# 今日大纲

1. 学习Nginx的使用
2. 实现商品的管理
   1. 新增商品
   2. 查询商品列表
   3. 编辑商品
   4. ~~删除商品~~
   5. ~~上架和下架商品~~

# 学习nginx

## 开发阶段中的环境

开发环境：自己的电脑

测试环境：提供给测试人员使用的环境

生成环境：项目最终发布上线的环境

预发布环境：数据是和生成环境的数据一致，运行最新的项目代码进去测试

每个环境的访问地址是不同的，可能因为访问地址不同导致一些问题的产生，所以，为了避免该类问题的产生，可以使不同的环境访问地址完全一致，通过域名访问即可实现。

## 需求：需要通过域名访问后台系统？

现在：<http://127.0.0.1/rest/page/index>

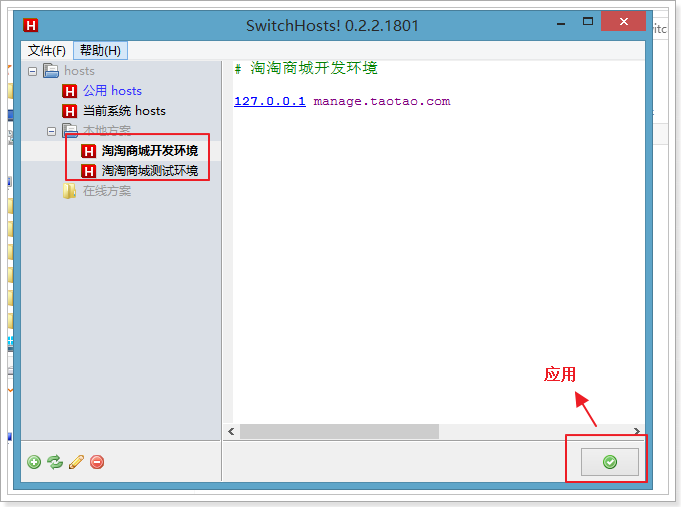
需要通过manage.taoao.com访问。

### Hosts

用户访问淘宝：

用户 🡺 [www.taobao.com](http://www.taobao.com) 🡺 本机的hosts文件中查找域名映射，如果查找到就返回 🡺浏览器通过域名到DNS查找服务器ip地址 🡺 执行访问

### 实现

1. 修改本地的hosts，将manage.taotao.com映射到127.0.0.1
   1. 

效果：



但是，如果后台系统占用80端口，其他系统就无法占用80，也就是无法通过域名直接访问，必须通过域名:端口访问<http://www.taotao.com:8081/>，

问题是：导致，生产环境和开发环境的访问路径是不同的。

如何解决以上问题？ -- 通过nginx来解决。

## 修改后台管理系统的端口为8081





## Nginx简介

目前电商和互联网系统都会采用nginx + 应用服务器(tomcat).

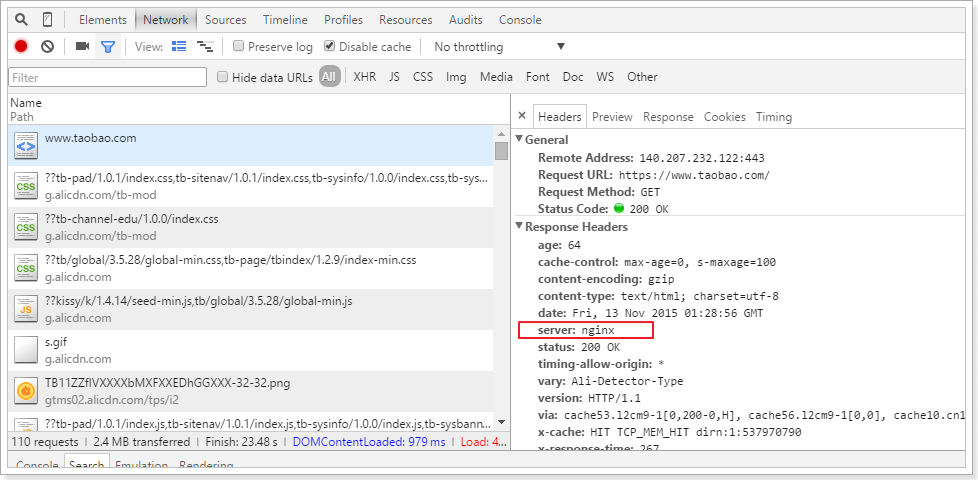
Web服务器分2类：

1. web服务器
   1. Apache 服务器
   2. Nginx
   3. IIS
2. web应用服务器
   1. tomcat
   2. resin
   3. jetty

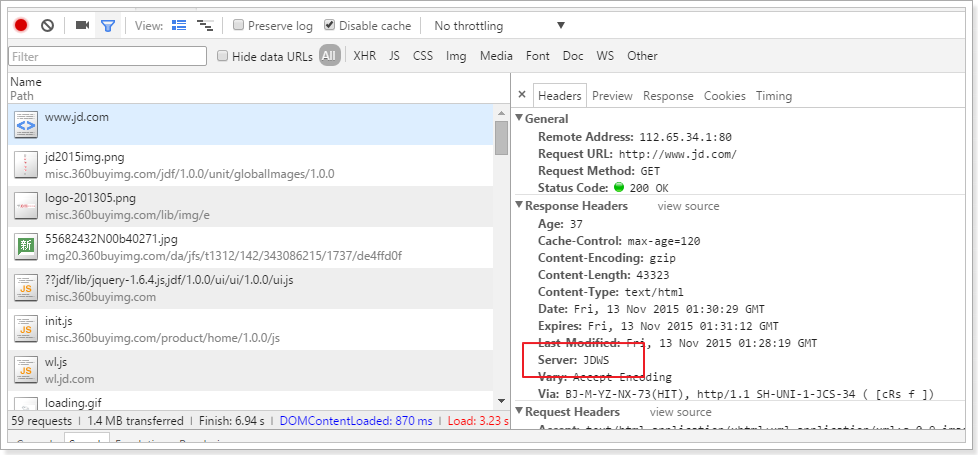
区分：web服务器不能解析jsp等页面，只能处理js、css、html等静态资源。

并发：web服务器的并发能力远高于web应用服务器。

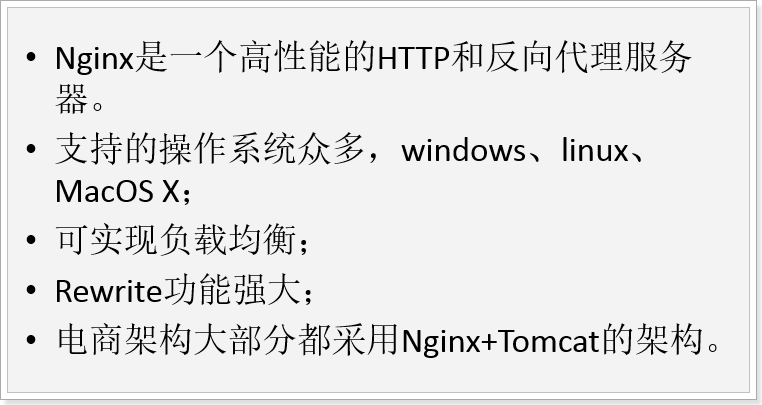
淘宝：



京东：（也是nginx，将服务名称改为JDWS）

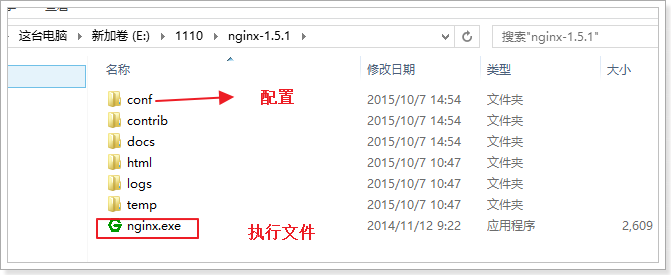


## 简介



## 使用

解压得到：



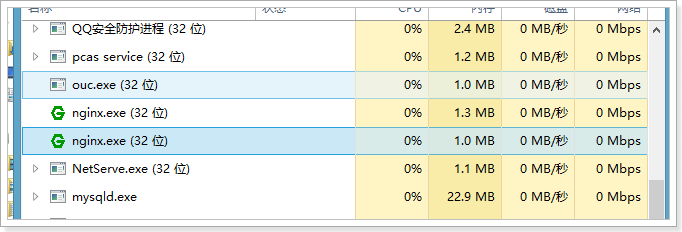
三个命令：(在 CMD 中执行)

启动：start nginx.exe

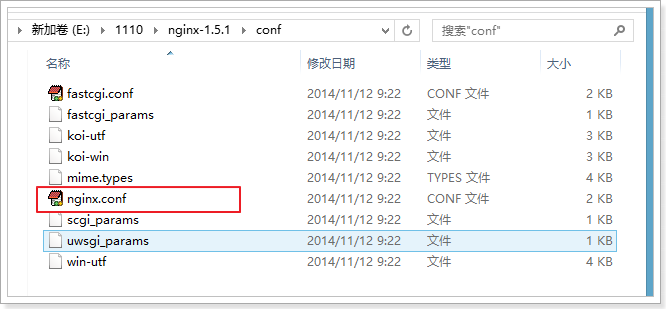
停止：nginx.exe -s stop

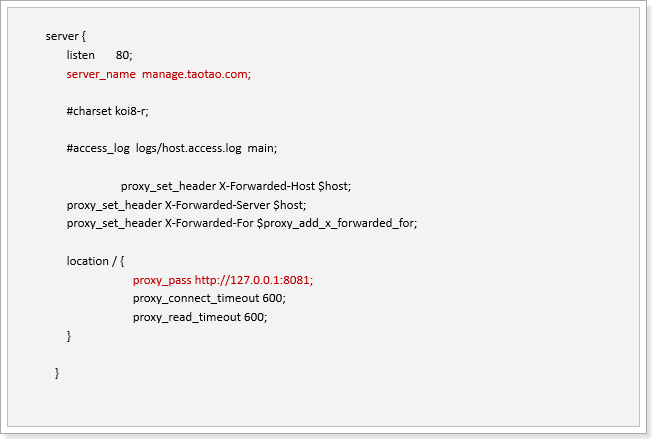
重新加载：nginx.exe -s reload

只有2个进程nginx才算是真正的启动成功：



## Nginx的配置





## 测试



现在的请求：

用户 🡺 Nginx 🡺 tomcat

## 启动nginx的注意事项

1. 其他服务占用80端口，会导致nginx启动失败，检查的方法是 cmd 🡺 nginx.exe
2. Nginx的路径不能包含中文

## 代理

### 正向代理

用户 🡺 代理 🡺 google

### 反向代理

用户 🡺 nginx 🡺 tomcat

解释：

Tomcat根本不知道请求的来源，只知道所有的请求都是来源于Nginx，不知道真正的来源。

# 封装BaseService

方法：

1. queryById
2. queryAll
3. queryOne
4. queryListByWhere
5. queryPageListByWhere
6. save
7. update
8. deleteById
9. deleteByIds
10. delete**ByWhere**

## 实现

**package** com.taotao.manage.service;

**import** java.util.List;

**import** com.github.abel533.entity.Example;

**import** com.github.abel533.mapper.Mapper;

**import** com.github.pagehelper.PageHelper;

**import** com.github.pagehelper.PageInfo;

**public** **abstract** **class** BaseService<T> {

**public** **abstract** Mapper<T> getMapper();

/\*\*

\* 根据id查询数据

\*

\* **@param** id

\* **@return**

\*/

**public** T queryById(Long id) {

**return** **this**.getMapper().selectByPrimaryKey(id);

}

/\*\*

\* 查询所有数据

\*

\* **@return**

\*/

**public** List<T> queryAll() {

**return** **this**.getMapper().select(**null**);

}

/\*\*

\* 根据条件查询一条数据，如果有多条数据会抛出异常

\*

\* **@param** record

\* **@return**

\*/

**public** T queryOne(T record) {

**return** **this**.getMapper().selectOne(record);

}

/\*\*

\* 根据条件查询数据列表

\*

\* **@param** record

\* **@return**

\*/

**public** List<T> queryListByWhere(T record) {

**return** **this**.getMapper().select(record);

}

/\*\*

\* 分页查询

\*

\* **@param** page

\* **@param** rows

\* **@param** record

\* **@return**

\*/

**public** PageInfo<T> queryPageListByWhere(Integer page, Integer rows, T record) {

// 设置分页条件

PageHelper.*startPage*(page, rows);

List<T> list = **this**.queryListByWhere(record);

**return** **new** PageInfo<T>(list);

}

/\*\*

\* 新增数据，返回成功的条数

\*

\* **@param** record

\* **@return**

\*/

**public** Integer save(T record) {

record.setCreated(new Date());

record.setUpdated(record.getCreated());

**return** **this**.getMapper().insert(record);

}

/\*\*

\* 新增数据，使用不为null的字段，返回成功的条数

\*

\* **@param** record

\* **@return**

\*/

**public** Integer saveSelective(T record) {

record.setCreated(new Date());

record.setUpdated(record.getCreated());

**return** **this**.getMapper().insertSelective(record);

}

/\*\*

\* 修改数据，返回成功的条数

\*

\* **@param** record

\* **@return**

\*/

**public** Integer update(T record) {

**return** **this**.getMapper().updateByPrimaryKey(record);

}

/\*\*

\* 修改数据，使用不为null的字段，返回成功的条数

\*

\* **@param** record

\* **@return**

\*/

**public** Integer updateSelective(T record) {

record.setUpdated(new Date());

**return** **this**.getMapper().updateByPrimaryKeySelective(record);

}

/\*\*

\* 根据id删除数据

\*

\* **@param** id

\* **@return**

\*/

**public** Integer deleteById(Long id) {

record.setUpdated(new Date());

**return** **this**.getMapper().deleteByPrimaryKey(id);

}

/\*\*

\* 批量删除

\* **@param** clazz

\* **@param** property

\* **@param** values

\* **@return**

\*/

**public** Integer deleteByIds(Class<T> clazz, String property, List<Object> values) {

Example example = **new** Example(clazz);

example.createCriteria().andIn(property, values);

**return** **this**.getMapper().deleteByExample(example);

}

/\*\*

\* 根据条件做删除

\*

\* **@param** record

\* **@return**

\*/

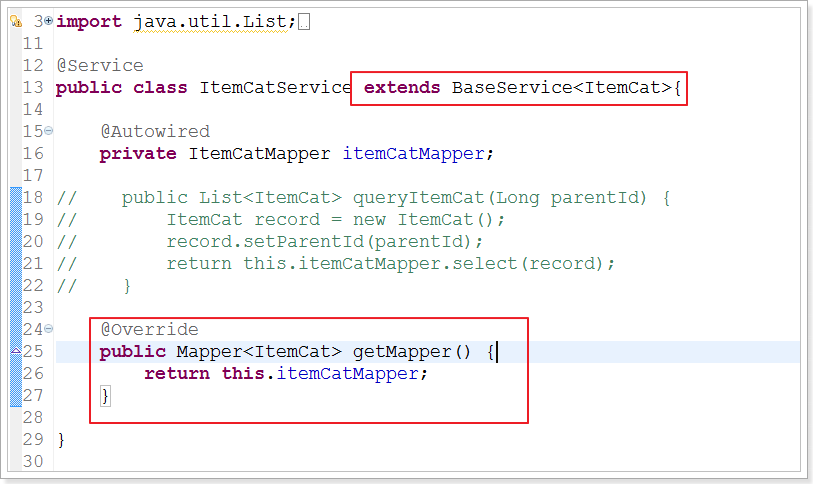
**public** Integer deleteByWhere(T record) {

**return** **this**.getMapper().delete(record);

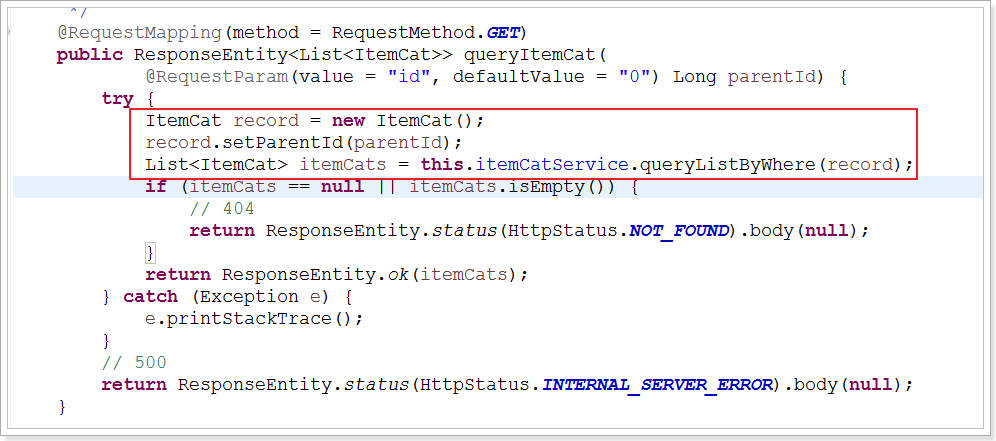
}

}

## 使用BaseService改造ItemCatService



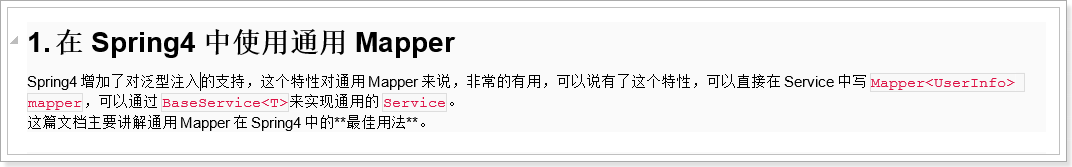
在Controller：



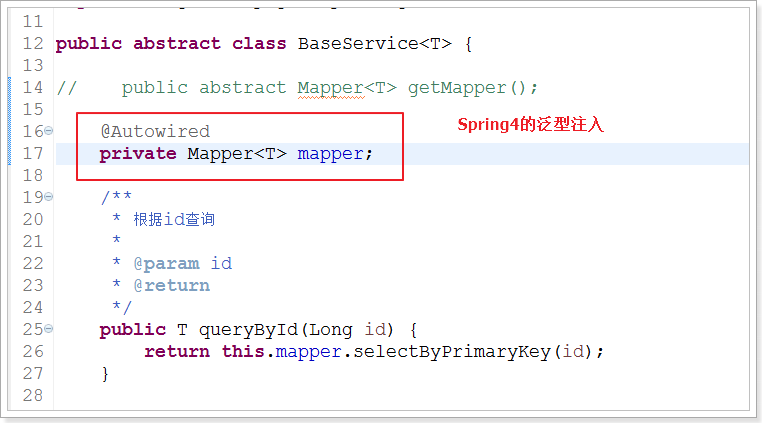
## 测试

测试结果：和原来的实现完全一致。

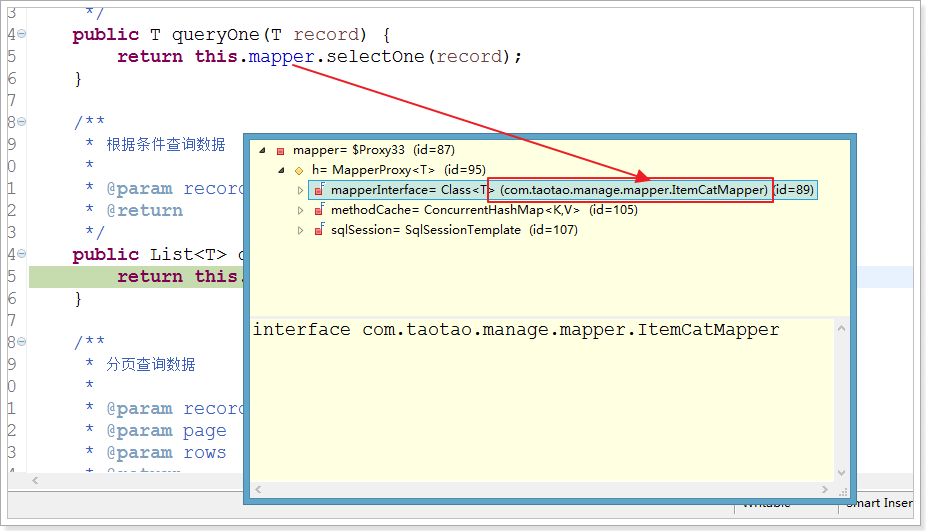
## Spring4的泛型注入



## BaseService的优化



测试：运行时注入具体的通用Mapper的子接口的实现类：



## 问题

ItemCatMapper在编码时没有使用到，是否将其删除？ -- **不能**。

原因：在Spring运行时会使用该对象，将其注入到BaseService中。

# 新增商品

## 商品数据结构

需求文档：

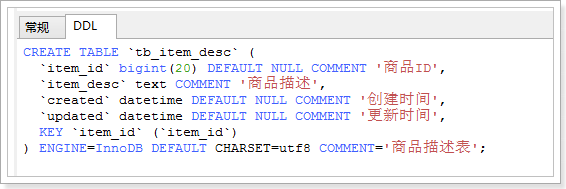


表结构：



问题1：为什么价格要以最小单位分存储？ -- 为了避免小数计算带来的问题。

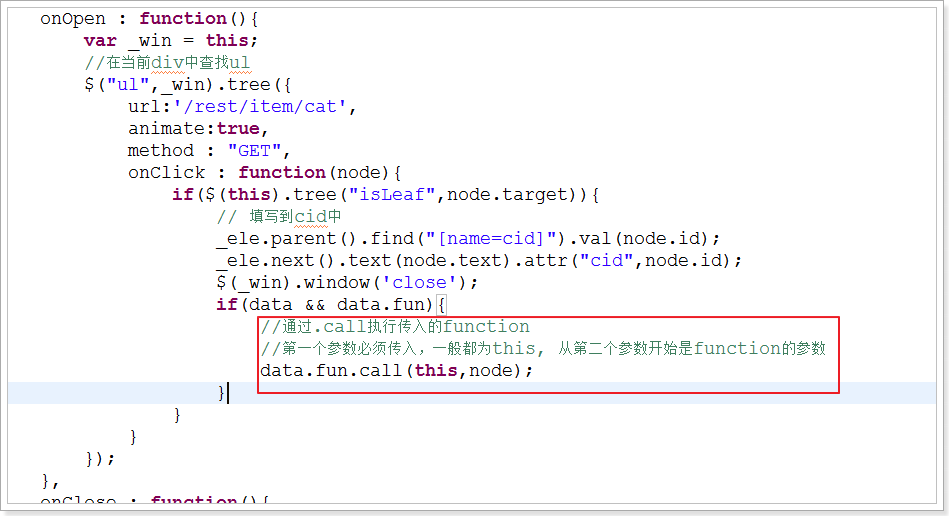
商品描述表：



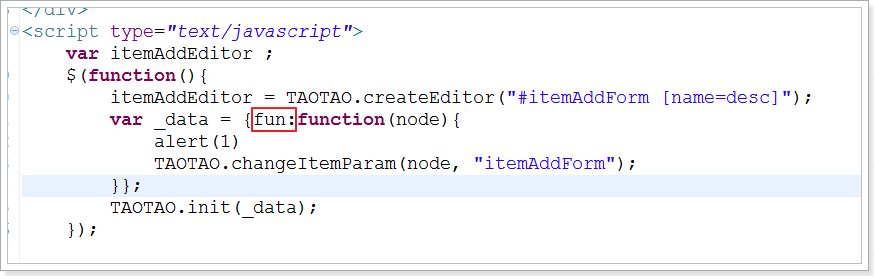
问题2：为什么要将商品描述和商品的基本数据分离？

1. 商品描述的数据量大，会导致数据文件变大，影响查询速度。
2. 后期会对商品描述数据的存储做改造，所以需要将描述数据分离

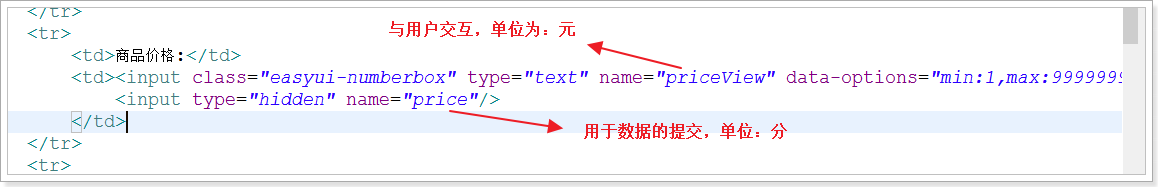
## 点击商品类目的叶子节点的事件



在item-add.jsp中定义：

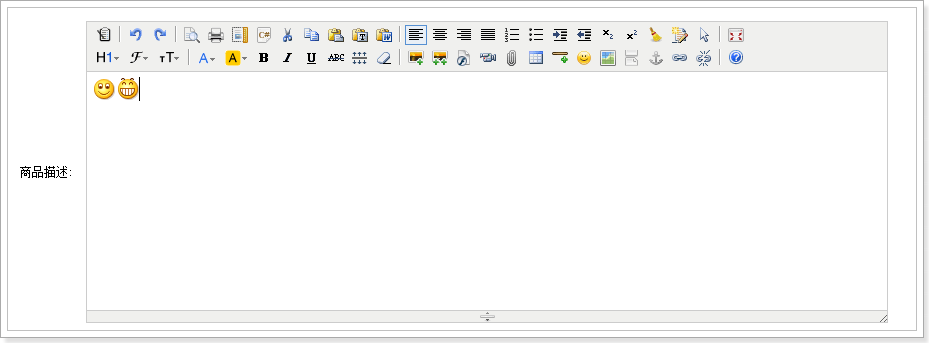


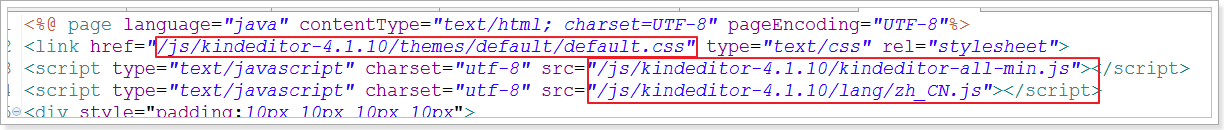
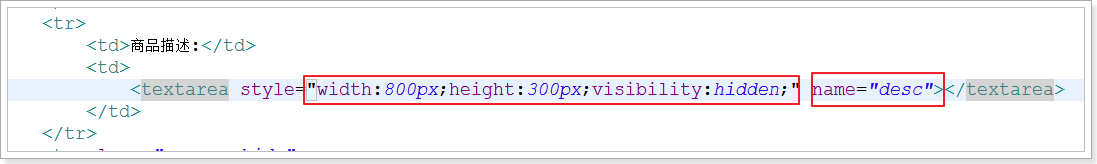
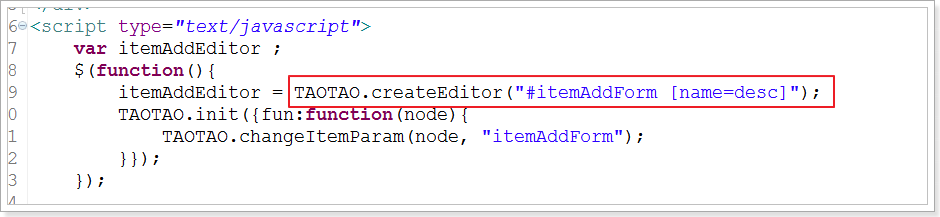
## 新增页面中价格

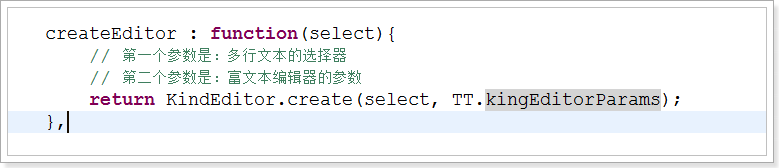


## KindEditor的使用

效果：



1. 导入js文件  
   
2. 定义多行文本  
   
3. 通过JS创建出富文本编辑器  
   

Common.js:  


## 提交事件

**function** submitForm(){

**if**(!$('#itemAddForm').form('validate')){

$.messager.alert('提示','表单还未填写完成!');

**return** ;

}

//处理商品的价格的单位，将元转化为分

$("#itemAddForm [name=price]").val(eval($("#itemAddForm [name=priceView]").val()) \* 100);

//将编辑器中的内容同步到隐藏多行文本中

itemAddEditor.sync();

//输入的规格参数数据保存为json

**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

});

});

paramJson = JSON.stringify(paramJson);

$("#itemAddForm [name=itemParams]").val(paramJson);

/\*

$.post("/rest/item/save",$("#itemAddForm").serialize(), function(data){

if(data.status == 200){

$.messager.alert('提示','新增商品成功!');

}

});

\*/

//提交到后台的RESTful

$.ajax({

type: "POST",

url: "/rest/item",

data: $("#itemAddForm").serialize(), //表单序列化，将所有的输入内容转化成K/V数据格式

statusCode : {

201 : **function**(){

$.messager.alert('提示','新增商品成功!');

},

400 : function(){

$.messager.alert('提示','提交的参数不合法!');

},

500 : **function**(){

$.messager.alert('提示','新增商品失败!');

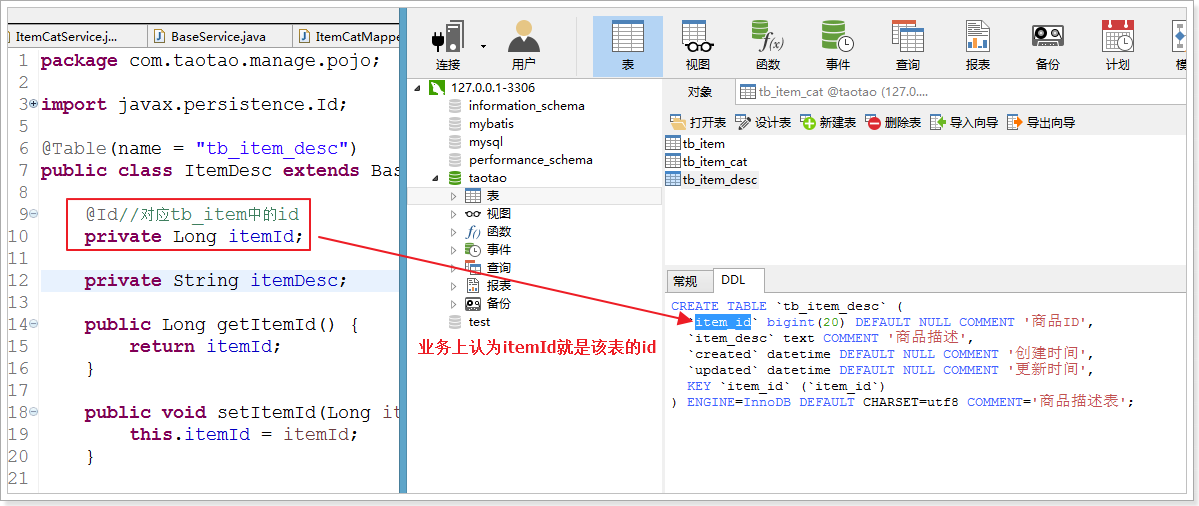
}

}

});

}

## 商品描述表中业务id



## 实现

### Controller

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

**public** ResponseEntity<Void> saveItem(Item item, @RequestParam("desc") String desc) {

**try** {

**if** (StringUtils.*isEmpty*(item.getTitle())) {

// 响应400

**return** ResponseEntity.*status*(HttpStatus.***BAD\_REQUEST***).build();

}

// 设置初始数据

item.setStatus(1);

item.setId(**null**);// 强制设置id为null

//保存商品的基本数据

**this**.itemService.save(item);

ItemDesc itemDesc = **new** ItemDesc();

itemDesc.setItemId(item.getId());

itemDesc.setItemDesc(desc);

//保存描述数据

**this**.itemDescService.save(itemDesc);

// 成功 201

**return** ResponseEntity.*status*(HttpStatus.***CREATED***).build();

} **catch** (Exception e) {

e.printStackTrace();

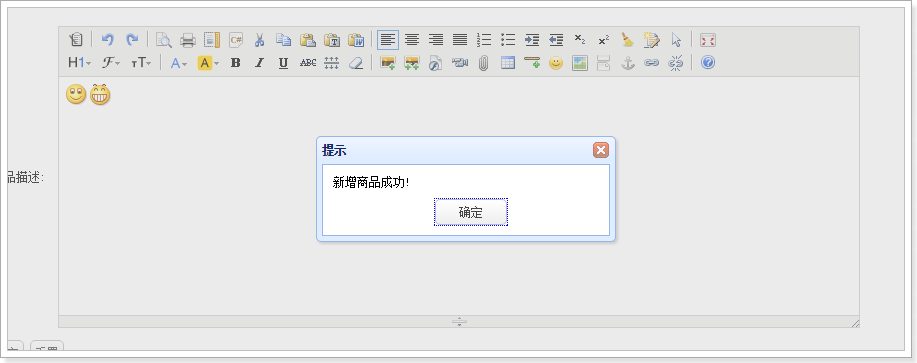
}

// 出错 500

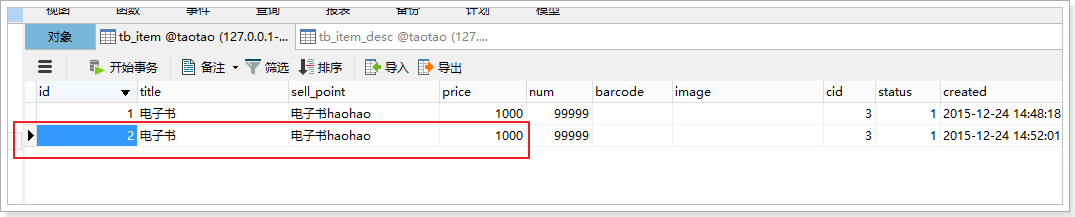
**return** ResponseEntity.*status*(HttpStatus.***INTERNAL\_SERVER\_ERROR***).build();

}

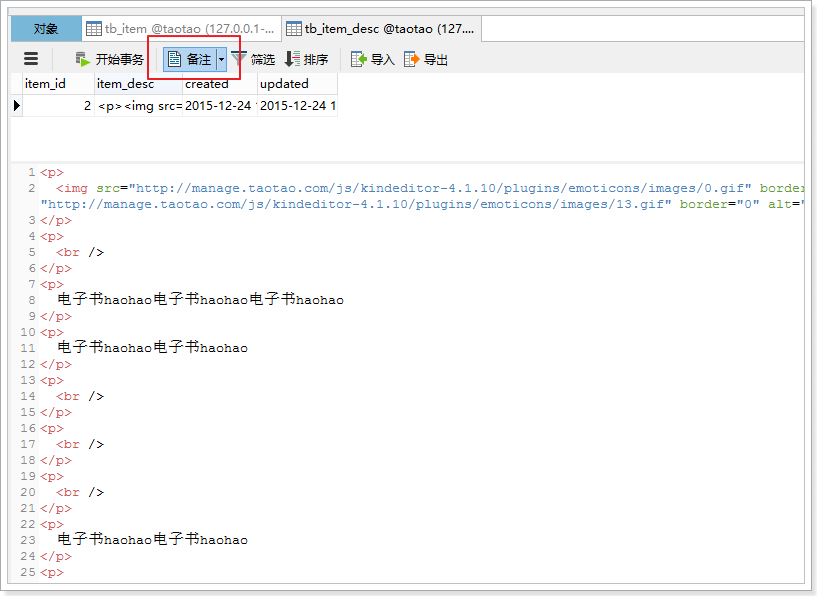
### 效果



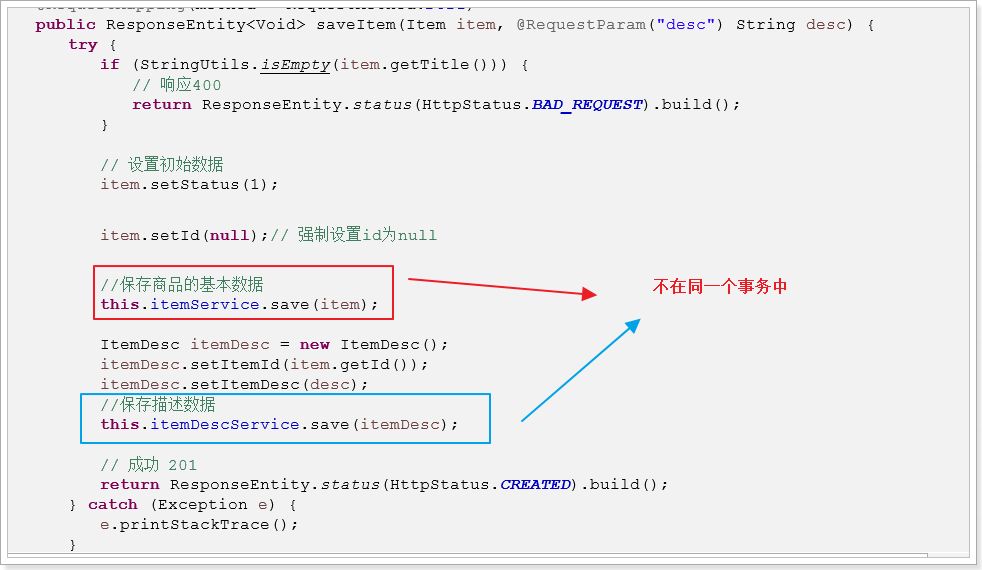
数据：



描述数据：



## 事务问题



解决：

将2次保存放到同一个Service的同一个方法中即可：



日志：

2016-01-17 15:44:12,404 [http-bio-8081-exec-1] [org.springframework.web.servlet.DispatcherServlet]-[DEBUG] DispatcherServlet with name 'taotao-manage' processing POST request for [/rest/item]

2016-01-17 15:44:12,407 [http-bio-8081-exec-1] [org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerMapping]-[DEBUG] Looking up handler method for path /item

2016-01-17 15:44:12,410 [http-bio-8081-exec-1] [org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerMapping]-[DEBUG] Returning handler method [public org.springframework.http.ResponseEntity<java.lang.Void> com.taotao.manage.controller.ItemController.saveItem(com.taotao.manage.pojo.Item,java.lang.String)]

2016-01-17 15:44:12,410 [http-bio-8081-exec-1] [org.springframework.beans.factory.support.DefaultListableBeanFactory]-[DEBUG] Returning cached instance of singleton bean 'itemController'

2016-01-17 15:44:12,460 [http-bio-8081-exec-1] [org.springframework.jdbc.datasource.DataSourceTransactionManager]-[DEBUG] Creating new transaction with name [com.taotao.manage.service.ItemService.saveItem]: PROPAGATION\_REQUIRED,ISOLATION\_DEFAULT

2016-01-17 15:44:12,472 [http-bio-8081-exec-1] [com.jolbox.bonecp.BoneCPDataSource]-[DEBUG] JDBC URL = jdbc:mysql://127.0.0.1:3306/taotao?useUnicode=true&characterEncoding=utf8&autoReconnect=true&allowMultiQueries=true, Username = root, partitions = 1, max (per partition) = 100, min (per partition) = 5, idle max age = 30 min, idle test period = 60 min, strategy = DEFAULT

2016-01-17 15:44:12,731 [http-bio-8081-exec-1] [org.springframework.jdbc.datasource.DataSourceTransactionManager]-[DEBUG] Acquired Connection [ConnectionHandle{url=jdbc:mysql://127.0.0.1:3306/taotao?useUnicode=true&characterEncoding=utf8&autoReconnect=true&allowMultiQueries=true, user=root, debugHandle=null, lastResetAgoInSec=0, lastUsedAgoInSec=0, creationTimeAgoInSec=0}] for JDBC transaction

2016-01-17 15:44:12,734 [http-bio-8081-exec-1] [org.springframework.jdbc.datasource.DataSourceTransactionManager]-[DEBUG] Switching JDBC Connection [ConnectionHandle{url=jdbc:mysql://127.0.0.1:3306/taotao?useUnicode=true&characterEncoding=utf8&autoReconnect=true&allowMultiQueries=true, user=root, debugHandle=null, lastResetAgoInSec=0, lastUsedAgoInSec=0, creationTimeAgoInSec=0}] to manual commit

2016-01-17 15:44:12,762 [http-bio-8081-exec-1] [org.mybatis.spring.SqlSessionUtils]-[DEBUG] Creating a new SqlSession

2016-01-17 15:44:12,782 [http-bio-8081-exec-1] [org.mybatis.spring.SqlSessionUtils]-[DEBUG] Registering transaction synchronization for SqlSession [org.apache.ibatis.session.defaults.DefaultSqlSession@6e42ea4a]

2016-01-17 15:44:12,885 [http-bio-8081-exec-1] [org.mybatis.spring.transaction.SpringManagedTransaction]-[DEBUG] JDBC Connection [ConnectionHandle{url=jdbc:mysql://127.0.0.1:3306/taotao?useUnicode=true&characterEncoding=utf8&autoReconnect=true&allowMultiQueries=true, user=root, debugHandle=null, lastResetAgoInSec=0, lastUsedAgoInSec=0, creationTimeAgoInSec=0}] will be managed by Spring

2016-01-17 15:44:12,889 [http-bio-8081-exec-1] [com.taotao.manage.mapper.ItemMapper.insert]-[DEBUG] ==> Preparing: INSERT INTO tb\_item (ID,NUM,SELL\_POINT,IMAGE,CREATED,PRICE,CID,UPDATED,STATUS,TITLE,BARCODE) VALUES ( ?,?,?,?,?,?,?,?,?,?,? )

2016-01-17 15:44:12,923 [http-bio-8081-exec-1] [com.taotao.manage.mapper.ItemMapper.insert]-[DEBUG] ==> Parameters: null, 99999(Integer), 电子书(String), (String), 2016-01-17 15:44:12.755(Timestamp), 1000(Long), 3(Long), 2016-01-17 15:44:12.755(Timestamp), 1(Integer), 电子书(String), wu(String)

2016-01-17 15:44:12,924 [http-bio-8081-exec-1] [com.taotao.manage.mapper.ItemMapper.insert]-[DEBUG] <== Updates: 1

2016-01-17 15:44:12,928 [http-bio-8081-exec-1] [com.taotao.manage.mapper.ItemMapper.insert!selectKey]-[DEBUG] ==> Executing: SELECT LAST\_INSERT\_ID()

2016-01-17 15:44:12,941 [http-bio-8081-exec-1] [com.taotao.manage.mapper.ItemMapper.insert!selectKey]-[DEBUG] <== Total: 1

2016-01-17 15:44:12,942 [http-bio-8081-exec-1] [org.mybatis.spring.SqlSessionUtils]-[DEBUG] Releasing transactional SqlSession [org.apache.ibatis.session.defaults.DefaultSqlSession@6e42ea4a]

2016-01-17 15:44:12,942 [http-bio-8081-exec-1] [org.springframework.jdbc.datasource.DataSourceTransactionManager]-[DEBUG] Participating in existing transaction

2016-01-17 15:44:12,956 [http-bio-8081-exec-1] [org.mybatis.spring.SqlSessionUtils]-[DEBUG] Fetched SqlSession [org.apache.ibatis.session.defaults.DefaultSqlSession@6e42ea4a] from current transaction

2016-01-17 15:44:12,957 [http-bio-8081-exec-1] [com.taotao.manage.mapper.ItemDescMapper.insert]-[DEBUG] ==> Preparing: INSERT INTO tb\_item\_desc (ITEM\_DESC,ITEM\_ID,CREATED,UPDATED) VALUES ( ?,?,?,? )

2016-01-17 15:44:12,958 [http-bio-8081-exec-1] [com.taotao.manage.mapper.ItemDescMapper.insert]-[DEBUG] ==> Parameters: <p>

<img src="http://manage.taotao.com/js/kindeditor-4.1.10/plugins/emoticons/images/42.gif" border="0" alt="" />

</p>

<p>

<br />

</p>

<p>

taotao 商城 哈哈

</p>(String), 41(Long), 2016-01-17 15:44:12.956(Timestamp), 2016-01-17 15:44:12.956(Timestamp)

2016-01-17 15:44:12,959 [http-bio-8081-exec-1] [com.taotao.manage.mapper.ItemDescMapper.insert]-[DEBUG] <== Updates: 1

2016-01-17 15:44:12,959 [http-bio-8081-exec-1] [org.mybatis.spring.SqlSessionUtils]-[DEBUG] Releasing transactional SqlSession [org.apache.ibatis.session.defaults.DefaultSqlSession@6e42ea4a]

2016-01-17 15:44:12,959 [http-bio-8081-exec-1] [org.mybatis.spring.SqlSessionUtils]-[DEBUG] Transaction synchronization committing SqlSession [org.apache.ibatis.session.defaults.DefaultSqlSession@6e42ea4a]

2016-01-17 15:44:12,959 [http-bio-8081-exec-1] [org.mybatis.spring.SqlSessionUtils]-[DEBUG] Transaction synchronization deregistering SqlSession [org.apache.ibatis.session.defaults.DefaultSqlSession@6e42ea4a]

2016-01-17 15:44:12,959 [http-bio-8081-exec-1] [org.mybatis.spring.SqlSessionUtils]-[DEBUG] Transaction synchronization closing SqlSession [org.apache.ibatis.session.defaults.DefaultSqlSession@6e42ea4a]

2016-01-17 15:44:12,960 [http-bio-8081-exec-1] [org.springframework.jdbc.datasource.DataSourceTransactionManager]-[DEBUG] Initiating transaction commit

2016-01-17 15:44:12,960 [http-bio-8081-exec-1] [org.springframework.jdbc.datasource.DataSourceTransactionManager]-[DEBUG] Committing JDBC transaction on Connection [ConnectionHandle{url=jdbc:mysql://127.0.0.1:3306/taotao?useUnicode=true&characterEncoding=utf8&autoReconnect=true&allowMultiQueries=true, user=root, debugHandle=null, lastResetAgoInSec=0, lastUsedAgoInSec=0, creationTimeAgoInSec=0}]

2016-01-17 15:44:12,961 [http-bio-8081-exec-1] [org.springframework.jdbc.datasource.DataSourceTransactionManager]-[DEBUG] Releasing JDBC Connection [ConnectionHandle{url=jdbc:mysql://127.0.0.1:3306/taotao?useUnicode=true&characterEncoding=utf8&autoReconnect=true&allowMultiQueries=true, user=root, debugHandle=null, lastResetAgoInSec=0, lastUsedAgoInSec=0, creationTimeAgoInSec=0}] after transaction

2016-01-17 15:44:12,961 [http-bio-8081-exec-1] [org.springframework.jdbc.datasource.DataSourceUtils]-[DEBUG] Returning JDBC Connection to DataSource

2016-01-17 15:44:12,980 [http-bio-8081-exec-1] [org.springframework.web.servlet.mvc.method.annotation.ResponseBodyAdviceChain]-[DEBUG] Invoking ResponseBodyAdvice chain for body=null

2016-01-17 15:44:12,981 [http-bio-8081-exec-1] [org.springframework.web.servlet.mvc.method.annotation.ResponseBodyAdviceChain]-[DEBUG] After ResponseBodyAdvice chain body=null

2016-01-17 15:44:12,981 [http-bio-8081-exec-1] [org.springframework.web.servlet.DispatcherServlet]-[DEBUG] Null ModelAndView returned to DispatcherServlet with name 'taotao-manage': assuming HandlerAdapter completed request handling

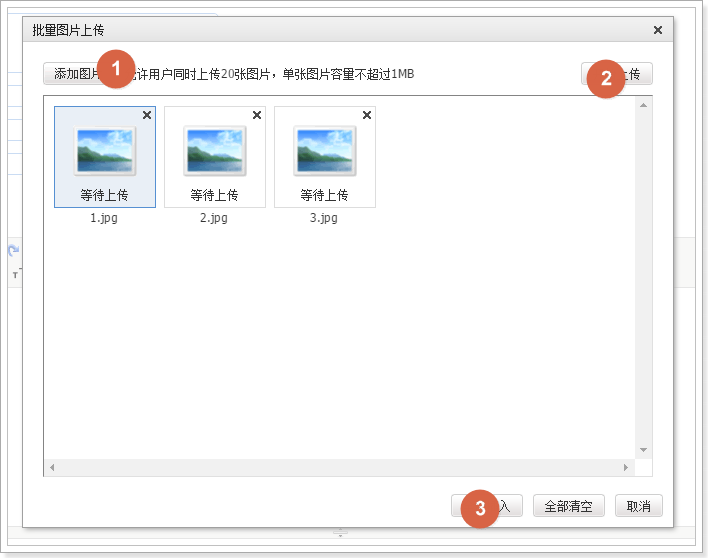
2016-01-17 15:44:12,981 [http-bio-8081-exec-1] [org.springframework.web.servlet.DispatcherServlet]-[DEBUG] Successfully completed request

2016-01-17 15:44:12,983 [http-bio-8081-exec-1] [org.springframework.beans.factory.support.DefaultListableBeanFactory]-[DEBUG] Returning cached instance of singleton bean 'org.mybatis.spring.SqlSessionFactoryBean#0'

# 图片上传

## 上传组件

使用KindEditor的上传组件：



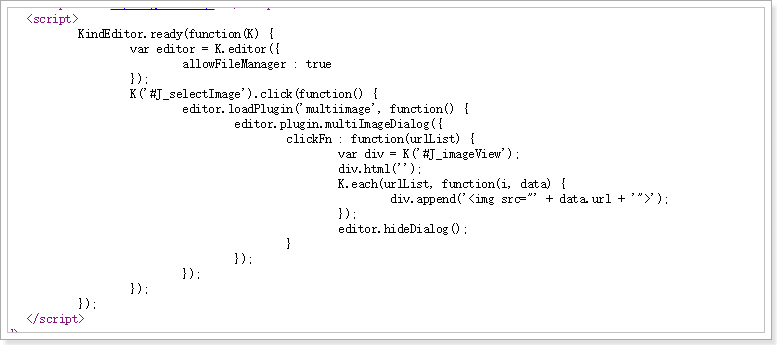
## 上传组件的JS实现

### 上传参数

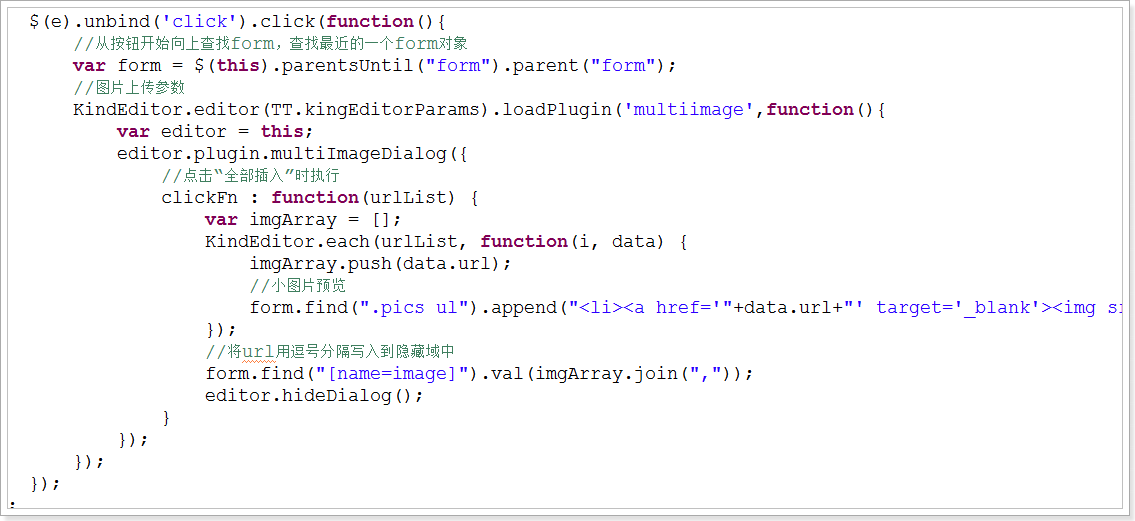


### 官方示例代码

workspace/taotao-manage/taotao-manage-web/src/main/webapp/js/kindeditor-4.1.10/examples/multi-image-dialog.html

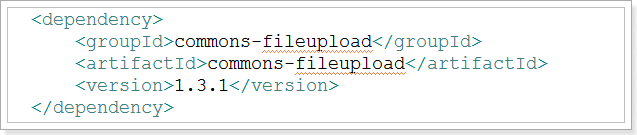
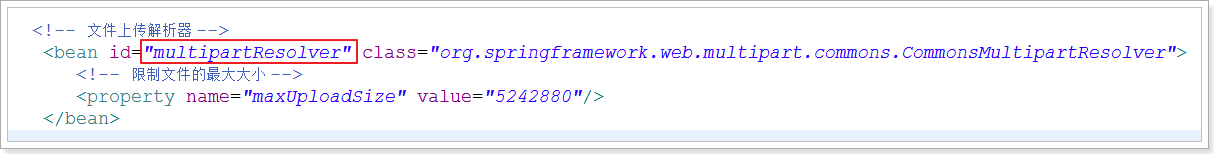


### 我们的实现



## 实现图片上传

在SpringMVC中实现图片上传：

1. 导入依赖 commons-fileupload  
   
2. 配置文件上传解析器  
   
3. 编写具体的上传逻辑（Controller）

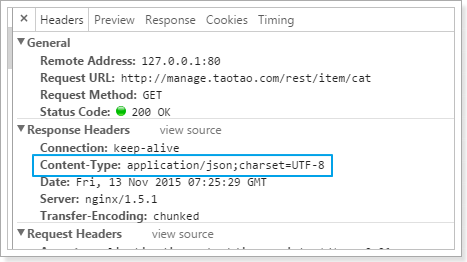
对文件做校验：

1. 类型，通过文件的后缀名
2. 大小
3. 上传文件的内容做校验
   1. 通过获取文件的宽和高来判断是否是图片

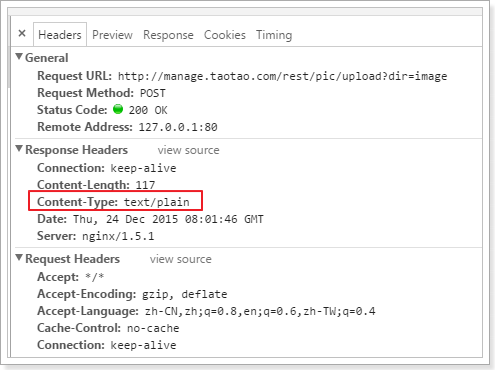
## 上传组件

需求：上传文件返回的数据是：文本类型的json数据。

响应类型：

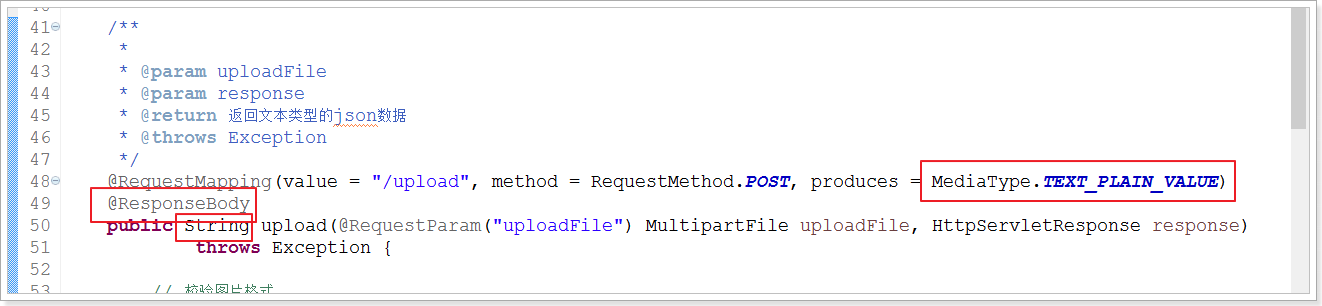


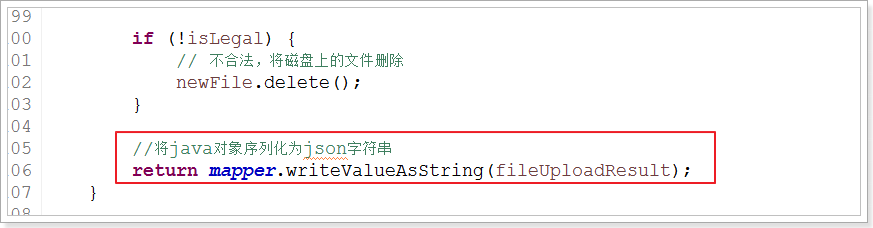
文本类型：



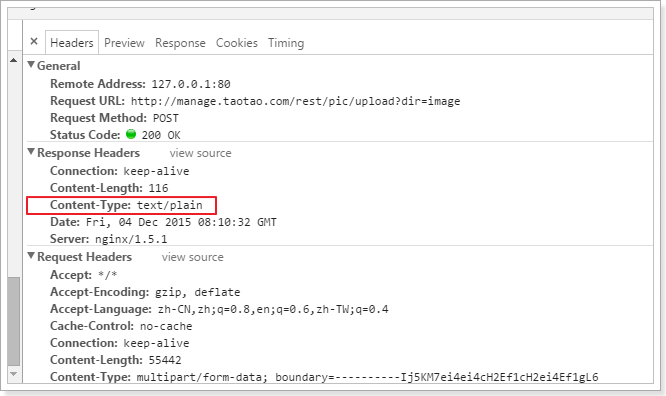
历史遗留问题。

解决：





测试：



## 完整的代码

**import** java.awt.image.BufferedImage;

**import** java.io.File;

**import** java.io.IOException;

**import** java.util.Date;

**import** javax.imageio.ImageIO;

**import** javax.servlet.http.HttpServletResponse;

**import** org.apache.commons.lang3.RandomUtils;

**import** org.apache.commons.lang3.StringUtils;

**import** org.joda.time.DateTime;

**import** org.slf4j.Logger;

**import** org.slf4j.LoggerFactory;

**import** org.springframework.http.MediaType;

**import** org.springframework.stereotype.Controller;

**import** org.springframework.web.bind.annotation.RequestMapping;

**import** org.springframework.web.bind.annotation.RequestMethod;

**import** org.springframework.web.bind.annotation.RequestParam;

**import** org.springframework.web.bind.annotation.ResponseBody;

**import** org.springframework.web.multipart.MultipartFile;

**import** com.fasterxml.jackson.databind.ObjectMapper;

**import** com.taotao.manage.bean.PicUploadResult;

/\*\*

\* 图片上传

\*/

@Controller

@RequestMapping("/pic")

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

**private** **static** **final** Logger ***LOGGER*** = LoggerFactory.*getLogger*(PicUploadController.**class**);

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

// 允许上传的格式

**private** **static** **final** String[] ***IMAGE\_TYPE*** = **new** String[] { ".bmp", ".jpg", ".jpeg", ".gif", ".png" };

/\*\*

\*

\* **@param** uploadFile

\* **@param** response

\* **@return** 返回文本类型的json数据

\* **@throws** Exception

\*/

@RequestMapping(value = "/upload", method = RequestMethod.***POST***, produces = MediaType.***TEXT\_PLAIN\_VALUE***)

@ResponseBody

**public** String upload(@RequestParam("uploadFile") MultipartFile uploadFile, HttpServletResponse response)

**throws** Exception {

// 校验图片格式

**boolean** isLegal = **false**;

**for** (String type : ***IMAGE\_TYPE***) {

**if** (StringUtils.*endsWithIgnoreCase*(uploadFile.getOriginalFilename(), type)) {

isLegal = **true**;

**break**;

}

}

// 封装Result对象，并且将文件的byte数组放置到result对象中

PicUploadResult fileUploadResult = **new** PicUploadResult();

// 状态

fileUploadResult.setError(isLegal ? 0 : 1);

// 文件新路径

String filePath = getFilePath(uploadFile.getOriginalFilename());

**if** (***LOGGER***.isDebugEnabled()) {

***LOGGER***.debug("Pic file upload .[{}] to [{}] .", uploadFile.getOriginalFilename(), filePath);

}

// 生成图片的绝对引用地址

String picUrl = StringUtils.*replace*(StringUtils.*substringAfter*(filePath, "E:\\0114\\taotao-upload"),

"\\", "/");

fileUploadResult.setUrl("http://image.taotao.com" + picUrl);

File newFile = **new** File(filePath);

// 写文件到磁盘

uploadFile.transferTo(newFile);

// 校验图片是否合法

isLegal = **false**;

**try** {

BufferedImage image = ImageIO.*read*(newFile);

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

fileUploadResult.setWidth(image.getWidth() + "");

fileUploadResult.setHeight(image.getHeight() + "");

isLegal = **true**;

}

} **catch** (IOException e) {

}

// 状态

fileUploadResult.setError(isLegal ? 0 : 1);

**if** (!isLegal) {

// 不合法，将磁盘上的文件删除

newFile.delete();

}

//将java对象序列化为json字符串

**return** ***mapper***.writeValueAsString(fileUploadResult);

}

//E:\\1110\\taotao-upload\\images\\2015\\11\\13\\20151113111111111.jpg

**private** String getFilePath(String sourceFileName) {

String baseFolder = "E:\\0114\\taotao-upload" + File.***separator*** + "images";

Date nowDate = **new** Date();

// yyyy/MM/dd

String fileFolder = baseFolder + File.***separator*** + **new** DateTime(nowDate).toString("yyyy")

+ File.***separator*** + **new** DateTime(nowDate).toString("MM") + File.***separator***

+ **new** DateTime(nowDate).toString("dd");

File file = **new** File(fileFolder);

**if** (!file.isDirectory()) {

// 如果目录不存在，则创建目录

file.mkdirs();

}

// 生成新的文件名

String fileName = **new** DateTime(nowDate).toString("yyyyMMddhhmmssSSSS")

+ RandomUtils.*nextInt*(100, 9999) + "." + StringUtils.*substringAfterLast*(sourceFileName, ".");

**return** fileFolder + File.***separator*** + fileName;

}

}

### 测试



## 解决显示图片的问题

之前的上传后，通过tomcat访问图片。

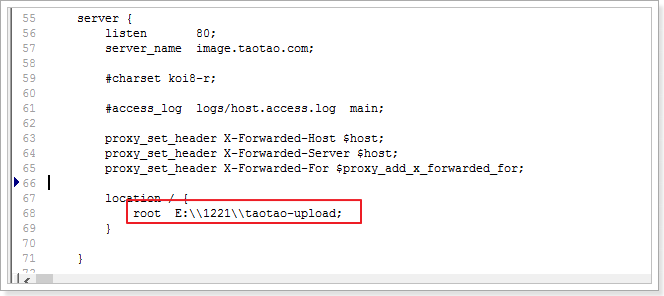
没有真实的项目是通过tomcat访问静态资源。 为什么？

Tomcat是应用服务器，作用：处理动态请求。

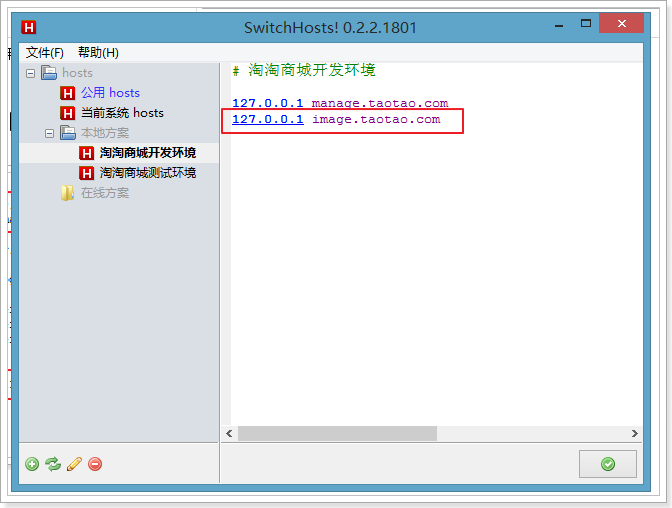
Nginx、Apache是web服务器，作用：处理静态资源、反向代理功能。

Web服务器的并发能力远高于应用服务器。

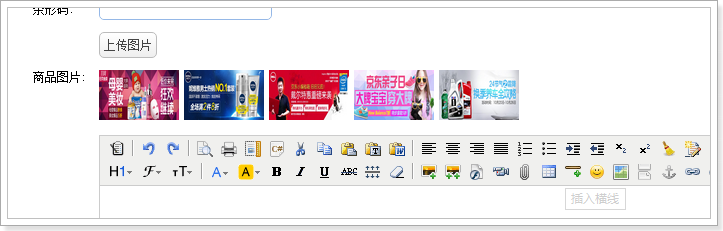
### 配置nginx访问图片



### 配置hosts



### 测试



## 图片存储路径硬编码到java代码中

解决方案：需要将路径配置到外部的配置文件中。

### @value

@Value作用：获取配置文件的值。

注入值：在Spring容器初始化（所有的bean）之后，在当前的所在容器中获取值，然后注入。

### Spring父子容器

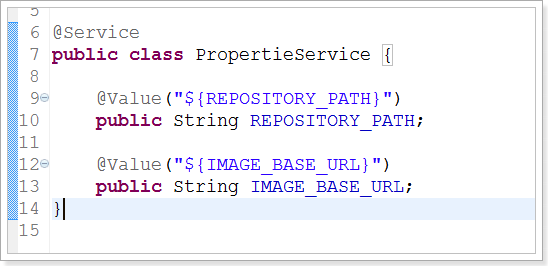
Spring容器 -- 父容器

SpringMVC容器 -- 子容器

父子容器的关系：

1. 子容器能够访问父容器的资源（bean）
   1. 示例：Controller可以注入Service
2. 父容器不能访问子容器的资源（bean）

## 实现

1. 定义ProperttiesService：  
   
2. 在Controller中注入Service：
3. 
4. 测试  
   

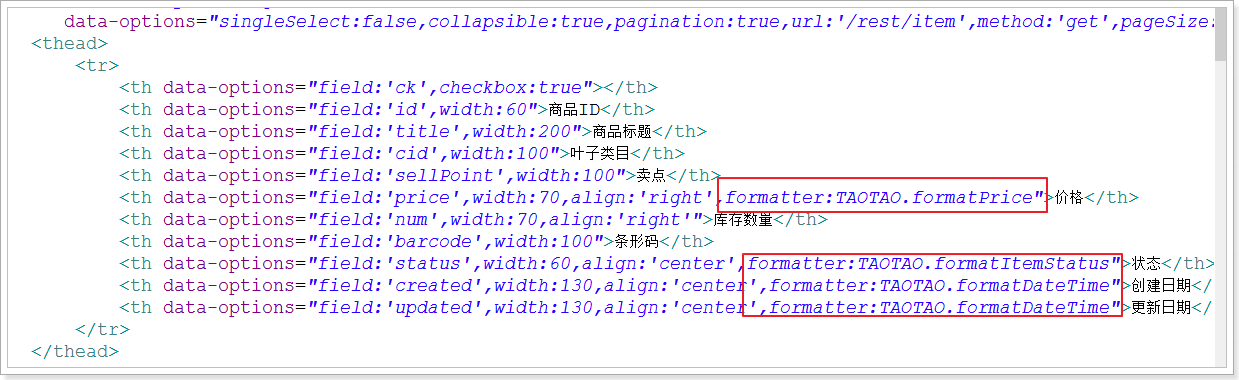
# 查询商品列表

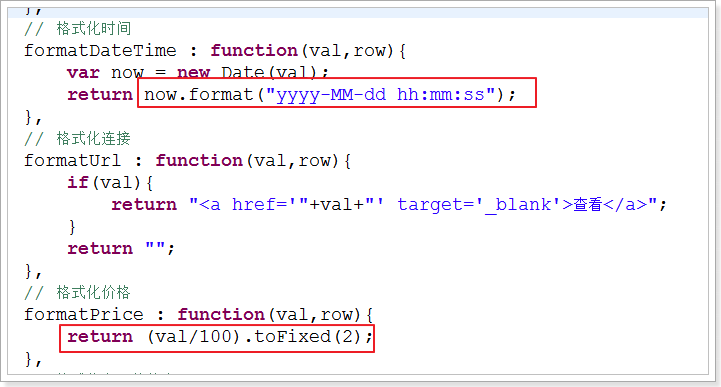
## JS



## EasyUI的datagrid的格式化输出

默认情况下，会直接显示返回的数据，但是有些情况下不能直接显示，如：价格、日期、性别，需要指定formatter函数。





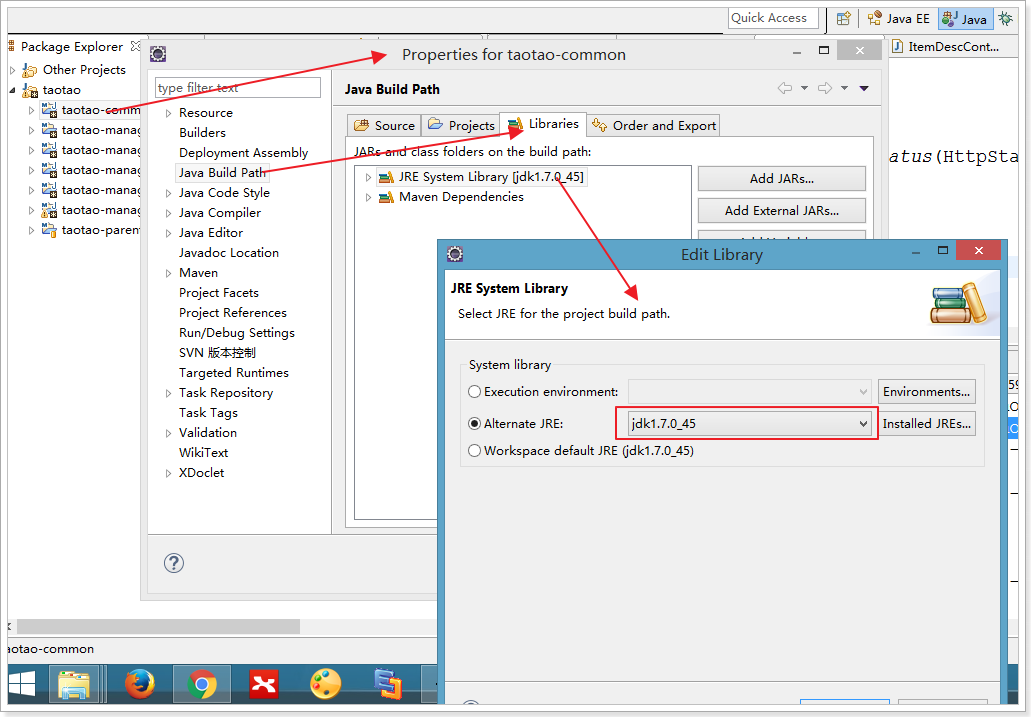


效果：



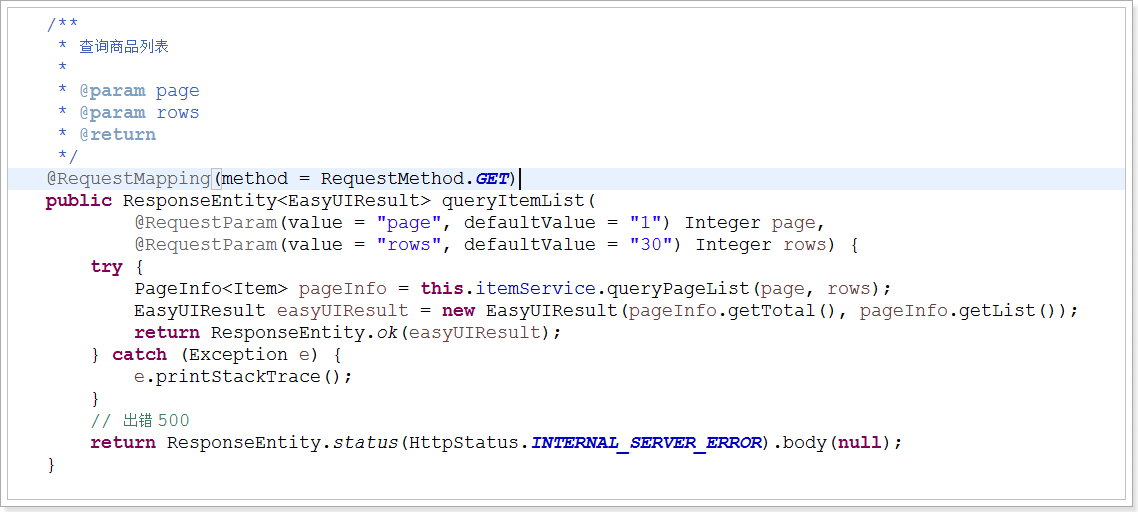
## 安装taotao-common

安装taotao-common出错，错误是：询问执行的是JRE还是JDK？

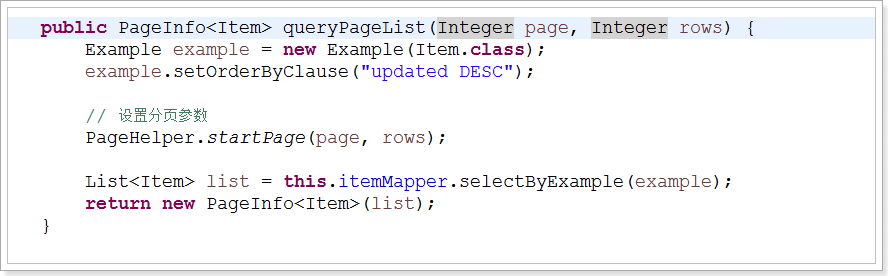


## 后台实现

Controller：



Service：



# 日志的书写



总结：

1. 方法的入参处需要将参数打印出
2. 业务执行的状态发生变化时，需要打印
3. 异常处需要打印