

淘淘商城

第六天

**讲师：传智.入云龙**

# 课程计划

1. CMS系统实现
2. Httpclient学习
3. 首页内容显示

# 首页的动态实现分析

网站的首页的实现必须是动态的，可以在后台管理维护。

## CMS -- 内容管理系统

思路：

1. 分析每个模块的共性
   1. 链接
   2. 图片
   3. 标题
   4. 子标题
2. 使用两张表来管理
   1. 内容分类表，管理内容的大分类
   2. 内容表，存储每个分类下的明细信息内容。

## 首页大广告的展示流程



# CMS 内容管理系统

## 需求分析（演示功能）

内容列表

新增节点

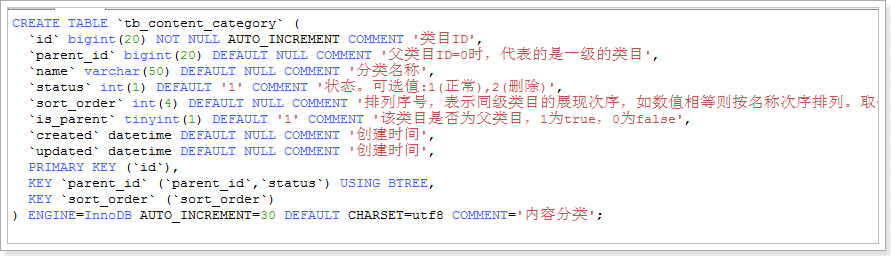
。。。。

## 数据库设计

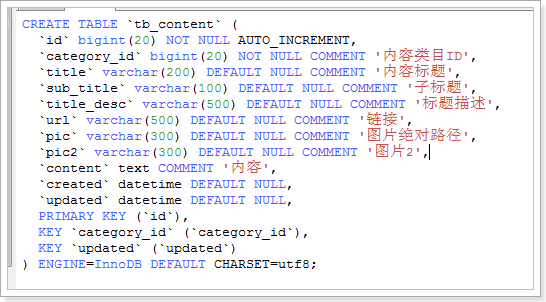
需要一张表存储内容数据：字段：标题、url、子标题、image、内容分类id等等

需要一张表存储内容的分类信息，树形结构的表。

内容分类表：

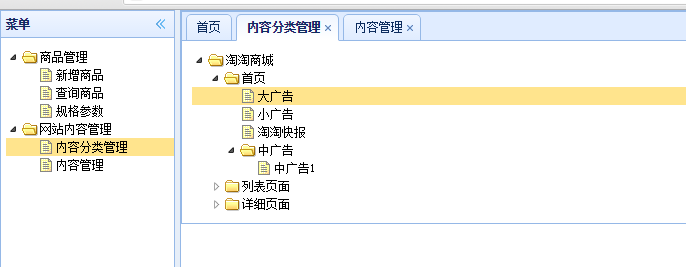


内容表：



## 内容分类列表

内容分类管理实际上是一个树形结构的管理，新增子节点、重命名、删除（级联删除所有的子节点）



### 分析

查询内容分类列表，返回一个分类列表。

请求的url：

/content/category/list

响应的数据格式：



必须包含id、text、state三个节点。

### Pojo定义

对应返回结果的数据类型定义一个pojo类，其中包含三个属性id、text、state。

|  |
| --- |
| **public** **class** EasyUITreeNode {    **private** Integer id;  **private** String text;  **private** String state;  } |

### Mapper

使用逆向工程生成的mapper映射文件和mapper接口文件。

### Service

|  |
| --- |
| @Service  **public** **class** ContentCategoryServiceImpl **implements** ContentCategoryService {  @Autowired  **private** TbContentCategoryMapper contentCategoryMapper;    @Override  **public** List<EasyUITreeNode> getContentCategoryList(**long** parentid) **throws** Exception {    //根据parentid查询内容分类列表  TbContentCategoryExample example = **new** TbContentCategoryExample();  Criteria criteria = example.createCriteria();  criteria.andParentIdEqualTo(parentid);  List<TbContentCategory> list = contentCategoryMapper.selectByExample(example);  List<EasyUITreeNode> resultList = **new** ArrayList<>();  **for** (TbContentCategory tbContentCategory : list) {  EasyUITreeNode node = **new** EasyUITreeNode();  node.setId(tbContentCategory.getId());  node.setText(tbContentCategory.getName());  //判断是否是父节点  **if** (tbContentCategory.getIsParent()) {  node.setState("closed");  } **else** {  node.setState("open");  }  resultList.add(node);  }  **return** resultList;  }  } |

### Controller

|  |
| --- |
| @Controller  @RequestMapping("/content/category")  **public** **class** ContentCategoryController {  @Autowired  **private** ContentCategoryService contentCategoryService;    @RequestMapping("/list")  @ResponseBody  **public** List<EasyUITreeNode> getContentCategoryList(@RequestParam(value="id", defaultValue="0")**long** parentid) **throws** Exception {  List<EasyUITreeNode> list = contentCategoryService.getContentCategoryList(parentid);    **return** list;  }  } |

## 新增子节点

### 需求

请求的url：/content/category/create

请求的参数：

parentId：父节点id

Name：新节点的名称

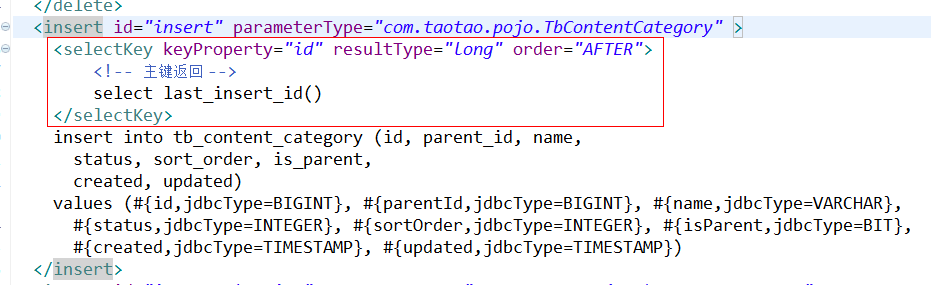
返回的数据格式：

TaotaoResult

### Mapper

使用逆向工程生成的mapper文件。

添加主键返回：



### Service

|  |
| --- |
| @Override  **public** TaotaoResult addNode(**long** parentid, String name) **throws** Exception {  Date date = **new** Date();  //添加一个新节点  //创建一个节点对象  TbContentCategory node = **new** TbContentCategory();  node.setName(name);  node.setParentId(parentid);  node.setIsParent(**false**);  node.setCreated(date);  node.setUpdated(date);  node.setSortOrder(1);  //状态。可选值:1(正常),2(删除)  node.setStatus(1);  //插入新节点。需要返回主键  contentCategoryMapper.insert(node);  //判断如果父节点的isparent不是true修改为true  //取父节点的内容  TbContentCategory parentNode = contentCategoryMapper.selectByPrimaryKey(parentid);  **if** (!parentNode.getIsParent()) {  parentNode.setIsParent(**true**);  contentCategoryMapper.updateByPrimaryKey(parentNode);  }  //把新节点返回  **return** TaotaoResult.*ok*(node);  } |

### Controller

|  |
| --- |
| @RequestMapping("/create")  @ResponseBody  **public** TaotaoResult addNode(Long parentId, String name) **throws** Exception {    TaotaoResult result = contentCategoryService.addNode(parentId, name);    **return** result;  } |

## 重命名节点

课后作业

## 删除节点

课后作业

## 内容列表

### 需求

请求的url：

/content/query/list

响应的数据格式：

EasyUIResult

### Mapper

使用逆向生成的mapper文件。

### Service

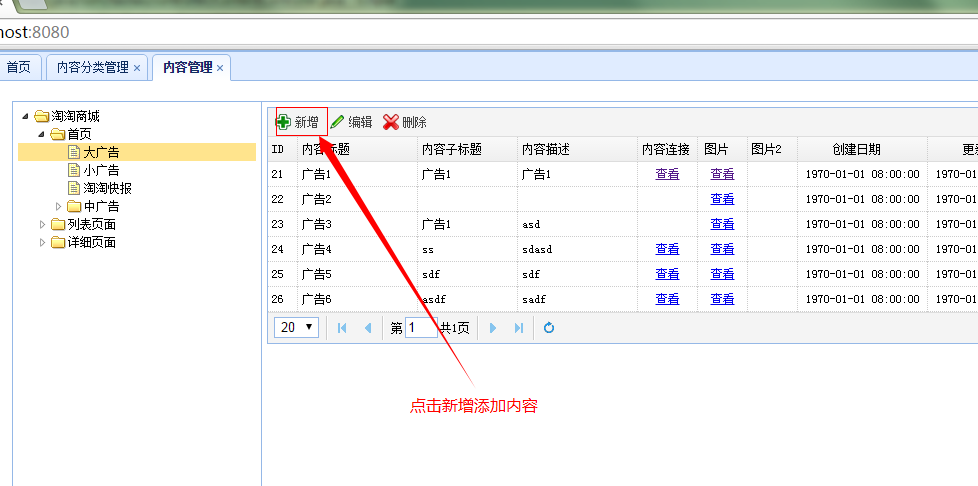
|  |
| --- |
| @Service  **public** **class** ContentServiceImpl **implements** ContentService {  @Autowired  **private** TbContentMapper contentMapper;    @Override  **public** EasyUIResult getContentList(**long** catId, Integer page, Integer rows) **throws** Exception {  //根据category\_id查询内容列表  TbContentExample example = **new** TbContentExample();  Criteria criteria = example.createCriteria();  criteria.andCategoryIdEqualTo(catId);  //分页处理  PageHelper.*startPage*(page, rows);  List<TbContent> list = contentMapper.selectByExampleWithBLOBs(example);  //取分页信息  PageInfo<TbContent> pageInfo = **new** PageInfo<>(list);  EasyUIResult result = **new** EasyUIResult(pageInfo.getTotal(), list);  **return** result;  }  } |

### Controller

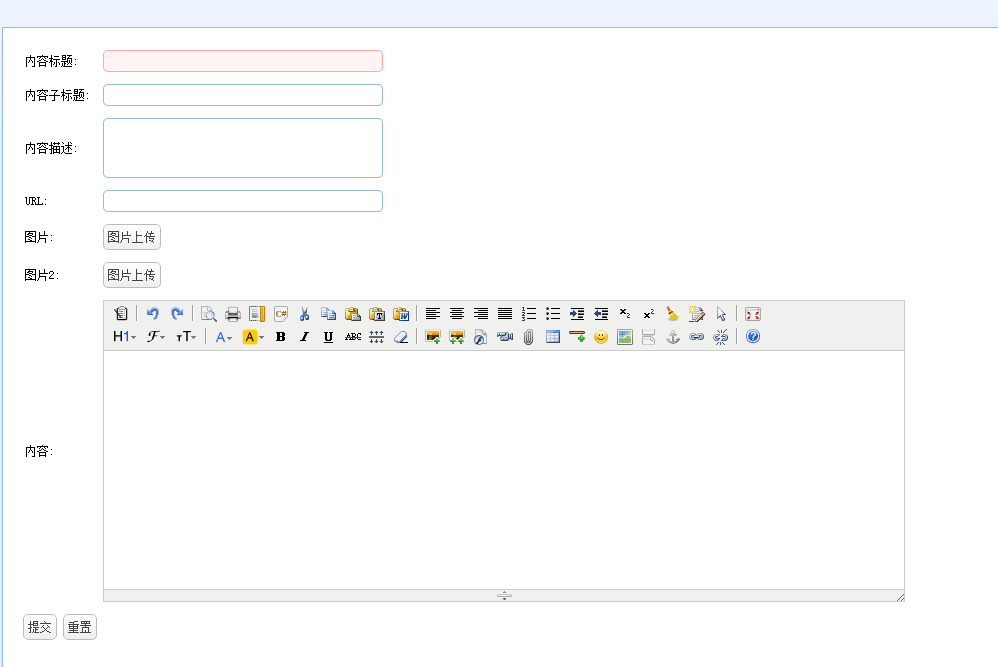
|  |
| --- |
| @Controller  @RequestMapping("/content")  **public** **class** ContentController {    @Autowired  **private** ContentService contentService;    @RequestMapping("/query/list")  @ResponseBody  **public** EasyUIResult getContentList(Long categoryId, Integer page, Integer rows) **throws** Exception {  EasyUIResult result = contentService.getContentList(categoryId, page, rows);    **return** result;    }  } |

## 添加内容

### 需求



点击新增按钮，打开content-add.jsp



点击提交按钮，使用ajax把表单中的数据提交给服务器。

1、请求的url：

/content/save

1. 请求的参数

表单中的数据

1. 响应的数据格式

TaotaoResult

### Mapper

创建一个添加内容的mapper文件，实现向tb\_content表中添加数据的功能。使用逆向工程生成。

### Service

|  |
| --- |
| @Override  **public** TaotaoResult addContent(TbContent content) **throws** Exception {    //把图片信息保存至数据库  content.setCreated(**new** Date());  content.setUpdated(**new** Date());  //把内容信息添加到数据库  contentMapper.insert(content);    **return** TaotaoResult.*ok*();  } |

### Controller

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

## 修改内容

课后作业

## 删除内容

课后作业

# 服务层发布服务

## 需求

根据内容的分类ID查询内容列表。

请求的url：

/rest/content/category/{cid}

参数：categoryId

响应的数据格式：

TaoTaoResult

## Mapper

创建一可以根据分类id查询内容列表的mapper。使用逆向工程生成的mapper即可。

## Service

|  |
| --- |
| @Service  **public** **class** ContentServiceImpl **implements** ContentService {  @Autowired  **private** TbContentMapper contentMapper;    @Override  **public** TaotaoResult getContentList(**long** cid) **throws** Exception {  TbContentExample example = **new** TbContentExample();  //添加条件  Criteria criteria = example.createCriteria();  criteria.andCategoryIdEqualTo(cid);  List<TbContent> list = contentMapper.selectByExample(example);  **return** TaotaoResult.*ok*(list);  }  } |

## Controller

|  |
| --- |
| @Controller  @RequestMapping("/content")  **public** **class** ContentController {  @Autowired  **private** ContentService contentService;    @RequestMapping("/category/{cid}")  @ResponseBody  **public** TaotaoResult getContentList(@PathVariable Long cid) {  TaotaoResult result = **null**;  **try** {  result = contentService.getContentList(cid);  } **catch** (Exception e) {  e.printStackTrace();  **return** TaotaoResult.*build*(500, e.getMessage());  }  **return** result;  }  } |

# 首页大广告方案

前端系统获取后端系统提供的接口，如何获取？

## 方案1

jsonp跨域请求



优点：

1. 效率高，没有通过后台中转
2. 减少内网的带宽开销

缺点：

网页中无内容，不利于搜索引擎优化。

## 方案二

通过后台java代码调用服务层



优点：

1. 网页中内容是变化的有利于搜索引擎优化

缺点：

1. 接口调用经过后台中转，效率较低，事实上可以忽略不计。

# Httpclient

## 什么是httpclient

HTTP 协议可能是现在 Internet 上使用得最多、最重要的协议了，越来越多的 Java 应用程序需要直接通过 HTTP 协议来访问网络资源。虽然在 JDK 的 java net包中已经提供了访问 HTTP 协议的基本功能，但是对于大部分应用程序来说，JDK 库本身提供的功能还不够丰富和灵活。HttpClient 是 Apache Jakarta Common 下的子项目，用来提供高效的、最新的、功能丰富的支持 HTTP 协议的客户端编程工具包，并且它支持 HTTP 协议最新的版本和建议。  
下载地址：

<http://hc.apache.org/>

## 功能介绍

以下列出的是 HttpClient 提供的主要的功能，要知道更多详细的功能可以参见 HttpClient 的主页。

（1）实现了所有 HTTP 的方法（GET,POST,PUT,HEAD 等）

（2）支持自动转向

（3）支持 HTTPS 协议

（4）支持代理服务器等

## 导入依赖



## 执行GET请求

|  |
| --- |
| **public** **class** DoGET {  **public** **static** **void** main(String[] args) **throws** Exception {  // 创建Httpclient对象  CloseableHttpClient httpclient = HttpClients.*createDefault*();  // 创建http GET请求  HttpGet httpGet = **new** HttpGet("http://www.baidu.com/");  CloseableHttpResponse response = **null**;  **try** {  // 执行请求  response = httpclient.execute(httpGet);  // 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("内容长度：" + content.length());  // FileUtils.writeStringToFile(new File("C:\\baidu.html"), content);  }  } **finally** {  **if** (response != **null**) {  response.close();  }  httpclient.close();  }  }  } |

## 执行GET带参数

|  |
| --- |
| **public** **class** DoGETParam {  **public** **static** **void** main(String[] args) **throws** Exception {  // 创建Httpclient对象  CloseableHttpClient httpclient = HttpClients.*createDefault*();  // 定义请求的参数  URI uri = **new** URIBuilder("http://www.baidu.com/s").setParameter("wd", "java").build();  System.***out***.println(uri);  // 创建http GET请求  HttpGet httpGet = **new** HttpGet(uri);  CloseableHttpResponse response = **null**;  **try** {  // 执行请求  response = httpclient.execute(httpGet);  // 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();  }  }  } |

## 执行post请求

|  |
| --- |
| **public** **class** DoPOST {  **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/");  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();  }  }  } |

## 带参数的post请求

|  |
| --- |
| **public** **class** DoPOSTParam {  **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");  // 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"));  // 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();  }  }  } |

## 封装HttpClient通用工具类

|  |
| --- |
| **public** **class** HttpClientUtil {  **public** **static** String doGet(String url, Map<String, String> param) {  // 创建Httpclient对象  CloseableHttpClient httpclient = HttpClients.*createDefault*();  String resultString = "";  CloseableHttpResponse response = **null**;  **try** {  // 创建uri  URIBuilder builder = **new** URIBuilder(url);  **if** (param != **null**) {  **for** (String key : param.keySet()) {  builder.addParameter(key, param.get(key));  }  }  URI uri = builder.build();  // 创建http GET请求  HttpGet httpGet = **new** HttpGet(uri);  // 执行请求  response = httpclient.execute(httpGet);  // 判断返回状态是否为200  **if** (response.getStatusLine().getStatusCode() == 200) {  resultString = EntityUtils.*toString*(response.getEntity(), "UTF-8");  }  } **catch** (Exception e) {  e.printStackTrace();  } **finally** {  **try** {  **if** (response != **null**) {  response.close();  }  httpclient.close();  } **catch** (IOException e) {  e.printStackTrace();  }  }  **return** resultString;  }  **public** **static** String doGet(String url) {  **return** *doGet*(url, **null**);  }  **public** **static** String doPost(String url, Map<String, String> param) {  // 创建Httpclient对象  CloseableHttpClient httpClient = HttpClients.*createDefault*();  CloseableHttpResponse response = **null**;  String resultString = "";  **try** {  // 创建Http Post请求  HttpPost httpPost = **new** HttpPost(url);  // 创建参数列表  **if** (param != **null**) {  List<NameValuePair> paramList = **new** ArrayList<>();  **for** (String key : param.keySet()) {  paramList.add(**new** BasicNameValuePair(key, param.get(key)));  }  // 模拟表单  UrlEncodedFormEntity entity = **new** UrlEncodedFormEntity(paramList);  httpPost.setEntity(entity);  }  // 执行http请求  response = httpClient.execute(httpPost);  resultString = EntityUtils.*toString*(response.getEntity(), "utf-8");  } **catch** (Exception e) {  e.printStackTrace();  } **finally** {  **try** {  response.close();  } **catch** (IOException e) {  // **TODO** Auto-generated catch block  e.printStackTrace();  }  }  **return** resultString;  }  **public** **static** String doPost(String url) {  **return** *doPost*(url, **null**);  }  } |

# 实现大广告位

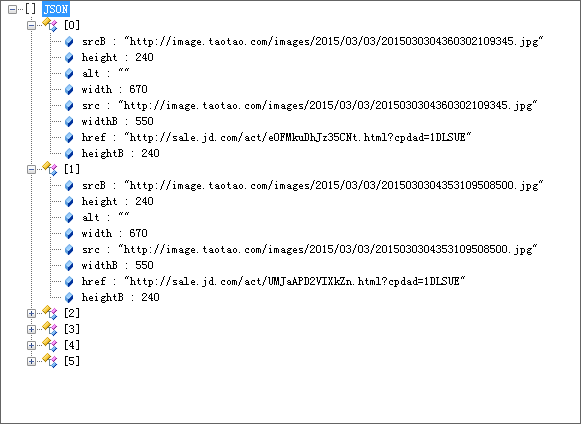
## 需求分析

在首页展示大广告位需要一个json数据来实现。

index.jsp



只需要生成此格式的json数据就可以了：



## Mapper

取广告位信息，不需要mapper。

## Service

调用服务层的服务根据内容分类id查询内容管理系统的内容。

需要一个描述数据结构的pojo类

|  |
| --- |
| **public** **class** ADItem {  **private** Integer height;  **private** Integer width;  **private** String src;  **private** Integer heightB;  **private** Integer widthB;  **private** String srcB;  **private** String alt;  **private** String href;  } |

|  |
| --- |
| @Service  **public** **class** ADServiceImpl **implements** AdService {  @Value("${REST\_BASE\_URL}")  **private** String REST\_BASE\_URL;  @Value("${INDEX\_AD1\_URL}")  **private** String INDEX\_AD1\_URL;    @Override  **public** String getAdItemList() **throws** Exception {  //调用服务层的服务查询打广告位的数据  String result = HttpClientUtil.*doGet*(REST\_BASE\_URL + INDEX\_AD1\_URL);  //把json数据转换成对象  TaotaoResult taotaoResult = TaotaoResult.*formatToList*(result, TbContent.**class**);  List<ADItem> itemList = **new** ArrayList<>();  **if** (taotaoResult.getStatus() == 200 ) {  List<TbContent> contentList = (List<TbContent>) taotaoResult.getData();  **for** (TbContent tbContent : contentList) {  ADItem item = **new** ADItem();  item.setHeight(240);  item.setWidth(670);  item.setSrc(tbContent.getPic());  item.setHeightB(240);  item.setWidth(550);  item.setSrcB(tbContent.getPic2());  item.setAlt(tbContent.getTitleDesc());  item.setHref(tbContent.getUrl());  itemList.add(item);  }    }  **return** JsonUtils.*objectToJson*(itemList);  }  } |

## Controller

|  |
| --- |
| @Controller  **public** **class** PageController {    @Autowired  **private** AdService adService;  @RequestMapping("/index")  **public** String showIndex(Model model) **throws** Exception {  String adResult = adService.getAdItemList();  model.addAttribute("ad1", adResult);    **return** "index";  }    } |

## Jsp页面

