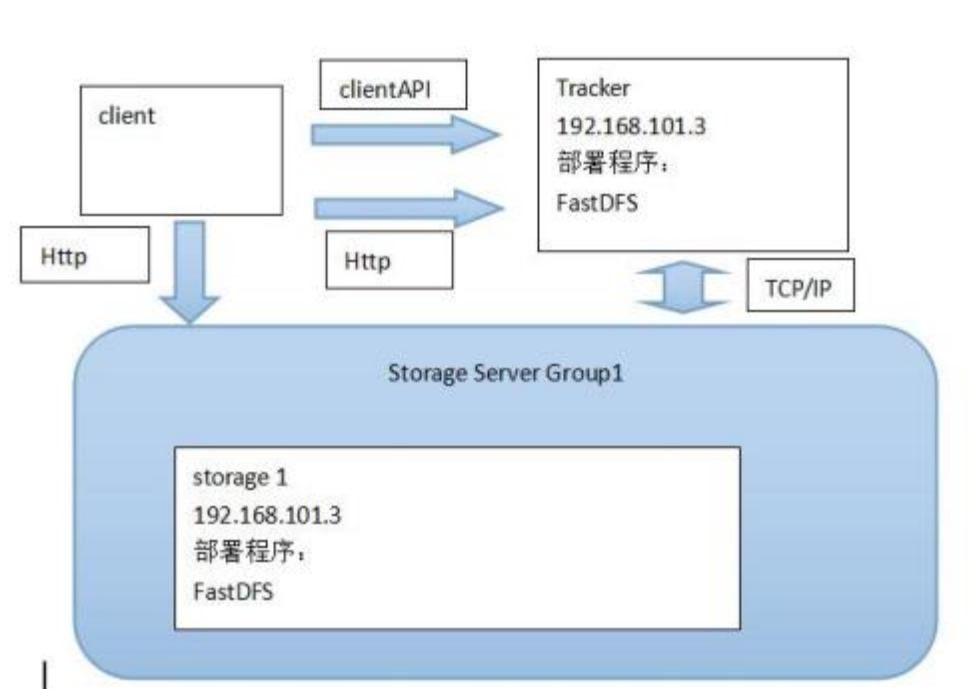
**文件名**：与文件上传时不同。是由存储服务器根据特定信息生成，文件名包含：源存储

服务器 IP 地址、文件创建时间戳、文件大小、随机数和文件拓展名等信息。

4.2.2 文件的下载流程



4.2.3 最简单的FastDFS架构



4.2.4 FastDFS的快速安装

考一个别人已经安装好的镜像虚拟机 把虚拟机启动

4.2.5 快速使用FastDFS

步骤：

* 导入jar包

<dependency>

<groupId>org.csource.fastdfs</groupId>

<artifactId>fastdfs</artifactId>

<version>1.2</version>

</dependency>

* 添加配置文件：

将其中的tracker\_server的地址设置为服务器部署时的地址

//......

tracker\_server=192.168.25.133:22122

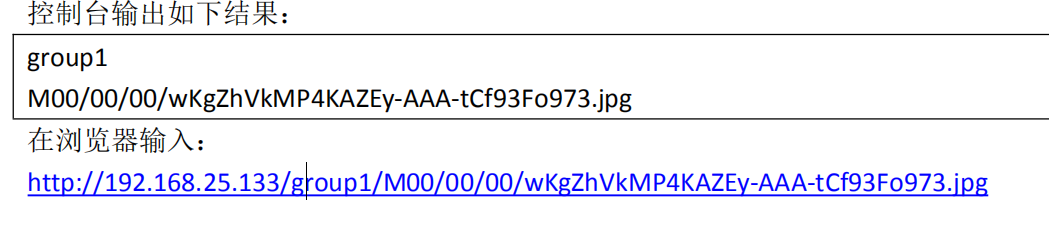
//......

* 创建java工具类：

***/\*  
 \* @Author:wtp  
 \* @Date:Create at 2019/11/14 0014  
 \* @TIME:下午 4:42  
 \* @Description:  
 \* @Tips:  
 \*/*public class** FastFDSClient {  
 **private** TrackerClient **trackerClient**;  
 **private** TrackerServer **trackerServer**;  
 **private** StorageClient1 **storageClient1**;  
 **private** StorageServer **storageServer**;  
  
 ***//* public** FastFDSClient(String conf) **throws** Exception {  
 **if** (conf.contains(**"classpath:"**)) {  
 conf = conf.replace(**"classpath:"**,  
 **this**.getClass().getResource(**"/"**).getPath());  
 }  
 ClientGlobal.*init*(conf);  
 **trackerClient** = **new** TrackerClient();  
 **trackerServer** = **trackerClient**.getConnection();  
 **storageServer** = **null**;  
 **storageClient1** = **new** StorageClient1(**trackerServer**, **storageServer**);  
 }  
  
 ***//上传文件的方法*** */\*\*  
 \** ***@param fileName*** *文件的绝对路径  
 \** ***@param extName*** *文件的扩展名  
 \** ***@param meta*** *文件的扩展信息  
 \** ***@return*** *返回一个字符串 形如  
 \** ***@throws*** *Exception  
 \*/* ***//三个参数* public** String uploadFile(String fileName, String extName, NameValuePair[] meta) **throws** Exception {  
 String result = **storageClient1**.upload\_appender\_file1(fileName, extName, meta);  
 **return** result;  
 }  
  
 ***//一个参数* public** String uploadFile(String fileName) **throws** Exception {  
 **return** uploadFile(fileName, **null**, **null**);  
 }  
  
 ***//两个参数* public** String uploadFile(String fileName, String extName) **throws** Exception {  
 **return** uploadFile(fileName, extName, **null**);  
 }  
  
 */\*\*  
 \*  
 \** ***@param fileContent*** *文件的内容 字节数组  
 \** ***@param extName*** *文件的扩展名  
 \** ***@param metas*** *文件的扩展信息  
 \** ***@return*** *\** ***@throws*** *Exception  
 \*/* **public** String uploadFile(**byte**[] fileContent, String extName,  
 NameValuePair[]metas) **throws** Exception {  
 String result = **storageClient1**.upload\_file1(fileContent, extName, metas);  
 **return** result;  
 }  
  
 **public** String uploadFile(**byte**[] fileContent) **throws** Exception {  
 **return** uploadFile(fileContent, **null**, **null**);  
 }  
  
 **public** String uploadFile(**byte**[] fileContent, String extName) **throws** Exception {  
 **return** uploadFile(fileContent, extName, **null**);  
 }  
}

如何查看结果：

服务器的ip 加上输出的文件完整名



## 商品的录入（2）

### 商品分类

需求：

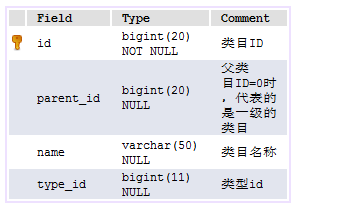
三个下拉框，第一个为顶级目录，第二个为二级目录，第三个为三级目录；

开始时第一个下拉框可以选择顶级目录，选择好第一个下拉框后可以选择其对应的子级 下拉框；

点击第二个下拉框的时候，可以把第三个下拉框可以供选择；

表结构：

自关联的一张表：



Js控制代码：

***//点击当前分类查询不同级别下的目录***$scope.selectList=(entity)=>{  
 **if** ($scope.**grade**==1){  
 ***//一级目录那么 其他的为空*** $scope.**entity\_1**=entity;  
 $scope.**entity\_2**=**null**;  
 $scope.**entity\_3**=**null**;  
 }**else if** ($scope.**grade**==2){  
 $scope.**entity\_2**=entity;  
 $scope.**entity\_3**=**null**;  
 } **else if** ($scope.**grade**==3){  
 $scope.**entity\_3**=entity;  
 }  
 ***//父类的parentId作为 当前对象的id来查询对应的分类列表*** $scope.findByParentId(entity.**id**);  
}

前端代码：

利用内置的watch对象

***//查询顶级分类***$scope.findByParentId = (parentId) => {  
 itemCatService.findByParentId(parentId).success(response => {  
 $scope.**itemCatList1** = response;  
 })  
}  
***//监控顶级分类的值 当发生变化时 调用函数查询该顶级分类的子分类***$scope.$watch(**"entity.goods.catagroy1Id"**, **function** (newValue, oldValue) {  
 itemCatService.findByParentId(newValue).success(response => {  
 $scope.**itemCatList2** = response;  
 })  
})  
  
***//监控二级分类的值 当发生变化时 调用函数查询该二级分类的子分类***$scope.$watch(**"entity.goods.catagroy2Id"**, **function** (newValue, oldValue) {  
 itemCatService.findByParentId(newValue).success(response => {  
 $scope.**itemCatList3** = response;  
 })  
})  
***//监控三级分类 当发生变话的时候 调用函数查询对应模板id***$scope.$watch(**"entity.goods.catagroy3Id"**, **function** (newValue, oldValue) {  
 itemCatService.findOne(newValue).success(response => {  
 $scope.**entity**.**goods**.**typeTemplateId** = response.typeId;  
 })  
})

页面模板：

<**td**>  
 <**select class="form-control" ng-model="entity.goods.catagroy1Id"  
 ng-options="item.id as item.name for item in itemCatList1"**>  
 </**select**>  
</**td**>  
<**td**>  
 <**select class="form-control select-sm" ng-model="entity.goods.catagroy2Id"  
 ng-options="item.id as item.name for item in itemCatList2"**></**select**>  
</**td**>  
<**td**>  
 <**select class="form-control select-sm" ng-model="entity.goods.catagroy3Id"  
 ng-options="item.id as item.name for item in itemCatList3"**></**select**>  
</**td**>

### 商品品牌录入

品牌其实跟商品分类差不多，

通过监控三级目录的变化来决定品牌的可选总共内容：

前端代码：

***//根据上面得到的模板id查找对应的品牌下拉框 （在 `tb\_type\_template` 表中 查找 brand\_ids）  
//监控模板id 查找对应的模板对象 只要模板对象出来了 什么都有了***$scope.$watch(**"entity.goods.typeTemplateId"**, **function** (newValue, oldValue) {  
 ***//根据模板id查询出模板对象 调用模板对应的服务*** typeTemplateService.findOne(newValue).success(response => {  
 ***//绑定模板对象*** $scope.**TypeTemplate** = response;  
 ***//获得对应的品牌列表 (!!!!!!!!!!!!!!!!!!!注意json格式)*** $scope.**brandList** = ***JSON***.parse(response.brandIds)  
 ***//将商品描述表中的扩展属性与 根据模板id查询到的模板对象中的扩展属性进行绑定 ( 注意json格式)*** $scope.**entity**.**goodsDesc**.**customAttributeItems** = ***JSON***.parse(response.**customAttributeItems**)  
 })  
  
 ***//根据模板id 查出规格列表*** typeTemplateService.findSpecList(newValue).success(response => {  
 $scope.**specList** = response;  
 })  
})

页面模板

<**div class="col-md-10 data"**>  
 <**select class="form-control" ng-model="entity.goods.brandId"  
 ng-options="brand.id as brand.text for brand in brandList"**></**select**>  
</**div**>

后台根据传去的id查询就行了

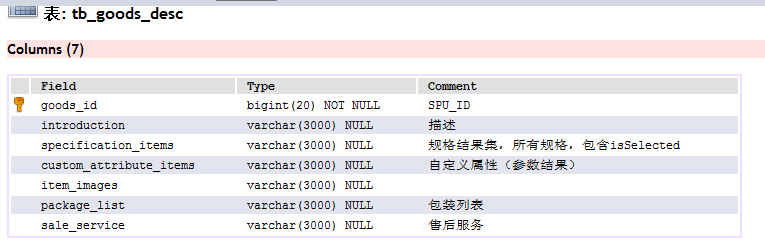
### 商品扩展属性录入

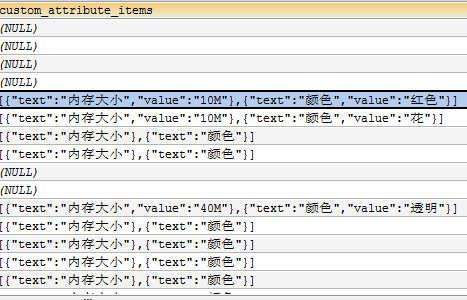
需求：

模板id查询到了也就可以根据模板id查找扩展属性



表结构：（商品描述表中）

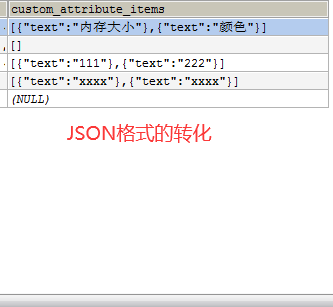




Js代码

***//根据上面得到的模板id查找对应的品牌下拉框 （在 `tb\_type\_template` 表中 查找 brand\_ids）  
//监控模板id 查找对应的模板对象 只要模板对象出来了 什么都有了***$scope.$watch(**"entity.goods.typeTemplateId"**, **function** (newValue, oldValue) {  
 ***//根据模板id查询出模板对象 调用模板对应的服务*** typeTemplateService.findOne(newValue).success(response => {  
 ***//绑定模板对象*** $scope.**TypeTemplate** = response;  
 ***//获得对应的品牌列表 (!!!!!!!!!!!!!!!!!!!注意json格式)*** $scope.**brandList** = ***JSON***.parse(response.brandIds)  
 ***//将商品描述表中的扩展属性与 根据模板id查询到的模板对象中的扩展属性进行绑定 ( 注意json格式)*** $scope.**entity**.**goodsDesc**.**customAttributeItems** = ***JSON***.parse(response.**customAttributeItems**)  
 })

后台根据模板id 查找注意json格式的转化



### 商品规格录入

Js代码

***//根据模板id 查出规格列表***typeTemplateService.findSpecList(newValue).success(response => {  
 $scope.**specList** = response;  
})

前端页面

<**div ng-repeat="spec in specList"**>  
 <**div class="col-md-2 title"**>{{**spec**.**text**}}</**div**>  
 <**div class="col-md-10 data"**>  
 <**span ng-repeat="option in spec.options"**>  
 <**input type="checkbox" ng-click="updateSelection($event,spec.text,option.optionName)"** >{{**option**.optionName}}  
 </**span**>  
 </**div**>

### SKU信息录入

Js代码

***//定义私有方法 传入 集合 key value 返回查询到的对象***searchObjectByKey = (list, key, value) => {  
 ***//遍历这个集合* for** (**let** i = 0; i < list.**length**; i++) {  
 ***//如果数组中这个key对应的值 与传来的value相等 那么说明这个集合中含有这个对象 返回这个对象* if** (list[i][key] == value) {  
 **return** list[i];  
 }  
 }  
 **return null**;  
}  
***//点击某个某个选项的时候就向$scope.entity.goodsDesc.specificationItems 这个数组中添加对象  
//1。第一个参数 event 表示点击事件  
//2.第二个参数表示 传来的key***$scope.updateSelection = (event, text, value) => {  
 ***//根据指定的规格名称的key（attributeName）以及名称对应的值 来查询这个对象是否在$scope.entity.goodsDesc.specificationItems中* var** obj = searchObjectByKey($scope.**entity**.**goodsDesc**.**specificationItems**, **"attributeName"**, text);  
 ***//如果存在* if** (obj) {  
 ***//判断是否被选中* if** (event.**target**.**checked**) {  
 ***//被选中将这个对象放进obj.attributeValue*** obj.**attributeValue**.push(value);  
 } **else** {  
 ***//将这个值从obj.attributeValue中删除*** obj.**attributeValue**.splice(obj.**attributeValue**.indexOf(value), 1);  
 ***//如果obj.attributeValue这个数组中没有值了 那么久将这个obj从 $scope.entity.goodsDesc.specificationItems中删除* if** (obj.**attributeValue**.length == 0) {  
 ***//被选中将这个对象从obj.attributeValue中删除*** $scope.**entity**.**goodsDesc**.**specificationItems**.splice($scope.**entity**.**goodsDesc**.**specificationItems**.indexOf(obj), 1)  
 }  
 }  
 } **else** {  
 ***//第一次添加*** $scope.**entity**.**goodsDesc**.**specificationItems**.push({**"attributeName"**: text, **"attributeValue"**: [value]})  
 }  
 ***//添加完成就生成具体的商品 SKU*** createItemList();  
}  
***//生成SKU列表***createItemList = () => {  
 ***//实体化SKU商品列表*** $scope.**entity**.**items** = [{**spec**: {}, **price**: 0, **num**: 99999, **status**: **'0'**, **isDefault**: **'0'**}];  
 ***//取得原始的规格选项内容* let** itemList = $scope.**entity**.**goodsDesc**.**specificationItems**;  
 ***//向这个规格列表中添加值  
 //遍历选中的规格 所在的列表* for** (**let** i = 0, len = itemList.length; i < len; i++) {  
 ***//向规格列表所在数组加入要插入数据库的值*** $scope.**entity**.**items** = addColumn($scope.**entity**.**items**, itemList[i].attributeName, itemList[i].**attributeValue**);  
 }  
}  
  
***//上面的方法 返回一个新的数组存放的是形如 [{spec:{“网络”:”3G”},price:0,num:99999,status:'0',isDefault:'0' }，  
//这样的数组 {spec:{“网络”:”4G”},price:0,num:99999,status:'0',isDefault:'0' }]***addColumn = (list, attributeName, attributeValue) => {  
 ***//定义一个新的集合用来接收值* var** newList = [];  
 ***//遍历传来的集合 进行重构* for** (**let** i = 0; i < list.**length**; i++) {  
 **var** oldRow = list[i];  
 **for** (**let** j = 0; j < attributeValue.**length**; j++) {  
 ***//根据老的集合构造新的对象* var** newRow = ***JSON***.parse(***JSON***.stringify(oldRow)); ***//深克隆  
 //重新对新的集合赋值*** newRow.**spec**[attributeName]=attributeValue[j];  
 ***//将新的集合放到newList中*** newList.push(newRow);  
 }  
 }  
 ***//返回* return** newList;  
}

前端页面的遍历

<**div class="row data-type"**>  
 <**table class="table table-bordered table-striped table-hover dataTable"**>  
 <**thead**>  
 <**tr**>  
 <**th class="sorting" ng-repeat="spec in entity.goodsDesc.specificationItems"**>{{**spec**.attributeName}}</**th**>  
 <**th class="sorting"**></**th**>  
 <**th class="sorting"**>价格</**th**>  
 <**th class="sorting"**>库存</**th**>  
 <**th class="sorting"**>是否启用</**th**>  
 <**th class="sorting"**>是否默认</**th**>  
 </**tr**>  
 </**thead**>  
 <**tbody**>  
 <**tr ng-repeat="item in entity.items"**>  
 <**td ng-repeat="spec in entity.goodsDesc.specificationItems"**>  
 {{***item***.**spec**[**spec**.attributeName]}}  
 </**td**>  
 <**td**>  
 <**input class="form-control" ng-model="item.price" placeholder="价格"**>  
 </**td**>  
 <**td**>  
 <**input class="form-control" ng-repeat="item.num" placeholder="库存数量"**>  
 </**td**>  
 <**td**>  
 <**input ng-model="item.status" type="checkbox" ng-true-value="1" ng-false-value="0"**>  
 </**td**>  
 <**td**>  
 <**input ng-model="item.isDefault" type="checkbox" ng-true-value="1" ng-false-value="0"**>  
 </**td**>  
 </**tr**>