**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 |

**目 录**

[1 系统需求分析 1](#_Toc516869500)

[1.1 用户需求分析 1](#_Toc516869501)

[1.1.1 客户 1](#_Toc516869502)

[1.1.2 跑腿员 1](#_Toc516869503)

[1.1.3 后台管理员 2](#_Toc516869504)

[1.2 功能需求分析 2](#_Toc516869505)

[1.2.1 跑腿客户端 2](#_Toc516869506)

[1.2.2 配送员端 2](#_Toc516869507)

[1.2.3 后台管理系统 3](#_Toc516869508)

[1.3 数据流分析 3](#_Toc516869509)

[2 系统设计 4](#_Toc516869510)

[2.1 系统架构以及工作原理 4](#_Toc516869511)

[2.1.1 系统整体架构 4](#_Toc516869512)

[2.1.2 系统技术架构 5](#_Toc516869513)

[2.1.3 各功能模块基本架构 7](#_Toc516869514)

[2.2 系统总体设计 9](#_Toc516869515)

[2.2.1 系统流程设计 9](#_Toc516869516)

[2.2.2 系统用例分析 10](#_Toc516869517)

[2.2.3 系统数据模型设计 13](#_Toc516869518)

[2.2.4 系统特色 19](#_Toc516869519)

[2.2.5 技术选型 19](#_Toc516869520)

[3 系统实现 20](#_Toc516869521)

[3.1 微服务注册发现组件—EurekaService 20](#_Toc516869522)

[3.2 微服务网关—ZuulProxy 21](#_Toc516869523)

[3.3 微服务的REST调用和熔断机制 22](#_Toc516869524)

[3.3.1 使用Feign实现声明式REST调用 22](#_Toc516869525)

[3.3.2 整合Feign和Hystrix 22](#_Toc516869526)

[3.4 跑腿订单微服务 23](#_Toc516869527)

[3.4.1 订单的查询 23](#_Toc516869528)

[3.4.2 订单状态的变动 24](#_Toc516869529)

[3.4.3 订单的评价 24](#_Toc516869530)

[3.4.4 订单的添加 25](#_Toc516869531)

[3.4.5 订单的统计 25](#_Toc516869532)

[3.4.6 订单的导出 26](#_Toc516869533)

[3.4.7 订单的计费规则 27](#_Toc516869534)

[3.4.8 流量统计 28](#_Toc516869535)

[3.5 跑腿员微服务 29](#_Toc516869536)

[3.5.1 跑腿员接单 29](#_Toc516869537)

[3.5.2 跑腿员任务 30](#_Toc516869538)

[3.5.3 跑腿员个人信息 31](#_Toc516869539)

[3.6 用户微服务 31](#_Toc516869540)

[3.6.1 用户下单 31](#_Toc516869541)

[3.6.2 用户个人信息 32](#_Toc516869542)

[3.6.3 用户历史订单 32](#_Toc516869543)

[3.7 跑腿员功能实现（HTML5） 33](#_Toc516869544)

[3.7.1 接单 33](#_Toc516869545)

[3.7.2 我的任务 35](#_Toc516869546)

[3.7.3 个人中心 36](#_Toc516869547)

[3.8 客户端功能实现(HTML5) 37](#_Toc516869548)

[3.8.1 首页 37](#_Toc516869549)

[3.8.2 帮买 38](#_Toc516869550)

[3.8.3 我的地址&我的订单 40](#_Toc516869551)

[3.8.4 我的评价&我的钱包 41](#_Toc516869552)

[3.9 后台管理系统功能实现 42](#_Toc516869553)

[3.9.1 概览界面 42](#_Toc516869554)

[3.9.2 计费规则 43](#_Toc516869555)

[3.9.3 订单管理 44](#_Toc516869556)

[3.9.4 客户管理 46](#_Toc516869557)

[3.9.5 配送员管理 47](#_Toc516869558)

[3.9.6 营销 48](#_Toc516869559)

[3.9.7 统计报表 49](#_Toc516869560)

[4 结语 52](#_Toc516869561)

# 1 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

#### 可以有多种跑腿服务

用户期望，可以有多种跑腿任务，比如帮买，帮送，帮取货等。

#### 轻量性的使用

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。

#### 跑腿价格透明

有一个透明的跑腿价格体系。

#### 账单

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

#### 接单

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。

#### 任务列表

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。

### 后台管理员

#### 对跑腿订单的管理

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。

#### 对用户，跑腿员的管理

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。

#### 统计报表

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

图 1‑1 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1-1所示。

# 可行性研究

## 技术可行性研究

## 经济可行性研究

## 操作可行性研究

# 系统总体设计

## 系统架构以及工作原理

### 系统整体架构

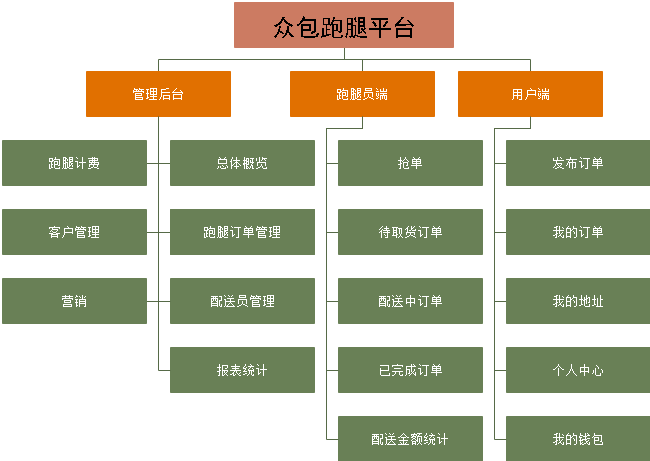


图 2‑1 系统整体架构

由图2-1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

#### 系统拓扑图

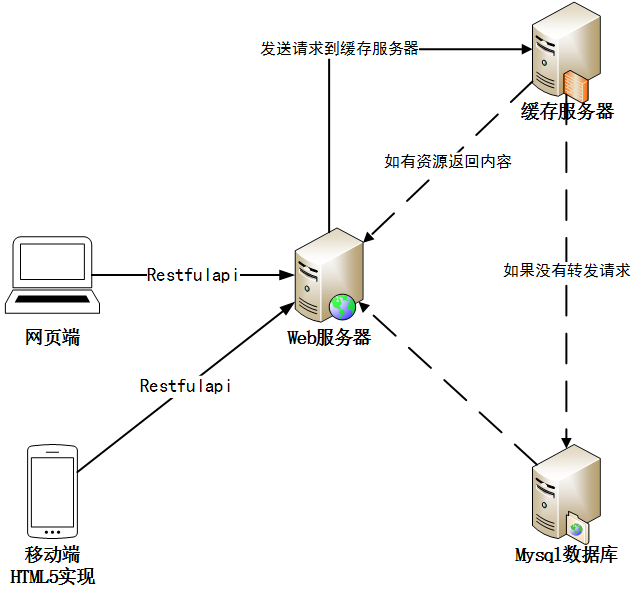
****

图 2‑2 系统拓扑图

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2-2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。

#### 基于Spring Cloud的微服务架构

图 2‑3 项目微服务架构

本系统分为表现层，接口访问层，业务服务层，数据访问层。

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

#### 客户端（HTML5）

图 2‑4 客户端整体架构

#### 跑腿员端（HTML5）

图 2‑5 跑腿员端基本架构

#### 跑腿后台管理系统

图 2‑6 后台管理系统整体架构

## 系统总体设计

### 系统流程设计

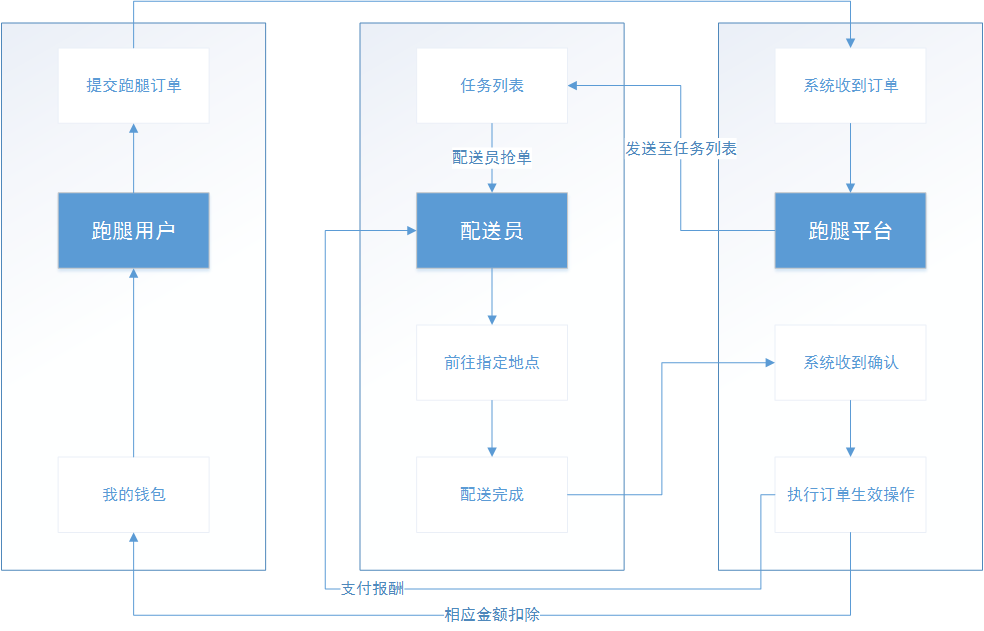


图 2‑7 总体流程图

系统整体流程如图2-7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，当跑腿员看到跑腿订单并抢单之后，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，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

#### 配送员用例分析

图 2‑8 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。

#### 跑腿用户用例分析

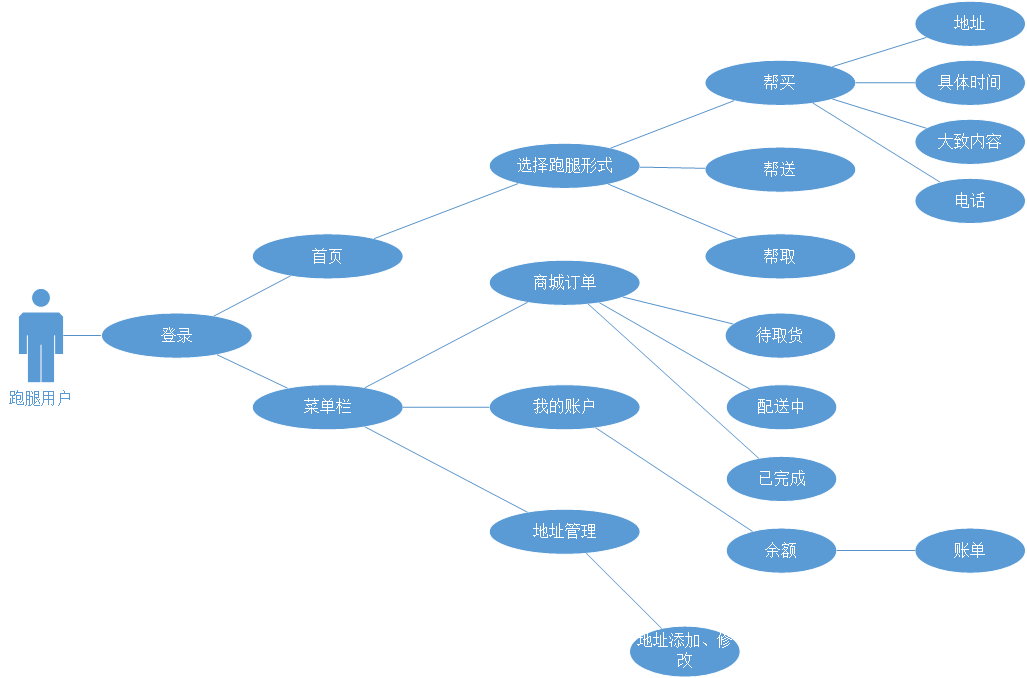
用户关心的是简单易用，用户在登录后进入主页，主页面有一个轮播图介绍我们的主要产品，下面三个就是我们的核心功能，跑腿业务，一共有三种功能，帮买，帮送，帮取。用户点击后需要输入相关信息，如：“我要买一杯咖啡”，“杭州东站”，“武林广场”，“130937987122”，“王女士”，并选择是否要小费，选择完以后下方会出现价格，点击提交后便会出现在配送员的任务列表中。当有配送员抢单后，订单便会自动转入我的订单中，显示待取货，配送完成后便会显示已完成。还有我的钱包，个人中心，用户可以在这里查看相关的个人信息。如图2-9所示。

图 2‑9 跑腿用户用例图

#### 后台管理员用例分析

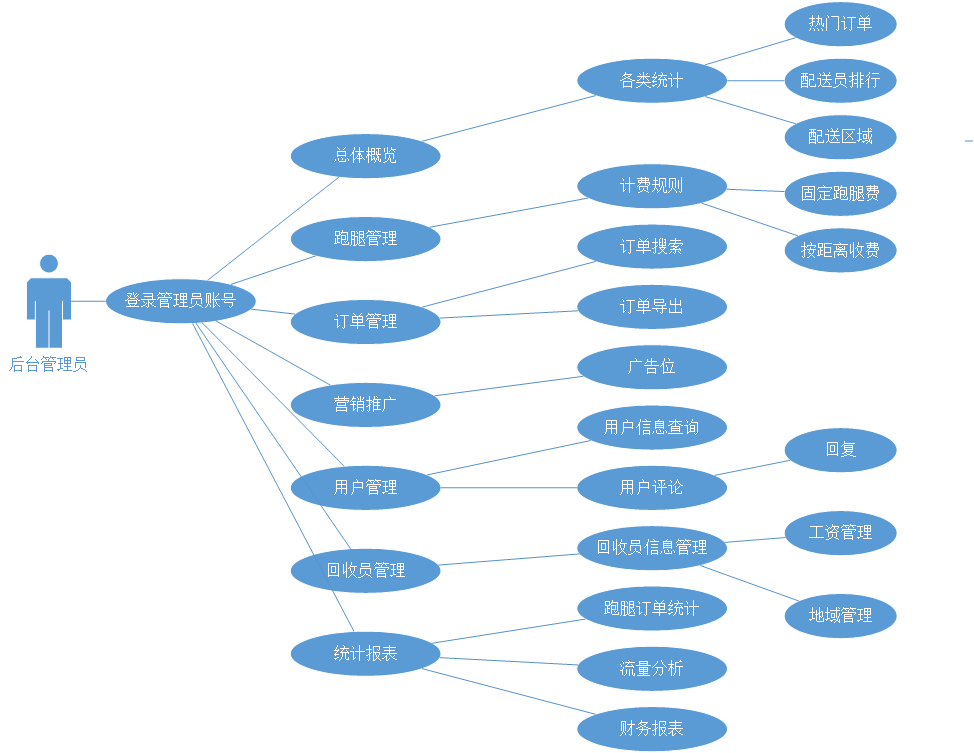


图 2‑10 后台管理员用例图

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。

### 系统数据模型设计

#### ER图设计

图 2‑11 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2-11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。

#### 数据库表结构

用户表User，如表2-1所示

表 2‑1 用户表

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名 | 类型 | 说明 | 备注 |
| UserCode | Int(8) | 用户编号 | 主键，不能为空，自增加 |
| UserName | Varchar(20) | 用户名 | 不能为空 |
| PassWord | Varchar(8) | 密码 | 不能为空 |
| Phone | Int(16) | 电话 | 不能为空 |
| Email | Varchar(20) | 邮箱 | 可以为空 |
| Create\_time | timestamp | 创建时间 | 创建时插入 |
| Money | Double(8) | 余额 | 初始0 |
| Card | varchar（20） | 提款账户 |  |

跑腿员表，如表2-2所示

表 2‑2 跑腿员表

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名 | 类型 | 说明 | 备注 |
| CollectorCode | Int(8) | 配送员ID | 主键，不能为空，自增加 |
| CollectorName | Varchar(20) | 用户名 | 不能为空 |
| PassWord | Varchar(8) | 密码 | 不能为空 |
| Phone | Int(16) | 电话 | 不能为空 |
| Email | Varchar(20) | 邮箱 | 可以为空 |
| Create\_time | timestamp | 创建时间 |  |
| State | Varchar(8) | 状态（上班，下班） | 默认值为下班 |
| Pay | Int | 赏金 |  |
| Evaluate | Double | 评价 | 满分为5.0 |
| Idcard | Varchar(14) | 身份证号 |  |
| Total | Int(6) | 配送总数 | 默认值为0 |
| Success | Int(6) | 配送成功 |  |
| Fail | Int(6) | 配送失败 |  |

群组表，如表2-3所示

表 2‑3 群组表

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名 | 类型 | 说明 | 备注 |
| GroupID | Int(8) | 群组id | 主键，不能为空，自增加 |
| Position | Varchar | 所属分区 | 不能为空 |
| Number | Int | 回收员数 |  |

员工表，如表2-4所示

表 2‑4 员工表

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名 | 类型 | 说明 | 备注 |
| EmployeeCode | Int(8) | 编号 | 主键，自增加 |
| EmployeeName | Varchar(20) | 管理员名 | 不能为空 |
| PassWord | Varchar(8) | 密码 | 不能为空 |
| Phone | Int(16) | 电话 | 不能为空 |
| Email | Varchar(20) | 邮箱 | 可以为空 |
| Role | Int | 员工角色 | Role表的外键 |

跑腿订单表，如表2-5所示。

表 2‑5 跑腿订单表

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名 | 类型 | 说明 | 备注 |
| BusinessCode | Int(8) | 编号 | 主键，自增加 |
| UserCode | Int | 用户编号 | 用户表的外键 |
| CollectorCode | Int(8) | 跑腿员 | 跑腿员表的外键 |
| Brief | Varchar(60) | 配送内容 | 不能为空，如：帮我买香烟等 |
| GoodsType | Int(2) | 配送类型 | 如：帮我买，帮我送、帮我排队等 |
| Postion\_create | Varchar(255) | 起始地 | 从哪里开始 |
| Postion\_finish | Varchar(255) | 目的地 | 送到哪里 |
| Create\_time | timestamp | 订单创建时间 | 可以为空 |
| Start\_time | timestamp | 订单接单时间 | 可以为空 |
| Finish\_time | timestamp | 订单结束时间 | 可以为空 |
| Money | Int | 配送金额 |  |
| Phone | Varchar | 联系方式 |  |

订单评价表，如表2-6所示

表 2‑6 订单评价表

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名 | 类型 | 说明 | 备注 |
| EvaluateCode | Int(8) | 评价编号 | 主键 |
| GoodsCode | Int(8) | 订单编号 | 订单表的外键 |
| CollectorCode | Int(8) | 跑腿人编号 | 跑腿员表的外键 |
| UserCode | Int | 用户编号 |  |
| Evaluate | Double | 订单评价 | 最高为5.0 |
| Time | Timestamp | 评论时间 |  |

计费规则表，如表2-7所示

表 2‑7 计费规则表

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名 | 类型 | 说明 | 备注 |
| ruleCode | Int(8) | 编号 | 主键，自增加 |
| ruleName | Varchar(20) | 规则名 | 不能为空 |
| Brief | Varchar(60) | 规则简介 | 不能为空 |
| Money | Int | 金额 | 应该使用字符串匹配算法实现 |

### 系统特色

#### 分布式开发-微服务架构

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。

#### 特色的报表统计

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

表2-8 技术配置

|  |  |
| --- | --- |
| 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

本系统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

图 3‑1 Eureka 运行截图

图 3‑2 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。

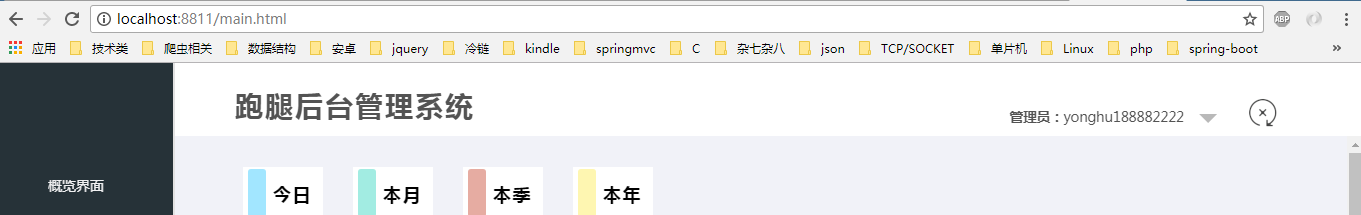
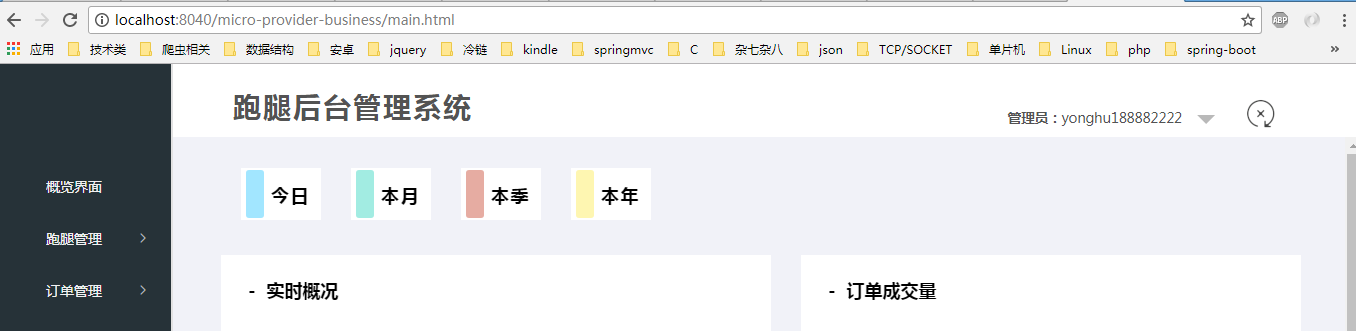
在启动zuul之后，可以看到项目已经通过了api网关，被转发到localhost:8040/micro-provider-Business 下面。如图3-3所示。

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

图 3‑4 配送端主页

图 3‑5 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

图 3‑6 配送中

图 3‑7 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

图 3‑8 我的余额

图 3‑9 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

图 3‑10 菜单

图 3‑11 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

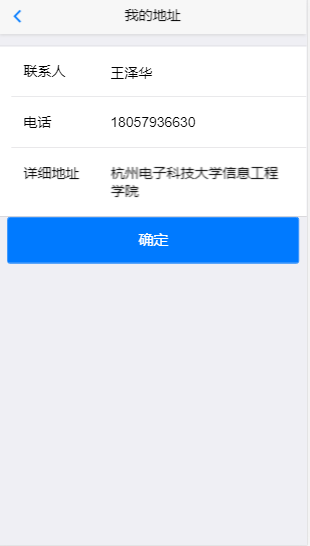


图 3‑12 帮我买

图 3‑13 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

图 3‑14 地址列表

图 3‑15 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

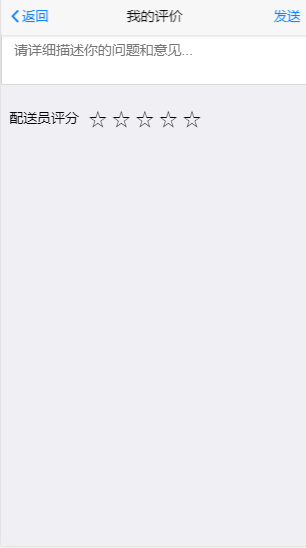


图 3‑16 我的余额

图 3‑17 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。

图 3‑18 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。

图 3‑19 固定跑腿费



图 3‑20 按距离收费

### 订单管理

如图 3‑21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

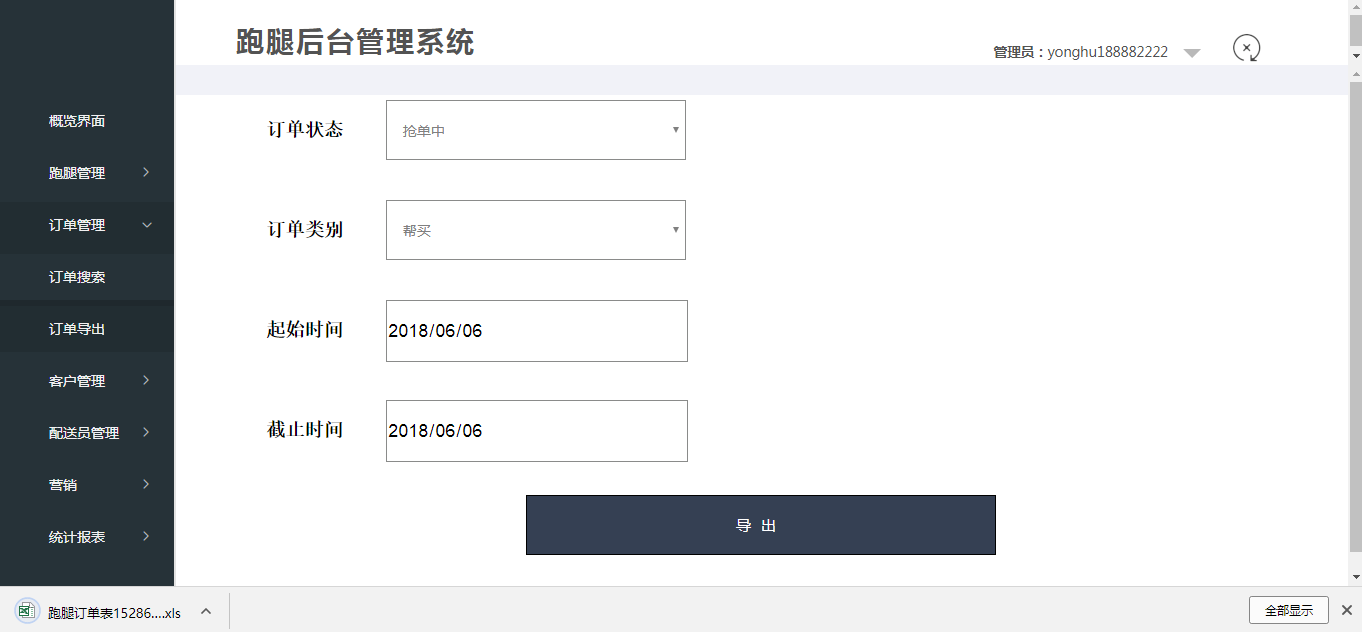
在订单管理中，还有一个订单导出功能，可以选择相应的订单状态和订单类别，然后通过表单提交的形式将信息发送至后台，后台会通过响应流将文件发送至前台。

图 3‑23 订单导出

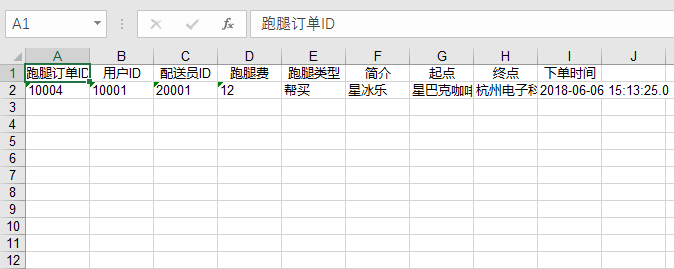
管理员在输入订单状态，订单类别，起止时间之后，便会获得相应的导出excel文件。通过form表单发送信息。

图 3‑24 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。

图 3‑25 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。

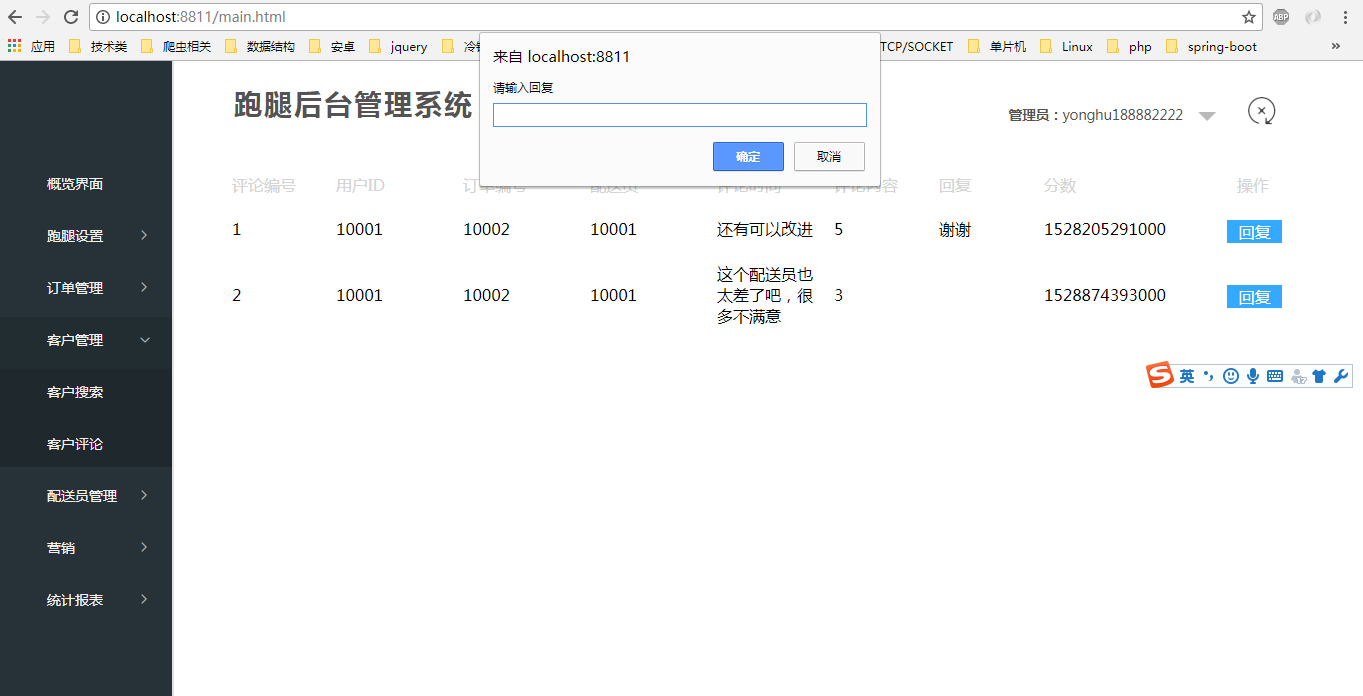


图 3‑26 客户评论

### 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，

图 3‑27 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。

图 3‑28 添加配送员

 如图 3‑30所示，可以查看用户的详细信息。

图 3‑29 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。



图 3‑31 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。

图 3‑32 财务统计

#### 跑腿订单统计

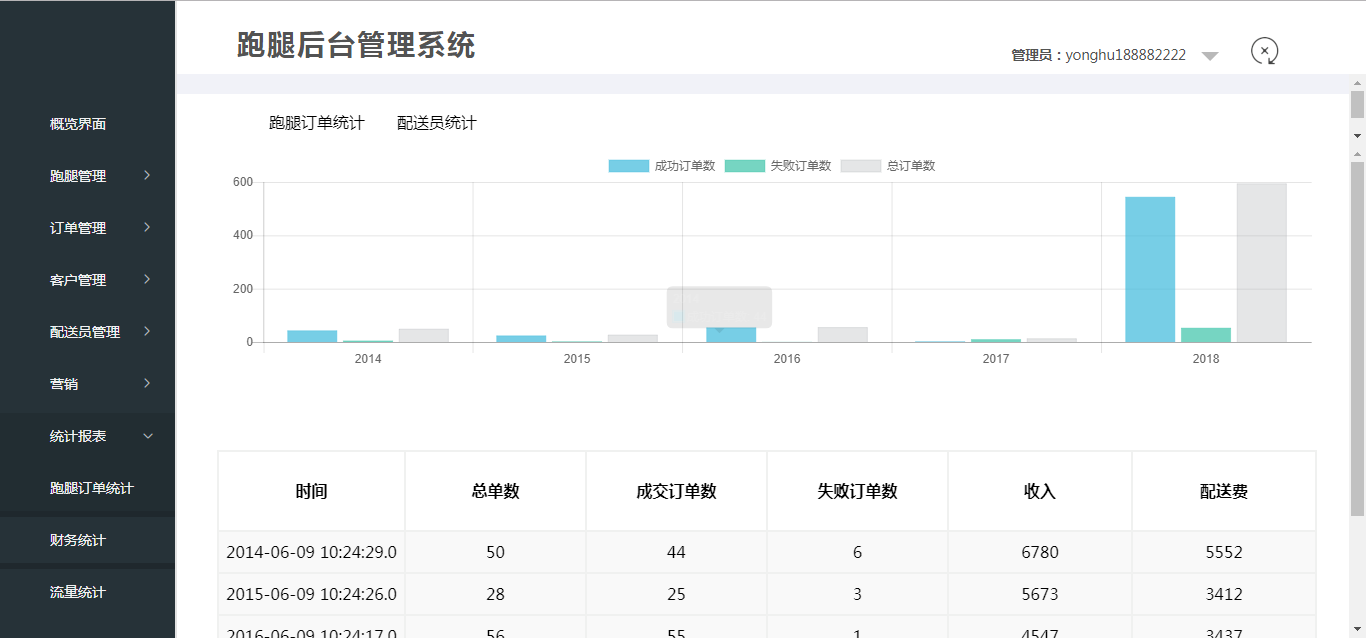
跑腿订单统计，顶部是每年的订单成交量，下面是最新的每日订单成交量。

图 3‑33 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。

图 3‑34 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。



图 3‑35 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  }, |

# 系统测试和验收

# 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，顺利完成了本次期末课程设计。

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。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，我也发现在这次课程设计中，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！

# 参考文献

# 致谢