**杭州电子科技大学信息工程学院**

**本科毕业设计**

**（2020届）**

|  |  |
| --- | --- |
| **题 目** | 众包同城跑腿配送平台 |
| **系** | 计算机系 |
| **专 业** | 软件工程 |
| **班 级** | 软件工程3班 |
| **学 号** | 16905530 |
| **学生姓名** | 王泽华 |
| **指导教师** | 金洁洁 |
| **完成日期** | 2019年5月 |

# 摘要

随着信息技术发展的突飞猛进，互联网已经延伸到大众生活的方方面面。其中，电子商务是对大众的生活方式改变最大的领域之一。我国的跑腿公司大概兴起于2005年下半年，但是发展速度之快也可以说是星火燎原。跑腿公司这类新兴行业的出现，也促进了社会分工的进一步细化，因此说跑腿公司的存在是有其根本依据的。这样的新兴服务行业不但能够满足一部分社会需要，还能解决一部分人的就业问题，是有其存在和发展的价值的。

本课题以Node.js作为服务器端的开发语言，用MongoDB为数据库，构建一个MVC模式的在线导购平台。同时，通过使用百度地图API、AJAX等技术实现用户体验良好的地理位置服务（LBS）。实体店店主在申请账号后可以发布与维护商品信息，用户可以根据商品关键字和地理位置搜索商品，检索出目标位置附近的匹配商品。用户可以对自己感兴趣的商品进行在线预约，最后到实体店进行消费。

本论文介绍了基于LBS的O2O应用的研究背景与现状，对实现平台所使用的技术进行了概述，并对平台的业务需求进行分析。此外，论文以“从整体设计到实现细节”的顺序对平台架构、主要功能的业务逻辑和具体实现进行了阐述。

关键词：众包模式；同城跑腿；Spring Cloud；redis；分布式；

**ABSTRACT**

With the rapid development of information technology, The Internet have been a crucial part in people's daily life. As one of the important fields of the Internet, e-commerce has brought a completely different lifestyle to public. People can shopping without leave home, and the costs of products was reduced through the intermediate links were decreased, the traditional e-commerce have proved its value to social life. However, there are a number of problems still remain to be solved. For instance, the quality of products, building trust between buyers and sellers, the guarantee of after-sales service and the payment security. But the LBS and O2O mode exactly are a feasible solution to deal with these defects. This mode could keep information's high transparency and transmission as the traditional e-commerce did. Moreover, it offers a better consumption experience to customers in the offline.

This subject uses Node.js as development language at server-side and MongoDB as database. In addition, it uses Baidu Map API (electronic map service) and AJAX to provide a platform with good interactivity and location-based service. The staff of stores could upload and edit commodity information after they got a account. And the customers in the platform could search goods by keywords and target address. The platform will show all the expected products nearby the target address.

The paper introduces the research background and the current status of the LBS and O2O mode. It offers a summary about the techniques which be used in this platform. It will make business analysis about the platform's requirements. Furthermore, this paper will state the technical architecture and the major functions' business logic and the specific implementation.

**Key words:** e-commerce; LBS; O2O; Node.js; MongoDB

目录

[摘要 1](#_Toc40258560)

[1 绪论 1](#_Toc40258561)

[1.1 本课题研究背景 1](#_Toc40258562)

[1.2 国内外研究现状 2](#_Toc40258563)

[1.2.1 国外研究现状 2](#_Toc40258564)

[1.2.2 国内研究状态 3](#_Toc40258565)

[1.3 本课题研究内容 4](#_Toc40258566)

[1.4 本论文的组织结构 5](#_Toc40258567)

[2 系统需求分析 6](#_Toc40258568)

[2.1 用户需求分析 6](#_Toc40258569)

[2.1.1 客户 6](#_Toc40258570)

[2.1.2 跑腿员 6](#_Toc40258571)

[2.1.3 后台管理员 6](#_Toc40258572)

[2.2 功能需求分析 6](#_Toc40258573)

[2.2.1 跑腿客户端 6](#_Toc40258574)

[2.2.2 配送员端 7](#_Toc40258575)

[2.2.3 后台管理系统 7](#_Toc40258576)

[2.3 数据流分析 7](#_Toc40258577)

[3 系统总体设计 8](#_Toc40258578)

[3.1 系统整体架构 8](#_Toc40258579)

[3.1.1 系统技术架构 9](#_Toc40258580)

[3.1.2 各功能模块基本架构 11](#_Toc40258581)

[3.2 系统流程设计 12](#_Toc40258582)

[3.3 系统用例分析 13](#_Toc40258583)

[3.4 系统数据模型设计 16](#_Toc40258584)

[3.5 系统特色 18](#_Toc40258585)

[3.6 技术选型 18](#_Toc40258586)

[4 系统详细设计 19](#_Toc40258587)

[4.1 跑腿用户发布模块说明 19](#_Toc40258588)

[4.1.1 功能描述 19](#_Toc40258589)

[4.1.2 流程逻辑 19](#_Toc40258590)

[4.1.3 功能项 19](#_Toc40258591)

[4.1.4 设计方法（算法或SQL） 19](#_Toc40258592)

[4.1.5 注释设计 20](#_Toc40258593)

[4.1.6 限制条件 20](#_Toc40258594)

[4.1.7 尚未解决的问题 20](#_Toc40258595)

[4.2 跑腿用户订单支付模块说明 20](#_Toc40258596)

[4.2.1 功能描述 20](#_Toc40258597)

[4.2.2 流程逻辑 21](#_Toc40258598)

[4.2.3 功能项 21](#_Toc40258599)

[4.2.4 设计方法（算法或SQL） 21](#_Toc40258600)

[4.2.5 注释设计 21](#_Toc40258601)

[4.2.6 限制条件 21](#_Toc40258602)

[4.2.7 尚未解决的问题 21](#_Toc40258603)

[4.3 跑腿用户地址管理模块说明 22](#_Toc40258604)

[4.3.1 功能描述 22](#_Toc40258605)

[4.3.2 流程逻辑 22](#_Toc40258606)

[4.3.3 设计方法（算法或SQL） 22](#_Toc40258607)

[4.3.4 限制条件 22](#_Toc40258608)

[4.4 跑腿用户订单评价模块说明 23](#_Toc40258609)

[4.4.1 功能描述 23](#_Toc40258610)

[4.4.2 流程逻辑 23](#_Toc40258611)

[4.4.3 功能项 23](#_Toc40258612)

[4.4.4 设计方法（算法或SQL） 23](#_Toc40258613)

[4.5 跑腿用户订单管理模块说明 24](#_Toc40258614)

[4.5.1 功能描述 24](#_Toc40258615)

[4.5.2 流程逻辑 24](#_Toc40258616)

[4.5.3 功能项 24](#_Toc40258617)

[4.5.4 设计方法（算法或SQL） 24](#_Toc40258618)

[4.5.5 注释设计 25](#_Toc40258619)

[4.5.6 限制条件 25](#_Toc40258620)

[4.5.7 尚未解决的问题 25](#_Toc40258621)

[4.6 跑腿用户我的钱包模块说明 25](#_Toc40258622)

[4.6.1 功能描述 25](#_Toc40258623)

[4.6.2 流程逻辑 26](#_Toc40258624)

[4.6.3 功能项 26](#_Toc40258625)

[**4.6.4** **设计方法（算法或SQL）** 26](#_Toc40258626)

[4.6.5 注释设计 27](#_Toc40258627)

[4.6.6 限制条件 27](#_Toc40258628)

[4.6.7 尚未解决的问题 27](#_Toc40258629)

[4.7 跑腿员接单模块说明 27](#_Toc40258630)

[4.7.1 功能描述 27](#_Toc40258631)

[4.7.2 流程逻辑 27](#_Toc40258632)

[4.7.3 功能项 27](#_Toc40258633)

[4.7.4 设计方法（算法或SQL） 28](#_Toc40258634)

[**4.7.5** **注释设计** 29](#_Toc40258635)

[**4.7.6** **限制条件** 29](#_Toc40258636)

[**4.7.7** **尚未解决的问题** 29](#_Toc40258637)

[4.8 跑腿员任务列表模块说明 29](#_Toc40258638)

[4.8.1 功能描述 29](#_Toc40258639)

[4.8.2 流程逻辑 29](#_Toc40258640)

[4.8.3 功能项 29](#_Toc40258641)

[4.8.4 设计方法（算法或SQL） 29](#_Toc40258642)

[4.8.5 注释设计 30](#_Toc40258643)

[4.8.6 限制条件 30](#_Toc40258644)

[4.8.7 尚未解决的问题 30](#_Toc40258645)

[4.9 跑腿员历史任务模块说明 30](#_Toc40258646)

[4.9.1 功能描述 30](#_Toc40258647)

[4.9.2 流程逻辑 30](#_Toc40258648)

[4.9.3 功能项 30](#_Toc40258649)

[4.9.4 设计方法（算法或SQL） 30](#_Toc40258650)

[4.9.5 注释设计 31](#_Toc40258651)

[4.9.6 限制条件 31](#_Toc40258652)

[4.9.7 尚未解决的问题 31](#_Toc40258653)

[4.10 管理员订单管理模块说明 31](#_Toc40258654)

[4.10.1 功能描述 31](#_Toc40258655)

[4.10.2 流程逻辑 31](#_Toc40258656)

[4.10.3 功能项 31](#_Toc40258657)

[4.10.4 设计方法（算法或SQL） 32](#_Toc40258658)

[4.10.5 注释设计 33](#_Toc40258659)

[4.10.6 限制条件 33](#_Toc40258660)

[4.10.7 尚未解决的问题 33](#_Toc40258661)

[4.11 管理员用户管理说明 33](#_Toc40258662)

[4.11.1 功能描述 33](#_Toc40258663)

[4.11.2 流程逻辑 33](#_Toc40258664)

[4.11.3 功能项 34](#_Toc40258665)

[4.11.4 设计方法（算法或SQL） 34](#_Toc40258666)

[4.11.5 注释设计 35](#_Toc40258667)

[4.11.6 限制条件 35](#_Toc40258668)

[4.11.7 尚未解决的问题 35](#_Toc40258669)

[4.12 管理员跑腿员管理说明 35](#_Toc40258670)

[4.12.1 功能描述 35](#_Toc40258671)

[4.12.2 流程逻辑 35](#_Toc40258672)

[4.12.3 功能项 35](#_Toc40258673)

[4.12.4 设计方法（算法或SQL） 36](#_Toc40258674)

[4.12.5 注释设计 37](#_Toc40258675)

[4.12.6 限制条件 37](#_Toc40258676)

[4.12.7 尚未解决的问题 37](#_Toc40258677)

[4.13 管理员营销推广模块说明 37](#_Toc40258678)

[4.13.1 功能描述 37](#_Toc40258679)

[4.13.2 流程逻辑 38](#_Toc40258680)

[4.13.3 功能项 38](#_Toc40258681)

[4.13.4 设计方法（算法或SQL） 38](#_Toc40258682)

[4.13.5 注释设计 39](#_Toc40258683)

[4.13.6 限制条件 40](#_Toc40258684)

[4.13.7 尚未解决的问题 40](#_Toc40258685)

[4.14 管理员计费规则模块说明 40](#_Toc40258686)

[4.14.1 功能描述 40](#_Toc40258687)

[4.14.2 流程逻辑 40](#_Toc40258688)

[4.14.3 功能项 40](#_Toc40258689)

[4.14.4 设计方法（算法或SQL） 40](#_Toc40258690)

[4.14.5 注释设计 41](#_Toc40258691)

[4.14.6 限制条件 41](#_Toc40258692)

[4.14.7 尚未解决的问题 42](#_Toc40258693)

[4.15 管理员统计模块说明 42](#_Toc40258694)

[4.15.1 功能描述 42](#_Toc40258695)

[4.15.2 流程逻辑 42](#_Toc40258696)

[4.15.3 功能项 42](#_Toc40258697)

[4.15.4 设计方法（算法或SQL） 42](#_Toc40258698)

[4.15.5 注释设计 45](#_Toc40258699)

[4.15.6 限制条件 45](#_Toc40258700)

[4.15.7 尚未解决的问题 45](#_Toc40258701)

[6 系统实现 46](#_Toc40258702)

[6.1 微服务注册发现组件—EurekaService 46](#_Toc40258703)

[6.2 微服务网关—ZuulProxy 47](#_Toc40258704)

[6.3 微服务的REST调用和熔断机制 47](#_Toc40258705)

[6.3.1 使用Feign实现声明式REST调用 47](#_Toc40258706)

[6.3.2 整合Feign和Hystrix 48](#_Toc40258707)

[6.4 跑腿订单微服务 48](#_Toc40258708)

[6.4.1 订单的查询 48](#_Toc40258709)

[6.4.2 订单状态的变动 49](#_Toc40258710)

[6.4.3 订单的评价 50](#_Toc40258711)

[6.4.4 订单的添加 50](#_Toc40258712)

[6.4.5 订单的统计 51](#_Toc40258713)

[6.4.6 订单的导出 51](#_Toc40258714)

[6.4.7 订单的计费规则 52](#_Toc40258715)

[6.4.8 流量统计 53](#_Toc40258716)

[6.5 跑腿员微服务 55](#_Toc40258717)

[6.5.1 跑腿员接单 55](#_Toc40258718)

[6.5.2 跑腿员任务 55](#_Toc40258719)

[6.5.3 跑腿员个人信息 56](#_Toc40258720)

[6.6 用户微服务 56](#_Toc40258721)

[6.6.1 用户下单 56](#_Toc40258722)

[6.6.2 用户个人信息 57](#_Toc40258723)

[6.6.3 用户历史订单 57](#_Toc40258724)

[7 系统实现 58](#_Toc40258725)

[7.1 跑腿员功能实现（HTML5） 58](#_Toc40258726)

[7.1.1 接单 58](#_Toc40258727)

[7.1.2 我的任务 60](#_Toc40258728)

[7.1.3 个人中心 61](#_Toc40258729)

[7.2 客户端功能实现(HTML5) 62](#_Toc40258730)

[7.2.1 首页 62](#_Toc40258731)

[7.2.2 帮买 63](#_Toc40258732)

[7.2.3 我的地址&我的订单 65](#_Toc40258733)

[7.2.4 我的评价&我的钱包 66](#_Toc40258734)

[7.3 后台管理系统功能实现 67](#_Toc40258735)

[7.3.1 概览界面 67](#_Toc40258736)

[7.3.2 计费规则 68](#_Toc40258737)

[7.3.3 订单管理 68](#_Toc40258738)

[7.3.4 客户管理 71](#_Toc40258739)

[7.3.5 配送员管理 72](#_Toc40258740)

[7.3.6 营销 73](#_Toc40258741)

[7.3.7 统计报表 74](#_Toc40258742)

[8 系统测试 77](#_Toc40258743)

[8.1 接口测试用例 77](#_Toc40258744)

[8.1.1 被测试对象的介绍 77](#_Toc40258745)

[8.1.2 测试范围与目的 80](#_Toc40258746)

[8.1.3 测试环境与测试辅助工具的描述 80](#_Toc40258747)

[8.1.4 测试驱动程序的设计 80](#_Toc40258748)

[8.1.5 接口测试用例 80](#_Toc40258749)

[8.2 功能测试用例 83](#_Toc40258750)

[8.2.1 被测试对象的介绍 83](#_Toc40258751)

[8.2.2 测试范围与目的 83](#_Toc40258752)

[8.2.3 测试环境与测试辅助工具的描述 83](#_Toc40258753)

[8.2.4 功能测试用例 84](#_Toc40258754)

[9 结语 87](#_Toc40258755)

[致谢 88](#_Toc40258756)

[参考文献 89](#_Toc40258757)

[10 附录 90](#_Toc40258758)

[表汇总 90](#_Toc40258759)

[表明细 91](#_Toc40258760)

# 绪论

## 本课题研究背景

 随着互联网深入地融入到大众的日常生活中，电子商务平台在现今的经济市场中可谓如鱼得水。艾瑞咨询在其发布的国内网上购物行业检测报（2013年年度）中预计，到2015年，国内网购规模将达到2.55万亿元人民币，占社会消费零售总额的8.6%，而其在2014年4月份发布的2013年中国中小企业B2B电子商务市场应收规模[1]中也指出仅中小企业便在13年达到了194.5亿元人民币的营收，到2017年，这些企业的营收额预计将达到414.7亿元的惊人规模。艾瑞咨询统计和预计的中国中小企业B2B市场总营收规模如图1-1所示。

图1-1 艾瑞咨询公司2009-2017年中国中小企业B2B市场总营收规模[1]

电子商务不仅仅推动的是企业的营业能力与经济的增长，其更重要的是对信息技术发展的促进和对大众消费者生活方式的一种革命性改变。对于前者，其促进了互联网软件对于分布式、集群、高并发处理等技术要求，还衍生了如支付宝、财付通等第三方支付的出现。对于后者，企业可以凭借减少中间环节和降低高额的成本获益，消费者也可足不出户的以比实体店更低的价格购买心仪的商品。

但是，“创新”一直是推动电子商务领域飞速发展的一个核心，平台方永远不会满足当前的销售模式和手段，一直以多元化的方式开拓新的领域。而买卖双方信任度的建立和售后服务、质量的参差不齐等问题也引发了诸多客户与销售方的矛盾。以目前来看，传统电商平台似乎不能成为网上购物的“银弹”[2]。传统零售业和实体店在当前被电商平台大量冲击和蚕食的背景下无比迫切地需要一条与线上销售整合的道路；传统电商带来的诸如服务、安全性等一系列弊端和矛盾也需要一种新的电商模式来消除。在这样的背景下，基于LBS（Location-based services）的O2O（Online To Offline）模式的出现和其成为大多数电商平台转型的方向成为了电商发展的必然之路。

O2O的电商营销模式对线上、线下购物进行了整合，取其精华去其糟粕。它延续了线上销售对互联网开放性好、透明度高和信息传播快等特点的充分利用，也通过线下消费和服务带来更好的消费体验与品质、售后的保障，从而为买卖双方带来双方共赢、互利互惠的局面。特别是在中国，BAT已经锁定线上消费的大局，而阿里、腾讯等互联网寡头也纷纷开始布局线下线上整合的2014年，传统电商进军O2O必然是最好的一年。移动互联网和地图导航技术在近几年来的发展已经为O2O奠定了良好的基础与实施平台，大数据时代的到来也为该模式发展中数据分享环节提供了有力的支撑[3]。政府在未来对O2O上的政策资源的支持鼓励也绝对是预料之中的，可以说属于O2O的大时代已经拉开了序幕[4]。本课题正是基于上述背景，旨在通过以Node.js、MongoDB和百度地图API作为技术支撑，并严格按照软件工程的要求，建立一个基于LBS的O2O在线导购平台。

## 国内外研究现状

### 国外研究现状

在国外，基于地理位置的电子商务平台中，Foursquare是一个不得不关注的平台。该平台起初未将自己标榜为电子商务平台，而是将其定位为社交定位。其上线一年后用户数量便突破百万大关，而在上线一年半后后就突破了300万，该平台的用户增长速度完全不亚于Twitter和Facebook。该平台与电商的切入点在于通过在平台以徽章和广告投入，并在客户端显示用户所在位置附近的诸如星巴克、麦当劳等实体店的门店位置等方式进行合作。单从技术层面讲，Foursquare并没有提供任何突破性的建树，但是其带来的与众不同的商业模式，成未了后来诸多基于LBS和本地化电子商务平台的参考的对象。

在eBay于10年12月公布对本地购物搜索引擎Milo进行收购后，更是让业界了解到这种基于地理位置服务的电商模式的价值。Milo涵盖全美5万多家店铺的300多万种商品，用户可以根据自己需求在网站上针对地区位置搜索拥有相应商品的店铺和商品的具体信息。而本次收购的价格虽未被正式披露，但据透露，其收购价格达到了7500万美金[5]。此外2011年宣布添加LBS购物功能的Google+，被亚马逊CEO和投行高盛一致看好并投资3200万美金的O2O租车公司Uber等，都再一次证实了基于LBS的电商模式在国外正发展的如火如荼。

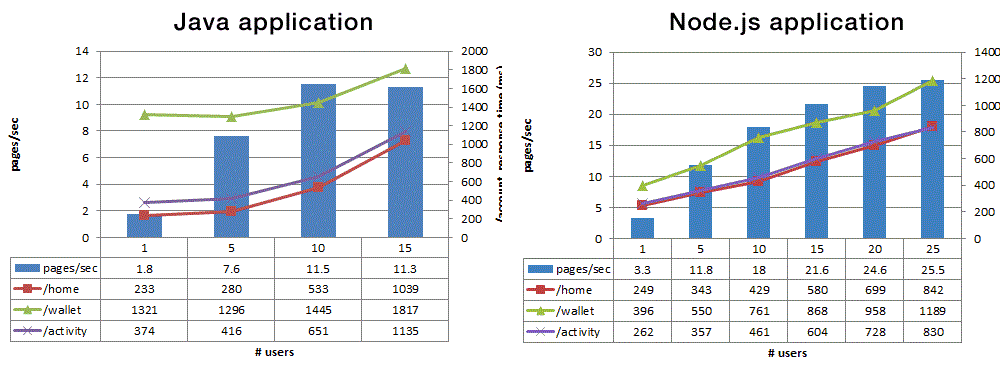
 从平台实现的技术层面来看，Node和MongoDB的开发模式在国外也已得到业界的认同和推崇。VMware的云计算平台Cloud Foundry和Microsoft的Azure云平台也纷纷在第一时间向Node抛出了橄榄枝。团购网站的鼻祖Groupon也将其大量业务实现用Node重新进行开发[6]。美国电子商务代表企业之一ebay也于2013年将它的支付模块PayPal从Java迁移到Node平台[7]，该团队用实践证明了Node在性能和开发效率上的优越性。如图1-2所给出的PayPal分别在Node平台与Java平台处理用户请求的性能分析结果所示，Node每秒钟能处理两倍与Java平台能处理的用户请求，并平均减少了35%的响应时间。

图1-2 PalPay团队对其项目在Java和Node平台上运行的性能测试结果[7]

### 国内研究状态

相对于国外基于位置的电商发展状况，国内同类型的平台虽起步较晚，但发展也较为迅速。毋庸置疑的是，这种模式好坏以否已经在国外得以证明，而在国内，团购网站以本地化+团购的O2O模式成为了第一批吃螃蟹的。拉手网作为第一家结合了团购与LBS的网站，短短一年之间便以迅雷不及掩耳之势在“千团大战”中脱颖而出，成为大众消费者最钟爱的团购网站之一。而美团则也在本地化团购中投入了大量的人力与资金，很好的根据用户的所在位置整合了线上消费与线下服务。统计到2013年十一月底，美团网在该年年的交易额便已经将近达到131亿元，平台上的商户涵盖两百多个城市，数量也已突破27万个。

在团购网站向LBS转型取得的成功的背景下，国内许多互联网公司也分分向该领域试水，从以往的纸上谈兵进入了真刀真枪的阶段。其中典型的模式包括LBS+娱乐，LBS+生活服务，LBS+户外活动等。而使用基于地理位置服务的用户也从2010年开始从300多万以近300%的增长率上涨到13年的8000多万人。而国内互联网界的龙头企业也纷纷加入到了LBS模式的应用中，百度收购糯米网显然是将在14年开始向LBS等本地服务发力。而在阿里巴巴中，对地图、购物和团购等的整合也井然有序的进行着[8]，并在2014年2月以16亿美元的高价完成了对数字地图公司高德的全面收购，以53.7亿港币对银泰商业进行战略投资，进一步布局020市场。国内的另一家互联网巨头腾讯也在当月以4亿多美元入股了国内LBS+O2O模式中最成功的企业之一大众点评网，弥补其在该领域的短板。仅从2013年来看，O2O的市场规模就达到了约为1200亿，增幅为64%[9]。垂直化、移动化的O2O模式已然成为了线上平台眼中的大蛋糕，各大巨头企业都在为抢占市场份额进行铺路，争做线下线上整合的水电煤。

在国外热受开发人员追捧的Node在国内同样也得到了业界的注目。虽然阿里并未发布基于Node平台的Web应用，但据阿里数据平台数据产品部资深工程师，《深入浅出NodeJS》的作者朴灵在他的书中所言，阿里内部已经有很多基于Node平台的应用正在使用。而网易的Pomelo也通过该技术实现的一款性能极好的分布式游戏服务Framework，并已在中国开源社区公开源码[10]。由于Node本身的优点和诸多国内互联网知名企业的推动，Node在中国软件业也必是前途一片光明。

## 本课题研究内容

本课题采用Node.js和MongoDB等技术实现了一个基于LBS的O2O在线导购平台，具体研究内容有以下几方面：

（1）对各类传统的电商平台的发展状况和这些平台本地化购物的建设进行研究。

（2）对诸如Milo、Google+等已有的基于LBS的购物平台或相关模块进行分析研究，了解其业务流程和主要特点。

（3）学习和研究使用Node构建Web应用，并对MVC框架Express进行深入理解。Express是一款非常灵活的Node框架，其所有功能模块都以中间件的形式存在，使开发人员可以根据实际需求对所需的模块进行最少化的导入。该框架提供了对应用进行分层设计的规范，使应用更易维护和有较好的健壮性。其使用路由转发技术实现MVC中的控制层，自动对客户端请求的URL和其HTTP请求方法的类型进行解析和转发。通过使用模板引擎进行页面渲染，以此来实现视图层。目前主流的模板引擎有十多种，本课题使用最为通用的EJS作为模板引擎。

（4）对JavaScript的面向对象设计和Prototype模式的使用进行深入研究，并用这些技术来完成平台的模型层。

（5）对MongoDB的文档模型及其相关操作进行深入研究和学习。

（6）对百度 Map API进行学习研究，通过实践掌握利用Node和jQuery对该API进行异步调用和数据交互。

（7）结合Node的开发规范和软件工程的相关概念，对平台制定详细的开发流程步骤。

（8）采用相关技术与要求对平台进行开发。

## 本论文的组织结构

本论文共有7大章。

第1章绪论，主要介绍了本平台的研究背景，众包跑腿平台的国内外研究情况。同时，该章节阐述了本课题的研究内容及论文的组织结构。

第2章为系统需求分析，该章节讲述了用户的需求分析，功能需求分析，以及数据流图表。

第3章为系统总体设计，该章节综述了平台的整体架构，系统的基本流程设计，用例分析和数据库设计，同时还提及了系统的相应特色和采用的新技术。

第4章为平台分析与设计，主要包括对平台功能的需求分析，平台总体架构的叙述，数据库需求分析，主要模块的设计等，并给出了相关的用例图、时序图和平台的开发环境等。

第5章为平台主要模块的实现，着重介绍了平台的几个核心模块的实现方式，包括用户购物模块，店铺信息管理模块等。

第6章为平台测试，主要介绍了对平台的功能测试，并对测试结果进行分析。

第7章为总结及展望，主要是对本次课题的工作进行总结，并提出对基于LBS的在线导购平台的进一步展望。

# 系统需求分析

开发系统的第一步是进行需求分析。需求分析的好坏直接决定着系统能否真正满足用户的需要。分析需求是数据库管理系统开发的第一步，也是最重要的一步。根据相关的调研和资料查找，确定了以下需求。

## 用户需求分析

### 客户

1. 用户期望，可以有多种跑腿任务，比如帮买，帮送，帮取货等。
2. 对于跑腿这件事，用户并不希望自己要去下载或使用一个重量级的APP，而是希望以更便捷的方式来卖废品。
3. 有一个透明的跑腿价格体系。
4. 一个能够展示历史获得金额的账单可以帮助用户更好管理他们的收入和支出。

### 跑腿员

1. 可以在线查看用户发布的任务，跑腿任务中有写明起点、终点，以及并可以选取某一个跑腿任务开始。
2. 在任务列表中，跑腿员可以查看待取货的任务，配送中的任务，已完成的任务。
3. 可以对自己的佣金收入有清晰的展示，多劳多得。
4. 期望能够以最短的时间、最短的距离，配送最多的订单。

### 后台管理员

1. 后台管理员可以在后台管理系统查看所有订单的情况，并根据条件进行查询。同时还可以导出xml格式。
2. 可以在后台查看用户的订单情况，用户的评论，用户的相关信息。可以新建跑腿员账户，对跑腿员账户进行修改，地域分配、工资管理等功能。
3. 可以查看所有订单统计情况，财务统计，流量分析等。

## 功能需求分析

在本次系统中，主要分为三大模块，一个是针对跑腿用户使用的客户端，一个是针对配送员使用的配送员端，最后是一个跑腿后台管理系统。

### 跑腿客户端

用户在跑腿客户端中可以选择跑腿类别，填写相关信息之后下单，之后还可以在我的账户中查看我的历史订单和消费情况。还有地址管理等功能。

### 配送员端

配送员可以在配送员端中查看用户发出的订单，并可以根据自己的需要进行抢单操作，抢到的单可以在我的任务中查看。在完成任务之后，可以获得相关的赏金并可以在我的账户中查看相应的情况。

### 后台管理系统

系统管理员可以在后台中查看所有跑腿订单的情况，指定跑腿费的计费规则，查看用户信息，新建跑腿员账户，修改跑腿员信息，财务统计，流量分析，以及跑腿订单大数据分析。

## 数据流分析

表 2‑1 数据流图

数据流图是[结构化分析方法](https://baike.baidu.com/item/%E7%BB%93%E6%9E%84%E5%8C%96%E5%88%86%E6%9E%90%E6%96%B9%E6%B3%95)中使用的工具,它以图形的方式描绘数据在系统中流动和处理的过程。本系统的数据流图如图3-1所示。主要有跑腿员接单，配送员抢单，后台管理员跑腿计费设置，订单查询等核心功能。主要用到了计费规则表，跑腿信息表等。

# 系统总体设计

## 系统整体架构

表 3‑1 系统整体架构

由表 3‑2可知，本系统主要分为三个端，三个端分别是配送员端，主要给配送员在手机上使用，具有抢单，查看历史配送订单的功能。用户端也是针对于手机APP，可以在上面发布跑腿，以及个人中心等相关功能。

### 系统技术架构

图 3‑2 网络拓扑图

拓扑结构图是指由网络节点设备和通信介质构成的网络结构图。网络拓扑定义了各种计算机、打印机、网络设备和其他设备的连接方式。换句话说，网络拓扑描述了线缆和网络设备的布局以及数据传输时所采用的路径。网络拓扑会在很大程度上影响网络如何工作。

图 3‑2展示了整个系统的拓扑结构，后台管理系统以及客户端均由restful api与后台交互，实现了前后端的分离，有利于把服务器的压力减小到最小，同时本系统还配备了redis缓存服务器，使系统更好的支持高并发访问。前台的请求首先通过web服务器，并将请求转发至redis服务器，若redis服务器已经有相关数据，则直接返回数据，如果没有相关数据，则转发到mysql数据库后进行查询。提高了系统的健壮性和可用性。

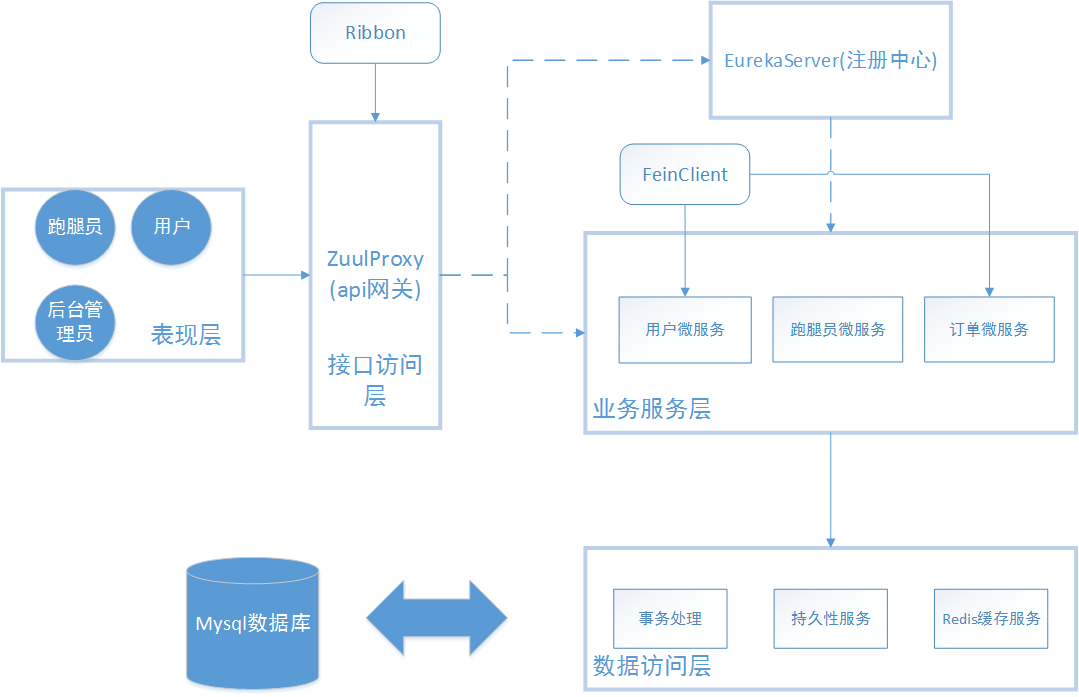
如图3-3所示，本系统分为表现层，接口访问层，业务服务层，数据访问层。

表 3‑3 项目微服务架构

视图层主要负责数据的展示，网页的展示。包含系统的显示逻辑，位于客户端。通过Restful API 向后台请求数据。

接口访问层主要负责对前台Restful API的解析，并通过API网关来转发请求。

业务层主要负责对具体问题的操作，在微服务架构中，会调取一个个的微服务用于实现相关的业务逻辑。

数据访问层主要负责对数据的处理，如数据的访问，事务处理，以及缓存。

Spring Cloud 相关组件解释

* 外部或者内部的非Spring Cloud项目都统一通过API网关（Zuul）来访问内部服务。
* 网关接收到请求后，从注册中心（Eureka）获取可用服务。
* 由Ribbon进行均衡负载后，分发到后端的具体实例。
* 微服务之间通过Feign进行通信处理业务。
* Hystrix负责处理服务超时熔断。
* Turbine监控服务间的调用和熔断相关指标。

### 各功能模块基本架构

表 3‑4 后台管理系统整体架构

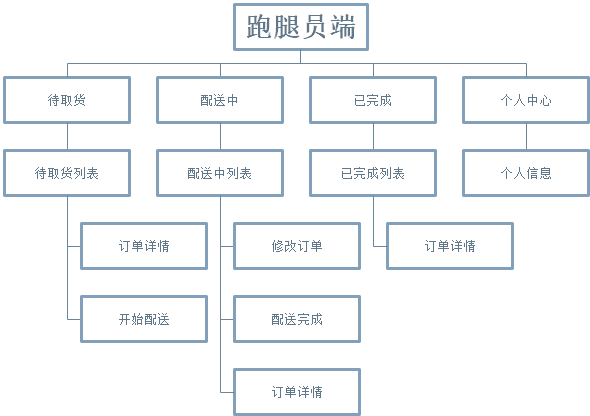
跑腿后台管理系统主要分了总体概览，跑腿设置，订单管理，客户管理，跑腿员管理，营销，统计分析这几个功能模块。总体概览中管理员可以实时查看订单统计数据，各个地区，跑腿大类的交易数据。跑腿设置模块中管理员可以进行跑腿计费的设置，跑腿分类的设置。订单管理中可以实现对跑腿订单的增删改查，客户管理中可以实现对跑腿客户的查询，客户数据的导出。跑腿员管理中可以实现对跑腿员的增删改查，跑腿员评论查看等功能。，管理员可在营销中实现轮播图的修改，优惠券的发放等功能。统计分析调用了百度统计api实现了对系统日活的访问。

图 3‑5 跑腿员端基本架构

跑腿员端主要有待取货、配送中、已完成，个人中心这几个功能。待取货中展示了需要配送的任务列表，列表上展示了各个任务的出发地，目的地，配送金额，配送内容等情况。配送中列表展示了正在配送的任务，点击后还可以查看详情，展现具体位置和最短的行进路线，期间调用了高德地图api来实现相关功能。同时还有提醒截至时间的功能，在离完成时间不到10分钟的时候，所示时间会变红，提醒配送员尽快配送。最后是个人中心，跑腿员可以看到自己的等级，服务评价，我的收入等相关内容。

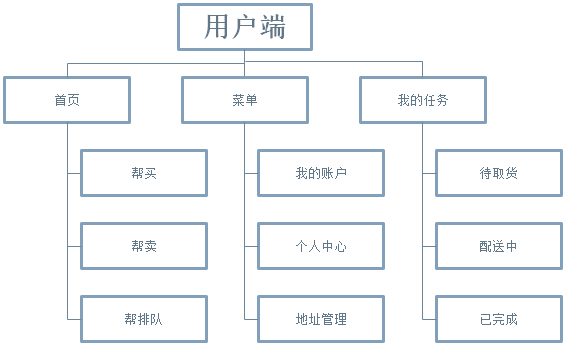


表 3‑5 用户端整体架构

表 3‑6 用户整体架构

用户端最核心的功能是首页的跑腿功能，用户可在首页选择相应的跑腿类型，如帮买，帮卖，帮排队等内容。用户需要填写相关参数才可以发布跑腿订单，如：跑腿类型，跑腿目的地，跑腿内容，预估金额等。用户端侧边栏是菜单，用户可以查看账户内剩余金额，添加地址，查看个人信息。最后是我的任务板块，可以看到用户历史发布的订单情况，并对相关的订单进行评价。

## 系统流程设计

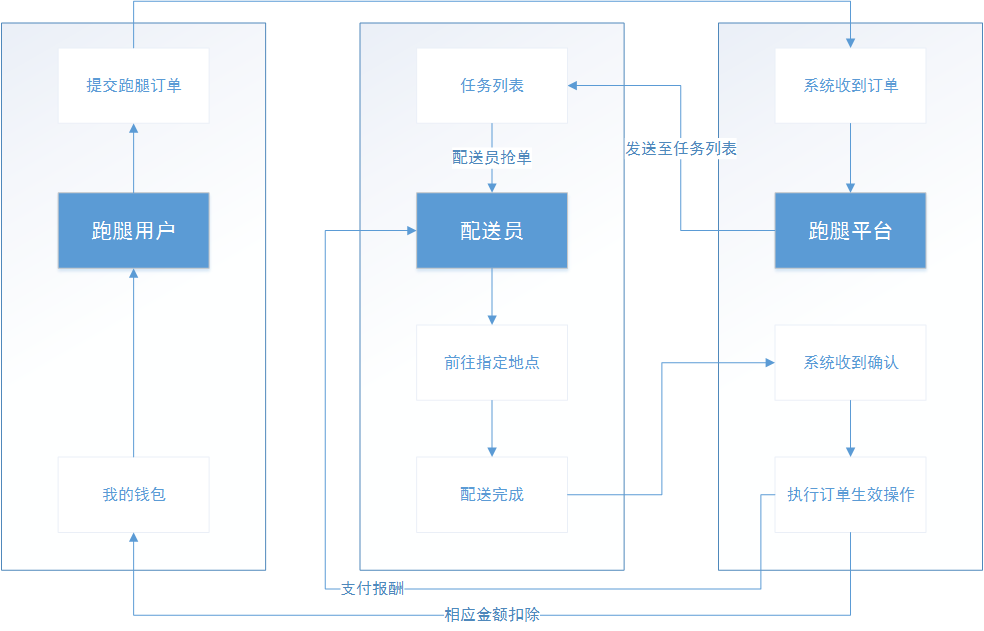


图 3‑7 总体流程图

表 3‑7 总体流程图

系统整体流程如图3-7所示，当用户发起一个跑腿订单时，系统会收到这个订单，并及时转发至配送员的任务列表中，跑腿员看到相应的订单可以选择是否接受，点击接取，便表示跑腿订单被跑腿员接单，，跑腿员便会前往指定地点完成任务，任务有相应的内容和时常限制，在完成任务之后，点击配送完成，便会向后台发送一个完成请求，后台便会处理订单结算操作，从跑腿用户中扣费，并给跑腿员支付相关报酬。

## 系统用例分析

表 3‑8 配送员用例图

对于配送员来说，最主要的功能就是能够抢单，并派送单。在抢单列表中，配送员能够查看订单的详细信息，订单在哪里，以及订单在地图上的具体位置，离我有多远，这一单获取的金额是多少，预计到达时间等等。配送中界面则要显示离目的地还有多远，预计还需要多少时间这些内容。已完成订单中则要显示历史配送过的订单，历史获得的金额。在左侧菜单栏配送员可以在个人中心中查询别人自己的服务评价，服务星级。我的账户中显示了自己的收入情况。

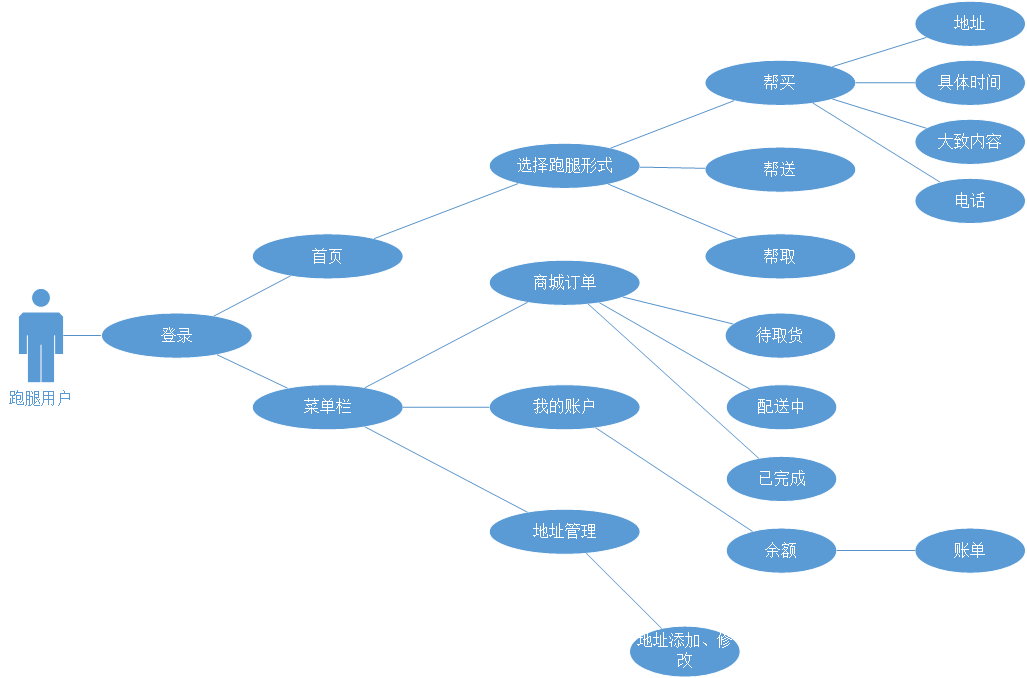
用户关心的是简单易用，用户在登录后进入主页，主页面有一个轮播图介绍我们的主要产品，下面三个就是我们的核心功能，跑腿业务，一共有三种功能，帮买，帮送，帮取。用户点击后需要输入相关信息，如：“我要买一杯咖啡”，“杭州东站”，“武林广场”，“130937987122”，“王女士”，并选择是否要小费，选择完以后下方会出现价格，点击提交后便会出现在配送员的任务列表中。当有配送员抢单后，订单便会自动转入我的订单中，显示待取货，配送完成后便会显示已完成。还有我的钱包，个人中心，用户可以在这里查看相关的个人信息。如图2-9所示。

表 3‑9 跑腿用户用例图

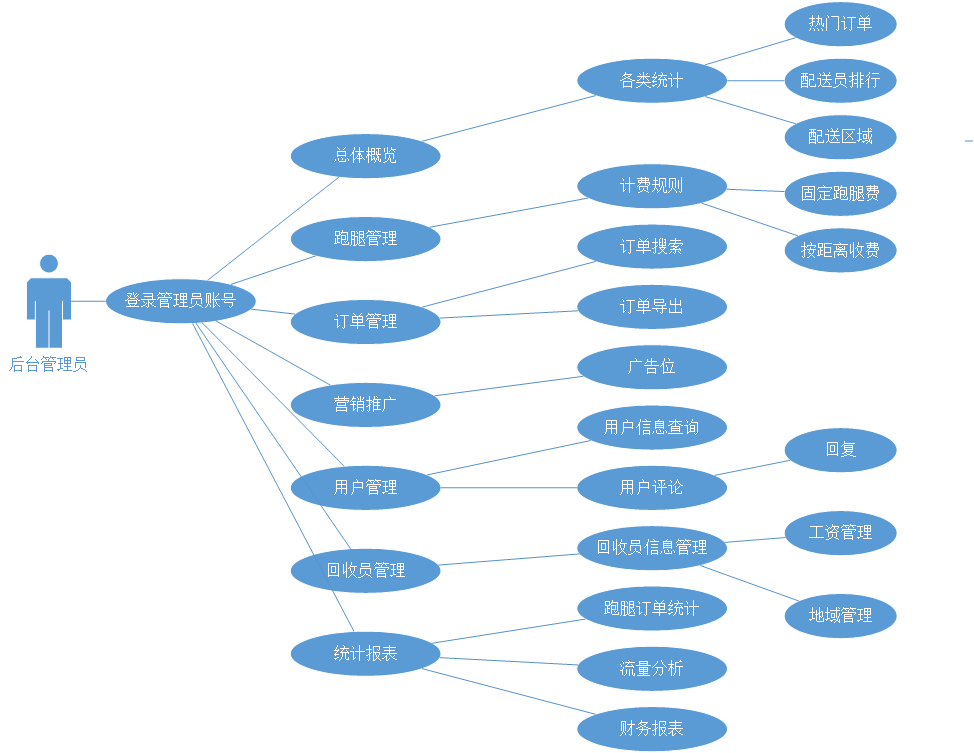
在首页后台管理员可以查看各种信息的统计，在跑腿管理中，可以设置跑腿的计费规则，如按固定跑腿费计费，按距离收费。在订单管理中，不仅可以输入相关条件查找到跑腿的订单信息，还可以导出数据至excel中。在营销中，管理员可以修改轮播图的图片。用户管理中管理员可以查看用户的订单信息，用户的评论，并可以回复用户的相关评论。在回收员管理中，可以进行修改回收员工资，修改回收员信息等操作。最后是统计报表，有跑腿订单统计，流量分析，财务报表等功能。流量分析是通过百度统计api埋点实现的。

表 3‑10 后台管理员用例图

## 系统数据模型设计

表 3‑11 ER图

E-R图也称实体-联系图(Entity Relationship Diagram)，提供了表示实体类型、属性和联系的方法，用来描述现实事件的[概念模型](https://baike.baidu.com/item/%E6%A6%82%E5%BF%B5%E6%A8%A1%E5%9E%8B/3187025)。

如图3-11所示，本系统主要有三个角色，用户，跑腿员，以及后台管理员，核心的操作是对跑腿订单的操作。用户会发布跑腿订单，而跑腿员会对获取到相关的订单。后台管理员可以对跑腿类型一定的计费规则，如按固定跑腿费收取，按距离收取等。用户的主要属性有姓名，电话，地址，邮箱等，与用户地址是一对多的关系，用户地址中包含了详细地址属性，电话属性，收货人名称属性。用户发布跑腿订单，跑腿订单需要设定时间，类型，跑腿费等内容。还要让跑腿员去接单。后台管理员可以设置计费规则表，对跑腿计费进行了相关设定。

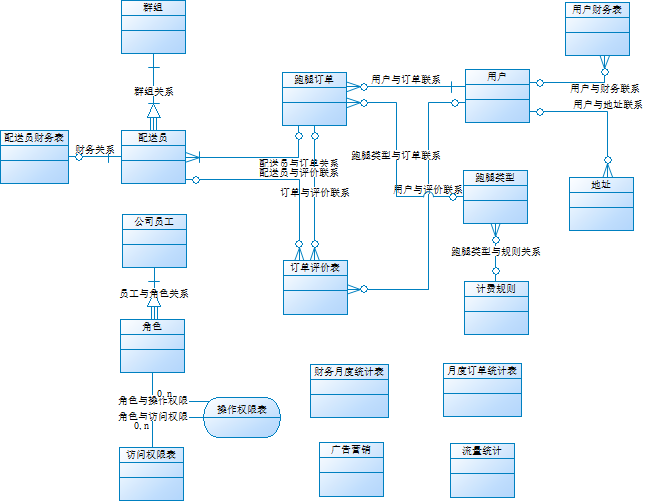


表 3‑12 物理PDM图

## 系统特色

微服务可以在“自己的程序”中运行，并通过“轻量级设备与HTTP型API进行沟通”。关键在于该服务可以在自己的程序中运行。通过这一点我们就可以将服务公开与微服务架构（在现有系统中分布一个API）区分开来。在服务公开中，许多服务都可以被内部独立进程所限制。如果其中任何一个服务需要增加某种功能，那么就必须缩小进程范围。在微服务架构中，只需要在特定的某种服务中增加所需功能，而不影响整体进程。

本次选用了Spring Cloud的微服务架构进行项目的开发，Spring Cloud 集成了zuul、Hysrix、EureKa、Feign等组件。

针对后台管理员需要检测整个系统的运行情况，特别使用了E-CHARTS图表控件，通过调用百度统计api，实现了对网站的PV,UV,IP流量监测。

## 技术选型

|  |  |
| --- | --- |
| 应用层面 | 名称 |
| 核心框架 | Spring Boot 1.5.3 |
| MVC框架 | Spring MVC 4.3.3 |
| 持久层框架 | MyBatis 3.3 |
| 视图框架 | Bootstrap+MUI+Jquery |
| 日志管理 | SLF4J 1.7、Log4j2 |
| 缓存 | Redis 4.0.9 |
| JDK | 1.8 |
| Tomcat | 9.0 |
| 定时器 | Quartz 2.3 |

# 系统详细设计

## 跑腿用户发布模块说明

### 功能描述

用户将跑腿订单发布至平台。

### 流程逻辑

图 4‑1 发布模块流程图

### 功能项

1.订单的发布

2.跑腿费用的计算

3.获得相应距离。

### 设计方法（算法或SQL）

在用户下单的时候，主要面临以下几个操作。1.是要获取用户发送来的表单信息，如起点、终点，电话、手机号等信息。2.是需要在用户发送这些信息之后，需要立刻计算出跑腿费用，跑腿费用通过调用跑腿订单微服务中的计费规则来实现。为了获取用户下单时起点到终点的距离，这里首先需要通过高德api将起始地的地理位置转换成经纬度地址，并将地址储存在address和address1中，之后再将拿到的经纬度地址再次发送给高德api计算出两点之间的距离。如表3-1 所示。

表 4‑1 api调用

|  |
| --- |
| public int ditance(String type, String start, String finish) {  url2="http://restapi.amap.com/v3/geocode/geo?batch=true&key="+key+"&address="+start+"|"+finish+"&city=杭州";  AddressCode addressCode = restTemplate.getForObject(url2, AddressCode.**class**);  String address =addressCode.getGeocodes().get(0).getLocation();  //这里可以用|批量计算地址  String url="http://restapi.amap.com/v3/distance?&origins="+address+""  + "&destination="+address1+"&output=JSON&key="+key;  Geocoder geocoder = restTemplate.getForObject(url, Geocoder.**class**); |

相关sql语句：

insert into business (id, user\_id, collector\_id, type, position\_create, position\_finish, price, start\_time,finish\_time, time, phone, gmt\_create, gmt\_modified, brief)

### 限制条件

用户发布订单必须写齐内容。前端需要进行表单验证，跑腿内容不能为空。

### 尚未解决的问题

暂不能识别非法字符，没有意义的内容。

## 跑腿用户订单支付模块说明

### 功能描述

该模块可完成用户的订单支付。

### 流程逻辑



图 4‑2 订单支付模块

### 功能项

1.订单的支付

### 设计方法（算法或SQL）

对订单支付表(pay)进行状态更改，同时修改用户的余额表(user\_collect)情况。

|  |  |
| --- | --- |
| business\_id | status |
| 8 | 0 |

表 4‑2 订单支付表部分内容

|  |  |
| --- | --- |
| user\_id | balance |
| 9 | 50 |

表 4‑3 余额表部分内容

相关sql：

设修改后支付表数据为”8”,”1”余额表数据为“9””40”

修改支付表状态语句为：update pay set status=1 where business\_id =’8’

修改余额表余额语句为：update user\_collect set balance =40 where user\_id=9

### 限制条件

必须立即支付，需要完成事务回滚。

### 尚未解决的问题

支付失败的情况，并发的情况，没有考虑事务锁的问题。

## 跑腿用户地址管理模块说明

### 功能描述

对用户个人的地址进行添加删除修改操作。

### 流程逻辑

图 4‑3 地址管理模块流程图

### 设计方法（算法或SQL）

通过对地址表的增加删除修改来实现地址的管理

表 4‑4 地址表部分内容

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| user\_id | name | city | province | detail | phone |
| 10001 | 李四 | 杭州 | 浙江省 | 临安区 | 1304390803 |

添加地址语句：

insert address(user\_id,name,city,province,detail,phone)

values(10001,李四,杭州,浙江省,临安区,1304390803)

Update使用动态的方式实现

update address set address =’1’ where user\_id=’10001’;

### 限制条件

必须有一个地址

### 尚未解决的问题

不能设置默认地址。

## 跑腿用户订单评价模块说明

### 功能描述

用户订单的评价

### 流程逻辑

图 4‑4 订单评价流程图

### 功能项

1.订单的评价和打星

### 设计方法（算法或SQL）

当用户完成一笔订单后，都需要对订单进行评价，在这里提供了用户评价的接口以及管理员回复评价的接口。用于对评价表进行增删改查操作。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| id | user\_id | business\_id | collector\_id | evaluate\_text | evaluate |
| 10001 | 11000 | 10122 | 1233 | 很好 | 4.5 |

添加评价语句：insert into evaluate(id, user\_id, business\_id, collector\_id, evaluate\_text, evaluate, gmt\_create, gmt\_modified, repeat) values(10001，11000，10122，1233，很好，4.5)

## 跑腿用户订单管理模块说明

### 功能描述

用户个人的订单查询以及相关操作

### 流程逻辑



图 4‑5 订单管理流程图

### 功能项

1. 订单的查询
2. 历史详情的查看

### 设计方法（算法或SQL）

订单查询使用的是searchpojo api，使用post方法传入searchPojo类，加入@RequestBody注解，并返回businessService方法中的Arraylist<Business>内容。在对订单查询的过程中，使用到了redis缓存，并用@Cacheable进行配置。如表3‑4所示。

表 4‑5 订单查询

|  |
| --- |
| @PostMapping("/searchpojo")@Cacheable(value = "business")//该注解表示可缓存的，会检查是否有这个缓存，如果有的话就直接取值了  **public** ArrayList<Business> searchPojo(@RequestBody BusinessPojo businessPojo){  **return** businessService.searchPojo(businessPojo) ;  } |

可以在chrome调试工具中看到返回的json格式数据。如图3-4所示。

图 3‑4 api返回结果

如表3‑5所示，这是mybatis中的相关查询语句，使用到了动态sql的方法。

表 3‑5 mybatis动态sql

|  |
| --- |
| <selectd="searchPojo"parameterType="com.wzh.entity.BusinessPojo"resultMap="ResultMapWithBLOBs">select \*from business  <where>  <if test="userId != null and userId !=0">  user\_id = #{userId,jdbcType=BIGINT}  </if>  …./此处省略  </where>  </select> |

### 注释设计

无

### 限制条件

只能查询最近一个月的订单

### 尚未解决的问题

暂不支持订单删除

订单查询效率不高

## 跑腿用户我的钱包模块说明

### 功能描述

该模块可查看用户的余额等情况

### 流程逻辑

图 4‑6 我的钱包流程图

### 功能项

1. 金额的充值
2. 历史账单的查询
3. 余额的查询
   * 1. **设计方法（算法或SQL）**

需要做到对用户财务表(userpay)的修改，以及财务表的查询，在这里，我只查询一周的账单情况

表 4‑6 财务表部分内容

|  |  |
| --- | --- |
| user\_id | balance |
| 11000 | 50 |

例如充值50元

则修改后数据为”11000”,”100”

充值语句：update userpay set balance =100 where user\_id=11000

查询一周账单情况

select \* from userpay where WEEK(gmt\_create) = WEEK(now()) order by gmt\_create

### 注释设计

无

### 限制条件

只可以查询一周的账单

### 尚未解决的问题

暂无

## 跑腿员接单模块说明

### 功能描述

该模块可实现跑腿员的接单功能

### 流程逻辑

图 4‑7 接单模块流程图

### 功能项

1. 查看订单
2. 自动派单
3. 接单

### 设计方法（算法或SQL）

首先需要向前台获取跑腿员的id,以及相应订单的id，再由跑腿员微服务封装内容后使用微服务架构中的feign方法实现请求订单微服务中的订单状态变动微服务。来实现跑腿员接单的功能。如表3-19所示。

表 4‑7 接单代码

|  |
| --- |
| @GetMapping("jiedan/{collectorid}/{id}")  public int jiedan(@PathVariable(value="collectorid") Long collectorid,@PathVariable(value="id") Long id){  return peisongService.jiedan(collectorid,id);  }  <select id="needAll" resultMap="ResultMapWithBLOBs">  select  <include refid="Base\_Column\_List" />  from business  where status=0  </select> |

接单需要更改订单表(business)的状态为待取货，并添加配送人。

表 4‑8 订单表部分内容

|  |  |  |
| --- | --- | --- |
| business\_id | status | collector |
| 10111 | 0 | null |

修改后数据为：”10111”,”1”,”22”

语句：update business set status =1,collector=22 where business\_id=10111

配送员的跑腿订单是通过系统自动派单的形式完成了，采用了如下方式：

1）采用迭代的方式，通过订单分配优化算法进行初始的订单分配，然后通过回收员路径优化算法获取各回收员的最佳行驶路线，进而，订单分配优化算法根据回收员路径优化结果调整分配方案。这两个层次不断反复迭代，最终获得比较满意的解

2）二分图解：先对一个人可以完成的订单打包成一个任务，然后用二分图匹配算法（匈牙利算法、KM算法）解决

3）强化学习：引入了离线学习和在线优化相结合的机制，离线学习得到策略模型，在线通过策略迭代，不断寻求更优解。通过不断地改进算法，在耗时下降的同时，算法的优化效果提升 50% 以上。

* + 1. **注释设计**

无

* + 1. **限制条件**

一次最多只能接三个单

* + 1. **尚未解决的问题**

刷新有时候会出现不响应的情况

## 跑腿员任务列表模块说明

### 功能描述

该模块可查询跑腿员所有的任务。

### 流程逻辑

图 4‑8 任务模块流程图

### 功能项

1.订单的支付

### 设计方法（算法或SQL）

对不同状态的订单进行查询。并完成一步操作后可以对任务状态进行更改。

|  |  |
| --- | --- |
| collector\_id | status |
| 22 | 0 |

修改后数据为：”22”,”1”

查询语句：select \* from business where status =0,collector\_id=22

更改状态语句update business set status=1 where collector\_id=22

### 注释设计

无

### 限制条件

必须立即支付

### 尚未解决的问题

支付失败的情况

## 跑腿员历史任务模块说明

### 功能描述

跑腿员可对历史完成的任务进行查询。

### 流程逻辑

****

### 功能项

1.历史订单的查看

### 设计方法（算法或SQL）

选择查询属于该跑腿员的，已完成状态的订单。

|  |  |
| --- | --- |
| collector\_id | status |
| 22 | 1 |

查询语句：select \* from business where status =1,collector\_id=22

### 注释设计

无

### 限制条件

历史订单一次只展示10条

### 尚未解决的问题

暂无

## 管理员订单管理模块说明

### 功能描述

该模块可以完成订单的导出，修改以及删除。

### 流程逻辑



### 功能项

1. 实现管理员对跑腿订单的修改
2. 跑腿订单的删除
3. 跑腿订单的导出

### 设计方法（算法或SQL）

在订单的导出功能中，我使用了将订单导出需要创建HSSFworkbook并将表头，列名，数据传入。如表3-11所示。

表 3‑11 使用poi实现订单导出

|  |
| --- |
| HSSFSheet sheet = wb.createSheet(sheetName);  HSSFRow row = sheet.createRow(0);  HSSFCellStyle style = wb.createCellStyle();  style.setAlignment(HSSFCellStyle.ALIGN\_CENTER); // 创建一个居中格式  for(int i=0;i<values.length;i++){  row = sheet.createRow(i + 1);  for(int j=0;j<values[i].length;j++){ //将内容按顺序赋给对应的列对象  row.createCell(j).setCellValue(values[i][j]); |

在获取到了excel表格后，需要将excel发送，通过response方法返回。

表 3‑12 发送响应流方法

|  |
| --- |
| HSSFWorkbook wb = ExcelUtil.*getHSSFWorkbook*(sheetName, title, content, **null**);  **this**.setResponseHeader(response, fileName);  OutputStream os =response.getOutputStream();  wb.write(os);  os.flush();  os.close(); |

在对订单进行修改时，采用动态sql的方式。

|  |
| --- |
| <update id="updateByPrimaryKeySelective" parameterType="com.wzh.model.Business">  update business  <set>  <if test="collectorId != null">  collector\_id = #{collectorId,jdbcType=BIGINT},  </if>  <if test="type != null">  type = #{type,jdbcType=VARCHAR},  </if> ../  </set>  where id = #{id,jdbcType=BIGINT}  </update> |

### 注释设计

无

### 限制条件

暂无

### 尚未解决的问题

暂无

## 管理员用户管理说明

### 功能描述

该模块可完成管理员对用户的查询，以及黑名单操作。

### 流程逻辑



### 功能项

1. 用户添加黑名单
2. 用户名单导出excel

### 设计方法（算法或SQL）

相关sql，动态sql更改相关信息。

|  |
| --- |
| <update id="updateByPrimaryKeySelective" parameterType="com.wzh.model.User">  update user  <set>  <if test="userName != null">  user\_name = #{userName,jdbcType=VARCHAR},  </if>  <if test="password != null">  password = #{password,jdbcType=VARCHAR},  </if>  <if test="phone != null">  phone = #{phone,jdbcType=VARCHAR},  </if>  <if test="email != null">  email = #{email,jdbcType=VARCHAR},  </if>  </set>  where id = #{id,jdbcType=BIGINT}  </update> |

同时还有用户的黑名单情况，需要把用户的状态置为1表示不可用。

|  |  |
| --- | --- |
| user\_id | status |
| 100001 | 0 |

修改后数据为：”100001”,”1”

拉入黑名单语句为：update user set status =1 where user\_id =100001

### 注释设计

无

### 限制条件

必须立即支付

### 尚未解决的问题

暂无

## 管理员跑腿员管理说明

### 功能描述

该模块可完成管理员对跑腿员信息的查询，修改，黑名单操作。

### 流程逻辑



### 功能项

1. 实现对跑腿员的添加
2. 跑腿员的删除
3. 群组的管理

### 设计方法（算法或SQL）

|  |
| --- |
| <sql id="Base\_Column\_List">  id, position, number, gmt\_create, gmt\_modified  </sql>  <select id="selectByPrimaryKey" parameterType="java.lang.Long" resultMap="BaseResultMap">  select  <include refid="Base\_Column\_List" />  from group  where id = #{id,jdbcType=BIGINT}  </select>  <delete id="deleteByPrimaryKey" parameterType="java.lang.Long">  delete from group  where id = #{id,jdbcType=BIGINT}  </delete>  <insert id="insert" parameterType="com.wzh.model.Group">  insert into group (id, position, number,  gmt\_create, gmt\_modified)  values (#{id,jdbcType=BIGINT}, #{position,jdbcType=VARCHAR}, #{number,jdbcType=INTEGER},  #{gmtCreate,jdbcType=TIMESTAMP}, #{gmtModified,jdbcType=TIMESTAMP})  </insert> |

动态sql插入跑腿员信息，使用<trim>标签实现动态。

|  |
| --- |
| <insert id="insertSelective" parameterType="com.wzh.model.Collector">  insert into collector  <trim prefix="(" suffix=")" suffixOverrides=",">  <if test="id != null">  id,  </if>  </trim>  <trim prefix="values (" suffix=")" suffixOverrides=",">  <if test="id != null">  #{id,jdbcType=INTEGER},  </if>  </trim>  </insert> |

### 注释设计

无

### 限制条件

暂无

### 尚未解决的问题

暂无

## 管理员营销推广模块说明

### 功能描述

该模块可实现广告的投放。

### 流程逻辑

### 功能项

1.订单的支付

### 设计方法（算法或SQL）

首先需要上传图片，并将图片链接等信息插入广告表中。如表3-36所示，后台通过BufferedOutputstrem接收文件，并放在项目的img文件夹中，前台form表单提交时需声明multipart/form-data。

表 3‑37 上传图片

|  |
| --- |
| BufferedOutputStream out = **new** BufferedOutputStream(  **New** FileOutputStream(**new** File("src/main/resources/static/img/"+file.getOriginalFilename())));  **try** {  String number =request.getParameter("number");  **int** id =Integer.*parseInt*(number);  String brief =request.getParameter("Brief");  String picturePath =file.getOriginalFilename();  Pic pic = **new** Pic();  pic.setId(id);  ….  picMapper.updateByPrimaryKey(pic);  out.write(file.getBytes());  out.flush();  out.close(); |

相关sql：

|  |
| --- |
| <insert id="insert" parameterType="com.wzh.model.Pic"> //用于插入  insert into pic (id, picturepath, brief )  values (#{id,jdbcType=INTEGER}, #{picturepath,jdbcType=VARCHAR}, #{brief,jdbcType=VARCHAR} )  </insert>  <sql id="Base\_Column\_List">  id, picturepath, brief  </sql>  <select id="selectByPrimaryKey" parameterType="java.lang.Integer" resultMap="BaseResultMap">  select //用于查询  <include refid="Base\_Column\_List" />  from pic  where id = #{id,jdbcType=INTEGER}  </select>  <delete id="deleteByPrimaryKey" parameterType="java.lang.Integer">  delete from pic //用于删除  where id = #{id,jdbcType=INTEGER}  </delete> |

### 注释设计

无

### 限制条件

必须立即支付

### 尚未解决的问题

支付失败的情况

## 管理员计费规则模块说明

### 功能描述

该模块可实现管理员对跑腿计费规则的设定

### 流程逻辑



### 功能项

1.订单的支付

### 设计方法（算法或SQL）

首先提供了用于后台管理员操作计费规则的接口，如添加距离段跑腿费、添加固定跑腿费。之后用户微服务调用获取计费规则的接口，在下单过程中计算价格。Api如表3‑13所示。

表 3‑13 计费规则api

|  |
| --- |
| public int insert(@RequestBody ArrayList<Moneyrule> moneylist){  return moneyRuleService.insert(moneylist)  } @GetMapping("/searchbytype/{type}/{positioncreate}/{positionfinal}")  Public String searchByType(@PathVariable(value="userid") String type,@PathVariable(value="positioncreate") double distance){  return moneyRuleService.searchByType(type,positioncreate,positionfinal); |

传入跑腿类型，距离到计费规则表中进行查询匹配的一条数据，最后返回跑腿费用。

Xml文件如表3-14所示。

表 3‑14 moneyrule.xml

|  |
| --- |
| <select id="selectByType" resultMap="BaseResultMap">  select  <include refid="Base\_Column\_List" />  from moneyrule  where type = #{type,jdbcType=BIGINT}  and start < #{distance}  and finish >#{distance}  </select> |

插入相关计费规则相关mapper映射，这里采用的是批量插入的形式，使用foreach标签。因为是按距离收费，所以不同距离的收费价格也会不同。

|  |
| --- |
| <insert id="insertlist" parameterType="com.wzh.model.Moneyrule">  insert into moneyrule (id, start, finish, money, type)  <foreach collection="list" index="index" item="item" open="(" separator="," close=")">  values (#{id,jdbcType=BIGINT}, #{start,jdbcType=INTEGER}, #{finish,jdbcType=INTEGER},  #{money,jdbcType=INTEGER}, #{type,jdbcType=VARCHAR})  </foreach>  </insert> |

### 注释设计

无

### 限制条件

必须立即支付

### 尚未解决的问题

支付失败的情况

## 管理员统计模块说明

### 功能描述

该模块可完成用户的订单支付。

### 流程逻辑



### 功能项

1. 财务统计
2. 跑腿订单统计
3. 配送员统计
4. 流量统计

### 设计方法（算法或SQL）

我使用了百度统计API来实现对整个系统的访问量监测，首先需要在每个页面中加入3‑15代码。用于监测每个页面的情况。Src后面的值是百度api给定的一串token值。

表 3‑15 页面监听

|  |
| --- |
| var hm = document.createElement("script");  hm.src = "https://hm.baidu.com/hm.js?609793ea3f86423729f46a66512b1a76";  var s = document.getElementsByTagName("script")[0];  s.parentNode.insertBefore(hm, s); |

之后编写了一个工具类，用于访问百度统计api，来获取json格式的数据，这里的难点是百度的api是https的，所以我需要用https的方式对百度api进行访问。首先需要建立一个SSL上下文，并且是单例模式的，之后创建URLconnection连接，最后通过DataOutputStream输出流将数据获得。如表3-16所示。

表 3‑16 HTTPS 连接

|  |
| --- |
| SSLContext sc = SSLContext.getInstance("SSL");  sc.init(null, new TrustManager[] { new TrustAnyTrustManager() },  new java.security.SecureRandom());  URL console = new URL(url);  HttpsURLConnection conn = (HttpsURLConnection) console.openConnection();  conn.setSSLSocketFactory(sc.getSocketFactory());  conn.setHostnameVerifier(new TrustAnyHostnameVerifier());  conn.setDoOutput(true);  conn.connect();  DataOutputStream out = new DataOutputStream(conn.getOutputStream());  out.write(content.getBytes(charset));  // 刷新、关闭  out.flush(); out.close();  InputStream is = conn.getInputStream();  if (is != null) {  ByteArrayOutputStream outStream = new ByteArrayOutputStream();  byte[] buffer = new byte[1024];  is.close();  return outStream.toByteArray(); |

之后调用该工具类，输入token、用户名、密码等信息，获取相应的json字符串。如3-17所示。

表 3‑17 调用方法

|  |
| --- |
| JSONObject header = **new** JSONObject();  header.put("username", "18057936630");//用户名  header.put("password", "143565");//用户密码  header.put("token", "67576");//申请到的token  header.put("account\_type", "1");  String urlStr = "https://api.baidu.com/json/tongji/v1/ReportService/getData";  **byte**[] res = HttpsUtil.*post*(urlStr, params.toString(), charset);  s = **new** String(res); |

最后再将获得的json数据插入到数据库中。Json信息如3-18所示。

表 3‑18 json字符串

|  |
| --- |
| "header":{"desc":"success", "failures":[],  "body":{"data":["result":{"items":[ [[ 17694,8811,8450],  "timeSpan":["2018/05/27 - 2018/05/29"],  "fields":["simple\_date\_title","pv\_count","visitor\_count", "ip\_count"]  ] |

相关sql：分别为按年查找，按月查找，倒序查找等等。金额统计的时候采用sum函数进行统计，并用group方式按月份排序。一次最多显示六条。

|  |
| --- |
| <select id="searchMoney" resultMap="MoneyPojo">  SELECT SUM(income),SUM(expend),MONTH(gmt\_create) FROM history  GROUP BY MONTH(gmt\_create) ORDER BY MONTH(gmt\_create) DESC LIMIT 12  </select>  <select id="searchByYear" resultMap="HistoryPojo">  SELECT SUM(business),SUM(success),SUM(fail),YEAR(gmt\_create) FROM history  GROUP BY YEAR(gmt\_create) ORDER BY YEAR(gmt\_create) LIMIT 5  </select>  <select id="searchByDesc" resultMap="BaseResultMap">  select  <include refid="Base\_Column\_List" />  from history  order by gmt\_create DESC LIMIT 6  </select> |

### 注释设计

统计还不够全面，没有很好的大数据统计。

### 限制条件

一次显示的统计数据不会过多。

### 尚未解决的问题

统计还不够全面，没有很好的大数据统计。

# 系统实现

本课题采用微服务架构开发，主要由Eureka微服务发现组件，zuul微服务网关，跑腿订单微服务，用户微服务，跑腿员微服务五大块构成。

## 微服务注册发现组件—EurekaService

Eureka是Netflix开源的服务发现组件，本身是一个基于REST的服务。他在微服务进行交互之前，都需要先在eureka上进行注册。Eureka的主要作用是用于追踪各个微服务组件的运行情况，在微服务在eureka上注册之后，会每90秒发送一次心跳，一旦eureka检测不到微服务的心跳，便会注销该实例。

在使用eureka之前，需要在application.yml中进行配置，写明端口号，eureka服务url。register-with-eureka表示是否将自己注册到Eureka Server，由于当前应用本身就是Eureka Server，所以默认为false，fetch-registry 表示是否注册多节点的Eureka Server，因为目前这是一个单点的Eureka Server，故设置为false。如果有建立Eureka集群的需要，则要设置为true，并指定节点。如表3-1所示

表 3‑1 eureka配置文件

|  |
| --- |
| server:  port: 8761  eureka:  client:  register-with-eureka: **false**  fetch-registry: **false**  service-url:  defaultZone: http://localhost:8761/eureka/ |

通过启动各个微服务以及Eureka，输入localhost：8761，可以在Instances currently registered with Eureaka中可以清楚的看到各个微服务已经被注册上了。被注册的微服务都需要在配置文件中指定defaultZone的url，才能被注册进eureka中。如图3-1所示。

## 微服务网关—ZuulProxy

表 6‑1 Eureka 运行截图

表 6‑2 Eureka 运行截图

Zuul是Netflix开源的微服务网关，它可以和Eureka、Ribbon、Hystrix等组件配合使用。Zuul默认使用HTTP Client，拥有身份认证，审查与监控等功能。

使用前需要对zuul启动类添加@EnableZuulProxy注解，spring boot便会自动配置Zuul的相关内容，同时整合了Ribbon和Hystrix。

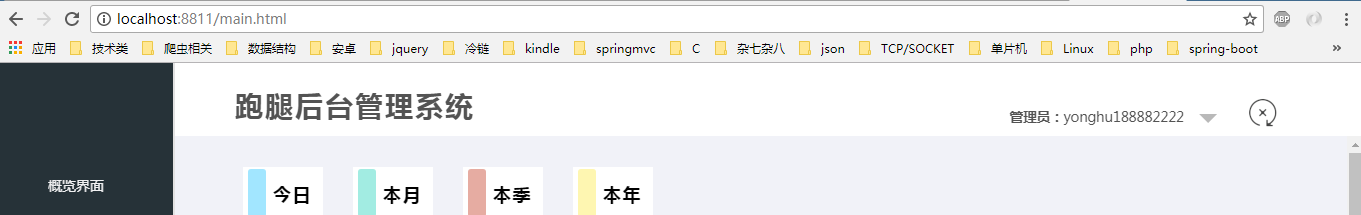
在启动zuul之后，可以看到项目已经通过了api网关，被转发到localhost:8040/micro-provider-Business 下面。如图3-3所示。

图 3‑2 使用zuul网关前

图 3‑3 使用zuul网关后

## 微服务的REST调用和熔断机制

### 使用Feign实现声明式REST调用

通过Feign可以帮助我们更加便捷的调用HTTP API，更好的实现微服务之间的

通信和交互。

通过@FeignClient注解规定所访问的微服务组件，fallback用于声明回退，通过接口的形式访问 micro-provider-business的business/request api，如表3-2所示。

表 3‑2 使用Feign

|  |
| --- |
| @FeignClient(name="micro-provider-business",fallback=FeignClientFallback.**class**)  **public** **interface** BusinessFeign {  @RequestMapping(value="business/request", method=RequestMethod.***GET***)  **public** ArrayList<Business> waitAll(); |

### 整合Feign和Hystrix

Hystrix是由Netflix开源的一个延迟容错库，用于隔离访问远程系统，防止级联失败，从而提升系统的可用性与容错性。

我将Feign和Hystrix整合起来，从而实现Feign的回退。通过继承Feign Callback类，并重写其方法，实现当微服务架构中出现雪崩效应时的处理，如表3-3所示，我直接返回为空。

表 3‑3 fallback回滚操作

|  |
| --- |
| @Component  class FeignClientFallback implements BusinessFeign{  @Override  public ArrayList<Business> waitAll() {  // TODO Auto-generated method stub  return null;  } |

## 跑腿订单微服务

在本系统中，最重要的便是对于跑腿订单的操作，需要实现对订单的增删改查，订单状态的更改，订单的支付，统计，等待。因此我将他独立成一个微服务。跑腿员微服务、用户微服务在需要进行订单操作时，便会调用跑腿订单微服务的相关api。

### 订单的查询

订单查询使用的是searchpojo api，使用post方法传入searchPojo类，加入@RequestBody注解，并返回businessService方法中的Arraylist<Business>内容。在对订单查询的过程中，使用到了redis缓存，并用@Cacheable进行配置。如表3‑4所示。

表 3‑4 订单查询

|  |
| --- |
| @PostMapping("/searchpojo")@Cacheable(value = "business")//该注解表示可缓存的，会检查是否有这个缓存，如果有的话就直接取值了  **public** ArrayList<Business> searchPojo(@RequestBody BusinessPojo businessPojo){  **return** businessService.searchPojo(businessPojo) ;  } |

可以在chrome调试工具中看到返回的json格式数据。如图3-4所示。

图 3‑4 api返回结果

如表3‑5所示，这是mybatis中的相关查询语句，使用到了动态sql的方法。

表 3‑5 mybatis动态sql

|  |
| --- |
| <selectd="searchPojo"parameterType="com.wzh.entity.BusinessPojo"resultMap="ResultMapWithBLOBs">select \*from business  <where>  <if test="userId != null and userId !=0">  user\_id = #{userId,jdbcType=BIGINT}  </if>  …./此处省略  </where>  </select> |

### 订单状态的变动

这里是进行接单操作的api接口，通过传入id值，订单状态，通过mybaits的update方法进行更新。如表3-6所示。

表 3‑6 订单状态变动

|  |
| --- |
| @GetMapping("/jiedan/{collectorid}/{id}")  @CacheEvict(value="business", allEntries=**true**) //删除缓存  **public** **int** jiedan(@PathVariable(value="collectorid") Long collectorid,@PathVariable(value="id") Long id){  Business business = **new** Business();  business.setCollectorId(collectorid);  business.setId(id);  business.setStatus(1);  **return** businessService.jiedan(business);  } |

### 订单的评价

当用户完成一笔订单后，都需要对订单进行评价，在这里提供了用户评价的接口以及管理员回复评价的接口。用于对评价表进行增删改查操作。如表3-7所示。

表 3‑7 订单评价的增删改查

|  |
| --- |
| @GetMapping("/repeat/{id}/{repeat}")  **public** **int** repeat(@PathVariable Long id,@PathVariable String repeat){  **return** evaluateService.repeat(id,repeat);  }  @GetMapping("/repeat/{id}/{content}")  **public** **int** repead(@PathVariable(value="id") Long id,@PathVariable(value="content") String content) {  Evaluate evaluate =**new** Evaluate();  evaluate.setId(id);  evaluate.setRepeat(content);  **return** evaluateService.repeat(evaluate); |

### 订单的添加

用户端通过Feign调用这个api来实现下单的操作。当订单进行添加操作时，会通过@CacheEvict来清除缓存。达到缓存更新的目的。如表3-8所示。

表 3‑8 订单添加

|  |
| --- |
| @PostMapping("/insert")  @CacheEvict(value="business", allEntries=**true**)  **public** **int** insert(@RequestBody Business business){  System.***out***.println(business.getStartTime());  **return** businessService.insert(business);  }  <insert id="insertSelective" parameterType="com.wzh.model.Business">  insert into business  <trim prefix="(" suffix=")" suffixOverrides=",">  <if test="id != null">  id,  </if>  ….. |

### 订单的统计

在对订单的统计中，在系统开始时，便会打开一个定时器，定时器负责每一个小时对当前的情况进行统计，定时器使用了quartz定时器，通过jobdetail实例化job\_dingshi之后再使用Crontrigger 配置定时时间，也就是(0 0 0/0 \*\*?),这段cron表达式的含义是每小时执行一次job。如表3-9所示。

表 3‑9 定时器

|  |
| --- |
| **public** **void** scheduleJob(Scheduler scheduler) **throws** SchedulerException {  JobDetail jobDetail = JobBuilder.*newJob*().withIdentity("job\_dingshi", "jGroup1").build();  JobDataMap jobDataMap = jobDetail.getJobDataMap();  CronScheduleBuilder cronScheduleBuilder = CronScheduleBuilder.*cronSchedule*("0 0 0/1 \* \* ? ");  CronTrigger cronTrigger = TriggerBuilder.*newTrigger*().withIdentity("trigger\_1", "tGroup1")  .withSchedule(cronScheduleBuilder).build();  scheduler.scheduleJob(jobDetail, cronTrigger); |

这里通过sum函数对history表的金额支出进行了统计，并用group by方法分组，最后得到一年，一月，一周的数据。

表 3‑10 订单统计

|  |
| --- |
| <resultMap type=*"com.wzh.entity.MoneyPojo"* id=*"MoneyPojo"*>  <result column=*"SUM(income)"* jdbcType=*"INTEGER"* property=*"income"* />  <result column=*"SUM(expend)"* jdbcType=*"INTEGER"* property=*"expend"* /> <result column=*"MONTH(gmt\_create)"* jdbcType=*"TIMESTAMP"* property=*"time"* />  </resultMap>  <select id=*"searchByYear"* resultMap=*"HistoryPojo"*>  SELECT SUM(business),SUM(success),SUM(fail),YEAR(gmt\_create) FROM |

### 订单的导出

在订单的导出功能中，我使用了将订单导出需要创建HSSFworkbook并将表头，列名，数据传入。如表3-11所示。

表 3‑11 使用poi实现订单导出

|  |
| --- |
| HSSFSheet sheet = wb.createSheet(sheetName);  HSSFRow row = sheet.createRow(0);  HSSFCellStyle style = wb.createCellStyle();  style.setAlignment(HSSFCellStyle.***ALIGN\_CENTER***); // 创建一个居中格式  **for**(**int** i=0;i<values.length;i++){  row = sheet.createRow(i + 1);  **for**(**int** j=0;j<values[i].length;j++){ //将内容按顺序赋给对应的列对象  row.createCell(j).setCellValue(values[i][j]); |

在获取到了excel表格后，需要将excel发送，通过response方法返回。

表 3‑12 发送响应流方法

|  |
| --- |
| HSSFWorkbook wb = ExcelUtil.*getHSSFWorkbook*(sheetName, title, content, **null**);  **this**.setResponseHeader(response, fileName);  OutputStream os =response.getOutputStream();  wb.write(os);  os.flush();  os.close(); |

### 订单的计费规则

首先提供了用于后台管理员操作计费规则的接口，如添加距离段跑腿费、添加固定跑腿费。之后用户微服务调用获取计费规则的接口，在下单过程中计算价格。Api如表3‑13所示。

表 3‑13 计费规则api

|  |
| --- |
| **public** **int** insert(@RequestBody ArrayList<Moneyrule> moneylist){  **return** moneyRuleService.insert(moneylist)  } @GetMapping("/searchbytype/{type}/{positioncreate}/{positionfinal}")  **public** String searchByType(@PathVariable(value="userid") String type,@PathVariable(value="positioncreate") double distance){  **return** moneyRuleService.searchByType(type,positioncreate,positionfinal); |

传入跑腿类型，距离到计费规则表中进行查询匹配的一条数据，最后返回跑腿费用。

Xml文件如表3-14所示。

表 3‑14 moneyrule.xml

|  |
| --- |
| <select id="selectByType" resultMap="BaseResultMap">  select  <include refid="Base\_Column\_List" />  from moneyrule  where type = #{type,jdbcType=BIGINT}  and start < #{distance}  and finish >#{distance}  </select> |

### 流量统计

我使用了百度统计API来实现对整个系统的访问量监测，首先需要在每个页面中加入3‑15代码。用于监测每个页面的情况。Src后面的值是百度api给定的一串token值。

表 3‑15 页面监听

|  |
| --- |
| var hm = document.createElement("script");  hm.src = "https://hm.baidu.com/hm.js?609793ea3f86423729f46a66512b1a76";  var s = document.getElementsByTagName("script")[0];  s.parentNode.insertBefore(hm, s); |

之后编写了一个工具类，用于访问百度统计api，来获取json格式的数据，这里的难点是百度的api是https的，所以我需要用https的方式对百度api进行访问。首先需要建立一个SSL上下文，并且是单例模式的，之后创建URLconnection连接，最后通过DataOutputStream输出流将数据获得。如表3-16所示。

表 3‑16 HTTPS 连接

|  |
| --- |
| SSLContext sc = SSLContext.getInstance("SSL");  sc.init(null, new TrustManager[] { new TrustAnyTrustManager() },  new java.security.SecureRandom());  URL console = new URL(url);  HttpsURLConnection conn = (HttpsURLConnection) console.openConnection();  conn.setSSLSocketFactory(sc.getSocketFactory());  conn.setHostnameVerifier(new TrustAnyHostnameVerifier());  conn.setDoOutput(true);  conn.connect();  DataOutputStream out = new DataOutputStream(conn.getOutputStream());  out.write(content.getBytes(charset));  // 刷新、关闭  out.flush(); out.close();  InputStream is = conn.getInputStream();  if (is != null) {  ByteArrayOutputStream outStream = new ByteArrayOutputStream();  byte[] buffer = new byte[1024];  is.close();  return outStream.toByteArray(); |

之后调用该工具类，输入token、用户名、密码等信息，获取相应的json字符串。如3-17所示。

表 3‑17 调用方法

|  |
| --- |
| JSONObject header = **new** JSONObject();  header.put("username", "18057936630");//用户名  header.put("password", "143565");//用户密码  header.put("token", "67576");//申请到的token  header.put("account\_type", "1");  String urlStr = "https://api.baidu.com/json/tongji/v1/ReportService/getData";  **byte**[] res = HttpsUtil.*post*(urlStr, params.toString(), charset);  s = **new** String(res); |

最后再将获得的json数据插入到数据库中。Json信息如3-18所示。

表 3‑18 json字符串

|  |
| --- |
| "header":{"desc":"success", "failures":[],  "body":{"data":["result":{"items":[ [[ 17694,8811,8450],  "timeSpan":["2018/05/27 - 2018/05/29"],  "fields":["simple\_date\_title","pv\_count","visitor\_count", "ip\_count"]  ] |

## 跑腿员微服务

跑腿员最主要就是要抢单，查看自己的任务，查看自己的个人信息。

### 跑腿员接单

首先需要向前台获取跑腿员的id,以及相应订单的id，再由跑腿员微服务封装内容后使用微服务架构中的feign方法实现请求订单微服务中的订单状态变动微服务。来实现跑腿员接单的功能。如表3-19所示。

表 3‑19 接单

|  |
| --- |
| @GetMapping("jiedan/{collectorid}/{id}")  **public** **int** jiedan(@PathVariable(value="collectorid") Long collectorid,@PathVariable(value="id") Long id){  **return** peisongService.jiedan(collectorid,id);  }  @FeignClient(name="micro-provider-business",fallback=FeignClientFallback.**class**)  **public** **interface** BusinessFeign {  @RequestMapping(value="business/jiedan/{collectorid}/{id}", method=RequestMethod.***GET***)  **public** **int** jiedan(@PathVariable(value="collectorid") Long collectorid,@PathVariable(value="id") Long id); } |

### 跑腿员任务

这里跑腿员任务调用订单微服务，查询所有订单中属于该跑腿员的订单，需要传入跑腿员ID，订单状态两个参数，便可以查询。如表3-20所示。

表 3‑20 跑腿员任务

|  |
| --- |
| @GetMapping("searchbystatus/{collectorid}/{status}")  **public** ArrayList<Business> searchbyStatus(@PathVariable(value="collectorid") Long collectorid,@PathVariable(value="status") Long status){  **return** peisongService.searchByStatus(collectorid, status);  } |

这里Feign调用了订单微服务中的search接口，并返回ArrayList<Business>数组。如表3-21所示。

表 3‑21 Feign调用

|  |
| --- |
| @RequestMapping(value="business/search/{collectorid}/{id}", method=RequestMethod.GET)  public ArrayList<Business> searchByStatus(@PathVariable(value="collectorid") Long collectorid,@PathVariable(value="id") Long id);  @PostMapping(value="business/search/searchbySelect")  public ArrayList<Business> searchByCollector(@RequestBody Business business);  } |

### 跑腿员个人信息

通过search/id找到员工的相关信息，还可以用update接口对跑腿员进行更新。

如表3-22所示。

表 3‑22 跑腿员个人信息

|  |
| --- |
| @GetMapping("search/{id}")  public Collector searchById(@PathVariable Integer id ){  return collectorService.search(id);  }  @PostMapping("update")  public int update(@RequestBody Collector collector){    return collectorService.update(collector);  } |

## 用户微服务

### 用户下单

在用户下单的时候，主要面临以下几个操作。1.是要获取用户发送来的表单信息，如起点、终点，电话、手机号等信息。2.是需要在用户发送这些信息之后，需要立刻计算出跑腿费用，跑腿费用通过调用跑腿订单微服务中的计费规则来实现。最后再统一插入表中。如表3-23所示。

表 3‑23 用户下单

|  |
| --- |
| **Public** **int** insert(@PathVariable(value="userid") Long userid, @PathVariable(value="positioncreate")String positioncreate,@PathVariable(value="positionfinish") ){  Business business = **new** Business();  business.setName(name);  business.setUserId(userid); |

为了获取用户下单时起点到终点的距离，这里首先需要通过高德api将起始地的地理位置转换成经纬度地址，并将地址储存在address和address1中，之后再将拿到的经纬度地址再次发送给高德api计算出两点之间的距离。如表3-24 所示。

表 3‑24 高德api调用

|  |
| --- |
| public int ditance(String type, String start, String finish) {  url2="http://restapi.amap.com/v3/geocode/geo?batch=true&key="+key+"&address="+start+"|"+finish+"&city=杭州";  AddressCode addressCode = restTemplate.getForObject(url2, AddressCode.**class**);  String address =addressCode.getGeocodes().get(0).getLocation();  String address1 =addressCode.getGeocodes().get(1).getLocation();  // System.out.println(addressCode.getGeocodes().get(0).getLocation());  //这里可以用|批量计算地址  String url="http://restapi.amap.com/v3/distance?&origins="+address+""  + "&destination="+address1+"&output=JSON&key="+key;  Geocoder geocoder = restTemplate.getForObject(url, Geocoder.**class**); |

### 用户个人信息

这里通过传入用户的userid来查看用户的个人信息，返回User实体类。这里不使用Feign，直接通过service层调用后台的mapper接口。

表 3‑25 用户个人信息

|  |
| --- |
| @GetMapping("/searchuser/{userid}")  Public User searchuser (@PathVariable Long userid){  return customerService.findById(userid); |

### 用户历史订单

首先需要向用户微服务发送用户id，再由用户微服务封装内容后转发至跑腿订单微服务。如表3-26所示。

表 3‑26 用户历史订单

|  |
| --- |
| @GetMapping("/searchbusiness/{userid}")  public ArrayList<Business> searchBusinessById(@PathVariable Long userid){  return customerFeign.findById(userid);  @FeignClient(name="micro-provider-business",fallback=FeignClientFallback.**class**)  public interface CustomerFeignClient {  @RequestMapping(value="/business/search2/{userid}", method=RequestMethod.***GET***)  public ArrayList<Business> findById(@PathVariable(value="userid") Long userid); |

# 系统实现

## 跑腿员功能实现（HTML5）

### 接单

表 7‑1 配送端主页

表 7‑2 订单详情页

主页是一个任务列表，跑腿员可以在这里直接进行接单操作，任务通过调用订单微服务中request api获取内容，在success回调方法中，首先清空div，之后再通过each方法遍历结果集，最后通过append方法添加到div中。如表3-27这里还通过pullRefresh()方法可以进行下拉刷新。

表 3‑27 动态ajax加载

|  |
| --- |
| mui('#maincontent').pullRefresh().endPullupToRefresh(true);  $.ajax({  type:"get",  url:"http://localhost:8040/micro-provider-business/business/request",  async:true,  dataType: "json",//预期服务器返回的数据类型  data:{}, |

当连接成功时，执行success的回调函数，通过append方法将任务显示在界面上。如表3-28所示。

表 3‑28 执行succes函数

|  |
| --- |
| success: function(result){  $('#maincontent2').empty();  $('#maincontent2').next().remove();  $.each(result, function(index, item){  con +='<div class="mui-card" id="'+item.id+'">'  …  con +='<button type="button" >接单</button></div></div>'  $('#maincontent2').append(con); |

点击后还可以查看详情，详情中的地图通过调用高德api实现，如3‑29所示。

表 3‑29 调用高德地图api关键代码

|  |
| --- |
| var driving = new AMap.Driving({  policy: AMap.DrivingPolicy.LEAST\_TIME,  map: map,  panel: "panel"  });  // 根据起终点名称规划驾车导航路线  driving.search([  {keyword: positionCreate1,city:'杭州'},  {keyword: positionFinish1,city:'杭州'}  ]);  }); |

### 我的任务

表 7‑3 配送中

表 7‑4 待取货

可以进行选项切换，并下拉刷新，下拉刷新时调用后台的searchbystatus接口，获得关于订单的相关信息。如果配送员将订单完成，便可点击配送完成，完成后的订单会切换至已完成订单中。代码中index表示不同的订单状态，1表示待取货的订单，2表示配送中的订单，3表示已完成的订单。如表3-30所示。

表 3‑30 下拉刷新关键代码

|  |
| --- |
| //循环初始化所有下拉刷新，上拉加载。  mui.each(document.querySelectorAll('.mui-slider-group .mui-scroll'), **function**(index, pullRefreshEl) {mui(pullRefreshEl).pullToRefresh({down: {  callback: **function**() {$.ajax({  type:"get",  url:"peisong/searchbystatus/"+collectorid+"/"+index1+"",  success:**function**(result){  **if**(index==0){  $('.mui-slider-group .mui-scroll').first().empty();  $.each(result, **function**(index, item){ ….以下代码省略  $('.mui-slider-group .mui-scroll').first().append(con1);  self.endPullDownToRefresh()； |

### 个人中心

表 7‑5 我的余额

表 7‑6 个人中心

这里是配送员的个人中心，可以对相关功能进行设置。可以查看我的余额。在我的余额中可以查看明细，自己获得的跑腿费。这里调用了跑腿员微服务中的个人信息模块。

## 客户端功能实现(HTML5)

### 首页

表 7‑7 菜单

表 7‑8 首页

首页顶部有一个轮播图，用来推送相关信息，并有帮买帮送帮排队三个功能。

左侧有滑动菜单栏，其中有我的订单，我的账户，我的地址等相关功能。轮播图我使用了mui中的slider组件，根据他的特性写出了轮播图。同时，这里的轮播图会根据后台营销中上传的图片而更改。如表3-31所示。

表 3‑31 动态更改图片

|  |
| --- |
| $.ajax{  <url:getpic>  success:function(result){  $(#pic).eq(0).attr(‘src’,’http://localhost:8040/micro-provider-business/static/img+”item.picpath)  $(#pic).eq(1).attr(‘src’,’http://localhost:8040/micro-provider-business/static/img+”item.picpath)  $(#pic).eq(2).attr(‘src’,’http://localhost:8040/micro-provider-business/static/img+”item.picpath)  $(#pic).eq(3).attr(‘src’,’http://localhost:8040/micro-provider-business/static/img+”item.picpath)  }} |

### 帮买

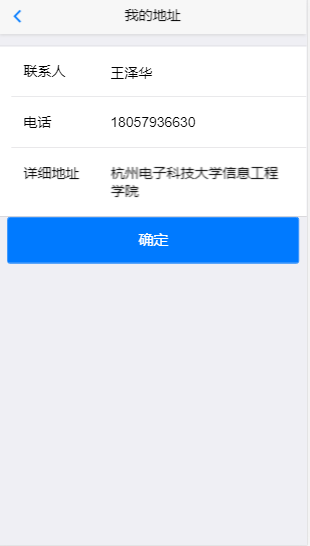


表 7‑9 帮我买

表 7‑10 我的地址

可以选择自己想买的商品，选择相应的店铺，并选择收货地址，还可以选择小费，最后提交。选择地址有输入提示功能，在完善收货人信息中，会跳转至我的地址界面，进行地址选择后便可出现在界面上。输入提示功能由高德地图API提供，调用形式如表3-32所示。在监听到起始地信息以及重量信息后，会将这两个数据发送至用户微服务，后台通过查询起始地之间的距离来确定价格，返回的数据会在合计中显示。如表3-33所示。

表 3‑32 下单代码

|  |
| --- |
| $.ajax({  type:"get",  url:"http://localhost:8040/micro-comsumer-user/paotui/insert/"+userid+"&"+brief+"&"+type+"&"+positionCreate  +"&"+address+"&"+money+"&"+phone+"&"+name,  success: function(result){  if (!mui.os.plus) {  mui.openWindow({  id:'main.html',  url:'main.html'  }) |

首先通过调用高德api获取地址的经纬度信息，geocoder.getLocation是高德地图获取经纬度的方法，positionCreate参数代表我输入的起点。在回调方法中获得地址的经纬度信息。

表 3‑33 高德地图获取地址编码

|  |
| --- |
| AMap.plugin('AMap.Geocoder', function() {  var geocoder = new AMap.Geocoder()  geocoder.getLocation('positionCreate', function(status, result) {  if (status === 'complete' && result.info === 'OK') {  // result中对应详细地理坐标信息  geocoder\_CallBack(result);  }  })  }) |

之后再使用高德地图的计算距离函数，获得距离。最后通过ajax方式将距离和跑腿类型发送至后台，后台通过查询计费表后返回值。

表 3‑34 计算距离

|  |
| --- |
| // 返回 p1 到 p2 间的地面距离，单位：米  var dis = AMap.GeometryUtil.distance(positionCreate, PositionFinal);  $.ajax{  [url:searchbytype/”+type”+/”dis](url:searchbytype/)”  success: function(result){  $(‘#money’).text(result) } |

### 我的地址&我的订单

表 7‑11 地址列表

表 7‑12 我的订单

在我的地址中，可以查看我的地址列表，地址列表存储在后台的用户表中，通过调用用户微服务的addess 接口获取信息。在我的订单中，可以查看待取货、配送中、已完成三种状态的订单。

### 我的评价&我的钱包

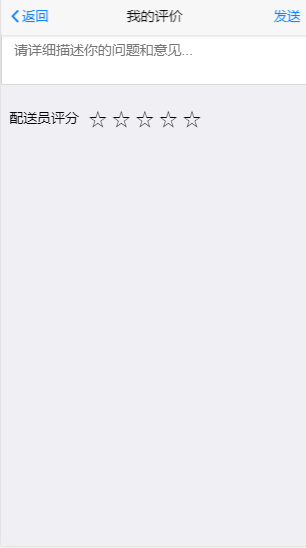


表 7‑13 我的余额

表 7‑14 我的评价

最后是我的评价和我的钱包，在用户完成订单后，可以对订单进行评价，评价的内容会在后台显示。我的钱包中用户可以对其进行充值，还有消费明细等内容。这里调用了用户微服务中的个人信息模块。

## 后台管理系统功能实现

### 概览界面

可以查看系统的实时概况，查看支出，营业额，当前跑腿订单等信息，还有一个订单成交量图表。

表 7‑15 总体概览

通过调用summary/currentapi接口获取当前信息，并通过jquery的text方法修改内容。

表 3‑35 获取实时概况信息

|  |
| --- |
| $.ajax({  type:"get",  url:"/summary/current",  async:true,  data:{},  dataType:"json",  success:function(result){  // $.each(result,function(result,item){  $('#expend').text(result.expend);  $('#income').text(result.income);  $('#business').text(result.business); |

### 计费规则

在跑腿设置中，可以对各个跑腿进行设置，如下图3-8,3-9所示，固定跑腿费可以设置在最大公里数内的固定跑腿金额。在按距离收费中，可以针对距离来设置跑腿金额。后台通过批量插入数据来实现对按距离收费的设置。

表 7‑16 固定跑腿费



表 7‑17 按距离收费

### 订单管理

如表 7‑18所示管理员可以对订单进行多条件查询，如配送员、用户编号、订单状态等等，还有下单的年月日期。



图 3‑21 订单搜索

当管理员在每个输入框输入相关信息，jquery会自动获取到字段的相关信息，并在最后封装成一个json对象，如表3‑36所示，每一个条件都被放入了json对象中，最后通过ajax的post方法发送至后端api接口，[由于后台是通过@reqeustbody来获取实体类的信息，需要将business通过json.stringify](mailto:由于后台是通过@reqeustbody来获取实体类的信息，需要将business通过json.stringify)方法变成一个字符串形式，并声明content-type为application/json。在success的回调方法中，使用jquery的append方法把内容添加至表单中。

表 3‑36 json对象

|  |
| --- |
| var businesspojo ={  "id":id,  "collectorId":id,  "userId":userId,  "time1":time1,  "time2":time2,  "phone":phone,  "type":type,  "status":status  } |

#### 订单导出

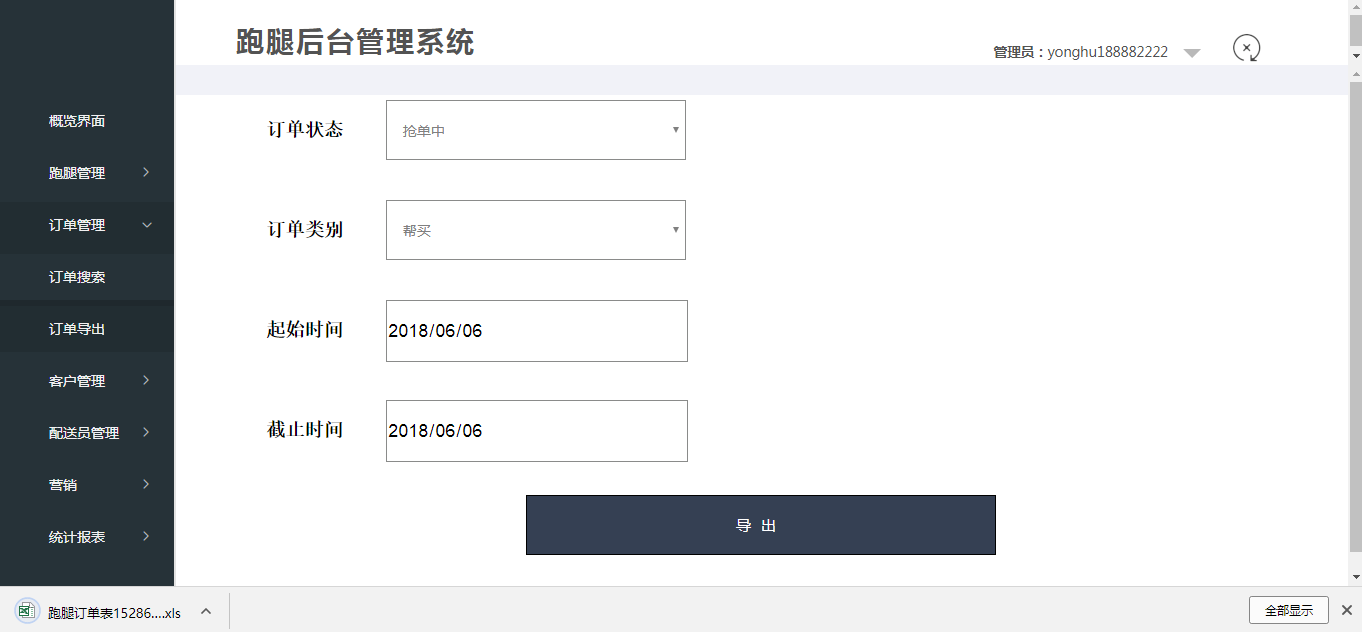
在订单管理中，还有一个订单导出功能，可以选择相应的订单状态和订单类别，然后通过表单提交的形式将信息发送至后台，后台会通过响应流将文件发送至前台。

表 7‑19 订单导出

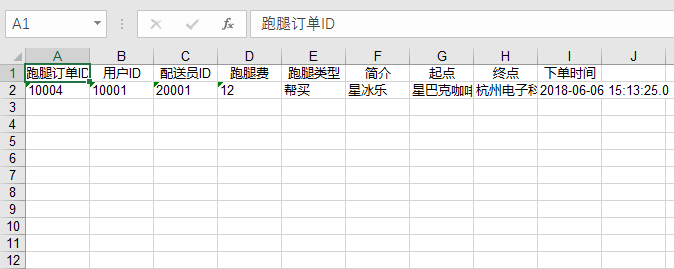
管理员在输入订单状态，订单类别，起止时间之后，便会获得相应的导出excel文件。通过form表单发送信息。

表 7‑20 导出的excel文件

导出的excel文件中，有跑腿订单ID，用户ID，配送员ID等相关信息。

### 客户管理

#### 客户搜索

可以根据用户名和用户ID来进行搜索。

表 7‑21 客户搜索

#### 客户评论

可以查看客户的评论，还可以回复客户的评论，客户便会在我的评论中收到回复。

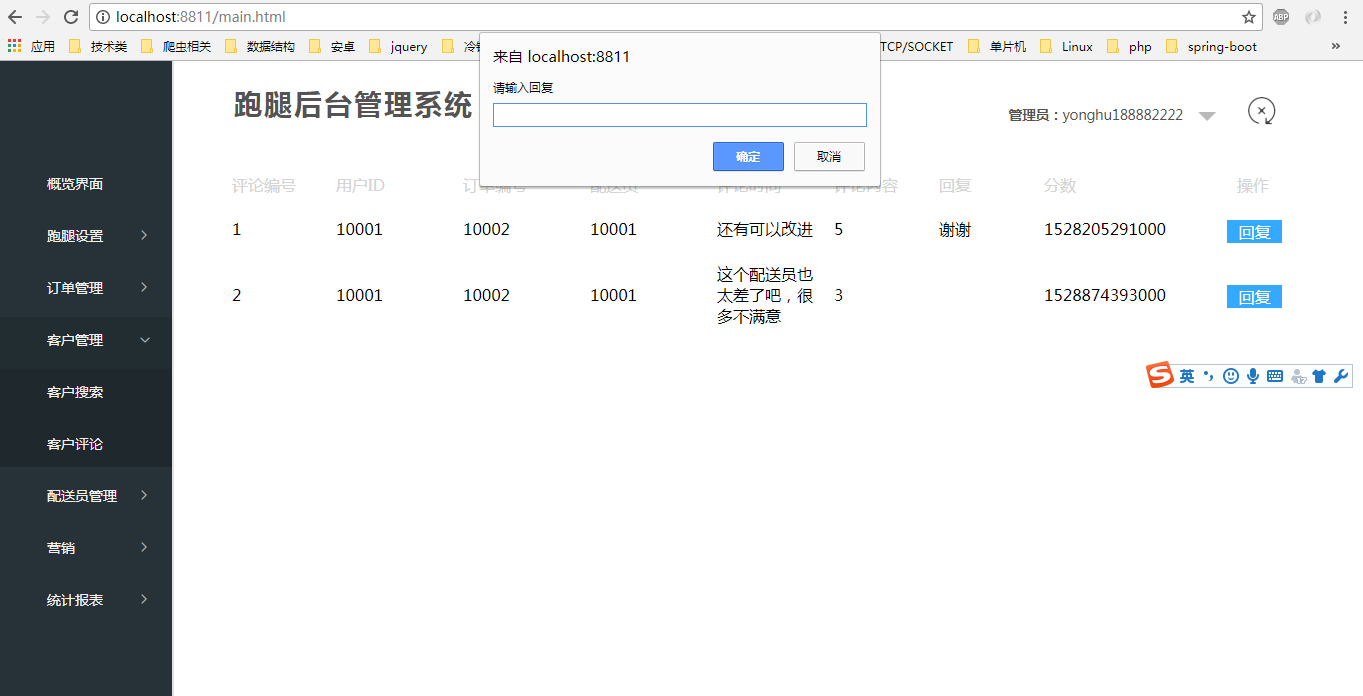


表 7‑22 客户评论

### 配送员管理

可以查看配送员的上下班情况，订单状况，收入等，还可以查看详细信息，

表 7‑23 配送员管理

右上角可以添加配送员。

添加配送员时需要填入姓名、密码、电话、工作地点、邮箱这些必要信息，不能为空。

表 7‑24 添加配送员

 如表 7‑26所示，可以查看用户的详细信息。

表 7‑25 配送员详细信息

### 营销

这里管理员可以更改用于用户首页的轮播图照片，先将照片上传至后台，并将图片路径记录在图片表中。



表 7‑27 图片上传

如表3-36所示，后台通过BufferedOutputstrem接收文件，并放在项目的img文件夹中，前台form表单提交时需声明multipart/form-data。

表 3‑37 上传图片

|  |
| --- |
| BufferedOutputStream out = **new** BufferedOutputStream(  **New** FileOutputStream(**new** File("src/main/resources/static/img/"+file.getOriginalFilename())));  **try** {  String number =request.getParameter("number");  **int** id =Integer.*parseInt*(number);  String brief =request.getParameter("Brief");  String picturePath =file.getOriginalFilename();  Pic pic = **new** Pic();  pic.setId(id);  ….  picMapper.updateByPrimaryKey(pic);  out.write(file.getBytes());  out.flush();  out.close(); |

### 统计报表

#### 财务统计

财务统计中有今日收入和支出，右边是一个金额统计的报表，可以进行周年月的切换，底部是每月的支出收入详细。

表 7‑28 财务统计

#### 跑腿订单统计

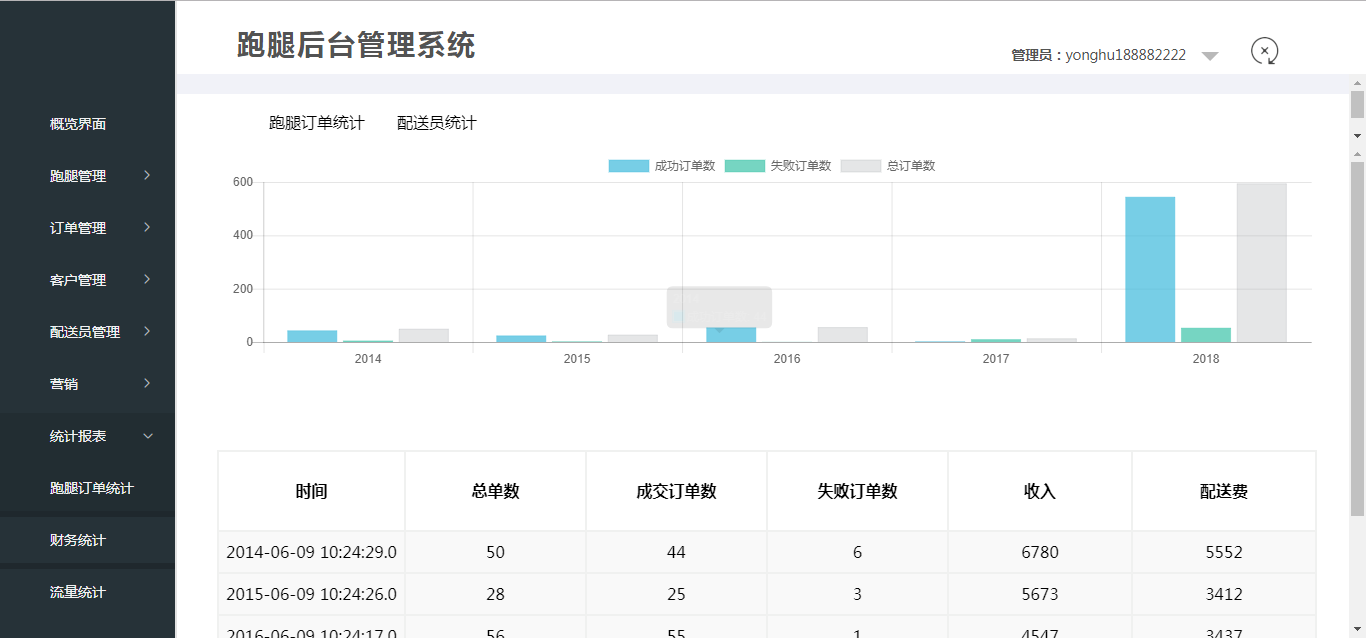
跑腿订单统计，顶部是每年的订单成交量，下面是最新的每日订单成交量。

表 7‑29 跑腿订单统计

#### 配送员统计

还有配送员统计，可以显示各个地区的配送员分布情况。

表 7‑30 配送员统计

#### 流量统计

流量统计中有平台访问量统计，配送员访问量统计，PV统计，UV统计，IV统计以及分享人数统计。



表 7‑31 流量统计

在统计报表中，所有的图表都运用到了如下图表控件，只需要从后台获取相关的ajax数据，便可以通过h5的画布来绘制图表。如表3-38所示。

表 3‑38 图表控件代码

|  |
| --- |
| **var** myChart = **new** Chart(ctx, {  type: 'bar', //柱状图改成bar  data: {  labels: [ "2014","2015","2016","2017","2018"],  datasets: [{  label: '成功订单数',  data: data1,  backgroundColor: [  'rgba(61, 185, 220, 0.7)',  ],  borderColor: [  'rgba(61, 185, 220, 0.4)',  ],  borderWidth: 1  }, |

# 系统测试

## 接口测试用例

### 被测试对象的介绍

测试对象为用户发布订单接口，订单查询接口，跑腿员接单接口

1. **用户发布订单接口:**

测试url：

int insert(@RequestBody Business business)

请求方式：

post

请求参数：

Business类：

表 8‑1 请求参数Business类

|  |  |  |  |
| --- | --- | --- | --- |
| **参数名** | **说明** | **是否必须** | **类型** |
| userId | 用户id | 是 | Bigint |
| type | 跑腿类型 | 是 | String |
| positionCreate | 起始地址 | 是 | String |
| positionFinish | 结束地址 | 是 | String |
| price | 价格 | 是 | Int |
| startTime | 开始时间 | 是 | Datetime |
| Time | 预计送达时间 | 是 | Datetime |
| Phone | 电话 | 是 | Datetime |
| Brief | 简介 | 是 | String |
| Status | 订单状态 | 是 | Int |
| name | 用户名称 | 否 | String |

返回参数：

表 8‑2 返回参数

|  |  |  |  |
| --- | --- | --- | --- |
| **参数名** | **说明** | **是否必须** | **类型** |
| Status | 状态码 | 1 | Int |

功能：

该函数用来给用户发布订单

请求实例：

http://localhost:8040/micro-provider-business/insert

返回实例：

{stauts:1}

状态码说明：

1表示发布订单成功，0表示发布失败。

1. **订单查询接口:**

测试方法：

ArrayList<Business> searchPojo(@RequestBody BusinessPojo businessPojo)

请求方式：

Post

请求参数：

表 8‑3 订单类

|  |  |  |  |
| --- | --- | --- | --- |
| **参数名** | **说明** | **是否必须** | **类型** |
| id | 订单号 | 否 | Bigint |
| Userid | 用户id | 否 | Bigint |
| Collectorid | 配送员id | 否 | Bigint |
| Type | 订单类型 | 否 | Int |
| Phone | 电话 | 否 | Int |
| Status | 订单状态 | 否 | Int |
| Time1 | 起止时间 | 否 | Datetime |
| Time2 | 起止时间 | 否 | Datetime |

返回参数：

表 8‑4 返回参数内容

|  |  |  |  |
| --- | --- | --- | --- |
| **参数名** | **说明** | **是否必须** | **类型** |
| userId | 用户id | 是 | Bigint |
| type | 跑腿类型 | 是 | String |
| positionCreate | 起始地址 | 是 | String |
| positionFinish | 结束地址 | 是 | String |
| price | 价格 | 是 | Int |
| startTime | 开始时间 | 是 | Datetime |
| Time | 预计送达时间 | 是 | Datetime |
| Phone | 电话 | 是 | Datetime |
| Brief | 简介 | 是 | String |
| Status | 订单状态 | 是 | Int |
| name | 用户名称 | 否 | String |
| Gmt\_create | 创建时间 | 是 | Datetime |
| Gmt\_modified | 修改时间 | 否 | Dateime |

功能**：**

该函数用来给用户查询跑腿订单

请求实例**：**

http://localhost:8040/micro-provider-business/business/searchpojo

返回实例**：**

{"id":10009,"userId":10001,"collectorId":null,"type":"帮买","positionCreate":"阿里巴巴西溪园区","positionFinish":" 杭州电子科技大学信息工程学院 ","price":12,"startTime":"2018-06-14 21:42:40.0","finishTime":null,"time":null,"phone":"18057936630","gmtCreate":"2018-06-14 21:42:39.0","gmtModified":null,"brief":"买一杯饮料","status":0,"city":"","name":"王泽华"},]

状态码说明**：**

1表示发布订单成功，0表示发布失败。

1. **跑腿员接单接口:**

测试方法：

int jiedan(@PathVariable(value="collectorid") Long collectorid,@PathVariable(value="id") Long id)

请求方式：

post

请求参数：

表 8‑5 business类

|  |  |  |  |
| --- | --- | --- | --- |
| **参数名** | **说明** | **是否必须** | **类型** |
| Collector\_id | 配送员id | 是 | Bigint |
| Business\_id | 订单id | 是 | Bigint |

返回参数：

表 8‑6 返回参数

|  |  |  |  |
| --- | --- | --- | --- |
| **参数名** | **说明** | **是否必须** | **类型** |
| Status | 返回状态码 | 是 | Int |
| Time | 时间 | 是 | Datetime |

功能**：**

该函数用来对给用户发布订单

请求实例**：**

http://localhost:8040/micro-provider-business/jiedan/100001/110001

返回实例**：**

{status：0，times：2018-12-22 19:45:35}

状态码说明**：**

1表示发布接单成功，0表示接单失败。

### 测试范围与目的

测试范围:在接口函数传参格式正确的情况下,观察函数返回值、数据库变化及设备状态的更新.

### 测试环境与测试辅助工具的描述

在测试用户发布订单接口时,数据库中已添加了相关数据,具体信息如下:

用户A(用户id:11001)

在测试订单查询接口时,存在了id为10009的相关订单

在测试跑腿员接单接口时,存在跑腿员id为10001的相关记录。

### 测试驱动程序的设计

向该接口函数传参,观察其返回结果、数据库中数据变化

### 接口测试用例

|  |  |  |  |
| --- | --- | --- | --- |
| 接口A的函数原型 | int insert(@RequestBody Business business) | | |
| 输入/动作 | | 期望的输出/响应 | 实际情况 |
| Business{10001,帮买，杭电信工，杭州西湖，20，2018-12-22 19：45，2018-12-22 20：22，13093761556，买一杯星巴克送到学校，0，沈先生  } | | 返回1,数据库中增加一条待取货的跑腿订单，订单id自增长 | 返回1,数据库中增加一条待取货的跑腿订单，订单id自增长 |
| Business{10002,帮买，杭电信工，杭州西湖，20，2018-12-22 19：45，2018-12-22 20：22，13093761556，买一杯星巴克送到学校，0，沈先生  } | | 返回0,因为数据库中不存在id为10002的用户 | 返回0,数据库中未添加新数据 |
| Business{10001,帮卖，杭电信工，杭州西湖，20，2018-12-22 19：45，2018-12-22 20：22，13093761556，买一杯星巴克送到学校，0，沈先生  } | | 返回0,因为数据库中跑腿类型不存在帮卖类型 | 返回1,数据库中增加了新数据 |
| Business{10001,帮卖，杭电信工， 20，2018-12-22 19：45，2018-12-22 20：22，13093761556，买一杯星巴克送到学校，0，沈先生  } | | 返回0,少了一个参数 | 返回0,数据库中未添加新数据 |
| Business{10001,帮卖，杭电信工， 杭州西湖，20，2018-12-22 19：45，2018-12-22 20：22，13093761556，…………￥￥，0，沈先生  } | | 返回0,存在乱码 | 返回1，数据中添加新数据，后台没有检测乱码的功能 |
| Business{10001,帮买，杭电信工，杭州西湖，20，2018-12-22 19：45，2018-12-22 20：22，13093761556，买一杯星巴克送到学校，5，沈先生  } | | 返回0,没有为5的状态 | 返回0,数据库中未添加新数据 |
| Business{10001,帮买，杭电信工，杭州西湖，20，2018-12-22 19：45，2018-12-22 20：22，13093761556213123123，买一杯星巴克送到学校，0，沈先生  } | | 返回0,手机号码应该为11位 | 返回1,数据库中添加新数据，后台没有判断手机号码位数的功能 |

|  |  |  |  |
| --- | --- | --- | --- |
| 接口B的函数原型 | ArrayList<Business> searchPojo(@RequestBody BusinessPojo businessPojo) | | |
| 输入/动作 | | 期望的输出/响应 | 实际情况 |
| Businesspojo{10009,10001,11001,帮买,0,1309346766,2018-12-20,2018-12-24,  } | | 返回business对象，内容为Business{10001,帮卖，杭电信工， 杭州西湖，20，2018-12-22 19：45，2018-12-22 20：22，13093761556，买一杯星巴克送到学校，0，沈先生  } | 返回business对象，内容为Business{10001,帮卖，杭电信工， 杭州西湖，20，2018-12-22 19：45，2018-12-22 20：22，13093761556，买一杯星巴克送到学校，0，沈先生  } |
| Businesspojo{10009,10001,11001,帮卖,0,1309346766,2018-12-20,2018-12-24,  } | | 返回0，没有内容 | 返回0,没有内容 |
| Businesspojo{business\_id：10009 } | | 返回Business对象{10001,帮买，杭电信工，杭州西湖，20，2018-12-22 19：45，2018-12-22 20：22，13093761556，买一杯星巴克送到学校，0，沈先生  } | 返回Business对象{10001,帮买，杭电信工，杭州西湖，20，2018-12-22 19：45，2018-12-22 20：22，13093761556，买一杯星巴克送到学校，0，沈先生  } |
| Businesspojo{10009,10001,11234,帮买,0,1309346766,2018-12-20,208-12-24,  } | | 返回0，没有内容 | 返回0,没有内容 |
| Businesspojo{collector\_id:10001} | | 返回配送员id为10001的所有订单 | 返回配送员id为10001的所有订单 |
| Businesspojo{user\_id:11001}  } | | 返回用户id为11001的所有订单 | 返回用户id为11001的所有订单 |
| Businesspojo{starttime:2018-12-20,finishtime:2018-12-24  } | | 返回2018-12-20至2018-12-24的所有订单 | 返回2018-12-20至2018-12-24的所有订单 |

|  |  |  |  |
| --- | --- | --- | --- |
| 接口C的函数原型 | int jiedan(@PathVariable(value="collectorid") Long collectorid,@PathVariable(value="id") Long | | |
| 输入/动作 | | 期望的输出/响应 | 实际情况 |
| Jiedan(10001,10009) | | 返回1,跑腿员接单成功 | 返回1,跑腿员接单成功 |
| Jiedan(10002,10009) | | 返回0,不存在此跑腿员 | 返回0，不存在此跑腿员 |
| Jiedan(10001,123456) | | 返回0,不存在此订单号 | 返回0,不存在此订单号 |

## 功能测试用例

### 被测试对象的介绍

课后习题功能：采用选择题的形式，在课程每节视频后设置课后习题。课程的查询功能。

### 测试范围与目的

测试用户完成课后习题后，显示正确答案与反馈答题情况的功能。

### 测试环境与测试辅助工具的描述

Java 1.8,eclipse

### 功能测试用例

|  |  |  |  |
| --- | --- | --- | --- |
| 用例ID | A001 | | |
| 功能A描述 | 用户发布订单 | | |
| 用例目的 | 检测系统在合法和不合法的输出用户发布订单的情况 | | |
| 前提条件 | 网络保持连通 | | |
| 输入/动作 | | 期望的输出/响应 | 实际情况 |
| 输入想买的商品 | | 正确显示 | 正确显示 |
| 输入店铺 | | 在输入时，有下拉提示 | 正确显示 |
| 更改小费 | | 自动计算跑腿价格，并变动 | 价格正确显示在页面上 |
| 更改重量 | | 自动计算跑腿价格，并变动 | 价格没有变动 |

|  |  |  |  |
| --- | --- | --- | --- |
| 用例ID | A002 | | |
| 功能B描述 | 用户个人中心 | | |
| 用例目的 | 检测系统能否正确响应用户个人中心的情况 | | |
| 前提条件 | 注册平台账号 | | |
| 输入/动作 | | 期望的输出/响应 | 实际情况 |
| 查询余额 | | 有正确的余额情况 | 正确显示 |
| 查询订单 | | 可以查看已购买的订单 | 正确显示 |
| 充值 | | 外接支付宝接口 | 支付宝未响应 |

|  |  |  |  |
| --- | --- | --- | --- |
| 用例ID | B001 | | |
| 功能B描述 | 跑腿员接受订单 | | |
| 用例目的 | 检测系统能否正确响应跑腿员的接单请求 | | |
| 前提条件 | 注册平台账号 | | |
| 输入/动作 | | 期望的输出/响应 | 实际情况 |
| 查看任务详情 | | 有明确的任务信息 | 正确显示 |
| 查看路径规划 | | 会有一个绿色的路径规划路线 | 正确显示 |
| 点击接单 | | 接单页面会消失该任务，并提示接单成功 | 能正确提示，但提示有滞后 |

|  |  |  |  |
| --- | --- | --- | --- |
| 用例ID | B002 | | |
| 功能B描述 | 跑腿员个人中心 | | |
| 用例目的 | 检测系统能否响应余额，订单查询请求。 | | |
| 前提条件 | 注册平台账号 | | |
| 输入/动作 | | 期望的输出/响应 | 实际情况 |
| 查询余额 | | 能查看到余额情况 | 正确显示 |
| 提现 | | 筛选输入的关键词匹配课程 | 正确显示 |
| 查询历史订单 | | 能查询到历史的订单列表 | 正确显示 |

|  |  |  |  |
| --- | --- | --- | --- |
| 用例ID | C001 | | |
| 功能B描述 | 管理员权限设计 | | |
| 用例目的 | 实现权限管理 | | |
| 前提条件 | 登录账号为管理员 | | |
| 输入/动作 | | 期望的输出/响应 | 实际情况 |
| 设置角色 | | 能够自定义角色 | 操作成功 |
| 设置角色对应权限 | | 角色能访问相应权限 | 正确响应 |
| 分配用户角色 | | 用户能访问所赋予的权限 | 父级菜单能成功响应，子级不能 |

|  |  |  |  |
| --- | --- | --- | --- |
| 用例ID | C002 | | |
| 功能B描述 | 管理员订单管理 | | |
| 用例目的 | 实现对用户的管理 | | |
| 前提条件 | 登录账号为管理员 | | |
| 输入/动作 | | 期望的输出/响应 | 实际情况 |
| 按条件查找订单 | | 输出相应条件订单 | 在针对时间查找时，只会显示后一天的相关订单。 |
| 订单导出 | | 筛选输入的关键词匹配课程 | 正确导出 |
| 查看订单详情 | | 一门课归属一小类 | Css样式不正确 |

|  |  |  |  |
| --- | --- | --- | --- |
| 用例ID | C003 | | |
| 功能B描述 | 管理员用户管理 | | |
| 用例目的 | 实现对用户的管理 | | |
| 前提条件 | 登录账号为管理员 | | |
| 输入/动作 | | 期望的输出/响应 | 实际情况 |
| 按条件查找用户 | | 相应条件筛选后的用户 | 只会显示后一天的相关列表 |
| 拉入用户黑名单 | | 用户无法使用账号 | 成功拉入黑名单，但用户依旧能够使用 |

|  |  |  |  |
| --- | --- | --- | --- |
| 用例ID | C004 | | |
| 功能B描述 | 管理员跑腿员管理 | | |
| 用例目的 | 实现对跑腿员的管理 | | |
| 前提条件 | 登录账号为管理员 | | |
| 输入/动作 | | 期望的输出/响应 | 实际情况 |
| 按条件查找用户 | | 相应条件筛选后的跑腿员 | 只会显示后一天的相关列表 |
| 拉入跑腿员黑名单 | | 跑腿员无法使用账号 | 跑腿员无法使用账号，显示拉黑 |

# 结语

在本次毕业设计中，我选择了“跑腿配送平台”作为了我期末大作业的题目，我不仅充分运用了在课程中所学的知识，还通过网上查找相关技术资料，自学了很多知识。在技术方面，在Spring boot的基础上，选用了Spring Cloud作为微服务框架，数据库上选用了Mysql，持久化框架选用了Mybaits。在经过四周的不懈努力之后，顺利完成了本次毕业设计。

跑腿配送服务就成了当前互联网和分享经济下一个非常好的“切入口”。而该服务在基于“顺路带货”的众包、分享经济，轻资产的运作的方式，和传统物流相比大大降低了成本，还能够带动就业，实现多方共赢。本系统主要有三大模块，跑腿员端，用于跑腿员进行线上接单。客户端，用于在线下单以及查看订单。后台管理系统，用于后台管理员对整个系统的订单监控和修改。在项目初期，我针对当今市面上已有的跑腿配送平台进行了调研，并通过网上查找资料，确定了项目的核心需求，在确立了需求之后，我以敏捷开发的模式将这些需求放在看板上，由核心需求衍生出一个个的小需求。同时，根据一个个的小需求组成模块。从而达到了将整个项目分布式，模块化的目的。做完这些前期准备工作之后，便根据需求对数据库进行设计，遵循数据库第二范式的原则，总共建立了11张表。因为项目对于分布式的需求，我便选择了Spring Cloud作为项目的技术开发架构，只需要将各个组件通过eureka进行注册，组件之间便可以互相访问。最后，在编码阶段，我首先编写好api文档，并根据api文档在后台写出一个个基于restful风格的api，在通过测试之后，再编写一个个前端页面，通过ajax技术进行信息交互。

在这次课程设计中，我不仅负责了数据的设计，软件架构的设计，后台的开发，还负责前端页面的设计和美化，让我充分体验到了当一个全栈工程师的感觉，之前我往往只注重于后台的开发，没有学习过前台的页面是怎么设计，怎么制作的，但我这次不仅自学了MUI框架，还完整的设计出一套前端页面，十分有成就感！同时，我也发现在这次毕业设计中，我并没有使用到很多关于算法的知识，这说明我这方面还是有些欠缺的，希望以后能够补足。最后，我想说我十分感谢学校有这么一门课，让我学会了很多！

# 致谢

在本科毕业设计的这几个月的时间中，从对选题的理解，需求的分析，数据库的设计，功能的分析，再到论文的编写，期间老师和同学给予了我巨大的帮助和建议。

感谢唐红军老师给予了我巨大的支持和帮助，唐红军老师在大学四年中在竞赛方面从课题的选题，到需求的分析，都给了我很多帮助。在毕业设计中也指点了我很多。

感谢毕业论文指导老师金洁洁的建议和指导，老师在功能方面提了很多建议，让我有所改进。

感谢沈益豪同学在我遇到困难时给予我的关心和支持，他的加油和打气让我度过了一个又一个的难关。

感谢辅导员老师刘宁在疫情期间为我提供的帮助和便利。

感谢母校和老师们在大学四年中对我的用心培养。

# 参考文献

# 附录

## 表汇总

|  |  |
| --- | --- |
| 表 名 | 功能说明 |
| **公司员工表** | 保存公司员工信息 |
| **地址表** | 保存用户地址信息 |
| **广告营销表** | 保存广告营销信息 |
| **操作权限表** | 保存操作权限信息 |
| **月度订单统计表** | 保存月度订单信息 |
| **流量统计** | 保存浏览量信息 |
| **用户表** | 保存用户信息 |
| **用户财务表** | 保存用户财务信息 |
| **群组表** | 保存群组信息 |
| **角色表** | 保存角色信息 |
| **计费规则表** | 保存计费规则信息 |
| **订单评价表** | 保存订单评价信息 |
| **访问权限表** | 保存访问权限信息 |
| **财务月度统计表** | 用于记录财务月度的各项数据，营业额，支出等及其时间 |
| **跑腿类型表** | 用于跑腿名称和类型 |
| **跑腿订单表** | 保存订单的用户，配送员，完成状态，地点类型等具体信息 |
| **配送员表** | 保存配送员的身份，交通，记录等信息 |
| **配送员财务表** | 用于记录配送员跑腿的金额及跑腿收入 |
| **支付表** | 用于订单的支付 |

## 表明细

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 表 名 | **公司员工表** | | | | |
| 功能说明 | 保存公司员工信息 | | | | |
| 中文列名 | 字段名 | 数据类型  （精度范围） | 空否 | 约束条件 | 含义说明 |
| 标识号 | id | bigint(8) | 否 | 主键 | 自增 |
| 管理员名称 | employee\_name | varchar(20) | 否 |  |  |
| 密码 | password | varchar(8) | 否 |  |  |
| 电话 | phone | varchar(20) | 否 |  |  |
| 邮箱 | email | varchar(20) | 否 |  |  |
| 角色 | role | int(8) | 否 |  |  |
| 创建时间 | gmt\_create | datetime | 否 |  |  |
| 修改时间 | gmt\_modified | datetime | 否 |  |  |
| 补充说明 |  | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 表 名 | **角色** | | | | |
| 功能说明 | 保存角色信息 | | | | |
| 中文列名 | 字段名 | 数据类型  （精度范围） | 空否 | 约束条件 | 含义说明 |
| 标识号 | id | bigint(8) | 否 | 主键 | 自增长 |
| 用户id | role\_name | varchar（20） | 否 |  |  |
| 消费金额 | role\_power | int(4) | 否 |  |  |
| 记录创建 | gmt\_create | datetime | 否 |  |  |
| 修改时间 | gmt\_modified | datetime | 否 |  |  |
| 补充说明 |  | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 表 名 | **访问权限表** | | | | |
| 功能说明 | 保存访问权限信息 | | | | |
| 中文列名 | 字段名 | 数据类型  （精度范围） | 空否 | 约束条件 | 含义说明 |
| 标识符 | id | bigint(8) | 否 | 主键 | 自增 |
| 菜单名称 | module\_name | varchar(40) | 否 |  |  |
| 上级菜单 | parent | varchar(40) | 否 |  | 父菜单 |
| 层级 | floor | varchar(40) | 否 |  | 3级菜单等 |
| 对应链接 | url | varchar(40) | 否 |  |  |
| 创建时间 | gmt\_create | datetime | 否 |  |  |
| 修改时间 | gmt\_modified | datetime | 否 |  |  |
| 补充说明 |  | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 表 名 | **操作权限表** | | | | |
| 功能说明 | 保存操作权限信息 | | | | |
| 中文列名 | 字段名 | 数据类型  （精度范围） | 空否 | 约束条件 | 含义说明 |
| 角色标识号 | id | bigint(8) | 否 | 联合主键、外键 | 自增 角色表的外键 |
| 模块标识号 | mod\_id | bigint(8) | 否 | 联合主键、外键 | 自增 访问权限表的外键 |
| 操作权限 | right | int(1) | 否 |  | 增加删除修改 采用累加形式 |
| 创建时间 | gmt\_create | datetime | 否 |  |  |
| 修改时间 | gmt\_modified | datetime | 否 |  |  |
| 补充说明 |  | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 表 名 | **广告营销表** | | | | |
| 功能说明 | 保存广告营销信息 | | | | |
| 中文列名 | 字段名 | 数据类型  （精度范围） | 空否 | 约束条件 | 含义说明 |
| 标识号 | id | bigint(8) | 否 | 主键 | 自增 |
| 图片路径 | picturepath | varchar(255) | 否 |  |  |
| 简介 | brief | varchar(255) | 否 |  |  |
| 创建时间 | gmt\_create | datetime | 否 |  |  |
| 修改时间 | gmt\_modified | datetime | 否 |  |  |
| 补充说明 |  | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 表 名 | **用户表** | | | | |
| 功能说明 | 保存用户信息 | | | | |
| 中文列名 | 字段名 | 数据类型  （精度范围） | 空否 | 约束条件 | 含义说明 |
| 用户id | user\_id | bigint(8) | 否 | 主键 | 自增长 |
| 用户名字 | user\_name | varchar（20） | 否 |  |  |
| 密码 | password | varchar（8） | 否 |  |  |
| 手机号 | phone | varchar（20） | 否 |  |  |
| 电子信箱 | email | varchar（20） | 否 |  |  |
| 记录创建 | gmt\_create | datetime | 否 |  |  |
| 修改时间 | gmt\_modified | datetime | 否 |  |  |
| 补充说明 |  | | | | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 表 名 | **地址表** | | | | | |
| 功能说明 | **保存用户地址信息** | | | | | |
| 中文列名 | | 字段名 | 数据类型  （精度范围） | 空否 | 约束条件 | 含义说明 |
| 标识号 | | id | bigint(8) | 否 | 主键 | 自增 |
| 用户标识号 | | user\_id | bigint(8) | 否 | 外键 | 自增 用户表的外键 |
| 城市 | | city | varchar(40) | 否 |  |  |
| 省份 | | province | varchar(40) | 否 |  |  |
| 详细地址 | | address | varchar(255) | 否 |  |  |
| 电话 | | phone | varchar(40) | 否 |  |  |
| 创建时间 | | gmt\_create | datetime | 否 |  |  |
| 修改时间 | | gmt\_modified | datetime | 否 |  |  |
| 补充说明 | |  | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 表 名 | **用户财务表** | | | | |
| 功能说明 | 保存用户财务信息 | | | | |
| 中文列名 | 字段名 | 数据类型  （精度范围） | 空否 | 约束条件 | 含义说明 |
| 标识号 | id | bigint(8) | 否 | 主键 | 自增长 |
| 用户id | user\_id | bigint(8) | 否 | 外键 | 用户表外键 |
| 消费金额 | price | decimal(4,1) | 否 |  |  |
| 余额 | balance | decimal(4,1) | 否 |  |  |
| 积分 | score | int(4) | 否 |  |  |
| 消费类型 | int(4) | int(1) | 否 |  |  |
| 记录创建 | gmt\_create | datetime | 否 |  |  |
| 修改时间 | gmt\_modified | datetime | 否 |  |  |
| 补充说明 |  | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 表 名 | **跑腿类型表** | | | | |
| 功能说明 | 用于跑腿名称和类型 | | | | |
| 中文列名 | 字段名 | 数据类型  （精度范围） | 空否 | 约束条件 | 含义说明 |
| 标识号 | id | bigint(8) | 否 | 主键 | 自增长 |
| 跑腿名称 | type\_name | Varchar(20) | 否 |  |  |
| 跑腿规则 | rule | int(8) | 否 |  |  |
| 图片路径 | picture\_path | Varchar(40) | 否 |  |  |
| 列表序号 | rank | int(2) | 否 |  |  |
| 计费规则 | payrule | int(2) | 否 | 外键 | 管理计费规则表中计费规则 |
| 简介 | brief | text | 否 |  |  |
| gmt\_create | gmt\_create | datetime | 否 |  |  |
| gmt\_modified | gmt\_modified | datetime | 否 |  |  |
| 补充说明 |  | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 表 名 | **计费规则表** | | | | |
| 功能说明 | 保存计费规则信息 | | | | |
| 中文列名 | 字段名 | 数据类型  （精度范围） | 空否 | 约束条件 | 含义说明 |
| 标识符 | id | bigint(8) | 否 | 主键 | 自增 |
| 起始距离 | start | int(8) | 否 |  |  |
| 结束距离 | finish | int(8) | 否 |  |  |
| 金额 | money | int(8) | 否 |  |  |
| 类型 | type | varchar(255) | 否 |  |  |
| 创建时间 | gmt\_create | datetime | 否 |  |  |
| 修改时间 | gmt\_modified | datetime | 否 |  |  |
| 补充说明 |  | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 表 名 | **跑腿订单表** | | | | |
| 功能说明 | 保存订单的用户，配送员，完成状态，地点类型等具体信息 | | | | |
| 中文列名 | 字段名 | 数据类型  （精度范围） | 空否 | 约束条件 | 含义说明 |
| 标识号 | id | bigint(8) | 否 | 主键 | 关联群组id，自增长 |
| 用户ID | use\_id | bigint(8) | 否 | 外键 | 关联用户表中id |
| 配送员ID | collector\_id | bigint(8) | 否 | 外键 | 关联用户表中password |
| 跑腿类型 | type | bigint(8) | 否 | 外键 | 关联用户表中phone |
| 跑腿内容 | brief | longtext | 否 |  |  |
| 订单金额 | price | Int(8) | 否 |  |  |
| 订单状态 | status | Int(4) | 否 |  |  |
| 跑腿起点 | start\_position | Varchar(40) | 否 |  |  |
| 跑腿终点 | finish\_position | Varchar(40) | 否 |  |  |
| 订单支付类型 | phone | Varchar(20) | 否 |  |  |
| 订单开始时间 | start\_time | datetime | 否 |  |  |
| 订单完成时间 | finish\_time | datetime | 否 |  |  |
| 订单关闭时间 | close\_time | datetime | 否 |  |  |
| 订单所在区域 | city | Varchar(255) | 否 |  |  |
| 订单创建时间 | gmt\_create | datetime | 否 |  | 记录修改时间 |
| 订单更改时间 | gmt\_modified | datetime | 否 |  | 记录创建时间 |
| 补充说明 |  | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 表 名 | **订单评价表** | | | | |
| 功能说明 | 保存订单评价信息 | | | | |
| 中文列名 | 字段名 | 数据类型  （精度范围） | 空否 | 约束条件 | 含义说明 |
| 标识符 | id | bigint(8) | 否 | 主键 | 自增 |
| 用户标识符 | user\_id | bigint(8) | 否 | 外键 | 自增 用户表的外键 |
| 跑腿订单标识符 | business\_id | bigint(8) | 否 | 外键 | 自增 跑腿订单表的外键 |
| 配送员标识符 | collector\_id | bigint(8) | 否 | 外键 | 自增 配送员表的外键 |
| 评价内容 | evaluate\_text | varchar(255) | 否 |  |  |
| 星级 | evaluate | decimal(4,1) | 否 |  |  |
| 回复 | repeat | varchar(255) | 否 |  |  |
| 创建时间 | gmt\_create | datetime | 否 |  |  |
| 修改时间 | gmt\_modified | datetime | 否 |  |  |
| 补充说明 |  | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 表 名 | **支付表** | | | | |
| 功能说明 | 用于订单支付 | | | | |
| 中文列名 | 字段名 | 数据类型  （精度范围） | 空否 | 约束条件 | 含义说明 |
| 标识号 | id | bigint(8) | 否 | 主键,外键 | 主键 |
| 订单标识号 | Businessid | bigint(8) | 否 |  | 关联订单表外键 |
| 支付状态 | Status | Demical(4,1 | 否 |  | 未支付，已支付，支付失败 |
| 支付金额 | money | Demical(4,1 | 否 |  |  |
| 支付方式 | Way | Int | 否 |  |  |
| 平台分成 | Divide | Varchar(20) | 否 |  |  |
| 跑腿员获利 | Gain | Demical(4,1 | 否 |  |  |
| Gmt\_modified | gmt\_modified | datetime | 否 |  | 记录修改时间 |
| Gmt\_create | gmt\_create | datetime | 否 |  | 记录创建时间 |
| 补充说明 |  | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 表 名 | **配送员表** | | | | |
| 功能说明 | 保存配送员的身份，交通，记录等信息 | | | | |
| 中文列名 | 字段名 | 数据类型  （精度范围） | 空否 | 约束条件 | 含义说明 |
| 标识号 | id | bigint(8) | 否 | 主键 |  |
| 群组ID | group\_id | bigint(8) | 否 |  | 外键，关联群组id |
| 密码 | type | Varchar(8) | 否 |  | 提现，跑腿收入 |
| 电话 | money | Varchar(20) | 否 |  |  |
| 配送员名称 | collector\_name | Varchar(20) | 否 |  |  |
| 邮箱 | Email | Varchar(20) | 否 |  |  |
| 工作状态 | is\_work | Varchar(8) | 否 |  |  |
| 当前位置 | position | Varchar(40) | 否 |  |  |
| 累计评价 | evaluate | Decimal(4,1) | 否 |  |  |
| 身份证号 | id\_card | Varchar(20) | 否 |  |  |
| 身份证照片路径 | id\_pic | Varchar(255) | 否 |  |  |
| Gmt\_modified | gmt\_modified | datetime | 否 |  | 记录修改时间 |
| Gmt\_create | gmt\_create | datetime | 否 |  | 记录创建时间 |
| 交通方式 | transport | Varchar(8) | 否 |  |  |
| 补充说明 |  | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 表 名 | **群组** | | | | |
| 功能说明 | 保存群组信息 | | | | |
| 中文列名 | 字段名 | 数据类型  （精度范围） | 空否 | 约束条件 | 含义说明 |
| 标识号 | id | bigint(8) | 否 | 主键 | 自增长 |
| 所属地区 | position | varchar（0） | 否 |  |  |
| 所属主管 | Column\_6 | <undefined> | 否 |  |  |
| 所属跑腿类型 | number | Int(8) | 否 |  |  |
| 记录创建 | gmt\_create | datetime | 否 |  |  |
| 修改时间 | gmt\_modified | datetime | 否 |  |  |
| 补充说明 |  | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 表 名 | **配送员财务表** | | | | |
| 功能说明 | 用于记录配送员跑腿的金额及跑腿收入 | | | | |
| 中文列名 | 字段名 | 数据类型  （精度范围） | 空否 | 约束条件 | 含义说明 |
| 标识号 | Id | bigint(8) | 否 | 主键 | 自增长 |
| 配送员ID | collector\_id | bigint(8) | 否 | 外键 | 关联配送员表的配送员id |
| 类型 | type | int(40) | 否 |  | 提现，跑腿收入 |
| 累计金额 | money | decimal(40) | 否 |  |  |
| 补充说明 |  | | | | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 表 名 | 流量统计 | | | | | |
| 功能说明 | 保存浏览量信息 | | | | | |
| 中文列名 | | 字段名 | 数据类型  （精度范围） | 空否 | 约束条件 | 含义说明 |
| 标识号 | | id | int(8) | 否 | 主键 | 自增长 |
| 页面浏览量 | | pv | int(8) | 否 |  |  |
| 独立访客 | | uv | int(8) | 否 |  |  |
| 独立IP数 | | ip | int(8) | 否 |  |  |
| 分享数量 | | share | int(8) | 否 |  |  |
| 客户浏览量 | | customer\_pv | int(8) | 否 |  |  |
| 收集浏览量 | | collector\_pv | int(8) | 否 |  |  |
| 平均浏览量 | | avg\_pv | int(8) | 否 |  |  |
| 记录创建 | | gmt\_create | timestamp | 否 |  |  |
| 修改时间 | | gmt\_modified | timestamp | 否 |  |  |
| 补充说明 | |  | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 表 名 | **财务月度统计表** | | | | |
| 功能说明 | 用于记录财务月度的各项数据，营业额，支出等及其时间 | | | | |
| 中文列名 | 字段名 | 数据类型  （精度范围） | 空否 | 约束条件 | 含义说明 |
| 标识号 | id | bigint(8) | 否 | 主键 | 自增长 |
| 净利润 | income | Decimal(4,1) | 否 |  |  |
| 营业额 | total | Decimal(4,1) | 否 |  |  |
| 支出 | output | Decimal(4,1) | 否 |  |  |
| 所属年份 | year | Year | 否 |  |  |
| 所属月份 | month | int(0) | 否 |  |  |
| gmt\_create | gmt\_create | datetime | 否 |  |  |
| gmt\_modified | gmt\_modified | datetime | 否 |  |  |
| 补充说明 |  | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 表 名 | **月度订单统计表** | | | | |
| 功能说明 | 保存月度订单信息 | | | | |
| 中文列名 | 字段名 | 数据类型  （精度范围） | 空否 | 约束条件 | 含义说明 |
| 标识号 | id | bight(8) | 否 | 主键 | 自增 |
| 成功订单数 | success | int(8) | 否 |  |  |
| 失败订单数 | fail | int(8) | 否 |  |  |
| 收入 | income | int(8) | 否 |  |  |
| 支出 | expend | int(8) | 否 |  |  |
| 总订单数 | business | int(8) | 否 |  |  |
| 所属月份 | month | int(2) | 否 |  |  |
| 所属年份 | year | year | 否 |  |  |
| 创建时间 | gmt\_create | datetime | 否 |  |  |
| 修改时间 | gmt\_modified | datetime | 否 |  |  |
| 补充说明 |  | | | | |