**目录**

[**1 绪论 1**](#_Toc23739)

[1.1 研究的背景及意义 1](#_Toc935)

[1.1.1 选题的背景 1](#_Toc7756)

[1.1.2 国内外研究现状 1](#_Toc11088)

[1.1.3 研究的意义 2](#_Toc3030)

[1.2 系统目标 2](#_Toc14401)

[**2 需求分析 3**](#_Toc23292)

[2.1 业务需求 3](#_Toc1285)

[2.1.1主要业务流程 3](#_Toc29100)

[2.2 功能需求 9](#_Toc18207)

[2.2.1 角色分析 9](#_Toc8577)

[2.2.2 业务功能 9](#_Toc15190)

[2.3 非功能需求 30](#_Toc1881)

[2.3.1环境需求 30](#_Toc22910)

[2.3.2 性能需求 31](#_Toc30839)

[2.3.3 安全需求 31](#_Toc29569)

[**3 总体设计 32**](#_Toc29168)

[3.1系统设计的原则 32](#_Toc434)

[3.2 系统体系结构设计 32](#_Toc25643)

[3.3 系统功能结构设计 34](#_Toc11953)

[**4 数据库设计 36**](#_Toc17343)

[4.1 概念结构设计 36](#_Toc1891)

[4.1.1 设计思路 36](#_Toc20989)

[4.1.2 E-R图 37](#_Toc6098)

[4.2 逻辑结构设计 41](#_Toc25301)

[4.2.1 设计思路 41](#_Toc17397)

[4.2.2 逻辑模型 41](#_Toc21257)

[4.3 物理结构设计 44](#_Toc25508)

[4.3.1 存取方式 44](#_Toc14530)

[4.3.2 存储结构 44](#_Toc9032)

[**5 界面设计 48**](#_Toc27142)

[5.1界面关系图 48](#_Toc30478)

[5.2 界面设计成果 49](#_Toc3998)

[5.2.1 主界面 49](#_Toc7330)

[5.2.2 子界面 50](#_Toc28781)

[**6 详细设计 54**](#_Toc18358)

[6.1系统主要功能模块介绍 54](#_Toc16461)

[6.1生鲜选购 55](#_Toc14313)

[6.2收货地址管理 59](#_Toc22238)

[6.3订单管理 63](#_Toc10958)

[6.4购物车管理 68](#_Toc5632)

[6.5商家账号管理 71](#_Toc29462)

[6.6生鲜管理 75](#_Toc17601)

[6.7用户管理 79](#_Toc2815)

[6.8财务管理 83](#_Toc7142)

[6.9统计管理 87](#_Toc23337)

[**7 编码 91**](#_Toc6062)

[7.1 代码实现与核心算法 91](#_Toc17063)

[7.1.1用户搜索生鲜业务 91](#_Toc29138)

[7.1.2添加收货地址业务 91](#_Toc7933)

[7.1.3用户评论订单业务 92](#_Toc18083)

[7.1.4生鲜添加购物车业务 92](#_Toc25774)

[7.1.5修改后台账号密码业务 93](#_Toc9282)

[7.1.6商家上架生鲜业务 93](#_Toc20057)

[7.1.7查看用户信息业务 94](#_Toc13262)

[7.1.8订单发货业务 94](#_Toc15791)

[7.1.9展示生鲜销售总额业务 95](#_Toc254)

[7.2 代码优化分析 95](#_Toc9522)

[**8 测试 96**](#_Toc31585)

[8.1 测试方案设计 96](#_Toc18658)

[8.1.1 测试策略 96](#_Toc26341)

[8.1.2 测试进度安排 96](#_Toc21817)

[8.1.3 测试资源 96](#_Toc23892)

[8.1.4 关键测试点 96](#_Toc1834)

[8.2 测试用例构建 97](#_Toc20705)

[8.2.1 测试用例编写约定 97](#_Toc13570)

[8.2.2 测试用例设计 97](#_Toc16383)

[8.2.3 关键测试用例 98](#_Toc18262)

[8.2.4 测试用例维护 99](#_Toc3354)

[**9 总结与展望 100**](#_Toc27238)

[9.1 设计工作总结 100](#_Toc16616)

[9.2 未来工作展望 100](#_Toc17312)

[**谢 辞 101**](#_Toc29802)

[**参考文献 102**](#_Toc5158)

[**附录A 外文翻译—原文部分 103**](#_Toc18394)

[**附录B 外文翻译—译文部分 109**](#_Toc14458)

[**附录C 软件使用说明书 114**](#_Toc13842)

[**附录D 主要源代码 117**](#_Toc14032)

1 绪论

## 1.1 研究的背景及意义

### 1.1.1选题的背景

在2020年初突发的新冠肺炎疫情期间，“不聚集、少出门”成为共识后，许多人都减少了到农贸市场和超市买菜的频率，于是居民需要一种足不出户的无接触式购置菜品的方式，基于微信小程序的生鲜外卖购物系统就很好地满足了特殊时期居民的迫切需求。并且，从长远来看，移动互联网已越来越融入人们的生活，同时人们的日常工作生活也逐渐繁忙起来，因此在网上订购货物让快递员配送到家越来越流行。很多人在一天繁忙的工作之后还要去菜市场、超市购买肉类、蔬菜等食材，这对他们而言时间既紧迫，又很大概率买不到新鲜的水果蔬菜，所以消费者对生鲜产品的需求开始发生转变，对于品类丰富、生鲜新鲜度、便捷度等需求全面提升，不再向以前一样需求单一，于是需要一个生鲜外卖小程序来满足消费者的多元化和高质量需求。

另外，随着人们的生活品质逐渐提高，大家对于吃的方面开始越来越讲究，这便促进了生鲜行业的发展，吸引众多商家纷纷进驻掘金，但是生鲜行业做强做大一定不能缺少线上销售这个领域。传统的生鲜电商通常会选择各种比较知名的外卖平台进驻，借助知名平台的流量来为自己的商店引流，但当大量商家入驻相同平台后，原有的平台优势也就逐渐丧失。虽然在平台打广告购买流量的方式也愈加流行，但是近几年流量的费用也愈加昂贵，广告成本逐渐增加，商家们可谓是叫苦连天。对于当前阶段而言，进驻平台的红利逐渐丧失，大量的顾客流量来源从平台转向了自有平台，同时腾讯也在积极倡导去中心化的思维方式，早期自有平台主要包括自有网站和微信的微商城。但是随着互联网技术和市场的迅速发展，小程序逐渐进入大众的视线，同时随着一些知名的生鲜品牌“百果园”、“每日优鲜”通过入驻小程序实现了巨大流量的增长。小程序作为一种方便快捷的互联网应用，“无需安装”“用完即走”的便利特性，让大家越来越多的选择小程序。

生鲜外卖小程序的出现极大满足了用户多元化和高质量需求，生鲜小程序可以在线下门店提供服务，线上进行营销，可以让顾客不用下载app就可直接使用微信订购下单，很方便地买到各种新鲜的生鲜食物。利用微信小程序打通线下线上销售壁垒，使商家不再拘泥于大众化的外卖平台，拥有自己专属的小程序，体现商家的个性化服务。

### 1.1.2 国内外研究现状

近十年，在互联网的不断高速发展之下，移动终端以及手机支付方式的普及让人们的生活发生了前所未有的变化，同时电子商务在互联网的快速发展之下也在发生更新和改变，智能化应用从电脑 PC端转移到手机移动端，变得十分普及，这就使得手机在人们的生活中扮演了一个非常重要的角色，人们在生活中越来越多使用各种购物软件进行购物，在软件上下单直接让快递员送货上门，人们的生活因此越来愈便利，商家的销售渠道也得到了拓展。但是这种模式也随着网络购物的不断发展也逐渐出现了一些瓶颈，例如物流时间的不够及时和“最后一公里”的配送问题，该配送问题在俄罗斯外卖配送中尤其突出，配送员往往要花费两三个小时才能把食物送到顾客手中。而基于社区商铺的新零售的诞生缩小了线上购物的区域，为用户提供了更方便快捷的邻近区域服务同时也为线下的商铺的带来了更多元的发展。而微信小程序作为一款不同于于 PC 端和移动APP 端的新产品，有着微信的海量用户优势，可以更好地连接线下用户。在新零售电子商务的第二次冲击下，微信小程序的问世可为中小型的传统线下零售行业在这场变革之中提供了重要的平台。

### 1.1.3 研究的意义

生鲜外卖小程序促使o2o商城和微信小程序二者得到完美结合，彼此相得益彰，以此实现了线上生鲜食品的“即买即得”，这样不仅方便用户，而且使商家还能拥有丰厚的流量。而且与传统的商店模式相比，生鲜外卖小程序的优点性更强，具体体现在：第一、其开发成本低，第二、其维护成本也较低，第三、其管理和运营模式也更为轻松，这些优点都有利于让商家把更多的精力投入至运营好产品和运营产品好当中。小程序不断更新迭代，可以说微信的每次系统更新改变基本上都是为小程序性能提供服务，而小程序作为微信未来主打品牌产品，将来必定会逐渐开放更多丰富、更加多元、更高质量的功能，所以建立公众号与小程序的数据汇通渠道，帮助商家建立用户的数据体系的系统，前景十分巨大。

## 1.2 系统目标

开发出一款为客户提供充足的信息和快捷锁鲜的送货上门方式的生鲜外卖小程序，给生鲜店家提供一套科学化、规范化的生鲜外卖新销售模式。通过这个小程序，可以让顾客直接使用微信订购下单，方便快捷地买到新鲜的蔬菜、鲜肉、生果、冻品、海鲜等。同时，也使商家不再拘泥于大众化的外卖平台，拥有自己专属的小程序，实现管理独立化，品牌效应化。

在生鲜外卖小程序中，用户可以通过首页生鲜分类，挑选各种生鲜产品，点击产品后可查看生鲜介绍、用户评价、好评率、销售数量等等，看到想要的生鲜可以将其直接加入购物车，并在购物车界面可以直接进行下单并支付等等。

# 2 需求分析

## 2.1 业务需求

* 1. 能够实现用户在小程序上获取生鲜展示,进行生鲜检索、生鲜选择、小程序上

下单、支付和评价订单等操作以及商家在后台管理端进行生鲜上架、修改账号信息和订单发货等操作，对微信小程序上订购社区生鲜的全过程进行管理。

* 1. 实现对微信小程序下单生鲜过程中产生的所有业务数据的管理，如订单信

息、生鲜上架信息、支付记录、收货地址信息、发货记录的管理与维护。

* 1. 小程序应具有配套的后台管理端，供商家能够对生鲜信息、小程序用户信

息、财务订单信息等信息进行管理与维护。

* 1. 响应速度良好，安全性较高。

### 2.1.1主要业务流程

1. 生鲜选购活动图如图2-1所示：



图2-1生鲜选购活动图

1. 收货地址管理活动图如图2-2所示：



图2-2收获地址管理活动图

1. 订单管理活动图如图2-3所示：



图2-3订单管理活动图

1. 购物车管理活动图如图2-4所示：



图2-4购物车管理活动图

1. 商家账号管理活动图如图2-5所示：



图2-5商家账号管理活动图

1. 生鲜管理活动图如图2-6所示：



图2-6生鲜管理活动图

1. 用户管理活动图如图2-7所示：



图2-7用户管理活动图

1. 财务管理活动图如图2-8所示：



图2-8财务管理活动图

1. 统计管理活动图如图2-9所示：



图2-9统计管理活动图

## 2.2 功能需求

### 2.2.1 角色分析

以下从生鲜外卖小程序系统的实际需求分析，系统涉及到以下角色

系统角色表如表2-1所示：

表2-1 系统角色表

|  |  |
| --- | --- |
| 角色 | 职责或功能 |
| 客户(买家) | 小程序的受众，查看、挑选、加购、下单、评论小程序内的生鲜。 |
| 商家管理员（卖家） | 管理和维护生鲜外卖小程序的整体组织结构，负责生鲜的添加与删除、对小程序上已支付的订单进行发货、查看小程序用户基本信息等。 |

### 2.2.2 业务功能

以下从业务角度出发，给出了系统的总体用例图如图2-10所示：



图2-10系统总体用例图

顾客：顾客可以通过微信帐号授权登录生鲜外卖小程序，进入小程序首页浏览生鲜产品，搜索想要的生鲜，点击展示的生鲜图片进入详情页面，如果喜欢当前生鲜可以将其加入购物车，在购物车中可以修改生鲜的数量、删除某个生鲜产品，并且可以选择直接下单购买。想要成功下单需先在收货地址管理页面添加收货地址，成功后可在订单页面查看到自己的订单，收到生鲜后可点击确认收货并评价订单。

商家：商家以通过登录后台管理端对小程序中生鲜进行添加、修改库存、分类、删除等操作。对商家账号的信息进行修改、删除，对微信小程序登录用户信息进行查看。并可通过财务管理查看历史订单，对已支付订单进行发货处理，在统计管理中查看统计信息对商铺营收情况进行了解分析。

（1）生鲜选购

生鲜选购用例图描述如图2-11所示：



图2-11生鲜选购用例图

生鲜选购管理用例描述如表2-2所示：

表2-2生鲜选购管理用例描述

|  |  |
| --- | --- |
| 描述项 | 说明 |
| 用例名称 | 生鲜选购管理 |
| 标识符\* | YL01 |
| 用例描述 | 描述了买家使用小程序进行生鲜选购的整个过程 |
| 参与者表 | 客户（买家） |
| 优先级 | 1 |
| 状态\* | 进行中 |
| 前置条件 | 用户已登录小程序 |
| 后置条件 | 系统给出操作成功提示 |
| 基本操作流 | 1.用户进入小程序生鲜展示界面，在页面顶部是生鲜搜索框，可以输入要购买的生鲜信息进行搜索，页面也将搜索到的生鲜依次展示；  2.对于想要详细了解的生鲜，可以直接点击生鲜图片，即可跳转至生鲜详细信息页面进行生鲜浏览；  3.在详情页面通过点击分享图标，把当前生鲜分享给微信好友； |
| 可选操作流 | 无 |

续表2-2

|  |  |
| --- | --- |
| 被泛化用例表 | 无 |
| 被包含用例表 | 无 |
| 被扩展用例表 | 无 |

生鲜搜索用例描述如表2-3所示：

表2-3搜索生鲜用例描述

|  |  |
| --- | --- |
| 描述项 | 说明 |
| 用例名称 | 搜索生鲜 |
| 标识符\* | YL02 |
| 用例描述 | 在生鲜页面的搜索框中输入搜索生鲜相关信息，显示出名称中包含该关键字的生鲜 |
| 参与者表 | 客户（买家） |
| 优先级 | 2 |
| 状态\* | 进行中 |
| 前置条件 | 用户已登录系统 |
| 后置条件 | 搜索结果显示在该页面 |
| 基本操作流 | 1. 用户进入小程序中生鲜展示页面， 2. 点击页面顶部的搜索框 3. 输入生鲜相关信息 4. 点击搜索 |
| 可选操作流 | 无 |
| 被泛化用例表 | 无 |
| 被包含用例表 | 生鲜选购 |
| 被扩展用例表 | 无 |

分享生鲜详情用例描述如表2-4所示：

表2-4分享生鲜详情用例描述

|  |  |
| --- | --- |
| 描述项 | 说明 |
| 用例名称 | 分享生鲜 |

续表2-4

|  |  |
| --- | --- |
| 标识符\* | YL04 |
| 用例描述 | 分享生鲜是顾客将生鲜分享给微信好友的功能模块，顾客通过点击生鲜图片下方的分享图标，把当前生鲜产品分享给微信好友； |
| 参与者表 | 客户（买家） |
| 优先级 | 2 |
| 状态\* | 进行中 |
| 前置条件 | 用户已登录系统 |
| 后置条件 | 成功分享页面 |
| 基本操作流 | 1.用户点击生鲜图片下方分享图标  2.弹出微信好友列表  3.点击确认，完成分享 |
| 可选操作流 | 无 |
| 被泛化用例表 | 无 |
| 被包含用例表 | 生鲜选购 |
| 被扩展用例表 | 无 |

（2）收货地址管理

收货地址管理用例图如图2-12所示：



图2-12收货地址管理用例图

收货地址管理用例描述如表2-5所示：

表2-5收货地址管理用例描述

|  |  |
| --- | --- |
| 描述项 | 说明 |
| 用例名称 | 收货地址管理 |
| 标识符\* | YL01 |
| 用例描述 | 收货地址管理模块是用户操作的功能模块，用户在登录小程序后在该页面可以添加、删除以及查看收货地址， |
| 参与者表 | 客户（买家） |
| 优先级 | 1 |
| 状态\* | 进行中 |
| 前置条件 | 用户进入地址管理页面 |
| 后置条件 | 系统给出操作成功提示，数据库中收货地址操作成功 |
| 基本操作流 | 1.用户点击小程序底部栏中的个人图标，进入个人中心  2.查看收货地址  3.点击删除并确认  4.数据库更新  5.修改收货地址  6.点击保存  7.提交到数据库 |
| 可选操作流 | 无 |
| 被泛化用例表 | 收货地址管理 |
| 被包含用例表 | 无 |
| 被扩展用例表 | 无 |

添加收货地址用例描述如表2-6所示：

表2-6添加收货地址用例描述

|  |  |
| --- | --- |
| 描述项 | 说明 |
| 用例名称 | 添加收货地址 |
| 标识符\* | YL02 |
| 用例描述 | 描述了买家进入个人地址界面进行地址添加的整个过程 |

续表2-6

|  |  |
| --- | --- |
| 参与者表 | 客户（买家） |
| 优先级 | 2 |
| 状态\* | 进行中 |
| 前置条件 | 用户进入地址管理页面 |
| 后置条件 | 系统给出操作成功提示，数据库中收货地址操作成功 |
| 基本操作流 | 1.用户选择小程序底部栏中的个人图标，进入个人中心  2.用户点击我的地址进入管理页面  3.进入添加地址填写信息  4.点击保存  5.提交到数据库 |
| 可选操作流 | 无 |
| 被泛化用例表 | 无 |
| 被包含用例表 | 收货地址管理 |
| 被扩展用例表 | 无 |

（3）订单管理

订单管理用例图如图2-13所示：



图2-13订单管理用例图

订单管理用例描述如表2-7所示：

表2-7订单管理用例描述

|  |  |
| --- | --- |
| 描述项 | 说明 |
| 用例名称 | 订单管理 |
| 标识符\* | YL01 |
| 用例描述 | 描述了用户对订单进行管理的整个过程 |
| 参与者表 | 客户（买家） |
| 优先级 | 2 |
| 状态\* | 进行中 |
| 前置条件 | 用户进入订单管理 |
| 后置条件 | 系统给出操作成功提示 |
| 基本操作流 | 1. 用户进入订单待支付页面  2. 点击付款按钮  3. 输入密码完成支付  4，或点击取消付款 |
| 可选操作流 | 无 |
| 被泛化用例表 | 订单管理 |
| 被包含用例表 | 无 |
| 被扩展用例表 | 无 |

完成订单用例描述如表2-8所示：

表2-8完成订单用例描述

|  |  |
| --- | --- |
| 描述项 | 说明 |
| 用例名称 | 完成订单 |
| 标识符\* | YL02 |
| 用例描述 | 买家使用本生鲜小程序进行确认收货与评价 |
| 参与者表 | 客户（买家） |
| 优先级 | 2 |
| 状态\* | 进行中 |

续表2-8

|  |  |
| --- | --- |
| 前置条件 | 用户进入订单管理页面 |
| 后置条件 | 系统给出操作成功提示，数据库中订单管理操作成功 |
| 基本操作流 | 1. 用户进入订单页面  2. 选择确认收货按钮  3.点击选择评价订单  4.输入信息、点击保存  5. 数据库更新 |
| 可选操作流 | 无 |
| 被泛化用例表 | 无 |
| 被包含用例表 | 订单管理 |
| 被扩展用例表 | 无 |

取消订单用例描述如表2-9所示：

表2-9取消订单用例描述

|  |  |
| --- | --- |
| 描述项 | 说明 |
| 用例名称 | 取消订单 |
| 标识符\* | YL03 |
| 用例描述 | 买家使用本生鲜外卖小程序端对已创建订单进行取消操作 |
| 参与者表 | 客户（买家） |
| 优先级 | 2 |
| 状态\* | 进行中 |
| 前置条件 | 用户进入订单管理界面 |
| 后置条件 | 系统给出操作成功提示 |
| 基本操作流 | 1. 点击取消订单按钮  2. 点击确认  3.数据库更新 |
| 可选操作流 | 无 |
| 被泛化用例表 | 无 |
| 被包含用例表 | 订单管理 |
| 被扩展用例表 | 无 |

确认收货用例描述如表2-10所示：

表2-10确认收货用例描述

|  |  |
| --- | --- |
| 描述项 | 说明 |
| 用例名称 | 确认收货 |
| 标识符\* | YL4 |
| 用例描述 | 买家使用本生鲜外卖小程序端收到生鲜后对订单进行确认收货操作 |
| 参与者表 | 客户（买家） |
| 优先级 | 2 |
| 状态\* | 进行中 |
| 前置条件 | 用户进入订单管理界面 |
| 后置条件 | 系统给出操作成功提示 |
| 基本操作流 | 1. 选择确认收货图标  2. 点击确认  3.数据库更新 |
| 可选操作流 | 无 |
| 被泛化用例表 | 无 |
| 被包含用例表 | 订单管理 |
| 被扩展用例表 | 无 |

（4）购物车管理

购物车管理用例如图2-14所示：



图2-14购物车管理用例图

购物车管理用例描述如表2-11所示：

表2-11购物车管理用例描述

|  |  |
| --- | --- |
| 描述项 | 说明 |
| 用例名称 | 购物车管理 |
| 标识符\* | YL01 |
| 用例描述 | 用户对购物车进行管理的整个过程，如把生鲜加入购物车，删除某个生鲜。 |
| 参与者表 | 客户（买家） |
| 优先级 | 1 |
| 状态\* | 进行中 |
| 前置条件 | 用户进入购物车界面 |
| 后置条件 | 系统给出操作成功提示 |
| 基本操作流 | 1. 选择‘编辑’图标 2. 用户选定要删除的生鲜并选择‘删除’图标 3. 系统删除选中的生鲜 4. 选择修改对生鲜数量进行修改 5. 提交到数据库 |
| 可选操作流 | 无 |
| 被泛化用例表 | 购物车管理 |

续表2-11

|  |  |
| --- | --- |
| 被包含用例表 | 无 |
| 被扩展用例表 | 无 |

删除购物车商品用例描述如表2-12所示：

表2-12删除购物车商品用例描述

|  |  |
| --- | --- |
| 描述项 | 说明 |
| 用例名称 | 删除购物车商品 |
| 标识符\* | YL02 |
| 用例描述 | 买家将购物车内的生鲜删除 |
| 参与者表 | 客户（买家） |
| 优先级 | 3 |
| 状态\* | 进行中 |
| 前置条件 | 用户已登录系统 |
| 后置条件 | 系统给出操作成功提示 |
| 基本操作流 | 1. 选择“购物车”按钮进入购物车管理界面   2. 选择“编辑”  3. 选择生鲜  4. 选择“删除” |
| 可选操作流 | 无 |
| 被泛化用例表 | 无 |
| 被包含用例表 | 购物车管理 |
| 被扩展用例表 | 无 |

1. 商家账号管理

商家账号管理用例图如图2-15所示：



图2-15商家账号管理用例图

商家账号管理用例描述如表2-13所示：

表2-13商家账号管理用例描述

|  |  |
| --- | --- |
| 描述项 | 说明 |
| 用例名称 | 商家账号管理 |
| 标识符\* | YL01 |
| 用例描述 | 商家对后台端账号进行管理，如修改账号信息，添加账号。 |
| 参与者表 | 管理员（商家） |
| 优先级 | 1 |
| 状态\* | 进行中 |
| 前置条件 | 商家已登录小程序后台端 |
| 后置条件 | 系统给出操作成功提示 |
| 基本操作流 | 1. 进入账号管理页面 2. 选择“添加”进行账号添加 3. 选择“删除”进行删除 4. 选择“编辑”进行信息修改 |
| 可选操作流 | 无 |
| 被泛化用例表 | 账号管理 |
| 被包含用例表 | 无 |
| 被扩展用例表 | 无 |

修改账号用例描述如表2-14所示：

表2-14修改账号用例描述

|  |  |
| --- | --- |
| 描述项 | 说明 |
| 用例名称 | 修改账号 |
| 标识符\* | YL03 |
| 用例描述 | 卖家使用后台管理系统修改商家账号信息 |
| 参与者表 | 管理员（商家） |
| 优先级 | 1 |
| 状态\* | 进行中 |
| 前置条件 | 商家已登录小程序后台端 |
| 后置条件 | 系统给出操作成功提示 |
| 基本操作流 | 1.商家进入账户管理  2.选择“编辑”  3.对账号信息进行修改  4.点击保存 |
| 可选操作流 | 无 |
| 被泛化用例表 | 无 |
| 被包含用例表 | 商家账号管理 |
| 被扩展用例表 | 无 |

（6）生鲜管理

生鲜管理用例图如图2-16所示：



图2-16生鲜管理用例图

生鲜管理用例描述如表2-15所示：

表2-15生鲜管理用例描述

|  |  |
| --- | --- |
| 描述项 | 说明 |
| 用例名称 | 生鲜管理 |
| 标识符\* | YL01 |
| 用例描述 | 商家在后台对生鲜进行管理，如上架生鲜，查看生鲜详细信息，删除某个生鲜和对生鲜信息进行修改。 |
| 参与者表 | 管理员（商家） |
| 优先级 | 1 |
| 状态\* | 进行中 |
| 前置条件 | 商家已登录小程序后台端 |
| 后置条件 | 系统给出操作成功提示 |
| 基本操作流 | 1.商家进入生鲜管理页面  2.选择“添加”添加生鲜  3.选择“删除”删除生鲜  4.选择“编辑”进行生鲜信息修改  5.用搜索框进行生鲜搜索 |
| 可选操作流 | 无 |

续表2-15

|  |  |
| --- | --- |
| 被泛化用例表 | 生鲜管理 |
| 被包含用例表 | 无 |
| 被扩展用例表 | 无 |

上架生鲜用例描述如表2-16所示：

表2-16上架生鲜用例描述

|  |  |
| --- | --- |
| 描述项 | 说明 |
| 用例名称 | 上架生鲜 |
| 标识符\* | YL02 |
| 用例描述 | 商家使用后台管理系统上架生鲜 |
| 参与者表 | 管理员（商家） |
| 优先级 | 1 |
| 状态\* | 进行中 |
| 前置条件 | 商家已登录小程序后台端 |
| 后置条件 | 系统给出操作成功提示 |
| 基本操作流 | 1.商家进入生鲜管理页面  2.选择“添加”  3.输入生鲜名、生鲜分类、价格、库存、生鲜介绍、生鲜入库信息等信息  4.点击保存 |
| 可选操作流 | 无 |
| 被泛化用例表 | 无 |
| 被包含用例表 | 生鲜管理 |
| 被扩展用例表 | 无 |

（7）用户管理

用户管理用例如图2-17所示：



图2-17用户管理用例图

用户管理用例描述如表2-17所示：

表2-17用户管理用例描述

|  |  |
| --- | --- |
| 描述项 | 说明 |
| 用例名称 | 用户管理 |
| 标识符\* | YL01 |
| 用例描述 | 商家在后台对用户进行管理，如查看用户详细信息与评论信息，修改用户昵称，进行用户搜索。 |
| 参与者表 | 管理员（商家） |
| 优先级 | 1 |
| 状态\* | 进行中 |
| 前置条件 | 商家已登录小程序后台端 |
| 后置条件 | 系统给出操作成功提示 |
| 基本操作流 | 1.用户点击用户管理进入用户管理页面  2.选择“查看”查看详细信息  3.选择“编辑”对昵称等信息进行修改  4.用搜索框进行用户搜索 |
| 可选操作流 | 无 |
| 被泛化用例表 | 用户管理 |
| 被包含用例表 | 无 |
| 被扩展用例表 | 无 |

查询用户用例描述如表2-18所示：

表2-18查询用户用例描述

|  |  |
| --- | --- |
| 描述项 | 说明 |
| 用例名称 | 查询用户 |
| 标识符\* | YL03 |
| 用例描述 | 商家使用后台管理系统查询用户 |
| 参与者表 | 管理员（商家） |
| 优先级 | 1 |
| 状态\* | 进行中 |
| 前置条件 | 商家已登录小程序后台端 |
| 后置条件 | 系统给出操作成功提示 |
| 基本操作流 | 1.商家进入用户管理  2.选择搜索框，输入搜索内容  3.确认搜索 |
| 可选操作流 | 无 |
| 被泛化用例表 | 无 |
| 被包含用例表 | 用户管理 |
| 被扩展用例表 | 无 |

（8）财务管理

财务管理用例如图2-18所示：



图2-18财务管理用例图

财务管理用例描述如表2-19所示：

表2-19财务管理用例描述

|  |  |
| --- | --- |
| 描述项 | 说明 |
| 用例名称 | 财务管理 |
| 标识符\* | YL01 |
| 用例描述 | 商家在后台对财务进行管理，如查看订单信息与财务流水，点击发货按钮对订单状态进行修改。 |
| 参与者表 | 管理员（商家） |
| 优先级 | 1 |
| 状态\* | 进行中 |
| 前置条件 | 商家已登录小程序后台端 |
| 后置条件 | 系统给出操作成功提示 |
| 基本操作流 | 1.商家进入财务管理  2.选择“查看”了解订单详情  3.选择“确认发货”，对已支付订单进行发货处理。 |
| 可选操作流 | 1.选择“财务流水”可查看总收款金额以及最近订单 |
| 被泛化用例表 | 财务管理 |
| 被包含用例表 | 无 |
| 被扩展用例表 | 无 |

查看订单信息用例描述如表2-20所示：

表2-20查看订单信息用例描述

|  |  |
| --- | --- |
| 描述项 | 说明 |
| 用例名称 | 查看订单信息 |
| 标识符\* | YL02 |
| 用例描述 | 卖家使用后台管理系统查看订单详情 |
| 参与者表 | 管理员（商家） |
| 优先级 | 1 |
| 状态\* | 进行中 |

续表2-20

|  |  |
| --- | --- |
| 前置条件 | 商家已登录小程序后台端 |
| 后置条件 | 系统给出操作成功提示 |
| 基本操作流 | 1.商家进入财务管理  2.选择“查看”查看订单详情 |
| 可选操作流 | 无 |
| 被泛化用例表 | 无 |
| 被包含用例表 | 财务管理 |
| 被扩展用例表 | 无 |
| 修改历史记录\* | 暂无 |
| 问题\* | 暂无 |
| 决策\* | 暂无 |
| 频率\* | 暂无 |

商家发货用例描述如表2-21所示：

表2-21商家发货用例描述

|  |  |
| --- | --- |
| 描述项 | 说明 |
| 用例名称 | 商家发货 |
| 标识符\* | YL03 |
| 用例描述 | 卖家使用后台管理系统进行发货 |
| 参与者表 | 管理员（商家） |
| 优先级 | 1 |
| 状态\* | 进行中 |
| 前置条件 | 商家已登录小程序后台端 |
| 后置条件 | 系统给出操作成功提示 |
| 基本操作流 | 1.商家进入财务管理  2.选择“查看”查看订单详情  3.选择“确认发货” |
| 可选操作流 | 无 |

续表2-21

|  |  |
| --- | --- |
| 被泛化用例表 | 无 |
| 被包含用例表 | 财务管理 |
| 被扩展用例表 | 无 |

（9）统计管理

统计管理用例图如图2-19所示：



图2-19统计管理用例图

统计管理用例描述如表2-22所示：

表2-22统计管理用例描述

|  |  |
| --- | --- |
| 描述项 | 说明 |
| 用例名称 | 统计管理 |
| 标识符\* | YL01 |
| 用例描述 | 管理员在成功登录后台系统后可以进入统计管理查看每天的营收金额数据，并可按日期段搜索该时间段内营收情况。 |
| 参与者表 | 管理员（商家） |
| 优先级 | 1 |
| 状态\* | 进行中 |
| 前置条件 | 商家已登录小程序后台端 |
| 后置条件 | 系统给出操作成功提示 |

续表2-22

|  |  |
| --- | --- |
| 基本操作流 | 1.商家进入统计管理  2.系统显示每日营收分类情况  3.选择日期  4.显示该日期段营收情况 |
| 可选操作流 | 无 |
| 被泛化用例表 | 统计管理 |
| 被包含用例表 | 无 |
| 被扩展用例表 | 无 |

查看每日营业数据用例描述如表2-23所示：

表2-23查看每日营业数据用例描述

|  |  |
| --- | --- |
| 描述项 | 说明 |
| 用例名称 | 查看每日营业数据 |
| 标识符\* | YL02 |
| 用例描述 | 卖家使用后台管理系统查看营业数据 |
| 参与者表 | 管理员（商家） |
| 优先级 | 1 |
| 状态\* | 进行中 |
| 前置条件 | 商家已登录小程序后台端 |
| 后置条件 | 系统给出操作成功提示 |
| 基本操作流 | 1.用户点击统计管理进入统计管理页面  2.点击不同分类查看不同类别每日营收情况 |
| 可选操作流 | 无 |
| 被泛化用例表 | 无 |
| 被包含用例表 | 统计管理 |
| 被扩展用例表 | 无 |

搜索时间段内营收情况用例描述如表2-24所示：

表2-24搜索时间段内营收情况用例描述

|  |  |
| --- | --- |
| 描述项 | 说明 |
| 用例名称 | 搜索某时间段内营收情况 |
| 标识符\* | YL03 |
| 用例描述 | 卖家使用后台管理系统进行搜索时间段内营收情况 |
| 参与者表 | 管理员（商家） |
| 优先级 | 1 |
| 状态\* | 进行中 |
| 前置条件 | 商家已登录小程序后台端 |
| 后置条件 | 系统给出操作成功提示 |
| 基本操作流 | 1.用户点击统计管理进入统计管理页面  2.选择日历，选择某段时间段  3.选择查询 |
| 可选操作流 | 无 |
| 被泛化用例表 | 无 |
| 被包含用例表 | 统计管理 |
| 被扩展用例表 | 无 |

## 2.3 非功能需求

### 2.3.1环境需求

服务器端环境需求如表2-34所示：

表 2-34服务器端环境

|  |  |
| --- | --- |
| 系统所使用数据库 | MySQL 8.0 |
| 操作系统 | Windows 10 专业版 |
| Web服务器 | WSGI |

客户端环境需求如表2-35所示：

表 2-35客户端环境

|  |  |
| --- | --- |
| 操作系统 | Windows 7及其以上版本的操作系统 |
| 浏览器 | Firefox、Chrome、IE10.0等 |
| 分辨率 | 最佳效果为1024×768像素 |

### 2.3.2 性能需求

1. 稳定性

稳定性的意思就是小程序在需要稳定运行的时候，能保持其正常运行，功能完整不会出现缺失以及维持一个良好的服务性能水平，尤其是在用户使用时保持稳定。因为微信小程序会存在官方更新接口等操作，因此需要我们对后端接口进行及时更新并做详细记录。

1. 可用性

可用性要求用户在不看操作说明书的情况下，就能在一定时间内对小程序的进行一些基本操作，比如登录授权、下单订购等等。同时尽可能地参照我们生产生活中的主流系统的设计，小程序内所有页面的框架和风格尽可能统一，按钮的设计、颜色的选取尽可能迎合用户的喜好和习惯，为用户与系统的交互提供便利。

1. 高并发性

高并发性是指当多个用户访问小程序时，应不会对小程序的平均响应时间造成多少影响。用户对小程序的各项操作的最长响应时间应该不超过8秒。同时因为服务器和数据库的资源有限，同时部署了小程序以及后台管理系统，因此需要确定一个比较小的资源占比。

### 2.3.3 安全需求

1. 用户安全

本系统小程序端和后台管理端都是通过登录的方式进入，没有得到授权登录的用户无法进入系统。在小程序端所采取的是https协议的方式进行通信，具有较强的安全性。同时通过绑定了用户的微信号实现在登录后再次登录无需再次进行授权操作，再次保证了用户登录信息的唯一性。在后台管理端，商家管理员的密码基于特定算法的加密密匙确定，在一定程度上提升了系统的安全性。

2. 记录日志

生鲜外卖系统能够对用户的关键性操作进行记录，同时记录生鲜外卖系统在运行时发生错误的所有信息。以便后续生鲜外卖系统出现故障时能够回到系统故障之前的状态，恢复正常运行。

3 总体设计

## 3.1系统设计的原则

1. 实用性原则。系统的实用性要求是笔者设计生鲜外卖小程序时的最重点考虑的方面。本系统以紧密切合社区生鲜商家和的社区住户的实际业务需求为目标，希望通过自身研究来彻底解决传统社区生鲜购物方式不够便利、效率低下的痛点。
2. 先进性原则。系统的先进性也是笔者考虑的重点内容。当今时代，信息化在不断高速发展，每一次技术上的突破都会促进互联网生态改变。所以，让本系统有一个高质量的起点，在本系统的设计过程中笔者选取了当下热门的微信小程序，这样可以使系统的性价比优势得到充分的显现，具有一定的竞争力。
3. 安全性原则。系统应当具备必要的保护措施，既能让资源信息实现各方面的共享，也能实现信息之间的保护与隔离，系统在各层面的访问中实现控制，严格限制操作权限，并通过日志系统记录后台商家管理员的操作，让系统具备良好的容错与恢复能力，使系统能够长期正常稳定地运行。
4. 经济性原则。在满足生鲜外卖小程序功能及性能要求的前提下，尽量降低生鲜外卖小程序的建设成本，采用经济且实效的技术手段，利用现有资源，综合考虑系统的搭建、更新升级以及维护时所需的成本。

## 3.2 系统体系结构设计

经过对现阶段社区生鲜商家和顾客的分析，发现在实际购物场景中商家和顾客往往倾向于使用简单便捷、占用内存少的应用，因此在用户进行操作的移动端选用依托于微信社交平台的微信小程序，商家管理后台web端根据需求分析并且考虑到开发周期较短的情况，采用前后端分离架构模式，使前端和后端可以在各自的部分更加专业化。小程序端选用MINA框架，这是微信开发者为小程序开发提供的一种框架，该框架具有的 Java Script API非常完善，开发人员可以很方便地获得其具有的各种功能，让小程序的搭建非常便捷。

后端使用python flask框架，服务器使用flask自带的WSGI。Flask框架的特点是框架本身非常的简略，体积小，但它却具备十分强大的扩展能力，通过pip可以安装各种python库扩展系统功能，例如使用 flask-sqlacodegen 扩展可方便快速生成 ORM model。对于不需要的功能库直接进行删除就可以了，这种可供用户定制功能的特点正flask框架流行的原因。前后端通过RESTful风格api进行通信。系统架构图如图3-1所示：



图3-1系统架构图

生鲜外卖小程序系统结构图如图3-2所示：



图3-2系统结构图

系统结构第一层为表现层，主要包括展现界面的UI展示类。第二层为业务逻辑层，主要包括实体类和服务类。第三层为数据层，主要包括映射类和数据控制类。数据层主要是对于数据库中的数据进行操作，具体为业务逻辑层提供数据服务，其依赖余SQAlchemy这一ORM框架将关系数据库的表结构映射到对象上。表现层则依赖Flask自带的WSGI服务器得以展现。

## 3.3 系统功能结构设计

系统总包图如图3-3所示：



3-3 系统总包图

1. 生鲜选购：顾客可通过微信账号授权登陆生鲜外卖小程序，在首页查看、搜索生鲜，并点击生鲜图片可进入生鲜详情页面查看详情，并可将该生鲜分享至微信好友。
2. 收货地址管理：进入收货地址页面添加与删除收货地址，也可以进入查看所有自己的收货地址。
3. 订单管理：进入订单列表页面查看订单状态，在待付款列表取消自己某个订单或支付。用户在收到商家配送的生鲜之后，可确认收货。进行评价时可点击选择好、中、差评，并可编辑评价内容。
4. 购物车管理：用户在成功登录后可以在生鲜详情页面将其加入购物车，进入购物车页面可查看所有生鲜或删除某个生鲜。
5. 商家账号管理：商家登陆后进入账户管理，对管理员账号进行查看、添加、删除和修改。
6. 生鲜管理：商家登录后可以进入生鲜信息管理页面添加新的生鲜，也可以查看生鲜详细信息，还在生鲜信息列表删除某个生鲜以及对生鲜信息进行修改。
7. 用户管理：商家登录后进入用户管理可查看用户详细信息、删除某个用户或对用户信息进行修改。商家还可通过搜索用户名查找用户。
8. 财务管理：商家登录后可以进入财务管理查看订单信息与财务流水，在列表可查看不同状态订单，对已支付订单进行发货。
9. 统计管理：商家登录后可以进入统计管理查看每天的营收金额，并可按日期段搜索该时间段内营收情况。

# 4 数据库设计

## 4.1 概念结构设计

### 4.1.1 设计思路

生鲜外卖小程序的设计与实现的实体有生鲜实体、用户实体、收货地址实体、购物车实体、评论实体**、**商家管理员实体**、**订单实体、统计信息实体**。**

再通过功能需求来确定实体的属性

生鲜实体，介绍生鲜基本信息，生鲜的实拍图片以及详细介绍，属性有生鲜id,生鲜名字、生鲜图片、生鲜详情介绍、生鲜库存、生鲜分类、生鲜价格、生鲜库存。

用户实体，通过微信账号信息登录该生鲜外卖小程序的使用者，属性有用户id、用户昵称、用户性别、用户头像、状态。

收货地址实体，用户填写的生鲜收货地址，属性有收货地址id、联系人、手机号、地区、详细地址。

购物车实体，存放用户购物中准备结算的生鲜，属性有生鲜名称、生鲜数量、生鲜图片、生鲜价格。

评论实体，展示生鲜的用户评价信息，属性有评论id、评论内容、生鲜名称、用户昵称，用户头像、评分。

商家管理员实体，控制后台子系统、进行添加删除生鲜等后台操作的管理员，属性有管理员id、管理员名字、密码、手机号、邮箱。

订单实体，属性包括订单id、创建时间、支付时间、价格、生鲜信息、用户名、收货地址、订单状态。

统计信息实体，记录某时间段内的营收金额、售卖统计信息等，属性有信息id、日期、金额、售卖数量。

### 4.1.2 E-R图

各实体关系E-R图如图4-1所示：

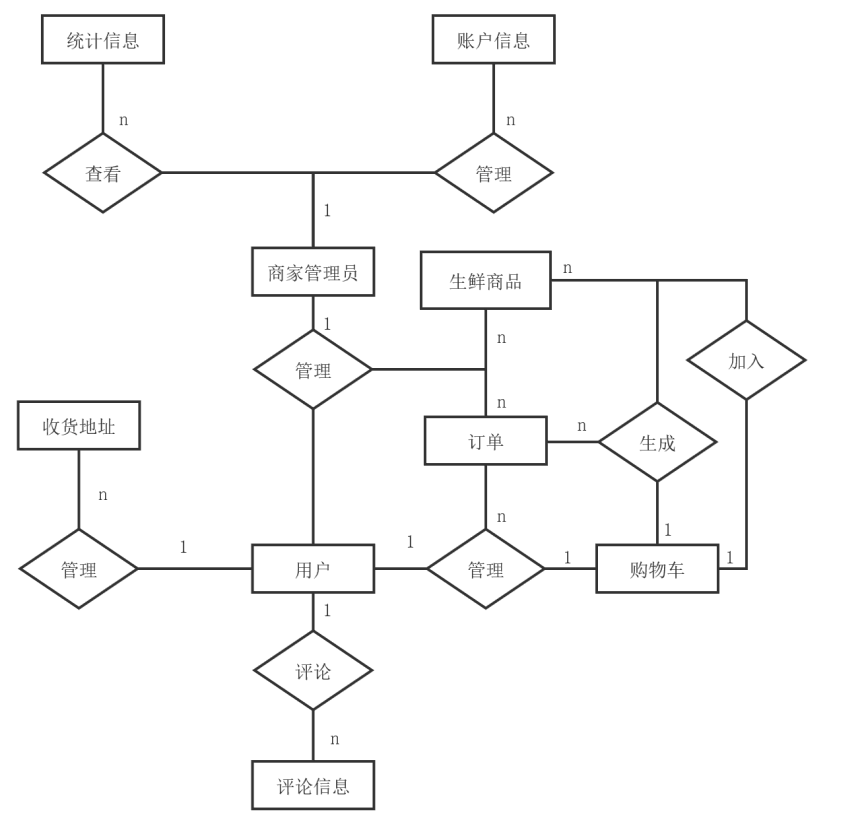


图 4-1 各实体关系E-R图

生鲜商品实体E-R图如图4-2所示：

实体：生鲜商品

属性：生鲜id,生鲜名称、生鲜图片、详情介绍、生鲜库存、生鲜分类、生鲜价格。



图 4-2 生鲜商品实体E-R图

用户实体E-R图如图4-3所示：

实体：用户

属性：用户id、用户昵称、用户性别、用户头像、状态。



图 4-3 用户信息实体E-R图

收货地址实体E-R图如图4-4所示：

实体：收货地址

属性：收货地址id、联系人、手机号、地区、详细地址。



图 4-4 收货地址实体E-R图

购物车实体E-R图如图4-5所示：

实体：购物车

属性：生鲜id、用户id、生鲜名称、生鲜数量、生鲜图片、生鲜价格。



图 4-5购物车实体E-R图

评论实体E-R图如图4-6所示：

实体：评论

属性：评论id、评论内容、所属商品、用户昵称，用户头像、评分。



图 4-6评论实体E-R图

商家管理员实体E-R用例图如图4-7所示：

实体：商家管理员

属性：管理员id、管理员姓名、登录名、密码、手机号、邮箱。



图 4-7商家管理员实体E-R图

订单实体E-R图如图4-8所示：

实体：订单

属性：订单id、创建时间、支付时间、价格、生鲜信息、用户名、收货地址、订单状态。



图 4-8订单信息实体E-R图

统计信息实体E-R图如图4-9所示：:

实体：统计信息

属性：信息id、日期、统计金额、统计商品数量数量。



图 4-9统计信息实体E-R图

## 4.2 逻辑结构设计

实体E-R图只能对存在于现实世界的事物类型做一个描述，这与数据库的各个表是不一样的，它只能表示各种实体的不同类型和它拥有的关联属性以及存在于各实体间的逻辑关系，这种抽象概念模型是一种从现实实体关系中得来的，所以要想对数据库进一步设计和实现，就需要将抽象概念模型转换为现实的关系数据库模型。

### 4.2.1 设计思路

首先根据功能需求来确认该小程序中有哪些数据库实体，再依据各实体的E-R图以及各实体关系E-R图来得出各数据库实体之间关系和它们具有哪些属性，最后是定义主键与外键，通常情况下主键是ID编号，定义外键则要根据数据实体之间不同的关系。

### 4.2.2 逻辑模型

下面是根据各实体E-R图和实体关系E-R图建立的生鲜外卖微信小程序的逻辑结构，下面是据此制作的各关系模型表如表4-1（逻辑关系表）。

生鲜商品表主要记录生鲜的详情信息，包括生鲜名称、生鲜图片、生鲜详情介绍、生鲜库存、生鲜分类、生鲜价格和生鲜id，表结构如表4-1（生鲜商品表）所示：

表4-1生鲜商品表（fresh\_goods）

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 序号 | 字段名 | 数据类型 | 字段长度 | 是否主键 | 说明 |
| 1 | Fid | 整型 | 10 | 是 | 生鲜id |
| 2 | Fname | 字符型 | 20 | 否 | 生鲜名称 |

续表4-1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 3 | Fclassify | 字符型 | 50 | 否 | 生鲜分类 |
| 4 | Fprice | 整型 | 10 | 否 | 生鲜价格 |
| 5 | Finventory | 整型 | 10 | 否 | 生鲜库存 |
| 6 | Fphoto | 字符型 | 50 | 否 | 生鲜图片 |
| 7 | Fdescription | 文本类型 | 50 | 否 | 详情介绍 |

用户表主要记录用户详情信息，包括用户id、用户昵称、用户性别、用户头像、状态，表结构如表4-2（用户表）所示：

表4-2用户表(user)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 序号 | 字段名 | 数据类型 | 字段长度 | 是否主键 | 说明 |
| 1 | Hid | 整型 | 10 | 是 | 用户id |
| 2 | Hname | 字符型 | 20 | 否 | 用户昵称 |
| 3 | Hphoto | 字符型 | 20 | 否 | 用户头像 |
| 4 | Hsex | 字符型 | 5 | 否 | 用户性别 |
| 5 | Hstate | 字符型 | 10 | 否 | 用户状态 |

收货地址表主要记录收货地址详情信息，包括收货地址id、联系人、手机号、地区、详细地址，表结构如表4-3（收货地址表）所示：

表4-3收货地址表(address)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 序号 | 字段名 | 数据类型 | 字段长度 | 是否主键 | 说明 |
| 1 | Aid | 字符型 | 10 | 是 | 地址id |
| 2 | Aname | 字符型 | 20 | 否 | 联系人 |
| 3 | Atelephone | 整型 | 11 | 否 | 手机号 |
| 4 | Aregion | 字符型 | 10 | 否 | 地区 |
| 5 | AddressDetail | 字符型 | 20 | 否 | 详细收货地址 |

购物车表主要记录用户购物车内准备结算的生鲜信息，包括用户id、生鲜id、生鲜名称、生鲜数量、生鲜图片、生鲜价格，表结构如表4-4（购物车表）所示：

表4-4购物车表(cart)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 序号 | 字段名 | 数据类型 | 字段长度 | 是否主键 | 说明 |
| 1 | Hid | 整型 | 10 | 是 | 用户id |
| 2 | Fname | 字符型 | 20 | 否 | 生鲜名称 |
| 3 | Fprice | 整型 | 10 | 否 | 生鲜价格 |
| 4 | Fnumber | 整型 | 10 | 否 | 生鲜数量 |

续表4-4

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 5 | Fphoto | 字符型 | 50 | 否 | 生鲜图片 |
| 6 | Fid | 整型 | 10 | 是 | 生鲜id |

评论表主要记录生鲜的用户评价信息，包括评论id、评论内容、所属商品、用户昵称，用户头像、评分，表结构如表4-5（评论表）所示：

表4-5评论表(comments)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 序号 | 字段名 | 数据类型 | 字段长度 | 是否主键 | 说明 |
| 1 | Pid | 整型 | 10 | 是 | 评论id |
| 2 | HName | 字符型 | 20 | 否 | 评论者名称 |
| 3 | Hphoto | 字符型 | 10 | 否 | 评论者头像 |
| 4 | Pscore | 整型 | 10 | 否 | 评分 |
| 5 | Fname | 字符型 | 20 | 否 | 所属商品 |
| 6 | Pdescription | 文本类型 | 50 | 否 | 评论内容 |

商家管理员表主要记录商家后台账户信息，包括管理员id、管理员姓名、密码、手机号、邮箱，表结构如表4-6（商家管理员表）所示：

表4-6商家管理员表(business\_admin)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 序号 | 字段名 | 数据类型 | 字段长度 | 是否主键 | 说明 |
| 1 | Uid | 整型 | 10 | 是 | 管理员id |
| 2 | Uname | 字符型 | 20 | 否 | 管理员姓名 |
| 3 | Upsword | 字符型 | 20 | 否 | 密码 |
| 4 | Uidentifiy | 字符型 | 10 | 否 | 登录名 |
| 5 | Utelephone | 整型 | 11 | 否 | 手机号 |
| 6 | Uemail | 字符型 | 20 | 否 | 邮箱 |

订单表主要记录用户下单后的订单信息，包括订单id、创建时间、支付时间、价格、生鲜信息、用户名、收货地址等，表结构如表4-7（订单表）所示：

表4-7订单表(order)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 序号 | 字段名 | 数据类型 | 字段长度 | 是否主键 | 说明 |
| 1 | Did | 整型 | 10 | 是 | 订单id |
| 2 | Dprice | 字符型 | 10 | 否 | 订单价格 |
| 3 | Dftime | 日期型 | 20 | 否 | 订单创建时间 |
| 4 | Dstime | 日期型 | 10 | 否 | 订单支付时间 |
| 5 | Fname | 字符型 | 10 | 否 | 生鲜名称 |

续表4-7

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 6 | Fnumber | 整型 | 10 | 否 | 订单生鲜数量 |
| 7 | Hname | 字符型 | 10 | 否 | 用户名 |
| 8 | Address | 字符型 | 20 | 否 | 收货地址 |

统计信息表主要记录用户下单后的订单信息，包括信息id、日期、金额、售卖数量，表结构如表4-8（订单表）所示：

表4-8统计信息表(statistics)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 序号 | 字段名 | 数据类型 | 字段长度 | 是否主键 | 说明 |
| 1 | Tid | 字符型 | 10 | 是 | 统计信息id |
| 2 | Tdate | 字符型 | 20 | 否 | 日期 |
| 3 | Tmoney | 整型 | 10 | 否 | 统计金额 |
| 4 | Fnumber | 整型 | 10 | 否 | 统计生鲜数量 |

## 4.3 物理结构设计

### 4.3.1 存取方式

通过分析生鲜外卖微信小程序的业务逻辑，对数据库的逻辑结构进行设计。根据数据库的逻辑结构确定使用MySQL数据库存储系统数据，整型数据类型具体化为Int类型，字符型数据类型具体化为Varchar，文本类型确定数据类型具体化为Text。

### 4.3.2 存储结构

通过上面的数据实体E-R图和逻辑关系模型表，进行物理结构设计分析，建立以下数据表。

生鲜商品物理表主要记录生鲜的详情信息，包括生鲜名称、生鲜图片、生鲜详情介绍、生鲜库存、生鲜分类、生鲜价格和生鲜id，表结构如表4-9所示：

表4-9生鲜商品物理表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 序号 | 字段名 | 数据类型 | 字段长度 | 是否主键 | 说明 |
| 1 | Fid | Int | 10 | 是 | 生鲜id |
| 2 | Fname | Varchar | 20 | 否 | 生鲜名称 |
| 3 | Fclassify | Varchar | 50 | 否 | 生鲜分类 |
| 4 | Fprice | Int | 10 | 否 | 生鲜价格 |
| 5 | Finventory | Int | 10 | 否 | 生鲜库存 |
| 6 | Fphoto | Varchar | 50 | 否 | 生鲜图片 |
| 7 | Fdescription | Text | 50 | 否 | 详情介绍 |

用户物理表主要记录用户详情信息，包括用户id、用户昵称、用户性别、用户头像、状态，表结构如表4-10所示：

表4-10用户物理表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 序号 | 字段名 | 数据类型 | 字段长度 | 是否主键 | 说明 |
| 1 | Hid | Int | 10 | 是 | 用户id |
| 2 | Hname | Varchar | 20 | 否 | 用户昵称 |
| 3 | Hphoto | Varchar | 20 | 否 | 用户头像 |
| 4 | Hsex | Varchar | 5 | 否 | 用户性别 |
| 5 | Hstate | Varchar | 10 | 否 | 用户状态 |

收货地址物理表主要记录收货地址详情信息，包括收货地址id、联系人、手机号、地区、详细地址，表结构如表4-11所示：

表4-11收货地址物理表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 序号 | 字段名 | 数据类型 | 字段长度 | 是否主键 | 说明 |
| 1 | Aid | Varchar | 10 | 是 | 地址id |
| 2 | Aname | Varchar | 20 | 否 | 联系人 |
| 3 | Atelephone | Int | 11 | 否 | 手机号 |
| 4 | Aregion | Varchar | 10 | 否 | 地区 |
| 5 | Addtrss | Varchar | 20 | 否 | 详细收货地址 |

购物车物理表主要记录购物车内准备结算的生鲜信息，包括生鲜名称、生鲜数量、生鲜图片、生鲜价格，表结构如表4-12所示：

表4-12购物车物理表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 序号 | 字段名 | 数据类型 | 字段长度 | 是否主键 | 说明 |
| 1 | Fid | Int | 10 | 是 | 生鲜id |
| 2 | Fname | Varchar | 20 | 否 | 生鲜名称 |
| 3 | Fclassify | Varchar | 50 | 否 | 生鲜分类 |
| 4 | Fprice | Int | 10 | 否 | 生鲜价格 |
| 5 | Finventory | Int | 10 | 否 | 生鲜库存 |
| 6 | Fphoto | Varchar |  | 否 | 生鲜图片 |
| 7 | Fdescription | Varchar | 50 | 否 | 详情介绍 |

评论物理表主要记录生鲜的用户评价信息，包括评论id、评论内容、生鲜名字、用户昵称，用户头像、评分，表结构如表4-13所示：

表4-13评论物理表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 序号 | 字段名 | 数据类型 | 字段长度 | 是否主键 | 说明 |
| 1 | Pid | Int | 10 | 是 | 评论id |
| 2 | HName | Varchar | 20 | 否 | 评论者名称 |
| 3 | Hphoto | Varchar | 10 | 否 | 评论者头像 |
| 4 | Pscore | Int | 10 | 否 | 评分 |
| 5 | Fname | Varchar | 20 | 否 | 生鲜名称 |
| 6 | Pdescription | text | 50 | 否 | 评论内容 |

商家管理员物理表主要记录商家后台账户信息，包括管理员id、管理员名字、密码、手机号、邮箱，表结构如表4-14所示：

表4-14商家管理员物理表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 序号 | 字段名 | 数据类型 | 字段长度 | 是否主键 | 说明 |
| 1 | Uid | Int | 10 | 是 | 管理员id |
| 2 | Uname | Varchar | 20 | 否 | 管理员名字 |
| 3 | Upsword | Varchar | 20 | 否 | 密码 |
| 4 | Uidentifiy | Varchar | 10 | 否 | 登录名 |
| 5 | Utelephone | Int | 11 | 否 | 手机号 |
| 6 | Uemail | Varchar | 20 | 否 | 邮箱 |

订单物理表主要记录用户下单后的订单信息，包括订单id、创建时间、支付时间、价格、生鲜信息、用户名、收货地址、订单状态，表结构如表4-15所示：

表4-15订单物理表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 序号 | 字段名 | 数据类型 | 字段长度 | 是否主键 | 说明 |
| 1 | Did | Int | 10 | 是 | 订单id |
| 2 | Dprice | Varchar | 10 | 否 | 订单价格 |
| 3 | Dftime | datetime | 20 | 否 | 订单创建时间 |
| 4 | Dstime | datetime | 10 | 否 | 订单支付时间 |
| 5 | Fname | Varchar | 10 | 否 | 订单生鲜名称 |
| 6 | Fnumber | Int | 10 | 否 | 订单生鲜数量 |
| 7 | Hname | Varchar | 10 | 否 | 会员名 |
| 8 | Address | Varchar | 20 | 否 | 收货地址 |

统计信息物理表主要记录用户下单后的订单信息，包括信息id、日期、金额、售卖数量，表结构如表4-16所示：

表4-16统计信息物理表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 序号 | 字段名 | 数据类型 | 字段长度 | 是否主键 | 说明 |
| 1 | Tid | Varchar | 10 | 是 | 信息id |
| 2 | Tdate | Varchar | 20 | 否 | 日期 |
| 3 | Tmoney | Int | 10 | 否 | 统计金额 |
| 4 | Fnumber | Int | 10 | 否 | 订单生鲜数量 |

1. 界面设计

## 5.1界面关系图

用户进入小程序后，授权进行登录，在生鲜页面进行生鲜选购，进入购物车页面，可实现对加购生鲜进行管理，进入个人中心页面，可查看并管理地址、订单和评论。生鲜外卖小程序端界面关系图如图5-1所示：

图5-1生鲜外卖小程序端界面关系图

商家通过账号密码登录管理后台，进入主页后可根据操作需求的不同，进入账号、生鲜、会员、财务等管理页面。生鲜外卖管理后台界面关系图如图5-2所示：

图5-2生鲜外卖管理后台界面关系图

## 5.2 界面设计成果

### 5.2.1 主界面

生鲜外卖小程序首页在底部分类栏上展示了首页、购物车和个人中心三个栏目信息，用户可通过直接点击进入到不同栏目页面中。首页中还具体展示了今日特惠、多买优惠、时令水果、新鲜蔬菜、牛羊猪肉等生鲜分类信息。生鲜外卖小程序主界面如图5-3所示：



图5-3小程序主界面

后台管理主页面在侧边栏展示了不同的管理模块等栏目信息，商家管理员可通过直接点击进入不同栏目页面中。后台管理子系统主界面如图5-4所示：

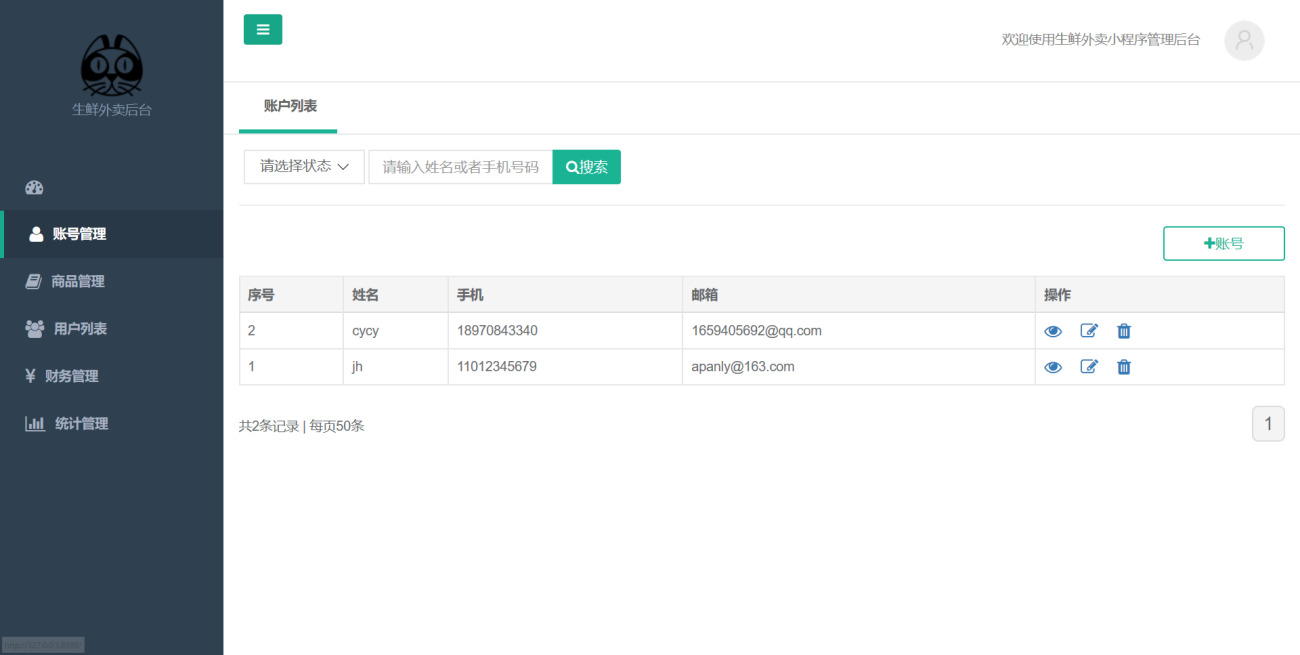


图5-4后台管理主界面

### 5.2.2 子界面

生鲜详情页面展示了生鲜图片、生鲜标题、生鲜价格、生鲜介绍、用户评价等信息，以及在下方栏目上固定展示了“加入购物车”以及“立即购买”等按钮图形，实现用户对生鲜加购及下单购买等操作，生鲜详情子界面如图5-5所示：



图5-5生鲜详情子界面

购物车界面展示了用户加入购物车的生鲜信息，如生鲜图片、生鲜名、生鲜价格和生鲜数量等信息，以及在下方栏目上固定展示了“去结算”等按钮图形，实现用户对生鲜下单购买等操作，购物车子界面如图5-6所示：

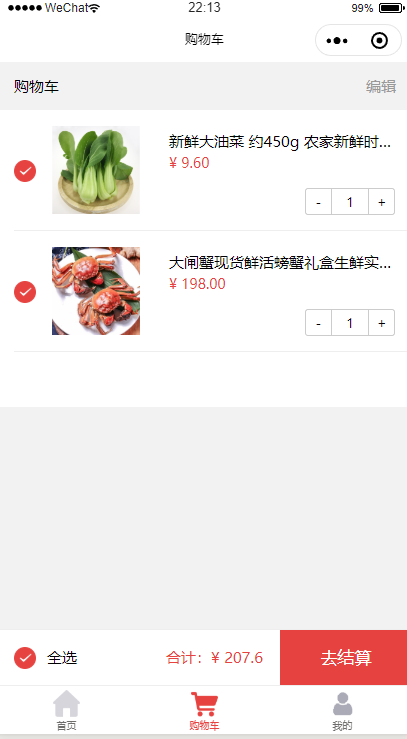


图5-6购物车子界面

订单详情页面在顶部展示了待付款、待发货、待确认、待评价等栏目信息，在下方展示了订单生鲜信息、订单价格等订单信息，订单子界面如图5-7所示：



图5-7订单详情子界面

收货地址界面展示了收货人姓名、手机号、详细收货地址等信息。在底部展示了新增地址栏目，用户可点击进入地址新增页面，填写新的地址信息。收货地址子界面如图5-8所示：



图5-8收货地址子界面

生鲜管理页面在顶部展示了生鲜列表及分类列表两个栏目信息，通过点击进入可查看生鲜信息列表及分类信息列表。生鲜信息列表展示了生鲜名、分类、价格、库存、入库信息等生鲜相关信息，生鲜管理子界面如图5-9所示：



图5-9生鲜管理子界面

用户列表页面在顶部展示了用户列表及用户评论两个栏目信息，通过点击进入可查看用户信息列表及评论信息列表。用户信息列表展示了用户头像、姓名等用户相关信息，用户管理子界面如图5-10所示：

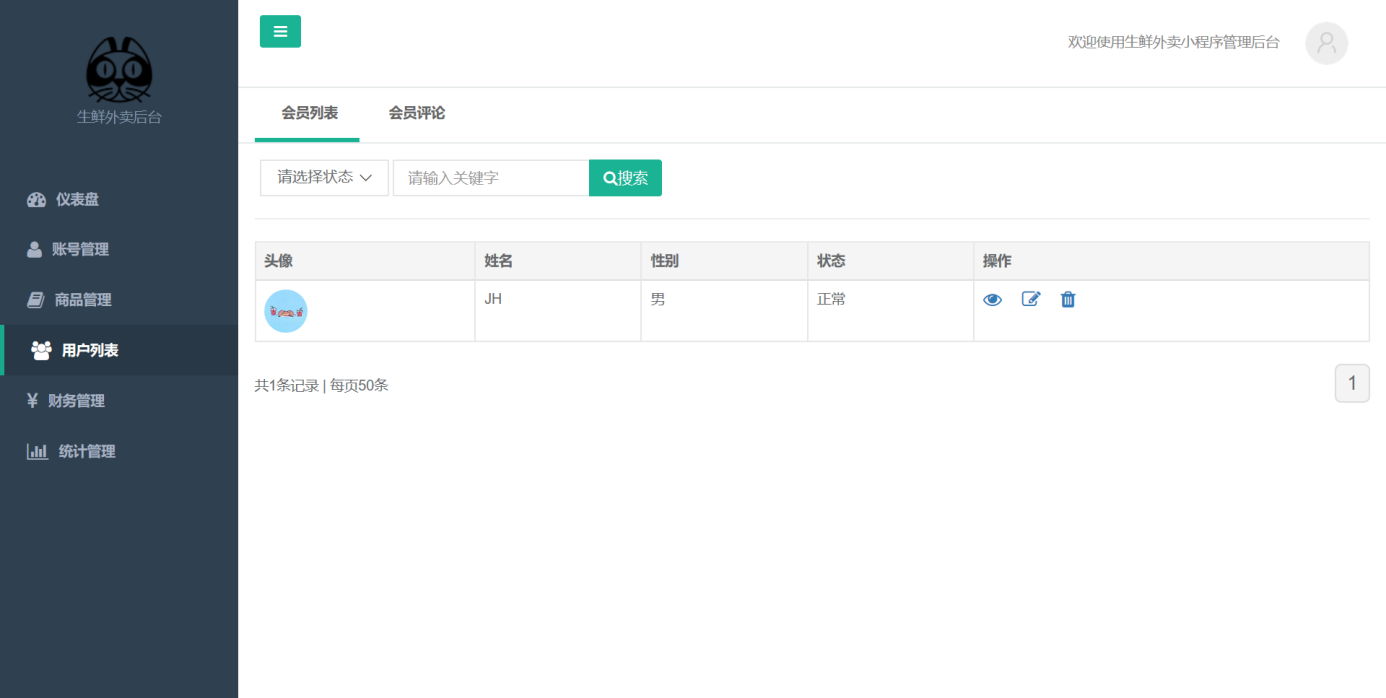


图5-10用户管理子界面

6 详细设计

## 6.1系统主要功能模块介绍

1）生鲜选购功能，主要是用户进行对生鲜的浏览，对生鲜的搜索，对生鲜的分享的操作。用户可通过分类来查看生鲜，通过关键字搜索想要的生鲜，进入生鲜详情页面查看生鲜的详细信息，并可将该生鲜分享至微信好友。

2）收货地址管理功能，主要是进入收货地址页面添加与删除收货地址，也可以进入查看所有自己的收货地址。

3）订单管理功能，主要是用户在成功登录后可以在生鲜详情页面将其加入购物车，进入购物车页面可查看所有生鲜或删除某个生鲜。

4） 购物车管理功能，主要是用户在成功登录后可以在生鲜详情页面将其加入购物车，进入购物车页面可查看所有生鲜或删除某个生鲜。

5）商家账号管理功能，主要是商家登陆后进入账户管理，对管理员账号进行查看、添加、删除和修改。

6）生鲜管理功能，主要是商家登录后可以进入生鲜信息管理页面添加新的生鲜，也可以查看生鲜详细信息，还在生鲜信息列表删除某个生鲜以及对生鲜信息进行修改。

7）用户管理功能，主要是商家登录后进入用户管理可查看用户详细信息、删除某个用户或对用户信息进行修改。商家还可通过搜索用户名查找用户。

8）财务管理功能，主要是商家登录后可以进入财务管理查看订单信息与财务流水，在列表可查看不同状态订单，对已支付订单进行发货。

9）统计管理功能，主要是商家登录后可以进入统计管理查看每天的营收金额，并可按日期段搜索该时间段内营收情况。

6.1生鲜选购

生鲜选购功能，主要是用户进行对生鲜的浏览，对生鲜的搜索，对生鲜的分享的操作。用户可通过分类来查看生鲜，通过关键字搜索想要的生鲜，并点击生鲜可进入生鲜详情页面查看生鲜的详细信息，并可将该生鲜分享至微信好友。

6.1.1 功能结构设计

生鲜选购功能结构图如图6-1所示：



图6-1生鲜选购功能结构图

“生鲜选购”功能主要目的是登录小程序后向买家显示卖家提供的生鲜的功能；此外，该功能还负责将用户搜索想要的生鲜以及将选中的生鲜分享至微信好友。因此，生鲜选购模块主要涉及搜索生鲜和浏览生鲜以及分享生鲜三个组成部分。

6.1.2 类图设计

生鲜选购类图如图6-2所示：



图6-2生鲜选购类图

生鲜选购界面类描述如表6-1所示：

表6-1生鲜选购界面类描述

|  |  |  |
| --- | --- | --- |
| 生鲜选购界面类 | | |
| 类的描述：负责展示生鲜，根据用户的搜索推荐出对应生鲜，并可分享生鲜 | | |
| 操作描述 | GetGoodsList() | 获取生鲜列表 |
| SerachGoods() | 根据关键字搜索生鲜 |
| ShareGoods() | 分享生鲜 |

生鲜选购控制器类描述如表6-2所示：

表6-2生鲜选购控制器类描述

|  |  |  |
| --- | --- | --- |
| 生鲜选购控制器类 | | |
| 类的描述：根据前台的业务逻辑不同跳转到对应不同方法，通过控制器类再跳转到前台展示 | | |
| 操作描述 | GetALLGoods() | 获取搜索的所有生鲜 |
| GoodsDisplay() | 展示所有生鲜 |
| GetGoodsDtails() | 展示生鲜详细信息 |
| ShareGoods(); | 分享生鲜 |

生鲜选购业务层接口类描述如表6-3所示：

表6-3生鲜选购业务层接口类描述

|  |  |  |
| --- | --- | --- |
| 生鲜选购业务层接口类 | | |
| 类的描述：生鲜选购展示、检索和分享方法的接口类 | | |
| 操作描述 | GetGoodsByid(); | 通过生鲜id获取生鲜实体对象 |
| GetGoodsBycategory() | 根据生鲜类别获取生鲜 |
| SearchByKeyword() | 根据关键字检索生鲜 |
| ShareGoodstofriends() | 分享生鲜至微信好友 |

生鲜选购业务层接口类描述如表6-4所示：

表6-4生鲜选购业务层实现类描述

|  |  |  |
| --- | --- | --- |
| 生鲜选购业务层实现类 | | |
| 类的描述：生鲜选购业务层实现类，实现业务层接口类的方法 | | |
| 操作描述 | GetGoodsByid(); | 通过生鲜id获取生鲜实体对象的实现方法 |
| GetGoodsBycategory() | 根据生鲜类别获取生鲜的实现方法 |
| SearchByKeyword() | 根据关键字检索生鲜的实现方法 |
| ShareGoodstofriends() | 分享生鲜至微信好友的实现方法 |

生鲜选购数据访问层类描述如表6-5所示：

表6-5生鲜选购数据访问层类描述

|  |  |  |
| --- | --- | --- |
| 生鲜选购数据访问层类 | | |
| 类的描述：生鲜选购数据访问层类 | | |
| 操作描述 | GetGoodsByid(); | 通过生鲜id获取生鲜实体对象 |
| GetGoodsBycategory() | 根据生鲜类别获取生鲜 |
| SearchByKeyword() | 根据关键字检索生鲜 |
| ShareGoodstofriends() | 分享生鲜至微信好友 |

生鲜实体类描述如表6-6所示：

表6-6生鲜实体类描述

|  |  |  |
| --- | --- | --- |
| 生鲜实体类 | | |
| 类的描述：生鲜实体类 | | |
| 属性描述 | Fid | 生鲜id |
| Fname | 生鲜名称 |
| Fclassify | 生鲜分类 |
| Fprice | 生鲜价格 |
| Finventory | 生鲜库存 |
| Fphoto | 生鲜图片 |
| Fdescription | 详情介绍 |

6.1.3 顺序图设计

用户搜索生鲜活动顺序图如图6-3所示：



图6-3生鲜选购顺序图

6.1.4 核心处理流程设计

生鲜选购中搜索生鲜功能活动流程图如图6-4所示：



图6-4搜索生鲜流程图

**6.2收货地址管理**

6.2.1功能结构设计

收货地址管理功能结构图如图6-5所示：



图6-5收货地址管理功能结构图

收货地址管理功能，主要是进入收货地址页面添加与删除收货地址，也可以进入查看所有自己的收货地址。

6.2.2类图设计

收货地址管理类图如图6-6所示：



图6-6收货地址管理类图

收货地址界面类描述如表6-7所示：

表6-7收货地址界面类描述

|  |  |  |
| --- | --- | --- |
| 收货地址界面类 | | |
| 类的描述：负责收货地址预览界面的前台展示 | | |
| 操作描述 | AddressDisplay () | 展示获取的收货地址列表 |
| AddressDetailDisplay () | 展示单一收货地址详细信息 |

收货地址管理控制器类描述如表6-8所示：

表6-8收货地址管理控制器类描述

|  |  |  |
| --- | --- | --- |
| 收货地址管理控制器类 | | |
| 类的描述：根据前台的业务逻辑不同跳转到对应不同方法，通过控制器类再跳转到前台展示 | | |
| 操作描述 | GetAllAddress() | 获取所有收货地址信息 |
| Add\_address() | 增加收货地址 |
| Del\_address() | 删除收货地址 |
| Modify\_address() | 修改收货地址信息 |

收货地址管理业务层接口类描述如表6-9所示：

表6-9收货地址管理业务层接口类描述

|  |  |  |
| --- | --- | --- |
| 收货地址管理业务层接口类 | | |
| 类的描述：为业务层类提供接口方法 | | |
| 操作描述 | GetAddressBYid() | 通过id获取收货地址实体对象 |
| AddAddressDetail() | 添加收货地址详细信息 |
| DelAddressDetail() | 删除收货地址详细信息 |
| ModifyAddressDetail() | 修改收货地址详细信息 |

收货地址管理业务层实现类描述如表6-10所示：

表6-10收货地址管理业务层实现类描述

|  |  |  |
| --- | --- | --- |
| 收货地址管理业务层实现类 | | |
| 类的描述：实现业务层类提供的接口方法 | | |
| 操作描述 | GetAddressBYid() | 通过id获取收货地址实体对象的实现方法 |
| AddAddressDetail() | 添加收货地址详细信息的实现方法 |
| DelAddressDetail() | 删除收货地址详细信息的实现方法 |
| ModifyAddressDetail() | 修改收货地址详细信息的实现方法 |

收货地址管理数据层类描述如表6-11所示：

表6-11收货地址管理数据层类描述

|  |  |  |
| --- | --- | --- |
| 收货地址管理数据层类 | | |
| 类的描述：为业务层方法的实现提供数据层的支持 | | |
| 操作描述 | GetAddressBYid() | 通过id获取收货地址实体对象 |
| AddAddressDetail() | 添加收货地址详细信息 |
| DelAddressDetail() | 删除收货地址详细信息 |
| ModifyAddressDetail() | 修改收货地址详细信息 |

收货地址实体类描述如表6-12所示：

表6-12收货地址实体类描述

|  |  |  |
| --- | --- | --- |
| 收货地址实体类 | | |
| 类的描述：收货地址实体类 | | |
| 属性描述 | Aid | 地址id |
| Aname | 联系人 |
| Atelephone | 手机号 |
| Aregion | 地区 |
| AddressDetail | 详细收货地址 |

6.2.3顺序图设计

用户添加地址活动顺序图如图6-7所示：



图6-7收货地址管理顺序图

6.2.4核心处理流程设计

收货地址管理中添加地址活动图如图6-8所示：



图6-8收获地址管理活动图

6.3订单管理

6.3.1功能结构设计

订单管理功能结构图如图6-9所示：



图6-9订单管理功能结构图

订单管理功能，主要是用户在成功登录后可以在生鲜详情页面将其加入购物车，进入购物车页面可查看所有生鲜或删除某个生鲜。

6.3.2 类图设计

订单管理类图如图6-10所示：



图6-10订单管理类图

订单管理界面类描述如表6-13所示：

表6-13订单管理界面类描述

|  |  |  |
| --- | --- | --- |
| 订单管理界面类 | | |
| 类的描述：负责订单信息预览界面的前台展示 | | |
| 操作描述 | OrderDisplay () | 展示获取的订单列表 |
| OrderDetailDisplay () | 展示单一订单详细信息 |

订单管理控制器类描述如表6-14所示：

表6-14订单管理控制器类描述

|  |  |  |
| --- | --- | --- |
| 订单管理控制器类 | | |
| 类的描述：根据前台的业务逻辑不同跳转到对应不同方法，通过控制器类再跳转到前台展示 | | |
| 操作描 | GetAllOrder() | 获取所有订单信息 |
| Create\_Order() | 创建订单 |
| Del\_Order() | 取消订单 |

续表6-14

|  |  |  |
| --- | --- | --- |
| 述 | Pay\_Order() | 支付订单 |
| Comfirm\_Order() | 确认订单 |
| Evalute\_Order() | 评价订单 |

订单管理业务层接口类描述如表6-15所示：

表6-15订单管理业务层接口类描述

|  |  |  |
| --- | --- | --- |
| 订单管理业务层接口类 | | |
| 类的描述：为业务层类提供接口方法 | | |
| 操作描述 | GetOrderBYid() | 通过id获取订单实体对象 |
| GetAddressBYid() | 通过id获取收货地址实体对象 |
| CreateOrderDetail() | 创建订单详细信息 |
| DelOrderDetail() | 删除订单详细信息 |
| Pay\_Order() | 支付订单 |
| Comfirm\_Order() | 确认订单 |
| EvaluateOrderdetail() | 添加订单评价详情 |

订单管理业务层实现类描述如表6-16所示：

表6-16订单管理业务层实现类描述

|  |  |  |
| --- | --- | --- |
| 订单管理业务层实现类 | | |
| 类的描述：实现业务层类提供的接口方法 | | |
| 操作描述 | GetOrderBYid() | 通过id获取订单实体对象的实现方法 |
| GetAddressBYid() | 通过id获取收货地址实体对象的实现方法 |
| CreateOrderDetail() | 创建订单详细信息的实现方法 |
| DelOrderDetail() | 删除订单详细信息的实现方法 |
| Pay\_Order() | 支付订单的实现方法 |
| Comfirm\_Order() | 确认订单的实现方法 |
| EvaluateOrderdetail() | 添加订单评价详情的实现方法 |

订单管理数据层类描述如表6-17所示：

表6-17订单管理数据层类描述

|  |  |  |
| --- | --- | --- |
| 订单管理数据层类 | | |
| 类的描述：为业务层方法的实现提供数据层的支持 | | |
| 操作描述 | GetOrderBYid() | 通过id获取订单实体对象 |
| GetAddressBYid() | 通过id获取收货地址实体对象 |
| CreateOrderDetail() | 创建订单详细信息 |
| DelOrderDetail() | 删除订单详细信息 |
| Pay\_Order() | 支付订单 |

续表6-17

|  |  |  |
| --- | --- | --- |
|  | Comfirm\_Order() | 确认订单 |
|  | EvaluateOrderdetail() | 添加订单评价详情 |

收货地址实体类描述如表6-18所示：

表6-18收货地址实体类描述

|  |  |  |
| --- | --- | --- |
| 收货地址实体类 | | |
| 类的描述：收货地址实体类 | | |
| 属性描述 | Aid | 地址id |
| Aname | 联系人 |
| Atelephone | 手机号 |
| Aregion | 地区 |
| AddressDetail | 详细收货地址 |

订单实体类描述如表6-19所示：

表6-19订单实体类描述

|  |  |  |
| --- | --- | --- |
| 订单实体类 | | |
| 类的描述：订单实体类 | | |
| 属性描述 | Did | 订单id |
| Dprice | 订单价格 |
| Dftime | 订单创建时间 |
| Dstime | 订单支付时间 |
| Fname | 订单生鲜名称 |
| Fnumber | 订单生鲜数量 |
| Hname | 会员名 |
| Address | 收货地址 |

6.3.3顺序图设计

用户评论订单活动顺序图如图6-11所示：



图6-11订单管理顺序图

6.3.4核心处理流程设计

订单管理中评论订单活动流程图如图6-12所示：



图6-12订单管理流程图

6.4购物车管理

6.4.1功能结构设计

购物车管理功能结构图如图6-13所示：



图6-13购物车管理功能结构图

购物车管理主要是用户在成功登录后可以在生鲜详情页面将其加入购物车，进入购物车页面可查看所有生鲜或删除某个生鲜。

6.4.2类图设计

购物车管理类图如图6-14所示：



图6-14购物车管理类图

购物车界面类描述如表6-20所示：

表6-20购物车界面类描述

|  |  |  |
| --- | --- | --- |
| 购物车界面类 | | |
| 类的描述：负责收货地址预览界面的前台展示 | | |
| 操作描述 | CartGoodsDisplay() | 展示获取的购物车生鲜列表 |
| CartGoodsDetailDisplay() | 展示购物车单一生鲜详细信息 |

购物车管理控制器类描述如表6-21所示：

表6-21购物车管理控制器类描述

|  |  |  |
| --- | --- | --- |
| 购物车管理控制器类 | | |
| 类的描述：根据前台的业务逻辑不同跳转到对应不同方法，通过控制器类再跳转到前台展示 | | |
| 操作描述 | GetALLCartGoods() | 获取所有购物车生鲜信息 |
| SetCartGoodsNumber() | 设置购物车内单个生鲜的数量 |
| AddGoodstoCart() | 添加生鲜至购物车 |
| DelGoodsformCart() | 把生鲜从购物车内移除 |

购物车管理业务层接口类描述如表6-22所示：

表6-22购物车管理业务层接口类描述

|  |  |  |
| --- | --- | --- |
| 购物车管理业务层接口类 | | |
| 类的描述：为业务层类提供接口方法 | | |
| 操作描述 | GetCartByUserid() | 通过用户id获取购物车实体对象 |
| UpdateCartRecord() | 更新购物车记录 |
| AddCartRecord() | 添加购物车记录 |
| DelCartRecord() | 删除购物车记录 |

购物车管理业务层实现类描述如表6-23所示：

表6-23收货地址管理业务层实现类描述

|  |  |  |
| --- | --- | --- |
| 购物车管理业务层实现类 | | |
| 类的描述：实现业务层类提供的接口方法 | | |
| 操作描述 | GetCartByUserid() | 通过用户id获取购物车实体的实现方法 |
| UpdateCartRecord() | 更新购物车记录的实现方法 |
| AddCartRecord() | 添加购物车记录的实现方法 |
| DelCartRecord() | 删除购物车记录的实现方法 |

购物车管理数据层类描述如表6-24所示：

表6-24购物车管理数据层类描述

|  |  |  |
| --- | --- | --- |
| 购物车管理数据层类 | | |
| 类的描述：为业务层方法的实现提供数据层的支持 | | |
| 操作描述 | GetCartByUserid() | 通过用户id获取购物车实体对象 |
| UpdateCartRecord() | 更新购物车记录 |
| AddCartRecord() | 添加购物车记录 |
| DelCartRecord() | 删除购物车记录 |

购物车实体类描述如表6-25所示：

表6-25购物车实体类描述

|  |  |  |
| --- | --- | --- |
| 购物车实体类 | | |
| 类的描述：购物车实体类 | | |
| 属性描述 | Hid | 用户id |
| Fname | 生鲜名称 |
| Fprice | 生鲜价格 |
| Fnumber | 生鲜数量 |
| Fphoto | 生鲜图片 |
| Fid | 生鲜id |

6.4.3顺序图设计

用户添加生鲜至购物车活动顺序图如图6-15所示：



图6-15购物车管理顺序图

6.4.4核心处理流程设计

购物车管理中生鲜加入购物车活动流程图如图6-16所示：



图6-16购物车管理活动图

6.5商家账号管理

6.5.1功能结构设计

商家账号管理功能结构图如图6-17所示：



图6-17商家账号管理功能结构图

商家账号管理功能，主要是商家登陆后进入账户管理，对管理员账号进行查看、添加、删除和修改。

6.5.2 类图设计

商家账号管理类图如图6-18所示：



图6-18商家账号管理类图

商家账号管理界面类描述如表6-26所示：

表6-26商家账号管理界面类描述

|  |  |  |
| --- | --- | --- |
| 商家账号管理界面类 | | |
| 类的描述：负责商家账号管理预览界面的前台展示 | | |
| 操作描述 | AdminiAccountDisplay() | 展示获取的商家账号列表 |
| AccountDetailDisplay () | 展示单一商家账号详细信息 |

商家账号管理控制器类描述如表6-27所示：

表6-27商家账号管理控制器类描述

|  |
| --- |
| 商家账号管理控制器类 |
| 类的描述：根据前台的业务逻辑不同跳转到对应不同方法，通过控制器类再跳转到前台展示 |

续表6-27

|  |  |  |
| --- | --- | --- |
| 操作描述 | GetAllAccount() | 获取所有商家账号信息 |
| Add\_account() | 增加商家账号 |
| Del\_account() | 删除商家账号 |
| Modify\_account() | 修改商家账号信息 |
| Search\_Account() | 检索商家账号 |

商家账号管理管理业务层接口类描述如表6-28所示：

表6-28商家账号管理业务层接口类描述

|  |  |  |
| --- | --- | --- |
| 商家账号管理业务层接口类 | | |
| 类的描述：为业务层类提供接口方法 | | |
| 操作描述 | GetAccountBYid() | 通过id获取商家账号实体对象 |
| AddAccountDetail() | 添加商家账号详细信息 |
| DelAccountDetail() | 删除商家账号详细信息 |
| ModifyAccountDetail() | 修改商家账号详细信息 |
| SearchByKeyword() | 通过关键字检索商家账号 |

商家账号管理业务层实现类描述如表6-29所示：

表6-29商家账号管理业务层实现类描述

|  |  |  |
| --- | --- | --- |
| 商家账号管理业务层实现类 | | |
| 类的描述：实现业务层类提供的接口方法 | | |
| 操作描述 | GetAccountBYid() | 通过id获取商家账号实体对象的实现方法 |
| AddAccountDetail() | 添加商家账号详细信息的实现方法 |
| DelAccountDetail() | 删除商家账号详细信息的实现方法 |
| ModifyAccountDetail() | 修改商家账号详细信息的实现方法 |
| SearchByKeyword() | 通过关键字检索商家账号的实现方法 |

商家账号管理数据层类描述如表6-30所示：

表6-30商家账号管理数据层类描述

|  |  |  |
| --- | --- | --- |
| 商家账号管理数据层类 | | |
| 类的描述：为业务层方法的实现提供数据层的支持 | | |
| 操作描述 | GetAccountBYid() | 通过id获取商家账号实体对象 |
| AddAccountDetail() | 添加商家账号详细信息 |
| DelAccountDetail() | 删除商家账号详细信息 |
| ModifyAccountDetail() | 修改商家账号详细信息 |
| SearchByKeyword() | 通过关键字检索商家账号 |

商家管理员实体类描述如表6-31所示：

表6-31商家管理员实体类描述

|  |  |  |
| --- | --- | --- |
| 商家管理员实体类 | | |
| 类的描述：商家管理员实体类 | | |
| 属性描述 | Uid | 管理员id |
| Uname | 管理员姓名 |
| Upsword | 密码 |
| Uidentifiy | 登录名 |
| Utelephone | 手机号 |
| Uemail | 邮箱 |

6.5.3顺序图设计

商家管理员修改后台账号信息活动顺序图如图6-19所示：



图6-19商家账号管理顺序图

6.5.4核心处理流程设计

商家账号管理中修改账号信息活动流程图如图6-20所示：



图6-20商家账号管理活动图

6.6生鲜管理

6.6.1功能结构设计

生鲜管理功能结构图如图6-21所示：



图6-21生鲜管理功能结构图

生鲜管理功能，主要是商家登录后可以进入生鲜信息管理页面添加新的生鲜，也可以查看生鲜详细信息，还在生鲜信息列表删除某个生鲜以及对生鲜信息进行修改。

6.6.2 类图设计

生鲜管理类图如图6-22所示：



图6-22生鲜管理类图

生鲜管理界面类描述描述如表6-32所示：

表6-32生鲜管理界面类描述

|  |  |  |
| --- | --- | --- |
| 生鲜管理界面类 | | |
| 类的描述：负责生鲜管理预览界面的前台展示 | | |
| 操作描述 | GoodsDisplay() | 展示获取的生鲜列表 |
| GoodsDetailDisplay () | 展示单一生鲜详细信息 |

生鲜管理控制器类描述如表6-33所示：

表6-33生鲜管理控制器类描述

|  |  |  |
| --- | --- | --- |
| 生鲜管理控制器类 | | |
| 类的描述：根据前台的业务逻辑不同跳转到对应不同方法，通过控制器类再跳转到前台展示 | | |
| 操作 | GetAllGoods() | 获取所有生鲜信息 |
| Add\_Goods() | 上架生鲜 |

续表6-33

|  |  |  |
| --- | --- | --- |
| 描述 | Del\_Goods() | 下架生鲜 |
| Modify\_Goods() | 修改生鲜详细信息 |
| Search\_Goods() | 检索生鲜 |

生鲜管理业务层接口类描述如表6-34所示：

表6-34生鲜管理业务层接口类描述

|  |  |  |
| --- | --- | --- |
| 生鲜管理业务层接口类 | | |
| 类的描述：为业务层类提供接口方法 | | |
| 操作描述 | GetGoodsBYid() | 通过id获取生鲜实体对象 |
| AddGoodsDetail() | 添加生鲜详细信息 |
| DelGoodsDetail() | 删除生鲜详细信息 |
| ModifyGoodsDetail() | 修改生鲜详细信息 |
| SearchGoodsByKeyword() | 通过关键字检索生鲜 |

生鲜管理业务层实现类描述如表6-35所示：

表6-35生鲜管理业务层实现类描述

|  |  |  |
| --- | --- | --- |
| 生鲜管理业务层实现类 | | |
| 类的描述：实现业务层类提供的接口方法 | | |
| 操作描述 | GetGoodsBYid() | 通过id获取生鲜实体对象的实现方法 |
| AddGoodsDetail() | 添加生鲜详细信息的实现方法 |
| DelGoodsDetail() | 删除生鲜详细信息的实现方法 |
| ModifyGoodsDetail() | 修改生鲜详细信息的实现方法 |
| SearchGoodsByKeyword() | 通过关键字检索生鲜的实现方法 |

生鲜管理数据层类描述如表6-36所示：

表6-36生鲜管理数据层类描述

|  |  |  |
| --- | --- | --- |
| 生鲜管理数据层类 | | |
| 类的描述：为业务层方法的实现提供数据层的支持 | | |
| 操作描述 | GetGoodsBYid() | 通过id获取生鲜实体对象 |
| AddGoodsDetail() | 添加生鲜详细信息 |
| DelGoodsDetail() | 删除生鲜详细信息 |
| ModifyGoodsDetail() | 修改生鲜详细信息 |
| SearchGoodsByKeyword() | 通过关键字检索生鲜 |

生鲜实体类描述如表6-37所示：

表6-37生鲜实体类描述

|  |
| --- |
| 生鲜实体类 |
| 类的描述：生鲜实体类 |

续表6-37

|  |  |  |
| --- | --- | --- |
| 属性描述 | Fid | 生鲜id |
| Fname | 生鲜名称 |
| Fclassify | 生鲜分类 |
| Fprice | 生鲜价格 |
| Finventory | 生鲜库存 |
| Fphoto | 生鲜图片 |
| Fdescription | 详情介绍 |

6.6.3顺序图设计

商家管理员上架生鲜活动顺序图如图6-23所示：



图6-23生鲜管理顺序图

6.6.4核心处理流程设计

生鲜管理活动中上架生鲜活动流程图描述如图6-24所示：



图6-24生鲜管理活动图

6.7用户管理

6.7.1功能结构设计

用户管理功能结构图如图6-25所示：



图6-25生鲜管理功能结构图

用户管理功能，主要是商家登录后进入用户管理可查看用户详细信息、删除某个用户或对用户信息进行修改。商家还可通过搜索用户名查找用户。

6.7.2 类图设计

用户管理类图如图6-26所示：

图6-26用户管理类图



用户管理界面类描述如表6-38所示：

表6-38用户管理界面类描述

|  |  |  |
| --- | --- | --- |
| 用户管理界面类 | | |
| 类的描述：负责用户管理预览界面的前台展示 | | |
| 操作描述 | UserDisplay() | 展示获取的生鲜列表 |
| UserDetailDisplay () | 展示单一生鲜详细信息 |

用户管理控制器类描述如表6-39所示：

表6-39用户管理控制器类描述

|  |  |  |
| --- | --- | --- |
| 用户管理控制器类 | | |
| 类的描述：根据前台的业务逻辑不同跳转到对应不同方法，通过控制器类再跳转到前台展示 | | |
| 操作描述 | GetAllUser() | 获取所有用户信息 |
| Modify\_User() | 修改用户详细信息 |
| Search\_Goods() | 检索用户 |

用户管理业务层接口类描述如表6-40所示：

表6-40用户管理业务层接口类描述

|  |  |  |
| --- | --- | --- |
| 用户管理业务层接口类 | | |
| 类的描述：为业务层类提供接口方法 | | |
| 操作描述 | GetUserBYid() | 通过id获取用户实体对象 |
| ModifyUserDetail() | 修改用户详细信息 |
| SearchUserByKeyword() | 通过关键字检索用户 |

用户管理业务层实现类描述如表6-41所示：

表6-41用户管理业务层实现类描述

|  |  |  |
| --- | --- | --- |
| 用户管理业务层实现类 | | |
| 类的描述：实现为业务层类提供接口方法 | | |
| 操作描述 | GetUserBYid() | 通过id获取用户实体对象的实现方法 |
| ModifyUserDetail() | 修改用户详细信息的实现方法 |
| SearchUserByKeyword() | 通过关键字检索用户的实现方法 |

用户管理数据层类描述如表6-42所示：

表6-42用户管理数据层类描述

|  |  |  |
| --- | --- | --- |
| 用户管理数据层类 | | |
| 类的描述：为业务层方法的实现提供数据层的支持 | | |
| 操作描述 | GetUserBYid() | 通过id获取用户实体对象 |
| ModifyUserDetail() | 修改用户详细信息 |
| SearchUserByKeyword() | 通过关键字检索用户 |

用户实体类描述如表6-43所示：

表6-43用户实体类描述

|  |  |  |
| --- | --- | --- |
| 用户实体类 | | |
| 类的描述：用户实体类 | | |
| 属性描述 | Hid | 用户id |
| Hname | 用户昵称 |
| Hphoto | 用户头像 |
| Hsex | 用户性别 |
| Hstate | 用户状态 |

6.7.3顺序图设计

用户搜索生鲜活动顺序图如图6-27所示：



图6-27生鲜选购顺序图

6.7.4核心处理流程设计

用户管理中搜索用户活动流程图描述如图6-28所示：



图6-28用户管理活动图

6.8财务管理

6.8.1功能结构设计

财务管理功能结构用例图描述如图6-29所示：



图6-29财务管理功能结构图

财务管理功能，主要是商家登录后可以进入财务管理查看订单信息与财务流水，在列表可查看不同状态订单，对已支付订单进行发货。

6.8.2 类图设计

财务管理类图用例图描述如图6-30所示：



图6-30财务管理类图

财务管理界面类描述如表6-44所示：

表6-44财务管理界面类描述

|  |  |  |
| --- | --- | --- |
| 财务管理界面类 | | |
| 类的描述：负责财务管理预览界面的前台展示 | | |
| 操作描述 | ALLOrderDisplay() | 展示获取的所有用户订单列表 |
| OrderDetailDisplay () | 展示单一订单详细信息 |

财务管理控制器类描述如表6-45所示：

表6-45财务管理控制器类描述

|  |  |  |
| --- | --- | --- |
| 财务管理控制器类 | | |
| 类的描述：根据前台的业务逻辑不同跳转到对应不同方法，通过控制器类再跳转到前台展示 | | |
| 操作描述 | GetAllOrder() | 获取所有订单信息 |
| DeliverGoods() | 商家发货 |
| ViewOrder() | 查看订单信息 |

财务管理业务层接口类描述如表6-46所示：

表6-46财务管理业务层接口类描述

|  |  |  |
| --- | --- | --- |
| 财务管理业务层接口类 | | |
| 类的描述：为业务层类提供接口方法 | | |
| 操作描述 | GetOrderBYid() | 通过id获取订单实体对象 |
| ModifyToDeliver() | 修改订单状态至发货 |
| ViewOrderByStatus() | 按状态查看订单信息 |

财务管理业务层实现类描述如表6-47所示：

表6-47财务管理业务层实现类描述

|  |  |  |
| --- | --- | --- |
| 统计管理业务层实现类 | | |
| 类的描述：实现为业务层类提供接口方法 | | |
| 操作描述 | GetOrderBYid() | 通过id获取订单实体对象的实现方法 |
| ModifyToDeliver() | 修改订单状态至发货的实现方法 |
| ViewOrderByStatus() | 按状态查看订单信息的实现方法 |

财务管理数据层类描述如表6-48所示：

表6-48财务管理数据层类描述

|  |  |  |
| --- | --- | --- |
| 统计管理数据层类 | | |
| 类的描述：为业务层方法的实现提供数据层的支持 | | |
| 操作描述 | GetOrderBYid() | 通过id获取订单实体对象 |
| ModifyToDeliver() | 修改订单状态至发货 |
| ViewOrderByStatus() | 按状态查看订单信息 |

收货地址实体类描述如表6-49所示：

表6-49收货地址实体类描述

|  |  |  |
| --- | --- | --- |
| 收货地址实体类 | | |
| 类的描述：收货地址实体类 | | |
| 属性描述 | Aid | 地址id |
| Aname | 联系人 |
| Atelephone | 手机号 |
| Aregion | 地区 |
| AddressDetail | 详细收货地址 |

订单实体类描述如表6-50所示：

表6-50订单实体类描述

|  |  |  |
| --- | --- | --- |
| 订单实体类 | | |
| 类的描述：订单实体类 | | |
| 属性描述 | Did | 订单id |
| Dprice | 订单价格 |
| Dftime | 订单创建时间 |
| Dstime | 订单支付时间 |
| Fname | 订单生鲜名称 |
| Fnumber | 订单生鲜数量 |
| Hname | 会员名 |
| Address | 收货地址 |

6.8.3顺序图设计

商家管理员对订单进行发货处理活动顺序图如图6-31所示：



图6-31财务管理顺序图

6.8.4核心处理流程设计

财务管理中商家发货活动流程图如图6-32所示：



图6-32财务管理活动图

6.9统计管理

6.9.1功能结构设计

统计管理功能结构图如图6-33所示：



图6-33统计管理功能结构图

统计管理功能，主要是商家登录后可以进入统计管理查看每天的营收金额，并可按日期段搜索该时间段内营收情况

6.9.2 类图设计

统计管理类图如图6-34所示：



图6-34统计管理类图

统计管理界面类描述如表6-51所示：

表6-51统计管理界面类描述

|  |  |  |
| --- | --- | --- |
| 统计管理界面类 | | |
| 类的描述：负责统计管理预览界面的前台展示 | | |
| 操作描述 | StatisticsDisplay() | 展示获取的所有统计信息列表 |
| StatisDetailDisplay() | 展示单一统计信息详情 |

统计管理控制器类描述如表6-52所示：

表6-52统计管理控制器类描述

|  |  |  |
| --- | --- | --- |
| 统计管理控制器类 | | |
| 类的描述：根据前台的业务逻辑不同跳转到对应不同方法，通过控制器类再跳转到前台展示 | | |
| 操作描述 | GetAllStatistics() | 获取所有统计数据信息 |
| Search\_Statistics() | 查找某一时间段统计数据 |

统计管理业务层接口类描述如表6-53所示：

表6-53统计管理业务层接口类描述

|  |  |  |
| --- | --- | --- |
| 统计管理业务层接口类 | | |
| 类的描述：为业务层类提供接口方法 | | |
| 操作描述 | GetStatisticsBYid() | 通过id获取统计数据实体对象 |
| GetStatisByCategory() | 按分类获取统计数据信息 |
| SearchStatisByDate() | 按日期查找统计数据信息 |

统计管理业务层实现类描述如表6-54所示：

表6-54统计管理业务层实现类描述

|  |  |  |
| --- | --- | --- |
| 统计管理业务层实现类 | | |
| 类的描述：实现为业务层类提供接口方法 | | |
| 操作描述 | GetStatisticsBYid() | 通过id获取统计数据实体对象的实现方法 |
| GetStatisByCategory() | 按分类获取统计数据信息的实现方法 |
| SearchStatisByDate() | 按日期查找统计数据信息的实现方法 |

统计管理数据层类描述如表6-55所示：

表6-55统计管理数据层类描述

|  |
| --- |
| 统计管理数据层类 |

续表6-55

|  |  |  |
| --- | --- | --- |
| 类的描述：为业务层方法的实现提供数据层的支持 | | |
| 操 | GetStatisticsBYid() | 通过id获取统计数据实体对象 |
| 作描述 | GetStatisByCategory() | 按分类获取统计数据信息 |
| SearchStatisByDate() | 按日期查找统计数据信息 |

统计数据实体类描述如表6-56所示：

表6-56统计数据实体类描述

|  |  |  |
| --- | --- | --- |
| 统计数据实体类 | | |
| 类的描述：收货统计数据实体类 | | |
| 属性描述 | Tid | 统计信息id |
| Tdate | 日期 |
| Tmoney | 统计金额 |
| Fnumber | 统计生鲜数量 |

6.9.3顺序图设计

商家管理员查看财务流水活动顺序图如图6-35所示：



图6-35财务管理顺序图

6.9.4核心处理流程设计

统计管理中查看财务流水用例图描述如图6-36所示：



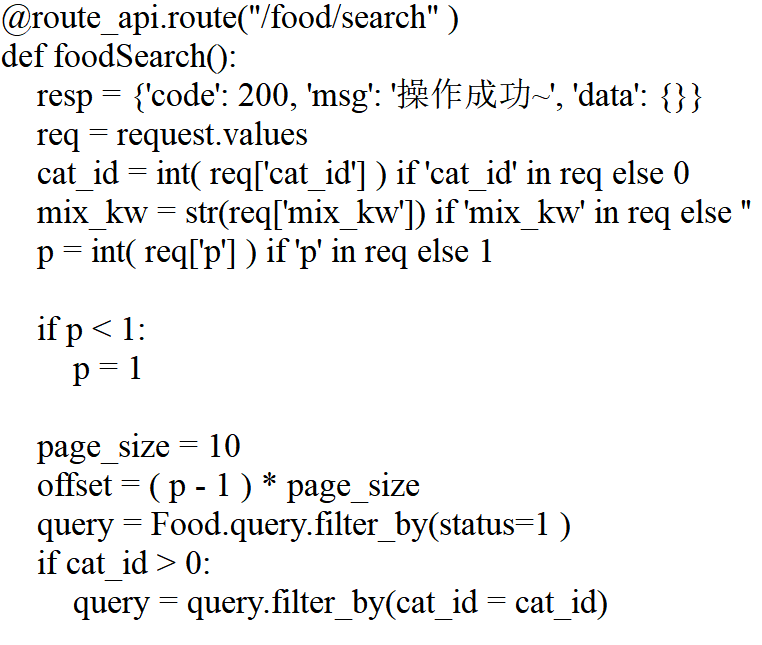
图6-36统计管理活动图

7 编码

## 7.1 代码实现与核心算法

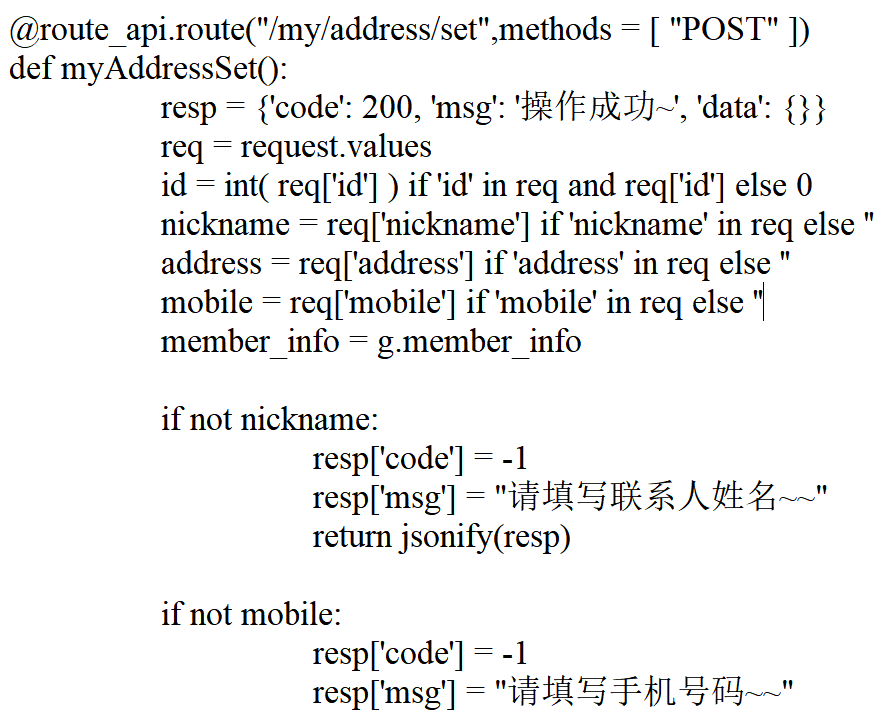
7.1.1用户搜索生鲜业务

搜索生鲜功能是生鲜选购模块的功能，通过关键字对生鲜名称进行查询，实现查询到的生鲜在搜索页面进行展示，用户搜索生鲜业务代码如下：



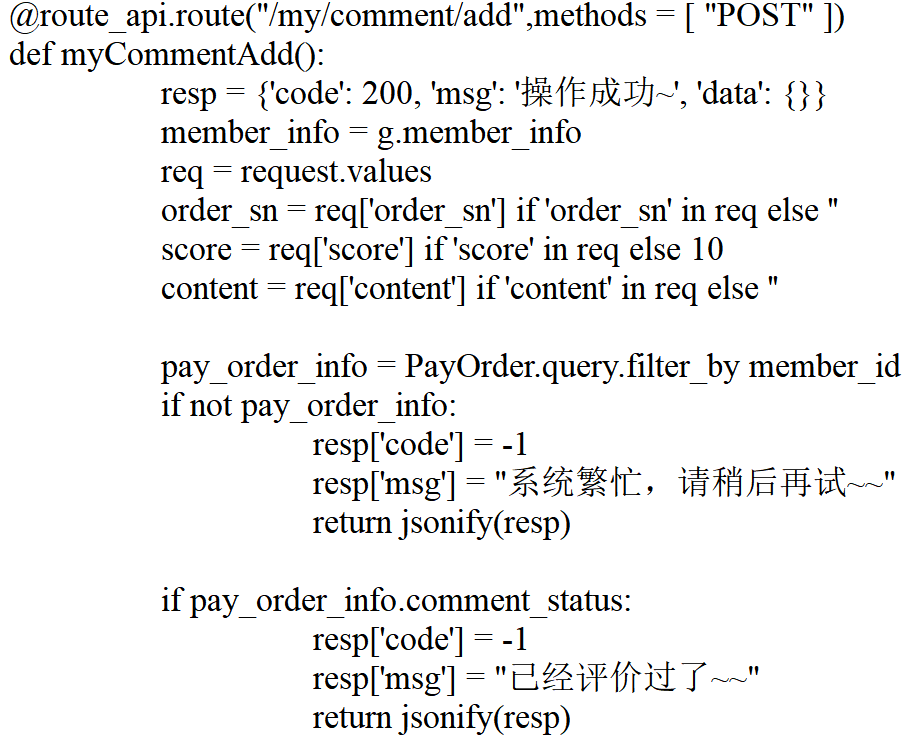
7.1.2添加收货地址业务

添加收货地址是收货地址管理中的功能，通过代码逻辑实现用户必须输入收货地址中的所有信息方可成功创建，当某个信息缺失时，提示用户填完善信息。添加收货地址业务代码实现如下：



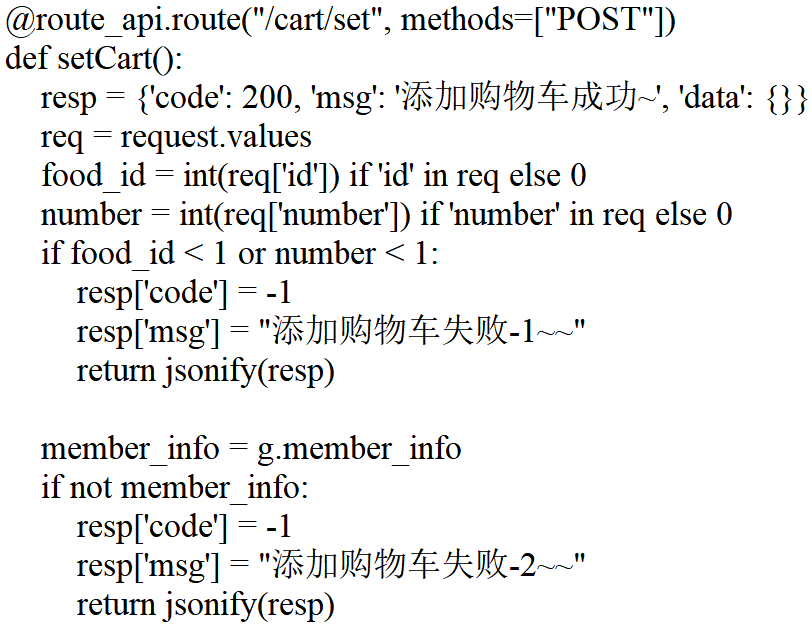
7.1.3用户评论订单业务

用户评论订单是订单管理中的功能，实现用户确认收货后对订单进行评价。通过代码逻辑实现当用户已经评价时提示订单已被评价，当系统未找到订单信息时，提示用户重新尝试评价。用户评论订单业务代码实现如下：



7.1.4生鲜添加购物车业务

生鲜添加购物车是购物车管理中的功能，实现用户把生鲜加入购物车的操作，通过代码逻辑实现当生鲜库存不足时提示用户当前生鲜库存不足，同时也对于系统本身可能出现的加入购物车失败问题做了分类报错处理，如数据库已不存在当前生鲜信息显示报错信息-3。生鲜添加购物车业务代码实现如下：



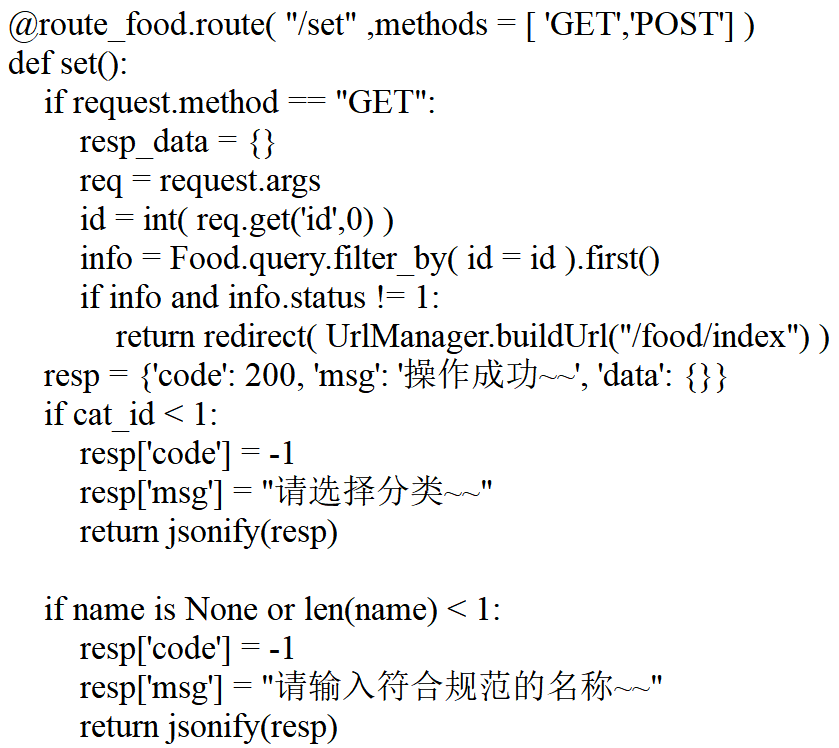
7.1.5修改后台账号密码业务

修改后台账号密码是后台管理端子系统中商家账号管理的功能，实现商家管理员业务对后台账号密码进行修改。修改后台账号密码代码实现如下：



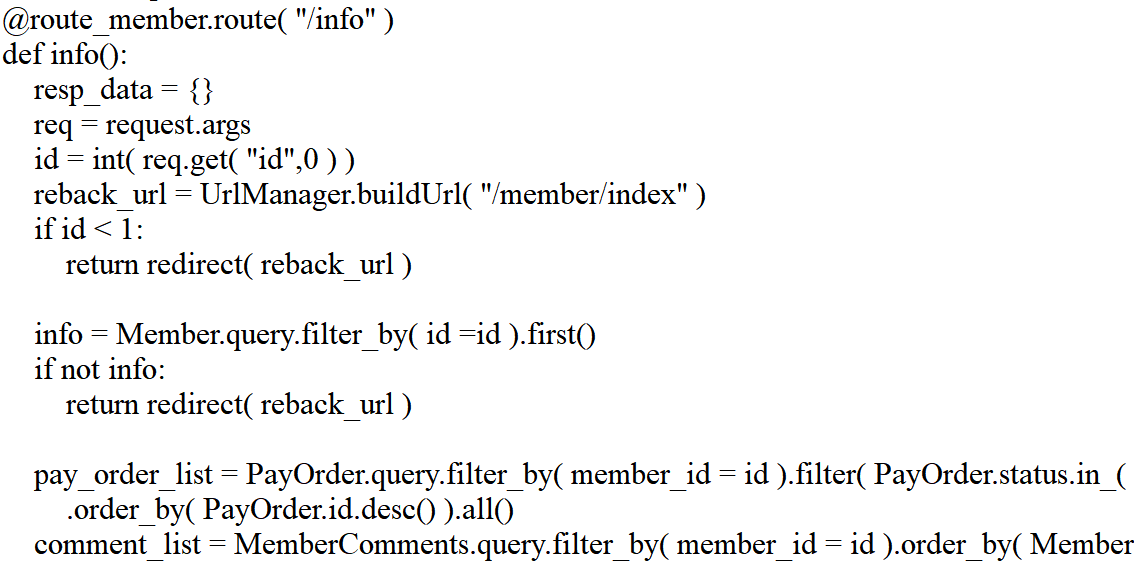
7.1.6商家上架生鲜业务

商家上架生鲜是后台管理端子系统中生鲜管理的功能，实现商家在后台添加生鲜，前台展示添加后的生鲜，通过代码逻辑实现当商家添加生鲜时未选择生鲜分类时，提示商家对要添加的生鲜做好分类。商家添加生鲜业务代码实现如下：



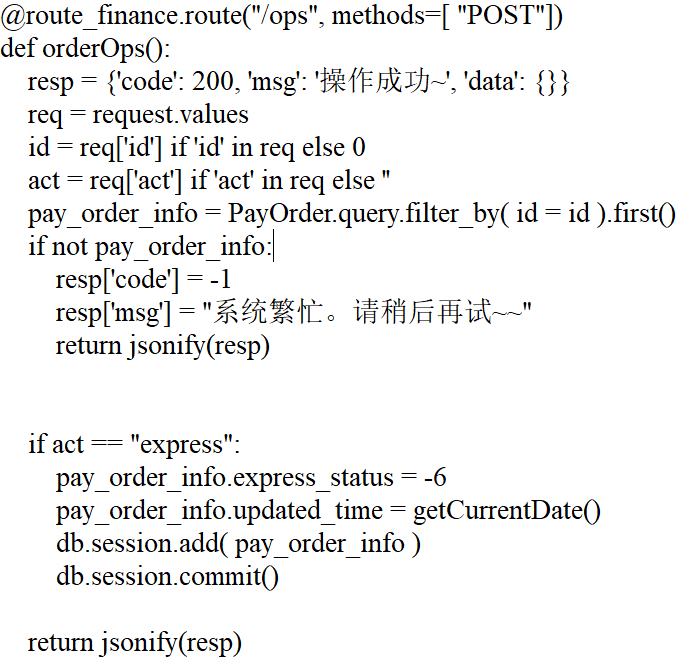
7.1.7查看用户信息业务

查看用户信息是后台管理端子系统中用户管理的功能，实现查看用户信息功能，不仅展示基本账号信息，还通过用户id匹配全部订单列表中该用户的订单信息以及评论信息，进行展示。查看用户信息业务代码实现如下：



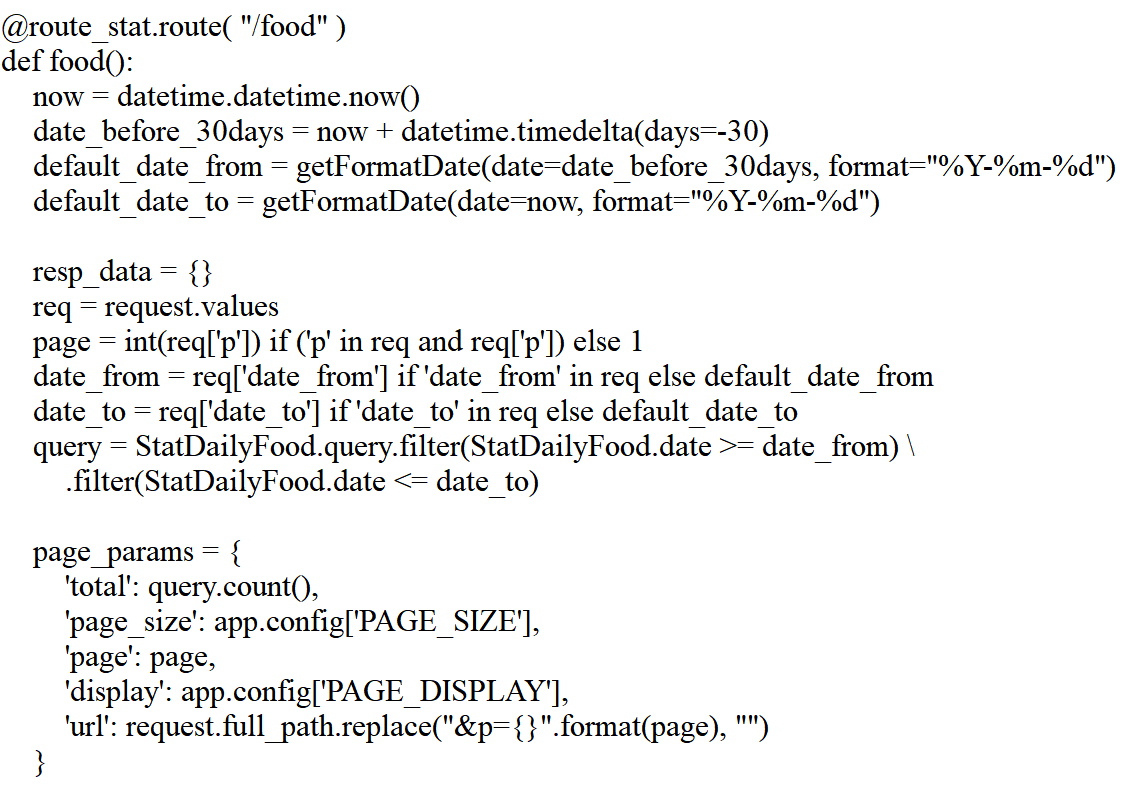
7.1.8订单发货业务

订单发货后是台管理端子系统中财务管理的功能，实现商家对已支付订单进行发货，当商家选择发货后，修改数据库中的订单状态，同时创建一个发货时间信息。订单发货业务代码实现如下：



7.1.9展示生鲜销售总额业务

展示生鲜销售总额是后台管理端子系统中统计管理模块的业务功能，实现对某时间段内生鲜销售总额进行统计，日期的确定使用了python中format函数，确定格式为年-月-日。展示生鲜销售总额业务代码实现如下：



## 7.2 代码优化分析

在系统的开发中，有许多代码可以复用，比如本系统中后台端中侧边是各个功能模块的选择界面栏，选择不同的模块右边将出现对应的不同的界面，而在这之中一直未改变的是侧边界面栏。因此在编写不同模块的界面代码时，无需重复编写侧边栏，这里可使用flask中模板继承，实现表现层与业务逻辑的分离，复用代码，无需在多个 HTML 模板中维护数个一样的导航栏。

8 测试

## 8.1 测试方案设计

### 8.1.1 测试策略

通常来说，小程序从产品研发到正式上线发布，以及后期的运行维护过程中，可能会会遇到一些问题，因此会导致系统有一定的开发失败的概率。生鲜外卖小程序也是这样的，小程序如果出现问题，比如说生鲜加入购物车出现问题，系统页面不跳转或者响应时间太慢，将会使用户的使用体验严重下降，如果不在程序上线前提前解决，很可能会导致大量用户的流失，同时也会导致后期的后期研发与维护需要大量的人力物力，造成极大的损失。产品测试是为了验证系统的可靠性从而执行程序的过程，因此要验证小程序的运行情况是否与预期结果一致。功能测试用例使用的是黑盒测试方法，依据用户在小程序购买生鲜时的实际操作流程，对小程序页面和功能，设计测试用例进行测试。

### 8.1.2 测试进度安排

对本生鲜外卖小程序的测试按照以下方案进行：

1. 小程序开发完成后，通过使用微信开发者工具对生鲜外卖小程序系统进行功能上的测试。在小程序分别对每个功能点展开测试，通过查看界面上的变化和表中数据的变化进行测评。
2. 对小程序使用中各界面展示是否正确、合理以及美观进行测试，并对小程序的各功能响应情况进行性能测试。

### 8.1.3 测试资源

（1）测试环境

Windows 10操作系统，CPU为INTEL CORE i5处理器， PyCharm 2018.1.4 x64。

（2）测试管理工具

微信开发者工具。

### 8.1.4 关键测试点

（1）用户登陆小程序 ，当用户进入小程序时是否能通过微信号进行授权登录。

（2）在生鲜导航栏中选择生鲜分类后，是否会推荐出对应分类的生鲜。

（3）当用户在生鲜搜索框输入生鲜名称后，是否会展示对应关键字的所有生鲜。

（4）当用户进入生鲜详情界面时，是否会展示出生鲜所对应的所有生鲜详情。

（5）当用户在生鲜详情界面点击添加购物车时，是否会将当前生鲜正确加入购物车当中，并且当用户进入购物车界面是可以正确查看到该生鲜。

（6）当用户在购物车页面修改购物车内生鲜信息时，点击删除生鲜或增减生鲜数量，是否会正确执行相应操作。

（7）当用户进入订单界面时，页面是否会显示出不同状态下的订单信息，点击不同状态订单分类，是否会展现对应分类的订单，如待支付、待发货、待收货、待评价等。

（8）当用户进入收货地址的详细界面后，选择添加收货地址，是否会跳转至收货地址添加界面，用户可输入收货地址信息，并选择保存进行保存。

（9）当用户重新进入收货地址的详细界面时，页面是否显示用户已添加的收货地址信息，用户选择编辑是否可以对收货地址信息进行修改。

（10）当用户选择评价后是否跳转至评论界面，输入想要发表的评论并点击确认后，评论列表中是否能刷新出出刚刚评论的信息。

## 8.2 测试用例构建

### 8.2.1 测试用例编写约定

测试项：对测试用例的描述。

操作：对进行该用例测试的操作的描述。

预期结果：完成以上操作后系统应当得到的结果。

### 8.2.2 测试用例设计

本系统测试用例设计大致如表8-1所示：

表 8-1测试用例设计

|  |  |  |  |
| --- | --- | --- | --- |
| 编号 | 测试项 | 操作 | 期望结果 |
| 01 | 生鲜分类 | 用户点击首页中的不同分类类目。 | 在新的页面上出现当前分类对应的生鲜 |
| 02 | 购物车 | 1）用户进入生鲜详情页后选择加入购物车并选择生鲜数量实现加购。  2）用户在购物车选择“+”、“-”实现对生鲜数量的修改。 | 当生鲜库存量不足，系统提示库存不足无法加购；当生鲜库存充足时，生鲜加入到购物车，且用户更新生鲜数量。 |
| 03 | 下单支付 | 1）用户在购物车页面选择“下单”。  2）用户在调转后出现的订单详情页选择“去付款” | 点击下单后会跳转到生成订单详情页面，用户付款成功后，可查看刚刚支付的订单详情。 |
| 04 | 评价生鲜 | 1. 选择“评价” 2. 输入评论内容，点击确认 | 跳转到生鲜评价页面评价完成后在评论列表出现用户评价的信息。 |
| 05 | 收货地址管理 | 1）在个人中心页面选择“我的收货地址”。  2）选择新增收货地址。 | 地址编辑完成后保存，在收货地址页面显示出的当前保存的收货地址信息。 |

### 8.2.3 关键测试用例

（1）生鲜分类选择测试

进入生鲜首页界面时，在生鲜推荐栏目上有分类生鲜选择的导航栏，用户可以根据购物意向选择想要的生鲜，从而展示出对应的生鲜分类推荐，随后出现对应生鲜的列表，但此时可能出现推荐的生鲜与后台不统一的情况，生鲜的分类选择测试如表8-2所示：

表8-2 分类选择测试

|  |  |
| --- | --- |
| 测试用例ID | 1 |
| 描述 | 生鲜分类测试 |
| 测试大纲 | 1.用户选择时令水果  预期结果：显示当前季节下的所有水果类生鲜信息，包括图片、名称和价格，点击，某一生鲜界面后进入生鲜的详细界面 |
| 2.用户选择牛羊猪肉  预期结果：显示所有牛羊猪肉类生鲜信息，包括图片、名称和价格，点击，某一生鲜界面后进入生鲜的详细界面 |
|  | 3.用户选择新鲜蔬菜  预期结果：显示所有新鲜蔬菜类生鲜信息，包括图片、名称和价格，点击，某一生鲜界面后进入生鲜的详细界面 |
| 4．用户选择海鲜水产  预期结果：显示所有海鲜水产类生鲜信息，包括图片、名称和价格，点击，某一生鲜界面后进入生鲜的详细界面 |
| 测试结果 | 正确 |

（2）界面测试

生鲜外卖小程序的整体布局与样式对用户的体验至关重要，因此进入小程序后，需要对购物过程中的界面进行实时考察，如图片能否清晰展示，小程序的图文排版是否美观等等，界面测试结果如表8-3所示：

表8-3 界面测试结果

|  |  |
| --- | --- |
| Windows 10下微信开发者工具 | 是否通过测试（是√否×） |
| 小程序界面的布局是否合理？ | √ |
| 查询的生鲜排版是否好看？ | √ |
| 查询生鲜时小程序的反应时间是否小于1秒？ | √ |

续表8-3

|  |  |
| --- | --- |
| 评价完生鲜后评论是否显示在生鲜详情页面？ | √ |
| 生鲜展示的图片是否清晰？ | √ |

（3）数据显示测试

数据显示测试的主要目的是验证在使用生鲜外卖小程序的过程中，用户的个人中心页面和与各类数据相关的功能是否实时、准确地显示，其中数据显示测试如表8-4所示：

表8-4 数据显示测试

|  |  |
| --- | --- |
| Windows 10下微信开发者工具 | 是否通过测试（是√否×） |
| 用户的加入购物车生鲜各项列表显示 | √ |
| 生鲜评论区评论的时间显示 | √ |
| 用户收货地址的各项收货信息显示 | √ |
| 点击修改购物车生鲜数量后数量显示改变 | √ |
| 点击支付后订单状态改变 | √ |

### 8.2.4 测试用例维护

为了满足用户对于更多生鲜更大范围搜索的要求和升级小程序版本的需要，将要添加、删除和修改部分生鲜外卖小程序功能，因此需要修改、增加部分测试用例来适应生鲜外卖小程序功能，更改生鲜外卖小程序测试用例的主要工作有下面几点：

（1）及时删除重复的测试用例

因为测试用例是由不同的测试人员编写，所以可能多个测试用例是被用来测试相同功能的，这样就会导致测试效率的大大降低，因此需要对重复的用例进行删除处理，并定期对用例集进行整理。

（2）增加新的测试用例

从使用本生鲜外卖小程序的用户那里得到反馈，收集并统计本小程序还有哪些不足的地方，还有需要改进哪些功能，有哪些功能需要增加，这就需要我们增加新的测试用例，

（3）及时删除需淘汰的测试用例

随着小程序功能的更新，部分生鲜外卖小程序的功能因为不再适应新版本功能需要删除，那么该功能的测试用例也失去了价值，因此就需要对这些无用的测试用例删除处理。

（4）改进难控制的测试用例

由于生鲜外卖小程序的功能不断更新，测试用例也越来越多，其中就会有部分的测试用例对输入条件非常敏感，所以很难很好地控制这些用例的结果，因此就需要改进这些用例，让它们达到可控制。

9 总结与展望

## 9.1 设计工作总结

本文通过分析当下传统社区生鲜商家在互联网新零售下面临的机遇和挑战，并且考虑到2020年初突发的新冠肺炎疫情期间居民“云买菜”的迫切需求，完成了生鲜外卖微信小程序这一系统。在本系统的设计与实现过程中，前端使用小程序技术，后端管理采用的是flask框架，相互结合搭建起这一系统。开发时首先通过分析系统受众的需求、设计与搭建完整的架构，再到系统的编码实现，最后对系统的功能和性能进行测试。一步步走来，比预期的难度要更大，因为在之前的学习过程中没有独立开发一个完整系统的经历，在这个过程中熟悉了开发一个完整系统的过程，同时也更加深入了解了微信小程序和flask框架等的知识以及如何运用。通过一步步的开发，进一步提高了购物系统开发能力和理论知识的应用能力，更加进一步地明白了在小程序和flask中应用MVC 设计模式，编码中使用template 模板的好处以及如何部署构建和使用 MySQL 数据库，从整体角度把握每一个系统和功能模块在项目中发挥的作用。

## 9.2 未来工作展望

系统也有不足和需要继续改善的地方，需要在接下来的学习中进一步的完善：

（1）对于小程序的本体功能还可以进行进一步的扩展，比如增加物流跟踪功能等等。

（2）小程序还可以进一步开发小程序与微信联通互动的功能，如下单成功后微信弹窗进行消息通知等。

（3）小程序在订单支付方面还可以增加多种支付方式，如银行卡、信用卡等。

谢 辞

参考文献

[1] 陈一欣.基于Flask 技术的分布式Android 产品验证系统[D].成都: 电子科技大学, 2019.

[2] 任丹.基于Flask和Vue的AIS数据分析系统设计与开发[J].软件, 2019, 40(10):111-114

[3] 方蓓工作室. 微信公众平台开发最佳实践[M]. 北京：机械工业出版社, 2015.5

[4] 李炜康. 基于微信的网上购物系统的设计与实现[D]. 南京大学, 2015

[5] 张翔. 微信小程序：分享微信创业 2.0 时代千亿红利[M]. 北京：清华大学出版社, 2017

[6] Luke Welling,Laura Thomson. PHP 和 MySQLWeb 开发[M]. 北京：机械工业出版社，2018.4

[7] 雷磊. 微信小程序开发入门与实践[M]. 北京: 清华大学出版社, 2017

[8] 储广昕. 分析软件开发中数据库设计理论的实践[J]. 信息通信, 2015, 149(5):127-128

[9] 贾兆颖.  [美国、英国、日本生鲜电商行业发展模式对中国的启示](https://kns8.cnki.net/kcms/detail/detail.aspx?filename=SJNY201608008&dbcode=CJFQ&dbname=CJFD2016&v=)[J]. 世界农业. 2016(08)

[10] 柴慧.  [上海生鲜O2O模式发展的瓶颈与对策](https://kns8.cnki.net/kcms/detail/detail.aspx?filename=SHNC201505008&dbcode=CJFQ&dbname=CJFD2015&v=)[J]. 上海农村经济. 2015(05)

[11] 萨师煊, 王珊等. 数据库系统概论（第三版）[M]. 北京:高等教育出版社, 2005.45-187

[12] 张立科 . Mysql 数据库通用模块及典型系统开发实力导航 [M].人民邮电出版社,2006.10-295

[13] 沈炜, 徐慧,汤倩. Mysql 数据库编程技术与实例[M]. 北京:人民邮电出版社, 205. 114-226

[14] 冯晓强,程晓昕. 基于 MVC 模式的网上购物系统的设计与实现[J]. 现代计算机:专业版. 2018(7):177-180

[15] 孔秀丽. 基于微信公众平台的大学生移动商城的构建[J]. 科技创业月刊，2015(3):25-26

[16] 于秀山.于洪敏,著. 软件测试新技术与实践[M]. 电子工业出版社, 2016

[17] 徐兰芳, 彭冰, 编. 数据库设计与实现[M]. 上海交通大学出版社, 2016

[18] 徐其帅. 面向对象软件开发方法的实例分析[D]. 浙江大学, 2018

[19] 孟祥磊. 微信公众平台开发实例教程[M]. 人民邮电出版社, 2017

[20] 李华明. 基于 PHP 和 MySQL 的网上购物系统设计与实现[D]. 电子科技大学, 2014

[21] 李蓉. 基于 MVC 模式的 WEB 应用研究[J]. 计算机应用与软件，2015(10):12-13

[22] 赵敬, 李贝. 微信公众平台发展现状初探, 新闻实践, 2013(8):8-10

[23] 金莹. 基于微信小程序的个人电台系统的设计与实现[D]. 吉林大学, 2017

[24] 朱勇贞. 基于微信的电商平台的设计与实现[D]. 东华大学, 2017

[25] 王勇锋. 基于微信公众平台的医药零售模式的研究和实现[D]. 华南理工大学, 2017

[26] 刘旺.[MVC架构下的教培信息发布系统的设计与实现[J]](http://lib.cqvip.com/Qikan/Article/Detail?id=7100118713&from=Qikan_Article_Detail).电脑知识与技术:学术版,2019,15(8Z):111-113

[27] 刘振宇.[数字化校园统一信息门户规划与实现[J]](http://lib.cqvip.com/Qikan/Article/Detail?id=7100709845).科教导刊(电子版),2019(35):80-80

[28] Voluntary sensemaking: the identity formation process of volunteers in sport organizations[J]. Christine E. Wegner,Gareth J. Jones,Jeremy S. Jordan. European Sport Management Quarterly. 2019(5)

[29] Volunteer Involvement and Organizational Performance: The Use of Volunteer Officers in Public Safety[J]. Seong C. Kang. Public Performance & Management Review. 2019(3)

[30] Police volunteers in the Netherlands: a study on policy and practice[J]. Ronald van Steden,Shanna M. Mehlbaum. Policing and Society. 2019

# 附录A 外文翻译—原文部分

The Source of Article：《[The New And Improved Flask](https://itbooks.pipipan.com/fs/18113597-319371576)》

**What Are Templates?**

I want the home page of my microblogging application to have a heading that welcomes the user. For the moment, I'm going to ignore the fact that the application does not have the concept of users yet, as this is going to come later. Instead, I'm going to use a mock user, which I'm going to implement as a Python dictionary, as follows:

user = {'username': 'Miguel'}

Creating mock objects is a useful technique that allows you to concentrate on one part of the application without having to worry about other parts of the system that don't exist yet. I want to design the home page of my application, and I don't want the fact that I don't have a user system in place to distract me, so I just make up a user object so that I can keep going.The view function in the application returns a simple string. What I want to do now is expand that returned string into a complete HTML page, maybe something like this:

*app/routes.py*: Return complete HTML page from view function

from app import app

@app.route('/')@app.route('/index')def index():

user = {'username': 'Miguel'}

return '''

<html>

<head>

<title>Home Page - Microblog</title>

</head>

<body>

<h1>Hello, ''' + user['username'] + '''!</h1>

</body>

</html>'''

If you are not familiar with HTML, I recommend that you read [HTML Markup](https://en.wikipedia.org/wiki/HTML#Markup) on Wikipedia for a brief introduction.Update the view function as shown above and give the application a try to see how it looks in your browser.

I hope you agree with me that the solution used above to deliver HTML to the browser is not good. Consider how complex the code in this view function will become when I have the blog posts from users, which are going to constantly change. The application is also going to have more view functions that are going to be associated with other URLs, so imagine if one day I decide to change the layout of this application, and have to update the HTML in every view function. This is clearly not an option that will scale as the application grows.

If you could keep the logic of your application separate from the layout or presentation of your web pages, then things would be much better organized, don't you think? You could even hire a web designer to create a killer web site while you code the application logic in Python.

Templates help achieve this separation between presentation and business logic. In Flask, templates are written as separate files, stored in a templates folder that is inside the application package. So after making sure that you are in the microblog directory, create the directory where templates will be stored:

(venv) $ mkdir app/templates

Below you can see your first template, which is similar in functionality to the HTML page returned by the index() view function above. Write this file in app/templates/index.html:

*app/templates/index.html*: Main page template

<html>

<head>

<title>{{ title }} - Microblog</title>

</head>

<body>

<h1>Hello, {{ user.username }}!</h1>

</body></html>

This is a mostly standard, very simply HTML page. The only interesting thing in this page is that there are a couple of placeholders for the dynamic content, enclosed in {{ ... }} sections. These placeholders represent the parts of the page that are variable and will only be known at runtime.Now that the presentation of the page was offloaded to the HTML template, the view function can be simplified:

*app/routes.py*: Use render\\_template() function

from flask import render\_templatefrom app import app

@app.route('/')@app.route('/index')def index():

user = {'username': 'Miguel'}

return render\_template('index.html', title='Home', user=user)

This looks much better, right? Try this new version of the application to see how the template works. Once you have the page loaded in your browser, you may want to view the source HTML and compare it against the original template.The operation that converts a template into a complete HTML page is called rendering. To render the template I had to import a function that comes with the Flask framework called render\_template(). This function takes a template filename and a variable list of template arguments and returns the same template, but with all the placeholders in it replaced with actual values.The render\_template() function invokes the [Jinja2](http://jinja.pocoo.org) template engine that comes bundled with the Flask framework. Jinja2 substitutes {{ ... }} blocks with the corresponding values, given by the arguments provided in the render\_template() call.

**Conditional Statements**

You have seen how Jinja2 replaces placeholders with actual values during rendering, but this is just one of many powerful operations Jinja2 supports in template files. For example, templates also support control statements, given inside {% ... %} blocks. The next version of the index.html template adds a conditional statement:

*app/templates/index.html*: Conditional statement in template

<html>

<head>

{% if title %}

<title>{{ title }} - Microblog</title>

{% else %}

<title>Welcome to Microblog!</title>

{% endif %}

</head>

<body>

<h1>Hello, {{ user.username }}!</h1>

</body></html>

Now the template is a bit smarter. If the view function forgets to pass a value for the title placeholder variable, then instead of showing an empty title the template will provide a default one. You can try how this conditional works by removing the title argument in the render\_template() call of the view function.

**Loops**

The logged in user will probably want to see recent posts from connected users in the home page, so what I'm going to do now is extend the application to support that.Once again, I'm going to rely on the handy fake object trick to create some users and some posts to show:*app/routes.py*: Fake posts in view function

from flask import render\_templatefrom app import app

@app.route('/')@app.route('/index')def index():

user = {'username': 'Miguel'}

posts = [

{

'author': {'username': 'John'},

'body': 'Beautiful day in Portland!'

},

{

'author': {'username': 'Susan'},

'body': 'The Avengers movie was so cool!'

}

]

return render\_template('index.html', title='Home', user=user, posts=posts)

T o represent user posts I'm using a list, where each element is a dictionary that has author and body fields. When I get to implement users and blog posts for real I'm going to try to preserve these field names as much as possible, so that all the work I'm doing to design and test the home page template using these fake objects will continue to be valid when I introduce real users and posts.

On the template side I have to solve a new problem. The list of posts can have any number of elements, it is up to the view function to decide how many posts are going to be presented in the page. The template cannot make any assumptions about how many posts there are, so it needs to be prepared to render as many posts as the view sends in a generic way.For this type of problem, Jinja2 offers a for control structure:

*app/templates/index.html*: for-loop in template

<html>

<head>

{% if title %}

<title>{{ title }} - Microblog</title>

{% else %}

<title>Welcome to Microblog</title>

{% endif %}

</head>

<body>

<h1>Hi, {{ user.username }}!</h1>

{% for post in posts %}

<div><p>{{ post.author.username }} says: <b>{{ post.body }}</b></p></div>

{% endfor %}

</body></html>

Simple, right? Give this new version of the application a try, and be sure to play with adding more content to the posts list to see how the template adapts and always renders all the posts the view function sends.

**Template Inheritance**

Most web applications these days have a navigation bar at the top of the page with a few frequently used links, such as a link to edit your profile, to login, logout, etc. I can easily add a navigation bar to the index.html template with some more HTML, but as the application grows I will be needing this same navigation bar in other pages. I don't really want to have to maintain several copies of the navigation bar in many HTML templates, it is a good practice to not repeat yourself if that is possible.

Jinja2 has a template inheritance feature that specifically addresses this problem. In essence, what you can do is move the parts of the page layout that are common to all templates to a base template, from which all other templates are derived.

So what I'm going to do now is define a base template called base.html that includes a simple navigation bar and also the title logic I implemented earlier. You need to write the following template in file app/templates/base.html:

*app/templates/base.html*: Base template with navigation bar

<html>

<head>

{% if title %}

<title>{{ title }} - Microblog</title>

{% else %}

<title>Welcome to Microblog</title>

{% endif %}

</head>

<body>

<div>Microblog: <a href="/index">Home</a></div>

<hr>

{% block content %}{% endblock %}

</body></html>

In this template I used the block control statement to define the place where the derived templates can insert themselves. Blocks are given a unique name, which derived templates can reference when they provide their content.

With the base template in place, I can now simplify index.html by making it inherit from base.html:

*app/templates/index.html*: Inherit from base template

{% extends "base.html" %}

{% block content %}

<h1>Hi, {{ user.username }}!</h1>

{% for post in posts %}

<div><p>{{ post.author.username }} says: <b>{{ post.body }}</b></p></div>

{% endfor %}{% endblock %}

Since the base.html template will now take care of the general page structure, I have removed all those elements from index.html and left only the content part. The extends statement establishes the inheritance link between the two templates, so that Jinja2 knows that when it is asked to render index.html it needs to embed it inside base.html. The two templates have matching block statements with name content, and this is how Jinja2 knows how to combine the two templates into one. Now if I need to create additional pages for the application, I can create them as derived templates from the same base.html template, and that is how I can have all the pages of the application sharing the same look and feel without duplication.

附录B 外文翻译—译文部分

文章来源：《[The New And Improved Flask](https://itbooks.pipipan.com/fs/18113597-319371576)》

**什么是模板?**

我想要我的 microblog 应用的首页头部有一个欢迎用户的显示。当然现在应用中还没有用户的概念，在后面会加上。取而代之的是，我将会使用一个 mock 用户(模拟用户)，我用一个 Python 字典来实现：user = {'username': 'Miguel'}

创建一些模拟对象是一项比较有用的技术，这样使得你能专心于应用的一部分，而不用担心系统的其他部分还不存在。我想为我的应用设计一个首页，但是我不想被系统目前没有用户系统而烦恼，因此我模拟了一个用户对象，这样我就可以继续我的工作了。

视图函数返回了一个简单的字符串。我要做的就是将这个返回的字符串展开成一个完整的 HTML 页面。比如这样：

from app import app

@app.route('/')@app.route('/index')def index():

user = {'username': 'miguel'}

return '''<html> <head> <title>Home Page - Microblog</title> </head> <body> <h1>Hello, ''' + user['username'] + '''!</h1> </body></html>'''

更新视图函数之后，重新启动应用，然后在浏览器中打开这个页面，我希望你能同意我上述将 HTML 传递到浏览器的方式不是很好这个观点。当我想将用户的文章返回的时候，视图函数将会变得多么复杂，而且文章还会经常改变。而且应用还会有与其他 URL 绑定的视图函数，那么想象以下将来有一天我要改变应用的布局，那么我就得在每个视图函数中更新 HTML。所以这绝不是一个应对应用规模不断增长的方案。如果你能将应用的逻辑和 web 页面的展示和布局分开的话，所有的东西都变得易于组织，不是吗？你甚至可以雇佣一个 web 页面设计者来创造非常牛逼的页面，而你只需要使用 Python 完成应用的逻辑代码。模板帮助实现了表示层和业务逻辑的分离。在 Flask 中，模板被写在单独的文件中，存储在应用的包的 templates 文件夹中。因此确保你在 microblog 文件夹下，然后创建 templates 文件夹: (venv) $ mkdir app/templates

下面你可以看到你的第一个模板，和上面 index() 视图函数返回的 HTML 页面很相似，将这个文件保存在 app/templates/index.html

<html>

<head>

<title>{{ title }} - Microblog</title>

</head>

<body>

<h1>Hello, {{ user.username }}!</h1>

</body></html>

这一个标准的，非常简单的 HTML 页面。里面值得注意的是，两个花括号括起来的用于动态内容的部分。这表示里面的内容是变量，而且之后再运行的时候才能生成。现在页面的展示已经放到了 HTML 模板中，视图函数就被简化了：

from flask import render\_templatefrom app import app

@app.route('/')@app.route('/index')def index():

user = {'username': 'Miguel'}

return render\_template('index.html', title='Home', user=user)

是不是看起来更好了。在应用中试试看看模板是怎么工作的。一旦你在浏览器中载入了这个页面，你可能想看看 HTML 源代码和以前有什么不一样。将模板转换成完整的 HTML 页面的过程叫做渲染(rendering)。为了渲染模板，我必须从 Flask 中导入一个函数叫做 render\_template，这个函数将模板名字和一个模板参数的变量列表作为参数，然后返回同样的模板，但是已经用变量将模板中的占位符给替换了。render\_template() 函数调用了 [Jinja2](http://jinja.pocoo.org/) 模板引擎，该引擎是和 Flask 绑定在一起的。Jinja2 将会使用通过 render\_template 函数传递进来的参数来替代对应的 {{...}} 块。

**条件语句**

你已经看到了 Jinja2 在渲染的时候是如何将真实的值替换占位符的，但是这只是 Jinja2 众多强大功能之一。比如，模板同样支持以 {%...%} 块的方式的控制语句。index.html 的下一个版本就是增加一个条件语句。

<html>

<head>

{% if title %}

<title>{{ title }} - Microblog</title>

{% else %}

<title>Welcome to Microblog!</title>

{% endif %}

</head>

<body>

<h1>Hello, {{ user.username }}!</h1>

</body></html>

现在模板变得更智能了。如果视图函数忘记传递 title 参数，模板会使用默认值来渲染而不是一个空的 title。你可以通过移除 title 参数来查看条件语句是如何工作的。

**循环**

登录的用户可能想看在首页的所有用户最近的文章列表。那么应该如何扩展应用来支持这个功能呢。

同样，我会创建一些模拟的用户对象和一些文章对象来展示。

from flask import render\_templatefrom app import app

@app.route('/')@app.route('/index')def index():

user = {'username': 'Miguel'}

posts = [

{

'author': {'username': 'John'},

'body': 'Beautiful day in Portland!'

},

{

'author': {'username': 'Susan'},

'body': 'The Avengers movie was so cool!'

}

]

return render\_template('index.html', title='Home', user=user, posts=posts)

为了展示用户文章，我使用了个列表，里面每个元素都是一个字典，并且都包含 author 和 body 字段。当我要实现用户和文章的时候我也会尽量保留现在的字段名字，所以在之后我在设计和测试之中使用的这些对象依然有效。在模板方面我必须得解决一个新的问题。posts 列表可能有任意多个元素，它取决于视图函数决定多少个 posts 将会在页面展示。模板不能对有多少 posts 做任何假定，因此它必须以通用的方式来渲染任意视图函数传递给它的 posts。

对于这个问题，Jinja2 提供了 for 控制结构：

<html>

<head>

{% if title %}

<title>{{ title }} - Microblog</title>

{% else %}

<title>Welcome to Microblog</title>

{% endif %}

</head>

<body>

<h1>Hi, {{ user.username }}!</h1>

{% for post in posts %}

<div><p>{{ post.author.username }} says: <b>{{ post.body }}</b></p></div>

{% endfor %}

</body></html>

是不是很简单？

**模板继承**

大多数 web 应用在页面顶部都会有一个导航栏，上面放置一些比较常用的链接，比如编辑个人资料，登录登出等。我可以很容易的给 index.html 模板来添加一个导航栏，但是随着应用的规模增长我需要将同样的导航栏放到其他页面上。但是我不想在多个 HTML 模板中维护数个一样的导航栏。not repeat yourself 不要重复你自己！Jinja2 拥有模板继承特性就是来解决这个问题的。在本质上，你要做的就是将所有模板共同的东西拿到一个基础模板中，然后其他模板从这个基础模板派生。所以我要做的就是定义一个叫 base.html 的基础模板，其包含了一个简单的导航栏以及之前实现的简单逻辑。你需要将下面的代码保存到app/template/base.html 模板中

<html>

<head>

{% if title %}

<title>{{ title }} - Microblog</title>

{% else %}

<title>Welcome to Microblog</title>

{% endif %}

</head>

<body>

<div>Microblog: <a href="/index">Home</a></div>

<hr>

{% block content %}{% endblock %}

</body></html>

在这个模板中我使用了 block 控制语句来定义派生模板可以插入的位置。block 的名字是唯一的，这样派生模板就可以在提供他们的内容时候引用。

在有了基础模板，我现在可以让 index.html 来继承 base.html

{% extends "base.html" %}

{% block content %}

<h1>Hi, {{ user.username }}!</h1>

{% for post in posts %}

<div><p>{{ post.author.username }} says: <b>{{ post.body }}</b></p></div>

{% endfor %}

{% endblock %}

既然 base.html 现在可以承担通用的页面结构，我将这些元素从 index.html 中移除了，只留下了内容部分。extends 语句在两个模板间建立了继承关系，因此 Jinja2 知道当要渲染 index.html 的时候，需要将其嵌入到 base.html。两个模板会来用名字 content 来匹配 block 语句，这也是 Jinja2 为什么知道如何将两个模板合并成一个。现在如果我需要为应用创建其他页面的话，我可以从同样的 base.html 来派生模板，这也是我可以使得应用的所有页面都看起来相似但是又不会感觉到重复。

附录C 软件使用说明书

生鲜外卖小程序主界面如图C-1所示，用户通过微信社交平台进入微信小程序主界面，界面最上方是搜索框，可供用户搜索想要的商品，首页中部菜单栏滑动点击可转到不同分类的生鲜商品页面。底部菜单栏点击可进入“购物车”、“我的”界面。



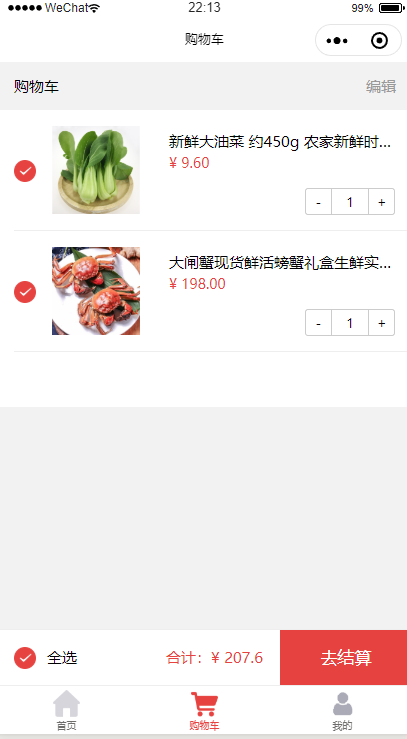
图C-1小程序主界面

商品详情页面展示了商品图片、商品标题、商品价格、商品介绍、用户评价等商品相关信息，以及在下方栏目上固定展示了“加入购物车”以及“立即购买”等按钮图形，用户可点击对商品进行加购及下单购买，商品详情子界面如图C-2所示。



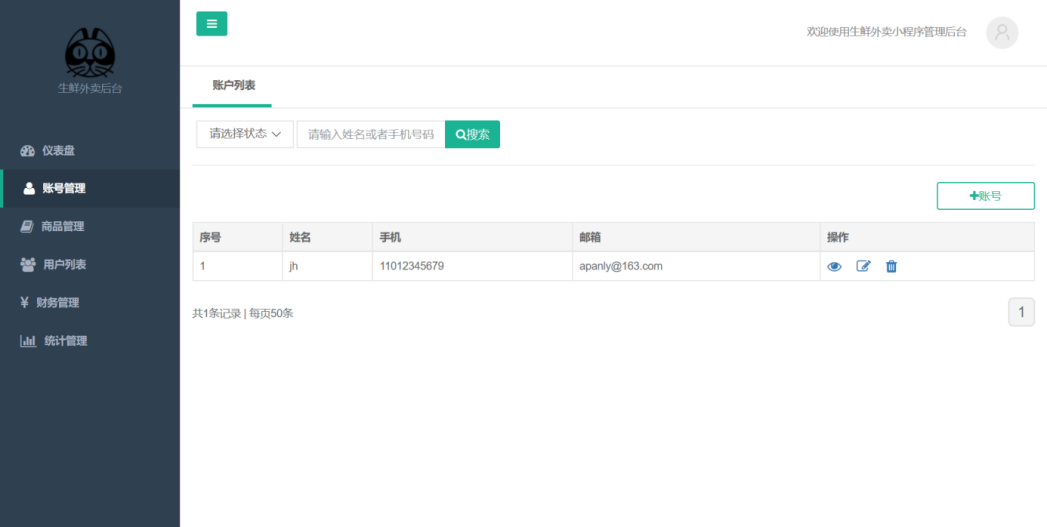
图C-2商品详情子界面

购物车界面展示了用户加入购物车的商品信息，如商品图片、商品名、商品价格和商品数量等信息，以及在下方栏目上固定展示了“去结算”等按钮图形，用户可点击对商品进行下单购买，购物车子界面如图C-3所示。



图C-3购物车子界面

后台管理主页面在侧边栏展示了账号管理、商品管理、用户列表、财务管理、统计管理等栏目信息，商家管理员可通过直接点击进入不同栏目页面中。后台管理子系统主界面如图C-4所示。



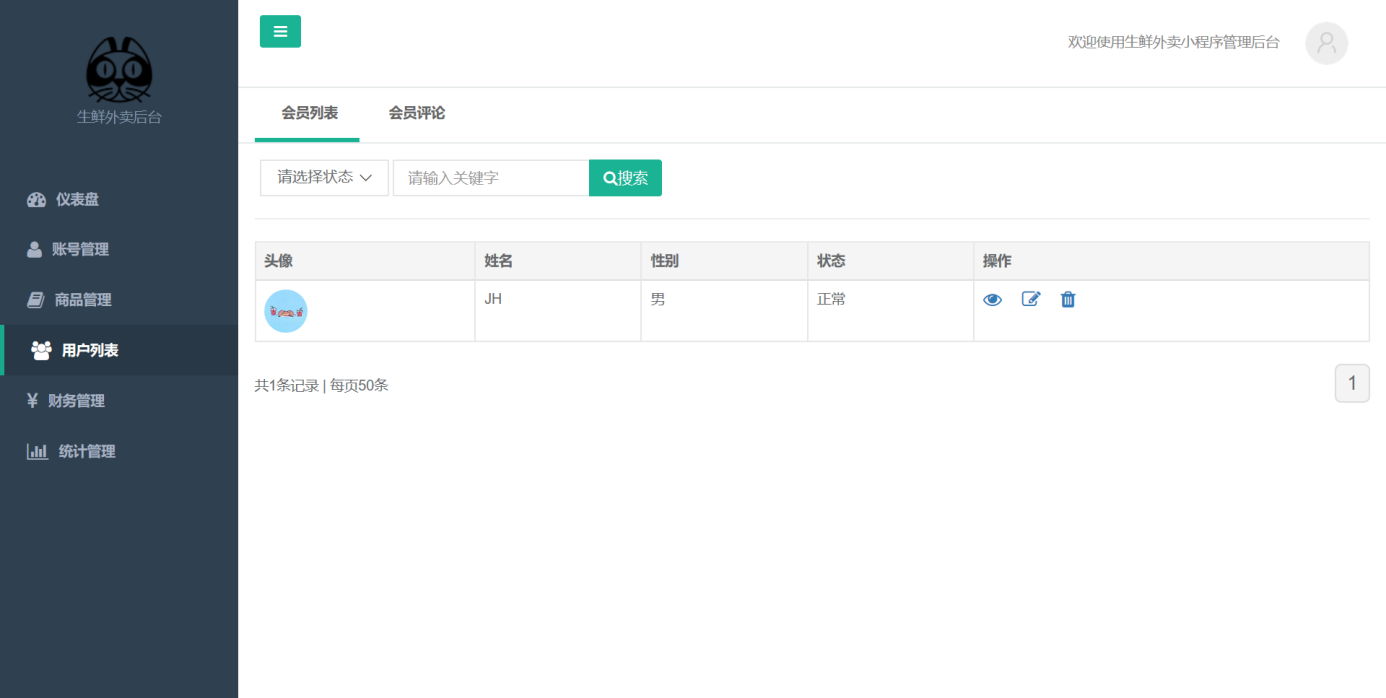
图C-4后台管理主界面

商品管理页面在顶部展示了商品列表及分类列表两个栏目信息，通过点击进入可查看商品信息列表及分类信息列表。页面上方设有搜索框，商家可进行商品搜索。右上角设有添加商品按钮，商家可点击进入商品添加页面实现上架商品。商品信息列表展示了商品名、分类、价格、库存、入库信息等商品相关信息，商品管理子界面如图C-5所示。



图C-5商品管理子界面

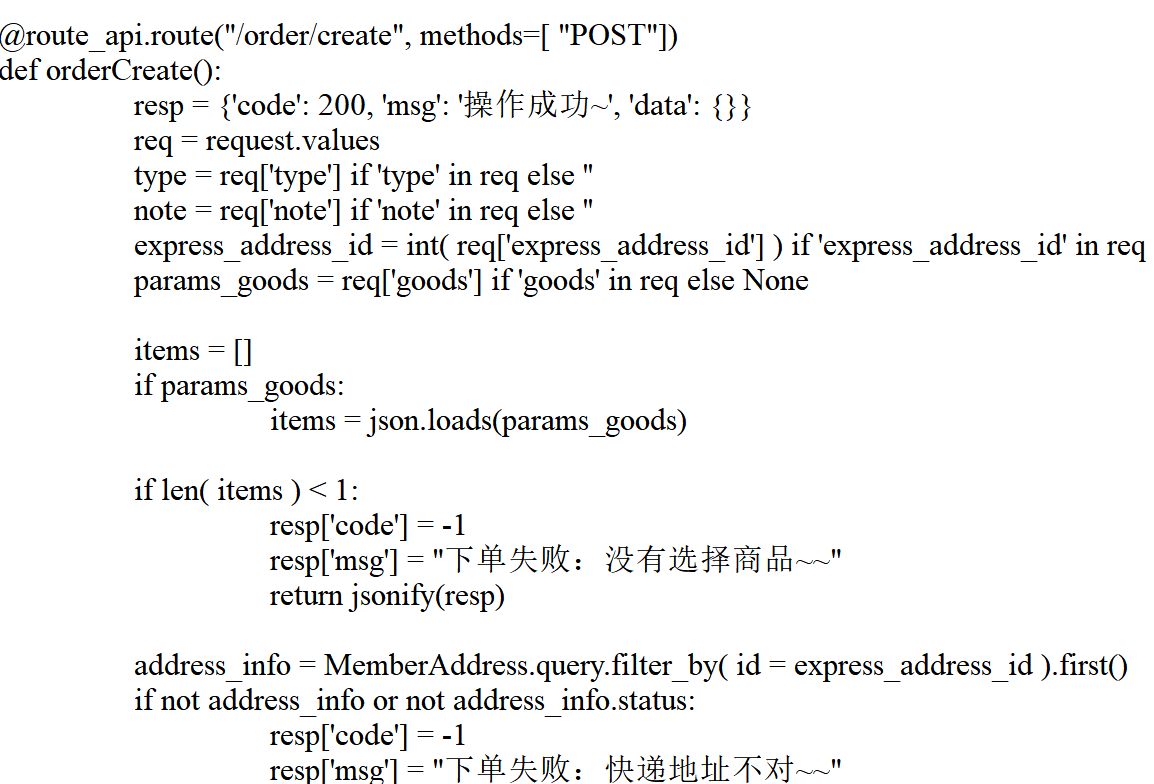
用户列表页面在顶部展示了用户列表及用户评论两个栏目信息，通过点击进入可查看用户信息列表及评论信息列表。页面上方设有搜索框，商家可通过关键字搜索用户。用户信息列表展示了用户头像、姓名、性别、状态等商品相关信息，点击操作框内不同图标按钮可对用户进行查看详情、便捷信息和删除用户等操作，用户管理子界面如图C-6所示。



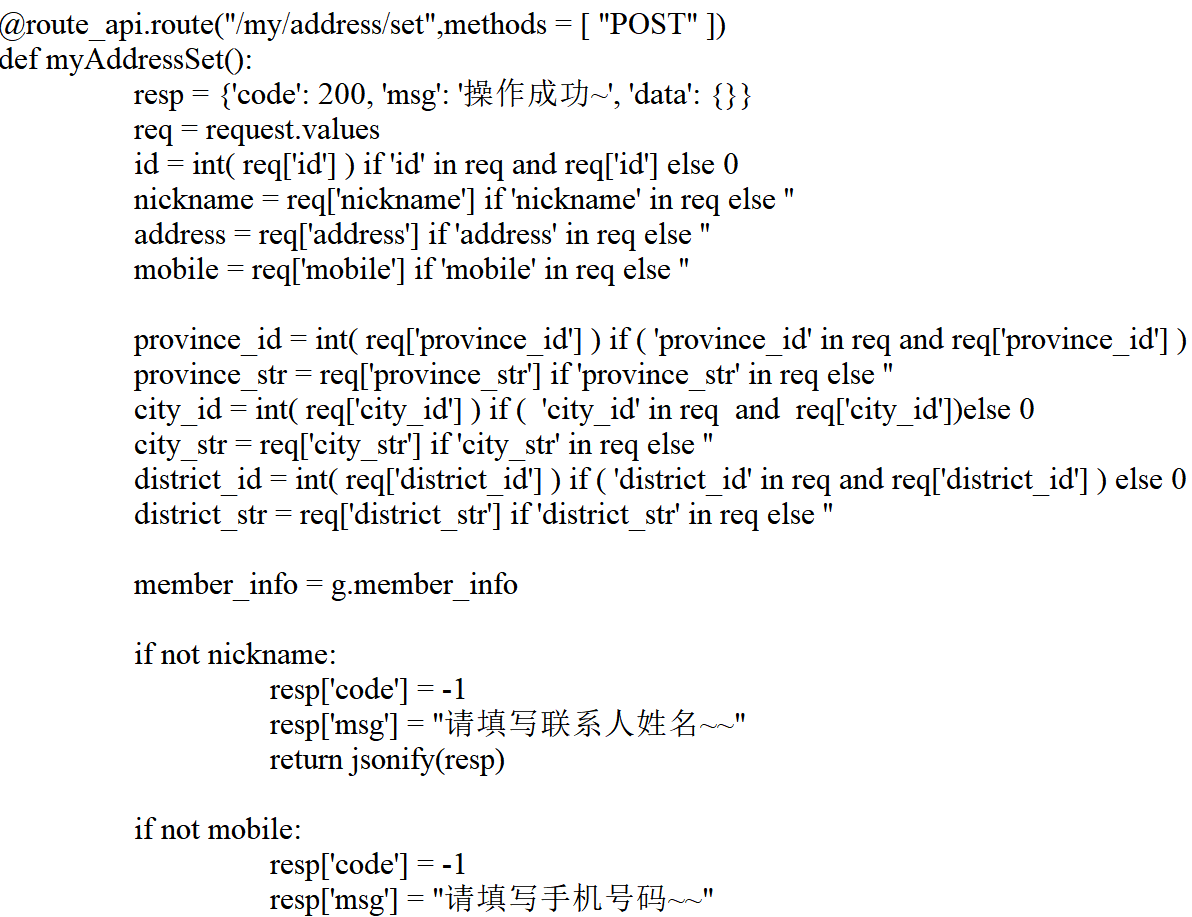
图C-6用户管理子界面

附录D 主要源代码

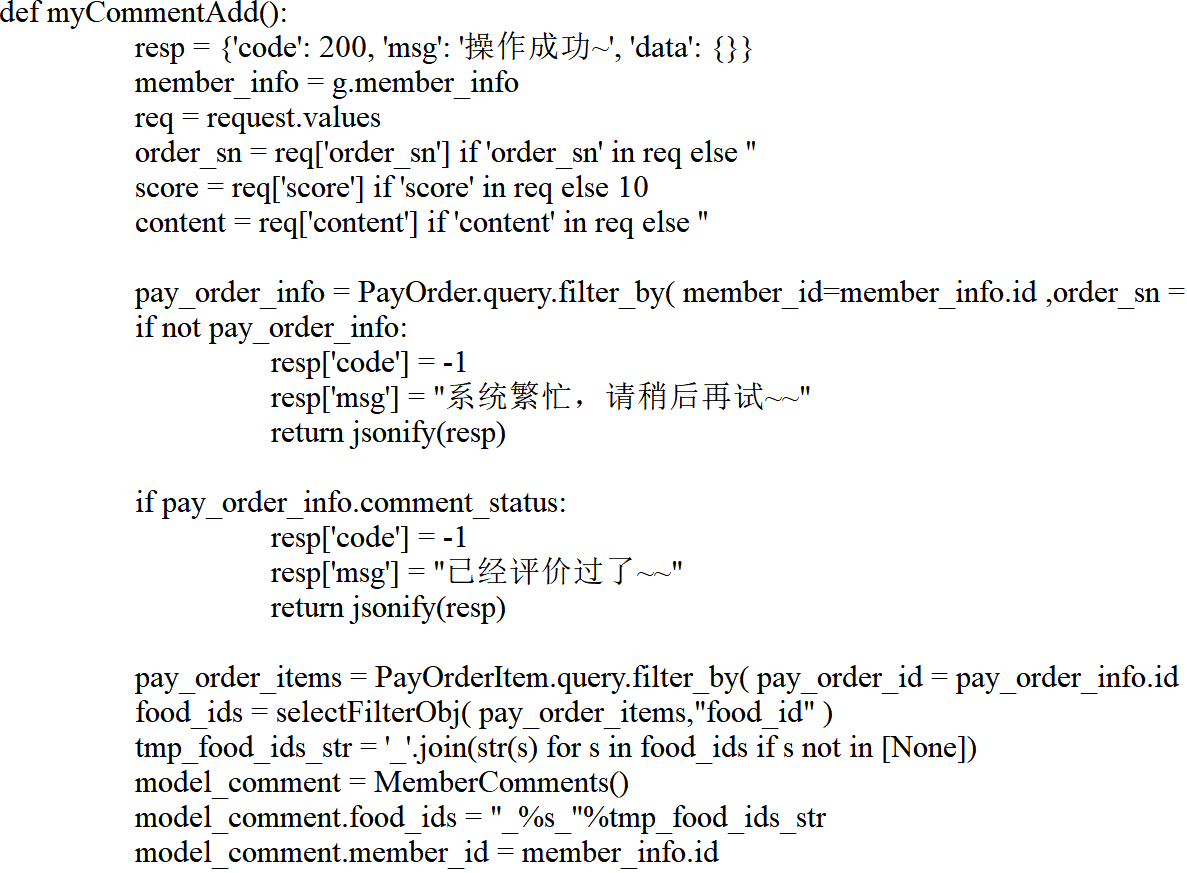
用户下单业务代码实现如下所示：



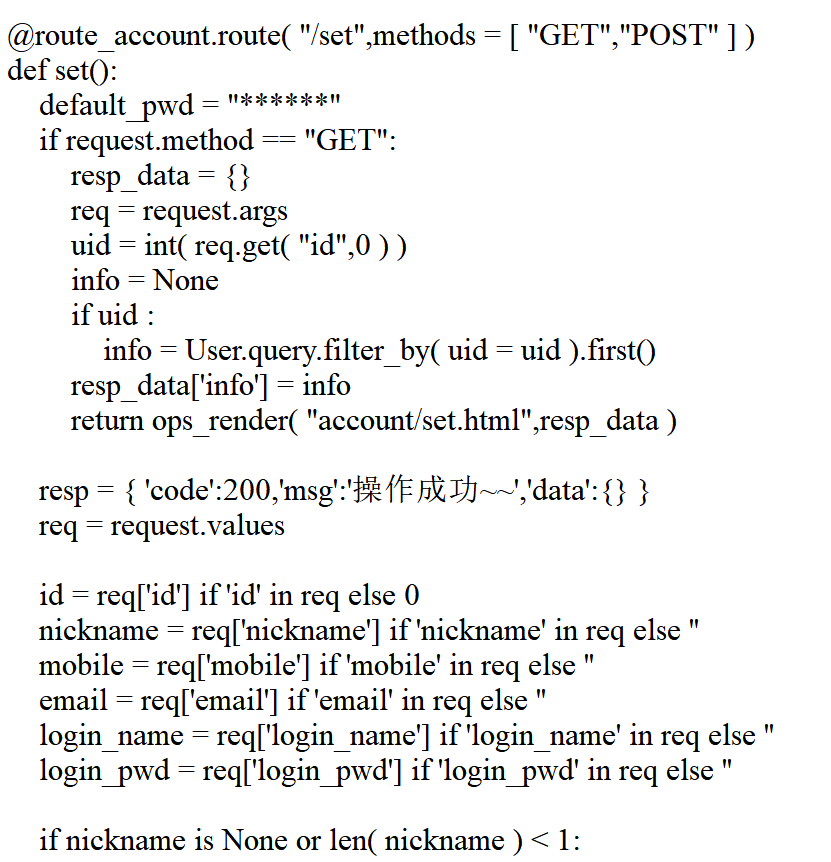
添加收货地址业务代码实现如下所示：



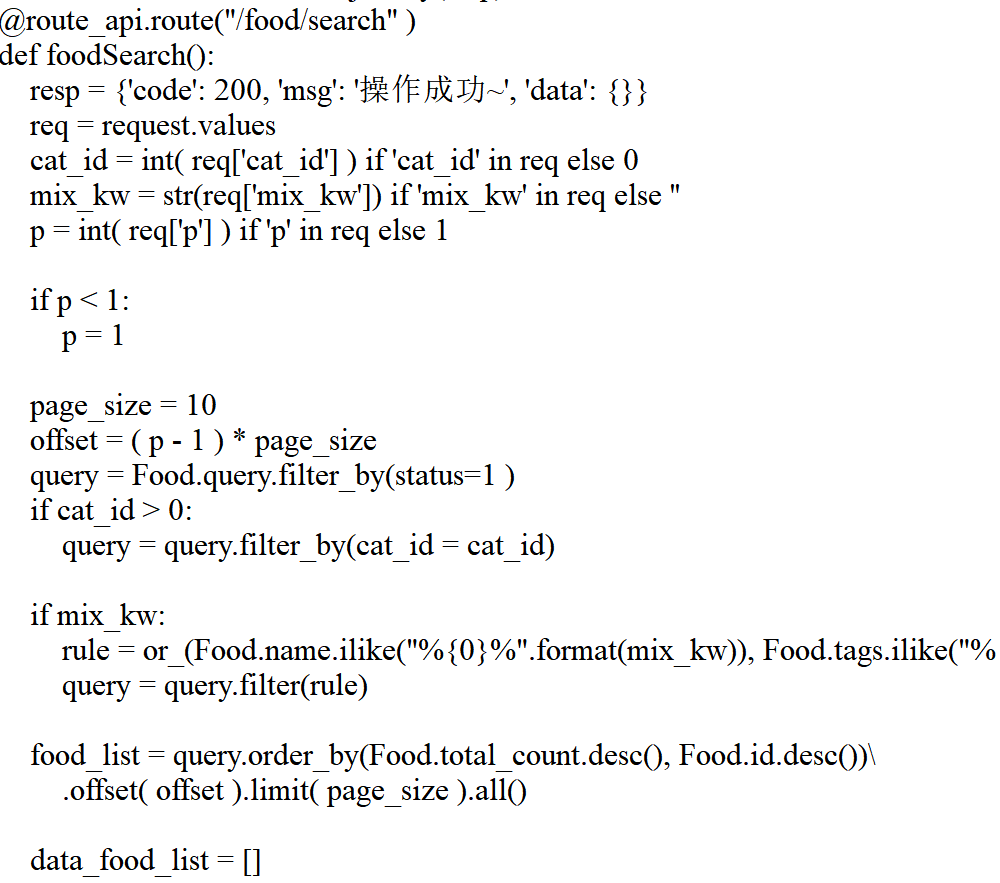
用户评论业务代码实现如下所示：



添加后台登录账号业务代码实现如下所示：



搜索商品业务代码实现如下所示：



查看收货地址业务代码实现如下所示：

