# 商品添加实现

## 1.实现内容

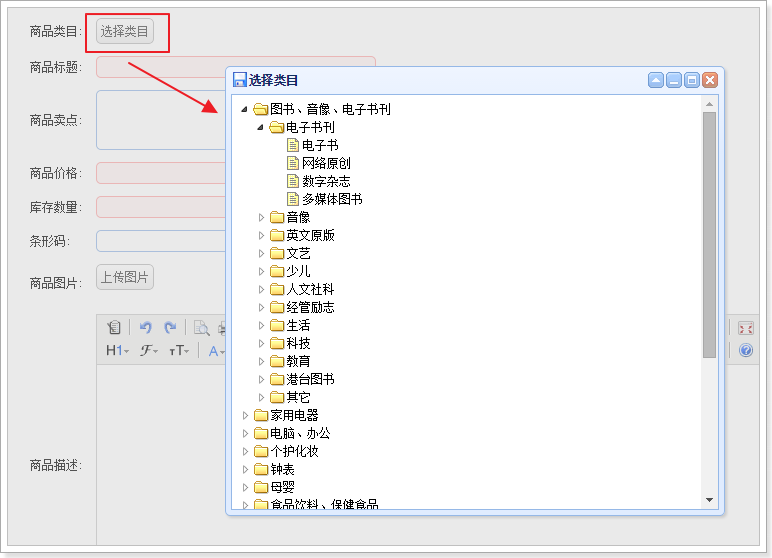
完成商品添加功能

* 商品类目选择（商品种类确定）
* 图片上传
* 图片服务器搭建
* kindEditor富文本编辑器的使用
* 商品添加功能

## 2.实现商品类目选择功能

### 需求分析：

在商品添加页面，点击“选择类目”显示商品类目列表：



### 实现步骤：

按钮添加点击事件，弹出窗口，加载数据显示tree

将选择类目的组件封装起来，通过TT.iniit()初始化，最终调用initItemCat()方法进行初始化

创建数据库、以及tb \_item\_cat表，初始化数据

编写Controller、Service、Mapper

#### EasyUI tree数据结构分析

数据结构中必须包含：

Id：节点id

Text：节点名称

State：如果不是叶子节点就是close，叶子节点就是open。Close的节点点击后会在此发送请求查询子项目。



可以根据parentid查询分类列表。

#### Mapper

使用逆向工程生成的mapper文件

#### Service

功能：根据parentId查询商品分类列表

参数：parentId

返回值：返回tree所需要的数据结构，是一个节点列表。

可以创建一个tree node的pojo表示节点的数据，也可以使用map。

List<TreeNode>

##### 创建一个tree node的pojo

**public** **class** TreeNode {

**private** **long** id;

**private** String text;

**private** String state;

**public** TreeNode(**long** id, String text, String state) {

**this**.id = id;

**this**.text = text;

**this**.state = state;

}

**public** **long** getId() {

**return** id;

}

**public** **void** setId(**long** id) {

**this**.id = id;

}

**public** String getText() {

**return** text;

}

**public** **void** setText(String text) {

**this**.text = text;

}

**public** String getState() {

**return** state;

}

**public** **void** setState(String state) {

**this**.state = state;

}

}

##### Service实现

|  |
| --- |
| ItemCatService接口：  **public** **interface** ItemCatService {  **public** List<TreeNode> getItemCatList(**long** parentId);  }  ItemCatServiceImpl实现类：  @Service  **public** **class** ItemCatServiceImpl **implements** ItemCatService {  @Autowired  **private** TbItemCatMapper itemCatMapper;  @Override  **public** List<TreeNode> getItemCatList(**long** parentId) {  //根据parentId查询分类列表  TbItemCatExample example = **new** TbItemCatExample();  //设置查询条件  Criteria criteria = example.createCriteria();  criteria.andParentIdEqualTo(parentId);  //执行查询  List<TbItemCat> list = itemCatMapper.selectByExample(example);  //分类列表转换成TreeNode的列表  List<TreeNode> resultList = **new** ArrayList<>();  **for** (TbItemCat tbItemCat : list) {  //创建一个TreeNode对象  TreeNode node = **new** TreeNode(tbItemCat.getId(), tbItemCat.getName(),  tbItemCat.getIsParent()?"closed":"open");  resultList.add(node);  }  **return** resultList;  }  } |

#### Controller

功能：接收页面传递过来的id，作为parentId查询子节点。

参数：Long id

返回值：要返回json数据要使用@ResponseBody。List<TreeNode>

|  |
| --- |
| @Controller  @RequestMapping("/item/cat")  **public** **class** ItemCatController {  @Autowired  **private** ItemCatService itemCatService;  @RequestMapping("/list")  @ResponseBody  **public** List<TreeNode> getItemCatList(@RequestParam(value="id", defaultValue="0")Long parentId) {  List<TreeNode> list = itemCatService.getItemCatList(parentId);  **return** list;  }  } |

## 3.图片上传

### <1>图片服务器

#### 传统项目中的图片管理

传统项目中，可以在web项目中添加一个文件夹，来存放上传的图片。例如在工程的根目录WebRoot下创建一个images文件夹。把图片存放在此文件夹中就可以直接使用在工程中引用。

优点：

引用方便，便于管理

缺点：

如果是分布式环境图片引用会出现问题

图片的下载会给服务器增加额外的压力

WEB工程

图片文件夹

例如：WebRoot/images

传统图片管理方式在分布式环境中的问题：

第一次请求：图片上传

用户请求

Tomcat1

Image（存放图片）

|-image.jpg

第二次请求：

下载图片

负载均衡服务器

Tomcat2

Image（存放图片）

说好的图片呢？？？？？

#### 分布式环境的图片管理

Tomcat1

~~Image（存放图片）~~

~~|-image.jpg~~

Tomcat2

~~Image（存放图片）~~

负载均衡服务器

用户请求

http图片上传

图片服务器

Ip或者域名

Image.taotao.com  
nginx实现http服务

/images/2015/07/29/image.jpg

ftp服务接收上传

ftp上传到图片服务器

http请求图片

http图片上传

分布式环境一般都有一个专门的图片服务器存放图片。

我们使用虚拟机搭建一个专门的服务器来存放图片。在此服务器上安装一个nginx来提供http服务，安装一个ftp服务器来提供图片上传服务。

#### 搭建图片服务器

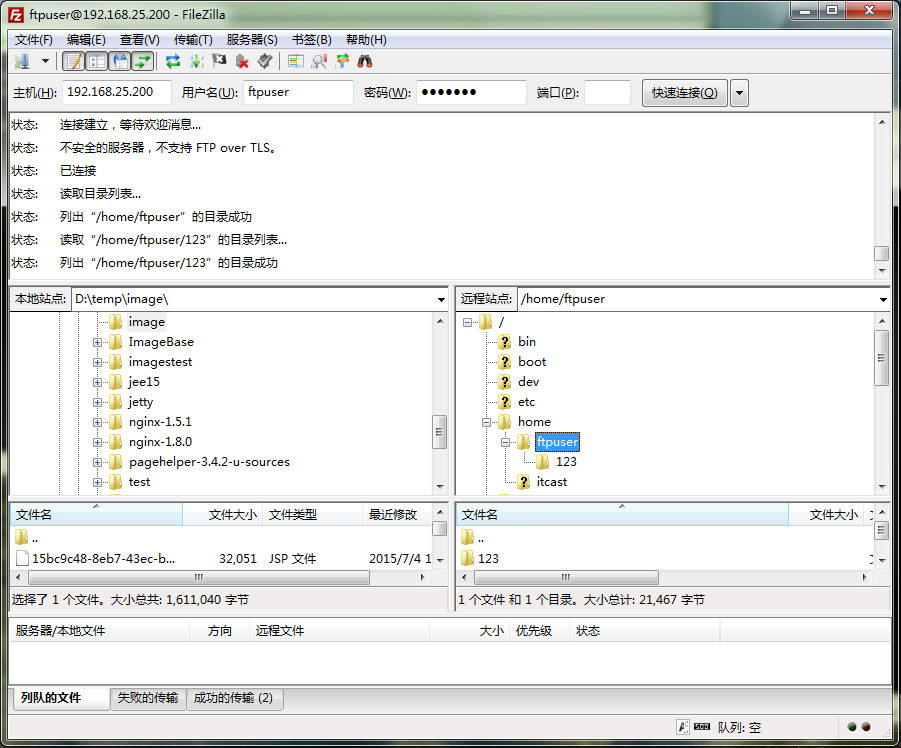
第一步：安装vsftpd提供ftp服务（详见：vsftpd安装手册.doc）

第二步：安装nginx提供http服务（详见：nginx安装手册.doc）

#### 测试图片服务器

##### ftp服务测试

使用ftp客户端



使用java程序

ftp可以需要依赖commons-net-3.3.jar包。

|  |
| --- |
| **public** **static** **void** main(String[] args) **throws** Exception {  FTPClient ftpClient = **new** FTPClient();  ftpClient.connect("192.168.25.200");  ftpClient.login("ftpuser", "ftpuser");  FileInputStream inputStream = **new** FileInputStream(**new** File("D:\\Documents\\Pictures\\pics\\21.jpg"));  ftpClient.setFileType(FTP.***BINARY\_FILE\_TYPE***);  ftpClient.storeFile("123.jpg", inputStream);  inputStream.close();    ftpClient.logout();  } |

##### http服务测试

浏览器测试

##### FastDFS文件上传测试

一般在taotao-parent定义依赖版本，有时候

在taotao-manager-web下通过Maven中引入相关jar包(需要将jar导入maven仓库，中央仓库可能没有该jar包的存在)

  <dependency>  
   <groupId>org.csource</groupId>  
 <artifactId>fastdfs-client-java</artifactId>  
 <version>1.27-SNAPSHOT</version>  
  </dependency>

编写文件client.conf：tracker\_server=10.0.7.68:22122

在taotao-common下加载工具类（相应地也要修改pom.xml）

编写测试代码测试服务器连接：

@Test

**public** **void** testUpload() **throws** Exception {

// 1、把FastDFS提供的jar包添加到工程中

// 2、初始化全局配置,加载一个配置文件

// String url = "e:/workspace/eclipse/Projects/taotao-manager/taotao-manager-web/src/main/resources/conf/client.conf";

String url = "E:\\workspace\\eclipse\\Projects\\taotao-manager\\taotao-manager-web\\src\\main\\resources\\conf\\client.conf";

ClientGlobal.*init*(url);

// 3、创建一个TrackerClient对象。

TrackerClient trackerClient = **new** TrackerClient();

// 4、创建一个TrackerServer对象。

TrackerServer trackerServer = trackerClient.getConnection();

// 5、声明一个StorageServer对象，null。

StorageServer storageServer = **null**;

// 6、获得StorageClient对象。

StorageClient storageClient = **new** StorageClient(trackerServer, storageServer);

// 7、直接调用StorageClient对象方法上传文件即可。

String source= "C:\\Users\\utry\\Desktop\\image\\1.png";

String[] strings = storageClient.upload\_file(source, "png", **null**);

**for** (String string : strings) {

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

}

}

#### SpringMVC中实现图片上传

上传思路：

第一步：

导入common-fileupload的依赖

|  |
| --- |
| <!-- 文件上传组件 -->  <dependency>  <groupId>commons-fileupload</groupId>  <artifactId>commons-fileupload</artifactId>  </dependency> |

第二步：

在SpringMVC配置文件中添加文件上传解析器

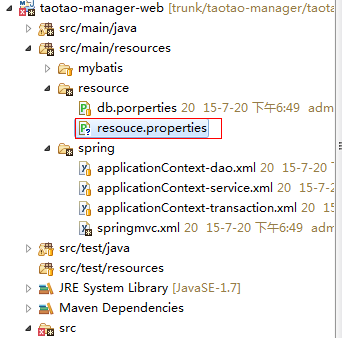
|  |
| --- |
| <!-- 定义文件上传解析器 -->  <bean id=*"multipartResolver"*  class=*"org.springframework.web.multipart.commons.CommonsMultipartResolver"*>  <!-- 设定默认编码 -->  <property name=*"defaultEncoding"* value=*"UTF-8"*></property>  <!-- 设定文件上传的最大值5MB，5\*1024\*1024 -->  <property name=*"maxUploadSize"* value=*"5242880"*></property>  </bean> |

#### Service实现

##### 获取资源配置文件的内容

第一步：

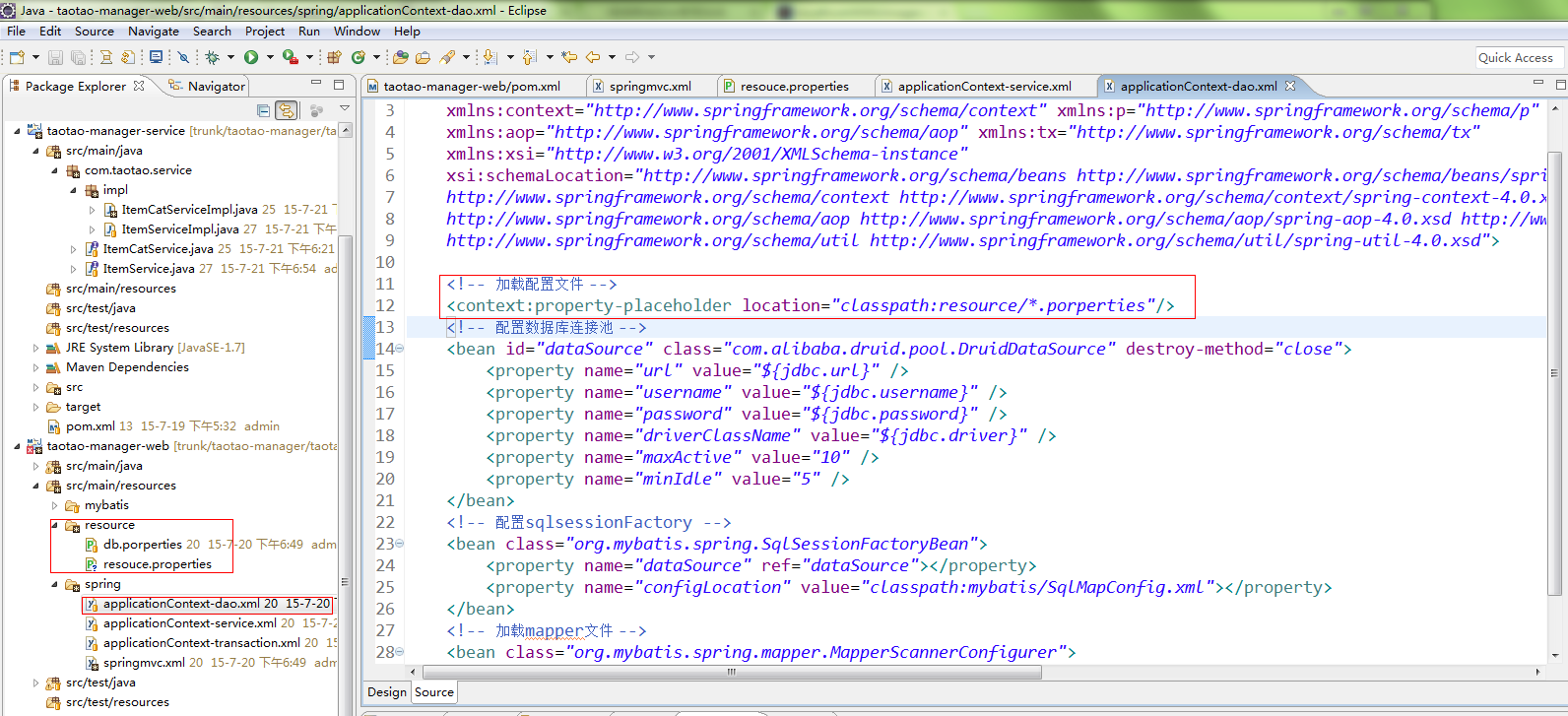
创建资源配置文件



|  |
| --- |
| FILI\_UPLOAD\_PATH=D:/temp/imagestest/webapps/images  IMAGE\_BASE\_URL=http://localhost:9000/images |

第二步：

在Spring（taotao-manage-servlet.xml）容器中加载资源文件



第三步：

在Service中获取资源配置：

|  |
| --- |
| @Value("${FILI\_UPLOAD\_PATH}")  **private** String FILI\_UPLOAD\_PATH;  @Value("${IMAGE\_BASE\_URL}")  **private** String IMAGE\_BASE\_URL; |

##### 图片名生成策略

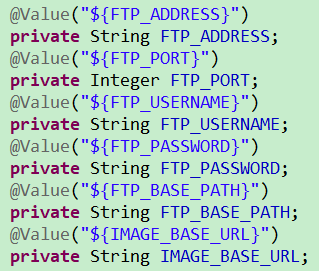
时间+随机数：

|  |
| --- |
| /\*\*  \* 图片名生成  \*/  **public** **static** String genImageName() {  //取当前时间的长整形值包含毫秒  **long** millis = System.*currentTimeMillis*();  //long millis = System.nanoTime();  //加上三位随机数  Random random = **new** Random();  **int** end3 = random.nextInt(999);  //如果不足三位前面补0  String str = millis + String.*format*("%03d", end3);    **return** str;  } |

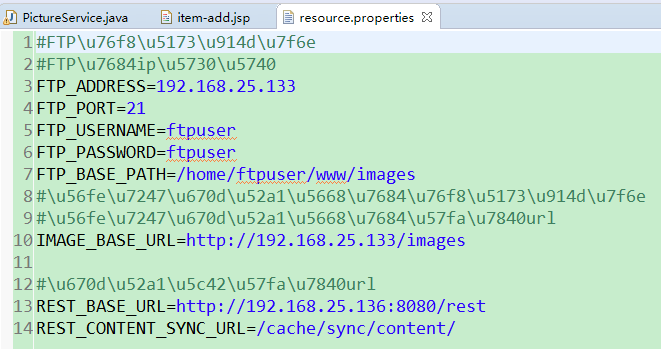
使用UUID：

|  |
| --- |
| UUID.randomUUID(); |

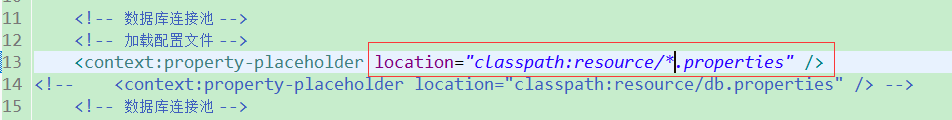
##### Service实现



读取配置文件中的数据（设置文件上传所需的基本属性），在taotao-manager-web下配置属性文件。



利用spring框架提供的读取文件的机制完成文件信息的读取，在applicationContext-dao.xml下加载相关的配置文件

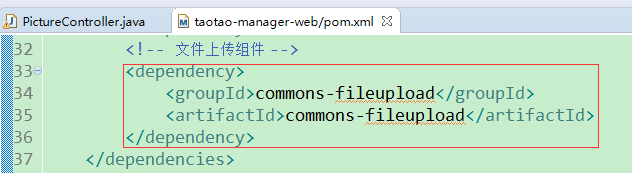


|  |
| --- |
| **public** **class** PictureServiceImpl **implements** PictureService{  @Value("${FTP\_ADDRESS}")  **private** String FTP\_ADDRESS;  @Value("${FTP\_PORT}")  **private** Integer FTP\_PORT;  @Value("${FTP\_USERNAME}")  **private** String FTP\_USERNAME;  @Value("${FTP\_PASSWORD}")  **private** String FTP\_PASSWORD;  @Value("${FTP\_BASE\_PATH}")  **private** String FTP\_BASE\_PATH;  @Value("${IMAGE\_BASE\_URL}")  **private** String IMAGE\_BASE\_URL;  @Override  **public** Map uploadPicture(MultipartFile uploadFile) {  Map resultMap = **new** HashMap<>();  **try** {  //生成一个新的文件名  //取原始文件名  String oldName = uploadFile.getOriginalFilename();  //生成新文件名  //UUID.randomUUID();  String newName = IDUtils.*genImageName*();  newName = newName + oldName.substring(oldName.lastIndexOf("."));  //图片上传  String imagePath = **new** DateTime().toString("/yyyy/MM/dd");  **boolean** result = FtpUtil.*uploadFile*(FTP\_ADDRESS, FTP\_PORT, FTP\_USERNAME, FTP\_PASSWORD,  FTP\_BASE\_PATH, imagePath, newName, uploadFile.getInputStream());  //返回结果  **if**(!result) {  resultMap.put("error", 1);  resultMap.put("message", "文件上传失败");  **return** resultMap;  }  resultMap.put("error", 0);  resultMap.put("url", IMAGE\_BASE\_URL + imagePath + "/" + newName);  **return** resultMap;    } **catch** (Exception e) {  resultMap.put("error", 1);  resultMap.put("message", "文件上传发生异常");  **return** resultMap;  }  }  } |

#### Controller实现

功能：接收MultiPartFile对象，调用Service上传图片，返回json数据格式，使用@ResponseBody注解。

此前需要在springmvc.xml中配置多部件解析器，





<!-- 定义文件上传解析器 -->

<bean id=*"multipartResolver"*

class=*"org.springframework.web.multipart.commons.CommonsMultipartResolver"*>

<!-- 设定默认编码 -->

<property name=*"defaultEncoding"* value=*"UTF-8"*></property>

<!-- 设定文件上传的最大值5MB，5\*1024\*1024 -->

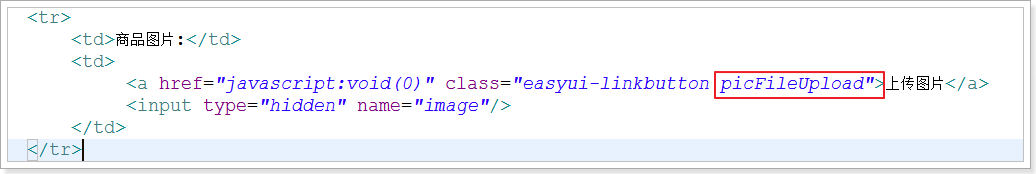
<property name=*"maxUploadSize"* value=*"5242880"*></property>

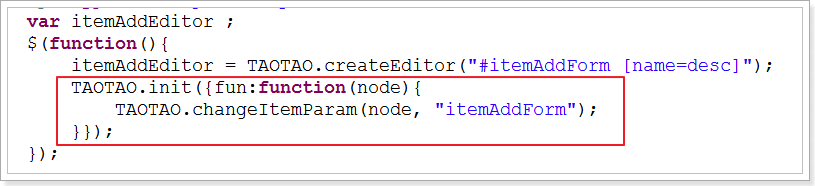
</bean>

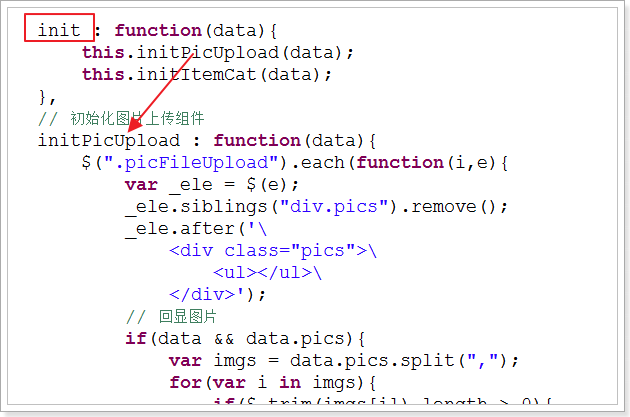
|  |
| --- |
| @Controller  **public** **class** PictureController {  @Autowired  **private** PictureService pictureService;  @RequestMapping("/pic/upload")  @ResponseBody  **public** String pictureUpload(MultipartFile uploadFile) {  Map result = pictureService.uploadPicture(uploadFile);  //为了保证功能的兼容性，需要把Result转换成json格式的字符串。  String json = JsonUtils.*objectToJson*(result);  **return** json;  }  } |

#### 前端JS实现图片上传

##### Js实现逻辑







KindEditor 4.x 文档

<http://kindeditor.net/doc.php>

上传图片使用kindeditor的上传组件实现。



##### 上传图片请求url：



##### 返回值

参考文档：

<http://kindeditor.net/docs/upload.html>

返回格式(JSON)

|  |
| --- |
| //成功时  {  "error" : 0,  "url" : "http://www.example.com/path/to/file.ext"  }  //失败时  {  "error" : 1,  "message" : "错误信息"  } |

返回值数据类型：

|  |
| --- |
| **public** **class** PictureResult {  /\*\*  \* 上传图片返回值，成功：0 失败：1  \*/  **private** Integer error;  /\*\*  \* 回显图片使用的url  \*/  **private** String url;  /\*\*  \* 错误时的错误消息  \*/  } |

文件上传可能存在的问题：

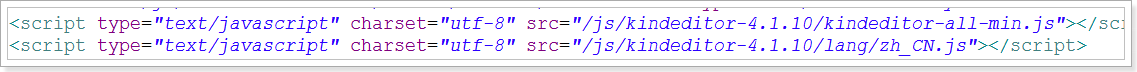
文件服务器部署存在问题，考虑调整为存储到本地的服务器

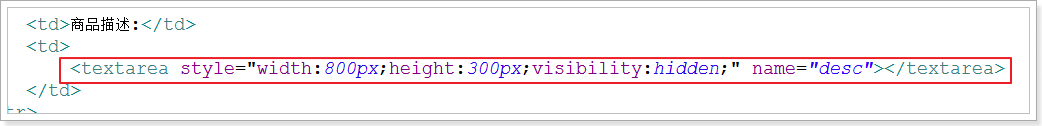
文件上传插件：存在浏览器兼容问题，在google能正常使用，在firefox无法正常使用（可以通过f12查看相关的json数据，考虑这一问题（保证功能的兼容性），在controller层需要将Result转化为相应的json格式的字符串数据）

## 4.kindeditor（富文本编辑器）的使用

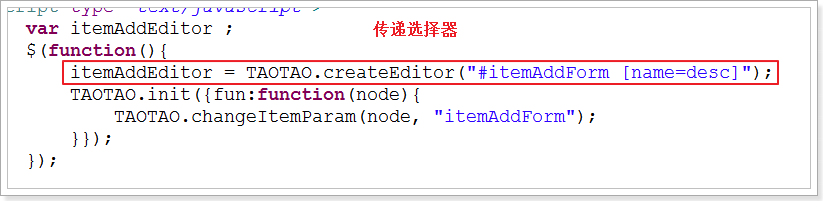
### kindeditor的使用过程：

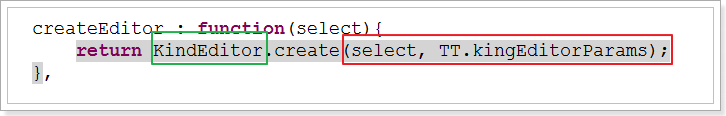
步骤1：导入js：（在item.add.jsp中引入）



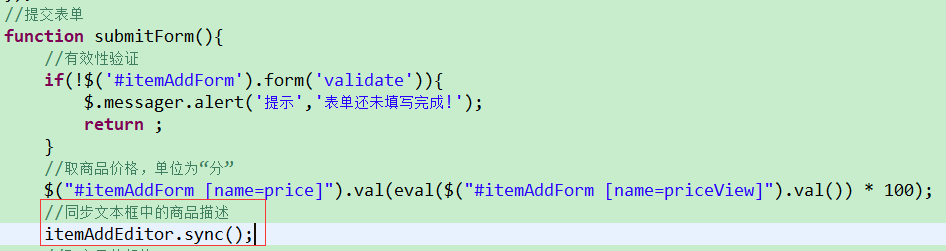
步骤2：定义多行文本（设置不可见、给定name）  


步骤3：调用TT.createEditor（查看common.js）

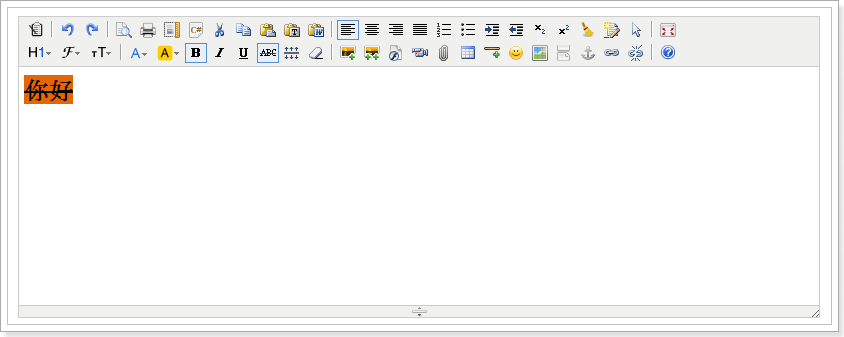
创建富文本编辑器，初始化类目选择和图片上传器  




提交表单之前，需要先将富文本编辑器中的内容和textarea中的内容进行同步



步骤4：测试效果



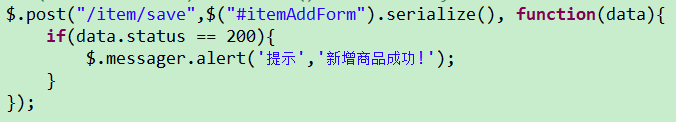
取文本编辑器中的内容：将编辑器的内容设置到原来的textarea控件里

editor**.**sync**();**

## 5.新增商品实现

### js编写逻辑

表单提交请求：（请求url、表单数据序列化成key-value形式）



|  |
| --- |
| //提交表单  **function** submitForm(){  //有效性验证  **if**(!$('#itemAddForm').form('validate')){  $.messager.alert('提示','表单还未填写完成!');  **return** ;  }  //取商品价格，单位为“分”  $("#itemAddForm [name=price]").val(eval($("#itemAddForm [name=priceView]").val()) \* 100);  //同步文本框中的商品描述  itemAddEditor.sync();  //取商品的规格  /\*  var paramJson = [];  $("#itemAddForm .params li").each(function(i,e){  var trs = $(e).find("tr");  var group = trs.eq(0).text();  var ps = [];  for(var i = 1;i<trs.length;i++){  var tr = trs.eq(i);  ps.push({  "k" : $.trim(tr.find("td").eq(0).find("span").text()),  "v" : $.trim(tr.find("input").val())  });  }  paramJson.push({  "group" : group,  "params": ps  });  });  //把json对象转换成字符串  paramJson = JSON.stringify(paramJson);  $("#itemAddForm [name=itemParams]").val(paramJson);  \*/  //ajax的post方式提交表单  //$("#itemAddForm").serialize()将表单序列号为key-value形式的字符串  $.post("/item/save",$("#itemAddForm").serialize(), **function**(data){  **if**(data.status == 200){  $.messager.alert('提示','新增商品成功!');  }  });  } |

### 提交请求的数据格式

$("#itemAddForm").serialize()将表单序列号为key-value形式的字符串，以post 的形式将表单的内容提交。

请求的url：/item/save

返回的结果：淘淘自定义返回结果：

状态码、响应的消息、响应的数据

|  |
| --- |
| /\*\*  \* 淘淘商城自定义响应结构  \*/  **public** **class** TaotaoResult {  // 定义jackson对象  **private** **static** **final** ObjectMapper ***MAPPER*** = **new** ObjectMapper();  // 响应业务状态  **private** Integer status;  // 响应消息  **private** String msg;  // 响应中的数据  **private** Object data;  **public** **static** TaotaoResult build(Integer status, String msg, Object data) {  **return** **new** TaotaoResult(status, msg, data);  }  **public** **static** TaotaoResult ok(Object data) {  **return** **new** TaotaoResult(data);  }  **public** **static** TaotaoResult ok() {  **return** **new** TaotaoResult(**null**);  }  **public** TaotaoResult() {  }  **public** **static** TaotaoResult build(Integer status, String msg) {  **return** **new** TaotaoResult(status, msg, **null**);  }  **public** TaotaoResult(Integer status, String msg, Object data) {  **this**.status = status;  **this**.msg = msg;  **this**.data = data;  }  **public** TaotaoResult(Object data) {  **this**.status = 200;  **this**.msg = "OK";  **this**.data = data;  }  // public Boolean isOK() {  // return this.status == 200;  // }  **public** Integer getStatus() {  **return** status;  }  **public** **void** setStatus(Integer status) {  **this**.status = status;  }  **public** String getMsg() {  **return** msg;  }  **public** **void** setMsg(String msg) {  **this**.msg = msg;  }  **public** Object getData() {  **return** data;  }  **public** **void** setData(Object data) {  **this**.data = data;  }  /\*\*  \* 将json结果集转化为TaotaoResult对象  \*  \* **@param** jsonData json数据  \* **@param** clazz TaotaoResult中的object类型  \* **@return**  \*/  **public** **static** TaotaoResult formatToPojo(String jsonData, Class<?> clazz) {  **try** {  **if** (clazz == **null**) {  **return** ***MAPPER***.readValue(jsonData, TaotaoResult.**class**);  }  JsonNode jsonNode = ***MAPPER***.readTree(jsonData);  JsonNode data = jsonNode.get("data");  Object obj = **null**;  **if** (clazz != **null**) {  **if** (data.isObject()) {  obj = ***MAPPER***.readValue(data.traverse(), clazz);  } **else** **if** (data.isTextual()) {  obj = ***MAPPER***.readValue(data.asText(), clazz);  }  }  **return** *build*(jsonNode.get("status").intValue(), jsonNode.get("msg").asText(), obj);  } **catch** (Exception e) {  **return** **null**;  }  }  /\*\*  \* 没有object对象的转化  \*  \* **@param** json  \* **@return**  \*/  **public** **static** TaotaoResult format(String json) {  **try** {  **return** ***MAPPER***.readValue(json, TaotaoResult.**class**);  } **catch** (Exception e) {  e.printStackTrace();  }  **return** **null**;  }  /\*\*  \* Object是集合转化  \*  \* **@param** jsonData json数据  \* **@param** clazz 集合中的类型  \* **@return**  \*/  **public** **static** TaotaoResult formatToList(String jsonData, Class<?> clazz) {  **try** {  JsonNode jsonNode = ***MAPPER***.readTree(jsonData);  JsonNode data = jsonNode.get("data");  Object obj = **null**;  **if** (data.isArray() && data.size() > 0) {  obj = ***MAPPER***.readValue(data.traverse(),  ***MAPPER***.getTypeFactory().constructCollectionType(List.**class**, clazz));  }  **return** *build*(jsonNode.get("status").intValue(), jsonNode.get("msg").asText(), obj);  } **catch** (Exception e) {  **return** **null**;  }  }  } |

### 获得商品id

临时主键生成策略：

|  |
| --- |
| /\*\*  \* 商品id生成  \*/  **public** **static** **long** genItemId() {  //取当前时间的长整形值包含毫秒  **long** millis = System.*currentTimeMillis*();  //long millis = System.nanoTime();  //加上两位随机数  Random random = **new** Random();  **int** end2 = random.nextInt(99);  //如果不足两位前面补0  String str = millis + String.*format*("%02d", end2);  **long** id = **new** Long(str);  **return** id;  } |

### ItemServiceImpl

调用mapper的insert方法添加商品信息

|  |
| --- |
| @Override  **public** **void** saveItem(TbItem item, String desc, String itemParams) **throws** Exception {  Date date = **new** Date();  //获得商品id  **long** id = IDUtils.*genItemId*();  //添加商品信息  item.setId(id);  //商品状态，1-正常，2-下架，3-删除  item.setStatus((**byte**) 1);  item.setCreated(date);  item.setUpdated(date);  itemMapper.insert(item);  //添加商品描述  //创建TbItemDesc对象  TbItemDesc itemDesc = **new** TbItemDesc();  //获得一个商品id  itemDesc.setItemId(id);  itemDesc.setItemDesc(desc);  itemDesc.setCreated(date);  itemDesc.setUpdated(date);  //插入数据  itemDescMapper.insert(itemDesc);    } |

### Controller实现

|  |
| --- |
| @RequestMapping("/item/save")  @ResponseBody  **public** TaotaoResult saveItem(TbItem item, String desc) **throws** Exception {  //添加商品信息  itemService.saveItem(item, desc, **null**);  **return** TaotaoResult.*ok*();  } |

在此基础上完成商品信息的修改

## 6.fastDFS实现文件上传

参考已有的虚拟机服务器，修改相关文件：

修改/etc/fdfs/storage.conf和/etc/fdfs/client.conf文件，修改对应的ip和端口号信息：

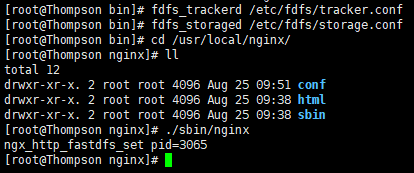
tracker\_server=192.168.187.128:22122

重新启动fastDFS、nginx使配置生效

cd/usr/bin/



重启nginx



再次测试访问链接：

<http://192.168.187.128/group1/M00/00/00/wKi7gFwu_ZyAUs-BAAHUEC1DZhw324.png>

使用fastDFS工具类实现图片上传：

@Test

**public** **void** testFastDfsClient() **throws** Exception {

String url = "E:\\workspace\\eclipse\\Projects\\taotao-manager\\taotao-manager-web\\src\\main\\resources\\conf\\client.conf";

FastDFSClient client = **new** FastDFSClient(url);

String source = "C:\\Users\\utry\\Desktop\\image\\2.jpg";

String uploadFile = client.uploadFile(source, "jpg");

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

}

上传的图片在服务器中存储的位置：

/home/fastdfs/storage/data/00/00

上传测试：第一次上传测试需要加载flash插件，否则可能出现“添加图片”的按钮不显示，其次添加图片完成之后，图片数据不回显，如下图所示

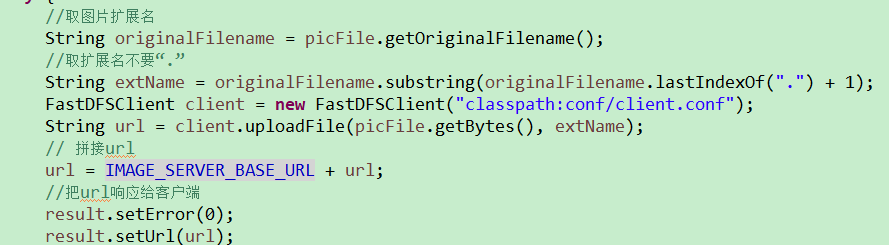


右键选择“检查元素”查看相应的数据，显示如下，发现文件路径出错，没有拼接相关的ip地址



在返回相应url时拼接相应的ip





### 火狐浏览器兼容性问题解决

要求返回的数据是一个文本类型，要求content-type 为text/plan

