

# 第7章 Thymeleaf实现静态页搜索页

## 学习目标

- Thymeleaf的介绍(模板引擎)
- Thymeleaf的入门

1 | 基本语法

- 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 | 1. 详情页静态化操作  
2 | 2. 填充基础数据 Spu、List<Sku>  
3 | 3. 规格切换

### 搜索页面渲染

1 | 1. 数据展示  
2 | 2. 搜索条件展示  
3 | 3. 实现条件搜索控制

## 1.Thymeleaf讲解(会用)

### 目标

- 掌握Thymeleaf的基本语法

### 路径

- Thymeleaf介绍
- Thymeleaf整合SpringBoot
- Thymeleaf基本语法

### 讲解

#### 1.1 Thymeleaf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。

- 1 模板：将一些重复内容写好，其中某些可能发生变化的内容，采用占位符方式动态加载（JSP）
- 2 模板引擎技术：可以基于写好的模板，动态给写好的模板加载数据。

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。

## 1.2 Springboot整合thymeleaf

使用springboot 来集成使用Thymeleaf可以大大减少单纯使用thymeleaf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springboot项目
- 添加thymeleaf的起步依赖
- 添加spring web的起步依赖
- 编写html 使用thymeleaf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

SpringMVC回顾:

- 1 1. 创建工程
- 2 2. 引入依赖包
- 3 3.web.xml-核心前端控制器、POST请求乱码过滤器
- 4 4.SpringMVC核心配置文件
  - 5 视图解析器 前缀(/WEB-INF/pages/)、后缀(.jsp)
  - 6 包扫描
  - 7 静态资源过滤
  - 8 注解驱动

### (1)创建工作

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。

#### pom.xml依赖

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4           xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
5           http://maven.apache.org/xsd/maven-4.0.0.xsd">
6     <modelVersion>4.0.0</modelVersion>
7
8     <groupId>com.itheima</groupId>
9     <artifactId>springboot-thymeleaf</artifactId>
10    <version>1.0-SNAPSHOT</version>
11
12    <parent>
```

```
12     <groupId>org.springframework.boot</groupId>
13     <artifactId>spring-boot-starter-parent</artifactId>
14     <version>2.1.4.RELEASE</version>
15 </parent>
16
17 <dependencies>
18     <!--web起步依赖-->
19     <dependency>
20         <groupId>org.springframework.boot</groupId>
21         <artifactId>spring-boot-starter-web</artifactId>
22     </dependency>
23
24     <!--thymeleaf配置-->
25     <dependency>
26         <groupId>org.springframework.boot</groupId>
27         <artifactId>spring-boot-starter-thymeleaf</artifactId>
28     </dependency>
29 </dependencies>
30 </project>
```

## (2)测试

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

```
1 @SpringBootApplication
2 public class ThymeleafApplication {
3
4     public static void main(String[] args) {
5         SpringApplication.run(ThymeleafApplication.class,args);
6     }
7 }
```

## (3)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：

```
1 @Controller
2 @RequestMapping("/test")
3 public class TestController {
4
5     /**
6      * 访问/test/hello 跳转到demo1页面
7      * @param model
8      * @return
9      */
10    @RequestMapping("/hello")
11    public String hello(Model model){
12        model.addAttribute("hello","hello welcome");
13        return "demo1";
14    }
15 }
```

#### (4)修改application.yml配置

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：

```
public class ThymeleafProperties {
    private static final Charset DEFAULT_ENCODING;
    public static final String DEFAULT_PREFIX = "classpath:/templates/";
    public static final String DEFAULT_SUFFIX = ".html";
    private boolean checkTemplate = true;
    private boolean checkTemplateLocation = true;
    private String prefix = "classpath:/templates/";
    private String suffix = ".html";
    private String mode = "HTML";
    private Charset encoding;
    private boolean cache;
    private Integer templateResolverOrder;
    private String[] viewNames;
    private String[] excludedViewNames;
    private boolean enableSpringElCompiler;
    private boolean renderHiddenMarkersBeforeCheckboxes;
    private boolean enabled;
    private final ThymeleafProperties.Servlet servlet;
    private final ThymeleafProperties.Reactive reactive;
```

#### (5)创建html

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

```
1 <!DOCTYPE html>
2 <html xmlns:th="http://www.thymeleaf.org">
3 <head>
4     <title>Thymeleaf的入门</title>
5     <meta http-equiv="Content-Type" content="text/html; charset=UTF-8"/>
6 </head>
7 <body>
8 <!--输出hello数据-->
9 <p th:text="${hello}"></p>
10 </body>
11 </html>
```

解释：

<html xmlns:th="http://www.thymeleaf.org"> :这句声明使用thymeleaf标签

<p th:text="\${hello}"></p> :这句使用 th:text="\${变量名}" 表示 使用thymeleaf获取文本数据，类似于EL表达式。

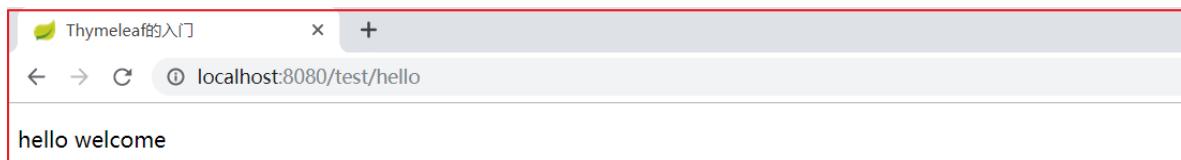
#### (6)关闭缓存

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。

```
1 spring:
2     thymeleaf:
3         cache: false
```

启动系统，并在浏览器访问

```
1 | http://localhost:8080/test/hello
```



## 1.3 Thymeleaf基本语法

### (1)文本输出

普通文本输出

```
1 | <p th:text="${description}"></p>
```

上面的输出会将数据全部以文本输出，无法显示html标签效果，效果如下：

```
<span>王五在黑马学习java</span>
```

th:utext 输出文本可以识别html标签：

```
1 | <p th:utext="${description}"></p>
```

效果如下：

```
王五在黑马学习java
```

### (2)th:each

对象遍历，功能类似jstl中的 <c:forEach> 标签。

创建com.itheima.model.User,代码如下：

```
1 public class User {
2     private Integer id;
3     private String name;
4     private String address;
5
6     public User() {
7     }
8     public User(Integer id, String name, String address) {
9         this.id = id;
10        this.name = name;
11        this.address = address;
12    }
13    //...get..set
14 }
```

## Controller添加数据

```
1 /**
2  * 访问/test/hello 跳转到demo1页面
3  * @param model
4  * @return
5 */
6 @RequestMapping("/hello")
7 public String hello(Model model){
8     model.addAttribute("hello","hello welcome");
9
10    //集合数据
11    List<User> users = new ArrayList<User>();
12    users.add(new User(1,"张三","深圳"));
13    users.add(new User(2,"李四","北京"));
14    users.add(new User(3,"王五","武汉"));
15    model.addAttribute("users",users);
16    return "demo1";
17 }
```

## 页面输出

```
1 <table>
2   <tr>
3     <td>下标</td>
4     <td>编号</td>
5     <td>姓名</td>
6     <td>住址</td>
7   </tr>
8   <tr th:each="user,userStat:${users}">
9     <td>
10       下标:<span th:text="${userStat.index}"></span>,
11     </td>
12     <td th:text="${user.id}"></td>
13     <td th:text="${user.name}"></td>
14     <td th:text="${user.address}"></td>
15   </tr>
16 </table>
```

## 测试效果



## (3)Date输出

### 后台添加日期

```
1 //日期  
2 model.addAttribute("now", new Date());
```

## 页面输出

```
1 <div>  
2     <span th:text="${#dates.format(now, 'yyyy-MM-dd hh:ss:mm')}"></span>  
3 </div>
```

## 测试效果



## (4)条件判断

th:if条件判断

后台添加年龄

```
1 //if条件  
2 model.addAttribute("age", 22);
```

## 页面输出

```
1 <div>  
2     <span th:if="${(age)>=18}">终于长大了! </span>  
3 </div>
```

## 测试效果



反向判断 th:unless，代码如下：

```
1 <p>th:unless 反向判断      age小于18岁的条件不成立的时候输出成年人</p>  
2 <div>  
3     <span th:unless="${age<18}">成年人</span>  
4 </div>
```

效果如下：

th:unless 反向判断

成年人

#### (4)Map输出

后台添加Map

```
1 //Map定义
2 Map<String, Object> dataMap = new HashMap<String, Object>();
3 dataMap.put("No", "123");
4 dataMap.put("address", "深圳");
5 model.addAttribute("dataMap", dataMap);
```

页面输出

```
1 <div th:each="map, mapStat:${dataMap}">
2     <div th:text="${map}"></div>
3     key:<span th:text="${mapStat.current.key}"></span><br/>
4     value:<span th:text="${mapStat.current.value}"></span><br/>
5     =====
6 </div>
```

测试效果



判断Map是否存在某个Key

```
1 <p>if判断Map中是否包含某个key</p>
2 <div th:if="#maps.containsKey(dataMap, 'address')">
3     <span th:text="${dataMap.address}"></span>
4 </div>
```

#### (5)字符处理

在后台存储一个name数据：

```
1 //字符判断和处理
2 model.addAttribute("name", "spec_小红");
```

以指定字符开始判断:

```
1 <p>判断是否以spec_开始, 如果是, 则输出name值</p>
2 <div th:if="#{strings.startsWith(name, 'spec_')}>
3   <span th:text="${name}"></span>
4 </div>
```

去掉字符串中指定字符串:

```
1 <p>将name中的spec_替换成空数据</p>
2 <div th:text="#{strings.replace(name, 'spec_', '')}"></div>
```

## (6)超链接处理

超链接使用 `th:href` 语法是 `th:href="@{url(name=xx,age=xx)}`, 案例如下:

在后台添加一个url参数

```
1 //添加一个地址
2 model.addAttribute("url", "/test/add");
```

在后台添加一个 `/test/add` 的跳转地址方法

```
1 /**
2  * 接收前端数据
3  * @return
4 */
5 @RequestMapping("/add")
6 public String add(String name, String address, Model model) {
7     System.out.println(name+" 住址是 "+address);
8     return "redirect:http://www.itheima.com";
9 }
```

前端跳转配置:

```
1 <p>跳转地址</p>
2 <a th:href="@{${url}(name='王五',address='深圳')}">跳转到/test/add方法</a>
```

## (7)图片

```
1 <p>图片显示</p>
2 
```

## (8)递增输出

输出1-10的数据

```
1 <p>输出1-10的数据</p>
2 <div th:each="i:${#numbers.sequence(1,10,1)}">
3   <span th:text="${i}"></span>
4 </div>
```

## 总结

# 2 搜索页面渲染

---

### 目标

- 商品搜索渲染

### 路径

- 搜索分析
- 搜索数据列表展示
- 搜索分页实现
- 搜索条件展示
- 动态添加搜索条件

### 讲解

#### 2.1 搜索析

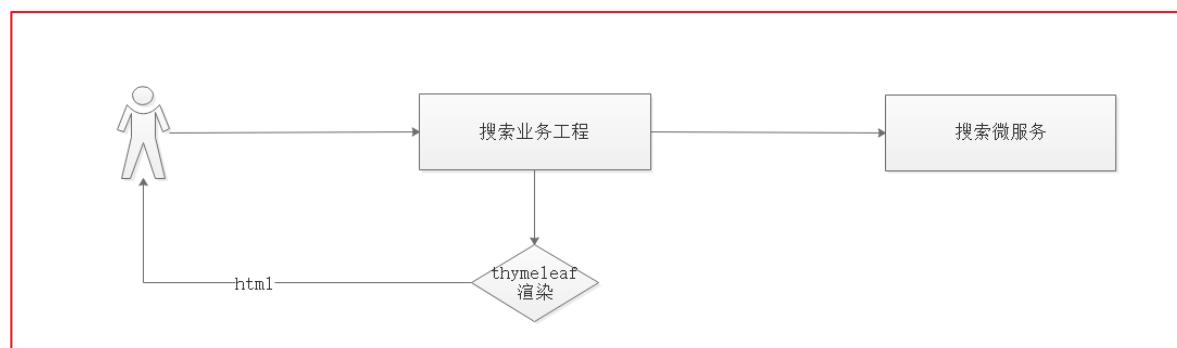
全部结果 / 智能手机 iPhone X 华为 X OPPO X

手机、数码、配件									
品牌	索尼 (SONY)	TCL	长虹 (CHAN...)	飞利浦 (PHIL...)	风行电视	ESR 亿色	ROCK	EXCO	BASEUS
多选 更多									
网络制式	GSM (移动/联通2G) 电信2G 电信3G 移动3G 联通3G 联通4G 电信3G 移动3G 联通3G 联通4G								
显示屏尺寸	4.0-4.9英寸 4.0-4.9英寸								
摄像头像素	1200万以上	800-1199万	1200-1599万	1600万以上	无摄像头				
价格	0-500元	500-1000元	1000-1500元	1500-2000元	2000-3000元	3000元以上			
更多筛选项	特点 系统 手机内存 单卡双卡 其他								
综合	销量 新品 评价 价格								
<p>¥6088.00 Apple 苹果 iPhone 6s (A1699) Apple 苹果 iPhone 6s (A1699) Apple 苹果...</p> <p>已有2000人评价</p> <p>¥6088.00 Apple 苹果 iPhone 6s (A1699) Apple 苹果 iPhone 6s (A1699) Apple 苹果...</p> <p>已有2000人评价</p> <p>¥6088.00 Apple 苹果 iPhone 6s (A1699) Apple 苹果 iPhone 6s (A1699) Apple 苹果...</p> <p>已有2000人评价</p> <p>¥6088.00 Apple 苹果 iPhone 6s (A1699) Apple 苹果 iPhone 6s (A1699) Apple 苹果...</p> <p>已有2000人评价</p>									

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

## 2.2 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。

### 2.2.1 搜索工程搭建

#### (1)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：

```

1 <dependencies>
2   <dependency>
3     <groupId>org.springframework.boot</groupId>
4     <artifactId>spring-boot-starter-thymeleaf</artifactId>
5   </dependency>
6   <dependency>
7     <groupId>org.springframework.boot</groupId>
8     <artifactId>spring-boot-starter-web</artifactId>
9   </dependency>
10  <dependency>
11    <groupId>org.springframework.cloud</groupId>
12    <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
13  </dependency>
14  <!--feign-->
15  <dependency>
16    <groupId>org.springframework.cloud</groupId>
17    <artifactId>spring-cloud-starter-openfeign</artifactId>
18  </dependency>
19  <!--amqp-->
20  <dependency>
21    <groupId>org.springframework.boot</groupId>
22    <artifactId>spring-boot-starter-amqp</artifactId>
23  </dependency>
24 </dependencies>

```

## (2)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:



记住将search.html中的相对路径全部改成绝对路径, 可以把search.html中路径配置的 ./ 全部换成 / 即可。这里如果不改的话, 页面显示图片和其他静态资源会报404。

## (3)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:

```
1  @FeignClient(name="search")
2  @RequestMapping("/search")
3  public interface SkuFeign {
4
5      /**
6       * 搜索
7       * @param searchMap
8       * @return
9       */
10      @GetMapping
11      Map search(@RequestParam(required = false) Map searchMap);
12 }
```

#### (4)changgou-web-search的pom.xml依赖

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4           xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
5 http://maven.apache.org/xsd/maven-4.0.0.xsd">
6   <parent>
7     <artifactId>changgou-web</artifactId>
8     <groupId>com.changgou</groupId>
9     <version>1.0-SNAPSHOT</version>
10    </parent>
11    <modelVersion>4.0.0</modelVersion>
12    <artifactId>changgou-web-search</artifactId>
13
14    <dependencies>
15      <!--search API依赖-->
16      <dependency>
17        <groupId>com.changgou</groupId>
18        <artifactId>changgou-service-search-api</artifactId>
19        <version>1.0-SNAPSHOT</version>
20      </dependency>
21    </dependencies>
22  </project>
```

### (5) 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:

```
1 @Controller
2 @RequestMapping(value = "/search")
3 public class SkuController {
4
5     @Autowired
6     private SkuFeign skuFeign;
7
8     /**
9      * 搜索
10     * @param searchMap
```

```
11     * @return
12     */
13     @GetMapping(value = "/list")
14     public String search(@RequestParam(required = false) Map searchMap,
15     Model model){
16         //调用changgou-service-search微服务
17         Map resultMap = skuFeign.search(searchMap);
18         model.addAttribute("result",resultMap);
19         return "search";
20     }
21 }
```

## (6)启动类创建

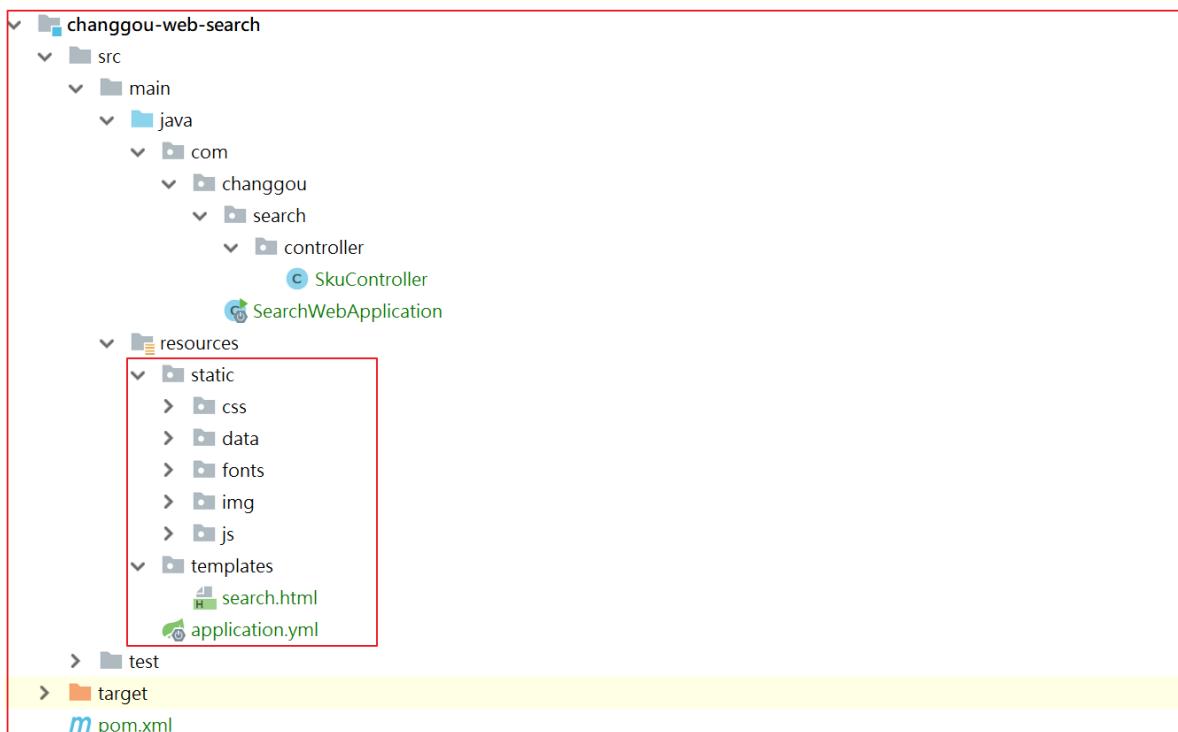
修改changgou-web-search,添加启动类com.changgou.SearchWebApplication, 代码如下:

```
1 @SpringBootApplication
2 @EnableEurekaClient
3 @EnableFeignClients(basePackages = "com.changgou.search.feign")
4 public class SearchwebApplication {
5     public static void main(String[] args) {
6         SpringApplication.run(SearchwebApplication.class,args);
7     }
8 }
```

## (7)application.yml配置文件

```
1 server:
2   port: 18086
3 eureka:
4   client:
5     service-url:
6       defaultZone: http://127.0.0.1:7001/eureka
7   instance:
8     prefer-ip-address: true
9 spring:
10   thymeleaf:
11     cache: false
12   application:
13     name: search-web
14   main:
15     allow-bean-definition-overriding: true
16 # 修改服务地址轮询策略, 默认是轮询, 配置之后变随机
17 ribbon:
18   ConnectTimeout: 10000 # 连接超时时间
19   ReadTimeout: 10000 # 数据读取超时时间
```

## (8)项目完整结构



在search.html的头部引入thymeleaf标签

```
1 | <html xmlns:th="http://www.thymeleaf.org">
```

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

## 2.2.2 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 1) 搜索的数据结果
- 2 2) 筛选出的数据搜索条件
- 3 3) 用户已经勾选的数据条件

## 2.2.3 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 1. 前端表单提交搜索的关键词
- 2 2. 后端根据关键词进行搜索
- 3 3. 将搜索条件存储到 `Model` 中
- 4 4. 页面循环迭代输出数据
- 5 5. 搜索表单回显搜索的关键词

### (1)后台搜索实现

修改 `SkuController` 的 `search` 方法，代码如下：

```
/*
 * 搜索
 */
@GetMapping(value = "/list")
public String search(@RequestParam(required = false) Map searchMap, Model model) {
    //调用搜索
    Map resultMap = skuFeign.search(searchMap);
    model.addAttribute("result", resultMap);

    //搜索条件存储用于页面回显
    model.addAttribute("searchMap", searchMap);
    return "search";
}
```

代码如下：

```
1 /**
2  * 搜索
3 */
4 @GetMapping(value = "/list")
5 public String search(@RequestParam(required = false) Map searchMap, Model
model){
6     //调用搜索
7     Map resultMap = skuFeign.search(searchMap);
8     model.addAttribute("result", resultMap);
9
10    //搜索条件存储用于页面回显
11    model.addAttribute("searchMap", searchMap);
12    return "search";
13 }
```

## (2)页面搜索实现

修改search.html

```
<form action="/search/list" class="sui-form form-inline">
    <!--searchAutoComplete--> 搜索地址
    <div class="input-append">
        <input type="text" id="autocomplete" name="keywords" 提交的name=keywords
            th:value="${#maps.containsKey(searchMap, 'keywords')) ? ${searchMap.keywords} : ''}" />关键词回显,先判断searchMap中是否存在keywords, 如果存在就回显
        <button class="sui-btn btn-xlarge btn-danger" type="submit">搜索</button>
    </div>
</form>
```

注意：搜索按钮为submit提交。

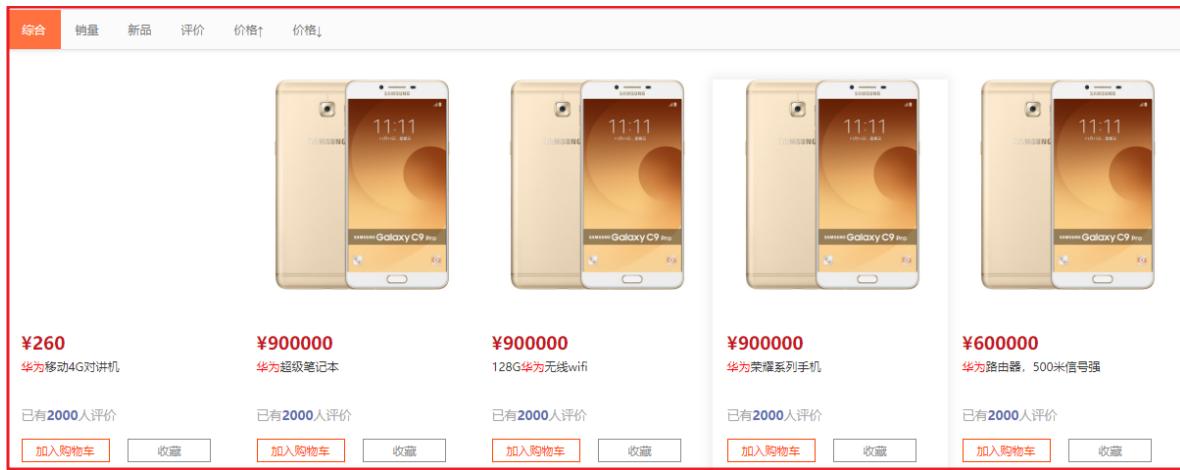
## (3)页面结果输出

修改search.html，代码如下：

```
<div class="goods-list">
    <ul class="yui3-g">
        <li class="yui3-u-1-5" th:each="sku, skuStat: ${result.rows}">循环result.rows
            <div class="list-wrap">
                <div class="p-img">
                    <a href="item.html" target="_blank"></a>
                </div>
                <div class="price">
                    <strong>
                        <em>¥</em>
                        <i th:text="${sku.price}"></i>
                    </strong>
                </div>
                <div class="attr">输出标题, 最多只输出45个字符, th:utext 可以识别html标签
                    <a th:utext="#{strings.abbreviate(sku.name, 45)}" target="_blank"></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>
    </ul>
</div>
```

## (4)测试

搜索 华为 关键字,效果如下：



## 2.3 搜索条件回显

### 2.3.1 搜索结果回显分析

分类	手机	小家电	五金家装	户外工具	平板电视	笔记本	手机配件
品牌	HTC	OPPO	华为	小米	<input type="checkbox"/> 多选 <input type="checkbox"/> 更多		
网络	联通2G	联通3G	联通4G	移动4G	双卡		
手机屏幕尺寸	5寸	5.5寸					
机身内存	16G	32G	128G				
存储	16G	32G	64G				
像素	300万像素	800万像素					
颜色	红	绿	蓝	紫	白	黑	
测试	实施	学习	实施	测试	显示	s11	
价格	0-500元	500-1000元	1000-1500元	1500-2000元	2000-3000元	3000元以上	

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.specList`

### 2.3.2 搜索结果回显实现

修改search.html的条件显示部分，代码如下：

```

<!--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, categoryStat: ${result.categoryList}">
        <a th:text="${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, brandStat: ${result.brandList}">
          <a th:text="${brand}"></a>
        </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>
<div class="type-wrap" th:each="spec, specStat: ${result.specList}" th:unless="${#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="op, opStat: ${spec.value}">
        <a th:text="${op}"></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> price条件为空的时候，则显示金额信息
  <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>

```

```

<div class="type-wrap">
  <div class="fl key">更多筛选项</div>
  <div class="fl value">
    <ul class="type-list">
      <li>
        <a>特点</a>
      </li>
      <li>
        <a>系统</a>
      </li>
      <li>
        <a>手机内存 </a>
      </li>
      <li>
        <a>单卡双卡</a>
      </li>
      <li>
        <a>其他</a>
      </li>
    </ul>
  </div>
</div>

```

```
</div>
<div class="f1 ext">
</div>
</div>
</div>
```

上图代码如下：

```
1 <!--selector-->
2 <div class="clearfix selector">
3     <div class="type-wrap" th:if="#maps.containsKey(result,
4         'categoryList')}">
5         <div class="f1 key">分类</div>
6         <div class="f1 value">
7             <span th:each="category,categoryStat:${result.categoryList}">
8                 <a th:text="${category}"></a>&ampnbsp&ampnbsp
9                 </span>
10            </div>
11            <div class="f1 ext"></div>
12        </div>
13        <div class="type-wrap logo" th:if="#maps.containsKey(result,
14            'brandList')}">
15            <div class="f1 key brand">品牌</div>
16            <div class="value logos">
17                <ul class="logo-list">
18                    <li th:each="brand,brandStat:${result.brandList}">
19                        <a th:text="${brand}"></a>
20                    </li>
21                </ul>
22            </div>
23            <div class="ext">
24                <a href="javascript:void(0);" class="sui-btn">多选</a>
25                <a href="javascript:void(0);">更多</a>
26            </div>
27        </div>
28        <div class="type-wrap" th:each="spec,specStat:${result.specList}"
29             th:unless="#maps.containsKey(searchMap, 'spec_'+spec.key)}">
30            <div class="f1 key" th:text="${spec.key}"></div>
31            <div class="f1 value">
32                <ul class="type-list">
33                    <li th:each="op,opStat:${spec.value}">
34                        <a th:text="${op}"></a>
35                    </li>
36                </ul>
37            </div>
38            <div class="f1 ext"></div>
39        </div>
40        <div class="type-wrap" th:unless="#maps.containsKey(searchMap,
41            'price')}">
42            <div class="f1 key">价格</div>
43            <div class="f1 value">
44                <ul class="type-list">
45                    <li>
46                        <a>0-500元</a>
47                    </li>
48                    <li>
49                        <a>500-1000元</a>
50                    </li>
51                </ul>
52            </div>
53        </div>
54    </div>
55
```

```

48         <li>
49             <a>1000-1500元</a>
50         </li>
51         <li>
52             <a>1500-2000元</a>
53         </li>
54         <li>
55             <a>2000-3000元</a>
56         </li>
57         <li>
58             <a>3000元以上</a>
59         </li>
60     </ul>
61 </div>
62 <div class="f1 ext">
63 </div>
64 </div>
65 <div class="type-wrap">
66     <div class="f1 key">更多筛选项</div>
67     <div class="f1 value">
68         <ul class="type-list">
69             <li>
70                 <a>特点</a>
71             </li>
72             <li>
73                 <a>系统</a>
74             </li>
75             <li>
76                 <a>手机内存 </a>
77             </li>
78             <li>
79                 <a>单卡双卡</a>
80             </li>
81             <li>
82                 <a>其他</a>
83             </li>
84         </ul>
85     </div>
86     <div class="f1 ext">
87     </div>
88 </div>
89 </div>

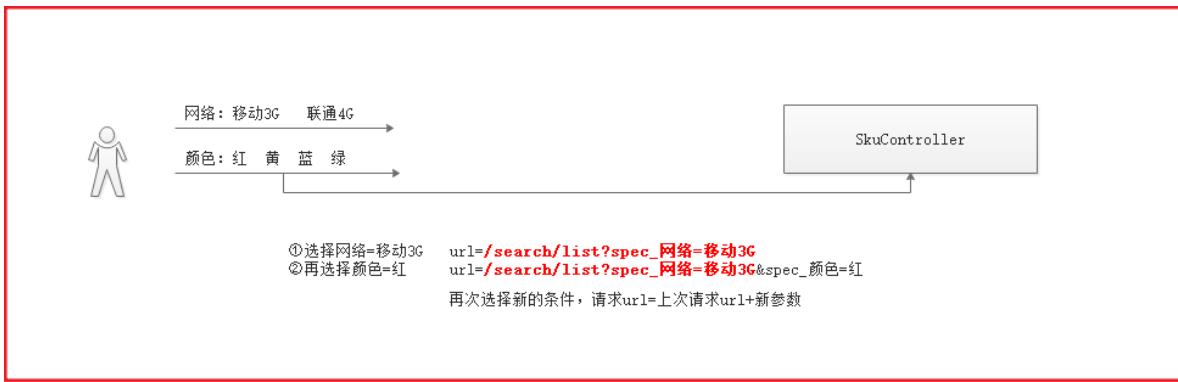
```

解释：

1	<code>th:unless:</code> 条件不满足时，才显示
2	<code>#{maps.containsKey(result, 'brandList')}</code> :map中包含某个key

## 2.4 条件搜索实现

### 2.4.1 搜索实现分析



## 2.4.2 搜索实现

(1)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：

```
/*
 * 搜索
 */
@GetMapping(value = "/list")
public String search(@RequestParam(required = false) Map searchMap, Model model) {
    //调用搜索
    Map resultMap = skuFeign.search(searchMap);
    model.addAttribute(s: "result", resultMap);

    //搜索条件存储用于页面回显
    model.addAttribute(s: "searchMap", searchMap);

    //获取url地址
    String url = url(searchMap);
    model.addAttribute(s: "url", url);
    return "search";
}

/**
 * url组装
 */
public String url(Map<String, String> searchMap) {
    //URL
    String url = "/search/list";

    //如果输入了对应的条件，则将对应的条件拼接到url后面
    if(searchMap!=null && searchMap.size()>0) {
        url+="?";
        for (Map.Entry<String, String> entry : searchMap.entrySet()) {
            //获取key, 也就是参数的名字
            String key = entry.getKey();
            //将key拼接到url后面
            url+=key+"="+entry.getValue()+"&";
        }
        //去掉最后一个&
        url=url.substring(0, url.length()-1);
    }
    return url;
}
```

## (2)页面搜索对接

```

<!--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, categoryStat:$result.categoryList">
        <a th:href="@{${url} (category=${category})}" th:text="${category}"></a>&ampnbsp&ampnbsp
      </span> 上次请求的URL上添加分类参数
    </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, brandStat:$result.brandList">
          <a th:href="@{${url} (brand=${brand})}" th:text="${brand}"></a>
        </li> 上次请求的URL上添加品牌参数
      </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, specStat:$result.specList" th:unless="#maps.containsKey(searchMap, 'spec' + spec.key)">
    <div class="fl key" th:text="${spec.key}"></div>
    <div class="fl value">
      <ul class="type-list">
        <li th:each="op, opStat:$spec.value">
          <a th:href="@{${url} ('spec' + ${spec.key} = ${op})}" th:text="${op}"></a>
        </li> 上次请求的URL上添加规格参数
      </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="@{${url} (price='0-500元')}">0-500元</a>
        </li>
        <li>
          <a th:href="@{${url} (price='500-1000元')}">500-1000元</a>
        </li>
        <li>
          <a th:href="@{${url} (price='1000-1500元')}">1000-1500元</a>
        </li>
        <li>
          <a th:href="@{${url} (price='1500-2000元')}">1500-2000元</a>
        </li>
        <li>
          <a th:href="@{${url} (price='2000-3000元')}">2000-3000元</a>
        </li>
        <li>
          <a th:href="@{${url} (price='3000元以上')}">3000元以上</a>
        </li>
      </ul>
    </div>
    <div class="fl ext"></div>
  </div>
  <div class="type-wrap">
    <div class="fl key">更多筛选项</div>
    <div class="fl value">
      <ul class="type-list">
        <li>
          <a>特点</a>
        </li>
        <li>
          <a>系统</a>
        </li>
        <li>
          <a>手机内存</a>
        </li>
        <li>
          <a>单卡双卡</a>
        </li>
        <li>
          <a>其他</a>
        </li>
      </ul>
    </div>
  </div>


```

```

</ul>
</div>
<div class="fl ext">
</div>
</div>
</div>

```

th:href 这里是超链接的语法，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。

## 2.5 移除搜索条件

### 2.5.1 搜索条件移除分析



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。

### 2.5.2 搜索条件移除实现

#### (1) 条件显示

修改 `search.html`，代码如下：

```

<ul class="f1 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>x</a>
    </li>
</ul>
<ul class="f1 sui-tag">
    <li th:if="${#maps.containsKey(searchMap, 'brand')}" class="with-x">
        <span th:text="${searchMap.brand}"></span><a>x</a>
    </li>
    <li th:if="${#maps.containsKey(searchMap, 'price')}" class="with-x">
        <span th:text="${searchMap.price}"></span><a>x</a>
    </li>
    <li th:if="${strings.startsWith(sm.key, 'spec_')}" th:each="sm : ${searchMap}" class="with-x">
        <span th:text="${#strings.replace(sm.key, 'spec_', '')}"></span><span th:text="${sm.value}"></span><a>x</a>
    </li>
</ul>

```

解释：

- 1  `${#strings.startsWith(sm.key, 'spec_')} : 表示以 spec_ 开始的 key`
- 2  `${#strings.replace(sm.key, 'spec_', '')} : 表示将 sm.key 中的 spec_ 替换成空`

## (2) 移除搜索条件

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="@{${#strings.replace(url, 'category=' + searchMap.category, '')}}"></a>
        移除当前分类条件
    </li>
</ul>
<ul class="fl sui-tag">
    <li th:if="${#maps.containsKey(searchMap, 'brand')}" class="with-x">
        <span th:text="${searchMap.brand}"></span><a th:href="@{${#strings.replace(url, 'brand=' + searchMap.brand, '')}}"></a>
        移除当前品牌条件
    </li>
    <li th:if="${#maps.containsKey(searchMap, 'price')}" class="with-x">
        <span th:text="${searchMap.price}"></span><a th:href="@{${#strings.replace(url, 'price=' + searchMap.price, '')}}"></a>
        移除价格搜索条件
    </li>
    <li th:if="${#strings.startsWith(sm.key, 'spec_')}" th:each="sm : ${searchMap}" class="with-x">
        <span th:text="${#strings.replace(sm.key, 'spec_', '')}"></span><span th:text="${sm.value}"></span>
        <a th:href="@{${#strings.replace(url, sm.key+'=' + sm.value, '')}}"></a>
        移除当前规格条件
    </li>
</ul>
```

## 2.6 排序

### 2.6.1 排序分析

```
//8. 排序
String sortRule=searchMap.get("sortRule");// 排序规则 ASC DESC
String sortField=searchMap.get("sortField");//排序字段 price
//是否排序
if(!StringUtils.isEmpty(sortField)){
    nativeSearchQueryBuilder.withSort(SortBuilders.fieldSort(sortField).order(SortOrder.valueOf(sortRule)));
}
```

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。

手机屏幕尺寸 5寸 5.5寸

网络	联通3G	移动4G	联通2G	联通4G	电信4G	
颜色	红	紫	白	绿		
机身内存	16G	128G	32G			
存储	16G	32G	64G			
像素	300万像素	800万像素				
价格	0-500元	500-1000元	1000-1500元	1500-2000元	2000-3000元	3000元以上
更多筛选项	特点	系统	手机内存	单卡双卡	其他	

这些赛道需要排序参数

比如原来的url为 /search/list?sortRule=ASC&sortField=time  
这里需要切换排序的时候，需要重新追加排序方式和排序的域  
追加后的地址 /search/list?sortRule=ASC&sortField=price &sortRule=ASC&sortField=price

综合 销量 新品 评价 价格↑ 价格↑

¥999 华为 胖 5寸 联通3G+移动4G 绿 实 施 32G 64G 800万像素 已有2000人评价 加入购物车 收藏

¥999 华为 胖 5.5寸 联通3G+移动4G 白 实 施 128G 16G 300万像素 已有2000人评价 加入购物车 收藏

¥999 小米 缤纷浪漫 5.5寸 联通3G 紫 实 施 128G 64G 300万像素 已有2000人评价 加入购物车 收藏

¥999 守护宝 奢浦绢 5.5寸 电信4G 白 实 施 128G 32G 800万像素 已有2000人评价 加入购物车 收藏

¥999 手机配件 便携 5寸 联通2G 白 实 施 16G 64G 300万像素 已有2000人评价 加入购物车 收藏

分页需要排序参数

«上一页 1 2 3 4 5 ... 下一页» 共10页 到第页 确定

## 2.6.2 排序处理

### (1)排序URL处理

在今天课件中有一个工具包 `urlutils.java` 可以把它拷贝到 `changgou-common` 工程中，使用该工具类可以去掉URL地址中的指定参数。

修改 `changgou-web-search` 的 `skuController` 类中的搜索方法，添加排序路径的处理，代码如下：

```
/***
 * 搜索
 */
@GetMapping(value = "/list")
public String search(@RequestParam(required = false) Map searchMap, Model model) {
    //调用搜索
    Map resultMap = skuFeign.search(searchMap);
    model.addAttribute(s: "result", resultMap);

    //搜索条件存储用于页面回显
    model.addAttribute(s: "searchMap", searchMap);

    //获取url地址
    String url = url(searchMap);
    model.addAttribute(s: "url", url);
    //去掉排序域和排序类型域的地址
    model.addAttribute(s: "sorturl", UrlUtils.replateUrlParameter(url, ...names: "sortRule", "sortField"));
    return "search";
}
```

### (2)前端排序实现

修改 `search.html`，实现排序，代码如下：

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

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。

## 2.7 分页

真实的分页应该像百度那样，如下图：



### (1) 分页工具类定义

在今天的课件中有一个 `Page.java` 分页工具，可以用该分页工具处理翻页数据，我们将该工具拷贝到 `changgou-common` 工程中，可以直接使用，该工具类的代码如下：

```
1 public class Page <T> implements Serializable{
2
3     // 页数（第几页）
4     private long currentpage;
5
6     // 查询数据库里面对应的数据有多少条
7     private long total;// 从数据库查处的总记录数
8
9     // 每页查5条
10    private int size;
11
12    // 下页
13    private int next;
14
15    private List<T> list;
16
17    // 最后一页
18    private int last;
19
20    private int lpage;
21
22    private int rpage;
23
24    // 从哪条开始查
25    private long start;
26
27    // 全局偏移量
28    public int offsize = 2;
29
30    public Page() {
31        super();
32    }
33
34    *****
35    *
```

```
36     * @param currentpage
37     * @param total
38     * @param pagesize
39     */
40     public void setCurrentpage(long currentpage, long total, long pagesize)
41     {
42         //可以整除的情况下
43         long pagecount = total/pagesize;
44
45         //如果整除表示正好分N页, 如果不能整除在N页的基础上+1页
46         int totalPages = (int) (total%pagesize==0? total/pagesize :
47             (total/pagesize)+1);
48
49         //总页数
50         this.last = totalPages;
51
52         //判断当前页是否越界, 如果越界, 我们就查最后一页
53         if(currentpage>totalPages){
54             this.currentpage = totalPages;
55         }else{
56             this.currentpage=currentpage;
57         }
58
59         //计算start
60         this.start = (this.currentpage-1)*pagesize;
61     }
62
63     //上一页
64     public long getUpper() {
65         return currentpage>1? currentpage-1: currentpage;
66     }
67
68     //总共有多少页, 即末页
69     public void setLast(int last) {
70         this.last = (int) (total%size==0? total/size : (total/size)+1);
71     }
72
73     /**
74      * 带有偏移量设置的分页
75      * @param total
76      * @param currentpage
77      * @param pagesize
78      * @param offsize
79      */
80     public Page(long total,int currentpage,int pagesize,int offsize) {
81         this.offsize = offsize;
82         initPage(total, currentpage, pagesize);
83     }
84
85     /**
86      *
87      * @param total    总记录数
88      * @param currentpage   当前页
89      * @param pagesize   每页显示多少条
90      */
91     public Page(long total,int currentpage,int pagesize) {
92         initPage(total,currentpage,pagesize);
93     }
```

```
92
93     *****
94     * 初始化分页
95     * @param total
96     * @param currentpage
97     * @param pagesize
98     */
99     public void initPage(long total,int currentpage,int pagesize){
100         //总记录数
101         this.total = total;
102         //每页显示多少条
103         this.size=pagesize;
104
105         //计算当前页和数据库查询起始值以及总页数
106         setCurrentpage(currentpage, total, pagesize);
107
108         //分页计算
109         int leftcount =this.offsize,      //需要向上一页执行多少次
110             rightcount =this.offsize;
111
112         //起点页
113         this.lpage =currentpage;
114         //结束页
115         this.rpage =currentpage;
116
117         //2点判断
118         this.lpage = currentpage-leftcount;           //正常情况下的起点
119         this.rpage = currentpage+rightcount;          //正常情况下的终点
120
121         //页差=总页数和结束页的差
122         int topdiv = this.last-rpage;                  //判断是否大于最大页数
123
124     ****
125     * 起点页
126     * 1、页差<0 起点页=起点页+页差值
127     * 2、页差>=0 起点和终点判断
128     */
129     this.lpage=topdiv<0? this.lpage+topdiv:this.lpage;
130
131     ****
132     * 结束页
133     * 1、起点页<=0 结束页=|起点页|+1
134     * 2、起点页>0 结束页
135     */
136     this.rpage=this.lpage<=0? this.rpage+(this.lpage*-1)+1:
137     this.rpage;
138
139     ****
140     * 当起点页<=0 让起点页为第一页
141     * 否则不管
142     */
143     this.lpage=this.lpage<=0? 1:this.lpage;
144
145     ****
146     * 如果结束页>总页数 结束页=总页数
147     * 否则不管
148     */
149     this.rpage=this.rpage>last? this.last:this.rpage;
```

```
149     }
150
151     public long getNext() {
152         return currentpage<last? currentpage+1: last;
153     }
154
155     public void setNext(int next) {
156         this.next = next;
157     }
158
159     public long getCurrentpage() {
160         return currentpage;
161     }
162
163     public long getTotal() {
164         return total;
165     }
166
167     public void setTotal(long total) {
168         this.total = total;
169     }
170
171     public long getSize() {
172         return size;
173     }
174
175     public void setSize(int size) {
176         this.size = size;
177     }
178
179     public long getLast() {
180         return last;
181     }
182
183     public long getLpage() {
184         return lpage;
185     }
186
187     public void setLpage(int lpage) {
188         this.lpage = lpage;
189     }
190
191     public long getRpage() {
192         return rpage;
193     }
194
195     public void setRpage(int rpage) {
196         this.rpage = rpage;
197     }
198
199     public long getStart() {
200         return start;
201     }
202
203     public void setStart(long start) {
204         this.start = start;
205     }
206
```

```

207     public void setCurrentpage(Long currentpage) {
208         this.currentpage = currentpage;
209     }
210
211     /**
212      * @return the list
213      */
214     public List<T> getList() {
215         return list;
216     }
217
218     /**
219      * @param list the list to set
220      */
221     public void setList(List<T> list) {
222         this.list = list;
223     }
224 }
```

## (2)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分页数据：

```

/**
 * 搜索数据
 * @param searchMap
 * @return
 */
@Override
public Map search(Map<String, String> searchMap) {
    //1. 条件构建
    NativeSearchQueryBuilder builder = buildBasicQuery(searchMap);

    //2. 搜索列表
    Map resultMap = searchList(builder);

    //分页数据保存
    resultMap.put("pageNum", builder.build().getPageable().getpageNumber() + 1);
    resultMap.put("pageSize", builder.build().getPageable().getPageSize());添加该方法
    //分组查询分类列表、品牌列表、规格列表
    resultMap.putAll(groupList(builder, searchMap));
    return resultMap;
}
```

上述代码如下：

```

1 //分页数据保存
2 resultMap.put("pageNum", builder.build().getPageable().getpageNumber() + 1);
3 resultMap.put("pageSize", builder.build().getPageable().getPageSize());
```

修改SkuController,实现分页信息封装，代码如下：

```

/**
 * 搜索
 */
@GetMapping(value = "/list")
public String search(@RequestParam(required = false) Map searchMap, Model model) {
    //调用搜索
    Map resultMap = skuFeign.search(searchMap);
    model.addAttribute("result", resultMap);

    //搜索条件存储用于页面回显
    model.addAttribute("searchMap", searchMap);

    //获取url地址
    String url = url(searchMap);
    model.addAttribute("url", url);
    //去掉排序域和排序类型域的地址
    model.addAttribute("sortUrl", UrlUtils.replateUrlParameter(url, ...names: "sortRule", "sortField"));

    //分页数据获取
    Page<SkuInfo> page = new Page<SkuInfo>(
        Long.parseLong(resultMap.get("totalElements").toString()), //总记录数
        Integer.parseInt(resultMap.get("pageNum").toString()), //每页显示条数
        Integer.parseInt(resultMap.get("pageSize").toString()) //当前页
    );
    model.addAttribute("page", page);
    return "search";
}

```

### (3) 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="@{${url}+'#list' (pageNum=page.upper)}" th:text="上一页"/>
            </li>
            <li th:each="i:${#numbers.sequence(page.lpage, page.rpage)}" th:class="${i}==$page.currentpage?'active':''">
                <a th:href="@{${url}+'#list' (pageNum=${i})}" th:text="${i}"></a>
            </li>
            <li class="next">
                <a th:href="@{${url}+'#list' (pageNum=${page.next})}" th:text="下一页"/>
            </li>
        </ul>
        <div>
            共<i th:text="${page.last}"></i>页&ampnbsp
        </div>
    </div>
</div>

```

注意：每次如果搜条件发生变化都要从第1页查询，而点击下一页的时候，分页数据在页面给出，不需要在后台拼接的url中给出，每次url都需要去掉pageNum参数，代码如下：

```

//获取url地址
String url = url(searchMap);
model.addAttribute("url", UrlUtils.replateUrlParameter(url, ...names: "pageNum"));
//去掉排序域和排序类型域的地址
model.addAttribute("sortUrl", UrlUtils.replateUrlParameter(url, ...names: "sortRule", "sortField", "pageNum"));

```

## 总结

### 3.畅购商品详情页

#### 目标

- 商品详情页静态化

#### 路径

- 搭建详情页静态化工程
- 生成静态页
- 静态页数据填充
- 静态页数据切换

#### 讲解

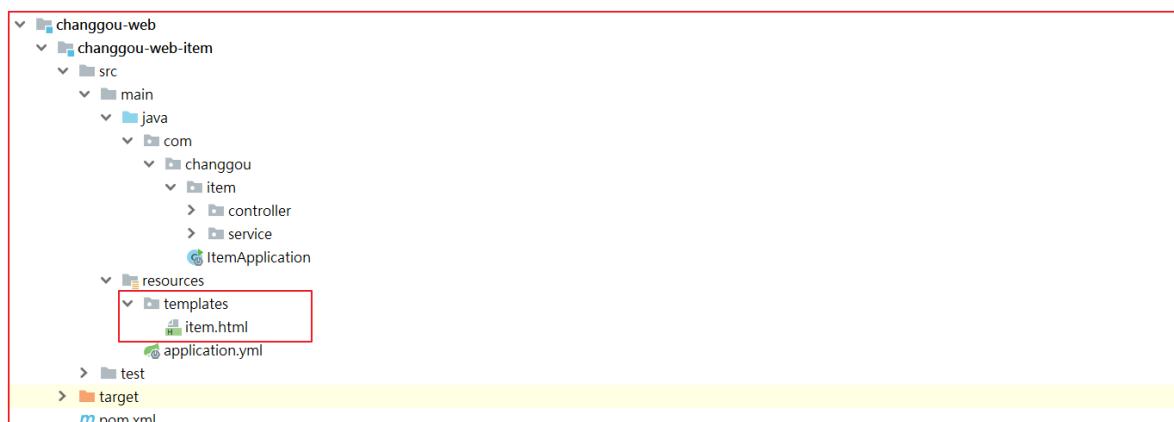
### 3.1 商品静态化微服务创建

#### 3.1.1 需求分析

该微服务只用于生成商品静态页，不做其他事情。

#### 3.1.2 搭建项目

(1) 在changgou-web下创建一个名称为changgou-web-item的模块，并且将item.html静态页拷贝到工程的templates目录下，如图：



(2) changgou-web-item中添加起步依赖，如下

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0"
3   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
5     http://maven.apache.org/xsd/maven-4.0.0.xsd">
6   <parent>
7     <artifactId>changgou-web</artifactId>
8     <groupId>com.changgou</groupId>
9     <version>1.0-SNAPSHOT</version>
10    </parent>
11    <modelVersion>4.0.0</modelVersion>
```

```
11 <artifactId>changgou-web-item</artifactId>
12 <description>
13     商品详情静态页生成微服务
14 </description>
15
16 <dependencies>
17     <!--api 模块-->
18     <dependency>
19         <groupId>com.changgou</groupId>
20         <artifactId>changgou-service-goods-api</artifactId>
21         <version>1.0-SNAPSHOT</version>
22     </dependency>
23 </dependencies>
24 </project>
```

### (3) 修改application.yml的配置

```
1 server:
2     port: 18087
3 eureka:
4     client:
5         service-url:
6             defaultZone: http://127.0.0.1:7001/eureka
7     instance:
8         prefer-ip-address: true
9 spring:
10    thymeleaf:
11        cache: false
12    application:
13        name: item
14    main:
15        allow-bean-definition-overriding: true
16    # 生成静态页的位置
17    pagepath: D:/items
```

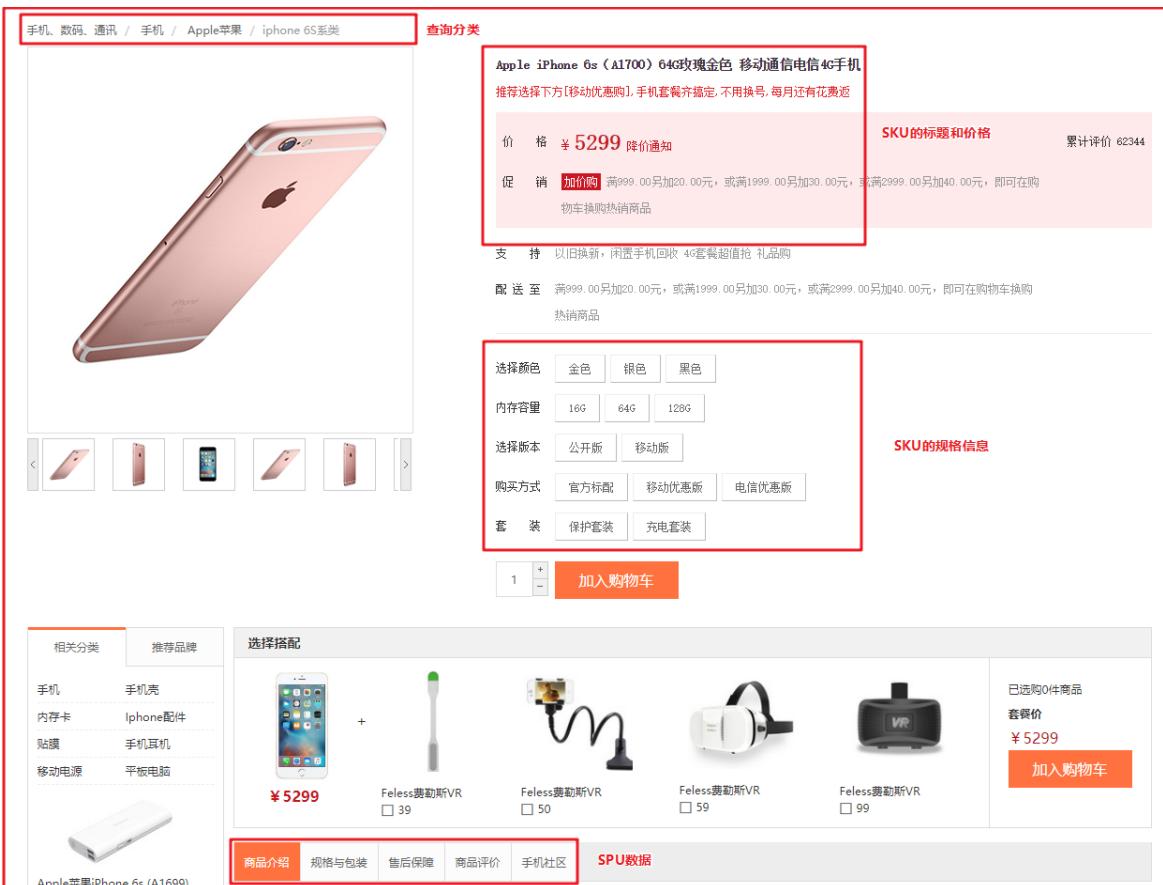
### (4) 创建系统启动类

```
1 @SpringBootApplication
2 @EnableEurekaClient
3 @EnableFeignClients(basePackages = "com.changgou.goods.feign")
4 public class ItemApplication {
5
6     public static void main(String[] args) {
7         SpringApplication.run(ItemApplication.class,args);
8     }
9 }
```

## 3.2 生成静态页

### 3.2.1 需求分析

页面发送请求，传递要生成的静态页的商品的SpuID.后台controller接收请求，调用thymeleaf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。

### 3.2.2 静态页生成

(1)在 changgou-web-item 中创建 com.changgou.item.service.PageService 接口，并在接口中添加生成静态页的方法，代码如下：

```
1 public interface PageService {  
2  
3     /**  
4      * 根据SPUID生成静态页  
5      * @param id : SpuId  
6      */  
7     void createPageHtml(String id);  
8 }
```

(2)在 changgou-web-item 中创建 com.changgou.item.service.impl.PageServiceImpl 接口，并在接口中添加生成静态页的方法，代码如下：

```
1 @Service  
2 public class PageServiceImpl implements PageService {  
3  
4     //静态文件所存储的位置  
5     @Value("${pagepath}")
```

```

6  private String pagepath;
7
8  //模板引擎对象
9  @Autowired
10 private TemplateEngine templateEngine;
11
12 /**
13  * 根据SPUID生成静态页
14  * @param id : SpuId
15  */
16 @Override
17 public void createPageHtml(String id) {
18     try {
19         //创建上下文对象
20         Context context = new Context();
21         //数据模型，用于存储页面需要填充的数据
22         Map<String, Object> dataModel = new HashMap<String, Object>();
23         context.setVariables(dataModel);
24
25         //生成的静态页的位置
26         File dest = new File(pagepath,id+".html");
27         //生成静态页
28         PrintWriter printwriter = new PrintWriter(dest,"UTF-8");
29         /**
30          * 1: 模板名字
31          * 2: 上下文对象
32          * 3: 文件输出对象
33          */
34         templateEngine.process("item",context,printwriter);
35     } catch (Exception e) {
36         e.printStackTrace();
37     }
38 }
39 }
```

(3)在 changgou-web-item 中创建 com.changgou.item.controller.PageController，代码如下：

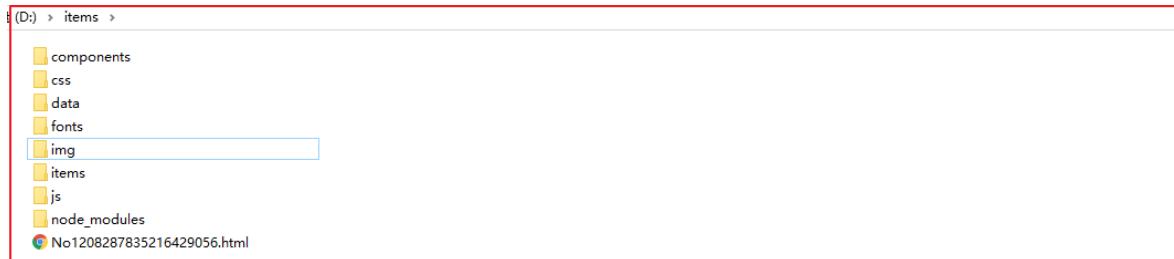
```

1  @RestController
2  @RequestMapping(value = "/page")
3  public class PageController {
4
5      @Autowired
6      private PageService pageService;
7
8      /**
9       * 创建静态页
10      */
11     @GetMapping(value = "/createHtml/{id}")
12     public Result createHtml(@PathVariable(value = "id")String id){
13         //生成静态页
14         pageService.createPageHtml(id);
15         return new Result(true, StatusCode.OK, "生成成功！");
16     }
17 }
```

启动服务测试生成静态页：

```
<http://localhost:18087/page/createHtml/No1208287835216429056>
```

在D盘的items目录下会生成静态文件，我们可以把静态文件所需的样式一起拷贝到该目录中来，让页面正常显示。



页面效果如下：

The screenshot shows a web browser displaying a product page for an iPhone 6s. The page has a header with the logo '畅购 CHANG GOU' and a search bar. Below the header, there are navigation links for '全部商品分类', '服装城', '美妆馆', '青橙超市', '全球购', '闪购', '团购', '有趣', and '秒杀'. The main content area shows a large image of a rose gold iPhone 6s. To the right of the image, the product name is 'Apple iPhone 6s (A1700) 64G玫瑰金色 移动通信电信4G手机'. Below the name, there is a red note: '推荐选择下方[移动优惠购]手机套餐齐搞定,不用换号,每月还有花费返'. The price is listed as '5299 降价通知'. There are sections for '促销' (including '加价购'), '支持' (such as '以旧换新'), and '配送至' (with options for '满999.00另加20.00元' etc.). Below these are dropdown menus for '选择颜色' (金色, 银色, 黑色), '内存容量' (16G, 64G, 128G), '选择版本' (公开版, 移动版), '购买方式' (官方标配, 移动优惠版, 电信优惠版), and '套装' (保护套装, 充电套装). At the bottom right is a '加入购物车' button with a quantity selector set to 1.

### 3.2.3 Feign创建

我们需要调用 `changgou-service-goods` 微服务获取商品详情页数据，这里会用到feign调用，我们分别要获取分类数据、Spu数据、Sku数据。

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：

```
1 @FeignClient(name="goods")
2 @RequestMapping(value = "/category")
3 public interface CategoryFeign {
4
5     /**
6      * 获取分类的对象信息
7      * @param id
8      * @return
9      */
10    @GetMapping("/{id}")
11    Result<Category> findById(@PathVariable(name = "id") Integer id);
12 }
```

在changgou-service-goods-api,添加SkuFeign,并添加根据SpuID查询Sku集合，代码如下：

```
1 /**
2  * 根据条件搜索
3  * @param sku
4  * @return
5  */
6 @PostMapping(value = "/search" )
7 Result<List<Sku>> findList(@RequestBody(required = false) Sku sku);
```

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

```
1 @FeignClient(name="goods")
2 @RequestMapping(value = "/spu")
3 public interface SpuFeign {
4
5     /**
6      * 根据SpuID查询Spu信息
7      * @param id
8      * @return
9      */
10    @GetMapping("/{id}")
11    Result<Spu> findById(@PathVariable(name = "id") String id);
12 }
```

### 3.2.4 静态页数据填充

实现类：

```
@Service
```

```
public class PageServiceImpl implements PageService {
```

```
//静态文件所存储的位置
```

```
@Value("${pagepath}")
```

```
private String pagepath;
```

```
//模板引擎对象
```

```
@Autowired
```

```
private TemplateEngine templateEngine;
```

```
@Autowired
```

```
private SpuFeign spuFeign;
```

```
@Autowired
```

```
private CategoryFeign categoryFeign;
```

```
@Autowired
```

```
private SkuFeign skuFeign;
```

```
/**
```

```
* 加载静态页所需的数据
```

```
*/
```

```
public Map<String, Object> buildDataModel(String spuid) {
```

```
//创建一个Map用于存储静态页所有数据
```

```
Map<String, Object> dataMap = new HashMap<String, Object>();
```

```
//查询Spu
```

```
Result<Spu> spuResult = spuFeign.findById(spuid);
```

```
Spu spu = spuResult.getData();
```

```
//List<Sku>查询
```

```
Sku sku = new Sku();
```

```
sku.setSpuId(spuid);
```

```
Result<List<Sku>> skuResult = skuFeign.findList(sku);
```

```
List<Sku> skuList = skuResult.getData();
```

```
//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());
```

```
//商品图片集合
```

```
dataMap.put("imageList", spu.getImages().split(regex: ", "));
```

```
//Spu存储
```

```
dataMap.put("spu", spu);
```

```
//List<Sku>存储
```

```
dataMap.put("skuList", skuList);
```

```
return dataMap;
```

```
}
```

## 详情页数据加载

```
/**
```

```
* 根据SPUID生成静态页
```

```
* @param id : SpuId
```

```
*/
```

```
@Override
```

```
public void createPageHtml(String id) {
```

```
try {
```

```
//创建上下文对象
```

```
Context context = new Context();
```

```
//数据模型，用于存储页面需要填充的数据
```

```
Map<String, Object> dataModel = buildDataModel(id);
```

```
context.setVariables(dataModel);
```

```
//生成的静态页的位置
```

```
File dest = new File(pagepath, child: id+".html");
```

```
//生成静态页
```

```
PrintWriter printWriter = new PrintWriter(dest, csn: "UTF-8");
```

```
/**
```

```
* 1: 模板名字
```

```
* 2: 上下文对象
```

```
* 3: 文件输出对象
```

```
*/
```

```
    ...
    templateEngine.process(template: "item", context, printWriter);
} catch (Exception e) {
    e.printStackTrace();
}
}
}
```

上图代码如下：

```
1 @Service
2 public class PageServiceImpl implements PageService {
3
4     //静态文件所存储的位置
5     @Value("${pagepath}")
6     private String pagepath;
7
8     //模板引擎对象
9     @Autowired
10    private TemplateEngine templateEngine;
11
12    @Autowired
13    private SpuFeign spuFeign;
14
15    @Autowired
16    private CategoryFeign categoryFeign;
17
18    @Autowired
19    private SkuFeign skuFeign;
20
21    /**
22     * 加载静态页所需的数据
23     */
24    public Map<String, Object> buildDataModel(String spuid){
25        //创建一个Map用于存储静态页所有数据
26        Map<String, Object> dataMap = new HashMap<String, Object>();
27
28        //查询Spu
29        Result<Spu> spuResult = spuFeign.findById(spuid);
30        Spu spu = spuResult.getData();
31
32        //List<Sku>查询
33        Sku sku = new Sku();
34        sku.setSpuId(spuid);
35        Result<List<Sku>> skuResult = skuFeign.findList(sku);
36        List<Sku> skuList = skuResult.getData();
37
38        //获取分类信息
39
40        dataMap.put("category1", categoryFeign.findById(spu.getCategory1Id()).getData());
41
42        dataMap.put("category2", categoryFeign.findById(spu.getCategory2Id()).getData());
43
44        dataMap.put("category3", categoryFeign.findById(spu.getCategory3Id()).getData());
45
46    }
47}
```

```

43     //商品图片集合
44     dataMap.put("imageList",spu.getImages().split(","));
45     //Spu存储
46     dataMap.put("spu",spu);
47     //List<Sku>存储
48     dataMap.put("skuList",skuList);
49     //specList
50     dataMap.put("specList",
51     JSON.parseObject(spu.getSpecItems(),Map.class));
52     return dataMap;
53 }
54 /**
55 * 根据SPUID生成静态页
56 * @param id : SpuId
57 */
58 @Override
59 public void createPageHtml(String id) {
60     try {
61         //创建上下文对象
62         Context context = new Context();
63         //数据模型，用于存储页面需要填充的数据
64         Map<String, Object> dataModel = buildDataModel(id);
65         context.setVariables(dataModel);
66
67         //生成的静态页的位置
68         File dest = new File(pagepath,id+".html");
69         //生成静态页
70         PrintWriter printwriter = new PrintWriter(dest,"UTF-8");
71         /**
72          * 1: 模板名字
73          * 2: 上下文对象
74          * 3: 文件输出对象
75          */
76         templateEngine.process("item",context,printwriter);
77     } catch (Exception e) {
78         e.printStackTrace();
79     }
80 }
81 }

```

### 3.2.5 模板填充

首先将页面的html换成 `<html xmlns:th="http://www.thymeleaf.org">`

(1)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>

```

## (2)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" 大图>
            <span class="jqzoom"></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="img:$imageList"></li>
                </ul>
            </div>
            <a class="next">&gt;;</a>
        </div>
    </div>
</div>

```

## (3)规格输出

```

<div id="specification" class="summary-wrap clearfix">
    <dl th:each="spec:$specList">
        <dt>
            <div class="fl title">
                <i th:text="${spec.key}"></i>
            </div>
        </dt>
        <dd th:each="opt:$spec.value">
            <a href="javascript:;" class="selected">
                <em th:text="${opt}"></em>
                <span title="点击取消选择">&ampnbsp</span>
            </a>
        </dd>
    </dl>
</div>

```

生成静态页后效果如下：



#### (4)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：

```
<!--  
    和Thymeleaf一起使用，这里需要按照如下方式写  
-->  
<script th:inline="javascript">  
    var app = new Vue({  
        el: '#app',  
        data: {  
            skuList: [[${skuList}]], //sku的集合  
            sku: {}, //用户选中的sku  
            spec: {} //用户选中的sku的spec  
        },  
        created:function () {  
            //默认选中第1个sku,采用深克隆方式  
            this.sku = JSON.parse(JSON.stringify(this.skuList[0]));  
            //默认用第1个sku的规格  
            this.spec = JSON.parse(this.skuList[0].spec);  
        }  
    });  
</script>
```

页面显示默认的Sku信息

```

<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>612188</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>

```

### 选中当前的Sku

```

<dd th:each="opt:${spec.value}">
  <a href="javascript:;" th:v-bind:class="spec['${spec.key}']==${opt}?'selected':''">
    <em th:text="${opt}"></em>          选中当前的sku
    <span title="点击取消选择">&ampnbsp</span>
  </a>
</dd>

```

### (5)记录选中的Sku

在当前Spu的所有Sku中spec值是唯一的，我们可以根据spec来判断用户选中的是哪个Sku，我们可以在Vue中添加代码来实现，代码如下：

```

<!--
    和Thymeleaf一起使用，这里需要按照如下方式写
-->
<script th:inline="javascript">
    var app = new Vue({
        el:'#app',
        data:{
            skuList:[${skuList}], //sku的集合
            sku:{}, //用户选中的sku
            spec:{} //用户选中的sku的spec
        },
        methods:{
            //记录选中的规格数据
            selectSpec:function (specName,specValue) {
                //将spec替换成当前选中的规格
                this.$set(this.spec,specName,specValue);

                //从所有的skuList中循环匹配
                for(var i=0;i<this.skuList.length;i++) {
                    //获取当前的sku
                    //如果当前sku的spec和用户点击选中的spec相同，则表明用户选中的是当前商品
                    if(this.matchObject(JSON.parse(this.skuList[i].spec),this.spec)) {
                        //当前匹配到的sku就是用户要找的sku
                        this.sku=JSON.parse(JSON.stringify(this.skuList[i]));
                        return;
                    }
                }
                this.sku.name='---该商品已下架---';
                this.sku.price=0;
                this.sku.id=0;
            },
            //匹配2个json是否是同一个json
            matchObject:function (map1,map2) {
                for(var key in map1) {
                    //获取map1当前key的值
                    let mv1 = map1[key];
                    //获取map2当前key的值
                    let mv2 = map2[key];
                    //如果mv1!=mv2，则表示当前map1和map2一定不是相同的对象
                    if(mv1!=mv2) {
                        return false;
                    }
                }
                return true;
            }
        },
        created:function () {
            //默认选中第1个sku，采用深克隆方式
            this.sku = JSON.parse(JSON.stringify(this.skuList[0]));
            //默认用第1个sku的规格
            this.spec = JSON.parse(this.skuList[0].spec);
        }
    });
</script>

```

### 记录用户选中的商品

添加规格点击事件

```

<dd th:each="opt:${spec.value}">
    <a href="javascript:" th:@click="|selectSpec('${spec.key}', '${opt}')|"
        th:v-bind:class="|spec['${spec.key}']==${opt}? 'selected' : ''|"
        <em th:text="${opt}"></em>
        <span title="点击取消选择">&ampnbsp</span>
    </a>
</dd>

```

页面效果如下：



### 3.2.6 添加数量

添加一个js方法用于添加购买数量，代码如下：

```

data: {
    skuList: [[${skuList}]], //所有的Sku集合
    sku: {}, //记录用户选中的Sku
    spec: {}, //用于记录选中的Sku的spec
    num: 1 //加入购物车的商品数量
},
methods: {
    //添加购物车
    addNum: function (count) {
        this.num+=count;
        if(this.num<=0) {
            this.num=1;
        }
    },
}

```

调用：

```
<div class="f1 title">
  <div class="control-group">
    <div class="controls">
      <input autocomplete="off" type="text" v-model="num" minnum="1" class="itxt" />
      <a href="javascript:void(0)" @click="addNum(1)" class="increment plus">+</a>
      <a href="javascript:void(0)" @click="addNum(-1)" class="increment mins">-</a>
    </div>
  </div>
</div>
```

### 2.3.7 加入购物车

点击加入购物车的时候，我们这里暂时只提示加入购物车的信息即可，等到了后面再实现购物车的统一操作。

添加购物车的JS方法：

```
//加入购物车
addCart:function () {
  alert('您购买的商品信息：' +this.sku.name+'，价格是：'+this.sku.price+'，唯一标识符是：' +this.sku.id)
},
```

页面调用：

```
<div class="f1">
  <ul class="btn-choose unstyled">
    <li>
      <a href="javascript:void(0)" @click="addCart()" class="sui-btn btn-danger addshopcar">加入购物车</a>
    </li>
  </ul>
</div>
```

## 总结