

苍穹外卖项目



黑马程序员 | 传智教育旗下
高端IT教育品牌
www.itheima.com



The screenshot shows the homepage of the management end. At the top, it displays "今日数据 2022.09.08" (Today's Data 2022.09.08). Below this are five key metrics: 营业额 (Revenue) ￥0, 有效订单 (Valid Orders) 0, 订单完成率 (Order Completion Rate) 0%, 平均客单价 (Average Order Value) ￥0, and 新增用户 (New Users) 0. Under the "订单管理" (Order Management) section, it shows 0 待接单 (Pending Pickup), 0 待派送 (Pending Delivery), 0 已完成 (Completed), 1 已取消 (Cancelled), and 1 全部订单 (All Orders). In the "菜品总览" (Food Category Overview) and "套餐总览" (Meal Plan Overview) sections, there are counts for 启售 (On Sale) and 停售 (Out of Stock) items, along with buttons for "新增菜品" (Add New Dish) and "新增套餐" (Add New Meal Plan).

管理端 - 外卖商家使用

The screenshot shows the user interface on a mobile device. It displays the restaurant information: 苍穹外卖 (The Sky Take-out) 营业中 (Open), 距离 1.5km, 配送费 6元, 预计时长 12min. Below this is a list of menu items with small images and details:

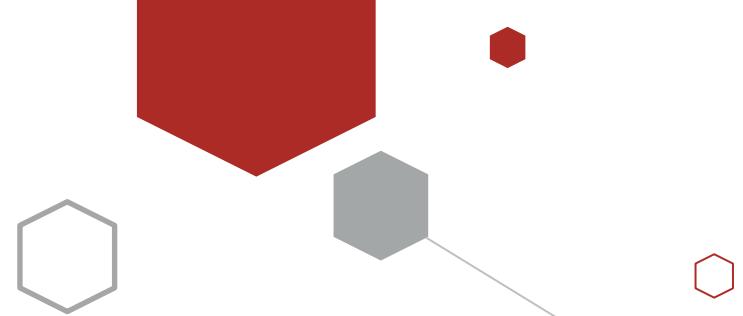
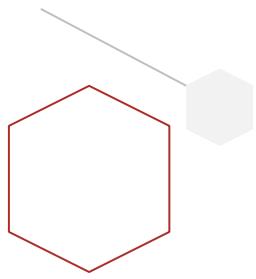
- 蜀味烤鱼 (Sichuan Flavor Grilled Fish) 草鱼2斤 (2 lbs Grass Carp) 原料: 草鱼, 黄豆芽, 莲藕 月销量0 ￥68.00 [选择规格] (Select Specification)
- 蜀味牛蛙 (Sichuan Flavor Frog Legs) 江团鱼2斤 (2 lbs Grass Carp) 配料: 江团鱼, 黄豆芽, 莲藕 月销量0 ￥119.00 [选择规格] (Select Specification)
- 特色蒸菜 (Special Steamed Dish) 鲈鱼2斤 (2 lbs Bass) 原料: 鲈鱼, 黄豆芽, 莲藕 月销量0 ￥72.00 [选择规格] (Select Specification)

A summary at the bottom shows ￥0.

用户端 - 点餐用户使用







项目概述、环境搭建



黑马程序员
www.itheima.com

传智教育旗下
高端IT教育品牌



目录

Contents

- ◆ 软件开发整体介绍
- ◆ 苍穹外卖项目介绍
- ◆ 开发环境搭建
- ◆ 导入接口文档
- ◆ Swagger

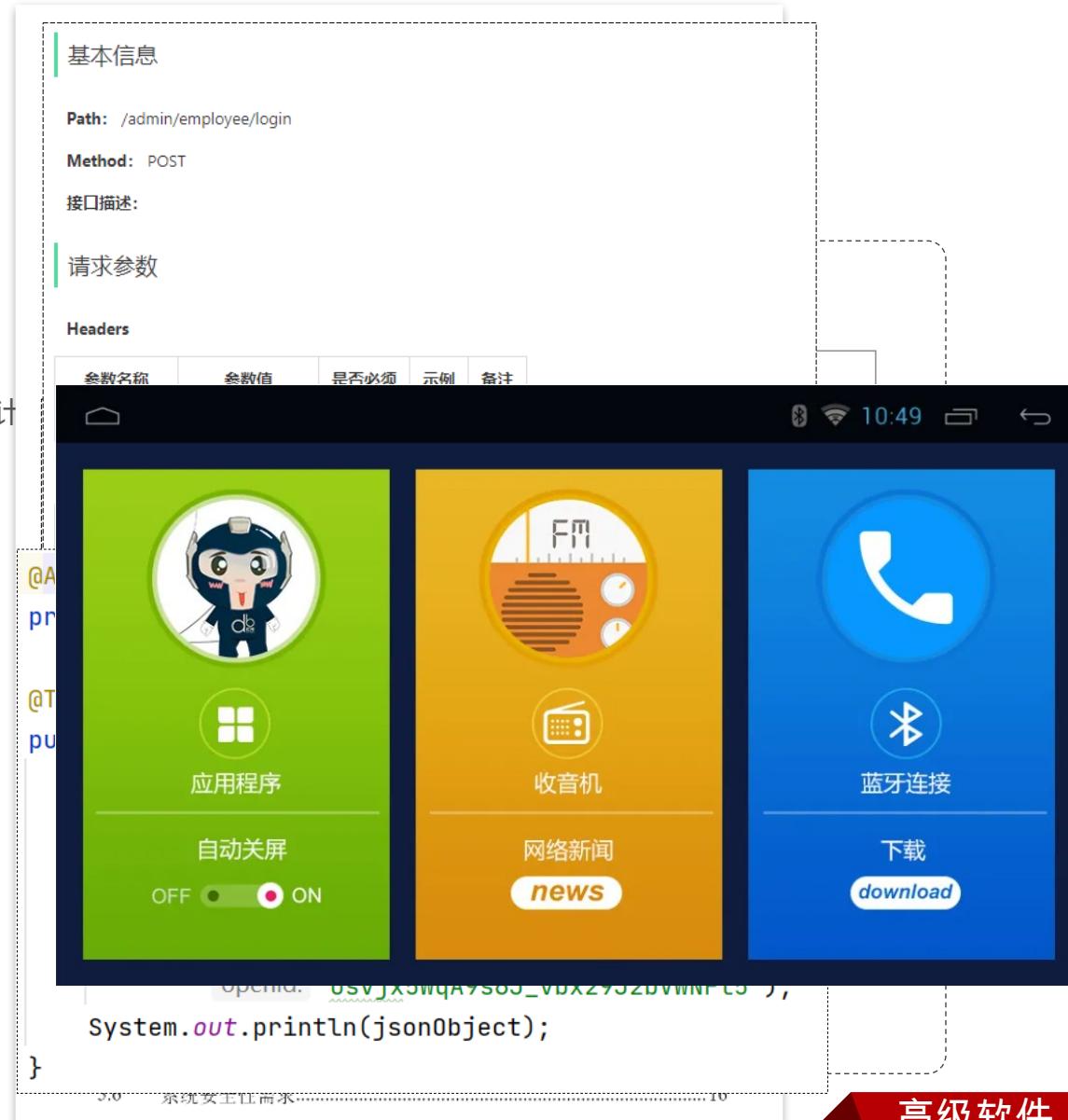
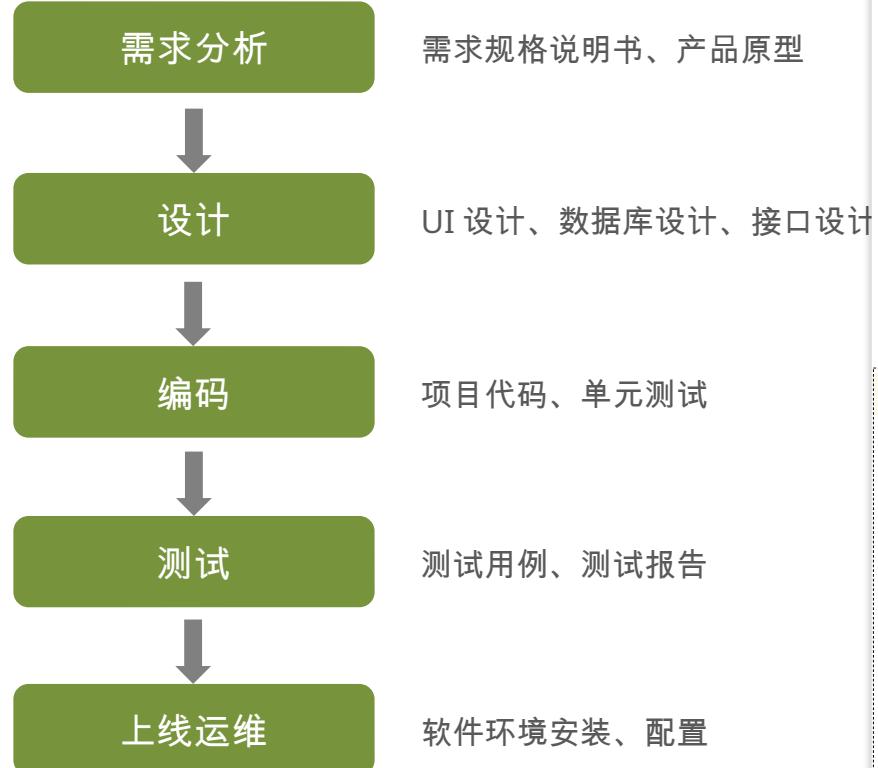


软件开发整体介绍

- 软件开发流程
- 角色分工
- 软件环境



软件开发流程





软件开发整体介绍

- 软件开发流程
- 角色分工
- 软件环境

角色分工

- 项目经理：对整个项目负责，任务分配、把控进度
- 产品经理：进行需求调研，输出需求调研文档、产品原型等
- UI 设计师：根据产品原型输出界面效果图
- 架构师：项目整体架构设计、技术选型等
- 开发工程师：代码实现
- 测试工程师：编写测试用例，输出测试报告
- 运维工程师：软件环境搭建、项目上线





软件开发整体介绍

- 软件开发流程
- 角色分工
- 软件环境

软件环境

- 开发环境 (development) : 开发人员在开发阶段使用的环境，一般外部用户无法访问
- 测试环境 (testing) : 专门给测试人员使用的环境，用于测试项目，一般外部用户无法访问
- 生产环境 (production) : 即线上环境，正式提供对外服务的环境





目录

Contents

- ◆ 软件开发整体介绍
- ◆ 苍穹外卖项目介绍
- ◆ 开发环境搭建
- ◆ 导入接口文档
- ◆ Swagger



苍穹外卖项目介绍

- 项目介绍
- 产品原型
- 技术选型



项目介绍

定位：专门为餐饮企业（餐厅、饭店）定制的一款软件产品

今日数据 2022.09.08

营业额 ¥ 0	有效订单 0	订单完成率 0%	平均客单价 ¥ 0	新增用户 0
------------	-----------	-------------	--------------	-----------

订单管理 2022.09.08

待接单 0	待派送 0	已完成 0	已取消 1	全部订单 1
----------	----------	----------	----------	-----------

菜品总览

已启售 24	已停售 0	新增菜品
-----------	----------	------

套餐总览

已启售 0	已停售 0	新增套餐
----------	----------	------

管理端 - 外卖商家使用

苍穹外卖 营业中

苍穹餐厅为顾客打造专业的大众化美食外送餐饮

北京市朝阳区新街大道一号楼8层

蜀味烤鱼 草鱼2斤 原料：草鱼，黄豆芽，莲藕 月销量0 ￥68.00 选择规格

蜀味牛蛙 新鲜时蔬 江团鱼2斤 配料：江团鱼，黄豆芽，莲藕 月销量0 ￥119.00 选择规格

特色蒸菜 水煮鱼 传统主食 鲈鱼2斤 原料：鲈鱼，黄豆芽，莲藕 月销量0 ￥72.00 选择规格

酒水饮料 汤类

用户端 - 点餐用户使用

项目介绍

功能架构：体现项目中的业务功能模块





苍穹外卖项目介绍

- 项目介绍
- 产品原型
- 技术选型



产品原型

产品原型：用于展示项目的业务功能，一般由产品经理进行设计

The image displays a product prototype for a food delivery management system. It consists of three mobile device screens showing different states of the service, and a sidebar on the right containing various management functions.

苍穹外卖LOGO

苍穹外卖_管理端

苍穹外卖_用户端

首页-未点餐

首页-已点餐

首页-休息中状态

管理员

查看详细数据 >

查看菜品管理 >

查看套餐管理 >

待派送 (10) 操作

单 拒单 查看

单 拒单 查看

单 拒单 查看

去结算

去结算

去结算

本店已打烊

工作台

订单管理

分类管理

菜品管理

套餐管理

数据统计

员工管理

精品热菜

精品套餐

爽口凉菜

特色点心

营养汤羹

冬季暖胃

家常主食

饮料酒水

佐餐小菜

秋日螃蟹

老鸭汤

龙井虾仁

桂花糯米糕

小米炖辽参

秋日螃蟹

老鸭汤

一壶龙井，最搭配绝美虾仁

月销657

¥998

散养二师兄，肉质有嚼劲

月销657

¥668

营养丰富，难以拒绝

月销657

¥488

来自老香港神秘味道

月销657

¥566

来自老香港神秘味道

月销657

¥866

散养二师兄，肉质有嚼劲

月销657

¥668

营养丰富，难以拒绝

月销657

¥488

来自老香港神秘味道

月销657

¥566

来自老香港神秘味道

月销657

¥866

一壶龙井，最搭配绝美虾仁

月销657

¥998

桂花糯米糕

散养二师兄，肉质有嚼劲

月销657

¥668

营养丰富，难以拒绝

月销657

¥488

秋日螃蟹

老鸭汤

龙井虾仁

桂花糯米糕

小米炖辽参

秋日螃蟹

老鸭汤

¥0

去结算

3

9:41

9:41

9:41

北京市朝阳区新街大道一号楼8层

北京市朝阳区新街大道一号楼8层

北京市朝阳区新街大道一号楼8层

瑞吉餐厅为顾客打造专业的大众化美食外送餐饮

瑞吉餐厅为顾客打造专业的大众化美食外送餐饮

瑞吉餐厅为顾客打造专业的大众化美食外送餐饮

距离1.5km 配送费6元 预计时长12min

距离1.5km 配送费6元 预计时长12min

距离1.5km 配送费6元 预计时长12min

去结算

去结算

去结算



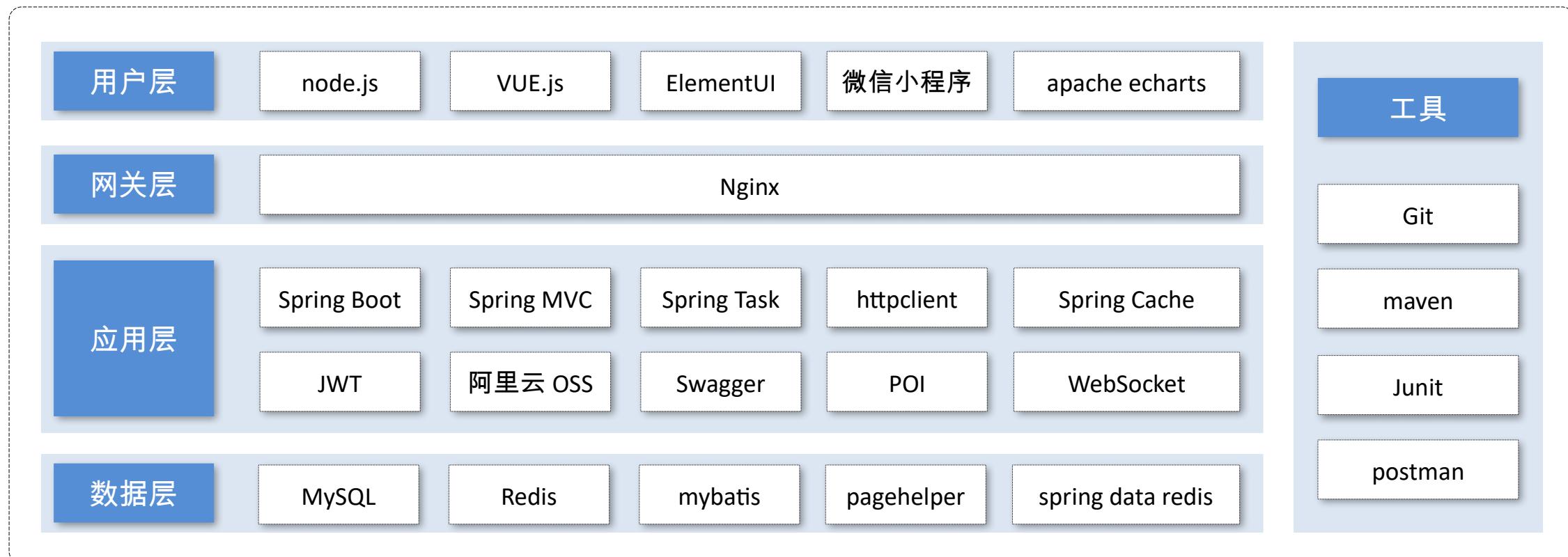
苍穹外卖项目介绍

- 项目介绍
- 产品原型
- 技术选型



技术选型

技术选型：展示项目中使用到的技术框架和中间件等





目录

Contents

- ◆ 软件开发整体介绍
- ◆ 苍穹外卖项目介绍
- ◆ 开发环境搭建
- ◆ 导入接口文档
- ◆ Swagger



开发环境搭建

- 前端环境搭建
- 后端环境搭建
- 完善登录功能

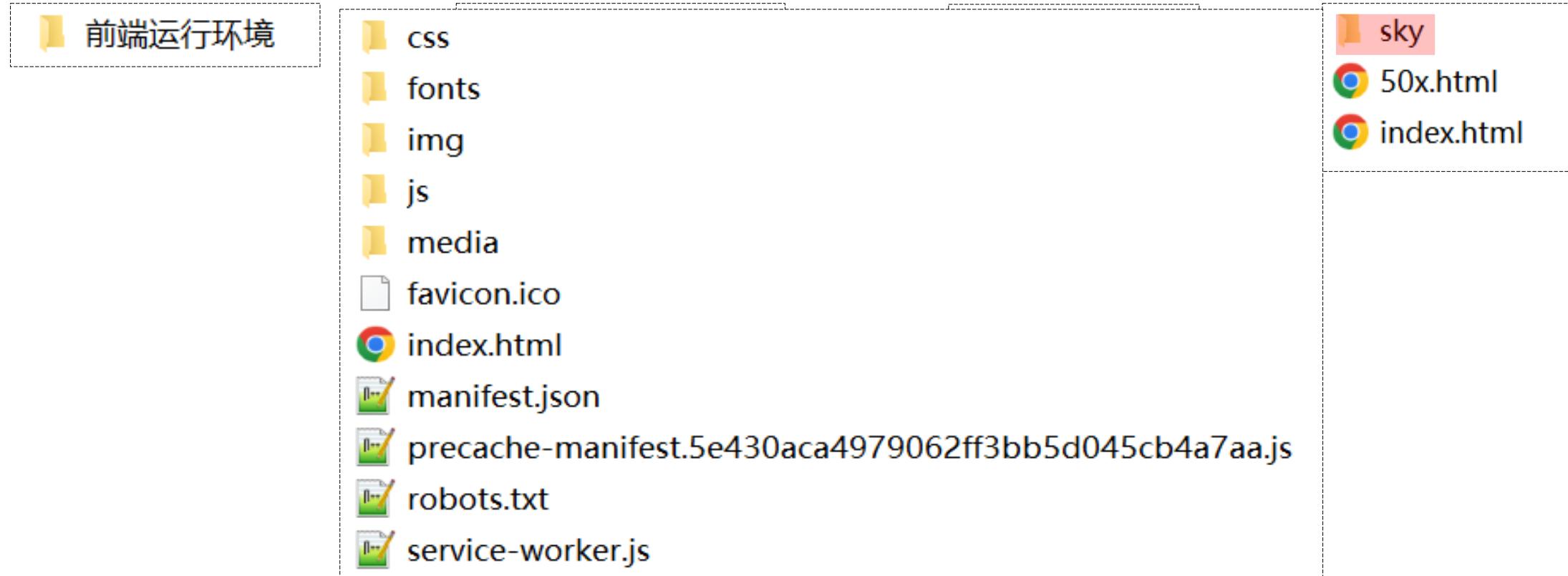
整体结构





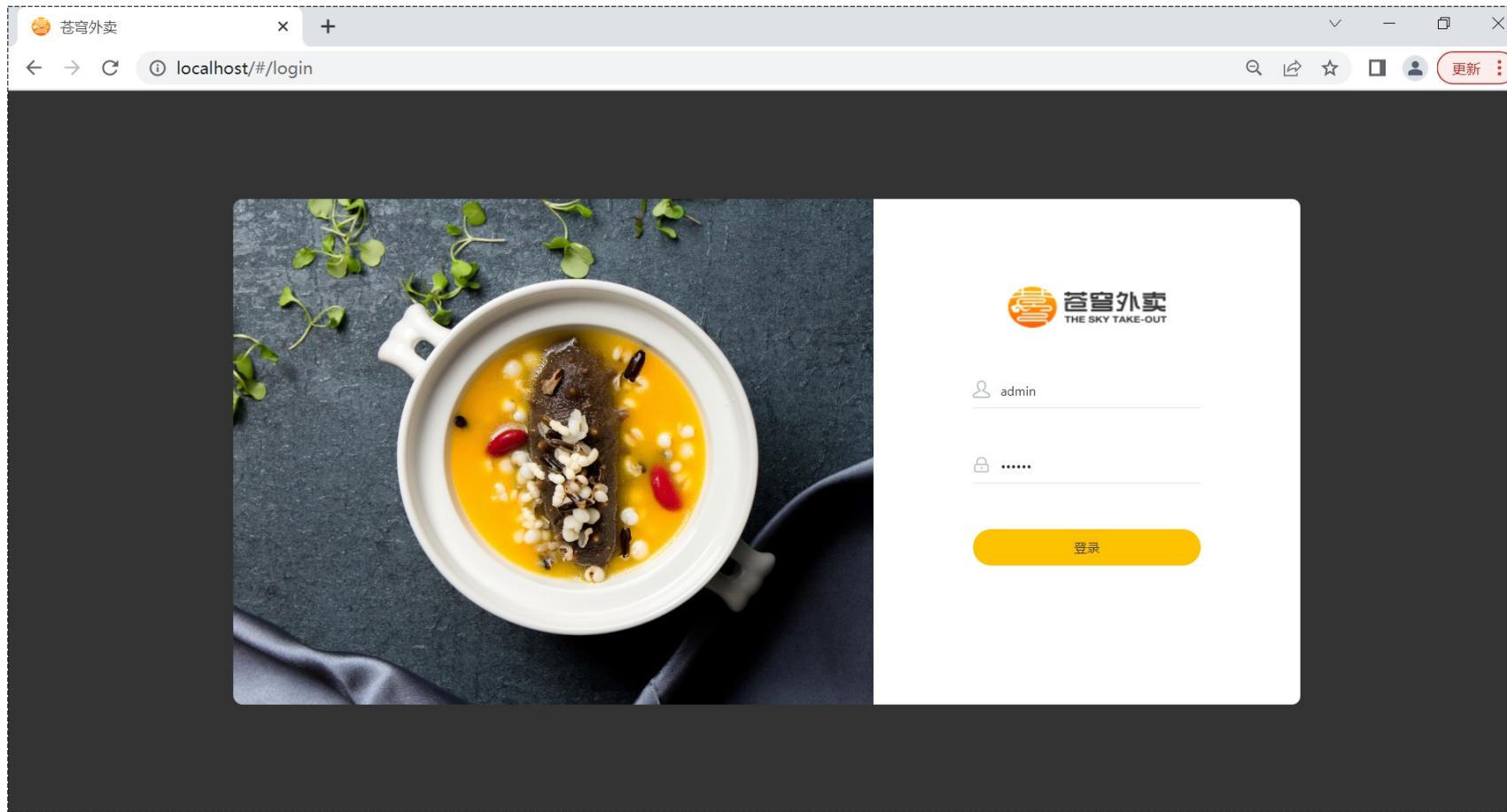
前端环境搭建

前端工程基于 **nginx** 运行



前端环境搭建

启动 nginx：双击 `nginx.exe` 即可启动 nginx 服务，访问端口号为 80





开发环境搭建

- 前端环境搭建
- 后端环境搭建
- 完善登录功能

后端环境搭建

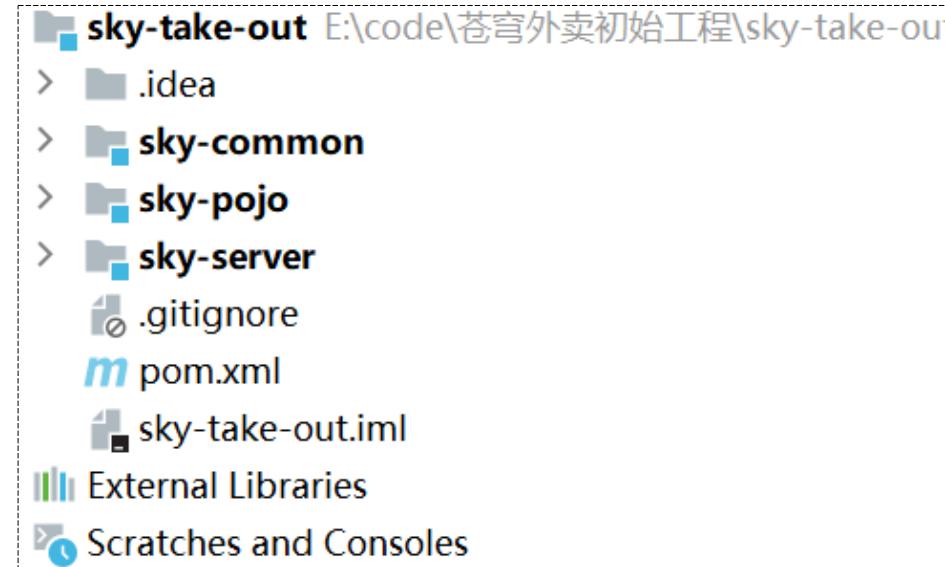
后端工程基于 **maven** 进行项目构建，并且进行**分模块**开发





后端环境搭建 – 熟悉项目结构

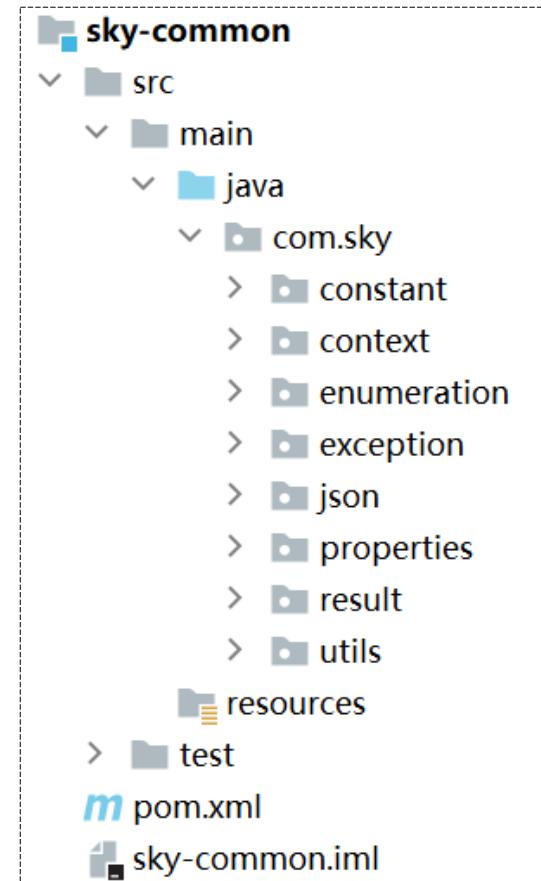
用 IDEA 打开初始工程，了解项目的整体结构



序号	名称	说明
1	sky-take-out	maven 父工程，统一管理依赖版本，聚合其他子模块
2	sky-common	子模块，存放公共类，例如：工具类、常量类、异常类等
3	sky-pojo	子模块，存放实体类、VO、DTO 等
4	sky-server	子模块，后端服务，存放配置文件、Controller、Service、Mapper 等

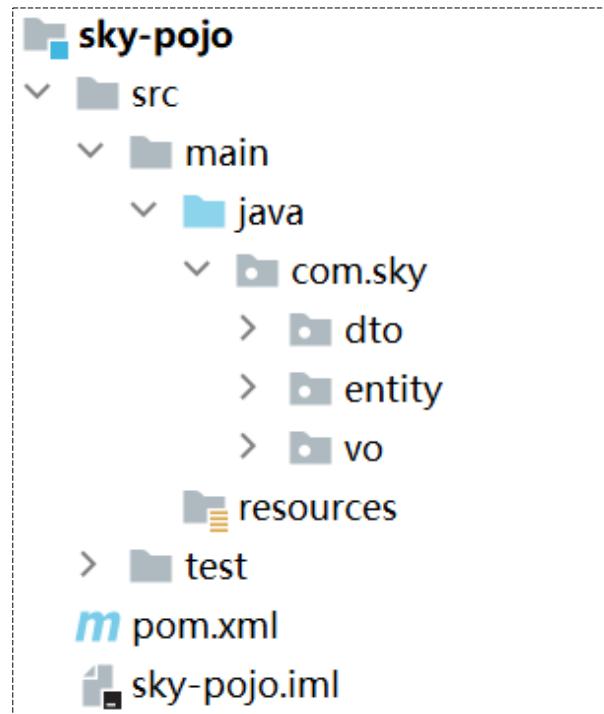
后端环境搭建 – 熟悉项目结构

sky-common 子模块中存放的是一些公共类，可以供其他模块使用



后端环境搭建 – 熟悉项目结构

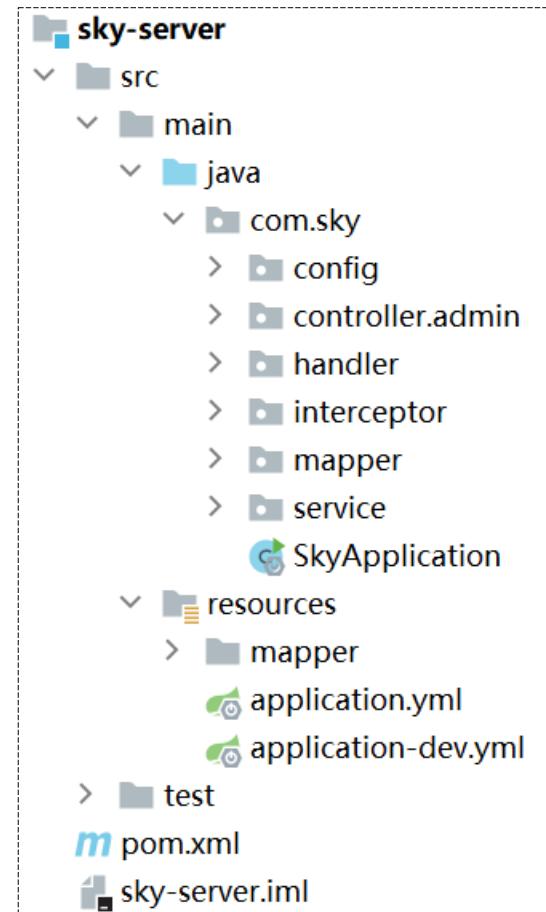
sky-pojo 子模块中存放的是一些 entity 、 DTO 、 VO



名称	说明
Entity	实体，通常和数据库中的表对应
DTO	数据传输对象，通常用于程序中各层之间传递数据
VO	视图对象，为前端展示数据提供的对象
POJO	普通 Java 对象，只有属性和对应的 getter 和 setter

后端环境搭建 – 熟悉项目结构

sky-server 子模块中存放的是 配置文件、配置类、拦截器、 controller 、 service 、 mapper 、启动类等



后端环境搭建 – 使用 Git 进行版本控制

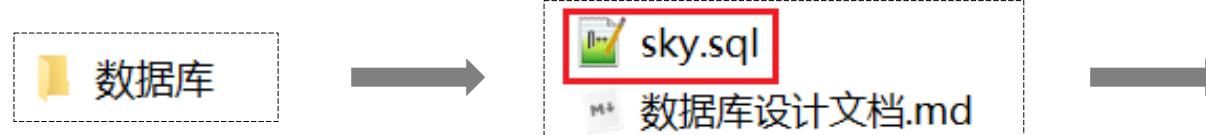
使用 Git 进行项目代码的版本控制，具体操作：

- 创建 Git 本地仓库
- 创建 Git 远程仓库
- 将本地文件推送到 Git 远程仓库



后端环境搭建 – 数据库环境搭建

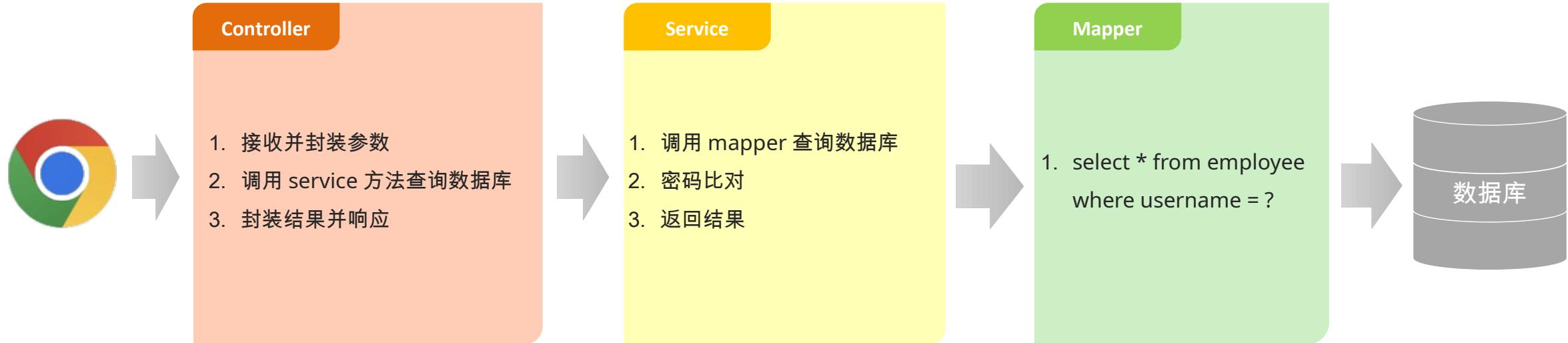
通过数据库建表语句创建数据库表结构：



序号	表名	中文名
1	employee	员工表
2	category	分类表
3	dish	菜品表
4	dish_flavor	菜品口味表
5	setmeal	套餐表
6	setmeal_dish	套餐菜品关系表
7	user	用户表
8	address_book	地址表
9	shopping_cart	购物车表
10	orders	订单表
11	order_detail	订单明细表

后端环境搭建 – 前后端联调

后端的初始工程中已经实现了登录功能，直接进行前后端联调测试即可



注：可以通过断点调试跟踪后端程序的执行过程



Request URL: http://localhost/api/employee/login
Request Method: POST
Status Code: 200
Remote Address: 127.0.0.1:80
Referrer Policy: strict-origin-when-cross-origin

前端发送的请求，是如何请求到后端服务的？

前端请求地址：http://localhost/api/employee/login

后端接口地址：http://localhost:8080/admin/employee/login

```
@RestController
@RequestMapping("/admin/employee")
@Slf4j
public class EmployeeController {

    @Autowired
    private EmployeeService employeeService;
    @Autowired
    private JwtProperties jwtProperties;

    /** 登录 ... */
    @PostMapping("/login")
    public Result<EmployeeLoginVO> login(@RequestBody EmployeeLoginDTO employeeLoginDTO) {...}
```



后端环境搭建 – 前后端联调

nginx 反向代理，就是将前端发送的动态请求由 nginx 转发到后端服务器

nginx 播报 编辑 讨论 上传视频

HTTP和反向代理web服务器

Nginx (engine x) 是一个高性能的HTTP和反向代理web服务器，同时也提供了IMAP/POP3/SMTP服务。Nginx是由伊戈尔·塞索耶夫为俄罗斯访问量第二的Rambler.ru站点（俄文：Рамблер）开发的，公开版本1.19.6发布于2020年12月15日。[\[12\]](#)

其将源代码以类BSD许可证的形式发布，因它的稳定性、丰富的功能集、简单的配置文件和低系统资源的消耗而闻名。2022年01月25日，nginx 1.21.6发布。[\[13\]](#)

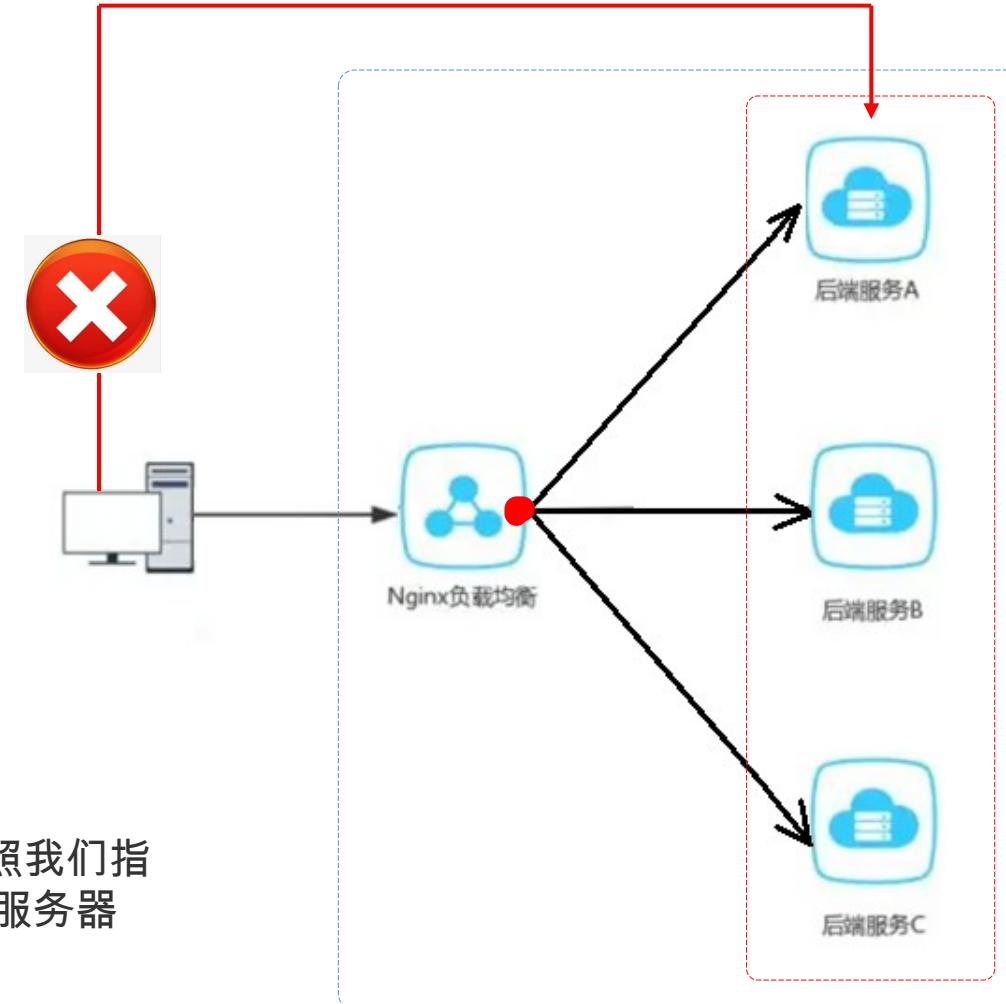
Nginx是一款轻量级的Web 服务器/反向代理服务器及电子邮件（IMAP/POP3）代理服务器，在BSD-like 协议下发行。其特点是占有内存少，并发能力强，事实上nginx的并发能力在同类型的网页服务器中表现较好。



后端环境搭建 – 前后端联调

nginx 反向代理的好处：

- 提高访问速度
- 进行负载均衡
- 保证后端服务安全



所谓**负载均衡**，就是把大量的请求按照我们指定的方式均衡的分配给集群中的每台服务器



后端环境搭建 – 前后端联调

nginx 反向代理的配置方式：

```
server{
    listen 80;
    server_name localhost;

    location /api/ {
        proxy_pass http://localhost:8080/admin/; # 反向代理
    }

}
```

nginx.conf



http://localhost/api/employee/login



http://localhost:8080/admin/employee/login





后端环境搭建 – 前后端联调

nginx 负载均衡的配置方式：

```
upstream webservers{
    server 192.168.100.128:8080;
    server 192.168.100.129:8080;
}

server{
    listen 80;
    server_name localhost;

    location /api/ {
        proxy_pass http://webservers/admin/; # 负载均衡
    }
}
```

nginx.conf



后端环境搭建 – 前后端联调

nginx 负载均衡策略：

名称	说明
轮询	默认方式
weight	权重方式，默认为 1，权重越高，被分配的客户端请求就越多
ip_hash	依据 ip 分配方式，这样每个访客可以固定访问一个后端服务
least_conn	依据最少连接方式，把请求优先分配给连接数少的后端服务
url_hash	依据 url 分配方式，这样相同的 url 会被分配到同一个后端服务
fair	依据响应时间方式，响应时间短的服务将会被优先分配



开发环境搭建

- 前端环境搭建
- 后端环境搭建
- 完善登录功能



完善登录功能

问题：员工表中的密码是明文存储，安全性太低。

	id	name	username	password	phone	sex	id_number	status	create_time	update_time	create_user	update_user
▶	1	管理员	admin	123456	13812312312	1	110101199001010047	1	2022-02-15 15:51:20	2022-02-17 09:16:20	10	1



思路

1. 将密码加密后存储，提高安全性
2. 使用 MD5 加密方式对明文密码加密

MD5信息摘要算法（英语：MD5 Message-Digest Algorithm），一种被广泛使用的[密码散列函数](#)，可以产生出一个128位（16字节）的散列值（hash value），用于确保信息传输完整一致。MD5由美国密码学家[罗纳德·李维斯特](#)（Ronald Linn Rivest）设计，于1992年公开，用以取代[MD4](#)算法。这套算法的程序在 RFC 1321 标准中被加以规范。1996年后该算法被证实存在弱点，可以被加以破解，对于需要高度安全性的数据，专家一般建议改用其他算法，如[SHA-2](#)。2004年，证实MD5算法无法防止碰撞（collision），因此不适用于安全性认证，如[SSL](#)公开密钥认证或是[数字签名](#)等用途。

123456

MD5 加密

e10adc3949ba59abbe56e057f20f883e



步骤

• 完善登录功能

1. 修改数据库中明文密码，改为 MD5 加密后的密文
2. 修改 Java 代码，前端提交的密码进行 MD5 加密后再跟数据库中密码比对

```
// 进行 md5 加密，然后再进行比对
password = DigestUtils.md5DigestAsHex(password.getBytes());
if (!password.equals(employee.getPassword())) {
    // 密码错误
    throw new PasswordErrorException(MessageConstant.PASSWORD_ERROR);
}
```

EmployeeServiceImpl



目录

Contents

- ◆ 软件开发整体介绍
- ◆ 苍穹外卖项目介绍
- ◆ 开发环境搭建
- ◆ 导入接口文档
- ◆ Swagger

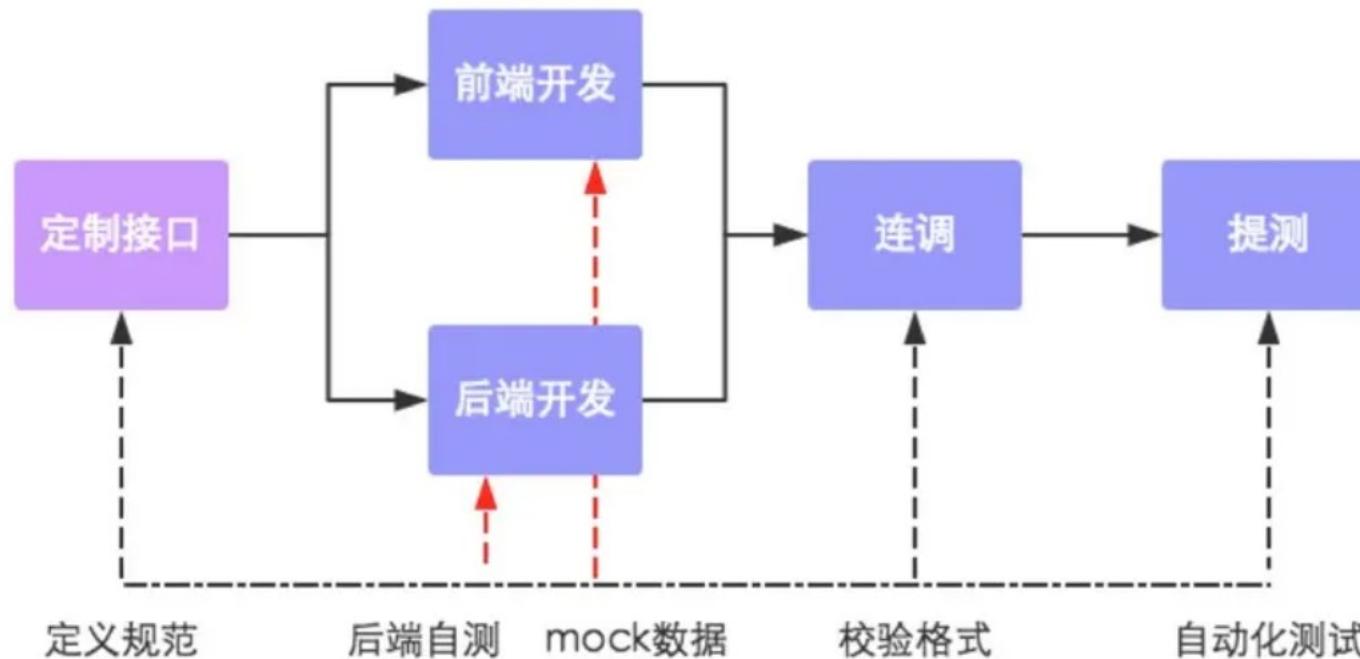
A red hexagonal icon containing the white text "04". To its left is a smaller, semi-transparent gray hexagon.

导入接口文档

- 前后端分离开发流程
- 操作步骤

说明

前后端分离开发流程：



A large red hexagon containing the white text "04". To its left is a smaller, semi-transparent light gray hexagon.

导入接口文档

- 前后端分离开发流程
- 操作步骤



操作步骤

将课程资料中提供的项目接口导入 YApi。

苍穹外卖-管理端接口.json
 苍穹外卖-用户端接口.json

The screenshot shows the YAPI interface with the following details:

个人空间 / 苍穹外卖-用户端接口

接口列表 (Interface List) - 25个接口

接口名称 (Interface Name)	接口路径 (Interface Path)	接口分类 (Interface Category)	状态 (Status)	tag
获取营业状态	GET /user/shop/status	C端-店铺操作接口	● 未完成	C端-店铺操作接口
历史订单查询	GET /user/order/historyOrders	C端-订单接口	● 未完成	C端-订单接口
设置默认地址	PUT /user/addressBook/default	C端-地址簿接口	● 未完成	C端-地址簿接口
根据套餐id查询包含的菜品	GET /user/setmeal/dish/{id}	C端-套餐浏览接口	● 未完成	C端-套餐浏览接口
订单支付	PUT /user/order/payment	C端-订单接口	● 未完成	C端-订单接口
根据分类id查询菜品	GET /user/dish/list	C端-菜品浏览接口	● 未完成	C端-菜品浏览接口
添加购物车	POST /user/shoppingCart/add	C端-购物车接口	● 未完成	C端-购物车接口



目录

Contents

- ◆ 软件开发整体介绍
- ◆ 苍穹外卖项目介绍
- ◆ 开发环境搭建
- ◆ 导入接口文档
- ◆ Swagger



Swagger

- 介绍
- 使用方式
- 常用注解

介绍

使用 Swagger 你只需要按照它的规范去定义接口及接口相关的信息，就可以做到生成接口文档，以及[在线接口调试](#)页面。
官网：<https://swagger.io/>

[Knife4j](#) 是为 Java MVC 框架集成 Swagger 生成 Api 文档的增强解决方案。

```
<dependency>
    <groupId>com.github.xiaoymin</groupId>
    <artifactId>knife4j-spring-boot-starter</artifactId>
    <version>3.0.2</version>
</dependency>
```

pom.xml



Swagger

- 介绍
- 使用方式
- 常用注解



步骤

• 使用方式

1. 导入 knife4j 的 maven 坐标
2. 在配置类中加入 knife4j 相关配置
3. 设置静态资源映射，否则接口文档页面无法访问

```
/*
 * 设置静态资源映射
 * @param registry
 */
protected void addResourceHandlers(ResourceHandlerRegistry registry) {
    log.info("开始设置静态资源映射 ...");
    registry.addResourceHandler("/doc.html").addResourceLocations("classpath:/META-INF/resources/");
    registry.addResourceHandler("/webjars/**").addResourceLocations("classpath:/META-INF/resources/webjars/");
}

/**
 * 指定生成接口需要扫描的包
 */
.apis(RequestHandlerSelectors.basePackage("com.sky.controller"))
    .paths(PathSelectors.any())
    .build();

return docket;
}
```

WebMvcConfiguration

WebMvcConfiguration



使用方式

接口文档访问路径为 `http://ip:port/doc.html`

The screenshot shows a web browser displaying the API documentation for the 'CangWan' delivery project. The URL in the address bar is `localhost:8080/doc.html#/home`. The page title is '苍穹外卖项目接口文档'. On the left, there is a sidebar with navigation links: 'default', '主页', 'Swagger Models', '文档管理', 'employee-controller' (with a '2' badge), 'POST login', and 'POST logout'. The main content area displays the following configuration information:

参数	值
简介	苍穹外卖项目接口文档
作者	
版本	2.0
host	localhost:8080
basePath	/
服务Url	
分组名称	default
分组Url	/v2/api-docs
分组location	/v2/api-docs
接口统计信息	POST 2

At the bottom of the page, it says 'Apache License 2.0 | Copyright © 2019-Knife4j'.



使用方式

接口测试

The screenshot shows the Knife4j Swagger UI interface for testing the 'employee-controller' module. The left sidebar lists available controllers: 'default', 'Swagger Models', '文档管理', and 'employee-controller'. Under 'employee-controller', the 'login' endpoint is selected, indicated by a blue background. The main area displays the '调试' (Debug) tab for a POST request to '/admin/employee/login'. The '请求参数' (Request Parameters) tab is active, showing raw JSON input:

```
1 {  
2   "password": "123456",  
3   "username": "admin"  
4 }
```

The 'Headers' section is empty. Below the request, the response content is shown in a code editor:

```
1 {  
2   "code": 1,  
3   "msg": null,  
4   "data": {  
5     "id": 1,  
6     "userName": "admin",  
7     "name": "管理员",  
8     "token": "eyJhbGciOiJIUzI1NiJ9.eyJlbXBZC16MSwiZXhwIjoxNjYzMjE2NjUzfQ.zmAq0EjaCnZ3-u5wqo2afMUwlmgmQyJ0U1CB6yL-Q"  
9   }  
10 }
```

At the bottom, it says 'Apache License 2.0 | Copyright © 2019-Knife4j'.



通过 Swagger 就可以生成接口文档，那么我们就不需要 Yapi 了？

- 1、Yapi 是设计阶段使用的工具，管理和维护接口
- 2、Swagger 在开发阶段使用的框架，帮助后端开发人员做后端的接口测试



Swagger

- 介绍
- 使用方式
- 常用注解



常用注解

通过注解可以控制生成的接口文档，使接口文档拥有更好的可读性，常用注解如下：

注解	说明
@Api	用在类上，例如 Controller，表示对类的说明
@ApiModelProperty	用在类上，例如 entity、 DTO、 VO
@ApiOperation	用在属性上，描述属性信息
@ApiOperation	用在方法上，例如 Controller 的方法，说明方法的用途、作用



传智教育旗下高端IT教育品牌



员工管理、分类管理



苍穹外卖 THE SKY TAKE-OUT

三 营业中

● 营业状态设置 管理员

工作台

数据统计

订单管理

套餐管理

菜品管理

分类管理

员工管理

分类名称:

分类类型:

查询

+ 新增菜品分类

+ 新增套餐分类

分类名称	分类类型	排序	状态	操作时间	操作
蜀味烤鱼	菜品分类	4	• 启用	2022-08-31 14:27	修改 删除 禁用
蜀味牛蛙	菜品分类	5	• 启用	2022-08-31 14:39	修改 删除 禁用
特色蒸菜	菜品分类	6	• 启用	2022-06-09 22:17	修改 删除 禁用
新鲜时蔬	菜品分类	7	• 启用	2022-06-09 22:18	修改 删除 禁用
水煮鱼	菜品分类	8	• 启用	2022-06-09 22:23	修改 删除 禁用
传统主食	菜品分类	9	• 启用	2022-06-09 22:18	修改 删除 禁用
酒水饮料	菜品分类	10	• 启用	2022-06-09 22:09	修改 删除 禁用
汤类	菜品分类	11	• 启用	2022-06-10 10:51	修改 删除 禁用
人气套餐	套餐分类	12	• 启用	2022-06-10 11:04	修改 删除 禁用
商务套餐	套餐分类	13	• 启用	2022-06-10 11:04	修改 删除 禁用



目录

Contents

- ◆ 新增员工
- ◆ 员工分页查询
- ◆ 启用禁用员工账号
- ◆ 编辑员工
- ◆ 导入分类模块功能代码



新增员工

- 需求分析和设计
- 代码开发
- 功能测试
- 代码完善



需求分析和设计

产品原型：

添加员工

* 账号： 账号必须是唯一的

* 员工姓名：

* 手机号： 手机号为合法的 11 位手机号码

* 性别： 男 女

身份证号： 身份证号为合法的 18 位身份证号码

密码默认为 123456

保存 保存并继续添加员工 返回

需求分析和设计

接口设计：

基本信息

Path: /admin/employee**Method:** POST

接口描述:

请求参数

Headers

参数名称	参数值	是否必须	示例	备注
Content-Type	application/json	是		

Body

名称	类型	是否必须	默认值	备注	其他信息
id	integer	非必须		员工id	format: int64
idNumber	string	必须		身份证证	
name	string	必须		姓名	
phone	string	必须		手机号	
sex	string	必须		性别	
username	string	必须		用户名	

返回数据

名称	类型	是否必须	默认值	备注	其他信息
code	integer	必须			format: int32
data	object	非必须			
msg	string	非必须			

本项目约定：

- 管理端发出的请求，统一使用 `/admin` 作为前缀
- 用户端发出的请求，统一使用 `/user` 作为前缀



需求分析和设计

数据库设计（ employee 表）：

字段名	数据类型	说明	备注
id	bigint	主键	自增
name	varchar(32)	姓名	
username	varchar(32)	用户名	唯一
password	varchar(64)	密码	
phone	varchar(11)	手机号	
sex	varchar(2)	性别	
id_number	varchar(18)	身份证号	
status	Int	账号状态	1 正常 0 锁定
create_time	Datetime	创建时间	
update_time	datetime	最后修改时间	
create_user	bigint	创建人 id	
update_user	bigint	最后修改人 id	



新增员工

- 需求分析和设计
- 代码开发
- 功能测试
- 代码完善



代码开发

根据新增员工接口设计对应的 DTO：

Body

名称	类型	是否必须	默认值	备注	其他信息
id	integer	非必须		员工id	format: int64
idNumber	string	必须		身份证	
name	string	必须		姓名	
phone	string	必须		手机号	
sex	string	必须		性别	
username	string	必须		用户名	



```
private Long id;  
  
private String username;  
  
private String name;  
  
private String password;  
  
private String phone;  
  
private String sex;  
  
private String idNumber;  
  
private Integer status;  
  
private LocalDateTime createTime;  
  
private LocalDateTime updateTime;  
  
private Long createUser;  
  
private Long updateUser;
```

implements Serializable

注意：当前端提交的数据和实体类中对应的属性差别比较大时，建议使用 DTO 来封装数据



代码开发

在 EmployeeController 中创建新增员工方法，接收前端提交的参数：

```
/*
 * 新增员工
 * @param employeeDTO
 * @return
 */
@PostMapping
@ApiOperation("新增员工接口")
public Result save(@RequestBody EmployeeDTO employeeDTO){
    log.info("新增员工：{}",employeeDTO);
    return Result.success();
}
```

EmployeeController



代码开发

在 EmployeeService 接口中声明新增员工方法：

```
/**  
 * 新增员工  
 * @param employeeDTO  
 */  
void save(EmployeeDTO employeeDTO);
```

EmployeeService



代码开发

在 EmployeeServiceImpl 中实现新增员工方法：

```
public void save(EmployeeDTO employeeDTO) {  
    Employee employee = new Employee();  
    // 属性拷贝  
    BeanUtils.copyProperties(employeeDTO, employee);  
  
    // 账号状态默认为 1，正常状态  
    employee.setStatus(StatusConstant.ENABLE);  
    // 默认密码为 123456  
    employee.setPassword(DigestUtils.md5DigestAsHex(PasswordConstant.DEFAULT_PASSWORD.getBytes()));  
    // 创建人、创建时间、修改人、修改时间  
    employee.setCreateTime(LocalDateTime.now());  
    employee.setUpdateTime(LocalDateTime.now());  
    employee.setCreateUser(10L);  
    employee.setUpdateUser(10L);  
  
    employeeMapper.insert(employee);  
}
```

EmployeeServiceImpl



代码开发

在 EmployeeMapper 中声明 insert 方法：

```
/*
 * 插入数据
 * @param employee
 */
@Insert("insert into employee" +
        "(name, username, password, phone, sex, id_number, status, create_time, update_time, create_user, update_user)" +
        "VALUES " +
        "(#{name}, #{username}, #{password}, #{phone}, #{sex}, #{idNumber}, #{status},
        #{createTime},#{updateTime},#{createUser}, #{updateUser})")
void insert(Employee employee);
```

EmployeeMapper



新增员工

- 需求分析和设计
- 代码开发
- 功能测试
- 代码完善

功能测试

功能测试方式：

- 通过接口文档测试
- 通过前后端联调测试

注意：由于开发阶段前端和后端是并行开发的，后端完成某个功能后，此时前端对应的功能可能还没有开发完成，导致无法进行前后端联调测试。所以在开发阶段，后端测试主要以接口文档测试为主。



功能测试

通过接口文档进行功能测试：

The screenshot shows the Knife4j API documentation interface. The top navigation bar includes a search bar and a language switcher (Chinese). The main content area displays a POST request to the endpoint `/admin/employee`. The request parameters tab is selected, showing the JSON body content:

```
1 | {  
2 |     "idNumber": "111222334455666",  
3 |     "name": "xiaozi",  
4 |     "phone": "13812344321",  
5 |     "sex": "1",  
6 |     "username": "小智"  
7 | }
```

The response content tab shows a single line of text: "1". Below the response, there is a status message: "显示说明 响应码: 401 耗时: 23ms 大小: 0 B".



功能测试

```
public boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object handler) throws Exception {  
    log.info("jwt 校验 ...");  
  
    // 判断当前拦截到的是 Controller 的方法还是其他资源  
    if(!(handler instanceof HandlerMethod)){  
        // 当前拦截到的不是动态方法，直接放行  
        return true;  
    }  
  
    //1、从请求头中获取令牌  
    String token = request.getHeader(jwtProperties.getAdminTokenName());  
  
    //2、校验令牌  
    try{  
        Claims claims = JwtUtil.parseJWT(jwtProperties.getAdminSecretKey(), token);  
        Long empId = Long.valueOf(claims.get(JwtClaimsConstant.EMP_ID).toString());  
        //3、通过，放行  
        return true;  
    }catch (Exception ex){  
        //4、不通过，响应 401 状态码  
        response.setStatus(401);  
        return false;  
    }  
}
```

JwtTokenAdminInterceptor

由于 JWT 令牌校验失败，导致 EmployeeController 的 save 方法没有被调



功能测试

调用员工登录接口获得一个合法的 JWT 令牌：

The screenshot shows the Knife4j API documentation interface. The top navigation bar includes links for '主页' (Home), '新增员工 X', '离线文档(商家管理端接口) X', '全局参数设置(商家管理端接口) X', and '员工登录 X'. The main content area is titled '苍穹外卖项目接口文档'. A 'POST' request is selected for the endpoint '/admin/employee/login'. The '请求参数' tab is active, showing a raw JSON payload:

```
1 {  
2   "password": "123456",  
3   "username": "admin"  
4 }
```

The '响应内容' tab displays the JSON response body:

```
1 {  
2   "code": 1,  
3   "msg": null,  
4   "data": {  
5     "id": 1,  
6     "userName": "admin",  
7     "name": "管理员",  
8     "token": "eyJhbGciOiJIUzI1NiJ9.eyJlbXBjZCI6MSwiZXhwIjoxNjY0MTUwfQ.zQvw_XfcdBEipJRcSuo-88ixWC1h3bJTERNqBT_1urg"  
9   }  
10 }
```

At the bottom, it says 'Apache License 2.0 | Copyright © 2019-Knife4j'.



功能测试

将合法的 JWT 令牌添加到全局参数中：

苍穹外卖项目接口文档

输入文档关键字搜索

主页 全局参数设置(商家管理端接口) X

Knife4j 提供全局参数Debug功能,目前默认提供header(请求头)、query(form)两种方式的入参.

在此添加全局参数后,默认Debug调试tab页会带上该参数

+ 添加参数

参数名称	参数值	参数类型	操作
token	eyJhbGciOiJIUzI1NiJ9.eyJlbXBjZCI6MSwiZXhwIjoxNjY0MTY4MzkyfQ.H-tG1p3iDYFbfDJAo3ujKRV12Pwhd3E9rmkLU5CDkVM	header	删除

Apache License 2.0 | Copyright © 2019-Knife4j



功能测试

苍穹外卖项目接口文档

主页 全局参数设置(商家管理端接口) X 新增员工 X

文档 调试 Open

POST /admin/employee

① 请求头部 请求参数 AfterScript

x-www-form-urlencoded form-data raw JSON

```
1 {  
2   "idNumber": "111222333444555666",  
3   "name": "xiaoZhi",  
4   "phone": "13812344321",  
5   "sex": "1",  
6   "username": "小智"  
7 }
```

响应内容 Raw Headers Curl

```
1 {  
2   "code": 1,  
3   "msg": null,  
4   "data": null  
5 }
```

Apache

苍穹外卖项目接口文档

输入文档关键字搜索

主页 全局参数设置(商家管理端接口) X 新增员工 X

文档 调试 Open

POST /admin/employee

① 请求头部 请求参数 AfterScript

请求头 内容

token eyJhbGciOiJIUzI1NiJ9eyJlbXBJZCI6MSwiZXhwIjoxNjY0MTY4MzkyfQ.H-tG1p3iDYFbfDAo3ujKRV12PwHd3E

删除

响应内容 Raw Headers Curl 显示说明 响应码: 200 耗时: 87ms 大小: 33 B

```
1 {  
2   "code": 1,  
3   "msg": null,  
4   "data": null  
5 }
```

Apache License 2.0 | Copyright © 2019-Knife4J



新增员工

- 需求分析和设计
- 代码开发
- 功能测试
- 代码完善



代码完善

程序存在的问题：

The screenshot shows the MySQL Workbench interface with the following details:

- Table Name:** employee
- Schema:** sky_take_out
- Columns:**
 - id:** BIGINT, PK, NN, AI
 - name:** VARCHAR(32)
 - username:** VARCHAR(32) (highlighted with a red box)
 - password:** VARCHAR(64)
 - phone:** VARCHAR(11)
 - sex:** VARCHAR(2)
 - id_number:** VARCHAR(18)
 - status:** INT
 - create_time:** DATETIME
 - update_time:** DATETIME
 - create_user:** BIGINT
 - update_user:** BIGINT
- Column 'username' Properties:**
 - Column Name:** username
 - Charset/Collation:** utf8 / utf8_bin
 - Comments:** 用户名
 - Data Type:** VARCHAR(32)
 - Default:**
 - Storage:** Stored
 - Options:**
 - Primary Key
 - Not Null
 - Unique (highlighted with a red box)
 - Binary
 - Unsigned
 - Zero Fill
 - Auto Increment
 - Generated

A yellow callout box labeled "EmployeeServiceImpl" points to the "Unique" checkbox in the column properties.

A red box highlights the "username" column in the table schema grid.

A warning message on the right side of the interface reads:

```
insert VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?);  
Duplicate entry 'zhangsan' for key 'employee.username'
```

At the bottom of the interface, there are tabs for **Columns**, **Indexes**, **Foreign Keys**, **Triggers**, **Partitioning**, and **Options**.



代码完善

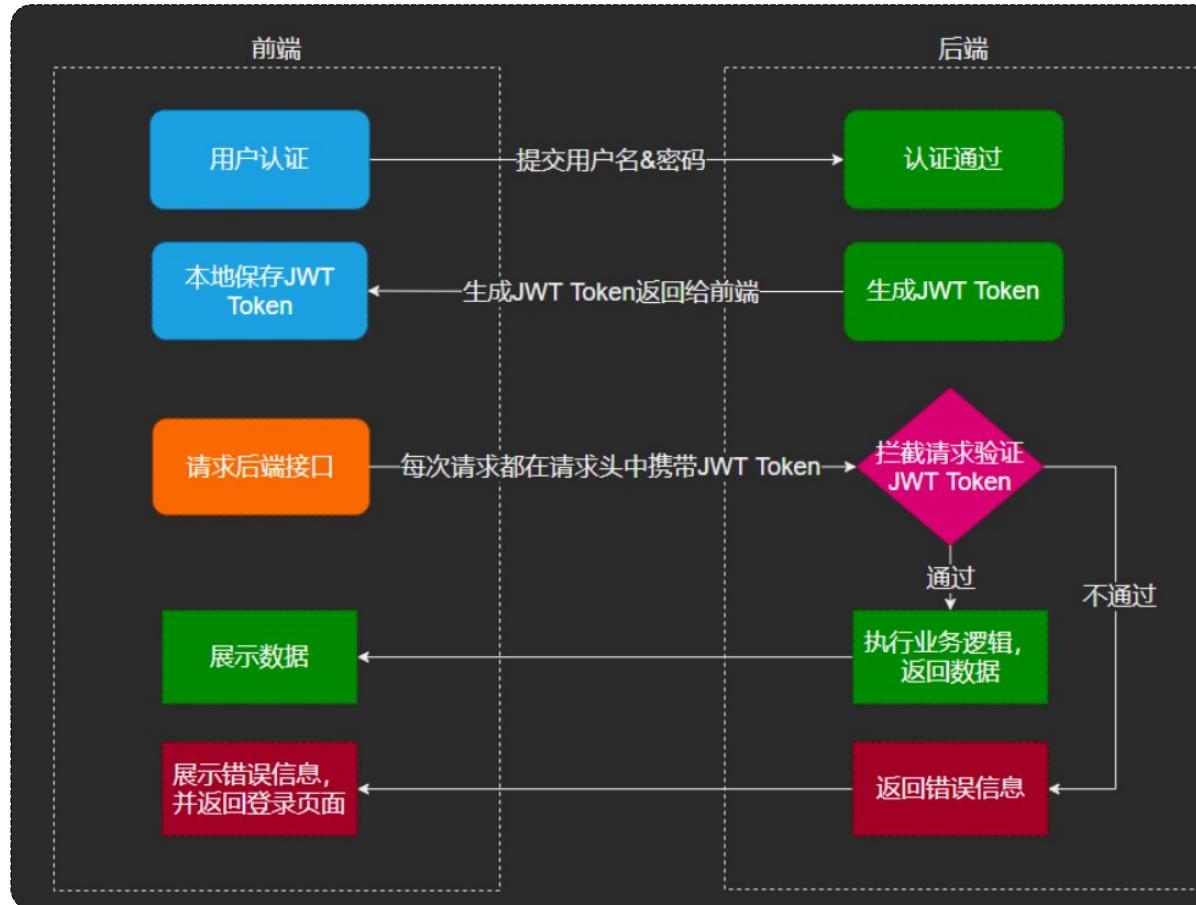
针对第一个问题，可以通过全局异常处理器来处理：

```
/*
 * 捕获sql 异常
 * @param ex
 * @return
 */
@ExceptionHandler
public Result exceptionHandler(SQLIntegrityConstraintViolationException ex){
    log.error(" 异常信息： {}", ex.getMessage());
    String message = ex.getMessage();
    if(message.contains("Duplicate entry")){
        String[] split = message.split(" ");
        String name = split[2];
        return Result.error(name + MessageConstant.ALREADY_EXISTS);
    }
    return Result.error(MessageConstant.UNKNOWN_ERROR);
}
```

GlobalExceptionHandler

代码完善

针对第二个问题，需要通过某种方式动态获取当前登录员工的 id：





代码完善

员工登录成功后会生成 JWT 令牌并响应给前端：

```
// 登录成功后，生成 jwt 令牌
Map<String, Object> claims = new HashMap<>();
claims.put(JwtClaimsConstant.EMP_ID,employee.getId());
String token = JwtUtil.createJWT(
        jwtProperties.getAdminSecretKey(),
        jwtProperties.getAdminTtl(),
        claims);
```

EmployeeController



代码完善

后续请求中，前端会携带 JWT 令牌，通过 JWT 令牌可以解析出当前登录员工 id：

```
//1、从请求头中获取令牌
String token = request.getHeader(jwtProperties.getAdminTokenName());

//2、校验令牌
try{
    Claims claims = JwtUtil.parseJWT(jwtProperties.getAdminSecretKey(), token);
    Long empId = Long.valueOf(claims.get(JwtClaimsConstant.EMP_ID).toString());
    //3、通过，放行
    return true;
} catch (Exception ex){
    //4、不通过，响应401状态码
    response.setStatus(401);
    return false;
}
```

JwtTokenAdminInterceptor

解析出登录员工 id 后，如何传递给 Service 的 save 方法？

代码完善

ThreadLocal 并不是一个 Thread , 而是 Thread 的局部变量。

ThreadLocal 为每个线程提供单独一份存储空间 , 具有线程隔离的效果 , 只有在线程内才能获取到对应的值 , 线程外则不能访问。

ThreadLocal 常用方法 :

- public void set(T value) 设置当前线程的线程局部变量的值
- public T get() 返回当前线程所对应的线程局部变量的值
- public void remove() 移除当前线程的线程局部变量

注意 : 客户端发送的每次请求 , 后端的 Tomcat 服务器都会分配一个单独的线程来处理请求



代码完善

初始工程中已经封装了 ThreadLocal 操作的工具类：

```
public class BaseContext {  
  
    public static ThreadLocal<Long> threadLocal = new ThreadLocal<>();  
  
    public static void setCurrentId(Long id) {  
        threadLocal.set(id);  
    }  
  
    public static Long getCurrentId() {  
        return threadLocal.get();  
    }  
  
    public static void removeCurrentId() {  
        threadLocal.remove();  
    }  
}
```

BaseContext



代码完善

在拦截器中解析出当前登录员工 id，并放入线程局部变量中：

```
//1、从请求头中获取令牌
String token = request.getHeader(jwtProperties.getAdminTokenName());

//2、校验令牌
try{
    Claims claims = JwtUtil.parseJWT(jwtProperties.getAdminSecretKey(), token);
    Long empId = Long.valueOf(claims.get(JwtClaimsConstant.EMP_ID).toString());
    BaseContext.setCurrentId(empId);
    //3、通过，放行
    return true;
} catch (Exception ex){
    //4、不通过，响应401状态码
    response.setStatus(401);
    return false;
}
```

JwtTokenAdminInterceptor



代码完善

在 Service 中获取线程局部变量中的值：

```
public void save(EmployeeDTO employeeDTO) {  
    Employee employee = new Employee();  
    // 属性拷贝  
    BeanUtils.copyProperties(employeeDTO, employee);  
  
    // 账号状态默认为 1，正常状态  
    employee.setStatus(StatusConstant.ENABLE);  
    // 默认密码为 123456  
    employee.setPassword(DigestUtils.md5DigestAsHex("123456".getBytes()));  
    // 创建人、创建时间、修改人、修改时间  
    employee.setCreateTime(LocalDateTime.now());  
    employee.setUpdateTime(LocalDateTime.now());  
    employee.setCreateUser(BaseContext.getCurrentId());  
    employee.setUpdateUser(BaseContext.getCurrentId());  
  
    employeeMapper.insert(employee);  
}
```

EmployeeServiceImpl



目录

Contents

- ◆ 新增员工
- ◆ 员工分页查询
- ◆ 启用禁用员工账号
- ◆ 编辑员工
- ◆ 导入分类模块功能代码



员工分页查询

- 需求分析和设计
- 代码开发
- 功能测试
- 代码完善



需求分析和设计

产品原型：

员工管理

员工姓名 搜索

添加员工

员工姓名	账号	手机号	账号状态	最后操作时间	操作
管理员	13333333333	13333333333	禁用	2022-4-29 09:39	修改 启用 删除
管理员	13333333333	13333333333	启用	2022-4-29 09:39	修改 禁用 删除
管理员	13333333333	13333333333	启用	2022-4-29 09:39	修改 禁用 删除
管理员	13333333333	13333333333	启用	2022-4-29 09:39	修改 禁用 删除
管理员	13333333333	13333333333	启用	2022-4-29 09:39	修改 禁用 删除
管理员	13333333333	13333333333	启用	2022-4-29 09:39	修改 禁用 删除
管理员	13333333333	13333333333	启用	2022-4-29 09:39	修改 禁用 删除
管理员	13333333333	13333333333	启用	2022-4-29 09:39	修改 禁用 删除
管理员	13333333333	13333333333	启用	2022-4-29 09:39	修改 禁用 删除
管理员	13333333333	13333333333	启用	2022-4-29 09:39	修改 禁用 删除

总共 85 个项目 < 1 2 3 4 5 >

业务规则：

- 根据页码展示员工信息
- 每页展示 10 条数据
- 分页查询时可以根据需要，输入员工姓名进行查询



需求分析和设计

接口设计：

基本信息

Path: /admin/employee/page

Method: GET

接口描述:

请求参数

Query

参数名称	是否必须	示例	备注
name	否	张三	员工姓名
page	是	1	页码
pageSize	是	10	每页记录数

返回数据

名称	类型	是否必须	默认值	备注	其他信息
code	number	必须			
msg	null	非必须			
data	object	必须			
└ total	number	必须			
└ records	object []	必须			item 类型: object
└ id	number	必须			
└ username	string	必须			
└ name	string	必须			
└ password	string	必须			
└ phone	string	必须			
└ sex	string	必须			
└ idNumber	string	必须			
└ status	number	必须			
└ createTime	string,null	必须			
└ updateTime	string	必须			
└ createUser	number,null	必须			
└ updateUser	number	必须			



员工分页查询

- 需求分析和设计
- 代码开发
- 功能测试
- 代码完善



代码开发

根据分页查询接口设计对应的 DTO：

请求参数

Query

参数名称	是否必须	示例	备注
name	否	张三	员工姓名
page	是	1	页码
pageSize	是	10	每页记录数



```
@Data
public class EmployeePageQueryDTO implements Serializable {

    // 员工姓名
    private String name;

    // 页码
    private int page;

    // 每页显示记录数
    private int pageSize;
}
```



代码开发

后面所有的分页查询，统一都封装成 PageResult 对象：

```
/*
 * 封装分页查询结果
 */
@Data
@AllArgsConstructor
@NoArgsConstructor
public class PageResult implements Serializable {

    private long total; // 总记录数

    private List records; // 当前页数据集合

}
```



代码开发

员工信息分页查询后端返回的对象类型为：Result<PageResult>

名称	类型	是否必须	默认值	备注	其他信息
code	number	非必须			
msg	null	非必须			
data	object	必须			
└ total	number	必须			
└ records	object []	必须			item 类型: object
└ id	number	必须			
└ username	string	必须			
└ name	string	必须			
└ password	string	必须			
└ phone	string	必须			
└ sex	string	必须			



代码开发

根据接口定义创建分页查询方法：

```
/*
 * 员工分页查询
 * @param employeePageQueryDTO
 * @return
 */
@GetMapping("/page")
@ApiOperation("员工分页查询")
public Result<PageResult> page(EmployeePageQueryDTO employeePageQueryDTO){
    Log.info("分页查询：{}",employeePageQueryDTO);
    PageResult pageResult = employeeService.pageQuery(employeePageQueryDTO);
    return Result.success(pageResult);
}
```

EmployeeController

代码开发

在 EmployeeService 接口中声明 pageQuery 方法：

```
/**  
 * 分页查询  
 * @param employeePageQueryDTO  
 * @return  
 */  
PageResult pageQuery(EmployeePageQueryDTO employeePageQueryDTO);
```

EmployeeService



代码开发

在 EmployeeServiceImpl 中实现 pageQuery 方法：

```
public PageResult pageQuery(EmployeePageQueryDTO employeePageQueryDTO) {
    //select * from employee limit 10,20
    // 基于PageHelper 插件实现动态分页查询
    PageHelper.startPage(employeePageQueryDTO.getPage(), employeePageQueryDTO.getPageSize());
    Page<Employee> page = employeeMapper.pageQuery(employeePageQueryDTO);
    return new PageResult(page.getTotal(), page.getResult());
}
```

EmployeeServiceImpl

注意：此处使用 mybatis 的分页插件 PageHelper 来简化分页代码的开发。底层基于 mybatis 的拦截器实现。

```
<dependency>
    <groupId>com.github.pagehelper</groupId>
    <artifactId>pagehelper-spring-boot-starter</artifactId>
    <version>${pagehelper}</version>
</dependency>
```

pom.xml



代码开发

在 EmployeeMapper 中声明 pageQuery 方法：

```
/**  
 * 分页查询  
 * @param employeePageQueryDTO  
 * @return  
 */  
Page<Employee> pageQuery(EmployeePageQueryDTO employeePageQueryDTO);
```



代码开发

在 EmployeeMapper.xml 中编写 SQL：

```
<select id="pageQuery" resultType="com.sky.entity.Employee">
    select * from employee
    <where>
        <if test="name != null and name != ''">
            and name like concat('%',#{name},'%')
        </if>
    </where>
    order by create_time desc
</select>
```



员工分页查询

- 需求分析和设计
- 代码开发
- 功能测试
- 代码完善



功能测试

可以通过接口文档进行测试，也可以进行前后端联调测试，最后操作时间字段展示有问题，如下：



账号状态	最后操作时间
● 启用	20215102249
● 启用	2022524112024
● 启用	2022524112934
● 启用	2022524114543
● 启用	202252412945



员工分页查询

- 需求分析和设计
- 代码开发
- 功能测试
- 代码完善



代码完善

解决方式：

- 方式一：在属性上加入注解，对日期进行格式化 `@JsonFormat(pattern = "yyyy-MM-dd HH:mm:ss") private LocalDateTime updateTime;`
- 方式二：在 `WebMvcConfiguration` 中扩展 Spring MVC 的消息转换器，统一对日期类型进行格式化处理

```
/**  
 * 扩展 mvc 框架的消息转换器  
 * @param converters  
 */  
protected void extendMessageConverters(List<HttpMessageConverter<?>> converters) {  
    Log.info("开始扩展消息转换器 ...");  
  
    // 创建一个消息转换器对象  
    MappingJackson2HttpMessageConverter converter = new MappingJackson2HttpMessageConverter();  
    // 设置对象转换器，可以将 Java 对象转为 json 字符串  
    converter.setObjectMapper(new JacksonObjectMapper());  
  
    // 将我们自己的转换器放入 spring MVC 框架的容器中  
    converters.add(0, converter);  
}
```

WebMvcConfiguration



代码完善

查看效果：

响应内容 Raw Headers Curl

```
1 {  
2   "code": 1,  
3   "msg": null,  
4   "data": {  
5     "total": 3,  
6     "records": [  
7       {  
8         "id": 20,  
9         "username": "admin123",  
10        "name": "itcast",  
11        "password": "e10adc3949ba59abbe56e057f20f883e",  
12        "phone": "131",  
13        "sex": "1",  
14        "idNumber": "123",  
15        "status": 1,  
16        "createTime": "2022-05-24 11:20",  
17        "updateTime": "2022-05-24 11:20",  
18        "createUser": 1,  
19        "updateUser": 1  
20      },  
21      {  
22        "id": 22,  
23        "username": "admin1234",  
24        "name": "itcast",  
25        "password": "e10adc3949ba59abbe56e057f20f883e",  
26        "phone": "131",  
27        "sex": "1",  
28        "idNumber": "123",  
29        "status": 1,  
30        "createTime": "2022-05-24 11:45",  
31        "updateTime": "2022-05-24 11:45",  
32        "createUser": 1,  
33        "updateUser": 1  
34      }]
```



账号状态	最后操作时间
• 启用	2021-05-10 02:24
• 启用	2022-05-24 11:20
• 启用	2022-05-24 11:29
• 启用	2022-05-24 11:45
• 启用	2022-05-24 12:09



目录

Contents

- ◆ 新增员工
- ◆ 员工分页查询
- ◆ 启用禁用员工账号
- ◆ 编辑员工
- ◆ 导入分类模块功能代码



启用禁用员工账号

- 需求分析和设计
- 代码开发
- 功能测试



需求分析和设计

产品原型：

员工姓名

搜索

添加员工

员工姓名	账号	手机号	账号状态	最后操作时间	操作
管理员	13333333333	13333333333	禁用	2022-4-29 09:39	修改 启用 删除
管理员	13333333333	13333333333	启用	2022-4-29 09:39	修改 禁用 删除
管理员	13333333333	13333333333	启用	2022-4-29 09:39	修改 禁用 删除
管理员	13333333333	13333333333	启用	2022-4-29 09:39	修改 禁用 删除
管理员	13333333333	13333333333	启用	2022-4-29 09:39	修改 禁用 删除
管理员	13333333333	13333333333	启用	2022-4-29 09:39	修改 禁用 删除
管理员	13333333333	13333333333	启用	2022-4-29 09:39	修改 禁用 删除
管理员	13333333333	13333333333	启用	2022-4-29 09:39	修改 禁用 删除
管理员	13333333333	13333333333	启用	2022-4-29 09:39	修改 禁用 删除
管理员	13333333333	13333333333	启用	2022-4-29 09:39	修改 禁用 删除

高级软件人才培训专家

需求分析和设计

业务规则：

- 可以对状态为“启用”的员工账号进行“禁用”操作
- 可以对状态为“禁用”的员工账号进行“启用”操作
- 状态为“禁用”的员工账号不能登录系统



需求分析和设计

接口设计：

基本信息

Path: /admin/employee/status/{status}

Method: POST

接口描述:

请求参数

Headers

参数名称	参数值	是否必须	示例	备注
Content-Type	application/json	是		

路径参数

参数名称	示例	备注
status	1	状态, 1为启用 0为禁用

Query

参数名称	是否必须	示例	备注
id	是		员工id

返回数据

名称	类型	是否必须	默认值	备注	其他信息
code	number	必须			
msg	string	非必须			
data	string	非必须			



启用禁用员工账号

- 需求分析和设计
- 代码开发
- 功能测试



代码开发

根据接口设计中的请求参数形式对应的在 EmployeeController 中创建启用禁用员工账号的方法：

```
@PostMapping("/status/{status}")
@ApiOperation("启用禁用员工账号")
public Result<String> startOrStop(@PathVariable Integer status, Long id){
    log.info("启用禁用员工账号：{},{}",status,id);
    employeeService.startOrStop(status,id);
    return Result.success();
}
```

EmployeeController



代码开发

在 EmployeeService 接口中声明启用禁用员工账号的业务方法：

```
/**  
 * 启用禁用员工账号  
 * @param status  
 * @param id  
 */  
void startOrStop(Integer status, Long id);
```

EmployeeService



代码开发

在 EmployeeServiceImpl 中实现启用禁用员工账号的业务方法：

```
/**  
 * 启用禁用员工账号  
 *  
 * @param status  
 * @param id  
 */  
public void startOrStop(Integer status, Long id) {  
    Employee employee = Employee.builder()  
        .id(id)  
        .status(status)  
        .build();  
    employeeMapper.update(employee);  
}
```

EmployeeServiceImpl



代码开发

在 EmployeeMapper 接口中声明 update 方法：

```
/**  
 * 根据id 修改员工信息  
 * @param employee  
 */  
void update(Employee employee);
```

EmployeeMapper



代码开发

在 EmployeeMapper.xml 中编写 SQL：

```
<update id="update">
    update employee
    <set>
        <if test="username != null">username = #{username},</if>
        <if test="name != null">name = #{name},</if>
        <if test="password != null">password = #{password},</if>
        <if test="phone != null">phone = #{phone},</if>
        <if test="sex != null">sex = #{sex},</if>
        <if test="idNumber != null">id_Number = #{idNumber},</if>
        <if test="updateTime != null">update_Time = #{updateTime},</if>
        <if test="updateUser != null">update_User = #{updateUser},</if>
        <if test="status != null">status = #{status},</if>
    </set>
    where id = #{id}
</update>
```

EmployeeMapper.xml



启用禁用员工账号

- 需求分析和设计
- 代码开发
- 功能测试



功能测试

可以通过接口文档进行测试，最后完成前后端联调测试即可

三 苍穹外卖项目接口文档

输入文档关键字搜索

主页 启用禁用员工账号 X

文档 调试 Open

POST /admin/employee/status/{status}

请求头部 请求参数 AfterScript

x-www-form-urlencoded form-data raw

参数名称	参数值	操作
id	3	删除
status	0	删除

响应内容 Raw Headers Curl 显示说明 响应码: 200 耗时: 40ms 大小: 33 B

```
1 {"f
2   "code": 1,
3   "msg": null,
4   "data": null
5 }
```

Apache License 2.0 | Copyright © 2019-Knife4j



目录

Contents

- ◆ 新增员工
- ◆ 员工分页查询
- ◆ 启用禁用员工账号
- ◆ 编辑员工
- ◆ 导入分类模块功能代码



编辑员工

- 需求分析和设计
- 代码开发
- 功能测试



需求分析和设计

产品原型：

员工管理

1 员工姓名 搜索

2 添加员工

员工姓名	账号	手机号	账号状态	最后操作时间	3 操作
管理员	13333333333	13333333333	禁用	2022-4-29 09:39	修改 启用 删除
管理员	13333333333	13333333333	启用	2022-4-29 09:39	修改 禁用 删除
管理员	13333333333	13333333333	启用	2022-4-29 09:39	修改 禁用 删除
管理员	13333333333	13333333333	启用	2022-4-29 09:39	修改 禁用 删除
管理员	13333333333	13333333333	启用	2022-4-29 09:39	修改 禁用 删除
管理员	13333333333	13333333333	启用	2022-4-29 09:39	修改 禁用 删除
管理员	13333333333	13333333333	启用	2022-4-29 09:39	修改 禁用 删除
管理员	13333333333	13333333333	启用	2022-4-29 09:39	修改 禁用 删除
管理员	13333333333	13333333333	启用	2022-4-29 09:39	修改 禁用 删除
管理员	13333333333	13333333333	启用	2022-4-29 09:39	修改 禁用 删除



需求分析和设计

编辑员工功能涉及到两个接口：

- 根据 id 查询员工信息
 - 编辑员工信息

基本信息

Path: /admin/employee

Method: PUT

请求参数

Headers

参数名称	参数值	是否必须	示例	备注
Content-Type	application/json	是		

Path: /adm

Method: GB

接口描述

名称	类型	是否必须	默认值	备注	其他信息
id	integer	必须			format: int64
idNumber	string	必须			
name	string	必须			
phone	string	必须			
sex	string	必须			
username	string	必须			

是否必须	默认值	备注	其他信息
必须			format: int32
必须			
必须			format: date-time
必须			format: int64
必须			format: int64
必须			

返回数据

名称	类型	是否必须	默认值	备注	其他信息
code	integer	必须			format: int32
data	object	非必须			
msg	string	非必须			



编辑员工

- 需求分析和设计
- 代码开发
- 功能测试



代码开发

在 EmployeeController 中创建 getById 方法：

```
/**  
 * 根据 id 查询员工  
 * @param id  
 * @return  
 */  
@GetMapping("/{id}")  
@ApiOperation("根据 id 查询员工")  
public Result<Employee> getById(@PathVariable Long id){  
    return Result.success(employeeService.getById(id));  
}
```

EmployeeController

代码开发

在 EmployeeService 接口中声明 getById 方法：

```
/**  
 * 根据 id 查询员工  
 * @param id  
 * @return  
 */  
Employee getById(Long id);
```

EmployeeService

代码开发

在 EmployeeServiceImpl 中实现 getById 方法：

```
/**  
 * 根据 id 查询员工  
 * @param id  
 * @return  
 */  
public Employee getById(Long id) {  
    Employee employee = employeeMapper.getById(id);  
    employee.setPassword("****");  
    return employee;  
}
```

EmployeeServiceImpl



代码开发

在 EmployeeMapper 接口中声明 getById 方法：

```
/**  
 * 根据 id 查询员工  
 * @param id  
 * @return  
 */  
@Select("select * from employee where id = #{id}")  
Employee getById(Long id);
```

EmployeeMapper

可以先通过接口测试确认数据回显是否有问题，如果没有问题再继续开发

代码开发

在 EmployeeController 中创建 update 方法：

```
/**  
 * 编辑员工信息  
 * @param employeeDTO  
 * @return  
 */  
  
@PutMapping  
@ApiOperation("编辑员工信息")  
public Result<String> update(@RequestBody EmployeeDTO employeeDTO){  
    log.info("编辑员工：{}", employeeDTO);  
    employeeService.update(employeeDTO);  
    return Result.success();  
}
```

EmployeeController



代码开发

在 EmployeeService 接口中声明 update 方法：

```
/**  
 * 根据 id 修改员工信息  
 * @param employeeDTO  
 */  
void update(EmployeeDTO employeeDTO);
```

EmployeeService



代码开发

在 EmployeeServiceImpl 中实现 update 方法：

```
/**  
 * 修改员工  
 * @param employeeDTO  
 */  
public void update(EmployeeDTO employeeDTO) {  
    // update employee set ... where id = ?  
    Employee employee = new Employee();  
    BeanUtils.copyProperties(employeeDTO, employee);  
  
    // 设置修改人和修改时间  
    employee.setUpdateUser(BaseContext.getCurrentId());  
    employee.setUpdateTime(LocalDateTime.now());  
  
    employeeMapper.update(employee);  
}
```

EmployeeServiceImpl



编辑员工

- 需求分析和设计
- 代码开发
- 功能测试



功能测试

通过 Swagger 接口文档进行测试，通过后再前端联调测试即可



目录

Contents

- ◆ 新增员工
- ◆ 员工分页查询
- ◆ 启用禁用员工账号
- ◆ 编辑员工
- ◆ 导入分类模块功能代码



导入分类模块功能代码

- 需求分析和设计
- 代码导入
- 功能测试



需求分析和设计

产品原型：

分类管理

1 分类名称 分类类型 搜索

1	分类名称	分类类型	排序	状态	操作时间	3 操作
	荤菜	菜品分类	1	启用	2019-01-01 11:11	<input type="button" value="修改"/> <input type="button" value="删除"/> <input type="button" value="禁用"/>
	素菜	菜品分类	2	禁用	2019-01-01 11:11	<input type="button" value="修改"/> <input type="button" value="删除"/> <input type="button" value="启用"/>
	凉菜	菜品分类	3	启用	2019-01-01 11:11	<input type="button" value="修改"/> <input type="button" value="删除"/> <input type="button" value="禁用"/>
	周一套餐	套餐分类	4	禁用	2019-01-02 11:11	<input type="button" value="修改"/> <input type="button" value="删除"/> <input type="button" value="启用"/>

需求分析和设计

业务规则：

- 分类名称必须是唯一的
- 分类按照类型可以分为**菜品分类**和**套餐分类**
- 新添加的分类状态默认为“禁用”

需求分析和设计

接口设计：

- 新增分类
- 分类分页查询
- 根据 id 删除分类
- 修改分类
- 启用禁用分类
- 根据类型查询分类

需求分析和设计

数据库设计 (category 表) :

字段名	数据类型	说明	备注
id	bigint	主键	自增
name	varchar(32)	分类名称	唯一
type	int	分类类型	1 菜品分类 2 套餐分类
sort	int	排序字段	用于分类数据的排序
status	int	状态	1 启用 0 禁用
create_time	datetime	创建时间	
update_time	datetime	最后修改时间	
create_user	bigint	创建人 id	
update_user	bigint	最后修改人 id	



导入分类模块功能代码

- 需求分析和设计
- 代码导入
- 功能测试

代码导入

导入资料中的分类管理模块功能代码即可

- CategoryController.java
- CategoryMapper.java
- CategoryMapper.xml
- CategoryService.java
- CategoryServiceImpl.java
- DishMapper.java
- SetmealMapper.java



导入分类模块功能代码

- 需求分析和设计
- 代码导入
- 功能测试



功能测试

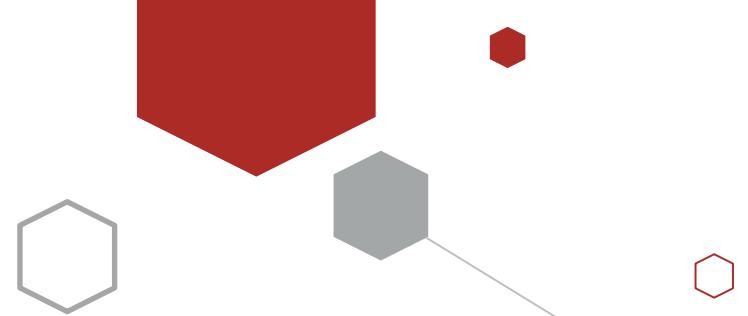
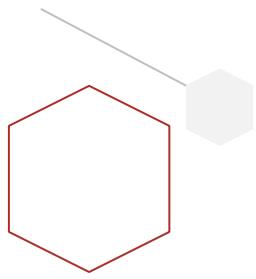
直接进行前后端联调测试即可

The screenshot shows a web-based management system for a restaurant. The top navigation bar includes the logo '苍穹外卖 THE SKY TAKE-OUT' and status indicators like '营业中' (Open). On the left, a sidebar menu lists various modules: 工作台, 数据统计, 订单管理, 套餐管理, 菜品管理, 分类管理 (highlighted with a red box), and 员工管理. The main content area displays a table of food categories. The columns are: 分类名称 (Category Name), 分类类型 (Category Type), 排序 (Sort Order), 状态 (Status), 操作时间 (Operation Time), and 操作 (Operations). The data in the table is as follows:

分类名称	分类类型	排序	状态	操作时间	操作
蜀味牛蛙	菜品分类	4	启用	2022-10-07 11:07	修改 删除 禁用
蜀味烤鱼	菜品分类	5	启用	2022-10-07 11:07	修改 删除 禁用
特色蒸菜	菜品分类	6	启用	2022-06-09 22:17	修改 删除 禁用
新鲜时蔬	菜品分类	7	启用	2022-06-09 22:18	修改 删除 禁用
水煮鱼	菜品分类	8	启用	2022-06-09 22:23	修改 删除 禁用
传统主食	菜品分类	9	启用	2022-06-09 22:18	修改 删除 禁用
酒水饮料	菜品分类	10	启用	2022-06-09 22:09	修改 删除 禁用
汤类	菜品分类	11	启用	2022-06-10 10:51	修改 删除 禁用
人气套餐	套餐分类	12	启用	2022-06-10 11:04	修改 删除 禁用
商务套餐	套餐分类	13	启用	2022-06-10 11:04	修改 删除 禁用



传智教育旗下高端IT教育品牌



菜品管理



黑马程序员 | 传智教育旗下
高端IT教育品牌
www.itheima.com



营业中

工作台

数据统计

订单管理

套餐管理

菜品管理

分类管理

员工管理

菜品名称:

菜品分类:

售卖状态:

查询

批量删除

+ 新建菜品

<input type="checkbox"/> 菜品名称	图片	菜品分类	售价	售卖状态	最后操作时间	操作
<input type="checkbox"/> 测试菜品		蜀味烤鱼	¥ 58	● 停售	2022-09-20 18:30	修改 删除 启售
<input type="checkbox"/> 平菇豆腐汤		汤类	¥ 6	● 启售	2022-06-10 10:55	修改 删除 停售
<input type="checkbox"/> 鸡蛋汤		汤类	¥ 4	● 启售	2022-06-10 10:54	修改 删除 停售
<input type="checkbox"/> 鲷鱼2斤		蜀味烤鱼	¥ 72	● 启售	2022-06-10 10:43	修改 删除 停售
<input type="checkbox"/> 江团鱼2斤		蜀味烤鱼	¥ 119	● 启售	2022-06-10 10:42	修改 删除 停售
<input type="checkbox"/> 草鱼2斤		蜀味烤鱼	¥ 68	● 启售	2022-06-10 10:41	修改 删除 停售
<input type="checkbox"/> 馋嘴牛蛙		蜀味牛蛙	¥ 88	● 启售	2022-06-10 10:37	修改 删除 停售
<input type="checkbox"/> 香锅牛蛙		蜀味牛蛙	¥ 88	● 启售	2022-06-10 10:35	修改 删除 停售



目录

Contents

- ◆ 公共字段自动填充
- ◆ 新增菜品
- ◆ 菜品分页查询
- ◆ 删除菜品
- ◆ 修改菜品



公共字段自动填充

- 问题分析
- 实现思路
- 代码开发
- 功能测试



问题分析

业务表中的公共字段：

序号	字段名	含义	数据类型
1	create_time	创建时间	datetime
2	create_user	创建人 id	bigint
3	update_time	修改时间	datetime
4	update_user	修改人 id	bigint

```
// 设置当前记录的创建时间、修改时间、创建人、修改人
employee.setCreateTime(LocalDateTime.now());
employee.setUpdateTime(LocalDateTime.now());
employee.setCreateUser(BaseContext.getCurrentId());
employee.setUpdateUser(BaseContext.getCurrentId());
```

```
// 设置创建时间、修改时间、创建人、修改人
category.setCreateTime(LocalDateTime.now());
category.setUpdateTime(LocalDateTime.now());
category.setCreateUser(BaseContext.getCurrentId());
category.setUpdateUser(BaseContext.getCurrentId());
```

问题：代码冗余、不便于后期维护



公共字段自动填充

- 问题分析
- 实现思路
- 代码开发
- 功能测试

实现思路

序号	字段名	含义	数据类型	操作类型
1	create_time	创建时间	datetime	insert
2	create_user	创建人 id	bigint	
3	update_time	修改时间	datetime	insert、 update
4	update_user	修改人 id	bigint	

- 自定义注解 AutoFill，用于标识需要进行公共字段自动填充的方法
- 自定义切面类 AutoFillAspect，统一拦截加入了 AutoFill 注解的方法，通过反射为公共字段赋值
- 在 Mapper 的方法上加入 AutoFill 注解

技术点：枚举、注解、AOP、反射



公共字段自动填充

- 问题分析
- 实现思路
- 代码开发
- 功能测试



代码开发

- 自定义注解 AutoFill

```
/**  
 * 自动填充  
 */  
@Target(ElementType.METHOD)  
@Retention(RetentionPolicy.RUNTIME)  
public @interface AutoFill {  
  
    /**  
     * 数据库操作类型  
     * @return  
     */  
    OperationType value();  
}
```



代码开发

- 自定义切面 AutoFillAspect

```
/**  
 * 自定义切面类，统一为公共字段赋值  
 */  
@Aspect  
@Component  
@Slf4j  
public class AutoFillAspect {  
    /**  
     * 切入点  
     */  
    @Pointcut("execution(* com.sky.mapper.*.*(..)) && @annotation(com.sky.annotation.AutoFill)")  
    public void autoFillPointCut() {}  
  
    /**  
     * 通知自动填充公共字段  
     * @param joinPoint  
     */  
    @Before("autoFillPointCut()")  
    public void autoFill(JoinPoint joinPoint) {  
        Log.info(" 公共字段自动填充 ...");  
    }  
}
```



代码开发

- 完善自定义切面 AutoFillAspect 的 autoFill 方法

```
log.info("公共字段自动填充 . . .");

// 获得方法签名对象
MethodSignature signature = (MethodSignature) joinPoint.getSignature();
// 获得方法上的注解
AutoFill autoFill = signature.getMethod().getAnnotation(AutoFill.class);
// 获得注解中的操作类型
OperationType operationType = autoFill.value();

// 获取当前目标方法的参数
Object[] args = joinPoint.getArgs();

if (args == null || args.length == 0) {
    return;
}

// 实体对象
Object entity = args[0];

// 准备赋值的数据
LocalDateTime time = LocalDateTime.now();
Long empId = BaseContext.getCurrentId();
```



代码开发

- 完善自定义切面 AutoFillAspect 的 autoFill 方法

```
if (operationType == OperationType.INSERT) {  
    // 当前执行的是insert 操作，为4个字段赋值  
    try {  
        // 获得set 方法对象 ----Method  
        Method setCreateTime = entity.getClass().getDeclaredMethod(AutoFillConstant.SET_CREATE_TIME, LocalDateTime.class);  
        Method setUpdateTime = entity.getClass().getDeclaredMethod(AutoFillConstant.SET_UPDATE_TIME, LocalDateTime.class);  
        Method setCreateUser = entity.getClass().getDeclaredMethod(AutoFillConstant.SET_CREATE_USER, Long.class);  
        Method setUpdateUser = entity.getClass().getDeclaredMethod(AutoFillConstant.SET_UPDATE_USER, Long.class);  
        // 通过反射调用目标对象的方法  
        setCreateTime.invoke(entity, time);  
        setUpdateTime.invoke(entity, time);  
        setCreateUser.invoke(entity, empId);  
        setUpdateUser.invoke(entity, empId);  
    } catch (Exception ex) {  
        log.error("公共字段自动填充失败：{}", ex.getMessage());  
    }  
}
```



代码开发

- 完善自定义切面 AutoFillAspect 的 autoFill 方法

```
else {
    // 当前执行的是 update 操作，为 2 个字段赋值
    try {
        // 获得 set 方法对象 ----Method
        Method setUpdateTime = entity.getClass().getDeclaredMethod(AutoFillConstant.SET_UPDATE_TIME, LocalDateTime.class);
        Method setUpdateUser = entity.getClass().getDeclaredMethod(AutoFillConstant.SET_UPDATE_USER, Long.class);
        // 通过反射调用目标对象的方法
        setUpdateTime.invoke(entity, time);
        setUpdateUser.invoke(entity, empId);
    } catch (Exception ex) {
        Log.error(" 公共字段自动填充失败 : {}", ex.getMessage());
    }
}
}
```



代码开发

- 在 Mapper 接口的方法上加入 AutoFill 注解

```
/**  
 * 插入数据  
 * @param category  
 */  
@AutoFill(OperationType.INSERT)  
@Insert("insert into category(type, name, sort, status, create_time, update_time, create_user, update_user)" +  
    " VALUES" +  
    " (#{}type}, #{}name}, #{}sort}, #{}status}, #{}createTime}, #{}updateTime}, #{}createUser}, #{}updateUser})")  
void insert(Category category);  
  
/**  
 * 根据 id 修改分类  
 * @param category  
 */  
@AutoFill(OperationType.UPDATE)  
void update(Category category);
```

代码开发

- 将业务层为公共字段赋值的代码注释掉



公共字段自动填充

- 问题分析
- 实现思路
- 代码开发
- 功能测试



功能测试

通过观察控制台输出的 SQL 来确定公共字段填充是否完成

```
开始进行公共字段自动填充...
==> Preparing: update employee SET name = ?, username = ?, phone = ?, sex = ?, id_Number = ?, update_Time = ?, update_User = ? where id = ?
==> Parameters: 王五(String), wangwu(String), 13312345678(String), 1(String), 222333444555666111(String), 2022-10-10T15:27:28.372093900(LocalDateTime), 1(Long), 14(Long)
<==   Updates: 1
```



目录

Contents

- ◆ 公共字段自动填充
- ◆ 新增菜品
- ◆ 菜品分页查询
- ◆ 删除菜品
- ◆ 修改菜品



新增菜品

- 需求分析和设计
- 代码开发
- 功能测试



需求分析和设计

产品原型：

< 新建菜品

*菜品名称:

*菜品分类:

*菜品价格: 元

口味做法配置: 口味名 (3个字内) 口味标签 (输入后, 回车添加)

甜度 半糖 X 半糖 X 半糖 X

*菜品图片:
图片大小不超过2M
仅能上传PNG JPG JPEG类型图片
建议上传200*200或300*300尺寸的图片

菜品描述: 菜品描述, 最长200字

需求分析和设计

业务规则：

- 菜品名称必须是唯一的
- 菜品必须属于某个分类下，不能单独存在
- 新增菜品时可以根据情况选择菜品的口味
- 每个菜品必须对应一张图片



需求分析和设计

接口设计：

- 根据类型查询分类（已完成）
- 文件上传
- 新增菜品

< 新建菜品

*菜品名称:

*菜品分类:

*菜品价格: 元

口味做法配置: 口味名 (3个字内) 口味标签 (输入后, 回车添加)

*菜品图片: 图片大小不超过2M
仅能上传PNG JPG JPEG类型图片
建议上传200*200或300*300尺寸的图片

菜品描述:



需求分析和设计

接口设计：

- 根据类型查询分类

基本信息

Path: /admin/category/list

Method: GET

接口描述：

请求参数

Query

参数名称	是否必须	示例	备注
type	否	2	分类类型：1为菜品分类，2为套餐分类

返回数据

名称	类型	是否必须	默认值	备注	其他信息
code	integer	必须			format: int32
data	object []	非必须			item 类型: object
— createTime	string	非必须			format: date-time
— createUser	integer	非必须			format: int64
— id	integer	非必须			format: int64
— name	string	非必须			
— sort	integer	非必须			format: int32
— type	integer	非必须			format: int32
— updateTime	string	非必须			format: date-time
— updateUser	integer	非必须			format: int64
msg	string	非必须			

需求分析和设计

接口设计：

- 文件上传

基本信息

Path: /admin/common/upload

Method: POST

接口描述:

请求参数

Headers

参数名称	参数值	是否必须	示例	备注
Content-Type	multipart/form-data	是		

Body

参数名称	参数类型	是否必须	示例	备注
file	file	是		文件

返回数据

名称	类型	是否必须	默认值	备注	其他信息
code	integer	必须			format: int32
data	string	必须		文件上传路径	
msg	string	非必须			



需求分析和设计

接口设计：

- 新增菜品

基本信息

Path: /admin/dish

Method: POST

接口描述：

请求参数

Headers

参数名称	参数值	是否必须	示例	备注
Content-Type	application/json	是		

Body

名称	类型	是否必须	默认值	备注	其他信息
categoryId	integer	必须		分类id	format: int64
description	string	非必须		菜品描述	
flavors	object []	非必须		口味	item 类型: object
└ dishId	integer	非必须		菜品id	format: int64
└ id	integer	非必须		口味id	format: int64
└ name	string	必须		口味名称	
└ value	string	必须		口味值	
id	integer	非必须		菜品id	format: int64
image	string	必须		菜品图片路径	
name	string	必须		菜品名称	
price	number	必须		菜品价格	
status	integer	非必须		菜品状态: 1为起售, 0为停售	format: int32

返回数据

名称	类型	是否必须	默认值	备注	其他信息
code	integer	必须			format: int32
data	object	非必须			
msg	string	非必须			

需求分析和设计

数据库设计（ dish 菜品表和 dish_flavor 口味表 ）：

字段名	数据类型	说明	备注
id	bigint	主键	自增
name	varchar(32)	菜品名称	唯一
category_id	bigint	分类 id	逻辑外键
price	decimal(10,2)	菜品价格	
image	varchar(255)	图片路径	
description	varchar(255)	菜品描述	
status	int	售卖状态	1 起售 0 停售
create_time	datetime	创建时间	
update_time	datetime	最后修改时间	
create_user	bigint	创建人 id	
update_user	bigint	最后修改人 id	

字段名	数据类型	说明	备注
id	bigint	主键	自增
dish_id	bigint	菜品 id	逻辑外键
name	varchar(32)	口味名称	
value	varchar(255)	口味值	



新增菜品

- 需求分析和设计
- 代码开发
- 功能测试

代码开发

开发文件上传接口：





代码开发

开发文件上传接口：

```
sky:  
  alioss:  
    endpoint: oss-cn-beijing.aliyuncs.com  
    access-key-id: LTAI5tPeFLzsPPT8gG3LPW64  
    access-key-secret: U6k1br0Z8ga0IXv3nXbulGTUzy6Pd7  
    bucket-name: sky-itcast
```

application-dev.yml

```
sky:  
  alioss:  
    endpoint: ${sky.alioss.endpoint}  
    access-key-id: ${sky.alioss.access-key-id}  
    access-key-secret: ${sky.alioss.access-key-secret}  
    bucket-name: ${sky.alioss.bucket-name}
```

application.yml



代码开发

开发文件上传接口：

```
@Configuration
@Slf4j
public class OssConfiguration {

    /**
     * 通过spring 管理对象
     *
     * @param aliOssProperties
     * @return
     */
    @Bean
    @ConditionalOnMissingBean
    public AliOssUtil aliOssUtil(AliOssProperties aliOssProperties) {
        Log.info("开始创建阿里云 OSS 工具类 ...");
        return new AliOssUtil(aliOssProperties.getEndpoint(),
            aliOssProperties.getAccessKeyId(),
            aliOssProperties.getAccessKeySecret(),
            aliOssProperties.getBucketName());
    }

}
```

OssConfiguration



代码开发

开发文件上传接口：

```
@RestController
@RequestMapping("/admin/common")
@Slf4j
@Api(tags = "通用接口")
public class CommonController {
    @Autowired
    private AliOssUtil aliOssUtil;

    /**
     * 文件上传
     * @param file
     * @return
     */
    @PostMapping("/upload")
    @ApiOperation("文件上传")
    public Result<String> upload(MultipartFile file){
        log.info(file.getName());

        // 原始文件名
        String originalFilename = file.getOriginalFilename();
        String extension = originalFilename.substring(originalFilename.lastIndexOf("."));

        // 将文件上传的阿里云
        String fileName = UUID.randomUUID().toString() + extension;
        try {
            String filePath = aliOssUtil.upload(file.getBytes(), fileName);
            return Result.success(filePath);
        } catch (IOException e) {
            log.error("文件上传失败 :{}", e.getMessage());
        }

        return Result.error(MessageConstant.UPLOAD_FAILED);
    }
}
```

OssConfiguration



代码开发

开发新增菜品接口：

Body

名称	类型	是否必须	默认值	备注
categoryId	integer	必须		分类id
description	string	非必须		菜品描述
flavors	object []	非必须		口味
- dishId	integer	非必须		菜品id
- id	integer	非必须		口味id
- name	string	必须		口味名称
- value	string	必须		口味值
id	integer	非必须		菜品id
image	string	必须		菜品图片路径
name	string	必须		菜品名称
price	number	必须		菜品价格
status	integer	非必须		菜品状态：1为起售，0为停售



```
@Data
public class DishDTO implements Serializable {

    private Long id;
    // 菜品名称
    private String name;
    // 菜品分类 id
    private Long categoryId;
    // 菜品价格
    private BigDecimal price;
    // 图片
    private String image;
    // 描述信息
    private String description;
    // 0 停售 1 起售
    private Integer status;
    // 口味
    private List<DishFlavor> flavors = new ArrayList<>();

}
```



代码开发

开发新增菜品接口：

```
@RestController
@RequestMapping("/admin/dish")
@Api(tags = "菜品相关接口")
@Slf4j
public class DishController {

    @Autowired
    private DishService dishService;

    /**
     * 新增菜品
     * @param dishDTO
     * @return
     */
    @PostMapping
    @ApiOperation("新增菜品")
    public Result<String> save(@RequestBody DishDTO dishDTO){
        log.info("新增菜品：{}", dishDTO);
        dishService.saveWithFlavor(dishDTO);

        return Result.success();
    }
}
```

DishController

代码开发

开发新增菜品接口：

```
public interface DishService {  
  
    /**  
     * 新增菜品  
     * @param dishDTO  
     */  
    void saveWithFlavor(DishDTO dishDTO);  
}
```

DishService



代码开发

开发新增菜品接口：

```
@Service
public class DishServiceImpl implements DishService {

    @Autowired
    private DishMapper dishMapper;
    @Autowired
    private DishFlavorMapper dishFlavorMapper;

    /**
     * 新增菜品
     * @param dishDTO
     */
    @Transactional
    public void saveWithFlavor(DishDTO dishDTO) {
        Dish dish = new Dish();
        BeanUtils.copyProperties(dishDTO, dish);
        // 向菜品表dish 插入 1 条数据
        dishMapper.insert(dish);

        // 获取菜品的主键值
        Long dishId = dish.getId();

        List<DishFlavor> flavors = dishDTO.getFlavors();
        if(flavors != null && flavors.size() > 0){
            // 向口味表dish_flavor 插入 n 条
            flavors.forEach(dishFlavor -> {
                dishFlavor.setDishId(dishId);
            });
            // 批量插入
            dishFlavorMapper.insertBatch(flavors);
        }
    }
}
```

DishServiceImpl



代码开发

开发新增菜品接口：

```
/**  
 * 插入菜品数据  
 * @param dish  
 */  
@AutoFill(OperationType.INSERT)  
void insert(Dish dish);
```

DishMapper

```
<!--  
    useGeneratedKeys:true 表示获取主键值  
    keyProperty="id" 表示将主键值赋给 id 属性  
-->  
<insert id="insert" useGeneratedKeys="true" keyProperty="id">  
    insert into dish  
        (status, name, category_id, price, image, description, create_time, update_time, create_user, update_user)  
    values  
        (#{$status}, #{$name}, #{$categoryId}, #{$price}, #{$image}, #{$description}, #{$createTime}, #{$updateTime}, #{$createUser}, #{$updateUser})  
</insert>
```

DishMapper.xml



代码开发

开发新增菜品接口：

```
@Mapper
public interface DishFlavorMapper {
    /**
     * 批量插入口味数据
     * @param flavors
     */
    void insertBatch(List<DishFlavor> flavors);
}
```

DishFlavorMapper

```
<insert id="insertBatch">
    insert into dish_flavor(dish_id, name, value) values
    <foreach collection="flavors" item="dishFlavor" separator=",">
        (#{dishFlavor.dishId},#{dishFlavor.name},#{dishFlavor.value})
    </foreach>
</insert>
```

DishFlavorMapper.xml



新增菜品

- 需求分析和设计
- 代码开发
- 功能测试

功能测试

可以通过接口文档进行测试，也可以进行前后端联调测试



目录

Contents

- ◆ 公共字段自动填充
- ◆ 新增菜品
- ◆ 菜品分页查询
- ◆ 删除菜品
- ◆ 修改菜品



菜品分页查询

- 需求分析和设计
- 代码开发
- 功能测试



需求分析和设计

产品原型：

1 菜品名称 菜品分类 售卖状态 搜索

批量删除 2 新建菜品

	菜品名称	图片	菜品分类	售价	售卖状态	最后操作时间	3 操作
<input type="checkbox"/>	鱼香肉丝		荤菜	¥20.00	启售	2019-02-02 11:11	修改 删除 停售
<input type="checkbox"/>	鱼香肉丝		荤菜	¥20.00	停售	2019-02-02 11:11	修改 删除 启售
<input type="checkbox"/>	鱼香肉丝		荤菜	¥20.00	启售	2019-02-02 11:11	修改 删除 停售
<input type="checkbox"/>	鱼香肉丝		荤菜	¥20.00	启售	2019-02-02 11:11	修改 删除 停售
<input type="checkbox"/>	鱼香肉丝		荤菜	¥20.00	启售	2019-02-02 11:11	修改 删除 停售
<input type="checkbox"/>	鱼香肉丝		荤菜	¥20.00	启售	2019-02-02 11:11	修改 删除 停售
<input type="checkbox"/>	鱼香肉丝		荤菜	¥20.00	启售	2019-02-02 11:11	修改 删除 停售
<input type="checkbox"/>	鱼香肉丝		荤菜	¥20.00	启售	2019-02-02 11:11	修改 删除 停售
<input type="checkbox"/>	鱼香肉丝		荤菜	¥20.00	启售	2019-02-02 11:11	修改 删除 停售
<input type="checkbox"/>	鱼香肉丝		荤菜	¥20.00	启售	2019-02-02 11:11	修改 删除 停售

总共 85 个项目 < 1 2 3 4 5 >

需求分析和设计

业务规则：

- 根据页码展示菜品信息
- 每页展示 10 条数据
- 分页查询时可以根据需要输入菜品名称、菜品分类、菜品状态进行查询



需求分析和设计

接口设计：

基本信息

Path: /admin/dish/page

Method: GET

接口描述:

请求参数

Query

参数名称	是否必须	示例	备注
page	是	1	页码
pageSize	是	10	每页记录数
name	否	鱼香肉丝	菜品名称
categoryId	否	1	分类id
status	否	1	菜品售卖状态

返回数据

名称	类型	是否必须	默认值	备注	其他信息
code	integer	必须			format: int32
data	object	必须			
records	object []	非必须			item 类型: object
id	number	必须			
name	string	必须			
price	number	必须			
image	string	必须			
description	string	必须			
status	integer	必须			
updateTime	string	必须			
categoryName	string	必须		分类名称	
total	integer	必须			format: int64
msg	string	非必须			



菜品分页查询

- 需求分析和设计
- 代码开发
- 功能测试



代码开发

根据菜品分页查询接口定义设计对应的 DTO：

请求参数

Query

参数名称	是否必须	示例	备注
page	是	1	页码
pageSize	是	10	每页记录数
name	否	鱼香肉丝	菜品名称
categoryId	否	1	分类id
status	否	1	菜品售卖状态



```
@Data
public class DishPageQueryDTO implements Serializable {

    private int page;

    private int pageSize;

    // 菜品名称
    private String name;

    // 分类 id
    private Integer categoryId;

    // 状态 0 表示禁用 1 表示启用
    private Integer status;
}
```



代码开发

根据菜品分页查询接口定义设计对应的 VO：

返回数据

名称	类型	是否必须	默认值	备注	其他信息
code	integer	必须			format: int32
data	object	必须			
— records	object []	非必须			item 类型: object
— id	number	必须			
— name	string	必须			
— price	number	必须			
— image	string	必须			
— description	string	必须			
— status	integer	必须			
— updateTime	string	必须			
— categoryName	string	必须		分类名称	
— total	integer	必须			format: int64
msg	string	非必须			



```
public class DishVO implements Serializable {  
  
    private Long id;  
    // 菜品名称  
    private String name;  
    // 菜品分类id  
    private Long categoryId;  
    // 菜品价格  
    private BigDecimal price;  
    // 图片  
    private String image;  
    // 描述信息  
    private String description;  
    //0 停售 1 起售  
    private Integer status;  
    // 更新时间  
    private LocalDateTime updateTime;  
    // 分类名称  
    private String categoryName;  
    // 菜品关联的口味  
    private List<DishFlavor> flavors = new ArrayList<>();  
  
}
```

代码开发

根据接口定义创建 DishController 的 page 分页查询方法：

```
/*
 * 菜品分页查询
 * @param dishPageQueryDTO
 * @return
 */
@GetMapping("/page")
@ApiOperation("菜品分页查询 ")
public Result<PageResult> page(DishPageQueryDTO dishPageQueryDTO){
    log.info("菜品分页查询 : {}", dishPageQueryDTO);
    PageResult pageResult = dishService.pageQuery(dishPageQueryDTO);
    return Result.success(pageResult);
}
```

DishController



代码开发

在 DishService 中扩展分页查询方法：

```
/**  
 * 菜品分页查询  
 * @param dishPageQueryDTO  
 * @return  
 */  
PageResult pageQuery(DishPageQueryDTO dishPageQueryDTO);
```

DishService



代码开发

在 DishServiceImpl 中实现分页查询方法：

```
/**  
 * 菜品分页查询  
 * @param dishPageQueryDTO  
 * @return  
 */  
public PageResult pageQuery(DishPageQueryDTO dishPageQueryDTO) {  
    PageHelper.startPage(dishPageQueryDTO.getPage(), dishPageQueryDTO.getPageSize());  
    Page<DishVO> page = dishMapper.pageQuery(dishPageQueryDTO);  
    return new PageResult(page.getTotal(), page.getResult());  
}
```

DishServiceImpl



代码开发

在 DishMapper 接口中声明 pageQuery 方法：

```
/**  
 * 菜品分页查询  
 * @param dishPageQueryDTO  
 * @return  
 */  
Page<DishVO> pageQuery(DishPageQueryDTO dishPageQueryDTO);
```

DishMapper



代码开发

在 DishMapper.xml 中编写 SQL：

```
<select id="pageQuery" resultType="com.sky.vo.DishVO">
    select d.*,c.name categoryName from dish d left join category c on d.category_id = c.id
    <where>
        <if test="name != null">
            and d.name like concat('%',#{name},'%')
        </if>
        <if test="categoryId != null">
            and d.category_id = #{categoryId}
        </if>
        <if test="status != null">
            and d.status = #{status}
        </if>
    </where>
    order by d.create_time desc
</select>
```

DishMapper.xml



菜品分页查询

- 需求分析和设计
- 代码开发
- 功能测试



功能测试

可以通过接口文档进行测试，最后完成前后端联调测试即可



目录

Contents

- ◆ 公共字段自动填充
- ◆ 新增菜品
- ◆ 菜品分页查询
- ◆ 删除菜品
- ◆ 修改菜品



删除菜品

- 需求分析和设计
- 代码开发
- 功能测试



需求分析和设计

产品原型：

	菜品名称	图片	菜品分类	售价	售卖状态	最后操作时间	操作
<input type="checkbox"/>	鱼香肉丝		荤菜	¥20.00	启售	2019-02-02 11:11	修改 删除 停售
<input type="checkbox"/>	鱼香肉丝		荤菜	¥20.00	停售	2019-02-02 11:11	修改 删除 启售
<input type="checkbox"/>	鱼香肉丝		荤菜	¥20.00	启售	2019-02-02 11:11	修改 删除 停售
<input type="checkbox"/>	鱼香肉丝		荤菜	¥20.00	启售	2019-02-02 11:11	修改 删除 停售
<input type="checkbox"/>	鱼香肉丝		荤菜	¥20.00	启售	2019-02-02 11:11	修改 删除 停售
<input type="checkbox"/>	鱼香肉丝		荤菜	¥20.00	启售	2019-02-02 11:11	修改 删除 停售
<input type="checkbox"/>	鱼香肉丝		荤菜	¥20.00	启售	2019-02-02 11:11	修改 删除 停售
<input type="checkbox"/>	鱼香肉丝		荤菜	¥20.00	启售	2019-02-02 11:11	修改 删除 停售
<input type="checkbox"/>	鱼香肉丝		荤菜	¥20.00	启售	2019-02-02 11:11	修改 删除 停售

需求分析和设计

业务规则：

- 可以一次删除一个菜品，也可以批量删除菜品
- 起售中的菜品不能删除
- 被套餐关联的菜品不能删除
- 删除菜品后，关联的口味数据也需要删除掉



需求分析和设计

接口设计：

基本信息

Path: /admin/dish

Method: DELETE

接口描述：

请求参数

Query

参数名称	是否必须	示例	备注
ids	是	1,2,3	菜品id，之间用逗号分隔

返回数据

名称	类型	是否必须	默认值	备注	其他信息
code	number	必须			
data	string	非必须			
msg	string	非必须			



需求分析和设计

数据库设计：

dish表

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G
id	BIGINT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
name	VARCHAR(64)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
category_id	BIGINT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
price	DECIMAL(10,2)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
image	VARCHAR(200)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
description	VARCHAR(400)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
status	INT	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
create_time	DATETIME	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
update_time	DATETIME	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
create_user	BIGINT	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
update_user	BIGINT	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

dish_flavor表

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G
id	BIGINT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>				
dish_id	BIGINT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				
name	VARCHAR(64)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

setmeal_dish表

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G
id	BIGINT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>				
setmeal_id	VARCHAR(32)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				
dish_id	VARCHAR(32)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				
name	VARCHAR(32)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
price	DECIMAL(10,2)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
copies	INT	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>



删除菜品

- 需求分析和设计
- 代码开发
- 功能测试



代码开发

根据删除菜品的接口定义在 DishController 中创建方法：

```
/*
 * 菜品批量删除
 * @param ids
 * @return
 */
@DeleteMapping
@ApiOperation("菜品批量删除")
public Result delete(@RequestParam List<Long> ids){
    log.info("菜品批量删除：{}", ids);
    dishService.deleteBatch(ids);
    return Result.success();
}
```

DishController



代码开发

在 DishService 接口中声明 deleteBatch 方法：

```
/**  
 * 菜品批量删除  
 * @param ids  
 */  
void deleteBatch(List<Long> ids);
```

DishService



代码开发

在 DishServiceImpl 中实现 deleteBatch 方法：

```
@Transactional
public void deleteBatch(List<Long> ids) {
    ids.forEach(id ->{
        Dish dish = dishMapper.getById(id);
        // 判断当前要删除的菜品状态是否为起售中
        if(dish.getStatus() == StatusConstant.ENABLE){
            // 如果是起售中，抛出业务异常
            throw new DeletionNotAllowedException(MessageConstant.DISH_ON_SALE);
        }
    });

    // 判断当前要删除的菜品是否被套餐关联了
    List<Long> setmealIds = setmealDishMapper.getSetmealIdsByDishIds(ids);
    if(setmealIds != null && setmealIds.size() > 0){
        // 如果关联了，抛出业务异常
        throw new DeletionNotAllowedException(MessageConstant.DISH_BE RELATED_BY_SETMEAL);
    }

    // 删除菜品表中的数据
    ids.forEach(id -> {
        dishMapper.deleteById(id);
        // 删除口味表中的数据
        dishFlavorMapper.deleteByDishId(id);
    });
}
```

DishServiceImpl



代码开发

在 DishMapper 中声明 getById 方法，并配置 SQL：

```
/**  
 * 根据主键查询菜品数据  
 * @param id  
 * @return  
 */  
@Select("select * from dish where id = #{id}")  
Dish getById(Long id);
```

DishMapper



代码开发

创建 SetmealDishMapper ，声明 getSetmealIdsByDishIds 方法，并在 xml 文件中编写 SQL :

```
@Mapper
public interface SetmealDishMapper {
    /**
     * 根据菜品 id 查询关联的套餐 id
     * @param ids
     * @return
     */
    List<Long> getSetmealIdsByDishIds(List<Long> ids);
}
```

SetmealDishMapper

```
<select id="getSetmealIdsByDishIds" resultType="java.lang.Long">
    select setmeal_id from setmeal_dish where dish_id in
    <foreach collection="ids" separator="," open="(" close=")" item="dishId">
        #{dishId}
    </foreach>
</select>
```

SetmealDishMapper.xml



代码开发

在 DishMapper.xml 中声明 deleteById 方法并配置 SQL：

```
/*
 * 根据主键删除菜品数据
 * @param id
 */
@Delete("delete from dish where id = #{id}")
void deleteById(Long id);
```

DishMapper



代码开发

在 DishFlavorMapper 中声明 deleteByDishId 方法并配置 SQL：

```
/**  
 * 根据菜品 id 删除口味数据  
 * @param dishId  
 */  
@Delete("delete from dish_flavor where dish_id = #{dishId}")  
void deleteByDishId(Long dishId);
```

DishFlavorMapper



删除菜品

- 需求分析和设计
- 代码开发
- 功能测试

功能测试

通过 Swagger 接口文档进行测试，通过后再前后端联调测试即可



目录

Contents

- ◆ 公共字段自动填充
- ◆ 新增菜品
- ◆ 菜品分页查询
- ◆ 删除菜品
- ◆ 修改菜品



修改菜品

- 需求分析和设计
- 代码开发
- 功能测试



需求分析和设计

产品原型：

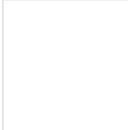
*菜品名称：

*菜品分类：

*菜品价格： 元

口味做法配置：

口味名 (3个字内)	口味标签 (输入后, 回车添加)
甜度	<input type="button" value="半糖 X"/> <input type="button" value="半糖 X"/> <input type="button" value="半糖 X"/> <input type="button" value="半糖 X"/>
	<input type="button" value="删除"/>
	<input type="button" value="删除"/>
<input type="button" value="添加口味"/>	

1 *菜品图片：

图片大小不超过2M
仅能上传PNG JPG JPEG类型图片
建议上传200*200或300*300尺寸的图片

菜品描述：



需求分析和设计

接口设计：

- 根据 id 查询菜品
- 根据类型查询分类（已实现）
- 文件上传（已实现）
- 修改菜品

*菜品名称:

*菜品分类:

*菜品价格: 元

口味做法配置: 口味名 (3个字内) 口味标签 (输入后, 回车添加)

半糖 X 半糖 X 半糖 X 半糖 X

1 *菜品图片:

图片大小不超过2M
仅能上传PNG JPG JPEG类型图片
建议上传200*200或300*300尺寸的图片

菜品描述: 菜品描述, 最长200字



需求分析和设计

接口设计：

- 根据 id 查询菜品

基本信息

Path: /admin/dish/{id}

Method: GET

接口描述:

请求参数

路径参数

参数名称	示例	备注
id	101	菜品id

返回数据

名称	类型	是否必须	默认值	备注	其他信息
data	object	非必须			
└ categoryId	integer	非必须			format: int64
└ description	string	非必须			
└ flavors	object []	非必须			item 类型: object
└ dishId	integer	非必须			format: int64
└ id	integer	非必须			format: int64
└ name	string	非必须			
└ value	string	非必须			
└ id	integer	非必须			format: int64
└ image	string	非必须			
└ name	string	非必须			
└ price	number	非必须			
msg	string	非必须			
code	number	必须			



需求分析和设计

接口设计：

- 修改菜品

基本信息

Path: /admin/dish

Method: PUT

接口描述：

请求参数

Headers

参数名称	参数值	是否必须	示例	备注
Content-Type	application/json	是		

Body

名称	类型	是否必须	默认值	备注	其他信息
categoryId	integer	必须			format: int64
description	string	非必须			
flavors	object []	非必须			item 类型: object
└ dishId	integer	必须			format: int64
└ id	integer	必须			format: int64
└ name	string	必须			
└ value	string	必须			
id	integer	必须			format: int64
image	string	必须			
name	string	必须			
price	number	必须			
status	integer	必须			format: int32

返回数据

名称	类型	是否必须	默认值	备注	其他信息
code	integer	必须			format: int32
data	string	非必须			
msg	string	非必须			



修改菜品

- 需求分析和设计
- 代码开发
- 功能测试



代码开发

根据 id 查询菜品 接口开发：

```
/**  
 * 根据 id 查询菜品和关联的口味数据  
 * @param id  
 * @return  
 */  
@GetMapping("/{id}")  
@ApiOperation("根据 id 查询菜品和关联的口味数据")  
public Result<DishVO> getById(@PathVariable Long id){  
    return Result.success(dishService.getByIdWithFlavor(id));  
}
```

DishController

代码开发

根据 id 查询菜品 接口开发：

```
/**  
 * 根据 id 查询菜品和关联的口味数据  
 * @param id  
 * @return  
 */  
DishVO getByIdWithFlavor(Long id);
```

DishService



代码开发

根据 id 查询菜品 接口开发：

```
/*
 * 根据 id 查询菜品和关联的口味数据
 * @param id
 * @return
 */
public DishVO getByIdWithFlavor(Long id) {
    // 查询菜品表
    Dish dish = dishMapper.getById(id);

    // 查询关联的口味
    List<DishFlavor> dishFlavorList = dishFlavorMapper.getByDishId(id);

    // 封装成 VO
    DishVO dishVO = new DishVO();
    BeanUtils.copyProperties(dish, dishVO);
    dishVO.setFlavors(dishFlavorList);

    return dishVO;
}
```

DishServiceImpl



代码开发

根据 id 查询菜品 接口开发：

```
/**  
 * 根据菜品 id 查询对应的口味  
 * @param dishId  
 * @return  
 */  
  
@Select("select * from dish_flavor where dish_id = #{dishId}")  
List<DishFlavor> getByDishId(Long dishId);
```

DishFlavorMapper



代码开发

修改菜品 接口开发：

```
/**  
 * 修改菜品  
 * @param dishDTO  
 * @return  
 */  
@PutMapping  
@ApiOperation("修改菜品")  
public Result update(@RequestBody DishDTO dishDTO){  
    log.info("修改菜品：{}", dishDTO);  
    dishService.updateWithFlavor(dishDTO);  
    return Result.success();  
}
```

DishController



代码开发

修改菜品 接口开发：

```
/**  
 * 根据 id 修改菜品和关联的口味  
 * @param dishDTO  
 */  
void updateWithFlavor(DishDTO dishDTO);
```

DishService



代码开发

修改菜品 接口开发：

```
/**  
 * 根据 id 修改菜品和关联的口味  
 * @param dishDTO  
 */  
  
@Transactional  
public void updateWithFlavor(DishDTO dishDTO) {  
    Dish dish = new Dish();  
    BeanUtils.copyProperties(dishDTO, dish);  
  
    // 修改菜品表dish , 执行update 操作  
    dishMapper.update(dish);  
  
    // 删除当前菜品关联的口味数据 , 操作dish_flavor , 执行delete 操作  
    dishFlavorMapper.deleteByDishId(dishDTO.getId());  
  
    // 插入最新的口味数据 , 操作dish_flavor , 执行insert 操作  
    List<DishFlavor> flavors = dishDTO.getFlavors();  
    if(flavors != null && flavors.size() > 0){  
        flavors.forEach(dishFlavor -> {  
            dishFlavor.setDishId(dishDTO.getId());  
        });  
        dishFlavorMapper.insertBatch(flavors);  
    }  
}
```

DishServiceImpl



代码开发

修改菜品 接口开发：

```
/**  
 * 根据主键修改菜品信息  
 * @param dish  
 */  
@AutoFill(OperationType.UPDATE)  
void update(Dish dish);
```

DishMapper



代码开发

修改菜品 接口开发：

```
<update id="update">
    update dish
    <set>
        <if test="name != null">
            name = #{name},
        </if>
        <if test="categoryId != null">
            category_id = #{categoryId},
        </if>
        <if test="price != null">
            price = #{price},
        </if>
        <if test="image != null">
            image = #{image},
        </if>
        <if test="description != null">
            description = #{description},
        </if>
        <if test="status != null">
            status = #{status},
        </if>
        <if test="updateTime != null">
            update_time = #{updateTime},
        </if>
        <if test="updateUser != null">
            update_user = #{updateUser},
        </if>
    </set>
    where id = #{id}
</update>
```

DishMapper.xml



修改菜品

- 需求分析和设计
- 代码开发
- 功能测试

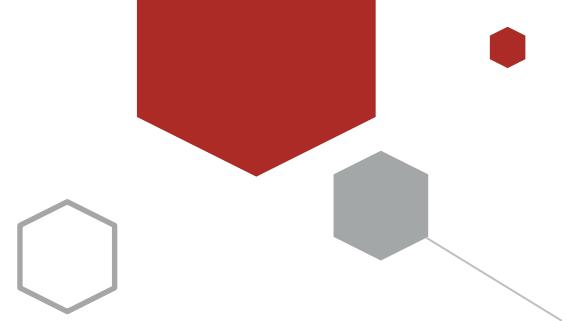
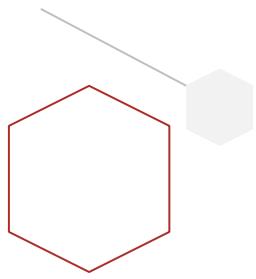


功能测试

通过 Swagger 接口文档进行测试，通过后再前端联调测试即可



传智教育旗下高端IT教育品牌



店铺营业状态设置



黑马程序员 | 传智教育旗下
www.itheima.com 高端IT教育品牌



The screenshot illustrates the process of changing the business status from 'Open' to 'On Hold' (休息中) through a modal dialog.

Left Panel (Workstation): Shows various management functions: Workstation, Data Statistics, Order Management, Package Management (highlighted), Menu Management, Category Management, and Employee Management.

Top Bar: Shows the storefront information: '苍穹外卖 THE SKY TAKE-OUT' (Business Open), distance 1.5km, delivery fee 6 yuan, estimated time 12min, location Beijing Chaoyang District, and a phone icon.

Center Dialog: 营业状态设置 (Business Status Settings)

- 营业中 (Open):** Describes the current state where the restaurant automatically receives orders. It includes a yellow-bordered box around the text and a yellow '+' button next to the status indicator.
- 打烊中 (Closed):** Describes the state where the restaurant only accepts orders during operating hours. It includes a yellow-bordered box around the text and a yellow '-' button next to the status indicator.

Buttons: 取消 (Cancel) and 确定 (Confirm).

Right Panel (Storefront View): Shows the same storefront information but with the status changed to '休息中' (Business On Hold). The background shows a list of menu items with their descriptions and prices.



- ◆ Redis 入门
- ◆ Redis 数据类型
- ◆ Redis 常用命令
- ◆ 在 Java 中操作 Redis
- ◆ 店铺营业状态设置



Redis 入门

- Redis 简介
- Redis 下载与安装
- Redis 服务启动与停止

Redis 简介

Redis 是一个基于内存的 key-value 结构数据库。

- 基于内存存储，读写性能高
- 适合存储热点数据（热点商品、资讯、新闻）
- 企业应用广泛

key	value
id	101
name	小智
city	北京

官网：<https://redis.io>

中文网：<https://www.redis.net.cn/>





Redis 入门

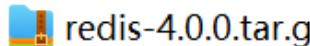
- Redis 简介
- Redis 下载与安装
- Redis 服务启动与停止



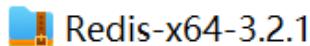
Redis 下载与安装

Redis 安装包分为 Windows 版和 Linux 版：

- Windows 版下载地址：<https://github.com/microsoftarchive/redis/releases>
- Linux 版下载地址：<https://download.redis.io/releases/>



redis-4.0.0.tar.gz



Redis-x64-3.2.100.zip



Redis 下载与安装

Redis 的 Windows 版属于绿色软件，直接解压即可使用，解压后目录结构如下：

- EventLog.dll
- Redis on Windows Release Notes.docx
- Redis on Windows.docx
- redis.windows.conf **Redis配置文件**
- redis.windows-service.conf
- redis-benchmark.exe
- redis-benchmark.pdb
- redis-check-aof.exe
- redis-check-aof.pdb
- redis-cli.exe **Redis客户端**
- redis-cli.pdb
- redis-server.exe **Redis服务端**
- redis-server.pdb
- Windows Service Documentation.docx



Redis 入门

- Redis 简介
- Redis 下载与安装
- Redis 服务启动与停止



Redis 服务启动与停止

服务启动命令：redis-server.exe redis.windows.conf

C:\Windows\System32\cmd.exe - redis-server.exe redis.windows.conf

Microsoft Windows [版本 10.0.19043.2130]
(c) Microsoft Corporation。保留所有权利。

D:\software\Redis-x64-3.2.100>redis-server.exe redis.windows.conf

Redis 3.2.100 (00000000/0) 64 bit
Running in standalone mode
Port: 6379
PID: 21980

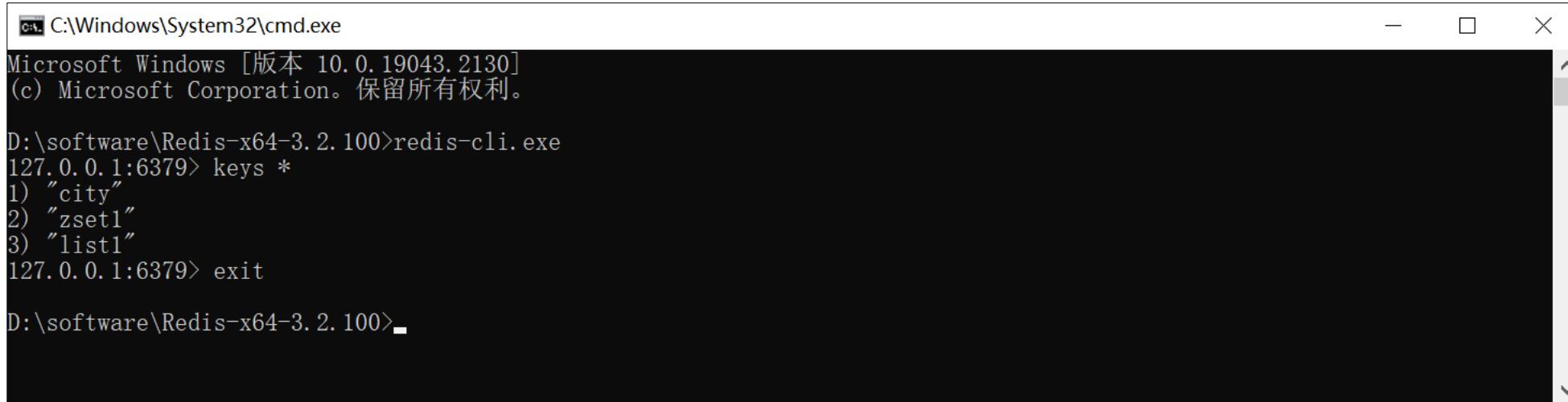
http://redis.io

[21980] 19 Oct 10:22:05.904 # Server started, Redis version 3.2.100
[21980] 19 Oct 10:22:05.912 * DB loaded from disk: 0.004 seconds
[21980] 19 Oct 10:22:05.912 * The server is now ready to accept connections on port 6379

Redis 服务默认端口号为 **6379**，通过快捷键 **Ctrl + C** 即可停止 Redis 服务

Redis 服务启动与停止

客户端连接命令：redis-cli.exe



A screenshot of a Windows Command Prompt window titled 'C:\Windows\System32\cmd.exe'. The window shows the following text:

```
C:\Windows\System32\cmd.exe
Microsoft Windows [版本 10.0.19043.2130]
(c) Microsoft Corporation。保留所有权利。

D:\software\Redis-x64-3.2.100>redis-cli.exe
127.0.0.1:6379> keys *
1) "city"
2) "zset1"
3) "list1"
127.0.0.1:6379> exit

D:\software\Redis-x64-3.2.100>
```

通过 redis-cli.exe 命令默认连接的是本地的 redis 服务，并且使用默认 6379 端口。也可以通过指定如下参数连接：

-h ip 地址

-p 端口号

-a 密码（如果需要）



Redis 服务启动与停止

设置 Redis 服务密码，修改 redis.windows.conf

```
requirepass 123456
```

注意：

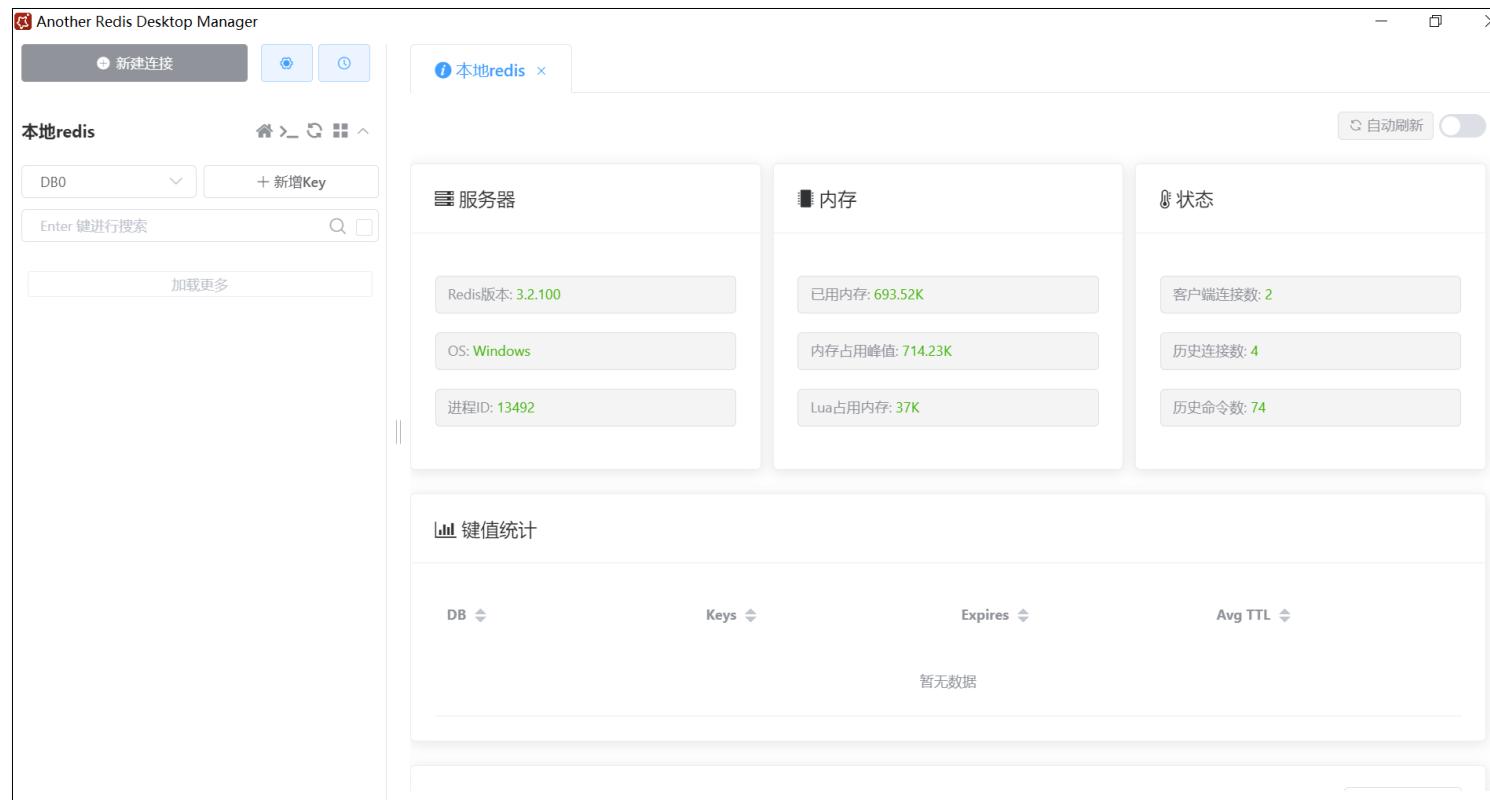
- 修改密码后需要重启 Redis 服务才能生效
- Redis 配置文件中 # 表示注释



Redis 服务启动与停止

Redis 客户端图形工具：

⭐ Another-Redis-Desktop-Manager.1.5.5.exe





- ◆ Redis 入门
- ◆ Redis 数据类型
- ◆ Redis 常用命令
- ◆ 在 Java 中操作 Redis
- ◆ 店铺营业状态设置



Redis 数据类型

- 5 种常用数据类型介绍
- 各种数据类型的特点



5 种常用数据类型介绍

Redis 存储的是 key-value 结构的数据，其中 key 是字符串类型，value 有 5 种常用的数据类型：

- 字符串 string
- 哈希 hash
- 列表 list
- 集合 set
- 有序集合 sorted set / zset

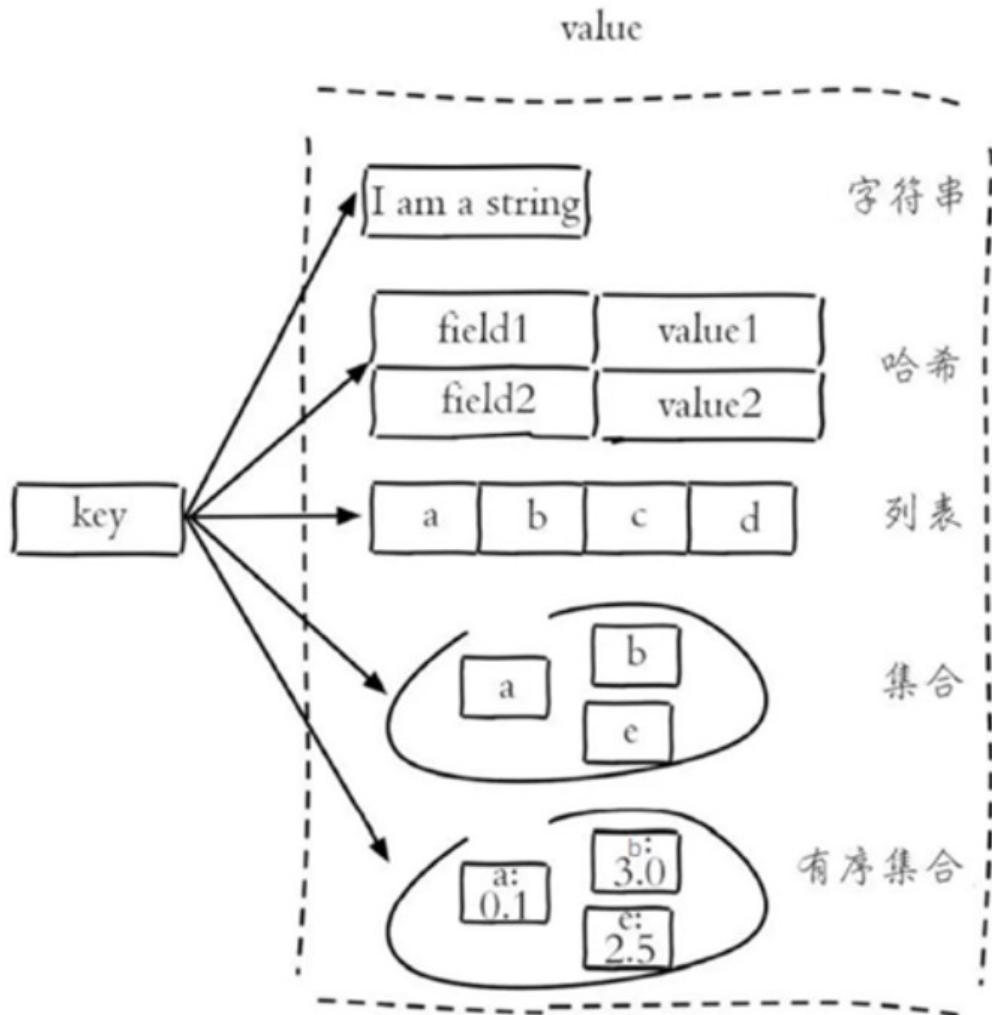


Redis 数据类型

- 5 种常用数据类型介绍
- 各种数据类型的特点



各种数据类型的特点



- 字符串 (string) : 普通字符串，Redis 中最简单的数据类型
- 哈希 (hash) : 也叫散列，类似于 Java 中的 HashMap 结构
- 列表 (list) : 按照插入顺序排序，可以有重复元素，类似于 Java 中的 LinkedList
- 集合 (set) : 无序集合，没有重复元素，类似于 Java 中的 HashSet
- 有序集合 (sorted set / zset) : 集合中每个元素关联一个分数 (score)，根据分数升序排序，没有重复元素



- ◆ Redis 入门
- ◆ Redis 数据类型
- ◆ Redis 常用命令
- ◆ 在 Java 中操作 Redis
- ◆ 店铺营业状态设置



Redis 常用命令

- 字符串操作命令
- 哈希操作命令
- 列表操作命令
- 集合操作命令
- 有序集合操作命令
- 通用命令



字符串操作命令

Redis 字符串类型常用命令：

- SET key value 设置指定 key 的值
- GET key 获取指定 key 的值
- SETEX key seconds value 设置指定 key 的值，并将 key 的过期时间设为 seconds 秒
- SETNX key value 只有在 key 不存在时设置 key 的值



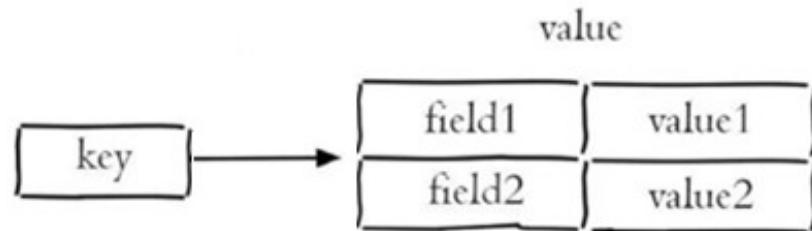
Redis 常用命令

- 字符串操作命令
- 哈希操作命令
- 列表操作命令
- 集合操作命令
- 有序集合操作命令
- 通用命令

哈希操作命令

Redis hash 是一个 string 类型的 field 和 value 的映射表，hash 特别适合用于存储对象，常用命令：

- HSET key field value 将哈希表 key 中的字段 field 的值设为 value
- HGET key field 获取存储在哈希表中指定字段的值
- HDEL key field 删除存储在哈希表中的指定字段
- HKEYS key 获取哈希表中所有字段
- HVALS key 获取哈希表中所有值





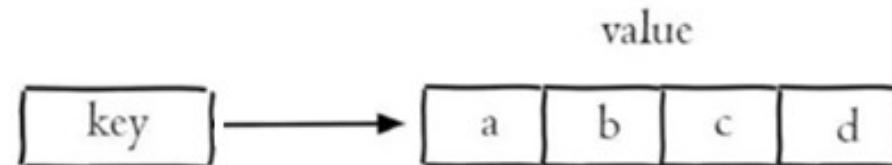
Redis 常用命令

- 字符串操作命令
- 哈希操作命令
- 列表操作命令
- 集合操作命令
- 有序集合操作命令
- 通用命令

列表操作命令

Redis 列表是简单的字符串列表，按照插入顺序排序，常用命令：

- LPUSH key value1 [value2] 将一个或多个值插入到列表头部（左边）
- LRANGE key start stop 获取列表指定范围内的元素
- RPOP key 移除并获取列表最后一个元素（右边）
- LLEN key 获取列表长度





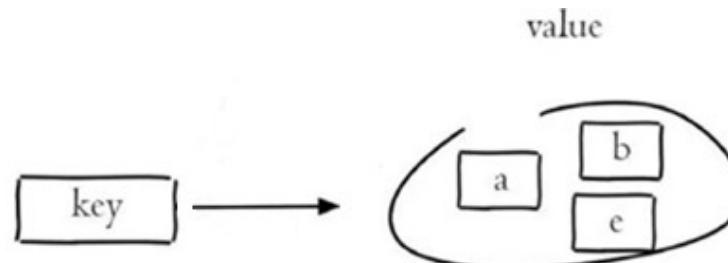
Redis 常用命令

- 字符串操作命令
- 哈希操作命令
- 列表操作命令
- 集合操作命令
- 有序集合操作命令
- 通用命令

集合操作命令

Redis set 是 string 类型的无序集合。集合成员是唯一的，集合中不能出现重复的数据，常用命令：

- SADD key member1 [member2] 向集合添加一个或多个成员
- SMEMBERS key 返回集合中的所有成员
- SCARD key 获取集合的成员数
- SINTER key1 [key2] 返回给定所有集合的交集
- SUNION key1 [key2] 返回所有给定集合的并集
- SREM key member1 [member2] 删除集合中一个或多个成员





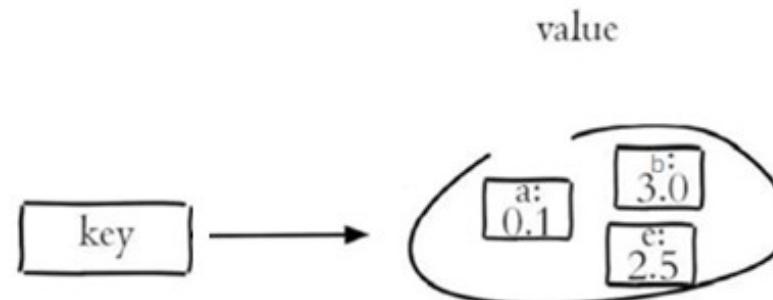
Redis 常用命令

- 字符串操作命令
- 哈希操作命令
- 列表操作命令
- 集合操作命令
- 有序集合操作命令
- 通用命令

有序集合操作命令

Redis 有序集合是 string 类型元素的集合，且不允许有重复成员。每个元素都会关联一个 double 类型的分数。常用命令：

- ZADD key score1 member1 [score2 member2] 向有序集合添加一个或多个成员
- ZRANGE key start stop [WITHSCORES] 通过索引区间返回有序集合中指定区间内的成员
- ZINCRBY key increment member 有序集合中对指定成员的分数加上增量 increment
- ZREM key member [member ...] 移除有序集合中的一个或多个成员





Redis 常用命令

- 字符串操作命令
- 哈希操作命令
- 列表操作命令
- 集合操作命令
- 有序集合操作命令
- 通用命令

通用命令

Redis 的通用命令是不分数据类型的，都可以使用的命令：

- KEYS pattern 查找所有符合给定模式 (pattern) 的 key
- EXISTS key 检查给定 key 是否存在
- TYPE key 返回 key 所储存的值的类型
- DEL key 该命令用于在 key 存在时删除 key



- ◆ Redis 入门
- ◆ Redis 数据类型
- ◆ Redis 常用命令
- ◆ 在 Java 中操作 Redis
- ◆ 店铺营业状态设置



在 Java 中操作 Redis

- Redis 的 Java 客户端
- Spring Data Redis 使用方式



Redis 的 Java 客户端

Redis 的 Java 客户端很多，常用的几种：

- Jedis
- Lettuce
- Spring Data Redis

Spring Data Redis 是 Spring 的一部分，对 Redis 底层开发包进行了高度封装。

在 Spring 项目中，可以使用 Spring Data Redis 来简化操作。



在 Java 中操作 Redis

- Redis 的 Java 客户端
- Spring Data Redis 使用方式



Spring Data Redis 使用方式

操作步骤：

- ① 导入 Spring Data Redis 的 maven 坐标
- ② 配置 Redis 数据源
- ③ 编写配置类，创建 RedisTemplate 对象
- ④ 通过 RedisTemplate 对象操作 Redis

```
@Configuration
@Slf4j
public class RedisConfiguration {

    @Bean
    public RedisTemplate redisTemplate(RedisConnectionFactory redisConnectionFactory){
        Log.info("开始创建 redis 模板类 ...");
        RedisTemplate redisTemplate = new RedisTemplate();
        // 设置Key的序列化器，默认为JdkSerializationRedisSerializer
        redisTemplate.setKeySerializer(new StringRedisSerializer());
        redisTemplate.setConnectionFactory(redisConnectionFactory);
        return redisTemplate;
    }

}
```



Spring Data Redis 使用方式

RedisTemplate 针对大量 api 进行了归类封装，将同一数据类型的操作封装为对应的 Operation 接口，具体分类如下：

- ValueOperations : string 数据操作
- SetOperations : set 类型数据操作
- ZSetOperations : zset 类型数据操作
- HashOperations : hash 类型的数据操作
- ListOperations : list 类型的数据操作



- ◆ Redis 入门
- ◆ Redis 数据类型
- ◆ Redis 常用命令
- ◆ 在 Java 中操作 Redis
- ◆ 店铺营业状态设置



店铺营业状态设置

- 需求分析和设计
- 代码开发
- 功能测试



需求分析和设计

产品原型：

The screenshot shows the Beiguo Takeout management system interface. On the left sidebar, there are links for Workstation, Order Management, Category Management, Product Management, Set Meal Management, Data Statistics, and Employee Management. The main area displays today's data (April 22, 2022):

- 营业额: ￥200.13
- 有效订单: 12
- 订单完成率: 16%
- 平均客单价: 30
- 新增用户: 18

Below this, there are two sections:

- 订单管理:** Shows counts for 待接单 (12), 待派送 (10), 已完成 (18), 已取消 (1), and 全部订单 (41).
- 菜品/套餐总览:** Shows counts for 已启售 (12) and 已停售 (2). There are also buttons for 新增菜品 and 新增套餐.

At the bottom, there is a table for 订单信息 (Order Information) with columns: 订单号, 订单菜品, 地址, 预计送达时间, 实收金额, 备注, and 操作. Several orders are listed, each with a "查看" (View) button in the operations column.

更改营业状态

默认

营业 当前餐厅处于营业状态，自动接收任何订单，可点击打烊进入店铺打烊状态

打烊 当前餐厅处于打烊状态，不接受任何订单，可点击营业手动恢复营业状态

状态	状态说明
营业	客户可在小程序下单点餐
打烊	客户无法下单点餐



需求分析和设计

接口设计：

- 设置营业状态
- 管理端查询营业状态
- 用户端查询营业状态

本项目约定：

- **管理端** 发出的请求，统一使用 `GET` 方式
- **用户端** 发出的请求，统一使用 `POST` 方式

基本信息

Path: /user/shop/status

Method: GET

接口描述：

请求参数

返回数据

名称	类型	是否必须	默认值	备注
code	integer	必须		
data	integer	必须		店铺状态：1为营业，0为打烊
msg	string	非必须		

基本信息

Path: /admin/shop/status

Method: GET

接口描述：

请求参数

返回数据

名称	类型	是否必须	默认值	备注
code	integer	必须		
data	integer	必须		店铺营业状态：1为营业，0为打烊
msg	string	非必须		

返回数据

名称	类型	是否必须	默认值	备注	其他信息
code	integer	必须			format: int32
data	string	非必须			
msg	string	非必须			

需求分析和设计

营业状态数据存储方式：基于 Redis 的字符串来进行存储

key	value
SHOP_STATUS	1

约定：**1** 表示营业 **0** 表示打烊



店铺营业状态设置

- 需求分析和设计
- 代码开发
- 功能测试



代码开发

```
@RestController("adminShopController")
@RequestMapping("/admin/shop")
@Api(tags = "店铺操作相关接口")
@Slf4j
public class ShopController {

    public static final String KEY = "SHOP_STATUS";

    @Autowired
    private RedisTemplate redisTemplate;

    /**
     * 设置店铺营业状态
     *
     * @param status
     * @return
     */
    @PutMapping("/{status}")
    @ApiOperation("设置店铺营业状态")
    public Result<String> setStatus(@PathVariable Integer status) {
        log.info("设置营业状态为：{}", status == 1 ? "营业中" : "打烊中");
        redisTemplate.opsForValue().set(KEY, status);
        return Result.success();
    }
}
```



代码开发

```
/**  
 * 查询店铺营业状态  
 *  
 * @return  
 */  
@GetMapping("/status")  
@ApiOperation("查询店铺营业状态")  
public Result<Integer> getStatus() {  
    Integer status = (Integer) redisTemplate.opsForValue().get(KEY);  
    log.info("查询店铺营业状态为：{}", status == 1 ? "营业中" : "打烊中");  
    return Result.success(status);  
}
```



代码开发

```
@RestController("userShopController")
@RequestMapping("/user/shop")
@Api(tags = "店铺操作相关接口")
@Slf4j
public class ShopController {

    public static final String KEY = "SHOP_STATUS";

    @Autowired
    private RedisTemplate redisTemplate;

    /**
     * 查询店铺营业状态
     *
     * @return
     */
    @GetMapping("/status")
    @ApiOperation("查询店铺营业状态")
    public Result<Integer> getStatus() {
        Integer status = (Integer) redisTemplate.opsForValue().get(KEY);
        log.info("查询店铺营业状态为：{}", status == 1 ? "营业中" : "打烊中");
        return Result.success(status);
    }
}
```



店铺营业状态设置

- 需求分析和设计
- 代码开发
- 功能测试

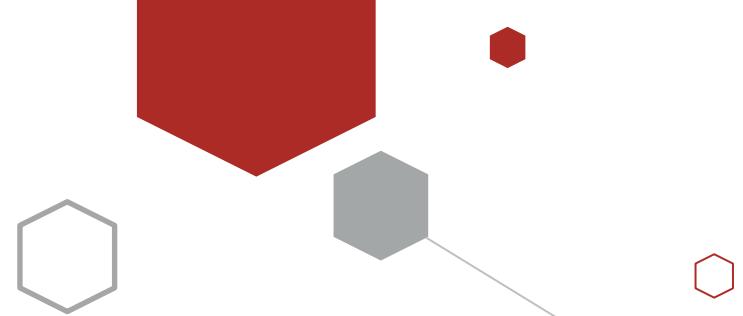
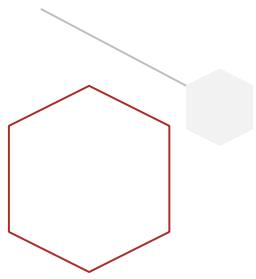
功能测试

可以通过如下方式进行测试：

- swagger 接口文档测试
- 前后端联调测试



传智教育旗下高端IT教育品牌



微信登录、商品浏览



黑马程序员
www.itheima.com

传智教育旗下
高端IT教育品牌

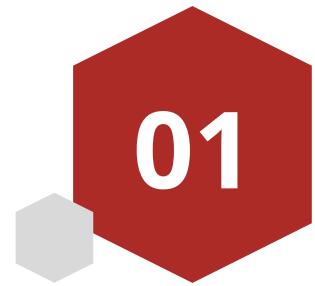




目录

Contents

- ◆ HttpClient
- ◆ 微信小程序开发
- ◆ 微信登录
- ◆ 导入商品浏览功能代码



HttpClient

- 介绍
- 入门案例

HttpClient 介绍 – 作用

HttpClient 是 Apache 的一个子项目，是高效的、功能丰富的支持 HTTP 协议的客户端编程工具包。



HttpClient 作用：

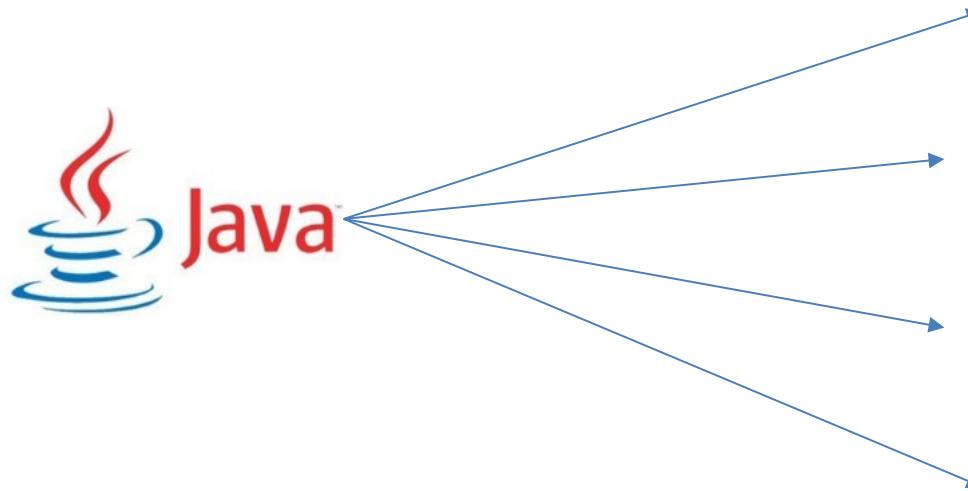
- 发送 HTTP 请求
- 接收响应数据



为什么要在 Java 程序中发送 Http 请求？



HttpClient 介绍 – 应用场景



微信服务



地图服务



短信服务



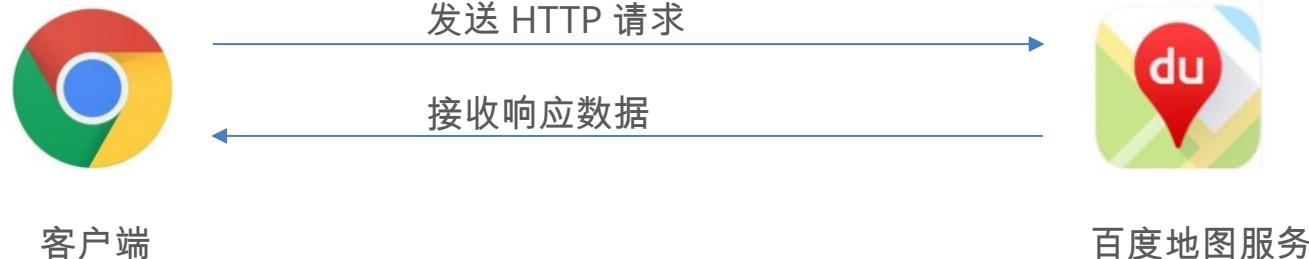
天气预报服务



HttpClient 介绍 – 效果展示

百度地图地理编码服务接口：

`https://api.map.baidu.com/geocoding/v3/?address=北京市海淀区上地十街 10 号 &output=json&ak=UEBQm9c3KZ5Lrs02C2qs0As1eSdLv1zM`



通过浏览器请求服务的步骤：

1. 构造请求地址和参数
2. 发送请求
3. 接收数据

介绍

HttpClient 是 Apache Jakarta Common 下的子项目，可以用来提供高效的、最新的、功能丰富的支持 HTTP 协议的客户端编程工具包，并且它支持 HTTP 协议最新的版本和建议。

```
<dependency>
    <groupId>org.apache.httpcomponents</groupId>
    <artifactId>httpclient</artifactId>
    <version>4.5.13</version>
</dependency>
```

核心 API :

- HttpClient
- HttpClients
- CloseableHttpClient
- HttpGet
- HttpPost

发送请求步骤：

- 创建 HttpClient 对象
- 创建 Http 请求对象
- 调用 HttpClient 的 execute 方法发送请求



HttpClient

- 介绍
- 入门案例



入门案例

GET 方式请求：

```
//http 客户端对象，可以发送 http 请求
CloseableHttpClient httpClient = HttpClients.createDefault();

// 构造 Get 方式请求
HttpGet httpGet = new HttpGet("http://localhost:8080/user/shop/status");

// 发送请求
CloseableHttpResponse response = httpClient.execute(httpGet);

//http 响应码
int statusCode = response.getStatusLine().getStatusCode();

//http 响应体
HttpEntity entity = response.getEntity();
// 将响应体转为 String 字符串
String body = EntityUtils.toString(entity);
System.out.println(body);

// 关闭资源
response.close();
httpClient.close();
```



入门案例

POST 方式请求：

```
CloseableHttpClient httpClient = HttpClients.createDefault();
//Post 方式请求
HttpPost httpPost = new HttpPost("http://localhost:8080/admin/employee/login");

// 构造 json 数据
JSONObject jsonObject = new JSONObject();
jsonObject.put("username", "admin");
jsonObject.put("password", "123456");

// 构造请求体
StringEntity stringEntity = new StringEntity(jsonObject.toString());
// 设置请求编码
stringEntity.setContentEncoding("utf-8");
// 设置数据类型
stringEntity.setContentType("application/json");
// 设置当前 Post 请求的请求体
httpPost.setEntity(stringEntity);

// 发送请求
CloseableHttpResponse response = httpClient.execute(httpPost);

//http 响应码
int statusCode = response.getStatusLine().getStatusCode();

//http 响应体
HttpEntity entity = response.getEntity();
// 将响应体转为 String 字符串
String body = EntityUtils.toString(entity);
System.out.println(body);

// 关闭资源
response.close();
httpClient.close();
```



目录

Contents

- ◆ HttpClient
- ◆ 微信小程序开发
- ◆ 微信登录
- ◆ 导入商品浏览功能代码



微信小程序开发

- 介绍
- 准备工作
- 入门案例

介绍



https://mp.weixin.qq.com/cgi-bin/wx?token=&lang=zh_CN



介绍

开放注册范围



个人



企业



政府



媒体



其他组织



介绍

开发支持

提供一系列工具帮助开发者快速接入并完成小程序开发。



开发文档



开发者工具



设计指南



小程序体验DEMO



介绍

接入流程

1 注册

在微信公众平台注册小程序，完成注册后可以同步进行信息完善和开发。

2 小程序信息完善

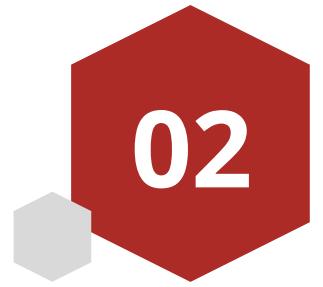
填写小程序基本信息，包括名称、头像、介绍及服务范围等。

3 开发小程序

完成小程序开发者绑定、开发信息配置后，开发者可下载开发者工具、参考开发文档进行小程序的开发和调试。

4 提交审核和发布

完成小程序开发后，提交代码至微信团队审核，审核通过后即可发布（公测期间不能发布）。



微信小程序开发

- 介绍
- 准备工作
- 入门案例

准备工作

开发微信小程序之前需要做如下准备工作：

- 注册小程序
- 完善小程序信息
- 下载开发者工具



准备工作 – 注册小程序

注册地址：

<https://mp.weixin.qq.com/wxopen/waregister?action=step1>

小程序注册

咨询客服

① 帐号信息 —— ② 邮箱激活 —— ③ 信息登记

每个邮箱仅能申请一个小程序

已有微信小程序? 立即登录

创建测试号, 免注册快速体验小程序开发。立即申请

邮箱 作为登录帐号, 请填写未被微信公众平台注册, 未被微信开放平台注册, 未被个人微信号绑定的邮箱

密码 字母、数字或者英文符号, 最短8位, 区分大小写

确认密码 请再次输入密码

验证码 XCgn 换一张

你已阅读并同意《微信公众平台服务协议》及《微信小程序平台服务条款》

准备工作 – 完善小程序信息

登录小程序后台：<https://mp.weixin.qq.com/>





准备工作 – 完善小程序信息

完善小程序信息、小程序类目

The screenshot shows the WeChat Mini Program Management Platform interface. On the left, there's a sidebar with navigation links: 首页 (Home), 管理 (Management) with sub-links 版本管理 (Version Management), 成员管理 (Member Management), and 用户反馈 (User Feedback); 统计 (Statistics); 功能 (Features) with sub-links 微信搜一搜 (WeChat Scan), 客服 (Customer Service), 订阅消息 (Subscription Messages), 直播 (Live Broadcast), 页面内容接入 (Page Content Integration), 实验工具 (Experimental Tools), and 小程序联盟 (Mini Program Alliance). The main content area is titled "小程序发布流程" (Mini Program Publishing Process). Step 1, "小程序信息" (Mini Program Information), is completed. Step 2, "小程序类目" (Mini Program Category), is also completed. Step 3, "小程序开发与管理" (Mini Program Development and Management), is partially visible. It includes sections for "自己开发" (Self-developed) and "找服务商开发" (Find Service Provider Development), each with sub-sections like 开发工具 (Development Tools), 添加开发者 (Add Developers), 配置服务器 (Configure Servers), and 帮助文档 (Help Documents).



准备工作 – 完善小程序信息

查看小程序的 AppID

The screenshot shows the WeChat Developer Platform interface. On the left, there's a sidebar with categories like Home, Management, Statistics, Functions, and Development. Under Development, the 'Development Management' option is highlighted with a red box. The main content area is titled 'Development Management' and includes tabs for Operation Center, Monitoring & Alerting, Development Settings (which is selected), Interface Settings, and Security Center. The 'Developer ID' section contains fields for 'AppID (小程序ID)' and 'AppSecret (小程序密钥)'. The 'AppID' field is highlighted with a red box. Below it, there's a section for '小程序代码上传' (Small Program Code Upload) with a 'Configuration Information' sub-section. At the bottom, there are sections for 'IP Whitelist' and 'Cloud Service'.



准备工作 – 下载开发者工具

下载地址：<https://developers.weixin.qq.com/miniprogram/dev/devtools/stable.html>

The screenshot shows the official WeChat developer documentation website. The top navigation bar includes links for 微信官方文档, 小程序, 开发, 介绍, 设计, 运营, 数据, 社区, 中文, EN, 搜索内容. Below the navigation is a toolbar with buttons for 指南, 框架, 组件, API, 平台能力, 服务端, 工具 (highlighted in green), 云开发, 云托管, and 更新日志. On the left, a sidebar menu lists: 小程序调试, 开发模式, 开发辅助, 小程序自动化, 工具插件, API 实现差异, 下载 (expanded), 稳定版更新日志 (selected), 预发布版更新日志, 开发版更新日志, 小游戏版更新日志, and 历史更新日志. The main content area displays the "Stable Build Update Log" for version 1.05.2204250, which supports Windows 64, Windows 32, and macOS. It also lists version 1.06.2204250 for macOS (ARM64). A note indicates that both versions 1.05.2204250 and 1.06.2204250 have been updated. The log details three fixes: 1. F 修复懒注入 wxml 热重载失效; 2. F 懒注入 relaunch 有概率virtual dom报错, 页面部分白屏; 3. A 新增小程序大赛工具提交作品插件.

准备工作 – 下载开发者工具

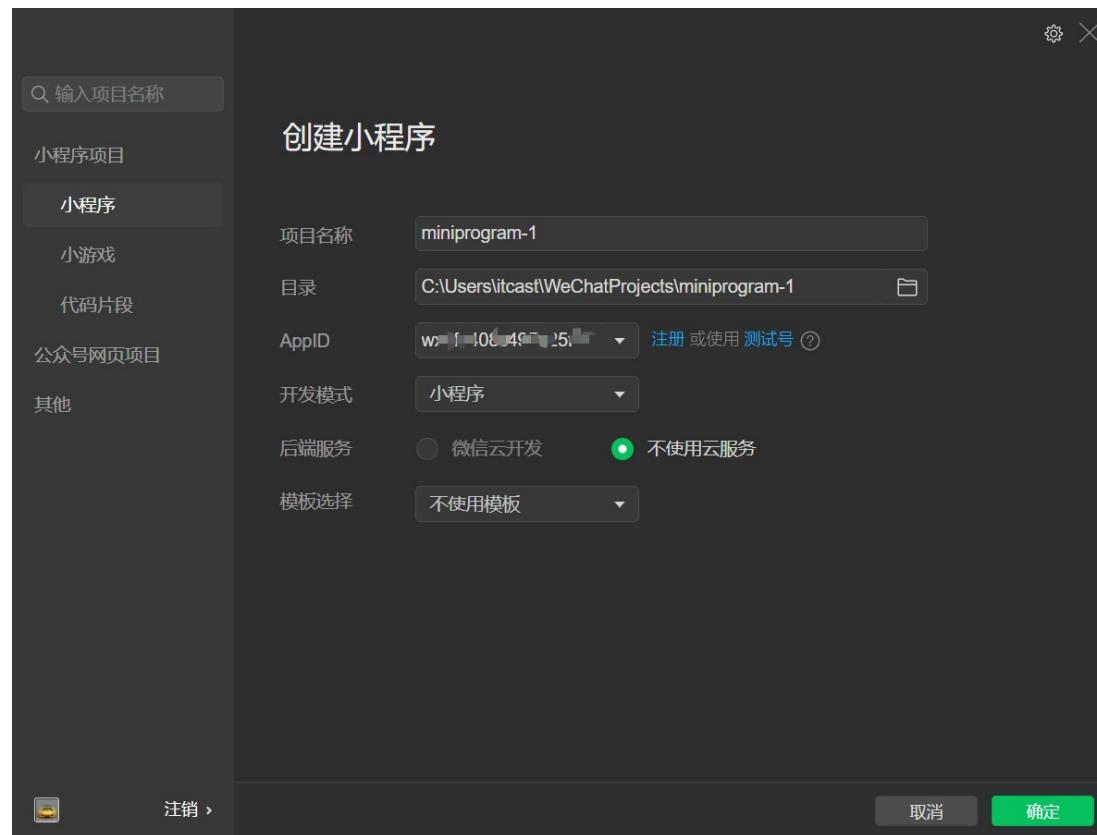
扫描登录开发者工具





准备工作 – 下载开发者工具

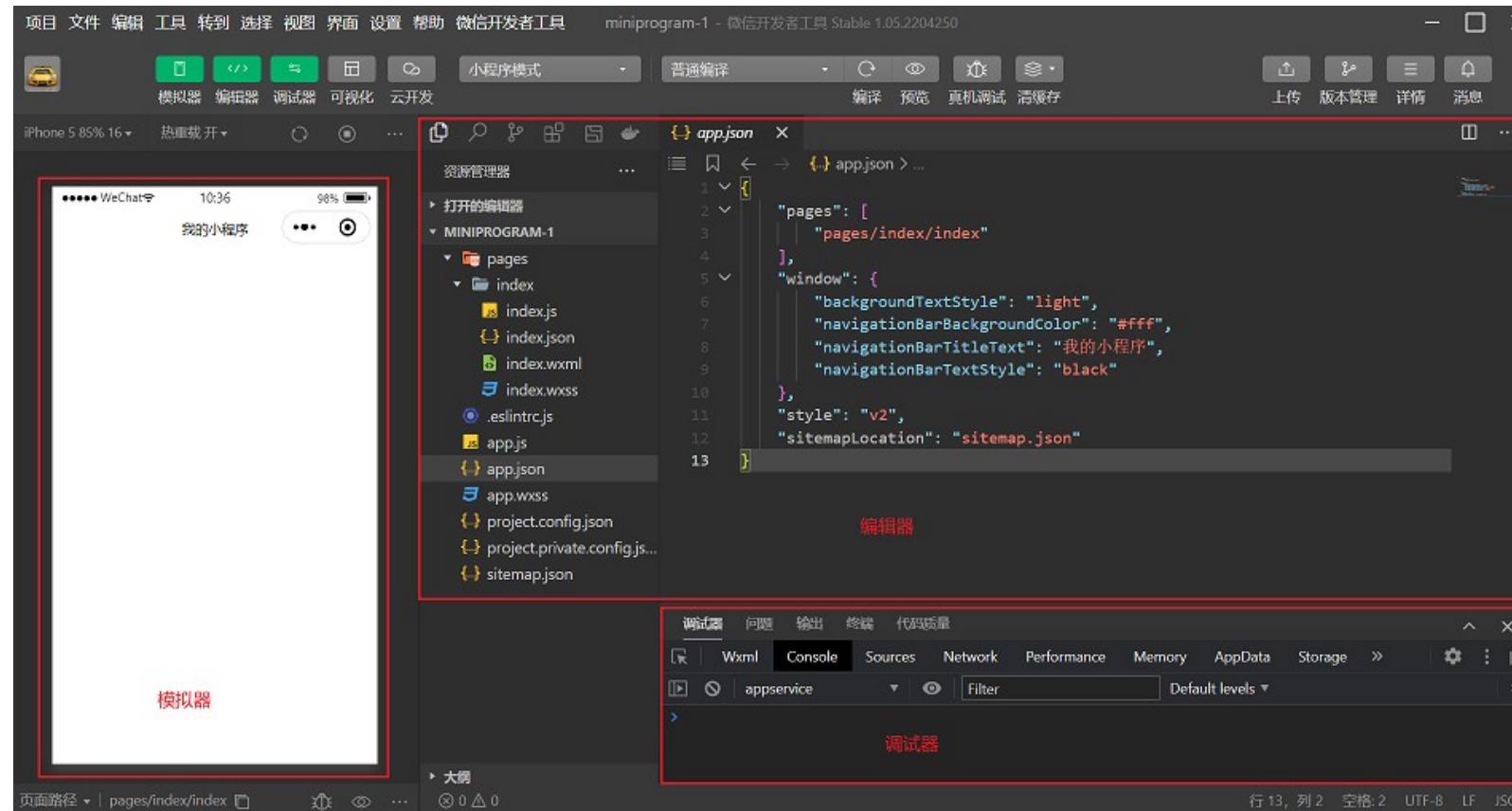
创建小程序项目





准备工作 – 下载开发者工具

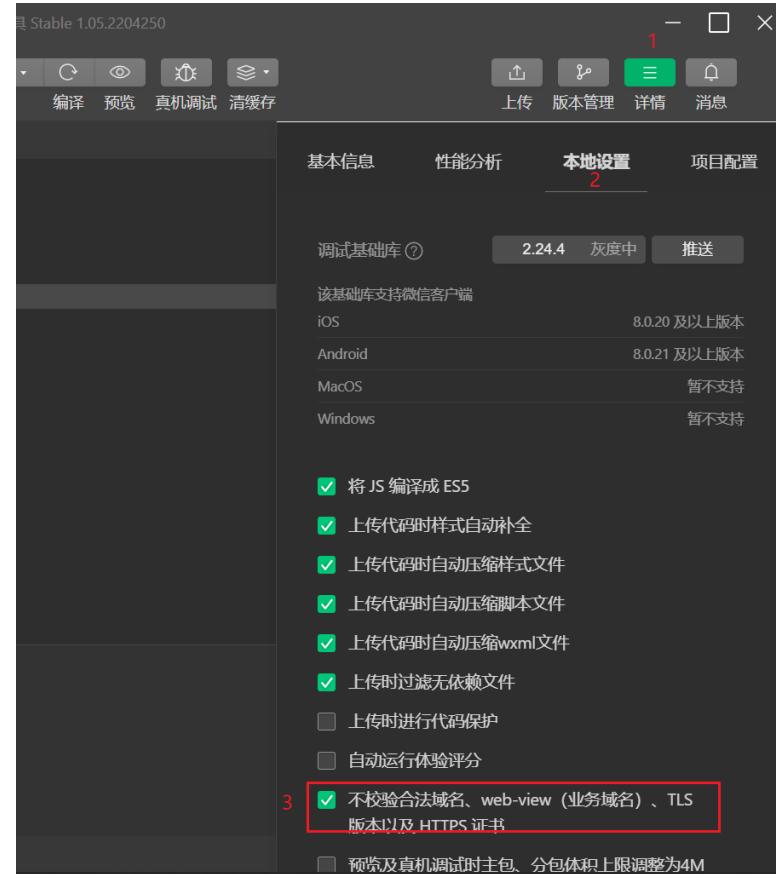
熟悉开发者工具布局

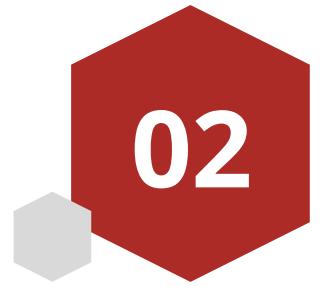




准备工作 – 下载开发者工具

设置不校验合法域名





微信小程序开发

- 介绍
- 准备工作
- 入门案例

入门案例

操作步骤：

- 了解小程序目录结构
- 编写小程序代码
- 编译小程序

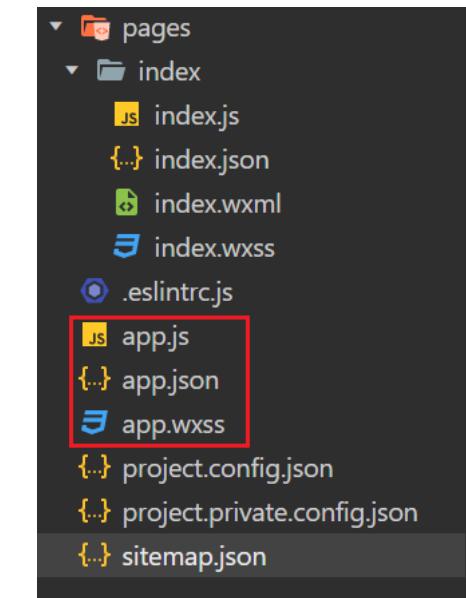


入门案例

- 了解小程序目录结构

小程序包含一个描述整体程序的 app 和多个描述各自页面的 page。
一个小程序主体部分由三个文件组成，必须放在项目的根目录，如下：

文件	必需	作用
app.js	是	小程序逻辑
app.json	是	小程序公共配置
app.wxss	否	小程序公共样式表



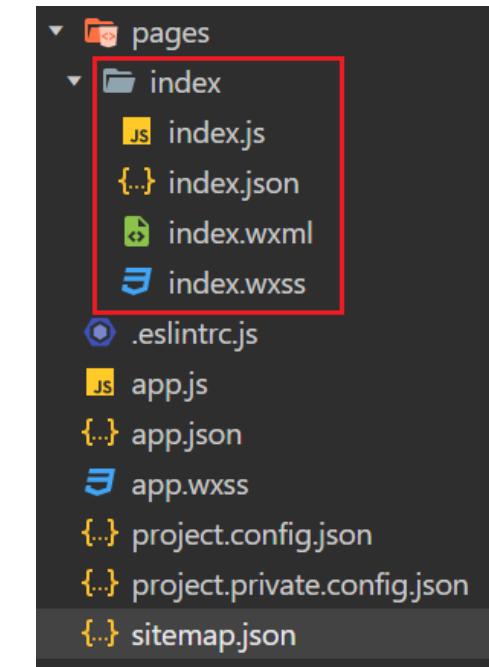


入门案例

- 了解小程序目录结构

一个小程序页面由四个文件组成：

文件类型	必需	作用
js	是	页面逻辑
wxml	是	页面结构
json	否	页面配置
wxss	否	页面样式表





入门案例

- 编写小程序代码

```
<view class="container">
  <view>{{msg}}</view>

  <view>
    <button type="default" bindtap="getUserInfo">获取用户信息 </button>
    <image style="width: 100px;height: 100px;" src="{{avatarUrl}}></image>
    {{nickName}}
  </view>

  <view>
    <button type="primary" bindtap="wxlogin">微信登录 </button>
    授权码：{{code}}
  </view>

  <view>
    <button type="warn" bindtap="sendRequest">发送请求 </button>
    响应结果：{{result}}
  </view>
</view>
```

index.wxml



入门案例

- 编写小程序代码

```
Page({
  data: {
    msg: 'hello world',
    avatarUrl: '',
    nickName: '',
    code: '',
    result: ''
  },
  getUserInfo: function() {
    wx.getUserProfile({
      desc: '获取用户信息',
      success: (res) => {
        console.log(res)
        this.setData({
          avatarUrl: res.userInfo.avatarUrl,
          nickName: res.userInfo.nickName
        })
      }
    })
  }
})
```

index.js



入门案例

- 编写小程序代码

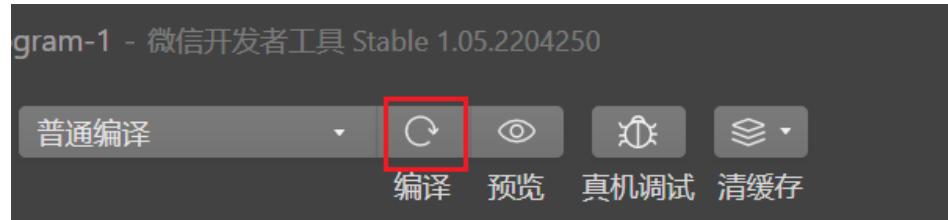
```
wxlogin:function(){
  wx.login({
    success: (res) => {
      console.log("授权码：" + res.code)
      this.setData({
        code:res.code
      })
    }
  }),
  sendRequest:function(){
    wx.request({
      url: 'http://localhost:8080/user/shop/status',
      method:'GET',
      success:(res) => {
        console.log("响应结果：" + res.data.data)
        this.setData({
          result:res.data.data
        })
      }
    })
  }
}
```

index.js



入门案例

- 编译小程序





目录

Contents

- ◆ HttpClient
- ◆ 微信小程序开发
- ◆ 微信登录
- ◆ 导入商品浏览功能代码



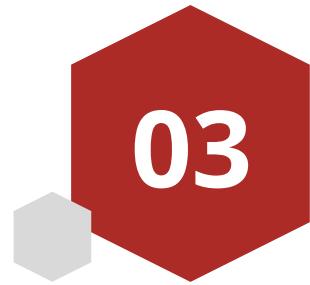
微信登录

- 导入小程序代码
- 微信登录流程
- 需求分析和设计
- 代码开发
- 功能测试



导入小程序代码

```
▼ MP-WEIXIN
  ▶ common
  ▶ components
  ▶ node-modules
  ▶ pages
  ▶ static
  ▶ uni_modules
  ◊ .gitignore
  JS app.js
  {…} app.json
  ⚡ app.wxss
  {…} project.config.json
  {…} project.private.config.json
  {…} sitemap.json
```



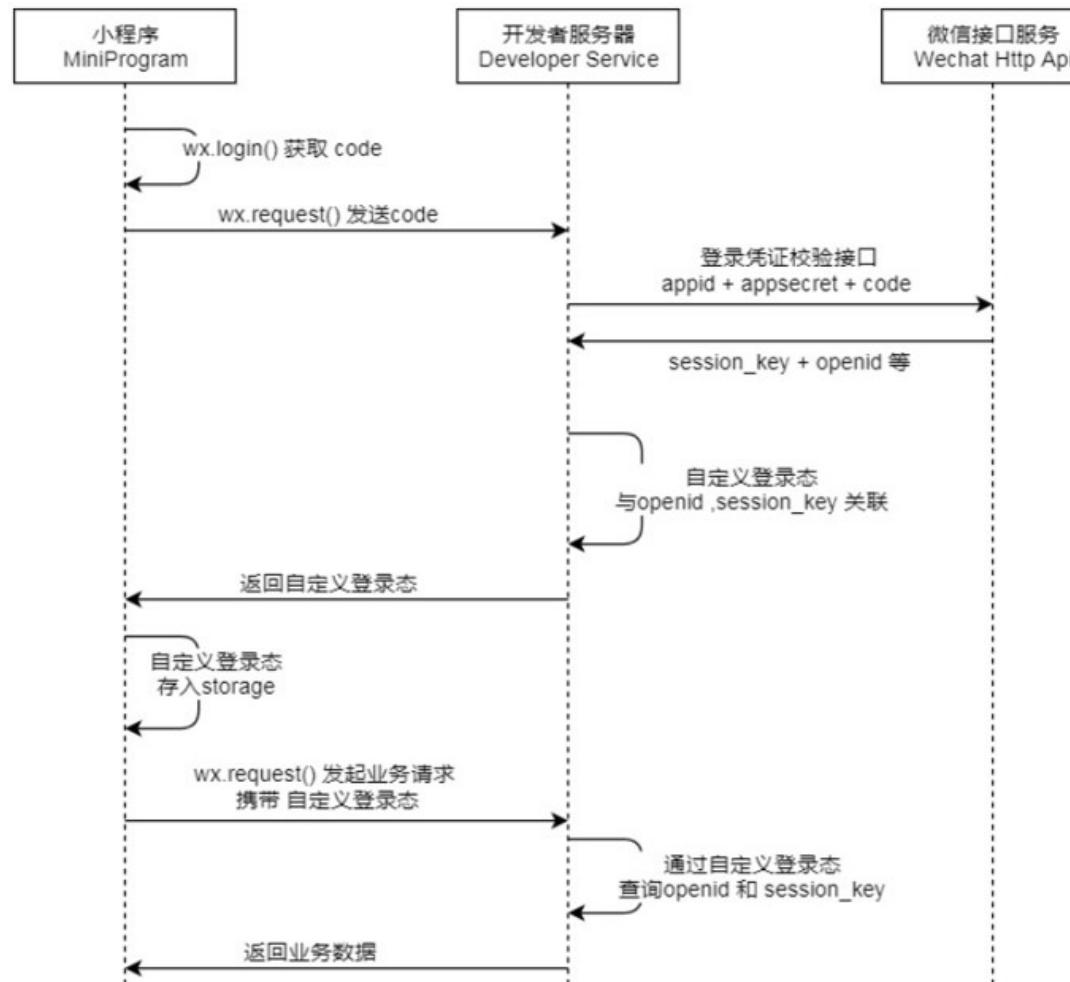
微信登录

- 导入小程序代码
- 微信登录流程
- 需求分析和设计
- 代码开发
- 功能测试



微信登录流程

微信登录：<https://developers.weixin.qq.com/miniprogram/dev/framework/open-ability/login.html>



注意：可以使用 postman 测试登录凭证校验接口



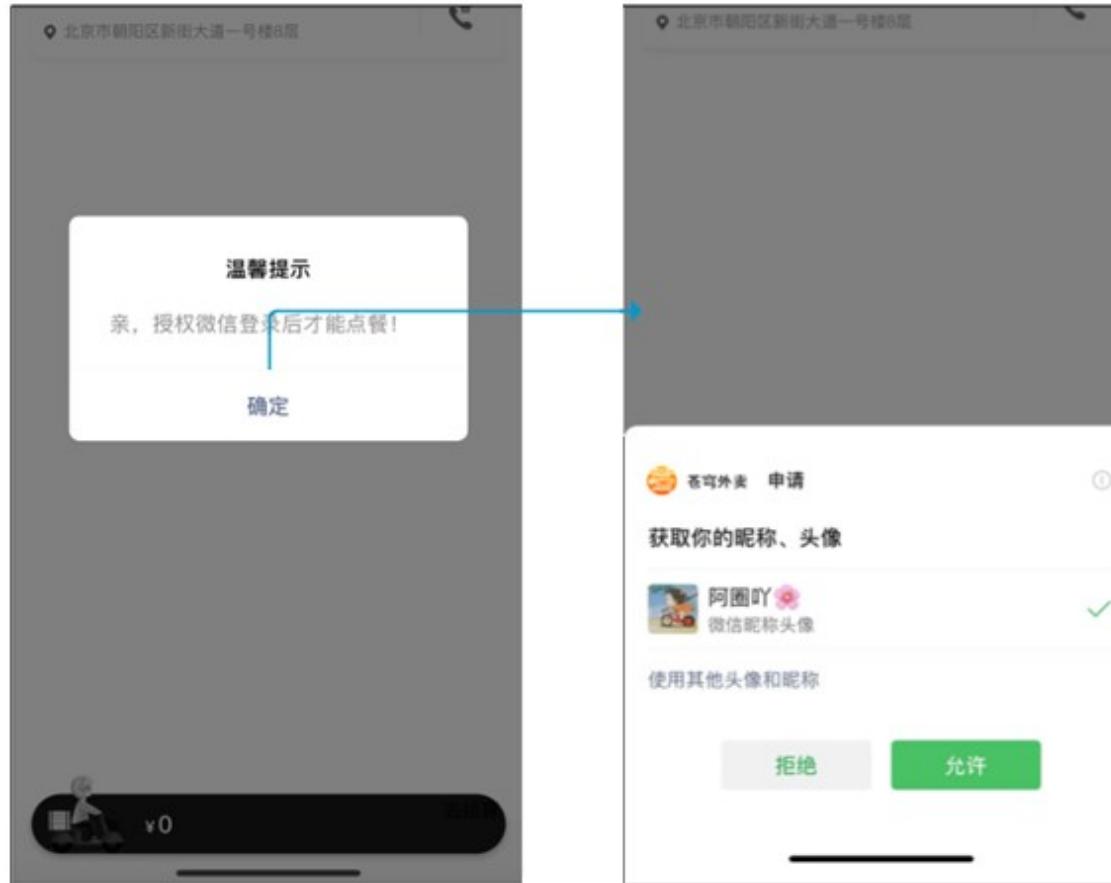
微信登录

- 导入小程序代码
- 微信登录流程
- 需求分析和设计
- 代码开发
- 功能测试



需求分析和设计

产品原型：



业务规则：

- 基于微信登录实现小程序的登录功能
- 如果是新用户需要自动完成注册



需求分析和设计

接口设计：

基本信息

Path: /user/user/login

Method: POST

接口描述：

请求参数

Headers

参数名称	参数值	是否必须	示例	备注
Content-Type	application/json	是		

Body

名称	类型	是否必须	默认值	备注	其他信息
code	string	必须		微信用户授权码	

返回数据

名称	类型	是否必须	默认值	备注	其他信息
code	integer	必须			format: int32
data	object	必须			
├─ id	integer	必须		用户id	format: int64
├─ openid	string	必须		微信openid	
├─ token	string	必须		jwt令牌	
msg	string	非必须			



需求分析和设计

数据库设计（ user 表）：

字段名	数据类型	说明	备注
id	bigint	主键	自增
openid	varchar(45)	微信用户的唯一标识	
name	varchar(32)	用户姓名	
phone	varchar(11)	手机号	
sex	varchar(2)	性别	
id_number	varchar(18)	身份证号	
avatar	varchar(500)	微信用户头像路径	
create_time	datetime	注册时间	



微信登录

- 导入小程序代码
- 微信登录流程
- 需求分析和设计
- 代码开发
- 功能测试



代码开发

配置微信登录所需配置项：

The screenshot shows two code editors side-by-side. The left editor is titled 'application-dev.yml' and the right one is 'application.yml'. Both files are in YAML format.

application-dev.yml:

```
sky:
  datasource: <6 keys>
  redis: <4 keys>
  alioss: <4 keys>
wechat:
  appid: w[REDACTED]36[REDACTED]22[REDACTED]3
  secret: 84[REDACTED]11df9[REDACTED]acdf[REDACTED]12d[REDACTED]b6[REDACTED]95[REDACTED]d
```

application.yml:

```
server: <1 key>
spring: <4 keys>
mybatis: <3 keys>
logging: <1 key>
sky:
  jwt: <6 keys>
  alioss: <4 keys>
wechat:
  appid: ${sky.wechat.appid}
  secret: ${sky.wechat.secret}
```

In both files, the 'wechat' configuration section is highlighted with a red box. In the 'application-dev.yml' file, the 'appid' and 'secret' values are also highlighted with a red box. In the 'application.yml' file, the 'appid' and 'secret' values are shown as environment variable placeholders: \${sky.wechat.appid} and \${sky.wechat.secret}, which are also highlighted with a red box.



代码开发

配置为微信用户生成 jwt 令牌时使用的配置项：

```
application.yml
1 server: <1 key>
3
4 spring: <4 keys>
20
21 mybatis: <3 keys>
28
29 logging: <1 key>
36
37 sky:
38   jwt:
39     # 设置jwt签名加密时使用的秘钥
40     admin-secret-key: itcast
41     # 设置jwt过期时间
42     admin-ttl: 7200000
43     # 设置前端传递过来的令牌名称
44     admin-token-name: token
45     # 设置jwt签名加密时使用的秘钥
46     user-secret-key: itheima
47     # 设置jwt过期时间
48     user-ttl: 7200000
49     # 设置前端传递过来的令牌名称
50     user-token-name: authentication
51
52   alioss: <4 keys>
53   wechat: <9 keys>
```

代码开发

DTO 设计：

Body

名称	类型	是否必须	默认值	备注	其他信息
code	string	必须		微信用户授权码	



```
/**  
 * C 端用户登录  
 */  
@Data  
public class UserLoginDTO implements Serializable {  
  
    // 微信用户授权码  
    private String code;  
  
}
```



代码开发

VO 设计：

名称	类型	是否必须	默认值	备注
code	integer	必须		
data	object	必须		
└ id	integer	必须		用户id
└ openid	string	必须		微信openid
└ token	string	必须		jwt令牌
msg	string	非必须		



```
@Data  
@Builder  
@NoArgsConstructor  
@AllArgsConstructor  
public class UserLoginVO implements Serializable {  
  
    private Long id;  
    private String openid;  
    private String token;  
}
```



代码开发

根据接口定义创建 UserController 的 login 方法：

```
@RestController
@RequestMapping("/user/user")
@Slf4j
@Api(tags = "C 端 - 用户接口")
public class UserController {

    @Autowired
    private UserService userService;
    @Autowired
    private JwtProperties jwtProperties;

    /**
     * C 端用户登录 -- 微信登录
     * @param userLoginDTO
     * @return
     */
    @PostMapping("/login")
    @ApiOperation("登录")
    public Result<UserLoginVO> login(@RequestBody UserLoginDTO userLoginDTO) {
        log.info("微信用户登录，授权码为：{}", userLoginDTO.getCode());
        return Result.success();
    }
}
```

UserController



代码开发

完善 UserController 的 login 方法：

```
/*
 * C 端用户登录 -- 微信登录
 * @param userLoginDTO
 * @return
 */
@PostMapping("/login")
@ApiOperation("登录")
public Result<UserLoginVO> login(@RequestBody UserLoginDTO userLoginDTO) {
    Log.info("微信用户登录，授权码为：{}", userLoginDTO.getCode());
    User user = userService.wxLogin(userLoginDTO);

    Map claims = new HashMap();
    claims.put(JwtClaimsConstant.USER_ID, user.getId());
    String token = JwtUtil.createJWT(jwtProperties.getUserSecretKey(), jwtProperties.getUserTtl(), claims);

    UserLoginVO userLoginVO = UserLoginVO.builder()
        .id(user.getId())
        .openid(user.getOpenid())
        .token(token)
        .build();

    return Result.success(userLoginVO);
}
```

UserController



代码开发

创建 UserService 接口：

```
public interface UserService {  
  
    /**  
     * 根据微信授权码实现微信登录  
     * @param userLoginDTO  
     * @return  
     */  
    User wxLogin(UserLoginDTO userLoginDTO);  
}
```



代码开发

创建 UserServiceImpl 实现类：

```
@Service
@Slf4j
public class UserServiceImpl implements UserService {

    public static final String WX_LOGIN = "https://api.weixin.qq.com/sns/jscode2session";

    @Autowired
    private UserMapper userMapper;

    @Autowired
    private WeChatProperties weChatProperties;

    /**
     * 根据微信授权码实现微信登录
     * @param userLoginDTO
     * @return
     */
    public User wxLogin(UserLoginDTO userLoginDTO) {
        return null;
    }
}
```



代码开发

在 UserServiceImpl 中创建私有方法 getOpenid：

```
/*
 * 获取微信用户的openid
 * @param code
 * @return
 */
private String getOpenid(String code){
    // 请求参数封装
    Map map = new HashMap();
    map.put("appid",weChatProperties.getAppid());
    map.put("secret",weChatProperties.getSecret());
    map.put("js_code",code);
    map.put("grant_type","authorization_code");

    // 调用工具类，向微信接口服务发送请求
    String json = HttpClientUtil.doGet(WX_LOGIN, map);
    Log.info("微信登录返回结果：{}", json);

    // 解析json字符串
    JSONObject jsonObject = JSON.parseObject(json);
    String openid = jsonObject.getString("openid");
    Log.info("微信用户的openid为：{}", openid);

    return openid;
}
```

UserServiceImpl



代码开发

完善 UserServiceImpl 的 wxLogin 方法：

```
public User wxLogin(UserLoginDTO userLoginDTO) {
    // 授权码
    String code = userLoginDTO.getCode();
    String openid = getOpenid(code);

    if(openid == null){
        throw new LoginFailedException(MessageConstant.LOGIN_FAILED);
    }

    // 根据openid 查询用户信息
    User user = userMapper.getByOpenid(openid);

    if(user == null){
        // 当前是一个新用户，自动完成注册
        user = new User();
        user.setOpenid(openid);
        user.setCreateTime(LocalDateTime.now());
        userMapper.insert(user);
    }
    return user;
}
```

UserServiceImpl



代码开发

创建 UserMapper 接口：

```
@Mapper
public interface UserMapper {
    /**
     * 插入
     * @param user
     */
    void insert(User user);

    /**
     * 根据openid 查询用户
     * @return
     */
    @Select("select * from user where openid = #{openid}")
    User getByOpenid(String openid);
}
```

UserMapper



代码开发

创建 UserMapper.xml 映射文件：

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://mybatis.org/dtd/mybatis-3-mapper.dtd" >
<mapper namespace="com.sky.mapper.UserMapper">

    <!-- 插入 -->
    <!--
        useGeneratedKeys 设置为 true : 在执行插入记录之后可以获取到数据库自动生成的主键值
        keyProperty : 指定 Java 对象的属性名
    -->
    <insert id="insert" parameterType="User" useGeneratedKeys="true" keyProperty="id">
        insert into user
            (openid,name,phone,sex,id_number,avatar,create_time)
        values
            (#{{openid}},#{{name}},#{{phone}},#{{sex}},#{{idNumber}},#{{avatar}},#{{createTime}})
    </insert>

</mapper>
```

UserMapper.xml



代码开发

编写拦截器 JwtTokenUserInterceptor，统一拦截用户端发送的请求并进行 jwt 校验：

```
//1、从请求头中获取令牌
String token = request.getHeader(jwtProperties.getUserTokenName());

//2、校验令牌
try{
    Claims claims = JwtUtil.parseJWT(jwtProperties.getUserSecretKey(), token);
    Long userId = Long.valueOf(claims.get(DwtClaimsConstant.USER_ID).toString());
    BaseContext.setCurrentId(userId);
    //3、通过，放行
    return true;
} catch (Exception ex){
    //4、不通过，响应401状态码
    response.setStatus(401);
    return false;
}
```



代码开发

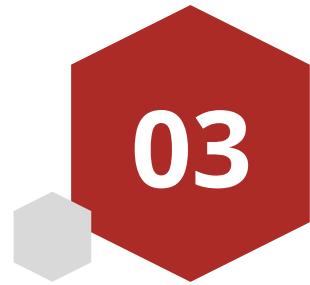
在 WebMvcConfiguration 配置类中注册拦截器：

```
@Configuration
@Slf4j
public class WebMvcConfiguration extends WebMvcConfigurationSupport {

    @Autowired
    private JwtTokenAdminInterceptor jwtTokenAdminInterceptor;
    @Autowired
    private JwtTokenUserInterceptor jwtTokenUserInterceptor;

    /**
     * 注册自定义拦截器
     * @param registry
     */
    protected void addInterceptors(InterceptorRegistry registry) {
        Log.info("开始注册自定义拦截器...");
        registry.addInterceptor(jwtTokenAdminInterceptor)
            .addPathPatterns("/admin/**")
            .excludePathPatterns("/admin/employee/login");

        registry.addInterceptor(jwtTokenUserInterceptor)
            .addPathPatterns("/user/**")
            .excludePathPatterns("/user/user/login")
            .excludePathPatterns("/user/shop/status");
    }
}
```



微信登录

- 导入小程序代码
- 微信登录流程
- 需求分析和设计
- 代码开发
- 功能测试

功能测试

可以通过接口文档进行测试，最后完成前后端联调测试即可



目录

Contents

- ◆ HttpClient
- ◆ 微信小程序开发
- ◆ 微信登录
- ◆ 导入商品浏览功能代码

A red hexagonal icon containing the white text "04". To its left is a smaller, semi-transparent light gray hexagon.

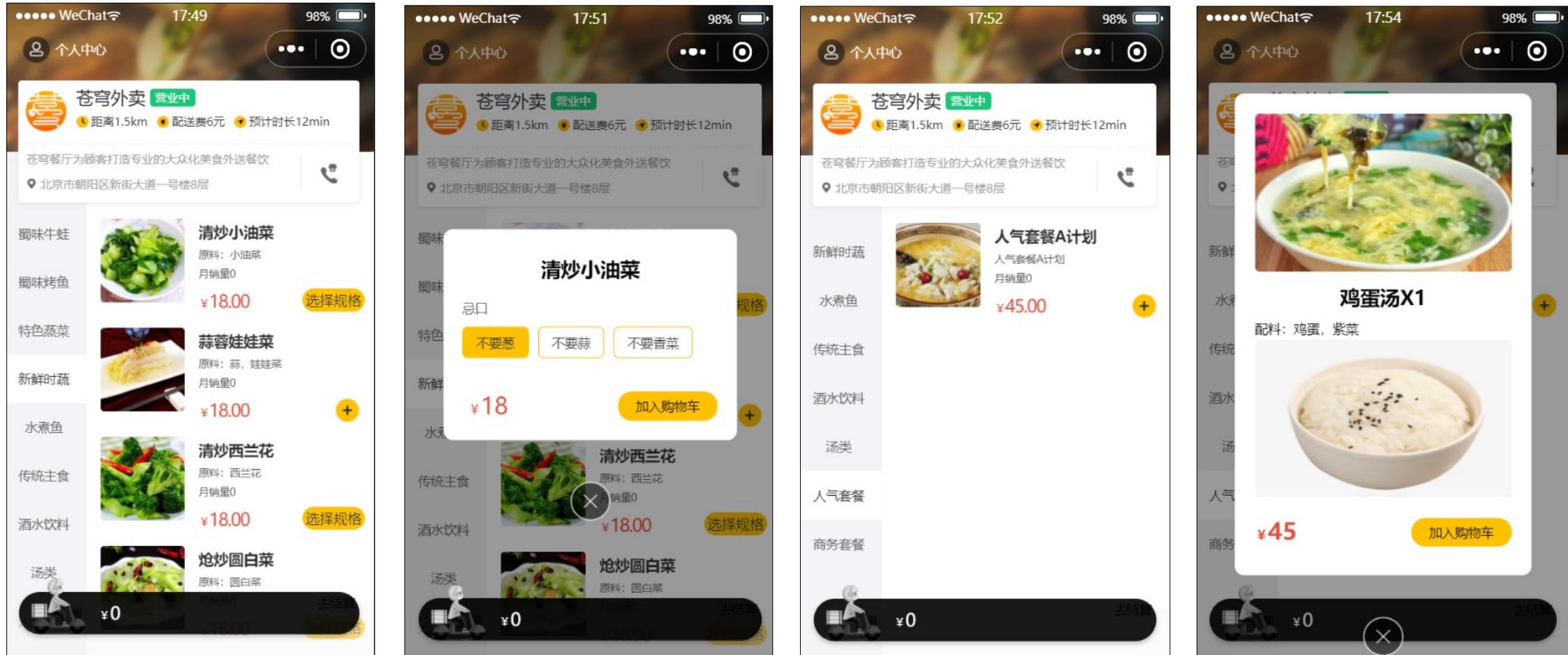
导入商品浏览功能代码

- 需求分析和设计
- 代码导入
- 功能测试



需求分析和设计

产品原型：



需求分析和设计

接口设计：

- 查询分类
- 根据分类 id 查询菜品
- 根据分类 id 查询套餐
- 根据套餐 id 查询包含的菜品



需求分析和设计

- 查询分类 接口

基本信息

Path: /user/category/list

Method: GET

接口描述：

请求参数

Query

参数名称	是否必须	示例	备注
type	否	1	分类类型：1 菜品分类 2 套餐分类

返回数据

名称	类型	是否必须	默认值	备注	其他信息
code	integer	必须			format: int32
data	object []	必须			item 类型: object
└ createTime	string	非必须			format: date-time
└ createUser	integer	非必须			format: int64
└ id	integer	必须			format: int64
└ name	string	必须			
└ sort	integer	非必须			format: int32
└ status	integer	非必须			format: int32
└ type	integer	非必须			format: int32
└ updateTime	string	非必须			format: date-time
└ updateUser	integer	非必须			format: int64
msg	string	非必须			



需求分析和设计

- 根据分类 id 查询菜品 接口

基本信息

Path: /user/dish/list

Method: GET

接口描述:

请求参数

Query

参数名称	是否必须	示例	备注
categoryId	是		分类id

返回数据

名称	类型	是否必须	默认值	备注	其他信息
code	integer	必须			format: int32
data	object []	必须			item 类型: object
└─ categoryId	integer	必须			format: int64
└─ categoryName	string	非必须			
└─ description	string	非必须			
└─ flavors	object []	非必须			item 类型: object
└─ dishId	integer	非必须			format: int64
└─ id	integer	非必须			format: int64
└─ name	string	非必须			
└─ value	string	非必须			
└─ id	integer	必须			format: int64
└─ image	string	必须			
└─ name	string	必须			
└─ price	number	必须			
└─ status	integer	非必须			format: int32
└─ updateTime	string	非必须			format: date-time
msg	string	非必须			



需求分析和设计

- 根据分类 id 查询套餐 接口

基本信息

Path: /user/setmeal/list

Method: GET

接口描述:

请求参数

Query

参数名称	是否必须	示例	备注
categoryId	是		分类id

返回数据

名称	类型	是否必须	默认值	备注	其他信息
code	integer	必须			format: int32
data	object []	必须			item 类型: object
└ categoryId	integer	必须			format: int64
└ createTime	string	非必须			format: date-time
└ createUser	integer	非必须			format: int64
└ description	string	非必须			
└ id	integer	必须			format: int64
└ image	string	必须			
└ name	string	必须			
└ price	number	必须			
└ status	integer	非必须			format: int32
└ updateTime	string	非必须			format: date-time
└ updateUser	integer	非必须			format: int64
msg	string	非必须			



需求分析和设计

- 根据套餐 id 查询包含的菜品 接口

基本信息

Path: /user/setmeal/dish/{id}

Method: GET

接口描述:

请求参数

路径参数

参数名称	示例	备注
id		套餐id

返回数据

名称	类型	是否必须	默认值	备注	其他信息
code	number	必须			
data	object []	非必须			item 类型: object
└ copies	number	必须		份数	
└ description	string	必须		菜品描述	
└ image	string	必须		菜品图片	
└ name	string	必须		菜品名称	
msg	string	非必须			

A red hexagonal icon containing the white text "04". To its left is a smaller, semi-transparent light gray hexagon.

导入商品浏览功能代码

- 需求分析和设计
- 代码导入
- 功能测试



代码导入

› 授课资料 › day06-微信登录、商品浏览 › 资料 › 代码导入 › 商品浏览

名称

- CategoryController.java
- DishController.java
- DishService.java
- DishServiceImpl.java
- SetmealController.java
- SetmealMapper.java
- SetmealMapper.xml
- SetmealService.java
- SetmealServiceImpl.java

A red hexagonal icon containing the white text "04". To its left is a smaller, semi-transparent light gray hexagon.

导入商品浏览功能代码

- 需求分析和设计
- 代码导入
- 功能测试

功能测试

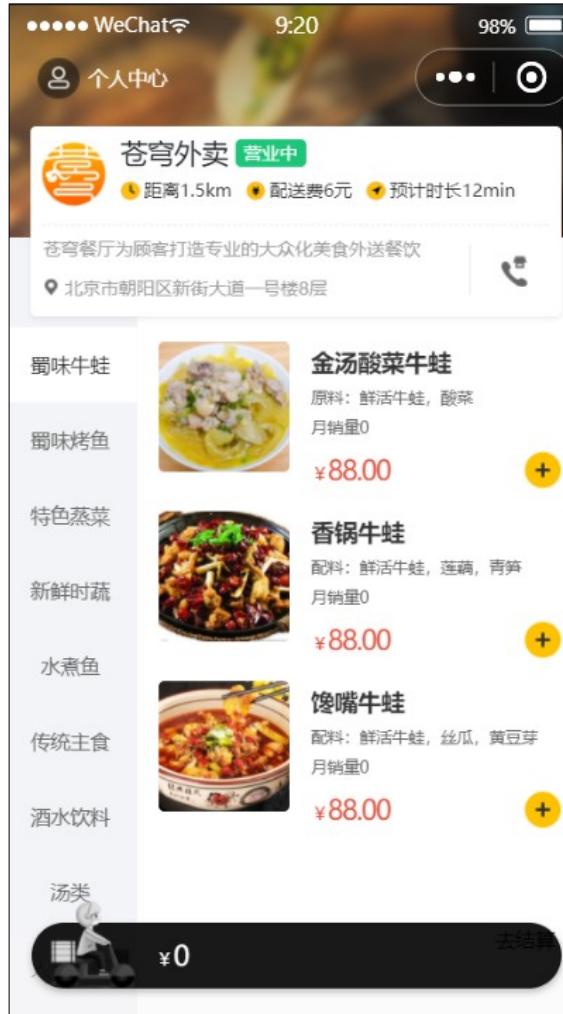
通过 Swagger 接口文档进行测试，通过后再前端联调测试即可



传智教育旗下高端IT教育品牌



缓存商品、购物车





目录

Contents

- ◆ 缓存菜品
- ◆ 缓存套餐
- ◆ 添加购物车
- ◆ 查看购物车
- ◆ 清空购物车



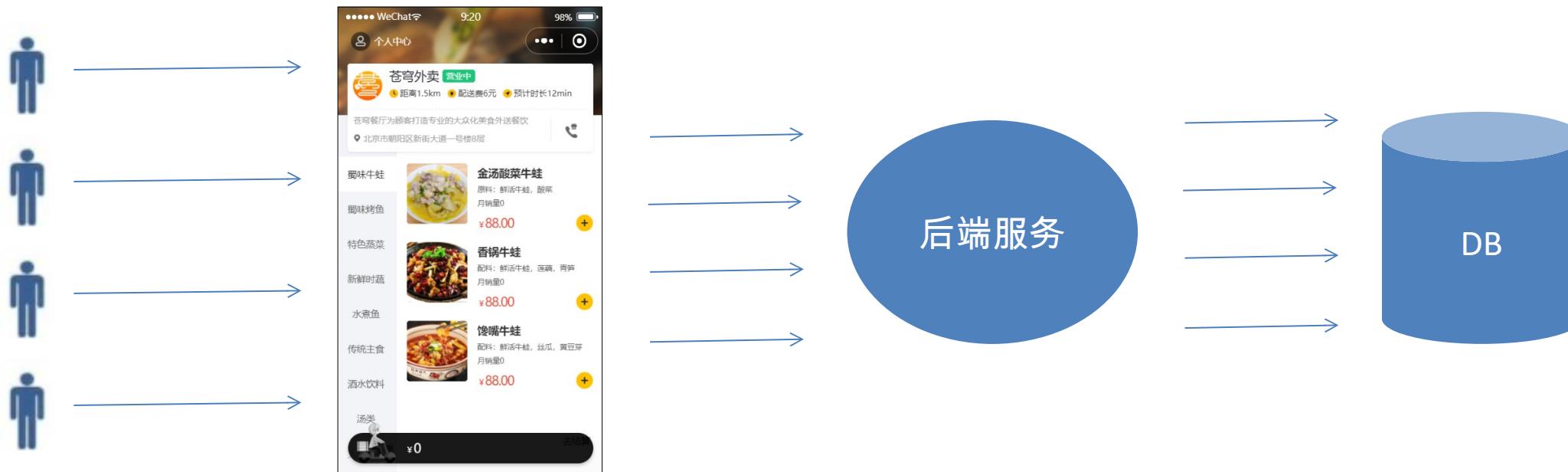
缓存菜品

- 问题说明
- 实现思路
- 代码开发
- 功能测试



问题说明

用户端小程序展示的菜品数据都是通过查询数据库获得，如果用户端访问量比较大，数据库访问压力随之增大。



结果：系统响应慢、用户体验差

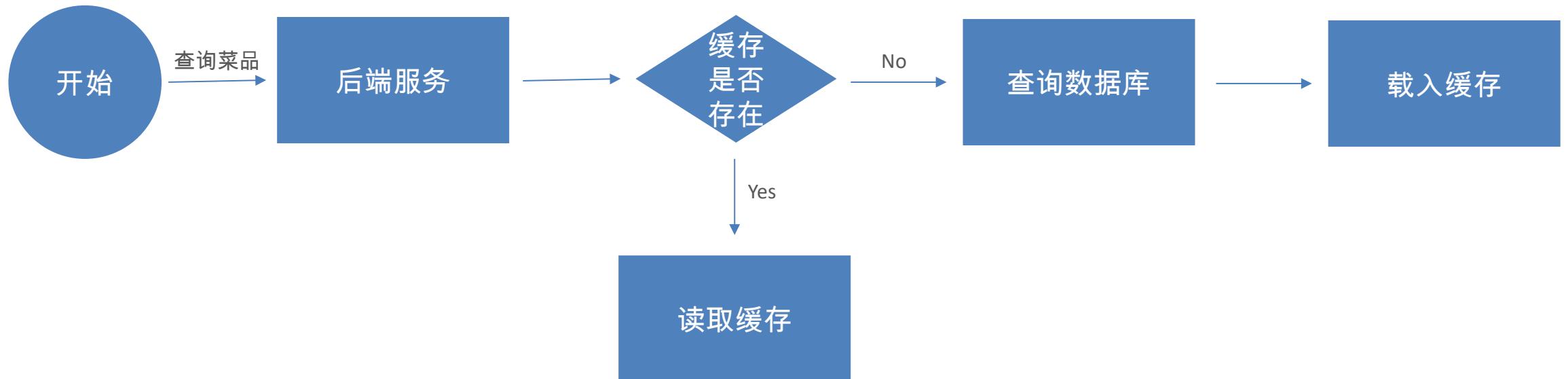


缓存菜品

- 问题说明
- 实现思路
- 代码开发
- 功能测试

实现思路

通过 Redis 来缓存菜品数据，减少数据库查询操作。

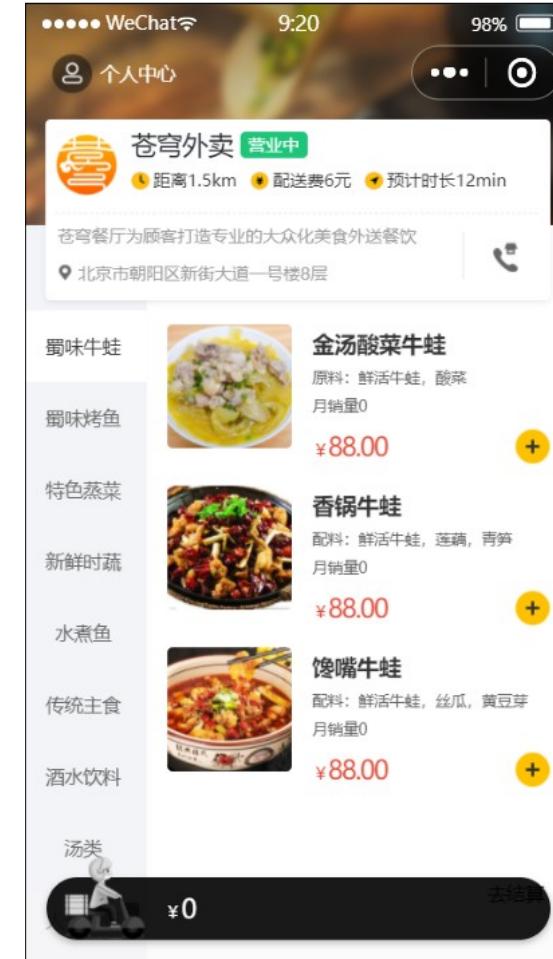


实现思路

缓存逻辑分析：

- 每个分类下的菜品保存一份缓存数据
- 数据库中菜品数据有变更时清理缓存数据

key	value
dish_1	string(...)
dish_2	string(...)
dish_3	string(...)





缓存菜品

- 问题说明
- 实现思路
- 代码开发
- 功能测试



代码开发

修改用户端接口 DishController 的 list 方法，加入缓存处理逻辑：

```
// 构造redis 缓存key，规则为：dish_ 分类id
String key = "dish_" + categoryId;

// 查询redis 中是否有缓存数据
List<DishVO> list = (List<DishVO>) redisTemplate.opsForValue().get(key);

// 存在缓存数据，直接返回给前端
if(list != null && list.size() > 0){
    return Result.success(list);
}

Dish dish = new Dish();
dish.setCategoryId(categoryId);
dish.setStatus(StatusConstant.ENABLE); // 查询起售中的菜品

list = dishService.listWithFlavor(dish);

// 将查询到的数据载入缓存
redisTemplate.opsForValue().set(key, list);

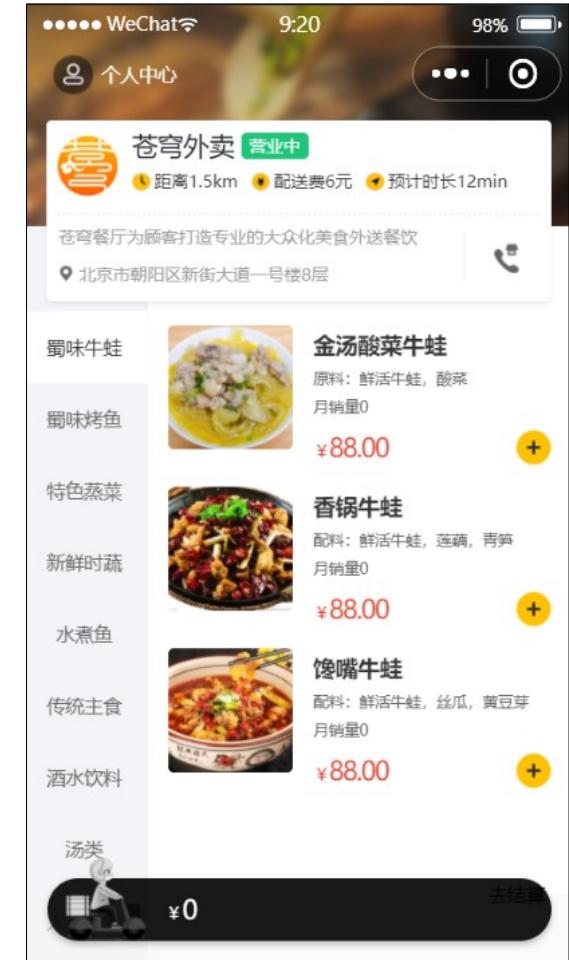
return Result.success(list);
```

DishController

代码开发

修改管理端接口 DishController 的相关方法，加入清理缓存的逻辑，需要改造的方法：

- 新增菜品
- 修改菜品
- 批量删除菜品
- 起售、停售菜品



代码开发

抽取清理缓存的方法：

```
/**  
 * 清理缓存数据  
 */  
private void cleanCache(String pattern){  
    Set keys = redisTemplate.keys(pattern);  
    redisTemplate.delete(keys);  
}
```

DishController



代码开发

调用清理缓存的方法，保证数据一致性：

```
@Autowired
private RedisTemplate redisTemplate;

/**
 * 新增菜品
 * @param dishDTO
 * @return
 */
@PostMapping
@ApiOperation("新增菜品")
public Result save(@RequestBody DishDTO dishDTO){
    log.info("新增菜品：{}", dishDTO);
    dishService.saveWithFlavor(dishDTO);

    Long categoryId = dishDTO.getCategoryId();
    String key = "dish_" + categoryId;
    cleanCache(key);

    return Result.success();
}
```

DishController

代码开发

调用清理缓存的方法，保证数据一致性：

```
/**  
 * 菜品批量删除  
 * @param ids  
 * @return  
 */  
  
@DeleteMapping  
@ApiOperation("菜品批量删除")  
public Result delete(@RequestParam List<Long> ids){  
    Log.info("菜品批量删除：{}", ids);  
    dishService.deleteBatch(ids);  
  
    // 删除所有菜品的缓存数据  
    cleanCache("dish_*");  
  
    return Result.success();  
}
```

DishController



代码开发

调用清理缓存的方法，保证数据一致性：

```
/*
 * 修改菜品
 * @param dishDTO
 * @return
 */
@PostMapping
@ApiOperation("修改菜品")
public Result<String> update(@RequestBody DishDTO dishDTO){
    log.info("修改菜品：{}", dishDTO);
    dishService.updateWithFlavor(dishDTO);

    // 删除所有菜品的缓存数据
    cleanCache("dish_*");

    return Result.success();
}
```

DishController



代码开发

调用清理缓存的方法，保证数据一致性：

```
/**  
 * 菜品起售停售  
 * @param status  
 * @param id  
 * @return  
 */  
@PostMapping("/status/{status}")  
@ApiOperation("菜品起售停售")  
public Result<String> startOrStop(@PathVariable Integer status, Long id){  
    dishService.startOrStop(status,id);  
  
    // 删除所有菜品的缓存数据  
    cleanCache("dish_*");  
  
    return Result.success();  
}
```

DishController



缓存菜品

- 问题说明
- 实现思路
- 代码开发
- 功能测试

功能测试

可以通过如下方式进行测试：

- 查看控制台 sql
- 前后端联调
- 查看 Redis 中的缓存数据



目录

Contents

- ◆ 缓存菜品
- ◆ 缓存套餐
- ◆ 添加购物车
- ◆ 查看购物车
- ◆ 清空购物车



缓存套餐

- Spring Cache
- 实现思路
- 代码开发
- 功能测试

Spring Cache

Spring Cache 是一个框架，实现了基于注解的缓存功能，只需要简单地加一个注解，就能实现缓存功能。

Spring Cache 提供了一层抽象，底层可以切换不同的缓存实现，例如：

- EHCache
- Caffeine
- Redis

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-cache</artifactId>
    <version>2.7.3</version>
</dependency>
```



Spring Cache

常用注解：

注解	说明
@EnableCaching	开启缓存注解功能，通常加在启动类上
@Cacheable	在方法执行前先查询缓存中是否有数据，如果有数据，则直接返回缓存数据；如果没有缓存数据，调用方法并将方法返回值放到缓存中
@CachePut	将方法的返回值放到缓存中
@CacheEvict	将一条或多条数据从缓存中删除



Spring Cache

入门案例：导入资料中的初始工程，在此基础上加入 Spring Cache 注解即可

```
springcache-demo E:\code\springcache-demo
> .idea
< src
  < main
    < java
      < com.itheima
        < config
          WebMvcConfiguration
        < controller
          UserController
        < entity
          User
        < mapper
          UserMapper
        CacheDemoApplication
      < resources
        application.yml
    > test
  pom.xml
  springcachedemo.sql
```



缓存套餐

- Spring Cache
- 实现思路
- 代码开发
- 功能测试

实现思路

具体的实现思路如下：

- 导入 Spring Cache 和 Redis 相关 maven 坐标
- 在启动类上加入 @EnableCaching 注解，开启缓存注解功能
- 在用户端接口 SetmealController 的 `list` 方法上加入 @Cacheable 注解
- 在管理端接口 SetmealController 的 `save`、`delete`、`update`、`startOrStop` 等方法上加入 CacheEvict 注解



缓存套餐

- Spring Cache
- 实现思路
- 代码开发
- 功能测试



代码开发

在用户端接口 SetmealController 的 `list` 方法上加入 `@Cacheable` 注解：

```
/*
 * 条件查询
 *
 * @param categoryId
 * @return
 */
@GetMapping("/list")
@ApiOperation("根据分类 id 查询套餐")
@Cacheable(cacheNames = "setmealCache",key = "#categoryId")
public Result<List<Setmeal>> list(Long categoryId) {
    Setmeal setmeal = new Setmeal();
    setmeal.setCategoryId(categoryId);
    setmeal.setStatus(StatusConstant.ENABLE);

    List<Setmeal> list = setmealService.list(setmeal);
    return Result.success(list);
}
```



代码开发

在管理端接口 SetmealController 的 `save`、`delete`、`update`、`startOrStop` 等方法上加入 `CacheEvict` 注解：

```
@PostMapping
@ApiOperation("新增套餐")
@CacheEvict(cacheNames = "setmealCache",key = "#setmealDTO.categoryId")
public Result save(@RequestBody SetmealDTO setmealDTO) {
    setmealService.saveWithDish(setmealDTO);
    return Result.success();
}

@DeleteMapping
@ApiOperation("批量删除套餐")
@CacheEvict(cacheNames = "setmealCache",allEntries = true)
public Result delete(@RequestParam List<Long> ids){
    setmealService.deleteBatch(ids);
    return Result.success();
}
```



缓存套餐

- Spring Cache
- 实现思路
- 代码开发
- 功能测试

功能测试

通过前后端联调方式来进行测试，同时观察 redis 中缓存的套餐数据。



目录

Contents

- ◆ 缓存菜品
- ◆ 缓存套餐
- ◆ 添加购物车
- ◆ 查看购物车
- ◆ 清空购物车



添加购物车

- 需求分析和设计
- 代码开发
- 功能测试



需求分析和设计

产品原型：

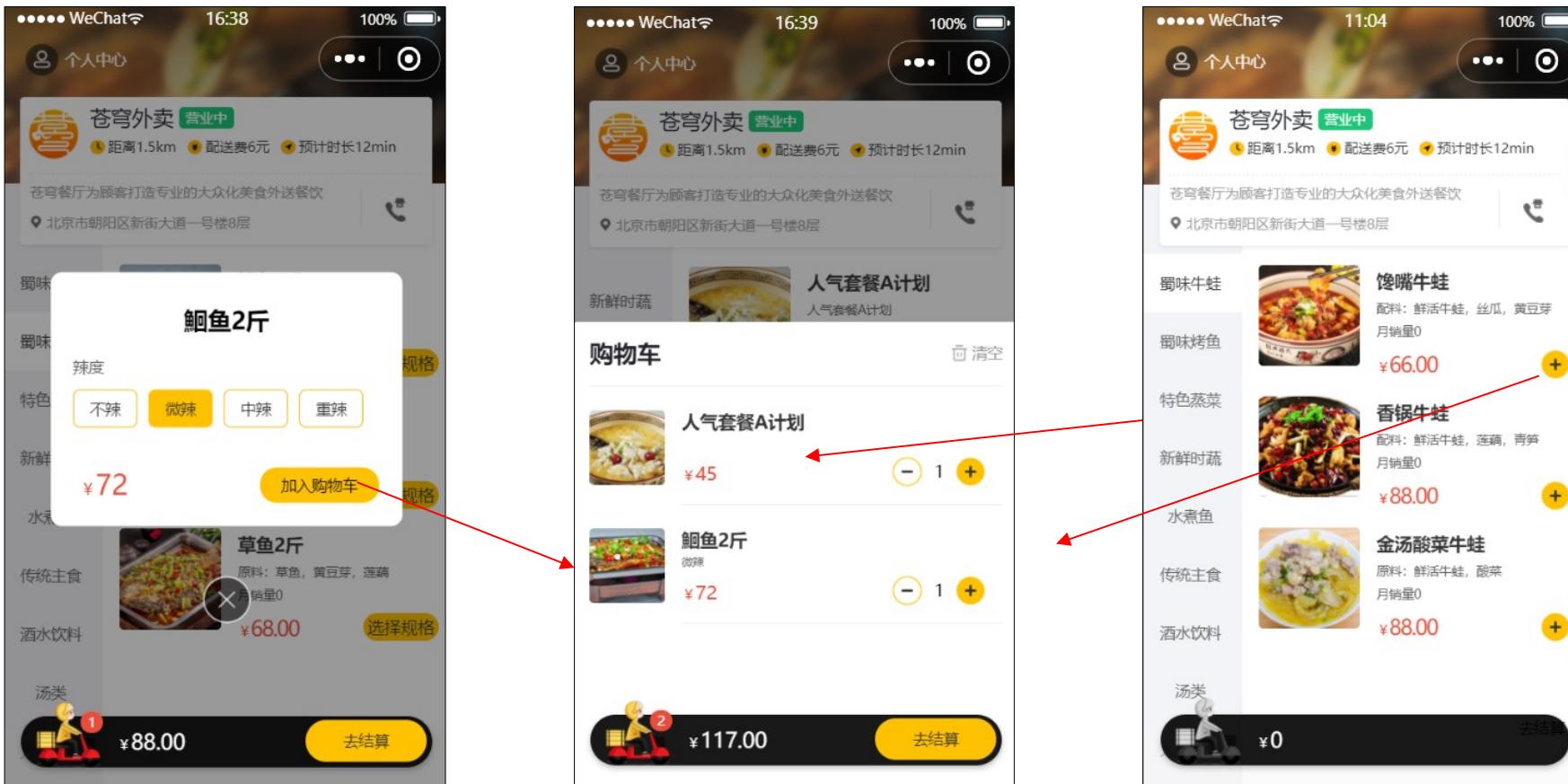


生活中的购物车：用于暂时存放所选商品的一种手推车



需求分析和设计

产品原型：



需求分析和设计

接口设计：

- 请求方式：POST
- 请求路径：/user/shoppingCart/add
- 请求参数：套餐 id、菜品 id、口味
- 返回结果：code、data、msg



需求分析和设计

接口设计：

基本信息

Path: /user/shoppingCart/add

Method: POST

接口描述：

请求参数

Headers

参数名称	参数值	是否必须	示例	备注
Content-Type	application/json	是		

Body

名称	类型	是否必须	默认值	备注	其他信息
dishFlavor	string	非必须		口味	
dishId	integer	非必须		菜品id	format: int64
setmealId	integer	非必须		套餐id	format: int64

返回数据

名称	类型	是否必须	默认值	备注	其他信息
code	integer	必须			format: int32
data	string	非必须			
msg	string	非必须			

需求分析和设计

数据库设计：

- 作用：暂时存放所选商品的地方
- 选的什么商品
- 每个商品都买了几个
- 不同用户的购物车需要区分开



需求分析和设计

数据库设计（shopping_cart 表）：

字段名	数据类型	说明	备注
id	bigint	主键	自增
name	varchar(32)	商品名称	冗余字段
image	varchar(255)	商品图片路径	冗余字段
user_id	bigint	用户 id	逻辑外键
dish_id	bigint	菜品 id	逻辑外键
setmeal_id	bigint	套餐 id	逻辑外键
dish_flavor	varchar(50)	菜品口味	
number	int	商品数量	
amount	decimal(10,2)	商品单价	冗余字段
create_time	datetime	创建时间	





添加购物车

- 需求分析和设计
- 代码开发
- 功能测试



代码开发

根据添加购物车接口的参数设计 DTO：

Body

名称	类型	是否必须	默认值	备注	其他信息
dishFlavor	string	非必须		口味	
dishId	integer	非必须		菜品id	format: int64
setmealId	integer	非必须		套餐id	format: int64



```
public class ShoppingCartDTO implements Serializable {  
  
    private Long dishId;  
    private Long setmealId;  
    private String dishFlavor;  
  
}
```



代码开发

根据添加购物车接口创建 ShoppingCartController：

```
@RestController
@RequestMapping("/user/shoppingCart")
@Slf4j
@Api(tags = "C 端 - 购物车接口")
public class ShoppingCartController {

    @Autowired
    private ShoppingCartService shoppingCartService;

    /**
     * 添加购物车
     * @param shoppingCartDTO
     * @return
     */
    @PostMapping("/add")
    @ApiOperation("添加购物车")
    public Result add(@RequestBody ShoppingCartDTO shoppingCartDTO){
        log.info("添加购物车： {}", shoppingCartDTO);
        shoppingCartService.addShoppingCart(shoppingCartDTO);
        return Result.success();
    }
}
```



代码开发

创建 ShoppingCartService 接口：

```
public interface ShoppingCartService {  
  
    /**  
     * 添加购物车  
     * @param shoppingCartDTO  
     */  
    void addShoppingCart(ShoppingCartDTO shoppingCartDTO);  
}
```



代码开发

创建 ShoppingCartServiceImpl 实现类，并实现 add 方法

```
@Service
public class ShoppingCartServiceImpl implements ShoppingCartService {
    @Autowired
    private ShoppingCartMapper shoppingCartMapper;
    @Autowired
    private DishMapper dishMapper;
    @Autowired
    private SetmealMapper setmealMapper;

    /**
     * 添加购物车
     * @param shoppingCartDTO
     */
    public void addShoppingCart(ShoppingCartDTO shoppingCartDTO) {
        ShoppingCart shoppingCart = new ShoppingCart();
        BeanUtils.copyProperties(shoppingCartDTO, shoppingCart);
        // 只能查询自己的购物车数据
        shoppingCart.setUserId(BaseContext.getCurrentId());

        // 判断当前商品是否在购物车中
        List<ShoppingCart> shoppingCartList = shoppingCartMapper.list(shoppingCart);
```



代码开发

创建 ShoppingCartServiceImpl 实现类，并实现 add 方法

```
if (shoppingCartList != null && shoppingCartList.size() == 1) {
    // 如果已经存在，就更新数量，数量加1
    shoppingCart = shoppingCartList.get(0);
    shoppingCart.setNumber(shoppingCart.getNumber() + 1);
    shoppingCartMapper.updateNumberById(shoppingCart);
} else {
    // 如果不存在，插入数据，数量就是1
    Long dishId = shoppingCartDTO.getDishId();
    if (dishId != null) {
        // 添加到购物车的是菜品
        Dish dish = dishMapper.getById(dishId);
        shoppingCart.setName(dish.getName());
        shoppingCart.setImage(dish.getImage());
        shoppingCart.setAmount(dish.getPrice());
    } else {
        // 添加到购物车的是套餐
        Setmeal setmeal = setmealMapper.getById(shoppingCartDTO.getSetmealId());
        shoppingCart.setName(setmeal.getName());
        shoppingCart.setImage(setmeal.getImage());
        shoppingCart.setAmount(setmeal.getPrice());
    }
    shoppingCart.setNumber(1);
    shoppingCart.setCreateTime(LocalDateTime.now());
    shoppingCartMapper.insert(shoppingCart);
}
```



代码开发

创建 ShoppingCartMapper 接口：

```
@Mapper
public interface ShoppingCartMapper {
    /**
     * 条件查询
     * @param shoppingCart
     * @return
     */
    List<ShoppingCart> list(ShoppingCart shoppingCart);

    /**
     * 更新商品数量
     * @param shoppingCart
     */
    @Update("update shopping_cart set number = #{number} where id = #{id}")
    void updateNumberById(ShoppingCart shoppingCart);

    /**
     * 插入购物车数据
     * @param shoppingCart
     */
    @Insert("insert into shopping_cart (name, user_id, dish_id, setmeal_id, dish_flavor, number, amount, image, create_time) " +
        " values (#{name},#{userId},#{dishId},#{setmealId},#{dishFlavor},#{number},#{amount},#{image},#{createTime})")
    void insert(ShoppingCart shoppingCart);
}
```



代码开发

创建 ShoppingCartMapper.xml :

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://mybatis.org/dtd/mybatis-3-mapper.dtd" >
<mapper namespace="com.sky.mapper.ShoppingCartMapper">

    <select id="list" parameterType="ShoppingCart" resultType="ShoppingCart">
        select * from shopping_cart
        <where>
            <if test="userId != null">
                and user_id = #{userId}
            </if>
            <if test="dishId != null">
                and dish_id = #{dishId}
            </if>
            <if test="setmealId != null">
                and setmeal_id = #{setmealId}
            </if>
            <if test="dishFlavor != null">
                and dish_flavor = #{dishFlavor}
            </if>
        </where>
        order by create_time desc
    </select>
</mapper>
```



添加购物车

- 需求分析和设计
- 代码开发
- 功能测试

功能测试

可以通过如下方式进行测试：

- 查看控制台 sql
- Swagger 接口文档测试
- 前后端联调



目录

Contents

- ◆ 缓存菜品
- ◆ 缓存套餐
- ◆ 添加购物车
- ◆ 查看购物车
- ◆ 清空购物车



查看购物车

- 需求分析和设计
- 代码开发
- 功能测试



需求分析和设计

产品原型：





需求分析和设计

接口设计：

返回数据

基本信息

Path: /user/shoppingCart/list

Method: GET

接口描述：

请求参数

名称	类型	是否必须	默认值	备注	其他信息
code	number	必须			
msg	null	非必须			
data	object []	必须			item 类型: object
— id	number	必须			
— name	string	必须			
— userId	number	必须			
— dishId	null,number	必须			
— setmealId	number,null	必须			
— dishFlavor	string	必须			
— number	number	必须			
— amount	number	必须			
— image	string	必须			
— createTime	string	必须			



查看购物车

- 需求分析和设计
- 代码开发
- 功能测试



代码开发

在 ShoppingCartController 中创建查看购物车的方法：

```
/**  
 * 查看购物车  
 * @return  
 */  
@GetMapping("/list")  
@ApiOperation("查看购物车")  
public Result<List<ShoppingCart>> list(){  
    return Result.success(shoppingCartService.showShoppingCart());  
}
```

代码开发

在 ShoppingCartService 接口中声明查看购物车的方法：

```
/**  
 * 查看购物车  
 * @return  
 */  
List<ShoppingCart> showShoppingCart();
```

代码开发

在 ShoppingCartServiceImpl 中实现查看购物车的方法：

```
/*
 * 查看购物车
 * @return
 */
public List<ShoppingCart> showShoppingCart() {
    return shoppingCartMapper.list(ShoppingCart.builder().userId(BaseContext.getCurrentId()).build());
}
```



查看购物车

- 需求分析和设计
- 代码开发
- 功能测试

功能测试

可以通过接口文档进行测试，最后完成前后端联调测试即可



目录

Contents

- ◆ 缓存菜品
- ◆ 缓存套餐
- ◆ 添加购物车
- ◆ 查看购物车
- ◆ 清空购物车



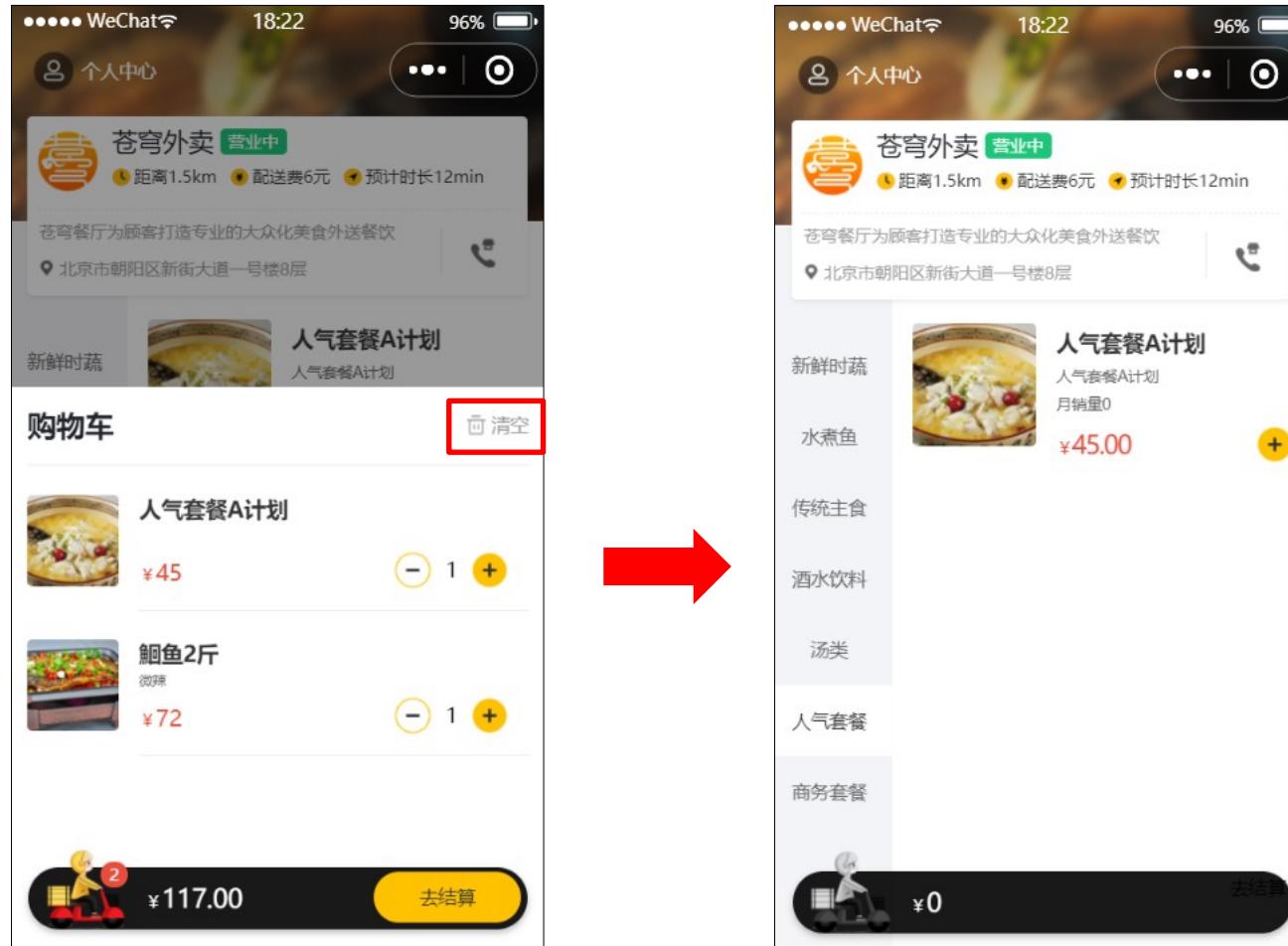
清空购物车

- 需求分析和设计
- 代码开发
- 功能测试



需求分析和设计

产品原型：



需求分析和设计

接口设计：

基本信息

Path: /user/shoppingCart/clean

Method: DELETE

接口描述：

请求参数

返回数据

名称	类型	是否必须	默认值	备注	其他信息
code	integer	必须			format: int32
data	string	非必须			
msg	string	非必须			



清空购物车

- 需求分析和设计
- 代码开发
- 功能测试

代码开发

在 ShoppingCartController 中创建清空购物车的方法：

```
/*
 * 清空购物车
 * @return
 */
@DeleteMapping("/clean")
@ApiOperation("清空购物车")
public Result<String> clean(){
    shoppingCartService.cleanShoppingCart();
    return Result.success();
}
```



代码开发

在 ShoppingCartService 接口中声明清空购物车的方法：

```
/**  
 * 清空购物车  
 */  
void cleanShoppingCart();
```



代码开发

在 ShoppingCartServiceImpl 中实现清空购物车的方法：

```
/*
 * 清空购物车
 */
public void cleanShoppingCart() {
    shoppingCartMapper.deleteByUserId(BaseContext.getCurrentId());
}
```



代码开发

在 ShoppingCartMapper 接口中创建删除购物车数据的方法：

```
/**
 * 根据用户 id 删除购物车数据
 * @param userId
 */
@Delete("delete from shopping_cart where user_id = #{userId}")
void deleteByUserId(Long userId);
```



清空购物车

- 需求分析和设计
- 代码开发
- 功能测试

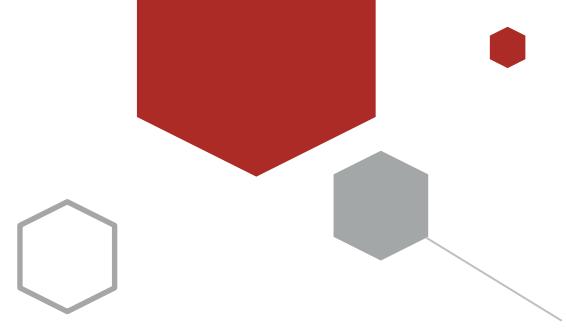
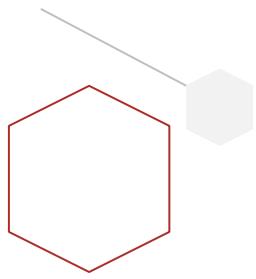


功能测试

通过 Swagger 接口文档进行测试，通过后再前端联调测试即可



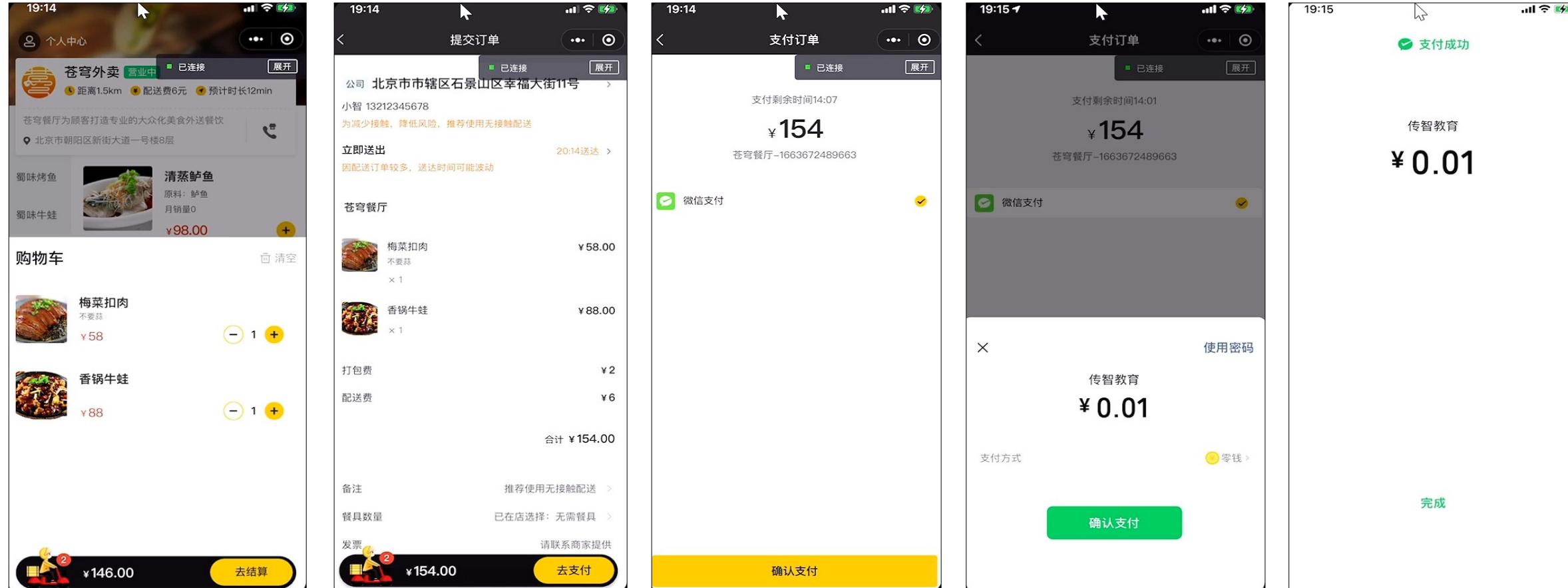
传智教育旗下高端IT教育品牌



用户下单、订单支付



黑马程序员 | 传智教育旗下
www.itheima.com 高端IT教育品牌





目录

Contents

- ◆ 导入地址簿功能代码
- ◆ 用户下单
- ◆ 订单支付



导入地址簿功能代码

- 需求分析和设计
- 代码导入
- 功能测试



需求分析和设计

产品原型：



业务功能：

- 查询地址列表
- 新增地址
- 修改地址
- 删除地址
- 设置默认地址
- 查询默认地址

需求分析和设计

接口设计：

新增地址

查询当前登录用户的所有地址信息

查询默认地址

根据id修改地址

根据id删除地址

根据id查询地址

设置默认地址



需求分析和设计

接口设计：新增地址

基本信息

Path: /user/addressBook

Method: POST

接口描述：

请求参数

Headers

参数名称	参数值	是否必须	示例	备注
Content-Type	application/json	是		

Body

名称	类型	是否必须	默认值	备注
cityCode	string	非必须		
cityName	string	非必须		
consignee	string	非必须		
detail	string	必须		详细地址
districtCode	string	非必须		
districtName	string	非必须		
id	integer	非必须		
isDefault	integer	非必须		
label	string	非必须		
phone	string	必须		手机号
provinceCode	string	非必须		
provinceName	string	非必须		
sex	string	必须		
userId	integer	非必须		

返回数据

名称	类型	是否必须	默认值	备注	其他信息
code	integer	必须			format: int32
data	object	非必须			
msg	string	非必须			

需求分析和设计

接口设计：查询登录用户所有地址

基本信息

Path: /user/addressBook/list

Method: GET

接口描述：

请求参数

返回数据

名称	类型	是否必须	默认值	备注	其他信息
code	integer	必须			format: int32
data	object	非必须			
— id	number	必须			
— userId	number	必须			
— consignee	string	必须			
— phone	string	必须			
— sex	string	必须			
— provinceCode	string	必须			
— provinceName	string	必须			
— cityCode	string	必须			
— cityName	string	必须			
— districtCode	string	必须			
— districtName	string	必须			
— detail	string	必须			
— label	string	必须			
— isDefault	number	必须			
msg	string	非必须			



需求分析和设计

接口设计：查询默认地址

基本信息

Path: /user/addressBook/default

Method: GET

接口描述：

请求参数

返回数据

名称	类型	是否必须	默认值	备注	其他信息
code	integer	必须			format: int32
data	object	非必须			
— cityCode	string	非必须			
— cityName	string	非必须			
— consignee	string	非必须			
— detail	string	非必须			
— districtCode	string	非必须			
— districtName	string	非必须			
— id	integer	非必须			format: int64
— isDefault	integer	非必须			format: int32
— label	string	非必须			
— phone	string	非必须			
— provinceCode	string	非必须			
— provinceName	string	非必须			
— sex	string	非必须			
— userId	integer	非必须			format: int64
msg	string	非必须			



需求分析和设计

接口设计：修改地址

基本信息

Path: /user/addressBook

Method: PUT

接口描述：

请求参数

Headers

参数名称	参数值	是否必须	示例	备注
Content-Type	application/json	是		

Body

名称	类型	是否必须	默认值	备注	其他信息
cityCode	string	非必须			
cityName	string	非必须			
consignee	string	非必须			
detail	string	必须		详细地址	
districtCode	string	非必须			
districtName	string	非必须			
id	integer	必须		主键值	format: int64
isDefault	integer	非必须			format: int32
label	string	非必须			
phone	string	必须		手机号	
provinceCode	string	非必须			
provinceName	string	非必须			
sex	string	必须			
userId	integer	非必须			format: int64

返回数据

名称	类型	是否必须	默认值	备注	其他信息
code	integer	必须			format: int32
data	object	非必须			
msg	string	非必须			



需求分析和设计

接口设计：根据 id 删除地址

基本信息

Path: /user/addressBook

Method: DELETE

接口描述：

请求参数

Query

参数名称	是否必须	示例	备注
id	是	101	地址id

返回数据

名称	类型	是否必须	默认值	备注	其他信息
code	integer	必须			format: int32
data	object	非必须			
msg	string	非必须			



需求分析和设计

接口设计：根据 id 查询地址

基本信息

Path: /user/addressBook/{id}

Method: GET

接口描述：

请求参数

路径参数

参数名称	示例	备注
id	101	地址id

返回数据

名称	类型	是否必须	默认值	备注	其他信息
code	number	必须			
data	object	必须			
— id	number	非必须			
— phone	string	非必须			
— consignee	string	非必须			
— userId	number	非必须			
— cityCode	string	非必须			
— provinceName	string	非必须			
— provinceCode	string	非必须			
— sex	string	非必须			
— districtName	string	非必须			
— districtCode	string	非必须			
— cityName	string	非必须			
— isDefault	number	非必须			
— label	string	非必须			
— detail	string	非必须			
msg	string	非必须			



需求分析和设计

接口设计：设置默认地址

基本信息

Path: /user/addressBook/default

Method: PUT

接口描述：

请求参数

Headers

参数名称	参数值	是否必须	示例	备注
Content-Type	application/json	是		

Body

名称	类型	是否必须	默认值	备注	其他信息
id	integer	必须		地址id	format: int64

返回数据

名称	类型	是否必须	默认值	备注	其他信息
code	integer	必须			format: int32
data	object	非必须			
msg	string	非必须			

需求分析和设计

数据库设计（ address_book 表）：

字段名	数据类型	说明	备注
id	bigint	主键	自增
user_id	bigint	用户 id	逻辑外键
consignee	varchar(50)	收货人	
sex	varchar(2)	性别	
phone	varchar(11)	手机号	
province_code	varchar(12)	省份编码	
province_name	varchar(32)	省份名称	
city_code	varchar(12)	城市编码	
city_name	varchar(32)	城市名称	
district_code	varchar(12)	区县编码	
district_name	varchar(32)	区县名称	
detail	varchar(200)	详细地址信息	具体到门牌号
label	varchar(100)	标签	公司、家、学校
is_default	tinyint(1)	是否默认地址	1 是 0 否



导入地址簿功能代码

- 需求分析和设计
- 代码导入
- 功能测试

代码导入

导入课程资料中的地址簿模块功能代码：

苍穹外卖项目 > 授课资料 > day08-用户下单、订单支付 > 资料 > 地址簿模块功能代码

名称

-  AddressBookController.java
-  AddressBookMapper.java
-  AddressBookMapper.xml
-  AddressBookService.java
-  AddressBookServiceImpl.java



导入地址簿功能代码

- 需求分析和设计
- 代码导入
- 功能测试

功能测试

可以通过如下方式进行测试：

- 查看控制台 sql 和数据库中的数据变化
- Swagger 接口文档测试
- 前后端联调



目录

Contents

- ◆ 导入地址簿功能代码
- ◆ 用户下单
- ◆ 订单支付



用户下单

- 需求分析和设计
- 代码开发
- 功能测试

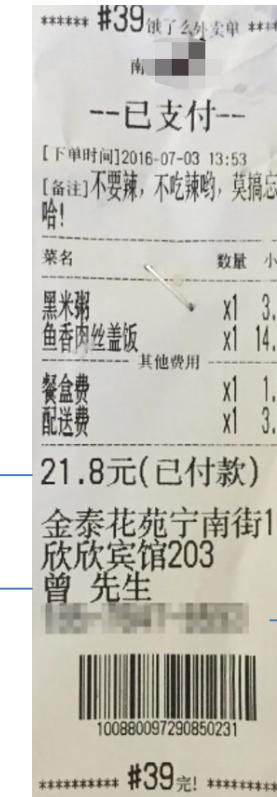


需求分析和设计

用户下单业务说明：

在电商系统中，用户是通过下单的方式通知商家，用户已经购买了商品，需要商家进行备货和发货。

用户下单后会产生订单相关数据，订单数据需要能够体现如下信息：



买的哪些商品？
每个商品数量是多少？

订单总金额是多少？

收货地址是哪？

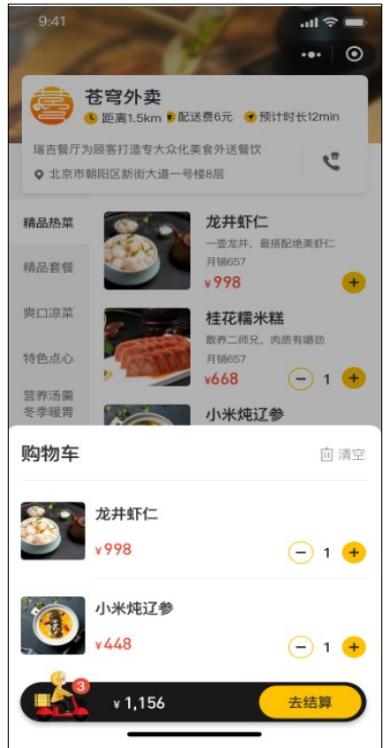
哪个用户下的单？

用户手机号是多少？



需求分析和设计

用户点餐业务流程：



购车页面

订单提交页面

订单支付页面

下单成功页面

需求分析和设计

接口设计（分析）：



请求方式：POST

请求路径：/user/order/submit

参数：

- 地址簿 id
- 配送状态（立即送出、选择送出时间）
- 打包费
- 总金额
- 备注
- 餐具数量

需求分析和设计

接口设计（分析）：



返回数据：

- 下单时间
- 订单总金额
- 订单号
- 订单 id



需求分析和设计

接口设计：

基本信息

Path: /user/order/submit

Method: POST

接口描述：

请求参数

Headers

参数名称	参数值	是否必须	示例	备注
Content-Type	application/json	是		

Body

名称	类型	是否必须	默认值	备注
addressBookId	integer	必须		地址簿id
amount	number	必须		总金额
deliveryStatus	integer	必须		配送状态： 1立即送出 0选择具体时间
estimatedDeliveryTime	string	必须		预计送达时间
packAmount	integer	必须		打包费
payMethod	integer	必须		付款方式
remark	string	必须		备注
tablewareNumber	integer	必须		餐具数量
tablewareStatus	integer	必须		餐具数量状态 1按餐量提供 0选择具体数量

返回数据

名称	类型	是否必须	默认值	备注
code	integer	必须		
data	object	必须		
└ id	integer	必须		订单id
└ orderAmount	number	必须		订单金额
└ orderNumber	string	必须		订单号
└ orderTime	string	必须		下单时间
msg	string	非必须		

需求分析和设计

数据库设计：

- 订单表 orders
- 订单明细表 order_detail

谁的订单？
送哪去？
打哪个电话联系？
多少钱？
什么时间下的单？
什么时间支付的？
订单的状态？
订单号是多少？

当前明细属于哪个订单？
具体点的是什么商品？
这个商品点了几份？

订单表和订单明细表的关系：一对多



需求分析和设计

数据库设计：订单表 orders

字段名	数据类型	说明	备注
id	bigint	主键	自增
number	varchar(50)	订单号	
status	int	订单状态	1 待付款 2 待接单 3 已接单 4 派送中 5 已完成 6 已取消
user_id	bigint	用户 id	逻辑外键
address_book_id	bigint	地址 id	逻辑外键
order_time	datetime	下单时间	
checkout_time	datetime	付款时间	
pay_method	int	支付方式	1 微信支付 2 支付宝支付
pay_status	tinyint	支付状态	0 未支付 1 已支付 2 退款
amount	decimal(10,2)	订单金额	
remark	varchar(100)	备注信息	

phone	varchar(11)	手机号	冗余字段
address	varchar(255)	详细地址信息	冗余字段
consignee	varchar(32)	收货人	冗余字段
cancel_reason	varchar(255)	订单取消原因	
rejection_reason	varchar(255)	拒单原因	
cancel_time	datetime	订单取消时间	
estimated_delivery_time	datetime	预计送达时间	
delivery_status	tinyint	配送状态	1 立即送出 0 选择具体时间
delivery_time	datetime	送达时间	
pack_amount	int	打包费	
tableware_number	int	餐具数量	
tableware_status	tinyint	餐具数量状态	1 按餐量提供 0 选择具体数量

需求分析和设计

数据库设计：订单明细表 order_detail

字段名	数据类型	说明	备注
id	bigint	主键	自增
name	varchar(32)	商品名称	冗余字段
image	varchar(255)	商品图片路径	冗余字段
order_id	bigint	订单 id	逻辑外键
dish_id	bigint	菜品 id	逻辑外键
setmeal_id	bigint	套餐 id	逻辑外键
dish_flavor	varchar(50)	菜品口味	
number	int	商品数量	
amount	decimal(10,2)	商品单价	



用户下单

- 需求分析和设计
- 代码开发
- 功能测试



代码开发

根据用户下单接口的参数设计 DTO：

Body

名称	类型	是否必须	默认值	备注
addressBookId	integer	必须		地址簿id
amount	number	必须		总金额
deliveryStatus	integer	必须		配送状态： 1立即送出 0选择具体时间
estimatedDeliveryTime	string	必须		预计送达时间
packAmount	integer	必须		打包费
payMethod	integer	必须		付款方式
remark	string	必须		备注
tablewareNumber	integer	必须		餐具数量
tablewareStatus	integer	必须		餐具数量状态 1按餐量提供 0选择具体数量



```
@Data
public class OrdersSubmitDTO implements Serializable {
    // 地址簿 id
    private Long addressBookId;
    // 付款方式
    private int payMethod;
    // 备注
    private String remark;
    // 预计送达时间
    @JsonFormat(shape = JsonFormat.Shape.STRING, pattern = "yyyy-MM-dd HH:mm:ss")
    private LocalDateTime estimatedDeliveryTime;
    // 配送状态 1 立即送出 0 选择具体时间
    private Integer deliveryStatus;
    // 餐具数量
    private Integer tablewareNumber;
    // 餐具数量状态 1 按餐量提供 0 选择具体数量
    private Integer tablewareStatus;
    // 打包费
    private Integer packAmount;
    // 总金额
    private BigDecimal amount;
}
```

代码开发

根据用户下单接口的返回结果设计 VO：

返回数据

名称	类型	是否必须	默认值	备注	其他信息
code	integer	必须			format: int32
data	object	必须			
└ id	integer	必须		订单id	format: int64
└ orderAmount	number	必须		订单金额	
└ orderNumber	string	必须		订单号	
└ orderTime	string	必须		下单时间	format: date-time
msg	string	非必须			



```
@Data  
@Builder  
@NoArgsConstructor  
@AllArgsConstructor  
public class OrderSubmitVO implements Serializable {  
    // 订单id  
    private Long id;  
    // 订单号  
    private String orderNumber;  
    // 订单金额  
    private BigDecimal orderAmount;  
    // 下单时间  
    private LocalDateTime orderTime;  
}
```



代码开发

创建 OrderController 并提供用户下单方法：

```
@RestController("userOrderController")
@RequestMapping("/user/order")
@Slf4j
@Api(tags = "C 端订单接口")
public class OrderController {

    @Autowired
    private OrderService orderService;

    /**
     * 用户下单
     * @param ordersSubmitDTO
     * @return
     */
    @PostMapping("/submit")
    @ApiOperation("用户下单")
    public Result<OrderSubmitVO> submit(@RequestBody OrdersSubmitDTO ordersSubmitDTO) {
        log.info("用户下单：{}", ordersSubmitDTO);
        OrderSubmitVO orderSubmitVO = orderService.submitOrder(ordersSubmitDTO);
        return Result.success(orderSubmitVO);
    }
}
```

代码开发

创建 OrderService 接口，并声明用户下单方法：

```
public interface OrderService {  
  
    /**  
     * 用户下单  
     * @param ordersSubmitDTO  
     * @return  
     */  
    OrderSubmitVO submitOrder(OrdersSubmitDTO ordersSubmitDTO);  
}
```



代码开发

创建 OrderServiceImpl 实现 OrderService 接口（1）：

```
/**  
 * 订单  
 */  
@Service  
@Slf4j  
public class OrderServiceImpl implements OrderService {  
  
    @Autowired  
    private OrderMapper orderMapper;  
    @Autowired  
    private OrderDetailMapper orderDetailMapper;  
    @Autowired  
    private ShoppingCartMapper shoppingCartMapper;  
    @Autowired  
    private AddressBookMapper addressBookMapper;
```



代码开发

创建 OrderServiceImpl 实现 OrderService 接口（2）：

```
/*
 * 用户下单
 * @param ordersSubmitDTO
 * @return
 */
@Transactional
public OrderSubmitVO submitOrder(OrdersSubmitDTO ordersSubmitDTO) {
    // 异常情况的处理（收货地址为空、购物车为空）
    AddressBook addressBook = addressBookMapper.getByName(ordersSubmitDTO.getAddressBookName());
    if (addressBook == null) {
        throw new AddressBookBusinessException(MessageConstant.ADDRESS_BOOK_IS_NULL);
    }

    Long userId = BaseContext.getCurrentId();
    ShoppingCart shoppingCart = new ShoppingCart();
    shoppingCart.setUserId(userId);

    // 查询当前用户的购物车数据
    List<ShoppingCart> shoppingCartList = shoppingCartMapper.list(shoppingCart);
    if (shoppingCartList == null || shoppingCartList.size() == 0) {
        throw new ShoppingCartBusinessException(MessageConstant.SHOPPING_CART_IS_NULL);
    }
}
```



代码开发

创建 OrderServiceImpl 实现 OrderService 接口（3）：

```
// 构造订单数据
Orders order = new Orders();
BeanUtils.copyProperties(ordersSubmitDTO,order);
order.setPhone(addressBook.getPhone());
order.setAddress(addressBook.getDetail());
order.setConsignee(addressBook.getConsignee());
order.setNumber(String.valueOf(System.currentTimeMillis()));
order.setUserId(userId);
order.setStatus(Orders.PENDING_PAYMENT);
order.setPayStatus(Orders.UN_PAID);
order.setOrderTime(LocalDateTime.now());

// 向订单表插入 1 条数据
orderMapper.insert(order);
```



代码开发

创建 OrderServiceImpl 实现 OrderService 接口（4）：

```
// 订单明细数据
List<OrderDetail> orderDetailList = new ArrayList<>();
for (ShoppingCart cart : shoppingCartList) {
    OrderDetail orderDetail = new OrderDetail();
    BeanUtils.copyProperties(cart, orderDetail);
    orderDetail.setOrderId(order.getId());
    orderDetailList.add(orderDetail);
}
// 向明细表插入 n 条数据
orderDetailMapper.insertBatch(orderDetailList);

// 清理购物车中的数据
shoppingCartMapper.deleteByUserId(userId);

// 封装返回结果
OrderSubmitVO orderSubmitVO = OrderSubmitVO.builder()
    .id(order.getId())
    .orderNumber(order.getNumber())
    .orderAmount(order.getAmount())
    .orderTime(order.getOrderTime())
    .build();
return orderSubmitVO;
}
```



代码开发

创建 OrderMapper 接口和对应的 xml 映射文件：

```
@Mapper
public interface OrderMapper {
    /**
     * 插入订单数据
     * @param order
     */
    void insert(Orders order);
}
```

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://mybatis.org/dtd/mybatis-3-mapper.dtd" >
<mapper namespace="com.sky.mapper.OrderMapper">

    <insert id="insert" parameterType="Orders" useGeneratedKeys="true" keyProperty="id">
        insert into orders
            (number, status, user_id, address_book_id, order_time, checkout_time, pay_method, pay_status,
            amount, remark, phone, address, consignee, estimated_delivery_time, delivery_status,
            pack_amount, tableware_number, tableware_status)
        values
            (#{number}, #{status}, #{userId}, #{addressBookId}, #{orderTime}, #{checkoutTime}, #{payMethod}, #{payStatus},
            #{amount}, #{remark}, #{phone}, #{address}, #{consignee}, #{estimatedDeliveryTime}, #{deliveryStatus},
            #{packAmount}, #{tablewareNumber}, #{tablewareStatus})
    </insert>

</mapper>
```



代码开发

创建 OrderDetailMapper 接口和对应的 xml 映射文件：

```
@Mapper
public interface OrderDetailMapper {
    /**
     * 批量插入订单明细数据
     * @param orderDetails
     */
    void insertBatch(List<OrderDetail> orderDetails);
}
```

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://mybatis.org/dtd/mybatis-3-mapper.dtd" >
<mapper namespace="com.sky.mapper.OrderDetailMapper">

    <insert id="insertBatch" parameterType="list">
        insert into order_detail (name, order_id, dish_id, setmeal_id, dish_flavor, number, amount, image)
        values
        <foreach collection="orderDetails" item="od" separator=",">
            (#{od.name},#{od.orderId},#{od.dishId},#{od.setmealId},#{od.dishFlavor},#{od.number},#{od.amount},#{od.image})
        </foreach>
    </insert>

</mapper>
```



用户下单

- 需求分析和设计
- 代码开发
- 功能测试

功能测试

可以通过如下方式进行测试：

- 查看控制台 sql
- 前后端联调
- 查看数据库中数据变化



目录

Contents

- ◆ 导入地址簿功能代码
- ◆ 用户下单
- ◆ 订单支付



订单支付

- 微信支付介绍
- 微信支付准备工作
- 代码导入
- 功能测试

微信支付介绍

微信支付产品：



参考：https://pay.weixin.qq.com/static/product/product_index.shtml



微信支付介绍

微信支付接入流程：



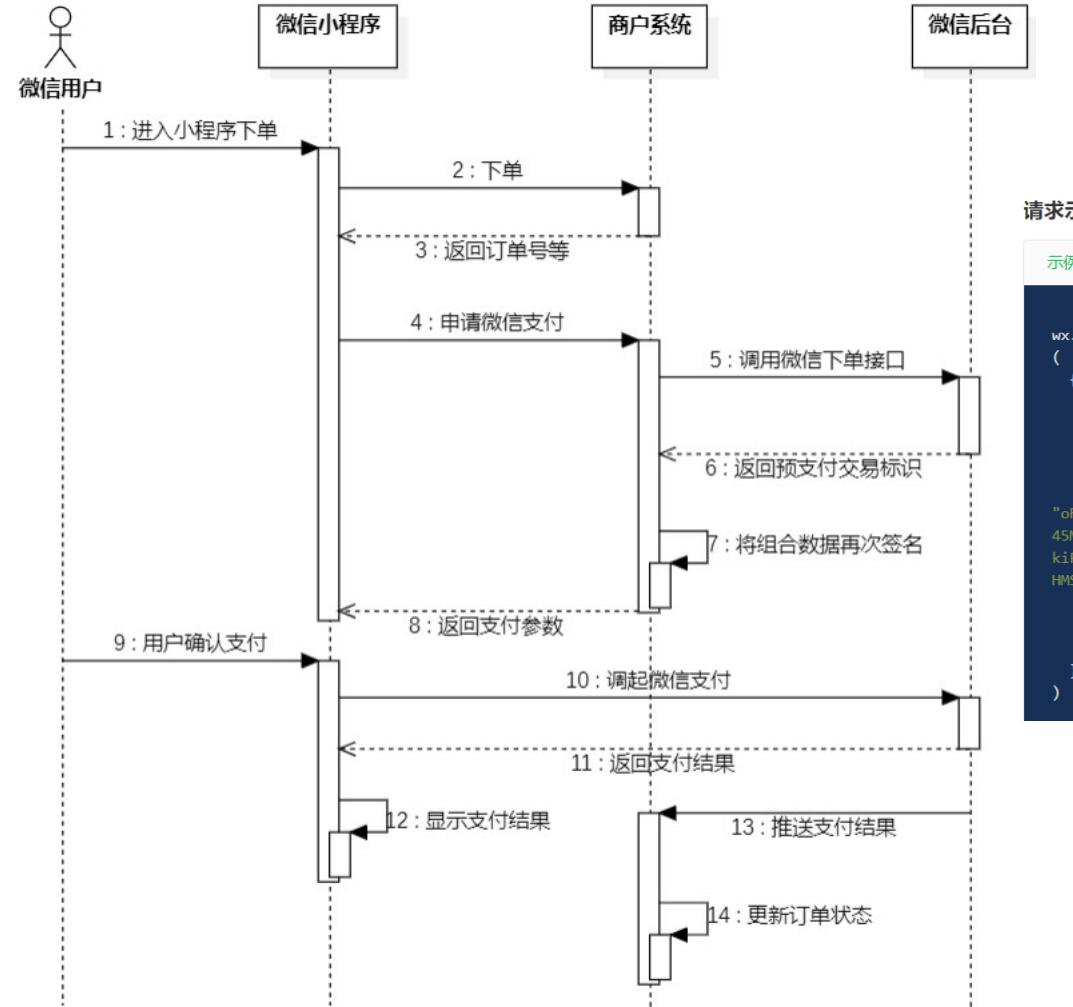
接入微信支付





微信支付介绍

微信小程序支付时序图：



请求示例

示例

```
wx.requestPayment
(
  {
    "timeStamp": "1414561699",
    "nonceStr": "5K8264ILTKCH16CQ2502SI8ZNMTM67VS",
    "package": "prepay_id=wx201410272009395522657a690389285100",
    "signType": "RSA",
    "paySign": "oR9d8PuhnIc+YZ8cBHFCwfgpak9gd7vaRvkYD7rthRAZ\X+QBhcCYL21N7cHCTUxbQ+EAt6Uy+1wSN22f5YZvI45MLko8Fso0jm46v5hqcVwrk6uddkGuT+Cdvu4WBqDzaDjnNa5UK3GFE1Wf12gHxIIY51LdUgWFts17D4WuoLLkiFZV+JSHMvH7eaLdt9NSGBovBwu5yYKUR7skR8Fu+LozcSqQixnLEZUfye55feLOQTUVzLmR9pNtPbPsu6WhbNHMS35s2+AehHvz+n64GDmXxbX++IOBvm2e1Hu3PsOUGRwhudhVf7UcGcunXt8cqNjKNqZLhLw4jq\xDg==",
    "success":function(res){},
    "fail":function(res){},
    "complete":function(res){}
  }
)
```

微信支付介绍

JSAPI下单：商户系统调用该接口在微信支付服务后台生成**预支付交易单**

适用对象：直连商户

请求URL: <https://api.mch.weixin.qq.com/v3/pay/transactions/jsapi>

请求方式: POST

```
{  
    "mchid": "1900006XXX",  
    "out_trade_no": "1217752501201407033233368318",  
    "appid": "wxda645e0bc2cXXX",  
    "description": "Image形象店-深圳腾大-QQ公仔",  
    "notify_url": "https://www.weixin.qq.com/wxpay/pay.php",  
    "amount": {  
        "total": 1,  
        "currency": "CNY"  
    },  
    "payer": {  
        "openid": "o4GgauInH_RCEdvrrNGrntXDuXXX"  
    }  
}
```

返回参数

参数名	变量	类型[长度限制]	必填	描述
预支付交易会话标识	prepay_id	string[1,64]	是	预支付交易会话标识。用于后续接口调用中使用，该值有效期为2小时 示例值: wx201410272009395522657a690389285100



微信支付介绍

微信小程序调起支付：通过 JSAPI 下单接口获取到发起支付的必要参数 prepay_id，然后使用微信支付提供的小程序方法调起小程序支付

接口名称：wx.requestPayment，详见[小程序API文档](#)

Object参数说明：

参数名	变量	类型[长度限制]	必填	描述
时间戳	timeStamp	string[1,32]	是	当前的时间，其他详见 时间戳规则 。 示例值：1414561699
随机字符串	nonceStr	string[1,32]	是	随机字符串，不长于32位。 示例值：5K8264ILTKCH16CQ2502SI8ZNMTM67VS
订单详情扩展字符串	package	string[1,128]	是	小程序下单接口返回的prepay_id参数值， 提交格式如：prepay_id=*** 示例值：prepay_id=wx20141027200939522657a690389285100 5522657a690389285100
签名方式	signType	string[1,32]	是	签名类型，默认为RSA，仅支持RSA。 示例值：RSA
签名	paySign	string[1,512]	是	签名，使用字段appId、timeStamp、nonceStr、package计算得出的签名值 示例值：oR9d8PuhnIc+YZ8cBHFCwfcpaK9gd7vaRvkYD7rthRAZ\X+QBhcCYL21N7cHCTUxbQ+EAt6Uy+lwSN22f5YZvI45MLko8Pfso0jm46v5hqcVwrk6uddkGuT+Cdvu4WBqDzaDjnNa5UK3GFe1Wf12gHxIIY5lLdUgWFts17D4Wuo1LLkiFZV+JSHMvH7eaLdT9N5GBovBwu5yYKUR7skR8Fu+LozcSqQixn1EZUFyE55feLoQTYzLmR9pNtPbPsu6WvhBNHMS3Ss2+AehHvz+n64GDmXxbX++IOBvm2o1Hu3Ps0UGRwhudhVf7UcGcunXt8cqNjKNqZLhLw4jq\xDg==， "success":function(res){}, "fail":function(res){}, "complete":function(res){} }

请求示例

示例

```
wx.requestPayment
(
{
  "timeStamp": "1414561699",
  "nonceStr": "5K8264ILTKCH16CQ2502SI8ZNMTM67VS",
  "package": "prepay_id=wx20141027200939522657a690389285100",
  "signType": "RSA",
  "paySign": "oR9d8PuhnIc+YZ8cBHFCwfcpaK9gd7vaRvkYD7rthRAZ\X+QBhcCYL21N7cHCTUxbQ+EAt6Uy+lwSN22f5YZvI45MLko8Pfso0jm46v5hqcVwrk6uddkGuT+Cdvu4WBqDzaDjnNa5UK3GFe1Wf12gHxIIY5lLdUgWFts17D4Wuo1LLkiFZV+JSHMvH7eaLdT9N5GBovBwu5yYKUR7skR8Fu+LozcSqQixn1EZUFyE55feLoQTYzLmR9pNtPbPsu6WvhBNHMS3Ss2+AehHvz+n64GDmXxbX++IOBvm2o1Hu3Ps0UGRwhudhVf7UcGcunXt8cqNjKNqZLhLw4jq\xDg==",
  "success":function(res){},
  "fail":function(res){},
  "complete":function(res){}
}
)
```

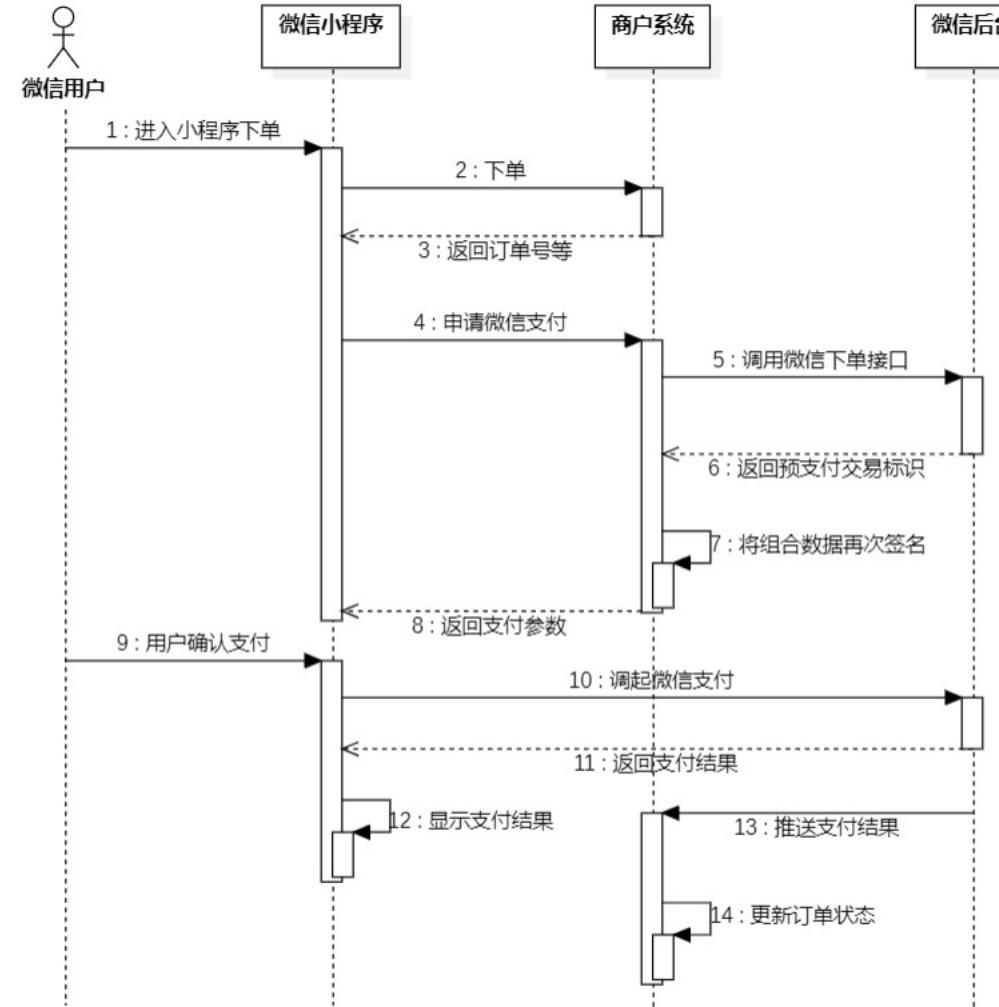


订单支付

- 微信支付介绍
- 微信支付准备工作
- 代码导入
- 功能测试

微信支付准备工作

微信小程序支付时序图：



调用过程如何保证数据安全？

微信后台如何调用到商户系统？

微信支付介绍

获取微信支付平台证书、商户私钥文件：

- apiclient_key.pem
- wechatpay_166D96F876F45C7D07CE98952A96EC980368ACFC.pem



微信支付准备工作

获取临时域名：支付成功后微信服务通过该域名回调我们的程序

设置与安装

① 下载 cpolar

cpolar易于安装。下载具有零运行时依赖性的单个二进制文件。

[Download for Windows](#)

[Mac OS X](#) [Linux](#) [Mac \(32-bit\)](#) [Windows \(32-bit\)](#)
[Linux \(ARM\)](#) [Linux \(mips\)](#) [Linux \(mipsle\)](#)
[Linux \(32-bit\)](#) [FreeBSD \(64-Bit\)](#) [FreeBSD \(32-bit\)](#)

② 解压缩安装

在Linux或OSX上，您可以使用以下命令从终端解压缩cpolar。在Windows上，只需双击cpolar.zip即可。

```
$ unzip /path/to/cpolar.zip
```

大多数人将cpolar保存在他们的用户文件夹中或设置别名以便于访问。

③ 连接您的帐户

运行此命令会将您帐户的authtoken添加到您的cpolar.yml文件中。这将为您提供更多功能，所有打开的隧道将在此处的仪表板中列出。

```
$ ./cpolar authtoken NzKxZGI0ZmMtYmQwMi00ZmQ3L
```

④ 燃烧起来,动起来

阅读有关如何使用cpolar的[文档](#)。通过从命令行运行它来尝试：

```
$ ./cpolar help
```

要在端口80上启动HTTP隧道，请运行以下命令：

```
$ ./cpolar http 80
```

C:\Windows\System32\cmd.exe

```
C:\Program Files\cpolar>cpolar.exe authtoken NzKxZGI0ZmMtYmQwMi00ZmQ3L
Authtoken saved to configuration file: C:\Users\itcast/.cpolar/cpolar.yml
```

```
C:\Program Files\cpolar>
```

```
C:\Program Files\cpolar>cpolar.exe http 8080
```

cpolar by @bestexpresser

Tunnel Status	online
Account	itcast (Plan: Free)
Version	2.86.16/2.96
Web Interface	127.0.0.1:4042
Forwarding	http://102bd5a9.vip.cpolar.cn -> http://localhost:8080
Forwarding	https://102bd5a9.vip.cpolar.cn -> http://localhost:8080
# Conn	0
Avg Conn Time	0.00ms



订单支付

- 微信支付介绍
- 微信支付准备工作
- 代码导入
- 功能测试



代码导入

微信支付相关配置：

```
sky:  
    jwt: <6 keys>  
    alioss: <4 keys>  
    wechat:  
        appId: ${sky.wechat.appId}  
        secret: ${sky.wechat.secret}  
        mchid : ${sky.wechat.mchid}  
        mchSerialNo: ${sky.wechat.mchSerialNo}  
        privateKeyFilePath: ${sky.wechat.privateKeyFilePath}  
        apiV3Key: ${sky.wechat.apiV3Key}  
        weChatPayCertFilePath: ${sky.wechat.weChatPayCertFilePath}  
        notifyUrl: ${sky.wechat.notifyUrl}  
        refundNotifyUrl: ${sky.wechat.refundNotifyUrl}
```

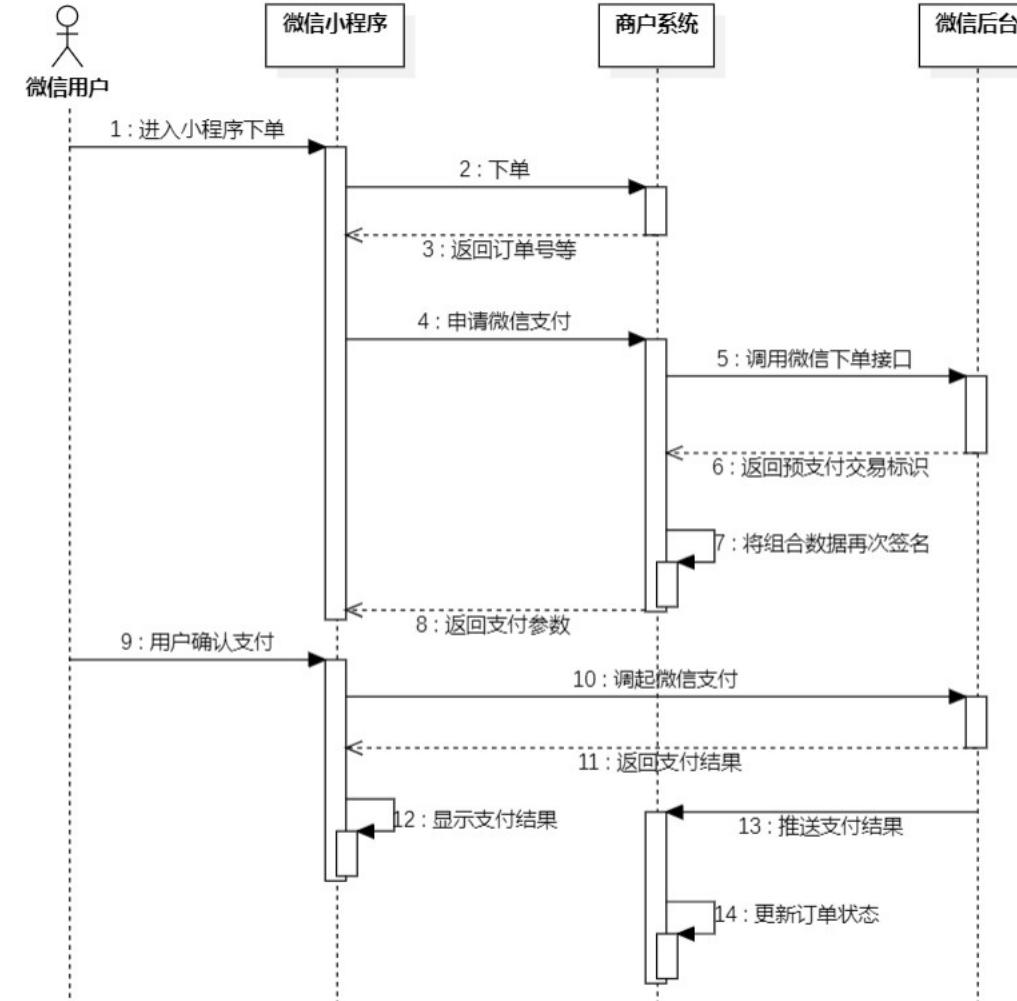
```
@Component  
@ConfigurationProperties(prefix = "sky.wechat")  
@Data  
public class WeChatProperties {  
  
    private String appId; //小程序的appId  
    private String secret; //小程序的秘钥  
    private String mchid; //商户号  
    private String mchSerialNo; //商户API证书的证书序列号  
    private String privateKeyFilePath; //商户私钥文件  
    private String apiV3Key; //证书解密的密钥  
    private String weChatPayCertFilePath; //平台证书  
    private String notifyUrl; //支付成功的回调地址  
    private String refundNotifyUrl; //退款成功的回调地址  
  
}
```



代码导入

导入微信支付功能代码：

- OrderController.java
- OrderMapper.java
- OrderMapper.xml
- OrderService.java
- OrderServiceImpl.java
- PayNotifyController.java



代码导入

微信支付相关配置：



传智教育旗下高端IT教育品牌



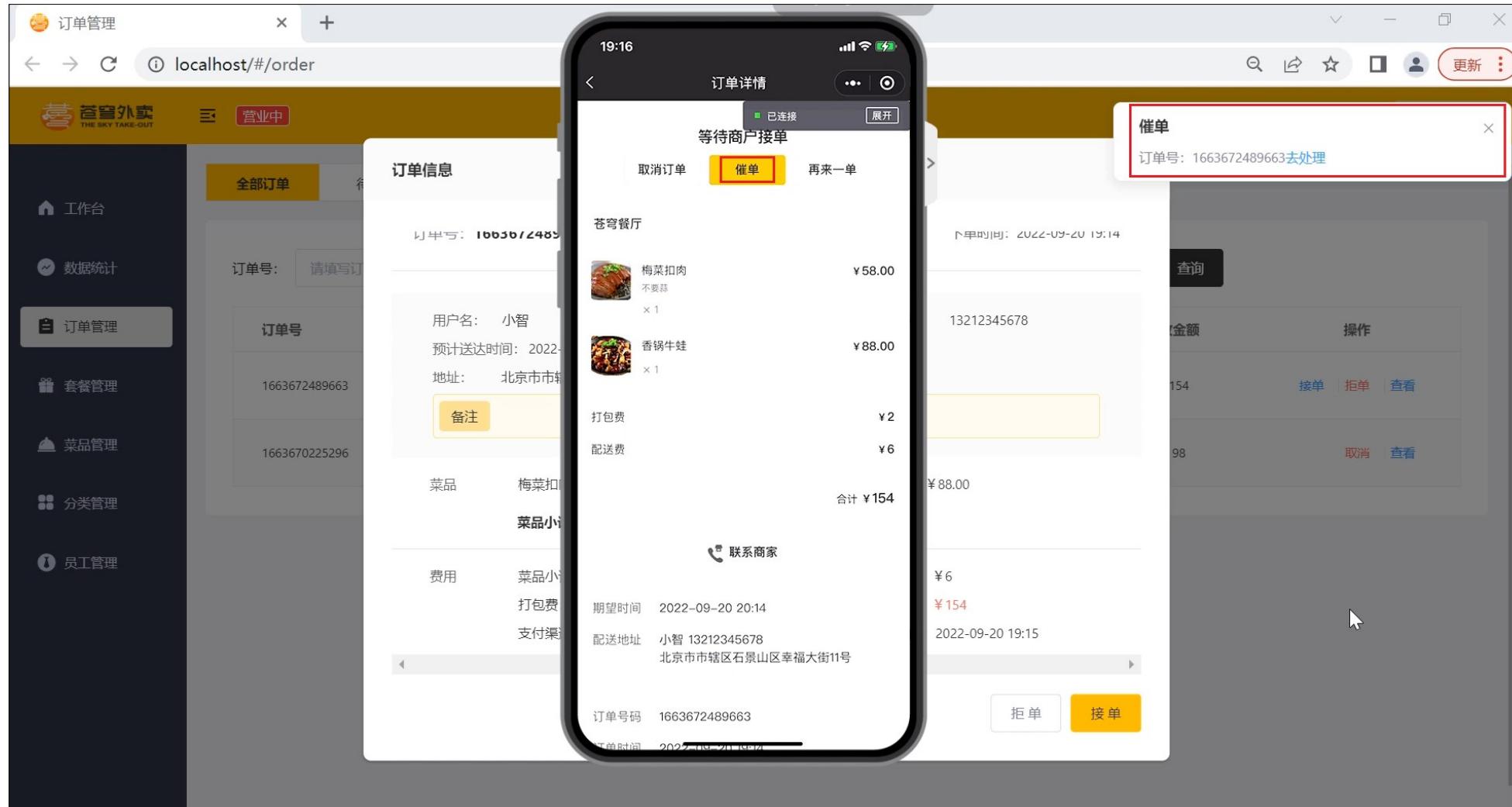
订单状态定时处理、来单提醒和客户催单



支付超时的订单如何处理？

A screenshot of a web-based order management system. The title bar says "订单管理" (Order Management) and the URL is "localhost/#/order". The header includes a search bar and user information. A yellow banner at the top says "营业中" (Open). The main area has tabs: 全部订单 (All Orders), 待接单 (Pending Pickup), 待派送 (Pending Delivery), 派送中 (In Progress) [highlighted], 已完成 (Completed), and 已取消 (Cancelled). Below the tabs is a search form with fields for 订单号 (Order ID), 手机号 (Phone Number), 下单时间 (Order Time), and a "查询" (Search) button. A table lists orders:
Order ID: 1663672489663; Items: 梅菜扣肉*1; 香锅牛蛙*1; Address: 北京市市辖区石景山区幸福大街11号; Expected Delivery Time: 2022-09-20 20:14; Notes: 0; Utensil Count: 0; Actions: 完成 (Complete), 取消 (Cancel), 查看 (View). A cursor is hovering over the "完成" button.

派送中的订单一直不点击完成如何处理？





目录

Contents

- ◆ Spring Task
- ◆ 订单状态定时处理
- ◆ WebSocket
- ◆ 来单提醒
- ◆ 客户催单



Spring Task

- 介绍
- cron 表达式
- 入门案例



Spring Task 介绍

Spring Task 是 Spring 框架提供的任务调度工具，可以按照约定的时间自动执行某个代码逻辑。

定位：定时任务框架

作用：定时自动执行某段 Java 代码



为什么要在 Java 程序中使用 Spring
Task ?

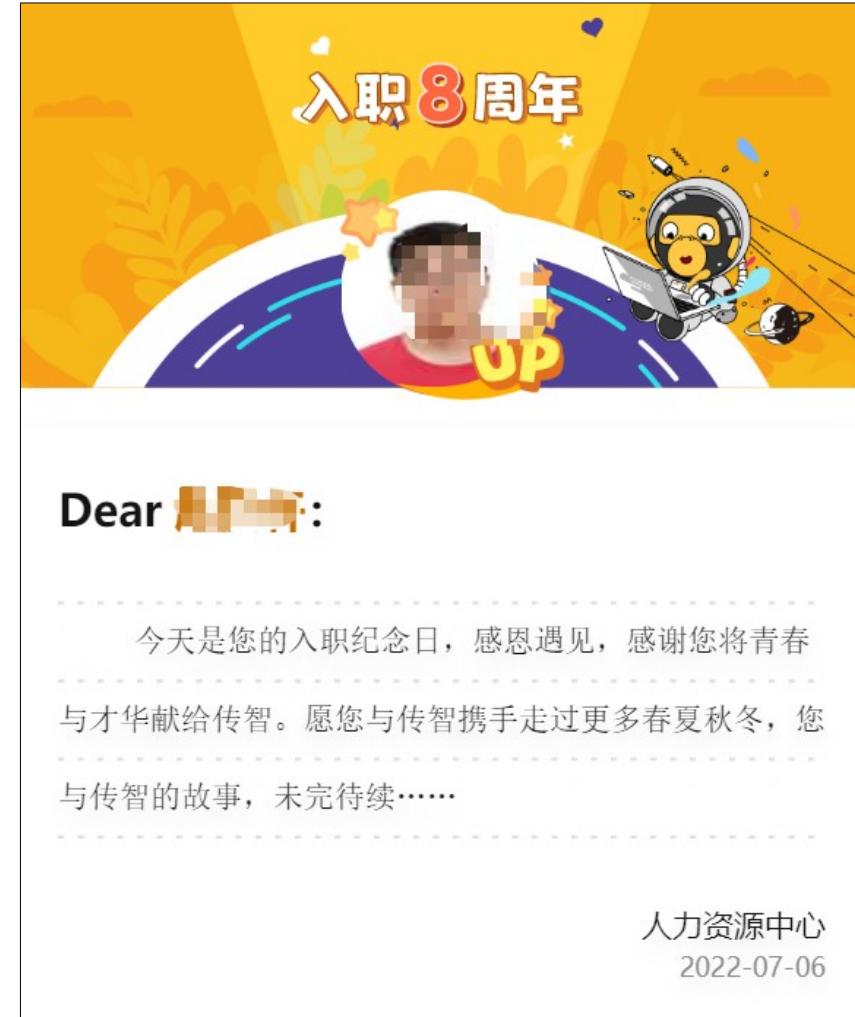


Spring Task 介绍

应用场景：

- 信用卡每月还款提醒
- 银行贷款每月还款提醒
- 火车票售票系统处理未支付订单
- 入职纪念日为用户发送通知

只要是需要定时处理的场景都可以使用 Spring Task





Spring Task

- 介绍
- cron 表达式
- 入门案例



cron 表达式

cron 表达式其实就是一个字符串，通过 cron 表达式可以定义任务触发的时间

构成规则：分为 6 或 7 个域，由空格分隔开，每个域代表一个含义

每个域的含义分别为：秒、分钟、小时、日、月、周、年（可选）

秒	分钟	小时	日	月	周	年
0	0	9	12	10	?	2022

2022 年 10 月 12 日上午 9 点整 对应的 cron 表达式为 **0 9 12 10 ? 2022**



cron 表达式

cron 表达式在线生成器：<https://cron.qqe2.com/>

秒 分钟 小时 日 月 周 年

每秒 允许的通配符 [- * /]

周期从 - 秒

从 秒开始, 每 秒执行一次

指定

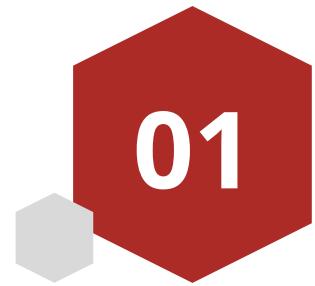
00 01 02 03 04 05 06 07 08 09
 10 11 12 13 14 15 16 17 18 19
 20 21 22 23 24 25 26 27 28 29
 30 31 32 33 34 35 36 37 38 39
 40 41 42 43 44 45 46 47 48 49
 50 51 52 53 54 55 56 57 58 59

表达式

秒	分钟	小时	日	月	星期	年
<input type="text" value="*"/>	<input type="text" value="?"/>	<input type="text" value="*"/>				

表达式字段:

Cron 表达式: 反解析到UI



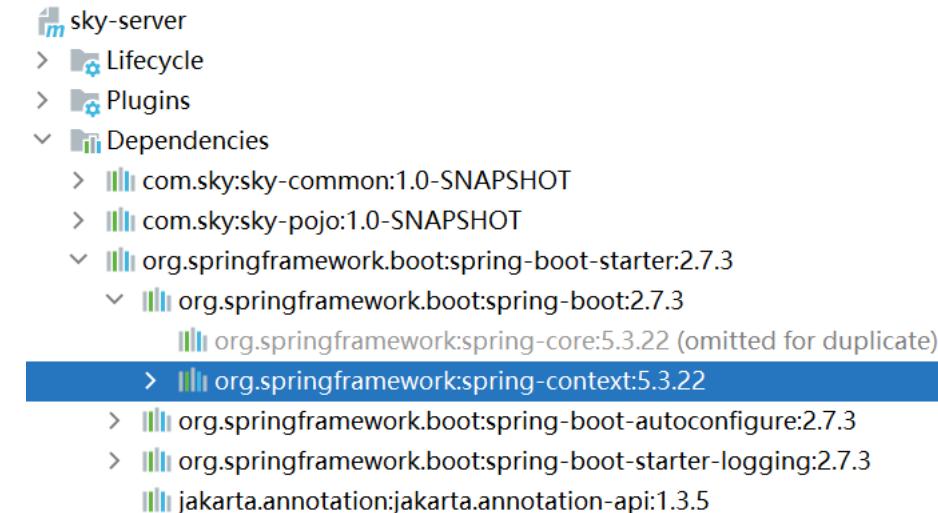
Spring Task

- 介绍
- cron 表达式
- 入门案例

入门案例

Spring Task 使用步骤：

- ① 导入 maven 坐标 spring-context (已存在)
- ② 启动类添加注解 @EnableScheduling 开启任务调度
- ③ 自定义定时任务类





目录

Contents

- ◆ Spring Task
- ◆ 订单状态定时处理
- ◆ WebSocket
- ◆ 来单提醒
- ◆ 客户催单



订单状态定时处理

- 需求分析
- 代码开发
- 功能测试



需求分析

用户下单后可能存在的情况：

- 下单后未支付，订单一直处于“待支付”状态
- 用户收货后管理端未点击完成按钮，订单一直处于“派送中”状态

对于上面两种情况

- 通过定时任务如果存在则修改
- 通过定时任务

支付超时订单），如

完成”



订单状态定时处理

- 需求分析
- 代码开发
- 功能测试



代码开发

自定义定时任务类 OrderTask（待完善）：

```
@Component
@Slf4j
public class OrderTask {
    @Autowired
    private OrderMapper orderMapper;

    /**
     * 支付超时订单处理
     * 对于下单后超过 15 分钟仍未支付的订单自动修改状态为 [已取消]
     */
    @Scheduled(cron = "0 * * * ?") // 每分钟执行一次
    public void processTimeoutOrder() {
        log.info("开始进行支付超时订单处理 :{}", LocalDateTime.now());
    }

    /**
     * 派送中状态的订单处理
     * 对于一直处于派送中状态的订单，自动修改状态为 [已完成]
     */
    @Scheduled(cron = "0 0 1 * * ?") // 每天凌晨 1 点执行一次
    public void processDeliveryOrder(){
        log.info("开始进行未完成订单状态处理 :{}", LocalDateTime.now());
    }
}
```



代码开发

在 OrderMapper 接口中扩展方法

```
/**  
 * 根据订单状态和下单时间查询订单  
 * @param status  
 * @param orderTime  
 */  
@Select("select * from orders where status = #{status} and order_time < #{orderTime}")  
List<Orders> getByStatusAndOrdertimeLT(Integer status, LocalDateTime orderTime);
```



代码开发

完善定时任务类的 processTimeoutOrder 方法：

```
@Scheduled(cron = "0 * * * * ?")
public void processTimeoutOrder(){
    Log.info(" 处理支付超时订单 : {}", new Date());

    LocalDateTime time = LocalDateTime.now().plusMinutes(-15);
    List<Orders> ordersList = orderMapper.getByStatusAndOrdertimeLT(Orders.PENDING_PAYMENT, time);

    if(ordersList != null && ordersList.size() > 0){
        ordersList.forEach(order -> {
            order.setStatus(Orders.CANCELLED);
            order.setCancelReason(" 支付超时，自动取消 ");
            order.setCancelTime(LocalDateTime.now());
            orderMapper.update(order);
        });
    }
}
```



代码开发

完善定时任务类的 processDeliveryOrder 方法：

```
@Scheduled(cron = "0 0 1 * * ?")
public void processDeliveryOrder(){
    log.info("处理派送中订单：{}", new Date());

    LocalDateTime time = LocalDateTime.now().plusMinutes(-60);
    List<Orders> ordersList = orderMapper.getByStatusAndOrdertimeLT(Orders.DELIVERY_IN_PROGRESS, time);

    if(ordersList != null && ordersList.size() > 0){
        ordersList.forEach(order -> {
            order.setStatus(Orders.COMPLETED);
            orderMapper.update(order);
        });
    }
}
```



订单状态定时处理

- 需求分析
- 代码开发
- 功能测试

功能测试

可以通过如下方式进行测试：

- 查看控制台 sql
- 查看数据库中数据变化



目录

Contents

- ◆ Spring Task
- ◆ 订单状态定时处理
- ◆ WebSocket
- ◆ 来单提醒
- ◆ 客户催单



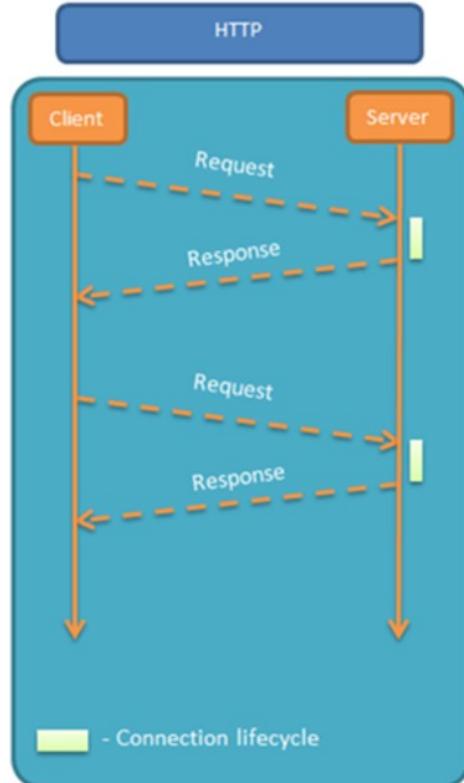
WebSocket

- 介绍
- 入门案例



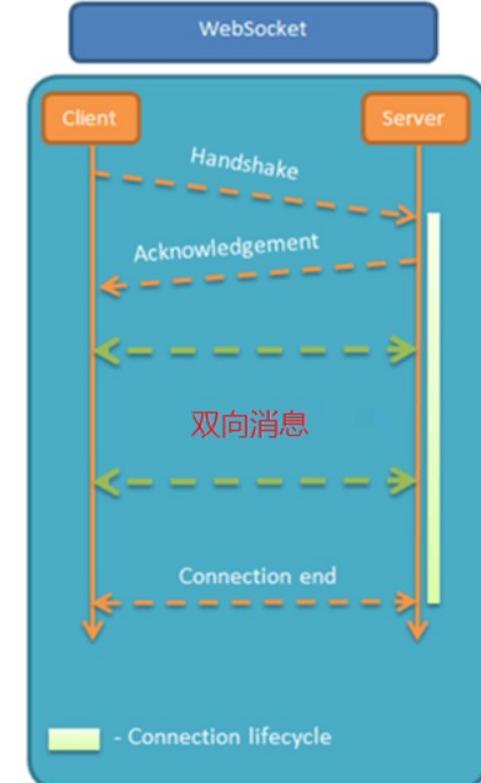
介绍

WebSocket 是基于 TCP 的一种新的**网络协议**。它实现了浏览器与服务器全双工通信——浏览器和服务器只需要完成一次握手，两者之间就可以创建**持久性**的连接，并进行**双向**数据传输。



HTTP 协议和 WebSocket 协议对比：

- HTTP 是**短连接**
- WebSocket 是**长连接**
- HTTP 通信是**单向**的，基于请求响应模式
- WebSocket 支持**双向**通信
- HTTP 和 WebSocket 底层都是 TCP 连接





介绍

应用场景：

- 视频弹幕
- 网页聊天
- 体育实况更新
- 股票基金报价实时更新

网站首页 加收藏 移动客户端 东方财富 天天基金网 东方财富证券 东方财富期货 Choice数据 股吧 登录 我的菜单 证券交易 基金交易 查看最新 恢复默认

行情中心

全景图

上证指数 深成指 创业板 深证综指 成份B指 期指 恒指 道指 日经 美元 黄金 原油 离岸

输入股票代码、名称或简称 设置分时图

上证指数 3089.31 ▼-0.25% 成份B指 7558.64 ▼-0.16% IF当月连续 3761.0 ▼-0.57% 东方财富 17.80 ▼-0.07 ▼-0.39%

沪深300 上证50 B股指数 科创50 7602.84 7570.88 3813.8 18.25

3115.39 3096.91 3078.43 18.25

3078.43 17.87 17.49

3096.91 17.87 17.49

3115.39 3078.43 3096.91 17.87 17.49

最新消息 信诺发力智能制造领域 18:19 嘉银金科第三季度实现营收约8.94亿元 同比增长55% 18:19 鹏辉能源：钠离子电池研发进展良好 电芯性能测试结果较理想 18:19 四方新

自选股 个股聚焦 个股公告 更多>> 行业板块 概念板块 地域板块 更多>> 个股资金流入 更多>> 个股资金流出 更多>>

代码	名称	最新价	涨跌幅	主力净流入	领涨股	涨幅	主力净流	占比	名称	最新价	涨跌幅	主力净流入	主力净占
08137	洪桥集团	0.30	3.45%	电池	3.03%	11.06亿	鹏辉能源	12.22%	九安医疗	62.84	9.99%	7.01亿	13.50
301165	C锐捷	39.30	1.55%	中药	2.78%	5.10亿	康缘药业	10.01%	多氟多	36.95	10.00%	4.32亿	27.20
002642	荣联科技	8.03	-0.37%	医药商业	2.46%	5571万	塞力医疗	10.03%	鹏辉能源	74.28	12.22%	3.23亿	12.60
688359	三孚新科	90.02	3.69%	贵金属	2.33%	6626万	赤峰黄金	5.58%	以岭药业	43.99	10.00%	2.74亿	3.45
600766	园城黄金	13.57	2.57%	房地产	1.98%	12.26亿	中国武夷	10.00%	中交地产	13.98	9.99%	2.64亿	21.46

盘口异动 1:33 上海易连 60日新高 6.83元 14:54:31 好上好 大笔卖出 489手 14:54:24 中远海科 封跌停板 12.01元 14:54:24 泰尔股份 大笔卖出 4286手 14:54:17 泰



介绍

效果展示：

The screenshot shows the Chrome DevTools Network tab for a WebSocket connection named "websocket.html". The "Messages" tab is selected, displaying five messages from the server. Each message is a red box containing the text "这是来自服务端的消息: 09:23:35", "09:23:40", "09:23:45", "09:23:50", and "09:23:55" respectively. The "Timing" tab shows a timeline with markers at 10 ms, 20 ms, 30 ms, 40 ms, 50 ms, 60 ms, 70 ms, 80 ms, 90 ms, 100 ms, and 110 ms. The "Headers" tab is also visible. The bottom of the screenshot shows the DevTools footer with tabs for Console, Issues, and Filter.

Data	Length	Time
这是来自服务端的消息: 09:23:35	19	09:23:35....
这是来自服务端的消息: 09:23:40	19	09:23:40....
这是来自服务端的消息: 09:23:45	19	09:23:45....
这是来自服务端的消息: 09:23:50	19	09:23:50....
这是来自服务端的消息: 09:23:55	19	09:23:55....



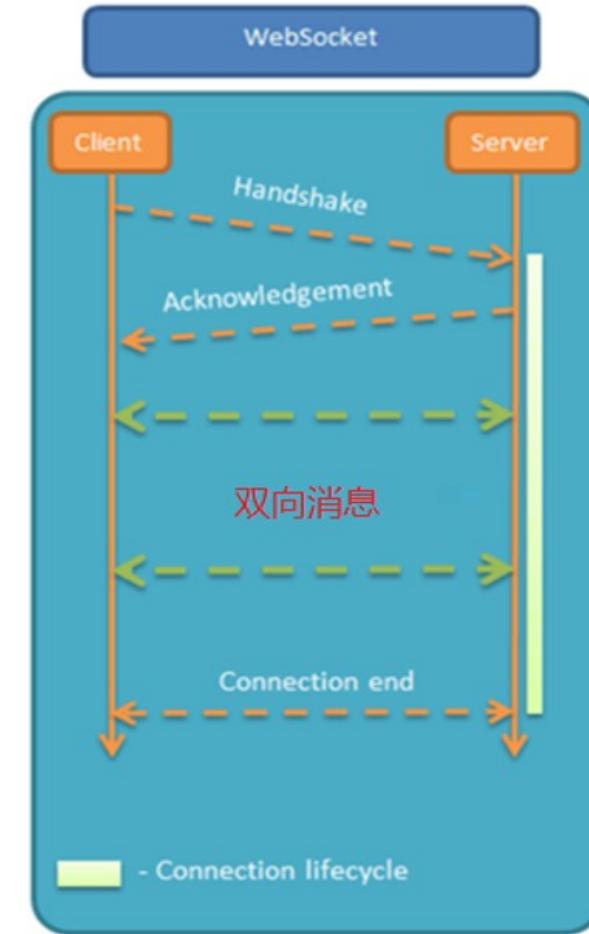
WebSocket

- 介绍
- 入门案例

入门案例

实现步骤：

- ① 直接使用 websocket.html 页面作为 WebSocket 客户端
- ② 导入 WebSocket 的 maven 坐标
- ③ 导入 WebSocket 服务端组件 WebSocketServer，用于和客户端通信
- ④ 导入配置类 WebSocketConfiguration，注册 WebSocket 的服务端组件
- ⑤ 导入定时任务类 WebSocketTask，定时向客户端推送数据





既然 WebSocket 支持双向通信，功能看似比 HTTP 强大，那么我们是不是可以基于 WebSocket 开发所有的业务功能？

WebSocket 缺点：

- 服务器长期维护长连接需要一定的成本
- 各个浏览器支持程度不一
- WebSocket 是长连接，受网络限制比较大，需要处理好重连

结论：WebSocket 并不能完全取代 HTTP，它只适合在特定的场景下使用



目录

Contents

- ◆ Spring Task
- ◆ 订单状态定时处理
- ◆ WebSocket
- ◆ 来单提醒
- ◆ 客户催单



来单提醒

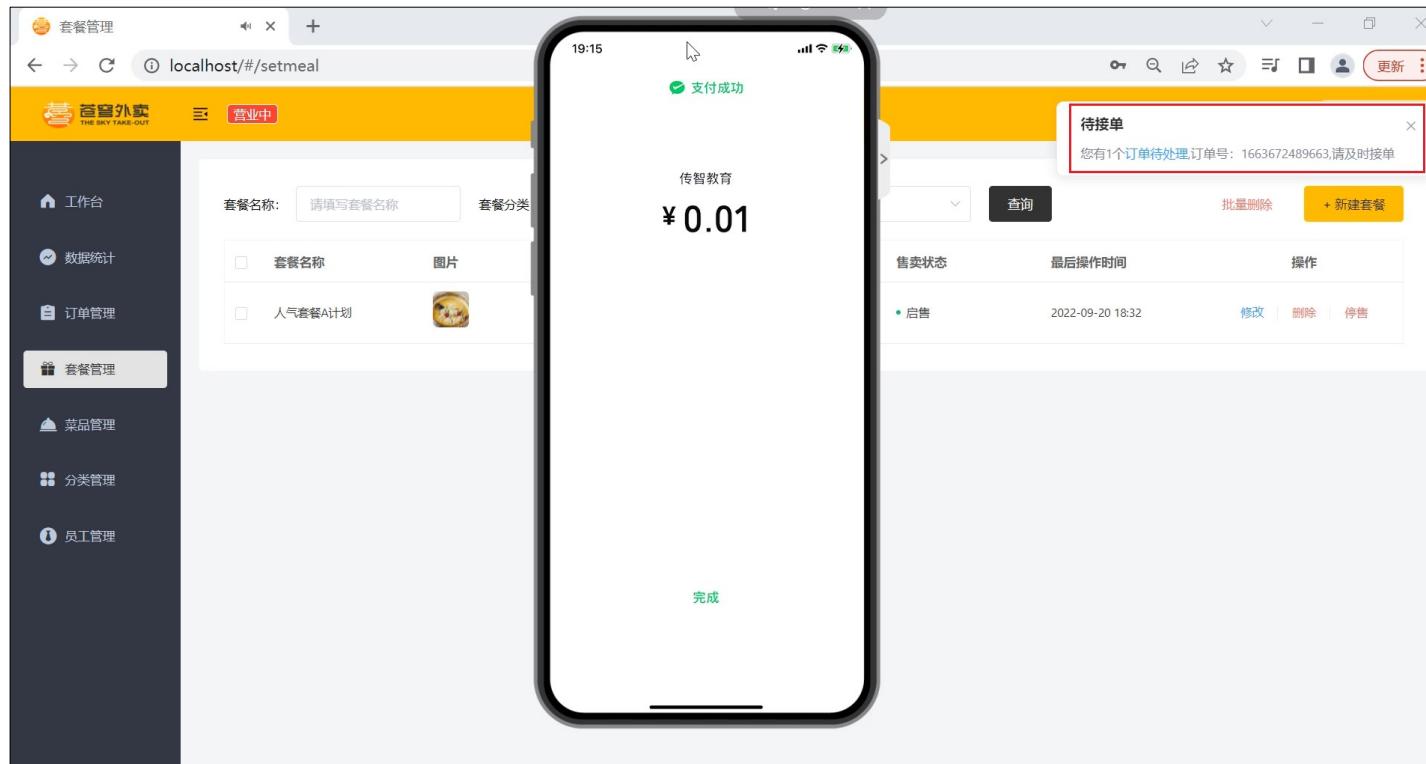
- 需求分析和设计
- 代码开发
- 功能测试



需求分析和设计

用户下单并且支付成功后，需要第一时间通知外卖商家。通知的形式有如下两种：

- 语音播报 
- 弹出提示框



需求分析和设计

设计：

- 通过 WebSocket 实现管理端页面和服务端保持长连接状态
- 当客户支付后，调用 WebSocket 的相关 API 实现服务端向客户端推送消息
- 客户端浏览器解析服务端推送的消息，判断是来单提醒还是客户催单，进行相应的消息提示和语音播报
- 约定服务端发送给客户端浏览器的数据格式为 JSON，字段包括：type，orderId，content
 - type 为消息类型，1 为来单提醒 2 为客户端催单
 - orderId 为订单 id
 - content 为消息内容



来单提醒

- 需求分析和设计
- 代码开发
- 功能测试



代码开发

在 OrderServiceImpl 中注入 WebSocketServer 对象，修改 paySuccess 方法，加入如下代码：

```
Map map = new HashMap();
map.put("type", 1); // 通知类型 1 来单提醒 2 客户催单
map.put("orderId", orders.getId()); // 订单 id
map.put("content", "订单号：" + outTradeNo);

webSocketServer.sendToAllClient(JSON.toJSONString(map));
```



来单提醒

- 需求分析和设计
- 代码开发
- 功能测试

功能测试

可以通过如下方式进行测试：

- 查看浏览器调试工具数据交互过程
- 前后端联调



目录

Contents

- ◆ Spring Task
- ◆ 订单状态定时处理
- ◆ WebSocket
- ◆ 来单提醒
- ◆ 客户催单



客户催单

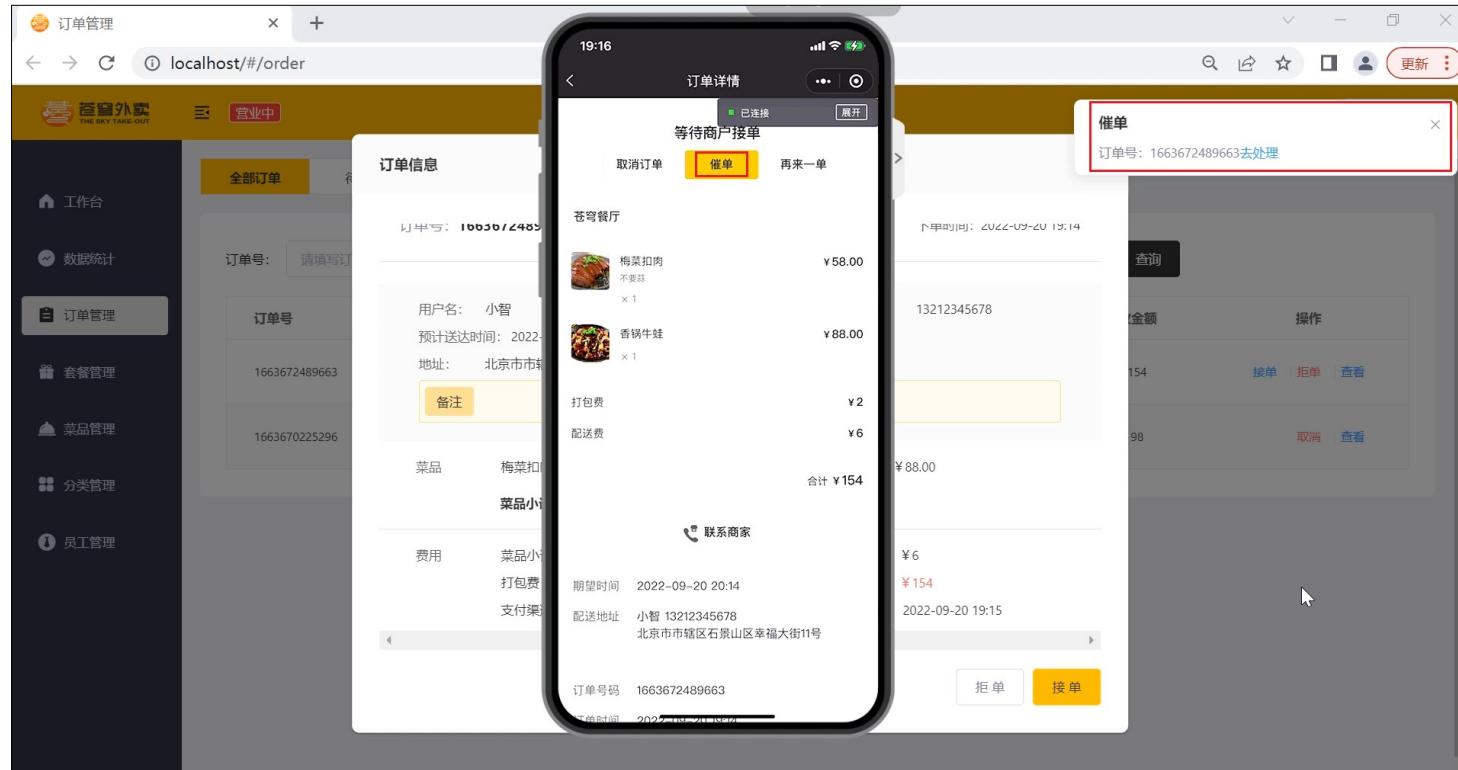
- 需求分析和设计
- 代码开发
- 功能测试



需求分析和设计

用户在小程序中点击催单按钮后，需要第一时间通知外卖商家。通知的形式有如下两种：

- 语音播报 
- 弹出提示框



需求分析和设计

设计：

- 通过 WebSocket 实现管理端页面和服务端保持长连接状态
- 当用户点击催单按钮后，调用 WebSocket 的相关 API 实现服务端向客户端推送消息
- 客户端浏览器解析服务端推送的消息，判断是来单提醒还是客户催单，进行相应的消息提示和语音播报
- 约定服务端发送给客户端浏览器的数据格式为 JSON，字段包括：type，orderId，content
 - type 为消息类型，1 为来单提醒 2 为客户端催单
 - orderId 为订单 id
 - content 为消息内容

需求分析和设计

接口设计：

基本信息

Path: /user/order/reminder/{id}

Method: GET

接口描述：

请求参数

路径参数

参数名称	示例	备注
id	101	订单id

返回数据

名称	类型	是否必须	默认值	备注	其他信息
code	integer	必须			format: int32
data	object	非必须			
msg	string	非必须			



客户催单

- 需求分析和设计
- 代码开发
- 功能测试



代码开发

根据用户催单的接口定义，在 user/OrderController 中创建催单方法：

```
/**  
 * 用户催单  
 *  
 * @param id  
 * @return  
 */  
@GetMapping("/reminder/{id}")  
@ApiOperation("用户催单")  
public Result reminder(@PathVariable("id") Long id) {  
    orderService.reminder(id);  
    return Result.success();  
}
```

代码开发

在 OrderService 接口中声明 reminder 方法：

```
/**  
 * 用户催单  
 * @param id  
 */  
void reminder(Long id);
```



代码开发

在 OrderServiceImpl 中实现 reminder 方法：

```
/*
 * 用户催单
 *
 * @param id
 */
public void reminder(Long id) {
    // 查询订单是否存在
    Orders orders = orderMapper.getById(id);
    if (orders == null) {
        throw new OrderBusinessException(MessageConstant.ORDER_NOT_FOUND);
    }

    // 基于 WebSocket 实现催单
    Map map = new HashMap();
    map.put("type", 2); // 2 代表用户催单
    map.put("orderId", id);
    map.put("content", "订单号：" + orders.getNumber());
    webSocketServer.sendToAllClient(JSON.toJSONString(map));
}
```



客户催单

- 需求分析和设计
- 代码开发
- 功能测试

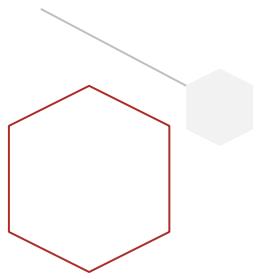
功能测试

可以通过如下方式进行测试：

- 查看浏览器调试工具数据交互过程
- 前后端联调



传智教育旗下高端IT教育品牌



数据统计-图形报表



黑马程序员 | 传智教育旗下
高端IT教育品牌
www.itheima.com



苍穹外卖 THE SKY TAKE-OUT

三 营业中

昨日 近7日 **近30日** 本周 本月 已选时间：2022-05-28 至 2022-06-26

工作台

数据统计

订单管理

套餐管理

菜品管理

分类管理

员工管理

营业额统计

用户统计

订单统计

销量排名TOP10

The screenshot displays a comprehensive dashboard for a restaurant's performance over the past month. It includes four main sections: 1) 营业额统计 (Sales Performance) showing daily revenue peaks around June 18th and 22nd; 2) 用户统计 (User Statistics) showing a steady increase in total users from approximately 5 to 8 million, with a small spike in new users around June 2nd; 3) 订单统计 (Order Statistics) showing a similar trend with order counts peaking on the same dates; 4) 销量排名TOP10 (Top 10 Best-Selling Products) listing items like 草鱼2斤 (10), 江团鱼2斤 (7), 馋嘴牛蛙 (3), 虾鱼2斤 (2), and 金汤酸菜牛蛙 (1). The interface is clean with a yellow header and dark sidebar.



目录

Contents

- ◆ Apache ECharts
- ◆ 营业额统计
- ◆ 用户统计
- ◆ 订单统计
- ◆ 销量排名 Top10



Apache ECharts

- 介绍
- 入门案例



Apache ECharts 介绍

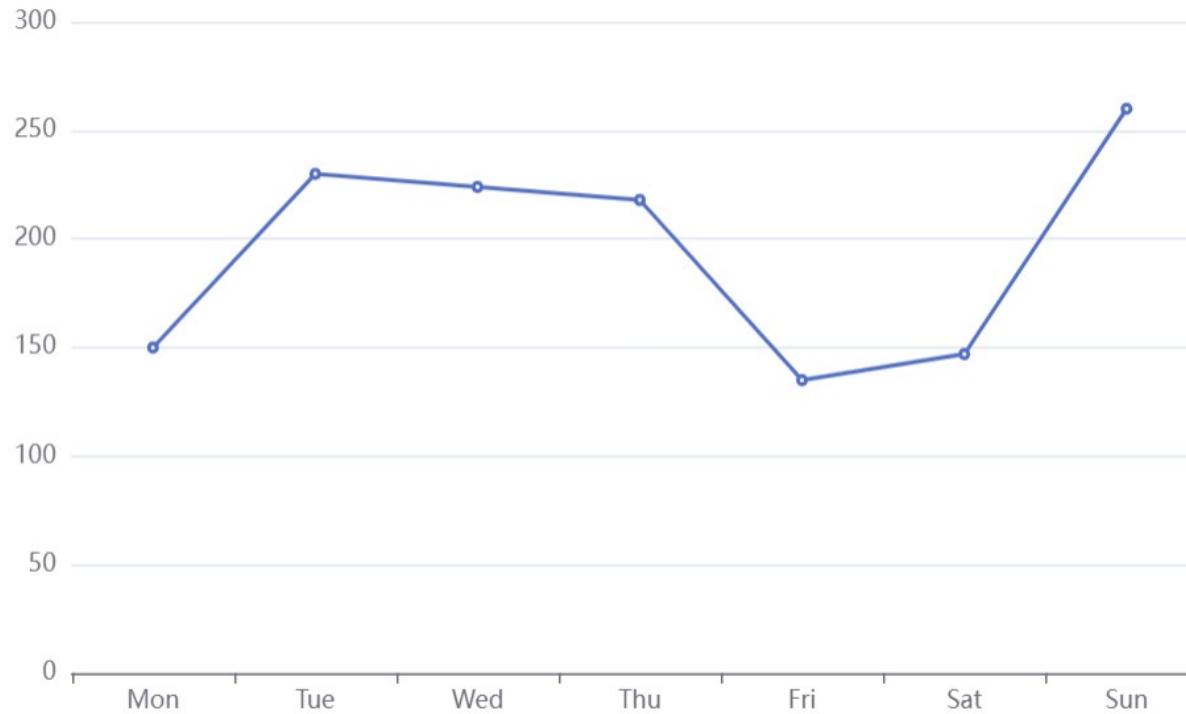
Apache ECharts 是一款基于 Javascript 的数据可视化图表库，提供直观，生动，可交互，可个性化定制的数据可视化图表。

官网地址：<https://echarts.apache.org/zh/index.html>



Apache ECharts 介绍

效果展示：



柱形图
饼形图
折线图

通过直观的图表来展示**数据**



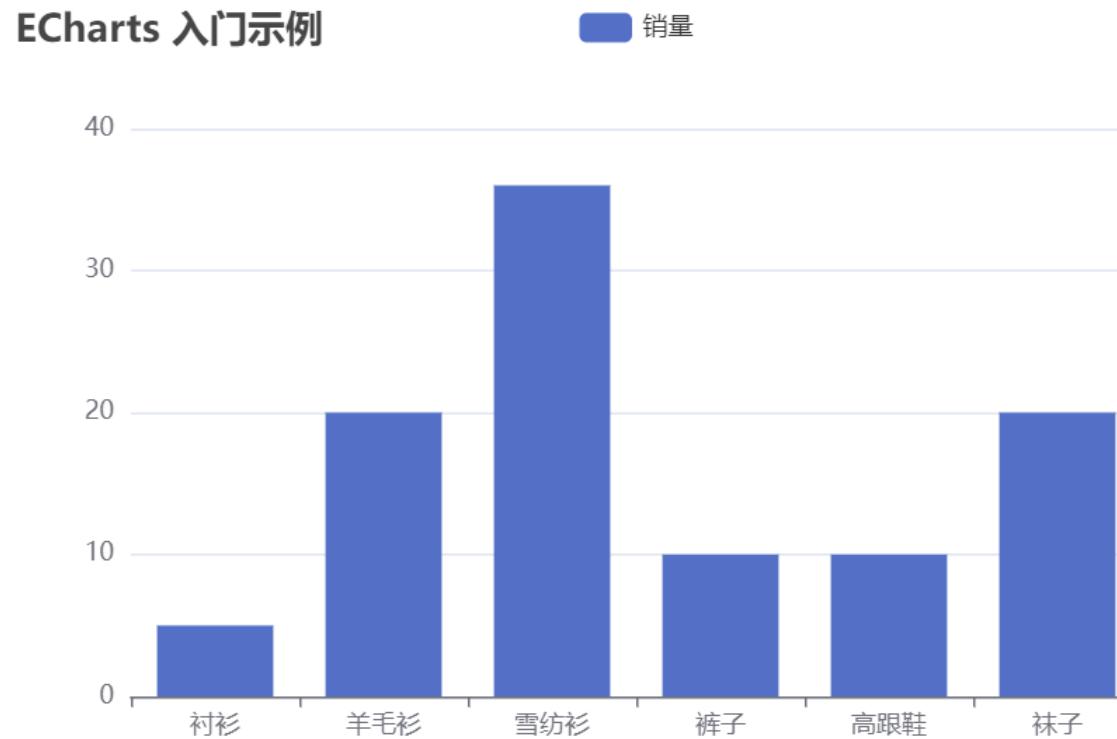
Apache ECharts

- 介绍
- 入门案例



入门案例

Apache Echarts 官方提供的快速入门：<https://echarts.apache.org/handbook/zh/get-started/>





入门案例

总结：使用 Echarts，重点在于研究当前图表所需的数据格式。通常需要后端提供符合格式要求的动态数据，然后响应给前端来展示图表。



目录

Contents

- ◆ Apache ECharts
- ◆ 营业额统计
- ◆ 用户统计
- ◆ 订单统计
- ◆ 销量排名 Top10



营业额统计

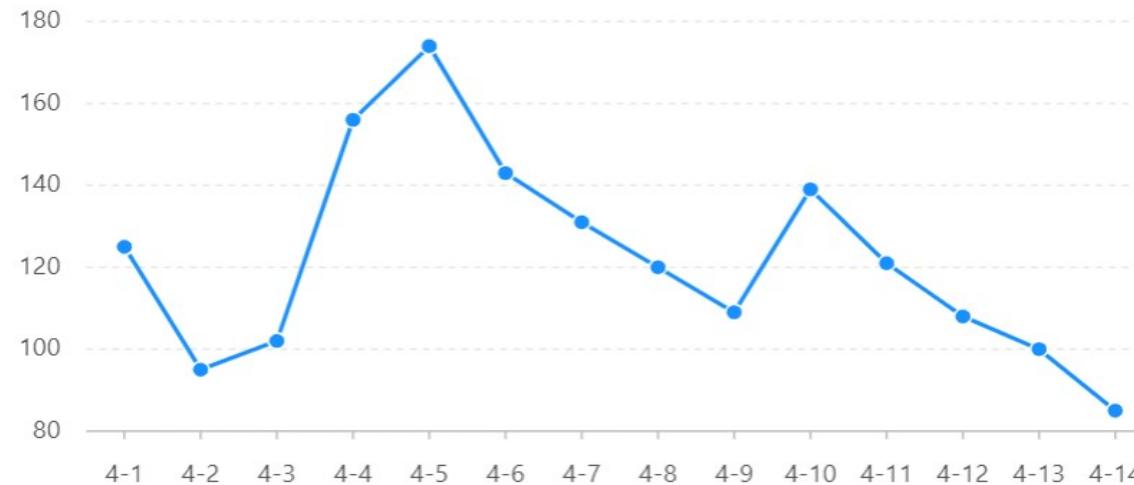
- 需求分析和设计
- 代码开发
- 功能测试

需求分析和设计

产品原型：

昨日 近7日 近30日 本周 本月

营业额统计



业务规则：

- 营业额指订单状态为已完成的订单金额合计
- 基于可视化报表的折线图展示营业额数据，X轴为日期，Y轴为营业额
- 根据时间选择区间，展示每天的营业额数据



需求分析和设计

接口设计：

基本信息

Path: /admin/report/turnoverStatistics

Method: GET

接口描述：

返回数据

请求参数

Query

参数名称	是否必须	示例	备注
begin	是	2022-05-01	开始日期
end	是	2022-05-31	结束日期

名称	类型	是否必须	默认值	备注	其他信息
code	integer	必须			format: int32
data	object	必须			
	dateList	string	必须	日期列表，日期之间以逗号分隔	
	turnoverList	string	必须	营业额列表，营业额之间以逗号分隔	
msg	string	非必须			



营业额统计

- 需求分析和设计
- 代码开发
- 功能测试



代码开发

根据接口定义设计对应的 VO：

返回数据

名称	类型	是否必须	默认值	备注
code	integer	必须		
data	object	必须		
└ dateList	string	必须		日期列表，日期之间以逗号分隔
└ turnoverList	string	必须		营业额列表，营业额之间以逗号分隔
msg	string	非必须		



```
public class TurnoverReportVO implements Serializable {  
  
    //日期，以逗号分隔，例如：2022-10-01,2022-10-02,2022-10-03  
    private String dateList;  
  
    //营业额，以逗号分隔，例如：406.0,1520.0,75.0  
    private String turnoverList;  
  
}
```



代码开发

根据接口定义创建 ReportController：

```
@RestController
@RequestMapping("/admin/report")
@Slf4j
@Api(tags = "统计报表相关接口")
public class ReportController {
    @Autowired
    private ReportService reportService;

    /**
     * 营业额数据统计
     * @param begin
     * @param end
     * @return
     */
    @GetMapping("/turnoverStatistics")
    @ApiOperation("营业额数据统计")
    public Result<TurnoverReportVO> turnoverStatistics(
        @DateTimeFormat(pattern = "yyyy-MM-dd") LocalDate begin,
        @DateTimeFormat(pattern = "yyyy-MM-dd") LocalDate end){

        return Result.success(reportService.getTurnover(begin,end));
    }
}
```



代码开发

创建 ReportService 接口，声明 getTurnover 方法：

```
public interface ReportService {  
  
    /**  
     * 根据时间区间统计营业额  
     * @param begin  
     * @param end  
     * @return  
     */  
    TurnoverReportVO getTurnover(LocalDateTime begin, LocalDateTime end);  
}
```



代码开发

创建 ReportServiceImpl 实现类，实现 getTurnover 方法（第 1 部分）：

```
@Service
@Slf4j
public class ReportServiceImpl implements ReportService {

    @Autowired
    private OrderMapper orderMapper;

    /**
     * 根据时间区间统计营业额
     * @param begin
     * @param end
     * @return
     */
    public TurnoverReportVO getTurnover(LocalDate begin, LocalDate end) {
        List<LocalDate> dateList = new ArrayList<>();
        dateList.add(begin);

        while (!begin.equals(end)){
            begin = begin.plusDays(1); // 日期计算，获得指定日期后 1 天的日期
            dateList.add(begin);
        }
    }
}
```



代码开发

创建 ReportServiceImpl 实现类，实现 getTurnover 方法（第 2 部分）：

```
List<Double> turnoverList = new ArrayList<>();
for (LocalDate date : dateList) {
    LocalDateTime beginTime = LocalDateTime.of(date, LocalTime.MIN);
    LocalDateTime endTime = LocalDateTime.of(date, LocalTime.MAX);
    // 查询营业额
    Map map = new HashMap();
    map.put("status", Orders.COMPLETED);
    map.put("begin", beginTime);
    map.put("end", endTime);
    Double turnover = orderMapper.sumByMap(map);
    turnover = turnover == null ? 0.0 : turnover;
    turnoverList.add(turnover);
}

// 数据封装
return TurnoverReportVO.builder()
    .dateList(StringUtils.join(dateList, ","))
    .turnoverList(StringUtils.join(turnoverList, ","))
    .build();
}
```



代码开发

在 OrderMapper 接口声明 sumByMap 方法：

```
/**  
 * 根据动态条件统计营业额  
 * @param map  
 */  
Double sumByMap(Map map);
```



代码开发

在 OrderMapper.xml 文件中编写动态 SQL：

```
<select id="sumByMap" resultType="java.lang.Double">
    select sum(amount) from orders
    <where>
        <if test="status != null">
            and status = #{status}
        </if>
        <if test="begin != null">
            and order_time &gt;= #{begin}
        </if>
        <if test="end != null">
            and order_time &lt;= #{end}
        </if>
    </where>
</select>
```



营业额统计

- 需求分析和设计
- 代码开发
- 功能测试

功能测试

可以通过如下方式进行测试：

- 接口文档测试
- 前后端联调测试

Name	Headers	Payload	Preview	Response	Initiator	Timing	Cookies	
<input type="checkbox"/> turnoverStatistics?begin=2022-06-20&end=2022-06-26 <input type="checkbox"/> userStatistics?begin=2022-06-20&end=2022-06-26 <input type="checkbox"/> ordersStatistics?begin=2022-06-20&end=2022-06-26 <input type="checkbox"/> top10?begin=2022-06-20&end=2022-06-26				▼ {code: 1, msg: null,...} code: 1 ▼ data: {dateList: "2022-06-20,2022-06-21,2022-06-22,2022-06-23,2022-06-24,2022-06-25,2022-06-26",...} dateList: "2022-06-20,2022-06-21,2022-06-22,2022-06-23,2022-06-24,2022-06-25,2022-06-26" turnoverList: "406.0,825.0,75.0,695.0,126.0,0.0,0.0" msg: null				



目录

Contents

- ◆ Apache ECharts
- ◆ 营业额统计
- ◆ 用户统计
- ◆ 订单统计
- ◆ 销量排名 Top10

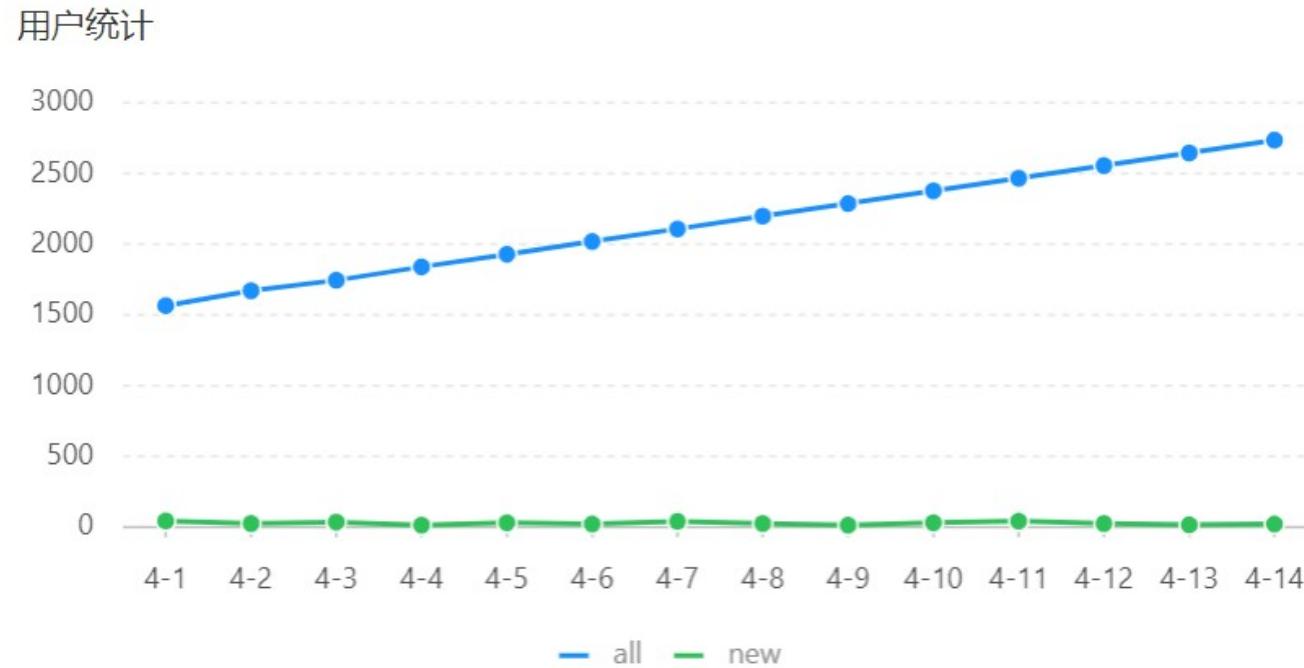


用户统计

- 需求分析和设计
- 代码开发
- 功能测试

需求分析和设计

产品原型：



业务规则：

- 基于可视化报表的折线图展示用户数据，X轴为日期，Y轴为用户数
- 根据时间选择区间，展示每天的用户总量和新增用户量数据



需求分析和设计

接口设计：

基本信息

Path: /admin/report/userStatistics

Method: GET

接口描述：

请求参数

Query

参数名称	是否必须	示例	备注
begin	是	2022-05-01	开始日期
end	是	2022-05-31	结束日期

返回数据

名称	类型	是否必须	默认值	备注	其他信息
code	integer	必须			format: int32
data	object	必须			
	└ dateList	string	必须	日期列表，以逗号分隔	
	└ newUserList	string	必须	新增用户数列表，以逗号分隔	
	└ totalUserList	string	必须	总用户量列表，以逗号分隔	
msg	string	非必须			



用户统计

- 需求分析和设计
- 代码开发
- 功能测试



代码开发

根据用户统计接口的返回结果设计 VO：

返回数据

名称	类型	是否必须	默认值	备注
code	integer	必须		
data	object	必须		
└ dateList	string	必须		日期列表，以逗号分隔
└ newUserList	string	必须		新增用户数列表，以逗号分隔
└ totalUserList	string	必须		总用户量列表，以逗号分隔
msg	string	非必须		



```
public class UserReportVO implements Serializable {  
  
    //日期，以逗号分隔，例如：2022-10-01,2022-10-02,2022-10-03  
    private String dateList;  
  
    //用户总量，以逗号分隔，例如：200,210,220  
    private String totalUserList;  
  
    //新增用户，以逗号分隔，例如：20,21,10  
    private String newUserList;  
}
```

代码开发

根据接口定义，在 ReportController 中创建 userStatistics 方法：

```
/**  
 * 用户数据统计  
 * @param begin  
 * @param end  
 * @return  
 */  
@GetMapping("/userStatistics")  
@ApiOperation("用户数据统计")  
public Result<UserReportVO> userStatistics(  
    @DateTimeFormat(pattern = "yyyy-MM-dd") LocalDate begin,  
    @DateTimeFormat(pattern = "yyyy-MM-dd") LocalDate end){  
  
    return Result.success(reportService.getUserStatistics(begin,end));  
}
```

代码开发

在 ReportService 接口中声明 getUserStatistics 方法：

```
/**  
 * 根据时间区间统计用户数量  
 * @param begin  
 * @param end  
 * @return  
 */  
UserReportVO getUserStatistics(LocalDate begin, LocalDate end);
```



代码开发

在 ReportServiceImpl 实现类中实现 getUserStatistics 方法（第 1 部分）：

```
/*
 * 根据时间区间统计用户数量
 * @param begin
 * @param end
 * @return
 */
public UserReportVO getUserStatistics(LocalDate begin, LocalDate end) {
    List<LocalDate> dateList = new ArrayList<>();
    dateList.add(begin);

    while (!begin.equals(end)){
        begin = begin.plusDays(1);
        dateList.add(begin);
    }

    List<Integer> newUserList = new ArrayList<>(); // 新增用户数
    List<Integer> totalUserList = new ArrayList<>(); // 总用户数
```



代码开发

在 ReportServiceImpl 实现类中实现 getUserStatistics 方法（第 2 部分）：

```
for (LocalDate date : dateList) {
    LocalDateTime beginTime = LocalDateTime.of(date, LocalTime.MIN);
    LocalDateTime endTime = LocalDateTime.of(date, LocalTime.MAX);
    // 新增用户数量 select count(id) from user where create_time > ? and create_time < ?
    Integer newUser = getUserCount(beginTime, endTime);
    // 总用户数量 select count(id) from user where create_time < ?
    Integer totalUser = getUserCount(null, endTime);

    newUserList.add(newUser);
    totalUserList.add(totalUser);
}

return UserReportVO.builder()
    .dateList(StringUtils.join(dateList, ","))
    .newUserList(StringUtils.join(newUserList, ","))
    .totalUserList(StringUtils.join(totalUserList, ","))
    .build();
}
```



代码开发

在 ReportServiceImpl 实现类中创建私有方法 getUserCount :

```
/*
 * 根据时间区间统计用户数量
 * @param beginTime
 * @param endTime
 * @return
 */
private Integer getUserCount(LocalDateTime beginTime, LocalDateTime endTime) {
    Map map = new HashMap();
    map.put("begin", beginTime);
    map.put("end", endTime);
    return userMapper.countByMap(map);
}
```

代码开发

在 UserMapper 接口中声明 countByMap 方法：

```
/**  
 * 根据动态条件统计用户数量  
 * @param map  
 * @return  
 */  
Integer countByMap(Map map);
```



代码开发

在 UserMapper.xml 文件中编写动态 SQL：

```
<select id="countByMap" resultType="java.lang.Integer">
    select count(id) from user
    <where>
        <if test="begin != null">
            and create_time &gt;= #{begin}
        </if>
        <if test="end != null">
            and create_time &lt;= #{end}
        </if>
    </where>
</select>
```



用户统计

- 需求分析和设计
- 代码开发
- 功能测试

功能测试

可以通过如下方式进行测试：

- 接口文档测试
- 前后端联调测试

Name	Headers	Payload	Preview	Response	Initiator	Timing	Cookies
<input type="checkbox"/> status							
<input type="checkbox"/> turnoverStatistics?begin=2022-06-20&end=2022-06-26							
<input type="checkbox"/> userStatistics?begin=2022-06-20&end=2022-06-26							
<input type="checkbox"/> ordersStatistics?begin=2022-06-20&end=2022-06-26							
<input type="checkbox"/> top10?begin=2022-06-20&end=2022-06-26							

Preview tab is selected.

```
▼ {code: 1, msg: null,...}
  code: 1
  ▼ data: {dateList: "2022-06-20,2022-06-21,2022-06-22,2022-06-23,2022-06-24,2022-06-25,2022-06-26",...}
    dateList: "2022-06-20,2022-06-21,2022-06-22,2022-06-23,2022-06-24,2022-06-25,2022-06-26"
    newUserList: "0,0,1,0,0,0,0"
    totalUserList: "7,7,8,8,8,8,8"
  msg: null
```



目录

Contents

- ◆ Apache ECharts
- ◆ 营业额统计
- ◆ 用户统计
- ◆ 订单统计
- ◆ 销量排名 Top10



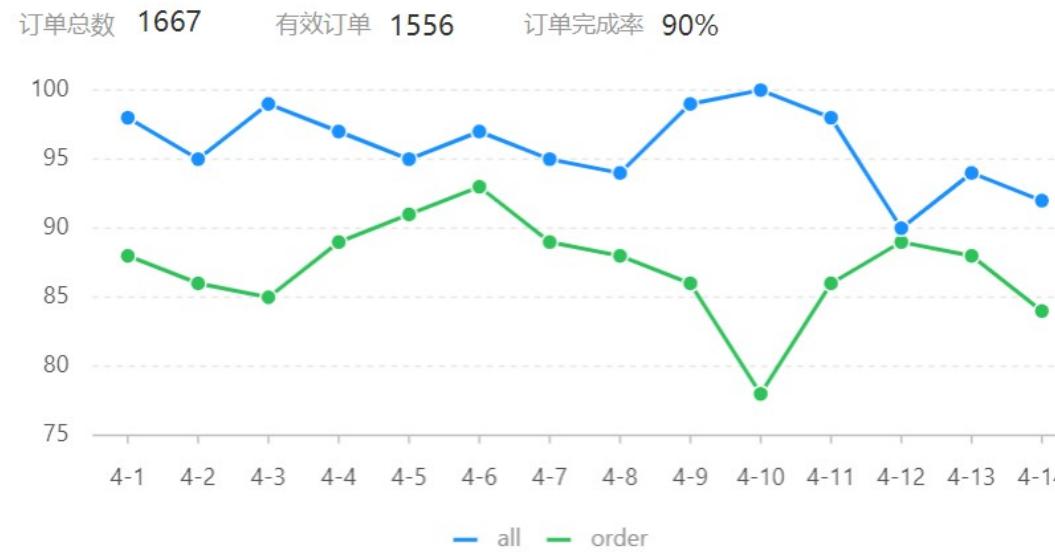
订单统计

- 需求分析和设计
- 代码开发
- 功能测试

需求分析和设计

产品原型：

订单统计



业务规则：

- 有效订单指状态为“已完成”的订单
- 基于可视化报表的折线图展示订单数据，X 轴为日期，Y 轴为订单数量
- 根据时间选择区间，展示每天的订单总数和有效订单数
- 展示所选时间区间内的有效订单数、总订单数、订单完成率， $\text{订单完成率} = \text{有效订单数} / \text{总订单数} * 100\%$



需求分析和设计

接口设计：

基本信息

Path: /admin/report/ordersStatistics

Method: GET

接口描述：

请求参数

Query

参数名称	是否必须	示例	备注
begin	是	2022-05-01	开始日期
end	是	2022-05-31	结束日期

返回数据

名称	类型	是否必须	默认值	备注	其他信息
code	integer	必须			format: int32
data	object	必须			
— dateList	string	必须		日期列表，以逗号分隔	
— orderCompletionRate	number	必须		订单完成率	format: double
— orderCountList	string	必须		订单数列表，以逗号分隔	
— totalOrderCount	integer	必须		订单总数	format: int32
— validOrderCount	integer	必须		有效订单数	format: int32
— validOrderCountList	string	必须		有效订单数列表，以逗号分隔	
msg	string	非必须			



订单统计

- 需求分析和设计
- 代码开发
- 功能测试



代码开发

根据订单统计接口的返回结果设计 VO：

返回数据

名称	类型	是否必须	默认值	备注
code	integer	必须		
data	object	必须		
├─dateList	string	必须		日期列表，以逗号分隔
├─orderCompletionRate	number	必须		订单完成率
├─orderCountList	string	必须		订单数列表，以逗号分隔
├─totalOrderCount	integer	必须		订单总数
├─validOrderCount	integer	必须		有效订单数
├─validOrderCountList	string	必须		有效订单数列表，以逗号分隔
msg	string	非必须		



```
public class OrderReportVO implements Serializable {  
  
    //日期，以逗号分隔，例如：2022-10-01,2022-10-02,2022-10-03  
    private String dateList;  
  
    //每日订单数，以逗号分隔，例如：260,210,215  
    private String orderCountList;  
  
    //每日有效订单数，以逗号分隔，例如：20,21,10  
    private String validOrderCountList;  
  
    //订单总数  
    private Integer totalOrderCount;  
  
    //有效订单数  
    private Integer validOrderCount;  
  
    //订单完成率  
    private Double orderCompletionRate;  
  
}
```



代码开发

在 ReportController 中根据订单统计接口创建 orderStatistics 方法：

```
/*
 * 订单数据统计
 * @param begin
 * @param end
 * @return
 */
@GetMapping("/ordersStatistics")
@ApiOperation("订单数据统计")
public Result<OrderReportVO> orderStatistics(
    @DateTimeFormat(pattern = "yyyy-MM-dd") LocalDate begin,
    @DateTimeFormat(pattern = "yyyy-MM-dd") LocalDate end){

    return Result.success(reportService.getOrderStatistics(begin,end));
}
```

代码开发

在 ReportService 接口中声明 getOrderStatistics 方法：

```
/**  
 * 根据时间区间统计订单数量  
 * @param begin  
 * @param end  
 * @return  
 */  
OrderReportVO getOrderStatistics(LocalDate begin, LocalDate end);
```



代码开发

在 ReportServiceImpl 实现类中实现 getOrderStatistics 方法（第 1 部分）：

```
/**  
 * 根据时间区间统计订单数量  
 * @param begin  
 * @param end  
 * @return  
 */  
public OrderReportVO getOrderStatistics(LocalDate begin, LocalDate end) {  
    List<LocalDate> dateList = new ArrayList<>();  
    dateList.add(begin);  
  
    while (!begin.equals(end)){  
        begin = begin.plusDays(1);  
        dateList.add(begin);  
    }  
  
    // 每天订单总数集合  
    List<Integer> orderCountList = new ArrayList<>();  
    // 每天有效订单数集合  
    List<Integer> validOrderCountList = new ArrayList<>();
```



代码开发

在 ReportServiceImpl 实现类中实现 getOrderStatistics 方法（第 2 部分）：

```
for (LocalDate date : dateList) {
    LocalDateTime beginTime = LocalDateTime.of(date, LocalTime.MIN);
    LocalDateTime endTime = LocalDateTime.of(date, LocalTime.MAX);
    // 查询每天的总订单数 select count(id) from orders where order_time > ? and order_time < ?
    Integer orderCount = getOrderByCount(beginTime, endTime, null);

    // 查询每天的有效订单数 select count(id) from orders where order_time > ? and order_time < ? and status = ?
    Integer validOrderCount = getOrderByCount(beginTime, endTime, Orders.COMPLETED);

    orderCountList.add(orderCount);
    validOrderCountList.add(validOrderCount);
}

// 时间区间内的总订单数
Integer totalOrderCount = orderCountList.stream().reduce(Integer::sum).get();
// 时间区间内的总有效订单数
Integer validOrderCount = validOrderCountList.stream().reduce(Integer::sum).get();
```



代码开发

在 ReportServiceImpl 实现类中实现 getOrderStatistics 方法（第 3 部分）：

```
// 订单完成率
Double orderCompletionRate = 0.0;
if(totalOrderCount != 0){
    orderCompletionRate = validOrderCount.doubleValue() / totalOrderCount;
}

return OrderReportVO.builder()
    .dateList(StringUtils.join(dateList, ","))
    .orderCountList(StringUtils.join(orderCountList, ","))
    .validOrderCountList(StringUtils.join(validOrderCountList, ","))
    .totalOrderCount(totalOrderCount)
    .validOrderCount(validOrderCount)
    .orderCompletionRate(orderCompletionRate)
    .build();
}
```

代码开发

在 ReportServiceImpl 实现类中提供私有方法 getCountOrderCount：

```
/*
 * 根据时间区间统计指定状态的订单数量
 * @param beginTime
 * @param endTime
 * @param status
 * @return
 */
private Integer getCountOrderCount(LocalDateTime beginTime, LocalDateTime endTime, Integer status) {
    Map map = new HashMap();
    map.put("status", status);
    map.put("begin", beginTime);
    map.put("end", endTime);
    return orderMapper.countByMap(map);
}
```



代码开发

在 OrderMapper 接口中声明 countByMap 方法：

```
/**  
 * 根据动态条件统计订单数量  
 * @param map  
 */  
Integer countByMap(Map map);
```

代码开发

在 OrderMapper.xml 文件中编写动态 SQL：

```
<select id="countByMap" resultType="java.lang.Integer">
    select count(id) from orders
    <where>
        <if test="status != null">
            and status = #{status}
        </if>
        <if test="begin != null">
            and order_time &gt;= #{begin}
        </if>
        <if test="end != null">
            and order_time &lt;= #{end}
        </if>
    </where>
</select>
```



订单统计

- 需求分析和设计
- 代码开发
- 功能测试

功能测试

可以通过如下方式进行测试：

- 接口文档测试
- 前后端联调

Name	Headers	Payload	Preview	Response	Initiator	Timing	Cookies
<input type="checkbox"/> status <input type="checkbox"/> turnoverStatistics?begin=2022-06-20&end=2022-06-26 <input type="checkbox"/> userStatistics?begin=2022-06-20&end=2022-06-26 <input type="checkbox"/> ordersStatistics?begin=2022-06-20&end=2022-06-26 <input type="checkbox"/> top10?begin=2022-06-20&end=2022-06-26		x {code: 1, msg: null,...} code: 1 ▼ data: {dateList: "2022-06-20,2022-06-21,2022-06-22,2022-06-23,2022-06-24,2022-06-25,2022-06-26",...} dateList: "2022-06-20,2022-06-21,2022-06-22,2022-06-23,2022-06-24,2022-06-25,2022-06-26" orderCompletionRate: 0.8235294117647058 orderCountList: "5,8,1,2,1,0,0" totalOrderCount: 17 validOrderCount: 14 validOrderCountList: "4,7,1,1,1,0,0" msg: null					



目录

Contents

- ◆ Apache ECharts
- ◆ 营业额统计
- ◆ 用户统计
- ◆ 订单统计
- ◆ 销量排名 Top10



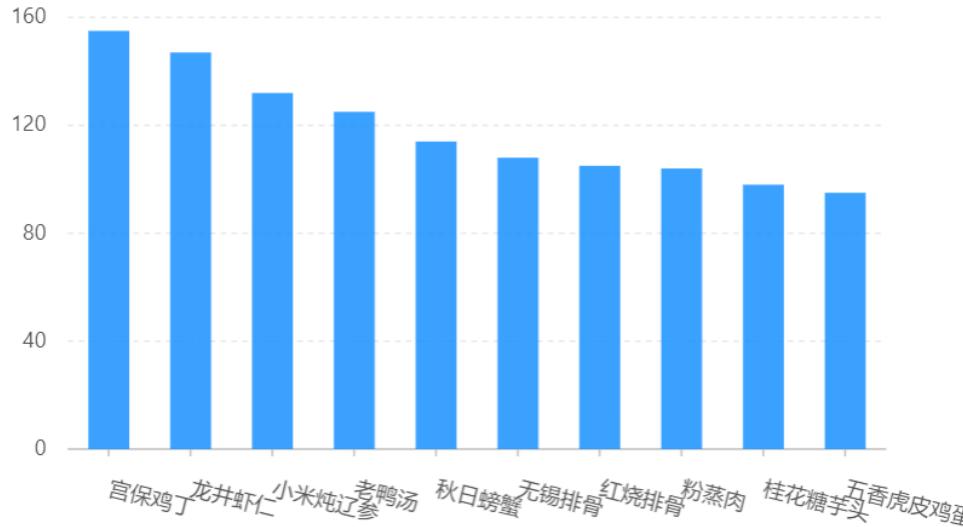
销量排名 Top10

- 需求分析和设计
- 代码开发
- 功能测试

需求分析和设计

产品原型：

销量排名TOP10



业务规则：

- 根据时间选择区间，展示销量前 10 的商品（包括菜品和套餐）
- 基于可视化报表的柱状图降序展示商品销量
- 此处的销量为商品销售的份数



需求分析和设计

接口设计：

基本信息

Path: /admin/report/top10

Method: GET

接口描述：

请求参数

Query

参数名称	是否必须	示例	备注
begin	是	2022-05-01	开始日期
end	是	2022-05-31	结束日期

返回数据

名称	类型	是否必须	默认值	备注	其他信息
code	integer	必须			format: int32
data	object	必须			
	nameList	string	必须	商品名称列表，以逗号分隔	
	numberList	string	必须	销量列表，以逗号分隔	
msg	string	非必须			



销量排名 Top10

- 需求分析和设计
- 代码开发
- 功能测试



代码开发

根据销量排名接口的返回结果设计 VO：

返回数据

名称	类型	是否必须	默认值	备注
code	integer	必须		
data	object	必须		
└ nameList	string	必须		商品名称列表，以逗号分隔
└ numberList	string	必须		销量列表，以逗号分隔
msg	string	非必须		



```
public class SalesTop10ReportVO implements Serializable {  
  
    //商品名称列表，以逗号分隔，例如：鱼香肉丝,宫保鸡丁,水煮鱼  
    private String nameList;  
  
    //销量列表，以逗号分隔，例如：260,215,200  
    private String numberList;  
}
```

代码开发

在 ReportController 中根据销量排名接口创建 top10 方法：

```
/**  
 * 销量排名统计  
 * @param begin  
 * @param end  
 * @return  
 */  
@GetMapping("/top10")  
@ApiOperation("销量排名统计")  
public Result<SalesTop10ReportVO> top10(  
    @DateTimeFormat(pattern = "yyyy-MM-dd") LocalDate begin,  
    @DateTimeFormat(pattern = "yyyy-MM-dd") LocalDate end){  
  
    return Result.success(reportService.getSalesTop10(begin,end));  
}
```



代码开发

在 ReportService 接口中声明 getSalesTop10 方法：

```
/*
 * 查询指定时间区间内的销量排名 top10
 * @param begin
 * @param end
 * @return
 */
SalesTop10ReportVO getSalesTop10(LocalDate begin, LocalDate end);
```



代码开发

在 ReportServiceImpl 实现类中实现 getSalesTop10 方法：

```
/*
 * 查询指定时间区间内的销量排名 top10
 * @param begin
 * @param end
 * @return
 */
public SalesTop10ReportVO getSalesTop10(LocalDate begin, LocalDate end) {
    LocalDateTime beginTime = LocalDateTime.of(begin, LocalTime.MIN);
    LocalDateTime endTime = LocalDateTime.of(end, LocalTime.MAX);
    List<GoodsSalesDTO> goodsSalesDTOList = orderMapper.getSalesTop10(beginTime, endTime);

    String nameList = StringUtils.join(goodsSalesDTOList.stream().map(GoodsSalesDTO::getName).collect(Collectors.toList()), ", ");
    String numberList = StringUtils.join(goodsSalesDTOList.stream().map(GoodsSalesDTO::getNumber).collect(Collectors.toList()), ", ");

    return SalesTop10ReportVO.builder()
        .nameList(nameList)
        .numberList(numberList)
        .build();
}
```



代码开发

在 OrderMapper 接口中声明 getSalesTop10 方法：

```
/**  
 * 查询商品销量排名  
 * @param begin  
 * @param end  
 */  
List<GoodsSalesDTO> getSalesTop10(LocalDateTime begin, LocalDateTime end);
```



代码开发

在 OrderMapper.xml 文件中编写动态 SQL：

```
<select id="getSalesTop10" resultType="com.sky.dto.GoodsSalesDTO">
    select od.name name,sum(od.number) number from order_detail od ,orders o
    where od.order_id = o.id
        and o.status = 5
        <if test="begin != null">
            and order_time &gt;= #{begin}
        </if>
        <if test="end != null">
            and order_time &lt;= #{end}
        </if>
    group by name
    order by number desc
    limit 0, 10
</select>
```



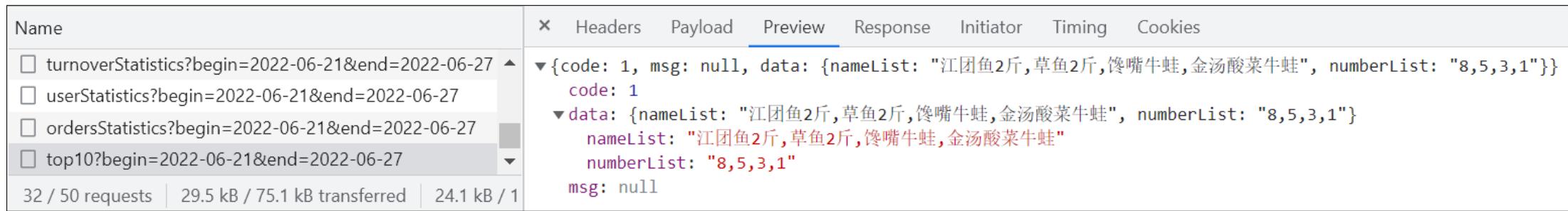
销量排名 Top10

- 需求分析和设计
- 代码开发
- 功能测试

功能测试

可以通过如下方式进行测试：

- 接口文档测试
- 前后端联调



Name	Headers	Payload	Preview	Response	Initiator	Timing	Cookies
turnoverStatistics?begin=2022-06-21&end=2022-06-27				code: 1, msg: null, data: {nameList: "江团鱼2斤,草鱼2斤,馋嘴牛蛙,金汤酸菜牛蛙", numberList: "8,5,3,1"} code: 1			
userStatistics?begin=2022-06-21&end=2022-06-27							
ordersStatistics?begin=2022-06-21&end=2022-06-27							
top10?begin=2022-06-21&end=2022-06-27							

32 / 50 requests | 29.5 kB / 75.1 kB transferred | 24.1 kB / 1



传智教育旗下高端IT教育品牌



数据统计- Excel 报表



苍穹外卖 THE SKY TAKE-OUT

三 营业中

工作台

数据统计

订单管理

套餐管理

菜品管理

分类管理

员工管理

今日数据 2022.09.20

营业额 ￥ 98

有效订单 1

订单完成率 100%

平均客单价 ￥ 98

新增用户 1

订单管理 2022.09.20

待接单 0 待派送 0 已完成 1 已取消 0 全部订单 1

订单明细 >

菜品总览

菜品管理 >

已启售 24 已停售 1 新增菜品

套餐总览

套餐管理 >

已启售 1 已停售 0 新增套餐

订单信息

待接单 待派送





目录

Contents

- ◆ 工作台
- ◆ Apache POI
- ◆ 导出运营数据 Excel 报表



工作台

- 需求分析和设计
- 代码导入
- 功能测试



需求分析和设计

工作台是系统运营的数据看板，并提供快捷操作入口，可以有效提高商家的工作效率。

工作台展示的数据：

- 今日数据
- 订单管理
- 菜品总览
- 套餐总览
- 订单信息

The screenshot shows the '苍穹外卖LOGO' workbench interface. At the top, it displays '营业中' (Open) and various daily statistics: 营业额 ￥200.13, 有效订单 12, 订单完成率 16%, 平均客单价 30, and 新增用户 18. Below this, there are sections for '订单管理' (Order Management) with counts for 待接单 (12), 待派送 (10), 已完成 (18), 已取消 (1), and 全部订单 (41); and '菜品/套餐总览' (Menu/Combination Overview) with counts for 已启动 (12) and 已停售 (2). The main area is titled '订单信息' (Order Details) and lists five recent orders with columns for 订单号 (Order ID), 订单菜品 (Order Items), 地址 (Address), 预计送达时间 (Estimated Delivery Time), 实收金额 (Actual Amount), 备注 (Notes), and 操作 (Actions). Each order row includes a '查看' (View) button and three operation buttons: 接单 (Accept), 拒单 (Reject), and 查看 (View). The bottom of the page features a navigation bar with icons for back, forward, search, and other system functions.

需求分析和设计

名词解释：

- 营业额：已完成订单的总金额
- 有效订单：已完成订单的数量
- 订单完成率：有效订单数 / 总订单数 * 100%
- 平均客单价：营业额 / 有效订单数
- 新增用户：新增用户的数量



需求分析和设计

接口设计：

- 今日数据接口
- 订单管理接口
- 菜品总览接口
- 套餐总览接口
- 订单搜索（已完成）
- 各个状态的订单数量统计（已完成）

苍穹外卖LOGO

工作中

今日数据 (2022年4月22日) 4

营业额 ￥200.13 | 有效订单 12 | 订单完成率 16% | 平均客单价 30 | 新增用户 18

订单管理 5 | 查看订单明细 >

待接单 12 | 待派送 10 | 已完成 18 | 已取消 1 | 全部订单 41

菜品/套餐总览 6 | 查看菜品管理 >

已启售 12 | 已停售 2

新增菜品

查看套餐管理 >

已启售 12 | 已停售 2

新增套餐

订单信息 7 | 待接单 (12) | 待派送 (10)

订单号	订单菜品	地址	预计送达时间	实收金额	备注	操作
2021010200001	宫保鸡丁* 3; 红烧带鱼* 2; 农家小炒肉* 1;	金燕龙办公楼 (建材城西路9号) 四层 (——宾馆北侧办公楼)	2021-01-02 11:11:11	40.00	不要香菜	接单 拒单 查看
2021010200001	宫保鸡丁* 3; 红烧带鱼* 2; 农家小炒肉* 1;	金燕龙办公楼 (建材城西路9号) 四层 (——宾馆北侧办公楼)	2021-01-02 11:11:11	40.00	不要香菜	接单 拒单 查看
2021010200001	宫保鸡丁* 3; 红烧带鱼* 2; 农家小炒肉* 1;	金燕龙办公楼 (建材城西路9号) 四层 (——宾馆北侧办公楼)	2021-01-02 11:11:11	40.00	不要香菜	接单 拒单 查看
2021010200001	宫保鸡丁* 3; 红烧带鱼* 2; 农家小炒肉* 1;	金燕龙办公楼 (建材城西路9号) 四层 (——宾馆北侧办公楼)	2021-01-02 11:11:11	40.00	不要香菜	接单 拒单 查看
2021010200001	宫保鸡丁* 3; 红烧带鱼* 2; 农家小炒肉* 1;	金燕龙办公楼 (建材城西路9号) 四层 (——宾馆北侧办公楼)	2021-01-02 11:11:11	40.00	不要香菜	接单 拒单 查看



需求分析和设计

今日数据的接口设计：

返回数据

名称	类型	是否必须	默认值	备注	其他信息
code	integer	必须			format: int32
data	object	必须			
└ newUsers	integer	必须		新增用户数	format: int32
└ orderCompletionRate	number	必须		订单完成率	format: double
└ turnover	number	必须		营业额	format: double
└ unitPrice	number	必须		平均客单价	format: double
└ validOrderCount	integer	必须		有效订单数	format: int32
msg	string	非必须			

基本信息

Path: /admin/workspace/businessData

Method: GET

接口描述:

请求参数



需求分析和设计

订单管理的接口设计：

基本信息

Path: /admin/workspace/overviewOrders

Method: GET

接口描述:

请求参数

返回数据

名称	类型	是否必须	默认值	备注	其他信息
code	integer	必须			format: int32
data	object	必须			
└ allOrders	integer	必须		全部订单	format: int32
└ cancelledOrders	integer	必须		已取消数量	format: int32
└ completedOrders	integer	必须		已完成数量	format: int32
└ deliveredOrders	integer	必须		待派送数量	format: int32
└ waitingOrders	integer	必须		待接单数量	format: int32
msg	string	非必须			



需求分析和设计

菜品总览的接口设计：

基本信息

Path: /admin/workspace/overviewDishes

Method: GET

接口描述:

请求参数

返回数据

名称	类型	是否必须	默认值	备注	其他信息
code	integer	必须			format: int32
data	object	必须			
└ discontinued	integer	必须		已停售菜品数量	format: int32
└ sold	integer	必须		已启售菜品数量	format: int32
msg	string	非必须			

需求分析和设计

套餐总览的接口设计：

基本信息

Path: /admin/workspace/overviewSetmeals

Method: GET

接口描述:

请求参数

返回数据

名称	类型	是否必须	默认值	备注	其他信息
code	integer	必须			format: int32
data	object	必须			
└ discontinued	integer	必须		已停售套餐数量	format: int32
└ sold	integer	必须		已启售套餐数量	format: int32
msg	string	非必须			



工作台

- 需求分析和设计
- 代码导入
- 功能测试

代码导入

直接导入课程资料中的工作台模块功能代码即可：

- DishMapper.java
- DishMapper.xml
- SetmealMapper.java
- SetmealMapper.xml
- WorkSpaceController.java
- WorkspaceService.java
- WorkspaceServiceImpl.java



工作台

- 需求分析和设计
- 代码导入
- 功能测试

功能测试

可以通过如下方式进行测试：

- 通过接口文档测试
- 前后端联调测试



目录

Contents

- ◆ 工作台
- ◆ Apache POI
- ◆ 导出运营数据 Excel 报表



Apache POI

- 介绍
- 入门案例

介绍

Apache POI 是一个处理 Microsoft Office 各种文件格式的开源项目。简单来说就是，我们可以使用 POI 在 Java 程序中对 Microsoft Office 各种文件进行读写操作。

一般情况下，POI 都是用于操作 Excel 文件。



为什么要在 Java 程序中操作 Excel 文件呢？



介绍

Apache POI 的应用场景：

- 银行网银系统导出交易明细
 - 各种业务系统导出 Excel 报表
 - 批量导入业务数据



介绍

例子程序运行效果展示：

D:\\itcast.xlsx

	A	B	C	D
1		姓名	爱好	
2		张三	篮球	
3		李四	足球	
4		王五	羽毛球	

```
11 > public class POITest {
12   /**
13    * 基于POI读取Excel文件
14    * @throws Exception
15   */
16 > public static void main(String[] args) throws Exception {
17   FileInputStream in = new FileInputStream(new File( pathname: "D:\\\\itcast.xlsx"));
18   //通过输入流读取指定的Excel文件
19   XSSFWorkbook excel = new XSSFF Workbook(in);
20   //获取Excel文件的第一个Sheet页
21   XSSFSheet sheet = excel.getSheetAt( index: 0);
22
23   //获取Sheet页中的最后一行的行号
24   int lastRowNum = sheet.getLastRowNum();
25
26   for (int i = 0; i <= lastRowNum; i++) {
27     //获取Sheet页中的行
28     Row row = sheet.getRow(i);
29     Cell cell = row.getCell(0);
30     String name = cell.getStringValue();
31     cell = row.getCell(1);
32     String hobby = cell.getStringValue();
33     System.out.println(name + " " + hobby);
34   }
35 }
```

Run: POITest x

```
姓名 爱好
张三 篮球
李四 足球
王五 羽毛球
```

Process finished with exit code 0

Git Run TODO Problems Statistic Terminal Profiler Endpoints Build Spring

Build completed successfully in 5 sec, 569 ms (2 minutes ago)



Apache POI

- 介绍
- 入门案例



入门案例

Apache POI 的 maven 坐标：

```
<dependency>
    <groupId>org.apache.poi</groupId>
    <artifactId>poi</artifactId>
    <version>3.16</version>
</dependency>
<dependency>
    <groupId>org.apache.poi</groupId>
    <artifactId>poi-ooxml</artifactId>
    <version>3.16</version>
</dependency>
```



入门案例

将数据写入 Excel 文件：

```
// 在内存中创建一个Excel 文件对象
XSSFWorkbook excel = new XSSFWorkbook();
// 创建Sheet 页
XSSFSheet sheet = excel.createSheet("itcast");

// 在Sheet 页中创建行，0 表示第1 行
XSSFRow row1 = sheet.createRow(0);
// 创建单元格并在单元格中设置值，单元格编号也是从 0 开始，1 表示第2 个单元格
row1.createCell(1).setCellValue("姓名");
row1.createCell(2).setCellValue("城市");

XSSFRow row2 = sheet.createRow(1);
row2.createCell(1).setCellValue("张三");
row2.createCell(2).setCellValue("北京");

XSSFRow row3 = sheet.createRow(2);
row3.createCell(1).setCellValue("李四");
row3.createCell(2).setCellValue("上海");

FileOutputStream out = new FileOutputStream(new File("D:\\\\itcast.xlsx"));
// 通过输出流将内存中的Excel 文件写入到磁盘上
excel.write(out);

// 关闭资源
out.flush();
out.close();
excel.close();
```



入门案例

读取 Excel 文件中的数据：

```
FileInputStream in = new FileInputStream(new File("D:\\itcast.xlsx"));
// 通过输入流读取指定的 Excel 文件
XSSFWorkbook excel = new XSSFWorkbook(in);
// 获取 Excel 文件的第一个 Sheet 页
XSSFSheet sheet = excel.getSheetAt(0);

// 获取 Sheet 页中的最后一行的行号
int lastRowNum = sheet.getLastRowNum();

for (int i = 0; i <= lastRowNum; i++) {
    // 获取 Sheet 页中的行
    XSSFRow titleRow = sheet.getRow(i);
    // 获取行的第 2 个单元格
    XSSFCell cell1 = titleRow.getCell(1);
    // 获取单元格中的文本内容
    String cellValue1 = cell1.getStringCellValue();
    // 获取行的第 3 个单元格
    XSSFCell cell2 = titleRow.getCell(2);
    // 获取单元格中的文本内容
    String cellValue2 = cell2.getStringCellValue();
    System.out.println(cellValue1 + " " +cellValue2);
}

// 关闭资源
in.close();
excel.close();
```



目录

Contents

- ◆ 工作台
- ◆ Apache POI
- ◆ 导出运营数据 Excel 报表



导出运营数据 Excel 报表

- 需求分析和设计
- 代码开发
- 功能测试



需求分析和设计

产品原型：





需求分析和设计

导出的 Excel 报表格式：

运营数据报表					
概览数据					
营业额		订单完成率		新增用户数	
有效订单		平均客单价			
明细数据					
日期	营业额	有效订单	订单完成率	平均客单价	新增用户数

业务规则：

- 导出 Excel 形式的报表文件
- 导出最近 30 天的运营数据

需求分析和设计

接口设计：

基本信息

Path: /admin/report/export

Method: GET

接口描述：

请求参数

返回数据

注意：当前接口没有返回数据，因为报表导出功能本质上是文件下载，

服务端会通过输出流将 Excel 文件下载到客户端浏览器



导出运营数据 Excel 报表

- 需求分析和设计
- 代码开发
- 功能测试

代码开发

实现步骤：

- ① 设计 Excel 模板文件
- ② 查询近 30 天的运营数据
- ③ 将查询到的运营数据写入模板文件
- ④ 通过输出流将 Excel 文件下载到客户端浏览器

运营数据报表					
时间：2022-05-28至2022-06-26					
概览数据					
营业额	¥2,506.00	订单完成率	76.00%	新增用户数	3
有效订单	19	平均客单价	¥131.89		
明细数据					
日期	营业额	有效订单	订单完成率	平均客单价	新增用户数
2022-05-28	¥0.00	0	0.00%	¥0.00	0
2022-05-29	¥0.00	0	0.00%	¥0.00	0
2022-05-30	¥0.00	0	0.00%	¥0.00	0
2022-05-31	¥0.00	0	0.00%	¥0.00	0
2022-06-01	¥0.00	0	0.00%	¥0.00	1
2022-06-02	¥0.00	0	0.00%	¥0.00	1
2022-06-03	¥0.00	0	0.00%	¥0.00	0
2022-06-04	¥0.00	0	0.00%	¥0.00	0





代码开发

根据接口定义，在 ReportController 中创建 export 方法：

```
/**  
 * 导出运营数据报表  
 * @param response  
 */  
@GetMapping("/export")  
@ApiOperation("导出运营数据报表")  
public void export(HttpServletRequest response){  
    reportService.exportBusinessData(response);  
}
```



代码开发

在 ReportService 接口中声明导出运营数据报表的方法：

```
/*
 * 导出近 30 天的运营数据报表
 * @param response
 */
void exportBusinessData(HttpServletRequest response);
```



代码开发

在 ReportServiceImpl 实现类中实现导出运营数据报表的方法（第 1 部分）：

```
/*
 * 导出近 30 天的运营数据报表
 * @param response
 */
public void exportBusinessData(HttpServletRequest response) {
    LocalDate begin = LocalDate.now().minusDays(30);
    LocalDate end = LocalDate.now().minusDays(1);
    // 查询概览运营数据，提供给 Excel 模板文件
    BusinessDataVO businessData = workspaceService.getBusinessData(LocalDateTime.of(begin, LocalTime.MIN), LocalDateTime.of(end,
        LocalTime.MAX));
    InputStream inputStream = this.getClass().getClassLoader().getResourceAsStream("template/运营数据报表模板.xlsx");
    try {
        // 基于提供好的模板文件创建一个新的 Excel 表格对象
        XSSFWorkbook excel = new XSSFWorkbook(inputStream);
        // 获得 Excel 文件中的一个 Sheet 页
        XSSFSheet sheet = excel.getSheet("Sheet1");
```



代码开发

在 ReportServiceImpl 实现类中实现导出运营数据报表的方法（第 2 部分）：

```
sheet.getRow(1).getCell(1).setCellValue(begin + " 至 " + end);

// 获得第 4 行
XSSFRow row = sheet.getRow(3);
// 获取单元格
row.getCell(2).setCellValue(businessData.getTurnover());
row.getCell(4).setCellValue(businessData.getOrderCompletionRate());
row.getCell(6).setCellValue(businessData.getNewUsers());

row = sheet.getRow(4);
row.getCell(2).setCellValue(businessData.getValidOrderCount());
row.getCell(4).setCellValue(businessData.getUnitPrice());

for (int i = 0; i < 30; i++) {
    LocalDate date = begin.plusDays(i);
    // 准备明细数据
    businessData = workspaceService.getBusinessData(LocalDateTime.of(date, LocalTime.MIN), LocalDateTime.of(date, LocalTime.MAX));
    row = sheet.getRow(7 + i);
```



代码开发

在 ReportServiceImpl 实现类中实现导出运营数据报表的方法（第 3 部分）：

```
row.getCell(1).setCellValue(date.toString());
row.getCell(2).setCellValue(businessData.getTurnover());
row.getCell(3).setCellValue(businessData.getValidOrderCount());
row.getCell(4).setCellValue(businessData.getOrderCompletionRate());
row.getCell(5).setCellValue(businessData.getUnitPrice());
row.getCell(6).setCellValue(businessData.getNewUsers());
}

// 通过输出流将文件下载到客户端浏览器中
ServletOutputStream out = response.getOutputStream();
excel.write(out);

// 关闭资源
out.flush();
out.close();
excel.close();
} catch (IOException e) {
    e.printStackTrace();
}
}
```



导出运营数据 Excel 报表

- 需求分析和设计
- 代码开发
- 功能测试

功能测试

可以通过如下方式进行测试：

- 前后端联调测试



传智教育旗下高端IT教育品牌