

**实 训 报 告**

**（信息工程学院）**

|  |  |
| --- | --- |
| **学期：** | 2023-2024学年第一学期 |
| **课程名称：** | 软件开发与网络实践 |
| **班级：** | 计科一班 |
| **学号：** | 12021242333 |
| **姓名：** | 梁天朗 |
| **指导老师：** | 马建乐 |

**2023年12月30日**

**《****软件开发与网络实践》成绩评定表**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 题目 | | **进销存系统数据接口设计、实现与应用** | | | |
| 实训内容简介：  （1）分别利用Postgresql数据库、MongoDB数据库和Neo4j数据库按要求设计、实现数据API接口（接口文档另附），要求接口请求和返回的数据结构对接口文档一致, 有必要的安全认证机制和说明文档，使用的编程语言和开发框架不限。  （2）解析给出的数据，将数据分别存入所设计的关系型数据库、文档数据库和图数据库。  （3）基于微服务结构，通过调用所设计的接口，实现一个百货公司进销存管理管理系统，实现产品信息、订单信息、员工信息的数据查询和展示，常用接口应提供实现数据缓存机制，减少不必要的数据库查询，提高查询效率。 | | | | | |
| **分组成员工作内容（本人标\*）** | | | | | | |
| **序号** | **角色** | | **学号/姓名** | **具体任务描述** | **工作量占比** | |
|  | 组长 | | 12021242333/梁天朗 | Pg模块，产品接口、订单接口实现。处理各种杂事 | 25% | |
|  | 组员 | | 12021242285/马涛 | 登录、主页、产品分页查询、订单分页查询页面 | 15% | |
|  | 组员 | | 12021242257/王子羽 | Web模块API接口，定义好model、service | 15% | |
|  | 组员 | | 12021242261/李强 | Pg模块，除了产品接口、订单接口，其它接口的实现 | 15% | |
|  | 组员 | | 12021242342/杨轩宇 | Mongo模块，除了产品接口、订单接口，其它接口的实现。完成Apifox | 15% | |
|  | 组员 | | 12021242317/程熙鹏 | Mongo模块，产品接口、订单接口实现。 | 15% | |

文档打分表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **序号** | **评分点** | **要求** | **分值** | **得分** |
| 1 | 接口设计 | 对各接口URL定义、传入参数格式、返回结果数据结构、功能描述有详细、准确的描述，参数和返回结果要有示例值进行说明。建议使用Apifox工具生成。 | 40 |  |
| 2 | 接口实现 | 基于Postgresql关系型数据库接口的实现，实现接口所使用的关键SQL语句和编程实现的关键代码 | 15 |  |
| 3 | 基于Mongo文档数据库接口的实现，有详细的文档结构设计与描述、实现接口所使用Mongo关键查询语句和编程实现的关键代码 | 15 |  |
| 4 | 接口应用 | 各接口的调用和应用效果展示 | 10 |  |
| 5 | 格式规范 | 文档结构（目录、标题、编号等）清晰、图表工作使用正确、无错字 | 10 |  |
| 6 | 总结内容 | 对大作业的完成情况、工作内容描述、课程学习心得，根据内容酌情给分 | 10 |  |
| 合计 | | | **100** |  |

功能打分表

| **序号** | **考查内容** | | **具体要求** | **分值** | **得分** |
| --- | --- | --- | --- | --- | --- |
| 1 | 接口的设计与实现 | 接口的关系型数据库设计与实现 | 接口1：POST /products | **6** |  |
| 2 | 接口2：GET /product/{id} | **6** |  |
| 3 | 接口4：POST /orders | **6** |  |
| 4 | 接口8：GET /order/{id} | **6** |  |
| 5 | 接口9：POST /employees | **6** |  |
| 6 | 接口10：POST /customers | **2** |  |
| 7 | 接口11：POST /shippers | **2** |  |
| 8 | 接口12：POST /suppliers | **2** |  |
| 9 | 接口13：GET /categories/level/{value} | **4** |  |
| 10 | 接口15：GET /authors | **5** |  |
| 11 | 接口的Mongo文档数据库设计与实现 | 接口1：POST /products | **3** |  |
| 12 | 接口2：GET /product/{id} | **3** |  |
| 13 | 接口4：POST /orders | **2** |  |
| 14 | 接口8：GET /order/{id} | **3** |  |
| 15 | 接口9：POST /employees | **2** |  |
| 16 | 接口10：POST /customers | **1** |  |
| 17 | 接口11：POST /shippers | **1** |  |
| 18 | 接口12：POST /suppliers | **1** |  |
| 19 | 接口13：GET /categories/level/{value} | **2** |  |
| 20 | 接口15：GET /authors | **5** |  |
| 21 |  | 接口的安全机制 | 基本要求：使用令牌等机制实现接口的鉴权，要求结合Redis实现。 | **2** |  |
| 23 | 接口的应用 | 产品接口的应用 | 编写前端页面，调用接口1和接口13，实现简单的产品查询和展示功能 | **10** |  |
|  | 项目部署 | 项目打包部署 | 将项目打包为Docker镜像，上传镜像到阿里云镜像仓库，与数据库镜像共同部署，可以使用docker-compose编排。 | **5** |  |
| 24 | 附加分 | 完成度 | 根据工作量、完整度、扩展能力，如其他接口的应用，酌情给分 | **15** |  |
| 总分 | | | | **100** | **0** |

目录

[文档打分表 3](#_Toc14924)

[功能打分表 I](#_Toc12521)

[目录 III](#_Toc27394)

[实训目的 1](#_Toc10764)

[1实训内容与要求 2](#_Toc810)

[1.1实训内容 2](#_Toc31482)

[1.2基本要求 2](#_Toc20159)

[1.2.1功能需求 2](#_Toc5059)

[1.2.2设计要求 2](#_Toc28137)

[1.2.3成果要求 3](#_Toc25940)

[2开发环境与开发技术 4](#_Toc10546)

[2.1开发环境 4](#_Toc27691)

[2.1.1运行环境 4](#_Toc4275)

[2.1.2开发工具 4](#_Toc20275)

[2.1.3Web框架 4](#_Toc29209)

[2.1.4前端技术 4](#_Toc1574)

[2.1.5数据库 5](#_Toc19440)

[2.2项目文件结构 5](#_Toc19507)

[3数据接口设计 9](#_Toc4148)

[3.1接口设计 9](#_Toc2248)

[3.1.1产品分组 9](#_Toc22644)

[3.1.2订单分组 23](#_Toc26538)

[3.1.3员工分组 42](#_Toc14)

[3.1.4客户分组 44](#_Toc25276)

[3.1.5运货商分组 48](#_Toc32080)

[3.1.6供应商分组 50](#_Toc21293)

[3.1.7产品分类分组 53](#_Toc22648)

[3.1.7作者分类分组 56](#_Toc3299)

[4数据接口实现 58](#_Toc24150)

[4.1根据产品id查询产品接口 58](#_Toc29493)

[4.1.1关系数据库实现 58](#_Toc24639)

[4.1.2Mongo数据库实现 59](#_Toc27229)

[4.2产品查询接口 62](#_Toc7109)

[4.2.1关系数据库实现 62](#_Toc3575)

[4.2.2Mongo数据库实现 65](#_Toc784)

[4.3根据供应商id及其它条件查询产品接口 69](#_Toc25577)

[4.3.1关系数据库实现 69](#_Toc22373)

[4.3.2Mongo数据库实现 71](#_Toc30603)

[4.4查询产品库存接口 75](#_Toc12604)

[4.4.1关系数据库实现 75](#_Toc22511)

[4.4.2Mongo数据库实现 77](#_Toc16880)

[4.5根据订单id查询接口 80](#_Toc7504)

[4.5.1关系数据库实现 80](#_Toc25588)

[4.5.2Mongo数据库实现 82](#_Toc22255)

[4.6订单查询接口 88](#_Toc30134)

[4.6.1关系数据库实现 88](#_Toc3309)

[4.6.2Mongo数据库实现 91](#_Toc7317)

[4.7根据跟单人员编号查询订单接口 94](#_Toc7651)

[4.7.1关系数据库实现 94](#_Toc14490)

[4.7.2Mongo数据库实现 97](#_Toc15939)

[4.8根据客户id查询订单接口 100](#_Toc22220)

[4.8.1关系数据库实现 101](#_Toc20431)

[4.8.2Mongo数据库实现 104](#_Toc8029)

[4.9根据运货商id查询订单接口 107](#_Toc14753)

[4.9.1关系数据库实现 107](#_Toc26110)

[4.9.2Mongo数据库实现 110](#_Toc28058)

[4.10员工查询接口 113](#_Toc30239)

[4.10.1关系数据库实现 113](#_Toc28215)

[4.10.2Mongo数据库实现 116](#_Toc16871)

[4.11客户查询接口 118](#_Toc28)

[4.11.1关系数据库实现 118](#_Toc4642)

[4.11.2Mongo数据库实现 120](#_Toc19662)

[4.12运货商查询接口 122](#_Toc1665)

[4.12.1关系数据库实现 122](#_Toc10360)

[4.12.2Mongo数据库实现 124](#_Toc28882)

[查询结果与pg数据库的相同。 125](#_Toc31414)

[4.13供应商查询接口 125](#_Toc11047)

[4.13.1关系数据库实现 125](#_Toc2732)

[4.13.2Mongo数据库实现 127](#_Toc18434)

[查询结果与pg数据库的相同。 129](#_Toc12543)

[4.14根据产品等级查询产品接口 129](#_Toc13562)

[4.14.1关系数据库实现 129](#_Toc18022)

[4.14.2Mongo数据库实现 130](#_Toc27513)

[4.15接口安全认证 132](#_Toc7148)

[4.15.1安全认证流程 133](#_Toc26756)

[4.15.2安全认证实现 133](#_Toc29210)

[5数据接口应用 134](#_Toc11181)

[5.1主页面 134](#_Toc20132)

[5.1.1使用目的 134](#_Toc16027)

[5.1.2前端主要代码 134](#_Toc19537)

[5.1.3应用效果 136](#_Toc4558)

[5.2产品查询接口 137](#_Toc7979)

[5.2.1使用目的 137](#_Toc8328)

[5.2.3应用效果 144](#_Toc5956)

[5.3订单查询接口 145](#_Toc7430)

[5.3.1使用目的 145](#_Toc32401)

[5.3.2接口调用 145](#_Toc12137)

[5.3.3应用效果 155](#_Toc21161)

[6总结与展望 156](#_Toc25856)

[6.1课程总结（个人） 156](#_Toc8553)

[6.1.1个人完成工作总结 156](#_Toc6868)

[6.1.2心得体会 156](#_Toc18381)

[6.2展望 156](#_Toc17831)

[6.2.1对技术方法的评价 156](#_Toc3197)

[6.2.2 经验与教训 158](#_Toc19133)

实训目的

计算机科学与技术专业和软件工程专业是实践性很强的学科，要求学生具备扎实的理论基础，更要求具备相当的实践开发能力，能解决实际工作中的需求和问题。

本实训应用传统的关系型数据库技术，并结合MongoDB、Neoc4j、Redis等NoSQL数据库，开发实际的应用，对学生对常用开发工具和技术的训练，同时掌握并实践软件项目设计规范及其开发流程。

通过实训，使学生培养学生的动手能力、良好编程规范、编程方法，团队合作与沟通能力、自主学习与创新能力，为今后应用数据库和从事软件开发与测试工作奠定坚实的基础。同时，也增强了学生的团队协作意识，了解软件开发的思考方法和主要流程。

# **1**实训内容与要求

## 1.1实训内容

（1）分别利用Postgresql数据库、MongoDB数据库和Neo4j数据库按要求设计、实现数据API接口（接口文档另附），要求接口请求和返回的数据结构对接口文档一致, 有必要的安全认证机制和说明文档，使用的编程语言和开发框架不限。

（2）解析给出的数据，将数据分别存入所设计的关系型数据库、文档数据库和图数据库。

（3）基于微服务结构，通过调用所设计的接口，实现一个百货公司进销存管理管理系统，实现产品信息、订单信息、员工信息的数据查询和展示，常用接口应提供实现数据缓存机制，减少不必要的数据库查询，提高查询效率。

## 1.2基本要求

### **1.2.1**功能需求

实现一个百货公司后台管理系统，系统主要功能包括但不限于，具体功能如下：查询产品、订单、员工、客户、运货商、供应商、产品分类、进货记录。

### **1.2.2**设计要求

* 代码要做好版本管理，建议使用GIT、Github或码云等公共代码仓库；
* 前端界面尽可能采用响应式设计，适应PC和移动终端，可以考虑使用Semantic－UI或Bootstrap框架进行快速开发；
* 编码中的各种命名要符合命名规范，重要语句也需加注释；
* 编程过程要进行异常捕获及处理；
* 对Web应用程序进行适当配置；

### **1.2.3**成果要求

* 提交完整的可运行的系统代码；
* 提交符合格式规范的实训报告。
* 通过学习认识了解企业在实际研发过程中采用的技术、开发过程和规范，加深对所学的软件开发技术、软件工程思想的理解和认识。
* 通过专业实习对本专业领域有深入了解，并根据实习内容对自己未来的工作方向进行定位。
* 可以根据自身情况，购买或使用云主机、云空间或免费空间，将最终项目部署在服务器上（可加分）

# **2**开发环境与开发技术

## 2.1开发环境

### **2.1.1**运行环境

* JDK：17
* 构建工具类型及版本：maven-3.8.8
* 数据库类型及版本：postgresSQL镜像v1
* 内存：16GB
* 接口调试工具：Apifox

### **2.1.2**开发工具

* IDE： idea2022
* 版本控制：git
* 开发技术：quarkus框架

### **2.1.3**Web框架

* Bootstarp框架

### **2.1.4**前端技术

* Html
* CSS
* JavaScript

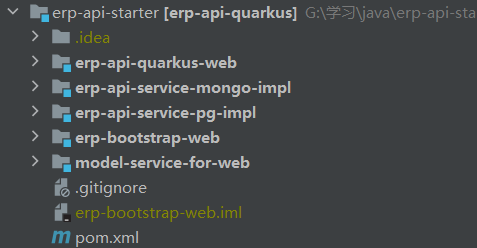
### **2.1.5**数据库

Postgresql:一个世界上最先进的、功能强大的开源对象关系数据库系统，经过超过 35 年的积极开发，在可靠性、功能稳健性和性能方面赢得了良好的声誉。

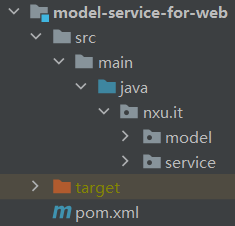
MongoDB:文档数据库。MongoDB中的一条记录就是一个文档，它是由字段和值对组成的数据结构。MongoDB 文档类似于 JSON 对象。字段的值可以包括其他文档、数组和文档数组。

## 2.2项目文件结构

总模块 erp-api-quarkus



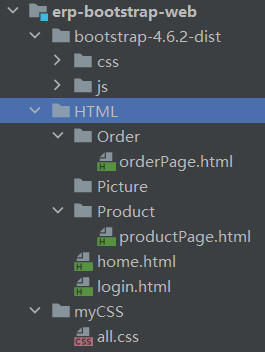
model-service-for-web模块



项目结构如上图所示。其中：

* src文件夹为项目源代码文件夹，包含model软件包、service软件包。为上层提供实体和接口。

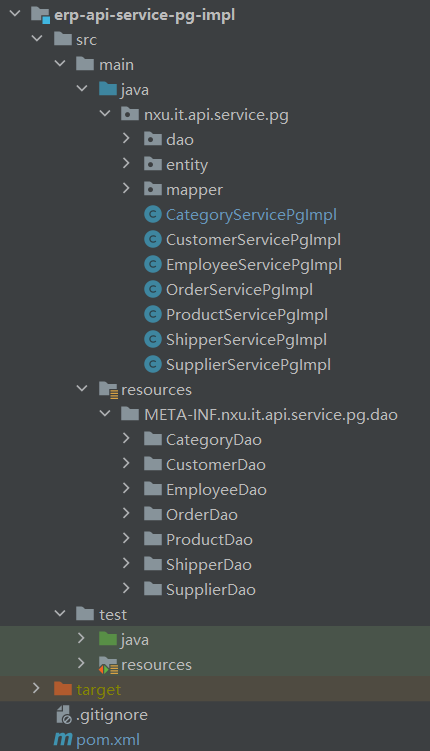
erp-bootstrap-web模块



项目结构如上图所示。其中：

* bootstrap-4.6.2-dist文件夹为需要用到bootstrap框架的css和js文件。HTML文件夹包含主页面、登录页面及各接口页面。myCSS为各页面自定义的css样式

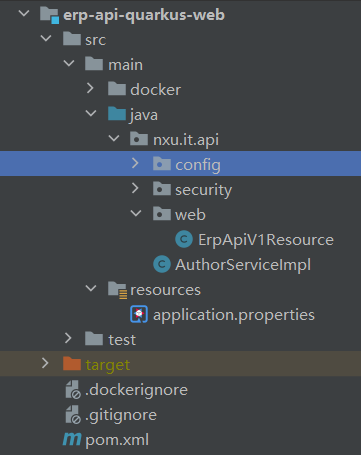
erp-api-service-pg-impl模块



项目结构如上图所示。其中：

* src文件夹为项目源代码文件夹，里面写了如何实现service接口。其中entity包内为与model相似的实体，用于映射数据库的查询结果。dao包内为映射接口，标明传入参数和传出参数。resources文件夹内包含映射接口实现用的sql语句。Mapper文件夹为用于将entity包中的实体转换为model中的实体的方法。
* test文件夹为项目测试代码文件夹。里面的java文件夹放了简单的测试类（只是用toString()做了显示）。resource文件夹内放了postgresql的连接信息。

erp-api-quarkus-web模块



项目结构如上图所示。其中：

* src文件夹为项目源代码文件夹，里面写了与网络相关和对项目打包用的docker。其中web包内为一个含有特定路径的类，里面方法为：根据url传入与传出json格式的拆封、封装及传入实现类中处理。

# **3**数据接口设计

## 3.1接口设计

Url前缀:/api/v1

### **3.1.1**产品分组

#### 3.1.1.1获取指定编号的产品接口

##### 3.1.1.1.1功能描述

接口地址：/product/{id}

功能：获取指定编号(id)的产品信息。

##### 3.1.1.1.2请求参数

请求类型：GET

参数类型：Path参数

参数描述：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **参数名称** | **参数说明** | **示例参数** | **必须** | **数据类型** |
| id | 产品编号 | 405870 | √ | string |

##### 3.1.1.1.3接口响应

响应类型：application/json

响应描述：

|  |  |  |  |
| --- | --- | --- | --- |
| **参数名称** | | **参数说明** | **数据类型** |
| code | | 代码 | integer |
| success | | 请求状态 | boolean |
| message | | 返回消息 | string |
| data | | 封装数据 | array |
| data中item | id | 产品编号 | integer |
| brand | 品牌 | string |
| name | 产品名字 | string |
| bar\_code | 条码 | string |
| quantity\_per\_unit | 规格 | string |
| unit\_price | 单价 | number |
| units\_in\_stock | 库存 | integer |
| units\_sell | 销售数量 | integer |
| photo\_url | 产品照片url | string |
| purchase\_unit\_price | 进货单价 | null |
| purchase\_amount | 进货数量 | null |
| supplier\_id | 供应商编号 | null |
| supplier\_name | 供应商名称 | null |
| category\_id1 | 1级的分类编号 | string |
| category\_name1 | 1级的分类名称 | string |
| category\_id2 | 2级的分类编号 | string |
| category\_name2 | 2级的分类名称 | string |
| category\_id3 | 3级的分类编号 | string |
| category\_name3 | 3级的分类名称 | string |
| pageNumber | | 页数 | integer |
| pageSize | | 每页记录数 | integer |
| totalPage | | 总页数 | integer |
| totalRow | | 总记录数 | integer |

示例响应：

{

    "code": 200,

    "success": true,

    "message": "请求成功",

    "data": [

        {

            "id": 405870,

            "brand": "苏菲",

            "name": "零敏肌超透气丝薄护垫清香型",

            "bar\_code": "0307010157",

            "quantity\_per\_unit": "155mm48片",

            "unit\_price": 9.9,

            "units\_in\_stock": 177,

            "units\_sell": 23,

            "photo\_url": "405870.jpg",

            "supplier\_id": 1,

            "supplier\_name": "湖南昌九清峰贸易有限公司",

            "category\_id1": "03",

            "category\_name1": "个护/美妆",

            "category\_id2": "0305",

            "category\_name2": "女性护理",

            "category\_id3": "030502",

            "category\_name3": "护垫"

        },

        {

            "id": 405870,

            "brand": "苏菲",

            "name": "零敏肌超透气丝薄护垫清香型",

            "bar\_code": "0307010157",

            "quantity\_per\_unit": "155mm48片",

            "unit\_price": 9.9,

            "units\_in\_stock": 177,

            "units\_sell": 23,

            "photo\_url": "405870.jpg",

            "supplier\_id": 14,

            "supplier\_name": "上海梅斯特姆贸易有限公司",

            "category\_id1": "03",

            "category\_name1": "个护/美妆",

            "category\_id2": "0305",

            "category\_name2": "女性护理",

            "category\_id3": "030502",

            "category\_name3": "护垫"

        }

    ]

}

#### 3.1.1.2产品查询接口

##### 3.1.1.2.1功能描述

接口地址：/products

功能：根据条件分页获取的产品列表

##### 3.1.1.2.2请求参数

请求类型：POST

参数类型：application/json

参数描述：ProductPageQueryParam、PageQueryParam为object类型

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **参数名称** | | | **参数说明** | **必须** | **数据类型** |
| ProductPageQueryParam | PageQueryParam | pageNumber | 页数 | √ | integer |
| pageSize | 每页记录数 | √ | integer |
| object | name | 产品名称 | × | string |
| brand | 品牌 | × | string |
| categoryId | 3级的分类编号 | × | string |
| minPrice | 价格范围中的小值 | × | number |
| maxPrice | 价格范围中的大值 | × | number |
| haveProduct | 库存中有此产品 | × | boolean |

示例参数:

{

    "pageNumber": 1,

    "pageSize": 10,

    "name": "水",

    "brand": "农"

}

##### 3.1.1.2.3接口响应

响应类型：application/json

响应描述：

示例响应：

{

    "code": 200,

    "success": true,

    "message": "请求成功",

    "data": [

        {

            "id": 417311,

            "brand": "农夫山泉",

            "name": "天然水",

            "bar\_code": "1102010049",

            "quantity\_per\_unit": "550ml×24瓶/件",

            "unit\_price": 39.9,

            "units\_in\_stock": 95,

            "units\_sell": 5,

            "photo\_url": "417311.jpg",

            "supplier\_id": null,

            "supplier\_name": null,

            "category\_id1": "01",

            "category\_name1": "食品饮料",

            "category\_id2": "0101",

            "category\_name2": "饮料",

            "category\_id3": "010104",

            "category\_name3": "饮用水"

        },

        {

            "id": 287471,

            "brand": "农夫山泉",

            "name": "天然水",

            "bar\_code": "1102010027",

            "quantity\_per\_unit": "550ml",

            "unit\_price": 1.8,

            "units\_in\_stock": 278,

            "units\_sell": 22,

            "photo\_url": "287471.jpg",

            "supplier\_id": null,

            "supplier\_name": null,

            "category\_id1": "01",

            "category\_name1": "食品饮料",

            "category\_id2": "0101",

            "category\_name2": "饮料",

            "category\_id3": "010104",

            "category\_name3": "饮用水"

        },

        {

            "id": 972253,

            "brand": "农夫山泉",

            "name": "饮用天然水/矿泉水",

            "bar\_code": "01040253",

            "quantity\_per\_unit": "380ml×24瓶/件",

            "unit\_price": 35.9,

            "units\_in\_stock": 99,

            "units\_sell": 1,

            "photo\_url": "972253.jpg",

            "supplier\_id": null,

            "supplier\_name": null,

            "category\_id1": "01",

            "category\_name1": "食品饮料",

            "category\_id2": "0101",

            "category\_name2": "饮料",

            "category\_id3": "010104",

            "category\_name3": "饮用水"

        }

    ],

    "pageNumber": 1,

    "pageSize": 10,

    "totalPage": 1,

    "totalRow": 3

}

#### 3.1.1.3根据供应商id产品查询接口

##### 3.1.1.3.1功能描述

接口地址：/supplier/{id}/products

功能：根据供应商号及条件分页获取的产品列表

##### 3.1.1.3.2请求参数

请求类型：POST

参数类型：path参数、application/json

参数描述：ProductPageQueryParam、PageQueryParam为object类型

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **参数名称** | | **参数说明** | | **必须** | **数据类型** |
| id | | 供应商号 | | √ | integer |
| ProductPageQueryParam | PageQueryParam | pageNumber | 页数 | √ | integer |
| pageSize | 每页记录数 | √ | integer |
| object | name | 产品名称 | × | string |
| brand | 品牌 | × | string |
| categoryId | 3级的分类编号 | × | string |
| minPrice | 价格范围中的小值 | × | number |
| maxPrice | 价格范围中的大值 | × | number |
| haveProduct | 库存中有此产品 | × | boolean |

示例参数:

{id}=87

{

"haveProduct": false

}

##### 3.1.1.3.3接口响应

响应类型：application/json

响应描述：

|  |  |  |  |
| --- | --- | --- | --- |
| **参数名称** | | **参数说明** | **数据类型** |
| code | | 代码 | integer |
| success | | 请求状态 | boolean |
| message | | 返回消息 | string |
| data | | 封装数据 | array |
| data中item | id | 产品编号 | integer |
| brand | 品牌 | string |
| name | 产品名字 | string |
| bar\_code | 条码 | string |
| quantity\_per\_unit | 规格 | string |
| unit\_price | 单价 | number |
| units\_in\_stock | 库存 | integer |
| units\_sell | 销售数量 | integer |
| photo\_url | 产品照片url | string |
| purchase\_unit\_price | 进货单价 | null |
| purchase\_amount | 进货数量 | null |
| supplier\_id | 供应商编号 | null |
| supplier\_name | 供应商名称 | null |
| category\_id1 | 1级的分类编号 | string |
| category\_name1 | 1级的分类名称 | string |
| category\_id2 | 2级的分类编号 | string |
| category\_name2 | 2级的分类名称 | string |
| category\_id3 | 3级的分类编号 | string |
| category\_name3 | 3级的分类名称 | string |
| pageNumber | | 页数 | integer |
| pageSize | | 每页记录数 | integer |
| totalPage | | 总页数 | integer |
| totalRow | | 总记录数 | integer |

示例响应：

{

    "code": 200,

    "success": true,

    "message": "请求成功",

    "data": [

        {

            "id": 417311,

            "brand": "农夫山泉",

            "name": "天然水",

            "bar\_code": "1102010049",

            "quantity\_per\_unit": "550ml×24瓶/件",

            "unit\_price": 39.9,

            "units\_in\_stock": 95,

            "units\_sell": 5,

            "photo\_url": "417311.jpg",

            "supplier\_id": null,

            "supplier\_name": null,

            "category\_id1": "01",

            "category\_name1": "食品饮料",

            "category\_id2": "0101",

            "category\_name2": "饮料",

            "category\_id3": "010104",

            "category\_name3": "饮用水"

        },

        {

            "id": 287471,

            "brand": "农夫山泉",

            "name": "天然水",

            "bar\_code": "1102010027",

            "quantity\_per\_unit": "550ml",

            "unit\_price": 1.8,

            "units\_in\_stock": 278,

            "units\_sell": 22,

            "photo\_url": "287471.jpg",

            "supplier\_id": null,

            "supplier\_name": null,

            "category\_id1": "01",

            "category\_name1": "食品饮料",

            "category\_id2": "0101",

            "category\_name2": "饮料",

            "category\_id3": "010104",

            "category\_name3": "饮用水"

        },

        {

            "id": 972253,

            "brand": "农夫山泉",

            "name": "饮用天然水/矿泉水",

            "bar\_code": "01040253",

            "quantity\_per\_unit": "380ml×24瓶/件",

            "unit\_price": 35.9,

            "units\_in\_stock": 99,

            "units\_sell": 1,

            "photo\_url": "972253.jpg",

            "supplier\_id": null,

            "supplier\_name": null,

            "category\_id1": "01",

            "category\_name1": "食品饮料",

            "category\_id2": "0101",

            "category\_name2": "饮料",

            "category\_id3": "010104",

            "category\_name3": "饮用水"

        }

    ],

    "pageNumber": 1,

    "pageSize": 10,

    "totalPage": 1,

    "totalRow": 3

}

#### 3.1.1.4进货记录接口

##### 3.1.1.4.1功能描述

接口地址：/stocks

功能：根据条件分页获取的产品库存列表

##### 3.1.1.4.2请求参数

请求类型：POST

参数类型：application/json

参数描述：PurchasePageQueryParam、PageQueryParam为object类型

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **参数名称** | | | **参数说明** | **必须** | **数据类型** |
| PurchasePageQueryParam | PageQueryParam | pageNumber | 页数 | √ | integer |
| pageSize | 每页记录数 | √ | integer |
| object | productName | 产品名称 | × | string |
| supplierName | 供应商名称 | × | string |
| startDate | 时间范围小值 | × | date |
| endDate | 时间范围大值 | × | date |

示例参数:

{

    "productName": "巾",

    "supplierName": "周",

    "startDate": "2018-12-10",

    "endDate": "2020-01-01"

}

##### 3.1.1.4.3接口响应

响应类型：application/json

响应描述：

|  |  |  |  |
| --- | --- | --- | --- |
| **参数名称** | | **参数说明** | **数据类型** |
| code | | 代码 | integer |
| success | | 请求状态 | boolean |
| message | | 返回消息 | string |
| data | | 封装数据 | array |
| data中item | id | 产品编号 | integer |
| brand | 品牌 | null |
| name | 产品名字 | string |
| bar\_code | 条码 | null |
| quantity\_per\_unit | 规格 | string |
| unit\_price | 单价 | null |
| units\_in\_stock | 库存 | null |
| units\_sell | 销售数量 | null |
| photo\_url | 产品照片url | null |
| purchase\_unit\_price | 进货单价 | integer |
| purchase\_amount | 进货数量 | integer |
| supplier\_id | 供应商编号 | integer |
| supplier\_name | 供应商名称 | string |
| category\_id1 | 1级的分类编号 | null |
| category\_name1 | 1级的分类名称 | null |
| category\_id2 | 2级的分类编号 | null |
| category\_name2 | 2级的分类名称 | null |
| category\_id3 | 3级的分类编号 | null |
| category\_name3 | 3级的分类名称 | null |
| pageNumber | | 页数 | integer |
| pageSize | | 每页记录数 | integer |
| totalPage | | 总页数 | integer |
| totalRow | | 总记录数 | integer |

示例响应：

{

    "code": 200,

    "success": true,

    "message": "请求成功",

    "data": [

        {

            "id": 211511,

            "brand": null,

            "name": "棉柔表层超吸纤薄夜用卫生巾K12",

            "bar\_code": null,

            "quantity\_per\_unit": "280mm\*8片",

            "unit\_price": null,

            "units\_in\_stock": null,

            "units\_sell": null,

            "photo\_url": null,

            "purchase\_unit\_price": 12,

            "purchase\_amount": 80,

            "supplier\_id": 87,

            "supplier\_name": "义乌立周贸易有限公司",

            "category\_id1": null,

            "category\_name1": null,

            "category\_id2": null,

            "category\_name2": null,

            "category\_id3": null,

            "category\_name3": null

        },

        {

            "id": 212424,

            "brand": null,

            "name": "减翼·简易棉柔表层超吸纤薄日用卫生巾K61",

            "bar\_code": null,

            "quantity\_per\_unit": "240mm\*8片",

            "unit\_price": null,

            "units\_in\_stock": null,

            "units\_sell": null,

            "photo\_url": null,

            "purchase\_unit\_price": 8,

            "purchase\_amount": 100,

            "supplier\_id": 87,

            "supplier\_name": "义乌立周贸易有限公司",

            "category\_id1": null,

            "category\_name1": null,

            "category\_id2": null,

            "category\_name2": null,

            "category\_id3": null,

            "category\_name3": null

        }

    ],

    "totalRow": 2,

    "totalPage": 1,

    "pageNumber": 1,

    "pageSize": 10

}

### **3.1.2订单**分组

#### 3.1.2.1获取指定编号的订单

##### 3.1.2.1.1功能描述

接口地址：/order/{id}

功能：获取指定编号的订单。

##### 3.1.2.1.2请求参数

请求类型：GET

参数类型：Path参数

参数描述：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **参数名称** | **参数说明** | **示例参数** | **必须** | **数据类型** |
| id | 订单编号 | 202001190001 | √ | string |

##### 3.1.2.1.3接口响应

响应类型：application/json

响应描述：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **参数名称** | | | **参数说明** | **数据类型** |
| code | | | 代码 | integer |
| success | | | 请求状态 | boolean |
| message | | | 返回消息 | string |
| data | | | 封装数据 | array |
| data中item | id | | 订单编号 | string |
| customer\_id | | 客户编号 | integer |
| customer\_name | | 客户名称 | string |
| customer\_contact | | 客户联系人 | string |
| customer\_phone | | 客户电话 | string |
| employee\_id | | 跟单员工编号 | string |
| employee\_name | | 跟单员工姓名 | string |
| employee\_phone | | 跟单员工电话 | string |
| order\_date | | 订货日期 | string |
| ship\_date | | 发货日期 | string |
| arrival\_date | | 到货日期 | string |
| freight | | 运费 | integer |
| shipper\_id | | 运货商编号 | integer |
| shipper\_name | | 运货商名称 | string |
| shipper\_phone | | 运货商电话 | string |
| ship\_address | | 收件详细地址 | string |
| ship\_province | | 收件省份 | string |
| ship\_city | | 收件城市 | string |
| ship\_region | | 收件地区 | string |
| contact | | 收件人姓名 | string |
| detail\_list | | 订单详情 | array |
| detail\_list内的item | unit\_price | 单价 | number |
| actual\_unit\_price | 销售单价 | number |
| quantity | 数量 | integer |
| discount | 折扣 | number |
| total\_price | 该产品的总价 | number |
| product\_id | 产品编号 | integer |
| product\_name | 产品名称 | string |
| product\_quantity\_per\_unit | 产品规格 | string |
| total\_price | | 订单总价 | number |

示例响应：

{

    "code": 200,

    "success": true,

    "message": "请求成功",

    "data": {

        "id": "202001190001",

        "customer\_id": 78,

        "customer\_name": "吉林省金易优选商贸有限公司",

        "customer\_contact": "金帮菊",

        "customer\_phone": "15910138510",

        "employee\_id": "2017001",

        "employee\_name": "马勇",

        "employee\_phone": "17836743577",

        "order\_date": "2020-01-19 14:10:29",

        "ship\_date": "2020-01-19 14:24:18",

        "arrival\_date": "2020-01-21 00:34:19",

        "freight": 19.0,

        "shipper\_id": 4,

        "shipper\_name": "圆通速递",

        "shipper\_phone": "95554",

        "ship\_address": "勃利县双河镇1028号",

        "ship\_province": "黑龙江",

        "ship\_city": "七台河",

        "ship\_region": "东北",

        "contact": "姚勇捷",

        "detail\_list": [

            {

                "unit\_price": 2980.0,

                "actual\_unit\_price": 2980.0,

                "quantity": 1,

                "discount": 0.0,

                "total\_price": 2980.0,

                "product\_id": 132370,

                "product\_name": "飞天茅台",

                "product\_quantity\_per\_unit": "500ML"

            },

            {

                "unit\_price": 9.9,

                "actual\_unit\_price": 9.41,

                "quantity": 9,

                "discount": 0.05,

                "total\_price": 84.69,

                "product\_id": 457716,

                "product\_name": "强力去垢亮洁洁厕液柠檬香",

                "product\_quantity\_per\_unit": "900g"

            }

        ],

        "total\_price": 3064.69

    }

}

#### 3.1.2.2订单查询接口

##### 3.1.2.2.1功能描述

接口地址：/orders

功能：根据条件分页获取的订单列表。

##### 3.1.2.2.2请求参数

请求类型：POST

参数类型：application/json

参数描述：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **参数名称** | | | **参数说明** | **必须** | **数据类型** |
| OrderPageQueryParam | PageQueryParam | pageNumber | 页数 | √ | integer |
| pageSize | 每页记录数 | √ | integer |
| object | startDate | 时间范围中小值 | × | string<date-time> |
| endDate | 时间范围中大值 | × | string<date-time> |
| customerName | 顾客名 | × | string |
| shipperName | 运货商名 | × | string |
| shipProvince | 目的地省份 | × | string |
| shipCity | 目的地城市 | × | string |
| shipRegion | 目的地地区 | × | string |

请求示例：

{

    "pageNumber": 1,

    "pageSize": 3,

    "startDate": "2019-07-03 10:08:00.000000",

    "endDate": "2021-12-01 00:00:01.000000",

    "customerName": "商贸",

    "shipperName": "圆通速递"

}

##### 3.1.2.2.3接口响应

响应类型：application/json

响应描述：

|  |  |  |  |
| --- | --- | --- | --- |
| **参数名称** | | **参数说明** | **数据类型** |
| code | | 代码 | integer |
| success | | 请求状态 | boolean |
| message | | 返回消息 | string |
| data | | 封装数据 | array |
| data中item | id | 订单编号 | string |
| customer\_id | 客户编号 | integer |
| customer\_name | 客户名称 | string |
| customer\_contact | 客户联系人 | string |
| customer\_phone | 客户电话 | string |
| employee\_id | 跟单员工编号 | string |
| employee\_name | 跟单员工姓名 | string |
| employee\_phone | 跟单员工电话 | string |
| order\_date | 订货日期 | string |
| ship\_date | 发货日期 | string |
| arrival\_date | 到货日期 | string |
| freight | 运费 | integer |
| shipper\_id | 运货商编号 | integer |
| shipper\_name | 运货商名称 | string |
| shipper\_phone | 运货商电话 | string |
| ship\_address | 收件详细地址 | string |
| ship\_province | 收件省份 | string |
| ship\_city | 收件城市 | string |
| ship\_region | 收件地区 | string |
| contact | 收件人姓名 | string |
| detail\_list | 订单详情 | null |
| total\_price | 订单总价 | null |
| pageNumber | | 页数 | integer |
| pageSize | | 每页记录数 | integer |
| totalPage | | 总页数 | integer |
| totalRow | | 总记录数 | integer |

示例响应：

{

    "code": 200,

    "success": true,

    "message": "请求成功",

    "data": [

        {

            "id": "201907250002",

            "customer\_id": 59,

            "customer\_name": "西安新福润商贸有限公司",

            "customer\_contact": "张晶",

            "customer\_phone": "15005415942",

            "employee\_id": "2018005",

            "employee\_name": "杨婧",

            "employee\_phone": "15661326148",

            "order\_date": "2019-07-25 01:19:23",

            "ship\_date": "2019-07-26 10:41:41",

            "arrival\_date": "2019-07-28 16:41:29",

            "freight": 59.0,

            "shipper\_id": 4,

            "shipper\_name": "圆通速递",

            "shipper\_phone": "95554",

            "ship\_address": "越秀区大新街076号",

            "ship\_province": "广东",

            "ship\_city": "广州",

            "ship\_region": "华南",

            "contact": "曹承恩",

            "detail\_list": null,

            "total\_price": null

        },

        {

            "id": "202001300001",

            "customer\_id": 24,

            "customer\_name": "青海永谐商贸有限公司",

            "customer\_contact": "冯迎曼",

            "customer\_phone": "17813961067",

            "employee\_id": "2018007",

            "employee\_name": "张雪",

            "employee\_phone": "17130107833",

            "order\_date": "2020-01-30 19:07:42",

            "ship\_date": "2020-02-02 13:58:58",

            "arrival\_date": "2020-02-12 05:13:34",

            "freight": 40.0,

            "shipper\_id": 4,

            "shipper\_name": "圆通速递",

            "shipper\_phone": "95554",

            "ship\_address": "香洲区翠香街办1309号",

            "ship\_province": "广东",

            "ship\_city": "珠海",

            "ship\_region": "华南",

            "contact": "石博涉",

            "detail\_list": null,

            "total\_price": null

        }

    ],

    "pageSize": 5,

    "pageNumber": 1,

    "totalPage": 1,

    "totalRow": 2

}

#### 3.1.2.3根据员工号订单查询接口

##### 3.1.2.3.1功能描述

接口地址：/employee/{id}/orders

功能：根据员工号及条件分页获取的订单列表。

##### 3.1.2.3.2请求参数

请求类型：POST

参数类型：path参数、application/json

参数描述：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **参数名称** | | | **参数说明** | **必须** | **数据类型** |
| id | | | 员工号 | √ | string |
| OrderPageQueryParam | PageQueryParam | pageNumber | 页数 | √ | integer |
| pageSize | 每页记录数 | √ | integer |
| object | startDate | 时间范围中小值 | × | string<date-time> |
| endDate | 时间范围中大值 | × | string<date-time> |
| customerName | 顾客名 | × | string |
| shipperName | 运货商名 | × | string |
| shipProvince | 目的地省份 | × | string |
| shipCity | 目的地城市 | × | string |
| shipRegion | 目的地地区 | × | string |

请求示例：

{id}=2017001

{

    "shipCity": "七台河"

}

##### 3.1.2.3.3接口响应

响应类型：application/json

响应描述：

|  |  |  |  |
| --- | --- | --- | --- |
| **参数名称** | | **参数说明** | **数据类型** |
| code | | 代码 | integer |
| success | | 请求状态 | boolean |
| message | | 返回消息 | string |
| data | | 封装数据 | array |
| data中item | id | 订单编号 | string |
| customer\_id | 客户编号 | integer |
| customer\_name | 客户名称 | string |
| customer\_contact | 客户联系人 | string |
| customer\_phone | 客户电话 | string |
| employee\_id | 跟单员工编号 | string |
| employee\_name | 跟单员工姓名 | string |
| employee\_phone | 跟单员工电话 | string |
| order\_date | 订货日期 | string |
| ship\_date | 发货日期 | string |
| arrival\_date | 到货日期 | string |
| freight | 运费 | integer |
| shipper\_id | 运货商编号 | integer |
| shipper\_name | 运货商名称 | string |
| shipper\_phone | 运货商电话 | string |
| ship\_address | 收件详细地址 | string |
| ship\_province | 收件省份 | string |
| ship\_city | 收件城市 | string |
| ship\_region | 收件地区 | string |
| contact | 收件人姓名 | string |
| detail\_list | 订单详情 | null |
| total\_price | 订单总价 | null |
| pageNumber | | 页数 | integer |
| pageSize | | 每页记录数 | integer |
| totalPage | | 总页数 | integer |
| totalRow | | 总记录数 | integer |

示例响应：

{

    "code": 200,

    "success": true,

    "message": "请求成功",

    "data": [

        {

            "id": "202001190001",

            "customer\_id": 78,

            "customer\_name": "吉林省金易优选商贸有限公司",

            "customer\_contact": "金帮菊",

            "customer\_phone": "15910138510",

            "employee\_id": "2017001",

            "employee\_name": "马勇",

            "employee\_phone": "17836743577",

            "order\_date": "2020-01-19 14:10:29",

            "ship\_date": "2020-01-19 14:24:18",

            "arrival\_date": "2020-01-21 00:34:19",

            "freight": 19.0,

            "shipper\_id": 4,

            "shipper\_name": "圆通速递",

            "shipper\_phone": "95554",

            "ship\_address": "勃利县双河镇1028号",

            "ship\_province": "黑龙江",

            "ship\_city": "七台河",

            "ship\_region": "东北",

            "contact": "姚勇捷",

            "detail\_list": null,

            "total\_price": null

        }

    ],

    "pageSize": 10,

    "pageNumber": 1,

    "totalPage": 1,

    "totalRow": 1

}

#### 3.1.2.4根据客户号订单查询接口

##### 3.1.2.4.1功能描述

接口地址：/customer/{id}/orders

功能：根据客户号及条件分页获取的订单列表。

##### 3.1.2.4.2请求参数

请求类型：POST

参数类型：path参数、application/json

参数描述：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **参数名称** | | | **参数说明** | **必须** | **数据类型** |
| id | | | 客户号 | √ | integer |
| OrderPageQueryParam | PageQueryParam | pageNumber | 页数 | √ | integer |
| pageSize | 每页记录数 | √ | integer |
| object | startDate | 时间范围中小值 | × | string<date-time> |
| endDate | 时间范围中大值 | × | string<date-time> |
| customerName | 顾客名 | × | string |
| shipperName | 运货商名 | × | string |
| shipProvince | 目的地省份 | × | string |
| shipCity | 目的地城市 | × | string |
| shipRegion | 目的地地区 | × | string |

请求示例：

{id}=78

{

    "shipRegion": "华"

}

##### 3.1.2.4.3接口响应

响应类型：application/json

响应描述：

|  |  |  |  |
| --- | --- | --- | --- |
| **参数名称** | | **参数说明** | **数据类型** |
| code | | 代码 | integer |
| success | | 请求状态 | boolean |
| message | | 返回消息 | string |
| data | | 封装数据 | array |
| data中item | id | 订单编号 | string |
| customer\_id | 客户编号 | integer |
| customer\_name | 客户名称 | string |
| customer\_contact | 客户联系人 | string |
| customer\_phone | 客户电话 | string |
| employee\_id | 跟单员工编号 | string |
| employee\_name | 跟单员工姓名 | string |
| employee\_phone | 跟单员工电话 | string |
| order\_date | 订货日期 | string |
| ship\_date | 发货日期 | string |
| arrival\_date | 到货日期 | string |
| freight | 运费 | integer |
| shipper\_id | 运货商编号 | integer |
| shipper\_name | 运货商名称 | string |
| shipper\_phone | 运货商电话 | string |
| ship\_address | 收件详细地址 | string |
| ship\_province | 收件省份 | string |
| ship\_city | 收件城市 | string |
| ship\_region | 收件地区 | string |
| contact | 收件人姓名 | string |
| detail\_list | 订单详情 | null |
| total\_price | 订单总价 | null |
| pageNumber | | 页数 | integer |
| pageSize | | 每页记录数 | integer |
| totalPage | | 总页数 | integer |
| totalRow | | 总记录数 | integer |

示例响应：

{

    "code": 200,

    "success": true,

    "message": "请求成功",

    "data": [

        {

            "id": "202002190002",

            "customer\_id": 78,

            "customer\_name": "吉林省金易优选商贸有限公司",

            "customer\_contact": "金帮菊",

            "customer\_phone": "15910138510",

            "employee\_id": "2019001",

            "employee\_name": "郭苹果",

            "employee\_phone": "15320844033",

            "order\_date": "2020-02-19 22:06:58",

            "ship\_date": "2020-02-22 01:22:39",

            "arrival\_date": "2020-02-29 18:15:42",

            "freight": 46.0,

            "shipper\_id": 1,

            "shipper\_name": "申通快递",

            "shipper\_phone": "95543",

            "ship\_address": "黄梅县严家闸811号",

            "ship\_province": "湖北",

            "ship\_city": "黄冈",

            "ship\_region": "华中",

            "contact": "余明达",

            "detail\_list": null,

            "total\_price": null

        }

    ],

    "pageSize": 10,

    "pageNumber": 1,

    "totalPage": 1,

    "totalRow": 1

}

#### 3.1.2.5根据运货商号订单查询接口

##### 3.1.2.5.1功能描述

接口地址：/shipper/{id}/orders

功能：根据运货商号及条件分页获取的订单列表。

##### 3.1.2.5.2请求参数

请求类型：POST

参数类型：path参数、application/json

参数描述：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **参数名称** | | | **参数说明** | **必须** | **数据类型** |
| id | | | 运货商号 | √ | integer |
| OrderPageQueryParam | PageQueryParam | pageNumber | 页数 | √ | integer |
| pageSize | 每页记录数 | √ | integer |
| object | startDate | 时间范围中小值 | × | string<date-time> |
| endDate | 时间范围中大值 | × | string<date-time> |
| customerName | 顾客名 | × | string |
| shipperName | 运货商名 | × | string |
| shipProvince | 目的地省份 | × | string |
| shipCity | 目的地城市 | × | string |
| shipRegion | 目的地地区 | × | string |

请求示例：

{id}=4

{

    "shipCity": "北京",

    "employeeName": "曹"

}

##### 3.1.2.5.3接口响应

响应类型：application/json

响应描述：

|  |  |  |  |
| --- | --- | --- | --- |
| **参数名称** | | **参数说明** | **数据类型** |
| code | | 代码 | integer |
| success | | 请求状态 | boolean |
| message | | 返回消息 | string |
| data | | 封装数据 | array |
| data中item | id | 订单编号 | string |
| customer\_id | 客户编号 | integer |
| customer\_name | 客户名称 | string |
| customer\_contact | 客户联系人 | string |
| customer\_phone | 客户电话 | string |
| employee\_id | 跟单员工编号 | string |
| employee\_name | 跟单员工姓名 | string |
| employee\_phone | 跟单员工电话 | string |
| order\_date | 订货日期 | string |
| ship\_date | 发货日期 | string |
| arrival\_date | 到货日期 | string |
| freight | 运费 | integer |
| shipper\_id | 运货商编号 | integer |
| shipper\_name | 运货商名称 | string |
| shipper\_phone | 运货商电话 | string |
| ship\_address | 收件详细地址 | string |
| ship\_province | 收件省份 | string |
| ship\_city | 收件城市 | string |
| ship\_region | 收件地区 | string |
| contact | 收件人姓名 | string |
| detail\_list | 订单详情 | null |
| total\_price | 订单总价 | null |
| pageNumber | | 页数 | integer |
| pageSize | | 每页记录数 | integer |
| totalPage | | 总页数 | integer |
| totalRow | | 总记录数 | integer |

示例响应：

{

    "code": 200,

    "success": true,

    "message": "请求成功",

    "data": [

        {

            "id": "202111100001",

            "customer\_id": 27,

            "customer\_name": "北京正云祥国际贸易有限公司",

            "customer\_contact": "赵万敏",

            "customer\_phone": "18310843487",

            "employee\_id": "2017007",

            "employee\_name": "曹润涵",

            "employee\_phone": "14617888040",

            "order\_date": "2021-11-18 11:14:34",

            "ship\_date": "2021-11-19 02:38:52",

            "arrival\_date": "2021-11-22 16:27:29",

            "freight": 49.0,

            "shipper\_id": 4,

            "shipper\_name": "圆通速递",

            "shipper\_phone": "95554",

            "ship\_address": "房山区十渡镇515号",

            "ship\_province": "北京",

            "ship\_city": "北京",

            "ship\_region": "华北",

            "contact": "廖星光",

            "detail\_list": null,

            "total\_price": null

        },

        {

            "id": "202103140001",

            "customer\_id": 27,

            "customer\_name": "北京正云祥国际贸易有限公司",

            "customer\_contact": "赵万敏",

            "customer\_phone": "18310843487",

            "employee\_id": "2017007",

            "employee\_name": "曹润涵",

            "employee\_phone": "14617888040",

            "order\_date": "2021-03-14 15:43:56",

            "ship\_date": "2021-03-15 19:39:10",

            "arrival\_date": "2021-03-25 04:00:43",

            "freight": 66.0,

            "shipper\_id": 4,

            "shipper\_name": "圆通速递",

            "shipper\_phone": "95554",

            "ship\_address": "昌平区马池口地区243号",

            "ship\_province": "北京",

            "ship\_city": "北京",

            "ship\_region": "华北",

            "contact": "袁宏浚",

            "detail\_list": null,

            "total\_price": null

        }

    ],

    "pageSize": 10,

    "pageNumber": 1,

    "totalPage": 1,

    "totalRow": 2

}

### **3.1.3员工**分组

#### 3.1.3.1员工查询接口

##### 3.1.3.1.1功能描述

接口地址：/employees

功能：根据条件分页获取的员工列表。

##### 3.1.3.1.2请求参数

请求类型：POST

参数类型：application/json

参数描述：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **参数名称** | | | **参数说明** | **必须** | **数据类型** |
| EmployeePageQueryParam | PageQueryParam | pageNumber | 页数 | √ | integer |
| pageSize | 每页记录数 | √ | integer |
| object | name | 姓名 | × | string |
| gender | 性别 | × | string |
| region | 所在地区 | × | string |
| title | 职务 | × | string |

请求示例：

{

    "name": "马",

    "gender": "男",

    "region": "北",

    "title": "代表"

}

##### 3.1.3.1.3接口响应

响应类型：application/json

响应描述：

|  |  |  |  |
| --- | --- | --- | --- |
| **参数名称** | | **参数说明** | **数据类型** |
| code | | 代码 | integer |
| success | | 请求状态 | boolean |
| message | | 返回消息 | string |
| data | | 封装数据 | array |
| data中item | id | 员工编号 | string |
| name | 姓名 | integer |
| birth\_date | 出生日期 | string |
| hire\_date | 雇佣日期 | string |
| title | 职务 | string |
| region | 所在地区 | string |
| phone | 联系电话 | string |
| pageNumber | | 页数 | integer |
| pageSize | | 每页记录数 | integer |
| totalPage | | 总页数 | integer |
| totalRow | | 总记录数 | integer |

示例响应：

{

    "code": 200,

    "success": true,

    "message": "请求成功",

    "data": [

        {

            "id": "2020002",

            "name": "马志炳",

            "birth\_date": "1990-07-20",

            "hire\_date": "2019-01-21",

            "title": "销售代表",

            "region": "西北",

            "phone": "17242331950"

        }

    ],

    "totalRow": 1,

    "totalPage": 1,

    "pageNumber": 1,

    "pageSize": 10

}

### **3.1.4客户**分组

#### 3.1.4.1客户查询接口

##### 3.1.4.1.1功能描述

接口地址：/customers

功能：根据条件分页获取的客户列表。

##### 3.1.4.1.2请求参数

请求类型：POST

参数类型：application/json

参数描述：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **参数名称** | | | **参数说明** | **必须** | **数据类型** |
| CustomerPageQueryParam | PageQueryParam | pageNumber | 页数 | √ | integer |
| pageSize | 每页记录数 | √ | integer |
| object | name | 客户名称 | × | string |
| region | 所在地区 | × | string |
| province | 省份 | × | string |
| city | 城市 | × | string |

请求示例：

{

    "name": "广",

    "city": "广"

}

##### 3.1.4.1.3接口响应

响应类型：application/json

响应描述：

|  |  |  |  |
| --- | --- | --- | --- |
| **参数名称** | | **参数说明** | **数据类型** |
| code | | 代码 | integer |
| success | | 请求状态 | boolean |
| message | | 返回消息 | string |
| data | | 封装数据 | array |
| data中item | id | 客户编号 | integer |
| name | 客户名称 | string |
| contact | 联系人 | string |
| province | 省份 | string |
| city | 城市 | string |
| address | 详细地址 | string |
| region | 所在地区 | string |
| phone | 联系电话 | string |
| pageNumber | | 页数 | integer |
| pageSize | | 每页记录数 | integer |
| totalPage | | 总页数 | integer |
| totalRow | | 总记录数 | integer |

示例响应：

{

    "code": 200,

    "success": true,

    "message": "请求成功",

    "data": [

        {

            "id": 10,

            "name": "广州靖心进出口贸易有限公司",

            "contact": "韩平夏",

            "province": "广东",

            "city": "广州",

            "address": "广东省广州市越秀区珠光街04号",

            "region": "华东",

            "phone": "17566461920"

        },

        {

            "id": 11,

            "name": "广州佳顺源科技有限公司",

            "contact": "卫桂兰",

            "province": "广东",

            "city": "广州",

            "address": "广东省广州市花都区梯面镇321号",

            "region": "华东",

            "phone": "15311607488"

        },

        {

            "id": 14,

            "name": "金梦飞扬贸易（广州）有限公司",

            "contact": "杨静",

            "province": "广东",

            "city": "广州",

            "address": "广东省广州市番禺区大石街1106号",

            "region": "华东",

            "phone": "15210426970"

        },

        {

            "id": 16,

            "name": "广州市新虹桥实业有限公司",

            "contact": "杨迎春",

            "province": "广东",

            "city": "广州",

            "address": "广东省广州市荔湾区海龙街1319号",

            "region": "华东",

            "phone": "19838724367"

        }

    ],

    "totalRow": 4,

    "totalPage": 1,

    "pageNumber": 1,

    "pageSize": 10

}

### **3.1.5运货商**分组

#### 3.1.5.1运货商查询接口

##### 3.1.5.1.1功能描述

接口地址：/shippers

功能：根据条件分页获取的运货商列表。

##### 3.1.5.1.2请求参数

请求类型：POST

参数类型：application/json

参数描述：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **参数名称** | | | **参数说明** | **必须** | **数据类型** |
| CustomerPageQueryParam | PageQueryParam | pageNumber | 页数 | √ | integer |
| pageSize | 每页记录数 | √ | integer |
| object | name | 运货商名称 | × | string |

请求示例：

{

    "name": "快"

}

##### 3.1.5.1.3接口响应

响应类型：application/json

响应描述：

|  |  |  |  |
| --- | --- | --- | --- |
| **参数名称** | | **参数说明** | **数据类型** |
| code | | 代码 | integer |
| success | | 请求状态 | boolean |
| message | | 返回消息 | string |
| data | | 封装数据 | array |
| data中item | id | 运货商编号 | integer |
| name | 名称 | string |
| phone | 联系电话 | string |
| pageNumber | | 页数 | integer |
| pageSize | | 每页记录数 | integer |
| totalPage | | 总页数 | integer |
| totalRow | | 总记录数 | integer |

示例响应：

{

    "code": 200,

    "success": true,

    "message": "请求成功",

    "data": [

        {

            "id": 1,

            "name": "申通快递",

            "phone": "95543"

        },

        {

            "id": 3,

            "name": "韵达快递",

            "phone": "95546"

        },

        {

            "id": 6,

            "name": "EMS快递",

            "phone": "11183"

        },

        {

            "id": 7,

            "name": "天天快递",

            "phone": "4001888888"

        },

        {

            "id": 10,

            "name": "京东快递",

            "phone": "4006033600"

        }

    ],

    "totalRow": 5,

    "totalPage": 1,

    "pageNumber": 1,

    "pageSize": 10

}

### **3.1.6供应商**分组

#### 3.1.6.1供应商查询接口

##### 3.1.6.1.1功能描述

接口地址：/suppliers

功能：根据条件分页获取的供货商列表。

##### 3.1.6.1.2请求参数

请求类型：POST

参数类型：application/json

参数描述：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **参数名称** | | | **参数说明** | **必须** | **数据类型** |
| CustomerPageQueryParam | PageQueryParam | pageNumber | 页数 | √ | integer |
| pageSize | 每页记录数 | √ | integer |
| object | name | 供货商名称 | × | string |

请求示例：

{

    "name": "广"

}

##### 3.1.6.1.3接口响应

响应类型：application/json

响应描述：

|  |  |  |  |
| --- | --- | --- | --- |
| **参数名称** | | **参数说明** | **数据类型** |
| code | | 代码 | integer |
| success | | 请求状态 | boolean |
| message | | 返回消息 | string |
| data | | 封装数据 | array |
| data中item | id | 供货商编号 | integer |
| name | 名称 | string |
| phone | 联系电话 | string |
| pageNumber | | 页数 | integer |
| pageSize | | 每页记录数 | integer |
| totalPage | | 总页数 | integer |
| totalRow | | 总记录数 | integer |

示例响应：

{

    "code": 200,

    "success": true,

    "message": "请求成功",

    "data": [

        {

            "id": 10,

            "name": "广州市凤凰进出口有限公司",

            "phone": "17228285641"

        },

        {

            "id": 18,

            "name": "广东昌晖商贸有限公司",

            "phone": "14749469067"

        },

        {

            "id": 43,

            "name": "广西佰方国际贸易有限公司",

            "phone": "18144115566"

        },

        {

            "id": 44,

            "name": "广西庆盈国际经贸有限公司",

            "phone": "15260225726"

        },

        {

            "id": 73,

            "name": "广西莹鑫国际物流有限公司",

            "phone": "13400793597"

        },

        {

            "id": 74,

            "name": "广西冠玖贸易有限公司",

            "phone": "18523712999"

        },

        {

            "id": 103,

            "name": "广西前海湾贸易有限公司",

            "phone": "17377004158"

        },

        {

            "id": 105,

            "name": "广州金达兴贸易有限公司",

            "phone": "13660502277"

        },

        {

            "id": 114,

            "name": "广州颇渐商贸有限公司",

            "phone": "18578678548"

        },

        {

            "id": 115,

            "name": "广州喜之吉进出口贸易有限公司",

            "phone": "17229529849"

        }

    ],

    "totalRow": 10,

    "totalPage": 1,

    "pageNumber": 1,

    "pageSize": 10

}

### **3.1.7产品分类**分组

#### 3.1.7.1产品分类查询接口

##### 3.1.7.1.1功能描述

接口地址：/categories/level/{value}

功能：获取指定等级的产品分类。

##### 3.1.7.1.2请求参数

请求类型：GET

参数类型：path参数

参数描述：

|  |  |  |  |
| --- | --- | --- | --- |
| **参数名称** | **参数说明** | **必须** | **数据类型** |
| value | 产品分类等级 | √ | string |

请求示例：

{value}=2

##### 3.1.7.1.3接口响应

响应类型：application/json

响应描述：结果包括分类编号、分类名称、上一级分类编号，第一级分类的上一级编号可以设置为NULL或０。

|  |  |  |  |
| --- | --- | --- | --- |
| **参数名称** | | **参数说明** | **数据类型** |
| code | | 代码 | integer |
| success | | 请求状态 | boolean |
| message | | 返回消息 | string |
| data | | 封装数据 | array |
| data中item | id3 | 分类等级3的编号 | string |
| name3 | 分类等级3的名称 | string |
| id2 | 分类等级2的编号 | string |
| name2 | 分类等级2的名称 | string |
| id1 | 分类等级1的编号 | string |
| name1 | 分类等级1的名称 | string |
| parent\_id | 上一级分类的编号 | string |
| level | 分类等级 | integer |

示例响应：

{

    "code": 200,

    "success": true,

    "message": "请求成功",

    "data": [

        {

            "id3": null,

            "name3": null,

            "id2": null,

            "name2": null,

            "id1": "01",

            "name1": "食品饮料",

            "parent\_id": null,

            "level": 1

        },

        {

            "id3": null,

            "name3": null,

            "id2": null,

            "name2": null,

            "id1": "02",

            "name1": "厨房食品",

            "parent\_id": null,

            "level": 1

        },

        {

            "id3": null,

            "name3": null,

            "id2": null,

            "name2": null,

            "id1": "03",

            "name1": "个护/美妆",

            "parent\_id": null,

            "level": 1

        },

        {

            "id3": null,

            "name3": null,

            "id2": null,

            "name2": null,

            "id1": "04",

            "name1": "生活家居",

            "parent\_id": null,

            "level": 1

        }

    ]

}

### **3.1.7作者分类**分组

#### 3.1.7.1作者查询接口

##### 3.1.7.1.1功能描述

接口地址：/authors

功能：获取作者信息，用于测试向前端传参的问题。

##### 3.1.7.1.2请求参数

请求类型：GET

参数类型：

参数描述：

|  |  |  |  |
| --- | --- | --- | --- |
| **参数名称** | **参数说明** | **必须** | **数据类型** |

请求示例：

##### 3.1.7.1.3接口响应

响应类型：application/json

响应描述：

|  |  |  |  |
| --- | --- | --- | --- |
| **参数名称** | | **参数说明** | **数据类型** |
| code | | 代码 | integer |
| success | | 请求状态 | boolean |
| message | | 返回消息 | string |
| data | | 封装数据 | array |
| data中item | name | 作者名称 | string |
| sno | 学号 | string |
| klass | 班级 | string |
| grade | 成绩 | integer |

示例响应：

{

    "code": 200,

    "success": true,

    "message": "请求成功",

    "data": [

        {

            "name": "赵大",

            "sno": "001",

            "klass": "计科1班",

            "grade": 2021

        },

        {

            "name": "王二",

            "sno": "002",

            "klass": "计科2班",

            "grade": 2021

        },

        {

            "name": "张三",

            "sno": "003",

            "klass": "计科3班",

            "grade": 2021

        },

        {

            "name": "李四",

            "sno": "004",

            "klass": "计科4班",

            "grade": 2021

        }

    ]

}

# **4**数据接口实现

## 4.1根据产品id查询产品接口

### **4.1.1**关系数据库实现

接口实现思路：通过表格连接，一次性将所有数据返回。其中结果中包含供应商表格的内容，得通过进货表格表格连接实现。而结果要包含3级产品分类等级则可以通过自连接3次。

Sql语句：

SELECT tpr.id, tpr.name, tpr.brand, tpr.bar\_code, tpr.quantity\_per\_unit, tpr.unit\_price, tpr.units\_in\_stock, tpr.units\_sell, tpr.photo\_url,

tsu.id AS supplier\_id, tsu.name AS supplier\_name,

t3.id AS category\_id3, t3.name AS category\_name3, t2.id AS category\_id2, t2.name AS category\_name2, t1.id AS category\_id1, t1.name AS category\_name1

FROM t\_product tpr

left join t\_purchase tpu on tpr.id = tpu.product\_id

left join t\_supplier tsu on tsu.id = tpu.supplier\_id

left join t\_category t3 on tpr.category\_id = t3.id

left join t\_category t2 on t2.id = t3.parent\_id

left join t\_category t1 on t1.id = t2.parent\_id

WHERE tpr.id = /\* productId \*/405870

Dao方法：

@Select

List<ProductEntity> findByProductId(Integer productId);

实现类的方法：

public List<Product> findByProductId(Integer productId) {  
 List<ProductEntity> productEntityList = productDao.findByProductId(productId);  
 return productMapper.toProductList(productEntityList);  
}

结果：



### **4.1.2**Mongo数据库实现

接口实现思路：

新建一个Entity中间类，用来映射数据库。再用filter过滤条件查询得到结果。最后用mapstruct映射到对应的model类，因为ProductEntity是一个产品内有多个供应商，而Product是一个供应商对应一个产品，同一名字的产品的多个供应商分为多个Product，所以自定义映射方法。

ProductMapper代码：

default Product toProduct(ProductEntity productEntity) {

if (productEntity == null) {

return null;

}

Product product = new Product();

product.setId(productEntity.getId());

product.setBrand(productEntity.getBrand());

product.setName(productEntity.getName());

product.setBarCode(productEntity.getBarCode());

product.setQuantityPerUnit(productEntity.getQuantityPerUnit());

product.setUnitPrice(productEntity.getUnitPrice().floatValue());

product.setUnitsInStock(productEntity.getUnitsInStock());

product.setUnitsSell(productEntity.getUnitsSell());

product.setPhoto(productEntity.getPhoto());

List<SimpleSupplierEntity> suppliers = productEntity.getSuppliers();

if (suppliers != null) {

product.setSupplierId(suppliers.get(0).getId().longValue());

product.setSupplierName(suppliers.get(0).getName());

suppliers.remove(0);

}

CategoriesOfProduct categories = productEntity.getCategory();

if (categories != null) {

String categoryId1 = categories.getLevel1().getId().toString();

if (categoryId1.length()%2 == 0) {

product.setCategoryId1(categoryId1);

product.setCategoryId2(categories.getLevel2().getId().toString());

product.setCategoryId3(categories.getLevel3().getId().toString());

} else {

product.setCategoryId1("0" + categoryId1);

product.setCategoryId2("0" + categories.getLevel2().getId().toString());

product.setCategoryId3("0" + categories.getLevel3().getId().toString());

}

product.setCategoryName1(categories.getLevel1().getName());

product.setCategoryName2(categories.getLevel2().getName());

product.setCategoryName3(categories.getLevel3().getName());

}

return product;

}

default List<Product> toProductList(List<ProductEntity> productEntityList) {

if (productEntityList == null) {

return null;

} else {

List<Product> list = new ArrayList();

Iterator productEntityIterator = productEntityList.iterator();

while(productEntityIterator.hasNext()) {

ProductEntity productEntity = (ProductEntity)productEntityIterator.next();

if (productEntity.getSuppliers() == null) { //映射不需要供应商信息时

list.add(this.toProduct(productEntity));

} else { //映射需要供应商信息时

while (!productEntity.getSuppliers().isEmpty()) {

list.add(this.toProduct(productEntity));

}

}

}

return list;

}

}

实现类代码：

@Inject

MongoDatabase db;

@Inject

ProductMapper productMapper;

@Override

public List<Product> findByProductId(Integer id) {

MongoCollection<ProductEntity> productEntityCollection = db.getCollection("product", ProductEntity.class);

Bson idFilter = Filters.eq("\_id", id);

ArrayList<ProductEntity> productEntityList = productEntityCollection.find(idFilter).into(new ArrayList<>());

List<Product> productList = productMapper.toProductList(productEntityList);

return productList;

}

查询结果与pg数据库的相同。

## 4.2产品查询接口

### **4.2.1**关系数据库实现

接口实现思路：通过多个if条件来判断有无对应的查询条件。通过表格连接，一次性将所有数据返回。其中结果包含3级产品分类等级则可以通过自连接3次来查询。从查询条件中获取每页的记录数、页数。计算总页数、总记录数、要显示的记录。最后封装成Pageable返回。

SQL语句：

SELECT tpr.id, tpr.name, tpr.brand, tpr.bar\_code, tpr.quantity\_per\_unit, tpr.unit\_price, tpr.units\_in\_stock, tpr.units\_sell, tpr.photo\_url,

t3.id AS category\_id3, t3.name AS category\_name3, t2.id AS category\_id2, t2.name AS category\_name2, t1.id AS category\_id1, t1.name AS category\_name1

FROM t\_product tpr

left join t\_category t3 on tpr.category\_id = t3.id

left join t\_category t2 on t2.id = t3.parent\_id

left join t\_category t1 on t1.id = t2.parent\_id

WHERE

/\*%if @isNotBlank(name) \*/

tpr.name ILIKE '%' || /\* name \*/'水' || '%'

/\*%end\*/

/\*%if @isNotBlank(brand) \*/

AND tpr.brand ILIKE '%' || /\* brand \*/'农' || '%'

/\*%end\*/

/\*%if @isNotBlank(categoryId) \*/

AND tpr.category\_id LIKE /\* categoryId \*/'01' || '%'

/\*%end\*/

/\*%if minPrice!=null && maxPrice!=null \*/

AND tpr.unit\_price BETWEEN /\* minPrice \*/20 AND /\* maxPrice \*/40

/\*%end\*/

/\*%if haveProduct!=null \*/

/\*%if !haveProduct \*/

AND tpr.units\_in\_stock = 0

/\*%else \*/

AND tpr.units\_in\_stock > 0

/\*%end\*/

/\*%end\*/

Dao方法：

@Select

List<ProductEntity> page(String name, String brand, String categoryId, Float minPrice, Float maxPrice, Boolean haveProduct);

接口实现方法：

public Pageable<Product> page(ProductPageQueryParam productPageQueryParam) {

List<ProductEntity> productEntityList = productDao.page(productPageQueryParam.getName(), productPageQueryParam.getBrand(),

productPageQueryParam.getCategoryId(), productPageQueryParam.getMinPrice(), productPageQueryParam.getMaxPrice(), productPageQueryParam.getHaveProduct());

List<Product> productList = productMapper.toProductList(productEntityList);

int totalRow = productList.size();

int pageSize = productPageQueryParam.getPageSize();

int pageNumber = productPageQueryParam.getPageNumber();

int totalPage = PageUtil.totalPage(totalRow, pageSize);

PageUtil.setOneAsFirstPageNo();

int start = PageUtil.getStart(pageNumber, pageSize);

int end = Math.min(start + pageSize, totalRow);

List<Product> subList = new ArrayList<>();

if (start < end) {

subList = productList.subList(start, end);

}

return new BasePage<>(subList, pageNumber, pageSize, totalPage, totalRow);

}

结果：



### **4.2.2**Mongo数据库实现

接口实现思路：

ProductEntity中间类，用来映射数据库。通过多个if条件来判断有无对应的查询条件，再用filterList装过滤条件，查询得到结果。从查询条件中获取每页的记录数、页数，查询结果中得到总记录数，计算总页数，查询结果中存储要显示的记录。最后用mapstruct映射到对应的model类。封装成Pageable返回。

ProductMapper代码同上。

实现类代码：

@Override

public Pageable<Product> pageBySupplierId(Integer supplierId, ProductPageQueryParam productPageQueryParam) {

String name = productPageQueryParam.getName();

String brand = productPageQueryParam.getBrand();

String categoryId = productPageQueryParam.getCategoryId();

Float minPrice = productPageQueryParam.getMinPrice();

Float maxPrice = productPageQueryParam.getMaxPrice();

Boolean haveProduct = productPageQueryParam.getHaveProduct();

ArrayList<Bson> filterList = new ArrayList<>();

Bson supplierIdFilter = Filters.elemMatch("suppliers", Filters.eq("id", supplierId));

filterList.add(supplierIdFilter);

if (name!=null && !name.isBlank()) {

Bson nameFilter = Filters.regex("name",

String.format(".\*%s.\*", name), "i");

filterList.add(nameFilter);

}

if (brand!=null && !brand.isBlank()) {

Bson brandFilter = Filters.regex("brand",

String.format(".\*%s.\*", brand), "i");

filterList.add(brandFilter);

}

if (categoryId!=null && !categoryId.isBlank()) {

int length = categoryId.length();

int category\_id = Integer.parseInt(categoryId.replaceFirst("^0+", "")); //去掉开头的0;

Bson categoryIdFilter;

if (length == 2) {

categoryIdFilter = Filters.eq("category.level1.\_id", category\_id);

} else if (length == 4) {

categoryIdFilter = Filters.eq("category.level2.\_id", category\_id);

} else if (length == 6) {

categoryIdFilter = Filters.eq("category.level3.\_id", category\_id);

} else {

categoryIdFilter = null;

}

filterList.add(categoryIdFilter);

}

if (minPrice!=null && maxPrice!=null) {

Bson minPriceFilter = Filters.gte("unitPrice", minPrice);

Bson maxPriceFilter = Filters.lte("unitPrice", maxPrice);

Bson unitPriceFilter = Filters.and(minPriceFilter, maxPriceFilter);

filterList.add(unitPriceFilter);

}

if (haveProduct!=null) {

Bson haveProductFilter;

if (haveProduct) {

haveProductFilter = Filters.gt("unitsInStock", 0);

} else {

haveProductFilter = Filters.eq("unitsInStock", 0);

}

filterList.add(haveProductFilter);

}

MongoCollection<ProductEntity> productCollection = db.getCollection("product", ProductEntity.class);

Bson allFilters = filterList.isEmpty() ? Filters.empty() : Filters.and(filterList);

long totalRow = productCollection.countDocuments(allFilters);

int pageSize = productPageQueryParam.getPageSize();

int pageNumber = productPageQueryParam.getPageNumber();

PageUtil.setOneAsFirstPageNo();

int totalPage = PageUtil.totalPage(totalRow, pageSize);

int offset = PageUtil.getStart(pageNumber, pageSize);

ArrayList<ProductEntity> productEntityList = productCollection.find(allFilters)

.skip(offset).limit(pageSize)

.projection(Projections.exclude("suppliers"))

.into(new ArrayList<>());

List<Product> productList = productMapper.toProductList(productEntityList);

BasePage<Product> productPage = new BasePage<>(productList, pageNumber, pageSize, totalPage, (int) totalRow);

return productPage;

}

查询结果与pg数据库的相同。

## 4.3根据供应商id及其它条件查询产品接口

### **4.3.1**关系数据库实现

接口实现思路：通过多个if条件来判断有无对应的查询条件。通过表格连接，一次性将所有数据返回。其中查询用到供应商id可以连接进货表获得。结果包含3级产品分类等级则可以通过自连接3次来查询。从查询条件中获取每页的记录数、页数。计算总页数、总记录数、要显示的记录。最后封装成Pageable返回。

SQL语句：

SELECT tpr.id, tpr.name, tpr.brand, tpr.bar\_code, tpr.quantity\_per\_unit, tpr.unit\_price, tpr.units\_in\_stock, tpr.units\_sell, tpr.photo\_url,

t3.id AS category\_id3, t3.name AS category\_name3, t2.id AS category\_id2, t2.name AS category\_name2, t1.id AS category\_id1, t1.name AS category\_name1

FROM t\_purchase tpu right join t\_product tpr on tpu.product\_id = tpr.id

left join t\_category t3 on tpr.category\_id = t3.id

left join t\_category t2 on t2.id = t3.parent\_id

left join t\_category t1 on t1.id = t2.parent\_id

WHERE supplier\_id = /\* supplierId \*/87

/\*%if @isNotBlank(name) \*/

AND tpr.name ILIKE '%' || /\* name \*/'水' || '%'

/\*%end\*/

/\*%if @isNotBlank(brand) \*/

AND tpr.brand ILIKE '%' || /\* brand \*/'农' || '%'

/\*%end\*/

/\*%if @isNotBlank(categoryId) \*/

AND tpr.category\_id LIKE /\* categoryId \*/'01' || '%'

/\*%end\*/

/\*%if minPrice!=null && maxPrice!=null \*/

AND tpr.unit\_price BETWEEN /\* minPrice \*/20 AND /\* maxPrice \*/40

/\*%end\*/

/\*%if haveProduct!=null \*/

/\*%if !haveProduct \*/

AND tpr.units\_in\_stock = 0

/\*%else \*/

AND tpr.units\_in\_stock > 0

/\*%end\*/

/\*%end\*/

Dao方法：

@Select

List<ProductEntity> pageBySupplierId(Integer supplierId, String name, String brand, String categoryId, Float minPrice, Float maxPrice, Boolean haveProduct);

接口实现方法：

public Pageable<Product> pageBySupplierId(Integer supplierId, ProductPageQueryParam productPageQueryParam) {

List<ProductEntity> productEntityList = productDao.pageBySupplierId(supplierId, productPageQueryParam.getName(), productPageQueryParam.getBrand(),

productPageQueryParam.getCategoryId(), productPageQueryParam.getMinPrice(), productPageQueryParam.getMaxPrice(), productPageQueryParam.getHaveProduct());

List<Product> productList = productMapper.toProductList(productEntityList);

int totalRow = productList.size();

int pageSize = productPageQueryParam.getPageSize();

int pageNumber = productPageQueryParam.getPageNumber();

int totalPage = PageUtil.totalPage(totalRow, pageSize);

PageUtil.setOneAsFirstPageNo();

int start = PageUtil.getStart(pageNumber, pageSize);

int end = Math.min(start + pageSize, totalRow);

List<Product> subList = new ArrayList<>();

if (start < end) {

subList = productList.subList(start, end);

}

return new BasePage<>(subList, pageNumber, pageSize, totalPage, totalRow);

}

结果：



### **4.3.2**Mongo数据库实现

接口实现思路：

ProductEntity中间类，用来映射数据库。supplierId查询条件直接装入filterList过滤条件，通过多个if条件来判断有无对应的查询条件，再装入filterList，查询得到结果。从查询条件中获取每页的记录数、页数，查询结果中得到总记录数，计算总页数，查询结果中存储要显示的记录。最后用mapstruct映射到对应的model类。封装成Pageable返回。

ProductMapper代码同上。

实现类代码：

@Override

public Pageable<Product> pageBySupplierId(Integer supplierId, ProductPageQueryParam productPageQueryParam) {

String name = productPageQueryParam.getName();

String brand = productPageQueryParam.getBrand();

String categoryId = productPageQueryParam.getCategoryId();

Float minPrice = productPageQueryParam.getMinPrice();

Float maxPrice = productPageQueryParam.getMaxPrice();

Boolean haveProduct = productPageQueryParam.getHaveProduct();

ArrayList<Bson> filterList = new ArrayList<>();

Bson supplierIdFilter = Filters.elemMatch("suppliers", Filters.eq("id", supplierId));

filterList.add(supplierIdFilter);

if (name!=null && !name.isBlank()) {

Bson nameFilter = Filters.regex("name",

String.format(".\*%s.\*", name), "i");

filterList.add(nameFilter);

}

if (brand!=null && !brand.isBlank()) {

Bson brandFilter = Filters.regex("brand",

String.format(".\*%s.\*", brand), "i");

filterList.add(brandFilter);

}

if (categoryId!=null && !categoryId.isBlank()) {

int length = categoryId.length();

int category\_id = Integer.parseInt(categoryId.replaceFirst("^0+", "")); //去掉开头的0;

Bson categoryIdFilter;

if (length == 2) {

categoryIdFilter = Filters.eq("category.level1.\_id", category\_id);

} else if (length == 4) {

categoryIdFilter = Filters.eq("category.level2.\_id", category\_id);

} else if (length == 6) {

categoryIdFilter = Filters.eq("category.level3.\_id", category\_id);

} else {

categoryIdFilter = null;

}

filterList.add(categoryIdFilter);

}

if (minPrice!=null && maxPrice!=null) {

Bson minPriceFilter = Filters.gte("unitPrice", minPrice);

Bson maxPriceFilter = Filters.lte("unitPrice", maxPrice);

Bson unitPriceFilter = Filters.and(minPriceFilter, maxPriceFilter);

filterList.add(unitPriceFilter);

}

if (haveProduct!=null) {

Bson haveProductFilter;

if (haveProduct) {

haveProductFilter = Filters.gt("unitsInStock", 0);

} else {

haveProductFilter = Filters.eq("unitsInStock", 0);

}

filterList.add(haveProductFilter);

}

MongoCollection<ProductEntity> productCollection = db.getCollection("product", ProductEntity.class);

Bson allFilters = filterList.isEmpty() ? Filters.empty() : Filters.and(filterList);

long totalRow = productCollection.countDocuments(allFilters);

int pageSize = productPageQueryParam.getPageSize();

int pageNumber = productPageQueryParam.getPageNumber();

PageUtil.setOneAsFirstPageNo();

int totalPage = PageUtil.totalPage(totalRow, pageSize);

int offset = PageUtil.getStart(pageNumber, pageSize);

ArrayList<ProductEntity> productEntityList = productCollection.find(allFilters)

.skip(offset).limit(pageSize)

.projection(Projections.exclude("suppliers"))

.into(new ArrayList<>());

List<Product> productList = productMapper.toProductList(productEntityList);

BasePage<Product> productPage = new BasePage<>(productList, pageNumber, pageSize, totalPage, (int) totalRow);

return productPage;

}

查询结果与pg数据库的相同。

## 4.4查询产品库存接口

### **4.4.1**关系数据库实现

接口实现思路：通过多个if条件来判断有无对应的查询条件。通过表格连接，一次性将所有数据返回。其中查询用到供应商表和产品表的信息可以通过连接获得。从查询条件中获取每页的记录数、页数。计算总页数、总记录数、要显示的记录。最后封装成Pageable返回。

SQL语句：

SELECT tpr.id, tpr.name, tpr.quantity\_per\_unit,

tpu.purchase\_unit\_price, tpu.amount AS purchase\_amount,

tsu.id AS supplier\_id, tsu.name AS supplier\_name

FROM t\_purchase tpu

left join t\_product tpr on tpu.product\_id = tpr.id

left join t\_supplier tsu on tpu.supplier\_id = tsu.id

WHERE

/\*%if @isNotBlank(name) \*/

tpr.name ILIKE '%' || /\* name \*/'巾' || '%'

/\*%end\*/

/\*%if @isNotBlank(supplierName) \*/

AND tsu.name ILIKE '%' || /\* supplierName \*/'周' || '%'

/\*%end\*/

/\*%if startDate!=null && endDate!=null \*/

AND tpu.purchase\_date BETWEEN /\* startDate \*/'2018-01-01' AND /\* endDate \*/'2022-01-01'

/\*%end\*/

Dao方法：

@Select

List<ProductEntity> purchasePage(String name, String supplierName, LocalDate startDate, LocalDate endDate);

接口实现方法：

public Pageable<Product> purchasePage(PurchasePageQueryParam purchasePageQueryParam) {

List<ProductEntity> productEntityList = productDao.purchasePage(purchasePageQueryParam.getProductName(), purchasePageQueryParam.getSupplierName(),

purchasePageQueryParam.getStartDate(), purchasePageQueryParam.getEndDate());

List<Product> productList = productMapper.toProductList(productEntityList);

int totalRow = productList.size();

int pageSize = purchasePageQueryParam.getPageSize();

int pageNumber = purchasePageQueryParam.getPageNumber();

int totalPage = PageUtil.totalPage(totalRow, pageSize);

PageUtil.setOneAsFirstPageNo();

int start = PageUtil.getStart(pageNumber, pageSize);

int end = Math.min(start + pageSize, totalRow);

List<Product> subList = new ArrayList<>();

if (start < end) {

subList = productList.subList(start, end);

}

return new BasePage<>(subList, pageNumber, pageSize, totalPage, totalRow);

}

结果：



### **4.4.2**Mongo数据库实现

接口实现思路：

ProductEntity中间类，用来映射数据库。supplierId查询条件直接装入filterList过滤条件，通过多个if条件来判断有无对应的查询条件，再装入filterList，查询得到结果。从查询条件中获取每页的记录数、页数，查询结果中得到总记录数，计算总页数，查询结果中存储要显示的记录。最后用mapstruct映射到对应的model类。封装成Pageable返回。

ProductMapper代码在上面的基础上，多用到的purchase部分：

@Mappings({

@Mapping(source = "product.id", target = "id"),

@Mapping(source = "product.name", target = "name"),

@Mapping(source = "product.quantityPerUnit", target = "quantityPerUnit"),

@Mapping(source = "amount", target = "purchaseAmount"),

@Mapping(source = "supplier.id", target = "supplierId"),

@Mapping(source = "supplier.name", target = "supplierName")

})

Product purchaseToProduct(PurchaseEntity purchaseEntity);

List<Product> purchasesToProductList(List<PurchaseEntity> purchaseEntityList);

实现类代码：

@Override

public Pageable<Product> purchasePage(PurchasePageQueryParam purchasePageQueryParam) {

String productName = purchasePageQueryParam.getProductName();

String supplierName = purchasePageQueryParam.getSupplierName();

LocalDate startDate = purchasePageQueryParam.getStartDate();

LocalDate endDate = purchasePageQueryParam.getEndDate();

ArrayList<Bson> filterList = new ArrayList<>();

if (productName!=null && !productName.isBlank()) {

Bson productNameFilter = Filters.regex("product.name",

String.format(".\*%s.\*", productName), "i");

filterList.add(productNameFilter);

}

if (supplierName!=null && !supplierName.isBlank()) {

Bson supplierNameFilter = Filters.regex("supplier.name",

String.format(".\*%s.\*", supplierName), "i");

filterList.add(supplierNameFilter);

}

if (startDate!=null && endDate!=null) {

Bson startDateFilter = Filters.gte("purchaseDate", startDate);

Bson endDateFilter = Filters.lte("purchaseDate", endDate);

Bson purchaseDateFilter = Filters.and(startDateFilter, endDateFilter);

filterList.add(purchaseDateFilter);

}

MongoCollection<PurchaseEntity> purchaseCollection = db.getCollection("purchase", PurchaseEntity.class);

Bson allFilters = filterList.isEmpty() ? Filters.empty() : Filters.and(filterList);

long totalRow = purchaseCollection.countDocuments(allFilters);

int pageSize = purchasePageQueryParam.getPageSize();

int pageNumber = purchasePageQueryParam.getPageNumber();

PageUtil.setOneAsFirstPageNo();

int totalPage = PageUtil.totalPage(totalRow, pageSize);

int offset = PageUtil.getStart(pageNumber, pageSize);

ArrayList<PurchaseEntity> purchaseEntityList = purchaseCollection.find(allFilters)

.skip(offset).limit(pageSize)

.into(new ArrayList<>());

List<Product> productList = productMapper.purchasesToProductList(purchaseEntityList);

BasePage<Product> productPage = new BasePage<>(productList, pageNumber, pageSize, totalPage, (int) totalRow);

return productPage;

}

查询结果与pg数据库的相同。

## 4.5根据订单id查询接口

### **4.5.1**关系数据库实现

接口实现思路：以订单表为主表，通过外键连接其它表格，一次性将与order关联的数据返回。再查询订单详情表，连接产品表，将与order\_detail关联的数据返回。计算订单的总价格。汇总所有结果于Order中。最后封装成Optional返回。

findByOrderId的SQL语句：

SELECT tor.id, tor.order\_date, tor.ship\_date, tor.arrival\_date, tor.freight, tor.ship\_address, tor.ship\_province, tor.ship\_city, tor.ship\_region,tor.contact,

tcu.id AS customer\_id, tcu.name AS customer\_name, tcu.contact AS customer\_contact, tcu.phone AS customer\_phone,

tem.id AS employee\_id, tem.name AS employee\_name, tem.phone AS employee\_phone,

tsh.id AS shipper\_id, tsh.name AS shipper\_name, tsh.phone AS shipper\_phone

FROM t\_order tor

left join t\_customer tcu on tor.customer\_id = tcu.id

left join t\_employee tem on tor.employee\_id = tem.id

left join t\_shipper tsh on tor.shipper\_id = tsh.id

WHERE tor.id = /\* orderId \*/'202001190001'

findDetailByOrderId的SQL语句：

SELECT tpr.id AS product\_id, tpr.name AS product\_name, tpr.quantity\_per\_unit AS product\_quantity\_per\_unit,

tor.quantity, tor.unit\_price, tor.actual\_unit\_price, tor.discount, tor.quantity\*tor.actual\_unit\_price AS total\_price

FROM t\_order\_detail tor left join t\_product tpr on tor.product\_id = tpr.id

WHERE tor.order\_id = /\* orderId \*/'201907030001'

Dao方法：

@Select

OrderEntity findByOrderId(String orderId);

@Select

List<OrderDetailEntity> findDetailByOrderId(String orderId);

接口实现方法：

public Optional<Order> findByOrderId(String id) {

OrderEntity orderEntity = orderDao.findByOrderId(id);

if (orderEntity == null) {

return Optional.empty();

}

Order order = orderMapper.toOrder(orderEntity);

List<OrderDetail> orderDetailList = orderMapper.toOrderDetailList(orderDao.findDetailByOrderId(id));

order.setDetail\_list(orderDetailList);

order.setTotal\_price(orderDetailList.stream().mapToDouble(OrderDetail::getTotal\_price).sum());

return Optional.of(order);

}

findByOrderId结果：



findDetailByOrderId结果：



### **4.5.2**Mongo数据库实现

接口实现思路：

OrdersEntity中间类，用来映射数据库。OrderId作为过滤条件，查询得到结果。最后用mapstruct映射到对应的model类。封装成Optional返回。

ProductMapper代码：

@Mapper(componentModel = MappingConstants.ComponentModel.CDI)

public interface OrderMapper {

private Integer orderDetailEntityProductId(OrderDetailEntity orderDetailEntity) {

if (orderDetailEntity == null) {

return null;

} else {

SimpleProductEntity product = orderDetailEntity.getProduct();

if (product == null) {

return null;

} else {

Integer id = product.getId();

return id == null ? null : id;

}

}

}

private String orderDetailEntityProductName(OrderDetailEntity orderDetailEntity) {

if (orderDetailEntity == null) {

return null;

} else {

SimpleProductEntity product = orderDetailEntity.getProduct();

if (product == null) {

return null;

} else {

String name = product.getName();

return name == null ? null : name;

}

}

}

private String orderDetailEntityProductQuantityPerUnit(OrderDetailEntity orderDetailEntity) {

if (orderDetailEntity == null) {

return null;

} else {

SimpleProductEntity product = orderDetailEntity.getProduct();

if (product == null) {

return null;

} else {

String quantityPerUnit = product.getQuantityPerUnit();

return quantityPerUnit == null ? null : quantityPerUnit;

}

}

}

/\*@Mappings({

@Mapping(source = "product.id", target = "productId"),

@Mapping(source = "product.name", target = "productName"),

@Mapping(source = "product.quantityPerUnit", target = "productQuantityPerUnit")

//totalPrice = quantity \* actualUnitPrice

})\*/

default OrderDetail toOrderDetail(OrderDetailEntity orderDetailEntity) {

if (orderDetailEntity == null) {

return null;

} else {

OrderDetail orderDetail = new OrderDetail();

orderDetail.setProductId(this.orderDetailEntityProductId(orderDetailEntity));

orderDetail.setProductName(this.orderDetailEntityProductName(orderDetailEntity));

orderDetail.setProductQuantityPerUnit(this.orderDetailEntityProductQuantityPerUnit(orderDetailEntity));

if (orderDetailEntity.getUnitPrice() != null) {

orderDetail.setUnitPrice(orderDetailEntity.getUnitPrice().floatValue());

}

Double actualUnitPrice = orderDetailEntity.getActualUnitPrice();

if (actualUnitPrice != null) {

orderDetail.setActualUnitPrice(actualUnitPrice.floatValue());

}

Integer quantity = orderDetailEntity.getQuantity();

orderDetail.setQuantity(quantity);

if (orderDetailEntity.getDiscount() != null) {

orderDetail.setDiscount(orderDetailEntity.getDiscount().floatValue());

}

if (actualUnitPrice != null && quantity != null) {

orderDetail.setTotalPrice(quantity\*actualUnitPrice);

}

return orderDetail;

}

}

@Named("toOrderDetailList")

List<OrderDetail> toOrderDetailList(List<OrderDetailEntity> orderDetailEntityList);

@Mappings({

@Mapping(source = "customer.id", target = "customerId"),

@Mapping(source = "customer.name", target = "customerName"),

@Mapping(source = "customer.contact", target = "customerContact"),

@Mapping(source = "customer.phone", target = "customerPhone"),

@Mapping(source = "employee.id", target = "employeeId"),

@Mapping(source = "employee.name", target = "employeeName"),

@Mapping(source = "employee.phone", target = "employeePhone"),

@Mapping(source = "shipInfo.freight", target = "freight"),

@Mapping(source = "shipInfo.shipper.id", target = "shipperId"),

@Mapping(source = "shipInfo.shipper.name", target = "shipperName"),

@Mapping(source = "shipInfo.phone", target = "shipperPhone"),

@Mapping(source = "shipInfo.shipAddress", target = "shipAddress"),

@Mapping(source = "shipInfo.shipProvince", target = "shipProvince"),

@Mapping(source = "shipInfo.shipCity", target = "shipCity"),

@Mapping(source = "shipInfo.shipRegion", target = "shipRegion"),

@Mapping(source = "shipInfo.contact", target = "contact"),

@Mapping(source = "details", target = "detailList", qualifiedByName = "toOrderDetailList"),

//在Impl中写totalPrice映射

})

Order toOrder(OrdersEntity ordersEntity);

List<Order> toOrderList(List<OrdersEntity> ordersEntityList);

}

实现类代码：

@Inject

MongoDatabase db;

@Inject

OrderMapper orderMapper;

@Override

public Optional<Order> findByOrderId(String id) {

MongoCollection<OrdersEntity> ordersEntityCollection = db.getCollection("orders", OrdersEntity.class);

Bson idFilter = Filters.eq("\_id", id);

OrdersEntity ordersEntity = ordersEntityCollection.find(idFilter).first();

if (ordersEntity==null) {

return Optional.empty();

}

Order order = orderMapper.toOrder(ordersEntity);

List<OrderDetail> detailList = order.getDetailList();

order.setTotalPrice(detailList.stream().mapToDouble(OrderDetail::getTotalPrice).sum());

return Optional.of(order);

}

查询结果与pg数据库的相同。

## 4.6订单查询接口

### **4.6.1**关系数据库实现

接口实现思路：先对时间查询条件进行处理，保证查询有效。通过多个if条件来判断有无对应的查询条件。通过表格连接，一次性将所有数据返回。从查询条件中获取每页的记录数、页数。计算总页数、总记录数、要显示的记录。最后封装成Pageable返回。

SQL语句：

SELECT tor.id, tor.order\_date, tor.ship\_date, tor.arrival\_date, tor.freight, tor.ship\_address, tor.ship\_province, tor.ship\_city, tor.ship\_region,tor.contact,

tcu.id AS customer\_id, tcu.name AS customer\_name, tcu.contact AS customer\_contact, tcu.phone AS customer\_phone,

tem.id AS employee\_id, tem.name AS employee\_name, tem.phone AS employee\_phone,

tsh.id AS shipper\_id, tsh.name AS shipper\_name, tsh.phone AS shipper\_phone

FROM t\_order tor

left join t\_customer tcu on tor.customer\_id = tcu.id

left join t\_employee tem on tor.employee\_id = tem.id

left join t\_shipper tsh on tor.shipper\_id = tsh.id

WHERE

/\*%if startDate != null || endDate != null \*/

tor.order\_date BETWEEN /\* startDate \*/'2019-07-03 10:08:00.000000' AND /\* endDate \*/'2021-12-01 00:00:01.000000'

/\*%end\*/

/\*%if @isNotBlank(customerName) \*/

AND tcu.name ILIKE '%' || /\* customerName \*/'商贸' || '%'

/\*%end\*/

/\*%if @isNotBlank(shipperName) \*/

AND tsh.name ILIKE '%' || /\* shipperName \*/'速' || '%'

/\*%end\*/

/\*%if @isNotBlank(shipProvince) \*/

AND tor.ship\_province ILIKE '%' || /\* shipProvince \*/'广东' || '%'

/\*%end\*/

/\*%if @isNotBlank(shipCity) \*/

AND tor.ship\_city ILIKE '%' || /\* shipCity \*/'北' || '%'

/\*%end\*/

/\*%if @isNotBlank(shipRegion) \*/

AND tor.ship\_region ILIKE '%' || /\* shipRegion \*/'华' || '%'

/\*%end\*/

/\*%if @isNotBlank(employeeName) \*/

AND tem.name ILIKE '%' || /\* employeeName \*/'王良伟' || '%'

/\*%end\*/

Dao方法：

@Select

List<OrderEntity> page(LocalDateTime startDate, LocalDateTime endDate, String customerName, String shipperName, String shipProvince, String shipCity, String shipRegion, String employeeName);

接口实现方法：

public Pageable<Order> page(OrderPageQueryParam orderPageQueryParam) {

LocalDateTime startDate = orderPageQueryParam.getStartDate();

if (startDate == null) {

startDate = LocalDateTime.of(LocalDate.of(2019, 1, 1), LocalTime.of(0, 0, 0));

}

LocalDateTime endDate = orderPageQueryParam.getEndDate();

if (endDate == null) {

endDate = LocalDateTime.of(LocalDate.of(2022, 1, 1), LocalTime.of(0, 0, 0));

}

List<OrderEntity> orderEntityList = orderDao.page(startDate, endDate, orderPageQueryParam.getCustomerName(), orderPageQueryParam.getShipperName(),

orderPageQueryParam.getShipProvince(), orderPageQueryParam.getShipCity(), orderPageQueryParam.getShipRegion(), orderPageQueryParam.getEmployeeName());

List<Order> orderList = orderMapper.toOrderList(orderEntityList);

int totalRow = orderList.size();

int pageSize = orderPageQueryParam.getPageSize();

int pageNumber = orderPageQueryParam.getPageNumber();

int totalPage = PageUtil.totalPage(totalRow, pageSize);

PageUtil.setOneAsFirstPageNo();

int start = PageUtil.getStart(pageNumber, pageSize);

int end = Math.min(start + pageSize, totalRow);

List<Order> subList = new ArrayList<>();

if (start < end) {

subList = orderList.subList(start, end);

}

return new BasePage<>(subList, pageNumber, pageSize, totalPage, totalRow);

}

结果：



### **4.6.2**Mongo数据库实现

接口实现思路：

OrdersEntity中间类，用来映射数据库。通过多个if条件来判断有无对应的查询条件，再装入filterList，查询得到结果。从查询条件中获取每页的记录数、页数，查询结果中得到总记录数，计算总页数，查询结果中存储要显示的记录。最后用mapstruct映射到对应的model类。封装成Pageable返回。

OrderMapper代码同findByOrderId的Mongo实现类。

实现类代码：

@Override

public Pageable<Order> page(OrderPageQueryParam orderPageQueryParam) {

LocalDateTime startDate = orderPageQueryParam.getStartDate();

LocalDateTime endDate = orderPageQueryParam.getEndDate();

String customerName = orderPageQueryParam.getCustomerName();

String shipperName = orderPageQueryParam.getShipperName();

String shipProvince = orderPageQueryParam.getShipProvince();

String shipCity = orderPageQueryParam.getShipCity();

String shipRegion = orderPageQueryParam.getShipRegion();

String employeeName = orderPageQueryParam.getEmployeeName();

ArrayList<Bson> filterList = new ArrayList<>();

if (startDate!=null && endDate!=null) {

Bson startDateFilter = Filters.gte("orderDate", startDate);

Bson endDateFilter = Filters.lte("orderDate", endDate);

Bson orderDateFilter = Filters.and(startDateFilter, endDateFilter);

filterList.add(orderDateFilter);

}

if (customerName!=null && !customerName.isBlank()) {

Bson customerNameFilter = Filters.regex("customer.name",

String.format(".\*%s.\*", customerName), "i");

filterList.add(customerNameFilter);

}

if (shipperName!=null && !shipperName.isBlank()) {

Bson shipperNameFilter = Filters.regex("shipInfo.shipper.name",

String.format(".\*%s.\*", shipperName), "i");

filterList.add(shipperNameFilter);

}

if (shipProvince!=null && !shipProvince.isBlank()) {

Bson shipProvinceFilter = Filters.regex("shipInfo.shipProvince",

String.format(".\*%s.\*", shipProvince), "i");

filterList.add(shipProvinceFilter);

}

if (shipCity!=null && !shipCity.isBlank()) {

Bson shipCityFilter = Filters.regex("shipInfo.shipCity",

String.format(".\*%s.\*", shipCity), "i");

filterList.add(shipCityFilter);

}

if (shipRegion!=null && !shipRegion.isBlank()) {

Bson shipRegionFilter = Filters.regex("shipInfo.shipRegion",

String.format(".\*%s.\*", shipRegion), "i");

filterList.add(shipRegionFilter);

}

if (employeeName!=null && !employeeName.isBlank()) {

Bson employeeNameFilter = Filters.regex("employee.name",

String.format(".\*%s.\*", employeeName), "i");

filterList.add(employeeNameFilter);

}

MongoCollection<OrdersEntity> ordersCollection = db.getCollection("orders", OrdersEntity.class);

Bson allFilters = filterList.isEmpty() ? Filters.empty() : Filters.and(filterList);

long totalRow = ordersCollection.countDocuments(allFilters);

int pageSize = orderPageQueryParam.getPageSize();

int pageNumber = orderPageQueryParam.getPageNumber();

PageUtil.setOneAsFirstPageNo();

int totalPage = PageUtil.totalPage(totalRow, pageSize);

int offset = PageUtil.getStart(pageNumber, pageSize);

ArrayList<OrdersEntity> orderEntityList = ordersCollection.find(allFilters)

.skip(offset).limit(pageSize)

.projection(Projections.exclude("details"))

.into(new ArrayList<>());

List<Order> orderList = orderMapper.toOrderList(orderEntityList);

BasePage<Order> orderPage = new BasePage<>(orderList, pageNumber, pageSize, totalPage, (int) totalRow);

return orderPage;

}

查询结果与pg数据库的相同。

## 4.7根据跟单人员编号查询订单接口

### **4.7.1**关系数据库实现

接口实现思路：先对时间查询条件进行处理，保证查询有效。通过多个if条件来判断有无对应的查询条件。通过表格连接，一次性将所有数据返回。从查询条件中获取每页的记录数、页数。计算总页数、总记录数、要显示的记录。最后封装成Pageable返回。

SQL语句：

SELECT tor.id, tor.order\_date, tor.ship\_date, tor.arrival\_date, tor.freight, tor.ship\_address, tor.ship\_province, tor.ship\_city, tor.ship\_region,tor.contact,

tcu.id AS customer\_id, tcu.name AS customer\_name, tcu.contact AS customer\_contact, tcu.phone AS customer\_phone,

tem.id AS employee\_id, tem.name AS employee\_name, tem.phone AS employee\_phone,

tsh.id AS shipper\_id, tsh.name AS shipper\_name, tsh.phone AS shipper\_phone

FROM t\_order tor

left join t\_customer tcu on tor.customer\_id = tcu.id

left join t\_employee tem on tor.employee\_id = tem.id

left join t\_shipper tsh on tor.shipper\_id = tsh.id

WHERE

tor.employee\_id = /\* employeeId \*/'2017001'

/\*%if startDate != null || endDate != null \*/

AND tor.order\_date BETWEEN /\* startDate \*/'2019-07-03 10:08:00.000000' AND /\* endDate \*/'2022-12-01 00:00:01.000000'

/\*%end\*/

/\*%if @isNotBlank(customerName) \*/

AND tcu.name ILIKE '%' || /\* customerName \*/'贸易' || '%'

/\*%end\*/

/\*%if @isNotBlank(shipperName) \*/

AND tsh.name ILIKE '%' || /\* shipperName \*/'速' || '%'

/\*%end\*/

/\*%if @isNotBlank(shipProvince) \*/

AND tor.ship\_province ILIKE '%' || /\* shipProvince \*/'黑龙江' || '%'

/\*%end\*/

/\*%if @isNotBlank(shipCity) \*/

AND tor.ship\_city ILIKE '%' || /\* shipCity \*/'七台河' || '%'

/\*%end\*/

/\*%if @isNotBlank(shipRegion) \*/

AND tor.ship\_region ILIKE '%' || /\* shipRegion \*/'华' || '%'

/\*%end\*/

Dao方法：

@Select

List<OrderEntity> pageByEmployeeId(String employeeId, LocalDateTime startDate, LocalDateTime endDate, String customerName,String shipperName, String shipProvince, String shipCity, String shipRegion);

接口实现方法：

public Pageable<Order> pageByEmployeeId(String employeeId, OrderPageQueryParam orderPageQueryParam) {

LocalDateTime startDate = orderPageQueryParam.getStartDate();

if (startDate == null) {

startDate = LocalDateTime.of(LocalDate.of(2019, 1, 1), LocalTime.of(0, 0, 0));

}

LocalDateTime endDate = orderPageQueryParam.getEndDate();

if (endDate == null) {

endDate = LocalDateTime.of(LocalDate.of(2022, 1, 1), LocalTime.of(0, 0, 0));

}

List<OrderEntity> orderEntityList = orderDao.pageByEmployeeId(employeeId ,startDate, endDate, orderPageQueryParam.getCustomerName(), orderPageQueryParam.getShipperName(),

orderPageQueryParam.getShipProvince(), orderPageQueryParam.getShipCity(), orderPageQueryParam.getShipRegion());

List<Order> orderList = orderMapper.toOrderList(orderEntityList);

int totalRow = orderList.size();

int pageSize = orderPageQueryParam.getPageSize();

int pageNumber = orderPageQueryParam.getPageNumber();

int totalPage = PageUtil.totalPage(totalRow, pageSize);

PageUtil.setOneAsFirstPageNo();

int start = PageUtil.getStart(pageNumber, pageSize);

int end = Math.min(start + pageSize, totalRow);

List<Order> subList = new ArrayList<>();

if (start < end) {

subList = orderList.subList(start, end);

}

return new BasePage<>(subList, pageNumber, pageSize, totalPage, totalRow);

}

结果：



### **4.7.2**Mongo数据库实现

接口实现思路：

OrdersEntity中间类，用来映射数据库。employeeId查询条件直接装入filterList过滤条件，通过多个if条件来判断有无对应的查询条件，再装入filterList，查询得到结果。从查询条件中获取每页的记录数、页数，查询结果中得到总记录数，计算总页数，查询结果中存储要显示的记录。最后用mapstruct映射到对应的model类。封装成Pageable返回。

OrderMapper代码同findByOrderId的Mongo实现类。

实现类代码：

@Override

public Pageable<Order> pageByEmployeeId(String employeeId, OrderPageQueryParam orderPageQueryParam) {

LocalDateTime startDate = orderPageQueryParam.getStartDate();

LocalDateTime endDate = orderPageQueryParam.getEndDate();

String customerName = orderPageQueryParam.getCustomerName();

String shipperName = orderPageQueryParam.getShipperName();

String shipProvince = orderPageQueryParam.getShipProvince();

String shipCity = orderPageQueryParam.getShipCity();

String shipRegion = orderPageQueryParam.getShipRegion();

String employeeName = orderPageQueryParam.getEmployeeName();

ArrayList<Bson> filterList = new ArrayList<>();

Bson employeeIdFilter = Filters.eq("employee.id", employeeId);

filterList.add(employeeIdFilter);

if (startDate!=null && endDate!=null) {

Bson startDateFilter = Filters.gte("orderDate", startDate);

Bson endDateFilter = Filters.lte("orderDate", endDate);

Bson orderDateFilter = Filters.and(startDateFilter, endDateFilter);

filterList.add(orderDateFilter);

}

if (customerName!=null && !customerName.isBlank()) {

Bson customerNameFilter = Filters.regex("customer.name",

String.format(".\*%s.\*", customerName), "i");

filterList.add(customerNameFilter);

}

if (shipperName!=null && !shipperName.isBlank()) {

Bson shipperNameFilter = Filters.regex("shipInfo.shipper.name",

String.format(".\*%s.\*", shipperName), "i");

filterList.add(shipperNameFilter);

}

if (shipProvince!=null && !shipProvince.isBlank()) {

Bson shipProvinceFilter = Filters.regex("shipInfo.shipProvince",

String.format(".\*%s.\*", shipProvince), "i");

filterList.add(shipProvinceFilter);

}

if (shipCity!=null && !shipCity.isBlank()) {

Bson shipCityFilter = Filters.regex("shipInfo.shipCity",

String.format(".\*%s.\*", shipCity), "i");

filterList.add(shipCityFilter);

}

if (shipRegion!=null && !shipRegion.isBlank()) {

Bson shipRegionFilter = Filters.regex("shipInfo.shipRegion",

String.format(".\*%s.\*", shipRegion), "i");

filterList.add(shipRegionFilter);

}

if (employeeName!=null && !employeeName.isBlank()) {

Bson employeeNameFilter = Filters.regex("employee.name",

String.format(".\*%s.\*", employeeName), "i");

filterList.add(employeeNameFilter);

}

MongoCollection<OrdersEntity> ordersCollection = db.getCollection("orders", OrdersEntity.class);

Bson allFilters = filterList.isEmpty() ? Filters.empty() : Filters.and(filterList);

long totalRow = ordersCollection.countDocuments(allFilters);

int pageSize = orderPageQueryParam.getPageSize();

int pageNumber = orderPageQueryParam.getPageNumber();

PageUtil.setOneAsFirstPageNo();

int totalPage = PageUtil.totalPage(totalRow, pageSize);

int offset = PageUtil.getStart(pageNumber, pageSize);

ArrayList<OrdersEntity> orderEntityList = ordersCollection.find(allFilters)

.skip(offset).limit(pageSize)

.projection(Projections.exclude("details"))

.into(new ArrayList<>());

List<Order> orderList = orderMapper.toOrderList(orderEntityList);

BasePage<Order> orderPage = new BasePage<>(orderList, pageNumber, pageSize, totalPage, (int) totalRow);

return orderPage;

}

查询结果与pg数据库的相同。

## 4.8根据客户id查询订单接口

### **4.8.1**关系数据库实现

接口实现思路：先对时间查询条件进行处理，保证查询有效。通过多个if条件来判断有无对应的查询条件。通过表格连接，一次性将所有数据返回。从查询条件中获取每页的记录数、页数。计算总页数、总记录数、要显示的记录。最后封装成Pageable返回。

SQL语句：

SELECT tor.id, tor.order\_date, tor.ship\_date, tor.arrival\_date, tor.freight, tor.ship\_address, tor.ship\_province, tor.ship\_city, tor.ship\_region,tor.contact,

tcu.id AS customer\_id, tcu.name AS customer\_name, tcu.contact AS customer\_contact, tcu.phone AS customer\_phone,

tem.id AS employee\_id, tem.name AS employee\_name, tem.phone AS employee\_phone,

tsh.id AS shipper\_id, tsh.name AS shipper\_name, tsh.phone AS shipper\_phone

FROM t\_order tor

left join t\_customer tcu on tor.customer\_id = tcu.id

left join t\_employee tem on tor.employee\_id = tem.id

left join t\_shipper tsh on tor.shipper\_id = tsh.id

WHERE

tor.customer\_id = /\* customerId \*/78

/\*%if startDate != null || endDate != null \*/

AND tor.order\_date BETWEEN /\* startDate \*/'2019-07-03 10:08:00.000000' AND /\* endDate \*/'2022-12-01 00:00:01.000000'

/\*%end\*/

/\*%if @isNotBlank(shipperName) \*/

AND tsh.name ILIKE '%' || /\* shipperName \*/'速' || '%'

/\*%end\*/

/\*%if @isNotBlank(shipProvince) \*/

AND tor.ship\_province ILIKE '%' || /\* shipProvince \*/'吉林' || '%'

/\*%end\*/

/\*%if @isNotBlank(shipCity) \*/

AND tor.ship\_city ILIKE '%' || /\* shipCity \*/'通' || '%'

/\*%end\*/

/\*%if @isNotBlank(shipRegion) \*/

AND tor.ship\_region ILIKE '%' || /\* shipRegion \*/'华' || '%'

/\*%end\*/

/\*%if @isNotBlank(employeeName) \*/

AND tem.name ILIKE '%' || /\* employeeName \*/'郑康伟业' || '%'

/\*%end\*/

Dao方法：

@Select

List<OrderEntity> pageByCustomerId(Integer customerId, LocalDateTime startDate, LocalDateTime endDate, String shipperName, String shipProvince, String shipCity, String shipRegion, String employeeName);

接口实现方法：

public Pageable<Order> pageByCustomerId(Integer customerId, OrderPageQueryParam orderPageQueryParam) {

LocalDateTime startDate = orderPageQueryParam.getStartDate();

if (startDate == null) {

startDate = LocalDateTime.of(LocalDate.of(2019, 1, 1), LocalTime.of(0, 0, 0));

}

LocalDateTime endDate = orderPageQueryParam.getEndDate();

if (endDate == null) {

endDate = LocalDateTime.of(LocalDate.of(2022, 1, 1), LocalTime.of(0, 0, 0));

}

List<OrderEntity> orderEntityList = orderDao.pageByCustomerId(customerId, startDate, endDate, orderPageQueryParam.getShipperName(),

orderPageQueryParam.getShipProvince(), orderPageQueryParam.getShipCity(), orderPageQueryParam.getShipRegion(), orderPageQueryParam.getEmployeeName());

List<Order> orderList = orderMapper.toOrderList(orderEntityList);

int totalRow = orderList.size();

int pageSize = orderPageQueryParam.getPageSize();

int pageNumber = orderPageQueryParam.getPageNumber();

int totalPage = PageUtil.totalPage(totalRow, pageSize);

PageUtil.setOneAsFirstPageNo();

int start = PageUtil.getStart(pageNumber, pageSize);

int end = Math.min(start + pageSize, totalRow);

List<Order> subList = new ArrayList<>();

if (start < end) {

subList = orderList.subList(start, end);

}

return new BasePage<>(subList, pageNumber, pageSize, totalPage, totalRow);

}

结果：



### **4.8.2**Mongo数据库实现

接口实现思路：

OrdersEntity中间类，用来映射数据库。customerId查询条件直接装入filterList过滤条件，通过多个if条件来判断有无对应的查询条件，再装入filterList，查询得到结果。从查询条件中获取每页的记录数、页数，查询结果中得到总记录数，计算总页数，查询结果中存储要显示的记录。最后用mapstruct映射到对应的model类。封装成Pageable返回。

OrderMapper代码同findByOrderId的Mongo实现类。

实现类代码：

@Override

public Pageable<Order> pageByCustomerId(Integer customerId, OrderPageQueryParam orderPageQueryParam) {

LocalDateTime startDate = orderPageQueryParam.getStartDate();

LocalDateTime endDate = orderPageQueryParam.getEndDate();

String customerName = orderPageQueryParam.getCustomerName();

String shipperName = orderPageQueryParam.getShipperName();

String shipProvince = orderPageQueryParam.getShipProvince();

String shipCity = orderPageQueryParam.getShipCity();

String shipRegion = orderPageQueryParam.getShipRegion();

String employeeName = orderPageQueryParam.getEmployeeName();

ArrayList<Bson> filterList = new ArrayList<>();

Bson customerIdFilter = Filters.eq("customer.id", customerId);

filterList.add(customerIdFilter);

if (startDate!=null && endDate!=null) {

Bson startDateFilter = Filters.gte("orderDate", startDate);

Bson endDateFilter = Filters.lte("orderDate", endDate);

Bson orderDateFilter = Filters.and(startDateFilter, endDateFilter);

filterList.add(orderDateFilter);

}

if (customerName!=null && !customerName.isBlank()) {

Bson customerNameFilter = Filters.regex("customer.name",

String.format(".\*%s.\*", customerName), "i");

filterList.add(customerNameFilter);

}

if (shipperName!=null && !shipperName.isBlank()) {

Bson shipperNameFilter = Filters.regex("shipInfo.shipper.name",

String.format(".\*%s.\*", shipperName), "i");

filterList.add(shipperNameFilter);

}

if (shipProvince!=null && !shipProvince.isBlank()) {

Bson shipProvinceFilter = Filters.regex("shipInfo.shipProvince",

String.format(".\*%s.\*", shipProvince), "i");

filterList.add(shipProvinceFilter);

}

if (shipCity!=null && !shipCity.isBlank()) {

Bson shipCityFilter = Filters.regex("shipInfo.shipCity",

String.format(".\*%s.\*", shipCity), "i");

filterList.add(shipCityFilter);

}

if (shipRegion!=null && !shipRegion.isBlank()) {

Bson shipRegionFilter = Filters.regex("shipInfo.shipRegion",

String.format(".\*%s.\*", shipRegion), "i");

filterList.add(shipRegionFilter);

}

if (employeeName!=null && !employeeName.isBlank()) {

Bson employeeNameFilter = Filters.regex("employee.name",

String.format(".\*%s.\*", employeeName), "i");

filterList.add(employeeNameFilter);

}

MongoCollection<OrdersEntity> ordersCollection = db.getCollection("orders", OrdersEntity.class);

Bson allFilters = filterList.isEmpty() ? Filters.empty() : Filters.and(filterList);

long totalRow = ordersCollection.countDocuments(allFilters);

int pageSize = orderPageQueryParam.getPageSize();

int pageNumber = orderPageQueryParam.getPageNumber();

PageUtil.setOneAsFirstPageNo();

int totalPage = PageUtil.totalPage(totalRow, pageSize);

int offset = PageUtil.getStart(pageNumber, pageSize);

ArrayList<OrdersEntity> orderEntityList = ordersCollection.find(allFilters)

.skip(offset).limit(pageSize)

.projection(Projections.exclude("details"))

.into(new ArrayList<>());

List<Order> orderList = orderMapper.toOrderList(orderEntityList);

BasePage<Order> orderPage = new BasePage<>(orderList, pageNumber, pageSize, totalPage, (int) totalRow);

return orderPage;

}

查询结果与pg数据库的相同。

## 4.9根据运货商id查询订单接口

### **4.9.1**关系数据库实现

接口实现思路：先对时间查询条件进行处理，保证查询有效。通过多个if条件来判断有无对应的查询条件。通过表格连接，一次性将所有数据返回。从查询条件中获取每页的记录数、页数。计算总页数、总记录数、要显示的记录。最后封装成Pageable返回。

SQL语句：

SELECT tor.id, tor.order\_date, tor.ship\_date, tor.arrival\_date, tor.freight, tor.ship\_address, tor.ship\_province, tor.ship\_city, tor.ship\_region,tor.contact,

tcu.id AS customer\_id, tcu.name AS customer\_name, tcu.contact AS customer\_contact, tcu.phone AS customer\_phone,

tem.id AS employee\_id, tem.name AS employee\_name, tem.phone AS employee\_phone,

tsh.id AS shipper\_id, tsh.name AS shipper\_name, tsh.phone AS shipper\_phone

FROM t\_order tor

left join t\_customer tcu on tor.customer\_id = tcu.id

left join t\_employee tem on tor.employee\_id = tem.id

left join t\_shipper tsh on tor.shipper\_id = tsh.id

WHERE

tor.shipper\_id = /\* shipperId \*/4

/\*%if startDate != null || endDate != null \*/

AND tor.order\_date BETWEEN /\* startDate \*/'2019-07-03 10:08:00.000000' AND /\* endDate \*/'2022-12-01 00:00:01.000000'

/\*%end\*/

/\*%if @isNotBlank(customerName) \*/

AND tcu.name ILIKE '%' || /\* customerName \*/'北' || '%'

/\*%end\*/

/\*%if @isNotBlank(shipProvince) \*/

AND tor.ship\_province ILIKE '%' || /\* shipProvince \*/'北京' || '%'

/\*%end\*/

/\*%if @isNotBlank(shipCity) \*/

AND tor.ship\_city ILIKE '%' || /\* shipCity \*/'北' || '%'

/\*%end\*/

/\*%if @isNotBlank(shipRegion) \*/

AND tor.ship\_region ILIKE '%' || /\* shipRegion \*/'华' || '%'

/\*%end\*/

/\*%if @isNotBlank(employeeName) \*/

AND tem.name ILIKE '%' || /\* employeeName \*/'曹' || '%'

/\*%end\*/

Dao方法：

@Select

List<OrderEntity> pageByShipperId(Integer shipperId, LocalDateTime startDate, LocalDateTime endDate, String customerName, String shipProvince, String shipCity, String shipRegion, String employeeName);

接口实现方法：

public Pageable<Order> pageByShipperId(Integer shipperId, OrderPageQueryParam orderPageQueryParam) {

LocalDateTime startDate = orderPageQueryParam.getStartDate();

if (startDate == null) {

startDate = LocalDateTime.of(LocalDate.of(2019, 1, 1), LocalTime.of(0, 0, 0));

}

LocalDateTime endDate = orderPageQueryParam.getEndDate();

if (endDate == null) {

endDate = LocalDateTime.of(LocalDate.of(2022, 1, 1), LocalTime.of(0, 0, 0));

}

List<OrderEntity> orderEntityList = orderDao.pageByShipperId(shipperId, startDate, endDate, orderPageQueryParam.getCustomerName(),

orderPageQueryParam.getShipProvince(), orderPageQueryParam.getShipCity(), orderPageQueryParam.getShipRegion(), orderPageQueryParam.getEmployeeName());

List<Order> orderList = orderMapper.toOrderList(orderEntityList);

int totalRow = orderList.size();

int pageSize = orderPageQueryParam.getPageSize();

int pageNumber = orderPageQueryParam.getPageNumber();

int totalPage = PageUtil.totalPage(totalRow, pageSize);

PageUtil.setOneAsFirstPageNo();

int start = PageUtil.getStart(pageNumber, pageSize);

int end = Math.min(start + pageSize, totalRow);

List<Order> subList = new ArrayList<>();

if (start < end) {

subList = orderList.subList(start, end);

}

return new BasePage<>(subList, pageNumber, pageSize, totalPage, totalRow);

}

结果：



### **4.9.2**Mongo数据库实现

接口实现思路：

OrdersEntity中间类，用来映射数据库。shipperId查询条件直接装入filterList过滤条件，通过多个if条件来判断有无对应的查询条件，再装入filterList，查询得到结果。从查询条件中获取每页的记录数、页数，查询结果中得到总记录数，计算总页数，查询结果中存储要显示的记录。最后用mapstruct映射到对应的model类。封装成Pageable返回。

OrderMapper代码同findByOrderId的Mongo实现类。

实现类代码：

@Override

public Pageable<Order> pageByShipperId(Integer shipperId, OrderPageQueryParam orderPageQueryParam) {

LocalDateTime startDate = orderPageQueryParam.getStartDate();

LocalDateTime endDate = orderPageQueryParam.getEndDate();

String customerName = orderPageQueryParam.getCustomerName();

String shipperName = orderPageQueryParam.getShipperName();

String shipProvince = orderPageQueryParam.getShipProvince();

String shipCity = orderPageQueryParam.getShipCity();

String shipRegion = orderPageQueryParam.getShipRegion();

String employeeName = orderPageQueryParam.getEmployeeName();

ArrayList<Bson> filterList = new ArrayList<>();

Bson shipperIdFilter = Filters.eq("shipInfo.shipper.id", shipperId);

filterList.add(shipperIdFilter);

if (startDate!=null && endDate!=null) {

Bson startDateFilter = Filters.gte("orderDate", startDate);

Bson endDateFilter = Filters.lte("orderDate", endDate);

Bson orderDateFilter = Filters.and(startDateFilter, endDateFilter);

filterList.add(orderDateFilter);

}

if (customerName!=null && !customerName.isBlank()) {

Bson customerNameFilter = Filters.regex("customer.name",

String.format(".\*%s.\*", customerName), "i");

filterList.add(customerNameFilter);

}

if (shipperName!=null && !shipperName.isBlank()) {

Bson shipperNameFilter = Filters.regex("shipInfo.shipper.name",

String.format(".\*%s.\*", shipperName), "i");

filterList.add(shipperNameFilter);

}

if (shipProvince!=null && !shipProvince.isBlank()) {

Bson shipProvinceFilter = Filters.regex("shipInfo.shipProvince",

String.format(".\*%s.\*", shipProvince), "i");

filterList.add(shipProvinceFilter);

}

if (shipCity!=null && !shipCity.isBlank()) {

Bson shipCityFilter = Filters.regex("shipInfo.shipCity",

String.format(".\*%s.\*", shipCity), "i");

filterList.add(shipCityFilter);

}

if (shipRegion!=null && !shipRegion.isBlank()) {

Bson shipRegionFilter = Filters.regex("shipInfo.shipRegion",

String.format(".\*%s.\*", shipRegion), "i");

filterList.add(shipRegionFilter);

}

if (employeeName!=null && !employeeName.isBlank()) {

Bson employeeNameFilter = Filters.regex("employee.name",

String.format(".\*%s.\*", employeeName), "i");

filterList.add(employeeNameFilter);

}

MongoCollection<OrdersEntity> ordersCollection = db.getCollection("orders", OrdersEntity.class);

Bson allFilters = filterList.isEmpty() ? Filters.empty() : Filters.and(filterList);

long totalRow = ordersCollection.countDocuments(allFilters);

int pageSize = orderPageQueryParam.getPageSize();

int pageNumber = orderPageQueryParam.getPageNumber();

PageUtil.setOneAsFirstPageNo();

int totalPage = PageUtil.totalPage(totalRow, pageSize);

int offset = PageUtil.getStart(pageNumber, pageSize);

ArrayList<OrdersEntity> orderEntityList = ordersCollection.find(allFilters)

.skip(offset).limit(pageSize)

.projection(Projections.exclude("details"))

.into(new ArrayList<>());

List<Order> orderList = orderMapper.toOrderList(orderEntityList);

BasePage<Order> orderPage = new BasePage<>(orderList, pageNumber, pageSize, totalPage, (int) totalRow);

return orderPage;

}

查询结果与pg数据库的相同。

## 4.10员工查询接口

### **4.10.1**关系数据库实现

接口实现思路：通过多个if条件来判断有无对应的查询条件。从查询条件中获取每页的记录数、页数。计算总页数、总记录数、要显示的记录。最后封装成Pageable返回。

SQL语句：

SELECT tem.id, name, birth\_date, hire\_date, title, region, phone

FROM t\_employee tem

WHERE

/\*%if @isNotBlank(name) \*/

tem.name ILIKE '%' || /\* name \*/'马' || '%'

/\*%end\*/

/\*%if @isNotBlank(gender) \*/

AND tem.gender ILIKE '%' || /\* gender \*/'男' || '%'

/\*%end\*/

/\*%if @isNotBlank(region) \*/

AND tem.region ILIKE '%' || /\* region \*/'北' || '%'

/\*%end\*/

/\*%if @isNotBlank(title) \*/

AND tem.title ILIKE '%' || /\* title \*/'代表' || '%'

/\*%end\*/

Dao方法：

@Select

List<EmployeeEntity> page(String name, String gender, String region, String title);

接口实现方法：

public Pageable<Employee> page(EmployeePageQueryParam employeePageQueryParam) {

List<EmployeeEntity> employeeEntityList = employeeDao.page(employeePageQueryParam.getName(), employeePageQueryParam.getGender(),

employeePageQueryParam.getRegion(), employeePageQueryParam.getTitle());

List<Employee> employeeList = employeeMapper.toEmployeeList(employeeEntityList);

int totalRow = employeeList.size();

int pageSize = employeePageQueryParam.getPageSize();

int pageNumber = employeePageQueryParam.getPageNumber();

int totalPage = PageUtil.totalPage(totalRow, pageSize);

PageUtil.setOneAsFirstPageNo();

int start = PageUtil.getStart(pageNumber, pageSize);

int end = Math.min(start + pageSize, totalRow);

List<Employee> subList = new ArrayList<>();

if (start < end) {

subList = employeeList.subList(start, end);

}

return new BasePage<>(subList, pageNumber, pageSize, totalPage, totalRow);

}

结果：



### **4.10.2Mongo数据库实现**

接口实现思路：

用model中Employee类，直接映射数据库。通过多个if条件来判断有无对应的查询条件，再装入filterList，查询得到结果。从查询条件中获取每页的记录数、页数，查询结果中得到总记录数，计算总页数，查询结果中存储要显示的记录。封装成Pageable返回。

实现类代码：

@Inject

MongoDatabase db;

@Override

public Pageable<Employee> page(EmployeePageQueryParam employeePageQueryParam) {

String name = employeePageQueryParam.getName();

String gender = employeePageQueryParam.getGender();

String region = employeePageQueryParam.getRegion();

String title = employeePageQueryParam.getTitle();

ArrayList<Bson> filterList = new ArrayList<>();

if (name!=null && !name.isBlank()) {

Bson nameFilter = Filters.regex("name",

String.format(".\*%s.\*", name), "i");

filterList.add(nameFilter);

}

if (gender!=null && !gender.isBlank()) {

Bson genderFilter = Filters.regex("gender",

String.format(".\*%s.\*", gender), "i");

filterList.add(genderFilter);

}

if (region!=null && !region.isBlank()) {

Bson regionFilter = Filters.regex("region",

String.format(".\*%s.\*", region), "i");

filterList.add(regionFilter);

}

if (title!=null && !title.isBlank()) {

Bson titleFilter = Filters.regex("title",

String.format(".\*%s.\*", title), "i");

filterList.add(titleFilter);

}

MongoCollection<Employee> employeesCollection = db.getCollection("employee", Employee.class);

Bson allFilters = filterList.isEmpty() ? Filters.empty() : Filters.and(filterList);

long totalRow = employeesCollection.countDocuments(allFilters);

int pageSize = employeePageQueryParam.getPageSize();

int pageNumber = employeePageQueryParam.getPageNumber();

PageUtil.setOneAsFirstPageNo();

int totalPage = PageUtil.totalPage(totalRow, pageSize);

int offset = PageUtil.getStart(pageNumber, pageSize);

ArrayList<Employee> employeeList = employeesCollection.find(allFilters)

.skip(offset).limit(pageSize)

.into(new ArrayList<>());

BasePage<Employee> employeePage = new BasePage<>(employeeList, pageNumber, pageSize, totalPage, (int) totalRow);

return employeePage;

}

查询结果与pg数据库的相同。

## 4.11客户查询接口

### **4.11.1**关系数据库实现

接口实现思路：通过多个if条件来判断有无对应的查询条件。从查询条件中获取每页的记录数、页数。计算总页数、总记录数、要显示的记录。最后封装成Pageable返回。

SQL语句：

SELECT tcu.id, name, contact, province, city, address, region, phone

FROM t\_customer tcu

WHERE

/\*%if @isNotBlank(name) \*/

tcu.name ILIKE '%' || /\* name \*/'广' || '%'

/\*%end\*/

/\*%if @isNotBlank(region) \*/

AND tcu.region ILIKE '%' || /\* region \*/'东' || '%'

/\*%end\*/

/\*%if @isNotBlank(province) \*/

AND tcu.province ILIKE '%' || /\* province \*/'东' || '%'

/\*%end\*/

/\*%if @isNotBlank(city) \*/

AND tcu.city ILIKE '%' || /\* city \*/'广' || '%'

/\*%end\*/

Dao方法：

@Select

List<CustomerEntity> page(String name, String region, String province, String city);

接口实现方法：

public Pageable<Customer> page(CustomerPageQueryParam customerPageQueryParam) {

List<CustomerEntity> customerEntityList = customerDao.page(customerPageQueryParam.getName(), customerPageQueryParam.getRegion(),

customerPageQueryParam.getProvince(), customerPageQueryParam.getCity());

List<Customer> customerList = customerMapper.toCustomerList(customerEntityList);

int totalRow = customerList.size();

int pageSize = customerPageQueryParam.getPageSize();

int pageNumber = customerPageQueryParam.getPageNumber();

int totalPage = PageUtil.totalPage(totalRow, pageSize);

PageUtil.setOneAsFirstPageNo();

int start = PageUtil.getStart(pageNumber, pageSize);

int end = Math.min(start + pageSize, totalRow);

List<Customer> subList = new ArrayList<>();

if (start < end) {

subList = customerList.subList(start, end);

}

return new BasePage<>(subList, pageNumber, pageSize, totalPage, totalRow);

}

结果：



### **4.11.2Mongo数据库实现**

接口实现思路：

用model中Customer类，直接映射数据库。通过多个if条件来判断有无对应的查询条件，再装入filterList，查询得到结果。从查询条件中获取每页的记录数、页数，查询结果中得到总记录数，计算总页数，查询结果中存储要显示的记录。封装成Pageable返回。

实现类代码：

@Inject

MongoDatabase db;

@Override

public Pageable<Customer> page(CustomerPageQueryParam customerPageQueryParam) {

String name = customerPageQueryParam.getName();

String region = customerPageQueryParam.getRegion();

String province = customerPageQueryParam.getProvince();

String city = customerPageQueryParam.getCity();

ArrayList<Bson> filterList = new ArrayList<>();

if (name!=null && !name.isBlank()) {

Bson nameFilter = Filters.regex("name",

String.format(".\*%s.\*", name), "i");

filterList.add(nameFilter);

}

if (region!=null && !region.isBlank()) {

Bson regionFilter = Filters.regex("region",

String.format(".\*%s.\*", region), "i");

filterList.add(regionFilter);

}

if (province!=null && !province.isBlank()) {

Bson provinceFilter = Filters.regex("province",

String.format(".\*%s.\*", province), "i");

filterList.add(provinceFilter);

}

if (city!=null && !city.isBlank()) {

Bson cityFilter = Filters.regex("city",

String.format(".\*%s.\*", city), "i");

filterList.add(cityFilter);

}

MongoCollection<Customer> customerCollection = db.getCollection("customer", Customer.class);

Bson allFilters = filterList.isEmpty() ? Filters.empty() : Filters.and(filterList);

long totalRow = customerCollection.countDocuments(allFilters);

int pageSize = customerPageQueryParam.getPageSize();

int pageNumber = customerPageQueryParam.getPageNumber();

PageUtil.setOneAsFirstPageNo();

int totalPage = PageUtil.totalPage(totalRow, pageSize);

int offset = PageUtil.getStart(pageNumber, pageSize);

ArrayList<Customer> customerList = customerCollection.find(allFilters)

.skip(offset).limit(pageSize)

.into(new ArrayList<>());

BasePage<Customer> customerPage = new BasePage<>(customerList, pageNumber, pageSize, totalPage, (int) totalRow);

return customerPage;

}

查询结果与pg数据库的相同。

## 4.12运货商查询接口

### **4.12.1**关系数据库实现

接口实现思路：通过多个if条件来判断有无对应的查询条件。从查询条件中获取每页的记录数、页数。计算总页数、总记录数、要显示的记录。最后封装成Pageable返回。

SQL语句：

SELECT tsh.id, name, phone

FROM t\_shipper tsh

WHERE

/\*%if @isNotBlank(name) \*/

tsh.name ILIKE '%' || /\* name \*/'通' || '%'

/\*%end\*/

Dao方法：

@Select

List<ShipperEntity> page(String name);

接口实现方法：

public Pageable<Shipper> page(ShipperPageQueryParam shipperPageQueryParam) {

List<ShipperEntity> shipperEntityList = shipperDao.page(shipperPageQueryParam.getName());

List<Shipper> shipperList = shipperMapper.toShipperList(shipperEntityList);

int totalRow = shipperList.size();

int pageSize = shipperPageQueryParam.getPageSize();

int pageNumber = shipperPageQueryParam.getPageNumber();

int totalPage = PageUtil.totalPage(totalRow, pageSize);

PageUtil.setOneAsFirstPageNo();

int start = PageUtil.getStart(pageNumber, pageSize);

int end = Math.min(start + pageSize, totalRow);

List<Shipper> subList = new ArrayList<>();

if (start < end) {

subList = shipperList.subList(start, end);

}

return new BasePage<>(subList, pageNumber, pageSize, totalPage, totalRow);

}

结果：



### **4.12.2Mongo数据库实现**

接口实现思路：

用model中Shipper类，直接映射数据库。通过多个if条件来判断有无对应的查询条件，再装入filterList，查询得到结果。从查询条件中获取每页的记录数、页数，查询结果中得到总记录数，计算总页数，查询结果中存储要显示的记录。封装成Pageable返回。

实现类代码：

@Inject

MongoDatabase db;

@Override

public Pageable<Shipper> page(ShipperPageQueryParam shipperPageQueryParam) {

String name = shipperPageQueryParam.getName();

Bson nameFilter = Filters.empty();

if (name!=null && !name.isBlank()) {

nameFilter = Filters.regex("name",

String.format(".\*%s.\*", name), "i");

}

MongoCollection<Shipper> shipperCollection = db.getCollection("shipper", Shipper.class);

long totalRow = shipperCollection.countDocuments(nameFilter);

int pageSize = shipperPageQueryParam.getPageSize();

int pageNumber = shipperPageQueryParam.getPageNumber();

PageUtil.setOneAsFirstPageNo();

int totalPage = PageUtil.totalPage(totalRow, pageSize);

int offset = PageUtil.getStart(pageNumber, pageSize);

ArrayList<Shipper> shipperList = shipperCollection.find(nameFilter)

.skip(offset).limit(pageSize)

.into(new ArrayList<>());

BasePage<Shipper> shipperPage = new BasePage<>(shipperList, pageNumber, pageSize, totalPage, (int) totalRow);

return shipperPage;

}

## 查询结果与pg数据库的相同。

## 4.13供应商查询接口

### **4.13.1**关系数据库实现

接口实现思路：通过多个if条件来判断有无对应的查询条件。从查询条件中获取每页的记录数、页数。计算总页数、总记录数、要显示的记录。最后封装成Pageable返回。

SQL语句：

SELECT tsu.id, name, phone

FROM t\_supplier tsu

WHERE

/\*%if @isNotBlank(name) \*/

tsu.name ILIKE '%' || /\* name \*/'广' || '%'

/\*%end\*/

Dao方法：

@Select

List<SupplierEntity> page(String name);

接口实现方法：

public Pageable<Supplier> page(SupplierPageQueryParam supplierPageQueryParam) {

List<SupplierEntity> supplierEntityList = supplierDao.page(supplierPageQueryParam.getName());

List<Supplier> supplierList = supplierMapper.toSupplierList(supplierEntityList);

int totalRow = supplierList.size();

int pageSize = supplierPageQueryParam.getPageSize();

int pageNumber = supplierPageQueryParam.getPageNumber();

int totalPage = PageUtil.totalPage(totalRow, pageSize);

PageUtil.setOneAsFirstPageNo();

int start = PageUtil.getStart(pageNumber, pageSize);

int end = Math.min(start + pageSize, totalRow);

List<Supplier> subList = new ArrayList<>();

if (start < end) {

subList = supplierList.subList(start, end);

}

return new BasePage<>(subList, pageNumber, pageSize, totalPage, totalRow);

}

结果：



### **4.13.2Mongo数据库实现**

接口实现思路：

用model中Supplier类，直接映射数据库。通过多个if条件来判断有无对应的查询条件，再装入filterList，查询得到结果。从查询条件中获取每页的记录数、页数，查询结果中得到总记录数，计算总页数，查询结果中存储要显示的记录。封装成Pageable返回。

实现类代码：

@Inject

MongoDatabase db;

@Override

public Pageable<Supplier> page(SupplierPageQueryParam supplierPageQueryParam) {

String name = supplierPageQueryParam.getName();

Bson nameFilter = Filters.empty();

if (name!=null && !name.isBlank()) {

nameFilter = Filters.regex("name",

String.format(".\*%s.\*", name), "i");

}

MongoCollection<Supplier> supplierCollection = db.getCollection("supplier", Supplier.class);

long totalRow = supplierCollection.countDocuments(nameFilter);

int pageSize = supplierPageQueryParam.getPageSize();

int pageNumber = supplierPageQueryParam.getPageNumber();

PageUtil.setOneAsFirstPageNo();

int totalPage = PageUtil.totalPage(totalRow, pageSize);

int offset = PageUtil.getStart(pageNumber, pageSize);

ArrayList<Supplier> supplierList = supplierCollection.find(nameFilter)

.skip(offset).limit(pageSize)

.into(new ArrayList<>());

BasePage<Supplier> supplierPage = new BasePage<>(supplierList, pageNumber, pageSize, totalPage, (int) totalRow);

return supplierPage;

}

## 查询结果与pg数据库的相同。

## 4.14根据产品等级查询产品接口

### **4.14.1**关系数据库实现

接口实现思路：根据level值查询，在对返回的结果手动调整对应等级的id、name。最后返回List<Category>

SQL语句：

SELECT id AS id3, name AS name3, parent\_id, level

FROM t\_category

WHERE level = /\* level \*/1

Dao方法：

@Select

List<CategoryEntity> findByLevel(Short level);

接口实现方法：

public List<Category> findByLevel(Short level) {

List<CategoryEntity> categoryEntityList = categoryDao.findByLevel(level);

List<Category> categoryList = categoryMapper.toCategoryList(categoryEntityList);

ListIterator<Category> categoryListIterator = categoryList.listIterator();

while (categoryListIterator.hasNext()) {

Category category = categoryListIterator.next();

if (level == 2) {

category.setId2(category.getId3());

category.setId3(null);

category.setName2(category.getName3());

category.setName3(null);

} else if (level == 1) {

category.setId1(category.getId3());

category.setId3(null);

category.setName1(category.getName3());

category.setName3(null);

}

}

return categoryList;

}

结果：



### **4.14.2**Mongo数据库实现

接口实现思路：

SimpleCategoryEntity中间类，用来映射数据库。产品级别level作为过滤条件。查询出ArrayList<SimpleCategoryEntity>类型的记录。最后用mapstruct映射到对应的model类。

CategoryMapper代码：

default Category toCategory(SimpleCategoryEntity simpleCategoryEntity) {

if (simpleCategoryEntity == null) {

return null;

}

Category category = new Category();

Integer idTemp = simpleCategoryEntity.getId();

if (idTemp != null) {

String id = idTemp.toString();

int length = id.length();

if (length % 2 == 1) { //奇数

id = "0" + id;

length++;

}

if (length == 2) {

category.setId1(id);

category.setName1(simpleCategoryEntity.getName());

} else if (length == 4) {

category.setId2(id);

category.setParentId(id.substring(2));

category.setName2(simpleCategoryEntity.getName());

} else if (6 == length) {

category.setId3(id);

category.setParentId(id.substring(2));

category.setName3(simpleCategoryEntity.getName());

}

}

Integer level = simpleCategoryEntity.getLevel();

if (level != null) {

category.setLevel(level.shortValue());

}

return category;

}

List<Category> toCategoryList(List<SimpleCategoryEntity> simpleCategoryEntityList);

实现类代码：

@Override

public List<Category> findByLevel(Short level) {

Bson levelFilter = Filters.eq("level", level);

MongoCollection<SimpleCategoryEntity> categoryCollection = db.getCollection("category", SimpleCategoryEntity.class);

ArrayList<SimpleCategoryEntity> simpleCategoryEntityList = categoryCollection.find(levelFilter).into(new ArrayList<>());

List<Category> categoryList = categoryMapper.toCategoryList(simpleCategoryEntityList);

return categoryList;

}

查询结果与pg数据库的相同。

## 4.15接口安全认证

### **4.15.1**安全认证流程

### **4.15.2**安全认证实现

# **5**数据接口应用

## 5.1主页面

### **5.1.1**使用目的

了解系统的版本等基础信息，导航到其它功能的目录。

### **5.1.2前端主要代码**

<div class="jumbotron top-main-container">  
 <div class="clearfix">  
 <a class="btn text-muted float-right" href="#">退出登录</a>  
 <h1 class="display-4 float-left">X公司货物管理系统</h1>  
 </div>  
 <p class="lead">  
 有以下功能模块：系统信息、订单数据查询、订单管理、产品管理</p>  
 <hr class="my-4">  
 <p id="author-info">/\*----作者信息---姓名: ----学号: ----年级: 级----班级: \*/</p>  
 </div>  
 <hr>  
 <ul class="nav nav-tabs">  
  
 <!-- 系统信息 -->  
 <li class="nav-item">  
 <a aria-controls="System-message" aria-expanded="false" class="nav-link active" data-toggle="collapse" href="#System-message"  
 role="button">系统信息</a>  
 <ul class="flex-column card collapse" id="System-message" style="width: 18rem;">  
 <li class="nav-item card-body">  
 <h5 class="card-title">版本信息</h5>  
 <p class="card-text">Version: V1.0 更新时间: 2024.1.29</p>  
 <a class="btn btn-primary" href="#">敬请期待</a>  
 </li>  
 </ul>  
 </li>  
  
 <!-- 产品数据查询 -->  
 <li class="nav-item">  
 <a aria-controls="Products" aria-expanded="false" class="nav-link active" data-toggle="collapse" href="#Products"  
 role="button">产品查询</a>  
 <ul class="flex-column card collapse" id="Products" style="width: 18rem;">  
 <li class="nav-item card-body">  
 <h5 class="card-title">产品条件查询</h5>  
 <p class="card-text">查询符合条件的产品信息</p>  
 <a class="btn btn-primary" href="Product/productPage.html">查询</a>  
 </li>  
 </ul>  
 </li>  
  
 <!-- 订单数据查询-->  
 <li class="nav-item">  
 <a aria-controls="Orders" aria-expanded="false" class="nav-link active" data-toggle="collapse" href="#Orders"  
 role="button">订单查询</a>  
 <ul class="flex-column card collapse" id="Orders" style="width: 18rem;">  
 <li class="nav-item card-body">  
 <h5 class="card-title">订单条件查询</h5>  
 <p class="card-text">查询符合条件的订单信息。</p>  
 <a class="btn btn-primary" href="Order/orderPage.html">查询</a>  
 </li>  
 </ul>  
 </li>  
 </ul>  
</div>

### **5.1.3**应用效果



## 5.2产品查询接口

## **5.2.1**使用目的

通过设置查询条件，查询对应的产品

**5.2.2**接口调用<script src="../../bootstrap-4.6.2-dist/js/jquery-3.7.1.js"></script>  
<script src="../../bootstrap-4.6.2-dist/js/bootstrap.bundle.min.js"></script>  
<script src="../../bootstrap-4.6.2-dist/js/bootstrap.bundle.js"></script>  
<script src="../../bootstrap-4.6.2-dist/js/bootstrap.js"></script>  
<script>  
 // 更新产品列表  
 function updateProductList(data) {  
 var productList = $("#productList");  
  
 // 清空现有的列表内容  
 productList.empty();  
  
 // 遍历返回的数据，动态生成列表项  
 for (var i = 0; i < data.length; i++) {  
 var product = data[i];  
  
 // 创建列表项  
 var listItem = $("<li class='media-lg' >");  
  
 // 添加图片列  
 listItem.append("<div class='col-2'><img src='" + product.photoUrl + "' class='mr-3' alt='...' style='width: 64px; height: 64px;'></div>");  
  
 // 添加文字列  
 listItem.append("<div class='col-10 media-body'>");  
 listItem.append("<h5 class='mt-0 mb-1'>" + product.name + "</h5>");  
 listItem.append("<p>产品编号:" + product.id + ",品牌:" + product.brand + ",分类编号:" + product.categoryId + ",分类名称:" + product.categoryName + ",条形码:" + product.barCode + ",规格:" + product.quantityPerUnit + ",单价:" + product.unitPrice + ",库存:" + product.unitsInStock + ",销量:" + product.unitsSell + "</p>");  
 listItem.append("<hr>");  
 listItem.append("</div>");  
  
 // 将列表项添加到产品列表  
 productList.append(listItem);  
 }  
 }  
  
 // 更新导航栏  
 function updatePagination(pageNumber, totalPage) {  
 var pagination = $("#pagination");  
  
 // 清空现有的导航栏内容  
 pagination.empty();  
  
 // 添加上一页按钮  
 if (pageNumber > 1) {  
 pagination.append("<li class='page-item' id='prevPage'><a class='page-link' href='#' aria-label='Previous'><span aria-hidden='true'>&laquo;</span></a></li>");  
 } else {  
 pagination.append("<li class='page-item disabled'><span class='page-link' aria-hidden='true'>&laquo;</span></li>");  
 }  
 // 添加当前页按钮  
 pagination.append("<li class='page-item active'><a class='page-link' href='#'>" + pageNumber + "</a></li>");  
  
 // 添加下一页按钮  
 if (pageNumber < totalPage) {  
 pagination.append("<li class='page-item' id='nextPage'><a class='page-link' href='#' aria-label='Next'><span aria-hidden='true'>&raquo;</span></a></li>");  
 } else {  
 pagination.append("<li class='page-item disabled'><span class='page-link' aria-hidden='true'>&raquo;</span></li>");  
 }  
 // 绑定上一页和下一页按钮的点击事件  
 $("#prevPage").click(function () {  
 handlePageChange(pageNumber - 1);  
 });  
  
 $("#nextPage").click(function () {  
 handlePageChange(pageNumber + 1);  
 });  
 // 绑定页码按钮的点击事件  
 bindPageClickEvent(totalPage);  
 }  
  
 // 处理页面变化逻辑  
 function handlePageChange(newPage) {  
 // 在这里处理页面变化的逻辑  
 fetchDataAndRender(newPage);  
 }  
  
 // 绑定页码按钮的点击事件  
 function bindPageClickEvent(totalPage) {  
 // 为每个页码按钮添加点击事件  
 for (var i = 1; i <= totalPage; i++) {  
 $("#page" + i).click(function () {  
 // 在这里处理点击页码按钮的逻辑  
 var clickedPage = $(this).text(); // 获取点击的页码  
 fetchDataAndRender(clickedPage);  
 });  
 }  
 }  
  
 // 发起POST请求并更新页面  
 function fetchDataAndRender(pageNumber) {  
 var name = $("#inputName").val();  
 var brand = $("#inputBrand").val();  
 var categoryId = $("#inputCategoryId").val();  
 var available = $("#inputAvailable").val();  
 var startUnitPrice = $("#inputStartUnitPrice").val();  
 var endUnitPrice = $("#inputEndUnitPrice").val();  
  
 var requestData = {  
 name: name,  
 brand: brand,  
 categoryId: categoryId,  
 available: available,  
 startUnitPrice: startUnitPrice,  
 endUnitPrice: endUnitPrice,  
 pageNumber: pageNumber // 添加页码信息  
 };  
  
 $.ajax({  
 type: "POST",  
 url: "http://localhost:8080/api/v1/products",  
 contentType: "application/json",  
 data: JSON.stringify(requestData),  
 success: function (response) {  
 console.log("成功响应:", response);  
  
 // 更新产品列表和导航栏  
 updateProductList(response.data);  
 updatePagination(response.pageNumber, response.totalPage);  
 },  
 error: function (error) {  
 console.error("错误响应:", error);  
 }  
 });  
 }  
  
 // 在文档加载完成后调用，发起POST请求并更新页面  
 $(document).ready(function () {  
 // 监听表单提交事件  
 $("form").submit(function (event) {  
 // 阻止表单默认提交行为  
 event.preventDefault();  
  
 // 获取表单中的各个输入字段的值  
 var name = $("#inputName").val();  
 var brand = $("#inputBrand").val();  
 var categoryId = $("#inputCategoryId").val();  
 var availableProducts = $("#inputAvailable").val();  
 var startUnitPrice = $("#inputStartUnitPrice").val();  
 var endUnitPrice = $("#inputEndUnitPrice").val();  
 var pageNumber = 1;  
 // 构建请求参数对象  
 var requestData = {  
 name: name,  
 brand: brand,  
 categoryId: categoryId,  
 availableProducts: availableProducts,  
 startUnitPrice: startUnitPrice,  
 endUnitPrice: endUnitPrice,  
 pageNumber: pageNumber  
 };  
  
 // 发起POST请求  
 $.ajax({  
 type: "POST",  
 url: "http://localhost:8080/api/v1/products",  
 contentType: "application/json",  
 data: JSON.stringify(requestData),  
 success: function (response) {  
 // 处理成功响应  
 console.log("成功响应:", response);  
  
 // 更新产品列表和导航栏  
 updateProductList(response.data);  
 updatePagination(response.pageNumber, response.totalPage);  
 },  
 error: function (error) {  
 // 处理错误响应  
 console.error("错误响应:", error);  
 }  
 });  
 });  
 });  
</script>

### **5.2.3**应用效果







## 5.3订单查询接口

### **5.3.1**使用目的

通过订单查询条件，查询订单

### **5.3.2**接口调用

<script src="../../bootstrap-4.6.2-dist/js/jquery-3.7.1.js"></script>

<script src="../../bootstrap-4.6.2-dist/js/bootstrap.bundle.min.js"></script>

<script src="../../bootstrap-4.6.2-dist/js/bootstrap.bundle.js"></script>

<script src="../../bootstrap-4.6.2-dist/js/bootstrap.js"></script>

<script src="../../bootstrap-4.6.2-dist/js/bootstrap-datepicker.min.js"></script>

<script>

// 更新产品列表

function updateProductList(data) {

var orderList = $("#orderList");

// 清空现有的列表内容

orderList.empty();

// 遍历返回的数据，动态生成列表项

for (var i = 0; i < data.length; i++) {

var order = data[i];

// 创建列表项

var listItem = $("<li class='media-lg'>");

// 添加文字列

listItem.append("<div class='col-10 media-body'>");

listItem.append("<p>订单编号: " + order.id + "</p>");

listItem.append("<p>客户编号: " + order.customerId + " 客户: " + order.customerName + " 客户联系人: " + order.customerContact + " 联系电话: " + order.customerPhone + "</p>");

listItem.append("<p>下单时间: " + order.orderDate + " 发货时间: " + order.shipDate + " 收货时间: " + order.arrivalDate + " 收件人: " + order.contact + " 收货地址: " + order.shipAddress + " </p>");

listItem.append("<p>跟单业务员工号: " + order.employeeId + " 跟单业务员: " + order.employeeName + " 联系电话: " + order.employeePhone + "</p>");

listItem.append("<p>运货商编号: " + order.shipperId + " 运货商: " + order.shipperName + " 运费: " + order.freight + "元 联系电话: " + order.shipperPhone + "</p>");

listItem.append("<hr>");

listItem.append("</div>");

// 将列表项添加到产品列表

orderList.append(listItem);

}

}

// 更新导航栏

function updatePagination(pageNumber, totalPage) {

var pagination = $("#pagination");

// 清空现有的导航栏内容

pagination.empty();

// 添加上一页按钮

if (pageNumber > 1) {

pagination.append("<li class='page-item' id='prevPage'><a class='page-link' href='#' aria-label='Previous'><span aria-hidden='true'>&laquo;</span></a></li>");

} else {

pagination.append("<li class='page-item disabled'><span class='page-link' aria-hidden='true'>&laquo;</span></li>");

}

// 添加当前页按钮

pagination.append("<li class='page-item active'><a class='page-link' href='#'>" + pageNumber + "</a></li>");

// 添加下一页按钮

if (pageNumber < totalPage) {

pagination.append("<li class='page-item' id='nextPage'><a class='page-link' href='#' aria-label='Next'><span aria-hidden='true'>&raquo;</span></a></li>");

} else {

pagination.append("<li class='page-item disabled'><span class='page-link' aria-hidden='true'>&raquo;</span></li>");

}

// 绑定上一页和下一页按钮的点击事件

$("#prevPage").click(function () {

handlePageChange(pageNumber - 1);

});

$("#nextPage").click(function () {

handlePageChange(pageNumber + 1);

});

// 绑定页码按钮的点击事件

bindPageClickEvent(totalPage);

}

// 处理页面变化逻辑

function handlePageChange(newPage) {

// 在这里处理页面变化的逻辑

fetchDataAndRender(newPage);

}

// 绑定页码按钮的点击事件

function bindPageClickEvent(totalPage) {

// 为每个页码按钮添加点击事件

for (var i = 1; i <= totalPage; i++) {

$("#page" + i).click(function () {

// 在这里处理点击页码按钮的逻辑

var clickedPage = $(this).text(); // 获取点击的页码

fetchDataAndRender(clickedPage);

});

}

}

// 发起POST请求并更新页面

function fetchDataAndRender(pageNumber) {

var CustomerName = $("#inputCustomerName").val();

var ShipperName = $("#inputShipperName").val();

var ShipProvince = $("#inputShipProvince").val();

var ShipCity = $("#inputShipCity").val();

var ShipRegion = $("#inputShipRegion").val();

var EmployeeName = $("#inputEmployeeName").val();

var StartOrderDate = $("#inputStartOrderDate").val();

var EndOrderDate = $("#inputEndOrderDate").val();

var requestData = {

customerName: CustomerName,

shipperName: ShipperName,

shipProvince: ShipProvince,

shipCity: ShipCity,

shipRegion: ShipRegion,

employeeName: EmployeeName,

startDate: StartOrderDate,

endDate: EndOrderDate,

pageNumber: pageNumber // 添加页码信息

};

$.ajax({

type: "POST",

url: "http://localhost:8080/api/v1/orders",

contentType: "application/json",

data: JSON.stringify(requestData),

success: function (response) {

console.log("成功响应:", response);

// 更新产品列表和导航栏

updateProductList(response.data);

updatePagination(response.pageNumber, response.totalPage);

},

error: function (error) {

console.error("错误响应:", error);

}

});

}

// 在文档加载完成后调用，发起POST请求并更新页面

$(document).ready(function () {

// 监听表单提交事件

$("form").submit(function (event) {

// 阻止表单默认提交行为

event.preventDefault();

// 获取表单中的各个输入字段的值

var CustomerName = $("#inputCustomerName").val();

var ShipperName = $("#inputShipperName").val();

var ShipProvince = $("#inputShipProvince").val();

var ShipCity = $("#inputShipCity").val();

var ShipRegion = $("#inputShipRegion").val();

var EmployeeName = $("#inputEmployeeName").val();

var StartOrderDateInput = $("#inputStartOrderDate");

var EndOrderDateInput = $("#inputEndOrderDate");

if (!compareDates(StartOrderDateInput,EndOrderDateInput)) {

return;

}

var StartOrderDate = StartOrderDateInput.val();

var EndOrderDate = EndOrderDateInput.val();

// 检查是否只存在一个时间

if (!StartOrderDate && EndOrderDate) {

StartOrderDateInput.datepicker('setDate', new Date('2019-01-01'));

StartOrderDate = StartOrderDateInput.val();

}

if (StartOrderDate && !EndOrderDate) {

EndOrderDateInput.datepicker('setDate', new Date());

EndOrderDate = EndOrderDateInput.val();

}

var requestData = {

customerName: CustomerName,

shipperName: ShipperName,

shipProvince: ShipProvince,

shipCity: ShipCity,

shipRegion: ShipRegion,

employeeName: EmployeeName,

startDate: StartOrderDate,

endDate: EndOrderDate

};

// 发起POST请求

$.ajax({

type: "POST",

url: "http://localhost:8080/api/v1/orders",

contentType: "application/json",

data: JSON.stringify(requestData),

success: function (response) {

// 处理成功响应

console.log("成功响应:", response);

// 更新产品列表和导航栏

updateProductList(response.data);

updatePagination(response.pageNumber, response.totalPage);

},

error: function (error) {

// 处理错误响应

console.error("错误响应:", error);

}

});

});

});

$(document).ready(function () {

// 初始化日期选择器

$('.datepicker').datepicker({

format: 'yyyy-mm-dd 00:00:00', // 日期格式

autoclose: true, // 选择日期后自动关闭日期选择器

todayHighlight: true // 高亮显示当前日期

});

// 获取起始时间和截止时间的输入框元素

var startOrderDateInput = $('#inputStartOrderDate');

var endOrderDateInput = $('#inputEndOrderDate');

// 添加事件监听器

startOrderDateInput.on('changeDate', function () {

compareDates(startOrderDateInput,endOrderDateInput);

});

endOrderDateInput.on('changeDate', function () {

compareDates(startOrderDateInput,endOrderDateInput);

});

});

function compareDates(startOrderDateInput,endOrderDateInput) {

// 获取起始时间和截止时间的日期值

var startOrderDate = startOrderDateInput.datepicker('getDate');

var endOrderDate = endOrderDateInput.datepicker('getDate');

var currentDate = new Date();

var limitDate = new Date('2019-01-01');

if (startOrderDate && (startOrderDate < limitDate || startOrderDate > currentDate)) {

alert('起始时间必须在2019-01-01到当前日期之间！');

startOrderDateInput.val('');

return false;

}

if (endOrderDate && (endOrderDate < limitDate || endOrderDate > currentDate)) {

alert('截止时间必须在2019-01-01到当前日期之间！');

endOrderDateInput.val('');

return false;

}

if (startOrderDate && endOrderDate && startOrderDate > endOrderDate) {

alert('起始时间不能晚于截止时间！');

startOrderDateInput.val('');

endOrderDateInput.val('');

return false;

}

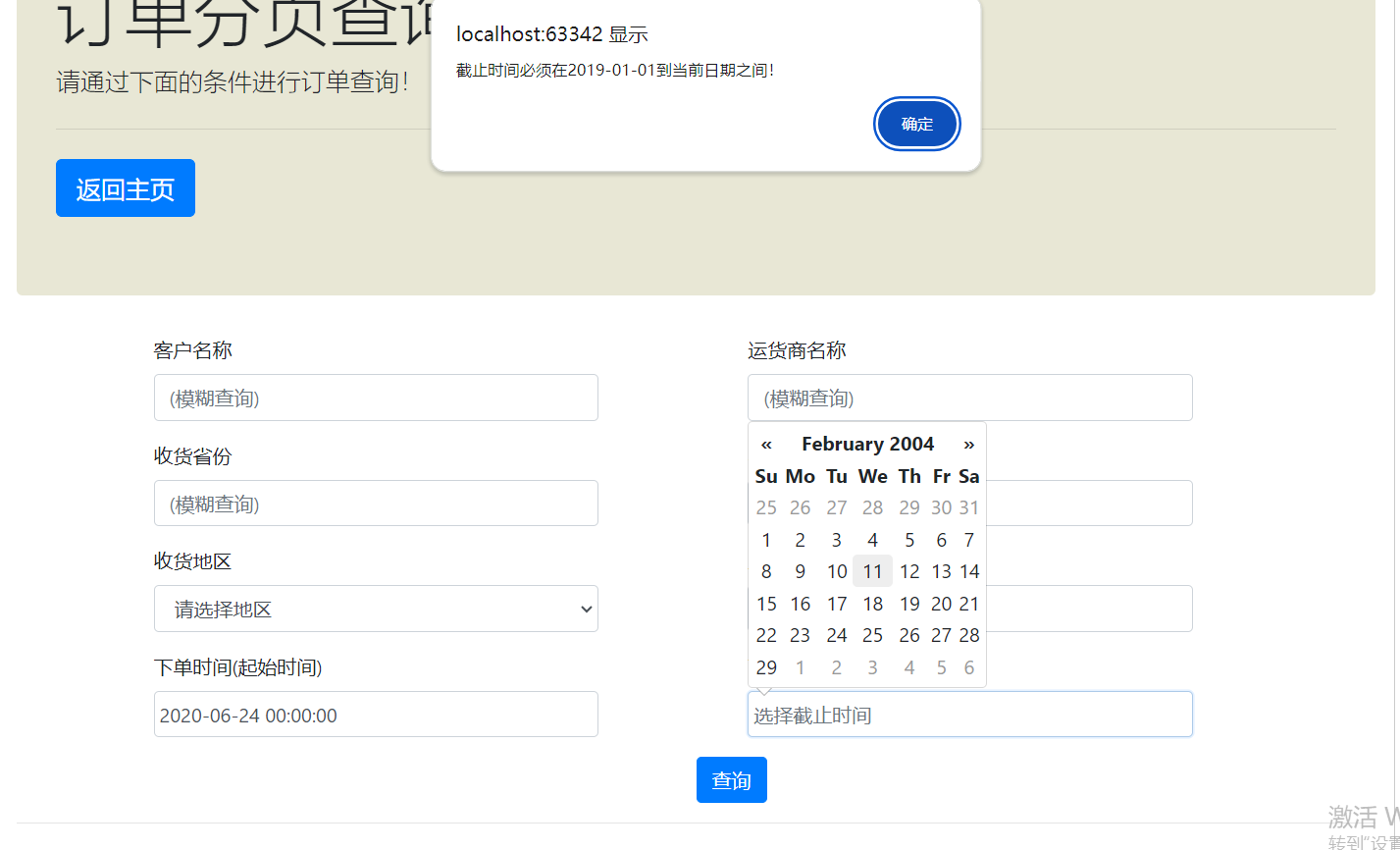
return true;

}

</script>

### 5.3.3应用效果





# **6**总结与展望

## 6.1课程总结（个人）

### **6.1.1**个人完成工作总结

我负责完成postgresql数据库产品接口和订单接口的实现，这两个接口涉及到的东西较多，同时也是给其它组员写接口作为的参考。为此我还得和负责model和service的组员协商，合理地定义model和service。同时在他们遇到后端数据库映射问题、用mapstruct从entity类到model类的映射问题、前端映射问题等各种问题时给予解决办法。还好依靠chatGPT、同学老师等帮助最终也解决了。工作量占25%，我当之无愧。

### **6.1.2**心得体会

1. 一开始参照着网页版的接口要求文档写，写到后面才发现对返回结果有要求，需要用到连接查询把其它表格的内容都查出来，导致我要改前面的sql语句。甚至修改Dao层和实现类。但好在软件工程的分层分块的概念，大部分只需要修改sql查询语句、model、entity这些底层和模块中的东西。
2. 通过本次课程，我了解到了postgresql与mysql类似，但可以单独用一个文件写sql语句，同时用注释标上自定义的方法来按条件执行对应的sql语句，但本人只会用老师写好的方法（例：/\*%if @isNotBlank(name) \*/）。对于多个数据库实现中，mapstruct映射确实方便，但有时要自定义写mapstruct方法，就体会不到mapstruct的方便之处。
3. 虽然直接用实战写软件的方式能提升的很快，但因为没有体系化的学习一些技术，导致只学了那门技术的皮毛。
4. 软件工程就是按照一定规范，借助一些工具，自定义一款软件。

## 6.2展望

### **6.2.1**对技术方法的评价

1.有时遇到报错也弄不明白哪里有问题，幸好有chatGPT给我解释和解决方法的提醒才能较为顺利的写出来。

2.因为根据productId查找产品信息时，要输出三级的产品分类，想了三种方法：

a.通过在实现类中用while循环将categoryId(如：“010106”)，切成三段(如：“01”、“0101”、“010106”)，传入三个参数，在sql语句的筛选条件中用OR连接三个参数来查，这样就能把分类相关的写到CategoryDao中，而不是杂糅在ProductDao中。但因为这样要写的代码较多，且如果产品是多个的话，要用for循环多次查找t\_category表，效率低。

b.改变ProductDao中findByProductId的sql语句，用递归的方式循环查找，但因为较复杂，且分类的层级关系明确（只有三级）。淘汰。

c.改变ProductDao中findByProductId的sql语句，用自连接的方式查找，虽然表连接要作笛卡尔积，但我需要查一次就能得到所有信息，写起来简单。

所以我选了c.方法。b.和c.方法都是参照CSDN博客大佬的文章，不得不说“姜还是老的辣”。

3.对于数据库和实体间的映射问题，一开始想用注解@Column来解决，但后来经chatGPT提示，在sql语句中用AS来取别名的方式让查询结果更清晰，同时也能保证正确的映射。

4.对于进货记录接口，因为要求返回的属性和Product中的属性有许多雷同，而且t\_purchase虽然是单独一个表，但他和Product关系很紧密，最终决定把进货记录接口放到ProductService中，返回的类也是Product。

5.一开始我是打算在实现类里对传入的非法时间进行处理，但后来组员写到前端，才发现应该应该写到javaScript中，对前端做出的非法行为进行处理，减少内部服务器的负担。

### 6.2.2 经验与教训

1.测试类还是要写。因为一般第一遍写的时候或多或少会有错误，而测试类是对第一小块的检验。

2.因为model中的实体类用的是驼峰写法，而entity中的实体类用的是蛇形写法，导致不能直接用mapstruct映射，只好都改为蛇形写法。但后面发现mongo数据库是驼峰写法，而且参考老师之前写的，它要通过命名来判断映射的，所以还是都改为驼峰写法。而postgresql用doma框架的@Entity(naming = NamingType.SNAKE\_LOWER\_CASE)。改的时候还遇到了mapstruct映射的奇怪问题。

3.@Inject与所处的框架有关，因为忘了加@QuarkusTest，导致识别不了注入。  
 4.因为之前老师讲课没用过时间的比较，导致查阅了许多LocalDateTime和timeStamp类型的文章才终于写出了关于下单日期查询的部分。与前端String类型的时间需要作如下映射处理：

@RegisterForReflection

public class OrderPageQueryParam extends PageQueryParam {

@JsonFormat(pattern = "yyyy-MM-dd HH:mm:ss.SSSSSS")

LocalDateTime startDate;  
传到sql语句中则可以直接用LocalDateTime作映射。

5.有的时候报错不知道怎么解决，或代码布置到怎么写。这时停止工作，出去走走玩玩，过段时间让脑子清醒后，就会突然意识到报错问题所在，知道该从何处着手写代码，在稍微想想，理清思路，就能一气呵成地写出来；或忽略这部分，先写其它部分，就能突然醒悟；或是向同学、老师求助，聊聊，最终就能解决看起来无解的问题。

6.对于既有默认映射，又有自定义的映射，mapstruct只能自定义映射方法，十分麻烦。但如果只有默认映射（包括@Mapping(source = "details", target = "detailList")）则很方便。