# 内容管理系统

## 内容复习

1. 前台系统的搭建
   1. 服务层
   2. 表现层
2. 首页的展示
3. 商品类目展示
   1. 使用ajax跨域调用服务。
   2. 使用jsonp实现

## 课程计划

1. 首页的内容展示：轮播图展示。
2. Cms系统的实现。Content Management System"的缩写,意为"内容管理系统
3. Taotao-rest发布服务。
4. 表现层调用服务，展示轮播图。

## 首页轮播图展示

### 首页内容管理思路

首页展示的内容都应该可以通过后台来管理。

1. 内容的分组
2. 分组下还可以有小组
3. 广告项目中有图片
4. 标题
5. 价格
6. 链接
7. 内容

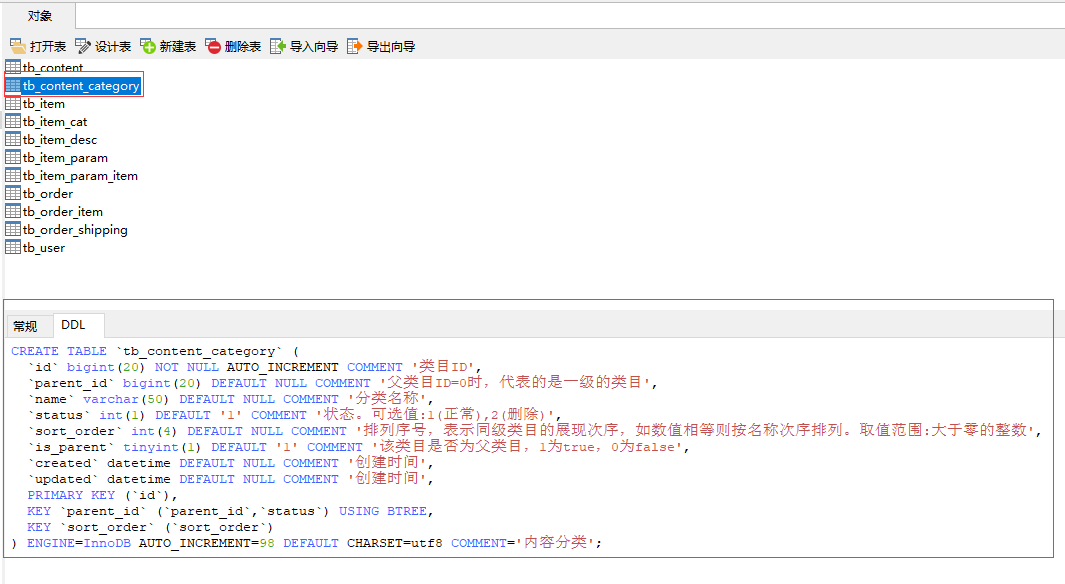
第一块：内容的分组管理，应该是一个树形结构。

第二块：内容管理。

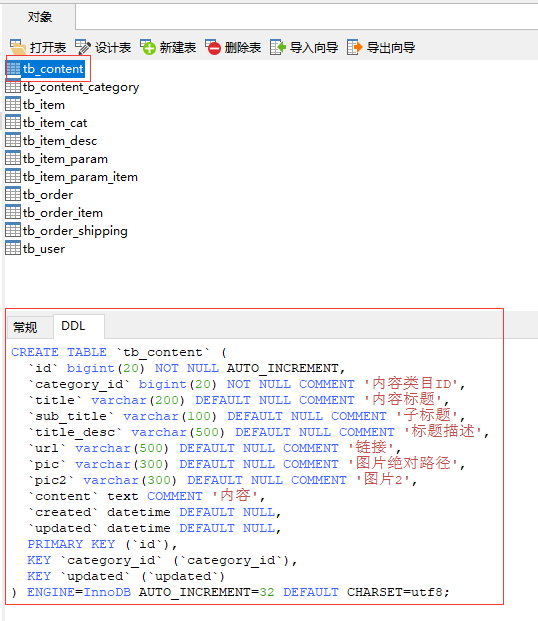
图片、链接、标题、价格、内容。

使用两个表来保存内容数据：

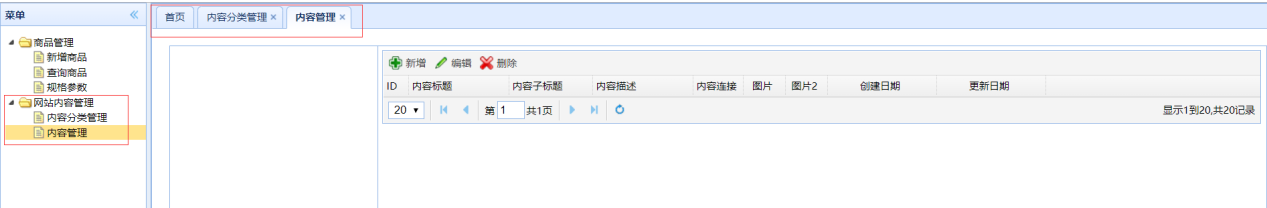
1. 内容分类表：



1. 内容表：



1. CMS系统：内容管理系统



## 内容分类管理

### 展示分类列表

#### 分析



初始化easyUI的tree控件url：content/category/list

请求的参数：id，父节点id

返回结果：json数据（EasyUITreeNode列表）

是一个列表，列表中每个元素包含三个属性：

1. Id
2. Text
3. state

#### Dao层

从tb\_content\_category表中取数据，根据parentid查询分类列表。可以使用逆向工程生成的代码。

#### Service层

接收一个parentId，根据parentID查询节点列表。创建一个EasyUITreeNode列表。返回。

参数：

Long parentId

返回值：

List<EasyUITreeNode>

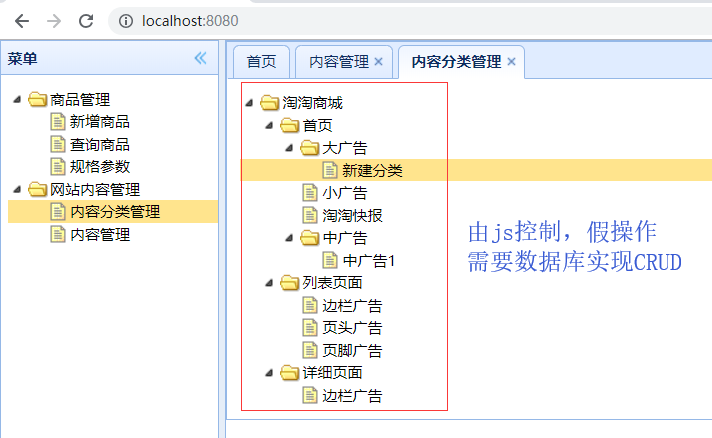
|  |
| --- |
| **public** **interface** ContentCatgoryService {  **public** List<EasyUITreeNode> getContentCatList(Long parentId) ;  }  // ServiceImpl:  @Service  **public** **class** ContentCatgoryServiceImpl **implements** ContentCatgoryService {  @Autowired  **private** TbContentCategoryMapper contentCategoryMapper;  @Override  **public** List<EasyUITreeNode> getContentCatList(Long parentId) {  //根据parentId查询子节点列表  TbContentCategoryExample example = **new** TbContentCategoryExample();  Criteria criteria = example.createCriteria();  criteria.andParentIdEqualTo(parentId);  //执行查询  List<TbContentCategory> list = contentCategoryMapper.selectByExample(example);  //转换成EasyUITreeNode列表  List<EasyUITreeNode> resultList = **new** ArrayList<>();  **for** (TbContentCategory tbContentCategory : list) {  //创建一EasyUITreeNode节点  EasyUITreeNode node = **new** EasyUITreeNode();  node.setId(tbContentCategory.getId());  node.setText(tbContentCategory.getName());  node.setState(tbContentCategory.getIsParent()?"closed":"open");  //添加到列表  resultList.add(node);  }  **return** resultList;  }  } |

#### Controller层

接收parentId调用Service查询节点列表，返回节点列表。返回json数据，需要使用@ResponseBody

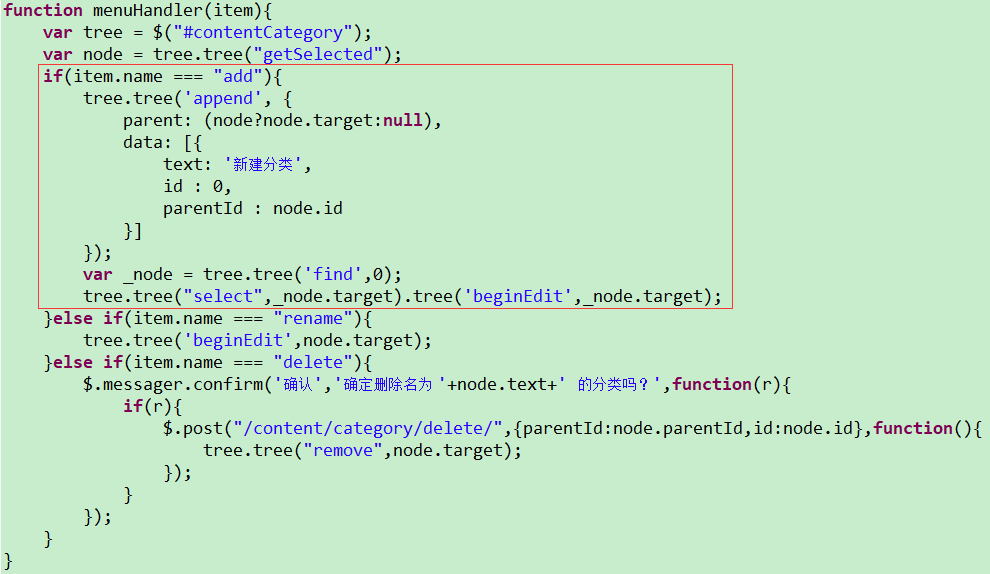
|  |
| --- |
| @Controller  @RequestMapping("/content/category")  **public** **class** ContentCategoryController {  @Autowired  **private** ContentCatgoryService contentCatgoryService;  @RequestMapping("/list")  @ResponseBody  **public** List<EasyUITreeNode> getContentCatList(@RequestParam(value="id", defaultValue="0")Long parentId) {  List<EasyUITreeNode> list = contentCatgoryService.getContentCatList(parentId);  **return** list;  }  } |

重启taotao-manager测试：



### 新增节点

#### 分析



光标离开事件：



请求的url：/content/category/create

请求的参数：parentId、name

返回结果：返回TaotaoResult，包含新添加记录的id。

#### Dao层

插入内容分类数据可以使用逆向工程实现

#### Service层

1、接收两个参数：parentId、name

2、创建一个对应tb\_content\_category表的pojo对象。

3、补全pojo对象的属性。

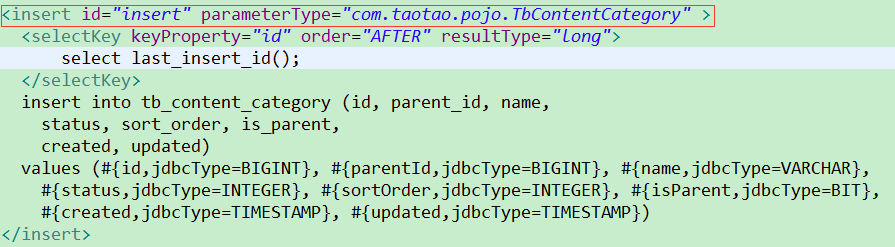
4、插入数据

5、需要mybatis返回主键。

6、使用TaotaoResult包装id，返回。

|  |
| --- |
| @Override  **public** TaotaoResult insertCatgory(Long parentId, String name) {  //创建一个pojo对象  TbContentCategory contentCategory = **new** TbContentCategory();  contentCategory.setName(name);  contentCategory.setParentId(parentId);  //1(正常),2(删除)  contentCategory.setStatus(1);  contentCategory.setIsParent(**false**);  //'排列序号，表示同级类目的展现次序，如数值相等则按名称次序排列。取值范围:大于零的整数'  contentCategory.setSortOrder(1);  contentCategory.setCreated(**new** Date());  contentCategory.setUpdated(**new** Date());  //插入数据  contentCategoryMapper.insert(contentCategory);  //取返回的主键  Long id = contentCategory.getId();  //判断父节点的isparent属性  //查询父节点  TbContentCategory parentNode = contentCategoryMapper.selectByPrimaryKey(parentId);  **if** (!parentNode.getIsParent()) {  parentNode.setIsParent(**true**);  //更新父节点  contentCategoryMapper.updateByPrimaryKey(parentNode);  }  //返回主键  **return** TaotaoResult.*ok*(id);  } |

数据库插入操作，返回主键id：TbContentCategoryMapper.xml



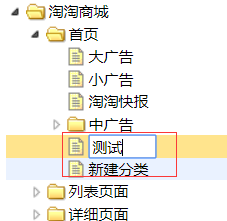
#### Controller

接收两个参数，parentId、name。调用Service插入数据，返回TaotaoResult。返回json格式的数据。

|  |
| --- |
| @RequestMapping("/create")  @ResponseBody  **public** TaotaoResult createNode(Long parentId, String name) {  TaotaoResult result = contentCatgoryService.insertCatgory(parentId, name);  **return** result;  } |

测试：可能存在的问题分析

问题1：如果连续插入两条数据，发现第二次插入数据的时候焦点却在第一次插入的数据处，从而导致数据无法正常插入，因此此处考虑可能是重复id导致，查看相应的jsp文件，修改如下：（此外要考虑新增分类的归属问题，看其是否存在父节点）





### 重命名节点、删除节点

#### dao层

Dao层采用逆向工程生成的mapper文件

#### Service层

// 重命名内容分类（修改）

**public** TaotaoResult updateCatgory(Long id, String name);

// 根据id删除内容分类（考虑删除的是父节点还是子节点，根据不同的情况处理）

**public** TaotaoResult deleteCatgory(Long id);

ServiceImpl：

@Service

**public** **class** ContentCatgoryServiceImpl **implements** ContentCatgoryService {

@Autowired

**private** TbContentCategoryMapper contentCategoryMapper;

@Override

**public** List<EasyUITreeNode> getContentCatList(Long parentId) {

// 根据parentId查询子节点列表

TbContentCategoryExample example = **new** TbContentCategoryExample();

Criteria criteria = example.createCriteria();

criteria.andParentIdEqualTo(parentId);

// 执行查询

List<TbContentCategory> list = contentCategoryMapper.selectByExample(example);

// 转换成EasyUITreeNode列表

List<EasyUITreeNode> resultList = **new** ArrayList<>();

**for** (TbContentCategory tbContentCategory : list) {

// 创建一EasyUITreeNode节点

EasyUITreeNode node = **new** EasyUITreeNode();

node.setId(tbContentCategory.getId());

node.setText(tbContentCategory.getName());

node.setState(tbContentCategory.getIsParent() ? "closed" : "open");

// 添加到列表

resultList.add(node);

}

**return** resultList;

}

@Override

**public** TaotaoResult insertCatgory(Long parentId, String name) {

// 创建一个pojo对象

TbContentCategory contentCategory = **new** TbContentCategory();

contentCategory.setName(name);

contentCategory.setParentId(parentId);

// 1(正常),2(删除)

contentCategory.setStatus(1);

contentCategory.setIsParent(**false**);

// '排列序号，表示同级类目的展现次序，如数值相等则按名称次序排列。取值范围:大于零的整数'

contentCategory.setSortOrder(1);

contentCategory.setCreated(**new** Date());

contentCategory.setUpdated(**new** Date());

// 插入数据

contentCategoryMapper.insert(contentCategory);

// 取返回的主键

Long id = contentCategory.getId();

// 判断父节点的isparent属性

// 查询父节点

TbContentCategory parentNode = contentCategoryMapper.selectByPrimaryKey(parentId);

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

parentNode.setIsParent(**true**);

// 更新父节点

contentCategoryMapper.updateByPrimaryKey(parentNode);

}

// 返回主键

**return** TaotaoResult.*ok*(id);

}

@Override

**public** TaotaoResult updateCatgory(Long id, String name) {

// 通过id查询节点对象

TbContentCategory contentCategory = contentCategoryMapper.selectByPrimaryKey(id);

// 判断新的name值与原来的值是否相同，如果相同则不用更新

**if** (name != **null** && name.equals(contentCategory.getName())) {

**return** TaotaoResult.*ok*();

}

contentCategory.setName(name);

// 设置更新时间

contentCategory.setUpdated(**new** Date());

// 更新数据库

contentCategoryMapper.updateByPrimaryKey(contentCategory);

// 返回结果

**return** TaotaoResult.*ok*();

}

@Override

**public** TaotaoResult deleteCatgory(Long id) {

// 首先要判断当前删除的节点是叶子节点还是父节点，要依次递归删除

// 删除分类，就是改节点的状态为2(为了避免数据冗余，选择硬删除数据)

TbContentCategory contentCategory = contentCategoryMapper.selectByPrimaryKey(id);

// 硬删除数据

contentCategoryMapper.deleteByPrimaryKey(id);

/\* 状态:可选值:1(正常),2(删除),软删除数据

contentCategory.setStatus(2);

contentCategoryMapper.updateByPrimaryKey(contentCategory);\*/

// 需要判断一下要删除的这个节点是否是父节点，如果是父节点，那么就级联

// 删除这个父节点下的所有子节点（采用递归的方式删除）

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

deleteNode(contentCategory.getId());

}

// 需要判断父节点当前还有没有子节点，如果有子节点就不用做修改

// 如果父节点没有子节点了，那么要修改父节点的isParent属性为false即变为叶子节点

TbContentCategory parent = contentCategoryMapper.selectByPrimaryKey(contentCategory.getParentId());

List<TbContentCategory> list = getContentCategoryListByParentId(parent.getId());

/\*\*

\* 判断父节点是否有子节点是判断这个父节点下的所有子节点的状态，

\* 如果是软删除，则依次判断每个子节点的状态，如果状态都是2就说明

\* 所有子节点都被“删除”，则需要修改当前节点的状态

\* 如果是硬删除，则只需要判断当前子节点列表是否为空即可

\*/

/\* boolean flag = false;

for (TbContentCategory tbContentCategory : list) {

if (tbContentCategory.getStatus() == 0) {

flag = true;

break;

}

}

// 如果没有子节点了

if (!flag) {

parent.setIsParent(false);

contentCategoryMapper.updateByPrimaryKey(parent);

}\*/

// 此处硬删除判断是否为父节点，直接判断这个父节点下是否有子节点

**if**(CollectionUtils.*isEmpty*(list)) {

// 子节点列表为空，说明没有子节点，则需要修改当前节点状态

parent.setIsParent(**false**);

contentCategoryMapper.updateByPrimaryKey(parent);

}

// 返回结果

**return** TaotaoResult.*ok*();

}

// 通过父节点id来查询所有子节点，这是抽离出来的公共方法

**private** List<TbContentCategory> getContentCategoryListByParentId(**long** parentId) {

TbContentCategoryExample example = **new** TbContentCategoryExample();

Criteria criteria = example.createCriteria();

criteria.andParentIdEqualTo(parentId);

List<TbContentCategory> list = contentCategoryMapper.selectByExample(example);

**return** list;

}

// 递归删除节点

**private** **void** deleteNode(**long** parentId) {

List<TbContentCategory> list = getContentCategoryListByParentId(parentId);

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

//contentCategory.setStatus(2);

//contentCategoryMapper.updateByPrimaryKey(contentCategory);

// 为了避免冗余数据，此处硬删除数据

contentCategoryMapper.deleteByPrimaryKey(contentCategory.getId());

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

deleteNode(contentCategory.getId());

}

}

}

}

#### Controller层

内容分类删除需要注意的问题：

需要判断删除的节点是否存在子节点，不存在则直接删除，存在则依次递归删除（级联删除）

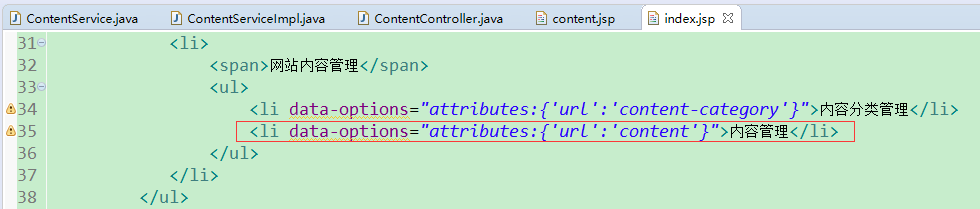
其次需要判断当前节点删除后会对其父节点有何影响，及时更新其对应父节点的状态



## 内容管理

### 内容列表

在之前的基础上完成内容列表分页显示





#### Dao

可以实现逆向工程生成的mapper文件+PageHelper实现

#### Service层

Service层接收分页参数，分别为page（第几页）、rows（多少行数据）。随后通过调用dao查询商品列表，结合pagehelper实现分页，返回对应的商品列表。

通过创建一个pojo对象，使其返回一个**EasyUIDateGrid**支持的数据格式。此pojo应该放到taotao-common工程（通用，使其可以被多个项目引用）中。创建完成要在taotao-common工程中执行mvn clean install，及时更新maven仓库。（此处省略）

Service层代码实现：

|  |
| --- |
| @Service  **public** **class** ContentServiceImpl **implements** ContentService {  @Autowired  **private** TbContentMapper contentMapper;  @Override  **public** EUDataGridResult getContentList(Integer page, Integer rows,Long categoryId) {  // 查询Content列表  TbContentExample example = **new** TbContentExample();  Criteria criteria = example.createCriteria();  criteria.andCategoryIdEqualTo(categoryId);  // 分页处理  PageHelper.*startPage*(page, rows);  // List<TbContent> list = contentMapper.selectByExample(example);  List<TbContent> list = contentMapper.selectByExampleWithBLOBs(example);  // 创建一个返回值对象  EUDataGridResult result = **new** EUDataGridResult();  result.setRows(list);  // 取记录总条数  PageInfo<TbContent> pageInfo = **new** PageInfo<>(list);  result.setTotal(pageInfo.getTotal());  **return** result;  }  } |

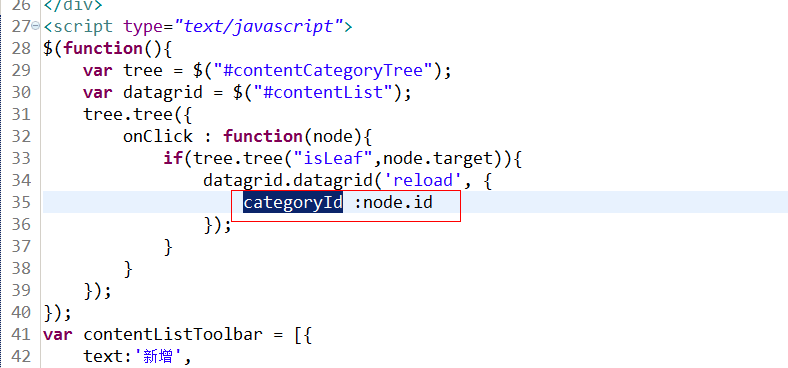
#### Controller

接收页面传递过来的参数page、rows。返回json格式的数据。EUDataGridResult

需要使用到@ResponseBody注解。

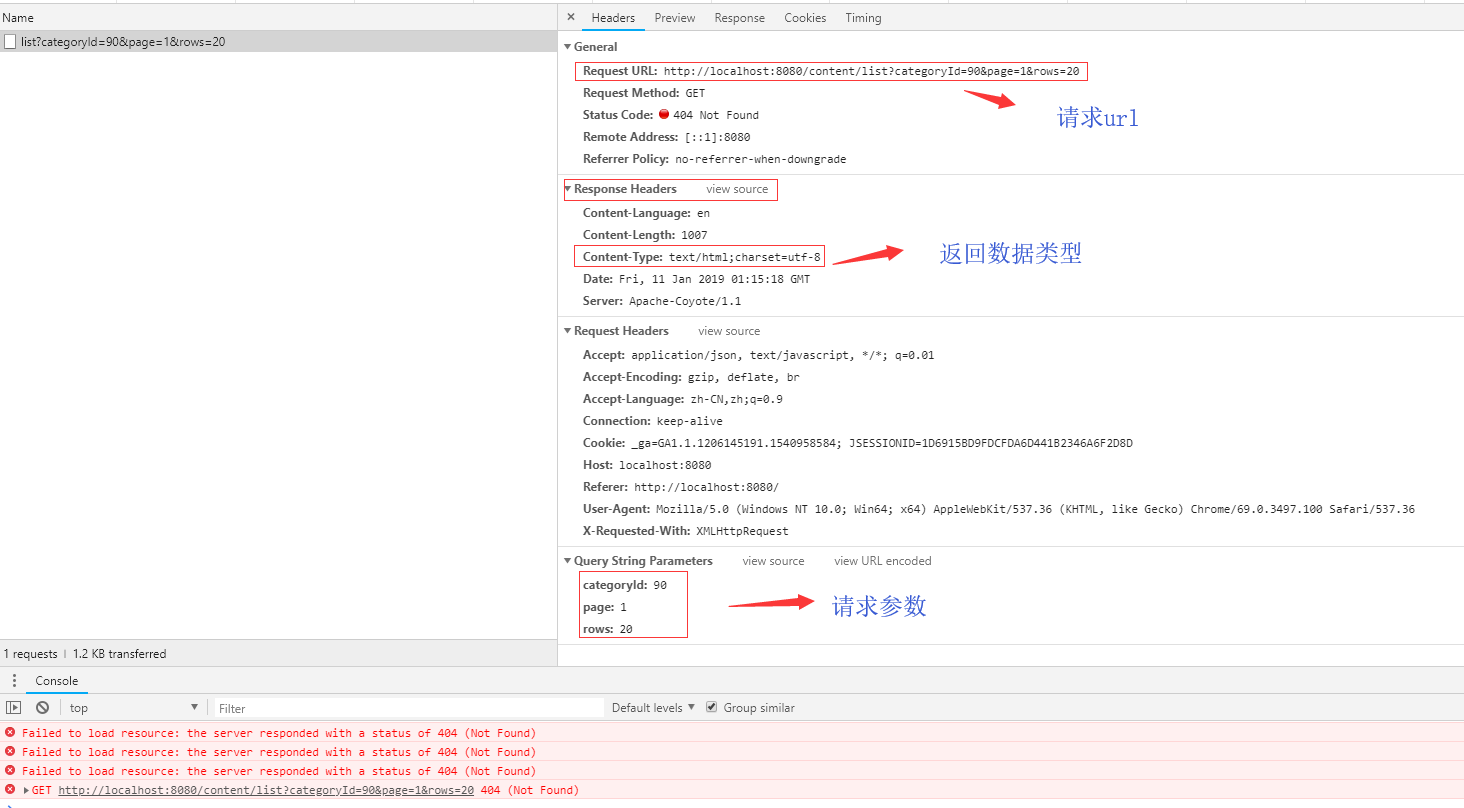
访问路径：/content//list

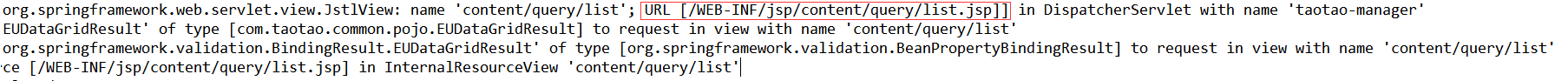
|  |
| --- |
| @RequestMapping("/item/list")  @ResponseBody  **public** EUDataGridResult getItemList(Integer page, Integer rows) {  EUDataGridResult result = itemService.getItemList(page, rows);  **return** result;  } |



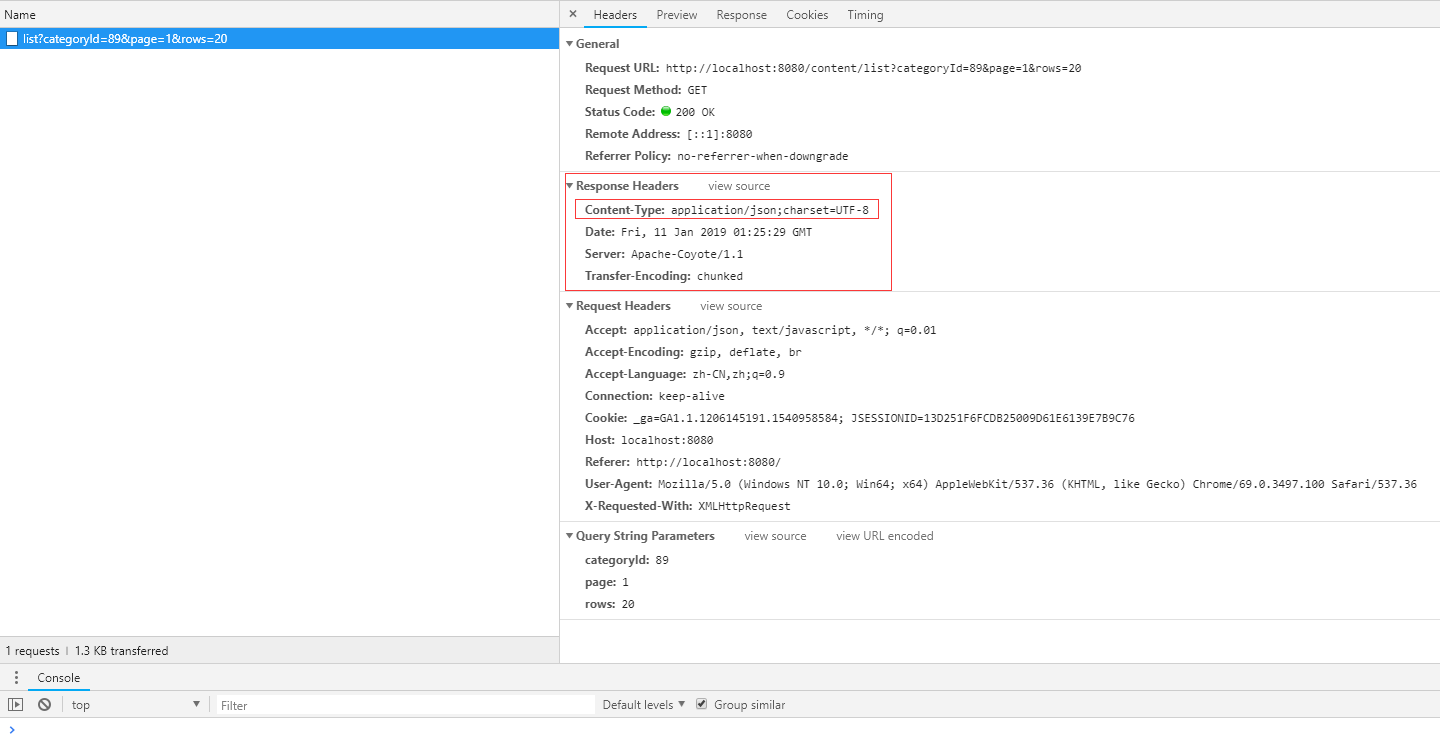
测试过程中遇到的问题，Controller层与相应的jsp文件不匹配？实质上页面能够访问到controller层，但其跳转访问页面显示是找不到对应的映射页面，导致报404出错。

在测试的时候Controller没有加@ResponseBody注解，导致其默认解析为视图，因此跳转到指定页面无法访问，报404错误。通过F12查看相关的参数，返回的数据类型是“text/html”,在controller层添加@ResponseBody注解进行测试





修正后正常访问数据：



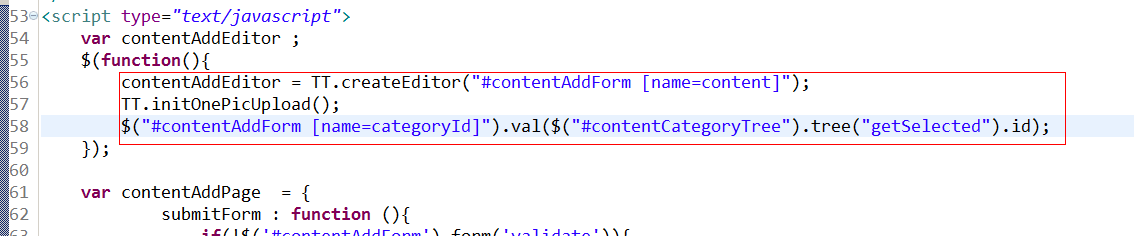
### 新增内容

#### 分析



请求url：/content-add

初始化页面：



表单提交处理：content-add.jsp



请求的url：/content/save

提交的内容：表单中的内容。使用TbContent接收

返回值：TaotaoResult

#### Dao层

tb\_content可以使用逆向工程。

#### Service层

接收一个pojo对象，补全属性。插入数据。返回TaotaoResult。

参数：TbContent

返回值：TaotaoResult

|  |
| --- |
| @Service  **public** **class** ContentServiceImpl **implements** ContentService {  @Autowired  **private** TbContentMapper contentMapper;    @Override  **public** TaotaoResult insertContent(TbContent content) {  content.setCreated(**new** Date());  content.setUpdated(**new** Date());  //插入数据  contentMapper.insert(content);    **return** TaotaoResult.*ok*();  }  } |

#### Controller

接收表单中的内容，使用TbContent接收。调用Service插入数据。返回TaotaoResult（json数据）

|  |
| --- |
| @RequestMapping("/content/save")  @ResponseBody  **public** TaotaoResult insertContent(TbContent content) {  TaotaoResult result = contentService.insertContent(content);  **return** result;  } |

### 编辑内容

#### 分析





#### Dao层

使用mybatis逆向工程

#### Service层

@Override

**public** TaotaoResult updateContent(TbContent content) {

content.setUpdated(**new** Date());

// 修改数据

contentMapper.updateByPrimaryKeyWithBLOBs(content);

**return** TaotaoResult.*ok*();

}

#### Controller层

@RequestMapping("/content/edit")

@ResponseBody

**public** TaotaoResult updateContent(TbContent content) {

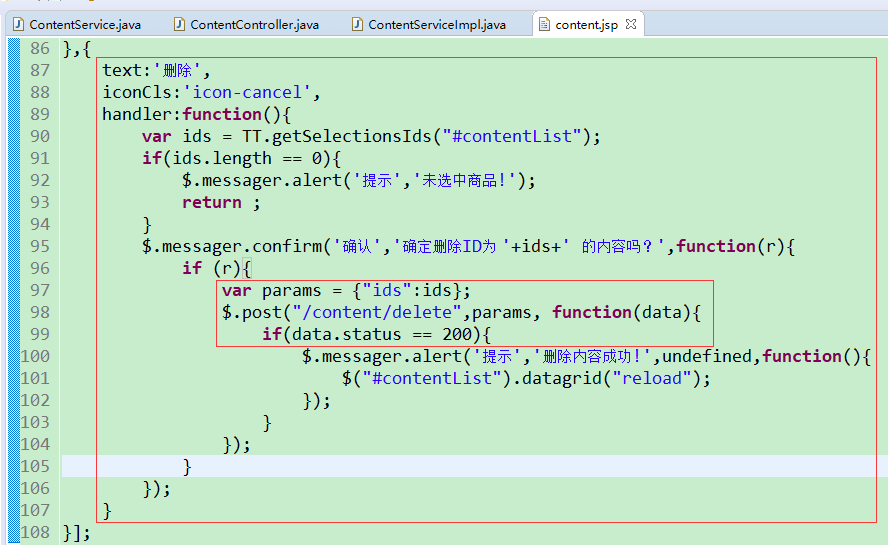
TaotaoResult result = contentService.updateContent(content);

**return** result;

}

### 删除内容

#### 分析



#### Dao层

使用mybatis生成的mapper文件

#### Service层

@Override

**public** TaotaoResult deleteContent(List<Long> contentIds) {

**for**(Long id : contentIds) {

contentMapper.deleteByPrimaryKey(id);

}

**return** TaotaoResult.*ok*();

}

#### Controller层

@RequestMapping("/content/delete")

@ResponseBody

**public** TaotaoResult deleteContent(@RequestParam(value="ids")List<Long> contentIds) {

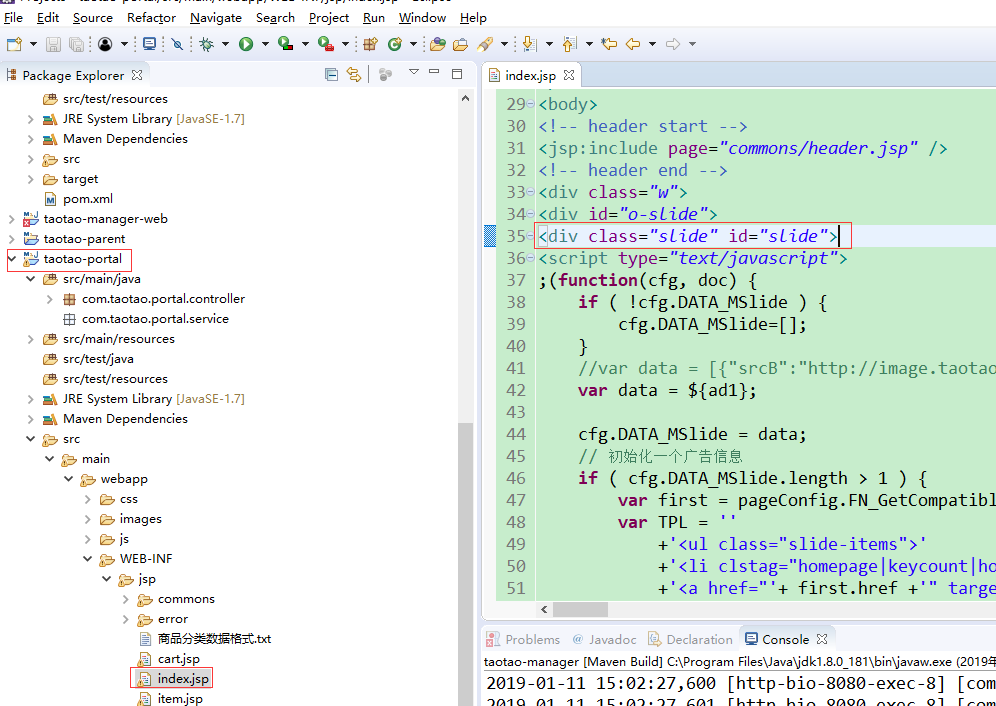
TaotaoResult result = contentService.deleteContent(contentIds);

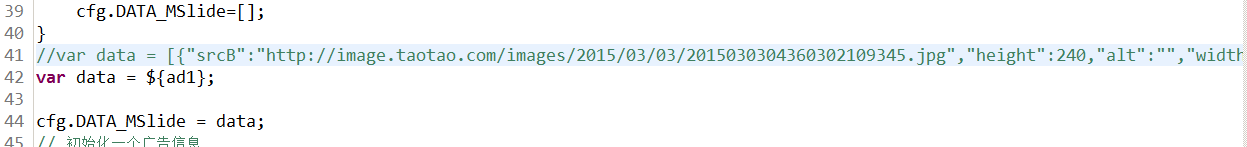
**return** result;

}

## 轮播图的展示

### 分析





可以使用jsonp获得数据。做网站不推荐大面积使用ajax。

不使用ajax如何获得数据？

只需要在portal的Service层调用taotao-rest的服务，获得数据，把数据传递给jsp即可。

要调用服务需要使用java代码模拟浏览器调用服务请求数据。

可以使用HttpClient。

### HttpClient

#### 使用步骤

第一步：把HttpClient使用的jar包添加到工程中。

第二步：创建一个HttpClient的测试类

第三步：创建测试方法。

第四步：创建一个HttpClient对象

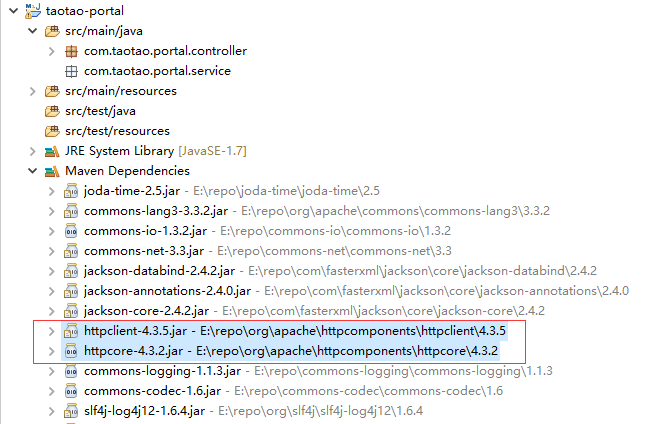
第五步：创建一个HttpGet对象，需要制定一个请求的url

第六步：执行请求。

第七步：接收返回结果。HttpEntity对象。

第八步：取响应的内容。

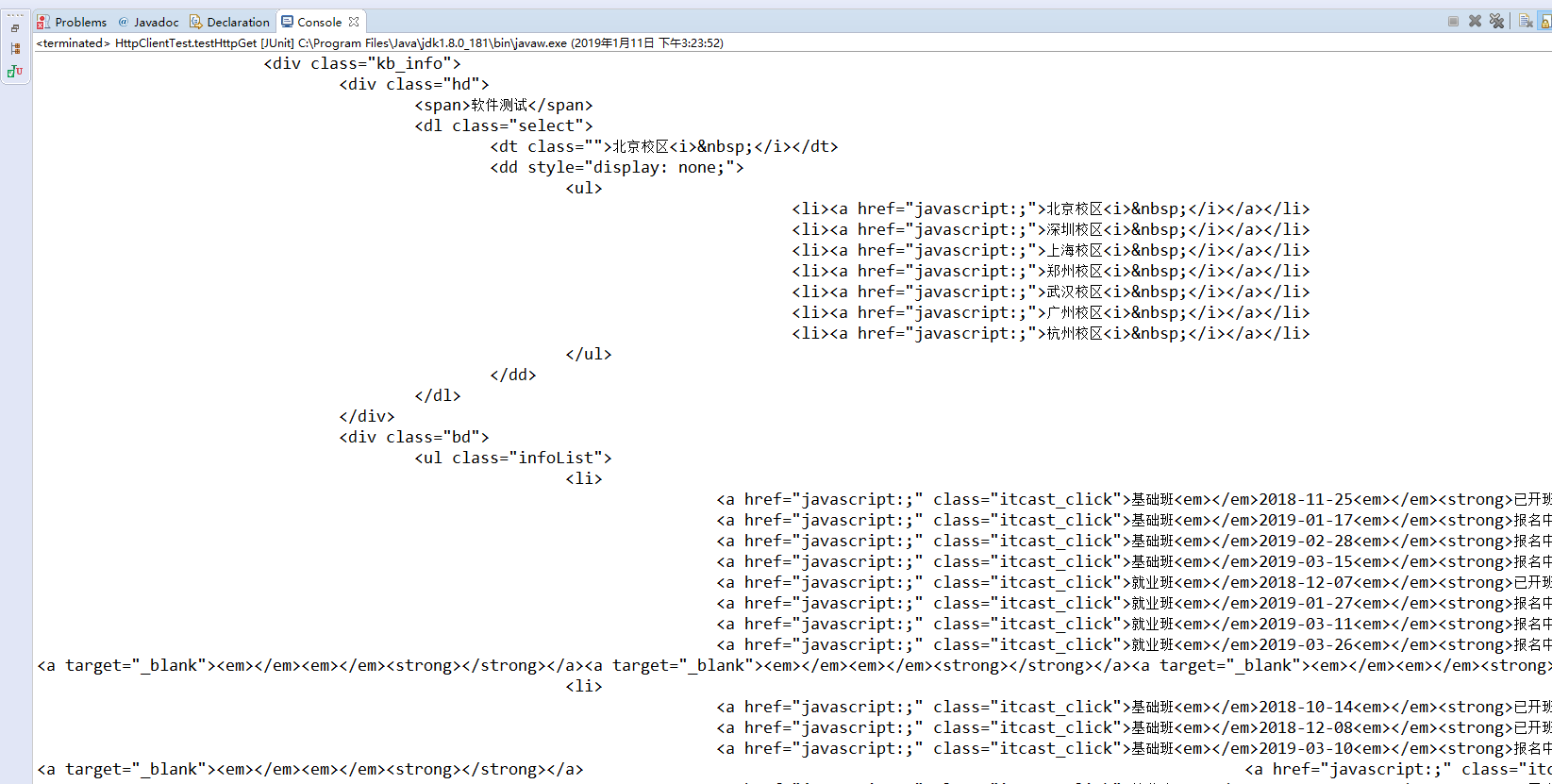
第九步：关闭HttpGet、HttpClient。



#### Get请求

|  |
| --- |
| **import** org.apache.http.HttpEntity;  **import** org.apache.http.client.methods.CloseableHttpResponse;  **import** org.apache.http.client.methods.HttpGet;  **import** org.apache.http.impl.client.CloseableHttpClient;  **import** org.apache.http.impl.client.HttpClients;  **import** org.apache.http.util.EntityUtils;  **import** org.junit.Test;  /\*\*  \* **@author** hahabibu  \* 系统名称:  \* 模块名称:  \* 类 名 称: HttpClientTest  \* 软件版权: 淘淘商城  \* 功能说明：测试服务连接  \* 系统版本：  \* 开发时间: create in 2019年1月11日 下午3:20:35  \* 开发说明:  \*/  **public** **class** HttpClientTest {  @Test  **public** **void** testHttpGet() **throws** Exception {  // 第一步：把HttpClient使用的jar包添加到工程中。  // 第二步：创建一个HttpClient的测试类  // 第三步：创建测试方法。  // 第四步：创建一个HttpClient对象  CloseableHttpClient httpClient = HttpClients.*createDefault*();  // 第五步：创建一个HttpGet对象，需要制定一个请求的url  HttpGet get = **new** HttpGet("http://www.itheima.com");  // 第六步：执行请求。  CloseableHttpResponse response = httpClient.execute(get);  // 第七步：接收返回结果。HttpEntity对象。  HttpEntity entity = response.getEntity();  // 第八步：取响应的内容。  String html = EntityUtils.*toString*(entity);  System.***out***.println(html);  // 第九步：关闭response、HttpClient。  response.close();  httpClient.close();  }  } |

连接测试：



#### Post请求

实现步骤：

第一步：创建一个httpClient对象

第二步：创建一个HttpPost对象。需要指定一个url

第三步：创建一个list模拟表单，list中每个元素是一个NameValuePair对象

第四步：需要把表单包装到Entity对象中。StringEntity

第五步：执行请求。

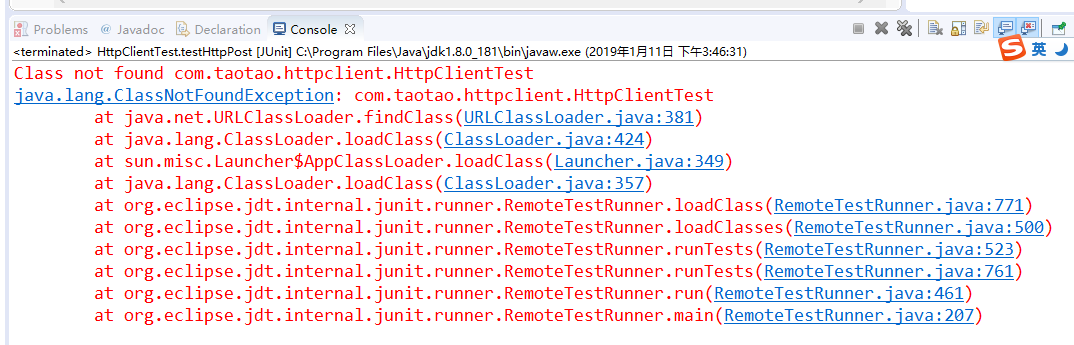
第六步：接收返回结果

第七步：关闭流。

|  |
| --- |
| 在taotao-portal中添加controller方法测试    对应测试方法：  **import** java.util.ArrayList;  **import** java.util.List;  **import** org.apache.http.HttpEntity;  **import** org.apache.http.NameValuePair;  **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.entity.StringEntity;  **import** org.apache.http.impl.client.CloseableHttpClient;  **import** org.apache.http.impl.client.HttpClients;  **import** org.apache.http.message.BasicNameValuePair;  **import** org.apache.http.util.EntityUtils;  **import** org.junit.Test;  **public** **class** HttpClientTest {  @Test  **public** **void** testHttpPost() **throws** Exception {  // 第一步：创建一个httpClient对象  CloseableHttpClient httpClient = HttpClients.*createDefault*();  // 第二步：创建一个HttpPost对象。需要指定一个url  HttpPost post = **new** HttpPost("http://localhost:8082/posttest.html");  // 第三步：创建一个list模拟表单，list中每个元素是一个NameValuePair对象  List<NameValuePair> formList = **new** ArrayList<>();  formList.add(**new** BasicNameValuePair("name", "张三"));  formList.add(**new** BasicNameValuePair("pass", "1243"));  // 第四步：需要把表单包装到Entity对象中。StringEntity  StringEntity entity = **new** UrlEncodedFormEntity(formList, "utf-8");  post.setEntity(entity);  // 第五步：执行请求。  CloseableHttpResponse response = httpClient.execute(post);  // 第六步：接收返回结果  HttpEntity httpEntity = response.getEntity();  String result = EntityUtils.*toString*(httpEntity);  System.***out***.println(result);  // 第七步：关闭流。  response.close();  httpClient.close();  }  } |

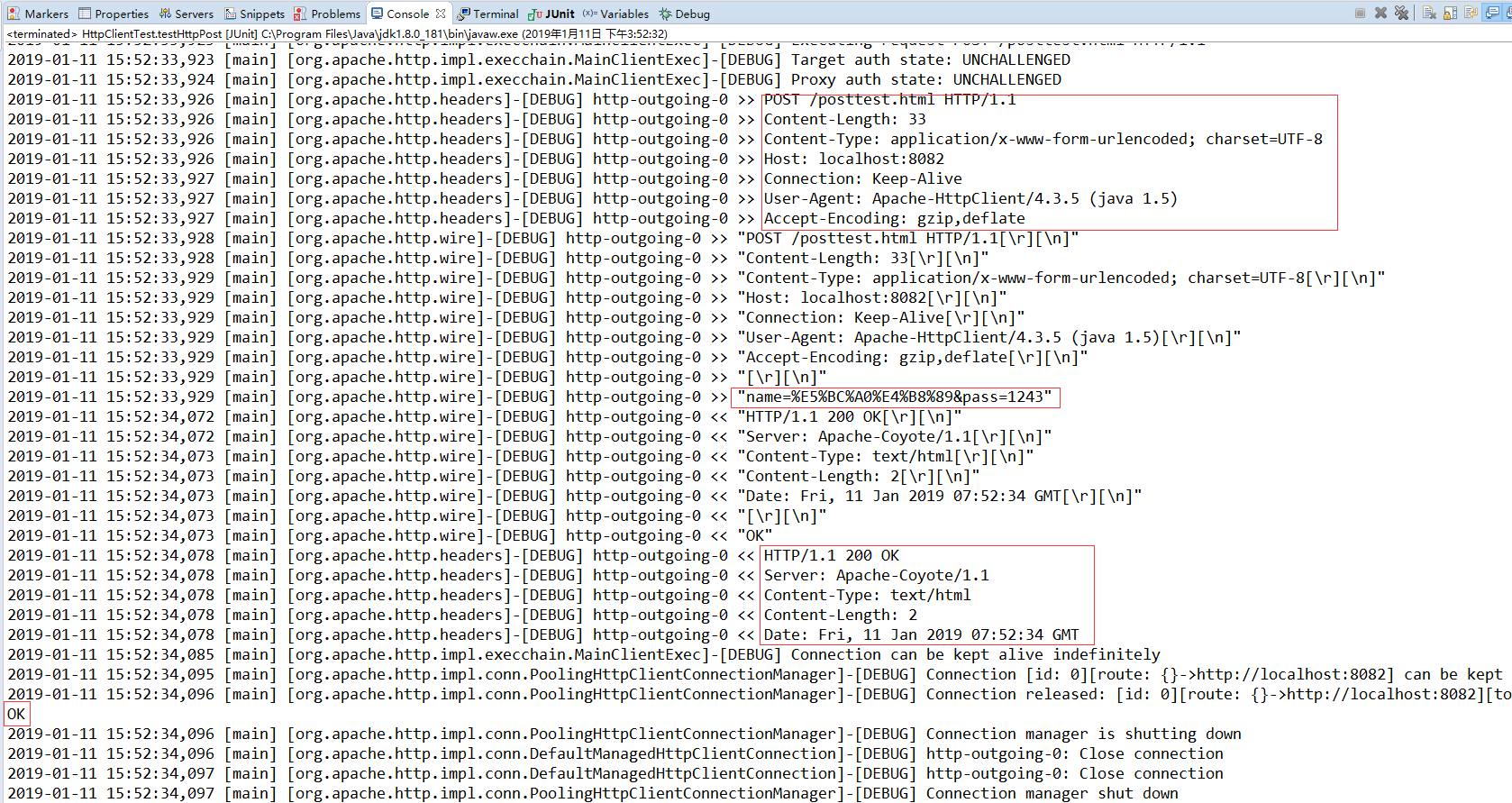
测试说明：

重新启动taotao-portal



当前编写的类文件没有编译，因此清理一下代码随后重新启动测试

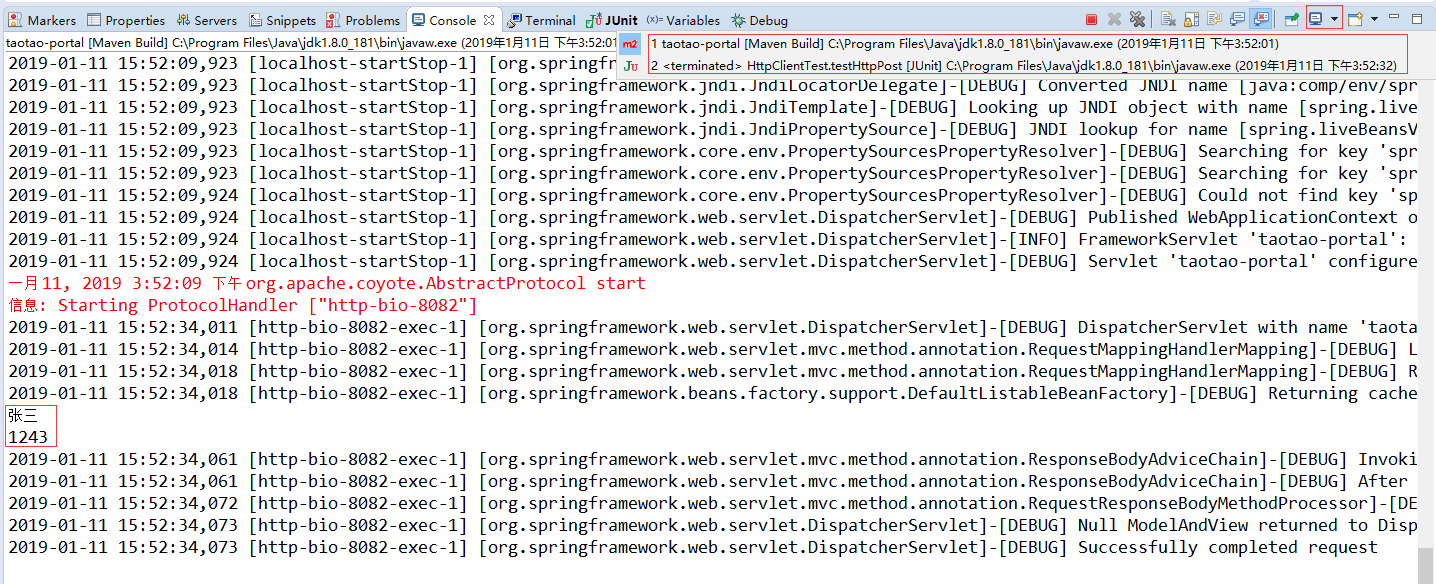
taotao-portal执行mvn clean，随后重新启动taotao-portal，若还无法执行则重新保存一下测试文件，再次debug测试



如果测试过程中出现乱码问题：



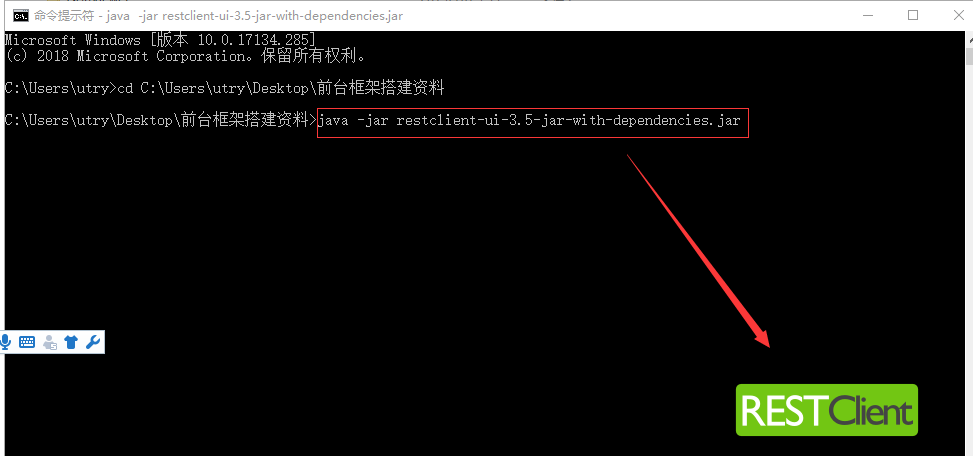
在控制台中选择相应的内容进行切换查看数据是否显示正常

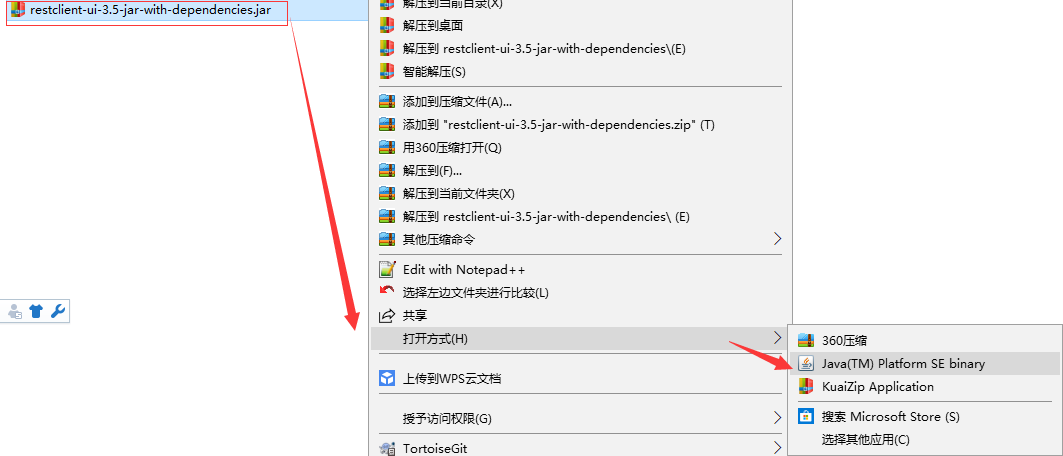


另一种服务器连接测试方式：

restclient-ui-3.5-jar-with-dependencies.jar说明，也可通过命令行打开

进入到当前jar包指定的目录，随后通过java命令行进行操作





测试说明：

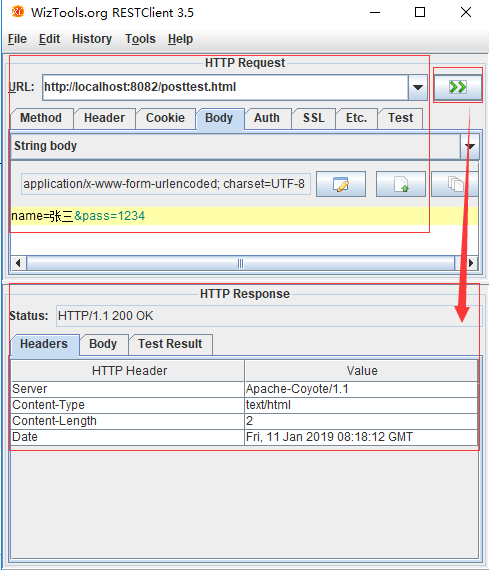
Method选择post

Body选择String body

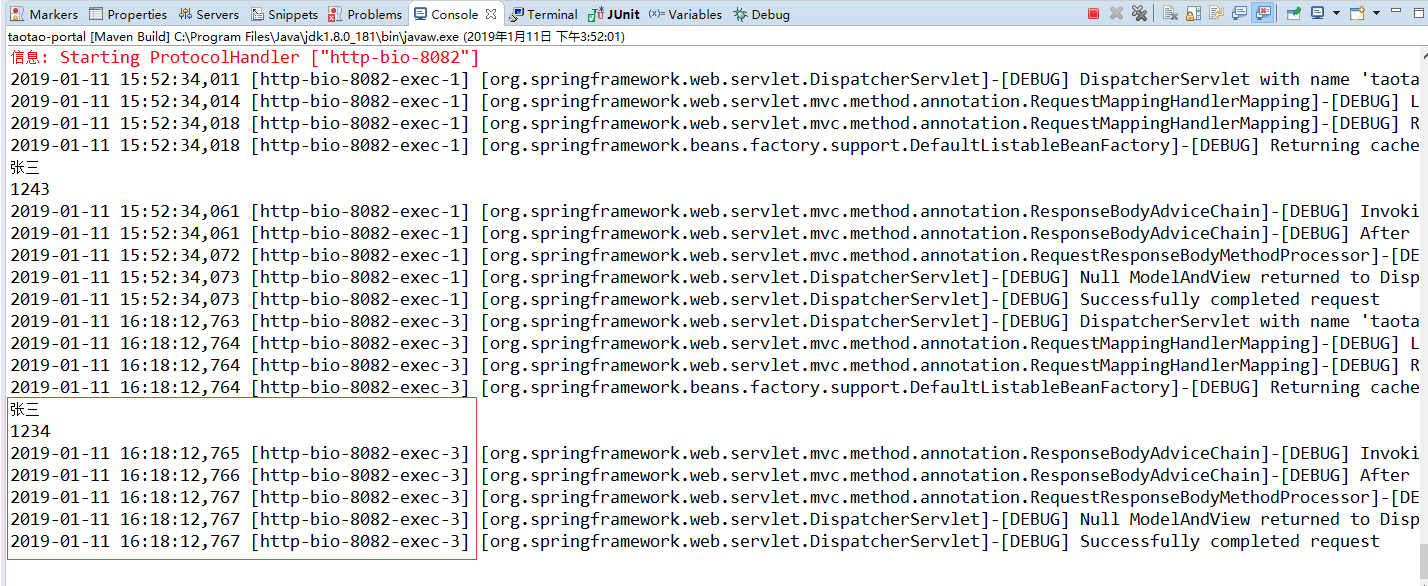
提交表单和提交json数据，content-Type不同

表单的content-type：application/x-www-form-urlencoded

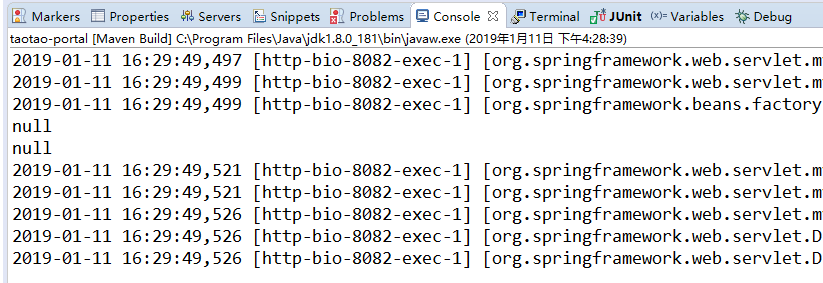
Json的content-type：application/json



测试通过：



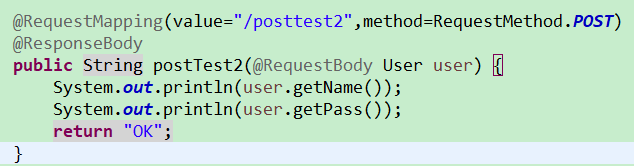
如果是json格式的数据直接测试，则不能正常获取数据



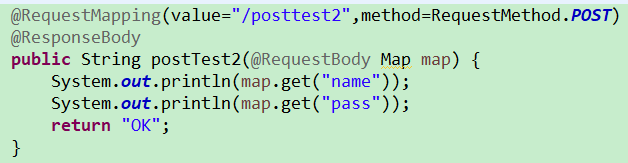
如果是json格式的数据，需要通过一个对象进行接收，借助@RequestBody注解完成转义

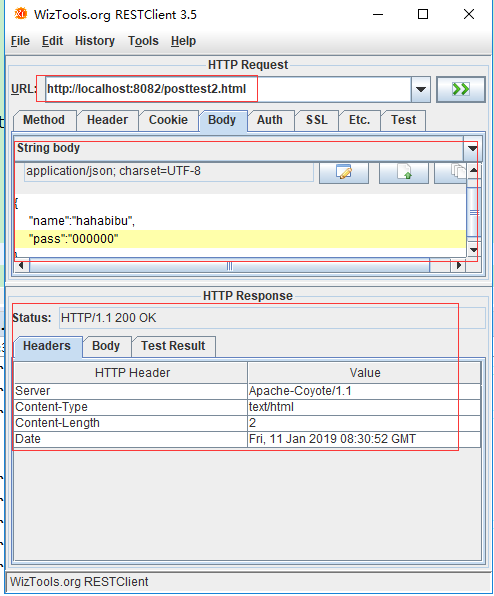
Json格式:

创建一个User对象，拥有name、pass属性，提供getter、setter方法，编写如下代码测试

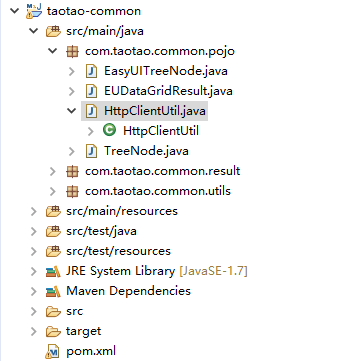


或者通过Map接收





实际使用时需要把HttpClient封装成一个工具类，可以把工具类放到taotao-common中。



实现步骤：

1.taotao-rest中发布服务

2.Taotao-portal中使用HttpClient调用服务获得数据

3.展示轮播图

### 发布内容查询服务

#### 分析

根据内容分类id查询tb\_content表，得到此分类下的内容列表，返回。

请求的url：/rest/content/{cid}

返回的数据：json数据：包含查询内容列表、包含状态、包含错误消息

使用TaotaoResult包装一个内容列表即可。

#### Dao层

使用mybatis逆向工程

#### Service层

接收内容分类id作为参数，使用mapper根据分类id查询内容列表，返回内容列表。

参数：Long cid

返回结果：List<TbContent>

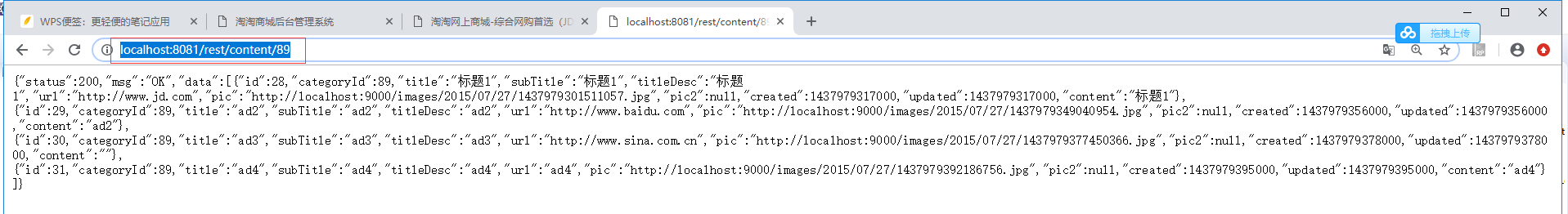
|  |
| --- |
| @Service  **public** **class** ContentServiceImpl **implements** ContentService {  @Autowired  **private** TbContentMapper contentMapper;  @Override  **public** List<TbContent> getContentList(Long cid) {  // 根据cid查询内容列表  TbContentExample example = **new** TbContentExample();  Criteria criteria = example.createCriteria();  criteria.andCategoryIdEqualTo(cid);  //执行查询  List<TbContent> list = contentMapper.selectByExampleWithBLOBs(example);  **return** list;  }  } |

#### Controller

接收参数cid，从url中取参数。调用Service查询内容列表，返回json数据，返回TaotaoResult包装内容列表。

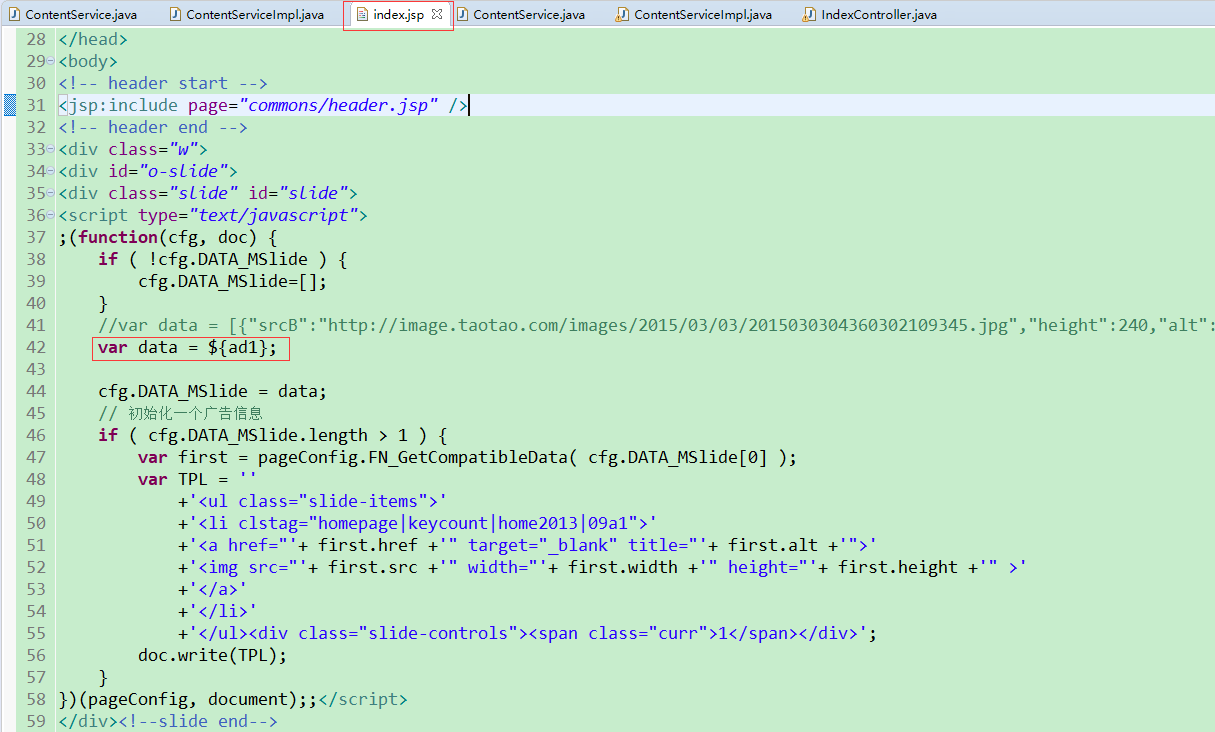
|  |
| --- |
| @Controller  **public** **class** ContentController {  @Autowired  **private** ContentService contentService;  @RequestMapping("/content/{cid}")  @ResponseBody  **public** TaotaoResult getContentList(@PathVariable Long cid) {  **try** {  List<TbContent> list = contentService.getContentList(cid);  **return** TaotaoResult.*ok*(list);  } **catch** (Exception e) {  e.printStackTrace();  **return** TaotaoResult.*build*(500, ExceptionUtil.*getStackTrace*(e));  }  }  } |

启动taotao-rest，测试：<http://localhost:8081/rest/content/89>



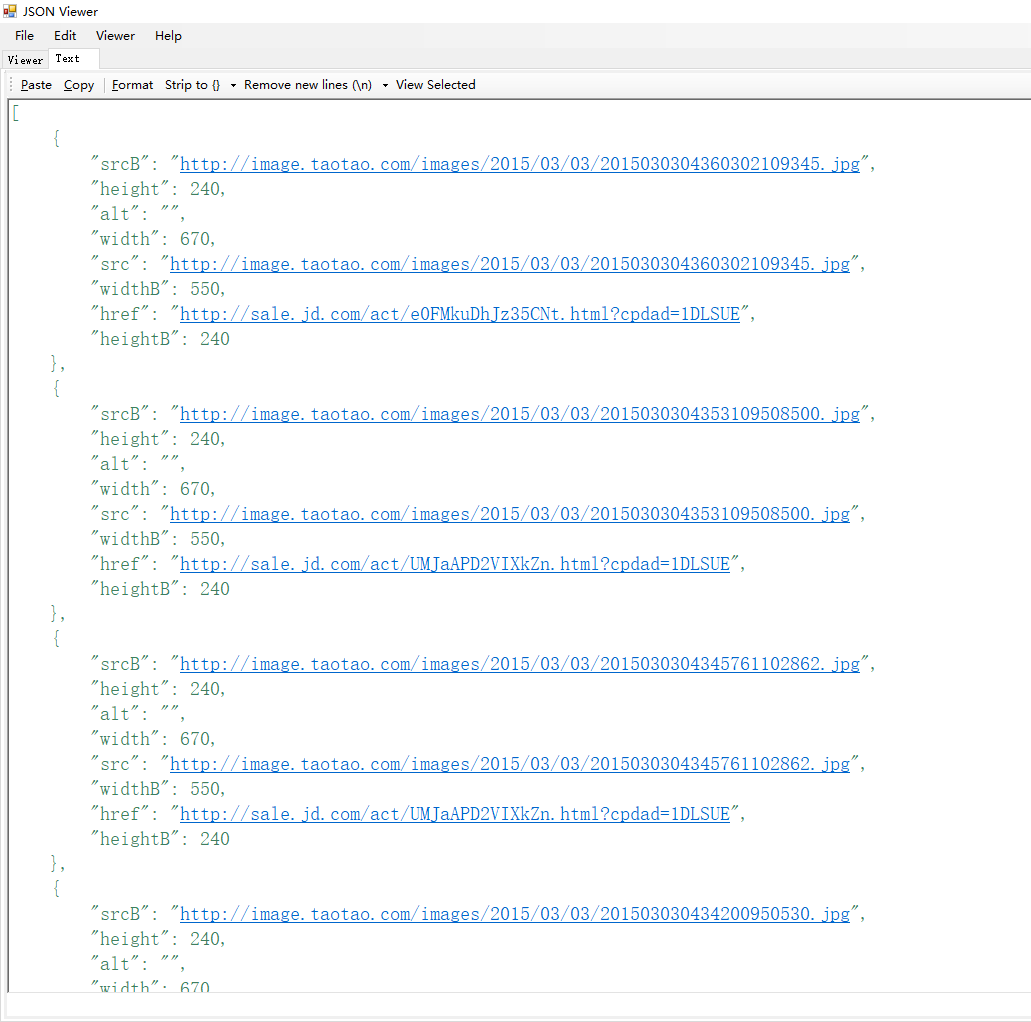
### 展示轮播图

#### 分析



需要在taotao-portal中实现。需要使用HttpClient调用rest发布的服务。把结果传递给jsp。

Jsp中需要的json数据格式：



需要创建一个pojo描述此格式。

|  |
| --- |
| **public** **class** AdNode {  **private** **int** height;  **private** **int** width;  **private** String src;  **private** **int** heightB;  **private** **int** widthB;  **private** String srcB;  **private** String alt;  **private** String href;  // 对应getter、setter构造器  } |

#### Service

使用HttpClient调用taotao-rest发布的服务，得到一个json字符串。

把json转换成java对象，取data属性，把list转换成AdNode的list。

最后需要把AdNode的list转换成json返回。

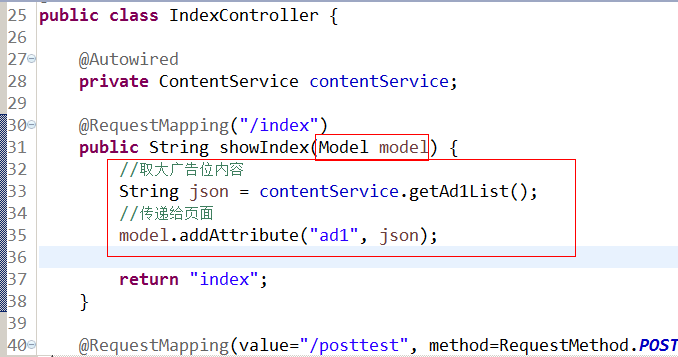
参数：cid（从配置文件中获得）

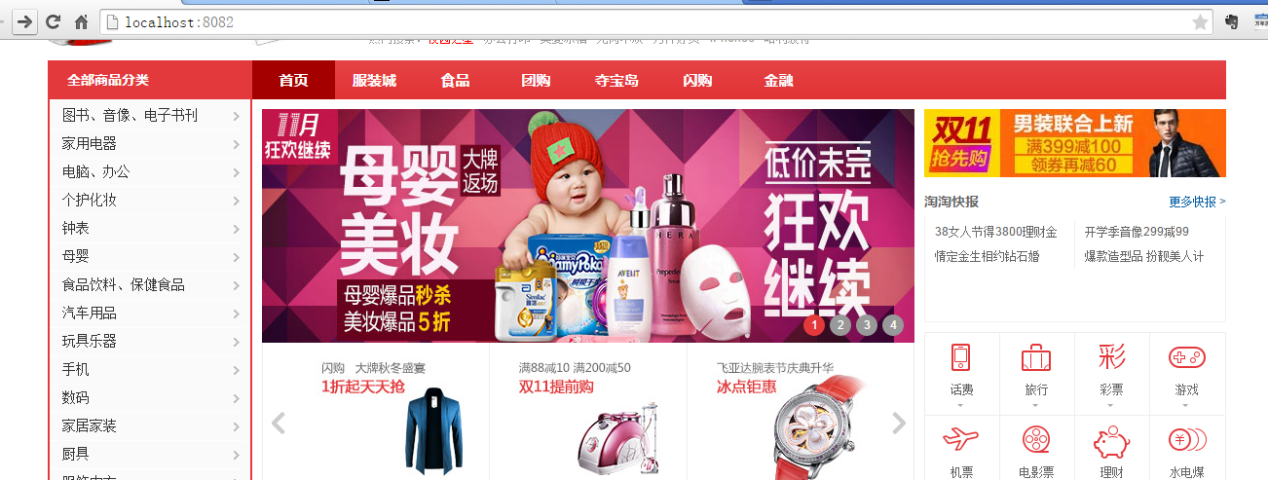
返回结果：String

|  |
| --- |
| **import** java.util.ArrayList;  **import** java.util.List;  **import** org.springframework.beans.factory.annotation.Value;  **import** org.springframework.stereotype.Service;  **import** com.taotao.common.pojo.HttpClientUtil;  **import** com.taotao.common.result.TaotaoResult;  **import** com.taotao.common.utils.JsonUtils;  **import** com.taotao.pojo.TbContent;  **import** com.taotao.portal.pojo.AdNode;  **import** com.taotao.portal.service.ContentService;  @Service  **public** **class** ContentServiceImpl **implements** ContentService {    @Value("${REST\_BASE\_URL}")  **private** String REST\_BASE\_URL;  @Value("${REST\_CONTENT\_URL}")  **private** String REST\_CONTENT\_URL;  @Value("${REST\_CONTENT\_AD1\_CID}")  **private** String REST\_CONTENT\_AD1\_CID;  /\*\*  \* 获得大广告位的内容  \* <p>Title: getAd1List</p>  \* <p>Description: </p>  \* **@return**  \* **@see** com.taotao.portal.service.ContentService#getAd1List()  \*/  @Override  **public** String getAd1List() {  //调用服务获得数据  // String json ="http://localhost/8081/rest/content/89";  String json = HttpClientUtil.*doGet*(REST\_BASE\_URL + REST\_CONTENT\_URL + REST\_CONTENT\_AD1\_CID);  //把json转换成java对象  TaotaoResult taotaoResult = TaotaoResult.*formatToList*(json, TbContent.**class**);  //取data属性，内容列表  List<TbContent> contentList = (List<TbContent>) taotaoResult.getData();  //把内容列表转换成AdNode列表  List<AdNode> resultList = **new** ArrayList<>();  **for** (TbContent tbContent : contentList) {  AdNode node = **new** AdNode();  node.setHeight(240);  node.setWidth(670);  node.setSrc(tbContent.getPic());    node.setHeightB(240);  node.setWidthB(550);  node.setSrcB(tbContent.getPic2());    node.setAlt(tbContent.getSubTitle());  node.setHref(tbContent.getUrl());    resultList.add(node);  }  //需要把resultList转换成json数据  String resultJson = JsonUtils.*objectToJson*(resultList);  **return** resultJson;  }  } |

#### Controller

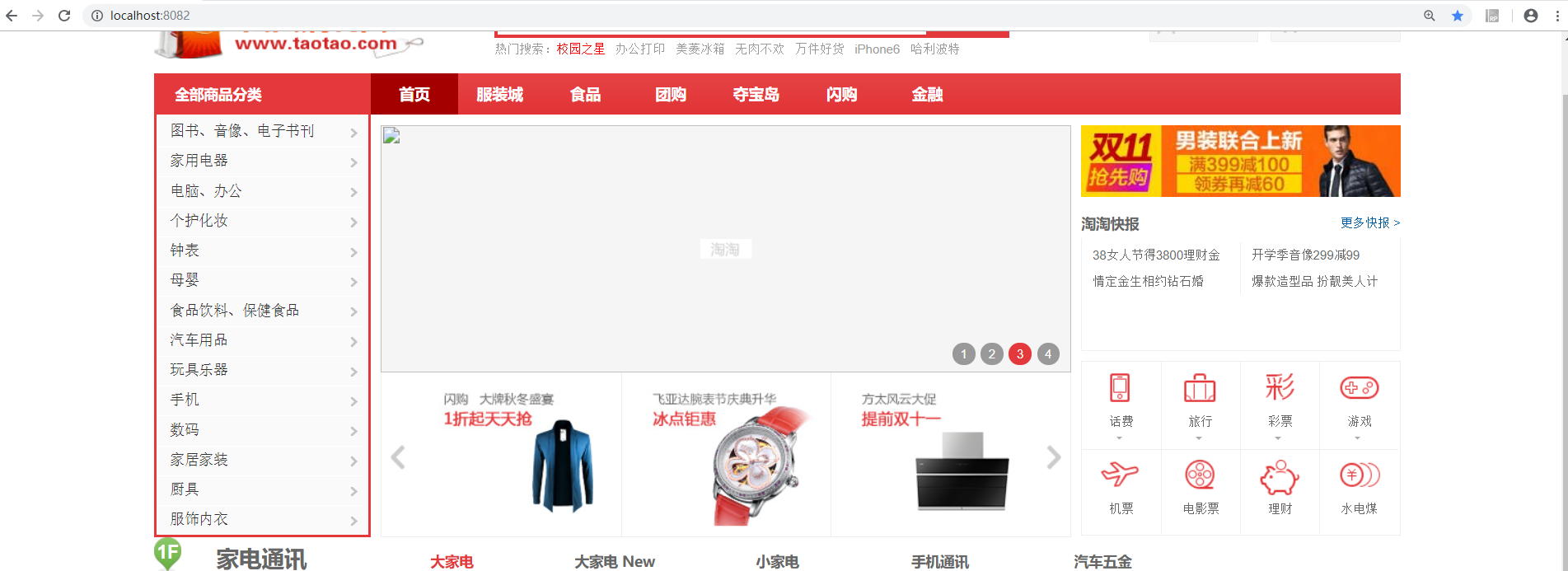
调用Service查询广告位内容，传递给jsp即可。





测试，将taotao-common重新打包，启动taotao-rest，随后启动taotao-portal进行测试

可能出现结果：没有找到指定的数据，因此此处需要借助taotao-manager后台管理系统对图片进行设置



右键查看源代码（检查），显示连接失败，说明图片链接地址有误，此处则通过后台管理系统设置相关图片，再次测试，结果正常显示

