# 今日大纲

1. 实现首页的大广告位功能
2. 实现内容管理系统

# 首页的大广告

## 什么是大广告

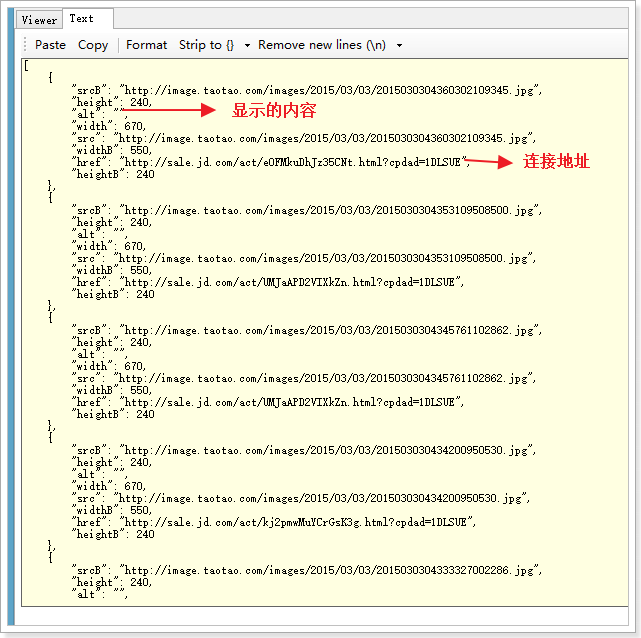


JS效果：

1. 点击下面的序号选择查询哪个广告
2. 自动切换
3. 点击图片查询具体的页面

以上是由前端团队来开发。

## 数据结构



说明：必须提供6条数据，才能显示效果。

## 如何实现？

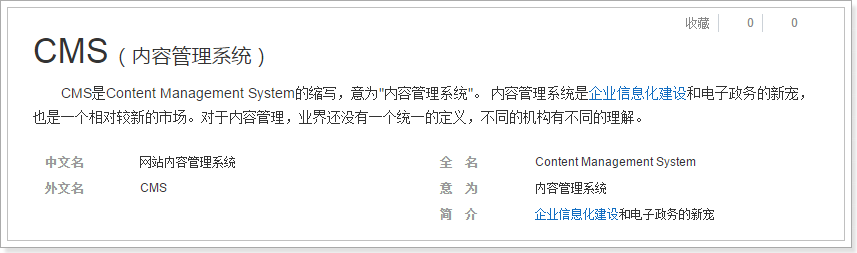
方案一：

1. 在后台系统中创建一张表，存储大广告位的广告数据
2. 在后台系统中对该表进行CRUD
3. 后台系统对外提供接口服务
4. 前台系统调用后台系统的提供的接口服务，即可获取到数据
5. 前台系统获取到数据后，封装成前端所需要的数据结构，功能即可实现

方案二：

1. 将首页显示的广告都抽象的看作是内容
2. 在后台系统中创建一张内容表
3. 创建一个内容分类表用于区分内容的分类
4. 后台系统对内容表以及分类表进行CRUD
5. 对外提供接口服务
6. 前端系统通过接口获取数据，进行封装，即可实现

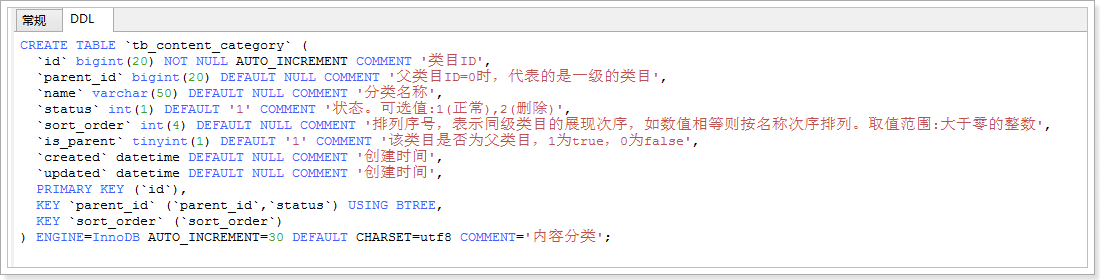
## 内容管理系统（CMS）



# 内容分类管理

## 表结构

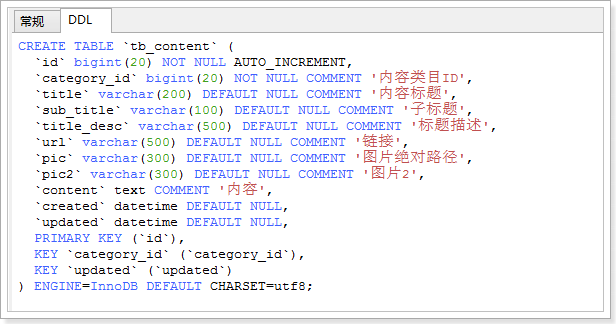
### 内容分类表



### 内容表

应该有的字段：

1. 图片
2. 连接
3. 标题
4. 子标题



表结构中字段是否添加索引判断依据是什么？ -- 字段是否是查询条件或者是排序条件。

是否将所有的字段都添加索引，来加快查询？ -- 不行的

1. 索引会占用存储空间，索引越多，使用的存储空间越多
2. 插入数据，存储索引也会消耗时间，索引越多，插入数据的速度越慢

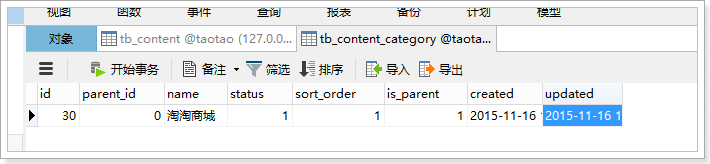
## 实现

### 创建pojo、mapper、service、controller

略。

### 内容分类管理

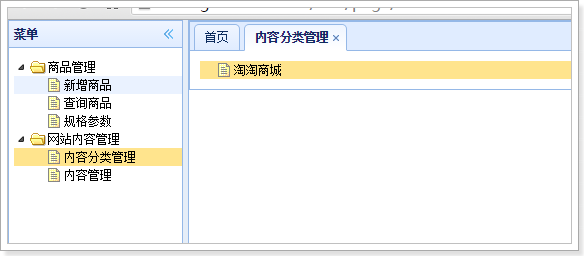
创建根节点：



### 分类列表查询

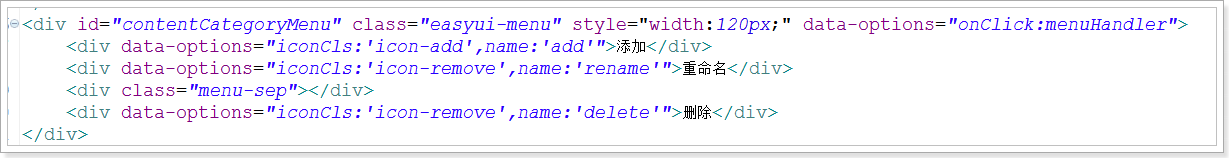


效果：

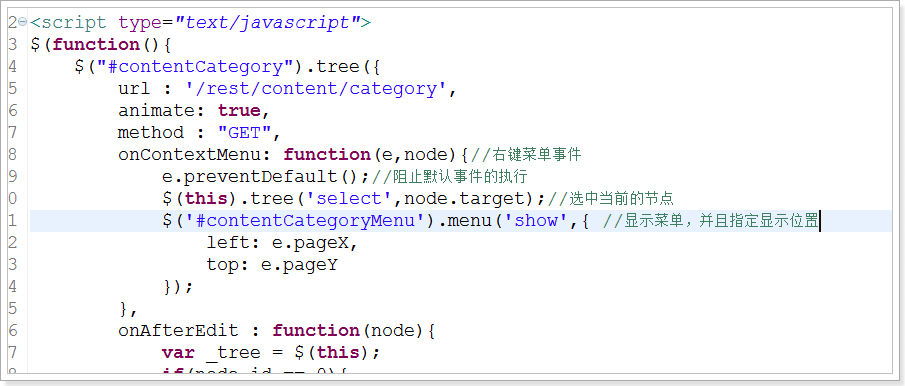


### 节点的右键菜单

定义菜单：

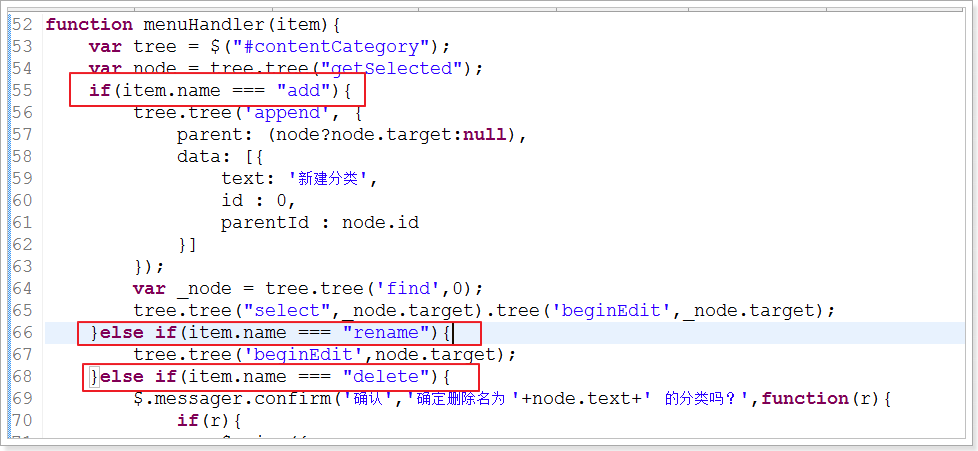


右键触发菜单显示:



具体的菜单事件：



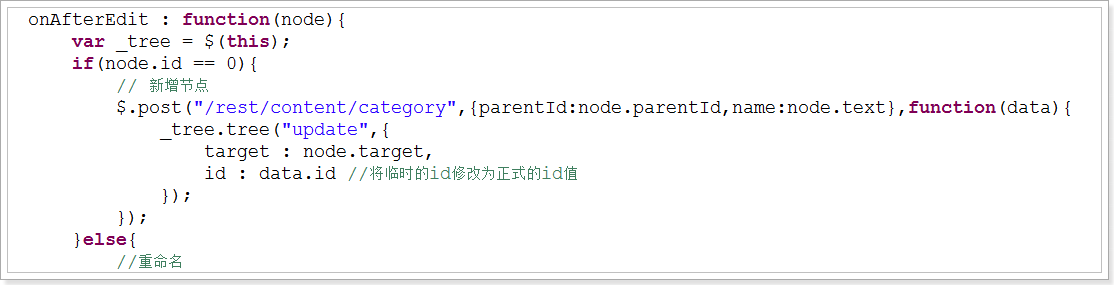


### 新增事件

JS实现：



按下回车，编辑完成事件：



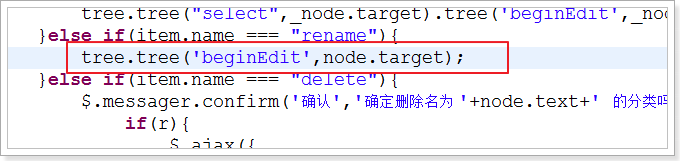
后台实现：

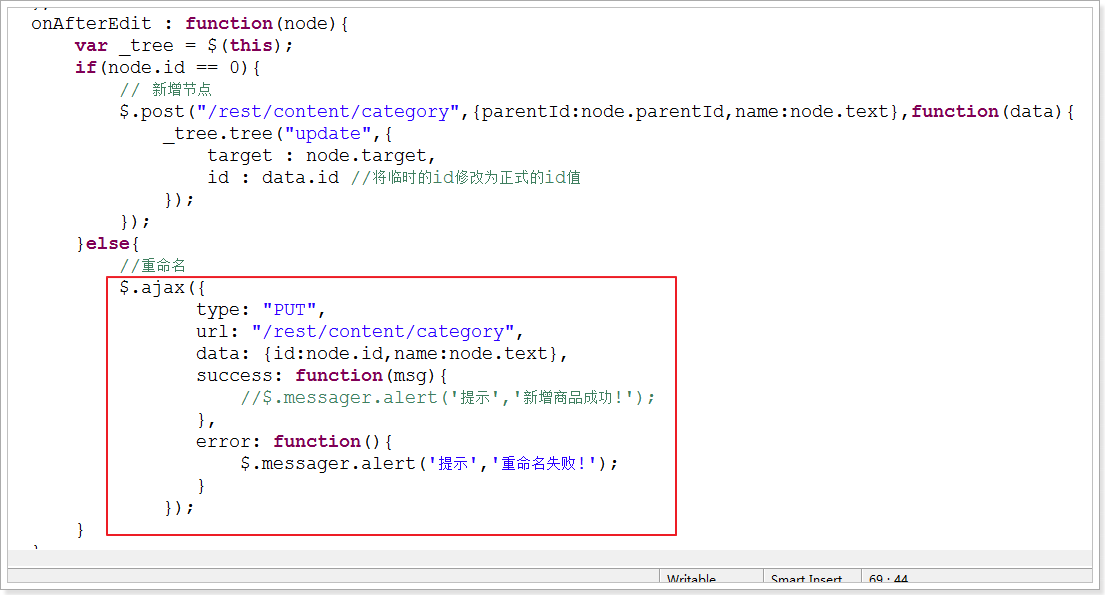
Controller：



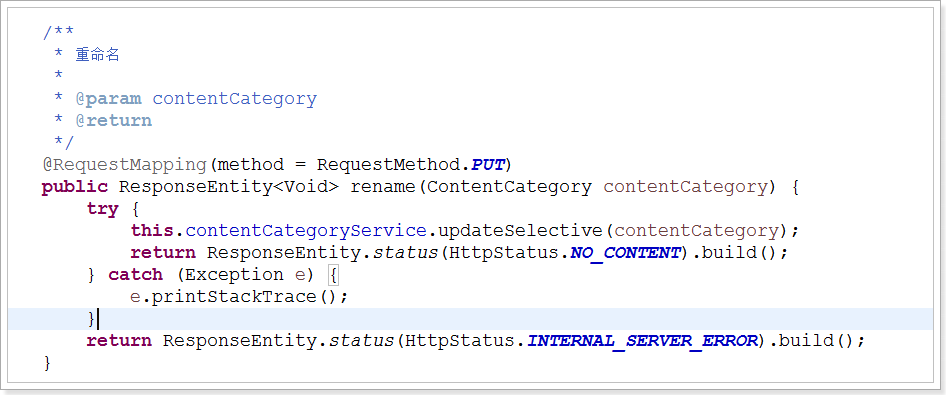
### 重命名

JS实现：





后端实现：



### 删除

JS实现：

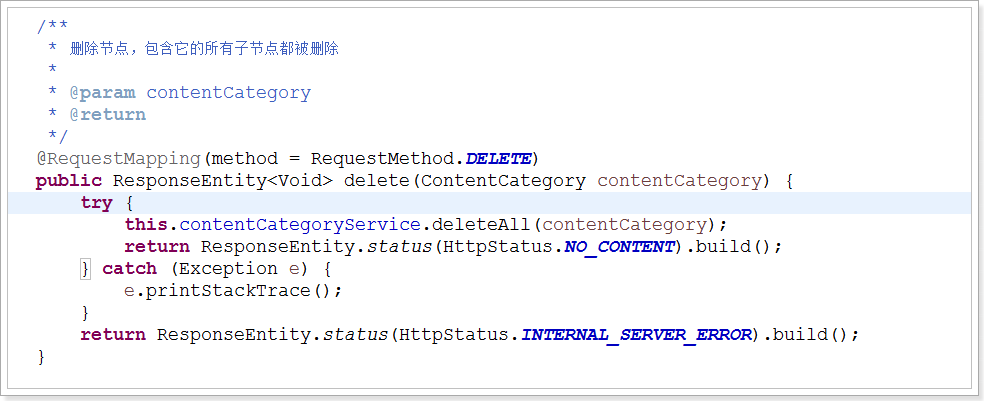


过滤器：



后端实现：

Controller：



Service：

**public** **void** deleteAll(ContentCategory contentCategory) {

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

ids.add(contentCategory.getId());

// 递归查找该节点下的所有子节点id

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

**super**.deleteByIds(ids, ContentCategory.**class**, "id");

// 判断该节点是否还有兄弟节点，如果没有，修改父节点的isParent为false

ContentCategory record = **new** ContentCategory();

record.setParentId(contentCategory.getParentId());

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

**if** (**null** == list || list.isEmpty()) {

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

parent.setId(contentCategory.getParentId());

parent.setIsParent(**false**);

**super**.updateSelective(parent);

}

}

**private** **void** findAllSubNode(List<Object> ids, Long pid) {

ContentCategory record = **new** ContentCategory();

record.setParentId(pid);

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

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

ids.add(contentCategory.getId());

// 判断该节点是否为父节点，如果是，继续调用该方法查找子节点

**if** (contentCategory.getIsParent()) {

// 开始递归

findAllSubNode(ids, contentCategory.getId());

}

}

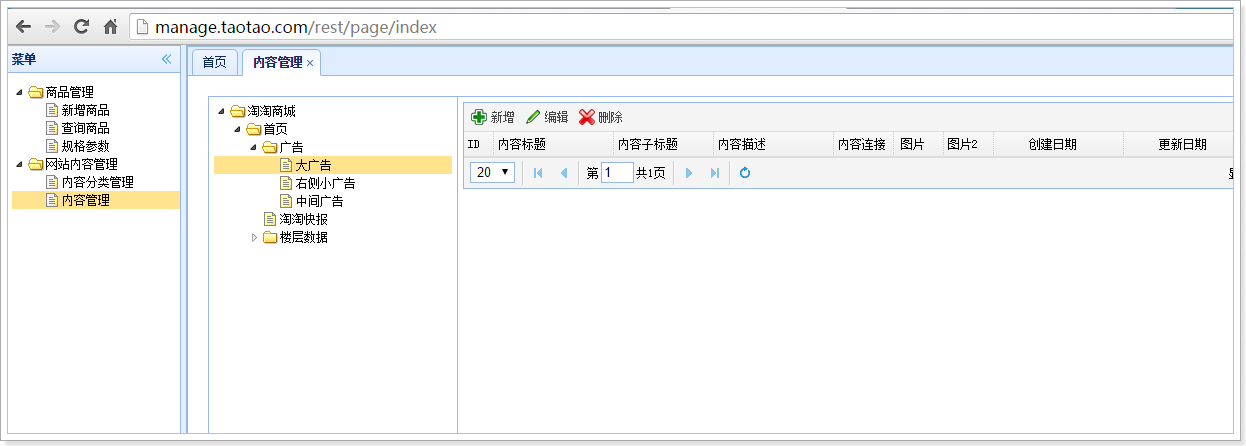
}

### 存在的问题（TODO）

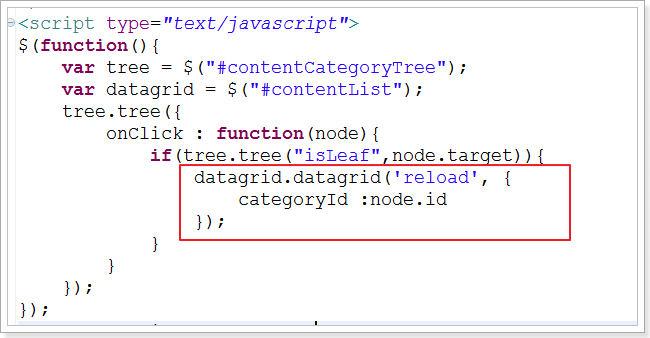
在Controller中实现数据库的操作，不在同一个事务中，需要将所有的操作移动至Service中完成。

# 内容管理

## 功能



## 选择内容分类加载数据

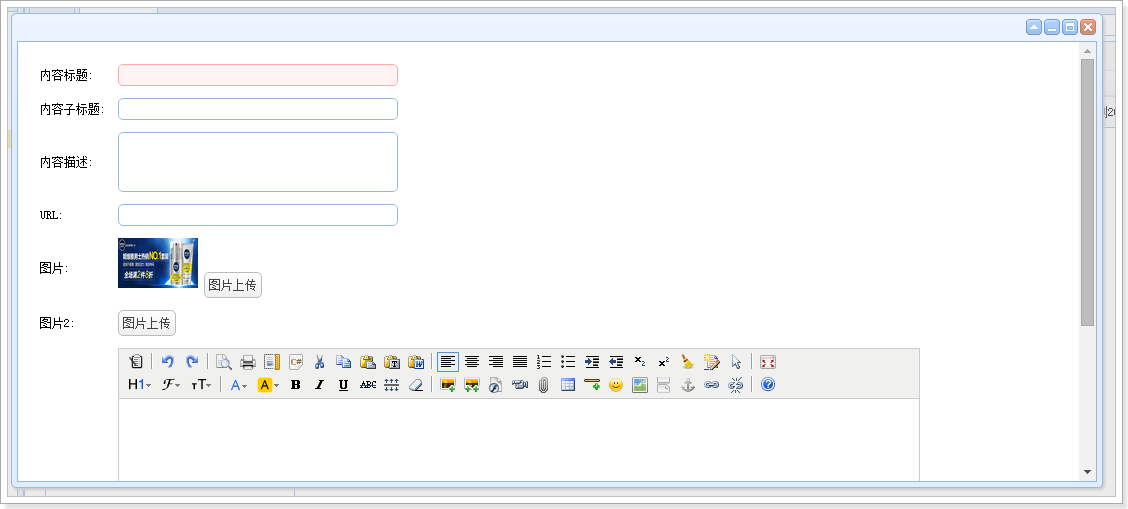


## 新增内容

校验，必须选中内容分类才能创建内容数据：



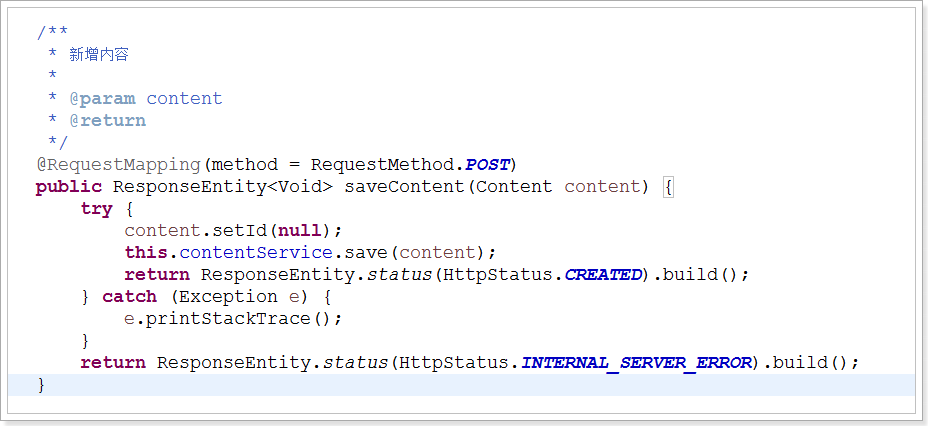
新增内容页面：



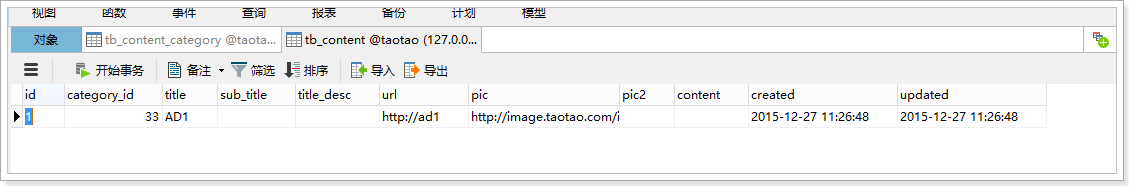
### 点击提交事件



### 后端实现



### 测试



## 查询内容列表

### JS实现



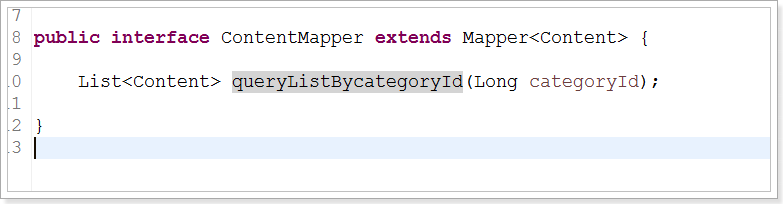
### Controller实现



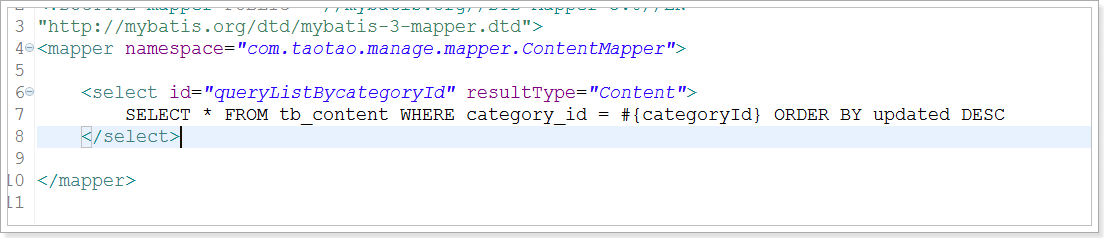
### Service



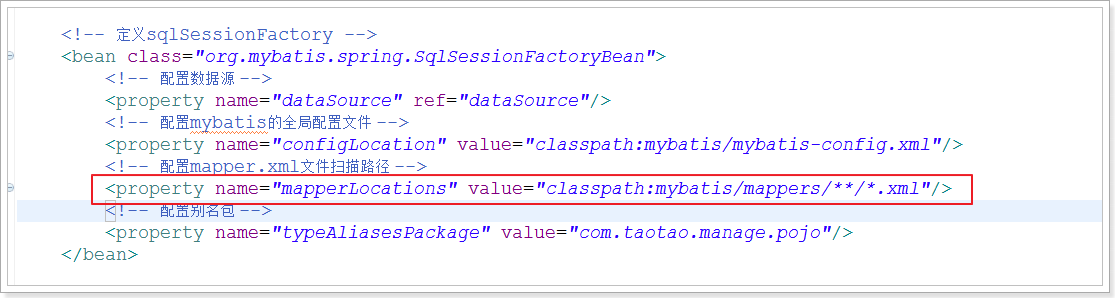
### Mapper



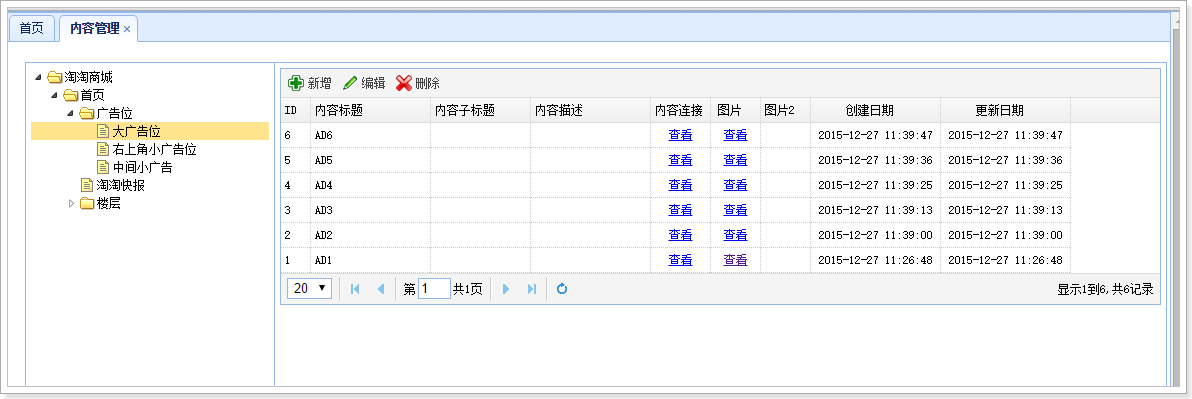
### Mapper.xml



### 在Spring和Mybatis的整合文件中读取Mapper.xml

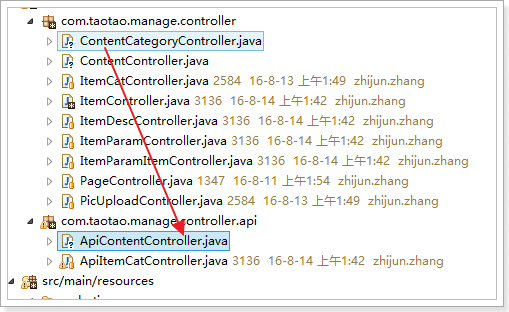


### 效果



## 对外的接口服务

将ContentController拷贝到api包下，对外提供接口服务：



实现：

@RequestMapping("api/content")

@Controller

**public** **class** ApiContentController {

@Autowired

**private** ContentService contentService;

/\*\*

\* 根据内容分类id查询分类列表

\*

\* **@param** categoryId

\* **@param** page

\* **@param** rows

\* **@return**

\*/

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

**public** ResponseEntity<EasyUIResult> queryListByCategoryId(@RequestParam("categoryId") Long categoryId,

@RequestParam(value = "page", defaultValue = "1") Integer page,

@RequestParam(value = "rows", defaultValue = "10") Integer rows) {

**try** {

EasyUIResult easyUIResult = **this**.contentService.queryListByCategoryId(categoryId, page, rows);

**return** ResponseEntity.*ok*(easyUIResult);

} **catch** (Exception e) {

e.printStackTrace();

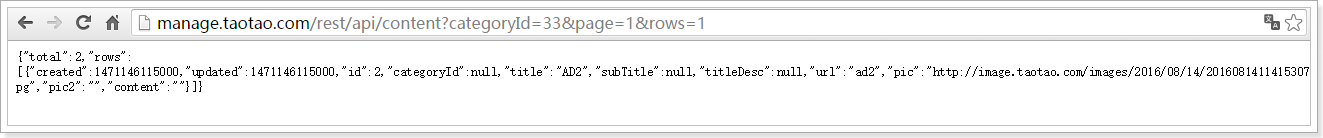
}

**return** ResponseEntity.*status*(HttpStatus.***INTERNAL\_SERVER\_ERROR***).body(**null**);

}

}

测试：



## 访问接口服务的方式

方式有2种：

1. js访问
   1. 有跨域 -- jsonp解决
   2. 无跨域 -- ajax解决
2. Java代码访问
   1. Httpclient

# Httpclient



## 导入依赖



## DoGET



## 带有参数的GET请求



## DoPOST



## 带有参数的POST请求

**public** **static** **void** main(String[] args) **throws** Exception {

// 创建Httpclient对象

CloseableHttpClient httpclient = HttpClients.*createDefault*();

// 创建http POST请求

HttpPost httpPost = **new** HttpPost("http://www.oschina.net/search");

// 伪装成浏览器

httpPost.setHeader("User-Agent", "Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/50.0.2661.94 Safari/537.36");

// 设置2个post参数，一个是scope、一个是q

List<NameValuePair> parameters = **new** ArrayList<NameValuePair>(0);

parameters.add(**new** BasicNameValuePair("scope", "project"));

parameters.add(**new** BasicNameValuePair("q", "java"));

parameters.add(**new** BasicNameValuePair("fromerr", "7nXH76r7"));

// 构造一个form表单式的实体

UrlEncodedFormEntity formEntity = **new** UrlEncodedFormEntity(parameters);

// 将请求实体设置到httpPost对象中

httpPost.setEntity(formEntity);

CloseableHttpResponse response = **null**;

**try** {

// 执行请求

response = httpclient.execute(httpPost);

// 判断返回状态是否为200

**if** (response.getStatusLine().getStatusCode() == 200) {

String content = EntityUtils.*toString*(response.getEntity(), "UTF-8");

System.***out***.println(content);

}

} **finally** {

**if** (response != **null**) {

response.close();

}

httpclient.close();

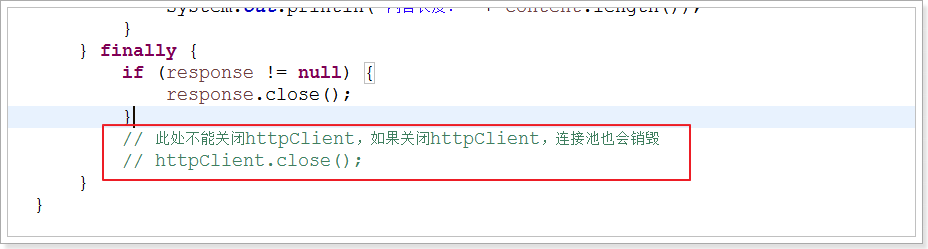
}

}

## 连接管理器



**注意**：



## 定期关闭无效连接

**public** **class** ClientEvictExpiredConnections {

**public** **static** **void** main(String[] args) **throws** Exception {

PoolingHttpClientConnectionManager cm = **new** PoolingHttpClientConnectionManager();

// 设置最大连接数

cm.setMaxTotal(200);

// 设置每个主机地址的并发数

cm.setDefaultMaxPerRoute(20);

**new** IdleConnectionEvictor(cm).start();

}

**public** **static** **class** IdleConnectionEvictor **extends** Thread {

**private** **final** HttpClientConnectionManager connMgr;

**private** **volatile** **boolean** shutdown;

**public** IdleConnectionEvictor(HttpClientConnectionManager connMgr) {

**this**.connMgr = connMgr;

}

@Override

**public** **void** run() {

**try** {

**while** (!shutdown) {

**synchronized** (**this**) {

wait(5000);

// 关闭失效的连接

connMgr.closeExpiredConnections();

}

}

} **catch** (InterruptedException ex) {

// 结束

}

}

**public** **void** shutdown() {

shutdown = **true**;

**synchronized** (**this**) {

notifyAll();

}

}

}

}

## 设置请求参数



# Httpclient和Spring的整合

<beans xmlns=*"http://www.springframework.org/schema/beans"*

xmlns:context=*"http://www.springframework.org/schema/context"* xmlns:p=*"http://www.springframework.org/schema/p"*

xmlns:aop=*"http://www.springframework.org/schema/aop"* xmlns:tx=*"http://www.springframework.org/schema/tx"*

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-4.0.xsd*

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

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

*http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spring-util-4.0.xsd"*>

<!-- 定义连接管理器 -->

<bean id=*"httpClientConnectionManager"*

class=*"org.apache.http.impl.conn.PoolingHttpClientConnectionManager"*>

<property name=*"maxTotal"* value=*"${http.maxTotal}"* />

<property name=*"defaultMaxPerRoute"* value=*"${http.defaultMaxPerRoute}"* />

</bean>

<!-- httpclient的构建器 -->

<bean id=*"httpClientBuilder"* class=*"org.apache.http.impl.client.HttpClientBuilder"*>

<property name=*"connectionManager"* ref=*"httpClientConnectionManager"* />

</bean>

<!-- 定义Httpclient对象 -->

<!-- 该对象是多例的 -->

<bean class=*"org.apache.http.impl.client.CloseableHttpClient"*

factory-bean=*"httpClientBuilder"* factory-method=*"build"* scope=*"prototype"*>

</bean>

<!-- 请求参数的构建器 -->

<bean id=*"requestConfigBuilder"* class=*"org.apache.http.client.config.RequestConfig.Builder"*>

<!-- 创建连接的最长时间 -->

<property name=*"connectTimeout"* value=*"${http.connectTimeout}"* />

<!-- 从连接池中获取到连接的最长时间 -->

<property name=*"connectionRequestTimeout"* value=*"${http.connectionRequestTimeout}"* />

<!-- 数据传输的最长时间 -->

<property name=*"socketTimeout"* value=*"${http.socketTimeout}"* />

<!-- 提交请求前测试连接是否可用 -->

<property name=*"staleConnectionCheckEnabled"* value=*"${http.staleConnectionCheckEnabled}"* />

</bean>

<!-- 定义请求参数对象 -->

<bean class=*"org.apache.http.client.config.RequestConfig"*

factory-bean=*"requestConfigBuilder"* factory-method=*"build"* />

<!-- 定期关闭无效连接 -->

<bean class=*"com.taotao.web.httpclient.IdleConnectionEvictor"*>

<constructor-arg index=*"0"* ref=*"httpClientConnectionManager"* />

</bean>

</beans>

# 封装ApiService

**package** com.taotao.web.service;

**import** java.io.IOException;

**import** java.net.URISyntaxException;

**import** java.util.ArrayList;

**import** java.util.List;

**import** java.util.Map;

**import** org.apache.http.NameValuePair;

**import** org.apache.http.ParseException;

**import** org.apache.http.client.ClientProtocolException;

**import** org.apache.http.client.config.RequestConfig;

**import** org.apache.http.client.entity.UrlEncodedFormEntity;

**import** org.apache.http.client.methods.CloseableHttpResponse;

**import** org.apache.http.client.methods.HttpGet;

**import** org.apache.http.client.methods.HttpPost;

**import** org.apache.http.client.utils.URIBuilder;

**import** org.apache.http.impl.client.CloseableHttpClient;

**import** org.apache.http.message.BasicNameValuePair;

**import** org.apache.http.util.EntityUtils;

**import** org.springframework.beans.BeansException;

**import** org.springframework.beans.factory.BeanFactory;

**import** org.springframework.beans.factory.BeanFactoryAware;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.stereotype.Service;

**import** com.taotao.web.httpclient.HttpResult;

@Service

**public** **class** ApiService **implements** BeanFactoryAware{

@Autowired

**private** RequestConfig requestConfig;

**private** BeanFactory beanFactory;

/\*\*

\* 指定GET请求，返回:null,请求失败，String数据，请求成功

\*

\* **@param** url

\* **@return**

\* **@throws** ClientProtocolException

\* **@throws** IOException

\*/

**public** String doGet(String url) **throws** ClientProtocolException, IOException {

// 创建http GET请求

HttpGet httpGet = **new** HttpGet(url);

httpGet.setConfig(requestConfig);

CloseableHttpResponse response = **null**;

**try** {

// 执行请求

response = getHttpClient().execute(httpGet);

// 判断返回状态是否为200

**if** (response.getStatusLine().getStatusCode() == 200) {

**return** EntityUtils.*toString*(response.getEntity(), "UTF-8");

}

} **finally** {

**if** (response != **null**) {

response.close();

}

}

**return** **null**;

}

/\*\*

\* 带有参数的GET请求，返回:null,请求失败，String数据，请求成功

\*

\* **@param** url

\* **@param** params

\* **@return**

\* **@throws** ClientProtocolException

\* **@throws** IOException

\* **@throws** URISyntaxException

\*/

**public** String doGet(String url, Map<String, String> params) **throws** ClientProtocolException, IOException,

URISyntaxException {

URIBuilder builder = **new** URIBuilder(url);

**for** (Map.Entry<String, String> entry : params.entrySet()) {

builder.setParameter(entry.getKey(), entry.getValue());

}

**return** **this**.doGet(builder.build().toString());

}

/\*\*

\* 带有参数的POST请求

\*

\* **@param** url

\* **@param** params

\* **@return**

\* **@throws** ParseException

\* **@throws** IOException

\*/

**public** HttpResult doPost(String url, Map<String, String> params) **throws** ParseException, IOException {

// 创建http POST请求

HttpPost httpPost = **new** HttpPost(url);

httpPost.setConfig(requestConfig);

**if** (**null** != params) {

// 设置post参

List<NameValuePair> parameters = **new** ArrayList<NameValuePair>(0);

**for** (Map.Entry<String, String> entry : params.entrySet()) {

parameters.add(**new** BasicNameValuePair(entry.getKey(), entry.getValue()));

}

// 构造一个form表单式的实体

UrlEncodedFormEntity formEntity = **new** UrlEncodedFormEntity(parameters);

// 将请求实体设置到httpPost对象中

httpPost.setEntity(formEntity);

}

CloseableHttpResponse response = **null**;

**try** {

// 执行请求

response = getHttpClient().execute(httpPost);

**return** **new** HttpResult(response.getStatusLine().getStatusCode(), EntityUtils.*toString*(

response.getEntity(), "UTF-8"));

} **finally** {

**if** (response != **null**) {

response.close();

}

}

}

/\*\*

\* 没有参数的POST请求

\*

\* **@param** url

\* **@return**

\* **@throws** ParseException

\* **@throws** IOException

\*/

**public** HttpResult doPost(String url) **throws** ParseException, IOException {

**return** **this**.doPost(url, **null**);

}

**private** CloseableHttpClient getHttpClient(){

//通过Bean工厂获取bean，保证HttpClient对象是多例

**return** **this**.beanFactory.getBean(CloseableHttpClient.**class**);

}

@Override

**public** **void** setBeanFactory(BeanFactory beanFactory) **throws** BeansException {

**this**.beanFactory = beanFactory;

}

}

# 实现大广告

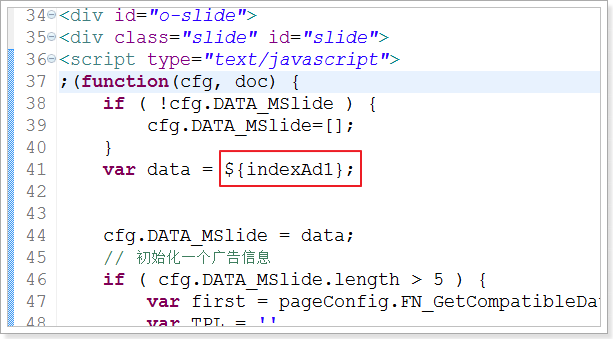
## 访问后台系统接口获取数据



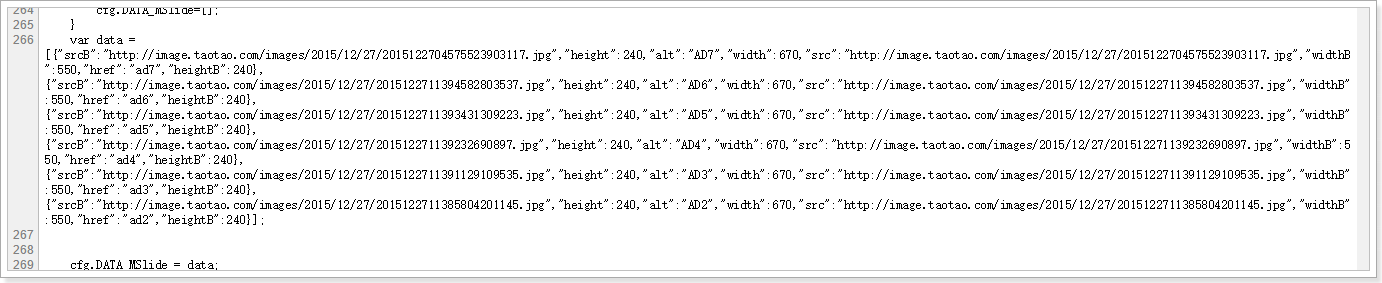
## 将json数据保存到模型数据中



## 通过EL表达式输出到页面



## 测试



效果：



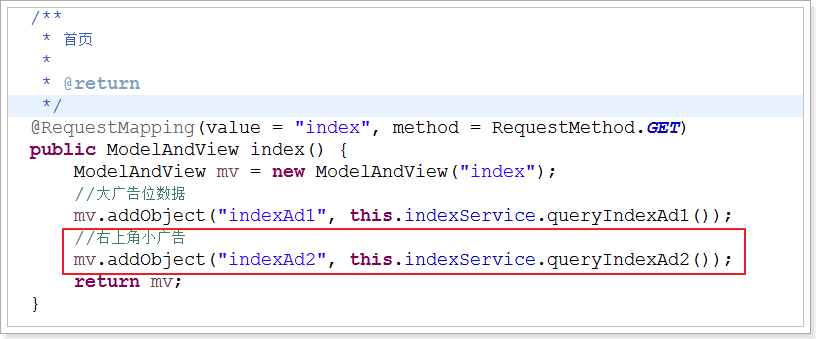
# 实现小广告

## 分析

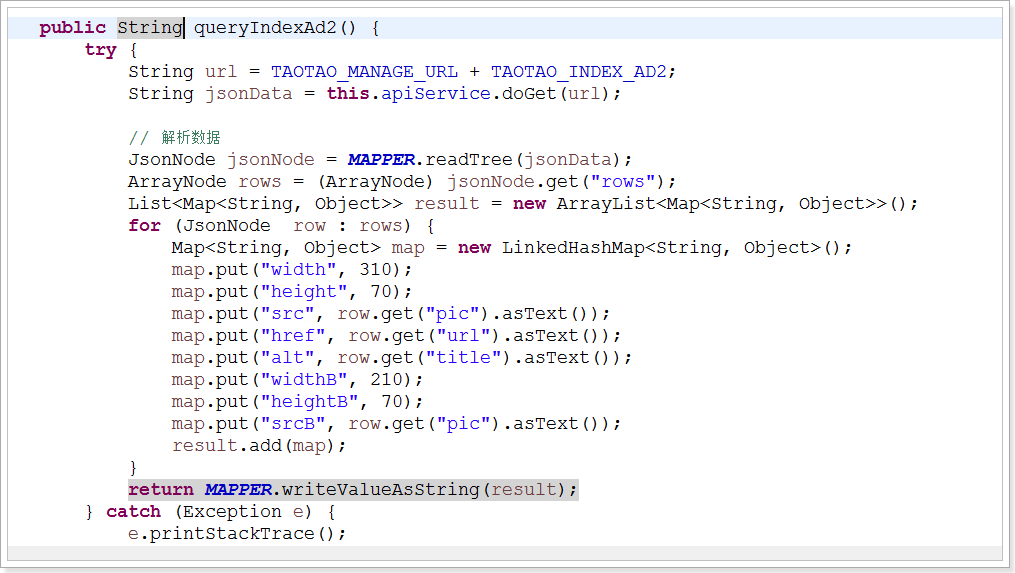


## 实现

### Controller



### Service



### 效果



作业：实现淘淘快报。

## 优化实现

