尚品汇商城

版本：V 2.0

**商品详情页**

# 一 业务介绍

商品详情页，简单说就是以购物者的角度展现一个sku的详情信息。

这个页面不同于传统的crud的详情页，使用者并不是管理员，需要对信息进行查删改查，取而代之的是点击购买、放入购物车、切换颜色等等。

另外一个特点就是该页面的高访问量，虽然只是一个查询操作，但是由于频繁的访问所以我们必须对其性能进行最大程度的优化。

# 二 难点分析

1 光从功能角度上来说，并没有太多难点，唯一实现起来麻烦的就是用户对于不同销售属性的切换操作。



2 从性能角度来看，需要最大程度的提升页面的访问速度。

|  |
| --- |
| 难点：  1.skuInfo 表中一些基本信息。  2.销售属性展示：应该从spu\_sale\_attr,spu\_sale\_attr\_value  3.通过选择销售属性，来切换商品，实际上：通过选择销售属性展示不同的商品。 |

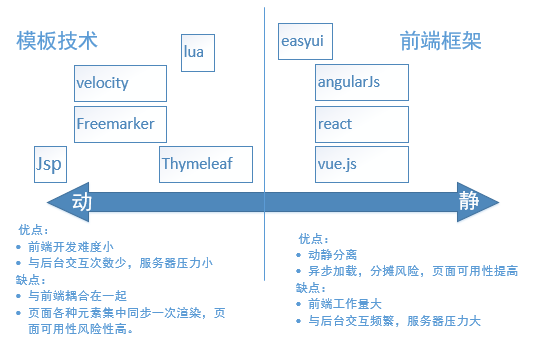
# 三 Thymeleaf

## 1 模板技术

把页面中的静态数据替换成从后台数据库中的数据。这种操作用jsp就可以实现。但是Spring boot 的架构不推荐使用Jsp，而且支持也不好，所以如果你是用springboot的话，一般使用Freemarker或者Thymeleaf。

而官方也是推荐使用**Thymeleaf**。

关于与前端有关的技术的比较



## 2 Thymeleaf 简介

Thymeleaf的主要目标是提供一个优雅和高度可维护的创建模板的方式。为了实现这一点，它建立在自然模板的概念上，将其逻辑注入到模板文件中，不会影响模板被用作设计原型。这改善了设计的沟通，弥合了设计和开发团队之间的差距。

比Jsp和Freemarker的优势，一般的模板技术都会在页面加各种表达式、标签甚至是java代码，而这些都必须要经过后台服务器的渲染才能打开。

但如果前端开发人员做页面调整，双击打开某个jsp或者ftl来查看效果，基本上是打不开的。

那么Thymeleaf的优势就出来了，因为Thymeleaf没有使用自定义的标签或语法，所有的模板语言都是扩展了标准H5标签的属性

比如 给div 添加一个值<div> hello </div>

|  |
| --- |
| <**div th:text="${item.skuName} "**></div> **th:text ：表示div显示的文本**  **${item.skuName} 表示在后台会有一个作用域将数据存储起来** |

它的效果和Jsp中的

|  |
| --- |
| <div>**${item.skuName}**</div> |

渲染后效果一样，但是如果你直接用浏览器打开页面文件，H5会把th:text这种不认识的属性忽略掉。效果就和<div></div> 没有区别，所以对于前端调页面影响更小。以上只是举了一个例子，如果是循环、分支的判断效果更明显。

## 3 快速入门：

项目创建，依赖模块web，Thymeleaf.模板。

### 3.1 所有头文件

就想Jsp的<%@Page %>一样 ，Thymeleaf的也要引入标签规范。不加这个虽然不影响程序运行，但是你的idea会认不出标签，不方便开发。

|  |
| --- |
| <**html xmlns="http://www.w3.org/1999/xhtml" xmlns:th="http://www.thymeleaf.org"**> |

### 3.2 取出请求域中的值，即取得request.attribute中的值，Model同样可以使用

|  |
| --- |
| <p th:text="${hello}" >打底值</p>或[[${hello}]]  除了input标签以后 输入后台的数据值 th:text=”${hello}”  <input th:value="${hello}" /> |

### 3.3 循环

|  |
| --- |
| <table border="1">  <tr th:each="skuImage:${skuInfo.skuImageList}">  <td th:text="${skuImage.id}">  </td>  <td th:text="${skuImage.imgName}">  </td>  </tr> </table> |
| *<!--测试循环-->* <**table**>  *<!-- <c:foreach items="${arrayList}" var = "arry">-->* <**tr th:each="arry,stat: ${arrayList}"**>  <**td th:text="${arry}"**></**td**>  <**td th:text="${stat.index}"**></**td**>  </**tr**> </**table**> |

### 3.4 判断

**Thymeleaf：判断的时候，数字类型，字符串类型，单引号可以加，也可以不加。**

|  |
| --- |
| 三元运算：  <td th:text="(${skuImage.id}=='10')?'是10':'不是10'"> </td>  If 判断 <td th:if="${skuImage.id}=='10'" >123456</td> |

### 3.5 取session中的属性

|  |
| --- |
| <div th:text="${session.userName}"> </div> |
| 如果在页面默认没有写级别的话，默认走的request${skuImage.id } |

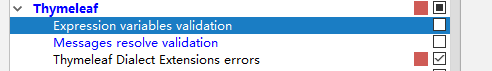
### 3.6 引用内嵌页

|  |
| --- |
| <div th:include="itemInner"/> |

如果idea编辑器质疑你页面中的元素是否存在



通常不准确，可以去掉验证，在settings->editor->Inspections中关掉验证。



### 3.7 th:utext :解析样式

Controller:

|  |
| --- |
| request.setAttribute(**"gname"**,**"<span style=\"color:green\">宝强</span>"**); |

Th:utext:识别html中的标签

|  |
| --- |
| <**p th:utext="${gname}"**>color</**p**> |

# 四 商品详情业务需求分析

## 1 详情渲染介绍



商品详情所需构建的数据如下：

1. sku基本信息
2. Sku图片信息
3. Sku分类信息
4. Sku销售属性相关信息
5. Sku价格信息（平台可以单独修改价格，sku后续会放入缓存，为了回去最新价格，所以单独获取）

...

## 2 详情模块规划

模块规划思路：

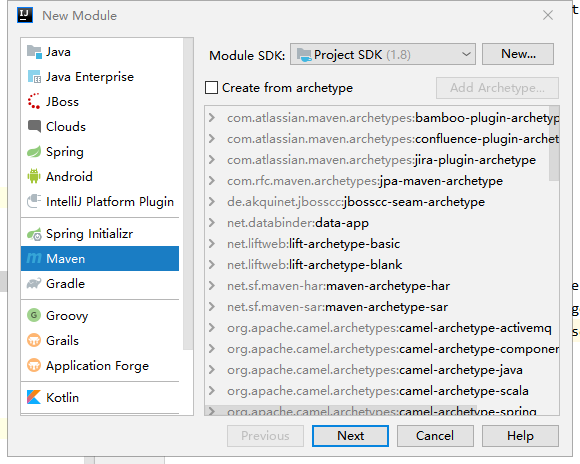
1. service-item微服务模块封装详情页面所需数据接口；
2. service-item通过feign client调用其他微服务数据接口进行数据汇总；
3. Pc端前台页面通过web-item调用service-item数据接口渲染页面；
4. service-item可以为ps端、H5、安卓与ios等前端应用提供数据接口，web-item为ps端页面渲染形式
5. service-item获取商品信息需要调用service-product服务sku信息等；
6. 由于service各微服务可能会相互调用，调用方式都是通过feign client调用，所以我们把feign client api接口单独封装出来，需要时直接引用feign client api模块接口即可，即需创建service-client父模块，管理各service微服务feign client api接口

# 五 商品详情实现

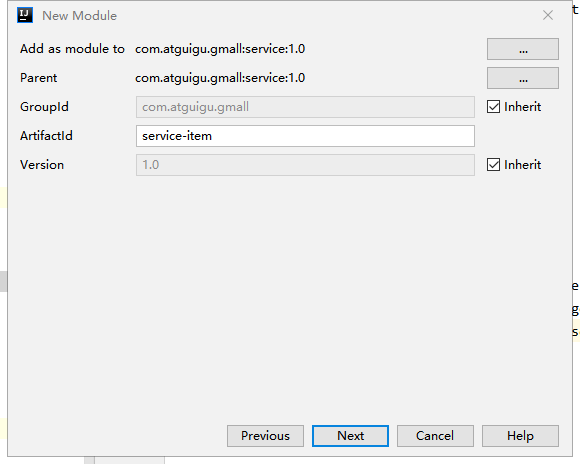
## 1 搭建service-item

### 1.1 构建模块

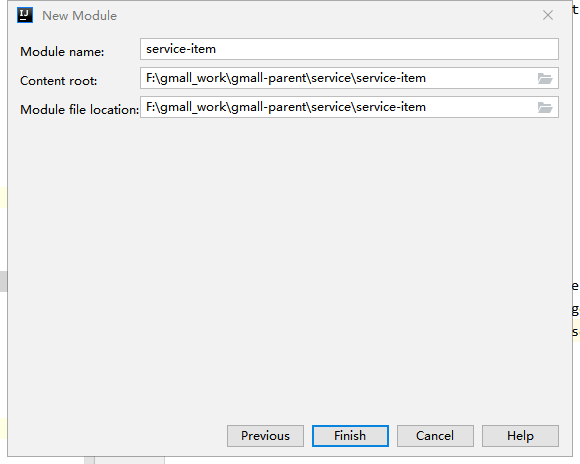
点击service，选择New–>Module,操作如下



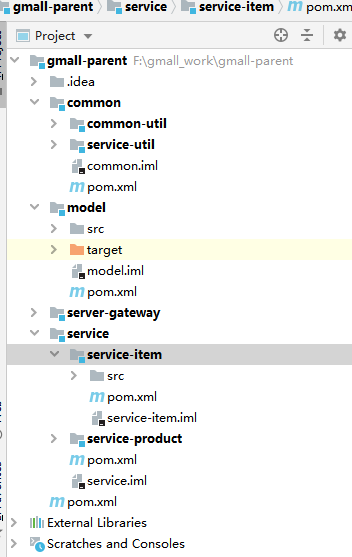
下一步



下一步



完成，结构如下



### 1.2 修改配置

修改配置pom.xml

|  |
| --- |
| *<?***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**>  <**parent**>  <**groupId**>com.atguigu.gmall</**groupId**>  <**artifactId**>service</**artifactId**>  <**version**>1.0</**version**>  </**parent**>   <**artifactId**>service-item</**artifactId**>  <**version**>1.0</**version**>   <**packaging**>jar</**packaging**>  <**name**>service-item</**name**>  <**description**>service-item</**description**>   <**dependencies**>  <**dependency**>  <**groupId**>com.atguigu.gmall</**groupId**>  <**artifactId**>service-product-client</**artifactId**>  <**version**>1.0</**version**>  </**dependency**> </**dependencies**>   <**build**>  <**finalName**>service-item</**finalName**>  <**plugins**>  <**plugin**>  <**groupId**>org.springframework.boot</**groupId**>  <**artifactId**>spring-boot-maven-plugin</**artifactId**>  </**plugin**>  </**plugins**>  </**build**>  </**project**> |

添加配置文件application.yml

|  |
| --- |
| **spring**:  **application**:  **name**: service-item  **profiles**:  **active**: dev  **cloud**:  **nacos**:  **discovery**:  **server-addr**: 127.0.0.1:8848 |

添加配置文件application-dev.yml

|  |
| --- |
| **server**:  **port**: 8202 **spring**:  **cloud**:  **sentinel**:  **transport**:  **dashboard**: 127.0.0.1:8080  **jackson**:  **date-format**: yyyy-MM-dd HH:mm:ss  **time-zone**: GMT+8  **redis**:  **host**: localhost  **port**: 6379  **database**: 0  **timeout**: 1800000  **password**:  **lettuce**:  **pool**:  **max-active**: 20 *#最大连接数* **max-wait**: -1 *#最大阻塞等待时间(负数表示没限制)* **max-idle**: 5 *#最大空闲* **min-idle**: 0 *#最小空闲* **feign**:  **sentinel**:  **enabled**: true  **client**:  **config**:  **default**:  **readTimeout**: 3000  **connectTimeout**: 1000 |

添加启动类

|  |
| --- |
| **package** com.atguigu.gmall.item;   @SpringBootApplication(exclude = DataSourceAutoConfiguration.**class**)*//取消数据源自动配置* @ComponentScan({**"com.atguigu.gmall"**}) @EnableDiscoveryClient @EnableFeignClients(basePackages= {**"com.atguigu.gmall"**}) **public class** ServiceItemApplication {   **public static void** main(String[] args) {  SpringApplication.*run*(ServiceItemApplication.**class**, args);  }  } |

### 1.3 service-item服务接口封装

|  |
| --- |
| **public interface** ItemService {   */\*\*  \* 获取sku详情信息  \** ***@param skuId*** *\** ***@return*** *\*/* Map<String, Object> getBySkuId(Long skuId); } |
| @Service **public class** ItemServiceImpl **implements** ItemService {   @Override  **public** Map<String, Object> getBySkuId(Long skuId) {  Map<String, Object> result = **new** HashMap<>();     **return** result;  } } |
| @RestController @RequestMapping(**"api/item"**) **public class** ItemApiController { @Autowired  **private** ItemService **itemService**;   */\*\*  \* 获取sku详情信息  \** ***@param skuId*** *\** ***@return*** *\*/* @GetMapping(**"{skuId}"**)  **public** Result getItem(@PathVariable Long skuId){  Map<String,Object> result = **itemService**.getBySkuId(skuId);  **return** Result.*ok*(result);  }  } |

说明：商品详情相关信息在service-product微服务都可以获取，所以我们在service-product模块编写所需要的接口

## 2 在service-product微服务提供api接口

### 2.1 获取sku基本信息与图片信息



|  |
| --- |
| ManageService  接口  */\*\*  \* 根据skuId 查询skuInfo  \** ***@param skuId*** *\** ***@return*** *\*/* SkuInfo getSkuInfo(Long skuId); |
| 实现类  @Override **public** SkuInfo getSkuInfo(Long skuId) {  SkuInfo skuInfo = **skuInfoMapper**.selectById(skuId);  *// 根据skuId 查询图片列表集合* QueryWrapper<SkuImage> queryWrapper = **new** QueryWrapper<>();  queryWrapper.eq(**"sku\_id"**, skuId);  List<SkuImage> skuImageList = **skuImageMapper**.selectList(queryWrapper);   skuInfo.setSkuImageList(skuImageList);  **return** skuInfo; } |
| 控制器  com.atguigu.gmall.product.api.ProductApiController  @RestController @RequestMapping(**"api/product"**) **public class** ProductApiController {   @Autowired  **private** ManageService **manageService**; */\*\*  \* 根据skuId获取sku信息  \** ***@param skuId*** *\** ***@return*** *\*/* @GetMapping(**"inner/getSkuInfo/{skuId}"**)  **public** SkuInfo getAttrValueList(@PathVariable(**"skuId"**) Long skuId){  SkuInfo skuInfo = **manageService**.getSkuInfo(skuId);  **return** skuInfo;  } |

### 2.2 获取分类信息

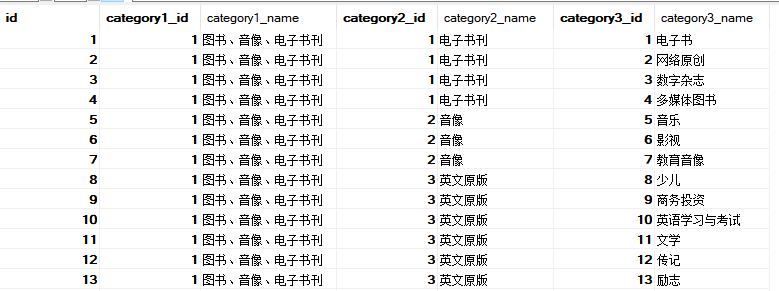


需求：

sku是挂在三级分类下面的，我们的分类信息分别在base\_category1、base\_category2、base\_category3这三张表里面，目前需要通过sku表的三级分类id获取一级分类名称、二级分类名称和三级分类名称

解决方案：

我们可以建立一个视图，把三张表关联起来，视图id就是三级分类id，这样通过三级分类id就可以查询到相应数据，效果如下：



|  |
| --- |
| 创建视图  CREATE VIEW base\_category\_view AS  select  c3.id as id,  c1.id as category1\_id, c1.name as category1\_name,  c2.id as category2\_id, c2.name as category2\_name,  c3.id as category3\_id, c3.name as category3\_name  from base\_category1 c1  inner join base\_category2 c2 on c2.category1\_id = c1.id  inner join base\_category3 c3 on c3.category2\_id = c2.id |
| 视图  @Data @ApiModel(description = **"分类视图"**) @TableName(**"base\_category\_view"**) **public class** BaseCategoryView **extends** BaseEntity {    **private static final long *serialVersionUID*** = 1L;    @ApiModelProperty(value = **"一级分类编号"**)  @TableField(**"category1\_id"**)  **private** Long **category1Id**;   @ApiModelProperty(value = **"一级分类名称"**)  @TableField(**"category1\_name"**)  **private** String **category1Name**;   @ApiModelProperty(value = **"二级分类编号"**)  @TableField(**"category2\_id"**)  **private** Long **category2Id**;   @ApiModelProperty(value = **"二级分类名称"**)  @TableField(**"category2\_name"**)  **private** String **category2Name**;   @ApiModelProperty(value = **"三级分类编号"**)  @TableField(**"category3\_id"**)  **private** Long **category3Id**;   @ApiModelProperty(value = **"三级分类名称"**)  @TableField(**"category3\_name"**)  **private** String **category3Name**;  } |
| Mapper  @Mapper **public interface** BaseCategoryViewMapper **extends** BaseMapper<BaseCategoryView> {  } |
| ManageService接口  */\*\*  \* 通过三级分类id查询分类信息  \** ***@param category3Id*** *\** ***@return*** *\*/* BaseCategoryView getCategoryViewByCategory3Id(Long category3Id); |
| 接口实现  @Override **public** BaseCategoryView getCategoryViewByCategory3Id(Long category3Id) {  **return baseCategoryViewMapper**.selectById(category3Id); } |
| ProductApiController  */\*\*  \* 通过三级分类id查询分类信息  \** ***@param category3Id*** *\** ***@return*** *\*/* @GetMapping(**"inner/getCategoryView/{category3Id}"**) **public** BaseCategoryView getCategoryView(@PathVariable(**"category3Id"**)Long category3Id){  **return manageService**.getCategoryViewByCategory3Id(category3Id); } |

### 2.3 获取价格信息

|  |
| --- |
| */\*\*  \* 获取sku价格  \** ***@param skuId*** *\** ***@return*** *\*/* BigDecimal getSkuPrice(Long skuId); |
| */\*\*  \* 获取sku价格  \** ***@param skuId*** *\** ***@return*** *\*/* @Override **public** BigDecimal getSkuPrice(Long skuId) {  SkuInfo skuInfo = **skuInfoMapper**.selectById(skuId);  **if**(**null** != skuInfo) {  **return** skuInfo.getPrice();  }  **return new** BigDecimal(**"0"**); } |
| */\*\*  \* 获取sku最新价格  \** ***@param skuId*** *\** ***@return*** *\*/* @GetMapping(**"inner/getSkuPrice/{skuId}"**) **public** BigDecimal getSkuPrice(@PathVariable Long skuId){  **return manageService**.getSkuPrice(skuId); } |

### 2.4 获取销售信息



思路：

1、查出该商品的spu的所有销售属性和属性值

2、标识出本商品对应的销售属性

3、点击其他销售属性值的组合，跳转到另外的sku页面

#### 2.4.1 查询出sku对应spu的销售属性

第1、2条通过此sql实现

|  |
| --- |
| SELECT sa.id ,sa.spu\_id, sa.sale\_attr\_name,sa.base\_sale\_attr\_id,  sv.id sale\_attr\_value\_id,  sv.sale\_attr\_value\_name,  skv.sku\_id,  IF(skv.sku\_id IS NULL,0,1) is\_checked  FROM spu\_sale\_attr sa  INNER JOIN spu\_sale\_attr\_value sv ON sa.spu\_id=sv.spu\_id AND sa.base\_sale\_attr\_id=sv.base\_sale\_attr\_id  LEFT JOIN sku\_sale\_attr\_value skv ON skv.sale\_attr\_value\_id= sv.id AND skv.sku\_id=#{skuId}  WHERE sa.spu\_id=#{spuId}  ORDER BY sv.base\_sale\_attr\_id,sv.id |

此sql列出所有该spu的销售属性和属性值，并关联某skuid如果能关联上is\_check设为1，否则设为0。

在对应的实体类中【SpuSaleAttrValue】添加属性字段

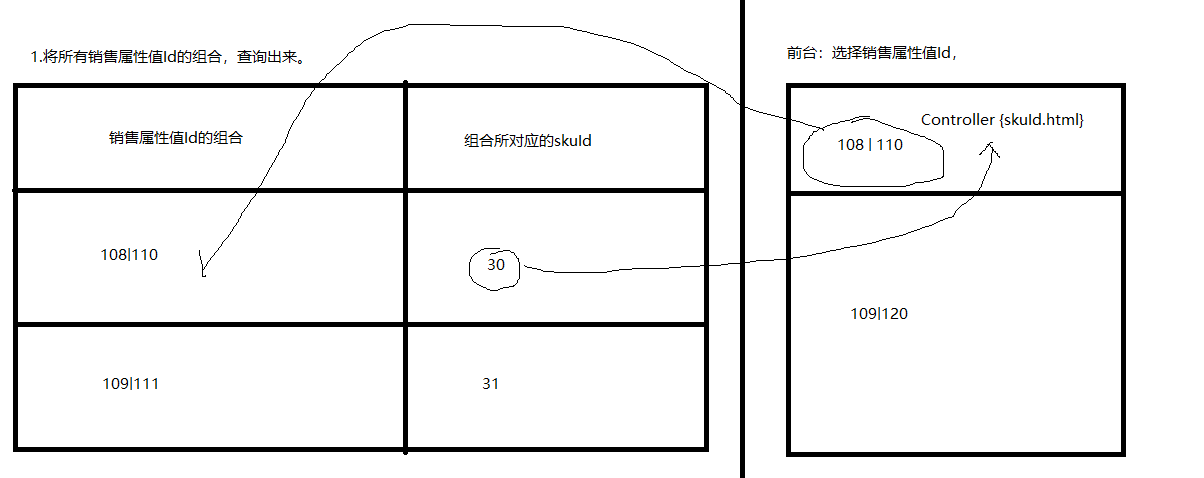
@TableField(exist = **false**)  
String **isChecked**

|  |
| --- |
| 接口 SpuSaleAttrMapper  *// 根据spuId，skuId 查询销售属性集合* List<SpuSaleAttr> selectSpuSaleAttrListCheckBySku(@Param(**"skuId"**) Long skuId, @Param(**"spuId"**)Long spuId); |
| <**select id="selectSpuSaleAttrListCheckBySku" resultMap="spuSaleAttrMap"**>  SELECT sa.id ,sa.spu\_id, sa.sale\_attr\_name,sa.base\_sale\_attr\_id,  sv.id sale\_attr\_value\_id,  sv.sale\_attr\_value\_name,  skv.sku\_id,  IF(skv.sku\_id IS NULL,0,1) is\_checked  FROM spu\_sale\_attr sa  INNER JOIN spu\_sale\_attr\_value sv ON sa.spu\_id=sv.spu\_id AND sa.base\_sale\_attr\_id=sv.base\_sale\_attr\_id  LEFT JOIN sku\_sale\_attr\_value skv ON skv.sale\_attr\_value\_id= sv.id AND skv.sku\_id=#{skuId}  WHERE sa.spu\_id=#{spuId}  ORDER BY sv.base\_sale\_attr\_id,sv.id </**select**> |
| ManageService 接口  */\*\*  \* 根据spuId，skuId 查询销售属性集合  \** ***@param skuId*** *\** ***@param spuId*** *\** ***@return*** *\*/* List<SpuSaleAttr> getSpuSaleAttrListCheckBySku(Long skuId, Long spuId); |
| 实现类  @Override **public** List<SpuSaleAttr> getSpuSaleAttrListCheckBySku(Long skuId, Long spuId) {  **return spuSaleAttrMapper**.selectSpuSaleAttrListCheckBySku(skuId, spuId); } |
| ProductApiController  */\*\*  \* 根据spuId，skuId 查询销售属性集合  \** ***@param skuId*** *\** ***@param spuId*** *\** ***@return*** *\*/* @GetMapping(**"inner/getSpuSaleAttrListCheckBySku/{skuId}/{spuId}"**) **public** List<SpuSaleAttr> getSpuSaleAttrListCheckBySku(@PathVariable(**"skuId"**) Long skuId,@PathVariable(**"spuId"**) Long spuId){  **return manageService**.getSpuSaleAttrListCheckBySku(skuId, spuId); } |

#### 2.4.2 点击其他销售属性值的组合，跳转到另外的sku页面



实现思路：



1 、从页面中获得得所有选中的销售属性进行组合比如：

“属性值1|属性值2|属性值3” 用这个字符串匹配一个对照表，来获得skuId。并进行跳转，或者告知无货。

2、后台要生成一个“属性值1|属性值2|属性值3：skuId”的一个json串以提供页面进行匹配。如

3、需要从后台数据库查询出该spu下的所有skuId和属性值关联关系。然后加工成如上的Json串，用该json串，跟前台匹配。

要注意排序，方便后面整理。

实现类很简单：

使用sql 语句来解决：

GROUP\_CONCAT：group\_concat( [distinct] 要连接的字段 [order by 排序字段 asc/desc ] [separator '分隔符'] )

|  |
| --- |
| SkuSaleAttrValueMapper.xml  *<!--定义Map的resultMap-->* <**resultMap id="spuValueIdsMap" type="java.util.Map" autoMapping="true"**>  </**resultMap**>  <**select id="getSaleAttrValuesBySpu" resultMap="spuValueIdsMap"**>  SELECT sku\_id , GROUP\_CONCAT(sale\_attr\_value\_id ORDER BY sp.base\_sale\_attr\_id ASC SEPARATOR '|') value\_ids  FROM `sku\_sale\_attr\_value` sv  INNER JOIN `spu\_sale\_attr\_value` sp on sp.id = sv.sale\_attr\_value\_id  WHERE sv.spu\_id=#{spuId}  GROUP BY sku\_id </**select**> |
| SkuSaleAttrValueMapper  *// 根据spuId 查询map 集合数据* List<Map> getSaleAttrValuesBySpu(Long spuId); |
| ManageService接口  */\*\*  \* 根据spuId 查询map 集合属性  \** ***@param spuId*** *\** ***@return*** *\*/* Map getSkuValueIdsMap(Long spuId); |
| 实现类  @Override **public** Map getSkuValueIdsMap(Long spuId) {  Map<Object, Object> map = **new** HashMap<>();  *// key = 125|123 ,value = 37* List<Map> mapList = **skuSaleAttrValueMapper**.getSaleAttrValuesBySpu(spuId);  **if** (mapList != **null** && mapList.size() > 0) {  *// 循环遍历* **for** (Map skuMap : mapList) {  *// key = 125|123 ,value = 37* map.put(skuMap.get(**"value\_ids"**), skuMap.get(**"sku\_id"**));  }  }  **return** map; } |
| ProductApiController  */\*\*  \* 根据spuId 查询map 集合属性  \** ***@param spuId*** *\** ***@return*** *\*/* @GetMapping(**"inner/getSkuValueIdsMap/{spuId}"**) **public** Map getSkuValueIdsMap(@PathVariable(**"spuId"**) Long spuId){  **return manageService**.getSkuValueIdsMap(spuId); } |

说明：目前我们在service-product里面把数据模型已经封装好了，接下封装feign client api接口，提供给service-item微服务调用汇总数据模型

## 3搭建service-client父模块

改模块管理所有微服务feign client api模块

### 3.1 搭建service-client父模块

搭建方式如：common父模块

### 3.2 修改配置

修改pom.xml文件

|  |
| --- |
| *<?***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 https://maven.apache.org/xsd/maven-4.0.0.xsd"**>  <**modelVersion**>4.0.0</**modelVersion**>   <**modules**>  <**module**>service-product-client</**module**>  <**module**>service-item-client</**module**>  </**modules**>  <**parent**>  <**groupId**>com.atguigu.gmall</**groupId**>  <**artifactId**>gmall-parent</**artifactId**>  <**version**>1.0</**version**>  </**parent**>   <**artifactId**>service-client</**artifactId**>  <**packaging**>pom</**packaging**>  <**version**>1.0</**version**>   <**dependencies**>  <**dependency**>  <**groupId**>com.atguigu.gmall</**groupId**>  <**artifactId**>common-util</**artifactId**>  <**version**>1.0</**version**>  </**dependency**>   <**dependency**>  <**groupId**>com.atguigu.gmall</**groupId**>  <**artifactId**>model</**artifactId**>  <**version**>1.0</**version**>  </**dependency**>   <**dependency**>  <**groupId**>org.springframework.boot</**groupId**>  <**artifactId**>spring-boot-starter-web</**artifactId**>  </**dependency**>   *<!-- 服务调用feign -->* <**dependency**>  <**groupId**>org.springframework.cloud</**groupId**>  <**artifactId**>spring-cloud-starter-openfeign</**artifactId**>  <**scope**>provided </**scope**>  </**dependency**>  </**dependencies**>  </**project**> |

## 4搭建service-product-client

### 3.1 构建模块

搭建方式如上

### 3.2 修改配置

修改pom.xml

|  |
| --- |
| *<?***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 https://maven.apache.org/xsd/maven-4.0.0.xsd"**>  <**modelVersion**>4.0.0</**modelVersion**>  <**parent**>  <**groupId**>com.atguigu.gmall</**groupId**>  <**artifactId**>service-client</**artifactId**>  <**version**>1.0</**version**>  </**parent**>   <**artifactId**>service-product-client</**artifactId**>  <**version**>1.0</**version**>   <**packaging**>jar</**packaging**>  <**name**>service-product-client</**name**>  <**description**>service-product-client</**description**>   <**dependencies**>  </**dependencies**>  </**project**> |

### 3.3 封装service-product接口

|  |
| --- |
| @FeignClient(value =**"service-product"**, fallback = ProductDegradeFeignClient.**class**) **public interface** ProductFeignClient {   */\*\*  \* 根据skuId获取sku信息  \*  \** ***@param skuId*** *\** ***@return*** *\*/* @GetMapping(**"/api/product/inner/getSkuInfo/{skuId}"**)  SkuInfo getSkuInfo(@PathVariable(**"skuId"**) Long skuId);    */\*\*  \* 通过三级分类id查询分类信息  \** ***@param category3Id*** *\** ***@return*** *\*/* @GetMapping(**"/api/product/inner/getCategoryView/{category3Id}"**)  BaseCategoryView getCategoryView(@PathVariable(**"category3Id"**)Long category3Id);   */\*\*  \* 获取sku最新价格  \*  \** ***@param skuId*** *\** ***@return*** *\*/* @GetMapping(**"/api/product/inner/getSkuPrice/{skuId}"**)  BigDecimal getSkuPrice(@PathVariable(value = **"skuId"**) Long skuId);   */\*\*  \* 根据spuId，skuId 查询销售属性集合  \*  \** ***@param skuId*** *\** ***@param spuId*** *\** ***@return*** *\*/* @GetMapping(**"/api/product/inner/getSpuSaleAttrListCheckBySku/{skuId}/{spuId}"**)  List<SpuSaleAttr> getSpuSaleAttrListCheckBySku(@PathVariable(**"skuId"**) Long skuId, @PathVariable(**"spuId"**) Long spuId);   */\*\*  \* 根据spuId 查询map 集合属性  \** ***@param spuId*** *\** ***@return*** *\*/* @GetMapping(**"/api/product/inner/getSkuValueIdsMap/{spuId}"**)  Map getSkuValueIdsMap(@PathVariable(**"spuId"**) Long spuId);  } |
| @Component **public class** ProductDegradeFeignClient **implements** ProductFeignClient {   @Override  **public** SkuInfo getSkuInfo(Long skuId) {  **return null**;  }   @Override  **public** BaseCategoryView getCategoryView(Long category3Id) {  **return null**;  }   @Override  **public** BigDecimal getSkuPrice(Long skuId) {  **return null**;  }   @Override  **public** List<SpuSaleAttr> getSpuSaleAttrListCheckBySku(Long skuId, Long spuId) {  **return null**;  }    @Override  **public** Map getSkuValueIdsMap(Long spuId) {  **return null**;  } } |

说明：接下来service-item引用service-product-client模块，就可以调用相应接口

在service-item pom.xml引用依赖：

<**dependency**>  
 <**groupId**>com.atguigu.gmall</**groupId**>  
 <**artifactId**>service-product-client</**artifactId**>  
 <**version**>1.0</**version**>  
</**dependency**>

## 5 service-item模块汇总数据

|  |
| --- |
| ItemServiceImpl  @Service **public class** ItemServiceImpl **implements** ItemService {   @Autowired  **private** ProductFeignClient **productFeignClient**;   @Override  **public** Map getItem(Long skuId) {  Map map = **new** HashMap();   *//SKuInfo* SkuInfo skuInfo = **productFeignClient**.getSkuInfo(skuId);   BaseCategoryView categoryView = **productFeignClient**.getCategoryView(skuInfo.getCategory3Id());   BigDecimal skuPrice = **productFeignClient**.getSkuPrice(skuId);   List<SpuSaleAttr> spuSaleAttrListCheckBySkuList = **productFeignClient**.  getSpuSaleAttrListCheckBySku(skuId, skuInfo.getSpuId());   Map skuValueIdsMap = **productFeignClient**.getSkuValueIdsMap(skuInfo.getSpuId());   map.put(**"skuInfo"**,skuInfo);  map.put(**"categoryView"**,categoryView);  map.put(**"price"**,skuPrice);  map.put(**"spuSaleAttrList"**,spuSaleAttrListCheckBySkuList);  map.put(**"valuesSkuJson"**, JSON.*toJSONString*(skuValueIdsMap));   **return** map; }  }  */\*\*  \* 商品详情管理数据汇总  \*/*  @RestController @RequestMapping(**"/api/item"**) **public class** ItemController {    @Autowired  **private** ItemService **itemService**;   *//查询汇总数据* @GetMapping(**"/{skuId}"**)  **public** Map getItem(@PathVariable(name = **"skuId"**) Long skuId){  **return itemService**.getItem(skuId);  } } |

说明：

1. 目前商品详情页的数据就汇总好了，可以提供给pc端、h5、安卓等前端应用使用了，可以通过swagger测试接口
2. 由于我们pc端是通过Thymeleaf实现页面渲染，所以我们需搭建web端页面渲染服务，移动端可以直接通过http+json形式调用service-item服务渲染页面；
3. 由于我们的微服务接口都是通过网关暴露服务，所以需要配置网关

网关添加配置:

**routes**:  
- **id**: service-product  
 **uri**: lb://service-product  
 **predicates**:  
 - Path=/\*/product/\*\* *# 路径匹配*- **id**: service-item  
 **uri**: lb://service-item  
 **predicates**:  
 - Path=/\*/item/\*\*

*#==================web前端==========================*  
- **id**: web-all  
 **uri**: lb://web-all  
 **predicates**:  
 - Host=item.gmall.com

## 6 商品详情页面渲染

### 6.1 构建web父模块

构建方式如：common父模块

修改配置pom.xml

|  |
| --- |
| *<?***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"**>  <**modules**>  <**module**>web-item</**module**>  </**modules**>   <**parent**>  <**groupId**>com.atguigu.gmall</**groupId**>  <**artifactId**>gmall-parent</**artifactId**>  <**version**>1.0</**version**>  </**parent**>   <**artifactId**>web</**artifactId**>  <**packaging**>pom</**packaging**>  <**version**>1.0</**version**>   <**dependencies**>  <**dependency**>  <**groupId**>org.springframework.boot</**groupId**>  <**artifactId**>spring-boot-starter-web</**artifactId**>  </**dependency**>   <**dependency**>  <**groupId**>org.springframework.boot</**groupId**>  <**artifactId**>spring-boot-starter-thymeleaf</**artifactId**>  </**dependency**>   *<!-- 服务注册 -->* <**dependency**>  <**groupId**>org.springframework.cloud</**groupId**>  <**artifactId**>spring-cloud-starter-alibaba-nacos-discovery</**artifactId**>  </**dependency**>   *<!-- 服务配置 -->  <!--<dependency>-->  <!--<groupId>org.springframework.cloud</groupId>-->  <!--<artifactId>spring-cloud-starter-alibaba-nacos-config</artifactId>-->  <!--</dependency>-->   <!-- 服务调用feign -->* <**dependency**>  <**groupId**>org.springframework.cloud</**groupId**>  <**artifactId**>spring-cloud-starter-openfeign</**artifactId**>  </**dependency**>   *<!-- 流量控制 -->* <**dependency**>  <**groupId**>org.springframework.cloud</**groupId**>  <**artifactId**>spring-cloud-starter-alibaba-sentinel</**artifactId**>  </**dependency**>  </**dependencies**>  </**project**> |

### 6.2 构建web-all模块

#### 6.2.1搭建web-all模块

搭建方式同上

#### 6.2.2修改pom.xml文件

|  |
| --- |
| <**modelVersion**>4.0.0</**modelVersion**> <**parent**>  <**groupId**>com.atguigu.gmall</**groupId**>  <**artifactId**>web</**artifactId**>  <**version**>1.0</**version**> </**parent**>  <**artifactId**>web-all</**artifactId**> <**version**>1.0</**version**>  <**packaging**>jar</**packaging**> <**name**>web-all</**name**> <**description**>web-all</**description**>  <**dependencies**>  <**dependency**>  <**groupId**>com.atguigu.gmall</**groupId**>  <**artifactId**>service-item-client</**artifactId**>  <**version**>1.0</**version**>  </**dependency**> </**dependencies**>   <**build**>  <**finalName**>web-all</**finalName**>  <**plugins**>  <**plugin**>  <**groupId**>org.springframework.boot</**groupId**>  <**artifactId**>spring-boot-maven-plugin</**artifactId**>  </**plugin**>  </**plugins**> </**build**> |

#### 6.2.3添加配置文件

|  |
| --- |
| application.yml  **spring**:  **application**:  **name**: web-all  **profiles**:  **active**: dev  **cloud**:  **nacos**:  **discovery**:  **server-addr**: 127.0.0.1:8848 |
| application-dev.yml  **server**:  **port**: 8304  **spring**:  **thymeleaf**:  *#模板的模式，支持 HTML, XML TEXT JAVASCRIPT* **mode**: HTML5  *#编码 可不用配置* **encoding**: UTF-8  *#开发配置为false,避免修改模板还要重启服务器* **cache**: **false** *#配置模板路径，默认是templates，可以不用配置* **prefix**: classpath:/templates/  **cloud**:  **sentinel**:  **transport**:  **dashboard**: 127.0.0.1:8080  **jackson**:  **date-format**: yyyy-MM-dd HH:mm:ss  **time-zone**: GMT+8 **feign**:  **sentinel**:  **enabled**: true  **client**:  **config**:  **default**:  **readTimeout**: 30000  **connectTimeout**: 10000 |

#### 6.2.4启动类

|  |
| --- |
| **package** com.atguigu.gmall.all;  **import** org.springframework.boot.SpringApplication; **import** org.springframework.boot.autoconfigure.SpringBootApplication; **import** org.springframework.boot.autoconfigure.jdbc.DataSourceAutoConfiguration; **import** org.springframework.cloud.client.discovery.EnableDiscoveryClient; **import** org.springframework.cloud.openfeign.EnableFeignClients; **import** org.springframework.context.annotation.ComponentScan;  @SpringBootApplication(exclude = DataSourceAutoConfiguration.**class**)*//取消数据源自动配置* @ComponentScan({**"com.atguigu.gmall"**}) @EnableDiscoveryClient @EnableFeignClients(basePackages= {**"com.atguigu.gmall"**}) **public class** WebAllApplication {   **public static void** main(String[] args) {  SpringApplication.*run*(WebAllApplication.**class**, args);  }  } |

### 6.4 调用接口

#### 6.4.1搭建service-item-client模块

搭建方式同service-product-client

接口类

|  |
| --- |
| **package** com.atguigu.gmall.item.client  @FeignClient(value = **"service-item"**, fallback = ItemDegradeFeignClient.**class**) **public interface** ItemFeignClient {   */\*\*  \*  \** ***@param skuId*** *\** ***@return*** *\*/* @GetMapping(**"/api/item/{skuId}"**)  Result getItem(@PathVariable(**"skuId"**) Long skuId);  } |
| @Component **public class** ItemDegradeFeignClient **implements** ItemFeignClient {    @Override  **public** Result getItem(Long skuId) {  **return** Result.*fail*();  } } |

#### 6.4.2搭建web-all引入依赖

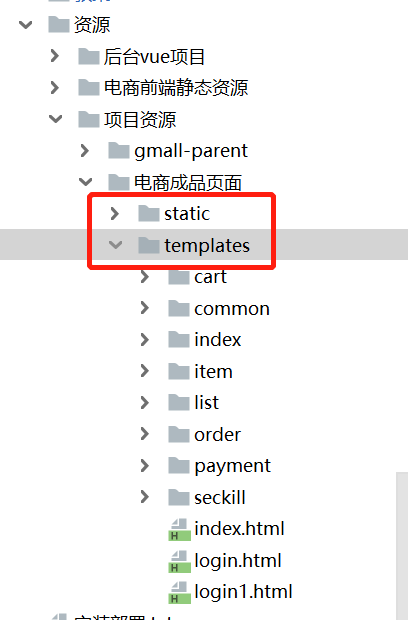
<**dependency**>  
 <**groupId**>com.atguigu.gmall</**groupId**>  
 <**artifactId**>service-item-client</**artifactId**>  
 <**version**>1.0</**version**>  
</**dependency**>

#### 6.4.3调用接口

|  |
| --- |
| **Package** com.atguigu.gmall.item.controller  @Controller @RequestMapping **public class** ItemController {   @Autowired  **private** ItemFeignClient **itemFeignClient**;   */\*\*  \* sku详情页面  \** ***@param skuId*** *\** ***@param model*** *\** ***@return*** *\*/* @RequestMapping(**"{skuId}.html"**)  **public** String getItem(@PathVariable Long skuId, Model model){  *// 通过skuId 查询skuInfo* Result<Map> result = **itemFeignClient**.getItem(skuId);  model.addAllAttributes(result.getData());  **return "item/index"**;  }  } |

#### 6.4.4页面渲染

说明：前端成品页面都放在 “资料\电商成品页面”目录下，有需要直接在该目录下寻找，后续不做说明，如图：



商品详情页面资源： \templates\item\index1.html

该目录有两个文件：

item/index.html：提取了公共头信息，登录讲解后提取

item/index1.html为没提取公共信息页面，当前取该页面，更改为index.html

前期讲解可以把包含部分文件copy过来，后续不做说明

1. 分类信息

<**ul class="sui-breadcrumb"**>  
 <**li**>  
 <**a th:href="@{http://list.gmall.com/list.html?category1Id={category1Id}(category1Id=${categoryView.category1Id})}" th:text="${categoryView.category1Name}"**>手机、数码、通讯</**a**>  
 </**li**>  
 <**li**>  
 <**a th:href="@{http://list.gmall.com/list.html?category2Id={category2Id}(category2Id=${categoryView.category2Id})}" th:text="${categoryView.category2Name}"**>手机</**a**>  
 </**li**>  
 <**li**>  
 <**a th:href="@{http://list.gmall.com/list.html?category3Id={category3Id}(category3Id=${categoryView.category3Id})}" th:text="${categoryView.category3Name}"**>Apple苹果</**a**>  
 </**li**>  
 <**li class="active" th:text="${skuInfo.skuName}"**>iphone 6S系类</**li**>  
</**ul**>

1. 图片信息

<**div class="zoom"**>  
 *<!--默认第一个预览-->* <**div id="preview" class="spec-preview"**>  
<**span class="jqzoom"**>  
 <**img th:jqimg="${skuInfo.skuDefaultImg}" th:src="${skuInfo.skuDefaultImg}" width="400" height="400"**/>  
</**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="skuImage: ${skuInfo.skuImageList}"**>  
 <**img th:src="${skuImage.imgUrl}" th:bimg="${skuImage.imgUrl}" onmousemove="***preview*(**this**)**"**/>  
 </**li**>  
 </**ul**>  
 </**div**>  
 <**a class="next"**>**&gt;**</**a**>  
 </**div**>  
 </**div**>

1. 销售属性

<**dl v-for="spuSaleAttr in spuSaleAttrList" :key="spuSaleAttr.id"**>  
 <**dt**>  
 <**div class="fl title"**>  
 <**i**>{{spuSaleAttr.saleAttrName}}</**i**>  
 </**div**>  
 </**dt**>  
 <**dd v-for="spuSaleAttrValue in spuSaleAttr.spuSaleAttrValueList" :key="spuSaleAttrValue.id" @click="spuSaleAttrFun(spuSaleAttr.index, spuSaleAttrValue.id)"**>  
 <**a href="javascript:;" :class="{'selected':spuSaleAttrValue.isChecked == 1}"**>{{spuSaleAttrValue.saleAttrValueName}}<**span title="点击取消选择"**>**&nbsp;**</**span**></**a**>  
 </**dd**>  
</**dl**>

<**script th:inline="javascript"**>  
 **var *item*** = **new Vue**({  
 **el**: **'#itemArray'**,  
 **data**: {  
 **skuId**: [[${skuInfo.id}]],  
 **valuesSku**: **''**,  
 **skuNum**: 1,  
  
 **selectValuesSkus**: [],  
 **isAddCart**: **true**,  
  
 **spuSaleAttrList**: [[${spuSaleAttrList}]],  
 **valuesSkuJson**: [[${valuesSkuJson}]]  
 },  
  
 created() {  
 **this**.init()  
 },  
  
 **methods**: {  
 init() {  
 **debugger  
 this**.**spuSaleAttrList**.forEach((spuSaleAttr, index) => {  
 spuSaleAttr.**index** = index  
 spuSaleAttr.spuSaleAttrValueList.forEach((spuSaleAttrValue) => {  
 **if** (spuSaleAttrValue.**isChecked** == 1) {  
 **this**.**selectValuesSkus**.push(spuSaleAttrValue.**id**)  
 }  
 })  
 })  
 ***console***.log(**'this.valuesSku:'** + **this**.**valuesSku**)  
 },  
  
 addToCart() {  
 *// 判断是否登录和是否存在临时用户，如果都没有，添加临时用户* **if**(!***auth***.isTokenExist() && !***auth***.isUserTempIdExist()) {  
 ***auth***.setUserTempId()  
 }  
 ***window***.**location**.**href** = **'http://cart.gmall.com/addCart.html?skuId='** + **this**.**skuId** + **'&skuNum='** + **this**.**skuNum** },  
  
 spuSaleAttrFun(index, spuSaleAttrValueId) {  
 **debugger  
 this**.**isAddCart** = **true  
 var** valuesSkuJson = ***JSON***.parse(**this**.**valuesSkuJson**);  
  
 *//根据索引替换已选中的spuSaleAttrValueId* **this**.**selectValuesSkus**[index] = spuSaleAttrValueId  
  
 **var** valuesSku = **this**.**selectValuesSkus**.join(**'|'**)  
 *//查看新组装的数据是否存在skuId，如果存在跳到该页面，如果不存在控制选中状态* **var** curSkuId = valuesSkuJson[valuesSku]  
  
 **if** (curSkuId) {  
 **if**(curSkuId != **this**.**skuId**) {  
 ***window***.**location**.**href** = **'/'** + curSkuId + **'.html'** } **else** {  
 **this**.**isAddCart** = **true** }  
 } **else** {  
 *//控制购物车是否可选* **this**.**isAddCart** = **false** }  
  
 *//控制销售属性是否选中* **this**.**spuSaleAttrList**.forEach((spuSaleAttr, i) => {  
 spuSaleAttr.spuSaleAttrValueList.forEach((spuSaleAttrValue) => {  
 **if** (index == i) {  
 **if** (spuSaleAttrValueId == spuSaleAttrValue.**id**) {  
 spuSaleAttrValue.**isChecked** = 1  
 } **else** {  
 spuSaleAttrValue.**isChecked** = 0  
 }  
 }  
 })  
 })  
 }  
 }  
 })  
</**script**>

说明：由于销售属性渲染相对复杂，所以采用vue渲染

4，其他属性看详情成品页