**淘淘商城**

**第四天**

# 课程计划

1. FastDFS介绍
2. FastDFS使用
3. 实现图片上传
4. 实现商品列表查询

# FastDFS介绍

## 图片服务器介绍

要实现图片上传功能，需要有一个图片服务器。图片服务器的特点：

1. 存储空间可扩展
2. 提供统一的访问方式
3. 访问效率高

原来使用Tomcat配置虚拟路径的方式不能满足需求。

企业中的图片服务器有两个方案：

1. 使用FastDFS分布式文件系统。

2. 使用Nginx服务器。

## 什么是FastDFS

FastDFS是用c语言编写的一款开源的分布式文件系统。FastDFS为互联网量身定制，充分考虑了冗余备份、负载均衡、横向扩展等机制，并注重高可用、高性能等指标，使用FastDFS很容易搭建一套高性能的文件服务器集群提供文件上传、下载等服务。

## FastDFS架构

FastDFS架构包括 Tracker server和Storage server。客户端请求Tracker server进行文件上传、下载，通过Tracker server调度最终由Storage server完成文件上传和下载。

Tracker server作用是负载均衡和调度，通过Tracker server在文件上传时可以根据一些策略找到Storage server提供文件上传服务。可以将tracker称为追踪服务器或调度服务器。

Storage server作用是文件存储，客户端上传的文件最终存储在Storage服务器上，Storage server没有实现自己的文件系统而是利用操作系统 的文件系统来管理文件。可以将storage称为存储服务器。

如下图：



Tracker 管理集群

收集信息，处理信息

为了保证高可用，可以搭建集群

Storage 保存文件

分为很多组，组和组之间的数据不一样

组内成员数据是一样的，保证数据的高可用

可以增加组，达到扩容的效果，

## 文件上传流程

文件上传流程如下图（时序图）：



客户端上传文件后存储服务器将文件ID返回给客户端，此文件ID用于以后访问该文件的索引信息。文件索引信息包括：组名，虚拟磁盘路径，数据两级目录，文件名。



* 组名：文件上传后所在的storage组名称，在文件上传成功后有storage服务器返回，需要客户端自行保存。
* 虚拟磁盘路径：storage配置的虚拟路径，与磁盘选项store\_path\*对应。如果配置了store\_path0则是M00，如果配置了store\_path1则是M01，以此类推。
* 数据两级目录：storage服务器在每个虚拟磁盘路径下创建的两级目录，用于存储数据文件。两级目录的范围都是 00~FF，备注(16进制中FF等于255，加上从00开始，也就是说第一级目录可以放255+1个目录，二级目录可放个数为：256\*256 =65,536个文件夹目录，为保证文件访问速度，粗略按照一个目录2,000个文件存放<00-FF>总共可保存文件: 131,072,000个)。
* 文件名：与文件上传时不同。是由存储服务器根据特定信息生成，文件名包含：源存储服务器IP地址、文件创建时间戳、文件大小、随机数和文件拓展名等信息。

## 文件下载流程

文件下载流程如下图



tracker根据请求的文件路径即文件ID 来快速定义文件。

比如请求下边的文件：



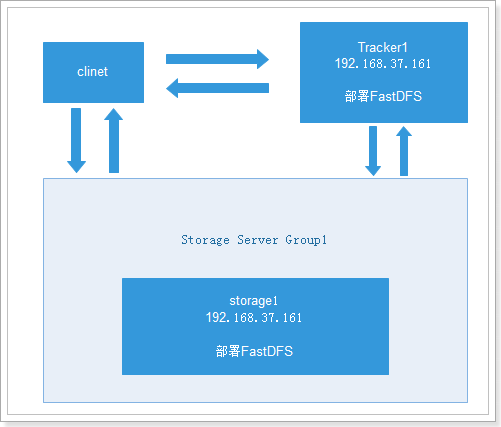
1.通过组名tracker能够很快的定位到客户端需要访问的存储服务器组是group1，并选择合适的存储服务器提供客户端访问。

2.存储服务器根据“文件存储虚拟磁盘路径”和“数据文件两级目录”可以很快定位到文件所在目录，并根据文件名找到客户端需要访问的文件。

# 实现文件服务器

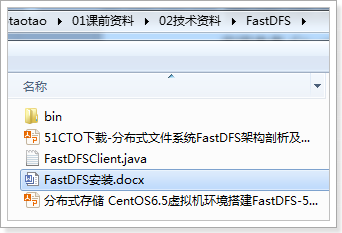
## 架构图

本课程使用的FastDFS安装架构图如下：



## 安装FastDFS

学员自己安装参考《FastDFS安装.docx》，文档位置如下：



课程使用已经准备好的FastDFS服务。

导入步骤参考第二天讲义

### 测试FastDFS是否可用

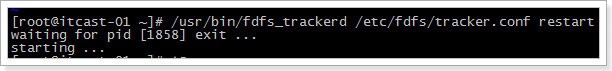
Linux账号:root 密码:root

#### 确认服务是否可用

确认tracker是否启动

[root@itcast-01 ~]# /usr/bin/fdfs\_trackerd /etc/fdfs/tracker.conf restart

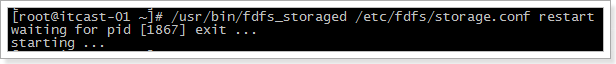
如下图关闭线程再执行则表示成功



确认storage是否启动

[root@itcast-01 ~]# /usr/bin/fdfs\_storaged /etc/fdfs/storage.conf restart

如下图关闭线程再执行则表示成功



如果不成功，同样的命令再执行一次

确认Nginx是否可用

[root@itcast-01 ~]# ps aux | grep nginx

查看线程是否存在

#### 测试

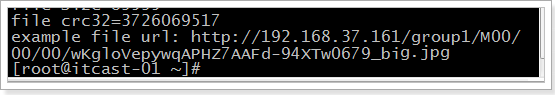
上传文件到Linux

按alt+p进入上传界面进行上传，上传一张图片

把图片上传到FastDFS中，上传命令如下，最后的文件名就是要上传的图片：

[root@itcast-01 ~]# /usr/bin/fdfs\_test /etc/fdfs/client.conf upload 394321.jpg

出现以下url地址表示上传成功。



查看文件，成功后图片保存在linux的/home/fastdfs/fdfs\_storage/data/00/00目录中，此目录的定义在安装storage服务器时配置。

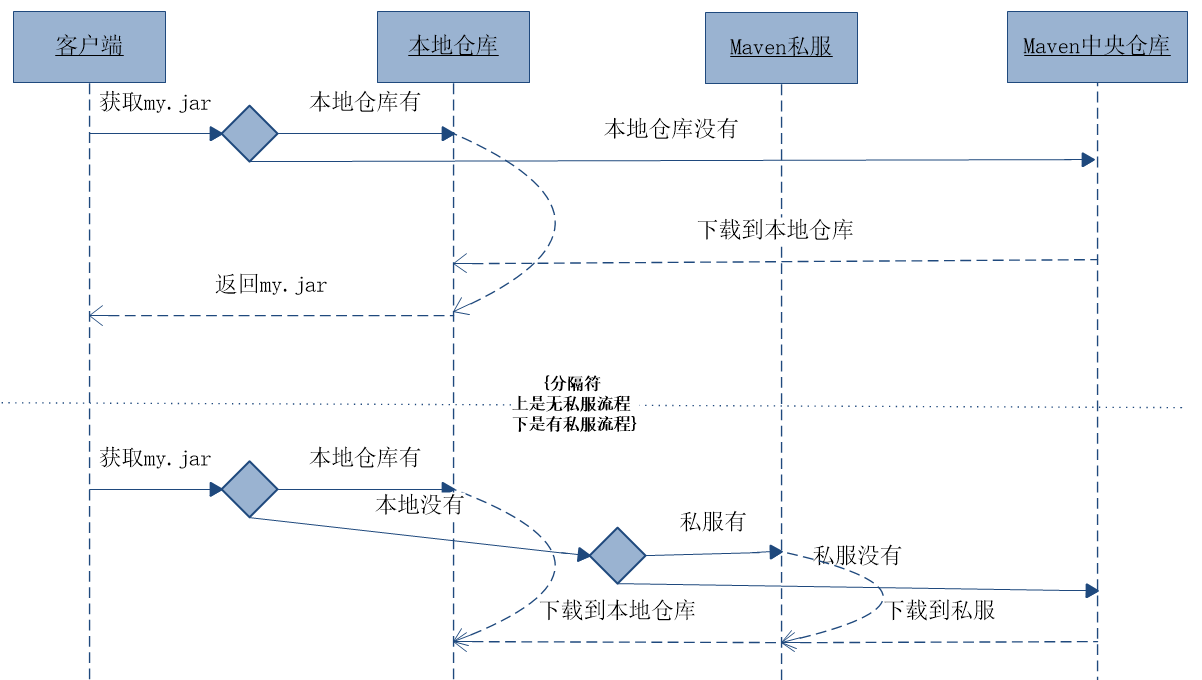
把刚刚生成的url地址在浏览器输入

如果显示成功则表示FastDFS可以使用



# 图片服务器的使用

我们一般使用java客户端进行上传图片，而使用图片服务器需要相关的依赖，但是中央仓库没有FastDFS的客户端依赖。



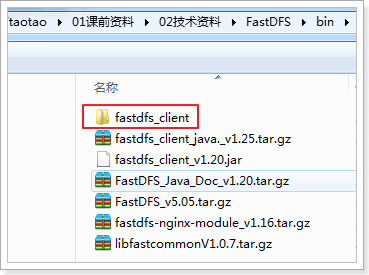
在企业中有Maven私服，我们可以把FastDFS的java客户端依赖手动添加到Maven私服的第三方库，直接加入坐标使用。

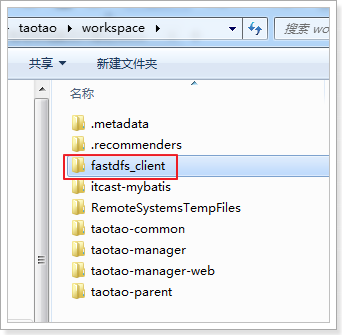
我们这里没有Maven私服，需要手动添加依赖，安装到本地仓库

## 添加依赖

### 复制工程

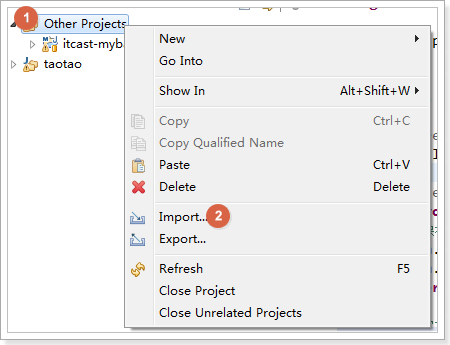
如下复制课后资料的工程到本地eclipse的工作空间中



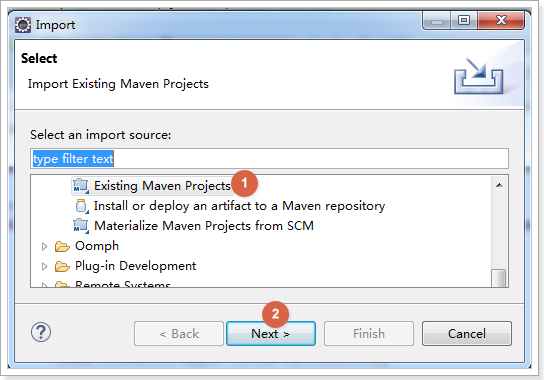


### 导入工程

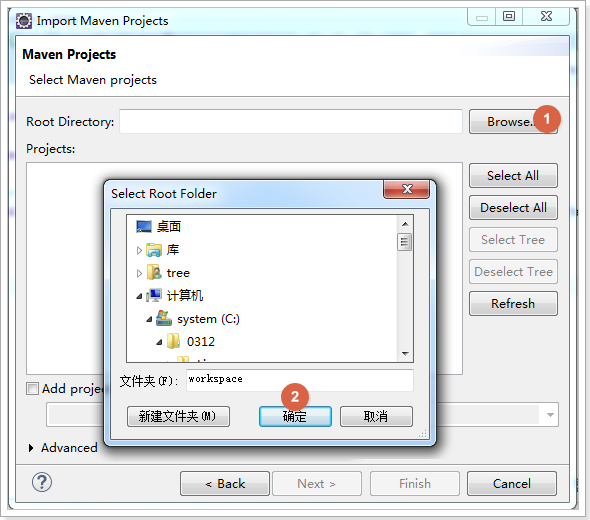
如下图在Other Project上点击右键，选择Import

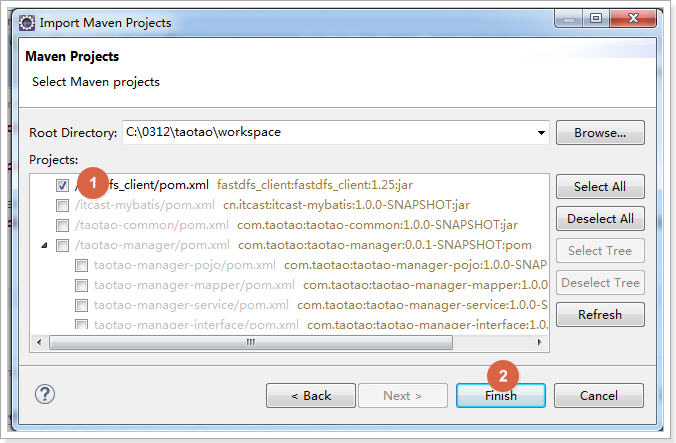


选择已存在的Maven工程，如下图



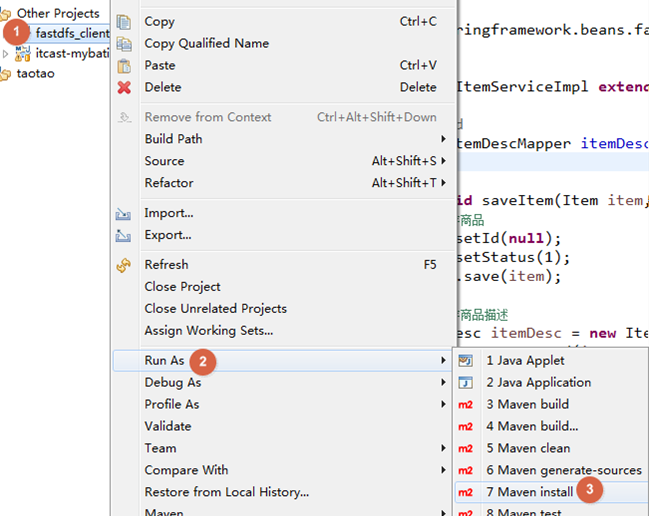
如下图进行导入





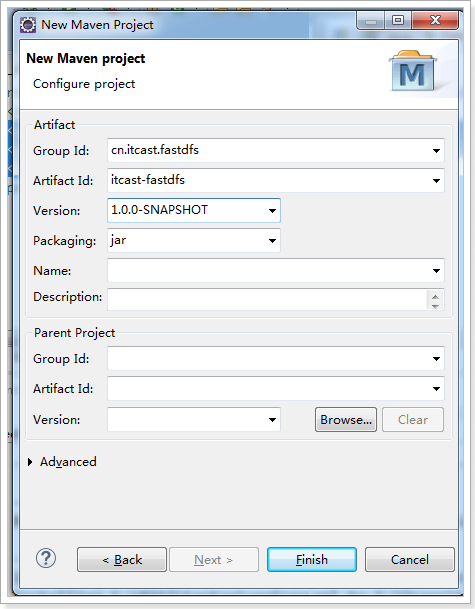
### 安装到本地仓库

把导入的工程安装到本地仓库，如下图：



## 图片上传测试

### 创建itcast-fastdfs工程进行测试



#### 加入依赖

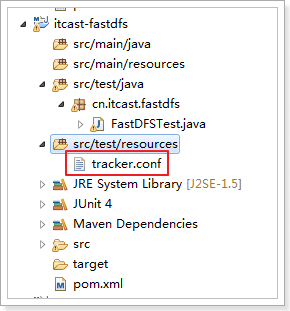
fastdfs\_client工程的坐标，就是我们需要的依赖坐标，加入效果如下：



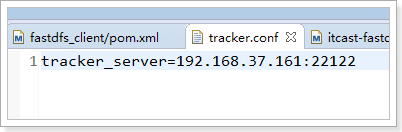
### java客户端上传步骤

1. 创建tracker.conf配置文件，内容就是tracker服务的地址。配置文件内容：tracker\_server=192.168.37.161:22122，然后加载配置文件(ClientGlobal.init方法加载)
2. 创建一个TrackerClient对象。直接new一个。
3. 使用TrackerClient对象创建连接，getConnection获得一个TrackerServer对象。
4. 创建一个StorageServer的引用，值为null，为接下来创建StorageClient使用
5. 创建一个StorageClient对象，直接new一个，需要两个参数TrackerServer对象、StorageServer的引用
6. 使用StorageClient对象upload\_file方法上传图片。
7. 返回数组。包含组名和图片的路径，打印结果。

#### 创建tracker.conf



内容为：



#### 编写测试代码

**public** **class** FastDFSTest {

@Test

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

// 1. 加载FastDFS的tracker的配置信息，其实就是 tracker\_server=192.168.37.161:22122

ClientGlobal.*init*(System.*getProperty*("user.dir") + "/src/main/resources/tracker.conf");

// 2. 创建TrackerClient，直接new

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

// 3. 使用TrackerClient，获取TrackerServer

TrackerServer trackerServer = trackerClient.getConnection();

// 4. 声明StorageServer，为null就可以了

StorageServer storageServer = **null**;

// 5. 创建StorageClient，需要两个参数TrackerServer，StorageServer

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

// 6. 使用StorageClient上传图片

String[] result = storageClient.upload\_file("C:/Users/tree/Desktop/新建文件夹/341928.jpg", "jpg", **null**);

// 7. 打印返回的结果，String数组

**for** (String s : result) {

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

}

}

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

System.***out***.println(System.*getProperty*("user.dir"));

}

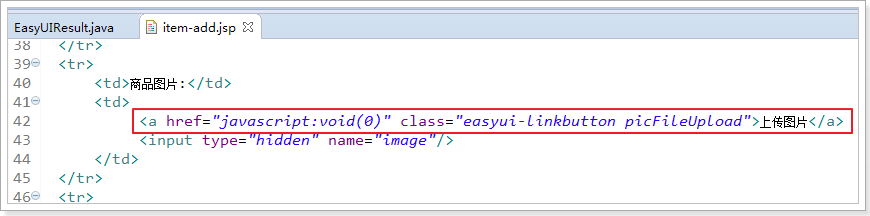
}

# 图片上传

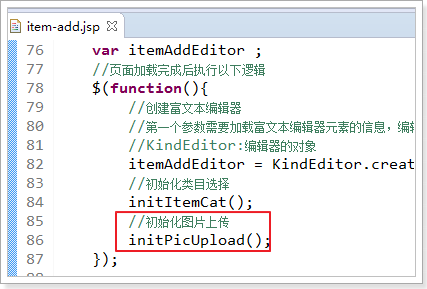
## 图片上传前端实现

### item-add.jsp：

上传按钮如下图：



页面加载完就会执行上传初始化逻辑：



初始化图片上传逻辑：

//图片上传初始化

**function** initPicUpload(){

//class选择器，其实获取到的就是上传图片按钮

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

//id选择器，其实获取到的就是form表单

**var** form = $('#itemAddForm');

//加载多图片上传组件（参考富文本编辑器的文档）

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

//editor:就是编辑器本身

**var** editor = **this**;

//执行插件的逻辑

editor.plugin.multiImageDialog({

//当点击“全部插入”按钮，执行以下逻辑

//urlList：多图片上传成功后，返回的图片url

clickFn : **function**(urlList) {

//获取class为pics的li的标签，删除，清空之前上传图片

$(".pics li").remove();

**var** imgArray = [];

//遍历返回的图片url

//i遍历的角标，data遍历的变量

KindEditor.each(urlList, **function**(i, data) {

//从遍历的数据中获取url，其实就是获取图片的url

//放到声明数组中

imgArray.push(data.url);

//获取class为pics的ul标签

//在后面追加li标签

$(".pics ul").append("<li><a href='"+data.url+"' target='\_blank'><img src='"+data.url+"' width='80' height='50' /></a></li>");

});

//获取name=image的元素，其实就是获取图片上传的input标签

//往input标签里赋值

//imgArray.join(",")：把数据转为字符串，数组中的元素用，分隔

form.find("[name=image]").val(imgArray.join(","));

//关闭上传界面

editor.hideDialog();

}

});

});

});

}

加载图片上传需要的参数

图片上传组件其实就是富文本编辑器的上传组件

下图是图片上传的请求参数，根据这个进行后台逻辑开发



## 上传实现分析

### 加入配置

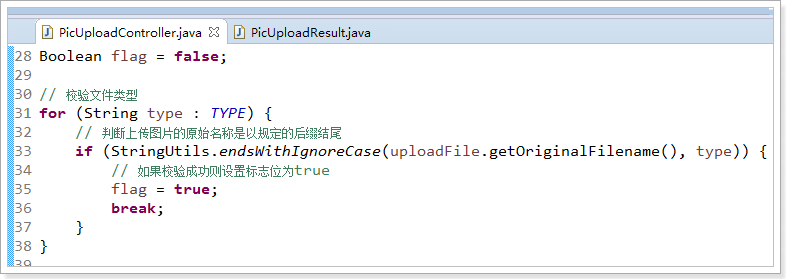
加入上传解析器的配置，如下图：



### 图片校验

1. 图片大小校验（上传解析器实现）
2. 校验文件类型，需要上传的是图片，实际上就是对文件的后缀校验

如下图：



1. 文件内容校验

可以通过图片唯一的属性来进行校验，例如高和宽



### 创建PicUploadResult

返回的数据需要有4个属性，需要json格式的数据

error：0上传成功，1代表上传失败

width:图片的宽

height:图片的高

url:图片的上传地址

**public** **class** PicUploadResult {

**private** Integer error;// 0上传成功，1代表上传失败

**private** String width;// 图片的宽

**private** String height;// 图片的高

**private** String url;// 图片的上传地址

get/set。。。。。。

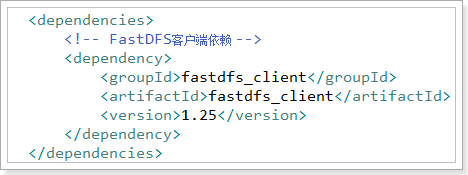
}

这个bean是共用的，直接放到taotao-common中

在taotao-manager-web添加taotao-common依赖，如下图：

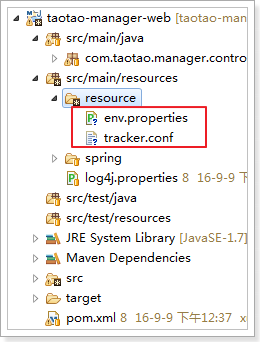


加入FastDFS的java客户端依赖

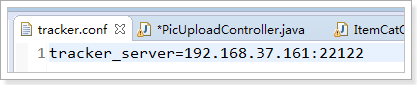


## 编写逻辑

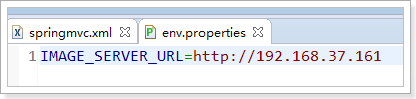
### 创建配置文件



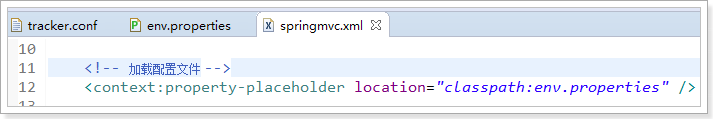
### 编写tracker.conf



### 编写env.properties



在taotao-manager-web的springmvc.xml配置加载env.properties



### 实现controller

在taotao-manager-web中编写上传controller

@Controller

@RequestMapping("pic/upload")

**public** **class** PicUploadController {

@Value("${IMAGE\_SERVER\_URL}")

**private** String IMAGE\_SERVER\_URL;

**private** **static** String[] *TYPE* = { ".jpg", ".jpeg", ".png", ".bmp", ".gif" };

// filePostName : "uploadFile", //上传文件名

// uploadJson : '/rest/pic/upload', //图片上传请求路径

// dir : "image" //上传文件类型

/\*\*

\* 图片上传

\*

\* **@param** uploadFile

\* **@return**

\* **@throws** Exception

\*/

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

@ResponseBody

**public** PicUploadResult upload(MultipartFile uploadFile) **throws** Exception {

// 声明标志位

**boolean** flag = **false**;

// 初始化返回数据,初始化上传失败

PicUploadResult picUploadResult = **new** PicUploadResult();

picUploadResult.setError(1);

// 校验后缀

**for** (String type : *TYPE*) {

String oname = uploadFile.getOriginalFilename();

// 如果后缀是要求的格式结尾，标志位设置为true，跳出寻汗

**if** (StringUtils.*endsWithIgnoreCase*(oname, type)) {

flag = **true**;

**break**;

}

}

// 如果校验失败，直接返回

**if** (!flag) {

**return** picUploadResult;

}

// 重置标志位

flag = **false**;

// 图片内容校验

**try** {

BufferedImage image = ImageIO.*read*(uploadFile.getInputStream());

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

picUploadResult.setHeight(String.*valueOf*(image.getHeight()));

picUploadResult.setWidth(String.*valueOf*(image.getWidth()));

flag = **true**;

}

} **catch** (Exception e) {

}

// 校验成功，需要上传图片

**if** (flag) {

// 1. 加载tracker配置文件

ClientGlobal.*init*(System.*getProperty*("user.dir") + "/src/main/resources/tracker.conf");

// 2. 创建TrackerClient

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

// 3. 获取TrackerServer

TrackerServer trackerServer = trackerClient.getConnection();

// 4. 声明StorageServer，为null

StorageServer storageServer = **null**;

// 5. 创建StorageClient

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

// abcabbaddd 1.2.3.4.jpg

// 6. 使用StorageClient上传图片

// 获取上传文件的后缀名

String ext = StringUtils.*substringAfterLast*(uploadFile.getOriginalFilename(), ".");

String[] str = storageClient.upload\_file(uploadFile.getBytes(), ext, **null**);

// 7. 进行返回的结果的拼接，上传图片的url

String picUrl = **this**.IMAGE\_SERVER\_URL + "/" + str[0] + "/" + str[1];

// 设置图片url

picUploadResult.setUrl(picUrl);

// 上传成功设置为0

picUploadResult.setError(0);

}

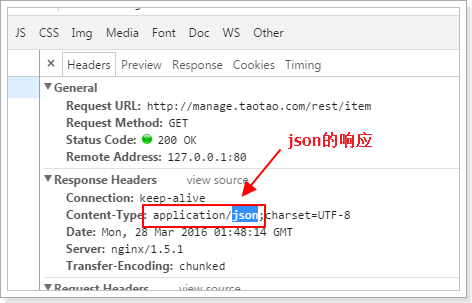
**return** picUploadResult;

}

## 解决火狐浏览器兼容性的问题

火狐浏览器上传不成功

因为富文本编辑器的上传组件对浏览器的兼容不好，不能使用json的响应



需要修改为响应文本的响应



修改底色标亮的部分

@Controller

@RequestMapping("pic/upload")

**public** **class** PicUploadController {

@Value("${IMAGE\_SERVER\_URL}")

**private** String IMAGE\_SERVER\_URL;

// 使用Jackson工具类把对象转换为json数据

**private** **static** **final** ObjectMapper ***MAPPER*** = **new** ObjectMapper();

**private** **static** String[] *TYPE* = { ".jpg", ".jpeg", ".png", ".bmp", ".gif" };

// filePostName : "uploadFile", //上传文件名

// uploadJson : '/rest/pic/upload', //图片上传请求路径

// dir : "image" //上传文件类型

@RequestMapping(method = RequestMethod.***POST***, produces = MediaType.***TEXT\_HTML\_VALUE***)

@ResponseBody

**public** String upload(MultipartFile uploadFile) **throws** Exception {

// 声明标志位

**boolean** flag = **false**;

// 初始化返回数据,初始化上传失败

PicUploadResult picUploadResult = **new** PicUploadResult();

picUploadResult.setError(1);

// 校验后缀

**for** (String type : *TYPE*) {

String oname = uploadFile.getOriginalFilename();

// 如果后缀是要求的格式结尾，标志位设置为true，跳出寻汗

**if** (StringUtils.*endsWithIgnoreCase*(oname, type)) {

flag = **true**;

**break**;

}

}

// 如果校验失败，直接返回

**if** (!flag) {

// 使用Jackson工具类把对象转为接送数据

String json = ***MAPPER***.writeValueAsString(picUploadResult);

**return** json;

}

// 重置标志位

flag = **false**;

// 图片内容校验

**try** {

BufferedImage image = ImageIO.*read*(uploadFile.getInputStream());

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

picUploadResult.setHeight(String.*valueOf*(image.getHeight()));

picUploadResult.setWidth(String.*valueOf*(image.getWidth()));

flag = **true**;

}

} **catch** (Exception e) {

}

// 校验成功，需要上传图片

**if** (flag) {

// 1. 加载tracker配置文件

ClientGlobal.*init*(System.*getProperty*("user.dir") + "/src/main/resources/tracker.conf");

// 2. 创建TrackerClient

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

// 3. 获取TrackerServer

TrackerServer trackerServer = trackerClient.getConnection();

// 4. 声明StorageServer，为null

StorageServer storageServer = **null**;

// 5. 创建StorageClient

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

// abcabbaddd 1.2.3.4.jpg

// 6. 使用StorageClient上传图片

// 获取上传文件的后缀名

String ext = StringUtils.*substringAfterLast*(uploadFile.getOriginalFilename(), ".");

String[] str = storageClient.upload\_file(uploadFile.getBytes(), ext, **null**);

// 7. 进行返回的结果的拼接，上传图片的url

String picUrl = **this**.IMAGE\_SERVER\_URL + str[0] + "/" + str[1];

// 设置图片url

picUploadResult.setUrl(picUrl);

// 上传成功设置为0

picUploadResult.setError(0);

}

// 使用Jackson工具类把对象转为接送数据

String json = ***MAPPER***.writeValueAsString(picUploadResult);

**return** json;

}

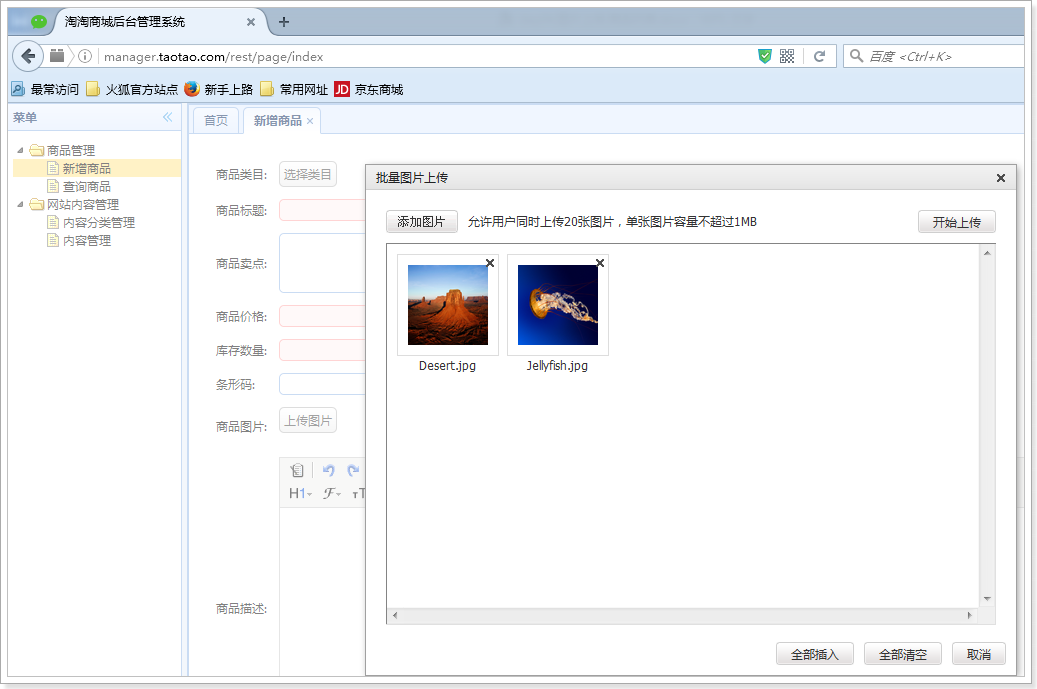
}

## 效果

谷歌浏览器上传成功



火狐浏览器上传成功



# 商品列表查询

## 前端分析

首页点击左侧“查询商品”进入商品列表显示界面



分析代码发现商品查询页面应该是item-list.jsp

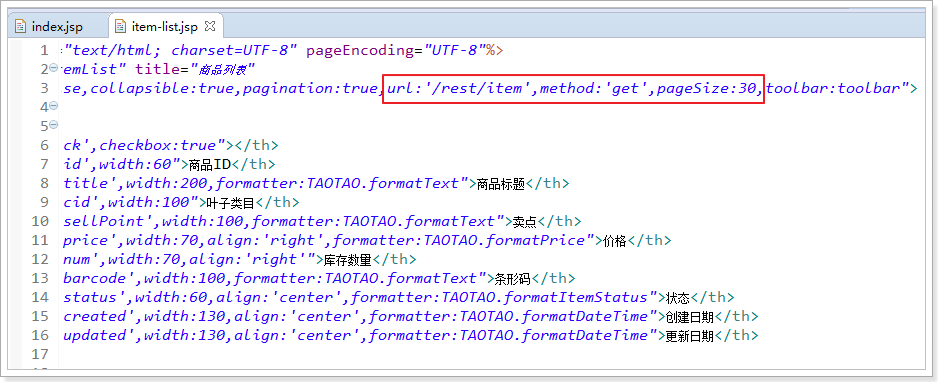


查看item-list.jsp代码，实现列表显示的是easyUI组件datagrid



创建datagrid，需要请求后台获取数据

请求的路径如下图



## 创建TaoResult

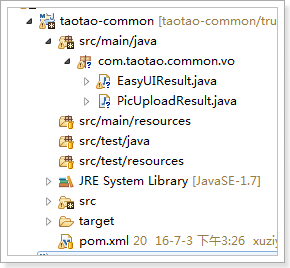
datagrid需要的数据有两个参数

total：查询到的数据总条数

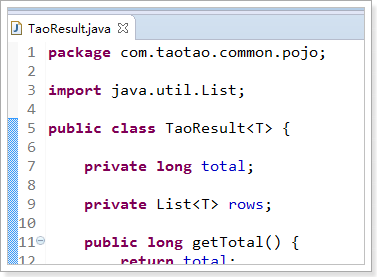
rows：查询的数据结果集

创建支持datagrid的EasyUIResult。

列表查询其他的工程会用到，上传的PicUploadResult也是一样，所以两个bean都放在taotao-common中。

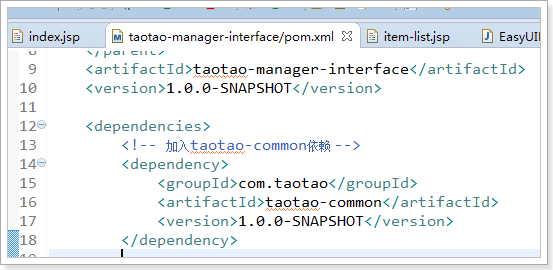


创建EasyUIResult



taotao-manager-interface和taotao-manager-web都需要添加taotao-common依赖

这样才可以使用TaoResult和PicUploadResult



## Controller实现

表现层实现controller

datagrid请求后台会携带分页数据page和rows

在ItemController里面增加分页查询方法

// url:'/rest/item',method:'get',pageSize:30,

/\*\*

\* 分页查询item

\* **@param** page

\* **@param** rows

\* **@return**

\*/

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

@ResponseBody

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

@RequestParam(value = "rows", defaultValue = "30") Integer rows) {

TaoResult<Item> easyUIResult = **this**.itemService.queryItemList(page, rows);

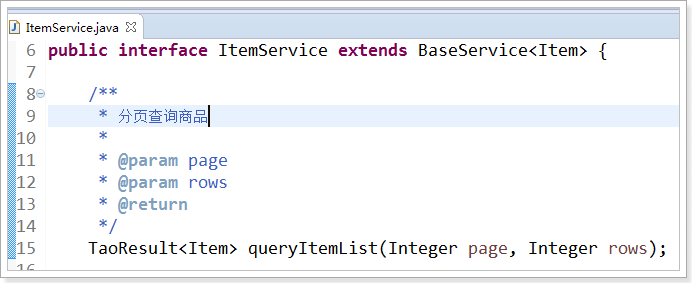
**return** easyUIResult;

}

## Service实现

### 编写接口方法

在taotao-manager-interface编写



### 实现方法

在taotao-manager-service实现

@Override

**public** TaoResult<Item> queryItemByPage(Integer page, Integer rows) {

// 设置分页数据

PageHelper.*startPage*(page, rows);

List<Item> list = **super**.queryListByWhere(**null**);

// 获取分页的详细数据

PageInfo<Item> pageInfo = **new** PageInfo<>(list);

// 封装返回对象

TaoResult<Item> taoResult = **new** TaoResult<>();

taoResult.setTotal(pageInfo.getTotal());

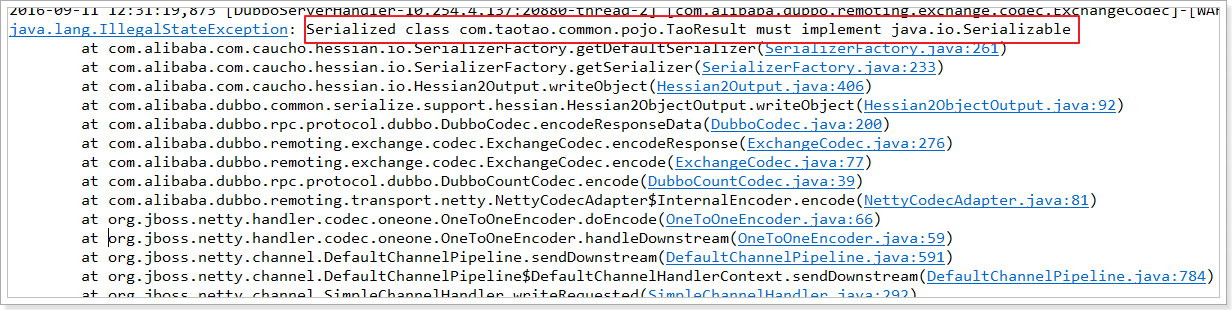
taoResult.setRows(list);

**return** taoResult;

}

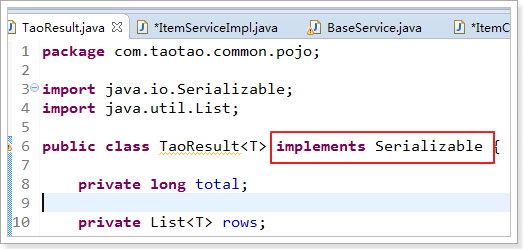
## 运行报错

访问发现出现错误提示：



使用dubbo传递的JavaBean，都需要实现序列化接口

需要给TaoResult实现序列化接口



## 效果

