尚品汇商城

一、商品详情相关业务介绍

- 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。
- 用户点击不同的销售属性值切换不同的商品
- 点击添加购物车，将商品放入购物车列表中

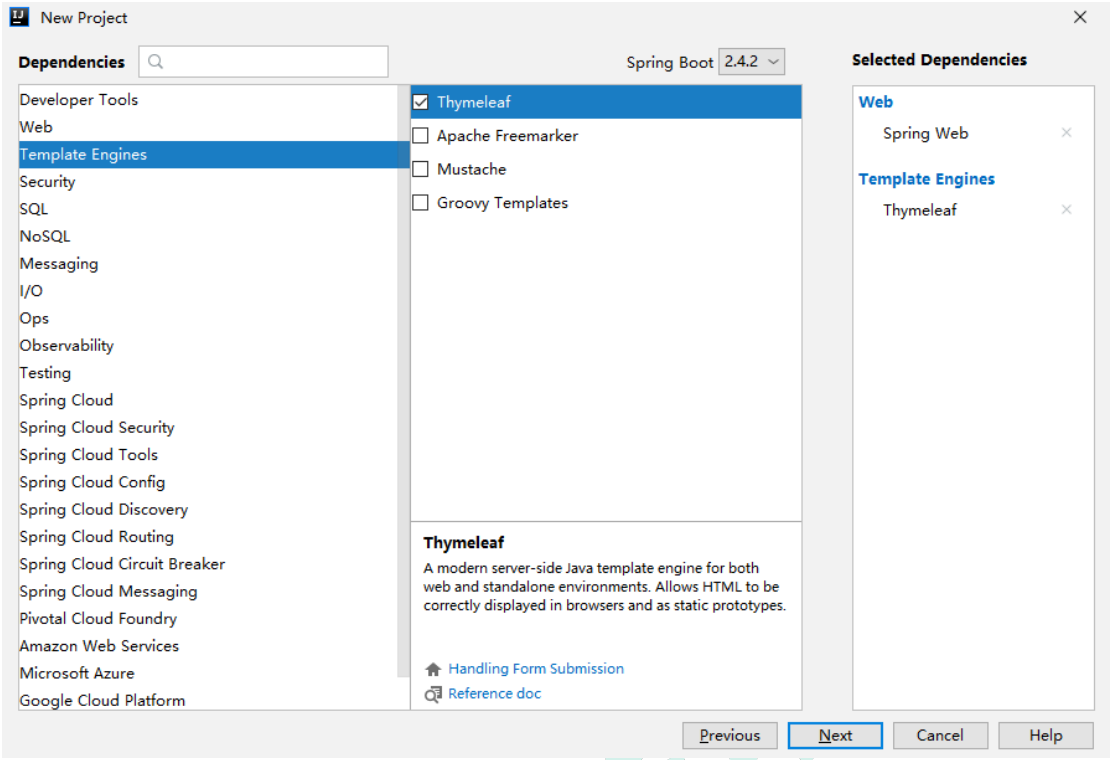
二、模板技术 Thymeleaf 介绍

2.1 Thymeleaf 简介

Thymeleaf 是一款用于**渲染 XML/XHTML/HTML5 内容的模板引擎**。类似 JSP，Velocity，FreeMaker 等， 它也可以轻易的与 Spring MVC 等 Web 框架进行集成作为 Web 应用的模板引擎。与其它模板引擎相比， Thymeleaf 最大的**特点是能够直接在浏览器中打开**并正确显示模板页面，而不需要启动整个 Web 应用!

2.2 快速入门

项目创建，依赖模块 web，Thymeleaf.模板。



2.2.1 设置头文件

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">
```

2.2.2 赋值字符串拼接

```
request.setAttribute("name","刘德华");

<p th:text="'hello' + ${name}" th:value="${name}"></p>
```

2.2.3 循环

```
List<String> list = Arrays.asList("郑爽", "刘德华", "张惠
```

```
妹", "成龙");

request.setAttribute("arrayList",list);



<!--测试循环-->
<table>
    <!--s 表示集合中的元素 ${slist}表示后台存储的集合 -->
    <tr th:each="s,stat: ${arrayList}">
        <td th:text="${s}"></td>
        <td th:text="${stat.index}"></td>
        <td th:text="${stat.count}"></td>
        <td th:text="${stat.size}"></td>
        <td th:text="${stat.even}"></td>
        <td th:text="${stat.odd}"></td>
        <td th:text="${stat.first}"></td>
        <td th:text="${stat.last}"></td>
    </tr>
</table>
```

stat 称作状态变量，属性有

index：当前迭代对象的 index（从 0 开始计算）

count：当前迭代对象的 index（从 1 开始计算）

size：被迭代对象的大小

even/odd：布尔值，当前循环是否是偶数/奇数

first：布尔值，当前循环是否是第一个

last：布尔值，当前循环是否是最后一个

2.2.4 判断

th:if 条件成立显示

th:unless 条件不成立的时候才会显示内容

```
model.addAttribute("age",18);

<h2>判断 if</h2>
<div th:if="${age}>=18" th:text="success">good</div>
<a th:unless="${age != 18}" th:text="success" >atguigu</a>
```

```
<h2>判断 三元</h2>
<div th:text="${age}>=18?'success':'failure'"></div>
```

2.2.5 取 session 中的属性

```
httpSession.setAttribute("addr","北京中南海");

<div th:text="${session.addr}"> </div>
```

2.2.6 引用内嵌页

```
<div th:include="itemInner"/>
```

2.2.7 th:utext :解析样式

th:utext:识别 html 中的标签

```
request.setAttribute("gname","<span style=color:green>绿色</span>");

<p th:utext="${gname}">color</p>
```

2.2.8 点击链接传值

```
<a
th:href="@{http://localhost:8080/list.html?category1Id={category1Id}
(category1Id=${category1Id})}">点我带你飞</a>
```

```
@RequestMapping("list.html")
@ResponseBody
public String list(String category1Id){
    // 接收传递过来的数据
    System.out.println("获取到的数据: \t"+category1Id);
    return category1Id;
}
```

在 index 控制器中先存储一个 category1Id

```
/*保存 category1Id*/
```

```
request.setAttribute("category1Id","2");
```

2.2.9 多种存储方式

```
model.addAttribute("addr","北京昌平区");
hashMap.put("id","101");

HashMap<String, Object> map = new HashMap<>();
map.put("stuNo","1000");
map.put("stuName","张三");
model.addAllAttributes(map);
```

```
<h2> 多种方式存储数据 1 </h2>

<div th:text="${addr}"></div>
<div th:text="${id}"></div>

<h2> 多种方式存储数据 2 </h2>

<div th:text="${stuNo}"></div>
<div th:text="${stuName}"></div>
```

三、商品详情业务需求分析

3.1 详情渲染功能介绍

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，为了回显最新价格，所以单独获取）

6，展示商品的海报

7，获取 skuId 对应的商品规格参数

...

3.2 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;

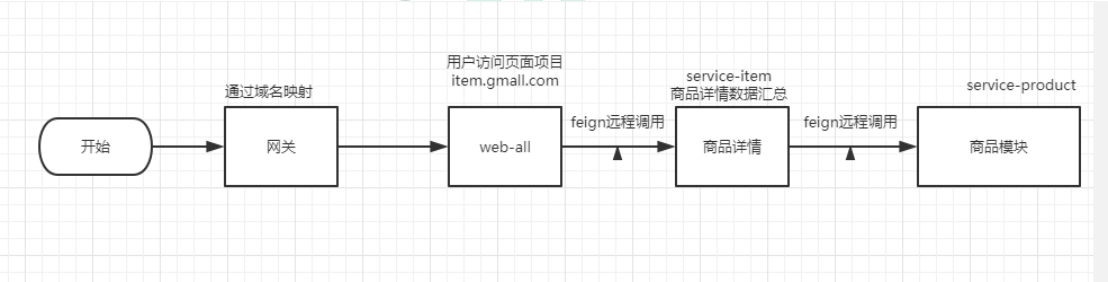
2，service-item 通过 feign client 调用其他微服务数据接口进行数据汇总;

3，pc 端前台页面通过 web-all 调用 service-item 数据接口渲染页面;

4，service-item 可以为 pc 端、H5、安卓与 ios 等前端应用提供数据接口，web-all 为 pc 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。

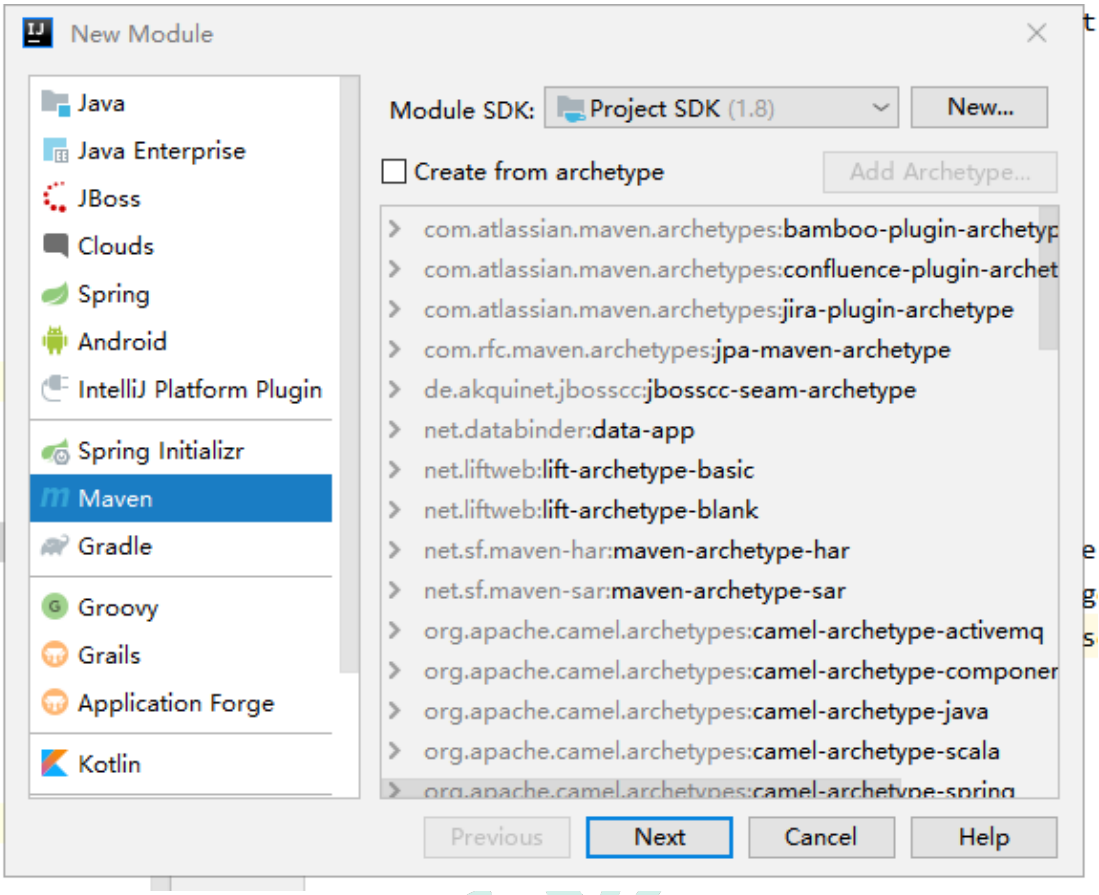


四、商品详情功能开发

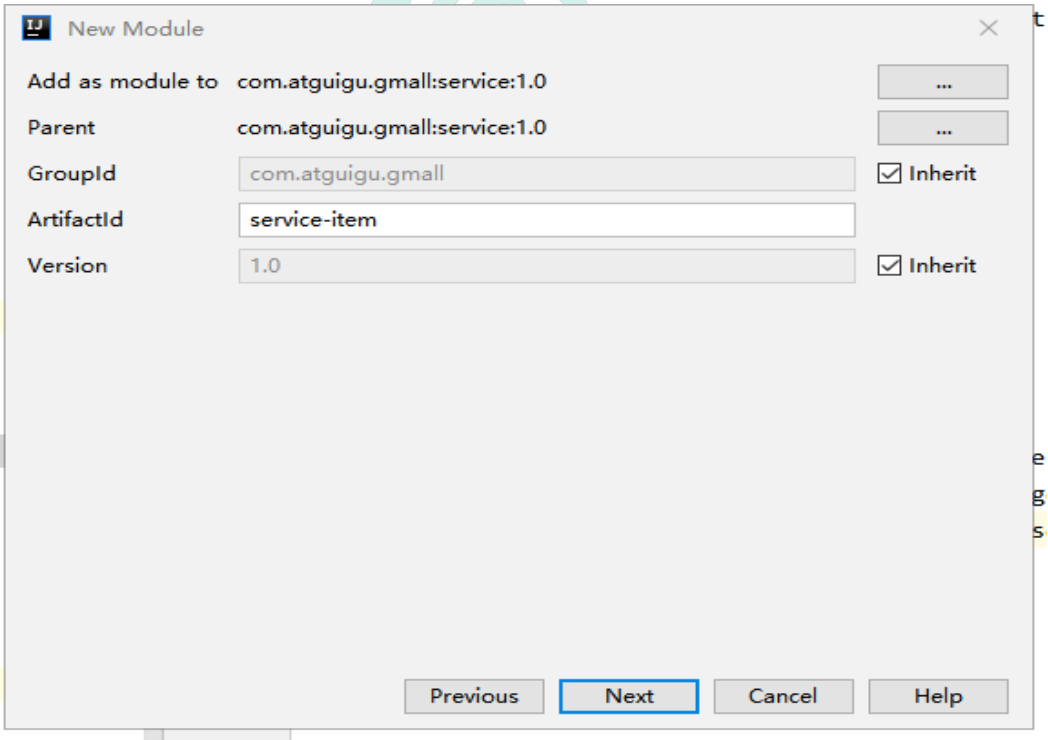
4.1 搭建 service-item

4.1.1 构建模块

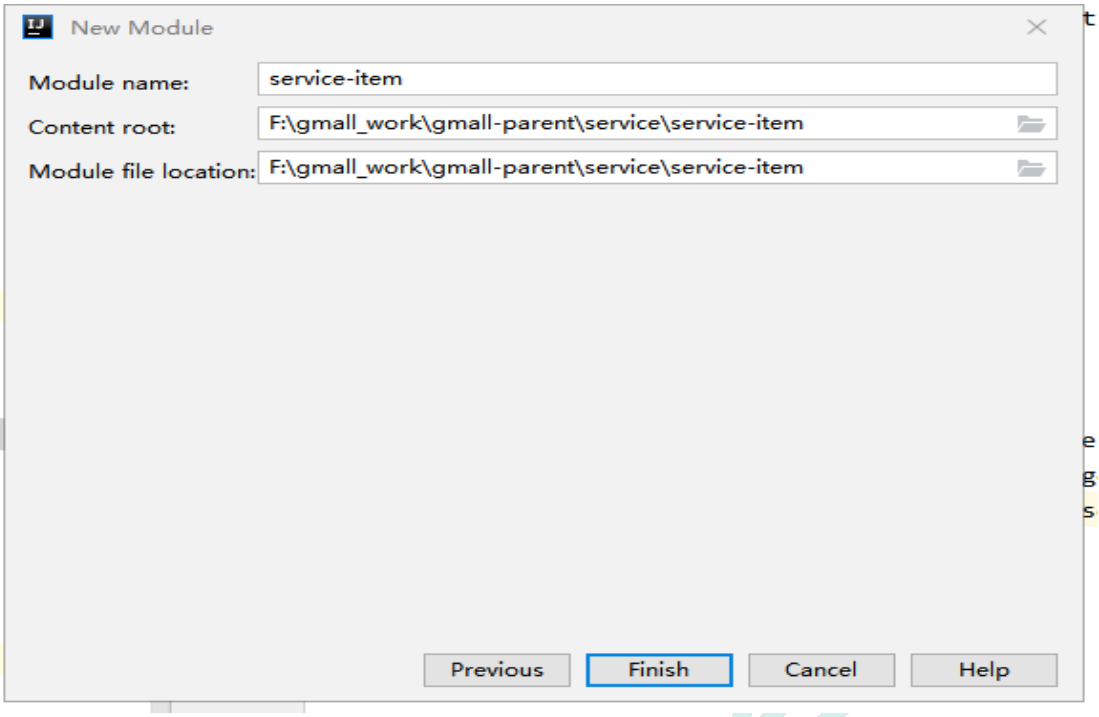
点击 service，选择 New–>Module,操作如下


New Module
Java
Java Enterprise
JBoss
Clouds
Spring
Android
IntelliJ Platform Plugin
Spring Initializr
Maven
Gradle
Groovy
Grails
Application Forge
Kotlin
Module SDK: Project SDK (1.8)
New...
Create from archetype
Add Archetype...
com.atlassian.maven.archetypes:bamboo-plugin-archetyp
com.atlassian.maven.archetypes:confluence-plugin-archet
com.atlassian.maven.archetypes:jira-plugin-archetype
com.rfc.maven.archetypes:jpa-maven-archetype
de.akquinet.jbosscc:jbosscc-seam-archetype
net.databinder:data-app
net.liftweb:lift-archetype-basic
net.liftweb:lift-archetype-blank
net.sf.maven-har:maven-archetype-har
net.sf.maven-sar:maven-archetype-sar
org.apache.camel.archetypes:camel-archetype-activemq
org.apache.camel.archetypes:camel-archetype-componer
org.apache.camel.archetypes:camel-archetype-java
org.apache.camel.archetypes:camel-archetype-scala
Previous
Next
Cancel
Help

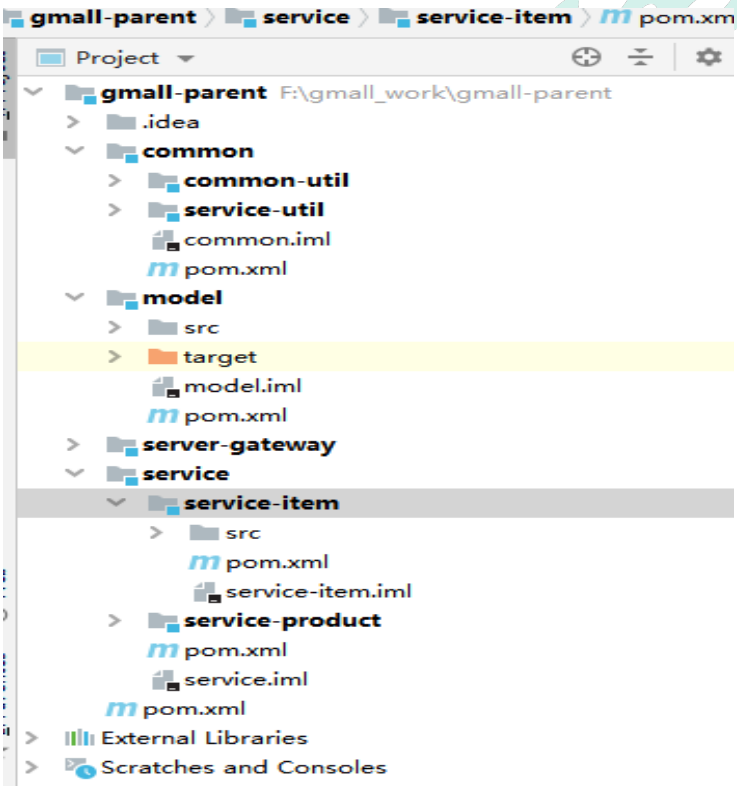
下一步


New Module
Add as module to com.atguigu.gmall:service:1.0
Parent com.atguigu.gmall:service:1.0
GroupId com.atguigu.gmall
Inherit
ArtifactId service-item
Version 1.0
Inherit
Previous
Next
Cancel
Help

下一步



完成，结构如下



4.1.2 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>


    <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 bootstrap.properties

```
spring.application.name=service-item
spring.profiles.active=dev
spring.cloud.nacos.discovery.server-addr=192.168.200.129:8848
spring.cloud.nacos.config.server-addr=192.168.200.129:8848
spring.cloud.nacos.config.prefix=${spring.application.name}
spring.cloud.nacos.config.file-extension=yaml
spring.cloud.nacos.config.shared-configs[0].data-id=common.yaml
```

添加启动类

exclude = DataSourceAutoConfiguration.class 排除数据库链接 jar

表示当前项目{service-item} 不参与数据库查询

```
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);
    }

}
```

4.1.3 service-item 服务接口封装

```
package com.atguigu.gmall.item.service;


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

4.2 在 service-product 微服务提供 api 接口

4.2.1 获取 sku 基本信息与图片信息

4.2.1.1 编写接口与实现类

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;
}
```

4.2.1.2 编写控制器

```
package 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;
    }
```

4.2.2 获取分类信息

4.2.2.1 需求分析：

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

MySQL 视图（View）是一种虚拟存在的表，同真实表一样，视图也由列和行构成，但视图并不实际存在于数据库中。行和列的数据来自于定义视图的查询中所使用的表，并且还是在使用视图时动态生成的。

特点：

数据库中只存放了视图的定义，并没有存放视图中的数据，这些数据都存放在定义视图查询所引用的真实表中。

解决方案：

我们可以建立一个**视图(view)**，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：

id	category1_id	category1_name	category2_id	category2_name	category3_id	category3_name
1	1	图书、音像、电子书刊	1	电子书刊	1	电子书
2	1	图书、音像、电子书刊	1	电子书刊	2	网络原创
3	1	图书、音像、电子书刊	1	电子书刊	3	数字杂志
4	1	图书、音像、电子书刊	1	电子书刊	4	多媒体图书
5	1	图书、音像、电子书刊	2	音像	5	音乐
6	1	图书、音像、电子书刊	2	音像	6	影视
7	1	图书、音像、电子书刊	2	音像	7	教育音像
8	1	图书、音像、电子书刊	3	英文原版	8	少儿
9	1	图书、音像、电子书刊	3	英文原版	9	商务投资
10	1	图书、音像、电子书刊	3	英文原版	10	英语学习与考试
11	1	图书、音像、电子书刊	3	英文原版	11	文学
12	1	图书、音像、电子书刊	3	英文原版	12	传记
13	1	图书、音像、电子书刊	3	英文原版	13	励志

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
```

4.2.2.2 创建 mapper

Mapper

```
@Mapper
public interface BaseCategoryViewMapper extends
BaseMapper<BaseCategoryView> {

}
```

4.2.2.3 编写接口与实现类

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);
}
```

4.2.2.4 编写控制器

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);
}
```

4.2.3 获取价格信息

4.2.3.1 编写接口与实现类

```
/**
 * 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");
}
```

4.2.3.2 编写控制器

```
/**
 * 获取 sku 最新价格
 * @param skuId
 * @return
 */
@GetMapping("inner/getSkuPrice/{skuId}")
public BigDecimal getSkuPrice(@PathVariable Long skuId){
    return manageService.getSkuPrice(skuId);
}
```

4.2.4 获取销售信息

id	spu_id	base_sale_attr_id	sale_attr_name	sale_attr_value_id	sale_attr_value_name	sku_id	is_checked
5206	7	1	颜色	3720	蓝色星球	18	1
5207	7	2	版本	3722	6G+64G	18	1
5206	7	1	颜色	3721	仙女渐变色	<null>	0
5207	7	2	版本	3723	6G+128G	<null>	0

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

4.2.4.1 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;
```

4.2.4.2 在 SpuSaleAttrMapper 接口中添加的方法

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>
```

4.2.4.3 编写接口与实现类

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);
}
```

4.2.4.4 编写控制器

ProductApiController

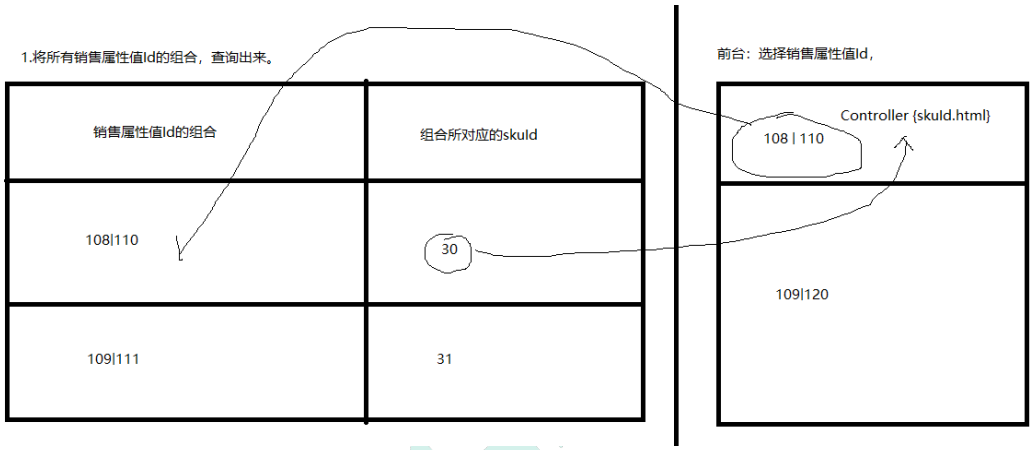
```
/**
 * 根据spuId，skuId 查询销售属性集合
 * @param skuId
 * @param spuId
 * @return
 */
@GetMapping("inner/getSpuSaleAttrListCheckBySku/{skuId}/{spuId}")
public                                              List<SpuSaleAttr>
getSpuSaleAttrListCheckBySku(@PathVariable("skuId")              Long
skuId,@PathVariable("spuId") Long spuId){
    return manageService.getSpuSaleAttrListCheckBySku(skuId, spuId);
}
```

4.2.5 实现商品切换

实现思路：

4 rows

	value_ids	sku_id
1	3720|3722	18
2	3720|3723	19
3	3721|3723	20
4	3721|3722	21



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” 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：skuId”的一个 json 串以提供页面进行匹配。如

```
valuesSku:{"46|50":"10","47|50":"13","48|49":"12","47|49":"11"}
```

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。

实现：

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'])

4.2.5.1 在 SkuSaleAttrValueMapper 中添加接口

SkuSaleAttrValueMapper

```
// 根据spuId 查询map 集合数据
List<Map> selectSaleAttrValuesBySpu(Long spuId);
```

SkuSaleAttrValueMapper.xml

```
<!--定义Map 的resultMap-->
<resultMap id="spuValueIdsMap" type="java.util.Map" autoMapping="true">

</resultMap>

<select id="selectSaleAttrValuesBySpu" 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>
```

4.2.5.2 编写接口与实现类

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.selectSaleAttrValuesBySpu(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;
}
```

4.2.5.3 编写控制器

```
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);
}
```

4.2.6 获取海报信息

4.2.6.1 编写接口与实现类

```
/**
 * 根据spuid获取商品海报
 * @param spuId
 * @return
 */
List<SpuPoster> findSpuPosterBySpuId(Long spuId);
```

```
@Override
public List<SpuPoster> findSpuPosterBySpuId(Long spuId) {
    QueryWrapper<SpuPoster> spuInfoQueryWrapper = new QueryWrapper<>();
    spuInfoQueryWrapper.eq("spu_id",spuId);
    List<SpuPoster>                    spuPosterList                    =
spuPosterMapper.selectList(spuInfoQueryWrapper);
    return spuPosterList;
}
```

4.2.6.2 编写控制器

```
//  根据spuId 获取海报数据
```

```
@GetMapping("inner/findSpuPosterBySpuId/{spuId}")
public List<SpuPoster> findSpuPosterBySpuId(@PathVariable Long spuId){
    return manageService.findSpuPosterBySpuId(spuId);
}
```

4.2.7 Sku 对应的平台属性

	id	attr_name	category_id	category_level	attr_value_id	attr_id	value_name
1	106	手机系统	2	1	176	106	安卓手机
2	107	手机品牌	13	2	177	107	小米
3	23	运行内存	61	3	169	23	6G
4	24	机身内存	61	3	81	24	64G
5	115	CPU型号	61	3	196	115	骁龙845
6	116	屏幕尺寸	61	3	203	116	6.55-6.64英寸

4.2.7.1 编写接口与实现类

显示在商品详情规格处

ManageService

```
/**
 * 通过skuId 集合来查询数据
 * @param skuId
 * @return
 */
List<BaseAttrInfo> getAttrList(Long skuId);
```

实现类

```
@Override
public List<BaseAttrInfo> getAttrList(Long skuId) {

    return baseAttrInfoMapper.selectBaseAttrInfoListBySkuId(skuId);
}
```

BaseAttrInfoMapper 添加方法

```
   /**
     *
     * @param skuId
     */
    List<BaseAttrInfo>
selectBaseAttrInfoListBySkuId(@Param("skuId")Long skuId);
```

BaseAttrInfoMapper.xml 添加

```
<select                                    id="selectBaseAttrInfoListBySkuId"
resultMap="baseAttrInfoMap">
    SELECT
    bai.id,
    bai.attr_name,
    bai.category_id,
    bai.category_level,
    bav.id attr_value_id,
    bav.value_name,
    bav.attr_id
    FROM
    base_attr_info bai
    INNER JOIN base_attr_value bav ON bai.id = bav.attr_id
    INNER JOIN sku_attr_value sav ON sav.value_id = bav.id
    WHERE
    sav.sku_id = #{skuId}
</select>
```

4.2.7.2 编写控制器

```
ProductApiController

/**
 * 通过skuId 集合来查询数据
 * @param skuId
 * @return
 */
@GetMapping("inner/getAttrList/{skuId}")
public  List<BaseAttrInfo>  getAttrList(@PathVariable("skuId")  Long
skuId){
    return manageService.getAttrList(skuId);
}
```

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

4.3 搭建 service-client 父模块

改模块管理所有微服务 feign client api 模块

4.3.1 搭建 service-client 父模块

搭建方式如：common 父模块

4.3.2 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>
```

4.4 搭建 service-product-client

4.4.1 构建模块

在 service-client 模块下创建

4.4.2 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>

</project>
```

4.4.3 封装 service-product-client 接口

```
package com.atguigu.gmall.product.client;


@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);

   //  根据 spuId 获取海报数据
   @GetMapping("/api/product/inner/findSpuPosterBySpuId/{spuId}")
   List<SpuPoster> getSpuPosterBySpuId(@PathVariable Long spuId);

    /**
     * 通过 skuId 集合来查询数据
     * @param skuId
     * @return
     */
    @GetMapping("/api/product/inner/getAttrList/{skuId}")
    List<BaseAttrInfo> getAttrList(@PathVariable("skuId") Long skuId);

}
```

```
@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;
    }

    @Override
    public List<SpuPoster> getSpuPosterBySpuId(Long spuId) {
        return null;
    }

    @Override
    public List<BaseAttrInfo> getAttrList(Long skuId) {
        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>
```

4.5 service-item 模块汇总数据

```
@Service
public class ItemServiceImpl implements ItemService {

    //  远程调用service-product-client
    @Autowired
    private ProductFeignClient productFeignClient;

    @Override
    public Map<String, Object> getItemBySkuId(Long skuId) {
        //  声明对象
       Map<String, Object> result = new HashMap<>();

        //  获取到的数据是skuInfo + skuImageList
        SkuInfo skuInfo = productFeignClient.getSkuInfo(skuId);

        //  判断skuInfo 不为空
```

```
        if (skuInfo!=null){
            //  获取分类数据
            BaseCategoryView categoryView = productFeignClient.getCategoryView(skuInfo.getCategory3Id());
            result.put("categoryView",categoryView);
            //  获取销售属性+销售属性值
            List<SpuSaleAttr> spuSaleAttrListCheckBySku = productFeignClient.getSpuSaleAttrListCheckBySku(skuId, skuInfo.getSpuId());
            result.put("spuSaleAttrList",spuSaleAttrListCheckBySku);
            //  查询销售属性值 Id 与 skuId 组合的 map
            Map skuValueIdsMap = productFeignClient.getSkuValueIdsMap(skuInfo.getSpuId());
            //  将这个 map 转换为页面需要的 Json 对象
            String valueJson = JSON.toJSONString(skuValueIdsMap);
            result.put("valuesSkuJson",valueJson);

        }
        //  获取价格
        BigDecimal skuPrice = productFeignClient.getSkuPrice(skuId);
        //  map 中 key 对应的谁? Thymeleaf 获取数据的时候 ${skuInfo.skuName}
        result.put("skuInfo",skuInfo);
        result.put("price",skuPrice);
        //  返回 map 集合 Thymeleaf 渲染: 能用 map 存储数据!

        //  spu 海报数据
        List<SpuPoster> spuPosterList = productFeignClient.findSpuPosterBySpuId(skuInfo.getSpuId());
        result.put("spuPosterList", spuPosterList);

        List<BaseAttrInfo> attrList = productFeignClient.getAttrList(skuId);
        //  使用拉姆达表示
        List<Map<String, String>> skuAttrList = attrList.stream().map((baseAttrInfo) -> {
            Map<String, String> attrMap = new HashMap<>();
            attrMap.put("attrName", baseAttrInfo.getAttrName());
            attrMap.put("attrValue", baseAttrInfo.getAttrValueList().get(0).getValueName());
            return attrMap;
        }).collect(Collectors.toList());
        result.put("skuAttrList", skuAttrList);


        return result;
    }
}
```

4.6 商品详情页面渲染

4.6.1 搭建 service-item-client 模块

在 service-client 目录下创建。

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();
    }
}
```

4.6.2 搭建 web-util 模块

4.6.2.1 搭建 web-util

搭建方式如 service-util

4.6.2.2 修改配置 pom.xml

```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">
    <parent>
        <artifactId>common</artifactId>
        <groupId>com.atguigu.gmall</groupId>
        <version>1.0</version>
    </parent>
    <modelVersion>4.0.0</modelVersion>

    <artifactId>web-util</artifactId>

    <dependencies>
        <dependency>
            <groupId>com.atguigu.gmall</groupId>
            <artifactId>common-util</artifactId>
            <version>1.0</version>
        </dependency>

        <!-- 服务调用feign -->
         <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-openfeign</artifactId>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
            <scope>provided </scope>
        </dependency>
    </dependencies>

</project>
```

导入工具类：

4.6.3 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-all</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>com.atguigu.gmall</groupId>
            <artifactId>web-util</artifactId>
            <version>1.0</version>
        </dependency>
        <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>com.alibaba.cloud</groupId>

<artifactId>spring-cloud-starter-alibaba-nacos-discovery</artifactId
>
        </dependency>

       <!-- 服务配置-->
```

```
        <dependency>
            <groupId>com.alibaba.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>com.alibaba.cloud</groupId>
<artifactId>spring-cloud-starter-alibaba-sentinel</artifactId>
        </dependency>
    </dependencies>

</project>
```

4.6.4 构建 web-all 模块

4.6.4.1 搭建 web-all 模块

搭建方式在 web 模块下创建

4.6.4.2 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>
```

```xml
<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>
```

4.6.4.3 添加配置文件

bootstrap.properties

```
spring.application.name=web-all
spring.profiles.active=dev
spring.cloud.nacos.discovery.server-addr=192.168.200.129:8848
spring.cloud.nacos.config.server-addr=192.168.200.129:8848
spring.cloud.nacos.config.prefix=${spring.application.name}
spring.cloud.nacos.config.file-extension=yaml
```

启动类

```
@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);
    }
}
```

4.6.4.4 将 web-all 模块添加到网关

1，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

server-gateway 网关添加配置：

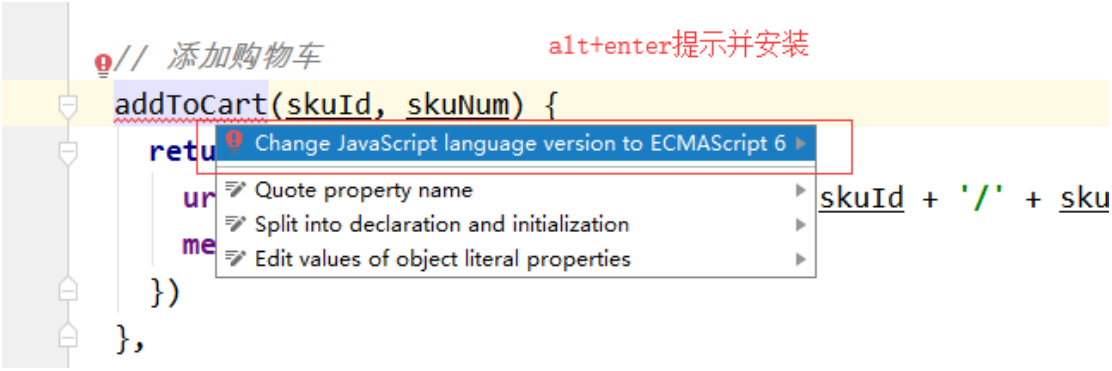
```
routes:
- id: service-product
  uri: lb://service-product
  predicates:
  - Path=/*/product/** # 路径匹配
- id: service-item
  uri: lb://service-item
  predicates:
  - Path=/*/item/**

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

4.6.4.5 导入静态资源

在 web-all 工 将这两个文件夹放入到 resouces 文件夹中。

导入之后，可能发送异常警告

解决方案：



4.6.5 编写 web-all 中的控制器

4.6.5.1 在 web-all 调用接口

```
Package com.atguigu.gmall.all.controller

@Controller
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";
    }
}
```