**淘淘商城**

**第五天**

# 课程计划

1. 搭建门户系统
2. 实现首页访问
3. 内容管理系统介绍
4. 实现内容分类管理
5. 实现内容管理

# 搭建门户系统

## 什么是门户？

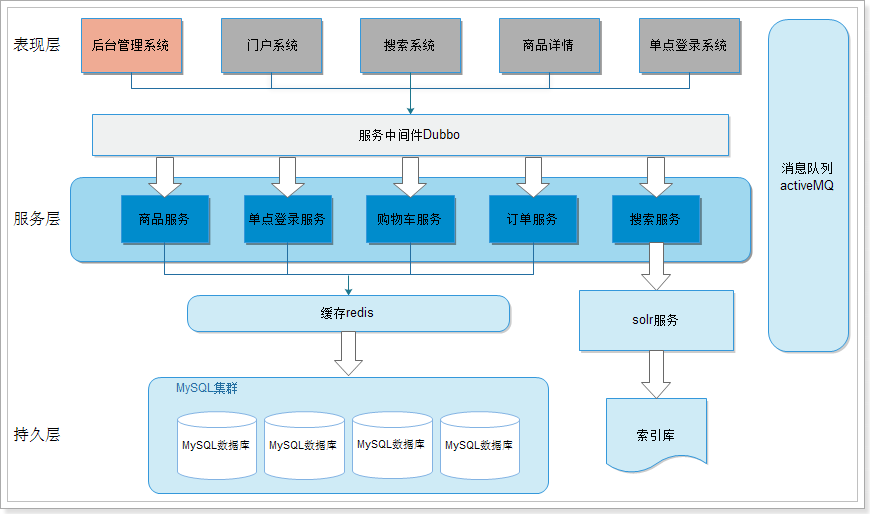
广义上的门户就是将各种应用系统、数据资源和互联网资源集成到一个信息管理平台之上，并以统一的用户界面提供给用户，并建立企业对客户、企业对内部员工和企业对企业的信息通道。

**其实就是网站入口。**

门户系统：用户，消费者

后台系统：入住的商家、网站管理者

## 创建门户系统



## 所使用的技术

门户系统不会直接访问数据库，只需要给用户提供页面展示

是通过后台系统服务（接口）调用数据

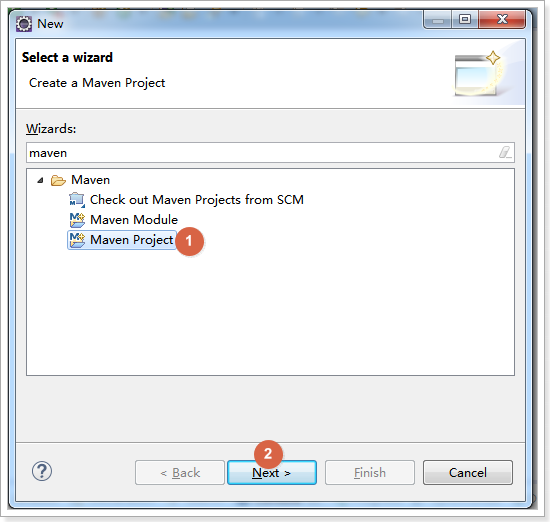
需要的技术：

springMVC

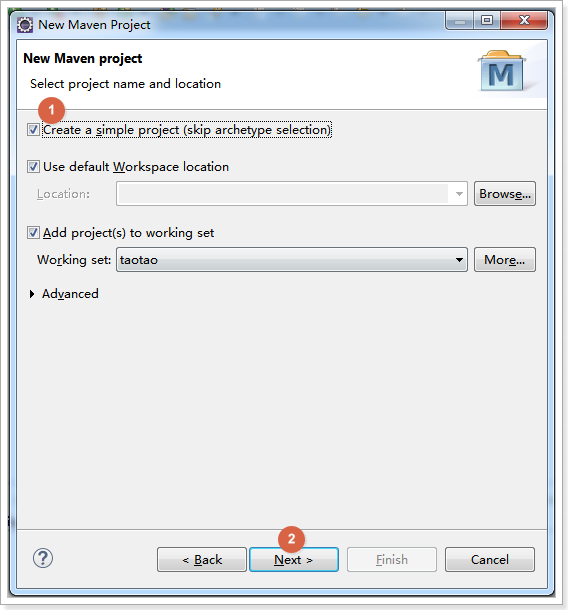
js，html，css，等

dubbo

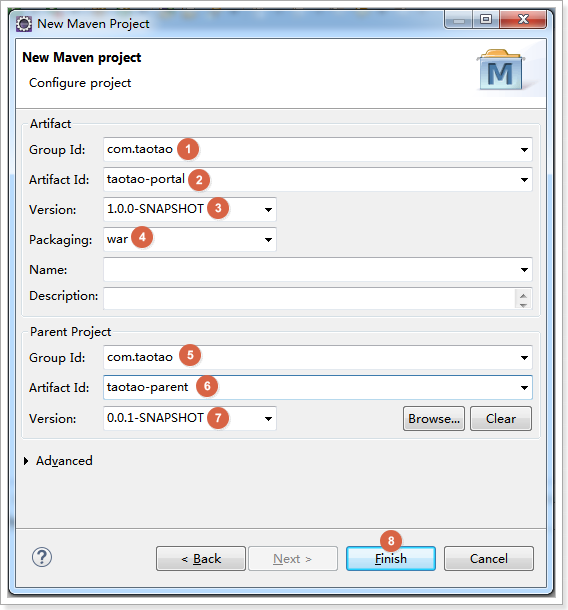
创建Maven工程



创建简单工程



填写创建信息，门户系统是web应用，需要打war包



## 加入依赖

<project xmlns=*"http://maven.apache.org/POM/4.0.0"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xsi:schemaLocation=*"http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"*>

<modelVersion>4.0.0</modelVersion>

<parent>

<groupId>com.taotao</groupId>

<artifactId>taotao-parent</artifactId>

<version>0.0.1-SNAPSHOT</version>

</parent>

<groupId>com.taotao</groupId>

<artifactId>taotao-portal</artifactId>

<version>1.0.0-SNAPSHOT</version>

<packaging>war</packaging>

<dependencies>

<!-- 单元测试 -->

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<scope>test</scope>

</dependency>

<!-- Spring -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-jdbc</artifactId>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aspects</artifactId>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context-support</artifactId>

</dependency>

<!-- 日志 -->

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-log4j12</artifactId>

</dependency>

<!-- Jackson Json处理工具包 -->

<dependency>

<groupId>com.fasterxml.jackson.core</groupId>

<artifactId>jackson-databind</artifactId>

</dependency>

<!-- JSP相关 -->

<dependency>

<groupId>jstl</groupId>

<artifactId>jstl</artifactId>

</dependency>

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>servlet-api</artifactId>

<scope>provided</scope>

</dependency>

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>jsp-api</artifactId>

<scope>provided</scope>

</dependency>

<!-- Apache通用工具组件 -->

<dependency>

<groupId>org.apache.commons</groupId>

<artifactId>commons-lang3</artifactId>

</dependency>

<dependency>

<groupId>org.apache.commons</groupId>

<artifactId>commons-io</artifactId>

</dependency>

<!-- dubbo相关 -->

<dependency>

<groupId>com.alibaba</groupId>

<artifactId>dubbo</artifactId>

<exclusions>

<exclusion>

<groupId>org.springframework</groupId>

<artifactId>spring</artifactId>

</exclusion>

</exclusions>

</dependency>

<dependency>

<groupId>org.apache.zookeeper</groupId>

<artifactId>zookeeper</artifactId>

</dependency>

<dependency>

<groupId>com.github.sgroschupf</groupId>

<artifactId>zkclient</artifactId>

</dependency>

</dependencies>

<build>

<plugins>

<!-- 配置Tomcat插件 -->

<plugin>

<groupId>org.apache.tomcat.maven</groupId>

<artifactId>tomcat7-maven-plugin</artifactId>

<configuration>

<port>8083</port>

<path>/</path>

</configuration>

</plugin>

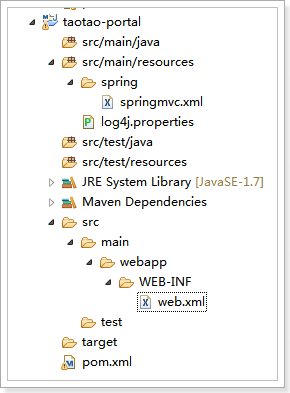
</plugins>

</build>

</project>

## 加入配置文件

需要加入的配置文件：springMVC，log4j，web.xml



### springmvc.xml

<?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:p=*"http://www.springframework.org/schema/p"*

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://code.alibabatech.com/schema/dubbo http://code.alibabatech.com/schema/dubbo/dubbo.xsd*

*http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc-4.0.xsd*

*http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context.xsd"*>

<!-- controller扫描 -->

<context:component-scan base-package=*"com.taotao.portal.controller"* />

<!-- 注解驱动 -->

<mvc:annotation-driven />

<!-- 视图解析器 -->

<bean

class=*"org.springframework.web.servlet.view.InternalResourceViewResolver"*>

<!-- 配置前缀 -->

<property name=*"prefix"* value=*"/WEB-INF/views/"* />

<!-- 配置后缀 -->

<property name=*"suffix"* value=*".jsp"* />

</bean>

<!-- 配置dubbo服务 -->

<dubbo:application name=*"taotao-portal"* />

<!-- 使用广播 -->

<!-- <dubbo:registry address="multicast://224.5.6.7:1234" /> -->

<dubbo:registry protocol=*"zookeeper"* address=*"192.168.37.161:2181"* />

<!-- 声明要调用的服务，timeout是设置连接超时最长时间,如果不设置，超时时间默认是3秒 -->

<!-- <dubbo:reference interface="com.taotao.manager.service.ItemCatService" -->

<!-- id="itemCatService" timeout="1000000" url="dubbo://127.0.0.1:20880/"

/> -->

</beans>

### web.xml

使用html结尾进入springMVC

为了SEO搜索引擎优化

搜素引擎的算法有分数，静态页面的分数比动态（jsp、asp、php）页面要高

<?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"*

id=*"MyWebApp"* version=*"2.5"*>

<display-name>taotao-portal</display-name>

<welcome-file-list>

<welcome-file>index.jsp</welcome-file>

</welcome-file-list>

<!-- 配置解决post提交乱码的问题 -->

<filter>

<filter-name>encoding</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>

</filter>

<filter-mapping>

<filter-name>encoding</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

<!-- 配置springMVC -->

<servlet>

<servlet-name>taotao-portal</servlet-name>

<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

<!-- 设置springMVC配置文件 -->

<init-param>

<param-name>contextConfigLocation</param-name>

<param-value>classpath:spring/springmvc.xml</param-value>

</init-param>

<load-on-startup>1</load-on-startup>

</servlet>

<!-- 配置springMVC映射 -->

<servlet-mapping>

<servlet-name>taotao-portal</servlet-name>

<!-- 所有以html结尾的请求进入springMVC -->

<!-- 伪静态，SEO优化 -->

<url-pattern>\*.html</url-pattern>

</servlet-mapping>

</web-app>

### log4j.properties

log4j.rootLogger=DEBUG,A1

log4j.logger.com.taotao = DEBUG

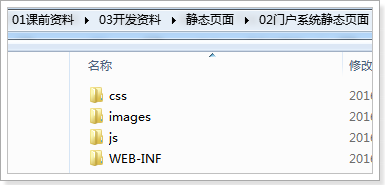
log4j.logger.org.mybatis = DEBUG

log4j.appender.A1=org.apache.log4j.ConsoleAppender

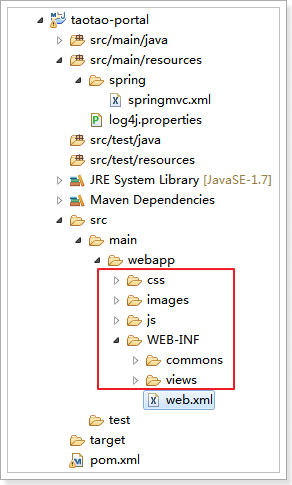
log4j.appender.A1.layout=org.apache.log4j.PatternLayout

log4j.appender.A1.layout.ConversionPattern=%-d{yyyy-MM-dd HH:mm:ss,SSS} [%t] [%c]-[%p] %m%n

## 加入静态文件



加入静态文件



## 实现首页访问

### 编写controller

在taotao-portal编写controller，实现首页跳转

首页有index.jsp，实际上就是跳转到这个页面

@Controller

@RequestMapping("index")

**public** **class** IndexController {

/\*\*

\* 跳转到首页

\*

\* **@return**

\*/

@RequestMapping(method = RequestMethod.***GET***)

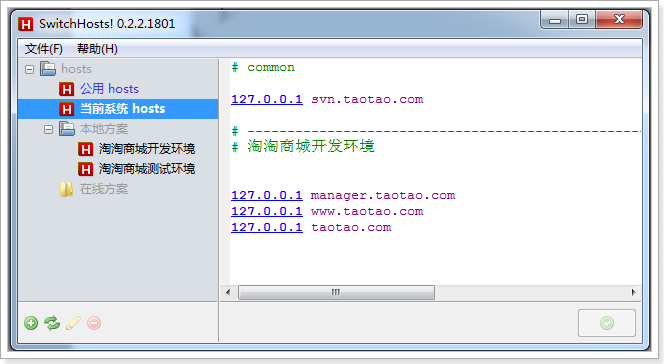
**public** String toIndex() {

**return** "index";

}

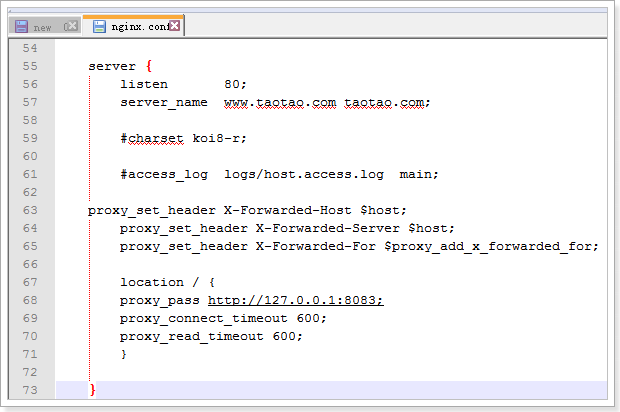
}

### 配置域名和ip的映射



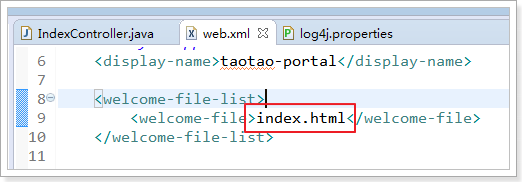
### 修改Nginx，实现反向代理

多个域名之间使用空格分隔

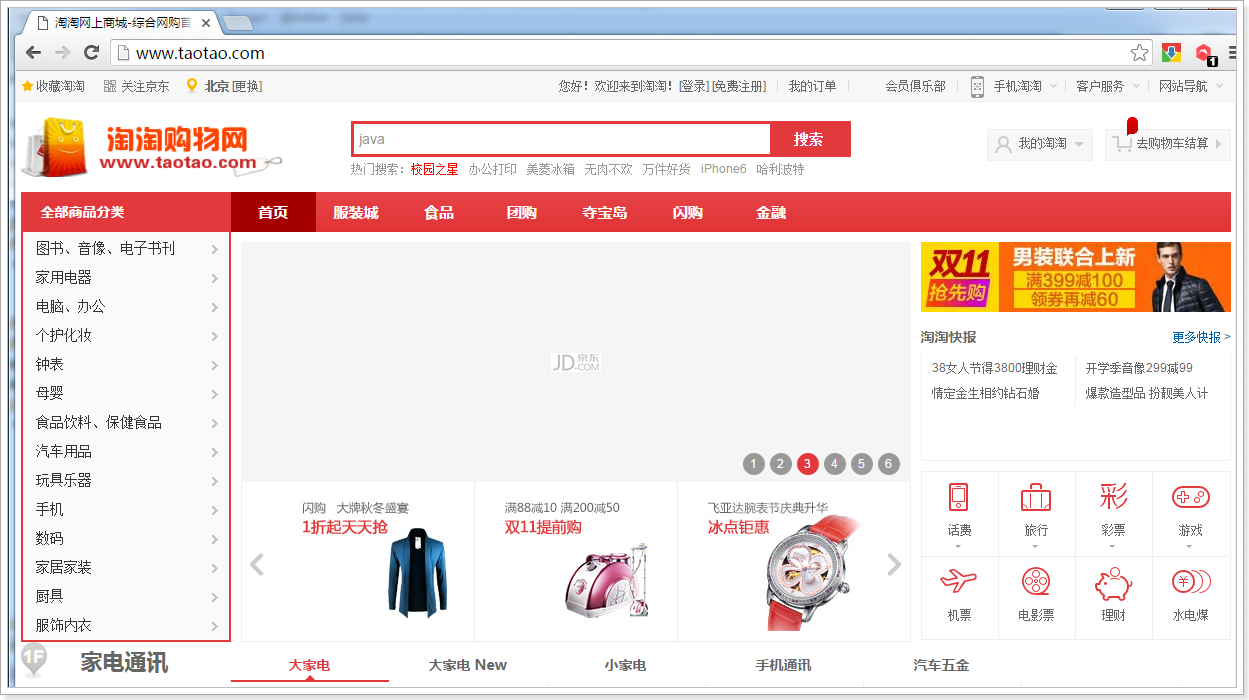


### 实现直接使用域名访问

配置web.xml



## 效果



# 内容管理系统介绍

## 首页大广告开发分析

可以根据首页大广告位的数据结构设计一张表，进行增删改查管理

其他部分的展示内容同样可以设计表，进行增删改查

存在的问题：

如果每一个前端展示内容（大广告位、小广告位等等），单独建立表，进行CRUD操作，会有以下问题

1. 页面大量信息堆积，发布显得异常沉重
2. 内容繁杂，管理效率低下
3. 许多工作需要技术人员配合完成
4. 改版工作量大，系统扩展性差

解决方案：

内容管理系统

## 内容管理系统

可以对首页展示功能进行抽取，发现有以下属性

1. 图片

2. url

3. 标题

4. 副标题（价格）

把首页的每个展示功能（大广告位，淘淘快报等），看作是一个分类

每个展示功能里面展示的多条信息，看作是分类下的内容

例如：首页大广告，对应的是大广告分类，而大广告位展示的多张图片，就是大广告分类下的内容

前台需要获取大广告的图片，只需要根据大广告的id查询对应的内容即可

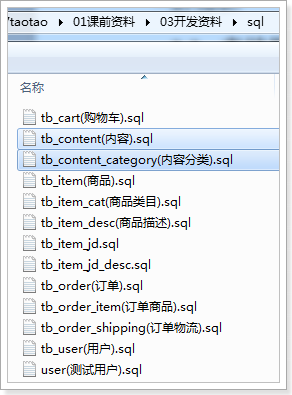
这样就需要两张表

一张内容分类表

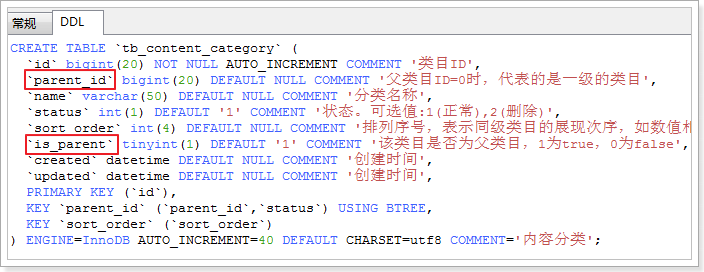
一张内容表

## 表结构

导入表：

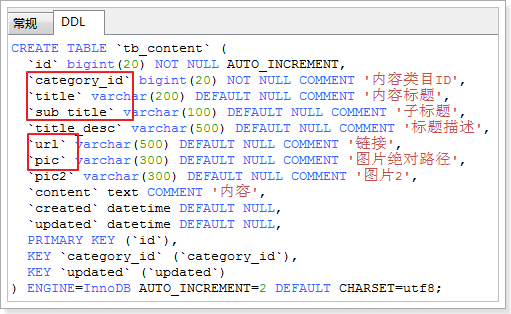


### 内容分类表



树形结构

### 内容表



1图片 pic

2 url url

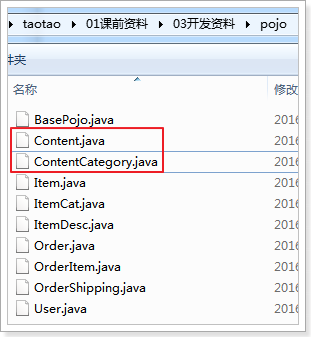
3 标题 title

4 副标题（价格） sub\_title

Category\_id 所属分类的id

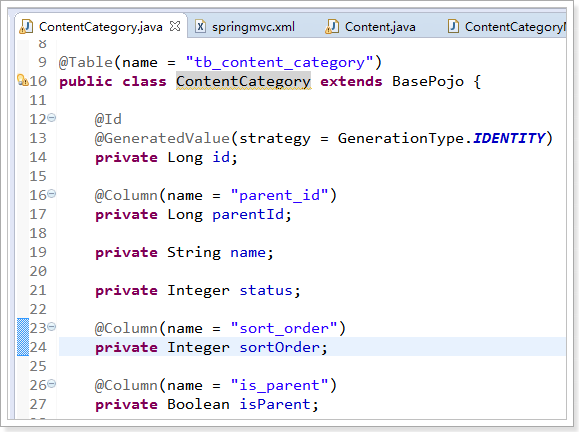
## 创建Pojo

课前资料：



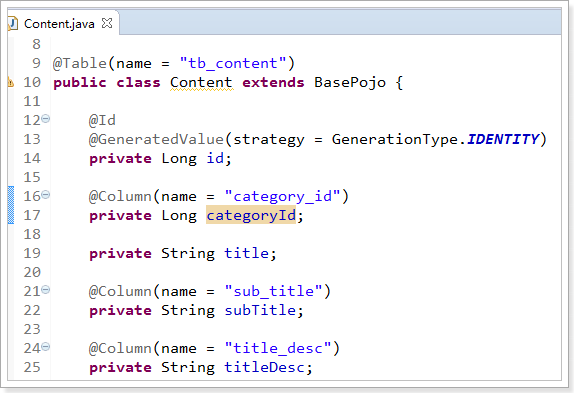
### ContentCategory

在taotao-manager-pojo创建



### Content

在taotao-manager-pojo创建

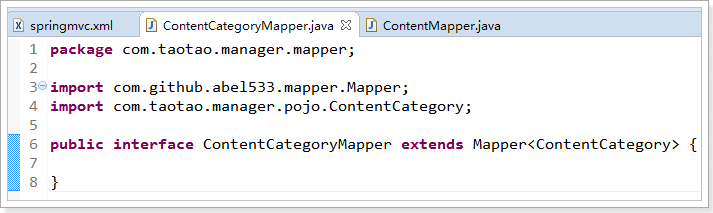


## 创建Mapper

需要分别创建ContentCategoryMapper和ContentMapper

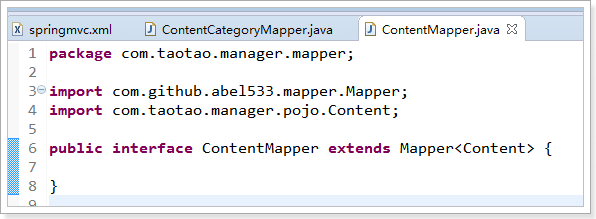
### 创建ContentCategoryMapper

在taotao-manager-mapper创建



### 创建ContentMapper

在taotao-manager-mapper创建

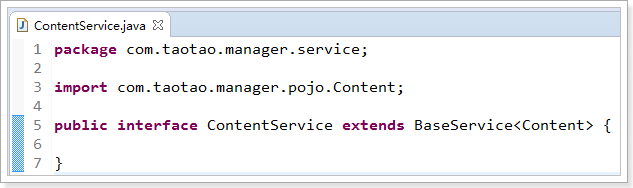


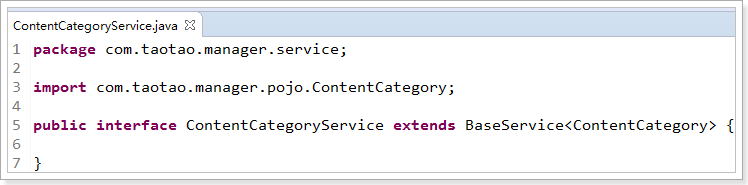
## 创建Service

分别创建内容分类和内容的Service接口和Service接口实现类

### 创建接口

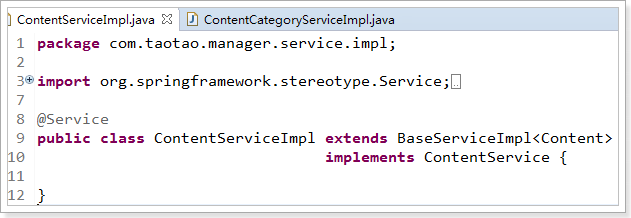
taotao-manager创建两个接口

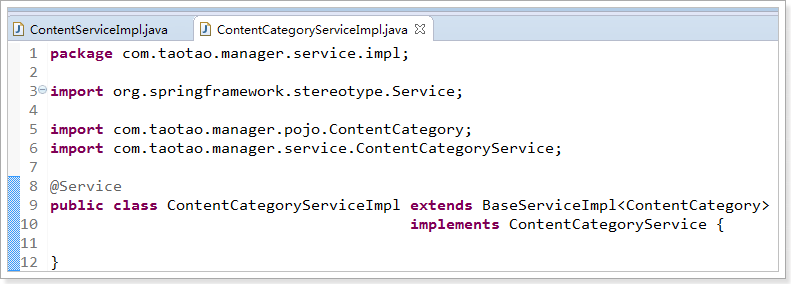




### 创建实现类

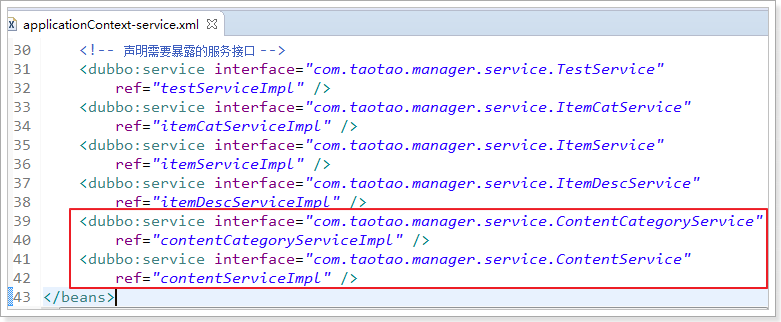
taotao-manager-service创建两个接口的实现类





### 发布服务

在taotao-manager-service进行服务的发布，发布刚刚编写的两个service



## 创建Controller

### 创建controller

在taotao-manager-web创建

@Controller

@RequestMapping("content")

**public** **class** ContentController {

@Autowired

**private** ContentService contentService;

}

在taotao-manager-web创建

@Controller

@RequestMapping("content/category")

**public** **class** ContentCategoryController {

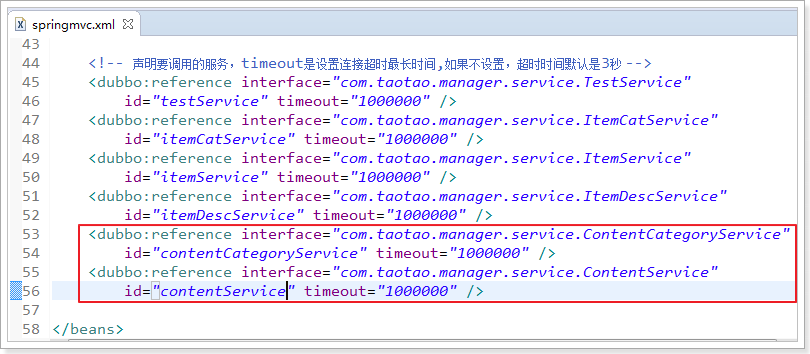
@Autowired

**private** ContentCategoryService contentCategoryService;

}

### 声明服务调用

在taotao-manager-web实现服务的调用

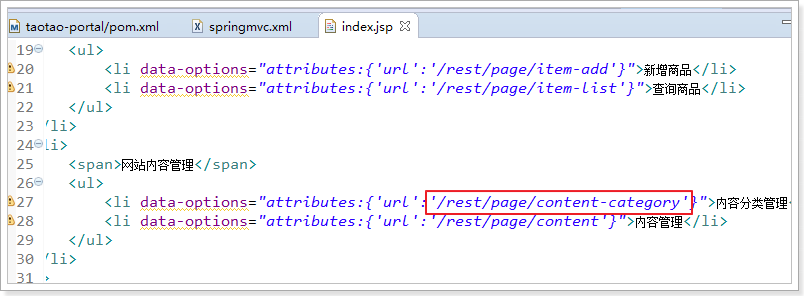


# 实现内容分类管理

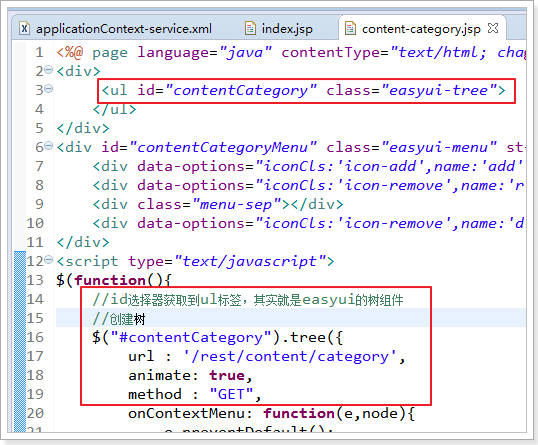
## 内容分类树创建

### 前端实现

查看index.jsp，确认内容分类管理的页面



content-category.jsp前端实现



### 后端实现

#### 编写controller

在taotao-manager-web

// url : '/rest/content/category',

// animate: true,

// method : "GET",

/\*\*

\* 前端树的初始化

\*

\* **@param** parentId

\* **@return**

\*/

@RequestMapping(method = RequestMethod.***GET***)

@ResponseBody

**public** List<ContentCategory> queryContentCategoryByParentId(

@RequestParam(value = "id", defaultValue = "0") Long parentId) {

// 调用服务查询

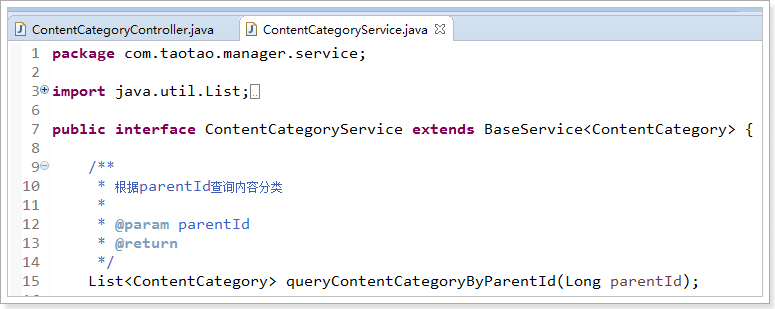
List<ContentCategory> list = **this**.contentCategoryService.queryContentCategoryByParentId(parentId);

**return** list;

}

#### 编写Service接口

在taotao-manager-inteface



#### 编写Service实现类

在taotao-manager-service

@Override

**public** List<ContentCategory> queryContentCategoryByParentId(Long parentId) {

// 设置查询条件

ContentCategory contentCategory = **new** ContentCategory();

contentCategory.setParentId(parentId);

// 查询

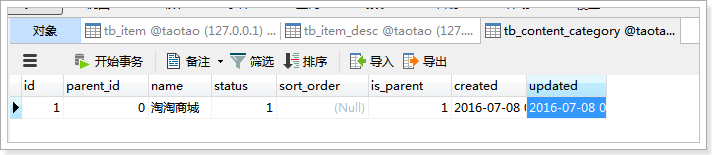
List<ContentCategory> list = **super**.queryListByWhere(contentCategory);

**return** list;

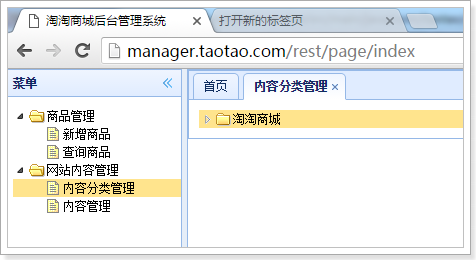
}

### 效果

数据库的内容分类表增加一条数据

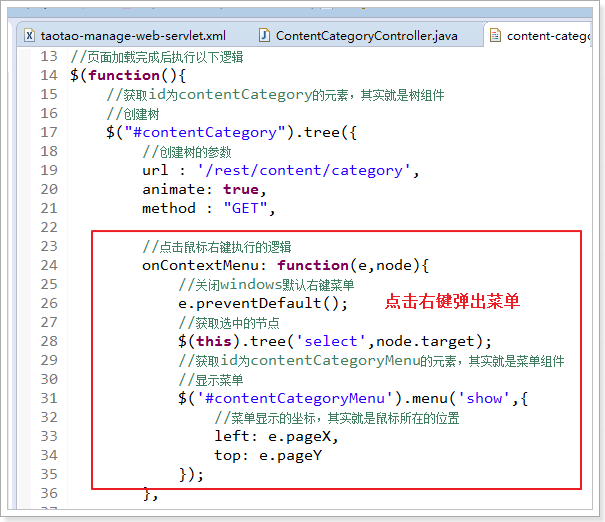


效果

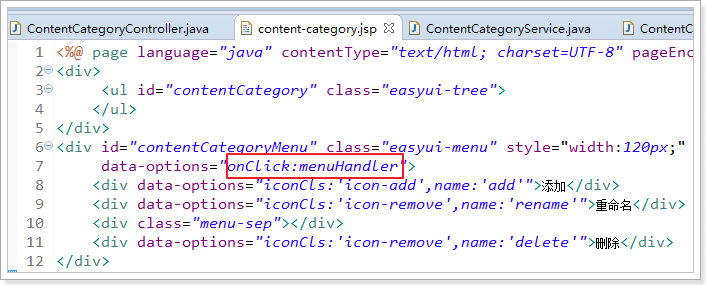


## 内容分类管理

### 前端实现



easyUI菜单组件。点击事件



菜单按钮点击事件

//菜单的点击事件，item就是选择的菜单项

**function** menuHandler(item){

//获取树

**var** tree = $("#contentCategory");

//获取选中的节点

**var** node = tree.tree("getSelected");

//通过判断菜单项的name属性，确定是什么操作

//添加操作 ===连同数据的类型也进行 "123"===123 结果是false

**if**(item.name === "add"){

//在树上追加一个节点

tree.tree('append', {

//追加节点的父

parent: (node?node.target:**null**),

//追加节点的数据

data: [{

text: '新建分类',

id : 0,

parentId : node.id

}]

});

//获取树上id为0的节点，其实就是获取新增的节点

**var** \_node = tree.tree('find',0);

//选中新增节点，开始编辑

tree.tree("select",\_node.target).tree('beginEdit',\_node.target);

//通过判断菜单项的name属性，确定是什么操作

//重命名操作

}**else** **if**(item.name === "rename"){

//对选中的节点开始编辑

tree.tree('beginEdit',node.target);

//通过判断菜单项的name属性，确定是什么操作

//删除操作

}**else** **if**(item.name === "delete"){

//提示用户确认删除，如果没有特殊要求，删除操作必须要提示，因为删除是危险操作

$.messager.confirm('确认','确定删除名为 '+node.text+' 的分类吗？',**function**(r){

//如果用户确认，就是性删除

**if**(r){

//url：/rest/content/category/delete?parentId=${节点的父id}&id=${节点的id}

$.ajax({

type: "POST",

url: "/rest/content/category/delete",

data : {parentId:node.parentId,id:node.id},

success: **function**(msg){

//$.messager.alert('提示','新增商品成功!');

//删除选中的节点

tree.tree("remove",node.target);

},

error: **function**(){

$.messager.alert('提示','删除失败!');

}

});

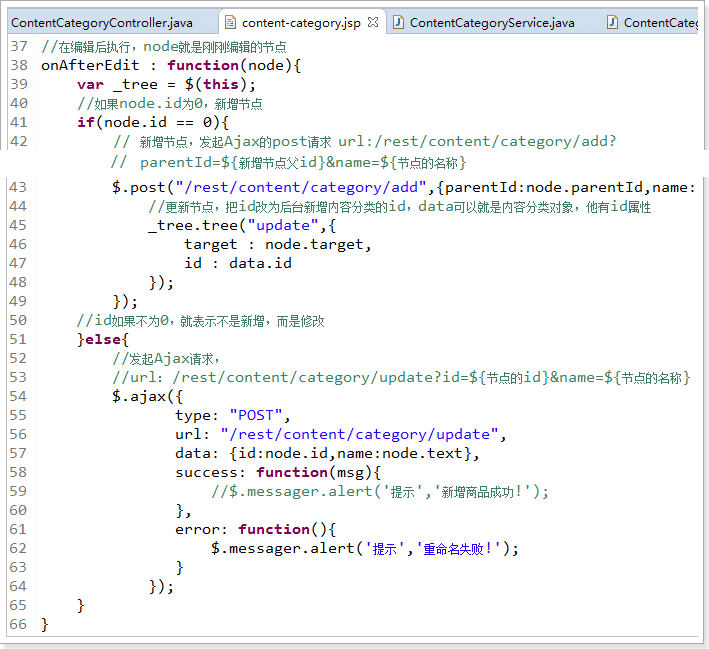
}

});

}

}

编辑之后执行逻辑,这里进行判断，做新增或者更新操作



### 后端实现

#### 实现controller

在taotao-manager-web实现

// // 新增节点，发起Ajax的post请求

// url:/rest/content/category/add?parentId=${新增节点父id}&name=${节点的名称}

// $.post("/rest/content/category/add",{parentId:node.parentId,name:node.text},function(data){

/\*\*

\* 新增

\*

\* **@param** contentCategory

\* **@return**

\*/

@RequestMapping(value = "add", method = RequestMethod.***POST***)

@ResponseBody

**public** ContentCategory saveContentCategory(ContentCategory contentCategory) {

// 调用服务保存

ContentCategory result = **this**.contentCategoryService.saveContentCategory(contentCategory);

**return** result;

}

// url： /rest/content/category/update?id=${节点的id}&name=${节点的名称}

// $.ajax({

// type: "POST",

// url: "/rest/content/category/update",

// data: {id:node.id,name:node.text},

// success: function(msg){

/\*\*

\* 更新

\*

\* **@param** contentCategory

\* **@return**

\*/

@RequestMapping(value = "update", method = RequestMethod.***POST***)

@ResponseBody

**public** String updateContentCategoryById(ContentCategory contentCategory) {

// 调用服务更新

**this**.contentCategoryService.updateByIdSelective(contentCategory);

**return** "200";

}

// url：/rest/content/category/delete?parentId=${节点的父id}&id=${节点的id}

// $.ajax({

// type: "POST",

// url: "/rest/content/category/delete",

/\*\*

\* 删除

\* **@param** parentId

\* **@param** id

\*/

@RequestMapping(value = "delete", method = RequestMethod.***POST***)

@ResponseBody

**public** String deleteContentCategoryById(Long parentId, Long id) {

// 调用服务删除

**this**.contentCategoryService.deleteContentCategoryById(parentId, id);

**return** "200";

}

#### 实现service接口

在taotao-manager-interface实现

/\*\*

\* 新增

\* **@param** contentCategory

\* **@return**

\*/

ContentCategory saveContentCategory(ContentCategory contentCategory);

/\*\*

\* 删除

\* **@param** parentId

\* **@param** id

\*/

**void** deleteContentCategoryById(Long parentId, Long id);

#### 实现service实现类

在taotao-manager-service实现

@Override

**public** ContentCategory saveContentCategory(ContentCategory contentCategory) {

// 保存内容分类

contentCategory.setIsParent(**false**);

contentCategory.setStatus(1);

**super**.save(contentCategory);

// 需要判断父节点的isParent 是否为true，如果不为true，修改为true

// 查询父节点

ContentCategory parent = **super**.queryById(contentCategory.getParentId());

// 判断父节点isParent是否为false

**if** (!parent.getIsParent()) {

// 修改父节点，保存

parent.setIsParent(**true**);

**super**.updateByIdSelective(parent);

}

**return** contentCategory;

}

@Override

**public** **void** deleteContentCategoryById(Long parentId, Long id) {

// 获取所有需要删除的节点id

List<Object> ids = **new** ArrayList<>();

// 把自己放进入

ids.add(id);

// 使用递归的方式获取节点的所有子节点

**this**.getIds(ids, id);

// 删除节点

**super**.deleteByIds(ids);

// 判断是否存在兄弟节点，如果没有兄弟节点，修改父节点的isParent为false

ContentCategory param = **new** ContentCategory();

param.setParentId(parentId);

Integer count = **super**.queryCountByWhere(param);

// 如果为0，没有兄弟节点，修改父节点的isParent为false

**if** (count == 0) {

ContentCategory parent = **new** ContentCategory();

parent.setId(parentId);

parent.setIsParent(**false**);

**super**.updateByIdSelective(parent);

}

}

// 递归获取id这个节点的所有的子节点

**private** **void** getIds(List<Object> ids, Long parentId) {

// 根据parentId查询子节点

ContentCategory param = **new** ContentCategory();

param.setParentId(parentId);

// 查询子节点

List<ContentCategory> list = **super**.queryListByWhere(param);

// 判断查询结果，其实就是判断是否有子节点

// 直接使用高级for循环，就不用判断

**for** (ContentCategory contentCategory : list) {

// 如果进入循环，表示有子节点，把子节点id放到ids中

ids.add(contentCategory.getId());

// 递归调用，获取子节点的子节点

**this**.getIds(ids, contentCategory.getId());

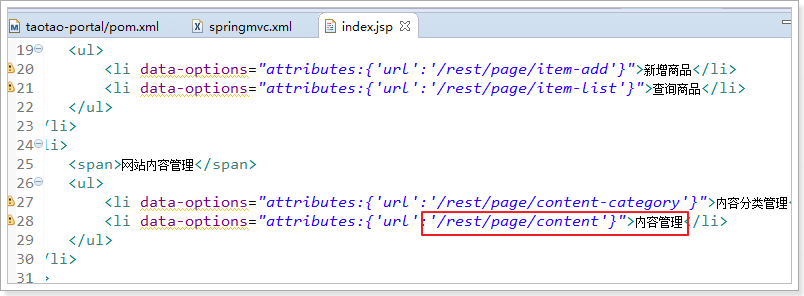
}

}

# 实现内容管理

## 前端实现

查看index.jsp，确认内容管理的页面是content.jsp



### 列表查询前端实现

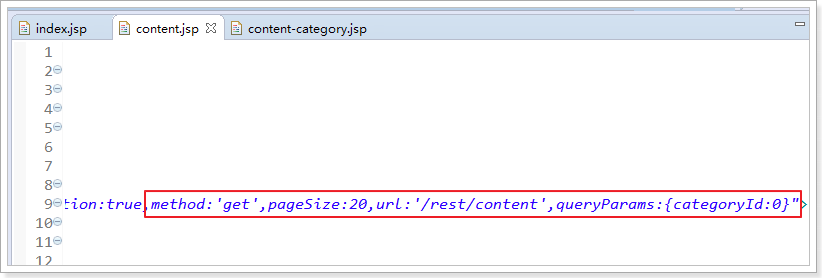
页面分成两个部分，

一个是树组件，已经实现

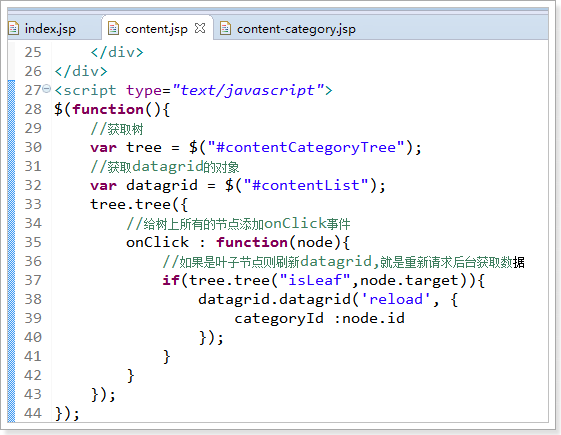
一个是datagrid组件需要实现



这是一个带参数的datagrid组件，每次发起请求都会携带categoryId这个参数，其实就是内容分类id，我们需要根据这个参数查询内容。

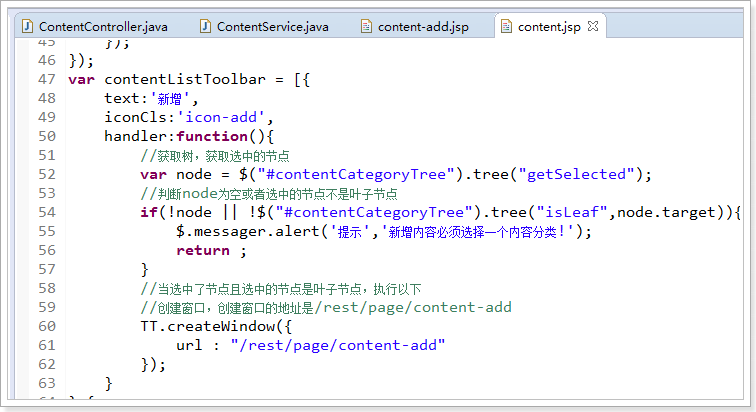


当点击树上的叶子节点，刷新datagrid



### 新增内容前端实现

必须先选中叶子节点，然后再点击新增按钮



content-add.jsp页面和商品新增页面基本上一样



初始化单图片上传组件（对应的图片上传后端逻辑已实现）

//单图片上传初始化

**function** initOnePicUpload(){

//获取上传按钮，绑定点击事件

$(".onePicUpload").click(**function**(){

//this就是按钮，获取同级的input元素

**var** input = $(**this**).siblings("input");

//加载单图片上传组件

KindEditor.editor(kingEditorParams).loadPlugin('image', **function**() {

**this**.plugin.imageDialog({

showRemote : **false**,

//点击“确定”按钮执行逻辑

clickFn : **function**(url, title, width, height, border, align) {

//获取同级的img标签，清除，其实就是回显前，清除原来的图片

input.parent().find("img").remove();

input.val(url);

//在input标签后添加a标签，里面是img标签，其实就是图片回显

input.after("<a href='"+url+"' target='\_blank'><img src='"+url+"' width='80' height='50'/></a>");

//关闭上传界面

**this**.hideDialog();

}

});

});

});

}

content-add.jsp提交逻辑

//提交逻辑

**function** submitForm(){

//校验

**if**(!$('#contentAddForm').form('validate')){

$.messager.alert('提示','表单还未填写完成!');

**return** ;

}

//编辑器的同步，把编辑器的内容同步到多行文本域中

contentAddEditor.sync();

//提交到后台的RESTful

$.ajax({

type: "POST",

url: "/rest/content",

data: $("#contentAddForm").serialize(),

success: **function**(msg){

$.messager.alert('提示','新增内容成功!');

//重新加载datagrid

$("#contentList").datagrid("reload");

TT.closeCurrentWindow();

},

error: **function**(){

$.messager.alert('提示','新增内容失败!');

}

});

}

## 后端实现

### 内容新增

在taotao-manager-web编写ContentController.

// type: "POST",

// url: "/rest/content",

// data: $("#contentAddForm").serialize(),

/\*\*

\* 新增

\*

\* **@param** content

\*/

@RequestMapping(method = RequestMethod.***POST***)

@ResponseBody

**public** **void** SaveContent(Content content) {

// 调用服务保存

**this**.contentService.save(content);

}

### 列表查询

#### 编写Controller

在taotao-manager-web编写ContentController.

//method:'get',pageSize:20,url:'/rest/content',queryParams:{categoryId:0}

/\*\*

\* 分页查询内容

\*

\* **@param** page

\* **@param** rows

\* **@param** categoryId

\* **@return**

\*/

@RequestMapping(method = RequestMethod.***GET***)

@ResponseBody

**public** TaoResult<Content> queryContentByPage(@RequestParam(value = "page", defaultValue = "1") Integer page,

@RequestParam(value = "rows", defaultValue = "20") Integer rows, Long categoryId) {

TaoResult<Content> taoUIResult = **this**.contentService.queryContentByPage(page, rows, categoryId);

**return** taoUIResult;

}

#### 编写Service接口

在taotao-manager-interface编写ContentService

/\*\*

\* 分页查询

\*

\* **@param** page

\* **@param** rows

\* **@param** categoryId

\* **@return**

\*/

TaoResult<Content> queryContentByPage(Integer page, Integer rows, Long categoryId);

#### 编写Service实现类

在taotao-manager-service编写ContentServiceImpl

@Override

**public** TaoResult<Content> queryContentByPage(Integer page, Integer rows, Long categoryId) {

Content param = **new** Content();

param.setCategoryId(categoryId);

TaoResult<Content> taoUIResult = **super**.queryListByPage(param, page, rows, "updated DESC");

**return** easyUIResult;

}

### 效果

