# 今日任务

1. 理解spu，sku等概念（重要）
2. 商品业务分析
3. 表设计讨论（重点）
4. 完成商品查询、新增（图片上传除外）
5. 商品发布管理之上架、下架

# spu、sku

SKU和SPU算是电商中最基础的概念，SKU是**最小库存单元**，SPU是**标准化产品单元**。 很多电商从业者在入门的时候都会被这两个概念绕晕。面试时一问就破了相了，笑中带泪。

SPU与SKU的关系有许多种，可以**一对多、 一对一**。绝大部分SPU与SKU都是一对一，**多规格**的SPU和SKU之间是通过**规格属性**来连接的。 **SPU的库存是由其对应的SKU库存共同决定的**。

举例说明：以iPhone 7 Plus（SPU） 为例，这个SPU的规格有多种（颜色包含金色、 白色、 黑色、 玫瑰金、 银色、 亮黑、 红色等6种；内存包含32G、128G、 256G等3种） ，对应18（即3×6） 种SKU。 比如“iPhone 7 Plus白色32G”、 “iPhone 7 Plus黑色32G”这两个SKU都能具化到实物。 仓库系统、 采购系统、 库存系统、 订单中心等系统都是主要管理**SKU**。

# 商品编码

在电商系统中流转的是系统生成的SPU编码和SKU编码。 这里要介绍一下**69码**，我国官方的商品编码是**69开头，俗称69码**，例如食品包装上的条码。 在编辑商品的时候，一般会有个条形码字段需要填写，就是系统中的SKU对应69码。 这个条形码主要在采购、 出入库时使用。

在仓库的条码方案以下有两种。

（1） 全部自建条码。 采用自己的SKU编码管理商品，需要入库前重新贴标，成本较高。

（2） 有69码的商品沿用69码，无69码的商品（比如蔬菜、 生鲜等） 以及无法指定到单一的有码商品重新贴SKU编码。 在仓库管理时做相应的条码映射。

**理解以上概念是为了大家能更好的应对面试，只有真正做过才是真的懂！！**

# 规格、价格

需要注意的是，在设置规格时，主要包括颜色、 尺寸，为了支持多样化的用户需求，选择之后可以编辑规格。 规格一对一确定之后，可单独设置价格、 库存、 商家SKU，淘宝或天猫上亦可添加条形码（69码） ，也可以设置统一价、 统一库存。 填写商家SKU主要是为了方便对应到具体的SKU。 对于自营电商，这里填写的就是SKU编码，库存无法设置，直接同步仓库中的SKU库存。 系统中的SKU名称和商品名称是有区别的，SKU名称是方便在系统中进行管理流转，而商品名称有一定的营销性质。平台上会针对商品设置平台价和市场价。 平台价主要是当商品具有不同规格、 价格时，出现在搜索、 筛选列表中只显示一个价格，相当于商品的均价。 毛重、 长宽高等数据主要是为了物流而设置的，自建仓库的自营电商一般在SKU数据层就会录入这些数据，直接调用。 货号即商品编码，在商城购物时会扫描的条形码就是货号。 货号不等同于SKU编码，同一商品编码的商品可能是不同SKU，有着不同的规格，所以不能直接拿货号来管理SKU。

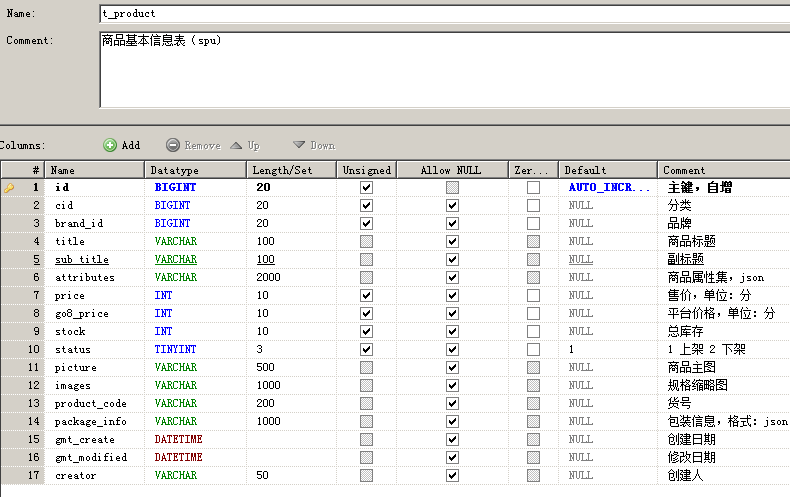
# 详细、图片

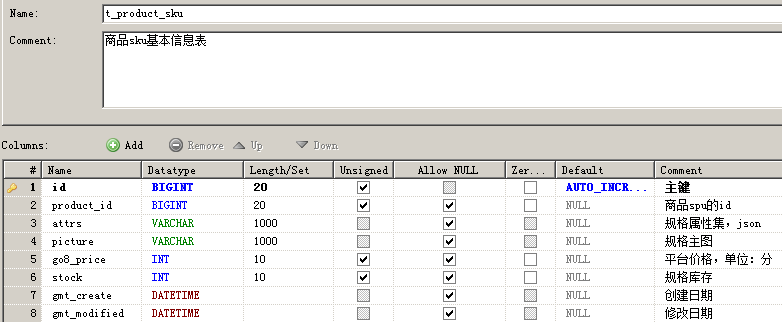
**除了不同规格对应的商品缩略图，商品图还包括商品主图，一般要求图片质量较高，包括整体图和细节图。** 商品主图是吸引顾客眼球的必要利器，不论是**列表页**，还是**活动页**，顾客除了关注价格，主要就是商品主图，运营上架时需慎重选择商品主图。商品详情页现在一般会区分电脑版和手机版，由于两者的使用场景和设备不同，侧重点也不相同。 为了更好地展示产品特点，可提供不同的产品详情模板，亦可支持富文本编辑。选择运费服务时，要选择对应的物流模板（包邮、 按重量、 按件数等） ，在订单处理方面是按照具体的物流模板计算运费。

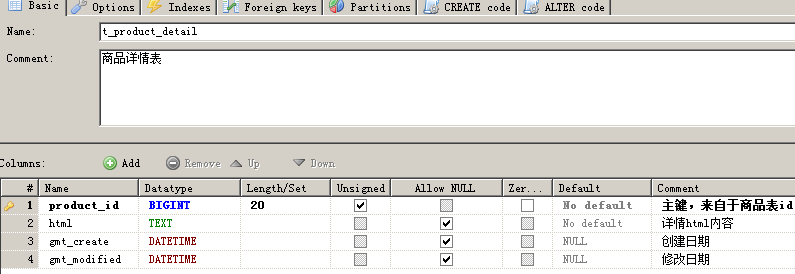
# 关系

这里主要有**商品表(spu)**，**商品规格表**，**详情表**三张表

# 表设计







# 表设计分析

商品表应该是电商表里的核心单元，设计差别比较大，各家不一，不同的理解，会有完全不同的设计结果。这里不想去讨论谁对谁错。我给出了三种设计方案，大家一起来分析。

1. 遵守范式的标准设计

商品表

 Spu\_id

........

规格表

Sku\_id

Spu\_id

........

商品、规格和属性关联表

Id

Spu\_id

Sku\_id

Attr\_value\_id

Attr\_id

1:n

n:n

n:n

优点：无冗余，可以关联查询

缺点：考虑到属性表，分组表，关联的表太多，速度是大问题

其次，sql语句书写也会很复杂，不信你可以试试，写的出来的同学给

你们点个赞。

适用场合：数据量少的小型电商

2) 保留部分关系设计

商品表

 Spu\_id

Attrs:

**[{attrId:valueId}]**

........

规格表

Sku\_id

Spu\_id

Sale\_attrs:

**[{attrId:valueId}]**

........

1:n

优点：去除了一张商品属性关联表，少了一次关联查询，直接得到了属性和值 的主键，**无冗余**，由于关联的是主键，属性值等修改时不需要更新该字段内容。

缺点：仍然需要关联查询，得到具体 属性值以及分组等信息

3) 无关联设计

商品表

Spu\_id

Attrs:

**[{**

**“name”:name,**

**“value”:value,**

**“group”:gname,**

**“type”: type**

**}]**

........

规格表

Sku\_id

Spu\_id

Sale\_attrs:

**[{**

**“name”:name,**

**“value”:value,**

**“group”:gname,**

**“type”: type**

**}]**

........

1:n

无分组属性，**“group”**值为null

优点：无任何关联查询，所有业务信息都只保存在一个字段中，查询速度极快

缺点：当修改属性值时，需要更新该字段，哪怕是仅仅修改其中的一个属性值而已

其次，增加了业务代码书写的难度

适用场景：大数据量的中大型电商

**经验：尽量避免大数据量表的关联查询**

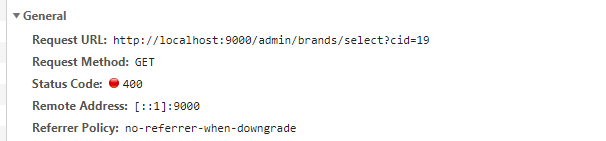
多个表关联，海量数据全表扫描查询，想想就心慌！！

# 接口约定

|  |  |  |  |
| --- | --- | --- | --- |
| **业务类型** | **method** | **url** | **return body** |
| 添加 | post | products | 返回一个统一的json  对象，  {  “code”: xxx,  “message”:”xxxx”,  “data”:{} or []  } |
| 修改 | put | products/{id} |
| 删除 | delete | products /{id} |
| 查询单体 | get | products /{id} |
| 分页查询 | get | products?page&size&[可选条件] |
| 商品批量上架 | put | products/up?ids |  |
| 商品批量下架 | put | products/down?ids |  |

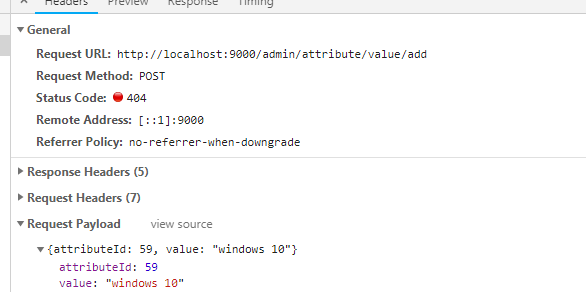
# 代码

## 查询品牌

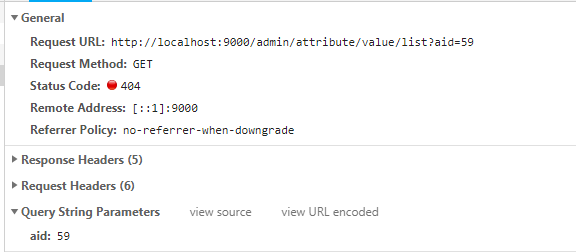




## 属性添加



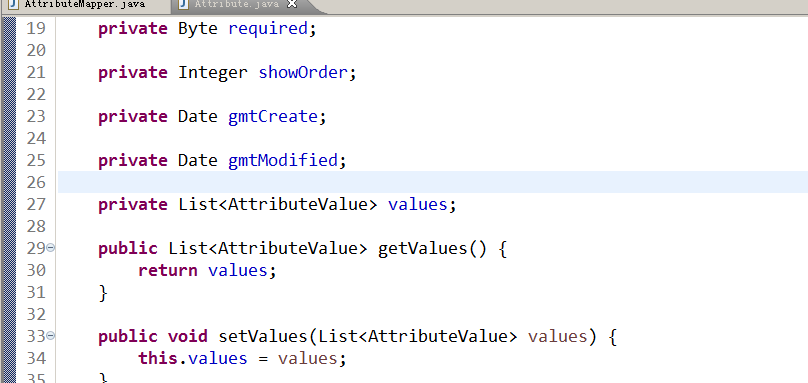
## 属性值查询



## 属性和属性值查询

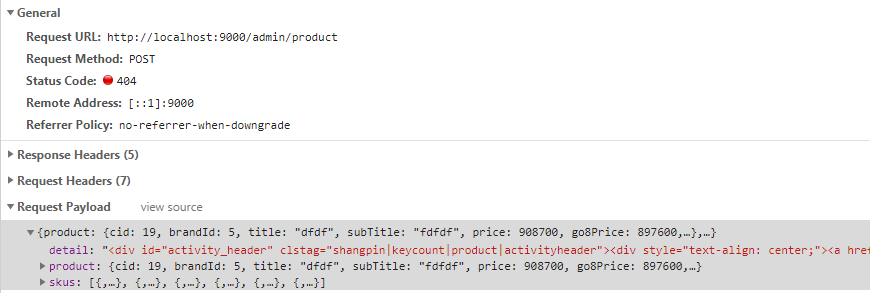
商品添加页面，选择分类后，同时查询属性和属性值（否则，无属性值显示）

一对多查询

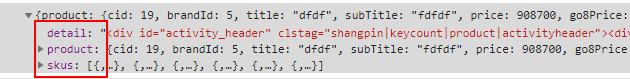


|  |
| --- |
| <!-- 一对多查询，自定义resultmap -->  <resultMap id=*"AttrAndValueResultMap"* type=*"com.go9.goods.pojo.Attribute"* extends=*"BaseResultMap"*>  <collection property=*"values"* ofType=*"com.go9.goods.pojo.AttributeValue"*>  <id property=*"id"* column=*"vid"*></id>  <result property=*"value"* column=*"value"*/>  </collection>  </resultMap>  <select id=*"selectAttrAndValueByAttribute"* parameterType=*"com.go9.goods.pojo.Attribute"* resultMap=*"AttrAndValueResultMap"*>  select  a.\*,v.id vid,v.value  from t\_attribute a  left join t\_attribute\_value v on a.id = v.attribute\_id  <where>  <if test=*"type != null"*>  type = #{type,jdbcType=TINYINT}  </if>    <if test=*"cid != null"*>  and cid = #{cid,jdbcType=BIGINT}  </if>  </where>  </select> |

## 添加商品



1 分析前端封装的数据格式



发现一个大对象里面封装了三个小对象，正好对应业务的三个对象。

2 后端需要封装一个java大对象，也就是反序列化过程！！！

内容补充：

**序列化 (Serialization)**是将对象的状态信息转换为可以存储或传输的形式的过程。在序列化期间，对象将其当前状态写入到临时或持久性存储区。以后，可以通过从存储区中读取或**反序列化**对象的状态，重新创建该对象。

1. 反向生成三个小对象

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <!DOCTYPE generatorConfiguration  PUBLIC "-//mybatis.org//DTD MyBatis Generator Configuration 1.0//EN"  "http://mybatis.org/dtd/mybatis-generator-config\_1\_0.dtd">  <generatorConfiguration>  <classPathEntry location="mysql-connector-java-8.0.15.jar" />  <context id="cjl-sc" targetRuntime="MyBatis3">  <commentGenerator>  <property name="suppressAllComments" value="true" />  </commentGenerator>  <jdbcConnection driverClass="com.mysql.cj.jdbc.Driver"  connectionURL="jdbc:mysql://localhost:3306/go9?characterEncoding=utf-8"  userId="root" password="111111">  </jdbcConnection>  <javaTypeResolver>  <property name="forceBigDecimals" value="false" />  </javaTypeResolver>  <javaModelGenerator targetPackage="com.go9.goods.pojo"  targetProject="src">  <property name="enableSubPackages" value="true" />  <property name="trimStrings" value="true" />  </javaModelGenerator>  <sqlMapGenerator targetPackage="com.go9.goods.mapper"  targetProject="src">  <property name="enableSubPackages" value="true" />  </sqlMapGenerator>  <javaClientGenerator type="XMLMAPPER"  targetPackage="com.go9.goods.mapper" targetProject="src">  <property name="enableSubPackages" value="true" />  </javaClientGenerator>    <!-- <table tableName="t\_brand" domainObjectName="Brand"  enableSelectByExample="false" enableDeleteByExample="false"  enableCountByExample="false" enableUpdateByExample="false"  selectByExampleQueryId="false">  <generatedKey column="id" sqlStatement="mysql" identity="true" />  <columnOverride column="is\_parent" property="yesParent" javaType="java.lang.Boolean" />  </table> -->    <!-- <table tableName="t\_attribute" domainObjectName="Attribute"  enableSelectByExample="false" enableDeleteByExample="false"  enableCountByExample="false" enableUpdateByExample="false"  selectByExampleQueryId="false">  <generatedKey column="id" sqlStatement="mysql" identity="true" />  </table> -->    <table tableName="t\_product" domainObjectName="Product"  enableSelectByExample="false" enableDeleteByExample="false"  enableCountByExample="false" enableUpdateByExample="false"  selectByExampleQueryId="false">  <generatedKey column="id" sqlStatement="mysql" identity="true" />  </table>  <table tableName="t\_product\_sku" domainObjectName="Sku"  enableSelectByExample="false" enableDeleteByExample="false"  enableCountByExample="false" enableUpdateByExample="false"  selectByExampleQueryId="false">  <generatedKey column="id" sqlStatement="mysql" identity="true" />  </table>  <table tableName="t\_product\_detail" domainObjectName="Detail"  enableSelectByExample="false" enableDeleteByExample="false"  enableCountByExample="false" enableUpdateByExample="false"  selectByExampleQueryId="false">  </table>  </context>  </generatorConfiguration> |

1. 创建大对象，聚合小对象，其中detail是字符串

|  |
| --- |
| **package** com.go9.goods.vo;  **import** java.util.List;  **import** com.go9.goods.pojo.Product;  **import** com.go9.goods.pojo.Sku;  **public** **class** ProductExt {  **private** Product product;  **private** String detail;  **private** List<Sku> skus;    **public** String getDetail() {  **return** detail;  }  **public** **void** setDetail(String detail) {  **this**.detail = detail;  }  **public** Product getProduct() {  **return** product;  }  **public** **void** setProduct(Product product) {  **this**.product = product;  }  **public** List<Sku> getSkus() {  **return** skus;  }  **public** **void** setSkus(List<Sku> skus) {  **this**.skus = skus;  }    } |

1. 创建controller类进行测试，反序列化对象是否成功

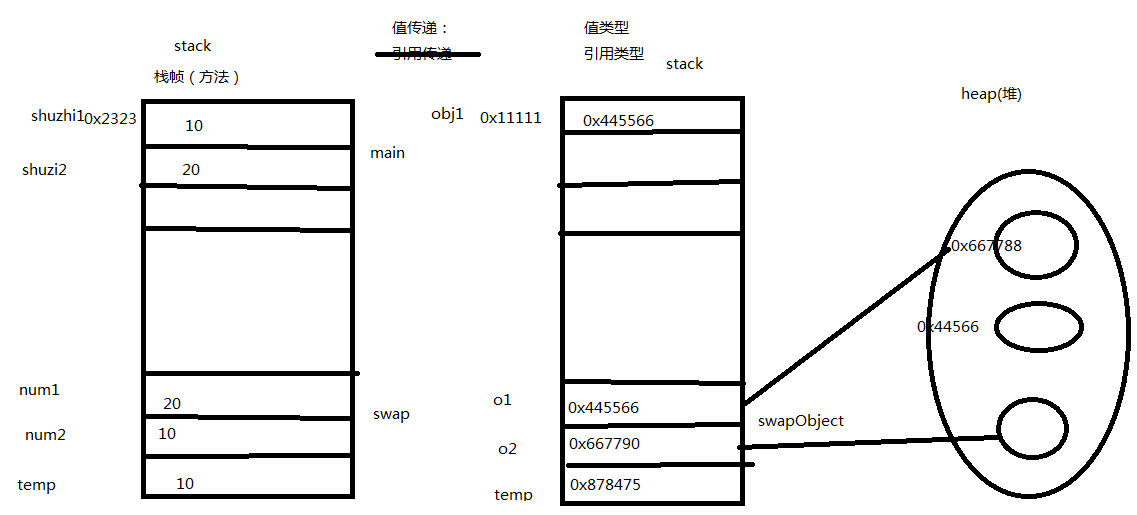
|  |
| --- |
| **package** com.go9.goods.controller;  **import** org.springframework.web.bind.annotation.PostMapping;  **import** org.springframework.web.bind.annotation.RequestBody;  **import** org.springframework.web.bind.annotation.RequestMapping;  **import** org.springframework.web.bind.annotation.RestController;  **import** com.go9.common.ServiceMessage;  **import** com.go9.goods.vo.ProductExt;  @RestController  @RequestMapping("/admin/product")  **public** **class** ProductController {    @PostMapping  **public** ServiceMessage add(@RequestBody ProductExt productExt) {  **return** ServiceMessage.*ok*(productExt);  }  } |

# 热加载技术

|  |
| --- |
| <!-- 热部署模块 -->  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-devtools</artifactId>  <optional>true</optional> <!-- 这个需要为 true 热部署才有效 -->  </dependency> |

一定要勾上**eclipse的 build automatically**

# 补充知识



# 思考题：

图片如何存储