**畅购电商系统开发**

**第7天**

传智播客.黑马程序员.深圳

# 学习目标

* Thymeleaf [taɪm liːf]的介绍(模板引擎)
* Thymeleaf的入门
* Thymeleaf的语法及标签
* 商品详情页静态化工程搭建
* 商品详情页静态化功能实现

1.详情页静态化操作

2.填充基础数据 Spu、List<Sku>

3.规格切换

* 搜索页面渲染

1.数据展示

2.搜索条件展示

3.实现条件搜索控制

* 修改商品数据同步创建静态页

# Thymeleaf [taɪm liːf]介绍

​ 

thymeleaf是一个XML/XHTML/HTML5模板引擎，可用于Web与非Web环境中的应用开发。它是一个开源的Java库，基于Apache License 2.0许可，由Daniel Fernández创建，该作者还是Java加密库Jasypt的作者。

Thymeleaf提供了一个用于整合Spring MVC的可选模块，在应用开发中，你可以使用Thymeleaf来完全代替JSP或其他模板引擎，如Velocity、FreeMarker等。Thymeleaf的主要目标在于提供一种可被浏览器正确显示的、格式良好的模板创建方式，因此也可以用作静态建模。你可以使用它创建经过验证的XML与HTML模板。相对于编写逻辑或代码，开发者只需将标签属性添加到模板中即可。接下来，这些标签属性就会在DOM（文档对象模型）上执行预先制定好的逻辑。

它的特点便是：开箱即用，Thymeleaf允许您处理六种模板，每种模板称为模板模式：就是可以在这种格式运行，编译成底层的web代码，**其实所谓的语言就是通过不同的模式方法来将程序代码解析为底层代码，看那种用起来更方便简洁，用起来更简单**

* XML(可扩展标记语言)
* 有效的XML(格式正确有效的XML)
* XHTML(可扩展超文本标记语言)
* 有效的XHTML(格式正确有效的XHTML)
* HTML5
* 旧版HTML5(html过度到html5的相关版本)

Thymeleaf:Thymeleaf是一个模板引擎工具，主要用于页面渲染操作，可以取代之前的jsp操作。

所有这些模式都指的是格式良好的XML文件，但\*Legacy HTML5\*模式除外，它允许您处理HTML5文件，其中包含独立（非关闭）标记，没有值的标记属性或不在引号之间写入的标记属性。为了在这种特定模式下处理文件，Thymeleaf将首先执行转换，将您的文件转换为格式良好的XML文件，这些文件仍然是完全有效的HTML5（实际上是创建HTML5代码的推荐方法）1。

另请注意，验证仅适用于XML和XHTML模板。

然而，这些并不是Thymeleaf可以处理的唯一模板类型，并且用户始终能够通过指定在此模式下\*解析\*模板的方法和\*编写\*结果的方式来定义他/她自己的模式。这样，任何可以建模为DOM树（无论是否为XML）的东西都可以被Thymeleaf有效地作为模板处理。

官网：<https://www.thymeleaf.org/index.html>

# Springboot整合thymeleaf

使用springboot 来集成使用Thymeleaf可以大大减少单纯使用thymleaf的代码量，所以我们接下来使用springboot集成使用thymeleaf.

实现的步骤为：

* 创建一个sprinboot项目
* 添加thymeleaf的起步依赖
* 添加spring web的起步依赖
* 编写html 使用thymleaf的语法获取变量对应后台传递的值
* 编写controller 设置变量的值到model中

视图:

prefix:默认前缀 classpath:/templates/

suffix: 默认后缀 .html

## 创建工程

创建一个独立的工程springboot\_thymeleaf,该工程为案例工程，不需要放到changgou-parent工程中。

*<?***xml version="1.0" encoding="UTF-8"***?>*<**project xmlns="http://maven.apache.org/POM/4.0.0"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"**>  
 <**modelVersion**>4.0.0</**modelVersion**>  
  
 <**groupId**>com.itheima</**groupId**>  
 <**artifactId**>springboot\_thymeleaf</**artifactId**>  
 <**version**>1.0-SNAPSHOT</**version**>  
  
 <**parent**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-parent</**artifactId**>  
 <**version**>2.1.7.RELEASE</**version**>  
 </**parent**>  
  
 <**dependencies**>  
 *<!--web起步依赖-->* <**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-web</**artifactId**>  
 </**dependency**>  
  
 *<!--thymeleaf配置-->* <**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-thymeleaf</**artifactId**>  
 </**dependency**>  
 </**dependencies**>  
</**project**>

## 创建html

在resources中创建templates目录，在templates目录创建 demo1.html,代码如下：

<!DOCTYPE **html**>  
*<!--引入thymeleaf标签库-->*<**html lang="cn" xmlns:th="http://www.thymeleaf.org"**>  
<**head**>  
 <**meta charset="UTF-8"**>  
 <**title**>thymeleaf入门案例</**title**>  
</**head**>  
<**body**>  
*<!--th:text 使用thymeleaf获取文本数据，类似于EL表达式-->*您好，<**span th:text="${name}"**></**span**>，欢迎学习Thymeleaf!  
</**body**>  
</**html**>

## application.yml配置

创建application.yml,并设置thymeleaf的缓存设置，设置为false。默认加缓存的，用于测试。

**spring**:  
 **thymeleaf**:  
 *#关闭缓存，便于测试* **cache**: **false**

在这里，其实还有一些默认配置，比如视图前缀：classpath:/templates/,视图后缀：.html

org.springframework.boot.autoconfigure.thymeleaf.ThymeleafProperties部分源码如下：



## 控制层

创建controller用于测试后台 设置数据到model中。

创建com.itheima.controller.TestController，代码如下

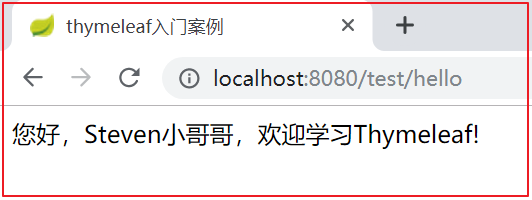
@Controller  
@RequestMapping(**"test"**)  
**public class** TestController {  
 */\*\*  
 \* 访问/test/hello 跳转到 demo1.html  
 \** ***@return*** *\*/* @RequestMapping(**"hello"**)  
 **public** String hello(Model model){  
 System.***out***.println(**"hello thymeleaf..."**);  
 model.addAttribute(**"name"**, **"Steven小哥哥"**);  
 *//直接返回视图名字就可，因为SpringBoot已经默认配置好前缀后缀* **return "demo1"**;  
 }  
}

## 测试

创建启动类com.itheima.ThymeleafApplication，代码如下：

@SpringBootApplication  
**public class** ThymeleafApplication {  
  
 **public static void** main(String[] args) {  
 SpringApplication.*run*(ThymeleafApplication.**class**,args);  
 }  
}

启动系统，并在浏览器访问



# Thymeleaf基本语法

## th:action

定义表单提交路径，类似<form>标签的action属性。

例如：

<**form id="login-form" th:action="@{/test/hello}"**>  
 <**button**>提交</**button**>  
</**form**>

## th:each

对象遍历，功能类似jstl中的<c:forEach>标签。

创建com.itheima.pojo.User,代码如下：

**public class** User {  
 **private** Integer **id**;  
 **private** String **name**;  
 **private** String **address**;  
 *//..get..set*}

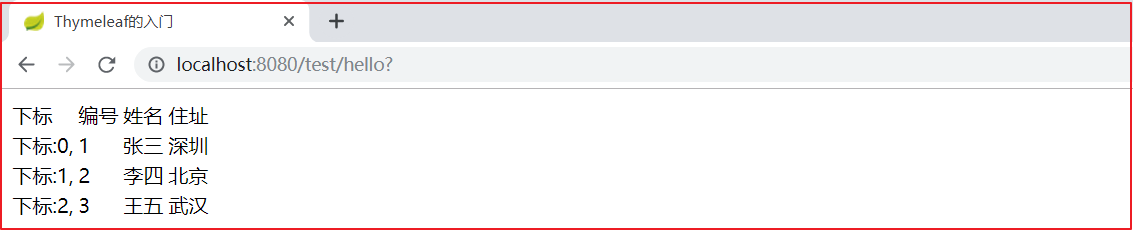
Controller添加数据

*/\*\*  
 \* 访问/test/hello 跳转到 demo1.html  
 \** ***@return*** *\*/*@RequestMapping(**"hello"**)  
**public** String hello(Model model){  
 System.***out***.println(**"hello thymeleaf..."**);  
 model.addAttribute(**"name"**, **"Steven小哥哥"**);  
  
 *//集合数据* List<User> users = **new** ArrayList<User>();  
 users.add(**new** User(1,**"张三"**,**"深圳"**));  
 users.add(**new** User(2,**"李四"**,**"北京"**));  
 users.add(**new** User(3,**"王五"**,**"武汉"**));  
 model.addAttribute(**"users"**,users);  
 *//直接返回视图名字就可，因为SpringBoot已经默认配置好前缀后缀* **return "demo1"**;  
}

页面输出

<**table**>  
 <**tr**>  
 <**td**>下标</**td**>  
 <**td**>编号</**td**>  
 <**td**>姓名</**td**>  
 <**td**>住址</**td**>  
 </**tr**>  
  
 <**tr th:each="user,userStat:${users}"**>  
 <**td th:text="${userStat.index}"**></**td**>  
 <**td th:text="${user.id}"**></**td**>  
 <**td th:text="${user.name}"**></**td**>  
 <**td th:text="${user.address}"**></**td**>  
 </**tr**>  
</**table**>

测试效果



## Map输出

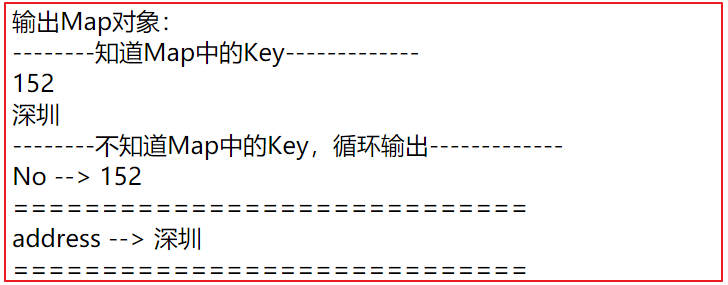
后台添加Map

*//Map定义*Map<String,Object> dataMap = **new** HashMap<String,Object>();  
dataMap.put(**"No"**,**"152"**);  
dataMap.put(**"address"**,**"深圳"**);  
model.addAttribute(**"dataMap"**,dataMap);

页面输出

输出Map对象：<**br**>  
--------知道Map中的Key-------------<**br**>  
<**span th:text="${dataMap.No}"**></**span**><**br**>  
<**span th:text="${dataMap.address}"**></**span**><**br**>  
--------不知道Map中的Key，循环输出-------------<**br**>  
<**div th:each="map:${dataMap}"**>  
 <**span th:text="${map.key}"**></**span**> -->  
 <**span th:text="${map.value}"**></**span**><**br**>  
 =============================  
</**div**>

输出结果



## 数组输出

后台添加数组

*//存储一个数组*String[] names = {**"张三"**,**"李四"**,**"王五"**};  
model.addAttribute(**"names"**,names);

页面输出

输出数组:  
<**div th:each="name,stat:${names}"**>  
 <**span th:text="${stat.count}"**></**span**> --> <**span th:text="${name}"**></**span**><**br**>  
</**div**>

输出结果



## Date输出

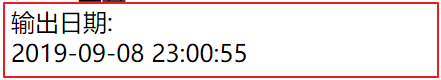
后台添加日期

*//日期*model.addAttribute(**"now"**,**new** Date());

页面输出

输出日期:  
<**div**>  
 <**span th:text="${#dates.format(now,'yyyy-MM-dd HH:mm:ss')}"**></**span**>  
</**div**>

输出结果



## th:if条件

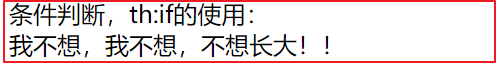
后台添加年龄

*//年龄*model.addAttribute(**"age"**,21);

页面输出

条件判断，th:if的使用：  
<**div th:if="${age} > 18"**>我不想，我不想，不想长大！！</**div**>

输出结果



## th:fragment 定义一个模块

可以定义一个独立的模块，创建一个footer.html代码如下：

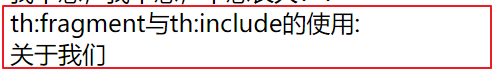
<**div id="C" th:fragment="copy"** >  
 关于我们<**br**/>  
</**div**>

## th:include

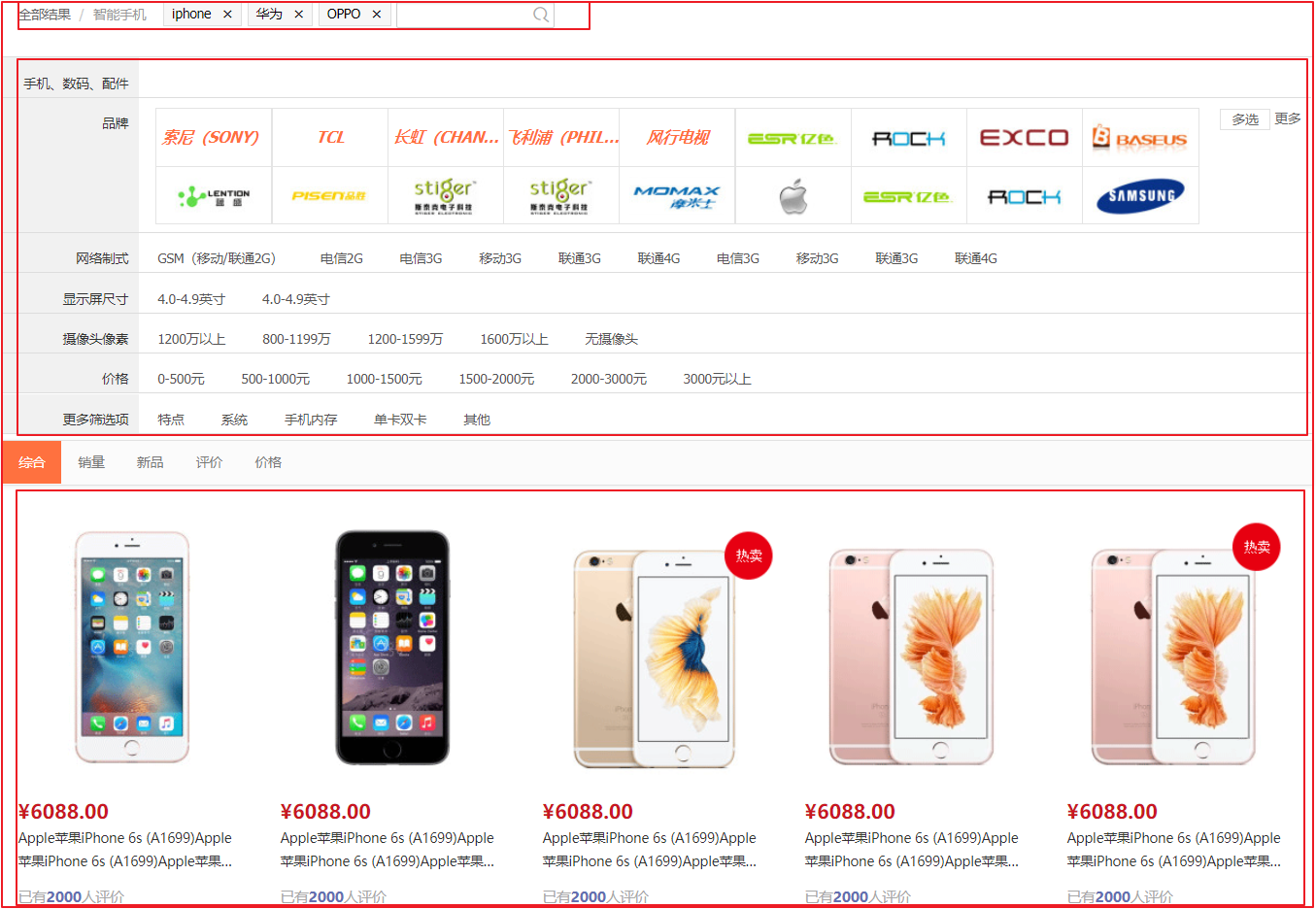
可以直接引入th:fragment,在demo1.html中引入如下代码：

th:fragment与th:include的使用:  
*<!--选择器，footer文件名::copy碎片名-->*<**div th:include="footer::copy"**></**div**>

输出结果



# 搜索页面渲染



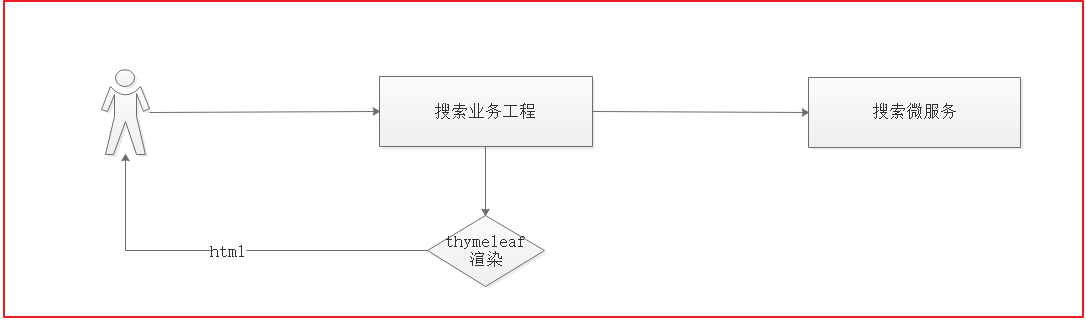
搜索页面要显示的内容主要分为3块。

1)搜索的数据结果

2)筛选出的数据搜索条件

3)用户已经勾选的数据条件

## 搜索实现



搜索的业务流程如上图，用户每次搜索的时候，先经过搜索业务工程，搜索业务工程调用搜索微服务工程，这里搜索业务工程单独挪出来的原因是它这里涉及到了模板渲染以及其他综合业务处理，以后很有可能会有移动端的搜索和PC端的搜索，后端渲染如果直接在搜索微服务中进行，会对微服务造成一定的侵入，不推荐这么做，推荐微服务独立，只提供服务，如果有其他页面渲染操作，可以搭建一个独立的消费工程调用微服务达到目的。

### 搜索工程搭建

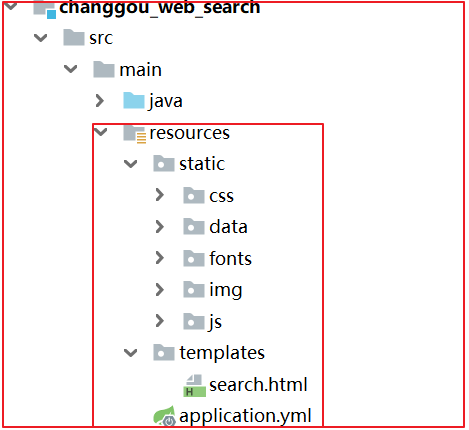
#### 工程创建

在changgou\_web工程中创建changgou\_web\_search工程,并在changgou\_web的pom.xml中引入如下依赖：

*<?***xml version="1.0" encoding="UTF-8"***?>*<**project xmlns="http://maven.apache.org/POM/4.0.0"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"**>  
 <**parent**>  
 <**artifactId**>changgou\_parent</**artifactId**>  
 <**groupId**>com.changgou</**groupId**>  
 <**version**>1.0-SNAPSHOT</**version**>  
 </**parent**>  
 <**modelVersion**>4.0.0</**modelVersion**>  
  
 <**artifactId**>changgou\_web</**artifactId**>  
 <**packaging**>pom</**packaging**>  
 <**modules**>  
 <**module**>changgou\_web\_search</**module**>  
 </**modules**>  
  
 <**dependencies**>  
 <**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-thymeleaf</**artifactId**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-web</**artifactId**>  
 </**dependency**>  
 <**dependency**>  
 <**groupId**>org.springframework.cloud</**groupId**>  
 <**artifactId**>spring-cloud-starter-netflix-eureka-client</**artifactId**>  
 </**dependency**>  
 *<!--feign-->* <**dependency**>  
 <**groupId**>org.springframework.cloud</**groupId**>  
 <**artifactId**>spring-cloud-starter-openfeign</**artifactId**>  
 </**dependency**>  
 *<!--amqp-->* <**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-amqp</**artifactId**>  
 </**dependency**>  
 </**dependencies**>  
</**project**>

#### 静态资源导入

将资源中的页面/前端页面/search.html拷贝到工程的resources/templates目录下，js、css等拷贝到static目录下，如下图：



#### Feign创建

修改changgou-service-search-api，添加com.changgou.search.feign.SkuFeign，实现调用搜索，代码如下：

@FeignClient(name=**"search"**)  
@RequestMapping(**"search"**)  
**public interface** SkuFeign {  
 */\*\*  
 \* 搜索**\** ***@return*** *\*/* @GetMapping  
 **public** Map search(@RequestParam(required = **false**) Map searchMap);  
}

由于以后做搜索都是基于GET请求，所以我们需要将之前的搜索改成GET请求操作，修改changgou-service-search微服务的com.changgou.search.controller.SkuController里面的search方法，代码如下：

@GetMapping  
**public** Map search(@RequestParam(required = **false**) Map searchMap){  
 **return skuService**.search(searchMap);  
}

#### changgou-web-search的pom.xml依赖

*<?***xml version="1.0" encoding="UTF-8"***?>*<**project xmlns="http://maven.apache.org/POM/4.0.0"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"**>  
 <**parent**>  
 <**artifactId**>changgou\_web</**artifactId**>  
 <**groupId**>com.changgou</**groupId**>  
 <**version**>1.0-SNAPSHOT</**version**>  
 </**parent**>  
 <**modelVersion**>4.0.0</**modelVersion**>  
  
 <**artifactId**>changgou\_web\_search</**artifactId**>  
  
 <**dependencies**>  
 <**dependency**>  
 <**groupId**>com.changgou</**groupId**>  
 <**artifactId**>changgou\_service\_search\_api</**artifactId**>  
 <**version**>1.0-SNAPSHOT</**version**>  
 </**dependency**>  
 </**dependencies**>  
</**project**>

#### 搜索调用

在changgou-web-search中创建com.changgou.search.controller.SkuController,实现调用搜索，代码如下：

@Controller  
@RequestMapping(value = **"search"**)  
**public class** SkuController {  
 @Autowired  
 **private** SkuFeign **skuFeign**;  
 */\*\*  
 \* 搜索商品  
 \* 注意此处的@GetMapping()要添加list的url请求，不然会跟SkuFeign中的请求url冲突  
 \*/* @GetMapping(**"list"**)  
 **public** String search(@RequestParam(required = **false**) Map searchMap, Model model){  
 *//查询数据* Map result = **skuFeign**.search(searchMap);  
 model.addAttribute(**"result"**,result);  
 *//响应视图* **return "search"**;  
 }  
}

#### 启动类创建

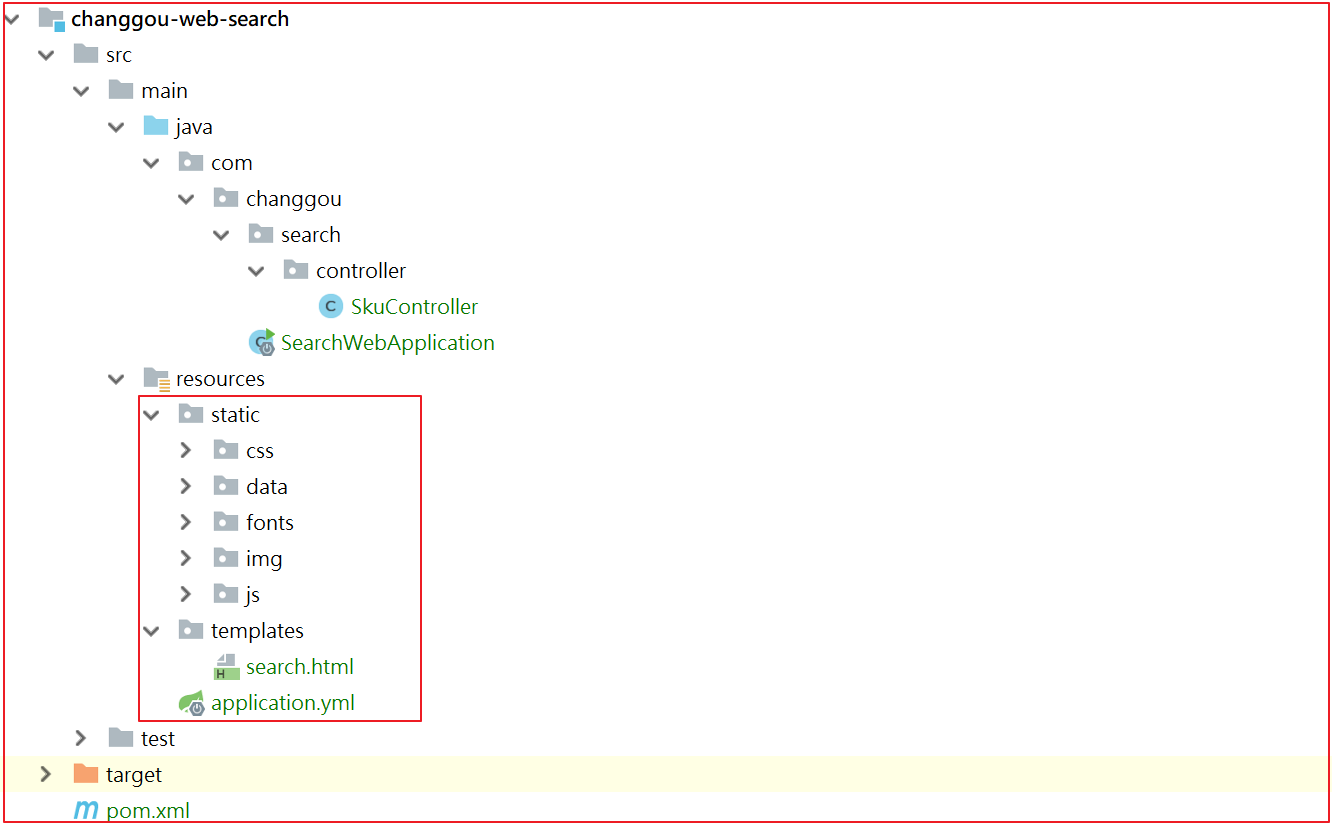
修改changgou-web-search,添加启动类com.changgou.SearchWebApplication，代码如下

@SpringBootApplication  
@EnableEurekaClient  
@EnableFeignClients(basePackages = **"com.changgou.search.feign"**)  
**public class** SearchWebApplication {  
 **public static void** main(String[] args) {  
 SpringApplication.*run*(SearchWebApplication.**class**,args);  
 }  
}

#### application.yml配置文件

**server**:  
 **port**: 18086  
**eureka**:  
 **client**:  
 **service-url**:  
 **defaultZone**: http://127.0.0.1:7001/eureka  
 **instance**:  
 **prefer-ip-address**: **true  
feign**:  
 **hystrix**:  
 **enabled**: **true  
spring**:  
 **thymeleaf**:  
 **cache**: **false  
 application**:  
 **name**: search-web  
 **main**:  
 **allow-bean-definition-overriding**: **true**

#### 项目完整结构



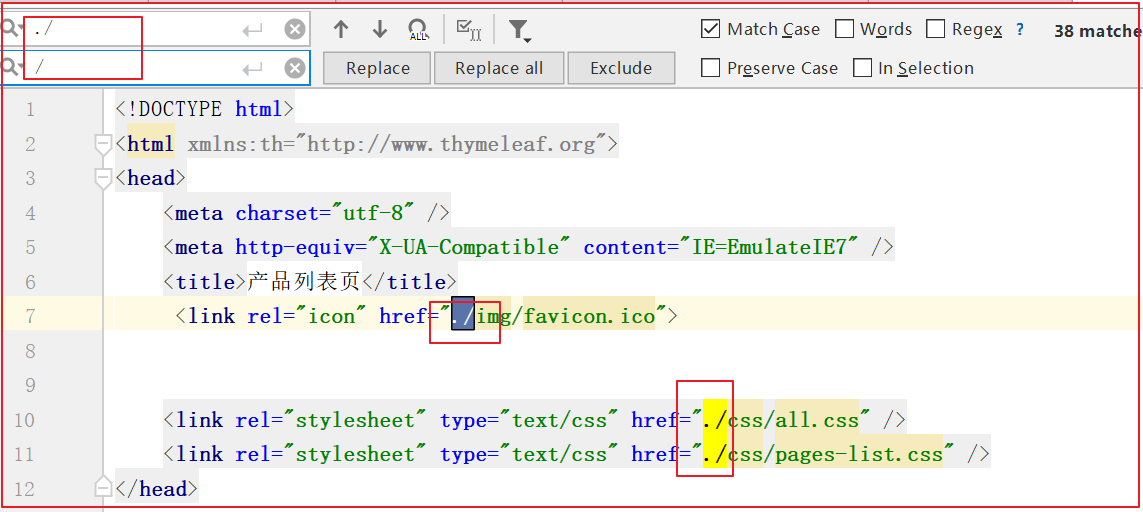
在search.html的头部引入thymeleaf标签

<**html xmlns:th="http://www.thymeleaf.org"**>

测试：<http://localhost:18086/search/list> ,效果如下：



修改search.html所有静态资源的读取目录为绝对路径



### 搜索数据填充

后端搜索到数据后，前端页面进行数据显示，显示的数据分为3部分

1)搜索的数据结果

2)筛选出的数据搜索条件

3)用户已经勾选的数据条件

### 关键字搜索

用户每次输入关键字的时候，直接根据关键字搜索，关键字搜索的数据会存储到result.rows中，页面每次根据result获取rows，然后循环输出即可,同时页面的搜索框每次需要回显搜索的关键词。

实现思路

1.前端表单提交搜索的关键词

2.后端根据关键词进行搜索

3.将搜索条件存储到Model中

4.页面循环迭代输出数据

5.搜索表单回显搜索的关键词

#### 后台搜索实现

修改SkuController的search方法，代码如下：

@GetMapping(**"list"**)  
**public** String search(@RequestParam(required = **false**) Map searchMap, Model model){  
 *//查询数据* Map result = **skuFeign**.search(searchMap);  
 *//返回结果集* model.addAttribute(**"result"**,result);  
 *//返回查询条件* model.addAttribute(**"searchMap"**,searchMap);  
 *//响应视图* **return "search"**;  
}

#### 页面搜索实现

修改search.html

<**form action="/search/list" class="sui-form form-inline"**>  
 *<!--searchAutoComplete-->* <**div class="input-append"**>  
 *<!--#maps.containsKey用于判断某个key是否存在-->* <**input type="text" name="keywords" th:value="${#maps.containsKey(searchMap,'keywords') ? searchMap.keywords : ''}" id="autocomplete" class="input-error input-xxlarge"**/>  
 <**button class="sui-btn btn-xlarge btn-danger" type="submit"**>搜索</**button**>  
 </**div**>  
</**form**>

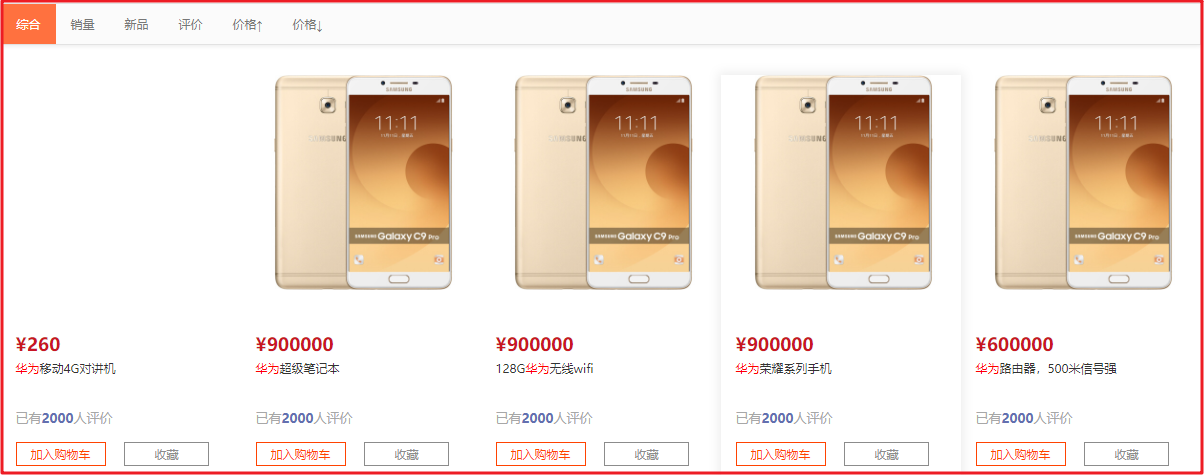
#### 页面结果输出

修改search.html，代码如下：

<**li class="yui3-u-1-5" th:each="sku : ${result.rows}"**>  
 <**div class="list-wrap"**>  
 <**div class="p-img"**>  
 <**a href="item.html" target="\_blank"**><**img th:src="${sku.image}"**/></**a**>  
 </**div**>  
 <**div class="price"**>  
 <**strong**>  
 <**em**>¥</**em**>  
 <**i th:text="${sku.price}"**></**i**>  
 </**strong**>  
 </**div**>  
 <**div class="attr"**>  
 *<!--如果要限制字符输出长度，可以使用th:utext="${#strings.abbreviate(目标,输出字符个数)}"-->* <**a target="\_blank" href="item.html" th:utext="${sku.name}"**></**a**>  
 </**div**>  
 <**div class="commit"**>  
 <**i class="command"**>已有<**span**>2000</**span**>人评价</**i**>  
 </**div**>  
 <**div class="operate"**>  
 <**a href="success-cart.html" target="\_blank" class="sui-btn btn-bordered btn-danger"**>加入购物车</**a**>  
 <**a href="javascript:void(0);" class="sui-btn btn-bordered"**>收藏</**a**>  
 </**div**>  
 </**div**>  
</**li**>

#### 测试

搜索华为关键字,效果如下：



## 搜索条件回显



搜索条件除了关键字外，还有分类、品牌、以及规格，这些在我们前面已经将数据存入到了Map中，我们可以直接从Map中将数据取出，然后在页面输出即可。

分类：result.categoryList

品牌：result.brandList

规格：result.specMap

修改search.html的条件显示部分，代码如下：

<**div class="clearfix selector"**>  
 <**div class="type-wrap" th:if="${#maps.containsKey(result,'categoryList')}"**>  
 <**div class="fl key"**>分类</**div**>  
 <**div class="fl value"**>  
 <**span th:each="category : ${result.categoryList}"**>  
 <**a th:text="${category}"**></**a**>**&nbsp;&nbsp;** </**span**>  
 </**div**>  
 <**div class="fl ext"**></**div**>  
 </**div**>  
 <**div class="type-wrap logo" th:if="${#maps.containsKey(result,'brandList')}"**>  
 <**div class="fl key brand"**>品牌</**div**>  
 <**div class="value logos"**>  
 <**ul class="logo-list"**>  
 <**li th:each="brand : ${result.brandList}"**>  
 <**a th:text="${brand}"**></**a**>**&nbsp;&nbsp;** </**li**>  
 </**ul**>  
 </**div**>  
 <**div class="ext"**>  
 <**a href="javascript:void(0);" class="sui-btn"**>多选</**a**>  
 <**a href="javascript:void(0);"**>更多</**a**>  
 </**div**>  
 </**div**>  
 <**div class="type-wrap" th:each="spec : ${result.specMap}"**>  
 <**div class="fl key" th:text="${spec.key}"**></**div**>  
 <**div class="fl value"**>  
 <**ul class="type-list"**>  
 <**li th:each="val : ${spec.value}"**>  
 <**a th:text="${val}"**></**a**>  
 </**li**>  
 </**ul**>  
 </**div**>  
 <**div class="fl ext"**></**div**>  
 </**div**>

## 隐藏已查询的条件

刚才的查询中我们需要根据用户选择的条件，隐藏相应的查询面板，这里主要是使用

th:unless:条件不满足时，才显示

### 代码实现

<**div class="type-wrap" th:if="${#maps.containsKey(result,'categoryList')}" th:unless="${#maps.containsKey(searchMap,'category')}"**>  
 <**div class="fl key"**>分类</**div**>  
 <**div class="fl value"**>  
 <**span th:each="category : ${result.categoryList}"**>  
 <**a th:text="${category}"**></**a**>**&nbsp;&nbsp;** </**span**>  
 </**div**>  
 <**div class="fl ext"**></**div**>  
</**div**>  
<**div class="type-wrap logo" th:if="${#maps.containsKey(result,'brandList')}" th:unless="${#maps.containsKey(searchMap,'brand')}"**>  
 <**div class="fl key brand"**>品牌</**div**>  
 <**div class="value logos"**>  
 <**ul class="logo-list"**>  
 <**li th:each="brand : ${result.brandList}"**>  
 <**a th:text="${brand}"**></**a**>**&nbsp;&nbsp;** </**li**>  
 </**ul**>  
 </**div**>  
 <**div class="ext"**>  
 <**a href="javascript:void(0);" class="sui-btn"**>多选</**a**>  
 <**a href="javascript:void(0);"**>更多</**a**>  
 </**div**>  
</**div**>  
<**div class="type-wrap" th:each="spec : ${result.specMap}" th:unless="${#maps.containsKey(searchMap,'spec\_' + spec.key)}"**>  
 <**div class="fl key" th:text="${spec.key}"**></**div**>  
 <**div class="fl value"**>  
 <**ul class="type-list"**>  
 <**li th:each="val : ${spec.value}"**>  
 <**a th:text="${val}"**></**a**>  
 </**li**>  
 </**ul**>  
 </**div**>  
 <**div class="fl ext"**></**div**>  
</**div**>  
<**div class="type-wrap" th:unless="${#maps.containsKey(searchMap,'price')}"**>  
 <**div class="fl key"**>价格</**div**>  
 <**div class="fl value"**>  
 <**ul class="type-list"**>  
 <**li**>  
 <**a**>0-500元</**a**>  
 </**li**>  
 <**li**>  
 <**a**>500-1000元</**a**>  
 </**li**>  
 <**li**>  
 <**a**>1000-1500元</**a**>  
 </**li**>  
 <**li**>  
 <**a**>1500-2000元</**a**>  
 </**li**>  
 <**li**>  
 <**a**>2000-3000元 </**a**>  
 </**li**>  
 <**li**>  
 <**a**>3000元以上</**a**>  
 </**li**>  
 </**ul**>  
 </**div**>  
 <**div class="fl ext"**>  
 </**div**>  
</**div**>

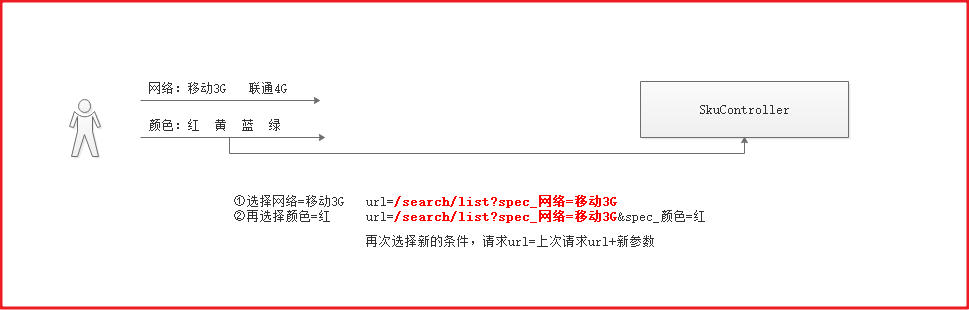
### 测试

浏览器输入相应的参数测试：

<http://localhost:18086/search/list?category=笔记本&brand=华为&spec_网络=移动4G&spec_颜色=红&price=0-500>



## 搜索条件点击特效



用户每次点击搜索的时候，其实在上次搜索的基础之上加上了新的搜索条件，也就是在上一次请求的URL后面追加了新的搜索条件，我们可以在后台每次拼接组装出上次搜索的URL，然后每次将URL存入到Model中，页面每次点击不同条件的时候，从Model中取出上次请求的URL，然后再加上新点击的条件参数实现跳转即可。

### 后台记录搜索URL

修改SkuController，添加组装URL的方法，并将组装好的URL存储起来,代码如下：

@GetMapping(**"list"**)  
**public** String search(@RequestParam(required = **false**) Map searchMap, Model model){  
 *//查询数据* Map result = **skuFeign**.search(searchMap);  
 *//返回结果集* model.addAttribute(**"result"**,result);  
 *//返回查询条件* model.addAttribute(**"searchMap"**,searchMap);  
  
 *//获取url* String url = **this**.getUrl(searchMap);  
 model.addAttribute(**"url"**, url);  
 *//响应视图* **return "search"**;  
}  
  
*/\*\*  
 \* 把Map转换成url  
 \** ***@param searchMap*** *\** ***@return*** *\*/***private** String getUrl(Map<String,String> searchMap){  
 *// /search/list?category=笔记本&brand=华为&spec\_网络=移动4G&price=0-500* String url = **"/search/list"**;  
 *//有参数* **if**(searchMap != **null**){  
 url += **"?"**;  
 **for** (String key : searchMap.keySet()) {  
 url += key + **"="** + searchMap.get(key) + **"&"**;  
 }  
 *//循环完毕后删除最后一个&* url = url.substring(0, url.length() - 1);  
 }  
 **return** url;  
}

### 页面搜索对接

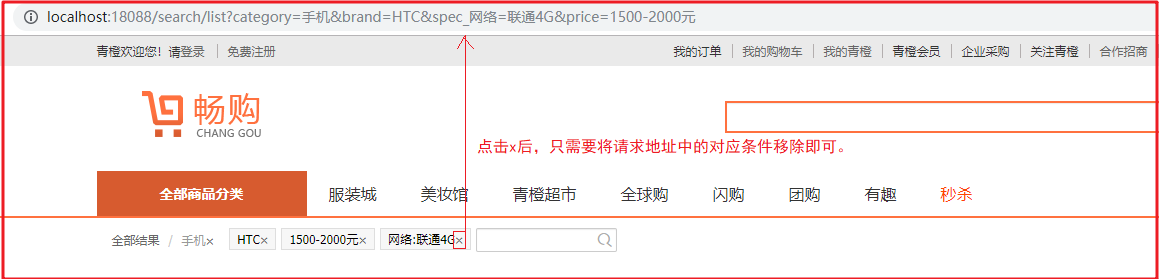
th:href 是超链接的语法，例如：th:href="@{${url}(price='500-1000元')}"表示请求地址是取url参数的值，同时向后台传递参数price的值为500-100元。

语法：@{ ${请求地址} (参数名=参数值)}

代码

*<!--selector-->*<**div class="clearfix selector"**>  
 <**div class="type-wrap" th:if="${#maps.containsKey(result,'categoryList')}" th:unless="${#maps.containsKey(searchMap,'category')}"**>  
 <**div class="fl key"**>分类</**div**>  
 <**div class="fl value"**>  
 <**span th:each="category : ${result.categoryList}"**>  
 <**a th:href="@{ ${url} (category=${category})}" th:text="${category}"**></**a**>**&nbsp;&nbsp;** </**span**>  
 </**div**>  
 <**div class="fl ext"**></**div**>  
 </**div**>  
 <**div class="type-wrap logo" th:if="${#maps.containsKey(result,'brandList')}" th:unless="${#maps.containsKey(searchMap,'brand')}"**>  
 <**div class="fl key brand"**>品牌</**div**>  
 <**div class="value logos"**>  
 <**ul class="logo-list"**>  
 <**li th:each="brand : ${result.brandList}"**>  
 <**a th:href="@{ ${url} (brand=${brand})}" th:text="${brand}"**></**a**>**&nbsp;&nbsp;** </**li**>  
 </**ul**>  
 </**div**>  
 <**div class="ext"**>  
 <**a href="javascript:void(0);" class="sui-btn"**>多选</**a**>  
 <**a href="javascript:void(0);"**>更多</**a**>  
 </**div**>  
 </**div**>  
 <**div class="type-wrap" th:each="spec : ${result.specMap}" th:unless="${#maps.containsKey(searchMap,'spec\_' + spec.key)}"**>  
 <**div class="fl key" th:text="${spec.key}"**></**div**>  
 <**div class="fl value"**>  
 <**ul class="type-list"**>  
 <**li th:each="val : ${spec.value}"**>  
 <**a th:href="@{ ${url} ('spec\_'+ ${spec.key}=${val})}" th:text="${val}"**></**a**>  
 </**li**>  
 </**ul**>  
 </**div**>  
 <**div class="fl ext"**></**div**>  
 </**div**>  
 <**div class="type-wrap" th:unless="${#maps.containsKey(searchMap,'price')}"**>  
 <**div class="fl key"**>价格</**div**>  
 <**div class="fl value"**>  
 <**ul class="type-list"**>  
 <**li**>  
 <**a th:href="@{ ${url} (price='0-500')}"**>0-500元</**a**>  
 </**li**>  
 <**li**>  
 <**a th:href="@{ ${url} (price='500-1000')}"**>500-1000元</**a**>  
 </**li**>  
 <**li**>  
 <**a th:href="@{ ${url} (price='1000-1500')}"**>1000-1500元</**a**>  
 </**li**>  
 <**li**>  
 <**a th:href="@{ ${url} (price='1500-2000')}"**>1500-2000元</**a**>  
 </**li**>  
 <**li**>  
 <**a th:href="@{ ${url} (price='2000-3000')}"**>2000-3000元 </**a**>  
 </**li**>  
 <**li**>  
 <**a th:href="@{ ${url} (price='3000')}"**>3000元以上</**a**>  
 </**li**>  
 </**ul**>  
 </**div**>  
 <**div class="fl ext"**>  
 </**div**>  
 </**div**>

## 移除搜索条件



如上图，用户点击条件搜索后，要将选中的条件显示出来，并提供移除条件的x按钮,显示条件我们可以从searchMap中获取，移除其实就是将之前的请求地址中的指定条件删除即可。

### 条件显示

修改search.html，代码如下：

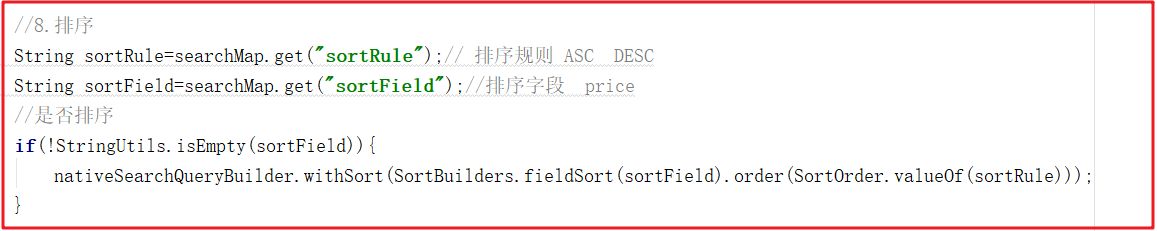
<**ul class="fl sui-breadcrumb"**>  
 <**li**>  
 <**a href="#"**>全部结果</**a**>  
 </**li**>  
 <**li th:if="${#maps.containsKey(searchMap,'category')}" class="active"**>  
 <**span th:text="${searchMap.category}"**></**span**><**i**>×</**i**>  
 </**li**>  
</**ul**>  
<**ul class="fl sui-tag"**>  
 <**li class="with-x" th:if="${#maps.containsKey(searchMap,'brand')}"** >  
 <**span th:text="${searchMap.brand}"**></**span**><**i**>×</**i**>  
 </**li**>  
 <**li class="with-x" th:if="${#maps.containsKey(searchMap,'price')}"** >  
 <**span th:text="${searchMap.price}"**></**span**><**i**>×</**i**>  
 </**li**>  
 <**li th:each="spec : ${searchMap}" th:if="${#strings.startsWith(spec.key,'spec\_')}" class="with-x"**>  
 <**span th:text="${#strings.replace(spec.key,'spec\_','')}"**></**span**>:  
 <**span th:text="${spec.value}"**></**span**>  
 <**i**>×</**i**>  
 </**li**>  
</**ul**>

### 移除搜素条件

修改search.html，移除分类、品牌、价格、规格搜索条件，代码如下：

<**ul class="fl sui-breadcrumb"**>  
 <**li**>  
 <**a href="#"**>全部结果</**a**>  
 </**li**>  
 <**li th:if="${#maps.containsKey(searchMap,'category')}" class="active"**>  
 <**span th:text="${searchMap.category}"**></**span**>  
 <**a th:href="@{${#strings.replace(url,'category='+searchMap.category,'')}}"**>×</**a**>  
 </**li**>  
</**ul**>  
<**ul class="fl sui-tag"**>  
 <**li class="with-x" th:if="${#maps.containsKey(searchMap,'brand')}"** >  
 <**span th:text="${searchMap.brand}"**></**span**>  
 <**a th:href="@{${#strings.replace(url,'brand='+searchMap.brand,'')}}"**>×</**a**>  
 </**li**>  
 <**li class="with-x" th:if="${#maps.containsKey(searchMap,'price')}"** >  
 <**span th:text="${searchMap.price}"**></**span**>  
 <**a th:href="@{${#strings.replace(url,'price='+searchMap.price,'')}}"**>×</**a**>  
 </**li**>  
 <**li th:each="spec : ${searchMap}" th:if="${#strings.startsWith(spec.key,'spec\_')}" class="with-x"**>  
 <**span th:text="${#strings.replace(spec.key,'spec\_','')}"**></**span**>:  
 <**span th:text="${spec.value}"**></**span**>  
 <**a th:href="@{${#strings.replace(url,spec.key + '=' + spec.value,'')}}"**>×</**a**>  
 </**li**>  
</**ul**>

## 排序(学员作业)



上图代码是排序代码，需要2个属性，sortRule:排序规则，ASC或者DESC，sortField:排序的域，前端每次只需要将这2个域的值传入到后台即可实现排序。

### 后台组装URL排除排序

*/\*\*  
 \* 把Map转换成url  
 \** ***@param searchMap*** *\** ***@return*** *\*/***private** String getUrl(Map<String,String> searchMap){  
 *// /search/list?category=笔记本&brand=华为&spec\_网络=移动4G&price=0-500* String url = **"/search/list"**;  
 *//有参数* **if**(searchMap != **null**){  
 url += **"?"**;  
 **for** (String key : searchMap.keySet()) {  
 *//如果是排序的参数，不拼接到url上，便于下次换种方式排序* **if**(key.indexOf(**"sort"**) > -1){  
 **continue**;  
 }  
 url += key + **"="** + searchMap.get(key) + **"&"**;  
 }  
 *//循环完毕后删除最后一个&* url = url.substring(0, url.length() - 1);  
 }  
 **return** url;  
}

### 前端排序实现

修改search.html，实现排序，代码如下：

<**ul class="sui-nav"**>  
 <**li class="active"**>  
 <**a href="#"**>综合</**a**>  
 </**li**>  
 <**li**>  
 <**a href="#"**>销量</**a**>  
 </**li**>  
 <**li**>  
 <**a href="#"**>新品</**a**>  
 </**li**>  
 <**li**>  
 <**a href="#"**>评价</**a**>  
 </**li**>  
 <**li**>  
 <**a th:href="@{ ${url} (sortRule='ASC',sortField='price')}"**>价格↑</**a**>  
 </**li**>  
 <**li**>  
 <**a th:href="@{ ${url} (sortRule='DESC',sortField='price')}"**>价格↓</**a**>  
 </**li**>  
</**ul**>

这一块我们实现了价格排序，同学们课后去实现以下销量和新品排序。

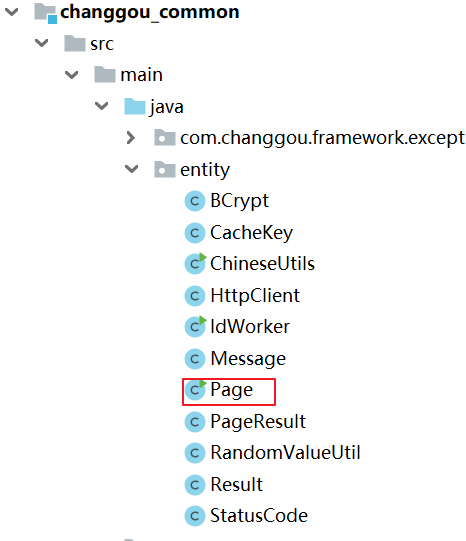
## 分页

真实的分页应该像百度那样，如下图：



### 分页工具类定义

在comm工程中添加Page分页对象，路径如下：



### 分页实现

由于这里需要获取分页信息，我们可以在changgou-service-search服务中修改搜索方法实现获取分页数据，修改com.changgou.search.service.impl.SkuServiceImpl的search方法，在return之前添加如下方法获取份额与数据：

*//5、包装结果并返回*map.put(**"rows"**, page.getContent());  
map.put(**"total"**, page.getTotalElements());  
map.put(**"totalPages"**, page.getTotalPages());  
**int** pageNum = query.getPageable().getPageNumber() + 1; *//当前页*map.put(**"pageNum"**, pageNum);  
**int** pageSize = query.getPageable().getPageSize();*//每页查询的条数*map.put(**"pageSize"**, pageSize);

修改SkuController,实现分页信息封装，代码如下：

@GetMapping(**"list"**)  
**public** String search(@RequestParam(required = **false**) Map searchMap, Model model){  
 *//查询数据* Map result = **skuFeign**.search(searchMap);  
 *//返回结果集* model.addAttribute(**"result"**,result);  
 *//返回查询条件* model.addAttribute(**"searchMap"**,searchMap);  
  
 *//获取url* String url = **this**.getUrl(searchMap);  
 model.addAttribute(**"url"**, url);  
  
 *//返回分页参数* Page page = **new** Page(  
 **new** Long(result.get(**"total"**).toString()),  
 **new** Integer(result.get(**"pageNum"**).toString()),  
 **new** Integer(result.get(**"pageSize"**).toString())  
 );  
 model.addAttribute(**"page"**, page);  
 *//响应视图* **return "search"**;  
}  
  
*/\*\*  
 \* 把Map转换成url  
 \** ***@param searchMap*** *\** ***@return*** *\*/***private** String getUrl(Map<String,String> searchMap){  
 *// /search/list?category=笔记本&brand=华为&spec\_网络=移动4G&price=0-500* String url = **"/search/list"**;  
 *//有参数* **if**(searchMap != **null**){  
 url += **"?"**;  
 **for** (String key : searchMap.keySet()) {  
 *//如果是排序的参数，不拼接到url上，便于下次换种方式排序  
 //排除当前页拼接url参数* **if**(key.indexOf(**"sort"**) > -1 || **"pageNum"**.equals(key)){  
 **continue**;  
 }  
 url += key + **"="** + searchMap.get(key) + **"&"**;  
 }  
 *//循环完毕后删除最后一个&* url = url.substring(0, url.length() - 1);  
 }  
 **return** url;  
}

### 页面分页实现

修改search.html，实现分页查询，代码如下：

<**div class="fr page"**>  
 <**div class="sui-pagination pagination-large"**>  
 <**ul**>  
 <**li class="prev disabled"**>  
 <**a th:href="@{${url} (pageNum=${page.upper})}"**>«上一页</**a**>  
 </**li**>  
 <**li th:each="i : ${#numbers.sequence(page.lpage,page.rpage)}"  
 th:class="${i} == ${page.currentpage} ? 'active' : ''"**>  
 <**a th:href="@{${url} (pageNum=${i})}" th:text="${i}"**></**a**>  
 </**li**>  
 <**li class="dotted"**><**span**>...</**span**></**li**>  
 <**li class="next"**>  
 <**a th:href="@{${url} (pageNum=${page.next})}"**>下一页»</**a**>  
 </**li**>  
 </**ul**>  
 <**div**><**span**>共<**span th:text="${page.last}"**></**span**>页**&nbsp;**</**span**><**span**>  
到第  
<**input type="text" class="page-num"**>  
页 <**button class="page-confirm" onclick="***alert*(1)**"**>确定</**button**></**span**></**div**>  
 </**div**>  
 </**div**>

## 特殊关键字处理(学员作业)

如果请求以GET请求，参数会拼接到URL上去，此时后台传输数据的时候，会将+>等特殊字符变成空格，我们需要对它进行转义，我们这里很有可能在规格中存在特殊字符，例如移动3G+联通3G这里的加号就属于特殊字符，我们可以在后台接收的参数那里做统一处理。

修改com.changgou.search.controller.SkuController,添加一个方法处理提交的参数，代码如下：

/\*\*\*\*

\* 替换特殊字符

\* @param searchMap

\*/

**public** **void** **handlerSearchMap**(Map<String,String> searchMap){

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

**for** (Map.Entry<String, String> entry : searchMap.entrySet()) {

**if**(entry.getKey().startsWith("spec\_")){

entry.setValue(entry.getValue().replace("+","%2B"));

}

}

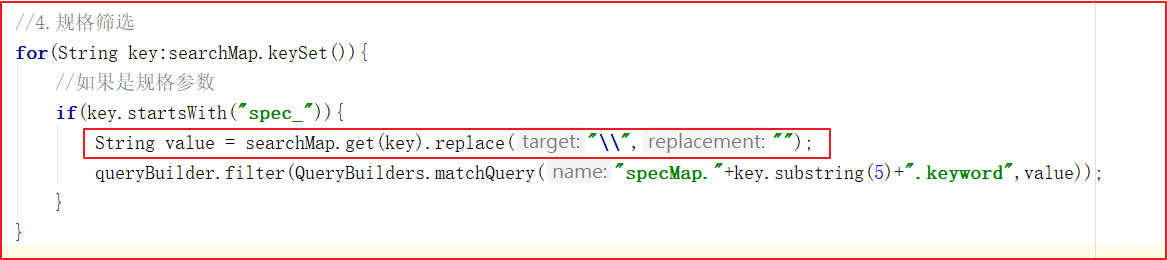
}

}

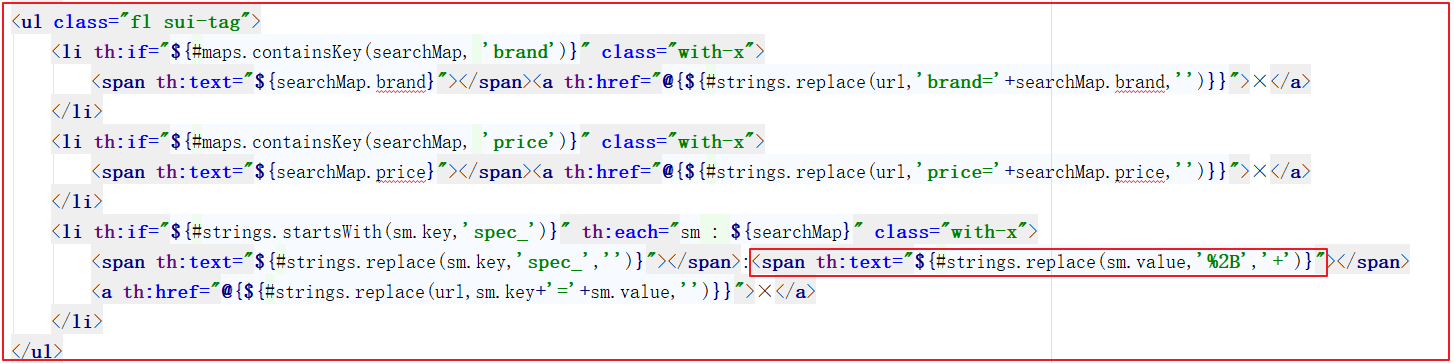
每次执行的时候，调用上面搜索方法处理。



修改changgou-service-search微服务的SkuServiceImpl类的buildBasicQuery方法，添加将\\替换的操作，代码如下：

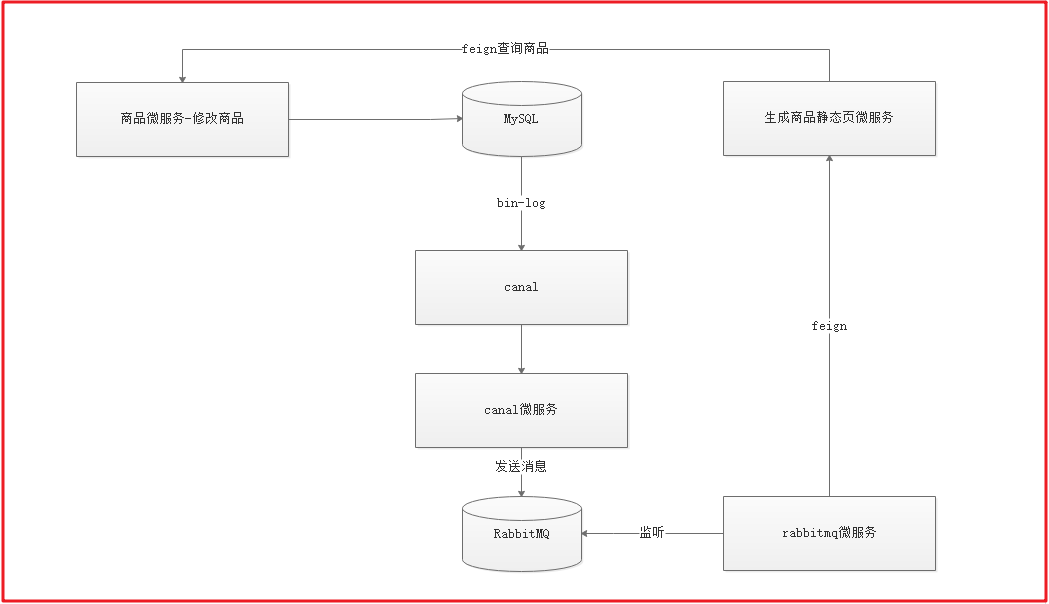


页面显示也做下简单处理即可,修改search.html，代码如下：



# 畅购商品详情页(学员作业)

当系统审核完成商品，需要将商品详情页进行展示，那么采用静态页面生成的方式生成，并部署到高性能的web服务器中进行访问是比较合适的。所以，开发流程如下图所示：



此处MQ我们使用Rabbitmq即可。

执行步骤解释：

* 系统管理员（商家运维人员）修改或者审核商品的时候，会触发canal监控数据
* canal微服务获取修改数据后，向RabbitMQ发送消息
* 创建一个RabbitMQ微服务监听消息，并调用生成静态页微服务
* 静态页微服务只负责使用thymeleaf的模板技术生成静态页
* 运维部署静态页到服务器上，并部署nginx作为http服务器
* 用户在浏览器中输入地址访问网关路由到nginx并展示静态页面。

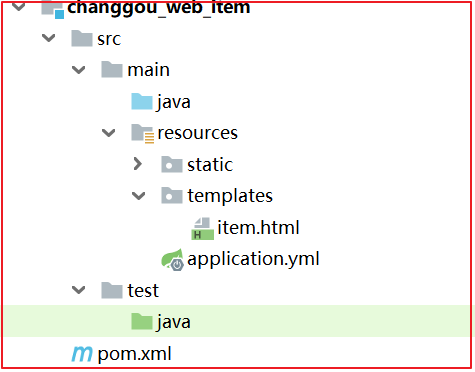
就是将展示的商品页面转化为静态页面，同时监听页面的修改，及时将静态页面进行更新

## 商品静态化微服务

该微服务只用于生成商品静态页，不做其他事情。

### 搭建项目

在changgou\_web下创建一个名称为changgou\_web\_item的模块，复制静态页面，如图：



#### 加入goods-api的依赖

*<?***xml version="1.0" encoding="UTF-8"***?>*<**project xmlns="http://maven.apache.org/POM/4.0.0"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"**>  
 <**parent**>  
 <**artifactId**>changgou\_web</**artifactId**>  
 <**groupId**>com.changgou</**groupId**>  
 <**version**>1.0-SNAPSHOT</**version**>  
 </**parent**>  
 <**modelVersion**>4.0.0</**modelVersion**>  
  
 <**artifactId**>changgou\_web\_item</**artifactId**>  
  
 <**dependencies**>  
 <**dependency**>  
 <**groupId**>com.changgou</**groupId**>  
 <**artifactId**>changgou\_service\_goods\_api</**artifactId**>  
 <**version**>1.0-SNAPSHOT</**version**>  
 </**dependency**>  
 </**dependencies**>  
</**project**>

#### 修改application.yml的配置

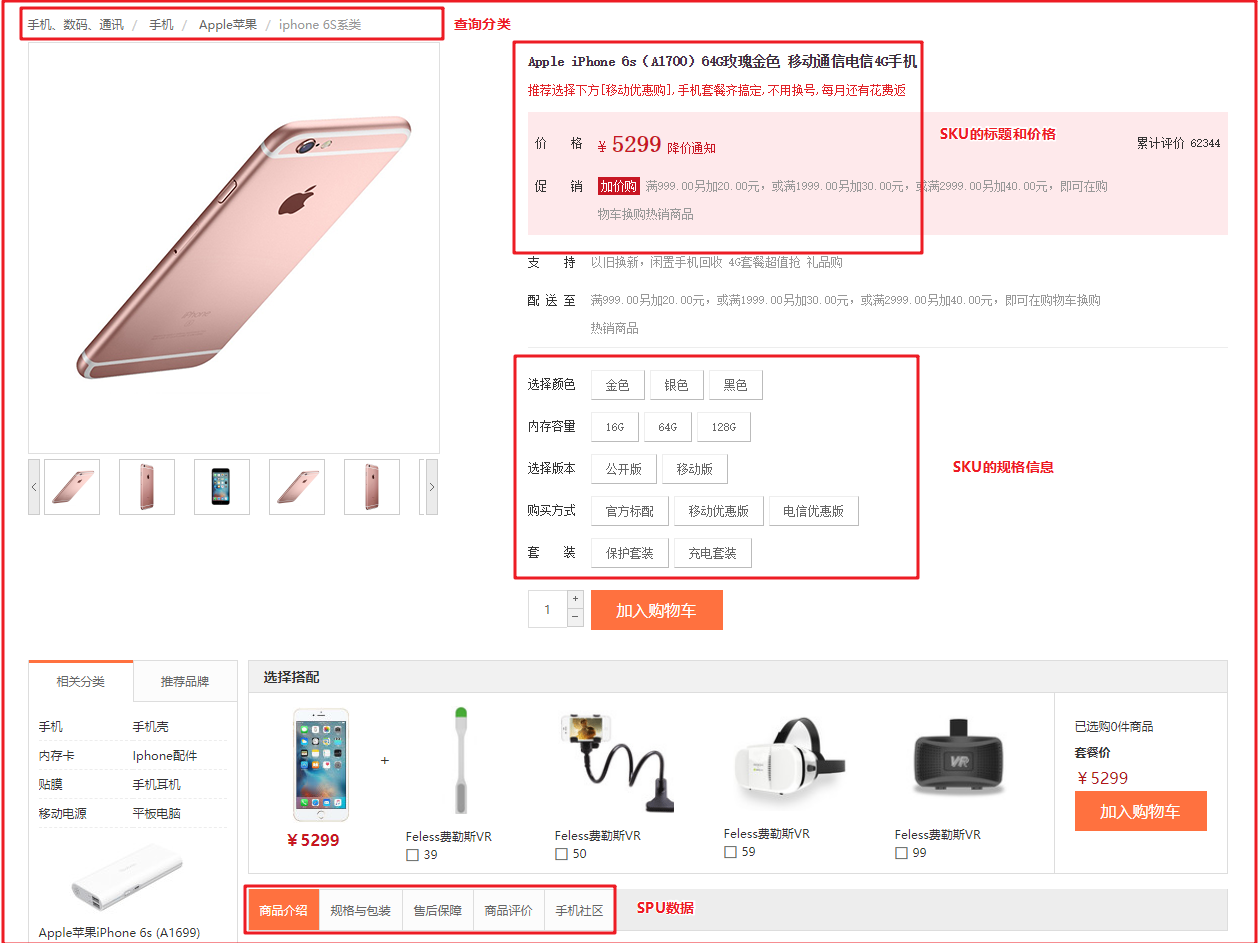
**server**:  
 **port**: 18087  
**eureka**:  
 **client**:  
 **service-url**:  
 **defaultZone**: http://127.0.0.1:7001/eureka  
 **instance**:  
 **prefer-ip-address**: **true  
feign**:  
 **hystrix**:  
 **enabled**: **true  
spring**:  
 **thymeleaf**:  
 **cache**: **false  
 application**:  
 **name**: item  
 **main**:  
 **allow-bean-definition-overriding**: **true** *#rabbitmq:  
 # host: 192.168.211.132  
# 生成静态页的位置***pagepath**: D:/tmp

#### 创建启动引导类

@SpringBootApplication  
@EnableEurekaClient  
@EnableFeignClients(basePackages = **"com.changgou.goods.feign"**)  
**public class** ItemApplication {  
 **public static void** main(String[] args) {  
 SpringApplication.*run*(ItemApplication.**class**,args);  
 }  
}

## 生成静态页

页面发送请求，传递要生成的静态页的的商品的SpuID.后台controller 接收请求，调用thyemleaf的原生API生成商品静态页。



上图是要生成的商品详情页，从图片上可以看出需要查询SPU的3个分类作为面包屑显示，同时还需要查询SKU和SPU信息。

### Feign创建

一会儿需要查询SPU和SKU以及Category，所以我们需要先创建Feign，修改changgou\_service\_goods\_api,添加CategoryFeign，并在CategoryFeign中添加根据ID查询分类数据，代码如下：

@FeignClient(name = **"goods"**)  
@RequestMapping(**"category"**)  
**public interface** CategoryFeign {  
 */\*\*  
 \* 获取分类的对象信息  
 \** ***@param id*** *\** ***@return*** *\*/* @GetMapping(**"{id}"**)  
 **public** Result<Category> findById(@PathVariable(name = **"id"**) Integer id);  
}

在changgou-service-goods-api,添加SpuFeign,并添加根据SpuID查询Spu与Sku信息，代码如下：

@FeignClient(name = **"goods"**)  
@RequestMapping(**"spu"**)  
**public interface** SpuFeign {  
 */\*\*  
 \* 根据id查询spu与sku列表  
 \** ***@param id*** *\** ***@return*** *\*/* @GetMapping(**"goods/{id}"**)  
 **public** Result<Goods> findById(@PathVariable Long id);  
}

### 静态页生成代码

#### 创建service

**接口代码：**

**public interface** PageService {  
 */\*\*  
 \* 根据商品的ID 生成静态页  
 \** ***@param spuId*** *\*/* **public void** createPageHtml(Long spuId) ;  
}

**实现类**

步骤：

//1、创建上下文对象-context = new Context()

//2、识别并生成静态页目录-File.exists

//3、创建静态页面文件对象=dest=new File(dir,spuId+".html")

//4、创建文件输出对象-out = new PrintWriter(dest,"UTF-8")

//5、输出文件-templateEngine.process(模板名称,内容上下文,输出对象)

代码：

@Service  
**public class** PageServiceImpl **implements** PageService {  
 @Autowired  
 **private** TemplateEngine **templateEngine**;  
 @Value(**"${pagepath}"**)  
 **private** String **pagepath**;  
 @Override  
 **public void** createPageHtml(Long spuId) {  
 **try** {  
 *//1、创建上下文对象-context = new Context()* Context context = **new** Context();  
 *//2、识别并生成静态页目录-File.exists* File dir = **new** File(**pagepath**);  
 **if**(!dir.exists()){  
 *//创建级连目录* dir.mkdirs();  
 }  
 *//3、创建静态页面文件对象=dest=new File(dir,spuId+".html")* File dest = **new** File(dir,spuId + **".html"**);  
 *//4、创建文件输出对象-out = new PrintWriter(dest,"UTF-8")* PrintWriter out = **new** PrintWriter(dest,**"UTF-8"**);  
 *//5、输出文件-templateEngine.process(模板名称,内容上下文,输出对象)* **templateEngine**.process(**"item"**,context,out);  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 }  
}

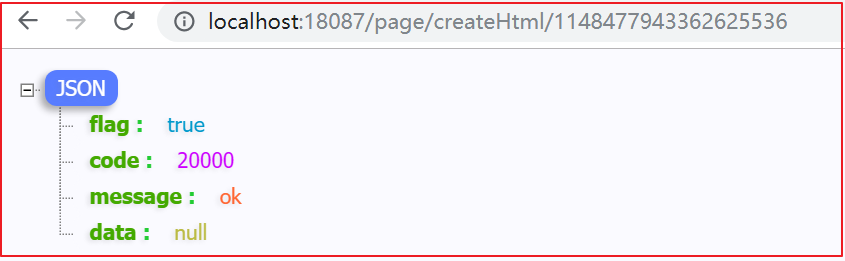
#### 创建Controller

在changgou-web-item中创建com.changgou.item.controller.PageController用于接收请求，测试生成静态页

@RestController  
@RequestMapping(**"page"**)  
**public class** PageController {  
  
 @Autowired  
 **private** PageService **pageService**;  
 */\*\*  
 \* 生成静态页面  
 \** ***@param id*** *\** ***@return*** *\*/* @RequestMapping(**"/createHtml/{id}"**)  
 **public** Result createHtml(@PathVariable(name=**"id"**) Long id){  
 **pageService**.createPageHtml(id);  
 **return new** Result(**true**, StatusCode.***OK***,**"ok"**);  
 }  
}

#### 测试

启动ItemApplication引导类，访问：<http://localhost:18087/page/createHtml/1148477943362625536>



在D:/tmp目录下也生成了相应的静态文件

### 查询静态页面所需数据

修改PageServiceImpl的逻辑实现数据加载

@Service  
**public class** PageServiceImpl **implements** PageService {  
 @Autowired  
 **private** TemplateEngine **templateEngine**;  
  
 @Autowired  
 **private** SpuFeign **spuFeign**;  
 @Autowired  
 **private** CategoryFeign **categoryFeign**;  
  
 @Value(**"${pagepath}"**)  
 **private** String **pagepath**;  
  
 */\*\*  
 \* 构建数据模型  
 \** ***@param spuId*** *\** ***@return*** *\*/* **private** Map<String,Object> buildDataModel(Long spuId){  
 *//构建数据模型* Map<String,Object> dataMap = **new** HashMap<>();  
 *//获取spu和SKU列表* Result<Goods> result = **spuFeign**.findById(spuId);  
 Spu spu = result.getData().getSpu();  
  
 *//获取分类信息* dataMap.put(**"category1"**,**categoryFeign**.findById(spu.getCategory1Id()).getData());  
 dataMap.put(**"category2"**,**categoryFeign**.findById(spu.getCategory2Id()).getData());  
 dataMap.put(**"category3"**,**categoryFeign**.findById(spu.getCategory3Id()).getData());  
 **if**(spu.getImages()!=**null**) {  
 dataMap.put(**"imageList"**, spu.getImages().split(**","**));  
 }  
 *//获取规格数据* dataMap.put(**"specificationList"**, JSON.*parseObject*(spu.getSpecItems(),Map.**class**));  
 dataMap.put(**"spu"**,spu);  
  
 *//返回sku列表* dataMap.put(**"skuList"**,result.getData().getSkuList());  
 **return** dataMap;  
 }  
  
 @Override  
 **public void** createPageHtml(Long spuId) {  
 **try** {  
 *//1、创建上下文对象-context = new Context()* Context context = **new** Context();  
 *//构建数据模型* Map<String, Object> map = **this**.buildDataModel(spuId);  
 context.setVariables(map);  
  
 *//2、识别并生成静态页目录-File.exists* File dir = **new** File(**pagepath**);  
 **if**(!dir.exists()){  
 *//创建级连目录* dir.mkdirs();  
 }  
 *//3、创建静态页面文件对象=dest=new File(dir,spuId+".html")* File dest = **new** File(dir,spuId + **".html"**);  
 *//4、创建文件输出对象-out = new PrintWriter(dest,"UTF-8")* PrintWriter out = **new** PrintWriter(dest,**"UTF-8"**);  
 *//5、输出文件-templateEngine.process(模板名称,内容上下文,输出对象)* **templateEngine**.process(**"item"**,context,out);  
 } **catch** (Exception e) {  
 e.printStackTrace();  
 }  
 }  
  
}

### 模板填充

先把模板的item.html的头部引入thymeleaf标签

<**html xmlns:th="http://www.thymeleaf.org"**>

#### 分类面包屑数据

修改item.html，填充三个分类数据作为面包屑，代码如下：

<**div class="crumb-wrap"**>  
 <**ul class="sui-breadcrumb"**>  
 <**li**>  
 <**a href="#" th:text="${category1.name}"**></**a**>  
 </**li**>  
 <**li**>  
 <**a href="#" th:text="${category2.name}"**></**a**>  
 </**li**>  
 <**li**>  
 <**a href="#" th:text="${category3.name}"**></**a**>  
 </**li**>  
 </**ul**>  
</**div**>

#### 商品图片

修改item.html，将商品图片信息输出，在真实工作中需要做空判断，代码如下：

<**div class="fl preview-wrap"**>  
 *<!--放大镜效果-->* <**div class="zoom"**>  
 *<!--默认第一个预览-->* <**div id="preview" class="spec-preview"**>  
<**span class="jqzoom"**>  
 <**img th:jqimg="${imageList[0]}" th:src="${imageList[0]}"**/>  
</**span**>  
 </**div**>  
 *<!--下方的缩略图-->* <**div class="spec-scroll"**>  
 <**a class="prev"**>**&lt;**</**a**>  
 *<!--左右按钮-->* <**div class="items"**>  
 <**ul**>  
 <**li th:each="imge : ${imageList}"**>  
 <**img th:src="${imge}" th:bimg="${imge}" onmousemove="***preview*(**this**)**"**/>  
 </**li**>  
 </**ul**>  
 </**div**>  
 <**a class="next"**>**&gt;**</**a**>  
 </**div**>  
 </**div**>  
 </**div**>

#### 规格输出

<**div id="specification" class="summary-wrap clearfix"**>  
 <**dl th:each="spec : ${specificationList}"**>  
 <**dt**>  
 <**div class="fl title"**>  
 <**i th:text="${spec.key}"**></**i**>  
 </**div**>  
 </**dt**>  
  
 <**dd th:each="val : ${spec.value}"**>  
 <**a href="javascript:;" class="selected"**>  
 <**i th:text="${val}"**></**i**>  
 <**span title="点击取消选择"**>**&nbsp;**</**span**>  
 </**a**>  
 </**dd**>  
 *<!--<dd v-for="coloritem in goodsdetail.color" :key="coloritem.index">  
 <a href="javascript:;">{{coloritem}}</a>  
 </dd>-->* </**dl**>  
</**div**>

#### 默认SKU显示

静态页生成后，需要显示默认的Sku，我们这里默认显示第1个Sku即可，这里可以结合着Vue一起实现。可以先定义一个集合，再定义一个spec和sku，用来存储当前选中的Sku信息和Sku的规格，代码如下：

*<!--thymeleaf整合Vue需要在此处加入 th:inline="javascript"-->*<**script th:inline="javascript"**>  
 **var *item*** = **new Vue**({  
 **el**:**"#app"**,  
 **data**:{  
 *//此处城要两个中括号，表示从Model中获取skuList参数,设置给vue的skuList变量* **skuList**:[[${skuList}]],  
 **sku**:{}, *//当前选中的sku* **spec**:{} *//选中的sku规格* },  
 created:**function** () {  
 *//默认加载第一个sku,使用深克隆* **this**.**sku** = ***JSON***.parse(***JSON***.stringify(**this**.**skuList**[0]));  
 *//加载第一个sku的规格* **this**.**spec** = ***JSON***.parse(**this**.**sku**.**spec**);  
 }  
 });  
</**script**>

页面显示默认的Sku信息

<**div class="fr itemInfo-wrap" id="app"**>  
 <**div class="sku-name"**>  
 <**h4**>{{sku.name}}</**h4**>  
 </**div**>  
 <**div class="news"**>  
 <**span th:text="${spu.caption}"**></**span**>  
 </**div**>  
 <**div class="summary"**>  
 <**div class="summary-wrap"**>  
 <**div class="fl title"**>  
 <**i**>价　　格</**i**>  
 </**div**>  
 <**div class="fl price"**>  
 <**i**>¥</**i**>  
 <**em**>{{sku.price}}</**em**>  
 <**span**>降价通知</**span**>  
 </**div**>  
 <**div class="fr remark"**>  
 <**i**>累计评价</**i**>  
 <**em**>10000</**em**>  
 </**div**>  
 </**div**>  
 <**div class="summary-wrap"**>  
 <**div class="fl title"**>  
 <**i**>促　　销</**i**>  
 </**div**>  
 <**div class="fl fix-width"**>  
 <**i class="red-bg"**>加价购</**i**>  
 <**em class="t-gray"**>满999.00另加20.00元，或满1999.00另加30.00元，或满2999.00另加40.00元，即可在购物车换购热销商品</**em**>  
 </**div**>  
 </**div**>  
 </**div**>

#### 勾选默认规格

我们可以根据每个规格判断该规格是否在当前选中的Sku规格中，如果在，则返回true添加selected样式，否则返回false不添加selected样式

**Vue添加代码：**

**methods**:{  
 */\*\*  
 \* 判断某规格选项是否被用户选中  
 \** ***@param*** *specName 规格名称  
 \** ***@param*** *optionName 选项名称  
 \** ***@return*** *选中的样式 selected  
 \*/* isSelected: **function** (specName, optionName) {  
 **if**(**this**.**spec**[specName] == optionName){  
 **return "selected"**;  
 }  
 **return ""**;  
 }  
}

页面添加样式绑定，代码如下：

<**div class="clearfix choose"**>  
 <**div id="specification" class="summary-wrap clearfix"**>  
 <**dl th:each="spec : ${specificationList}"**>  
 <**dt**>  
 <**div class="fl title"**>  
 <**i th:text="${spec.key}"**></**i**>  
 </**div**>  
 </**dt**>  
  
 <**dd th:each="val : ${spec.value}"**>  
 <**a href="javascript:;" th:v-bind:class="|isSelected('${spec.key}','${val}')|"**>  
 <**i th:text="${val}"**></**i**>  
 <**span title="点击取消选择"**>**&nbsp;**</**span**>  
 </**a**>  
 </**dd**>  
 *<!--<dd v-for="coloritem in goodsdetail.color" :key="coloritem.index">  
 <a href="javascript:;">{{coloritem}}</a>  
 </dd>-->* </**dl**>  
 </**div**>

#### 用户选中的规格

用户可以修改想买的商品规格

**Vue实现：**

添加一个用户选项规格的方法

*/\*\*  
 \* 用户选择规格  
 \** ***@param*** *specName 规格名称  
 \** ***@param*** *optionName 选项名称  
 \*/*selectSpec: **function** (specName, optionName) {  
 *//this.specificationItems[specName] = optionName;* **this**.$set(**this**.**spec**, specName, optionName);  
}

页面点击事件绑定，代码如下：

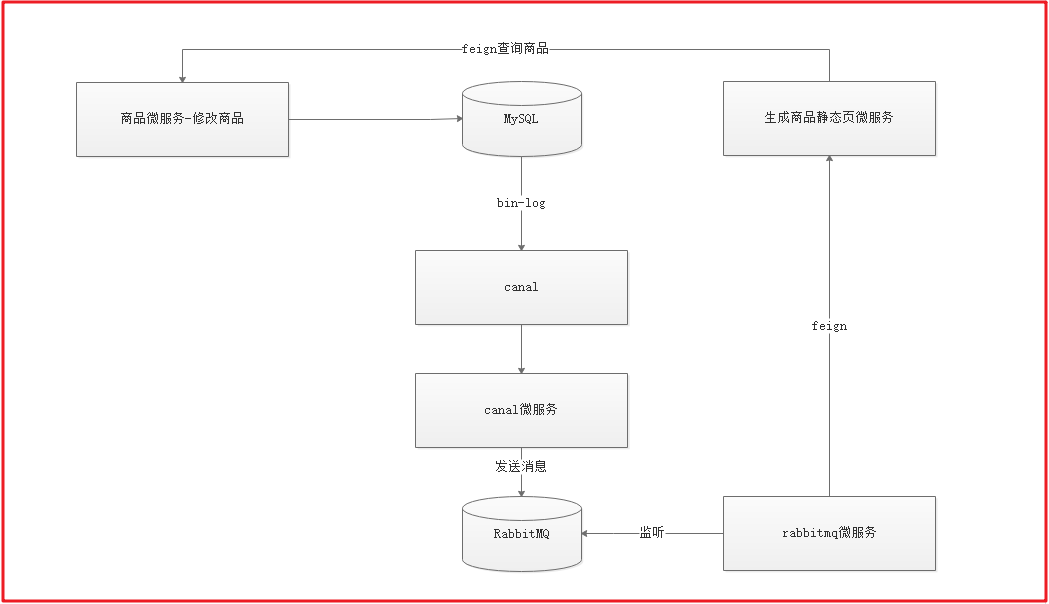
<**div id="specification" class="summary-wrap clearfix"**>  
 <**dl th:each="spec : ${specificationList}"**>  
 <**dt**>  
 <**div class="fl title"**>  
 <**i th:text="${spec.key}"**></**i**>  
 </**div**>  
 </**dt**>  
  
 <**dd th:each="val : ${spec.value}"**>  
 <**a href="javascript:;" th:@click="|selectSpec('${spec.key}','${val}')|" th:v-bind:class="|isSelected('${spec.key}','${val}')|"**>  
 <**i th:text="${val}"**></**i**>  
 <**span title="点击取消选择"**>**&nbsp;**</**span**>  
 </**a**>  
 </**dd**>  
 *<!--<dd v-for="coloritem in goodsdetail.color" :key="coloritem.index">  
 <a href="javascript:;">{{coloritem}}</a>  
 </dd>-->* </**dl**>  
</**div**>

#### 选择规格更新SKU

**修改Vue代码：**

*/\*\*  
 \* 用户选择规格  
 \** ***@param*** *specName 规格名称  
 \** ***@param*** *optionName 选项名称  
 \*/*selectSpec: **function** (specName, optionName) {  
 *//this.specificationItems[specName] = optionName;* **this**.$set(**this**.**spec**, specName, optionName);  
 *//选择规格后查找对应的SKU* **this**.searchSku();  
},  
*//匹配两个对象的内容是否一致*matchObject:**function**(map1,map2){  
 **for** (**var** k **in** map1) {  
 **if** (map1[k] != map2[k]) {  
 **return false**;  
 }  
 }  
 **for** (**var** k **in** map2) {  
 **if** (map2[k] != map1[k]) {  
 **return false**;  
 }  
 }  
 **return true**;  
},  
*//选择规格后，查询相应的sku*searchSku: **function** () {  
 **for** (**let** i = 0; i < **this**.**skuList**.**length**; i++) {  
 *//使用用户选中的规格信息与sku列表的规格信息对比* **if** (**this**.matchObject(***JSON***.parse(**this**.**skuList**[i].**spec**), **this**.**spec**)) {  
 *//更新sku* **this**.**sku** = **this**.**skuList**[i];  
 **return**;  
 }  
 }  
}

# RabbitMQ微服务搭建(学员作业)



从上面流程图中我们可以看出，我们需要使用canal实现数据库数据监控，Spu数据发生变化的时候，需要向RabbitMQ中发送一条消息，其他应用如果需要一句该消息实现指定功能，可以将功能写好，创建一个RabbitMQ微服务实现调用即可，这里RabbitMQ单独抽取出来的主要目的是让微服务只负责业务流程处理，不负责与其他微服务通信，多个微服务之间的异步通信在RabbitMQ微服务中完成。

## 需求分析

当商品微服务审核商品之后，应当发送消息，这里采用了Canal监控数据变化，数据变化后，此时需要该数据的可能由多个服务，我们可以将数据发送给RabbitMQ，多个微服务如若需要该数据，可以在RabbitMQ微服务中使用Feign调用即可。

## RabbitMQ微服务搭建

在changgou-service中创建changgou-service-rabbitmq，该工程主要集成RabbitMQ，实现微服务之间的异步通信。

在changgou-service-api中创建changgou-web-item-api，该工程中主要创建changgou-web-item的对外依赖抽取信息。

(1)Feign创建

在changgou-web-item-api中创建com.changgou.item.feign.PageFeign,代码如下：

@FeignClient(name="item")

@RequestMapping("/page")

**public** **interface** **PageFeign** {

/\*\*\*

\* 根据SpuID生成静态页

\* @param id

\* @return

\*/

@RequestMapping("/createHtml/{id}")

Result **createHtml**(@PathVariable(name="id") Long id);

}

(2)pom.xml依赖

修改changgou-service-rabbitmq工程的pom.xml，引入如下依赖：

<?xml version="1.0" encoding="UTF-8"?>

<**project** **xmlns**="http://maven.apache.org/POM/4.0.0"

**xmlns:xsi**="http://www.w3.org/2001/XMLSchema-instance"

**xsi:schemaLocation**="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<**parent**>

<**artifactId**>changgou-service</**artifactId**>

<**groupId**>com.changgou</**groupId**>

<**version**>1.0-SNAPSHOT</**version**>

</**parent**>

<**modelVersion**>4.0.0</**modelVersion**>

<**artifactId**>changgou-service-rabbitmq</**artifactId**>

<**dependencies**>

<**dependency**>

<**groupId**>org.springframework.boot</**groupId**>

<**artifactId**>spring-boot-starter-amqp</**artifactId**>

</**dependency**>

<!--依赖changgou-web-item-api-->

<**dependency**>

<**groupId**>com.changgou</**groupId**>

<**artifactId**>changgou-web-item-api</**artifactId**>

<**version**>1.0-SNAPSHOT</**version**>

</**dependency**>

</**dependencies**>

</**project**>

(3)application.yml配置

在changgou-service-rabbitmq中创建application.yml文件，代码如下：

server:

port: 18086

eureka:

client:

service-url:

defaultZone: http://127.0.0.1:7001/eureka

instance:

prefer-ip-address: true

feign:

hystrix:

enabled: true

spring:

application:

name: rabbitmq

main:

allow-bean-definition-overriding: true

rabbitmq:

host: 192.168.211.132

username: guest

password: guest

port: 5672

(4)创建启动类

在changgou-service-rabbitmq微服务中创建启动类，代码如下：

@SpringBootApplication(exclude={DataSourceAutoConfiguration.class})

@EnableEurekaClient

@EnableFeignClients

**public** **class** **RabbitMqApplication** {

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

SpringApplication.run(RabbitMqApplication.class,args);

}

}

## 微服务异步通信

### Canal监控数据

使用Canal监控tb\_spu数据变化，修改和删除的时候向RabbitMQ发送消息，其他地方也会用到该数据，所以我们可以发送Topic模式的消息。

(1)引入依赖

在canal微服务中引入如下依赖

<!--MQ依赖-->

<**dependency**>

<**groupId**>org.springframework.boot</**groupId**>

<**artifactId**>spring-boot-starter-amqp</**artifactId**>

</**dependency**>

(2)RabbitMQ配置

在application.yml中引入rabbitmq配置，代码如下：

#rabbitmq配置

rabbitmq:

host: 192.168.211.132

port: 5672

username: guest

password: guest

(3)消息封装

以后其他应用也会涉及到消息发送，我们可以把消息信息封装起来，消息信息主要包括执行的操作类型、消息的routekey、exechange等信息，在common工程总创建Message，代码如下：

**public** **class** **Message** **implements** **Serializable**{

//执行的操作 1：增加，2：修改,3：删除

**private** **int** code;

//数据

**private** Object content;

//发送的routkey

@JSONField(serialize = **false**)

**private** String routekey;

//交换机

@JSONField(serialize = **false**)

**private** String exechange;

**public** **Message**() {

}

**public** **Message**(**int** code, Object content) {

**this**.code = code;

**this**.content = content;

}

**public** **Message**(**int** code, Object content, String routekey, String exechange) {

**this**.code = code;

**this**.content = content;

**this**.routekey = routekey;

**this**.exechange = exechange;

}

**public** String **getRoutekey**() {

**return** routekey;

}

**public** **void** **setRoutekey**(String routekey) {

**this**.routekey = routekey;

}

**public** String **getExechange**() {

**return** exechange;

}

**public** **void** **setExechange**(String exechange) {

**this**.exechange = exechange;

}

**public** **int** **getCode**() {

**return** code;

}

**public** **void** **setCode**(**int** code) {

**this**.code = code;

}

**public** Object **getContent**() {

**return** content;

}

**public** **void** **setContent**(Object content) {

**this**.content = content;

}

}

(4)队列创建

在canal微服务中创建com.changgou.canal.mq.queue.TopicQueue实现创建队列以及交换机绑定的操作，代码如下：

@Configuration

**public** **class** **TopicQueue** {

**public** **static** **final** String TOPIC\_QUEUE\_SPU = "topic.queue.spu";

**public** **static** **final** String TOPIC\_EXCHANGE\_SPU = "topic.exchange.spu";

/\*\*

\* Topic模式 SPU变更队列

\* @return

\*/

@Bean

**public** Queue **topicQueueSpu**() {

**return** **new** **Queue**(TOPIC\_QUEUE\_SPU);

}

/\*\*\*

\* SPU队列交换机

\* @return

\*/

@Bean

**public** TopicExchange **topicSpuExchange**() {

**return** **new** **TopicExchange**(TOPIC\_EXCHANGE\_SPU);

}

/\*\*\*

\* 队列绑定交换机

\* @return

\*/

@Bean

**public** Binding **topicBinding1**() {

**return** BindingBuilder.bind(topicQueueSpu()).to(topicSpuExchange()).with(TOPIC\_QUEUE\_SPU).noargs();

}

}

(5)消息发送

创建一个消息发送对象com.changgou.canal.mq.send.TopicMessageSender，主要用于发送Topic消息，代码如下：

@Component

**public** **class** **TopicMessageSender** {

@Autowired

**private** RabbitTemplate rabbitTemplate;

/\*\*\*

\* Topic消息发送

\* @param message

\*/

**public void sendMessage(Message message){**

**rabbitTemplate.convertAndSend(message.getExechange(), message.getRoutekey(), JSON.toJSONString(message));**

**}**

}

同时在启动类中开启RabbitMQ，在CanalApplication上添加如下注解：

@EnableRabbit

在CanalDataEventListener中添加消息发送代码如下：

/\*\*\*

\* 规格、分类数据修改监听

\* 同步数据到Redis

\* @param eventType

\* @param rowData

\*/

@ListenPoint(destination = "example", schema = "changgou\_goods", table = {"tb\_spu"}, eventType = {CanalEntry.EventType.UPDATE,CanalEntry.EventType.DELETE})

**public** **void** **onEventCustomSpu**(CanalEntry.EventType eventType, CanalEntry.RowData rowData) {

//操作类型

**int** number = eventType.getNumber();

//操作的数据

String id = getColumn(rowData,"id");

//封装Message

Message message = **new** Message(number, id, TopicQueue.TOPIC\_QUEUE\_SPU,TopicQueue.TOPIC\_EXCHANGE\_SPU);

//发送消息

topicMessageSender.sendMessage(message);

}

其中getColumn方法如下：

/\*\*\*

\* 获取某个列的值

\* @param rowData

\* @param name

\* @return

\*/

**public** String **getColumn**(CanalEntry.RowData rowData , String name){

//操作后的数据

**for** (CanalEntry.Column column : rowData.getBeforeColumnsList()) {

String columnName = column.getName();

**if**(columnName.equalsIgnoreCase(name)){

**return** column.getValue();

}

}

//操作前的数据

**for** (CanalEntry.Column column : rowData.getBeforeColumnsList()) {

String columnName = column.getName();

**if**(columnName.equalsIgnoreCase(name)){

**return** column.getValue();

}

}

**return** **null**;

}

### 调用静态页生成

在RabbitMQ微服务中监听消息并调用changgou-web-item微服务来生成静态页，在changgou-service-rabbitmq微服务中创建com.changgou.rabbitmq.listener.item.HtmlGeneratListener类来监听消息，代码如下：

@Component

@RabbitListener(queues = "topic.queue.spu")

**public** **class** **HtmlGeneratListener** {

@Autowired

**private** PageFeign pageFeign;

/\*\*\*

\* 生成静态页/删除静态页

\* @param msg

\*/

@RabbitHandler

**public** **void** **getInfo**(String msg){

//将数据转成Message

Message message = JSON.parseObject(msg,Message.class);

**if**(message.getCode()==2){

//审核，生成静态页

pageFeign.createHtml(Long.parseLong(message.getContent().toString()));

}

}

}

这里只演示了生成静态页，删除静态页同学们完成。