# 1、数据架构

实际上，这个电商系统，他的这个整个数据库的表结构，可能是很复杂的。但是呢，业界也有一些已经完成的比较好的电商的数据库的建模。我不会说上来，带着大家把100%完美的数据库给建模好，这个是不可能的。到了后期，随着你的整个系统架构的演进，数据库的设计，可能也是不断的在变化的。

不要上来就想着万年以后的事情，所有的系统设计，都是一步一步来的。可能我们现在设计的这个电商的商品中心的表，评论中心的，订单中心的，可能有不完美的地方，可能存在一些缺陷，这个都没关系。

我们纯粹就是按照现在当前的这个阶段，就是v1.0内测版的阶段，没有任何的压力，现在的需求相对来说，还是简单一些，后面的很多复杂的业务需求还没上。所以这个数据库的表设计我们就按照现在的需求来就可以了。

然后，可能未来我们的数据库的设计，会不断的迭代和演进，所以说，大家如果感觉现在的这个表有些设计的问题，都很正常，不用太过于担心，到了一定的阶段，比如说你的系统架构本身在往2.0，3.0去演进，那么我们对应的数据库模型也会不断的演进，很正常的。

## 1.1 数据库ER模型

## 1.2 数据库逻辑模型

## 1.3 数据库物理模型

# 2、接口

## 2.1 属性管理

### 2.1.1 查询属性列表

（1）接口类

（2）接口名称

GET /api/v1.0/commodity/property/

（3）接口描述

查询属性列表

1. 输入

{

"offset": 0,

"size": 20,

"property\_name": "测试属性"

}

（5）输出

[

{

"id": 1,

"propertyName": "测试属性01",

"propertyDesc": "测试属性01",

"inputType": 1,

"inputValues": "值1,值2,值3",

"gmt\_create": "2017-01-01 10:00:00",

"gmt\_modified": "2017-01-01 10:05:00"

},{

"id": 1,

"propertyName": "测试属性01",

"propertyDesc": "测试属性01",

"inputType": 1,

"inputValues": "值1,值2,值3",

"gmt\_create": "2017-01-01 10:00:00",

"gmt\_modified": "2017-01-01 10:05:00"

}

]

### 2.1.2 新增属性

（1）接口类

（2）接口名称

POST /api/v1.0/commodity/property/

（3）接口描述

新增属性

1. 输入

{

"propertyName": "测试属性01",

"propertyDesc": "测试属性01",

"inputType": 1,

"propertyDesc": "黑色,白色"

}

（5）输出

{

"status": "success",

"message": "成功"

}

### 2.1.3 根据ID查询属性

（1）接口类

（2）接口名称

GET /api/v1.0/commodity/property/{id}

（3）接口描述

根据ID查询属性

1. 输入

属性ID

（5）输出

{

"id": 1,

"propertyName": "测试属性01",

"propertyDesc": "测试属性01",

"inputType": 1,

"inputValues": "值1,值2,值3",

"gmt\_create": "2017-01-01 10:00:00",

"gmt\_modified": "2017-01-01 10:05:00"

}

### 2.1.4 更新属性

（1）接口类

（2）接口名称

PUT /api/v1.0/commodity/property/{id}

（3）接口描述

新增属性

1. 输入

{

"id": 1,

"propertyDesc": "测试属性01"

}

（5）输出

{

"status": "success",

"message": "成功"

}

### 2.1.5 删除属性

（1）接口类

（2）接口名称

DELETE /api/v1.0/commodity/property/{id}

（3）接口描述

新增属性

1. 输入

睡醒ID

（5）输出

{

"status": "success",

"message": "成功"

}

## 2.2 类目管理

### 2.2.1 查询根类目列表

（1）接口类

（2）接口名称

GET /api/v1.0/commodity/category/root

（3）接口描述

查询根类目列表

1. 输入

无

（5）输出

[

{

"id": 1,

"categoryName": "测试属性01",

"categoryDesc": "测试属性01",

"leaf": 0,

"gmt\_create": "2017-01-01 10:00:00",

"gmt\_modified": "2017-01-01 10:05:00"

},{

"id": 1,

"categoryName": "测试属性01",

"categoryDesc": "测试属性01",

"leaf": 0,

"gmt\_create": "2017-01-01 10:00:00",

"gmt\_modified": "2017-01-01 10:05:00"

}

]

### 2.2.2 查询子类目列表

（1）接口类

（2）接口名称

GET /api/v1.0/commodity/category/children/{id}

（3）接口描述

查询子类目列表

1. 输入

类目ID

（5）输出

[

{

"id": 1,

"categoryName": "测试属性01",

"categoryDesc": "测试属性01",

"leaf": 0,

"gmt\_create": "2017-01-01 10:00:00",

"gmt\_modified": "2017-01-01 10:05:00"

},{

"id": 1,

"categoryName": "测试属性01",

"categoryDesc": "测试属性01",

"leaf": 0,

"gmt\_create": "2017-01-01 10:00:00",

"gmt\_modified": "2017-01-01 10:05:00"

}

]

### 2.2.3 新增类目

（1）接口类

（2）接口名称

POST /api/v1.0/commodity/category/

（3）接口描述

新增类目

1. 输入

{

"categoryName": "测试类目01",

"categoryDesc": "测试类目01",

"leaf": 0,

"parentId": 1,  
 "properties": [

{

"propertyId": 1,

"required": 1,

"propertyType": 1

},{

"propertyId": 2,

"required": 1,

"propertyType": 1

}

],

"propertyGroups": [

{

"propertyName": "属性分组01",

"properties": [

{

"propertyId": 1,

"required": 1,

"propertyType": 1

},{

"propertyId": 2,

"required": 1,

"propertyType": 1

}

]

},{

"propertyName": "属性分组02",

"properties": [

{

"propertyId": 1,

"required": 1,

"propertyType": 1

},{

"propertyId": 2,

"required": 1,

"propertyType": 1

}

]

}

]

}

（5）输出

{

"status": "success",

"message": "成功"

}

### 2.2.4 根据ID查询类目

（1）接口类

（2）接口名称

GET /api/v1.0/commodity/category/{id}

（3）接口描述

根据ID查询类目

1. 输入

类目ID

（5）输出

{

"id": 1,

"categoryName": "测试属性01",

"categoryDesc": "测试属性01",

"leaf": 0,

"gmt\_create": "2017-01-01 10:00:00",

"gmt\_modified": "2017-01-01 10:05:00",

"properties": [

{

"propertyId": 1,

"required": 1,

"propertyType": 1

},{

"propertyId": 2,

"required": 1,

"propertyType": 1

}

],

"propertyGroups": [

{

"propertyName": "属性分组01",

"properties": [

{

"propertyId": 1,

"required": 1,

"propertyType": 1

},{

"propertyId": 2,

"required": 1,

"propertyType": 1

}

]

},{

"propertyName": "属性分组02",

"properties": [

{

"propertyId": 1,

"required": 1,

"propertyType": 1

},{

"propertyId": 2,

"required": 1,

"propertyType": 1

}

]

}

]

}

### 2.2.5 更新类目

（1）接口类

（2）接口名称

PUT /api/v1.0/commodity/category/{id}

（3）接口描述

更新类目

1. 输入

{

"id": 1,

"categoryName": "测试类目01",

"categoryDesc": "测试类目01",

"leaf": 0,

"parentId": 1,  
 "properties": [

{

"propertyId": 1,

"required": 1,

"propertyType": 1

},{

"propertyId": 2,

"required": 1,

"propertyType": 1

}

],

"propertyGroups": [

{

"propertyName": "属性分组01",

"properties": [

{

"propertyId": 1,

"required": 1,

"propertyType": 1

},{

"propertyId": 2,

"required": 1,

"propertyType": 1

}

]

},{

"propertyName": "属性分组02",

"properties": [

{

"propertyId": 1,

"required": 1,

"propertyType": 1

},{

"propertyId": 2,

"required": 1,

"propertyType": 1

}

]

}

]

}

（5）输出

{

"status": "success",

"message": "成功"

}

### 2.2.6 删除类目

（1）接口类

（2）接口名称

DELETE /api/v1.0/commodity/category/{id}

（3）接口描述

删除类目

1. 输入

类目ID

（5）输出

{

"status": "success",

"message": "成功"

}

## 2.3 品牌管理

### 2.3.1 查询品牌列表

（1）接口类

（2）接口名称

GET /api/v1.0/commodity/brand/

（3）接口描述

查询品牌列表

1. 输入

{

"offset": 0,

"size": 20,

"brandChineseName": "中文名",

"brandEnglishName": "英文名",

"brandAliasName": "别名",

"enabled": 1

}

（5）输出

[

{

"id": 1,

"brandChineseName": "中文名",

"brandEnglishName": "英文名",

"brandAliasName": "别名",

"brandIntro": "品牌介绍",

"location": "品牌所在地",

"brandComment": "品牌说明备注",

"enabled": 1,

"gmtCreate": "2017-01-01 10:00:00",

"gmtModified": "2017-01-01 10:05:00"

},{

"id": 1,

"brandChineseName": "中文名",

"brandEnglishName": "英文名",

"brandAliasName": "别名",

"brandIntro": "品牌介绍",

"location": "品牌所在地",

"brandComment": "品牌说明备注",

"enabled": 1,

"gmtCreate": "2017-01-01 10:00:00",

"gmtModified": "2017-01-01 10:05:00"

}

]

### 2.3.2 新增品牌

（1）接口类

（2）接口名称

POST /api/v1.0/commodity/brand/

（3）接口描述

新增品牌s

1. 输入

{

"brandChineseName": "中文名",

"brandEnglishName": "英文名",

"brandAliasName": "别名",

"brandIntro": "品牌介绍",

"location": "品牌所在地",

"brandComment": "品牌说明备注",

"enabled": 1,

"gmtCreate": "2017-01-01 10:00:00",

"gmtModified": "2017-01-01 10:05:00"

}

再加上两张图片的上传，logo，brandAuthorizationVoucher

（5）输出

{

"status": "success",

"message": "成功"

}

### 2.3.3 根据ID查询品牌

（1）接口类

（2）接口名称

GET /api/v1.0/commodity/brand/{id}

（3）接口描述

根据ID查询品牌

（4）输入

品牌ID

（5）输出

{

"brandChineseName": 1,

"brandChineseName": "中文名",

"brandEnglishName": "英文名",

"brandAliasName": "别名",

"brandIntro": "品牌介绍",

"location": "品牌所在地",

"brandComment": "品牌说明备注",

"enabled": 1,

"gmtCreate": "2017-01-01 10:00:00",

"gmtModified": "2017-01-01 10:05:00"

}

### 2.3.4 根据ID下载logo图片

GET /api/v1.0/commodity/brand/logo/{id}，回返回图片流，给下载

### 2.3.5 根据ID下载品牌授权证明图片

GET /api/v1.0/commodity/brand/authorization/{id}，回返回图片流，给下载

### 2.3.6 编辑品牌

（1）接口类

（2）接口名称

PUT /api/v1.0/commodity/brand/{id}

（3）接口描述

编辑品牌

（4）输入

{

"id": 1,

"brandChineseName": "中文名",

"brandEnglishName": "英文名",

"brandAliasName": "别名",

"brandIntro": "品牌介绍",

"location": "品牌所在地",

"brandComment": "品牌说明备注",

"enabled": 1,

"gmtCreate": "2017-01-01 10:00:00",

"gmtModified": "2017-01-01 10:05:00"

}

再加上两张图片的上传，logo，brandAuthorizationVoucher

（5）输出

{

"status": "success",

"message": "成功"

}

### 2.3.7 删除品牌

（1）接口类

（2）接口名称

DELETE /api/v1.0/commodity/brand/{id}

（3）接口描述

删除品牌

（4）输入

品牌ID

（5）输出

{

"status": "success",

"message": "成功"

}

### 2.3.8 启用/停用品牌

（1）接口类

（2）接口名称

PUT /api/v1.0/commodity/brand/enabled/{id}

（3）接口描述

更新品牌的启用状态

（4）输入

{

"id": 1,

"enabled": 1

}

（5）输出

{

"status": "success",

"message": "成功"

}

## 2.4 商品管理

### 2.4.1 查询商品列表

（1）接口类

（2）接口名称

GET /api/v1.0/commodity/goods/

（3）接口描述

查询商品列表

1. 输入

{

"offset": 0,

"size": 20,

"categoryId": 1,

"brandId": 1,

"goodsCode": "商品编号",

"goodsName": "商品名称"

}

（5）输出

[

{

"id": 1,

"categoryId": 1,

"brandId": 1,

"goodsCode": "商品编号",

"goodsName": "商品名称",

"goodsSubName": "商品副名称",

"grossWeight": 43.6,

"goodsLength": 20.3,

"goodsWidth": 25.6,

"goodsHeight": 38.9,

"goodsStatus": 2,

"gmt\_create": "2017-01-01 10:00:00",

"gmt\_modified": "2017-01-01 10:05:00"

},{

"id": 1,

"categoryId": 1,

"brandId": 1,

"goodsCode": "商品编号",

"goodsName": "商品名称",

"goodsSubName": "商品副名称",

"grossWeight": 43.6,

"goodsLength": 20.3,

"goodsWidth": 25.6,

"goodsHeight": 38.9,

"goodsStatus": 2,

"gmt\_create": "2017-01-01 10:00:00",

"gmt\_modified": "2017-01-01 10:05:00"

}

]

### 2.4.2 新增商品

（1）接口类

（2）接口名称

POST /api/v1.0/commodity/goods/

（3）接口描述

新增类目

1. 输入

{

"categoryId": 1,

"brandId": 1,

"goodsCode": "商品编号",

"goodsName": "商品名称",

"goodsSubName": "商品副名称",

"grossWeight": 43.6,

"goodsLength": 20.3,

"goodsWidth": 25.6,

"goodsHeight": 38.9,

"serviceGuarantees": “服务保证”

“packageList”: “包装清单”,

“freightTemplateId”: 1,

“goodsDesc”: “一大段详情内容文本”,  
 "properties": [

{

"categoryPropertyRelationshipId": 1,

"propertyValue": 1

},{

"categoryPropertyRelationshipId": 1,

"propertyValue": 1

}

],

"propertyGroups": [

{

"PropertyGroupRelationshipId": 1,

"propertyValue": 1

},{

"propertyGroupRelationshipId": 1,

"propertyValue": 1

}

],

"goodsSku": [

{

"skuCode": 1,

"purcahsePrice": 1,

“salePrice”: 1,

“discountPrice”: 1

},{

"skuCode": 2s,

"purcahsePrice": 1,

“salePrice”: 1,

“discountPrice”: 1

}

],

}

会上传一些图片列表，goodsPictures

会上传一些详情内容里面的各个图片

（5）输出

{

"status": "success",

"message": "成功"

}

### 2.2.4 根据ID查询商品

（1）接口类

（2）接口名称

GET /api/v1.0/commodity/goods/{id}

（3）接口描述

根据ID查询类目

1. 输入

类目ID

（5）输出

（1）接口类

（2）接口名称

GET /api/v1.0/commodity/goods/

（3）接口描述

查询商品列表

1. 输入

{

"offset": 0,

"size": 20,

"categoryId": 1,

"brandId": 1,

"goodsCode": "商品编号",

"goodsName": "商品名称"

}

（5）输出

{

"id": 1,

"categoryId": 1,

"brandId": 1,

"goodsCode": "商品编号",

"goodsName": "商品名称",

"goodsSubName": "商品副名称",

"grossWeight": 43.6,

"goodsLength": 20.3,

"goodsWidth": 25.6,

"goodsHeight": 38.9,

"goodsStatus": 2,

"gmt\_create": "2017-01-01 10:00:00",

"gmt\_modified": "2017-01-01 10:05:00",

“goodsDesc”: “一大段详情内容文本”,  
 "properties": [

{

"categoryPropertyRelationshipId": 1,

"propertyValue": 1

},{

"categoryPropertyRelationshipId": 1,

"propertyValue": 1

}

],

"propertyGroups": [

{

"PropertyGroupRelationshipId": 1,

"propertyValue": 1

},{

"propertyGroupRelationshipId": 1,

"propertyValue": 1

}

],

"goodsSku": [

{

"skuCode": 1,

"purcahsePrice": 1,

“salePrice”: 1,

“discountPrice”: 1

},{

"skuCode": 2s,

"purcahsePrice": 1,

“salePrice”: 1,

“discountPrice”: 1

}

],

}

### 2.2.5 更新商品

（1）接口类

（2）接口名称

PUT /api/v1.0/commodity/goods/{id}

（3）接口描述

更新商品

1. 输入

{

“id”: 1,

"categoryId": 1,

"brandId": 1,

"goodsCode": "商品编号",

"goodsName": "商品名称",

"goodsSubName": "商品副名称",

"grossWeight": 43.6,

"goodsLength": 20.3,

"goodsWidth": 25.6,

"goodsHeight": 38.9,

"serviceGuarantees": “服务保证”

“packageList”: “包装清单”,

“freightTemplateId”: 1,

“goodsDesc”: “一大段详情内容文本”,  
 "properties": [

{

"categoryPropertyRelationshipId": 1,

"propertyValue": 1

},{

"categoryPropertyRelationshipId": 1,

"propertyValue": 1

}

],

"propertyGroups": [

{

"PropertyGroupRelationshipId": 1,

"propertyValue": 1

},{

"propertyGroupRelationshipId": 1,

"propertyValue": 1

}

],

"goodsSku": [

{

"skuCode": 1,

"purcahsePrice": 1,

“salePrice”: 1,

“discountPrice”: 1

},{

"skuCode": 2s,

"purcahsePrice": 1,

“salePrice”: 1,

“discountPrice”: 1

}

],

}

会上传一些图片列表，goodsPictures

会上传一些详情内容里面的各个图片

（5）输出

{

"status": "success",

"message": "成功"

}

### 2.2.6 删除商品

（1）接口类

（2）接口名称

DELETE /api/v1.0/commodity/goods/{id}

（3）接口描述

删除商品

1. 输入

类目ID

（5）输出

{

"status": "success",

"message": "成功"

}

### 2.2.7 审核商品

（1）接口类

（2）接口名称

PUT /api/v1.0/commodity/goods/approve/{id}

（3）接口描述

更新品牌的启用状态

（4）输入

{

"id": 1,

"approved": 1 // 1是通过，0是未通过

}

（5）输出

{

"status": "success",

"message": "成功"

}

### 2.2.8 上下架商品

（1）接口类

（2）接口名称

PUT /api/v1.0/commodity/goods/shelves/{id}

（3）接口描述

上下架商品

（4）输入

{

"id": 1,

"shelvesStatus": 1 // 1是上架，0是下架

}

（5）输出

{

"status": "success",

"message": "成功"

}

## 2.5 商品查询

### 2.5.1 搜索商品

（1）接口类

（2）接口名称

GET /api/v1.0/commodity/goods/search?keyword=iphone

（3）接口描述

搜索商品

1. 输入

搜索关键词

（5）输出

[

{

"goodsId": 1,

“goodsSkuId”: 1,

"goodsPictures": [

“<img src=’/api/v1.0/commodity/goods/picture/1’ />”,

“<img src=’/api/v1.0/commodity/goods/picture/2’ />”,

“<img src=’/api/v1.0/commodity/goods/picture/3’ />”

]

"salePrice": 8388.00,

"goodsName": "商品名称"

},{

"goodsId": 1,

“goodsSkuId”: 1,

"goodsPictures": [

“<img src=’/api/v1.0/commodity/goods/picture/1’ />”,

“<img src=’/api/v1.0/commodity/goods/picture/2’ />”,

“<img src=’/api/v1.0/commodity/goods/picture/3’ />”

]

"salePrice": 8388.00,

"goodsName": "商品名称"

}

]

### 2.5.2 品类筛选

（1）接口类

（2）接口名称

GET /api/v1.0/commodity/goods/category/{categoryId}

（3）接口描述

根据品类筛选商品

1. 输入

品类ID

（5）输出

[

{

"goodsId": 1,

“goodsSkuId”: 1,

"goodsPictures": [

“<img src=’/api/v1.0/commodity/goods/picture/1’ />”,

“<img src=’/api/v1.0/commodity/goods/picture/2’ />”,

“<img src=’/api/v1.0/commodity/goods/picture/3’ />”

]

"salePrice": 8388.00,

"goodsName": "商品名称"

},{

"goodsId": 1,

“goodsSkuId”: 1,

"goodsPictures": [

“<img src=’/api/v1.0/commodity/goods/picture/1’ />”,

“<img src=’/api/v1.0/commodity/goods/picture/2’ />”,

“<img src=’/api/v1.0/commodity/goods/picture/3’ />”

]

"salePrice": 8388.00,

"goodsName": "商品名称"

}

]

### 2.5.3 商品列表

（1）接口类

（2）接口名称

GET /api/v1.0/commodity/goods/list?keyword=iphone&offset=0&size=20

GET /api/v1.0/commodity/goods/list?categoryId=1&offset=0&size=20

（3）接口描述

查询商品列表

1. 输入

搜索词或者是品类ID

（5）输出

[

{

"goodsId": 1,

“goodsSkuId”: 1,

"goodsPictures": [

“<img src=’/api/v1.0/commodity/goods/picture/1’ />”,

“<img src=’/api/v1.0/commodity/goods/picture/2’ />”,

“<img src=’/api/v1.0/commodity/goods/picture/3’ />”

]

"salePrice": 8388.00,

"goodsName": "商品名称"

},{

"goodsId": 1,

“goodsSkuId”: 1,

"goodsPictures": [

“<img src=’/api/v1.0/commodity/goods/picture/1’ />”,

“<img src=’/api/v1.0/commodity/goods/picture/2’ />”,

“<img src=’/api/v1.0/commodity/goods/picture/3’ />”

]

"salePrice": 8388.00,

"goodsName": "商品名称"

}

]

### 2.5.4 商品详情页

（1）接口类

（2）接口名称

GET /api/v1.0/commodity/goods/detail/{id}

（3）接口描述

查询商品详情页

1. 输入

商品ID

（5）输出

{

"id": 1,

"goodsPictures": [

“<img src=’/api/v1.0/commodity/goods/picture/1’ />”,

“<img src=’/api/v1.0/commodity/goods/picture/2’ />”,

“<img src=’/api/v1.0/commodity/goods/picture/3’ />”

],

"categoryId": 1,

"brandId": 1,

"goodsCode": "商品编号",

"goodsName": "商品名称",

"goodsSubName": "商品副名称",

“promotions”: [

{

“id”: 1,

“promotionType”: 1,

“promotionName”: “满100元减5元”

},

{

“id”: 2,

“promotionType”: 2,

“promotionName”: “买手机送耳机”

}

],  
 "saleProperties": [

{

"propertyId": 1,

“propertyName”: “机身颜色”,

"propertyValue": “银色,深空灰色”

},{

"propertyId": 2,

“propertyName”: “机身容量”

"propertyValue": “64G,256G”

}

],

“goodsDesc”: “文本+<img src=’/api/v1.0/commodity/goods/desc/picture/1’>+文本+<img src=’’>”,

"propertyGroups": [

{

“propertyGroupName”: “主体”,

“properties”: {

"propertyName": “上市年份”,

"propertyValue": “2017年”

}

},{

“propertyGroupName”: “主体”,

“properties”: {

"propertyName": “上市年份”,

"propertyValue": “2017年”

}

}

],

“packageList”: “”,

“serviceGuarantees”: “”,

}

## 2.6 系统内部接口

### 2.6.1 根据ID查询商品sku信息

GoodsService

GoodsSkuDTO getGoodsSkuById(Long goodsSkuId)

# 3、开发架构

## 3.1 实现类图

[商品中心实现类图](https://www.processon.com/diagraming/5a7d4775e4b0615ac053ed39)

## 3.2 包设计

com.zhss.eshop.commdoity.mapper

com.zhss.eshop.commdoity.dao

com.zhss.eshop.commdoity.service

com.zhss.eshop.commdoity.web

com.zhss.eshop.commdoity.vo

com.zhss.eshop.commdoity.dto

com.zhss.eshop.commdoity.do

com.zhss.eshop.commdoity.state

com.zhss.eshop.commdoity.visitor

com.zhss.eshop.commdoity.composite

# 4、运行流程

## 4.1 属性管理

### 4.1.1 查询属性列表

按照筛选条件，以分页的形式来查询属性列表即可

### 4.1.2 新增属性

将页面上传递过来的属性信息保存到数据库的表中即可

### 4.1.3 根据ID查询属性

根据传递过来的ID将属性的信息查询出来即可

### 4.1.4 更新属性

将页面上传递过来的属性信息保存到数据库的表中即可，这里只有属性的说明备注是可以修改的，其他的字段不让修改。

### 4.1.5 删除属性

删除之前需要先检查一下，如果有类目或者是属性组还关联着这个属性，那么是不能够将这个属性给删除的。如果没有任何类目或者是属性分组还关联这个属性，那么就可以删除。

## 4.2 类目管理

### 4.2.1 查询根类目列表

将所有parent\_id字段为NULL的类目给查询出来，这就是根类目列表

### 4.2.2 查询子类目列表

根据parent\_id查询某个父类目下的子类目的列表

### 4.2.3 新增类目

将页面上传递过来的数据保存到数据库中即可，但是这里的数据较为复杂的，需要分成以下几个步骤来执行，因为类目是关联着属性和属性分组的：

1. 保存类目的基本信息
2. 对于那些直接将类目关联的属性，将关联的数据保存到类目和属性关联关系表中即可
3. 对于那些类目关联的属性分组，首先将类目对应的属性分组保存到属性分组表中
4. 然后将属性分组跟属性的关联关系，保存到对应的关联关系表中即可

### 4.2.4 根据ID查询类目

（1）查询类目的基本信息

（2）查询类目直接关联的属性信息

（3）查询类目关联的属性分组以及属性分组关联的属性信息

### 4.2.5 更新类目

（1）如果类目被已经上架的商品关联了，那么就不能随意更新类目的信息了；但是如果关联类目的商品不是上架状态，就可以更新类目

（2）更新类目的基本信息

（3）将类目直接关联的属性的关联关系，给删除

（4）重新插入类目和最新的关联的属性的关联关系到数据库中

（5）将类目关联的属性分组与属性的关联关系给删除

（6）删除类目所有的属性分组

（7）重新插入类目的属性分组，以及属性分组与属性的关联关系

### 4.2.6 删除类目

（1）如果类目已经被商品给关联了，就不能删除类目

（2）删除类目与直接关联的属性之间的关联关系

（3）删除类目关联的属性分组与属性之间的关联关系

（4）删除类目关联的属性分组

（5）删除类目自己本身的信息就可以了

上面的步骤仅仅是删除一个类目，但是实际上删除一个父类目的时候，是要将这个父类目名下的所有子类目都按照上述的这个思路给删除的

所以这里采用组合模式+访问者模式的设计模式

1. 首先对类目树进行是否有任何一个类目是被商品给关联着的，其实就是可以传入一个visitor，由这个visitor从当前类目开始，无限递子类目，对每个子类目都判断一下是否被某个商品在关联着，只要当前要删除的这个类目有任何一个子类目，是被商品给关联了，那么就不让删除了
2. 如果上面的visitor判断发现ok，可以删除，那么执行删除的逻辑
3. 再传入一个visitor，该visitor也是动态查询每一个类目的子类目，然后对这个类目去执行上面的整套删除逻辑，直到将这个类目树给完全删除掉

## 4.3 品牌管理

### 4.3.1 查询品牌列表

按照筛选条件来分页查询数据即可

### 4.3.2 新增品牌

将页面传递过来的数据保存到数据库即可，可能稍微特殊一点点的就是那个图片的上传，这个接收到页面过来的二进制流数据以后直接存入数据库即可

### 4.3.3 根据ID查询品牌

将品牌的数据查询出来，支持品牌的两个图片的下载即可

### 4.3.4 编辑品牌

对页面上传递过来的数据保存到数据库即可，两个图片也是一样的

### 4.3.5 删除品牌

如果有商品关联着这个品牌，不允许删除品牌，否则的话，就是直接删除品牌自己即可

### 4.3.6 启用/停用品牌

如果有商品关联着这个品牌，不允许启用/停用品牌，否则的话，就是直接更新品牌的is\_enabled字段即可

## 4.4 商品管理

### 4.4.1 查询商品列表

按照筛选条件来筛选，同时分页查询即可

这个查询的时候，有一个特殊的逻辑，实际上是按照spus粒度去查询出来的一个一个的商品

### 4.4.2 新建商品

（1）保存商品自己的基本信息

（2）保存商品的图片列表

（3）保存商品的商品详情介绍内容

（4）保存商品对类目直接关联的属性的值是什么，将商品关联的那个叶子类目上面所有层级的父类目的直接关联属性需要全部拿出来，给这个商品来填写的

（5）保存商品对类目关联的属性分组对应的属性的值是什么，将商品关联的那个叶子类目上面所有层级的父类目的关联的那些属性分组都拿出来，给这个商品来填写

（6）保存商品对应的每个sku的数据，包括了sku对应的每个销售属性的值

~~（7）保存商品的sku的库存数据到库存中心去~~

（8）新建完商品之后，商品的状态是：待审核

商品这里比较有技术含量的，就是这个状态的处理以及状态的这个流转，这块我们就可以考虑用这个状态模式。如果我们不用状态模式，直接就是在每个service的增删改查或者其他的一些方法中来处理这个状态模式的一些流转逻辑，会导致一个商品数据的状态相关的操作以及流转的逻辑分散在了很多个地方和方法中。

未来如果要对商品的状态流转的逻辑进行维护，比如调整状态流转的策略，可能会导致比较难以维护，因为状态相关的逻辑太分散了。

所以如果采取状态模式的思路，也就是说，在一个地方维护这个商品数据所有的状态相关的流转的逻辑，然后就可以将状态的相关操作集中维护和管理，对后续的维护是很有好处的。

调用状态管理组件的一个方法，那个方法就是支持新建商品的状态维护，此时这个方法可以创建一个ApprovingState状态组件，执行里面的方法，完成待审核状态的一些操作，比如说最简单的就是将商品的状态给设置成待审核。

### 4.4.3 根据ID查询商品

（1）查询商品自己的基本信息

（2）查询商品的图片列表

（3）查询商品的详情内容介绍

（4）查询商品对类目直接关联的属性的值

（5）查询商品对类目关联的每个属性分组的值

（6）查询商品的每个sku对应的数据，包括销售属性的值，包括价格的数据~~，同时从库存中心查询这个sku对应的库存数据~~

### 4.4.4 编辑商品

（1）保存商品自己的基本信息

（2）保存商品的图片列表

（3）保存商品的商品详情介绍内容

（4）保存商品对类目直接关联的属性的值是什么

（5）保存商品对类目关联的属性分组对应的属性的值是什么

（6）保存商品对应的每个sku的数据，包括了sku对应的每个销售属性的值

（7）保存商品的sku的库存数据到库存中心去

（8）在商品编辑之前，总是要考虑一下，当前是否能够编辑这个商品。只有待审核、审核未通过两种状态下，是可以对商品进行编辑的。

编辑商品之前，可以走一下状态管理组件，调用一个准备编辑的方法，状态管理组件根据当前商品所处的这个状态，根据这个状态创建一个对应的state组件，然后调用这个组件中的某个方法，判断一下当前状态能否执行编辑操作

去执行编辑

创建一个ApprovingState的一个状态组件，完成状态流转到待审核的这么一个操作

### 4.4.5 审核商品

（1）根据ID查询商品，显示出商品所有的数据

（2）点击审核通过/未通过

（3）在后台处理的时候，实际上就还是要通过一个唯一的状态管理组件去管理这次状态的流转，商品的状态会流转到待上架，或者是审核未通过

（4）在审核之前，还要做一个校验，就是说只有待审核状态的商品，是可以进行审核的

在审核之前，先根据当前的状态创建一个state组件，然后调用一个准备审核的方法，判断一下当前这个状态能否执行审核操作，如果可以的话，就执行审核操作，会根据审核的结果，创建不同的state组件，PuttingOnShelvesState，通过这个state完成状态流转到待上架的这么个一个操作；如果审核不通过，ApproveRejectedState，通过这个state完成状态流转到审核不通过的状态

### 4.4.6 上下架商品

（1）只有是待上架状态的商品，可以执行上架操作；只有是已上架状态的商品可以执行下架操作

（2）就到一个唯一的状态流转的管理组件中去，负责去将商品的状态进行流转

思路同上，先创建一个当前state组件，判断能否执行某个操作，如果可以就执行，然后创建下一个状态的state组件，完成状态流转的逻辑

### 4.4.7 删除商品

（1）只有待审核、审核未通过、待上架，这三种状态的商品是可以进行删除的；商品必须是没有被任何采购单给关联的时候，我们才可以删除这个

（2）删除商品的图片列表

（3）删除商品的详情内容数据

（4）删除商品对属性关联的值

（5）删除商品关联的sku的数据，包括了sku的属性值，以及sku的库存数据

（6）删除商品自己的数据

创建一个当前的状态组件，调用一下准备删除的方法，判断能否执行删除操作

## 4.5 商品查询

### 4.5.1 搜索商品

用全模糊匹配的方式，来搜索对应的商品的名称，然后支持分页，搜索完以后直接返回第一页的数据即可。

这里要强调的一点是，在这一块，都是针对sku级别去搜索的，所以需要将spu与sku都关联在一起，然后根据商品名称去搜索，返回的都是sku级别的数据。

### 4.2.5 品类筛选

我们的商品是只能挂在叶子类目下，不能挂在父类目下面的，所以我们要求的是首页进行品类筛选的时候，只能点击这个叶子类目。

后台直接根据叶子类目筛选出来对应的商品，然后支持分页查询即可。

### 4.2.6 查询商品列表

搜索商品或者是品类筛选，都会进入一个商品列表，在这个列表里，分不同的情况来支持查询：

1. 如果是搜索之后进入的商品列表页面，仅仅能支持分页查询，不支持条件筛选。每次翻页的时候，都会将这个搜索词给传递过来，查的是那个搜索词下的第几页。
2. 如果是品类筛选之后进入的商品列表页面，仅仅支持分页查询，不支持条件筛选，每次翻页的时候，都会将这个品类的ID传递过来，查的都是那个品类下的第几页

### 4.2.7 商品详情页

点击某个商品进入商品详情页，直接根据商品ID将商品详情页中的所有数据给展示出来

# 5、测试用例设计

## 5.1 单元测试用例设计

要求的是所有的包下的所有的类的每一个public方法都要执行单元测试

## 5.2 冒烟测试用例设计

要求的是对每个controller接口都要执行冒烟测试

# 6、日志设计

要求的是对每个类都在自己这一层try catch异常，在catch异常的时候打印出完整的异常信息